ГЕОМЕТРИЧЕСКОЕ ЯДРО C3D

Содержание

ВВЕДЕНИЕ

1. Назначение

Геометрическое ядро C3D предназначено для использования в системах автоматизированного проектирования в качестве программного компонента.

2. Общие сведения

Геометрическое ядро C3D представляет собой 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.

3. Выполняемые функции

Геометрическое ядро C3D содержит следующие составляющие геометрической модели: описание формы моделируемого объекта, описание связей элементов геометрической модели, историю построения модели, атрибуты элементов геометрической модели.

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

Геометрическое ядро C3D 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 выполняет построение плоских проекций геометрической модели (Mapping), вычисление инерционных характеристик геометрической модели (Inertia Properties), определение столкновений элементов модели (Collision Detection).

Связи элементов модели обеспечивают геометрические ограничения (Geometric Constraints), которые работают с трехмерными и двумерными объектами геометрической модели. Геометрические ограничения представляют собой условия, наложенные на элементы модели, выраженные с помощью уравнений и неравенств. Геометрические ограничения позволяют редактировать модель, создавать сборочные единицы, создавать подобные модели, моделировать механизмы.

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

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

Для обмена данными с другими системами геометрическое ядро C3D выполняет чтение и запись геометрической модели в форматах STEP, IGES, ACIS, XT, STL, VRML.

4. Структура

Геометрическое ядро C3D состоит из трёх модулей, приведённых на рис. 4.1.

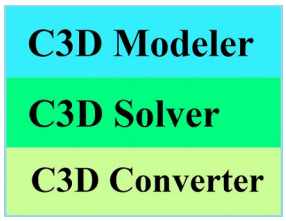


Рис 4.1.

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

Модуль C3D Solver обеспечивает взаимосвязь элементов геометрической модели, что позволяет путем пересчета вариационных связей редактировать модели, строить подобные модели, моделировать механизмы.

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

5. Комплектация

В комплект геометрического ядра C3D входят: библиотечные файлы c3d.lib, c3d.dll, libmath.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 VisualStudio2005, VisualStudio2008, VisualStudio2010, VisualStudio2012, VisualStudio2013. В операционной системе Linux библиотечные файлы собраны компилятором GCC в конфигурациях 64bit, Unicode, Debug/Release. В операционной системе Mac Os библиотечные файлы собраны компилятором Clank в конфигурациях 64bit, Unicode, Debug/Release.

По заголовочным файлам Include/*.h сгенерирована документация геометрического ядра C3D. Заголовочные файлы содержат описание объектов и методов геометрического ядра C3D на русском и английском языках. Описание объектов и методов геометрического ядра C3D также дано в руководстве пользователя. В файле Changes.txt приведены изменения интерфейса геометрического ядра.

В комплект поставки входит обертка геометрического ядра C3D, позволяющая использовать технологию .NET при разработке приложений.

Вместе с геометрическим ядром C3D поставляется приложение test.exe для системы Windows, демонстрирующее возможности геометрического ядра C3D, исходные тексты этого приложения, файл CMakeLists.txt для генерации проекта приложения и набор файлов с моделями.

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

6. Тестовое приложение

Готовое тестовое приложение геометрического ядра C3D для системы Windows располагается в каталоге Example/Demo.

Файлы Test_VS2005.sln и Test_VS2005.vcproj содержат соответственно решение и проект тестового приложения геометрического ядра C3D для Microsoft VisualStudio 2005.

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.

Для создания проекта и компиляции тестового приложения геометрического ядра C3D выполните следующие действия.

1. Создайте тестовый каталог в удобном для работы месте, например, TestApp.
2. Скопируйте каталог«Include» вместе с содержимым в тестовый каталог TestApp.
3. В каталоге «C3D» выберите версию и конфигурацию геометрического ядра и скопируйте каталог с файлами:c3d.dll, c3d.lib (libmath.so для системы Linux) в тестовый каталог TestApp. Переименуйте каталог с файлами c3d.dll и c3d.lib в «Debug» или «Release», соответственно, оставив в исходном названии каталога только последнее слово.
4. Скопируйте каталог «Sourse» с содержимым в тестовый каталог TestApp.
5. Убедитесь, что в тестовом каталоге TestApp находятся папки: Include, Source и Release (или/и Debug).
6. Установите CMake, выбрав в процессе установки опцию «Add CMake to the system PATH for all users».
7. Создайте проект тестового приложения, выполнив следующие действия.
   Запустите CMake, который сгенерирует проект по файлу CMakeLists.txt.
   Для «Where is the source code» укажите каталог <path_to_testapp>\TestApp\Source.
   Для «Where to build the binaries» укажите каталог <path_to_testapp>\TestApp\Build.
   Нажмите кнопку **Configure** для конфигурирования проекта.
   На запрос «Create Directory» подтвердите согласие на создание каталога <path_to_testapp>\TestApp\Build.
   По запросу «Specify the generator for this project» укажите соответствующую версии C3D конфигурацию среды разработки.
   Нажмите кнопку **Generate** для генерации файлов проекта.
8. Запустите созданный проект TestApp\Build\Test.sln тестового приложения в среде разработки.
9. Для активации геометрического ядра C3D перед компиляцией измените вызов метода EnableMathModules(...) в конструкторе объекта «Manager» в файле test_manager.cpp с реальными ключом и сигнатурой.
10. После компиляции запустите созданное тестовое приложение test.exe из каталога TestApp\Debug или TestApp\Release, соответственно.

Указанные выше действия описаны в файле readme.txt.

7. Разработка в среде .NET

Геометрическое ядро C3D может работать в среде .NET. Для разработки приложений в среде .NET следует использовать обертку, входящую в комплект геометрического ядра C3D.

Обертка геометрического ядра 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 геометрического ядра C3D в приложениях, разрабатываемых на C#, нужно выполнить следующие действия:

