



C3D

Руководство разработчика

2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	10
M.1. МЕТОДЫ ПОСТРОЕНИЯ ТЕЛ.....	14
M.2. ВЫПОЛНЕНИЕ ОПЕРАЦИЙ НАД ТЕЛАМИ.....	88
M.3. МЕТОДЫ ПОСТРОЕНИЯ ДВУМЕРНЫХ КРИВЫХ.....	179
M.4. МЕТОДЫ ПОСТРОЕНИЯ КРИВЫХ.....	194
M.5. МЕТОДЫ ПОСТРОЕНИЯ ПОВЕРХНОСТЕЙ.....	215
M.6. МЕТОДЫ ПРЯМОГО МОДЕЛИРОВАНИЯ.....	237
M.7. МЕТОДЫ ПОСТРОЕНИЯ ТЕЛ ИЗ ЛИСТОВОГО МЕТАЛЛА.....	252
M.8. ВСПОМОГАТЕЛЬНЫЕ МЕТОДЫ.....	283
O.1. ЭЛЕМЕНТАРНЫЕ ОБЪЕКТЫ.....	290
O.2. ГЕОМЕТРИЧЕСКИЕ ОБЪЕКТЫ.....	297
O.3. КРИВЫЕ ДВУМЕРНОГО ПРОСТРАНСТВА.....	303
O.4. КРИВЫЕ.....	318
O.5. ПОВЕРХНОСТИ.....	344
O.6. СПЕЦИАЛЬНЫЕ ОБЪЕКТЫ.....	379
O.7. ТОПОЛОГИЧЕСКИЕ ОБЪЕКТЫ.....	389
O.8. ОБЪЕКТЫ ГЕОМЕТРИЧЕСКОЙ МОДЕЛИ.....	401
R.1. ПОСТРОЕНИЕ ТРИАНГУЛЯЦИИ.....	414
R.2. ПОСТРОЕНИЕ ПЛОСКИХ ПРОЕКЦИЙ.....	422
R.3. ВЫЧИСЛЕНИЕ ИНЕРЦИОННЫХ ХАРАКТЕРИСТИК.....	427
R.4. ОПРЕДЕЛЕНИЕ СТОЛКНОВЕНИЙ ТЕЛ.....	433
S.1. ДВУМЕРНЫЙ ГЕОМЕТРИЧЕСКИЙ РЕШАТЕЛЬ.....	438
S.2. ДВУМЕРНЫЕ РАЗМЕРЫ.....	451
S.3. ДВУМЕРНЫЕ ЛОГИЧЕСКИЕ ОГРАНИЧЕНИЯ.....	458
S.4. ВЫЧИСЛЕНИЕ И ДИАГНОСТИКА ДВУМЕРНЫХ ОГРАНИЧЕНИЙ.....	465
S.5. ДВУМЕРНЫЕ СПЛАЙНЫ И ПАРАМЕТРИЧЕСКИЕ КРИВЫЕ.....	470
T.1. ОБМЕН ДАННЫМИ С ДРУГИМИ СИСТЕМАМИ.....	472
T.2. КОНВЕРТЕРЫ ГРАНИЧНОГО ПРЕДСТАВЛЕНИЯ.....	481
T.3. КОНВЕРТЕРЫ ПОЛИГОНАЛЬНОГО ПРЕДСТАВЛЕНИЯ.....	486

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	10
Общие сведения.....	10
Структура и отличительные особенности.....	10
Выполняемые функции.....	10
Теоретические основы.....	11
Комплектация.....	12
Тестовое приложение.....	12
Разработка в среде .NET.....	13
М.1. МЕТОДЫ ПОСТРОЕНИЯ ТЕЛ	14
М.1.1. Построение элементарного тела.....	14
М.1.2. Построение элементарного тела по заданной поверхности.....	18
М.1.3. Построение тела выдавливания.....	20
М.1.4. Построение тела вращения.....	34
М.1.5. Построение тела заметания.....	46
М.1.6. Построение тела по плоским сечениям.....	58
М.1.7. Создание тела по заданному множеству граней.....	66
М.1.8. Построение незамкнутого тела на базе поверхности.....	67
М.1.9. Построение линейчатого тела.....	68
М.1.10. Построение тела по сети кривых.....	69
М.1.11. Построение тела сопряжения несвязанных граней.....	73
М.1.12. Построение заплатки.....	76
М.1.13. Сшивка граней тел.....	79
М.1.14. Построение тела по кривым или точкам кривых.....	80
М.1.15. Построение эквидистантного тела.....	82
М.1.16. Продление грани тела.....	84
М.2. ВЫПОЛНЕНИЕ ОПЕРАЦИЙ НАД ТЕЛАМИ	88
М.2.1. Булева операция над телами.....	88
М.2.2. Булева операция над незамкнутыми телами.....	93
М.2.3. Булева операция с телом выдавливания.....	97
М.2.4. Булева операция с телом вращения.....	102
М.2.5. Булева операция с телом заметания.....	109
М.2.6. Булева операция с телом, построенным по плоским сечениям.....	112
М.2.7. Резка тела поверхностью.....	115
М.2.8. Резка тела плоским контуром.....	118
М.2.9. Построение симметричного тела.....	121
М.2.10. Скругление рёбер тела.....	123
М.2.11. Скругление рёбер тела переменным радиусом.....	137
М.2.12. Построение тела с фасками рёбер.....	142
М.2.13. Построение тонкостенного тела.....	150
М.2.14. Построение тонкостенного тела с различной толщиной стенки.....	153
М.2.15. Построение тела приданием толщины поверхности.....	155
М.2.16. Построение зеркального тела.....	156
М.2.17. Булева операция тела и множества тел.....	158
М.2.18. Объединение множества тел.....	163
М.2.19. Разделить тело на несвязанные части.....	165
М.2.20. Отделение несвязанной части тела.....	166
М.2.21. Разбиение граней тела.....	168
М.2.22. Построение отверстия, кармана или паза в теле.....	169
М.2.23. Построение тела с ребром жёсткости.....	173
М.2.24. Уклонение граней тела.....	175
М.2.25. Размножение тела.....	177
М.3. МЕТОДЫ ПОСТРОЕНИЯ ДВУМЕРНЫХ КРИВЫХ	179
М.3.1. Построение двумерной прямой линии и отрезка.....	179
М.3.2. Построение двумерной окружности, эллипса и их дуг.....	180
М.3.3. Построение двумерных кривых по контрольным точкам.....	181

М.3.4. Построение двумерной NURBS кривой.....	184
М.3.5. Построение выпуклой равносторонней двумерной ломаной.....	187
М.3.6. Построение двумерной косинусоиды.....	189
М.3.7. Построение двумерной составной кривой.....	190
М.3.8. Построение кривых пересечения поверхности и плоскости.....	190
М.3.9. Построение двумерной кривой ребра грани.....	191
М.3.10. Построение проекции кривой на поверхность.....	192
М.4. МЕТОДЫ ПОСТРОЕНИЯ КРИВЫХ.....	194
М.4.1. Построение прямой линии и отрезка.....	194
М.4.2. Построение окружности, эллипса и их дуг.....	195
М.4.3. Построение кривых по контрольным точкам.....	196
М.4.4. Построение NURBS кривой.....	199
М.4.5. Построение выпуклой равносторонней ломаной.....	202
М.4.6. Построение спиралей.....	202
М.4.7. Построение составной кривой.....	205
М.4.8. Построение каркаса.....	206
М.4.9. Построение проекции кривой на поверхность.....	207
М.4.10. Построение кривых пересечения поверхностей.....	209
М.4.11. Построение силуэтных кривых.....	210
М.4.12. Построение кривой сопряжения кривых.....	212
М.5. МЕТОДЫ ПОСТРОЕНИЯ ПОВЕРХНОСТЕЙ.....	215
М.5.1. Построение элементарной поверхности.....	215
М.5.2. Построение NURBS поверхности.....	218
М.5.3. Построение поверхности выдавливания.....	222
М.5.4. Построение поверхности вращения.....	223
М.5.5. Построение поверхностей заметания.....	224
М.5.6. Построение поверхности по семейству кривых.....	226
М.5.7. Построение линейчатых поверхностей.....	229
М.5.8. Построение поверхности по трем кривым.....	230
М.5.9. Построение поверхности по четырем кривым.....	231
М.5.10. Построение поверхности по сети кривых.....	232
М.5.11. Построение эквидистантной поверхности.....	233
М.5.12. Построение поверхности с произвольными границами.....	235
М.6. МЕТОДЫ ПРЯМОГО МОДЕЛИРОВАНИЯ.....	237
М.6.1. Построение трансформированного тела.....	237
М.6.2. Построение модифицированного тела.....	239
М.6.3. Построение деформируемого тела.....	244
М.6.4. Построение деформируемой призмы.....	249
М.6.5. Построение сглаженной поверхности.....	250
М.7. МЕТОДЫ ПОСТРОЕНИЯ ТЕЛ ИЗ ЛИСТОВОГО МЕТАЛЛА.....	252
М.7.1. Построение листового тела.....	252
М.7.2. Построение обечайки.....	253
М.7.3. Построение сгиба листового тела по линии.....	255
М.7.4. Построение подсечки листового тела.....	256
М.7.5. Сгиб разогнутого листового тела.....	257
М.7.6. Разгиб сгибов листового тела.....	259
М.7.7. Добавление пластины к листовому телу.....	260
М.7.8. Построение выреза в листовом теле.....	261
М.7.9. Построение сгиба листового тела на ребрах.....	262
М.7.10. Построение сгиба по эскизу.....	264
М.7.11. Замыкание угла листового тела.....	265
М.7.12. Построение штампованного тела.....	267
М.7.13. Построение буртика листового тела.....	269
М.7.14. Построение жалюзи листового тела.....	271
М.7.15. Восстановление рёбер сгибов листового тела.....	273
М.7.16. Разделение сгибов листового тела на группы.....	274
М.7.17. Разделение сгибов листового тела на пары.....	275

М.7.18. Проверка грани на возможность быть фиксированной.....	276
М.7.19. Поиск граней для кривой.....	276
М.7.20. Поиск листовой грани тела.....	276
М.7.21. Поиск парной грани сгиба листового тела.....	277
М.7.22. Поиск плоской грани листового тела.....	278
М.7.23. Поиск парной грани листового тела.....	278
М.7.24. Определение расстояния между подобными гранями.....	280
М.7.25. Поиск подобных сгибов.....	281
М.7.26. Поиск касательной точки сгиба листового тела.....	282
М.7.27. Поиск осевой линии сгиба.....	282
М.8. ВСПОМОГАТЕЛЬНЫЕ МЕТОДЫ.....	283
М.8.1. Вычисление глубины тела выдавливания или угла тела вращения.....	283
М.8.2. Определение образа кривой для выдавливания или вращения.....	283
М.8.3. Определение параметров для выдавливания или вращения.....	284
М.8.4. Определение ориентации образующей кривой.....	284
М.8.5. Определение ориентации секущей поверхности.....	285
М.8.6. Ориентация кривых тела заметания.....	285
М.8.7. Построение копии направляющей кривой тела заметания.....	286
М.8.8. Построение ребра жёсткости.....	286
М.8.9. Проверка кривой для построения линейчатого тела.....	286
М.8.10. Проверка параметров кривой для построения линейчатого тела.....	287
М.8.11. Проверка кривой для построения тела соединения.....	287
М.8.12. Проверка параметров кривой для построения тела соединения.....	287
М.8.13. Построение кривой по множеству рёбер.....	288
О.1. ЭЛЕМЕНТАРНЫЕ ОБЪЕКТЫ.....	290
О.1.1. Вектор в трёхмерном пространстве MbVector3D.....	290
О.1.2. Радиус-вектор точки в трёхмерном пространстве MbCartPoint3D.....	290
О.1.3. Расширенный вектор в трёхмерном пространстве MbHomogenius3D.....	290
О.1.4. Локальная система координат MbPlacement3D.....	291
О.1.5. Расширенная матрица в трёхмерном пространстве MbMatrix3D.....	292
О.1.6. Габаритный параллелепипед в трёхмерном пространстве MbCube.....	292
О.1.7. Одномерный габарит MbRect1D.....	293
О.1.8. Вектор в двумерном пространстве MbVector.....	293
О.1.9. Нормализованный вектор в двумерном пространстве MbDirection.....	294
О.1.10. Радиус-вектор точки в двумерном пространстве MbCartPoint.....	294
О.1.11. Расширенный вектор в двумерном пространстве MbHomogenius.....	294
О.1.12. Локальная система координат MbPlacement.....	295
О.1.13. Расширенная матрица в двумерном пространстве MbMatrix.....	295
О.1.14. Габаритный прямоугольник в двухмерном пространстве MbRect.....	296
О.2. ГЕОМЕТРИЧЕСКИЕ ОБЪЕКТЫ.....	297
О.2.1. Счётчик ссылок MbRefItem.....	297
О.2.2. Трёхмерный геометрический объект MbSpaceItem.....	298
О.2.3. Топологический объект MbTopItem.....	299
О.2.4. Двумерный геометрический объект MbPlaneItem.....	301
О.3. КРИВЫЕ ДВУМЕРНОГО ПРОСТРАНСТВА.....	303
О.3.1. Двумерная кривая MbCurve.....	303
О.3.2. Двумерная прямая MbLine.....	305
О.3.3. Двумерный отрезок прямой MbLineSegment.....	305
О.3.4. Двумерная дуга эллипса MbArc.....	305
О.3.5. Двумерная ломаная линия MbPolyline.....	306
О.3.6. Двумерная NURBS-кривая MbNurbs.....	307
О.3.7. Двумерная кривая Эрмита MbHermit.....	308
О.3.8. Двумерная составная кривая Безье MbBezier.....	309
О.3.9. Двумерная кубический сплайн MbCubicSpline.....	310
О.3.10. Двумерная усечённая кривая MbTrimmedCurve.....	311
О.3.11. Двумерная репараметризованная кривая MbReparamCurve.....	312
О.3.12. Двумерная эквидистантная кривая MbOffsetCurve.....	313

O.3.13. Двумерная символьная кривая MbCharCurve.....	313
O.3.14. Двумерная косинусоида MbCosinusoid.....	314
O.3.15. Двумерная кривая-точка MbPointCurve.....	315
O.3.16. Двумерная проекционная кривая MbProjCurve.....	315
O.3.17. Двумерный контур MbContour.....	316
O.4. КРИВЫЕ.....	318
O.4.1. Кривая MbCurve3D.....	318
O.4.2. Прямая линия MbLine3D.....	320
O.4.3. Отрезок прямой MbLineSegment3D.....	320
O.4.4. Дуга эллипса MbArc3D.....	320
O.4.5. Ломаная линия MbPolyline3D.....	321
O.4.6. NURBS-кривая MbNurbs3D.....	322
O.4.7. Кривая Эрмита MbHermit3D.....	323
O.4.8. Составная кривая Безье MbBezier3D.....	324
O.4.9. Кубический сплайн MbCubicSpline3D.....	325
O.4.10. Усечённая кривая MbTrimmedCurve3D.....	326
O.4.11. Репараметризованная кривая MbReparamCurve3D.....	327
O.4.12. Эквидистантная кривая MbOffsetCurve3D.....	328
O.4.13. Символьная кривая MbCharacterCurve3D.....	329
O.4.14. Коническая спираль MbConeSpiral.....	329
O.4.15. Спираль переменного радиуса MbCurveSpiral.....	330
O.4.16. Спираль с криволинейной плоской осью MbCrookedSpiral.....	331
O.4.17. Соединительная кривая MbBridgeCurve3D.....	332
O.4.18. Контур MbContour3D.....	333
O.4.19. Плоская кривая MbPlaneCurve.....	334
O.4.20. Кривая на поверхности MbSurfaceCurve.....	335
O.4.21. Силуэтная кривая MbSilhouetteCurve.....	336
O.4.22. Контур на поверхности MbContourOnSurface.....	337
O.4.23. Контур на плоскости MbContourOnPlane.....	338
O.4.24. Кривая пересечения поверхностей MbSurfaceIntersectionCurve.....	339
O.5. ПОВЕРХНОСТИ.....	344
O.5.1. Поверхность MbSurface.....	344
O.5.2. Плоскость MbPlane.....	346
O.5.3. Цилиндрическая поверхность MbCylinderSurface.....	347
O.5.4. Коническая поверхность MbConeSurface.....	348
O.5.5. Сферическая поверхность MbSphereSurface.....	349
O.5.6. Поверхность тора MbTorusSurface.....	350
O.5.7. Поверхность выдавливания MbExtrusionSurface.....	351
O.5.8. Поверхность вращения MbRevolutionSurface.....	352
O.5.9. Поверхность перемещения MbExpansionSurface.....	353
O.5.10. Спиральная поверхность MbSpiralSurface.....	355
O.5.11. Кинематическая поверхность MbEvolutionSurface.....	356
O.5.12. Кинематическая поверхность с адаптацией MbExactionSurface.....	357
O.5.13. Секториальная поверхность MbSectorSurface.....	358
O.5.14. Линейчатая поверхность MbRuledSurface.....	359
O.5.15. Поверхность на семействе кривых MbLoftedSurface.....	360
O.5.16. Поверхность на семействе кривых и направляющей MbElevationSurface.....	361
O.5.17. Поверхность на трёх кривых MbCornerSurface.....	362
O.5.18. Поверхность Кунса MbCoverSurface.....	363
O.5.19. Поверхность Кунса MbCoonsPatchSurface.....	365
O.5.20. Поверхность на сети кривых MbMeshSurface.....	366
O.5.21. Поверхность соединения MbJoinSurface.....	367
O.5.22. NURBS-поверхность MbSplineSurface.....	368
O.5.23. Эквидистантная поверхность MbOffsetSurface.....	370
O.5.24. Поверхность фаски MbChamferSurface.....	372
O.5.25. Поверхность скругления MbFilletSurface.....	372
O.5.26. Поверхность скругления MbChannelSurface.....	375

O.5.27. Поверхность с произвольными границами MbCurveBoundedSurface.....	376
O.6. СПЕЦИАЛЬНЫЕ ОБЪЕКТЫ.....	379
O.6.1. Функция MbFunction.....	379
O.6.2. Константная функция MbConstFunction.....	380
O.6.3. Линейная функция MbLineFunction.....	380
O.6.4. Кубическая функция Эрмита MbCubicFunction.....	381
O.6.5. Кубическая сплайн-функция MbCubicSplineFunction.....	381
O.6.6. Символьная функция MdCharacterFunction.....	382
O.6.7. Мультилиния MbMultiline.....	383
O.6.8. Двумерный контур с разрывами MbContourWithBreaks.....	383
O.6.9. Регион MbRegion.....	384
O.6.10. Вспомогательный геометрический объект MbLegend.....	385
O.6.11. Маркер MbMarker.....	386
O.6.12. Условное обозначение резьбы MbThread.....	386
O.6.13. Условное обозначение MbPointsSymbol.....	387
O.6.14. Обозначение шероховатости MbRough.....	387
O.6.15. Условное обозначение линии выноски MbLeader.....	388
O.7. ТОПОЛОГИЧЕСКИЕ ОБЪЕКТЫ.....	389
O.7.1. Топологический объект MbTopologyItem.....	389
O.7.2. Грань MbFace.....	390
O.7.3. Ребро MbEdge.....	391
O.7.4. Вершина MbVertex.....	391
O.7.5. Ребро грани MbCurveEdge.....	392
O.7.6. Цикл грани MbLoop.....	394
O.7.7. Ориентированное ребро грани MbOrientedEdge.....	395
O.7.8. Множество граней MbFaceShell.....	396
O.7.9. Копирование множества граней MbFaceShell.....	398
O.7.10. Именованное множество граней, рёбер и вершин.....	399
O.8. ОБЪЕКТЫ ГЕОМЕТРИЧЕСКОЙ МОДЕЛИ.....	401
O.8.1. Объект геометрической модели MbItem.....	401
O.8.2. Твёрдое тело MbSolid.....	402
O.8.3. Проволочный каркас MbWireFrame.....	406
O.8.4. Точечный каркас MbPointFrame.....	408
O.8.5. Полигональный объект MbMesh.....	408
O.8.6. Вставка MbInstance.....	410
O.8.7. Сборочная единица MbAssembly.....	410
O.8.8. Вставка трёхмерного объекта MbSpaceInstance.....	411
O.8.9. Вставка двумерных объектов MbPlaneInstance.....	412
O.8.10. Вспомогательный объект MbAssistingItem.....	412
R.1. ПОСТРОЕНИЕ ТРИАНГУЛЯЦИИ.....	414
R.1.1. Управление вычислением триангуляции.....	414
R.1.2. Построение полигонального объекта.....	415
R.1.3. Добавление полигонального объекта.....	417
R.1.4. Построение полигонов объекта.....	417
R.1.5. Построение триангуляции грани.....	419
R.1.6. Построение триангуляции тела.....	420
R.1.7. Построение полигональных объектов множества тел.....	421
R.2. ПОСТРОЕНИЕ ПЛОСКИХ ПРОЕКЦИЙ.....	422
R.2.1. Данные построения плоских проекций.....	422
R.2.2. Построение плоской проекции модели.....	423
R.2.3. Построение полигональной проекции тел.....	424
R.2.4. Построение линий очерка триангуляции.....	425
R.3. ВЫЧИСЛЕНИЕ ИНЕРЦИОННЫХ ХАРАКТЕРИСТИК.....	427
R.3.1. Инерционные характеристики модели.....	427
R.3.2. Инерционные характеристики тела.....	428
R.3.3. Инерционные характеристики множества тел.....	429
R.3.4. Инерционные характеристики модели.....	430

R.3.5. Вычисление площади поверхности.....	430
R.3.6. Вычисление объема тела.....	432
R.4. ОПРЕДЕЛЕНИЕ СТОЛКНОВЕНИЙ ТЕЛ.....	433
R.4.1. Определение пересечения двух тел.....	433
R.4.2. Определение столкновений в наборе тел.....	433
R.4.3. Запросы на поиск столкновений.....	435
R.4.4. Настройка запроса на поиск столкновений.....	436
R.4.5. Группировка тел из контрольного набора.....	437
S.1. ДВУМЕРНЫЙ ГЕОМЕТРИЧЕСКИЙ РЕШАТЕЛЬ.....	438
S.1.1. Назначение геометрического решателя GCE.....	438
S.1.2. Встраивание в приложение.....	438
S.1.3. Типы поддерживаемой геометрии.....	439
S.1.4. Типы геометрических ограничений и размеров.....	440
S.1.5. Базовые типы данных API решателя GCE.....	441
S.1.6. Система геометрических ограничений.....	442
S.1.7. Представление геометрических объектов.....	442
S.1.8. Степень свободы.....	444
S.1.9. Добавление и удаление геометрических объектов.....	444
S.1.10. Фиксация и заморозка геометрических объектов.....	446
S.1.11. Контрольные точки геометрических объектов.....	446
S.1.12. Скалярная переменная.....	447
S.1.13. Линейное уравнение.....	448
S.1.14. Журналирование вызовов API.....	448
S.2. ДВУМЕРНЫЕ РАЗМЕРЫ.....	451
S.2.1. Вспомогательные точки линейного размера.....	451
S.2.2. Управляющие и вариационные размеры.....	452
S.2.3. Нулевые и знакопеременные размеры.....	452
S.2.4. Линейный размер (расстояние).....	453
S.2.5. Направленный линейный размер.....	454
S.2.6. Расстояние от точки до отрезка.....	455
S.2.7. Угловые размеры.....	455
S.2.8. Угловой размер по трем или четырем точкам.....	456
S.2.9. Радиальные и диаметральные размеры.....	457
S.3. ДВУМЕРНЫЕ ЛОГИЧЕСКИЕ ОГРАНИЧЕНИЯ.....	458
S.3.1. Совпадение точки и другого объекта.....	458
S.3.2. Выравнивание точек.....	458
S.3.3. Параллельность/перпендикулярность.....	458
S.3.4. Коллинеарность.....	458
S.3.5. Равенство длин и радиусов.....	459
S.3.6. Унарные ограничения: горизонтальность/вертикальность и варианты фиксации.....	459
S.3.7. Фиксация длины.....	460
S.3.8. Фиксация радиуса.....	460
S.3.9. Ограничение угловое положение.....	460
S.3.10. Касание.....	460
S.3.11. Множественные и концевые касания.....	462
S.3.12. Зеркальная симметрия.....	463
S.3.13. Биссектриса.....	463
S.3.14. Средняя точка.....	464
S.3.15. Взаимное размещение объектов.....	464
S.4. ВЫЧИСЛЕНИЕ И ДИАГНОСТИКА ДВУМЕРНЫХ ОГРАНИЧЕНИЙ.....	465
S.4.1. Вычисление системы ограничений.....	465
S.4.2. Изменение или получение состояния геометрии.....	465
S.4.3. Начальное приближение.....	466
S.4.4. Переопределенные совместные и несовместные системы ограничений.....	466
S.4.5. Недоопределенные системы ограничений.....	467
S.4.6. Анализ степеней свободы.....	467
S.4.7. Информационные запросы.....	468

S.4.8. Драггинг геометрических объектов.....	468
S.4.9. Геометрическая трансформация.....	469
S.4.10. Тест избыточности.....	469
S.5. ДВУМЕРНЫЕ СПЛАЙНЫ И ПАРАМЕТРИЧЕСКИЕ КРИВЫЕ.....	470
S.5.1. Сплайновые кривые.....	470
S.5.2. Параметрические кривые общего вида.....	470
S.5.3. Ограничения, основанные на параметризации кривых.....	470
T.1. ОБМЕН ДАННЫМИ С ДРУГИМИ СИСТЕМАМИ.....	472
T.1.1. Принципы работы конвертера.....	472
T.1.2. Порядок работы с конвертером.....	472
T.1.3. Свойство конвертера IConvertorProperty3D.....	474
T.1.4. Документ модели ItModelDocument.....	476
T.1.5. Индикатор процесса выполнения IProgressIndicator.....	477
T.1.6. Архитектура дерева модели.....	478
T.1.7. Свойства элементов модели ItModelInstanceProperties.....	478
T.1.8. Компоненты ItModelDetail и ItModelAssembly.....	479
T.1.9. Вставки ItModelInstance.....	480
T.2. КОНВЕРТЕРЫ ГРАНИЧНОГО ПРЕДСТАВЛЕНИЯ.....	481
T.2.1. Общая характеристика функций конвертеров граничного представления.....	481
T.2.2. Общие сведения о параметрах конвертеров граничного представления.....	481
T.2.3. Импорт модели в формате SAT.....	481
T.2.4. Экспорт модели в формат SAT.....	482
T.2.5. Импорт модели в формате IGES.....	482
T.2.6. Экспорт модели в формат IGES.....	483
T.2.7. Импорт модели форматов X_T и X_B.....	483
T.2.8. Экспорт модели в форматы X_T, X_B.....	484
T.2.9. Импорт модели формата STEP.....	484
T.2.10. Экспорт модели в формат STEP.....	484
T.3. КОНВЕРТЕРЫ ПОЛИГОНАЛЬНОГО ПРЕДСТАВЛЕНИЯ.....	486
T.3.1. Общая характеристика функций конвертеров полигонального представления.....	486
T.3.2. Общие сведения о параметрах конвертеров полигонального представления.....	486
T.3.3. Импорт модели формата STL.....	486
T.3.4. Экспорт модели в формате STL.....	487
T.3.5. Импорт модели формата VRML.....	487
T.3.6. Экспорт модели в формат VRML.....	488

ВВЕДЕНИЕ

Общие сведения

C3D представляет собой программный продукт, включающий в себя геометрическое ядро, параметрическое ядро и конвертер.

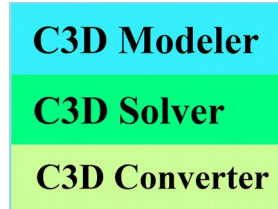
C3D предназначен для использования в системах автоматизированного проектирования в качестве программного компонента.

C3D сосредотачивает в себе программную реализацию математических методов построения численных моделей геометрии реальных и воображаемых объектов, а также математических методов обслуживания этих моделей. Численные модели используются в системах, выполняющих проектирование (Computer Aided Design), расчёты (Computer Aided Engineering) и производство (Computer Aided Manufacturing) моделируемых объектов. Численные модели геометрии реальных и воображаемых объектов называют геометрическими моделями.

Геометрическая модель содержит описание формы моделируемого объекта и описание связей элементов модели. Кроме того, в геометрическую модель включают историю её построения, хранящую способы и последовательность построения модели, а элементы геометрической модели наделяют атрибутами, несущими информацию о физических, технологических и других свойствах элементов.

Структура и отличительные особенности

C3D состоит из трёх модулей, приведённых на рисунке: геометрического ядра C3D Modeler, параметрического ядра C3D Solver и конвертера C3D Converter.



Геометрическое ядро C3D Modeler выполняет построение геометрической модели, редактирование модели путем изменения ее внутренних данных, построение триангуляции, вычисление инерционных характеристик модели, построение плоских проекций модели, определение столкновений элементов модели.

Параметрическое ядро C3D Solver обеспечивает взаимосвязь элементов геометрической модели, что позволяет редактировать модель, синхронно изменяя ее элементы, строить подобные модели, моделировать механизмы.

Конвертер C3D Converter выполняет обмен данными геометрической модели с другими системами.

Наличие в одном программном продукте геометрического ядра, параметрического ядра и конвертера является одной из отличительных особенностей C3D. Другими особенностями C3D являются открытость и расширяемость. C3D предоставляет прямой доступ к объектам, что позволяет расширять функционал путем наследования от объектов C3D.

Выполняемые функции

Геометрическое ядро C3D Modeler строит геометрическую модель, которая содержит: описание формы моделируемого объекта, описание связей элементов геометрической модели, историю построения модели, атрибуты элементов геометрической модели.

Для описания формы моделируемого объекта геометрическое ядро C3D Modeler использует граничное представление (Boundary Representation). Геометрическое ядро C3D Modeler поддерживает

также полигональное представление (Polygonal Representation). Для построения геометрической модели используются методы твердотельного моделирования (Solid Modeling), методы поверхностного моделирования (Surface Modeling), методы прямого моделирования (Direct Modeling) и методы моделирования из листового металла (Sheet Metal Modeling).

Используемые для построения модели методы, их последовательность и необходимые исходные данные хранятся в журнале построения. Журнал построения позволяет редактировать геометрическую модель путем полного перестроения модели с новыми параметрами.

Дополнительная информация об элементах геометрической модели хранится в атрибутах. Атрибутами снабжены объекты геометрической модели, а также отдельные элементы этих объектов.

Геометрическое ядро C3D Modeler выполняет построение полигональной модели по ее граничному представлению. Полигональная модель строится путем триангуляции элементов геометрической модели и используется для визуализации и расчетов. Кроме того, геометрическое ядро C3D Modeler выполняет построение плоских проекций геометрической модели (Mapping), вычисление инерционных характеристик геометрической модели (Inertia Properties), определение столкновений элементов модели (Collision Detection).

Параметрическое ядро C3D Solver налагает вариационные связи на элементы геометрической модели. Эти связи называют геометрическими ограничениями (Geometric Constraints). Геометрические ограничения работают с трехмерными и двумерными объектами геометрической модели. Геометрические ограничения представляют собой условия, наложенные на элементы модели, выраженные с помощью уравнений. Геометрические ограничения позволяют создавать сборочные единицы путем наложения зависимостей на составляющие ее детали. Размеры одной или нескольких деталей можно связать уравнениями, что даст возможность синхронно редактировать элементы геометрической модели. Если связать почти все степени свободы геометрической модели и описать закон изменения для оставшихся степеней свободы, то можно привести в движение геометрическую модель и смоделировать механизм.

Модуль C3D Converter обеспечивает обмен данными с другими системами путем чтения и записи геометрической модели в форматах STEP, IGES, SAT (ACIS), X_T, X_B (Parasolid), STL, VRML, JT. Форматы STEP, IGES, SAT, X_T, X_B передают геометрическую модель в граничном представлении, форматы STL и VRML передают геометрическую модель в полигональном представлении, формат JT передает геометрическую модель в гибридном представлении.

Теоретические основы

Граничное представление, с которым работает C3D, даёт точное описание геометрической формы моделируемого объекта. Для описания геометрической формы C3D использует набор граней, проходящих по границе, отделяющей внутреннее пространство моделируемого объекта от остальной части пространства. Грани представляют собой криволинейные поверхности, стыкующиеся друг с другом по своим краям. Края граней могут иметь сложную форму. Формирование и стыковка граней выполняются во время построения модели. Это обеспечивают методы построения модели и организация данных в C3D.

Геометрические ограничения, описывающие связи элементов модели и прочие условия, формулируются в виде уравнений. Для поиска решения, удовлетворяющего уравнениям, параметрическое ядро C3D Solver использует вариационный подход. Вариационный подход обеспечивает равноправие всех геометрических ограничений.

С помощью триангуляции граничное представление позволяет построить полигональное представление модели, которое используется для визуализации и геометрических расчётов. Полигональные объекты состоят из треугольных и четырёхугольных пластин, аппроксимирующих грани, и ломаных, аппроксимирующих рёбра. Триангуляция в геометрическом ядре C3D Modeler выполняется по принципу Делоне в плоскости параметров поверхностей.

Для кривых и поверхностей геометрическое ядро C3D Modeler может создать NURBS (Non-Uniform Rational B-Spline) копии. NURBS - объекты используются для прямого моделирования и для обмена данными, когда отсутствует прямое соответствие между объектами C3D и объектами обменных форматов.

Применяемые в C3D математические объекты, методы и алгоритмы приведены в книге Голованова Н.Н. «Геометрическое моделирование», Москва, Курс, Инфра-М, 2016,

<http://znanium.com/catalog.php?bookinfo=520536> и в книге «Geometric Modeling», Nikolay Golovanov, <http://www.amazon.com/Geometric-Modeling-The-mathematics-shapes/dp/1497473195>.

Комплектация

В комплект поставки C3D входят: библиотечные файлы c3d.lib, c3d.dll, libc3d.so, libmath.dylib и набор заголовочных файлов Include/*.h. В операционной системе Windows библиотечные файлы собраны в конфигурациях 32bit/64bit, ISO/Unicode, Debug/Release в средах разработки VisualStudio 2008, VisualStudio 2010, VisualStudio 2012, VisualStudio 2013 и VisualStudio 2015. В операционной системе Linux библиотечные файлы собраны компилятором GCC в конфигурациях 64bit, Unicode, Debug/Release. В операционной системе Mac Os библиотечные файлы собраны компилятором Clang в конфигурациях 64bit, Unicode, Debug/Release. Для операционной системы Android библиотечные файлы собраны компилятором GCC для архитектуры armeabi-v7a и arm64-v8a.

По заголовочным файлам Include/*.h сгенерирована документация C3D в виде *.chm файлов. Заголовочные файлы содержат описание объектов и методов C3D на русском и английском языках. Описание объектов и методов C3D также приведено в данном руководстве. В файле Changes.txt приведены изменения интерфейса программного продукта.

В комплект поставки входит обертка для C3D, позволяющая использовать технологию .NET при разработке приложений на языке C#.

Вместе с C3D поставляется приложение test.exe для системы Windows, демонстрирующее возможности C3D, исходные тексты этого приложения, файл CMakeLists.txt для генерации проекта приложения и набор файлов с моделями. Также доступны исходные тексты примера использования C3D в операционной системе Android.

После запуска приложения test.exe необходимо ввести ключ и сигнатуру, выбрав в меню: Помощь->Лицензионный_ключ, Сигнатура.

Тестовое приложение

Готовое тестовое приложение test.exe вместе с c3d.dll для системы Windows располагается в каталоге Example/Demo.

Файлы Test_VS2008.sln и Test_VS2008.vcproj содержат соответственно решение и проект тестового приложения для Microsoft VisualStudio2008.

Файлы Test_VS2012.sln, Test_VS2012.vcxproj и Test_VS2012.vcxproj.filters содержат соответственно решение и проект тестового приложения геометрического ядра C3D для Microsoft VisualStudio 2012.

Проект тестового приложения в различных средах разработки можно создать с помощью системы Сmake. Для создания проекта и компиляции тестового приложения требуется выполнить следующие действия:

1. Создать тестовый каталог (например, TestApp) в удобном для работы месте.
2. Выбрать в каталоге «C3D» архив, соответствующий используемой среде разработки.
3. Скопировать каталоги <Include>, <Debug> и <Release> из выбранного архива в тестовый каталог.
4. Скопировать каталог <Source> из архива «Example» в тестовый каталог.
5. Убедиться, что в тестовом каталоге (TestApp) находятся папки: <Debug>, <Include>, <Release> и <Source>.
6. Установить CMake, выбрав в процессе установки опцию «Add CMake to the system PATH for all users».
7. Создать проект тестового приложения, выполнив следующие действия:
Запустить CMake, который сгенерирует проект по файлу CMakeLists.txt.
Для «Where is the source code» указать каталог <path_to_testapp>\TestApp\Source.
Для «Where to build the binaries» указать каталог <path_to_testapp>\TestApp\Build.
Нажать кнопку **Configure** для конфигурирования проекта.

На запрос «Create Directory» подтвердить согласие на создание каталога <path_to_testapp>\TestApp\Build.

По запросу «Specify the generator for this project» указать соответствующую версии C3D конфигурацию среды разработки.

Нажать кнопку **Generate** для генерации файлов проекта.

8. Запустить созданный проект TestApp\Build\Test.sln тестового приложения в среде разработки.
9. Для активации C3D перед компиляцией необходимо изменить вызов метода EnableMathModules(...) в конструкторе объекта «Manager» в файле test_manager.cpp с реальными ключом и сигнатурой.
10. После компиляции необходимо запустить созданное тестовое приложение test.exe из каталога TestApp\Debug или TestApp\Release, соответственно.

Указанные выше действия описаны в файле readme.txt.

Разработка в среде .NET

C3D может работать в среде .NET. Для разработки приложений в среде .NET следует использовать обертку, входящую в комплект поставки C3D.

Обертка для C3D представляет собой dll-файл NetC3D.dll, собираемый на платформе .Framework4.5 в конфигурациях 32bit/64bit, Debug/Release и средах разработки VisualStudio2012, VisualStudio2013. Библиотека скомпилирована с поддержкой подписи «Strong Name».

Для использования C3D в приложениях, разрабатываемых на C#, нужно выполнить следующие действия:

1. Выбрать из комплекта C3D файл NetC3D.dll нужной конфигурации: 32bit/64bit, Debug/Release в одной из сред разработки: VisualStudio2012/VisualStudio2013.
2. Положить в один каталог с NetC3D.dll файл c3d.dll из этого же комплекта, той же конфигурации и среды разработки: 32bit/64bit, Debug/Release, VisualStudio2012/VisualStudio2013.
3. Добавить файл NetC3D.dll в разрабатываемый проект: References->Add Reference->Browse..., далее выбрать файл NetC3D.dll.
4. Для дальнейшей работы перед вызовом функций из NetC3D.dll нужно ввести лицензионный ключ и сигнатуру. Способ ввода ключа и сигнатуры может быть следующим:

```
var key = Environment.GetEnvironmentVariable("C3Dkey");  
var signature = Environment.GetEnvironmentVariable("C3Dsignature");  
NetC3D.ToolEnabler.EnableMathModules(key,signature );
```

где C3Dkey и C3Dsignature – переменные среды, содержащие ключ и сигнатуру.

М.1. МЕТОДЫ ПОСТРОЕНИЯ ТЕЛ

Основными элементами геометрической модели служат тела. Геометрическое ядро C3D строит тела, описывающие всю поверхность или часть поверхности моделируемого объекта. Тела могут быть замкнутыми и незамкнутыми. Замкнутое тело не содержит краевых ребер и описывает всю поверхность моделируемого объекта, а также множество его внутренних точек. Незамкнутое тело содержит краевые ребра и описывает только часть поверхности моделируемого объекта. Многие тела имеют простую форму и строятся на основе точек, кривых и поверхностей.

М.1.1. Построение элементарного тела

Метод
MbResultType
ElementarySolid (SArray<MbCartPoint3D> & **points**,
ElementaryShellType *solidType*,
const MbSNameMaker & names,
MbSolid *& **result**)

выполняет построение элементарного тела по заданным точкам в форме сферы, тора, цилиндра, конуса, прямого параллелепипеда, пирамиды, скруглённой плиты.

Входными параметрами метода являются:

- **points** – множество контрольных точек,
- *solidType* – тип создаваемого тела,
- names – именователю граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле *action_solid.h*.

Параметр **points** содержит контрольные точки для построения тела. Параметр *solidType* определяет тип создаваемого тела. Параметр names обеспечивает именование граней построенного тела.

Для разных типов создаваемого тела требуется разное количество контрольных точек. В табл. М.1.1.1 приведено количество контрольных точек множества **points**, необходимое для создания тела типа *solidType*.

Табл. М.1.1.1.

<i>solidType</i>	Тип тела	Количество контрольных точек
et_Sphere	шар	3 точки
et_Torus	тор	3 точки
et_Cylinder	цилиндр	3 точки
et_Cone	конус	3 точки
et_Block	блок	4 точки
et_Wedge	клин	4 точки
et_Plate	плита	4 точки
et_Prism	призма	количество вершин основания+1 точка
et_Pyramid	пирамида	количество вершин основания+1 точка

При построении сферы точка множества **points**[0] определяет центр сферы, точка **points**[1] определяет направление оси **axisZ** локальной системы координат сферы, точка **points**[2] вместе с предыдущими точками определяет плоскость расположения осей **axisX** и **axisZ** локальной системы

координат сферы. Расстояние между точками **points[0]** и **points[2]** определяет радиус сферы, рис. М.1.1.1.

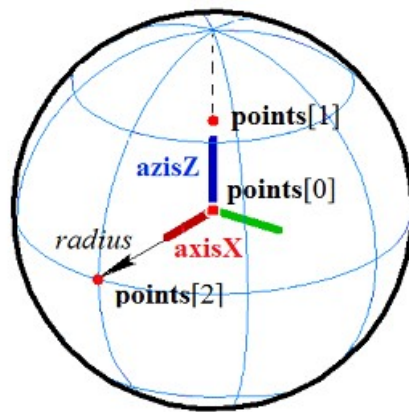


Рис. М.1.1.1.

При построении тора точка множества **points[0]** определяет центр тора, точка **points[1]** определяет направление оси **axisX** локальной системы координат тора, точка **points[2]** вместе с предыдущими точками определяют плоскость, в которой располагаются оси **axisX** и **axisZ** локальной системы координат тора. Расстояние между точками **points[0]** и **points[1]** определяет больший радиус тора, расстояние между точками **points[1]** и **points[2]** определяет меньший радиус тора, рис. М.1.1.2.

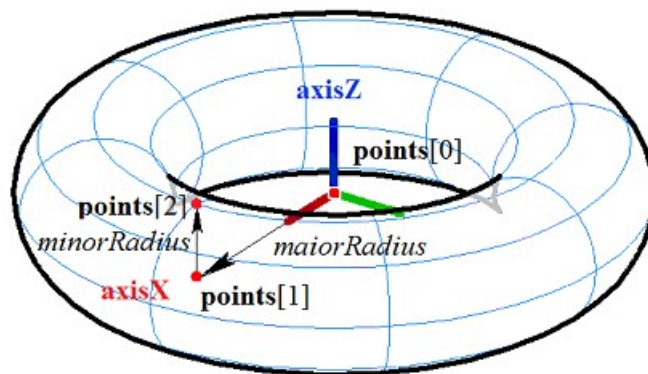


Рис. М.1.1.2.

При построении цилиндра точка множества **points[0]** определяет центр нижнего основания цилиндра, точка **points[1]** определяет центр верхнего основания цилиндра и направление оси **axisZ** локальной системы координат цилиндра, точка **points[2]** вместе с предыдущими точками определяет плоскость расположения осей **axisX** и **axisZ** локальной системы координат цилиндра. Расстояние между точками **points[0]** и **points[1]** определяет высоту цилиндра, расстояние от оси **axisZ** до точки **points[2]** определяет радиус цилиндра, рис. М.1.1.3.

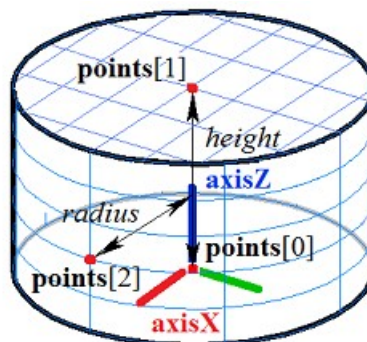


Рис. М.1.1.3.

При построении конуса точка множества **points[0]** определяет вершину конуса, точка **points[1]** определяет центр основания конуса и направление оси **axisZ** локальной системы координат конуса,

точка **points[2]** вместе с предыдущими точками определяет плоскость расположения осей **axisX** и **axisZ** локальной системы координат конуса. Расстояние между точками **points[0]** и **points[1]** определяет высоту конуса, угол конуса определяется из условия, что точка **points[2]** лежит на боковой поверхности конуса, рис. М.1.1.4.

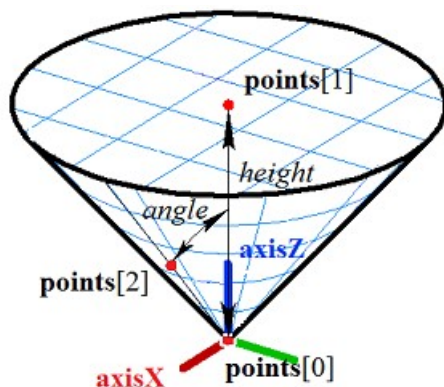


Рис. М.1.1.4.

При построении прямоугольного блока точки множества **points[0]** и **points[1]** определяют ребро и две вершины блока, точка **points[2]** вместе с предыдущими точками определяет плоскость нижнего основания блока, через точку **points[2]** проходит ребро блока, параллельное ребру **points[0]** и **points[1]**, точка **points[3]** определяет плоскость верхнего основания блока, рис. М.1.1.5.

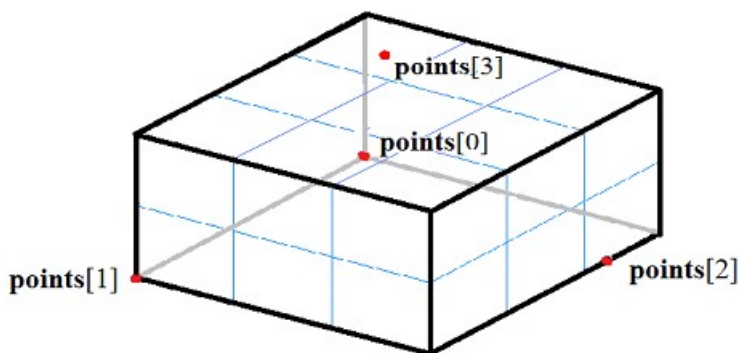


Рис. М.1.1.5.

При построении прямоугольного клина точки множества **points[0]** и **points[1]** определяют ребро и две вершины клина, точка **points[2]** вместе с предыдущими точками определяет плоскость нижнего основания клина и его вершину, через точку **points[2]** проходит ребро клина, параллельное ребру **points[0]** и **points[1]**, точка **points[3]** определяет плоскость верхнего основания клина, рис. М.1.1.6.

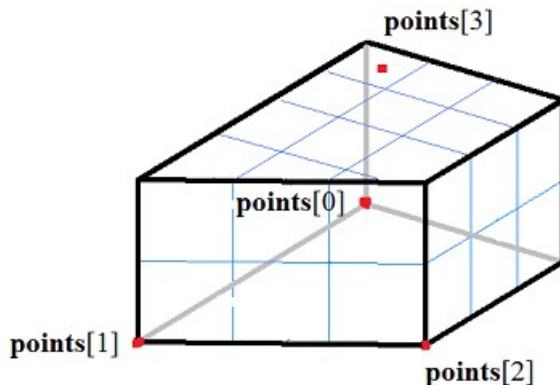


Рис. М.1.1.6.

При построении прямоугольной пластины с цилиндрическими торцами точки множества **points[0]** и **points[1]** определяют ребро и две вершины пластины, точка **points[2]** вместе с предыдущими точками определяет плоскость нижнего основания пластины, через точку **points[2]** проходит ребро

пластины, параллельное ребру **points[0]** и **points[1]**, точка **points[3]** определяет плоскость верхнего основания пластины, рис. М.1.1.7.

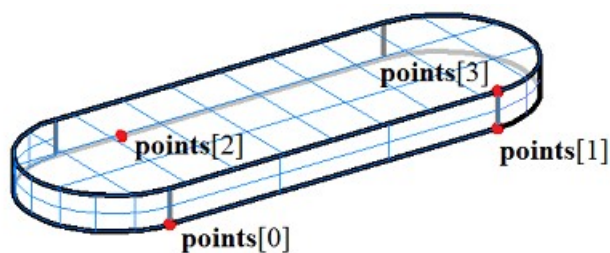


Рис. М.1.1.7.

При построении прямой призмы с многоугольником в основании точки **weightCentre**, **points[0]** и **points[1]** определяют плоскость нижнего основания призмы, где **weightCentre** центр тяжести точек основания. Проекция точек **points[0]**, **points[1]**, ..., **points[n-1]** определяют многоугольник основания, высоту призмы определяет расстояние от плоскости нижнего основания до последней точки **points[n]**. На рис. М.1.1.8 приведена прямая призма с пятиугольным основанием.

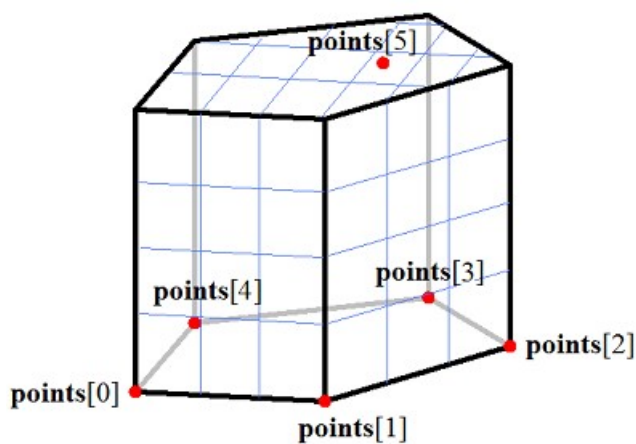


Рис. М.1.1.8.

При построении пирамиды с многоугольником в основании точки **weightCentre**, **points[0]**, **points[1]** определяют плоскость нижнего основания пирамиды, где **weightCentre** центр тяжести точек основания. Проекция точек **points[0]**, **points[1]**, ..., **points[n-1]** определяют многоугольник основания, последняя точка **points[n]** определяет вершину пирамиды. На рис. М.1.1.9 приведена пирамида с пятиугольным основанием.

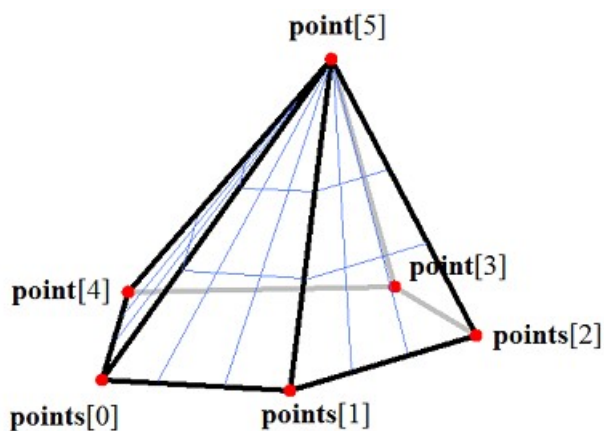


Рис. М.1.1.9.

Точки множества **points**, определяющие основание призмы или пирамиды, можно расположить в вершинах правильного многоугольника. Многоугольник основания призмы или пирамиды может быть произвольным.

Метод **ElementarySolid** добавляет в журнал построенного тела строитель MbElementarySolid, который содержит все необходимые для построения тела данные. Строитель MbElementarySolid объявлен в файле `cg_elementary_solid.h`.

Тестовое приложение `test.exe` выполняет построение элементарного тела по заданным точкам командами меню «Создать->Тело->Элементарное->» и «Создать->Тело->По точкам->».

М.1.2. Построение элементарного тела по заданной поверхности

Метод
MbResultType
ElementarySolid (const MbSurface & **surface**,
const MbSNameMaker & names,
MbSolid *& **result**)

выполняет построение элементарного тела по заданной поверхности.

Входными параметрами метода являются:

- **surface** – элементарная поверхность,
- names – именователъ граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_solid.h`.

Параметр **surface** содержит исходную поверхность. Параметр names обеспечивает именование граней построенного тела.

Элементарной поверхностью может быть сфера MbSphere, поверхность тора MbTorus, цилиндрическая поверхность MbCylinder, коническая поверхность MbCone. На рис. М.1.2.1 приведена сферическая поверхность и построенное по ней тело.

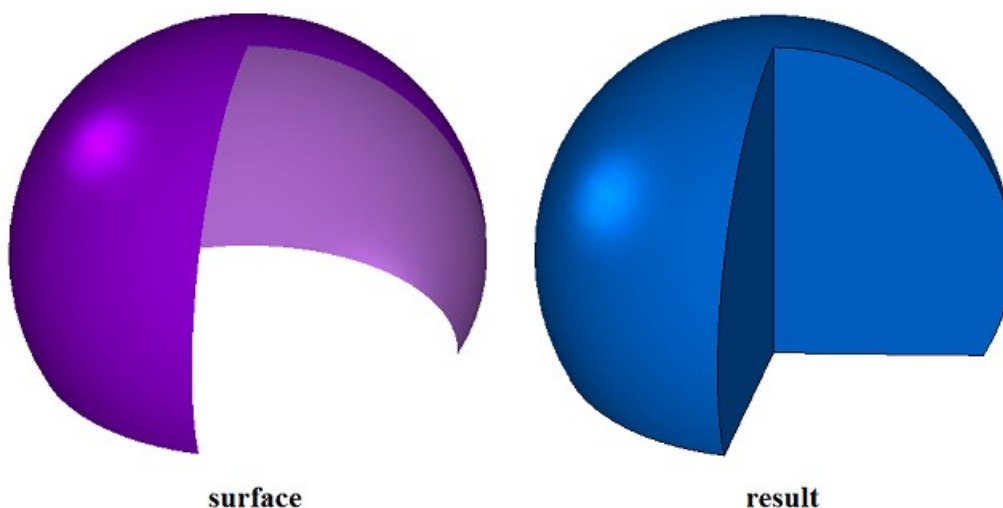


Рис. М.1.2.1.

На рис. М.1.2.2 приведена поверхность тора и построенное по ней тело.

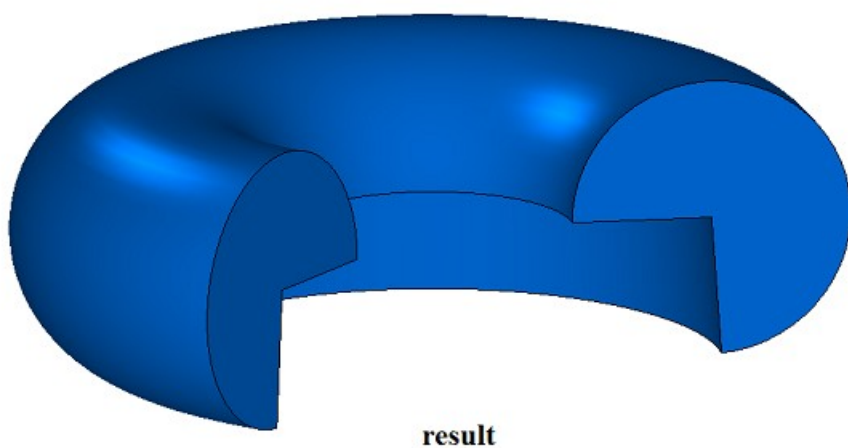
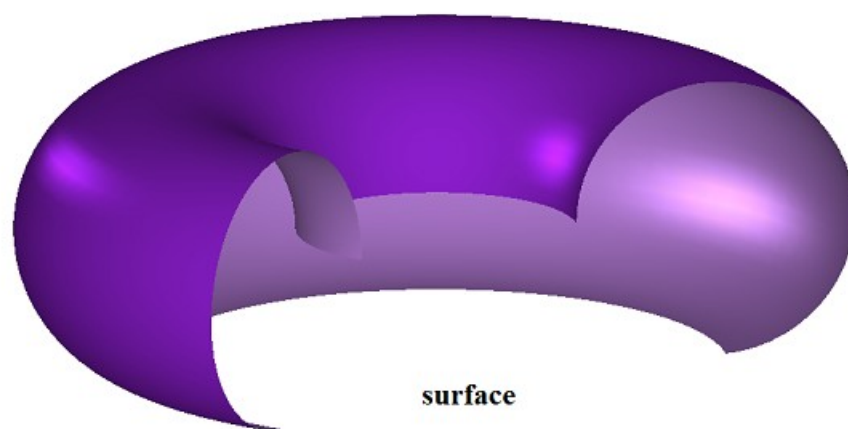


Рис. М.1.2.2.

На рис. М.1.2.3 приведена цилиндрическая поверхность и построенное по ней тело.

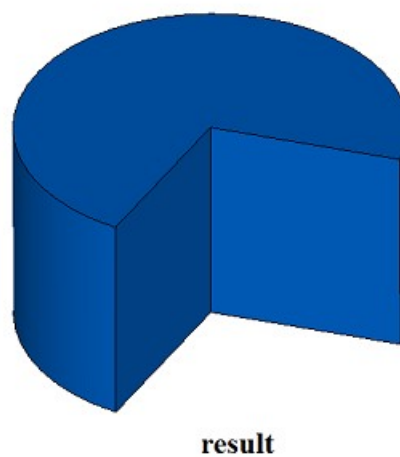
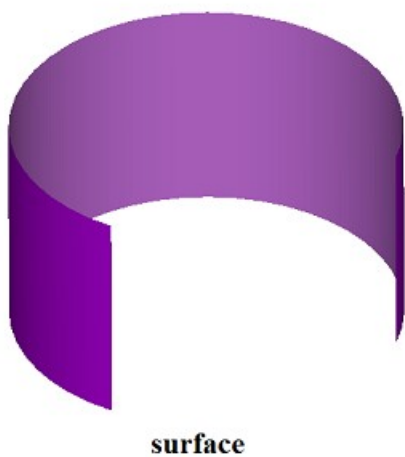


Рис. М.1.2.3.

На рис. М.1.2.4 приведена коническая поверхность и построенное по ней тело.

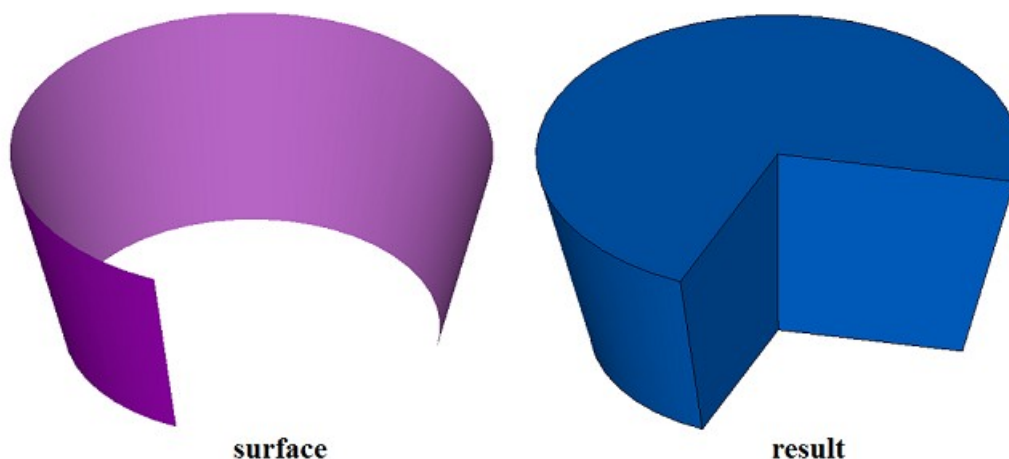


Рис. М.1.2.4.

Если перечисленные поверхности будут циклически замкнутыми, то построенные по ним тела будут иметь соответствующую форму. Если поверхность не является ни одной из перечисленных элементарных поверхностей, то метод возвращает код ошибки `rt_Error`.

Метод **ElementarySolid** добавляет в журнал построенного тела строитель `MbRevolutionSolid`, который содержит все необходимые для построения тела данные. Строитель `MbRevolutionSolid` объявлен в файле `cg_revolution_solid.h`.

Тестовое приложение `test.exe` выполняет построение элементарного тела по заданной поверхности командой меню «Создать->Тело->По поверхности->По элементарной поверхности».

М.1.3. Построение тела выдавливания

Метод
`MbResultType`
ExtrusionSolid (`const MbSweptData & sweptData`,
`const MbVector3D & direction`,
`const MbSolid * solid1`,
`const MbSolid * solid2`,
`bool checkIntersection`,
`ExtrusionValues & params`,
`const MbSNameMaker & names`,
`PArray<MbSNameMaker> & cnames`,
`MbSolid *& result`)

выполняет построение тела выдавливания.

Входными параметрами метода являются:

- **sweptData** – данные об образующих кривых,
- **direction** – направление выдавливания,
- **solid1** – используется при опции «До ближайшего объекта» в прямом направлении,
- **solid2** – используется при опции «До ближайшего объекта» в обратном направлении,
- **checkIntersection** – флаг для объединения тел `solid1` и `solid2` с проверкой пересечения,
- **params** – параметры построения,
- **names** – именованная граней,
- **cnames** – именователи сегментов образующих кривых.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Тело выдавливания относится к разновидности тел движения, которые получают путем движения образующей кривой вдоль направляющей кривой. Направляющей кривой тела выдавливания служит

отрезок прямой. Тело выдавливания строится путем движения одной или нескольких кривых вдоль отрезка, направление которого задает вектор **direction**.

Параметр **sweptData** содержит информацию об образующих кривых. Класс MbSweptData и структура ExtrusionValues описаны в файле swept_parameter.h. Образующие кривые могут представлять собой двумерные контуры **contours** на поверхности **surface** или контуры в пространстве **contours3D**. В частном случае двумерные контуры **contours** могут располагаться на плоскости. Ориентация контуров **contours** может быть произвольной. Контуры **contours** могут быть вложены друг в друга. Контуры **contours** не должны пересекать друг друга.

Построение тела может выполняться в прямом направлении относительно вектора **direction**, в обратном направлении относительно вектора **direction** и в обоих направлениях. Параметры построения в каждом направлении задают объекты MbSweptSide.

Параметр *params* содержит информацию о способе выдавливания в прямом направлении MbSweptSide *side1* и информацию о способе выдавливания в обратном направлении MbSweptSide *side2*. Выдавливание в каждую сторону может выполняться тремя способами. При *way==sw_scalarValue* выдавливание выполняется на длину *scalarValue* в направлении *side1* или *side2*, соответственно. При *way==sw_shell* выдавливание выполняется до ближайшего объекта **solid1** или **solid2**, соответственно. При *way==sw_surface* выдавливание выполняется до поверхности *side1.surface* или *side2.surface*, соответственно, при *side1.distance=0* или *side2.distance=0*. При *way==sw_surface* и *distance!=0* выдавливание выполняется до эквидистантной поверхности к *side1.surface* или до эквидистантной поверхности к *side2.surface*, соответственно. Если *side1.rake!=0* или *side2.rake!=0*, то выдавливание выполняется с уклоном *side1.rake* или *side2.rake* в соответствующую сторону. Параметр *params.thickness1* задает отступ наружу от образующей кривой, а параметр *params.thickness2* задает отступ внутрь от образующей кривой. Параметр *params.shellClosed* управляет замкнутостью построенного тела. Параметр *params.checkSelfInt* сообщает о необходимости проверки результата построения на самопересечение. По умолчанию *params.checkSelfInt=false*, проверка не выполняется.

На рис. М.1.3.1 приведены данные, используемые при построении, и схема наследования параметров построения тела выдавливания ExtrusionValues & *params*.

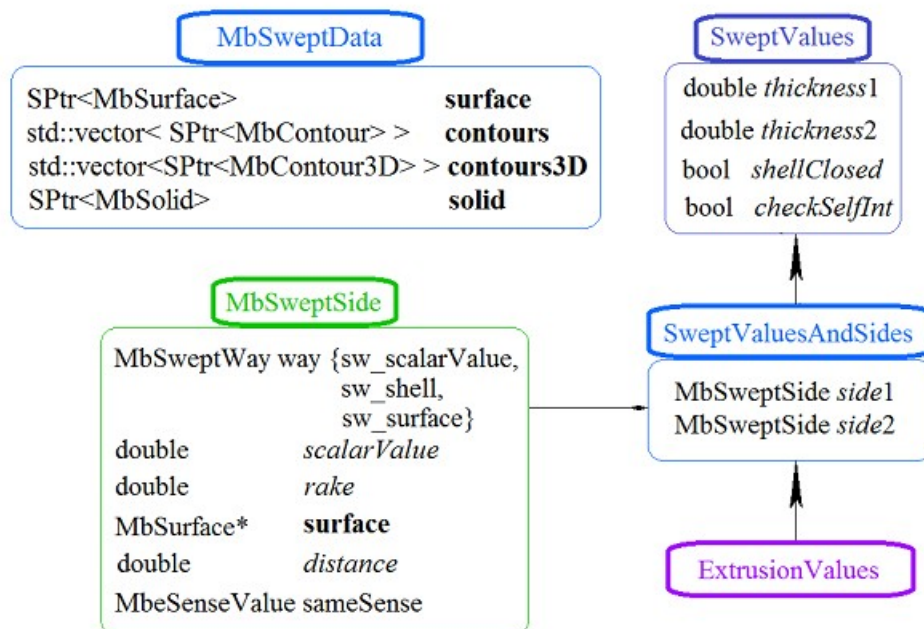


Рис. М.1.3.1.

Параметры names и snames обеспечивают именование граней построенного тела.

На рис. М.1.3.2 приведен двумерный контур **contour** и плоская поверхность **surface** ([MbPlane](#)).

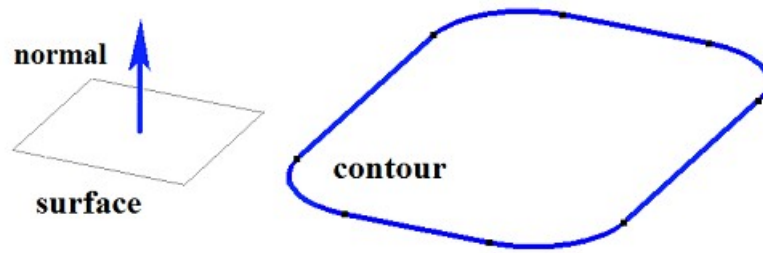


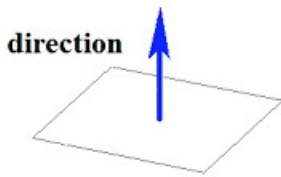
Рис. М.1.3.2.

На рис. М.1.3.3 приведено замкнутое тело, построенное выдавливанием по заданным параметрам контура, показанного на рис. М.1.3.2. Каждому сегменту контура соответствует грань тела, имя которой взято от соответствующего элемента генератора имен `snames[0]`.

`params.shellClosed = true`
`params.thickness1 = 0`
`params.thickness2 = 0`

`solid1 = 0`

`params.side1.surface = 0`



`sweptData.surface`

`sweptData.contours[0]`

`params.side2.surface = 0`

`solid2 = 0`

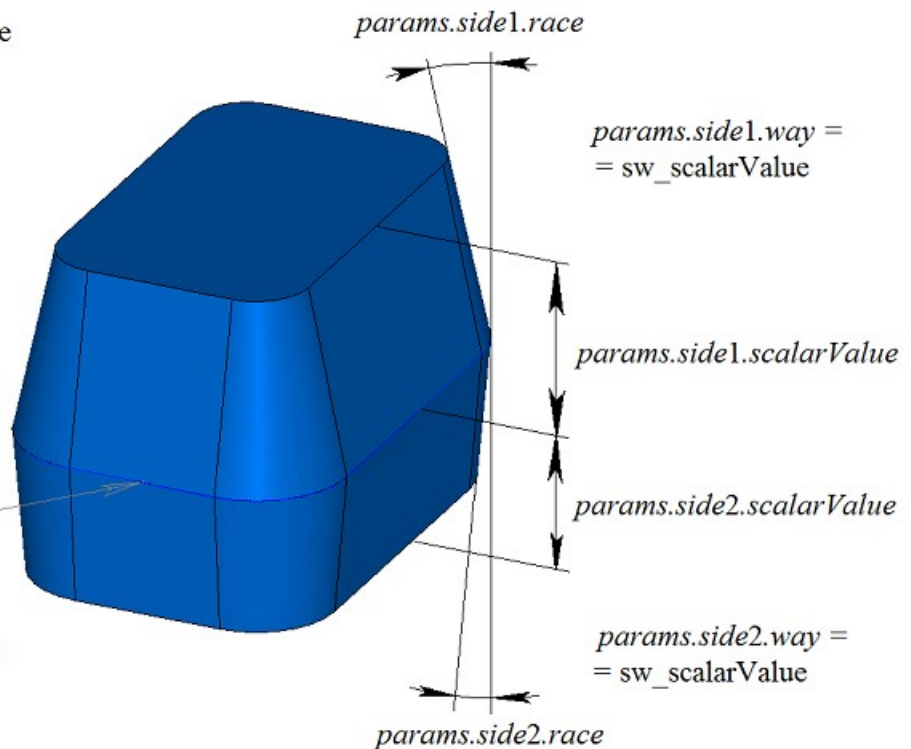


Рис. М.1.3.3.

На рис. М.1.3.4 приведено замкнутое тонкостенное тело, построенное выдавливанием по заданным параметрам контура, показанного на рис. М.1.3.2.

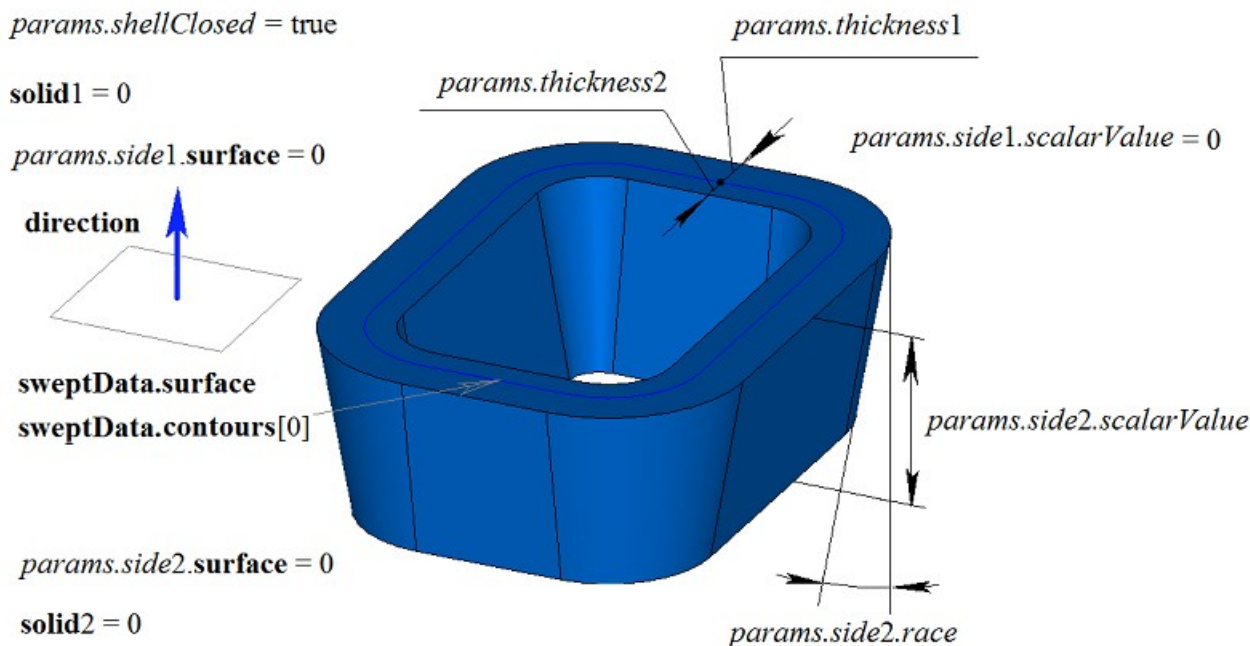


Рис. М.1.3.4.

На рис. М.1.3.5 приведено незамкнутое тело, построенное выдавливанием по заданным параметрам контура, показанного на рис. М.1.3.2. Параметры построения тела, показанного на рис. М.1.3.3, отличаются от параметров построения тела, показанного на рис. М.1.3.5, только величиной $params.shellClosed$.

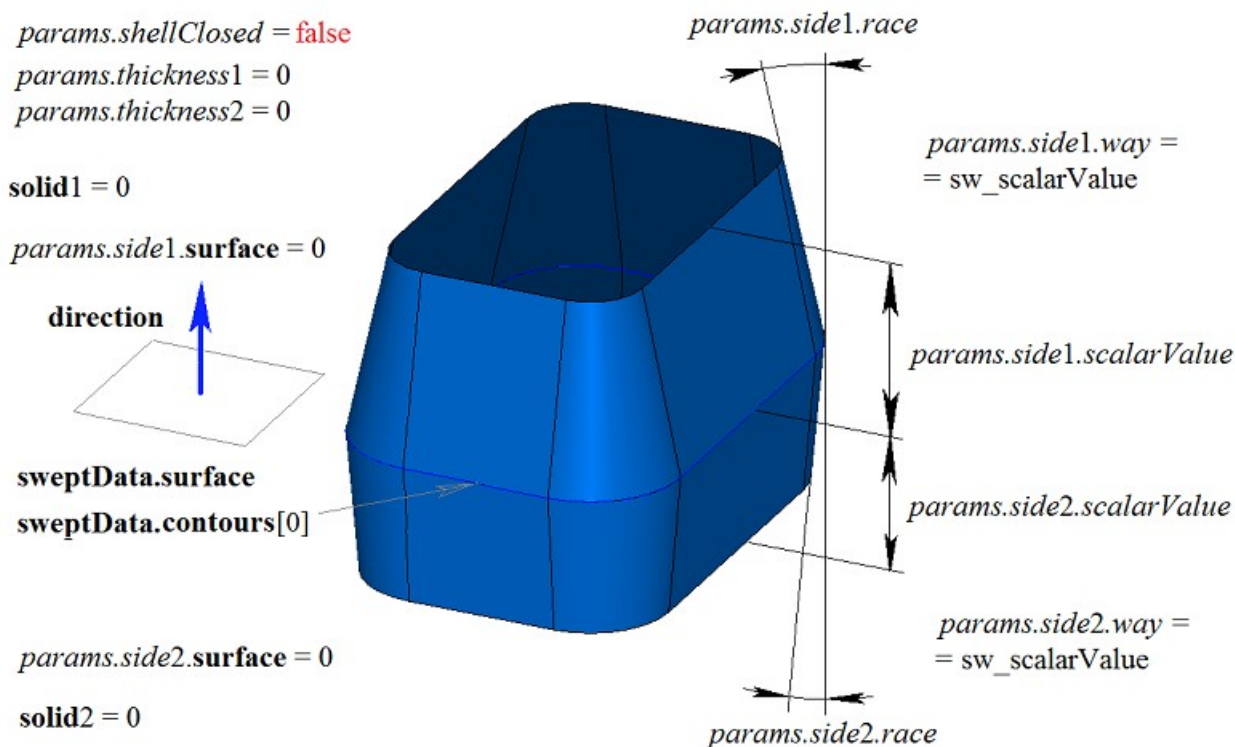


Рис. М.1.3.5.

На рис. М.1.3.6 приведен двумерный контур **contour**, плоская поверхность **surface** ([MbPlane](#)) и два тела **solid1** и **solid2**, которые будут использоваться при построении тела выдавливания. Для построения необходимо, чтобы тела **solid1** и **solid2** полностью перекрывали путь движения контура в

соответствующем направлении. При этом следует учитывать параметры *params.side1.rake*, *params.side2.rake*, *params.thickness1*, *params.thickness2*.

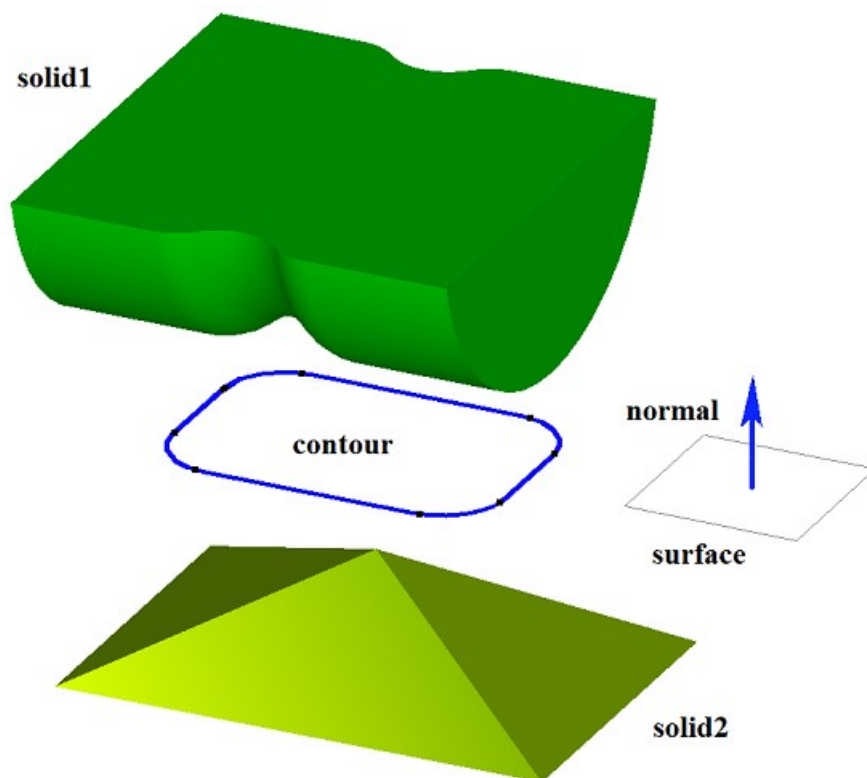


Рис. М.1.3.6.

Такое построение выполняется выдавливанием контура на длину, превосходящую максимальное расстояние до заданного тела с последующим вычитанием заданного тела из построенного тела.

На рис. М.1.3.7 приведено тело, построенное выдавливанием контура, показанного на рис. М.1.3.6, с опциями «До ближайших объектов», в качестве которых заданы тела **solid1** и **solid2**.

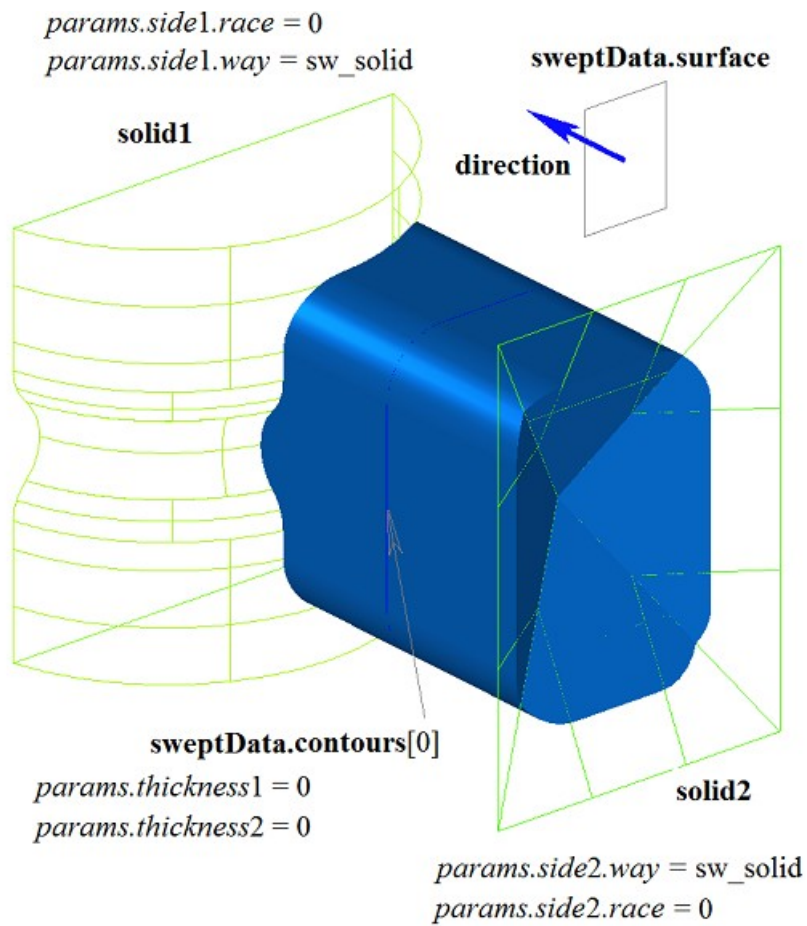


Рис. М.1.3.7.

На рис. М.1.3.8 приведено тонкостенное тело с уклоном граней, построенное выдавливанием контура, показанного на рис. М.1.3.6, с опциями «До ближайших объектов», в качестве которых заданы тела **solid1** и **solid2**.

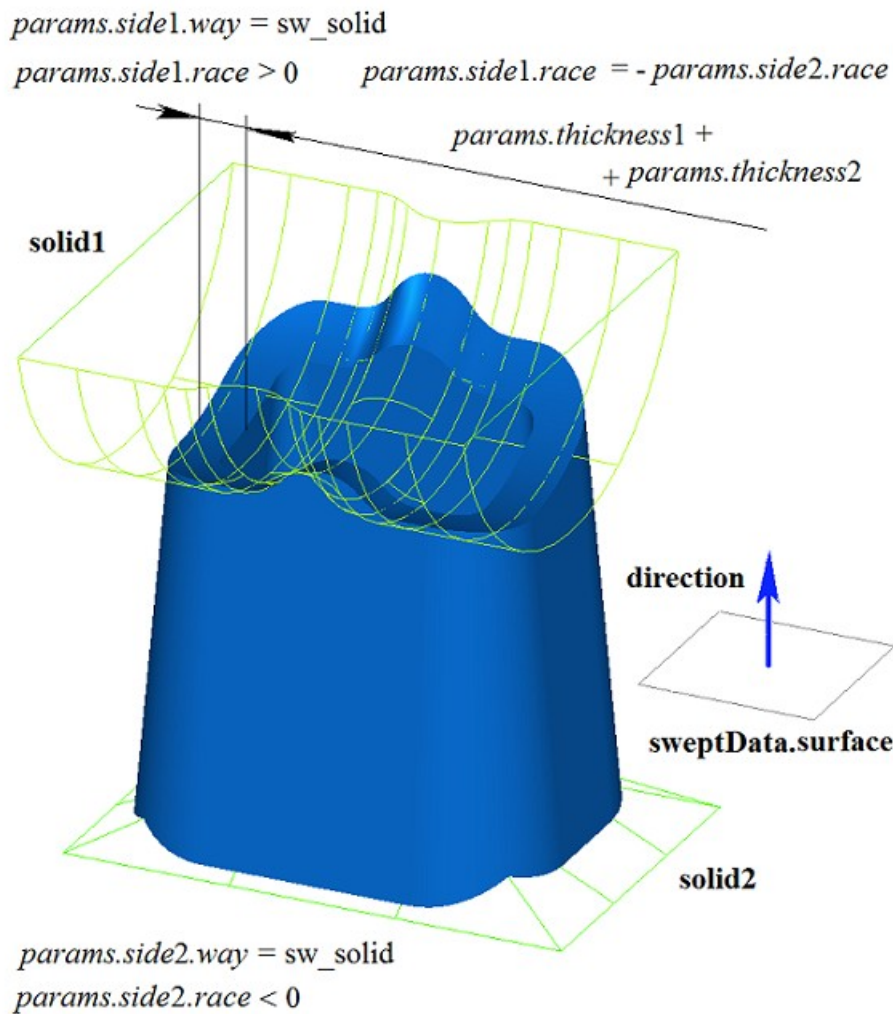


Рис. М.1.3.8.

На рис. М.1.3.9 приведен двумерный контур **contour**, плоская поверхность **surface** ([MbPlane](#)) и две поверхности **surface1** и **surface2**, которые будут использоваться при построении тела выдавливания. Для построения необходимо, чтобы поверхности **surface1** и **surface2** полностью перекрывали путь движения контура в соответствующем направлении. При этом следует учитывать параметры $params.side1.rake$, $params.side2.rake$, $params.thickness1$, $params.thickness2$. Тело выдавливания обрезается заданными поверхностями или эквидистантными к ним поверхностями, если $params.side1.distance$ или $params.side2.distance$ не равны нулю.

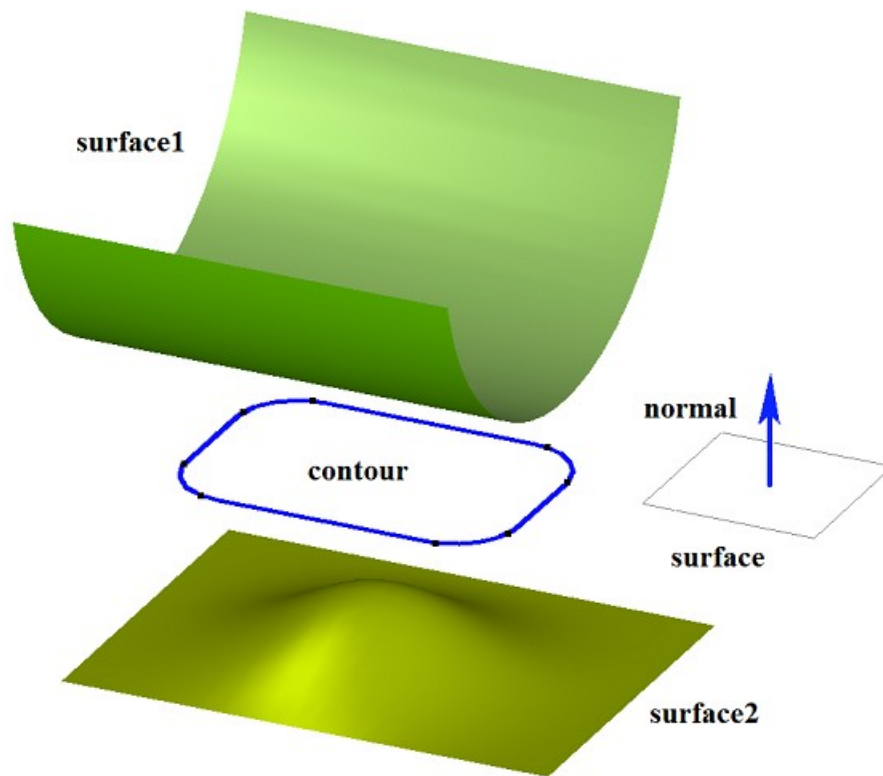


Рис. М.1.3.9.

На рис. М.1.3.10 приведено тело, построенное выдавливанием контура, показанного на рис. М.1.3.9, с опциями «До поверхности», в качестве которых заданы **surface1** и **surface2**.

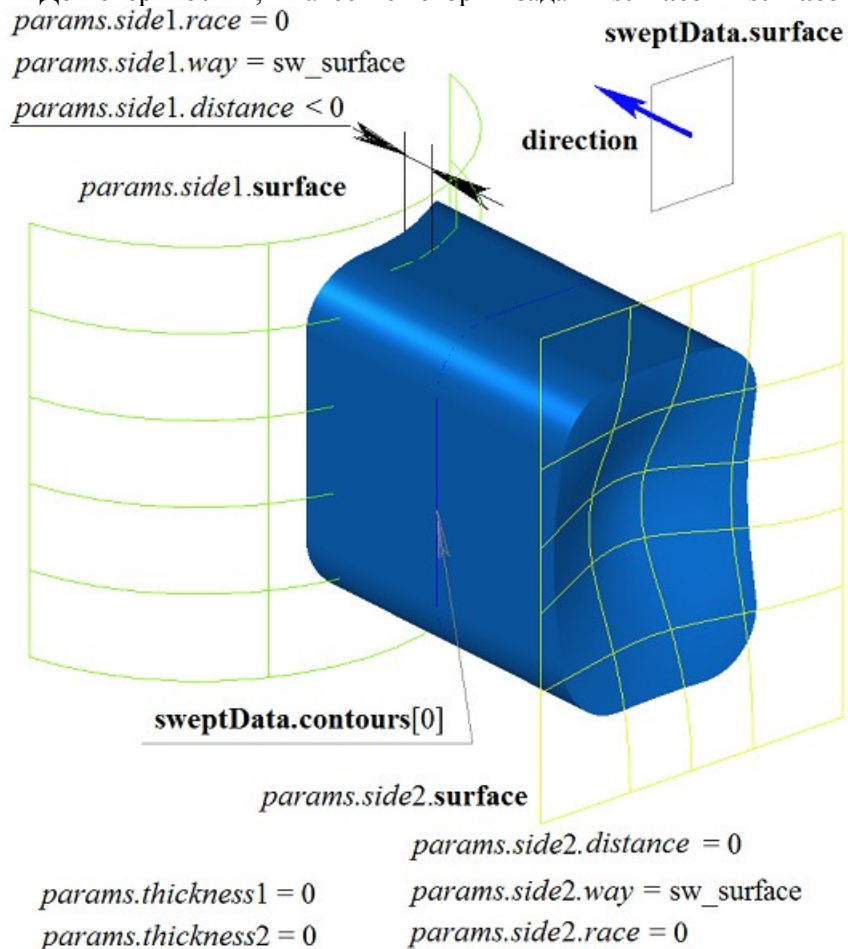


Рис. М.1.3.10.

На рис. М.1.3.11 приведено тонкостенное тело с уклоном грани, построенное выдавливанием контура, показанного на рис. М.1.3.9, с опциями «До поверхности», в качестве которых заданы **surface1** и **surface2**.

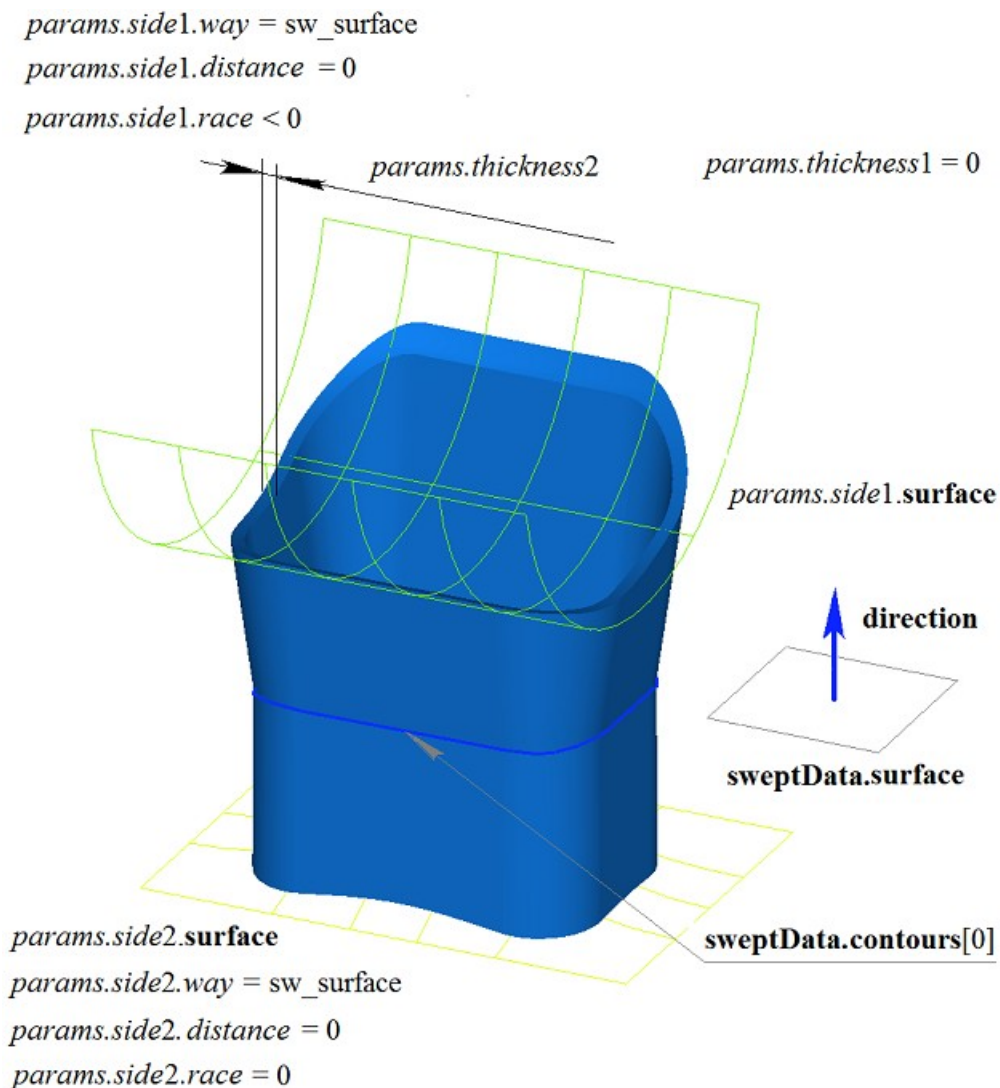


Рис. М.1.3.11.

Двумерный контур может располагаться на плоской или криволинейной поверхности. Например, тело можно построить выдавливанием контура на криволинейной поверхности, полученного от цикла одной из граней твердого тела, показанного на рис. М.1.3.12.

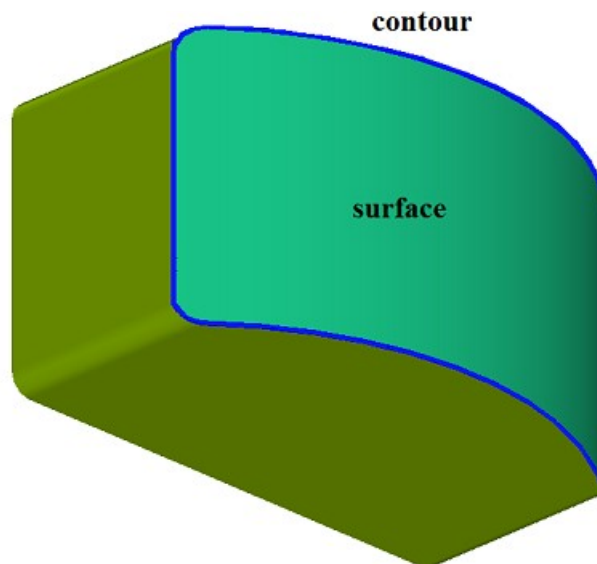


Рис. М.1.3.12.

На рис. М.1.3.13 приведено тело, полученное выдавливанием контура на криволинейной поверхности, приведенной на рис. М.1.3.12.

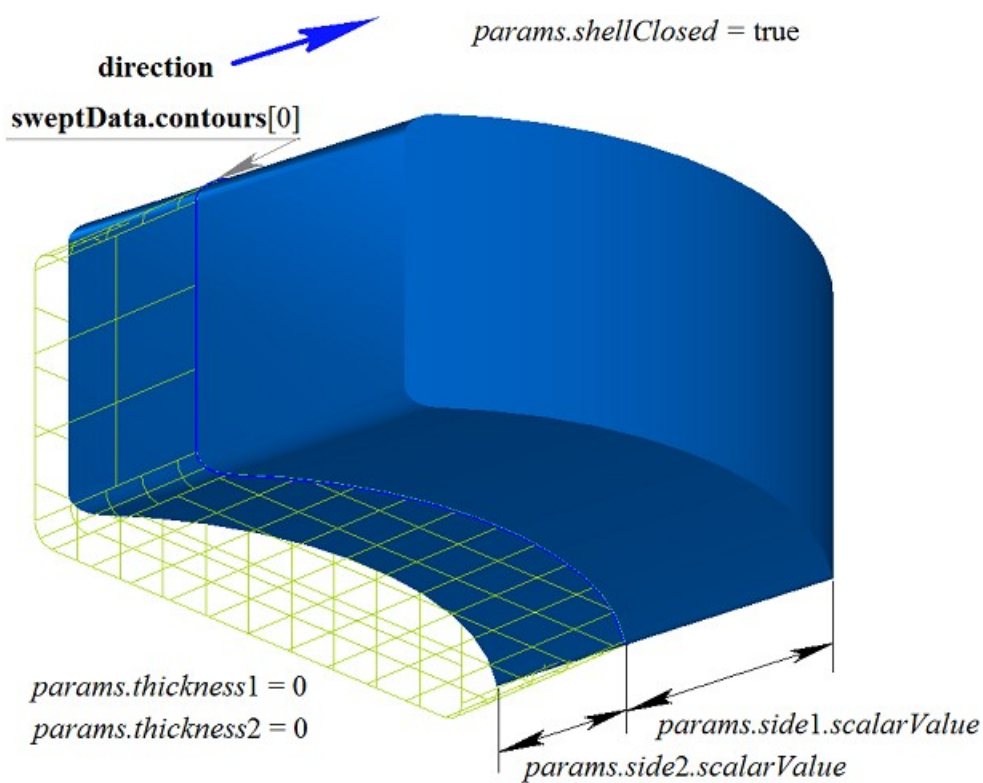


Рис. М.1.3.13.

На рис. М.1.3.14 приведено тонкостенное тело, полученное выдавливанием контура на криволинейной поверхности, приведенной на рис. М.1.3.12.

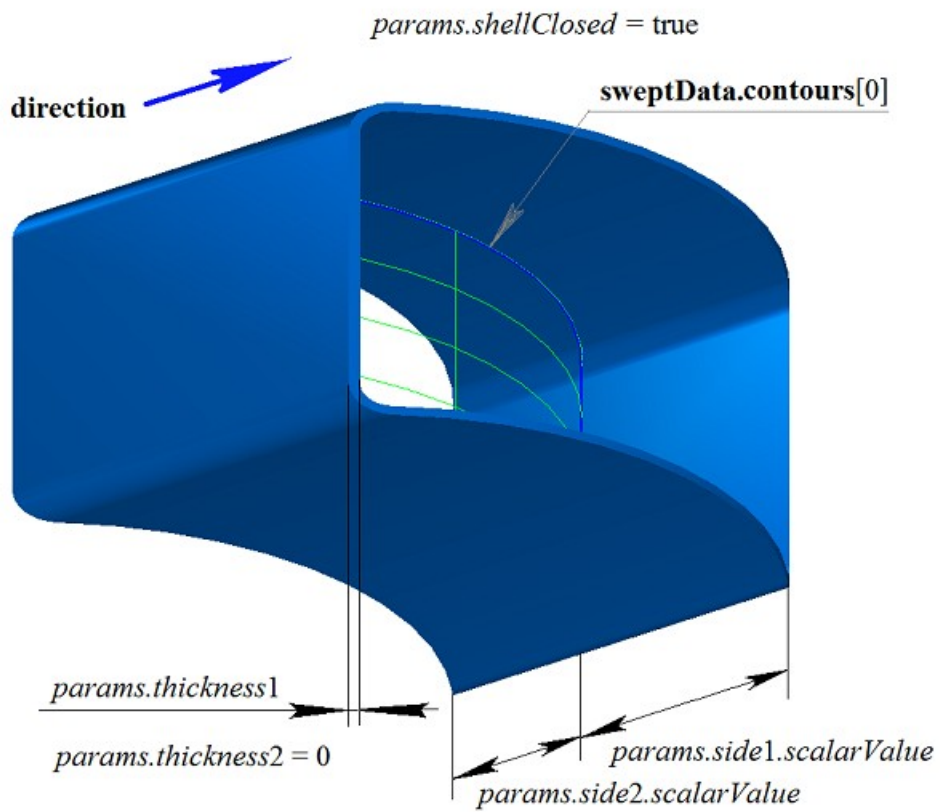


Рис. М.1.3.14.

На рис. М.1.3.15 приведено незамкнутое тело, полученное выдавливанием контура на криволинейной поверхности, приведенной на рис. М.1.3.12.

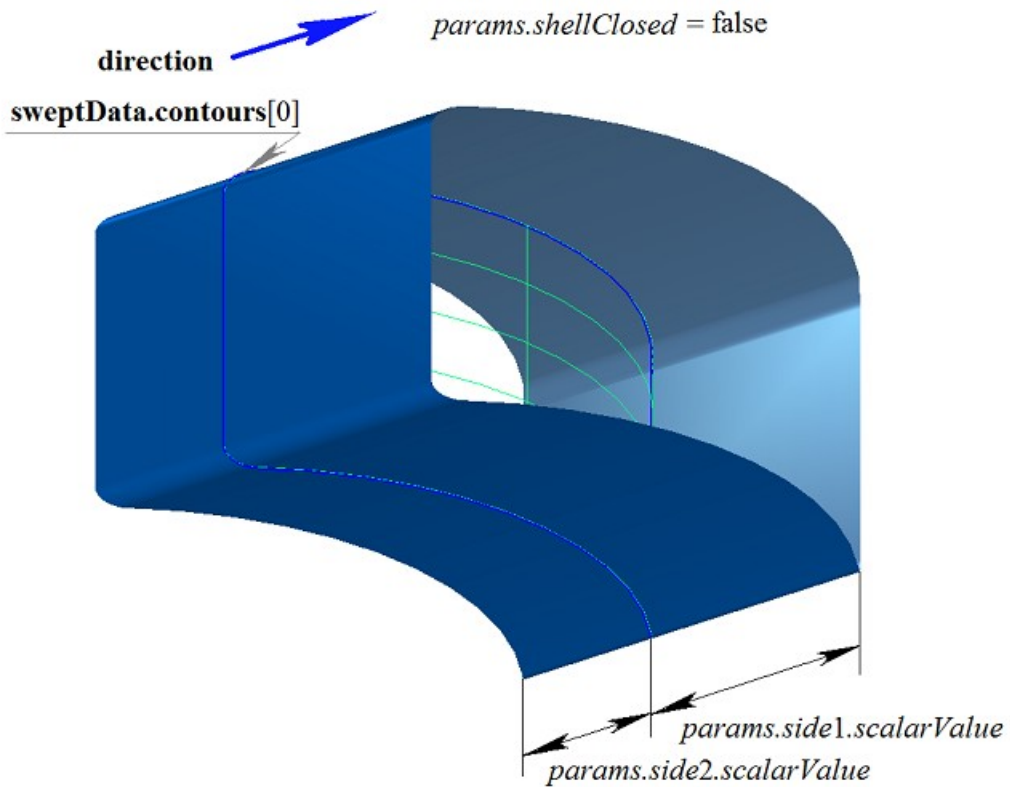


Рис. М.1.3.15.

Если на одной поверхности расположено множество не пересекающихся двумерных контуров, то рассматриваемый метод определяет внешние контуры и вложенные в них внутренние контуры, причем вложение может быть многократным. На рис. М.1.3.16 приведено множество не пересекающихся двумерных контуров **contours** и плоская поверхность **surface** (MbPlane).

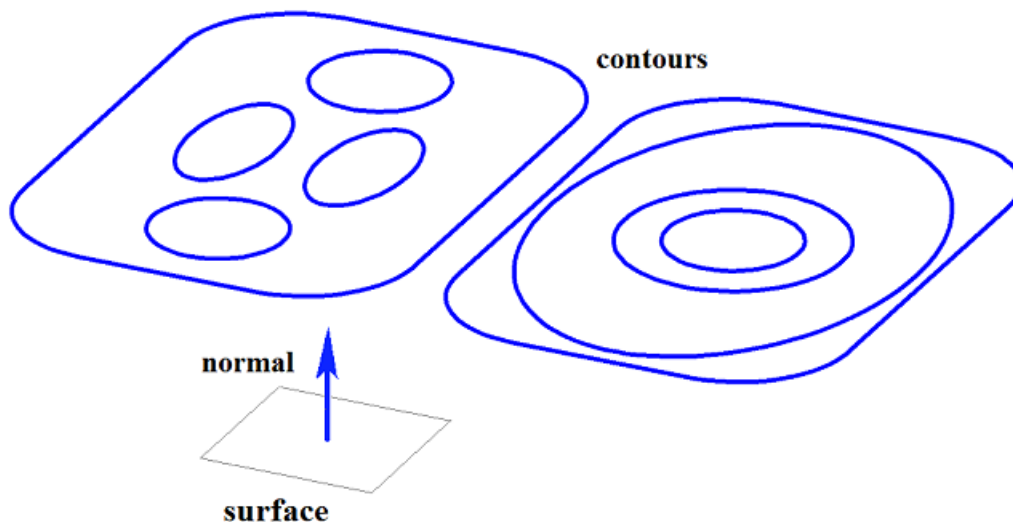


Рис. М.1.3.16.

На рис. М.1.3.17 приведено замкнутое тело, состоящее из нескольких частей, построенное выдавливанием множества контуров, показанного на рис. М.1.3.16.

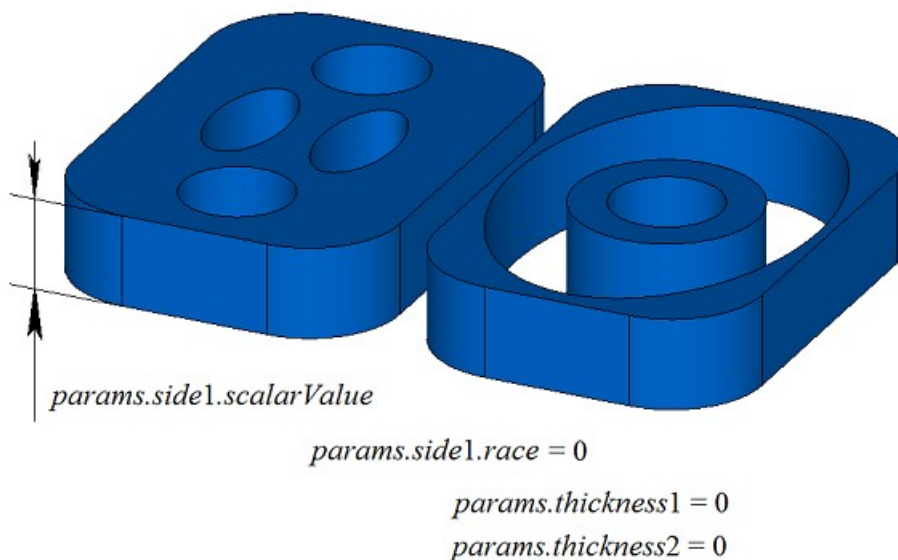


Рис. М.1.3.17.

На рис. М.1.3.18 приведено замкнутое тело, состоящее из нескольких частей, построенное выдавливанием с уклоном множества контуров, показанного на рис. М.1.3.16.

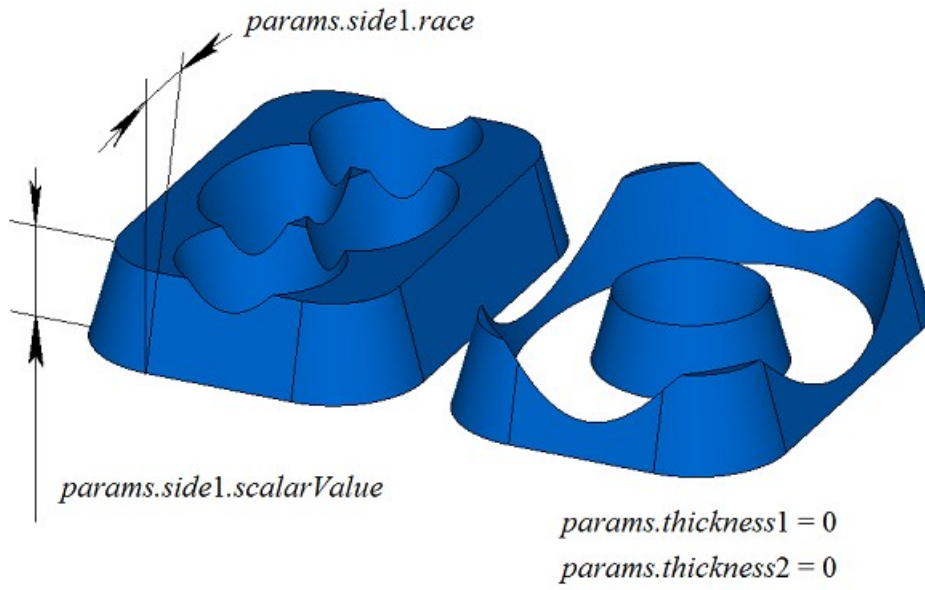


Рис. М.1.3.18.

На рис. М.1.3.19 приведено замкнутое тонкостенное тело, состоящее из нескольких частей, построенное выдавливанием множества контуров, показанного на рис. М.1.3.16.

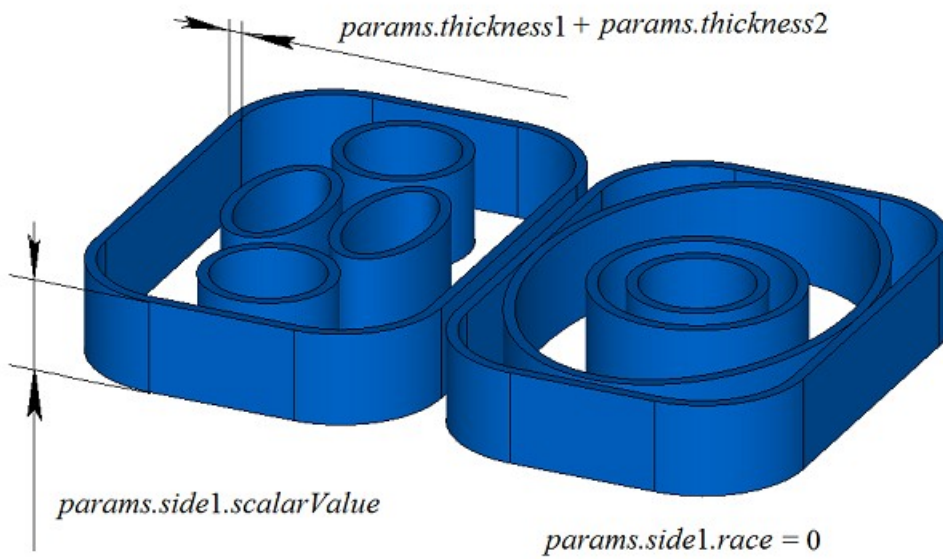


Рис. М.1.3.19.

На рис. М.1.3.20 приведены два трехмерных контура.

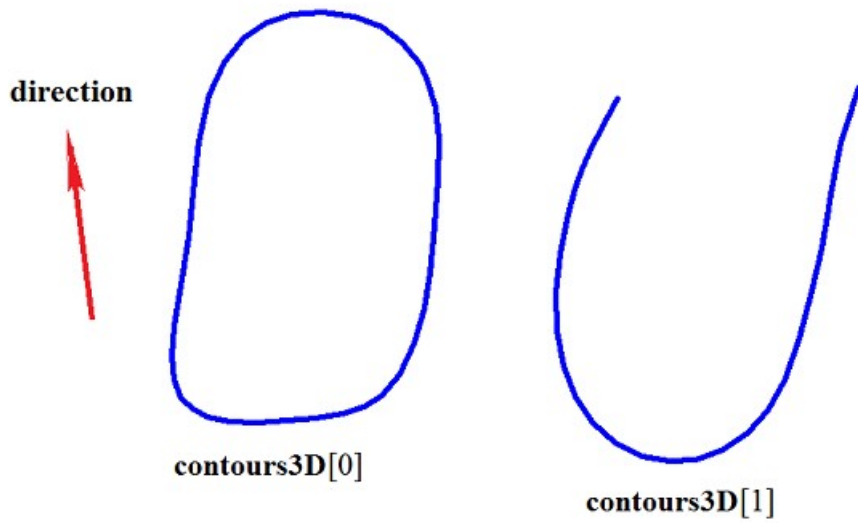


Рис. М.1.3.20.

На рис. М.1.3.21 приведено двусвязное тонкостенное замкнутое тело, полученное выдавливанием трехмерных контуров, приведенных на рис. М.1.3.20.

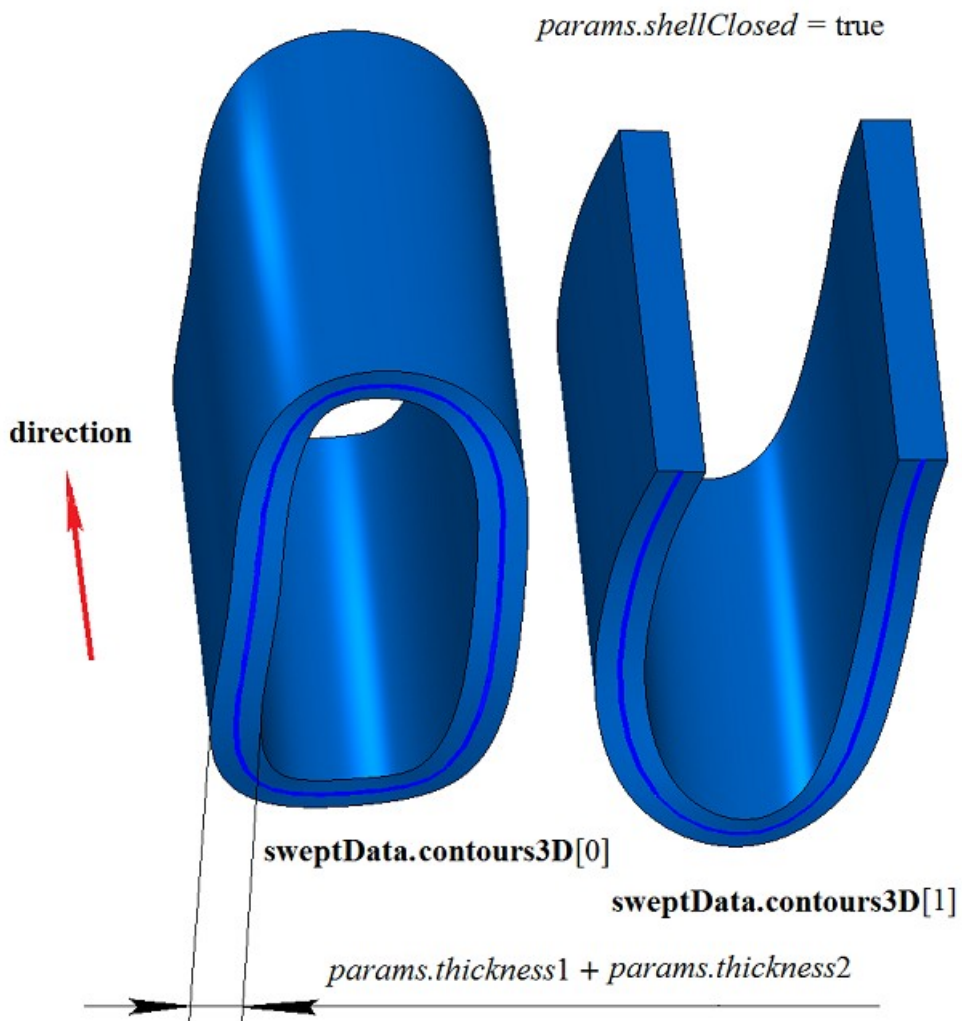


Рис. М.1.3.21.

На рис. М.1.3.22 приведены два незамкнутых тела, полученных выдавливанием трехмерных контуров, приведенных на рис. М.1.3.20. Построение выполнено для каждого контура отдельно.

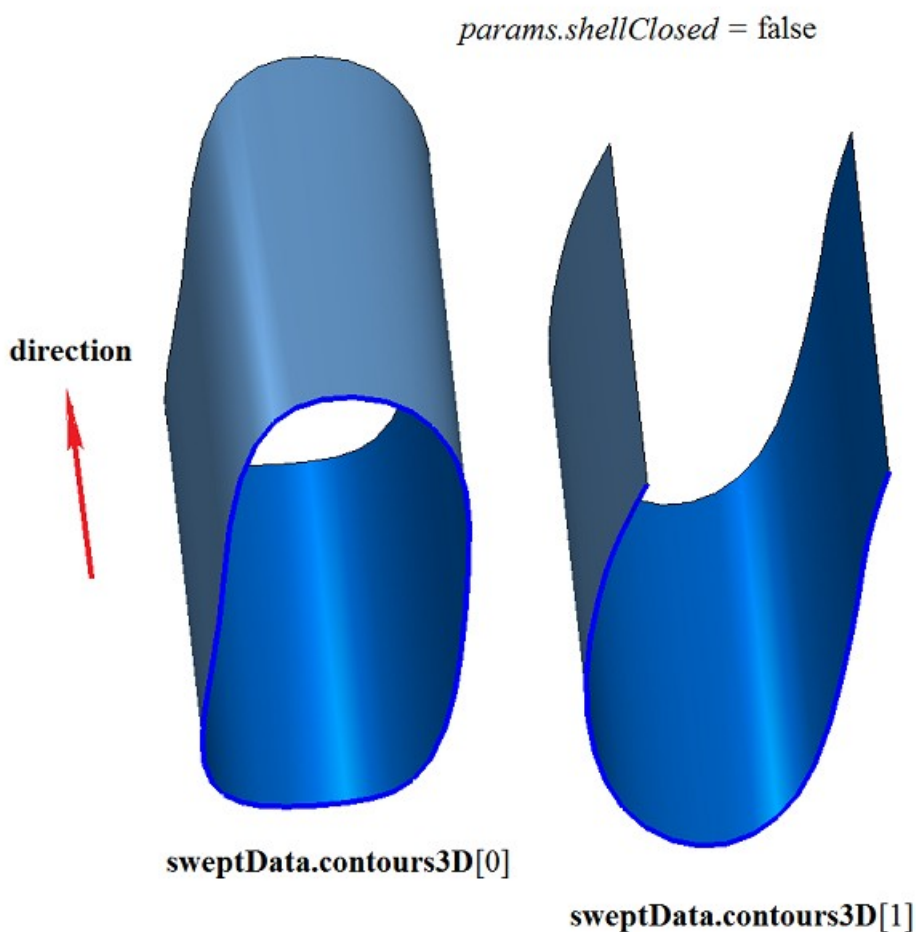


Рис. М.1.3.22.

Метод построения тела выдавливания **ExtrusionSolid** добавляет в журнал построенного тела строитель **MbExtrusionSolid**, который содержит все необходимые для построения тела данные. Строитель **MbExtrusionSolid** объявлен в файле `cg_extrusion_solid.h`.

Тестовое приложение `test.exe` выполняет построение тела выдавливания командами меню «Создать->Тело->На базе кривых->Выдавливанием поверхностной кривой» и «Создать->Тело->На базе кривых->Выдавливанием трехмерной кривой».

М.1.4. Построение тела вращения

Метод

MbResultType

RevolutionSolid (`const MbSweptData & sweptData,`
`const MbAxis3D & axis,`
`RevolutionValues & params,`
`const MbSNameMaker & names,`
`PArray<MbSNameMaker> & cnames,`
`MbSolid *& result`)

выполняет построение тела вращения.

Входными параметрами метода являются:

- **sweptData** – данные об образующих кривых,
- **axis** – ось вращения,
- **params** – параметры построения,

- `names` – именователь граней,
- `spnames` – именователи сегментов образующих кривых.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Тело вращения относится к разновидности тел движения, которые получают путем движения образующей кривой вдоль направляющей кривой. Направляющей кривой тела вращения служит окружность или ее дуга. Тело вращения строится путем вращения одной или нескольких кривых вокруг оси **axis**.

Параметр **sweptData** содержит информацию об образующих кривых. Класс `MbSweptData` и структура `RevolutionValues` описаны в файле `swept_parameter.h`. Образующие кривые могут представлять собой двумерные контуры **contours** на поверхности **surface** или контуры в пространстве **contours3D**. В частном случае двумерные контуры **contours** могут располагаться на плоскости. Ориентация контуров **contours** может быть произвольной. Контуры **contours** могут быть вложены друг в друга. Контуры **contours** не должны пересекать друг друга.

Вращение кривых может выполняться в прямом направлении относительно оси **axis**, в обратном направлении относительно оси **axis** и в обоих направлениях. Вращение в прямом направлении выполняется против часовой стрелки при взгляде навстречу оси **axis**. Параметры построения в каждом направлении задают объекты `MbSweptSide`.

Параметр *params* содержит информацию о способе вращения в прямом направлении `MbSweptSide side1` и информацию о способе вращения в обратном направлении `MbSweptSide side2`. Вращение в каждую сторону может выполняться двумя способами. При *way*==`sw_scalarValue` вращение выполняется на угол *scalarValue* в направлении *side1* или *side2*, соответственно. При *way*==`sw_surface` вращение выполняется до поверхности *side1.surface* или *side2.surface*, соответственно, при *side1.distance*=0 или *side2.distance*=0. При *way*==`sw_surface` и *distance*!=0 вращение выполняется до эквидистантной поверхности к *side1.surface* или до эквидистантной поверхности к *side2.surface*, соответственно. Параметры *params.thickness1* и *params.thickness2* определяют толщину стенки тонкостенного тела. Параметр *params.thickness1* задает отступ наружу от образующей кривой, а параметр *params.thickness2* задает отступ внутрь от образующей кривой. Параметр *params.shellClosed* управляет замкнутостью построенного тела. Параметр *params.checkSelfInt* сообщает о необходимости проверки результата построения на самопересечение. По умолчанию *params.checkSelfInt*=false, проверка не выполняется. Параметр *params.shape* управляет формой построенного тела. При *params.shape*=1 построенное тело имеет топологию тора, при *params.shape*=0 построенное тело имеет топологию сферы.

На рис. М.1.4.1 приведены данные, используемые при построении, и схема наследования параметров построения тела вращения `RevolutionValues & params`.

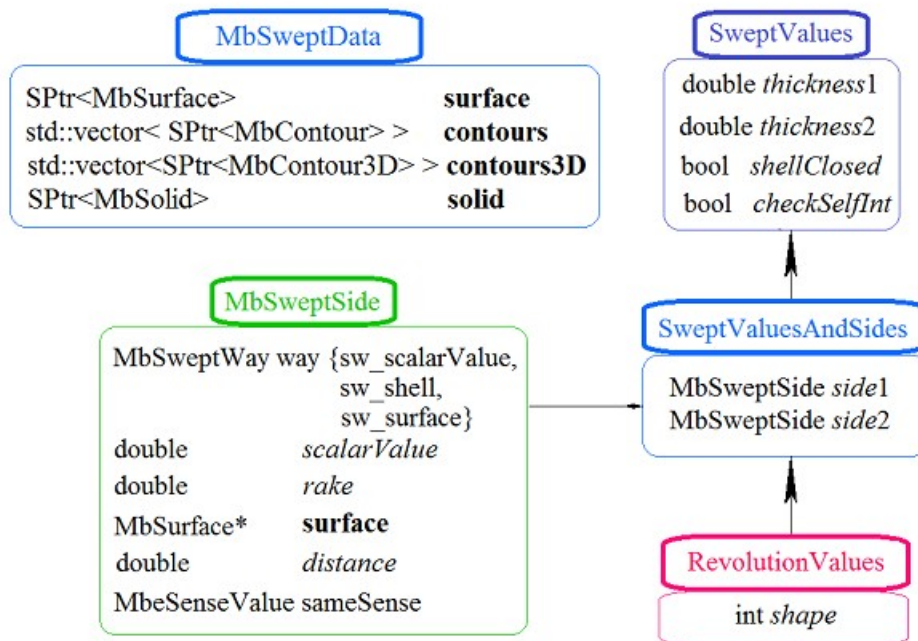


Рис. М.1.4.1.

Параметры *names* и *snames* обеспечивают именование граней построенного тела.

На рис. М.1.4.2 приведен двумерный контур **contour**, плоская поверхность **surface** ([MbPlane](#)) и ось вращения **axis**.

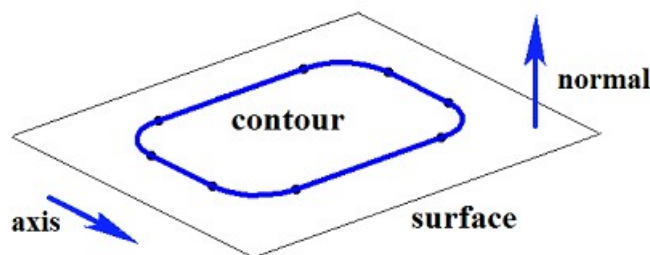


Рис. М.1.4.2.

На рис. М.1.4.3 приведено замкнутое тело, построенное вращением по заданным параметрам контура, показанного на рис. М.1.4.2. Каждому сегменту контура соответствует грань тела, имя которой взято от соответствующего элемента генератора имен *snames*[0].

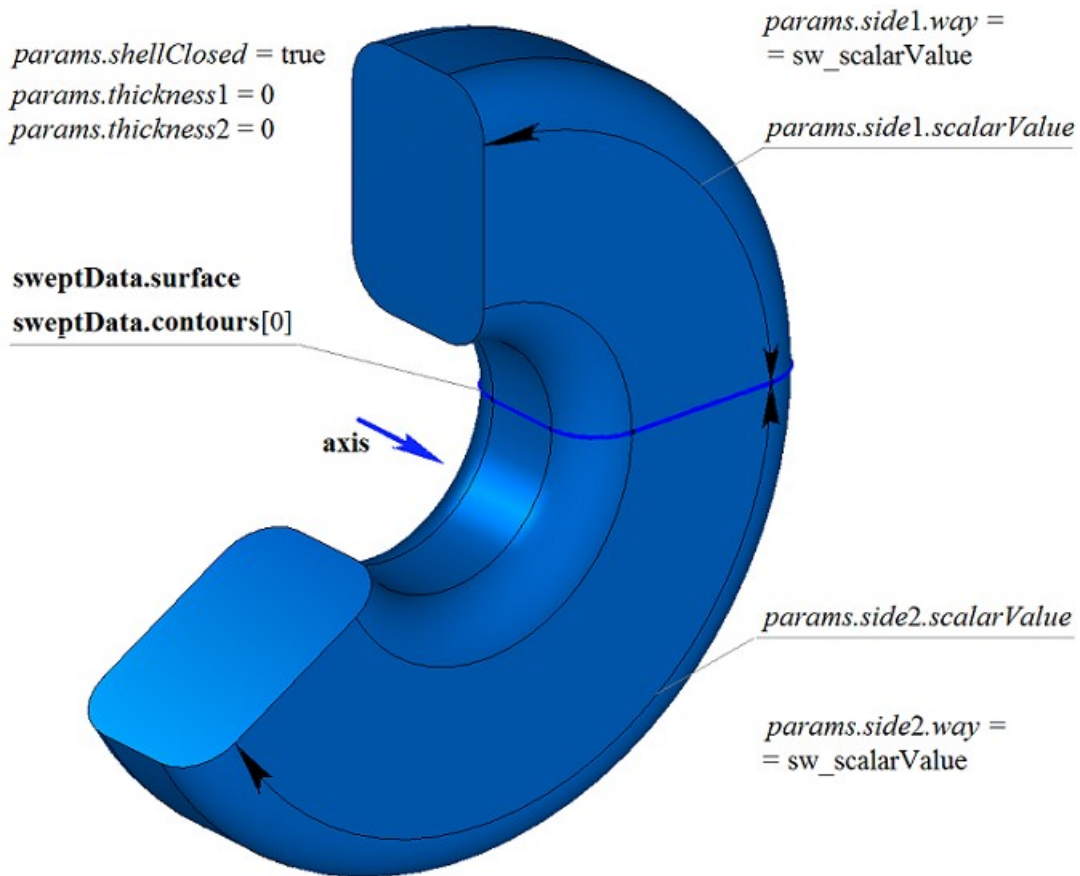


Рис. М.1.4.3.

На рис. М.1.4.4 приведено замкнутое тонкостенное тело, построенное вращением по заданным параметрам контура, показанного на рис. М.1.4.2.

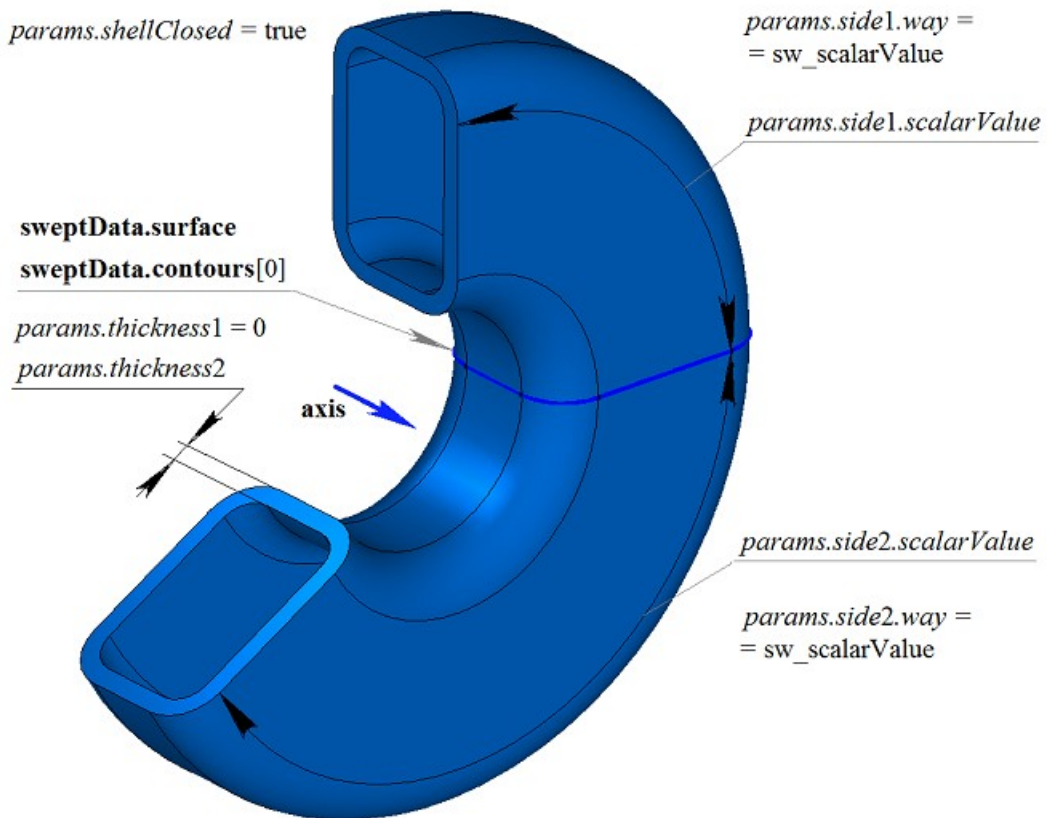


Рис. М.1.4.4.

На рис. М.1.4.5 приведено незамкнутое тело, построенное вращением по заданным параметрам контура, показанного на рис. М.1.4.2. Параметры построения тела, показанного на рис. М.1.4.3, отличаются от параметров построения тела, показанного на рис. М.1.4.5, только величиной *params.shellClosed*.

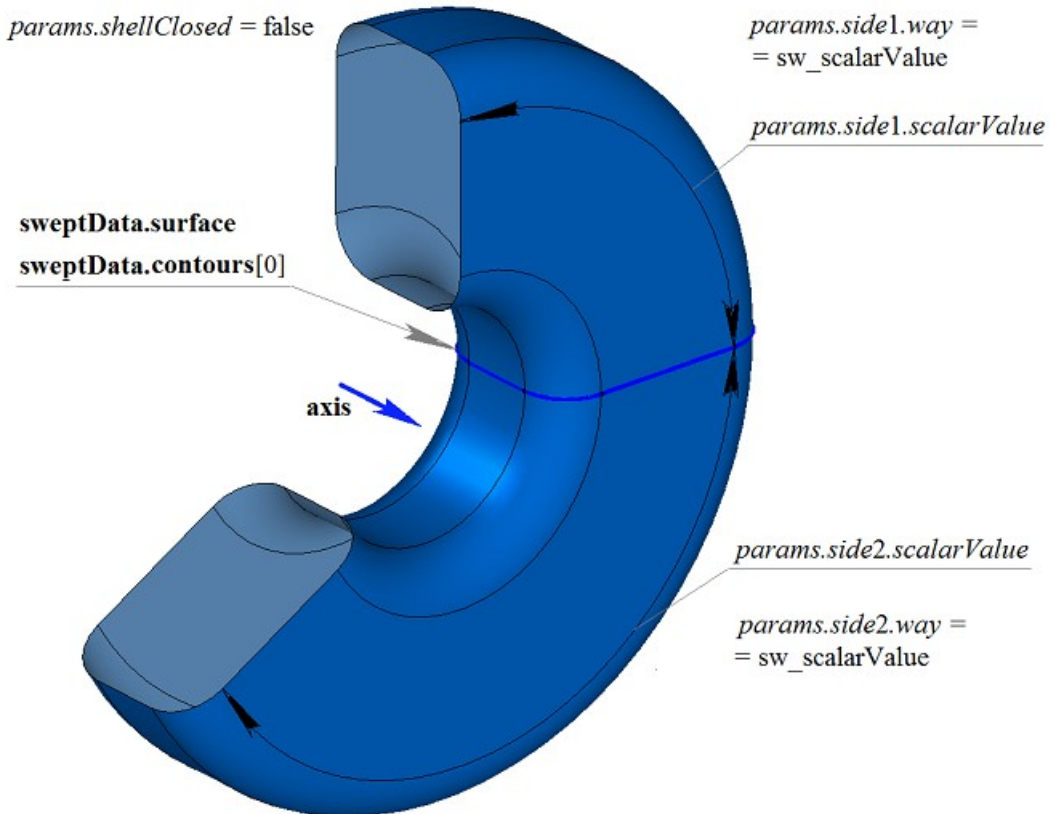


Рис. М.1.4.5.

На рис. М.1.4.6 приведен двумерный контур **contour**, плоская поверхность **surface** ([MbPlane](#)) и две поверхности **surface1** и **surface2**, которые будут использоваться при построении тела вращения. Для построения необходимо, чтобы поверхности **surface1** и **surface2** полностью перекрывали путь движения контура в соответствующем направлении. При этом следует учитывать параметры *params.thickness1*, *params.thickness2*. Тело вращения обрезается заданными поверхностями или эквидистантными к ним поверхностями, если *params.side1.distance* или *params.side2.distance* не равны нулю.

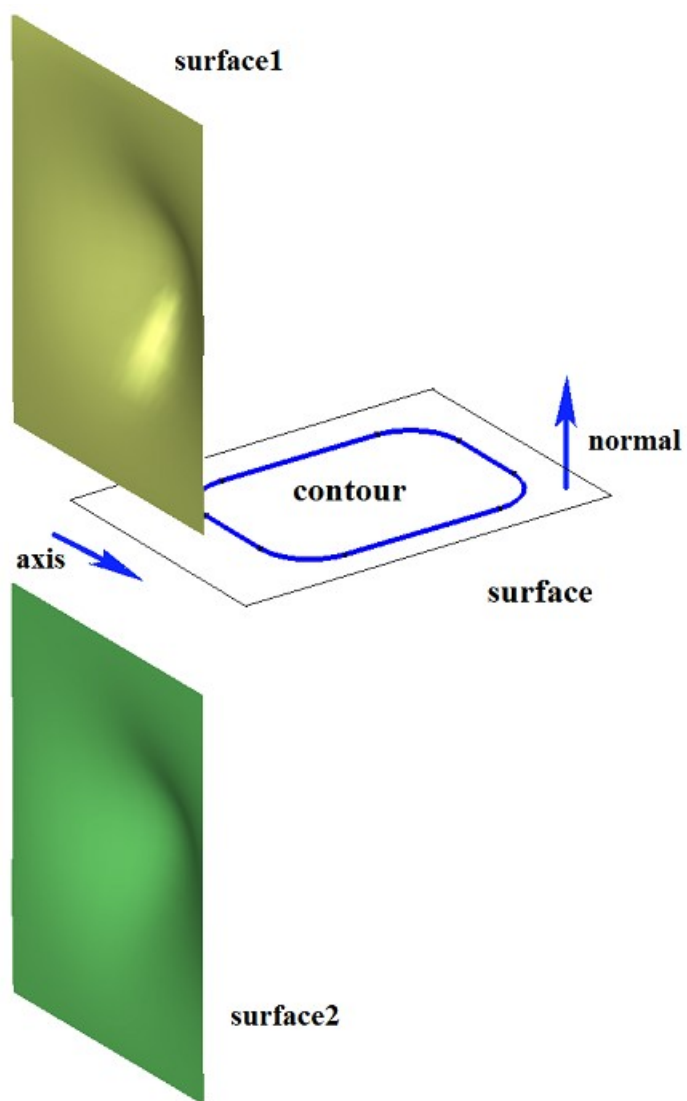


Рис. М.1.4.6.

На рис. М.1.4.7 приведено тело, построенное вращением контура, показанного на рис. М.1.4.6, с опциями «До поверхности», в качестве которых заданы **surface1** и **surface2**.

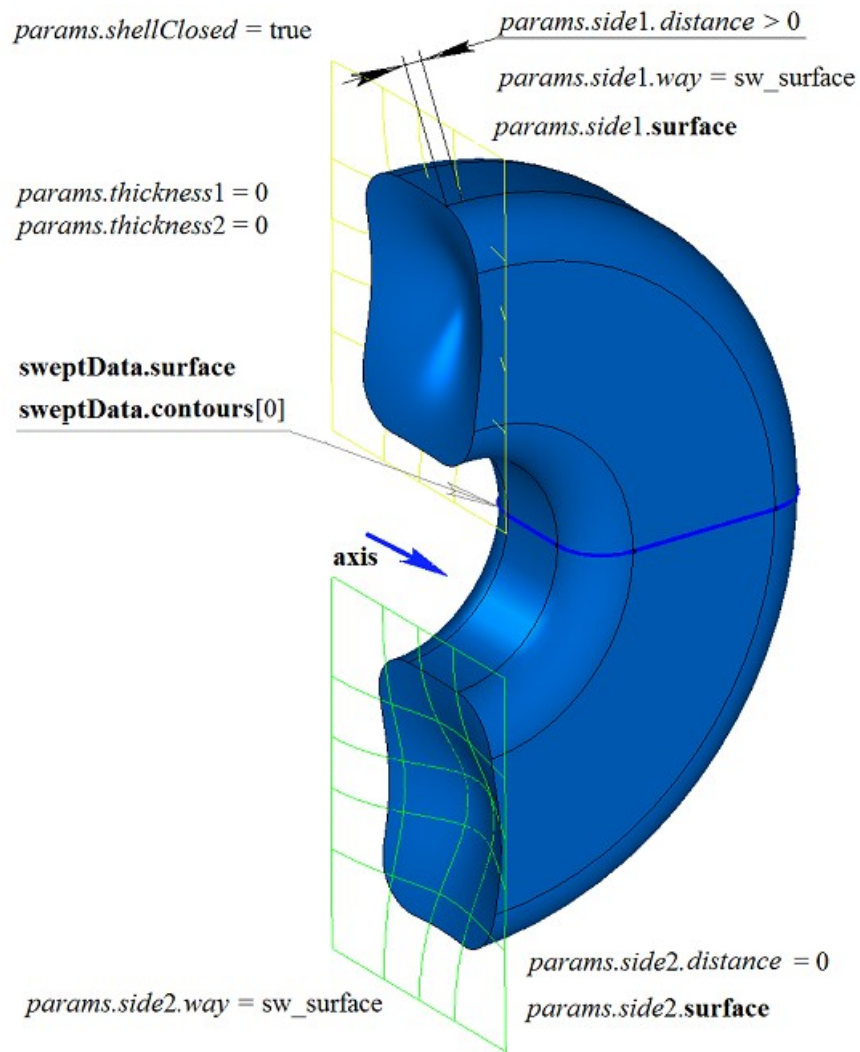


Рис. М.1.4.7.

На рис. М.1.4.8 приведено тонкостенное тело, построенное вращением контура, показанного на рис. М.1.4.6, с опциями «До поверхности», в качестве которых заданы **surface1** и **surface2**.

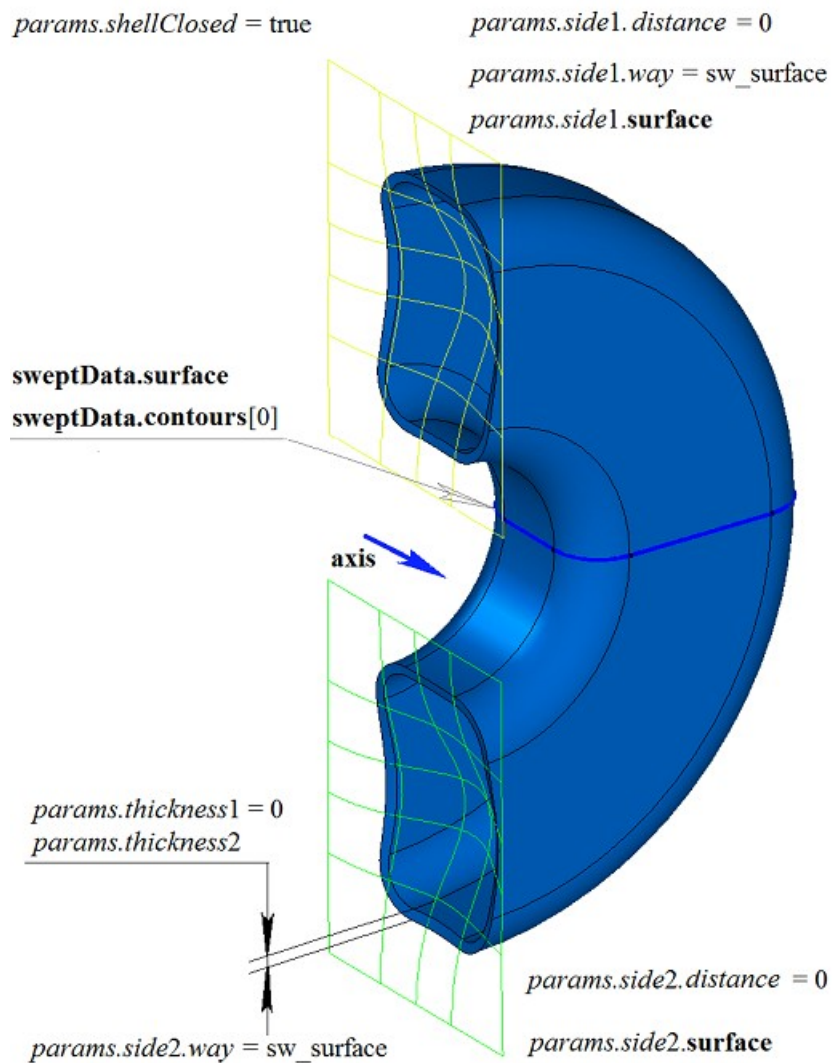


Рис. М.1.4.8.

Двумерный контур может располагаться на плоской или криволинейной поверхности. Например, тело можно построить вращением контура на криволинейной поверхности, полученного от цикла одной из граней твердого тела, показанного на рис. М.1.4.9.

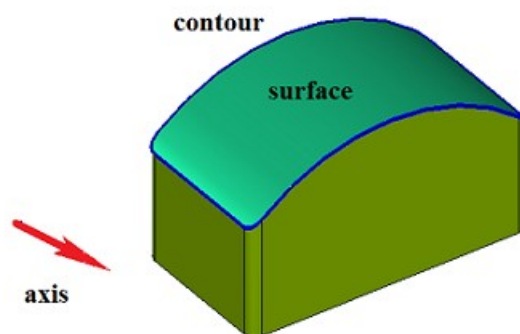


Рис. М.1.4.9

На рис. М.1.4.10 приведено тело, полученное вращением контура на криволинейной поверхности, приведенной на рис. М.1.4.9.

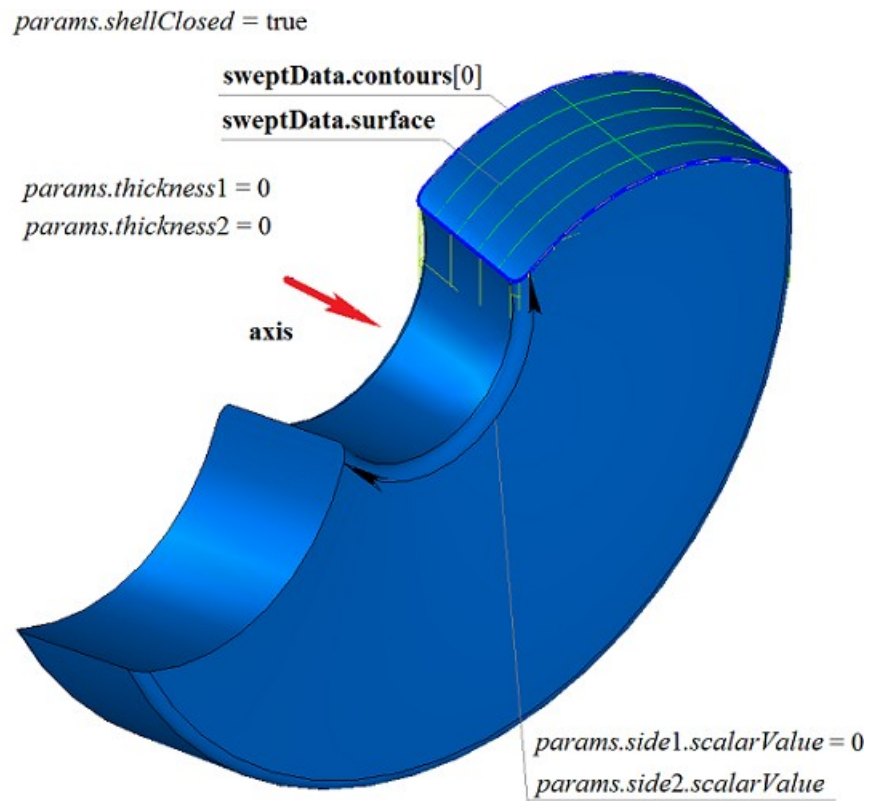


Рис. М.1.4.10.

На рис. М.1.4.11 приведено тонкостенное тело, полученное вращением контура на криволинейной поверхности, приведенной на рис. М.1.4.9.

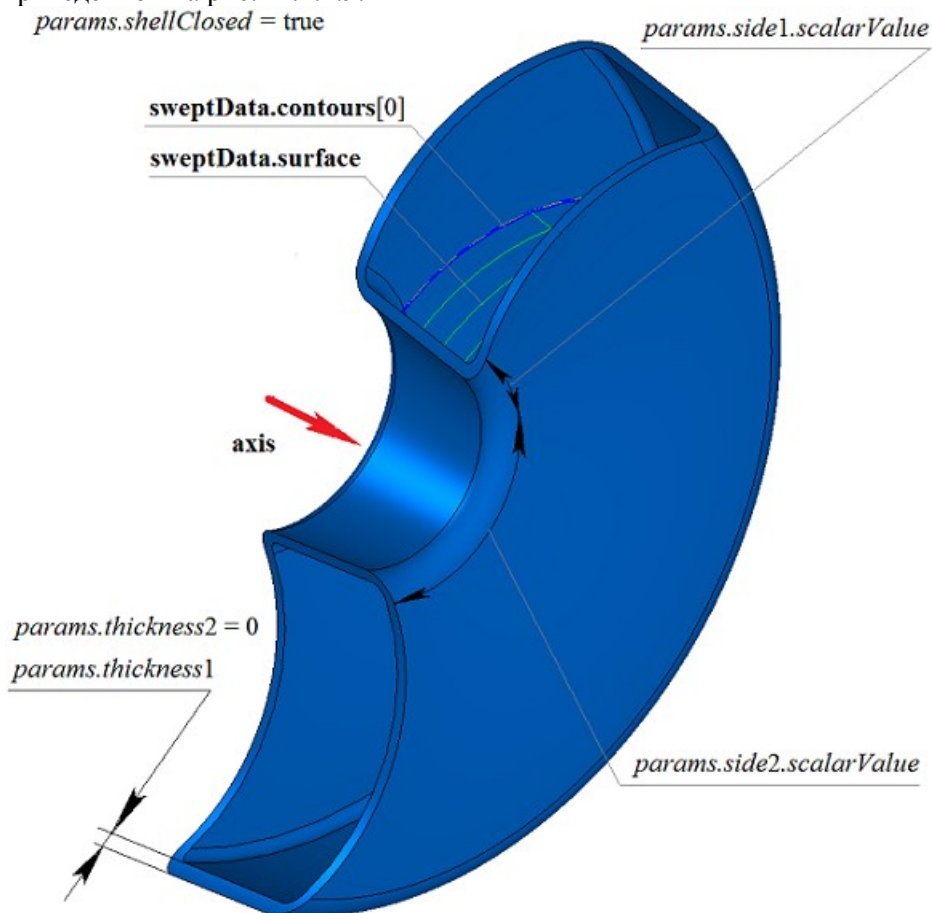


Рис. М.1.4.11.

На рис. М.1.4.12 приведено незамкнутое тело, полученное вращением контура на криволинейной поверхности, приведенной на рис. М.1.4.9.

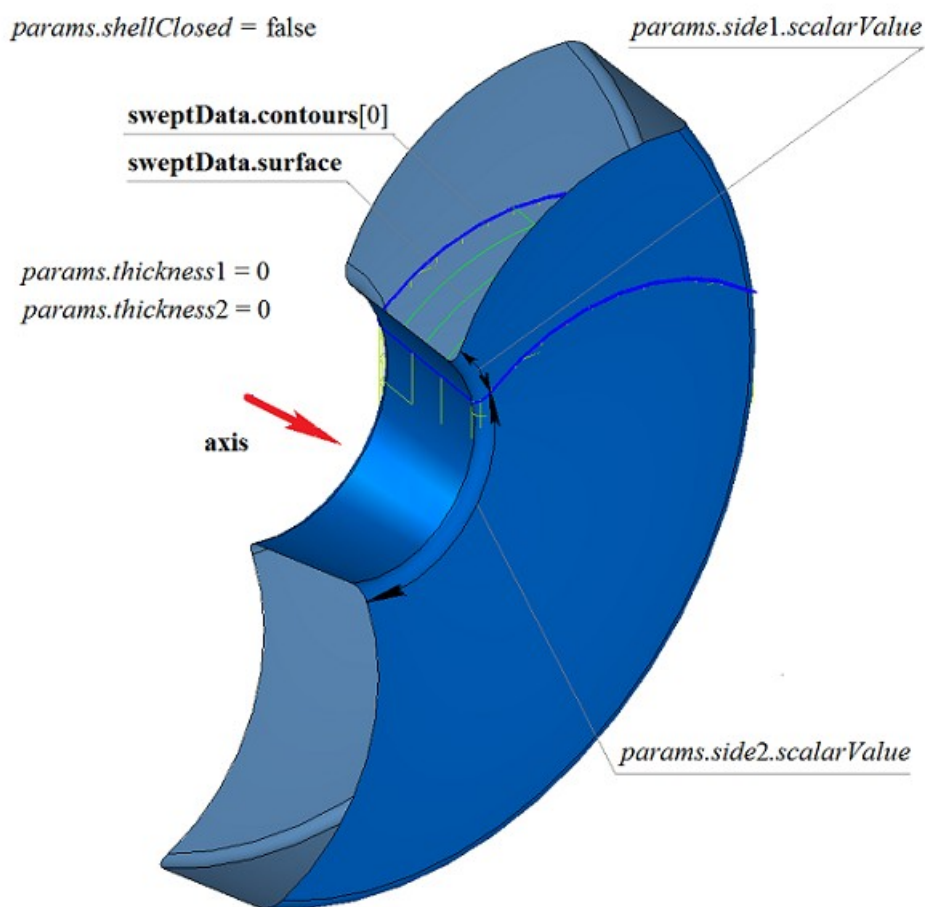


Рис. М.1.4.12.

Если на одной поверхности расположено множество не пересекающихся двумерных контуров, то рассматриваемый метод определяет внешние контуры и вложенные в них внутренние контуры, причем вложение может быть многократным. На рис. М.1.4.13 приведено множество не пересекающихся двумерных контуров **contours** и плоская поверхность **surface** ([MbPlane](#)).

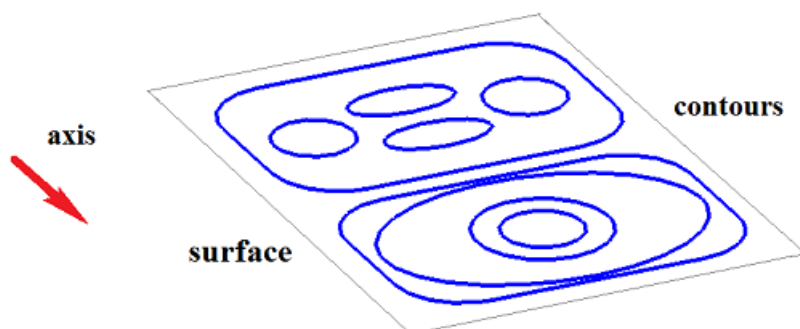


Рис. М.1.4.13.

На рис. М.1.4.14 приведено замкнутое тело, состоящее из нескольких частей, построенное вращением множества контуров, показанного на рис. М.1.4.13.

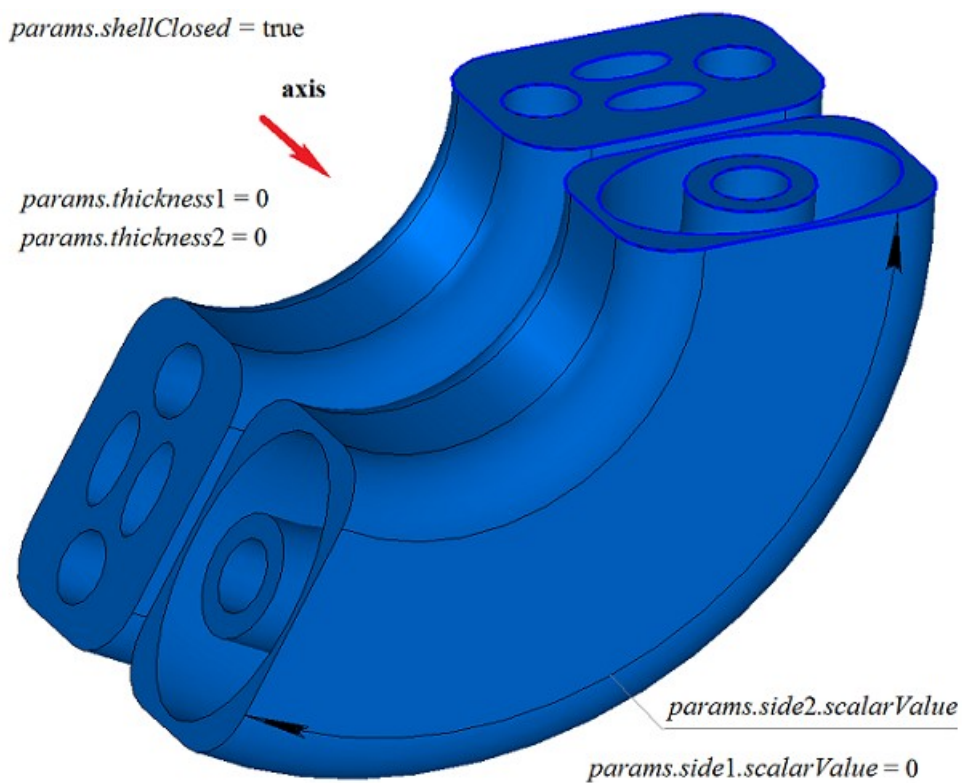


Рис. М.1.4.14.

На рис. М.1.4.15 приведено замкнутое тонкостенное тело, состоящее из нескольких частей, построенное вращением множества контуров, показанного на рис. М.1.4.13.

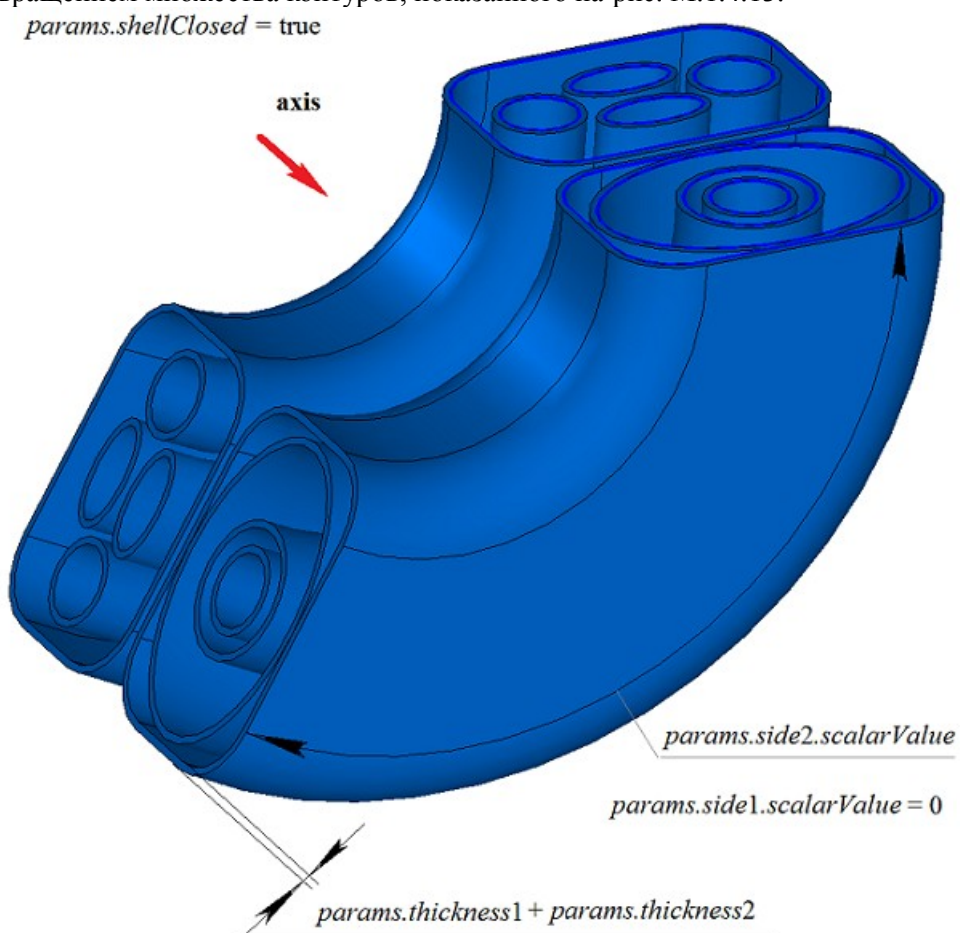


Рис. М.1.4.15.

На рис. М.1.4.16 приведены два трехмерных контура.

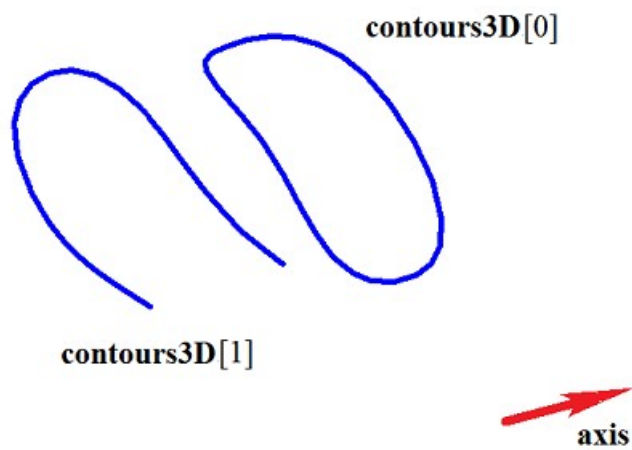


Рис. М.1.4.16.

На рис. М.1.4.17 приведено двусвязное тонкостенное замкнутое тело, полученное вращением трехмерных контуров, приведенных на рис. М.1.4.16.

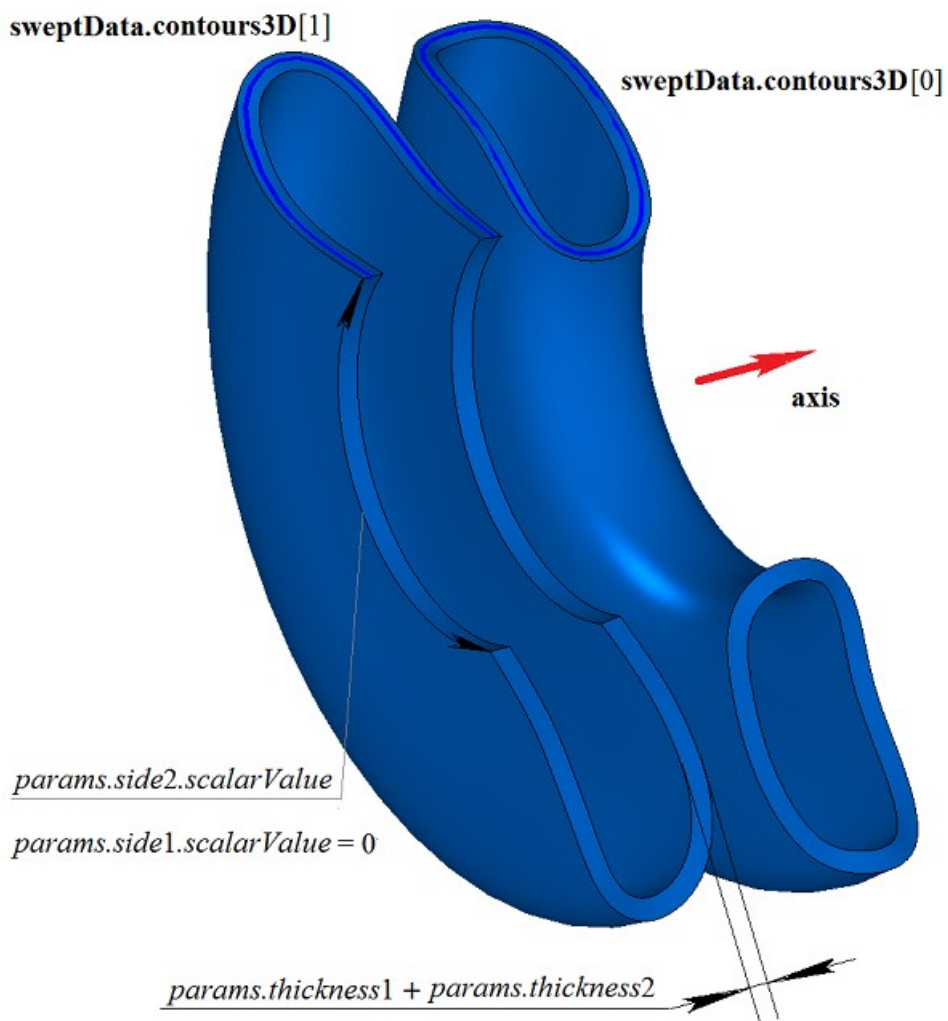


Рис. М.1.4.17.

На рис. М.1.4.18 приведены два незамкнутых тела, полученных вращением трехмерных контуров, приведенных на рис. М.1.4.16.

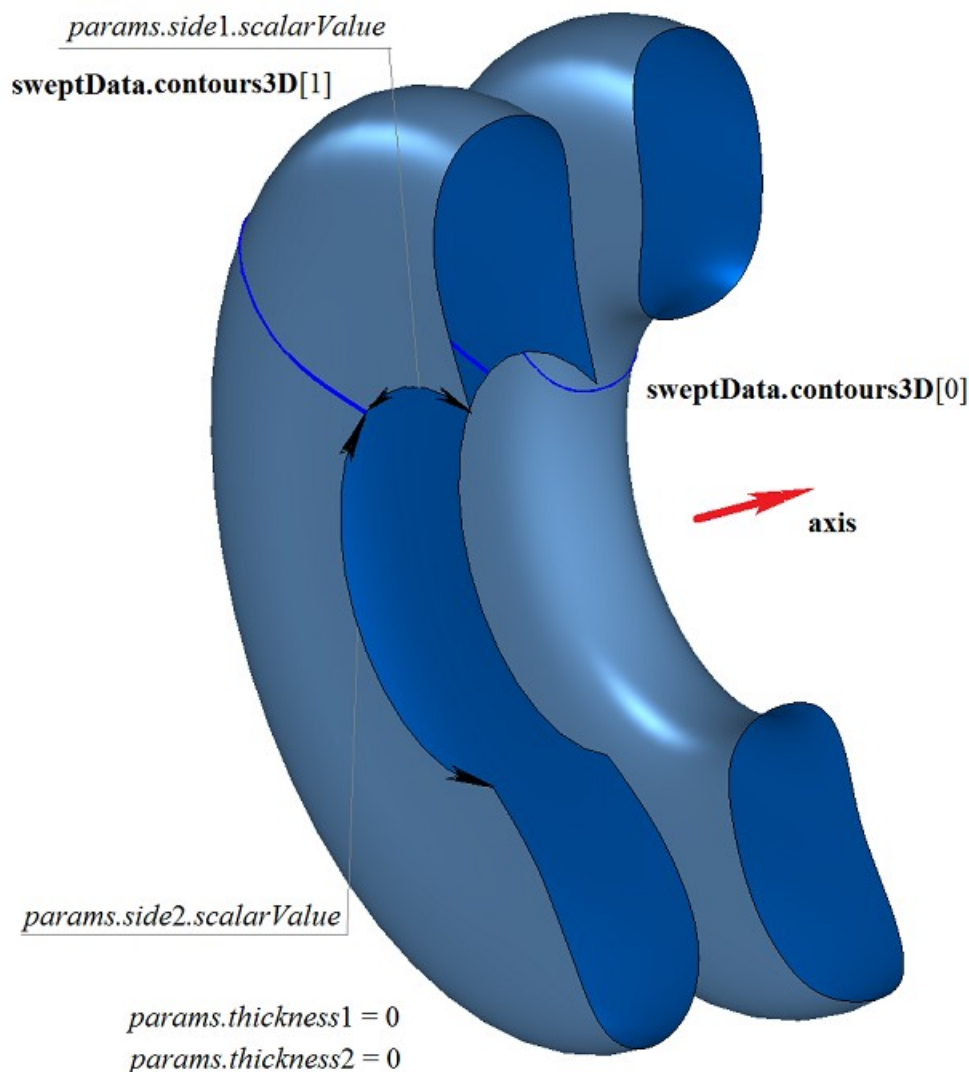


Рис. М.1.4.18.

Метод построения тела вращения **RevolutionSolid** добавляет в журнал построенного тела строитель **MbRevolutionSolid**, который содержит все необходимые для построения тела данные. Строитель **MbRevolutionSolid** объявлен в файле `cr_revolution_solid.h`.

Тестовое приложение `test.exe` выполняет построение тела вращения командами меню «Создать->Тело->На базе кривых->Вращением поверхностной кривой» и «Создать->Тело->На базе кривых->Вращением трехмерной кривой».

М.1.5. Построение тела заметания

Метод
MbResultType
EvolutionSolid (`const MbSweptData & sweptData,`
`const MbCurve3D & spine,`
`EvolutionValues & params,`
`const MbSNameMaker & names,`
`const MbSNameMaker & cnames,`
`const MbSNameMaker & snames,`
`MbSolid *& result`)

выполняет построение тела заметания путем движения образующей кривой вдоль направляющей кривой.

Входными параметрами метода являются:

- **place** – локальная система координат образующего контура,
- **contour** – образующий контур,
- **spine** – направляющая кривая,
- *params* – параметры построения,
- *names* – именователъ граней,
- *snames* – именователъ образующей,
- *snames* – именователъ направляющей.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Тело заметания представляет собой общий случай тела движения, которые получают путем движения образующей кривой вдоль направляющей кривой. Направляющей кривой тела заметания служит произвольная кривая.

Параметр **sweptData** содержит информацию об образующих кривых. Класс `MbSweptData` и структура `EvolutionValues` описаны в файле `swept_parameter.h`. Образующие кривые могут представлять собой двумерные контуры **contours** на поверхности **surface** или контуры в пространстве **contours3D**. В частном случае двумерные контуры **contours** могут располагаться на плоскости. Ориентация контуров **contours** может быть произвольной. Контуры **contours** могут быть вложены друг в друга. Контуры **contours** не должны пересекать друг друга.

Движение образующих кривых выполняется вдоль направляющей кривой **spine**. Параметр *params* содержит информацию о способе движения, наличии и толщине стенок тела, замкнутости построенного тела. Параметры *params.thickness1* и *params.thickness2* определяют толщину стенки построенного тонкостенного тела. Параметр *params.thickness1* задает отступ наружу от образующей кривой, а параметр *params.thickness2* задает отступ внутрь от образующей кривой. Параметр *params.shellClosed* управляет замкнутостью построенного тела. Параметр *params.checkSelfInt* сообщает о необходимости проверки результата построения на самопересечение. По умолчанию *params.checkSelfInt=false*, проверка не выполняется, и метод допускает построение самопересекающегося тела. Движение может выполняться тремя способами. Способом движения управляет параметр *params.parallel*. При *params.parallel=0* образующие кривые движутся плоскопараллельно, при *params.parallel=1* при движении образующие кривые сохраняют свое положение в локальной системе координат, касательной к образующей кривой, при *params.parallel=2* до начала движения образующие кривые переносятся в плоскость, перпендикулярную направляющей кривой в ее начале, а далее движению сохраняют свое положение в локальной системе координат, касательной к образующей кривой.

На рис. М.1.5.1 приведены данные, используемые при построении, и схема наследования параметров построения тела заметания `EvolutionValues` & *params*.

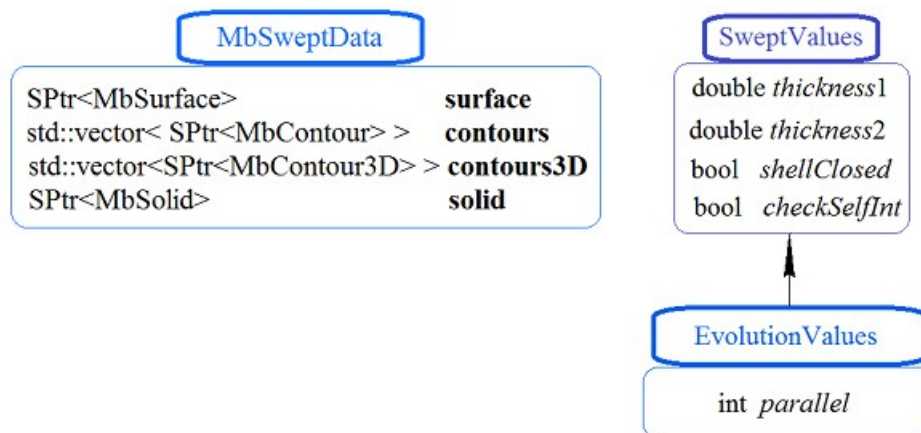


Рис. М.1.5.1.

Параметры *names*, *snames* и *snames* обеспечивают именованье граней построенного тела.

На рис. М.1.5.2 приведен двумерный контур **contour**, плоская поверхность **surface** ([MbPlane](#)) и направляющая кривая **spine**.

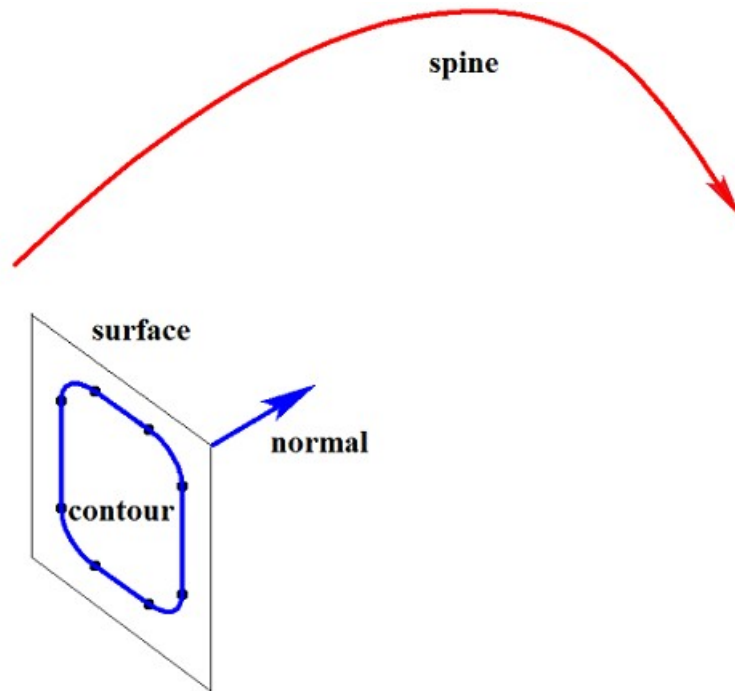


Рис. М.1.5.2.

На рис. М.1.5.3 приведено тело заметания, построенное движением контура вдоль направляющей, показанного на рис. М.1.5.2, способом, определяемым параметром *params.parallel=0*, при котором плоскости торцев тела сохраняют параллельность.

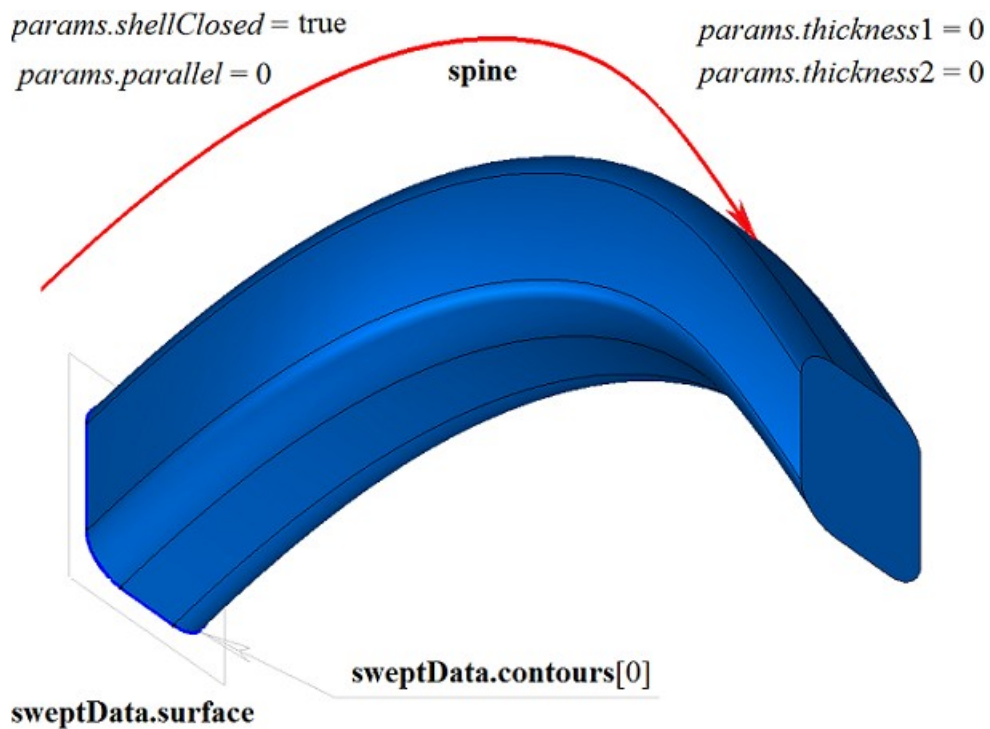


Рис. М.1.5.3.

На рис. М.1.5.4 приведено тело заметания, построенное движением контура вдоль направляющей, показанного на рис. М.1.5.2, способом, определяемым параметром *params.parallel=1*, при котором

плоскость конечного торца тела сохраняет положение относительно конца направляющей таким же, какое положение имеет плоскость начального торца относительно начала направляющей кривой.

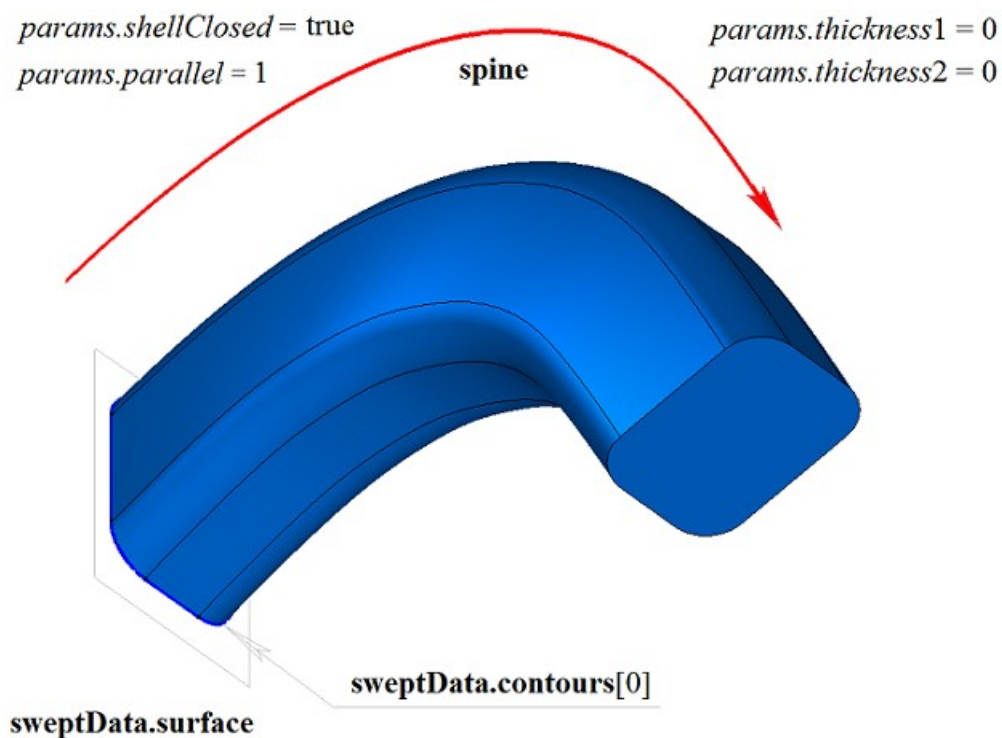


Рис. М.1.5.4.

На рис. М.1.5.5 приведено тело заметания, построенное движением контура вдоль направляющей, показанного на рис. М.1.5.2, способом, определяемым параметром $params.parallel=2$, при котором плоскости торцев тела сохраняют перпендикулярность направляющей в начале и конце кривой.

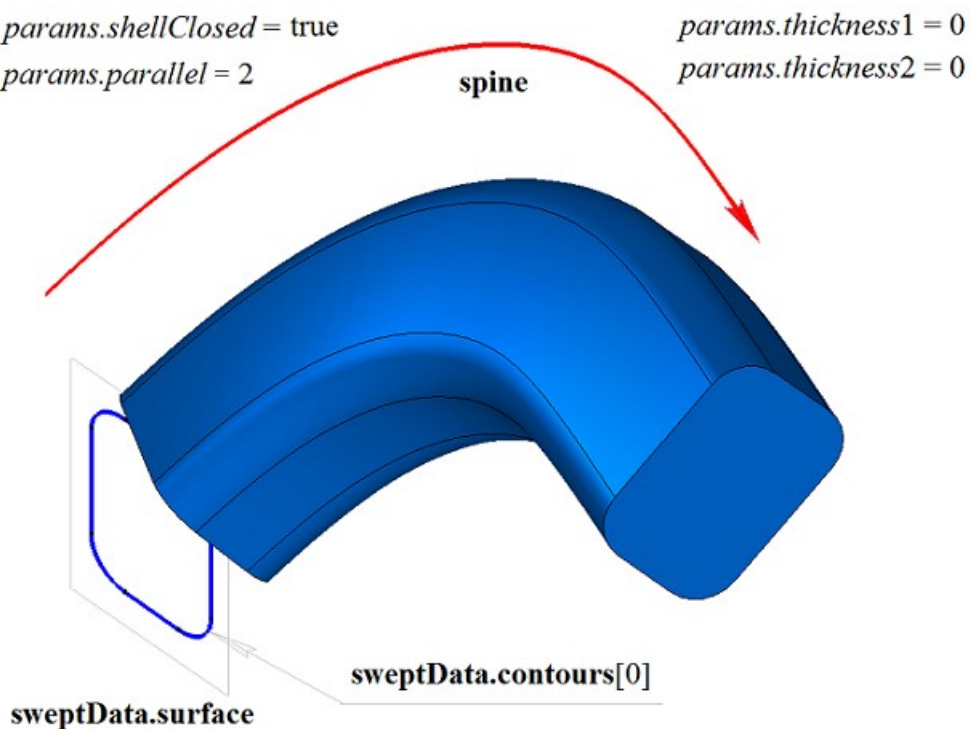


Рис. М.1.5.5.

На рис. М.1.5.6 приведено замкнутое тонкостенное тело заметания, построенное движением контура вдоль направляющей, показанного на рис. М.1.5.2, способом, определяемым параметром *params.parallel*=1. Каждому сегменту контура соответствует грань тела, имя которой взято от соответствующего элемента генератора имен *spines*[0].

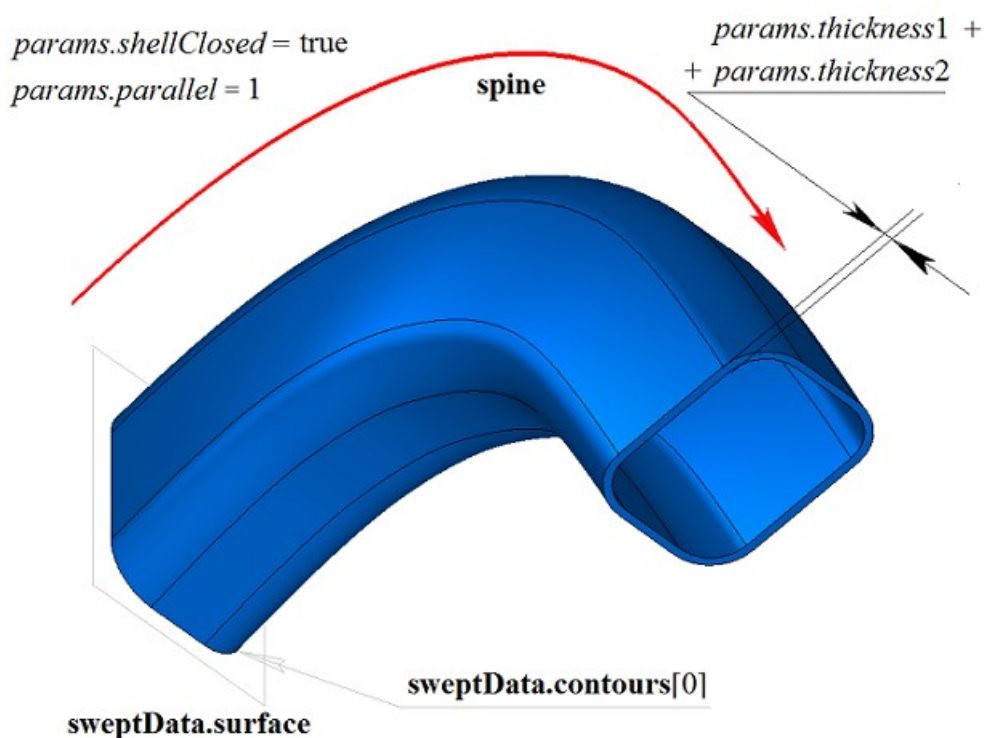


Рис. М.1.5.6.

На рис. М.1.5.7 приведено незамкнутое тело заметания, построенное движением контура вдоль направляющей, показанного на рис. М.1.5.2, определяемым параметром *params.parallel*=1. Параметры построения тела, показанного на рис. М.1.5.4, отличаются от параметров построения тела, показанного на рис. М.1.5.7, только величиной *params.shellClosed*=false.

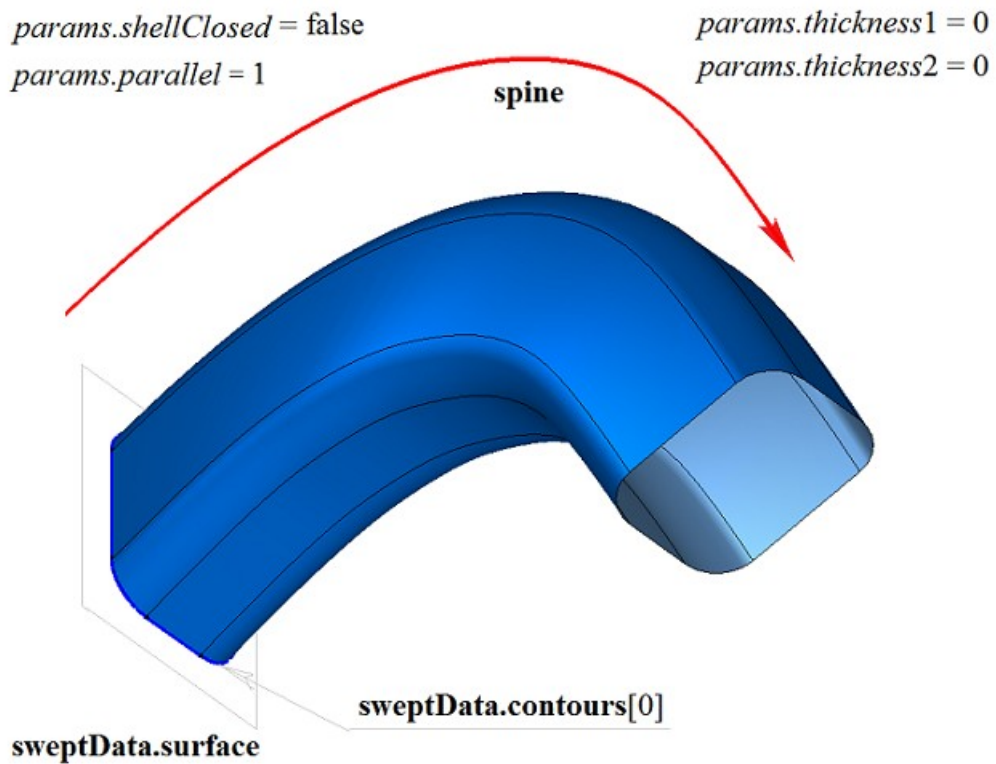


Рис. М.1.5.7.

Двумерный контур может располагаться на плоской или криволинейной поверхности. Например, тело можно построить движением контуров на криволинейной поверхности, полученных от циклов одной из граней твердого тела, показанного на рис. М.1.5.8.

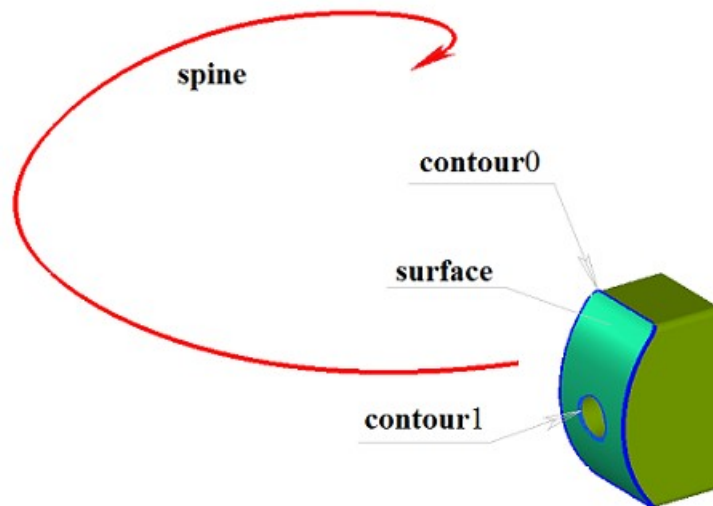


Рис. М.1.5.8.

На рис. М.1.5.9 приведено тело заметания, полученное движением двух контуров на криволинейной поверхности вдоль направляющей, показанных на рис. М.1.5.8. Движение контуров на криволинейной поверхности выполняется способом, соответствующим параметру $params.parallel=1$.

params.shellClosed = true
params.parallel = 1

params.thickness1 = 0
params.thickness2 = 0

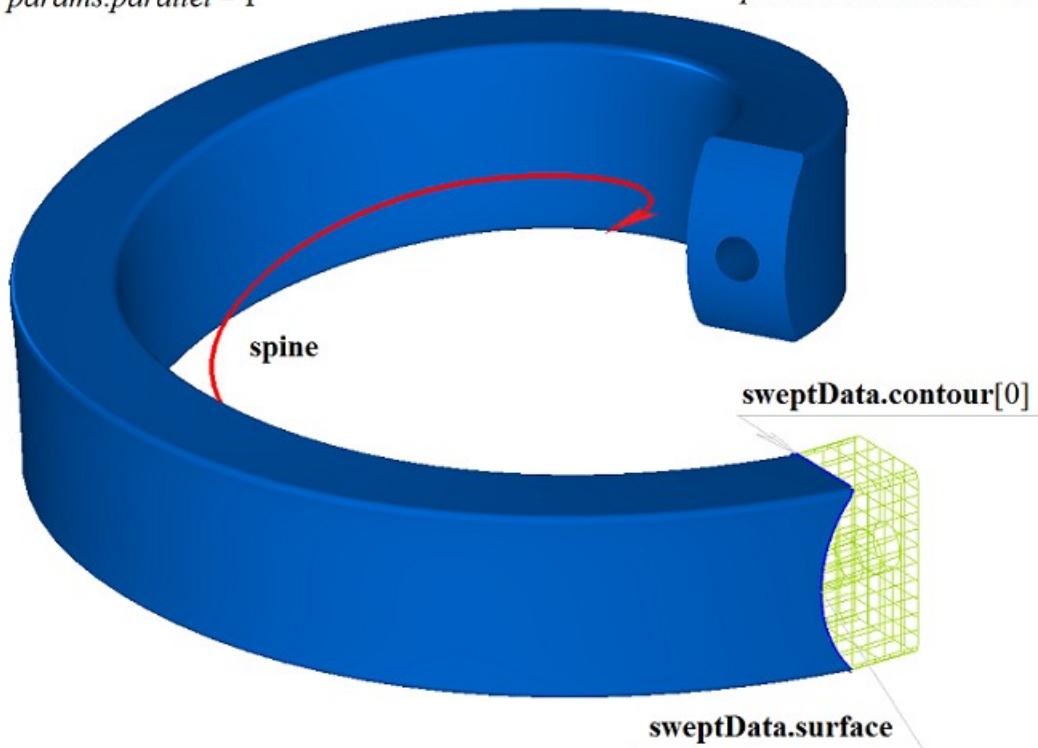


Рис. М.1.5.9.

На рис. М.1.5.10 приведено двусвязное тонкостенное тело заметания, полученное движением двух контуров на криволинейной поверхности вдоль направляющей, показанных на рис. М.1.5.9.

params.shellClosed = true
params.parallel = 1

params.thickness2 = 0
params.thickness1

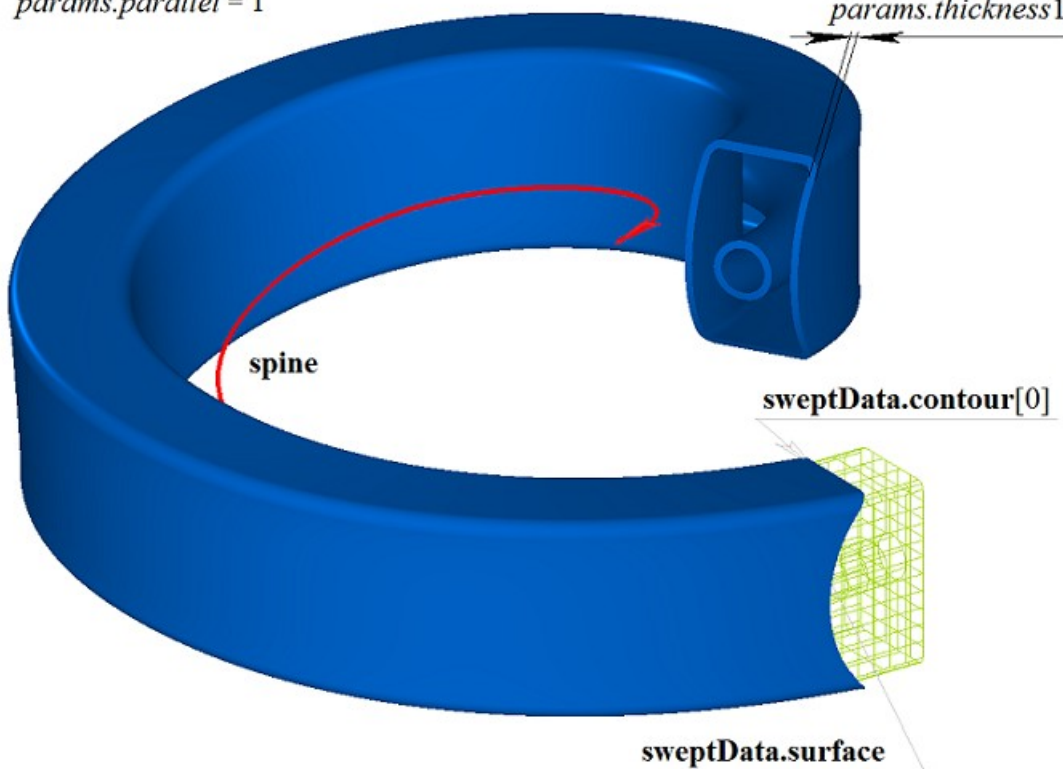


Рис. М.1.5.10.

На рис. М.1.5.11 приведено двусвязное незамкнутое тело заметания, полученное движением двух контуров на криволинейной поверхности вдоль направляющей, показанных на рис. М.1.5.9.

`params.shellClosed = false`
`params.parallel = 1`

`params.thickness1 = 0`
`params.thickness2 = 0`

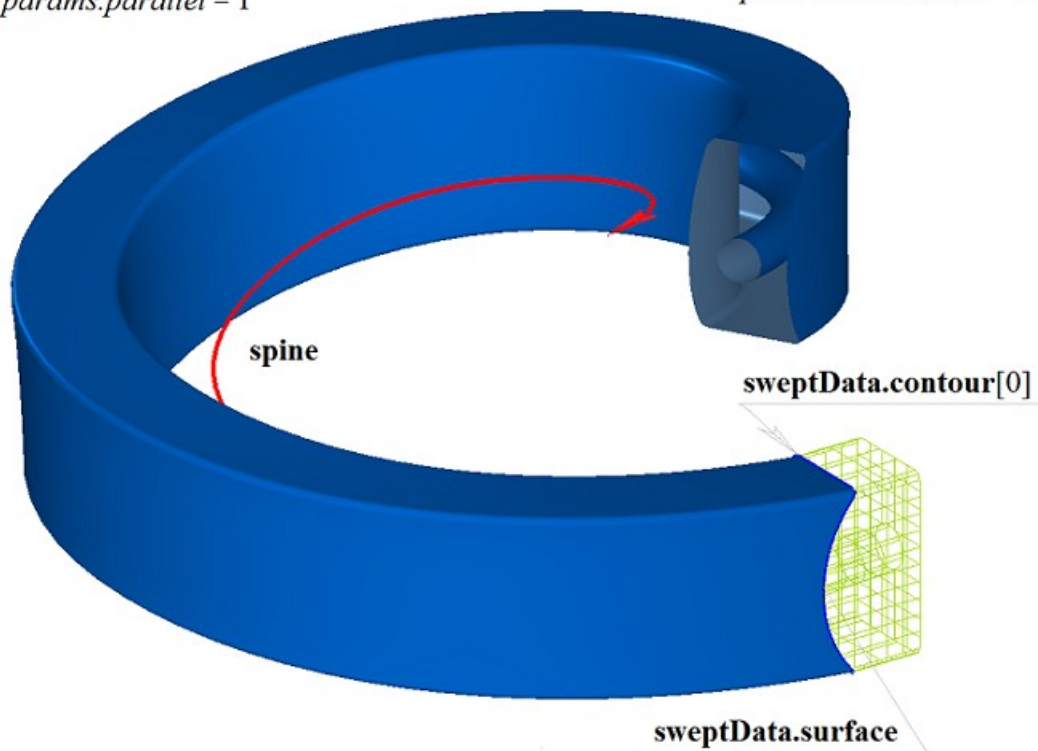


Рис. М.1.5.11.

Если на одной поверхности расположено множество не пересекающихся двумерных контуров, то рассматриваемый метод определяет внешние контуры и вложенные в них внутренние контуры, причем вложение может быть многократным. На рис. М.1.4.12 приведено множество не пересекающихся двумерных контуров **contours**, плоская поверхность **surface** ([MbPlane](#)) и направляющая кривая **spine**.

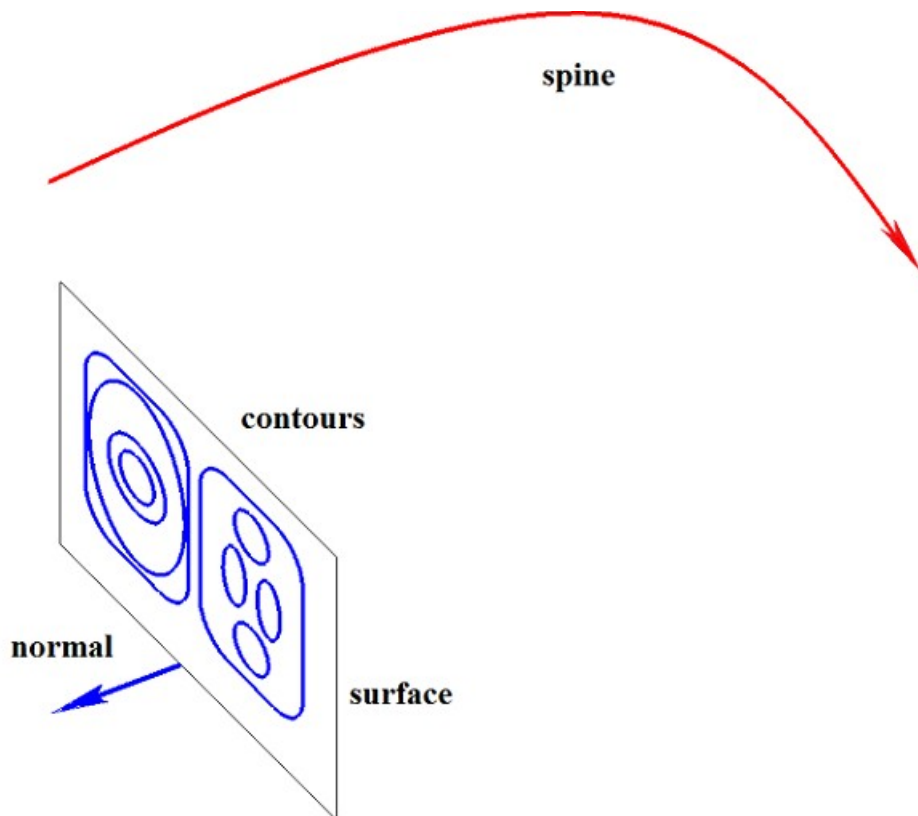


Рис. М.1.5.12.