1. Выбрать из комплекта геометрического ядра 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 name = Environment.GetEnvironmentVariable("c3dname");
   NetC3D.ToolEnabler.EnableMathModules(name, key);
   ```
   где c3dkey и c3dname – переменные среды, содержащие ключ и сигнатуру.

8. Теоретические основы

Граничное представление, используемое геометрическим ядром C3D, даёт точное описание геометрической формы моделируемого объекта. Для описания геометрической формы ядро 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 геометрического ядра C3D.

Геометрические ограничения, описывающие связи элементов модели и прочие условия, формулируются в виде уравнений и неравенств. Для поиска решения, удовлетворяющего уравнениям и неравенствам геометрических ограничений, геометрическое ядро C3D 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 выполняется по принципу Делоне в плоскости параметров поверхностей.

Для кривых и поверхностей геометрическое ядро C3D 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 геометрического ядра C3D и объектами обменных форматов.

Применяемые в геометрическом ядре C3D математические объекты, методы и алгоритмы приведены в книгах:

1. Голованов Н.Н. Геометрическое моделирование. – М: «Физматлит», 2002.
2. Компьютерная геометрия / Голованов Н.Н., Ильютко Д.П., Носовский Г.В., Фоменко А.Т.. – М: «Академия», 2006.
3. Голованов Н.Н. Геометрическое моделирование. – М: «Академия», 2011.
4. Geometric Modeling, Nikolay Golovanov. – A: 2015.

М.1. МЕТОДЫ ПОСТРОЕНИЯ ТЕЛ

Геометрическое ядро С3D строит тела, описывающие всю поверхность или часть поверхности моделируемого объекта. 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. Незамкнутому телу принадлежит только множество точек описывающих его граней. Моделирование начинается с построения тел простой формы. Тела простой формы 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 – 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_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.

solidType	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.

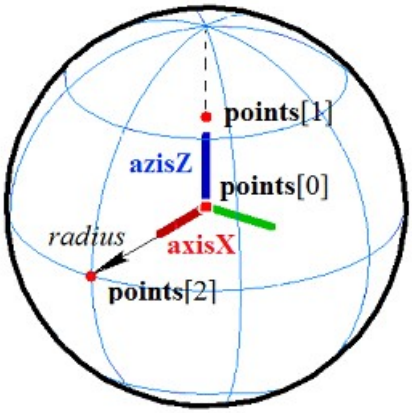


Рис. М.1.1.1.

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.

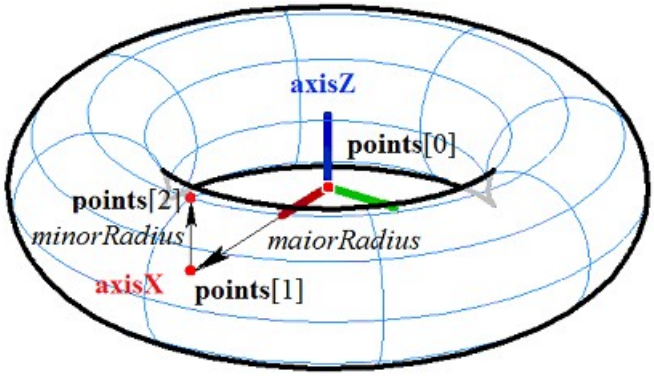


Рис. М.1.1.2.

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.

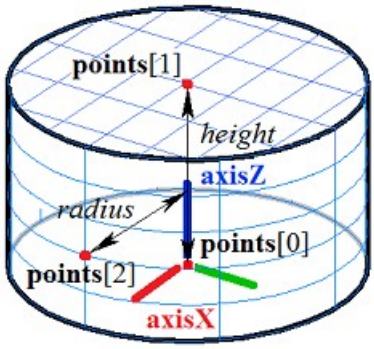


Рис. М.1.1.3.

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.

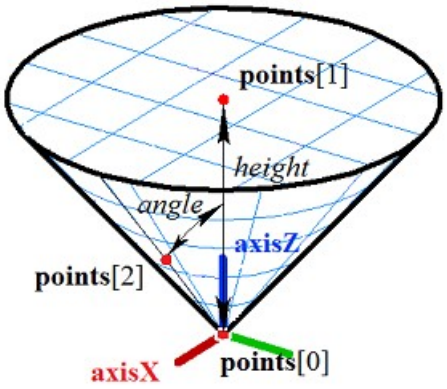


Рис. М.1.1.4.

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.

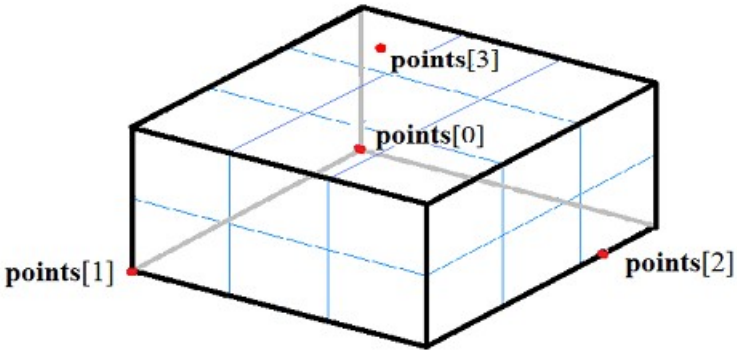


Рис. М.1.1.5.

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.

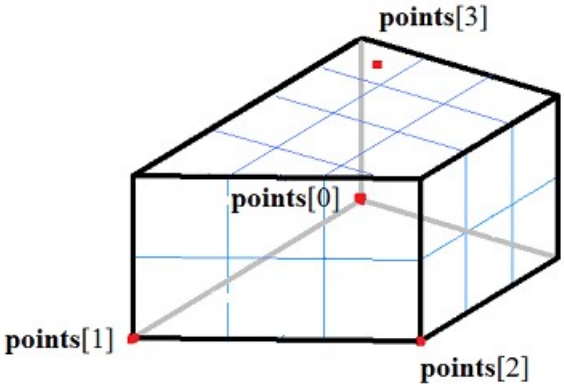


Рис. М.1.1.6.

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.

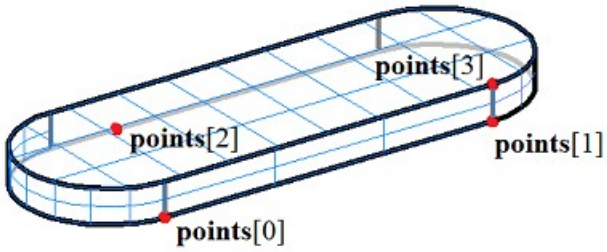


Рис. М.1.1.7.

При построении прямой призмы с многоугольником в основании точки **points**[0], **points**[1] и **points**[2] определяют плоскость нижнего основания призмы, точки **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.

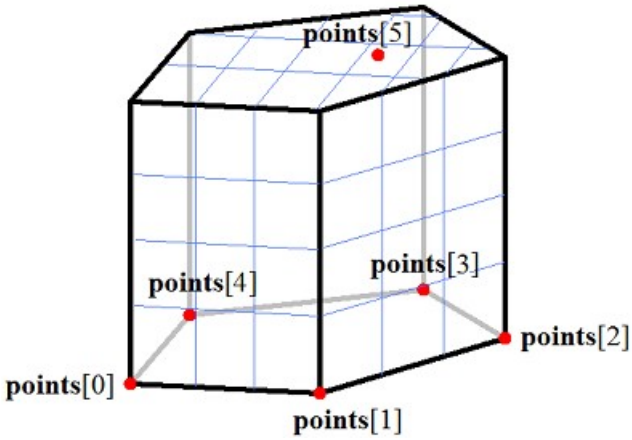


Рис. М.1.1.8.

При построении пирамиды с многоугольником в основании точки **points**[0], **points**[1] и **points**[2] определяют плоскость нижнего основания пирамиды, точки **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.

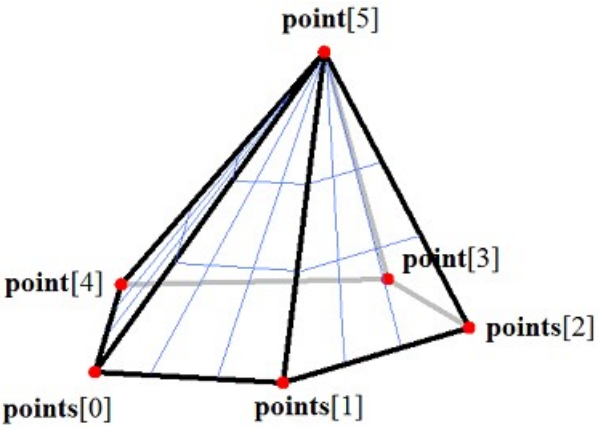


Рис. М.1.1.9.

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 cr_elementary_solid.h.

Тестовое приложение test.exe выполняет построение элементарного тела по заданным точкам командой 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 – 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_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.

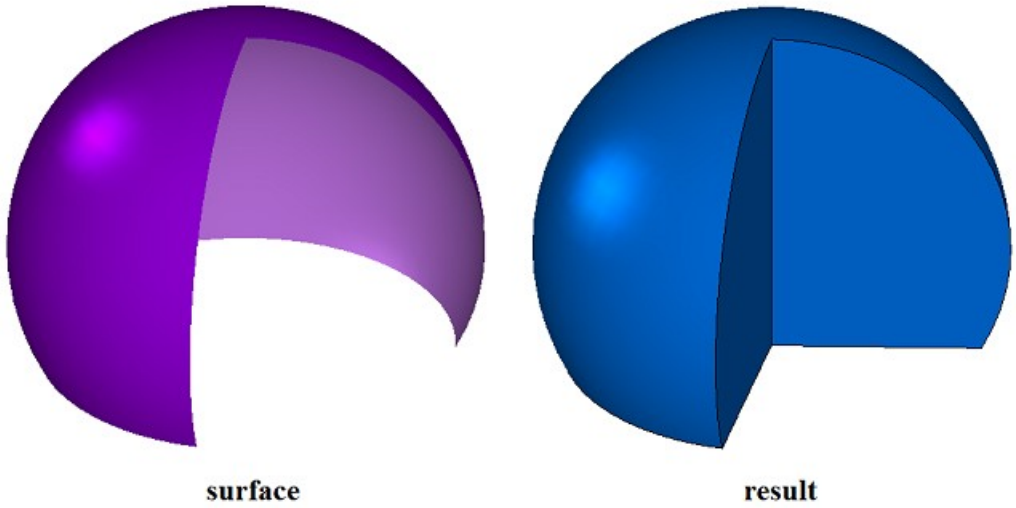


Рис. М.1.2.1.

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

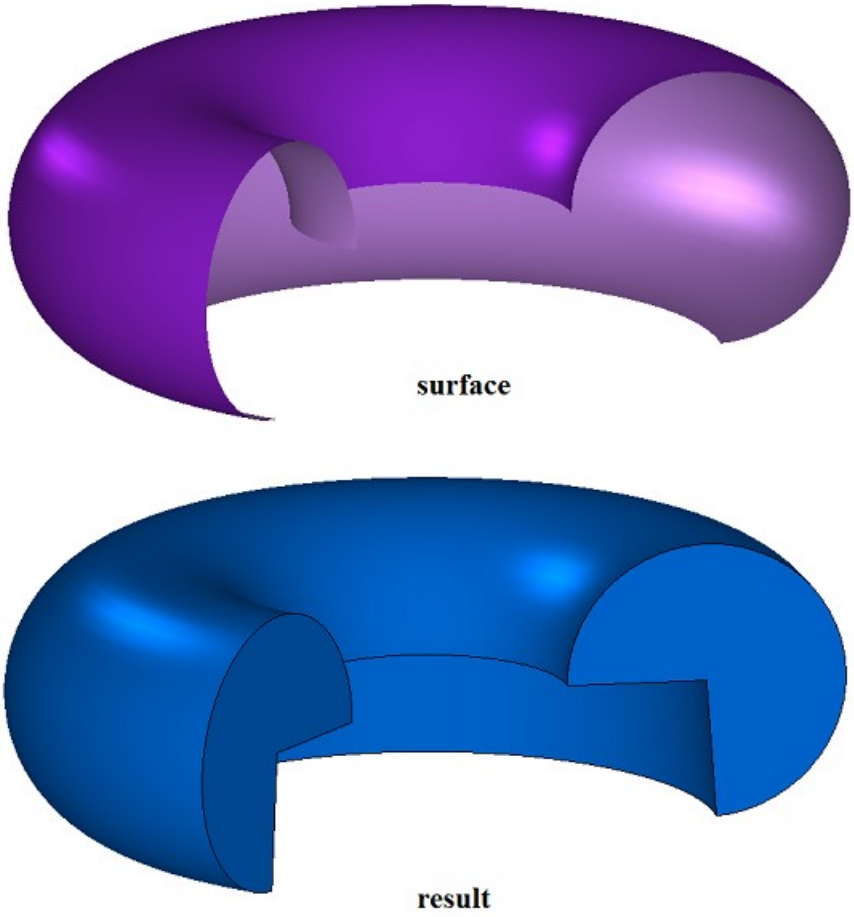


Рис. М.1.2.2.

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

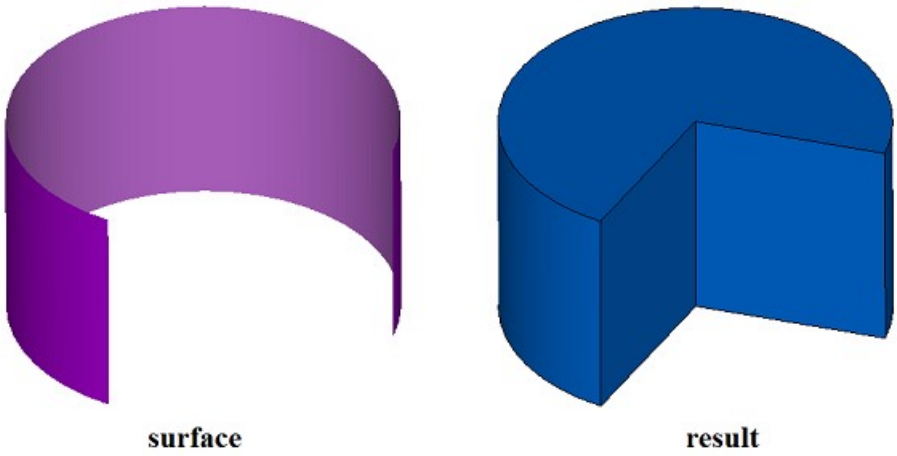


Рис. М.1.2.3.

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

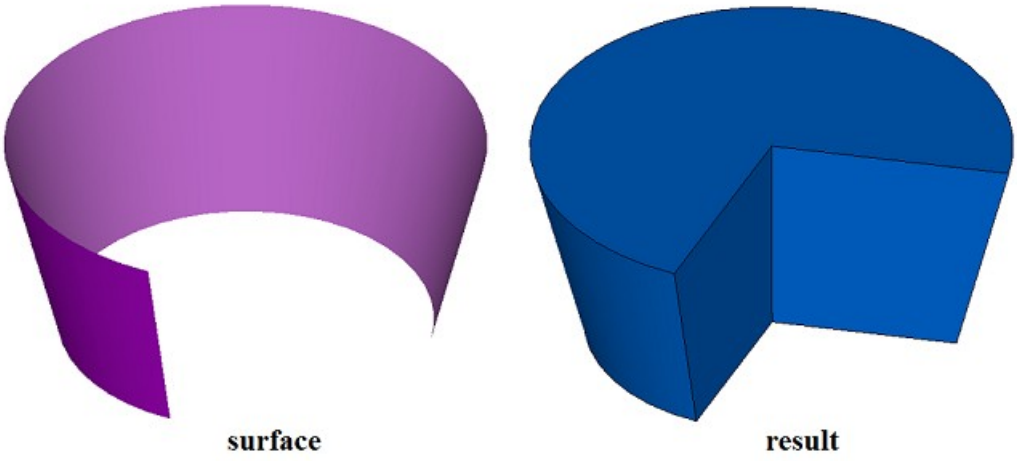


Рис. М.1.2.4.

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 cr_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 * **solid**1,
const MbSolid * **solid**2,
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 – именователь 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 описан в файле shell_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 *side*1 и информацию о способе выдавливания в обратном направлении MbSweptSide *side*2. Выдавливание в каждую сторону может выполнятся тремя способами. При *way*==sw_scalarValue выдавливание выполняется на длину *scalarValue* в направлении *side*1 или *side*2, соответственно. При *way*==sw_shell выдавливание выполняется до ближайшего объекта **solid**1 или **solid**2, соответственно. При *way*==sw_surface выдавливание выполняется до поверхности *side*1–>**surface** или *side*2–>**surface**, соответственно, при *side*1.*distance*=0 или *side*2.*distance*=0. При *way*==sw_surface и *distance*!=0 выдавливание выполняется до эквидистантной поверхности к *side*1–>**surface** или до эквидистантной поверхности к *side*2–>**surface**, соответственно. Если *side*1.*rake*!=0 или *side*2.*rake*!=0, то выдавливание выполняется с уклоном *side*1.*rake* или *side*2.*rake* в соответствующую сторону. Параметр *params.thickness*1 задает отступ наружу от образующей кривой, а параметр *params.thickness*2 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*.

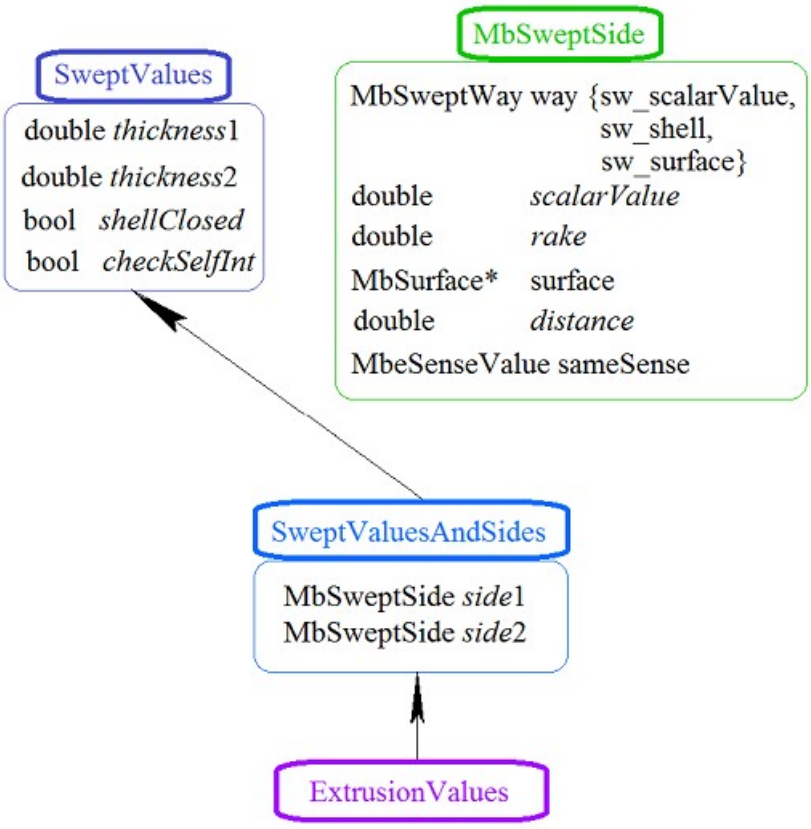


Рис. М.1.3.1.

Параметры names и cnames обеспечивают именование граней построенного тела.

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

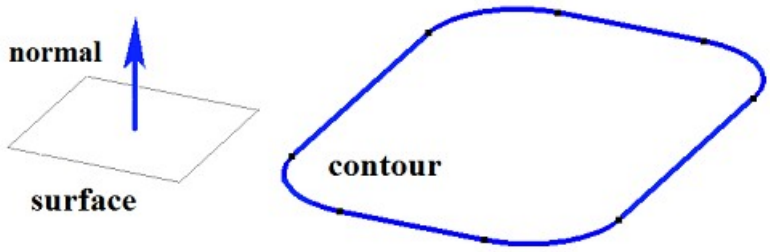


Рис. М.1.3.2.

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 cnames[0].

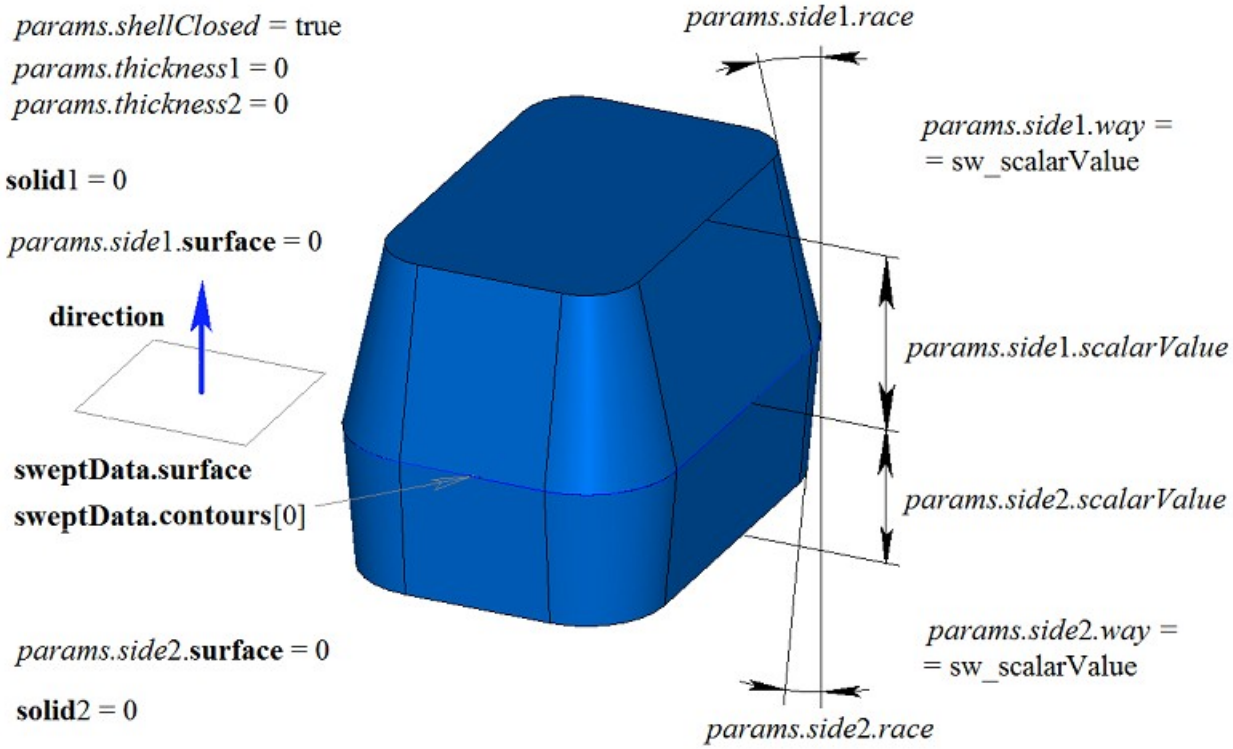


Рис. М.1.3.3.

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

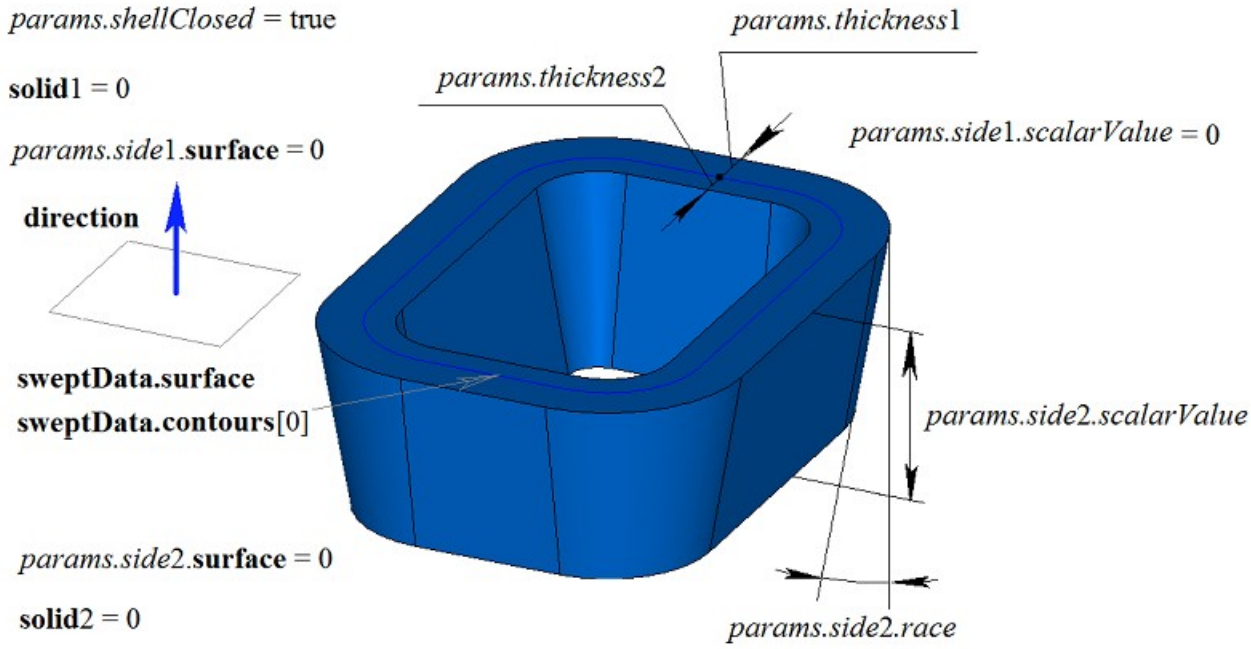


Рис. М.1.3.4.

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*.

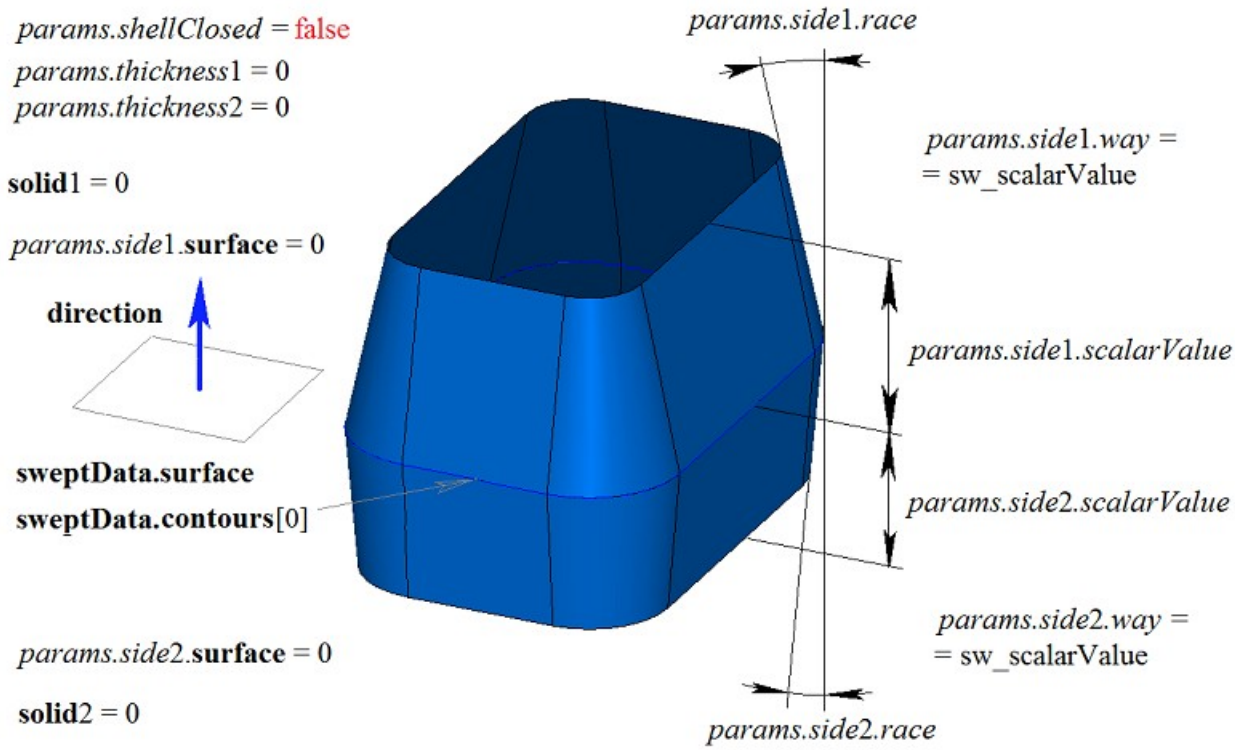


Рис. М.1.3.5.

На рис. М.1.3.6 приведен двумерный контур **contour**, плоская поверхность **surface** (MbPlane) и два тела **solid**1 и **solid**2, которые будут использоваться при построении тела выдавливания. Для построения необходимо, чтобы тела **solid**1 и **solid**2 полностью перекрывали путь движения контура в соответствующем направлении. При этом следует учитывать параметры *params.side*1.*rake*, *params.side*2.*rake*, *params.thickness*1, *params.thickness*2.

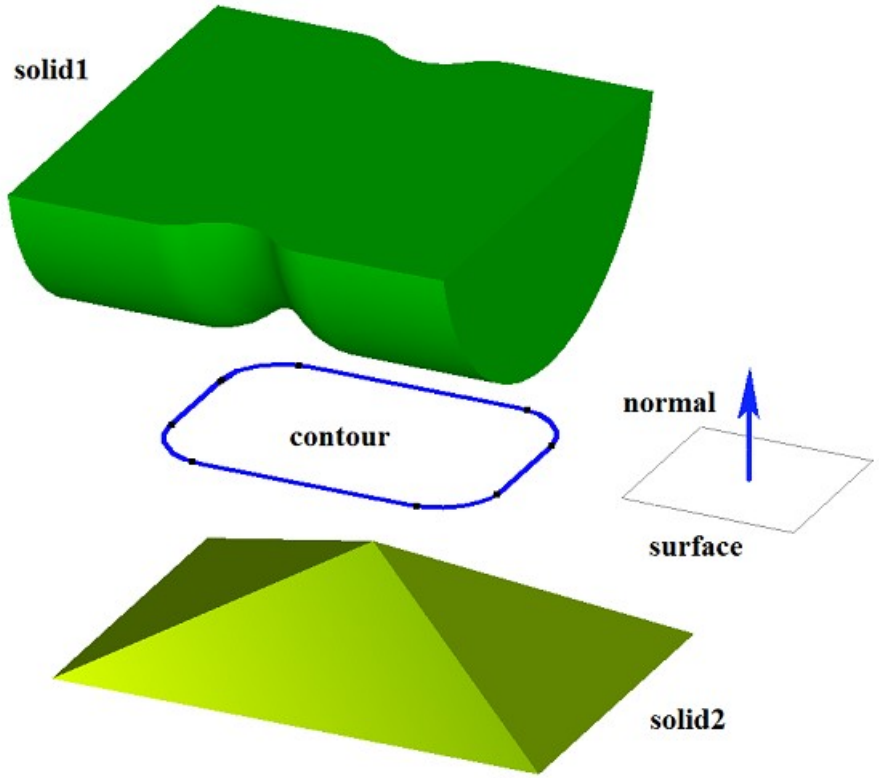


Рис. М.1.3.6.

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 **solid**1 и **solid**2.

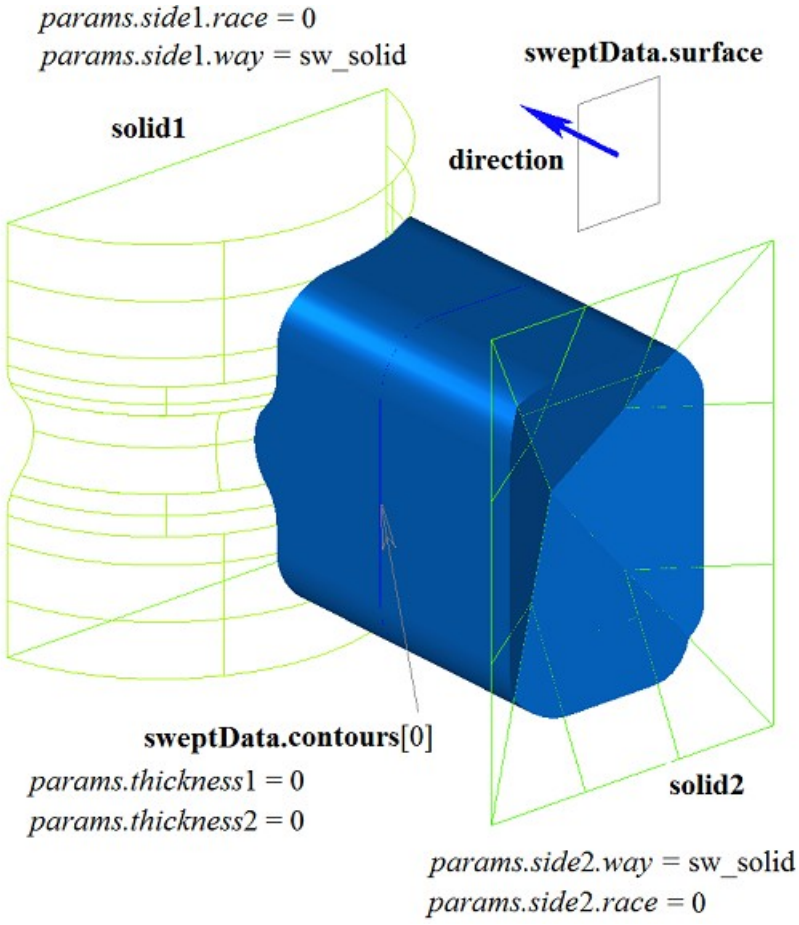


Рис. М.1.3.7.

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 **solid**1 и **solid**2.

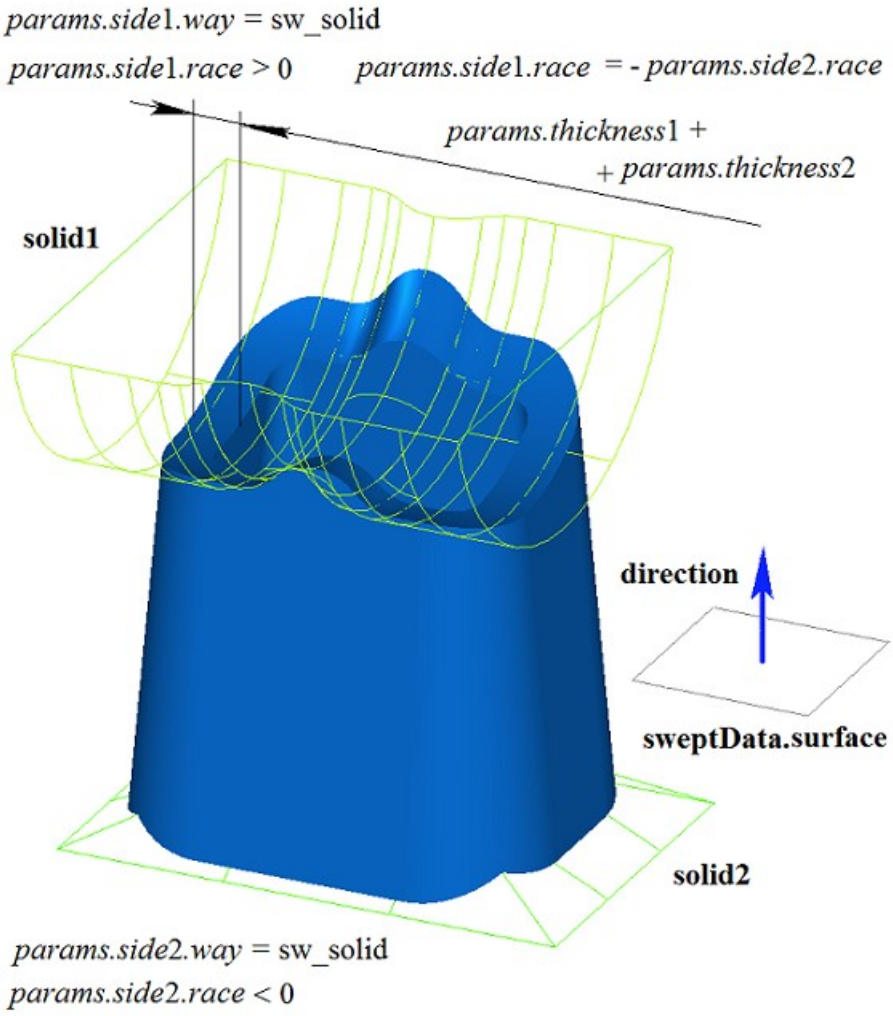


Рис. М.1.3.8.

На рис. М.1.3.9 приведен двумерный контур **contour**, плоская поверхность **surface** (MbPlane) и две поверхности **surface**1 и **surface**2, которые будут использоваться при построении тела выдавливания. Для построения необходимо, чтобы поверхности **surface**1 и **surface**2 полностью перекрывали путь движения контура в соответствующем направлении. При этом следует учитывать параметры *params.side*1.*rake*, *params.side*2.*rake*, *params.thickness*1, *params.thickness*2. Тело выдавливания обрезается заданными поверхностями или эквидистантными к ним поверхностями, если *params.side*1.*distance* или *params.side*2.*distance* не равны нулю.

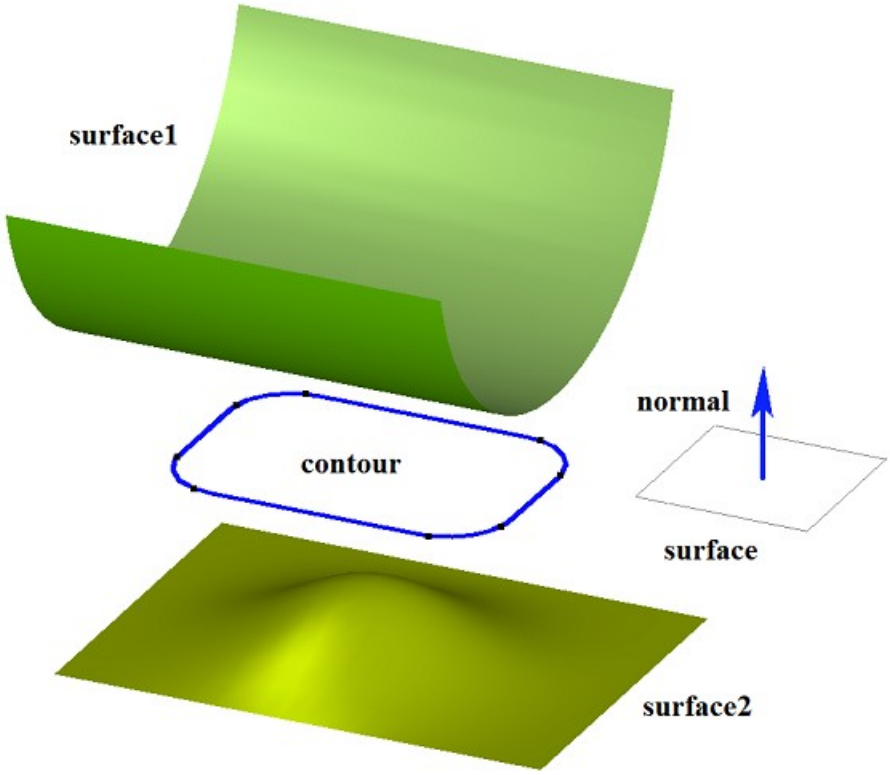


Рис. М.1.3.9.

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 **surface**1 и **surface**2.

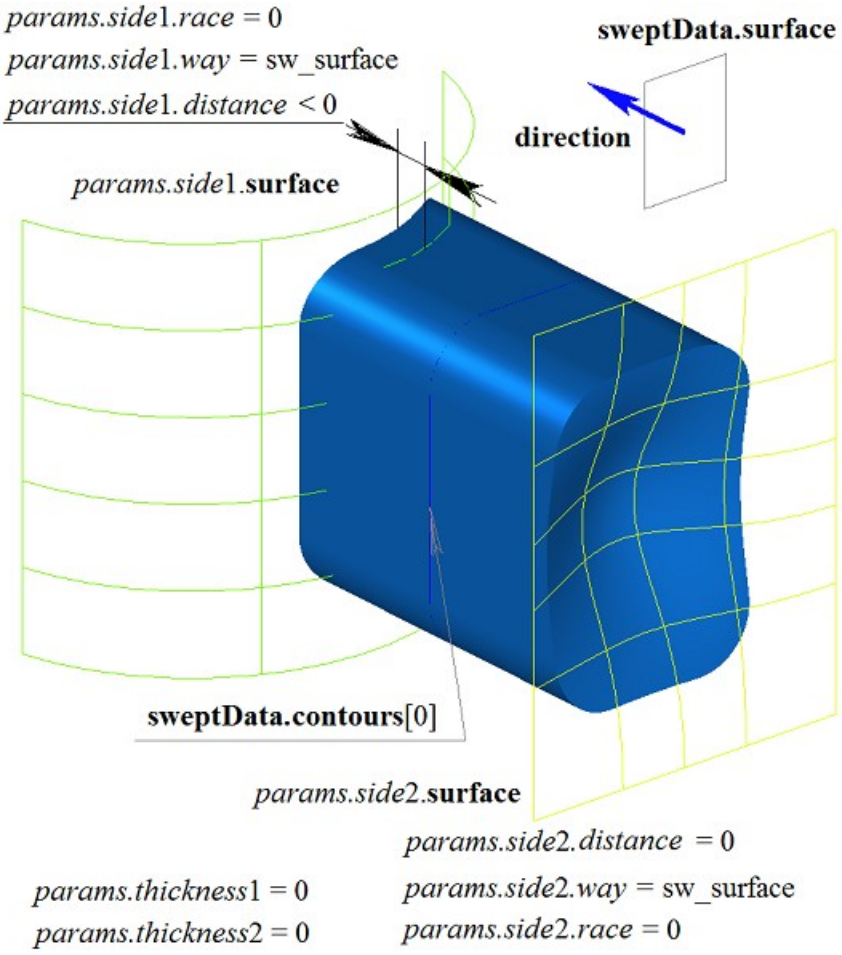


Рис. М.1.3.10.

На рис. М.1.3.11 приведено тонкостенное тело с уклоном граней, построенное выдавливанием контура, показанного на рис. М.1.3.9, с опциями «До поверхности» , в качестве которых заданы **surface**1 и **surface**2.

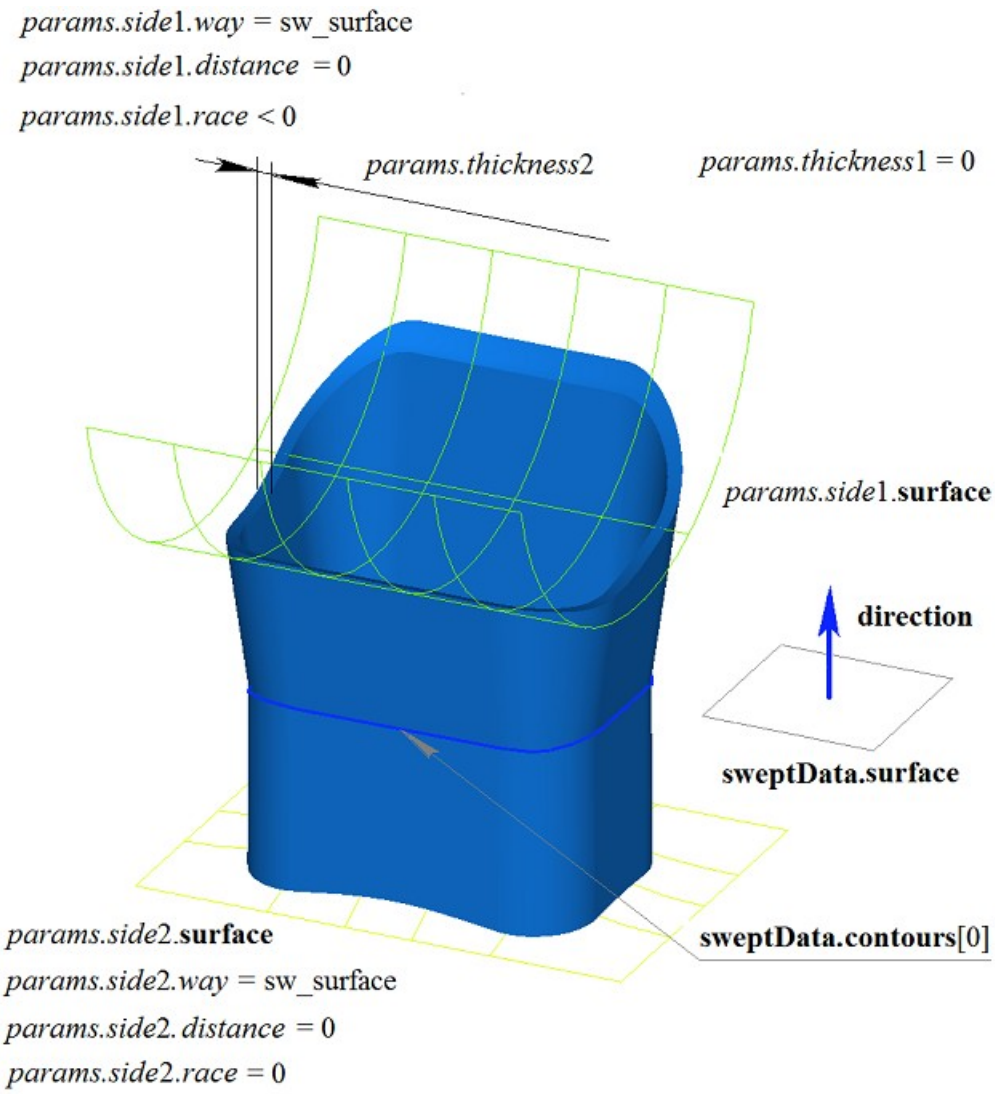


Рис. М.1.3.11.

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.

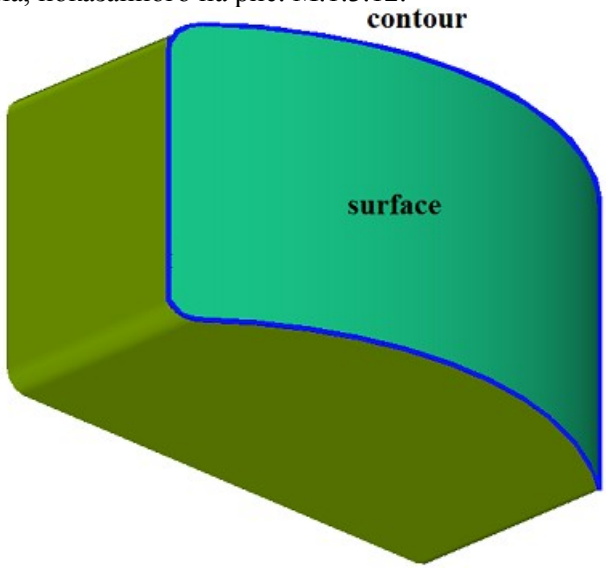


Рис. М.1.3.12.

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

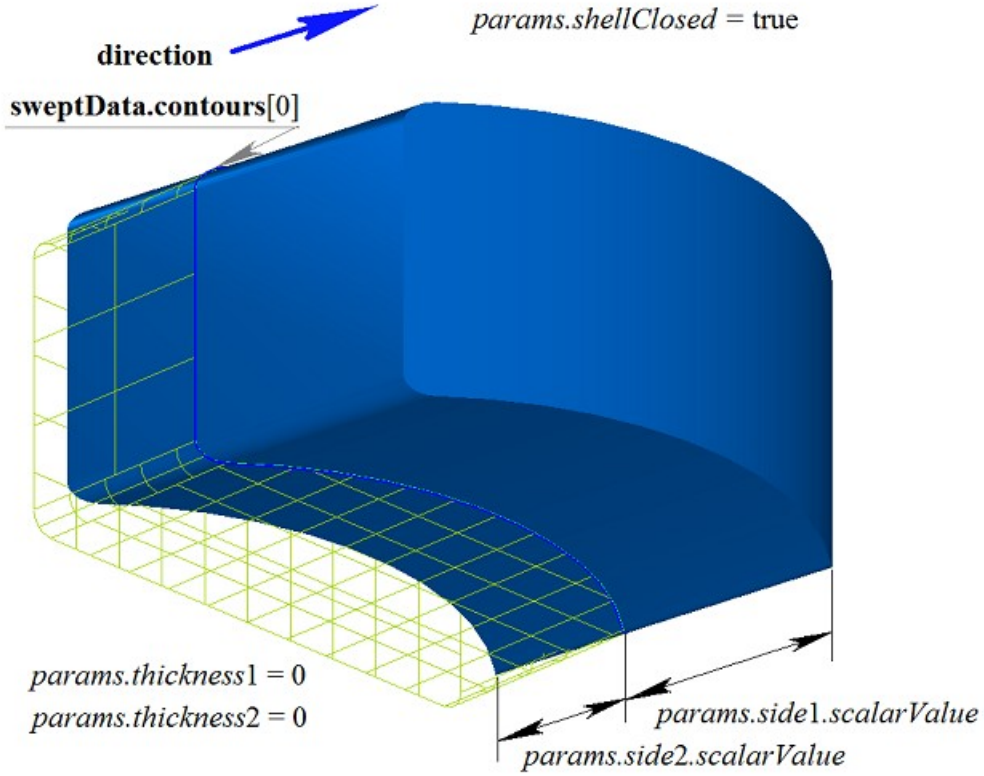


Рис. М.1.3.13.

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

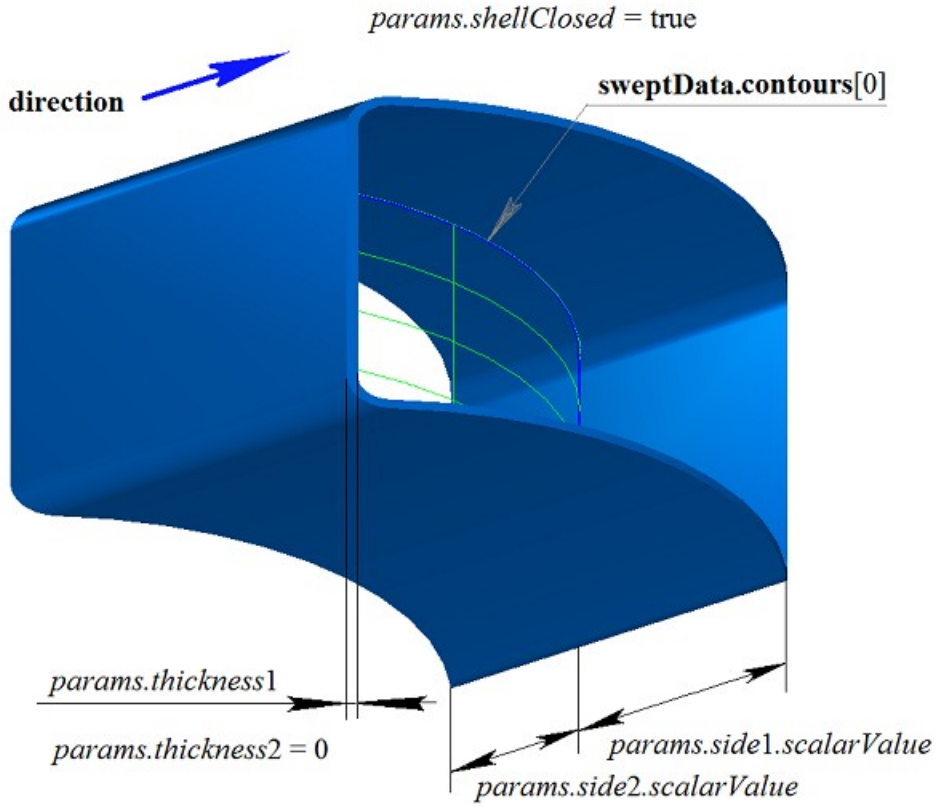


Рис. М.1.3.14.

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

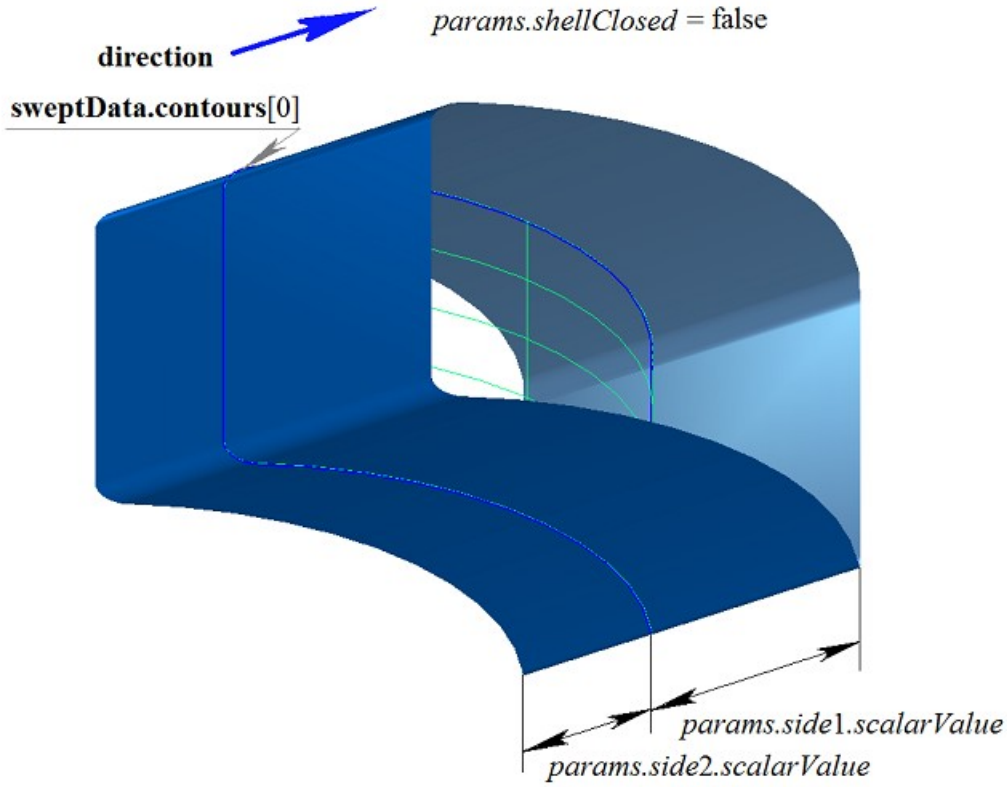


Рис. М.1.3.15.

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).

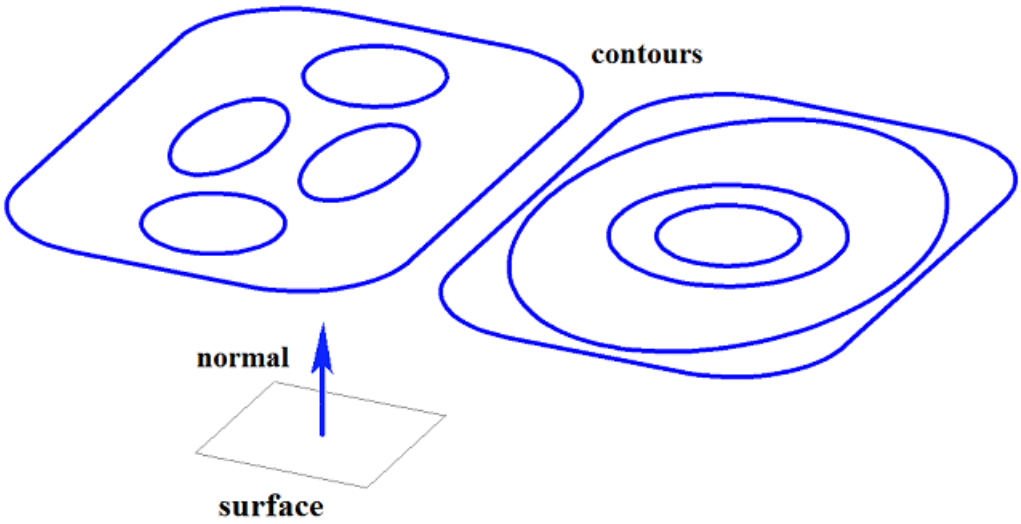


Рис. М.1.3.16.

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

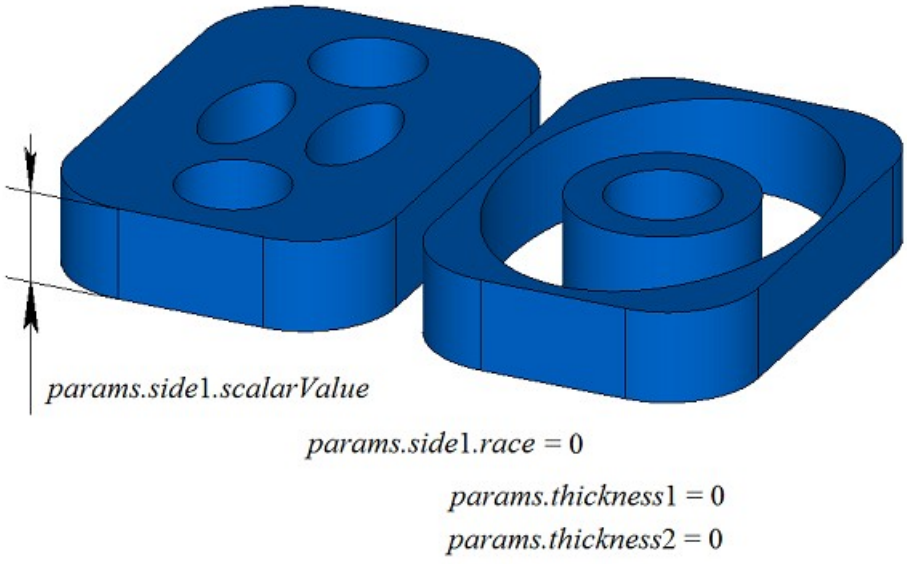


Рис. М.1.3.17.

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.

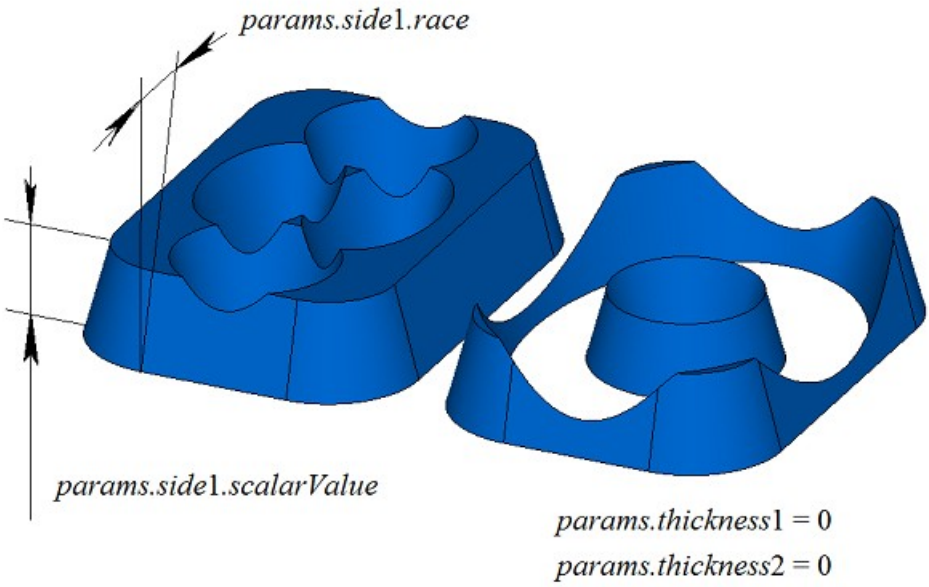


Рис. М.1.3.18.

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.

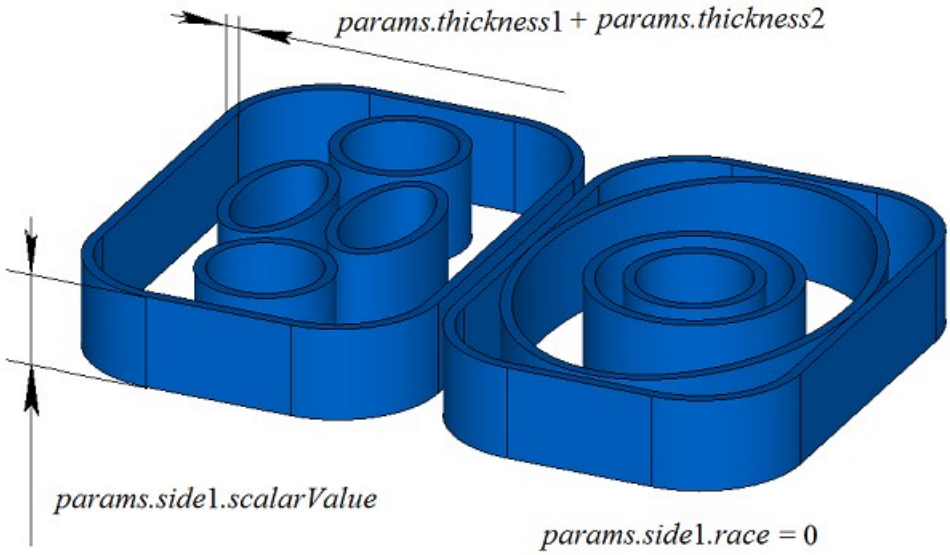


Рис. М.1.3.19.

На рис. М.1.3.20 приведены два трехмерных контура.

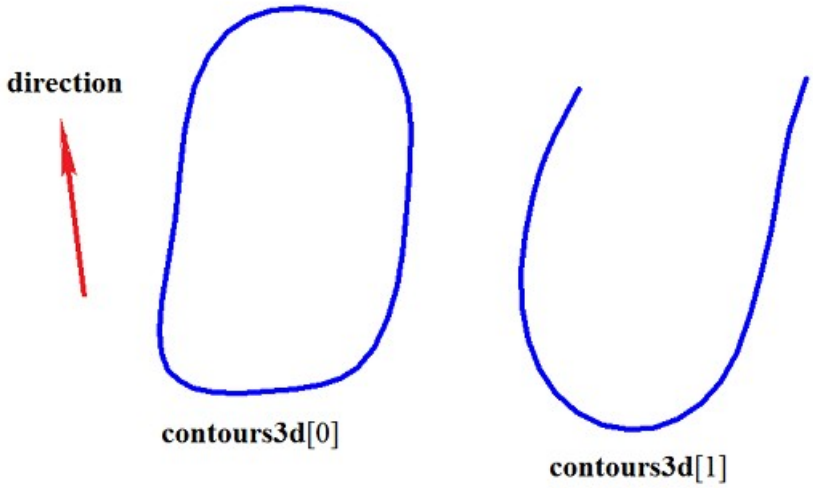


Рис. М.1.3.20.

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

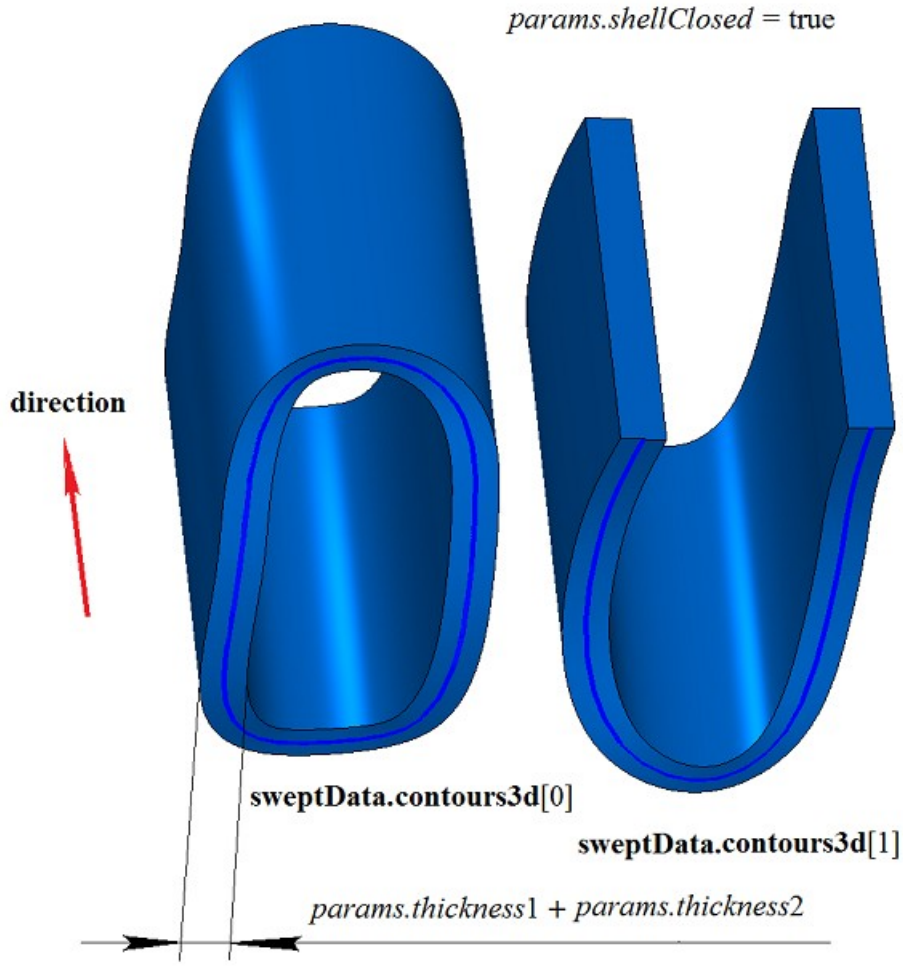


Рис. М.1.3.21.

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.

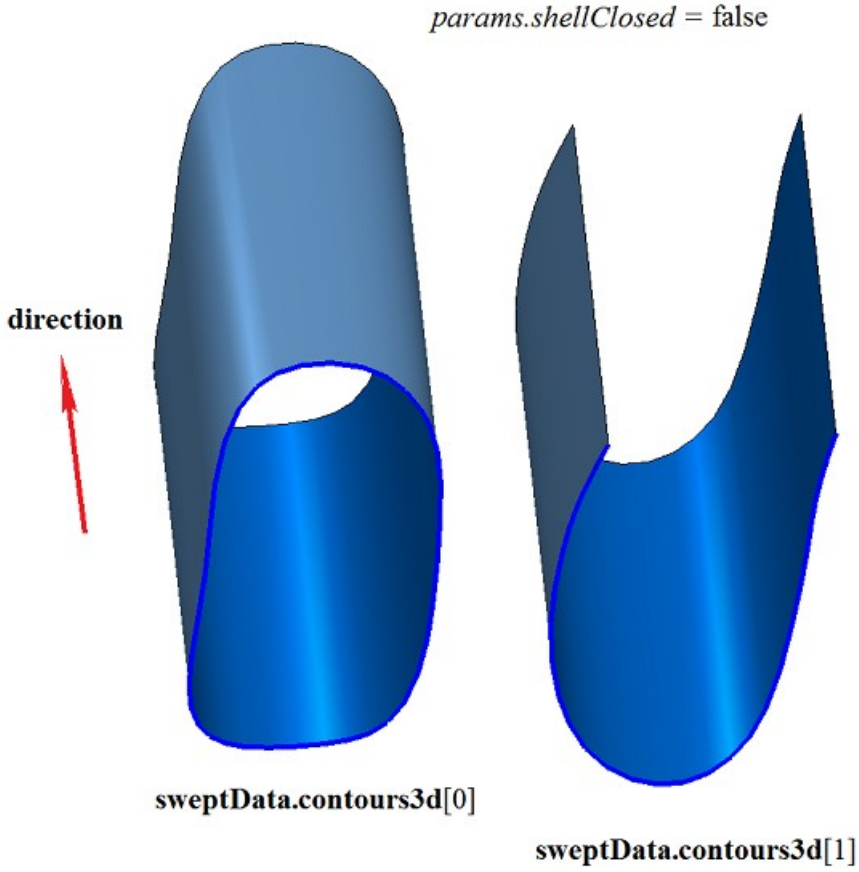


Рис. М.1.3.22.

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 cr_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,
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 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 описан в файле shell_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 выполнять в прямом направлении относительно оси **axis**, в обратном направлении относительно оси **axis** и в обоих направлениях. Параметры построения в каждом направлении задают объекты MbSweptSide.

Параметр *params* содержит информацию о способе вращения в прямом направлении MbSweptSide *side*1 и информацию о способе вращения в обратном направлении MbSweptSide *side*2. Вращение в каждую сторону может выполняться двумя способами. При *way*==sw_scalarValue вращение выполняется на угол *scalarValue* в направлении *side*1 или *side*2, соответственно. При *way*==sw_surface вращение выполняется до поверхности *side*1–>**surface** или *side*2–>**surface**, соответственно, при *side*1.*distance*=0 или *side*2.*distance*=0. При *way*==sw_surface и *distance*!=0 вращение выполняется до эквидистантной поверхности к *side*1–>**surface** или до эквидистантной поверхности к *side*2–>**surface**, соответственно. Параметры *params.thickness*1 и *params.thickness*2 определяют толщину стенки тонкостенного тела. Параметр *params.thickness*1 задает отступ наружу от образующей кривой, а параметр *params.thickness*2 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*.

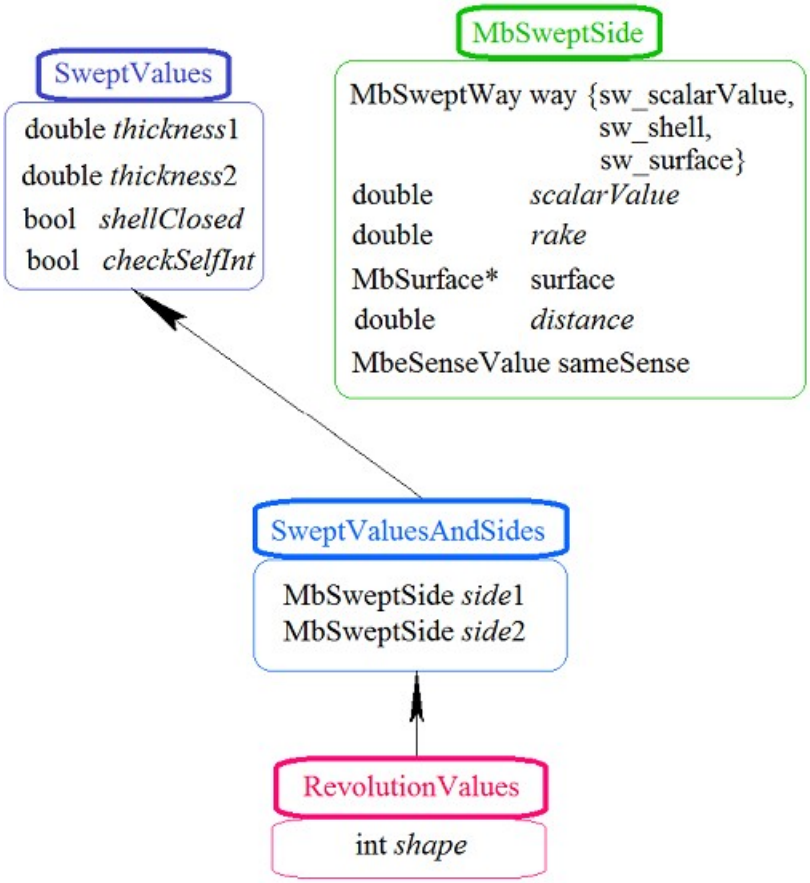


Рис. М.1.4.1.

Параметры names и cnames обеспечивают именование граней построенного тела.

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

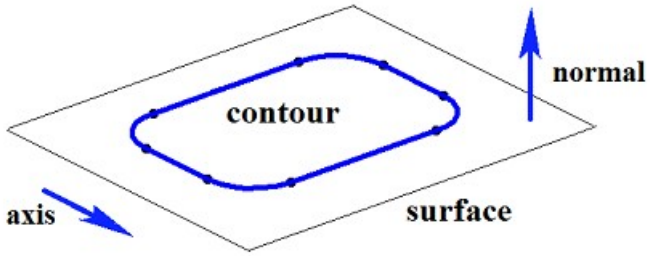


Рис. М.1.4.2.

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 cnames[0].

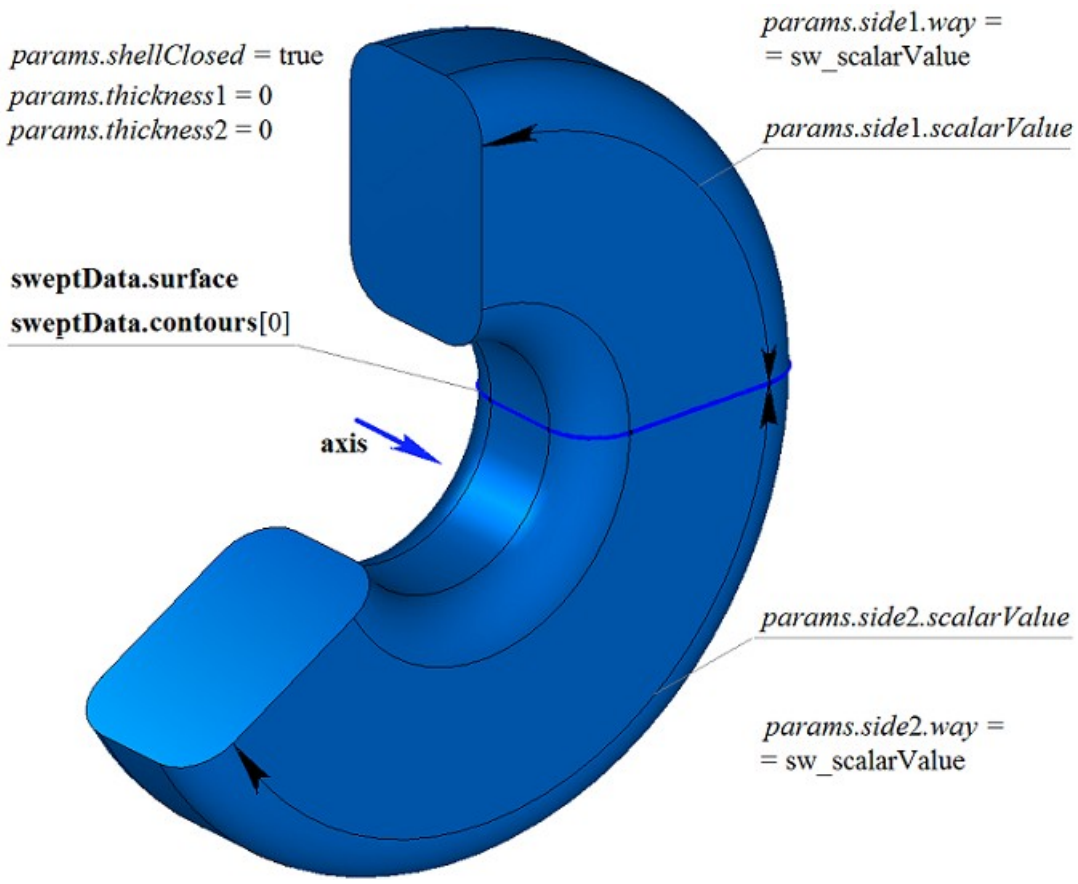


Рис. М.1.4.3.

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

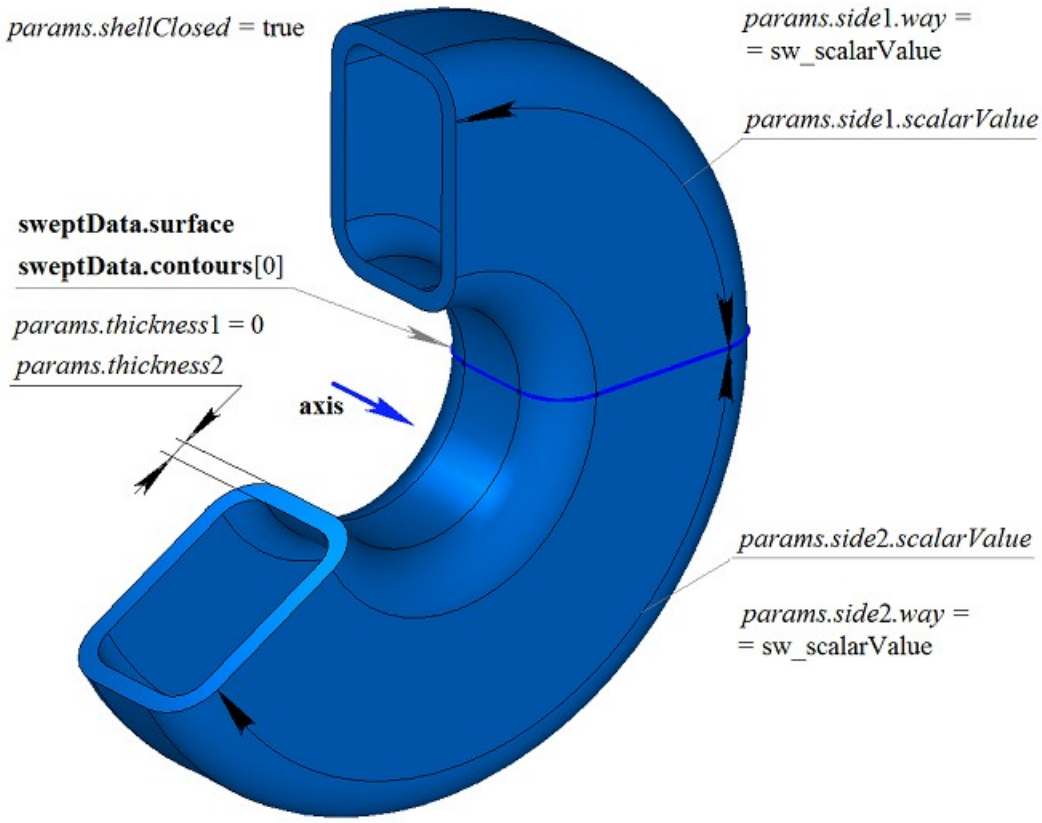


Рис. М.1.4.4.

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*.

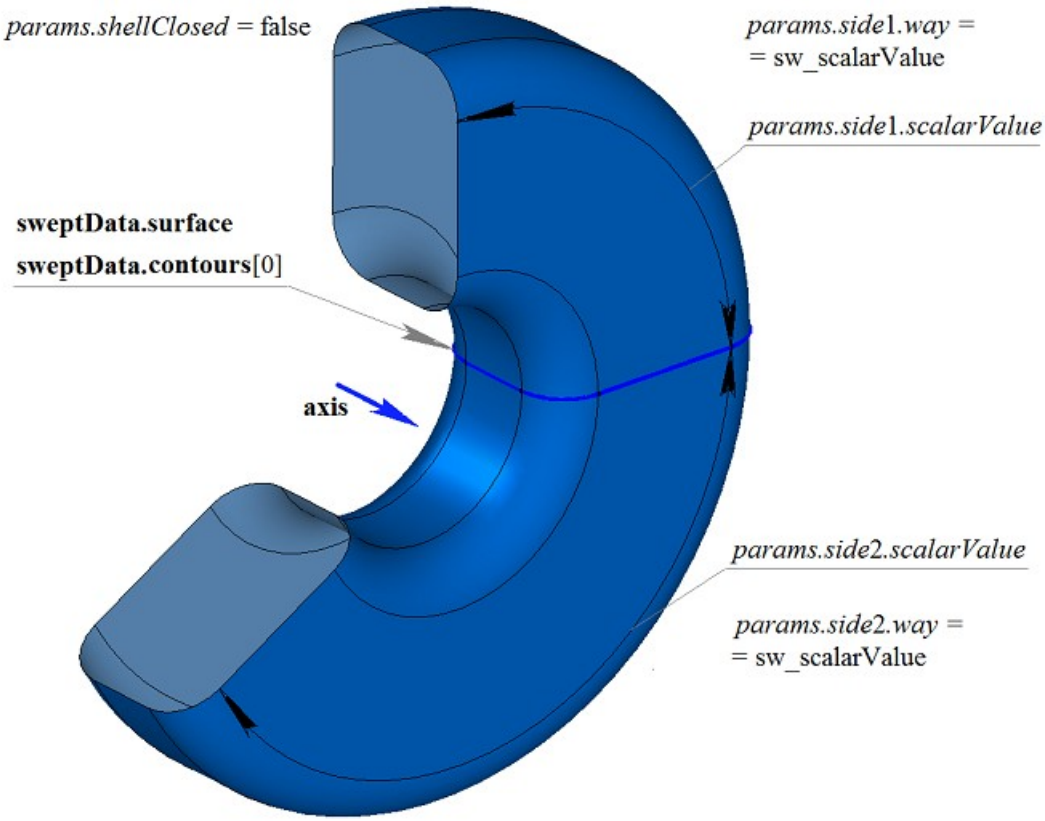


Рис. М.1.4.5.

На рис. М.1.4.6 приведен двумерный контур **contour**, плоская поверхность **surface** (MbPlane) и две поверхности **surface**1 и **surface**2, которые будут использоваться при построении тела вращения. Для построения необходимо, чтобы поверхности **surface**1 и **surface**2 полностью перекрывали путь движения контура в соответствующем направлении. При этом следует учитывать параметры *params.thickness*1, *params.thickness*2. Тело вращения обрезается заданными поверхностями или эквидистантными к ним поверхностями, если *params.side*1.*distance* или *params.side*2.*distance* не равны нулю.