На рис. М.1.5.13 приведено многосвязное тело заметания, состоящее из нескольких частей, построенное движением множества плоских контуров вдоль направляющей, показанных на рис. М.1.5.12. Контур не должны пересекаться, но могут быть вложены друг в друга, в том числе многократно.

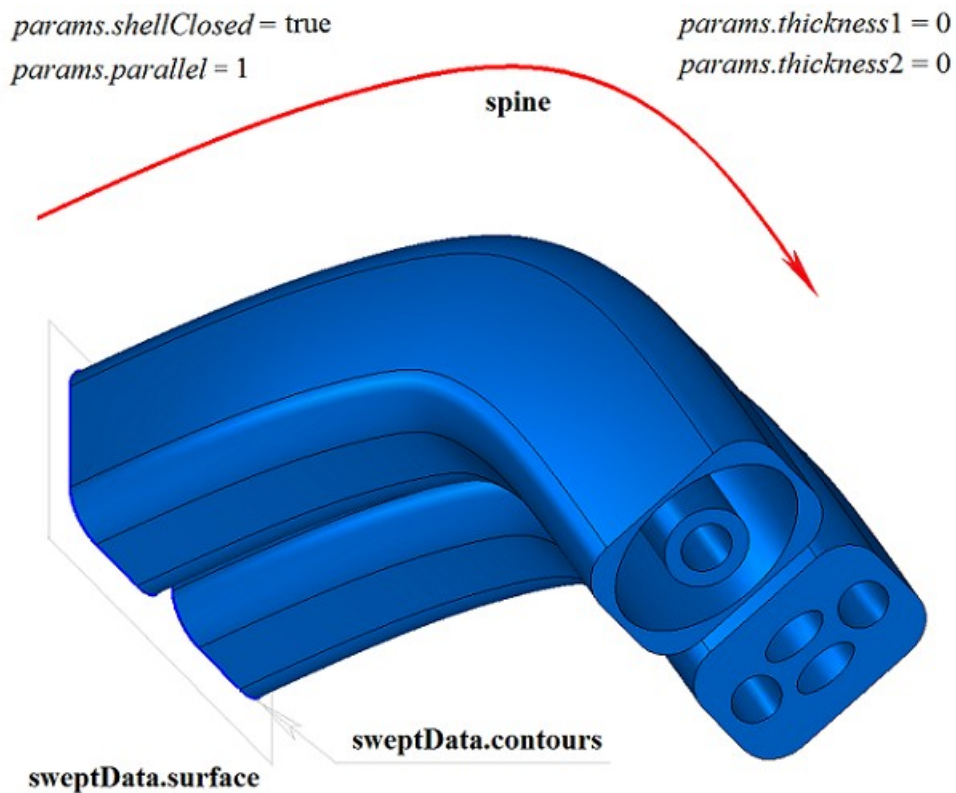


Рис. М.1.5.13.

На рис. М.1.5.14 приведено многосвязное тонкостенное тело заметания, состоящее из нескольких частей, построенное движением множества плоских контуров вдоль направляющей, показанных на рис. М.1.5.12.

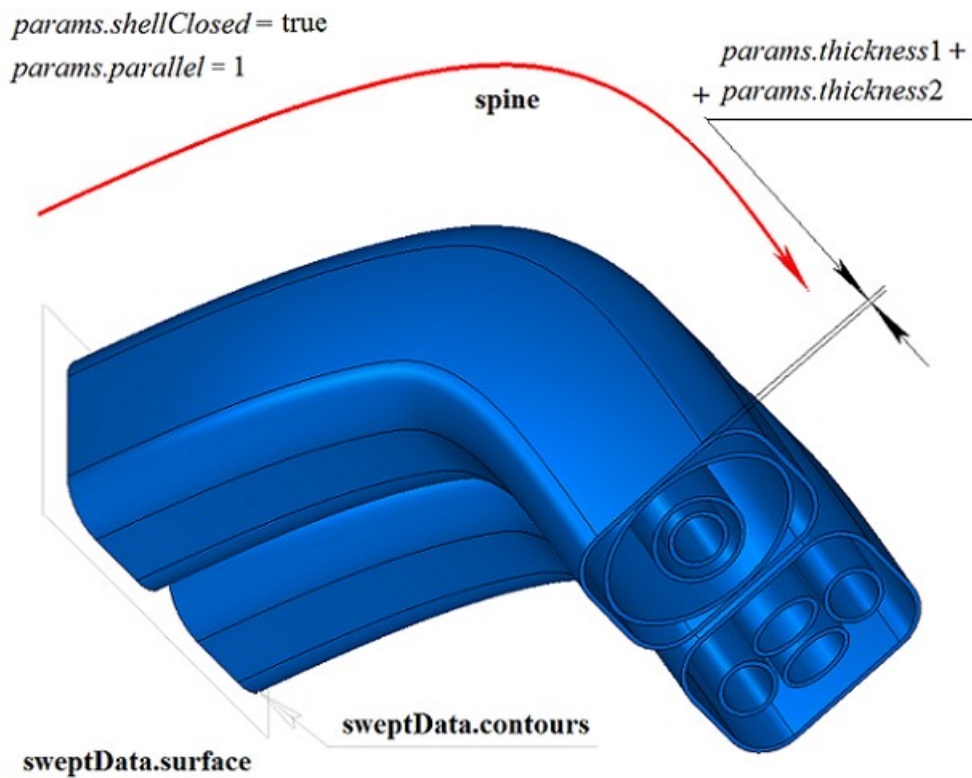


Рис. М.1.5.14.

На рис. М.1.5.15 приведено незамкнутое тело заметания, состоящее из нескольких частей, построенное движением множества плоских контуров вдоль направляющей, показанных на рис. М.1.5.12. При построении незамкнутого тела заметания контуры не должны быть вложены друг друга.

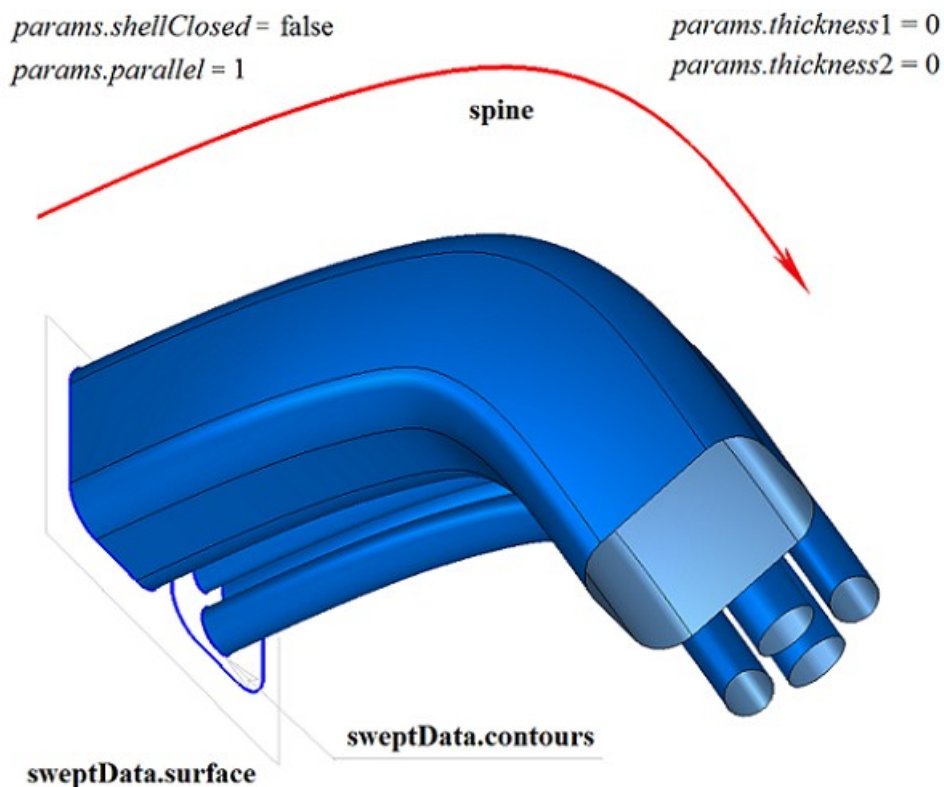


Рис. М.1.5.15.

На рис. М.1.5.16 приведены два трехмерных контура **contour3D0**, **contour3D1** и направляющая кривая **spine**, которые будут использоваться для построения тел заметания.

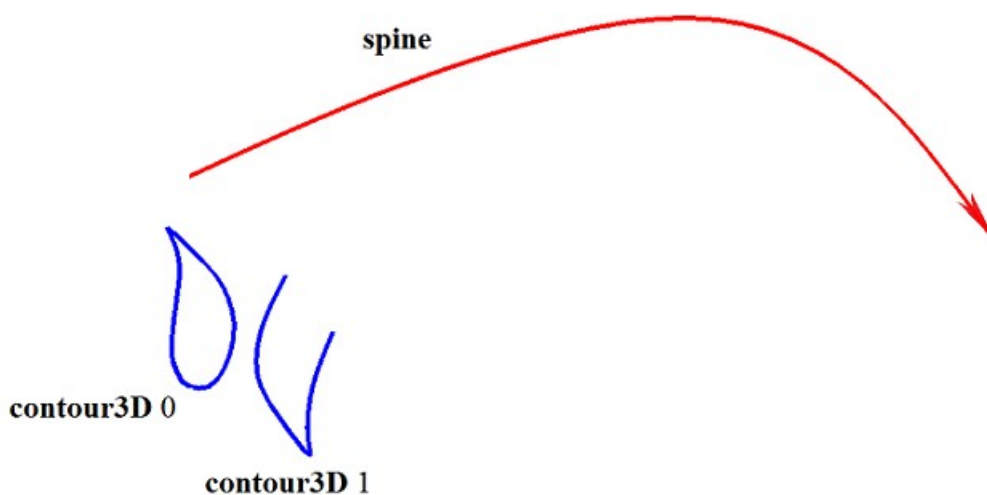


Рис. М.1.5.16.

На рис. М.1.5.17 приведено двусвязное тонкостенное замкнутое тело заметания, полученное движением трехмерных контуров вдоль направляющей, показанных на рис. М.1.5.16.

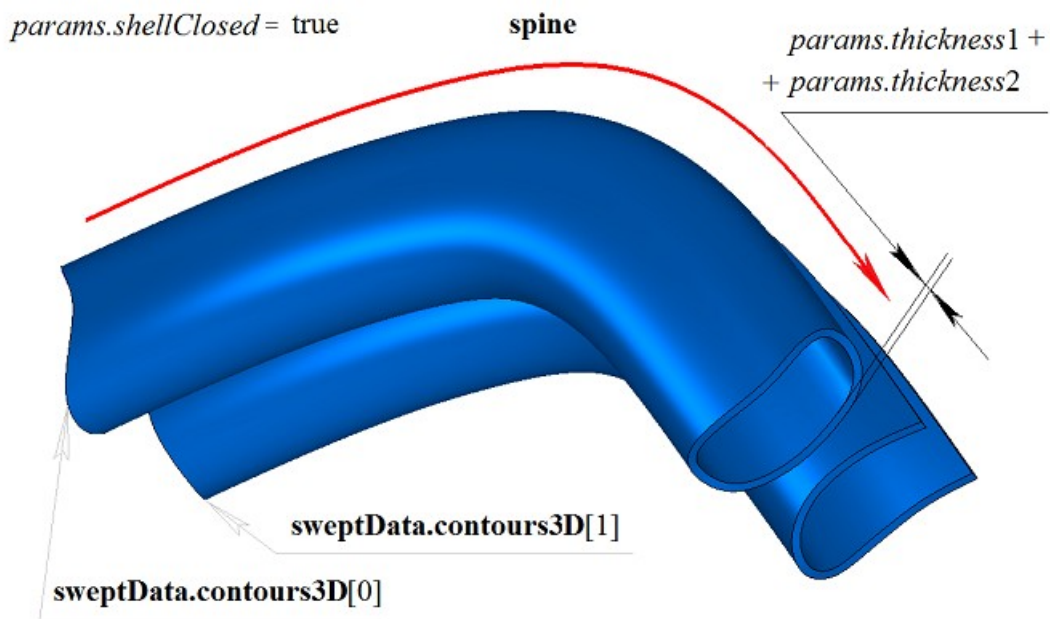


Рис. М.1.5.17.

На рис. М.1.5.18 приведены два незамкнутых тела, полученных движением трехмерных контуров вдоль направляющей, показанных на рис. М.1.5.16.

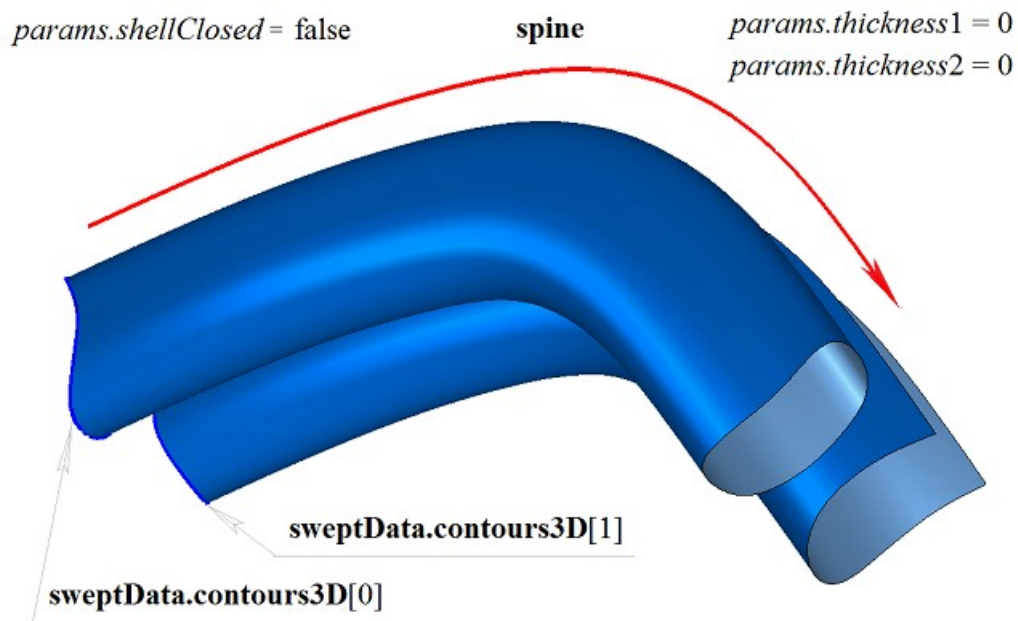


Рис. М.1.5.18.

Метод построения тела заметания **EvolutionSolid** добавляет в журнал построенного тела строитель MbEvolutionSolid, который содержит все необходимые для построения тела данные. Строитель MbEvolutionSolid объявлен в файле sr_evolution_solid.h.

Тестовое приложение test.exe выполняет построение тела заметания командой меню «Создать->Тело->На базе кривых->Движением кривой».

М.1.6. Построение тела по плоским сечениям

Метод
MbResultType
LoftedSolid (SArray<MbPlacement3D> & **places**,
RPAArray<MbContour> & **contours**,
const MbCurve3D * **spine**,
LoftedValues & **params**,
SArray<MbCartPoint3D> * **points**,
const MbSNameMaker & **names**,
PArray<MbSNameMaker> & **snames**,
MbSolid *& **result**)

выполняет построение тела по плоским сечениям.

Входными параметрами метода являются:

- **places** – множество локальных систем координат образующих контуров,
- **contours** – множество образующих контуров,
- **spine** – направляющая кривая (может отсутствовать),
- **params** – параметры построения,
- **points** – множество контрольных точек (может отсутствовать),
- **names** – именователь граней,
- **snames** – именователи образующих контуров.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_solid.h`.

Поверхность построенного по плоским сечениям тела проходит по всем плоским кривым, определяющим тело. Множество **places** содержит локальные системы координат, в плоскости XY которых располагаются двумерные контуры **contours**. Множества **places** и **contours** согласованы по индексу: **contours**[*i*] располагается в плоскости XY локальной системы координат **places**[*i*]. Ориентация контуров **contours** может быть произвольной. Если все контуры **contours** замкнуты, то для предотвращения перекручивания поверхностей положения начал контуров меняются так, чтобы начала располагались как можно ближе друг к другу. Изменить требуемым образом соответствие точек стыковки кривых множества контуров позволяют контрольные точки **points**. Если множество **points** не пустое, то оно должно быть согласовано с множествами **places** и **contours**. Для управления формой тела между сечениями может использоваться направляющая кривая **spine**. Направляющей кривой тела может служить произвольная кривая.

Параметр *params* содержит информацию о способе движения, наличии и толщине стенок тела, замкнутости построенного тела. Параметры *params.thickness1* и *params.thickness2* определяют толщину стенки построенного тонкостенного тела. Параметр *params.thickness1* задает отступ наружу от образующей кривой, а параметр *params.thickness2* задает отступ внутрь от образующей кривой. Параметр *params.shellClosed* управляет замкнутостью построенного тела. Параметр *params.checkSelfInt* сообщает о необходимости проверки результата построения на самопересечение. По умолчанию *params.checkSelfInt=false*, проверка не выполняется. Параметр *params.closed* управляет наличием торцов у тела. При *params.closed=true* торцы тела отсутствуют и тело имеет топологию тора. Векторы *params.vector1* и *params.vector2* задают направление тела в районе начального и конечного торцов, например они позволяют задать направление тела в районе торцов ортогональным плоскостям торцов. По умолчанию векторы *params.vector1* и *params.vector2* равны нулю.

На рис. М.1.6.1 приведены данные, используемые при построении, и схема наследования параметров построения тела по плоским сечениям `LoftedValues & params`.

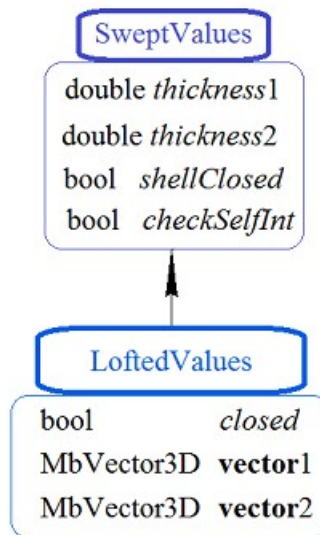


Рис. М.1.6.1.

Параметры *names* и *snames* обеспечивают именование граней построенного тела. На рис. М.1.6.2 приведено множество двумерных контуров **contours** и их локальные системы координат **places**. Стрелками указаны направления нормалей локальных систем координат.

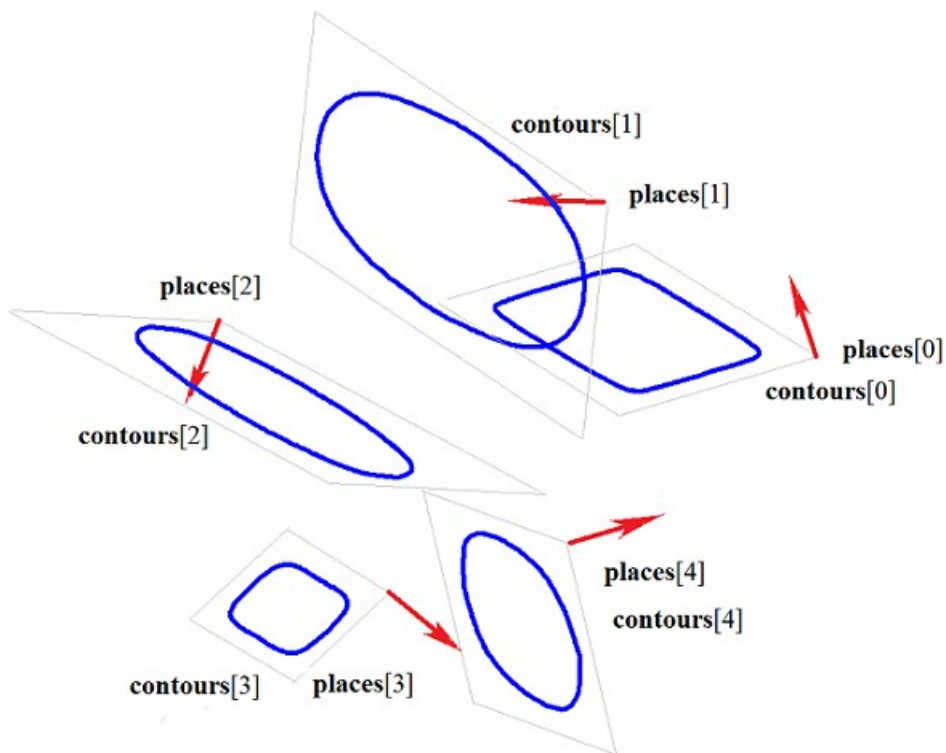


Рис. М.1.6.2.

На рис. М.1.6.3 приведено тело, построенное по плоским сечениям, приведенным на рис. М.1.6.2, с заданными направлениями нормалей на торцах для значения параметра *params.closed=false*.

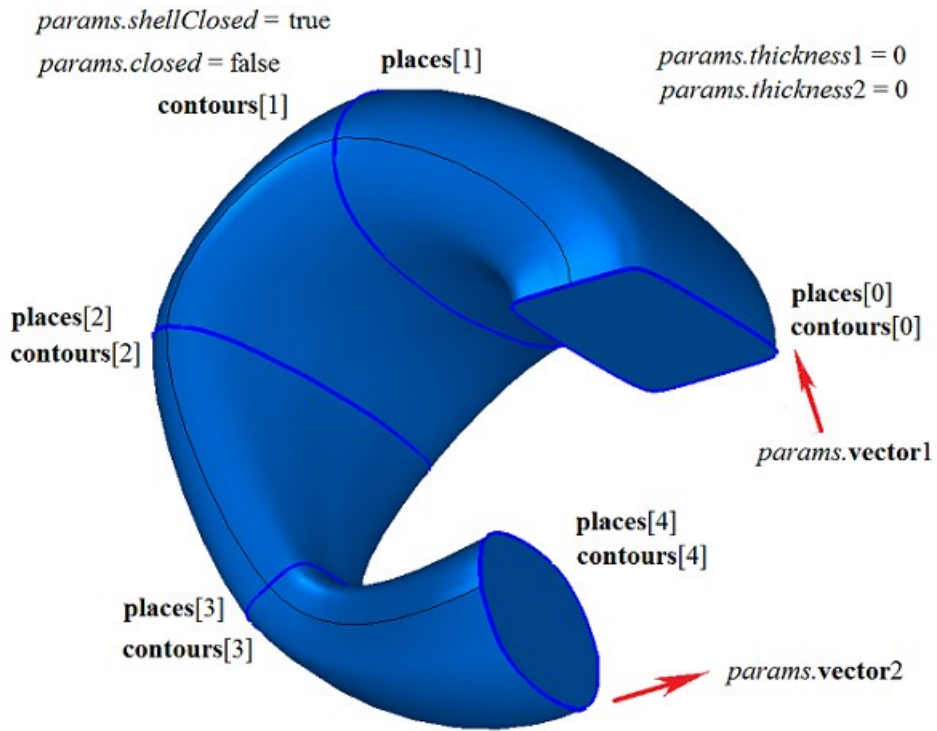


Рис. М.1.6.3.

На рис. М.1.6.4 приведено тело, построенное по плоским сечениям, приведенным на рис. М.1.6.2, для значения параметра *params.closed*=true. Построенное тело имеет топологию тора и у него отсутствуют торцы.

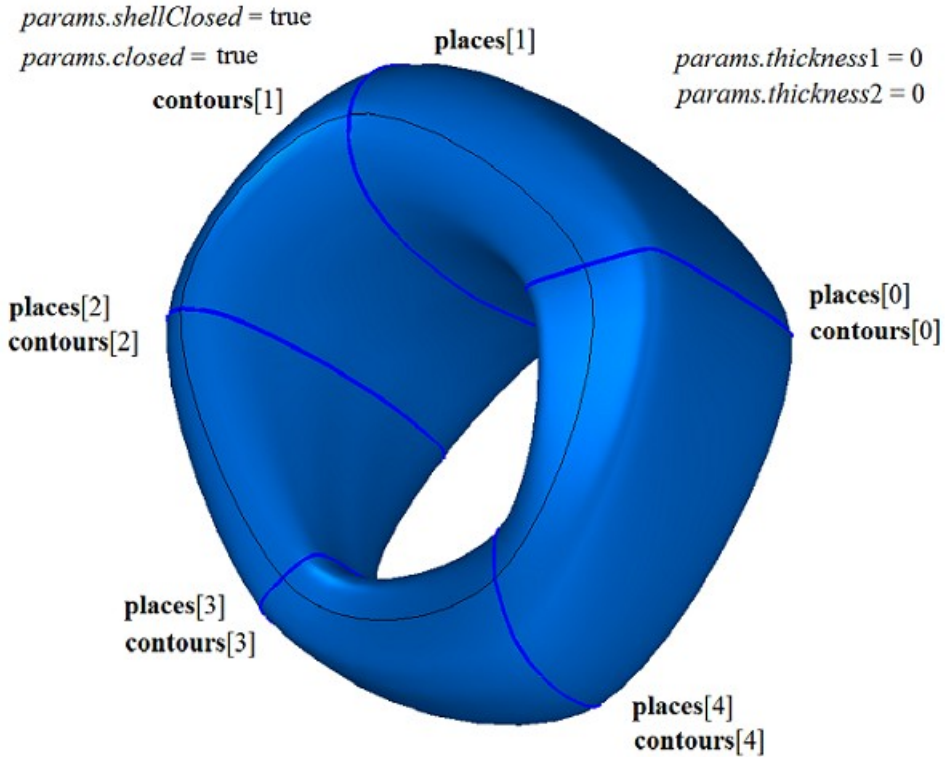


Рис. М.1.6.4.

На рис. М.1.6.5 приведено тонкостенное тело, построенное по плоским сечениям, приведенным на рис. М.1.6.2, без задания нормалей на торцах для значения параметра *params.closed*=false.

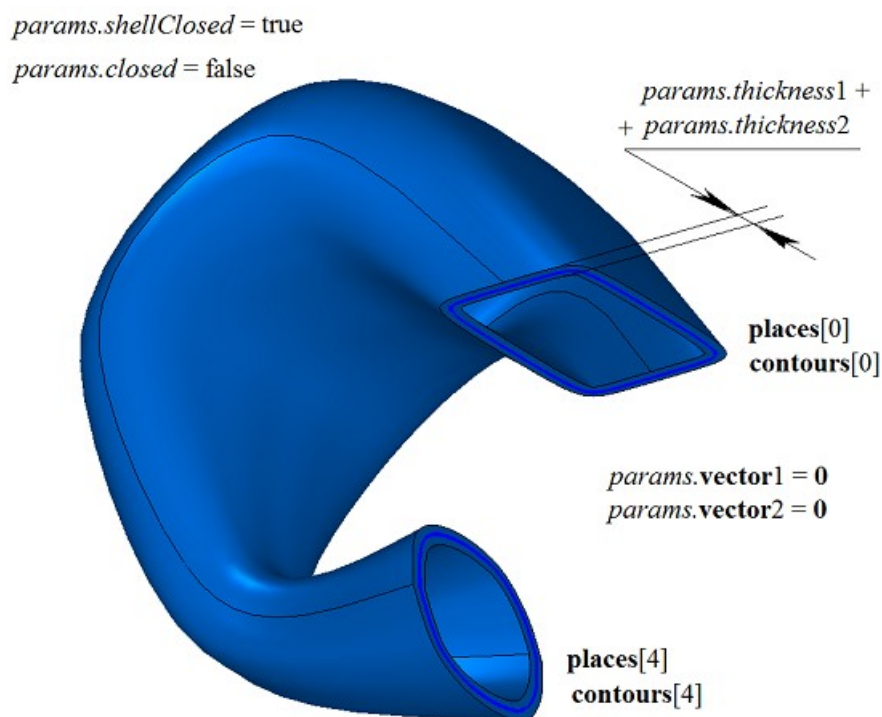


Рис. М.1.6.5.

На рис. М.1.6.6 приведено тонкостенное тело, построенное по плоским сечениям, приведенным на рис. М.1.6.2, с заданными нормальными на торцах для значения параметра *params.closed*=false.

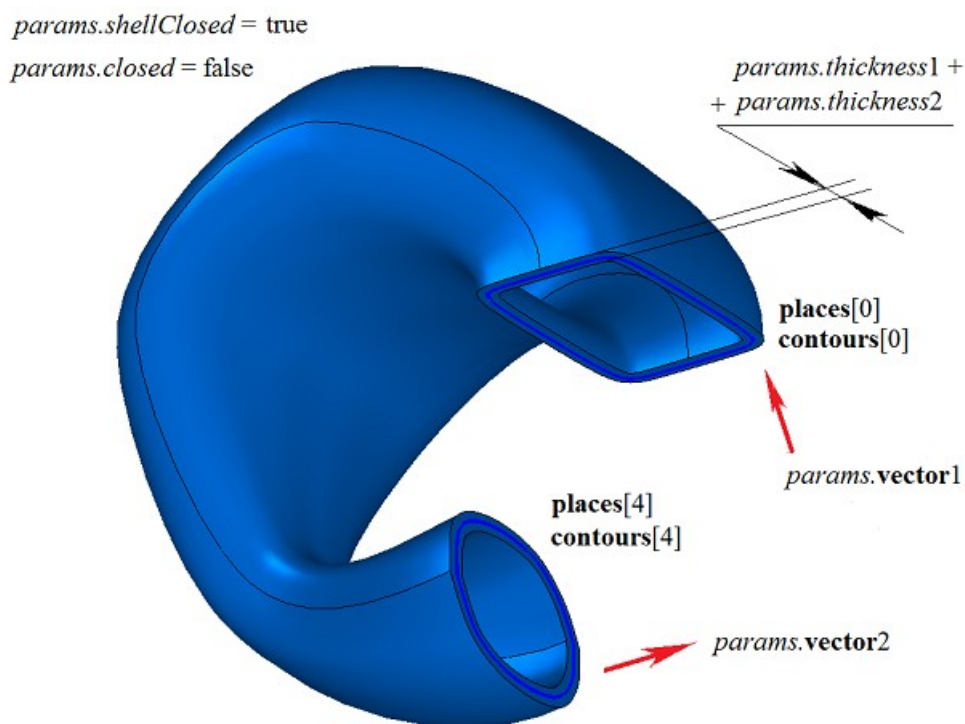


Рис. М.1.6.6.

На рис. М.1.6.7 приведено тонкостенное тело, построенное по плоским незамкнутым контурам с не заданными нормальными на торцах для значения параметра *params.closed*=false. Параметры *params.thickness1* и *params.thickness2* должны быть не нулевыми.

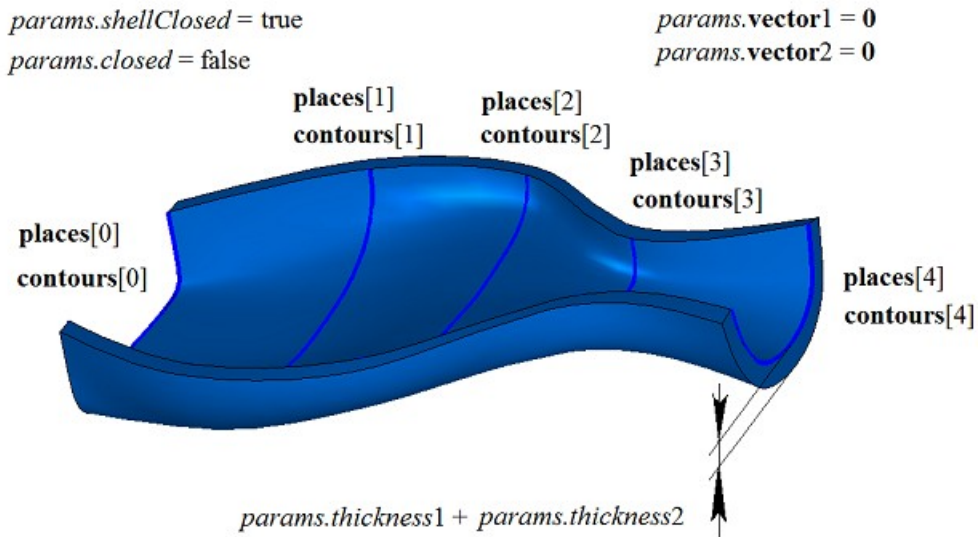


Рис. М.1.6.7.

На рис. М.1.6.8 приведено незамкнутое тело, построенное по плоским незамкнутым контурам с не заданными нормальями на торцах для значения параметра *params.closed*=false. Параметры *params.thickness1* и *params.thickness2* могут быть нулевыми.

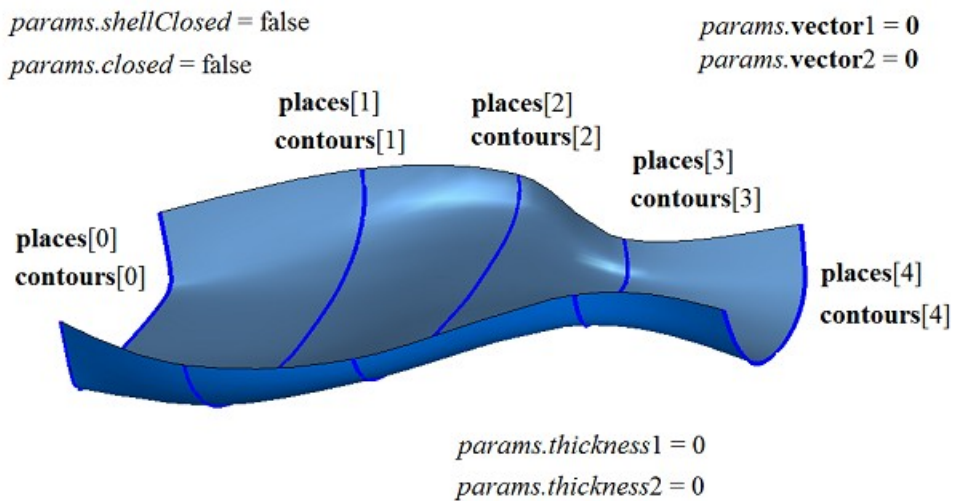


Рис. М.1.6.8.

При неравномерном числе сегментов плоских контуров выполняется разбивка некоторых сегментов так, чтобы число сегментов в каждом контуре множества **contours** было бы равным. На рис. М.1.6.9 приведены три контура с разным числом сегментов.

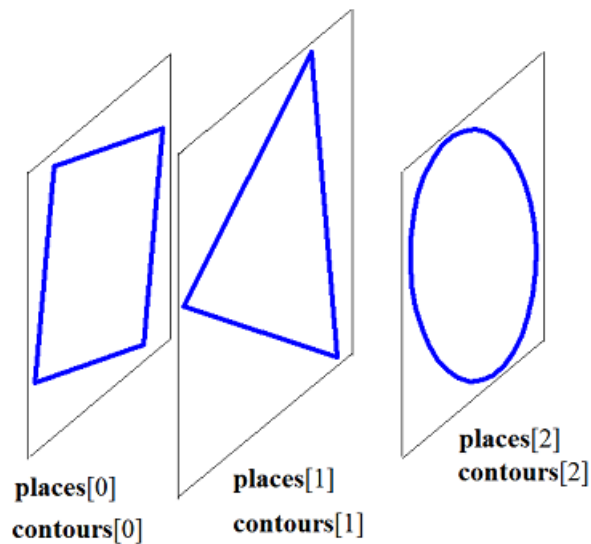


Рис. М.1.6.9.

На рис. М.1.6.10 приведено построенное по этим контурам тело, в котором один из сегментов треугольного контура разбит на два сегмента, а окружность разбита на четыре дуги.

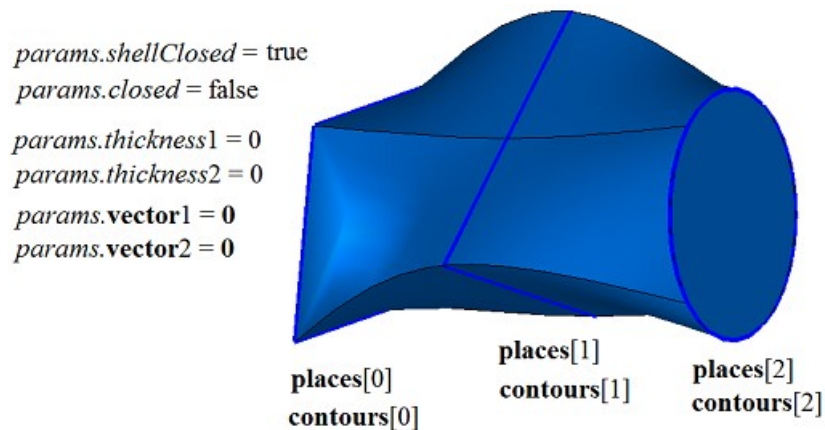


Рис. М.1.6.10.

Управлять положением ребер, соединяющих вершины разных контуров множества, можно с помощью контрольных точек **points**. Точки **points[i]** указывают положения стыков сегментов разных контуров множества, которые должны соединяться ребрами. Для демонстрации применения контрольных точек **points** построим тело по плоским сечениям, приведенным на рис. М.1.6.11.

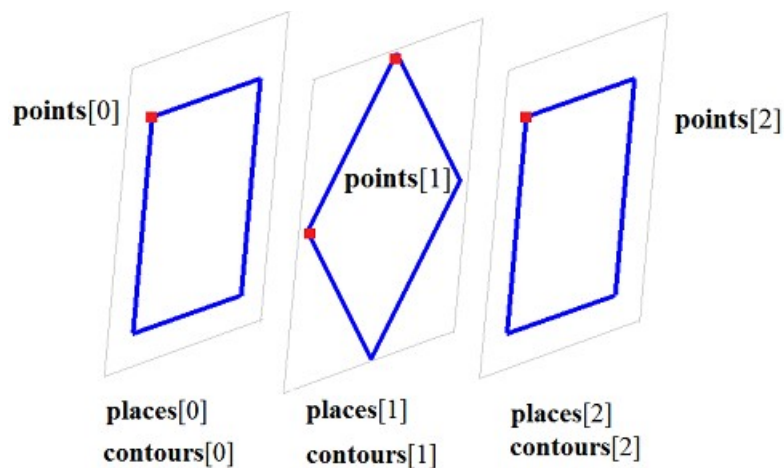


Рис. М.1.6.11.

На рис. М.1.6.12 и М.1.6.13 показаны тела, построенные по плоским сечениям, приведенным на рис. М.1.6.11, с различным положением контрольных точек **points**.

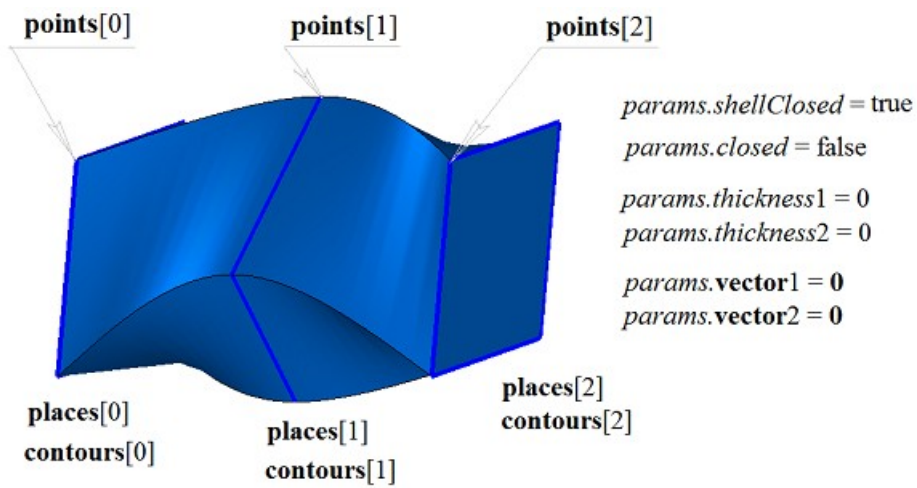


Рис. М.1.6.12.

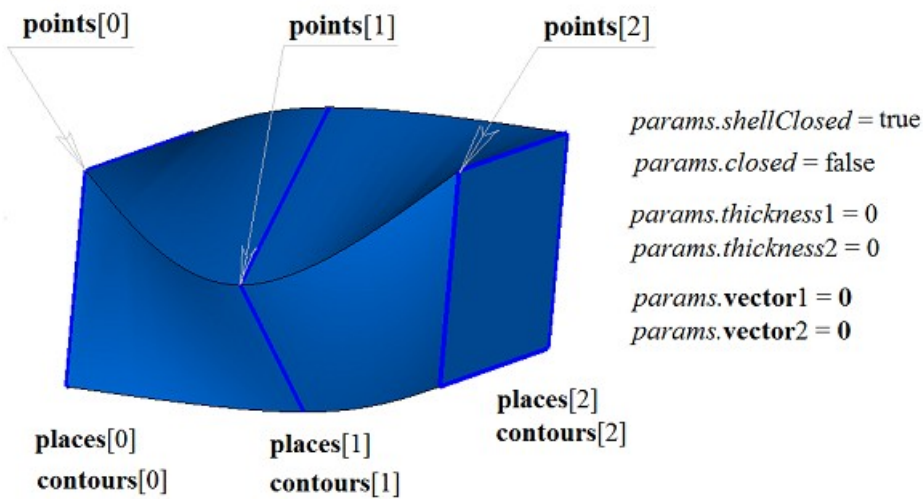


Рис. М.1.6.13.

На рис. М.1.6.14 приведены два двумерных контура и кривая **spine**, которая будет служить направляющей при построения тела по плоским сечения с направляющей кривой.

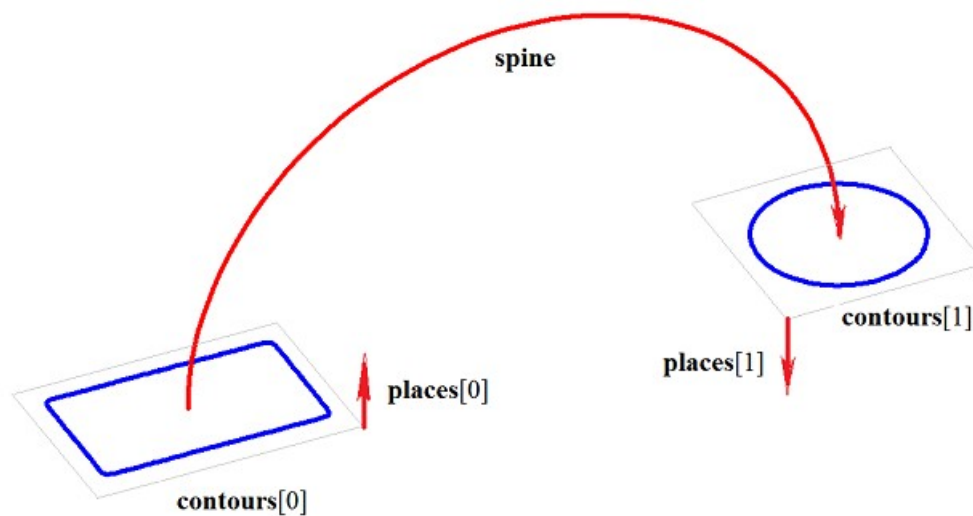


Рис. М.1.6.14.

На рис. М.1.6.15 приведено тело, построенное по плоским сечениям и направляющей, приведенным на рис. М.1.6.14.

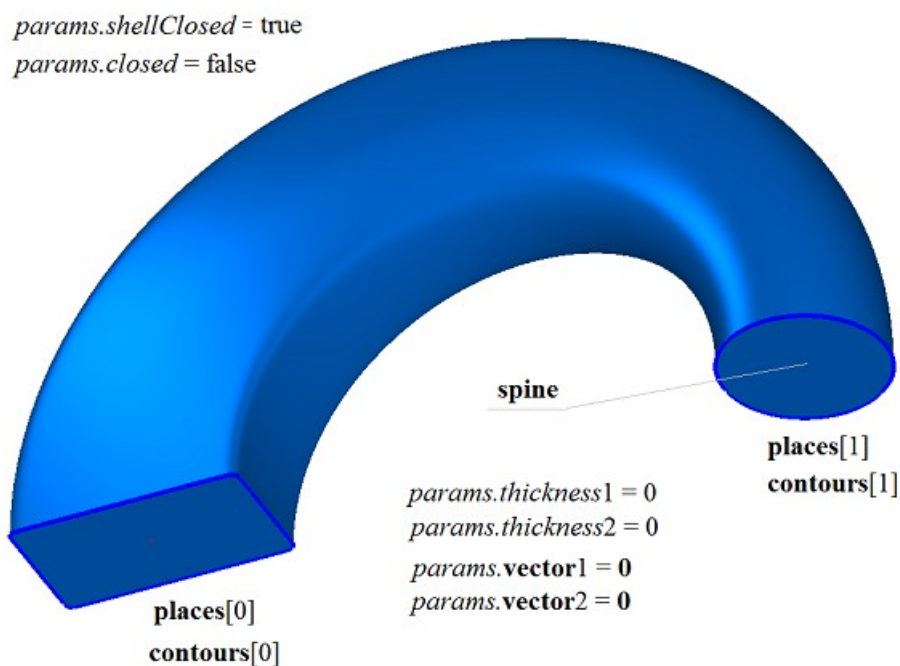


Рис. М.1.6.15.

На рис. М.1.6.16 приведено тонкостенное тело, построенное по плоским сечениям и направляющей, приведенным на рис. М.1.6.14.

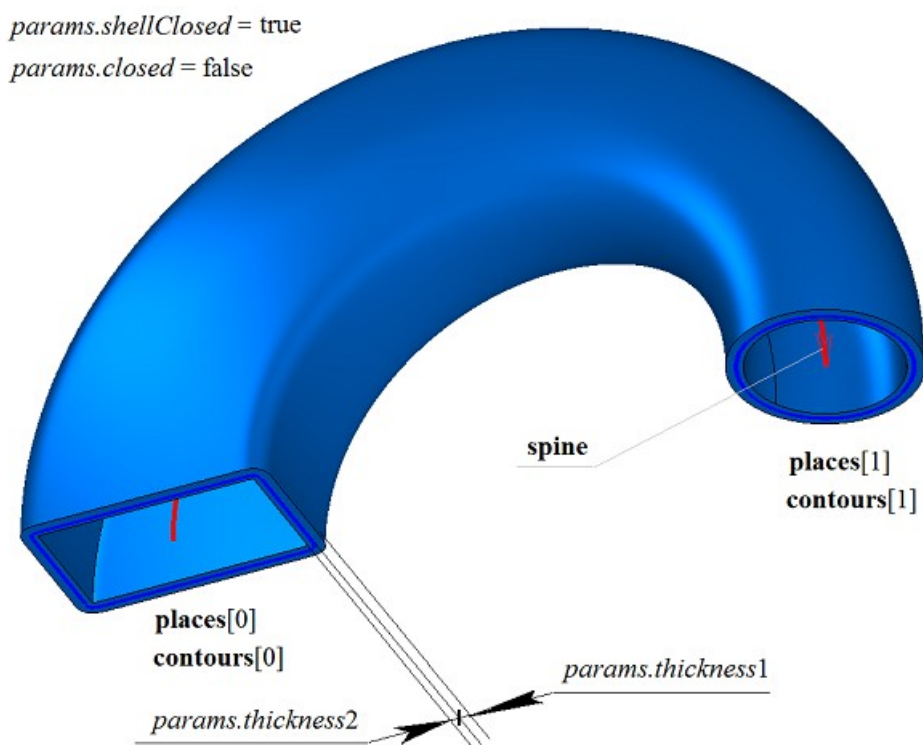


Рис. М.1.6.16.

На рис. М.1.6.17 приведено незамкнутое тело, построенное по плоским сечениям и направляющей, приведенным на рис. М.1.6.14.

```
params.shellClosed = false
params.closed = false
```

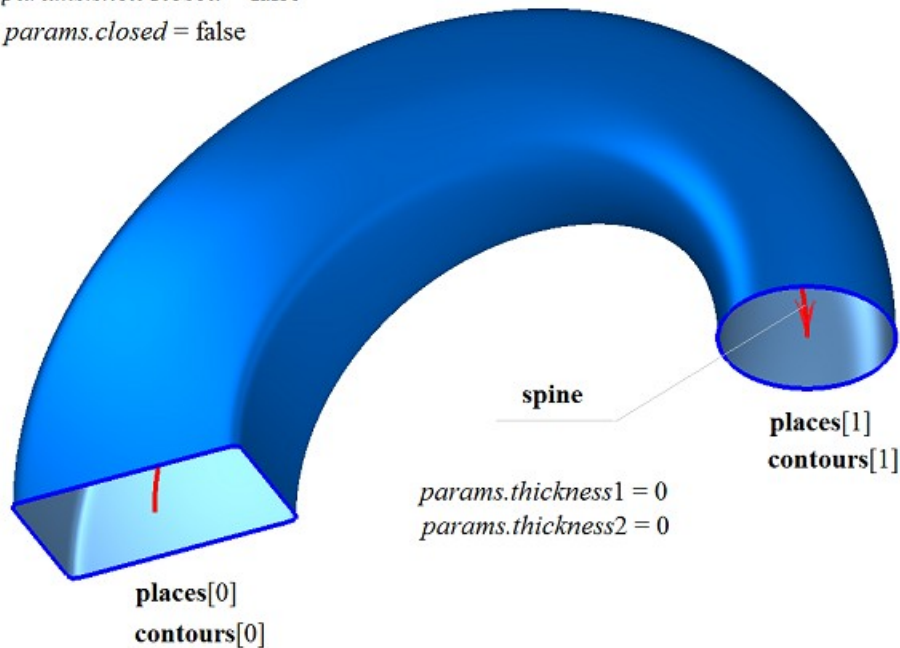


Рис. М.1.6.17.

Метод построения тела по плоским сечениям **LoftedSolid** добавляет в журнал построенного тела строитель **MbLoftedSolid**, который содержит все необходимые для построения тела данные. Строитель **MbLoftedSolid** объявлен в файле `cr_lofted_solid.h`.

Тестовое приложение `test.exe` выполняет построение тела по плоским сечениям командами меню «Создать->Тело->На базе кривых->По сечениям», «Создать->Тело->На базе кривых->По сечениям с направляющей», «Создать->Оболочку->На базе кривых->По сечениям» и «Создать->Оболочку->На базе кривых->По сечениям с направляющей».

М.1.7. Создание тела по заданному множеству граней

Метод

MbSolid *

CreateSolid (**MbFaceShell** & **faceSet**,
const **MbSNameMaker** & **names**)

создаёт тело с заданным множеством граней без истории построения.

Входными параметрами метода являются:

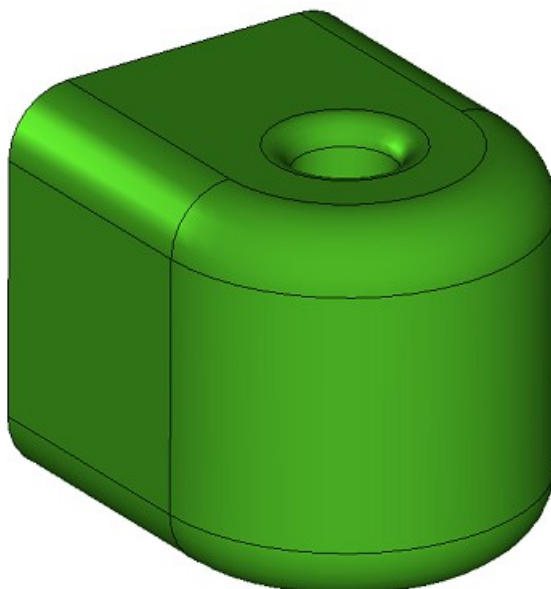
- **faceSet** – множество граней,
- **names** – именователю граней.

При удачной работе метод возвращает построенное тело, в противном случае метод возвращает ноль.

Метод объявлен в файле `action_solid.h`.

Параметр **faceSet** содержит исходное множество граней тела. Параметр **names** обеспечивает именование граней построенного тела.

На рис. М.1.7.1 приведено тело, построенное по множеству граней.



faceSet

Рис. М.1.7.1.

Метод выполняет именованье неименованных граней, ребер, вершин и создаёт тело по заданному множеству граней. Рассматриваемый метод не выполняет никаких проверок и построений. Если множество граней содержит краевые ребра, то метод построит незамкнутое тело. В журнал созданного тела метод **CreateSolid** добавляет простой строитель MbSimpleCreator, который объявлен в файле cr_simple_creator.h.

М.1.8. Построение незамкнутого тела на базе поверхности

Метод

MbResultType

SurfaceShell (const [MbSurface](#) & **surface**,
const MbSNameMaker & names,
[MbSolid](#) *& **result**)

выполняет построение тела, состоящего из одной грани, на базе исходной поверхности.

Входными параметрами метода являются:

- **surface** – исходная поверхность,
- **names** – именователь грани.

Выходным параметром метода является построенное тело **result**. При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_shell.h`.

Параметр `names` обеспечивает именованье грани, ребер и вершин построенного тела.

На рис. М.1.8.1 приведено тело, построенное по поверхности, циклически замкнутой по первому параметрическому направлению.

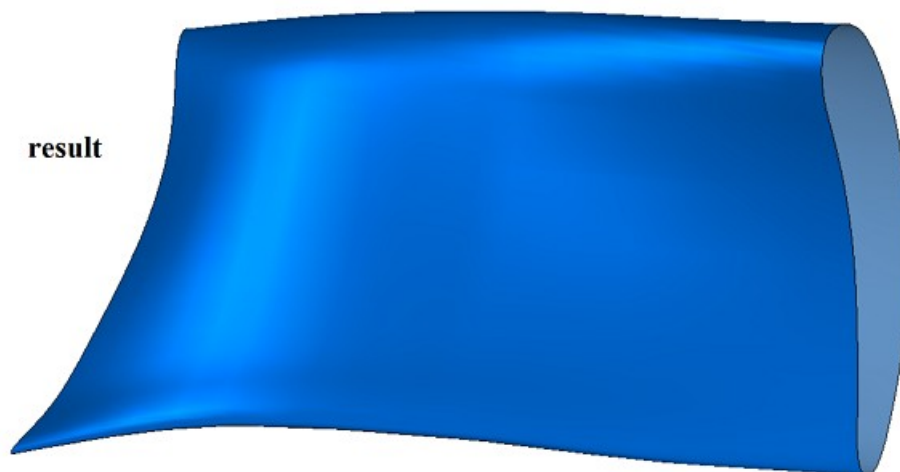


Рис. М.1.8.1.

Рассматриваемый метод не выполняет никаких проверок и вычислений. Построенное тело состоит из одной грани. Грань от исходной поверхности отличается наличием ребер на краях и швах и наличием вершин в точках стыковки рёбер. Если исходная поверхность циклически замкнута по одному или обоим параметрическим направлениям, то построенное тело будет состоять из циклически замкнутой по соответствующим ребрам грани. Если исходная поверхность имеет края, то метод построит незамкнутое тело. В журнал созданного тела метод добавляет простой строитель MbSimpleCreator, который объявлен в файле cr_simple_creator.h.

М.1.9. Построение линейчатого тела

Метод

MbResultType

RuledShell (RuledSurfaceValues & **params**,
const MbSNameMaker & names,
bool *isPhantom*,
MbSolid *& **result**)

выполняет построение линейчатого незамкнутого тела по двум кривым, заданным в параметрах.

Входными параметрами метода являются:

- **params** – параметры построения,
- names – именователь граней,
- *isPhantom* – флаг цели построения (true – режим фантома).

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает rt_Success, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле action_shell.h.

Параметры построения **params** содержат геометрическую информацию, необходимую для построения тела, рис. М.1.9.1.

RuledSurfaceValues

```
MbCurve3D * curve0  
MbCurve3D * curve1  
SArray<double> breaks0  
SArray<double> breaks1  
bool   joinByVertices  
bool   checkSelfInt
```

Рис. М.1.9.1.

Линейчатое тело строится по кривым **params.curve0** и **params.curve1**. Если кривые являются составными, то на каждой паре сегментов составных кривых будет построена линейчатая поверхность, а на каждой поверхности будет построена грань тела. Смежные грани будут иметь общие ребра. Кривые **params.curve0** и **params.curve1** могут быть дополнительно разбиты на сегменты параметрами **params.breaks0** и **params.breaks1**. Параметр **params.joinByVertices** указывает, соединять ли контуры с одинаковым количеством сегментов через вершины. Параметр **params.checkSelfInt** указывает, проверять ли кривые на самопересечение. По умолчанию **params.joinByVertices=false** и **params.checkSelfInt=false**.

На рис. М.1.9.2 приведено линейчатое тело, построенное по составным кривым, каждая из которых содержит три сегмента.

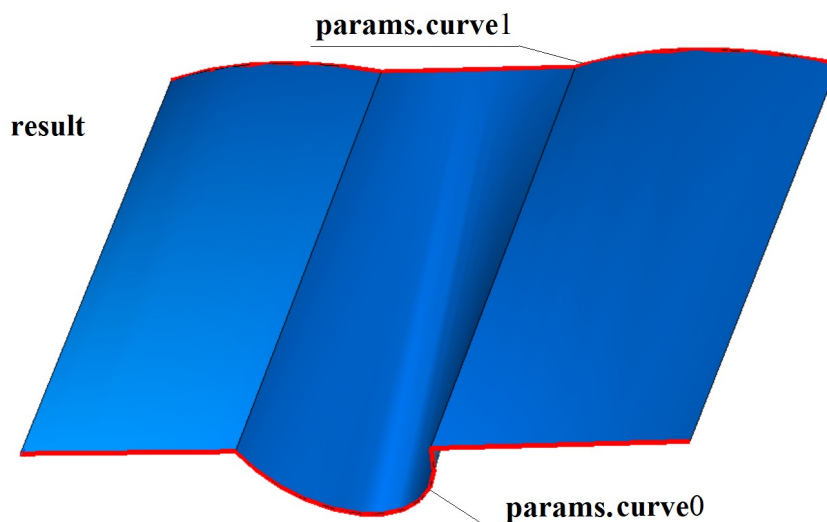


Рис. М.1.9.2.

Построенное тело состоит из одной или нескольких граней. Линейчатое тело всегда имеет края, проходящие по исходным кривым. Рассматриваемый метод может выполнять проверку на самопересечение граней. Кривые **params.curve0** и **params.curve1** могут быть циклически замкнуты. В журнал созданного тела метод добавляет строитель **MbRuledShell**, который объявлен в файле `cr_ruled_shell.h`.

М.1.10. Построение тела по сети кривых

Метод
MbResultType
MeshShell (MeshSurfaceValues & params,

```

const MbSNameMaker & names,
bool isPhantom,
MbSolid *& result )

```

выполняет построение тела по сети кривых, заданной в параметрах построения.

Входными параметрами метода являются:

- **params** – параметры построения,
- *names* – именователь граней.
- *isPhantom* – флаг цели построения (true – режим фантома).

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_shell.h*.

Параметры построения **params** содержат геометрическую информацию, необходимую для построения тела, рис. М.1.10.1.

MeshSurfaceValues	
<code>RPAArray<MbCurve3D></code>	curvesU
<code>RPAArray<MbCurve3D></code>	curvesV
<code>bool</code>	<i>uClosed</i>
<code>bool</code>	<i>vClosed</i>
<code>bool</code>	<i>checkSelfInt</i>
<code>MbeMatingType</code>	<i>type0</i>
<code>MbeMatingType</code>	<i>type1</i>
<code>MbeMatingType</code>	<i>type2</i>
<code>MbeMatingType</code>	<i>type3</i>
<code>MbSurface *</code>	surface0
<code>MbSurface *</code>	surface1
<code>MbSurface *</code>	surface2
<code>MbSurface *</code>	surface3
<code>MbPoint3D * :</code>	<i>point</i>
<code>bool</code>	<i>defaultDir0</i>
<code>bool</code>	<i>defaultDir1</i>
<code>bool</code>	<i>defaultDir2</i>
<code>bool</code>	<i>defaultDir3</i>

Рис. М.1.10.1.

Тело строится по двум множествам кривых **params.curveU** и **params.curveV**. Кривые **params.curveU** расположены вдоль первого параметрического направления граней тела. Кривые **params.curveV** расположены вдоль второго параметрического направления граней тела. Параметр **params.uClosed** указывает на замкнутость поверхности тела вдоль первого параметрического направления, для этого все кривые множества **params.curveU** должны быть циклически замкнуты. Параметр **params.vClosed** указывает на замкнутость поверхности тела вдоль второго параметрического направления, для этого все кривые множества **params.curveV** должны быть циклически замкнуты. С помощью параметров **params.uClosed** и **params.vClosed** возможно построение незамкнутых поверхностей на циклически замкнутых кривых. Параметр **params.checkSelfInt** указывает, проверять ли поверхность построенного тела на самопересечение. По умолчанию **params.checkSelfInt=false**.

Параметры **params.type0**, **params.type1**, **params.type2**, **params.type3** совместно с поверхностями **params.surface0**, **params.surface1**, **params.surface2**, **params.surface3** определяют поведение тела на краях при значениях параметров грани $v=vMin$, $u=uMax$, $v=vMax$, $u=uMin$, соответственно. Параметры **params.type0**, **params.type1**, **params.type2**, **params.type3** могут принимать значения *trt_Position*, *trt_Tangent*, *trt_Normal* из перечисления *MbeMatingType*. Значение *trt_Position* параметры

принимают по умолчанию, оно означает прохождение поверхности тела по кривым, поверхности **params.surface0**, **params.surface1**, **params.surface2**, **params.surface3** при этом могут быть равны нулю. Значение **trt_Tangent** означает касание соответствующего края тела поверхности **params.surface0**, **params.surface1**, **params.surface2**, **params.surface3**, соответственно. Значения **trt_Normal** означает ортогональность соответствующего края тела поверхности **params.surface0**, **params.surface1**, **params.surface2**, **params.surface3**, соответственно.

На рис. М.1.10.2 приведена сеть кривых, состоящая из четырех циклически замкнутых кривых множества **params.curveU** и двух незамкнутых кривых множества **params.curveV**.

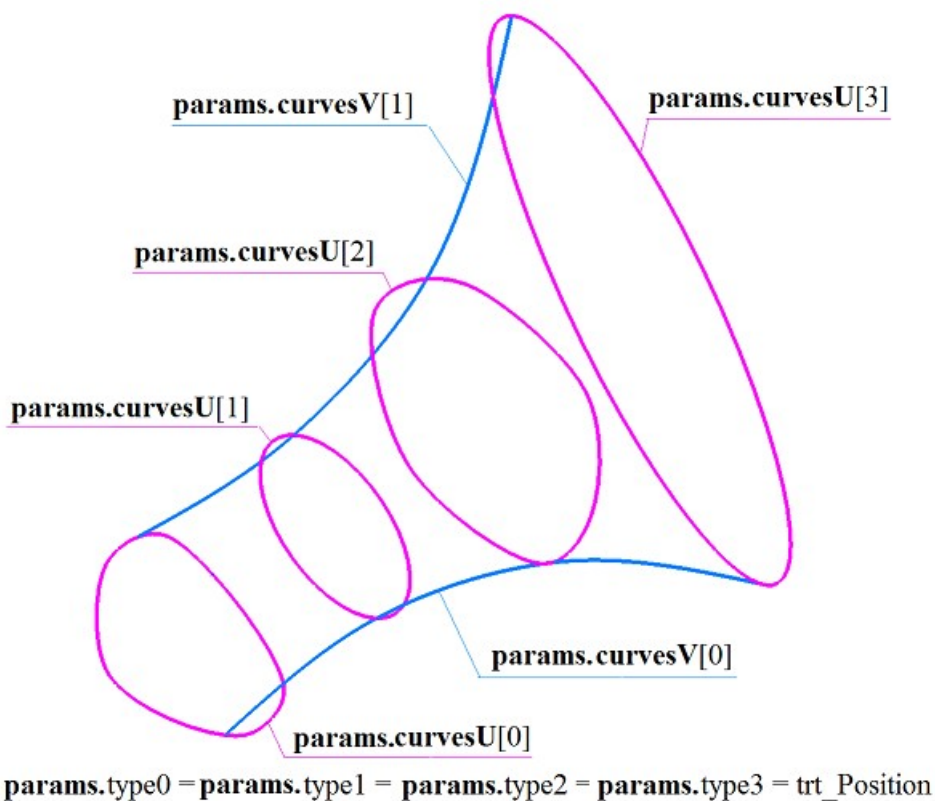


Рис. М.1.10.2.

На рис. М.1.10.3 приведено тело, построенное по сети кривых, приведенных на рис. М.1.10.2.

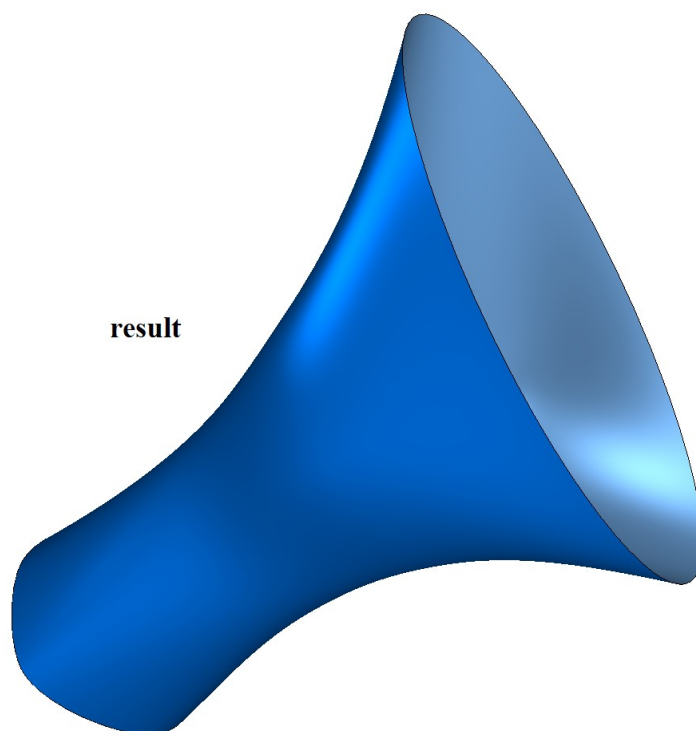


Рис. М.1.10.3.

На рис. М.1.10.4 приведена сеть кривых, состоящая из двух незамкнутых кривых множества **params.curveU** и трех незамкнутых кривых множества **params.curveV**, две крайние из которых вырождены в точки.

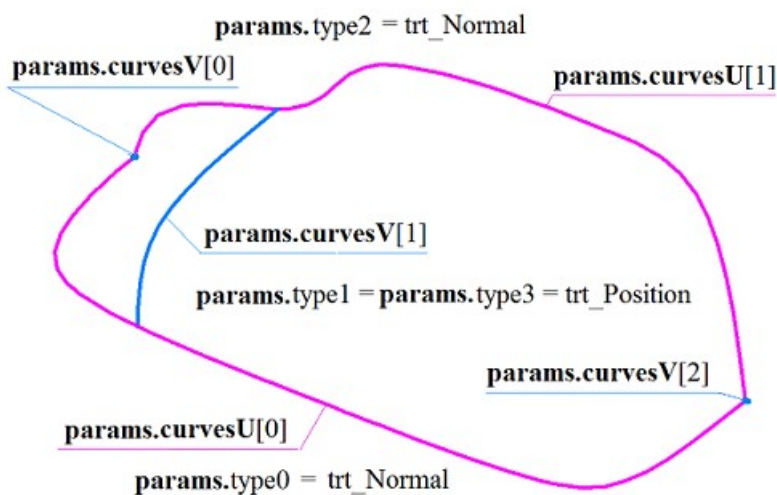


Рис. М.1.10.4.

На рис. М.1.10.5 приведено тело, построенное по сети кривых, приведенных на рис. М.1.10.4. Края построенного тела, соответствующие параметрам грани $v=vMin$ и $v=vMax$, ортогональны плоским поверхностям **params.surface0** и **params.surface2**, так как параметры **params.type0** и **params.type2** имеют значения **trt_Normal**.

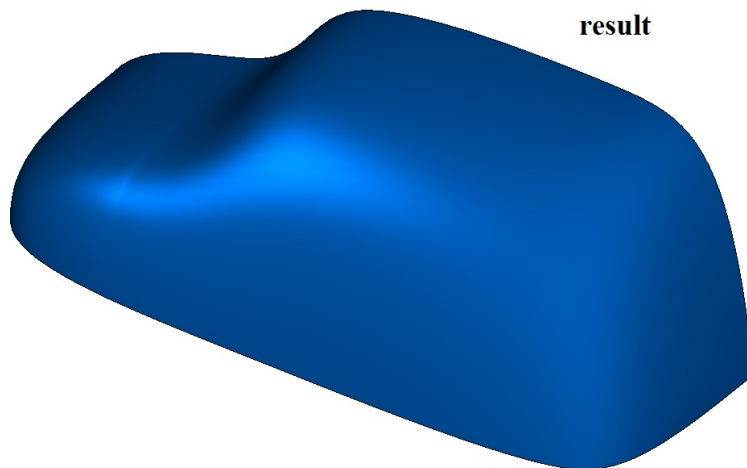


Рис. М.1.10.5.

В журнал созданного тела метод добавляет строитель `MbMeshShell`, который объявлен в файле `cr_mesh_shell.h`.

М.1.11. Построение тела сопряжения несвязанных граней

Метод
`MbResultType`

FacesFillet (`const MbSolid & solid1`,
`const MbFace & face1`,
`const MbSolid & solid2`,
`const MbFace & face2`,
`const SmoothValues & params`,
`const MbSNameMaker & names`,
`MbSolid *& result`)

выполняет построение незамкнутого тела, состоящего из грани скругления между двумя несвязанными гранями.

Входными параметрами метода являются:

- **solid1** – первое тело,
- **face1** – сопрягаемая грань первого тела,
- **solid2** – второе тело,
- **face2** – сопрягаемая грань второго тела,
- *params* – параметры построения,
- *names* – именователь грани.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_shell.h`.

Метод выполняет построение тела, состоящего из грани сопряжения, обеспечивающей гладкое сопряжение граней **face1** и **face2**. Параметры **solid1** и **solid2** содержат исходные тела, которым принадлежат грани **face1** и **face2**.

Построенная грань в поперечном сечении может иметь форму дуги окружности, эллипса, гиперболы, параболы. Формой грани сопряжения управляет параметр *params*. Параметр построения *params* содержит геометрическую информацию, необходимую для построения тела, рис. М.1.11.1.

SmoothValues

```
double distance1
double distance2
double conic
double begLength
double endLength
MbSmoothForm form = { st_Span,
                       st_Fillet,
                       st_Chamfer,
                       st_Slant1,
                       st_Slant2 }

ComeForm smoothCorner = { ec_pointer,
                           ec_either,
                           ec_uniform,
                           ec_sharp }

bool prolong
bool autoSurface
bool keepCant
bool strict
bool equable
MbVector3D vector1
MbVector3D vector2
```

Рис. М.1.11.1.

Из приведенных на рис. М.1.11.1 данных в рассматриваемом методе используются только четыре: *params.distance1*, *params.distance2*, *params.conic* и *params.prolong*. Остальные параметры используются для построения поверхностей сопряжения на базе ребер тела. Величина *params.form* принимает значение *st_Fillet*.

Тело строится по граням **face1** и **face2**. Параметры *params.distance1*, *params.distance2* определяют радиус дуги (или полуоси эллипса) поперечного сечения поверхности сопряжения. Коэффициент *params.conic* управляет формой поверхности сопряжения. Коэффициент *params.conic* может принимать значения от 0.05 до 0.95. При *params.distance1=params.distance2* и *params.conic=0* выполняется построение поверхности сопряжения с поперечным сечением в форме дуги окружности. Поверхность сопряжения строится путем движения сферы, касающейся граней **face1** и **face2**. Опорные края тела сопряжения проходят по точкам касания сферы и граней **face1** и **face2**. При *params.conic=0.5* поперечное сечение грани сопряжения представляет собой дугу параболы. При *params.conic>0.5* поперечное сечение грани сопряжения представляет собой дугу гиперболы. При *params.conic<0.5* поперечное сечение грани сопряжения представляет собой дугу эллипса.

На рис. М.1.11.2 приведены два тела, по двум граням которых далее будут построены тела сопряжения различной формы.

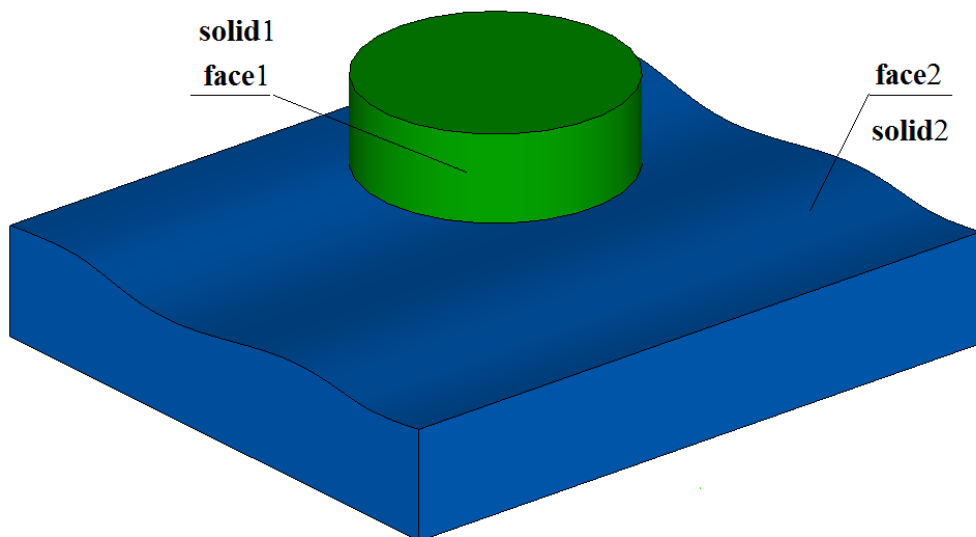


Рис. М.1.11.2.

На рис. М.1.11.3 приведено тело сопряжения с поперечным сечением поверхности в форме дуги окружности, что достигается равенствами: $params.distance1=params.distance2$, $params.conic=0$.

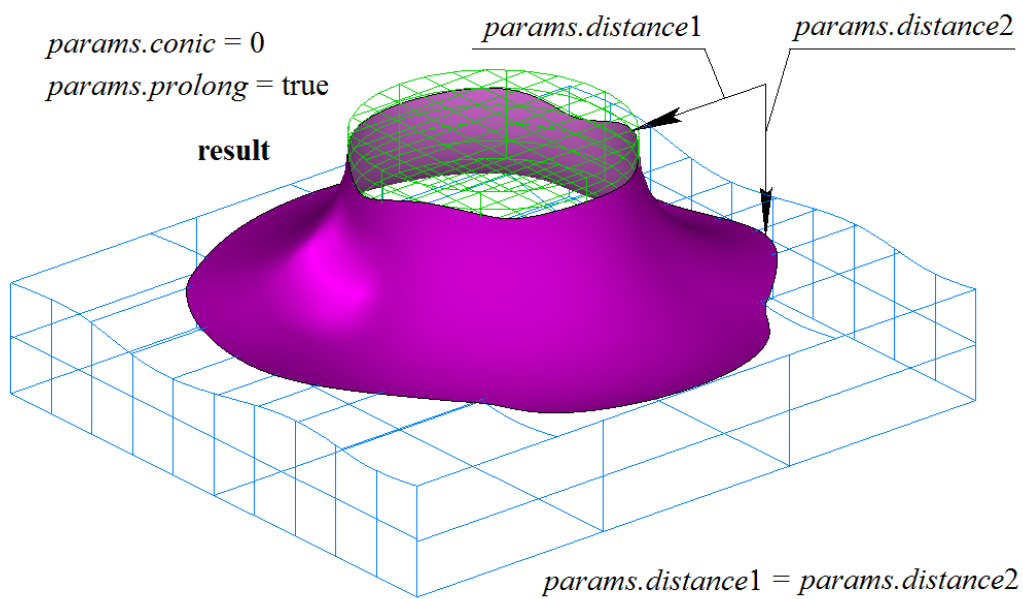


Рис. М.1.11.3.

На рис. М.1.11.4 приведено тело сопряжения с поперечным сечением поверхности в форме дуги гиперболы, что обеспечивает равенство $params.conic=0.8$. Значение параметра $params.prolong=true$ позволяет опорной кривой грани сопряжения на некотором участке выходить за пределы грани **face2**.

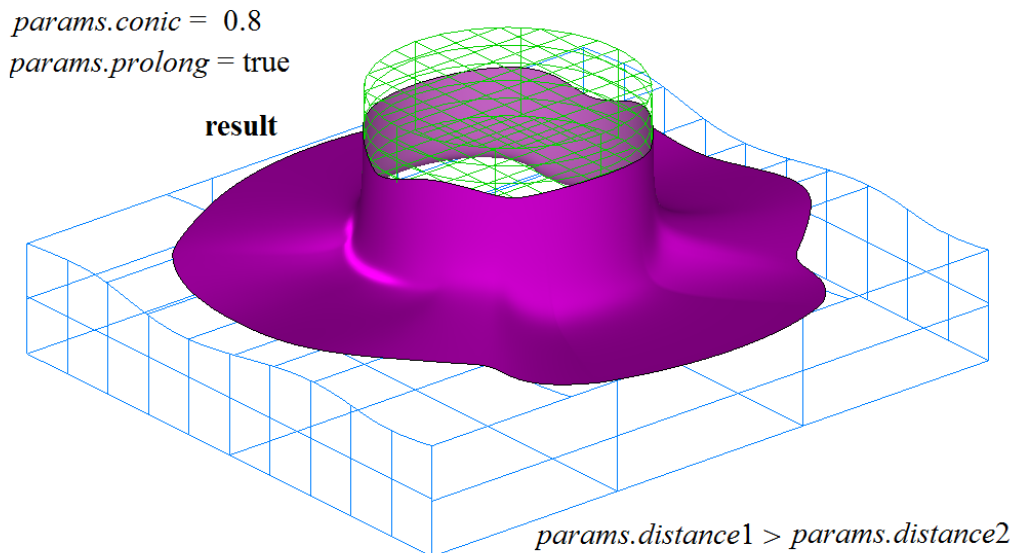


Рис. М.1.11.4.

На рис. М.1.11.5 приведено тело сопряжения с поперечным сечением поверхности в форме дуги эллипса, что обеспечивает равенство *params.conic*=0.2. Значение параметра *params.prolong*=false обрезает грань сопряжения на участке, где опорная кривая построенной грани выходит за пределы грани **face2**.

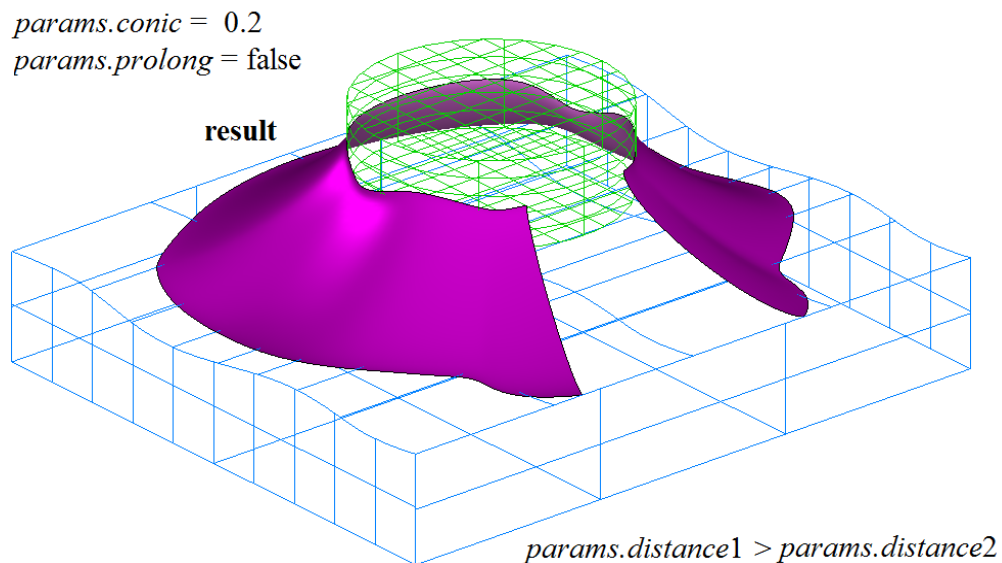


Рис. М.1.11.5.

Метод **FacesFillet** добавляет в журнал построенного тела строитель тела **solid1**, строитель тела **solid2** и строитель MbFilletShell, который объявлен в файле `sr_fillet_shell.h`.

М.1.12. Построение заплатки

Метод
 MbResultType
PatchShell (const RPAArray<MbCurve3D> & curves,
 const PatchValues & params,
 const MbSNameMaker & names,

MbSolid *& result)

выполняет построение зачатки по множеству кривых.

Входными параметрами метода являются:

- **curves** – множество кривых,
- *params* – параметры построения,
- *names* – именованье граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_shell.h*.

Параметры **curves** и *params* содержат необходимую для построения зачатки геометрическую информацию. Параметр *names* обеспечивает именованье грани построенного тела.

Метод выполняет построение тела, состоящего из грани, края которой проходят по заданным кривым. Для построения корректного тела кривые множества **curves** должны стыковаться друг с другом и образовывать замкнутый контур.

Если кривые **curves** лежат в одной плоскости, то зачатка будет представлять собой часть плоскости, лежащую внутри контура, образованного кривыми. Если все кривые **curves** лежат на одной и той же криволинейной поверхности, то зачатка будет представлять собой часть общей поверхности, ограниченную заданными кривыми. Если множество **curves** состоит из одной кривой, то эта кривая будет разбита на две части, а поверхность зачатки будет линейчатой поверхностью, имеющей два полюса. Аналогично строится зачатка, если множество **curves** содержит две кривые – поверхность зачатки будет линейчатой поверхностью, имеющей два полюса. Если количество кривых **curves** равно трем, то поверхность зачатки будет поверхностью *MbCornerSurface*, построенной по трем кривым и имеющей полюс. Если количество кривых **curves** равно четырем, то поверхность зачатки будет поверхностью *MbCoverSurface*, построенной по четырем кривым. В других случаях возможен отказ от построения.

На рис. М.1.12.1 приведено восемь кривых, лежащих на краю внутреннего выреза грани. На рис. М.1.12.2 приведена зачатка, построенная по кривым, приведенным на рис. М.1.12.1.



Рис. М.1.12.1.

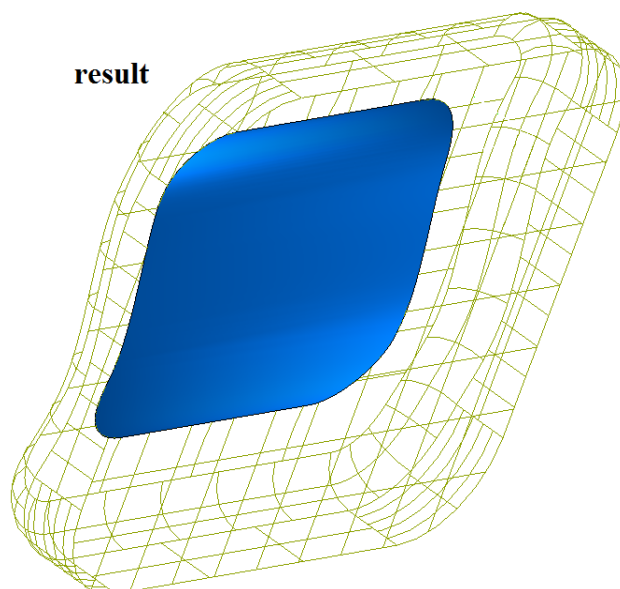


Рис. М.1.12.2.

На рис. М.1.12.3 приведено шесть кривых, лежащих на краях разных граней, полученных путем резки граней общей поверхностью.

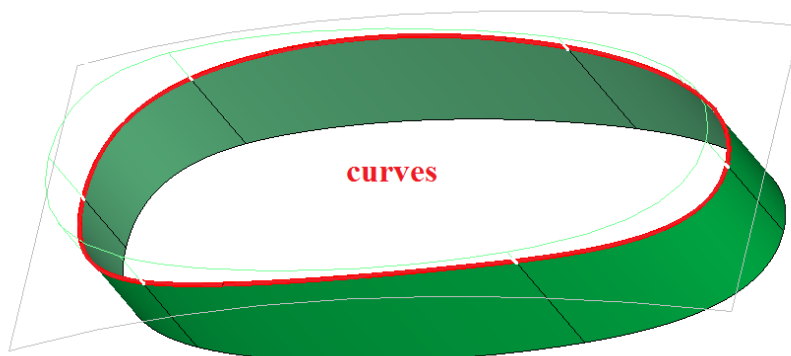


Рис. М.1.12.3.

Если после резки в краевых ребрах сохранилась информация о режущей поверхности, то при построении заплатки может быть использована режущая поверхность. На рис. М.1.12.4 приведен пример построения заплатки в виде некоторой части режущей поверхности, которая образовала краевые ребра, приведенные на рис. М.1.12.3.

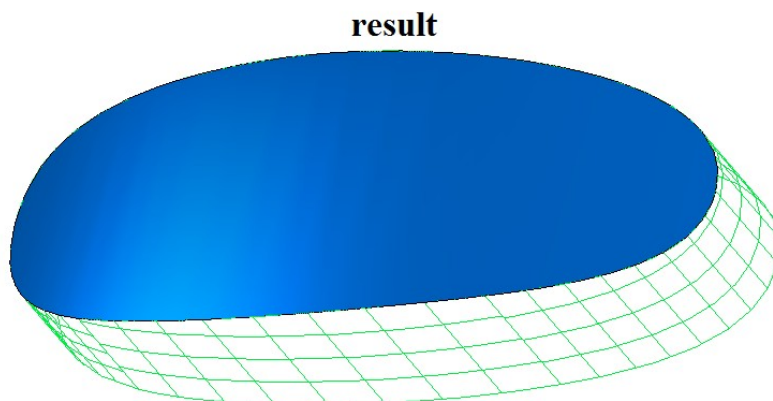


Рис. М.1.12.4.

Как правило, кривые для множества **curves** берутся от краевых ребер. После построения заплатку часто сшивают с другими телами, примыкающими к ее краям.

Метод **PatchShell** добавляет в журнал построенного тела строитель **MbPatchCreator**. Строитель **MbPatchCreator** объявлен в файле `sr_patch_creator.h`.

М.1.13. Сшивка граней тел

Метод
MbeStitchResType
StitchToOneSheetSolid (`const RPAArray<constMbSolid> & solids,`
`const MbSNameMaker & names,`
`bool formSolidBody,`
`double stitchPrecision,`
`MbSolid *& result`)

выполняет сшивку граней нескольких тел в одно тело.

Входными параметрами метода являются:

- **solids** – сшиваемые тела,
- **names** – именованье операции,
- **formSolidBody** – флаг управления сшивкой и обработкой вложенности,
- **stitchPrecision** – точность сшивки.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_shell.h`.

Метод выполняет поиск пар краевых ребер, соответствующих друг другу по длине и расположению, и строит кривую пересечения граней найденных ребер на заданном участке. После успешного построения кривой пересечения пара краевых ребер заменяется одним ребром, соединяющим смежные грани тел. При необходимости выполняется переориентация внешней стороны некоторых граней. Параметр **stitchPrecision** определяет максимальное расстояние, которое не должны превышать соответствующие точки сшиваемых ребер. Множество **solids** может содержать любое конечное число тел. Параметр **names** обеспечивает именованье сшитых ребер тела. Если параметр **formSolidBody=true**, то построенное тело **result** проверяется на отсутствие краевых ребер, выполняется поиск вложенных друг в друга оболочек и переориентация внутренних оболочек при необходимости. Если параметр **formSolidBody=false**, то в построенном теле **result** допускается присутствие краевых ребер, а определение вложенности оболочек не выполняется.

На рис. М.1.13.1 приведены два сшиваемых тела **solids[0]** и **solids[1]**. На рис. М.1.13.2 приведено тело **result**, причем после сшивки сшитые ребра были скруглены.

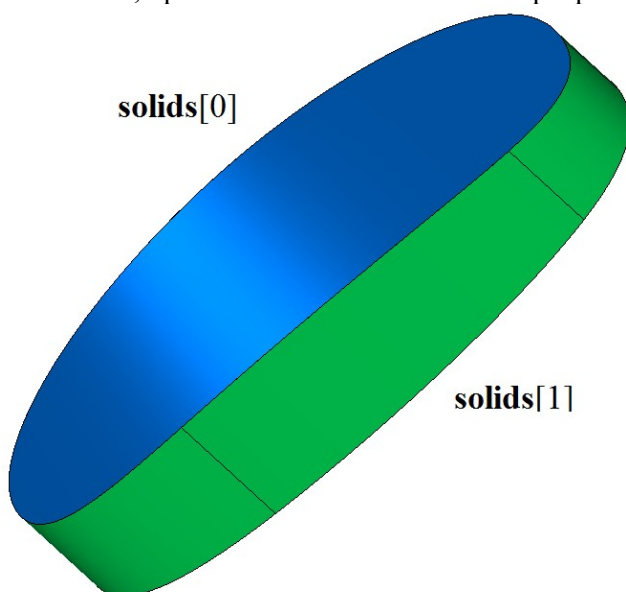


Рис. М.1.13.1.

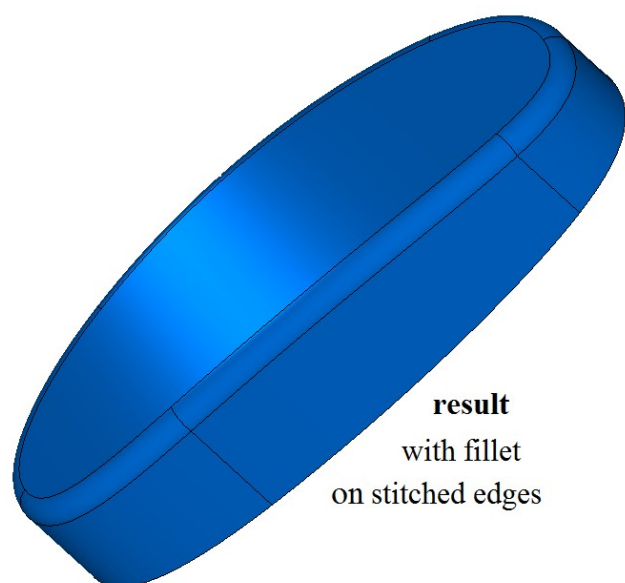


Рис. М.1.13.2.

Метод **StitchToOneSheetSolid** добавляет в журнал построенного тела строитель **MbStitchedSolid**, который содержит все необходимые данные для построения тела. Строитель **MbStitchedSolid** объявлен в файле `cg_stitch_solid.h`.

М.1.14. Построение тела по кривым или точкам кривых

Метод
MbResultType
LoftedShell (const RPAArray<**MbCurve3D**> & **curves**,
const **MbSNameMaker** & **names**,
SimpleName **name**,
MbSolid *& **result**)

выполняет построение тела, состоящего из одной грани, поверхность которой проходит по семейству кривых.

Входными параметрами метода являются:

- **curves** – множество кривых,
- **names** – именователъ граней,
- **name** – идентификатор.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_shell.h`.

Поверхность построенного по тела проходит по всем кривым **curves**. На форму поверхности тела влияет не только форма и взаимное расположение кривых, но и их ориентация и положение начал. Для предотвращения самопересечения и перекручивания поверхности тела соседние кривые множества **curves** должны иметь одинаковое направление, а начальные точки замкнутых кривых должны иметь близкое расположение. Если все кривые **curves** замкнуты, то поверхность построенного тела будет замкнута по одному из параметрических направлений.

Параметры **names** и **name** обеспечивают именование грани построенного тела.

На рис. М.1.14.1 приведено множество кривых **curves**.

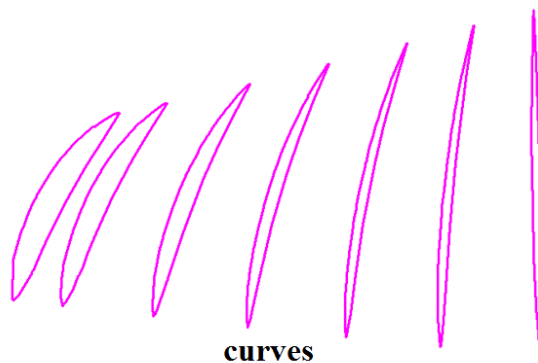


Рис. М.1.14.1.

На рис. М.1.14.2 приведено тело, построенное по кривым, приведенным на рис. М.1.14.1.

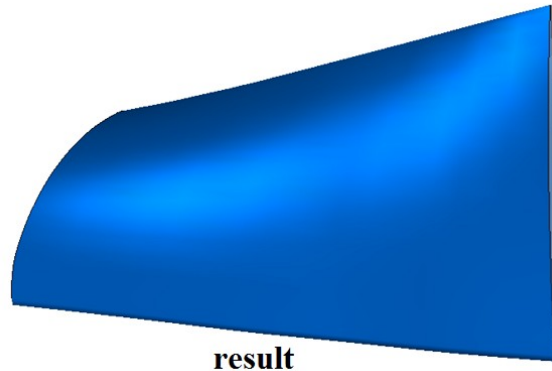


Рис. М.1.14.2.

```

Метод
MbResultType
LoftedShell ( const RPAArray< SArray<MbCartPoint3D> > & points,
               const MbSNameMaker & names,
               SimpleName name,
               MbSolid *& result )

```

выполняет построение незамкнутого тела из одной грани, поверхность которой проходит по контрольным точкам.

Входными параметрами метода являются:

- **points** – группы контрольных точек,
- **names** – именователи граней,
- **name** – идентификатор.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_shell.h`.

Контрольные точки сгруппированы по контейнерам `SArray`. Перед построением тела выполняется построение сплайновых кривых по множеству точек, расположенных в каждом контейнере `SArray`. Далее выполняется построение тела по семейству кривых описанным выше методом **LoftedShell**. Поверхность построенного тела проходит по всем точкам **points**. На форму поверхности тела влияет не только взаимное расположение точек, но и их последовательность в группах. Для предотвращения самопересечения и перекручивания поверхности тела последовательности точек в соседних группах должны иметь одинаковое направление, а начальные точки в соседних группах должны иметь близкое расположение. Для построения замкнутой по одному из параметрических направлений поверхности первые и последние точки в каждом контейнере `SArray` должны совпадать.

Параметры `names` и `name` обеспечивают именование грани построенного тела.

На рис. М.1.14.3 приведено множество контрольных точек **points**, группы которых имеют разный цвет.

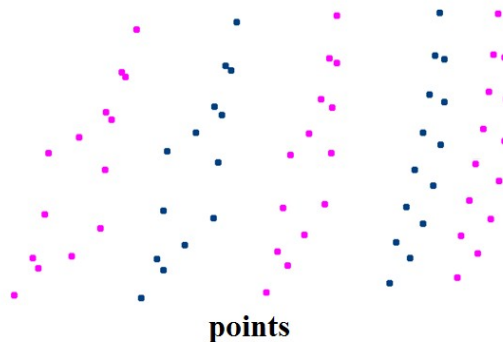


Рис. М.1.14.3.

На рис. М.1.14.4 приведено тело и контрольные точки, по которым построено тело.

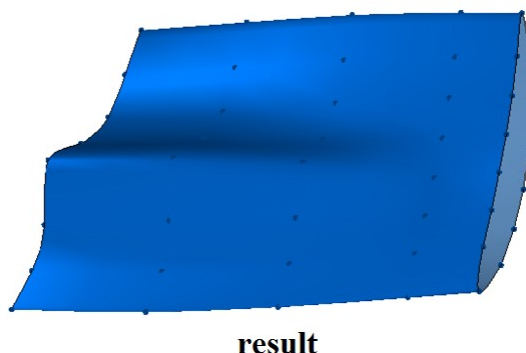


Рис. М.1.14.4.

Метод **LoftedShell** добавляет в журнал построенного тела строитель **MbThinShellCreator**, который содержит все необходимые данные для построения тела. Строитель **MbThinShellCreator** объявлен в файле `cr_thin_sheet.h`.

М.1.15. Построение эквидистантного тела

Метод
MbResultType
OffsetShell (**MbSolid** & **solid**,
MbcCopyMode *sameShell*,
RPAArray<**MbFace**> & **faces**,
bool *checkFacesConnection*,
SweptValues & *params*,
const MbSNameMaker & *names*,
MbSolid *& **result**)

выполняет построение эквидистантного незамкнутого тела для множества граней.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **faces** – грани исходного тела,
- *checkFacesConnection* – флаг проверки связности выбранных граней,
- *params* – параметры построения,
- *names* – именователи граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_shell.h`.

Метод строит тело, грани которого эквидистантны граням **faces** исходного тела **solid**. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**. Параметр *params* содержит информацию о расстоянии построенных граней от граней исходного тела. Расстояние может быть равным *params.thickness1* в положительном направлении нормали граней или *params.thickness2* в отрицательном направлении нормали граней исходного тела. Если *params.shellClosed=false*, то будет построено незамкнутое тело. Параметр *names* обеспечивает именование граней построенного тела.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление **MbcCopyMode** описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#)

На рис. М.1.15.1 приведено тело, на базе указанных граней которого будет построено эквидистантное тел.

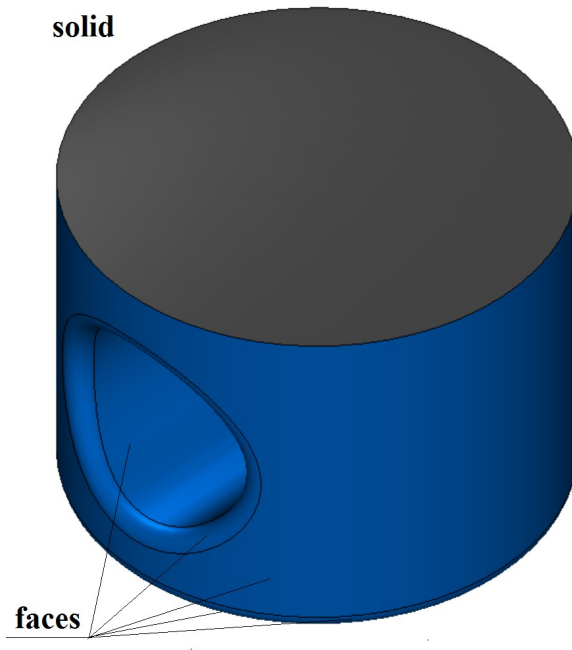
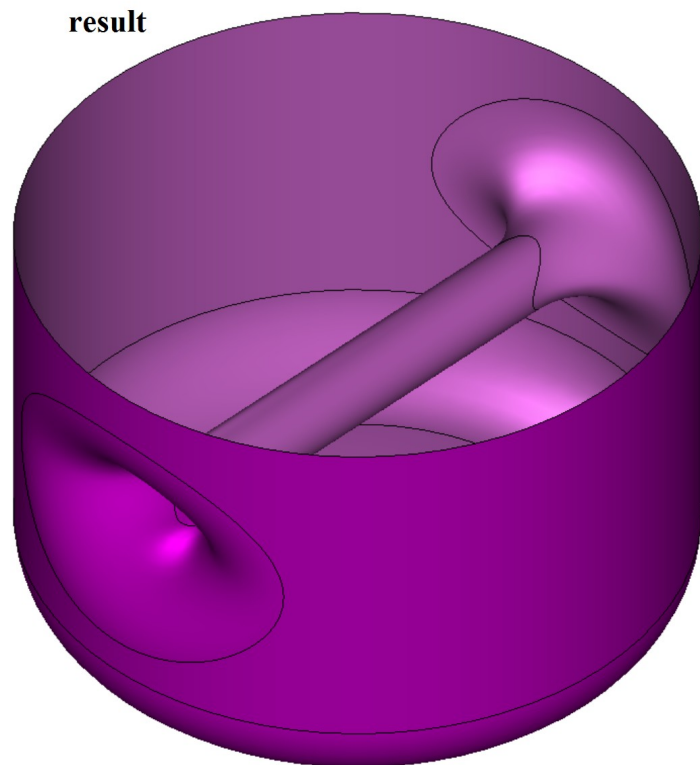


Рис. М.1.15.1.

На рис. М.1.15.2 приведено построенное эквидистантное тело.



$params.thickness1 > 0$ $params.thickness2 = 0$

Рис. М.1.15.2.

Если множество граней **faces** будет содержать все грани исходного тела **solid**, то построенное тело **result** будет эквидистантным исходному телу.

Метод **OffsetShell** добавляет в журнал построенного тела строитель MbShellSolid, который содержит все необходимые данные для построения тела. Строитель MbShellSolid объявлен в файле cr_thin_shell_solid.h.

Тестовое приложение test.exe выполняет построение эквидистантного незамкнутого тела командой меню «Создать->Оболочку->На базе оболочки->Эквидистантную к граням».

М.1.16. Продление грани тела

Метод

MbResultType

ExtensionShell ([MbSolid](#) & **solid**,
MbcCopyMode *sameShell*,
[MbFace](#) & **face**,
const RPAArray<[MbCurveEdge](#)> & **edges**,
const ExtensionValues & *params*,
const MbSNameMaker & *names*,
[MbSolid](#) * & **result**)

выполняет построение продолжения незамкнутого тела путём продления указанной грани крайних рёбер заданной грани тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **face** – грань исходного тела,
- **edges** – множество ребер грани, которые продляются выдавливанием,
- *params* – параметры построения,
- *names* – именователь граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле *action_shell.h*. Метод продлевает грань **face** со стороны крайних ребер **edges**. Грань **face** входит в состав тела **solid**. Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbcCopyMode описано в параграфе [0.7.9. Копирование множества граней MbFaceShell](#).

Параметр *params* управляет способами продления грани и содержит данные, приведенные на рис. М.1.16.1.

ExtensionValues	
ExtentionType	type = { et_same, et_tangent, et_direction }
ExtentionWay	way = { ew_distance, ew_vertex, ew_surface }
LateralKind	kind = { le_normal, le_prolong }
MbCartPoint3D	point
MbVector3D	direction
double	<i>distance</i>
bool	<i>prolong</i>
bool	<i>combine</i>
MbFaceShell *	shell
MbItemIndex	faceIndex

Рис. М.1.16.1.

Продление грани может быть трех типов.

Тип продления определяет параметр *params.type*. При *params.type=et_same* продлевается исходная грань **face** путем расширения области определения параметров ее поверхности и смещения краевых ребер **edges**. При *params.type=et_tangent* к грани **face** пристыковываются новые линейчатые грани, гладко стыкующиеся с гранью **face** через ребра **edges**. При *params.type=et_direction* к грани **face** через ребра **edges** пристыковываются новые грани, полученные выдавливанием кривых ребер **edges** в направлении *params.direction*.

Расстояние продления края грани **face** определяет параметр *params.way*. При *params.way=ew_distance* продление выполняется на расстояние *params.distance*. При *params.way=ew_vertex* продление выполняется на расстояние, определяемое точкой *params.point*. При *params.way=ew_surface* продление выполняется до оболочки *params.shell*. Параметр *params.faceIndex* определяет ближайшую к грани **face** грань оболочки *params.shell*. Если *params.way!=ew_surface*, то *params.shell* может быть равен нулю.

Параметр *params.kind* определяет способ обрезки краев граней продления в начале и в конце каждого ребра множества **edges**. Если *params.kind=le_normal*, то край продления грани **face** обрезается по нормали к соответствующему ребру множества **edges**. Если *params.kind=le_prolong*, то край продления грани **face** определяется соседними к ребрам множества **edges** ребрами.

Параметр *names* обеспечивает именование граней построенного тела.

На рис. М.1.16.2 приведена грань тела, два краевых ребра которого будут продлены.

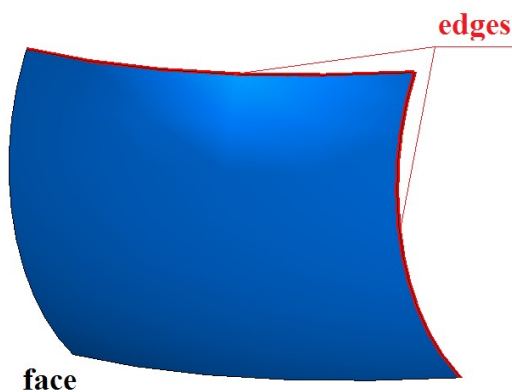


Рис. М.1.16.2.

На рис. М.1.16.3 и М.1.16.4 приведены результаты продления грани путем расширения области определения параметров поверхности с разными значениями параметра *params.kind*.

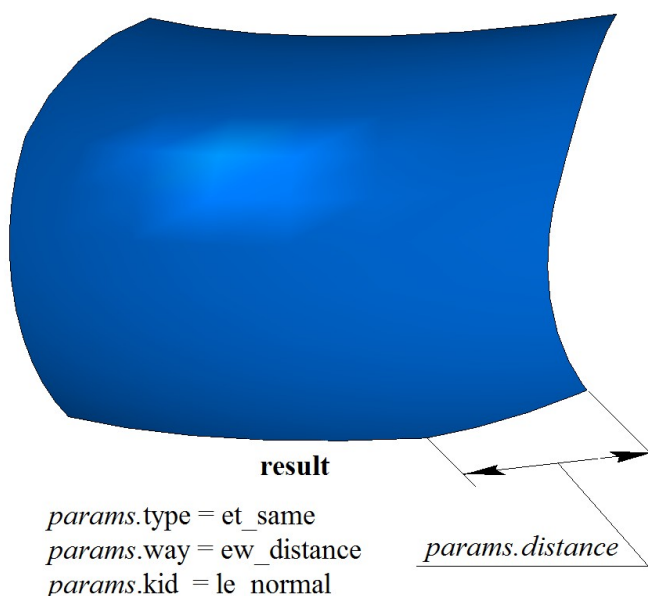


Рис. М.1.16.3.

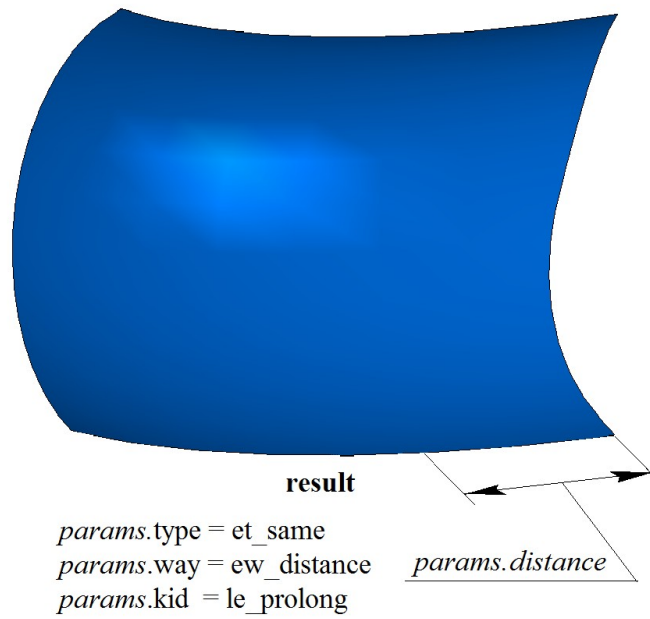


Рис. М.1.16.4.

На рис. М.1.16.5 приведен результат продления грани **face** добавлением новых граней, гладко стыкующихся с гранью **face**. На рис. М.1.16.6 приведен результат продления грани **face** в заданном направлении.

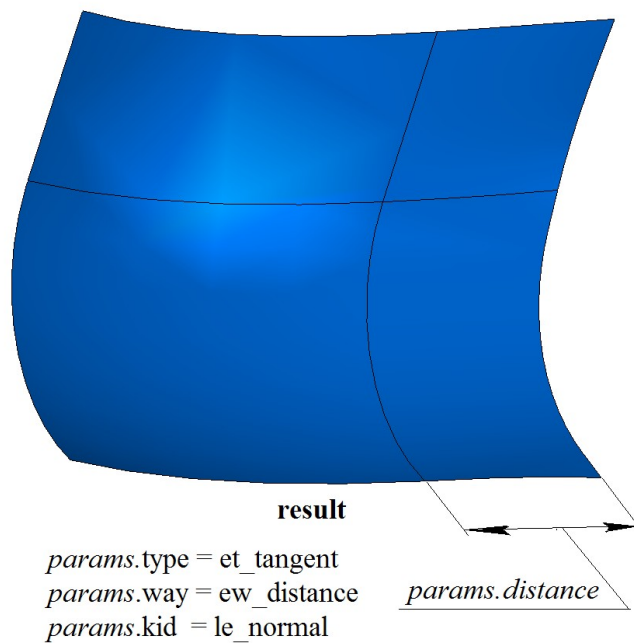


Рис. М.1.16.5.

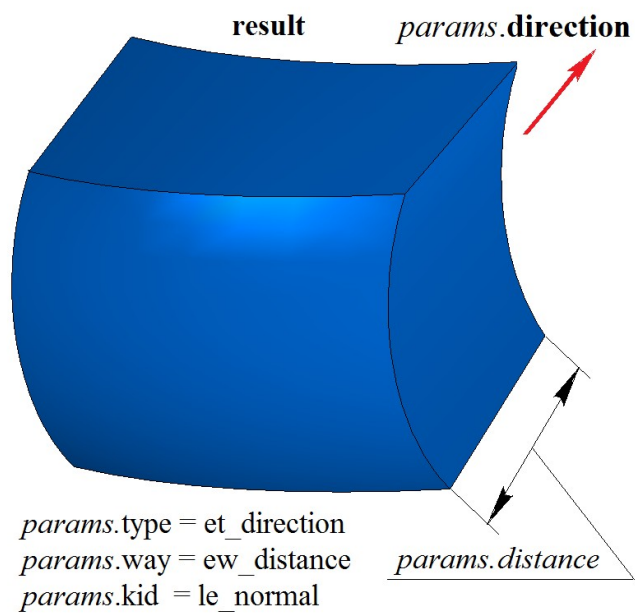


Рис. М.1.16.6.

Метод **ExtensionShell** добавляет в журнал построенного тела строитель MbExtensionShell, который содержит все необходимые данные для построения тела. Строитель MbExtensionShell объявлен в файле `cr_extension_shell.h`.

М.2. ВЫПОЛНЕНИЕ ОПЕРАЦИЙ НАД ТЕЛАМИ

Один из подходов к построению тел в геометрическом моделировании похож на процесс изготовления моделируемого объекта. Сначала создаются тела простой формы, а далее выполняется набор действий, позволяющих из тел простой формы получить более сложные тела. Более сложные тела строятся путем выполнения операций над построенными телами. Операции фиксируются в журнале построения. Для замкнутых и незамкнутых тел одни и те же операции могут приводить к разным результатам

М.2.1. Булева операция над телами

Метод
MbResultType
BooleanResult ([MbSolid](#) & **solid1**,
MbcCopyMode *sameShell1*,
[MbSolid](#) & **solid2**,
MbcCopyMode *sameShell2*,
OperationType *oType*,
const MbSNameMaker & *names*,
bool *mergeFaces*,
bool *closed*,
[MbSolid](#) *& **result**)

выполняет построение нового тела путем булевой операции над двумя заданными телами.

Входными параметрами метода являются:

- **solid1** – первое тело для булевой операции,
- *sameShell1* – вариант копирования первого тела,
- **solid2** – второе тело для булевой операции,
- *sameShell2* – вариант копирования второго тела,
- *oType* – тип булевой операции: `bo_Union` – объединение тел,
`bo_Intersect` – пересечение тел,
`bo_Difference` – вычитание тел,
- *names* – именователь (используется для версионирования),
- *mergeFaces* – объединять ли подобные грани,
- *closed* – необходима ли проверка замкнутости построенного тела.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_solid.h`.

Метод выполняет операции объединения, пересечения и вычитания точек двух тел **solid1** и **solid2**. Параметры *sameShell1* и *sameShell2* управляют передачей граней, ребер и вершин от исходных тел **solid1** и **solid2** построенному телу **result**.

Параметры *sameShell1* и *sameShell2* могут принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbcCopyMode описано в параграфе [O.7.9. Копирование множества граней MbFaceShell](#).

Параметр OperationType *oType* определяет тип булевой операции и принимает три значения: `bo_Union`, `bo_Intersect`, `bo_Difference`. При *oType=bo_Union* рассматриваемый метод выполняет объединение тел **solid1** и **solid2**, при *oType=bo_Intersect* рассматриваемый метод выполняет пересечение тел **solid1** и **solid2**, при *oType=bo_Difference* рассматриваемый метод выполняет вычитание из тела **solid1** тела **solid2**.

Параметр *names* используется для версионирования булевой операции.

Параметр *mergeFaces* управляет объединением подобных граней. При *mergeFaces==false* подобные грани не объединяются.

Параметр *closed* работает только для незамкнутых тел и сообщает операции о необходимости выполнить проверку результата на замкнутость. Для незамкнутых тел булева операция выполняется методом **BooleanShell**, описанным в параграфе [М.2.2. Булева операция над незамкнутыми телами](#).

На рис. М.2.1.1 приведены исходные тела-операнды **solid1** и **solid2**.

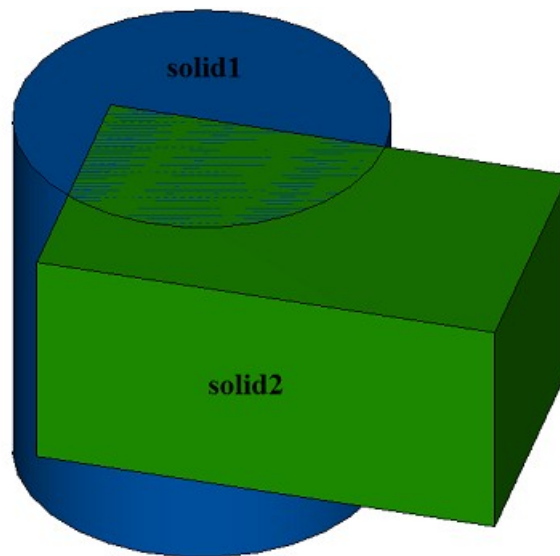
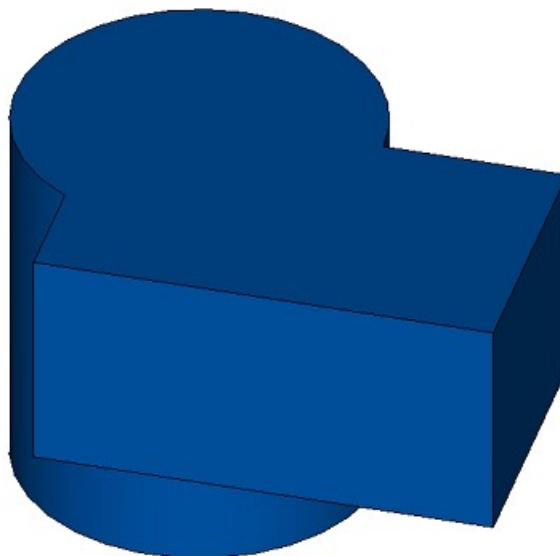


Рис. М.2.1.1.

На рис. М.2.1.2 приведен результат булевой операции объединения тел **solid1** и **solid2**, показанных на рис. М.2.1.1.

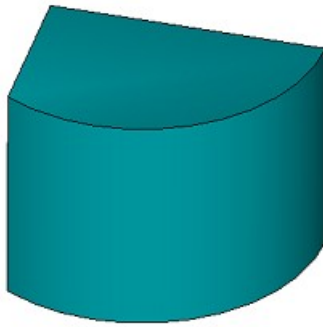


oType = bo_Union

solid1 + solid2

Рис. М.2.1.2.

На рис. М.2.1.3 приведен результат булевой операции пересечения тел **solid1** и **solid2**, показанных на рис. М.2.1.1.

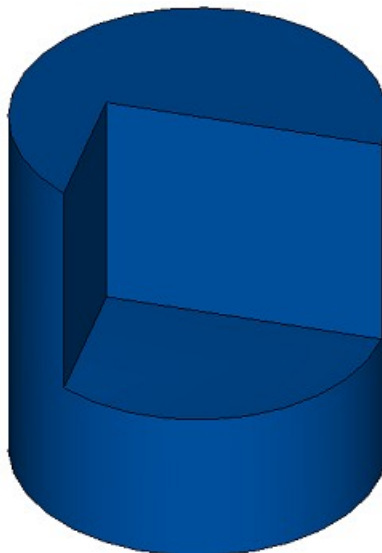


oType = bo_Intersect

solid1&solid2

Рис. М.2.1.3.

На рис. М.2.1.4 приведен результат булевой операции вычитания тела **solid2** из тела **solid1**, показанных на рис. М.2.1.1.

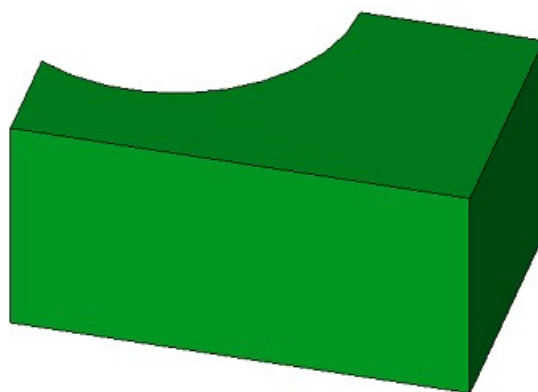


oType = bo_Difference

solid1 - solid2

Рис. М.2.1.4.

На рис. М.2.1.5 приведен результат булевой операции вычитания тела **solid1** из тела **solid2**, показанных на рис. М.2.1.1.



$oType = bo_Difference$

solid2 - solid1

Рис. М.2.1.5.

Для демонстрации работы параметра *mergeFaces* рассмотрим булевы операции над исходными телами **solid1** и **solid2**, приведенными на рис. М.2.1.6.

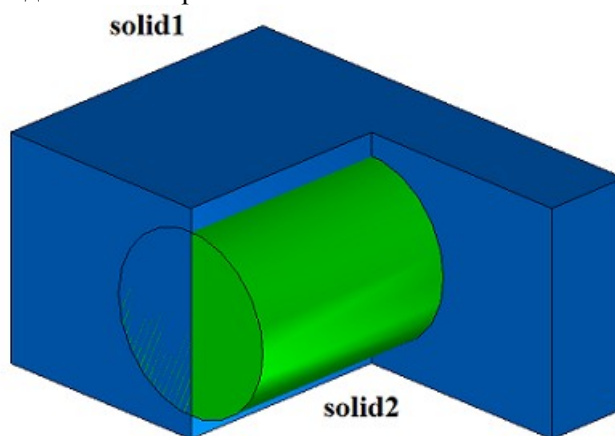
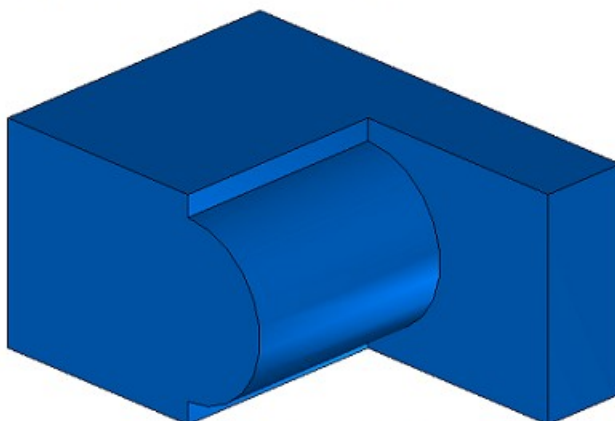


Рис. М.2.1.6.

На рис. М.2.1.7 приведен результат **result** объединения тел **solid1** и **solid2** при работе рассматриваемого метода с параметром *mergeFaces*==true. На рис. М.2.1.8 приведен результат **result** объединения тел **solid1** и **solid2** при работе рассматриваемого метода с параметром *mergeFaces*==false. Совпадающие грани на рис. М.2.1.8 не объединены.

$oType = bo_Union$ *mergeFaces = true*



solid1 + solid2

Рис. М.2.1.7.

oType = bo_Union *mergeFaces* = false

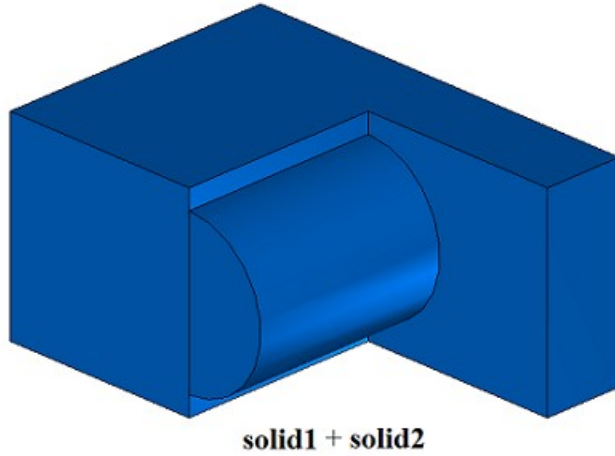


Рис. М.2.1.8.

На рис. М.2.1.9 приведен результат **result** вычитания тела **solid2** из тела **solid1** при работе рассматриваемого метода с параметром *closed*== true. На рис. М.2.1.10 приведен результат **result** вычитания тела **solid2** из тела **solid1** при работе рассматриваемого метода с параметром *closed*==false. Совпадающие грани на рис. М.2.1.10 не объединены.

oType = bo_Difference *mergeFaces* = true

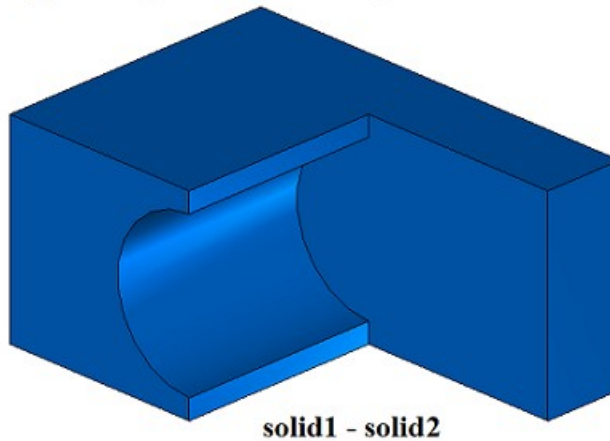
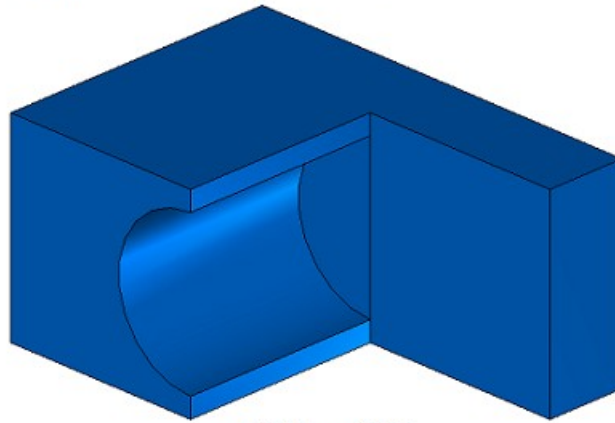


Рис. М.2.1.9.

oType = bo Difference *mergeFaces* = false



solid1 - solid2

Рис. М.2.1.10.

Метод
MbResultType
BooleanSolid ([MbSolid](#) & **solid1**,
MbeCopyMode *sameShell1*,
[MbSolid](#) & **solid2**,
MbeCopyMode *sameShell2*,
OperationType *oType*,
const MbSNameMaker & names,
[MbSolid](#) *& **result**)

выполняет те же действия, что и метод **BooleanResult**, полагая значения параметров *mergeFaces*==true и *closed*==true. Метод **BooleanSolid** работает только с замкнутыми телами.

Методы **BooleanResult** и **BooleanSolid** добавляют в журнал построенного тела строитель MbBooleanSolid, который содержит все необходимые данные для выполнения операции. Строитель MbBooleanSolid объявлен в файле `sr_boolean_solid.h`.

Тестовое приложение `test.exe` выполняет булевы операции над телами командами меню «Создать->Тело->Приклеиванием к телу->Тела», «Создать->Тело->Вырезанием из тела->Тела», «Создать->Тело->Пересечением с телом->Тела».

М.2.2. Булева операция над незамкнутыми телами

Метод
MbResultType
BooleanShell ([MbSolid](#) & **solid1**,
MbeCopyMode *sameShell1*,
[MbSolid](#) & **solid2**,
MbeCopyMode *sameShell2*,
OperationType *oType*,
const MbSNameMaker & names,
[MbSolid](#) *& **result**)

выполняет построение нового тела путем булевой операции над двумя заданными незамкнутыми телами.

Входными параметрами метода являются:

- **solid1** – первое тело для булевой операции,
- *sameShell1* – вариант копирования первого тела,
- **solid2** – второе тело для булевой операции,
- *sameShell2* – вариант копирования второго тела,
- *oType* – тип булевой операции: `bo_Variety` – объединение тел,
`bo_Internal` – пересечение тел,

bo_External – вычитание тел,

- names – именователь (используется для версионирования).

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает rt_Success, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле action_solid.h.

Метод выполняет операции объединения, пересечения и вычитания точек двух незамкнутых тел **solid1** и **solid2**. Параметры *sameShell1* и *sameShell2* управляют передачей граней, ребер и вершин от исходных тел **solid1** и **solid2** построенному телу **result**.

Параметры *sameShell1* и *sameShell2* могут принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbeCopyMode описано в параграфе [O.7.9. Копирование множества граней MbFaceShell](#).

Параметр OperationType *oType* определяет тип булевой операции и принимает три значения: bo_Variety, bo_Internal, bo_External. При *oType*=bo_Variety рассматриваемый метод выполняет объединение поверхностей тел **solid1** и **solid2**, при *oType*=bo_Internal рассматриваемый метод выполняет пересечение поверхностей тел **solid1** и **solid2**, при *oType*=bo_External рассматриваемый метод выполняет вычитание из тела **solid1** тела **solid2**. Параметр names используется для версионирования булевой операции.

На рис. М.2.2.1 приведены исходные незамкнутые тела **solid1** и **solid2**.

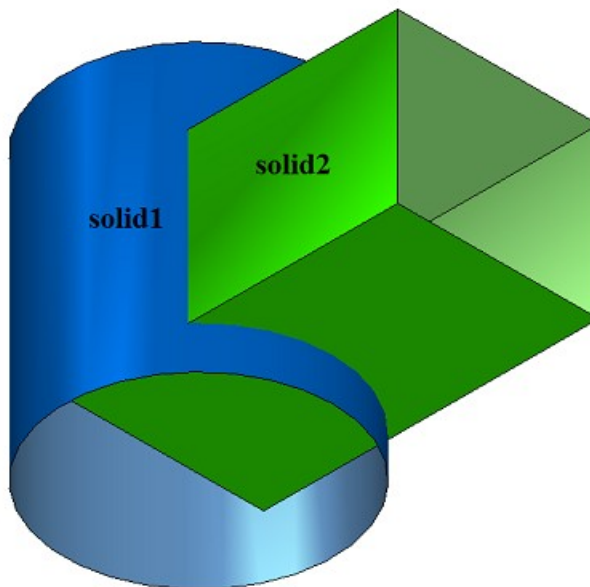
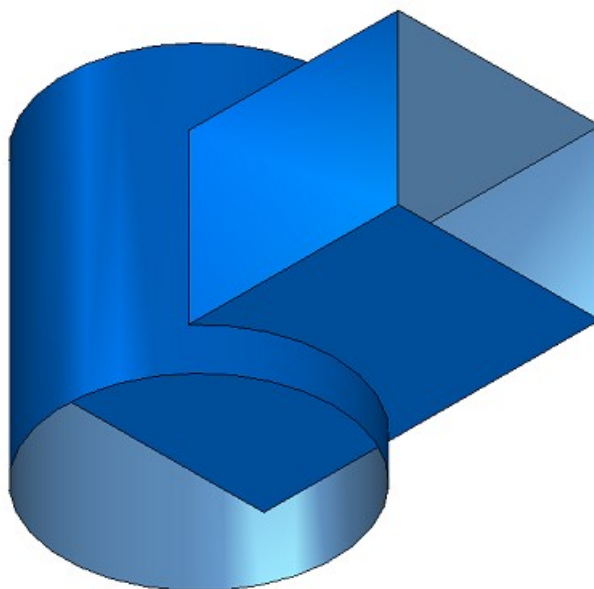


Рис. М.2.2.1.

На рис. М.2.2.2 приведен результат булевой операции объединения незамкнутых тел **solid1** и **solid2**, показанных на рис. М.2.2.1.

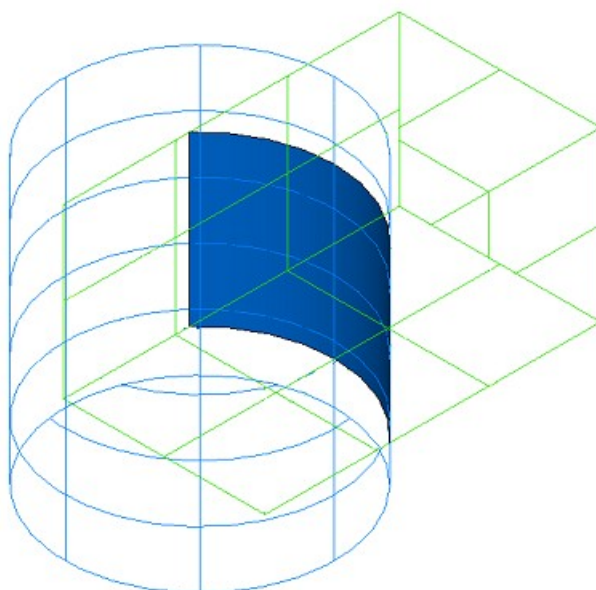


oType = bo_Variety

solid1 + solid2

Рис. М.2.2.2.

На рис. М.2.2.3 приведен результат булевой операции усечения незамкнутых тел **solid1** и **solid2**, показанных на рис. М.2.2.1.

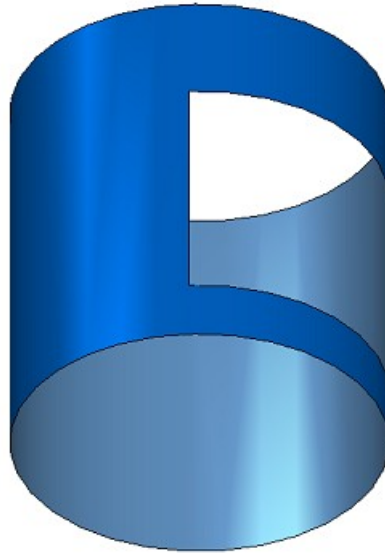


oType = bo_Internal

solid1&solid2

Рис. М.2.2.3.

На рис. М.2.2.4 приведен результат булевой операции вычитания незамкнутого тела **solid2** из незамкнутого тела **solid1**, показанных на рис. М.2.2.1.

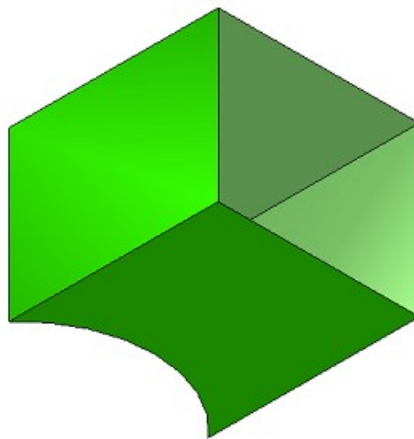


oType = bo_External

solid1 - solid2

Рис. М.2.2.4.

На рис. М.2.2.5 приведен результат булевой операции вычитания незамкнутого тела **solid1** из незамкнутого тела **solid2**, показанных на рис. М.2.2.1.



oType = bo_External

solid2 - solid1

Рис. М.2.2.5.

Метод работает с незамкнутыми телами, хотя вторым операндом может быть замкнутое тело. Метод выполняет одноимённую булеву операцию над множествами точек, расположенными на поверхности тел.

Метод **BooleanShell** добавляет в журнал построенного тела строитель MbBooleanSolid, который содержит все необходимые данные для выполнения операции. Строитель MbBooleanSolid объявлен в файле `sr_boolean_solid.h`.

Тестовое приложение `test.exe` выполняет булевы операции над телами командами меню «Создать->Оболочку->На основе оболочки->Соединением с оболочкой», «Создать->Оболочку->На основе

оболочки->Вычитанием оболочки», «Создать->Оболочку->На основе оболочки->Ограничением оболочкой».

М.2.3. Булева операция с телом выдавливания

Метод

MbResultType

ExtrusionResult ([MbSolid](#) & **solid**,
MbcCopyMode *sameShell*,
const MbSweptData & **sweptData**,
const [MbVector3D](#) & **direction**,
ExtrusionValues & *params*,
OperationType *oType*,
const MbSNameMaker & *names*,
PArray<MbSNameMaker> & *snames*,
[MbSolid](#) *& **result**)

выполняет построение тела выдавливания и булеву операцию заданного тела с построенным телом.

Входными параметрами метода являются:

- **solid** – заданное тело для булевой операции,
- *sameShell* – вариант копирования заданного тела,
- **sweptData** – данные об образующих кривых для построения тела выдавливания,
- **direction** – направление выдавливания,
- *params* – параметры построения,
- *oType* – тип булевой операции: bo_Union – объединение тел,
bo_Intersect – пересечение тел,
bo_Difference – вычитание тел,
- *names* – именователь операции,
- *snames* – именователи граней тела выдавливания.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле *action_solid.h*.

Метод представляет собой последовательное объединение двух методов: метода **ExtrusionSolid**, выполняющего построение тела путем выдавливания кривых **sweptData** по заданным параметрам *params* в направлении **direction**, и метода **BooleanSolid**, выполняющего булеву операцию *oType* тела **solid** с построенным на предыдущем шаге телом. Метод **ExtrusionSolid** описан в параграфе [М.1.3. Построение тела выдавливания](#), метод **BooleanSolid** описан в параграфе [М.2.1. Булева операция над телами](#). Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbcCopyMode описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

Параметр OperationType *oType* определяет тип булевой операции и принимает три значения: bo_Union, bo_Intersect, bo_Difference. При *oType*=bo_Union рассматриваемый метод выполняет объединение тел **solid** и тела выдавливания, при *oType*=bo_Intersect рассматриваемый метод выполняет пересечение тел **solid** и тела выдавливания, при *oType*=bo_Difference рассматриваемый метод выполняет вычитание из тела **solid** тела выдавливания. Параметры *names* и *snames* обеспечивают именование граней построенного тела.

Метод **ExtrusionResult** при построении тела путем выдавливания кривых имеет те же возможности, что и метод **ExtrusionSolid**: выдавливаемые кривые могут располагаться на плоскости (рис. М.1.3.2), криволинейной поверхности (рис. М.1.3.12) или в пространстве (рис. М.1.3.20); выдавливание может выполняться в прямом, обратном и в обоих направлениях; с уклоном и без уклона граней; тело может полностью заполнять замкнутые кривые (рис. М.1.3.13) или иметь тонкую стенку (рис. М.1.3.14). Мы не будем повторять описание всех возможностей рассматриваемого метода, а остановимся только на некоторых из них, связанных с булевыми операциями.

На рис. М.2.3.1 показаны тело **solid**, образующая кривая, входящая в данные **sweptData**, и направление выдавливания **direction**.

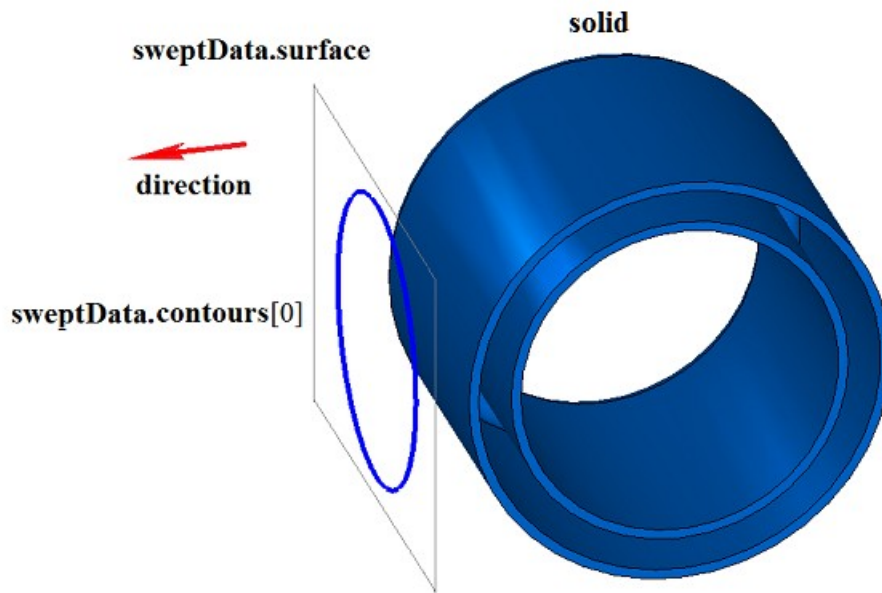


Рис. М.2.3.1.

На рис. М.2.3.2 приведен результат булевой операции объединения тела **solid** и тонкостенного тела, полученного выдавливанием образующей кривой **sweptData** по направлению **direction**, показанных на рис. М.2.3.1, на заданное расстояние.

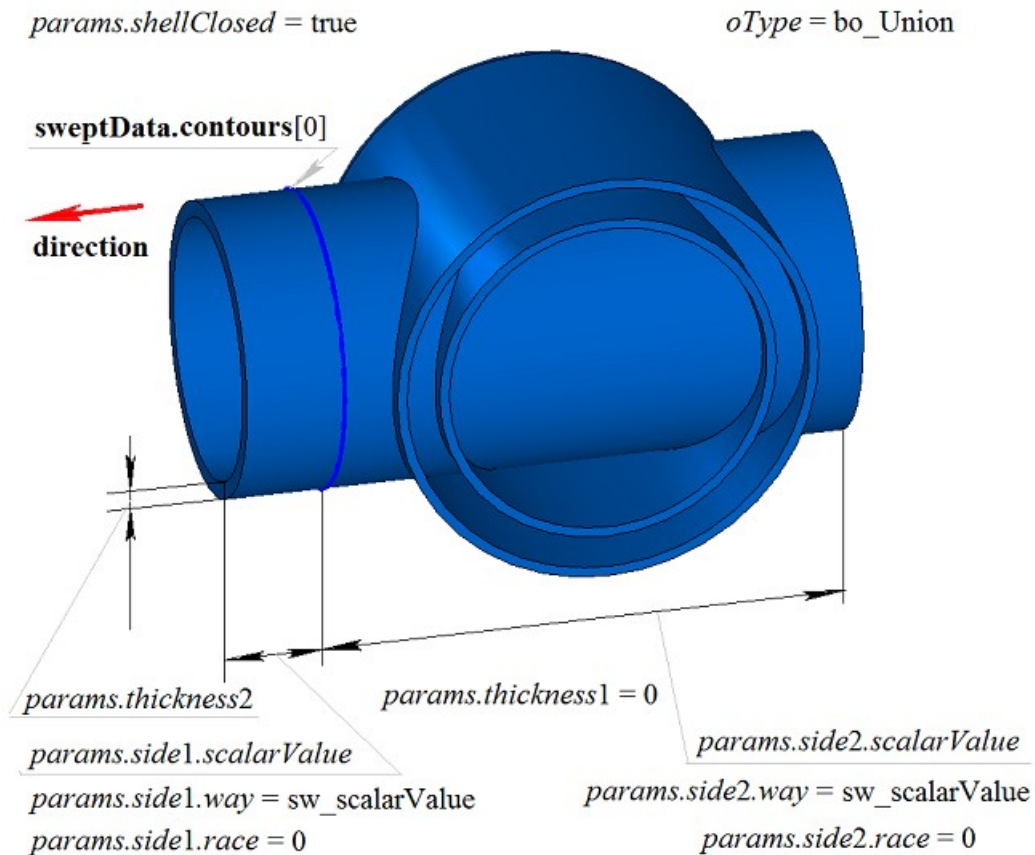


Рис. М.2.3.2.

На рис. М.2.3.3 приведен результат булевой операции объединения тела **solid** и тела, полученного выдавливанием образующей кривой **sweptData** по направлению **direction**, показанных на рис.

М.2.3.1. Выдавливание образующей кривой выполнено в обратном направлении без уклона с опцией «До ближайших объектов» (*params.side2.way=sw_shell*).

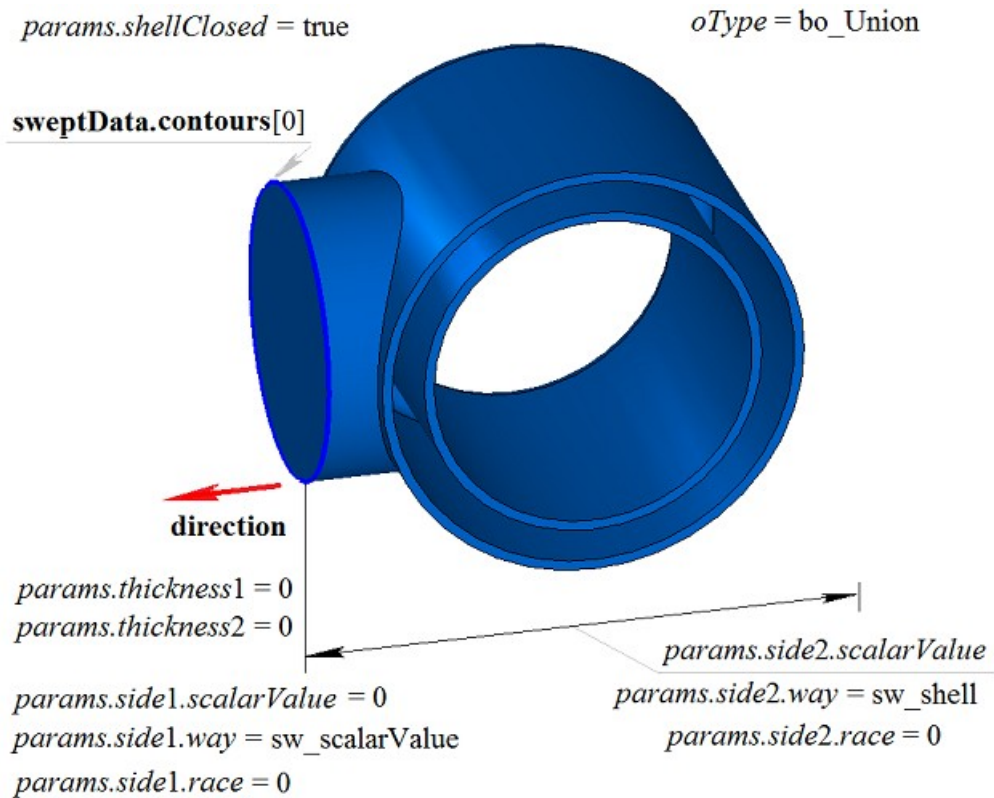


Рис. М.2.3.3.

На рис. М.2.3.4 приведен результат булевой операции вычитания из тела **solid** тела, полученного выдавливанием образующей кривой **sweptData** по направлению **direction**, показанных на рис. М.2.3.1.

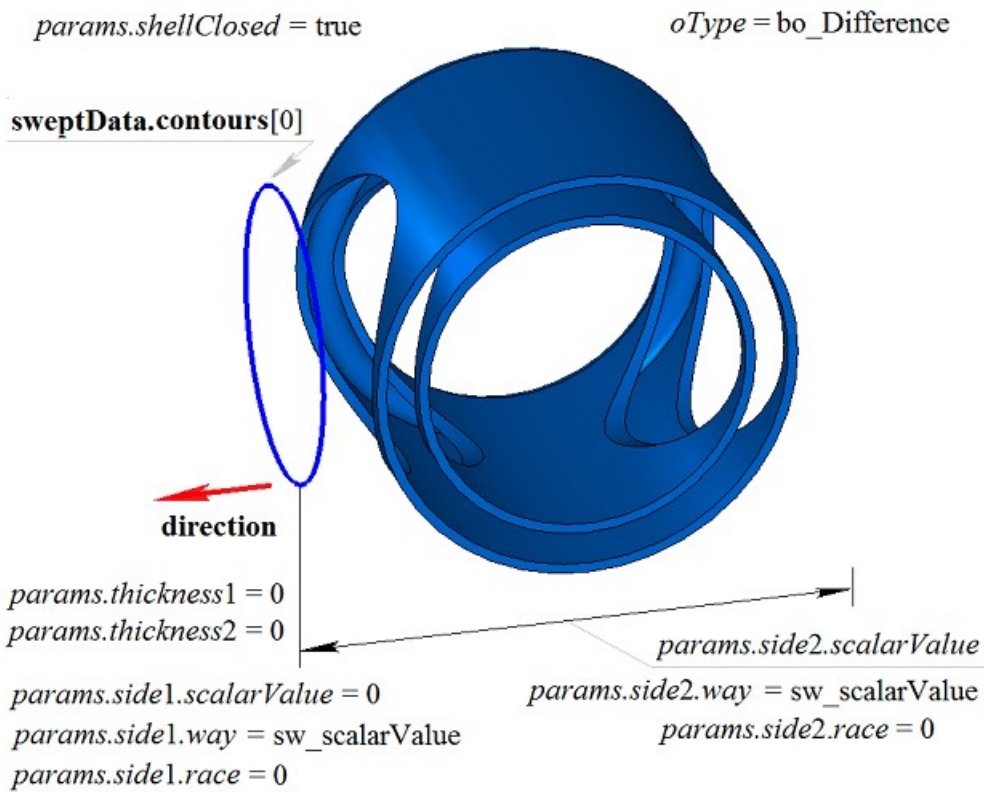


Рис. М.2.3.4.

На рис. М.2.3.5 приведен результат булевой операции вычитания из тела **solid** тела, полученного выдавливанием образующей кривой **sweptData** по направлению **direction**, показанных на рис. М.2.3.1. Выдавливание образующей кривой выполнено в обратном направлении без уклона с опцией «До ближайших объектов» (*params.side2.way=sw_shell*).

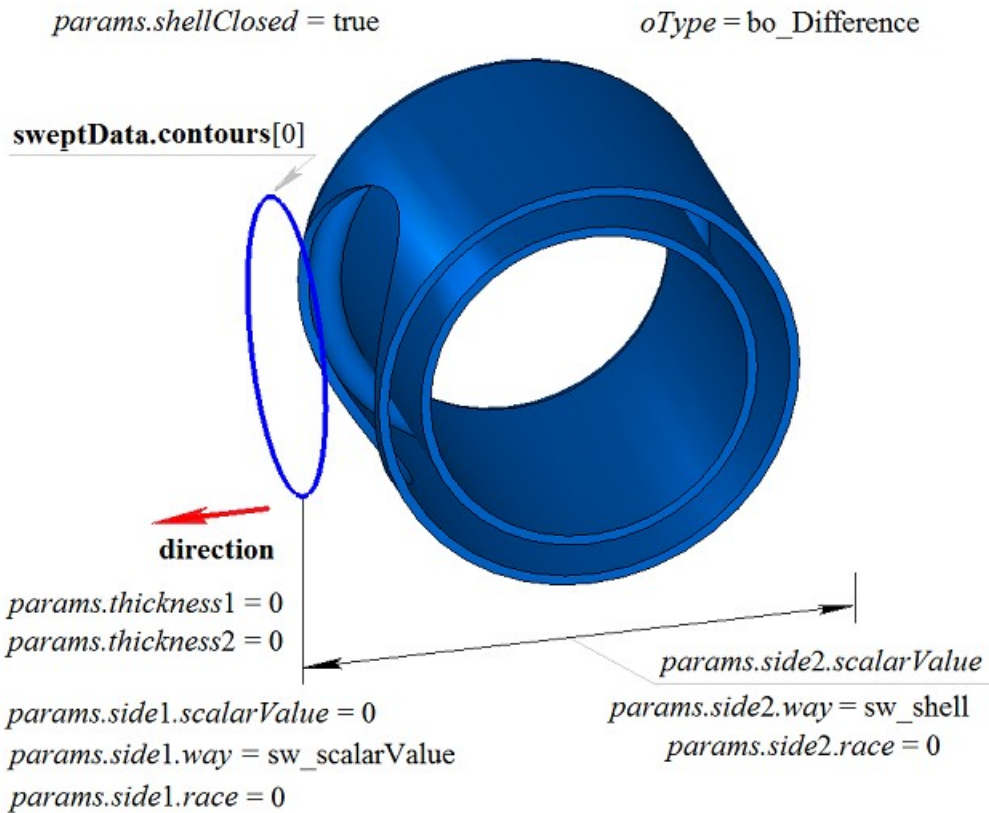


Рис. М.2.3.5.

На рис. М.2.3.6 приведен результат булевой операции пересечения тела **solid** и тела, полученного выдавливанием образующей кривой **sweptData** по направлению **direction**, показанных на рис. М.2.3.1.

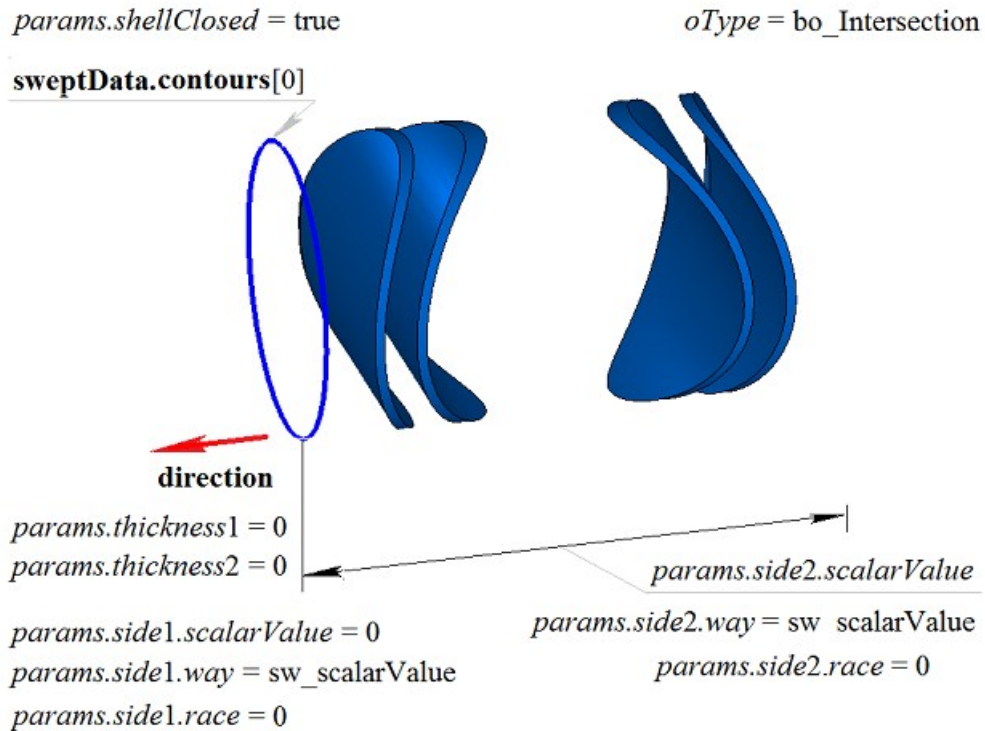


Рис. М.2.3.6.

На рис. М.2.3.7 приведен результат булевой операции пересечения тела **solid** и тела, полученного выдавливанием образующей кривой **sweptData** по направлению **direction**, показанных на рис. М.2.3.1. Выдавливание образующей кривой выполнено в обратном направлении без уклона с опцией «До ближайших объектов».

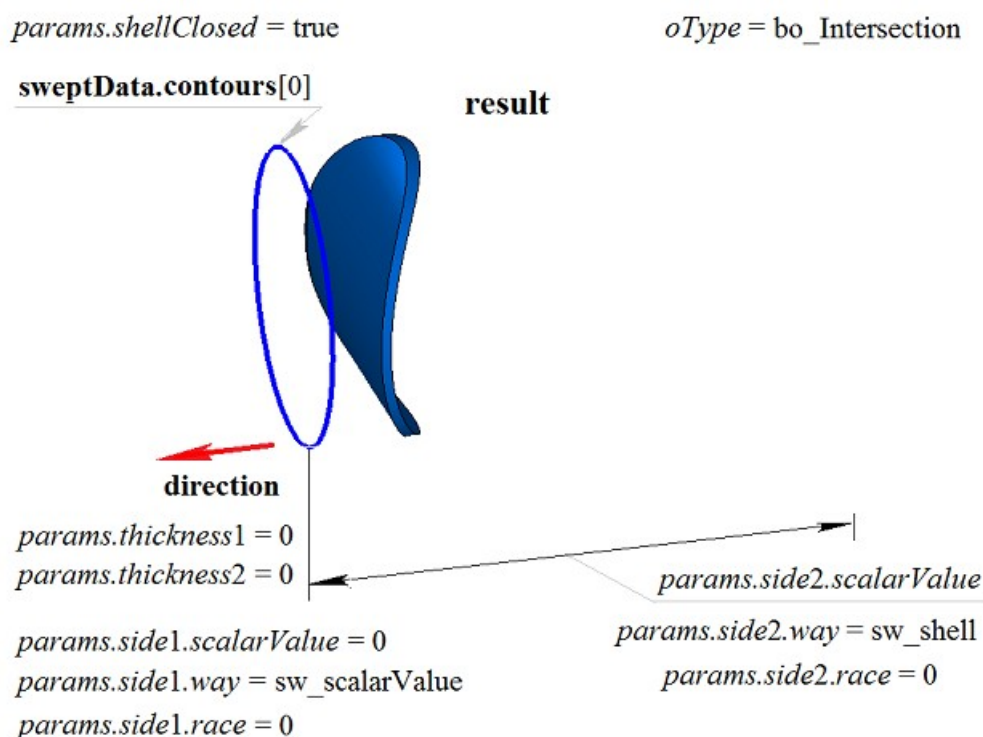


Рис. М.2.3.7.

Метод **ExtrusionResult** добавляет в журнал построенного тела строитель MbExtrusionSolid, который содержит все необходимые данные для выполнения операции. Строитель MbExtrusionSolid объявлен в файле `cg_extrusion_solid.h`.

Тестовое приложение `test.exe` выполняет булеву операцию с построенным телом выдавливания командами меню «Создать->Тело->Приклеиванием к телу->Выдавливанию кривой», «Создать->Тело->Вырезанием из тела->Выдавливанию кривой» и «Создать->Тело->Пересечением с телом->Выдавливанию кривой».

М.2.4. Булева операция с телом вращения

Метод
MbResultType
RevolutionResult ([MbSolid](#) & **solid**,
MbeCopyMode *sameShell*,
const MbSweptData & **sweptData**,
const MbAxis3D & **axis**,
RevolutionValues & *params*,
OperationType *oType*,
const MbSNameMaker & *names*,
PArray<MbSNameMaker> & *snames*,
[MbSolid](#) *& **result**)

выполняет построение тела вращения и булеву операцию заданного тела с построенным телом.

Входными параметрами метода являются:

- **solid** – заданное тело для булевой операции,
- *sameShell* – вариант копирования заданного тела,
- **sweptData** – данные об образующих кривых для построения тела выдавливания,

- **axis** – ось вращения,
- *params* – параметры построения,
- *oType* – тип булевой операции: bo_Union – объединение тел,
bo_Intersect – пересечение тел,
bo_Difference – вычитание тел,
- *names* – именователь операции,
- *snames* – именователи граней тела вращения.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Метод представляет собой последовательное объединение двух методов: метода **RevolutionSolid**, выполняющего построение тела путем выдавливания кривых **sweptData** по заданным параметрам *params* в направлении **direction**, и метода **BooleanSolid**, выполняющего булеву операцию *oType* тела **solid** с построенным на предыдущем шаге телом. Метод **RevolutionSolid** описан в параграфе [М.1.4. Построение тела вращения](#), метод **BooleanSolid** описан в параграфе [М.2.1. Булева операция над телами](#). Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление `MbCopyMode` описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

Параметр `OperationType oType` определяет тип булевой операции и принимает три значения: `bo_Union`, `bo_Intersect`, `bo_Difference`. При *oType=bo_Union* рассматриваемый метод выполняет объединение тел **solid** и тела вращения, при *oType=bo_Intersect* рассматриваемый метод выполняет пересечение тел **solid** и тела вращения, при *oType=bo_Difference* рассматриваемый метод выполняет вычитание из тела **solid** тела вращения. Параметры *names* и *snames* обеспечивают именование граней построенного тела.

Метод **RevolutionResult** при построении тела путем вращения кривых имеет те же возможности, что и метод **RevolutionSolid**: вращаемые кривые могут располагаться на плоскости (рис. М.1.4.2), криволинейной поверхности (рис. М.1.4.9) или в пространстве (рис. М.1.4.16); вращение может выполняться в прямом, обратном и в обоих направлениях; тело может полностью заполнять замкнутые кривые (рис. М.1.4.10) или иметь тонкую стенку (рис. М.1.4.11). Мы не будем повторять описание всех возможностей рассматриваемого метода, а остановимся только на некоторых из них, связанных с булевыми операциями.

На рис. М.2.4.1 показаны тело **solid**, образующая кривая, входящая в данные **sweptData**, и ось вращения **axis**.

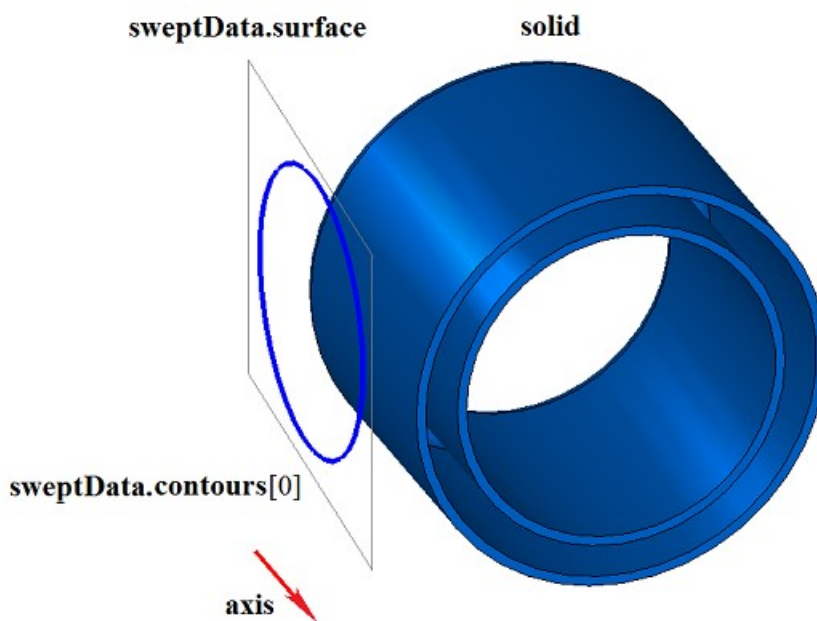


Рис. М.2.4.1.

На рис. М.2.4.2 приведен результат булевой операции объединения тела **solid** и тонкостенного тела, полученного вращением образующей кривой **sweptData** вокруг оси **axis**, показанных на рис. М.2.4.1.

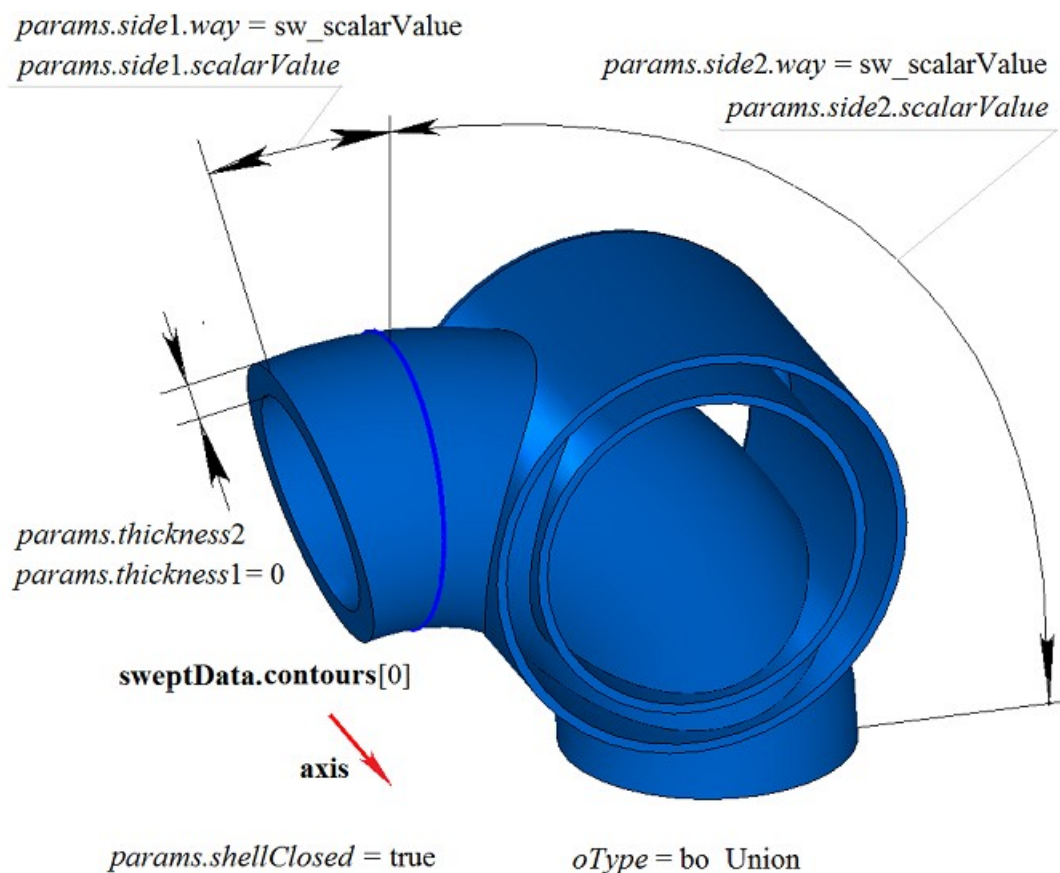


Рис. М.2.4.2.

На рис. М.2.4.3 приведен результат булевой операции объединения тела **solid** и тонкостенного тела, полученного вращением образующей кривой **sweptData** вокруг оси **axis**, показанных на рис. М.2.4.1. Вращение образующей кривой выполнено в обратном направлении с опцией «До поверхности» (*params.side2.way=sw_surface*).

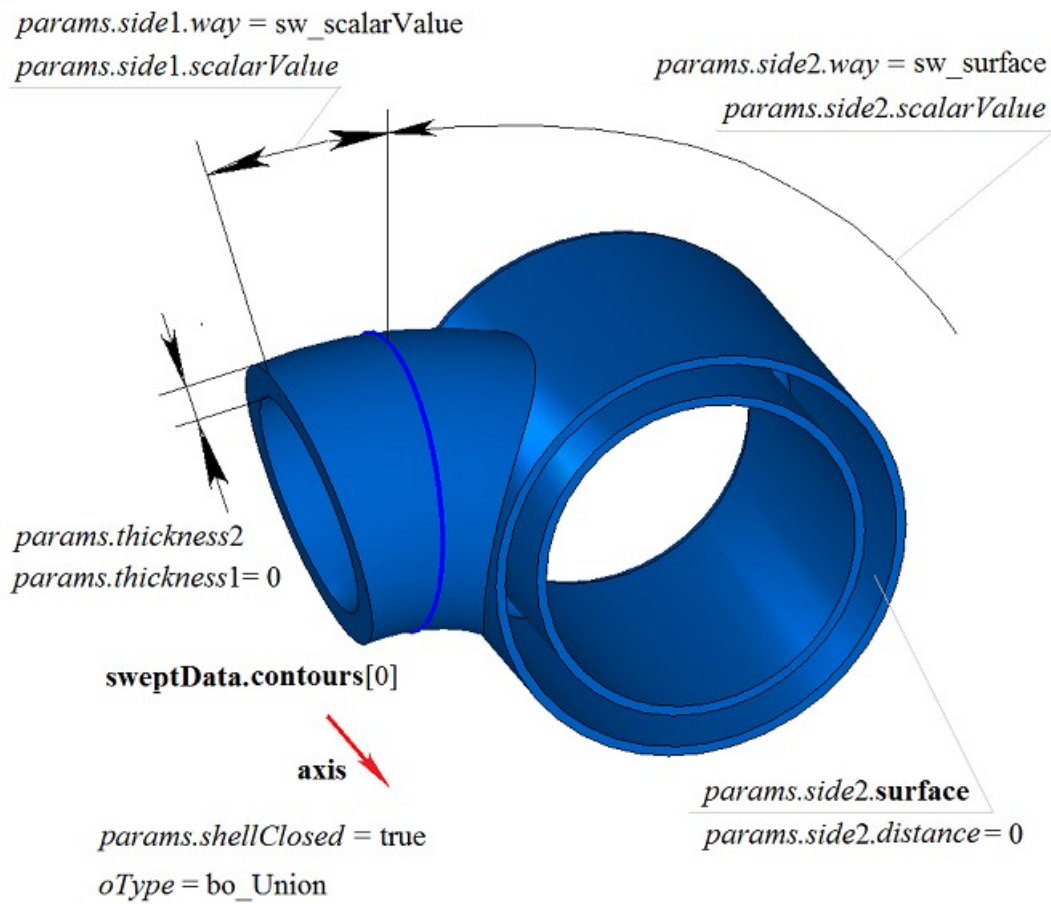


Рис. М.2.4.3.

На рис. М.2.4.4 приведен результат булевой операции вычитания из тела **solid** тела, полученного вращением образующей кривой **sweptData** вокруг оси **axis**, показанных на рис. М.2.4.1.

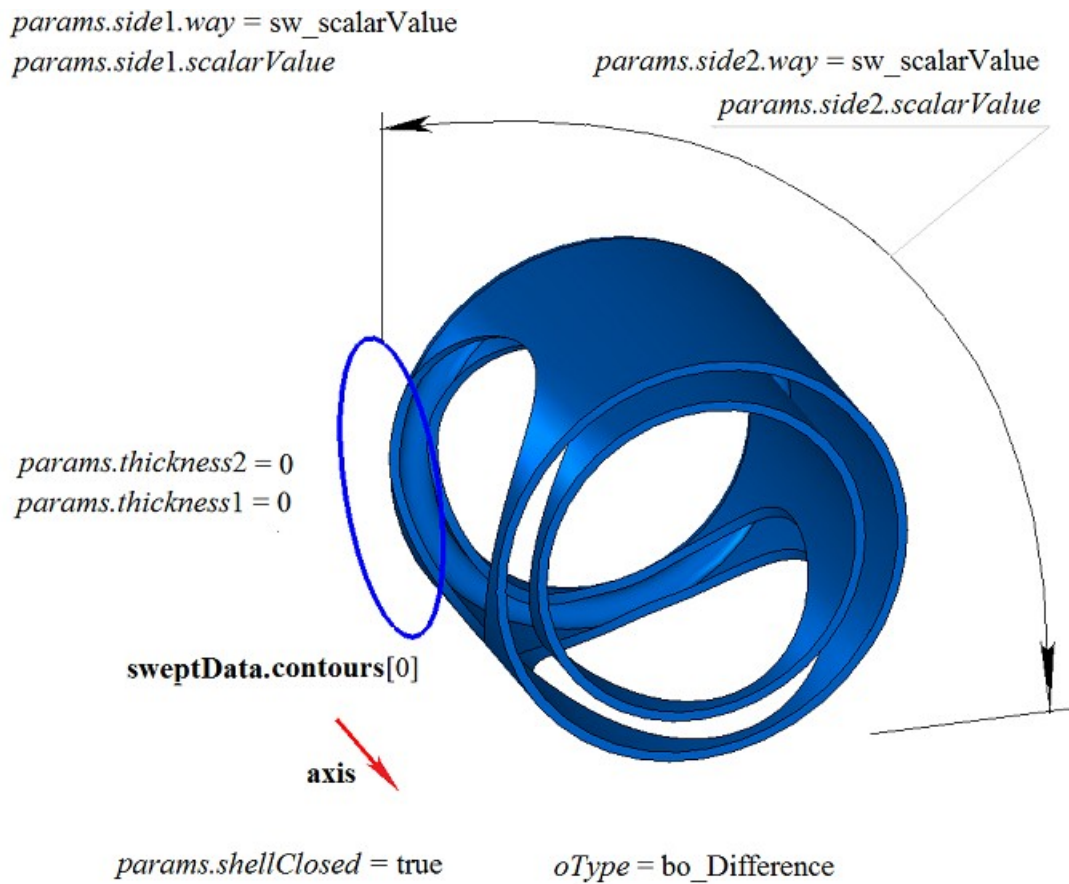


Рис. М.2.4.4.

На рис. М.2.4.5 приведен результат булевой операции вычитания из тела **solid** тела, полученного вращением образующей кривой **sweptData** вокруг оси **axis**, показанных на рис. М.2.4.1. Вращение образующей кривой выполнено в обратном направлении с опцией «До поверхности» ($params.side2.way=sw_surface$).

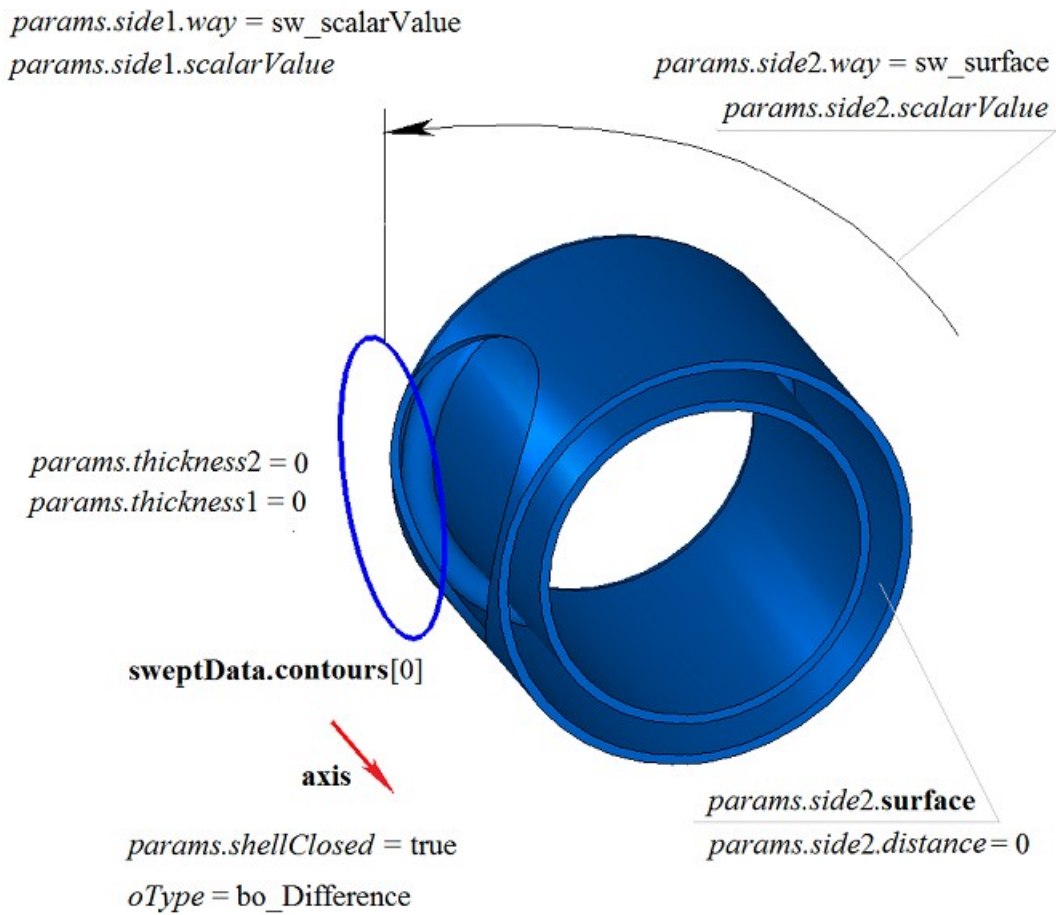


Рис. М.2.4.5.

На рис. М.2.4.6 приведен результат булевой операции пересечения тела **solid** и тела, полученного вращением образующей кривой **sweptData** вокруг оси **axis**, показанных на рис. М.2.4.1.

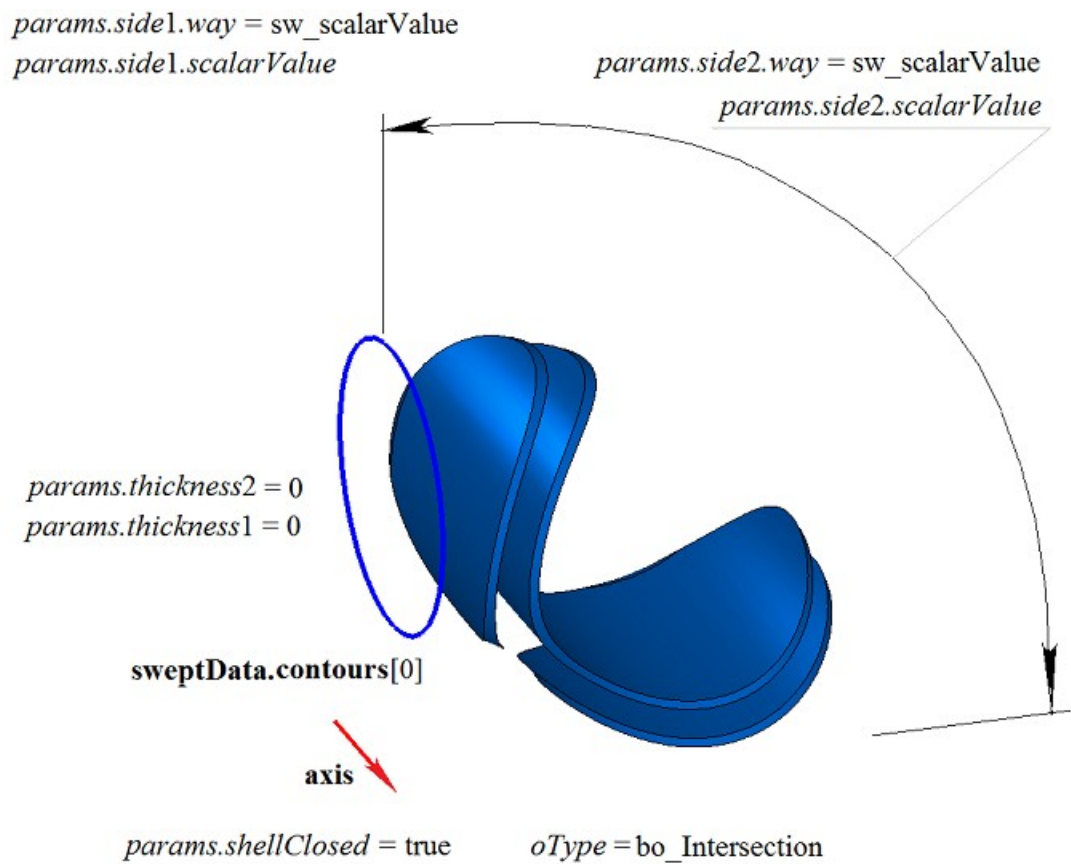


Рис. М.2.4.6.

На рис. М.2.4.7 приведен результат булевой операции пересечения тела **solid** и тела, полученного вращением образующей кривой **sweptData** вокруг оси **axis**, показанных на рис. М.2.4.1. Вращение образующей кривой выполнено в обратном направлении с опцией «До поверхности» (*params.side2.way=sw_surface*).

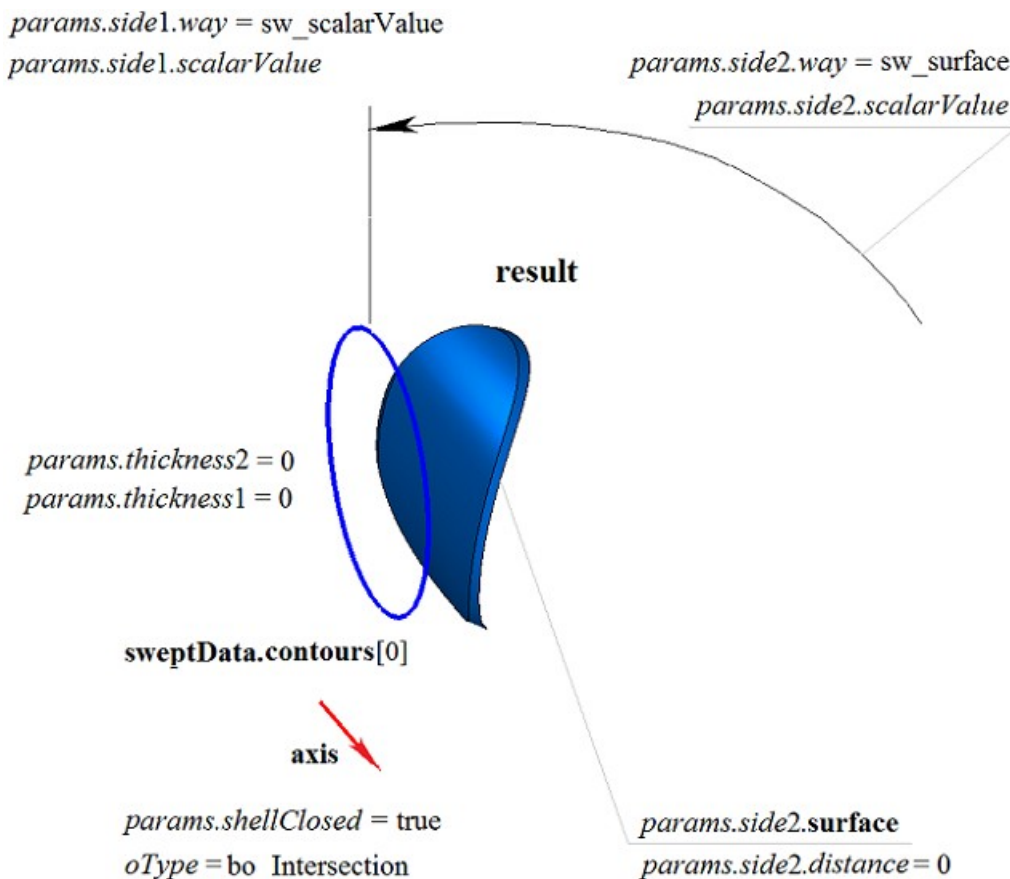


Рис. М.2.4.7.

Метод **RevolutionResult** добавляет в журнал построенного тела строитель `MbRevolutionSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbRevolutionSolid` объявлен в файле `cr_revolution_solid.h`.

Тестовое приложение `test.exe` выполняет булеву операцию с построенным телом выдавливания командами меню «Создать->Тело->Приклеиванием к телу->Вращения кривой», «Создать->Тело->Вырезанием из тела->Вращения кривой» и «Создать->Тело->Пересечением с телом->Вращения кривой».

М.2.5. Булева операция с телом заметания

Метод

`MbResultType`

EvolutionResult (`MbSolid & solid`,
`MbcCopyMode sameShell`,
`const MbSweptData & sweptData`,
`const MbCurve3D & spine`,
`EvolutionValues & params`,
`OperationType oType`,
`const MbSNameMaker & names`,
`PArray<MbSNameMaker> & cnames`,
`const MbSNameMaker & snames`,
`MbSolid *& result`)

выполняет построение тела заметания и булеву операцию заданного тела с построенным телом.

Входными параметрами метода являются:

- **solid** – заданное тело для булевой операции,
- *sameShell* – вариант копирования заданного тела,
- **sweptData** – данные об образующих кривых для построения тела выдавливания,

- **spine** – направляющая кривая,
- *params* – параметры построения,
- *oType* – тип булевой операции: bo_Union – объединение тел,
bo_Intersect – пересечение тел,
bo_Difference – вычитание тел,
- *names* – именователи граней,
- *spnames* – именователи граней тела заметания,
- *snames* – именователи направляющей.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает rt_Success, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле action_solid.h.

Метод представляет собой последовательное объединение двух методов: метода **EvolutionSolid**, выполняющего построение тела путем движения кривых **sweptData** вдоль направляющей кривой **spine** по заданным параметрам *params*, и метода **BooleanSolid**, выполняющего булеву операцию *oType* тела **solid** с построенным на предыдущем шаге телом. Метод **EvolutionSolid** описан в параграфе [М.1.5. Построение тела заметания](#), метод **BooleanSolid** описан в параграфе [М.2.1. Булева операция над телами](#). Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbCopyMode описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

Параметр OperationType *oType* определяет тип булевой операции и принимает три значения: bo_Union, bo_Intersect, bo_Difference. При *oType*=bo_Union рассматриваемый метод выполняет объединение тел **solid** и тела заметания, при *oType*=bo_Intersect рассматриваемый метод выполняет пересечение тел **solid** и тела заметания, при *oType*=bo_Difference рассматриваемый метод выполняет вычитание из тела **solid** тела заметания. Параметры *names*, *spnames* и *snames* обеспечивают именование граней построенного тела.

Метод **EvolutionResult** при построении тела путем движения кривых имеет те же возможности, что и метод **EvolutionSolid**: направляющие кривые могут располагаться на плоскости (рис. М.1.5.2), криволинейной поверхности (рис. М.1.5.8) или в пространстве (рис. М.1.5.16); тело может полностью заполнять замкнутые кривые (рис. М.1.5.9) или иметь тонкую стенку (рис. М.1.5.10). Мы не будем повторять описание всех возможностей рассматриваемого метода, а остановимся только на некоторых из них, связанных с булевыми операциями.

На рис. М.2.5.1 показаны тело **solid**, образующая кривая, входящая в данные **sweptData**, и направляющая кривая **spine**.

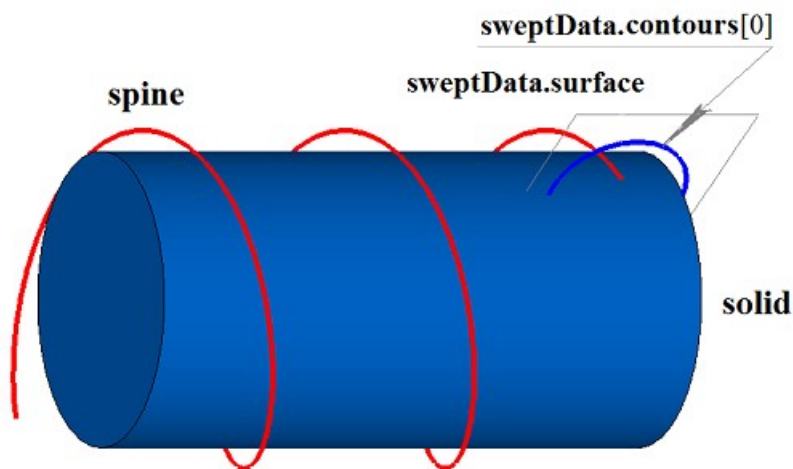


Рис. М.2.5.1.

На рис. М.2.5.2 приведен результат булевой операции объединения тела **solid** и тела, полученного движением образующей кривой **sweptData** вдоль направляющей кривой **spine**, показанных на рис. М.2.5.1.

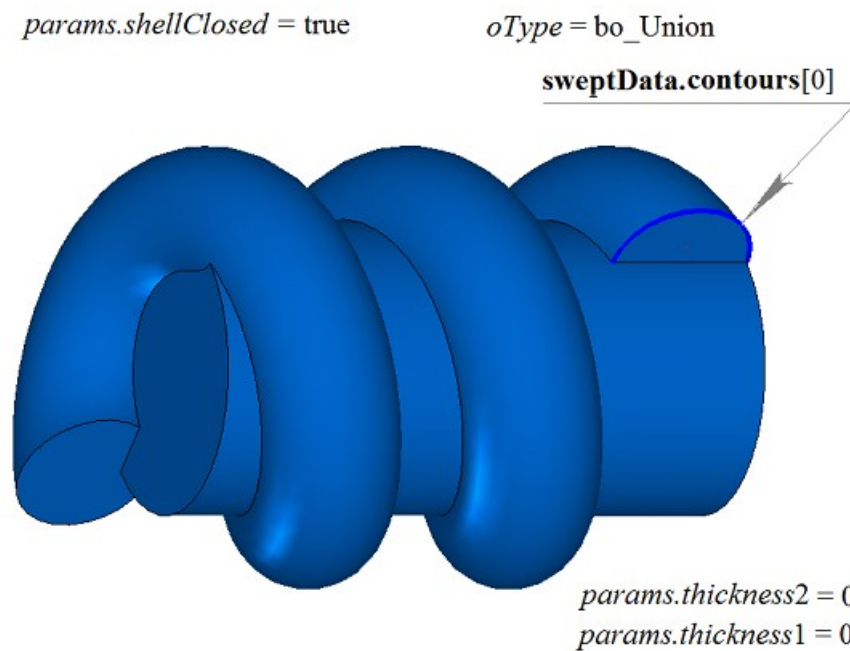


Рис. М.2.5.2.

На рис. М.2.5.3 приведен результат булевой операции вычитания из тела **solid** тела, полученного движением образующей кривой **sweptData** вдоль направляющей кривой **spine**, показанных на рис. М.2.5.1.

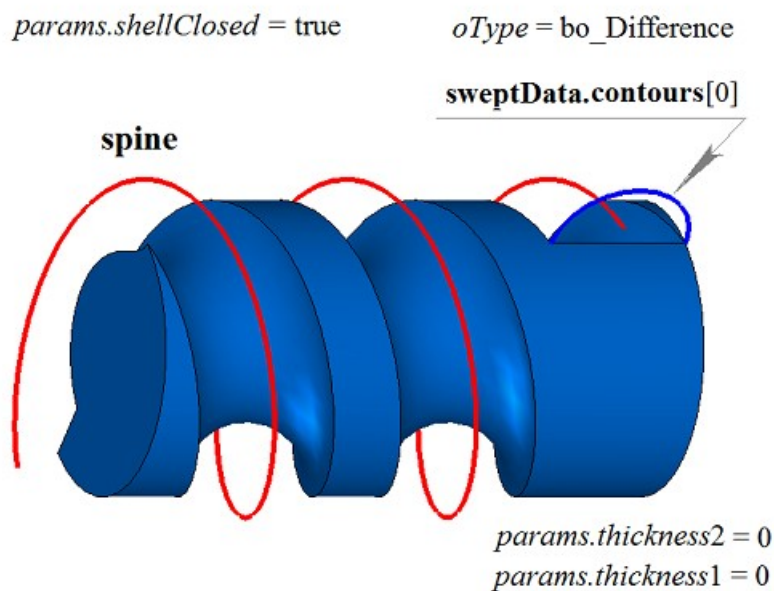


Рис. М.2.5.3.

На рис. М.2.5.4 приведен результат булевой операции объединения тела **solid** и тела, полученного движением образующей кривой **sweptData** вдоль направляющей кривой **spine**, показанных на рис. М.2.5.1.

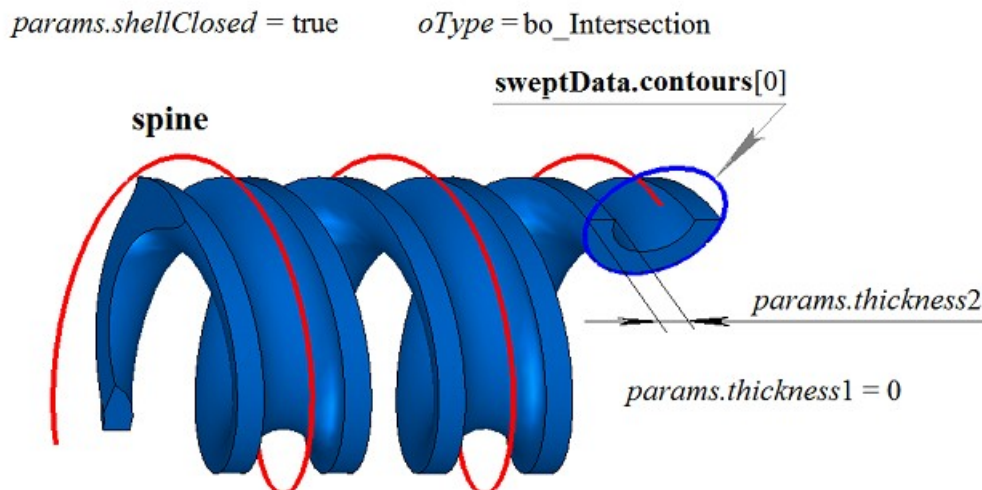


Рис. М.2.5.4.

Метод **EvolutionResult** добавляет в журнал построенного тела строитель MbEvolutionSolid, который содержит все необходимые данные для выполнения операции. Строитель MbEvolutionSolid объявлен в файле st_evolution_solid.h.

Тестовое приложение test.exe выполняет булеву операцию с построенным телом заметания командами меню «Создать->Тело->Приклеиванием к телу->Движением кривой», «Создать->Тело->Вырезанием из тела->Движением кривой» и «Создать->Тело->Пересечением с телом->Движением кривой».

М.2.6. Булева операция с телом, построенным по плоским сечениям

Метод
MbResultType
LoftedResult (MbSolid & solid,
MbeCopyMode sameShell,
SArray<MbPlacement3D> & places,
RPAArray<MbContour> & contours,
const MbCurve3D * spine,
LoftedValues & params,
OperationType oType,
Sarray<MbCartPoint3D> * points,
const MbSNameMaker & names,
PArray<MbSNameMaker> & snames,
MbSolid *& result)

выполняет построение тела по плоским сечениям и булеву операцию заданного тела с построенным телом.

Входными параметрами метода являются:

- **solid** – заданное тело для булевой операции,
- *sameShell* – вариант копирования заданного тела,
- **places** – множество локальных систем координат образующих контуров,
- **contours** – множество образующих контуров,
- **spine** – направляющая кривая (может отсутствовать),
- *params* – параметры построения,
- *oType* – тип булевой операции: bo_Union – объединение тел,
bo_Intersect – пересечение тел,
bo_Difference – вычитание тел,

- **points** – множество контрольных точек (может отсутствовать),
- **names** – именователи граней,
- **snames** – именователи образующих контуров.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Метод представляет собой последовательное объединение двух методов: метода **LoftedSolid**, выполняющего построение тела по плоским сечениям **contours** на плоскостях **places** по заданным параметрам *params*, и метода **BooleanSolid**, выполняющего булеву операцию *oType* тела **solid** с построенным на предыдущем шаге телом. Метод **LoftedSolid** описан в параграфе [М.1.6. Построение тела по плоским сечениям](#), метод **BooleanSolid** описан в параграфе [М.2.1. Булева операция над телами](#). Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление `MbCopyMode` описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

Параметр `OperationType` *oType* определяет тип булевой операции и принимает три значения: `bo_Union`, `bo_Intersect`, `bo_Difference`. При *oType=bo_Union* рассматриваемый метод выполняет объединение тел **solid** и тела заметания, при *oType=bo_Intersect* рассматриваемый метод выполняет пересечение тел **solid** и тела заметания, при *oType=bo_Difference* рассматриваемый метод выполняет вычитание из тела **solid** тела заметания. Параметры *names*, *snames* и *snames* обеспечивают именование граней построенного тела.

Метод **LoftedResult** при построении тела по плоским сечениям имеет те же возможности, что и метод **LoftedSolid**: тело может строиться разомкнутым (рис. М.1.6.3) и замкнутым (рис. М.1.6.4); тело в районе торцов может иметь разную форму (рис. М.1.6.5 и рис. М.1.6.6); тело может полностью заполнять замкнутые кривые (рис. М.1.6.3) или иметь тонкую стенку (рис. М.1.6.6); форма тела по сечениям может управляться направляющей (рис. М.1.6.15 и рис. М.1.6.16). Мы не будем повторять описание всех возможностей рассматриваемого метода, а остановимся только на некоторых из них, связанных с булевыми операциями.

На рис. М.2.6.1 показаны тело **solid** и плоские замкнутые кривые.

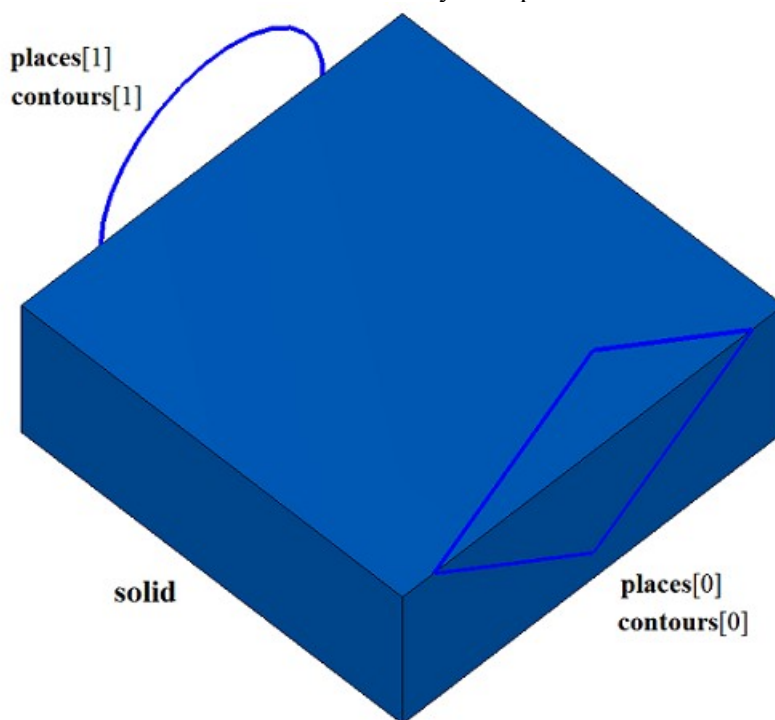


Рис. М.2.6.1.

На рис. М.2.6.2 приведен результат булевой операции объединения тела **solid** и тела, построенного по плоским сечениям **contours**, показанных на рис. М.2.6.1.

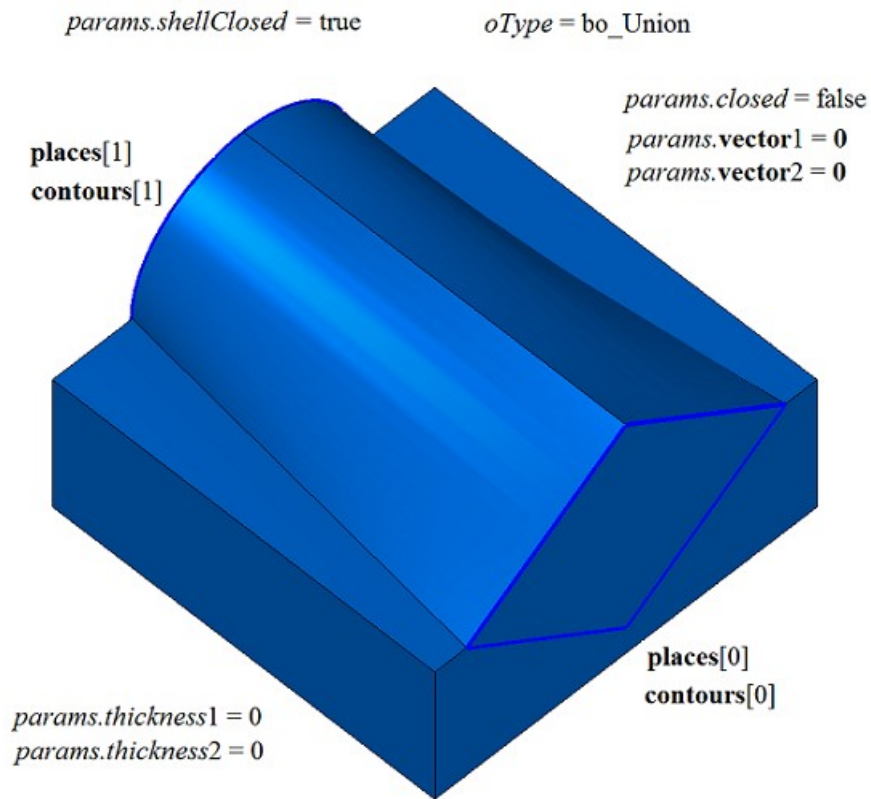


Рис. М.2.6.2.

На рис. М.2.6.3 приведен результат булевой операции вычитания из тела **solid** тела, построенного по плоским сечениям **contours**, показанных на рис. М.2.6.1.