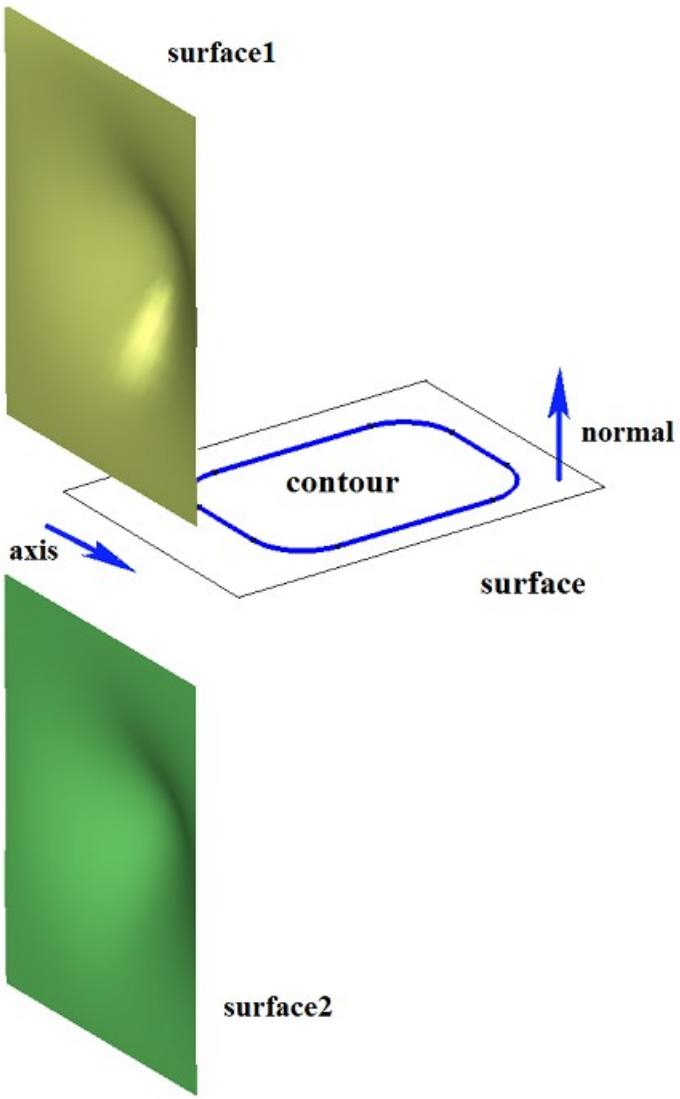


Рис. М.1.4.6.

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 **surface**1 и **surface**2.

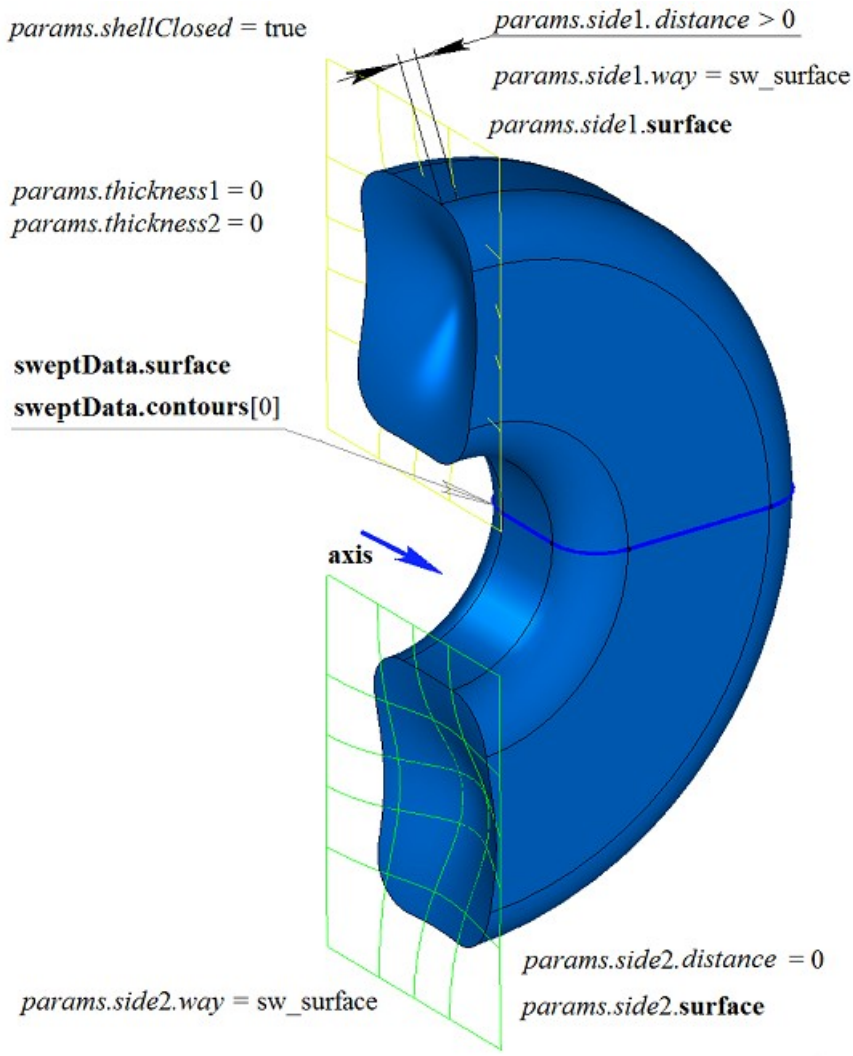


Рис. М.1.4.7.

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 **surface**1 и **surface**2.

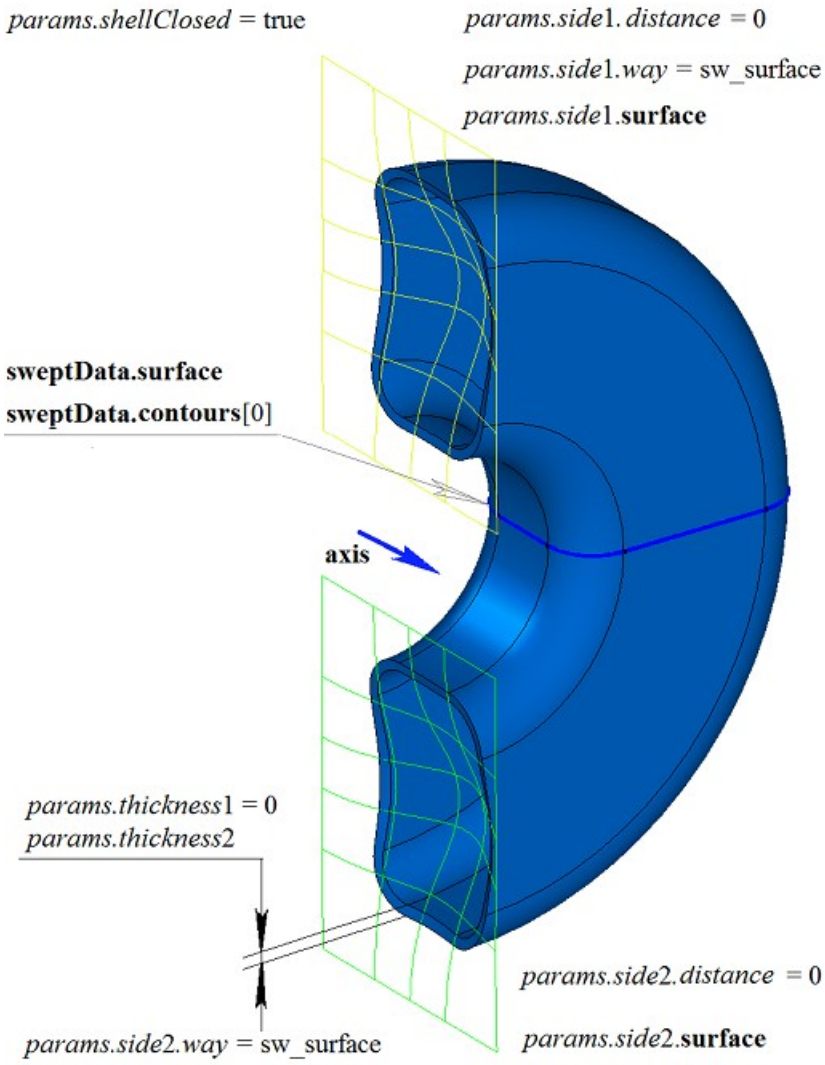


Рис. М.1.4.8.

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.

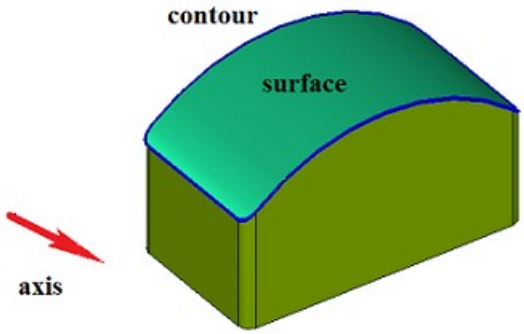


Рис. М.1.4.9

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

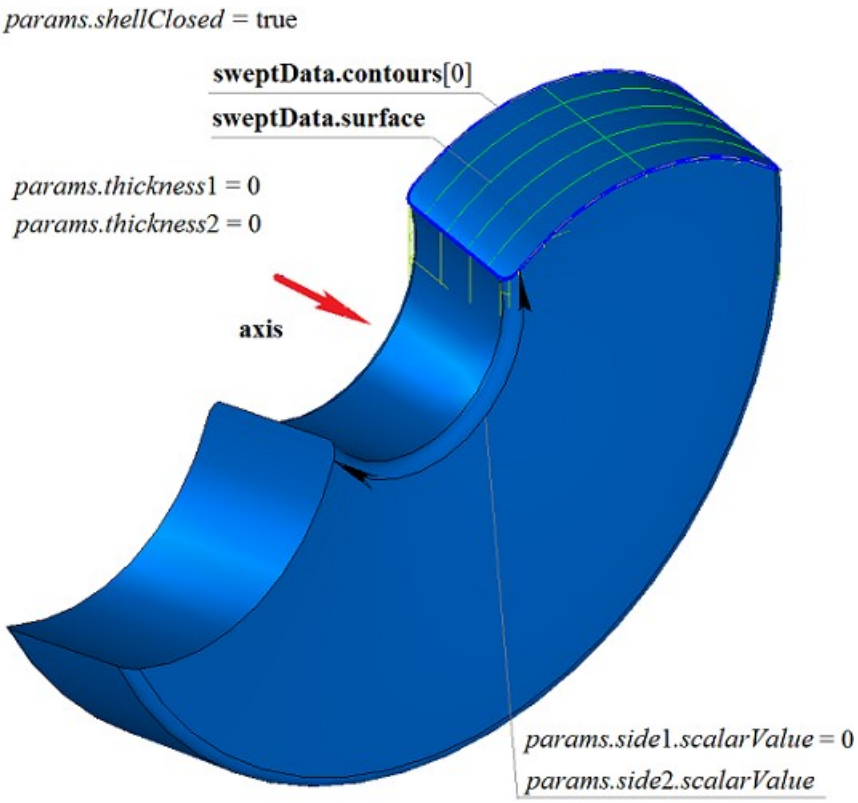


Рис. М.1.4.10.

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

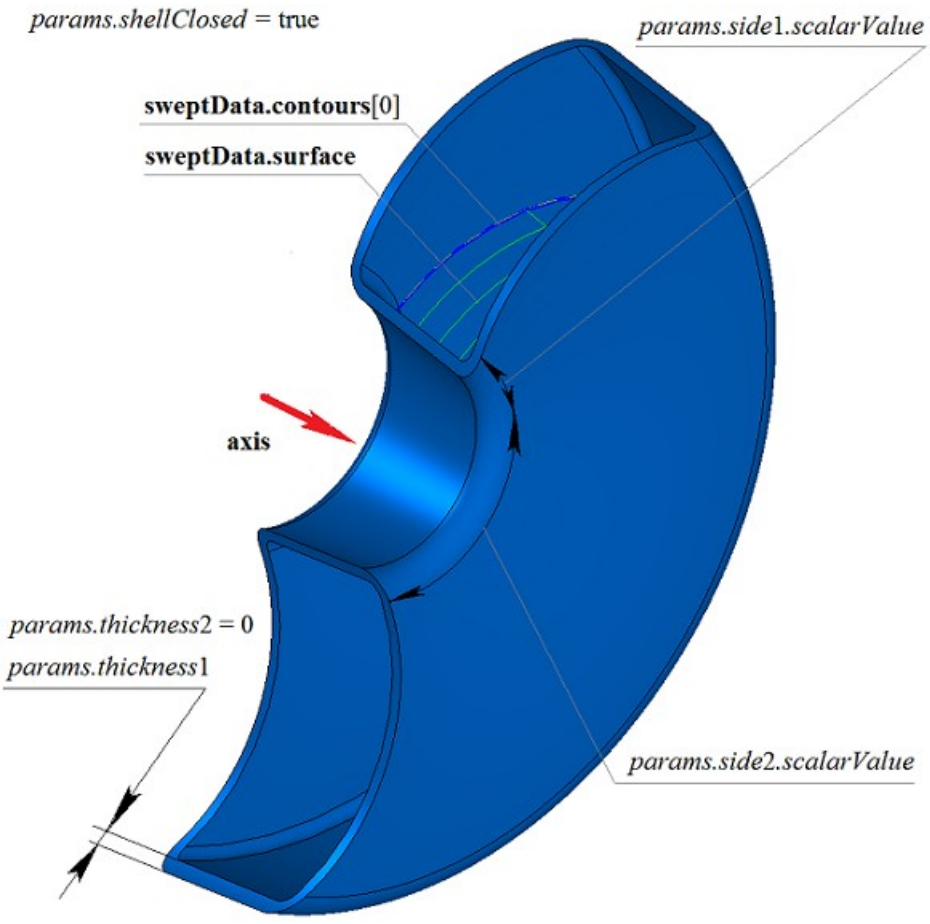


Рис. М.1.4.11.

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

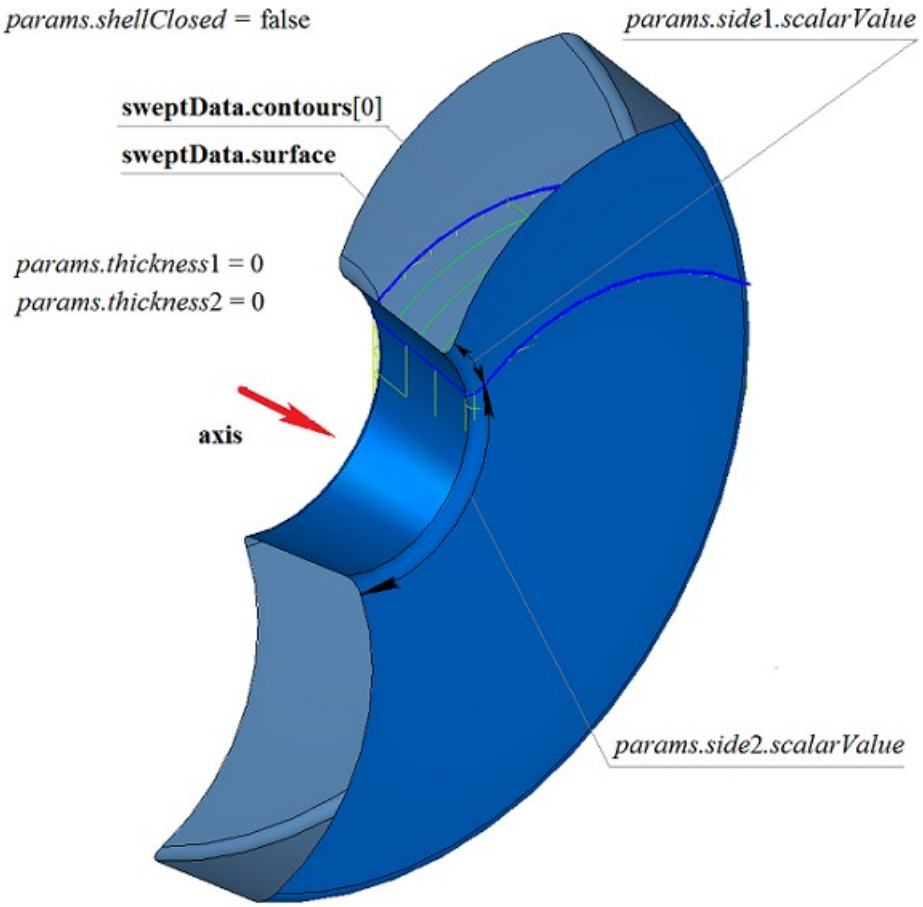


Рис. М.1.4.12.

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).

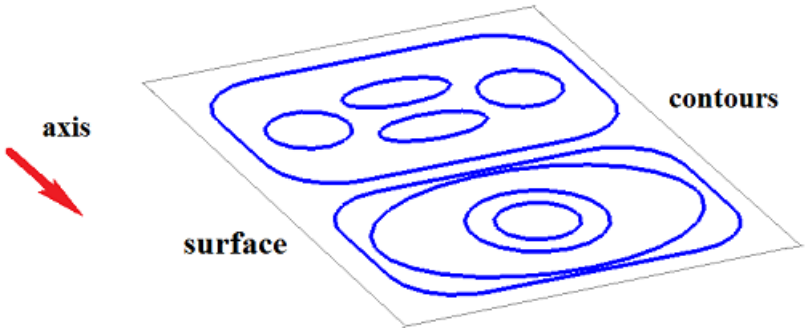


Рис. М.1.4.13.

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

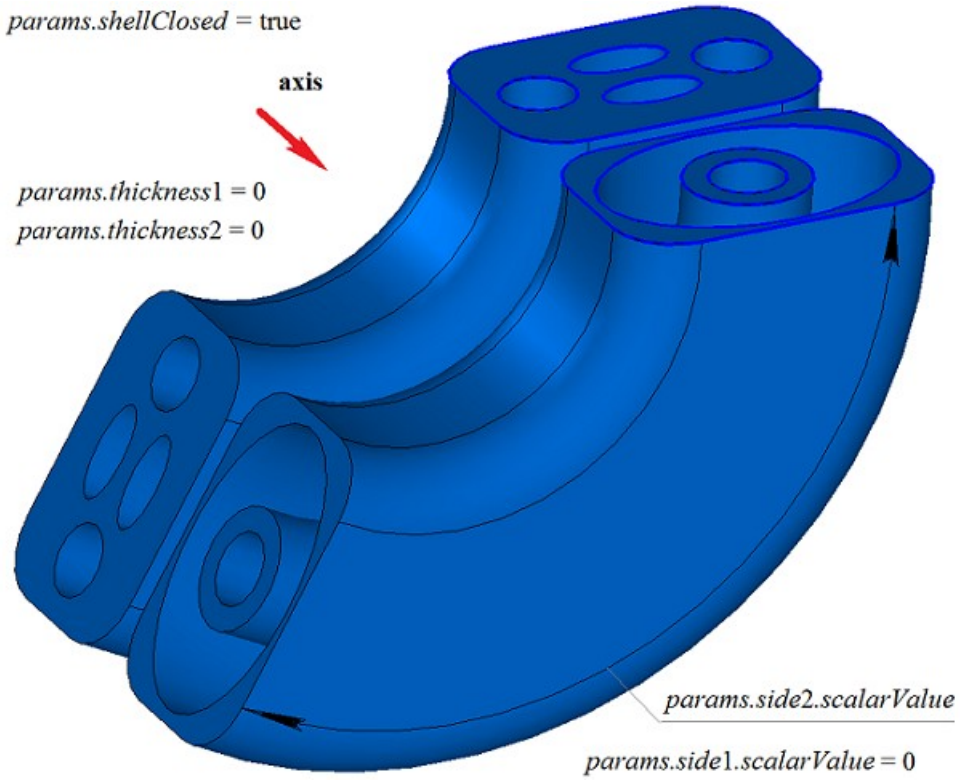


Рис. М.1.4.14.

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.

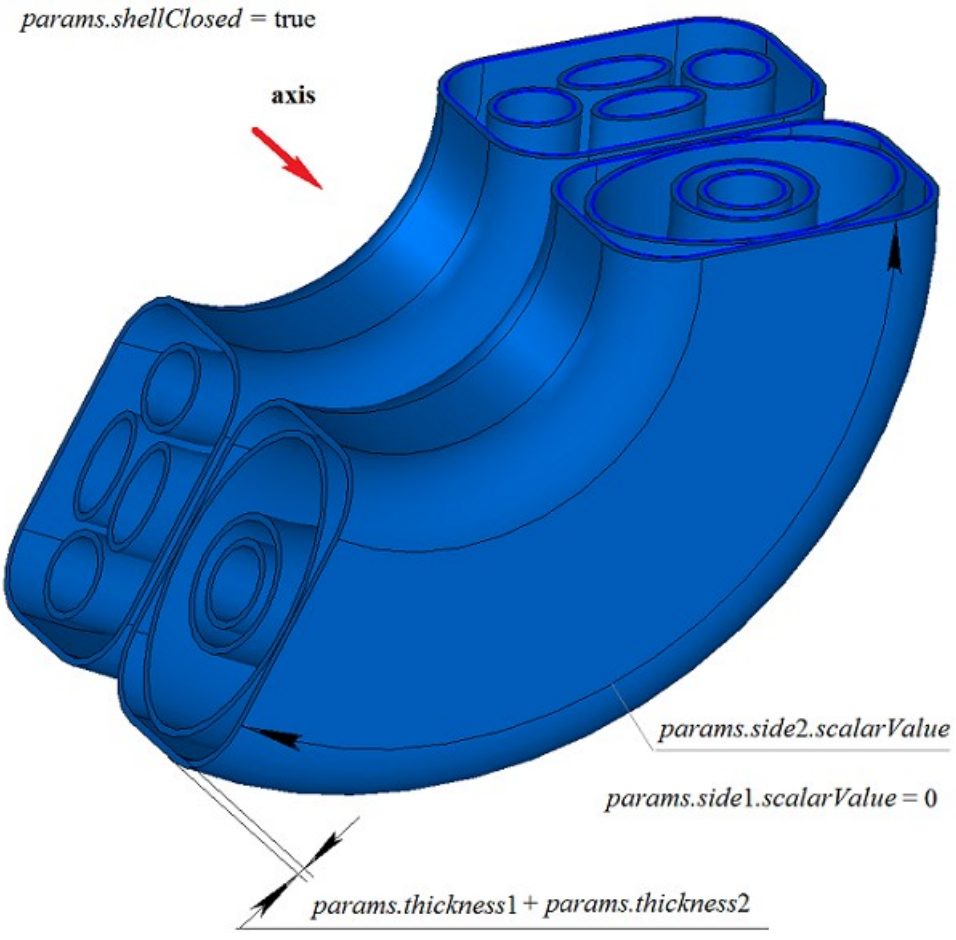


Рис. М.1.4.15.

На рис. М.1.4.16 приведены два трехмерных контура.

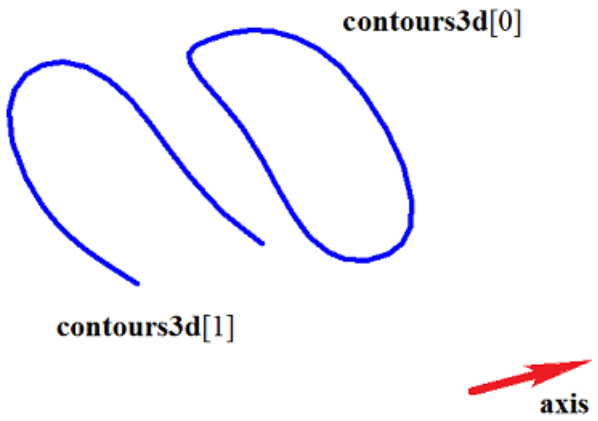


Рис. М.1.4.16.

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

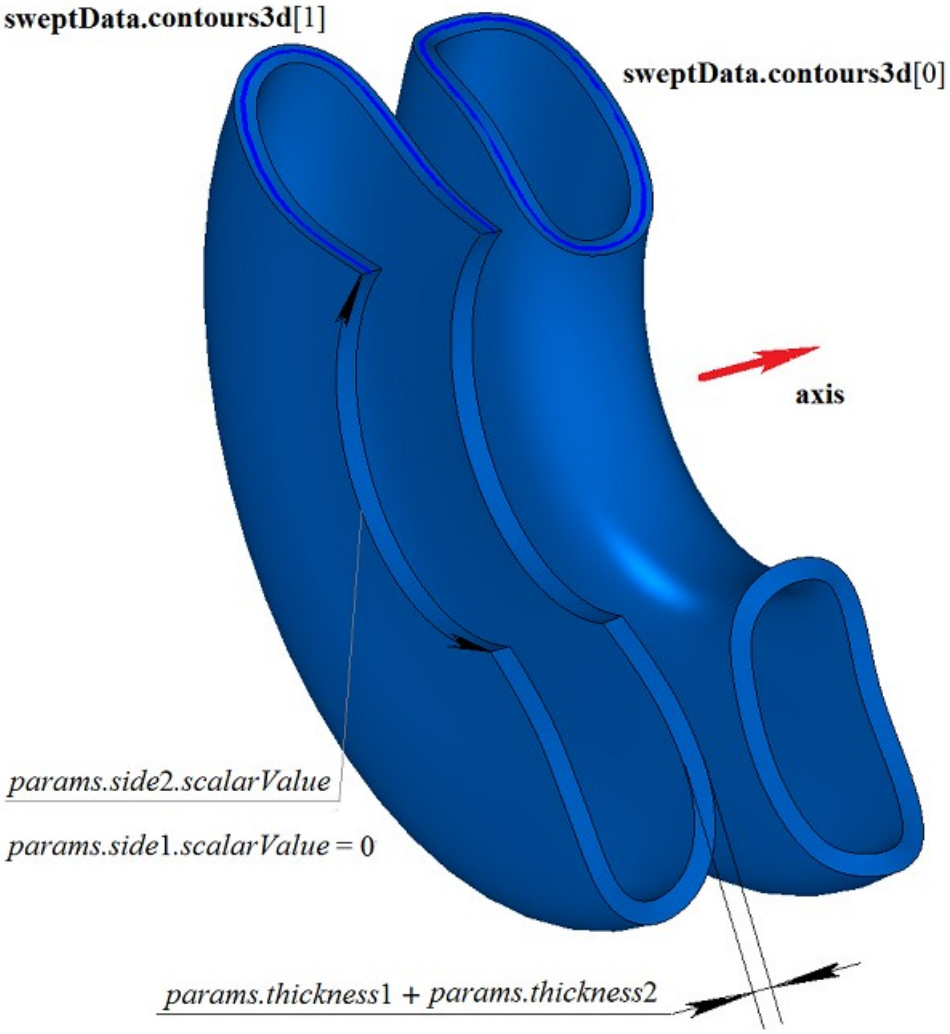


Рис. М.1.4.17.

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

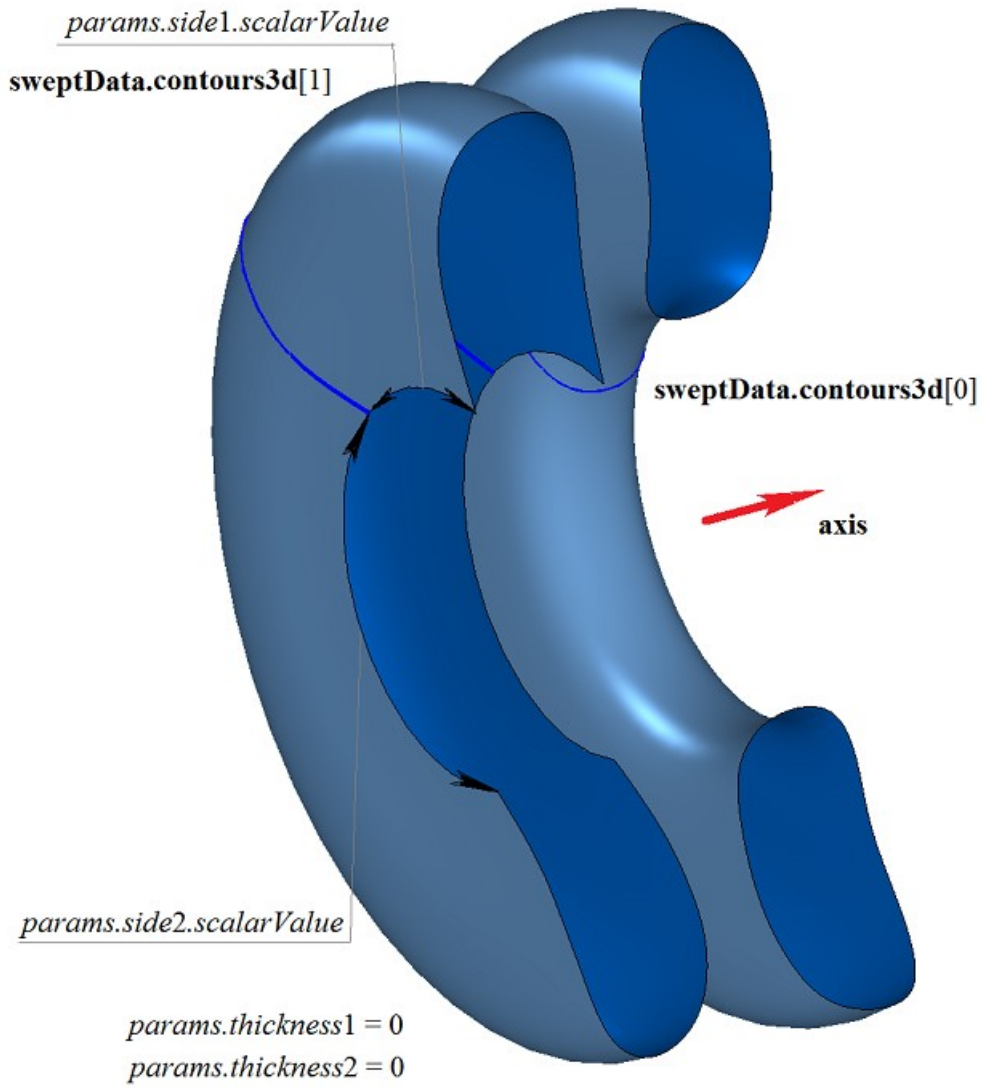


Рис. M.1.4.18.

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 ».

M.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 – именователь граней,

cnames – именователь образующей,
snames – именователь 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 описан в файле shell_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.thickness*1 и *params.thickness*2 определяют толщину стенки построенного тонкостенного тела. Параметр *params.thickness*1 задает отступ наружу от образующей кривой, а параметр *params.thickness*2 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, *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 движении 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*.

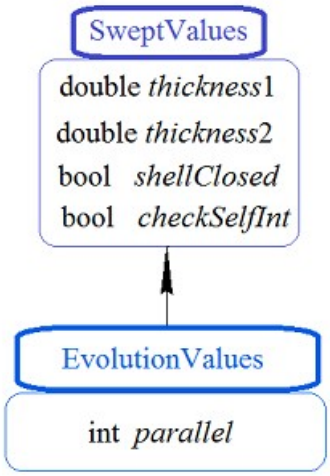


Рис. М.1.5.1.

Параметры names, cnames и snames обеспечивают именование граней построенного тела.

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

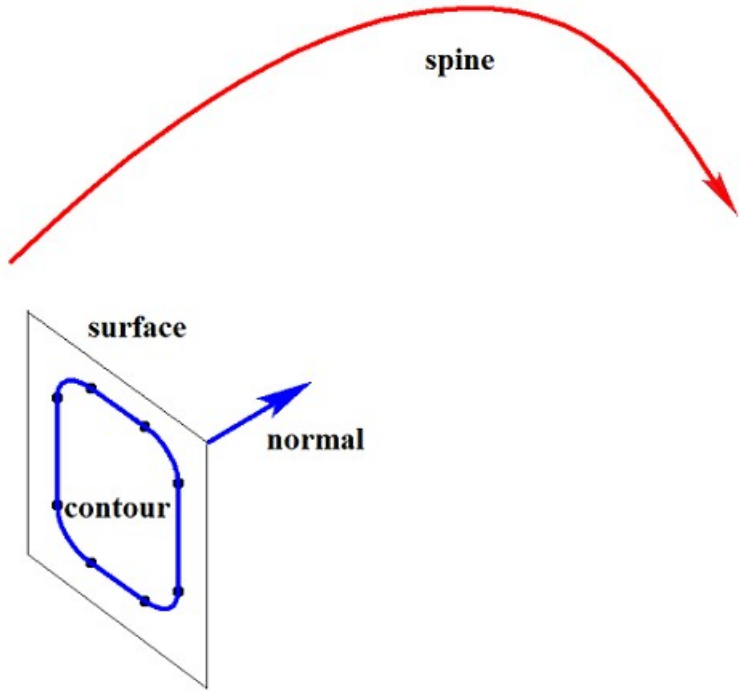


Рис. М.1.5.2.

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.

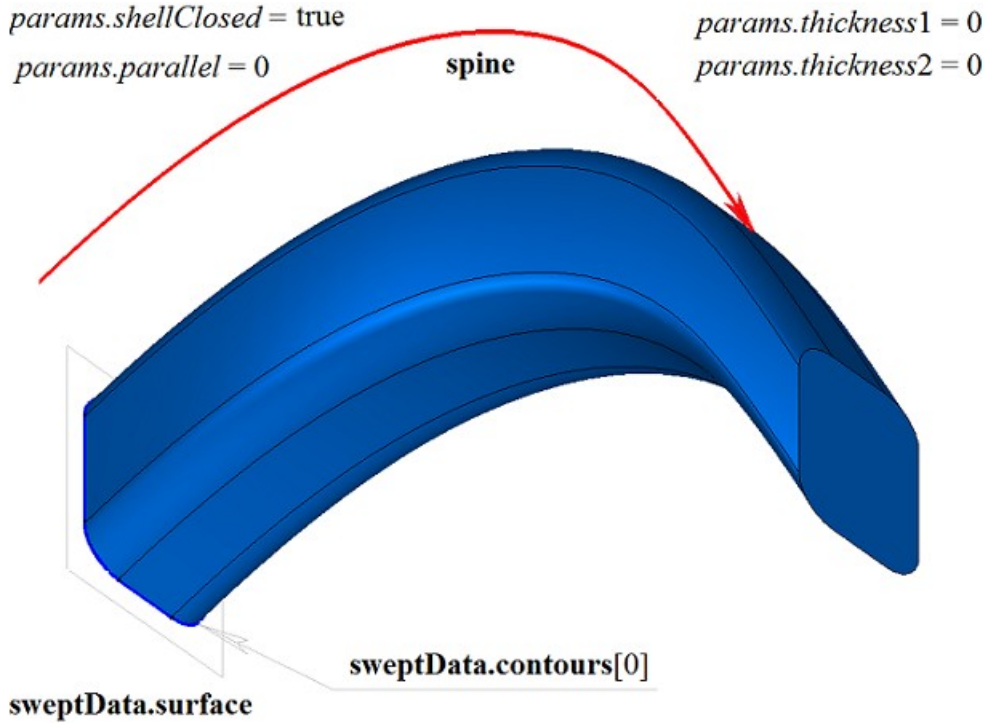


Рис. М.1.5.3.

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.

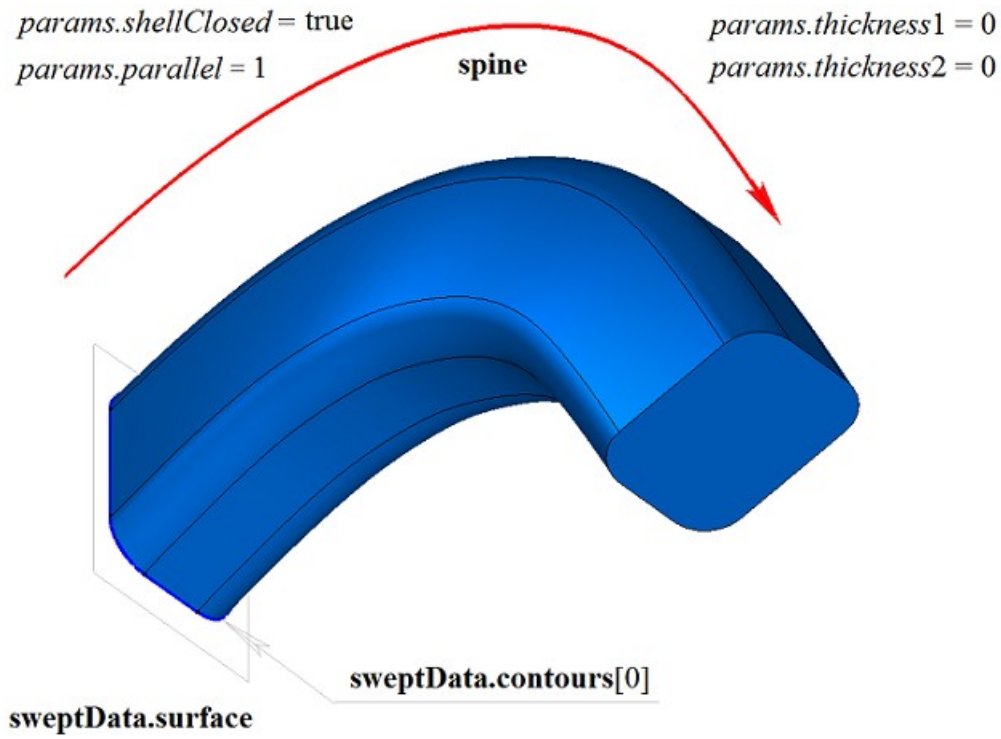


Рис. М.1.5.4.

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.

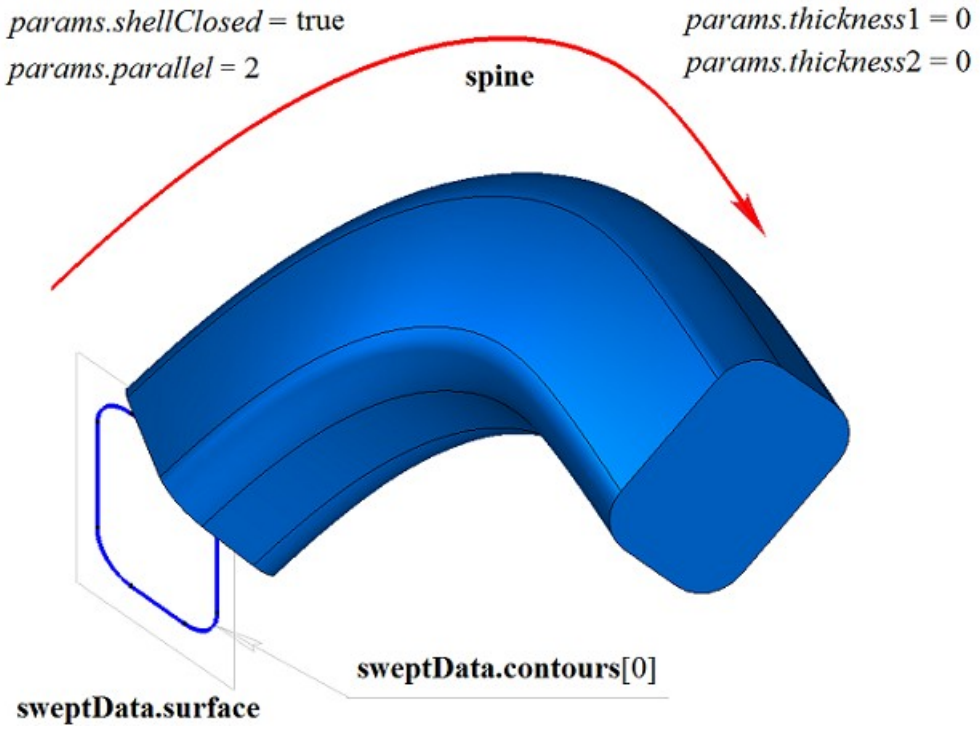


Рис. М.1.5.5.

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.paralle*=1. Каждому сегменту контура соответствует грань тела, имя которой взято от соответствующего элемента генератора имен cnames[0].

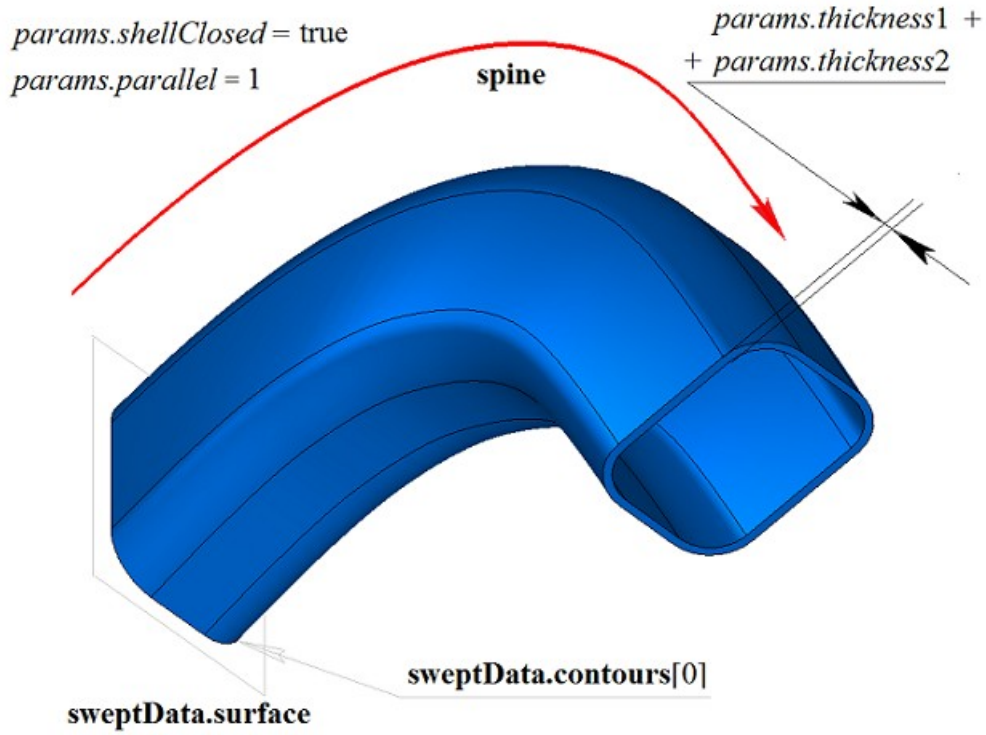


Рис. М.1.5.6.

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.

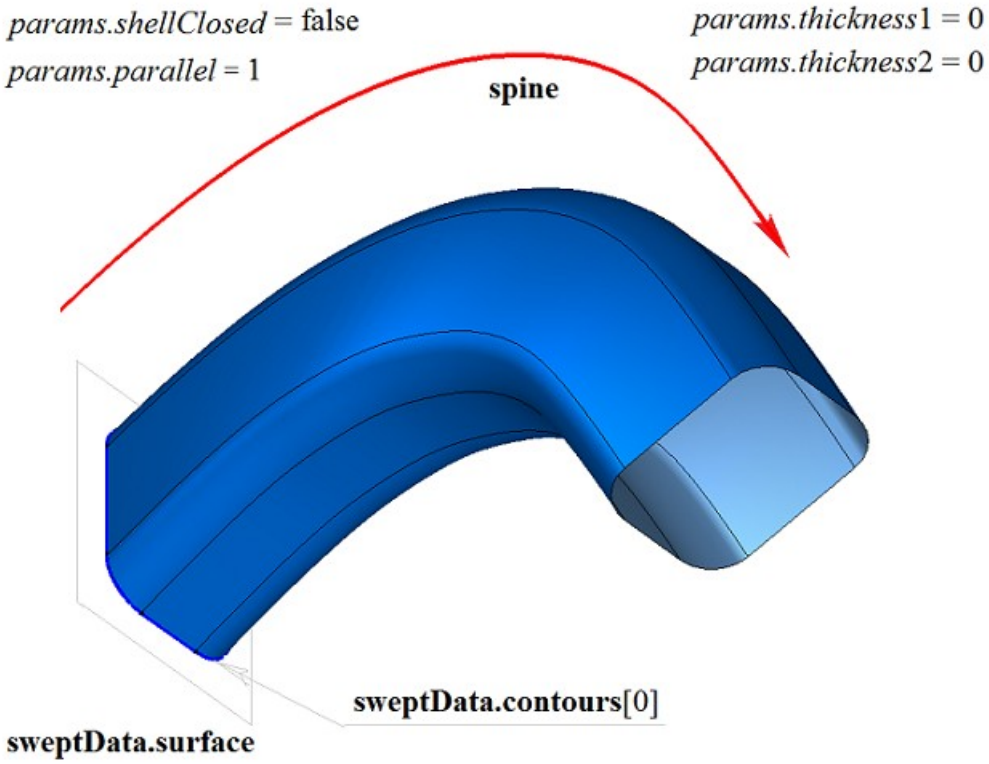


Рис. М.1.5.7.

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.

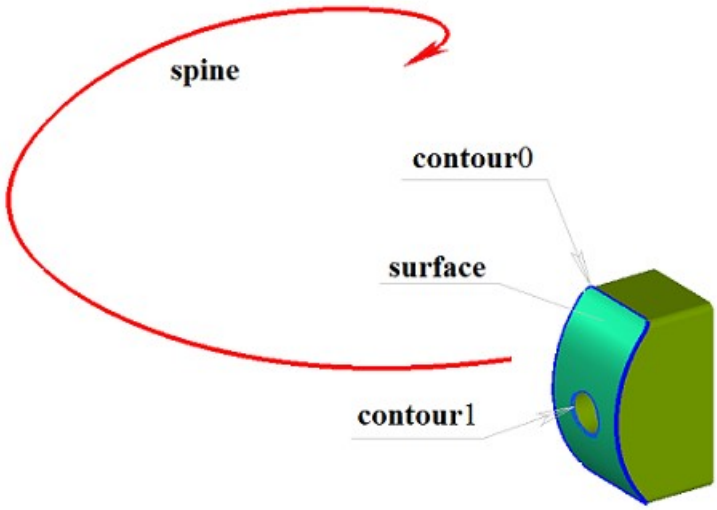


Рис. М.1.5.8.

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.paralle*=1.

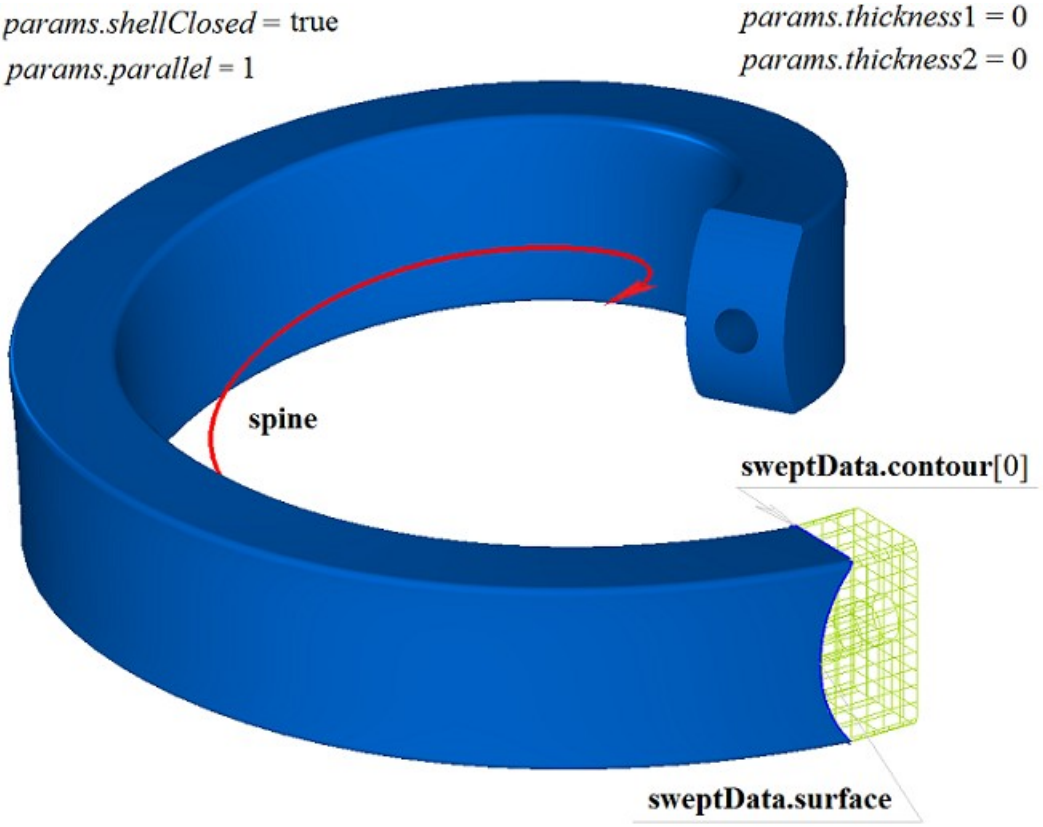


Рис. М.1.5.9.

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.

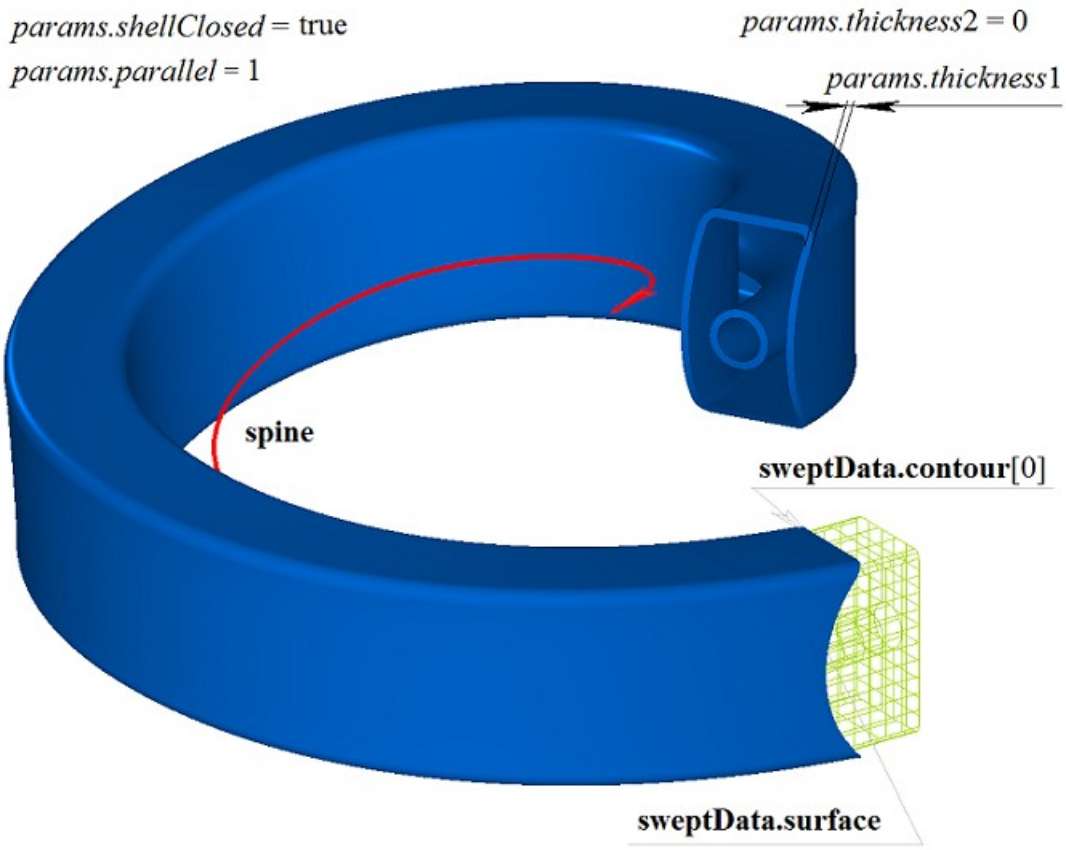


Рис. М.1.5.10.

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.

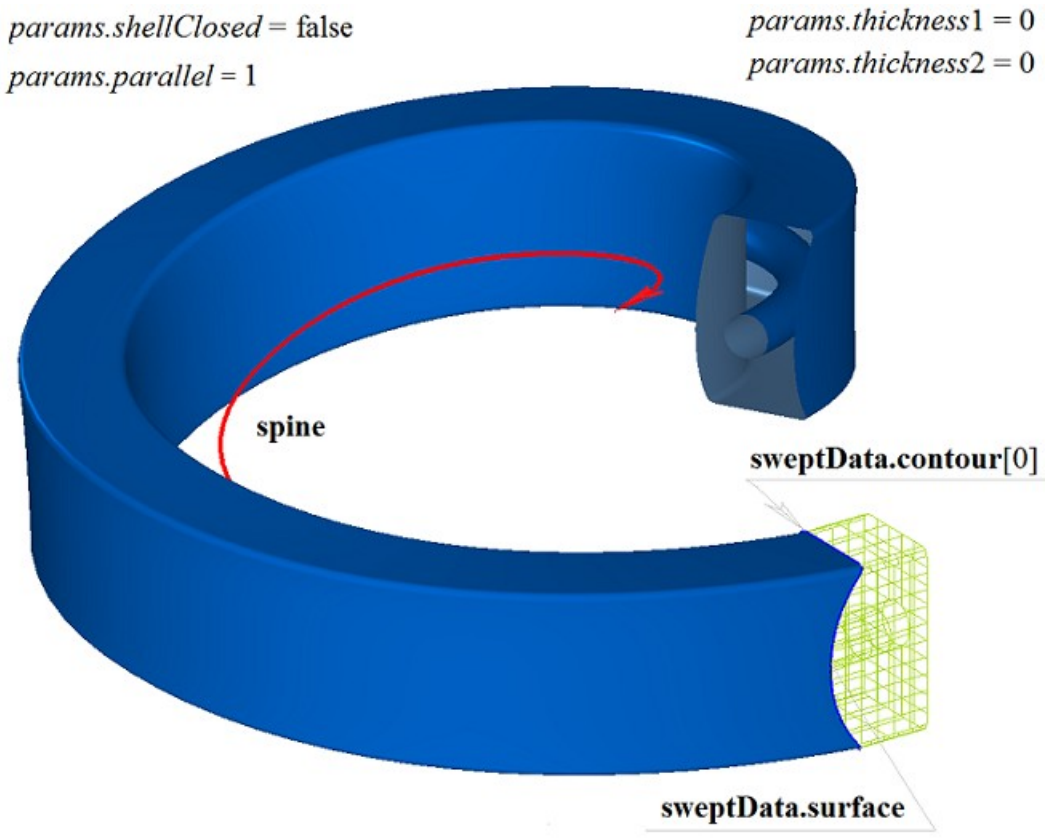


Рис. М.1.5.11.

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**.

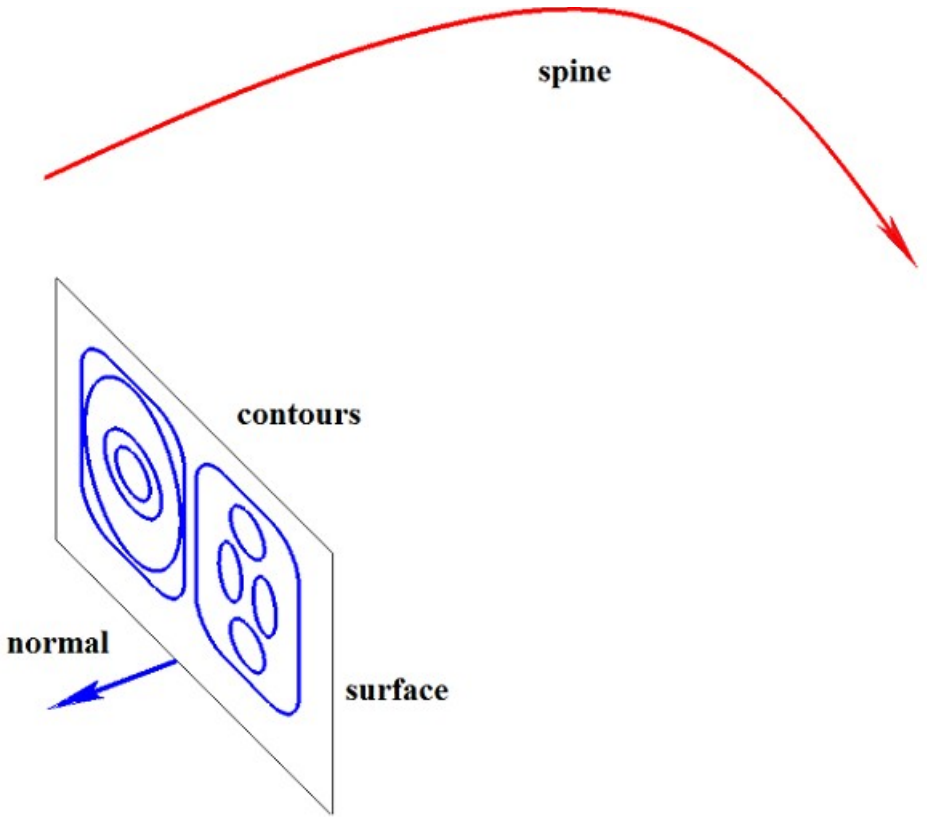


Рис. М.1.5.12.

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. Контуры не должны пересекаться, но могут быть вложены друг в друга, в том числе многократно.

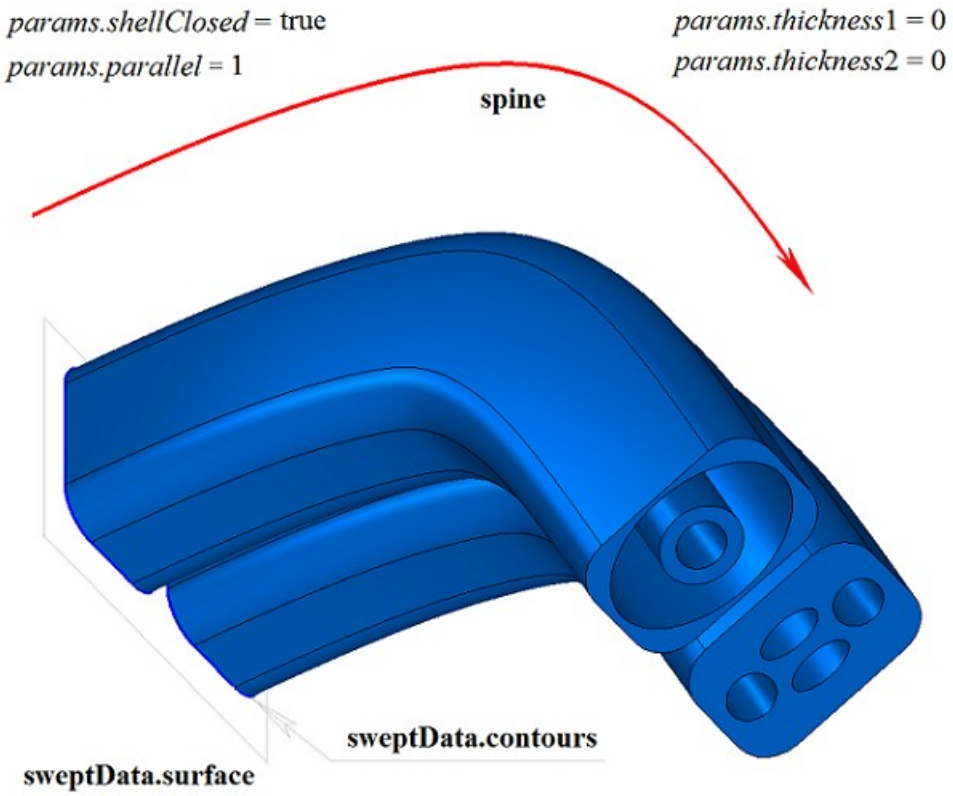


Рис. М.1.5.13.

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.

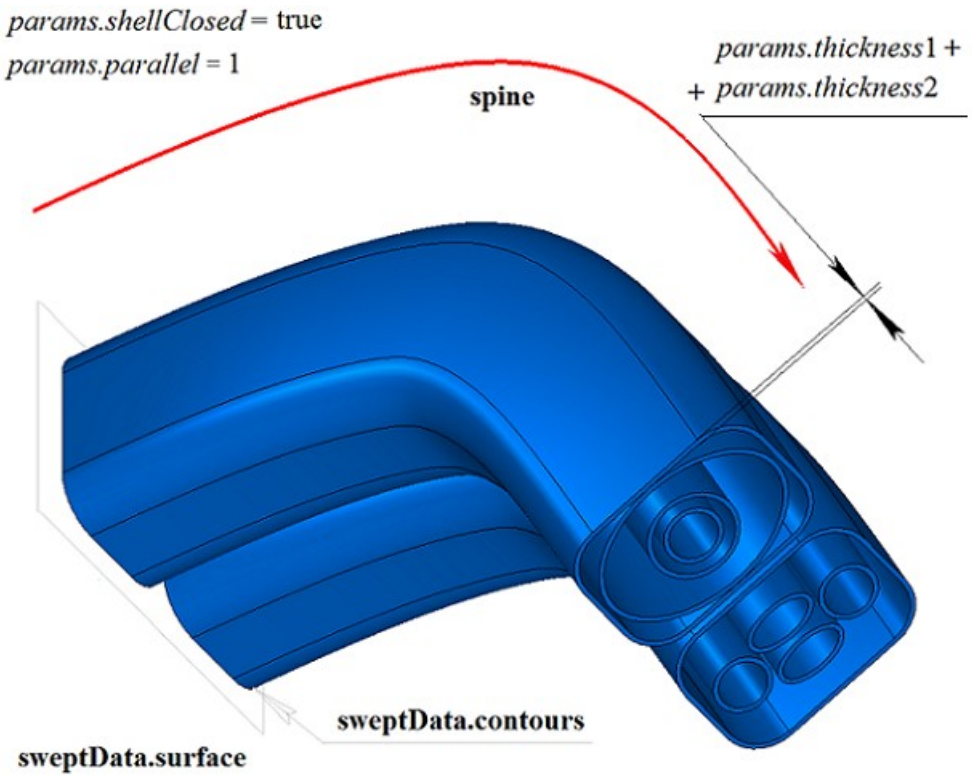


Рис. М.1.5.14.

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.

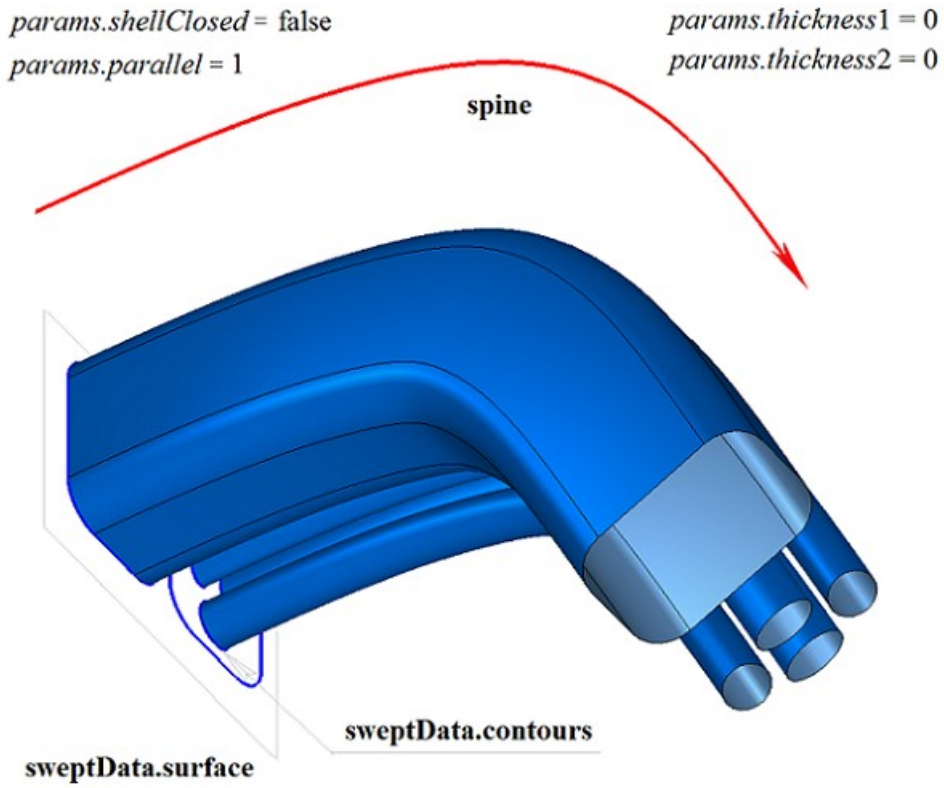


Рис. М.1.5.15.

На рис. М.1.5.16 приведены два трехмерных контура **contour3D**0, **contour3D**1 и направляющая кривая **spine**, которые будут использоваться для построения тел заметания.

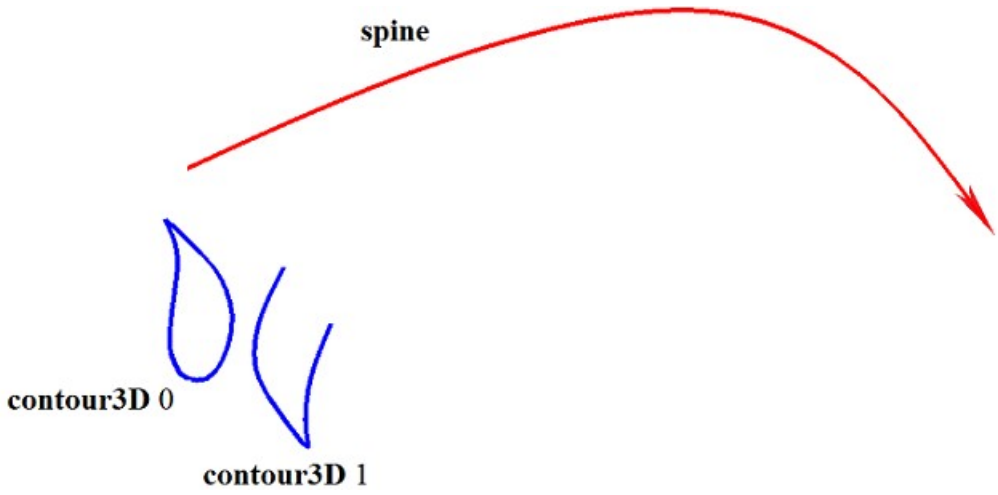


Рис. М.1.5.16.

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.

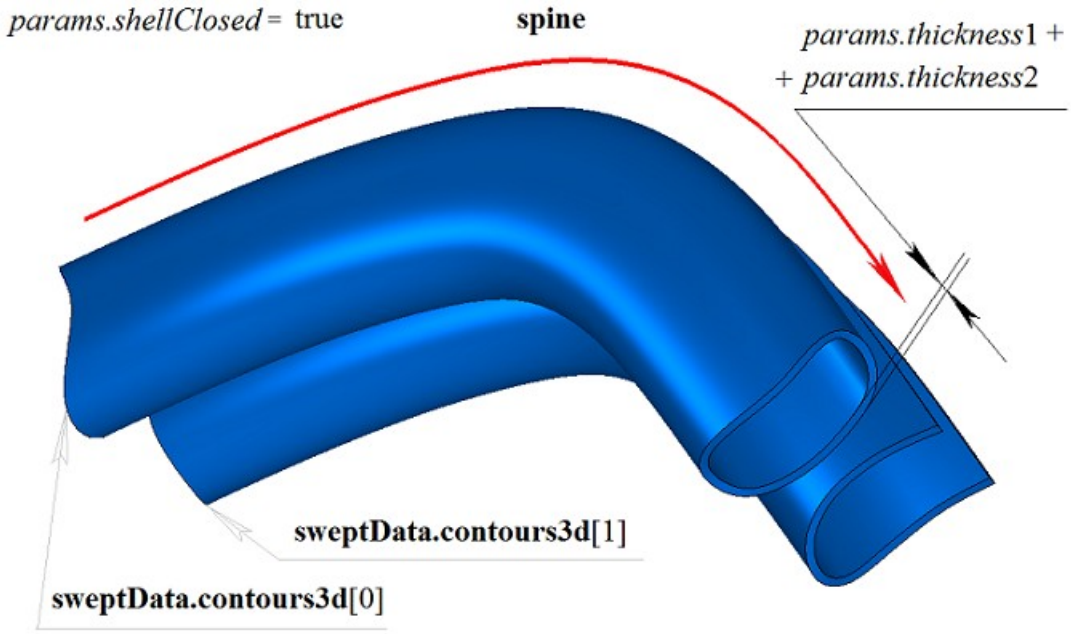


Рис. М.1.5.17.

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

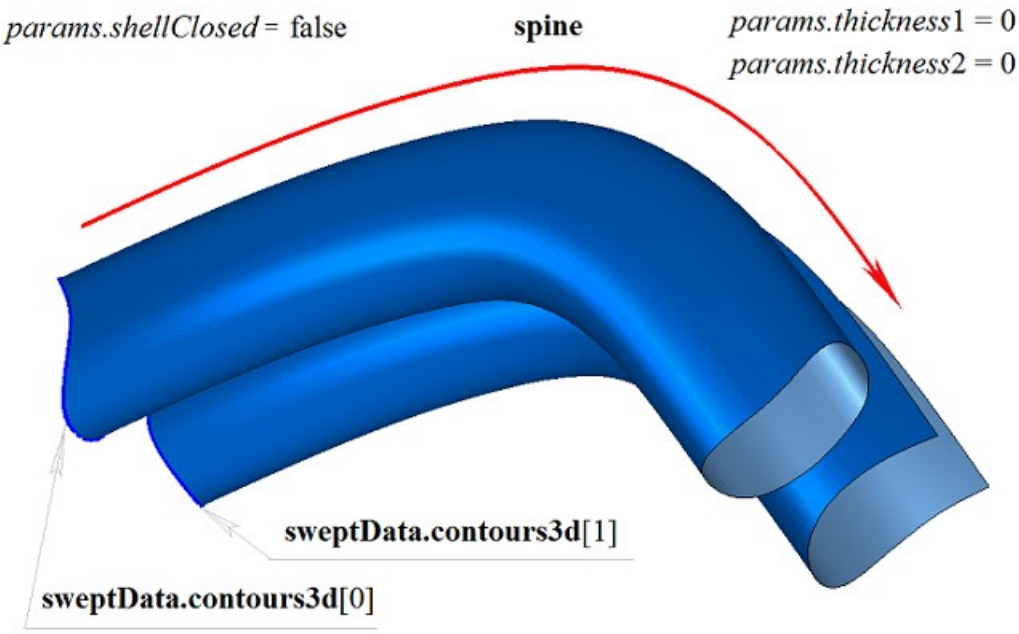


Рис. M.1.5.18.

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 cr_evolution_solid.h.

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

M.1.6. Построение тела по плоским сечениям

Метод

MbResultType

LoftedSolid (SArray<MbPlacement3D> & **places**,
RPArray<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.thickness*1 и *params.thickness*2 определяют толщину стенки построенного тонкостенного тела. Параметр *params.thickness*1 задает отступ наружу от образующей кривой, а параметр *params.thickness*2 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*.**vector**1 и *params*.**vector**2 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*.**vector**1 и *params*.**vector**2 равны нулю.

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

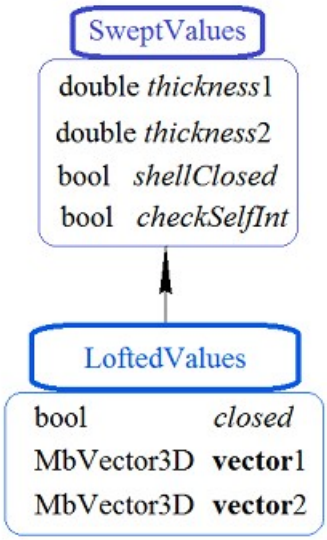


Рис. М.1.6.1.

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.

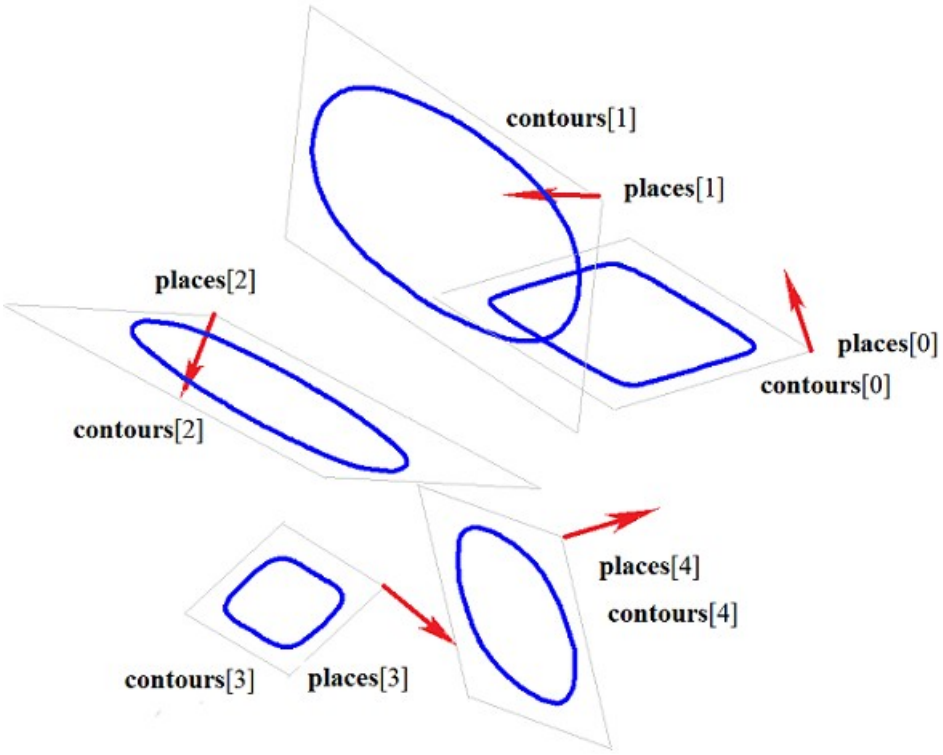


Рис. М.1.6.2.

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.

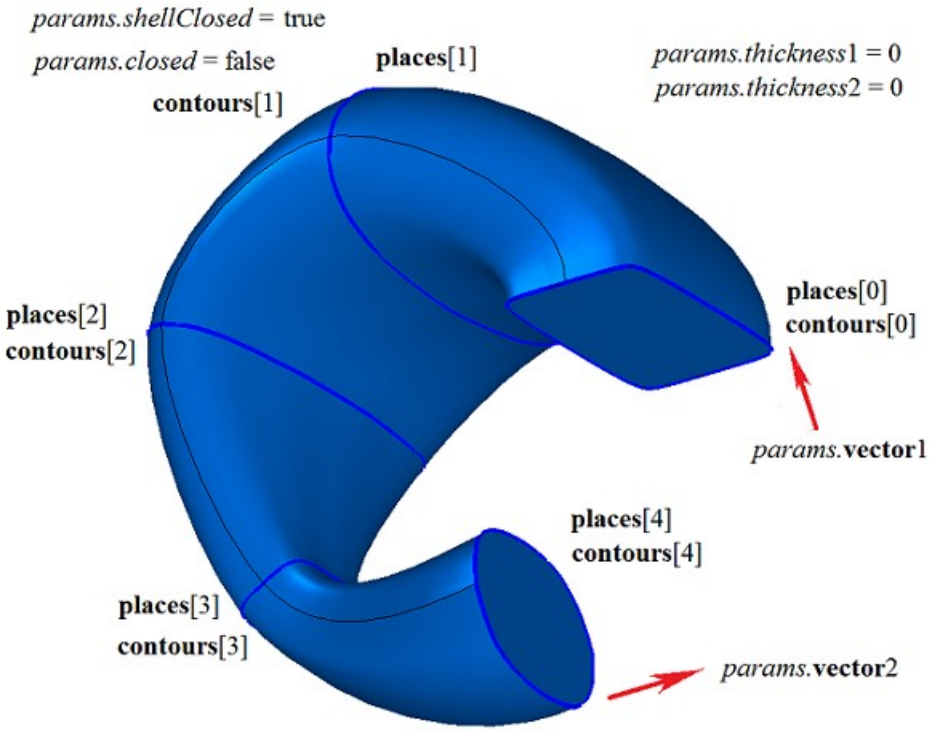


Рис. М.1.6.3.

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 теле имеет топологию тора и у него отсутствуют торцы.

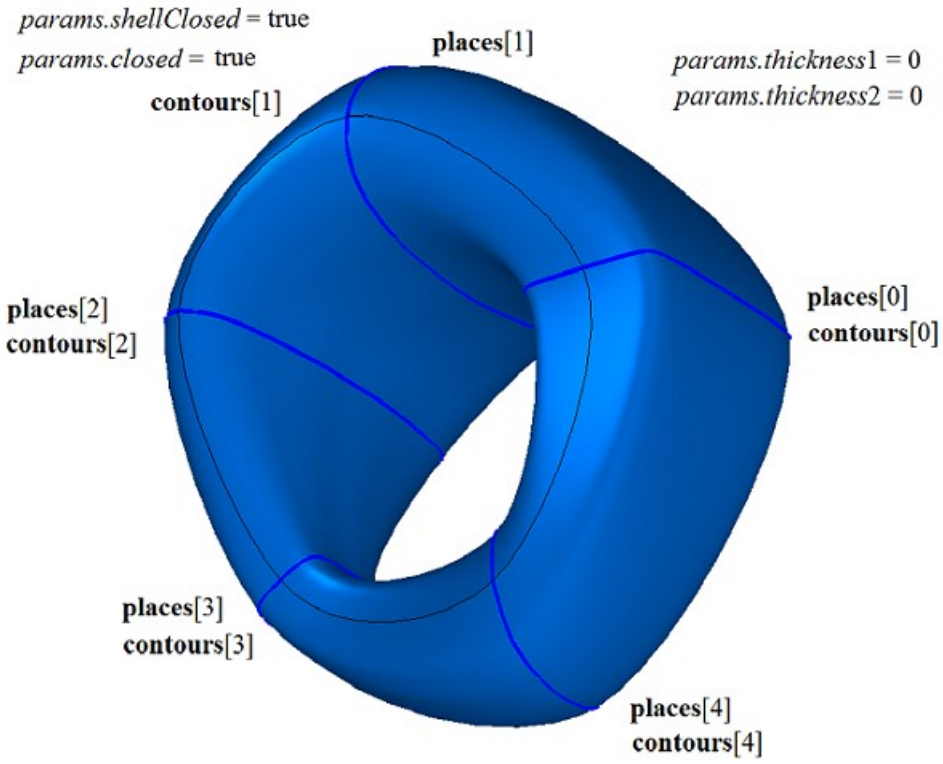


Рис. М.1.6.4.

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.

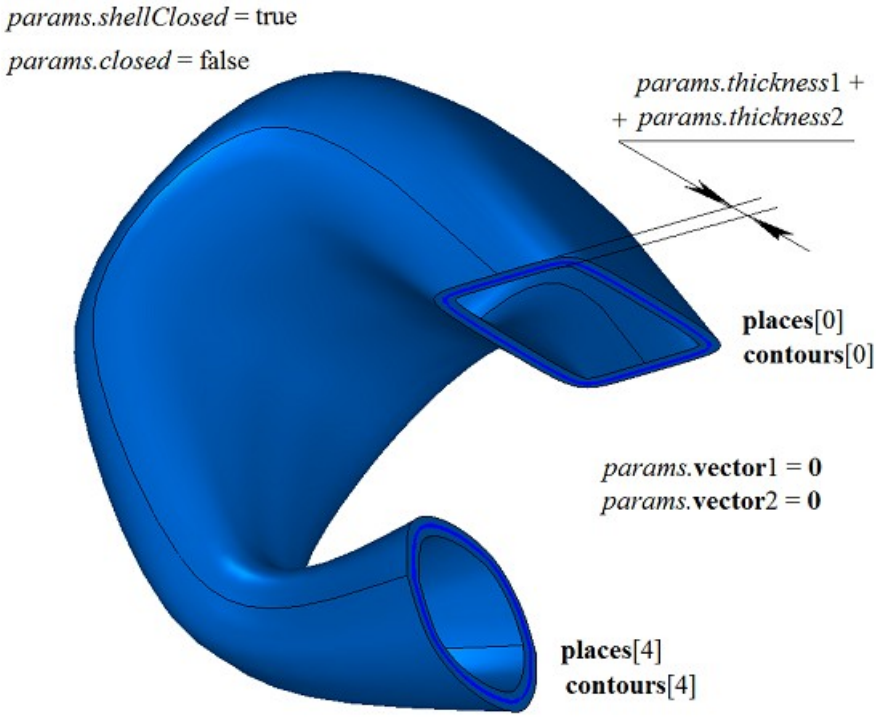


Рис. М.1.6.5.

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

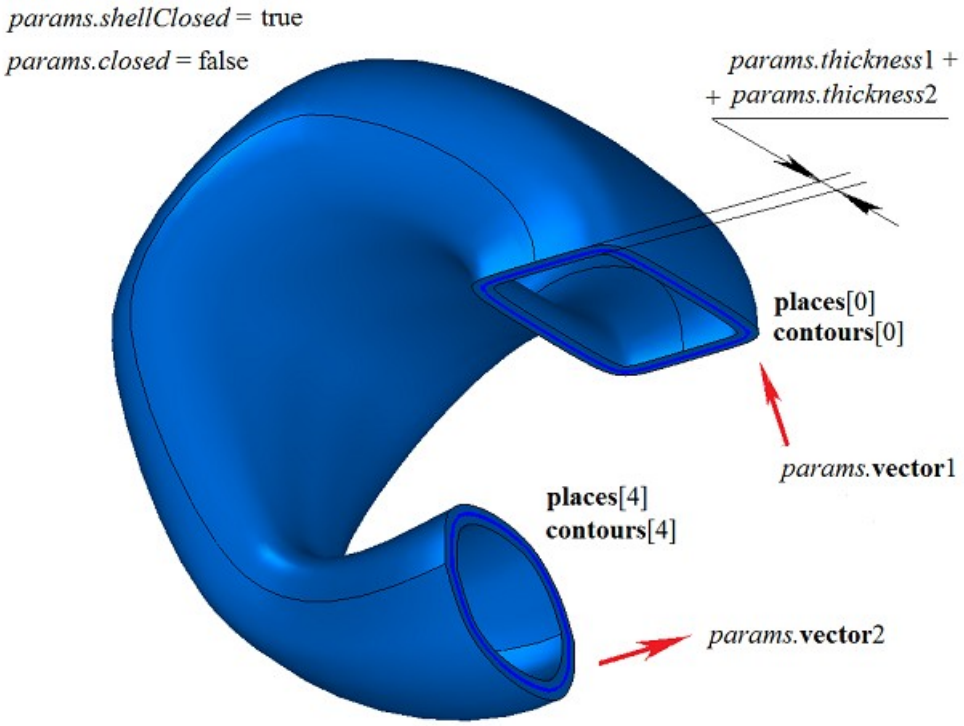


Рис. М.1.6.6.

На рис. М.1.6.7 приведено тонкостенное тело, построенное по плоским незамкнутым контурам с не заданными нормалями на торцах для значения параметра *params.closed*=false. Параметры *params.thickness*1 и *params.thickness*2 должны быть не нулевыми.

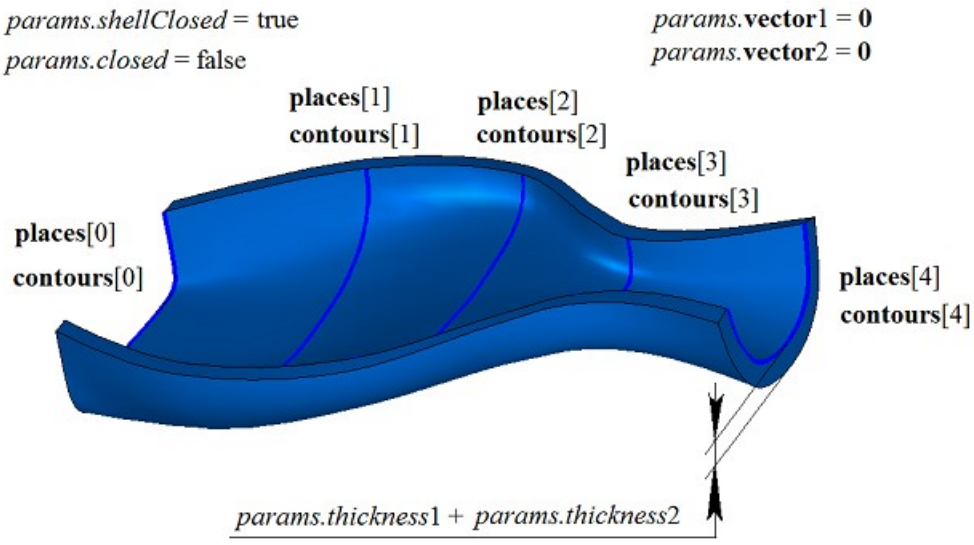


Рис. М.1.6.7.

На рис. М.1.6.8 приведено незамкнутое тело, построенное по плоским незамкнутым контурам с не заданными нормалями на торцах для значения параметра *params.closed*=false. Параметры *params.thickness*1 и *params.thickness*2 могут быть нулевыми.

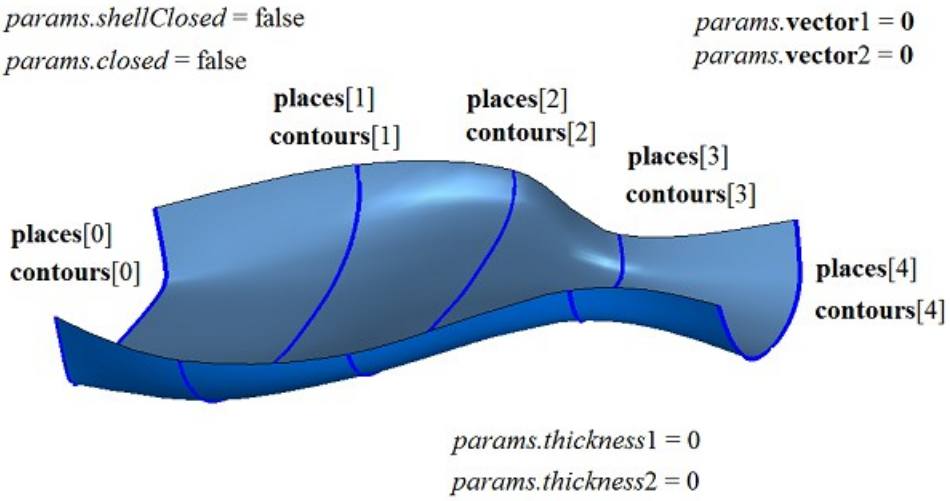


Рис. М.1.6.8.

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

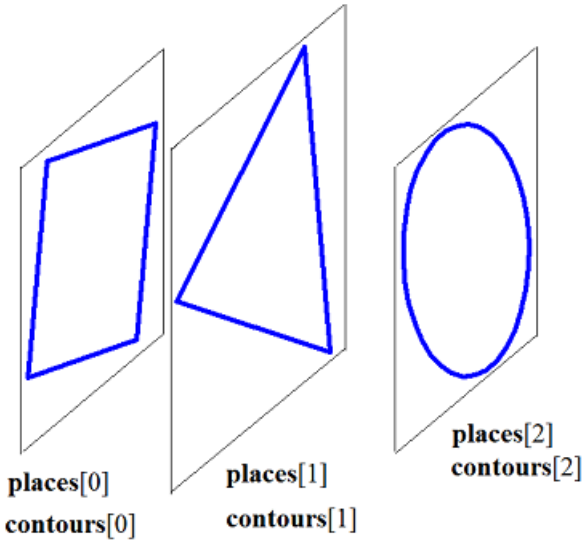


Рис. М.1.6.9.

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

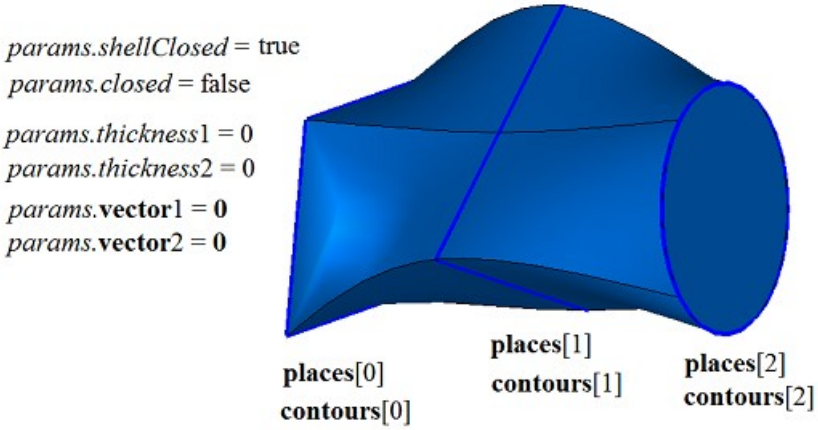


Рис. М.1.6.10.

Управлять положением ребер, соединяющих вершины разных контуров множества, можно с помощью контрольных точек **points**. Тоски **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.

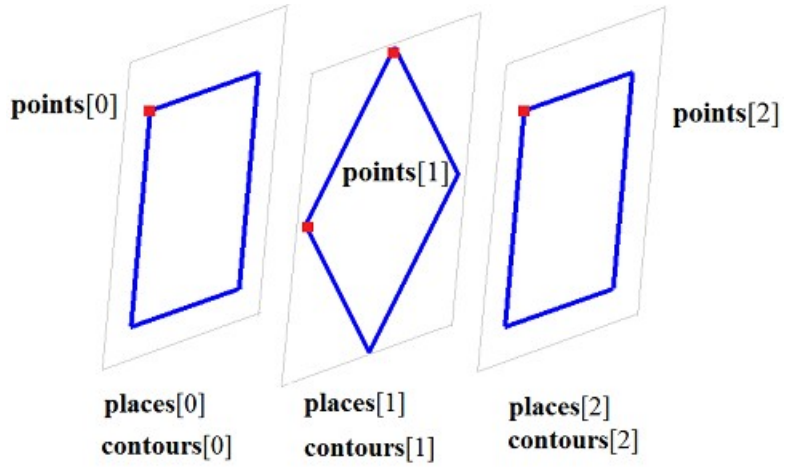


Рис. М.1.6.11.

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**.

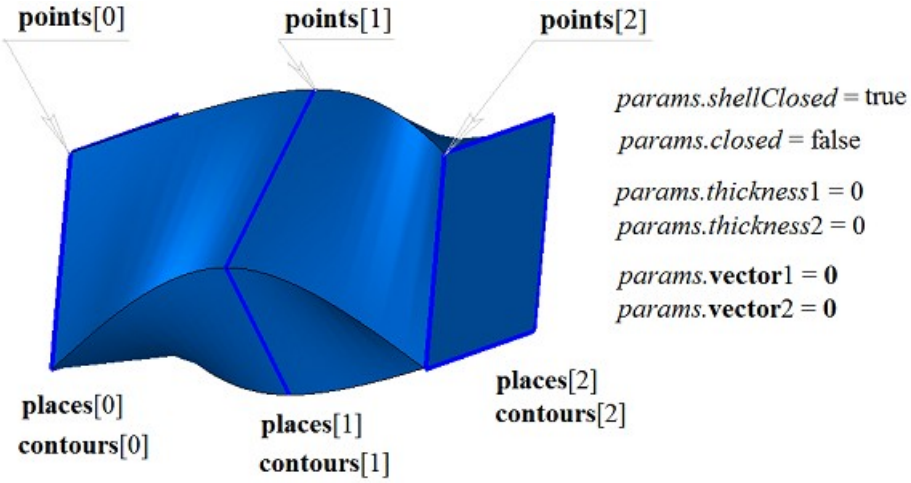


Рис. М.1.6.12.

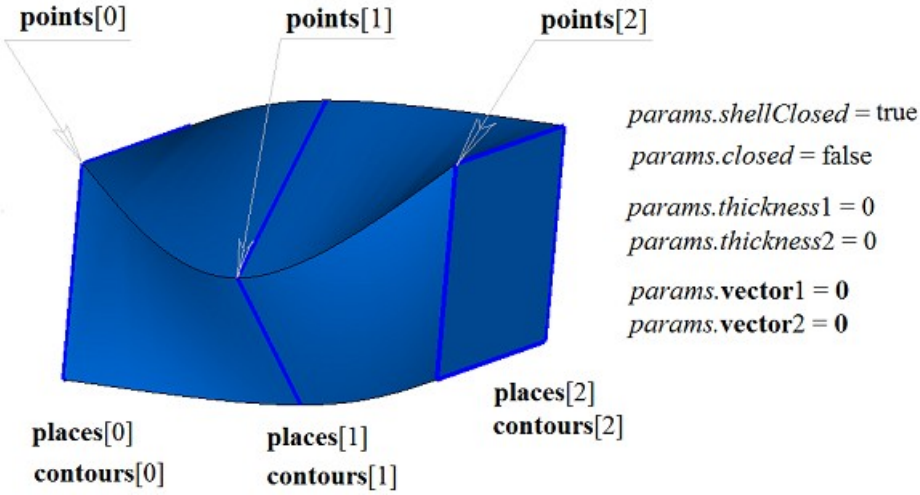


Рис. М.1.6.13.

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.

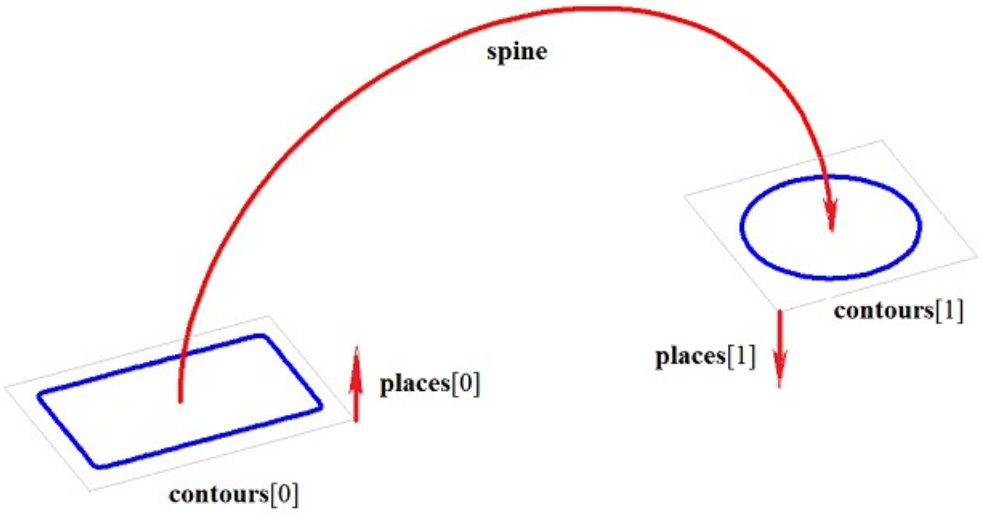


Рис. М.1.6.14.

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

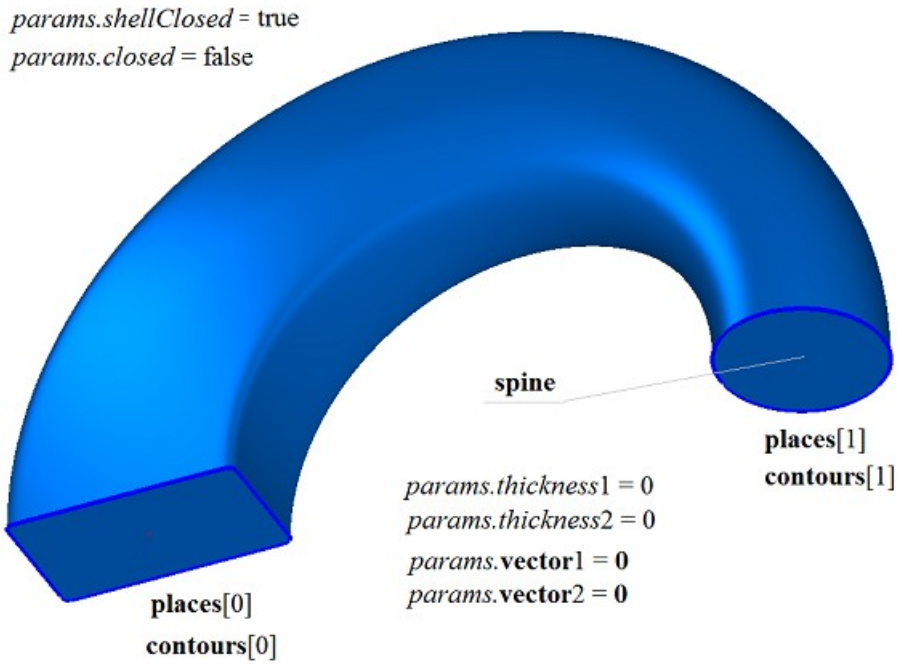


Рис. М.1.6.15.

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

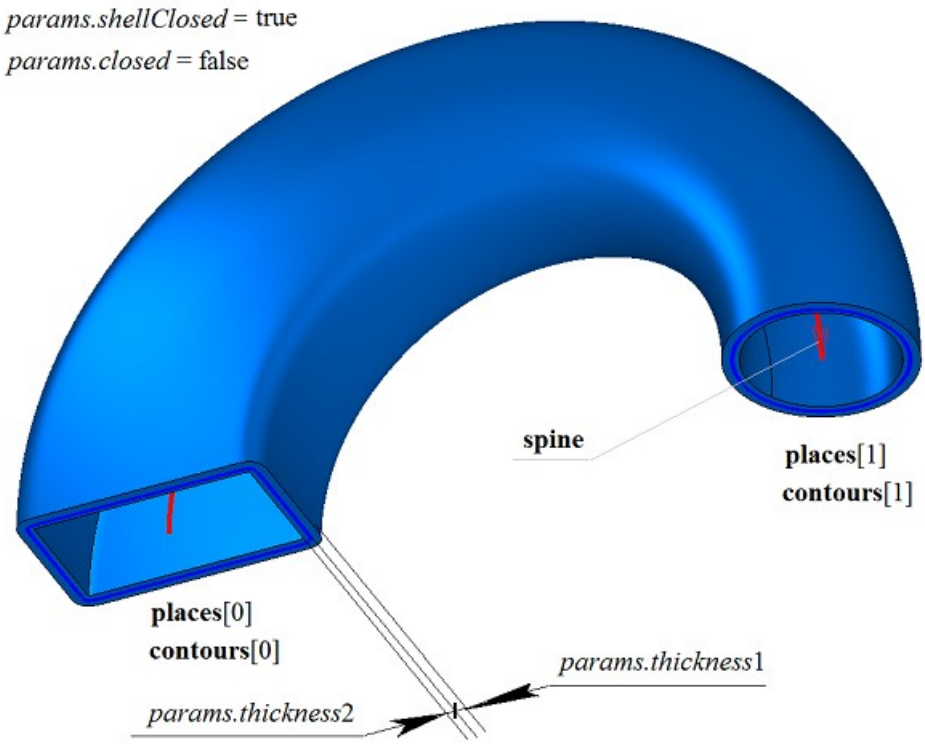


Рис. М.1.6.16.

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

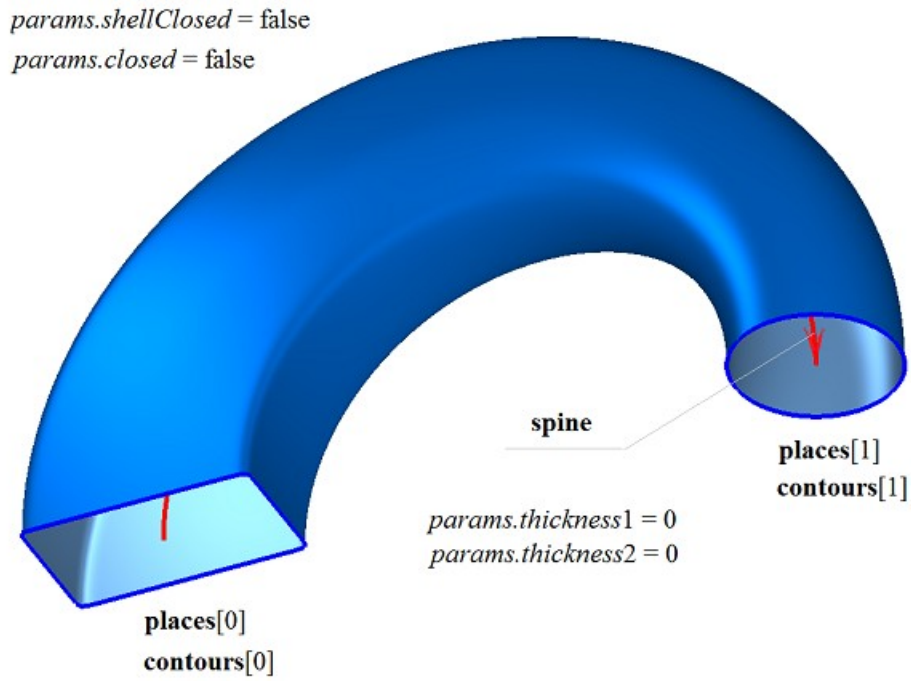


Рис. М.1.6.17.

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.

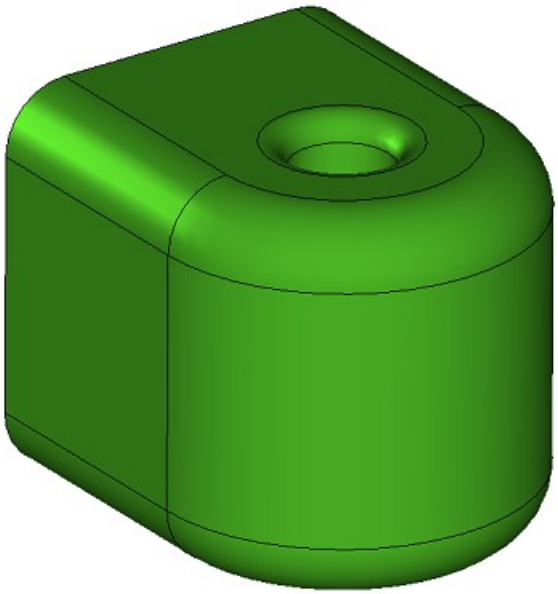
Входными параметрами метода являются:
faceSet – множество граней,
names – именователь граней.

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 именование 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
ElementaryShell (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.

M2. 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. Замкнутое тело описывает всю поверхность моделируемого объекта. Замкнутому телу принадлежат точки его граней и множество точек, расположенных с внутренней стороны граней. Незамкнутое тело описывает только часть поверхности моделируемого объекта. Незамкнутому телу принадлежат только точки его граней. Для точек пространства, не принадлежащих граням, невозможно определить, находятся ли они внутри или снаружи относительно незамкнутого тела, поэтому некоторые операции с незамкнутыми телами носят иной характер, чем операции с замкнутыми телами.

M.2.1. Булева операция над телами

Метод
MbResultType
BooleanResult (MbSolid & **solid1**,
MbeCopyMode *sameShell*1,
MbSolid & **solid2**,
MbeCopyMode *sameShell*2,
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,
*sameShell*1 – вариант копирования первого тела,
solid2 – второе тело для булевой операции,
*sameShell*2 – 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 *sameShell*1 и *sameShell*2 управляют передачей граней, ребер и вершин от исходных тел **solid1** и **solid2** построенному телу **result**.

Параметры *sameShell*1 и *sameShell*2 могут принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* соответствующее исходное тело (**solid1** или **solid2**) полностью копируется, поэтому построенное тело **result** и соответствующее исходное тело не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и соответствующее исходное тело (**solid1** или **solid2**) будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и соответствующее исходное тело (**solid1** или **solid2**) будут иметь одни и те же вершины, базовые поверхности граней и не затронутые операцией грани. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты соответствующего исходного тела (**solid1** или **solid2**), поэтому после построения соответствующее исходное тело должно быть удалено. Перечисление MbeCopyMode 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 **BooleanSolid**, описанным в параграфе М.2.2. Булева операция над незамкнутыми телами.

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

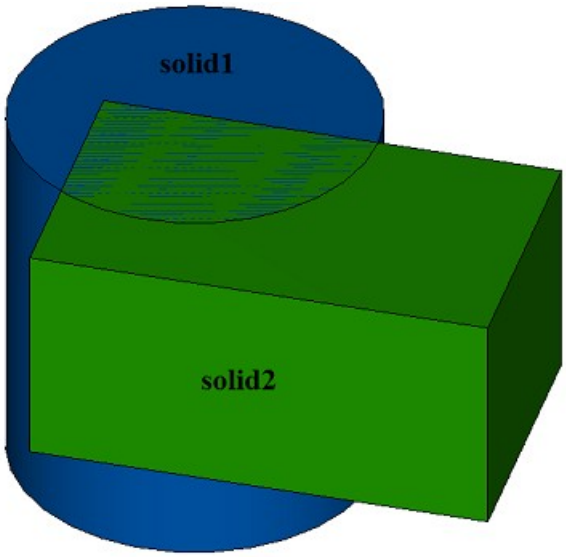
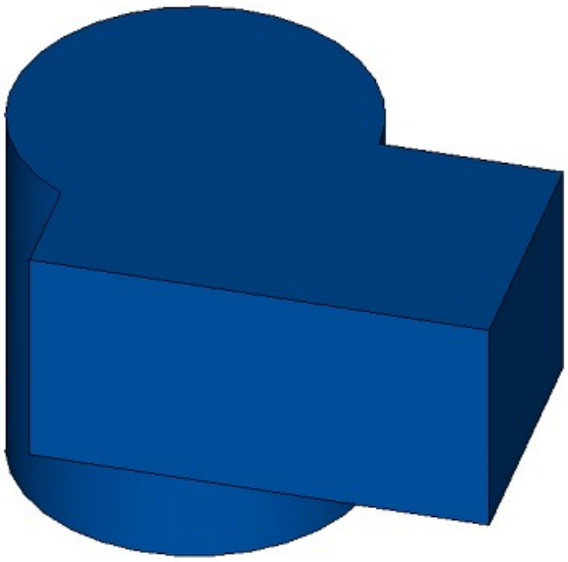


Рис. М.2.1.1.

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

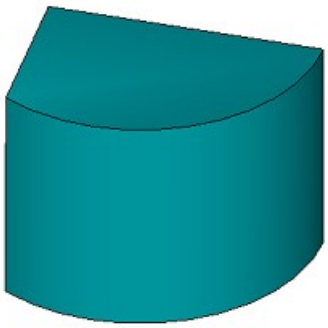


oType = bo_Union

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.

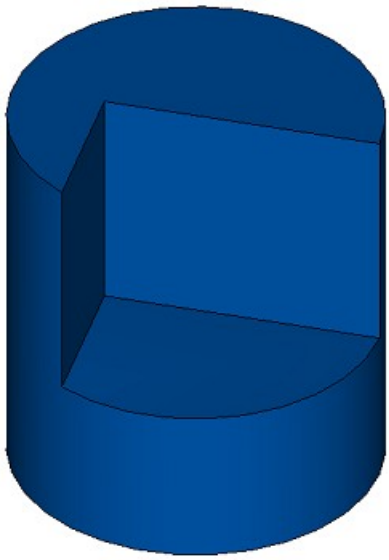


oType = bo_Intersect

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.

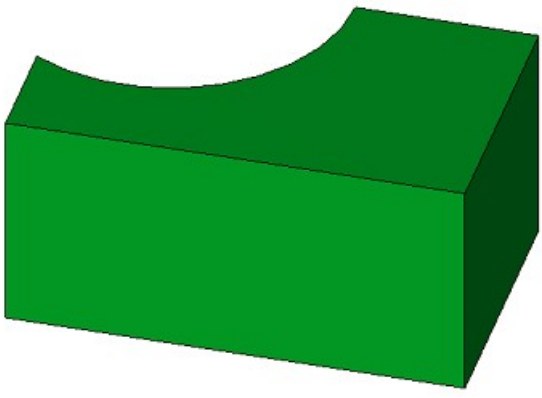


oType = bo_Difference

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.

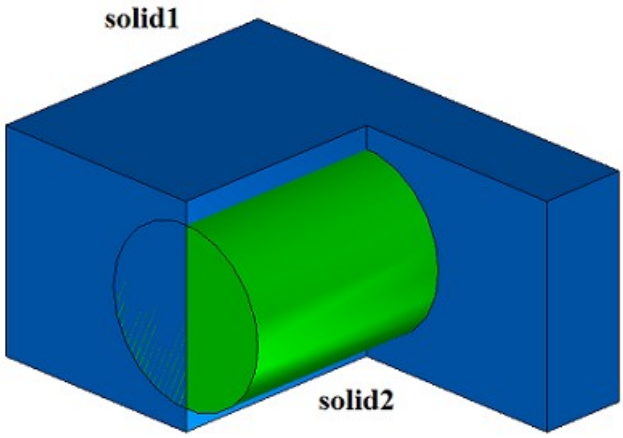


Рис. М.2.1.6.

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.

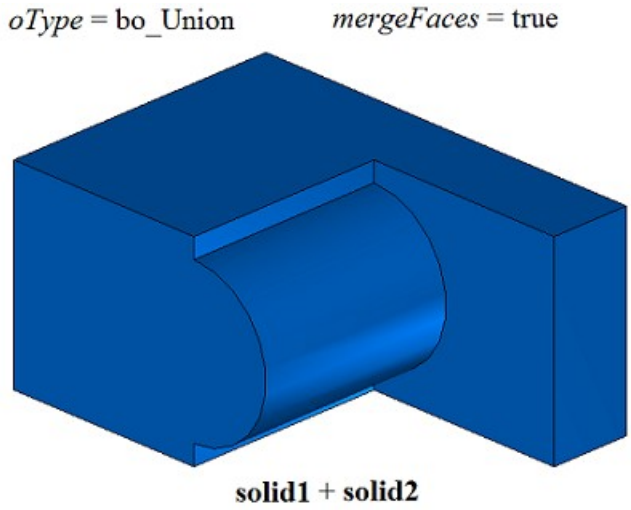


Рис. М.2.1.7.

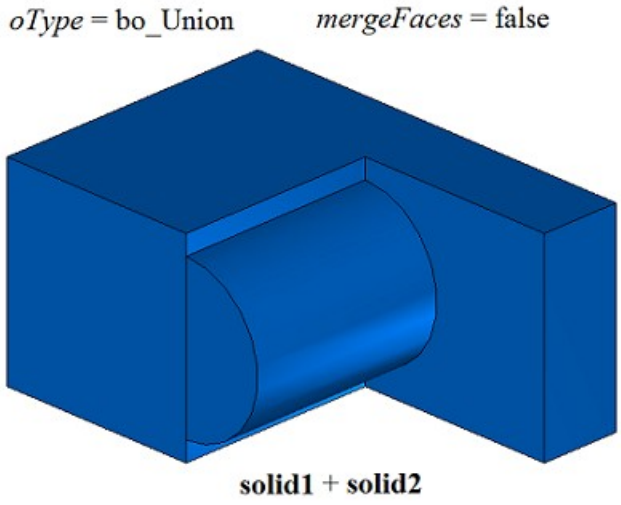


Рис. М.2.1.8.

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.

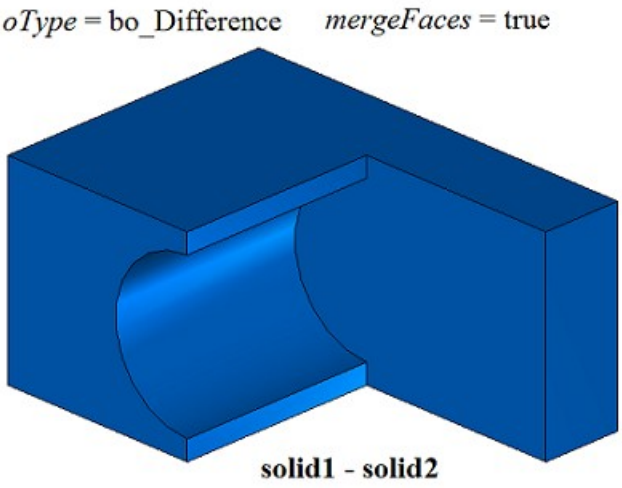


Рис. М.2.1.9.

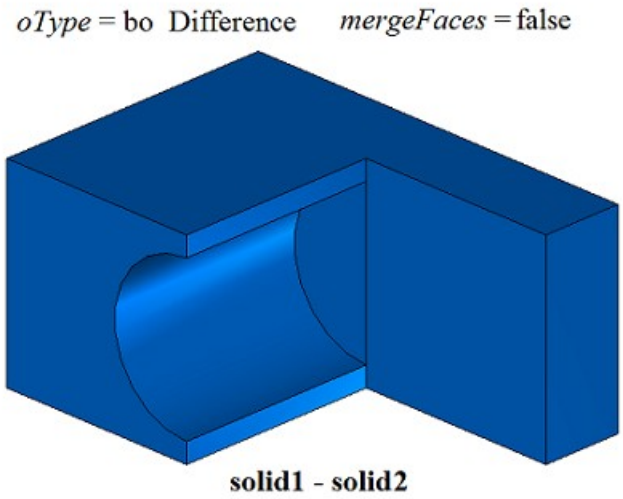


Рис. М.2.1.10.

Метод
MbResultType
BooleanSolid (MbSolid & **solid1**,
MbeCopyMode *sameShell*1,
MbSolid & **solid2**,
MbeCopyMode *sameShell*2,
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 cr_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».

M.2.2. Булева операция над незамкнутыми телами

Метод
MbResultType
BooleanShell (MbSolid & **solid1**,
MbeCopyMode *sameShell*1,
MbSolid & **solid2**,
MbeCopyMode *sameShell*2,
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,
*sameShell*1 – вариант копирования первого тела,
solid2 – второе тело для булевой операции,
*sameShell*2 – 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 *sameShell*1 и *sameShell*2 управляют передачей граней, ребер и вершин от исходных тел **solid1** и **solid2** построенному телу **result**.