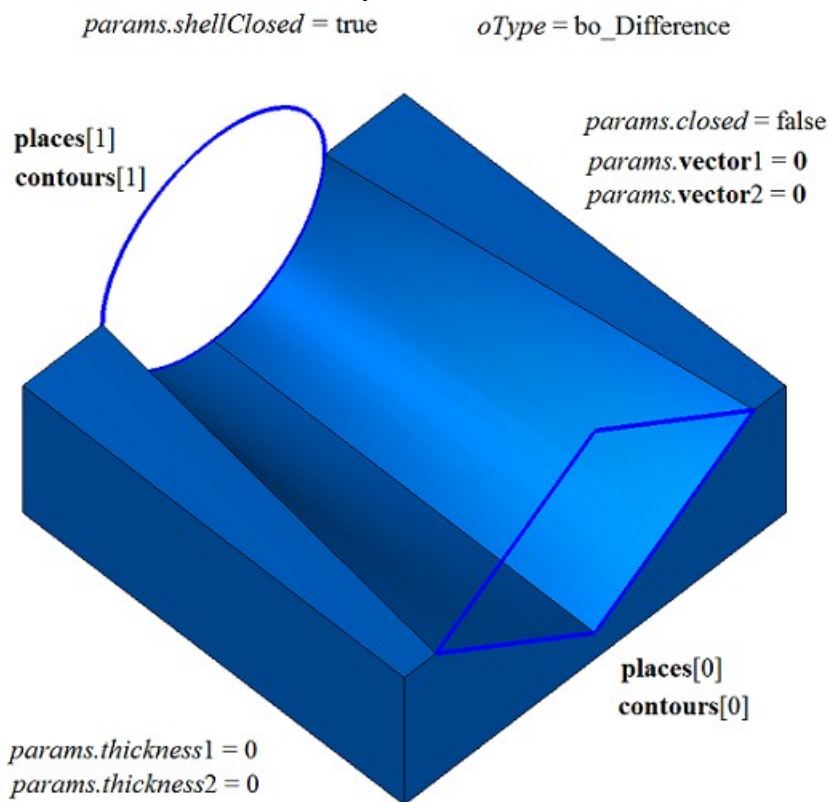


Рис. М.2.6.3.

На рис. М.2.6.4 приведен результат булевой операции объединения тела **solid** и тела, построенного по плоским сечениям **contours**, показанных на рис. М.2.6.1.

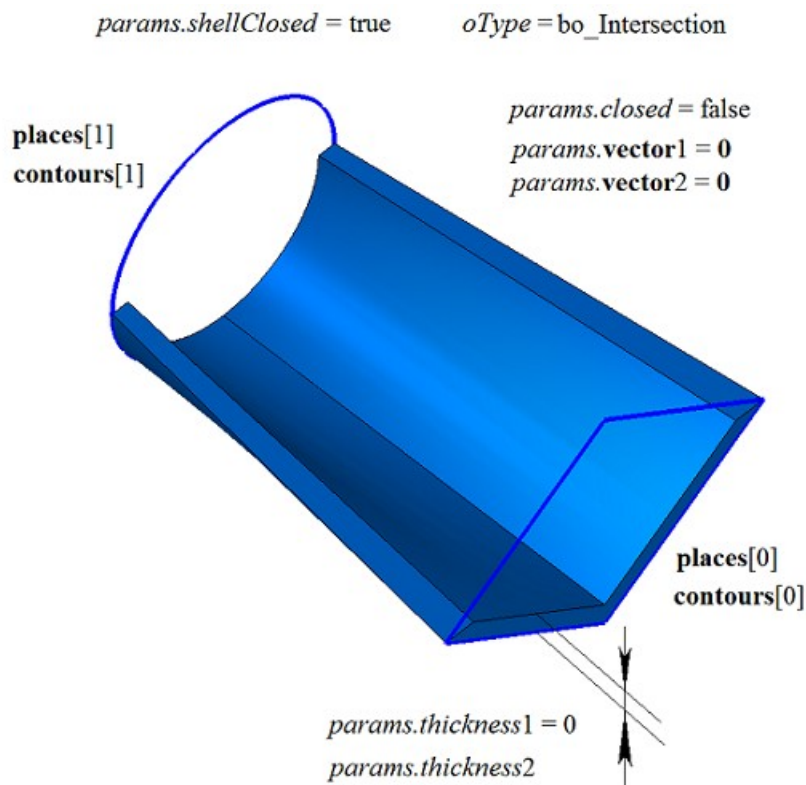


Рис. М.2.6.4.

Метод **LoftedResult** добавляет в журнал построенного тела строитель **MbLoftedSolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbLoftedSolid** объявлен в файле `sr_lofted_solid.h`.

Тестовое приложение `test.exe` выполняет булеву операцию с построенным телом, построенным по плоским сечениям, командами меню «Создать->Тело->Приклеиванием к телу->По сечениям», «Создать->Тело->Приклеиванием к телу->По сечениям и направляющей кривой», «Создать->Тело->Вырезанием из тела->По сечениям», «Создать->Тело->Вырезанием из тела->По сечениям и направляющей кривой», «Создать->Тело->Пересечением с телом->По сечениям» и «Создать->Тело->Пересечением с телом->По сечениям и направляющей кривой».

М.2.7. Резка тела поверхностью

Метод
MbResultType
SolidCutting (**MbSolid** & **solid**,
 MbeCopyMode *sameShell*,
 const **MbSurface** & **surface**,
 int *part*,
 const **MbSNameMaker** & *names*,
 bool *closed*,
MbSolid *& **result**)

отрезает часть тела пересекающей его поверхностью.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **surface** – секущая поверхность,
- *part* – сохраняемая часть тела:
 - part* = +1 – сохраняется часть тела, расположенная сверху поверхности,
 - part* = 0 – сохраняются все разрезанные части тела,
 - part* = -1 – сохраняется часть тела, расположенная снизу поверхности,

- *names* – именователь грани среза,
- *closed* – флаг замкнутости тела в операции:
true – тело считается замкнутым,
false – тело считается не замкнутым.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_solid.h*.

Метод строит незамкнутую оболочку с одной гранью на базе режущей поверхности **surface** и выполняет булеву операцию пересечения исходного тела **solid** с незамкнутой оболочкой. Для выполнения операции режущая поверхность должна полностью пересекать исходное тело. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление *MbCopyMode* описано в параграфе [0.7.9. Копирование множества граней MbFaceShell](#).

Параметр *part* определяет оставляемую часть исходного тела **solid**: при *part*=+1 сохраняется часть тела, расположенная над поверхностью (с той стороны, в которую направлена нормаль поверхности), при *part*=-1 сохраняется часть тела, расположенная под поверхностью, при *part*=0 сохраняются обе части тела.

Параметр *names* обеспечивает именование граней построенного тела. Параметр *closed* говорит о том, каким считать исходное тело **solid**: замкнутым или незамкнутым.

При *closed*=*true* операция выполняется над множеством точек, расположенным внутри и на поверхности тела. При *closed*=*false* операция выполняется над множеством точек, расположенным на поверхности тела.

На рис. М.2.7.1 показаны исходное тело **solid** и режущая поверхность **surface**.

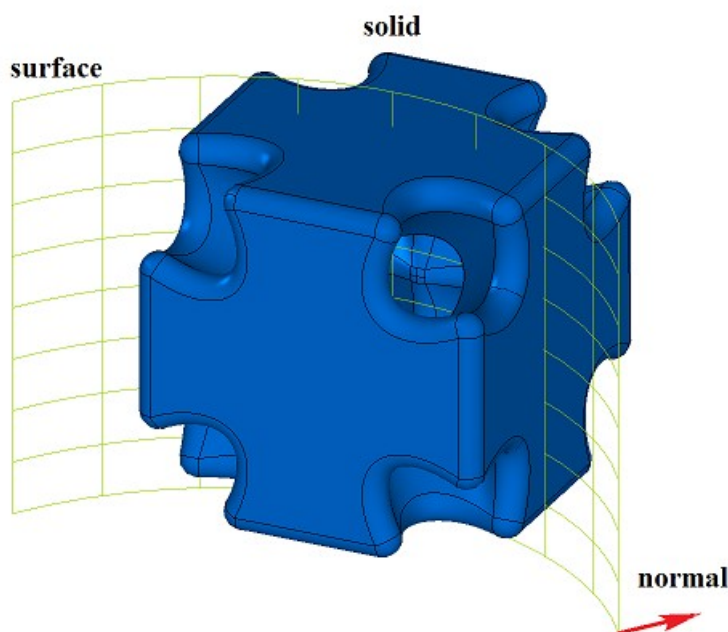


Рис. М.2.7.1.

На рис. М.2.7.2 приведено построенное тело **result** при *part*=+1 и *closed*=*true*. На рис. М.2.7.3 приведено построенное тело **result** при *part*=-1 и *closed*=*true*.

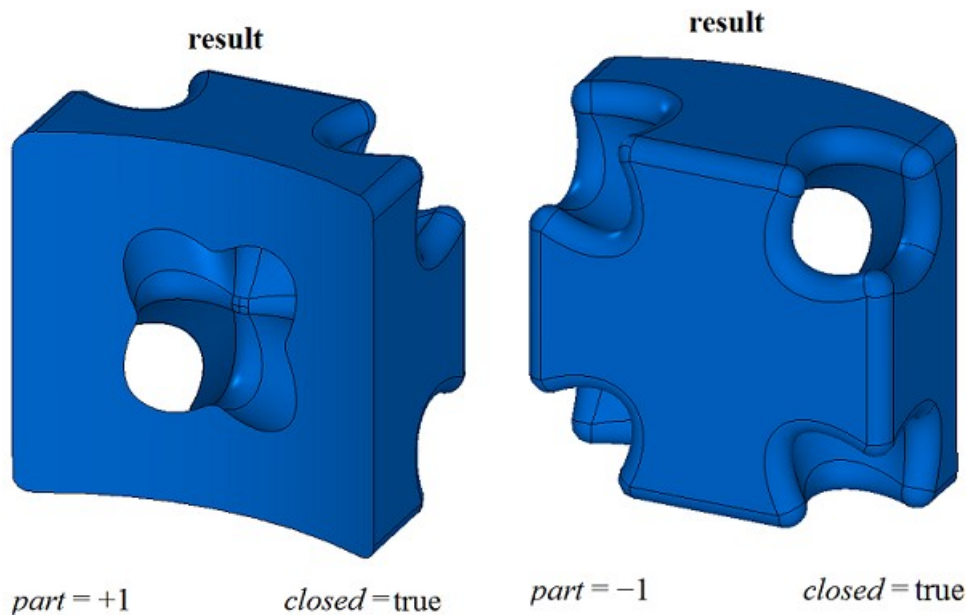


Рис. М.2.7.2.

Рис. М.2.7.3.

На рис. М.2.7.4 приведено построенное тело **result** при $part=+1$ и $closed=false$.

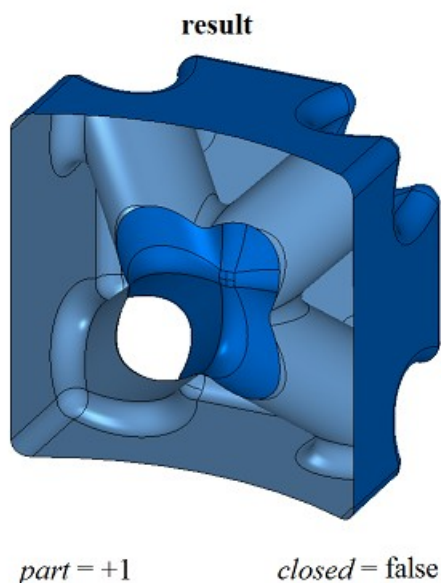


Рис. М.2.7.4.

При $part=0$ рассматриваемый метод построит тело **result**, в которое войдут все части исходного тела **solid**, полученные после резки. Для разделения тела **result** на части следует воспользоваться методом **DetachParts** или **CreateParts**, описанным в параграфе [М.2.19. Разделить тело на несвязанные части](#). Вместо вызова рассматриваемого метода при $part=0$ можно использовать метод, приведенный ниже.

Метод
 MbResultType
SolidCutting ([MbSolid](#) & **solid**,
 MbeCopyMode *sameShell*,
 const [MbSurface](#) & **surface**,
 const MbSNameMaker & names,
 bool *closed*,

RPAArray<MbSolid> & result)

режет тело пересекающей его поверхностью и из всех частей строит отдельные тела. Параметры метода, кроме *part*, совпадают с описанными выше. На рис. М.2.7.5 приведено построенное данным методом тело **result** при *closed=true*

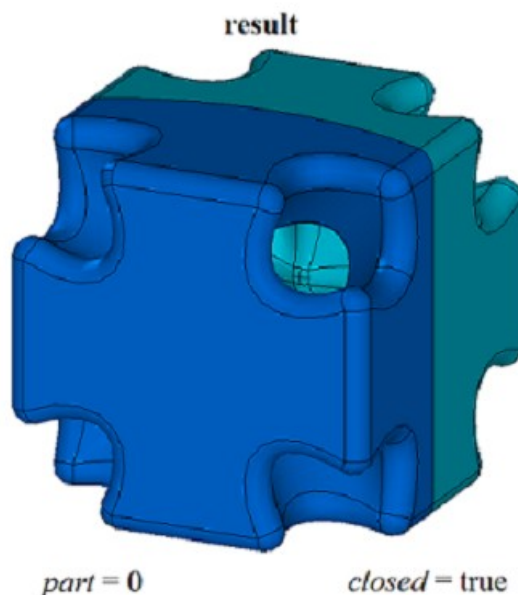


Рис. М.2.7.5.

Методы **SolidCutting** добавляют в журнал построенного тела строитель MbCuttingSolid, который содержит все необходимые данные для выполнения операции. Строитель MbCuttingSolid объявлен в файле *cg_cutting_solid.h*.

Тестовое приложение *test.exe* выполняет резку тела поверхностью командами меню «Создать->Тело->На базе тела->Разрезанное поверхностью» и «Создать->Оболочку->На базе оболочки->Разрезанную поверхностью».

М.2.8. Резка тела плоским контуром

Метод

MbResultType

SolidCutting (MbSolid & solid,
MbeCopyMode *sameShell*,
const MbPlacement3D & place,
const MbContour & contour,
const MbVector3D & direction,
int *part*,
const MbSNameMaker & names,
bool *closed*,
MbSolid *& result)

отрезает часть тела пересекающей его поверхностью тела, полученного выдавливанием плоского контура.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **place** – локальная система координат образующего контура,
- **contour** – образующий контур,
- **direction** – направление выдавливания образующего контура,
- *part* – сохраняемая часть тела:

- $part = +1$ – сохраняется часть тела, расположенная сверху поверхности,
- $part = 0$ – сохраняются все разрезанные части тела,
- $part = -1$ – сохраняется часть тела, расположенная снизу поверхности,
- *names* – именованье грани среза,
- *closed* – флаг замкнутости тела в операции:
 - true* – тело считается замкнутым,
 - false* – тело считается не замкнутым.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Метод строит незамкнутую оболочку путем выдавливания в направлении **direction** двумерного контура **contour**, расположенного в плоскости XY локальной системы координат **place**, и выполняет булеву операцию пересечения исходного тела **solid** с незамкнутой оболочкой. При равенстве нулю вектора **direction** выдавливание контура выполняется вдоль вектора **place.axisZ**. Для выполнения операции секущий контур должен полностью пересекать проекцию исходного тела на плоскость XY локальной системы координат **place** в направлении вектора выдавливания. Длина выдавливания контура рассчитывается так, чтобы незамкнутая оболочка полностью пересекала бы исходное тело. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление `MbCopyMode` описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

Параметр *part* определяет оставляемую часть исходного тела **solid**: при $part=+1$ сохраняется часть тела, расположенная справа от контура, при $part=-1$ сохраняется часть тела, расположенная слева от контура, если смотреть вдоль контура навстречу оси **place.axisZ**. Параметр *names* обеспечивает именованье граней построенного тела. Параметр *closed* говорит о том, каким считать исходное тело **solid**: замкнутым или незамкнутым.

При *closed=true* операция выполняется над множеством точек, расположенным внутри и на поверхности тела. При *closed=false* операция выполняется над множеством точек, расположенным на поверхности тела.

На рис. М.2.8.1 показаны исходное тело **solid**, режущий контур **contour** и плоскость XY локальной системы координат **place**.

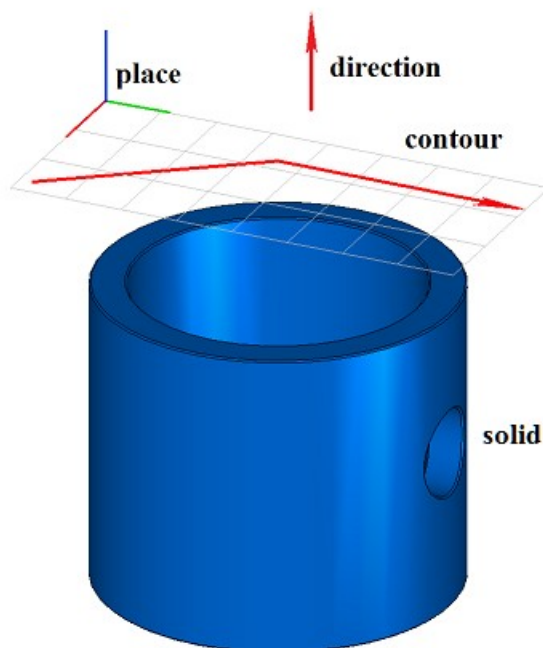


Рис. М.2.8.1.

На рис. М.2.8.2 приведено построенное тело **result** при $part=+1$ и $closed=true$. На рис. М.2.8.3 приведено построенное тело **result** при $part=-1$ и $closed=true$.

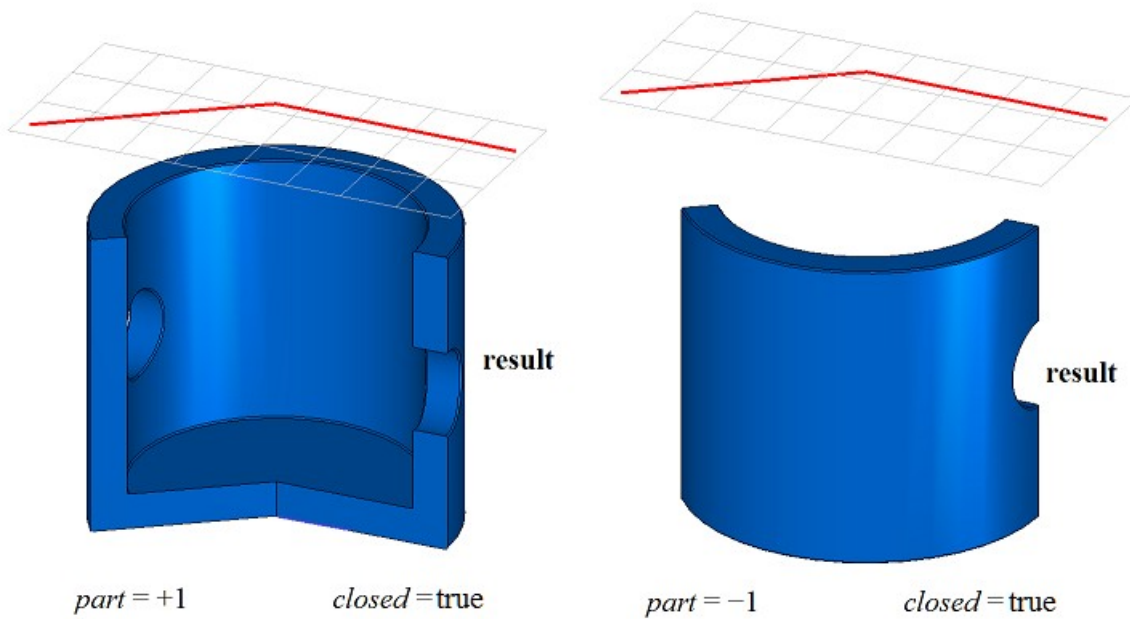


Рис. М.2.8.2.

Рис. М.2.8.3.

На рис. М.2.8.4 приведено построенное тело **result** при $part=+1$ и $closed=false$.

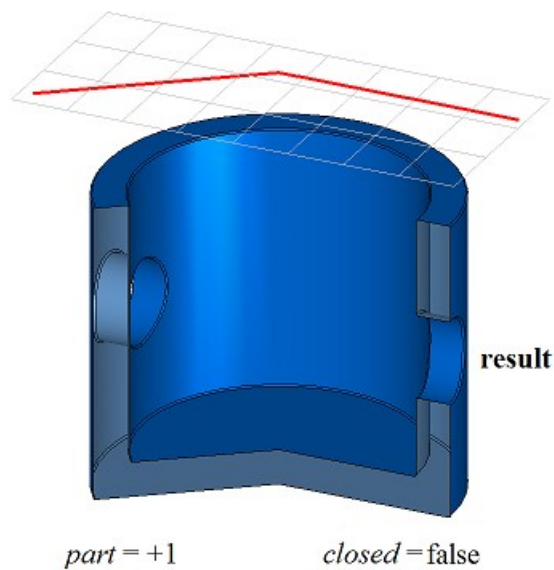


Рис. М.2.8.4.

При $part=0$ рассматриваемый метод построит тело **result**, в которое войдут все части исходного тела **solid**, полученные после резки. Для разделения тела **result** на части следует воспользоваться методом **DetachParts** или **CreateParts**, описанным в параграфе [М.2.19. Разделить тело на несвязанные части](#). Вместо вызова рассматриваемого метода при $part=0$ можно использовать метод, приведенный ниже.

Метод
MbResultType
SolidCutting ([MbSolid](#) & **solid**,
 MbeCopyMode *sameShell*,
 const [MbPlacement3D](#) & **place**,


```

const MbContour & contour,
const MbVector3D & direction,
const MbSNameMaker & names,
bool closed,
RPAArray<MbSolid> & result )

```

режет тело поверхностью, полученной выдавливанием плоского контура, и из всех частей строит отдельные тела. Параметры метода, кроме *part*, совпадают с описанными выше. На рис. М.2.8.5 приведено построенное данным методом тело **result** при *closed=true*

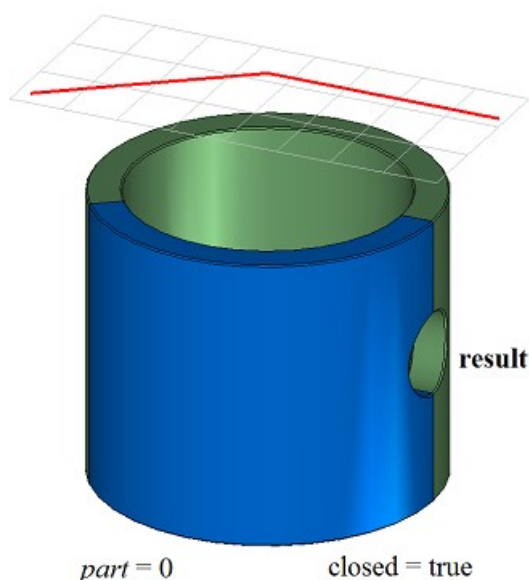


Рис. М.2.8.5.

Методы **SolidCutting** добавляют в журнал построенного тела строитель MbCuttingSolid, который содержит все необходимые данные для выполнения операции. Строитель MbCuttingSolid объявлен в файле *cg_cutting_solid.h*.

Тестовое приложение *test.exe* выполняет резку тела поверхностью командами меню «Создать->Тело->На базе тела->Разрезанное кривой» и «Создать->Оболочку->На базе оболочки->Разрезанную кривой».

М.2.9. Построение симметричного тела

```

Метод
MbResultType
SymmetrySolid ( MbSolid & solid,
                  MbeCopyMode sameShell,
                  const MbPlacement3D & place,
                  const MbSNameMaker & names,
                  MbSolid *& result )

```

выполняет построение симметричного тела с заданной плоскостью симметрии.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **place** – локальная система координат, плоскость XY которой является плоскостью симметрии,
- *names* – именованная граница среза,

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Метод создаёт симметричное тело с заданной плоскостью симметрии следующим образом. Исходное тело **solid** режется плоскостью XY локальной системы координат **place**, берётся часть исходного тела, расположенная снизу режущей плоскости, строится зеркальная копия выбранной части исходного тела и объединяется с выбранной частью исходного тела. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление `MbCopyMode` описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

На рис. М.2.9.1 показаны исходное тело **solid** и плоскость симметрии **place**.

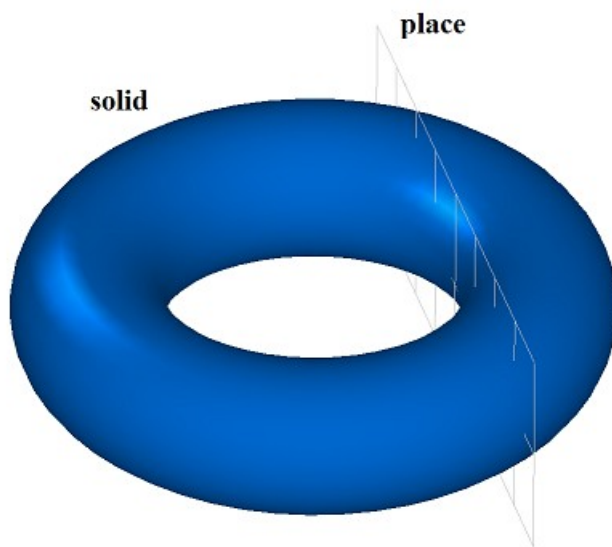


Рис. М.2.9.1.

На рис. М.2.9.2 приведено построенное тело **result**.

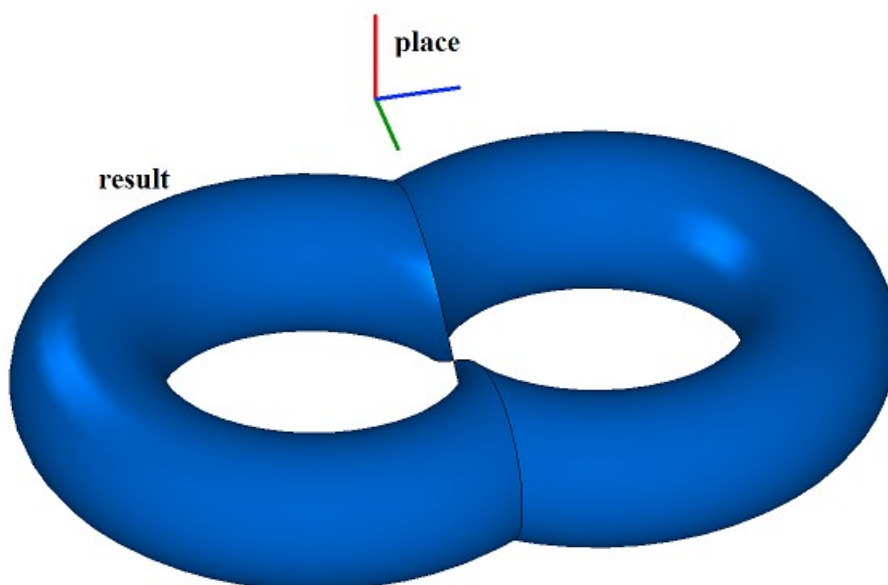


Рис. М.2.9.2.

На рис. М.2.9.3 приведено построенное тело **result**, для плоскости симметрии с противоположной нормалью.

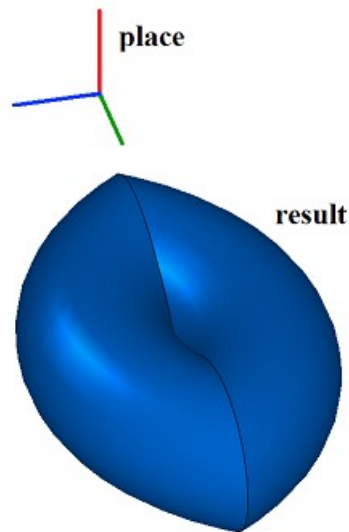


Рис. М.2.9.3.

Если исходное тело **solid** не соприкасается с плоскостью XY локальной системы координат **place**, то построение не выполняется. Построить симметричное тело в последнем случае можно методом **MirrorSolid**.

Метод **SymmetrySolid** добавляет в журнал построенного тела строитель **MbSymmetrySolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbSymmetrySolid** объявлен в файле `cg_symmetry_solid.h`.

Тестовое приложение `test.exe` выполняет построение симметричного тела командой меню «Создать->Тело->На базе тела->Симметричное».

М.2.10. Скругление рёбер тела

Метод

MbResultType

```
FilletSolid ( MbSolid & solid,  
             MbeCopyMode sameShell,  
             RPAArray<MbCurveEdge> & edges,  
             RPAArray<MbFace> & bounds,  
             const SmoothValues & params,  
             const MbSNameMaker & names,  
             MbSolid *& result )
```

выполняет скругление указанных рёбер копии исходного тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **edges** – множество скругляемых рёбер.
- **bounds** – множество граней для обрезки краев граней скругления (может быть пустым),
- *params* – параметры построения,
- *names* – именователь построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_solid.h`.

Метод выполняет замену указанных рёбер исходного тела гранями скругления, обеспечивающими гладкое сопряжение смежных граней указанных рёбер. При скруглении ребер сопрягающие грани в поперечном сечении могут иметь форму дуги окружности, эллипса, гиперболы, параболы.

Параметр **solid** содержит исходное тело, ребра которого подлежат обработке. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** к построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление *MbeCopyMode* описано в параграфе [0.7.9. Копирование множества граней MbFaceShell](#).

Параметр **edges** содержит обрабатываемые ребра тела **solid**. Параметр **bounds** содержит грани тела **solid**, которые следует использовать для обрезки скругления в неоднозначной ситуации. Параметр *names* обеспечивает именование граней сопряжения.

Параметры построения скруглений *params* содержит информацию о форме и способе сопряжения смежных граней обрабатываемых ребер, рис. М.2.10.1. Класс *SmoothValues* описан в файле *shell_parameter.h*.

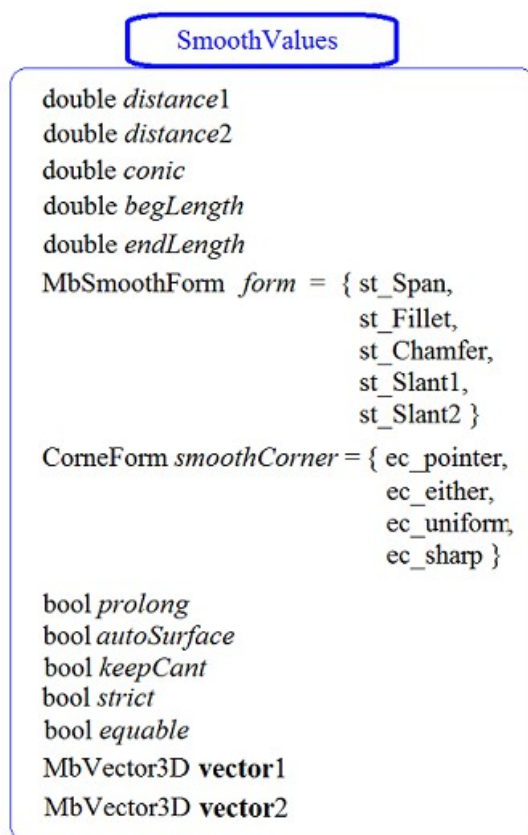


Рис. М.2.10.1.

Входной параметр *params* содержит следующие данные:

- *distance1* – первый радиус скругления,
- *distance2* – второй радиус скругления,
- *conic* – коэффициент формы поверхности сопряжения,
- *begLength* – расстояние от начальной вершины до точки остановки сопряжения (отрицательное значение означает отсутствие остановки),
- *endLength* – расстояние от конечной вершины до точки остановки сопряжения (отрицательное значение означает отсутствие остановки),
- *form* – тип сопряжения из перечисления *MbeSmoothForm*,
- *smoothCorner* – способ скругления чешуйчатых углов,
- *prolong* – флаг продолжения скругления по касательным рёбрам,
- *autoSurface* – флаг автоопределения сохранения кромки,
- *keepCant* – флаг сохранения кромки,
- *strict* – флаг строгости построения: при false скруглить хотя бы то, что возможно,

- *equable* – флаг вставки тороидальной поверхности в углах сочленения поверхности сопряжения,
- *vector1* – вектор нормали плоскости остановки сопряжения в начале,
- *vector2* – вектор нормали плоскости остановки сопряжения в конце.

Типом скругления управляет параметр *form*. Для скругления ребер используются значения параметра *form* равные *st_Fillet* и *st_Span*, другие значения *form* рассматриваемый метод не использует. При значении *form=st_Fillet* рассматриваемый метод строит поверхность скругления с заданными радиусами, которые определяют параметры *distance1* и *distance2*. На рис. М.2.10.2 приведено скругление с заданными радиусами ребра, соединяющего две цилиндрические поверхности.

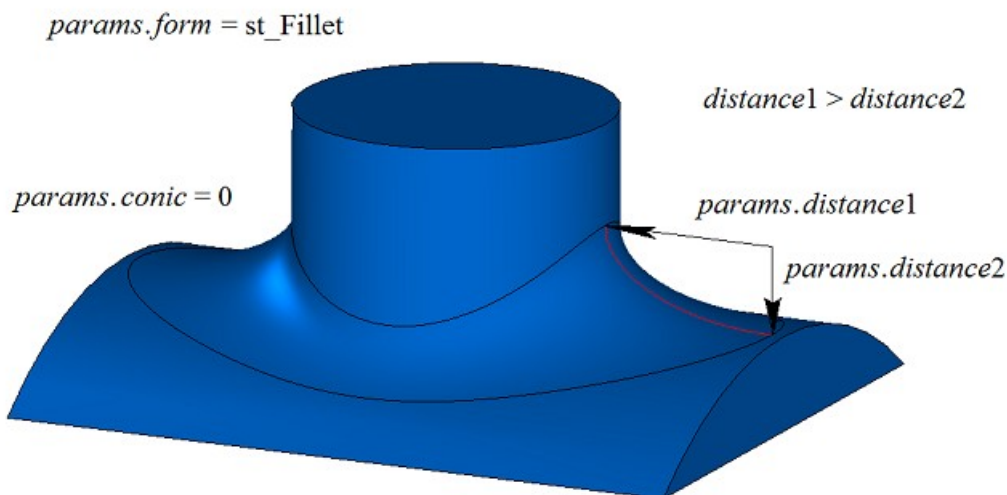


Рис. М.2.10.2.

При *distance1=distance2* и *conic=0* поверхность скругления строится путем движения сферы, касающейся двух смежных для скругляемого ребра граней. Опорные края грани сопряжения проходят по точкам касания сферы и соответствующей смежной грани. Поперечное сечение грани сопряжения представляет собой дугу окружности. На рис. М.2.10.3 приведено скругление с заданными одинаковыми радиусами ребра, соединяющего две цилиндрические поверхности.

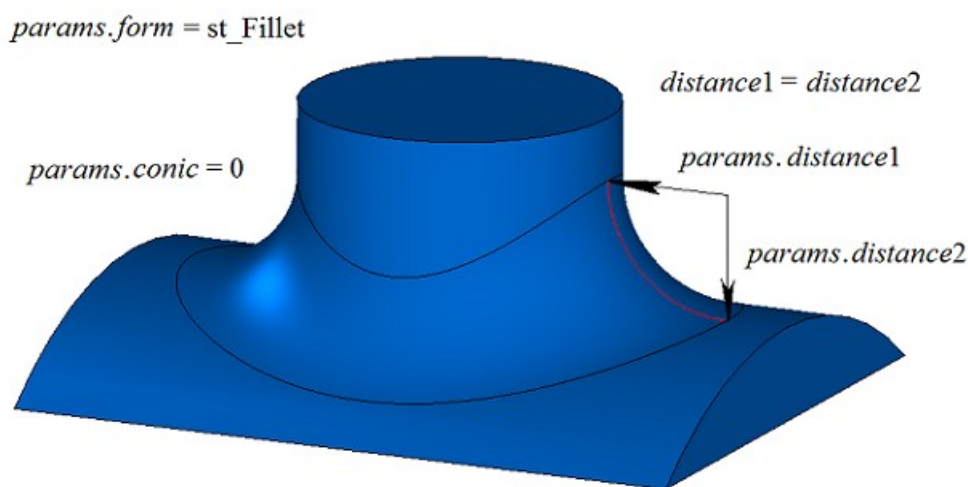


Рис. М.2.10.3.

Коэффициент *conic* управляет формой поверхности скругления. При *conic=0* (макрос *_ARC_*) сечение поверхности сопряжения имеет форму дуги окружности или эллипса с заданными радиусами. Кроме нулевого значения коэффициент формы может принимать значения от 0.05 до 0.95.

При $conic=0.5$ поперечное сечение грани скругления представляет собой дугу параболы. При $conic>0.5$ поперечное сечение грани скругления представляет собой дугу гиперболы. При $conic<0.5$ поперечное сечение грани скругления представляет собой дугу эллипса. На рис. М.2.10.4, М.2.10.5 приведены скругления одинаковыми радиусами ребра, соединяющего две цилиндрические поверхности, с разными коэффициентами формы.

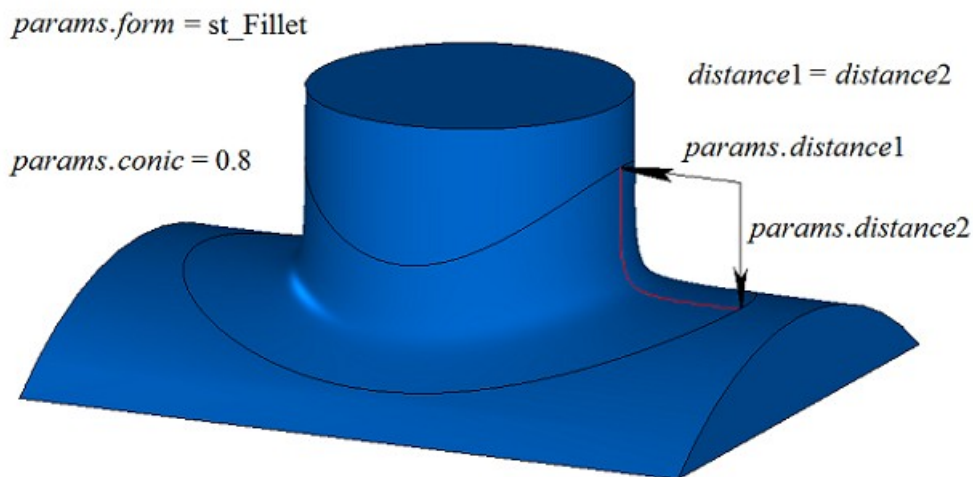


Рис. М.2.10.4.

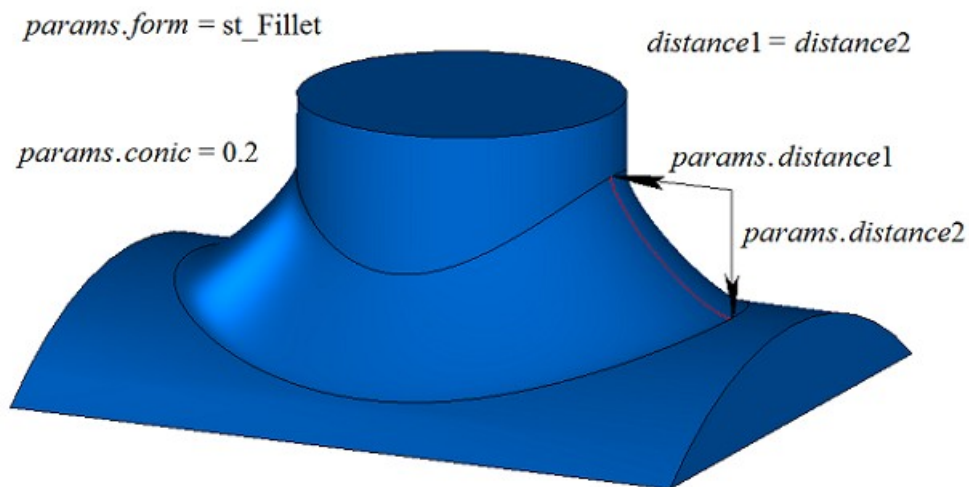


Рис. М.2.10.5.

При значении $form=st_Span$ рассматриваемый метод строит поверхность скругления с заданной хордой. Параметры $distance1$ и $distance2$ равны и определяют расстояние между опорными краями грани сопряжения. Поперечное сечение грани скругления представляет собой дугу окружности. В общем случае в каждом поперечном сечении грани скругления радиус дуги различный, а параметры $distance1$ и $distance2$ равны хорде дуги окружности. На рис. М.2.10.6 приведено скругление с заданной хордой ребра, соединяющего две цилиндрические поверхности.

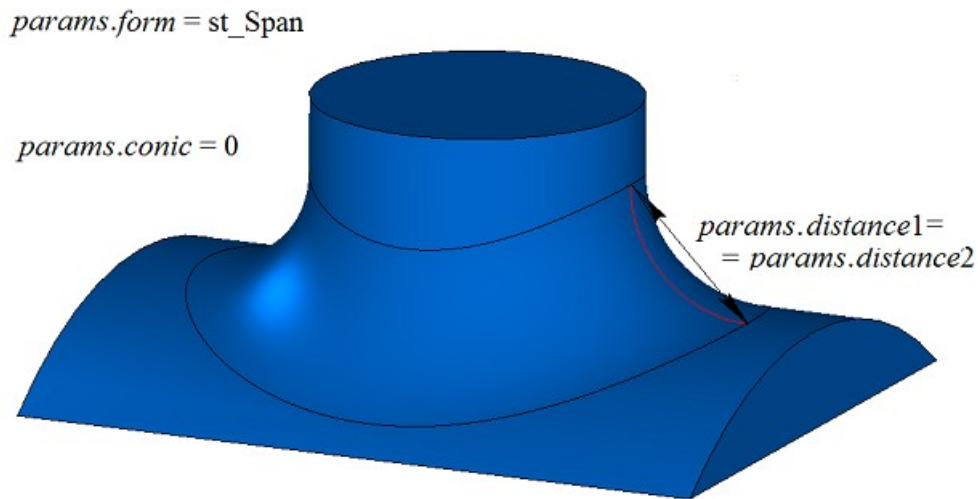


Рис. М.2.10.6.

На рис. М.2.10.6 приведено скругление с заданной хордой ребра, соединяющего две цилиндрические поверхности, с ненулевым коэффициентом формы.

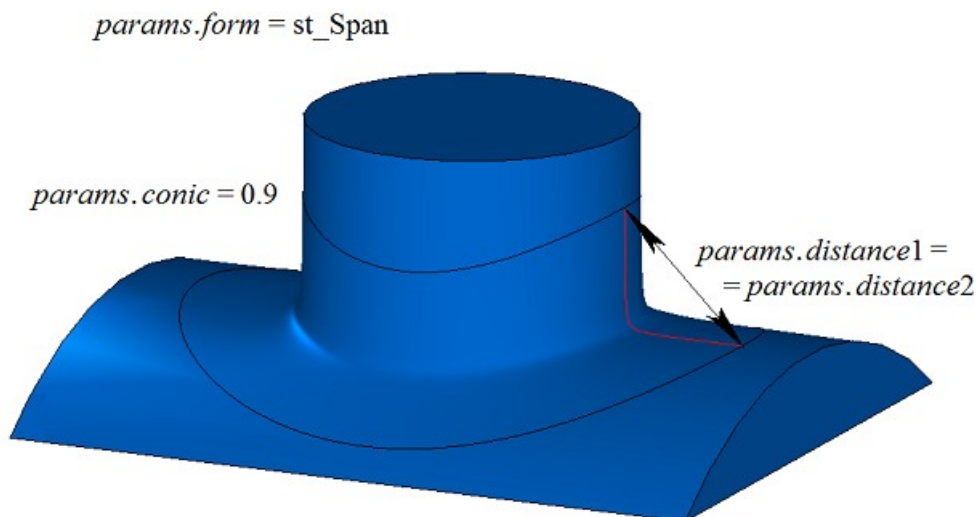


Рис. М.2.10.7.

На рис. М.2.10.8 приведен пример остановки скругления на расстоянии *begLength* от начальной вершины и на расстоянии *endLength* от конечной вершины обрабатываемого ребра. Если не требуется останавливать сопряжение, то расстояния *begLength* и *endLength* должны принять отрицательные значения. По умолчанию остановка скругления выполняется гранями, перпендикулярными скругляемому ребру. Можно переопределить поведение остановки скругления, используя вектор **vector1** как нормаль останавливающей грани в начале сопряжения и вектор **vector2** как нормаль останавливающей грани в конце сопряжения.

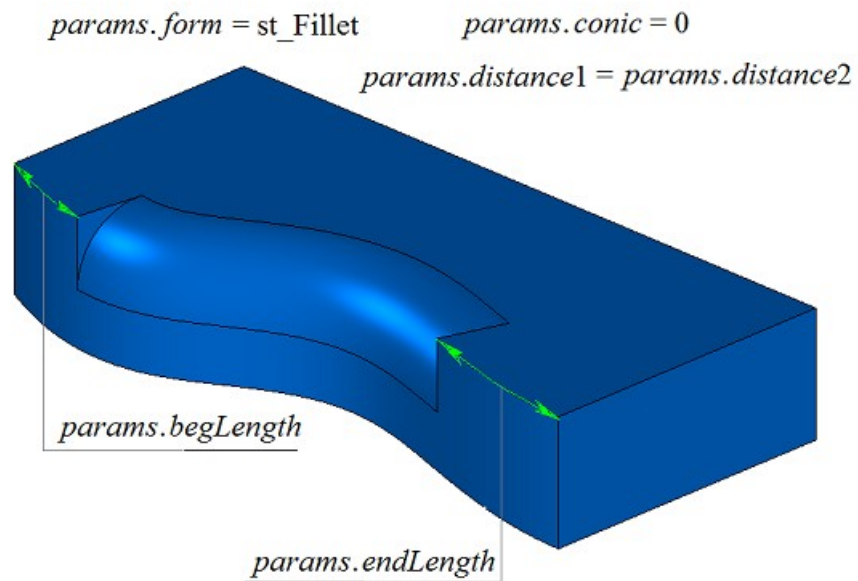


Рис. М.2.10.8.

На примере тела, приведенного на рис. М.2.10.9, продемонстрируем работу флагов *prolong*, *autoSurface*, *keepCant*, при скруглении выделенного на рис. М.2.10.9 ребра.

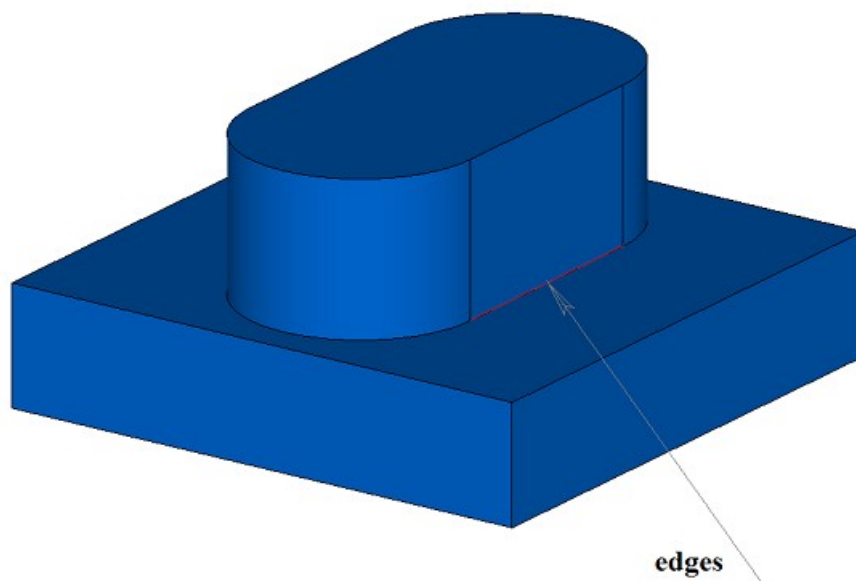
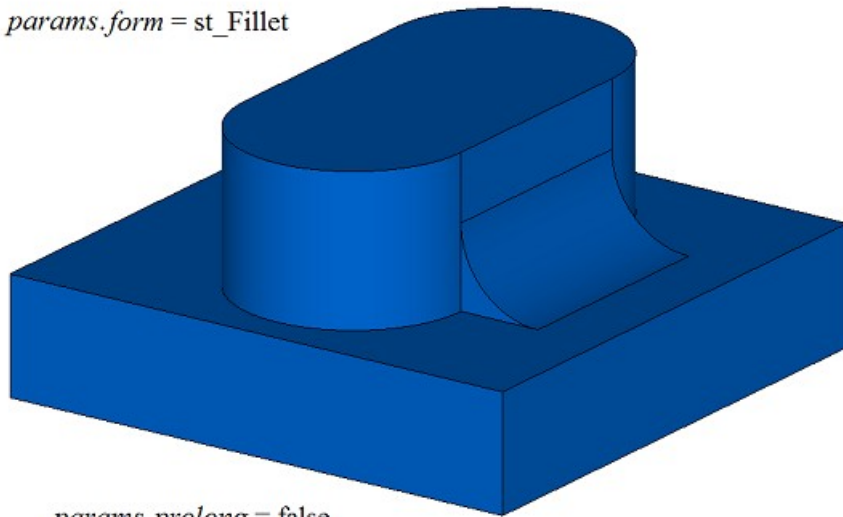


Рис. М.2.10.9.

Флаг *prolong* указывает на то, какие ребра тела должны быть обработаны. Если *prolong=false*, то обработке подлежат только те ребра, которые указаны в контейнере **edges**, рис. М.2.10.10.

params.form = st_Fillet

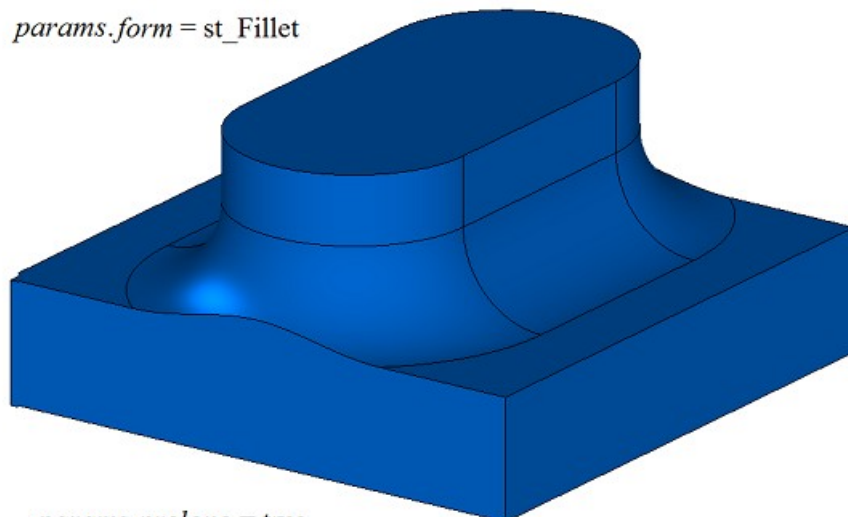


params.prolong = false

Рис. М.2.10.10.

Если *prolong=true*, то обработке подлежат ребра, указанные в контейнере **edges**, а также ребра, гладко стыкующиеся с ними, рис. М.2.10.11.

params.form = st_Fillet

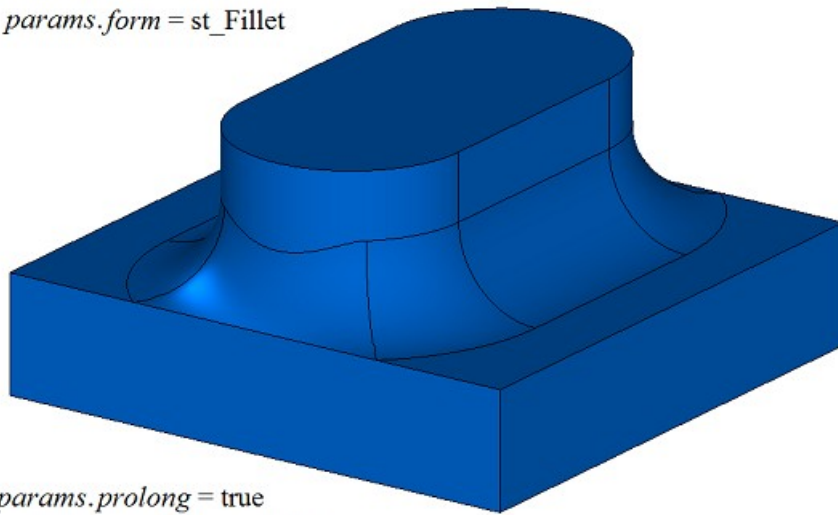


params.prolong = true
params.autoSurface = false
params.keepCant = false

Рис. М.2.10.11.

Флаги *autoSurface* и *keepCant* указывают на обработку ситуаций, когда опорные края грани сопряжения выходят за границу смежной грани. Если *autoSurface=false* и *keepCant=false*, то при выходе опорного края грани сопряжения за границу смежной грани через острое ребро грань сопряжения не меняет своей формы, а «подрезается» соседней гранью, как приведено на рис. М.2.10.11. Если *autoSurface=true* или *keepCant=true*, то при выходе опорного края грани сопряжения за границу смежной грани через острое ребро грань сопряжения меняет свою форму и проходит опорным краем по границе, сохраняя ее неизменной, как приведено на рис. М.2.10.12.

params.form = st_Fillet

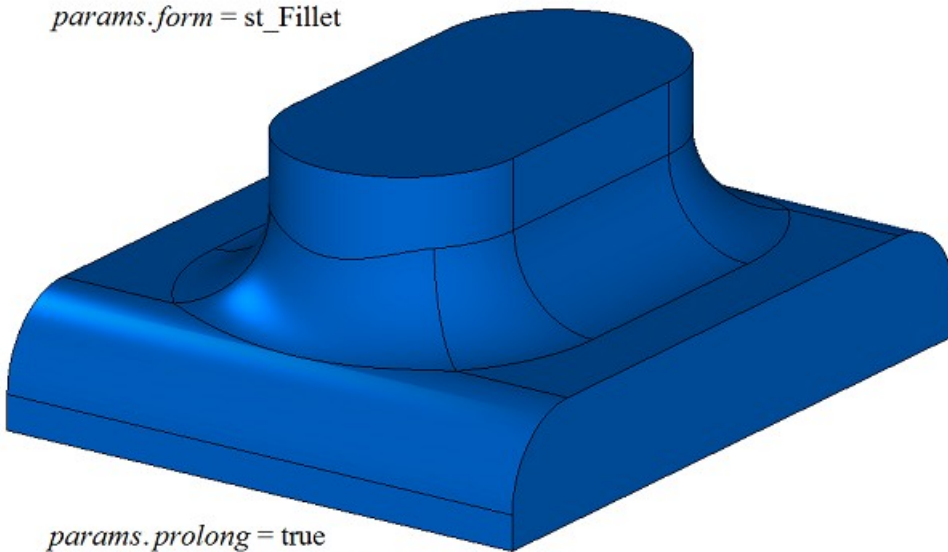


params.prolong = true
params.autoSurface = true
params.keepCant = true

Рис. М.2.10.12.

Если *autoSurface=true* и *keepCant=false*, то при выходе опорного края грани сопряжения за границу смежной грани через гладкое ребро грань сопряжения заменяет смежную грань на соседнюю, изменяя форму на данном участке, рис. М.2.10.13.

params.form = st_Fillet



params.prolong = true
params.autoSurface = true
params.keepCant = false

Рис. М.2.10.13.

На рис. М.2.10.14 приведено скругление четырех ребер, выполненное одним вызовом рассматриваемого метода с флагом *equable=false*.

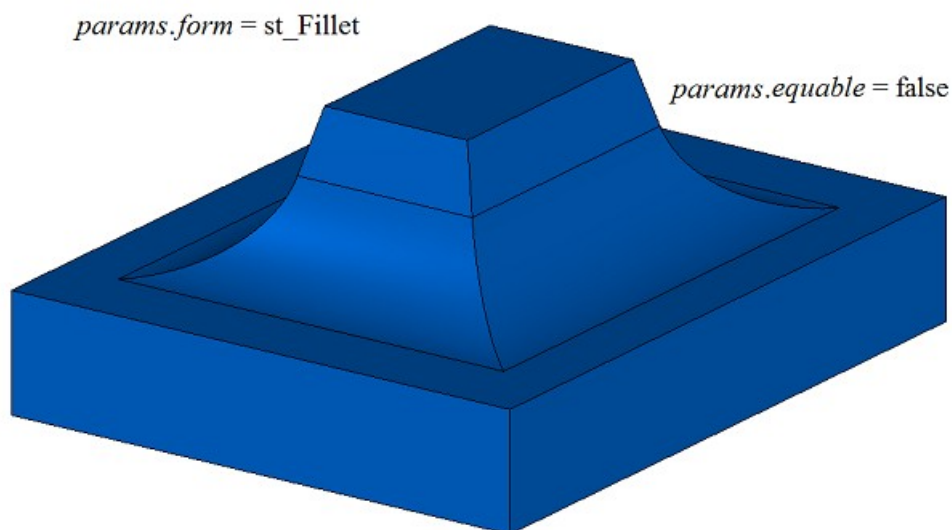


Рис. М.2.10.14.

На рис. М.2.10.15 приведено скругление четырех ребер, выполненное одним вызовом рассматриваемого метода с флагом *equable=true*, указывающим на необходимость вставлять тороидальную поверхность в углах сочленения поверхностей сопряжения.

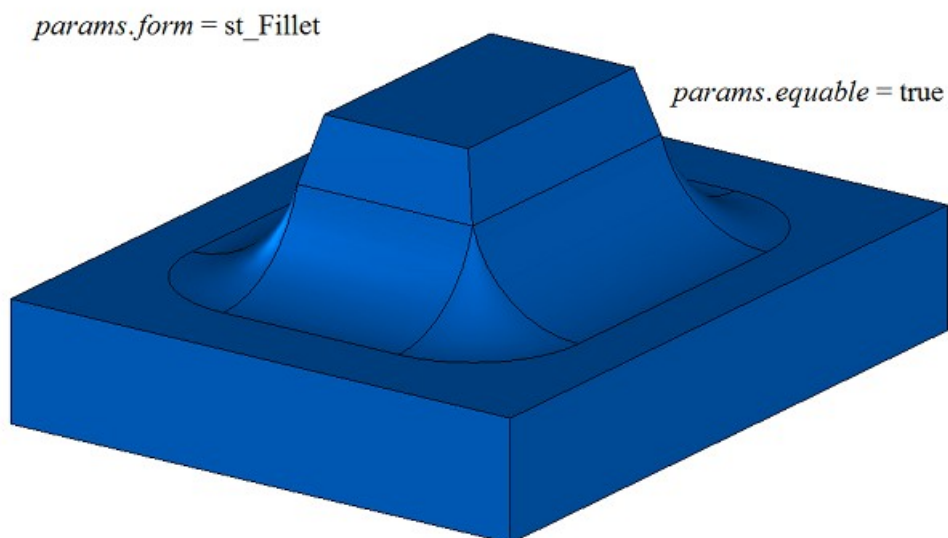
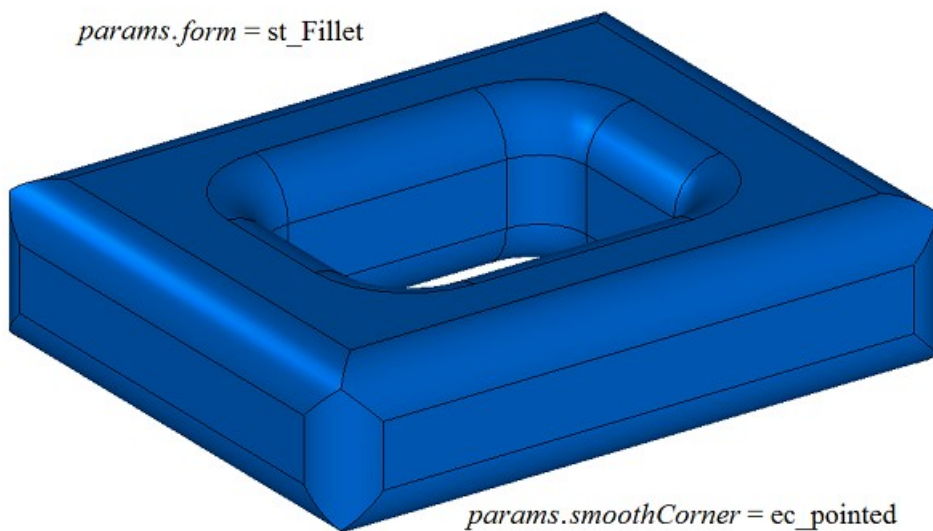


Рис. М.2.10.15.

При построении скругления трех ребер, стыкующихся в одной вершине, параметр *smoothCorner* определяет способ обработки скругления чешонанных углов. Если *smoothCorner=es_pointed*, то обработка углов, в которых стыкуются три ребра одинаковой выпуклости, отсутствует, рис. М.2.10.16.

params.form = st_Fillet

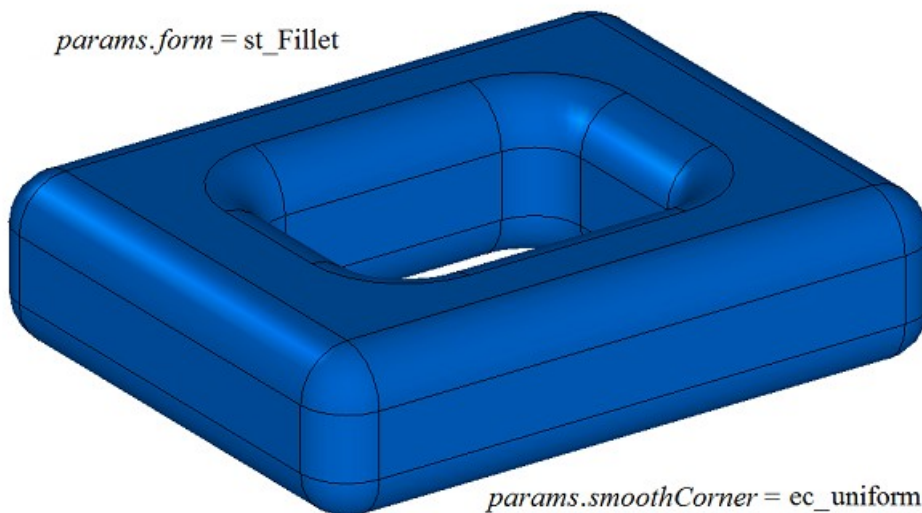


params.smoothCorner = ec_pointed

Рис. М.2.10.16.

Если *smoothCorner=ec_uniform*, то углы, в которых стыкуются три ребра различной выпуклости, обрабатываются одинаковым образом, как показано на рис. М.2.10.17.

params.form = st_Fillet



params.smoothCorner = ec_uniform

Рис. М.2.10.17.

Если *smoothCorner=ec_sharp*, то углы, в которых стыкуются три ребра различной выпуклости, обрабатываются одинаковым образом, как показано на рис. М.2.10.18.

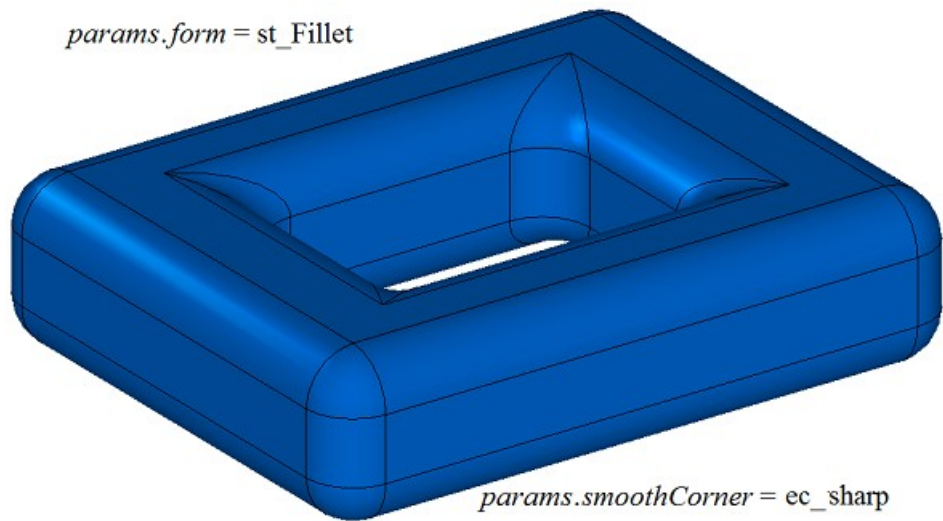


Рис. М.2.10.18

Если *smoothCorner=ec_either*, то углы, в которых стыкуются три ребра различной выпуклости, могут быть обработаны разным образом.

В неоднозначной ситуации на торцах грани сопряжения может быть задействован параметр **bounds** с гранями тела **solid**, которые следует использовать для обрезки граней сопряжения. Пример использования параметра **bounds** приведен на рис. М.2.10.19 и М.2.10.20.

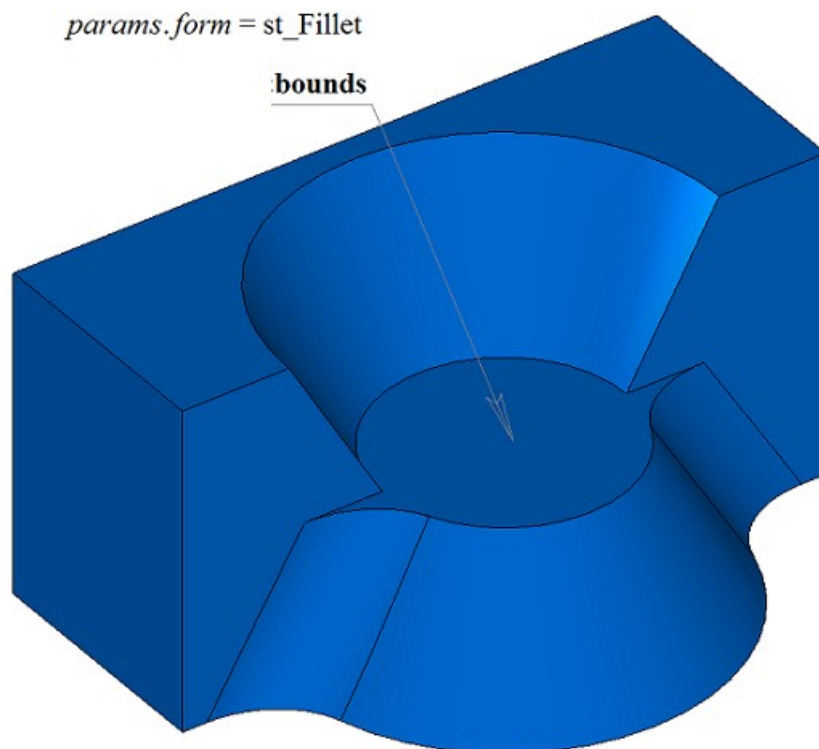


Рис. М.2.10.19

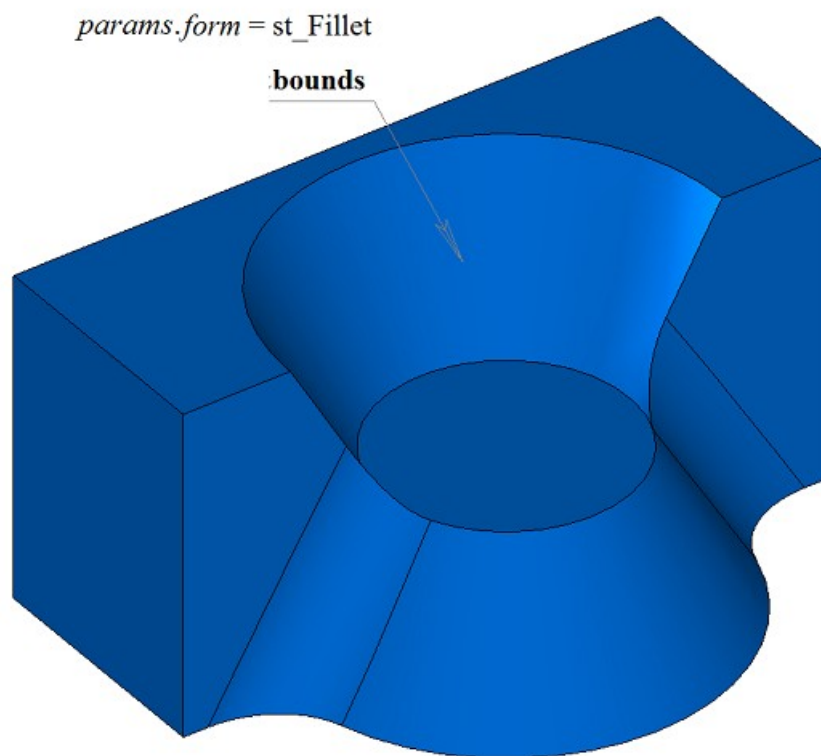


Рис. М.2.10.20

Параметр **bounds** может использоваться для остановки граней сопряжения в начале и конце. При этом грани, определяемые параметром **bounds**, должны принадлежать исходному телу **solid**.

На рис. М.2.10.21 приведена модель, у которой требуется построить скругления указанных ребер, полностью охватывающие отверстие и выступ.

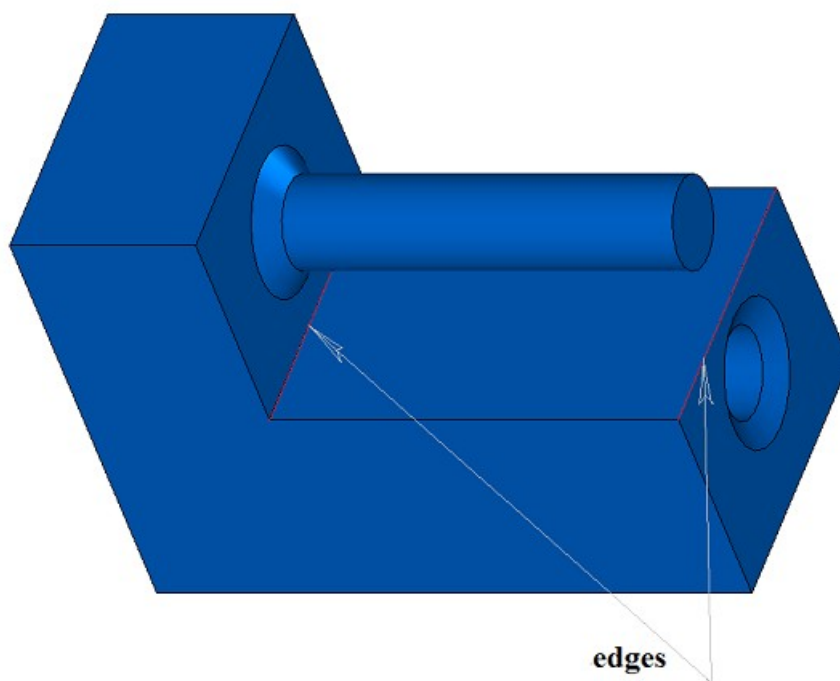


Рис. М.2.10.21

Результат построения скругления с обходом препятствий приведен на рис. М.2.10.22.

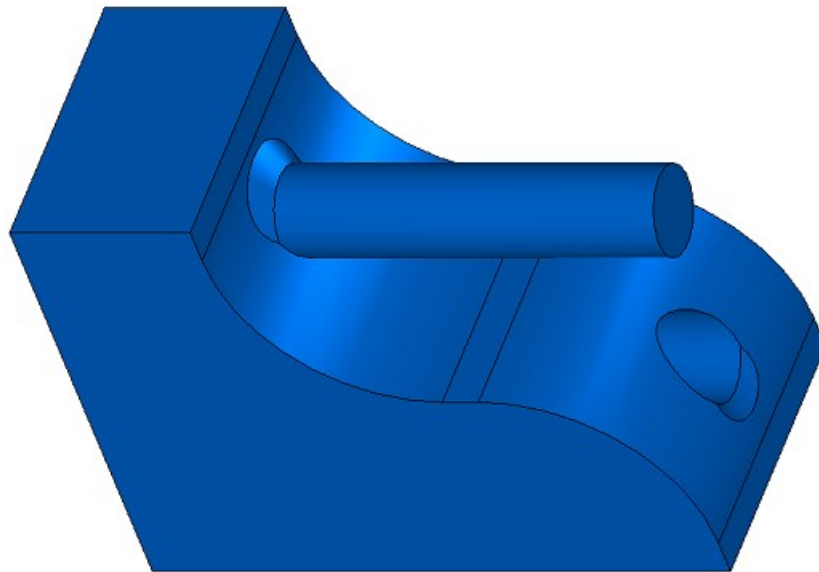


Рис. М.2.10.22

На рис. М.2.10.23 приведен пример одновременного скругления группы из шести ребер, стыкующихся в общей вершине.

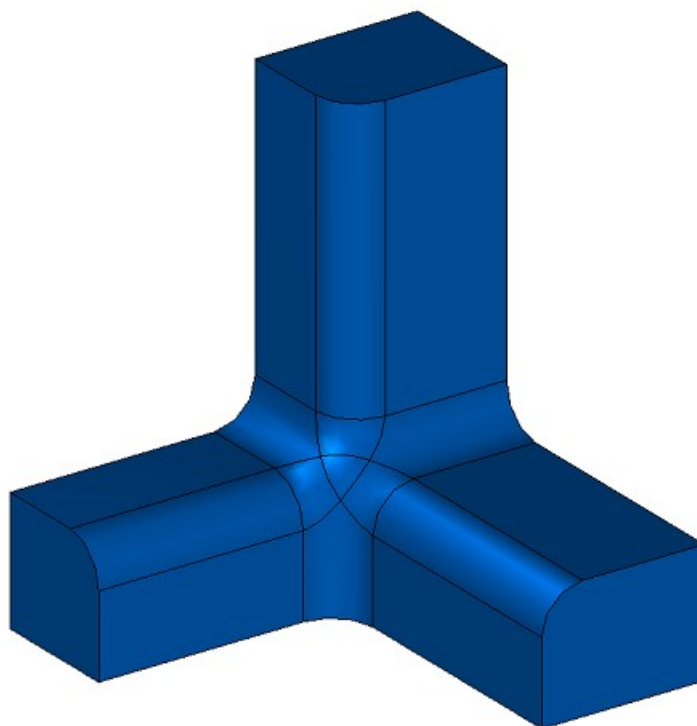


Рис. М.2.10.23

На рис. М.2.10.24 приведен пример скругления нескольких групп из четырех ребер, стыкующихся в общих вершинах. Особенностью скругления служит то обстоятельство, что группы связаны между собой и могут обрабатываться только одновременно. Исходное тело, для показанного на рис. М.2.10.24 тела, получено путем вычитания из куба четырех цилиндров, оси которых совпадают с диагоналями куба.

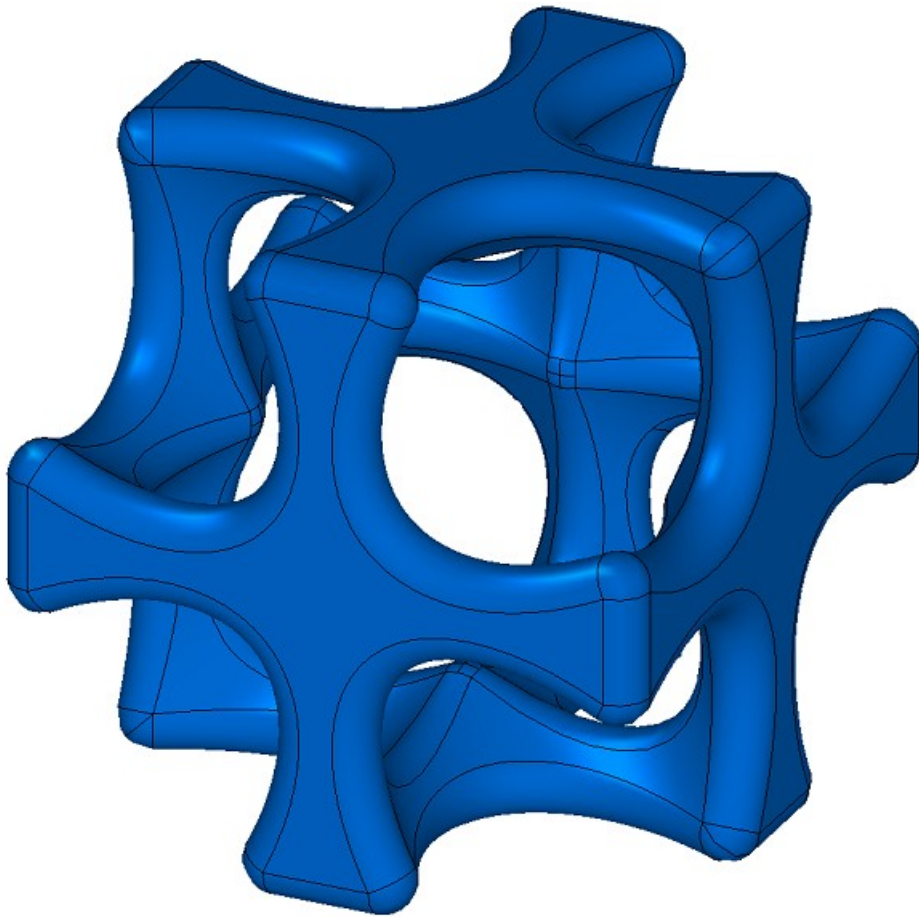


Рис. М.2.10.24

При построении скругления ребер рассматриваемый метод добавляет в журнал построенного тела строитель `MbFilletSolid`, который объявлен в файле `sr_fillet_solid.h`.

Тестовое приложение `test.exe` выполняет обработку ребер тела командами меню «Создать->Тело->Обработкой ребер->Скругление по радиусу», «Создать->Тело->Обработкой ребер->Скругление по хорде».

Когда в методах построения скругления параметры *distance1* и *distance2* не равны и для обработки заданы несколько ребер, то трудно разобраться, что чему соответствует. В такой ситуации предлагается использовать вспомогательные методы: `SmoothPhantom(...)`, `SmoothSequence(...)`, `SmoothPositionData(...)`, объявленные в файле `action_phantom.h`. Метод `SmoothPhantom(...)` строит упрощенные поверхности для имитации будущих скруглений. Метод `SmoothSequence(...)` строит последовательности ребер, к которым по желанию могут быть добавлены гладко стыкующиеся ребра, обрабатываемые совместно. Метод `SmoothPositionData(...)` вычисляет по три точки для обрабатываемых ребер, которые используются для фантомных размеров радиусов, катетов и углов операции. Обычно с помощью указанных методов рисуется некий фантом, по которому можно понять, что получится в результате операции, и дается возможность поменять параметры при необходимости.

Перед построением скруглений обрабатываемые ребра сортируются, при необходимости или по требованию к ним добавляются гладко стыкующиеся ребра, и из ребер создаются последовательности гладко стыкующихся ребер. Параметры *distance1* и *distance2* привязаны к первой и второй поверхностям, соответственно, кривой пересечения первого ребра в последовательности гладко стыкующихся ребер. Поверхности можно получить методами кривой пересечения `GetCurveOneSurface()` и `GetCurveTwoSurface()`. По *distance1* и *distance2* первого ребра определяются параметры для остальных ребер каждой последовательности. Так, если для первого ребра указатель `edge->GetIntersectionCurve().GetSurfaceOne()` равен указателю `&edge->GetFacePlus()->GetSurface().GetSurface()`, то при построении скругления *distance1* будет соответствовать радиусу для грани **facePlus**, а *distance2* будет соответствовать радиусу для грани **faceMinus** первого ребра последовательности.

M.2.11. Скругление рёбер тела переменным радиусом

Метод
MbResultType
FilletSolid ([MbSolid](#) & **solid**,
MbeCopyMode *sameShell*,
SArray<MbEdgeFunction> & **edges**,
RPAArray<[MbFace](#)> & **bounds**,
const SmoothValues & *params*,
const MbSNameMaker & *names*,
[MbSolid](#) *& **result**)

выполняет скругление переменным радиусом указанных рёбер копии исходного тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **edges** – множество скругляемых рёбер с заданными методами изменения радиуса.
- **bounds** – множество граней для обрезки краев граней скругления (может быть пустым),
- *params* – параметры построения,
- *names* – именователь построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType. Метод объявлен в файле `action_solid.h`.

Метод выполняет замену указанных рёбер исходного тела гранями скругления, обеспечивающими гладкое сопряжение смежных граней указанных рёбер. При скруглении ребер сопрягающие грани в поперечном сечении могут иметь форму дуги окружности переменного радиуса. Метод аналогичен методу, описанному в предыдущем параграфе, и отличается от нее третьим параметром **edges**.

Параметр **solid** содержит исходное тело, ребра которого подлежат обработке. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** к построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbeCopyMode описано в параграфе [O.7.9. Копирование множества граней MbFaceShell](#).

Параметр **bounds** содержит грани тела **solid**, которые следует использовать для обрезки скругления в неоднозначной ситуации. Параметр *names* обеспечивает именование граней сопряжения.

Параметры построения скруглений *params* содержит информацию о форме и способе сопряжения смежных граней обрабатываемых ребер, рис. M.2.10.1. Класс SmoothValues описан в файле `shell_parameter.h`.

Входной параметр *params* содержит следующие данные:

- *distance1* – первый радиус скругления,
- *distance2* – второй радиус скругления,
- *conic* – коэффициент формы поверхности сопряжения,
- *begLength* – расстояние от начальной вершины до точки остановки сопряжения (отрицательное значение означает отсутствие остановки),
- *endLength* – расстояние от конечной вершины до точки остановки сопряжения (отрицательное значение означает отсутствие остановки),
- *form* – тип сопряжения из перечисления MbeSmoothForm,
- *smoothCorner* – способ скругления чешуйчатых углов,
- *prolong* – флаг продолжения скругления по касательным рёбрам,
- *autoSurface* – флаг автоопределения сохранения кромки,
- *keepCant* – флаг сохранения кромки,
- *strict* – флаг строгости построения: при false - скруглить хотя бы то, что возможно,
- *equable* – флаг вставки тороидальной поверхности в углах сочленения поверхности сопряжения,
- **vector1** – вектор нормали плоскости остановки сопряжения в начале,
- **vector2** – вектор нормали плоскости остановки сопряжения в конце.

Параметр **edges** содержит обрабатываемые ребра тела **solid** и функции изменения радиуса вдоль ребра. Каждый элемент множества **edges** состоит из указателя на ребро и указателя на скалярную

функцию, на значения которой будут умножены первый и второй радиусы скругления $distance1$ и $distance2$, рис. М.2.11.1.

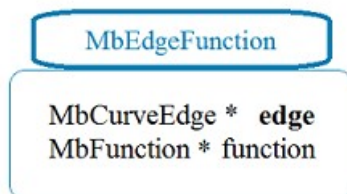


Рис. М.2.11.1.

Типом скругления управляет параметр $form$. Для скругления ребер переменного радиуса используются значения параметра $form$ равное st_Fillet , другие значения $form$ рассматриваемый метод не использует. Для каждой точки $point$ обрабатываемого ребра $edges[i].edge \rightarrow Point(t, point)$ радиусы поверхности скругления равны параметрам $distance1$ и $distance2$, умноженным на значение функции ребра $edges[i].function \rightarrow Value(t)$. На рис. М.2.11.2 приведено скругление с переменными радиусами ребра прямоугольной призмы. При $distance1=distance2$ и $conic=0$ поверхность скругления строится путем движения сферы переменного радиуса, касающейся двух смежных для скругляемого ребра граней. Опорные края грани сопряжения проходят по точкам касания сферы и соответствующей смежной грани. Поперечное сечение грани сопряжения представляет собой дугу окружности.

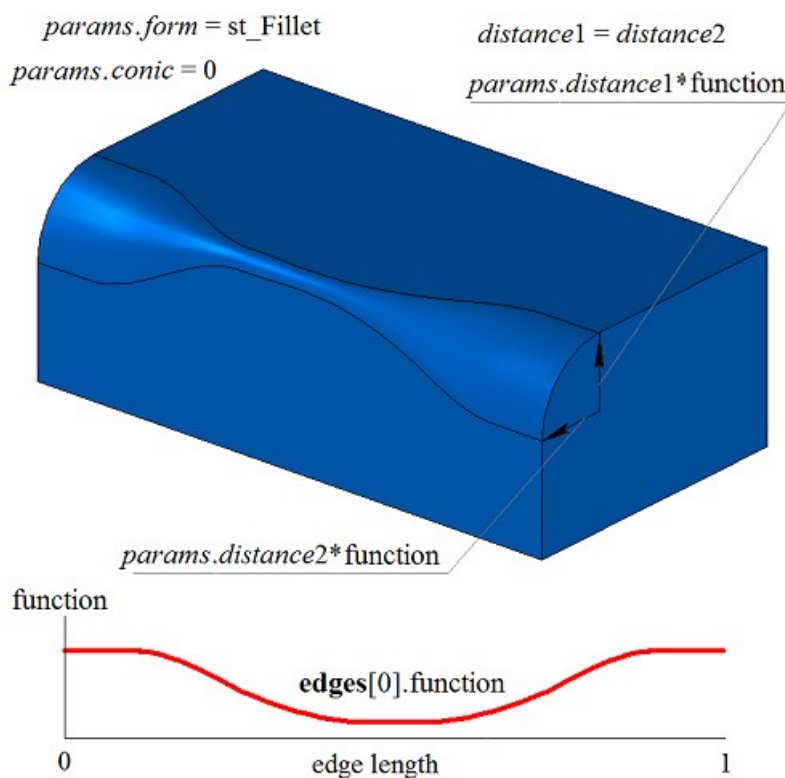


Рис. М.2.11.2.

На рис. М.2.11.3 приведено эллиптическое скругление с переменными радиусами ребра прямоугольной призмы.

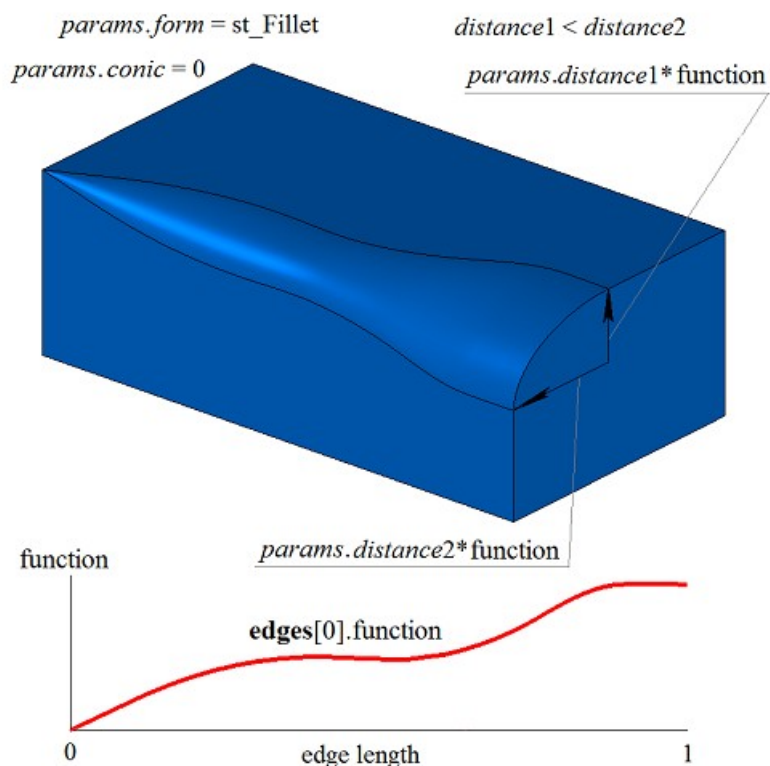


Рис. М.2.11.3.

Коэффициент *conic* управляет формой поверхности скругления. При *conic*=0 (макрос `_ARC_`) сечение поверхности сопряжения имеет форму дуги окружности или эллипса с заданными радиусами. Кроме нулевого значения коэффициент формы может принимать значения от 0.05 до 0.95. При *conic*=0.5 поперечное сечение грани скругления представляет собой дугу параболы. При *conic*>0.5 поперечное сечение грани скругления представляет собой дугу гиперболы. При *conic*<0.5 поперечное сечение грани скругления представляет собой дугу эллипса. На рис. М.2.11.4, М.2.11.5 приведены скругления с переменными радиусами ребра прямоугольной призмы с разными коэффициентами формы.

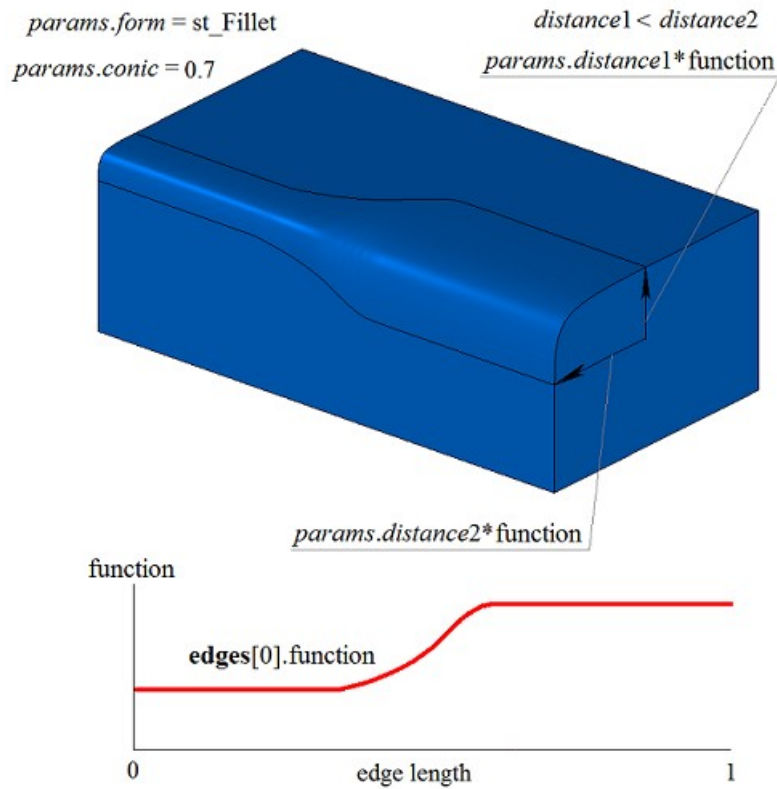


Рис. М.2.11.4.

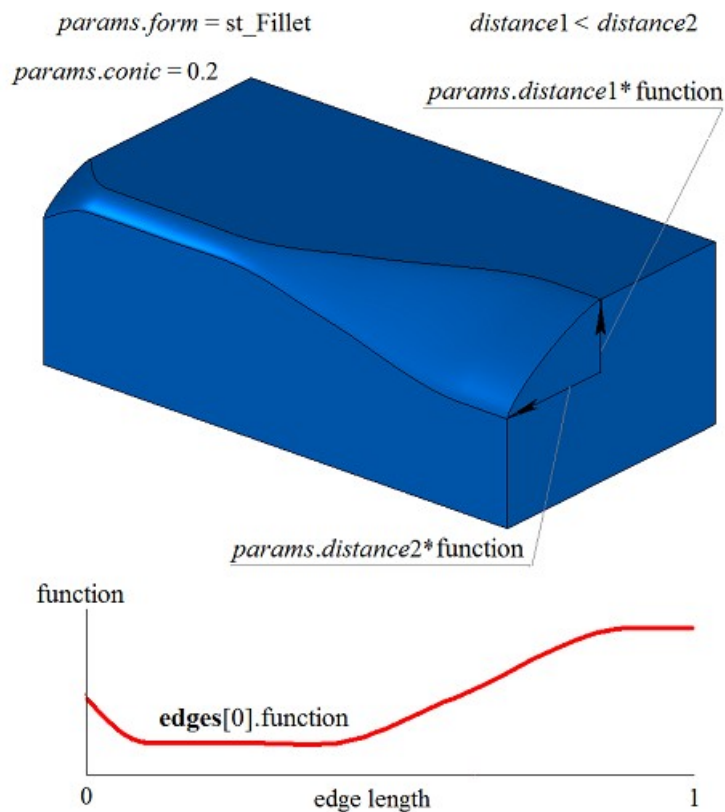


Рис. М.2.11.5.

На рис. М.2.11.6 приведен пример остановки скругления на расстоянии *begLength* от начальной вершины и на расстоянии *endLength* от конечной вершины обрабатываемого ребра. При задании параметров остановки скругления метод изменения радиусов скругления задается для всего ребра.

Если не требуется останавливать сопряжение, то расстояния *begLength* и *endLength* должны принять отрицательные значения.

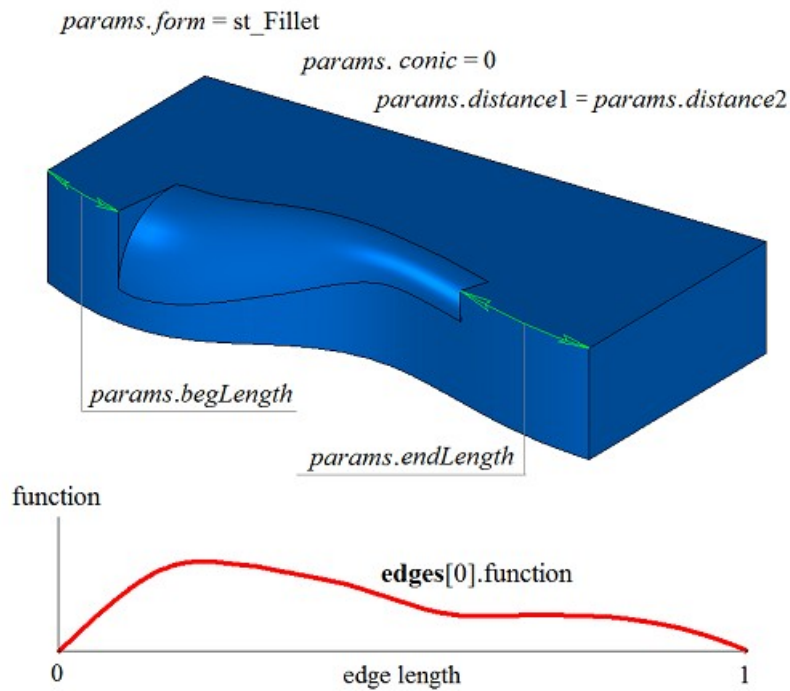


Рис. М.2.11.6.

Флаг *prolong* указывает на то, какие ребра тела должны быть обработаны. Если *prolong=false*, то обработке подлежат только те ребра, которые указаны в контейнере **edges**. Если *prolong=true*, то обработке подлежат ребра, указанные в контейнере **edges**, а также ребра, гладко стыкующиеся с ними. Для ребер продолжения скругления метод изменения радиуса принимает константное значение, равное значению функции на краю предыдущего гладко стыкующегося ребра. На рис. М.2.11.7 приведено исходное тело, ребра, подлежащие скруглению, и метод изменения радиуса для указанных ребер.

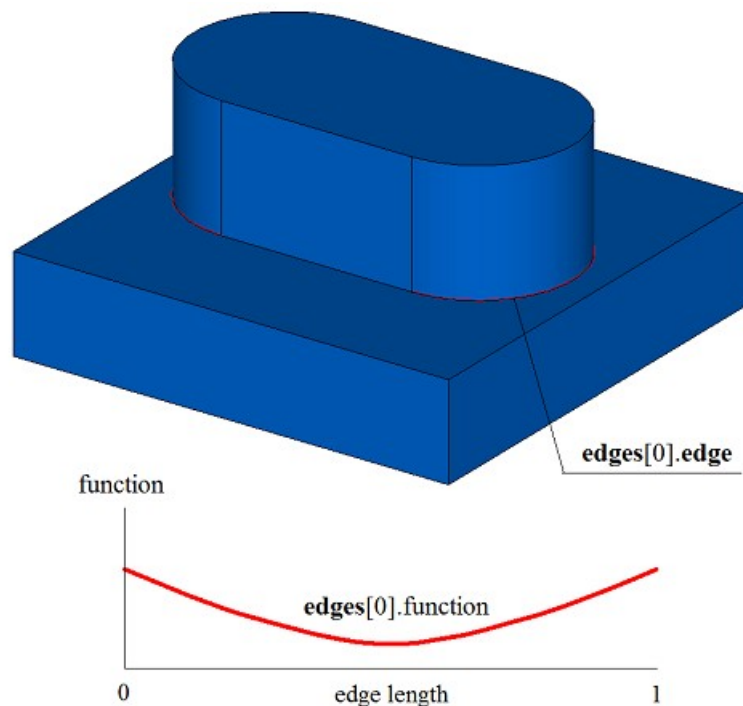


Рис. М.2.11.7.

На рис. М.2.11.8 приведен результат работы рассматриваемого метода для исходного тела и метод изменения радиуса для ребер, указанных на рис. М.2.11.7. На рис. М.2.11.8 видно, что на краях метод изменения радиуса был подобран, чтобы обеспечить гладкое сопряжение с соседней гранью скругления.

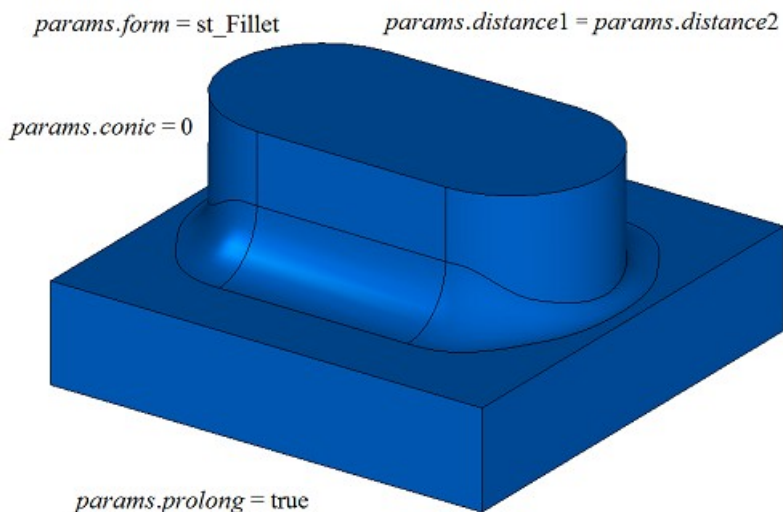


Рис. М.2.11.8.

Флаги *autoSurface*, *keepCant* и *equable* в рассматриваемом методе не используются.

При построении скругления трех ребер, стыкующихся в одной вершине, параметр *smoothCorner* определяет способ обработки скругления чешоданных углов. Если *smoothCorner=ec_pointed*, то обработка углов, в которых стыкуются три ребра одинаковой выпуклости, отсутствует, рис. М.2.10.16. Если *smoothCorner=ec_uniform*, то углы, в которых стыкуются три ребра различной выпуклости, обрабатываются одинаковым образом, как показано на рис. М.2.10.17. Если *smoothCorner=ec_sharp*, то углы, в которых стыкуются три ребра различной выпуклости, обрабатываются одинаковым образом, как показано на рис. М.2.10.18. Если *smoothCorner=ec_either*, то углы, в которых стыкуются три ребра различной выпуклости, могут быть обработаны разным образом. При несовпадении функций изменения радиуса на краях граней скругления функции изменения радиуса корректируются так, чтобы обеспечить гладкое сопряжение стыкующихся граней скругления.

В неоднозначной ситуации на торцах грани сопряжения может быть задействован параметр **bounds** с гранями тела **solid**, которые следует использовать для обрезки граней сопряжения. Пример использования параметра **bounds** приведен на рис. М.1.20.19 и М.1.20.20.

Параметр **bounds** может использоваться для остановки граней сопряжения в начале и конце. При этом грани, определяемые параметром **bounds**, должны принадлежать исходному телу **solid**.

При построении скругления ребер рассматриваемый метод добавляет в журнал построенного тела строитель **MbFilletSolid**, который объявлен в файле `cr_fillet_solid.h`.

Тестовое приложение `test.exe` выполняет обработку ребер тела командой меню «Создать->Тело->Обработкой ребер->Скругление переменное».

М.2.12. Построение тела с фасками рёбер

Метод

MbResultType

```
ChamferSolid ( MbSolid & solid,  
               MbeCopyMode sameShell,  
               RPAArray<MbCurveEdge> & edges,  
               const SmoothValues & params,  
               const MbSNameMaker & names,  
               MbSolid *& result )
```

выполняет построение фасок указанных рёбер на копии исходного тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- **sameShell** – вариант копирования исходного тела,
- **edges** – множество обрабатываемых рёбер,
- **params** – параметры построения,
- **names** – именованная коллекция построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод выполняет замену указанных рёбер исходного тела гранями фасок.

Метод объявлен в файле `action_solid.h`.

Метод выполняет замену указанных рёбер исходного тела гранями фаски.

Параметр **solid** содержит исходное тело, ребра которого подлежат обработке. Параметр *sameShell* управляет передачей граней, рёбер и вершин от исходного тела **solid** к построенному телу **result**.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление `MbFaceShell` описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

Параметр **edges** содержит обрабатываемые ребра тела **solid**. Параметр **names** обеспечивает именование граней фаски.

Для построения фасок рёбер используются те же параметры `SmoothValues` & *params*, что и при построении скруглений рёбер, рис. М.1.20.1. Класс `SmoothValues` описан в файле `shell_parameter.h`. Параметры построения фаски *params* содержит информацию о форме и способе сопряжения смежных граней обрабатываемых рёбер. Для построения фасок используются следующие данные параметра *params*:

- *distance1* – первый катет фаски,
- *distance2* – второй катет фаски,
- *begLength* – расстояние от начальной вершины до точки остановки сопряжения (отрицательное значение означает отсутствие остановки),
- *endLength* – расстояние от конечной вершины до точки остановки сопряжения (отрицательное значение означает отсутствие остановки),
- *form* – тип сопряжения из перечисления `MbSmoothForm`,
- *smoothCorner* – способ скругления чешуйчатых углов,
- *prolong* – флаг продолжения скругления по касательным рёбрам,
- **vector1** – вектор нормали плоскости остановки сопряжения в начале,
- **vector2** – вектор нормали плоскости остановки сопряжения в конце,

величины *conic*, *autoSurface*, *keepCant*, *strict*, *equable* при построении фаски не используются,

Способом описания фаски управляет параметр *form*. Для построения фаски рёбер используются значения параметра *form* равные `st_Chamfer`, `st_Slant1` и `st_Slant2`. При значении *form*=`st_Chamfer` рассматриваемый метод строит поверхность фаски с заданными катетами, которые определяют параметры *distance1* и *distance2*, рис. М.2.12.1.

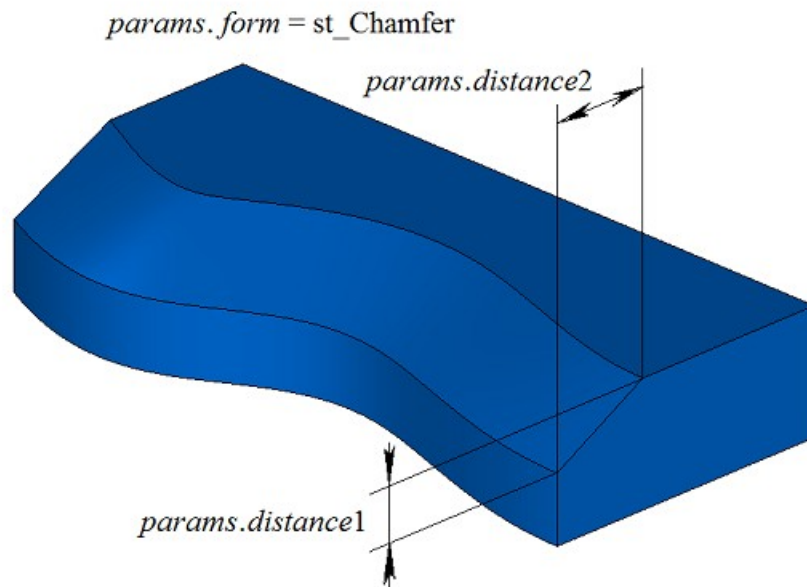


Рис. М.2.12.1.

При значении *form=st_Slant1* рассматриваемый метод строит поверхность фаски с заданным катетом и прилегающим к нему углом. Катет определяет параметр *distance1*, а *distance2* соответствует катету, обеспечивающему заданный прилежащий угол, рис. М.2.12.2.

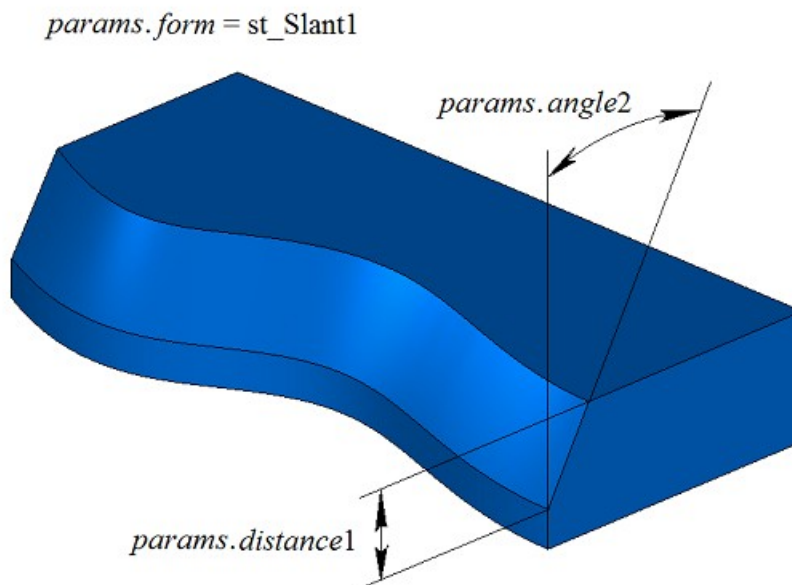


Рис. М.2.12.2.

При значении *form=st_Slant2* рассматриваемый метод строит поверхность фаски с заданным углом и прилегающим к нему катетом. Параметр *distance1* соответствует катету, обеспечивающему заданный угол, а *distance2* определяет прилежащий катет, рис. М.2.12.3.

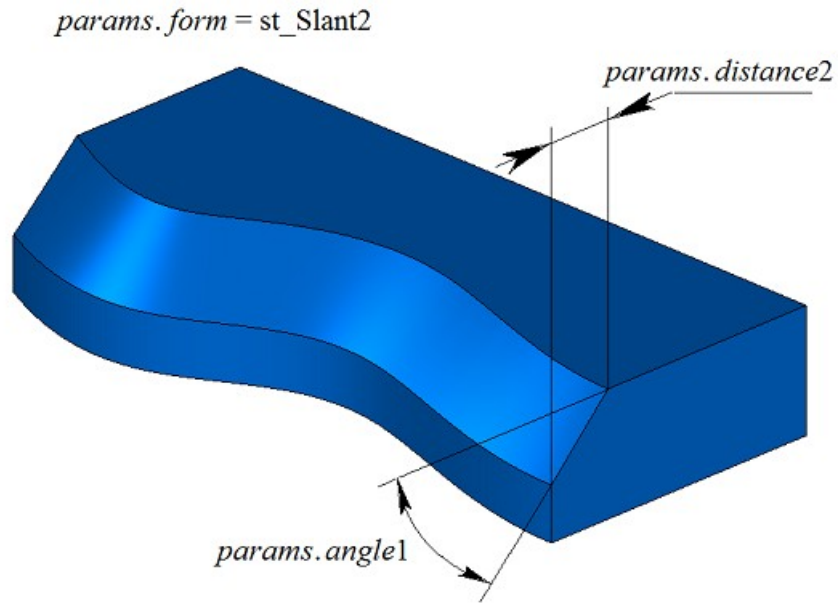


Рис. М.2.12.3.

На рис. М.2.12.4 приведен пример остановки фаски на расстоянии *begLength* от начальной вершины и на расстоянии *endLength* от конечной вершины обрабатываемого ребра. Если не требуется останавливать сопряжение, то расстояния *begLength* и *endLength* должны принять отрицательные значения.

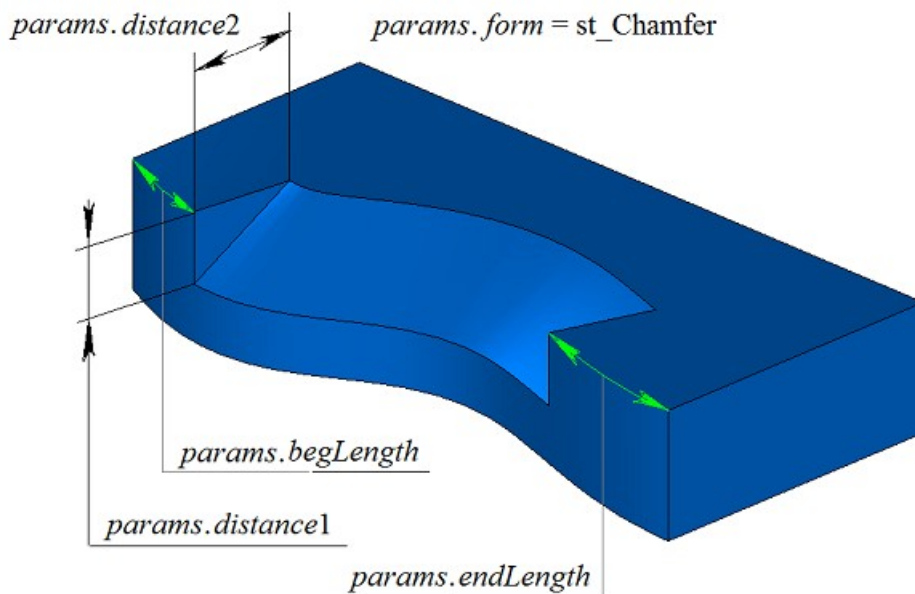


Рис. М.2.12.4.

На примере тела, приведенного на рис. М.2.12.5, продемонстрируем работу флага *prolong* при фаске выделенного на рис. М.2.12.5 ребра.

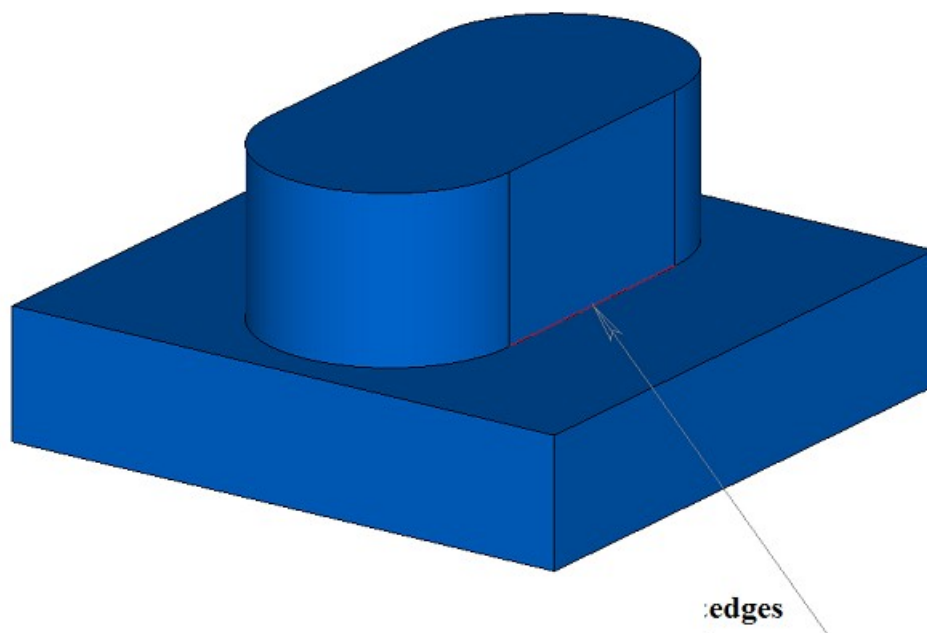


Рис. М.2.12.5.

Флаг *prolong* указывает на то, какие ребра тела должны быть обработаны. Если *prolong=false*, то обработке подлежат только те ребра, которые указаны в контейнере **edges**, рис. М.2.12.6.

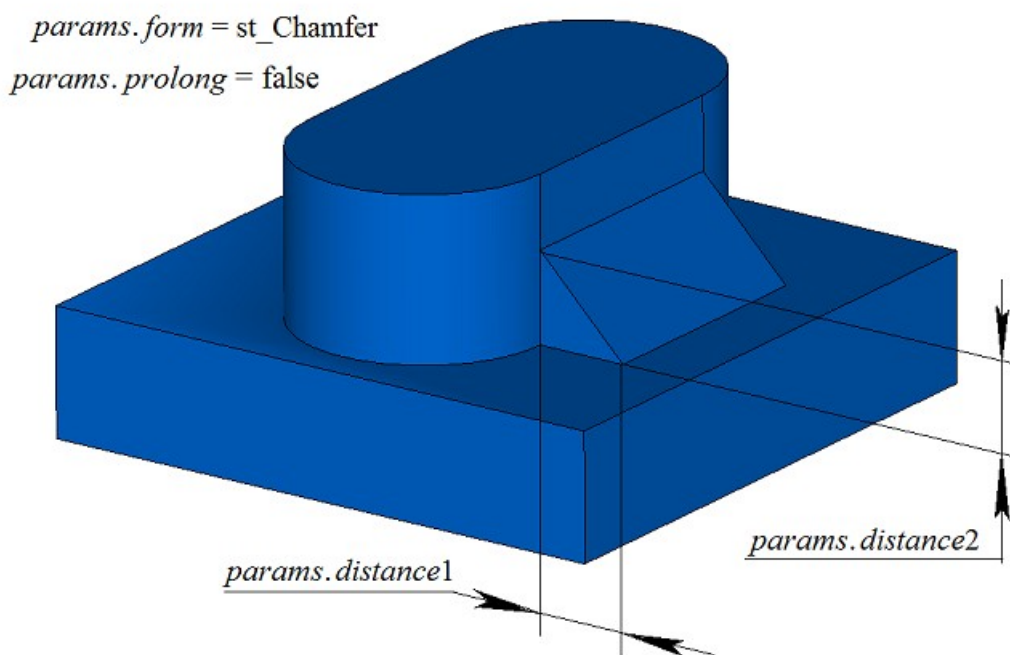


Рис. М.2.12.6.

Если *prolong=true*, то обработке подлежат ребра, указанные в контейнере **edges**, а также ребра, гладко стыкующиеся с ними, рис. М.2.12.7.

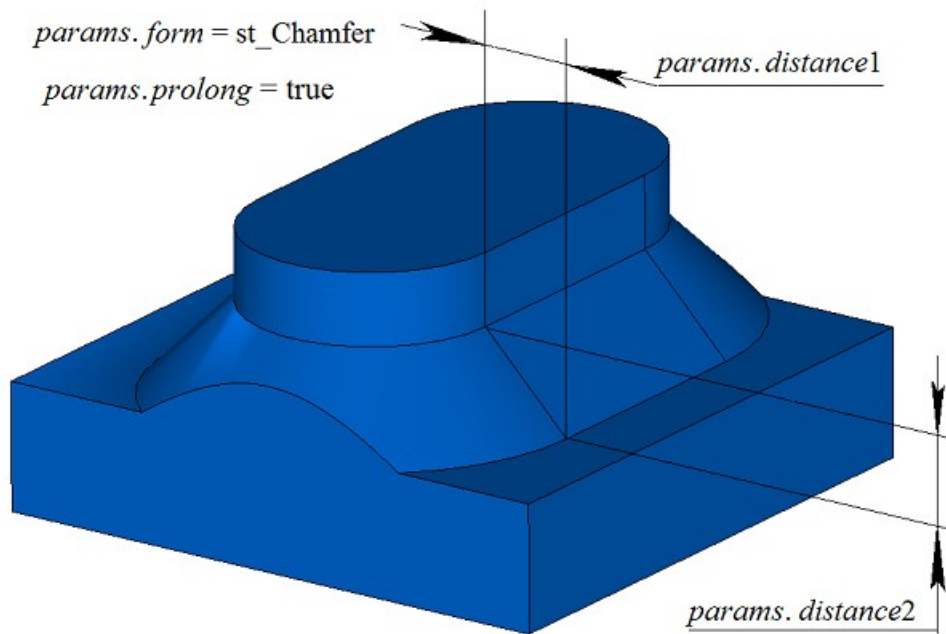


Рис. М.2.12.7.

При построении фаски трех ребер, стыкующихся в одной вершине, параметр *smoothCorner* определяет способ обработки чешотанных углов. На примере тела, приведенного на рис. М.2.12.8, покажем работу параметра *smoothCorner* при построении фаски на всех ребрах тела.

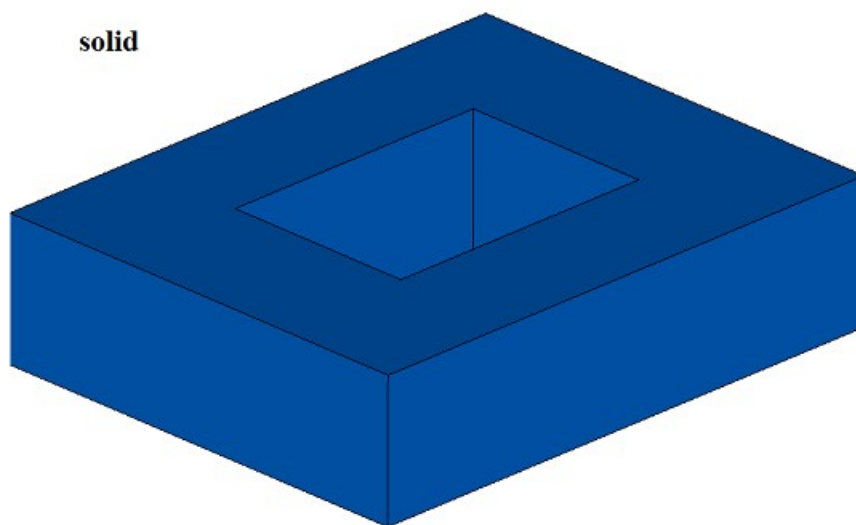


Рис. М.2.12.8.

Если *smoothCorner=ec_pointed*, то обработка углов, в которых стыкуются три ребра одинаковой выпуклости, отсутствует, и в построенном теле присутствует точка, в которой стыкуются три грани фаски, рис. М.2.12.9.

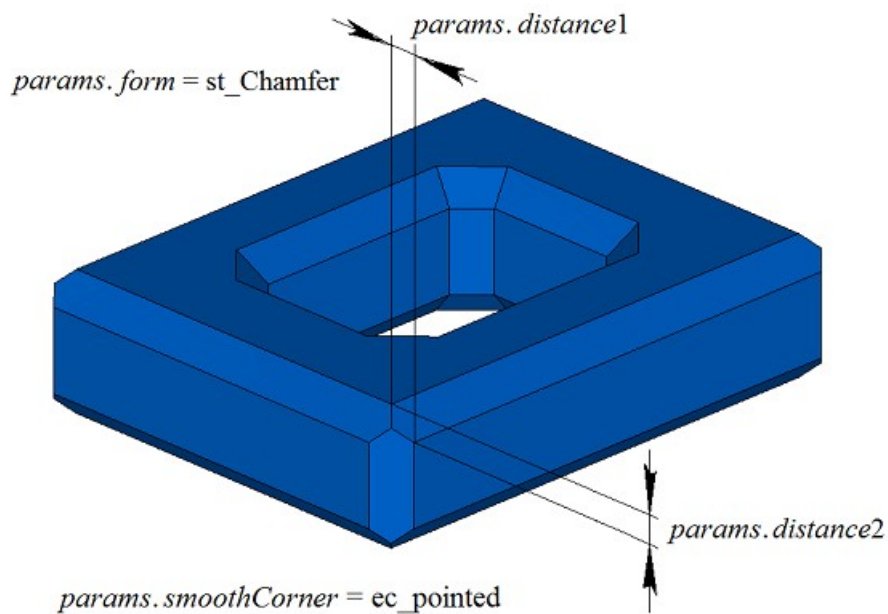


Рис. М.2.12.9.

При всех других значениях параметра *smoothCorner* углы, в которых стыкуются три ребра, обрабатываются путем построения дополнительной грани, как показано на рис. М.2.12.10.

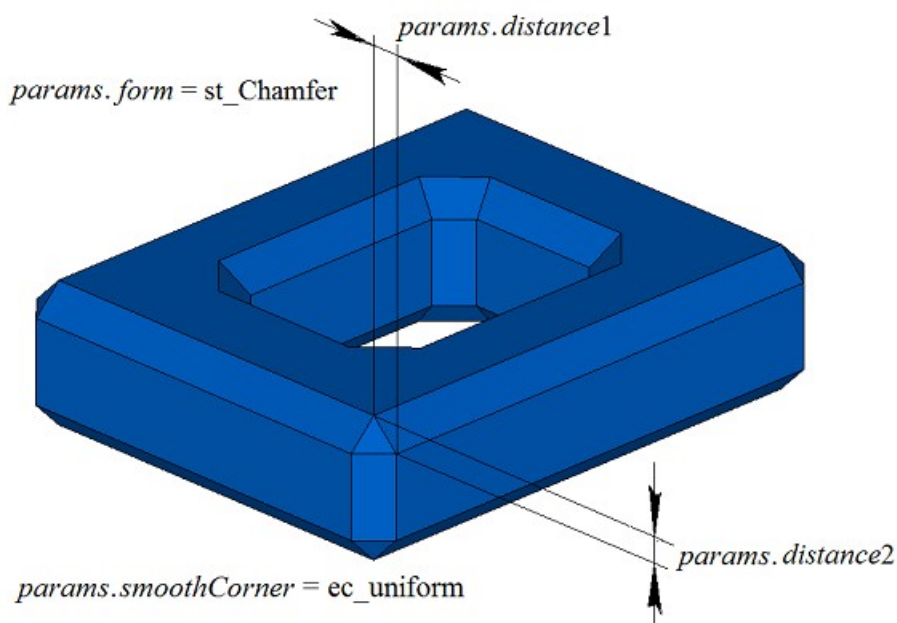


Рис. М.2.12.10.

На примере тела в форме пирамиды, приведенного на рис. М.2.12.11, покажем работу рассматриваемого метода при построении фаски в частных случаях. Результат построения при симметричной конфигурации ребер и симметричной фаске приведен на рис. М.2.12.12.

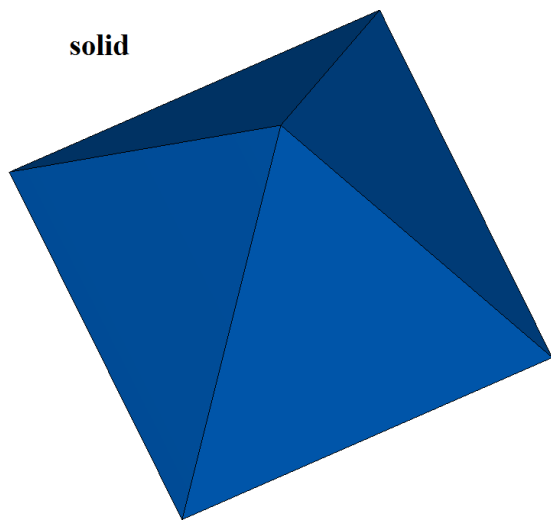


Рис. М.2.12.11.

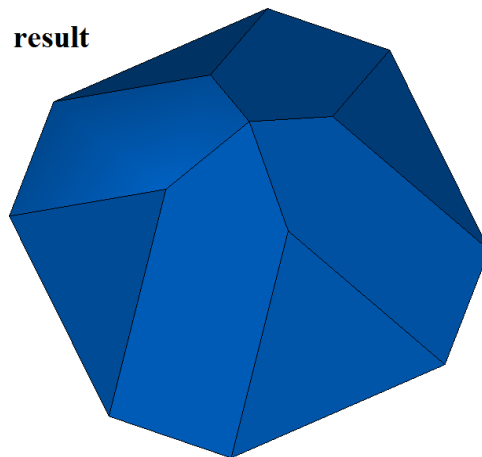


Рис. М.2.12.12.

Следует заметить, что для построения фаски стыкующихся в одной вершине четырех и более ребер необходимо пересечение всех поверхностей фаски в одной точке. Пример симметричной фаски семи ребер, стыкующихся в общей вершине, приведен на рис. М.2.12.13.

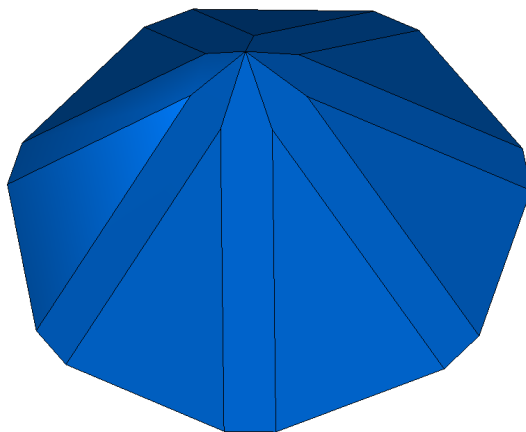


Рис. М.2.12.13.

При построении фаски ребер рассматриваемый метод добавляет в журнал построенного тела строитель `MbChamferSolid`, который объявлен в файле `sr_chamfer_solid.h`.

Тестовое приложение `test.exe` выполняет обработку ребер тела командами меню «Создать->Тело->Обработкой ребер->Фаска катет-катет», «Создать->Тело->Обработкой ребер->Фаска катет-угол», «Создать->Тело->Обработкой ребер->Фаска угол- катет».

Когда в методе построения фаски параметры `distance1` и `distance2` не равны и для обработки заданы несколько ребер, то трудно разобраться, что чему соответствует. В такой ситуации предлагается использовать вспомогательные методы: `SmoothPhantom(...)`, `SmoothSequence(...)`, `SmoothPositionData(...)`, объявленные в файле `action_phantom.h`. Метод `SmoothPhantom(...)` строит упрощенные поверхности для имитации будущих фасок. Метод `SmoothSequence(...)` строит последовательности ребер, к которым по желанию могут быть добавлены гладко стыкующиеся ребра, обрабатываемые совместно. Метод `SmoothPositionData(...)` вычисляет по три точки для обрабатываемых ребер, которые используются для фантомных размеров катетов и углов. Обычно с помощью указанных методов рисуется некий фантом, по которому можно понять, что получится в результате операции, и дается возможность поменять параметры при необходимости.

Перед построением фасок обрабатываемые ребра сортируются, при необходимости или по требованию к ним добавляются гладко стыкующиеся ребра, и из ребер создаются последовательности

гладко стыкующихся ребер. Параметры *distance1* и *distance2* привязаны к первой и второй поверхностям, соответственно, кривой пересечения первого ребра в последовательности гладко стыкующихся ребер. Поверхности можно получить методами кривой пересечения [GetCurveOneSurface\(\)](#) и [GetCurveTwoSurface\(\)](#). По *distance1* и *distance2* первого ребра определяются параметры для остальных ребер каждой последовательности. Так, если для первого ребра указатель **edge**->[GetIntersectionCurve\(\).GetSurfaceOne\(\)](#) равен указателю **&edge**->[GetFacePlus\(\)](#)->[GetSurface\(\).GetSurface\(\)](#), то *distance1* будет соответствовать катету на грани **faceMinus**, а *distance2* будет соответствовать катету на грани **facePlus** первого ребра последовательности.

М.2.13. Построение тонкостенного тела

Метод
 MbResultType
ThinSolid ([MbSolid](#) & **solid**,
 MbcCopyMode *sameShell*,
 RPAArray<[MbFace](#)> & **outFaces**,
 SweptValues & *params*,
 const MbSNameMaker & names,
 [MbSolid](#) *& **result**)

выполняет построение тонкостенного тела исключением указанных граней исходного тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **outFaces** – множество исключаемых граней,
- *params* – параметры построения,
- names – именованная коллекция построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_solid.h`.

Метод удаляет из исходного тела **solid** указанные грани **outFaces**, а оставшимся граням «придаёт заданную толщину». Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**. Параметр *params* содержит информацию о толщине стенки сохраняемых граней и замкнутости построенного тела **result**. Толщина стенки сохраняемых граней может быть равна или *params.thickness1* в положительном направлении нормали грани или *params.thickness2* в отрицательном направлении нормали грани. Если *params.shellClosed=false*, то будет построено незамкнутое тело. Параметр names обеспечивает именование граней построенного тела. Для выполнения операции удаляемые грани **outFaces** по общему периметру не должны иметь гладких ребер стыковки с сохраняемыми гранями исходного тела.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbcCopyMode описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

На рис. М.2.13.1 показаны исходное тело **solid** и удаляемые грани **outFaces**.

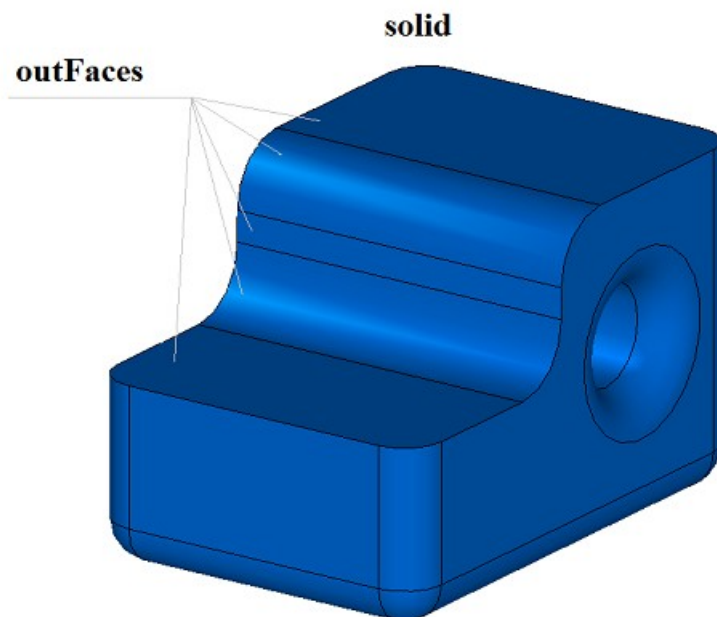


Рис. М.2.13.1.

На рис. М.2.13.2 приведено построенное тонкостенное тело **result** с утолщением сохраненных граней внутрь исходного тела.

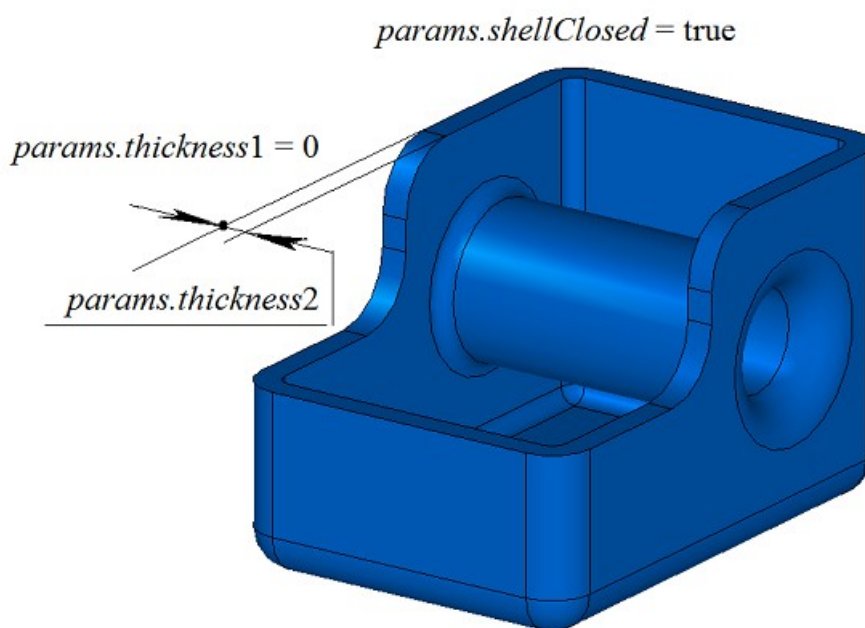


Рис. М.2.13.2.

На рис. М.2.13.3 приведено построенное тонкостенное тело **result** с утолщением сохраненных граней наружу исходного тела.

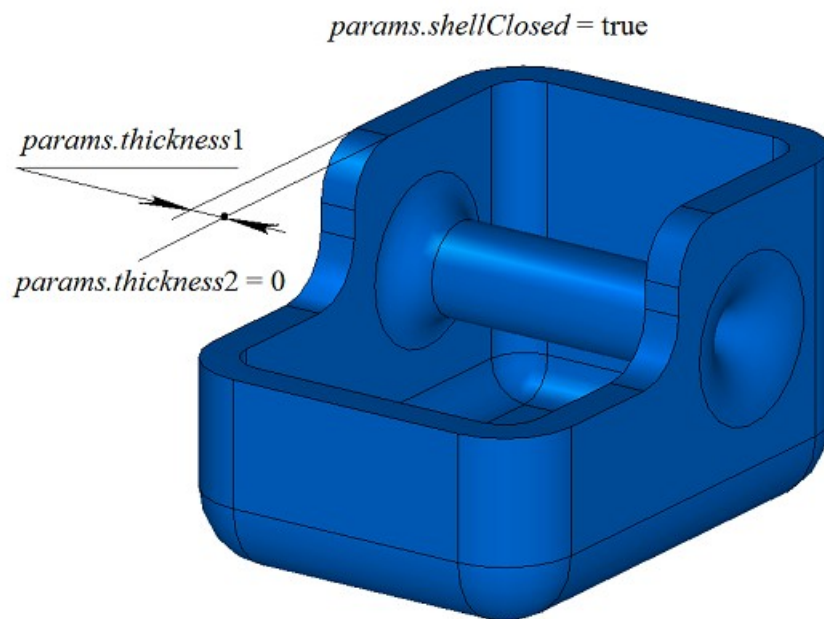


Рис. М.2.13.3.

На рис. М.2.13.4 приведено построенное не замкнутое тело **result**.

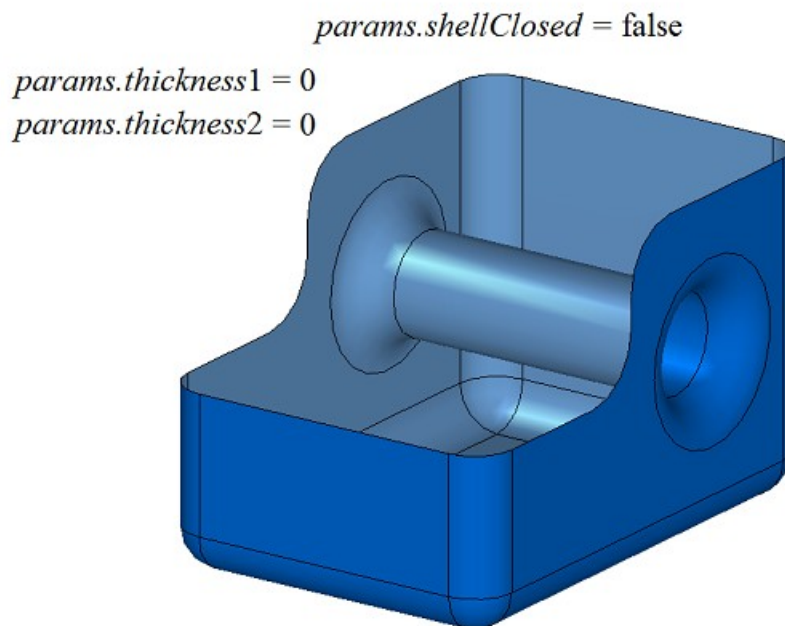


Рис. М.2.13.4.

На рис. М.2.13.5 приведено тонкостенное тело, построенное при пустом множестве удаляемых граней.


```
outFaces.Count() = 0
params.thickness2 = 0
params.thickness1 > 0
```

```
params.shellClosed = true
```

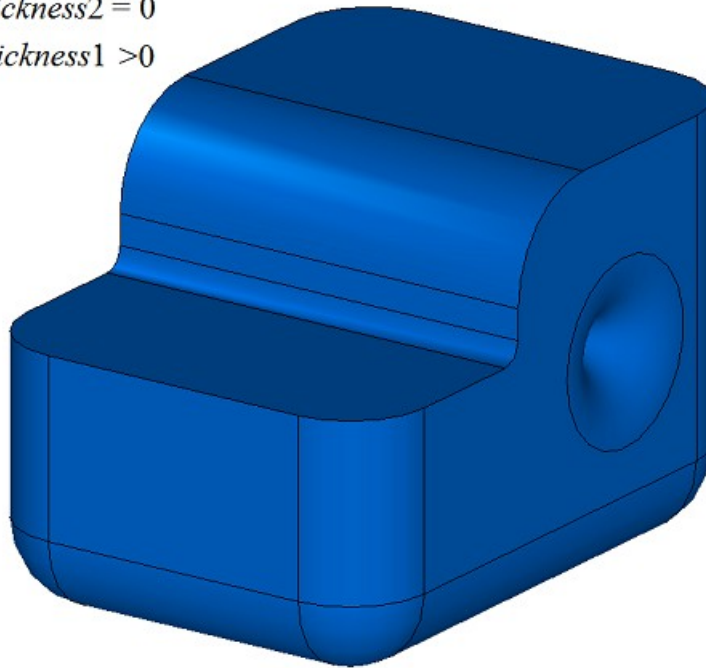


Рис. М.2.13.5.

Метод **ThinSolid** добавляет в журнал построенного тела строитель **MbShellSolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbShellSolid** объявлен в файле `cr_thin_shell_solid.h`.

Тестовое приложение `test.exe` выполняет построение тонкостенного тела командой меню «Создать->Тело->Обработкой граней->Приданием одинаковой толщины».

М.2.14. Построение тонкостенного тела с различной толщиной стенки

Метод

MbResultType

```
ThinSolid ( MbSolid & solid,
             MbeCopyMode sameShell,
             RPAArray<MbFace> & outFaces,
             RPAArray<MbFace> & offFaces,
             SArray<double> & offDistances,
             SweptValues & params,
             const MbSNameMaker & names,
             MbSolid *& result )
```

выполняет построение тонкостенного тела исключением указанных граней и приданием различной толщины оставшимся граням исходного тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **outFaces** – множество исключаемых граней,
- **offFaces** – множество граней, которым заданы индивидуальные значения толщин,
- *offDistances* – множество индивидуальных значений толщин (синхронен с **offFaces**),
- *params* – параметры построения,
- *names* – именованье построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Метод удаляет из исходного тела **solid** указанные грани **outFaces**, а оставшимся граням «придаёт заданную толщину», причём толщина для разных граней может отличаться. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**. Параметр **offFaces** содержит грани, которым заданы индивидуальные значения толщин *offDistances*. Грани **offFaces**[*i*] будет «придана толщина» *offDistances*[*i*]. Остальным граням будет придана толщина, определяемая параметром *params*. Параметр *params* содержит информацию о замкнутости построенного тела **result** и толщине стенки сохраняемых граней, не входящих в множество **offFaces**. Толщина стенки сохраняемых граней может быть равна или *params.thickness1* в положительном направлении нормали грани или *params.thickness2* в отрицательном направлении нормали грани. Параметр *names* обеспечивает именование граней построенного тела. Для выполнения операции удаляемые грани **outFaces** по общему периметру не должны иметь гладких ребер стыковки с сохраняемыми гранями исходного тела.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление `MbCopyMode` описано в параграфе [0.7.9. Копирование множества граней MbFaceShell](#).

На рис. М.2.14.1 показаны исходное тело **solid**, удаляемые грани **outFaces** и грани **offFaces**, которым заданы индивидуальные значения толщин *offDistances*.

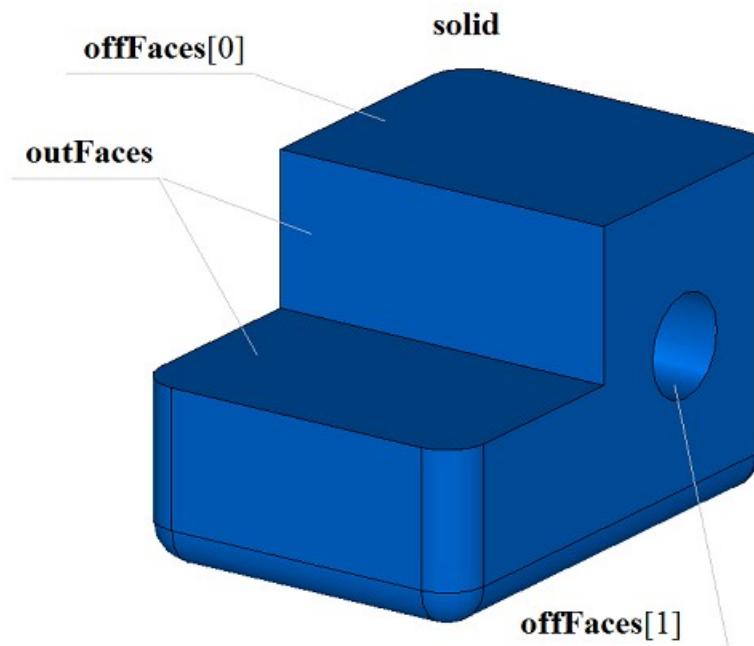


Рис. М.2.14.1.

На рис. М.2.14.2 приведено построенное тело **result** с сохраненными и вновь построенными гранями.

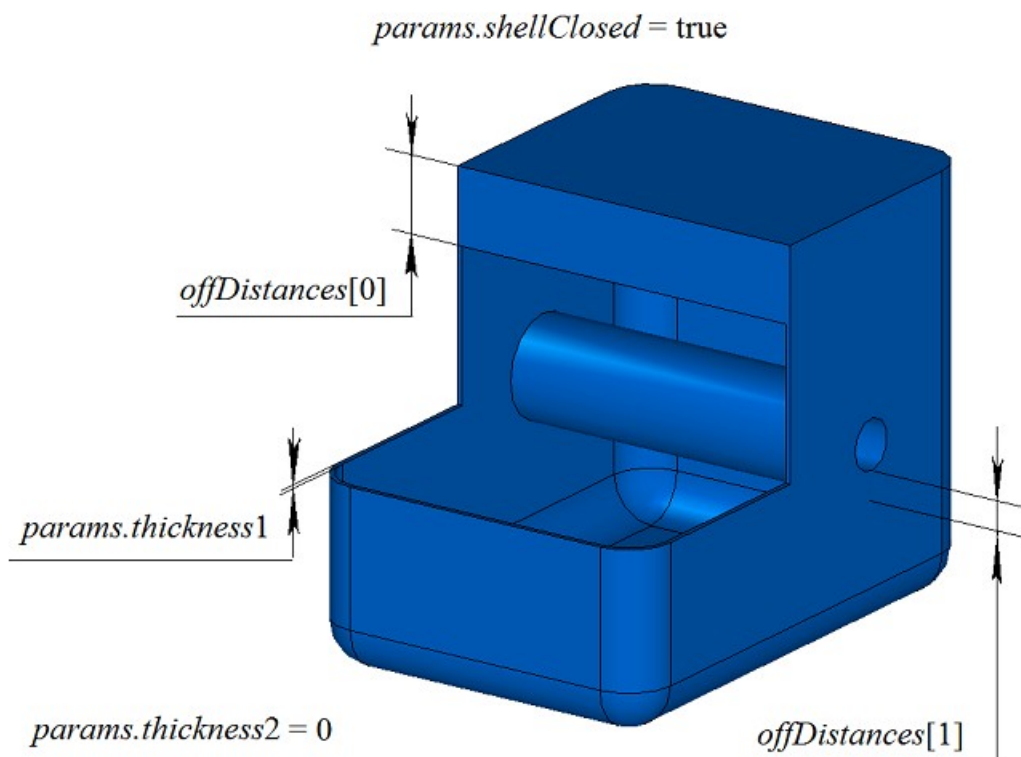


Рис. М.2.14.2.

Для выполнения операции каждая из граней **offFaces** по общему периметру не должна иметь гладких ребер стыковки с гранями исходного тела другой толщины.

Метод **ThinSolid** добавляет в журнал построенного тела строитель **MbShellSolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbShellSolid** объявлен в файле `cr_thin_shell_solid.h`.

Тестовое приложение `test.exe` выполняет построение тонкостенного тела командой меню «Создать->Тело->Обработкой граней->Приданием различной толщины».

М.2.15. Построение тела приданием толщины поверхности

Метод
MbResultType
ThinSolid (const **MbSurface** & **surface**,
 bool *faceSense*,
 SweptValues & *params*,
 const **MbSNameMaker** & *names*,
 SimpleName *name*,
MbSolid *& **result**)

выполняет построение тела путём придания толщины заданной поверхности.

Входными параметрами метода являются:

- **surface** – заданная поверхность,
- *faceSense* – ориентация нормали поверхности в грани построенного тела,
- *params* – параметры построения,
- *names* – именователь граней,
- *name* – имя операции.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_solid.h`.

Метод строит грань на основе поверхности **surface** и далее строит тело путем «придания этой грани заданной толщины». Параметр *faceSense* указывает, совпадает ли направление нормали поверхности **surface** с направлением нормали грани. Грани будет придана толщина, определяемая параметром *params*. Параметр *params* содержит информацию о толщине стенки построенного тела **result**. Толщина стенки может быть равна или *params.thickness1* в положительном направлении нормали грани или *params.thickness2* в отрицательном направлении нормали грани. Параметры *names* и *name* обеспечивают именование граней построенного тела.

На рис. М.2.15.1 показаны исходная поверхность **surface**.

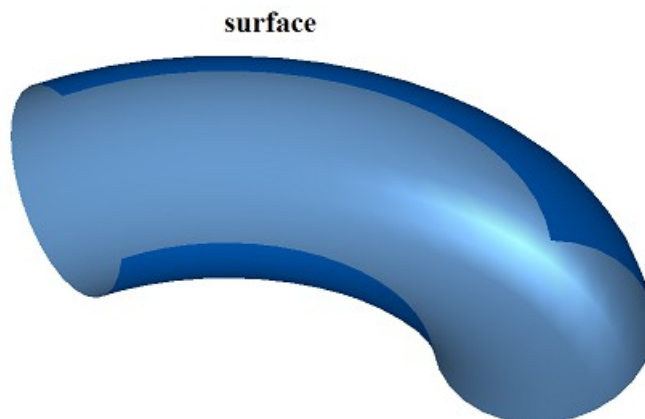


Рис. М.2.15.1.

На рис. М.2.15.2 приведено построенное тело **result**.

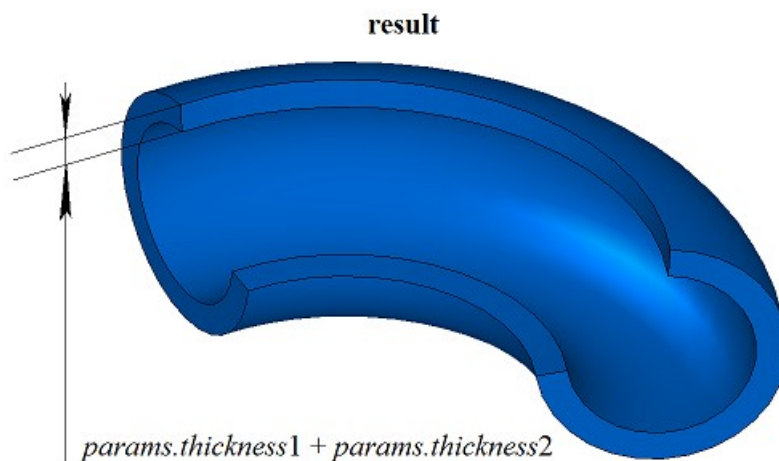


Рис. М.2.15.2.

Метод **ThinSolid** добавляет в журнал построенного тела строитель *MbShellSolid*, который содержит все необходимые данные для выполнения операции. Строитель *MbShellSolid* объявлен в файле *cg_thin_shell_solid.h*.

Тестовое приложение *test.exe* выполняет построение тонкостенного тела командой меню «Создать->Тело->На базе поверхности->Приданием толщины».

М.2.16. Построение зеркального тела

Метод
MbResultType

MirrorSolid (const MbSolid & **solid**,
const MbPlacement3D & **place**,
const MbSNameMaker & names,
MbSolid *& **result**)

выполняет построение зеркальной копии тела относительно заданной плоскости.

Входными параметрами метода являются:

- **solid** – исходное тело,
- **place** – локальная система координат, плоскость XY которой является плоскостью зеркала,
- names – именованье грани среза.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Метод создаёт зеркальную копию тела **solid** относительно плоскости XY заданной локальной системы координат **place**. Параметр names обеспечивает именованье граней построенного тела.

На рис. М.2.16.1 показаны исходное тело **solid** и плоскость симметрии **place**.

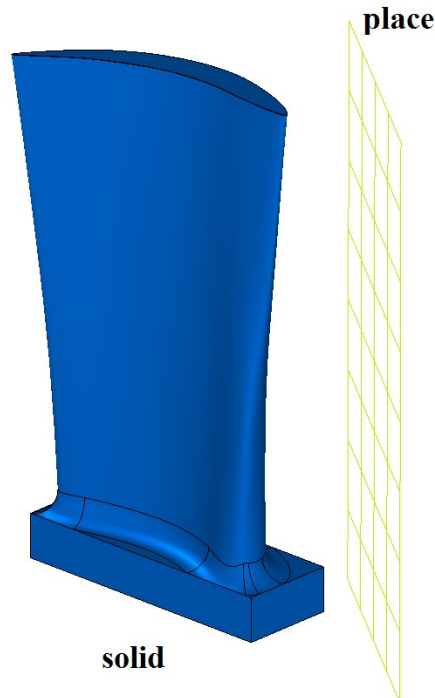


Рис. М.2.16.1.

На рис. М.2.16.2 приведено исходное тело **solid** и построенное тело **result**.

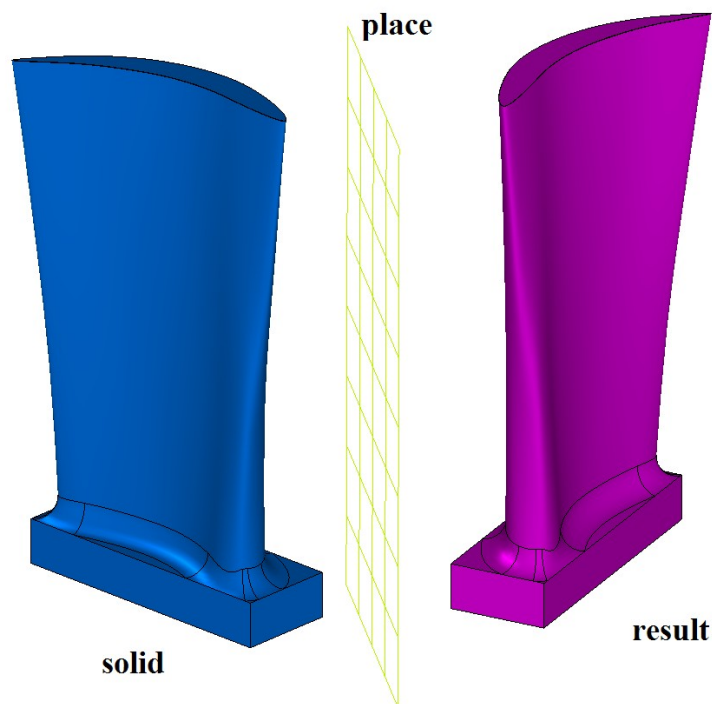


Рис. М.2.16.2.

Метод **MirrorSolid** добавляет в журнал построенного тела строитель **MbSymmetrySolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbSymmetrySolid** объявлен в файле `cr_symmetry_solid.h`.

Тестовое приложение `test.exe` выполняет построение симметричного тела командой меню «Создать->Тело->На базе тел->Симметричное».

М.2.17. Булева операция тела и множества тел

Метод

MbResultType

```
UnionResult ( MbSolid * solid,
               MbeCopyMode sameShell,
               RPAArray<MbSolid> & solids,
               MbeCopyMode sameShells,
               OperationType oType,
               bool checkIntersect,
               bool mergeFaces,
               const MbSNameMaker & names,
               bool isArray,
               MbSolid *& result,
               RPAArray<MbSolid> * notGluedSolids = NULL )
```

выполняет объединение тел заданного множества и заданную булеву операцию с исходным телом, если оно задано.

Входными параметрами метода являются:

- **solid** – исходное тело (может быть ноль),
- *sameShell* – вариант копирования исходного тела,
- **solids** – множество тел,
- *sameShells* – вариант копирования тел множества,
- *oType* – тип булевой операции: `bo_Union` – объединение тел,
`bo_Intersect` – пересечение тел,
`bo_Difference` – вычитание тел,

- *checkIntersect* – флаг проверки пересечение тел множества (false – не проверять),
- *mergeFaces* – объединять ли подобные грани,
- *names* – именователю граней,
- *isArray* – флаг регулярности множества тел.

Выходным параметром метода является построенное тело **result** и множество тел **notGluedSolids**, которые оказались не используемыми в операции (может быть ноль).

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_solid.h*.

Метод представляет собой разновидность булевой операции **BooleanSolid** и ускоряет работу, когда требуется выполнить одну и ту же булеву операцию тела **solid** с большим количеством других тел. Сначала метод выполняет объединение множества тел **solids** в общее промежуточное тело, а затем выполняет указанную булеву операцию *oType* тела **solid** с промежуточным телом. Тела **solids** могут не пересекаться друг с другом. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**. Параметр *sameShells* управляет передачей граней, ребер и вершин от множества тел **solids** построенному телу **result**. Параметры *checkIntersect* и *isArray* управляют построением общего промежуточного тела для множества тел **solids**. Параметр *mergeFaces* управляет объединением подобных граней. Параметр *names* обеспечивает именование граней построенного тела.

Параметр *sameShell* (*sameShells*) может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление *MbCopyMode* описано в параграфе [O.7.9. Копирование множества граней MbFaceShell](#).

Параметр *OperationType* *oType* определяет тип булевой операции и принимает три значения: *bo_Union*, *bo_Intersect*, *bo_Difference*. При *oType=bo_Union* рассматриваемый метод выполняет объединение тел **solid** и множества тел **solids**, при *oType=bo_Intersect* рассматриваемый метод выполняет пересечение тел **solid** и множества тел **solids**, при *oType=bo_Difference* рассматриваемый метод выполняет вычитание из тела **solid** множества тел **solids**.

Параметры *checkIntersect* и *isArray* служат для ускорения работы метода **UnionResult**.

Параметр *checkIntersect* дает команду на проверку пересечения тел множества **solids** между собой. Если параметр *checkIntersect==true*, то при построении общего промежуточного тела выполняется булева операция объединения всех пересекающихся тел множества **solids** в одно тело. В противном случае объединение тел заданного множества выполняется простым переключиванием граней всех тел в одно новое тело. Вне зависимости от значения параметра *checkIntersect* все не пересекающиеся тела множества **solids** передают свои грани общему промежуточному телу.

Параметр *mergeFaces* позволяет объединять подобные грани в построенном теле **result** или оставлять их разделенными. Работа параметра *mergeFaces* приведена на рис. М.2.17.2 и М.2.17.3. При *mergeFaces==false* подобные грани не объединяются.

Параметр *isArray* работает только при *checkIntersect==true* и сообщает о регулярности множества тел **solids**. Если *isArray==true*, то тела множества расположены в узлах прямоугольной или круговой сетки и позиции тел заданы в именах граней.

Параметр **notGluedSolids** содержит тела, которые оказались не используемыми в операции из-за невозможности объединения их с общим промежуточным телом.

На рис. М.2.17.1 приведены исходное тело **solid** и множество тел **solids**.

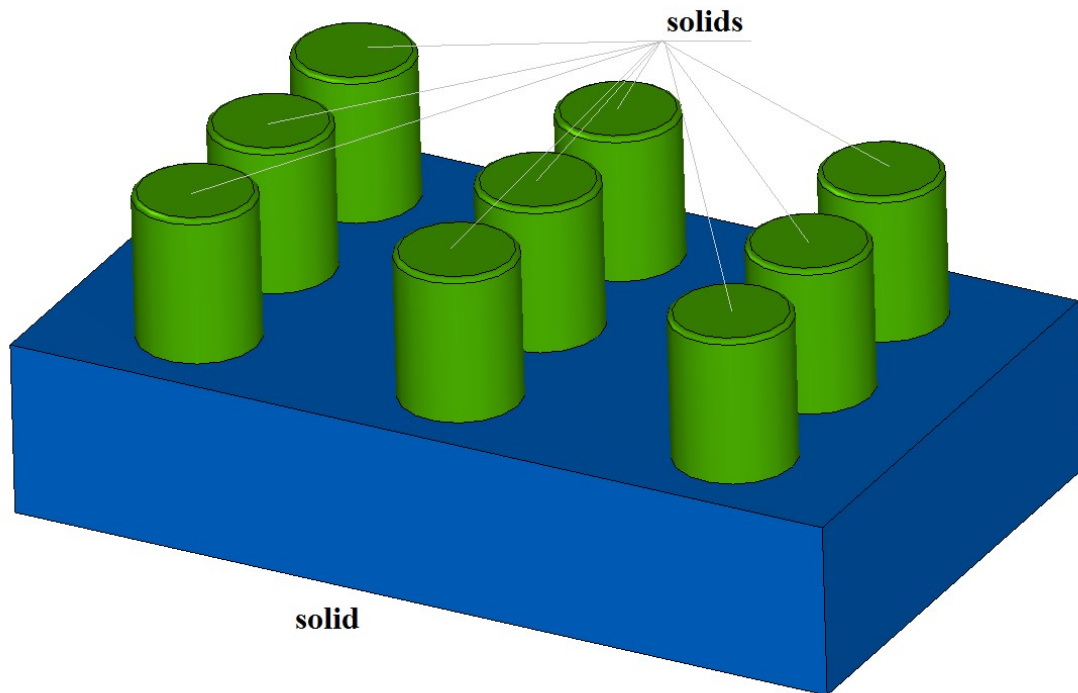


Рис. М.2.17.1.

На рис. М.2.17.2 приведен результат **result** приклеивания тел **solids** к телу **solid**. В данном случае параметр *checkIntersect* можно положить равным `false`, так как тела **solids** не пересекаются друг с другом.

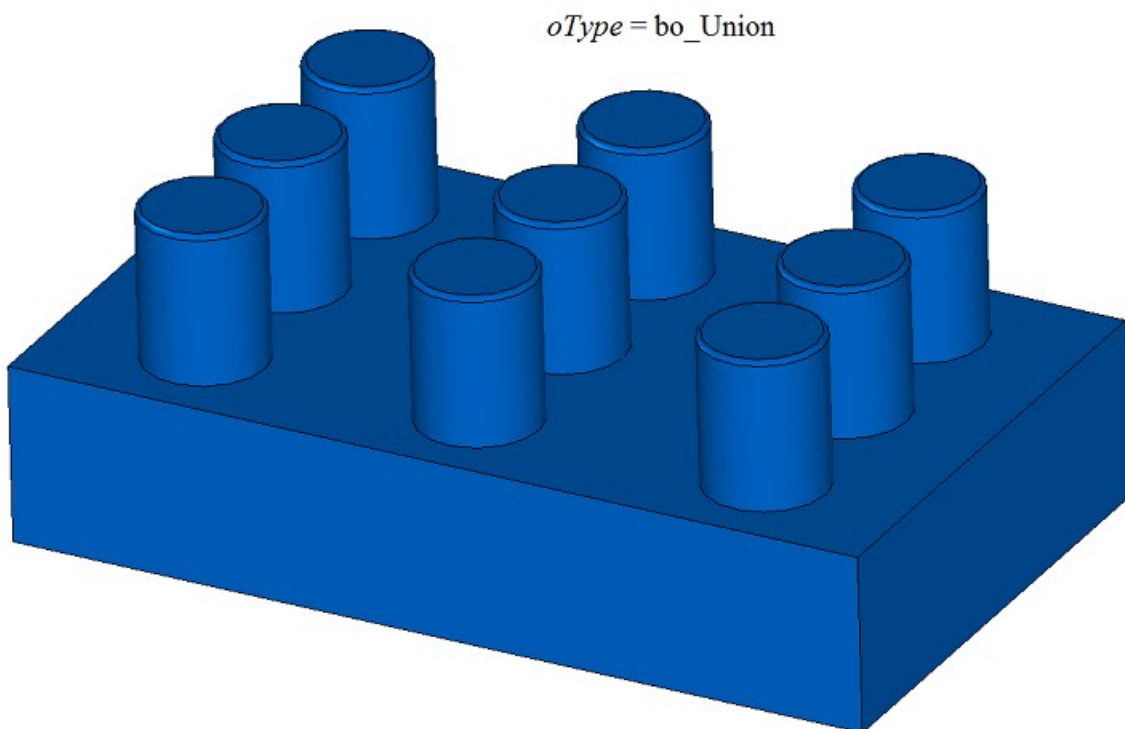


Рис. М.2.17.2.

На рис. М.2.17.3 приведены исходное тело **solid** и множество тел **solids**.

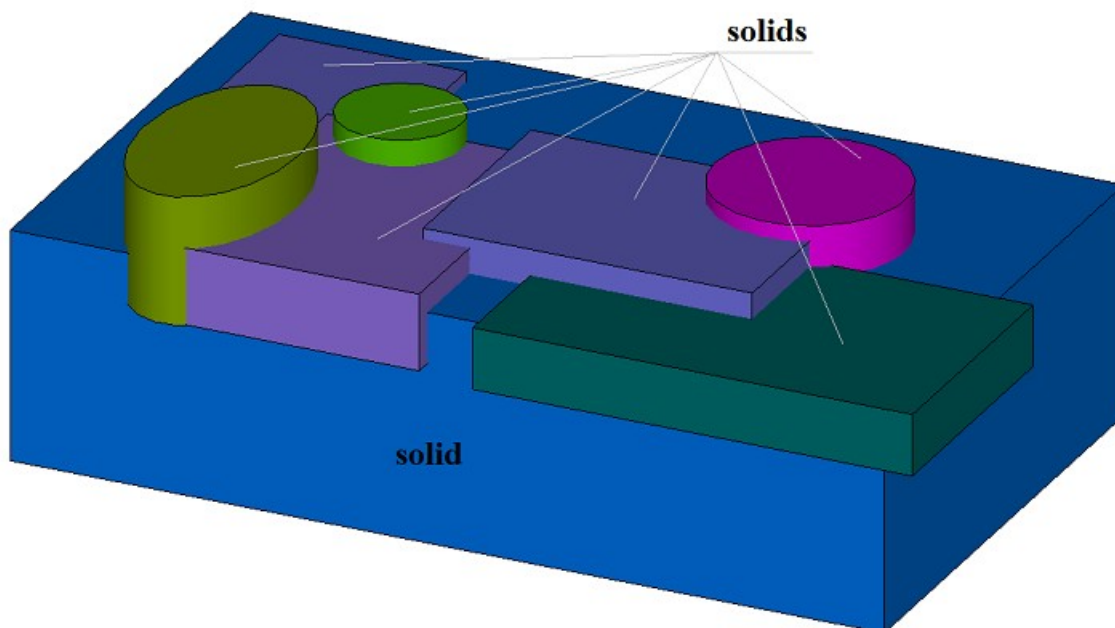


Рис. М.2.17.3.

На рис. М.2.17.4 приведен результат **result** вычитания тел **solids** из тела **solid** при работе рассматриваемого метода с параметром *mergeFaces*==true. В данном случае параметр *checkIntersect* необходимо положить равным true, так как тела **solids** пересекаются друг с другом.

oType = bo_Difference *mergeFaces* = true

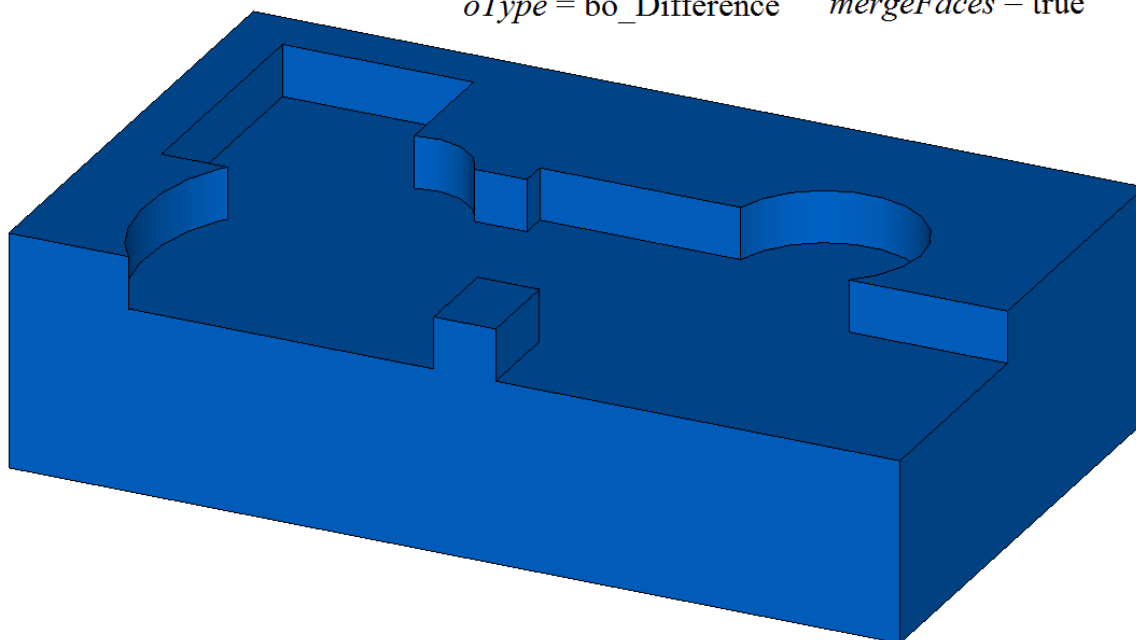


Рис. М.2.17.4.

На рис. М.2.17.5 приведен результат **result** вычитания тел **solids** из тела **solid** при работе рассматриваемого метода с параметром *mergeFaces*==false.

oType = bo Difference mergeFaces = false

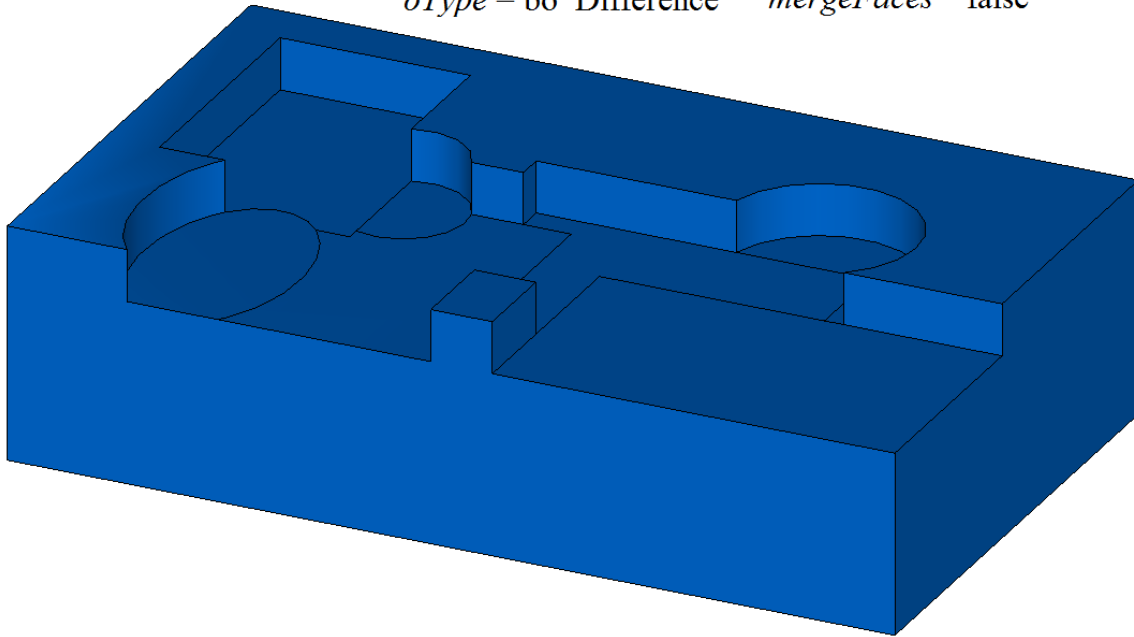


Рис. М.2.17.5.

На рис. М.2.17.6 для наглядности грани результата вычитания тел **solids** из тела **solid**, приведенного на рис. М.2.17.5, раскрашены в цвета исходных тел. По форме граней можно определить последовательность включения тел **solids** в промежуточное тело: тело оставляет более полный отпечаток, если оно включено в промежуточное тело раньше остальных.

oType = bo Difference mergeFaces = false

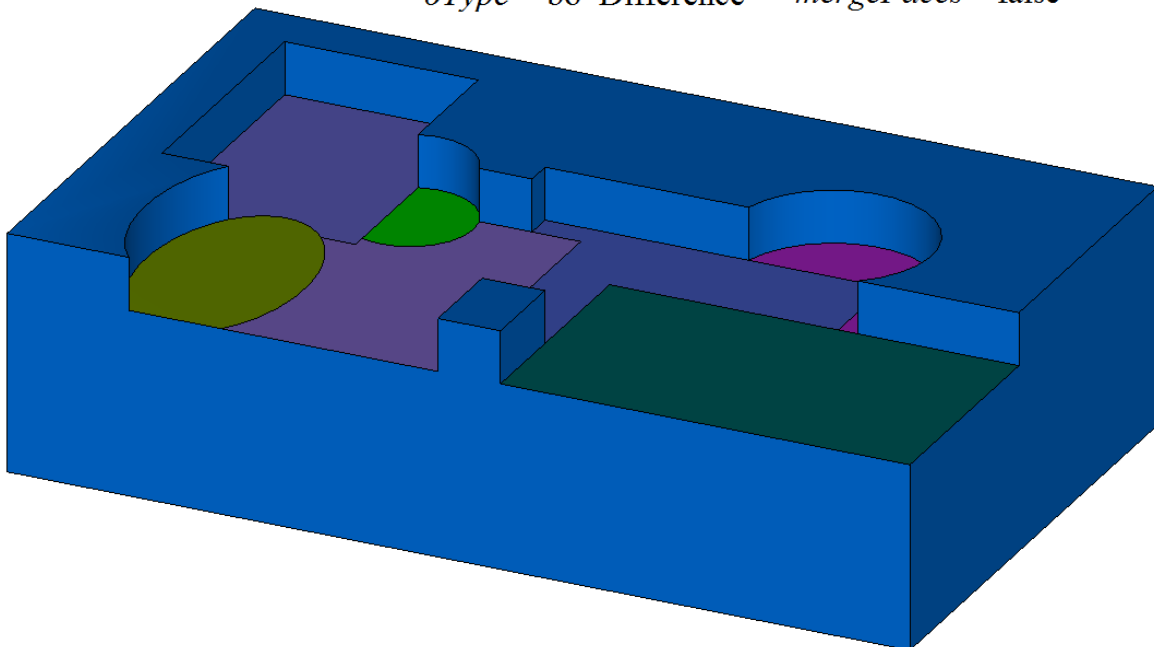


Рис. М.2.17.6.

Метод **UnionResult** добавляет в журнал построенного тела строитель MbUnionSolid, который содержит все необходимые данные для выполнения операции. Строитель MbUnionSolid объявлен в файле `cg_union_solid.h`.

Тестовое приложение test.exe выполняет булеву операцию тела с множеством тел командами меню «Создать->Тело->Приклеиванием к телу->Набора тел», «Создать->Тело->Вырезанием из тела->Набора тел», «Создать->Тело->Пересечением с телом->Набора тел».

M.2.18. Объединение множества тел

Метод

MbResultType

```
UnionSolid ( RPAArray<MbSolid> & solids,  
             MbeCopyMode sameShells,  
             bool checkIntersect,  
             const MbSNameMaker & names,  
             bool isArray,  
             MbSolid*& result,  
             RPAArray<MbSolid> * notGluedSolids = NULL );
```

выполняет объединение тел заданного множества.

Входными параметрами метода являются:

- **solids** – множество тел,
- *sameShells* – вариант копирования тел множества,
- *checkIntersect* – флаг проверки пересечение тел множества (false – не проверять),
- *names* – именовател граней,
- *isArray* – флаг регулярности множества тел,

Выходным параметром метода является построенное тело **result** и множество тел **notGluedSolids**, которые оказались не используемыми в операции (может быть ноль).

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_solid.h`.

Рассматриваемый метод работает так же, как метод **UnionResult** при значениях параметров последнего **solid=0**, *sameShell=cm_Same*, *oType=bo_Base*, *mergeFaces=true*. Метод ускоряет работу, когда требуется объединить большое количество тел вместе. Метод выполняет объединение множества тел **solids** в одно тело **result**, при этом тела могут не пересекаться друг с другом. Параметр *sameShells* управляет передачей граней, ребер и вершин от множества тел **solids** построенному телу **result**. Параметры *checkIntersect* и *isArray* управляют построением общего промежуточного тела для множества тел **solids**. Параметр *names* обеспечивает именование граней построенного тела.

Параметр *sameShells* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbeCopyMode описано в параграфе [O.7.9. Копирование множества граней MbFaceShell](#)

Параметры *checkIntersect* и *isArray* служат для ускорения работы метода **UnionResult**.

Параметр *checkIntersect* дает команду на проверку пересечения тел множества **solids** между собой. Если параметр *checkIntersect==true*, то при построении результата выполняется булева операция объединения всех пересекающихся тел множества **solids**. В противном случае объединение тел заданного множества выполняется простым перекладыванием граней всех тел в построенное тело. Вне зависимости от значения параметра *checkIntersect* все не пересекающиеся тела множества **solids** передают свои грани общему промежуточному телу.

Параметр *isArray* работает только при *checkIntersect==true* и сообщает о регулярности множества тел **solids**. Если *isArray==true*, то тела множества расположены в узлах прямоугольной или круговой сетки и позиции тел заданы в именах граней.

Параметр **notGluedSolids** содержит тела, которые оказались не используемыми в операции из-за невозможности их объединения.

На рис. M.2.18.1 приведены исходные тела **solids**.

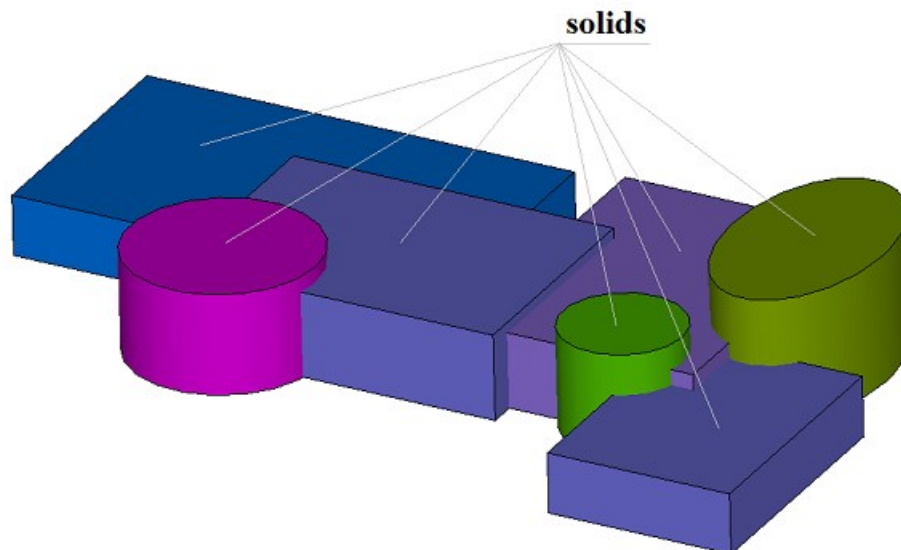


Рис. М.2.18.1.

На рис. М.2.18.2 приведен результат **result** объединения тел **solids**. В данном случае параметр *checkIntersect* необходимо положить равным true, так как тела **solids** пересекаются друг с другом.

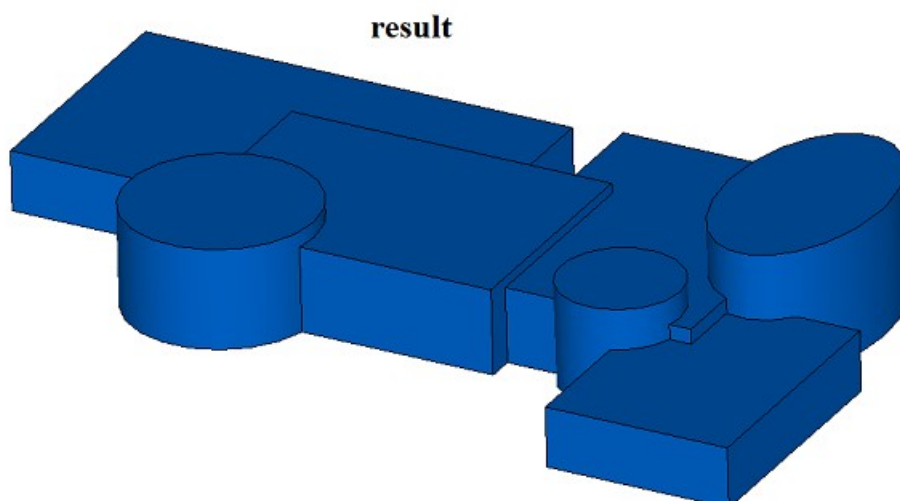


Рис. М.2.18.2.

Метод
 MbResultType
UnionSolid (const RPAArray<MbSolid> & solids,
 const MbSNameMaker & names,
 MbSolid *& result)

является упрощенным вариантом рассматриваемого одноименного метода и совпадает с ним при параметрах *sameShells=cm_Same*, *checkIntersect==false*, *isArray==false*, **notGluedSolids==NULL**. Последний метод не выполняет никаких проверок и построений, а просто собирает в построенном теле **result** все грани тел **solids**. Таким образом, построенное тело и исходные тела имеют одни и те же грани.

Методы **UnionSolid** добавляют в журнал построенного тела строитель MbUnionSolid, который содержит все необходимые данные для выполнения операции. Строитель MbUnionSolid объявлен в файле *cr_union_solid.h*.

Тестовое приложение test.exe выполняет булеву операцию тела с множеством тел командой меню «Создать->Тело->На базе тела ->Набор тел».

М.2.19. Разделить тело на несвязанные части

Метод
unsigned int
DetachParts (MbSolid & **solid**,
RPAarray<MbSolid> & **parts**,
bool *sort*,
const MbSNameMaker & names)

отделяет от тела не связанные части.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sort* – флаг сортировки отделённых частей по убыванию габарита,
- *names* – именователь граней.

Выходными параметрами метода являются исходное тело **solid** и множество отделённых частей исходного тела **parts**.

Метод возвращает количество отделённых частей. Метод объявлен в файле action_solid.h.

После вычитания тела **solid2** из тела **solid1**, приведенных на рис. М.2.19.1, результат булевой операции **solid** будет состоять из нескольких топологически не связанных частей, рис. М.2.19.2, хотя будет вести себя как единый объект. Рассматриваемый метод позволяет разбить тело **solid**, состоящее из нескольких топологически не связанных частей, на отдельные тела. Одна часть остаётся в исходном теле **solid**, а остальные части будут сложены в присланный контейнер тел **parts**.

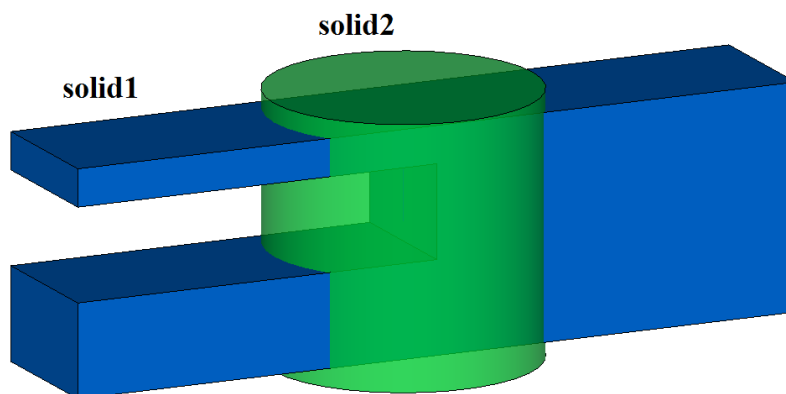


Рис. М.2.19.1.

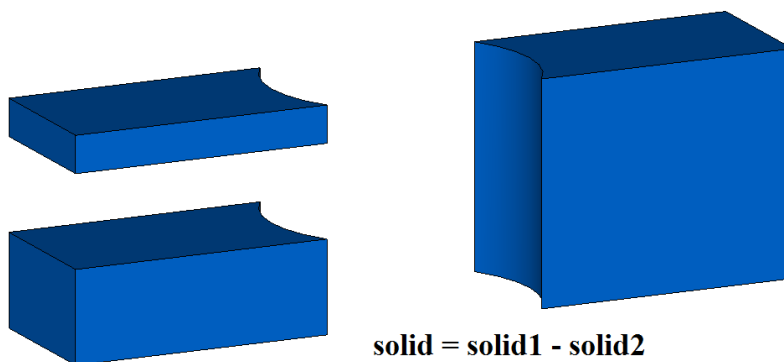


Рис. М.2.19.2.

Если флаг сортировки *sort*==true, то в исходном теле останется часть с наибольшим габаритом, а отделённые части будут сортированы по убыванию габарита, рис. М.2.19.3. В противном случае в

исходном теле останется часть, топологически связанная с первой гранью, а отделённые части будут сортированы по номеру начальной грани в исходном теле.

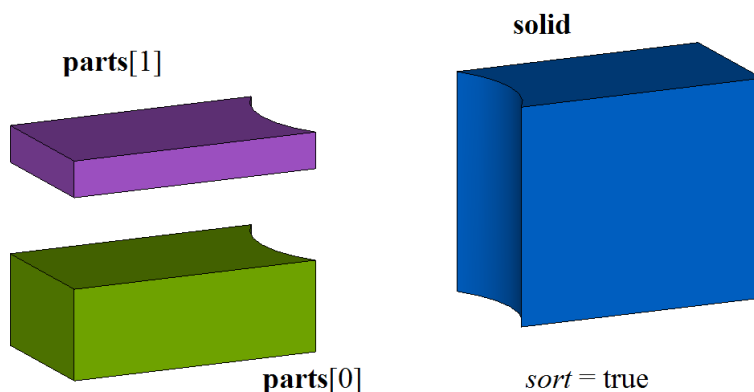


Рис. М.2.19.3.

Параметр `names` обеспечивает именование граней построенного тела и версионирование операции.

Метод

unsigned int

CreateParts (const [MbSolid](#) & **solid**,
RArray<[MbSolid](#)> & **parts**,
const MbSNameMaker & names)

выполняет те же действия, что и рассмотренный выше метод, с той разницей, что он не меняет исходное тело **solid**, а все топологически не связанные части исходного тела добавляет в тела **parts**, рис. М.2.19.4.

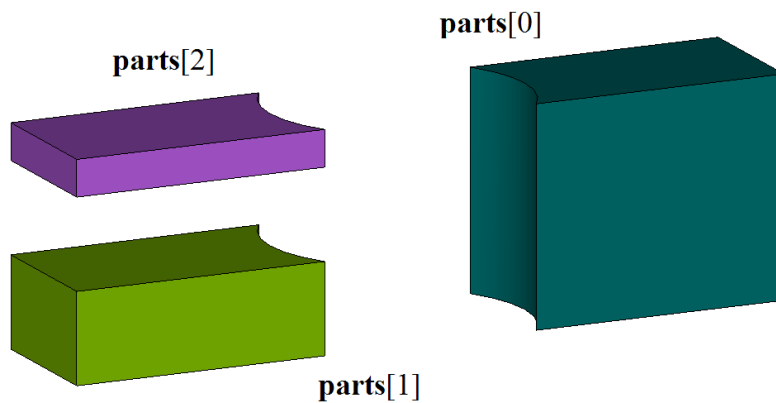


Рис. М.2.19.4.

Тела **parts** будут построены на тех же гранях, что и исходное тело **solid**.

Методы **DetachParts** и **CreateParts** добавляют в журнал построенного тела строители `MbDetachSolid`, которые содержат все необходимые данные для выполнения операции. Строитель `MbDetachSolid` объявлен в файле `sr_detach_solid.h`.

Тестовое приложение `test.exe` выполняет булеву операцию тела с множеством тел командой меню «Модифицировать->Тело или оболочку->Разделить части».

М.2.20. Отделение несвязанной части тела

Метод

MbResultType

ShellPart (const [MbSolid](#) & **solid**,

```

size_t id,
const MbPath & path,
const MbSNameMaker & names,
MbPartSolidIndices & partIndices,
MbSolid * & result )

```

выделяет в отдельное тело указанную часть распадающегося на части исходного тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *id* – номер выбранной части исходного тела,
- *path* – идентификатор выбранной части исходного тела в модели,
- *names* – именователь граней.
- *partIndices* – индексы частей тела.

Выходными параметрами метода являются построенное тело **result** и индексы частей тела *partIndices*.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_solid.h*.

Метод создает тело из указанной части исходного тела. Исходное тело должно состоять из отдельных частей. На рис. М.2.20.1 приведен результат булевой операции вычитания тел, показанных на М.2.19.1, который состоит из нескольких топологически не связанных частей. Рассматриваемый метод позволяет создать тело, сохраняющее только одну часть из нескольких топологически не связанных частей исходного тела.

Параметр *id* указывает номер части исходного тела **solid**. Параметр *path* содержит путь к части тела. В простом случае путь к части тела содержит номер части исходного тела *id*.

На рис. М.2.20.1 приведено исходное тело, состоящее из нескольких топологически не связанных частей.

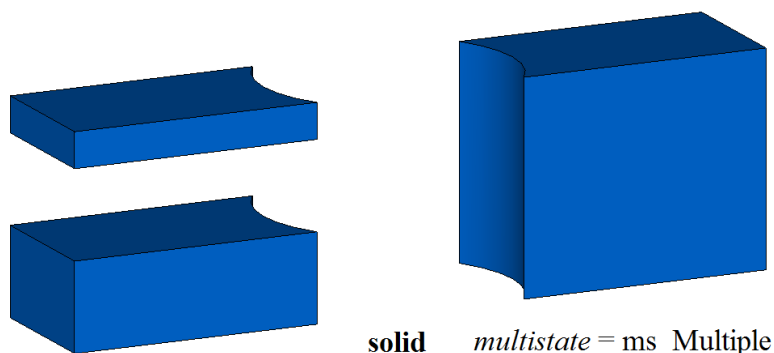


Рис. М.2.20.1.

На рис. М.2.20.2 приведено построенное тело, состоящее из одной выбранной части исходного тела.

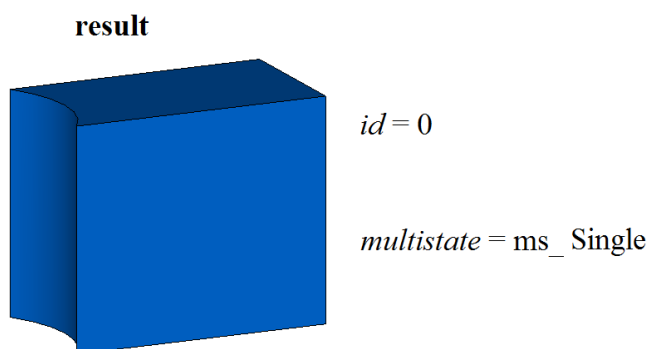


Рис. М.2.20.2.

Тело **result** будет построено на тех же гранях, что и исходное тело **solid**.

Метод **ShellPart** добавляет в журнал построенного тела строитель MbDetachSolid, который содержит все необходимые данные для выполнения операции. Строитель MbDetachSolid объявлен в файле `sr_detach_solid.h`.

Тестовое приложение `test.exe` выполняет булеву операцию тела с множеством тел командой меню «Создать->Тело->На базе тела ->Часть набора тел».

М.2.21. Разбиение граней тела

Метод
MbResultType
SplitSolid ([MbSolid](#) & **solid**,
MbeCopyMode *sameShell*,
const RPAArray<[MbSpaceItem](#)> & **items**,
bool *same*,
RPAArray<[MbFace](#)> & **faces**,
const MbSNameMaker & names,
[MbSolid](#) *& **result**)

выполняет разбиение указанных граней тела пространственными кривыми, поверхностями и оболочками.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **items** – пространственные элементы разбиения граней,
- *same* – использовать оригинальные пространственные элементы (*true*) или их копии (*false*),
- **faces** – множество разбиваемых граней,
- names – именованье построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType. Метод объявлен в файле `action_solid.h`.

Метод разбивает указанные грани **faces** исходного тела **solid** пространственными объектами **items**, если указанные грани пересекаются с объектами **items**. Объектами **items** могут служить или кривые, или поверхности, или тело. Для выполнения операции режущие объекты должны полностью пересекать указанные грани исходного тела. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**. Параметр *same* управляет копированием режущих объектов. Параметр names обеспечивает именование граней построенного тела.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbeCopyMode описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#). На рис. М.2.21.1 показаны исходное тело **solid**, разрезаемые грани **faces** и режущая поверхность **items[0]**.

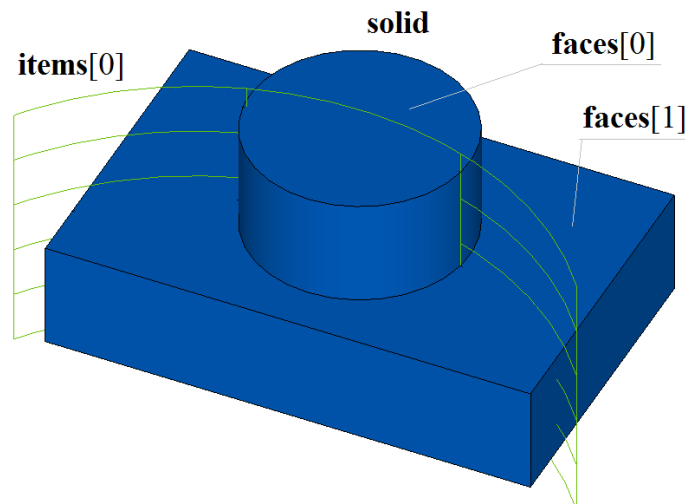


Рис. М.2.21.1.

На рис. М.2.21.2 приведено построенное тело **result** с разбитыми на части указанными гранями. Новые ребра на запрос **IsSplit** возвращают true. На рис. М.2.21.3 разбитые грани построенного тела раскрашены в разные цвета.

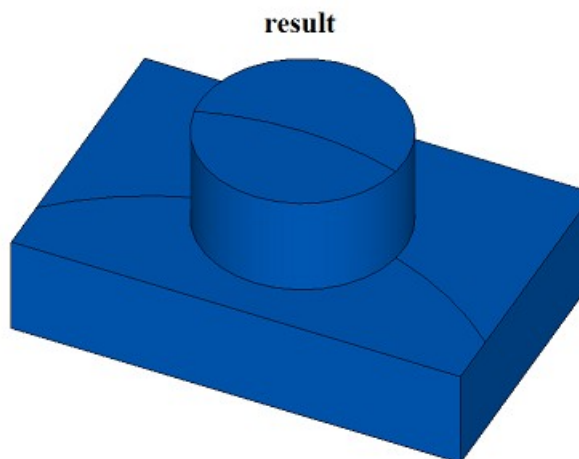


Рис. М.2.21.2.

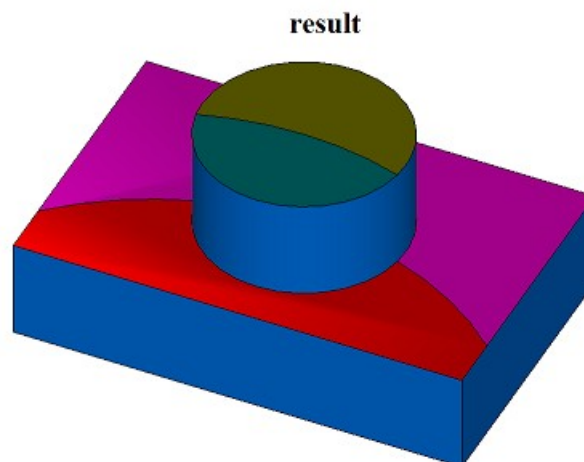


Рис. М.2.21.3.

Метод
 MbResultType
SplitSolid ([MbSolid](#) & **solid**,
 MbeCopyMode *sameShell*,
 const [MbPlacement3D](#) & **place**,
 MbeSenseValue *type*,
 const RPAArray<[MbContour](#)> & **contours**,
 bool *same*,
 RPAArray<[MbFace](#)> & **faces**,
 const MbSNameMaker & **names**,
[MbSolid](#) *& **result**)

выполняет те же действия, что и выше рассмотренный метод с той разницей, что вместо разбивающих объектов **items** разбиение граней тела **solid** выполняют поверхности, построенные выдавливанием двумерных контуров **contours**, расположенных в плоскости XY локальной системы координат **place**. Выдавливание контуров выполняется в направлении оси **place.axisZ** локальной системы координат контуров на длину, обеспечивающую полное пересечение исходного тела.

Методы **SplitSolid** добавляют в журнал построенного тела строитель MbSplitShell, который содержит все необходимые данные для выполнения операции. Строитель MbSplitShell объявлен в файле `cg_split_shell.h`.

Тестовое приложение `test.exe` выполняет разбиение указанных граней тела командой меню «Создать->Тело->Обработкой граней->Разбивкой грани».

М.2.22. Построение отверстия, кармана или паза в теле

Метод
 MbResultType
HoleSolid ([MbSolid](#) * **solid**,
 MbeCopyMode *sameShell*,
 const [MbPlacement3D](#) & **place**,
 const HoleValues & *parameters*,
 const MbSNameMaker & **names**,
[MbSolid](#) *& **result**)

выполняет построение отверстия, кармана или фигурного паза в теле.

Входными параметрами метода являются:

- **solid** – исходное тело (может быть ноль),
- *sameShell* – вариант копирования тела,
- **place** – локальная система координат, позиционирующая режущий инструмент,
- *parameters* – параметры построения,
- *names* – именователъ граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_solid.h`.

Метод строит вспомогательное тело в форме удаляемого объема для отверстия, кармана или паза. Если исходное тело **solid** задано, метод возвращает разность исходного тела и вспомогательного тела. Если исходное тело не задано (**solid**==0), то метод возвращает вспомогательное тело. Для выполнения операции вспомогательное тело должно пересекать исходное тело. Параметр *parameters* задает форму отверстия, кармана или паза. Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**. Параметр *names* обеспечивает именование граней построенного тела.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление `MbCopyMode` описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#)

Построения выполняются в локальной системе координат **place** с учетом углов поворота *parameters.placeAngle* и *parameters.azimuthAngle*. Параметр *parameters.placeAngle* определяет угол поворота локальной системы координат **place** относительно оси **place.axisY**. Параметр *parameters.azimuthAngle* определяет угол поворота локальной системы координат **place** относительно оси **place.axisZ**. Поверхность *parameters.surface* в может быть не задана. Если *parameters.surface* не равна нулю, то она служит для правильной обработки входа в отверстие, карман или паз.

На рис. М.2.22.1 приведены данные, используемые при построении, и схема наследования параметров построения от абстрактного класса `HoleValues`.

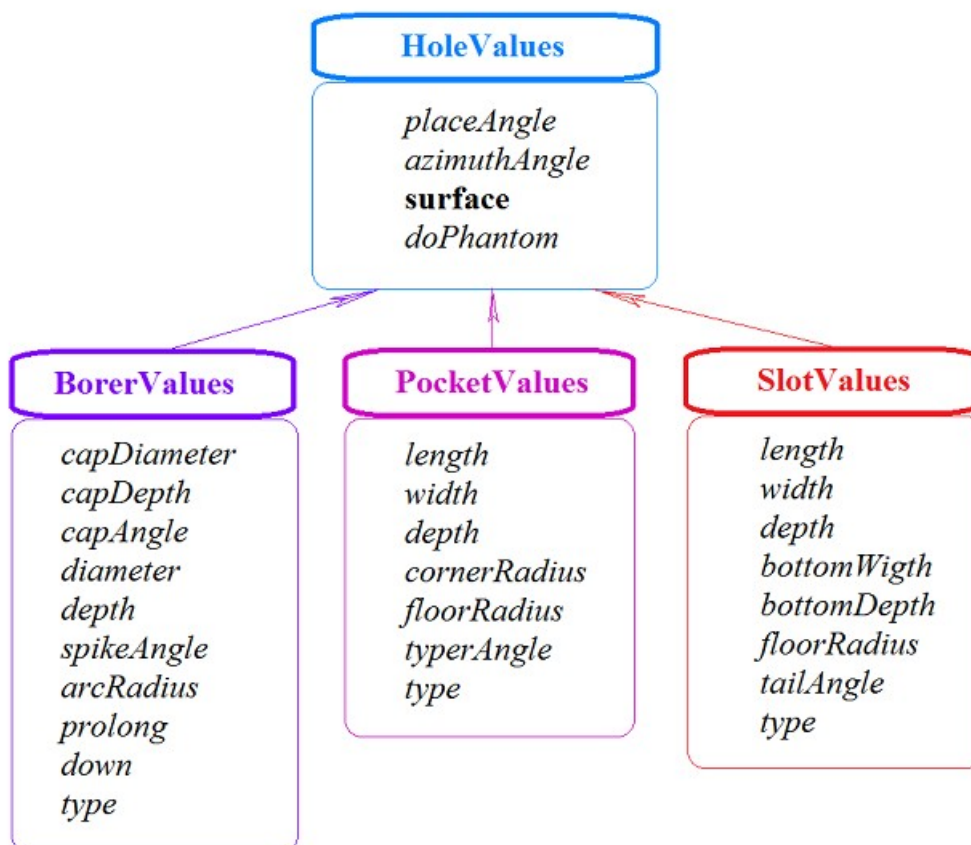
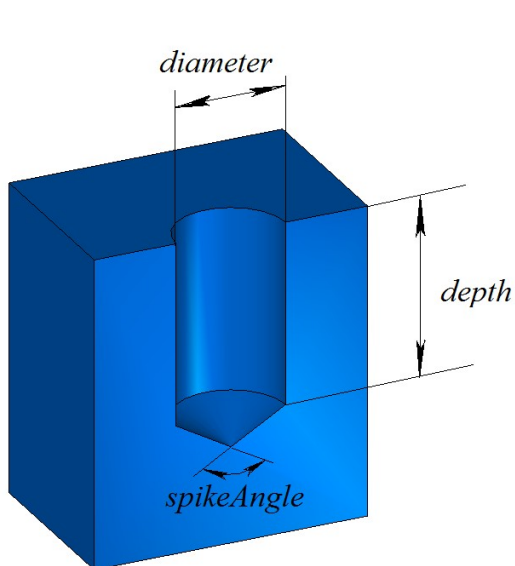


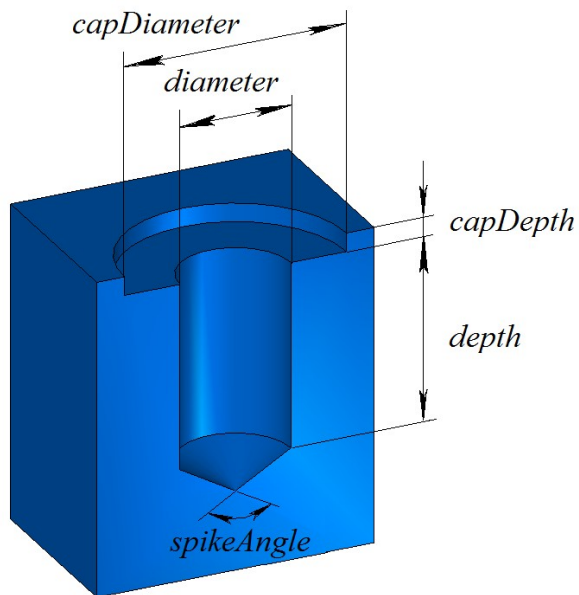
Рис. М.2.22.1.