Параметры *sameShell*1 и *sameShell*2 могут принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* соответствующее исходное тело (**solid1** или **solid2**) полностью копируется, поэтому построенное тело **result** и соответствующее исходное тело не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и соответствующее исходное тело (**solid1** или **solid2**) будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и соответствующее исходное тело (**solid1** или **solid2**) будут иметь одни и те же вершины, базовые поверхности граней и не затронутые операцией грани. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты соответствующего исходного тела (**solid1** или **solid2**), поэтому после построения соответствующее исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.7.9. Копирование множества граней MbFaceShell.

Параметр OperationType *oType* определяет тип былевой 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**.

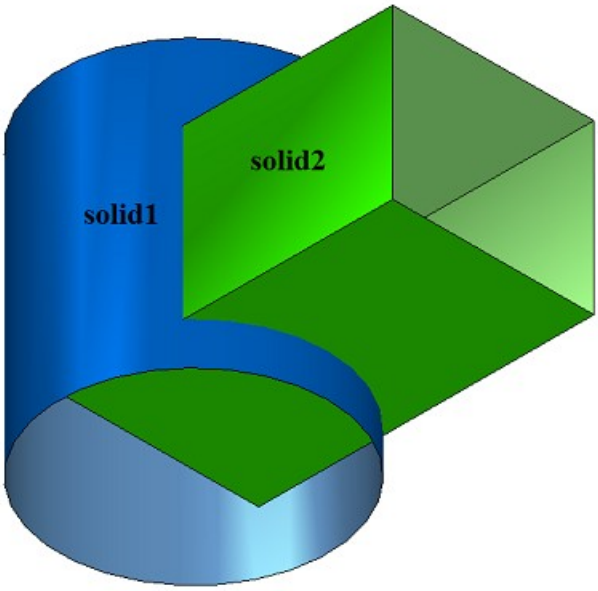
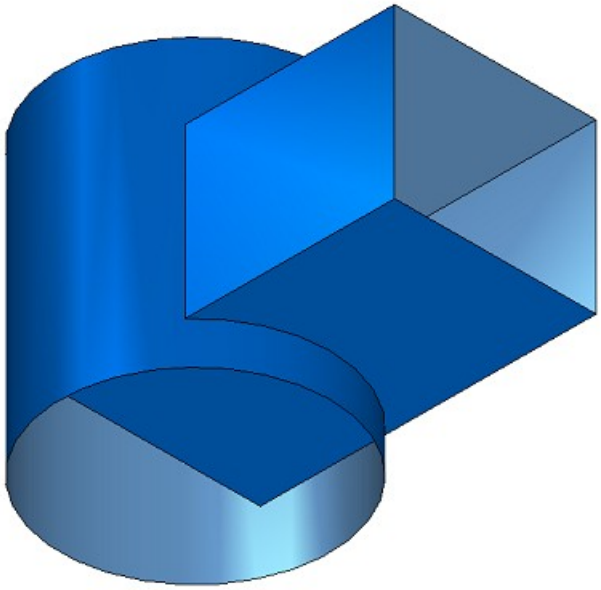


Рис. М.2.2.1.

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

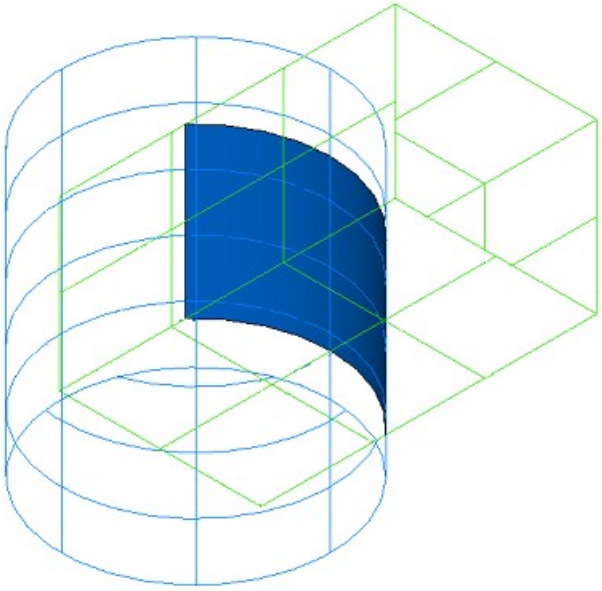


oType = bo_Variety

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.

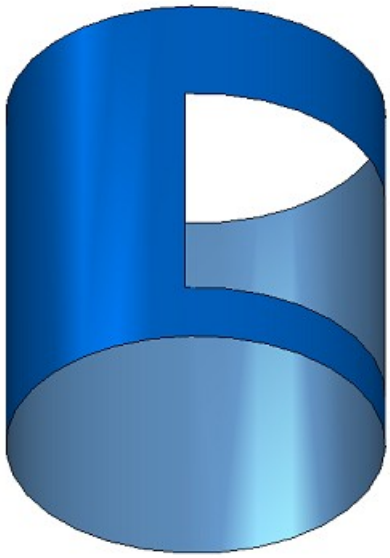


oType = bo_Internal

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.

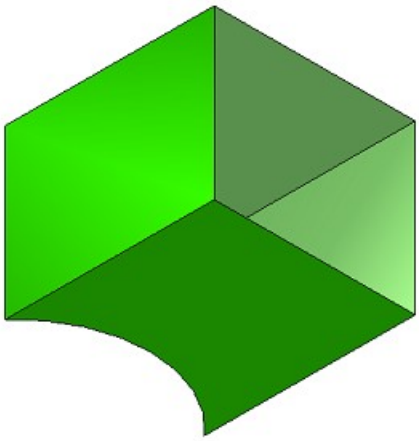


oType = bo_External

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 cr_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**,
MbeCopyMode *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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode 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 их них, связанных с булевыми операциями.

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

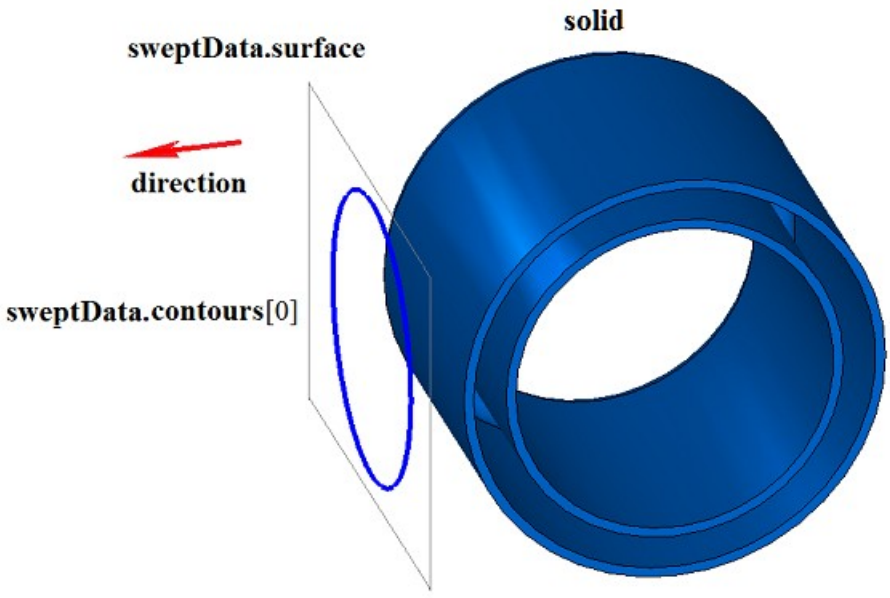


Рис. М.2.3.1.

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.

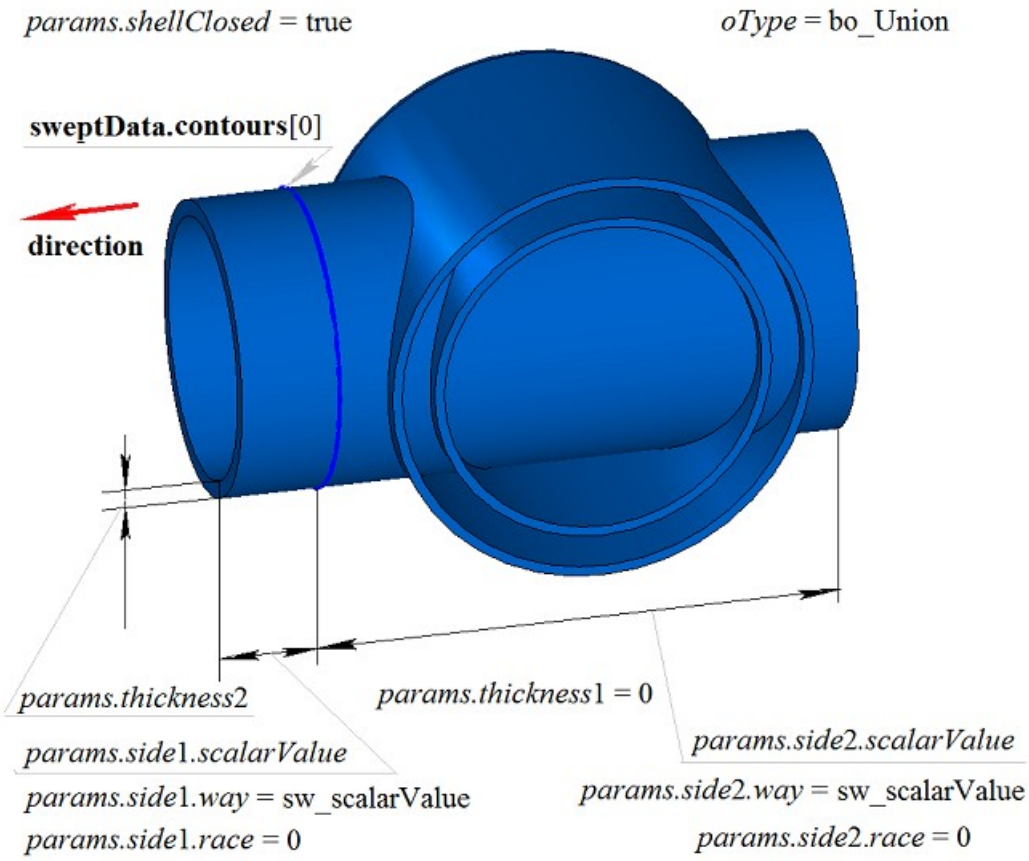


Рис. М.2.3.2.

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.side*2.*way*=sw_shell).

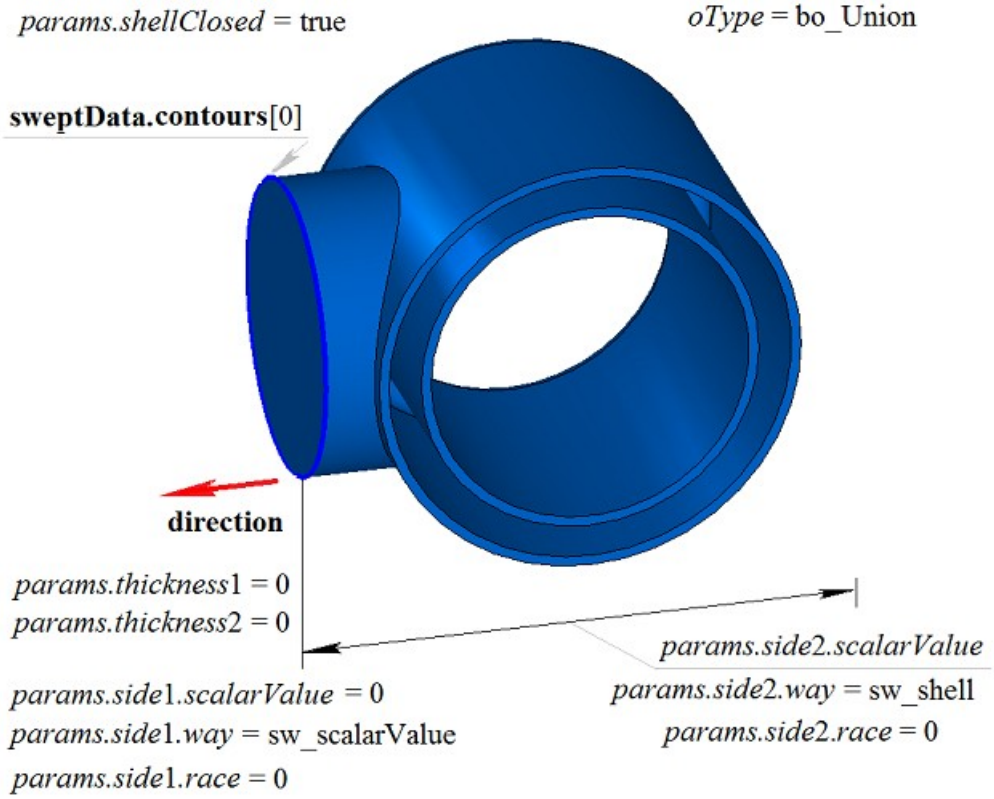


Рис. М.2.3.3.

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.

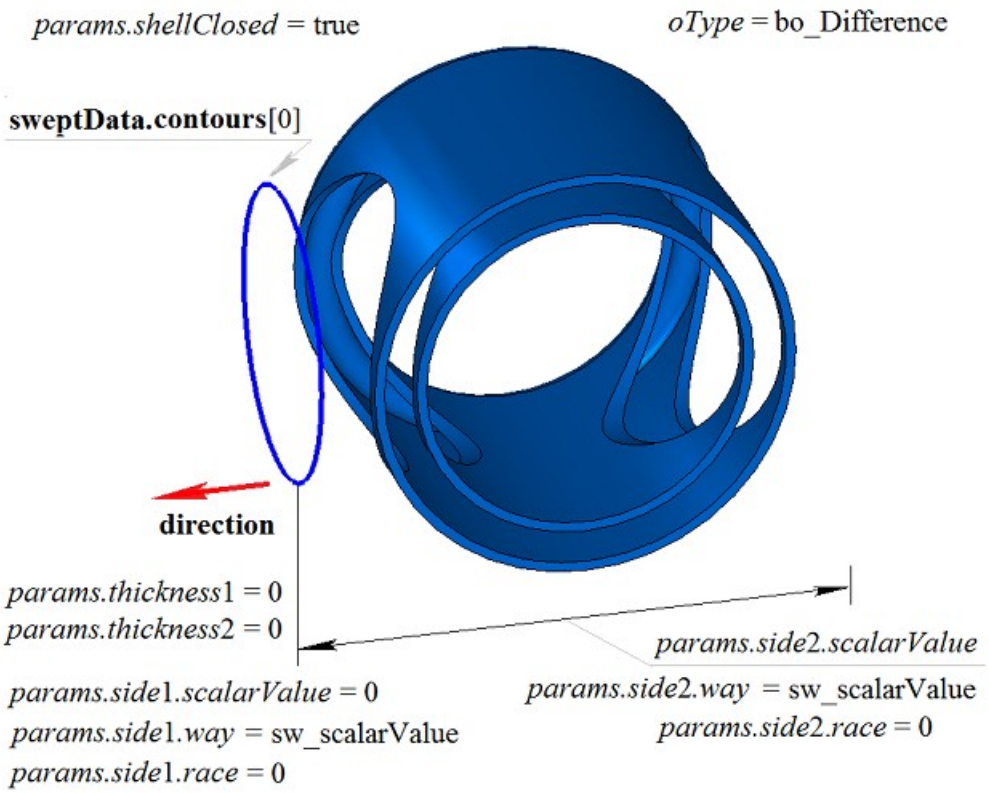


Рис. М.2.3.4.

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.side*2.*way*=sw_shell).

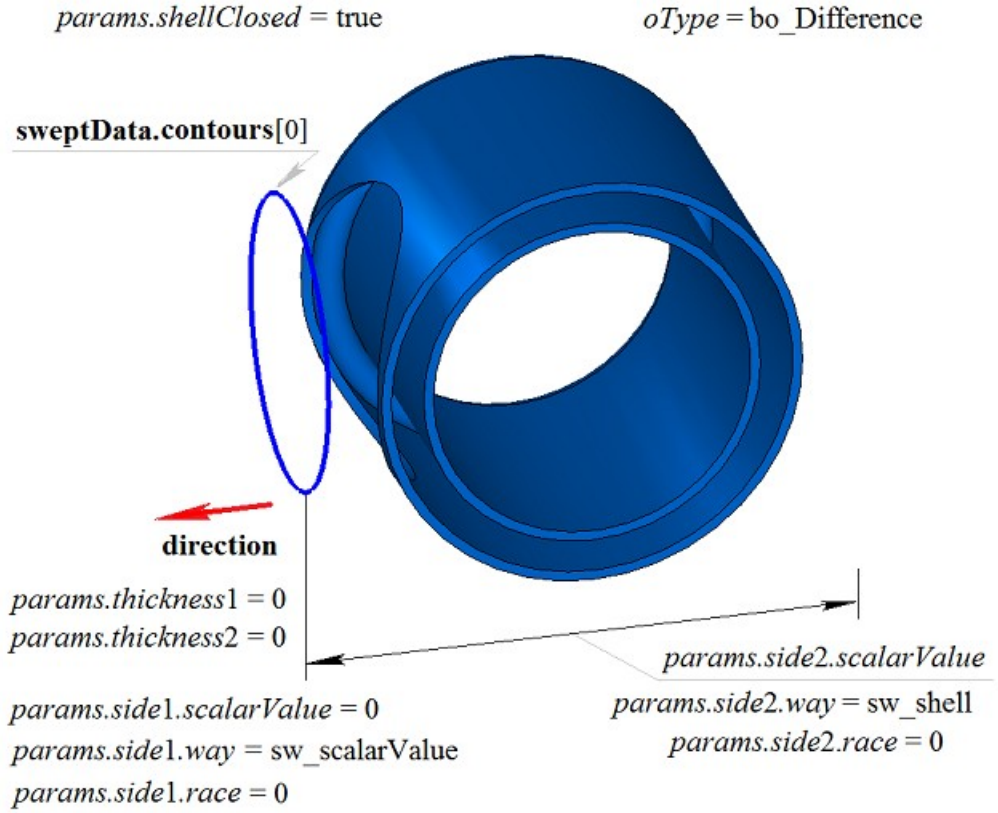


Рис. М.2.3.5.

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.

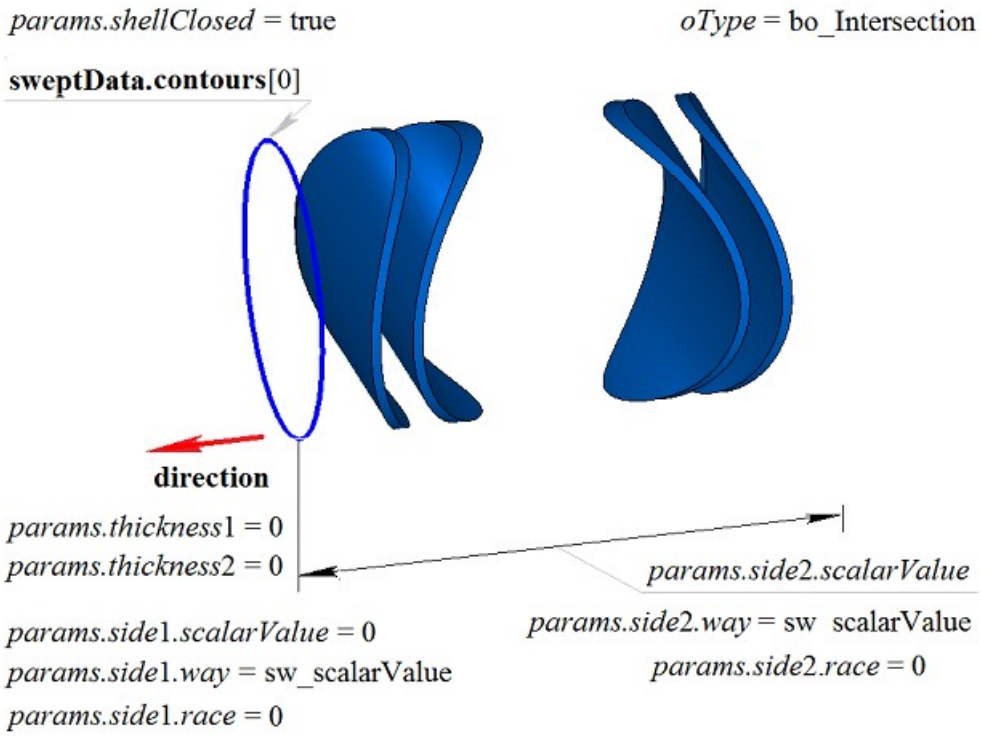


Рис. М.2.3.6.

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».

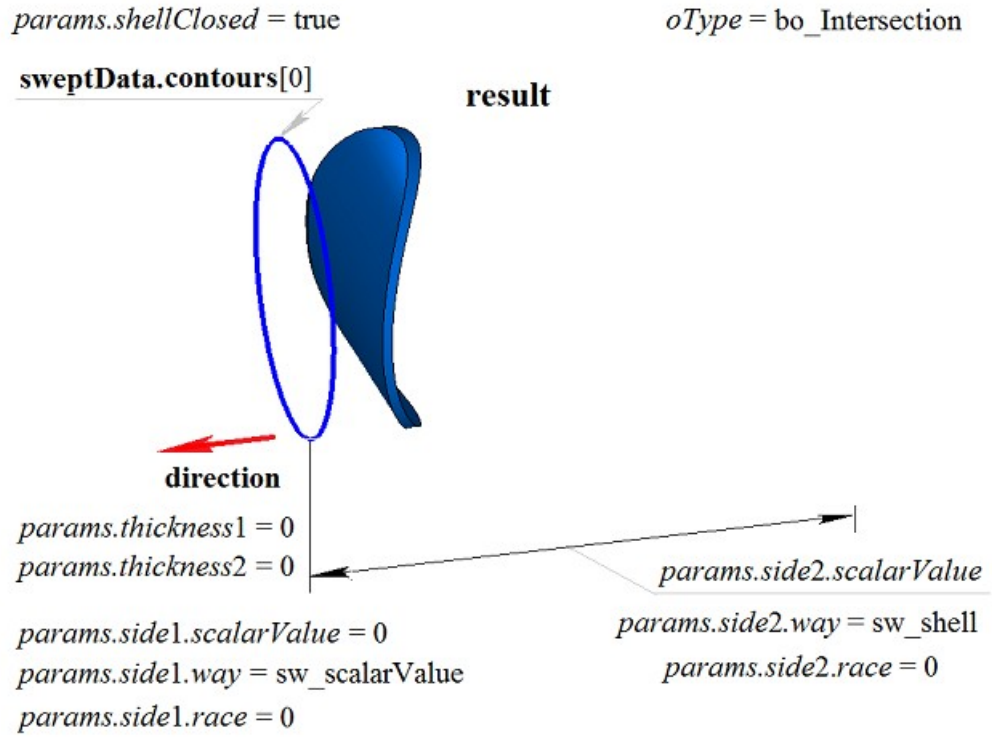


Рис. М.2.3.7.

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

Тестовое приложение test.exe выполняет булеву операцию с построенным телом выдавливания командами меню «Создать->Тело->Приклеиванием к телу->Выдавливания кривой», «Создать->Тело->Вырезанием из тела->Выдавливания кривой» и «Создать->Тело->Пересечением с телом-> Выдавливания 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode 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 их них, связанных с булевыми операциями.

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

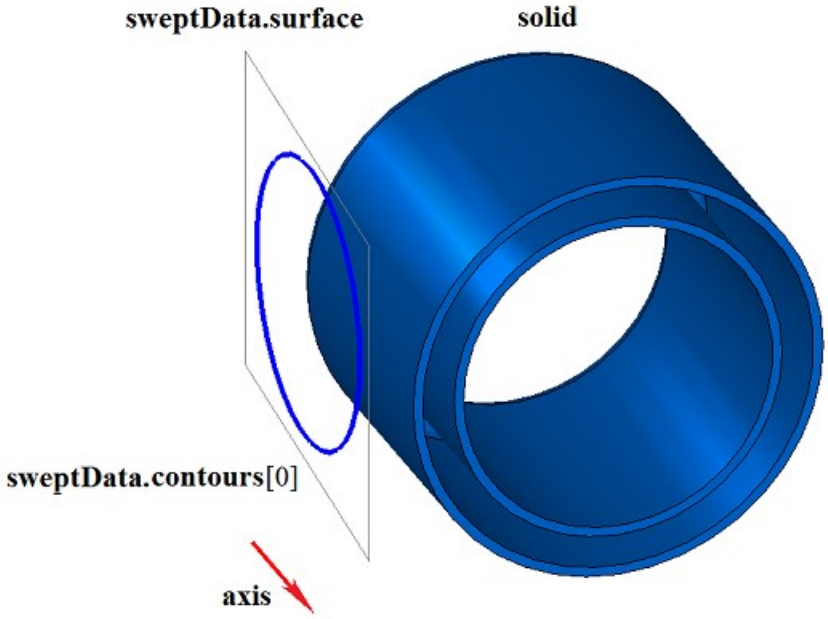


Рис. М.2.4.1.

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.

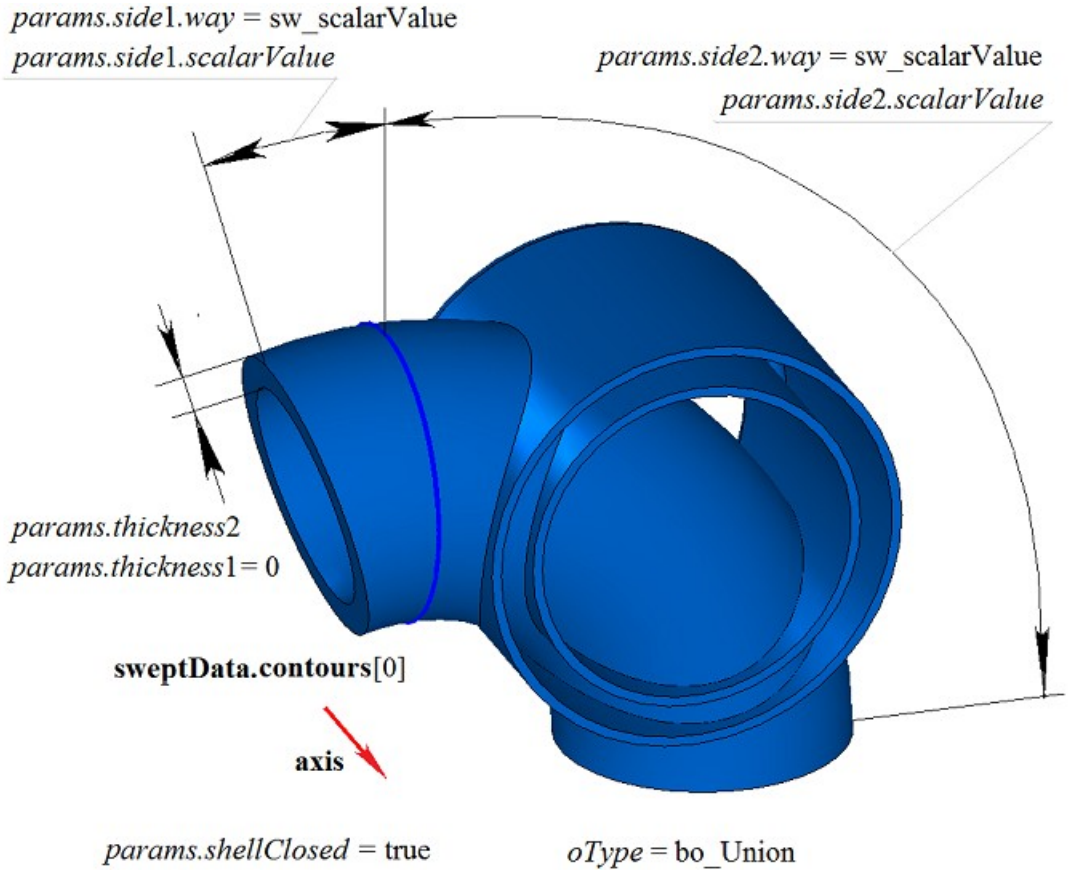


Рис. М.2.4.2.

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.side*2.*way*=sw_surface).

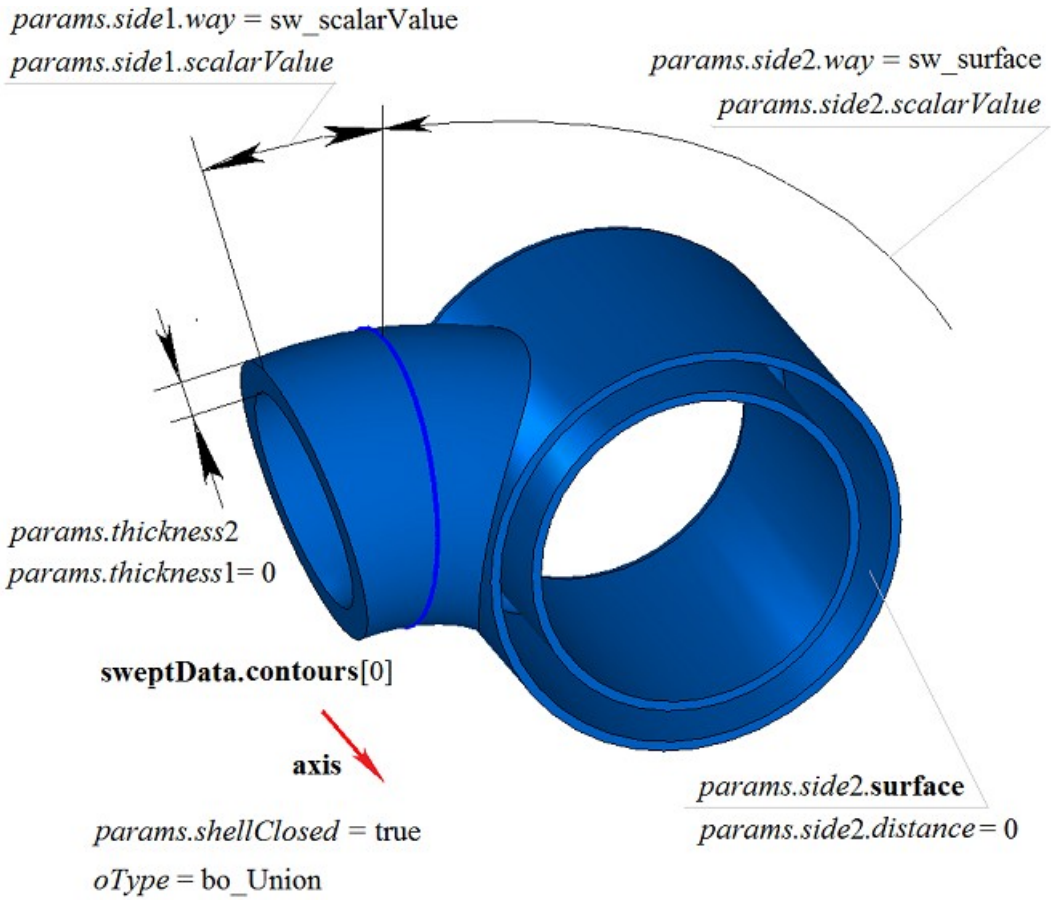


Рис. М.2.4.3.

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.

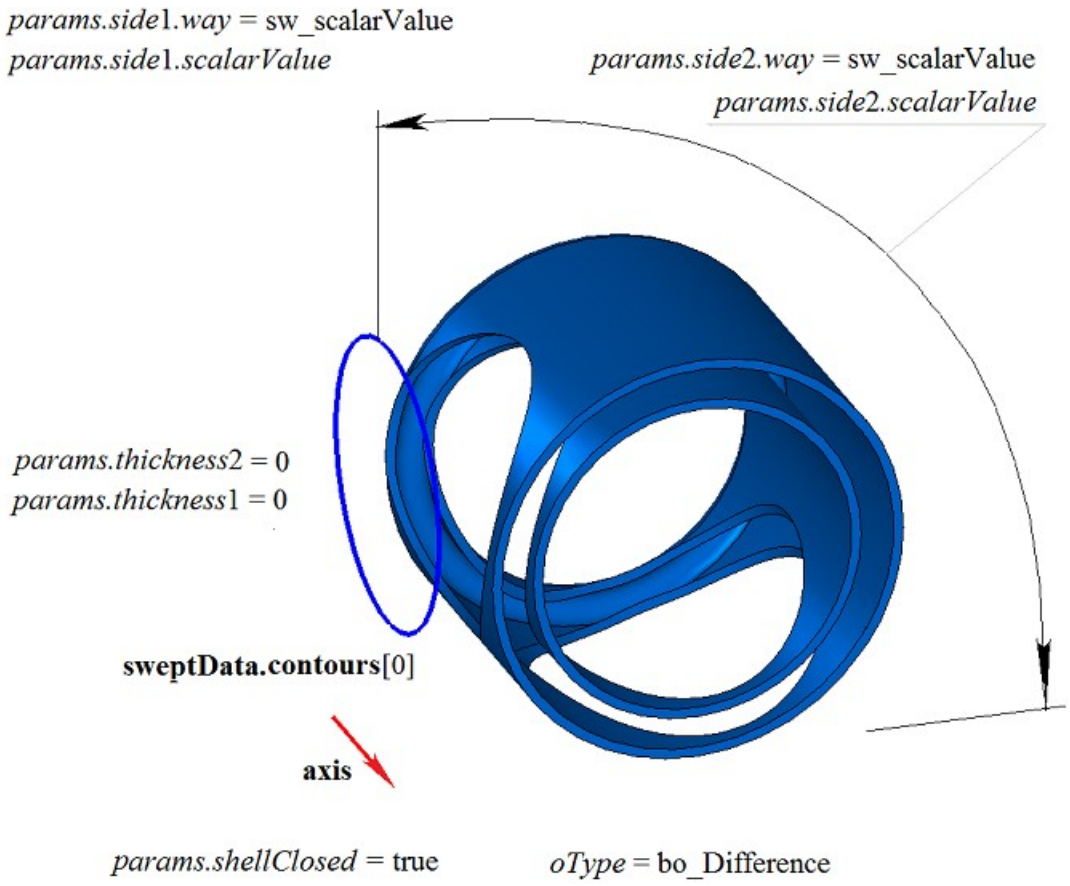


Рис. М.2.4.4.

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.side*2.*way*=sw_surface).

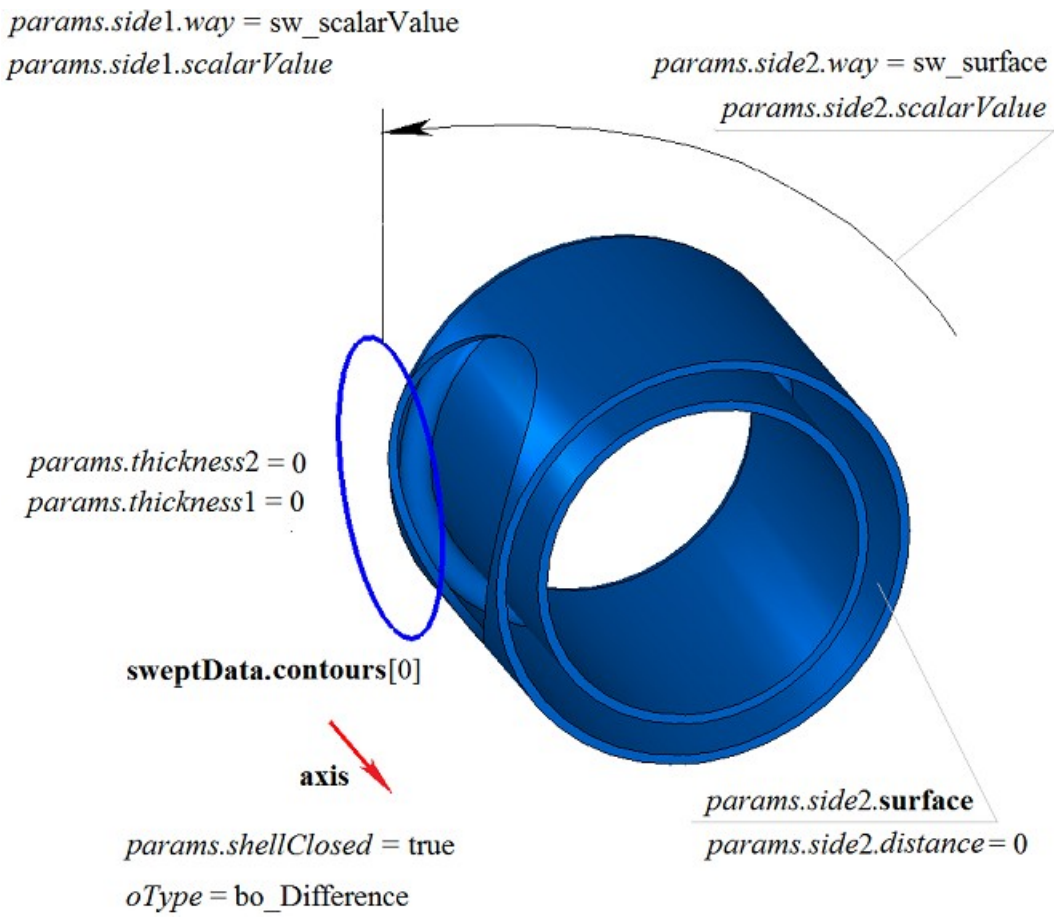


Рис. М.2.4.5.

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.

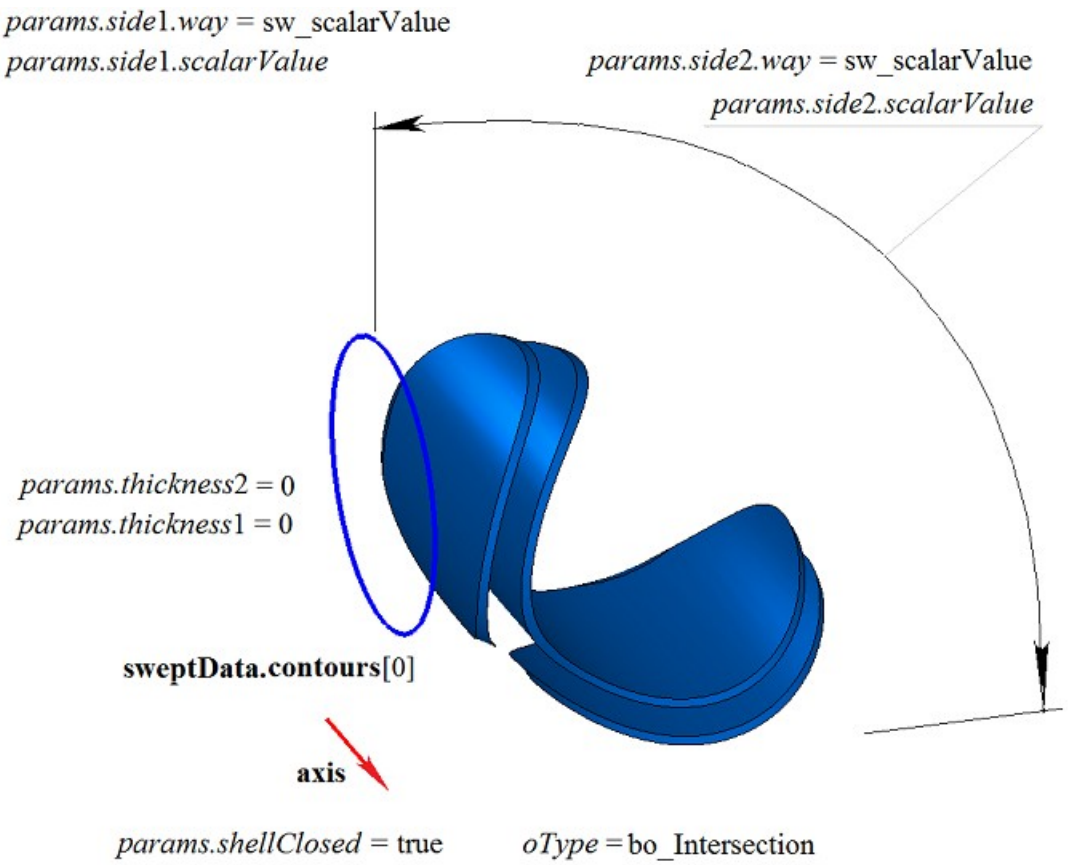


Рис. М.2.4.6.

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.side*2.*way*=sw_surface).

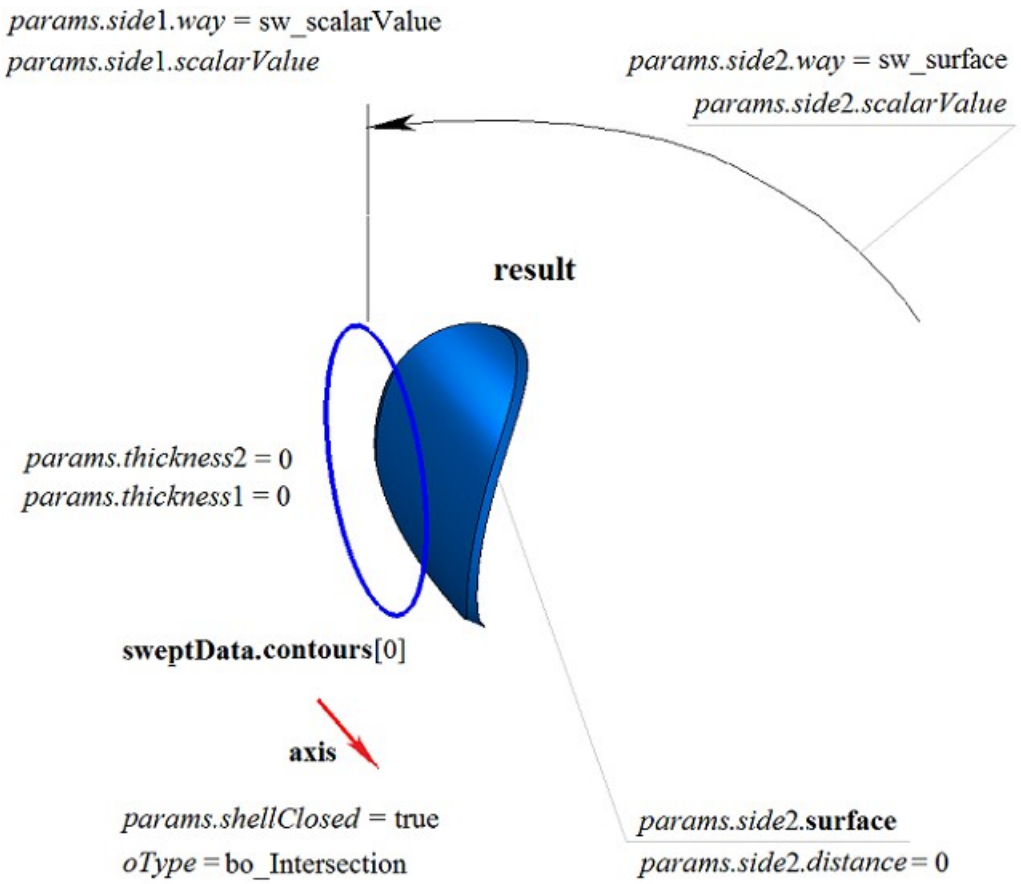


Рис. М.2.4.7.

Метод **EvolutionResult** 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,
                  MbeCopyMode 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 – именователь граней,

cnames – именователи граней тела заметания,
snames – именователь 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.7.9. Копирование множества граней MbFaceShell.

Параметр OperationType *oType* определяет тип былевой 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, cnames 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 их них, связанных с булевыми операциями.

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

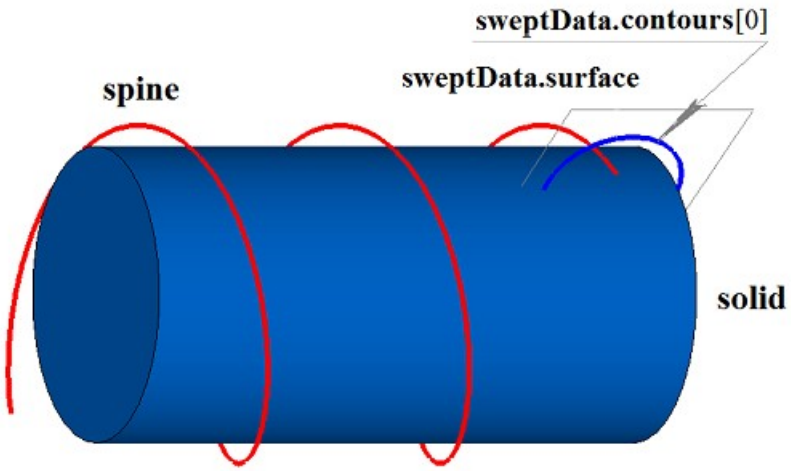


Рис. М.2.5.1.

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.

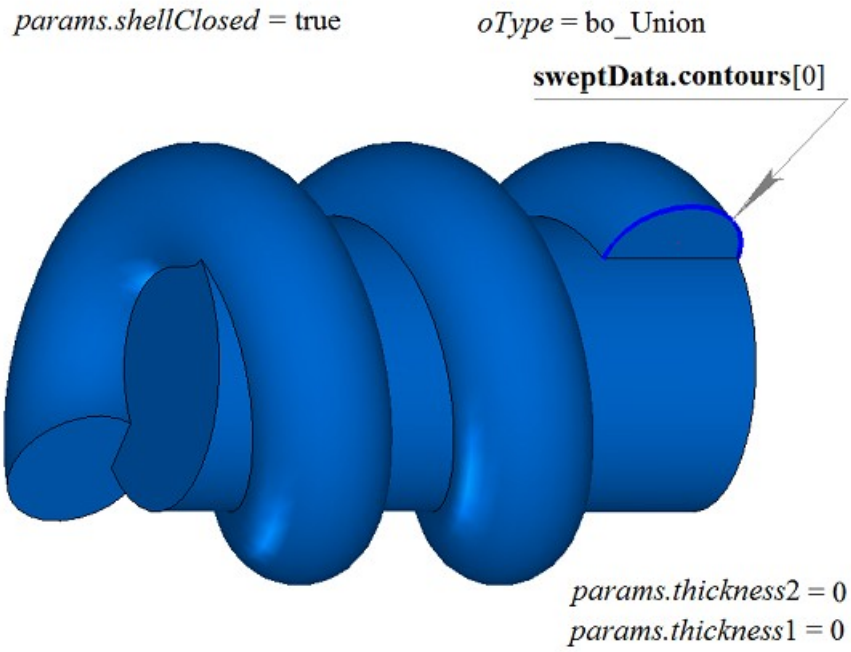


Рис. М.2.5.2.

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.

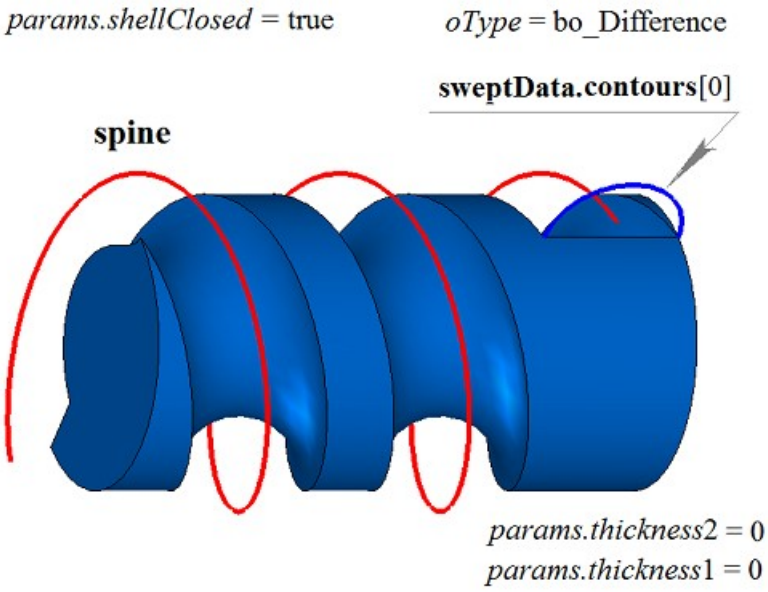


Рис. М.2.5.3.

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.

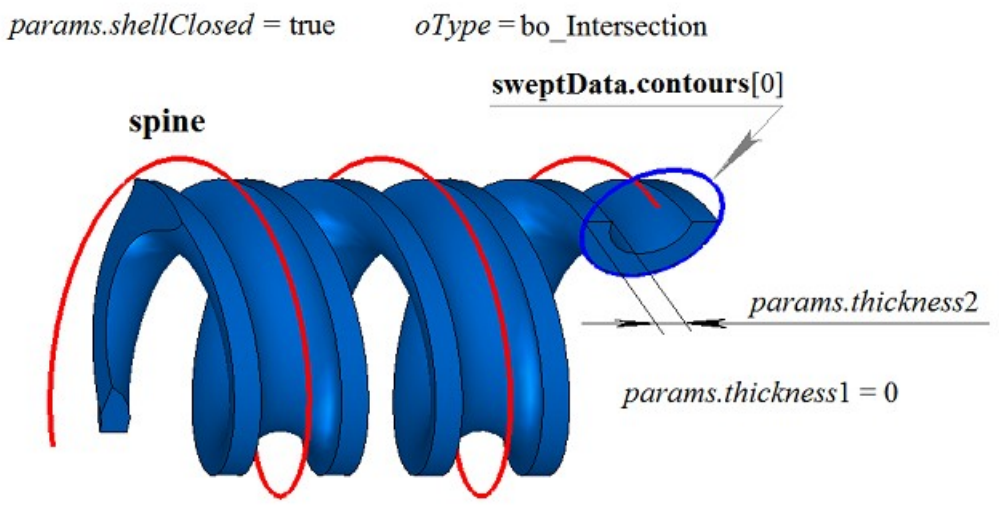


Рис. М.2.5.4.

Метод **EvolutionResult** добавляет в журнал построенного тела строитель MbEvolutionSolid, который содержит все необходимые данные для выполнения операции. Строитель MbEvolutionSolid объявлен в файле cr_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> & **place**,
RPArray<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 – 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.

Метод представляет собой последовательное объединение двух методов: метода **LoftedSolid**, выполняющего построение тела по плоским сечениям **contours** на плоскостях **place** 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode 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, cnames 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 их них, связанных с булевыми операциями.

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

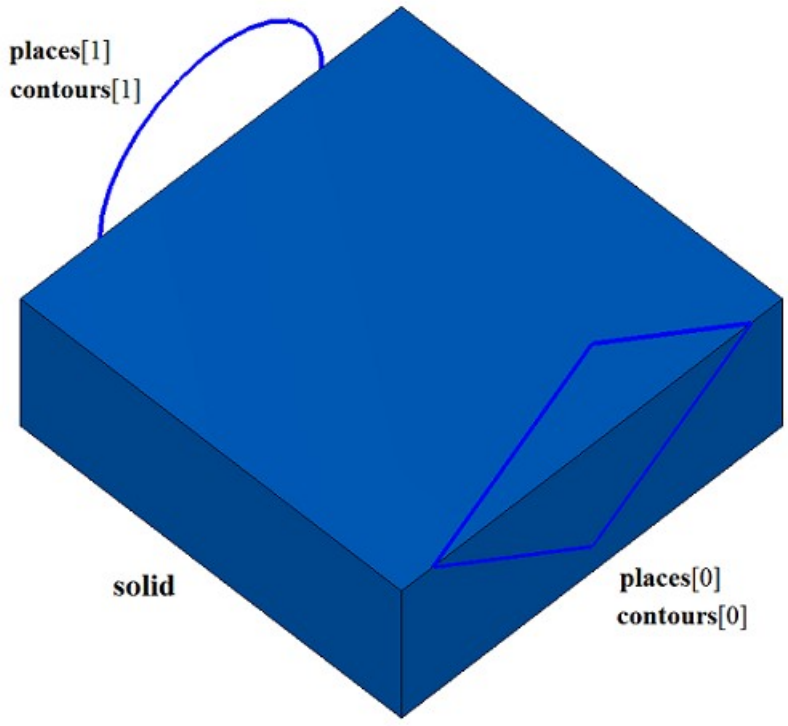


Рис. М.2.6.1.

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.

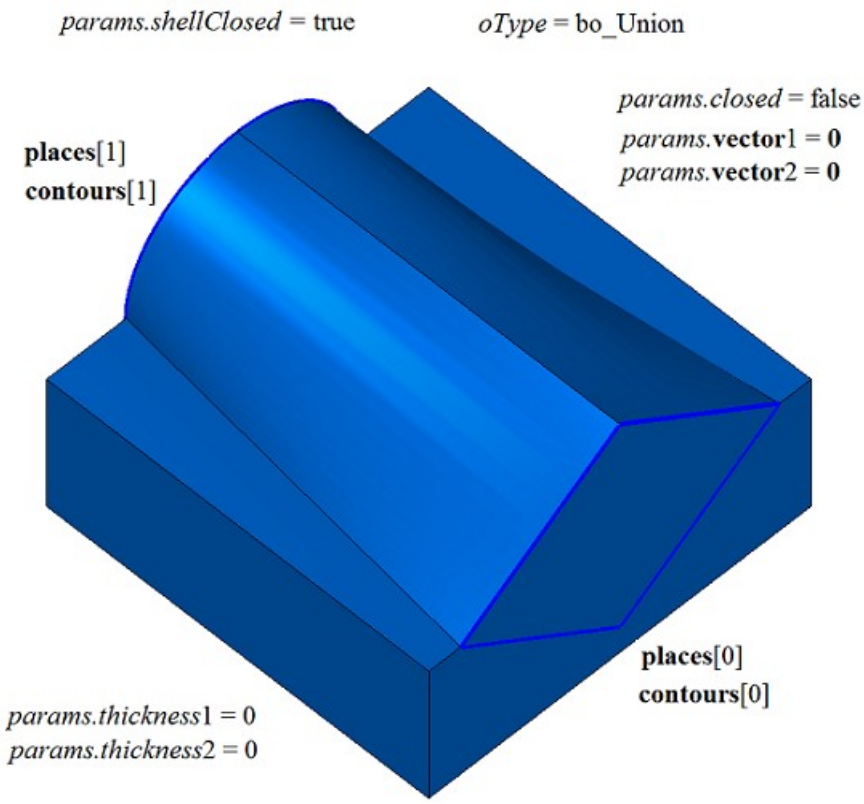


Рис. М.2.6.2.

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.

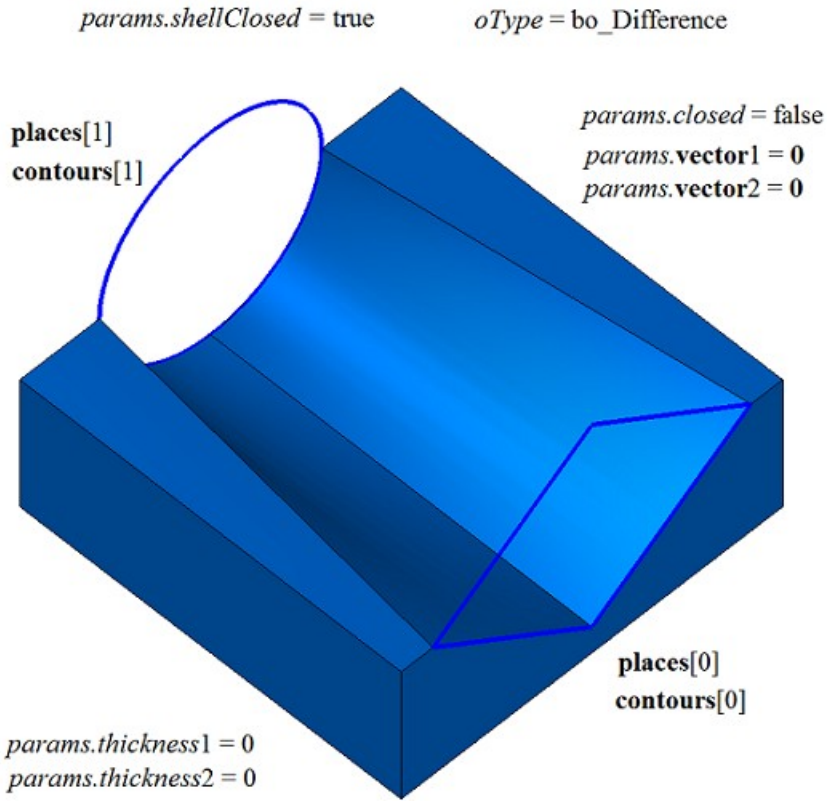


Рис. М.2.6.3.

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.

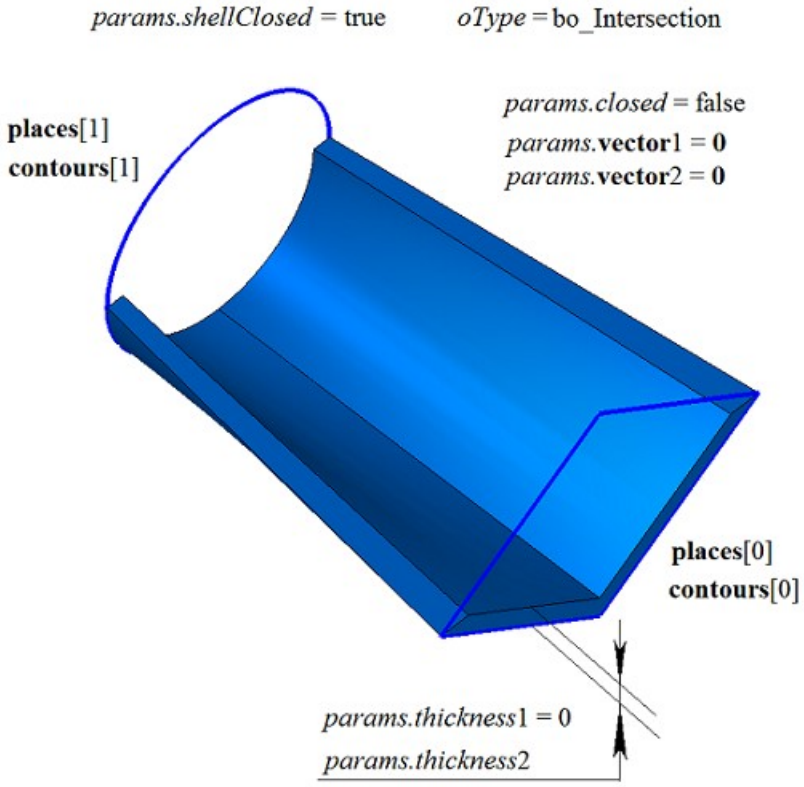


Рис. М.2.6.4.

Метод **LoftedResult** добавляет в журнал построенного тела строитель MbLoftedSolid, который содержит все необходимые данные для выполнения операции. Строитель MbLoftedSolid объявлен в файле cr_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 = –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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode 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 над поверхностью, при *par*=–1 сохраняется часть тела, расположенная под поверхностью. 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**.

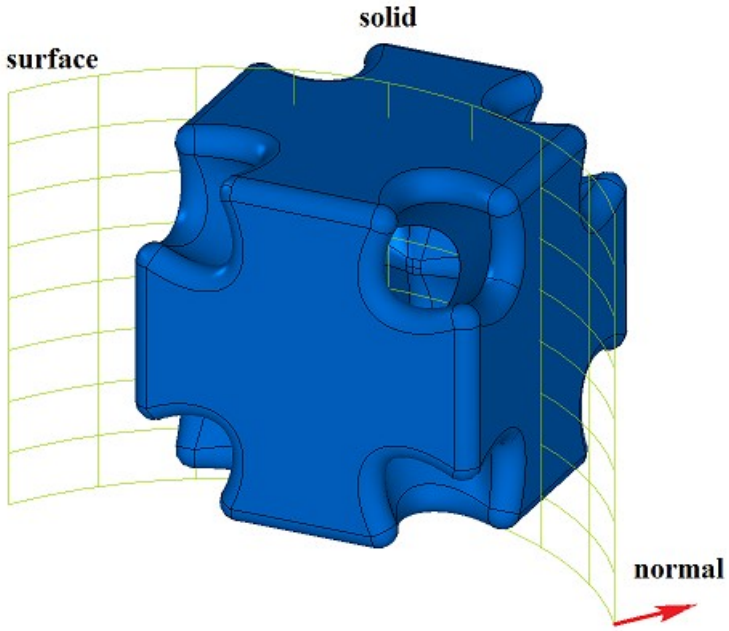
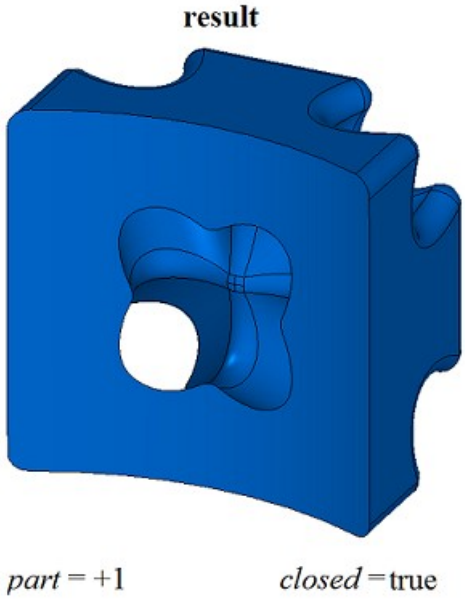


Рис. М.2.7.1.

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.



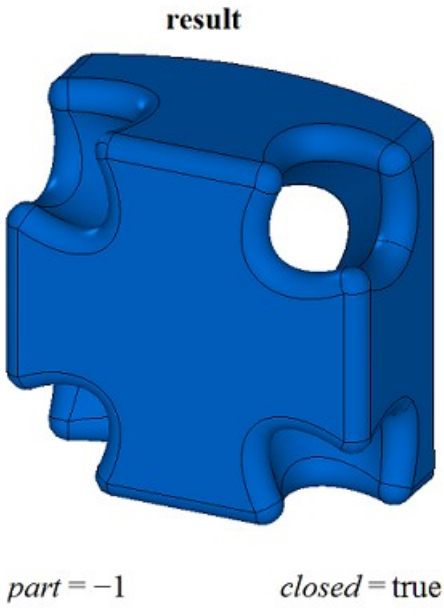


Рис. М.2.7.2. *Рис. М.2.7.3.*

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

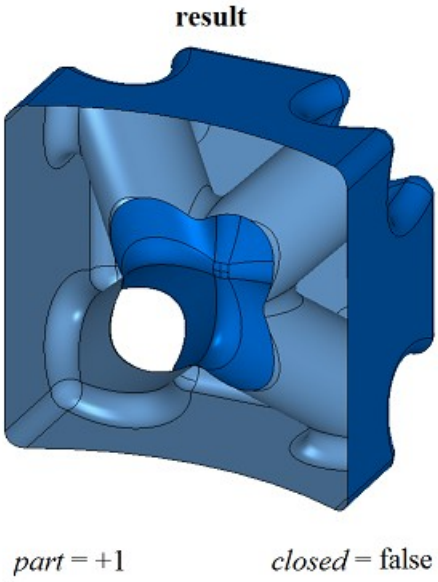


Рис. М.2.7.4.

Метод **SolidCutting** добавляет в журнал построенного тела строитель MbCuttingSolid, который содержит все необходимые данные для выполнения операции. Строитель MbCuttingSolid объявлен в файле cr_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 = –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 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode 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 права от контура, при *par*=–1 сохраняется часть тела, расположенная слева от контура, если смотреть вдоль контура навстречу оси **place.axisZ**. 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.2.8.1 показаны исходное тело **solid**, режущий контур **contour** и плоскость XY локальной системы координат **place**.

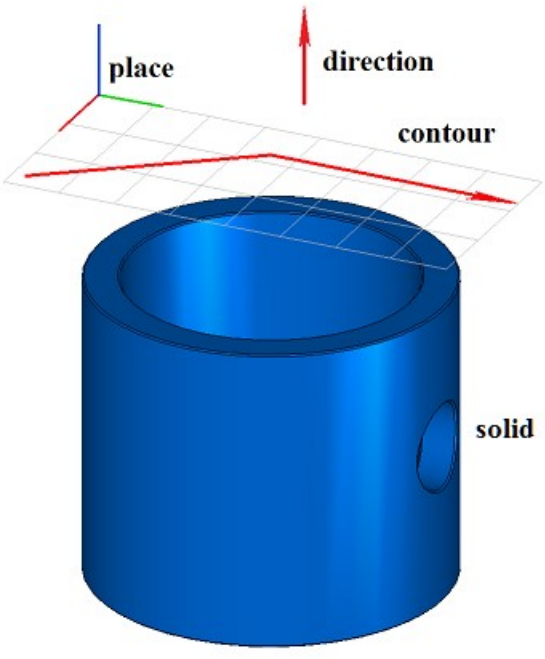


Рис. М.2.8.1.

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.

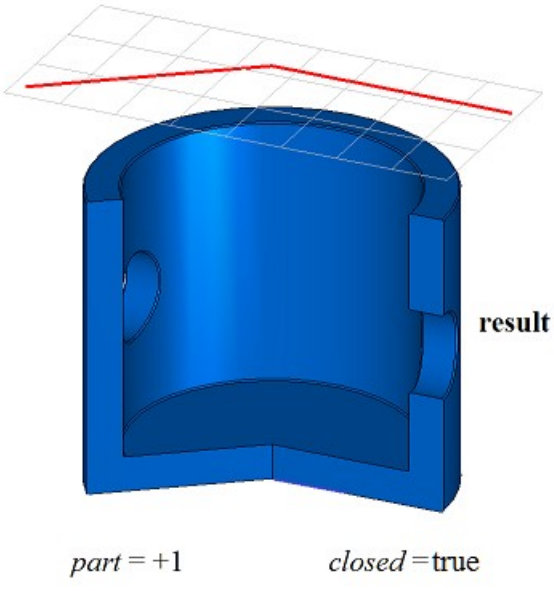


Рис. М.2.8.2.

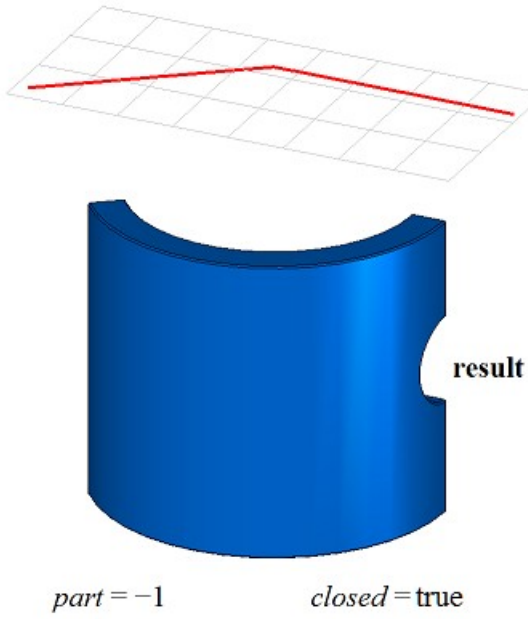


Рис. М.2.8.3.

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

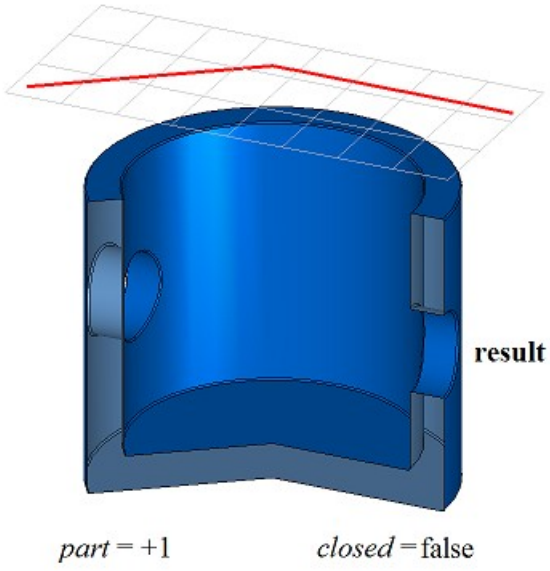


Рис. М.2.8.4.

Метод **SolidCutting** добавляет в журнал построенного тела строитель MbCuttingSolid, который содержит все необходимые данные для выполнения операции. Строитель MbCuttingSolid объявлен в файле cr_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 – именователь 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 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.7.9. Копирование множества граней MbFaceShell.

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

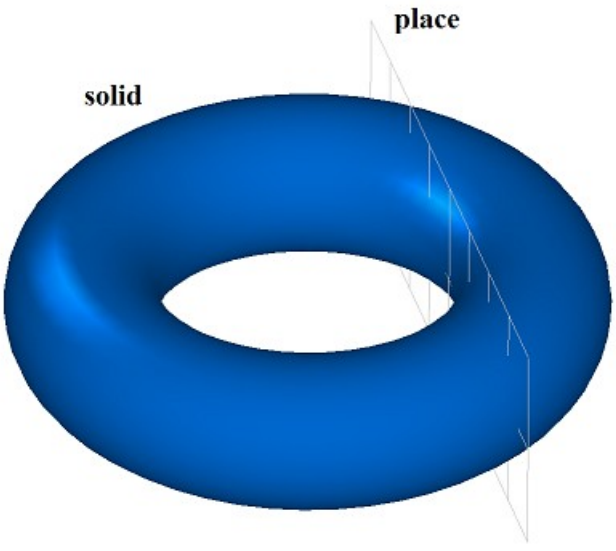


Рис. М.2.9.1.

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

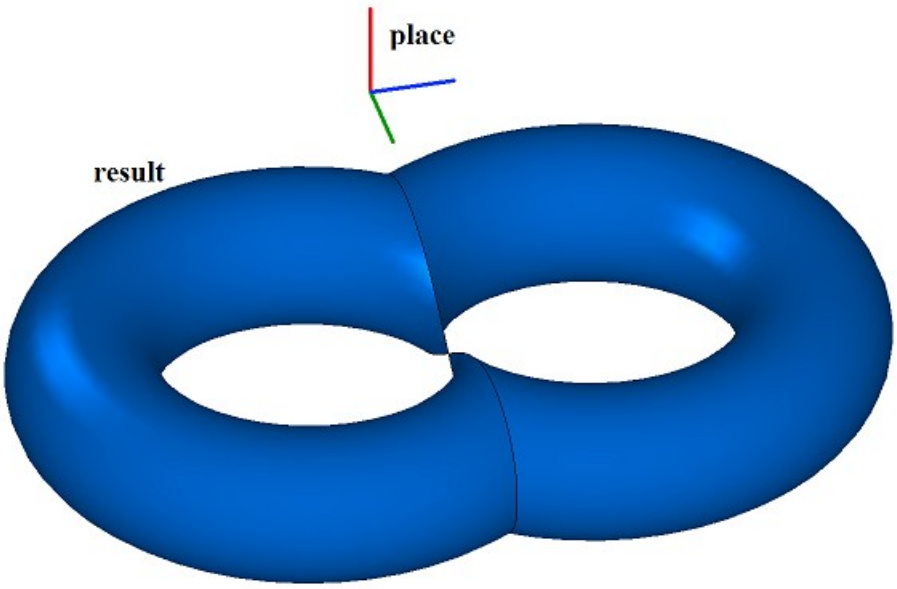


Рис. М.2.9.2.

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

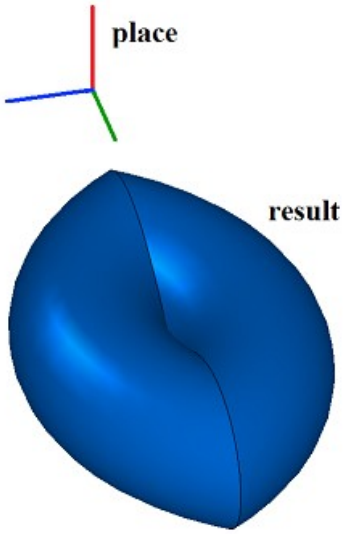


Рис. М.2.9.3.

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 cr_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*,
RPArray<MbCurveEdge> & **edges**,
RPArray<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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.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.

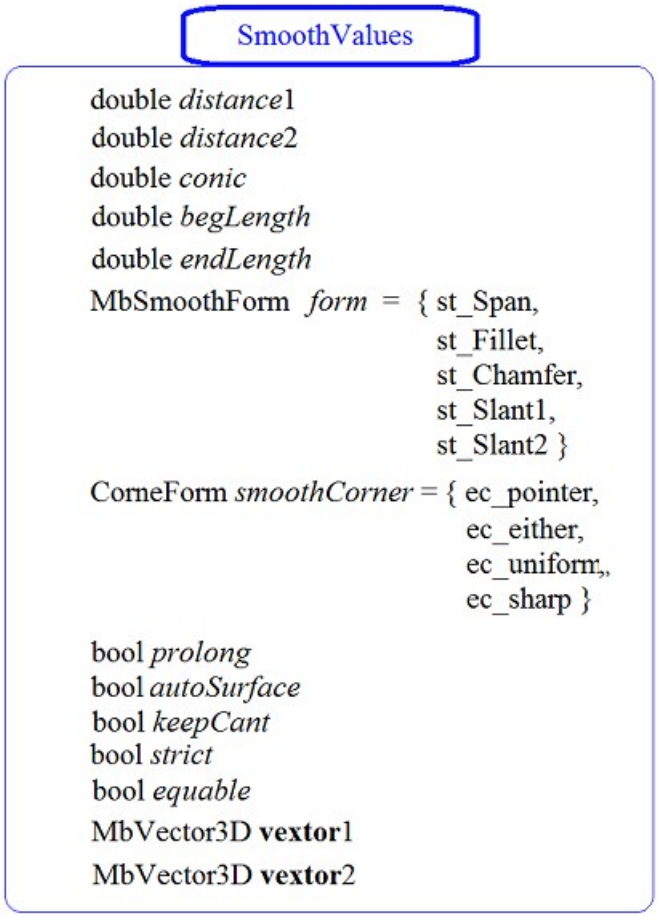


Рис. М.2.10.1.

Входной параметр *params* содержит следующие данные:
*distance*1 – первый радиус скругления,
*distance*2 – 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),
MbeSmoothForm *form* – тип сопряжения,
smoothCorner – способ скругления чемоданных 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 *distance*1 и *distance*2. На рис. М.2.10.2 приведено скругление с заданными радиусами ребра, соединяющего две цилиндрические поверхности.

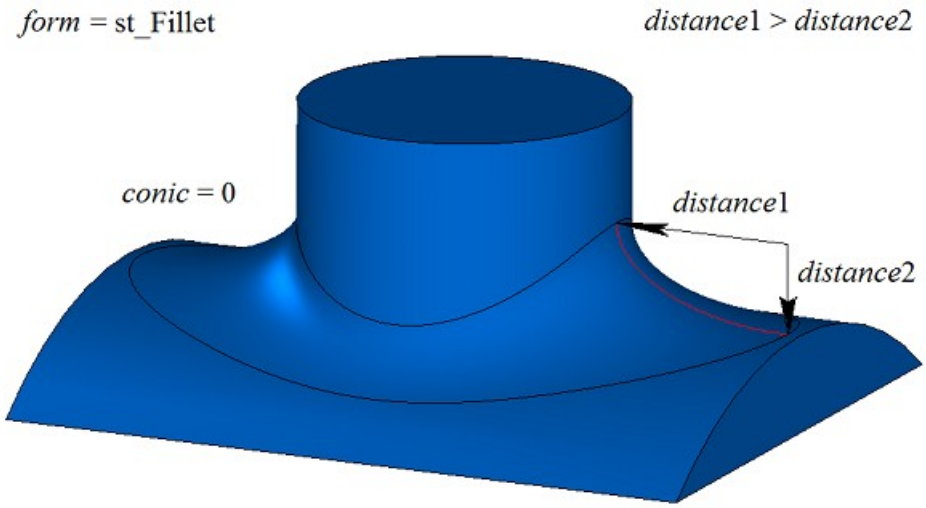


Рис. М.2.10.2.

При *distance*1=*distance*2 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.

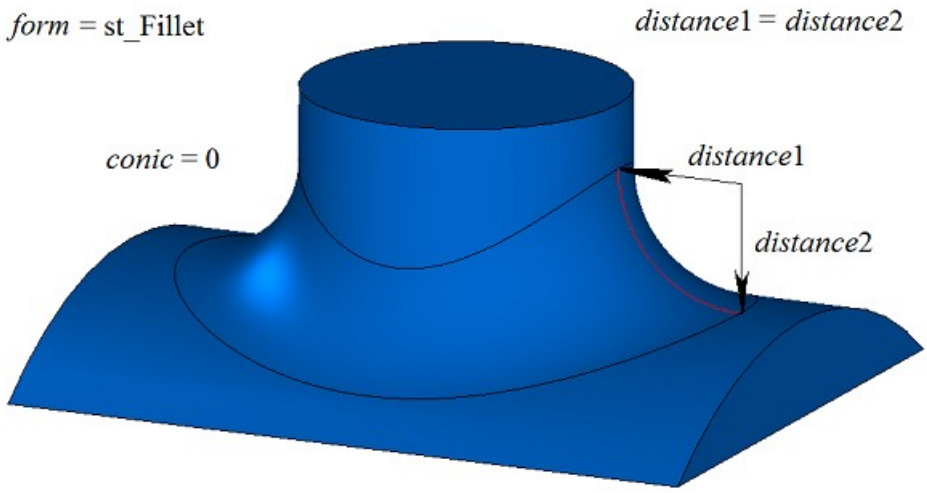


Рис. М.2.10.3.

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.

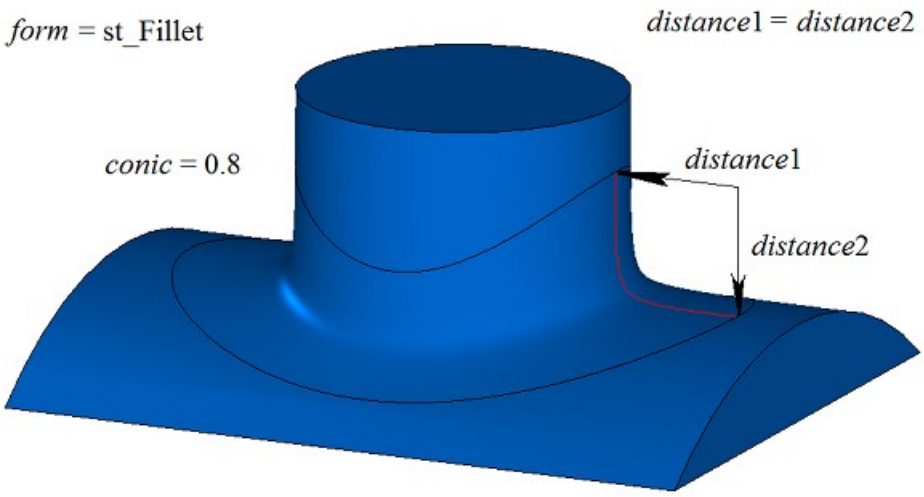


Рис. М.2.10.4.

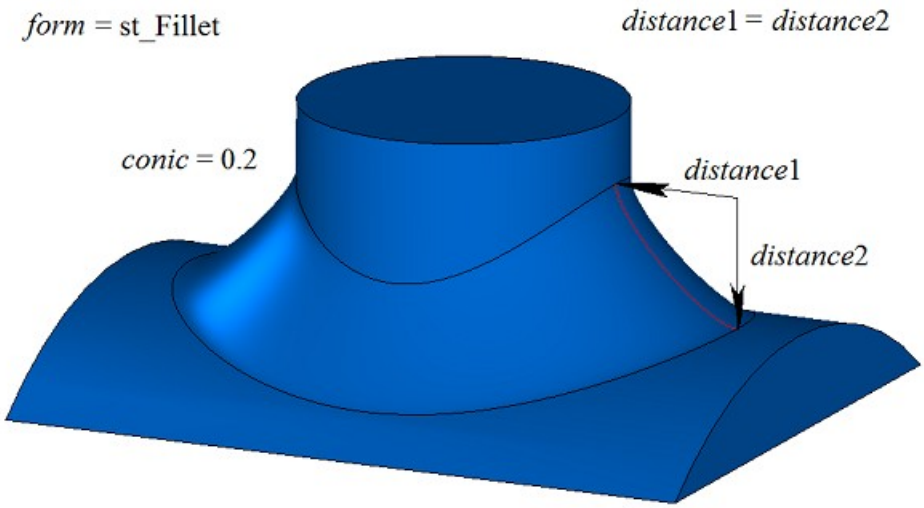


Рис. М.2.10.5.

При значении *form*=st_Span рассматриваемый метод строит поверхность скругления с заданной хордой. Параметры *distance*1 и *distance*2 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 *distance*1 и *distance*2 равны хорде дуги окружности. На рис. М.2.10.6 приведено скругление с заданной хордой ребра, соединяющего две цилиндрические поверхности.

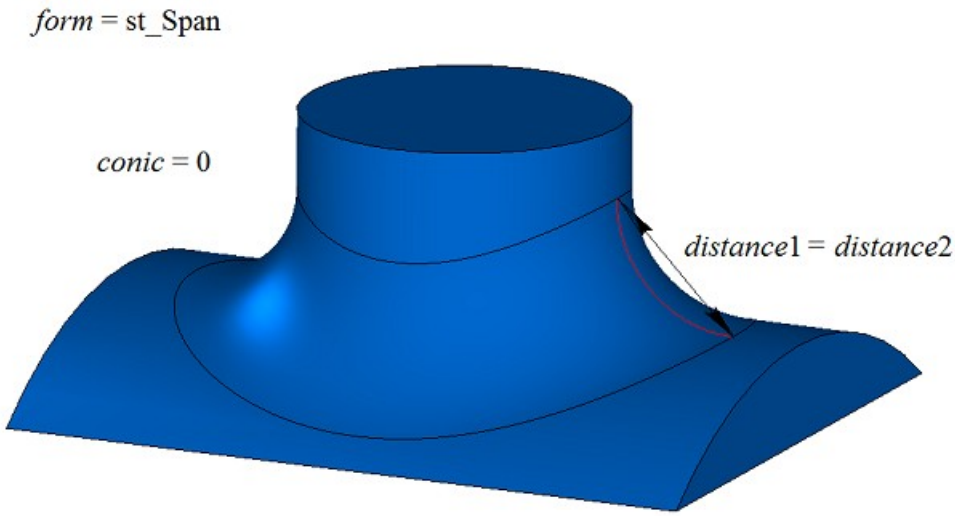


Рис. М.2.10.6.

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

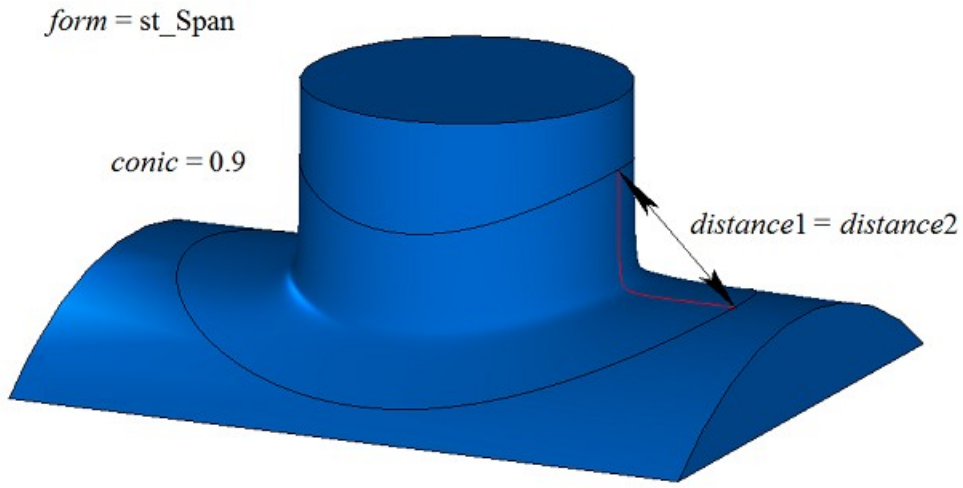


Рис. М.2.10.7.

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.

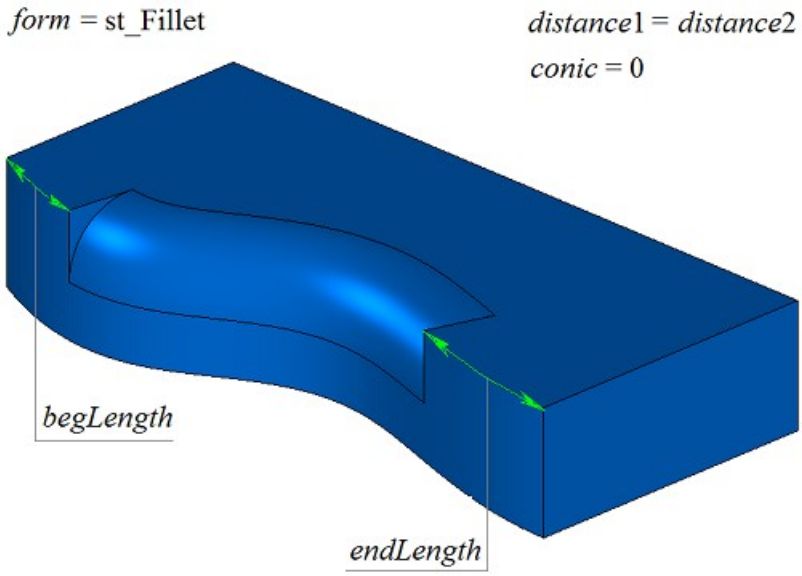


Рис. М.2.10.8.

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.

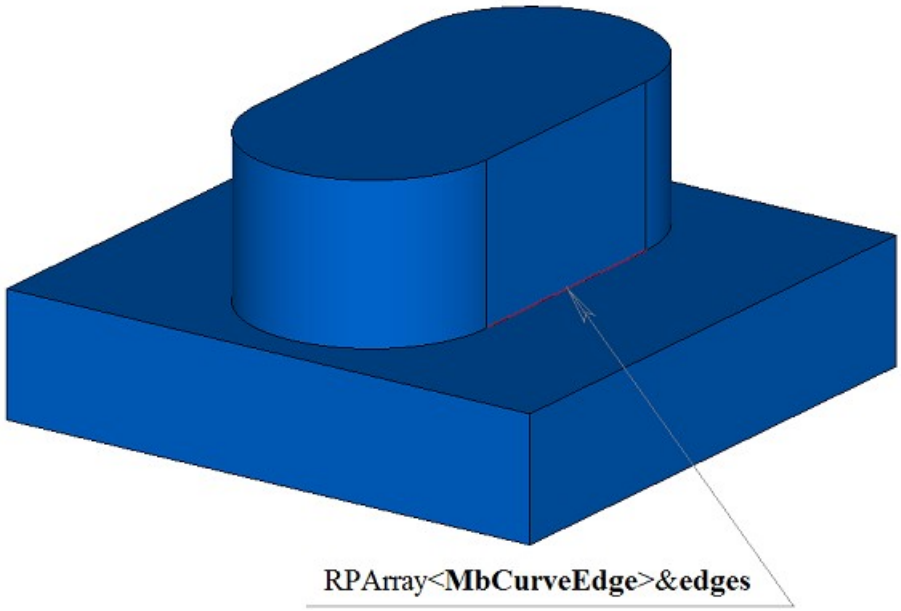


Рис. М.2.10.9.

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.

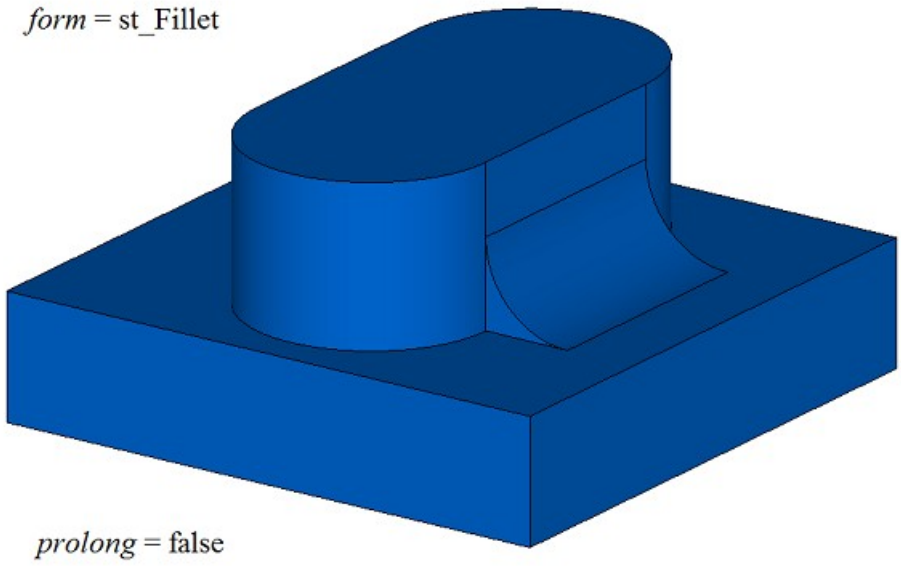


Рис. М.2.10.10.

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

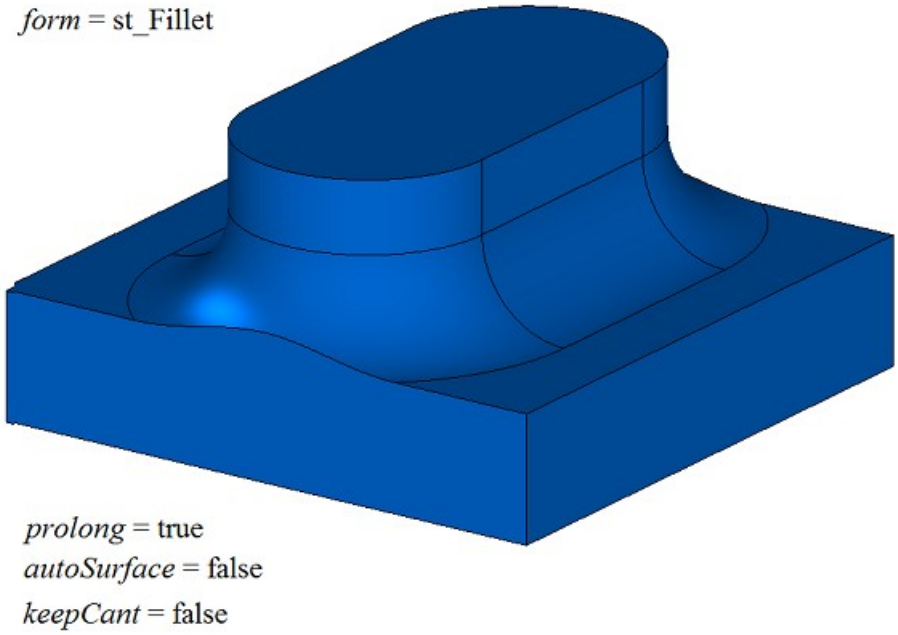


Рис. М.2.10.11.

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.

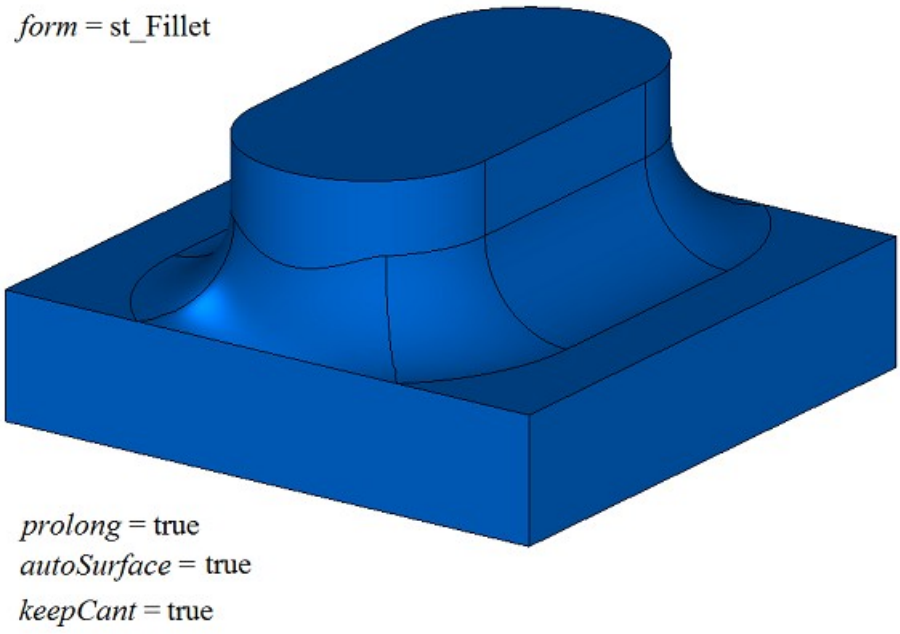


Рис. М.2.10.12.

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.

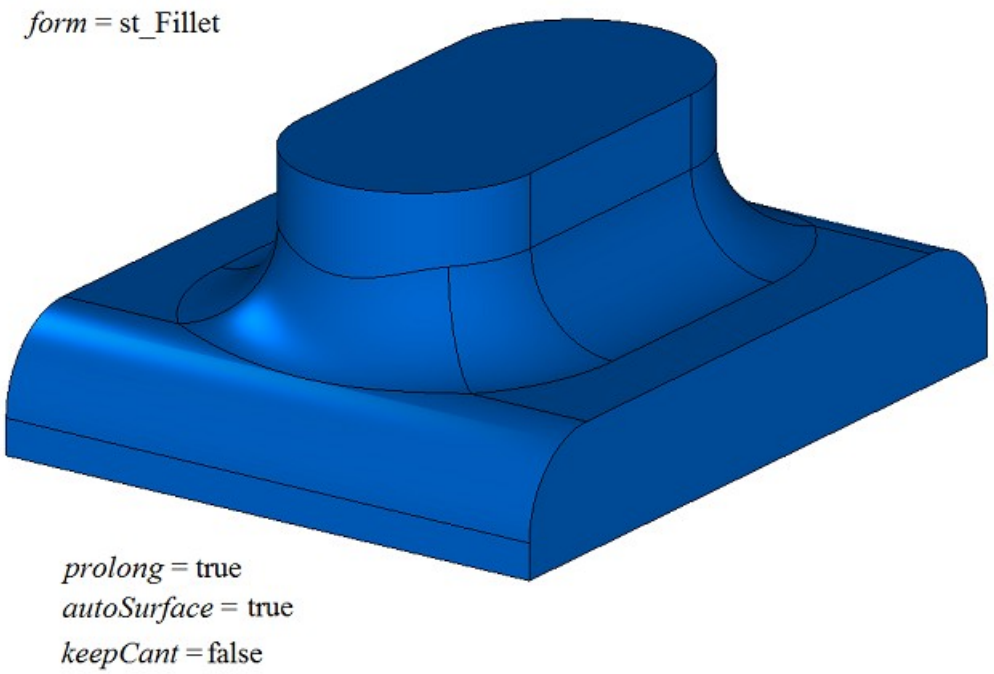


Рис. М.2.10.13.

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

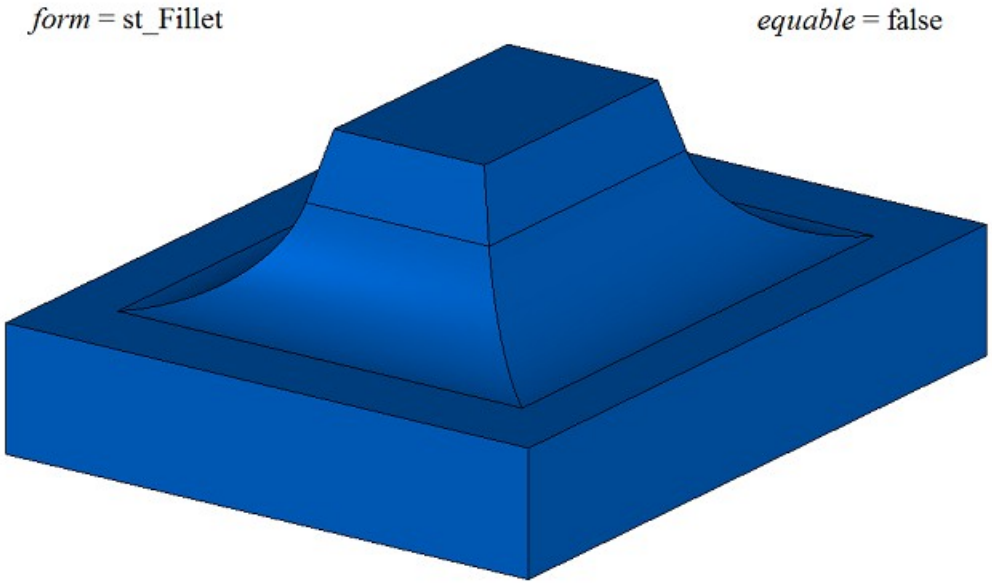


Рис. М.2.10.14.

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

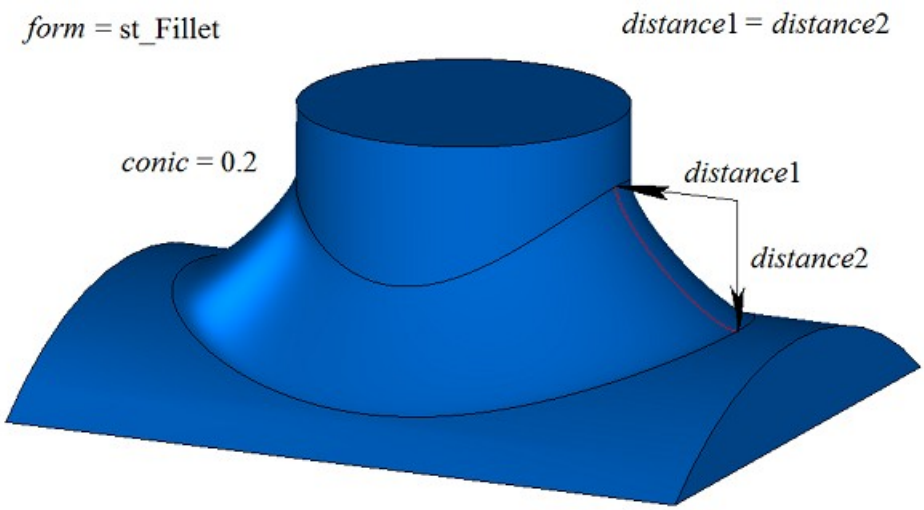


Рис. М.2.10.15.

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

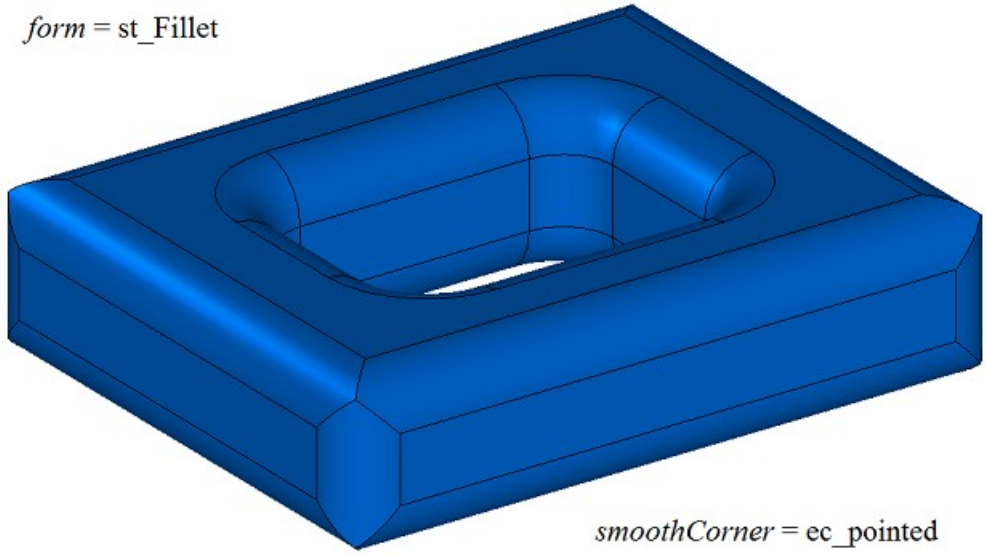


Рис. М.2.10.16.

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

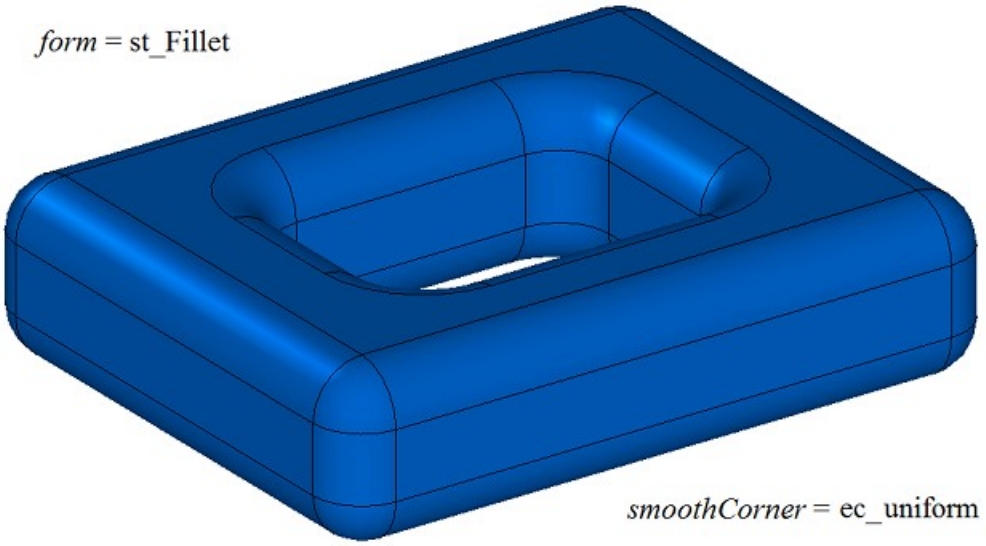


Рис. М.2.10.17.