Для построения отверстия следует использовать параметры `BorerValues`. Поддерживаются шесть типов отверстий, которые определяются параметром `BorerValues::type`, принимающим значения: `bt_SimpleCylinder`, `bt_TwofoldCylinder`, `bt_ChamferCylinder`, `bt_ComplexCylinder`, `bt_SimpleCone`, `bt_ArcCylinder`. На рис. М.2.22.2, М.2.22.3, М.2.22.4, М.2.22.5, М.2.22.6, М.2.22.7 приведены тела с отверстиями различной формы.



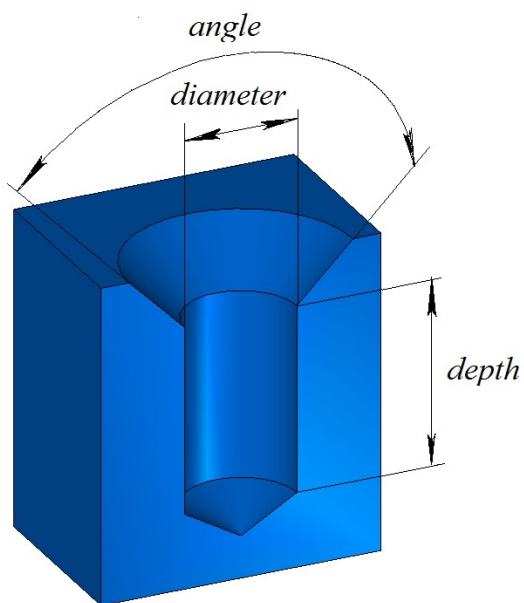
`BorerValues::type = bt_SimpleCylinder`

Рис. М.2.22.2.



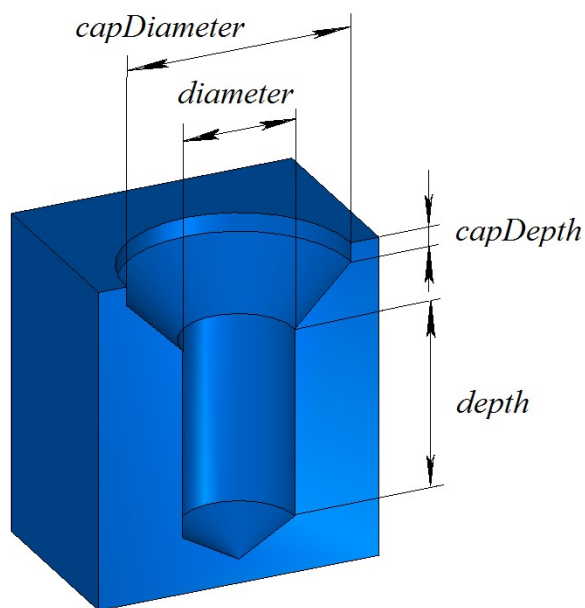
`BorerValues::type = bt_TwofoldCylinder`

Рис. М.2.22.3.



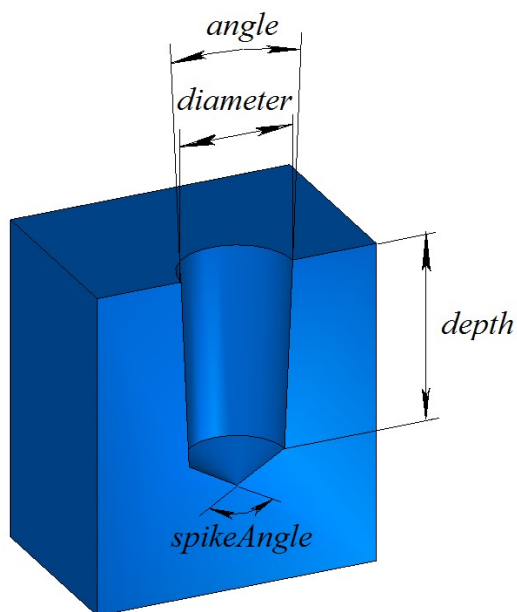
`BorerValues::type = bt_ChamferCylinder`

Рис. М.2.22.4.



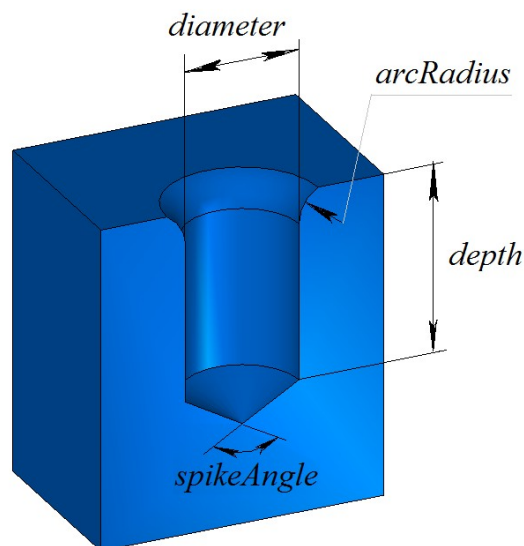
`BorerValues::type = bt_ComplexCylinder`

Рис. М.2.22.5.



BorerValues::type = bt_SimpleCone

Рис. М.2.22.6.



BorerValues::type = bt_ArcCylinder

Рис. М.2.22.7.

Для построения кармана или бобышки следует использовать параметры PocketValues. При PocketValues::type=false по заданным параметрам *parameters* строится карман, при PocketValues::type=true по заданным параметрам *parameters* строится бобышка. На рис. М.2.22.8 приведено тело с карманом прямоугольной формы без уклона боковых граней.

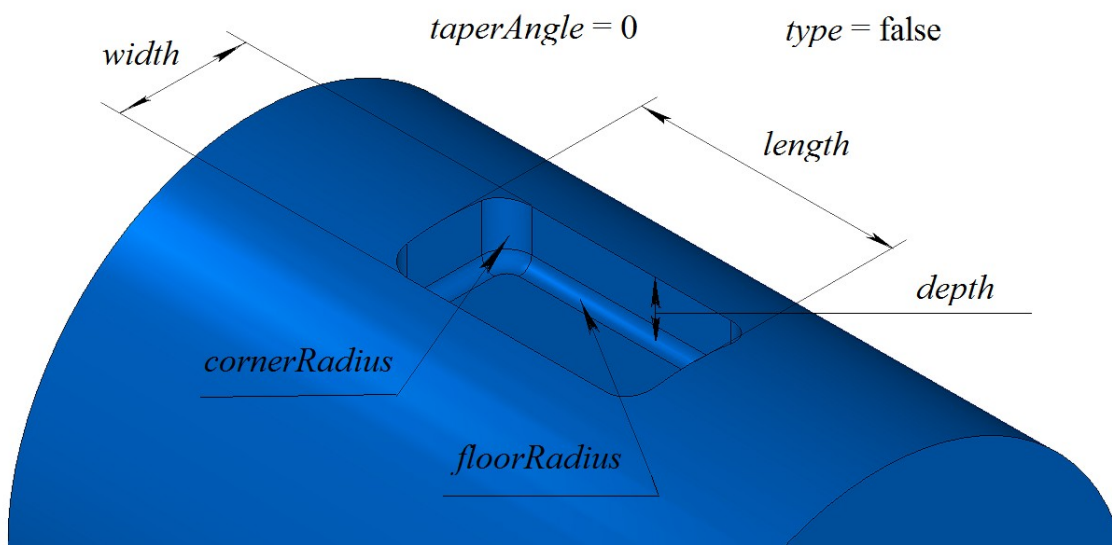


Рис. М.2.22.8.

Для построения паза следует использовать параметры SlotValues. Поддерживаются xtnsht типа пазов, которые определяются параметром SlotValues::type, принимающим значения: st_BallEnd, st_Rectangular, st_TShaped, st_DoveTail.

Метод **HoleSolid** добавляет в журнал построенного тела строитель MbHoleSolid, который содержит все необходимые данные для выполнения операции. Строитель MbHoleSolid объявлен в файле cr_hole_solid.h.

Тестовое приложение test.exe выполняет разбиение указанных граней тела командой меню «Создать->Тело->На базе тела->С отверстием».

М.2.23. Построение тела с ребром жёсткости

Метод

MbResultType

```
RibSolid ( MbSolid & solid,  
           MbeCopyMode sameShell,  
           const MbPlacement3D & place,  
           const MbContour & contour,  
           size_t index,  
           RibValues & params,  
           const MbSNameMaker & names,  
           MbSolid *& result )
```

выполняет построение тела с ребром жёсткости.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **place** – локальная система координат, плоскость XY которой является плоскостью симметрии,
- **contour** – формообразующий контур на плоскости XY локальной системы координат,
- *index* – номер сегмента в контуре,
- *params* – параметры ребра жёсткости,
- *names* – именованная граней ребра жесткости.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле *action_solid.h*.

Метод строит ребро жёсткости по заданному контуру **contour** и объединяет его с исходным телом **solid**. Сегмент контура с указанным номером устанавливает вектор уклона.

Параметр *params* задает данные для построения, см. рис. М.2.23.1.

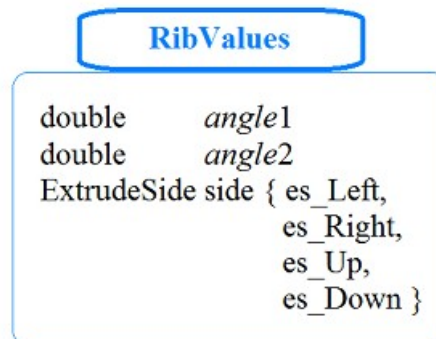


Рис. М.2.23.1.

Структура RibValues определена в файле *swept_parameter.h*.

Параметр *sameShell* управляет передачей граней, ребер и вершин от исходного тела **solid** построенному телу **result**. Параметр *names* обеспечивает именование граней построенного тела.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbeCopyMode описано в параграфе [О.7.9. Копирование множества граней MbFaceShell](#).

На рис. М.2.23.2 показаны исходное тело **solid**, локальная система координат **place**, в плоскости XY которой расположен контур **contour**.

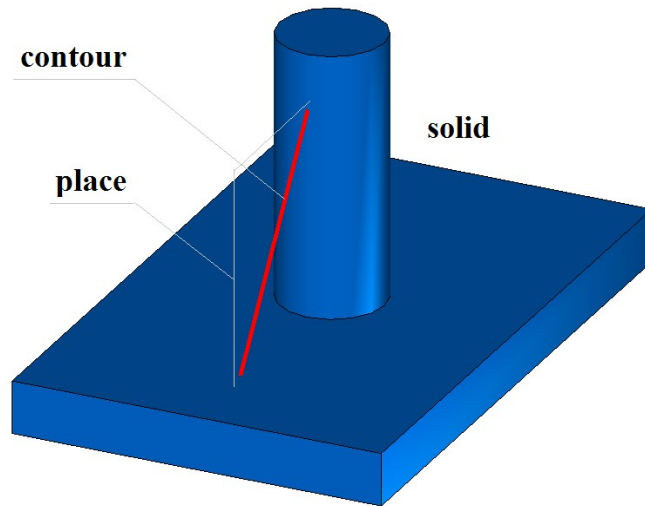


Рис. М.2.23.2.

На рис. М.2.23.3 приведен результат построения ребра жесткости без уклона боковых граней ребра жесткости.

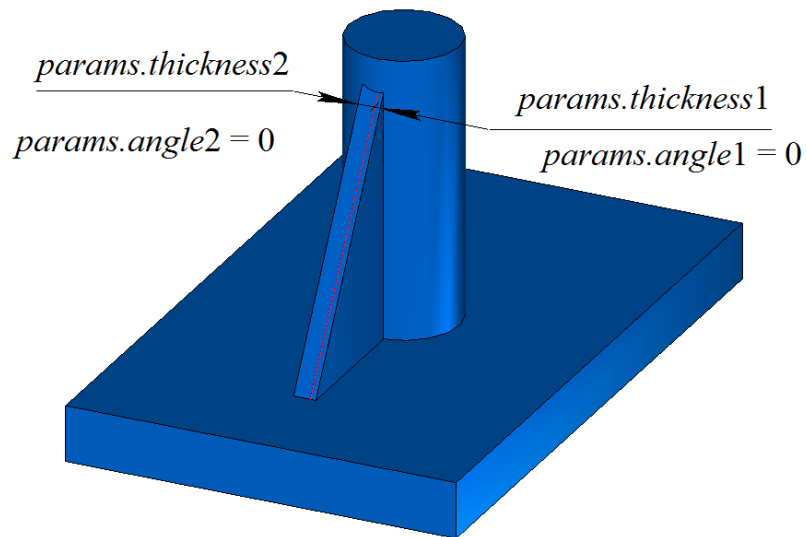


Рис. М.2.23.3.

На рис. М.2.23.4 приведен результат построения ребра жесткости с уклоном боковых граней ребра жесткости.

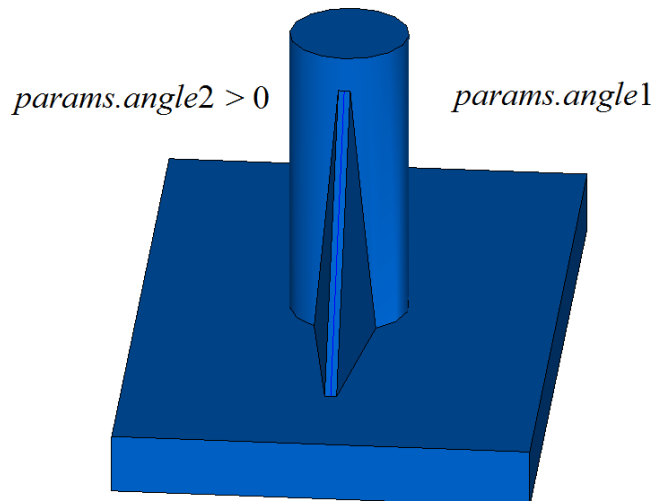


Рис. М.2.23.4.

Метод **RibSolid** добавляет в журнал построенного тела строитель MbRibSolid, который содержит все необходимые данные для выполнения операции. Строитель MbRibSolid объявлен в файле cr_rib_solid.h.

Тестовое приложение test.exe выполняет разбиение указанных граней тела командой меню «Создать->Тело->На базе тела->С ребром жесткости».

М.2.24. Уклонение граней тела

Метод
 MbResultType
DraftSolid (**MbSolid** & **solid**,
 MbeCopyMode *sameShell*,
 const **MbPlacement3D** & **place**,
 double *angle*,
 const RPAArray<**MbFace**> & **faces**,
 MbeFacePropagation *propagation*,
 bool *reverse*,
 const MbSNameMaker & *names*,
MbSolid *& **result**)

выполняет построение тела с уклонением указанных граней тела от нейтральной изоплоскости на заданный угол.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **place** – нейтральная плоскость,
- *angle* – угол уклона,
- **faces** – множество уклоняемых граней,
- *propagation* – признак захвата граней, гладко стыкующихся с уклоняемыми гранями,
- *reverse* – флаг обратного направления уклона,
- *names* – именованье построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле action_solid.h.

Метод строит тело, грани которого уклонены относительно их положения в исходном теле **solid**. Параметр *params* задает параметры построения. Параметр *sameShell* управляет передачей граней,

ребер и вершин от исходного тела **solid** построенному телу **result**. Плоскость XY локальной системы координат **place** определяет плоскость, относительно которой уклоняются грани тела. Параметр *angle* задает угол уклона. Множество **faces** содержит уклоняемые грани. Параметр *propagation* управляет добавлением к множеству уклоняемых граней **faces** других граней тела, гладко стыкующихся с уклоняемыми гранями. Параметр *reverse* управляет направлением уклона. Параметр *names* обеспечивает именование граней построенного тела.

Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление *MbeCopyMode* описано в параграфе [0.7.9. Копирование множества граней MbFaceShell](#).

На рис. М.2.24.1 показаны исходное тело **solid**, локальная система координат **place**, относительно плоскости XY которой выполняется уклон граней, и уклоняемые грани **faces**.

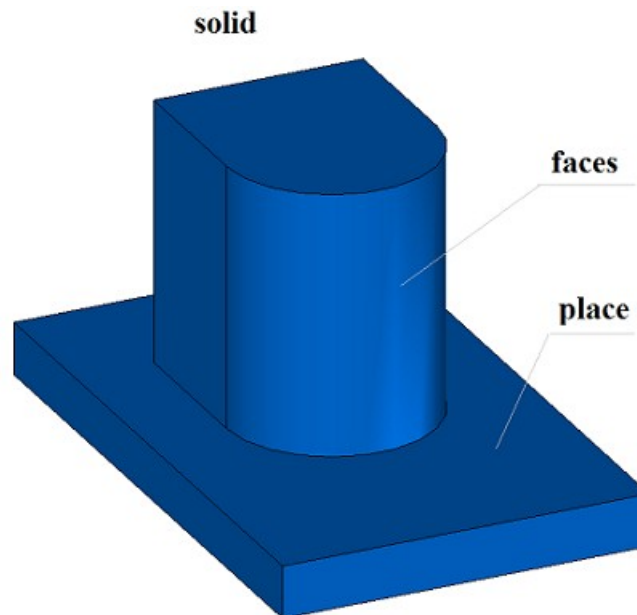


Рис. М.2.24.1.

На рис. М.2.24.2 приведен результат построения тела с уклоном указанных граней.

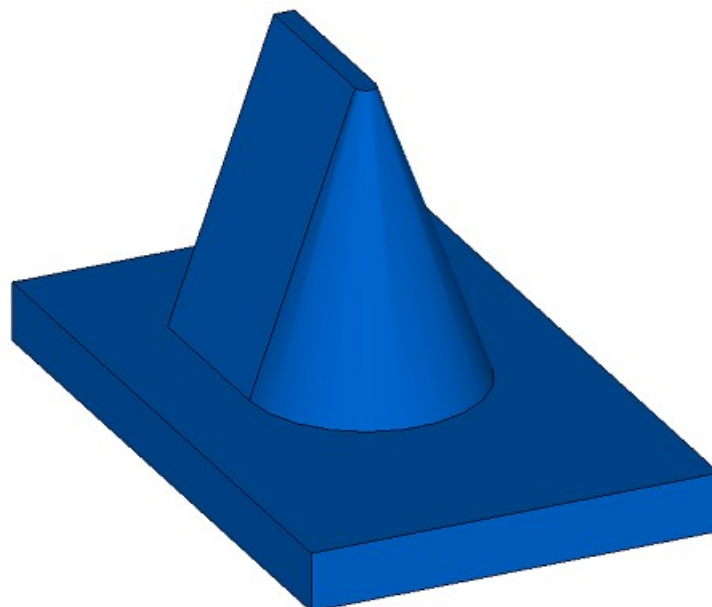


Рис. М.2.24.2.

Метод **DraftSolid** добавляет в журнал построенного тела строитель MbDraftSolid, который содержит все необходимые данные для выполнения операции. Строитель MbDraftSolid объявлен в файле `cg_draft_solid.h`.

Тестовое приложение `test.exe` выполняет разбиение указанных граней тела командой меню «Создать->Тело->Обработкой граней->Уклонением граней».

M.2.25. Размножение тела

Метод
MbResultType

DuplicationSolid (const MbSolid & **solid**,
const DuplicationValues & *parameters*,
const MbSNameMaker & *names*,
MbSolid *& **result**)

выполняет построение копий исходного тела, трансформацию их по указанному закону и объединение в одно тело.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *parameters* – параметры построения,
- *names* – именователъ граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_solid.h`.

Параметр *names* обеспечивает именование граней построенного тела. Параметр *parameters* задает параметры построения. На рис. M.2.25.1 приведены данные, используемые при построении, и схема наследования параметров построения от абстрактного класса DuplicationValues.

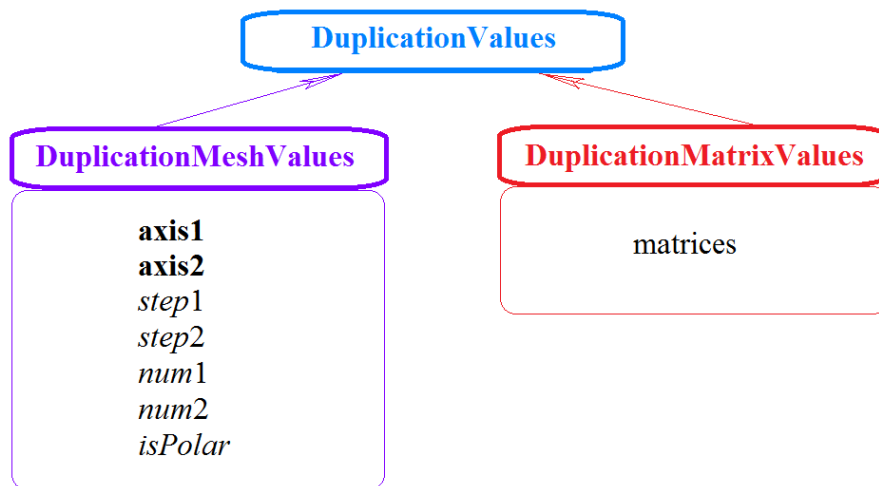


Рис. M.2.25.1.

Для размножения тела, копии которого располагаются по двумерной сетке, следует использовать параметры DuplicationMeshValues. Поддерживаются два типа размножения: по двум направлениям и по полярной сетке. Параметр *parameters.isPolar* определяет тип сетки. Если *parameters.isPolar=false*, то исходное тело и построенные копии располагаются в узлах двумерной сетки с осями *parameters.axis1* и *parameters.axis2*. Началом отсчета служит исходное тело. Вдоль оси *parameters.axis1* располагаются *parameters.num1* копий тела с шагом *parameters.step1*, вдоль оси *parameters.axis2* располагаются *parameters.num2* копий тела с шагом *parameters.step2*, включая исходное тело. Если *parameters.isPolar=true*, построенные копии располагаются в узлах полярной сетки. Началом отсчета служит исходное тело. Радиальное направление сетки определяется вектором *parameters.axis1*, а ось вращения определяется векторным произведением *parameters.axis1* и

parameters.axis2. Вдоль радиальных направлений располагаются *parameters.num1* копий тела с шагом *parameters.step1*, на каждой окружности располагаются *parameters.num2* копий тела с угловым шагом *parameters.step2*.

Для размножения тела, копии которого трансформированы по набору матриц, следует использовать параметры *DuplicationMatrixValues*. Множество матриц трансформации определено параметром *parameters.matrices*.

Если после построения копии или исходное тело пересекаются друг с другом, то выполняется булева операция объединения пересекшихся тел. На рис. М.2.25.2 приведен результат построения размноженного по полярной сетки тела.

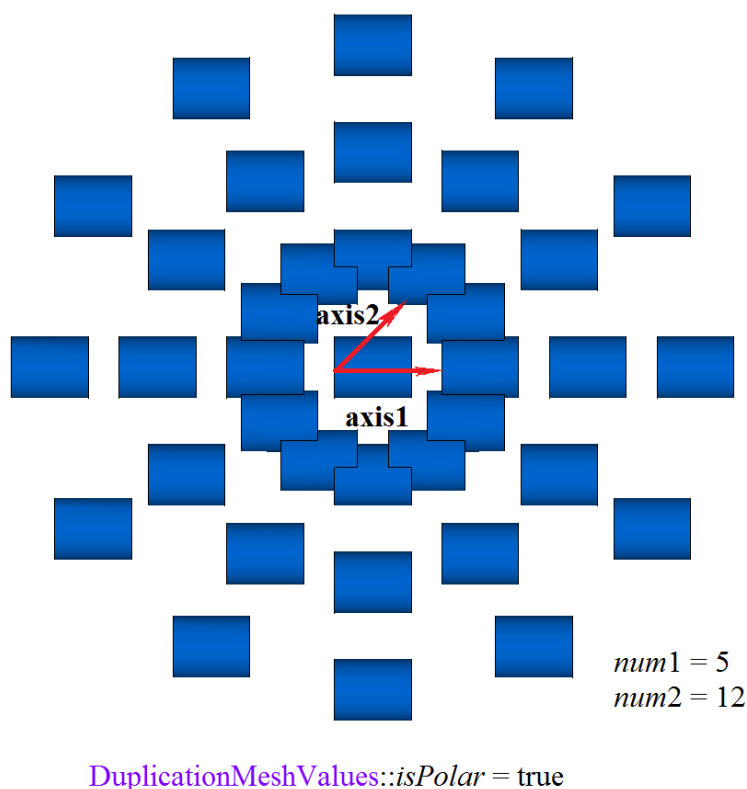


Рис. М.2.25.2.

Метод **DuplicationSolid** добавляет в журнал построенного тела строитель *MbDuplicationSolid*, который содержит все необходимые данные для выполнения операции. Строитель *MbDuplicationSolid* объявлен в файле *cg_duplication_solid.h*.

Тестовое приложение *test.exe* выполняет разбиение указанных граней тела командами меню «Создать->Тело->На базе тела->Размножением по сетке» и «Создать->Тело->На базе тела->Размножением матрицами».

М.3. МЕТОДЫ ПОСТРОЕНИЯ ДВУМЕРНЫХ КРИВЫХ

Двумерные кривые используются для описания области определения поверхностей, для работы с кривыми на поверхностях, для построения кривых пересечения поверхностей, для построения поверхностей сопряжения. При построении твердых тел двумерные кривые применяются в качестве входных параметров в виде элементов эскизов. Кроме того, двумерные кривые служат элементами плоских проекций геометрических моделей. Все двумерные кривые являются наследниками класса `MbCurve` и представлены в главе [О.3. КРИВЫЕ ДВУМЕРНОГО ПРОСТРАНСТВА](#). Кривые могут быть построены непосредственным вызовом соответствующих конструкторов или методами, приведёнными в данном параграфе.

М.3.1. Построение двумерной прямой линии и отрезка

Метод

`MbResultType`

```
Line ( const MbCartPoint & point1,  
        const MbCartPoint & point2,  
        MbCurve *& result )
```

выполняет построение двумерной прямой по двум несовпадающим точкам.

Входными параметрами метода являются:

- *point1* – первая точка, по которой проходит прямая,
- *point2* – вторая точка, по которой проходит прямая.

Выходным параметром метода является построенная кривая *result*.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`. Метод объявлен в файле `action_curve.h`.

Параметр *point1* определяет начальную точку прямой, соответствующую нулевому параметру прямой. Вектор, начинающийся в точке *point1* и оканчивающийся в точке *point2*, определяет направление прямой, рис. М.3.1.1. Производная прямой линии имеет единичную длину.

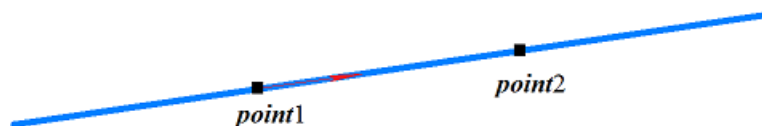


Рис. М.3.1.1.

Прямая линия описана в параграфе [3.2. Двумерная прямая MbLine](#).

Метод

`MbResultType`

```
Segment ( const MbCartPoint & point1,  
           const MbCartPoint & point2,  
           MbCurve *& result )
```

выполняет построение двумерного отрезка прямой по двум несовпадающим точкам.

Входными параметрами метода являются:

- *point1* – начальная точка отрезка,
- *point2* – конечная точка отрезка.

Выходным параметром метода является построенная кривая *result*.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`. Метод объявлен в файле `action_curve.h`.

Параметр *point1* определяет начальную точку отрезка прямой, соответствующую нулевому параметру кривой. Параметр *point2* определяет конечную точку отрезка прямой, соответствующую параметру кривой, равному единице, рис. М.3.1.2.



Рис. М.3.1.2.

Отрезок прямой описан в параграфе [О.3.3. Двумерный отрезок прямой MbLineSegment](#).

М.3.2. Построение двумерной окружности, эллипса и их дуг

Метод

MbResultType

```
Arc ( const MbCartPoint & centre,
      const SArray<MbCartPoint> & points,
      bool closed,
      double angle,
      double & a,
      double & b,
      MbCurve *& result )
```

выполняет построение двумерной дуги эллипса, в частном случае метод выполняет построение двумерной дуги окружности.

Входными параметрами метода являются:

- **centre** – центр эллипса,
- **points** – множество точек, которое может быть пустым,
- **closed** – флаг циклической замкнутости кривой,
- **angle** – угол, определяющий размер дуги эллипса,
- **a** – длина первой полуоси эллипса (при непустом множестве **points** вычисляется),
- **b** – длина второй полуоси эллипса (при непустом множестве **points** вычисляется).

Выходными параметрами метода являются длины полуосей эллипса и построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType. Метод объявлен в файле `action_curve.h`.

Кривая может быть построена как по заданным точкам, так и по скалярным параметрам. Параметр **centre** определяет центральную точку эллипса. Множество точек **points** может быть пустым, в этом случае должны быть ненулевыми длины полуосей эллипса **a** и **b**.

Если множество точек **points** содержит два элемента, то точки **centre**, **points[0]**, **points[1]** определяют плоскость расположения осей **axisX** и **axisY** локальной системы координат эллипса: ось **axisX** локальной системы координат эллипса направлена из центра в точку **points[0]**, ось **axisY** локальной системы координат эллипса ортогональна оси **axisX** и направлена из центра в сторону точки **points[1]**. Расстояние между точками **centre** и **points[0]** определяет длину первой полуоси эллипса **a**, а расстояние между точкой **centre** и проекцией точки **points[1]** на ось **axisY** определяет длину второй полуоси эллипса **b**, рис. М.3.2.1.

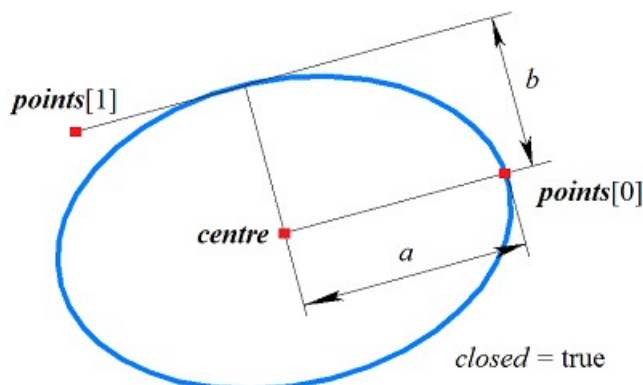


Рис. М.3.2.1.

Если множество точек *points* содержит один элемент, то будет построена окружность радиусом, равным расстоянию между точками *centre* и *points[0]*. Ось *axisX* локальной системы координат окружности будет направлена из центра в точку *points[0]*, а ось *axisY* локальной системы координат будет ортогональна оси *axisX*.

Если множество точек *points* содержит три элемента, то будет построен эллипс с центром в точке *centre*, проходящий через точки *points[0]*, *points[1]* и *points[2]*. Положение осей локальной системы координат эллипса и длины полуосей эллипса будут вычислены по точкам *centre*, *points[0]*, *points[1]* и *points[2]*, рис. М.3.2.2.

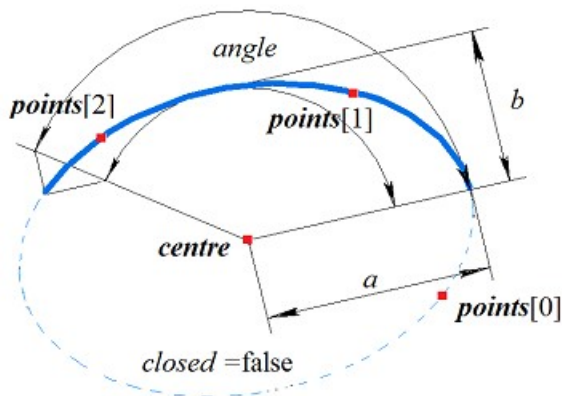


Рис. М.3.2.2.

Дуга эллипса описана в параграфе [О.3.4. Двумерная дуга эллипса MbArc](#).

Если множество точек *points* пусто, то длины полуосей эллипса *a* и *b* являются входными параметрами, а оси *axisX* и *axisY* локальной системы координат эллипса совпадают с глобальными осями координат.

Параметр *closed* определяет циклическую замкнутость кривой. Если *closed=false*, то должен быть ненулевым параметр *angle*, который определяет угол раствора дуги эллипса в параметрических единицах.

В частном случае, когда $a=b$, рассматриваемый метод выполняет построение окружности ($closed=true$ или $angle=0$) или дуги окружности ($closed=false$ и $angle>0$ и $angle<2\pi$).

М.3.3. Построение двумерных кривых по контрольным точкам

Метод

MbResultType

SplineCurve (const SArray<MbCartPoint> & *points*,
 bool *closed*,
 MbPlaneType *curveType*,
 MbCurve *& *result*)

выполняет построение двумерной кривой заданного типа по указанному множеству контрольных точек.

Входными параметрами метода являются:

- *points* – множество контрольных точек,
- *closed* – флаг циклической замкнутости кривой,
- *curveType* – тип кривой.

Выходным параметром метода является построенная кривая *result*.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_curve.h`.

Параметр *points* содержит контрольные точки кривой. Параметр *closed* определяет циклическую замкнутость построенной кривой. Параметры *curveType* определяют тип кривой, от которого зависит

форма кривой. Для разных типов создаваемой кривой требуется разное количество контрольных точек. В табл. М.3.3.1 приведено количество контрольных точек множества *points*, необходимое для создания кривой типа *curveType*.

Таблица М.3.3.1.

<i>curveType</i>	Тип кривой	Количество контрольных точек
pt_LineSegment	отрезок прямой	2 точки
pt_Arc	дуга окружности	3 точки
pt_Polyline	ломаная	2 и более точек
pt_Nurbs	NURBS кривая	2 и более точек
pt_Hermit	сплайн Эрмита	2 и более точек
pt_Bezier	кривая Безье	2 и более точек
pt_CubicSpline	кубический сплайн	2 и более точек

Для типа *curveType*=pt_LineSegment будет построен отрезок прямой, начинающийся в точке *points*[0] и оканчивающийся в точке *points*[1]. Отрезок прямой описан в параграфе [О.3.3. Двумерный отрезок прямой MbLineSegment](#).

Для типа *curveType*=pt_Arc при *closed*=false будет построена дуга окружности, начинающаяся в точке *points*[0], проходящая через точку *points*[1] и оканчивающаяся в точке *points*[2], рис. М.3.3.1. При *closed*=true будет построена окружность, проходящая через точки *points*[0], *points*[1], *points*[2]. Дуга окружности и эллипса описана в параграфе [О.3.4. Двумерная дуга эллипса MbArc](#).

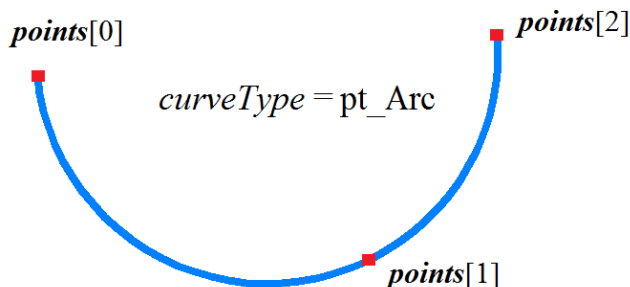


Рис. М.3.3.1.

Для типа *curveType*=pt_Polyline будет построена ломаная, проходящая через точки *points*[0], *points*[1], ..., *points*[*n*], рис. М.3.3.2. При *closed*=true будет построена циклически замкнутая ломаная, содержащая участок между точками *points*[0] и *points*[*n*]. Ломаная линия описана в параграфе [О.3.5. Двумерная ломаная линия MbPolyline](#).

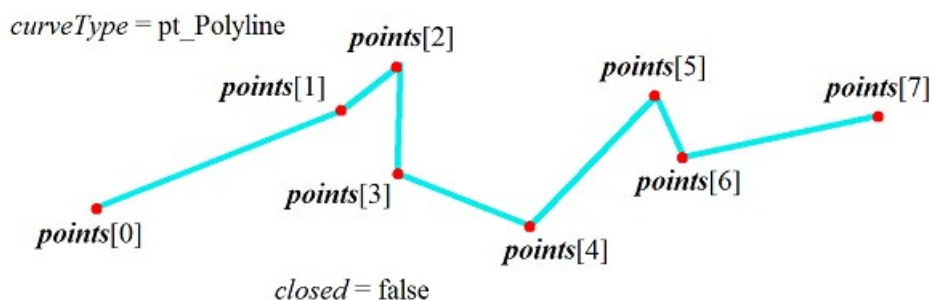


Рис. М.3.3.2.

Для типа $curveType=pt_Nurbs$ будет построен неоднородный рациональный B -сплайн (NURBS) четвертого порядка, рис. М.3.3.3. Контрольные точки сплайна будут определены из условия прохождения сплайна через точки $points[0]$, $points[1]$,..., $points[n]$. При $closed=true$ будет построен циклически замкнутый сплайн. NURBS кривая описана в параграфе [О.3.6. Двумерная NURBS-кривая MbNurbs](#).

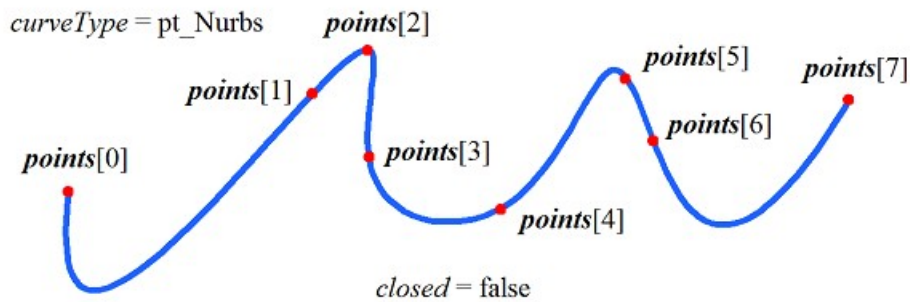


Рис. М.3.3.3.

Для типа $curveType=pt_Hermit$ будет построена составная кривая, состоящая из гладко стыкующихся сплайнов Эрмита третьего порядка. Каждый из сплайнов Эрмита третьего порядка будет соединять соседние точки $points[i-1]$ и $points[i]$, рис. М.3.3.4. При $closed=true$ будет построена циклически замкнутая кривая, содержащая сплайн Эрмита между точками $points[0]$ и $points[n]$. Составной сплайн Эрмита третьей степени описан в параграфе [О.3.7. Двумерная кривая Эрмита MbHermit](#).

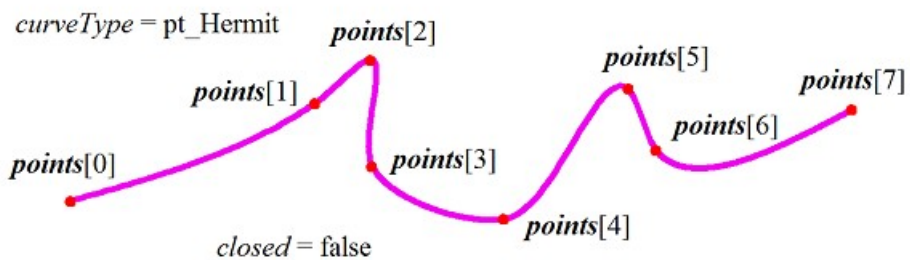


Рис. М.3.3.4.

Для типа $curveType=pt_Bezier$ будет построена составная кривая, состоящая из гладко стыкующихся сплайнов Безье третьего порядка. Каждый из сплайнов Безье третьего порядка будет соединять соседние точки $points[i-1]$ и $points[i]$, рис. М.3.3.5. При $closed=true$ будет построена циклически замкнутая кривая, содержащая сплайн Безье между точками $points[0]$ и $points[n]$. Составная кривая Безье третьей степени описана в параграфе [О.3.8. Двумерная составная кривая Безье MbBezier](#).

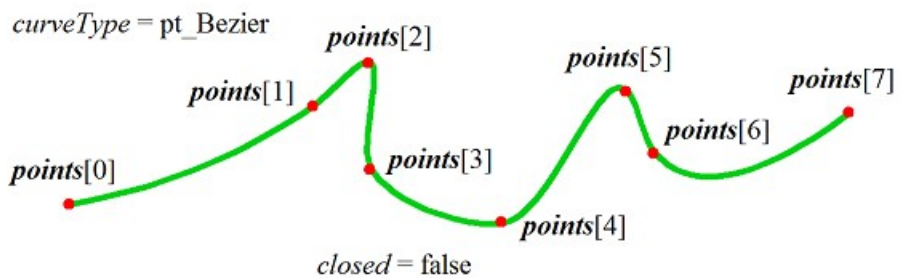


Рис. М.3.3.5.

Для типа $curveType=pt_CubicSpline$ будет построен кубический сплайн, проходящий через точки $points[0]$, $points[1]$,..., $points[n]$, рис. М.3.3.6. При $closed=true$ будет построена циклически замкнутая

кривая, содержащая участок между точками *points[0]* и *points[n]*. Кубический сплайн описан в параграфе [О.3.9. Двумерная кубический сплайн MbCubicSpline](#).

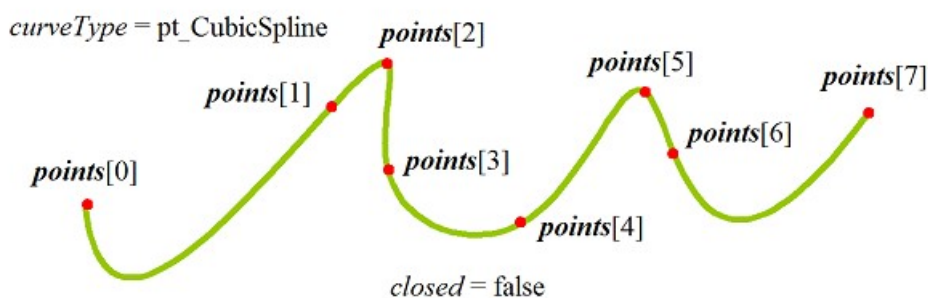


Рис. М.3.3.6.

Для сравнения на рис. М.3.3.7 приведены NURBS кривая, составной сплайн Эрмита, составная кривая Безье, кубический сплайн, построенные по одним и тем же контрольным точкам *points[0]*, *points[1]*,..., *points[n]* имеющие одну и ту же степень.

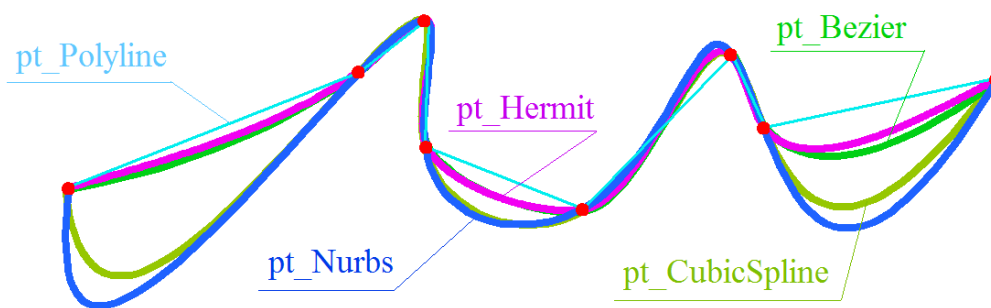


Рис. М.3.3.7.

Как можно видеть, кривые имеют различную форму.

М.3.4. Построение двумерной NURBS кривой

Метод

MbResultType

```
NurbsCurve ( const SArray<MbCartPoint> & points,
              const SArray<double> & weights,
              size_t degree,
              const SArray<double> & knots,
              bool closed,
              MbCurve *& result )
```

выполняет построение двумерной NURBS-кривой на заданном множестве контрольных точек.

Входными параметрами метода являются:

- *points* – множество контрольных точек,
- *weights* – множество весов контрольных точек,
- *degree* – порядок кривой (*B*-сплайнов),
- *knots* – множество параметрических узлов (узловой вектор),
- *closed* – флаг циклической замкнутости кривой.

Выходным параметром метода является построенная кривая *result*.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле *action_curve.h*.

Рассматриваемый метод строит NURBS-кривую (кривую на основе B -сплайнов), которая описана в параграфе [O.3.6. Двумерная NURBS-кривая MbNurbs](#). Множество весов $weights$ должно быть согласовано с множеством контрольных точек $points$. Порядок $degree$ кривой не должен превосходить количество контрольных точек. Параметр $closed$ определяет циклическую замкнутость построенной кривой. Узловой вектор $knots$ представляет собой неубывающую последовательность действительных чисел и определяет область определения параметра кривой и форму кривой. Для $closed=false$ узловой вектор должен содержать число элементов, равное числу контрольных точек плюс порядок кривой. Чтобы незамкнутая NURBS-кривая проходила через крайние контрольные точки, необходимо, чтобы первые $degree$ элементов узлового вектора $knots$ были равны между собой и последние $degree$ элементов узлового вектора $knots$ были равны между собой. Для $closed=true$ узловой вектор должен содержать число элементов, равное числу контрольных точек плюс удвоенный порядок кривой и минус единица. На рис. М.3.4.1 приведена замкнутая NURBS-кривая четвертого порядка.

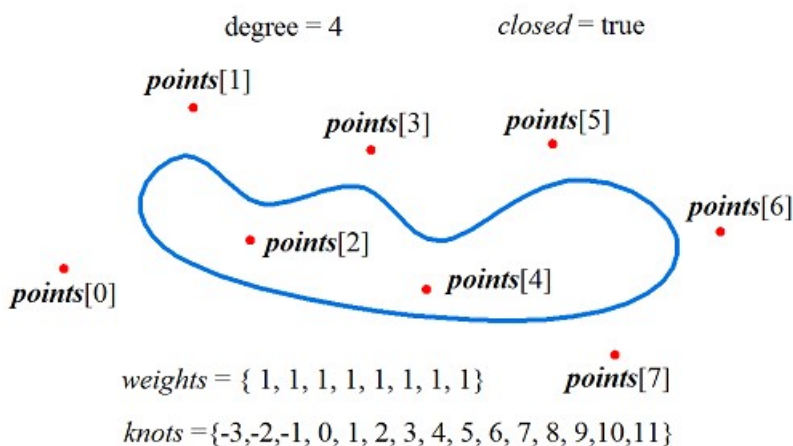


Рис. М.3.4.1.

На рис. М.3.4.2 приведена незамкнутая NURBS-кривая четвертого порядка.

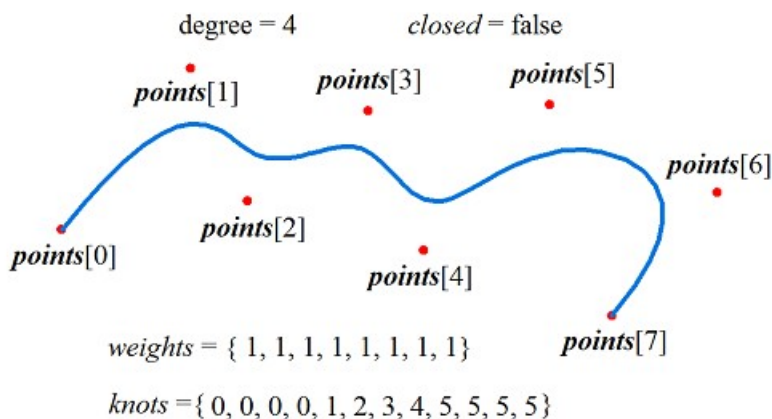


Рис. М.3.4.2.

На рис. М.3.4.3 приведены три замкнутые NURBS-кривые различного порядка, построенные по одним и тем же контрольным точкам.

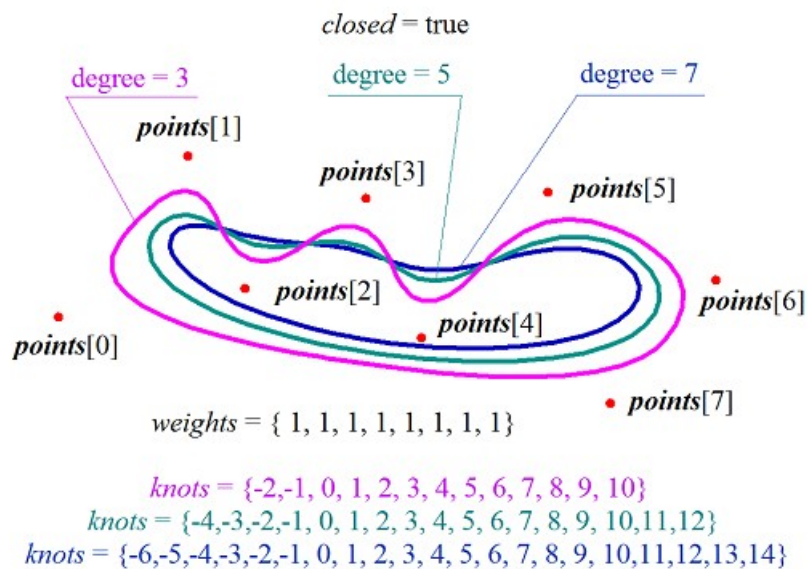


Рис. М.3.4.3.

На рис. М.3.4.4 приведены три незамкнутые NURBS-кривые различного порядка, построенные по одним и тем же контрольным точкам.

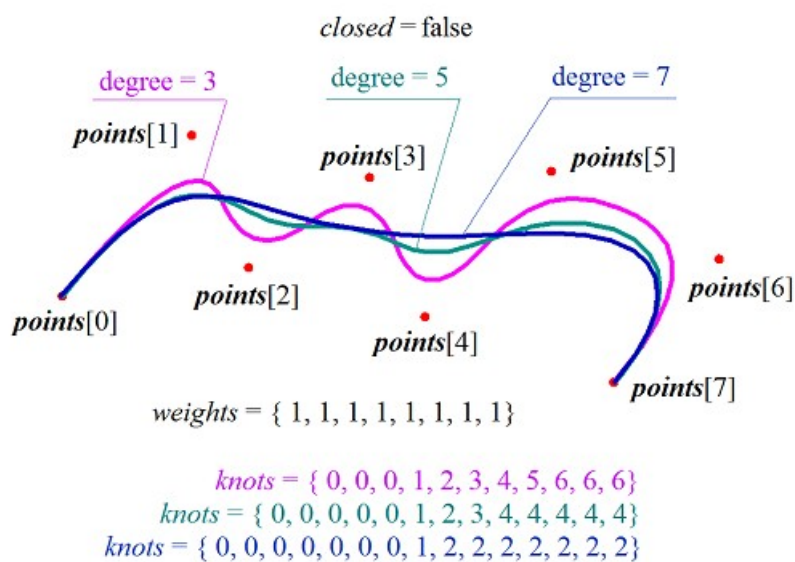


Рис. М.3.4.4.

На рис. М.3.4.5 приведены три незамкнутые NURBS-кривые четвертого порядка, имеющие различные значения весов контрольных точек.

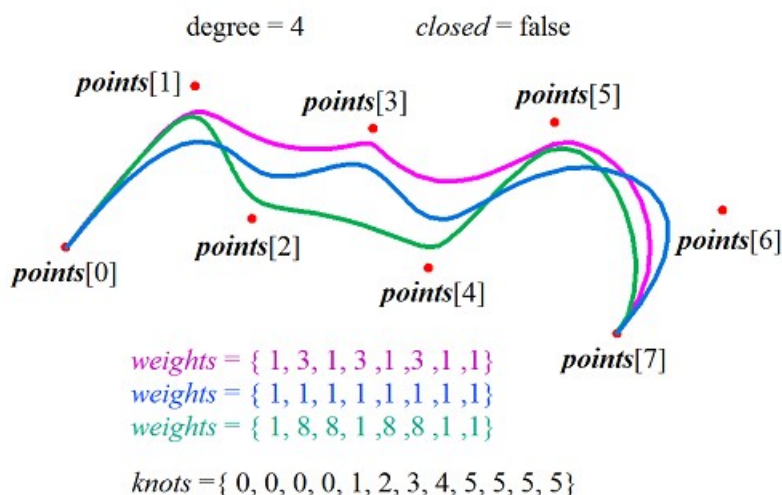


Рис. М.3.4.5.

На рис. М.3.4.6 приведена незамкнутая NURBS-кривая третьего порядка, по форме совпадающая с дугой окружности. Расстояние между контрольными точками **points[0]** и **points[1]** равно расстоянию между контрольными точками **points[1]** и **points[2]**, а вес **weights[1]** средней контрольной точки кривой равен весу крайних контрольных точек, умноженному на косинус половины угла дуги.

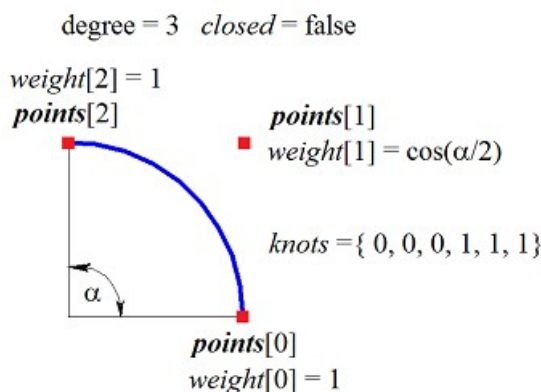


Рис. М.3.4.6.

Метод

MbResultType

NurbsCopy (const MbCurve & curve,
MbCurve *& result)

выполняет построение NURBS-копии заданной двумерной кривой.

Входным параметром метода является исходная кривая **curve**.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает **rt_Success**, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле **action_curve.h**.

Рассматриваемый метод построит NURBS-кривую, форма которой копирует форму исходной кривой. Для большинства типов кривых NURBS-копия по форме полностью совпадает с исходной кривой. В случае невозможности точно воспроизвести форму исходной кривой NURBS-копия аппроксимирует исходную кривую с погрешностью, не превышающей 0,0001.

М.3.5. Построение выпуклой равносторонней двумерной ломаной

Метод

MbResultType

RegularPolygon (const [MbCartPoint](#) & *centre*,
const [MbCartPoint](#) & *point*,
size_t *vertexCount*,
bool *describe*,
[MbCurve](#) *& *result*)

выполняет построение замкнутой двумерной ломаной линии, представляющей собой правильный многоугольник, вписанный в обозначенную окружность или описанный около окружности.

Входными параметрами метода являются:

- *centre* – центр окружности, описанной или вписанной в равносторонний многоугольник,
- *point* – точка на окружности,
- *vertexCount* – количество вершин многоугольника,
- *describe* – флаг описанной или вписанной окружности.

Выходным параметром метода является построенная кривая *result*.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_curve.h`.

Параметры *centre* и *point* определяют окружность, в которую будет вписан (*describe=false*) многоугольник с числом вершин *vertexCount* или вокруг которой будет описан (*describe=true*) многоугольник с числом вершин *vertexCount*. Центр окружности будет расположен в точке *centre*, окружность пройдет через точку *point*. Параметр *vertexCount* определяет число вершин правильного многоугольника. Построенная кривая будет замкнутой ломаной линией, описанной в параграфе [O.3.5. Двумерная ломаная линия MbPolyline](#). На рис. М.3.5.1 приведен вписанный в окружность правильный многоугольник, а на рис. М.3.5.2 приведен описанный вокруг окружности правильный многоугольник.

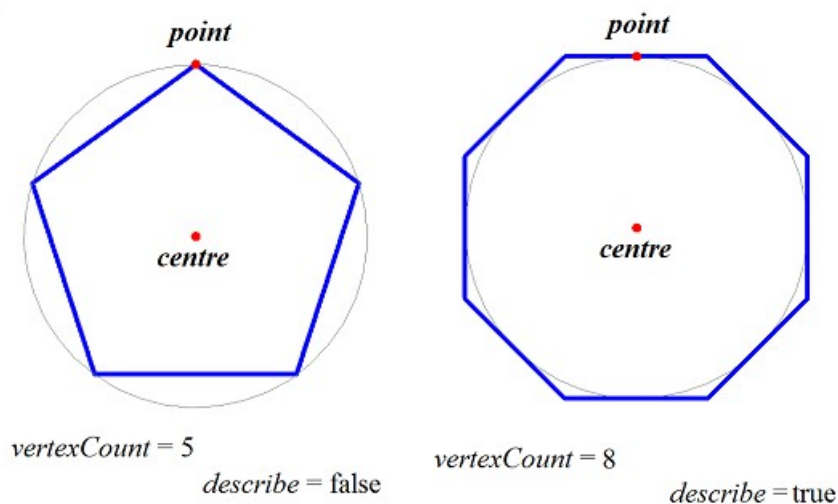


Рис. М.3.5.1.

Рис. М.3.5.2.

При $vertexCount \leq 1$ рассматриваемый метод построит окружность с центром в точке *centre*, проходящую через точку *point*. При $vertexCount = 2$ рассматриваемый метод построит прямоугольник с противоположными сторонами, параллельными глобальным осям координат и противоположными вершинами, располагающимися в точках *centre* и *point*, рис. М.3.5.3.



Рис. М.3.5.3.

М.3.6. Построение двумерной косинусоиды

Метод
 MbResultType
Cosinusoid (const [MbCartPoint](#) & *point0*,
 const [MbCartPoint](#) & *point1*,
 const [MbCartPoint](#) & *point2*,
 double *phase*,
 double *waveLength*,
[MbCurve](#) *& *result*)

выполняет построение косинусоиды (двумерной гармонической кривой).

Входными параметрами метода являются:

- *point0* – начало локальной системы координат,
- *point1* – точка на оси X локальной системы координат,
- *point2* – точка, определяющая ось Y локальной системы координат,
- *phase* – сдвиг фазы гармонической кривой,
- *waveLength* – длина волны гармонической кривой.

Выходным параметром метода является построенная кривая *result*.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve.h`.

Точки *point0*, *point1* и *point2* определяют локальную систему координат косинусоиды. Начало локальной системы координат косинусоиды будет расположено в точке *point0*, ось *axisX* локальной системы координат будет направлена из точки *point0* в точку *point1*, ось *axisY* локальной системы координат будет ортогональна оси *axisX* и направлена из точки *point0* в сторону точки *point2*. Точки *point0*, *point1* и *point2* не должны совпадать или лежать на одной прямой. Двумерная косинусоида описана в параграфе [О.3.14. Двумерная косинусоида MbCosinusoid](#). Амплитуда косинусоиды равна длине проекции точки *point2* на ось *axisY*. Длина волны косинусоиды равна *waveLength*. Число волн косинусоиды определяется расстоянием между точками *point0* и *point1*. Параметр *phase* определяет сдвиг фазы косинусоиды, рис. М.3.6.1.

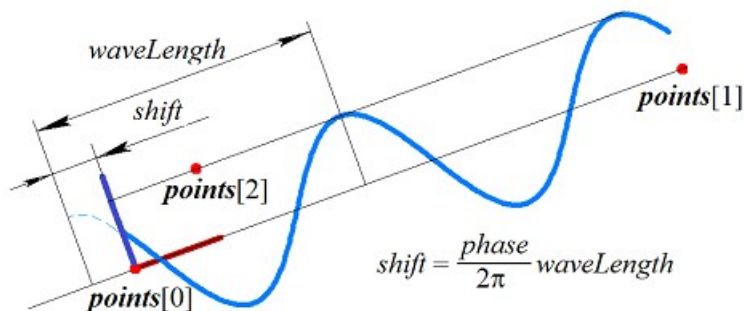


Рис. М.3.6.1.

М.3.7. Построение двумерной составной кривой

Метод
MbResultType
CreateContour ([MbCurve](#) & *curve*,
[MbContour](#) *& *result*)

выполняет построение составной кривой на базе одной исходной кривой.

Входным параметром метода является исходная кривая *curve*.

Выходным параметром метода является построенная кривая *result*.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve.h`.

Метод создает составную кривую *result* на базе исходной кривой *curve*. Если исходная кривая *curve* является составной кривой, то в кривую *result* будут положены составляющие элементы исходной кривой. Двумерная составная кривая описана в параграфе [О.3.17. Двумерный контур MbContour](#).

Метод
MbResultType
AddCurveToContour ([MbCurve](#) & *curve*,
[MbContour](#) & *contour*,
bool *toEnd*)

модифицирует составную кривую путём добавления кривой.

Входными параметрами метода являются:

- *curve* – добавляемая кривая,
- *contour* – модифицируемая составная кривая,
- *toEnd* – флаг места добавления кривой.

Выходным параметром метода является модифицируемая кривая *contour*.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve.h`.

Метод модифицирует составную кривую *contour* путём добавления кривой *curve* в начало или конец составной кривой. Если *toEnd*=true, то кривая *curve* будет добавлена в конец составной кривой *contour*, если *toEnd*=false, то кривая *curve* будет добавлена в начало составной кривой *contour*. Точки стыковки модифицируемой кривой и добавляемой кривой должны совпадать, рис. М.3.7.1.

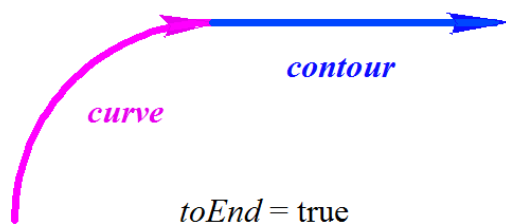


Рис. М.3.7.1.

Если добавляемая кривая *curve* является составной кривой, то в кривую *contour* будут добавлены составляющие элементы добавляемой кривой.

М.3.8. Построение кривых пересечения поверхности и плоскости

Метод
void
SurfaceSection (const [MbSurface](#) & *surface*,
const [MbPlacement3D](#) & *place*,

RPArray<MbCurve> & result)

выполняет построение кривых пересечения поверхности с плоскостью.

Входными параметрами метода являются:

- **surface** – поверхность,
- **place** – локальная система координат плоскости.

Выходным параметром метода является множество построенных кривых **result**.

Метод не имеет возвращаемого значения.

Метод объявлен в файле action_curve.h.

Метод строит пересечение поверхности **surface** с плоскостью XY локальной системы координат **place**. Кривые **result** строятся в плоскости XY локальной системы координат **place**. На рис. М.3.8.1 приведен пример построения двумерных кривых пересечения поверхности тора и плоскости XY локальной системы координат.

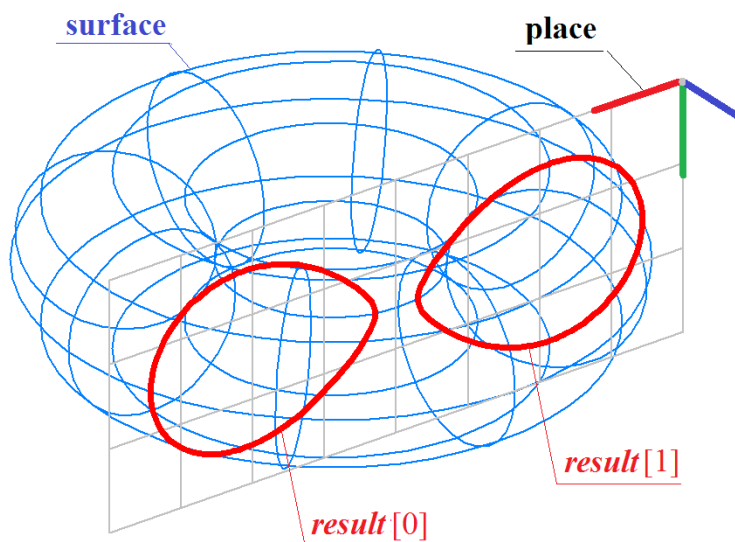


Рис. М.3.8.1.

М.3.9. Построение двумерной кривой ребра грани

Метод

MbResultType

FaceBoundSegment (const MbFace & face,
size_t loopIndex,
size_t edgeIndex,
const MbSurface & surface,
VERSION version,
MbCurve *& result)

выполняет построение проекции ребра грани на поверхность.

Входными параметрами метода являются:

- **face** – грань,
- **loopIndex** – индекс цикла в грани, в котором расположено проецируемое ребро,
- **edgeIndex** – индекс ребра в цикле,
- **surface** – поверхность проецирования,
- **version** – версия построения.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле action_curve.h.

Метод строит проекцию ребра грани **face** на поверхность **surface**. Ребро в грани определено индексом цикла грани **loopIndex** и индексом ребра в цикле. Кривая **result** строится в пространстве

параметров поверхности **surface**. На рис. М.3.9.1 приведен пример построения проекции ребра грани на заданную поверхность.

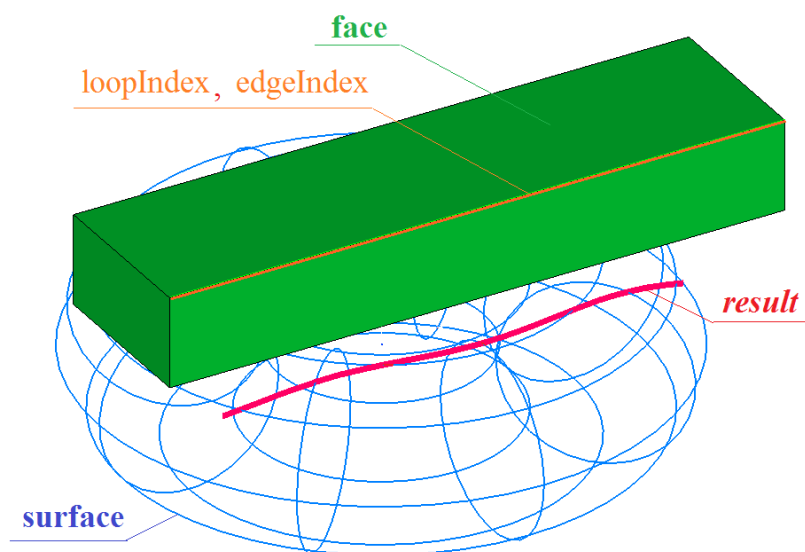


Рис. М.3.9.1.

М.3.10. Построение проекции кривой на поверхность

Метод
MbResultType
SurfaceBoundContour (const MbSurface & **surface**,
const MbCurve3D & **curve**,
VERSION version,
MbContour *& **result**)

выполняет построение двумерной кривой в пространстве параметров поверхности для пространственной кривой, лежащей на этой поверхности.

Входными параметрами метода являются:

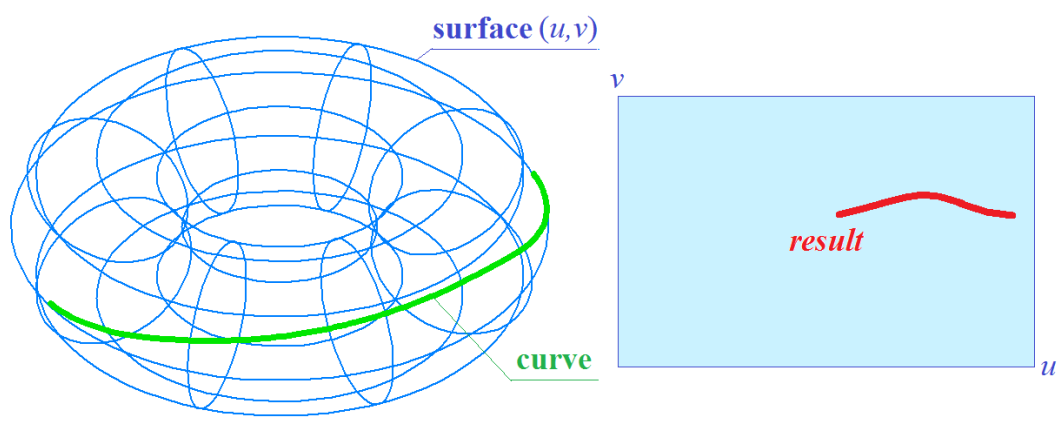
- **surface** – поверхность,
- **curve** – пространственная кривая,
- version – версия построения.

Выходным параметром метода является построенная составная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve.h`.

Метод строит проекцию пространственной кривой **curve** на поверхность **surface**. Пространственная кривая должна располагаться на поверхности **surface**. Кривая **result** строится в пространстве параметров поверхности **surface**. На рис. М.3.10.1 приведен пример построения проекции пространственной кривой на заданную поверхность.



Puc. M.3.10.1.

М.4. МЕТОДЫ ПОСТРОЕНИЯ КРИВЫХ

Кривые служат строительным материалом для построения поверхностей. По кривым выполняется стыковка поверхностей друг с другом. На базе кривых строятся рёбра, которые используются в твёрдотельной и каркасной моделях. В определенных случаях кривые выступают в роли опорных объектов для позиционирования других элементов геометрической модели. Все кривые являются наследниками класса `MbCurve3D` и представлены в главе [0.4. КРИВЫЕ](#). Кривые могут быть построены непосредственным вызовом соответствующих конструкторов или методами, приведёнными в данном параграфе.

М.4.1. Построение прямой линии и отрезка

Метод

`MbResultType`

```
Line ( const MbCartPoint3D & point1,  
        const MbCartPoint3D & point2,  
        MbCurve3D *& result )
```

выполняет построение прямой по двум несовпадающим точкам.

Входными параметрами метода являются:

- **point1** – первая точка, по которой проходит прямая,
- **point2** – вторая точка, по которой проходит прямая.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve3d.h`.

Параметр **point1** определяет начальную точку прямой, соответствующую нулевому параметру прямой. Вектор, начинающийся в точке **point1** и оканчивающийся в точке **point2**, определяет направление прямой, рис. М.4.1.1. Производная прямой линии имеет единичную длину.



Рис. М.4.1.1.

Прямая линия описана в параграфе [0.4.2. Прямая линия MbLine3D](#).

Метод

`MbResultType`

```
Segment ( const MbCartPoint3D & point1,  
           const MbCartPoint3D & point2,  
           MbCurve3D *& result )
```

выполняет построение отрезка прямой по двум несовпадающим точкам.

Входными параметрами метода являются:

- **point1** – начальная точка отрезка,
- **point2** – конечная точка отрезка.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve3d.h`.

Параметр **point1** определяет начальную точку отрезка прямой, соответствующую нулевому параметру кривой. Параметр **point2** определяет конечную точку отрезка прямой, соответствующую параметру кривой, равному единице, рис. М.4.1.2.



Рис. М.4.1.2.

Отрезок прямой описан в параграфе [О.4.3. Отрезок прямой MbLineSegment3D](#).

М.4.2. Построение окружности, эллипса и их дуг

Метод

MbResultType

Arc (const [MbCartPoint3D](#) & centre,
 const SArray<[MbCartPoint3D](#)> & points,
 bool closed,
 double angle,
 double & a,
 double & b,
[MbCurve3D](#) *& result)

выполняет построение дуги эллипса, в частном случае метод выполняет построение дуги окружности.

Входными параметрами метода являются:

- **centre** – центр эллипса,
- **points** – множество точек, которое может быть пустым,
- *closed* – флаг циклической замкнутости кривой,
- *angle* – угол, определяющий размер дуги эллипса,
- *a* – длина первой полуоси эллипса (при непустом множестве **points** вычисляется),
- *b* – длина второй полуоси эллипса (при непустом множестве **points** вычисляется).

Выходными параметрами метода являются длины полуосей эллипса и построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType. Метод объявлен в файле `action_curve3d.h`.

Кривая может быть построена как по заданным точкам, так и по скалярным параметрам. Параметр **centre** определяет центральную точку эллипса. Множество точек **points** может быть пустым, в этом случае должны быть ненулевыми длины полуосей эллипса *a* и *b*.

Если множество точек **points** содержит два элемента, то точки **centre**, **points[0]**, **points[1]** определяют плоскость расположения осей **axisX** и **axisY** локальной системы координат эллипса: ось **axisX** локальной системы координат эллипса направлена из центра в точку **points[0]**, ось **axisY** локальной системы координат эллипса ортогональна оси **axisX** и направлена из центра в сторону точки **points[1]**. Расстояние между точками **centre** и **points[0]** определяет длину первой полуоси эллипса *a*, а расстояние между точкой **centre** и проекцией точки **points[1]** на ось **axisY** определяет длину второй полуоси эллипса *b*, рис. М.4.2.1.

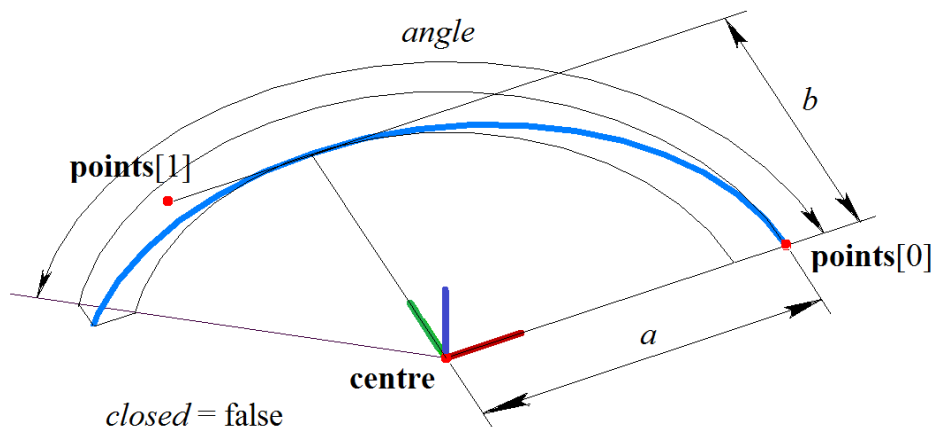


Рис. М.4.2.1.

Дуга эллипса описана в параграфе [O.4.4. Дуга эллипса MbArc3D](#).

Если множество точек **points** пусто, то длины полуосей эллипса *a* и *b* являются входными параметрами, а оси **axisX**, **axisY** и **axisZ** локальной системы координат эллипса совпадают с глобальными осями координат.

Параметр *closed* определяет циклическую замкнутость кривой. Если *closed=false*, то должен быть ненулевым параметр *angle*, который определяет угол раствора дуги эллипса в параметрических единицах.

В частном случае, когда *a=b*, рассматриваемый метод выполняет построение окружности (*closed=true* или *angle=0*) или дуги окружности (*closed=false* и *angle>0* и *angle<2π*).

М.4.3. Построение кривых по контрольным точкам

Метод
 MbResultType
SplineCurve (const SArray<[MbCartPoint3D](#)> & **points**,
 bool *closed*,
 MbeSpaceType *curveType*,
[MbCurve3D](#) *& **result**)

выполняет построение кривой заданного типа по указанному множеству контрольных точек.

Входными параметрами метода являются:

- **points** – множество контрольных точек,
- *closed* – флаг циклической замкнутости кривой,
- *curveType* – тип кривой.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле *action_curve3d.h*.

Параметр **points** содержит контрольные точки кривой. Параметр *closed* определяет циклическую замкнутость построенной кривой. Параметры *curveType* определяют тип кривой, от которого зависит форма кривой. Для разных типов создаваемой кривой требуется разное количество контрольных точек. В табл. М.4.3.1 приведено количество контрольных точек множества **points**, необходимое для создания кривой типа *curveType*.

Таблица М.4.3.1.

<i>curveType</i>	Тип кривой	Количество контрольных точек
pt_LineSegment	отрезок прямой	2 точки
pt_Arc	дуга окружности	3 точки
pt_Polyline	ломаная	2 и более точек
pt_Nurbs	NURBS кривая	2 и более точек
pt_Hermit	сплайн Эрмита	2 и более точек
pt_Bezier	кривая Безье	2 и более точек
pt_CubicSpline	кубический сплайн	2 и более точек

Для типа *curveType=st_LineSegment3D* будет построен отрезок прямой, начинающийся в точке **points[0]** и оканчивающийся в точке **points[1]**. Отрезок прямой описан в параграфе [O.4.3. Отрезок прямой MbLineSegment3D](#).

Для типа *curveType=st_Arc3D* при *closed=false* будет построена дуга окружности, начинающаяся в точке **points[0]**, проходящая через точку **points[1]** и оканчивающаяся в точке **points[2]**, рис. М.4.3.1.

При $closed=true$ будет построена окружность, проходящая через точки $points[0]$, $points[1]$, $points[2]$. Дуга окружности и эллипса описана в параграфе [О.4.4. Дуга эллипса MbArc3D](#).

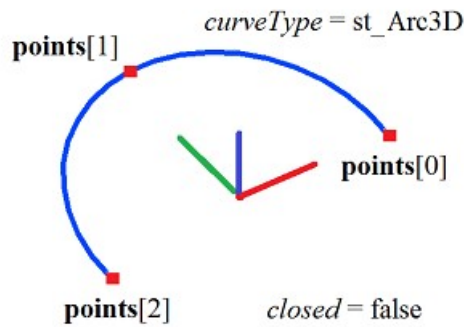


Рис. М.4.3.1.

Для типа $curveType=st_Polyline3D$ будет построена ломаная, проходящая через точки $points[0]$, $points[1]$, ..., $points[n]$, рис. М.4.3.2. При $closed=true$ будет построена циклически замкнутая ломаная, содержащая участок между точками $points[0]$ и $points[n]$. Ломаная линия описана в параграфе [О.4.5. Ломаная линия MbPolyline3D](#).

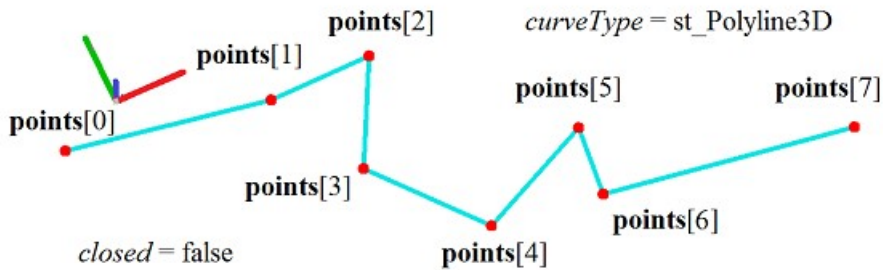


Рис. М.4.3.2.

Для типа $curveType=st_Nurbs3D$ будет построен неоднородный рациональный B -сплайн (NURBS) четвертого порядка, рис. М.4.3.3. Контрольные точки сплайна будут определены из условия прохождения сплайна через точки $points[0]$, $points[1]$, ..., $points[n]$. При $closed=true$ будет построен циклически замкнутый сплайн. NURBS-кривая описана в параграфе [О.4.6. NURBS-кривая MbNurbs3D](#).

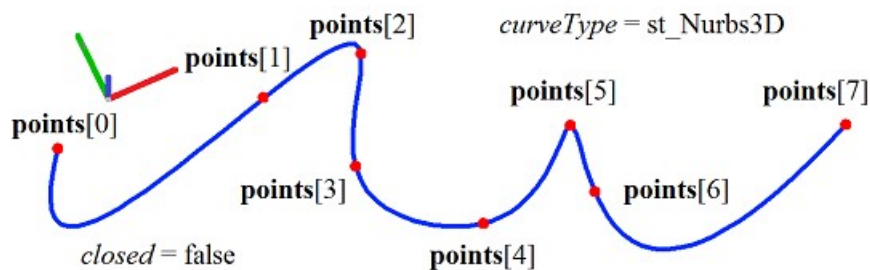


Рис. М.4.3.3.

Для типа $curveType=st_Hermit3D$ будет построена составная кривая, состоящая из гладко стыкующихся сплайнов Эрмита третьего порядка. Каждый из сплайнов Эрмита третьего порядка будет соединять соседние точки $points[i-1]$ и $points[i]$, рис. М.4.3.4. При $closed=true$ будет построена циклически замкнутая кривая, содержащая сплайн Эрмита между точками $points[0]$ и $points[n]$. Составной сплайн Эрмита третьей степени описан в параграфе [О.4.7. Кривая Эрмита MbHermit3D](#).

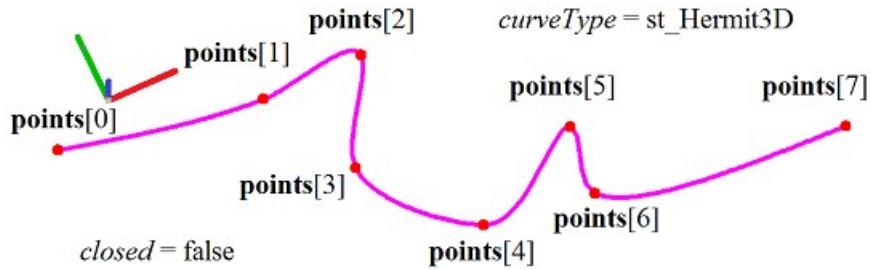


Рис. М.4.3.4.

Для типа *curveType*=st_Bezier3D будет построена составная кривая, состоящая из гладко стыкующихся сплайнов Безье третьего порядка. Каждый из сплайнов Безье третьего порядка будет соединять соседние точки **points**[*i*-1] и **points**[*i*], рис. М.4.3.5. При *closed*=true будет построена циклически замкнутая кривая, содержащая сплайн Безье между точками **points**[0] и **points**[*n*]. Составная кривая Безье третьей степени описана в параграфе [О.4.8. Составная кривая Безье MbBezier3D](#).

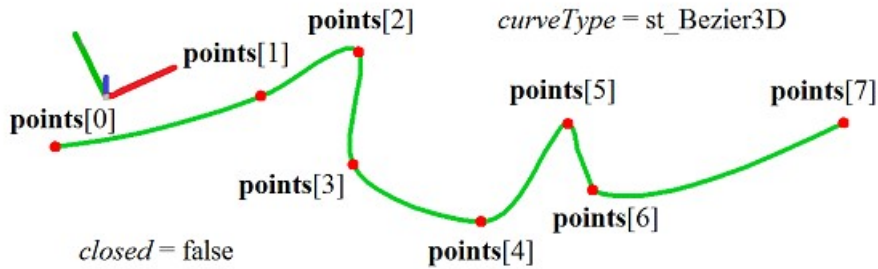


Рис. М.4.3.5.

Для типа *curveType*=st_CubicSpline3D будет построен кубический сплайн, проходящий через точки **points**[0], **points**[1],..., **points**[*n*], рис. М.4.3.6. При *closed*=true будет построена циклически замкнутая кривая, содержащая участок между точками **points**[0] и **points**[*n*]. Кубический сплайн описан в параграфе [О.4.9. Кубический сплайн MbCubicSpline3D](#).

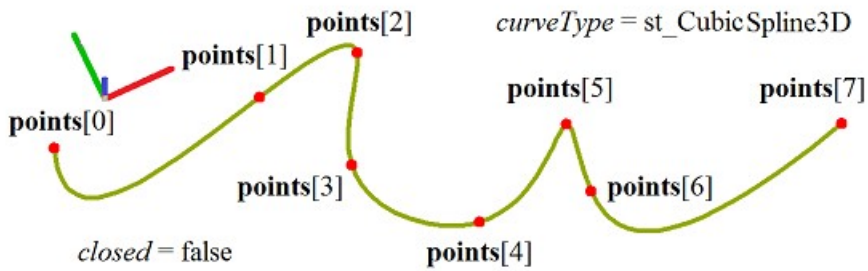


Рис. М.4.3.6.

Для сравнения на рис. М.4.3.7 приведены NURBS кривая, составной сплайн Эрмита, составная кривая Безье, кубический сплайн, построенные по одним и тем же контрольным точкам **points**[0], **points**[1],..., **points**[*n*] имеющие одну и ту же степень.

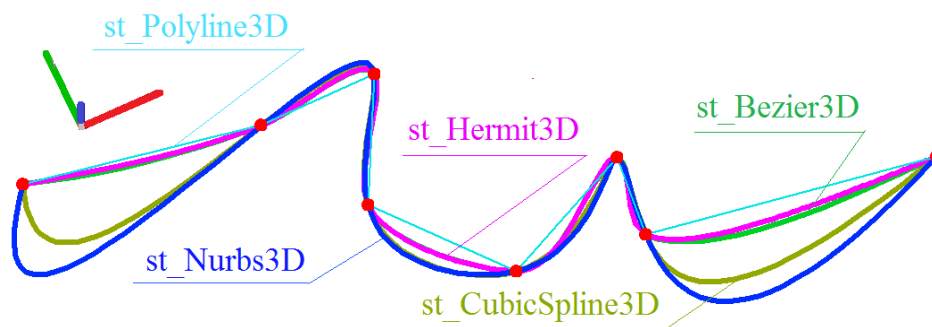


Рис. М.4.3.7.

Как можно видеть, кривые имеют различную форму.

М.4.4. Построение NURBS кривой

Метод

MbResultType

```
NurbsCurve ( const SArray<MbCartPoint3D> & points,
             const SArray<double> & weights,
             size_t degree,
             const SArray<double> & knots,
             bool closed,
             MbCurve3D *& result )
```

выполняет построение NURBS-кривой на заданном множестве контрольных точек.

Входными параметрами метода являются:

- **points** – множество контрольных точек,
- *weights* – множество весов контрольных точек,
- *degree* – порядок кривой (*B*-сплайнов),
- *knots* – множество параметрических узлов (узловой вектор),
- *closed* – флаг циклической замкнутости кривой.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_curve3d.h`.

Рассматриваемый метод строит NURBS-кривую (кривую на основе *B*-сплайнов), которая описана в параграфе [0.4.6. NURBS-кривая MbNurbs3D](#). Множество весов *weights* должно быть согласовано с множеством контрольных точек *points*. Порядок *degree* кривой не должен превосходить количество контрольных точек. Параметр *closed* определяет циклическую замкнутость построенной кривой. Узловой вектор *knots* представляет собой неубывающую последовательность действительных чисел и определяет область определения параметра кривой и форму кривой. Для *closed=false* узловой вектор должен содержать число элементов, равное числу контрольных точек плюс порядок кривой. Чтобы незамкнутая NURBS-кривая проходила через крайние контрольные точки, необходимо, чтобы первые *degree* элементов узлового вектора *knots* были равны между собой и последние *degree* элементов узлового вектора *knots* были равны между собой. Для *closed=true* узловой вектор должен содержать число элементов, равное числу контрольных точек плюс удвоенный порядок кривой и минус единица. На рис. М.4.4.1 приведена замкнутая NURBS-кривая четвертого порядка с равноотстоящими значениями узлов.

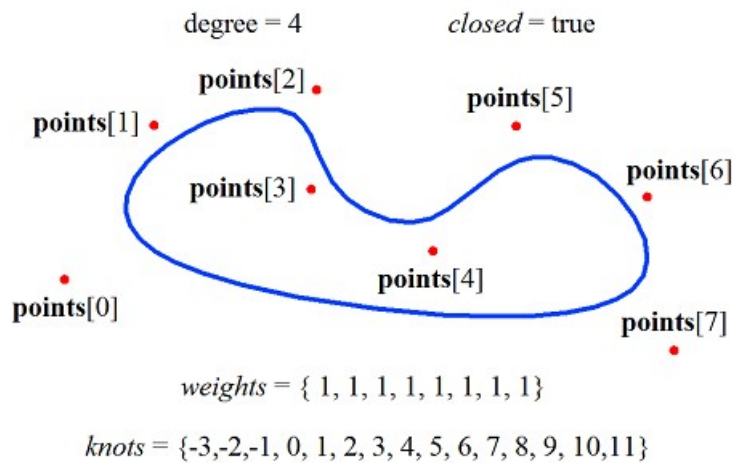


Рис. М.4.4.1.

На рис. М.4.4.2 приведены две незамкнутые NURBS-кривые четвертого порядка. Первая кривая имеет равноотстоящие значения узлов и совпадает с частью кривой, приведенной на рис. М.4.4.1. Вторая кривая имеет равные между собой первые degree элементов и последние degree элементов узлового вектора.

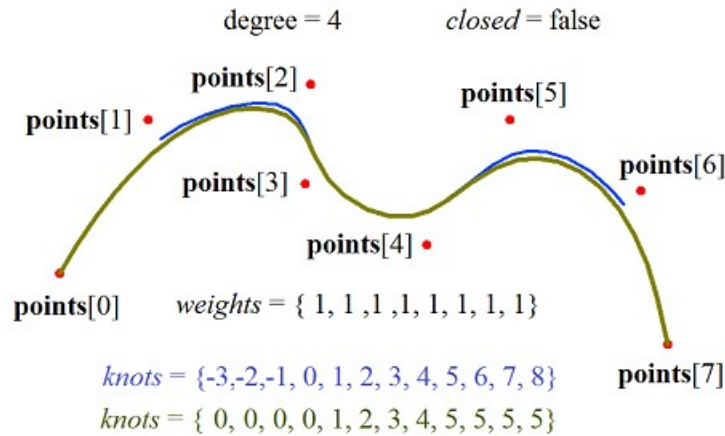


Рис. М.4.4.2.

На рис. М.4.4.3 приведены три незамкнутые NURBS-кривые четвертого порядка, имеющие различные значения весов $weights[2]$ и $weights[4]$ контрольных точек **points[2]** и **points[4]**.

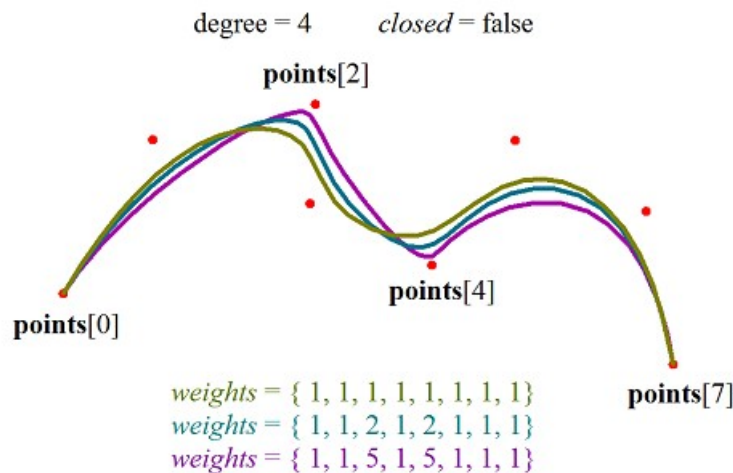


Рис. М.4.4.3.

На рис. М.4.4.4 приведены три незамкнутые NURBS-кривые различного порядка, построенные по одним и тем же контрольным точкам.

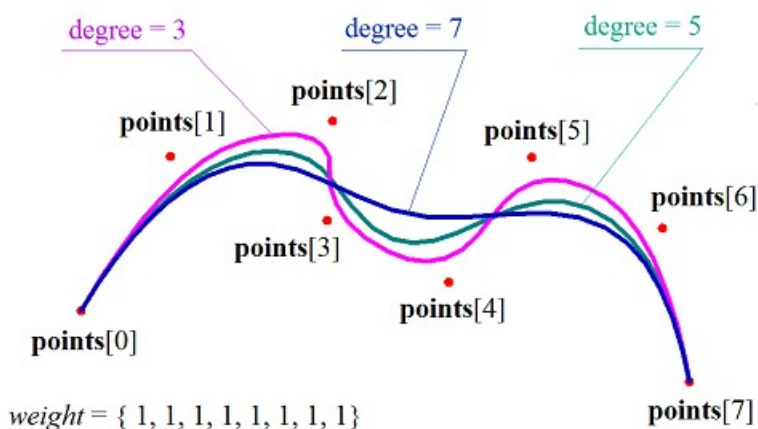


Рис. М.4.4.4.

На рис. М.4.4.5 приведена незамкнутая NURBS-кривая третьего порядка, по форме совпадающая с дугой окружности. Расстояние между контрольными точками `points[0]` и `points[1]` равно расстоянию между контрольными точками `points[1]` и `points[2]`, а вес `weights[1]` средней контрольной точки кривой равен весу крайних контрольных точек, умноженному на косинус половины угла дуги.

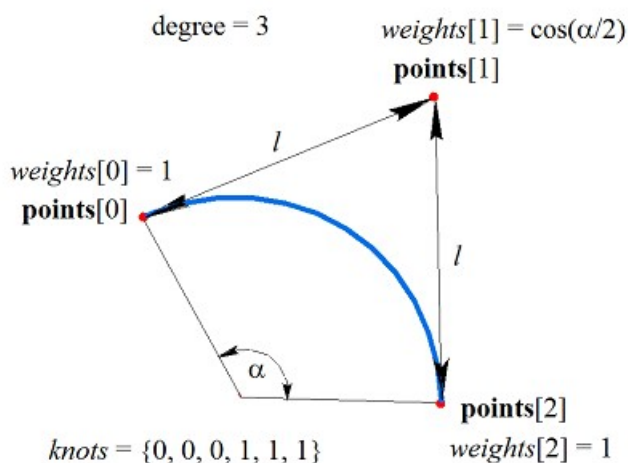


Рис. М.4.4.5.

Метод
`MbResultType`
`NurbsCopy` (const `MbCurve3D` & `curve`,
`MbCurve3D` *& `result`)

выполняет построение NURBS-копии заданной кривой.

Входным параметром метода является исходная кривая `curve`.

Выходным параметром метода является построенная кривая `result`.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve3d.h`.

Рассматриваемый метод построит NURBS-кривую, форма которой копирует форму исходной кривой. Для большинства типов кривых NURBS-копия по форме полностью совпадает с исходной кривой. В случае невозможности точно воспроизвести форму исходной кривой NURBS-копия аппроксимирует исходную кривую с погрешностью, не превышающей 0,0001.

М.4.5. Построение выпуклой равносторонней ломаной

Метод
MbResultType
RegularPolygon (const [MbCartPoint3D](#) & **centre**,
const [MbCartPoint3D](#) & **point**,
const [MbVector3D](#) & **axisZ**,
size_t *vertexCount*,
bool *describe*,
[MbCurve3D](#) *& **result**)

выполняет построение замкнутой ломаной линии, представляющей собой правильный многоугольник, вписанный в обозначенную окружность или описанный около окружности.

Входными параметрами метода являются:

- **centre** – центр окружности, описанной или вписанной в равносторонний многоугольник,
- **point** – точка на окружности,
- **axisZ** – вектор, перпендикулярный плоскости окружности,
- *vertexCount* – количество вершин многоугольника,
- *describe* – флаг описанной или вписанной окружности.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_curve3d.h`.

Параметры **centre**, **point** и **axisZ** определяют окружность, в которую будет вписан (*describe=false*) многоугольник с числом вершин *vertexCount* или вокруг которой будет описан (*describe=true*) многоугольник с числом вершин *vertexCount*. Центр окружности будет расположен в точке **centre**, окружность пройдет через точку **point**, ось окружности будет параллельна вектору **axisZ**. Параметр *vertexCount* определяет число вершин правильного многоугольника. Построенная кривая будет замкнутой ломаной линией, описанной в параграфе [O.4.5. Ломаная линия MbPolyline3D](#). На рис. М.4.5.1 приведен вписанный в окружность правильный многоугольник, а на рис. М.4.5.2 приведен описанный вокруг окружности правильный многоугольник.

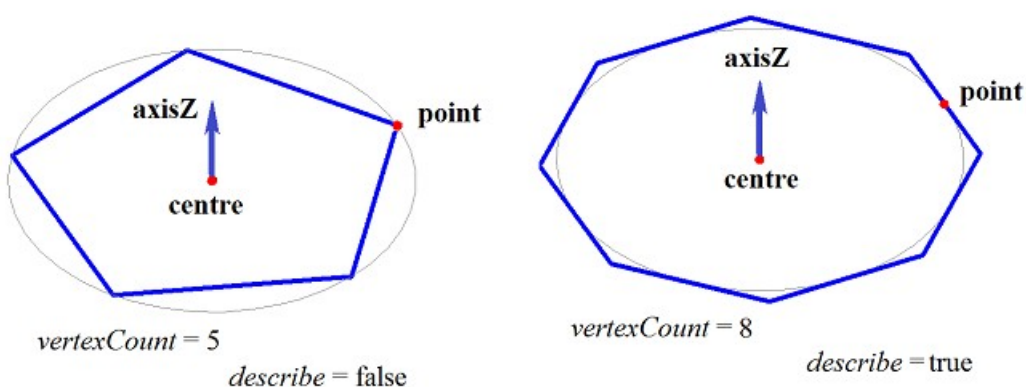


Рис. М.4.5.1.

Рис. М.4.5.2.

При $vertexCount \leq 1$ рассматриваемый метод построит окружность с центром в точке **centre**, проходящую через точку **point**. При $vertexCount = 2$ рассматриваемый метод построит ломаную линию с одним участком, начинающуюся в точке **centre** и оканчивающуюся в точке **point**.

М.4.6. Построение спиралей

Метод
MbResultType

SpiralCurve (const [MbPlacement3D](#) & **place**,
double *radius*,
double *step*,
[MbCurve](#) & *lawCurve*,
bool *spiralAxis*,
[MbCurve3D](#) *& **result**)

выполняет построение спирали переменного радиуса или спирали с искривлённой осью.

Входными параметрами метода являются:

- **place** – локальная система координат спирали,
- *radius* – радиус спирали,
- *step* – шаг спирали,
- *lawCurve* – формообразующая двумерная кривая,
- *spiralAxis* – режим формообразования.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve3d.h`.

Если *spiralAxis*=false, то *lawCurve* определяет изменение радиуса спирали. В данном случае ось спирали совпадает с осью **axisZ** локальной системы координат **place**, кривая *lawCurve* будет помещена в плоскость ZX локальной системы координат спирали и опишет закон изменения радиуса спирали. Первая координата (*x*) каждой двумерной точки кривой *lawCurve* будет откладываться по оси **axisZ** локальной системы координат спирали, вторая координата (*y*) каждой двумерной точки кривой *lawCurve* будет откладываться по оси **axisX** локальной системы координат спирали, рис. М.4.6.1.

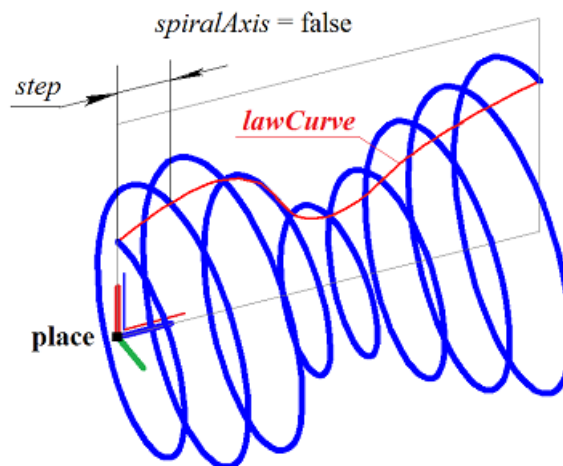


Рис. М.4.6.1.

Вторая координата (*y*) двумерной кривой *lawCurve* определит радиус спирали как функцию первой координаты (*x*) кривой. Вторая координата (*y*) каждой двумерной точки кривой *lawCurve* должна быть больше нуля, чтобы кривая *lawCurve* не пересекала ось **axisZ**. Спираль переменного радиуса описана в параграфе [0.4.15. Спираль переменного радиуса MbCurveSpiral](#). Параметр *step* описывает шаг спирали, параметр *radius* не используется.

Если *spiralAxis*=true, то *lawCurve* определяет ось спирали. В данном случае кривая *lawCurve* будет помещена в плоскость ZX локальной системы координат спирали и опишет ось спирали. Первая координата (*x*) каждой двумерной точки кривой *lawCurve* будет откладываться по оси **axisZ** локальной системы координат спирали, вторая координата (*y*) каждой двумерной точки кривой *lawCurve* будет откладываться по оси **axisX** локальной системы координат спирали, рис. М.4.6.2.

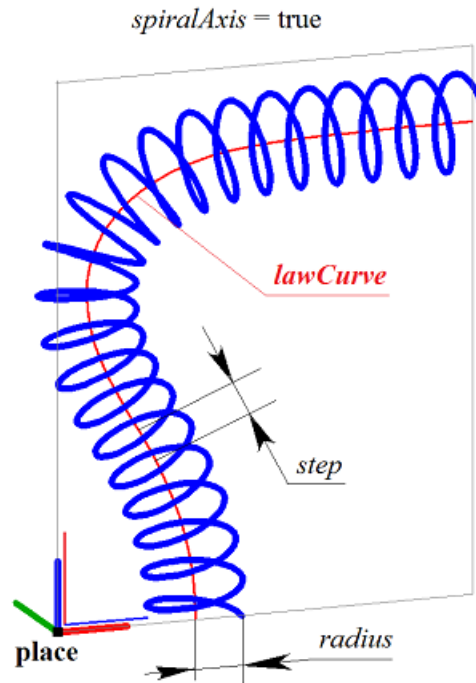


Рис. М.4.6.2.

Радиус кривизны двумерной кривой *lawCurve* в каждой точке должен быть больше радиуса спирали *radius*, чтобы спираль не пересекала сама себя. Спираль с криволинейной осью описана в параграфе [0.4.16. Спираль с криволинейной плоской осью MbCrookedSpiral](#). Параметр *step* описывает шаг спирали вдоль её оси.

Метод

MbResultType

SpiralCurve (const [MbCartPoint3D](#) & **point0**,
const [MbCartPoint3D](#) & **point1**,
const [MbCartPoint3D](#) & **point2**,
double *radius*,
double *step*,
double *angle*,
[MbCurve](#) * *lawCurve*,
bool *spiralAxis*,
[MbCurve3D](#) *& **result**)

выполняет построение или конической спирали, или спирали переменного радиуса, или спирали с криволинейной осью.

Входными параметрами метода являются:

- **point0** – начало локальной системы координат спирали,
- **point1** – точка на оси Z локальной системы координат спирали,
- **point2** – точка, определяющая ось X локальной системы координат спирали,
- *radius* – радиус спирали,
- *step* – шаг спирали,
- *angle* – угол конической спирали,
- *lawCurve* – формообразующая двумерная кривая, может быть нулём,
- *spiralAxis* – режим формообразования.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_curve3d.h`.

Точки **point0**, **point1** и **point2** определяют локальную систему координат спирали. Начало локальной системы координат спирали будет расположено в точке **point0**, ось **axisZ** локальной системы координат будет направлена из точки **point0** в точку **point1**, ось **axisX** локальной системы

координат будет ортогональна оси **axisZ** и направлена из точки **point0** в сторону точки **point2**. Точки **point0**, **point1** и **point2** не должны совпадать или лежать на одной прямой.

Если **lawCurve=0**, то метод построит коническую спираль с осью **axisZ** локальной системы координат, с шагом спирали **step**, углом конусности **angle**. Радиус в начале спирали будет равен **radius**, рис. М.4.6.3.

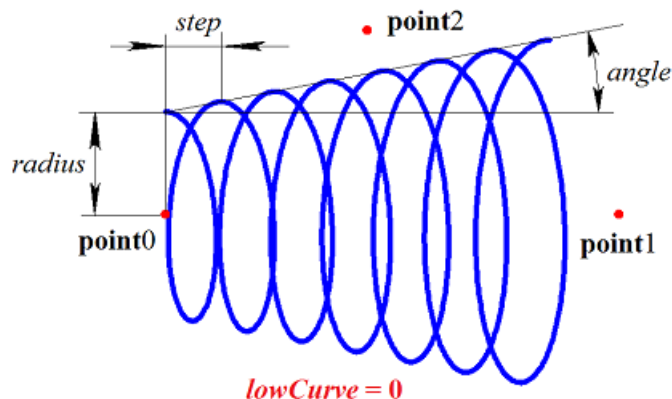


Рис. М.4.6.3.

При **angle=0** будет построена цилиндрическая спираль. Параметр **spiralAxis** не будет использован.

Если **lawCurve** не равна нулю и **spiralAxis=false**, то метод построит спираль переменного радиуса описанным выше образом в локальной системе координат, определённой точками **point0**, **point1** и **point2**, приведенную на рис. М.4.6.1. Параметр **lawCurve** определит изменение радиуса спирали.

Если **lawCurve** не равна нулю и **spiralAxis=true**, метод построит спираль с криволинейной осью описанным выше образом в локальной системе координат, определённой точками **point0**, **point1** и **point2**, приведенную на рис. М.4.6.2. Параметр **lawCurve** определит ось спирали.

М.4.7. Построение составной кривой

Метод
 MbResultType
CreateContour ([MbCurve3D](#) & **curve**,
 [MbContour3D](#) *& **result**)

выполняет построение составной кривой на базе одной исходной кривой.

Входным параметром метода является исходная кривая **curve**.

Выходным параметром метода является построенная кривая **result**.

При удачной работе метод возвращает **rt_Success**, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле **action_curve3d.h**.

Метод создает составную кривую **result** на базе исходной кривой **curve**. Если исходная кривая **curve** является составной кривой, то в кривую **result** будут положены составляющие элементы исходной кривой. Составная кривая описана в параграфе [О.4.18. Контур MbContour3D](#).

Метод
 MbResultType
AddCurveToContour ([MbCurve3D](#) & **curve**,
 [MbCurve3D](#) & **contour**,
 bool **toEnd**)

модифицирует составную кривую путём добавления кривой.

Входными параметрами метода являются:

- **curve** – добавляемая кривая,
- **contour** – модифицируемая составная кривая,
- **toEnd** – флаг места добавления кривой.

Выходным параметром метода является модифицируемая кривая **contour**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve.h`.

Метод модифицирует составную кривую **contour** путём добавления кривой **curve** в начало или конец составной кривой. Если `toEnd=true`, то кривая **curve** будет добавлена в конец составной кривой **contour**, если `toEnd=false`, то кривая **curve** будет добавлена в начало составной кривой **contour**. Точки стыковки модифицируемой кривой и добавляемой кривой должны совпадать, рис. М.4.7.1.

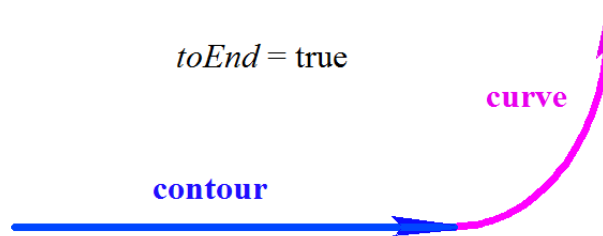


Рис. М.4.7.1.

Если добавляемая кривая **curve** является составной кривой, то в кривую **contour** будут добавлены составляющие элементы добавляемой кривой.

Метод

`MbResultType`

```
CreateContours ( RPAArray<MbCurve3D> & curves,  
                  double epsilon,  
                  RPAArray<MbContour3D> & result,  
                  bool onlySmoothConnected = false )
```

выполняет построения составных кривых из множества исходных кривых.

Входными параметрами метода являются:

- **curves** – множество исходных кривых,
- *epsilon* – радиус захвата для стыковки исходных кривых,
- *onlySmoothConnected* – флаг использования только гладкой стыковки исходных кривых.

Выходным параметром метода является множество построенных кривых **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve3d.h`.

Метод на базе исходных кривых **curves** создает составные кривые и складывает их в множество **result**. Если флаг *onlySmoothConnected=true*, то все построенные составные кривые будут гладкими, то есть в точках стыковки кривые будут иметь одинаково направленные касательные, но длина производной в точках стыковки может меняться скачком. Если какая-либо исходная кривая множества **curves** является составной кривой, то в составную кривую будут добавлены составляющие элементы такой исходной кривой.

М.4.8. Построение каркаса

Метод

`MbResultType`

```
WireFrame ( const MbCurve3D & curve,  
             const MbName & name,  
             SimpleName mainName,  
             MbWireFrame *& result )
```

создаёт каркас на базе кривой.

Входными параметрами метода являются:

- **curve** – исходная кривая,
- *name* – имя кривой,
- *mainName* – главное имя каркаса.

Выходным параметром метода является построенный каркас **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve3d.h`.

Метод на базе кривой **curve** создаёт ребро `MbEdge`, которое служит элементом каркаса **result**. Каркас `MbWireFrame` описан в параграфе [O.8.3. Проволочный каркас MbWireFrame](#).

Метод

`MbResultType`

```
WireFrame ( const RPAArray<MbCurve3D> & curves,  
             const RPAArray<MbName> & names,  
             SimpleName mainName,  
             MbWireFrame *& result )
```

создаёт каркас на базе множества кривых.

Входными параметрами метода являются:

- **curves** – множество кривых,
- **names** – множество имён кривых,
- **mainName** – главное имя каркаса.

Выходным параметром метода является построенный каркас **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_curve3d.h`.

Метод на базе каждой кривой множества **curve** создаёт ребро `MbEdge`, далее из рёбер создается каркас **result**. Ребра каркаса стыкуются в вершинах, каждое ребро и каждая вершина наделены атрибутами. Каркас предоставляет информацию о стыкующихся в общих вершинах ребрах, каркас наделён атрибутами и журналом построения. Каркас `MbWireFrame` описан в параграфе [O.8.3. Проволочный каркас MbWireFrame](#).

M.4.9. Построение проекции кривой на поверхность

Метод

`MbResultType`

```
CurveProjection ( const MbSurface & surface,  
                 const MbCurve3D & curve,  
                 MbVector3D * direction,  
                 bool createExact,  
                 bool truncateByBounds,  
                 RPAArray<MbCurve3D> & result,  
                 VERSION version = Math::DefaultMathVersion() );
```

выполняет построение проекций кривой на заданную поверхность при нормальном проецировании или при параллельном проецировании.

Входными параметрами метода являются:

- **surface** – поверхность, на которую выполняется проецирование,
- **curve** – проецируемая кривая,
- **direction** – вектор направления проецирования, может быть ноль,
- *createExact* – флаг точности построенных кривых,
- *truncateByBounds* – флаг обрезки проекций границами поверхности,
- **version** – версия построения.

Выходным параметром метода является множество построенных кривых **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface_curve.h`.

Если **direction**=0, то метод создает нормальную проекцию кривой **curve** на поверхность **surface**, при которой вектор, построенный из точки проекции к соответствующей точке проецируемой кривой, направлен по нормали к поверхности.

Если задан вектор **direction**, то метод создает параллельную проекцию кривой **curve** на поверхность **surface**, при которой вектор, построенный из точки проекции к соответствующей точке проецируемой кривой, параллелен вектору **direction**.

Кривые **result** являются кривыми на поверхности MbSurfaceCurve, представленными в параграфе [О.4.20. Кривая на поверхности MbSurfaceCurve](#). Если *createExact=false*, то в общем случае двумерные кривые, фигурирующие в кривых на поверхности **result**, будут сплайновыми кривыми, проходящими через конечное число точек, являющихся проекциями отдельных точек проецируемой кривой. Если *createExact=true*, то в общем случае двумерные кривые, фигурирующие в кривых на поверхности **result**, будут проекционными кривыми MbProjCurve, которые точно описывают проекции пространственных кривых на поверхность. В частных случаях кривые **result** точно представляют проекцию кривой **curve**. Если задан вектор **direction**, то считается, что параметр *createExact=false*.

Если параметр *truncateByBounds=false*, то кривые **result** расположены внутри параметрического прямоугольника, охватывающего область определения параметров поверхности **surface**. В случаях, когда область определения параметров поверхности не совпадает с охватывающим её габаритным прямоугольником, кривые **result** могут выходить за края поверхности **surface**.

Если параметр *truncateByBounds=true*, то кривые **result** полностью расположены внутри поверхности **surface**. На рис. М.4.9.1 приведена нормальная проекция кривой на поверхность.

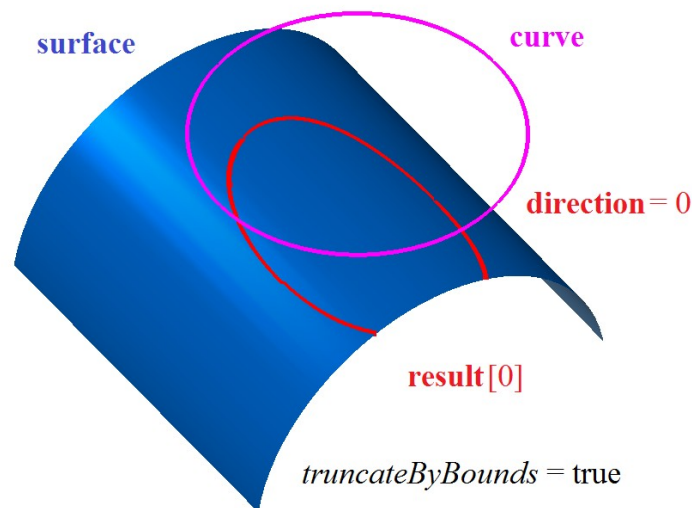


Рис. М.4.9.1.

Если параметр *truncateByBounds=false*, то кривые **result** могут выходить за пределы поверхности **surface**. На рис. М.4.9.2 приведена параллельная проекция кривой на поверхность.

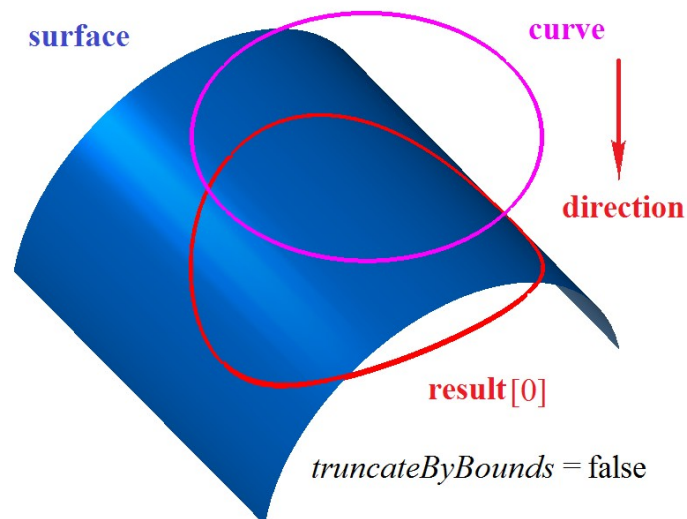


Рис. М.4.9.2.

М.4.10. Построение кривых пересечения поверхностей

Метод

MbResultType

IntersectionCurve (constMbSurface & surface1,
 constMbSurface & surface2,
 const MbSNameMaker & names,
 MbWireFrame *& result)

выполняет построение кривых пересечения двух поверхностей.

Входными параметрами метода являются:

- **surface1** – первая поверхность,
- **surface2** – вторая поверхность,
- **names** – именованье элементов результата построения.

Выходным параметром метода является построенный каркас **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface_curve.h`.

Две поверхности могут пересекаться по нескольким кривым. Рассматриваемый метод строит все кривые пересечения и объединяет их в каркас. Кривые каркаса являются кривыми пересечения поверхностей MbSurfaceIntersectionCurve, описанными в параграфе [0.4.24. Кривая пересечения поверхностей MbSurfaceIntersectionCurve](#). Параметр `names` обеспечивает именованье рёбер построенного каркаса. На рис. М.4.10.1 приведен результат пересечения поверхностей.

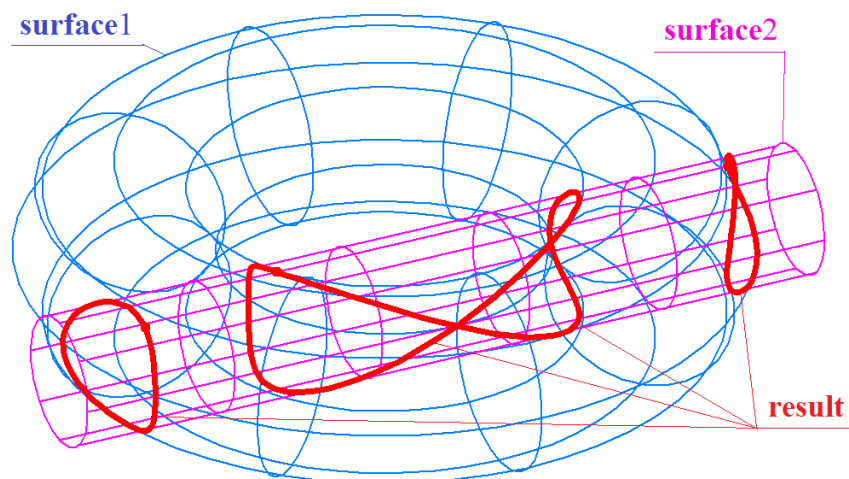


Рис. М.4.10.1.

М.4.11. Построение силуэтных кривых

Метод

MbResultType

SilhouetteCurve (const [MbFace](#) & **face**,
 const [MbVector3D](#) & **eye**,
 bool *perspective*,
 RPAarray<[MbCurve3D](#)> & **result**)

выполняет построение силуэтных кривых грани при параллельном или перспективном проецировании.

Входными параметрами метода являются:

- **face** – грань,
- **eye** – направление взгляда или вектор точки наблюдения,
- *perspective* – флаг перспективного проецирования.

Выходным параметром метода является множество построенных кривых **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface_curve.h`.

Силуэтные кривые лежат на грани и отделяют видимую из точки наблюдения часть грани от её невидимой части. Для грани может существовать несколько силуэтных кривых. Для разных направлений взгляда и разных точек наблюдения набор силуэтных кривых разный.

Если *perspective=false*, то вектор **eye** определяет направления взгляда, которое постоянно для всех точек грани. Если *perspective=true*, то вектор **eye** определяет положение точки наблюдения, а не направление взгляда, так как направление взгляда для разных точек грани будет разным. Для параллельных проекций грани строятся силуэтные кривые с *perspective=false*. Для перспективных проекций грани строятся силуэтные кривые с *perspective=true*.

В частных случаях силуэтные кривые могут совпадать с краями грани. Силуэтные кривые, совпадающие с краями грани, не строятся, так как они совпадут с ребрами грани.

На рис. М.4.11.1 приведен пример силуэтных кривых грани при параллельном проецировании.

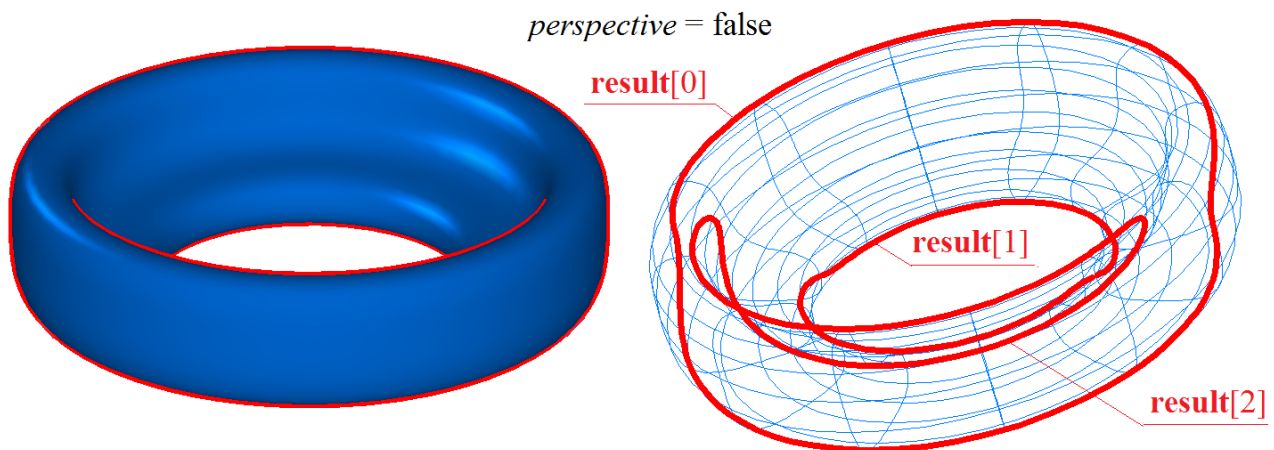


Рис. М.4.11.1.

На рис. М.4.11.2 приведена пример силуэтных кривых грани при перспективном проецировании с точкой наблюдения вблизи грани.

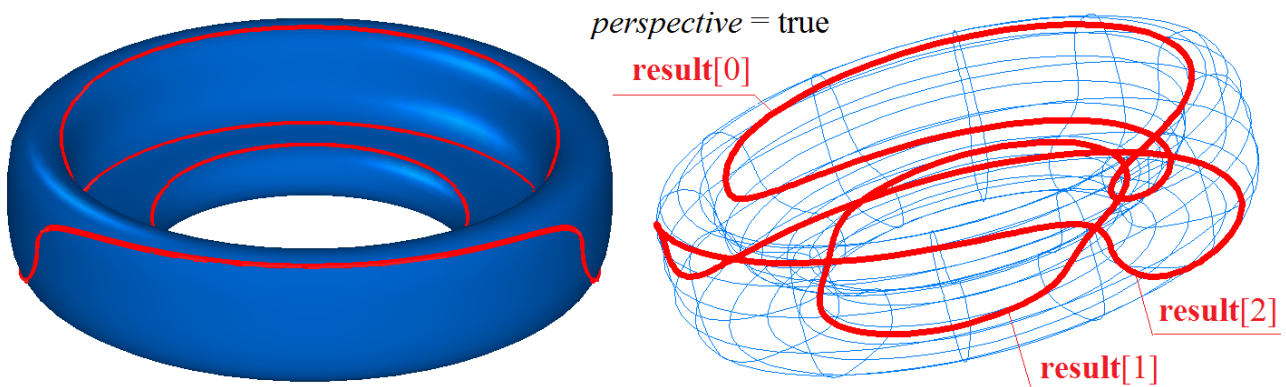


Рис. М.4.11.2.

Силуэтная кривая описана в параграфе [0.4.21. Силуэтная кривая MbSilhouetteCurve](#).

Метод

MbResultType

SilhouetteCurve (const [MbFace](#) & **face**,
const MbAxis3D& **axis**,
RPAArray<[MbCurve3D](#)> & **result**)

выполняет построение силуэтных кривых грани при её вращении вокруг заданной оси.

Входными параметрами метода являются:

- **face** – грань,
- **axis** – ось вращения.

Выходным параметром метода является множество построенных кривых **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface_curve.h`.

Для построения токарного сечения требуется строить набор силуэтных кривых вращения грани. При вращении грани вокруг оси можно считать, что направление взгляда для точек грани ортогонально оси вращения и отрезку, соединяющему точку грани и проекцию этой точки на ось.

В частных случаях силуэтные кривые вращения могут совпадать с краями грани. Силуэтные кривые, совпадающие с краями грани, не строятся, так как они совпадут с ребрами грани.

На рис. М.4.11.3 приведен пример силуэтных кривых при вращении грани вокруг оси.

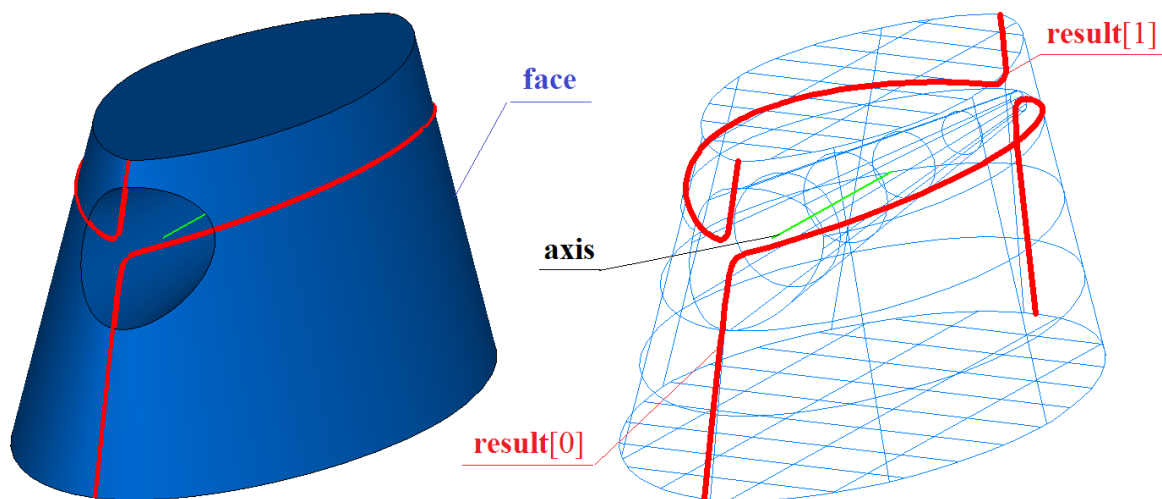


Рис. М.4.11.3.

М.4.12. Построение кривой сопряжения кривых

Метод

MbResultType

FilletCurve (const [MbCurve3D](#) & **curve1**,
double & *t1*,
double & *w1*,
const [MbCurve3D](#) & **curve2**,
double & *t2*,
double & *w2*,
double & *radius*,
bool *sense*,
bool & *unchanged*,
const MbConnectingType *type*,
const MbSNameMaker & *names*,
MbElementarySurface *& **surface**,
[MbWireFrame](#) *& **result**)

выполняет построение кривой, гладко соединяющей две кривые и лежащей на цилиндрической поверхности, касающейся соединяемых кривых.

Входными параметрами метода являются:

- **curve1** – первая кривая,
- *t1* – параметр точки сопряжения первой кривой,
- *w1* – параметр крайней точки первой кривой.
- **curve2** – вторая кривая,
- *t2* – параметр точки сопряжения второй кривой,
- *w2* – параметр крайней точки второй кривой,
- *radius* – радиус сопряжения,
- *sense* – направление построенного сопряжения,
- *type* – способ сопряжения,
- *names* – именовател построенного сопряжения.

Выходными параметрами метода являются построенный каркас **result**, цилиндрическая поверхность **surface**, касающаяся кривых, параметры *t1* и *t2* точек сопряжения кривых, параметры *w1* и *w2* крайних точек сопрягаемых кривых, радиус *radius* сопряжения, флаг *unchanged* частного случая построенного сопряжения.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface_curve.h`.

Метод даёт возможность выполнить сопряжение двух кривых пятью способами. При `type=ft_Fillet` будет построена кривая на цилиндрической поверхности (в частном случае дуга окружности), касающейся в точках сопряжения кривых `curve1` и `curve2`. Для способа `type=ft_Fillet` `radius` может быть как входным, так и выходным параметром. Если `radius` является входным параметром (больше нуля), то параметры `t1` и `t2` будут рассчитаны, если `radius` не задан (равен нулю), то `t1` и `t2` должны быть входными параметрами. В данном способе возвращаемым параметром будет цилиндрическая поверхность `surface`, на которой построена кривая сопряжения.

Если кривые `curve1` и `curve2` являются кривыми на общей поверхности, то они могут быть сопряжены кривой пересечения общей поверхности с цилиндрической поверхностью. В этом случае параметр `type=ft_OnSurface`. Параметр `radius` может устанавливать радиус цилиндрической поверхности.

При `type=ft_Spline` будет построена NURBS-кривая, сопрягающая кривую `curve1` в точке с параметром `t1` и кривую `curve2` в точке с параметром `t2`. Параметр `radius` может устанавливать натяжение NURBS-кривой в точках сопряжения.

При `type=ft_Double` кривые `curve1` и `curve2` будут сопряжены двумя дугами радиуса `radius`. Точки сопряжения будут располагаться на краях кривых. При необходимости между дугами может быть вставлен отрезок прямой.

При `type=ft_Bridge` кривые `curve1` и `curve2` будут сопряжены соединительной кривой `MbBridgeCurve3D`, представленной в параграфе [0.4.17. Соединительная кривая MbBridgeCurve3D](#). Соединительная кривая представляет собой кубический сплайн Эрмита, построенный по двум крайним точкам и производным кривой в этих точках. В этом случае параметр `radius` не используется, а `t1` и `t2` должны быть входными параметрами.

Параметр `t1` определяет точку параметрической области кривой `curve1`, в которой кривая `curve1` сопрягается с построенной кривой. Для циклически замкнутой кривой `curve1` параметр `w1` совпадает с `t1`. Для незамкнутой кривой параметр `w1` в совокупности с параметром `t1` определяет участок кривой `curve1`, гладко переходящий в построенную кривую. Параметры `t1` и `w1` могут использоваться для усечения кривой `curve1`.

Параметр `t2` определяет точку параметрической области кривой `curve2`, в которой кривая `curve2` сопрягается с построенной кривой. Для циклически замкнутой кривой `curve2` параметр `w2` совпадает с `t2`. Для незамкнутой кривой параметр `w2` в совокупности с параметром `t2` определяет участок кривой `curve2`, гладко переходящий в построенную кривую. Параметры `t2` и `w2` могут использоваться для усечения кривой `curve2`.

Параметр `radius` определяет радиус кривой сопряжения (для случаев не равенства `type` значениям `ft_Spline` и `ft_Bridge`). Если `radius` больше нуля, то он является входным параметром. В этом случае будут найдены значения параметров `t1` и `t2` точек сопряжения. Если `radius` меньше или равен нулю, то он является выходным параметром. В этом случае должны быть заданы значения `t1` и `t2` параметров точек на сопрягаемых кривых.

Параметр `sense` определяет направление построенной кривой. При `sense=true` построенная кривая направлена от `curve1` к `curve2`. При `sense=false` построенная кривая направлена от `curve2` к `curve1`.

Параметр `unchanged` сообщает о изменениях в построенной кривой. Если `unchanged=true`, то была построена дуга окружности указанного радиуса. В общем случае возвращается значение `unchanged=false`.

Параметр `names` обеспечивает именование рёбер построенного каркаса.

На рис. М.4.12.1 приведен пример скругления кривых для способа `type=ft_Fillet`.

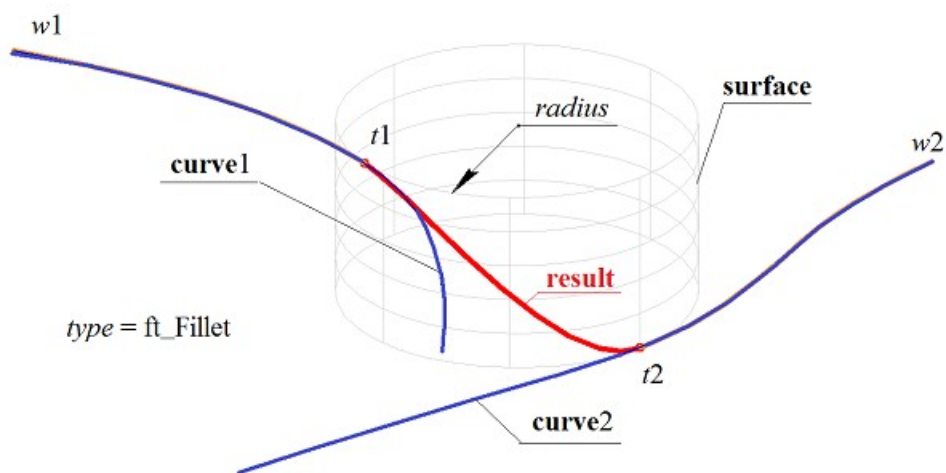


Рис. М.4.12.1.

На рис. М.4.12.2 приведен пример скругления кривых для способа $type=ft_Double$.

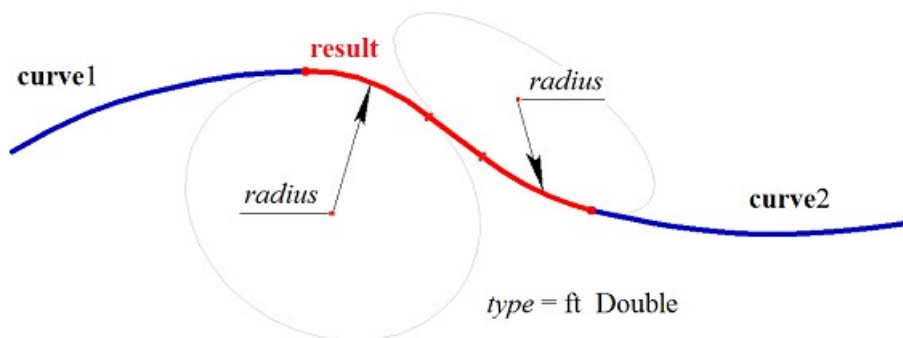


Рис. М.4.12.2.

На рис. М.4.12.3 приведен пример скругления кривых для способа $type=ft_OnSurface$.

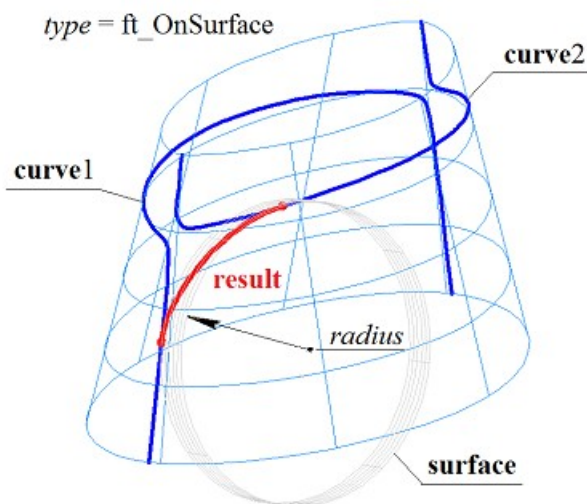


Рис. М.4.12.3.

Метод **FilletCurve** добавляет в журнал построенного каркаса строитель `MbConnectingCurveCreator`, который содержит все необходимые данные для построения сопряжения. Строитель `MbConnectingCurveCreator` объявлен в файле `cr_connecting_curve.h`.

М.5. МЕТОДЫ ПОСТРОЕНИЯ ПОВЕРХНОСТЕЙ

Поверхности описывают форму моделируемых объектов. На базе поверхностей строятся грани, из которых состоят твёрдые тела. В определенных случаях поверхности выступают в роли вспомогательных объектов для построения элементов геометрической модели. Все поверхности являются наследниками класса MbSurface и представлены в главе [0.5. ПОВЕРХНОСТИ](#). Поверхности могут быть построены непосредственным вызовом соответствующих конструкторов или методами, приведёнными в данном параграфе.

М.5.1. Построение элементарной поверхности

Метод

MbResultType

```
ElementarySurface ( const MbCartPoint3D & point0,  
                    const MbCartPoint3D & point1,  
                    const MbCartPoint3D & point2,  
                    MbeSpaceType surfaceType,  
                    MbSurface *& result )
```

выполняет построение элементарной поверхности.

Входными параметрами метода являются:

- **point0** – точка, определяющая начало локальной системы координат поверхности,
- **point1** – точка, определяющая направление оси локальной системы и радиус поверхности,
- **point2** – точка, определяющая направление оси локальной системы,
- *surfaceType* – тип поверхности.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Параметр *surfaceType* определяет тип создаваемой поверхности. Контрольные точки **point0**, **point1** и **point2** определяют локальную систему координат и размеры элементарной поверхности. Точка **point0** определяет начало локальной системы координат. В табл. М.5.1.1 приведены тип поверхности и ось локальной системы координат и величину, которую определяет точка **point1** для создания поверхности типа *surfaceType*.

Таблица М.5.1.1.

<i>surfaceType</i>	Тип поверхности	Точка point1 определяет ось
<code>st_Plane</code>	плоскость	axisX
<code>st_CylinderSurface</code>	цилиндрическая поверхность	axisZ , высоту
<code>st_ConeSurface</code>	коническая поверхность	axisZ , высоту
<code>st_SphereSurface</code>	сферическая поверхность	axisZ
<code>st_TorusSurface</code>	поверхность тора	AxisX , большой радиус

Точка **point1** определяет высоту цилиндра, высоту конуса, большой радиус тора. Точка **point2** определяет радиус цилиндра, радиус конуса и угол конусности, радиус сферы, малый радиус тора.

При построении плоскости точки **point0**, **point1** и **point2** определяют локальную систему координат плоскости с началом в точке **point0**. Ось **axisX** локальной системы координат плоскости направлена из точки **point0** в точку **point1**, ось **axisY** локальной системы координат плоскости ортогональна оси **axisX** и направлена в сторону точки **point2**. Область определения первого параметра равна удвоенному расстоянию между точками **point0** и **point1**. Область определения

второго параметра равна удвоенному расстоянию между точками **point0** и проекцией на ось **axisY** точки **point2**, рис. М.5.1.1.

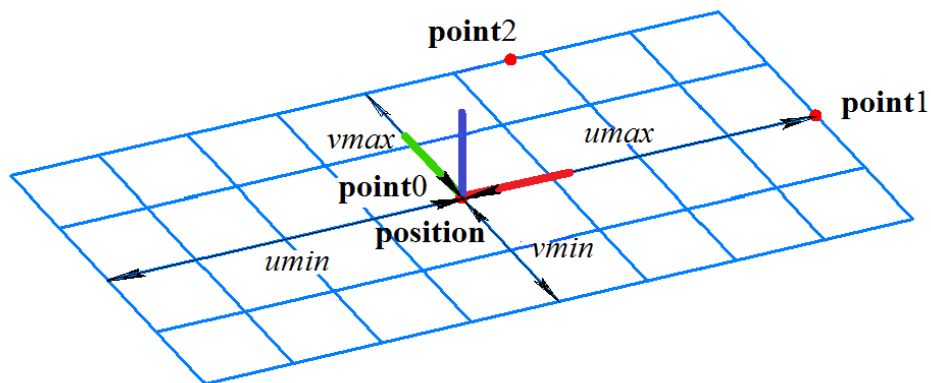


Рис. М.5.1.1.

При построении цилиндрической поверхности точка **point0** определяет центр нижнего основания цилиндра, в котором располагается начало локальной системы координат. Точка **point1** определяет центр верхнего основания цилиндра. Ось **axisZ** локальной системы координат поверхности направлена из точки **point0** в точку **point1**. Точка **point2** вместе с предыдущими точками определяет плоскость расположения осей **axisX** и **axisZ** локальной системы координат поверхности. Расстояние между точками **point0** и **point1** определяет высоту цилиндрической поверхности, расстояние от оси **axisZ** до точки **point2** определяет радиус цилиндрической поверхности, рис. М.5.1.2.

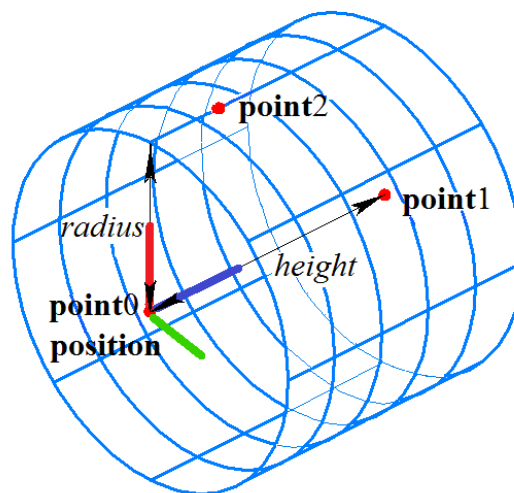


Рис. М.5.1.2.

Цилиндрическая поверхность описана в параграфе [0.5.3. Цилиндрическая поверхность MbCylinderSurface](#).

При построении конической поверхности точка **point0** определяет вершину конуса, в которой располагается начало локальной системы координат. Точка **point1** определяет центр основания конуса и направление оси **axisZ** локальной системы координат конуса. Точка **point2** вместе с предыдущими точками определяет плоскость расположения осей **axisX** и **axisZ** локальной системы координат поверхности. Расстояние между точками **point0** и **point1** определяет высоту конической поверхности. Точка **point2** определяет угол конусности, из условия, что точка **point2** лежит на конической поверхности, рис. М.5.1.3.

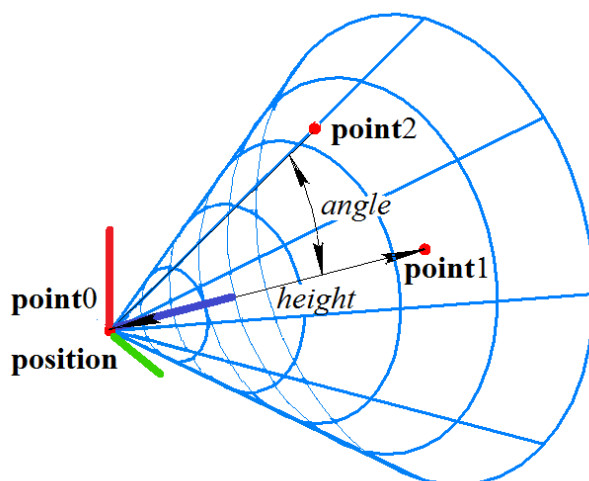


Рис. М.5.1.3.

Коническая поверхность описана в параграфе [0.5.4. Коническая поверхность MbConeSurface](#).

При построении сферической поверхности точка **point0** определяет центр сферы, в котором располагается начало локальной системы координат. Ось **axisZ** локальной системы координат поверхности направлена из точки **point0** в точку **point1**. Точка **point2** вместе с предыдущими точками определяет плоскость расположения осей **axisX** и **axisZ** локальной системы координат поверхности. Расстояние между точками **point0** и **point2** определяет радиус сферической поверхности, рис. М.5.1.4.

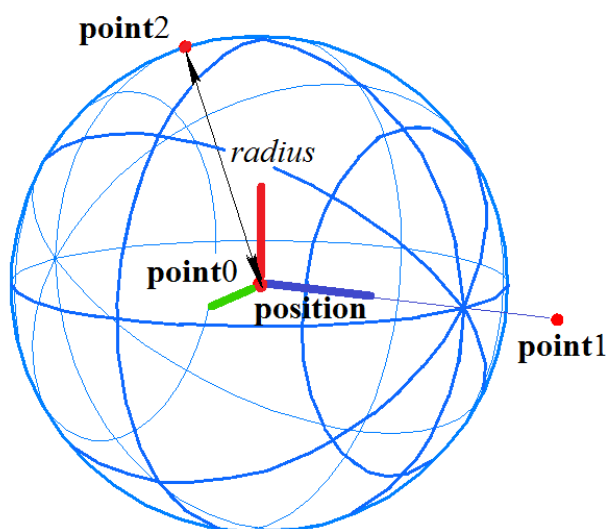


Рис. М.5.1.4.

Сферическая поверхность описана в параграфе [0.5.5. Сферическая поверхность MbSphereSurface](#).

При построении поверхности тора точка **point0** определяет центр тора, в котором располагается начало локальной системы координат. Ось **axisX** локальной системы координат поверхности направлена из точки **point0** в точку **point1**. Точка **point2** вместе с предыдущими точками определяют плоскость, в которой располагаются оси **axisX** и **axisZ** локальной системы координат поверхности. Расстояние между точками **point0** и **point1** определяет больший радиус тора, расстояние между точками **point1** и **point2** определяет меньший радиус тора, рис. М.5.1.5.

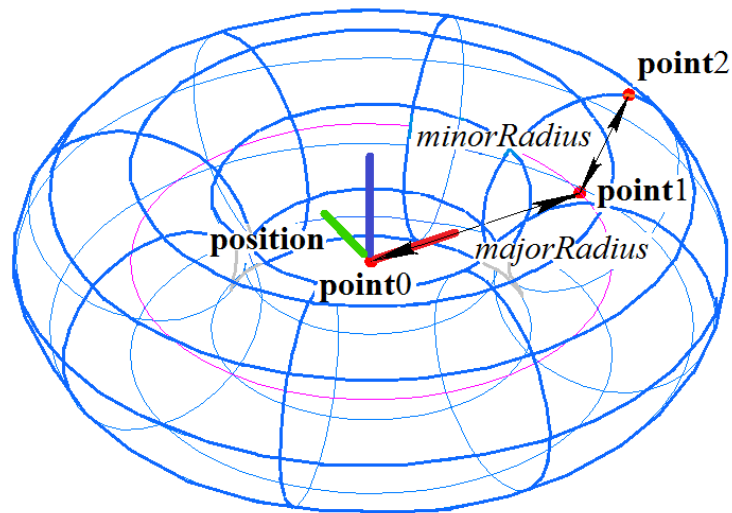


Рис. М.5.1.5.

Поверхность тора описана в параграфе [0.5.6. Поверхность тора MbTorusSurface](#).

Тестовое приложение test.exe выполняет построение элементарной поверхности по заданным точкам командой меню «Создать->Поверхность->Элементарную->».

М.5.2. Построение NURBS поверхности

Метод

MbResultType

SplineSurface (const [MbCartPoint3D](#) & pUMinVMin,
 const [MbCartPoint3D](#) & pUMaxVMin,
 const [MbCartPoint3D](#) & pUMaxVMax,
 const [MbCartPoint3D](#) & pUMinVMax,
 size_t uCount,
 size_t vCount,
 size_t uDegree,
 size_t vDegree,
[MbSurface](#) *& result)

выполняет построение плоской NURBS поверхности по угловым контрольным точкам.

Входными параметрами метода являются:

- **pUMinVMin** – левая нижняя угловая точка поверхности,
- **pUMaxVMin** – правая нижняя угловая точка поверхности,
- **pUMaxVMax** – правая верхняя угловая точка поверхности,
- **pUMinVMax** – левая верхняя угловая точка поверхности,
- **uCount** – количество контрольных точек вдоль первого параметра (по горизонтали),
- **vCount** – количество контрольных точек вдоль второго параметра (по вертикали),
- **uDegree** – порядок B-сплайнов вдоль первого параметра,
- **vDegree** – порядок B-сплайнов вдоль второго параметра.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Рассматриваемый метод строит NURBS-поверхность, контрольные точки которой расположены в узлах четырехугольной таблицы с `uCount` колонками и `vCount` строками. Порядок B-сплайнов поверхности по первому параметру равен `uDegree`, порядок B-сплайнов поверхности по второму параметру равен `vDegree`. Контрольные точки поверхности будут насчитаны из условия равномерного расположения и совпадения угловых точек с точками **pUMinVMin**, **pUMaxVMin**, **pUMaxVMax**, **pUMinVMax**, рис. М.5.2.1.

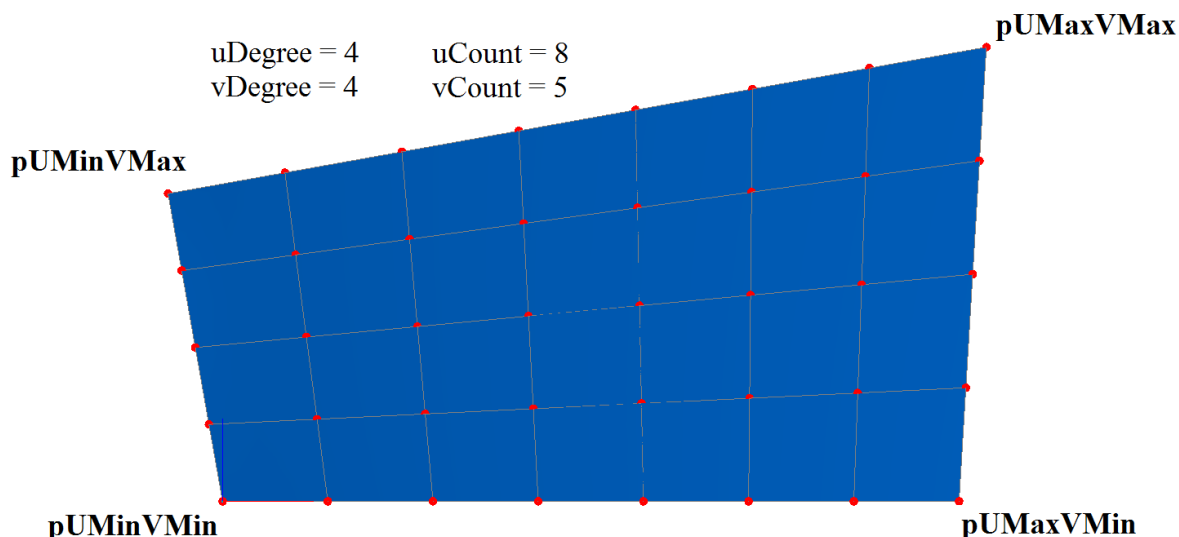


Рис. М.5.2.1.

Веса всех контрольных точек равны единице. Построенная поверхность предназначена для дальнейшей модификации. Для параметров метода должны выполняться равенства: $uCount \geq uDegree$ и $vCount \geq vDegree$. NURBS-поверхность описана в параграфе [О.5.22. NURBS-поверхность MbSplineSurface](#).

Метод

MbResultType

```
SplineSurface ( const SArray<MbCartPoint3D> & points,
                const SArray<double> & weights,
                size_t uCount,
                size_t vCount,
                size_t uDegree,
                const SArray<double> & uKnots,
                bool uClosed,
                size_t vDegree,
                const SArray<double> & vKnots,
                bool vClosed,
                MbSurface *& result )
```

выполняет построение NURBS поверхности по контрольным точкам и их весам.

Входными параметрами метода являются:

- **points** – множество контрольных точек, условно расположенных в виде $vCount$ строк по $uCount$ точек в каждой строке,
- *weights* – множество весов контрольных точек, согласованное с множеством контрольных точек,
- $uCount$ – количество контрольных точек вдоль первого параметра (в каждой строке),
- $vCount$ – количество контрольных точек вдоль второго параметра (количество строк),
- $uDegree$ – порядок B-сплайнов вдоль первого параметра,
- *uKnots* – узловой вектор первого параметра,
- *uClosed* – циклическая замкнутость поверхности по первому параметру,
- $vDegree$ – порядок B-сплайнов вдоль второго параметра.
- *vKnots* – узловой вектор второго параметра,
- *vClosed* – циклическая замкнутость поверхности по второму параметру.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Рассматриваемый метод строит NURBS-поверхность с контрольными точками **points**. Контрольные точки множества **points** будут разбиты на условные строки и столбцы. В каждой строке

будет содержаться $uCount$ контрольных точек из множества **points**, количество строк будет равно $vCount$. Таким образом множество **points** должно содержать $uCount \cdot vCount$ контрольных точек. Порядок B -сплайнов поверхности по первому параметру равен $uDegree$, порядок B -сплайнов поверхности по второму параметру равен $vDegree$. Для параметров метода должны выполняться равенства: $uCount \geq uDegree$ и $vCount \geq vDegree$. NURBS-поверхность описана в параграфе [O.5.22. NURBS-поверхность MbSplineSurface](#).

Множества $weights$, $uKnots$, $vKnots$ могут быть пустыми.

Если множество $weights$ не пусто, то оно должно быть согласовано с множеством контрольных точек **points**.

Если множества $uKnots$ и $vKnots$ не пусты, то они должны содержать определенное количество элементов: для $uClosed=false$ узловой вектор $uKnots$ должен содержать число элементов, равное $uCount+uDegree$; для $uClosed=true$ узловой вектор $uKnots$ должен содержать число элементов, равное $uCount+2uDegree-1$; для $vClosed=false$ узловой вектор $vKnots$ должен содержать число элементов, равное $vCount+vDegree$; для $vClosed=true$ узловой вектор $vKnots$ должен содержать число элементов, равное $vCount+2vDegree-1$. На рис. М.5.2.2 приведено расположение контрольных точек для построения сплайновой поверхности в форме тора.

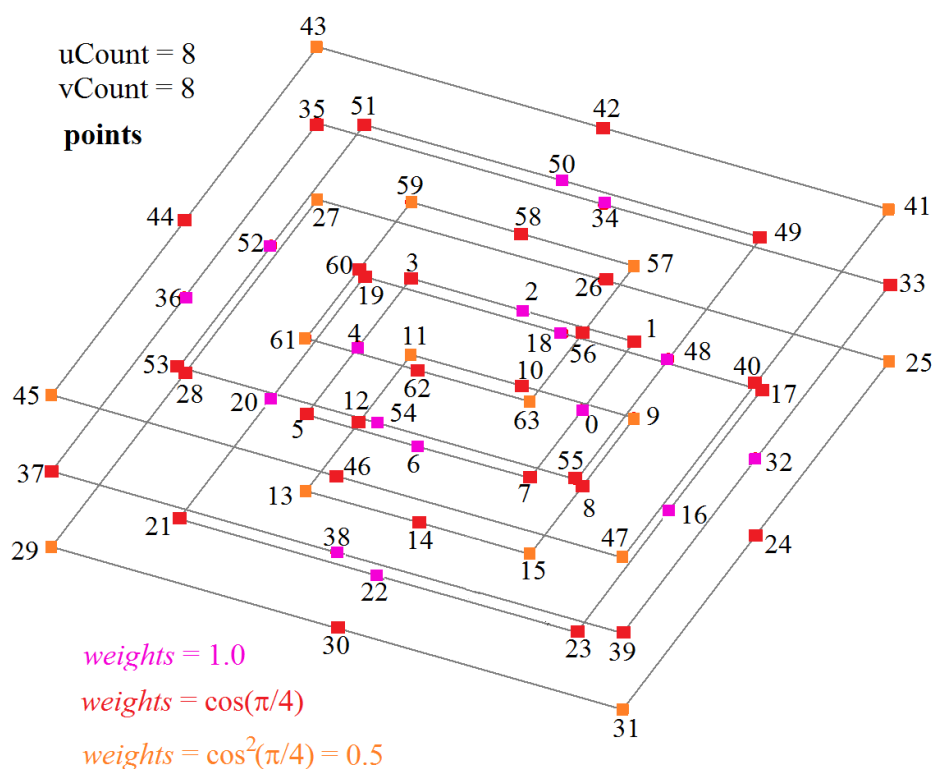


Рис. М.5.2.2.

На рис. М.5.2.3 приведена сплайновая поверхность в форме тора, построенная на указанных выше контрольных точках.

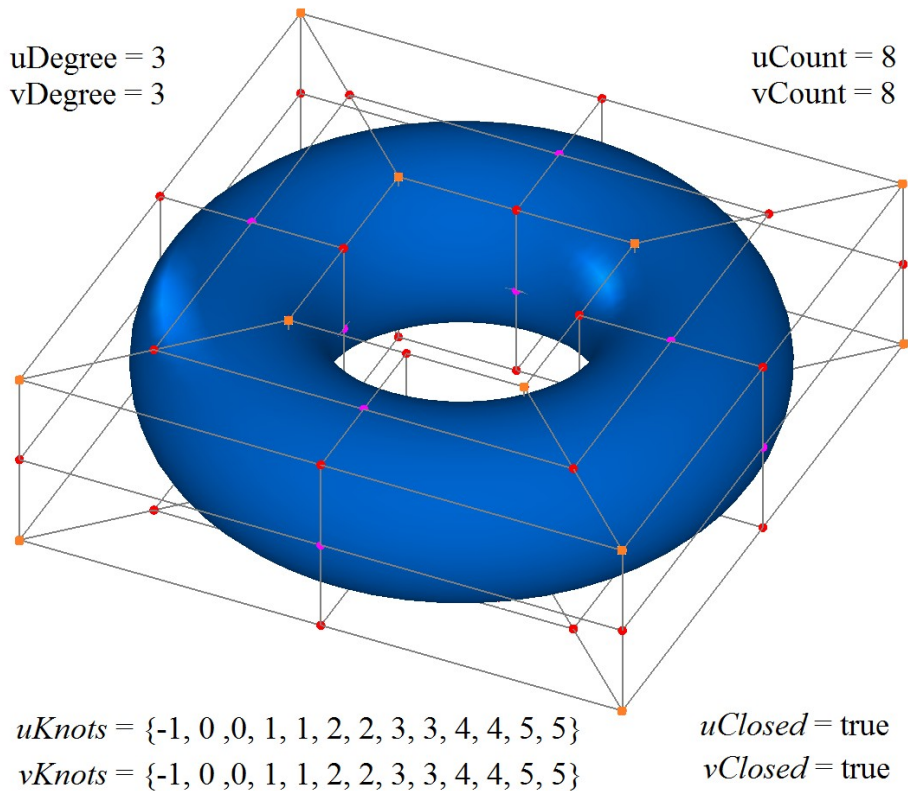


Рис. М.5.2.3.

Ниже приведен текст на языке программирования С++, реализующий построение сплайновой поверхности в форме тора рассматриваемым методом (центр тора совпадает с началом глобальной системы координат, ось тора направлена по глобальной оси Z).

```

//-----
void AddTorusPoints( double r, double z, SArray<MbCartPoint3D> & points, double w, SArray<double> & weights )
{
    MbCartPoint3D p( r, 0.0, z );
    MbVector3D toX( 1.0, 0.0, 0.0 ), toY( 0.0, 1.0, 0.0 );
    double wI( w ), wA = w * ::cos( M_PI_4 );
    points.Add( p ); weights.Add( wI );
    p.Add( toY, r ); points.Add( p ); weights.Add( wA );
    p.Add( toX,-r ); points.Add( p ); weights.Add( wI );
    p.Add( toX,-r ); points.Add( p ); weights.Add( wA );
    p.Add( toY,-r ); points.Add( p ); weights.Add( wI );
    p.Add( toY,-r ); points.Add( p ); weights.Add( wA );
    p.Add( toX, r ); points.Add( p ); weights.Add( wI );
    p.Add( toX, r ); points.Add( p ); weights.Add( wA );
}
//-----
void GetTorusPoints( double majorR, double minorR, SArray<MbCartPoint3D> & points, SArray<double> & weights )
{
    double zLavel( 0.0 );
    double w0( 1.0 ), wA = ::cos( M_PI_4 );
    ::AddTorusPoints( majorR - minorR, zLavel, points, w0, weights );
    ::AddTorusPoints( majorR - minorR,-minorR, points, wA, weights );
    ::AddTorusPoints( majorR      ,-minorR, points, w0, weights );
    ::AddTorusPoints( majorR + minorR,-minorR, points, wA, weights );
    ::AddTorusPoints( majorR + minorR, zLavel, points, w0, weights );
    ::AddTorusPoints( majorR + minorR, minorR, points, wA, weights );
    ::AddTorusPoints( majorR      , minorR, points, w0, weights );
    ::AddTorusPoints( majorR - minorR, minorR, points, wA, weights );
}
//-----
void GetTorusKnots( double tBeg, double tEnd, SArray<double> & knots )
{
    double dt = ( tEnd - tBeg ) / 4.0;
    double t = tBeg - dt; knots.Add( t );
    t = tBeg; knots.Add( t ); knots.Add( t );
    t += dt; knots.Add( t ); knots.Add( t );
}

```

```

t += dt; knots.Add( t ); knots.Add( t );
t += dt; knots.Add( t ); knots.Add( t );
t = tEnd; knots.Add( t ); knots.Add( t );
t += dt; knots.Add( t ); knots.Add( t );
}
//-----
MbSurface * CreateTorusSurface( double majorR, double minorR )
{
MbSurface * result( NULL );
MbResultType res( rt_Error );
if ( majorR > METRIC_NEAR && majorR > minorR - EPSILON ) {
SArray<MbCartPoint3D> points( 64, 1 );
SArray<double> weights( 64, 1 );
SArray<double> uKnots( 13, 1 );
SArray<double> vKnots( 13, 1 );
size_t uDegree( 3 ), vDegree( 3 );
size_t uCount( 8 ), vCount( 8 );
bool uClosed( true ), vClosed( true );
::GetTorusPoints( majorR, minorR, points, weights );
::GetTorusKnots( 0.0, 4.0, uKnots );
::GetTorusKnots( -2.0, 2.0, vKnots );
res = ::SplineSurface( points, weights, uCount, vCount, uDegree, uKnots, uClosed, vDegree, vKnots, vClosed, result );
}
return ( res == rt_Success ) ? result : NULL;
}

```

Метод

MbResultType

NurbsSurface (constMbSurface & **surface**,
VERSION version,
MbSurface *& **result**)

выполняет построение NURBS-копии заданной поверхности.

Входными параметрами метода являются:

- **surface** – исходная поверхность,
- **version** – версия построения.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Рассматриваемый метод построит NURBS-поверхность, форма которой копирует форму исходной поверхности. NURBS-поверхность описана в параграфе [O.5.22. NURBS-поверхность MbSplineSurface](#). Для большинства типов поверхностей NURBS-копия по форме полностью совпадает с исходной поверхностью. В случае невозможности точно воспроизвести форму исходной поверхности NURBS-копия аппроксимирует исходную поверхность с погрешностью, не превышающей 0,0001.

Если исходная поверхность является поверхностью с произвольными границами MbCurveBoundedSurface, описанной в параграфе [O.5.27. Поверхность с произвольными границами MbCurveBoundedSurface](#), то рассматриваемый метод построит поверхность с произвольными границами, базовой поверхностью которой будет NURBS-копия базовой поверхности исходной поверхности.

M.5.3. Построение поверхности выдавливания

Метод

MbResultType

ExtrusionSurface (MbCurve3D & **curve**,
const MbVector3D & **direction**,
MbSurface *& **result**)

выполняет построение поверхности выдавливания.

Входными параметрами метода являются:

- **curve** – образующая кривая,
- **direction** – вектор, определяющий направление и длину выдавливания.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface.h`.

Поверхность выдавливания относится к разновидности поверхностей движения, которые получают путем движения образующей кривой вдоль направляющей кривой. Поверхность выдавливания получают путем движения образующей кривой вдоль отрезка, направление и длину которого дает вектор **direction**. Поверхность выдавливания представлена в параграфе [O.5.7. Поверхность выдавливания MbExtrusionSurface](#). На рис. М.5.3.1 приведена поверхность, построенная выдавливанием кривой вдоль заданного вектора.

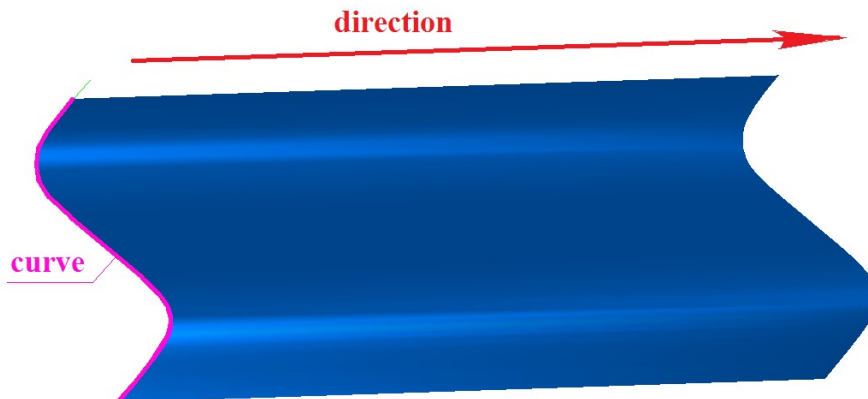


Рис. М.5.3.1.

М.5.4. Построение поверхности вращения

Метод

`MbResultType`

RevolutionSurface ([MbCurve3D](#) & **curve**,
const [MbCartPoint3D](#) & **origin**,
const [MbVector3D](#) & **axis**,
double *angle*,
[MbSurface](#) *& **result**)

выполняет построение поверхности вращения.

Входными параметрами метода являются:

- **curve** – образующая кривая,
- **origin** – точка на оси вращения,
- **axis** – направление оси вращения,
- *angle* – угол вращения.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface.h`.

Поверхность вращения относится к разновидности поверхностей движения, которые получают путем движения образующей кривой вдоль направляющей кривой. Поверхность вращения получают путем движения образующей кривой вдоль дуги окружности, у которой центр расположен в точке **origin**, ось параллельна вектору **axis**, угол раствора равен величине *angle*. Поверхность вращения представлена в параграфе [O.5.8. Поверхность вращения MbRevolutionSurface](#). На рис. М.5.4.1 приведена поверхность, построенная вращением кривой вокруг оси на заданный угол.

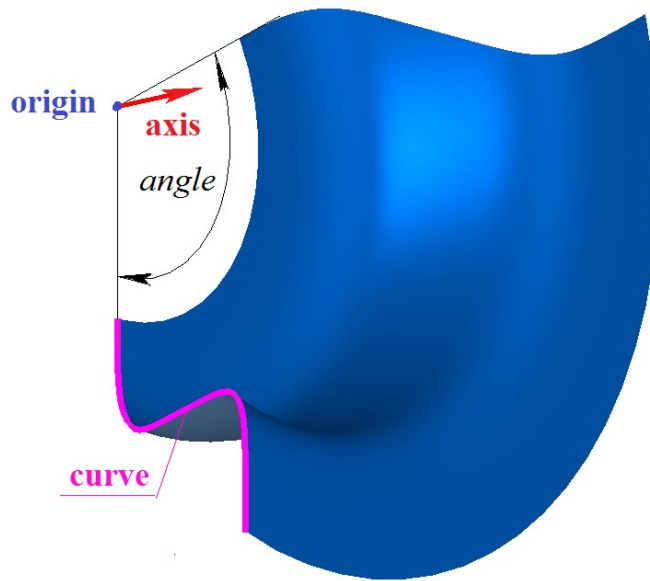


Рис. М.5.4.1.

М.5.5. Построение поверхностей заметания

Метод

MbResultType

ExpansionSurface ([MbCurve3D](#) & **curve**,
[MbCurve3D](#) & **spine**,
[MbSurface](#) *& **result**)

выполняет построение поверхности перемещения.

Входными параметрами метода являются:

- **curve** – образующая кривая,
- **spine** – направляющая кривая.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Поверхности заметания относятся к разновидности поверхностей движения, которые получают путем движения образующей кривой вдоль направляющей кривой. Поверхность перемещения получают путем плоскопараллельного движения образующей кривой вдоль направляющей кривой. Поверхность перемещения представлена в параграфе [0.5.9. Поверхность перемещения MbExpansionSurface](#). На рис. М.5.5.1 приведена поверхность, построенная плоскопараллельным перемещением образующей кривой вдоль направляющей кривой.

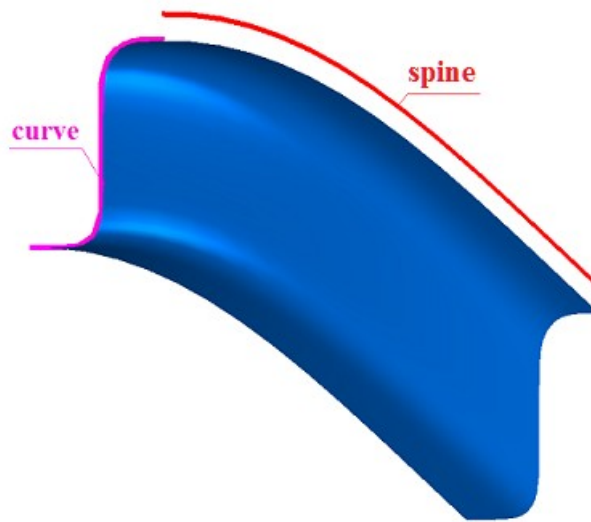


Рис. М.5.5.1.

Метод
 MbResultType
EvolutionSurface ([MbCurve3D](#) & **curve**,
 [MbCurve3D](#) & **spine**,
 [MbSurface](#) *& **result**)

выполняет построение кинематической поверхности.

Входными параметрами метода являются:

- **curve** – образующая кривая,
- **spine** – направляющая кривая.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface.h`.

Кинематическую поверхность получают путем движения образующей кривой **curve** вдоль направляющей кривой **spine**. Кинематическая поверхность представлена в параграфе [O.5.11. Кинематическая поверхность MbEvolutionSurface](#). На рис. М.5.5.2 приведена кинематическая поверхность, построенная заметанием образующей кривой вдоль направляющей кривой.

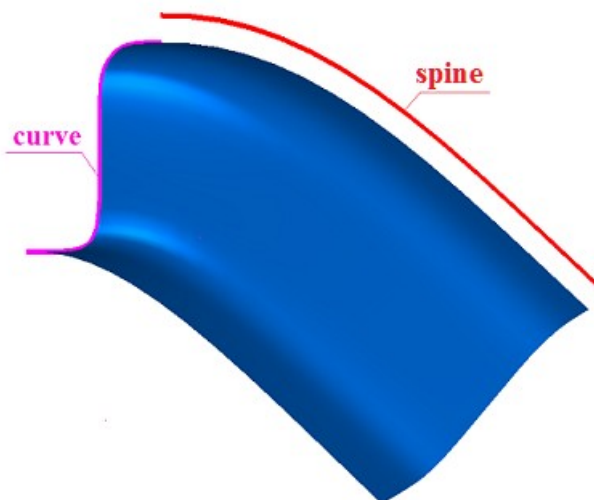


Рис. М.5.5.2.

Метод
 MbResultType
SpiralSurface ([MbCurve3D](#) & **curve**,
 const [MbCartPoint3D](#) & **point0**,
 const [MbCartPoint3D](#) & **point1**,
 const [MbCartPoint3D](#) & **point2**,
 double *step*,
[MbSurface](#) *& **result**)

выполняет построение спиральной поверхности.

Входными параметрами метода являются:

- **curve** – образующая кривая,
- **point0** – начало локальной системы координат,
- **point1** – точка на оси Z локальной системы координат,
- **point2** – точка в направлении оси X локальной системы координат.
- *step* – шаг спирали.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface.h`.

Спиральная поверхность является частным случаем кинематической поверхности, описанной выше. Её получают путем движения образующей кривой **curve** вдоль цилиндрической спирали. Контрольные точки **point0**, **point1** и **point2** определяют локальную систему координат и длину спирали. Точка **point0** определяет начало локальной системы координат. Ось **axisZ** локальной системы координат спирали направлена из точки **point0** в точку **point1**. Точка **point2** вместе с предыдущими точками определяет плоскость расположения осей **axisX** и **axisZ** локальной системы координат спирали. Расстояние между точками **point0** и **point1** определяет высоту спирали. Параметр *step* определяет шаг спирали. Спиральная поверхность представлена в параграфе [O.5.10. Спиральная поверхность MbSpiralSurface](#). На рис. М.5.5.3 приведена спиральная поверхность

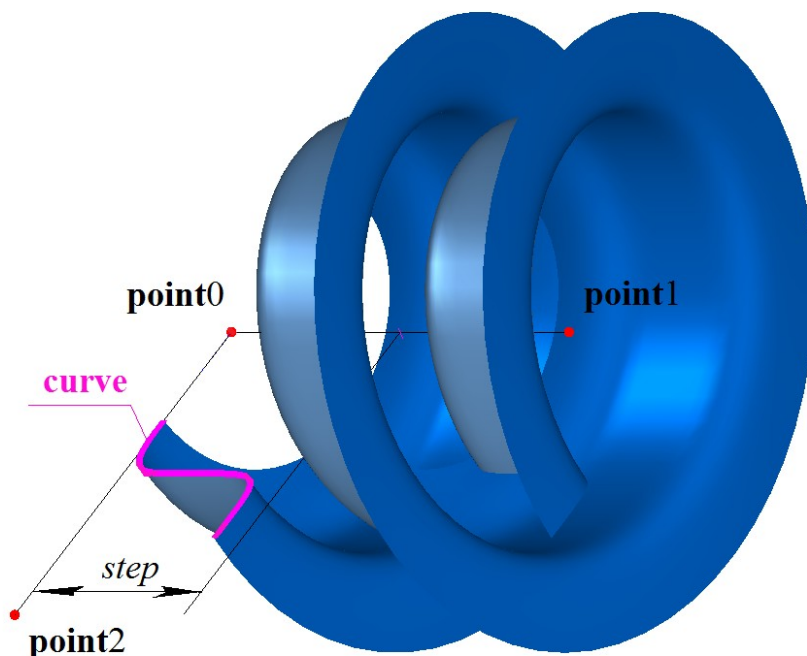


Рис. М.5.5.3.

М.5.6. Построение поверхности по семейству кривых

Метод

MbResultType

```
LoftedSurface ( const RPAArray<MbCurve3D> & curves,  
                bool closed,  
                const MbVector3D & begDirection,  
                const MbVector3D & endDirection,  
                MbSurface *& result )
```

выполняет построение поверхности по семейству кривых.

Входными параметрами метода являются:

- **curves** – семейство кривых,
- *closed* – циклическая замкнутость поверхности по второму параметру,
- **begDirection** – вектор направления в начале поверхности,
- **endDirection** – вектор направления в конце поверхности.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Построенная поверхность проходит по множеству кривых **curves**. Если все кривые семейства циклически замкнуты, то поверхность будет циклически замкнутой по первому параметру. При построении поверхности следует следить за направлениями кривых, так как при разнонаправленности кривых возможно самопересечение поверхности. Параметр *closed* определяет циклическую замкнутость поверхности по второму параметру. Для значения *closed=false* параметры **begDirection** и **endDirection** определяют направление поверхности на крайних кривых. Если длина векторов **begDirection** и **endDirection** равна нулю, то направление поверхности на крайних кривых будет определено из условия равенства нулю вторых производных на торцах поверхности. Поверхность на семействе кривых представлена в параграфе [0.5.15. Поверхность на семействе кривых MbLoftedSurface](#). На рис. М.5.6.1 приведена циклически замкнутая по второму параметру поверхность, построенная по семейству кривых.

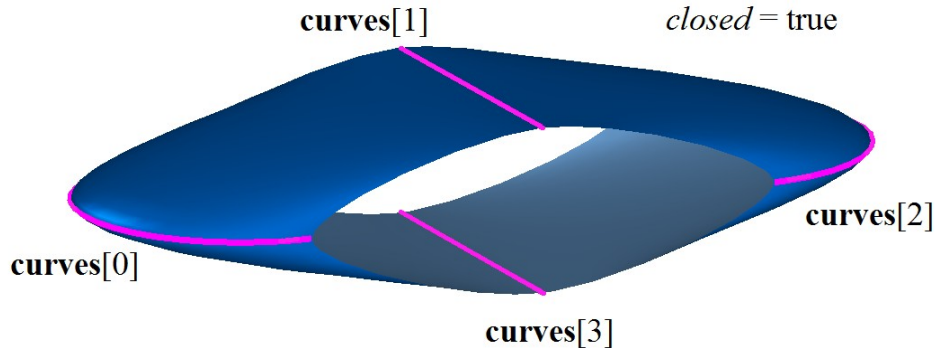


Рис. М.5.6.1.

На рис. М.5.6.2 приведена незамкнутая поверхность, построенная по семейству кривых, с заданными направлениями на торцах.

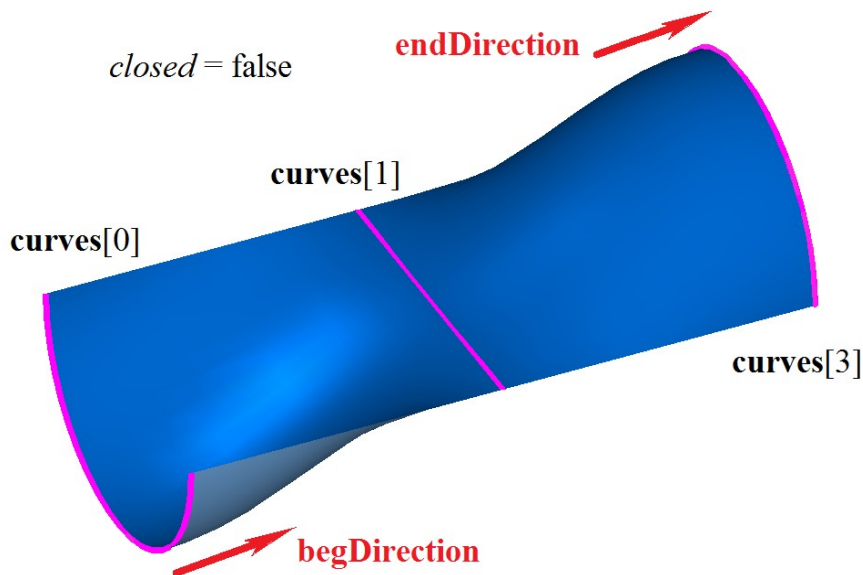


Рис. М.5.6.2.

Метод
 MbResultType
LoftedSurface (const RPAArray<MbCurve3D> & curves,
 MbCurve3D & spine,
 MbSurface *& result)

выполняет построение поверхности по семейству кривых и направляющей кривой.

Входными параметрами метода являются:

- **curves** – семейство кривых,
- **spine** – направляющая кривая.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Построенная поверхность проходит по множеству кривых **curves**. Направляющая кривая **spine** определяет форму поверхности между кривыми семейства. Поверхность на семействе кривых и направляющей представлена в параграфе [0.5.16. Поверхность на семействе кривых и направляющей MbElevationSurface](#). На рис. М.5.6.3 приведена поверхность, построенная по семейству кривых и направляющей.

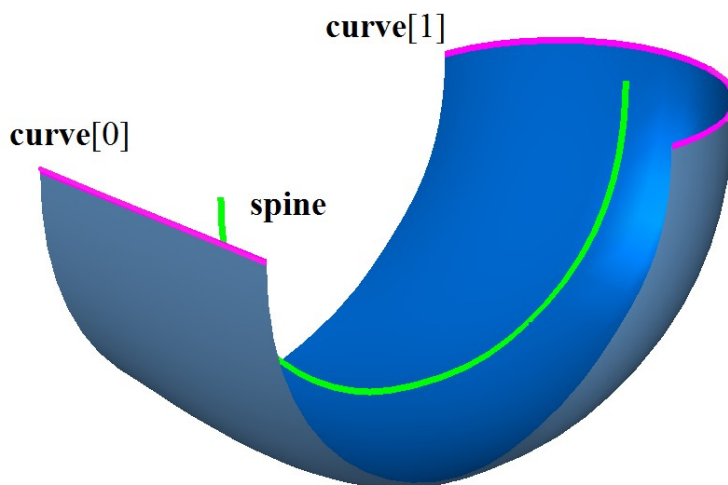


Рис. М.5.6.3.

М.5.7. Построение линейчатых поверхностей

Метод
MbResultType
SectorSurface ([MbCurve3D](#) & **curve**,
const [MbCartPoint3D](#) & **point**,
[MbSurface](#) *& **result**)

выполняет построение секториальной поверхности по кривой и точке.

Входными параметрами метода являются:

- **curve** – кривая,
- **point** – точка.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface.h`.

Построенная поверхность проходит по кривой **curve** и точке **point**. В точке **point** поверхность имеет полюс, в котором обращается в ноль производная радиуса-вектора поверхности по первому параметру. Кривая **curve** не должна быть вырожденной и не должна проходить через точку **point**. Секториальная поверхность представлена в параграфе [0.5.13. Секториальная поверхность MbSectorSurface](#). На рис. М.5.7.1 приведена секториальная поверхность.

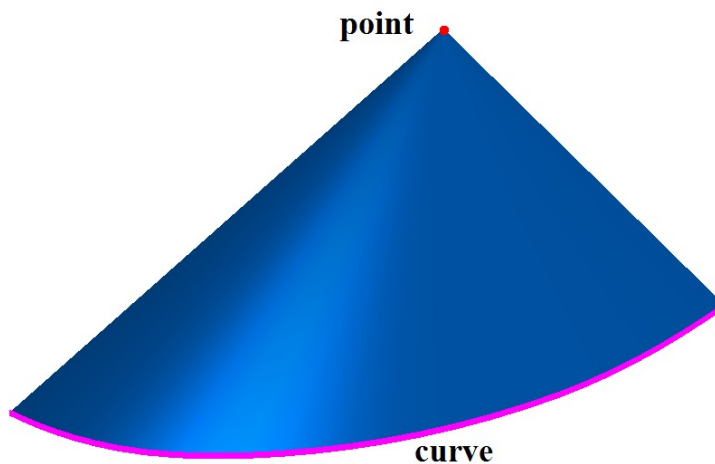


Рис. М.5.7.1.

Метод
MbResultType
RuledSurface ([MbCurve3D](#) & **curve1**,
[MbCurve3D](#) & **curve2**,
[MbSurface](#) *& **result**)

выполняет построение линейчатой поверхности по двум кривым.

Входными параметрами метода являются:

- **curve1** – первая кривая,
- **curve2** – вторая кривая.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface.h`.

Построенная поверхность проходит по кривым **curve1** и **curve2**. Чтобы поверхность не имела самопересечения, кривые **curve1** и **curve2** не должны пересекаться в точках, не совпадающих с краями кривых. Линейчатая поверхность представлена в параграфе [0.5.14. Линейчатая поверхность MbRuledSurface](#). На рис. М.5.7.2 приведена линейчатая поверхность.

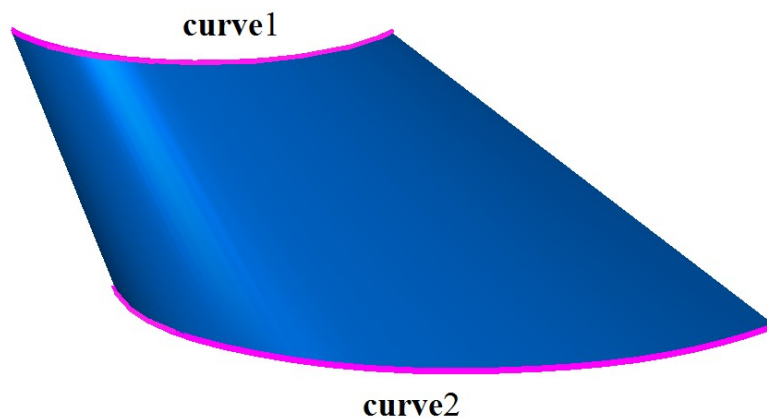


Рис. М.5.7.2.

М.5.8. Построение поверхности по трем кривым

Метод

MbResultType

CornerSurface ([MbCurve3D](#) & **curve1**,
[MbCurve3D](#) & **curve2**,
[MbCurve3D](#) & **curve3**,
[MbSurface](#) *& **result**)

выполняет построение поверхности по трем кривым.

Входными параметрами метода являются:

- **curve1** – первая кривая,
- **curve2** – вторая кривая,
- **curve3** – третья кривая.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

В общем случае построенная поверхность не проходит по кривым **curve1**, **curve2** и **curve3**. Для построения поверхности кривые должны иметь точки скрещения, в которых отрезок, соединяющий точки скрещения, ортогонален кривым. Построенная поверхность будет проходить по кривым **curve1**, **curve2** и **curve3**, если кривые попарно пересекаются. Поверхность имеет полюс, в котором обращается в ноль производная радиуса-вектора поверхности по первому параметру. Поверхность, построенная по трем кривым, представлена в параграфе [О.5.17. Поверхность на трёх кривых MbCornerSurface](#). На рис. М.5.8.1 приведена поверхность, построенная по трем кривым, в двух ракурсах.

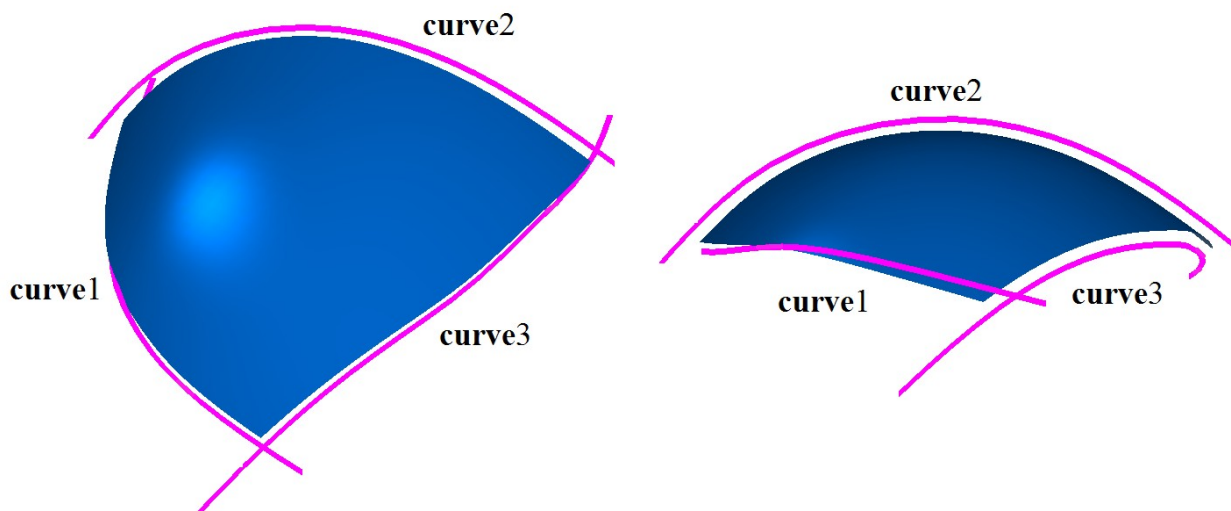


Рис. М.5.8.1.

М.5.9. Построение поверхности по четырем кривым

Метод
 MbResultType
CoverSurface ([MbCurve3D](#) & **curve1**,
 [MbCurve3D](#) & **curve2**,
 [MbCurve3D](#) & **curve3**,
 [MbCurve3D](#) & **curve4**,
 [MbSurface](#) *& **result**)

выполняет построение поверхности по четырем кривым.

Входными параметрами метода являются:

- **curve1** – первая кривая,
- **curve2** – вторая кривая,
- **curve3** – третья кривая,
- **curve4** – четвертая кривая.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_surface.h`.

В общем случае построенная поверхность не проходит по кривым **curve1**, **curve2**, **curve3** и **curve4**. Для построения поверхности кривые **curve1** и **curve2**, **curve3** и **curve4**, **curve1** и **curve4**, **curve2** и **curve3** должны иметь точки скрещения, в которых отрезок, соединяющий точки скрещения, ортогонален кривым. Построенная поверхность будет проходить по кривым **curve1**, **curve2**, **curve3** и **curve4**, если кривые попарно пересекаются. Поверхность, построенная по четырём кривым, представлена в параграфе [0.5.18. Поверхность Кунса MbCoverSurface](#). На рис. М.5.9.1 приведена поверхность, построенная по четырём кривым, в двух ракурсах.

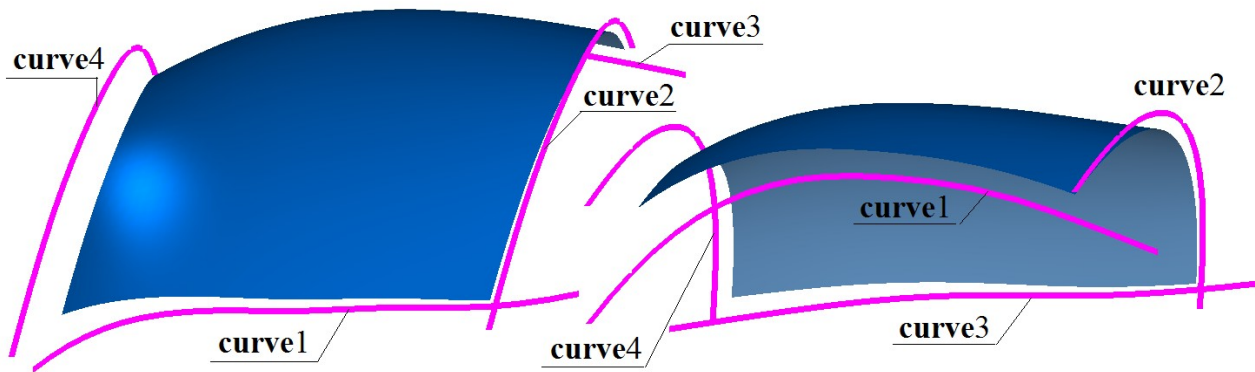


Рис. М.5.9.1.

М.5.10. Построение поверхности по сети кривых

Метод

MbResultType

MeshSurface (const RPAarray<MbCurve3D> & uCurves,
const RPAarray<MbCurve3D> & vCurves,
MbSurface *& result)

выполняет построение поверхности по двум семействам кривых.

Входными параметрами метода являются:

- **uCurves** – первое множество кривых (вдоль первого параметра),
- **vCurves** – второе множество кривых (вдоль второго параметра).

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Для построения поверхности каждая кривая множества **uCurves** должна пересекаться со всеми кривыми множества **vCurves** и, соответственно, каждая кривая множества **vCurves** должна пересекаться со всеми кривыми множества **uCurves**. Чтобы не было самопересечения поверхности, соседние кривые множества **uCurves** должны иметь одинаковое направление и соседние кривые множества **vCurves** должны иметь одинаковое направление. Построенная поверхность будет проходить по кривым множества **uCurves** и множества **vCurves**. Поверхность, построенная по сети кривых, представлена в параграфе [О.5.20. Поверхность на сети кривых MbMeshSurface](#). На рис. М.5.10.1 приведены два множества кривых, а на рис. М.5.10.2 приведена построенная поверхность, построенная на этих кривых.

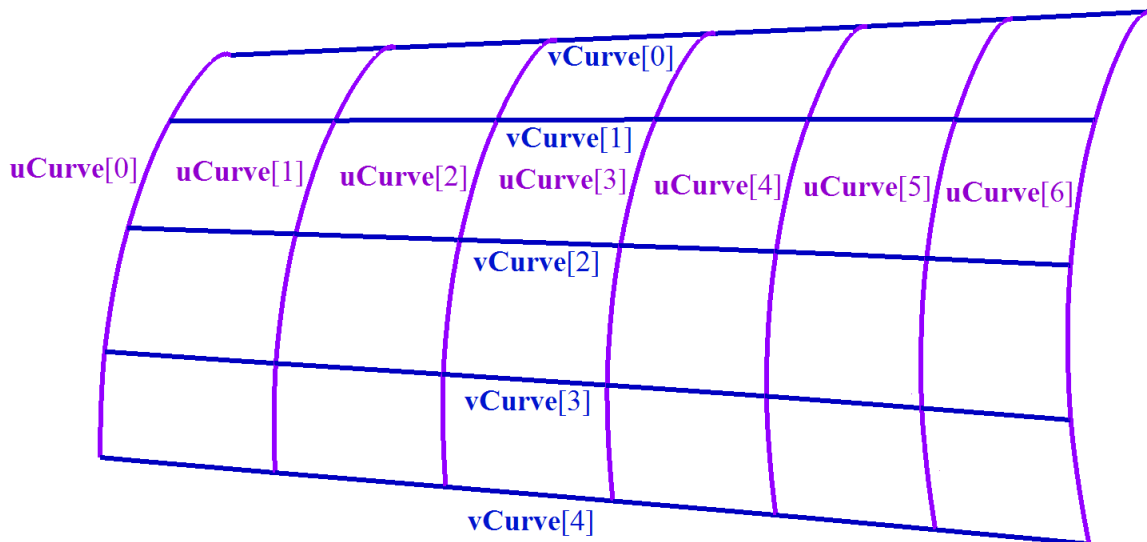


Рис. М.5.10.1.

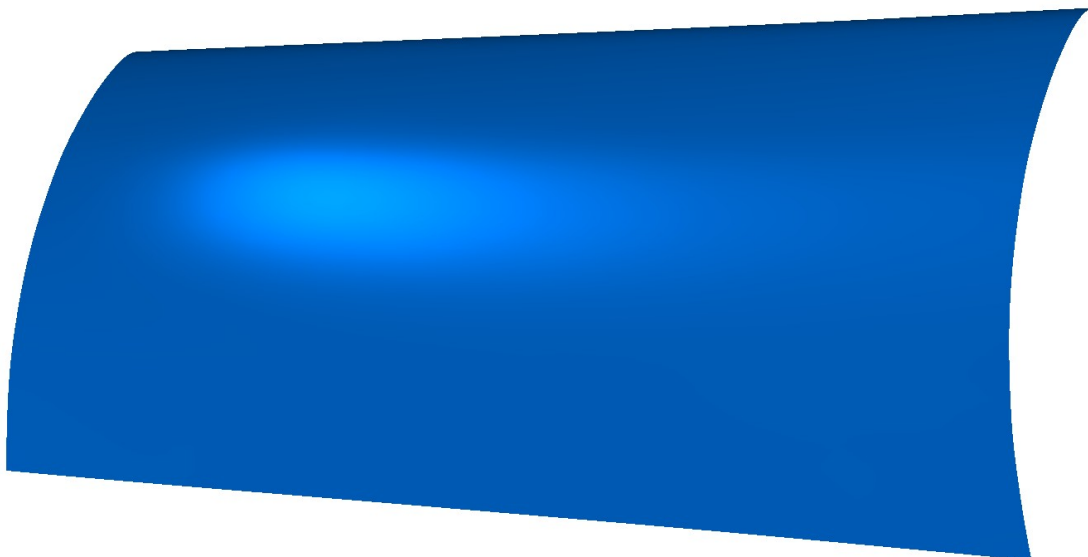


Рис. М.5.10.2.

М.5.11. Построение эквидистантной поверхности

Метод

MbResultType

OffsetSurface ([MbSurface](#) & **surface**,
double *distance*,
[MbSurface](#) *& **result**)

выполняет построение эквидистантной поверхности.

Входными параметрами метода являются:

- **surface** – базовая поверхность,
- *distance* – величина эквидистанты (со знаком).

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Построение поверхности возможно при отсутствии у поверхности **surface** точек, в которых поверхность изгибается в стороны эквидистанты с радиусом кривизны меньше чем *distance*. Каждая точка построенной поверхности располагается на расстоянии *distance* по местной нормали поверхности **surface** с такими же параметрами. Эквидистантная поверхность представлена в параграфе [0.5.23. Эквидистантная поверхность MbOffsetSurface](#). На рис. М.5.11.1 приведена эквидистантная поверхность.

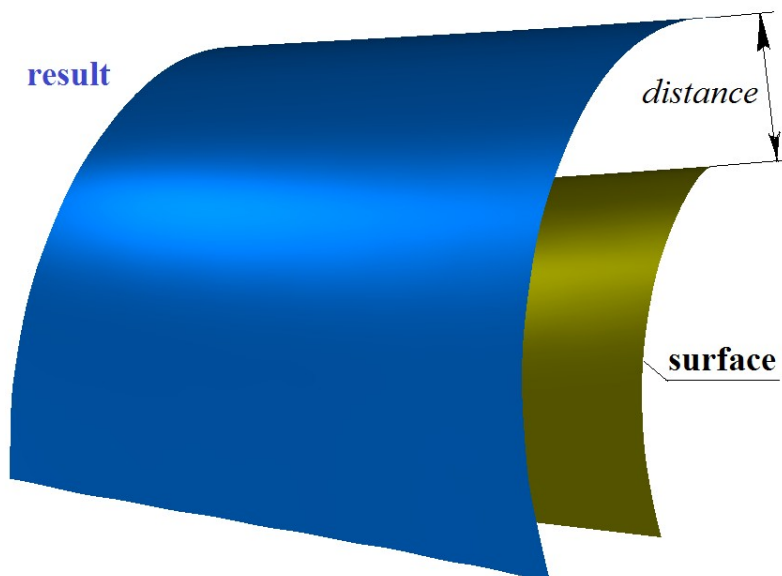


Рис. М.5.11.1.

После построения эквидистантная поверхность имеет область определения параметров, совпадающую с областью определения параметров поверхности **surface**. Область определения параметров эквидистантной поверхности может быть расширена.

Метод

MbResultType

ExtendedSurface ([MbSurface](#) & **surface**,

double *uMin*,

double *uMax*,

double *vMin*,

double *vMax*,

[MbSurface](#) *& **result**)

выполняет построение расширенной поверхности.

Входными параметрами метода являются:

- **surface** – базовая поверхность,
- *uMin* – минимальное значение первого параметра поверхности,
- *uMax* – максимальное значение первого параметра поверхности,
- *vMin* – минимальное значение второго параметра поверхности,
- *vMax* – максимальное значение второго параметра поверхности.

Выходным параметром метода является построенная поверхность **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

Построение поверхности возможно при отсутствии полюсов и других особых точек на границе области определения поверхности и на ее расширении. Построенная поверхность совпадает с поверхностью **surface**, но имеет другую область определения параметров. Область может быть как уменьшена, так и расширена. При выходе за пределы области определения параметра поверхность **surface** продолжается по касательной. Расширенная поверхность строится как эквидистантная поверхность с нулевым значением смещения. На рис. М.5.11.2 приведена расширенная поверхность.

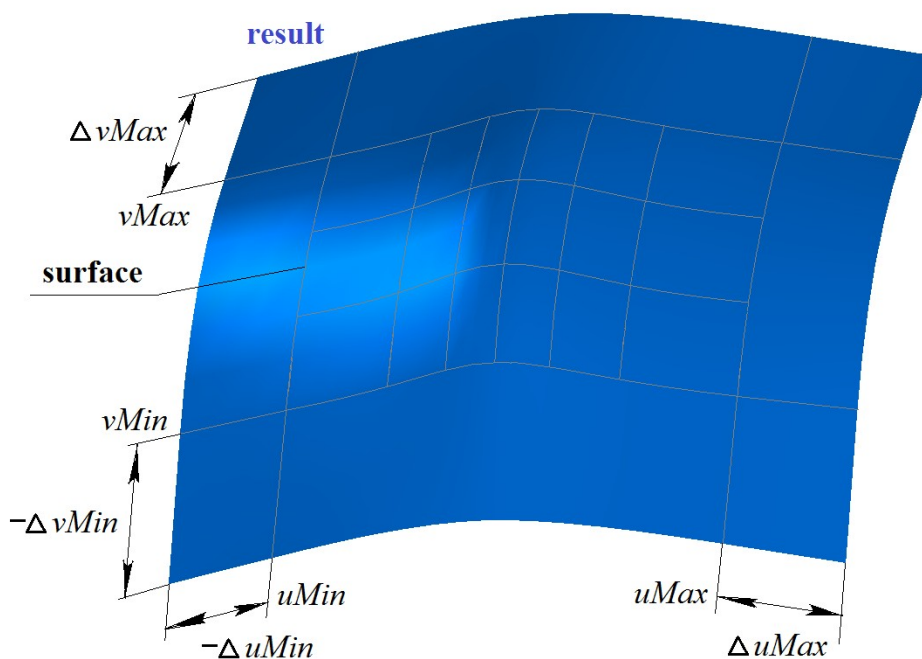


Рис. М.5.11.2.

М.5.12. Построение поверхности с произвольными границами

Метод

MbResultType

BoundedSurface ([MbSurface](#) & **surface**,
const RPAArray<[MbCurve](#)> & **bounds**,
[MbSurface](#) *& **result**)

выполняет построение поверхности с заданной границей.

Входными параметрами метода являются:

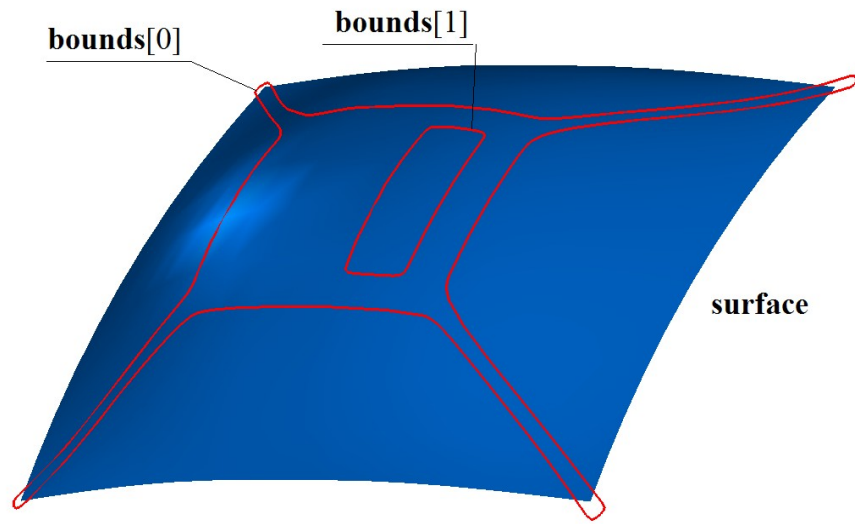
- **surface** – исходная поверхность,
- **bounds** – множество границ в пространстве параметров.

Выходным параметром метода является построенная поверхность **result**.

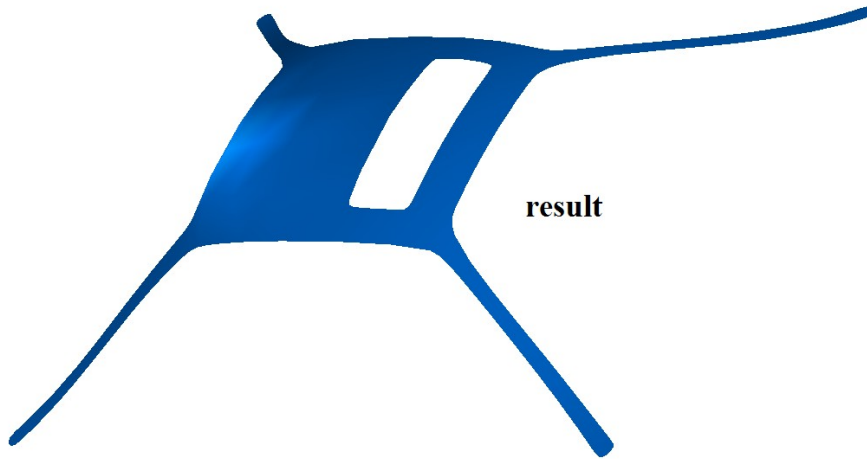
При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_surface.h`.

По умолчанию поверхность имеют прямоугольную область определения параметров. Исходная область определения параметров может быть изменена путем описания границы области двумерными замкнутыми кривыми **bounds**. Если кривых множества **bounds** больше одной, то первая кривая множества **bounds** должна описывать внешнюю границу, а остальные кривые должны лежать внутри первой кривой множества. Кривые множества **bounds** могут выходить за пределы исходной области определения параметров, при условии, что там отсутствуют особые точки. Если множество границ пусто, то будет построен замкнутая составная кривая по исходной границе области определения параметров поверхности **surface**. Поверхность с произвольными границами представлена в параграфе [О.5.27. Поверхность с произвольными границами MbCurveBoundedSurface](#). На рис. М.5.12.1 приведена поверхность и две замкнутые кривые на ней, а на рис. М.5.12.2 приведен результат построения поверхности с границами в форме этих кривых.



Puc. M.5.12.1.



Puc. M.5.12.2.

М.6. МЕТОДЫ ПРЯМОГО МОДЕЛИРОВАНИЯ

Прямое моделирование позволяет модифицировать геометрическую модель путем изменения составляющих её элементов. Геометрическая модель может иметь любую степень готовности – она может представлять собой заготовку или быть готовой моделью. Методами прямого моделирования могут модифицироваться как все элементы геометрической модели, так и отдельная группа элементов. Например, группа граней тела может быть перемещена относительно остальных граней тела, группа граней тела может быть заменена эквидистантными гранями или деформируемыми гранями. Из указанной группы граней может быть образовано новое тело. У геометрической модели могут быть удалены указанные скругления или характерные особенности, например, отверстия или выступы.

М.6.1. Построение трансформированного тела

Метод

MbResultType

TransformedSolid ([MbSolid](#)& **solid**,
MbcCopyMode *sameShell*,
const TransformValues & *params*,
const MbSNameMaker & *names*,
[MbSolid](#) *& **result**)

выполняет трансформацию копии исходного тела по заданной матрице.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- *params* – параметры трансформации,
- *names* – именователь граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_direct.h`.

Рассматриваемый метод выполняет копирование тела и масштабирование копии тела с помощью матрицы, имеющей одинаковые или разные масштабы преобразования по осям глобальной системы координат.

Параметр **solid** содержит исходное тело, которое подлежит обработке. Параметр *sameShell* управляет передачей неизменных граней, ребер и вершин от исходного тела **solid** к построенному телу **result**. Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbcCopyMode объявлено в файле `mb_enum.h` и описано в параграфе [0.7.9. Копирование множества граней MbFaceShell](#)

Параметр *params* содержит матрицу трансформации *params.matrix* и величины *params.fixedPoint*, *params.useFixed*, *params.isotropy*, которые используются для расчета матрицы трансформации. Параметр трансформации TransformValues объявлен в файле `op_shell_parameter.h`.

Параметр *names* используется для версионирования операции.

Матрица трансформации рассчитывается по габаритному кубу исходного тела **solid** и смещению одной из точек габаритного куба методом

`bool`

`MbCube::CalculateMatrix(size_t pIndex,
const MbCartPoint3D & point,
const MbCartPoint3D & fixedPoint,
bool useFixed,
bool isotropy,
MbMatrix3D & matrix) const`, где

- *pIndex* – номер точки габаритного куба;
- **point** – точка пространства, с которой нужно совместить точку габаритного куба с номером *pIndex*;
- **fixedPoint** – неподвижная точка преобразования; *useFixed* – флаг использования неподвижной точки;
- *isotropy* – флаг равенства масштабов преобразования.

На рис. М.6.1.1 приведена нумерация точек габаритного куба тела **solid**. Вершины габаритного куба имеют номера 0-7, середины рёбер габаритного куба имеют номера 8-19, центры граней габаритного куба имеют номера 20-25.

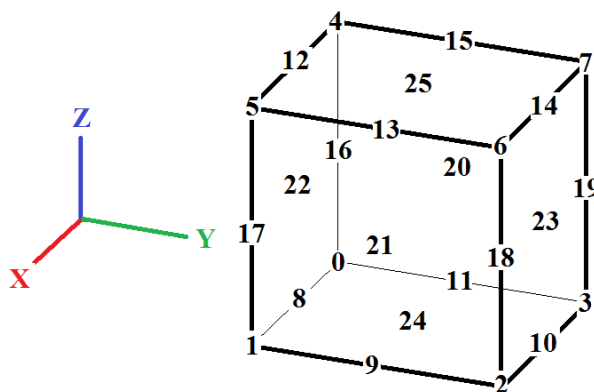


Рис. М.6.1.1.

Матрица трансформации **matrix** рассчитывается по деформированному кубу, полученному путём совмещения точки габаритного куба с номером *pIndex* с точкой пространства **point**. Если *useFixed=true*, то неподвижной точкой преобразования будет точка **fixedPoint**. Если *useFixed=false*, то неподвижной точкой преобразования будет точка габаритного куба, противоположная точке с номером *pIndex*. Противоположными точками габаритного куба считаются точки: 0-6, 1-7, 2-4, 3-5, 8-14, 9-15, 10-12, 11-13, 16-18, 17-19, 20-21, 22-23, 24-25. Если *isotropy=true*, то масштабы преобразования по всем осям будут одинаковы и пропорциональны вектору смещения точки габаритного куба с номером *pIndex*. Если *isotropy=false*, то масштабы преобразования будут пропорциональны проекциям на стороны габаритного куба вектора смещения точки габаритного куба с номером *pIndex*. Величины *params.fixedPoint*, *params.useFixed*, *params.isotropy* и *params.matrix* используются в качестве параметров **fixedPoint**, *useFixed*, *isotropy* и **matrix** метода **CalculateMatrix**.

На рис. М.6.1.2 приведено исходное тело **solid**, точки габаритного куба и точка пространства **point**, с которой нужно совместить указанную точку габаритного куба, а на рис. М.6.1.3 приведено трансформированное тело **result**, построенное рассматриваемым методом с параметрами *useFixed=false* и *isotropy=false*.

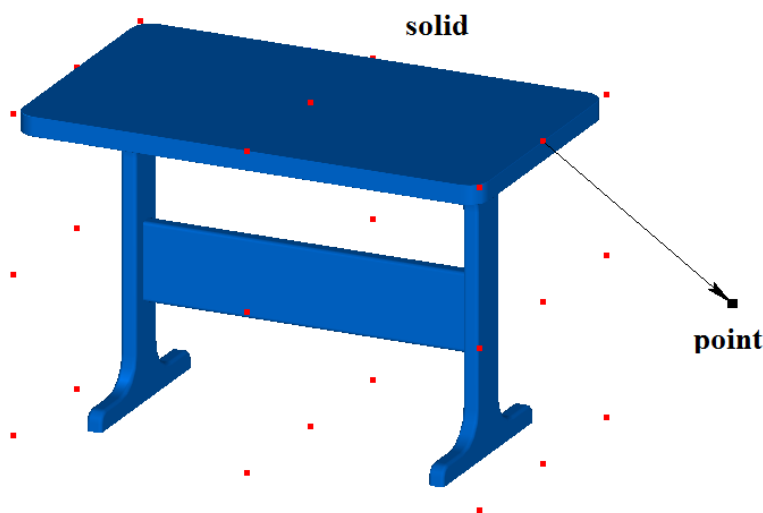


Рис. М.6.1.2.

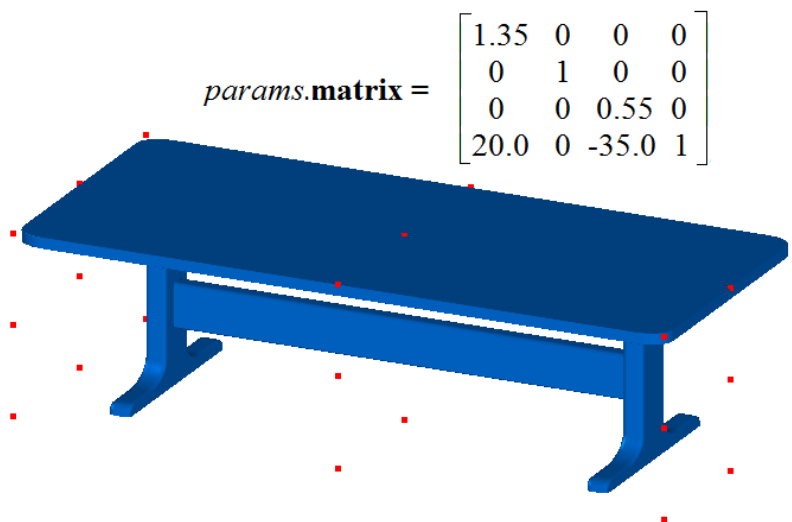


Рис. М.6.1.3.

Трансформировать тело по заданной матрице можно также классным методом **Transform**, который перегружен у всех геометрических объектов, но при этом тело не будет содержать информацию о выполненных над ним действиях.

Метод **TransformedSolid** добавляет в журнал построенного тела строитель **MbTransformedSolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbTransformedSolid** объявлен в файле `cr_transformed_solid.h`.

Тестовое приложение `test.exe` выполняет построение трансформированного тела командой меню «Создать->Тело->Прямым редактированием->Трансформированием габарита».

М.6.2. Построение модифицированного тела

Метод
MbResultType
FaceModifiedSolid (**MbSolid** & **solid**,
MbeCopyMode *sameShell*,
const **ModifyValues** & *params*,
const **RPAArray**<**MbFace**> & **faces**,
const **MbSNameMaker** & **names**,
MbSolid* & **result**)

выполняет модификацию указанных граней копии исходного тела одним из указанных способов.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- *params* – параметры модификации,
- **faces** – модифицируемые грани тела,
- **names** – именованье граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_direct.h`.

Рассматриваемый метод выполняет копирование тела и одно из следующих действий: удаление из копии тела указанных граней, создание нового тела из копий указанных граней с окружением, перемещение указанных граней относительно оставшихся граней в копии тела, замену указанных граней в копии тела эквидистантными гранями, замену указанных граней в копии тела деформируемыми гранями, удаление указанных граней скругления в копии тела.

Параметр **solid** содержит исходное тело, которое подлежит обработке. Параметр *sameShell* управляет передачей неизменных граней, ребер и вершин от исходного тела **solid** к построенному телу **result**. Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление *MbeCopyMode* объявлено в файле *mb_enum.h* и описано в параграфе [0.7.9. Копирование множества граней MbFaceShell](#).

Параметр *params* описывает способ модификации, определяемый типом *params.way*, и величину модификации, определяемую вектором *params.direction*. Тип модификации *params.way* может принимать одно из семи значений перечисления *MbeModifyingType*: *dmt_Remove*, *dmt_Create*, *dmt_Action*, *dmt_Offset*, *dmt_Fillet*, *dmt_Supple*, *dmt_Purify*. Параметры модификации *ModifyValues* и перечисление *MbeModifyingType* объявлены в файле *op_shell_parameter.h*

Параметр **faces** содержит множество граней тела **solid**, которые будут тем или иным способом изменены рассматриваемым методом в копии тела.

Параметр *names* используется для именования новых граней, ребер, вершин и версионирования операции.

При *params.way=dmt_Remove* рассматриваемый метод удаляет грани **faces** из копии тела **solid** и, используя окружение, обрабатывает место удаленных граней сохраняя замкнутость тела. Если грани **faces** связаны с остальными гранями скруглениями, то грани скругления добавляются к граням **faces**. Если множество граней **faces** пусто, то рассматриваемый метод удаляет цилиндрические и плоские грани, радиус которых меньше или равен длине вектора *params.direction*, что позволяет удалять мелкие сквозные или глухие отверстия не перечисляя их. На рис. М.6.2.1 приведено исходное тело, а на рис. М.6.2.2 приведено построенное тело, полученное путем удаления указанных граней исходного тела.

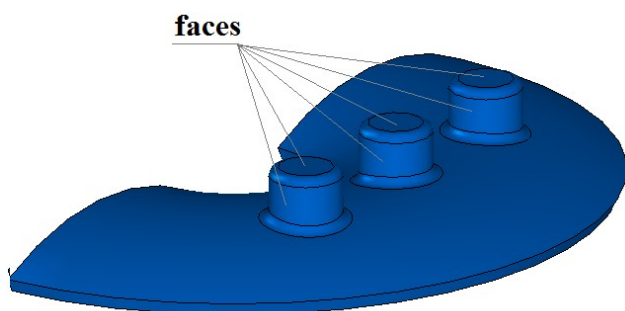


Рис. М.6.2.1.

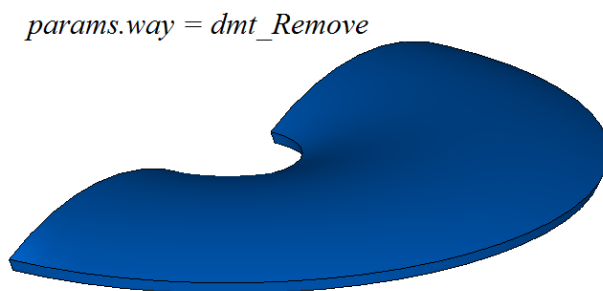


Рис. М.6.2.2.

При *params.way=dmt_Create* рассматриваемый метод создает новое тело из копий граней **faces**. Если грани **faces** связаны с остальными гранями скруглениями, то сначала удаляются скругления, а затем из граней **faces** создается новое тело. Новое тело будет замкнутым, так как в него войдут грани **faces** и грани, созданные на базе поверхностей смежных с гранями **faces**. Дополнительные грани необходимы для ликвидации краевых ребер. На рис. М.6.2.3 приведено исходное тело, а на рис. М.6.2.4 приведено вновь построенное тело, полученное из указанных граней исходного тела. Тело, приведенное на рис. М.6.2.4, состоит из пяти граней – трех граней **faces** и двух граней на базе цилиндрической поверхности, на которую опирались два удаленных скругления.

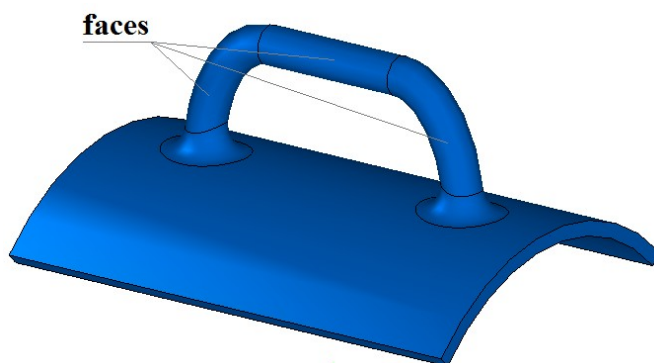


Рис. М.6.2.3.

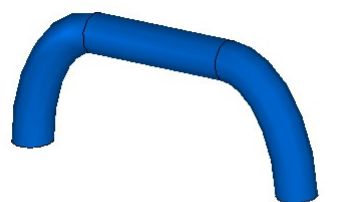


Рис. М.6.2.4.

При $params.way=dmt_Action$ рассматриваемый метод перемещает грани **faces** относительно остальных граней в копии тела **solid**. Перемещение выполняется в направлении вектора $params.direction$ на расстояние его длины. Если грани **faces** связаны с остальными гранями скруглениями, то сначала удаляются скругления, далее перемещаются грани **faces** и затем восстанавливаются скругления на новых местах. На рис. М.6.2.5 и рис. М.6.2.6 приведены тела, построенные путем перемещения указанных граней исходных тел, приведенных на рис. М.6.2.1 и рис. М.6.2.3, соответственно.

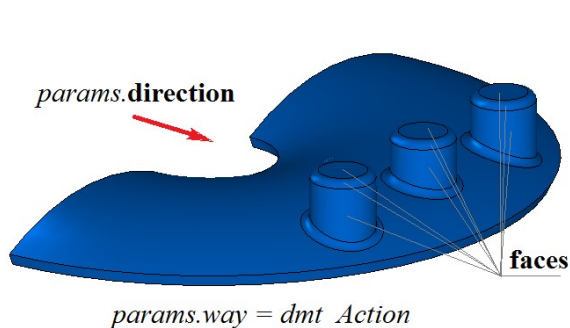


Рис. М.6.2.5.

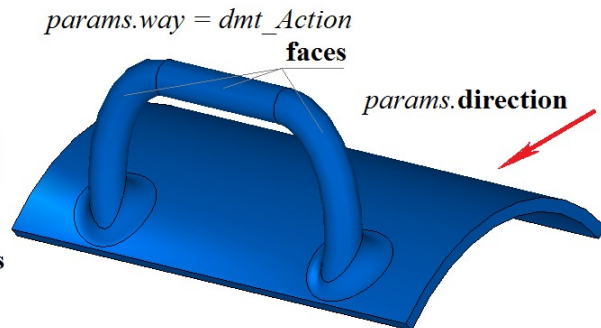


Рис. М.6.2.6.

При $params.way=dmt_Offset$ рассматриваемый метод заменяет грани **faces** в копии тела **solid** эквидистантными гранями. Эквидистантные грани смещены на расстояние длины вектора $params.direction$. Если грани **faces** связаны с остальными гранями скруглениями, то сначала удаляются скругления, далее заменяются грани **faces** эквидистантными и затем восстанавливаются скругления на новых местах. На рис. М.6.2.7 и рис. М.6.2.8 приведены тела, построенные путем замены указанных граней эквидистантными гранями. Исходные тела приведены на рис. М.6.2.1 и рис. М.6.2.3, соответственно.

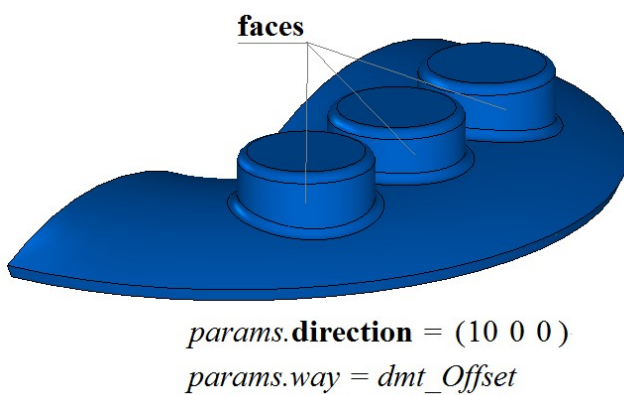


Рис. М.6.2.7.

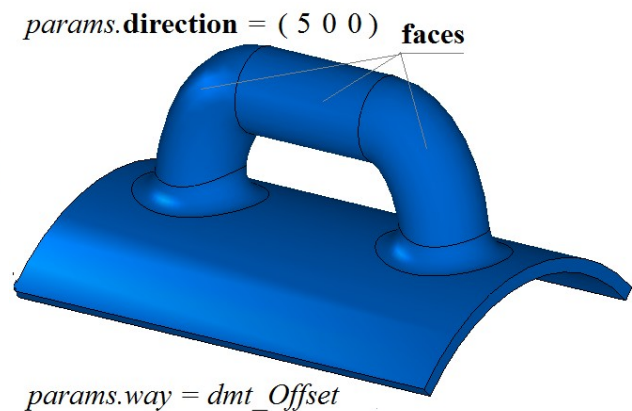


Рис. М.6.2.8.

Если одна из компонент вектора $params.direction$ отрицательная, то эквидистантные грани смещены на отрицательное расстояние. На рис. М.6.2.9 и рис. М.6.2.10 приведены тела, построенные путем замены указанных граней эквидистантными гранями с отрицательным расстоянием. Исходные тела приведены на рис. М.6.2.5 и рис. М.6.2.6, соответственно.

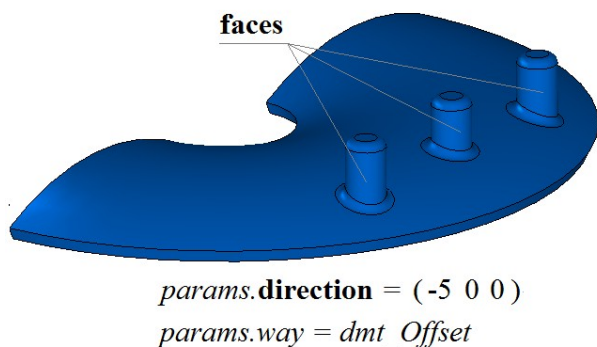


Рис. М.6.2.9.

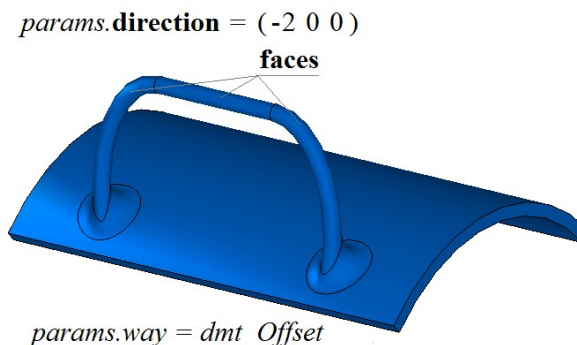


Рис. М.6.2.10.

При $params.way=dmt_Fillet$ рассматриваемый метод изменяет радиусы граней скругления **faces** в копии тела **solid**. Радиус граней скругления увеличивается на длину вектора $params.direction$. Если одна из компонент вектора $params.direction$ отрицательная, то радиус граней скругления уменьшается на длину вектора $params.direction$. На рис. М.6.2.11 и рис. М.6.2.12 приведены тела, построенные путем изменения радиуса указанных граней скругления. Исходные тела приведены на рис. М.6.2.1 и рис. М.6.2.3, соответственно.

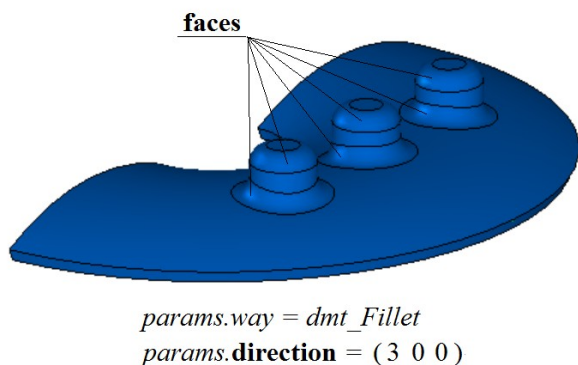


Рис. М.6.2.11.

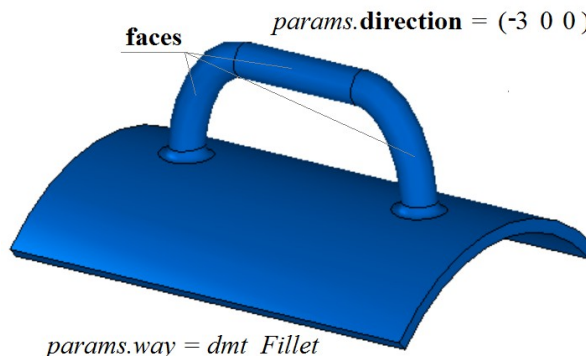


Рис. М.6.2.12.

При $params.way=dmt_Supple$ рассматриваемый метод заменяет грани **faces** в копии тела **solid** деформируемыми гранями, давая возможность дальнейшего редактирования этих граней. Если грани **faces** связаны с остальными гранями скруглениями, то сначала удаляются скругления, далее заменяются грани **faces** на деформируемые грани и затем восстанавливаются скругления на новых местах. На рис. М.6.2.13 приведено исходное тело, а на рис. М.6.2.14 приведены построенное тело, полученное путем замены указанной грани исходного тела деформируемой гранью, и контрольные точки деформируемой грани. Деформируемая грань построена на базе NURBS-поверхности.

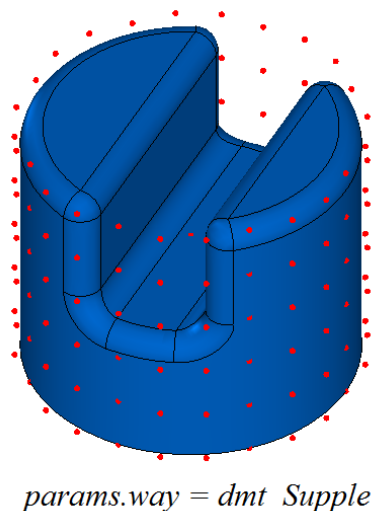
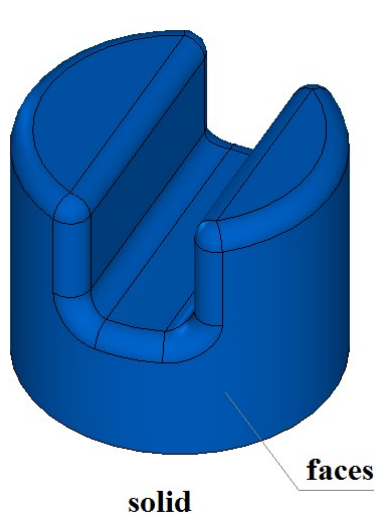
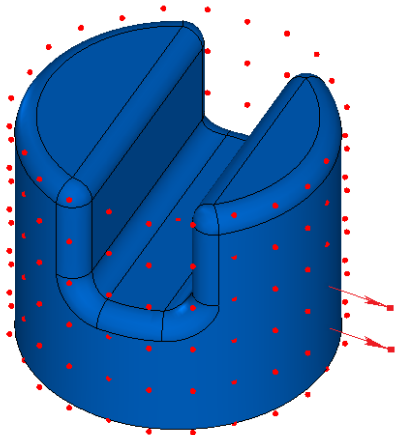


Рис. М.6.2.13.

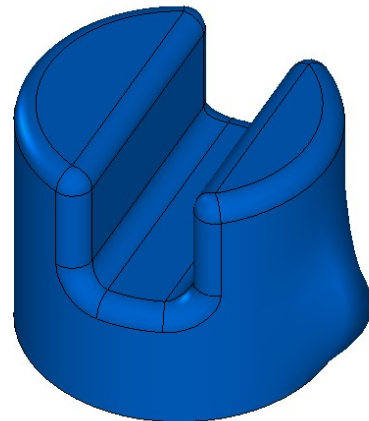
Рис. М.6.2.14.

На рис. М.6.2.15 приведено деформируемое тело с перемещенными контрольными точками деформируемой грани. На рис. М.6.2.16 приведен результат деформирования тела.



params.way = dmt_Supple

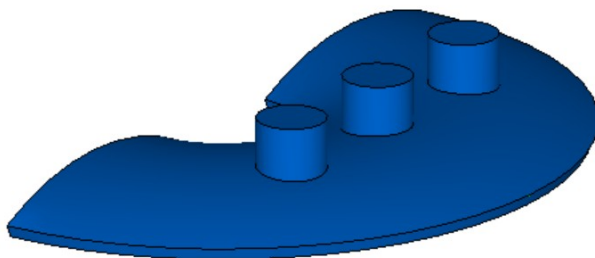
Рис. М.6.2.15.



result

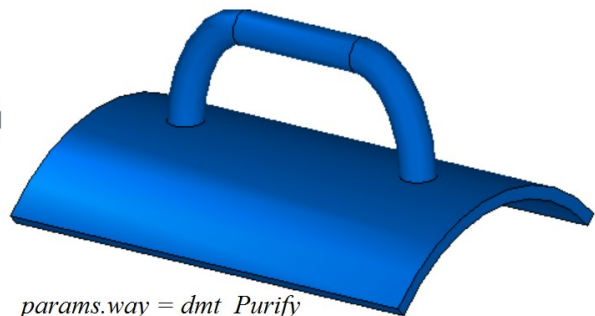
Рис. М.6.2.16.

При *params.way=dmt_Purify* рассматриваемый метод удаляет грани скруглений **faces** в копии тела **solid** и, используя окружение, обрабатывает место удаленных граней, сохраняя замкнутость тела. На рис. М.6.2.17 и рис. М.6.2.18 приведены тела, построенные путем удаления указанных граней скругления. Исходные тела приведены на рис. М.6.2.1 и рис. М.6.2.3, соответственно.



params.way = dmt_Purify

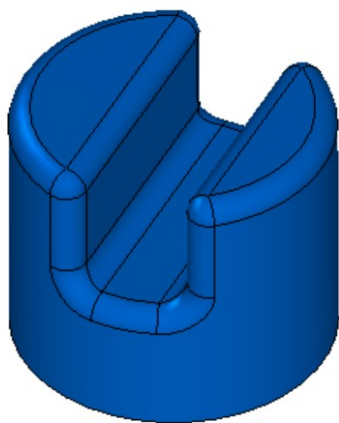
Рис. М.6.2.17.



params.way = dmt_Purify

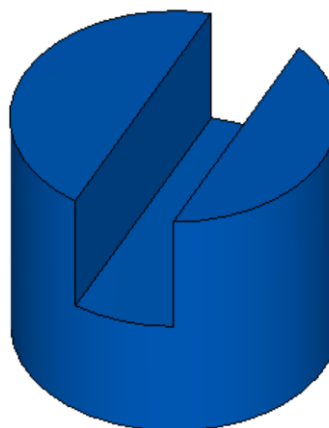
Рис. М.6.2.18.

Если множество граней **faces** пусто, то рассматриваемый метод удаляет грани скругления, радиус которых меньше или равен длине вектора *params.direction*. На рис. М.6.2.19 приведено исходное тело, а на рис. М.6.2.20 приведено построенное тело, полученное путем удаления всех граней скругления, радиус которых не превосходит $|params.direction|$.



solid

Рис. М.6.2.19.



params.way = dmt_Purify

Рис. М.6.2.20.

Метод **FaceModifiedSolid** работает при условии сохранения неизменной топологии редактируемого тела.

Метод **FaceModifiedSolid** добавляет в журнал построенного тела строитель **MbFaceModifiedSolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbFaceModifiedSolid** объявлен в файле `sr_modified_solid.h`.

Тестовое приложение `test.exe` выполняет построение модифицированного тела командами меню «Создать->Тело->Прямым редактированием->Удалением граней», «Создать->Тело->Прямым редактированием->Созданием тела из граней», «Создать->Тело->Прямым редактированием->Перемещением граней», «Создать->Тело->Прямым редактированием->Заменой эквидистантными гранями», «Создать->Тело->Прямым редактированием->Модифицированием скруглений», «Создать->Тело->Прямым редактированием->Деформированием граней», «Создать->Тело->Прямым редактированием->Удалением скруглений».

М.6.3. Построение деформируемого тела

При построении деформируемого тела выполняется несколько действий: замена поверхностей указанных граней на деформируемые поверхности, получение контрольных точек деформируемых поверхностей, перемещение контрольных точек, инициализация деформируемых поверхностей новыми контрольными точками. Для этих действий используются перечисленные ниже методы.

Метод

MbResultType

ModifiedNurbsItem (**MbSolid** & **solid**,
MbcCopyMode sameShell,
 const *NurbsValues* & *params*,
 const *RPAArray*<**MbFace**> & **faces**,
 const *MbSNameMaker* & *names*,
MbSolid *& **result**)

выполняет замену выбранных граней копии тела деформируемыми гранями для последующего редактирования.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- *params* – параметры преобразования,
- **faces** – заменяемые грани тела,
- *names* – именователь граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_direct.h`.

Рассматриваемый метод выполняет копирование тела **solid** и подготовку копии тела для дальнейшего деформирования указанных граней исходного тела.

Параметр **solid** содержит исходное тело, в котором указанные грани будут заменены на деформируемые грани на базе Nurbs-поверхностей. Параметр *sameShell* управляет передачей неизменных граней, ребер и вершин от исходного тела **solid** к построенному телу **result**. Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление *MbeCopyMode* объявлено в файле *mb_enum.h* и описано в параграфе [O.7.9. Копирование множества граней MbFaceShell](#).

Параметр *params* содержит данные *MbNurbsParameters uParameters* и *MbNurbsParameters vParameters*, необходимые для построения Nurbs-поверхностей, см. рис. М.6.3.1. *MbNurbsParameters uParameters* содержит порядок *degree*, число контрольных точек *pointsCount*, область изменения *region*, узловой вектор **knots** для первого параметра Nurbs-поверхностей. *MbNurbsParameters vParameters* содержит порядок *degree*, число контрольных точек *pointsCount*, область изменения *region*, узловой вектор **knots** для второго параметра Nurbs-поверхностей. Флаги *uParameters.useApprox* и *vParameters.useApprox* позволяют строить деформируемые поверхности, которые в начальном состоянии аппроксимируют исходные поверхности граней.

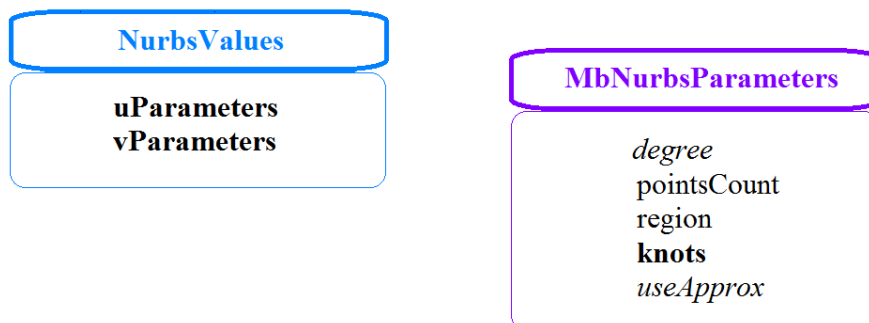


Рис. М.6.3.1.

Параметр **faces** содержит множество граней тела **solid**, которые будут заменены на деформируемые грани на базе Nurbs-поверхностей в построенном теле **result**.

Параметр *names* используется для именования новых граней, ребер, вершин и версионирования операции.

Одноименный описанному выше методу

Метод

MbResultType

ModifiedNurbsItem (*MbSolid* & **solid**,

MbeCopyMode sameShell,

const *NurbsValues* & *params*,

const *MbFace* & **face**,

const *MbSNameMaker* & *names*,

*MbSolid** & **result**)

выполняет замену одной грани тела деформируемой гранью для последующего редактирования. Данный метод отличается от вышеописанного четвертым параметром, который содержит не множество граней, а одну заменяемую грань **face**. Метод объявлен в файле *action_direct.h*.

Построенное тело **result** внешне может быть похожим на исходное тело **solid**, но в отличие от исходного тела может предоставлять Nurbs-поверхности или контрольные точки Nurbs-поверхностей для модификации. Модифицированные контрольные точки и их веса в дальнейшем можно вернуть соответствующим поверхностям граней и, таким образом, изменить форму граней. На рис. М.6.3.2 приведено исходное цилиндрическое тело **solid**, боковая грань которого является цилиндром, а на рис. М.6.3.3 приведено тело **result** с боковой гранью, базирующейся на Nurbs-поверхности. Контрольные точки Nurbs-поверхности также приведены на рис. М.6.3.3.



Рис. М.6.3.2.

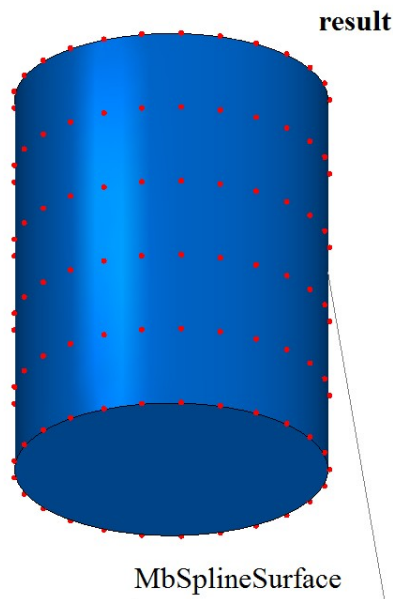


Рис. М.6.3.3.

Методы **ModifiedNurbsItem** добавляют в журнал построенного тела строитель **MbModifiedNurbsItem**, который содержит все необходимые данные для выполнения операции. Строитель **MbModifiedNurbsItem** объявлен в файле `sr_modified_nurns_.h`.

Метод
MbSurface *

GetControlSurface (const **MbFace** & **face**)

копирует Nurbs-поверхность грани **face**, для дальнейшего редактирования.

Входным параметром метода является исходная грань **face**.

При удачной работе метод возвращает указатель на копию поверхности грани **face**, в противном случае метод возвращает нуль.

Метод объявлен в файле `action_direct.h`.

Метод возвращает ненулевой указатель на поверхность, если грань **face** базируется на Nurbs-поверхности. Это требование будет выполняться после успешной работы методов **ModifiedNurbsItem**, описанных в предыдущем параграфе. Метод возвращает указатель на копию базовой поверхности грани **face**, а не на саму базовую поверхность грани. Далее у возвращаемой поверхности можно модифицировать множество контрольных точек и их веса и далее заменить поверхность грани тела модифицированной поверхностью.

Метод
MbResultType

FaceControlPoints (const **MbFace** & **face**,
Array2<**MbCartPoint3D**> & **controlPoints**,
Array2<double> & **weights**)

выдаёт множество контрольных точек поверхности грани и множества их весов.

Входными параметрами метода является исходная грань **face**.

Выходными параметрами метода являются:

- **controlPoints** – контрольные точки грани,
- **weights** – веса контрольных точек.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_direct.h`.

Метод возвращает `rt_Success`, если грань **face** базируется на Nurbs-поверхности. Изменяя положение контрольных точек **controlPoints** и изменяя веса **weights** контрольных точек, можно менять форму поверхности. Рассматриваемый метод следует использовать после успешной работы методов **ModifiedNurbsItem**, описанных выше.

Контрольные точки **controlPoints** передаются в виде прямоугольной матрицы, строки которой соответствуют первому параметру Nurbs-поверхности, а столбцы соответствуют второму параметру Nurbs-поверхности. То есть точки, лежащие в одной строке прямоугольной матрицы **controlPoints**, расположены вдоль координатной линии поверхности с фиксированным вторым параметром Nurbs-поверхности, а точки, лежащие в одном столбце прямоугольной матрицы **controlPoints**, расположены вдоль координатной линии поверхности с фиксированным первым параметром Nurbs-поверхности. На рис. М.6.3.4 приведены контрольные точки на фоне координатных линий поверхности.

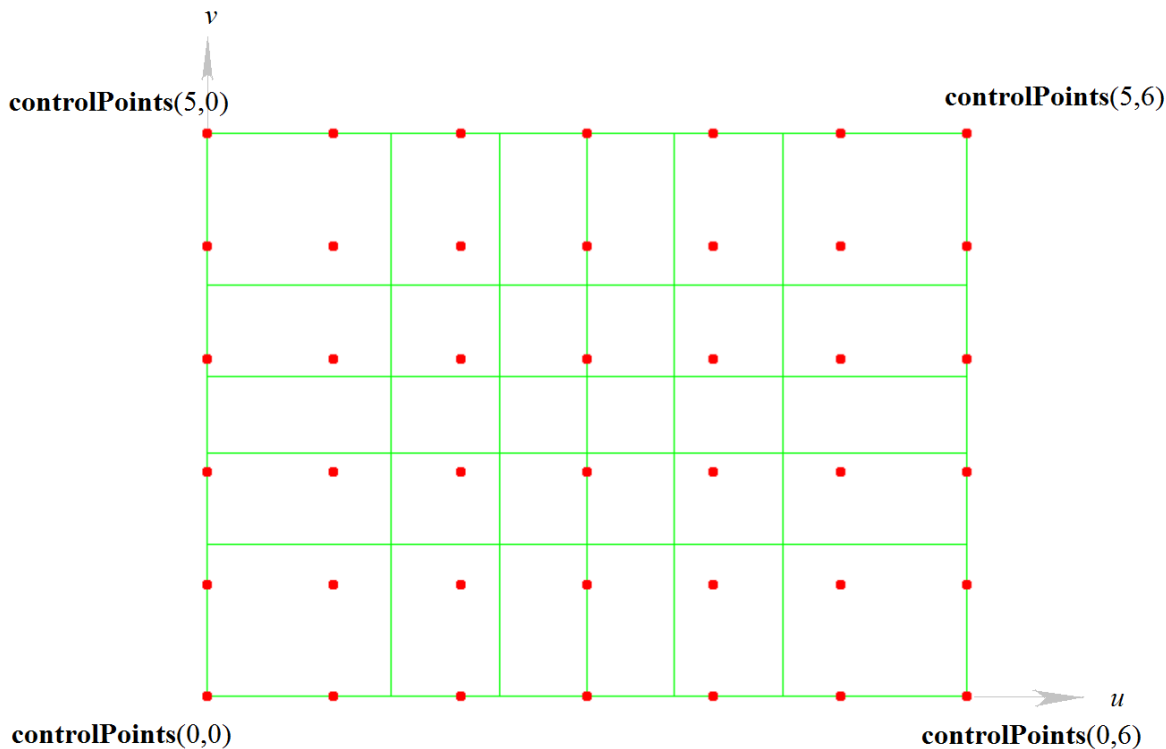
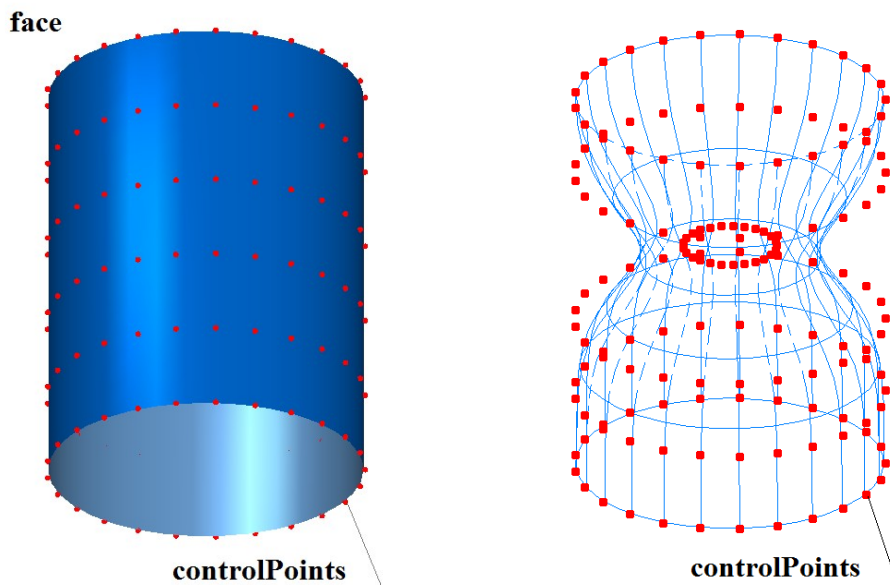


Рис. М.6.3.4.

Прямоугольная матрица *weights* весов соответствует прямоугольной матрице контрольных точек **controlPoints**.

На рис. М.6.3.5 приведена грань **face** и контрольные точки **controlPoints** ее поверхности, а на рис. М.6.3.6 приведены отредактированные контрольные точки **controlPoints** для последующей модификации грань **face**.



Метод
MbResultType

NurbsModification ([MbSolid](#) & **solid**,
MbeCopyMode *sameShell*,
[MbFace](#) * **face**,
[MbSurface](#) & **faceSurface**,
Array2<bool> & *fixedPoints*,
const MbSNameMaker & names,
[MbSolid](#) *& **result**)

выполняет деформирование заданной грани копии исходного тела путём подстановки присланной поверхности в копию изменяемой грани.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **face** – изменяемая грань тела,
- **faceSurface** – поверхность изменяемой грани,
- *fixedPoints* – маска неподвижных контрольных точек поверхности,
- names – именователю граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле *action_direct.h*.

Рассматриваемый метод копирует тело **solid** и подставляет присланную поверхность **faceSurface** в копию грани **face** тела **solid**.

Параметр **solid** содержит исходное тело. Параметр *sameShell* управляет передачей неизмененных граней, ребер и вершин от исходного тела **solid** к построенному телу **result**. Параметр *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. Перечисление MbeCopyMode объявлено в файле *mb_enum.h* и описано в параграфе [O.7.9. Копирование множества граней MbFaceShell](#).

Поверхность **faceSurface** должна быть Nurbs-поверхностью. Параметр *fixedPoints* представляет собой прямоугольную матрицу, строки и столбцы которой соответствуют строкам и столбцам матрицы контрольных точек Nurbs-поверхности **faceSurface**. Элементы матрицы со значением true определяют неподвижные контрольные точки Nurbs-поверхности,

Параметр names используется для именования новых граней, ребер, вершин и версионирования операции.

Одноименный описанному выше методу

Метод
MbResultType

NurbsModification ([MbSolid](#) & **solid**,
MbeCopyMode *sameShell*,
[MbFace](#) * **face**,
const Array2<[MbCartPoint3D](#)> & **controlPoints**,
const Array2<double> & *weights*,
Array2<bool> * *fixedPoints*,
const MbSNameMaker & names,
[MbSolid](#) *& **result**)

выполняет деформирование указанной грани копии исходного тела по заданным контрольным точкам и их весам.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **face** – изменяемая грань тела,
- **controlPoints** – заданные контрольные точки изменяемой грани,
- *weights* – веса контрольных точек,
- *fixedPoints* – маска неподвижных контрольных точек грани,

- `names` – именователь граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_direct.h`.

Рассматриваемый метод следует использовать после успешной работы методов **ModifiedNurbsItem**, **GetControlSurface** и **FaceControlPoints**, описанных выше. Методы **ModifiedNurbsItem** подготавливают грани для модификации. На рис. М.6.3.7 приведено исходное цилиндрическое тело **solid**, боковая грань которого базируется на Nurbs-поверхности, и у которой подлежат модификации контрольные точки. На рис. М.6.3.8 приведено тело **result** с модифицированной боковой гранью, контрольные точки **controlPoints** которой приведены на рис. М.6.3.6.

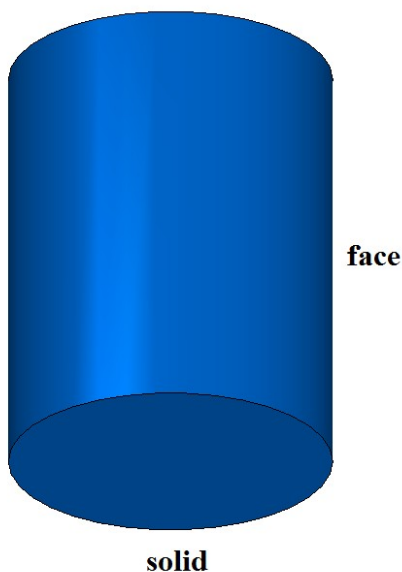


Рис. М.6.3.7.

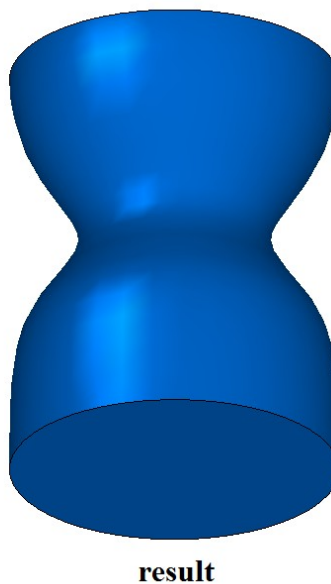


Рис. М.6.3.8.

Методы **NurbsModification** добавляют в журнал построенного тела строитель `MbNurbsModification`, который содержит все необходимые данные для выполнения операции. Строитель `MbNurbsModification` объявлен в файле `cr_modified_nurbs_.h`.

М.6.4. Построение деформируемой призмы

Метод

`MbResultType`

```
NurbsBlockSolid ( const MbPlacement3D & place,
                  double x, double y, double z,
                  bool orientation,
                  const MbSNameMaker & names,
                  SimpleName name,
                  NurbsBlockValues & params,
                  MbSolid *& result )
```

выполняет построение тела в форме прямого параллелепипеда с деформируемыми гранями.

Входными параметрами метода являются:

- **place** – локальная система координат,
- *x* – размер по оси X,
- *y* – размер по оси Y,
- *z* – размер по оси Z,
- *orientation* – ориентация нормалей граней наружу тела (`true`),

- *names* – именователи граней,
- *name* – главное имя,
- *params* – параметры построения граней.

Выходным параметром метода является построенное тело **result** NURBS-поверхностей.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_direct.h*.

Рассматриваемый метод позволяет получить деформируемое тело в форме прямоугольной призмы, размеры которой в локальной системе координат **place** определяют параметры *x*, *y*, *z*. Все грани призмы построены на базе плоских Nurbs-поверхностей, порядок которых и количество контрольных точек определяются параметром *params*. Параметр *NurbsBlockValues* объявлен в файле *op_shell_parameter.h*.

На рис. М.6.4.1 приведена призма **result**, построенная методом **NurbsBlockSolid**, и ее контрольные точки. На рис. М.6.4.2 приведена эта же призма после деформирования путем перемещением контрольных точек граней.

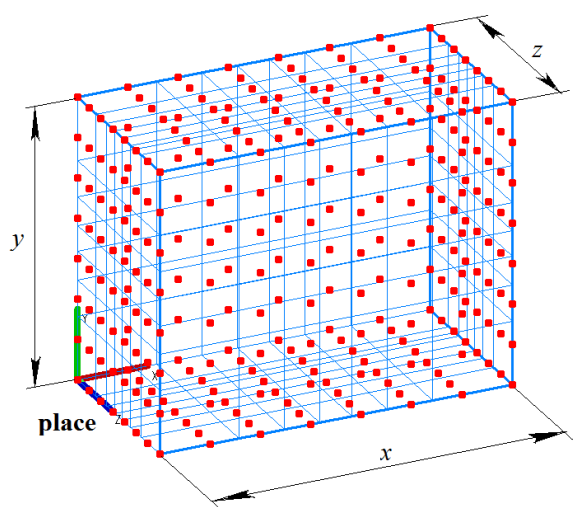


Рис. М.6.4.1.

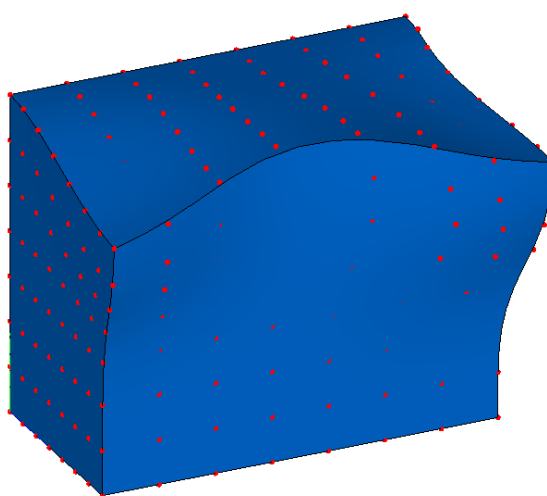


Рис. М.6.4.2.

Метод **NurbsBlockSolid** добавляет в журнал построенного тела строитель *MbNurbsBlockSolid*, который содержит все необходимые данные для выполнения операции. Строитель *MbNurbsBlockSolid* объявлен в файле *cr_nurbs_block_solid.h*.

М.6.5. Построение сглаженной поверхности

Метод
MbResultType
SplineSurfaceSmoothing (const *MbSplineSurface* & **surface**,
 size_t *udegree*,
 size_t *vdegree*,
MbSplineSurface *& **result**)

выполняет сглаживание копии исходной поверхности.

Входными параметрами метода являются:

- **surface** – исходная поверхность,
- *udegree* – параметр сглаживания по первому параметру поверхности.
- *vdegree* – параметр сглаживания по второму параметру поверхности.

Выходным параметром метода является построенная сглаженная поверхность **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_direct.h*.

Рассматриваемый метод позволяет сгладить исходную поверхность **surface**, не меняя ее порядок сплайнов и количество контрольных точек. Сглаживанием управляют параметры *udegree* и *vdegree*, которые должны быть больше соответствующих порядков сплайнов исходной поверхности. Метод возвращает сглаженную копию **result** исходной поверхности.

На рис. М.6.5.1 приведена исходная Nurbs-поверхность. На рис. М.6.5.2 приведена сглаженная поверхность.



surface

Рис. М.6.5.1.



result

Рис. М.6.5.2.

М.7. МЕТОДЫ ПОСТРОЕНИЯ ТЕЛ ИЗ ЛИСТОВОГО МЕТАЛЛА

Геометрическое ядро C3D поддерживает методы, которые позволяют строить модели конструкций из листового металла. Это методы создания листовых тел: пластины и обечайки, а также операции над листовыми телами: сгиб по рёбрам, сгиб вдоль линии, подсечка, сгиб по эскизу, развёртка, замыкание углов, вырез, добавление пластины, штамповка, жалюзи и буртик. Также представлен ряд сервисных функций для работы с листовыми телами. Будем называть широкие грани пластины листовыми, а остальные – боковыми, при этом листовую грань сгиба с меньшим радиусом будем называть внутренней гранью сгиба, а с большим радиусом – внешней гранью сгиба.

М.7.1. Построение листового тела

Метод

MbResultType

```
CreateSheetSolid ( const MbPlacement3D & placement,  
                  RPAArray<MbContour> & contours,  
                  bool unbended,  
                  const MbSheetMetalValues & params,  
                  PArray<MbSNameMaker> * names,  
                  PArray<MbSMBendNames> & bends,  
                  MbSolid *& result )
```

выполняет построение листового тела выдавливанием плоских контуров.

Входными параметрами метода являются:

- **placement** – локальная система координат,
- **contours** – контуры листового тела, заданные в плоскости XY локальной системы координат,
- *unbended* – флаг построения тела в разогнутом состоянии,
- **params** – параметры построения,
- **names** – именователи построенных граней,
- **bends** – параметры формируемых сгибов.

Выходными параметрами метода являются имена листовых граней построенных сгибов, записываемые в **bends** и само тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_sheet.h`.

Если контуры замкнутые, то листовое тело строится на заданную толщину с заданным контуром. Среди замкнутых контуров один контур должен быть внешним, он должен содержать внутри себя остальные внутренние контуры. По внутренним контурам будут сформированы вырезы.

Если контуры незамкнутые, то листовое тело строится выдавливанием на заданные расстояния замкнутых контуров, полученных из незамкнутых контуров скруглением углов между отрезками заданными в **params** или **bends** радиусами. Получившимся контурам придается толщина листового тела в ту или иную сторону. Сторона придания толщины указывается в параметрах операции **params**. Если контур уже содержит дугу, то она должна гладко стыковаться с соседними сегментами. Только в случае незамкнутых контуров возможно построение листового тела со сгибами и только для этого случая имеет смысл флаг *unbended*. В случае, если массив параметров сгибов **bends** пустой, сгибы строятся по единым параметрам, заданным в **params**. В этом случае имена листовых граней построенных сгибов никуда не записываются. Если же массив параметров **bends** заполнен, то количество элементов в этом массиве должно совпадать с количеством формируемых сгибов.

Параметры для сгиба ищутся следующим образом. Каждому контуру из массива **contours** соответствует именователь из массива **names**, а в каждом именователе содержится массив имён сегментов контура. Если сгиб формируется в месте соединения двух отрезков, то параметры этого сгиба находятся в элементе массива **bends**, содержащем имена этих отрезков в полях `segName1` и

segName2, если сгиб формируется на месте дуги, то параметры для него находятся в элементе массива bends, у которого в segName1 лежит имя дуги, а segName2 равен SYMPLENAME_MAX.

На рис. М.7.1.1 приведено листовое тело, построенное по нескольким замкнутым эскизам.

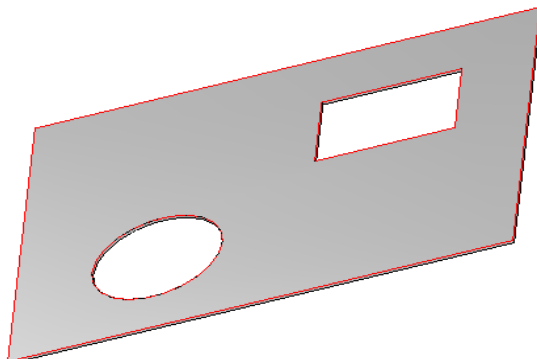


Рис. М.7.1.1

На рис. М.7.1.2 приведено листовое тело, построенное по незамкнутому эскизу, содержащему дугу и пару стыкующихся по углом отрезков.

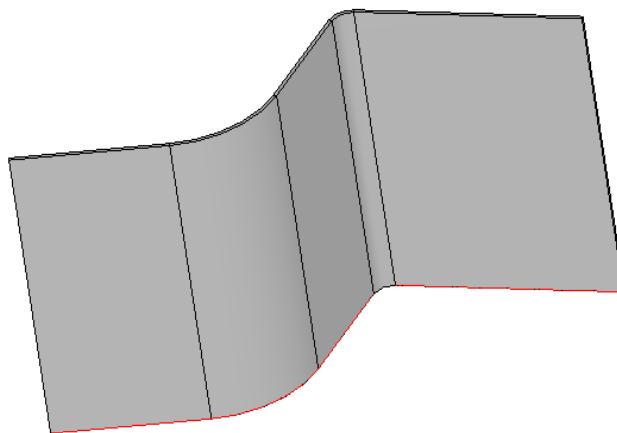


Рис. М.7.1.2

Метод **CreateSheetSolid** добавляет в журнал построенного тела строитель MbSheetMetalSolid, который содержит все необходимые данные для выполнения операции. Строитель MbSheetMetalSolid объявлен в файле cr_sheet_metal_solid.h.

М.7.2. Построение обечайки

Метод
MbResultType

CreateRuledSolid (const MbRuledSolidValues & params,
const MbSNameMaker & names,
PArray<MbSMBendNames> & bends,
MbContour* & **resultContour**,
MbSolid * & **result**)

выполняет построение обечайки.

Входными параметрами метода являются:

- params – параметры построения,
- names – именованье построенных граней,
- bends – параметры формируемых сгибов.

Выходными параметрами метода являются имена листовых граней построенных сгибов, записываемые в bends, **resultContour** – скруглённый по заданным параметрам контур params.contour1 и само тело **result**. Также операция заполняет массивы параметров разбиения контуров, если они были пустыми.

При удачной работе метод возвращает rt_Success, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле action_sheet.h.

Обечайка строится по одному или двум плоским контурам.

В случае одного контура обечайка строится выдавливанием, возможно с уклоном скруглённого по параметрам контура и приданием получившейся поверхности толщины. Если контур замкнут, то в нём создаётся зазор в указанном в параметрах операции месте. В случае построения с уклоном возможно создание конических сгибов или сохраняющих радиус цилиндрических сгибов в месте скруглений изначального контура. Также можно воспользоваться сегментацией дуг контура, когда дуга заменяется набором аппроксимирующих её отрезков.

В случае построения обечайки по двум контурам, сегменты контуров соединяются, в общем случае, линейчатыми поверхностями, которым потом придаётся толщина. Контур, во избежание перекрученных поверхностей, разбиваются на более мелкие сегменты в автоматическом режиме или по указанию пользователя. Также как и при построении по одному эскизу для обеспечения разгибаемости создаётся зазор в указанном в параметрах месте.

На рис. М.7.2.1 приведена обечайка, построенная без сохранения радиуса сгиба.

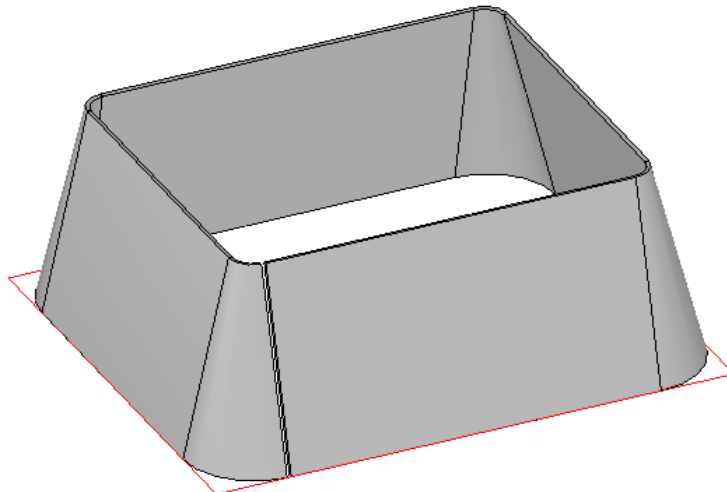


Рис. М.7.2.1

На рис. М.7.2.2 приведена обечайка, построенная с сохранением радиуса сгиба.

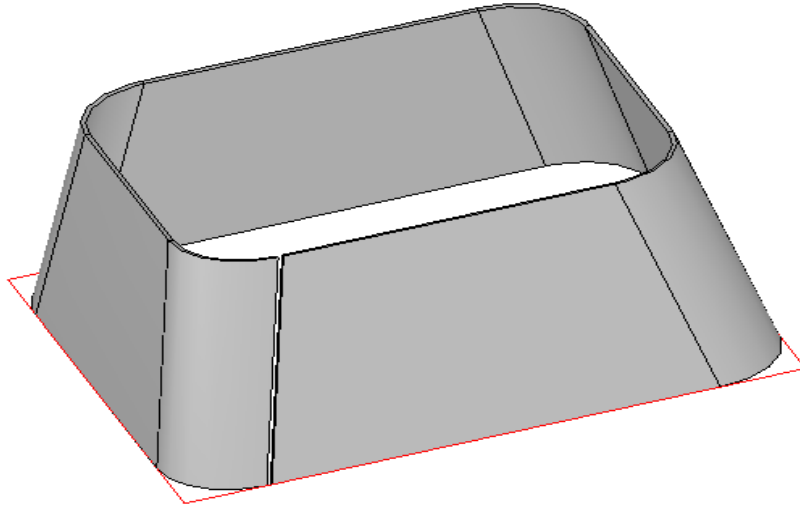


Рис. М.7.2.2

На рис. М.7.2.3 приведена обечайка, построенная с сегментацией дуг второго контура.

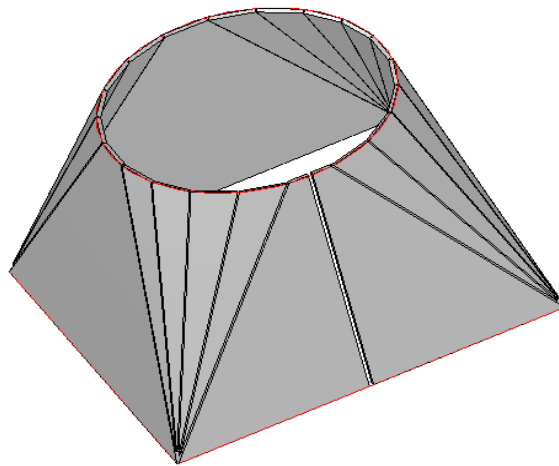


Рис. М.7.2.3

Метод **CreateRuledSolid** добавляет в журнал построенного тела строитель `MbRuledSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbRuledSolid` объявлен в файле `cr_sheet_ruled_solid.h`.

М.7.3. Построение сгиба листового тела по линии

Метод
`MbResultType`
BendSheetSolidOverSegment (`MbSolid & solid`,
`MbeCopyMode sameShell`,
`const RPAArray<MbFace> & bendingFaces`,

[MbCurve3D](#) & **curve**,
 bool *unbended*,
 const MbBendOverSegValues & params,
 MbSNameMaker & names,
[MbSolid](#) *& **result**)

выполняет сгиб листового тела вдоль линии, лежащей на листовой грани тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **bendingFaces** – изгибаемые грани,
- **curve** – прямолинейная кривая, вдоль которой выполнить сгиб,
- *unbended* – флаг построения элемента в разогнутом состоянии,
- params – параметры построения,
- names – именователь построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_sheet.h`.

Прямолинейной кривой **curve** может быть отрезок, лежащий на плоских гранях **bendingFaces**, либо прямая. Грани **bendingFaces** должны располагаться на общей для них плоскости. Отрезок **curve** может лежать одновременно на нескольких плоских гранях, но сгибы будут формироваться только на тех из них, которые занесены в массив **bendingFaces**.

На рис. М.7.3.1 приведено листовое тело с выполненной операцией «сгиб по линии».

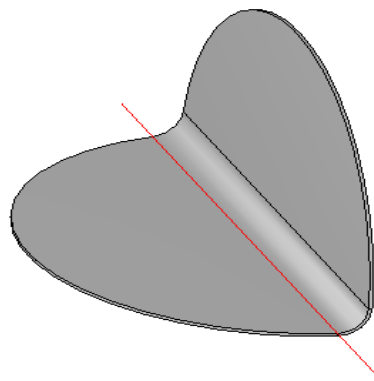


Рис. М.7.3.1

Метод **BendSheetSolidOverSegment** добавляет в журнал построенного тела строитель `MbBendOverSegSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbBendOverSegSolid` объявлен в файле `sr_sheet_bend_over_seg_solid.h`.

М.7.4. Построение подсечки листового тела

Метод

`MbResultType`

SheetSolidJog ([MbSolid](#)& **solid**,
 MbCopyMode *sameShell*,
 const RPAArray<[MbFace](#)> & **bendingFaces**,
[MbCurve3D](#) & **curve**,
 bool *unbended*,
 const MbJogValues & params,


```

const MbBendValues & secondBendParams,
MbSNameMaker & names,
RPAArray<MbFace> & firstBendFaces,
RPAArray<MbFace> & secondBendFaces,
MbSolid *& result )

```

выполняет подсечку тела из листового материала вдоль прямой линии.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **bendingFaces** – изгибаемые грани,
- **curve** – прямолинейная кривая, вдоль которой выполнить подсечку,
- *unbended* – флаг построения элемента в разогнутом состоянии,
- *params* – параметры построения,
- *secondBendParams* – параметры второго сгиба,
- *names* – именованье построенных граней.

Выходными параметрами метода являются:

- **result** – построенное тело,
- **firstBendFaces** – листовые грани сгибов, примыкающие к неподвижной части базовых граней,
- **secondBendFaces** – листовые грани сгибов, поднятых над базовыми гранями.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_sheet.h*.

Линией может служить отрезок, лежащий на плоских гранях **bendingFaces**, либо прямая. Грани **bendingFaces** должны располагаться на общей для них плоскости. Отрезок **curve** может лежать одновременно на нескольких плоских гранях, но сгибы будут формироваться только на тех из них, которые занесены в массив **bendingFaces**. Подсечка выполняется в виде двух смещённых друг относительно друга сгибов по линии. Формируемые при этом листовые грани сгибов возвращаются в массивах **firstBendFaces** и **secondBendFaces**. На рис. М.7.4.1 приведено листовое тело с выполненной операцией «подсечка».

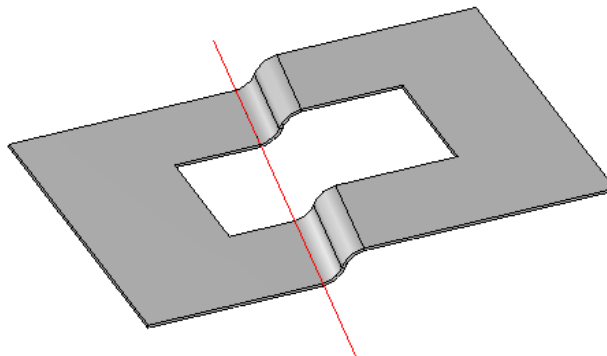


Рис. М.7.4.1

Метод **SheetSolidJog** добавляет в журнал построенного тела строитель *MbJogSolid*, который содержит все необходимые данные для выполнения операции. Строитель *MbJogSolid* объявлен в файле *sr_stamp_jog_solid.h*.

М.7.5. Сгиб разогнутого листового тела

Метод

MbResultType

BendSheetSolid ([MbSolid](#) & **solid**,
MbcCopyMode *sameShell*,
const RPAArray<MbSheetMetalBend> & bends,
const [MbFace](#) & **face**,
const [MbCartPoint](#) & **point**,
MbSNameMaker & names,
[MbSolid](#) *& **result**)

переводит указанные сгибы в согнутое состояние.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- bends – множество сгибов, состоящих из массивов пар граней: внутренних и внешних граней сгиба,
- **face** – грань, остающаяся неподвижной,
- **point** – точка параметрической области поверхности сгибовой грани **face**,
- names – именователи построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления MbResultType.

Метод объявлен в файле `action_sheet.h`.

Метод сгибает разогнутые сгибы bends листового тела относительно неподвижной грани **face**. Если **face** – листовая грань, принадлежащая одному из сгибов bends, то сгиб осуществляется так, чтобы неподвижной осталась плоскость, касательная к поверхности, лежащей под **face** в точке **point**. Как правило, в каждом элементе массива bends лежит только одна пара листовых граней – внутренняя и внешняя листовые грани сгиба, но в случае линейчатой обечайки по двум эскизам может возникнуть ситуация, когда цепочка таких смежных сгибов может быть согнута только одновременно, тогда элемент массива bends должен включать в себя всю такую цепочку в виде пар соответствующих друг другу внешних и внутренних граней составного сгиба.

На рис. М.7.5.1 приведено листовое тело в разогнутом состоянии.

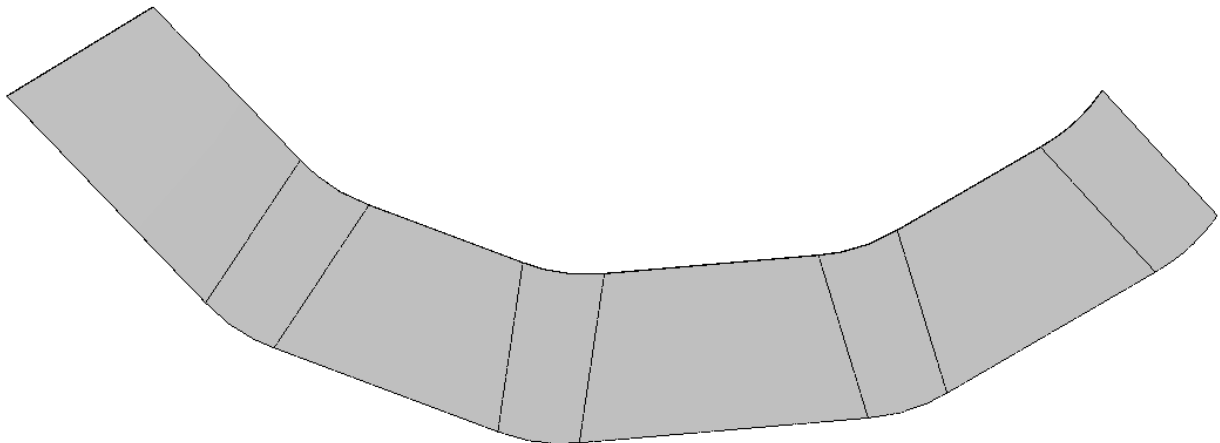


Рис. М.7.5.1

На рис. М.7.5.2 приведено листовое тело с рис. М.7.5.1 после выполнения операции «согнуть».

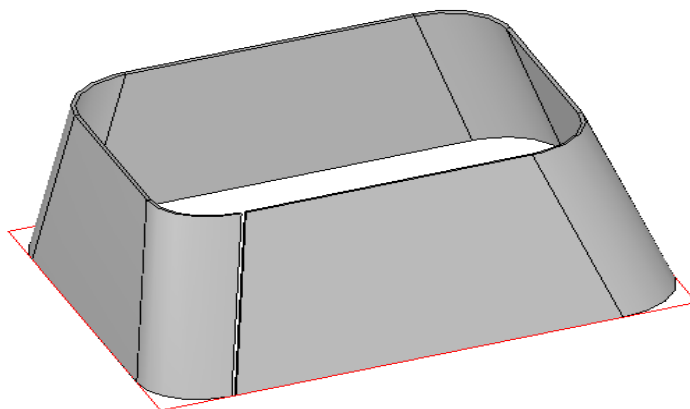


Рис. М.7.5.2

Метод **BendSheetSolid** добавляет в журнал построенного тела строитель **MbBendUnbendSolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbBendUnbendSolid** объявлен в файле `cr_sheet_bend_unbend_solid.h`.

М.7.6. Разгиб сгибов листового тела

Метод
MbResultType
UnbendSheetSolid (**MbSolid** & **solid**,
MbCopyMode *sameShell*,
const **RPAArray**<**MbSheetMetalBend**> & **bends**,
const **MbFace** & **face**,
const **MbCartPoint** & **point**,
MbSNameMaker & **names**,
MbSolid* & **result**)

разгибает сгибы листового тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **bends** – множество сгибов, состоящих из пар граней: внутренней и внешней граней сгиба,
- **face** – грань, остающаяся неподвижной,
- **point** – точка параметрической области поверхности сгибовой грани **face**,
- **names** – именованье построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления **MbResultType**.

Метод объявлен в файле `action_sheet.h`.

Метод разгибает сгибы **bends** листового тела относительно неподвижной грани **face**. Если **face** – листовая грань, принадлежащая одному из сгибов **bends**, то разгиб осуществляется так, чтобы неподвижной осталась плоскость, касательная к поверхности, лежащей под **face** в точке **point**. Как правило, в каждом элементе массива **bends** лежит только одна пара листовых граней – внутренняя и внешняя листовые грани сгиба, но в случае линейчатой обечайки по двум эскизам может возникнуть ситуация, когда цепочка таких смежных сгибов может быть разогнута только одновременно, тогда элемент массива **bends** должен включать в себя всю такую цепочку в виде пар соответствующих друг другу внешних и внутренних граней составного сгиба.

На рис. М.7.6.1 приведено листовое тело в согнутом состоянии.

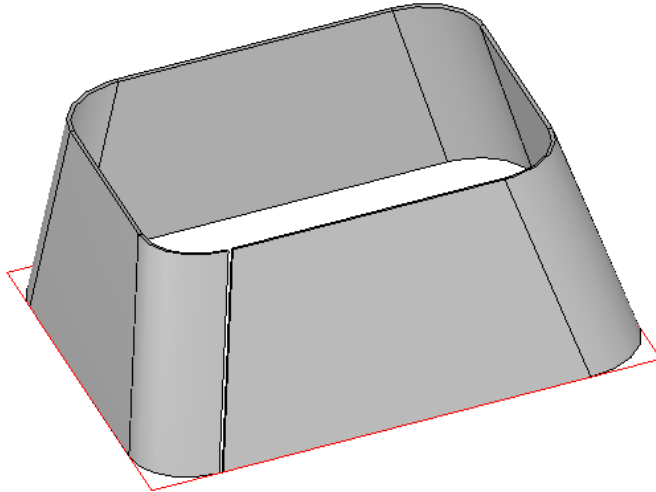


Рис. М.7.6.1

На рис. М.7.6.2 приведено листовое тело с рис. М.7.6.1 после выполнения операции «разогнуть».

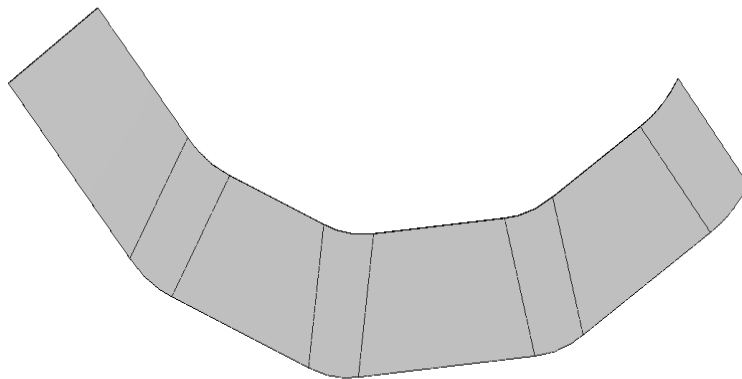


Рис. М.7.6.2

Метод **UnbendSheetSolid** добавляет в журнал построенного тела строитель `MbBendUnbendSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbBendUnbendSolid` объявлен в файле `cr_sheet_bend_unbend_solid.h`.

М.7.7. Добавление пластины к листовому телу

Метод
`MbResultType`
SheetSolidPlate (`MbSolid` & `solid`,
MbeCopyMode `sameShell`,
const `MbPlacement3D` & `placement`,
RPAArray<`MbContour`> & `contours`,

```
const MbSheetMetalValues & params,  
PArray<MbSNameMaker> * names,  
MbSolid *& result )
```

выполняет добавление пластины к листовому телу.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **placement** – локальная система координат,
- **contours** – контуры пластины, заданные в плоскости XY локальной системы координат,
- *params* – параметры построения,
- *names* – именователь построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_sheet.h*.

Пластина строится по одному или нескольким замкнутым не пересекающимся контурам, причём среди них может быть несколько внешних.

На рис. М.7.7.1 приведено листовое тело после выполнения операции «пластина».

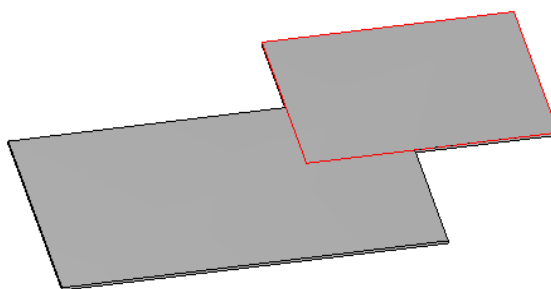


Рис. М.7.7.1

Метод [SheetSolidPlate](#) добавляет в журнал построенного тела строитель *MbSheetMetalSolid*, который содержит все необходимые данные для выполнения операции. Строитель *MbSheetMetalSolid* объявлен в файле *cr_sheet_sheet_metal_solid.h*.

М.7.8. Построение выреза в листовом теле

Метод

MbResultType

```
SheetSolidHole ( MbSolid & solid,  
MbcCopyMode sameShell,  
const MbPlacement3D & placement,  
RArray<MbContour> & contours,  
const MbSheetMetalValues & params,  
bool difference,  
PArray<MbSNameMaker> * names,  
MbSolid *& result)
```

вырезает отверстия в листовом теле по замкнутым контурам.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **placement** – локальная система координат,
- **contours** – контуры выреза/пересечения, заданные в плоскости XY локальной системы координат,
- *params* – параметры построения,
- *difference* – флаг способа построения: отверстие (true), пересечение (false),
- *names* – именованье построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_sheet.h*.

Метод примечателен тем, что помимо обычного булевого выреза имеет опцию выреза по толщине, когда вырез как бы огибает все сгибы листового тела, то есть листовое тело, построенное по контурам из параметров операции вырезания с толщиной исходного тела, перед вырезанием повторяет все встречающиеся на пути сгибы исходного тела.

На рис. М.7.8.1 приведено листовое тело после выполнения операции «вырез в листовом теле» с опцией «по толщине».

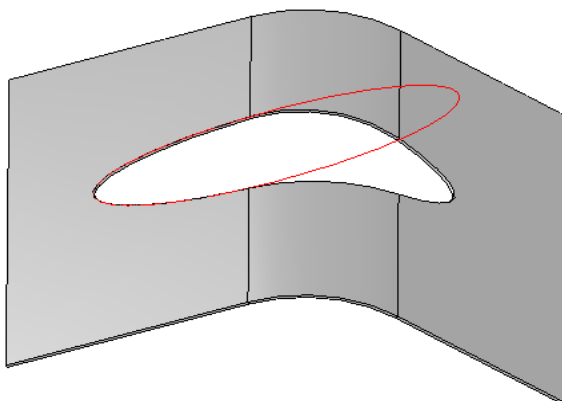


Рис. М.7.8.1

Метод **SheetSolidHole** добавляет в журнал построенного тела строитель *MbSheetMetalSolid*, который содержит все необходимые данные для выполнения операции. Строитель *MbSheetMetalSolid* объявлен в файле *cr_sheet_sheet_metal_solid.h*.

М.7.9. Построение сгиба листового тела на ребрах

Метод

MbResultType

BendSheetSolidByEdges (*MbSolid* & **solid**,
 const *MbeCopyMode* *sameShell*,
 const *RPAArray*<*MbCurveEdge*> & **edges**,
 const bool *unbended*,
 const *MbBendByEdgeValues* & *params*,
MbSNameMaker & *names*,
MbSolid *& **result**)

выполняет построение сгиба листового тела на указанных прямолинейных ребрах.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **edges** – множество рёбер для построения сгибов,
- *unbended* – флаг построения сгибов в разогнутом состоянии,
- *params* – параметры построения,
- *names* – именователь построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_sheet.h*.

Сгиб строится согласно заданным параметрам на одном или нескольких рёбрах, принадлежащих плоской грани листового тела, путём присоединения к торцу пластины сгиба с плоским продолжением или без него с возможными расширениями или уклонами боковых сторон в зависимости от параметров операции. Для цепочек смежных рёбер возможно построение замыкания углов.

На рис. М.7.9.1 приведено листовое тело после выполнения операции «сгиб».

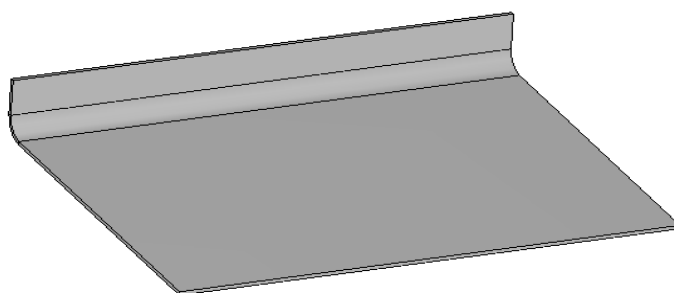


Рис. М.7.9.1

На рис. М.7.9.2 приведено листовое тело после выполнения операции «сгиб» с опцией «уклон» слева и опцией «расширение продолжения» справа.

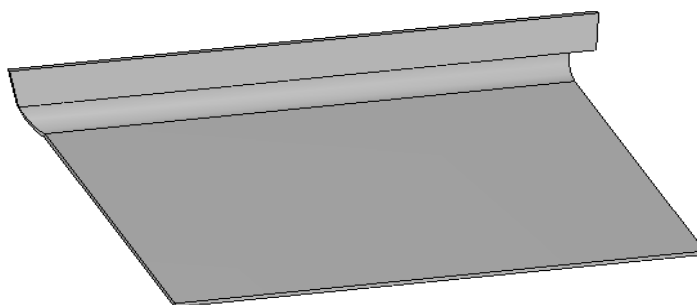


Рис. М.7.9.2

На рис. М.7.9.3 приведено листовое тело с выполненной операцией «сгиб» с опцией «освобождение сгиба».

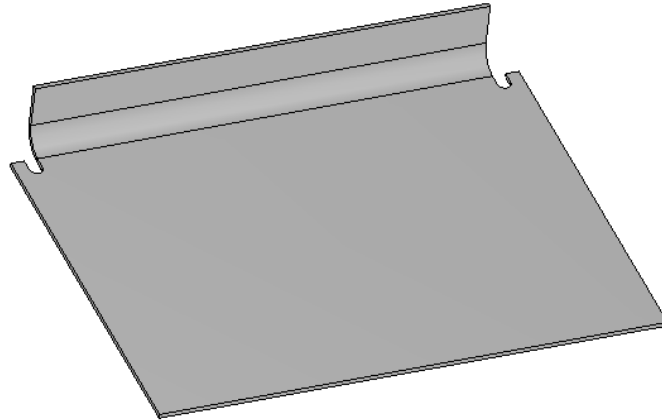


Рис. М.7.9.3

На рис. М.7.9.4 приведено листовое тело после выполнения операции «сгиб» с множественным выбором рёбер и опцией «замыкание угла».

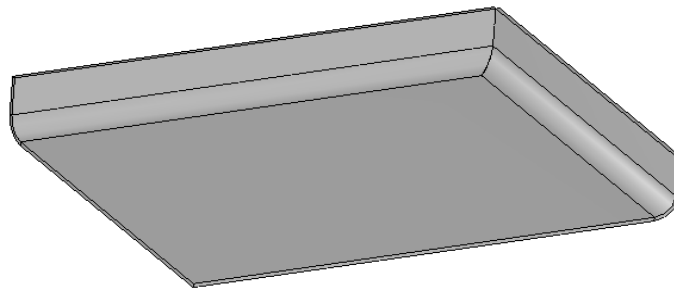


Рис. М.7.9.4

Метод **BendSheetSolidByEdges** добавляет в журнал построенного тела строитель **MbBendByEdgeSolid**, который содержит все необходимые данные для выполнения операции. Строитель **MbBendByEdgeSolid** объявлен в файле `sr_sheet_bend_by_edge_solid.h`.

М.7.10. Построение сгиба по эскизу

Метод
MbResultType

SheetSolidJointBend ([MbSolid](#)& **solid**,
const MbeCopyMode *sameShell*,
const [MbPlacement3D](#) & **placement**,
const [MbContour](#) & **contour**,
const RPAArray<[MbCurveEdge](#)> & **edges**,
const bool *unbended*,
const MbJointBendValues & **params**,
MbSNameMaker & **names**,
PArray< PArray<MbSMBendNames>> & **bends**,
[MbSolid](#) *& **result**)

выполняет построение сгиба листового тела по эскизу.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **placement** – локальная система координат,
- **contour** – контур сгибов, заданный в плоскости XY локальной системы координат,
- **edges** – множество рёбер для построения сгибов,
- *unbended* – флаг построения сгибов в разогнутом состоянии,
- *params* – параметры построения,
- *names* – именователь построенных граней.

Выходными параметрами метода являются построенное тело **result** и формируемые сгибы **bends**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_sheet.h`.

Комбинированный сгиб листового тела, он же сгиб по эскизу, может строиться на одном или нескольких соседних прямолинейных рёбрах одной листовой грани или нескольких листовых граней, расположенных через сгиб. Эскиз, состоящий из отрезков и дуг, должен лежать в плоскости, перпендикулярной одному из рёбер построения и одним концом располагаться на его проекции на эту плоскость.

Данный эскиз применяется к каждому ребру, участвующему в построении. По нему и его копиям для всех рёбер строятся листовые тела с формированием сгибов на дугах и негладких стыковках прямолинейных сегментов контура, а также между контуром и листовой пластиной в случае их негладкой стыковки. Построенные тела объединяются с исходным телом, и затем осуществляются замыкания углов согласно заданным параметрам. После выполнения операции в множестве **bends** находятся все созданные ей сгибы.

На рис. М.7.10.1 приведено листовое тело после выполнения операции «сгиб по эскизу».

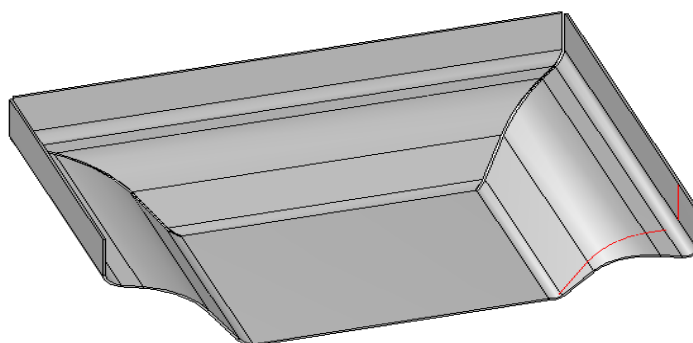


Рис. М.7.10.1

Метод **SheetSolidJointBend** добавляет в журнал построенного тела строитель `MbJointBendSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbJointBendSolid` объявлен в файле `cr_sheet_joint_bend_solid.h`.

М.7.11. Замыкание угла листового тела

Метод

`MbResultType`

CloseCorner (`MbSolid` & `solid`,
MbcCopyMode *sameShell*,
`MbCurveEdge` * `edgePlus`,
`MbCurveEdge` * `edgeMinus`,
const `MbClosedCornerValues` & *params*,
`MbSNameMaker` & *names*,

MbSolid *& result)

выполняет замыкание угла листового тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **edgePlus** – ребро сгиба, условно принятое за положительное,
- **edgeMinus** – ребро сгиба, условно принятое за отрицательное,
- *params* – параметры построения,
- *names* – именованье построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_sheet.h*.

Если на соседних рёбрах листовой грани или на рёбрах разделённых сгибом построены два сгиба, то между ними образуется угол, который можно затянуть материалом, расширив соответствующие стороны этих сгибов, что и осуществляет данная операция. В необходимых случаях вместо расширения выполняется подрезка. В параметрах можно выставить величину зазора и виды замыкания отдельно для сгибов и отдельно для их плоских продолжений. Также, в случае замыкания угла между сгибами, построенными на смежных рёбрах одной грани, можно выбрать несколько способов обработки угла.

На рис. М.7.11.1 приведено листовое тело с двумя сгибами после выполнения операции «замыкание угла» без обработки сгиба.

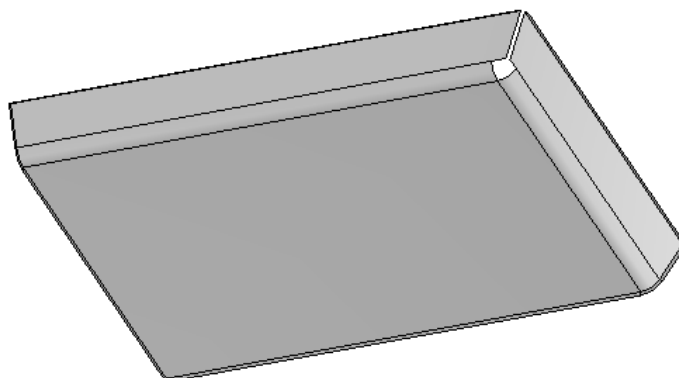


Рис. М.7.11.1

На рис. М.7.11.2 приведено листовое тело с двумя сгибами после выполнения операции «замыкание угла» с плотным замыканием, стыком по кромке и зазором.

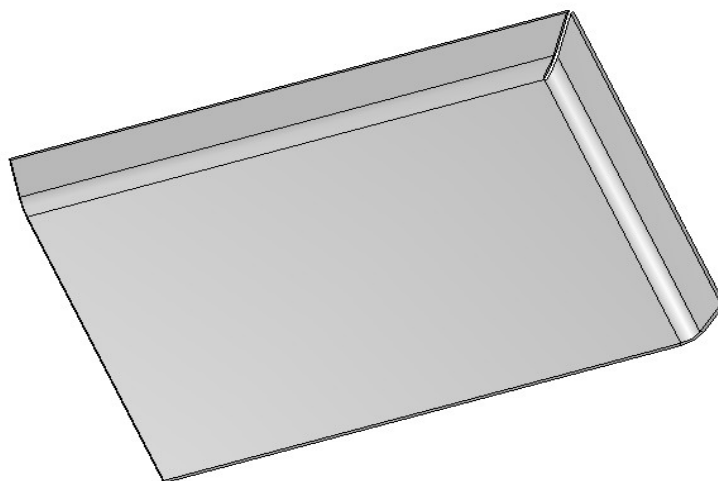


Рис. М.7.11.2

На рис. М.7.11.3 приведено листовое тело с двумя сгибами после выполнения операции «замыкание угла» с плотным замыканием, зазором и круговой обработкой угла.

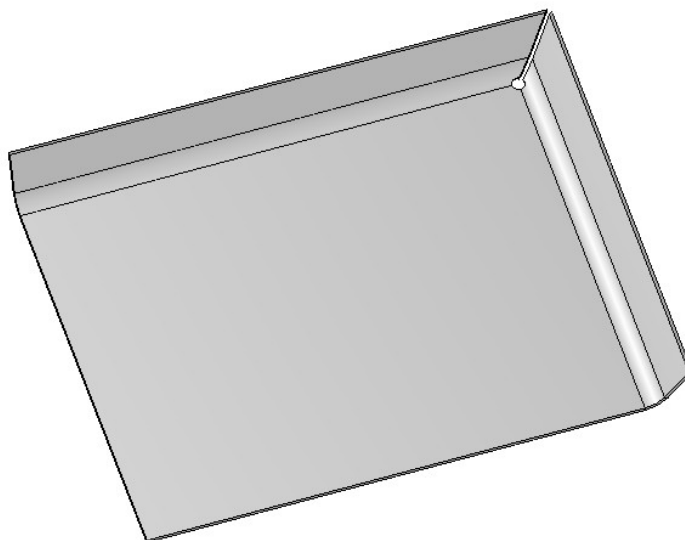


Рис. М.7.11.3

Метод **CloseCorner** добавляет в журнал построенного тела строитель `MbClosedCornerSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbClosedCornerSolid` объявлен в файле `cr_sheet_closed_corner_solid.h`.

М.7.12. Построение штампованного тела

Метод
`MbResultType`
Stamp (`MbSolid& solid`,
 `MbcCopyMode sameShell`,
 const `MbFace & face`,

```
const MbPlacement3D & placement,  
const MbContour & contour,  
const MbStampingValues & params,  
MbSNameMaker & names,  
MbSolid*& result )
```

выполняет штамповку заданной формы на указанной грани.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **face** – грань штамповки,
- **placement** – локальная система координат,
- **contour** – контур штамповки, заданный в плоскости XY локальной системы координат,
- *params* – параметры построения,
- *names* – именованье построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_sheet.h*.

Штамповка строится по одному замкнутому или незамкнутому контуру, лежащему на плоской листовой грани. Замкнутый эскиз может лежать на листовой грани полностью или частично, а незамкнутый должен начинаться и заканчиваться за пределами грани. Штамповка подрезается границами листовой грани, на которой располагается эскиз. Эскиз определяет форму доньшка штамповки. Штамповка в зависимости от параметров может быть открытой. Это означает, что листовая пластина пробивается по контуру насквозь.

На рис. М.7.12.1 приведено листовое тело после выполнения операции «закрытая штамповка».

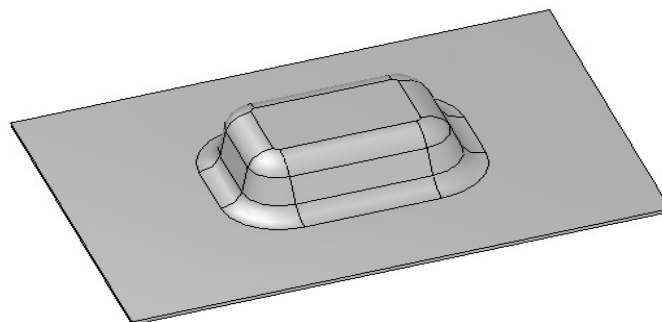


Рис. М.7.12.1

На рис. М.7.12.2 приведено листовое тело после выполнения операции «открытая штамповка».

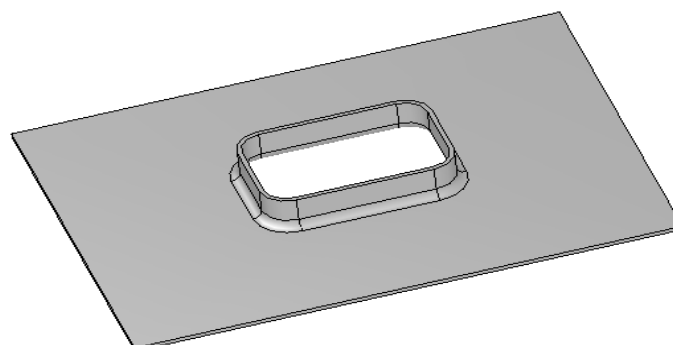


Рис. М.7.12.2

На рис. М.7.12.3 приведено листовое тело после выполнения операции «закрытая штамповка» по незамкнутому эскизу.

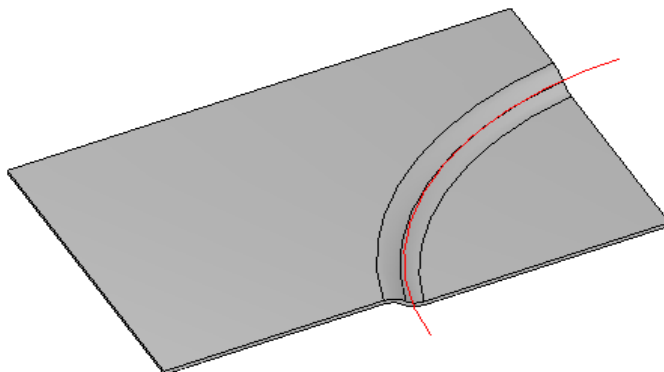


Рис. М.7.12.3

Метод **Stamp** добавляет в журнал построенного тела строитель `MbStampSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbStampSolid` объявлен в файле `cr_stamp_solid.h`.

М.7.13. Построение буртика листового тела

Метод

`MbResultType`

```
CreateBead ( MbSolid & solid,  
             MbeCopyMode sameShell,  
             const MbFace& face,  
             const MbPlacement3D & placement,  
             const RPAArray<MbContour> & contours,  
             const MbBeadValues & params,  
             MbSNameMaker & names,  
             MbSolid *& result )
```

выполняет построение буртика листового тела.

Входными параметрами метода являются:

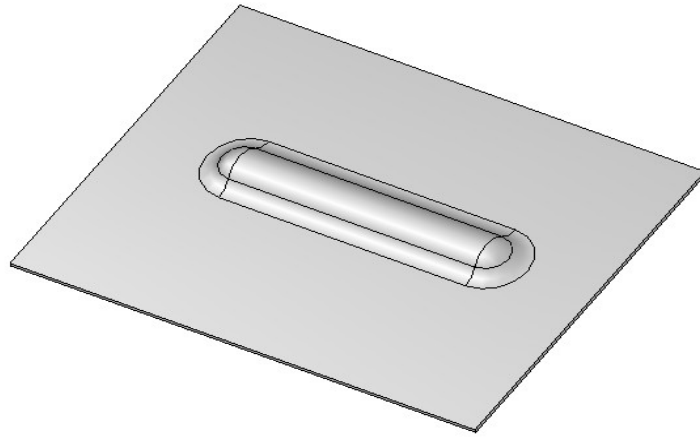
- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **face** – грань буртика,
- **placement** – локальная система координат,
- **contour** – контуры буртика, заданные в плоскости XY локальной системы координат,
- **params** – параметры построения,
- **names** – именователь построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_sheet.h`.

Буртик строится по одному или нескольким замкнутым или незамкнутым контурам, лежащим на плоской листовой грани. Если контур выходит за пределы этой грани, то буртик



подрезается её границами. Буртик по незамкнутому контуру в начале и в конце имеет законцовки, вид которых задаётся в параметрах операции.

На рис. М.7.13.1 приведено листовое тело после выполнения операции «буртик» с круглой формой сечения и закрытой законцовкой.

Рис. М.7.13.1

На рис. М.7.13.2 приведено листовое тело после выполнения операции «буртик» с U-образной формой сечения и открытой законцовкой.

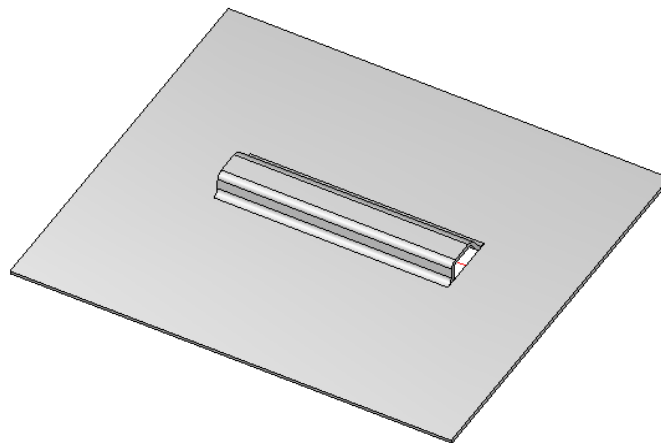


Рис. М.7.13.2

На рис. М.7.13.3 приведено листовое тело после выполнения операции «буртик» с V-образной формой сечения и рубленной законцовкой.

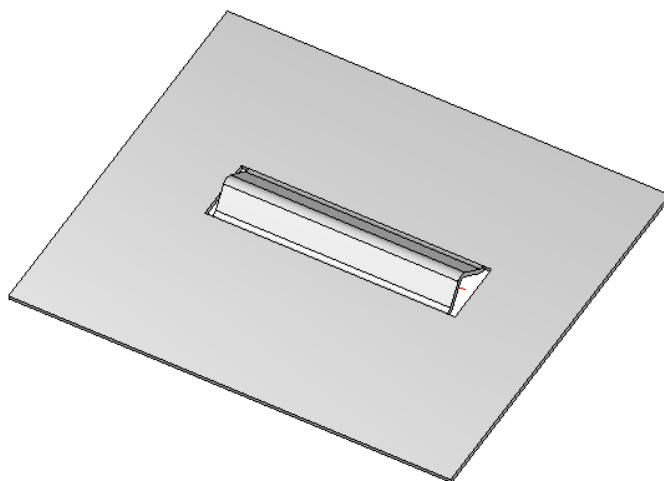


Рис. М.7.13.3

На рис. М.7.13.4 приведено листовое тело после выполнения операции «буртик» с круглой формой сечения по эскизу, выходящему за пределы листа.

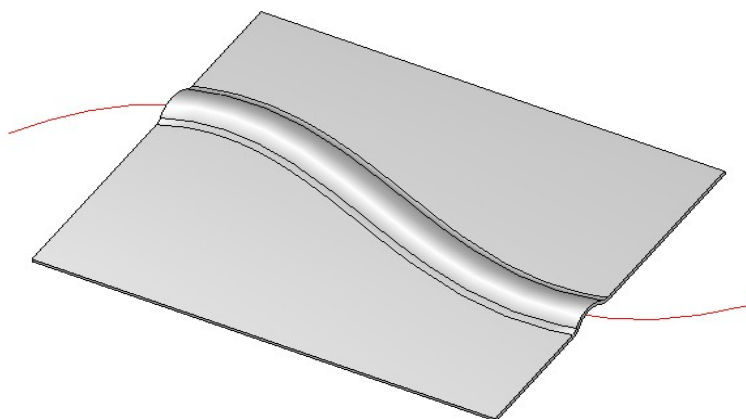


Рис. М.7.13.4

Метод **CreateBead** добавляет в журнал построенного тела строитель MbBeadSolid, который содержит все необходимые данные для выполнения операции. Строитель MbBeadSolid объявлен в файле `cg_stamp_bead_solid.h`.

М.7.14. Построение жалюзи листового тела

Метод
MbResultType
CreateJalousie ([MbSolid](#) & **solid**,
MbeCopyMode *sameShell*,
const [MbFace](#) & **face**,
const [MbPlacement3D](#) & **placement**,

```
const RPAArray<MbLineSegment> & segments,  
const MbJalousieValues & params,  
MbSNameMaker & names,  
MbSolid *& result )
```

выполняет построение жалюзи листового тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **face** – грань жалюзи,
- **placement** – локальная система координат,
- **segments** – отрезки жалюзи, заданные в плоскости XY локальной системы координат,
- *params* – параметры построения,
- *names* – именовател построенных граней.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает *rt_Success*, в противном случае метод возвращает код ошибки из перечисления *MbResultType*.

Метод объявлен в файле *action_sheet.h*.

Жалюзи строятся на одном или нескольких отрезках, лежащих на плоской листовой грани. Жалюзи не могут выходить за пределы грани и пересекаться сами с собой. Жалюзи бывают двух видов: вытяжка и подрезка. Вытяжка имеет вид половины элемента, разрезанного вдоль прямолинейного буртика, а подрезка имеет вид отогнутой пластины.

На рис. М.7.14.1 приведено листовое тело после выполнения операции «жалюзи» способом «вытяжка».

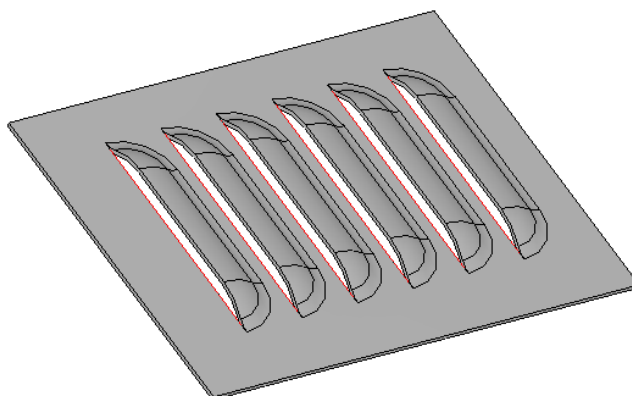


Рис. М.7.14.1

На рис. М.7.14.2 приведено листовое тело после выполнения операции «жалюзи» способом «подрезка».

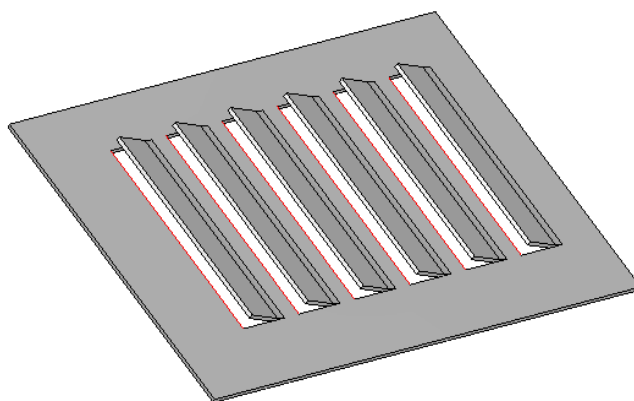


Рис. М.7.14.2

Метод **CreateJalousie** добавляет в журнал построенного тела строитель `MbJalousieSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbJalousieSolid` объявлен в файле `cr_stamp_jalousie_solid.h`.

М.7.15. Восстановление рёбер сгибов листового тела

Метод
`MbResultType`
RestoreSideEdges (`MbSolid& solid`,
`MbeCopyMode sameShell`,
`const RPAArray<MbFace> & faces`,
`const bool strict`,
`PArray<MbSheetMetalBend> & bends`,
`MbSNameMaker & names`,
`MbSolid *& result`)

восстанавливает боковые рёбра сгибов листового тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- *sameShell* – вариант копирования исходного тела,
- **faces** – множество внешних граней сгибов, у которых требуется восстановить боковые рёбра,
- *strict* – флаг строгости восстановления (*false* – восстанавливать, где возможно),
- *names* – именователь построенных граней.

Выходными параметрами метода являются построенное тело **result** и сгибы **bends**, у которых восстановили боковые рёбра.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод объявлен в файле `action_sheet.h`.

Метод служит для восстановления боковых границ сгибов после построений, которые могли их удалить, например, вырез или скругление. Сейчас метод восстановления боковых рёбер вызывается автоматически с флагом *strict* равным *false* в булевой операции и в операции скругления.

На рис. М.7.15.1 изображены боковые рёбра сгиба.

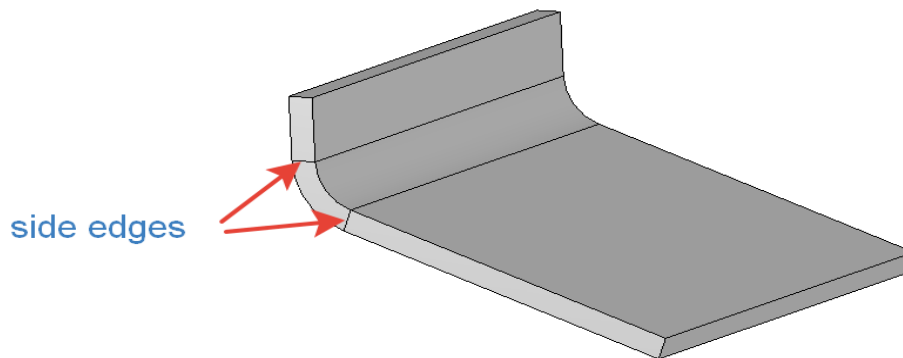


Рис. М.7.15.1

Метод **RestoreSideEdges** добавляет в журнал построенного тела строитель `MbRestoredEdgesSolid`, который содержит все необходимые данные для выполнения операции. Строитель `MbRestoredEdgesSolid` объявлен в файле `cr_sheet_restored_edges_solid.h`.

М.7.16. Разделение сгибов листового тела на группы

Метод

`bool`

SeparateBendsBySubshells (`const MbSolid & solid`,
`const RPAArray<MbSheetMetalBend> & bends`,
`const MbName & fixedFaceName`,
`RPAArray< RPAArray<MbSheetMetalBend> > & bendsGroups`,
`RPAArray<const MbFace> & fixedFaces`)

разделяет сгибы по принадлежности разным топологическим частям листового тела.

Входными параметрами метода являются:

- **solid** – листовое тело,
- **bends** – сгибы тела,
- **fixedFaceName** – имя неподвижной грани.

Выходными параметрами метода являются множество сгибов **bendsGroups**, разделённых на группы по принадлежности разным топологическим частям тела, и соответствующие этим частям неподвижные грани **fixedFaces**.

При удачной работе метод возвращает `true`, в противном случае метод возвращает `false`.

Метод объявлен в файле `action_sheet.h`.

Метод используется после операций, разрезающих тело на несколько несвязанных между собой частей для того, чтобы определить какие сгибы и куски неподвижной грани в какую часть попали. В результате работы метода формируется взаимоднозначное соответствие групп сгибов и соответствующих этим группам кусков неподвижной грани.

На рис. М.7.16.1 изображены две группы сгибов со своими неподвижными гранями.

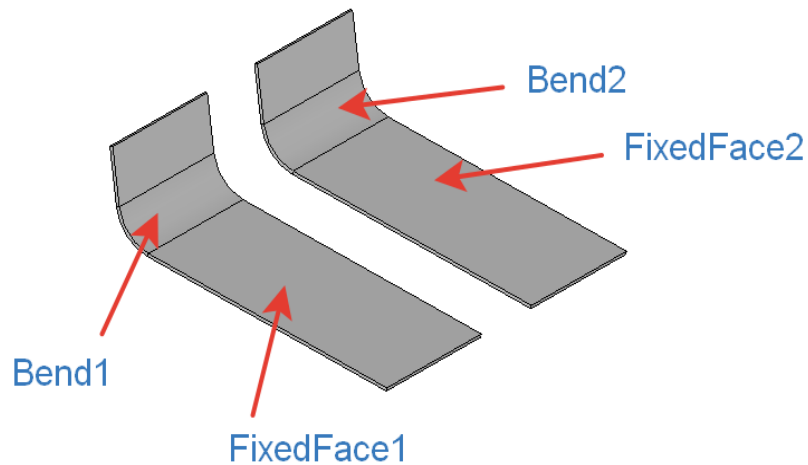


Рис. М.7.16.1

М.7.17. Разделение сгибов листового тела на пары

Метод

bool

CollectBends (const [MbFaceShell](#) & **shell**,
const RArray<[MbFace](#)> & **innerFaces**,
const RArray<[MbFace](#)> & **outerFaces**,
PArray<[MbSheetMetalBend](#)> & **bends**)

разделяет грани сгибов листового тела на пары.

Входными параметрами метода являются:

- **shell** – множество граней листового тела,
- **innerFaces** – внутренние грани сгибов,
- **outerFaces** – внешние грани сгибов.

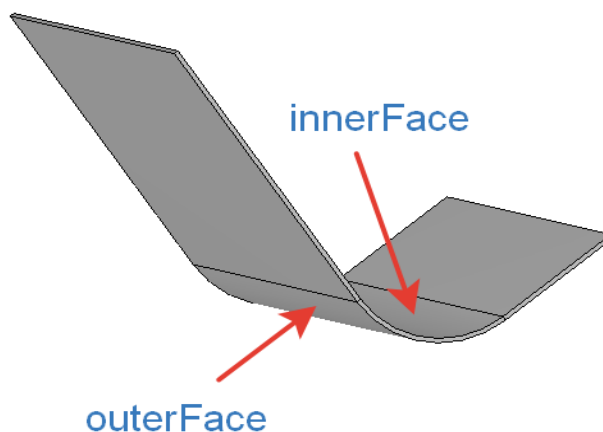
Выходным параметром метода является множество найденных пар граней, составляющих сгибы, **bends**.

При удачной работе метод возвращает true, в противном случае метод возвращает false.

Метод объявлен в файле `action_sheet.h`.

Ищутся составляющие сгиб внутренняя и внешняя грань среди неупорядоченного набора внешних и внутренних граней сгибов, по ним формируется сгиб, который добавляется в множество **bends**.

На рис. М.7.17.1 изображены внутренняя и внешняя грани сгиба.



М.7.18. Проверка грани на возможность быть фиксированной

Метод

bool

IsSuitableForFixed (const [MbFace](#)& **face**)

выполняет проверку грани на возможность использования в качестве фиксированной при сгибе или разгибе сгиба листового тела.

Входным параметром метода является проверяемая грань **face**.

Метод возвращает true, если грань может быть выбрана в качестве фиксированной для сгиба и разгиба листового тела, в противном случае метод возвращает false.

Метод объявлен в файле action_sheet.h.

М.7.19. Поиск граней для кривой

Метод

void

FindCurveFaces (const RPAarray<[MbFace](#)> & **faces**,
const [MbCurve3D](#) & **curve**,
RPAarray<[MbFace](#)> & **curveFaces**)

выполняет поиск граней, на которых лежит указанная прямолинейная кривая.

Входными параметрами метода являются:

- **faces** – множество граней для поиска,
- **curve** – прямолинейная кривая, лежащая на некоторых гранях множества.

Выходным параметром метода является множество граней **curveFaces**, на которых лежит кривая **curve**.

Метод не возвращает значений.

Метод объявлен в файле action_sheet.h.

На рис. М.7.19.1 изображены две листовые грани, лежащие под отрезком.

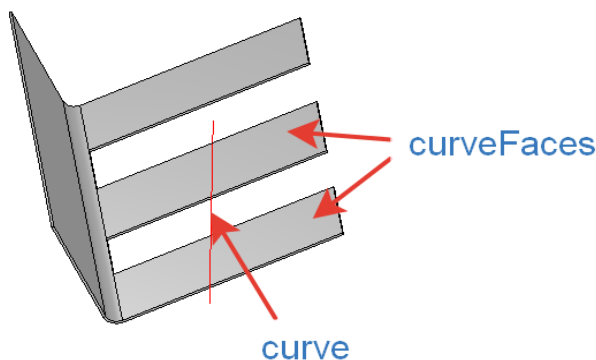


Рис. М.7.19.1

М.7.20. Поиск листовой грани тела

Метод

[MbFace](#)*

FindSheetFace (const MbCurveEdge & **edge**)

выполняет поиск верхней или нижней грани листового тела, содержащей заданное ребро.

Входным параметрами метода является ребро листовой грани **edge**.

При удачной работе метод возвращает указатель на найденную грань, в противном случае метод возвращает NULL.

Метод объявлен в файле `action_sheet.h`.

Поиск верхней или нижней грани листового тела выполняется среди двух стыкующихся в ребре **edge** граней.

На рис. М.7.20.1 изображена листовая грань, смежная с ребром.

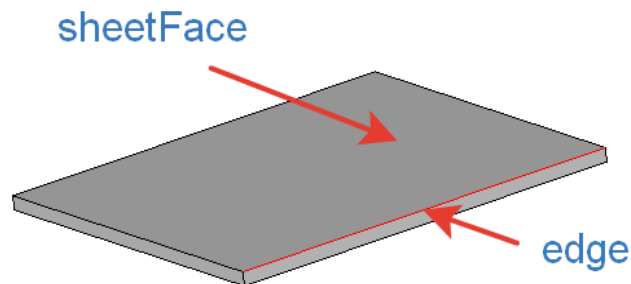


Рис. М.7.20.1

М.7.21. Поиск парной грани сгиба листового тела

Метод

MbFace *

FindPairBendFace (const MbFace & **face**)

выполняет поиск парной грани сгиба листового тела.

Входным параметром метода является грань сгиба листового тела **face**.

При удачной работе метод возвращает указатель на найденную грань, в противном случае метод возвращает NULL.

Метод объявлен в файле `action_sheet.h`.

Метод ищет грань сгиба листового тела, противоположную заданной грани.

На рис. М.7.21.1 изображены грань сгиба и парная ей грань.

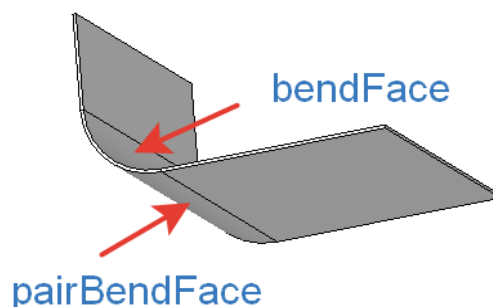


Рис. М.7.21.1

М.7.22. Поиск плоской грани листового тела

Метод

MbFace *

GetPairPlanarFaceByEdge (const MbCurveEdge & **edge**,
const double *begDistance*,
const double *endDistance*)

выполняет поиск плоской грани листового тела по заданному ребру и расстояниям от его концов.

Входными параметрами метода являются:

- **edge** – ребро, по которому искать.
- *begDistance* – расстояние от начала ребра,
- *endDistance* – расстояние от конца ребра.

При удачной работе метод возвращает указатель на найденную грань, в противном случае метод возвращает NULL.

Метод объявлен в файле `action_sheet.h`.

Метод ищет парную грань листового тела для содержащей ребро **edge** листовой грани. Используется в операции сгиб на ребре для определения толщины листа в месте подклейки сгиба. Метод применяется для многотолщинных листовых тел в условиях, когда выбранной грани листового тела соответствует несколько парных ей граней, находящихся на разном расстоянии от неё. Положительные расстояния *begDistance* и *endDistance* означают отступ наружу вдоль ребра, а отрицательные расстояния *begDistance* и *endDistance* означают отступ внутрь вдоль ребра.

На рис. М.7.22.1 изображена плоская листовая грань, найденная по ребру и отступам от его концов.

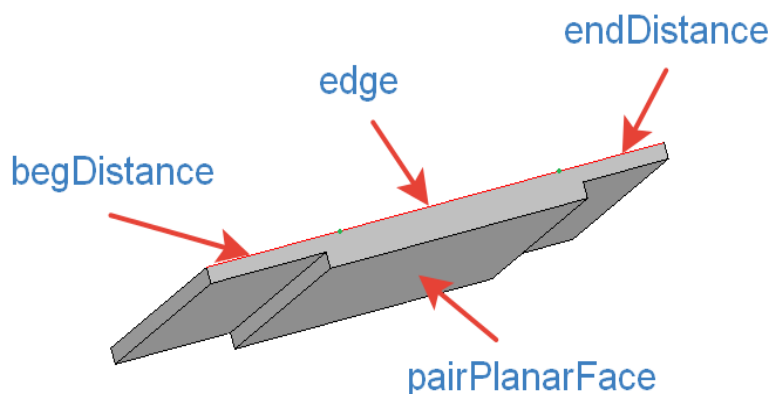


Рис. М.7.22.1

М.7.23. Поиск парной грани листового тела

Метод

MbFace *

GetPairPlanarFaceByCurve (const MbFace & **face**,
const MbCurve3D & **curve**)

выполняет поиск плоской парной грани листового тела по лежащей на грани **face** прямолинейной кривой **curve**.

Входными параметрами метода являются:

- **face** – плоская грань листового тела,
- **curve** – лежащая на ней прямолинейная кривая.

При удачной работе метод возвращает указатель на найденную грань, в противном случае метод возвращает NULL.

Метод объявлен в файле `action_sheet.h`.

Метод поиска парной грани листового тела является вспомогательной для метода построения сгиба по линии. Метод применяется для многотолщинных листовых тел в условиях, когда выбранной листовой грани соответствует несколько парных ей граней, находящихся на разном расстоянии от неё.

На рис. М.7.23.1 изображена плоская листовая грань, найденная по противоположной ей грани и лежащей на ней кривой.

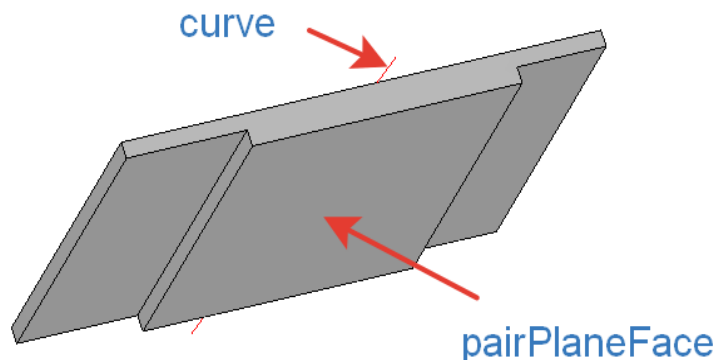


Рис. М.7.23.1

Метод

`MbFace` *

`GetPairPlanarFaceByContour` (const `MbFaceShell` & `shell`,
const `MbFace` & `face`,
const `MbPlacement3D` & `place`,
const `RPAArray`<const `MbCurve` > & `segments`)

выполняет поиск плоской парной грани листового тела по множеству сегментов контура.

Входными параметрами метода являются:

- **shell** – множество граней листового тела,
- **face** – грань листового тела,
- **place** – локальная система координат на грани **face**,
- **segments** – кривые, лежащие в плоскости XY локальной системы координат **place**.

При удачной работе метод возвращает указатель на найденную грань, в противном случае метод возвращает NULL.

Метод объявлен в файле `action_sheet.h`.

Метод поиска парной грани листового тела является вспомогательной для функций построения листового тела, базирующихся на контурах. Метод применяется для многотолщинных листовых тел в условиях, когда выбранной листовой грани соответствует несколько парных ей граней, находящихся на разном расстоянии от неё.

На рис. М.7.23.2 изображена плоская листовая грань, найденная по противоположной ей грани и лежащему на ней контуру.

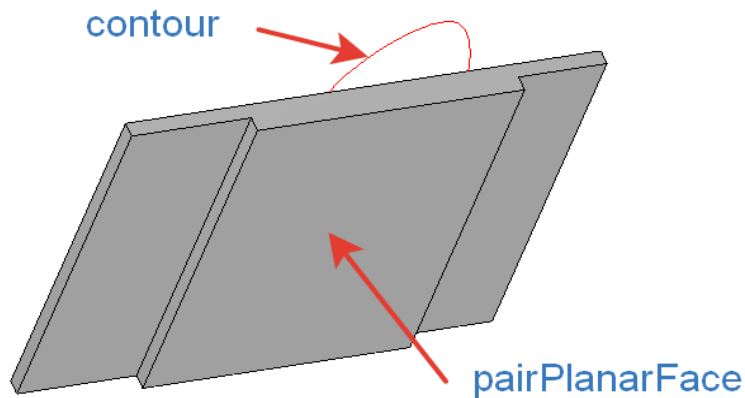


Рис. М.7.23.2

Метод

MbFace *

GetPairPlanarFace (const MbFaceShell * **shell**,
const MbFace & **face**)

выполняет поиск парной грани по заданной грани листового тела.

Входными параметрами метода являются:

- **shell** – множество граней листового тела,
- **face** – грань листового тела,

При удачной работе метод возвращает указатель на найденную грань, в противном случае метод возвращает NULL.

Метод объявлен в файле action_sheet.h.

Поиск осуществляется сначала через рёбра внешнего цикла грани **face**, в случае неудачи - через вершины этого цикла, и если грань не найдена, то перебором по всем связным граням или граням множества **shell**. В последнем случае предпочтение отдаётся более близко расположенным граням.

На рис. М.7.23.3 изображена плоская листовая грань, найденная по противоположной ей плоской листовой грани.

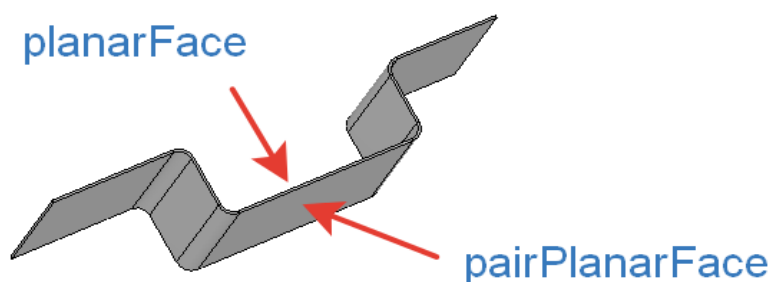


Рис. М.7.23.3

М.7.24. Определение расстояния между подобными гранями

Метод

double

GetDistanceIfSameAndOpposite (const MbFace & **face1**,
const MbFace * **face2**)

определяет расстояние между парой подобных листовых граней.

Входными параметрами метода являются:

- **face1** – первая грань,
- **face2** – вторая грань.

При удачной работе метод возвращает расстояние между гранями, в противном случае метод возвращает ноль.

Метод объявлен в файле `action_sheet.h`.

Подобными считаются пары плоских, цилиндрических и конических граней, а в случае линейчатого сгиба, линейчатые в паре с офсетными гранями, у которых нормали коллинеарны и противоположно направлены. Расстояние считается положительным, если грани располагаются со стороны, противоположной направлению нормали, и отрицательным в противном случае.

На рис. М.7.24.1 изображена пара подобных граней с расстоянием между ними.

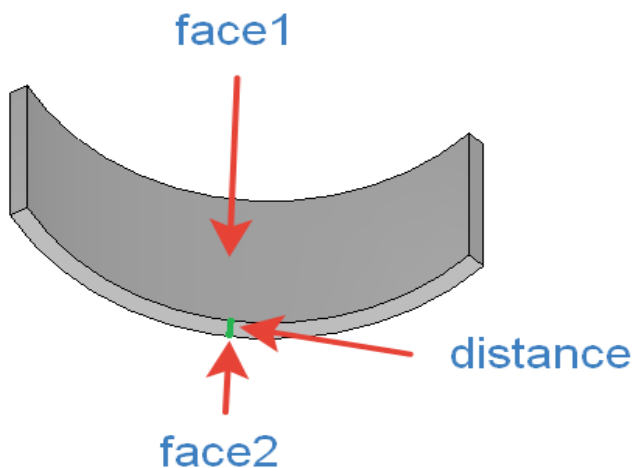


Рис. М.7.24.1

М.7.25. Поиск подобных сгибов

Метод

void

GetSimilarCylindricBends (const [MbFaceShell](#) & shell,
PArray<MbSheetMetalBend> & bends)

выполняет поиск подобных сгибов.

Входными параметрами метода является множество граней листового тела **shell** и массив сгибов **bands**, для которых надо найти подобные сгибы.

Выходным параметром метода является множество найденных подобных сгибов **bends** вместе с исходными сгибами.

Метод не возвращает значений.

Метод объявлен в файле `action_sheet.h`.

В множестве граней листового тела ищутся согнутые цилиндрические, конические или линейчатые сгибы, которые надо добавить к сгибам из **bends**, чтобы они могли разогнуться, то есть сгибы разгибаемые только совместно.

На рис. М.7.25.1 изображена пара подобных сгибов, которые могут сгибаться или разгибаться только вместе.

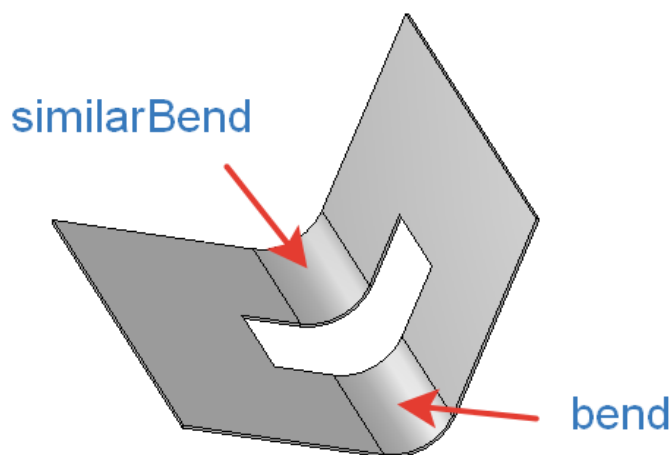


Рис. М.7.25.1

М.7.26. Поиск касательной точки сгиба листового тела

Метод

bool

CalculateTangentPoint (const [MbFace](#) & **face**,
const [MbPlane](#) & **plane**,
[MbCartPoint](#) & **point**)

выполняет расчёт касательной точки для сгиба/разгиба листового тела.

Входными параметрами метода являются:

- **face** – грань листового тела, содержащая точку касания,
- **plane** – касательная плоскость,
- **point** – точка касания.

Выходным параметром метода является точка касания **point**.

При удачной работе метод возвращает true, в противном случае метод возвращает false.

Метод объявлен в файле `action_sheet.h`.

Точка **point** может находиться внутри грани ($0.0 \leq x \leq 1.0$ и $0.0 \leq y \leq 1.0$) или за её пределами. В первом случае точка касания пересчитывается в координаты лежащей под гранью поверхности, во втором – находится одна из точек касания поверхности, лежащей под гранью **face**, и плоскости **plane**.

М.7.27. Поиск осевой линии сгиба

Метод

bool

CalculateConicAxisLine (const [MbFace](#) & **face**,
[MbLineSegment](#) & **axis**)

выполняет расчёт осевой линии разогнутого конического сгиба.

Входным параметром метода являются грань **face** разогнутого конического сгиба листового тела.

Выходным параметром метода является осевая линия **axis**.

При удачной работе метод возвращает true, в противном случае метод возвращает false.

Метод возвращает осевую линию в координатах параметрической области плоской грани **face**.

Метод объявлен в файле `action_sheet.h`.

M.8. ВСПОМОГАТЕЛЬНЫЕ МЕТОДЫ

Для построения тел требуются параметры, которые в определенных случаях нужно вычислить, или требуются вспомогательные элементы. В данной главе перечислены методы, выполняющие вспомогательные вычисления и построения.

M.8.1. Вычисление глубины тела выдавливания или угла тела вращения

Метод

bool

GetSweptValue (const MbSweptData & **sweptData**,
const MbAxis3D & **axis**,
const MbVector3D & **direction**,
const bool rotation,
const bool operationDirection,
const MbCartPoint3D & **point**,
double & *value*)

вычисляет глубину выдавливания или угол вращения для последующего построения тела путем выдавливания или вращения образующей кривой.

Входными параметрами метода являются:

- **sweptData** – данные об образующей кривой,
- **axis** – ось вращения (для расчёта вращения),
- **direction** – направление выдавливания (для расчёта выдавливания),
- rotation – флаг выполнения расчёта: для вращения (true), для выдавливание (false),
- operationDirection – направление выполнения последующей операции: вперед (true), назад (false),
- **point** – точка, до которой требуется вращать или выдавливать образующую кривую.

Выходным параметром метода является *value* – глубина выдавливания или угол вращения.

При удачном выполнении расчёта метод возвращает true, в противном случае метод возвращает false.

Данный метод является вспомогательным для функций **ExtrusionSolid**, **RevolutionSolid**, **ExtrusionResult** и **RevolutionResult**.

M.8.2. Определение образа кривой для выдавливания или вращения

Метод

GetSweptImagePosition (const MbCurve3D & **curve**,
const MbSurface & **surface**,
const MbVector3D & **direction**,
const MbAxis3D & **axis**,
const bool rotation,
MbCartPoint & **imagePosition**,
MbResultType & resType)

вычисляет положение образа точки образующей кривой на поверхности для последующего построения тела путем выдавливания или вращения образующей кривой до заданной поверхности.

Входными параметрами метода являются:

- **curve** – образующая кривая,
- **surface** – поверхность, до которой будет выполняться выдавливание или вращение кривой
- **direction** – направление выдавливания (для расчёта выдавливания),
- **axis** – ось вращения (для расчёта вращения),
- rotation – флаг выполнения расчёта: для вращения (true), для выдавливание (false).

Выходными параметрами метода являются:

- **imagePosition** – точка на поверхности, в которой лежит образ образующей кривой.

- `resType` – код результата операции: при удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Данный метод является вспомогательным для функций [ExtrusionSolid](#), [RevolutionSolid](#), [ExtrusionResult](#) и [RevolutionResult](#).

М.8.3. Определение параметров для выдавливания или вращения

Метод

```
GetRangeToSurface ( const MbSurface & surface,
                    const MbCurve3D & curve,
                    const MbVector3D & direction,
                    const MbAxis3D & axis,
                    const bool rotation,
                    const bool operationDirection,
                    const MbCartPoint & imagePosition,
                    double range[2],
                    MbRect & rectOnSurface,
                    MbResultType & resType )
```

вычисляет глубины выдавливания в прямом и обратном направлениях или углы вращения в прямом и обратном направлениях для последующего построения тела путем выдавливания или вращения образующей кривой до заданной поверхности, а также и габарит образа кривой.

Входными параметрами метода являются:

- **surface** – поверхность, до которой будет выполняться выдавливание или вращение кривой
- **curve** – образующая кривая,
- **direction** – направление выдавливания (для расчёта выдавливания),
- **axis** – ось вращения (для расчёта вращения),
- `rotation` – флаг выполнения расчёта: для вращения (`true`), для выдавливание (`false`),
- `operationDirection` – направление выполнения последующей операции: вперед (`true`), назад (`false`),
- ***imagePosition*** – точка на поверхности, в которой лежит образ образующей кривой.

Выходными параметрами метода являются:

- `range` – расстояния до поверхности: `range[0]` – в обратном направлении, `range[1]` – в прямом направлении,
- `rectOnSurface` – габарит образа кривой на поверхности,
- `resType` – код результата операции: при удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Данный метод является вспомогательным для функций [ExtrusionSolid](#), [RevolutionSolid](#), [ExtrusionResult](#) и [RevolutionResult](#).

М.8.4. Определение ориентации образующей кривой

Метод

```
AreaSign ( const MbCurve3D & curve,
            const MbAxis3D & axis,
            const MbVector3D & direction,
            bool rotation )
```

вычисляет площадь проекции кривой на виртуальную координатную плоскость для определения ориентации образующей кривой при последующем построении тела путем выдавливания или вращения образующей кривой. Незамкнутая образующая кривая замыкается отрезком.

Входными параметрами метода являются:

- **curve** – образующая кривая,
- **axis** – ось вращения (для расчёта вращения),
- **direction** – направление выдавливания (для расчёта выдавливания),
- `rotation` – флаг выполнения расчёта: для вращения (`true`), для выдавливание (`false`).

Выходным параметром метода является площадь проекции кривой.

Данный метод является вспомогательным для функций [ExtrusionSolid](#), [RevolutionSolid](#), [ExtrusionResult](#) и [RevolutionResult](#).

M.8.5. Определение ориентации секущей поверхности

Метод

[AnalyzeSurfaceRelationToSweptOperation](#) (const [MbSurface](#) & **surface**,
const [MbCartPoint](#) & **imagePosition**,
const [MbCurve3D](#) & **curve**,
const [MbVector3D](#) & **direction**,
const [MbAxis3D](#) & **axis**,
const bool rotation,
bool operationDirection,
bool & relativeSense,
[MbResultType](#) & resType)

определяет ориентацию секущей поверхности относительно тела, которое будет строиться путём выдавливания или вращения образующей кривой до заданной поверхности.

Входными параметрами метода являются:

- **surface** – поверхность, до которой будет выполняться выдавливание или вращение кривой
- **imagePosition** – точка на поверхности, в которой лежит образ образующей кривой,
- **curve** – образующая кривая,
- **direction** – направление выдавливания (для расчёта выдавливания),
- **axis** – ось вращения (для расчёта вращения),
- rotation – флаг выполнения расчёта: для вращения (true), для выдавливание (false),
- operationDirection – направление выполнения последующей операции: вперед (true), назад (false).

Выходными параметрами метода являются:

- relativeSense – ориентация поверхности для операции выдавливания или вращения,
- resType – код результата операции: при удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Данный метод является вспомогательным для функций [ExtrusionSolid](#), [RevolutionSolid](#), [ExtrusionResult](#) и [RevolutionResult](#).

M.8.6. Ориентация кривых тела заметания

Метод

[MbResultType](#)

[EvolutionNormalize](#) (const [MbPlacement3D](#) & **place**,
const [MbContour](#) & **contour**,
const [MbCurve3D](#) & **guide**,
[EvolutionValues](#) & parameters,
[MbAxis3D](#) & **axis**,
double & **angle**,
VERSION version)

выполняет ориентацию образующего контура и направляющей кривой для построения тела заметания.

Входными параметрами метода являются:

- **place** – локальная система координат образующего контура,
- **contour** – образующий контур,
- **guide** – направляющая кривая,
- parameters – параметры операции заметания,
- version – версия операции.

Выходными параметрами метода являются:

- **axis** – ось доворота образующей,
- **angle** – угол доворота образующей,

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Данный метод является вспомогательным для функций **EvolutionSolid** и **EvolutionResult**.

M.8.7. Построение копии направляющей кривой тела заметания

Метод

`MbCurve3D * TrimClosedSpine (MbCurve3D & curve, double t)`

выполняет построение копии замкнутой кривой с началом в точке, определяемой параметром *t*.

Входными параметрами метода являются:

- **curve** – направляющая кривая,
- *t* – параметр кривой.

При удачной работе метод возвращает построенную копию кривой с началом в заданной точке, в противном случае метод возвращает ноль.

Данный метод является вспомогательной для функций **EvolutionSolid** и **EvolutionResult**.

M.8.8. Построение ребра жёсткости

Метод

`MbResultType`

`RibElement (const MbSolid & solid,
const MbPlacement3D & place,
MbContour & contour,
size_t index,
RibValues & params,
const MbSNameMaker & names,
MbSolid *& result)`

выполняет построение ребра жёсткости для исходного тела.

Входными параметрами метода являются:

- **solid** – исходное тело,
- **sameShell** – вариант копирования исходного тела,
- **place** – локальная система координат, плоскость XY которой является плоскостью симметрии,
- **contour** – формообразующий контур на плоскости XY локальной системы координат,
- **index** – номер сегмента в контуре,
- **params** – параметры ребра жёсткости,
- **names** – именователъ граней ребра жесткости.

Выходным параметром метода является построенное тело **result**.

При удачной работе метод возвращает `rt_Success`, в противном случае метод возвращает код ошибки из перечисления `MbResultType`.

Метод выполняет построение отдельного ребра жёсткости без объединения с исходным телом.

M.8.9. Проверка кривой для построения линейчатого тела

Метод

`void CheckRuledCurve (const MbCurve3D & curve1,
const MbCurve3D & curve2,
bool & isInverted,
bool & isShifted,
VERSION version)`

проверяет вторую кривую на согласованность с первой кривой для построения линейчатого незамкнутого тела и выполняет необходимую модификацию второй кривой.

Входными параметрами метода являются:

- **curve1** – первая кривая,
- **curve2** – вторая кривая,
- **version** – версия операции.

Выходными параметрами метода являются:

- флаг **isInverted**, сигнализирующий о изменении направления второй кривой,
- флаг **isShifted**, сигнализирующий о смещении начала второй кривой.

Возвращаемое значение у метода отсутствует.

Данный метод является вспомогательным для метода построения линейчатого незамкнутого тела по двум кривым **RuledShell**.

M.8.10. Проверка параметров кривой для построения линейчатого тела

Метод

CheckRuledParams (const [MbCurve3D](#) & **curve**,
SArray<double> & **params**,
bool **isAscending**)

выполняет проверку параметров кривой и нормализацию параметров замкнутой кривой.

Входными параметрами метода являются:

- **curve** – кривая,
- **params** – множество параметров кривой.
- **isAscending** – флаг возрастания параметров множества.

Выходным параметром метода является множество параметров кривой **params**.

При удачной работе метод возвращает **true**, в противном случае метод возвращает **false**.

Данный метод является вспомогательным для метода построения линейчатого незамкнутого тела **RuledShell**.

M.8.11. Проверка кривой для построения тела соединения

Метод

CheckJoinedCurve (const [MbCurve3D](#) & **curve1**,
const [MbCurve3D](#) & **curve2**,
bool & **isInverted1**,
bool & **isShifted1**,
VERSION **version**)

проверяет вторую кривую на согласованность с первой кривой для построения незамкнутого тела соединения и выполняет необходимую модификацию второй кривой.

Входными параметрами метода являются:

- **curve1** – первая кривая,
- **curve2** – вторая кривая,
- **version** – версия операции.

Выходными параметрами метода являются:

- флаг **isInverted**, сигнализирующий о изменении направления второй кривой,
- флаг **isShifted**, сигнализирующий о смещении начала второй кривой.

Возвращаемое значение у метода отсутствует.

Данный метод является вспомогательным для метода построения незамкнутого тела соединения по двум кривым **JoinShell**.

M.8.12. Проверка параметров кривой для построения тела соединения

Метод

CheckJoinedParams (const [MbCurve3D](#) & **curve**,
SArray<double> & **params**,

bool isAscending)

выполняет проверку параметров кривой и нормализацию параметров замкнутой кривой.

Входными параметрами метода являются:

- **curve** – кривая,
- **params** – множество параметров кривой,
- **isAscending** – флаг возрастания параметров множества.

Выходным параметром метода является множество параметров кривой **params**.

При удачной работе метод возвращает true, в противном случае метод возвращает false.

Данный метод является вспомогательным для метода построения незамкнутого тела **JoinShell**.

М.8.13. Построение кривой по множеству рёбер

Метод

CreateJoinedCurve (const RArray<MbCurveEdge> & **edges**,
const SArray<bool> & **orientations**,
const MbMatrix3D & **matrix**,
MbResultType & **result**)

выполняет построение кривой по множеству рёбер.

Входными параметрами метода являются:

- **edges** – множество ребер,
- **orientations** – ориентация ребер,
- **matrix** – матрица преобразования ребер,

Выходным параметром метода является значение **result** из перечисления MbResultType.

При удачной работе метод возвращает указатель на построенную кривую, в противном случае метод возвращает ноль.

Данный метод является вспомогательным для метода построения незамкнутого тела **JoinShell**.

О.1. ЭЛЕМЕНТАРНЫЕ ОБЪЕКТЫ

К элементарным объектам относятся объекты геометрического ядра C3D, которые описывают такие математические сущности, как: вектор, точка, ось, локальная система координат, матрица преобразования, габаритный параллелепипед, габаритный прямоугольник. Элементарные объекты имеют простые структуры данных. Элементарные объекты являются инструментом и строительным материалом для более сложных геометрических объектов и используются всеми модулями геометрического ядра.

О.1.1. Вектор в трёхмерном пространстве MbVector3D

Класс MbVector3D объявлен в файле mb_vector3d.h.

Вектор MbVector3D описывает перемещение или направление в трёхмерном пространстве и определяется тремя компонентами x , y , z в декартовой системе координат.

Для обозначения векторов в трёхмерном пространстве будем использовать одну или несколько строчных букв латинского алфавита, выделенных полужирным шрифтом, а для объединения компонент вектора будем заключать их в квадратные скобки, например,

$$\mathbf{vector} = [x \ y \ z].$$

Вектор MbVector3D не привязан к точкам пространства и поэтому не имеет метода, перемещающего его в пространстве.

О.1.2. Радиус-вектор точки в трёхмерном пространстве MbCartPoint3D

Класс MbCartPoint3D объявлен в файле mb_cart_point3d.h.

Радиус-вектор MbCartPoint3D (картезианская точка) описывает положение точки в трёхмерном пространстве и определяется тремя координатами x , y , z в декартовой системе координат. Радиус-вектор описывает преобразование, переводящее начальную точку декартовой системы координат в точку пространства с заданными координатами в этой декартовой системе координат.

Для обозначения точек в трёхмерном пространстве будем использовать одну или несколько строчных букв латинского алфавита, выделенных полужирным шрифтом, а для объединения координат точки будем заключать их в квадратные скобки, например,

$$\mathbf{point} = [x \ y \ z].$$

Радиус-вектор в отличие от вектора связан с началом координат. Координаты радиуса-вектора MbCartPoint3D и компоненты вектора [MbVector3D](#) по-разному изменяются при переходе от одной системы координат к другой системе координат и при изменениях положения в пространстве, реализованных в методах:

MbCartPoint3D & **Transform**(const [MbMatrix3D](#) &),

MbCartPoint3D & **Rotate**(const MbAxis3D &, double angle),

MbCartPoint3D & **Move**(const [MbVector3D](#) &).

Перечисленные методы возвращают ссылку на себя после преобразования.

О.1.3. Расширенный вектор в трёхмерном пространстве MbHomogenius3D

Класс MbHomogenius3D объявлен в файле mb_homogenius3d.h.

Расширенный радиус-вектор MbHomogenius3D описывает положение точки в трёхмерном пространстве и определяется четырьмя координатами x , y , z , w . Четвёртая координата называется весом. Расширенный радиус-вектор используется при вычислении радиуса-вектора B -кривых и B -поверхностей, построенных на основе B -сплайнов. Координаты x_w , y_w , z_w , w расширенного радиуса-вектора MbHomogenius3D связаны с координатами x , y , z радиуса-вектора соотношениями

$$x = \frac{x_w}{w}, y = \frac{y_w}{w}, z = \frac{z_w}{w}.$$

В операциях умножения на расширенную матрицу [MbMatrix3D](#) можно считать, что векторы и точки также имеют четвёртую координату, для вектора [MbVector3D](#) она равна нулю, а для радиуса-вектора [MbCartPoint3D](#) она равна единице.

О.1.4. Локальная система координат MbPlacement3D

Класс MbPlacement3D объявлен в файле mb_placement3d.h.

Локальная система координат в трёхмерном пространстве MbPlacement3D описывается начальной точкой **origin** и тремя некопланарными векторами **axisX**, **axisY**, **axisZ**, рис. О.1.4.1.

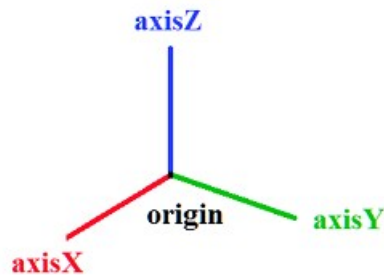


Рис. О.1.4.1.

В большинстве случаев система координат является правой, а векторы системы ортонормированны. С помощью преобразований система координат может стать левой и не ортонормированной. Состояние локальной системы координат можно запросить с помощью методов:

bool **IsLeft**() – является ли системы координат левой,

bool **IsRight**() – является ли системы координат правой.

bool **IsTranslation**() – присутствует ли сдвиг начала **origin** системы координат,

bool **IsRotation**() – присутствует ли поворот системы координат,

bool **IsOrt**() – является ли системы координат ортогональной, но ненормированной,

bool **IsSingle**() – совпадает ли система координат с той, в которой она задана,

bool **IsNormal**() – является ли системы координат ортонормированной,

bool **IsOrtogonal**() – является ли системы координат ортогональной,

bool **IsCircular**() – является ли системы координат ортогональной с равными по длине осями **axisX**, **axisY**, в которой окружность остается окружностью,

bool **IsIsotropic**() – является ли системы координат ортогональной с равными по длине осями **axisX**, **axisY**, **axisZ**, в которой объекты не искажаются, а только масштабируются,

bool **IsAffine**() – является ли системы координат аффинной (если нет - то она ортонормированная).

Локальная система координат является декартовой. Точка в декартовой системе координат определяется тремя координатами x , y , z . Локальная система может выступать в роли цилиндрической или сферической системы координат.

При использовании локальной системы координат MbPlacement3D в роли цилиндрической системы ось Z цилиндрической системы координат совпадает с осью Z декартовой системы, полярная ось цилиндрической системы совпадает с осью X декартовой системы, полярный угол цилиндрической системы отсчитывается от оси X в сторону оси Y . Координата x играет роль проекции радиуса-вектора на плоскость XY , координата y играет роль полярного угла.

При использовании локальной системы координат MbPlacement3D в роли сферической системы плоскость сферической системы координат совпадает с плоскостью XY декартовой системы, долгота сферической системы отсчитывается от оси X в сторону оси Y . Координата x играет роль длины радиуса-вектора, координата y играет роль долготы.

О.1.5. Расширенная матрица в трёхмерном пространстве MbMatrix3D

Класс MbMatrix3D объявлен в файле mb_matrix3d.h.

В трёхмерном пространстве матрица MbMatrix3D описывает преобразование из одной системы координат в другую. Матрица имеет размерность четыре на четыре. Пусть в выбранной системе координат построена локальная аффинная система координат с началом в точке \mathbf{r} с координатами r_1, r_2, r_3 и базисными векторами $\mathbf{a}=[a_1 \ a_2 \ a_3]$, $\mathbf{b}=[b_1 \ b_2 \ b_3]$, $\mathbf{c}=[c_1 \ c_2 \ c_3]$. Векторы $\mathbf{a}, \mathbf{b}, \mathbf{c}$ должны быть линейно независимыми, но могут быть не ортогональными друг другу и иметь произвольную длину. Матрица MbMatrix3D преобразования координат из локальной системы в выбранную систему координат имеет вид

$$\mathbf{M} = \begin{bmatrix} a_1 & a_2 & a_3 & 0 \\ b_1 & b_2 & b_3 & 0 \\ c_1 & c_2 & c_3 & 0 \\ r_1 & r_2 & r_3 & 1 \end{bmatrix}.$$

Для обозначения расширенных матриц в трёхмерном пространстве будем использовать прописные буквы латинского алфавита, выделенные полужирным шрифтом, например \mathbf{M} . Заметим, что каждому базисному вектору $\mathbf{a}, \mathbf{b}, \mathbf{c}$ и начальной точке \mathbf{r} локальной системы координат соответствует своя строка в матрице преобразования из локальной системы в выбранную систему координат.

Матрица MbMatrix3D является расширенной и работает с однородными радиусами-векторами и однородными векторами [MbHomogenius3D](#) в трёхмерном пространстве.

При преобразовании радиуса-вектора [MbCartPoint3D](#) по матрице MbMatrix3D точке следует добавить четвёртую координату, равную единице. Пусть точка с координатами x_1, x_2, x_3 в локальной системе координат имеет координаты p_1, p_2, p_3 в выбранной системе координат, тогда при участии расширенной матрицы MbMatrix3D координаты будут связаны соотношением

$$\begin{bmatrix} p_1 & p_2 & p_3 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & 0 \\ b_1 & b_2 & b_3 & 0 \\ c_1 & c_2 & c_3 & 0 \\ r_1 & r_2 & r_3 & 1 \end{bmatrix}.$$

Заметим, что трёхмерный радиус-вектор умножается на расширенную матрицу MbMatrix3D справа.

При преобразовании вектора [MbVector3D](#) по матрице MbMatrix3D вектору следует добавить четвёртую компоненту, равную нулю. Пусть вектор с компонентами y_1, y_2, y_3 в локальной системе координат имеет компоненты r_1, r_2, r_3 в выбранной системе координат, тогда при участии расширенной матрицы MbMatrix3D компоненты будут связаны соотношением

$$\begin{bmatrix} r_1 & r_2 & r_3 & 0 \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & y_3 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & 0 \\ b_1 & b_2 & b_3 & 0 \\ c_1 & c_2 & c_3 & 0 \\ r_1 & r_2 & r_3 & 1 \end{bmatrix}.$$

Заметим, что трёхмерный вектор умножается на расширенную матрицу MbMatrix3D справа.

О.1.6. Габаритный параллелепипед в трёхмерном пространстве MbCube

Класс MbCube объявлен в файле mb_cube.h.

Габаритный параллелепипед MbCube описывает габариты протяжённого объекта (кривой, поверхности, тела, нескольких тел) в трёхмерном пространстве и определяется двумя точками **pmin** и **pmax**. Грани габаритного параллелепипеда параллельны плоскостям системы координат, в которой описан куб. Точки **pmin** и **pmax** описывают две противоположные вершины габаритного куба, имеющие наименьшие и наибольшие координаты, соответственно, рис. О.1.6.1.

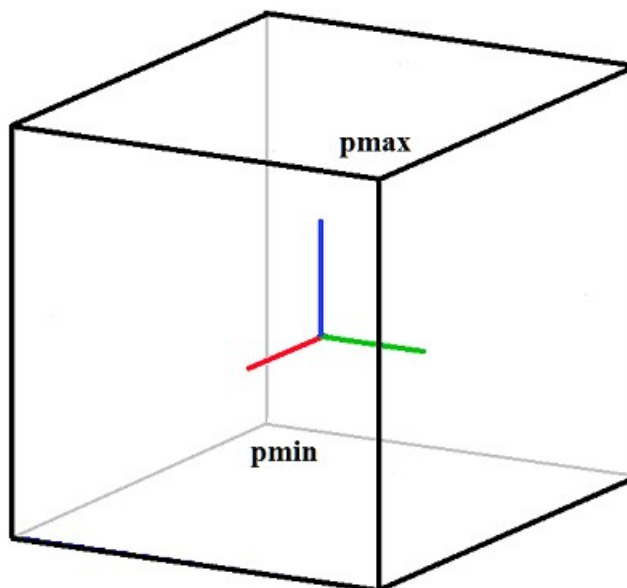


Рис. О.1.6.1.

Если габариты протяжённого объекта не определены, то габаритный параллелепипед считается пустым и **pmin**=[10^{-300} 10^{-300} 10^{-300}], **pmax**=[-10^{-300} -10^{-300} -10^{-300}]. Для пустого габаритного куба **pmin**>**pmax** и метод IsEmpty() возвращает true.

О.1.7. Одномерный габарит MbRect1D

Класс MbRect1D объявлен в файле mb_rect1d.h.

Одномерный габарит MbRect1D описывает одномерную область, например, область определения параметра кривой и определяется двумя числами: *zmin* и *zmax*, рис. О.1.7.1.



Рис. О.1.7.1.

Числа *zmin* и *zmax* описывают начальный и конечный край области. Если одномерная область не определена, то одномерный габарит считается пустым и *zmin*>*zmax*. Для пустого одномерного габарита метод IsEmpty() возвращает true.

О.1.8. Вектор в двумерном пространстве MbVector

Класс MbVector объявлен в файле mb_vector.h.

Вектор MbVector описывает перемещение или направление в двумерном пространстве и определяется двумя компонентами *x* и *y* в декартовой системе координат.

Для обозначения векторов в двумерном пространстве будем использовать одну или несколько строчных букв латинского алфавита, выделенных полужирным наклонным шрифтом, а для объединения компонент вектора будем заключать их в квадратные скобки, например,

$$\mathbf{vector} = [x \ y].$$

Вектор `MbVector` не привязан к точкам пространства и поэтому не имеет метода, перемещающего его в пространстве.

О.1.9. Нормализованный вектор в двумерном пространстве `MbDirection`

Класс `MbDirection` объявлен в файле `mb_vector.h`.

Нормализованный вектор `MbDirection` описывает направление или угол поворота в двумерном пространстве и определяется двумя компонентами ax и ay в декартовой системе координат. Длина нормализованного вектора равна единице, а его компоненты равны соответственно синусу и косинусу угла между осью OX и нормализованным вектором. Таким образом, $ax = \cos(\alpha)$, $ay = \sin(\alpha)$, где α – угол между нормализованным вектором и осью абсцисс системы координат.

О.1.10. Радиус-вектор точки в двумерном пространстве `MbCartPoint`

Класс `MbCartPoint` объявлен в файле `mb_cart_point.h`.

Радиус-вектор `MbCartPoint` (картезианская точка) описывает положение точки в двумерном пространстве и определяется двумя координатами x и y в декартовой системе координат. Радиус-вектор описывает преобразование, переводящее начальную точку декартовой системы координат в точку пространства с заданными координатами в этой декартовой системе координат.

Для обозначения точек в двумерном пространстве будем использовать одну или несколько строчных букв латинского алфавита, выделенных полужирным наклонным шрифтом, а для объединения координат точки будем заключать их в квадратные скобки, например,

$$\mathbf{point} = [x \ y].$$

Радиус-вектор в отличие от вектора связан с началом координат. Координаты радиуса-вектора `MbCartPoint` и компоненты вектора [MbVector](#) по-разному изменяются при переходе от одной системы координат к другой системе координат и при изменениях положения в пространстве, реализованных в методах

```
void Transform( const MbMatrix & ),  
void Rotate( const MbCartPoint &, double angle ),  
void Move( const MbVector & ).
```

О.1.11. Расширенный вектор в двумерном пространстве `MbHomogenius`

Класс `MbHomogenius` объявлен в файле `mb_homogenius.h`.

Расширенный радиус-вектор `MbHomogenius` описывает положение точки в двумерном пространстве и определяется тремя координатами x , y , w . Третья координата называется весом. Расширенный радиус-вектор используется при вычислении радиуса-вектора двумерных B -кривых, построенных на основе B -сплайнов. Координаты x_w , y_w , w расширенного радиуса-вектора `MbHomogenius` связаны с координатами x , y радиуса-вектора [MbCartPoint](#) соотношениями

$$x = \frac{x_w}{w}, \quad y = \frac{y_w}{w}.$$

В операциях умножения на расширенную матрицу [MbMatrix](#) можно считать, что двумерные векторы и точки также имеют третью координату, для вектора [MbVector](#) она равна нулю, а для радиуса-вектора [MbCartPoint](#) она равна единице.

О.1.12. Локальная система координат MbPlacement

Класс MbPlacement объявлен в файле mb_placement.h.

Локальная система координат в двумерном пространстве MbPlacement описывается начальной точкой *origin* и двумя непараллельными векторами *axisX* и *axisY*, рис. О.1.12.1.

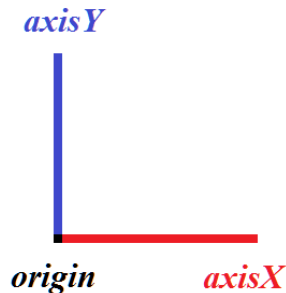


Рис. О.1.12.1.

В большинстве случаев система координат является правой, а векторы системы ортонормированны. С помощью преобразований система координат может стать левой и не ортонормированной. Состояние локальной системы координат можно запросить с помощью методов:

bool **IsLeft**() – является ли системы координат левой,

bool **IsSingle**() – совпадает ли система координат с той, в которой она задана,

bool **IsNormal**() – является ли системы координат ортонормированной,

bool **IsCircular**() – является ли системы координат ортогональной с равными по длине осями *axisX*, *axisY*, в которой окружность остается окружностью,

bool **IsIsotropic**() – является ли системы координат ортогональной с равными по длине осями *axisX*, *axisY*, в которой объекты не искажаются, а только масштабируются,

bool **IsAffine**() – является ли системы координат аффинной (если нет - то она ортонормированная).

Локальная система координат является декартовой.

О.1.13. Расширенная матрица в двумерном пространстве MbMatrix

Класс [MbMatrix3D](#) объявлен в файле mb_matrix3d.h.

В двумерном пространстве матрица MbMatrix описывает преобразование из одной системы координат в другую. Матрица имеет размерность три на три. Пусть в выбранной системе координат построена локальная аффинная система координат с началом в точке \mathbf{r} с координатами r_1, r_2 и базисными векторами $\mathbf{a}=[a_1 \ a_2]$, $\mathbf{b}=[b_1 \ b_2]$. Векторы \mathbf{a} и \mathbf{b} не должны быть коллинеарными, но могут быть не ортогональными друг другу и иметь произвольную длину. Матрица MbMatrix преобразования координат из локальной системы в выбранную систему координат имеет вид

$$\mathbf{M} = \begin{bmatrix} a_1 & a_2 & 0 \\ b_1 & b_2 & 0 \\ r_1 & r_2 & 1 \end{bmatrix}.$$

Для обозначения расширенных матриц в двумерном пространстве будем использовать прописные буквы латинского алфавита, выделенные полужирным наклонным шрифтом, например \mathbf{M} . Заметим, что каждому базисному вектору \mathbf{a} , \mathbf{b} и начальной точке \mathbf{r} локальной системы координат соответствует своя строка в матрице преобразования из локальной системы в выбранную систему координат.

Матрица MbMatrix является расширенной и работает с однородными радиусами-векторами и однородными векторами [MbHomogenius](#) в двумерном пространстве.

При преобразовании радиуса-вектора [MbCartPoint](#) по матрице MbMatrix точке следует добавить третью координату, равную единице. Пусть точка с координатами x_1, x_2 в локальной системе координат имеет координаты p_1, p_2 в выбранной системе координат, тогда при участии расширенной матрицы MbMatrix координаты будут связаны соотношением

$$[p_1 \quad p_2 \quad 1] = [x_1 \quad x_2 \quad 1] \cdot \begin{bmatrix} a_1 & a_2 & 0 \\ b_1 & b_2 & 0 \\ r_1 & r_2 & 1 \end{bmatrix}.$$

Заметим, что двумерный радиус-вектор умножается на расширенную матрицу MbMatrix справа.

При преобразовании вектора [MbVector](#) по матрице MbMatrix вектору следует добавить третью компоненту, равную нулю. Пусть вектор с компонентами y_1, y_2 в локальной системе координат имеет компоненты r_1, r_2 в выбранной системе координат, тогда при участии расширенной матрицы MbMatrix компоненты будут связаны соотношением

$$[r_1 \quad r_2 \quad 0] = [y_1 \quad y_2 \quad 0] \cdot \begin{bmatrix} a_1 & a_2 & 0 \\ b_1 & b_2 & 0 \\ r_1 & r_2 & 1 \end{bmatrix}.$$

Заметим, что двумерный вектор умножается на расширенную матрицу MbMatrix справа.

О.1.14. Габаритный прямоугольник в двумерном пространстве MbRect

Класс MbRect объявлен в файле mb_rect.h.

Габаритный прямоугольник MbRect описывает габариты протяжённого объекта (кривой, нескольких кривых) в двумерном пространстве и определяется четырьмя числами: *left*, *right*, *bottom*, *top*. Стороны габаритного прямоугольника параллельны осям системы координат, в которой описан прямоугольник. Числа *left* и *right* описывают минимальную и максимальную абсциссу габаритного прямоугольника, а числа *bottom* и *top* описывают минимальную и максимальную ординату габаритного прямоугольника, рис. О.1.14.1.

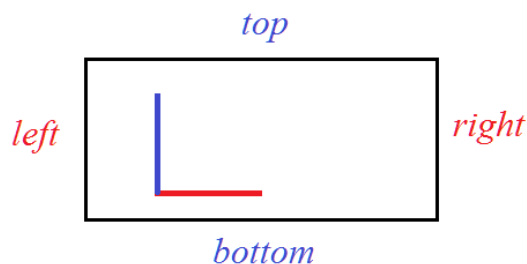


Рис. О.1.14.1.

Если габариты протяжённого объекта не определены, то габаритный прямоугольник считается пустым и $left > right$, и $bottom > top$. Для пустого габаритного прямоугольника метод **IsEmpty()** возвращает true.

0.2. ГЕОМЕТРИЧЕСКИЕ ОБЪЕКТЫ

Геометрические объекты описывают форму моделируемых объектов. К геометрическим объектам относятся кривые, поверхности, тела, а также топологические объекты, описывающие геометрические свойства, не зависящие от количественных характеристик и характеризующие непрерывную связь точек трёхмерного пространства друг с другом. Геометрические объекты могут быть трёхмерные и двумерные. Двумерные объекты используются для работы в области определения параметров поверхностей и для работы в плоскостях трёхмерных локальных систем координат. В данной части описаны родительские классы геометрических объектов.

0.2.1. Счётчик ссылок MbRefItem

Класс MbRefItem объявлен в файле reference_item.h.

Класс MbRefItem описывается количеством своих владельцев *useCount* и представляет собой счётчик объектов, владеющих данным объектом.

Все геометрические объекты ядра C3D делятся на три группы: двумерные геометрические объекты, трёхмерные геометрические объекты и топологические объекты. Все геометрические объекты являются наследниками классов MbRefItem и TapeBase, рис. 0.2.1.1.

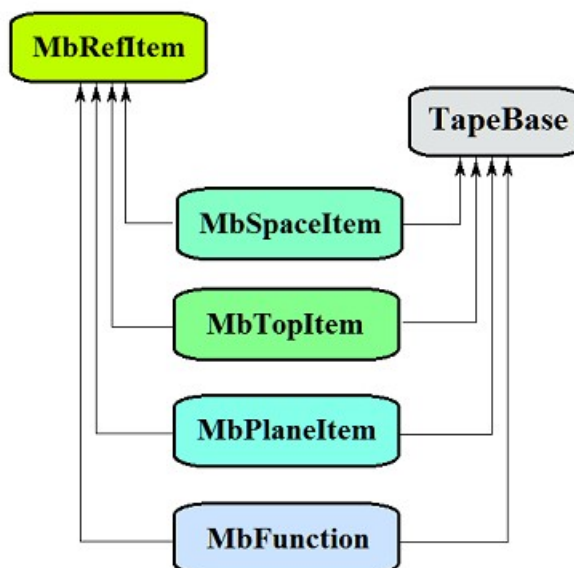


Рис 0.2.1.1.

Класс TapeBase обеспечивает для своих наследников запись в поток и и чтение из потока.

Наследниками классов MbRefItem и TapeBase являются следующие геометрические объекты:

[MbSpaceItem](#) – базовый абстрактный класс трёхмерных геометрических объектов,

[MbTopItem](#) – базовый абстрактный класс топологических объектов,

[MbPlaneItem](#) – базовый абстрактный класс двумерных геометрических объектов,

[MbFunction](#) – базовый абстрактный класс скалярных функций.

Счётчик ссылок обеспечивает корректную работу классов и методов, содержащих указатели на геометрические объекты. Если некоторый класс в своих данных содержит указатель на геометрический объект, то он обязан в конструкторе увеличить счётчик ссылок у геометрического объекта на единицу методом [AddRef\(\)](#), а в деструкторе должен вызвать у геометрического объекта метод [Release\(\)](#), который уменьшает счётчик ссылок геометрического объекта на единицу и, при достижении счётчиком ссылок нуля, удаляет геометрический объект. Метод [DecRef\(\)](#) уменьшает счётчик ссылок геометрического объекта на единицу. Класс MbRefItem обрабатывается регистратором дублирования MbRegDuplicate и регистратором преобразования MbRegTransform.

Метод
MbeRefType [RefType\(\)](#)
возвращает регистрационный тип объекта, использующего счётчик ссылок.

О.2.2. Трёхмерный геометрический объект MbSpaceItem

Класс MbSpaceItem объявлен в файле space_item.h.

Класс MbSpaceItem является наследником классов [MbRefItem](#), TapeBase и родительским классом для трёхмерных геометрических объектов.

К трёхмерным геометрическим объектам ядра C3D относятся: точка, кривые, поверхности, вспомогательные объекты и объекты геометрической модели, рис. О.2.2.1.

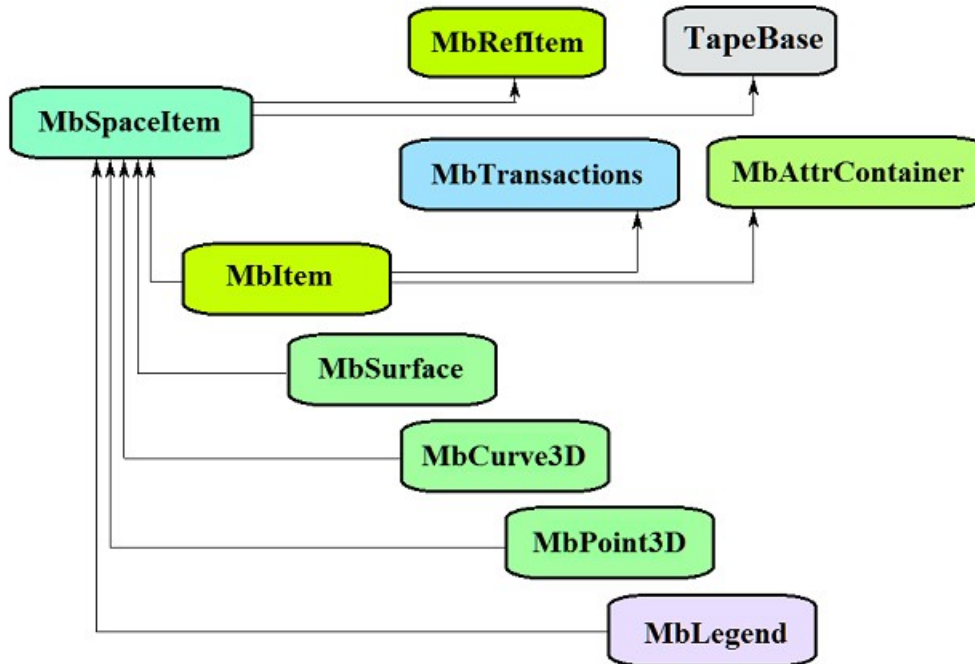


Рис О.2.2.1.

Наследниками класса MbSpaceItem являются следующие семейства трёхмерных геометрических объектов:

MbPoint3D – точка, кривая,

[MbSurface](#) – поверхность,

[MbLegend](#) – вспомогательный геометрический объект,

[MbItem](#) – объект геометрической модели.

Основными методами трёхмерных геометрических объектов являются
void **Move**(const [MbVector3D](#) & v, MbRegTransform * iReg = NULL), [MbMatrix3D](#)
void **Rotate**(const MbAxis3D & axis, double angle, MbRegTransform * iReg = NULL),
void **Transform**(const & m, MbRegTransform * iReg = NULL),

обслуживающие преобразование геометрического объекта. При преобразовании используется регистратор MbRegTransform для предотвращения многократного преобразования вложенных объектов. Если объект содержит указатели или ссылки на другие объекты, то вложенные объекты так же трансформируются. Регистратор необходимо использовать, если надо последовательно преобразовать несколько взаимосвязанных объектов, связь которых обусловлена наличием в них указателей или ссылок на общие объекты. При преобразовании без регистратора можно получить многократное преобразование общих вложенных объектов.

Кроме того, все геометрические объекты имеют методы, обеспечивающие копирование, проверку на совпадение, проверку на возможность сделать совпадающими и делающие объекты совпадающими:

MbSpaceItem & **Duplicate**(MbRegDuplicate * iReg = NULL),
bool **IsSame**(const MbSpaceItem & **item**),
bool **IsSimilar**(const MbSpaceItem & **item**),
bool **SetEqual**(const MbSpaceItem & **item**).

При создании копии используется регистратор MbRegDuplicate для предотвращения многократного копирования вложенных объектов. Если объект содержит указатели или ссылки на другие объекты, то вложенные объекты так же копируются. Регистратор необходимо использовать, если надо последовательно копировать несколько взаимосвязанных объектов, связь которых обусловлена наличием в них указателей или ссылок на общие объекты. При копировании без регистратора можно получить набор копий одного и того же вложенного объекта вместо одной его копии.

Для идентификации типа геометрические объекты снабжены методами

MbSpaceItem **IsA**(),
MbSpaceItem **Type**(),
MbSpaceItem **Family**().

возвращающими тип из перечисления трёхмерных геометрических объектов.

Методы

MbProperty & **CreateProperty**(MbPrompt name),
void **GetProperties**(MbProperties & *properties*),
void **SetProperties**(MbProperties & *properties*)

обеспечивают выдачу и редактирование внутренних данных геометрических объектов. Метод **GetProperties** добавляет к множеству *properties* данные объекта в виде наследников класса MbProperty.

Метод **CalculateWire**(double sag, [MbMesh](#) & **mesh**) строит полигональную копию геометрического объекта, которая используется для визуализации.

О.2.3. Топологический объект MbTopItem

Класс MbTopItem объявлен в файле topology_item.h.

Класс MbTopItem является наследником классов [MbRefItem](#), TareBase и родительским классом для топологических объектов. Среди топологических объектов выделен класс именованных топологических объектов [MbTopologyItem](#), который является наследником классов MbTopItem и MbAttributeContainer. Класс [MbTopologyItem](#) объявлен также в файле topology_item.h.

Геометрическое ядро C3D оперирует топологическими объектами, которые приведены на рис. О.2.3.1.

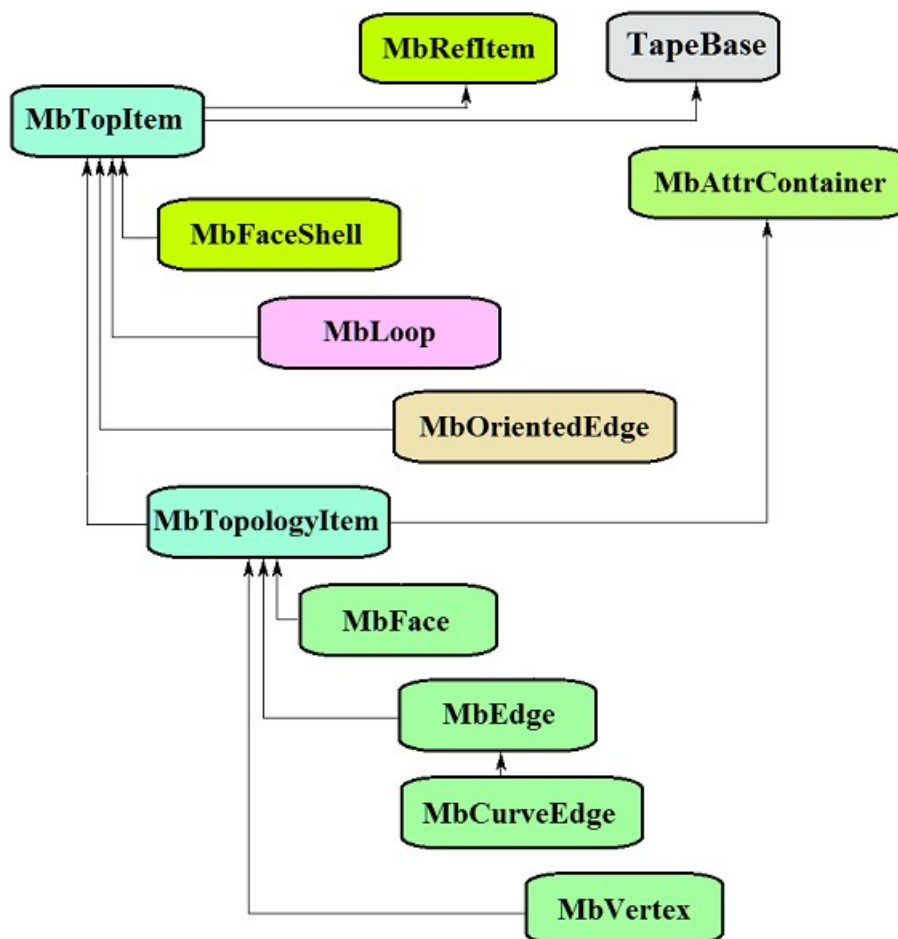


Рис 0.2.3.1.

Наследниками класса MbTopItem являются следующие топологические объекты:

- [MbFaceShell](#) – множество граней,
- [MbLoop](#) – цикл рёбер границы грани,
- [MbOrientedEdge](#) – ориентированное ребро цикла,
- [MbTopologyItem](#) – именованный топологический объект.

Наследниками именованного топологического объекта [MbTopologyItem](#) являются следующие объекты:

- [MbVertex](#) – вершина,
- [MbEdge](#) – ребро,
- [MbFace](#) – грань.

Ребро имеет наследника [MbCurveEdge](#), описывающее гладкий участок стыковки двух граней или край грани.

Работу именованных топологических объектов с атрибутами обеспечивает контейнер атрибутов MbAttrContainer.

Основными методами именованных топологических объектов являются
void **Move**(const [MbVector3D](#) & v, MbRegTransform * iReg = NULL),
void **Rotate**(const MbAxis3D & axis, double angle, MbRegTransform * iReg = NULL),
void **Transform**(const [MbMatrix3D](#) & m, MbRegTransform * iReg = NULL),
обслуживающие преобразование топологического объекта и методы работы с именем и атрибутами. При преобразовании используется регистратор MbRegTransform для предотвращения многократного преобразования вложенных объектов. Если объект содержит указатели или ссылки на другие объекты, то вложенные объекты так же трансформируются. Регистратор необходимо использовать, если надо последовательно преобразовать несколько взаимосвязанных объектов, связь которых обусловлена наличием в них указателей или ссылок на общие объекты. При преобразовании без регистратора можно получить многократное преобразование общих вложенных объектов.

Для идентификации типа топологические объекты снабжены методом **IsA()**, возвращающим тип из перечисления топологических объектов **MbeTopologyType**.

О.2.4. Двумерный геометрический объект **MbPlaneItem**

Класс **MbPlaneItem** объявлен в файле **plane_item.h**.

Класс **MbPlaneItem** является наследником классов **MbRefItem**, **TapeBase** и родительским классом для всех двумерных геометрических объектов.

К двумерным геометрическим объектам ядра C3D относятся: кривые, мультилиния и регион, рис. О.2.4.1.

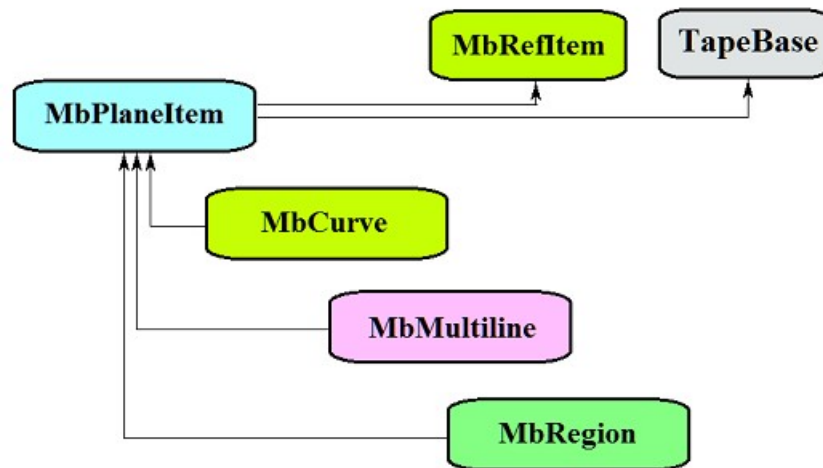


Рис О.2.4.1.

Наследниками класса **MbPlaneItem** являются следующие семейства двумерных геометрических объектов:

MbCurve – двумерная кривая,

MbMultiline – мультилиния,

MbRegion – регион.

Основными методами двумерных геометрических объектов являются

void **Move**(const **MbVector3D** & v, MbRegTransform * iReg = NULL, ...),

void **Rotate**(const **MbCartPoint** & p, const **MbDirection** & angle, MbRegTransform * iReg = NULL, ...),

void **Transform**(const **MbMatrix** & m, MbRegTransform * iReg = NULL, ...),

обслуживающие преобразование двумерного геометрического объекта. При преобразовании используется регистратор **MbRegTransform** для предотвращения многократного преобразования вложенных объектов. Если объект содержит указатели или ссылки на другие объекты, то вложенные объекты так же трансформируются. Регистратор необходимо использовать, если надо последовательно преобразовать несколько взаимосвязанных объектов, связь которых обусловлена наличием в них указателей или ссылок на общие объекты. При преобразовании без регистратора можно получить многократное преобразование общих вложенных объектов.

Кроме того все геометрические объекты имеют методы, обеспечивающие дублирование, проверку на совпадение, проверку на возможность сделать совпадающими и делающие объекты совпадающими:

MbPlaneItem & **Duplicate**(MbRegDuplicate * iReg = NULL),

bool **IsSame**(const **MbPlaneItem** & item),

bool **IsSimilar**(const **MbPlaneItem** & item),

bool **SetEqual**(const **MbPlaneItem** & item).

При создании копии используется регистратор **MbRegDuplicate** для предотвращения многократного копирования вложенных объектов. Если объект содержит указатели или ссылки на другие объекты, то вложенные объекты так же копируются. Регистратор необходимо использовать, если надо последовательно копировать несколько взаимосвязанных объектов, связь которых обусловлена

наличием в них указателей или ссылок на общие объекты. При копировании без регистратора можно получить набор копий одного и того же вложенного объекта вместо одной его копии.

Для идентификации типа геометрические объекты снабжены методами

MbePlaneType **IsA**(),

MbePlaneType **Type**(),

MbePlaneType **Family**(),

возвращающими тип из перечисления двумерных геометрических объектов.

Методы

MbProperty & **CreateProperty**(MbePrompt name),

void **GetProperties**(MbProperties & *properties*),

void **SetProperties**(MbProperties & *properties*)

обеспечивают выдачу и редактирование внутренних данных геометрических объектов. Метод **GetProperties** добавляет к множеству *properties* данные объекта в виде наследников класса MbProperty.

О.3. КРИВЫЕ ДВУМЕРНОГО ПРОСТРАНСТВА

Двумерные кривые используются для описания области определения параметров поверхностей, построения плоских эскизов, построения трёхмерных кривых на поверхностях, кривых пересечения поверхностей, проекций трёхмерных кривых на поверхности и плоскости локальных систем координат. Многие двумерные кривые устроены аналогично трёхмерным кривым с той разницей, что вместо трёхмерных точек и векторов в двумерных кривых используются двумерные точки и векторы. Векторы, радиусы-векторы точек, матрицы в двумерном пространстве будем обозначать буквами латинского алфавита, выделенными *полужирным наклонным* шрифтом.

О.3.1. Двумерная кривая MbCurve

Абстрактный класс MbCurve объявлен в файле curve.h.

Двумерная кривая MbCurve является наследником класса [MbPlaneItem](#) рис. О.3.1.1.

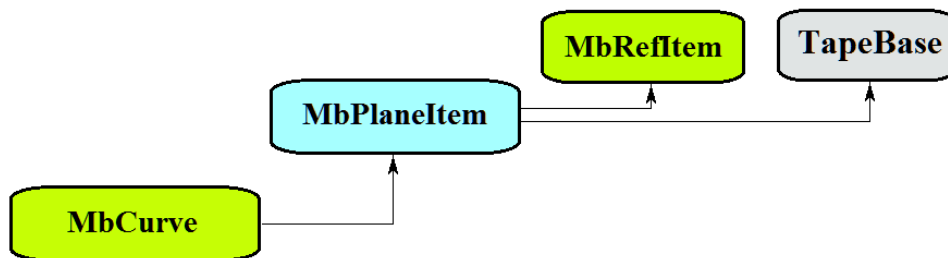


Рис. О.3.1.1.

Двумерная кривая является абстрактным классом. В геометрическом ядре C3D реализованы следующие двумерные кривые, которые являются наследниками класса MbCurve:

[MbLine](#) – двумерная прямая линия,

[MbLineSegment](#) – двумерный отрезок прямой,

[MbArc](#) – двумерная дуга эллипса,

[MbPolyline](#) – двумерная ломаная линия,

[MbNurbs](#) – двумерная B-кривая (NonUniform Rational B-Spline),

[MbBezier](#) – двумерная составная кривая Безье,

[MbHermit](#) – двумерная кривая Эрмита,

[MbCubicSpline](#) – двумерный кубический сплайн,

[MbOffsetCurve](#) – двумерная эквидистантная кривая,

[MbTrimmedCurve](#) – двумерная усеченная кривая,

[MbReparamCurve](#) – двумерная репараметризованная кривая,

[MbCharCurve](#) – двумерная кривая, координатные функции которой заданы в символьном виде,

[MbCosinusoid](#) – двумерная косинусоида,

[MbPointCurve](#) – кривая-точка,

[MbProjCurve](#) – проекционная кривая,

[MbContour](#) – двумерный контур (составная кривая),

[MbContourWithBreaks](#) – двумерный контур с разрывами.

Двумерная кривая MbCurve представляет собой векторную функцию

$$\text{curve}(t) = [u(t) \quad v(t)]$$

скалярного параметра t , принимающего значения на отрезке $[t_{\min}, t_{\max}]$. Кривая представляет собой непрерывное отображение некоторого участка числовой оси в двумерное пространство. Двумерным пространством будут служить плоскость XY локальной трёхмерной системы координат и область определения параметров поверхности. Область изменения параметра кривой есть отрезок $[t_{\min}, t_{\max}]$ в

одномерном пространстве. Координаты $u(t)$, $v(t)$ точки кривой $curve(t)$ являются однозначными непрерывными функциями параметра t .

Граничные значения t_{\min} и t_{\max} области определения параметра выдают методы кривой `double GetTMin()` и `double GetTMax()`, соответственно.

Кривую будем называть периодической, если существует $p > 0$, такое, что $curve(t \pm kp) = curve(t)$, где k – целое число. Метод `bool IsClosed()` периодической кривой возвращает `true`. Метод `double GetPeriod()` периодической кривой или кривой, которая может быть расширена до периодической, выдает период p . Область определения параметра периодической кривой всегда лежит в пределах одного периода.

Основным методом кривой является метод `void PointOn(double & t, MbCartPoint & r)`.

Он выдаёт радиус-вектор r точки кривой для заданного параметра t . Методы

`void FirstDer(double & t, MbVector & rt),`

`void SecondDer(double & t, MbVector & rtt),`

`void ThirdDer(double & t, MbVector & rttt)`

выдают соответственно первую r_t , вторую r_{tt} и третью r_{ttt} производные радиуса-вектора кривой для заданного параметра t . Перечисленные методы корректируют параметр кривой при его выходе за пределы области определения (исключение составляет прямая [MbLine](#)). При выходе параметра t кривой за пределы отрезка $[t_{\min}, t_{\max}]$ непериодические кривые смещают параметр t к ближайшей границе t_{\min} или t_{\max} , а периодические кривые добавляют или вычитают необходимое количество периодов.

Метод

`void _PointOn(double t, MbCartPoint & r)`

выдаёт радиус-вектор r точки кривой для заданного параметра t как в области определения параметра t кривой, так и за её пределами. В общем случае непериодическая кривая за пределами области определения параметра продолжается по касательной в крайней точке. Исключение составляют периодические кривые, дуга ([MbArc](#)), косинусоида ([MbCosinusoid](#)), символьная кривая ([MbCharacterCurve](#)) и усечённая кривая ([MbTrimmedCurve](#)) в пределах базовой кривой. Периодические кривые за пределами области определения параметра продолжают циклически.

Методы

`void _FirstDer(double t, MbVector & rt),`

`void _SecondDer(double t, MbVector & rtt),`

`void _ThirdDer(double t, MbVector & rttt)`

выдают соответственно первую r_t , вторую r_{tt} и третью r_{ttt} производные радиуса-вектора кривой для заданного параметра t как в области определения кривой, так и за её пределами.

Кривые перегружают такие методы двумерного геометрического объекта как:

методы, обслуживающие преобразование геометрического объекта,

`void Move(const MbVector & v, MbRegTransform * iReg = NULL, ...),`

`void Rotate(const MbCartPoint & p, const MbDirection & angle, MbRegTransform * iReg = NULL, ...),`

`void Transform(const MbMatrix & m, MbRegTransform * iReg = NULL, ...),`

методы, обеспечивающие копирование, проверку на совпадение, проверку на возможность сделать совпадающими, делающие объекты совпадающими,

`MbPlaneItem & Duplicate(MbRegDuplicate * iReg = NULL),`

`bool IsSame(const MbPlaneItem & item),`

`bool IsSimilar(const MbPlaneItem & item),`

`bool SetEqual(const MbPlaneItem & item),`

методы, возвращающие тип из перечисления геометрических объектов,

`MbPlaneType IsA(),`

`MbPlaneType Type(),`

`MbPlaneType Family(),`

методы, обеспечивающие выдачу и редактирование внутренних данных объекта,

`MbProperty & CreateProperty(MbPrompt name),`

`void GetProperties(MbProperties & properties),`

`void SetProperties(MbProperties & properties).`

Все кривые, кроме [MbContour](#) и [MbContourWithBreaks](#), как правило не имеют изломов. [MbContour](#) и [MbContourWithBreaks](#) представляют собой составные кривые и могут иметь изломы в точках сочленения составляющих их сегментов.

О.3.2. Двумерная прямая MbLine

Класс MbLine объявлен в файле cur_line.h.

Двумерная прямая линия MbLine описывается начальной точкой [MbCartPoint](#) *origin* и вектором направления [MbVector](#) *direction*, рис. О.3.2.1.

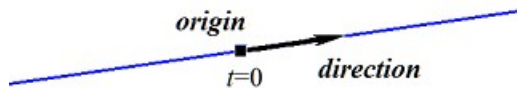


Рис. О.3.2.1.

В методе [PointOn](#)(double & *t*, [MbCartPoint](#) & *r*) радиус-вектор прямой *r* описывается векторной функцией

$$r(t) = origin + t direction.$$

Прямая линия ведёт себя как бесконечный объект, хотя в своих данных имеет граничные значения параметра *tmin* и *tmax*. Заметим, что в отличие от других кривых, в методах вычисления радиуса-вектора и его производных прямая не корректирует параметр *t* при его выходе за предельные значения *tmin* и *tmax*.

О.3.3. Двумерный отрезок прямой MbLineSegment

Класс MbLineSegment объявлен в файле cur_line_segment.h.

Двумерный отрезок прямой MbLineSegment описывается начальной точкой [MbCartPoint](#) *point1* и конечной точкой [MbCartPoint](#) *point2*, рис. О.3.3.1.



Рис. О.3.3.1.

В методе [PointOn](#)(double & *t*, [MbCartPoint](#) & *r*) радиус-вектор отрезка *r* описывается векторной функцией

$$r(t) = (1 - t) point1 + t point2.$$

Область определения параметра отрезка располагается в пределах от нуля до единицы. Начальной точке отрезка *point1* соответствует параметр $t_{min}=0$, конечной точке отрезка *point2* соответствует параметр $t_{max}=1$.

О.3.4. Двумерная дуга эллипса MbArc

Класс MbArc объявлен в файле cur_arc.h.

Двумерная дуга эллипса является наследником кривой [MbCurve](#). Дуга эллипса MbArc описывается двумя радиусами *a* и *b*, двумя углами *trim1* и *trim2* и направлением *sense*, заданными в локальной системе координат [MbPlacement](#) *position*.

Углы $trim1$ и $trim2$ отсчитываются по дуге в направлении движения от вектора $position.axisX$ к вектору $position.axisY$. Углы $trim1$ и $trim2$ будем называть параметрами усечения. Значения параметров усечения, равные нулю и 2π , соответствуют точке на оси $position.axisX$. Параметр кривой t принимает значения на отрезке: $0 \leq t \leq |trim2 - trim1|$. Кривая может быть периодической. У периодической кривой $|trim2 - trim1| = 2\pi$. Параметр $sense$ принимает значения $+1$ или -1 и указывает направление построения дуги. Если $sense = +1$, то $trim1 < trim2$ и дуга строится от параметра $trim1$ в сторону возрастания угла. Если $sense = -1$, то $trim1 > trim2$ и дуга строится от параметра $trim1$ в сторону уменьшения угла.

В методе **PointOn**($double \& t$, **MbCartPoint** & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = position.origin + a \cos(trim1 + (sense)t) position.axisX + b \sin(trim1 + (sense)t) position.axisY.$$

Дуга эллипса приведена на рис. О.3.4.1.

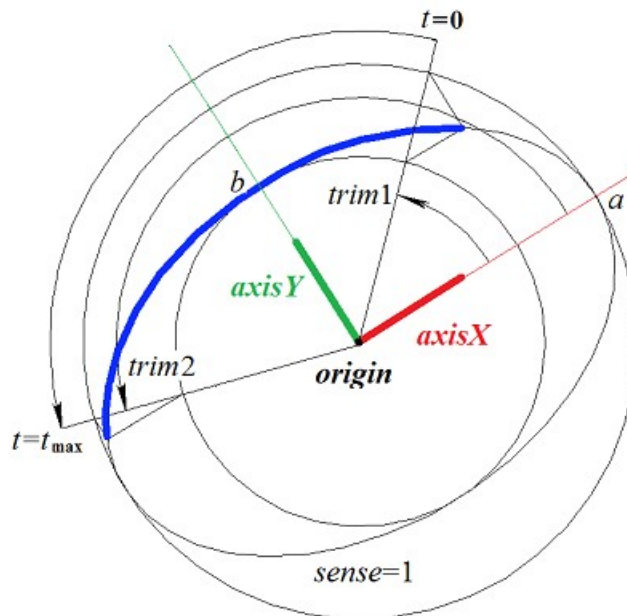


Рис. О.3.4.1.

Радиусы кривой должны быть больше нуля: $a > 0$, $b > 0$. Для параметров усечения должны соблюдаться неравенства: $trim1 < trim2$ при $sense = 1$ и $trim1 > trim2$ при $sense = -1$.

Локальная система координат $position$ может быть как правой, так и левой. Если локальная система координат правая и $sense = +1$ или локальная система координат левая и $sense = -1$, то дуга направлена против движения часовой стрелки.

О.3.5. Двумерная ломаная линия MbPolyline

Класс MbPolyline объявлен в файле cur_polyline.h.

Ломаная является наследником кривой PolyCurve. Двумерная ломаная MbPolyline описывается количеством сегментов $segmentsCount$, множеством контрольных точек $SAgray < MbCartPoint > pointList$ и признаком периодичности кривой $closed$.

Кривая проходит через множество точек $pointList[i]$, $i = 0, \dots, segmentsCount$. при значениях параметра $t = 0, \dots, segmentsCount$. Если $closed = true$, то кривая содержит сегмент, соединяющий последнюю точку множества $pointList[segmentsCount - 1]$ с начальной точкой $pointList[0]$. Параметр кривой t принимает значения на отрезке: $0 \leq t \leq segmentsCount$.

В методе **PointOn**($double \& t$, **MbCartPoint** & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = pointList[i] (1-w) + pointList[i+1] w,$$

где $w = \frac{t - t_i}{t_{i+1} - t_i}$, а $t_i \leq t \leq t_{i+1}$.

Ломаная является простейшей кривой, построенной по набору точек. Она состоит из отрезков, последовательно соединяющих контрольные точки. Кривая может быть периодической. Период периодической кривой равен *segmentsCount*. Периодическая ломаная приведена на рис. О.3.5.1.

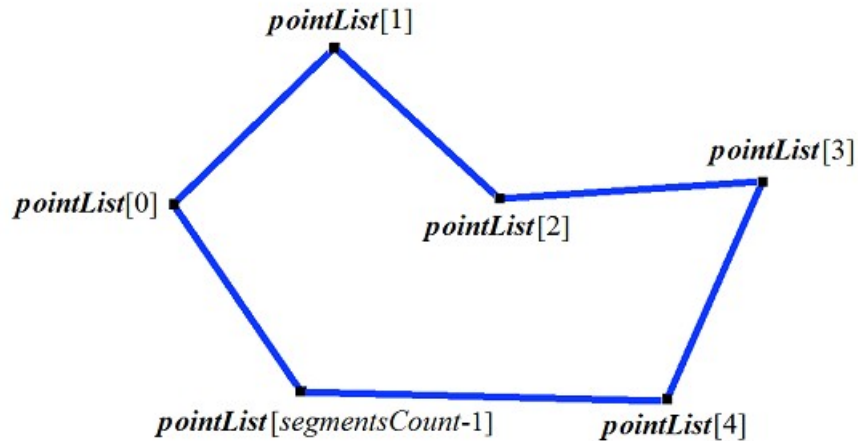


Рис. О.3.5.1.

Производные кривой в контрольных точках (при целочисленных значениях параметра) теряют непрерывность по длине и направлению. Производные кривой в контрольных точках имеют особое направление и длину.

О.3.6. Двумерная NURBS-кривая MbNurbs

Класс MbNurbs объявлен в файле `cur_nurbs.h`.

В-кривая или NURBS-кривая получила название от первых букв сочетания слов NonUniform Rational B-Spline. Кривая является наследником кривой MbPolyCurve. Кривая описывается множеством двумерных контрольных точек `SArray<MbCartPoint>pointList`, множеством весов контрольных точек *weights*, узловым вектором *knots*, порядком сплайна *degree*, параметром формы кривой *form* и признаком периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривая построена на основе В-сплайнов. Узловой вектор *knots* представляет собой неубывающую последовательность действительных чисел и определяет область определения параметра кривой и форму кривой. Параметр формы кривой *form* в общем случае принимает значение `ncf_Unspecified`, в частных случаях он хранит информацию об исходной кривой, с которой была получена NURBS-копия. Порядок *degree* NURBS-кривой равен порядку разделённых разностей, по которым вычисляются В-сплайны. Пусть узловой вектор содержит *knotsCount* элементов, а множество контрольных точек содержит *pointsCount* элементов. Для не периодической NURBS-кривой количества элементов в множествах связаны равенством $knotsCount = pointsCount + degree$. Для периодической NURBS-кривой количество элементов в множествах связаны равенством $knotsCount = pointsCount + 2degree - 1$.

В методе `PointOn`(`double & t`, `MbCartPoint & r`) радиус-вектор кривой *r* описывается векторной функцией

$$r(t) = \frac{\sum_{j=0}^{pointsCount-1} N_{j,degree}(t) weight[j] pointList[j]}{\sum_{j=0}^{pointsCount-1} N_{j,degree}(t) weight[j]},$$

где $N_j^{degree}(t)$ – B-сплайны порядка $degree$ для j -ой контрольной точки $pointList[j]$. NURBS-кривая четвёртого порядка приведена на рис. О.3.6.1.

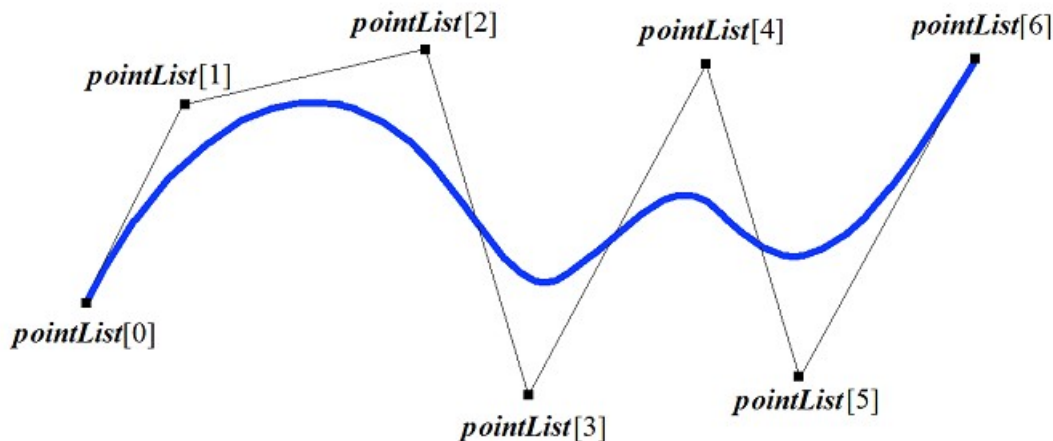


Рис. О.3.6.1.

Кривая может быть периодической. Периодическая NURBS-кривая приведена на рис. О.3.6.2.

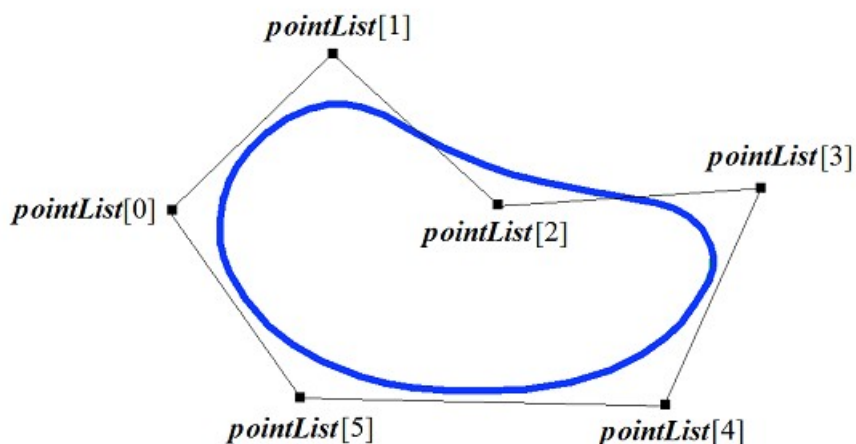


Рис. О.3.6.2.

Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$, где $tmin = knots[degree-1]$, $tmax = knots[knotsCount-degree]$.

Форма NURBS-кривой зависит от расположения контрольных точек, от веса контрольных точек и от значений узлового вектора. NURBS-кривая в общем случае не проходит через множество точек $pointList[i]$, $i=0, \dots, pointsCount-1$. Чтобы незамкнутая NURBS-кривая проходила через крайние контрольные точки, требуется чтобы первые $degree$ элементов и последние $degree$ элементов узлового вектора $knots$ совпадали. Чем больше вес контрольной точки, тем ближе к этой точке проходит кривая при прочих равных условиях.

Каждая кривая может построить свою NURBS-копию виртуальным методом `NurbsCurve(const MbNurbsParameters & tParameters)`.

О.3.7. Двумерная кривая Эрмита MbHermit

Класс MbHermit объявлен в файле cur_hermit.h.

Двумерная кривая Эрмита является наследником кривой MbPolyCurve. Кривая описывается множеством контрольных точек `SArray<MbCartPoint>pointList`, множеством производных кривой в контрольных точках `SArray<MbVector>vectorList`, множеством значений параметра кривой в контрольных точках `tList`, количеством `splinesCount` кубических сплайнов Эрмита и признаком периодичности кривой `closed`. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривая Эрмита при значении параметра $tList[i]$, $i=0,1,\dots,splinesCount$, проходит через контрольную точку $pointList[i]$ и имеет в ней производную $vectorList[i]$. Кривая построена на основе $splinesCount$ двумерных сплайнов Эрмита третьей степени, которые гладко стыкуются между собой. Каждый кубический сплайн Эрмита описывает участок кривой между двумя соседними контрольными точками. Каждый кубический сплайн Эрмита определяется двумя крайними точками и двумя производными кривой в этих точках.

При вычислении радиуса-вектора точки кривой Эрмита сначала по значению параметра t кривой определяется номер i рабочего участка (номер кубического сплайна Эрмита), из условия $tList[i] \leq t \leq tList[i+1]$. Радиус-вектор кривой вычисляется как радиус-вектор найденного участка для его локального параметра w , который определяется по $tList[i]$ и $tList[i+1]$.

В методе **PointOn**(double & t , **MbCartPoint** & r) радиус-вектор кривой r описывается векторной функцией найденного участка для его локального параметра w :

$$r(t) = (1 - 3w^2 + 2w^3) pointList[i] + (3w^2 + 2w^3) pointList[i+1] + ((w - 2w^2 + w^3) vectorList[i] + (-w^2 + w^3) vectorList[i+1]) (tList[i+1] - tList[i]),$$

где $w = \frac{t - tList[i]}{tList[i+1] - tList[i]}$, а $tList[i] \leq t \leq tList[i+1]$.

Кривая Эрмита приведена на рис. О.3.7.1.

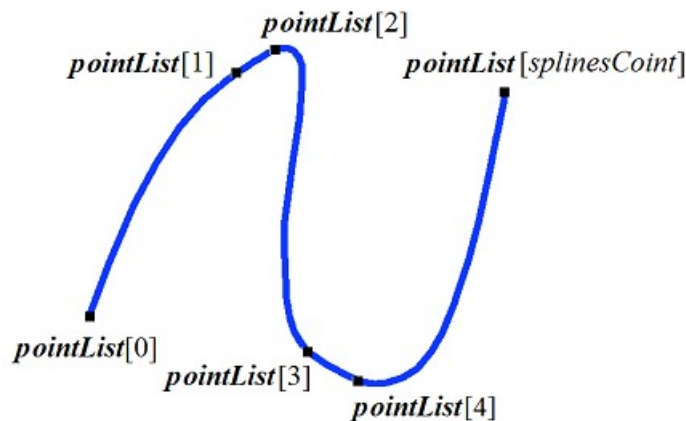


Рис. О.3.7.1.

Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$, где $tmin = tList[0]$, $tmax = tList[splinesCount]$. Кривая может быть периодической.

Форма кривой зависит от расположения контрольных точек, от производных кривой в контрольных точках и от множества $tList$ значений параметра в контрольных точках. При построении кривой только по контрольным точкам значения параметра кривой в контрольных точках $tList[i]$, $i=0,1,\dots,splinesCount$, изменяются пропорционально расстоянию между точками, а производные $vectorList[i]$, $i=1,2,\dots,splinesCount-1$ вычисляются путём построения параболы, проходящей по трём соседним точкам $pointList[i-1]$, $pointList[i]$, $pointList[i+1]$ при соответствующих значениях параметра $tList[i-1]$, $tList[i]$, $tList[i+1]$, и вычисления производной параболы в средней точке.

О.3.8. Двумерная составная кривая Безье MbBezier

Класс MbBezier объявлен в файле cur_bezier.h.

Двумерная составная кривая Безье является наследником кривой MbPolyCurve. Кривая описывается множеством контрольных точек **SArray**<**MbCartPoint**> $pointList$, количеством $splinesCount$ кривых Безье и признаком периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривая построена на основе $splinesCount$ кривых Безье третьей степени, которые гладко стыкуются между собой. Каждая кривая Безье определяется четырьмя контрольными точками и проходит только через две крайние точки. Составная кривая используется для построения сплайна, проходящего через

заданные точки. Заданные точки служат точками сочленения кривых Безье третьей степени. Пара внутренних контрольных точек каждой кривой Безье третьей степени определяется из условия гладкой стыковки кривой с соседними кривыми. Для составной кривой количество контрольных точек равно $3(\text{splinesCount}+1)$. У непериодической составной кривой первая $\text{pointList}[0]$ и последняя контрольные точки не используются.

На каждой кривой Безье третьей степени параметр составной кривой увеличивается на единицу. При вычислении радиуса-вектора точки составной кривой сначала по значению параметра t кривой определяется номер рабочего участка (номер кривой Безье третьей степени), который равен максимальному целому числу, не превосходящему t . Пусть номер кривой Безье третьей степени равен n . Далее определяется дробная часть параметра $w=t-n$. Радиус-вектор составной кривой вычисляется как радиус-вектор найденного участка для его локального параметра w .

В методе `PointOn(double & t, MbCartPoint & r)` радиус-вектор кривой r описывается векторной функцией найденного участка для его локального параметра w :

$$r(t) = \sum_{j=0}^3 \frac{3!}{j!(3-j)!} w^j (1-w)^{3-j} \text{pointList}[3n+j],$$

где $w=t-n$, $n \leq t < n+1$, $0 \leq w < 1$, $B_j^3(w) = \frac{3!}{j!(3-j)!} w^j (1-w)^{3-j}$ – функции Бернштейна третьей степени для j -ой, $j=0,1,2,3$, контрольной точки $\text{pointList}[3n+j]$ найденного участка с номером n .

Составная кривая Безье приведена на рис. О.3.8.1.

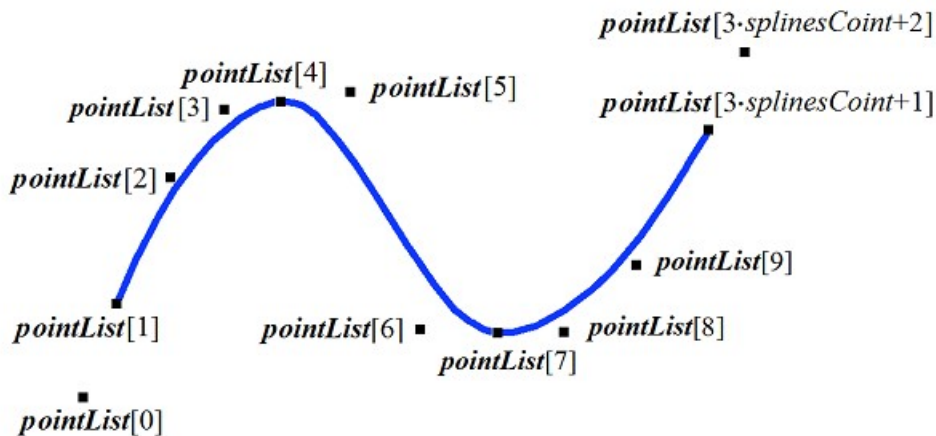


Рис. О.3.8.1.

Параметр кривой t принимает значения на отрезке: $0 \leq t \leq \text{splinesCount}$. Кривая может быть периодической. Период периодической кривой равен splinesCount .

При целочисленных значениях параметра кривая проходит через контрольные точки, например, при параметре $t=n$ кривая пройдет через контрольную точку $\text{pointList}[3n]$, $n=0,1,\dots,\text{splinesCount}$. Производные кривой в точках сочленения кривых Безье третьей степени (при целочисленных значениях параметра) теряют непрерывность по длине.

О.3.9. Двумерная кубический сплайн MbCubicSpline

Класс MbCubicSpline объявлен в файле `cur_cubic_spline.h`.

Двумерный кубический сплайн является наследником кривой MbPolyCurve. Кривая описывается множеством двумерных контрольных точек `SArray<MbCartPoint>pointList`, множеством вторых производных кривой в контрольных точках `SArray<MbVector>vectorList`, множеством значений параметра кривой в контрольных точках `tList`, максимальным значением индекса множества параметров `splinesCount` и признаком периодичности кривой `closed`. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кубический сплайн при значении параметра $tList[i]$, $i=0,1,\dots,\text{splinesCount}$, проходит через контрольную точку $\text{pointList}[i]$ и имеет в ней вторую производную $\text{vectorList}[i]$. Кривая построена так,

что при переходе из точки $pointList[i]$ в точку $pointList[i+1]$ вторая производная радиуса вектора кривой изменяется линейно от $vectorList[i]$ до $vectorList[i+1]$.

При вычислении радиуса-вектора точки составной кривой сначала по значению параметра t кривой определяется номер i рабочего участка, из условия $tList[i] \leq t \leq tList[i+1]$. Радиус-вектор кривой вычисляется по значениям $pointList[i]$, $pointList[i+1]$, $vectorList[i]$, $vectorList[i+1]$ найденного участка для локального параметра w , который определяется по $tList[i]$ и $tList[i+1]$.

В методе **PointOn**($double \& t$, [MbCartPoint](#) & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = (1-w) pointList[i] + w pointList[i+1] + \left((-2w+3w^2-w^3) vectorList[i] + (-w+w^3) vectorList[i+1] \right) \frac{(tList[i+1]-tList[i])^2}{6},$$

где $w = \frac{t-tList[i]}{tList[i+1]-tList[i]}$, а $tList[i] \leq t \leq tList[i+1]$.

Кубический сплайн, построенный по тем же контрольным точкам, что и составная кривая Эрмита, приведён на рис. О.3.9.1.

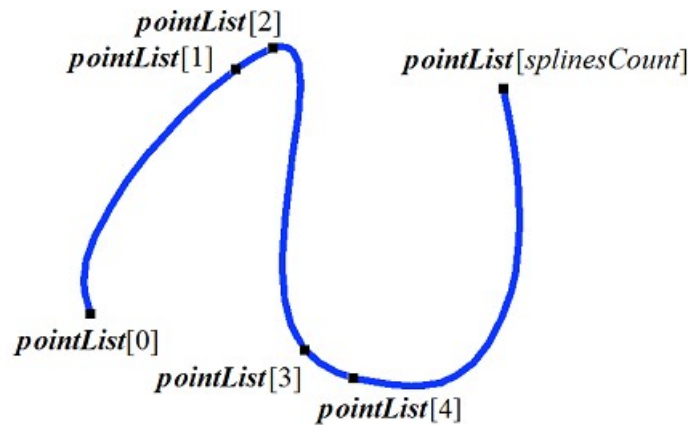


Рис. О.3.9.1.

Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$, где $tmin = tList[0]$, $tmax = tList[splinesCount]$. Кривая может быть периодической.

Форма кривой зависит от расположения контрольных точек и от множества $tList$ значений параметра в контрольных точках. При построении кривой только по контрольным точкам значения параметра кривой в контрольных точках $tList[i]$, $i=0,1,\dots,splinesCount$, изменяются пропорционально расстоянию между точками, а вторые производные $vectorList[i]$, $i=1,2,\dots,splinesCount-1$ вычисляются путём решения системы уравнений.

О.3.10. Двумерная усечённая кривая MbTrimmedCurve

Класс MbTrimmedCurve объявлен в файле cur_trimmed_curve.h.

Двумерная усечённая кривая описывается базовой кривой [MbCurve](#)* *basisCurve*, начальным параметром усечения базовой кривой *trim1*, конечным параметром усечения базовой кривой *trim2* и признаком совпадения направлений базовой кривой и усечённой кривой *sense*.

Усечённая кривая совпадает с базовой кривой на участке, определённом параметрами *trim1* и *trim2*, но может иметь противоположное с ним направление. Если *sense=1*, то *trim1 < trim2* и усечённая кривая совпадает по направлению с базовой кривой. Если *sense=-1*, то *trim2 < trim1* и усечённая кривая направлена против базовой кривой.

В методе **PointOn**($double \& t$, [MbCartPoint](#) & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = basisCurve(trim1 + sense * t).$$

Усечённая кривая приведена на рис. О.3.10.1.

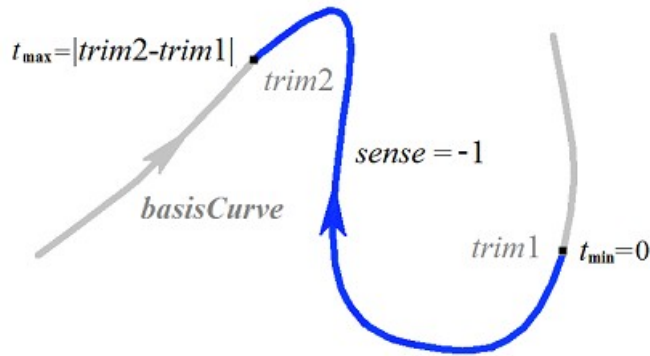


Рис. О.3.10.1.

Параметр кривой t принимает значения на отрезке: $0 \leq t \leq \text{sense}(\text{trim2} - \text{trim1})$.

Усечённая кривая теоретически может применяться для изменения направления кривой, но лучше пользоваться методом **Inverse()**.

Усечённая кривая может применяться для изменения положения начальной точки периодической кривой. Для этого базовая кривая должна быть периодической и $\text{trim2} = \text{trim1} + \text{period}$. В этом случае усечённая кривая так же будет периодической.

В качестве базовой кривой для усеченной кривой не должна использоваться другая усеченная кривая, а должна использоваться базовая кривая последней с соответствующим пересчетом параметров усечения.

Каждая кривая может построить свою усеченную копию виртуальным методом **Trimmed**(double $t1$, double $t2$, int $sense$).

О.3.11. Двумерная репараметризованная кривая MbReparamCurve

Класс MbReparamCurve объявлен в файле cur_reparam_curve.h.

Двумерная репараметризованная кривая описывается базовой кривой **MbCurve*** *basisCurve*, начальным параметром *tmin*, конечным параметром *tmax* и производной *dt* параметра базовой кривой по параметру репараметризованной кривой.

Репараметризованная кривая полностью совпадает с базовой кривой, но имеет другую область изменения параметра.

В методе **PointOn**(double & t , **MbCartPoint** & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = \text{basisCurve}(v(t)),$$

где $v(t) = b_{\min} \frac{\text{trim2} - t}{\text{trim2} - \text{trim1}} + b_{\max} \frac{t - \text{trim1}}{\text{trim2} - \text{trim1}}$, b_{\min} , b_{\max} – граничные значения области определения параметра базовой кривой.

Параметр кривой t принимает значения на отрезке: $t_{\min} \leq t \leq t_{\max}$.

Репараметризованная кривая совпадает с базовой кривой, но имеет другую область определения параметра. Кривая с измененной длиной параметра применяется для согласования областей изменения параметра двух кривых. Например, если требуется, чтобы отрезок и дуга имели одинаковые области изменения параметра, то на базе одной из указанных кривых создаётся репараметризованная кривая с областью изменения параметра, взятой у другой кривой.

В качестве базовой кривой для репараметризованной кривой не должна использоваться другая репараметризованная кривая, а должна использоваться базовая кривая последней.

О.3.12. Двумерная эквидистантная кривая MbOffsetCurve

Класс MbOffsetCurve объявлен в файле cur_offset_curve.h.

Двумерная эквидистантная кривая описывается базовой кривой [MbCurve](#)* *basisCurve*, смещением [MbVector](#) *distance*, изменением минимального параметра базовой кривой *dmin*, изменением максимального параметра базовой кривой *dmax*, минимальным параметром базовой кривой *tmin*, максимальным параметром базовой кривой *tmax*, матрицей преобразования [MbMatrix](#) *transform* и признаком периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Двумерная эквидистантная кривая представляет собой кривую, соответствующие по параметру точки которой смещены на расстояние *distance* от соответствующих точек базовой кривой *basisCurve*. Область изменения параметра двумерной эквидистантной кривой отличается от области изменения параметра базовой кривой на *dmin* для минимального значения и на *dmax* для максимального значения.

Вычисление радиуса-вектора точки эквидистантной кривой выполняется следующим образом. Для заданного параметра базовой кривой вычисляется точка и нормаль. Далее точка смещается на расстояние *distance* вдоль нормали кривой.

В методе [PointOn](#)(double & *t*, [MbCartPoint](#) & *r*) радиус-вектор кривой *r* описывается векторной функцией

$$r(t) = \text{basisCurve}(t) + \text{normal}(t) \cdot \text{distance},$$

где *normal(t)* – нормаль базовой кривой, получена поворотом на 90 градусов (против часовой стрелки) касательной базовой кривой в заданной точке.

Эквидистантная и базовая кривые приведены на рис. О.3.12.1.

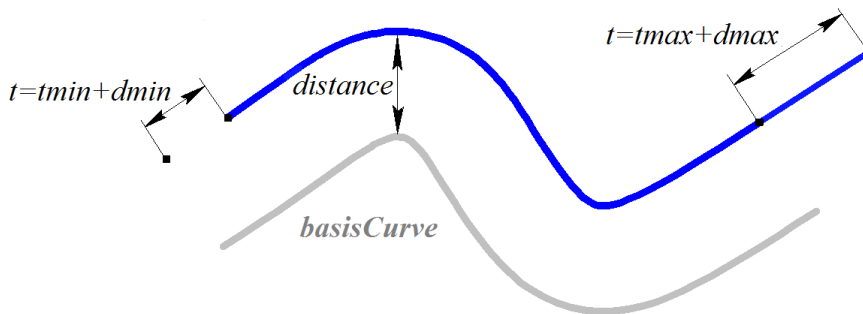


Рис. О.3.12.1.

Параметр кривой *t* принимает значения на отрезке: $t_{min}+dmin \leq t \leq t_{max}+dmax$. При выходе параметра за пределы области определения радиус-вектор точки базовой кривой вычисляется методом [_PointOn](#)(double *t*, [MbCartPoint](#) & *r*). При *distance*=0, *dmin*=0, *dmax*=0 эквидистантная кривая совпадает с базовой кривой.

В качестве базовой кривой для эквидистантной кривой не должна использоваться другая эквидистантная кривая, а должна использоваться базовая кривая последней с соответствующим пересчетом смещения.

Каждая кривая может построить эквидистантную кривую виртуальным методом [Offset](#)(double *distance*).

О.3.13. Двумерная символьная кривая MbCharCurve

Класс MbCharacterCurve объявлен в файле cur_character_curve.h.

Символьная кривая описывается координатными функциями *xFunction*, *yFunction*, локальной системой координат [MbPlacement](#) *position* функций, матрицей трансформации *transform*, граничными значениями области определения параметра кривой *tmin* и *tmax*, признаком периодичности кривой *closed* и типом системы координат (декартова, полярная) *coordinateType*, в которой заданы

координатные функции. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Координатные функции $xFunction(t)$, $yFunction(t)$ символьной кривой представляют собой скалярные функции общего параметра t и заданы в виде символьных выражений. Для каждого символьного выражения выполнен лексический анализ и построено дерево, которое вычисляет значение символьного выражения для заданного параметра и производные символьного выражения по параметру. Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$.

В методе **PointOn**(double & t , [MbCartPoint](#) & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = [xFunction(t) \quad yFunction(t)].$$

Символьная кривая приведена на рис. О.3.13.1.

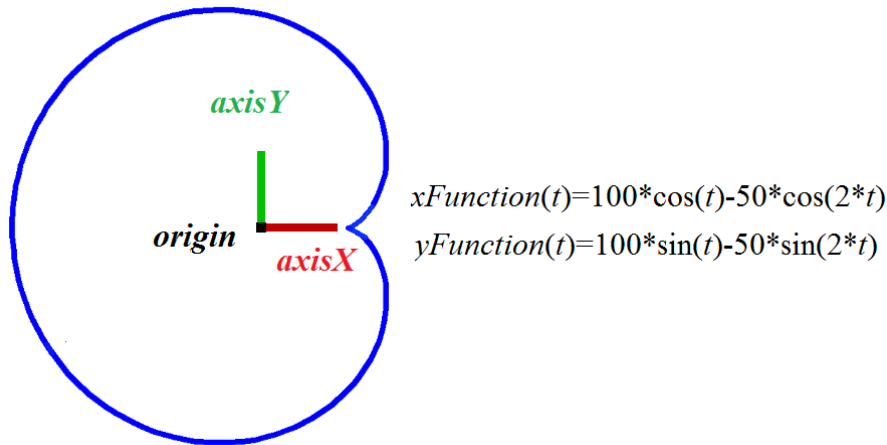


Рис. О.3.13.1.

Кривая может быть периодической. Символьные выражения на области определения кривой должны описывать непрерывные конечные и однозначные функции.

О.3.14. Двумерная косинусоида MbCosinusoid

Класс MbCosinusoid объявлен в файле cur_cosinusoid.h.

Двумерная косинусоида описывается локальной системой координат [MbPlacement](#) *position*, циклической частотой *frequency*, начальной фазой *phase*, амплитудой *amplitude*, минимальным параметром кривой *tmin*, максимальным параметром кривой *tmax*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Двумерная косинусоида представляет собой функцию косинуса, аргумент которой задается вдоль вектора *position.axisX*, а значение функции откладывается вдоль вектора *position.axisY*. Функция имеет амплитуду *amplitude*, частоту *frequency* и начальную фазу *phase*. В методе **PointOn**(double & t , [MbCartPoint](#) & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = position.origin + (((tmin+t-phase) / frequency) position.axisX) + (amplitude \cos(tmin+t) position.axisY).$$

Косинусоида приведена на рис. О.3.14.1.

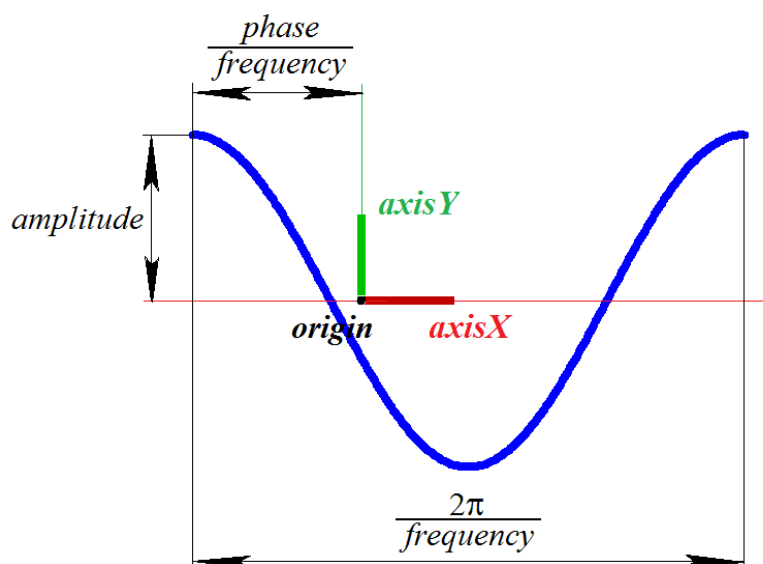


Рис. O.3.14.1.

Параметр кривой t принимает значения на отрезке: $t_{min} \leq t \leq t_{max}$. Для параметров должно соблюдаться неравенство: $t_{min} < t_{max}$. Кривая не может быть периодической. Амплитуда и частота кривой должны быть больше нуля: $amplitude > 0, frequency > 0$.

Локальная система координат **position** может быть как правой, так и левой. Косинусоида используется для описания пересечения цилиндрической поверхности и плоскости.

O.3.15. Двумерная кривая-точка MbPointCurve

Класс MbPointCurve объявлен в файле cur_point_curve.h.

Двумерная кривая-точка описывается точкой [MbCartPoint](#) **point**, минимальным параметром кривой t_{min} , максимальным параметром кривой t_{max} и признаком периодичности кривой **closed**.

В методе **PointOn**(double & t , [MbCartPoint](#) & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = point.$$

Параметр кривой t принимает значения на отрезке: $t_{min} \leq t \leq t_{max}$. Кривая может быть периодической. Для параметров должно соблюдаться неравенство: $t_{min} < t_{max}$.

Двумерная кривая-точка используется в паре с другой двумерной кривой для описания пересечения поверхностей, одна из которых имеет особую точку, например, полюс. Параметры t_{min} , t_{max} , **closed** кривой-точки соответствуют параметрам двумерной кривой, используемой в паре с кривой-точкой.

O.3.16. Двумерная проекционная кривая MbProjCurve

Класс MbProjCurve объявлен в файле cur_projection_curve.h.

Двумерная проекционная кривая описывается пространственной кривой [MbCurve3D](#)* **spaceCurve**, поверхностью [MbSurface](#)* **surface** и двумерной кривой [MbCurve](#)* **curve**. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Двумерная проекционная кривая представляет собой проекцию пространственной кривой **spaceCurve** на поверхность **surface**, которая приближенно описывается двумерной кривой **curve** в области определения параметров поверхности. Области определения параметров кривых **spaceCurve** и **curve** совпадают. Двумерная кривая **curve**, как правило, является сплайном, контрольные точки которого получены проецированием точек пространственной кривой **spaceCurve** на поверхность **surface**. Параметризация кривой **curve** согласована с параметризацией пространственной кривой в

контрольных точек. Двумерная кривая *curve* может располагаться за пределами области определения параметров поверхности.

В методе `PointOn(double & t, MbCartPoint & r)` радиус-вектор кривой *r* описывается векторной функцией

$$r(t) = [u \ v],$$

где *u*, *v* – параметры проекции точки `spaceCurve(t)` на поверхность `surface`. Начальное приближение параметров *u* и *v* вычисляется методом `curve->PointOn(t,point)`, $u=point.x$, $v=point.y$. Далее параметры *u* и *v* уточняются итерационным методом, использующим уравнения

$$\begin{aligned} \text{deriveU} \cdot (\text{spaceCurve}(t) - \text{surface}(u, v)) &= 0, \\ \text{deriveV} \cdot (\text{spaceCurve}(t) - \text{surface}(u, v)) &= 0, \end{aligned}$$

где `deriveU` и `deriveV` – частные производные радиуса-вектора поверхности, которые вычисляются методами `surface->_DeriveU(u,v,deriveU)` и `surface->_DeriveV(u,v,deriveV)`, соответственно.

Проекционная кривая приведена на рис. О.3.16.1.

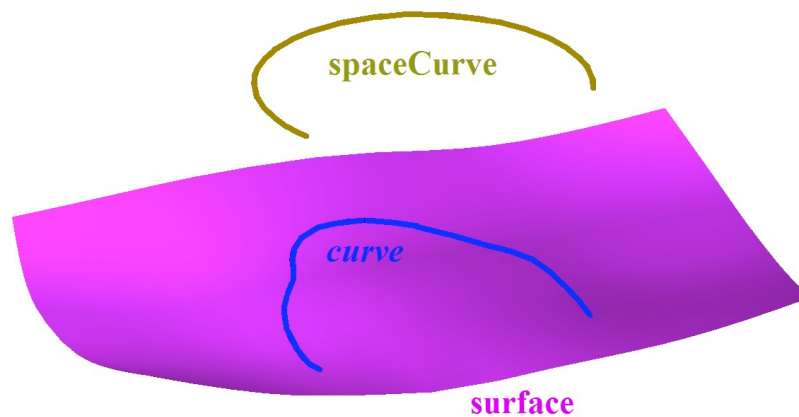


Рис. О.3.16.1.

Проекционная кривая используется для точного описания проекции пространственной кривой на поверхность.

О.3.17. Двумерный контур MbContour

Класс MbContour объявлен в файле `cur_contour.h`.

Двумерный контур MbContour описывается множеством `RPAArray<MbCurve>segments` стыкующихся друг да другом кривых и признаком периодичности кривой *closed*.

Двумерный контур представляет собой составную кривую. В отличие от других кривых контур может иметь изломы. Кривые, образующие контур, будем называть сегментами. Для сегментов контура выполняются следующие условия: начало каждого последующего сегмента совпадает с концом предыдущего сегмента. Для периодического контура начало первого сегмента совпадает с концом последнего сегмента. В общем случае в местах стыковки сегментов производные контура терпят разрыв по длине и направлению.

Начальное значение параметра контура равно нулю: $t_{\min}=0$. Параметрическая длина контура равна сумме параметрических длин составляющих его сегментов: $t_{\max} = \sum (w_{i\max} - w_{i\min})$, где $w_{i\min}$ и $w_{i\max}$ – минимальное и максимальное значение параметра *i*-го сегмента. При вычислении радиуса-вектора точки контура сначала по значению параметра определяется рабочий сегмент и значение его локального параметра, далее вычисляется радиус-вектор рабочего сегмента, который служит радиусом-вектором контура.

В методе `PointOn(double & t, MbCartPoint & r)` радиус-вектор кривой *r* описывается векторной функцией

$$r(t) = \text{segments}[k](w_k),$$

где $\text{segments}[k](w_k)$ – рабочий сегмент контура с индексом k , w_k – параметр рабочего сегмента, равный:

$$w_k = w_{k \min} + t - \sum_{i=0}^{k-1} (w_{i \max} - w_{i \min}).$$

Сегмент с индексом k определяется по значению параметра контура t из условия $\sum_{i=0}^{k-1} (w_{i \max} - w_{i \min}) \leq t < \sum_{i=0}^k (w_{i \max} - w_{i \min})$, где $w_{i \min}$ и $w_{i \max}$ – минимальное и максимальное значение параметра i -го сегмента.

Контур приведён на рис. О.3.17.1.

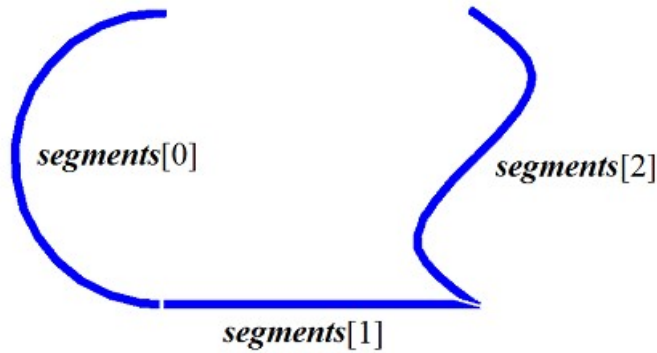


Рис. О.3.17.1.

В качестве сегментов двумерного контура не должны использоваться другие двумерные контуры. Если контур нужно построить на основе других контуров, то последние должны рассматриваться как совокупность составляющих их кривых, а не как единые кривые.

О.4. КРИВЫЕ

Кривые являются представителями семейства трёхмерных геометрических объектов [MbSpaceItem](#). Все кривые имеют общий родительский класс [MbCurve3D](#). В геометрическом ядре C3D используются кривые, которые построены с помощью аналитических функций, по набору точек, на базе кривых и на базе поверхностей. Кривые используются для построения поверхностей, а также вспомогательных элементов геометрической модели. Векторы, радиусы-векторы точек, матрицы в трехмерном пространстве будем обозначать буквами латинского алфавита, выделенными **полужирным** шрифтом.

О.4.1. Кривая MbCurve3D

Класс MbCurve3D объявлен в файле curve3d.h.

Кривая MbCurve3D является наследником класса [MbSpaceItem](#), рис. О.4.1.1.

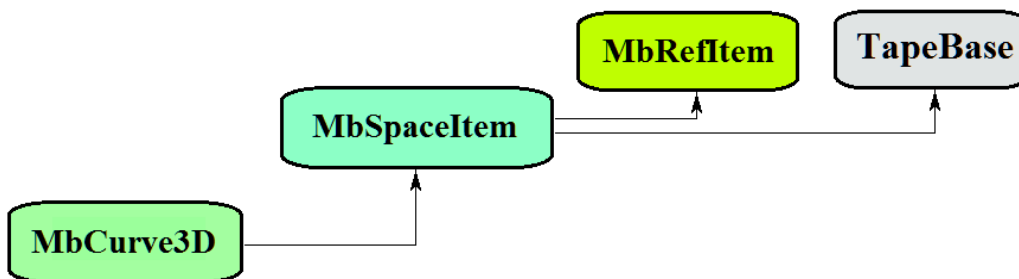


Рис. О.4.1.1.

Кривая является абстрактным классом. В геометрическом ядре C3D реализованы следующие кривые, которые являются наследниками класса MbCurve3D:

[MbLine3D](#) – прямая линия,

[MbLineSegment3D](#) – отрезок прямой,

[MbArc3D](#) – дуга эллипса,

[MbPolyline3D](#) – ломаная линия,

[MbNurbs3D](#) – B-кривая (NonUniform Rational B-Spline),

[MbBezier3D](#) – составная кривая Безье,

[MbHermit3D](#) – кривая Эрмита,

[MbCubicSpline3D](#) – кубический сплайн,

[MbOffsetCurve3D](#) – эквидистантная кривая,

[MbTrimmedCurve3D](#) – усеченная кривая,

[MbReparamCurve3D](#) – репараметризованная кривая,

[MbReparamCurve3D](#) – кривая, координатные функции которой заданы в символьном виде,

[MbConeSpiral](#) – коническая спираль,

[MbCurveSpiral](#) – спираль с прямолинейной осью и переменным радиусом,

[MbCrookedSpiral](#) – спираль с осью в виде плоской кривой,

[MbBridgeCurve3D](#) – сплайн Эрмита, соединяющий две кривые,

[MbContour3D](#) – контур (составная кривая),

[MbPlaneCurve](#) – плоская кривая в пространстве,

[MbSurfaceCurve](#) – кривая на поверхности,

[MbSilhouetteCurve](#) – силуэтная кривая поверхности,

[MbContourOnSurface](#) – контур на поверхности,

[MbContourOnPlane](#) – контур на плоскости,

[MbSurfaceIntersectionCurve](#) – кривая пересечения поверхностей.

Кривая MbCurve3D представляет собой векторную функцию

$$\mathbf{curve}(t) = [x(t) \quad y(t) \quad z(t)]$$

скалярного параметра t , принимающего значения на отрезке $[t_{\min}, t_{\max}]$. Кривая представляет собой непрерывное отображение некоторого участка числовой оси в трёхмерное пространство. Область изменения параметра кривой есть отрезок $[t_{\min}, t_{\max}]$ в одномерном пространстве. Координаты $x(t)$, $y(t)$, $z(t)$ точки кривой $\mathbf{curve}(t)$ являются однозначными непрерывными функциями параметра t .

Граничные значения t_{\min} и t_{\max} области определения параметра выдают методы кривой `double GetTMin()` и `double GetTMax()`, соответственно.

Кривую будем называть периодической, если существует $p > 0$, такое, что $\mathbf{curve}(t \pm kp) = \mathbf{curve}(t)$, где k – целое число. Метод `bool IsClosed()` периодической кривой возвращает `true`. Метод `double GetPeriod()` периодической кривой или кривой, которая может быть расширена до периодической, выдает период p . Область определения параметра периодической кривой всегда лежит в пределах одного периода.

Основным методом кривой является метод

`void PointOn(double & t, MbCartPoint3D & r)`.

Он выдаёт радиус-вектор \mathbf{r} точки кривой для заданного параметра t . Методы

`void FirstDer(double & t, MbVector3D & r_t)`,

`void SecondDer(double & t, MbVector3D & r_tt)`,

`void ThirdDer(double & t, MbVector3D & r_ttt)`

выдают соответственно первую \mathbf{r}_t , вторую \mathbf{r}_{tt} и третью \mathbf{r}_{ttt} производные радиуса-вектора кривой для заданного параметра t . Перечисленные методы корректируют параметр кривой при его выходе за пределы области определения (исключение составляет прямая `MbLine3D`). При выходе параметра t кривой за пределы отрезка $[t_{\min}, t_{\max}]$ непериодические кривые смещают параметр t к ближайшей границе t_{\min} или t_{\max} , а периодические кривые добавляют или вычитают необходимое количество периодов.

Метод

`void _PointOn(double t, MbCartPoint3D & r)`

выдаёт радиус-вектор \mathbf{r} точки кривой для заданного параметра t как в области определения параметра кривой, так и за её пределами. В общем случае непериодическая кривая за пределами области определения параметра продолжается по касательной в крайней точке. Исключение составляют периодические кривые, дуга (`MbArc3D`), спирали (`MbSpiral`), символьная кривая (`MbCharacterCurve3D`) и усечённая кривая (`MbTrimmedCurve3D`) в пределах базовой кривой. Периодические кривые за пределами области определения параметра продолжают циклически.

Методы

`void _FirstDer(double t, MbVector3D & r_t)`,

`void _SecondDer(double t, MbVector3D & r_tt)`,

`void _ThirdDer(double t, MbVector3D & r_ttt)`

выдают соответственно первую \mathbf{r}_t , вторую \mathbf{r}_{tt} и третью \mathbf{r}_{ttt} производные радиуса-вектора кривой для заданного параметра t как в области определения кривой, так и за её пределами.

Кривые перегружают такие методы трёхмерного геометрического объекта как:

методы, обслуживающие преобразование геометрического объекта,

`void Move(const MbVector3D & v, MbRegTransform * iReg = NULL)`,

`void Rotate(const MbAxis3D & axis, double angle, MbRegTransform * iReg = NULL)`,

`void Transform(const MbMatrix3D & m, MbRegTransform * iReg = NULL)`,

методы, обеспечивающие копирование, проверку на совпадение, проверку на возможность сделать совпадающими, делающие объекты совпадающими,

`MbSpaceItem & Duplicate(MbRegDuplicate * iReg = NULL)`,

`bool IsSame(const MbSpaceItem & item)`,

`bool IsSimilar(const MbSpaceItem & item)`,

`bool SetEqual(const MbSpaceItem & item)`,

методы, возвращающие тип из перечисления геометрических объектов,

`MbeSpaceType IsA()`,

`MbeSpaceType Type()`,

`MbeSpaceType Family()`,

методы, обеспечивающие выдачу и редактирование внутренних данных объекта,

`MbProperty & CreateProperty(MbePrompt name)`,

`void GetProperties(MbProperties & properties)`,

`void SetProperties(MbProperties & properties)`,

метод, наполняющий полигональную копию геометрического объекта,

CalculateWire(double sag, [MbMesh](#) & mesh).

Все кривые, кроме [MbContour3D](#), [MbContourOnSurface](#) [MbContourOnPlane](#), как правило не имеют изломов. [MbContour3D](#), [MbContourOnSurface](#) [MbContourOnPlane](#) представляют собой составные кривые и могут иметь изломы в точках сочленения составляющих их сегментов.

О.4.2. Прямая линия MbLine3D

Класс MbLine3D объявлен в файле cur_line_3d.h.

Прямая линия MbLine3D описывается начальной точкой [MbCartPoint3D](#) **origin** и вектором направления [MbVector3D](#) **direction**, рис. О.4.2.1.

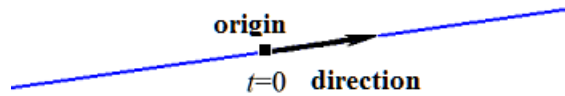


Рис. О.4.2.1.

В методе **PointOn**(double & t, [MbCartPoint3D](#) & r) радиус-вектор прямой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{origin} + t \mathbf{direction}.$$

Прямая линия ведёт себя как бесконечный объект, хотя в своих данных имеет граничные значения параметра *t_{min}* и *t_{max}*. Заметим, что в отличие от других кривых в методах вычисления радиуса-вектора и его производных прямая не корректирует параметр *t* при его выходе за предельные значения *t_{min}* и *t_{max}*.

О.4.3. Отрезок прямой MbLineSegment3D

Класс MbLineSegment3D объявлен в файле cur_line_segment_3d.h.

Отрезок прямой MbLineSegment3D описывается начальной точкой [MbCartPoint3D](#) **point1** и конечной точкой [MbCartPoint3D](#) **point2**, рис. О.4.3.1.



Рис. О.4.3.1.

В методе **PointOn**(double & t, [MbCartPoint3D](#) & r) радиус-вектор отрезка **r** описывается векторной функцией

$$\mathbf{r}(t) = (1 - t) \mathbf{point1} + t \mathbf{point2}.$$

Область определения параметра отрезка располагается в пределах от нуля до единицы. Начальной точке отрезка **point1** соответствует параметр *t_{min}*=0, конечной точке отрезка **point2** соответствует параметр *t_{max}*=1.

О.4.4. Дуга эллипса MbArc3D

Класс MbArc3D объявлен в файле cur_arc_3d.h.

Дуга эллипса является наследником кривой [MbCurve3D](#). Дуга эллипса MbArc3D описывается двумя радиусами a и b и двумя углами $trim1$ и $trim2$, заданными в локальной системе координат [MbPlacement3D position](#).

Углы $trim1$ и $trim2$ отсчитываются по дуге в направлении движения от вектора **position.axisX** к вектору **position.axisY**. Углы $trim1$ и $trim2$ будем называть параметрами усечения. Значения параметров усечения, равные нулю и 2π , соответствуют точке на координатной оси **position.axisX**. Параметр кривой t принимает значения на отрезке: $0 \leq t \leq trim2 - trim1$. Кривая может быть периодической. У периодической кривой $trim2 - trim1 = 2\pi$.

В методе **PointOn**(double & t , [MbCartPoint3D](#) & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = \text{position.origin} + a \cos(trim1+t) \text{position.axisX} + b \sin(trim1+t) \text{position.axisY}.$$

Дуга эллипса приведена на рис. О.4.4.1.

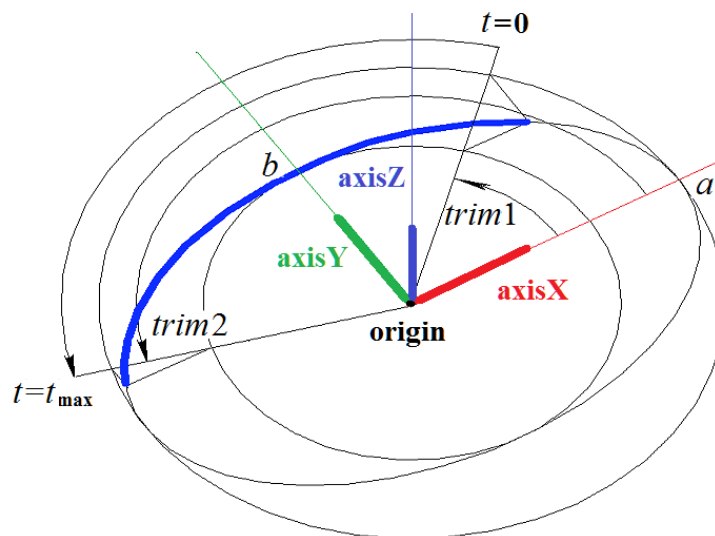


Рис. О.4.4.1.

Радиусы кривой должны быть больше нуля: $a > 0$, $b > 0$. Для параметров усечения должны соблюдаться неравенства: $trim1 < trim2$.

Локальная система координат **position** может быть как правой, так и левой.

О.4.5. Ломаная линия MbPolyline3D

Класс MbPolyline3D объявлен в файле cur_polyline3d.h.

Ломаная является наследником кривой MbPolyCurve3D. Ломаная MbPolyline3D описывается количеством сегментов *segmentsCount*, множеством контрольных точек `SArray<MbCartPoint3D>pointList` и признаком периодичности кривой *closed*.

Кривая проходит через множество точек **pointList**[i], $i=0, \dots, segmentsCount$. при значениях параметра $t=0, \dots, segmentsCount$. Если *closed*=true, то кривая содержит сегмент, соединяющий последнюю точку множества **pointList**[$segmentsCount-1$] с начальной точкой **pointList**[0]. Параметр кривой t принимает значения на отрезке: $0 \leq t \leq segmentsCount$.

В методе **PointOn**(double & t , [MbCartPoint3D](#) & r) радиус-вектор кривой r описывается векторной функцией

$$r(t) = \text{pointList}[i] (1-w) + \text{pointList}[i+1] w,$$

где $w = \frac{t - t_i}{t_{i+1} - t_i}$, а $t_i \leq t \leq t_{i+1}$. Ломаная является простейшей кривой, построенной по набору точек. Она состоит из отрезков, последовательно соединяющих контрольные точки.

Ломаная приведена на рис. О.4.5.1.

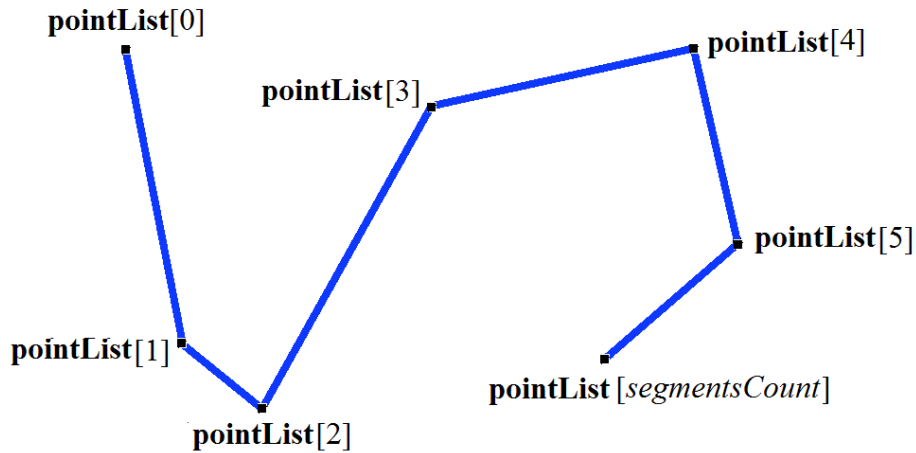


Рис. О.4.5.1.

Кривая может быть периодической. Период периодической кривой равен *segmentsCount*. Производные кривой в контрольных точках (при целочисленных значениях параметра) теряют непрерывность по длине и направлению. Производные кривой в контрольных точках имеют особое направление и длину. Ломаная обладает рядом полезных свойств: работа с ней требует минимум вычислений, проекция ломаной линии также будет ломаной линией.

О.4.6. NURBS-кривая MbNurbs3D

Класс MbNurbs3D объявлен в файле cur_nurbs3d.h.

NURBS-кривая получила название от первых букв сочетания слов NonUniform Rational B-Spline. Кривая является наследником кривой MbPolyCurve3D. Кривая описывается множеством контрольных точек `SArray<MbCartPoint3D>pointList`, множеством весов контрольных точек *weights*, узловым вектором *knots*, порядком сплайна *degree*, параметром формы кривой *form* и признаком периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривая построена на основе В-сплайнов. Узловой вектор *knots* представляет собой неубывающую последовательность действительных чисел и определяет область определения параметра кривой и форму кривой. Параметр формы кривой *form* в общем случае принимает значение `ncf_Unspecified`, в частных случаях он хранит информацию об исходной кривой, с которой была получена NURBS-копия. Порядок *degree* NURBS-кривой равен порядку разделённых разностей, по которым вычисляются В-сплайны. Пусть узловой вектор содержит *knotsCount* элементов, а множество контрольных точек содержит *pointsCount* элементов. Для не периодической NURBS-кривой количества элементов в множествах связаны равенством $knotsCount = pointsCount + degree$. Для периодической NURBS-кривой количество элементов в множествах связаны равенством $knotsCount = pointsCount + 2degree - 1$.

В методе `PointOn(double & t, MbCartPoint3D & r)` радиус-вектор кривой *r* описывается векторной функцией

$$\mathbf{r}(t) = \frac{\sum_{j=0}^{pointsCount-1} N_j^{degree}(t) weight[j] \mathbf{pointList}[j]}{\sum_{j=0}^{pointsCount-1} N_j^{degree}(t) weight[j]},$$

где $N_j^{degree}(t)$ – B-сплайны порядка $degree$ для j -ой контрольной точки **pointList**[j].
 NURBS-кривая приведена на рис. О.4.6.1.

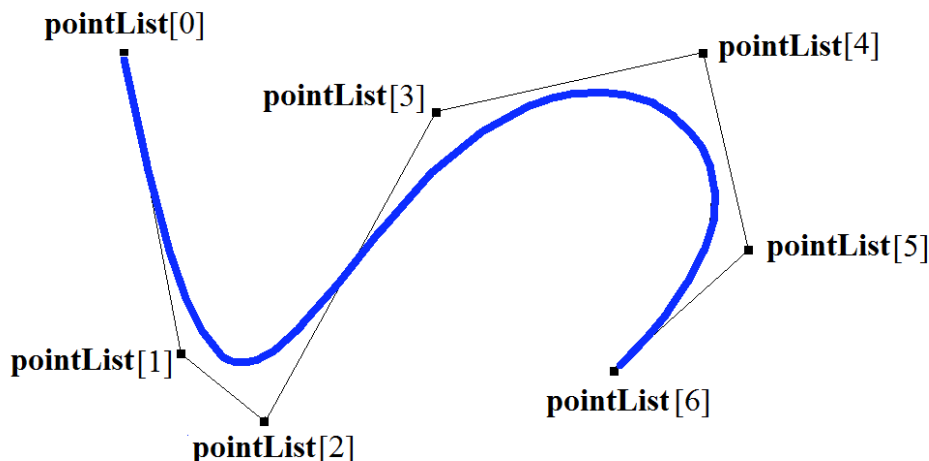


Рис. О.4.6.1.

Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$, где $tmin = knots[degree-1]$, $tmax = knots[knotsCount-degree]$. Кривая может быть периодической. Периодическая NURBS-кривая приведена на рис. О.4.6.2.

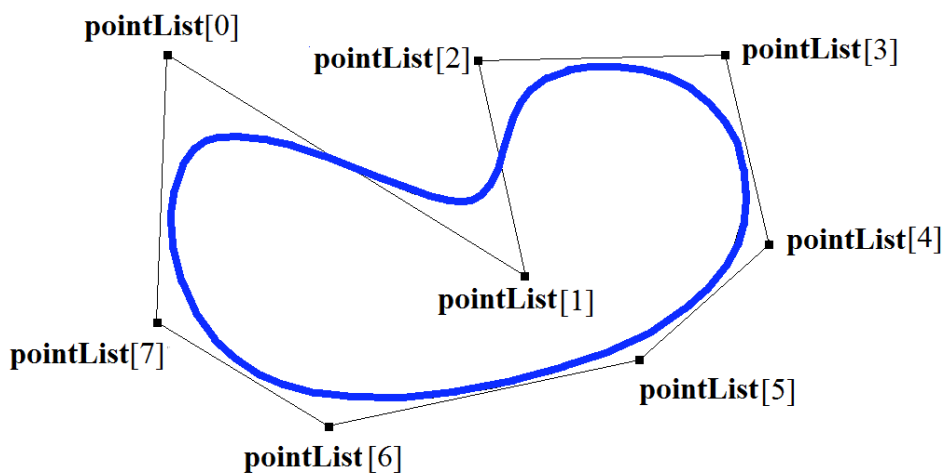


Рис. О.4.6.2.

Форма NURBS-кривой зависит от расположения контрольных точек, от веса контрольных точек и от значений узлового вектора. NURBS-кривая в общем случае не проходит через множество точек **pointList**[i], $i=0, \dots, pointsCount-1$. Чтобы незамкнутая NURBS-кривая проходила через крайние контрольные точки, требуется чтобы первые $degree$ элементов и последние $degree$ элементов узлового вектора $knots$ совпадали. Чем больше вес контрольной точки, тем ближе к этой точке проходит кривая при прочих равных условиях.

Каждая кривая может построить свою NURBS-копию виртуальным методом **NurbsCurve**(const MbNurbsParameters & tParameters).

О.4.7. Кривая Эрмита MbHermit3D

Класс MbHermit3D объявлен в файле cur_hermit3d.h.

Кривая Эрмита является наследником кривой MbPolyCurve3D. Кривая описывается множеством контрольных точек **SArray**<MbCartPoint3D>**pointList**, множеством производных кривой в контрольных точках **SArray**<MbVector3D>**vectorList**, множеством значений параметра кривой в контрольных точках **tList**, количеством *splinesCount* кубических сплайнов Эрмита и признаком

периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривая Эрмита при значении параметра $tList[i]$, $i=0,1,\dots,splinesCount$, проходит через контрольную точку **pointList**[i] и имеет в ней производную **vectorList**[i]. Кривая построена на основе $splinesCount$ сплайнов Эрмита третьей степени, которые гладко стыкуются между собой. Каждый кубический сплайн Эрмита описывает участок кривой между двумя соседними контрольными точками. Каждый кубический сплайн Эрмита определяется двумя крайними точками и двумя производными кривой в этих точках.

При вычислении радиуса-вектора точки кривой Эрмита сначала по значению параметра t кривой определяется номер i рабочего участка (номер кубического сплайна Эрмита), из условия $tList[i] \leq t \leq tList[i+1]$. Радиус-вектор кривой вычисляется как радиус-вектор найденного участка для его локального параметра w , который определяется по $tList[i]$ и $tList[i+1]$.

В методе **PointOn**(double & t , **MbCartPoint3D** & **r**) радиус-вектор кривой **r** описывается векторной функцией найденного участка для его локального параметра w :

$$\mathbf{r}(t) = (1 - 3w^2 + 2w^3)\mathbf{pointList}[i] + (3w^2 + 2w^3)\mathbf{pointList}[i + 1] + ((w - 2w^2 + w^3)\mathbf{vectorList}[i] + (-w^2 + w^3)\mathbf{vectorList}[i + 1])(tList[i + 1] - tList[i]),$$

где $w = \frac{t - tList[i]}{tList[i + 1] - tList[i]}$, а $tList[i] \leq t \leq tList[i + 1]$.

Кривая Эрмита приведена на рис. О.4.7.1.

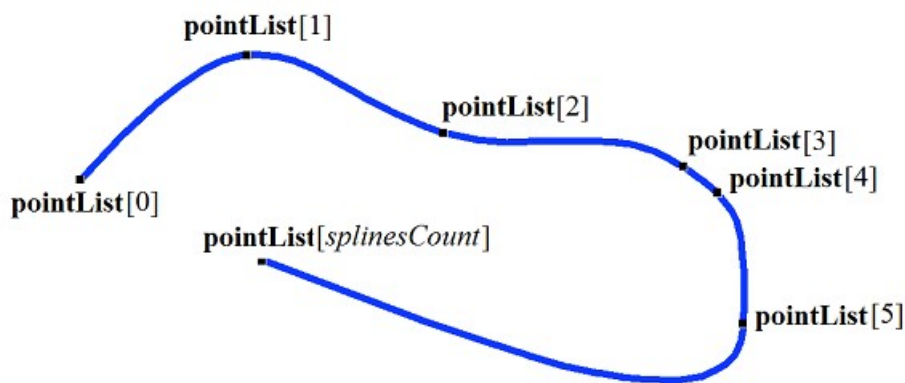


Рис. О.4.7.1.

Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$, где $tmin = tList[0]$, $tmax = tList[splinesCount]$. Кривая может быть периодической.

Форма кривой зависит от расположения контрольных точек, от производных кривой в контрольных точках и от множества $tList$ значений параметра в контрольных точках. При построении кривой только по контрольным точкам значения параметра кривой в контрольных точках $tList[i]$, $i=0,1,\dots,splinesCount$, изменяются пропорционально расстоянию между точками, а производные **vectorList**[i], $i=1,2,\dots,splinesCount-1$ вычисляются путём построения параболы, проходящей по трём соседним точкам **pointList**[$i-1$], **pointList**[i], **pointList**[$i+1$] при соответствующих значениях параметра $tList[i-1]$, $tList[i]$, $tList[i+1]$, и вычисления производной параболы в средней точке.

О.4.8. Составная кривая Безье MbBezier3D

Класс MbBezier3D объявлен в файле cur_bezier3d.h.

Составная кривая Безье является наследником кривой MbPolyCurve3D. Кривая описывается множеством контрольных точек **SArray**<**MbCartPoint3D**>**pointList**, количеством $splinesCount$ кривых Безье и признаком периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривая построена на основе $splinesCount$ кривых Безье третьей степени, которые гладко стыкуются между собой. Каждая кривая Безье определяется четырьмя контрольными точками и проходит только

через две крайние точки. Составная кривая используется для построения сплайна, проходящего через заданные точки. Заданные точки служат точками сочленения кривых Безье третьей степени. Пара внутренних контрольных точек каждой кривой Безье третьей степени определяется из условия гладкой стыковки кривой с соседними кривыми. Для составной кривой количество контрольных точек равно $3(splinesCount+1)$. У непериодической составной кривой первая **pointList[0]** и последняя контрольные точки не используются.

На каждой кривой Безье третьей степени параметр составной кривой увеличивается на единицу. При вычислении радиуса-вектора точки составной кривой сначала по значению параметра t кривой определяется номер рабочего участка (номер кривой Безье третьей степени), который равен максимальному целому числу, не превосходящему t . Пусть номер кривой Безье третьей степени равен n . Далее определяется дробная часть параметра $w=t-n$. Радиус-вектор составной кривой вычисляется как радиус-вектор найденного участка для его локального параметра w .

В методе **PointOn**(double & t , **MbCartPoint3D** & **r**) радиус-вектор кривой **r** описывается векторной функцией найденного участка для его локального параметра w :

$$\mathbf{r}(t) = \sum_{j=0}^3 \frac{3!}{j!(3-j)!} w^j (1-w)^{3-j} \mathbf{pointList}[3n+j],$$

где $w=t-n$, $n \leq t < n+1$, $0 \leq w \leq 1$, $B_j^3(w) = \frac{3!}{j!(3-j)!} w^j (1-w)^{3-j}$ – функции Бернштейна третьей степени для j -ой, $j=0,1,2,3$, контрольной точки **pointList[3n+j]** найденного участка с номером n .

Составная кривая Безье приведена на рис. О.4.8.1.

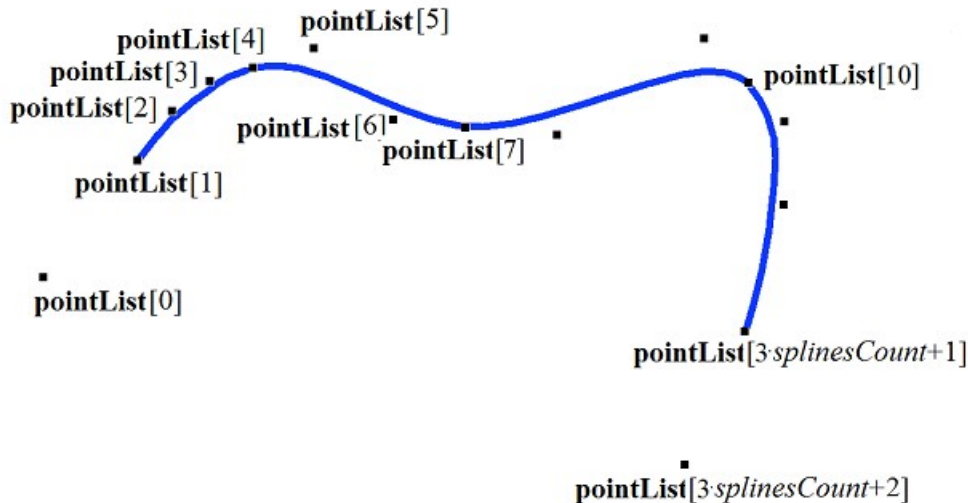


Рис. О.4.8.1.

Параметр кривой t принимает значения на отрезке: $0 \leq t \leq splinesCount$. Кривая может быть периодической. Период периодической кривой равен $splinesCount$.

При целочисленных значениях параметра кривая проходит через контрольные точки, например, при параметре $t=n$ кривая пройдет через контрольную точку **pointList[3n]**, $n=0,1,\dots,splinesCount$. Производные кривой в точках сочленения кривых Безье третьей степени (при целочисленных значениях параметра) теряют непрерывность по длине.

О.4.9. Кубический сплайн MbCubicSpline3D

Класс MbCubicSpline3D объявлен в файле cur_cubic_spline3d.h.

Кубический сплайн является наследником кривой MbPolyCurve3D. Кривая описывается множеством контрольных точек **SArray<MbCartPoint3D>pointList**, множеством вторых производных кривой в контрольных точках **SArray<MbVector3D>vectorList**, множеством значений параметра кривой в контрольных точках **tList**, максимальным значением индекса множества параметров

splinesCount и признаком периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кубический сплайн при значении параметра $tList[i]$, $i=0,1,\dots,splinesCount$, проходит через контрольную точку **pointList**[*i*] и имеет в ней вторую производную **vectorList**[*i*]. Кривая построена так, что при переходе из точки **pointList**[*i*] в точку **pointList**[*i*+1] вторая производная радиуса вектора кривой изменяется линейно от **vectorList**[*i*] до **vectorList**[*i*+1].

При вычислении радиуса-вектора точки составной кривой сначала по значению параметра *t* кривой определяется номер *i* рабочего участка, из условия $tList[i] \leq t \leq tList[i+1]$. Радиус-вектор кривой вычисляется по значениям **pointList**[*i*], **pointList**[*i*+1], **vectorList**[*i*], **vectorList**[*i*+1] найденного участка для локального параметра *w*, который определяется по $tList[i]$ и $tList[i+1]$.

В методе **PointOn**(double & *t*, **MbCartPoint3D** & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = (1-w)\mathbf{pointList}[i] + w\mathbf{pointList}[i+1] + \\ + \left((-2w+3w^2-w^3)\mathbf{vectorList}[i] + (-w+w^3)\mathbf{vectorList}[i+1] \right) \frac{(tList[i+1]-tList[i])^2}{6},$$

где $w = \frac{t - tList[i]}{tList[i+1] - tList[i]}$, а $tList[i] \leq t \leq tList[i+1]$.

Кубический сплайн, построенный по тем же контрольным точкам, что и составная кривая Эрмита, приведён на рис. О.4.9.1.

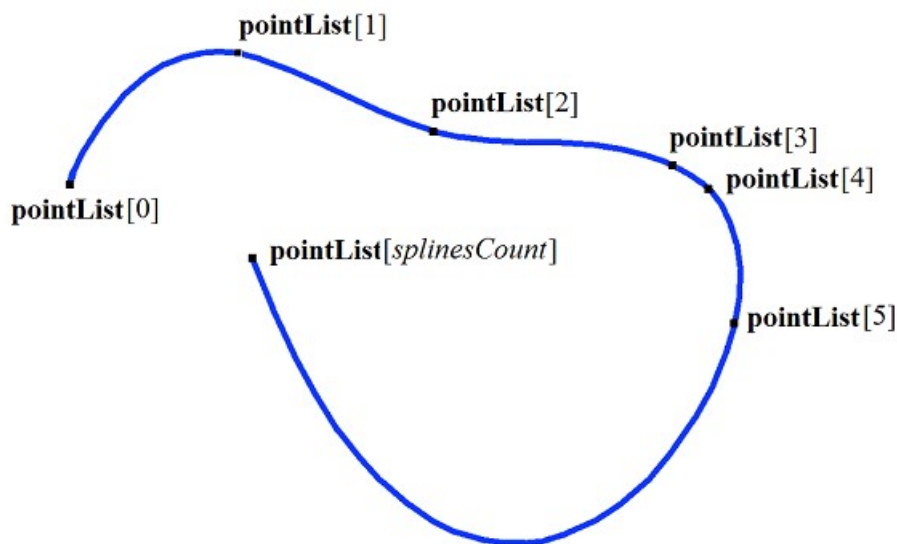


Рис. О.4.9.1.

Параметр кривой *t* принимает значения на отрезке: $tmin \leq t \leq tmax$, где $tmin=tList[0]$, $tmax=tList[splinesCount]$. Кривая может быть периодической.

Форма кривой зависит от расположения контрольных точек и от множества *tList* значений параметра в контрольных точках. При построении кривой только по контрольным точкам значения параметра кривой в контрольных точках $tList[i]$, $i=0,1,\dots,splinesCount$, изменяются пропорционально расстоянию между точками, а вторые производные **vectorList**[*i*], $i=1,2,\dots,splinesCount-1$ вычисляются путём решения системы уравнений.

О.4.10. Усечённая кривая **MbTrimmedCurve3D**

Класс **MbTrimmedCurve3D** объявлен в файле `cur_trimmed_curve3d.h`.

Усечённая кривая описывается базовой кривой **MbCurve3D*** **basisCurve**, начальным параметром усечения базовой кривой *trim1*, конечным параметром усечения базовой кривой *trim2* и признаком совпадения направлений базовой кривой и усечённой кривой *sense*.

Усечённая кривая совпадает с базовой кривой на участке, определённом параметрами $trim1$ и $trim2$, но может иметь противоположное с ним направление. Если $sense=1$, то $trim1 < trim2$ и усечённая кривая совпадает по направлению с базовой кривой. Если $sense=-1$, то $trim2 < trim1$ и усечённая кривая направлена против базовой кривой.

В методе **PointOn**(double & t , **MbCartPoint3D** & \mathbf{r}) радиус-вектор кривой \mathbf{r} описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{basisCurve}(trim1 + sense \cdot t).$$

Усеченная кривая приведена на рис. О.4.10.1.

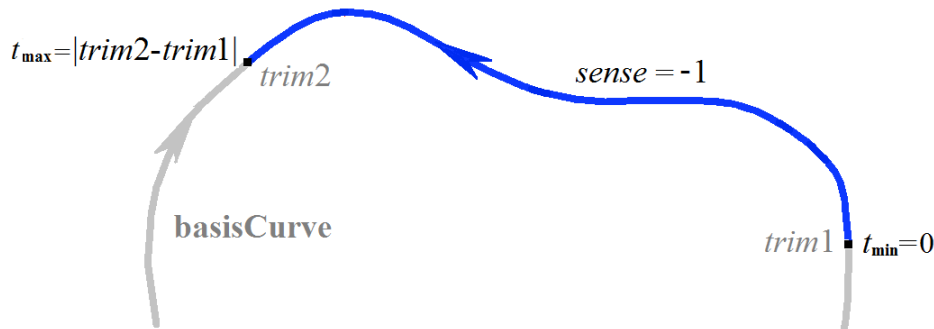


Рис. О.4.10.1.

Параметр кривой t принимает значения на отрезке: $0 \leq t \leq sense(trim2 - trim1)$.

Усечённая кривая теоретически может применяться для изменения направления кривой, но лучше пользоваться методом **Inverse**() .

Усечённая кривая может применяться для изменения положения начальной точки периодической кривой. Для этого базовая кривая должна быть периодической и $trim2 = trim1 + period$. В этом случае усечённая кривая так же будет периодической.

В качестве базовой кривой для усеченной кривой не должна использоваться другая усеченная кривая, а должна использоваться базовая кривая последней с соответствующим пересчетом параметров усечения.

Каждая кривая может построить свою усеченную копию виртуальным методом **Trimmed**(double $t1$, double $t2$, int $sense$).

О.4.11. Репараметризованная кривая MbReparamCurve3D

Класс MbReparamCurve3D объявлен в файле cur_reparam_curve3d.h.

Репараметризованная кривая описывается базовой кривой **MbCurve3D*** **basisCurve**, начальным параметром $tmin$, конечным параметром $tmax$ и производной dt параметра базовой кривой по параметру репараметризованной кривой.

Репараметризованная кривая полностью совпадает с базовой кривой, но имеет другую область изменения параметра.

В методе **PointOn**(double & t , **MbCartPoint3D** & \mathbf{r}) радиус-вектор кривой \mathbf{r} описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{basisCurve}(v(t)),$$

где $v(t) = bmin \frac{trim2 - t}{trim2 - trim1} + bmax \frac{t - trim1}{trim2 - trim1}$, $bmin$, $bmax$ – граничные значения области определения параметра базовой кривой.

Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$.

Репараметризованная кривая совпадает с базовой кривой, но имеет другую область определения параметра. Кривая с измененной длиной параметра применяется для согласования областей изменения параметра двух кривых. Например, если требуется, чтобы отрезок и дуга имели

одинаковые области изменения параметра, то на базе одной из указанных кривых создаётся репараметризованная кривая с областью изменения параметра, взятой у другой кривой.

В качестве базовой кривой для репараметризованной кривой не должна использоваться другая репараметризованная кривая, а должна использоваться базовая кривая последней.

О.4.12. Эквидистантная кривая MbOffsetCurve3D

Класс MbOffsetCurve3D объявлен в файле cur_offset_curve3d.h.

Эквидистантная кривая описывается базовой кривой [MbCurve3D](#)* **basisCurve** и вектором смещения [MbVector3D](#) **offset**. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Базовая кривая **basisCurve**, представляет собой объект MbSpine, который для кривой строит перемещающуюся по ней локальную систему координат, первая координатная ось которой направлена по касательной к кривой. Вектор смещения **offset** определяет смещение начальной точки базовой кривой в начальную точку эквидистантной кривой. Вектор смещения **offset** ортогонален касательному вектору базовой кривой в начальной точке. В движущейся локальной системе координат смещение всех точек базовой кривой в соответствующую точку эквидистантной кривой равно вектору смещения и ортогонально касательному вектору базовой кривой в текущей точке.

Вычисление радиуса-вектора точки эквидистантной кривой выполняется следующим образом. Для заданного параметра базовой кривой вычисляется точка на направляющей кривой и локальная система координат с началом в этой точке и первой координатной осью, направленной по касательной к кривой в этой точке. Далее вычисляется матрица поворота локальной системы координат при её перемещении из начальной точки базовой кривой в заданную точку. По матрице поворота трансформируется копия вектора смещения и на вычисленный вектор смещается вычисленная точка базовой кривой.

В методе [PointOn](#)(double & t, [MbCartPoint3D](#) & r) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{basisCurve}(t) + \mathbf{offset} \cdot \mathbf{A}(t),$$

где $\mathbf{A}(t)$ – матрица поворота локальной системы координат при её перемещении из начальной точки базовой кривой в заданную точку.

Эквидистантная кривая к конической спирали приведена на рис. О.4.12.1.

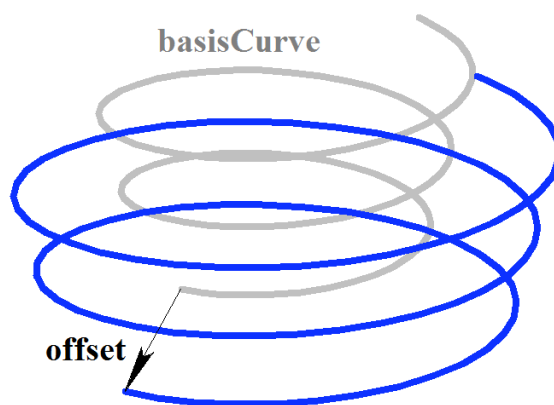


Рис. О.4.12.1.

Область изменения параметра эквидистантной кривой совпадает с областью изменения параметра базовой кривой.

В качестве базовой кривой для эквидистантной кривой не должна использоваться другая эквидистантная кривая, а должна использоваться направляющая кривая последней с соответствующим пересчетом вектора смещения.

О.4.13. Символьная кривая MbCharacterCurve3D

Класс MbCharacterCurve3D объявлен в файле cur_character_curve3d.h.

Символьная кривая описывается координатными функциями $xFunction$, $yFunction$, $zFunction$, локальной системой координат [MbPlacement3D](#) **position** функций, матрицей трансформации **transform**, граничными значениями области определения параметра кривой $tmin$ и $tmax$, признаком периодичности кривой $closed$ и типом системы координат (декартова, цилиндрическая, сферическая) $coordinateType$, в которой заданы координатные функции. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Координатные функции $xFunction(t)$, $yFunction(t)$, $zFunction(t)$ символьной кривой представляют собой скалярные функции общего параметра t и заданы в виде символьных выражений. Для каждого символьного выражения выполнен лексический анализ и построено дерево, которое вычисляет значение символьного выражения для заданного параметра и производные символьного выражения по параметру. Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$.

В методе **PointOn**(double & t, [MbCartPoint3D](#) & r) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = [xFunction(t) \ yFunction(t) \ zFunction(t)].$$

Символьная кривая приведена на рис. О.4.13.1.

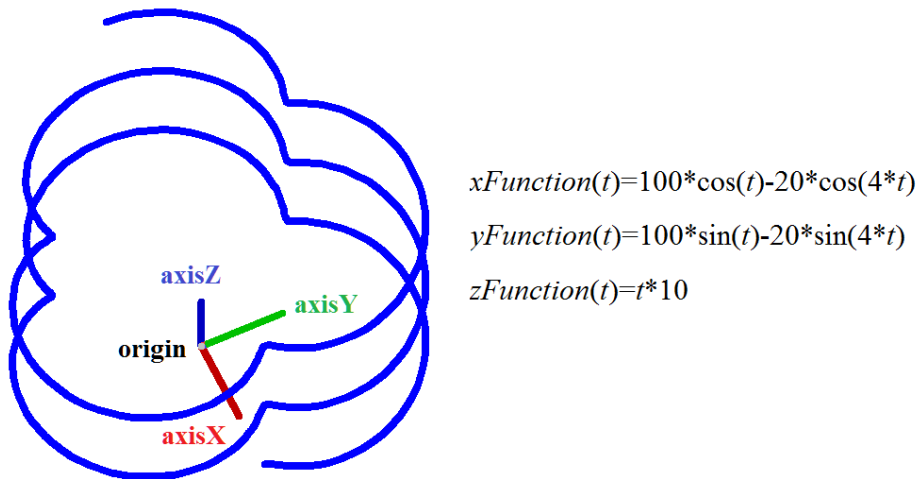


Рис. О.4.13.1.

Кривая может быть периодической. Символьные выражения на области определения кривой должны описывать непрерывные конечные и однозначные функции.

О.4.14. Коническая спираль MbConeSpiral

Класс MbConeSpiral объявлен в файле cur_cone_spiral.h.

Коническая спираль является наследником кривой MbSpiral. Спираль описывается локальной системой координат [MbPlacement3D](#) **position**, радиусом $radius$, тангенсом угла конусности $tgAlpha$, шагом спирали, делённым на 2π , $step_2pi$ и двумя граничными параметрами спирали $tmin$ и $tmax$. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Ось спирали совпадает с координатной осью **position.axisZ** локальной системы координат. Параметр $tgAlpha$ равен тангенсу угла между осью спирали и образующей конуса спирали. Параметры $tmin$ и $tmax$ являются углами и отсчитываются от вектора **position.axisX** к вектору **position.axisY**. Значения углов, кратные 2π , соответствуют точке кривой в плоскости XZ локальной системы координат. Параметр кривой t принимает значения на отрезке: $tmin \leq t \leq tmax$.

В методе **PointOn**(double & t, [MbCartPoint3D](#) & r) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{position.origin} + (radius + t \operatorname{tg}Alpha \operatorname{step_}2\pi) (\cos(t) \mathbf{position.axisX} + \sin(t) \mathbf{position.axisY}) + ((t \operatorname{step_}2\pi) \mathbf{position.axisZ}).$$

Коническая спираль приведена на рис. О.4.14.1.

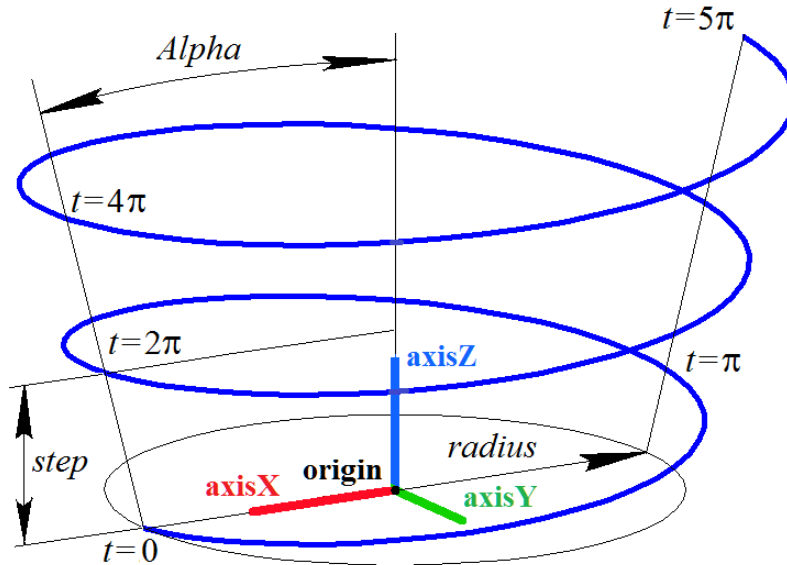


Рис. О.4.14.1.

Радиус кривой должен быть больше нуля: $radius > 0$. Для граничных параметров должны соблюдаться неравенства: $tmin < tmax$. Кривая не может быть периодической.

Локальная система координат **position** может быть как правой, так и левой. Для цилиндрической спирали $\operatorname{tg}Alpha = 0$.

О.4.15. Спираль переменного радиуса MbCurveSpiral

Класс MbCurveSpiral объявлен в файле cur_curve_spiral.h.

Спираль переменного радиуса является наследником кривой MbSpiral. Спираль описывается локальной системой координат **MbPlacement3D position**, двумерной кривой **curve**, задающей закон изменения радиуса, шагом спирали **step**, двумя граничными параметрами спирали **tmin** и **tmax**. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Ось спирали совпадает с координатной осью **position.axisZ** локальной системы координат. Двумерная кривая **curve** располагается в плоскости XZ локальной системы координат и определяет закон изменения радиуса спирали. Ось **position.axisZ** служит осью абсцисс, а ось **position.axisX** служит осью ординат двумерного пространства кривой **curve**. Начало двумерной системы координат кривой **curve** совпадает с началом локальной системы координат **position**. Радиус спирали равен ординате точек двумерной кривой **curve**. Параметры **tmin** и **tmax** являются углами и отсчитываются от вектора **position.axisX** к вектору **position.axisY**. Значения углов, кратные 2π , соответствуют точке кривой в плоскости XZ локальной системы координат.

В методе **PointOn**(double & t, **MbCartPoint3D** & r) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{position.origin} + radius(t) (\cos(t) \mathbf{position.axisX} + \sin(t) \mathbf{position.axisY}) + ((t \operatorname{step} / 2\pi) \mathbf{position.axisZ}),$$

где $radius(t)$ – локальный радиус. Локальный радиус $radius(t)$ вычисляется следующим образом. По заданному параметру t определяем абсциссу $t \operatorname{step} / 2\pi$ искомой двумерной точки кривой. Далее

определяем точку пересечения кривой *curve* и вертикальной прямой, пересекающей ось абсцисс в точке $tstep/2\pi$. Ордината двумерной точки пересечения *curve* и вертикальной прямой равна искомому радиусу спирали $radius(t)$. Локальный радиус равен расстоянию от локальной оси абсцисс до точки пересечения вертикальной прямой с кривой *curve* в двумерном пространстве плоскости XZ локальной системы координат **position**.

Спираль переменного радиуса приведена на рис. O.4.15.1.

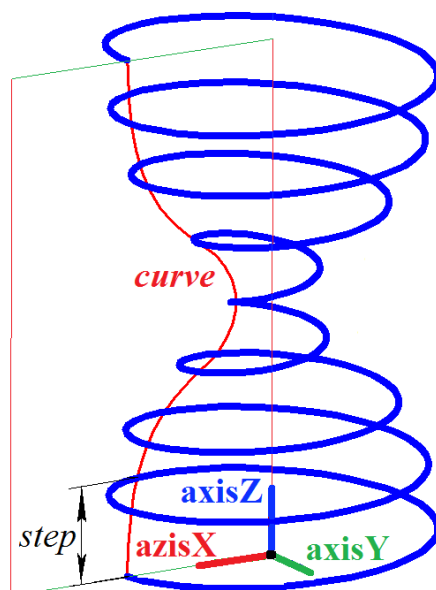


Рис. O.4.15.1.

Кривая *curve* должна располагаться выше оси абсцисс и не должна пересекать ось абсцисс своей двумерной системы координат. Кривая *curve* не должна иметь вертикальных касательных. Для граничных параметров должны соблюдаться неравенства: $tmin < tmax$. Кривая не может быть периодической.

Локальная система координат **position** может быть как правой, так и левой.

O.4.16. Спираль с криволинейной плоской осью MbCrookedSpiral

Класс MbCrookedSpiral объявлен в файле cur_crooked_spiral.h.

Спираль с криволинейной плоской осью является наследником кривой MbSpiral. Спираль описывается локальной системой координат **MbPlacement3D position**, двумерной кривой **MbCurve* curve**, определяющей ось спирали, радиусом спирали *radius*, шагом спирали *step*, двумя граничными параметрами спирали *tmin* и *tmax*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Двумерная кривая *curve* располагается в плоскости XZ локальной системы координат **position** и определяет ось спирали. Ось **position.axisZ** служит осью абсцисс, а ось **position.axisX** служит осью ординат двумерного пространства кривой *curve*. Начало двумерной системы координат кривой *curve* совпадает с началом локальной системы координат **position**. Радиус спирали постоянен. Параметры *tmin* и *tmax* являются углами и отсчитываются от вектора **position.axisX** к вектору **position.axisY**. Значения углов, кратные 2π , соответствуют точке кривой в плоскости XZ локальной системы координат.

В методе **PointOn**(double & t, **MbCartPoint3D** & r) радиус-вектор кривой **r** описывается векторной функцией

$$\begin{aligned} \mathbf{r}(t) = & \mathbf{position.origin} + \\ & ((\mathbf{point.y} + \mathbf{radius} \cos(t) \mathbf{normal.y}) \mathbf{position.axisX}) + \\ & (\mathbf{radius} \sin(t) \mathbf{position.axisY}) + \\ & ((\mathbf{point.x} + \mathbf{radius} \cos(t) \mathbf{normal.x}) \mathbf{position.axisZ}), \end{aligned}$$

где *point* – точка двумерной кривой, которая вычисляется методом *curve*→**PointOn**(*t,point*), *normal* – нормаль двумерной кривой, которая вычисляется методом *curve*→**Normal**(*t,normal*).

Спираль переменного радиуса приведена на рис. 0.4.16.1.

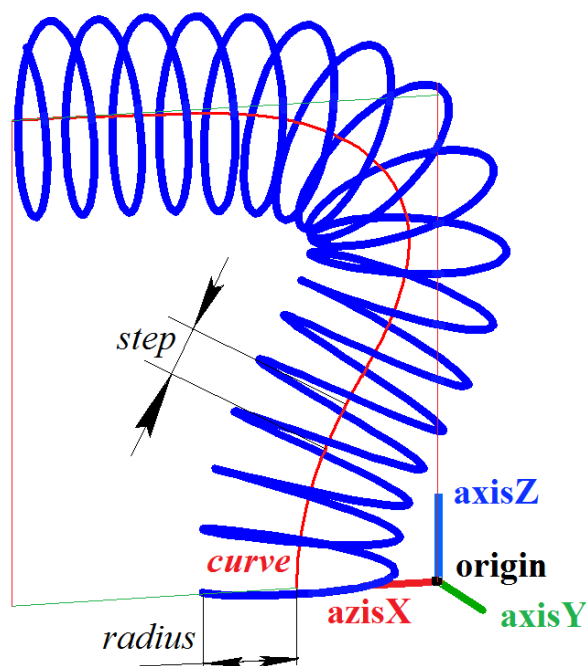


Рис. 0.4.16.1.

Минимальный радиус кривизны кривой *curve* не должен быть меньше радиуса спирали. Для граничных параметров должны соблюдаться неравенства: $t_{min} < t_{max}$. Кривая не может быть периодической.

Локальная система координат **position** может быть как правой, так и левой.

0.4.17. Соединительная кривая MbBridgeCurve3D

Класс MbBridgeCurve3D объявлен в файле cur_bridge3d.h.

Соединительная кривая является наследником кривой MbCurve3D. Кривая описывается двумя кривыми MbCurve3D* **curve1** и MbCurve3D* **curve2**, параметрами точек этих кривых *param1* и *param2*, признаками *sense1* и *sense2* совпадения направлений производных соединительной кривой и соединяемых кривых, двумя граничными параметрами соединительной кривой *tmin* и *tmax*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Соединительная кривая служит для плавного соединения двух указанных точек кривых **curve1** и **curve2**. Точки кривых **curve1** и **curve2** заданы параметрами *param1* и *param2*. Направление соединительной кривой в этих точках определяют параметры *sense1* и *sense2*. Соединительная кривая представляет собой кубический сплайн Эрмита, построенный по двум крайним точкам и производным кривой в этих точках. Параметр кривой *t* принимает значения на отрезке: $t_{min} \leq t \leq t_{max}$.

В методе **PointOn**(double & *t*, MbCartPoint3D & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = (1 - 3w^2 + 2w^3)\mathbf{point1} + (3w^2 + 2w^3)\mathbf{point2} + ((w - 2w^2 + w^3)\mathbf{derive1} + (-w^2 + w^3)\mathbf{derive2})(t_{max} - t_{min}),$$

где $w = \frac{t - t_{min}}{t_{max} - t_{min}}$ – относительное значение параметра, **point1** – точка кривой **curve1**, которая вычисляется методом **curve1**→**PointOn**(*param1,point1*), **point2** – точка кривой **curve2**, которая вычисляется методом **curve2**→**PointOn**(*param2,point2*), **derive1** и **derive2** – производные

соединительной кривой в крайних точках. Векторы **derive1** и **derive2** направлены параллельно производным соединяемых кривых. Длина векторов **derive1** и **derive2** равна расстоянию между крайними точками, делённому на $t_{max}-t_{min}$. Соединительная кривая приведена на рис. О.4.17.1.

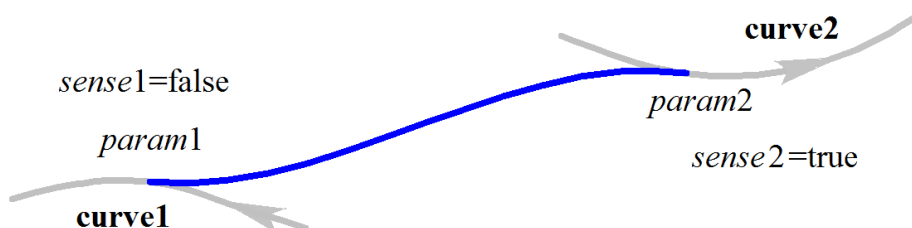


Рис. О.4.17.1.

Форма кривой зависит от расположения крайних точек и направлений соединяемых кривых в этих точках. Для граничных параметров должны соблюдаться неравенства: $t_{min} < t_{max}$. Кривая не может быть периодической.

О.4.18. Контур MbContour3D

Класс MbContour3D объявлен в файле cur_contour3d.h.

Контур MbContour3D описывается множеством RPArray<MbCurve3D>segments стыкующихся друг с другом кривых и признаком периодичности кривой *closed*.

Контур представляет собой составную кривую. В отличие от других кривых контур может иметь изломы. Кривые, образующие контур, будем называть сегментами. Для сегментов контура выполняются следующие условия: начало каждого последующего сегмента совпадает с концом предыдущего сегмента. Для периодического контура начало первого сегмента совпадает с концом последнего сегмента. В общем случае в местах стыковки сегментов производные контура терпят разрыв по длине и направлению.

Начальное значение параметра контура равно нулю: $t_{min}=0$. Параметрическая длина контура равна сумме параметрических длин составляющих его сегментов: $t_{max} = \sum (w_{i\max} - w_{i\min})$, где $w_{i\min}$ и $w_{i\max}$ – минимальное и максимальное значение параметра i -го сегмента. При вычислении радиуса-вектора точки контура сначала по значению параметра определяется рабочий сегмент и значение его локального параметра, далее вычисляется радиус-вектор рабочего сегмента, который служит радиусом-вектором контура.

В методе **PointOn**(double & t , MbCartPoint3D & r) радиус-вектор кривой r описывается векторной функцией

$$\mathbf{r}(t) = \text{segments}[k](w_k),$$

где $\text{segments}[k](w_k)$ – рабочий сегмент контура с индексом k , w_k – параметр рабочего сегмента, равный: $w_k = w_{k\min} + t - \sum_{i=0}^{k-1} (w_{i\max} - w_{i\min})$.

Сегмент с индексом k определяется по значению параметра контура t из условия $\sum_{i=0}^{k-1} (w_{i\max} - w_{i\min}) \leq t < \sum_{i=0}^k (w_{i\max} - w_{i\min})$, где $w_{i\min}$ и $w_{i\max}$ – минимальное и максимальное значение параметра i -го сегмента. Контур приведён на рис. О.4.18.1.

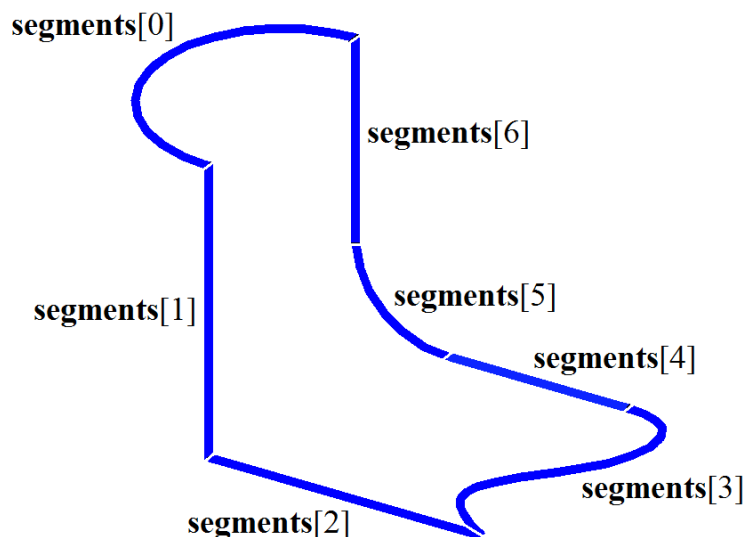


Рис. О.4.18.1.

В качестве сегментов контура не должны использоваться другие контуры. Если контур нужно построить на основе других контуров, то последние должны рассматриваться как совокупность составляющих их кривых, а не как единые кривые.

Контур MbContour3D представляет собой наиболее общий вид кривой.

О.4.19. Плоская кривая MbPlaneCurve

Класс MbPlaneCurve объявлен в файле cur_plane_curve.h.

Плоская кривая MbPlaneCurve описывается локальной системой координат [MbPlacement3D](#) **position** и двумерной кривой [MbCurve](#)* **curve** в плоскости XY локальной системы координат.

Плоская кривая представляет собой отображение кривой двумерного пространства плоскости XY локальной системы координат в трёхмерное пространство.

В методе [PointOn](#)(double & t, [MbCartPoint3D](#) & r) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{position.origin} + (\mathit{point.x} \mathbf{position.axisX}) + (\mathit{point.y} \mathbf{position.axisY}),$$

где **point** – точка двумерной кривой, которая вычисляется методом **curve**→[PointOn](#)(t,**point**).

Плоская кривая в составе двумерного эллипса и локальной системы координат приведена на рис. О.4.19.1.

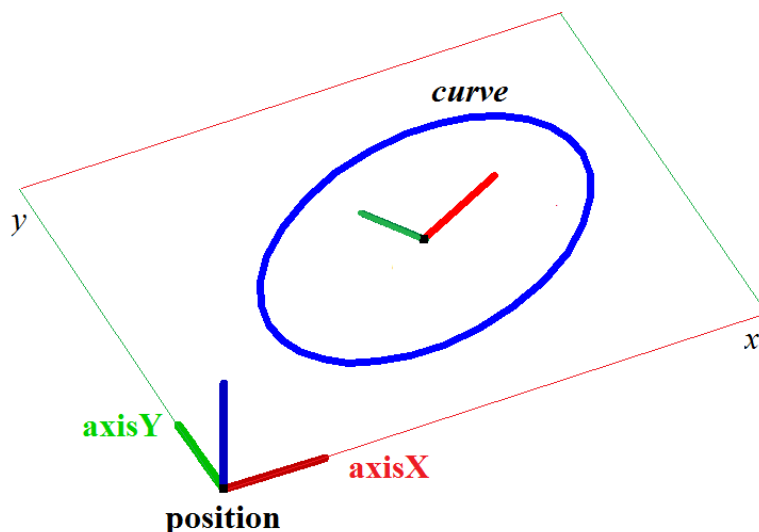


Рис. О.4.19.1.

Область определения параметров и периодичность плоской кривой совпадают с областью определения параметров и периодичностью двумерной кривой *curve*.

О.4.20. Кривая на поверхности MbSurfaceCurve

Класс MbSurfaceCurve объявлен в файле cur_surface_curve.h.

Кривая на поверхности MbSurfaceCurve описывается поверхностью [MbSurface](#)* *surface*, двумерной кривой [MbCurve](#)* *curve* в пространстве параметров поверхности и признаком периодичности кривой *closed*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривая на поверхности представляет собой отображение кривой двумерного пространства параметров поверхности в трёхмерное пространство. Двумерная кривая *curve* может располагаться за пределами области определения параметров поверхности. Область определения параметров кривой на поверхности совпадает с областью определения параметров двумерной кривой *curve*.

В методе [PointOn](#)(double & t, [MbCartPoint3D](#) & r) радиус-вектор кривой *r* описывается векторной функцией

$$\mathbf{r}(t) = \text{surface}(\text{point}.x, \text{point}.y),$$

где *point* – точка двумерной кривой, которая вычисляется методом *curve*→[PointOn](#)(t,*point*). Координаты *x* и *y* двумерной точки *point* служат параметрами *u* и *v* поверхности *surface*(*u,v*). Кривая на поверхности построена путём введения зависимости параметров поверхности *u* и *v* от некоторого общего для них параметра *t*: *u*=*u*(*t*), *v*=*v*(*t*). Эту зависимость описывает двумерная кривая *curve*.

Кривая на поверхности приведена на рис. О.4.20.1.

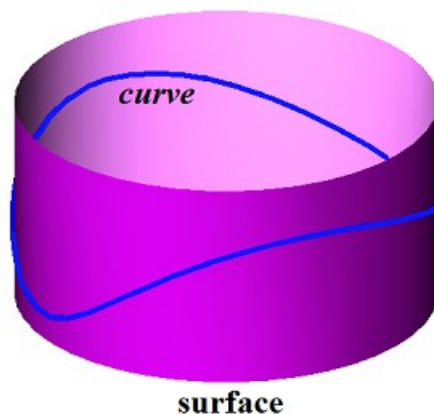


Рис. О.4.20.1.

Двумерная кривая в области определения параметров поверхности приведена на рис. О.4.20.2.

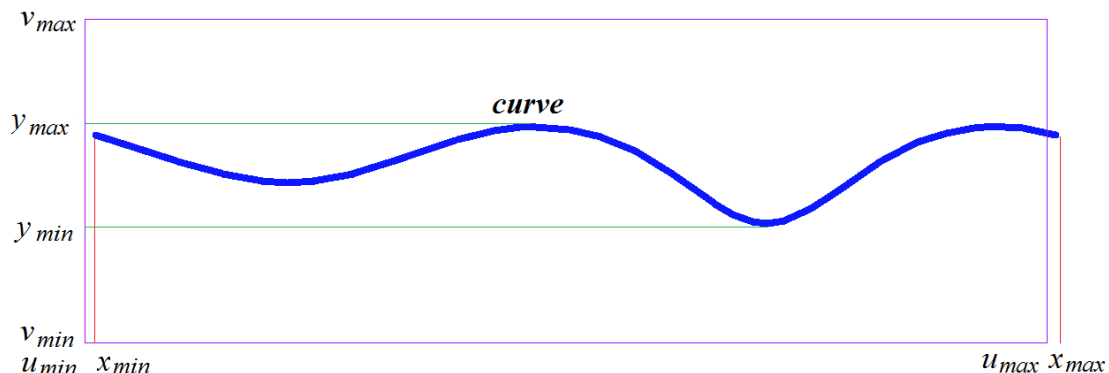


Рис. О.4.20.2.

Производная кривой на поверхности вычисляется как сложная функция

$$\frac{dr(t)}{dt} = \frac{\partial \text{surface}(u,v)}{\partial u} \text{derive}.x + \frac{\partial \text{surface}(u,v)}{\partial v} \text{derive}.y,$$

где *derive* – производная двумерной кривой, которая вычисляется методом *curve*→**FirstDer**(*t,derive*). Производная кривой на поверхности лежит в касательной плоскости поверхности, построенной в рассматриваемой точке.

Кривая на поверхности может быть периодической, если периодической является двумерная кривая *curve* или если периодической является поверхность *surface*, а кривая *curve* имеет совпадающие производные на краях и крайние точки кривой смещены на соответствующий период периодической по первому или второму параметру поверхности *surface*.

Поверхностью *surface* кривой на поверхности может служить любая поверхность, кроме поверхности, ограниченной кривыми [MbCurveBoundedSurface](#). Если требуется построить кривую на поверхности, ограниченной кривыми, то поверхностью будет служить базовая поверхность [MbCurveBoundedSurface](#).

О.4.21. Силуэтная кривая MbSilhouetteCurve

Класс MbSilhouetteCurve объявлен в файле cur_silhouette_curve.h.

Силуэтная кривая поверхности MbSilhouetteCurve является наследником кривой на поверхности [MbSurfaceCurve](#). Силуэтная кривая описывается поверхностью [MbSurface*](#) *surface*, двумерной кривой [MbCurve*](#) *curve* в пространстве параметров поверхности, признаком периодичности кривой *closed*, признаком перспективы *perspective*, вектором взгляда *eye* и типом кривой *species*. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Силуэтная кривая представляет собой кривую на поверхности *surface*, разделяющую видимую из точки наблюдения часть этой поверхности и невидимую из точки наблюдения часть этой поверхности. Если *perspective=true*, то точку наблюдения описывает вектор взгляда *eye*. Если *perspective=false*, то точка наблюдения находится на бесконечном расстоянии, а вектор взгляда *eye* описывает направление из точки наблюдения к поверхности. Нормаль поверхности *surface* на силуэтной кривой ортогональна прямой линии, соединяющей эту точку поверхности и точку наблюдения.

В частном случае, когда можно построить точную силуэтную кривую поверхности, тип кривой *species* принимает значение *cbt_Ordinary*. Например, для сферы силуэтной кривой служит окружность. В частном случае строится точная пространственная кривая *exactCurve*, которая точно описывает силуэт поверхности и используется для вычисления радиуса-вектора силуэтной кривой и его производных.

В общем случае тип кривой *species* принимает значение *cbt_Specific*, двумерная кривая *curve* является сплайном и аппроксимирует силуэт поверхности. В общем случае точка силуэтной кривой вычисляется итерационным методом, использующим двумерную кривую *curve* в качестве начального приближения.

Область определения параметров силуэтной кривой поверхности совпадает с областью определения параметров двумерной кривой *curve*.

В методе **PointOn**(double & t, [MbCartPoint3D](#) & r) радиус-вектор кривой *r* описывается векторной функцией

$$\mathbf{r}(t) = \text{surface}(u, v),$$

где *u, v* – координаты двумерной точки, начальное приближение которой вычисляется методом *curve*→**PointOn**(*t,point*), *u=point.x*, *v=point.y*. Далее параметры *u* и *v* уточняются итерационным методом, использующим уравнение

$$\mathbf{vector} \cdot \mathbf{n}(u,v) = 0,$$

где *n(u,v)* – нормаль поверхности, которая вычисляется методом *surface*→**Normal**(*u,v,n*), *vector* – вектор взгляда (для бесконечно удалённой точки наблюдения *vector=eye*).

Силуэтная кривая поверхности тора приведена на рис. О.4.21.1.

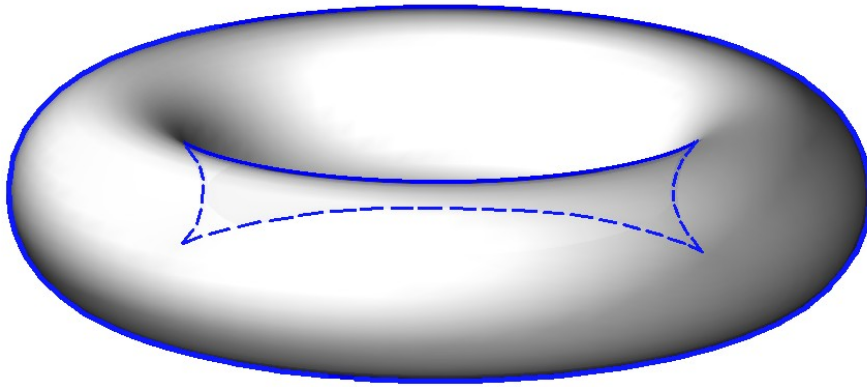


Рис. О.4.21.1.

Силуэтная кривая поверхности тора с другого направления приведена на рис. О.4.21.2.

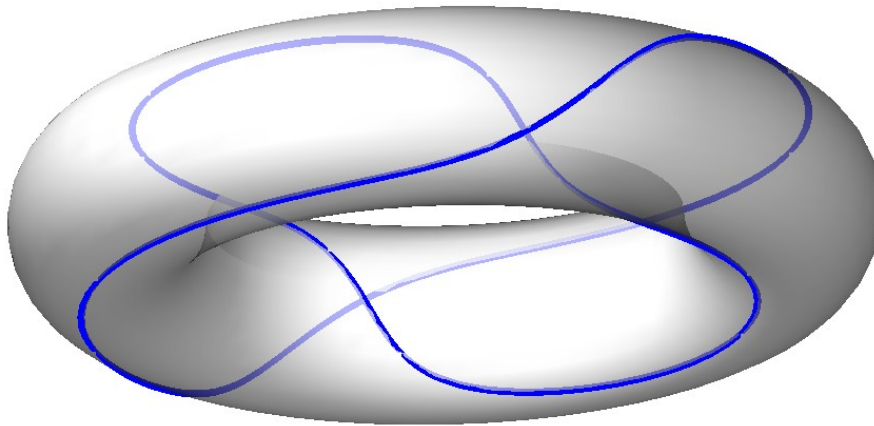


Рис. О.4.21.2.

При переходе через силуэтную кривую скалярное произведение вектора нормали поверхности и вектора взгляда меняет знак. Силуэтная кривая всегда или замкнута, или начинается и оканчивается на краях поверхности. Силуэтная кривая используется для построения проекций силуэта криволинейной поверхности на плоскость.

О.4.22. Контур на поверхности MbContourOnSurface

Класс MbContourOnSurface объявлен в файле cur_contour_on_surface.h.

Контур на поверхности MbContourOnSurface описывается поверхностью [MbSurface](#)* **surface** и двумерным контуром [MbContour](#)* **contour** в пространстве параметров поверхности. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Контур на поверхности представляет собой составную кривую, поэтому может иметь изломы в точках стыковки сегментов двумерного контура. Контур на поверхности представляет собой отображение контура двумерного пространства параметров поверхности в трёхмерное пространство. Двумерный контур **contour** может располагаться за пределами области определения параметров поверхности. Область определения параметров контура на поверхности совпадает с областью определения параметров двумерного контура **contour**.

В методе [PointOn](#)(double & t, [MbCartPoint3D](#) & r) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \text{surface}(\text{point.x}, \text{point.y}),$$

где *point* – точка двумерного контура, которая вычисляется методом *contour*→**PointOn**(*t,point*). Координаты *x* и *y* двумерной точки *point* служат параметрами *u* и *v* поверхности **surface**(*u,v*). Контур на поверхности построен путём введения зависимости параметров поверхности *u* и *v* от некоторого общего для них параметра *t*: *u*=*u*(*t*), *v*=*v*(*t*). Эту зависимость описывает двумерный контур *contour*. Производная контура на поверхности вычисляется как сложная функция

$$\frac{d\mathbf{r}(t)}{dt} = \frac{\partial \mathbf{surface}(u,v)}{\partial u} \mathit{derive}.x + \frac{\partial \mathbf{surface}(u,v)}{\partial v} \mathit{derive}.y,$$

где *derive* – производная двумерного контура, которая вычисляется методом *contour*→**FirstDer**(*t,derive*). Производная контура на поверхности лежит в касательной плоскости поверхности, построенной в рассматриваемой точке.

Контур на поверхности приведен на рис. О.4.22.1.

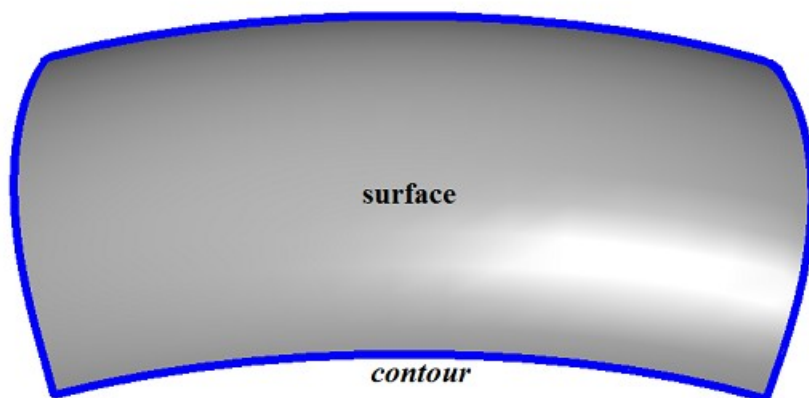


Рис. О.4.22.1.

Контур на поверхности может быть периодическим, если периодическим является двумерный контур *contour* или если периодической является поверхность **surface**, а крайние точки контура *contour* смещены на соответствующий период периодической по первому или второму параметру поверхности **surface**.

Периодический контур на поверхности обычно используется для описания границы этой поверхности.

Поверхностью **surface** контура на поверхности может служить любая поверхность, кроме поверхности, ограниченной кривыми **MbCurveBoundedSurface**. Если требуется построить контур на поверхности, ограниченной кривыми, то поверхностью будет служить базовая поверхность **MbCurveBoundedSurface**.

О.4.23. Контур на плоскости MbContourOnPlane

Класс MbContourOnPlane объявлен в файле cur_contour_on_plane.h.

Контур на плоскости MbContourOnPlane является наследником класса **MbContourOnSurface**. Контур на плоскости описывается плоскостью **MbSurface*** **surface** и двумерным контуром **MbContour*** *contour* в пространстве параметров плоскости. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Контур на плоскости представляет собой составную кривую, поэтому может иметь изломы в точках стыковки сегментов двумерного контура. Контур на плоскости представляет собой отображение контура двумерного пространства параметров плоскости в трёхмерное пространство. Двумерный контур *contour* может располагаться за пределами области определения параметров плоскости. Область определения параметров контура на плоскости совпадает с областью определения параметров двумерного контура *contour*.

В методе `PointOn(double & t, MbCartPoint3D & r)` радиус-вектор кривой `r` описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{position.origin} + (\mathit{point.x} \mathbf{position.axisX}) + (\mathit{point.y} \mathbf{position.axisY}),$$

где `position` – локальная система координат плоскости `surface`, `point` – точка двумерного контура, которая вычисляется методом `contour->PointOn(t,point)`.

Контур на плоскости приведен на рис. О.4.23.1.

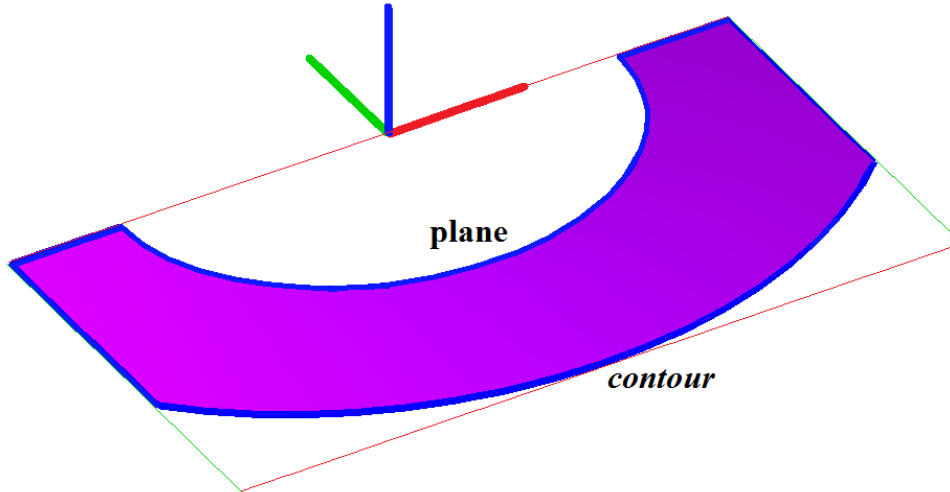


Рис. О.4.23.1.

Контур на плоскости может быть периодическим, если периодическим является двумерный контур `contour` или если периодической является плоскость `surface`, а крайние точки контура `contour` смещены на соответствующий период периодической по первому или второму параметру плоскости `surface`.

Периодический контур на плоскости обычно используется для описания границы этой плоскости.

Контур на плоскости аналогичен контуру на поверхности, но обладает более высокой скоростью вычислений.

О.4.24. Кривая пересечения поверхностей `MbSurfaceIntersectionCurve`

Класс `MbSurfaceIntersectionCurve` объявлен в файле `cur_surface_intersection.h`.

Кривая пересечения поверхностей `MbSurfaceIntersectionCurve` описывается двумя кривыми `MbSurfaceCurve curveOne` и `MbSurfaceCurve curveTwo` на пересекающихся поверхностях, параметром построения `buildType` и точностью `tolerance`. У кривой есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов кривой.

Кривые `curveOne(t)` и `curveTwo(t)` имеют одинаковые области определения параметра `t` и совпадают в пространстве с некоторой известной точностью. Параметр `buildType` кривой пересечения описывает тип кривой и несет информацию о способе вычисления радиуса-вектора точки кривой. Параметр `buildType` принимает значения: `cbt_Specific`, `cbt_Ordinary`, `cbt_Boundary`, `cbt_Tolerant`. На рис. О.4.24.1 приведены две поверхности и кривая их пересечения.

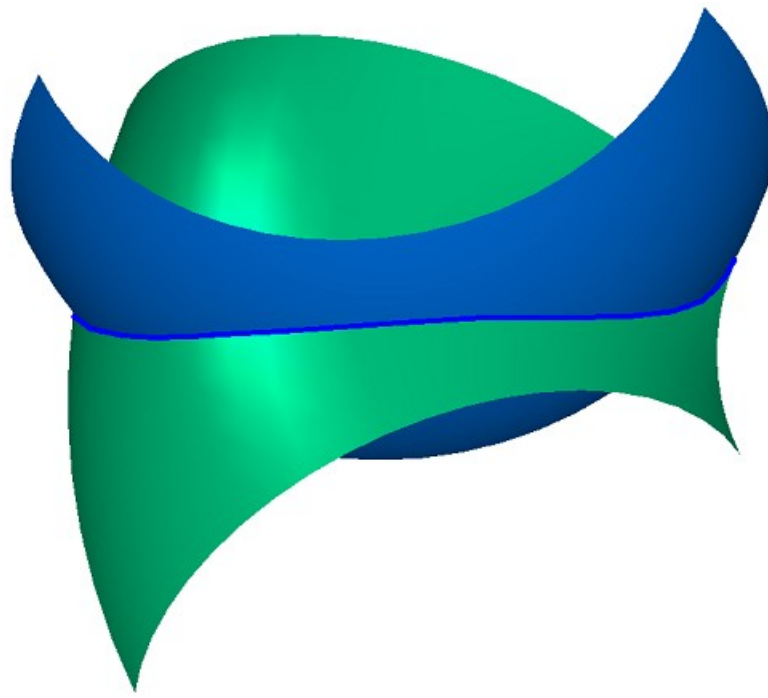


Рис. О.4.24.1.

На рис. О.4.24.2 и О.4.24.3 приведены кривые на поверхностях, из которых строится кривая пересечения.

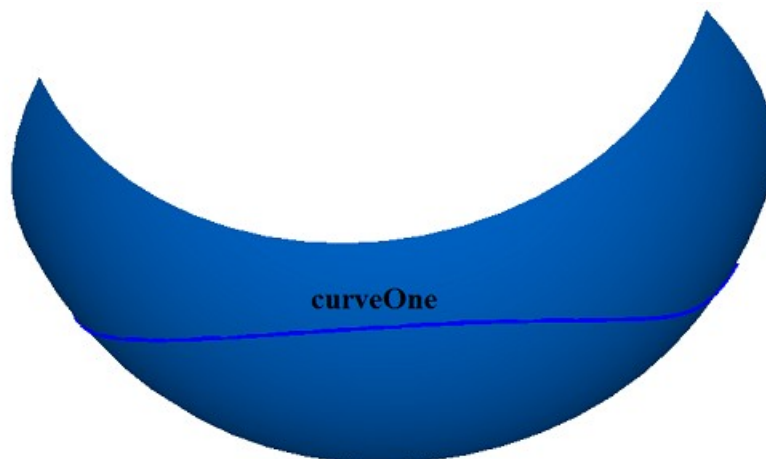


Рис. О.4.24.2.

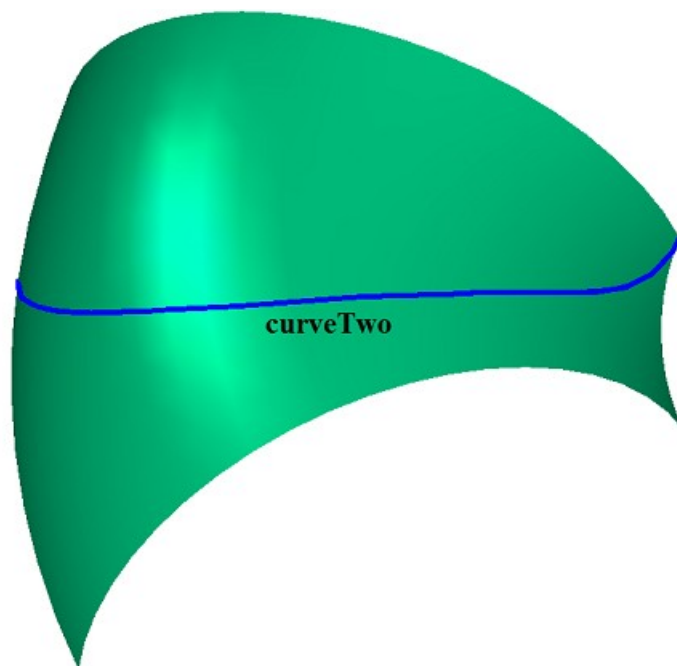


Рис. О.4.24.3.

В общем случае кривая пересечения имеет тип `cbt_Specific`, двумерные кривые **curveOne.curve** и **curveTwo.curve** являются сплайнами и аппроксимируют пересечение поверхностей **curveOne.surface** и **curveTwo.surface**. Сплайны на поверхностях имеют согласованные контрольные точки. В контрольных точках сплайны на поверхностях **curveOne** и **curveTwo** совпадают и имеют одинаковые значения параметров. На участках между контрольными точками сплайнов кривая `MbSurfaceIntersectionCurve` также выдает точное значение радиуса-вектора точки. В общем случае точка кривой пересечения поверхностей вычисляется итерационным методом, использующим двумерные кривые **curveOne.curve** и **curveTwo.curve** в качестве начального приближения.

В частных случаях кривая пересечения имеет типы `cbt_Ordinary`, `cbt_Boundary`, `cbt_Tolerant`, а точка кривой пересечения поверхностей вычисляется как среднее арифметическое радиусов-векторов кривых **curveOne(t)** и **curveTwo(t)**.

Если `buildType=cbt_Ordinary`, то кривая `MbSurfaceIntersectionCurve` точно описывает пересечение поверхностей, а кривые **curveOne(t)** и **curveTwo(t)** совпадают в пространстве. Примером такой кривой может служить кривая пересечения плоскости и цилиндрической поверхности, ось которой ортогональна плоскости, рис. О.4.24.4.

buildType = `cbt_Ordinary`

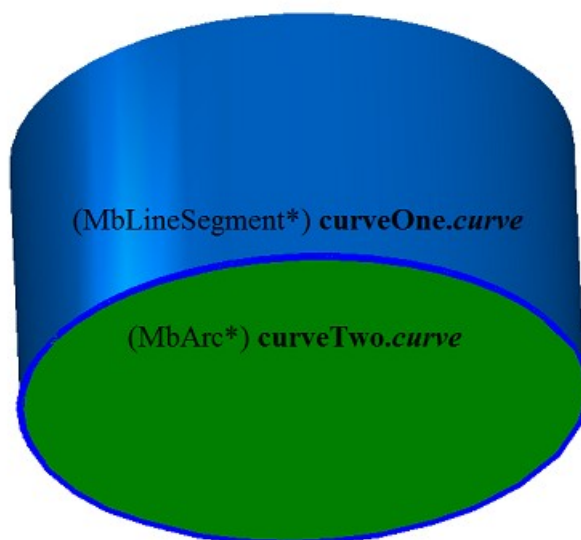


Рис. О.4.24.4.

На плоскости **curveOne.curve** является окружностью, а на цилиндрической поверхности **curveTwo.curve** является отрезком с параметрической длиной, равной параметрической длине двумерной кривой на плоскости. Равенство параметрических длин достигается построением, на базе отрезка репараметризованной кривой [MbReparamCurve](#).

Если *buildType*=cbt_Boundary, то кривая MbSurfaceIntersectionCurve описывает край поверхности, рис. О.4.24.5.. Для такой кривой выполняются равенства **curveOne.curve=curveTwo.curve** и **curveOne.surface=curveTwo.surface**.

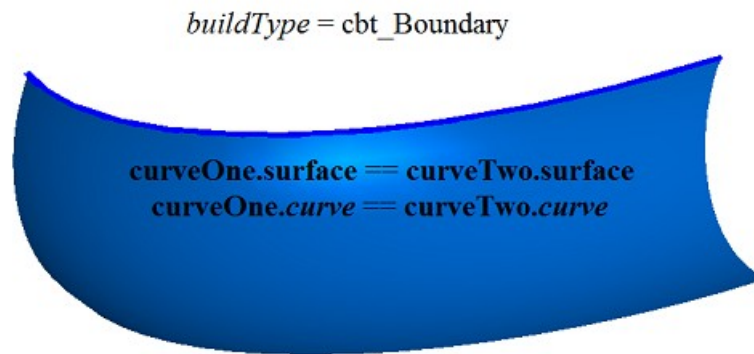


Рис. О.4.24.5.

Если *buildType*=cbt_Tolerant, то кривая MbSurfaceIntersectionCurve описывает пересечение поверхностей приближённо. Кривые **curveOne(t)** и **curveTwo(t)** совпадают в пространстве с точностью *tolerance*. Такие кривые строятся в тех случаях, когда другое построение невозможно, например, при необходимости пересечь две поверхности, касающиеся друг друга не точно, а с некоторым «шумом».

В методе **PointOn**(double & *t*, [MbMatrix3D](#) & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = 0.5 (\mathbf{curveOne.surface}(u_1, v_1) + \mathbf{curveTwo.surface}(u_2, v_2)),$$

где u_1, v_1 – координаты двумерной точки, начальное приближение которой вычисляется методом **curveOne.curve**→**PointOn**(*t*, *point1*), $u_1=point1.x, v_1=point1.y$, u_2, v_2 – координаты двумерной точки, начальное приближение которой вычисляется методом **curveTwo.curve**→**PointOn**(*t*, *point2*), $u_2=point2.x, v_2=point2.y$. Далее в общем случае (*buildType*=cbt_Specific) параметры u_1, v_1, u_2, v_2 уточняются итерационным методом, использующим уравнения

$$\begin{aligned} \mathbf{curveOne.surface}(u_1, v_1) &= \mathbf{plane}(x, y), \\ \mathbf{curveTwo.surface}(u_2, v_2) &= \mathbf{plane}(x, y), \end{aligned}$$

где **plane** – плоскость, перпендикулярная отрезку, соединяющему две ближайшие контрольные точки кривой пересечения. Кривая пересечения в общем случае и контрольные точки, по которым построены кривые **curveOne** и **curveTwo**, приведены на рис. О.4.24.6.

buildType = cbt_Specific

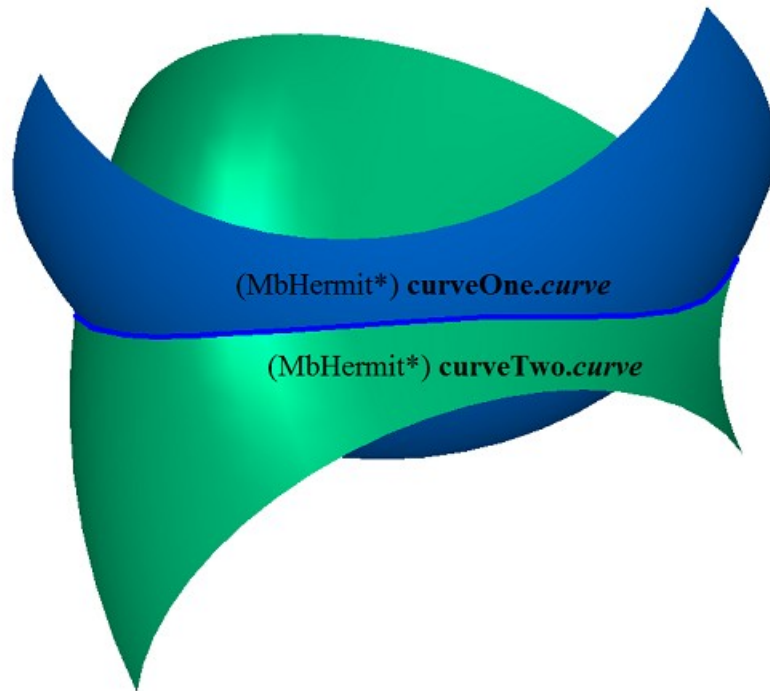


Рис. O.4.24.6.

Область определения параметра кривой пересечения совпадает с областью определения общего параметра кривых **curveOne** и **curveTwo**. Кривая пересечения поверхностей может быть периодической.

Кривая пересечения поверхностей среди своих данных содержит трёхмерную кривую **spaceCurve**, которая с точностью *tolerance* совпадает с кривой пересечения. Кривая **spaceCurve** используется для построения плоских проекций рёбер. Кривая **spaceCurve** является вспомогательным объектом и насчитывается только при необходимости.

0.5. ПОВЕРХНОСТИ

Поверхности являются представителями семейства трёхмерных геометрических объектов [MbSpaceItem](#). Поверхности играют главную роль в построении геометрической модели. Поверхностями описывают гладкие участки геометрической формы моделируемых объектов. Поверхности строятся с помощью аналитических функций, по набору точек, на базе кривых и на базе поверхностей. Векторы, радиусы-векторы точек, матрицы в трехмерном пространстве будем обозначать буквами латинского алфавита, выделенными **полужирным** шрифтом.

0.5.1. Поверхность MbSurface

Класс MbSurface объявлен в файле surface.h.

Поверхность MbSurface является наследником класса [MbSpaceItem](#), рис. 0.5.1.1.

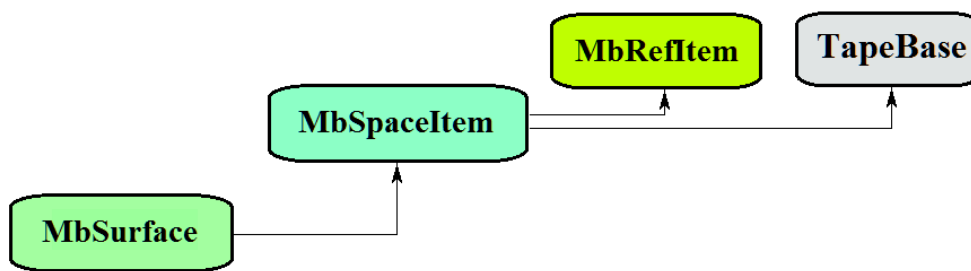


Рис. 0.5.1.1.

Поверхность является абстрактным классом. В геометрическом ядре C3D реализованы следующие поверхности, которые являются наследниками класса MbSurface:

[MbPlane](#) – плоскость,

[MbCylinderSurface](#) – цилиндрическая поверхность,

[MbConeSurface](#) – коническая поверхность,

[MbSphereSurface](#) – сферическая поверхность,

[MbTorusSurface](#) – поверхность тора,

[MbExtrusionSurface](#) – поверхность выдавливания,

[MbRevolutionSurface](#) – поверхность вращения,

[MbExpansionSurface](#) – плоскопараллельная кинематическая поверхность,

[MbSpiralSurface](#) – спиральная поверхность,

[MbEvolutionSurface](#) – кинематическая поверхность,

[MbExactionSurface](#) – кинематическая поверхность с адаптацией,

[MbSectorSurface](#) – секториальная поверхность,

[MbRuledSurface](#) – линейчатая поверхность,

[MbLoftedSurface](#) – поверхность на семействе кривых,

[MbElevationSurface](#) – поверхность на семействе кривых и направляющей,

[MbCornerSurface](#) – поверхность на трёх кривых,

[MbCoverSurface](#) – поверхность на четырёх кривых,

[MbCoonsPatchSurface](#) – бикубическая поверхность Кунса,

[MbMeshSurface](#) – поверхность на сети кривых,

[MbJoinSurface](#) – поверхность соединения,

[MbSplineSurface](#) – NURBS -поверхность (NonUniform Rational B-Spline поверхность),

[MbOffsetSurface](#) – эквидистантная поверхность,

[MbChamferSurface](#) – поверхность фаски,

[MbFilletSurface](#) – поверхность скругления,

[MbChannelSurface](#) – поверхность скругления с переменным радиусом,

[MbCurveBoundedSurface](#) – поверхность с произвольными границами.

Поверхность MbSurface представляет собой векторную функцию

$$\mathbf{surface}(u, v) = [x(u, v) \quad y(u, v) \quad z(u, v)]$$

двух скалярных параметров u и v , принимающих значения на двумерной связной области Ω . Поверхность представляет собой непрерывное отображение двумерной связной области Ω в трёхмерное пространство. Область Ω будем описывать в двумерной декартовой системе координат. В частном случае область Ω представляет собой прямоугольник, и параметры поверхности принимают значения в пределах $u_{\min} \leq u \leq u_{\max}$, $v_{\min} \leq v \leq v_{\max}$. В общем случае область Ω описывается двумерными кривыми. Координаты $x(u, v)$, $y(u, v)$, $z(u, v)$ точки поверхности $\mathbf{surface}(u, v)$ являются однозначными непрерывными функциями параметров u и v .

Граничные значения u_{\min} , u_{\max} , v_{\min} , v_{\max} области определения параметров выдают методы поверхности `double GetUMin()`, `double GetUMax()`, `double GetVMin()`, `double GetVMax()`, соответственно.

Поверхность будем называть периодической по первому параметру, если существует $p_u > 0$, такое, что $\mathbf{surface}(u \pm kp_u, v) = \mathbf{surface}(u, v)$, где k – целое число. Поверхность будем называть периодической по второму параметру, если существует $p_v > 0$, такое, что $\mathbf{surface}(u, v \pm kp_v) = \mathbf{surface}(u, v)$, где k – целое число. Область определения периодического параметра поверхности лежит в пределах одного периода для соответствующего параметра.

Метод `bool IsUClosed()` периодической по первому параметру поверхности возвращает `true`.

Метод `bool IsVClosed()` периодической по второму параметру поверхности возвращает `true`.

Метод `double GetUPeriod()` периодической по первому параметру поверхности или поверхности, которая может быть расширена до периодической, выдает период p_u . Метод `double GetVPeriod()` периодической по второму параметру поверхности или поверхности, которая может быть расширена до периодической, выдает период p_v . Область определения параметра периодической поверхности всегда лежит в пределах одного периода.

Введём обозначения

$$\begin{aligned} \mathbf{s}_u &= \frac{\partial \mathbf{surface}(u, v)}{\partial u}; \quad \mathbf{s}_v = \frac{\partial \mathbf{surface}(u, v)}{\partial v}; \\ \mathbf{s}_{uu} &= \frac{\partial^2 \mathbf{surface}(u, v)}{\partial u^2}; \quad \mathbf{s}_{vv} = \frac{\partial^2 \mathbf{surface}(u, v)}{\partial v^2}; \quad \mathbf{s}_{uv} = \frac{\partial^2 \mathbf{surface}(u, v)}{\partial u \partial v} = \frac{\partial^2 \mathbf{surface}(u, v)}{\partial v \partial u}; \\ \mathbf{s}_{uuu} &= \frac{\partial^3 \mathbf{surface}(u, v)}{\partial u^3}; \quad \mathbf{s}_{uuv} = \frac{\partial^3 \mathbf{surface}(u, v)}{\partial u \partial u \partial v}; \quad \mathbf{s}_{uvv} = \frac{\partial^3 \mathbf{surface}(u, v)}{\partial u \partial v \partial v}; \quad \mathbf{s}_{vvv} = \frac{\partial^3 \mathbf{surface}(u, v)}{\partial v^3}; \end{aligned}$$

для частных производных поверхности по её параметрам.

Основным методом поверхности является метод `void PointOn(double & u, double & v, MbCartPoint3D & s)`.

Он выдаёт радиус-вектор $\mathbf{s}(u, v)$ точки поверхности для заданных параметров u и v . Методы

`void DeriveU(double & u, double & v, MbVector3D & \mathbf{s}_u)`,

`void DeriveV(double & u, double & v, MbVector3D & \mathbf{s}_v)`,

`void DeriveUU(double & u, double & v, MbVector3D & \mathbf{s}_{uu})`,

`void DeriveUV(double & u, double & v, MbVector3D & \mathbf{s}_{uv})`,

`void DeriveVV(double & u, double & v, MbVector3D & \mathbf{s}_{vv})`,

`void DeriveUUU(double & u, double & v, MbVector3D & \mathbf{s}_{uuu})`,

`void DeriveUUV(double & u, double & v, MbVector3D & \mathbf{s}_{uuv})`,

`void DeriveUVV(double & u, double & v, MbVector3D & \mathbf{s}_{uvv})`,

`void DeriveVVV(double & u, double & v, MbVector3D & \mathbf{s}_{vvv})`

выдают соответственно производные \mathbf{s}_u , \mathbf{s}_v , \mathbf{s}_{uu} , \mathbf{s}_{uv} , \mathbf{s}_{vv} , \mathbf{s}_{uuu} , \mathbf{s}_{uuv} , \mathbf{s}_{uvv} , \mathbf{s}_{vvv} радиуса-вектора поверхности для заданных параметров u и v . Перечисленные методы корректируют параметры поверхности при их выходе за пределы области определения (исключение составляет плоскость [MbPlane](#)). При выходе параметра u за пределы отрезка $[u_{\min}, u_{\max}]$ непериодические по первому параметру поверхности смещают параметр u к ближайшей границе u_{\min} или u_{\max} , а периодические по первому параметру поверхности добавляют или вычитают необходимое количество периодов. При выходе параметра v за пределы отрезка $[v_{\min}, v_{\max}]$ непериодические по второму параметру поверхности смещают параметр v к ближайшей границе v_{\min} или v_{\max} , а периодические по второму параметру поверхности добавляют или вычитают необходимое количество периодов.

Метод

void **PointOn**(double *u*, double *v*, [MbCartPoint3D](#) & *s*)

выдаёт радиус-вектор $s(u,v)$ точки поверхности для заданных параметров *u* и *v* как в области определения параметров поверхности, так и за её пределами. Каждая непериодическая поверхность за пределами области определения параметров продолжается по своему закону. При отсутствии такого закона (в общем случае) непериодическая поверхность за пределами области определения параметров продолжается по касательной к соответствующей крайней точке поверхности. Методы

void **DeriveU**(double *u*, double *v*, [MbVector3D](#) & *s_u*),

void **DeriveV**(double *u*, double *v*, [MbVector3D](#) & *s_v*),

void **DeriveUU**(double *u*, double *v*, [MbVector3D](#) & *s_{uu}*),

void **DeriveUV**(double *u*, double *v*, [MbVector3D](#) & *s_{uv}*),

void **DeriveVV**(double *u*, double *v*, [MbVector3D](#) & *s_{vv}*),

void **DeriveUUU**(double *u*, double *v*, [MbVector3D](#) & *s_{uuu}*),

void **DeriveUUV**(double *u*, double *v*, [MbVector3D](#) & *s_{uuv}*),

void **DeriveUVV**(double *u*, double *v*, [MbVector3D](#) & *s_{uvv}*),

void **DeriveVVV**(double *u*, double *v*, [MbVector3D](#) & *s_{vvv}*)

выдают соответственно производные s_{u^2} , s_{v^2} , s_{uv} , s_{uu} , s_{uv} , s_{vv} , s_{uuu} , s_{uuv} , s_{uvv} , s_{vvv} радиуса-вектора поверхности для заданных параметров *u* и *v* как в области определения поверхности, так и за её пределами.

Поверхности перегружают такие методы трёхмерного геометрического объекта как:

методы, обслуживающие преобразование геометрического объекта,

void **Move**(const [MbVector3D](#) & *v*, MbRegTransform * *iReg* = NULL),

void **Rotate**(const MbAxis3D & *axis*, double *angle*, MbRegTransform * *iReg* = NULL),

void **Transform**(const [MbMatrix3D](#) & *m*, MbRegTransform * *iReg* = NULL),

методы, обеспечивающие копирование, проверку на совпадение, проверку на возможность сделать совпадающими, делающие объекты совпадающими,

[MbSpaceItem](#) & **Duplicate**(MbRegDuplicate * *iReg* = NULL),

bool **IsSame**(const [MbSpaceItem](#) & *item*),

bool **IsSimilar**(const [MbSpaceItem](#) & *item*),

bool **SetEqual**(const [MbSpaceItem](#) & *item*),

методы, возвращающие тип из перечисления геометрических объектов,

MbSpaceType **IsA**(),

MbSpaceType **Type**(),

MbSpaceType **Family**(),

методы, обеспечивающие выдачу и редактирование внутренних данных объекта,

MbProperty & **CreateProperty**(MbPrompt name),

void **GetProperties**(MbProperties & *properties*),

void **SetProperties**(MbProperties & *properties*),

метод, наполняющий полигональную копию геометрического объекта,

void **CalculateWire**(double sag, [MbMesh](#) & *mesh*).

Большинство поверхностей имеют прямоугольную область определения параметров. Среди всех поверхностей выделим [MbCurveBoundedSurface](#), которая является универсальной поверхностью. [MbCurveBoundedSurface](#) имеет криволинейные края и может иметь произвольные вырезы внутри. [MbCurveBoundedSurface](#) строится на основе любой поверхности с прямоугольной областью определения параметров.

0.5.2. Плоскость MbPlane

Класс MbPlane объявлен в файле surf_plane.h.

Плоскость MbPlane принадлежит к группе элементарных поверхностей MbElementarySurface. Плоскость описывается плоскостью XY локальной системе координат [MbPlacement3D](#) **position**. Первый параметр плоскости отсчитывается вдоль вектора **position.axisX**, второй параметр плоскости отсчитывается вдоль вектора **position.axisY**. Область определения плоскости описывают граничные значения параметров *umin*, *umax* и *vmin*, *vmax*, рис. 0.5.2.1.

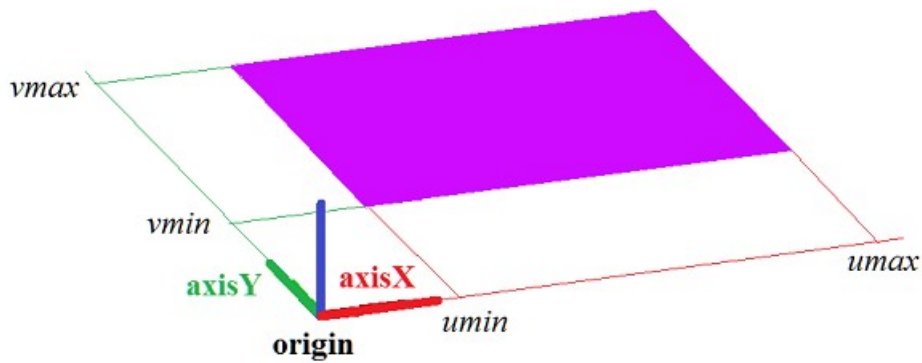


Рис. 0.5.2.1.

В методе **PointOn**(double *u*, double *v*, [MbCartPoint3D](#) & **s**) радиус-вектор плоскости **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{position.origin} + u \mathbf{position.axisX} + v \mathbf{position.axisY}.$$

Плоскость ведёт себя как бесконечный объект, хотя в своих данных имеет граничные значения параметров *umin*, *umax* и *vmin*, *vmax*. Заметим, что в отличие от других поверхностей в методах вычисления радиуса-вектора и его производных плоскость не корректирует параметры *u* и *v* при их выходе за пределы области определения, заданной значениями *umin*, *umax* и *vmin*, *vmax*.

Локальная система координат **position** может быть как правой, так и левой. Если локальная система координат левая, то нормаль плоскости направлена в сторону, противоположную направлению вектора **position.axisZ**.

0.5.3. Цилиндрическая поверхность MbCylinderSurface

Класс MbCylinderSurface объявлен в файле surf_cylinder_surface.h.

Цилиндрическая поверхность MbCylinderSurface принадлежит к группе элементарных поверхностей MbElementarySurface. Цилиндрическая поверхность описывается радиусом *radius* и высотой *height*, заданными в локальной системе координат [MbPlacement3D](#) **position**.

Первый параметр поверхности отсчитывается по дуге от вектора **position.axisX** в направлении вектора **position.axisY**. Первый параметр поверхности *u* принимает значения на отрезке: $umin \leq u \leq umax$. Значения $u=0$ и $u=2\pi$ соответствуют точке на плоскости XZ. Поверхность может быть периодической по первому параметру. У периодической поверхности $umax-umin=2\pi$, у не периодической поверхности $umax-umin < 2\pi$.

Второй параметр поверхности отсчитывается по прямой вдоль вектора **position.axisZ**. Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=0$ соответствует началу локальной системы координат, а значение $v=1$ соответствует точке на расстоянии *height* от плоскости XY локальной системы координат поверхности.

В методе **PointOn**(double *u*, double *v*, [MbCartPoint3D](#) & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{position.origin} + \mathit{radius} (\cos(u) \mathbf{position.axisX} + \sin(u) \mathbf{position.axisY}) + \mathit{height} v \mathbf{position.axisZ}.$$

Цилиндрическая поверхность приведена на рис. 0.5.3.1.

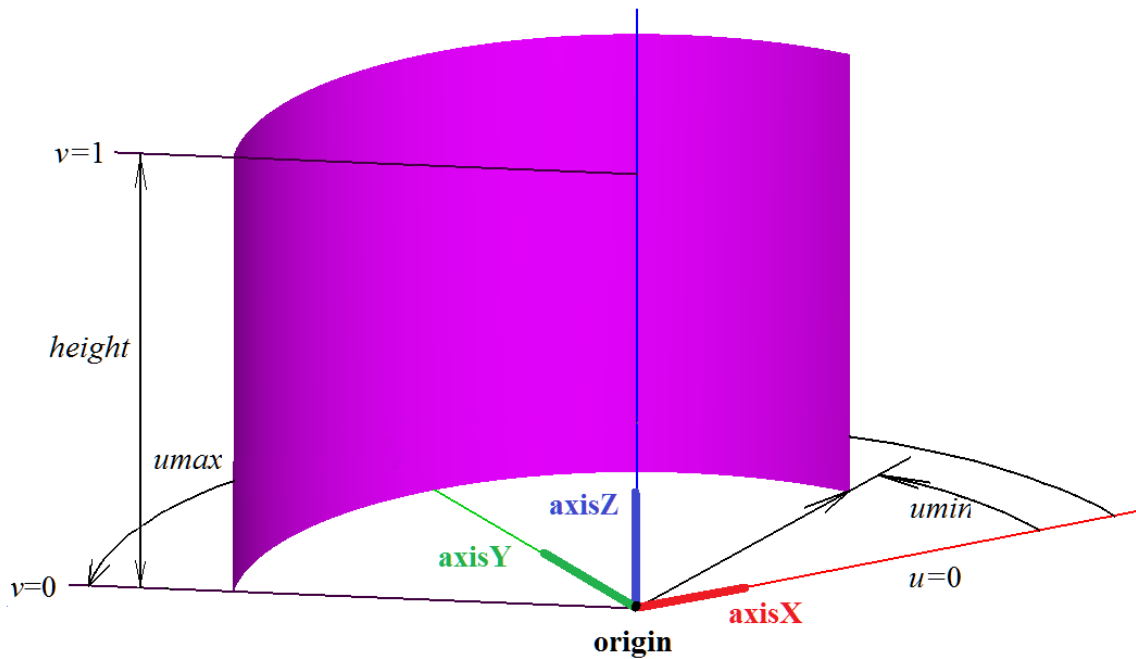


Рис. O.5.3.1.

Радиус и высота должны быть больше нуля: $radius > 0$, $height > 0$. Для граничных параметров поверхности должны соблюдаться неравенства: $umin < umax$, $vmin < vmax$.

Локальная система координат **position** может быть как правой, так и левой. Если локальная система координат правая, то нормаль направлена в сторону выпуклости поверхности (от оси поверхности), если локальная система координат левая, то нормаль направлена в сторону вогнутости поверхности (в сторону оси поверхности).

O.5.4. Коническая поверхность MbConeSurface

Класс MbConeSurface объявлен в файле surf_cone_surface.h.

Коническая поверхность MbConeSurface принадлежит к группе элементарных поверхностей MbElementarySurface. Коническая поверхность описывается радиусом $radius$, высотой $height$ и углом конусности $angle$, заданными в локальной системе координат **MbPlacement3D position**.

Первый параметр поверхности отсчитывается по дуге от вектора **position.axisX** в направлении вектора **position.axisY**. Первый параметр поверхности u принимает значения на отрезке: $umin \leq u \leq umax$. Значения $u=0$ и $u=2\pi$ соответствуют точке на плоскости XZ. Поверхность может быть периодической по первому параметру. У периодической поверхности $umax - umin = 2\pi$, у не периодической поверхности $umax - umin < 2\pi$.

Второй параметр поверхности отсчитывается по прямой вдоль вектора **position.axisZ**. Второй параметр поверхности v принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=0$ соответствует началу локальной системы координат, а значение $v=1$ соответствует точке на расстоянии $height$ от плоскости XY локальной системы координат поверхности.

В методе **PointOn**(double u , double v , **MbCartPoint3D** & s) радиус-вектор поверхности s описывается векторной функцией

$$s(u,v) = \mathbf{position.origin} + (radius + height \cdot v \cdot \tan(angle)) (\cos(u) \mathbf{position.axisX} + \sin(u) \mathbf{position.axisY}) + height \cdot v \mathbf{position.axisZ}.$$

Коническая поверхность приведена на рис. O.5.4.1.

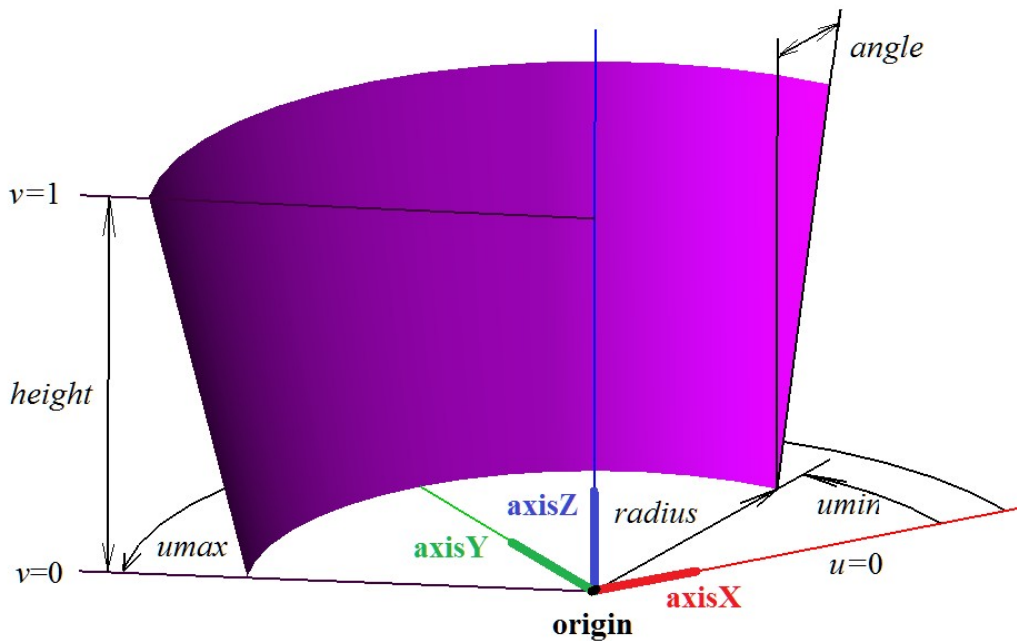


Рис. О.5.4.1.

Радиус и высота должны быть больше нуля, а угол по модулю не должен превышать $\pi/2$: $radius > 0$, $height > 0$, $-\pi/2 < angle < \pi/2$. При $angle = 0$ коническая поверхность эквивалентна цилиндрической поверхности. Для граничных параметров поверхности должны соблюдаться неравенства: $umin < umax$, $vmin < vmax$. Полюсу поверхности соответствует значение второго параметра $v = -radius / (height \cdot tg(angle))$. Граничные параметры $vmax$ и $vmin$ принимают такие значения, при которых поверхность располагается с одной стороны от полюса.

Локальная система координат **position** может быть как правой, так и левой. Если локальная система координат правая, то нормаль направлена в сторону выпуклости поверхности (от оси поверхности), если локальная система координат левая, то нормаль направлена в сторону вогнутости поверхности (в сторону оси поверхности).

О.5.5. Сферическая поверхность MbSphereSurface

Класс MbSphereSurface объявлен в файле surf_sphere_surface.h.

Сфера MbSphereSurface принадлежит к группе элементарных поверхностей MbElementarySurface. Сфера описывается радиусом *radius*, заданными в локальной системе координат [MbPlacement3D](#) **position**.

Первый параметр поверхности отсчитывается по дуге от вектора **position.axisX** в направлении вектора **position.axisY**. Первый параметр поверхности *u* принимает значения на отрезке: $umin \leq u \leq umax$. Значения $u=0$ и $u=2\pi$ соответствуют точке на плоскости XZ. Поверхность может быть периодической по первому параметру. У периодической поверхности $umax - umin = 2\pi$, у не периодической поверхности $umax - umin < 2\pi$.

Второй параметр поверхности отсчитывается по дуге от плоскости XY локальной системы координат поверхности в направлении вектора **position.axisZ**. Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=0$ соответствует точке на плоскости XY локальной системы координат поверхности. Поверхность не периодическа по второму параметру.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & *s*) радиус-вектор поверхности *s* описывается векторной функцией

$$s(u,v) = \mathbf{position.origin} + \\ radius (\cos(u) \mathbf{position.axisX} + \sin(u) \mathbf{position.axisY}) + \\ radius \sin(v) \mathbf{position.axisZ}.$$

Сфера приведена на рис. О.5.5.1.

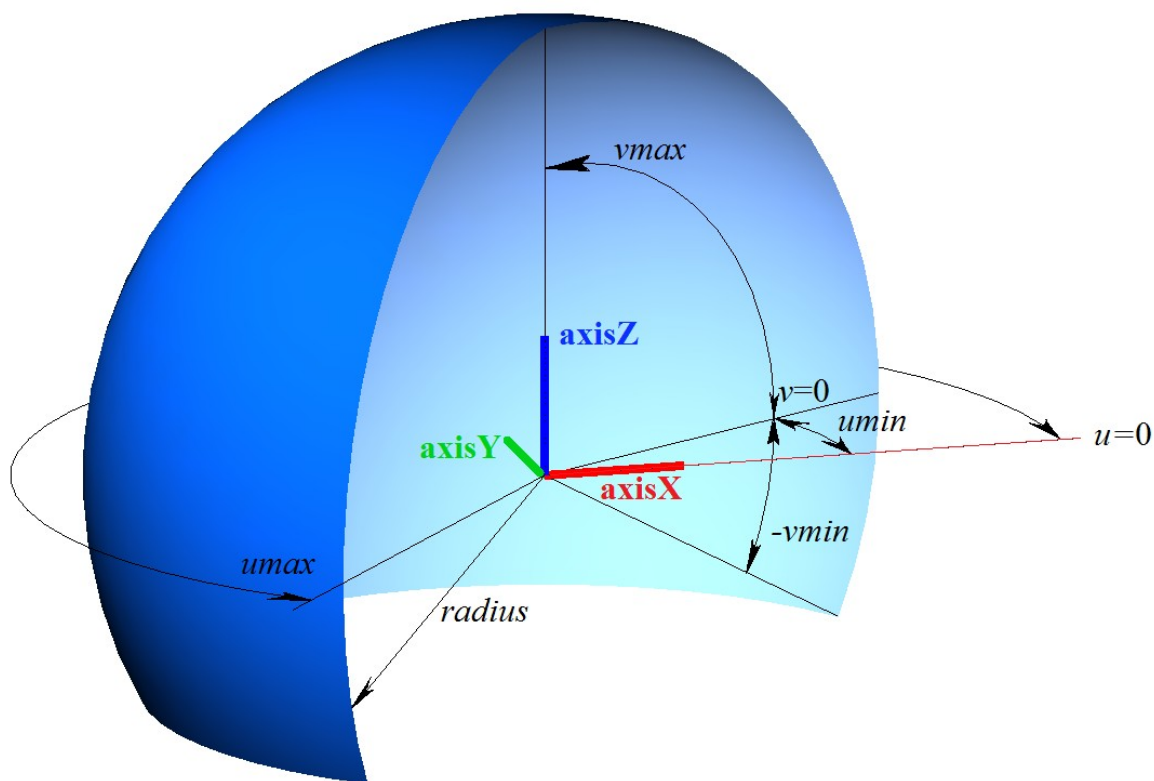


Рис. O.5.5.1.

Радиус сферы должен быть больше нуля: $radius > 0$. Сфера имеет полюсы для параметра $v = \pi/2$ и $v = -\pi/2$. Для граничных параметров поверхности должны соблюдаться неравенства: $umin < umax$, $vmin < vmax$, $vmax \leq \pi/2$, $vmin \geq -\pi/2$.

Локальная система координат **position** может быть как правой, так и левой. Если локальная система координат правая, то нормаль направлена наружу сферы, если локальная система координат левая, то нормаль направлена внутрь сферы.

O.5.6. Поверхность тора MbTorusSurface

Класс MbTorusSurface объявлен в файле surf_torus_surface.h.

Поверхность тора MbTorusSurface принадлежит к группе элементарных поверхностей MbElementarySurface. Поверхность тора описывается радиусом центров *majorRadius* и радиусом трубки *minorRadius*, заданными в локальной системе координат **MbPlacement3D position**.

Первый параметр поверхности отсчитывается по дуге от вектора **position.axisX** в направлении вектора **position.axisY**. Первый параметр поверхности *u* принимает значения на отрезке: $umin \leq u \leq umax$. Значения $u=0$ и $u=2\pi$ соответствуют точке на плоскости XZ. Поверхность может быть периодической по первому параметру. У периодической поверхности $umax - umin = 2\pi$, у не периодической поверхности $umax - umin < 2\pi$.

Второй параметр поверхности отсчитывается по дуге от плоскости XY локальной системы координат поверхности в направлении вектора **position.axisZ**. Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значения $v=0$ и $v=2\pi$ соответствуют точке на плоскости XY локальной системы координат поверхности. Поверхность может быть периодической по второму параметру при $majorRadius > minorRadius$. У периодической поверхности $vmax - vmin = 2\pi$, у не периодической поверхности $vmax - vmin < 2\pi$.

В методе **PointOn**(double *u*, double *v*, **MbCartPoint3D** & *s*) радиус-вектор поверхности *s* описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{position.origin} + (majorRadius + (minorRadius \cos(v)) (\cos(u) \mathbf{position.axisX} + \sin(u) \mathbf{position.axisY}) +$$

$minorRadius \sin(v)$ **position.axisZ**.

Поверхность тора приведена на рис. О.5.6.1.

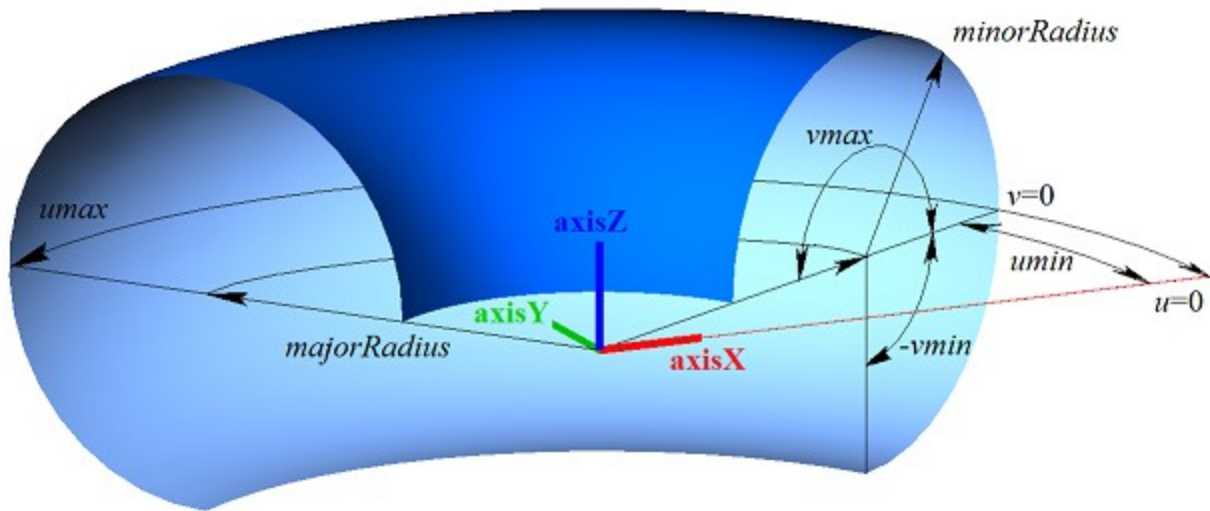


Рис. О.5.6.1.

Радиус трубки должен быть больше нуля: $minorRadius > 0$. Радиус центров должен быть не меньше радиуса трубки, взятого с обратным знаком: $majorRadius > -minorRadius$. Если $majorRadius < minorRadius$, то поверхность имеет полюс для параметра $v = \arccos(majorRadius/minorRadius)$ и $v = 2\pi - \arccos(majorRadius/minorRadius)$. Для граничных параметров поверхности должны соблюдаться неравенства: $umin < umax, vmin < vmax$.

Локальная система координат **position** может быть как правой, так и левой. Если локальная система координат правая, то нормаль направлена от трубки поверхности, если локальная система координат левая, то нормаль направлена внутрь трубки поверхности.

О.5.7. Поверхность выдавливания MbExtrusionSurface

Класс MbExtrusionSurface объявлен в файле surf_extrusion_surface.h.

Поверхность выдавливания MbExtrusionSurface принадлежит к группе поверхностей движения MbSweptSurface. Поверхность выдавливания является частным случаем поверхности движения с прямолинейной направляющей кривой. Поверхность выдавливания описывается образующей кривой [MbCurve3D](#)* **curve**, вектором направления выдавливания [MbVector3D](#) **direction** и длиной выдавливания *distance*.

Первый параметр поверхности *u* совпадает с параметром образующей кривой. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения образующей кривой. Поверхность может быть периодической по первому параметру, если периодической является образующая кривая.

Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=0$ соответствует точке на образующей кривой, значение $v=1$ соответствует точке образующей кривой, смещённой на вектор **direction****distance*. Поверхность не может быть периодической по второму параметру.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{curve}(u) + (\mathbf{direction} \text{ distance } v).$$

Поверхность выдавливания приведена на рис. О.5.7.1.

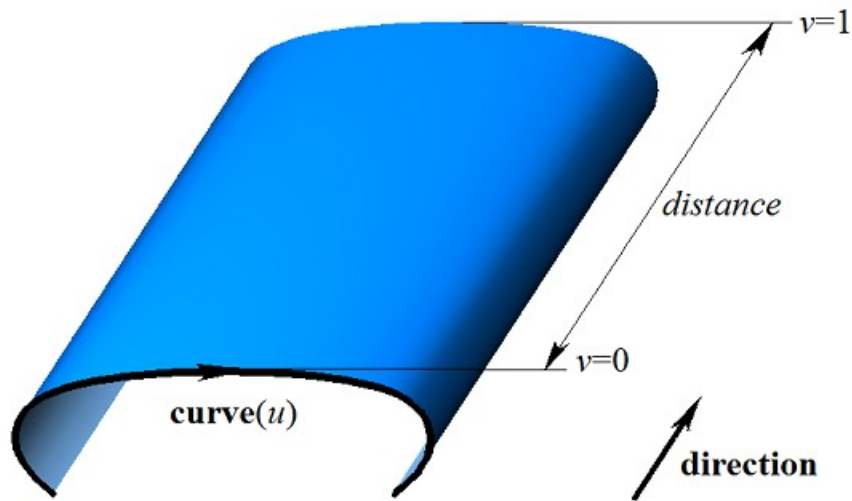


Рис. 0.5.7.1.

Для граничных значений второго параметра поверхности должно соблюдаться неравенство: $v_{min} < v_{max}$.

0.5.8. Поверхность вращения MbRevolutionSurface

Класс MbRevolutionSurface объявлен в файле surf_revolution_surface.h.

Поверхность вращения MbRevolutionSurface принадлежит к группе поверхностей движения MbSweptSurface. Поверхность вращения является частным случаем поверхности движения с направляющей кривой в форме окружности или её дуги. Поверхность вращения описывается образующей кривой [MbCurve3D](#)* **curve**, локальной системой координат [MbPlacement3D](#) **position**, вектор **position.axisZ** которой является осью вращения, признаком расположения кривой и оси вращения в одной плоскости *planeData*, признаком наличия полюса поверхности при начальном значении первого параметра *poleMin*, признаком наличия полюса поверхности при конечном значении первого параметра *poleMax*, значениями первого параметра поверхности в полюсах поверхности *uPoleMin*, *uPoleMax*, если соответствующий полюс присутствует. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности *u* совпадает с параметром образующей кривой. Первый параметр поверхности принимает значения на отрезке $u_{min} \leq u \leq u_{max}$, который соответствует области определения образующей кривой. Поверхность может быть периодической по первому параметру, если периодической является образующая кривая.

Второй параметр поверхности *v* принимает значения на отрезке: $v_{min} \leq v \leq v_{max}$. Значения $v=0$ и $v=2\pi$ соответствуют точке на образующей кривой. Поверхность может быть периодической по второму параметру. У периодической поверхности $v_{max}-v_{min}=2\pi$, у не периодической поверхности $v_{max}-v_{min} < 2\pi$.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{position.origin} + (\mathbf{curve}(u) - \mathbf{position.origin}) \mathbf{M}(v),$$

где $\mathbf{M}(v)$ – матрица вращения. Заметим, что умножение вектора $(\mathbf{curve}(u) - \mathbf{position.origin})$ на матрицу $\mathbf{M}(v)$ выполняется справа. Матрица вращения имеет вид

$$\mathbf{M}(v) = \mathbf{A}^{-1} \cdot \begin{bmatrix} \cos v & -\sin v & 0 \\ \sin v & \cos v & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{A} =$$

$$= \begin{bmatrix} \mathbf{position.axisX} \\ \mathbf{position.axisY} \\ \mathbf{position.axisZ} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \cos v & -\sin v & 0 \\ \sin v & \cos v & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{position.axisX} \\ \mathbf{position.axisY} \\ \mathbf{position.axisZ} \end{bmatrix}.$$

Матрица **A** является матрицей преобразования координат радиуса-вектора точки из локальной системы координат **position** в глобальную систему координат. Строки матрицы **A** составлены из компонент базисных векторов локальной системы координат. Матрица **M(v)** переводит вектор **curve(u) – position.origin** в локальную систему координат, поворачивает его в ней на угол *v* вокруг оси вращения и возвращает повёрнутый вектор обратно в глобальную систему координат. Поверхность вращения приведена на рис. 0.5.8.1.

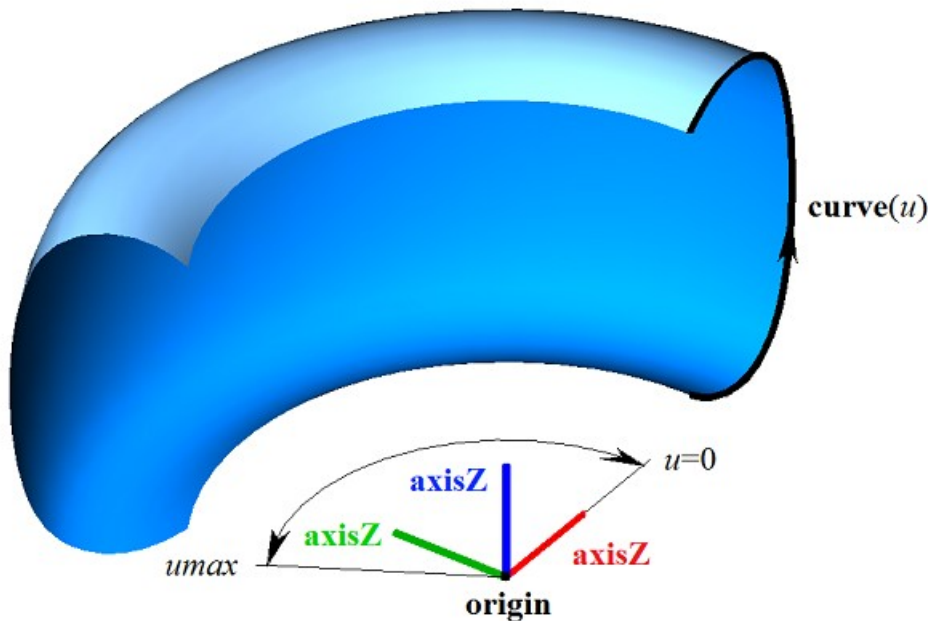


Рис. 0.5.8.1.

Если начальный или конечный край образующей кривой проходит через ось вращения, то поверхность имеет полюс для параметра *umin* или *umax*, соответственно. Для граничных значений второго параметра поверхности должно соблюдаться неравенство: *vmin*<*vmax*.

Вращение точек образующей кривой выполняется по дуге вокруг вектора **position.axisZ** от вектора **position.axisX** в направлении вектора **position.axisY**. Локальная система координат **position** может быть как правой, так и левой.

0.5.9. Поверхность перемещения MbExpansionSurface

Класс MbExpansionSurface объявлен в файле surf_expansion_surface.h.

Поверхность перемещения MbExpansionSurface принадлежит к группе поверхностей движения MbSweptSurface. Поверхность перемещения является частным случаем поверхности движения с криволинейной направляющей кривой. Поверхность перемещения описывается образующей кривой **MbCurve3D* curve**, направляющей кривой **MbCurve3D* spine**, точкой в начале направляющей **MbCartPoint3D origin**. Поверхность образована перемещением образующей кривой вдоль направляющей кривой. В частном случае образующая кривая поверхности перемещения может менять свою форму. В последнем случае в данных кривой присутствует вторая образующая кривая **brink**, точка в конце направляющей **ending**, начальный параметр *tmin* кривой **brink** и производная *dt* параметра кривой **brink** по параметру образующей кривой **curve**. Производная *dt* определяется равенством

$$dt = \frac{tmax - tmin}{umax - umin},$$

где $tmax$ – конечный параметр кривой **brink**. В общем случае указатель на вторую образующую кривую **brink** может быть равен нулю, что означает, что вторая образующая кривая отсутствует.

Первый параметр u поверхности совпадает с параметром образующей кривой **curve**. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения образующей кривой. Поверхность может быть периодической по первому параметру, если периодической является образующая кривая.

Второй параметр поверхности v совпадает с параметром направляющей кривой и принимает значения на отрезке: $vmin \leq v \leq vmax$. Поверхность может быть периодической по второму параметру, если периодической является направляющая кривая и отсутствует вторая образующая кривая.

В методе **PointOn**(double u , double v , [MbCartPoint3D](#) & s) в общем случае радиус-вектор поверхности s описывается векторной функцией

$$s(u,v) = spine(v) + curve(u) - origin.$$

Поверхность перемещения в общем случае приведена на рис. О.5.9.1.

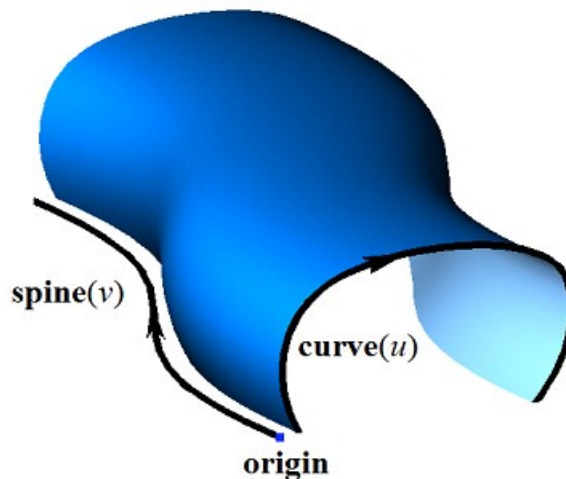


Рис. О.5.9.1.

В методе **PointOn**(double u , double v , [MbCartPoint3D](#) & s) в частном случае радиус-вектор поверхности s описывается векторной функцией

$$s(u,v) = spine(v) + (curve(u) - origin) (1-w) + (brink(t) - ending) w,$$

где $w = \frac{v - vmin}{vmax - vmin}$, $t = tmin + (u - umin)dt$. Поверхность перемещения в частном случае приведена на рис. О.5.9.2.

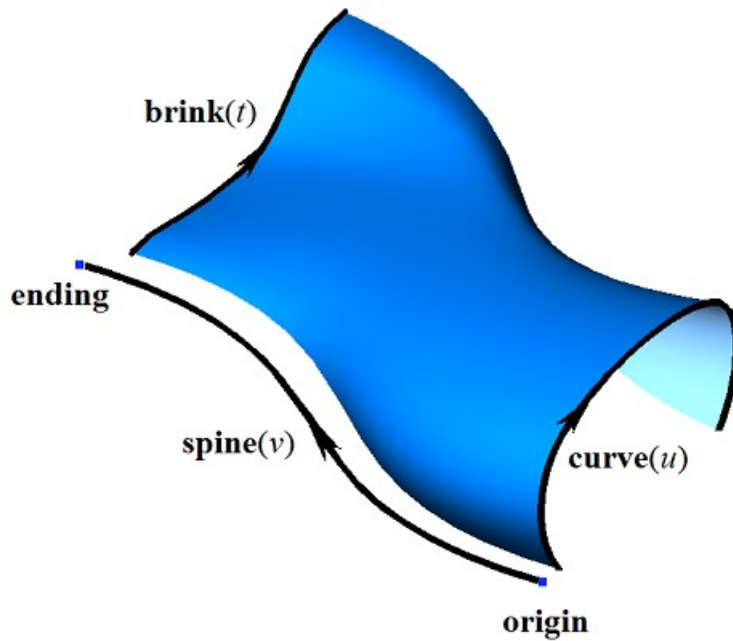


Рис. 0.5.9.2.

Для отсутствия самопересечений поверхности образующая и направляющие кривые не должны иметь параллельных друг другу участков. В определенных случаях поверхность перемещения может иметь особые точки.

0.5.10. Спиральная поверхность MbSpiralSurface

Класс MbSpiralSurface объявлен в файле surf_spiral_surface.h.

Спиральная поверхность MbSpiralSurface принадлежит к группе поверхностей движения MbSweptSurface. Спиральная поверхность является частным случаем поверхности движения с направляющей кривой в форме цилиндрической спирали. Спиральная поверхность описывается образующей кривой [MbCurve3D](#)* **curve**, локальной системой координат [MbPlacement3D](#) **position**, вектор **position.axisZ** которой является осью спирали, радиусом спирали *radius*, шагом спирали *step*, положением начала спирали **origin** и граничными параметрами спирали *vmin* и *vmax*. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Ось спирали совпадает с координатной осью **position.axisZ** локальной системы координат. Первый параметр поверхности *u* совпадает с параметром образующей кривой. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения образующей кривой. Поверхность может быть периодической по первому параметру, если периодической является образующая кривая.

Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=0$ соответствует точке на образующей кривой. Значения второго параметра *v*, равное 2π , соответствуют точкам образующей кривой, смещенным на вектор **position.axisZ**, умноженный на *step*. Поверхность не может быть периодической по второму параметру.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$s(u,v) = \mathbf{position.origin} + \mathit{radius} (\cos(t) \mathbf{position.axisX} + \sin(t) \mathbf{position.axisY}) + ((t \mathit{step}/2\pi) \mathbf{position.axisZ}) + (\mathbf{curve}(u) - \mathbf{origin}) \mathbf{M}(v),$$

где **M(v)** – матрица вращения. Заметим, что умножение вектора **(curve(u)–origin)** на матрицу **M(v)** выполняется справа. Матрица вращения имеет вид

$$\mathbf{M}(v) = \mathbf{A}^{-1} \cdot \begin{bmatrix} \cos v & -\sin v & 0 \\ \sin v & \cos v & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{A} =$$

$$= \begin{bmatrix} \text{position.a axisX} \\ \text{position.a axisY} \\ \text{position.a axisZ} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \cos v & -\sin v & 0 \\ \sin v & \cos v & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \text{position.a axisX} \\ \text{position.a axisY} \\ \text{position.a axisZ} \end{bmatrix}.$$

Матрица \mathbf{A} является матрицей преобразования координат радиуса-вектора точки из локальной системы координат **position** в глобальную систему координат. Строки матрицы \mathbf{A} составлены из компонент базисных векторов локальной системы координат. Матрица $\mathbf{M}(v)$ переводит вектор **curve(u)–origin** в локальную систему координат, поворачивает его в ней на угол v вокруг оси вращения и возвращает повернутый вектор обратно в глобальную систему координат. Спиральная поверхность приведена на рис. O.5.10.1.

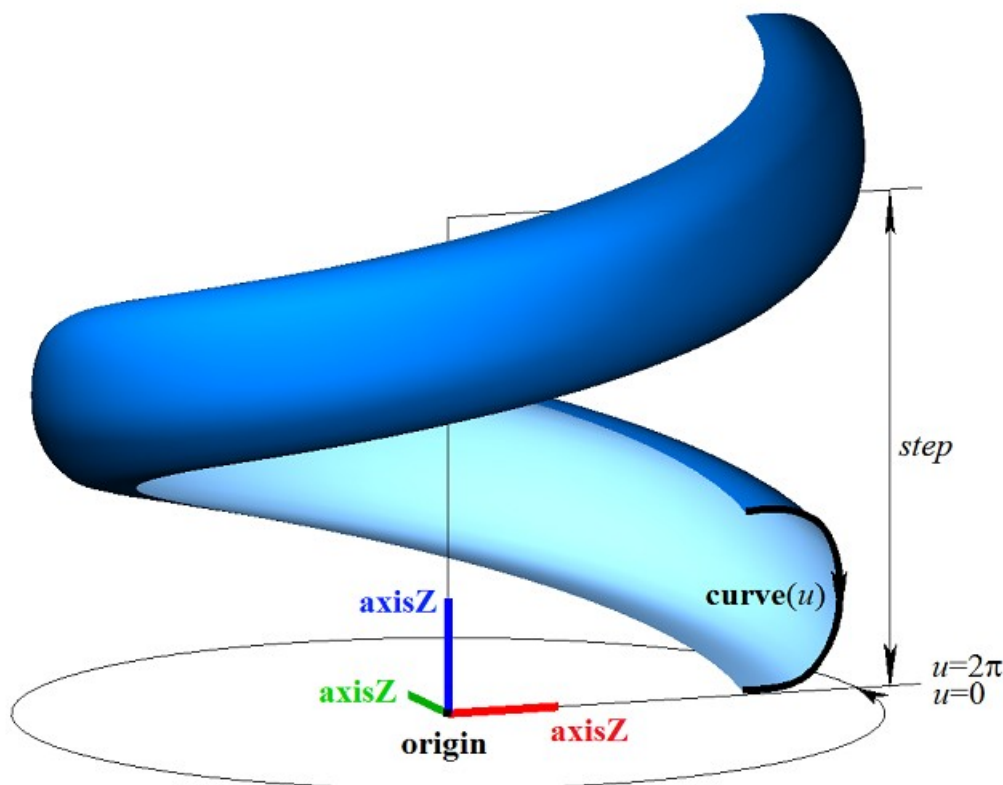


Рис. O.5.10.1.

Для граничных значений второго параметра поверхности должно соблюдаться неравенство: $v_{min} < v_{max}$.

O.5.11. Кинематическая поверхность MbEvolutionSurface

Класс MbEvolutionSurface объявлен в файле surf_evolution_surface.h.

Кинематическая поверхность MbEvolutionSurface принадлежит к группе поверхностей движения MbSweptSurface. Кинематическая поверхность является общим случаем поверхности движения с произвольной направляющей кривой. Поверхность вращения описывается образующей кривой [MbCurve3D](#)* **curve**, направляющим объектом [MbCurve3D](#)* **spine**, положением начала направляющей [MbCartPoint3D](#) **origin**. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности u совпадает с параметром образующей кривой **curve**. Первый параметр поверхности принимает значения на отрезке $u_{min} \leq u \leq u_{max}$, который соответствует области

определения образующей кривой. Поверхность может быть периодической по первому параметру, если периодической является образующая кривая.

Направляющий объект **spine** заменяет собой направляющую кривую, построен на базе кривой и отличается от последней тем, что может генерировать локальную систему координат, связанную с кривой. Второй параметр поверхности v совпадает с параметром кривой направляющего объекта **spine**. Второй параметр поверхности принимает значения на отрезке $v_{min} \leq v \leq v_{max}$, который соответствует области определения направляющей кривой. Поверхность может быть периодической по второму параметру, если периодической является направляющая кривая.

В методе **PointOn**(double u , double v , [MbCartPoint3D](#) & s) радиус-вектор поверхности s описывается векторной функцией

$$s(u,v) = \mathbf{spine}(v) + (\mathbf{curve}(u) - \mathbf{origin}) \mathbf{M}(v),$$

где $\mathbf{M}(v)$ – матрица, связанная с направляющей кривой. Заметим, что умножение вектора $(\mathbf{curve}(u) - \mathbf{origin})$ на матрицу $\mathbf{M}(v)$ выполняется справа. Матрица $\mathbf{M}(v)$ имеет вид

$$\mathbf{M}(v) = \mathbf{A}^{-1}(v_{min}) \cdot \mathbf{A}(v),$$

где матрица $\mathbf{A}(v)$ является матрицей преобразования координат радиуса-вектора точки из подвижной системы координат, связанной с направляющей кривой, в глобальную систему координат. Матрица $\mathbf{A}(v)$ зависит от второго параметра поверхности. Строки матрицы $\mathbf{A}(v)$ составлены из компонент базисных векторов подвижной системы координат:

$$\mathbf{A}(v) = \begin{bmatrix} \mathbf{i}_1(v) \\ \mathbf{i}_2(v) \\ \mathbf{i}_3(v) \end{bmatrix},$$

где $\mathbf{i}_1(v)$ – касательный вектор направляющей кривой, $\mathbf{i}_2(v)$ – вектор, ортогональный $\mathbf{i}_1(v)$ и связанный с вектором **direction** направляющего объекта **spine**, $\mathbf{i}_3(v)$ – вектор, ортогональный $\mathbf{i}_1(v)$ и $\mathbf{i}_2(v)$. Поверхность вращения приведена на рис. О.5.11.1.

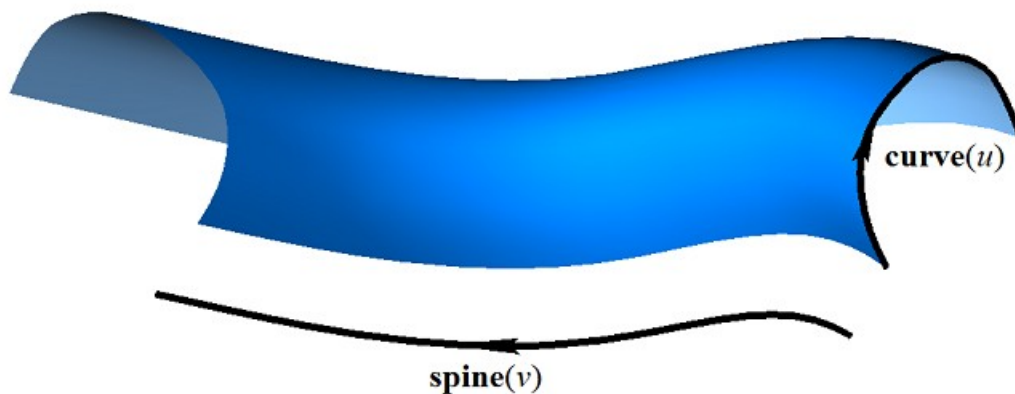


Рис. О.5.11.1.

Касательный вектор $\mathbf{i}_1(v)$ вычисляется по направляющей кривой. Вектор $\mathbf{i}_2(v)$ вычисляется из условия плавного изменения при переходе от точки к точке направляющей кривой и ортогональности $\mathbf{i}_1(v)$. Вектор $\mathbf{i}_3(v)$ вычисляется как векторное произведение векторов $\mathbf{i}_1(v)$ и $\mathbf{i}_2(v)$.

О.5.12. Кинематическая поверхность с адаптацией MbExactionSurface

Класс MbExactionSurface объявлен в файле surf_exaction_surface.h.

Кинематическая поверхность с адаптацией MbExactionSurface является наследником кинематической поверхности [MbEvolutionSurface](#). Кинематическая поверхность с адаптацией

используется для построения тел кинематической операцией с направляющей в форме составной кривой, сегменты которой стыкуются с изломом, рис. О.5.12.1.

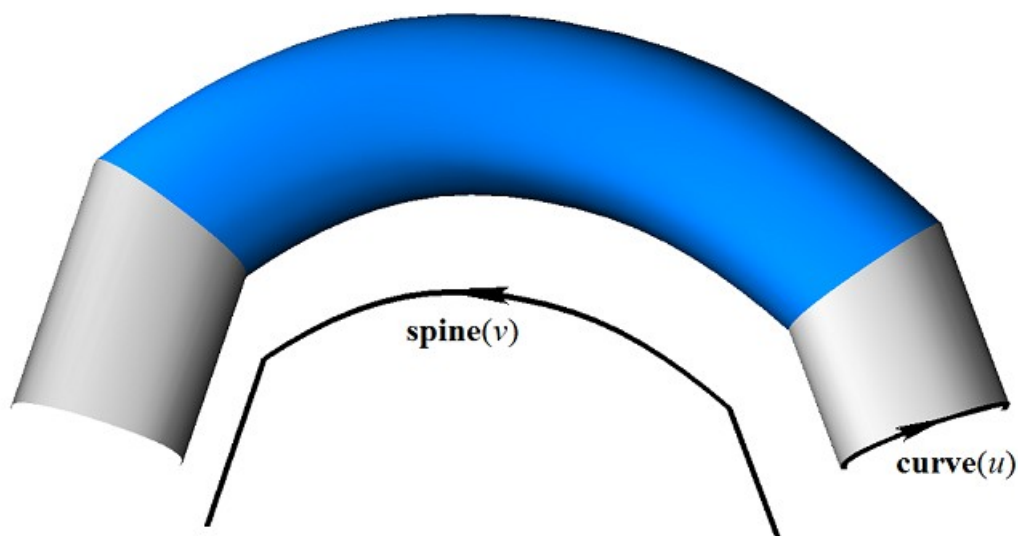


Рис. О.5.12.1.

Кинематическая поверхность MbExactionSurface подстраивает свои торцы для обеспечения стыковки с другой поверхностью.

О.5.13. Секториальная поверхность MbSectorSurface

Класс MbSectorSurface объявлен в файле surf_sector_surface.h.

Секториальная поверхность MbSectorSurface принадлежит к группе поверхностей движения MbSweptSurface. Секториальная поверхность описывается кривой [MbCurve3D](#)* **curve** и точкой [MbCartPoint3D](#) **origin**.

Первый параметр поверхности u совпадает с параметром кривой **curve**. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения кривой **curve**. Поверхность может быть периодической по первому параметру, если периодической является кривая **curve**.

Второй параметр поверхности v принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=vmin$ соответствует точке на кривой **curve**, значение $v=vmax$ соответствует точке **origin**. Поверхность не может быть периодической по второму параметру.

В методе [PointOn](#)(double u , double v , [MbCartPoint3D](#) & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{curve}(u) (1-w) + \mathbf{origin} w,$$

где $w = \frac{v - vmin}{vmax - vmin}$.

Секториальная поверхность приведена на рис. О.5.13.1.

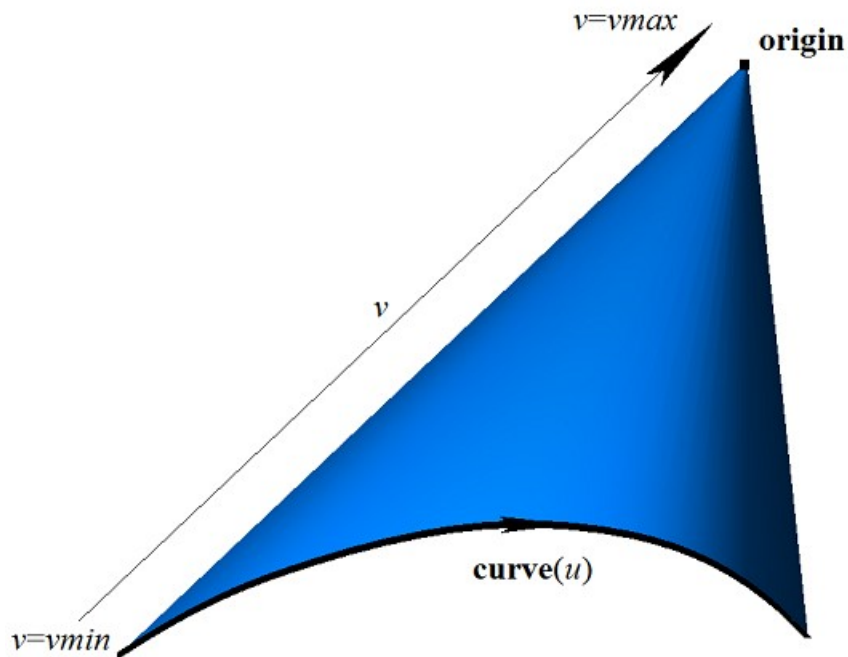


Рис. O.5.13.1.

Кривые поверхности $s(const, v)$ являются отрезками прямой. Поверхность имеет полюс в точке **origin** при $v=vmax$. Секториальная поверхность является частным случаем линейчатой поверхности.

O.5.14. Линейчатая поверхность MbRuledSurface

Класс MbRuledSurface объявлен в файле surf_ruled_surface.h.

Линейчатая поверхность MbRuledSurface принадлежит к группе поверхностей движения MbSweptSurface. Линейчатая поверхность описывается кривой [MbCurve3D](#)* **curve**, кривой [MbCurve3D](#)* **sline**, признаком наличия полюса поверхности при начальном значении первого параметра *poleMin*, признаком наличия полюса поверхности при конечном значении первого параметра *poleMax*, начальным параметром *tmin* кривой **sline**, производной *dt* параметра кривой **sline** по параметру кривой **curve** и типом формы поверхности *type*. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности *u* совпадает с параметром кривой **curve**. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения кривой **curve**. Производная *dt* определяется равенством

$$dt = \frac{tmax - tmin}{umax - umin},$$

где *tmax* – конечный параметр кривой **sline**. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve** и **sline**.

Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=vmin$ соответствует точке на кривой **curve**, значение $v=vmax$ соответствует точке на кривой **sline**. Поверхность не может быть периодической по второму параметру.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$s(u, v) = \mathbf{curve}(u) (1-w) + \mathbf{sline}(t) w,$$

где $w = \frac{v - vmin}{vmax - vmin}$, $t = tmin + (u - umin)dt$.

Линейчатая поверхность приведена на рис. O.5.14.1.

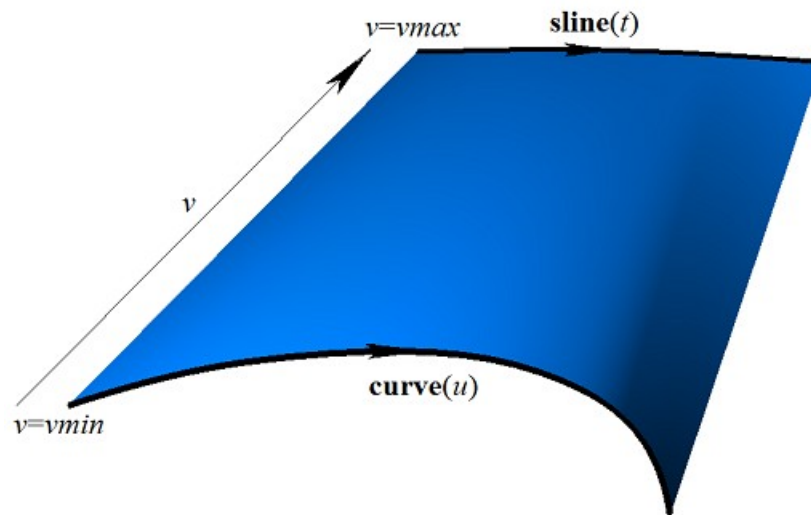


Рис. О.5.14.1.

Кривые поверхности $s(const, v)$ с параметрами $u=const$ являются отрезками прямой. Поверхность может иметь полюс, если одна из кривых **curve** или **sline** стянута в точку, или если кривые **curve** и **sline** совпадают на одном из краёв.

О.5.15. Поверхность на семействе кривых MbLoftedSurface

Класс MbLoftedSurface объявлен в файле surf_lofted_surface.h.

Поверхность MbLoftedSurface описывается множеством кривых `RArray<MbCurve3D>uCurves`, множеством значений второго параметра поверхности для кривых `vParams`, множеством признаков одинаковых кривых `vLabels`, граничными значениями параметров `umin`, `umax`, `vmin`, `vmax`, признаками замкнутости поверхности по первому и второму параметрам `uClosed` и `vClosed`, вектором направления непериодической поверхности при `vmin` `MbVector3D derive1`, вектором направления непериодической поверхности при `vmax` `MbVector3D derive2`, признаками наличия полюсов поверхности на границе области определения `poleUMin`, `poleUMax`, `poleVMin`, `poleVMax`. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности u совпадает с параметрами кривых **curves**. Все кривые **curves** должны иметь одинаковую область определения параметра, для этого могут использоваться кривые MbReparamCurve3D. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения кривых **curves**. Поверхность может быть периодической по первому параметру, если периодическими являются все кривые множества **curves**.

Второй параметр поверхности v принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=vmin$ соответствует начальному значению множества `vParams[0]`, значение $v=vmax$ соответствует конечному значению множества `vParams[vParams.MaxIndex()]`. Поверхность может быть периодической по второму параметру.

Если все кривые множества **curves** разные, то значения множества `vLabels` равны индексу кривых множества **curves**. Если среди рядом расположенных кривых множества **curves** есть одинаковые кривые (но смещенные друг относительно друга), то соответствующие значения множества `vLabels` равны минимальному индексу размноженной кривой множества **curves**.

В методе `PointOn(double u, double v, MbCartPoint3D & s)` радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u, v) = (1-3w^2+2w^3) \mathbf{curves}[i](u) + (3w^2+2w^3) \mathbf{curves}[i+1](u) + (w-2w^2+w^3) \mathbf{derive}[i](u) + (-w^2+w^3) \mathbf{derive}[i+1](u) (vParams[i+1] - vParams[i]),$$

где $w = \frac{v - vParams[i]}{vParams[i+1] - vParams[i]}$, **derive**[i] и **derive**[i+1] – производные кривых **curves**[i] и **curves**[i+1], соответственно. Индекс i рабочего участка для вычисления радиуса-вектора точки и его производных вычисляется из условия $vParams[i] \leq v \leq vParams[i+1]$. Если среди соседних элементов

множества $vLabels$ есть одинаковые значения, то между соответствующими кривыми поверхность эквивалентна поверхности выдавливания. Поверхность на семействе кривых приведена на рис. O.5.15.1.

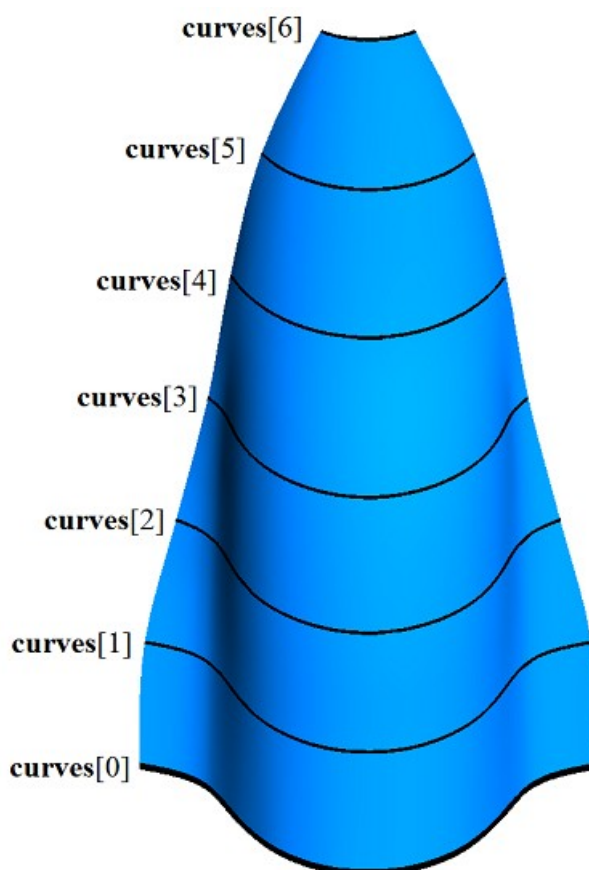


Рис. O.5.15.1.

Форма поверхности зависит от расположения кривых и от множества $vParams$ значений параметров, при которых поверхность проходит по кривым. Для предотвращения самопересечений поверхности необходимо, чтобы значения множества $vParams$ изменялись пропорционально среднему расстоянию между кривыми.

Кривые поверхности $s(const, v)$ с параметрами $u=const$ являются кривыми Эрмита [MbHermit3D](#). Поверхность может иметь полюсы, если первая и/или последняя из кривых **curves** стянута в точку, или если все кривые **curves** совпадают на одном из краёв.

O.5.16. Поверхность на семействе кривых и направляющей **MbElevationSurface**

Класс MbElevationSurface объявлен в файле surf_elevation_surface.h.

Поверхность на семействе кривых и направляющей является наследником класса [MbLoftedSurface](#). Так же как и поверхность [MbLoftedSurface](#) поверхность MbElevationSurface описывается множеством образующих кривых `RArray<MbCurve3D>uCurves`, множеством значений второго параметра поверхности для кривых $vParams$, граничными значениями параметров $umin$, $umax$, $vmin$, $vmax$, признаками замкнутости поверхности по первому и второму параметрам $uClosed$ и $vClosed$, признаками наличия полюсов поверхности на границе области определения $poleUMin$, $poleUMax$, $poleVMin$, $poleVMax$. Кроме перечисленных данных поверхность MbElevationSurface описывается направляющей кривой [MbCurve3D](#)* **spine**. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности u совпадает с параметрами кривых **curves**. Все кривые **curves** должны иметь одинаковую область определения параметра, для этого могут использоваться кривые MbReparamCurve3D. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$,

который соответствует области определения кривых **curves**. Поверхность может быть периодической по первому параметру, если периодическими являются все кривые множества **curves**.

Второй параметр поверхности v совпадает с параметром направляющей кривой **spine**. Второй параметр поверхности принимает значения на отрезке $v_{min} \leq v \leq v_{max}$, который соответствует области определения направляющей кривой. Поверхность может быть периодической по второму параметру, если периодической является направляющая кривая.

В методе **PointOn**(double u , double v , **MbCartPoint3D** & s) радиус-вектор поверхности s вычисляется аналогично вычислению радиуса-вектора поверхности **MbLoftedSurface** с коррекцией смещения направляющей кривой. Поверхность на семействе кривых и направляющей приведена на рис. O.5.16.1.

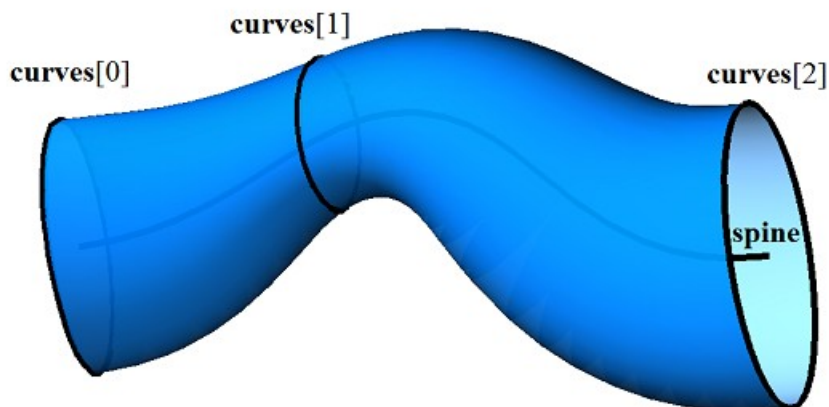


Рис. O.5.16.1.

Форма поверхности зависит от расположения образующих кривых, направляющей кривой и от множества $vParams$ значений параметров, при которых поверхность проходит по кривым. Значения множества $vParams$ определяются путем проецирования центров масс образующих кривых на направляющую кривую.

O.5.17. Поверхность на трёх кривых **MbCornerSurface**

Класс **MbCornerSurface** объявлен в файле `surf_corner_surface.h`.

Поверхность на трёх кривых **MbCornerSurface** описывается кривыми **MbCurve3D*** **curve0**, **curve1**, **curve2**, тремя точками **MbCartPoint3D** **vertex[3]** и тремя парами граничных значений параметров соответствующих кривых: $t0min$, $t0max$, $t1min$, $t1max$, $t2min$, $t2max$. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности u принимает значения на отрезке $0 \leq u \leq 1$. Поверхность не может быть периодической по первому параметру. Поверхность имеет особую точку при минимальном значении первого параметра $u=0$. В особой точке производная радиуса-вектора поверхности по второму параметру равна нулю.

Второй параметр поверхности v принимает значения на отрезке $0 \leq v \leq 1$. Поверхность не может быть периодической по второму параметру.

Кривые **curve0**, **curve1**, **curve2** должна иметь точки пересечения или точки скрещения друг с другом. Параметры кривых $t0min$, $t0max$, $t1min$, $t1max$, $t2min$, $t2max$ вычисляются по точкам пересечения или скрещения кривых и определяют рабочие участки кривых и точки **vertex[3]**. Направления кривых для поверхности не имеют значения.

В методе **PointOn**(double u , double v , **MbCartPoint3D** & s) радиус-вектор поверхности s описывается векторной функцией

$$s(u,v) = w_0 (\mathbf{curve2}(t_2) + \mathbf{curve1}(s_1) - \mathbf{vertex}[0]) + \\ + w_1 (\mathbf{curve0}(t_0) + \mathbf{curve2}(s_2) - \mathbf{vertex}[1]) + \\ + w_2 (\mathbf{curve1}(t_1) + \mathbf{curve0}(s_0) - \mathbf{vertex}[2]),$$

где $w_0=1-u$, $w_1=0.5(u-uv)$, $w_2=0.5(u+uv)$ – барицентрические координаты поверхности, $t_0=w_2t_{0min}+(1-w_2)t_{0max}$, $s_0=(1-w_1)t_{0min}+w_1t_{0max}$ – параметры кривой **curve0**, $t_1=w_0t_{1min}+(1-w_0)t_{1max}$, $s_1=(1-w_2)t_{1min}+w_2t_{1max}$ – параметры кривой **curve1**, $t_2=w_1t_{2min}+(1-w_1)t_{2max}$, $s_2=(1-w_0)t_{2min}+w_0t_{2max}$ – параметры кривой **curve2**. Поверхность на трех скрещивающихся кривых приведена на рис. O.5.17.1.

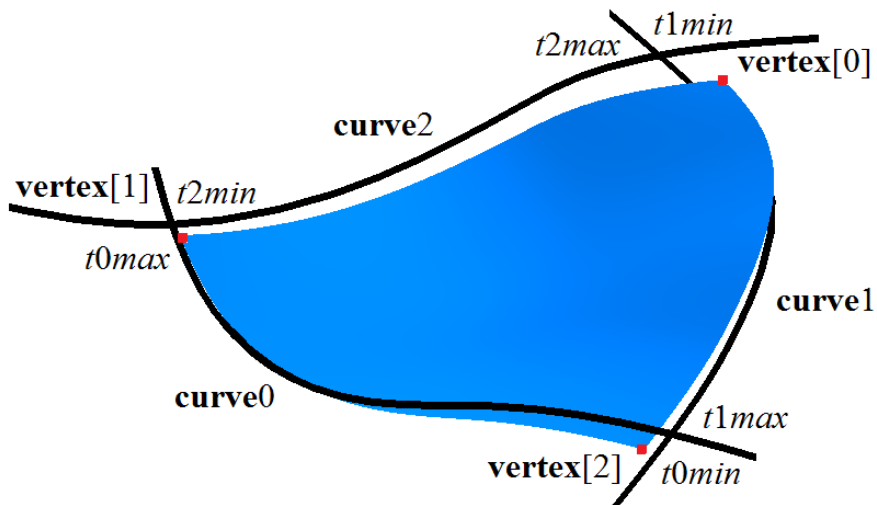


Рис. O.5.17.1.

На рис. O.5.17.2 приведена совпадающая с частью поверхности сферы поверхность, построенная на трёх одинаковых дугах окружностей: плоскости дуг окружностей ортогональны друг другу, дуги пересекаются в крайних точках, каждая дуга содержит четверть окружности.

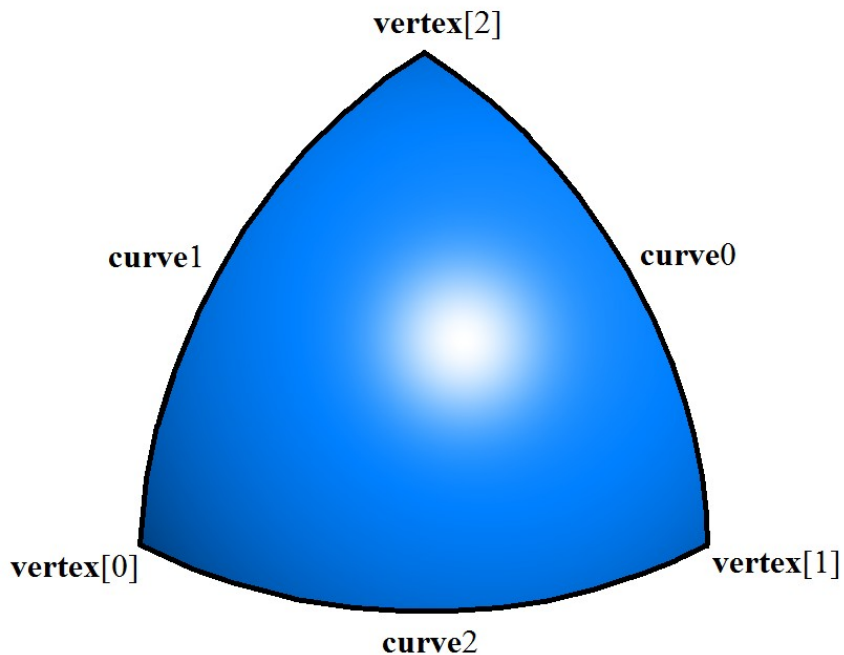


Рис. O.5.17.2.

Форма поверхности зависит от формы кривых. Если кривые не пересекаются, то поверхность не проходит по кривым. Если кривые пересекаются, то точки **vertex[3]** расположены в точках пересечения, и поверхность проходит по участкам кривых: при $w_0=0$ поверхность проходит по участку кривой **curve0**, при $w_1=0$ поверхность проходит по участку кривой **curve1**, при $w_2=0$ поверхность проходит по участку кривой **curve2**.

O.5.18. Поверхность Кунса MbCoverSurface

Класс MbCoverSurface объявлен в файле surf_cover_surface.h.

Поверхность Кунса MbCoverSurface описывается кривыми [MbCurve3D](#)* **curve0**, **curve1**, **curve2**, **curve3**, четырьмя точками [MbCartPoint3D](#) **vertex[4]**, четырьмя парами граничных значений параметров соответствующих кривых: $t0min$, $t0max$, $t1min$, $t1max$, $t2min$, $t2max$, $t3min$, $t3max$, признаком периодичности по первому параметру поверхности *uclosed*, признаком периодичности по второму параметру поверхности *vclosed*, признаком наличия полюса поверхности при начальном значении первого параметра *poleUMin*, признаком наличия полюса поверхности при конечном значении первого параметра *poleUMax*, признаком наличия полюса поверхности при начальном значении второго параметра *poleVMin*, признаком наличия полюса поверхности при конечном значении второго параметра *poleVMax*. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности *u* принимает значения на отрезке $0 \leq u \leq 1$. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve0** и **curve2**.

Второй параметр поверхности *v* принимает значения на отрезке $0 \leq v \leq 1$. Поверхность может быть периодической по второму параметру, если периодическими являются кривые **curve1** и **curve3**.

Соседние кривые **curve0**, **curve1**, **curve2**, **curve3** должна иметь точки пересечения или точки скрещения друг с другом. Параметры кривых $t0min$, $t0max$, $t1min$, $t1max$, $t2min$, $t2max$, $t3min$, $t3max$ вычисляются по точкам пересечения или скрещения кривых и определяют рабочие участки кривых и точки **vertex[4]**. Направления кривых для поверхности не имеют значения.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & *s*) радиус-вектор поверхности *s* описывается векторной функцией

$$\begin{aligned}
 \mathbf{s}(u,v) = & (1-v) (\mathbf{curve0}(t0) - (1-u) \mathbf{vertex}[0]) + \\
 & + u (\mathbf{curve1}(t1) - (1-v) \mathbf{vertex}[1]) + \\
 & + v (\mathbf{curve2}(t2) - u \mathbf{vertex}[2]) + \\
 & + (1-u) (\mathbf{curve3}(t3) - v \mathbf{vertex}[3]),
 \end{aligned}$$

где $w0=1-u$, $w1=0.5(u-uv)$, $w2=0.5(u+uv)$ – барицентрические координаты поверхности, $t0=(1-u)t0min+ut0max$ – параметр кривой **curve0**, $t1=(1-v)t1min+vt1max$ – параметр кривой **curve1**, $t2=(1-u)t2min+ut2max$ – параметр кривой **curve2**, $t3=(1-v)t3min+vt3max$ – параметр кривой **curve3**. Поверхность Кунса на четырёх скрещивающихся кривых приведена на рис. O.5.18.1.

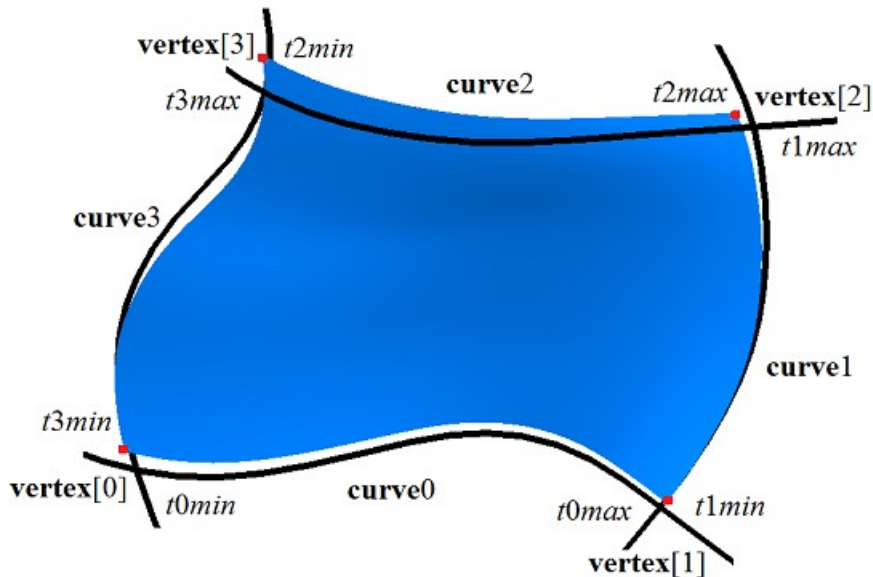


Рис. O.5.18.1.

Форма поверхности зависит от формы кривых. Если соседние кривые пересекаются в точках **vertex[4]**, то поверхность проходит по участкам кривых: при $u=0$ поверхность проходит по участку кривой **curve3**, при $v=0$ поверхность проходит по участку кривой **curve0**, при $u=1$ поверхность проходит по участку кривой **curve1**, при $v=1$ поверхность проходит по участку кривой **curve2**.

О.5.19. Поверхность Кунса MbCoonsPatchSurface

Класс MbCoonsPatchSurface объявлен в файле surf_coons_surface.h.

Бикубическая поверхность Кунса MbCoonsPatchSurface построена аналогично поверхности Кунса [MbCoverSurface](#) и имеет дополнительные условия для радиуса-вектора на краях. Бикубическая поверхность Кунса MbCoonsPatchSurface описывается четырьмя кривыми [MbCurve3D](#)* **curve0**, **curve1**, **curve2**, **curve3**, производной **curve0V** радиуса-вектора поверхности вдоль кривой **curve0** по второму параметру поверхности, производной **curve1U** радиуса-вектора поверхности вдоль кривой **curve1** по первому параметру поверхности, производной **curve2V** радиуса-вектора поверхности вдоль кривой **curve2** по второму параметру поверхности, производной **curve3U** радиуса-вектора поверхности вдоль кривой **curve3** по первому параметру поверхности, четырьмя угловыми точками **vertex[4]**, четырьмя производными по первому параметру поверхности в угловых точках **vertexU[4]**, четырьмя производными по второму параметру поверхности в угловых точках **vertexV[4]**, четырьмя смешанными производными по первому и второму параметрам поверхности в угловых точках **vertexUV[4]**, четырьмя парами граничных значений параметров соответствующих кривых: *t0min*, *t0max*, *t1min*, *t1max*, *t2min*, *t2max*, *t3min*, *t3max*, признаком периодичности по первому параметру поверхности *uclosed*, признаком периодичности по второму параметру поверхности *vclosed*. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности *u* принимает значения на отрезке $0 \leq u \leq 1$. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve0**, **curve2**, **curveV0**, **curveV2**, а кривые **curveU1** и **curveU3** совпадают.

Второй параметр поверхности *v* принимает значения на отрезке $0 \leq v \leq 1$. Поверхность может быть периодической по второму параметру, если периодическими являются кривые **curve1** и **curve3**, **curveU1**, **curveU3**, а кривые **curveV0** и **curveV2** совпадают.

Соседние кривые **curve0**, **curve1**, **curve2**, **curve3** должны иметь точки пересечения или точки скрещения друг с другом. Параметры кривых *t0min*, *t0max*, *t1min*, *t1max*, *t2min*, *t2max*, *t3min*, *t3max* вычисляются по точкам пересечения или скрещения кривых и определяют рабочие участки кривых, точки **vertex[4]**, **vertexU[4]**, **vertexV[4]**, **vertexUV[4]**. Направления кривых для поверхности не имеют значения, но параметризация пар кривых **curve0** и **curveV0**, **curve2** и **curveV2**, **curve1** и **curveU1**, **curve3** и **curveU3** должна совпадать.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & *s*) радиус-вектор поверхности *s* описывается векторной функцией, которая приведена в книге «Геометрическое моделирование», автор Голованов Н.Н. Бикубическая поверхность Кунса строится по заранее рассчитанным данным и используется для построения сопряжений и заплаток с условиями сопряжения на краях. Бикубическая поверхность Кунса приведена на рис. О.5.19.1.

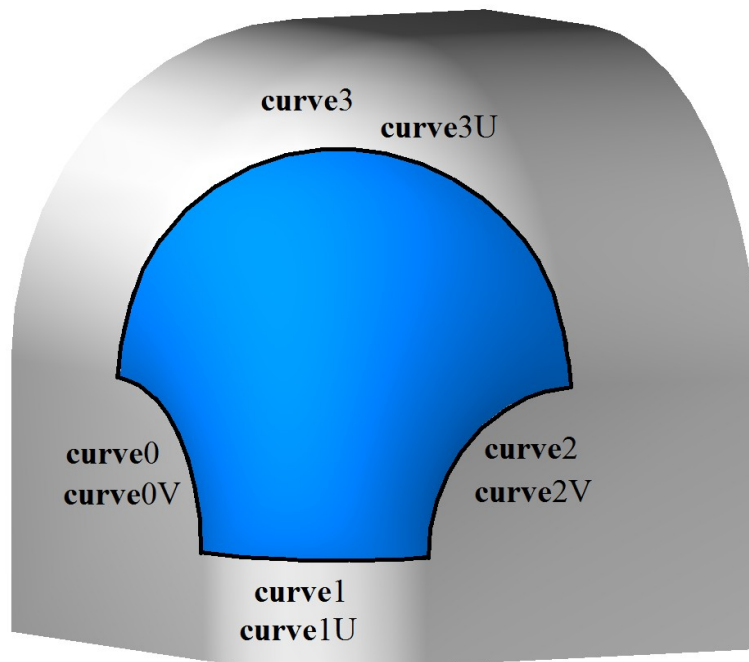


Рис. О.5.19.1.

Форма поверхности зависит от формы кривых и производных. Если соседние кривые пересекаются в точках **vertex[4]**, то поверхность проходит по участкам кривых: при $u=0$ поверхность проходит по участку кривой **curve3**, при $v=0$ поверхность проходит по участку кривой **curve0**, при $u=1$ поверхность проходит по участку кривой **curve1**, при $v=1$ поверхность проходит по участку кривой **curve2**.

О.5.20. Поверхность на сети кривых MbMeshSurface

Класс MbMeshSurface объявлен в файле surf_mesh_surface.h.

Поверхность на сети кривых MbMeshSurface описывается множеством кривых `RPArray<MbCurve3D>uCurves`, множеством кривых `RPArray<MbCurve3D>vCurves`, множеством значений первого параметра поверхности `uParams`, множеством значений второго параметра поверхности `vParams`, граничными значениями первого параметра поверхности `umin` и `umax`, граничными значениями второго параметра поверхности `vmin` и `vmax`, признаками наличия полюса поверхности при граничных значениях первого параметра `poleUMin`, `poleUMax`, признаками наличия полюса поверхности при граничных значениях второго параметра `poleVMin`, `poleVMax`, признаком периодичности по первому параметру поверхности `uclosed`, признаком периодичности по второму параметру поверхности `vclosed`, типом `type0` сопряжения поверхности на краю, соответствующем второму параметру `vmin`, типом `type1` сопряжения поверхности на краю, соответствующем первому параметру `umin`, типом `type2` сопряжения поверхности на краю, соответствующем второму параметру `vmax`, типом `type3` сопряжения поверхности на краю, соответствующем первому параметру `umax`. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Количество элементов множества кривых `vCurves` и множества значений первого параметра поверхности `uParams` согласованы так, что точкам кривой `vCurves[j]` соответствует параметр `uParams[j]`. Первый параметр поверхности `u` принимает значения на отрезке `uParams[0] ≤ u ≤ uParams[uParams.MaxIndex()]`. Поверхность может быть периодической по первому параметру, если периодическими являются все кривые `uCurves`.

Количество элементов множества кривых `uCurves` и множества значений второго параметра поверхности `vParams` согласованы так, что точкам кривой `uCurves[i]` соответствует параметр `vParams[i]`. Второй параметр поверхности `v` принимает значения на отрезке

$vParams[0] \leq v \leq vParams[vParams.MaxIndex()]$. Поверхность может быть периодической по второму параметру, если периодическими являются все кривые **vCurves**.

Каждая кривая **uCurves**[*i*] должна иметь точки пересечения или точки скрещения с каждой кривой **vCurves**[*j*]. Соседние кривые множества **uCurves** не должны иметь противоположные направления. Соседние кривые множества **vCurves** также не должны иметь противоположные направления.

В методе **PointOn**(double *u*, double *v*, **MbCartPoint3D** & **s**) радиус-вектор поверхности **s** описывается векторной функцией, которая приведена в книге «Геометрическое моделирование», автор Голованов Н.Н. Поверхность на сети кривых приведена на рис. О.5.20.1.

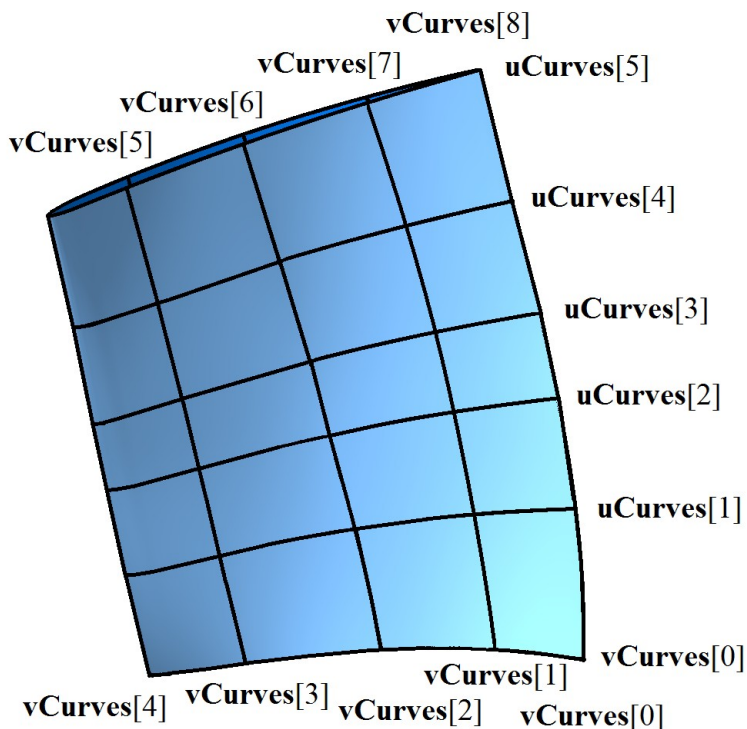


Рис. О.5.20.1.

Форма поверхности зависит от формы кривых, их взаимного расположения и значений параметров множеств **uParams** и **vParams**. Если каждая кривая **uCurves**[*i*] пересекается с каждой кривой **vCurves**[*j*], то поверхность проходит по кривым **vCurves**[*j*] при параметрах $u=uParams[j]$ и по кривым **uCurves**[*i*] при параметрах $v=vParams[i]$.

О.5.21. Поверхность соединения **MbJoinSurface**

Класс **MbJoinSurface** объявлен в файле `surf_joint_surface.h`.

Поверхность соединения **MbJoinSurface** описывается множеством кривых `RPArray<MbCurve3D>curves`, узловым вектором *knots*, порядком сплайна *degree*, граничными значениями первого параметра поверхности *umin* и *umax*, признаком периодичности первого параметра поверхности *closedU*, признаком периодичности второго параметра поверхности *closedV*, признаком наличия полюса поверхности при граничных значениях первого параметра *isPoleUmin*, *isPoleUmax*, признаком наличия полюса поверхности при граничных значениях второго параметра *isPoleVmin*, *isPoleVmax*.

Кривые **curves** согласованы друг с другом: имеют одинаковое направление и одинаковую область определения параметра. Первый параметр поверхности *u* совпадает с параметром кривых **curves**, который является общим для них. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения кривых **curves**. Поверхность может быть периодической по первому параметру, если периодическими являются все кривые **curves**. Пусть узловым вектором *knots* содержит *knotsCount* элементов, а количество кривых в множестве **curves** равно *curvesCount*. Количество элементов в множествах связаны равенством $curvesCount + degree = knotsCount$.

Второй параметр поверхности v принимает значения на отрезке $vmin \leq v \leq vmax$, где $vmin = knots[degree-1]$, $vmax = knots[knotsCount-degree]$. Поверхность не может быть периодической по второму параметру.

В методе **PointOn**(double u , double v , **MbCartPoint3D** & s) радиус-вектор поверхности s описывается векторной функцией

$$s(u, v) = \frac{\sum_{j=0}^{curvesCount-1} N_j^{degree}(v) \mathbf{curves}[j](u)}{\sum_{j=0}^{curvesCount-1} N_j^{degree}(v)},$$

где $N_j^{degree}(v)$ – B-сплайны порядка $degree$ для j -ой кривой $\mathbf{curves}[j]$. Поверхность соединения приведена на рис. O.5.21.1.

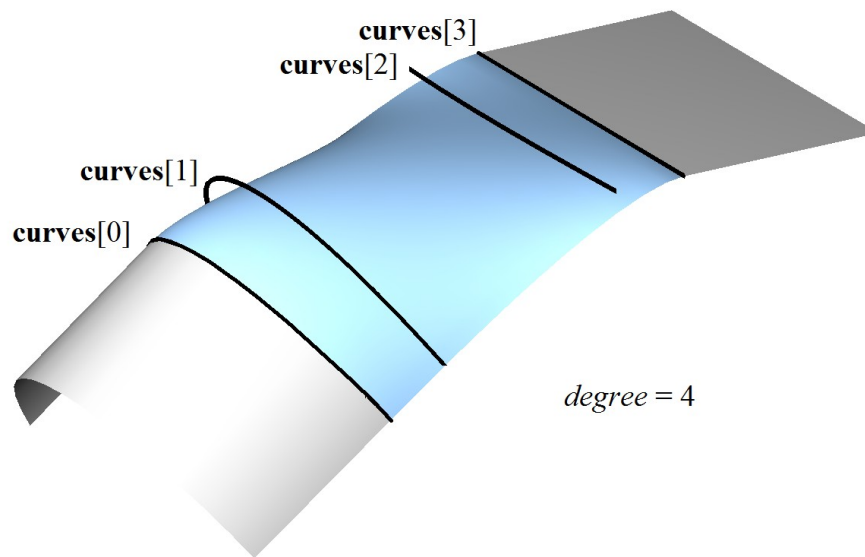


Рис. O.5.21.1.

Каждая кривая $s(const, v)$ с фиксированным первым параметром поверхности $u = const$ является NURBS-кривой порядка $degree$, построенной по точкам $\mathbf{curves}[i](const)$.

O.5.22. NURBS-поверхность MbSplineSurface

Класс MbSplineSurface объявлен в файле surf_spline_surface.h.

NURBS-поверхность (NonUniform Rational B-Spline поверхность) MbSplineSurface описывается контрольными точками `SArray<MbCartPoint3D>points[i][j]`, $i=0,1,\dots,vcount-1$, $j=0,1,\dots,ucount-1$, условно расположенными в узлах прямоугольной таблицы с $ucount$ колонками и $vcount$ строками, весами контрольных точек, заданных в таблице $weight[i][j]$, порядком B-сплайнов вдоль первого параметра поверхности $udegree$, порядком B-сплайнов вдоль второго параметра поверхности $vdegree$, узловым вектором вдоль первого параметра $uknots$, узловым вектором вдоль второго параметра $vknots$, признаками периодичности поверхности по первому и второму параметрам $uclosed$ и $vclosed$. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Порядок B-сплайнов вдоль параметров поверхности совпадает с порядком разделённой разности, по которой вычисляются соответствующие B-сплайны. Узловые векторы $uknots$ и $vknots$ представляют собой неубывающие последовательности действительных чисел и определяют область определения параметров поверхности и ее форму. Пусть узловой вектор $uknots$ содержит $uknotsCount$ элементов, а количество точек в каждой строке прямоугольной таблицы равно $ucount$. Для непериодической по первому параметру NURBS-поверхности количества элементов в множествах связаны равенством

$u_{count} + degree = uknotsCount$. Для периодической NURBS-поверхности количество элементов в множествах связаны равенством $knotsCount + 2degree - 1 = uknotsCount$.

В методе **PointOn**(double u , double v , **MbCartPoint3D** & s) радиус-вектор поверхности s описывается векторной функцией

$$\mathbf{r}(u, v) = \frac{\sum_{i=0}^{v_{count}-1} \sum_{j=0}^{u_{count}-1} N_i^{vdegree}(v) N_j^{udegree}(u) weight[i][j] \mathbf{points}[i][j]}{\sum_{i=0}^{v_{count}-1} \sum_{j=0}^{u_{count}-1} N_i^{vdegree}(v) N_j^{udegree}(u) weight[i][j]},$$

где $N_i^{vdegree}(v)$ и $N_j^{udegree}(u)$ – B-сплайны.

На рис. O.5.22.1 приведены отрезки, связывающие соседние контрольные точки $\mathbf{points}[i][j]$.

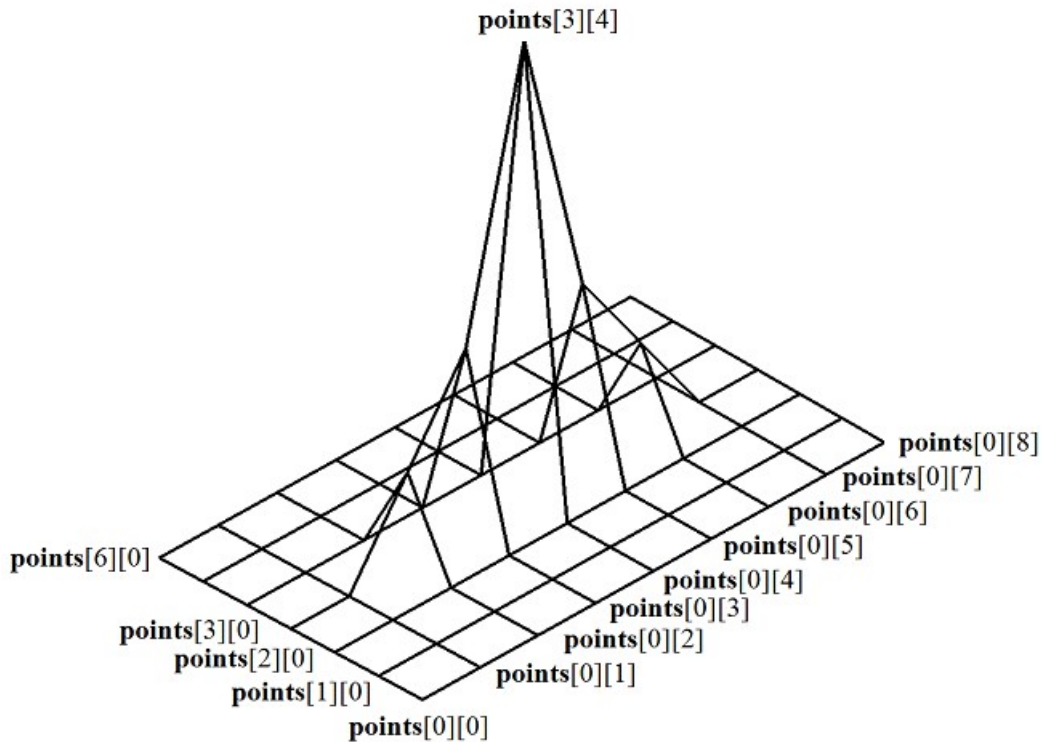


Рис. O.5.22.1.

Рис. O.5.22.1 демонстрирует условное расположение контрольных точек в узлах прямоугольной таблицы. NURBS-поверхность, построенная на тех же контрольных точках, приведена на рис. O.5.22.2.

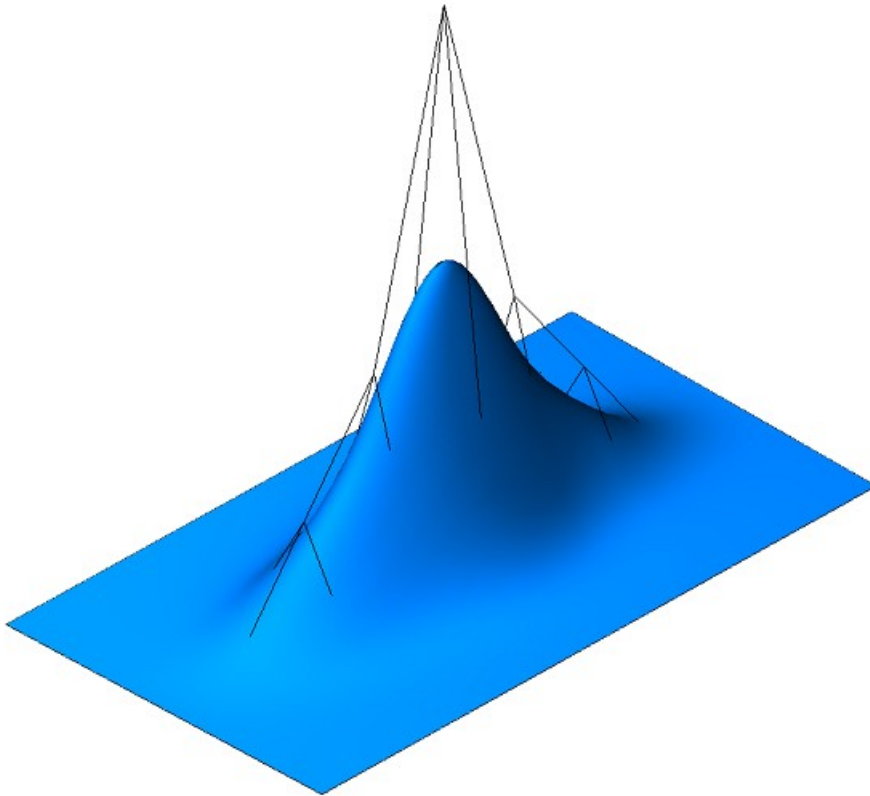


Рис. О.5.22.2.

Кривые на поверхности $s(const, v)$ и $s(u, const)$ с параметрами $u=const$ или $v=const$ представляют собой B -кривые степени $udegree$ и $vdegree$, соответственно. По первому параметру поверхность имеет порядок $udegree$, а по второму параметру поверхность имеет порядок $vdegree$. Область изменения параметров непериодической NURBS-поверхности представляет собой прямоугольник: $uknots[udegree-1] \leq u \leq uknots[ucount]$, $vknots[vdegree-1] \leq v \leq vknots[vcount]$.

Поверхность может быть периодической как по первому параметру, так и по второму параметру. Для периодически замкнутой поверхности узловые векторы $uknots$ и $vknots$ имеют на $udegree-1$ и $vdegree-1$ элементов, соответственно, больше. Область изменения параметров периодической по обоим параметрам NURBS-поверхности представляет собой прямоугольник: $uknots[udegree-1] \leq u \leq uknots[ucount+udegree-1]$, $vknots[vdegree-1] \leq v \leq vknots[vcount+vdegree-1]$.

Каждая поверхность может построить свою NURBS-копию виртуальным методом [NurbsSurface](#)(const MbNurbsParameters & $uParam$, const MbNurbsParameters & $vParam$).

О.5.23. Эквидистантная поверхность MbOffsetSurface

Класс MbOffsetSurface объявлен в файле surf_offset_surface.h.

Эквидистантная поверхность MbOffsetSurface описывается базовой поверхностью [MbSurface*](#) **basisSurface**, смещением вдоль нормали базовой поверхности $distance$, граничными значениями первого параметра базовой поверхности $u0min$, $u0max$, граничными значениями второго параметра базовой поверхности $v0min$, $v0max$, признаками периодичности базовой поверхности $u0closed$, $v0closed$, увеличениями граничных значений первого параметра базовой поверхности $dumin$, $dumax$, увеличениями граничных значений второго параметра базовой поверхности $dvmin$, $dvmax$. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Вычисление радиуса-вектора точки эквидистантной поверхности выполняется следующим образом. Для заданного параметра вычисляется точка базовой поверхности и нормаль в этой точке.

В методе [PointOn](#)(double u , double v , [MbCartPoint3D](#) & s) радиус-вектор поверхности s описывается векторной функцией

$$\mathbf{r}(u,v) = \mathbf{basisSurface}(u,v) + \mathbf{normal}(u,v) \cdot distance,$$

где $\mathbf{normal}(u,v)$ – нормаль поверхности в заданной точке. Эквидистантная поверхность и её базовая поверхность приведены на рис. O.5.23.1.

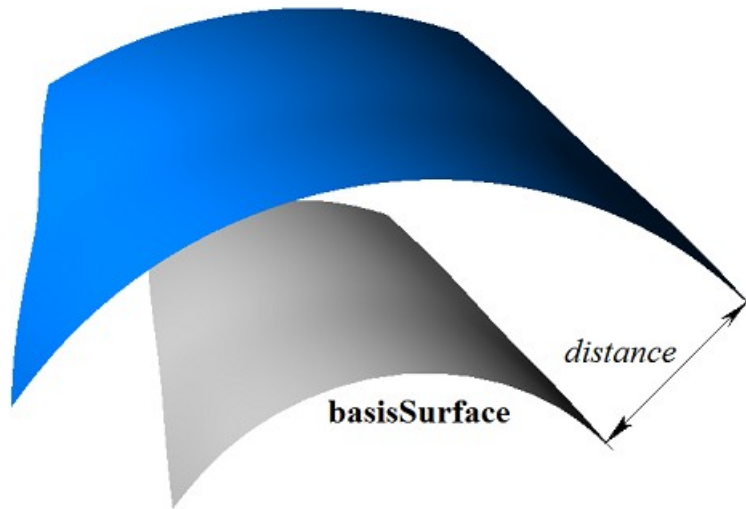


Рис. O.5.23.1.

Область изменения параметра эквидистантной поверхности может отличаться от области изменения параметров базовой поверхности. Область изменения первого параметра эквидистантной поверхности определяется неравенствами: $u0min+dumin \leq u \leq u0max+dumax$. Область изменения второго параметра эквидистантной поверхности определяется неравенствами: $v0min+dvmin \leq v \leq v0max+dvmax$. Эквидистантная поверхность с отрицательным смещением и увеличенной областью определения параметров и её базовая поверхность приведены на рис. O.5.23.2.

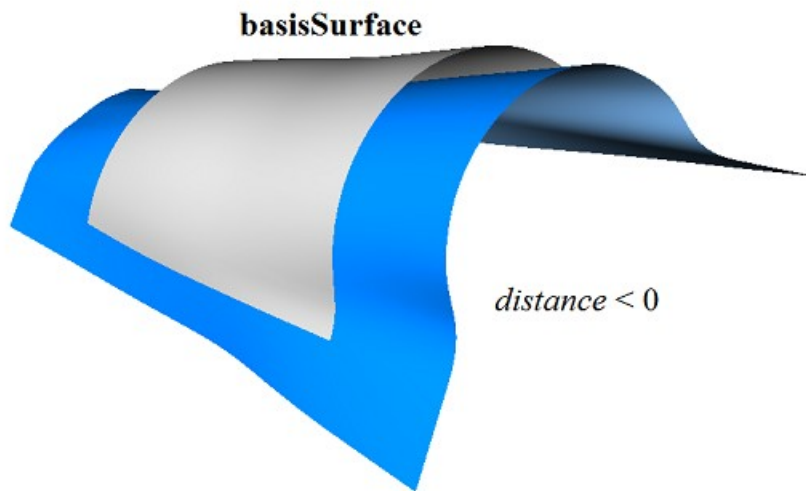


Рис. O.5.23.2.

В качестве базовой поверхности для эквидистантной поверхности не должна использоваться другая эквидистантная поверхность, а должна использоваться базовая поверхность последней с соответствующим пересчетом величины смещения.

Каждая поверхность может построить эквидистантную поверхность виртуальным методом **Offset**(double *distance*, bool *sense*).

О.5.24. Поверхность фаски MbChamferSurface

Класс MbChamferSurface объявлен в файле surf_chamfer_surface.h.

Поверхность фаски MbChamferSurface принадлежит к группе поверхностей сопряжения MbSmoothSurface. Поверхность фаски описывается кривой на первой сопрягаемой поверхности [MbSurfaceCurve*](#) **curve1**, кривой на второй сопрягаемой поверхности [MbSurfaceCurve*](#) **curve2**, способом построения фаски *form*, катетами фаски *distance1* и *distance2*, граничными значениями *umin* и *umax* параметра кривых **curve1** и **curve2**, граничными значениями второго параметра поверхности *vmin* и *vmax*, признаком периодичности первого параметра поверхности *uclosed*, признаком наличия полюса поверхности при начальном значении первого параметра *poleMin*, признаком наличия полюса поверхности при конечном значении первого параметра *poleMax*.

Кривые **curve1** и **curve2** согласованы друг с другом и имеют одинаковую область определения параметра. Первый параметр поверхности *u* совпадает с параметром кривых **curve1** и **curve2**, который является общим для них. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения кривых **curve1** и **curve2**. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve1** и **curve2**.

Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=vmin$ соответствует точке на кривой **curve1**, значение $v=vmax$ соответствует точке на кривой **curve2**. Поверхность не может быть периодической по второму параметру.

В методе [PointOn](#)(double *u*, double *v*, [MbCartPoint3D](#) & *s*) радиус-вектор поверхности *s* описывается векторной функцией

$$s(u,v) = \mathbf{curve1}(u) (1-w) + \mathbf{curve2}(u) w,$$

где $w = \frac{v - vmin}{vmax - vmin}$.

Поверхность фаски приведена на рис. О.5.24.1.

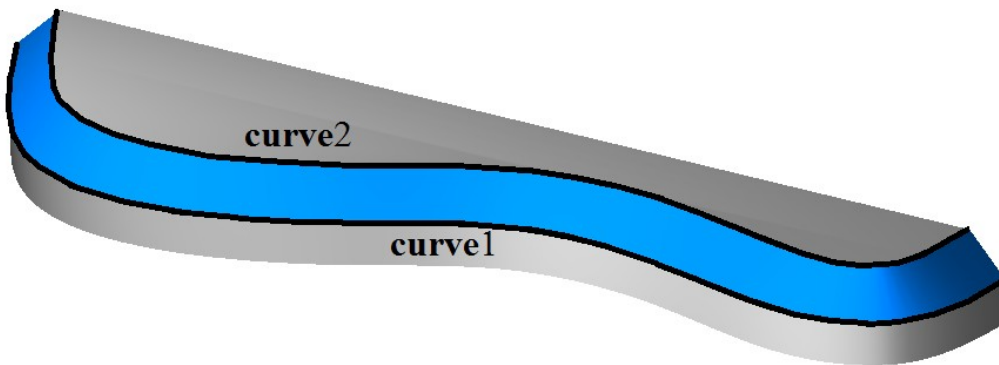


Рис. О.5.24.1.

Кривые поверхности $s(const,v)$ с параметрами $u=const$ являются отрезками прямой. Поверхность может иметь полюс при $u=umin$ и $u=umax$, если соответствующие края кривых **curve1** и **curve2** совпадают.

О.5.25. Поверхность скругления MbFilletSurface

Класс MbFilletSurface объявлен в файле surf_fillet_surface.h.

Поверхность скругления MbFilletSurface принадлежит к группе поверхностей сопряжения MbSmoothSurface. Поверхность скругления описывается кривой на первой сопрягаемой поверхности [MbSurfaceCurve*](#) **curve1**, кривой на второй сопрягаемой поверхности [MbSurfaceCurve*](#) **curve2**, кривой [MbCurve3D*](#) **curve0**, функцией [MbFunction*](#) **weights0** веса кривой **curve0**, способом

построения скругления *form*, радиусами скругления *distance1* и *distance2*, коэффициентом формы *conic*, граничными значениями *umin* и *umax* параметра кривых **curve1**, **curve2**, **curve0** и функции **weights0**, граничными значениями второго параметра поверхности *vmin* и *vmax*, признаком периодичности первого параметра поверхности *uclosed*, признаком наличия полюса поверхности при начальном значении первого параметра *poleMin*, признаком наличия полюса поверхности при конечном значении первого параметра *poleMax*, признаком равномерной параметризации поверхности по второму параметру *even*, признаком гладкого сопряжения поверхности с сопрягаемыми поверхностями *equable*, признаком *byCurve1* наличия кромки вдоль кривой **curve2** или **curve1** (*equable=false*). У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Кривые **curve1**, **curve2**, **curve0** и функция **weights0** согласованы друг с другом и имеют одинаковую область определения параметра. Первый параметр поверхности *u* совпадает с параметром кривых **curve1**, **curve2**, **curve0** и функции **weights0**, который является общим для них. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения кривых **curve1**, **curve2**, **curve0** и функции **weights0**. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve1**, **curve2**, **curve0** и функция **weights0**.

Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=vmin$ соответствует точке на кривой **curve1**, значение $v=vmax$ соответствует точке на кривой **curve2**. Поверхность не может быть периодической по второму параметру.

В методе **PointOn**(double *u*, double *v*, **MbCartPoint3D** & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u, v) = \frac{(1-v)^2 \mathbf{curve1}(u) + 2v(1-v) \mathbf{weights0}(u) \mathbf{curve0}(u) + v^2 \mathbf{curve2}(u)}{(1-v)^2 + 2v(1-v) \mathbf{weights0}(u) + v^2}.$$

Каждая кривая $\mathbf{s}(const, v)$ с фиксированным первым параметром поверхности $u=const$ является NURBS-кривой третьего порядка, построенной по точкам **curve1**(*u*), **curve0**(*u*), **curve2**(*u*), у которой крайние точки имеют единичный вес, а средняя точка **curve0**(*u*) имеет вес **weights0**(*u*). При *conic=_ARC_* функция **weights0**(*u*) вычисляется из условия, что все кривые $\mathbf{s}(const, v)$ являются дугами окружности. При *conic≠_ARC_* функция **weights0** является константой и равна коэффициенту формы *conic*. Поверхность скругления приведена на рис. O.5.25.1.

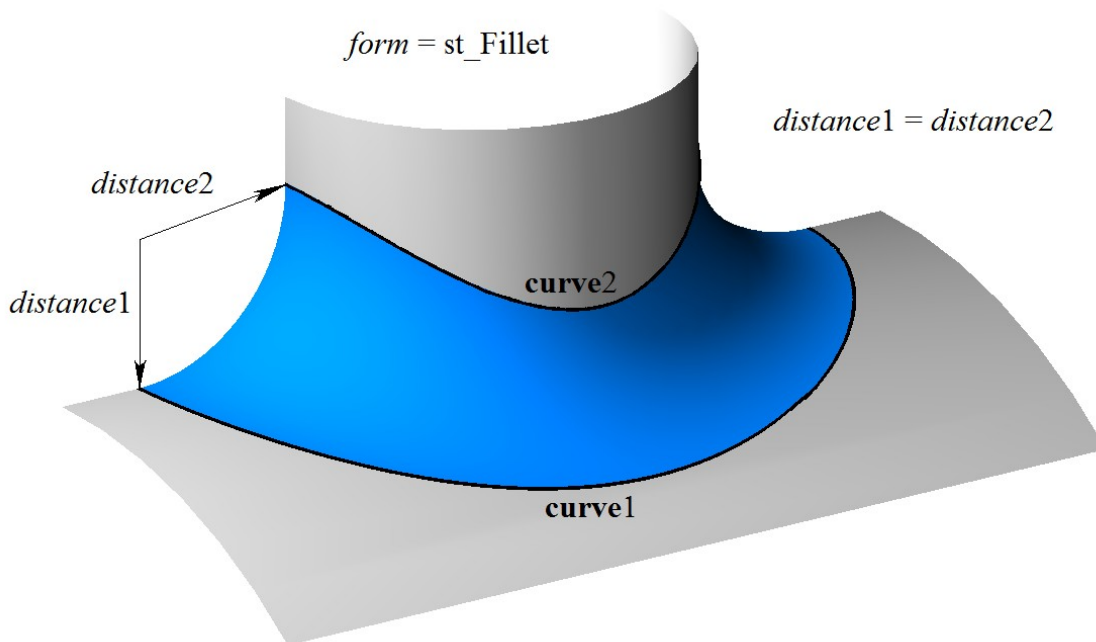


Рис. O.5.25.1.

В общем случае поверхность гладко сопрягается с поверхностями, на которых расположены кривые **curve1**(*u*) и **curve2**(*u*). В этом случае параметр *equable=true*. Если *equable=false*, то по одной

из кривых **curve1** или **curve2** сопряжение является гладким, а другая кривая является кромкой. При $byCurve1=true$ сопряжение является гладким по кривой **curve1**, в противном случае сопряжение является гладким по кривой **curve2**. Поверхность скругления с сохранением кромки приведена на рис. O.5.25.2.

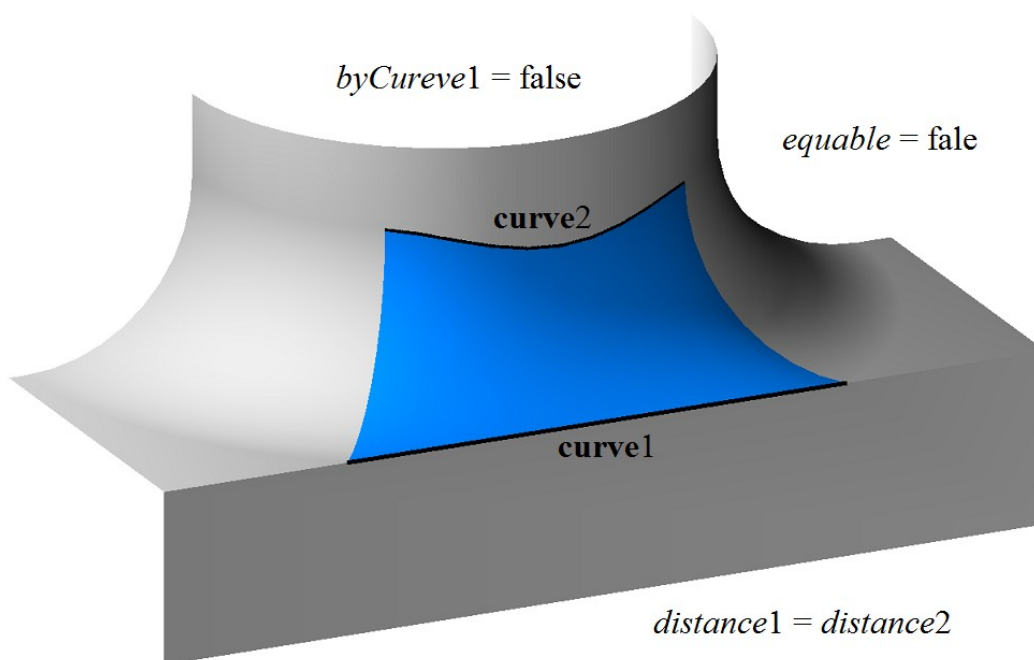


Рис. O.5.25.2.

Если радиусы скругления $distance1$ и $distance2$ не равны, то поверхность скругления в сечении $s(const,v)$ с параметрами $u=const$ описывает эллипс. Эллиптическая поверхность скругления приведена на рис. O.5.25.3.

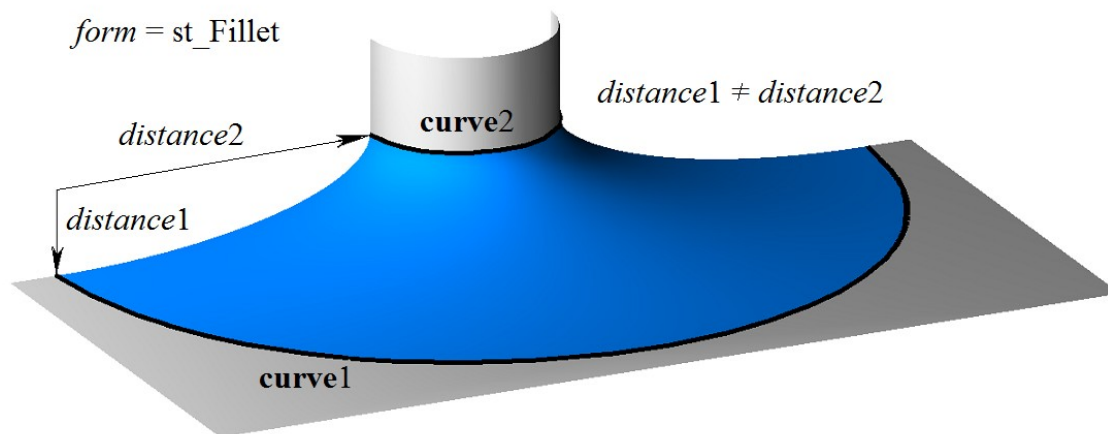


Рис. O.5.25.3.

В общем случае способ скругления $form=st_Fillet$. Если способ скругления $form=st_Span$, то $distance1=distance2$ и равны расстоянию между кривыми **curve1** и **curve2**, а радиус поверхности скругления является переменным. Поверхность скругления с сохранением расстояния между опорными кривыми приведена на рис. O.5.25.4.

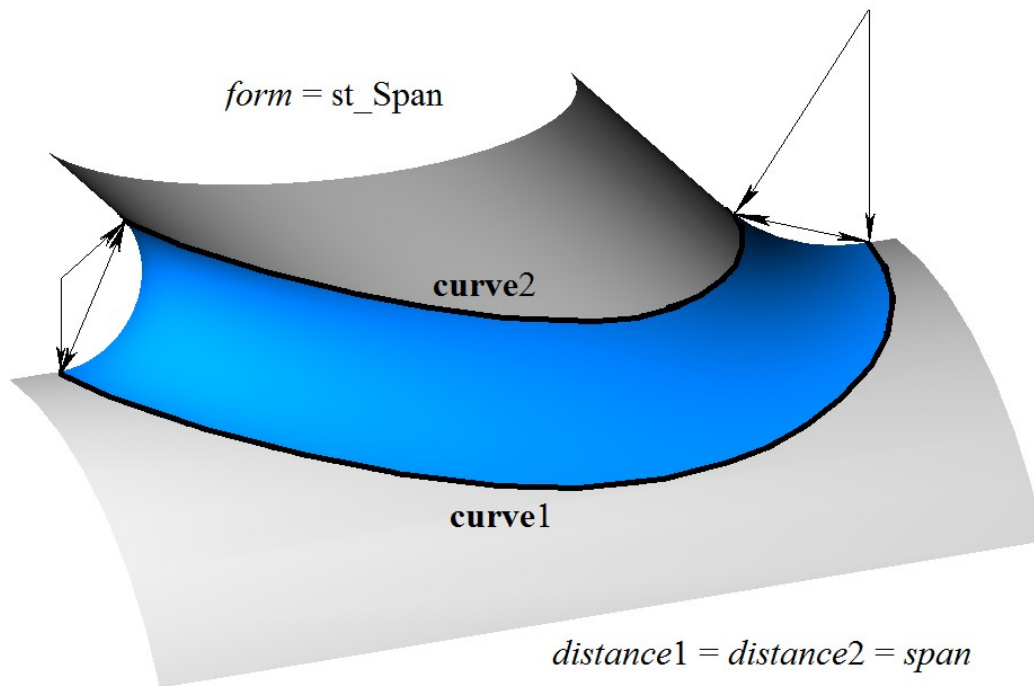


Рис. 0.5.25.4.

Кривые поверхности $s(const, v)$ с параметрами $u=const$ являются коническими сечениями, форма которых зависит от параметра $conic$. Если $conic=_ARC_=0$, то кривые поверхности $s(const, v)$ являются дугами окружности. Если $conic=0.5$, то кривые поверхности $s(const, v)$ являются дугами параболы. Если $0.05 < conic < 0.5$, то кривые поверхности $s(const, v)$ являются дугами эллипса. Если $0.5 < conic < 0.95$, то кривые поверхности $s(const, v)$ являются дугами гиперболы. Поверхность может иметь полюс при $u=umin$ и $u=umax$, если соответствующие края кривых **curve0**, **curve1** и **curve2** совпадают.

0.5.26. Поверхность скругления MbChannelSurface

Класс MbChannelSurface объявлен в файле surf_channel_surface.h.

Поверхность скругления с переменным радиусом MbChannelSurface является наследником поверхности скругления [MbFilletSurface](#). Поверхность скругления с переменным радиусом описывается кривой на первой сопрягаемой поверхности [MbSurfaceCurve*](#) **curve1**, кривой на второй сопрягаемой поверхности [MbSurfaceCurve*](#) **curve2**, кривой [MbCurve3D*](#) **curve0**, функцией [MbFunction*](#) **weights0** веса кривой **curve0**, функцией изменения радиуса [MbFunction*](#) **function**, способом построения скругления *form*, радиусами скругления *distance1* и *distance2*, коэффициентом формы *conic*, граничными значениями *umin* и *umax* параметра кривых **curve1**, **curve2**, **curve0** и функций **weights0** и **function**, граничными значениями второго параметра поверхности *vmin* и *vmax*, признаком периодичности первого параметра поверхности *uclosed*, признаком наличия полюса поверхности при начальном значении первого параметра *poleMin*, признаком наличия полюса поверхности при конечном значении первого параметра *poleMax*, У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Кривые **curve1**, **curve2**, **curve0** и функции **weights0**, **function** согласованы друг с другом и имеют одинаковую область определения параметра. Первый параметр поверхности *u* совпадает с параметром кривых **curve1**, **curve2**, **curve0** и функций **weights0** и **function**, который является общим для них. Первый параметр поверхности принимает значения на отрезке $umin \leq u \leq umax$, который соответствует области определения кривых **curve1**, **curve2**, **curve0** и функций **weights0** и **function**. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve1**, **curve2**, **curve0** и функции **weights0** и **function**.

Второй параметр поверхности *v* принимает значения на отрезке: $vmin \leq v \leq vmax$. Значение $v=vmin$ соответствует точке на кривой **curve1**, значение $v=vmax$ соответствует точке на кривой **curve2**. Поверхность не может быть периодической по второму параметру.

В методе **PointOn**(double *u*, double *v*, [MbCartPoint3D](#) & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u, v) = \frac{(1-v)^2 \mathbf{curve1}(u) + 2v(1-v) \mathbf{weight0}(u) \mathbf{curve0}(u) + v^2 \mathbf{curve2}(u)}{(1-v)^2 + 2v(1-v) \mathbf{weight0}(u) + v^2}$$

Каждая кривая $\mathbf{s}(const, v)$ с фиксированным первым параметром поверхности $u=const$ является NURBS-кривой третьего порядка, построенной по точкам $\mathbf{curve1}(u)$, $\mathbf{curve0}(u)$, $\mathbf{curve2}(u)$, у которой крайние точки имеют единичный вес, а средняя точка $\mathbf{curve0}(u)$ имеет вес $\mathbf{weights0}(u)$. При изменении первого параметра поверхности радиусы скругления меняются по законам $R1(u)=distance1 \mathbf{function}(u)$ и $R2(u)=distance2 \mathbf{function}(u)$. При $conic=_ARC_$ функция $\mathbf{weights0}(u)$ вычисляется из условия, что все кривые $\mathbf{s}(const, v)$ являются дугами окружности. При $conic \neq _ARC_$ функция $\mathbf{weights0}$ является константой и равна коэффициенту формы $conic$. Поверхность скругления с переменным радиусом приведена на рис. O.5.26.1.

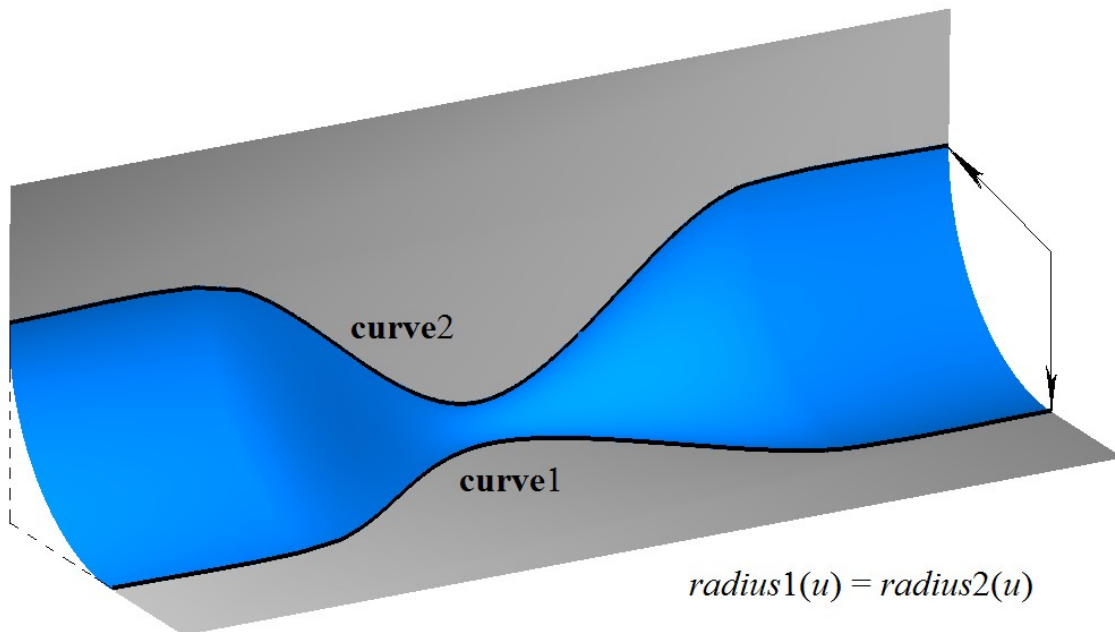


Рис. O.5.26.1.

Поверхность скругления с переменным радиусом гладко сопрягается с поверхностями, на которых расположены кривые $\mathbf{curve1}(u)$ и $\mathbf{curve2}(u)$, при этом параметр $equable=true$ и $form=st_Fillet$.

Кривые поверхности $\mathbf{s}(const, v)$ с параметрами $u=const$ являются коническими сечениями, форма которых зависит от параметра $conic$. Если $conic=_ARC_=0$, то кривые поверхности $\mathbf{s}(const, v)$ являются дугами окружности. Если $conic=0.5$, то кривые поверхности $\mathbf{s}(const, v)$ являются дугами параболы. Если $0.05 < conic < 0.5$, то кривые поверхности $\mathbf{s}(const, v)$ являются дугами эллипса. Если $0.5 < conic < 0.95$, то кривые поверхности $\mathbf{s}(const, v)$ являются дугами гиперболы. Поверхность может иметь полюс при $u=umin$ и $u=umax$, если соответствующие края кривых $\mathbf{curve0}$, $\mathbf{curve1}$ и $\mathbf{curve2}$ совпадают.

O.5.27. Поверхность с произвольными границами MbCurveBoundedSurface

Класс MbCurveBoundedSurface объявлен в файле surf_curve_bounded_surface.h.

Поверхность MbCurveBoundedSurface описывается базовой поверхностью [MbSurface*](#) **basisSurface**, множеством граничных кривых `RPAArray<MbContourOnSurface>curves` (контуров на поверхности) и граничными значениями параметров $umin$, $umax$, $vmin$, $vmax$, определяющими габаритный прямоугольник области определения параметров.

Поверхность MbCurveBoundedSurface имеет криволинейные края и может иметь произвольные вырезы внутри. Границы поверхности описывают контуры на поверхности контейнера **curves**.

В методе **PointOn**(double u , double v , [MbCartPoint3D](#) & \mathbf{s}) радиус-вектор поверхности \mathbf{s} описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{basisSurface}(u,v), \quad u,v \in \Omega,$$

где Ω – область определения параметров, которая представляет собой двумерную связную область. Радиус-вектор ограниченной контурами поверхности описывается тем же законом, что и поверхность **basisSurface**, но имеет другую область определения параметров. Базовая поверхность и контуры на ней приведены на рис. O.5.27.1

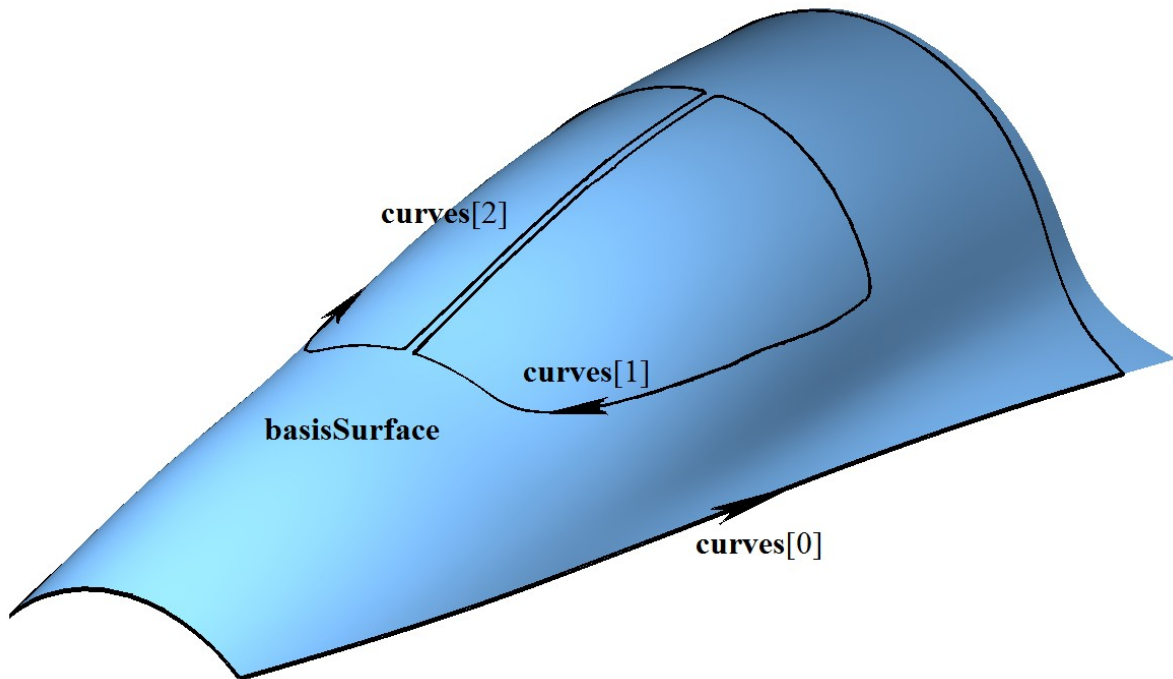


Рис. O.5.27.1

В общем случае область определения параметров Ω ограниченной контурами поверхности может выходить за область определения параметров поверхности **basisSurface**. Вне области определения параметров поверхности **basisSurface** радиус-вектор $\mathbf{r}(u,v)$ вычисляется методом **_PointOn**(u,v,\mathbf{s}) в соответствии с правилами продолжения поверхности **basisSurface**. Поверхность с произвольной границей приведена на рис. O.5.27.2.

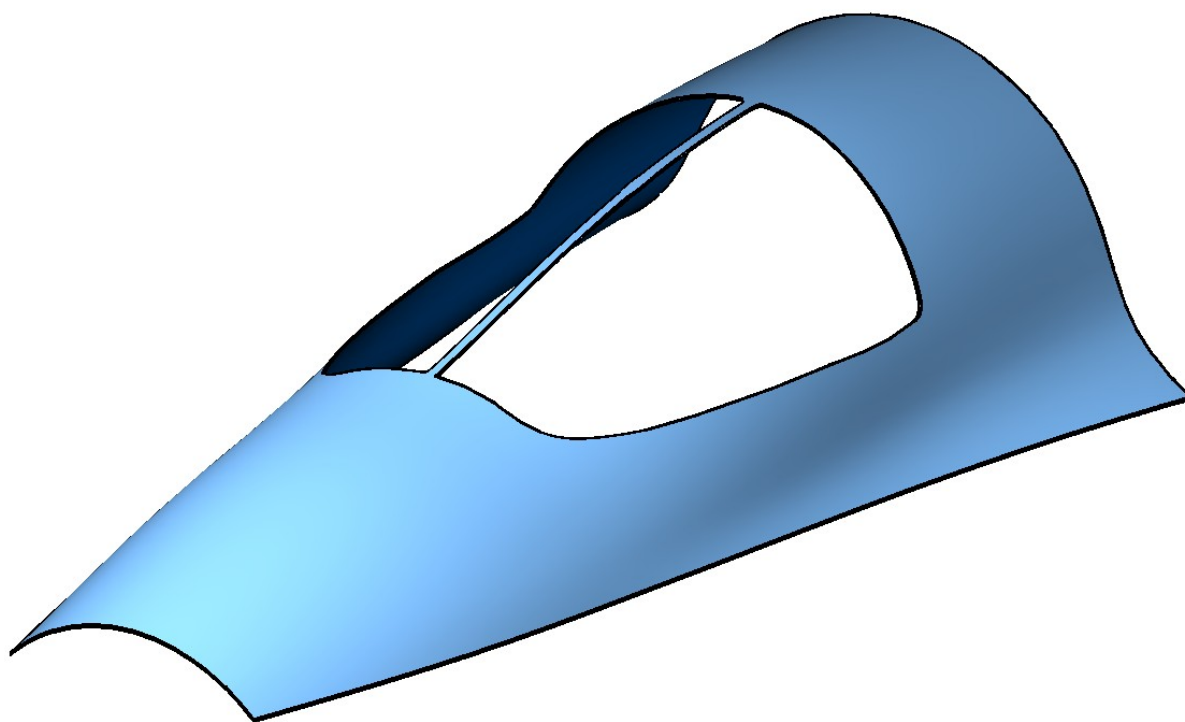


Рис. O.5.27.2.

Каждая кривая контейнера **curves** описывает одну замкнутую границу поверхности. Каждая кривая контейнера **curves** представляет собой контур на поверхности [MbContourOnSurface](#). Каждый контур на поверхности описывается поверхностью **surface**, совпадающей с **basisSurface**, и двумерным контуром **contour**. Контур **contour** представляет собой составную замкнутую двумерную кривую. Сегментами контура **contour** могут быть любые двумерные кривые [MbCurve](#) за исключением составных кривых [MbContour](#). В общем случае в местах стыковки сегментов производные контура терпят разрыв по длине и направлению. Двумерные контуры описывают границы области определения Ω поверхности [MbCurveBoundedSurface](#). Граничные контуры контейнера **curves** удовлетворяют следующим условиям: они не пересекают сами себя и друг друга, первый контур **curves[0]** контейнера описывает внешнюю границу и содержит внутри все остальные контуры, которые описывают внутренние вырезы в поверхности. Внутренние контуры не могут быть вложены друг в друга. Для быстрого определения положения некоторой двумерной точки относительно области определения параметров поверхности граничные контуры ориентированы так, что при движении вдоль границы поверхность всегда находится слева, если смотреть навстречу нормали поверхности. Таким образом, внешний контур ориентирован так, что обход по нему осуществляется против движения часовой стрелки, если смотреть на поверхность навстречу её нормали, а внутренние контуры ориентированы в противоположном направлении.

Базовой поверхностью **basisSurface** не может быть поверхность с произвольными границами [MbCurveBoundedSurface](#). Если нужно построить поверхность с произвольными границами на основе другой поверхности с произвольными границами, то в качестве базовой поверхности следует использовать базовую поверхность последней.

0.6. СПЕЦИАЛЬНЫЕ ОБЪЕКТЫ

К специальным объектам относятся скалярные функции, представляющие собой аналоги кривых в одномерном пространстве. Специальные объекты используются для конкретно определенных целей, например, для описания изменения радиуса поверхности скругления в виде функции одного из параметров поверхности. В двумерном пространстве к специальным объектам относятся мультилиния и область. Для обслуживания мультилинии на базе двумерного контура создан контур с разрывами. В трёхмерном пространстве специальные объекты описывают базовые точки других объектов, резьбу, выносные линии, шероховатости и прочие условные обозначения.

0.6.1. Функция MbFunction

Класс MbFunction объявлен в файле function.h.

Функция MbFunction является абстрактным классом, наследником классов [MbRefItem](#) и TapeBase, рис. 0.6.1.1.

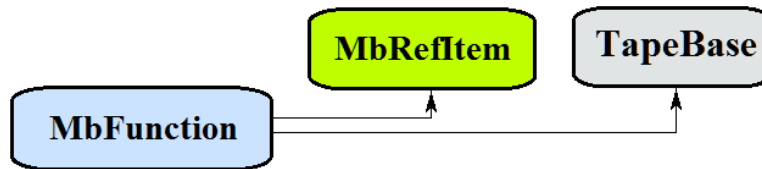


Рис. 0.6.1.1.

В геометрическом ядре C3D реализованы следующие скалярные функции, которые являются наследниками класса MbFunction:

[MbConstFunction](#) – константная функция,

[MbLineFunction](#) – линейная функция,

[MbCubicFunction](#) – кубическая функция Эрмита,

[MbCubicSplineFunction](#) – кубическая сплайн-функция,

[MdCharacterFunction](#) – аналитическая функция,

MbFunction представляет собой скалярную функцию

$$function(t) = f(t)$$

скалярного параметра t , принимающего значения на отрезке $[t_{\min}, t_{\max}]$. Область изменения параметра функции есть отрезок $[t_{\min}, t_{\max}]$ в одномерном пространстве. Функция $f(t)$ должна быть однозначной и непрерывной.

Граничные значения t_{\min} и t_{\max} области определения параметра выдают методы функции `double GetTMin()` и `double GetTMax()`, соответственно.

Функцию будем называть периодической, если существует $p > 0$, такое, что $f(t \pm kp) = f(t)$, где k – целое число. Метод `bool IsClosed()` периодической функции возвращает `true`. Метод `double GetPeriod()` периодической функции или функции, которая может быть расширена до периодической, выдает период p . Область определения параметра периодической функции всегда лежит в пределах одного периода.

Основным методом функции является метод `double Value(double & t)`.

Он выдаёт значение функции для заданного параметра t . Методы

`double FirstDer(double & t)`,

`double SecondDer(double & t)`,

`double ThirdDer(double & t)`

выдают соответственно первую, вторую и третью производные функции для заданного параметра t . Перечисленные методы корректируют параметр функции при его выходе за пределы области определения. При выходе параметра t за пределы отрезка $[t_{\min}, t_{\max}]$ непериодические функции

смещают параметр t к ближайшей границе t_{\min} или t_{\max} , а периодические функции добавляют или вычитают необходимое количество периодов.

Метод

double **_Value**(double t)

выдаёт значение функции для заданного параметра t как в области определения параметра функции, так и за её пределами. В общем случае непериодическая функции за пределами области определения параметра продолжается по касательной в крайней точке. Исключение составляют аналитические функции. Периодические функции за пределами области определения параметра продолжают циклически. Методы

double **_FirstDer**(double t),

double **_SecondDer**(double t),

double **_ThirdDer**(double t)

выдают соответственно первую, вторую и третью производные функции для заданного параметра t как в области определения функции, так и за её пределами.

Функции перегружают такие методы как:

методы, обеспечивающие копирование, проверку на совпадение, проверку на возможность сделать совпадающими, делающие объекты совпадающими,

MbFunction & **Duplicate**() ,

bool **IsSame**(const MbFunction & **item**) ,

bool **IsSimilar**(const MbFunction & **item**) ,

bool **SetEqual**(const MbFunction & **item**) ,

метод, возвращающий тип из перечисления функция,

MbFunctionType **IsA**() ,

методы, обеспечивающие выдачу и редактирование внутренних данных объекта,

MbProperty & **CreateProperty**(MbPrompt name) ,

void **GetProperties**(MbProperties & *properties*) ,

void **SetProperties**(MbProperties & *properties*) .

Все функции, как правило, не имеют изломов. Функция MbAnalyticalFunction, задаваемая пользователем, также должна быть непрерывной и не иметь особых точек.

О.6.2. Константная функция MbConstFunction

Класс MbConctFunction объявлен в файле func_const_function.h.

Константная функция MbConctFunction описывается одним значение функции *value*.

В методе double **Value**(double & t) работает функция

$$f(t) = value.$$

Область определения параметра функции располагается в пределах $0 \leq t \leq 1$. Функция не может быть периодической.

О.6.3. Линейная функция MbLineFunction

Класс MbLineFunction объявлен в файле func_line_function.h.

Линейная функция MbLineFunction описывается двумя граничными значениями функции *value1*, *value2* и граничными значениями параметра *tmin*, *tmax*.

В методе double **Value**(double & t) работает функция

$$f(t) = value1 (1-t) + value2 t .$$

Область определения параметра функции располагается в пределах $tmin \leq t \leq tmax$. Функция не может быть периодической.

О.6.4. Кубическая функция Эрмита MbCubicFunction

Класс MbCubicFunction объявлен в файле cur_cubic_function.h.

Кубическая функция Эрмита MbCubicFunction описывается множеством контрольных точек *valueList*, множеством производных функции в контрольных точках *firstList*, множеством значений параметра функции в контрольных точках *tList* и признаком периодичности функции *closed*. У функции есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов функции.

Кубическая функция Эрмита при значении параметра *tList[i]*, $i=0,1,\dots,splinesCount$, где $splinesCount=tList.MaxIndex()$, проходит через контрольную точку *valueList[i]* и имеет в ней производную *firstList[i]*. Функция построена на основе *splinesCount* сплайнов Эрмита третьей степени, которые гладко стыкуются между собой. Каждый кубический сплайн Эрмита описывает участок функции между двумя соседними контрольными точками. Каждый кубический сплайн Эрмита определяется двумя крайними точками и двумя производными кривой в этих точках.

При вычислении функции сначала по значению параметра *t* кривой определяется номер *i* рабочего участка (номер кубического сплайна Эрмита), из условия $tList[i] \leq t \leq tList[i+1]$. Функция вычисляется для найденного рабочего участка по его локальному параметру *w*, который определяется по *tList[i]* и *tList[i+1]*.

В методе double **Value**(double & *t*) работает функция

$$f(t) = (1 - 3w^2 + 2w^3)valueList[i] + (3w^2 + 2w^3)valueList[i + 1] + \\ + ((w - 2w^2 + w^3)valueList[i] + (-w^2 + w^3)valueList[i + 1])(tList[i + 1] - tList[i]),$$

где $w = \frac{t - tList[i]}{tList[i + 1] - tList[i]}$ - локальный параметр рабочего участка $tList[i] \leq t \leq tList[i+1]$.

Кубическая функция Эрмита приведена на рис. О.6.4.1.

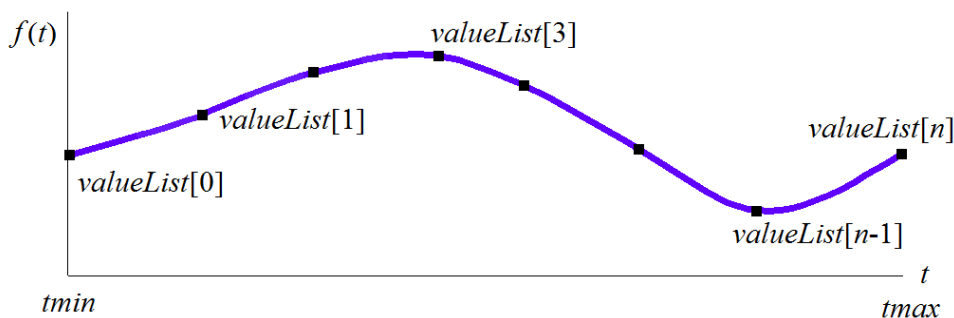


Рис. О.6.4.1.

Область определения параметра функции располагается в пределах $tmin \leq t \leq tmax$, где $tmin=tList[0]$, $tmax=tList[splinesCount]$. Функция может быть периодической.

Форма функции зависит от расположения контрольных точек, от производных функции в контрольных точках и от множества *tList* значений параметра в контрольных точках. При построении функции только по контрольным точкам производные *firstList[i]* вычисляются путём построения параболы, проходящей по трём соседним точкам $valueList[i-1]$, $valueList[i]$, $valueList[i+1]$ при соответствующих значениях параметра $tList[i-1]$, $tList[i]$, $tList[i+1]$, и вычисления производной параболы в средней точке.

О.6.5. Кубическая сплайн-функция MbCubicSplineFunction

Класс MbCubicSplineFunction объявлен в файле cur_cubic_spline_function.h.

Кубическая сплайн-функция MbCubicSplineFunction описывается множеством контрольных точек *valueList*, множеством вторых производных функции в контрольных точках *secondList*, множеством значений параметра функции в контрольных точках *tList* и признаком периодичности функции *closed*.

У функции есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов функции.

Кубическая сплайн-функция при значении параметра $tList[i]$, $i=0,1,\dots,splinesCount$, где $splinesCount=tList.MaxIndex()$, проходит через контрольную точку $valueList[i]$ и имеет в ней вторую производную $secondList[i]$.

При вычислении функции сначала по значению параметра t кривой определяется номер i рабочего участка, из условия $tList[i] \leq t \leq tList[i+1]$. Функция вычисляется для найденного рабочего участка по его локальному параметру w , который определяется по $tList[i]$ и $tList[i+1]$.

В методе `double Value(double & t)` работает функция

$$r(t) = (1 - w)valueList[i] + wvalueList[i + 1] + \\ + \left((-2w + 3w^2 - w^3)valueList[i] + (-w + w^3)valueList[i + 1] \right) \frac{(tList[i + 1] - tList[i])^2}{6},$$

где $w = \frac{t - tList[i]}{tList[i + 1] - tList[i]}$ - локальный параметр рабочего участка $tList[i] \leq t \leq tList[i+1]$.

Кубическая сплайн-функция приведена на рис. О.6.5.1.

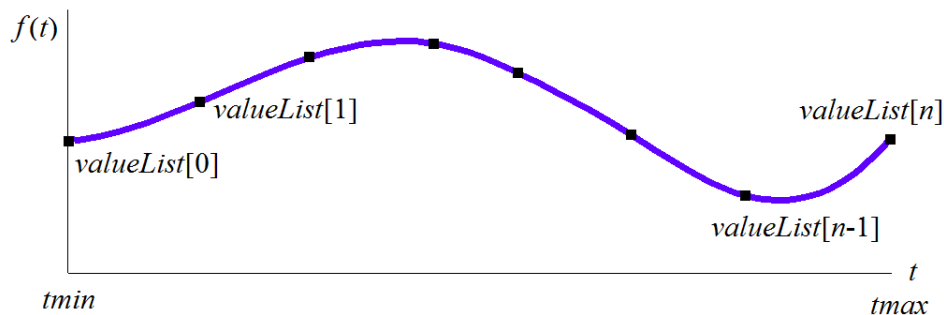


Рис. О.6.5.1.

Область определения параметра функции располагается в пределах $tmin \leq t \leq tmax$, где $tmin=tList[0]$, $tmax=tList[splinesCount]$. Функция может быть периодической.

Форма функции зависит от расположения контрольных точек и от множества $tList$ значений параметра в контрольных точках. При построении функции по контрольным точкам производные $secondList[i]$ вычисляются из условия линейного изменения вторых производных между контрольными точками.

О.6.6. Символьная функция MdCharacterFunction

Класс `MdCharacterFunction` объявлен в файле `func_analytical_function.h`.

Символьная функция `MdCharacterFunction` описывается строковым выражением функции, деревом действий для вычисления выражения, граничными значениями параметра $tmin$, $tmax$ и направлением $sense$. У функции есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов функции.

С помощью символьной функции можно описать любую функцию параметра t в виде строкового выражения, содержащего аналитические функции и арифметические операции.

В методе `double Value(double & t)` работает дерево действий для вычисления значения символьного выражения.

Область определения параметра функции располагается в пределах $tmin \leq t \leq tmax$. Функция может быть периодической.

О.6.7. Мультилиния MbMultiline

Класс MbMultiline объявлен в файле multiline.h.

Мультилиния MbMultiline является наследником класса [MbPlaneItem](#), рис. О.6.7.1.

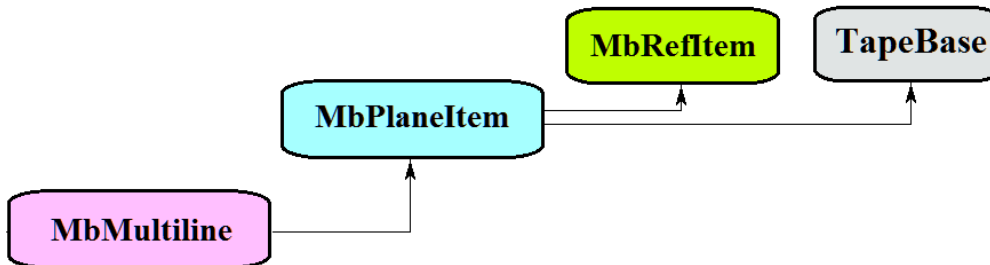


Рис. О.6.7.1.

Мультилиния описывается базовым контуром *basisCurve*, множеством вершин *vertices*, множеством радиусов *equidRadii*, параметрами начального края мультилинии *begTipParams*, параметрами конечного края мультилинии *endTipParams*, признаком обработки периодичности *processClosed*, признаком прозрачности *isTransparent*, множеством кривых *curves*, множеством законцовок в вершинах мультилинии *tipCurves*, законцовкой в начальной вершине мультилинии *begTipCurves*, законцовкой в конечной вершине мультилинии *endTipCurves*,

Мультилиния представляет собой контур, имеющий толщину. Толщина контура переменная. Края контура и места соединения сегментов контура могут иметь различную форму.

Мультилиния приведена на рис. О.6.7.2.

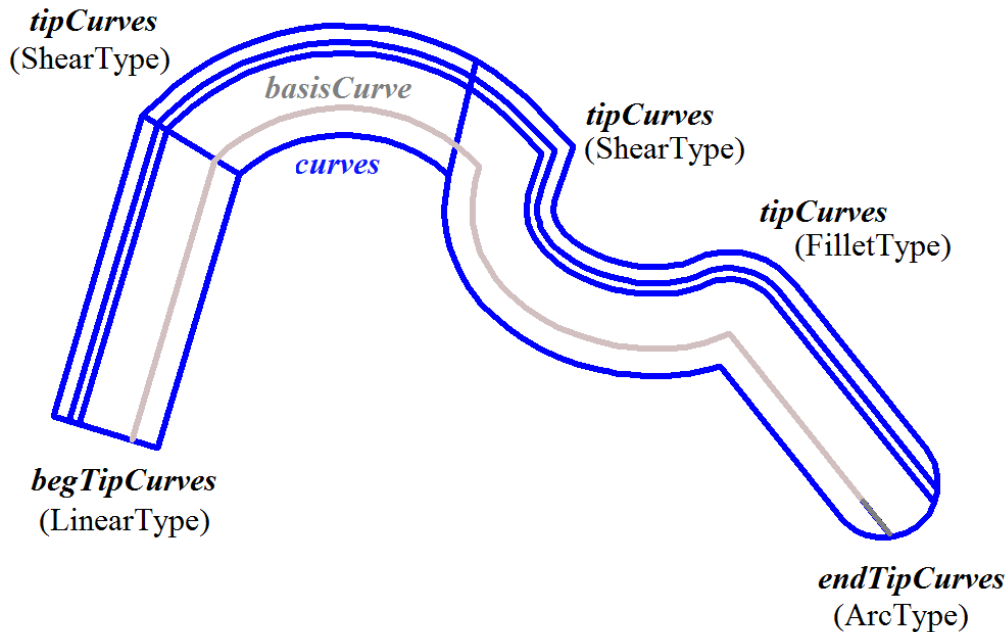


Рис. О.6.7.2.

Мультилиния используется для обмена данными с другими системами.

О.6.8. Двумерный контур с разрывами MbContourWithBreaks

Класс MbContourWithBreaks объявлен в файле cur_contour_with_breaks.h.

Двумерный контур с разрывами является наследником двумерного контура [MbContour](#), рис. О.6.8.1.

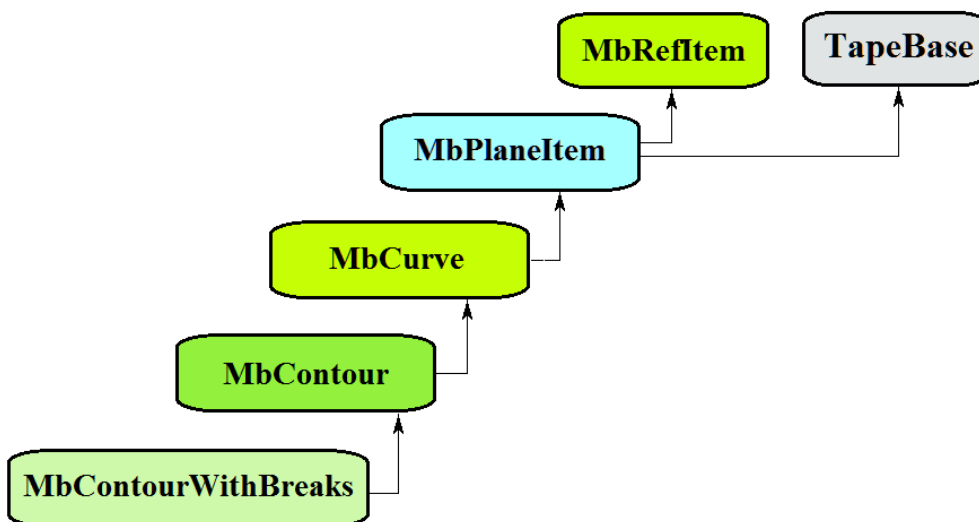


Рис. О.6.8.1.

Двумерный контур с разрывами описывается множеством *segments* стыкующихся друг да другом кривых, признаком периодичности кривой *closed*, разрывами *breaks*, множеством видимых участков контура *visibleContours* и множеством номеров сегментов контура *baseSegNumbers* для задания неподвижных точек разрыва.

Контур с разрывами приведён на рис. О.6.8.2.

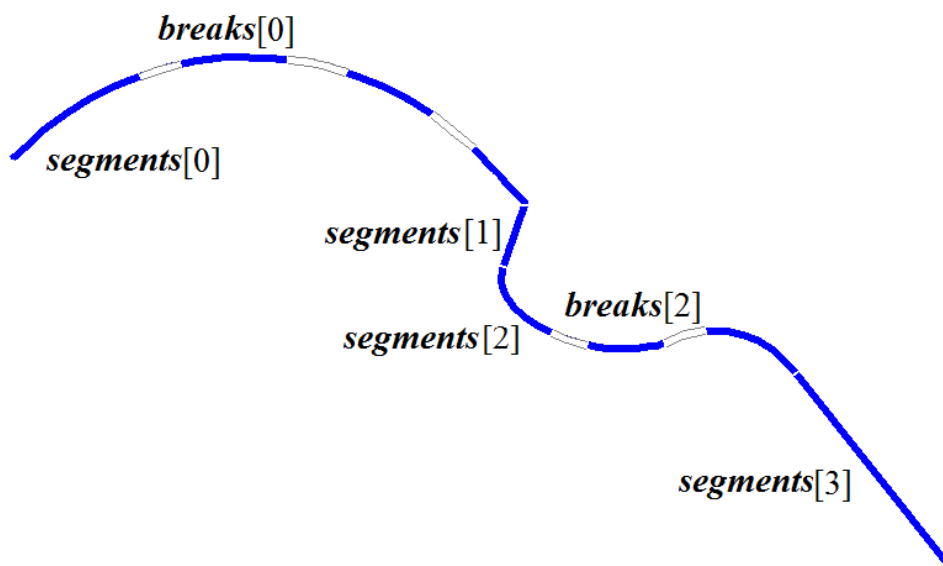


Рис. О.6.8.2.

В качестве сегментов двумерного контура с разрывами не должны использоваться другие двумерные контуры.

Двумерный контур с разрывами используется только для построения мультилиний.

О.6.9. Регион MbRegion

Класс MbRegion объявлен в файле region.h.

Регион MbRegion является наследником класса [MbPlaneItem](#), рис. О.6.9.1.

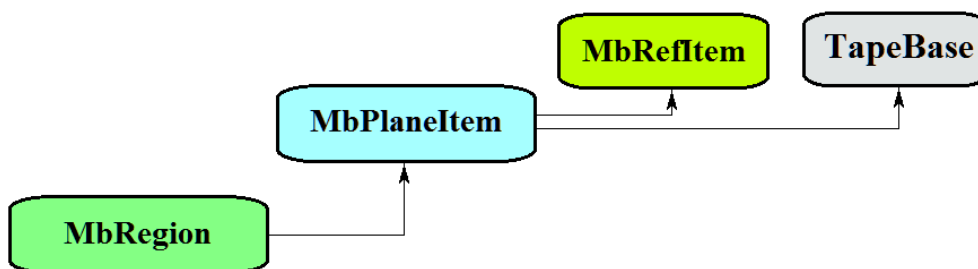


Рис. О.6.9.1.

Регион описывается множеством контуров *contours*. Контуров региона периодические и не пересекают сами себя и друг друга. Среди контуров множества один контур является внешним, а остальные являются внутренними и располагаются внутри внешнего контура. В множестве *contours* первым всегда лежит внешний контур.

Регион представляет собой связное множество точек двумерного пространства, границы которого описывают двумерные периодические контуры. Контуров региона ориентированы так, что при движении вдоль любого контура описываемое множество точек располагается слева от контура. То есть, внешний контур региона ориентирован против движения часовой стрелки, а внутренние контуры ориентированы по движению часовой стрелки.

Регион приведен на рис. О.6.9.2.

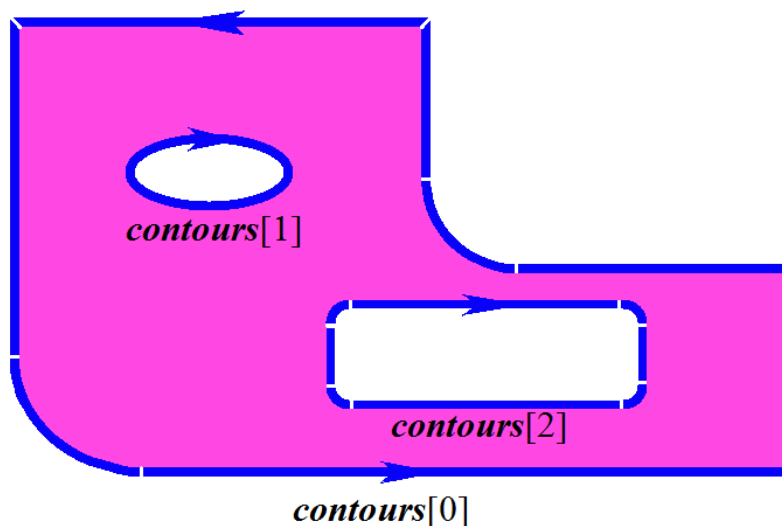


Рис. О.6.9.2.

Регион используется для описания двумерных связных областей. Над регионами можно выполнять булевы операции.

О.6.10. Вспомогательный геометрический объект MbLegend

Класс MbLegend объявлен в файле legend.h.

Вспомогательный объект MbLegend является наследником класса [MbSpaceItem](#), рис. О.6.10.1.

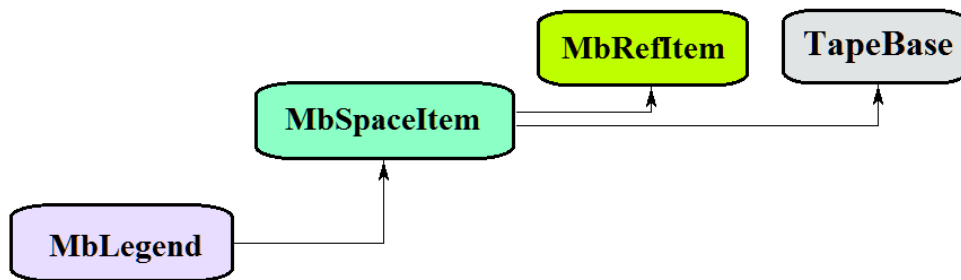


Рис. О.6.10.1.

Вспомогательный объект является абстрактным классом. В геометрическом ядре C3D реализованы следующие вспомогательные объекты, которые являются наследниками класса MbLegend:

- [MbMarker](#) – точка и два ортонормированных вектора,
- [MbThread](#) – резьба,
- [MbPointsSymbol](#) – условное обозначение на базовых точках,
- [MbRough](#) – обозначение шероховатости,
- [MbLeader](#) – обозначение линии выноски.

Вспомогательные объекты используются для различных целей, но объединяет их взаимодействие с кривыми, поверхностями и объектами геометрической модели.

Вспомогательные объекты перегружают такие методы трёхмерного геометрического объекта как:

- методы, обслуживающие преобразование геометрического объекта,
- [Move](#)(const [MbVector3D](#) & v, MbRegTransform * iReg = NULL),
- [Rotate](#)(const MbAxis3D & axis, double angle, MbRegTransform * iReg = NULL),
- [Transform](#)(const [MbMatrix3D](#) & m, MbRegTransform * iReg = NULL),
- методы, обеспечивающие копирование, проверку на совпадение, проверку на возможность сделать совпадающими, делающие объекты совпадающими,
- [MbSpaceItem](#)& [Duplicate](#)(MbRegDuplicate * iReg = NULL),
- bool [IsSame](#)(const [MbSpaceItem](#) & item),
- bool [IsSimilar](#)(const [MbSpaceItem](#) & item),
- bool [SetEqual](#)(const [MbSpaceItem](#) & item),
- методы, возвращающие тип из перечисления геометрических объектов,
- MbeSpaceType [IsA](#)(),
- MbeSpaceType [Type](#)(),
- MbeSpaceType [Family](#)(),
- методы, обеспечивающие выдачу и редактирование внутренних данных объекта,
- MbProperty & [CreateProperty](#)(MbePrompt name),
- [GetProperties](#)(MbProperties & properties),
- [SetProperties](#)(MbProperties & properties),
- метод, наполняющий полигональную копию геометрического объекта,
- [CalculateWire](#)(double sag, [MbMesh](#) & mesh).

О.6.11. Маркер MbMarker

Класс MbMarker объявлен в файле marker.h.

Вспомогательный объект маркер MbMarker описывается точкой **origin** и двумя ортогональными векторами **axisZ**, **axisX**.

Маркер используется для установки геометрических ограничений на объекты трехмерного пространства. Маркер является представителем геометрического объекта и может заменять трехмерную точку, прямую, плоскость, локальную систему координат и другие объекты при работе геометрических ограничений.

О.6.12. Условное обозначение резьбы MbThread

Класс MbThread объявлен в файле mb_thread.h.

Условное обозначение резьбы MbThread описывается локальной системой координат резьбы **place**, начальным радиусом резьбы на поверхности *radObj*, начальным радиусом резьбы в теле *radThr*, длиной резьбы *length*, углом конусности резьбы *angle*, именем резьбы **name**, множеством тел **bodies**. У объекта есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов объекта.

Ось резьбового соединения направлена вдоль оси **place.axisZ**. Имя служит для идентификации условного обозначения резьбы среди плоских проекций граней тела из множества **bodies**. Условное обозначение резьбы приведено на рис. О.6.12.1.

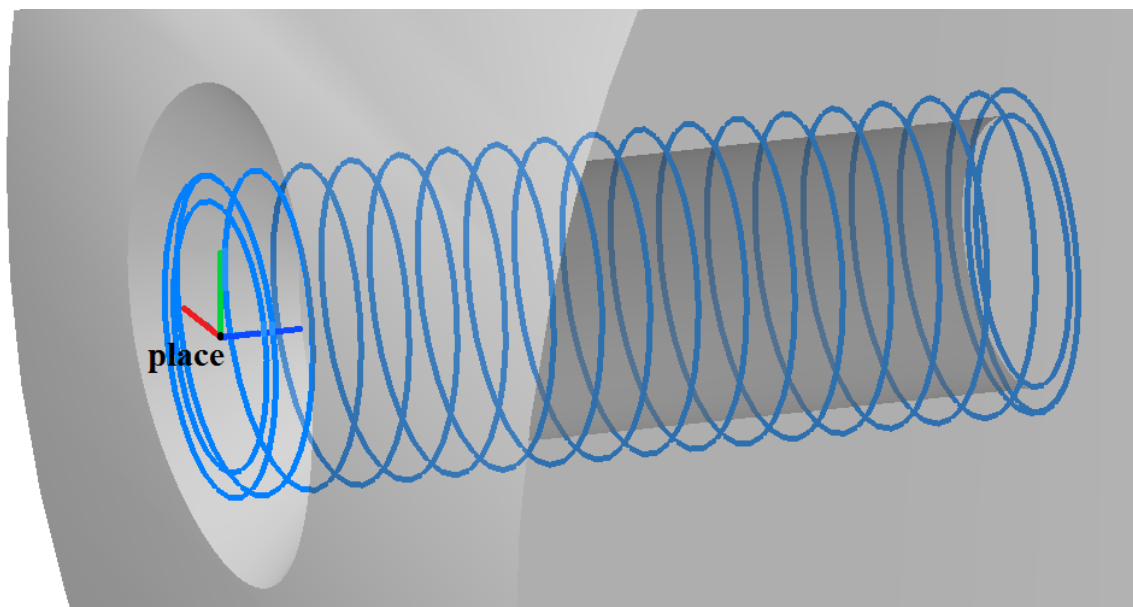


Рис. О.6.12.1.

Условное обозначение резьбы описывает элемент резьбового соединения геометрической модели и используется для построения плоских проекций резьбовых соединений.

О.6.13. Условное обозначение MbPointsSymbol

Класс MbPointsSymbol объявлен в файле mb_symbol.h.

Условное обозначение MbPointsSymbol является наследником класса MbSymbol. Условное обозначение MbPointsSymbol описывается набором идентификаторов, множеством трехмерных точек **points** и данными об участках сложных разрезов *steps*.

Объект несет информацию о базовых точках условных обозначений, связанных с элементами геометрической модели.

Объект используется при построении плоских проекций условных обозначений, для которых достаточно знать положение базовых точек. Он определяет базовые точки условных обозначений элементов геометрической модели.

О.6.14. Обозначение шероховатости MbRough

Обозначение шероховатости MbRough.

Класс MbRough объявлен в файле mb_rough.h.

Условное обозначение шероховатости MbRough является наследником класса [MbPointsSymbol](#). Условное обозначение шероховатости описывается множеством трехмерных точек **points**, данными об участках сложных разрезов *steps* и топологическим объектом привязки **item**.

Объект несет информацию о базовых точках условного обозначения шероховатости, связанных с топологическим объектом **item**.

Объект используется при построении плоских проекций условного обозначения шероховатости элемента геометрической модели **item**.

О.6.15. Условное обозначение линии выноски MbLeader

Класс MbLeader объявлен в файле mb_rough.h.

Условное обозначение линии выноски MbLeader является наследником класса MbSymbol. Условное обозначение линии выноски описывается набором идентификаторов и множеством обозначений шероховатости **branches**.

Объект несет информацию о линии выноски обозначения шероховатости для топологических объектов.

Объект используется при построении плоских проекций условного обозначения линии выноски шероховатости для элементов геометрической модели.

0.7. ТОПОЛОГИЧЕСКИЕ ОБЪЕКТЫ

Топологическими называют геометрические свойства, которые не зависят от количественных характеристик (длин и углов), а отражают непрерывную связь элементов объекта и его окружения. Топологические объекты геометрического ядра C3D описывают и геометрические свойства объекта, зависящие от количественных характеристик, и геометрические свойства, отражающие непрерывную связь объекта с соседними элементами. Топологические объекты строятся на основе поверхностей, кривых и точек путём добавления к их данным, свойствам и методам новых данных, свойств и методов, отражающих связь объекта с его окружением.

0.7.1. Топологический объект MbTopologyItem

Класс MbTopologyItem объявлен в файле topology_item.h.

Топологический объект MbTopologyItem отличается от других топологических объектов наличием имени **name**, признака изменения *changed* и метки *label*. Топологический объект MbTopologyItem является наследником топологического объекта [MbTopItem](#), рис. 0.7.1.1.

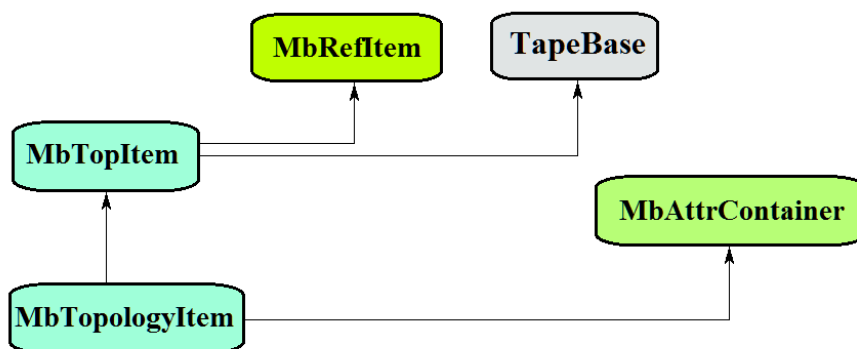


Рис. 0.7.1.1

Контейнер атрибутов MbAttrContainer наделяет именованные топологические объекты атрибутами.

В геометрическом ядре C3D реализованы следующие топологические объекты, которые являются наследниками класса MbTopologyItem:

[MbFace](#) – грань,

[MbEdge](#) – ребро,

[MbVertex](#) – вершина.

Ребро [MbEdge](#) имеет наследника [MbCurveEdge](#) – ребро стыковки граней.

Именованные топологические объекты имеют такие методы как:

методы, обслуживающие преобразование топологического объекта,

void **Move**(const [MbVector3D](#) & v, MbRegTransform * iReg = NULL),

void **Rotate**(const MbAxis3D & axis, double angle, MbRegTransform * iReg = NULL),

void **Transform**(const [MbMatrix3D](#) & m, MbRegTransform * iReg = NULL),

методы, обслуживающие работу с именем топологического объекта:

MbName & **GetName**(),

SimpleName & **GetMainName**(),

SimpleName & **GetFirstName**(),

метод, возвращающий тип из перечисления топологических объектов,

MbTopologyType **IsA**() .

Именованные топологические объекты используются в качестве элементов для построения объектов геометрической модели.

О.7.2. Грань MbFace

Класс MbFace объявлен в файле topology.h.

Грань MbFace является наследником топологического объекта [MbTopologyItem](#) и представляет собой конечный кусок поверхности, которому приписано направление нормали и определены границы. Сторону поверхности и грани, при взгляде на которую мы смотрим навстречу нормали, будем называть внешней, другую сторону будем называть внутренней. Стороны поверхности [MbSurface](#) не обладают равноправием относительно нормали, так как у поверхности одна сторона всегда внешняя, а другая сторона всегда внутренняя. В отличие от поверхности для грани мы имеем возможность назначить направление нормали и тем самым назначить внешнюю и внутреннюю стороны.

Структура данных грани содержит указатель на поверхность [MbSurface](#)* **surface**, признак совпадения направления нормали грани с направлением нормали поверхности **sameSense** и совокупность циклов грани `RPArray<MbLoop>` **loops**. У грани есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов грани.

Указатель на поверхность не может быть нулём. Признак совпадения направления нормали грани с направлением нормали поверхности принимает значение true, если нормали грани и поверхности совпадают, в противном случае признак принимает значение false. Сторону грани, при взгляде на которую мы смотрим навстречу нормали, называют внешней стороной, а другую сторону грани называют внутренней стороной.

Циклы грани описывают границы грани. Каждая граница грани замкнута. Каждый цикл описывается последовательностью рёбер [MbOrientedEdge](#) в порядке их следования вдоль границы. Количество циклов грани равно количеству границ поверхности грани. Одна из границ грани является внешней и содержит границы внутренних вырезов. Цикл, располагающийся первым по счёту в контейнере циклов, описывает внешнюю границу грани и содержит внутри себя внутренние циклы, которые описывают внутренние границы грани. Внешний цикл грани ориентирован против часовой стрелки, а внутренние циклы ориентированы по часовой стрелке, если смотреть навстречу нормали грани. Таким образом, при движении вдоль цикла грани по её внешней стороне грань всегда располагается слева. На рис. О.7.2.1 стрелки указывают направления циклов грани и её нормали.

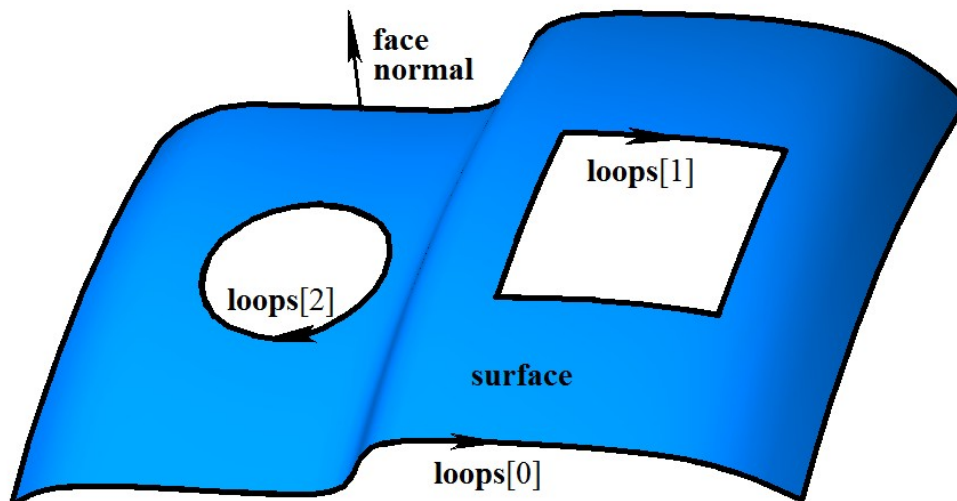


Рис. О.7.2.1.

Циклы одной грани не должны пересекать друг друга и сами себя.

Грань может быть именована и может иметь атрибуты. Имя может использоваться для идентификации грани, атрибуты могут нести дополнительную информацию о грани, например, цвет, прозрачность, происхождение и так далее.

Грань используется для твердотельного и гибридного моделирования.

О.7.3. Ребро MbEdge

Класс MbEdge объявлен в файле topology.h.

Ребро MbEdge является наследником топологического объекта [MbTopologyItem](#) и представляет собой кривую, которой приписано направление. Направление кривой [MbCurve3D](#) жёстко связано с направлением возрастания её параметра. В отличие от кривой ребро может быть направлено как в сторону возрастания параметра, так и в сторону уменьшения параметра кривой. Ребро всегда начинается и оканчивается в некоторой вершине [MbVertex](#).

Структура данных ребра содержит указатель на кривую [MbCurve3D](#)* **curve**, признак совпадения направления ребра с направлением кривой *sameSense*, указатель на начальную вершину [MbVertex](#)* **begVertex** и указатель на конечную вершину [MbVertex](#)* **endVertex**. На рис. О.7.3.1 приведено ребро.

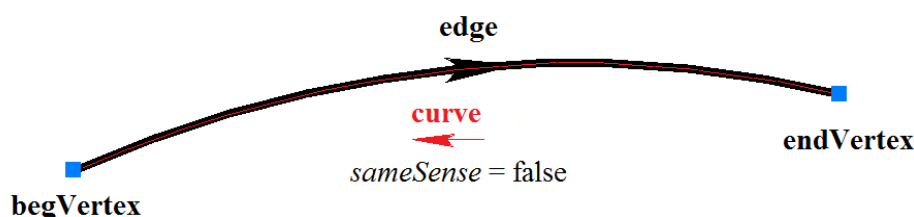


Рис. О.7.3.1.

Указатели на кривую и вершины не могут быть нулём. Признак совпадения направления ребра с направлением кривой принимает значение true, если ребро совпадает с кривой, и принимает значение false, если ребро направлено против кривой. Если ребро начинается и заканчивается в одной и той же вершине, то такое ребро является замкнутым. Указатели на начальную и конечную вершину замкнутого ребра равны.

Ребро может быть именовано и может иметь атрибуты. Имя может использоваться для идентификации ребра, атрибуты могут нести дополнительную информацию о ребре, например, цвет, стиль отображения, происхождение и так далее.

Ребро используется для каркасного моделирования.

О.7.4. Вершина MbVertex

Класс MbVertex объявлен в файле topology.h.

Вершина MbVertex является наследником топологического объекта [MbTopologyItem](#) и представляет собой точку с известной погрешностью расположения. Структура данных вершины содержит точку [MbCartPoint](#) **point** и погрешность *tolerance* расположения этой точки.

Вершина может описывать отдельную точку точечного каркаса или стык рёбер. В вершине может стыковаться любое конечное число рёбер. Стыкующиеся рёбра указывают на одну и ту же общую для них вершину. На рис. О.7.4.1 приведена вершина, в которой стыкуются три ребра.

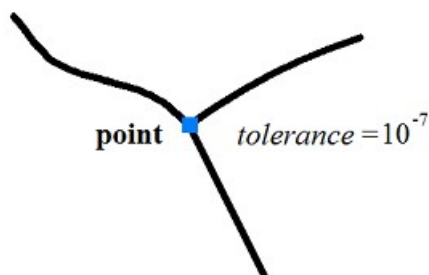


Рис. О.7.4.1.

При неточной стыковки рёбер погрешность расположения вершины равна расстоянию от точки вершины до края наиболее удалённого ребра. На рис. О.7.4.2 приведена толерантная вершина, в которой стыкуются четыре ребра.

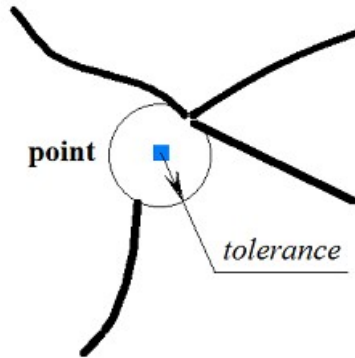


Рис. О.7.4.2.

Вершина может быть именована и может иметь атрибуты. Имя может использоваться для идентификации вершины, атрибуты могут нести дополнительную информацию о вершине, например, цвет, стиль отображения, происхождение и так далее. Вершина используется для всех методов моделирования.

О.7.5. Ребро грани MbCurveEdge

Класс MbCurveEdge объявлен в файле topology.h.

Ребро MbCurveEdge является наследником ребра MbEdge и представляет собой ребро, построенное на кривой пересечения поверхностей MbSurfaceIntersectionCurve. Ребро MbCurveEdge предназначено для описания участка границы грани. В отличие от ребра MbEdge ребро MbCurveEdge описывает не просто кривую, а участок стыковки двух граней или участок края грани.

Структура данных ребра грани содержит указатель на кривую пересечения поверхностей MbSurfaceIntersectionCurve * curve, признак совпадения направления ребра с направлением кривой sameSense, указатель на начальную вершину MbVertex* begVertex, указатель на конечную вершину MbVertex* endVertex, указатель на грань слева MbFace* facePlus и указатель на грань справа MbFace* faceMinus от ребра. На рис. О.7.5.1 приведено ребро, соединяющее две грани.

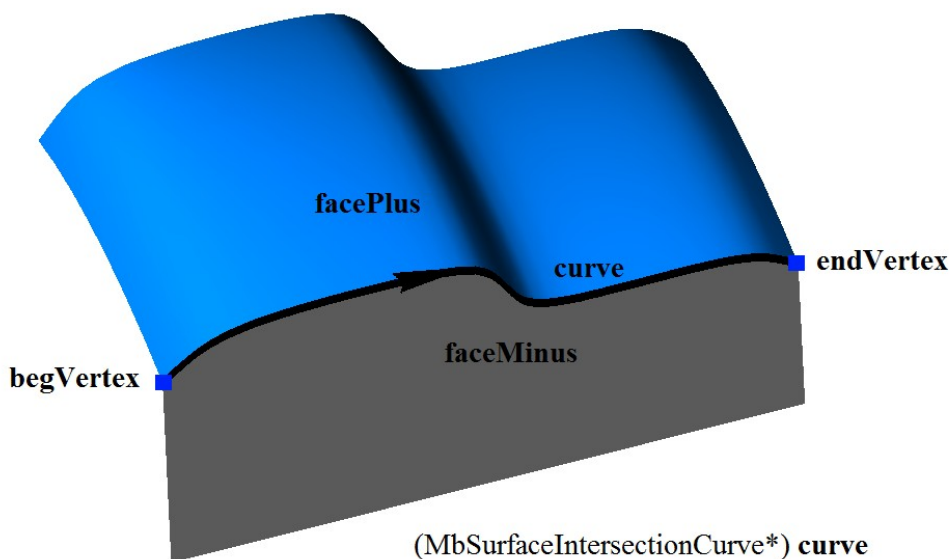


Рис. О.7.5.1.

Ребро грани может описывать участок стыковки двух различных граней. В этом случае указатели на грань слева и грань справа от ребра не равны нулю и не равны друг другу. Поверхности, лежащие в

структурах данных соединяемых ребром граней, совпадают с поверхностями, лежащими в структуре данных кривой ребра, а именно:

facePlus->surface->GetSurface() == curve->curveOne.suface и

faceMinus->surface->GetSurface() == curve->curveTwo.suface

или

facePlus->surface->GetSurface() == curve->curveTwo.suface и

faceMinus->surface->GetSurface() == curve->curveOne.suface.

У циклически замкнутой по одному или обоим параметрам поверхности грани присутствуют участки границы, по которым грань стыкуется сама с собой. Такое ребро является швом. В этом случае указатели на грань слева и грань справа от ребра равны друг другу, рис. O.7.5.2.

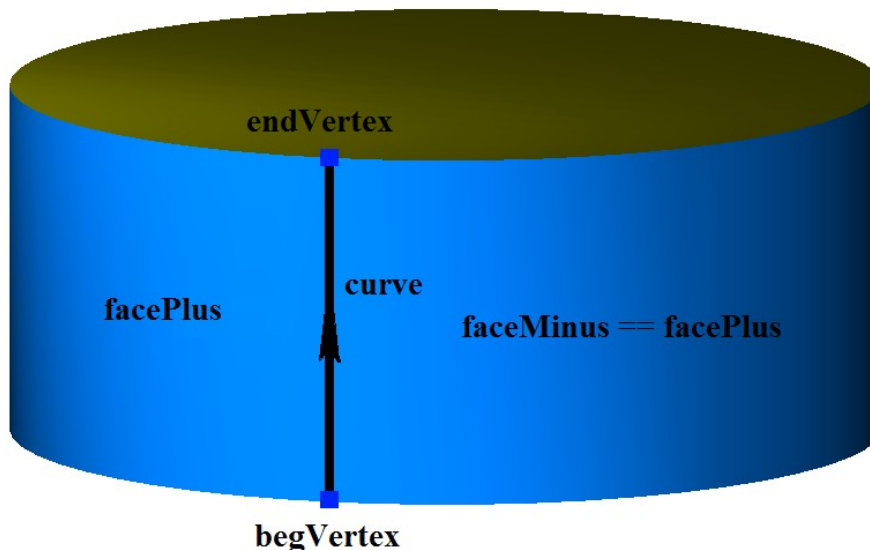


Рис. O.7.5.2.

Ребро грани может описывать участок края грани. Такое ребро является краевым. В этом случае указатель на грань слева или грань справа от ребра равен нулю, рис. O.7.5.3.

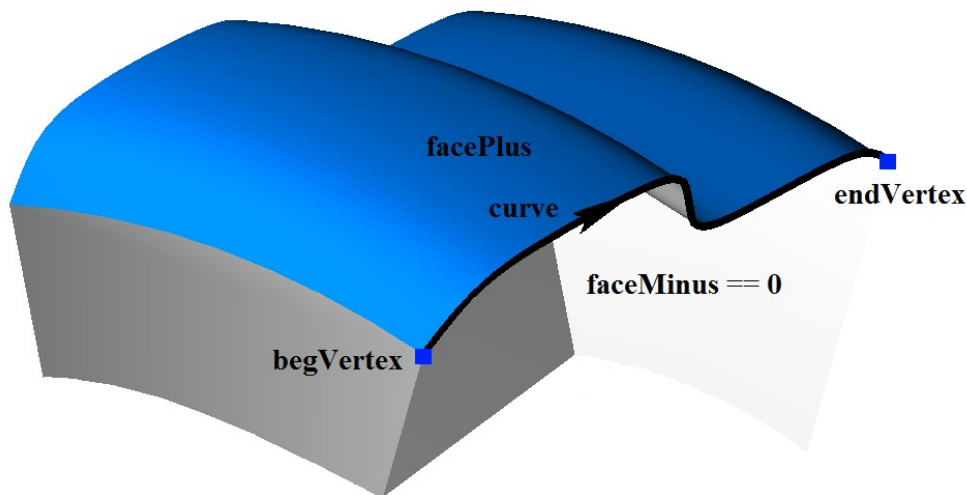


Рис. O.7.5.3.

Ребро, имеющее нулевую длину, является полюсным и описывает полюс грани. Полюсное ребро не является краевым, так как грань в полюсном ребре не имеет края. Как правило, полюсное ребро располагается в особых точках поверхности грани. В области параметров поверхности грани полюсному участку соответствует некоторая кривая. Указатели на начальную и конечную вершину полюсного ребра равны. Полюсное ребро приведено на рис. O.7.5.4.

begVertex == endVertex

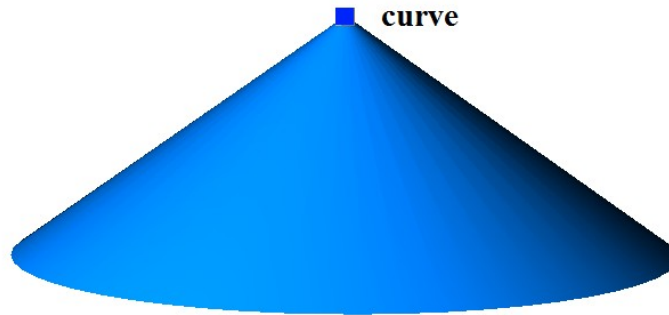


Рис. О.7.5.4.

У полюсного ребра указатель на грань слева или грань справа от ребра равен нулю. Кривая пересечения полюсного ребра имеет следующие данные:

curve->curveOne.surface == curve->curveTwo.surface, а отрезок **curve->curveOne.curve** является дублем отрезка **curve->curveTwo.curve**.

Ребро грани может быть именовано и может иметь атрибуты. Имя может использоваться для идентификации ребра, атрибуты могут нести дополнительную информацию о ребре грани, например, цвет, стиль, происхождение и так далее.

Ребро грани используется для твердотельного и гибридного моделирования.

О.7.6. Цикл грани MbLoop

Класс MbLoop объявлен в файле topology.h.

Цикл грани MbLoop является наследником топологического объекта [MbTopItem](#) и описывает последовательность рёбер, исчерпывающую некоторую границу грани.

Структура данных цикла содержит множество ориентированных рёбер `RPAArray<MbOrientedEdge>` **edgeList** в порядке их следования вдоль границы грани. У цикла есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов цикла.

Направление ориентированных рёбер совпадает с направлением цикла. Конец каждого ориентированного ребра цикла стыкуется с началом следующего за ним ориентированного ребра цикла, рис. О.7.6.1.

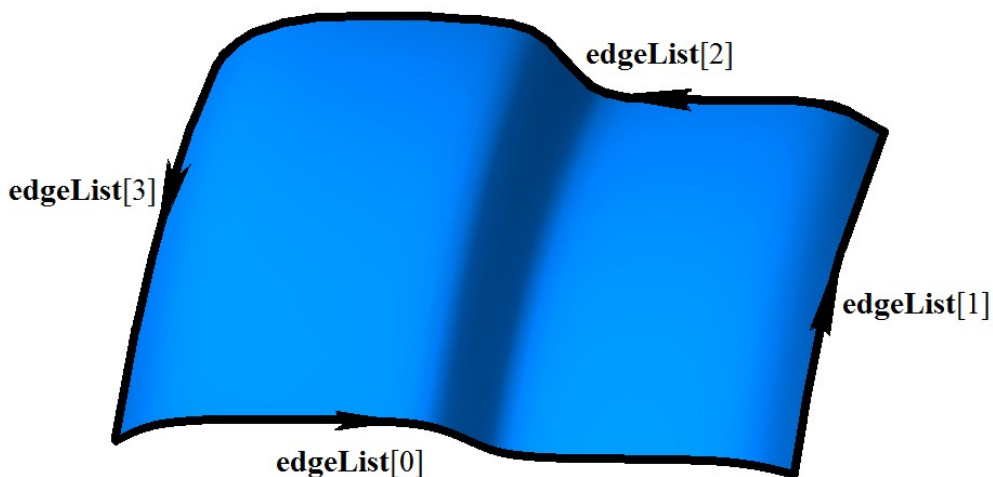


Рис. О.7.6.1.

Конец последнего ориентированного ребра цикла стыкуется с началом первого ориентированного ребра цикла.

На рис. О.7.6.2 приведён цикл сферической грани, состоящий из четырёх ориентированных рёбер, два из которых построены на ребре шва и два из которых являются полюсными рёбрами.

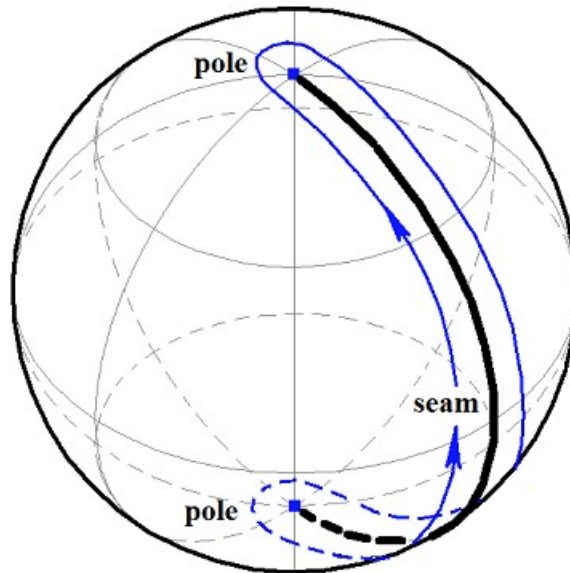


Рис. О.7.6.2.

Если цикл сферической грани нарисовать в пространстве параметров сферы, то он будет представлять собой прямоугольный четырёхугольник. Цикл также, как и граница грани всегда замкнут. Цикл направлен так, чтобы грань всегда для нас находилась бы слева, если мы движемся вдоль цикла с внешней стороны грани.

О.7.7. Ориентированное ребро грани MbOrientedEdge

Класс MbOrientedEdge объявлен в файле topology.h.

Ориентированное ребро MbOrientedEdge является наследником топологического объекта [MbTopItem](#) и описывает участок границы грани. Структура данных ориентированного ребра содержит ребро грани [MbCurveEdge*](#) **curveEdge** и признак совпадения направления ребра грани с направлением ориентированного ребра и, соответственно, с направлением цикла грани *orientation*.

Если направление ребра грани **curveEdge** совпадает с направлением цикла, то в цикле этой грани соответствующее ориентированное ребро имеет признак *orientation==true*. Если направление ребра грани **curveEdge** не совпадает с направлением цикла, то в цикле этой грани соответствующее ориентированное ребро имеет признак *orientation==false*. На рис. О.7.7.1 приведены два ориентированных ребра, построенных на одном и том же ребре грани **curveEdge**.

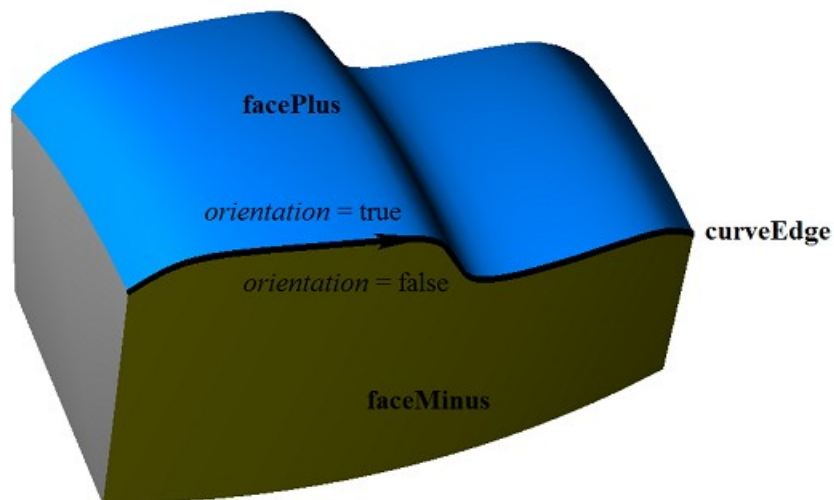


Рис. О.7.7.1.

В общем случае ребро грани [MbCurveEdge](#) входит в два цикла, эти циклы принадлежат соединяемым ребром граням. В один цикл ребро грани входит с признаком ориентации *orientation*==true, эта грань находится слева от ребра и поле данных **facePlus** указывает на неё. В другой цикл ребро грани входит с признаком ориентации *orientation*==false, эта грань находится справа от ребра и поле данных **faceMinus** указывает на неё. Таким образом смежные грани и соединяющие их рёбра связаны друг с другом.

О.7.8. Множество граней MbFaceShell

Класс MbFaceShell объявлен в файле topology_faceset.h.

Множество граней MbFaceShell является наследником топологического объекта [MbTopItem](#). Структура данных MbFaceShell содержит множество граней RPArray<[MbFace](#)> **faceSet** и признак замкнутости *closed* множества граней. У множества граней есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов множества граней.

Множество граней, как правило, описывает связный кусок поверхности моделируемого объекта. Связанные между собой грани множества удовлетворяют определенным условиям, а именно: грани стыкуются между собой по общим ребрам, в каждом ребре стыкуются только две грани, причём грани стыкуются так, что внешняя сторона одной грани переходит во внешнюю сторону другой грани, и образуемая стыкующимися гранями общая поверхность не должна иметь самопересечений.

На рис. О.7.8.1 приведено множество связанных между собой граней. Все рёбра граней входят в два цикла, у всех ребер указатели **facePlus** и **faceMinus** не равны нулю, а кривые [MbSurfaceIntersectionCurve](#), на которых построены рёбра граней, имеют **curveOne.surface**!=**curveTwo.surface**.

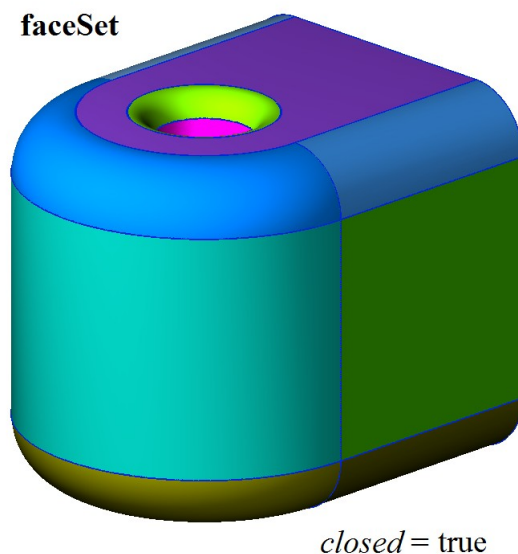


Рис. О.7.8.1.

Грани, приведенные на рис. О.7.8.1, образуют общую замкнутую поверхность, внешняя сторона каждой грани переходит во внешнюю сторону соседней грани. У множества граней, не имеющего ни одного краевого ребра, признак *closed* принимает значение true.

Если хотя бы одна грань множества имеет хотя бы одно краевое ребро, то признак *closed* множества граней принимает значение false. На рис. О.7.8.2 приведено множество граней, некоторые из которых имеют краевые ребра. Краевые рёбра входят только в один цикл и у краевых ребер грани только один из указателей **facePlus** или **faceMinus** не равен нулю, а кривые [MbSurfaceIntersectionCurve](#), на которых построены рёбра, имеют **curveOne.surface**==**curveTwo.surface** и параметр *buildType*=*cbt_Boundary*. Полносное ребро не является краевым, так как не образует край.

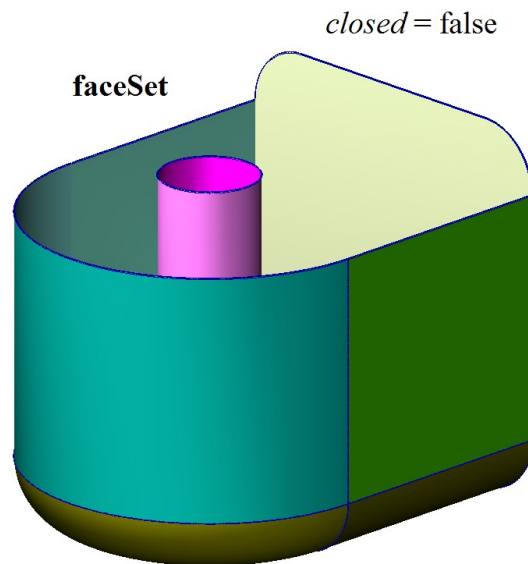


Рис. О.7.8.2.

Связное множество граней называют *замкнутой оболочкой*, если грани множества не имеют края. Если стыкующиеся друг с другом грани имеют хотя бы одно краевое ребро, то такое связное множество граней называют *незамкнутой оболочкой*. Подчеркнем, что замкнутая оболочка и незамкнутая оболочка представляют собой связное множество стыкующихся друг с другом граней.

Множество граней может состоять из нескольких не связанных между собой частей. На рис. О.7.8.3. приведено множество граней, описывающее две незамкнутые оболочки.

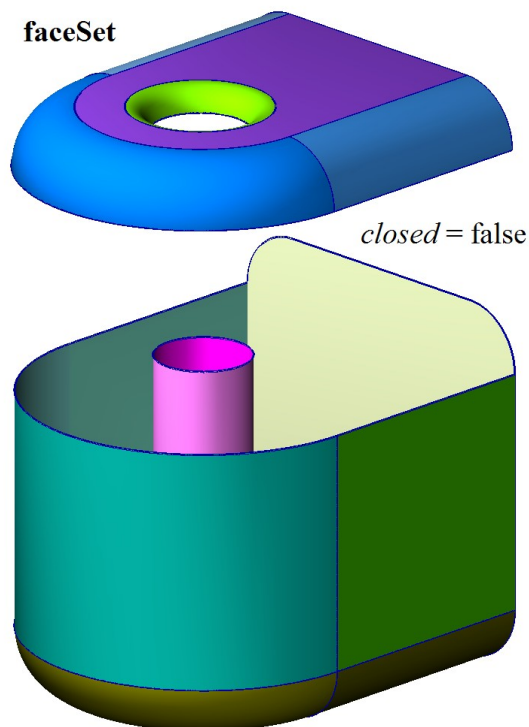


Рис. О.7.8.3.

Формально на грани множества не наложено никаких ограничений. Множество граней может состоять из отдельных граней. На рис. О.7.8.4 приведено множество не связанных между собой граней. Каждая грань, приведенная на рис. О.7.8.4, образует отдельную оболочку, все ребра граней являются краевыми.

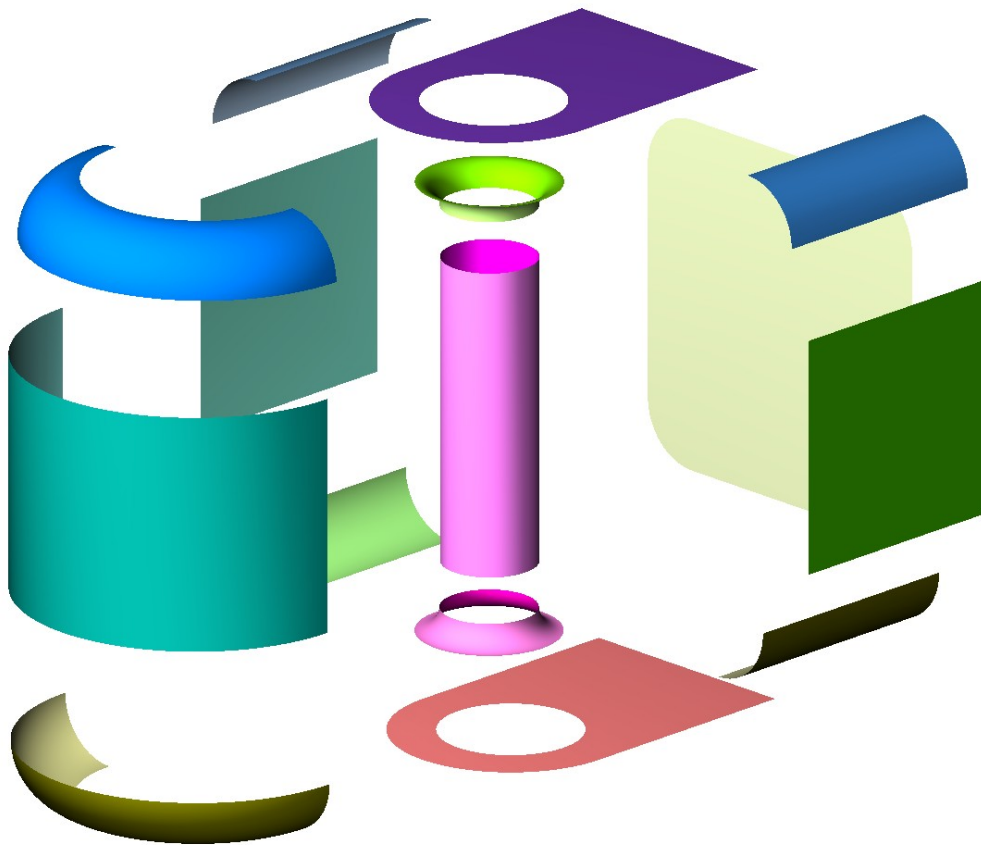


Рис. 0.7.8.4.

Основными методами оболочки являются методы преобразования её в пространстве:
 void **Move**(const [MbVector3D](#) & v, MbRegTransform * iReg = NULL),
 void **Rotate**(const MbAxis3D & axis, double angle, MbRegTransform * iReg = NULL),
 void **Transform**(const [MbMatrix3D](#) & m, MbRegTransform * iReg = NULL),
 методы поиска граней, рёбер и вершин, а также методы определения положения точки относительно замкнутой оболочки:
 bool **DistanceToBound**(const [MbCartPoint3D](#) &,...),
 bool **PointClassification**(const [MbCartPoint3D](#) &,...).

Незамкнутая оболочка описывает только часть граничной поверхности моделируемого объекта. Незамкнутые оболочки используются для поверхностного моделирования. Если к замкнутой оболочке добавить множество точек, лежащих внутри неё, то мы получим *твердое тело*. Замкнутые оболочки используются для твердотельного моделирования. Обрезка и стыковка граней выполняется одновременно с построением оболочки. Это обеспечивают методы геометрического ядра C3D. На базе замкнутых и незамкнутых оболочек строятся объекты геометрической модели.

0.7.9. Копирование множества граней MbFaceShell

Каждый метод построения, который в составе входящих параметров использует множество граней, представленных в виде [MbFaceShell](#) или [MbSolid](#), модифицирует некоторые вершины, рёбра и грани исходных объектов. Для ускорения построений и сохранения неизменным исходного множества граней применяется полное или частичное копирование данных объекта MbFaceShell. В геометрическом ядре C3D используются четыре способа копирования множества граней MbFaceShell, определяемые перечислением MbCopyMode. Как правило, в методы построения вместе с модифицируемым множеством граней передается параметр типа MbCopyMode, который управляет передачей граней, ребер и вершин от исходного объекта построенному объекту.

Перечисление MbCopyMode объявлено в файле mb_enum.h. Параметр типа MbCopyMode может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*.

При значении *cm_Copy* исходное множество граней модифицируемого объекта полностью копируется, поэтому исходный объект и построенный объект не будут иметь общих поверхностей, кривых, граней, ребер, вершин и других объектов. Этот вариант используется в случаях, когда требуется, чтобы построенный объект не был связан с исходным объектом.

При значении *cm_KeepSurface* исходный объект и построенный объект будут иметь одни и те же базовые поверхности граней. Этот вариант используется в случаях, когда требуется высокая скорость построения.

При значении *cm_KeepHistory* исходный объект и построенный объект будут иметь одни и те же вершины, базовые поверхности граней и грани, не модифицированные выполненным построением или другим действием. Этот вариант используется в случаях, когда требуется максимально экономить память.

При значении *cm_Same* в построенный объект будут перенесены все необходимые данные исходного объекта, поэтому после построения исходный объект должен быть удален. Этот вариант используется в случаях, исходный объект в дальнейшем не нужен и был построен специально для выполненного построения.

Перечисление *MbeCopyMode* присутствует в методе копирования множества граней *MbFaceShell* MbFaceShell::Copy(MbeCopyMode, MbShellHistory*)*. Этот метод используется в операциях построения тел, в составе входящих параметров которых присутствуют другие тела.

При значении перечисления *cm_Copy* исходное множество граней и его копия не имеют общих данных. Варианту, при котором исходное множество граней и его копия имеют общие базовые поверхности, соответствует значение перечисления *cm_KeepSurface*. Варианту, при котором исходное множество граней и его копия имеют общие базовые поверхности, вершины и не изменённые операцией грани, соответствует значение перечисления *cm_KeepHistory*. В этом метод *Copy(...)* использует указатель на объект *MbShellHistory*, который запоминает соответствие между исходным множеством граней и его копией. После выполнения операции копия множества граней передается параметром в метод *MbShellHistory::SetOrigins(MbFaceShell&)* для замены в копии не изменённых граней их оригиналами исходного множества граней. Варианту, при котором множество граней в методе *Copy(...)* не копируется, соответствует значение перечисления *cm_Same*. Следует заметить, что при неудачном выполнении операции, исходное множество граней будет модифицировано.

Для копирования множества граней *MbFaceShell* можно использовать метод *MbFaceShell* Duplicate(MbRegDuplicate* iReg)*. Объект *MbRegDuplicate* используется для сохранения в копии структуры взаимных ссылок, которая присутствует в исходном множестве граней. Копируемое множество граней и их копия не будут связаны между собой.

О.7.10. Именованние граней, рёбер и вершин

Грани, рёбра и вершины в структуре данных имеют имя *MbName*. Класс *MbName* объявлен в файле *name_item.h*. Именованние граней, рёбер и вершин выполняется геометрическим ядром C3D при построении множества граней *MbFaceShell* во всех формообразующих операциях. В параметрах каждой формообразующей операции присутствует объект *MbSNameMaker*. Объект *MbSNameMaker* содержит главное имя операции и контейнер простых имен типа *SimpleName*, которые используются для именования граней. Объект *MbSNameMaker* выполняет именование построенных граней с помощью элементов контейнера простых имен. Контейнер простых имен может иметь или набор уникальных чисел, или одно целое число, или ни одного целого числа.

Если контейнер простых имен содержит одно число, то остальные требуемые простые имена *MbSNameMaker* создаст сам путем прибавления к исходному числу чисел натуральной последовательности. Если контейнер простых имен пуст, то стартовым числом будет нуль. Каждое число из контейнера простых имен будет соответствовать одному из геометрических входных параметров (сегменту формообразующего контура операции, обрабатываемому ребру операции, модифицируемой грани операции, и тому подобное) и будет использоваться для именования новой грани, рожденной операцией.

Имена граней будут уникальными при уникальности элементов контейнера простых имен. Ребра получают имена хешированием имён соединяемых ими граней. Вершины получают имена хешированием имён стыкующихся в них рёбер. В качестве идентификатора грани, ребра или вершины можно использовать число, которое выдаст метод *MbName::Hash()* соответствующего

топологического объекта. Грани, ребра и вершины в теле можно искать по известному имени MbName методами **FindFaceByName(...)**, **FindEdgeByName(...)**, **FindVertexByName(...)**.

Кроме того, объект MbSNameMaker содержит версию способа построения и обеспечивает сохранение старых способов построений при их модификации в процессе развития геометрического ядра..

О.8. ОБЪЕКТЫ ГЕОМЕТРИЧЕСКОЙ МОДЕЛИ

Среди трёхмерных геометрических объектов выделен класс объектов геометрической модели. Для твердотельного, поверхностного и прямого моделирования используется такой объект геометрической модели, как тело. Тела строятся и при моделировании объектов из листового металла. Кроме тела к объектам геометрической модели относятся проволочный каркас, точечный каркас и полигональный объект. Из объектов геометрической модели можно создавать сборочные единицы. Для вспомогательных построений могут использоваться такие объекты геометрической модели как локальная система координат. Среди объектов геометрической модели есть объект для построения эскизов.

О.8.1. Объект геометрической модели MbItem

Класс MbItem объявлен в файле model_item.h.

Объект геометрической модели MbItem является наследником классов [MbSpaceItem](#), MbTransactions и MbAttributeContainer. Геометрическое ядро C3D оперирует объектами геометрической модели, которые приведены на рис. О.8.1.1.

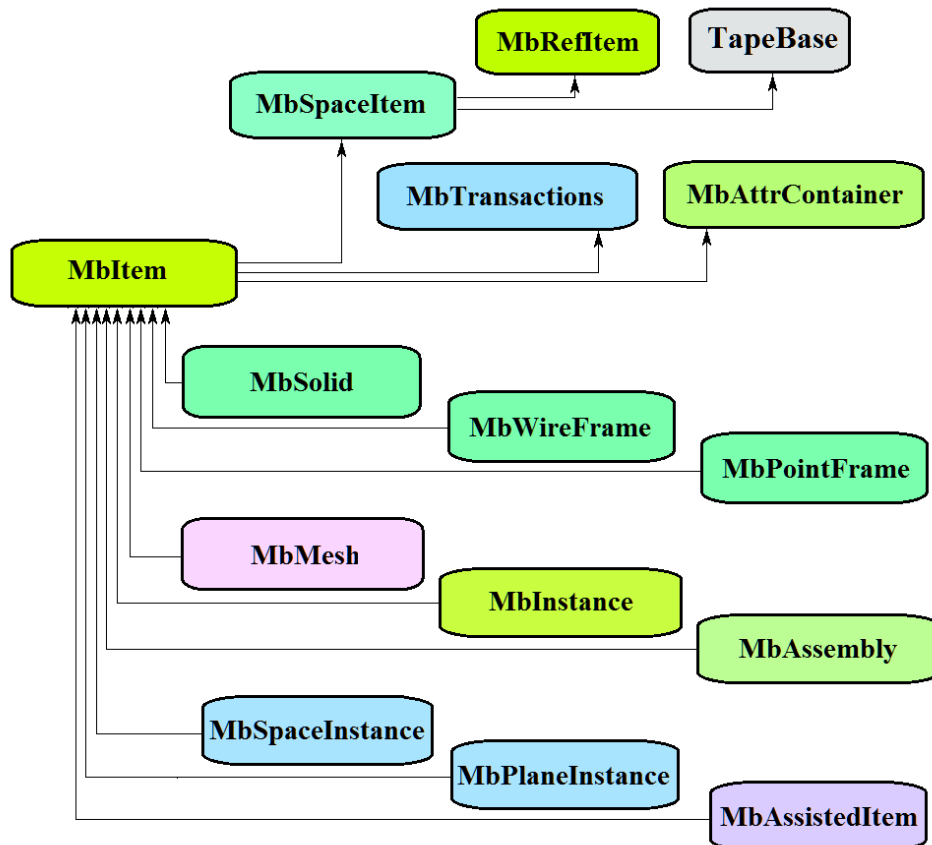


Рис О.8.1.1.

Журнал построения MbTransactions содержит данные, необходимые для построения объекта и позволяет повторить построение объекта с отредактированными параметрами. Контейнер атрибутов MbAttrContainer наделяет объекты геометрической модели атрибутами. Таким образом, все объекты геометрической модели кроме своих специфических данных содержат следующие данные:

size_t m_countRegistrable – количество регистраций объекта при записи и чтении,
ptrdiff_t useCount – количество использований объекта другими объектами,
std::vector<MbCreator *> transactions – упорядоченное множество строителей объекта,
std::multimap<int, MbAttribute*> attributes – множество атрибутов объекта.

Наследниками класса MbItem являются следующие объекты геометрической модели:

[MbSolid](#) – твёрдое тело,
[MbWireFrame](#) – проволочный каркас,
[MbPointFrame](#) – точечный каркас,
[MbMesh](#) – полигональный объект,
[410](#) – вставка объекта геометрической модели,
[MbAssembly](#) – сборочная единица объектов геометрической модели,
[MbAssistingItem](#) – вспомогательный объект,
[MbSpaceItem](#) – вставка трёхмерного объекта,
[MbPlaneInstance](#) – вставка множества двумерных объектов.

Основными методами объектов геометрической модели являются методы, обеспечивающие редактирование и визуализацию объектов.

Метод

bool **RebuildItem**(MbeCopyMode sameShell, RPArray<[MbSpaceItem](#)> * **items**)
выполняет построение объекта заново по журналу построения. Этот метод вызывается после редактирования внутренних данных объекта.

Метод

MbItem * **CreateMesh**(MbeStepData data, bool wire, bool grid, MbRegDuplicate * iReg)
создаёт полигональную копию объекта. Если объект представляет собой сборочную единицу или вставку, то копия объекта будет также сборочной единицей или вставкой с полигональными объектами.

Метод

bool **AddYourMesh**(MbeStepData data, bool wire, bool grid, [MbMesh](#) & **mesh**)
добавляет полигональную копию объекта в объект **mesh**.

Метод

bool **NearestMesh**(MbeSpaceType sType, MbeTopologyType tType, MbePlaneType pType,
const MbAxis3D & **axis**, double maxDistance, double & t, double & dMin,
MbItem *& **find**, SimpleName & findName,
[MbRefItem](#) *& **element**, SimpleName & elementName,
MbPath & path, [MbMatrix3D](#) & **from**)
выполняет поиск ближайшего полигонального объекта **find**, его элемента **element**, их имён findName и elementName, пути в структуре сборочной единицы path и матрицы преобразования в глобальную систему координат **from**.

Объекты геометрической модели перегружают такие методы трёхмерного объекта как:

методы, обслуживающие преобразование геометрического объекта,
void **Move**(const [MbVector3D](#) & v, MbRegTransform * iReg = NULL),
void **Rotate**(const MbAxis3D & **axis**, double **angle**, MbRegTransform * iReg = NULL),
void **Transform**(const [MbMatrix3D](#) & m, MbRegTransform * iReg = NULL),
методы, обеспечивающие копирование, проверку на совпадение, проверку на возможность сделать совпадающими, делающие объекты совпадающими,
[MbSpaceItem](#) & **Duplicate**(MbRegDuplicate * iReg = NULL),
bool **IsSame**(const [MbSpaceItem](#) & **item**),
bool **IsSimilar**(const [MbSpaceItem](#) & **item**),
bool **SetEqual**(const [MbSpaceItem](#) & **item**),
методы, возвращающие тип из перечисления геометрических объектов,
MbeSpaceType **IsA**(),
MbeSpaceType **Type**(),
MbeSpaceType **Family**(),
методы, обеспечивающие выдачу и редактирование внутренних данных объекта,
MbProperty & **CreateProperty**(MbePrompt name),
GetProperties(MbProperties & *properties*),
SetProperties(MbProperties & *properties*).

0.8.2. Твёрдое тело MbSolid

Класс MbSolid объявлен в файле solid.h.

Твёрдое тело или просто тело `MbSolid` является наследником класса `MbItem` и описывается множеством граней `MbFaceShell* outer` и типом связности `multiState`.

Тело представляет собой набор граней, стыкующихся друг с другом по ребрам и описывающих поверхность моделируемого объекта. Тело может описывать одно или несколько связанных множеств точек. Тип связности `multiState` сообщает о том, что тело описывает одно связанное множество точек, или, что тело описывает несколько связанных множеств точек и может быть разбито на несколько тел.

Множество граней тела `outer` в зависимости наличия краевых ребер может описывать два принципиально разных множества точек. Если множество граней не имеет края, то тело описывает множество точек, располагающихся на поверхности граней и с внутренней стороны от этих граней. Такое тело называется *замкнутым* и описывается замкнутой оболочкой. Замкнутое тело приведено на рис 0.8.2.1.

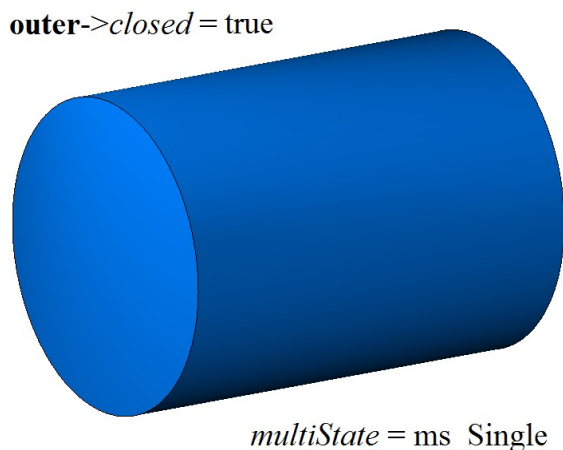


Рис. 0.8.2.1.

Если множество граней тела имеет край или края, то тело описывает множество точек, располагающихся на поверхности граней и только. Такое тело называется *незамкнутым* и описывается незамкнутой оболочкой. Незамкнутое тело приведено на рис 0.8.2.2.

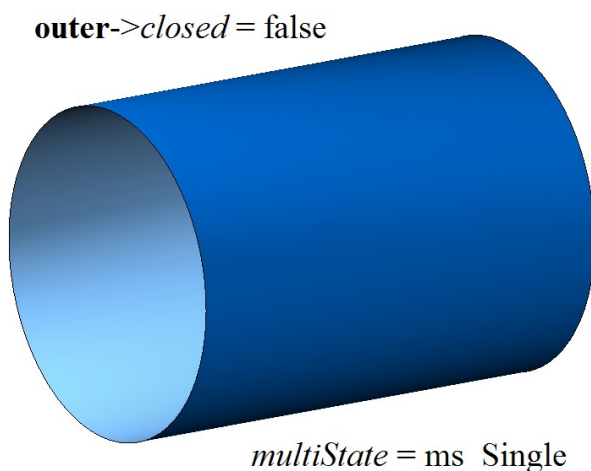
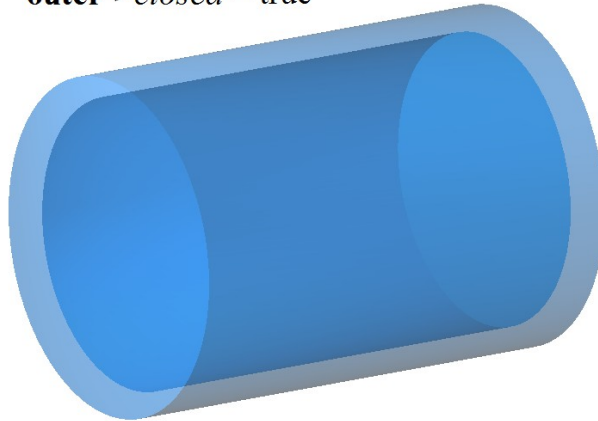


Рис. 0.8.2.2.

В большинстве случаев замкнутое тело описывается одним связным множеством граней – одной оболочкой. Если у замкнутого тела есть внутренние полости, тело описывается несколькими связными наборами граней. На рис 0.8.2.3 приведено замкнутое тело, которое описывается двумя замкнутыми оболочками, одной внешней, а другой внутренней, располагающейся внутри внешней оболочки.

outer->closed = true



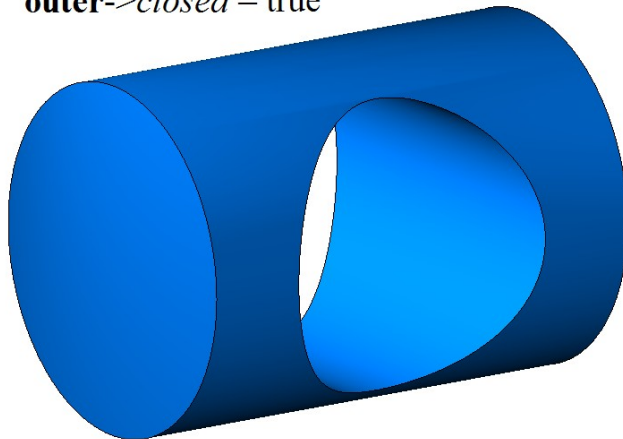
multiState = ms_Single

Рис. 0.8.2.3.

Чтобы увидеть полость тела, тело выполнено полупрозрачным.

Над телами можно выполнять различные операции – совокупность действий, которая приводит к образованию тела иной формы, например, булевы операции. Результат вычитания двух замкнутых тел приведен на рис. 0.8.2.4.

outer->closed = true

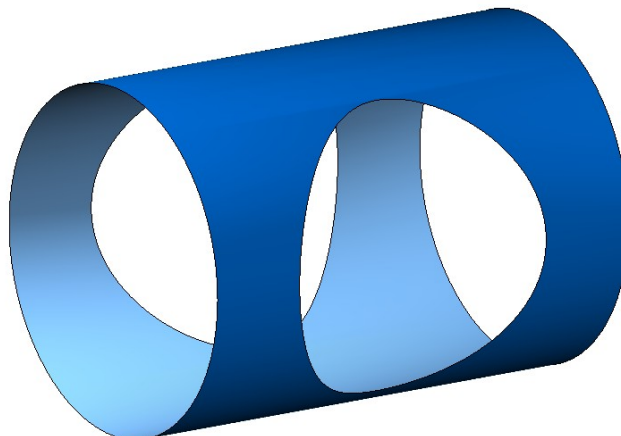


multiState = ms_Single

Рис. 0.8.2.4.

Результат операции над замкнутым и незамкнутым телом принципиально отличается, так как действия выполняются над разными множествами точек. Результат вычитания тела из незамкнутого тела приведен на рис. 0.8.2.5.

outer->closed = false

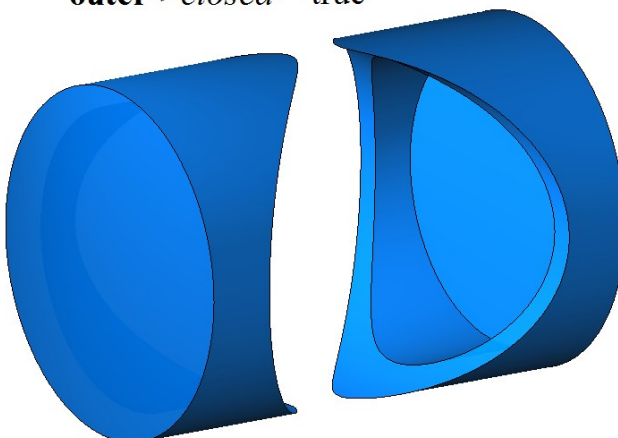


multiState = ms_Single

Рис. O.8.2.5.

Тело может быть многосвязным, то есть может состоять из нескольких отдельных частей. В этом случае *multiState* принимает значение *ms_Multiple*. Двусвязное тело, описываемое двумя замкнутыми оболочками, приведено на рис. O.8.2.6.

outer->closed = true

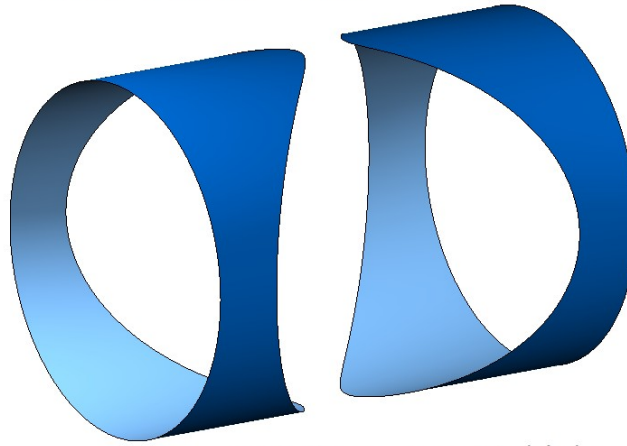


multiState = ms_Multiple

Рис. O.8.2.6.

Такое тело может быть разделено на два односвязных замкнутых тела методом [::165\(...\)](#). Тело на рис. O.8.2.6 выполнено полупрозрачным. Двусвязное тело, описываемое двумя незамкнутыми оболочками, приведено на рис. O.8.2.7.

`outer->closed = false`



`multiState = ms_Multiple`

Рис. О.8.2.7.

Такое тело может быть разделено на два односвязных незамкнутых тела.

Тела создаются методами геометрического ядра С3D. Тела простой формы создаются по точкам, кривым и поверхностям. С помощью операций из тел простой формы можно получить более сложные тела. Редактировать исходные тела и создавать подобные тела можно путём изменения параметров журнала построения `MbTransactions` или путём непосредственной модификации элементов уже построенных тел. Незамкнутые тела используются в поверхностном моделировании. Незамкнутое тело позволяет сосредоточить усилия на сложных формах моделируемого объекта.

О.8.3. Проволочный каркас `MbWireFrame`

Класс `MbWireFrame` объявлен в файле `wire_frame.h`.

Проволочный каркас `MbWireFrame` является наследником класса `MbItem` и описывается множеством рёбер `std::vector<MbEdge*>edges`, количеством связанных частей `parts` и признаком отсутствия краевых вершин `closed`.

Проволочный каркас представляет собой множество рёбер, стыкующихся друг с другом в вершинах и описывающих каркасную конструкцию моделируемого объекта. Проволочный каркас может описывать одно или несколько связанных множеств точек. Количеством связанных частей `parts` сообщает о том, что каркас описывает одно связное множество точек, или, что каркас описывает несколько связанных множеств точек и может быть разбит на несколько проволочных каркасов.

В зависимости от отсутствия или наличия краевых вершин проволочный каркас будем называть *замкнутым* или *незамкнутым*. На рис О.8.3.1 приведен замкнутый проволочный каркас.

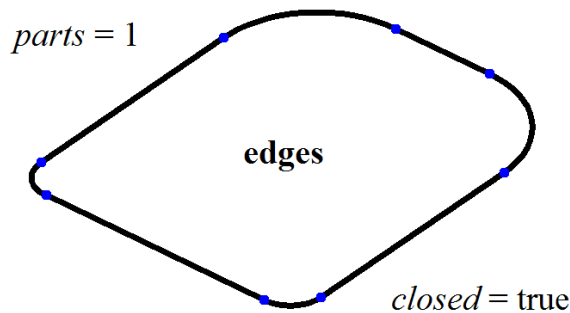


Рис. О.8.3.1.

Незамкнутый проволочный каркас приведен на рис О.8.3.2.

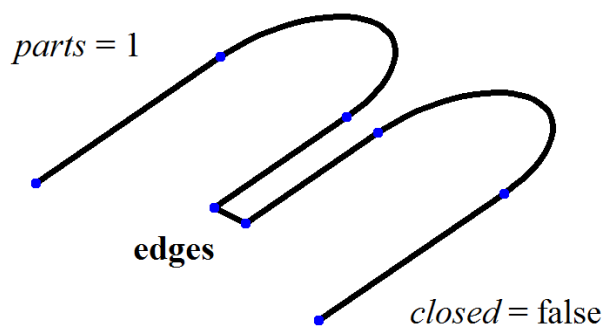


Рис. O.8.3.2.

На рис O.8.3.3 приведен незамкнутый проволочный каркас, состоящий из двух связных множеств рёбер.

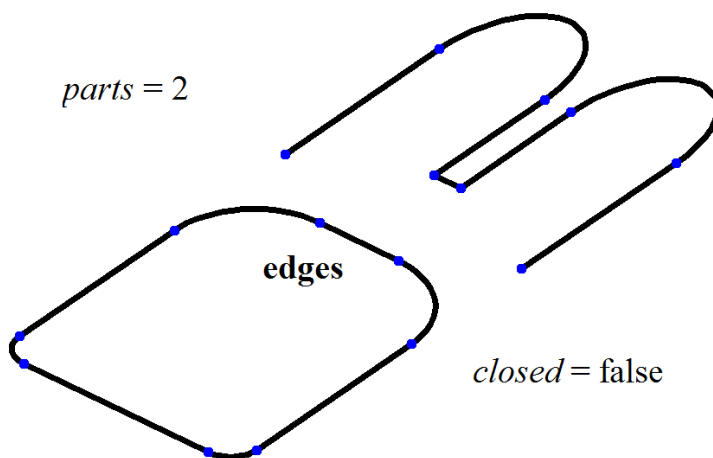


Рис. O.8.3.3.

На рис O.8.3.4 приведен замкнутый проволочный каркас, состоящий из двух связных множеств рёбер.

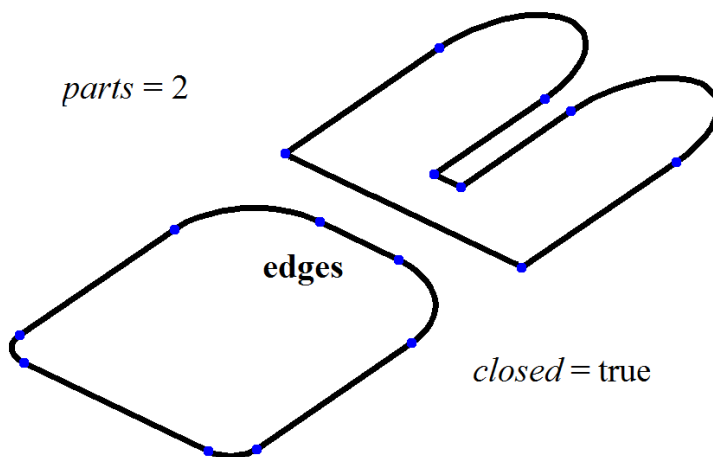


Рис. O.8.3.4.

Проволочный каркас может использоваться для построения траекторий, пространственных эскизов и для вспомогательных построений.

О.8.4. Точечный каркас MbPointFrame

Класс MbPointFrame объявлен в файле point_frame.h.

Точечный каркас [MbWireFrame](#) является наследником класса [MbItem](#) и описывается множеством точек, представленных в виде вершин, `std::vector<MbVertex*>vertices`.

На рис О.8.4.1 приведен точечный каркас.

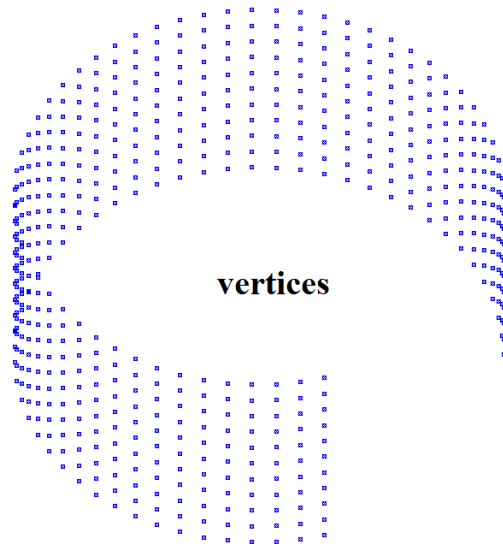


Рис. О.8.4.1.

Точечный каркас может использоваться для позиционирования других объектов и для вспомогательных построений.

О.8.5. Полигональный объект MbMesh

Класс MbMesh объявлен в файле mesh.h.

Полигональный объект MbMesh является наследником класса [MbItem](#) и описывается множеством триангуляций `RArray<MbGrid>grids`, множеством полигонов `RArray<MbPolygon3D>wires`, множеством апексов `RArray<MbApex3D>peaks`, указателем на оригинальный объект [MbRefItem*](#) `item`, типом `type`, габаритным кубом `cube` и признаком замкнутости `closed`.

Полигональный объект представляет собой набор треугольных и четырехугольных пластин, ломаных линий и отдельных точек. Одним из способов построения полигонального объекта является аппроксимация других объектов геометрической модели, например, тела. Каждая i -ая грань тела аппроксимируется триангуляцией `grids[i]`, каждое j -ое ребро тела аппроксимируется полигоном `wires[j]`, каждой k -ой вершине тела соответствует апекс `peaks[k]`, признак замкнутости `closed` соответствует замкнутости тела. Ещё одним способом получения полигонального объекта является импорт данных с помощью конвертера полигонального представления.

Триангуляция MbGrid представляет собой множество точек `Sarray<MbFloatPoint3D>points`, согласованное с ним множество нормалей `Sarray<MbFloatVector3D>normals` (количество точек равно количеству нормалей), множество двумерных точек параметрической области поверхности `Sarray<MbFloatPoint>params` (количество двумерных точек равно количеству трехмерных точек или равно нулю – множество может быть пустым), множество треугольных пластин `Sarray<MbTriangle>triangles` в виде трех индексов множества точек `points`, множество четырехугольных пластин `SArray<MbQuadrangle>quadrangles` в виде четырех индексов множества точек `points`. Пластины триангуляции аппроксимируют некоторую поверхность.

Полигон MbPolygon3D представляет собой упорядоченное множество точек, последовательное соединение которых даст ломаную линию, аппроксимирующую некоторую кривую.

Апекс MbApex3D представляет собой точку, наделенную дополнительными данными.

Указатель на оригинальный объект **item** может быть равен нулю, а тип *type* может быть неопределенным.

Векторное изображение полигонального объекта приведено на рис. О.8.5.1.

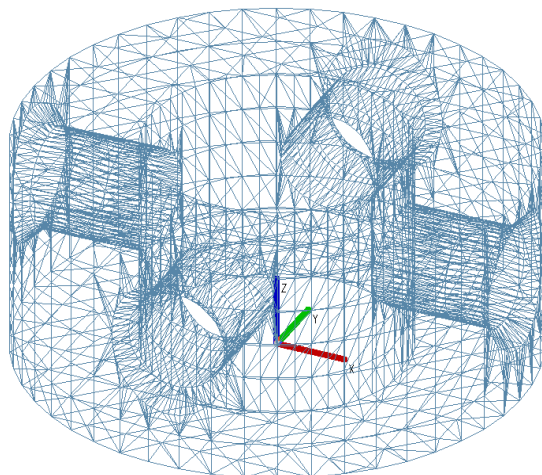


Рис. О.8.5.1.

Тонированное изображение полигонального объекта приведено на рис. О.8.5.2.

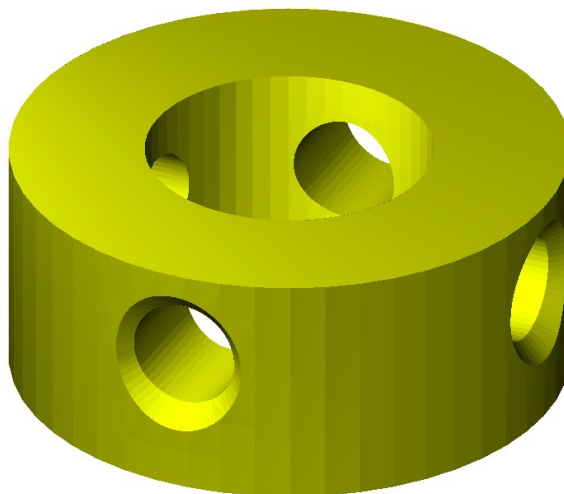


Рис. О.8.5.2.

На рис. О.8.5.2 видны отдельные треугольники объекта, из-за того, что направление нормалей в каждом треугольнике постоянно. Если направление нормалей в треугольниках триангуляции непрерывно меняется, то отдельные треугольники объекта становятся не видны, рис. О.8.5.3.

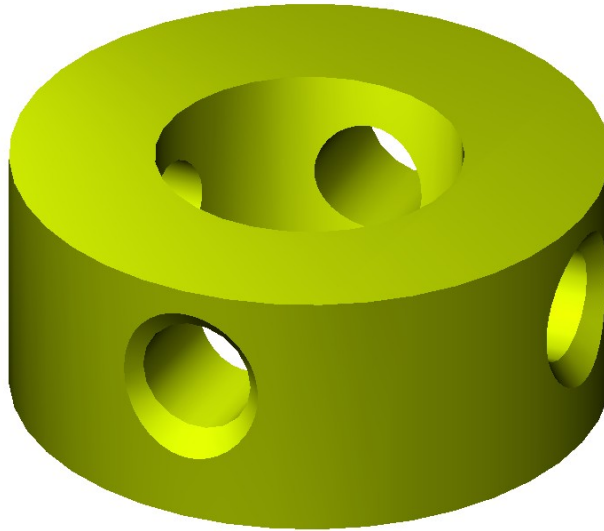


Рис. О.8.5.3.

Полигональный объект может быть создан методом [MbItem::CreateMesh\(...\)](#) или методом [MbItem::AddYourMesh\(...\)](#). Полигональный объект используется для визуализации, расчетов, производства моделируемых объектов. Основным преимуществом полигонального объекта является простота и высокая скорость вычислений, например, пересечения с прямой линией.

О.8.6. Вставка MbInstance

Класс MbInstance объявлен в файле instance.h.

Вставка объекта MbInstance является наследником класса [MbItem](#) и описывается объектом геометрической модели [MbItem*](#) **item** и локальной системой координат [MbPlacement3D](#) **place**. Вставка представляет собой объект **item**, перенесенный в локальную систему координат **place**.

Вставка обладает всеми свойствами объекта **item**. Отличие заключается в том, что методы [Move\(...\)](#), [Rotate\(...\)](#), [Transform\(...\)](#) изменяют систему координат **place** не меняя объект **item**.

Вставка объекта может содержать тело, проволочный каркас, точечный каркас, полигональный объект, но не может содержать другую вставку или сборочную единицу.

О.8.7. Сборочная единица MbAssembly

Класс MbAssembly объявлен в файле assembly.h.

Сборочная единица MbAssembly или сборка является наследником класса [MbItem](#) и описывается множеством объектов геометрической модели `std::vector<MbItem*>` **assemblyItems** и локальной системой координат [MbPlacement3D](#) **place**.

Сборка представляет собой множество объектов геометрической модели, которыми можно оперировать как единым объектом.

Сборочная единица приведена на рис. О.8.7.1.

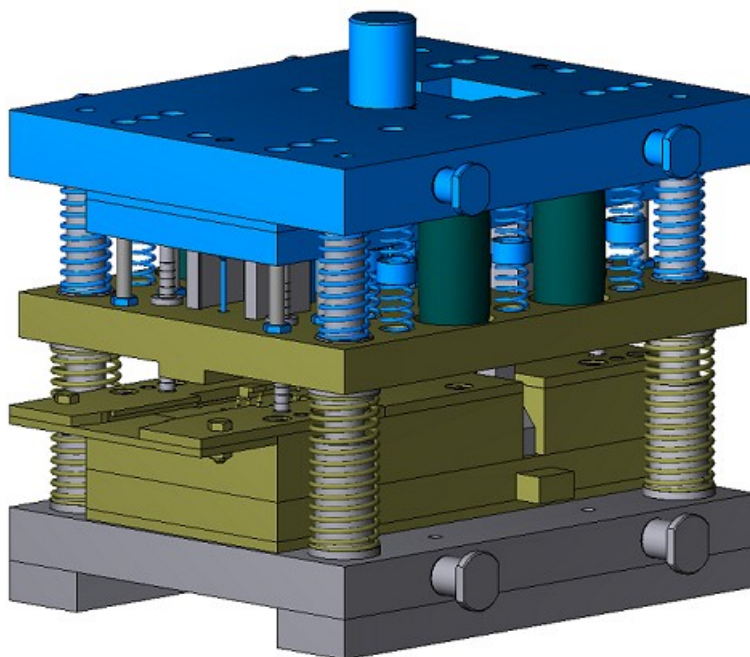


Рис. О.8.7.1.

Сборочная единица может содержать другие сборочные единицы в качестве своих объектов, то есть может иметь древовидную структуру.

О.8.8. Вставка трёхмерного объекта `MbSpaceItem`

Класс `MbSpaceItem` объявлен в файле `space_instanse.h`.

Вставка трёхмерного объекта `MbSpaceItem` является наследником класса [MbItem](#) и описывается геометрическим объектом [MbSpaceItem](#)* `spaceItem`.

Вставка играет роль обёртки геометрического объекта, позволяющей работать с ним как с объектом геометрической модели. Вставка придает обычному геометрическому объекту журнал построения, наделяет объект атрибутами и методами объекта геометрической модели [MbItem](#). Вставка трёхмерного объекта предназначена для вспомогательных построений. Вставка поверхности приведена на рис. О.8.8.1.

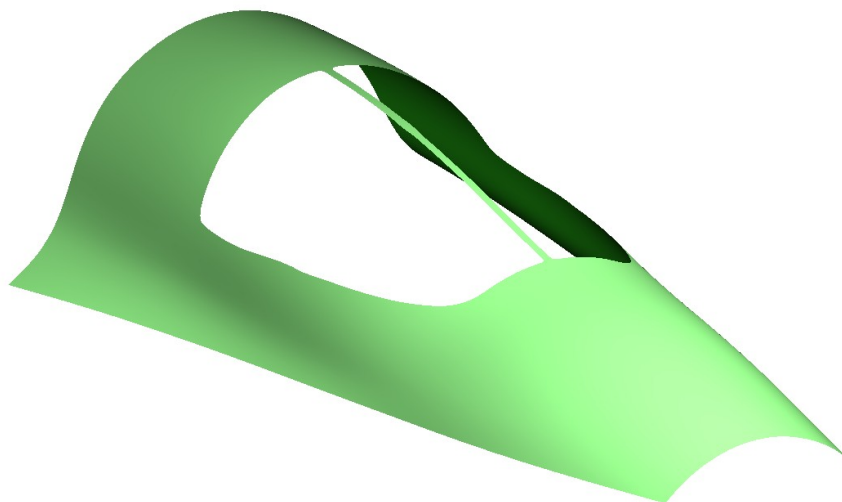


Рис. О.8.8.1.

Вставка объекта может содержать поверхность [MbSurface](#), кривую [MbCurve3D](#), точку [MbPoint3D](#) или вспомогательный геометрический объект [MbLegend](#). Для объектов, наследников объекта геометрической модели [MbItem](#) (тело, проволочный каркас, точечный каркас, полигональный объект, сборка, вставка), вставка геометрического объекта не применяется.

О.8.9. Вставка двумерных объектов [MbPlaneInstance](#)

Класс [MbPlaneInstance](#) объявлен в файле `plane_instanse.h`.

Вставка двумерных объектов [MbPlaneInstance](#) является наследником класса [MbItem](#) и описывается множеством двумерных геометрических объектов `std::vector<MbPlaneItem>` `planeItems` и локальной системой координат [MbPlacement3D](#) `place`. Двумерные объекты располагаются в плоскости XY локальной системы координат.

Вставка играет роль обёртки двумерных геометрических объектов, позволяющей работать с ними как с объектом геометрической модели. Вставка придает двумерным геометрическим объектам журнал построения, наделяет их атрибутами и методами объекта геометрической модели [MbItem](#). Вставка двумерных объектов предназначена для вспомогательных построений. Вставка двумерных кривых приведена на рис. О.8.9.1.

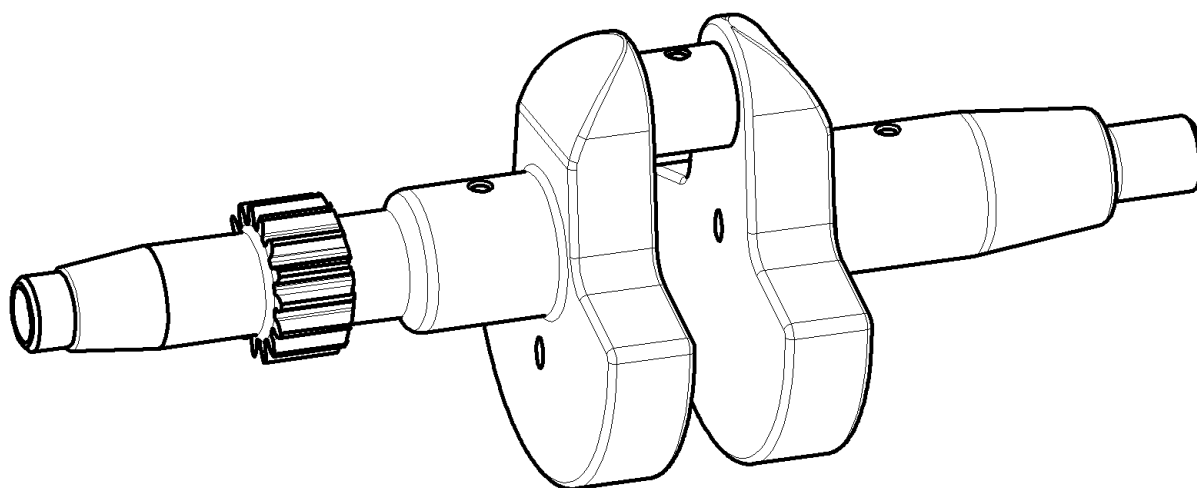


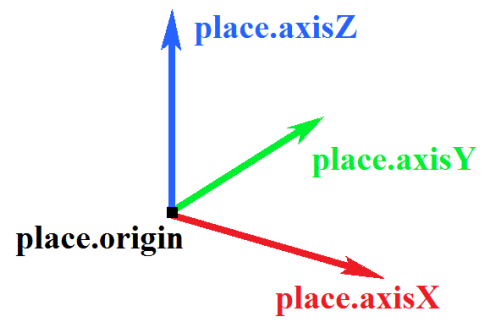
Рис. О.8.9.1.

Вставка двумерных объектов может содержать двумерную кривую [MbCurve](#), мультилинию [MbMultiline](#) или регион [MbRegion](#).

О.8.10. Вспомогательный объект [MbAssistingItem](#)

Класс [MbAssistingItem](#) объявлен в файле `assisting_item.h`.

Вспомогательный объект геометрической модели [MbAssistingItem](#) является наследником класса [MbItem](#) и описывается локальной системой координат [MbPlacement3D](#) `place`. Вспомогательный объект предназначен для позиционирования других объектов. Вспомогательный объект обладает журналом построения, атрибутами и методами объекта геометрической модели [MbItem](#). Вспомогательных построений. Вспомогательный объект приведен на рис. О.8.10.1.



Puc. O.8.10.1.

R.1. ПОСТРОЕНИЕ ТРИАНГУЛЯЦИИ

Геометрическое ядро C3D выполняет построение полигонального представления геометрической модели по её граничному представлению. Полигональное представление содержит набор триангуляций. Каждая триангуляция аппроксимирует отдельную грань моделируемого объекта плоскими пластинами треугольной и четырёхугольной формы. Полигональное представление используется для визуализации геометрической модели, вычисления инерционных характеристик, определения соударений элементов модели.

R.1.1. Управление вычислением триангуляции

На входе в методы построения полигональных представлений используется структура **MbStepData**, которая приведена на рис. R.1.1.1.

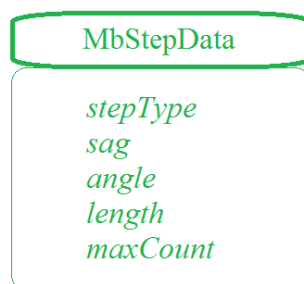


Рис. R.1.1.1.

Структура **MbStepData** объявлена в файле `mb_data.h`. Структура **MbStepData** содержит следующие данные:

- `unsigned char stepType` – поле, определяющее способ вычисления приращения параметра,
- `double sag` – максимально допустимое отклонение по прогибу,
- `double angle` – максимально допустимое отклонение по углу касательных или нормалей,
- `double length` – максимально допустимое расстояние между соседними точками,
- `unsigned int maxCount` – максимальное количество ячеек в строке или столбце триангуляционной сетки.

Структура **MbStepData** управляет плотностью сетки полигонального объекта и содержит данные, по которым вычисляется шаг по параметру при движении вдоль кривых и поверхностей модели. Поле `stepType` определяет способ вычисления приращения параметра при движении вдоль кривой или поверхности и может содержать маски перечисления `MbStepType`, объявленного в файле `mb_enum.h`:

- `ist_SpaceStep` – для визуализации геометрической формы;
- `ist_DeviationStep` – для операций построения;
- `ist_MetricStep` – для 3D принтеров;
- `ist_ParamStep` – для визуализации геометрической формы объектов с привязкой текстуры к параметрам поверхности;
- `ist_CollisionStep` – для определения столкновений элементов модели;
- `ist_MipStep` – для вычисления инерционных характеристик.

Параметр `sag` ограничивает шаг по параметру кривой или поверхности исходя из максимально допустимого отклонения полигонального объекта от оригинала по прогибу. Параметр `angle` ограничивает шаг по параметру кривой или поверхности исходя из максимально допустимого отклонения полигонального объекта от оригинала по угловому отклонению касательных кривой или нормалей поверхности в соседних точках на расстоянии шага. Параметр `length` ограничивает шаг по параметру кривой или поверхности исходя из максимально допустимого размера элемента полигонального объекта – стороны треугольника или отрезка полигона. Параметр `maxCount` ограничивает шаг по параметру кривой или поверхности исходя из максимально допустимого количества разбиений в строке или столбце триангуляционной сетки.

На входе в методы построения полигональных представлений используется структура **MbFormNote**, которая приведена на рис. R.1.1.2.

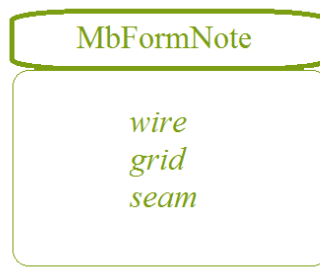


Рис. R.1.1.2.

Структура **MbFormNote** объявлена в файле `mb_data.h`. Структура **MbFormNote** содержит следующие данные:

- `bool wire` – флаг построения полигонного объекта,
- `bool grid` – флаг построения полигонального объекта,
- `bool seam` – флаг использования шовных ребер.

Структура **MbFormNote** определяет способ построения полигонального объекта: при `wire==true` полигональный объект заполняется ломаными линиями, при `grid==true` полигональный объект заполняется триангуляцией. Параметр `seam` определяет способ представления швов в полигональном объекте: при `seam==true` триангуляция не замыкается на швах и шовные ребра обрабатываются как обычные ребра – в триангуляции их точки считаются краевыми и для них строятся полигоны, при `seam==false` триангуляция замыкается на швах и шовные ребра игнорируются – триангуляции строятся замкнутыми, а полигоны для швов не строятся.

R.1.2. Построение полигонального объекта

Виртуальный метод объектов геометрической модели

[MbItem](#) *

[MbItem::CalculateMesh](#) (const MbStepData & *stepData*,
const MbFormNote & *note*,
MbRegDuplicate * *iReg*) const

строит полигональный объект, аппроксимирующий рассматриваемый объект геометрической модели.

Входными параметрами метода являются:

- *stepData* – данные для вычисления шага аппроксимации,
- *note* – способ построения полигонального объекта,
- *iReg* – регистратор копируемых объектов.

В случае успешной работы метод возвращает указатель вновь построенный объект геометрической модели [MbItem](#)*, в противном случае возвращает ноль.

Метод объявлен в файле `item.h` и заголовочных файлах наследников [MbItem](#).

Рассматриваемый метод аппроксимирует объекты модели, создавая полигональные копии с аналогичной структурой. Для тела, проволочного каркаса и точечного каркаса рассматриваемый метод создаст полигональный объект [MbMesh](#), аппроксимирующий исходный объект, и вернет указатель на созданный объект. Каждая триангуляция `grids[i]` объекта [MbMesh](#) будет аппроксимировать *i*-ую грань, каждый полигон `wires[i]` объекта [MbMesh](#) будет аппроксимировать *i*-ое ребро, каждый апекс `peaks[i]` объекта [MbMesh](#) будет аппроксимировать *i*-ую вершину. Для полигонального объекта [MbMesh](#) рассматриваемый метод создаст полигональный объект копию. Для вставки рассматриваемый метод создаст вставку с объектом, который создаст этот же метод для содержимого вставки. Для сборочной единицы рассматриваемый метод создаст сборочную единицу с объектами, которые создаст этот же метод для объектов сборочной единицы.

Параметр *stepData* управляет густотой сетки полигонального объекта и содержит данные для вычисления шага при движении вдоль кривых и поверхностей объекта модели. Параметр *note* определяет способ построения полигонального объекта. Параметры *stepData* и *note* описаны в параграфе [R.1.1. Управление вычислением триангуляции](#). При `note.wire==true` рассматриваемый метод заполняет множество указателей на полигоны `mesh.wires`; при `note.grid==true` рассматриваемый

метод заполняет множество указателей на триангуляции **mesh.grids** (для граней и поверхностей), множество указателей на полигоны **mesh.wires** (для рёбер), множество указателей на апексы **mesh.peakes** (для вершин).

Параметр *iReg* может быть равен нулю. Он служит для передачи вложенным методам информации о уже обработанных объектах.

На рис. R.1.2.1 приведен полигональный объект тела, построенный с параметрами *note.wire*==false, *note.grid*==true, содержащий триангуляции, полигоны и апексы. На рис. R.1.2.2 приведен полигональный объект тела, построенный с параметрами *note.wire*==true, *note.grid*==false, содержащий только полигоны.

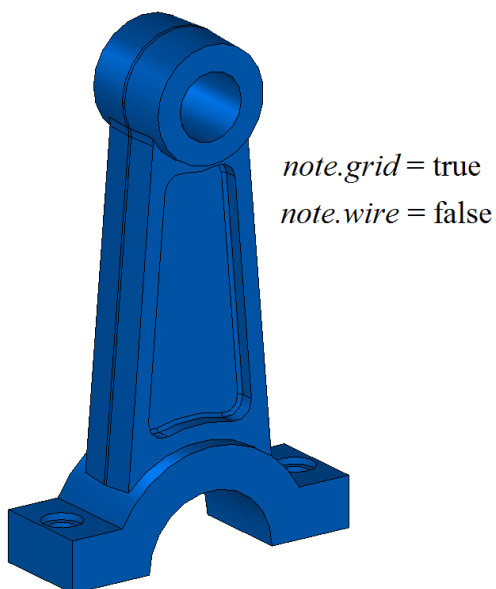


Рис. R.1.2.1.

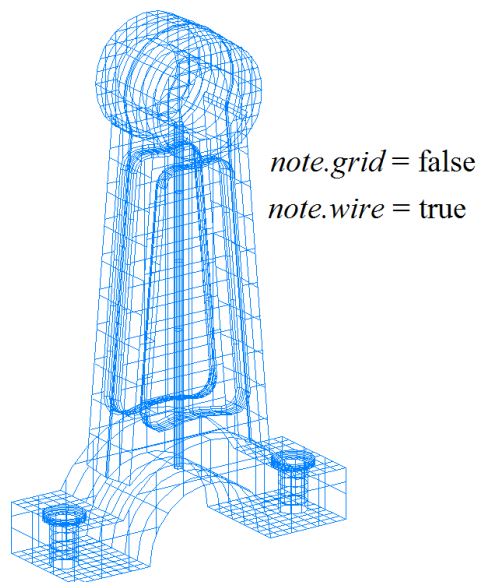


Рис. R.1.2.2.

На рис. R.1.2.3 приведен полигональный объект сборочной единицы, построенный с параметрами *note.wire*==false, *note.grid*==true, состоящий из сборочной единицы полигональных объектов, аппроксимирующих детали.

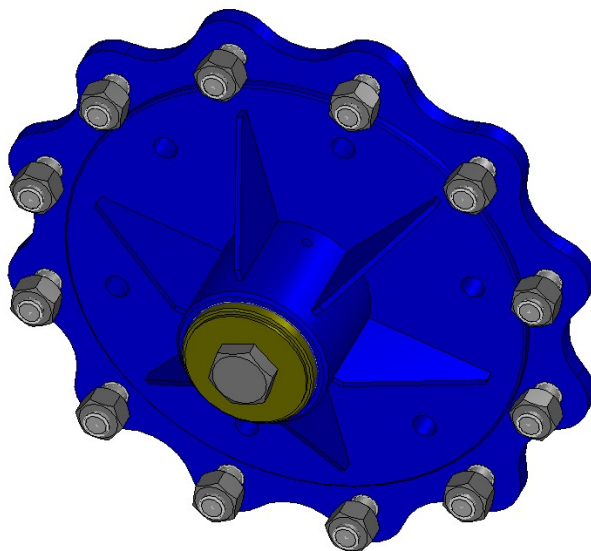


Рис. R.1.2.3.

Рассматриваемый метод используется для визуализации объектов геометрической модели. Полигональные объекты легко трансформируются, для них можно быстро найти пересечение с

прямой линией. На экране рисуется именно полигональная копия объекта геометрической модели, а объект в исходном представлении остаётся за кадром.

R.1.3. Добавление полигонального объекта

Виртуальный метод объектов геометрической модели

bool

```
MbItem::AddYourMesh ( const MbStepData & stepData,  
                      const MbFormNote & note,  
                      MbMesh & mesh ) const
```

строит и добавляет свою полигональную копию в присланный полигональный объект **mesh**.

Входными параметрами метода являются:

- *stepData* – данные для вычисления шага аппроксимации,
- *note* – способ построения полигонального объекта.

Выходным параметром метода является полигональный объект **mesh**.

В случае успешной работы метод возвращает true.

Метод объявлен в файле `item.h` и заголовочных файлах наследников [MbItem](#).

Рассматриваемый метод аппроксимирует объекты модели полигональными копиями и добавляет их в присланный объект **mesh**. Для вставки рассматриваемый метод создаст полигональную копию содержимого вставки, трансформирует её в глобальную систему координат и добавит в присланный объект **mesh**. Для сборочной единицы рассматриваемый метод создаст полигональную копию содержимого сборочной единицы, трансформирует её в глобальную систему координат и добавит в присланный объект **mesh**.

Так же, как в методе [CalculateMesh](#), параметр *stepData* управляет густотой сетки полигонального объекта, а параметр *note* определяет способ построения полигонального объекта. Параметры *stepData* и *note* описаны в параграфе [R.1.1. Управление вычислением триангуляции](#). В отличие от вышеупомянутого метода рассматриваемый метод для сложных объектов создает единый полигональный объект. На рис. R.1.3.1 приведен полигональный объект показанной на рис. R.1.2.3 сборочной единицы, построенный с рассматриваемым методом.

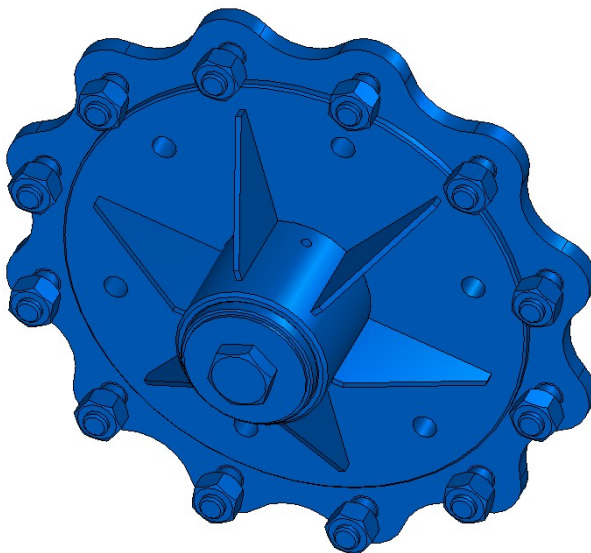


Рис. R.1.3.1.

R.1.4. Построение полигонов объекта

Виртуальный метод трехмерных геометрических объектов

void

[MbSpaceItem::CalculateWire](#) (double *sag*,
[MbMesh](#) & *mesh*)

заполняет присланный полигональный объект **mesh** набором полигонов, аппроксимирующих геометрический объект.

Входным параметром метода является *sag* – максимально допустимое отклонение полигонального элемента от оригинала по прогибу.

Выходным параметром метода является полигональный объект **mesh**.

Метод объявлен в файле `space_item.h` и заголовочных файлах наследников [MbSpaceItem](#).

Полигональный объект описан в параграфе [0.8.5. Полигональный объект MbMesh](#). Параметр *sag* определяет максимально допустимое расстояние между объектом и ломаной линией, проходящей по точкам полигонов. Рассматриваемый метод заполняет только множество указателей на полигоны **mesh.wires** (ломаные линии).

Кривую рассматриваемый метод аппроксимирует одним полигоном. Контур рассматриваемый метод аппроксимирует несколькими полигонами, каждый из которых проходит по соответствующему сегменту контура. На рис. R.1.4.1 приведена кривая и ее полигональный объект, состоящий из одного полигона.

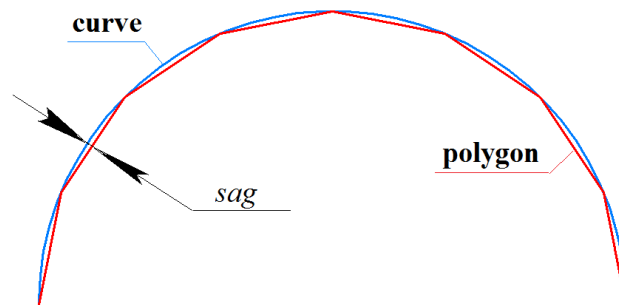


Рис. R.1.4.1.

Поверхность рассматриваемый метод аппроксимирует набором полигонов, проходящих по *u*-линиям, *v*-линиям и по границе поверхности. На рис. R.1.4.2 приведен полигональный объект поверхности, состоящий из нескольких ломаных линий.

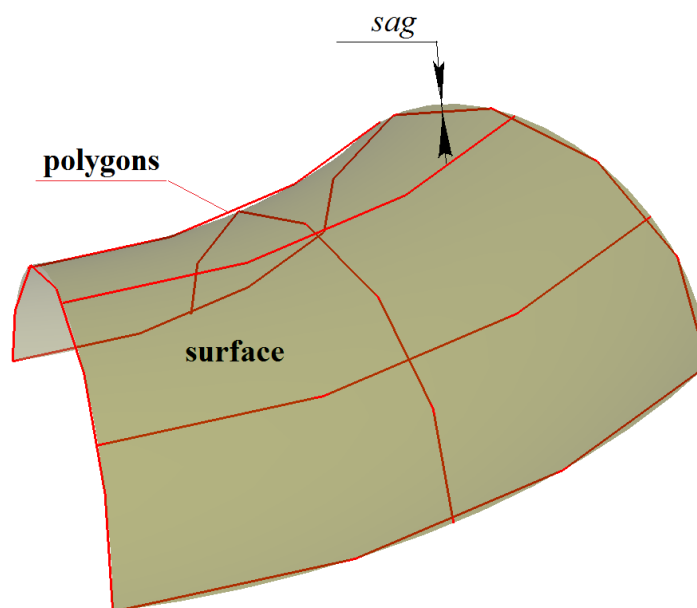


Рис. R.1.4.2.

Тело рассматриваемый метод аппроксимирует набором ломаных линий, проходящих внутри грани вдоль *u*-линий и *v*-линий поверхностей граней, и набором ломаных, аппроксимирующих кривые, на которых базируются ребра тела. На рис. R.1.4.3 справа приведен полигональный объект тела.

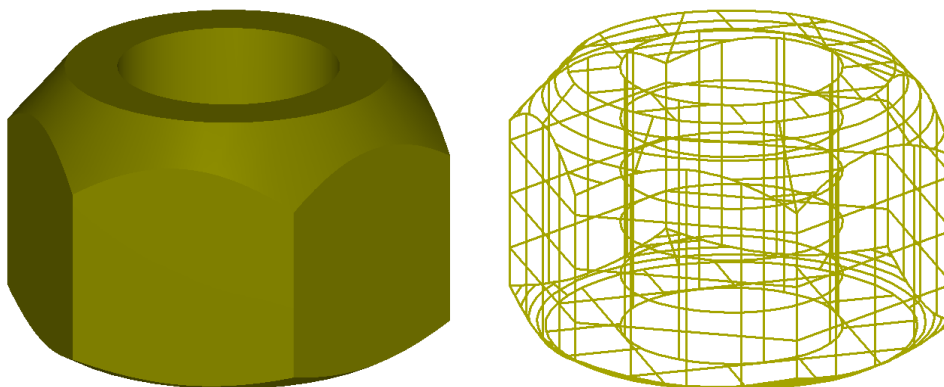


Рис. R.1.4.3.

Для объектов геометрической модели рассматриваемый метод работает аналогично методу [CalculateMesh](#) при `stepData.stepType==ist_SpaceStep`, `stepData.sag==sag`, `note.wire==true`, `note.seam==true`.

R.1.5. Построение триангуляции грани

Метод

void

```
CalculateGrid ( const MbFace & face,  
               const MbStepData & stepData,  
               bool edgePoints,  
               MbGrid & grid,  
               bool dualSeams = true )
```

аппроксимирует грань треугольными и четырёхугольными пластинами.

Входными параметрами метода являются:

- **face** – грань,
- *stepData* – данные для вычисления шага аппроксимации,
- *edgePoints* – флаг использования пространственных точек,
- *dualSeams* – флаг обработки швов.

Выходным параметром метода является триангуляция **grid**.

Метод объявлен в файле `tri_face.h`.

Параметр *stepData* управляет густотой триангуляции и содержит данные для вычисления шага при движении вдоль поверхности грани. Параметр *stepData* описан в параграфе [R.1.1. Управление вычислением триангуляции](#). Для различных значений *stepData.stepType* заполняются различные данные триангуляции **grid**. Если *stepData.stepType* содержит маску *ist_MipStep*, то заполняется множество **grid.params**. Если *stepData.stepType* содержит маску *ist_CollisionStep* или *ist_ParamStep*, то заполняются множества **grid.params**, **grid.points** и **grid.normals**. В остальных случаях заполняются множества **grid.params** и **grid.points**.

На рис. R.1.5.1 приведена триангуляция плоской грани с маской *ist_SpaceStep*.

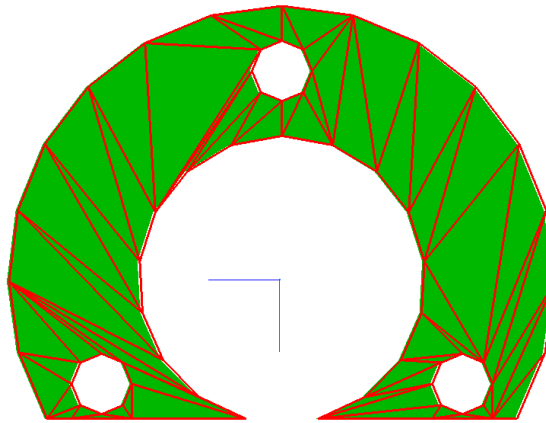


Рис. R.1.5.1.

На рис. R.1.5.2 приведена триангуляция криволинейной грани с маской *ist_SpaceStep*.

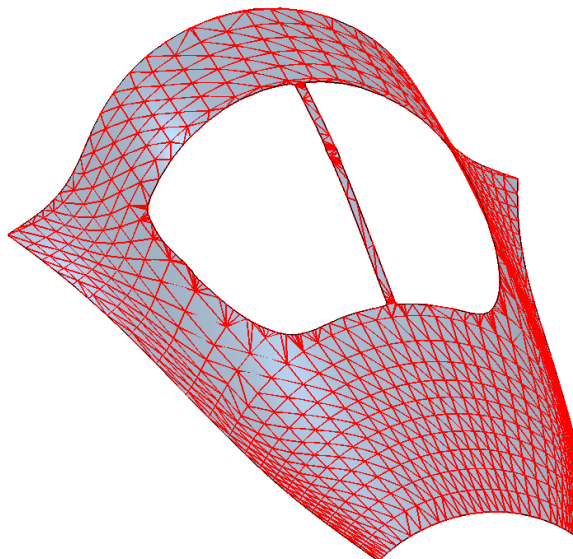


Рис. R.1.5.2.

Рассматриваемый метод вызывается при построении полигональных объектов методами **CalculateMesh** и **AddYourMesh**, если *note.grid==true*.

R.1.6. Построение триангуляции тела

Метод

void

CalculateGrid (const [MbSolid](#) & **solid**,
const MbStepData & *stepData*,
RPAarray<[MbGrid](#)> & **grids**)

строит множество триангуляций для граней тела.

Входными параметрами метода являются:

- **solid** – тело,
- *stepData* – данные для вычисления шага аппроксимации.

Выходным параметром метода является множество **grids**.

Метод не возвращает значений.

Метод объявлен в файле *mip_solid_area_volume.h*.

Рассматриваемый метод аппроксимирует грани тела **solid** триангуляциями **grids**. При вызове метода множество **meshs** должно быть пустым. При вызове метода множество **grids** должно быть пустым. Каждой *i*-ой грани тела **solid** соответствует построенный объект **grids[i]**.

Параметр *stepData* управляет густотой сетки полигональных объектов и содержит данные для вычисления шага при движении вдоль кривых и поверхностей тел. Параметр *stepData* описан в параграфе [R.1.1. Управление вычислением триангуляции](#).

R.1.7. Построение полигональных объектов множества тел

Метод

void

```
CalculateGrid ( const RPAArray<MbSolid> & solids,  
               const MbStepData & stepData,  
               RPAArray<MbMesh> & meshs )
```

строит множество полигональных объектов для множества тел.

Входными параметрами метода являются:

- **solids** – множество тел,
- *stepData* – данные для вычисления шага аппроксимации.

Выходным параметром метода является множество **meshs**.

Метод не возвращает значений.

Метод объявлен в файле `mip_solid_area_volume.h`.

Рассматриваемый метод аппроксимирует тела **solids** полигональными объектами **meshs**. При вызове метода множество **meshs** должно быть пустым. Каждому телу **solids[i]** соответствует построенный объект **meshs[i]**.

Параметр *stepData* управляет густотой сетки полигональных объектов и содержит данные для вычисления шага при движении вдоль кривых и поверхностей тел. Параметр *stepData* описан в параграфе [R.1.1. Управление вычислением триангуляции](#).

R.2. ПОСТРОЕНИЕ ПЛОСКИХ ПРОЕКЦИЙ

Для построения плоской проекции моделируемого объекта геометрическое ядро C3D использует каркасную модель. Каркасную модель получим из граничного представления геометрической модели, взяв рёбра и добавив вместо граней их линии очерка. Линии очерка проходят внутри граней и делят их на видимые и невидимые из точки наблюдения части. Плоские проекции более информативны, если в каркасной модели скрыты невидимые из точки наблюдения рёбра и линии очерка.

R.2.1. Данные построения плоских проекций

На входе в метод построения плоских проекций для передачи тел используется структура **MbLump**, которая приведена на рис. R.2.1.1.



Рис. R.2.1.1.

Структура **MbLump** объявлена в файле `lump.h`. Структура **MbLump** содержит указатель на тело **solid**, матрицу преобразования тела из локальной системы координат **from**, идентификационные параметры тела *component* и *identifier*.

Для передачи вспомогательных объектов в метод построения плоских проекций используется класс **MbProjectionsObjects**, который приведен на рис. R.2.1.2.

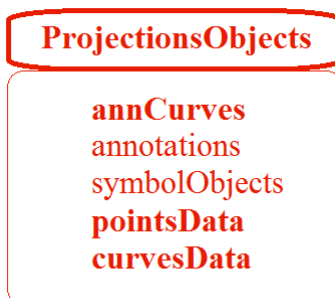


Рис. R.2.1.2.

Класс **MbProjectionsObjects** объявлен в файле `map_create.h`. Класс **MbProjectionsObjects** содержит следующие данные:

- `TPointer<PArray<MbAnnCurves>>` **annCurves** – аннотационные кривые,
- `TPointer<RPAArray<MbSimbolthThreadView>>` **annotations** – вспомогательные объекты,
- `TPointer<RPAArray<MbSymbol>>` **symbolObjects** – условные обозначения,
- `TPointer<RPAArray<MbSpacePoints>>` **pointsData** – точки,
- `TPointer<RPAArray<MbSpaceCurves>>` **curvesData** – кривые.

Для передачи результатов построения плоской проекции объекта используется структура **MbVEFVestiges**, которая приведена на рис. R.2.1.3.

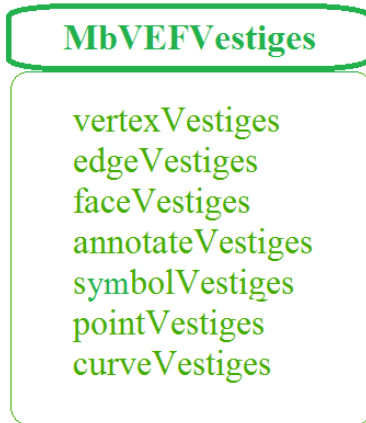


Рис. R.2.1.3.

Структура **MbVEFVestiges** объявлена в файле `map_vestiges.h`. Структура **MbVEFVestiges** содержит проекции элементов объекта на проекционную плоскость, объединенные в группы. Каждый элемент группы проекций состоит из построенной проекции, указателя на породивший проекцию объект, сведений о видимости этой проекции и другой информации о проекции. Структура **MbVEFVestiges** содержит следующие группы:

- PArray<MbVertexVestige> vertexVestiges – множество проекций вершин,
- PArray<MbEdgeVestige> edgeVestiges – множество проекций ребер,
- PArray<MbFaceVestige> faceVestiges – множество проекций граней,
- PArray<MbAnnotationVestige> annotateVestiges – множество проекций аннотационных объектов,
- PArray<MbSymbolVestige> symbolVestiges – множество проекций условных обозначений,
- PArray<MbVertexVestige> pointVestiges – множество проекций точек,
- PArray<MbEdgeVestige> curveVestiges – множество проекций кривых.

R.2.2. Построение плоской проекции модели

Метод

void

```
GetVestiges ( const MbPlacement3D & place,
             double znear,
             const RArray<MbLump> & lumps,
             const MbProjectionsObjects & objects,
             MbVEFVestiges & result,
             bool invisible,
             VERSION version )
```

выполняет построение плоской проекции множества тел и других объектов.

Входными параметрами метода являются:

- **place** – проекционная плоскость,
- *znear* – параметр точки наблюдения,
- **lumps** – проецируемые тела в локальных системах координат,
- **objects** – прочие проецируемые объекты,
- *invisible* – признак построения невидимых линий,
- *version* – версия построения, последняя версия `Math::DefaultMathVersion()`.

Выходным параметром метода является **result** – структура, содержащая информацию о проекции.

Метод не возвращает значений.

Метод объявлен в файле `map_create.h`.

Проекционной плоскостью является плоскость XY локальной системы координат **place**.

Параметр *znear* определяет тип изображения. При *znear*=0 строится параллельная проекция объектов. Параметр **lumps** (рис. R.2.1.1) содержит тела и матрицы преобразования их из локальной

системы координат. Параметр **objects** (рис. R.2.1.2) содержит вспомогательные объекты, необходимые для оформления проекции: вспомогательные точки, кривые, условные обозначения, объекты аннотации. Параметр *invisible* сообщает о необходимости строить невидимые линии. В некоторых случаях при отказе от построения невидимых линий метод работает заметно быстрее.

На рис. R.2.2.1 приведено тело, линии проекции которого на плоскость, параллельную плоскости экрана, приведены на рис. R.2.2.2. На рис. R.2.2.3 приведены только видимые линии проекции тела, показанного на рис. R.2.2.1.

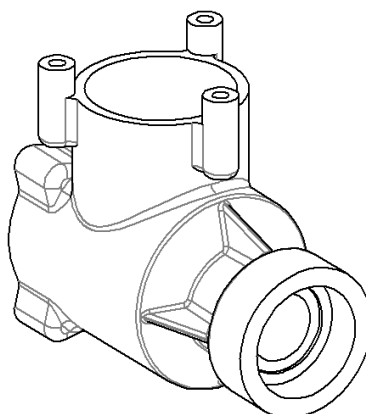
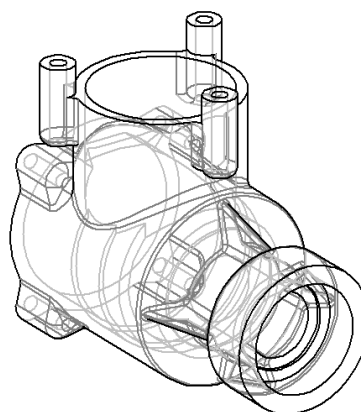
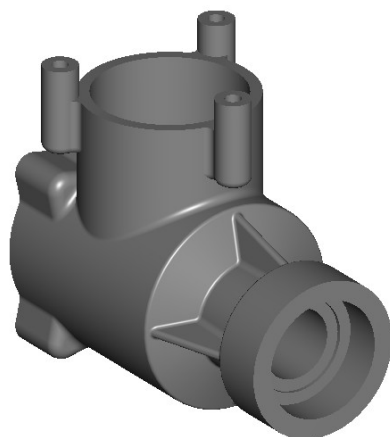


Рис. R.2.2.1.

Рис. R.2.2.2.

Рис. R.2.2.3.

Последний параметр рассматриваемого метода используется для поддержки предыдущих версий построения.

Метод

void

```
VisualLinesMapping ( const MbPlacement3D & place,
                     double znear,
                     const RArray<MbLump> & lumps,
                     MbVEFVestiges & result,
                     bool invisible = true )
```

выполняет построение плоской проекции множества тел и отличается от метода [GetVestiges](#) отсутствием дополнительных объектов **objects**.

R.2.3. Построение полигональной проекции тел

Метод

void

```
HiddenLinesMapping ( const RArray<MbLump> & lumps,
                     const MbPlacement3D & place,
```

```

double                znear,
double                sag,
PArray<MbPolygon3DSolid> & visibleEdges,
PArray<MbPolygon3DSolid> & hiddenEdges,
PArray<MbPolygon3DSolid> & visibleTangs,
PArray<MbPolygon3DSolid> & hiddenTangs )

```

выполняет построение полигональной проекции множества тел на заданную плоскость.

Входными параметрами метода являются:

- **lumps** – проецируемые тела в локальных системах координат,
- **place** – проекционная плоскость,
- *znear* – параметр точки наблюдения,
- *sag* – максимально допустимое отклонение по прогибу.

Выходными параметрами метода являются:

- *visibleEdges* – полигоны видимых негладких ребер,
- *hiddenEdges* – полигоны невидимых негладких ребер,
- *visibleTangs* – полигоны видимых гладких ребер,
- *hiddenTangs* – полигоны невидимых гладких ребер.

Метод не возвращает значений.

Метод объявлен в файле `map_create.h`.

Проекционной плоскостью является плоскость XY локальной системы координат **place**.

Параметр **lumps** (рис. R.2.1.1) содержит тела и матрицы преобразования их из локальной системы координат. Параметр *znear* определяет тип изображения. При *znear*=0 строится параллельная проекция объектов. Класс `MbPolygon3DSolid` содержит номер компонента и указатель на полигон, состоящий из множества точек, последовательное соединение которых даст аппроксимацию проекции ребра на плоскость. Точность аппроксимации определяет параметр *sag*. Полигоны *visibleEdges* и *hiddenEdges* представляют собой соответственно видимые и невидимые полигональные проекции негладких ребер на плоскость XY локальной системы координат **place**. Полигоны *visibleTangs* и *hiddenTangs* представляют собой соответственно видимые и невидимые полигональные проекции гладких ребер на плоскость XY локальной системы координат **place**.

Метод

void

```

VisualLinesMapping ( const RPAArray<MbLump> &    lumps,
                     const MbPlacement3D &      place,
                     double                znear,
                     double                sag,
                     PArray<MbPolygon3DSolid> & visibleEdges,
                     PArray<MbPolygon3DSolid> & visibleTangs )

```

выполняет построение только видимой полигональной проекции множества тел на заданную плоскость. Рассматриваемый метод выполняет те же построения, что и метод **HiddenLinesMapping**, с той разницей, что строит только видимые проекции. В некоторых случаях метод **VisualLinesMapping** работает заметно быстрее метода **HiddenLinesMapping**.

R.2.4. Построение линий очерка триангуляции

Метод

void

```

CalculateBoundsSlTFast ( const MbGrid &          grid,
                         const MbMatrix3D &     matrix,
                         bool                perspective,
                         RPAArray<MbFloatPoint3D> & points )

```

выполняет построение линий очерка триангуляции.

Входными параметрами метода являются:

- **grid** – триангуляция грани тела,
- **matrix** – матрица, определяющая вектор взгляда,
- *perspective* – признак перспективного отображения.

Выходным параметром метода является:

points – множество указателей на точки из триангуляции **grid.points**.

Метод не возвращает значений.

Метод объявлен в файле `map_create.h`.

Метод предназначен для построения полигональных линий очерка и используется для визуализации силуэта триангуляции.

R.3. ВЫЧИСЛЕНИЕ ИНЕРЦИОННЫХ ХАРАКТЕРИСТИК

Геометрическое ядро C3D вычисляет площадь поверхности, объём, положение центра масс и моменты инерции моделируемого объекта. В общем случае вычисление выполняется путем численного интегрирования. Интегрирование по объему с помощью теоремы Остроградского-Гаусса сводится к интегрированию по поверхности моделируемого объекта. Интегрирование по поверхности использует триангуляцию двумерной области определения параметров поверхности. При вычислении перечисленных характеристик модели предусмотрена возможность использования готовых данных для отдельных элементов модели.

R.3.1. Инерционные характеристики модели

Для передачи инерционных характеристик модели используется класс **InertiaProperties**, который приведен на рис. R.3.1.1.

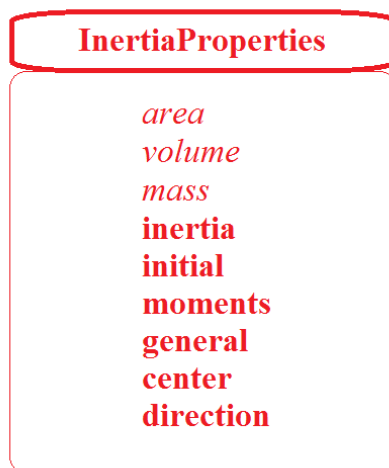


Рис. R.3.1.1.

Класс **InertiaProperties** объявлен в файле `mip_solid_mass_inertia.h`. Класс **InertiaProperties** содержит следующие данные о модели:

- `double area` – площадь поверхности,
- `double volume` – объем,
- `double mass` – масса,
- `double inertia[3]` – статические моменты в исходной системе координат,
- `double initial[3][3]` – моменты инерции в исходной системе координат,
- `double moments[3][3]` – моменты инерции в центральной системе координат,
- `double general[3]` – главные центральные моменты инерции,
- [MbCartPoint3D](#) `center` – центр масс,
- [MbVector3D](#) `direction[3]` – векторы направлений главных осей инерции.

При инициализации класса **InertiaProperties** все данные принимают нулевые значения, а координаты центра масс полагаются равными `NOT_INITIAL_DBL`. Моменты инерции **initial** вычисляются в системе координат, в которой описана модель. Моменты инерции **moments** вычисляются в системе координат, начало которой находится в точке **center**, а координатные оси совпадают с осями исходной системы координат, в которой описана модель. Моменты инерции **general** вычисляются в главной центральной системе координат, начало которой находится в точке **center**, а координатные оси получены вычислением и совпадают с главными осями инерции модели. Центробежные моменты инерции в главных системах координат отсутствуют.

Векторы **direction** дают направления главных осей инерции. Если все главные моменты инерции **general[i]** $i=1,2,3$ разные, то все векторы **direction[i]** $i=1,2,3$ не равны нулю. Если все главные

моменты инерции **general**[*i*] *i*=1,2,3 одинаковые, то все векторы **direction**[*i*] *i*=1,2,3 равны нулю и главными направлениями могут служить любые три взаимно ортогональных вектора. Если два из трех главных моментов инерции равны, например **general**[*j*]=**general**[*k*], то два из трёх векторов равны нулю **direction**[*j*]=**direction**[*k*]=0, а не равный нулю вектор **direction**[*i*] определяет направление главной оси инерции, момент относительно которой отличается от других, двумя другими главными направлениями могут служить любые два взаимно ортогональных и ортогональных не равному нулю вектору **direction**[*i*] вектора.

Возможность использования готовых данных для отдельных элементов модели позволяют реализовать классы **SolidMIAttire** и **AssemblyMIAttire**, которые приведены на рис. R.3.1.2 и рис. R.3.1.3. Классы объявлены в файле `mip_solid_mass_inertia.h`.

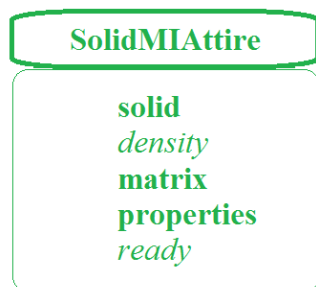


Рис. R.3.1.2.



Рис. R.3.1.3.

Класс **SolidMIAttire** содержит следующие данные:

- `const MbSolid & solid` – тело,
- `double density` – плотность или удельная масса на единицу площади,
- `MbMatrix3D matrix` – матрица преобразования тела в систему ближайшей сборки,
- `InertiaProperties * properties` – назначенные инерционные характеристики тела, могут быть ноль или заданы не полностью,
- `bool ready` – признак, показывающий, что характеристики не требуется считать.

Класс **AssemblyMIAttire** содержит следующие данные:

- `RPAarray<AssemblyMIAttire> assemblies` – множество сборочных единиц следующего уровня,
- `RPAarray<SolidMIAttire> solids` – тела сборочной единицы,
- `MbMatrix3D matrix` – матрица преобразования сборочной единицы в систему ближайшей сборки,
- `InertiaProperties * properties` – назначенные инерционные характеристики сборочной единицы, могут быть нулями или заданы не полностью,
- `bool ready` – признак, показывающий, что характеристики не требуется считать.

Если экземпляры классов **SolidMIAttire** и **AssemblyMIAttire** содержат не нулевые **properties** и `ready==true`, то инерционные характеристики соответствующего объекта не будут вычисляться, а результат вычисления будет содержать данные из **properties**.

Если экземпляры классов **SolidMIAttire** и **AssemblyMIAttire** содержат не нулевые **properties** и `ready==false`, то результат вычисления будет содержать данные из **properties**, которые не равны `NULL_EPSILON` или `NOT_INITIAL_DBL`. Данные, которые в **properties** равны `NULL_EPSILON` или `NOT_INITIAL_DBL`, будут вычислены. Рассматриваемые классы позволяют смешивать рассчитанные данные с назначенными.

R.3.2. Инерционные характеристики тела

Функция

`void`

MassInertiaProperties (`const MbSolid * solid`,
`double density`,
`double deviateAngle`,
`InertiaProperties & properties`,
`IfProgressIndicator * progress = 0`)

вычисляет площадь поверхности, объем, массу, центр масс и моменты инерции тела.

Входными параметрами метода являются:

- **solid** – тело,
- *density* – плотность или удельная масса на единицу площади,
- *deviateAngle* – параметр управления точностью расчёта.

Выходными параметрами метода являются:

- **properties** – рассчитанные инерционные характеристики,
- *progress* – индикатор выполнения расчета.

Метод не возвращает значения.

Метод объявлен в файле `mip_solid_mass_inertia.h`.

Для замкнутого тела **solid** параметр *density* определяет плотность тела. Для незамкнутого тела **solid** параметр *density* определяет удельную массу единицы площади тела. Вычисление в общем случае выполняется численно и использует триангуляцию области определения параметров поверхности граней. Триангуляция параметрической области граней выполняется методом **CalculateGrid**, описанным в параграфе [R.1.4. Построение полигонов объекта](#), при значениях *stepData.stepType=ist MipStep* и *stepData.angle=deviateAngle*. Параметр *deviateAngle* определяет максимально допустимый угол между нормальными соседних треугольников и четырехугольников триангуляции поверхности. Параметр *deviateAngle* управляет точностью расчёта и от его значения зависит время расчета. Параметр *deviateAngle* должен находиться в пределах $0.01 \leq \textit{deviateAngle} \leq 0.35$ радиан. Следует заметить, что при малых значениях *deviateAngle* для сложных моделей требуется большое время расчета.

Инерционные характеристики **properties** описаны в параграфе [R.1.4. Построение полигонов объекта](#). Параметр *progress* выдает информацию о выполнении в процессе расчета и может использоваться для остановки вычислений.

R.3.3. Инерционные характеристики множества тел

Функция

void

```
MassInertiaProperties ( const RPAArray<MbSolid> & solids,  
                        const SArray<double> & densities,  
                        const SArray<MbMatrix3D> & matrices,  
                        const PArray<InertiaProperties> & mpSolids,  
                        double deviateAngle,  
                        InertiaProperties & properties,  
                        IfProgressIndicator * progress = 0 )
```

вычисляет площадь поверхности, объем, массу, центр масс и моменты инерции множества тел.

Входными параметрами метода являются:

- **solids** – множество тел,
- *densities* – множество плотностей или удельных масс на единицу площади,
- **matrices** – множество матриц преобразования тел в общую систему координат,
- **mpSolids** – множество имеющихся характеристик тел (может содержать нули),
- *deviateAngle* – параметр управления точностью расчёта.

Выходными параметрами метода являются:

- **properties** – рассчитанные инерционные характеристики,
- *progress* – индикатор выполнения расчета.

Метод не возвращает значения.

Метод объявлен в файле `mip_solid_mass_inertia.h`.

Количество элементов в множествах *densities*, **matrices**, **mpSolids** должно совпадать с количеством тел в множестве **solids**. Второй параметр определяет плотность тела для замкнутых тел или удельную массу единицы площади для незамкнутых тел. Параметр **matrices** содержит матрицы преобразования тел в систему координат, в которой требуется выполнить расчет. Параметр **mpSolids** содержит назначенные характеристики соответствующих тел, которыми необходимо заменить соответствующие характеристики, полученные в результате расчета. Рассматриваемый метод может использоваться для вычисления инерционных характеристик сборочной единицы, инерционные характеристики элементов которой были вычислены ранее в локальных системах координат.

Вычисление в общем случае выполняется численно и использует триангуляцию области определения параметров поверхности граней. Триангуляция параметрической области граней выполняется методом **CalculateGrid**, описанным в параграфе [R.1.4. Построение полигонов объекта](#), при значениях `stepData.stepType=ist_MipStep` и `stepData.angle=deviateAngle`. Параметр `deviateAngle` определяет максимально допустимый угол между нормальными соседних треугольников и четырёхугольников триангуляции поверхности. Параметр `deviateAngle` управляет точностью расчёта и от его значения зависит время расчёта. Параметр `deviateAngle` должен находиться в пределах $0.01 \leq \text{deviateAngle} \leq 0.35$ радиан. Следует заметить, что при малых значениях `deviateAngle` для сложных моделей требуется большое время расчёта.

Инерционные характеристики **properties** описаны в параграфе [R.3.1. Инерционные характеристики модели](#). Параметр `progress` выдает информацию о выполнении в процессе расчёта и может использоваться для остановки вычислений.

R.3.4. Инерционные характеристики модели

Функция

void

MassInertiaProperties (const [AssemblyMIAttire](#) & **assembly**,
double `deviateAngle`,
InertiaProperties & **properties**,
IfProgressIndicator * `progress` = 0)

вычисляет площадь поверхности, объем, массу, центр масс и моменты инерции модели, представленной в виде сборочной единицы.

Входными параметрами метода являются:

- **assembly** – сборочная единица, которая может содержать рассчитанные ранее характеристики,
- `deviateAngle` – параметр управления точностью расчёта.

Выходными параметрами метода являются:

- **properties** – рассчитанные инерционные характеристики,
- `progress` – индикатор выполнения расчёта.

Метод не возвращает значения.

Метод объявлен в файле `mip_solid_mass_inertia.h`.

Параметр **assembly** представляет собой аналог сборочной единицы, элементы которой есть такие же сборочные единицы в локальных системах координат и тела с заданной плотностью в локальных системах координат. Элементы сборочной единицы могут иметь рассчитанные ранее инерционные характеристики и параметры управления ими. При наличии у элемента рассчитанных ранее характеристик последние будут использоваться в общей сумме, что сократит общее время расчёта.

Параметр `deviateAngle` управляет точностью расчёта и от его значения зависит время расчёта. Параметр `deviateAngle` определяет максимально допустимый угол между нормальными соседних треугольников и четырёхугольников триангуляции поверхности. Параметр `deviateAngle` должен находиться в пределах $0.01 \leq \text{deviateAngle} \leq 0.35$ радиан. Следует заметить, что при малых значениях `deviateAngle` для сложных моделей требуется большое время расчёта.

Инерционные характеристики **properties** описаны в параграфе [R.3.1. Инерционные характеристики модели](#). Параметр `progress` выдает информацию о выполнении в процессе расчёта и может использоваться для остановки вычислений.

R.3.5. Вычисление площади поверхности

Метод

double

CalculateArea (const RPAArray<[MbFace](#)> & **faces**,
double `deviateAngle`)

вычисляет площадь поверхности набора граней.

Входными параметрами метода являются:

- **faces** – набор граней,

- *deviateAngle* – параметр управления точностью расчёта.

Метод возвращает площадь набора граней.

Метод объявлен в файле `mip_solid_mass_inertia.h`.

Параметр *deviateAngle* управляет точностью расчёта, и от его значения зависит время расчета. Параметр *deviateAngle* должен находиться в пределах $0.01 \leq \textit{deviateAngle} \leq 0.35$ радиан. Вычисление в общем случае выполняется численно и использует триангуляцию области определения параметров поверхности граней. Триангуляция параметрической области граней выполняется методом [CalculateGrid](#), описанным в параграфе [R.1.4. Построение полигонов объекта](#), при значениях *stepData.stepType=ist_MipStep* и *stepData.angle=deviateAngle*. Параметр *deviateAngle* определяет максимально допустимый угол между нормальными соседних треугольников и четырёхугольников триангуляции поверхности.

Метод

double

CalculateArea (const [MbFace](#) & **face**,
double *deviateAngle*)

вычисляет площадь поверхности одной грани.

Входными параметрами метода являются:

- **face** – грань,
- *deviateAngle* – параметр управления точностью расчёта.

Метод возвращает площадь грани **face**.

Метод объявлен в файле `tri_face.h`.

Параметр *deviateAngle* управляет точностью расчёта, и от его значения зависит время расчета. Параметр *deviateAngle* должен находиться в пределах $0.01 \leq \textit{deviateAngle} \leq 0.35$ радиан.

Метод

double

CalculateArea (const [MbSurface](#) & **surface**,
double *deviateAngle*);

вычисляет площадь поверхности.

Входными параметрами метода являются:

- **surface** – поверхность,
- *deviateAngle* – параметр управления точностью расчёта.

Метод возвращает площадь поверхности **surface**.

Метод объявлен в файле `mip_solid_area_volume.h`.

Параметр *deviateAngle* управляет точностью расчёта, и от его значения зависит время расчета. Параметр *deviateAngle* должен находиться в пределах $0.01 \leq \textit{deviateAngle} \leq 0.35$ радиан.

Метод

double

CalculateAreaCentre (const [MbFace](#) & **face**,
double *deviateAngle*,
bool *byOuter*,
VERSION *version*,
[MbCartPoint3D](#) & **centre**)

вычисляет площадь поверхности и центр масс грани.

Входными параметрами метода являются:

- **face** – грань,
- *deviateAngle* – параметр управления точностью расчёта,
- *byOuter* – параметр игнорирования внутренних вырезов грани,
- *version* – параметр версионирования расчета.

Выходным параметром метода является **centre** – центр масс грани.

Метод возвращает площадь грани.

Метод объявлен в файле `mip_solid_area_volume.h`.

Параметр *deviateAngle* управляет точностью расчёта, и от его значения зависит время расчета. Параметр *deviateAngle* должен находиться в пределах $0.01 \leq \textit{deviateAngle} \leq 0.35$ радиан.

Параметр *byOuter* позволяет исключить внутренние вырезы грани при расчете. Если *byOuter=true*, то в расчете считается, что внутренние вырезы грани отсутствуют. Для нормального расчета площади поверхности и центра масс грани следует положить *byOuter=false*. Параметр *version* необходим для поддержания старых версий расчета.

Метод
double
CalculateAreaCentre (const [MbFaceShell](#) & **shell**,
double *deviateAngle*,
[MbCartPoint3D](#) & **centre**)

вычисляет площадь поверхности и центр масс набора граней.

Входными параметрами метода являются:

- **shell** – набор граней,
- *deviateAngle* – параметр управления точностью расчёта.

Выходным параметром метода является **centre** – центр масс набора граней.

Метод возвращает площадь набора граней.

Метод объявлен в файле `mip_solid_area_volume.h`.

Параметр *deviateAngle* управляет точностью расчёта, и от его значения зависит время расчета.

Параметр *deviateAngle* должен находиться в пределах $0.01 \leq \textit{deviateAngle} \leq 0.35$ радиан.

R.3.6. Вычисление объема тела

Метод
double
CalculateVolume (const [MbSolid](#) & **solid**,
double *deviateAngle*)

вычисляет объем тела.

Входными параметрами метода являются:

- **solid** – тело,
- *deviateAngle* – параметр управления точностью расчёта.

Метод возвращает объем тела.

Метод объявлен в файле `mip_solid_area_volume.h`.

Параметр *deviateAngle* управляет точностью расчёта, и от его значения зависит время расчета.

Параметр *deviateAngle* должен находиться в пределах $0.01 \leq \textit{deviateAngle} \leq 0.35$ радиан.

R.4. ОПРЕДЕЛЕНИЕ СТОЛКНОВЕНИЙ ТЕЛ

Геометрическое ядро C3D предоставляет два способа оценки столкновения твердотельных объектов. Первый способ основан на алгоритмах булевой операции и предназначен для точной оценки пересечения двух неподвижных тел с вычислением ребер границы пересечения. Второй способ основан на работе с полигональным представлением модели и предназначен для динамических сцен, заданных набором тел. Второй способ рекомендуется использовать, если требуется многократная оценка столкновения твердотельных моделей, изменяющих свое положение в режиме реального времени.

R.4.1. Определение пересечения двух тел

Метод
MbResultType
InterferenceSolids([MbSolid](#) & **solid1**, [MbSolid](#) & **solid2**,
std::vector<[MbCurveEdge](#)*> * **edges**,
std::vector<ptrdiff_t> (*faceNumbers)[4])

выполняет оценку пересечения двух тел.

Входными параметрами метода являются:

- **solid1** – первое тело,
- **solid2** – второе тело,
- **edges** – ребра пересечения тел (может быть 0),
- (*faceNumbers)[4] – контейнер номеров граней (может быть 0).

Выходными параметрами метода являются:

- ребра пересечения **edges** граней тел **solid1** и **solid2**,
- номера пересекшихся граней (*faceNumbers)[4].

Метод возвращает `rt_Intersect`, если тела пересекаются, или `rt_NoIntersect`, если тела не пересекаются.

Метод объявлен в файле `cdet_bool.h`.

Метод модифицирует ребра тел **solid1** и **solid2**, поэтому если требуется сохранить тела неизменными, необходимо передавать в метод копии, сделанные с помощью [MbSolid::Duplicate\(\)](#). Модифицирование заключается в резке ребер тел ребрами пересечения.

Возвращаемое значение `rt_Intersect` означает, что пара тел имеет некоторый объем пересечения или некоторую площадь соприкосновения. Функция вернет `rt_NoIntersect`, если тела не имеют объема пересечения. Касание в одной точке не считается пересечением, так как имеет нулевой объем пересечения.

Если указатель на контейнер ребер **edges** не равен NULL, то в **edges** будут помещены ребра пересечения тел **solid1** и **solid2**.

Если указатель на контейнер номеров граней `faceNumbers` не равен NULL, то в `faceNumbers[0]` и `faceNumbers[1]` будут помещены номера пересекшихся граней соответственно первого и второго тела, кроме того, в `faceNumbers[2]` и `faceNumbers[3]` будут помещены номера соприкасающихся граней соответственно первого и второго тела.

R.4.2. Определение столкновений в наборе тел

Данный способ оценки столкновений применяется в случае, когда необходимо осуществить серию тестов на столкновение для одного и того же набора тел, меняющих свое положение во времени. На рис. R.4.2.1 изображен механизм, представляющий собой сборочную единицу из семи деталей, положение которых подчинено кинематическим связям. Требуется оценить границы, в пределах которых детали не мешают друг другу при работе механизма. На рис. R.4.2.2. приведено положение деталей механизма, при котором возникают столкновения между ними. Красным цветом на рис. R.4.2.2. показаны грани тел, встретившихся друг с другом при выходе за границы свободного движения.

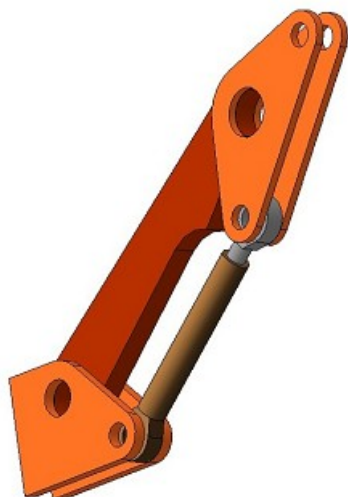


Рис. R.4.2.1

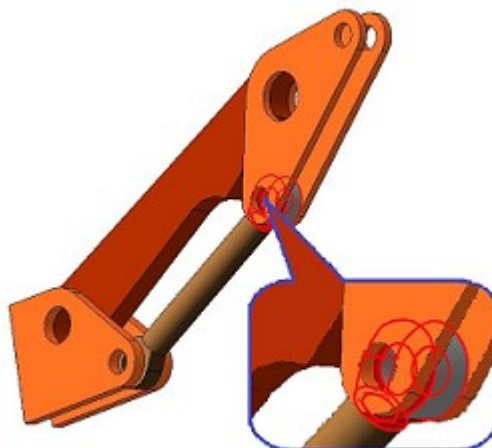


Рис. R.4.2.2

Для контроля столкновения между телами из некоторого набора необходимо завести экземпляр класса `MbCollisionDetectionUtility`, с помощью метода `AddItem` добавить тела в набор для контроля столкновений. В момент времени, когда одно или несколько тел изменили положение, приложение сообщает телам набора новое положение вызовом `Reposition` и определяет, имеются ли в геометрической модели коллизии с помощью вызова `CheckCollisions`.

Класс для контроля столкновений `MbCollisionDetectionUtility` объявлен в файле `cdet_utility.h`, вспомогательные типы данных объявлены в файле `cdet_data.h`.

Метод
`cdet_item`

```
MbCollisionDetectionUtility::AddItem ( const MbSolid & solid,
                                       const MbPlacement3D & place,
                                       cdet_app_item applItem = CDET_APP_NULL )
```

добавляет твердое тело **solid** в набор для контроля столкновений.

Входными параметрами метода являются:

- **solid** – твердотельная модель,
- **place** – локальная система координат, задающая положение тела **solid** в пространстве,
- *applItem* – геометрический объект клиентского приложения, которому принадлежит тело.

Метод возвращает дескриптор объекта для контроля столкновений.

Параметр *applItem* используется для идентификации объектов модели клиентского приложения при их столкновении. Если параметр *applItem* не определен, то идентификация осуществляется только на основе дескриптора (тип `cdet_item`), с которым тело было зарегистрировано в наборе. Предполагается, что одному объекту *applItem*, может принадлежать не одно тело `MbSolid`, то есть для одного и того же объекта с *applItem* метод `AddItem` может быть вызван более одного раза. В этом случае тела из группы, добавленной для одного и того же *applItem*, не проверяются на столкновения между собой.

Метод
`cdet_item`

```
MbCollisionDetectionUtility::RemoveItem ( cdet_item cdlItem )
```

удаляет твердое тело из набора для контроля столкновений.

Входным параметром метода является *cdlItem* – дескриптор объекта для контроля столкновений, ранее добавленный методом `AddItem`.

Метод возвращает дескриптор объекта.

Метод
`cdet_result`

```
MbCollisionDetectionUtility::CheckCollisions ( cdet_query cdQuery = defaultQuery )
```

проверяет столкновения между объектами из контрольного набора тел.

Входным параметром метода является `cdQuery` – структура запроса деталей столкновения.

Метод возвращает код результата поиска столкновений.

Метод `CheckCollisions` ищет столкновения между телами, добавленными в контрольный набор с помощью вызова `AddItem`. Метод вернет значение `CDET_RESULT_Intersected`, если в данном наборе

тел найдено столкновение. Если столкновений не обнаружено, то метод вернет `CDET_RESULT_NoIntersection`. Код `cdet_result` является основным результатом работы алгоритма **CheckCollisions**, однако с помощью структуры **cdQuery** можно заказать более детальную информацию о результатах поиска столкновений, а также ограничить или даже прервать поиск, когда это требуется. Если параметр **cdQuery** не определен, то алгоритм поиска не тратит время на выявление всех подробностей столкновения и останавливается, как только будет найдена первая пара пересекающихся граней.

Если для одного и того же объекта `cdet_app_item` добавлены два и более тел, то столкновения в пределах такой группы не проверяется.

R.4.3. Запросы на поиск столкновений

Кроме самого факта столкновения тел иногда требуется получить список граней, которые участвуют в столкновении, как показано на рис. R.4.2.2, или выделить группы тел, между которыми искать столкновения не требуется. Для этого предусмотрен набор типовых классов, наследников от `cdet_query`, настроенных на запрос подробностей столкновения:

- Структура `cdet_query_result` – самый простой и быстрый вариант, отвечающий на вопрос имеются ли столкновения в контрольной группе тел. Именно эта структура используется в качестве параметра по умолчанию для **CheckCollisions**.
- Структура `cdet_first_collided` – выдает первую найденную пару граней столкновения.
- Структура `cdet_collided_faces` – выдает список граней, участвующих в столкновениях, в виде упорядоченного множества пар `std::pair<cdet_app_item,const MbRefItem*>`, где первый элемент пары – твердотельная модель клиентского приложения, второй элемент пары – грань, принадлежащая модели. Также с помощью метода `cdet_collided_faces::Group` можно сгруппировать объекты контрольного набора тел так, что бы искать только пересечения между группами.

Метод `cdet_collided_faces::Group` объединяет в группу пару тел; для объединения в группу *n* тел необходимо вызвать `cdet_collided_faces::Group` *n-1* раз для первого и всех остальных тел группы.

В параграфе [4.4. Настройка запроса на поиск столкновений](#) описано, как с помощью наследования от `cdet_query` пользователь геометрического ядра может настроить собственные запросы, однако, в большинстве случаев необходимые запросы можно осуществить с помощью класса `cdet_collided_faces`. Для этого необходимо завести экземпляр `cdet_collided_faces` и применить его в качестве параметра **CheckCollisions**. Перед этим можно настроить фильтр поиска с помощью методов `cdet_collided_faces::Group` и `cdet_collided_faces::ExcludeGroup`. При необходимости фильтр поиска можно сбросить с помощью вызова `cdet_collided_faces::Reset` и перенастроить фильтр заново.

Метод

`cdet_collided_faces::Group(cdet_app_item fst, cdet_app_item snd)`

объявляет группу тел, элементы которой не проверяются на столкновение друг с другом.

Входными параметрами метода являются *fst* и *snd* – пара тел, которые уже состоят в группах или впервые объединяются в группу.

Данный метод объединяет в одну группу два тела или присоединяет одно тело к группе другого тела. Таким образом, для того, чтобы объединить *n* тел в группу необходимо вызвать `cdet_collided_faces::Group` *n-1* раз для первого и всех остальных тел группы.

Метод

`cdet_collided_faces::ExcludeGroup(cdet_app_item member)`

исключает все элементы группы из поиска на столкновения.

Входным параметром метода является *member* – любое тело, участник группы, элементы которой исключаются.

Данный метод отключает группу из любых проверок на столкновение. Его рекомендуется применять для временного отключения проверки тел на столкновения без полного удаления из набора с помощью `MbCollisionDetectionUtility::RemoveItem`, так как последующее добавление с помощью **AddItem** займет больше вычислительного времени, чем отменить исключение с помощью `cdet_collided_faces::Reset`. Для исключения набора тел из проверки на столкновения допускается многократно вызвать метод **ExcludeGroup** для всех элементов набора либо предварительно объединить все исключаемых тела в группу, а затем один раз вызвать **ExcludeGroup** для любого тела группы.

Метод

`cdet_collided_faces::Reset()`

отменяет результаты всех вызовов **Group** и **ExcludeGroup**. Метод применяется, если требуется перенастроить группы и исключения для однажды заданной структуры запросов `cdet_collided_faces`.

R.4.4. Настройка запроса на поиск столкновений

Если возможности настроек структуры `cdet_collided_faces` не достаточны, пользователь геометрического ядра C3D может реализовать собственный запрос. Для этого необходимо определить наследника класса `cdet_query` и реализовать для него специальную функцию обратного вызова (callback функцию) вида:

```
typedef cback_res (*cback_func)( cdet_query *, message, cback_data & ).
```

Ниже приведен пример реализации на языке программирования C++ структуры `cdet_query_result`, являющейся наследником структуры запроса `cdet_query`. Пример задает поиск до первого найденного пересечения и определяет, есть ли хотя бы одно пересечение в контрольном наборе тел. Пример можно найти в файле `cdet_data.h`

```
//-----  
struct cdet_query_result: public cdet_query  
{  
    cdet_result result;  
  
    cdet_query_result()  
        : cdet_query( QueryFunc )  
        , result( CDET_RESULT_NoIntersection )  
    {}  
  
private:  
    static cback_res QueryFunc( cdet_query * query, message code, cback_data & )  
    {  
        cdet_query_result * q = static_cast<cdet_query_result*>( query );  
        switch( code )  
        {  
            case CDET_QUERY_STARTED: // The collision query is started for all solids of the set  
            {  
                q->result = CDET_RESULT_NoIntersection;  
                return CBACK_VOID_RESULT;  
            }  
            case CDET_INTERSECTED: // First intersection is founded.  
            {  
                q->result = CDET_RESULT_Intersection;  
                return CBACK_SUFFICIENT;  
            }  
            case CDET_FINISHED: // A pair of solids is finished.  
                return (q->result == CDET_RESULT_Intersection) ? CBACK_BREAK:CBACK_VOID_RESULT;  
  
            default:  
                return CBACK_VOID;  
        }  
    }  
};
```

В основе запроса лежит функция обратного вызова

```
static cback_res QueryFunc( cdet_query * query, message code, cback_data & cData ),
```

которую пользователь может реализовать на свое усмотрение. Данная функция передается в алгоритм **CheckCollisions** через структуру `cdet_query` и будет вызываться при наступлении одного из следующих событий (enum `cdet_query::message`):

<code>CDET_QUERY_STARTED</code>	(Начат поиск столкновений в контрольной группе тел),
<code>CDET_STARTED</code>	(Начат поиск столкновений для пары объектов),
<code>CDET_FINISHED</code>	(Завершен поиск столкновений для пары объектов),
<code>CDET_INTERSECTED</code>	(Найдено пересечение граней от пары объектов),
<code>CDET_TOUCHED</code>	(Найдены соприкасающиеся грани пары объектов).

Входными параметрами функции **QueryFunc** являются:

- **query** – указатель на структуру запроса,
- **code** – целочисленное значение, кодирующее событие поиска,

- **cData** – структура передачи данных.

Функция возвращает одно из значений перечисления `sback_res`:

CBACK_SUFFICIENT (Реакция клиентского приложения на события `CDET_INTERSECTED` или `CDET_TOUCHED`; Прерывает поиск для данной пары тел, но продолжает поиск столкновений других пар из контрольного набора тел),

CBACK_SKIP (Пропустить поиск для данной пары тел по событию `CDET_STARTED`: Поиск для других пар из набора будет продолжен),

CBACK_BREAK (Прерывает дальнейший поиск пересечений для всех событий),

CBACK_VOID (Не оказывает влияния на запрос столкновений, продолжает поиск),

которое передает клиентское приложение ядру C3D для настройки алгоритма поиска.

Целочисленное значение `code` определяет событие поиска и может принимать одно из значений:

`CDET_QUERY_STARTED`,
`CDET_STARTED`,
`CDET_FINISHED`,
`CDET_INTERSECTED`,
`CDET_TOUCHED`,

Структура данных **cData** предназначена для передачи пары элементов типа `cdet_query::geom_element` и должна иметь вид:

```
struct cback_data
{
    geom_element first, second;
};
```

где элемент пары `cdet_query::geom_element` – это структура деталей поиска, содержащая следующие:

- `geom_element::appItem` – геометрический объект клиентского приложения,
- `geom_element::refItem` – геометрический объект типа [MbRefItem](#), как правило это грань [MbFace](#),
- `geom_element::wMatrix` – матрица перехода в общую, мировую СК из локальной СК тела.

Различная реализация метода обратного вызова **QueryFunc** позволяет выполнять следующие настройки процесса поиска:

- Собрать все пары граней, участвующих в столкновении тел;
- В любой момент исключать из рассмотрения определенные тела или пары тел;
- Различать случай пересечения и касания граней;
- Прерывать поиск всех пересечений при достижении определенного результата;
- Прерывать поиск при обнаружении пересечения или касания;
- Группировать тела, чтобы в пределах группы пересечения не искались.

R.4.5. Группировка тел из контрольного набора

При контроле столкновений часто требуется искать столкновения не для всех пар из контрольного набора тел, а только между группами тел. Например, можно повысить быстродействие, если не тратить время на тесты пересечения, когда известно, что тела внутри группы не пересекаются, либо по какой-то причине проверки внутри группы не требуется. Организовать такую группировку можно двумя способами.

Первый способ – передавать одно и то же значение `appItem` при добавлении тел одной группы в метод [AddItem](#). Этот способ удобен, когда тела группируются естественным образом по их принадлежности некоторому «монолитному» объекту на стороне пользовательского приложения.

Второй способ – настроить запрос `cdet_query`, позволяющий пропускать тест пересечения для пары тел, принадлежащих одной и той же группе. Пример этого можно увидеть в исходном коде класса `cdet_collided_faces`. Для этого нужно в методе обратного вызова вернуть `CBACK_SKIP` по событию `CDET_STARTED` для пары тел, принадлежащей одной и той же группе.

S.1. ДВУМЕРНЫЙ ГЕОМЕТРИЧЕСКИЙ РЕШАТЕЛЬ

Любой чертеж с геометрической точки зрения можно представить набором объектов плоскости – это точки, прямые, отрезки, дуги, эллипсы, сплайновые кривые, но не всякий произвольный набор плоской геометрии есть чертеж. В чертеже геометрические объекты всегда подчинены взаимосвязям, задающим положения одних объектов относительно других. Такие взаимосвязи предполагаются всегда, вне зависимости от того, чертим ли мы эскиз на бумаге с помощью карандаша или создаем его на компьютере. Если речь идет о компьютерном черчении, то такие взаимосвязи задаются с помощью *геометрических ограничений*, которые включают в себя *логические ограничения* такие, как совпадение, параллельность/перпендикулярность, касание, горизонтальность/вертикальность, симметричность, и *размерные ограничения*, задающие разного рода *угловые и линейные размеры*, а также размеры, задающие длину кривой или радиус. Эскиз или чертеж, имеющий хотя бы одно размерное ограничение, называется *параметрическим*.

Решатель двумерных геометрических ограничений позволяет вычислять положение геометрических объектов, удовлетворяющее всем заданным ограничениям и размерам эскиза. Процесс черчения, при котором положение объектов, подчиняется заданным ограничениям и размерам называется *параметрическим черчением*.

S.1.1. Назначение геометрического решателя GCE

В составе геометрического ядра C3D решатель двумерной геометрии имеет внутреннее техническое название – GCE (*сокр. Geometric Constraint Engine*). GCE является компонентом модуля C3D Solver, включающего в себя еще один компонент – решатель трехмерных сопряжений (GCM). Основное предназначение компонента GCE – это удовлетворение системы ограничений, заданных над геометрическими объектами плоского чертежа (эскиза). Предметом вычислений решателя GCE являются геометрические объекты на плоскости: точка, прямая, окружность, эллипс, сплайн или параметрическая кривая, на которые пользователь накладывает связи в виде ограничений из заданного набора типов, включающий логические ограничения: параллельность, перпендикулярность, касание, точка на кривой, симметрия и разного рода размерные ограничения, задающие расстояния, длины и углы. Решатель также может работать с кривыми, имеющими концы (*bounded curves*), как например, отрезки (*line segments*) и дуги окружности (*arcs*).

Вся функциональность геометрического решателя GCE доступна через заголовочные файлы `gce_types.h` и `gce_api.h` и состоит из набора простых структур данных и функций. Система типов решателя отражает основные понятия для формулировки задачи геометрических ограничений чертежа и манипулирования состоянием геометрических объектов. Проблемная область геометрического решателя включает такие понятия, как геометрический объект, геометрическое ограничение, размер, система ограничений, числовая переменная и т.д. Взаимодействие приложения с решателем ограничений осуществляется на основе простых структур данных C++, все они объявлены в файле `gce_types.h`.

S.1.2. Встраивание в приложение

Геометрический решатель GCE построен, как общецелевой компонент параметрического черчения. Это значит, что его можно встроить в любое приложение, имеющее дело с плоской (*planar*) геометрией, чертежами или эскизами. Предполагается, что такое приложение уже имеет свои структуры данных, представляющие геометрию эскиза. Перед программистом, встраивающим компонент GCE в свое приложение, фактически ставится задача передать под управление решателя свои геометрические объекты, подчиненные системе ограничений. Программный интерфейс решателя имеет свою собственную отвлеченную (*abstract*) систему типов данных, составляющую необходимый минимум того, чтобы сформулировать задачу удовлетворения ограничений. Поэтому, чтобы подключить геометрический решатель к управлению эскизом, необходимо реализовать специальную обертку, которая, выполняя роль моста между C3D Solver и приложением, будет передавать данные о геометрии и ограничениях в решатель, и наоборот, применять результаты вычислений решателя к геометрии эскиза. На рис. S.1.2.1. изображена примерная схема

взаимодействия решателя ограничений с эскизом. Обертка **GCE wrapper** обслуживает параметрическую модель эскиза и выполняет следующие функции:

- Загружает в решатель геометрическую модель эскиза, выраженную в типах данных модуля C3D Solver;
- Обрабатывает запросы редактора такие, как добавление/удаление данных системы ограничений, вычисление эскиза, драггинг геометрических объектов;
- Управляет состоянием геометрии эскиза, подчиненной ограничениям;
- Обновляет данные решателя по актуальному состоянию геометрии эскиза;
- Скрывает (адаптирует) API решателя с его математическими типами данных и предоставляет более удобное API с типами данных приложения;

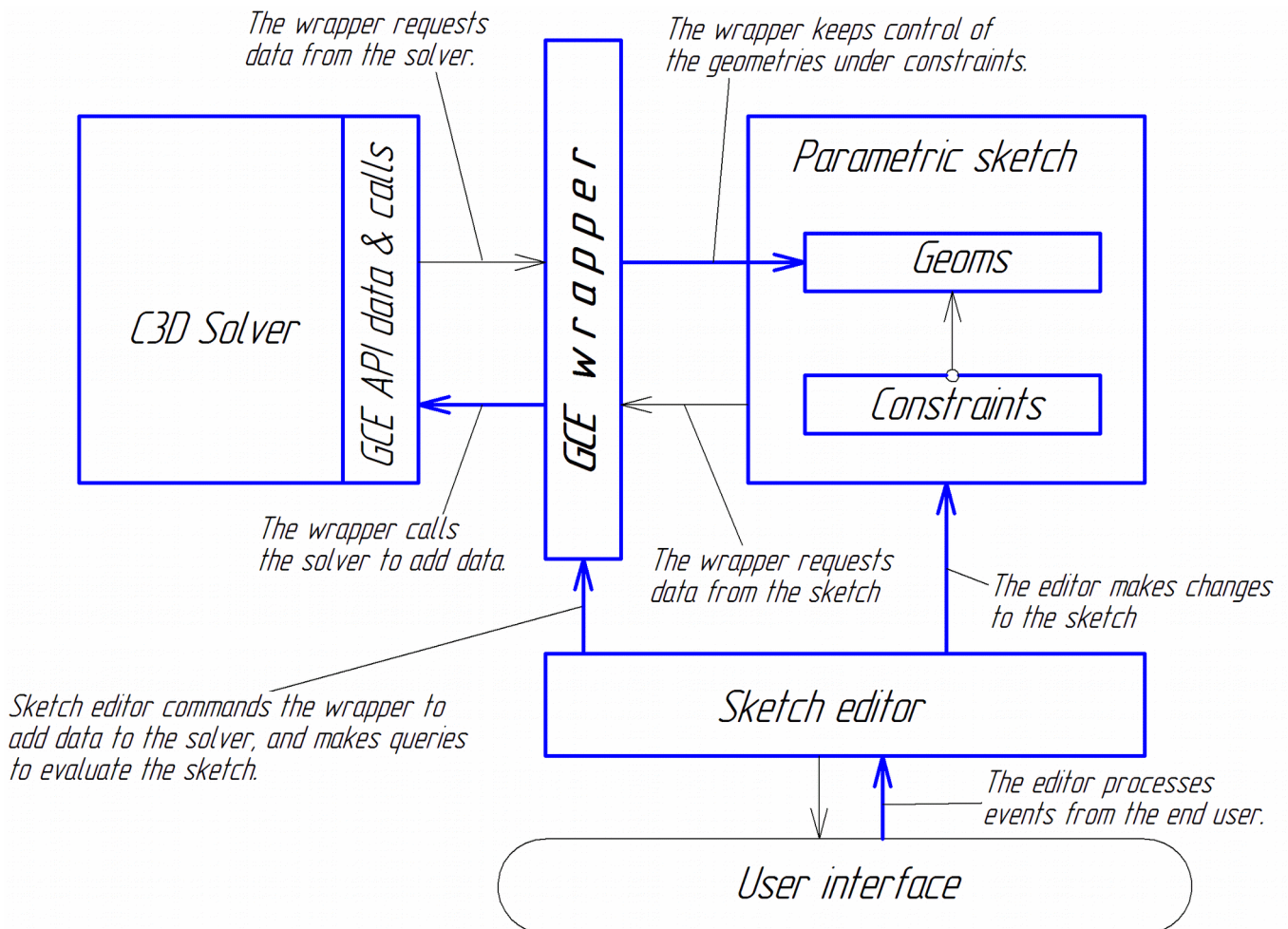


Рис. S.1.2.1.

Схема на рис. S.1.2..1. не является неперменным руководством по встраиванию решателя, а только показывает пример, демонстрирующий возможный способ интеграции в приложение с адаптацией математических структур данных GCE к типам данных приложения.

Также компонент GCE не позволяет сохранять данные пользователя в файл документа, поэтому разработчик приложения должен позаботиться о чтении/записи системы ограничений своего эскиза в документе приложения. Кроме того, следует принимать во внимание, что набор типов геометрического решателя не может перечислить всё многообразие геометрических типов приложения. Например, в решателе нет типа «прямоугольник», однако он может быть представлен, как четверка точек и прямых, связанных определенными ограничениями. Возможное программное решение предусматривает, что каждый чертежный объект будет иметь свой метод, формулирующий его описание в решателе.

S.1.3. Типы поддерживаемой геометрии

Геометрический решатель GCE поддерживает базовый набор типов геометрии, достаточный, чтобы на его основе строить разнообразные чертежные объекты эскиза. Базовый набор включает следующие типы геометрии:

- Точка (point) – задается двумя декартовыми координатами X, Y;
- Прямая (line) – задается точкой и вектором нормали;
- Окружность (circle) – задается точкой центра и радиусом;
- Эллипс (ellipse) – задается точкой центра, направляющим вектором и двумя значениями полуосей;
- Сплайн (spline) – задается структурой данных, определяющих NURBS;
- Параметрическая кривая (parametric curve) – задается объектом кривой общего типа MbCurve;

Также решатель позволяет создавать отдельные куски или дуги на основе бесконечных кривых. Для этого предусмотрены тип bounded curve и специально выделенный её частный случай – отрезок прямой (line segment):

- Ограниченная кривая (bounded curve) – задается одной базовой кривой и двумя концевыми точками;
- Отрезок прямой (line segment) – задается парой точек;

S.1.4. Типы геометрических ограничений и размеров

Любое геометрическое ограничение связывает между собой геометрические объекты, называемые *аргументами* ограничения. Аргументом ограничения может быть любой объект, относящийся к одному из типов геометрии, перечисленных в параграфе [S.1.3. Типы поддерживаемой геометрии](#). Например, окружность может быть аргументом ограничения **касание**. По числу геометрических аргументов ограничения различают, как унарные, бинарные и тернарные. Они связывают соответственно один, два или три объекта. Примером тернарного ограничения является **симметрия**, в которой участвуют три объекта: две точки и ось симметрии. В общем случае ограничение может иметь N аргументов. *Логические ограничения* предусматривают только зависимости между геометрическими объектами. *Размерные ограничения* устанавливают связь между геометрическими объектами и числовым параметром, означающим какой-то линейный или угловой размер, поэтому у размерных ограничений последний аргумент всегда числовой, называемый также *скаляром*. Типы ограничений, которые поддерживает геометрический решатель GCE приведены в таблице S.1.4.1..

Таблица S.1.4.1. Поддерживаемые типы ограничений

Логические ограничения	Размерные ограничения	Специальные ограничения
фиксация	длина отрезка	управляющий параметр
горизонтальность	радиус	фиксация производной сплайна
вертикальность	расстояние	точка на кривой с позицией по параметру кривой
длина отрезка	направленное расстояние	линейное уравнение вида $a_1*v_1 + a_2*v_2 + \dots + a_n*v_n + c = 0$
совпадение	угол	
точка на кривой		
выравнивание точек		
коллинеарность		
равенство длин		
равенство радиусов		
параллельность		
перпендикулярность		
касание		

средняя точка		
зеркальная симметрия		
биссектриса		

Более подробно каждое из этих ограничений будет рассмотрено ниже. Логические ограничения рассматриваются в главе [S.3. ДВУМЕРНЫЕ ЛОГИЧЕСКИЕ ОГРАНИЧЕНИЯ](#). Размерные ограничения и вопросы параметризации эскиза рассматриваются в главе [S.2. ДВУМЕРНЫЕ РАЗМЕРЫ](#).

S.1.5. Базовые типы данных API решателя GCE

Взаимодействие приложения с решателем ограничений осуществляется на основе простых структур данных C++, все они объявлены в файле `gce_types.h`. Среди них центральное место занимают дескрипторные типы данных, которые служат для идентификации любых объектов, находящихся под контролем решателя, см. табл. S.1.5.1.

Таблица S.1.5.1. Дескрипторные типы данных

Тип данных решателя	Реализация	Интерпретация
<code>GCE_system</code>	<code>void *</code>	дескриптор системы ограничений
<code>geom_item</code>	<code>size_t</code>	дескриптор геометрического объекта
<code>constraint_item</code>	<code>size_t</code>	дескриптор ограничения
<code>var_item</code>	<code>size_t</code>	дескриптор числовой переменной

Типы данных, которые перечисляют конечные наборы значений, приведены в табл. S.1.5.2.

Таблица S.1.5.2. Перечислительные типы данных

Тип данных решателя	Интерпретация
geom_type	тип геометрического объекта
constraint_type	тип ограничения
<code>point_type</code>	тип запрашиваемой контрольной точки
<code>coord_name</code>	имя координаты геометрического объекта
GCE_result	диагностический код выполнения функции
<code>GCE_bisec_variant</code>	выбор между решениями биссектрисы

Структуры данных, перечисленные ниже в табл. S.1.5.3., служат для передачи набора параметров объекта. Например, с помощью структуры [GCE_ellipse](#) в решатель передаются данные эллипса: координаты центра, размеры полуосей и направляющий вектор.

Таблица S.1.5.3. Структуры данных

Тип данных решателя	Интерпретация
<code>GCE_vec2d</code>	координаты двумерного вектора
<code>GCE_point</code>	координаты точки на плоскости
<code>GCE_point_dof</code>	запись степени свободы точки

GCE_line	координаты прямой
GCE_circle	координаты окружности
GCE_ellipse	координаты эллипса
GCE_spline	координаты и характеристики сплайна
GCE_dim_pars	параметры размерного ограничения
GCE_adim_pars	параметры углового размерного ограничения
GCE_ldim_pars	параметры линейного размерного ограничения
GCE_dragging_point	контрольная точка драггинга

S.1.6. Система геометрических ограничений

Система геометрических ограничений это набор геометрических объектов и ограничений, связывающих эти объекты. Типы поддерживаемых геометрических объектов и ограничений приведены соответственно в параграфах [S.1.3. Типы поддерживаемой геометрии](#), [S.1.4. Типы геометрических ограничений и размеров](#). Предполагается, что параметрический эскиз, который создает приложение, с его объектами и ограничениями, имеет свою систему ограничений. С программной точки зрения это означает, что каждый параметрический эскиз ассоциируется с системой ограничений через дескриптор типа [GCE_system](#), а каждый его геометрический объект и ограничение представлены своими уникальными дескрипторами типов [geom_item](#) и [constraint_item](#).

Перед тем как начать работать с геометрическими ограничениями эскиза, необходимо объявить для него систему ограничений с помощью вызова функции

```
GCE_system GCE\_CreateSystem()
```

Функция вернет систему ограничений в виде дескриптора [GCE_system](#), который представляет собой указатель на экземпляр внутренней структуры данных геометрического решателя. Все дальнейшие манипуляции с системой ограничений будут осуществляться через данный дескриптор. Например, если мы хотим объявить точку в эскизе, необходимо для его системы ограничений вызвать функцию

```
geom_item GCE\_AddPoint( GCE_system gSys, GCE_point pVal )
```

Результатом функции является дескриптор геометрической точки, принадлежащей системе ограничений [gSys](#). Параметр [pVal](#) задает начальные координаты <X,Y> точки.

При завершении работы с эскизом, обязательно следует вызвать функцию удаления системы ограничений

```
void GCE\_RemoveSystem( GCE_system gSys ),
```

которая полностью освободит память, занимаемую системой ограничений со всеми данными объектов и ограничений. После вызова [GCE_RemoveSystem](#) дескриптор системы ограничений становится недействительным, т.е. дальнейшее использование такого дескриптора может привести к аварийному прекращению работы приложения.

Функция

```
void GCE\_ClearSystem( GCE_system gSys )
```

делает систему ограничений пустой, освобождая память только от объектов и ограничений, но оставляет систему ограничений действительной для дальнейшей работы.

S.1.7. Представление геометрических объектов

Решатель геометрических ограничений работает с определенной формой представления геометрических объектов, проиллюстрированной на рис. S.1.7.1. Все объекты выражаются через координаты точек, вектора и числа (скаляры).

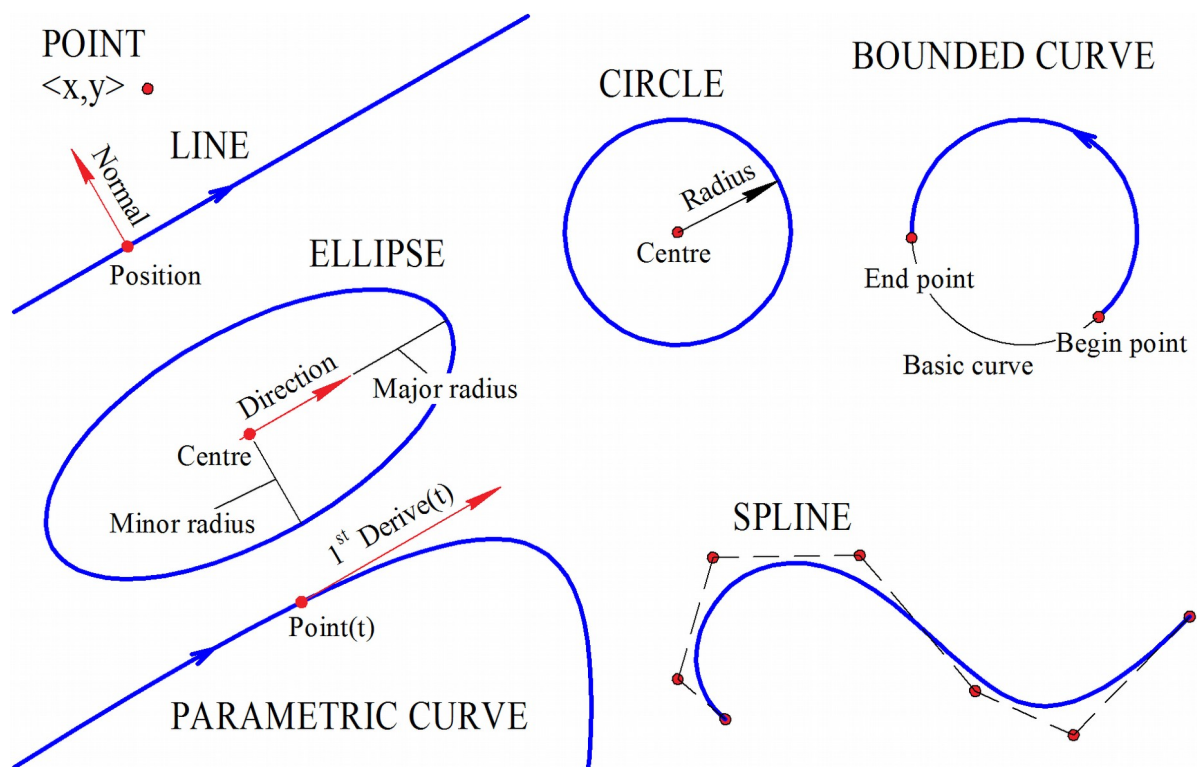


Рис S.1.7.1.

Приложение может иметь свое представление геометрических объектов, отличное от решателя, однако передача данных о состоянии объектов в решатель и получение обратно результатов вычислений основываются на том, что каждый тип геометрии имеет свое представление:

- **Точка** представлена парой декартовых координат $\langle X, Y \rangle$.
- **Прямая** задается точкой позиции и вектором нормали. Предполагается, что у прямой есть направляющий вектор, который равен вектору нормали повернутому по часовой стрелке на 90 градусов. Другими словами, направляющему вектору с координатами $\langle X, Y \rangle$ будет соответствовать нормаль с координатами $\langle Y, -X \rangle$.
- **Окружность** задается точкой центра и радиусом. В настоящий момент радиус может быть только ненулевым положительным числом.
- **Эллипс** задается точкой центра, радиусами по «главной» (major) и по «минорной» (minor) полуосям и направляющим вектором главной полуоси. Также эллипс параметризуется величиной с периодом от 0 до 2π , пробегающей по эллипсу против часовой стрелки, начиная от точки на «главной» полуоси.
- **Параметрическая кривая** передается в решатель в виде класса `MbCurve`. Такая кривая рассматривается как неподвижная, а вычисления связанные с этой кривой, строятся на виртуальных функциях: `MbCurve::PointOn`, `MbCurve::FirstDer` и `MbCurve::SecondDer`, выдающие соответственно точку на кривой по параметру, первую и вторую производную в этой точке. Подробнее о параметрических кривых см. параграф [S.5.2. Параметрические кривые общего вида](#)

Замечание: Пока для создания параметрической кривой в системе ограничений потребуется создать экземпляр класса `MbCurve`, однако в будущем будет реализован альтернативный вариант, основанный на простых функциях, реализуемых самим пользователем. Также пользователь ядра C3D имеет возможность реализовать своего наследника для `MbCurve`.

- **Сплайн** основан на NURBS-представлении, в основе которого лежит список контрольных точек. Более подробно работа со сплайном будет рассмотрена в главе [S.5. ДВУМЕРНЫЕ СПЛАЙНЫ И ПАРАМЕТРИЧЕСКИЕ КРИВЫЕ](#)
- **Ограниченная кривая** представляет собой участок кривой, ограниченный концевыми точками. Задается тройкой: базовая кривая, начальная точка участка, конечная точка участка.

S.1.8. Степень свободы

Каждому типу геометрии свойственна степень свободы, равная минимальному числу координат, необходимых для определения состояния геометрического объекта. Например, степень свободы двумерной точки равна 2. Окружность имеет степень свободы 3, поскольку она полностью определяется тройкой значений $\langle X, Y, R \rangle$ – координатами центра и радиусом. Согласно параграфу [S.1.7. Представление геометрических объектов](#), прямая представляется точкой позиции и вектором нормали, однако это удобное, но избыточное представление; минимально-достаточным представлением прямой могла бы быть пара значений смещения и угла наклона, поэтому прямая имеет степень свободы 2. Также эллипс имеет степень свободы равную 5. Степень свободы сплайна равна сумме степеней свобод его контрольных точек. Параметрическая кривая полностью определена на стороне приложения, т.е. её степень свободы равна 0. В таблице S.1.8.1. приведены степени свободы всех типов, поддерживаемых решателем.

Таблица S.1.8.1. Степень свободы геометрических объектов

Тип геометрии	Степень свободы
Точка	2
Прямая	2
Окружность	3
Эллипс	5
Сплайн	2 · Количество контрольных точек
Параметрическая кривая	0
Ограниченная кривая	2 + Степень свободы базовой кривой

Каждый геометрический объект, участвующий в системе, добавляет определенное количество своих степеней свобод к общей степени свободы эскиза. С другой стороны, каждое добавляемое ограничение отнимает определенную степень свободы. Для того, чтобы определить состояние всех геометрических объектов эскиза, требуется определенное количество ограничений, отнимающее все степени свободы объектов.

Большинство ограничений отнимает одну степень свободы, это такие ограничения, как параллельность, перпендикулярность, горизонтальность/вертикальность, равенство радиусов, равенство длин, точка на прямой, касание и большая часть размерных ограничений. Другие ограничения снимают две степени свободы: средняя точка, коллинеарность, симметрия, биссектриса.

Можно сказать, что задача параметрического черчения состоит в том, чтобы полностью определить геометрические объекты эскиза. Количество геометрических ограничений, которые необходимы для полного определения эскиза, соответствуют сумме степеней свобод всех объектов, входящих в эскиз.

S.1.9. Добавление и удаление геометрических объектов

Основным предметом вычислений геометрического решателя являются геометрические объекты, поэтому формирование системы ограничений начинается с того, что приложение объявляет в системе ограничений геометрические объекты, которым предстоит стать аргументами ограничений.

Каждый геометрический объект, объявленный в системе ограничений, получает свой уникальный идентификатор – дескриптор типа `geom_item`, а его геометрический тип (`geom_type`) остается неизменным в течении жизни объекта. Ниже приведены вызовы API C3D Solver, добавляющие геометрические объекты в систему.

Точка добавляется с помощью метода

`geom_item GCE_AddPoint(GCE_system gSys, GCE_point pVal).`

Результатом функции является дескриптор геометрической точки, принадлежащей системе ограничений `gSys`. Параметр `pVal` задает начальные значения координат точки $\langle X, Y \rangle$.

Прямая добавляется с помощью метода

```
geom_item GCE_AddLine( GCE_system gSys, GCE_line lVal ).
```

Результатом функции является дескриптор прямой на плоскости, принадлежащей системе ограничений **gSys**. Параметр **lVal** задает начальные значения позиции **lVal.p** и нормали **lVal.norm** прямой.

Окружность добавляется с помощью метода

```
geom_item GCE_AddCircle( GCE_system gSys, GCE_circle cVal ).
```

Результатом функции является дескриптор окружности, принадлежащей системе ограничений **gSys**. Параметр **cVal** задает начальные значения центра **cVal centre** и радиуса **cVal.radius**.

Эллипс добавляется с помощью метода

```
geom_item GCE_AddEllipse( GCE_system gSys, GCE_ellipse eVal ).
```

Результатом функции является дескриптор эллипса, принадлежащего системе ограничений **gSys**. Параметр **eVal** определяет начальные значения эллипса:

<eVal centre, eVal.direct, eVal.majorR, eVal.minorR>, означающие соответственно центр, направляющий вектор и радиусы по главной и минорной осям.

Сплайн добавляется с помощью метода

```
geom_item GCE_AddSpline( GCE_system gSys, GCE_spline spl ).
```

Результатом функции является дескриптор сплайна, принадлежащего системе ограничений **gSys**. Структура данных **spl** определяет сплайн в начальном состоянии. Подробности, связанные со сплайновой кривой и способами его определения, рассмотрены в параграфе [S.5.1. Сплайновые кривые](#).

Параметрическая кривая общего вида добавляется с помощью метода

```
geom_item GCE_AddParametricCurve( GCE_system gSys, const MbCurve & crv ).
```

Результатом функции является дескриптор параметрической кривой, добавленной в систему ограничений **gSys**. Объект **crv** полностью определяет абстрактное математическое описание кривой в параметрической форме. Подробнее класс двумерной кривой общего вида обсуждается в параграфе [O.3.1. Двумерная кривая MbCurve](#). Надо заметить, что тип данных **MbCurve** входит в иерархию классов геометрического ядра C3D, а значит его время жизни регулируется счетчиком ссылок. Другими словами система ограничений гарантирует действительность экземпляра **MbCurve**, пока параметрическая кривая не будет удалена из системы. Подробнее о работе с параметрической кривой рассказывается в параграфе [S.5.2. Параметрические кривые общего вида](#).

Ограниченная кривая добавляется с помощью метода

```
geom_item GCE_AddBoundedCurve( GCE_system gSys, geom_item curve, geom_item p[2] ).
```

Результатом функции является дескриптор ограниченной кривой, принадлежащей системе ограничений **gSys**. Созданная кривая основывается на кривой **curve** и будет представлять её участок, ограниченный концевыми точками **p[0]** и **p[1]**. Данный метод фактически связывает три объекта – базовую кривую и две точки в единый объект и автоматически обеспечивает два ограничения «точка на кривой» для концевых точек. Фрагмент кода ниже показывает пример создания дуги **окружности**.

```
GCE_system gSys = GCE_CreateSystem();
GCE_circle circPars; // Circle values
GCE_point endP1, endP2; // Coordinates of the end points.
// ...
// It is required to insert the code that assigns start values of the circle and its endpoints.
// ...
geom_item p[2] = { GCE_AddPoint(gSys, endP1), GCE_AddPoint(gSys, endP2) };
geom_item circItem = GCE_AddCircle( gSys, circPars ); // Create a circle. It's a base curve
of the arc.
geom_item arc = GCE_AddBoundedCurve( gSys, circItem, p ); // Create the circular arc.
// ...
GCE_RemoveSystem( gSys );
```

Аналогично с помощью вызова **GCE_AddBoundedCurve** можно создавать дуги эллипса, участки на параметрической кривой, отрезки прямой и т.д. Ограниченная кривая, созданная на основе прямой, называется **отрезком**, а все ограничения действующие для прямой также применимы для отрезка.

В любой момент жизни геометрический объект, добавленный в систему ограничений одним из вышеперечисленных методов, может быть удален вызовом метода

```
bool GCE_RemoveGeom( GCE_system gSys, geom_item g ).
```

Данный метод удаляет геометрический объект из системы и делает его дескриптор **g** не действительным. Если геометрический объект на момент вызова является аргументом одного из ограничений, то он фактически остается в системе, но его неизбежное удаление откладывается до тех пор, пока не будет удалено последнее из ограничений, в которых он участвует.

S.1.10. Фиксация и заморозка геометрических объектов

Любой геометрический объект, созданный в системе ограничений, изначально свободен, т.е. имеет полную степень свободы, свойственную его типу. При вычислениях C3D Solver может менять состояние геометрических объектов, когда это требуется для удовлетворения ограничений. Иногда требуется обездвижить часть геометрии так, чтобы решатель оставлял положение геометрического объекта без изменений. Для этого в API GCE предусмотрены вызовы, которые могут **заморозить** или **зафиксировать** геометрический объект. Только приложение может поменять состояние замороженных или фиксированных геометрических объектов с помощью вызовов метода

```
GCE_SetPointXY и GCE_SetCoordValue.
```

Заморозку геометрических объектов удобно применять, когда требуется обеспечить неподвижность какой-то части чертежа. Например, в системе САПР часть геометрии чертежа может быть получена проецированием из трехмерной модели, и поэтому навсегда связана с ней односторонней ассоциативной связью: «объемная модель → плоское отображение». Заморозить геометрический объект можно с помощью метода

```
bool GCE_FreezeGeom( GCE_system gSys, geom_item g ).
```

Метод вернет значение **true**, если геометрический объект **g** стал фактически замороженным. Надо заметить, что заморозка не считается ограничением, поэтому метод **GCE_IsConstrainedGeom** не принимает во внимание заморожен объект или нет. Степень свободы замороженного объекта равна 0.

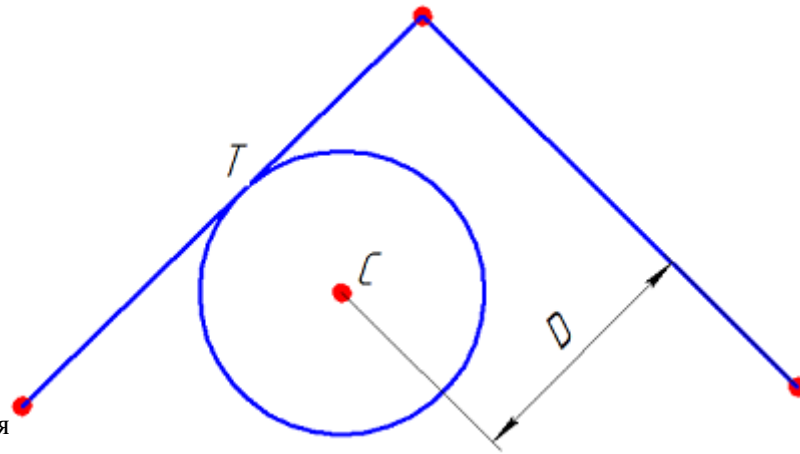
Альтернативный способ сделать геометрический объект неподвижным для решателя – это задать ограничение фиксация объекта с помощью метода

```
constraint_item GCE_FixGeom( GCE_system gSys, geom_item g ).
```

Функция зафиксирует объект **g**, принадлежащий системе ограничений **gSys**, и вернет дескриптор вновь добавленного ограничения «фиксация объекта». В отличие от **GCE_FreezeGeom** данный метод создает ограничение, которое в любой момент можно удалить методом **GCE_RemoveConstraint** и тем самым освободить объект. Ограниченная фиксация рассмотрена также в параграфе [S.3.6. Унарные ограничения: горизонтальность/вертикальность и варианты фиксации](#).

S.1.11. Контрольные точки геометрических объектов

Среди геометрических типов, поддерживаемых решателем (см. параграф [S.1.3. Типы поддерживаемой геометрии](#)) особенное место занимает **точка**. Во-первых, точка самый элементарный геометрический объект. Во-вторых, через точки, называемые *контрольными*, выражаются все остальные типы геометрии. Например, окружность выражается структурой $\langle C, R \rangle$, где **C** – точка центра, **R** – скаляр, числовое значение радиуса. В-третьих, контрольные точки могут самостоятельно участвовать в ограничениях, в качестве их аргументов. Например, можно задавать ограничения, как для самой окружности, так и отдельно для её точки, скажем, расстояние от центра окружности до прямой, см. рис. S.1.11.1.



C – контрольная точка центра
 T – касание окружности и прямой
 D – расстояние от центра до прямой

Рис. S.1.11.1.

Чтобы работать с контрольной точкой, как с самостоятельной точкой, необходимо запросить её дескриптор у геометрического объекта с помощью функции:

geom_item **GCE_PointOf**(GCE_system gSys, geom_item g, point_type pnt).

Результатом функции является дескриптор контрольной точки геометрического объекта **g**, принадлежащего системе ограничений **gSys**. Параметр **pnt** определяет какая контрольная точка объекта запрашивается и принимает одно из следующих значений:

- GCE_FIRST_END – начальная точка кривой (первый конец);
- GCE_SECOND_END – конечная точка кривой (второй конец);
- GCE_CENTRE – центр окружности, эллипса, а также их дуги (bounded curve);
- GCE_Q1 – квадрантная точка эллипса (3 часа);
- GCE_Q2 – квадрантная точка эллипса (12 часов);
- GCE_Q3 – квадрантная точка эллипса (6 часов);
- GCE_Q4 – квадрантная точка эллипса (9 часов).

Запросить контрольные точки сплайновой кривой позволяет метод

geom_item **GCE_SplinePoint**(GCE_system gSys, geom_item spl, size_t pntIdx).

Функция вернет дескриптор контрольной точки сплайна **spl**, взятой по номеру **pntIdx**, принимающий значения от 0 до N-1, где N – число контрольных точек сплайна. На рис. S.1.11.2. изображен сплайн, имеющий N = 7 контрольных точек (красные кружки), с нумерацией от 0.

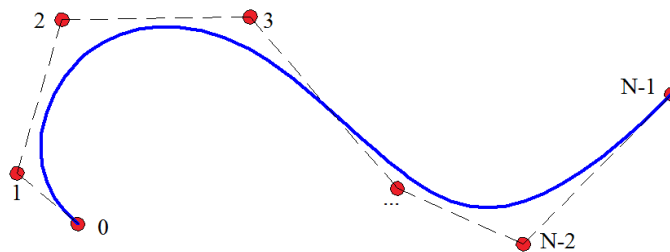


Рис. S.1.11.2.

S.1.12. Скалярная переменная

Основным предметом вычисления модуля GCE являются геометрические объекты двумерной плоскости, подчиненные ограничениям и размерам. Эти типы данных по своей природе имеют векторное представление. Параметрическое черчение предполагает связь геометрии с управляющими либо вариационными числовыми параметрами. Поэтому дело не обходится без еще одного типа –

числовой переменной или **скаляра** со значением, заданным, как правило, в единицах длины или угла.

В параметрической системе ограничений могут участвовать переменные такого рода:

- Числовая переменная, ассоциированная с размером – **размерная переменная**. В решенном состоянии размерная переменная равна измеренному значению размера. Размерная переменная может служить как управляющим, независимым параметром эскиза, так и вариационным или измерительным. Подробнее об использовании переменных для создания управляющих и вариационных размеров обсуждается в параграфе [S.2.2. Управляющие и вариационные размеры](#).
- Вспомогательный параметр геометрического ограничения, например, параметр, указывающий локализацию точек касания для кривых сложной формы. Подробности в параграфе [S.3.10. Касание](#).
- Переменная, не ассоциированная с геометрическим ограничением и размером, может участвовать в скалярных линейных или нелинейных уравнениях, устанавливающих законы связи одних величин с другими.

В API модуля GCE переменную представляет тип данных `var_item`. А создать переменную можно вызовом метода

```
var_item GCE_AddVariable( GCE_system gSys, double val ).
```

Функция вернет дескриптор скалярной переменной с начальным значением `val`. В дальнейшем переменную по данному дескриптору можно применять для определения размеров, ограничений и уравнений.

Любая переменная может быть **зависимой** или **независимой**. Зависимая переменная подчинена измерительным размерам или ограничениям, а её значение вычисляется решателем. Независимая переменная не может быть изменена решателем, а её значение задается приложением. Чтобы переменная стала независимой, достаточно вызвать метод API

```
constraint_item GCE_FixVariable( GCE_system gSys, var_item var ),
```

который создаст ограничение «управляющий параметр» и вернет дескриптор ограничения, фиксирующий её значение. Независимая переменная доступна для варьирования с помощью вызова [GCE_ChangeDrivingDimension](#). Ограничение «управляющий параметр» вместе с другими типами управляющих размеров позволяют создать параметрический эскиз, геометрия которого может меняться по определенному пользователем закону варьированием независимых параметров. Про создание управляющих параметров и размеров см. параграф [S.2.2. Управляющие и вариационные размеры](#).

S.1.13. Линейное уравнение

Линейное уравнение выражается формулой $a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + \dots + a_n \cdot v_n + c = 0$, где $a_1, a_2, a_3, \dots, a_n, c$ – постоянные коэффициенты, $v_1, v_2, v_3, \dots, v_n$ – скалярные переменные уравнения. Линейное уравнение создается вызовом метода

```
constraint_item GCE_AddLinearEquation( GCE_system gSys, const double * a,  
const var_item * v, size_t n, double c ),
```

который вернет дескриптор линейного уравнения, зарегистрированного в системе `gSys`.

Исходные данные вызова:

- `gSys` – система ограничений;
- `a` – массив постоянных коэффициентов при переменных;
- `v` – массив переменных линейного уравнения;
- `n` – количество переменных линейного уравнения;
- `c` – постоянный коэффициент «с» без переменной;

S.1.14. Журналирование вызовов API

C3D Solver позволяет для конкретной системы ограничений [GCE_system](#) записать всю историю вызовов API, включая исходные параметры и возвращаемые значения вызванных функций. Результатом записи будет файл журнала, который в дальнейшем может быть использован разработчиками геометрического ядра C3D для отладки и тестирования функций C3D Solver.

Устройство журнала

Файл журнала имеет текстовый формат, основанный на простом синтаксисе S-выражений, позволяющий записать любые вложенные и списочные структуры данных. Любая функция API, вызванная для конкретной системы ограничений, фиксируется в журнале в следующем виде:

(GCE_Func (arg list) (returned value)),

где *GCE_Func* – имя вызванной GCE-функции, *(arg list)* – список аргументов функции, *(returned value)* – возвращаемое значение.

Например, так выглядит запись вызова, создающего окружность с центром в начале координат и радиусом 2.5:

(GCE_AddCircle ((0.0 0.0) 2.5) #1)

Окружность получила символический идентификатор #1, который является возвращаемым значением функции *GCE_AddCircle*.

Журнал фактически является записью сеанса работы с конкретной системой ограничений на языке, который может быть интерпретирован специальными средствами, которыми располагает разработчик C3D Solver. Также журнал может быть отредактирован вручную или упрощен, если для достижения определенной ситуации не требуются все вызовы.

Включение режима журналирования

Чтобы включить режим записи журнала системы ограничений, необходимо вызвать функцию

`bool GCE_SetJournal(GCE_system gSys, const char * fName),`

которая для системы ограничений **gSys** назначит файл журнала **fName**. В случае успешной инициализации режима функция вернет **true**.

Входными параметрами метода являются:

- **gSys** – пустая система ограничений, созданная методом [GCE_CreateSystem](#),
- **fName** – строка, задающая полный путь к файлу. Например: `”..\\Journals\\sample.jrn”`

После включения журналирования будет осуществляться запись информации о всех обращениях к API C3D Solver, адресованных системе ограничений **gSys**.

Внимание: файл журнала будет готов только после завершения сеанса работы с системой ограничений, а именно сразу после вызова [GCE_RemoveSystem](#).

Пример кода

Пример кода демонстрирующий запись сеанса работы системы ограничений в журнал приведен ниже.

```
GCE_system gSys = GCE\_CreateSystem();
// Switch on journalling for the constraint system.
GCE\_SetJournal( gSys, "C:\\Logs\\gce_sample.jrn" );
// Make some calls of C3D Solver ...
GCE_point p1Val, p2Val; // Coordinates of two points = (1,1) and (2,2)
p1Val.x = p1Val.y = 1.0;
p2Val.x = p2Val.y = 2.0;
geom_item pnt[2] = { GCE\_AddPoint(gSys, p1Val), GCE\_AddPoint(gSys, p2Val) };
GCE\_AddCoincidence( gSys, pnt );
// ...
// ...
GCE\_Evaluate( gSys );
// Finalize the constraint system
GCE\_RemoveSystem( gSys );
// The journal file is ready!
```

Замечание

Журналирование нужно только для отладочных целей, поэтому рекомендуется включать её только в режиме отладки. Не рекомендуется оставлять журналирование включенным для продуктовой (release) или рабочей версии приложения, т.к. это может приводить к напрасному расходованию ресурсов времени и памяти.

Отладка и тестирование

Журналирование дает возможность разработчику геометрического ядра отлаживать функций C3D Solver путем воспроизведения журнала вызовов, записанного при работе в приложении. Данный способ использует тот факт, что внутреннее состояние системы ограничений полностью определяется последовательностью вызовов API, поэтому достаточно повторить ту же последовательность вызовов с теми же данными, чтобы полностью воспроизвести проблему, возникшую при эксплуатации решателя приложением.

Обширная коллекция журналов образует базу тестовых примеров, которая является неотъемлемым инструментом для развития и контроля качества модуля C3D Solver. Каждая ревизия C3D Solver проверяется на тестовой базе журналов, что гарантирует сохранение результатов всех ранее сделанных исправлений и усовершенствований.

Таким образом, чтобы получить консультацию или решить проблемную ситуацию, связанную со встраиванием C3D Solver в приложение, достаточно временно включить режим журналирования, выполнить интересующий сценарий действий и переслать полученный файл с необходимыми комментариями или вопросами в техническую поддержку C3D Labs. После исправления ошибки соответствующий журнал добавляется в базу тестовых примеров.

S.2. ДВУМЕРНЫЕ РАЗМЕРЫ

Размеры представляют собой ограничения, связывающие геометрические объекты с числовым параметром (значением размера). Каждый размер измеряет какую-то числовую характеристику геометрии, например, расстояние между точками, угол между прямыми, радиус окружности. Логические ограничения отличаются от размеров тем, что не имеют ни одного числового аргумента. Каждый размер всегда имеет, кроме геометрических аргументов, один числовой (скаляр).

О том, как с помощью параметров и размеров можно управлять геометрией эскиза, обсуждается в параграфе [S.2.2. Управляющие и вариационные размеры](#).

S.2.1. Вспомогательные точки линейного размера

Линейные размеры (distance dimension) для некоторых типов объектов, а также диаметральные размеры не могут быть однозначно заданы без дополнительного указания точек приложения размера. Например, расстояние от окружности до отрезка прямой может измеряться двумя способами: от «дальней» точки приложения к окружности до отрезка либо от «ближней» точки до отрезка, см. рис. S.2.1.1. Понятия «дальний» и «ближний» взяты в кавычки, т.к. ситуация может поменяться при варьировании размера и тогда «дальняя точка» станет ближе к отрезку, чем «ближняя».

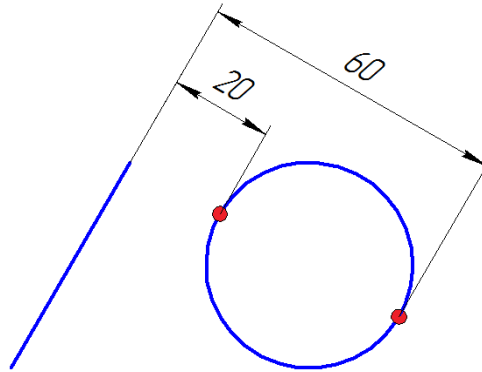


Рис S.2.1.1.

Для однозначного определения размера потребуется с помощью вызова `GCE_AddPoint` создать вспомогательную точку, из которой начинается выносная линия размера со стороны окружности. Дескриптор этой точки будет использован в качестве вспомогательного при создании линейного размера. В примере, приведенном на рис. S.2.1.1., требуется только одна точка – на стороне окружности. В других случаях, таких как расстояние окружность-окружность, потребуется две вспомогательные точки, поскольку для пары окружностей возможны четырех варианта определения размера. На рис S.2.1.2. размеры на 30, 50, 60 и 90 показывают альтернативные варианты измерения расстояний между двумя окружностями и соответственно комбинации выбора пары вспомогательных точек на окружностях.

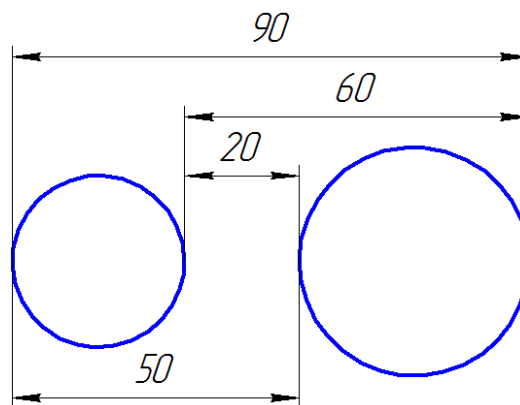


Рис S.2.1.2.

S.2.2. Управляющие и вариационные размеры

Размеры отличаются от логических ограничений (параллельность, перпендикулярность и т.д.) тем, что они связаны с числовой величиной (скаляром) – *значением размера*. Значение размера можно задать двумя способами:

- **Управляющий размер.** Значение размера задается независимым числовым параметром, который дается пользователем и может меняться только извне через API решателя (вызов **GCE_ChangeDrivingDimension**). В этом случае размер не зависит от геометрии, а наоборот, геометрия подчиняется размеру также как геометрическому ограничению. Размеры с таким способом задания требуются для того, чтобы управлять геометрией с помощью размерных параметров, и называются соответственно **управляющими**.
- **Вариационный размер.** Значение размера задается числовой переменной, которая является таким же предметом вычислений решателя, как и геометрия. Фактически текущее значение будет ассоциировано с переменной, которая специально создается с помощью вызова **GCE_AddVariable**, и может быть повторно применена в других размерах или уравнениях. Значение переменной размера может меняться под влиянием геометрии. Такой способ задания применяется в тех случаях, когда требуется не установить размер, а измерить его. Размеры, связанные с варьируемой переменной, называются **вариационными**.

Иногда требуется создать *вариационный* размер, действующий, как *управляющий*, особенно когда необходимо управлять одновременно двумя и более размерами, завязанными на один числовой параметр. В этом случае можно создать один или несколько вариационных размеров с одной переменной, которую можно сделать *управляющим параметром* с помощью вызова:

```
constraint_item GCE_FixVariable( GCE_system gSys, var_item var),
```

где **var** – дескриптор переменной размера.

Функция вернет ограничение «*управляющий параметр*», фиксирующее переменную **var** с её текущим значением. После вызова **GCE_FixVariable** переменная фактически становится независимым параметром системы ограничений и может меняться только извне, с помощью вызова **GCE_ChangeDrivingDimension**, примененным к дескриптору полученного ограничения.

Варьирование управляющими размерами и параметрами осуществляется с помощью метода

```
GCE_result GCE_ChangeDrivingDimension( GCE_system gSys, constraint_item dItem,  
double dVal).
```

Входными параметрами метода являются:

- **gSys** – система ограничений параметрического эскиза;
- **dItem** – дескриптор управляющего размера или управляющего параметра;
- **dVal** – требуемое значение управляющего размера или управляющего параметра.

Функция вернет код GCE_RESULT_OK, если операция выполнена успешно. Если функция вернет код ошибки (отличный от GCE_RESULT_OK), это может быть связано с одной из следующих причин:

- Недействительные дескрипторы системы или ограничения **gSys** и **dItem**;
- Присланное ограничение не является управляющим размером или управляющим параметром, например, если ограничение – вариационный размер;

Следует учитывать, что данная функция не выполняет вычислений, а только подготавливает изменение управляющего размера или параметра. Чтобы изменение вступило в силу, необходимо вызвать вычислительную функцию **GCE_Evaluate**. Если требуется изменить одновременно два и более управляющих размеров, следует сначала сделать серию подготовительных вызовов **GCE_ChangeDrivingDimension** на каждый управляющий параметр или размер, а затем одним вызовом **GCE_Evaluate** вычислить новое положение геометрии, подчиненное новому состоянию управляющих размеров или параметров.

S.2.3. Нулевые и знакопеременные размеры

Все типы линейных размеров (distance dimension) имеют непрерывную область, включающую нуль, отрицательные и положительные значения. На рис. S.2.3.1. изображен эскиз с размером, позиционирующим геометрию, как в положительной, так и в отрицательной области значений. Слева

эскиз имеет отрицательное значения размера, справа изображен тот же эскиз, которому поменяли знак размера с отрицательного на положительный.

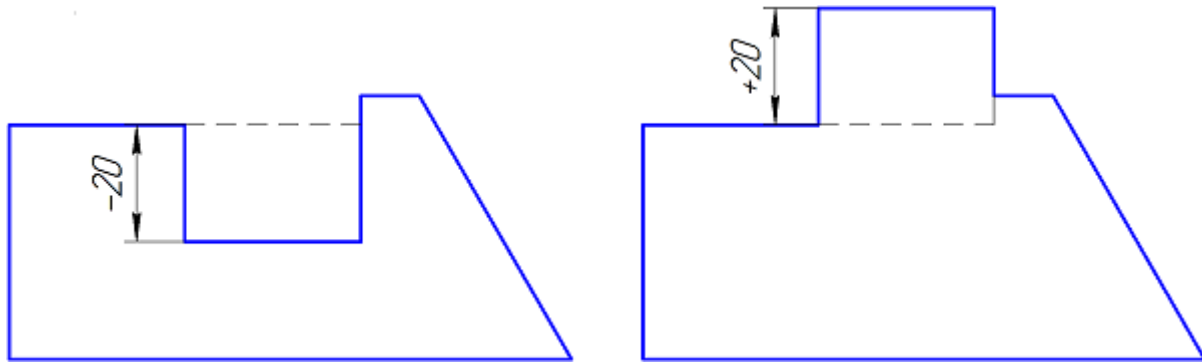


Рис.S.2.3.1.

Нулевые значения допустимо принимать всем видам линейных и угловых размеров, штриховой линией изображена геометрия в «нулевом» положении, когда значение размера равно нулю.

Знакопеременность свойственна прежде всего для размеров, заданных для ориентированного объекта, такого, как прямая, имеющая вектор нормали. Прямая делит пространство на две полуплоскости, поэтому, к примеру, линейный размер между прямой и точкой будет иметь разные знаки в зависимости от того, на какой стороне от прямой лежит точка. Точка, лежащая на прямой, будет соответствовать нулевому размеру.

Заметим, что расстояние от точки до точки по природе является неотрицательной величиной; несмотря на это, линейный размер, заданный для двух точек, может быть отрицательным или нулевым. Это вполне имеет практический смысл, когда мы имеем дело с параметрическим эскизом. Например, знакопеременный размер, проиллюстрированный на левом и правом эскизе из рис. S.2.3.1., можно задать, как ограничение «расстояние от точки до прямой», так и «расстояние от точки до точки». В обоих случаях его поведение будет одинаковым.

S.2.4. Линейный размер (расстояние)

Линейный размер задается для пары геометрических объектов. В таблице S.2.4.1 приведены пары типов геометрии, которые поддерживает C3D Solver применительно к линейному размеру.

Таблица S.2.4.1. Пары геометрических объектов, поддерживаемых для линейного размера

	Точка	Прямая	Окружность	Эллипс	Сплайн	Параметрическая кривая
Точка	√	√	√			
Прямая	√	√	√			
Окружность	√	√	√			
Эллипс						
Сплайн						
Параметрическая кривая						

Любой линейный размер, вне зависимости от типов кривых, для которой он применен, измеряет расстояние между точками, лежащими на кривых. Обычно это точки с минимальным расстоянием между объектами. Однако можно выбрать другие пары точек для измерения размера. На рис S.2.1.2. из параграфа S.2.1. Вспомогательные точки линейного размера показан пример четырех вариантов линейного размера для одной и той же пары окружностей.

Линейный размер для пары геометрических объектов задается методом

```
constraint_item GCE_AddDistance( GCE_system gSys, geom_item g[2],
```

const GCE_ldim_pars & dPars);

Возвращаемое значение функции – дескриптор линейного размера, заданного для объектов **g[0]** и **g[1]**, принадлежащих системе ограничений **gSys**.

Структура параметров **dPars** определяет следующие настройки линейного размера:

- **dPars.dPars** – задает значение размера, который может быть, как независимым параметром, так и переменной;
- **dPars.hp[2]** – пара вспомогательных точек, задающих вариант измерения, когда это требуется (подробнее см. параграф [S.2.1. Вспомогательные точки линейного размера](#)).

Параметр **dPars.dPars** фактически определяет источник значения размера. Если **dPars.dPars.var** = **GCE_NULL_V**, то значение размера определяется независимым числовым параметром **dPars.dPars.dimValue**, что делает размер управляющим. Если в **dPars.dPars.var** задан дескриптор переменной, то размер является вариационным. Подробнее об управляющих и вариационных размерах обсуждалось в параграфе [S.2.2. Управляющие и вариационные размеры](#).

Пара дескрипторов вспомогательных точек **dPars.hp[2]** не является обязательной, если вариант измерения размера однозначен, как например, расстояние от точки до прямой, либо оставляется на выбор решателя, который подберет вариант измерения наиболее близкий к текущим расположению объектов и значению размера. В этом случае по умолчанию **dPars.hp[2]** равен паре «нулевых» дескрипторов {**GCE_NULL**, **GCE_NULL**}. Если одна или обе вспомогательные точки **dPars.hp[0]** и **dPars.hp[1]** заданы, то координаты этих точек будут указывать линейному размеру вариант измерения. Подробнее о выборе варианта измерения обсуждается в параграфе [S.2.1. Вспомогательные точки линейного размера](#).

S.2.5. Направленный линейный размер

Обычный линейный размер измеряет расстояние между точками, лежащими на геометрических объектах. Направленный линейный размер также задается для пары точек, однако расстояние измеряется между проекциями точек вдоль прямой с конкретным направлением. В основном данный тип размера используется для создания вертикальных и горизонтальных размеров, которые задают расстояние между точками в проекции на оси «X» или «Y». На рис. S.2.5.1. показаны примеры вертикального и горизонтального размеров для пары точек. В общем случае направление размера может быть произвольным.

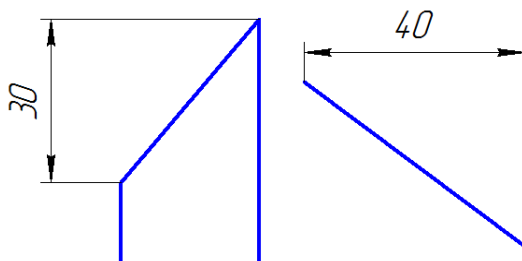


Рис. S.2.5.1.

Также направленный размер предусматривает определенное поведение при геометрической трансформации эскиза методом [GCE_Transform](#). Направленный размер повернется, если матрица трансформации будет содержать ротационный компонент.

Направленный размер создается с помощью метода

```
constraint_item GCE_AddDirectedDistance( GCE_system gSys, geom_item p[2],  
const GCE_ldim_pars & dPars ).
```

Возвращаемое значение функции – дескриптор наклонного размера.

Входными параметрами метода являются:

- **gSys** – система ограничений параметрического эскиза;
- **p[2]** – дескрипторы двух точек;
- **dPars** – структура данных, в которой кроме обычных настроек, свойственных линейным размерам (см. аналогичные настройки для метода [GCE_AddDistance](#)), дополнительно задается угловое направление размера **dPars.dirAngle**. Например, если требуется задать

горизонтальный размер, то $\mathbf{dPars.dirAngle} = 0$; если требуется задать вертикальный размер, то $\mathbf{dPars.dirAngle} = \pi/2$. В общем случае значение $\mathbf{dPars.dirAngle}$ берется от 0 до 2π .

Замечание. В текущей версии C3D Solver направленный размер можно задать только для точек.

S.2.6. Расстояние от точки до отрезка

Расстояние от точки до отрезка – это линейный размер, вовлекающий тройку точек, из которых две точки задают прямолинейный отрезок, от которого измеряется расстояние до третьей точки. Данный тип размера создается с помощью метода

```
constraint_item GCE_AddDistancePLs( GCE_system gSys, geom_item p[3],
                                     const GCE_dim_pars & dPars ).
```

Особенность данного ограничения в том, что он требует ненулевую длину отрезка. Это означает, что линейный размер перестанет работать, если две точки, задающие отрезок, совпадут. Размер может быть применен, когда требуется задавать линейный размер только по точкам, без наличия линейного объекта, однако если этого не требуется, то рекомендуется использовать функцию **GCE_AddDistance**, которой будет достаточно для создания практически всех вариантов линейных размеров.

S.2.7. Угловые размеры

Угловой размер задается для двух геометрических объектов, имеющих направляющие вектора. Значение углового размера задается в радианах с областью значений от 0 до 2π и определяется углом, измеряемым против часовой стрелки, между направляющими векторами первого и второго объектов. Угловой размер с таким способом измерения показан на рис. S.2.7.1. на примере двух пересекающихся прямых с данными направлениями. Справа на рис. S.2.7.2. показана пара прямых, имеющих тот же угол между векторами, однако размер охватывает обратный сектор угла. Значение такого размера определяется по формуле $D = 2\pi - \alpha$, где α – измерительное значение угла между векторами, а D – значение размера. Данный размер также называется сопряженным (conjugate), по другому его можно получить, просто поменяв местами аргументы углового размера.

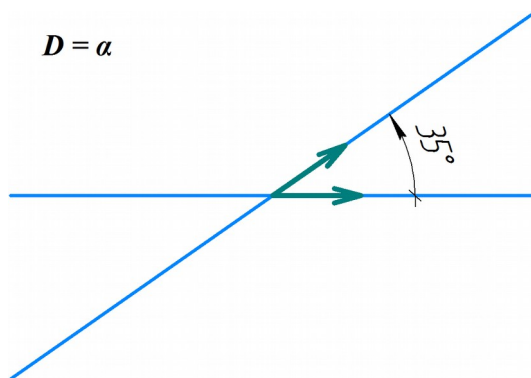


Рис. S.2.7.1.

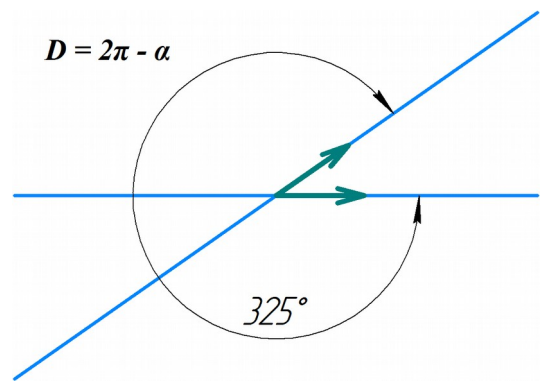


Рис. S.2.7.2.

Всего для одной и той же пары направлений, включая два рассмотренных выше, возможны четыре варианта задания углового размера. На рис. S.2.7.3. изображен размер, значение которого определяется по формуле смежного (adjacent) угла $D = \pi - \alpha$. На рис. S.2.7.4. справа показан альтернативный вариант смежного угла, который определяется суммой $D = \pi + \alpha$.

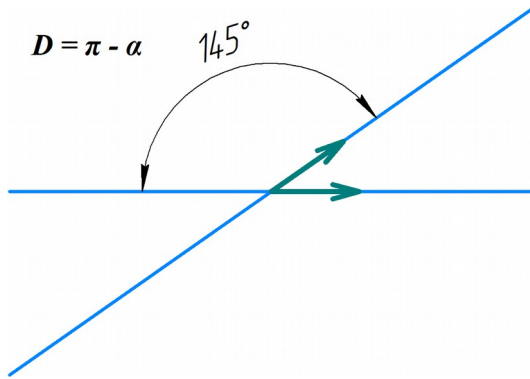


Рис. S.2.7.3.

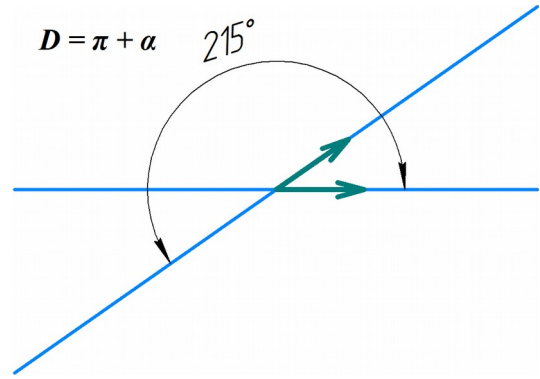


Рис. S.2.7.4.

Еще одна полезная закономерность, связанная с четырьмя способами задания углового размера, состоит в том, что размер со значением $D = 2\pi - \alpha$ получается из размера $D = \alpha$, если просто поменять местами первый и второй объекты размерного ограничения. Аналогично, переменной местами аргументов, получается 4-й вариант ($D = \pi + \alpha$) из 3-го варианта ($D = \pi - \alpha$).

Угловой размер для пары направленных объектов, таких, как отрезок, прямая или эллипс, задается методом

```
constraint_item GCE_AddAngle( GCE_system gSys,
                             geom_item I1, geom_item I2,
                             const GCE_adim_pars & dPars ).
```

Функция создаст в системе ограничений **gSys** новый угловой размер, заданный для геометрических объектов **I1** и **I2**. Значение размера и способ его задания определяется структурой **dPars**.

Возвращаемое значение – дескриптор нового углового размера.

Структура **dPars** определяет значение углового размера, заданного через структуру **dPars.dPars** (**GCE_dim_par**), в которой значение размера хранится в переменной **dPars.dPars.var** либо, в случае **dPars.dPars.var = GCE_NULL_V**, значение размера задано независимым числовым параметром **dPars.dPars.dimValue**, что делает размер управляющим. Подробнее об управляющих и варьируемых разделах рассмотрено в .

Если параметр **dPars.adjacent = false**, то значение размера будет определяться формулой $D = f \cdot \alpha$. Множитель **f** в данной формуле задается параметром **dPars.factor** и позволяет задавать угловой размер в других единицах. Например, если **dPars.factor = 180/π**, то приложение получает возможность задавать угловой размер в градусах. Что бы задать угловой размер со значением $D = f \cdot (2\pi - \alpha)$, необходимо поменять местами аргументы **I1** и **I2**.

Для определения угловых размеров со смежным вариантом угла, показанных на рис. S.2.7.3. и рис. S.2.7.4., необходимо задать флаг **dPars.adjacent = true**, в этом случае значение размера будет определять по формулам $D = f \cdot (\pi - \alpha)$ или $D = f \cdot (\pi + \alpha)$. Вторая формула получается, если поменять местами аргументы **I1** и **I2**. Множитель **f** также задается параметром **dPars.factor**.

S.2.8. Угловой размер по трем или четырем точкам

C3D Solver также предоставляет еще один способ определения углового размера, для которого направляющие вектора определяются парами точек. Этот же способ позволяет задавать угловой размер по трем точкам, которые определяют угол измерительного треугольника.

Угловой размер по тройке или четверке точек создается с помощью метода

```
constraint_item GCE_AddAngle4P( GCE_system gSys, geom_item fPair[2],
                                geom_item sPair[2], const GCE_adim_pars & dPars ).
```

Пары точек, заданные дескрипторами **fPair** и **sPair**, определяют измерительные вектора углового размера. Вектор, построенный от точки **fPair[0]** до точки **fPair[1]**, определяет первую сторону угла,

вектор от **sPair[0]** до **sPair[1]**, определяет вторую сторону угла. На рис.2.8.1. показаны измерительные углы размера для трех точек (слева) и четырех точек (справа). Чтобы задать размер по трем точкам, необходимо точку вершины угла задать дважды, в первой и во второй паре, например, **fPair[0]=sPair[0]**.

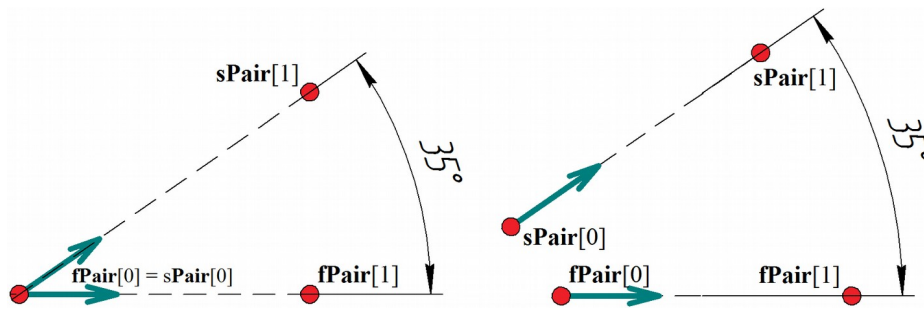


Рис.2.8.1.

Остальные параметры функции **GCE_AddAngle4P** задаются аналогично **GCE_AddAngle**.

S.2.9. Радиальные и диаметральные размеры

Данные типы размеров действуют для окружностей или дуг и задают соответственно радиус окружности или диаметр, равный удвоенному значению радиуса. Радиальные и диаметральные размеры, как и линейные так и угловые, могут быть ассоциированы с независимым числовым параметром или скалярной переменной, т.е. быть *управляющими* или *вариационными*, что настраивается структурой **GCE_dim_pars**.

Радиальный размер создается методом:

```
constraint_item GCE_AddRadiusDimension( GCE_system gSys,
                                          geom_item cir,
                                          GCE_dim_pars dPar ).
```

Диаметральный размер создается методом:

```
constraint_item GCE_AddDiameter( GCE_system gSys,
                                   geom_item cir,
                                   GCE_dim_pars dPar ).
```

Оба метода имеют одинаковые исходные данные:

- **gSys** – система геометрических ограничений, где регистрируется новый размер;
- **cir** – дескриптор окружности или дуги окружности;
- **dPars** – структура, которая определяет значение размера, заданного в единицах длины.

Значение размера хранится в переменной **dPars.var** либо, в случае **dPars.var = GCE_NULL_V**, значение размера задано независимым числовым параметром **dPars.dimValue**, что делает размер управляющим. Подробнее управляющие и варьируемые размеры рассмотрены в параграфе [S.2.2. Управляющие и вариационные размеры](#).

S.3. ДВУМЕРНЫЕ ЛОГИЧЕСКИЕ ОГРАНИЧЕНИЯ

Логические ограничения, в отличие от размеров, не имеют числовых параметров и задаются только для геометрических объектов. C3D Solver позволяет задавать для геометрических объектов такие логические ограничения, как горизонтальность/вертикальность, совпадение, параллельность/перпендикулярность, касание, равенство радиусов и т. д.

S.3.1. Совпадение точки и другого объекта

В случае, когда два геометрических объекта имеют один и тот же тип, ограничение совпадение подразумевает равенство двух геометрических объектов. Однако, в настоящей версии C3D Solver данное ограничение действительно только, когда один из аргументов является точкой.

Если один из аргументов совпадения точка, а другой – кривая, то ограничение определит точку, лежащую на кривой. Ограничение совпадение задается вызовом метода

```
constraint_item GCE_AddCoincidence( GCE_system gSys, geom_item g[2] ).
```

Функция добавит в систему **gSys** ограничение совпадения двух геометрических объектов **g[0]** и **g[1]** и вернет его дескриптор.

S.3.2. Выравнивание точек

Данный тип ограничения чаще всего используется для выравнивания пары точек по горизонтали или по вертикали, в общем случае, данное ограничение будет размещать точки на воображаемой прямой с заданным угловым направлением.

Выравнивание точек задается вызовом метода

```
constraint_item GCE_AddAlignPoints(GCE_system gSys, geom_item p[2], double ang ).
```

Функция вернет дескриптор нового ограничения, зарегистрированного в системе **gSys**.

Параметры пары **p[0]** и **p[1]** задают дескрипторы выравниваемых точек, а параметр **ang** задается в радианах и определяет направление, вдоль которого будут выравниваться точки. Например, угловое направление **ang=0** будет соответствовать выравниванию точек по горизонтали, значение **ang= $\pi/2$** будет соответствовать выравниванию по вертикали.

S.3.3. Параллельность/перпендикулярность

Ограничения параллельность и перпендикулярность задаются для пары прямых или отрезков с помощью методов:

```
constraint_item GCE_AddParallel( GCE_system gSys, geom_item g[2] ),  
constraint_item GCE_AddPerpendicular( GCE_system gSys, geom_item g[2] ),
```

которые принимают следующие исходные данные:

- **gSys** – система геометрических ограничений,
- **g[2]** – пара прямолинейных кривых.

Возвращаемый результат – дескриптор геометрического ограничения, зарегистрированного в системе **gSys**.

S.3.4. Коллинеарность

Коллинеарность подразумевает, что геометрические объекты, участвующие в данном ограничении, лежат на одной прямой. Коллинеарность можно задать как для двух геометрических объектов, так и для трех точек. Коллинеарность для пары геометрических объектов создается вызовом метода

```
constraint_item GCE_AddColinear( GCE_system gSys, geom_item g[2] ),
```

который требует, чтобы один из аргументов ограничения был прямолинейным, а другой являлся либо прямой, либо точкой. Геометрические аргументы задаются парой **g[2]**. Функция вернет дескриптор нового ограничения, зарегистрированного в системе **gSys**.

Коллинеарность для трех точек создается вызовом метода

`constraint_item GCE_AddColinear3Points(GCE_system gSys, geom_item p[3])`.

Через параметр **p[3]** передаются дескрипторы трех точек, которые требуется разместить на одной прямой. Функция вернет дескриптор нового ограничения, зарегистрированного в системе **gSys**.

S.3.5. Равенство длин и радиусов

Обе функции, которые создают ограничения равенство длин и равенство радиусов, принимают одинаковые типы данных: система ограничений и пара геометрических объектов. Геометрическое условие, задающее равенство длин двух ограниченных кривых (bounded curve), создается методом

`constraint_item GCE_AddEqualLength(GCE_system gSys, geom_item g1, geom_item g2)`.

В настоящей версии C3D Solver данный метод действует только для пары отрезков. Аргументы ограничения задаются дескрипторами отрезков **g1** и **g2**. Функция вернет дескриптор нового ограничения, зарегистрированного в системе **gSys**. Равенство радиусов для двух окружностей или дуг задается вызовом метода

`constraint_item GCE_AddEqualRadius(GCE_system gSys, geom_item c1, geom_item c2)`,

принимающим соответственно систему ограничений **gSys**, дескриптор первой окружности или дуги **c1** и дескриптор второй окружности **c2**. Функция вернет дескриптор нового ограничения, зарегистрированного в системе **gSys**.

S.3.6. Унарные ограничения: горизонтальность/вертикальность и варианты фиксации

Такие ограничения, как горизонтальность, вертикальность, фиксация объекта, фиксация длины и фиксация направления действуют для одного геометрического объекта, а значит являются *унарными*. Для создания одного из этих ограничения необходимо вызвать функцию

`constraint_item GCE_AddUnaryConstraint(GCE_system gSys, constraint_type cType, geom_item geom)`.

Функция добавит в систему **gSys** унарное ограничение для геометрического объекта **geom**. Параметр **cType** задает перечислительный тип унарного ограничения и может принимать одно из значений, приведенных в таблице S.3.6.1.

Таблица S.3.6.1. Типы унарных ограничений

Тип ограничения	Описание
GCE_FIX_GEOM	Фиксация геометрического объекта делает объект неподвижным. Решатель не может поменять состояние данного объекта при вычислениях. Фиксированный геометрический объект может быть изменен только приложением с помощью вызовов GCE_SetPointXY и GCE_SetCoordValue.
GCE_HORIZONTAL GCE_VERTICAL	Горизонтальность и вертикальность применяются для типов прямая и отрезок прямой. Эти ограничения выравнивают линейный объект с системой координат эскиза: соответственно горизонтальность по оси OX, вертикальность по оси OY. Заметим, что данные ограничения допускают переворот направляющего вектора прямой на 180 градусов.
GCE_ANGLE_OX	Данное ограничение фиксирует угловое положение прямой или отрезка. Ограничение типа GCE_ANGLE_OX подробнее рассмотрено в параграфе S.3.9. Ограничение угловое положение .

GCE_LENGTH	Данное ограничение фиксирует длину отрезка прямой. Подробнее см. параграф S.3.7. Фиксация длины .
GCE_RADIUS_DIM	Данное ограничение подробно рассмотрено в параграфе S.3.8. Фиксация радиуса . Применительно к вызову GCE_UnaryConstraint данный тип ограничения действует только для окружности. Для применения к эллипсу следует вызывать GCE_FixRadius.

S.3.7. Фиксация длины

Предполагается, что данное ограничение фиксирует длину кривой, имеющей начало и конец (spline, bounded curve), однако, в настоящей версии C3D Solver применяется только для отрезка прямой. Фиксация длины является размерным ограничением, поэтому допускает изменение значения длины методом GCE_ChangeDrivingDimension.

Фиксация длины осуществляется вызовом метода

```
constraint_item GCE_FixLength( GCE_system gSys, geom_item ls ).
```

S.3.8. Фиксация радиуса

Ограничение фиксация радиуса применяется для окружности или эллипса. C3D Solver не может менять радиус окружности или эллипса, если для объекта задано ограничение фиксация радиуса. Приложение может работать с данным ограничением, как с управляющим размером, допускающим варьирование радиуса методом [GCE_ChangeDrivingDimension](#). Применительно к окружности данное ограничение является аналогом управляющего радиального размера, рассмотренного в параграфе [S.2.9. Радиальные и диаметральные размеры](#). Фиксация радиуса осуществляется вызовом метода

```
constraint_item GCE_FixRadius( GCE_system gSys, geom_item g, coord_name cName = GCE_RADIUS ).
```

Исходные данные вызова:

- **gSys** – система геометрических ограничений;
- **g** – дескриптор окружности или эллипса;
- **cName** – тип радиуса, принимающий следующие значения: GCE_RADIUS – значение по умолчанию для окружности; GCE_MAJOR_RADIUS или GCE_MINOR_RADIUS, означающие фиксацию радиуса на главной или минорной оси эллипса.

Возвращаемый результат – дескриптор ограничения, фиксирующего радиус окружности или эллипса.

S.3.9. Ограничение угловое положение

Ограничение угловое положение применяется для типов прямая и отрезок прямой. Фактически данный тип ограничения является размерным и создает управляющий размер, фиксирующий угловое положение линейного объекта в его начальном состоянии относительно оси OX общей системы координат эскиза. Угловое положение измеряется в диапазоне от 0 до 2π. Чтобы создать фиксацию направления, необходимо вызвать [GCE_AddUnaryConstraint](#), задав тип ограничения GCE_ANGLE_OX. В отличие от GCE_HORIZONTAL и GCE_VERTICAL, данный тип ограничения не допускает переворот направляющего вектора прямой на 180 градусов. Поскольку данное ограничение размерное, угловое положение прямой можно менять с помощью вызова GCE_ChangeDrivingDimension.

S.3.10. Касание

Касание – это геометрическое ограничение, позиционирующее пару кривых таким образом, что они будут касаться в одной точке. В таблице S.3.10.1. приведены поддерживаемые пары кривых касания.

Таблица S.3.10.1. Пары кривых, поддерживаемых ограничением касания

	Прямая	Окружность	Эллипс	Сплайн	Параметрическая кривая
Прямая		√	√	√	√
Окружность	√	√	√	√	√
Эллипс	√	√		√	
Сплайн	√	√	√	√	
Параметрическая кривая	√	√			

Выбор варианта касания. На рис. S.3.10.1. изображены два примера касания для пар окружность-окружность и окружность-прямая. Штриховой линией изображены окружности с альтернативным вариантом касания. Рисунок наглядно показывает, что любое касание может иметь два решения. Например, окружности могут касаться, оставаясь снаружи друг друга, либо одна внутри другой; в другом примере, на рис. S.3.10.1. справа показано, что при касании прямой и окружности возможны два варианта их взаимного размещения, «слева» и «справа» от прямой.

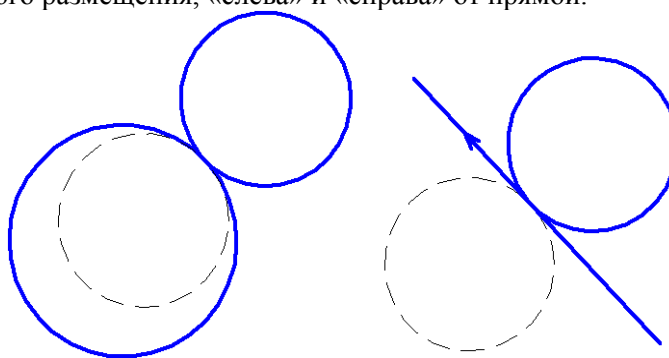


Рис.S.3.10.1.

Геометрический решатель GCE осуществляет выбор одного из двух вариантов касания, руководствуясь принципом «близости к желаемому решению». Фактически сами объекты касания указывают на выбор одного из двух вариантов взаимного размещения. В момент создания ограничения решатель «запоминает» текущее взаимное размещение объектов и сохраняет его при дальнейших изменениях геометрии эскиза (см. параграф [S.3.15. Взаимное размещение объектов](#)).

Вспомогательные параметры касания. Рассмотренный выше подход к выбору варианта касания на практике достаточен, если мы имеем дело только с аналитическими кривыми (прямая, окружность, эллипс). Для кривых с произвольной формой, таких как сплайн или параметрическая кривая, кроме выбора «стороны» взаимного размещения, необходимо указать локализацию точки касания вдоль кривых.

На рис. S.3.10.2. показан пример, где уже сделан выбор взаимного размещения прямой и сплайна (сонаправленность кривых в точке касания), однако этот выбор оставляет другие варианты локализации точек соприкосновения. Переменные t_1 и t_2 , взятые на параметрической области сплайна, указывают две альтернативные точки касания сплайна и отрезка, один конец которого закреплен.

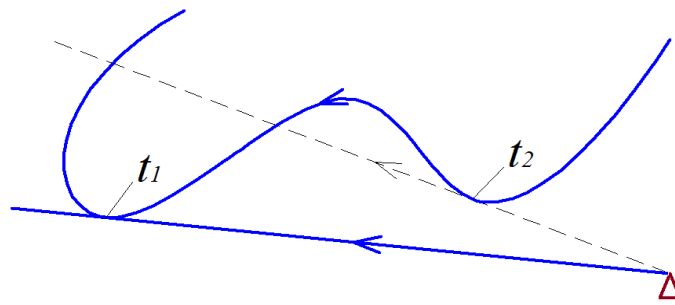


Рис.S.3.10.2.

Для локализации точек касания на сплайне или параметрической кривой заводят вспомогательные переменные, указывающие в параметрической области кривой предполагаемую точку касания. В конечном приложении пользователя это обычно реализуется так, что пользователь выбирает «указателем» объект касания, а координаты указателя рассматриваются, как приближительная точка предполагаемого касания. Зная координаты указателя, можно оценить приближительное значение параметра касания (параметр ближайшей точки на кривой).

Создание ограничения касания. Ограничение касания задается вызовом метода

constraint_item **GCE_AddTangent**(GCE_system gSys, geom_item g[2], var_item tPar[2]).

Входными параметрами метода являются:

- **gSys** – система ограничений параметрического эскиза;
- **g[2]** – дескрипторы двух кривых;
- **tPar[2]** – дескрипторы переменных, вспомогательные параметры касания для первой и второй кривой.

Функция возвращает дескриптор нового ограничения касания, созданного для данной пары кривых. Значения вспомогательных параметров **tPar[0]**, **tPar[1]** принимаются во внимание только, если соответственно первая или вторая кривая является сплайном или параметрической кривой. Для аналитических кривых (прямая, окружность, эллипс) следует передать пустой дескриптор **GCE_NULL_V**. Например, в **tPar** можно передать массив {**GCE_NULL_V**,**GCE_NULL_V**}, если обе кривые аналитические. Если касание задается для сплайна или параметрической кривой, то для ограничения заводится вспомогательная переменная с помощью вызова **GCE_AddVariable**. Начальное значение переменной будет указывать на предполагаемую точку касания в параметрической области кривой. Надо заметить, что любая переменная, как и геометрические объекты, является предметом вычислений, поэтому её значение будет обновляться при всех вычислительных запросах решателя (**GCE_Evaluate**, см. параграф [S.4.1. Вычисление системы ограничений](#)), т.е. фактически решатель будет отслеживать актуальное значение параметра касания. Если для сплайна или параметрической кривой **tPar[idx] = GCE_NULL_V**, то локализация точки касания будет выполнена автоматически, по принципу близости к начальному положению объектов.

S.3.11. Множественные и концевые касания

Геометрический решатель GCE поддерживает множественные касания, т.е. имеется возможность задать два и более ограничений касания для одной и той же пары кривых, в которых участвуют сплайны или параметрические кривые. На рис. S.3.11.1 изображены горизонтальная прямая и сплайн, для которых заданы два ограничения касания, имеющих различные точки соприкосновения.

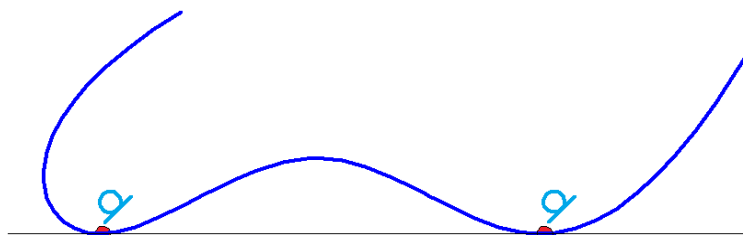


Рис.S.3.11.1.

Заметим, что в случае, когда пара касания состоит только из аналитических кривых (прямая, окружность, эллипс), возможно задать только единственное ограничение касания для одной и той же пары.

Концевое касание с непрерывностью G1 – характерный случай касания с участием сплайна, когда точка соприкосновения совпадает с одним из концов сплайна. В параметрическом черчении концевые касания позволяют создать гладкую стыковку двух кривых, с непрерывностью типа **G1**. На

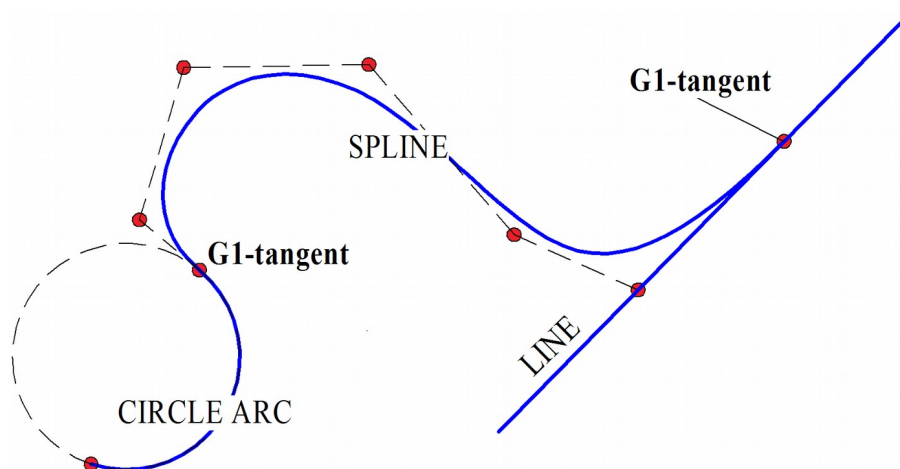


рис S.3.11.2. приведены случаи гладкой стыковки сплайна с другими кривыми.

Рис S.3.11.2.

Метод **GCE_AddTangent** воспринимает касание на конце незамкнутого сплайна, как особый случай, и автоматически добавляет к касанию условие принадлежности конца сплайна к другой кривой касания. Таким образом, если в общем случае, точка касания имеет свободу перемещения вдоль сплайна, то в случае концевого касания, точка касания прикрепляется к одному из концов сплайна.

Концевое касание можно создать одним из следующих способов:

- Передать в метод **GCE_AddTangent** вспомогательный параметр сплайна **tPar[splineIdx]**, равный параметру начала или конца сплайна.
- Вызвать метод **GCE_AddTangent** без вспомогательного параметра касания, т.е. **tPar[splineIdx]=GCE_NULL_V**, но концевая точка касания сплайна должна лежать на другой кривой касания. Это можно обеспечить в момент создания сплайна или предварительно выполнить совпадение конца сплайна с другой кривой с помощью вызова **GCE_AddIncidence** и последующего вызова **GCE_Evaluate**.

S.3.12. Зеркальная симметрия

Зеркальная симметрия задается для тройки геометрических объектов. Первый и второй объекты являются зеркальным отражением друг друга, относительно третьего объекта – прямой, определяющей ось симметрии.

Ограничение симметрии создается вызовом метода

constraint_item **GCE_AddSymmetry**(GCE_system gSys, geom_item g[2], geom_item lObj).

S.3.13. Биссектриса

Ограничение биссектриса применяется для тройки линейных объектов, из которых первые два делят плоскость на четыре сектора, а третий объект – ось, делит выбранный сектор пополам. Биссектриса задается с помощью метода

constraint_item **GCE_AddAngleBisector**(GCE_system gSys,
geom_item l1, geom_item l2,
geom_item bl,
GCE_bisec_variant variant).

Исходные данные метода:

gSys – система геометрических ограничений;

I1, I2 – пара прямолинейных объектов, делящих плоскость на секторы;

b1 – биссектриса, прямая или отрезок прямой, делящей сектор между прямыми **I1** и **I2** пополам;

variant – выбор из трех вариантов, определяющий сектор биссектрисы:

- **GCE_BISEC_CLOSEST** выбирает один из секторов биссектрисы по относительному размещению объектов в начальном состоянии (см. параграф [S.4.3. Начальное приближение](#));
- **GCE_BISEC_PLUS** – биссектриса делит сектор, на который указывает векторная сумма направлений **I1** и **I2**;
- **GCE_BISEC_MINUS** – биссектриса делит сектор, на который указывает векторная разница направлений **I1** и **I2**.

Функция возвращает дескриптор ограничения, которое объявляет, что объект **b1** является биссектрисой для сектора, образованного прямыми **I1** и **I2**.

Также следует заметить, что данное ограничение не утрачивает смысл, если прямые **I1** и **I2** не пересекаются. В этом случае биссектриса делит пополам часть плоскости, заключенной между прямыми **I1** и **I2**, располагаясь от них на равных расстояниях.

S.3.14. Средняя точка

Ограничение *средняя точка* задается для тройки точек, которые связаны тем, что третья точка располагается на середине отрезка, образованного первой и второй точками. Данное ограничение создается вызовом метода

```
constraint_item GCE_AddMiddlePoint( GCE_system gcSys, geom_item pnt[3] ).
```

S.3.15. Взаимное размещение объектов

В параграфе [S.3.10. Касание](#) приведен пример касания окружности и прямой, которые могут иметь два варианта взаимного размещения, см. рис. S.3.10.1. C3D Solver запоминает взаимную ориентацию в момент создания ограничения и стремится сохранить её при последующих изменениях. Аналогичный выбор может стоять для линейных и угловых размеров. Например, линейный размер для точки и прямой, предусматривает, что при одном и том же знаке размера точка будет всегда оставаться с одной и той же стороны от прямой (относительно нормали).

Сохранение взаимного размещения объектов поддерживается для таких ограничений, как *касание*, *линейный* и *угловой* размеры. Остальные ограничения, такие как *параллельность*, *перпендикулярность*, *коллинеарность* и прочие допускают смену взаимного размещения. Например, перпендикулярность двух прямых может сменить взаимное размещение с левой пары векторов на правую, так что угол между их направлениями может быть как 90, так и 270 градусов.

S.4. ВЫЧИСЛЕНИЕ И ДИАГНОСТИКА ДВУМЕРНЫХ ОГРАНИЧЕНИЙ

S.4.1. Вычисление системы ограничений

Основная задача C3D Solver, удовлетворение системы ограничений эскиза, выполняемая с помощью функции API

GCE_result **GCE_Evaluate**(GCE_system **gSys**),
которая возвращает код результат вычислений системы ограничений **gSys**.

Функция не будет затрачивать время на вычисления, если на момент её вызова все ограничения удовлетворены.

Возвращаемые значения:

- **GCE_RESULT_Ok** означает успешное решение системы ограничений, решатель присвоил всем геометрическим объектам новое состояние, удовлетворяющее все заданные ограничения системы.
- **GCE_RESULT_Overconstrained** означает, что при попытке вычисления обнаружены ограничения противоречащие друг другу, т.е. в системе ограничений присутствует подмножество, в котором ограничения не могут быть удовлетворены одновременно. Состояние геометрии не меняется.
- **GCE_RESULT_Not_Satisfied** означает, что функция не нашла решения, удовлетворяющего всем заданным ограничениям. Состояние геометрии не меняется. Данный диагностический код возвращается при следующих ситуациях:
 - Размер вышел за пределы области значений, допускающих существование решения;
 - Положения фиксированных или замороженных точек вышли за область, допускающую существования решения.
 - В редких случаях – плохое начальное приближение. Удовлетворение ограничений требует значительные изменение параметрических координат;
 - Сильные взаимные влияния. Например заданы такие условия, что изменение угла на один градус приводит к смещению объекта на значительные расстояния.
 - В редких случаях – численная неустойчивость, обусловленная большим порядком разницы между объектами одного и того же чертежа, например размеры наименьшего и наибольшего объектов эскиза различаются более чем на 7 порядков (Например, один эллипс имеет размер 1 мм, а другой – 10 км.).
 - В общем случае – иные ситуации, когда решение не существует или не может быть вычислено.
- **GCE_RESULT_InvalidGeometry**. Найдено решение, которое приводит к вырождению геометрических объектов, такие как «отрезок или дуга нулевой длины», окружность или эллипс «нулевого радиуса». Состояние геометрии не меняется.

S.4.2. Изменение или получение состояния геометрии

Во время работы с геометрическими ограничениями приложение может осуществлять запрос состояния геометрии, вычисленное решателем, а решатель, в свою очередь, может передавать состояние координат объектов приложению.

Запрос состояния геометрии выполняют функции:

```
GCE_point GCE_GetPointXY( GCE_system gSys,  
                           geom_item g,  
                           point_type pName = GCE_PROPER_POINT );  
double GCE_GetCoordValue( GCE_system gSys, geom_item g, coord_name cName );  
double GCE_GetVarValue( GCE_system gSys, var_item var );
```

Для передачи состояния координат объектов из приложения в решатель используются функции:

```
bool GCE_SetPointXY( GCE_system gSys,
```



```

geom_item g,
point_type pName,
GCE_point xyVal );
bool      GCE_SetCoordValue( GCE_system gSys,
                             geom_item g,
                             coord_name cName, double crdVal );
bool      GCE_SetVarValue( GCE_system gSys, var_item var, double val );

```

S.4.3. Начальное приближение

Создание параметрического чертежа – это пошаговое внесение изменений, добавление или удаление объектов и ограничений, варьирование размеров. Каждый новый запрос на вычисление системы ограничений опирается на начальное состояние геометрических объектов, оставшееся в результате предыдущего запроса **GCE_Evaluate**.

Важно понимать, что начальное состояние геометрии является неотъемлемым условием задачи удовлетворения ограничений. Два эскиза, имеющие одинаковый набор геометрических объектов и ограничений, но отличающиеся начальным состоянием, могут иметь различные решения. C3D Solver принимает во внимание начальное состояние и стремится удовлетворить все заданные ограничения с минимальными отклонениями от начального приближения при минимальном количестве изменяемых объектов.

Начальное приближение имеет определяющее значение не только при запросе на вычисление, но и в момент создания ограничений. Взаимное размещение геометрических объектов показывает вариант формулировки ограничения. Например, если в момент создания касания двух окружностей (**GCE_AddTangent**), одна окружность располагается внутри другой, то решатель сформулирует касание, при котором одна окружность всегда будет оставаться внутри другой. Подробнее о взаимном размещении геометрических объектов см. параграф [S.3.15. Взаимное размещение объектов](#).

S.4.4. Переопределенные совместные и несовместные системы ограничений

Многие практические эскизы оказываются переопределены лишними ограничениями. В самом простом случае ограничения могут дублировать друг друга, например, два касания для одной и той же пары окружностей. В более общем случае лишнее ограничение полностью удовлетворяется другими ограничениями, обеспечивающими его геометрическое условие. Например, если между двумя прямыми задана «перпендикулярность» и одновременно заданы «горизонтальность» для одной прямой и «вертикальность» для другой, то любое из этих трех ограничений можно считать лишним. C3D Solver исключает избыточные ограничения из вычислений. Данный тип переопределения имеет отношение к **совместным** системам, допускающим лишние ограничения, если они не противоречат друг другу.

Также переопределение может стать причиной **несовместности** системы ограничений, когда лишнее ограничение противоречит другим ограничениям, т.е. не может быть удовлетворено одновременно с другими. Для исправления данной ситуации необходимо удалить одно из ограничений, участвующих в группе избыточности. Когда лишнее ограничение является причиной несовместности системы, метод **GCE_Evaluate** возвращает код ошибки **GCE_RESULT_Overconstrained**.

Блокированные размеры. Рекомендуется избегать избыточности ограничений, даже если они не приводят к противоречиям. В частности, избыточные ограничения могут блокировать управляющие размеры, созданные для параметризации эскиза. Например, на рис S.4.4.1. слева изображен параметрический прямоугольник, который определен тремя управляющими размерами (это расстояния 40, 40 и 60) и попарными вертикальностями и горизонтальностями (символы V и H). В данном примере присутствует избыточная цепочка ограничений, включающая в себя два ограничения V, два ограничения H и два размера на 40. Вся цепочка удовлетворена, однако ни один из двух управляющих размеров на 40 не может быть изменен отдельно без соответствующего изменения другого. Для освобождения размера потребуется сделать выбор: удалить один из размеров на 40 или удалить одно из ограничений горизонтальность/вертикальность. Эскиз справа показывает выбор, удаляющий вертикальность верхнего отрезка. Тогда все размеры можно менять. Заметим, что размер

на 60 в цепочке избыточных ограничений не участвует, что говорит о том, что любое переопределение вовлекает какую-то группу ограничений, не обязательно весь эскиз, а какую-то его часть. Любой размер, попавший в группу переопределения становится блокированным. Диагностика блокированных размеров рассмотрена в параграфе [S.4.10. Тест избыточности](#).

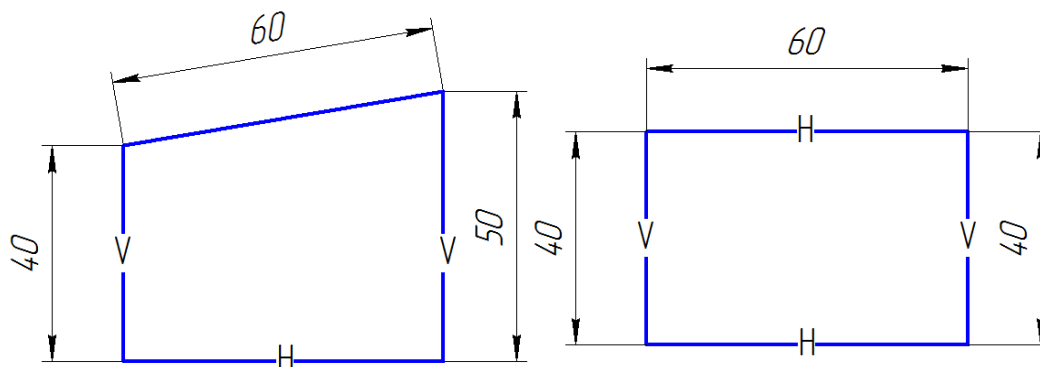


Рис S.4.4.1.

S.4.5. Недоопределенные системы ограничений

В предыдущем параграфе обсуждалась ситуация переопределения, когда в системе ограничений присутствует больше ограничений, чем необходимо. В данном параграфе обсуждается другая ситуация, естественная для процесса черчения, когда в эскизе остаются геометрические объекты, не полностью определенные ограничениями. В начале черчения эскиз может вообще не иметь ограничений, а все геометрические объекты будут иметь полную степень свободы. Соответственно, чем больше логических ограничений и размеров добавляется в эскиз, тем меньше степени свободы остается объектам. У полностью определенного эскиза не остается и одной степени свободы. Заметим, что в одном и том же эскизе одновременно могут быть обе ситуации – часть эскиза переопределена, остальная часть может быть недоопределена, и в тоже время, какая-то часть геометрии может быть определена полностью.

Большая часть методов решения, реализованных в C3D Solver, ориентирована именно на недоопределенные системы ограничений. Кроме того, используются некоторые преимущества недоопределенных случаев для разбиения системы ограничений в целом на последовательность подсистем размером меньше, вычисляемых одна после другой. В тех или иных ситуациях недоопределенности решатель выбирает одно из оптимальных решений, основываясь на следующих принципах:

Для удовлетворения ограничений меняется как можно меньше геометрических объектов;

Те геометрические объекты, которые затрагиваются решением, получают минимальное смещение от своего начального приближения (см. параграф [S.4.3. Начальное приближение](#)).

S.4.6. Анализ степеней свободы

В процессе параметрического черчения важно контролировать, какие из геометрических объектов эскиза уже полностью определены, а какие нуждаются в дополнительных ограничениях. В параграфе [S.4.5. Недоопределенные системы ограничений](#) обсуждались принципы выбора решения для тех геометрических объектов, которым не хватает ограничений, что бы однозначно определить свое состояние. Показать пользователю (визуализировать) недоопределенные геометрические объекты помогает функции анализа степеней свободы, одна из которых вычисляет степень свободы точки:

GCE_point_dof **GCE_PointDOF**(GCE_system gSys, geom_item pnt).

Данный метод позволяет определить текущую степень свободы точки, которая может быть контрольной точкой любого геометрического объекта, как например центр окружности или эллипса, контрольная точка сплайна, один из концов отрезка и т.д.

S.4.7. Информационные запросы

В данном параграфе приведен ряд функций, с помощью которых приложение может узнать различные сведения об объектах, зарегистрированных в системе ограничений, или результаты диагностики и вычислений. Эти функции предназначены для осуществления справочных запросов, поэтому их вызов не приведет к изменению состояния системы ограничений или отдельных её данных.

Тип геометрии. Тип геометрического объекта, зарегистрированного в системе ограничений, можно узнать с помощью функции

geom_type **GCE_GeomType**(GCE_system gSys, geom_item g),
значением которой является перечислительный тип геометрического объекта **g**, зарегистрированного в системе **gSys**.

Возвращаемые значения:

- GCE_POINT – двумерная точка;
- GCE_LINE – двумерная прямая;
- GCE_CIRCLE – двумерная окружность;
- GCE_ELLIPSE – двумерный эллипс;
- GCE_SPLINE – двумерный сплайн;
- GCE_PARAMETRIC_CURVE – двумерная параметрическая кривая общего вида;
- GCE_BOUNDED_CURVE – кривая, ограниченная концевыми точками.

Ниже приведен перечень функций для информационных запросов.

Тип базовой кривой возвращает функция

geom_type **GCE_BaseCurveType**(GCE_system gSys, geom_item crv).

Связность с ограничениями оценивает функция

bool **GCE_IsConstrainedGeom**(GCE_system gSys, geom_item g).

Удовлетворенность ограничения оценивает функция

bool **GCE_IsSatisfied**(GCE_system gSys, constraint_item cItem)

Вырожденные геометрические объекты возвращает функция

std::vector<geom_item> **GCE_DiagnoseGeometry**(GCE_system gcSys).

Текущее заданное значение размера (не измеренное) возвращает функция

double **GCE_DimensionParameter**(GCE_system gSys, constraint_item dItem).

Оценку соответствия координат точки заданным ограничениям выдает функция

bool **GCE_CheckPointSatisfaction**(GCE_system gSys, geom_item pnt, point_type cp, double px, double py).

Состояние решения данного ограничения возвращает функция

GcConState **GCE_GetConstraintState**(GCE_system, constraint_item gc_item).

Состояние системы ограничений по результатам последнего решения возвращает функция

GcConstraintStatus **GCE_GetConstraintStatus**(GCE_system gSys).

S.4.8. Драггинг геометрических объектов

В данном параграфе приведены функции, обслуживающие режим для интерактивного манипулирования недоопределенным эскизом, называемым драггинг.

Инициализировать режим драггинга контрольной точки объекта:

GCE_result **GCE_PrepareDraggingPoint**(GCE_system gSys, GCE_dragging_point drgPnt, GCE_point curXY).

Инициализировать режим драггинга контрольной точки множества объектов:

GCE_result **GCE_PrepareDraggingPoint**(GCE_system gSys
, const std::vector<GCE_dragging_point> & cPntArr
, GCE_point curXY).

Инициализировать режим перетаскивания множества объектов:

GCE_result **GCE_PrepareMovingGeoms**(GCE_system gSys
, std::vector<geom_item> & geoms
, GCE_point curXY).

Переместить указатель драггинга:

GCE_result **GCE_MovePoint**(GCE_system gcSys, GCE_point curXY).

S.4.9. Геометрическая трансформация

C3D Solver позволяет осуществить геометрическое преобразование параметрического эскиза по матрице, которая может содержать сдвиг, поворот и масштабирование. Для выполнения указанной трансформации необходимо для системы ограничений эскиза вызвать метод

```
GCE_result GCE_Transform( GCE_system gSys, const MbMatrix & mat ).
```

Следует принимать во внимание, что данный метод осуществляет преобразование, как геометрии, так и ограничений. Например, линейные размеры могут менять свои значения, если матрица трансформации содержит масштабирование.

Надо заметить, что в результате трансформации возможно, что часть ограничений станет неудовлетворенной, о чем можно запросить с помощью вызова **GCE_IsSatisfied**. Ограничения, которые сохранили удовлетворенность, называются *инвариантными* относительно данной трансформации.

S.4.10. Тест избыточности

C3D Solver предусматривает различные способы, позволяющие оценить избыточность ограничений. Методы

```
GCE_result GCE_DeviateDimension( GCE_system gSys, constraint_item dItem, double delta );
```

```
GCE_result GCE_DeviationTest( GCE_system gSys, constraint_item dItem, double delta ).
```

оценивают избыточность ограничений на основе отклонения размерных ограничений.

S.5. ДВУМЕРНЫЕ СПЛАЙНЫ И ПАРАМЕТРИЧЕСКИЕ КРИВЫЕ

В этой главе рассматривается работа с такими кривыми, как сплайн, параметрическая кривая общего типа, а также эллипс. Эти кривые объединяет то, что C3D Solver принимает во внимание не только геометрию, но и параметризацию данных кривых.

S.5.1. Сплайновые кривые

C3D Solver использует NURBS-кривые в качестве математической основы сплайновых кривых. NURBS-представление стало фактически промышленным стандартом для представления кривых свободной формы и поддерживается практически всеми САПР. Общеизвестным каноническим трудом на данную тему является книга "The NURBS Book" авторов Les Piegl и Wayne Tiller (издательство Springer).

C3D Solver предоставляет два способа определения сплайна:

- Сплайн по набору характеристических точек, заданных значениями их координат;
- Сплайн, проходящий через набор интерполяционных точек, заданных дескрипторами, ранее зарегистрированных геометрических объектов.

Сплайн на основе NURBS-кривой создается с помощью метода

```
geom_item GCE_AddSpline( GCE_system gSys, GCE_spline spl ).
```

Функция принимает структуру данных `GCE_spline`, которая содержит в себе информацию, определяющую сплайн: начальные значения контрольных точек, массив интерполяционных точек (при необходимости), веса, признак замкнутости и т.д.

Специальное ограничение, фиксирующее вектор производной сплайна, создается методом

```
constraint_item GCE_FixSplineDerivative( GCE_system gSys, geom_item spline,  
double par, uint derOrder, GCE_vec2d * fixVal = NULL );
```

S.5.2. Параметрические кривые общего вида

Применения параметрической кривой. Одно из полезных применений параметрической кривой связано с участием в эскизе неподвижных кривых с не поддерживаемым типом геометрии. Например, планарный эскиз может быть создан в контексте трехмерной модели САПР, откуда в эскиз могут приходиться произвольной формы проекции 3D-кривых. Проекционные кривые, участвующие в системе ограничений в виде параметрических кривых, дают возможность привязывать геометрию эскиза к проекционным объектам. Другое применение это оформление чертежей, фрагменты которых получены проецированием геометрии из 3D-моделей и сохраняющими с ней ассоциативную связь.

S.5.3. Ограничения, основанные на параметризации кривых

Для создания специальных ограничений кривых в параметрическом представлении, а также сплайна и эллипса, используются методы:

```
constraint_item GCE_AddPointOnPercent( GCE_system gSys,  
geom_item curve,  
geom_item pnt[3],  
double k ),  
constraint_item GCE_AddPointByMetricPercent( GCE_system gSys,  
geom_item curve,  
geom_item pnt[3],  
double k ),  
constraint_item GCE_AddFixCurvePoint( GCE_system gSys,  
geom_item curve,  
geom_item pnt ),  
constraint_item GCE_AddPointOnParEllipse( GCE_system gSys,
```

geom_item **pnt**,
geom_item **ellipse**, double **t**).

Т.1. ОБМЕН ДАННЫМИ С ДРУГИМИ СИСТЕМАМИ

Конвертеры геометрического ядра C3D обеспечивают обмен данными с другими системами. В свою очередь, обмен данными включает в себя две задачи.

Первая задача — импорт. Это чтение из файла или потока сторонней геометрической модели в одном из обменных форматов и построение на основе прочитанных данных геометрической модели C3D.

Вторая задача — экспорт. Это представление геометрической модели C3D в одном из обменных форматов и запись этого представления в указанный поток или файл.

Максимальный набор данных, которые могут передаваться посредством конвертера, таков:

1. Информация о форме, которая может быть описана с помощью тел, поверхностей, проволочных каркасов и групп точек.
2. Информация о структуре моделей: выделены составляющие её компоненты, которые могут быть многократно использованы в модели.
3. Информация об изделиях: идентификатор, комментарии, сведения об авторах и пр.
4. Атрибуты: визуальные свойства, элементарные атрибуты.
5. Элементы аннотации: размеры, технические требования, обозначения.

Геометрическое ядро C3D поддерживает шесть текстовых форматов и два бинарных. Четыре текстовых формата ACIS, IGES, STEP, X_T и бинарный формат X_B передают информацию о геометрической форме моделируемого объекта с помощью граничного представления (Boundary representation). Два текстовых формата STL, VRML и бинарный формат STL для передачи геометрической формы моделируемого объекта используют полигональное представление (Polygonal representation).

Т.1.1. Принципы работы конвертера

Работа конвертера предполагает использование интерфейсов, которые можно разделить на две группы. В первую входят интерфейсы, которые имеют реализацию на стороне C3D, во вторую — те интерфейсы, которые должны быть реализованы пользователем. Кроме того, геометрическое ядро C3D предоставляет стандартную реализацию всех интерфейсов, необходимых для обмена данными.

Интерфейсы соответствуют следующим понятиям:

- Конвертер (интерфейс `IConvertor3D`) — специальный объект, который используется для передачи данных. Реализован на стороне C3D.
- Свойства конвертера (интерфейс `IConvertorProperty3D`) — служат для передачи настроек обмена в конвертер. Интерфейс имеет стандартную реализацию.
- Документ (интерфейс `ItModelDocument`) — объект, которому соответствует передаваемая модель в целом. Интерфейс имеет стандартную реализацию.
- Деталь (интерфейс `ItModelDetail`) и сборка (интерфейс `ItModelAssembly`) — части модели, которые могут быть представлены как детали или как сборочные единицы. Интерфейсы имеют стандартную реализацию.
- Вставка (интерфейс `ItModelInstance`) — объект, отвечающий за позиционирование и/или повторное использование компонента в модели. Интерфейс имеет стандартную реализацию.

Обобщающим понятием детали и сборки является компонент. Предполагается, что компонент может быть минимальной единицей, которой может соответствовать отдельный файл, с которым работает конечное приложение. Использование компонентов и вставок позволяет сформировать дерево модели, в котором компоненты могут быть многократно использованы.

Т.1.2. Порядок работы с конвертером

Для начала работы с конвертером необходимо получить его экземпляр.

Определённая в глобальной области видимости функция `IConvertor3D * GetConvertor3D ()`

создает конвертер, который может выполнять импорт и экспорт моделей в любом из поддерживаемых форматах. Функция не имеет параметров. В случае успеха функция возвращает конвертер, в противном случае возвращает нулевой указатель.

Функция объявлена в файле `conv_i_converter.h`.

После создания конвертера обмен данными осуществляется путем вызова методов конвертера.

Методы конвертера

```

MbeConvResType SATRead ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        std::istream * stream,
                        IProgressIndicator * indicator = 0 ),
MbeConvResType SATWrite ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        std::ostream * stream,
                        IProgressIndicator * indicator = 0 ),
MbeConvResType SATRead ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType SATWrite ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType IGSRead ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType IGSWrite ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType XTRead ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType XTWrite ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType STEPRead ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType STEPWrite ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType STLRead ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType STLWrite ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,
                        MbRefItem * queryStitch = 0 ),
MbeConvResType VRMLRead ( IConverterProperty3D & property,
                        ItModelDocument & document,
                        IProgressIndicator * indicator = 0,

```



```

MbeConvResType VRMLWrite ( MbRefItem * queryStitch = 0 ),
( IConvertorProperty3D & property,
  ItModelDocument & document,
  IProgressIndicator * indicator = 0,
  MbRefItem * queryStitch = 0 ),

```

выполняют импорт и экспорт модели в форматах SAT, IGES, X_T, STEP, STL, VRML, соответственно.

Входными параметрами метода являются:

- **property** – свойства конвертера,
- **stream** – поток для получения или записи модели обменного формата.
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи,
- *queryStitch* – запросчик сшивки поверхностей или единиц измерения при импорте (устаревший параметр, его не рекомендуется использовать, для управления сшивкой следует пользоваться свойствами конвертера).

В случае успеха метод возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

По окончании работы полученный экземпляр конвертера необходимо удалить.

Конвертер наследует интерфейс класса `IConvertor3D`, который объявлен в файле `conv_i_converter.h`.

T.1.3. Свойство конвертера `IConvertorProperty3D`

Свойство конвертера (или просто свойство) является одним из параметров методов обмена данными. Поскольку один и тот же интерфейс `IConvertorProperty3D` используется для управления работой конвертеров всех форматов, то он содержит как методы, общие для всех форматов, так и специфичные для того или иного формата. Кроме того, отличается набор данных, которые управляют работой экспорта и импорта.

Методы, которые вызываются во всех случаях, отвечают за получение пути файла обменного формата (если явно не указан поток), настройку фильтра передаваемых объектов по типам и за журнал обмена данными:

```

const c3d::path_string FullFilePath () const;
bool GetIoPermission( MbeIOPermiss nPermission );
void GetIoPermissions( std::vector<bool>& ioPermissions );
void LogReport( ptrdiff_t id, eMsgType msgType, eMsgDetail msgText );

```

Методы, которые вызываются при экспорте в любой формат, отвечают за выдачу ряда сведений о документе в целом, о системе координат, относительно которой геометрическая модель должна быть ориентирована, а также запрашивают, следует ли принудительно трансформировать объекты обменного формата так, чтобы сохранялась их форма и взаимное расположение, но использовались только правые системы координат:

```

bool GetPropertyString ( MbeConverterStrings nString, std::string & propertyString );
MbPlacement3D GetOriginLocation();
bool ReplaceLocationsToRight();

```

Методы, которые вызываются при импорте из форматов SAT, X_T, X_B, STEP и IGES, выдают запрет или разрешение на автоматическую сшивку отдельных поверхностей в тела и указывают точность сшивки:

```

bool EnableAutoStitch( double& stitchPrecision );

```

При импорте из IGES и SAT вызывается метод, который разрешает или запрещает конвертеру обратиться к запросчику:

```

bool CanShowMessages();

```

Остальные методы являются специфическими для экспорта в разные форматы или введены для отладки.

При экспорте в форматы Parasolid (выбор X_T или X_B) и STL вызывается метод, определяющий, будут ли данные предоставляться в текстовом или двоичном виде:

```

bool IsFileAscii ();

```

При экспорте в STEP вызываются методы, имеющие реализацию по умолчанию, и служащие для задания представления текста в элементах аннотации и прикладного протокола STEP, который будет использоваться:

```
eTextForm GetAnnotationTextRepresentation ();
```

```
MbeImpExpFormat GetFormat ();
```

При экспорте в IGES вызывается метод, который определяет, будет ли экспортироваться информация о топологии:

```
bool IsOutOnlySurfaces();
```

При экспорте в полигональные форматы (STL и VRML) вызываются методы, имеющие реализацию по умолчанию, и управляющие параметрами расчёта триангуляции:

```
MbStepData TesselationParameters();
```

```
bool DualSeams();
```

```
void DualSeams( bool dSeams );
```

Следующие методы введены для отладки:

```
std::string GetDocumentName() const;
```

```
bool IsAssembling () const;
```

```
void SetIoPermission( MbeIOPermiss nPermission, bool setF );
```

```
void SetPropertyString ( MbeConverterStrings nString, const std::string & propertyString );
```

```
bool ExportComponentsSeparately ().
```

Геометрическое ядро C3D предоставляет стандартную реализацию интерфейса IConvvertorProperty3D.

Класс ConvConvvertorProperty3D представляет собой стандартную реализацию свойства конвертера, рис. Т.1.3.1. Свойство наследует интерфейс класса IConvvertorProperty3D.

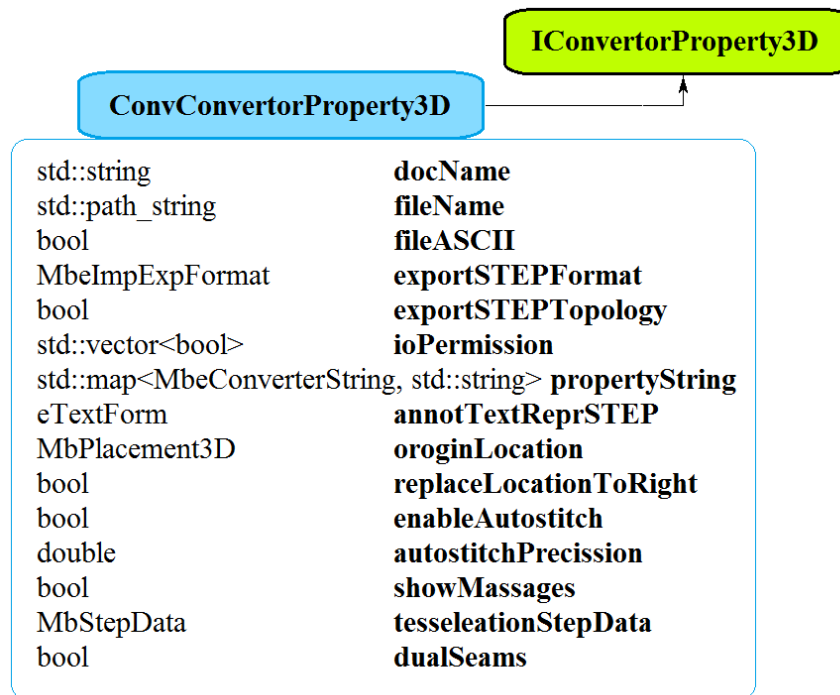


Рис. Т.1.3.1

В своих полях, к которым открыт доступ, класс ConvConvvertorProperty3D хранит следующую информацию:

- **fileName** – путь к файлу обменного формата (по умолчанию пустая строка),
- **docName** – название документа (по умолчанию пустая строка),
- **fileASCII** – признак экспорта в текстовый файл (по умолчанию true),
- **exportSTEPFormat** – протокол формата STEP для экспорта (по умолчанию ief_STEP203), флаг экспорта топологии в формат IGES,
- **exportIGESTopology** – (по умолчанию true),
- **ioPermission** – фильтр объектов по типам (обеспечивает разрешение экспортировать и импортировать объекты любого типа),
- **propertyStrings** – особые сведения о документе (пустой контейнер),
- **annotTextReprSTEP** – представление текста аннотационных элементов (по умолчанию exf_TextOnly),

- **originLocation** – локальную систему координат модели (по умолчанию),
- **enableAutostitch** – признак сшивки граней (по умолчанию true),
- **autostitchPrecision** – точность сшивки граней (по умолчанию 0.3),
- **tessellationStepData** – параметры триангуляции (по умолчанию),
- **showMessages** – управление обращением к запросчику (по умолчанию false),
- **dualSeams** – признак сшивки швов (по умолчанию true),
- **logRecords** – контейнер для кодов журнала обмена данными (пустой контейнер).

Интерфейс IConvertorProperty3D объявлен в файле conv_i_converter.h.

Класс ConvConvertorProperty3D объявлен в файле conv_model_properties.h.

T.1.4. Документ модели ItModelDocument

Интерфейс ItModelDocument входит в число интерфейсов, использование которых позволяет передавать данные, которые имеют сложную древовидную многокомпонентную структуру, в которой допускается многократное использование компонентов. Документ предоставляет два способа работы с моделью. Первый — прямое использование модели, формируемой с использованием объектов геометрического ядра C3D. Второй — пользователь реализует интерфейсы [ItModelDetail](#), [ItModelAssembly](#), [ItModelInstance](#) и методы, которые работают с реализациями. Конвертер в свою очередь проводит обращение к интерфейсным методам возвращаемых объектов.

Методы интерфейса группируются следующим образом.

Отдельно стоит метод, возвращающий единицы измерения длины:

MbeModelLengthUnits [GetLengthUnits\(\)](#).

Объект, к которому относятся единицы длины, различный для полигональных форматов (STL и VRML) и для форматов, поддерживающих граничное представление. В первом случае считается, что единицы измерения относятся к данным обменного формата, а все размеры исходной модели выражены в миллиметрах. Во втором случае имитируются единицы длины исходной модели.

Группа методов, предназначенных для работы с моделью и контейнером элементов аннотации напрямую:

```
void SetContent( MblItem* content);
MblItem * GetContent();
map_of_visual_items GetAnnotationItems( eTextForm form );
void SetAnnotationItems( const map_of_visual_items& visItems );
```

Методы, которые ориентированы на использование интерфейсов, в свою очередь делятся на следующие подгруппы.

Группа методов, предназначенных для формирования модели при импорте:

```
SPtr<ItModelAssembly> CreateAssembly( const std::vector< SPtr<MblItem> > & componentItems, const
c3d::path_string& fileName );
SPtr<ItModelDetail> CreateDetail( const std::vector< SPtr<MblItem> > & componentItems,
const c3d::path_string& fileName );
```

Группа методов, предназначенных для получения связанных компонент модели при экспорте:

```
bool IsAssembly();
bool IsEmpty();
SPtr<ItModelAssembly> GetInstanceAssembly();
SPtr<ItModelDetail> GetInstanceDetail();
```

Методы, используемые для отладки:

```
bool FinishImport( IProgressIndicator * indicator);
void OpenDocument();
```

Поскольку заранее неизвестно, какая реализация будет применяться для передачи данных, то при импорте вызывается как метод [SetContent](#), так и один из методов [CreateAssembly](#) или [CreateDetail](#), в зависимости от результата импорта. Соответственно, при экспорте сначала вызываются методы [GetInstanceAssembly](#) или [GetInstanceDetail](#), в зависимости от результата [IsAssembly](#), и, только если конвертеру не удастся сформировать модель, вызывается метод [GetContent](#).

В модели, сформированной с использованием объектов геометрического ядра C3D, некоторые объекты типа [MblItem](#) соответствуют компонентам. Это определяет структуру и содержимое контейнера элементов аннотации, с которым работают методы [GetAnnotationItems](#) и [SetAnnotationItems](#).

В геометрическом ядре C3D имеется две стандартные реализации интерфейса [ItModelInstance](#): RegularModelDocument и ConvModelDocument.

Класс RegularModelDocument ориентирован на работу с использованием интерфейсных методов. Эта реализация и соответствующие ей реализации интерфейсов [ItModelDetail](#), [ItModelAssembly](#), [ItModelInstance](#) формируют с использованием объектов геометрического ядра C3D корректную с точки зрения конвертера модель, и может быть рекомендована в качестве генератора таковой. Модель может быть получена как результат вызова метода [GetContent](#). Работа с единицами измерения не предусмотрена.

Класс ConvModelDocument представляет собой одну из возможных реализаций документа модели, рис. Т.1.4.1. Документ наследует интерфейс класса ItModelDocument. Первой особенностью этой реализации является то, что использование интерфейсов не предполагает передачу сведений об изделии и визуальных свойствах. Вторая особенность заключается в том, что модель, сформированная с использованием объектов геометрического ядра C3D, доступ к которой осуществляется с помощью методов [GetContent](#) и [SetContent](#), не связана с той моделью, которая формируется на основе интерфейсных вызовов. Данная реализация используется в тестовом приложении для передачи модели напрямую, без вызова интерфейсных методов.

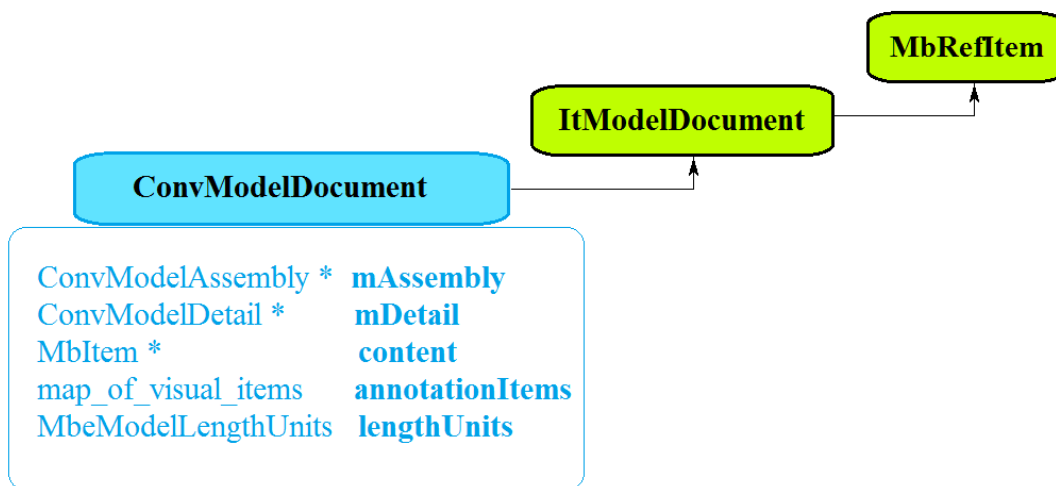


Рис. Т.1.4.1

Класс ConvModelDocument в качестве модели может содержать или сборочную единицу **mAssembly**, или деталь **mDetail**, или произвольный геометрический объект **content**, а также аннотационные объекты геометрической модели **annotationItems**. Кроме того, класс ConvModelDocument содержит единицы измерения **lengthUnits**, в которых представлена модель. По умолчанию все указатели нулевые, контейнеры пустые, единицы измерения соответствуют миллиметрам.

Интерфейс ItModelDocument объявлен в файле conv_i_converter.h. Классы ConvModelDocument и RegularModelDocument объявлены в файле conv_model_properties.h.

Т.1.5. Индикатор процесса выполнения IProgressIndicator

Индикатор процесса выполнения (или просто индикатор) обеспечивает обратную связь конвертера с пользователем. В ходе обмена данными конвертер обращается к различным методам.

Для установки диапазона шагов, размера шага и отображаемых сведений о процессе служит метод `bool Initialize(size_t range, size_t delta, IStrData & message)`.

Для сообщения о выполнении некоторого числа шагов служит метод `bool Progress(size_t n)`.

Метод возвращает признак того, следует ли продолжать операцию.

Для получения сведений о том, следует ли прерывать операцию по требованию пользователя, служит метод `bool IsCancel()`.

Для информирования пользователя о том, что операция завершена, служит метод `void Success()`.

Для информирования пользователя об аварийной остановке процесса служит метод `void Stop()`.

Пользователь может определить способ передать информацию о том, следует ли выполнить принудительную остановку процесса. В качестве стандартного способа пользователю предлагается реализовать метод

`void SetCancel(bool cancel)`

Интерфейс `IProgressIndicator` объявлен в файле `alg_indicator.h`.

В тестовом приложении класс `ProgressIndicatorImp` представляет собой одну из возможных реализаций индикатора.

Класс `ProgressIndicatorImp` объявлен и реализован в файле `test_converter.cpp` тестового приложения геометрического ядра `C3D`.

T.1.6. Архитектура дерева модели

Для модели, сформированной с использованием объектов геометрического ядра `C3D`, приняты следующие соглашения.

- Документ эквивалентен корневому компоненту (компоненту высшего уровня).
- Компоненту всегда соответствует объект типа [MbAssembly](#) в случае формата с граничным представлением. В случае формата с полигональным представлением в качестве компонента допустим объект типа [MbMesh](#). Ожидается, что на компонент назначены свойства, и это выражается в том, что компонент обладает атрибутами типа `MbProductInfo` и `MbPersonOrganizationInfo`.
- Собственными объектами компонента могут быть объекты типа [MbSolid](#), [MbWireFrame](#), [MbPointFrame](#), [MbSpaceInstance](#) и `MbPlaneInstance` для форматов с граничным представлением, либо [MbMesh](#) для полигональных форматов, которые лежат в [MbAssembly](#).
- Вложенность компонент и их позиционирование осуществляется с помощью объектов типа [MbInstance](#). `MbInstance` лежит в компоненте верхнего уровня и ссылается на компонент нижнего уровня. Циклические зависимости недопустимы.

Подход, согласно которому модель для обмена данными формируется средствами геометрического ядра `C3D`, имеет ряд недостатков. Наиболее существенный из них заключается в том, что использование конкретных объектов и конкретной конфигурации затрудняет развитие функционала: внесение любых изменений приведёт к необходимости внесения ответных изменений у всех пользователей. Использование интерфейсов модели, компонента и вставки, реализуемых на стороне пользователя, даёт последним независимость от конкретной реализации.

В терминах интерфейсов архитектура выглядит следующим образом.

- Документ [ItModelDocument](#) содержит в себе компонент высшего уровня.
- Компоненты бывают двух типов: детали [ItModelDetail](#) и сборки [ItModelAssembly](#). Разница между ними лишь в интерпретации, оба типа могут ссылаться на компоненты более низкого уровня.
- Собственными объектами компонентов могут быть объекты типа [MbSolid](#), [MbWireFrame](#), [MbPointFrame](#), [MbSpaceInstance](#) и [MbPlaneInstance](#).
- Вложенность компонент и их позиционирование осуществляется с помощью объектов [ItModelInstance](#). Циклические зависимости недопустимы.

T.1.7. Свойства элементов модели `ItModelInstanceProperties`

`ItModelInstanceProperties` — базовый класс, предок интерфейсов [ItModelDetail](#), [ItModelAssembly](#), [ItModelInstance](#). В интерфейсе объявлены методы, которые можно группировать следующим образом.

Первая группа методов обеспечивает передачу сведений о компоненте.

Идентификатор изделия передают методы:

`std::string Name()`,

`bool SetName(const std::string& name)`.

Передачу обозначения изделия выполняют методы:

`std::string Marking()`,

bool **SetMarking**(const std::string& **marking**).

Передачу сведений об авторе выполняют методы:

std::string **Author**(),

bool **SetAuthor**(const std::string& **author**).

Передачу сведений об организации выполняют методы

std::string **Organization**(),

bool **SetOrganization**(const std::string& **organization**).

Передачу комментариев выполняют методы:

std::vector< std::string> **GetComments**(),

bool **SetComments**(const std::vector< std::string>& **comments**).

Вторая группа методов предназначена для передачи визуальных свойств.

Для передачи собственных визуальных свойств компонента или вставки предназначены методы:

bool **GetColor**(MbAttributeContainer& **visual**),

bool **SetColor**(const MbAttributeContainer& **visual**).

Для передачи визуальных свойств объектов, которые принадлежат какому-либо из элементов, например, граням тел, предназначены методы:

bool **GetColor**(MbAttributeContainer& **visual**, const MbName& **name**),

bool **SetColor**(const MbAttributeContainer& **visual**, const MbName& **name**).

Для передачи визуальных свойств какого-либо объектов, принадлежащего компоненту, например, тела, индекс которого известен, предназначен метод:

bool **SetColor**(const MbAttributeContainer& **visual**, size_t **index**).

Третья группа методов предназначена для передачи технических требований, относящихся к компонентам. К этой группе относятся методы:

void **GetRequirements**(vector_of_annotation& **annot**, eTextForm **textRepr**),

void **SetRequirements**(const vector_of_annotation& **annot**).

T.1.8. Компоненты ItModelDetail и ItModelAssembly

Интерфейсы ItModelDetail и ItModelAssembly имеют одинаковый набор методов, поэтому их можно считать идентичными с точностью до интерпретации. Назначение свойств компонентов, визуальных свойств и технических требований реализуется посредством методов базового класса [ItModelInstanceProperties](#).

Метод **PureFileName**() вызывается при экспорте и возвращает строку, смысл которой — имя файла компонента.

Работа с собственными объектами осуществляется с помощью следующей группы методов:

void **GetItems**(std::vector< SPtr<MbItem> > & **items**, MbeGettingItemType **itemType**, bool **includeInvisible**),

\void **AddItems**(const std::vector< SPtr<MbItem> > & **items**).

Метод **GetItems** предназначен для выдачи элементов в контейнер с использованием фильтров типа и видимости, которые формируются на основе значений, возвращаемых методом [IConverterProperty3D::GetIoPermission](#).

Метод **AddItems** используется при импорте и предназначен для добавления элементов в компонент.

Для работы со вставками предназначены методы:

SPtr<ItModelInstance> **PrepareInstance**();

SPtr<ItModelInstance> **NextInstance**(bool **includeInvisible**).

Метод **PrepareInstance** предназначен для осуществления вставки компонента при импорте.

Метод **NextInstance** представляет собой итератор вставок при экспорте. Предполагается, что итератор готов к выдаче вставок с начала. Признаком окончания является возврат нулевого указателя. Значение фильтра видимости в процессе обхода соответствует результату вызова метода [IConverterProperty3D::GetIoPermission](#) (iop_wInvisible).

Для получения и задания элементов аннотации при экспорте и импорте, соответственно, предусмотрены парные методы:

vector_of_annotation **GetAnnotationItems**(eTextForm **textForm**, bool **includeInvisible**),

void **SetAnnotationItems**(const vector_of_annotation & **annot**).

Метод **GetAnnotationItems**(eTextForm **textForm**) в конвертерах не вызывается.

Интерфейсы ItModelDetail и ItModelAssembly объявлены в файле conv_model_properties.h

T.1.9. Вставки ItModelInstance

Наследование интерфейса вставки ItModelInstance от [ItModelInstanceProperties](#) сложилось исторически, и сохранено для совместимости. В рамках сложившегося подхода корректным является только назначение собственных визуальных свойств вставке.

Метод `void * GetId\(\)` позволяет различать вставки эквивалентных компонентов. Если различные вставки возвращают одинаковое значение, эквивалентность определяется по результатам анализа содержания. В процессе экспорта конвертер стремится к оптимизации модели за счёт многократного использования эквивалентных компонент. Пользователь может предоставить информацию о том, какие компоненты точно не являются эквивалентными для сокращения вычислительных затрат при экспорте.

При экспорте для получения такой информации о вставке, как локальная система координат вставки, содержимое вставки, отсутствие наполнения вставки, используется методы:

```
bool GetPlacement( MbPlacement3D & place ),  
bool IsAssembly(),  
bool IsEmpty().
```

Если вставка не пуста, то в зависимости от результата метода [IsAssembly\(\)](#) вызывается один из методов:

```
SPtr<ItModelAssembly> GetInstanceAssembly(),  
SPtr<ItModelDetail> GetInstanceDetail().
```

Эти методы возвращают интерфейс компонента, на который ссылается вставка.

Для создания компонентов при импорте предназначены методы:

```
SPtr<ItModelAssembly> CreateAssembly( const MbPlacement3D &place,  
                                const std::vector<SPtr<MbItem> > &  
                                componentItems,  
                                const c3d::path_string& fileName );  
SPtr<ItModelDetail> CreateDetail( const MbPlacement3D &place,  
                                const std::vector< SPtr<MbItem> > &  
                                componentItems,  
                                const c3d::path_string& fileName );
```

Результатом работы этих методов является интерфейс компонента, размещение которого в компоненте верхнего уровня описано локальной системой координат **place**, компонент наполняется элементами из контейнера **componentItems**, ему соответствует имя файла **fileName**.

Для размещения во вставке ранее созданных компонентов используются методы:

```
bool SetAssembly( const MbPlacement3D & place, const ItModelAssembly * existing ),  
bool SetDetail( const MbPlacement3D & place, const ItModelDetail * existing ).
```

Размещение компонента **existing** определяется локальной системой координат **place**.

Интерфейс ItModelInstance объявлен в файле `conv_model_properties.h`.

T.2. КОНВЕРТЕРЫ ГРАНИЧНОГО ПРЕДСТАВЛЕНИЯ

Геометрическое ядро C3D выполняет чтение файлов в форматах ACIS, IGES, STEP, X_T, X_B и построение по ним внутренней модели, а также запись внутренней модели в перечисленные форматы. Текстовые форматы ACIS, IGES, STEP, X_T и бинарный формат X_B передают информацию о геометрической форме моделируемого объекта в граничном представлении (Boundary representation). В граничном представлении для описания геометрической формы используются тела, которые представлены гранями, рёбрами и вершинами.

T.2.1. Общая характеристика функций конвертеров граничного представления

Определённые в глобальной области видимости специальные функции по сравнению с методами конвертера (описан в разделе [T.1.2. Порядок работы с конвертером](#)) имеют незначительные в подавляющем большинстве случаев ограничения: они не работают с потоками и не принимают в качестве аргумента запросчик шивки поверхностей.

Все функции имеют однотипную сигнатуру: принимают в качестве аргументов свойства конвертера `IConvortorProperty3D` (описан в разделе [T.1.3. Свойство конвертера IConvortorProperty3D](#)), модельный документ `ItModelDocument` (описан в разделе [T.1.4. Документ модели ItModelDocument](#)) и, в качестве необязательного параметра, индикатор процесса выполнения `IProgressIndicator` (описан в разделе [T.1.5. Индикатор процесса выполнения IProgressIndicator](#)). Логика работы всех функций также однотипна: все функции получают экземпляр конвертера, вызывают один из его методов и, по окончании работы, удаляют конвертер. В случае успеха функции возвращают `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

T.2.2. Общие сведения о параметрах конвертеров граничного представления

При передаче данных конвертер обращается к методам `FullFilePath`, `GetIoPermission`, `GetIoPermissions`, `LogReport` интерфейса `IConvortorProperty3D`.

При импорте конвертер обращается к методу `EnableAutoStitch` интерфейса `IConvortorProperty3D`.

При экспорте конвертер обращается к методам `GetPropertyString`, `GetOriginLocation`, `ReplaceLocationsToRight` интерфейса `IConvortorProperty3D`.

В случае использования стандартной реализации `ConvConvortorProperty3D` (описана в разделе [T.1.3. Свойство конвертера IConvortorProperty3D](#)) поле `fileName` должно содержать корректный полный путь к файлу обменного формата. Значения других полей, задаваемые по умолчанию, обеспечивают корректную работу методов. При экспорте файл будет создан либо перезаписан автоматически, если нет ограничений со стороны файловой системы.

В качестве модельного документа может быть выбрана одна из стандартных реализаций `RegularModelDocument` или `ConvModelDocument`. Использование последней предпочтительно, если нужно максимально подробно передавать данные об изделии, используя обменный формат STEP.

В качестве индикатора процесса выполнения допускается передача нулевого указателя.

T.2.3. Импорт модели в формате SAT

Функция

`MbeConvResType SATRead (IConvortorProperty3D & property,
ItModelDocument & document,
IProgressIndicator * indicator)`

выполняет импорт геометрической модели формата SAT версий вплоть до 22.0.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек импорта приведено в разделе [Т.2.2. Общие сведения о параметрах конвертеров граничного представления](#). Хотя конвертер SAT обращается к методу `IConverterProperty3D::CanShowMessages`, возвращаемое значение ни на что не влияет, поскольку отсутствует запросчик масштаба.

Геометрическая модель представляет собой либо один компонент, содержащий объекты типа `MbSolid`, либо одноуровневую сборку, компоненты которой содержат тела. Импортируются обозначения компонентов, не являющихся корневыми, визуальные свойства и элементарные атрибуты граней, рёбер, вершин.

Пример использования функции `SATRead` отсутствует. Для демонстрации импорта из SAT в тестовом приложении используется метод конвертера `IConverter3D`, в котором явно задан поток.

Т.2.4. Экспорт модели в формат SAT

Функция

`MbeConvResType SATWrite (IConverterProperty3D & property,
ItModelDocument & document,
IProgressIndicator * indicator)`

выполняет экспорт геометрической модели в формат SAT версии 2.0.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек экспорта приведено в разделе [Т.2.2. Общие сведения о параметрах конвертеров граничного представления](#).

При экспорте модель, не являющаяся однокомпонентной, преобразуется в одноуровневую сборку с сохранением формы. Многократно используемые элементы дублируются. В формат SAT экспортируются только объекты типа `MbSolid`. Экспортируются обозначения компонентов, не являющихся корневыми, визуальные свойства и элементарные атрибуты граней, рёбер, вершин.

Пример использования метода `SATWrite` приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

Т.2.5. Импорт модели в формате IGES

Функция

`MbeConvResType IGSRead (IConverterProperty3D & property,
ItModelDocument & document,
IProgressIndicator * indicator)`

выполняет импорт геометрической модели формата IGES версии 5.3.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек импорта приведено в разделе [Т.2.2. Общие сведения о параметрах конвертеров граничного представления](#). Хотя конвертер IGES обращается в методу

[IConverterProperty3D::CanShowMessages](#), возвращаемое значение ни на что не влияет, поскольку отсутствует запросчик на сшивку поверхностей.

Импортированная модель может иметь произвольную степень вложенности и содержать объекты типа [MbSolid](#), [MbWireFrame](#), [MbPointFrame](#). Импортируются сведения об авторе и организации корневого компонента, обозначения компонентов, не являющихся корневыми, визуальные свойства граней тел и рёбер проволочных каркасов.

Пример использования метода [IGSRead](#) приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

T.2.6. Экспорт модели в формат IGES

Функция

MbeConvResType [IGSWrite](#) ([IConverterProperty3D](#) & **property**,
[ItModelDocument](#) & **document**,
[IProgressIndicator](#) * *indicator*)

выполняет экспорт геометрической модели в формат IGES версии 5.3.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек экспорта приведено в разделе [T.2.2. Общие сведения о параметрах конвертеров граничного представления](#).

Экспортируемая модель может иметь произвольную степень вложенности и содержать объекты типа [MbSolid](#), [MbWireFrame](#), [MbPointFrame](#), [MbSpaceInstance](#), [MbPlaneInstance](#). Экспортируются сведения об авторе и организации корневого компонента, обозначения компонентов, не являющихся корневыми, визуальные свойства граней тел и рёбер проволочных каркасов.

Пример использования метода [IGSWrite](#) приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

T.2.7. Импорт модели форматов X_T и X_B

Функция

MbeConvResType [XTRead](#) ([IConverterProperty3D](#) & **property**,
[ItModelDocument](#) & **document**,
[IProgressIndicator](#) * *indicator*)

выполняет импорт геометрической модели форматов Parasolid (текстовый X_T и бинарный X_B) версии вплоть до 25.0.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек импорта приведены в разделе [T.2.2. Общие сведения о параметрах конвертеров граничного представления](#).

Импортированная модель может иметь произвольную степень вложенности и содержать объекты типа [MbSolid](#). Импортируются обозначения компонентов, визуальные свойства и элементарные атрибуты граней, рёбер и вершин тел.

Пример использования метода [XTRead](#) приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

T.2.8. Экспорт модели в форматы X_T, X_V

Функция

MbeConvResType **XTWrite** ([IConverterProperty3D](#) & **property**,
[ItModelDocument](#) & **document**,
[IProgressIndicator](#) * *indicator*)

выполняет экспорт геометрической модели в текстовый формат X_T или в бинарный формат X_V версии 10.0.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек экспорта приведено в разделе [T.2.2. Общие сведения о параметрах конвертеров граничного представления](#). В зависимости от значения, возвращаемого методом [IConverterProperty3D::IsFileAscii](#), конвертер формирует текстовый (*.X_T) либо двоичный (*.X_V) файл.

Экспортируемая модель может иметь произвольную степень вложенности и содержать объекты типа [MbSolid](#). Экспортируются обозначения компонентов, а также визуальные свойства граней, рёбер и вершин тел.

Пример использования метода **XTWrite** приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

T.2.9. Импорт модели формата STEP

Функция

MbeConvResType **STEPRead** ([IConverterProperty3D](#) & **property**,
[ItModelDocument](#) & **document**,
[IProgressIndicator](#) * *indicator*)

выполняет импорт геометрической модели формата STEP. Поддерживаемые прикладные протоколы — 203, 214.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек импорта приведены в разделе [T.2.2. Общие сведения о параметрах конвертеров граничного представления](#).

Импортированная модель может иметь произвольную степень вложенности и содержать объекты типа [MbSolid](#), [MbWireFrame](#), [MbPointFrame](#), [MbSpaceInstance](#). Импортируются свойства компонентов, визуальные свойства и элементарные атрибуты граней, рёбер и вершин тел, плотность материалов, а также размеры и технические требования.

Пример использования метода **STEPRead** приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

T.2.10. Экспорт модели в формат STEP

Функция

MbeConvResType **STEPWrite** ([IConverterProperty3D](#) & **property**,
[ItModelDocument](#) & **document**,
[IProgressIndicator](#) * *indicator*)

выполняет экспорт геометрической модели в формат STEP. Поддерживаемые прикладные протоколы — 203, 214.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек экспорта приведено в разделе [Т.2.2. Общие сведения о параметрах конвертеров граничного представления](#). В зависимости от значения, возвращаемого методом `IConverterProperty3D::GetFormat`, конвертер генерирует данные прикладного протокола 203 или 214. Значение, возвращаемое методом `IConverterProperty3D::GetAnnotationTextRepresentation` передаётся в качестве аргументов при вызове методов `ItModelDetail::GetAnnotationItems`, `ItModelAssembly::GetAnnotationItems`, `ItModelDocument::GetAnnotationItems`, `ItModelInstanceProperties::GetRequirements`.

Экспортируемая модель может иметь произвольную степень вложенности и содержать объекты типа `MbSolid`, `MbWireFrame`, `MbPointFrame`, `MbSpaceInstance`. Экспортируются свойства компонентов, визуальные свойства и элементарные атрибуты граней, рёбер и вершин тел, плотность материалов. Экспортируются размеры и технические требования компонентов.

Пример использования метода `STEPWrite` приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

Т.3. КОНВЕРТЕРЫ ПОЛИГОНАЛЬНОГО ПРЕДСТАВЛЕНИЯ

Геометрическое ядро C3D выполняет чтение файлов в форматах STL (текстовый и бинарный), VRML и построение по ним внутренней модели, а также запись внутренней модели в перечисленные форматы. Текстовые форматы STL, VRML и бинарный формат STL используют полигональное представление (Polygonal representation) для передачи геометрической формы моделируемого объекта. В полигональном представлении для описания геометрической формы используются треугольники и полигоны.

Т.3.1. Общая характеристика функций конвертеров полигонального представления

Определённые в глобальной области видимости специальные функции имеют единственное отличие от методов конвертера (описан в разделе [Т.1.2. Порядок работы с конвертером](#))), которое заключается в том, что они не принимают в качестве аргумента запросчик сшивки поверхностей.

Все функции имеют однотипную сигнатуру: принимают в качестве аргументов свойства конвертера `IConvertorProperty3D` (описан в разделе [Т.1.3. Свойство конвертера IConvertorProperty3D](#)), модельный документ `ItModelDocument` (описан в разделе [Т.1.4. Документ модели ItModelDocument](#)) и, в качестве необязательного параметра, индикатор процесса выполнения `IProgressIndicator` (описан в разделе [Т.1.5. Индикатор процесса выполнения IProgressIndicator](#)). Логика работы всех функций также однотипна: все функции получают экземпляр конвертера, вызывают один из его методов и, по окончании работы, удаляют конвертер. В случае успеха функции возвращают `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Т.3.2. Общие сведения о параметрах конвертеров полигонального представления

При передаче данных конвертер обращается к методам [FullFilePath](#), [GetIoPermission](#), [GetIoPermissions](#), [LogReport](#) интерфейса [IConvertorProperty3D](#).

При экспорте конвертер обращается к методам [ReplaceLocationsToRight](#), [TessellationParameters](#), [DualSeams](#) интерфейса [IConvertorProperty3D](#). Параметры расчёта триангуляции используются для построения полигонального представления всех экспортируемых объектов за исключением [MbMesh](#).

В случае использования стандартной реализации `ConvConvertorProperty3D` (описана в разделе [Т.1.3. Свойство конвертера IConvertorProperty3D](#)) поле `fileName` должно содержать корректный полный путь к файлу обменного формата. Значения других полей, задаваемые по умолчанию, обеспечивают корректную работу методов. При экспорте файл будет создан либо перезаписан автоматически, если нет ограничений со стороны файловой системы.

Если предполагается использовать одну из стандартных реализаций модельного документа, то выбор определяется тем, планируется ли использовать интерфейсные методы или модель, сформированную с использованием объектов геометрического ядра C3D. В первом случае следует пользоваться реализацией [RegularModelDocument](#), во втором – [ConvModelDocument](#).

В качестве индикатора процесса выполнения допускается передача нулевого указателя.

Т.3.3. Импорт модели формата STL

Функция
`MbeConvResType` [STLRead](#) ([IConvertorProperty3D](#) & `property`,
[ItModelDocument](#) & `document`,
[IProgressIndicator](#) * `indicator`)

выполняет импорт геометрической модели формата STL текстовой или бинарной формы.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек импорта приведено в разделе [Т.3.2. Общие сведения о параметрах конвертеров полигонального представления](#). Импортированная модель однокомпонентная, компонент содержит один объект типа `MbMesh`. Формат STL не предоставляет средств для передачи полигонов.

Пример использования метода **STLRead** приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

Т.3.4. Экспорт модели в формате STL

Функция

`MbeConvResType` **STLWrite** (`IConverterProperty3D` & **property**,
`ItModelDocument` & **document**,
`IProgressIndicator` * *indicator*)

выполняет экспорт геометрической модели в формат STL текстовой или бинарной формы.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек экспорта приведено в разделе [Т.3.2. Общие сведения о параметрах конвертеров полигонального представления](#). В зависимости от значения, возвращаемого методом `IConverterProperty3D::IsFileAscii`, конвертер формирует файл формата STL текстовой или бинарной формы. Кроме того, конвертер формата STL обращается к методу `IConverterProperty3D::GetOriginLocation`.

При экспорте сохраняется форма модели (с точностью до преобразования к полигональному представлению), но полностью теряются полигоны и информация о её структуре. Многократно используемые элементы дублируются.

Пример использования метода **STLWrite** приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

Т.3.5. Импорт модели формата VRML

Функция

`MbeConvResType` **VRMLRead** (`IConverterProperty3D` & **property**,
`ItModelDocument` & **document**,
`IProgressIndicator` * *indicator*)

выполняет импорт геометрической модели формата VRML версии 2.0.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек импорта приведено в разделе [Т.3.2. Общие сведения о параметрах конвертеров полигонального представления](#).

Импортированная модель может иметь произвольную степень вложенности и содержать объекты типа `MbMesh`. Импортируются визуальные свойства сеток.

Пример использования метода **VRMLRead** приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.

Т.3.6. Экспорт модели в формат VRML

Функция

MbeConvResType **VRMLWrite** ([IConverterProperty3D](#) & **property**,
[ItModelDocument](#) & **document**,
[IProgressIndicator](#) * *indicator*)

выполняет экспорт геометрической модели в формат VRML версии 2.0.

Входными и выходными параметрами метода являются:

- **property** – свойства конвертера,
- **document** – документ, передающий геометрическую модель и другую информацию,
- *indicator* – индикатор хода процесса чтения или записи.

При удачной работе конвертера функция возвращает `cnv_Success`, в противном случае метод возвращает код ошибки из перечисления `MbeConvResType`.

Описание общих настроек экспорта приведено в разделе [Т.3.2. Общие сведения о параметрах конвертеров полигонального представления](#).

При экспорте сохраняется форма модели (с точностью до преобразования к полигональному представлению), информация о её структуре, визуальные свойства сеток.

Пример использования метода **VRMLWrite** приведен в файле `test_converter.cpp` демонстрационного приложения геометрического ядра C3D.