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

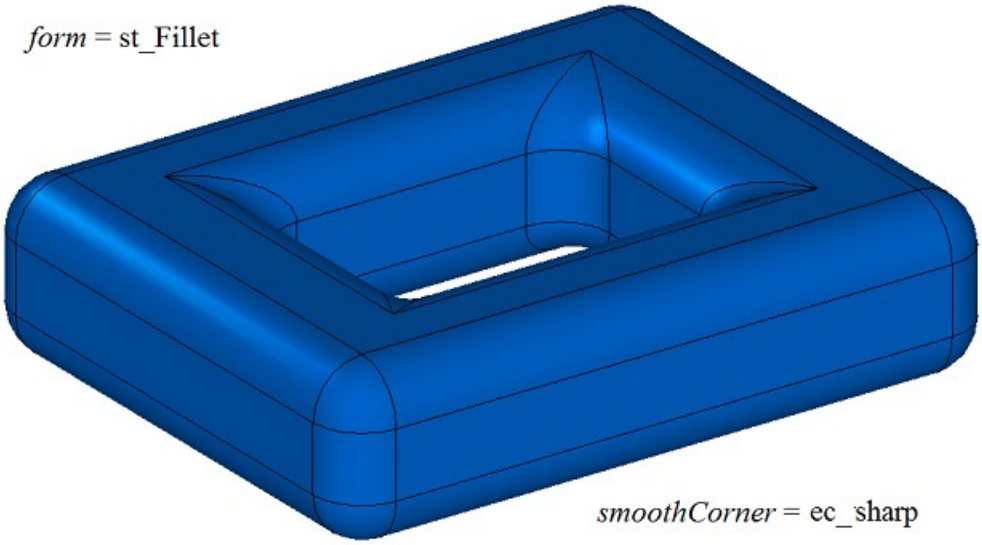


Рис. М.2.10.18

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.

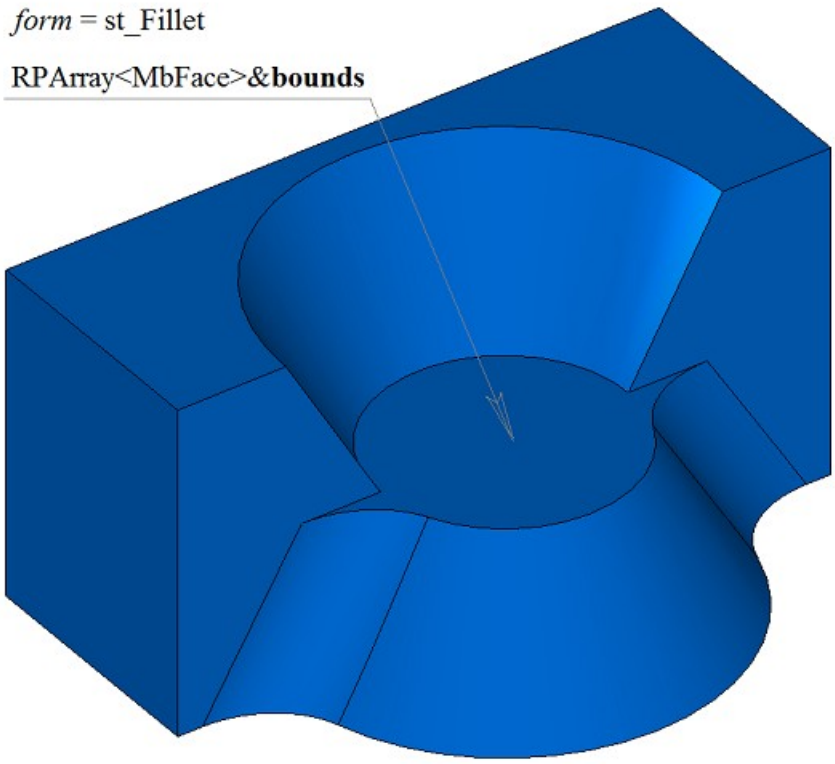


Рис. М.2.10.19

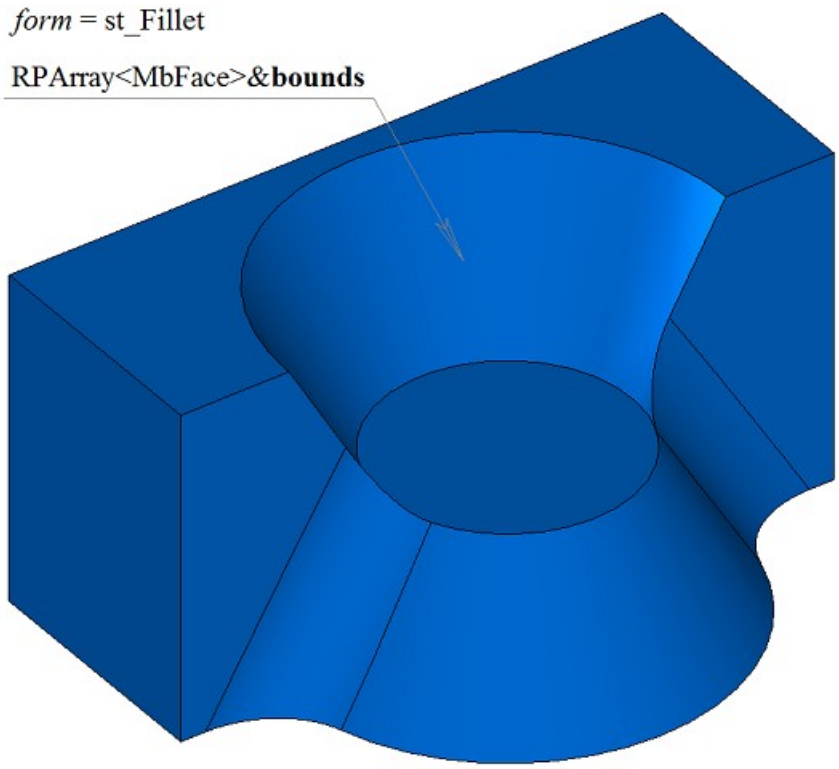


Рис. М.2.10.20

Параметр **bounds** может использоваться для остановки граней сопряжения в начале и конце. При этом грани, определяемые параметром **bounds**, не должны принадлежать исходному телу **solid**. В неоднозначной ситуации направления нормали останавливающей грани в начале сопряжения следует использовать вектор **vector**1, а в неоднозначной ситуации направления нормали останавливающей грани в конце сопряжения следует использовать вектор **vector**2.

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

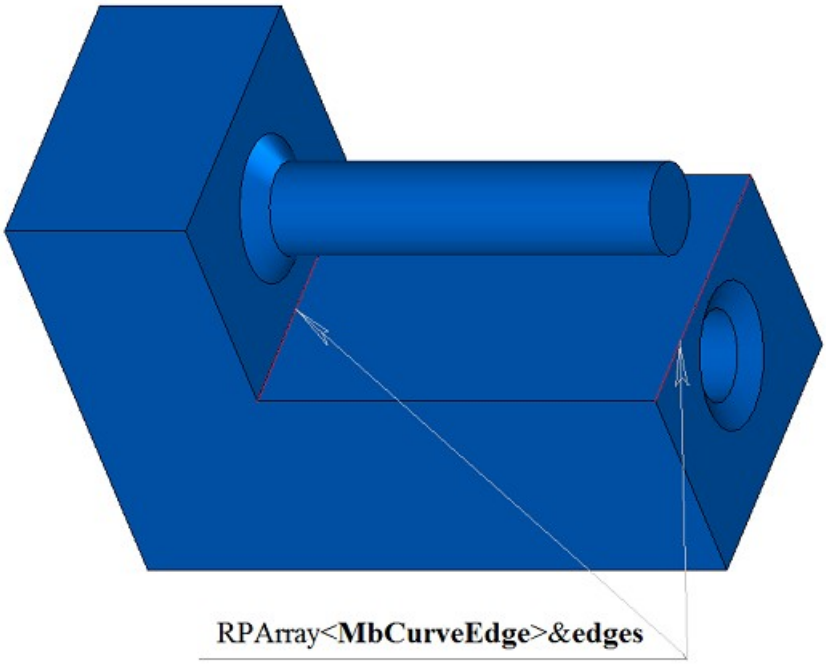


Рис. М.2.10.21

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

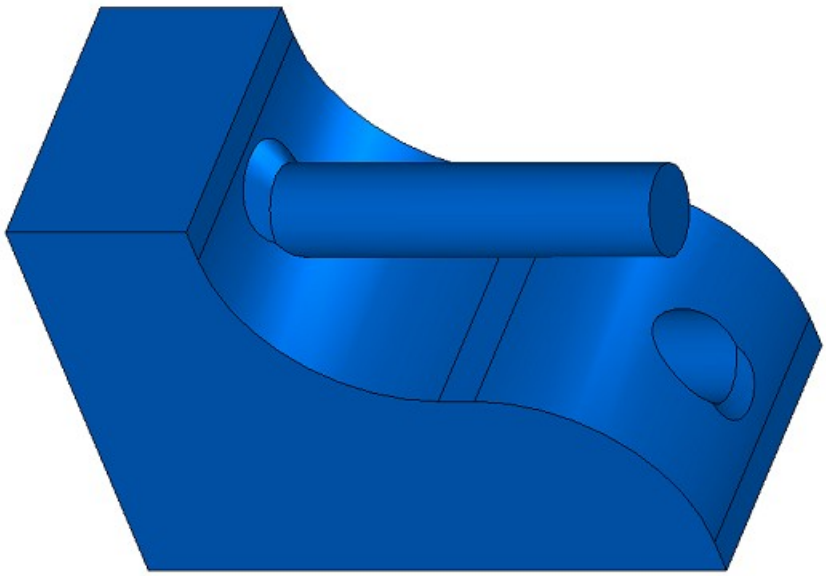


Рис. М.2.10.22

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

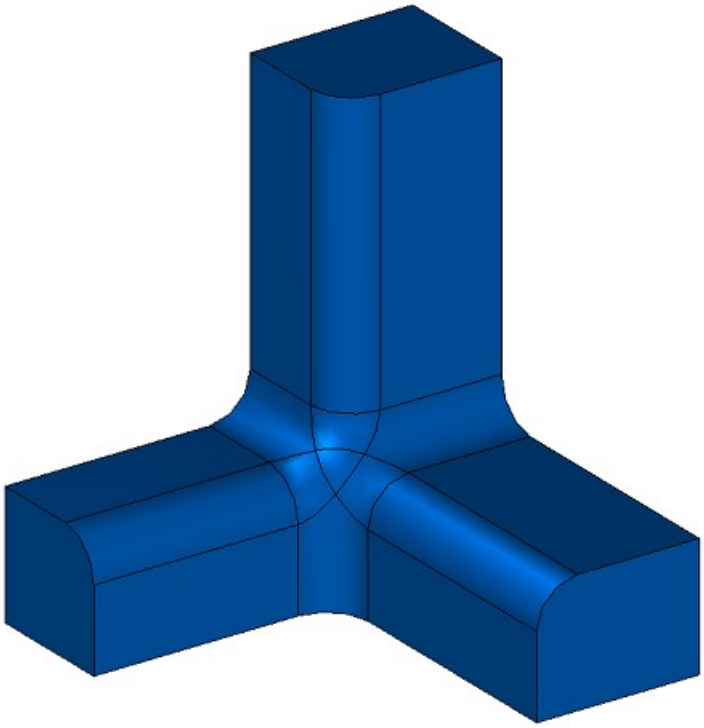


Рис. М.2.10.23

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 проходят совпадают с диагоналями куба.

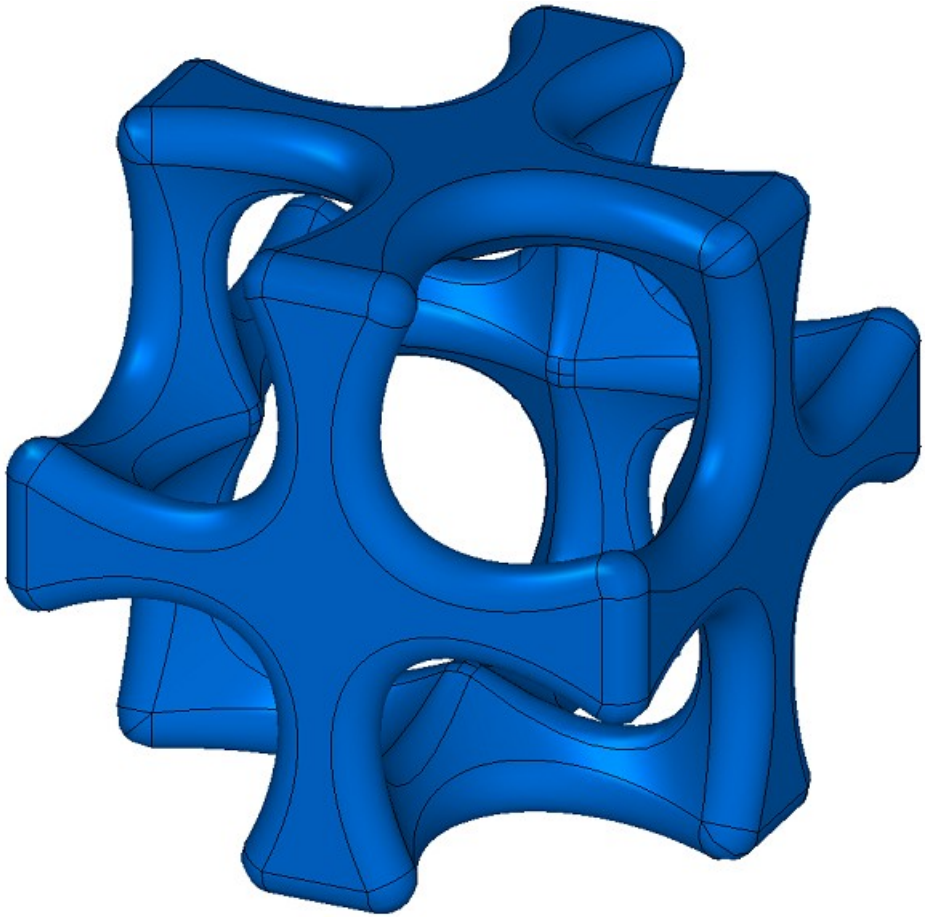


Рис. М.2.10.24

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

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

М.2.11. Скругление рёбер тела переменным радиусом

Метод
MbResultType
FilletSolid (MbSolid & **solid**,
MbeCopyMode *sameShell*,
SArray<MbEdgeFunction> & **edges**,
RPArray<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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.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. М.2.10.1. Класс SmoothValues описан в файле shell_parameter.h.

Входной параметр *params* содержит следующие данные:
*distance*1 – первый радиус скругления или первый катет фаски,
*distance*2 – второй радиус скругления или второй катет фаски,
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),
MbeSmoothForm *form* – тип сопряжения,
smoothCorner – способ скругления чемоданных 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 *distance*1 и *distance*2, рис. М.2.11.1.

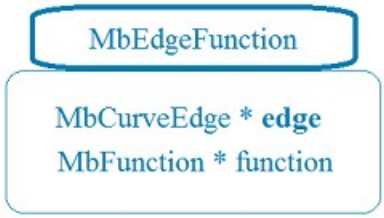


Рис. М.2.11.1.

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**->Point(*t*,**point**) радиусы поверхности скругления равны параметрам *distance*1 и *distance*2, умноженным на значение функции ребра **edges**[i].function->Value(*t*). На рис. М.2.11.2 приведено скругление с переменными радиусами ребра прямоугольной призмы. При *distance*1=*distance*2 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.

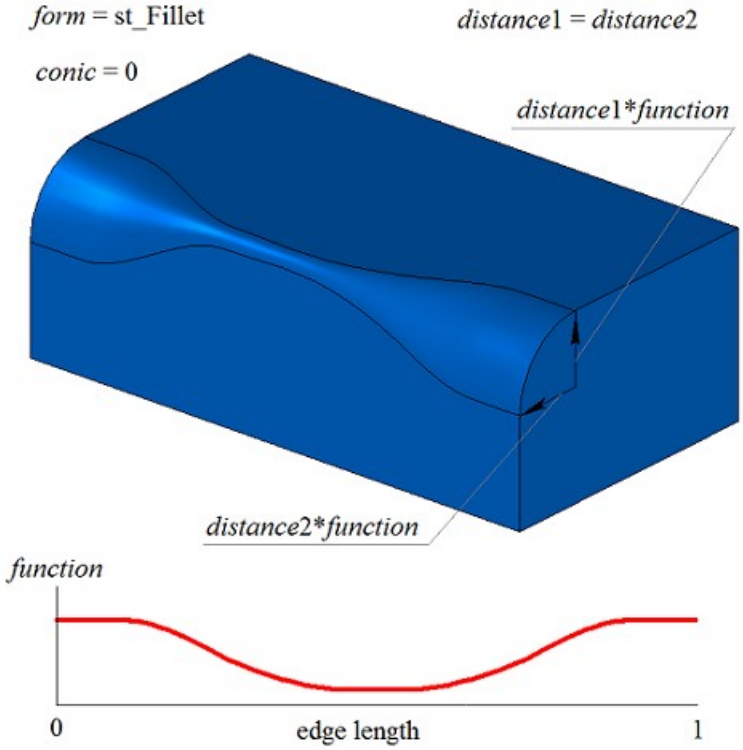


Рис. М.2.11.2.

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

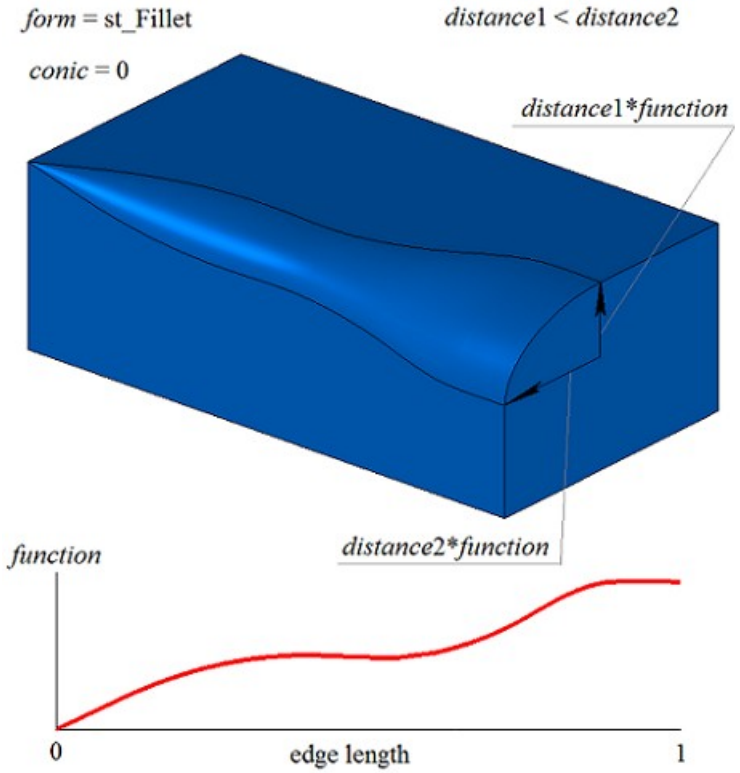


Рис. М.2.11.3.

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.

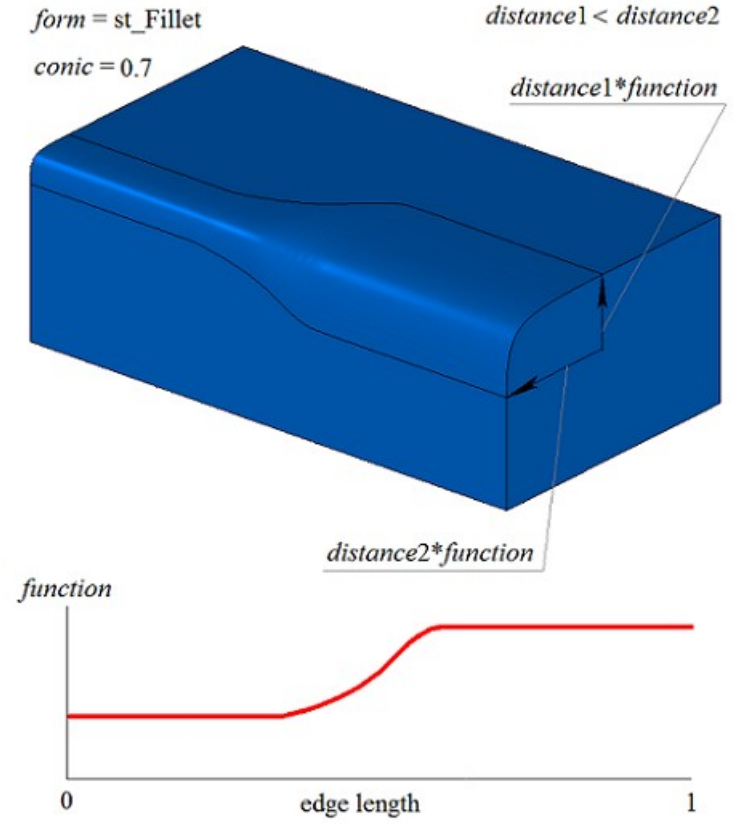


Рис. М.2.11.4.

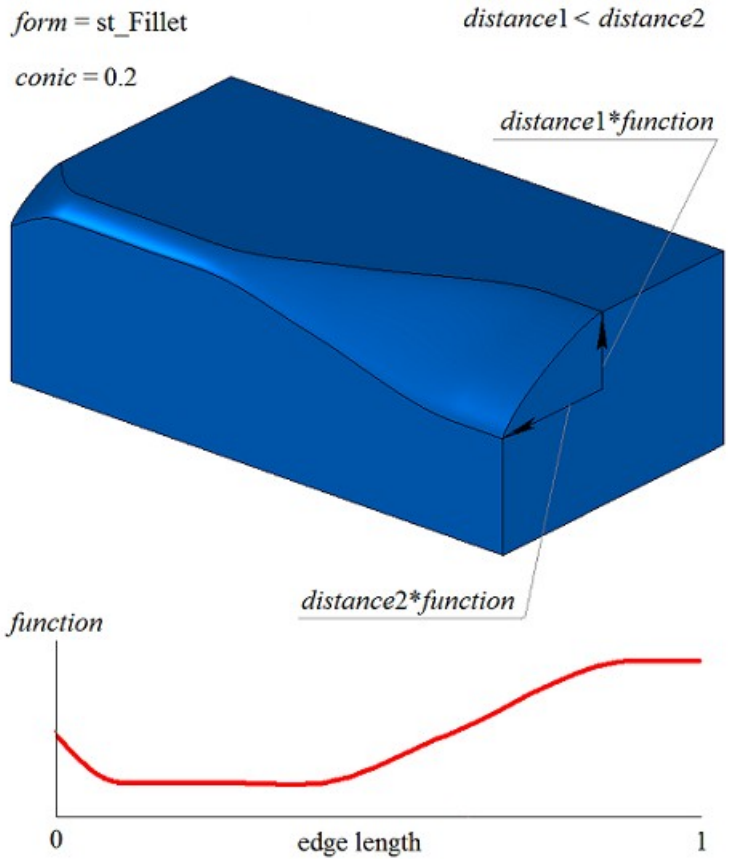


Рис. М.2.11.5.

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.

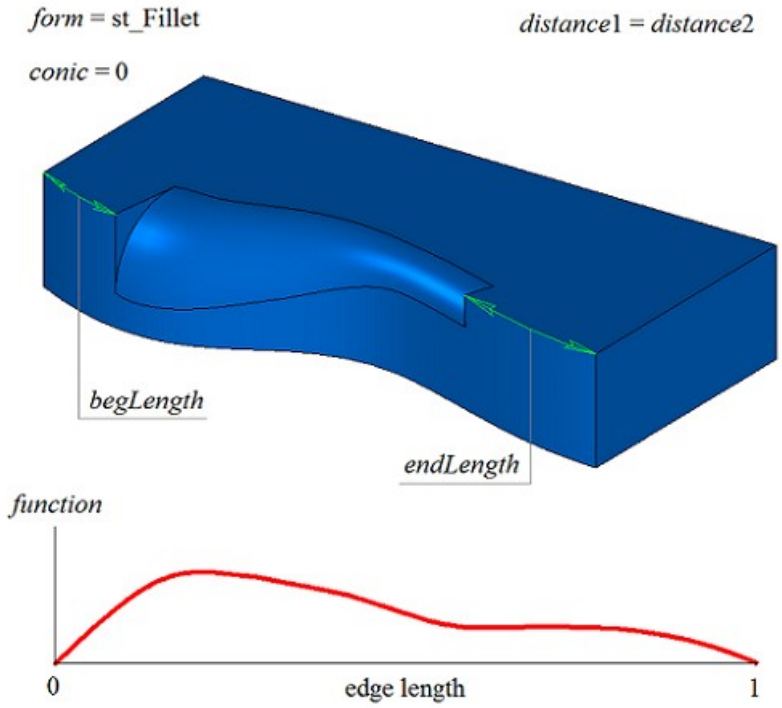


Рис. М.2.11.6.

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.

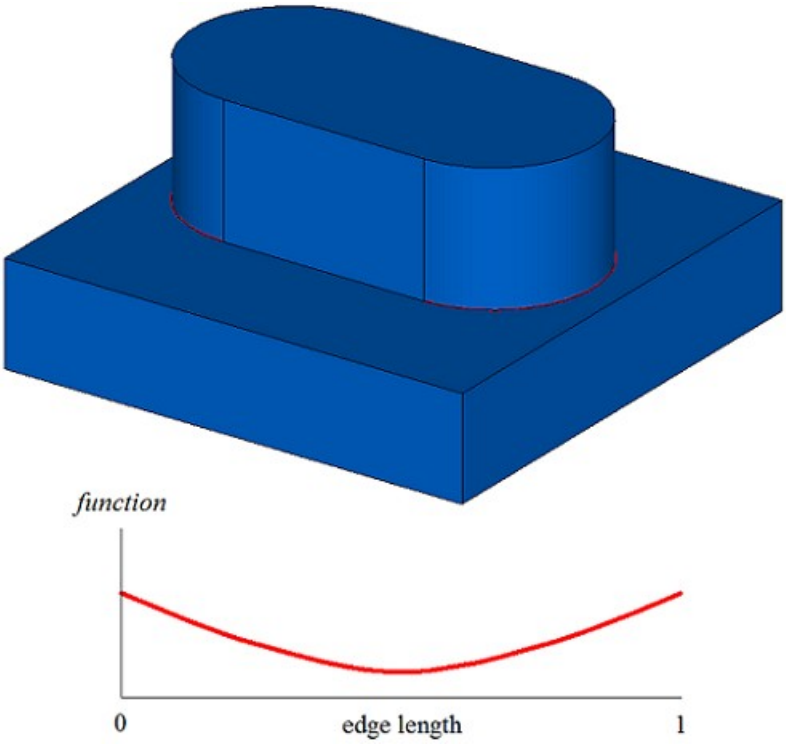


Рис. М.2.11.7.

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 изменен так, чтобы обеспечить гладкое сопряжение с соседней гранью скругления.

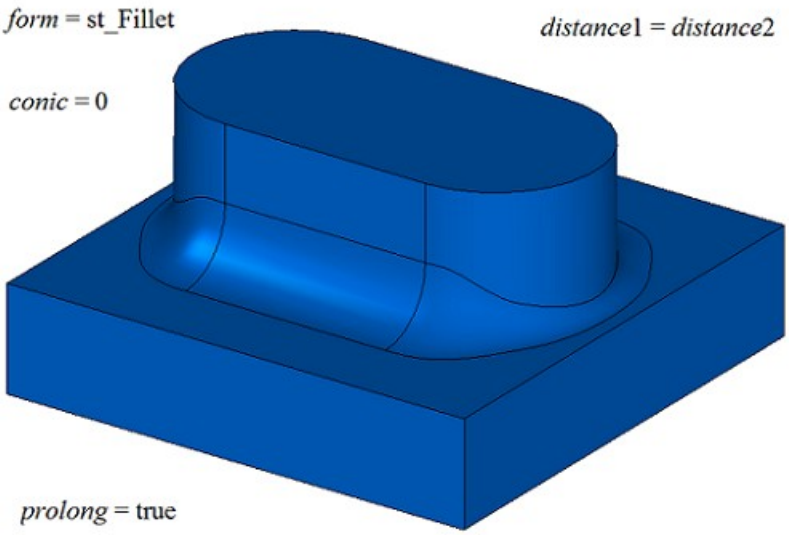


Рис. М.2.11.8.

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 чемоданных 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**, не должны принадлежать исходному телу **solid**. В неоднозначной ситуации направления нормали останавливающей грани в начале сопряжения следует использовать вектор **vector**1, а в неоднозначной ситуации направления нормали останавливающей грани в конце сопряжения следует использовать вектор **vector**2.

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*,
RPArray<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 – 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 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, ребер и вершин от исходного тела **solid** к построенному телу **result**.

Параметр перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode 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 ребер используются те же параметры SmoothValues & *params*, что и при построении скруглений ребер, 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 ребер. Для построения фасок используются следующие данные параметра *params*:

*distance*1 – первый катет фаски,

*distance*2 – 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),
MbeSmoothForm *form* – тип сопряжения,
smoothCorner – способ скругления чемоданных 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 ребер используются значения параметра *form* равные st_Chamfer, stSlant1 и stSlant2. При значении *form*=st_Chamfer рассматриваемый метод строит поверхность фаски с заданными катетами, которые определяют параметры *distance*1 и *distance*2, рис. М.2.12.1.

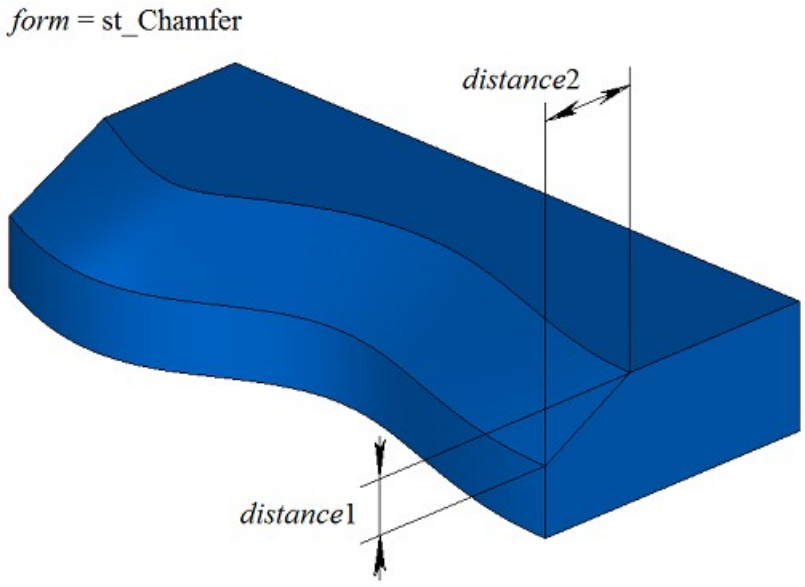


Рис. М.2.12.1.

При значении *form*=st_Slant1 рассматриваемый метод строит поверхность фаски с заданным катетом и прилегающим к нему углом. Катет определяет параметр *distance*1, а *distance*2 соответствует катету, обеспечивающему заданный прилежащий угол, рис. М.2.12.2.

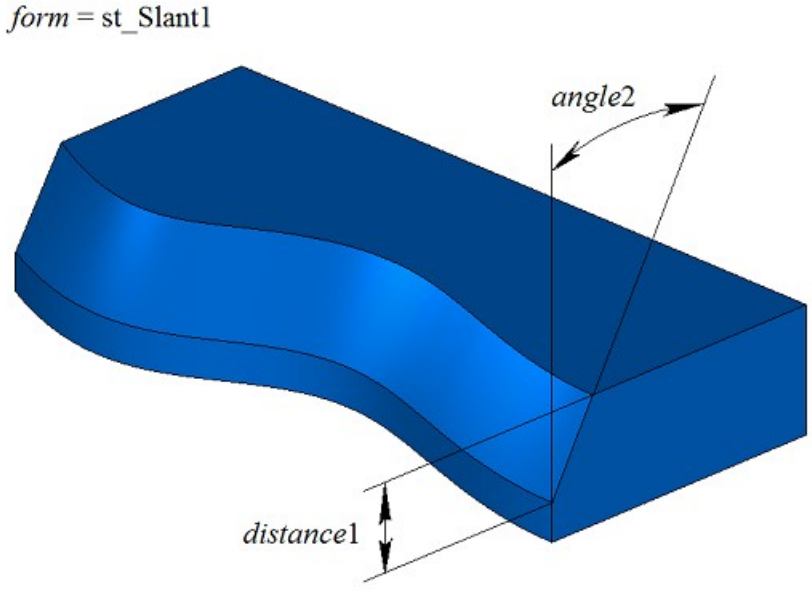


Рис. М.2.12.2.

При значении *form*=st_Slant2 рассматриваемый метод строит поверхность фаски с заданным углом и прилегающим к нему катетом. Параметр *distance*1 соответствует катету, обеспечивающему заданный угол, а *distance*2 определяет прилежащий катет, рис. М.2.12.3.

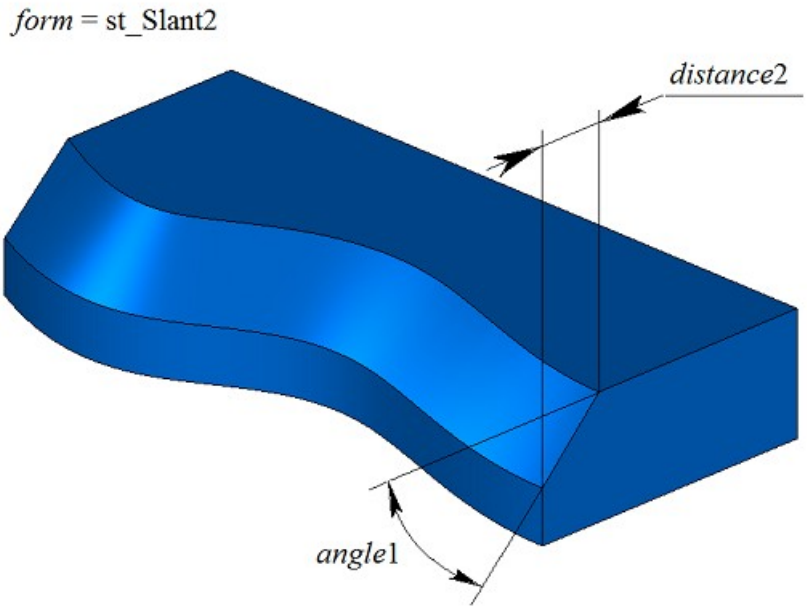


Рис. М.2.12.3.

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.

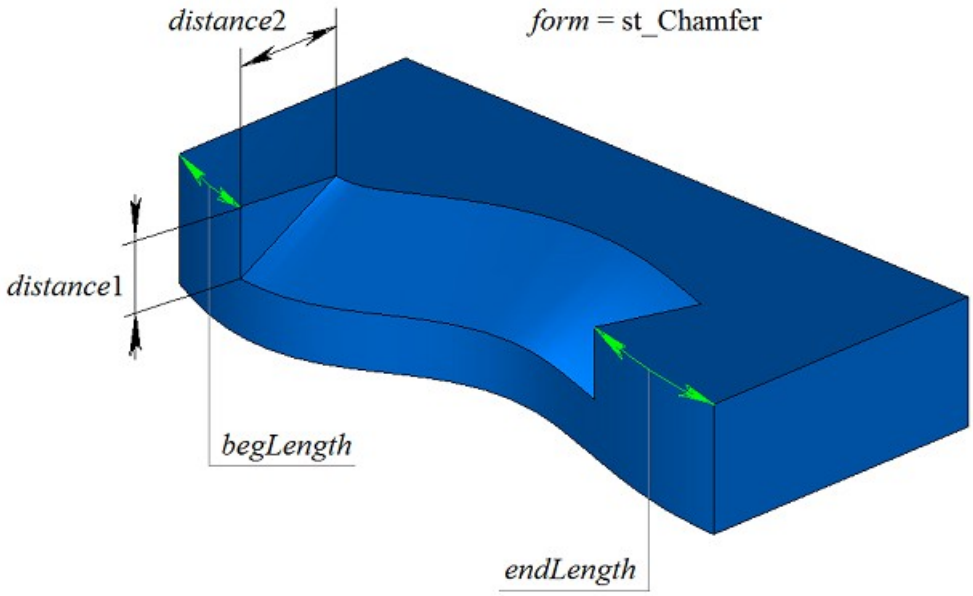


Рис. М.2.12.4.

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

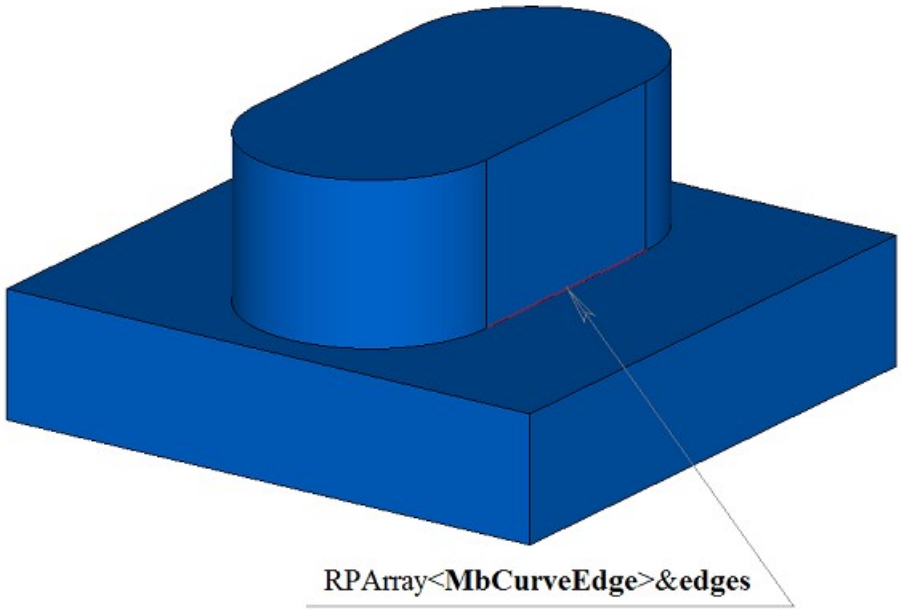


Рис. М.2.12.5.

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.

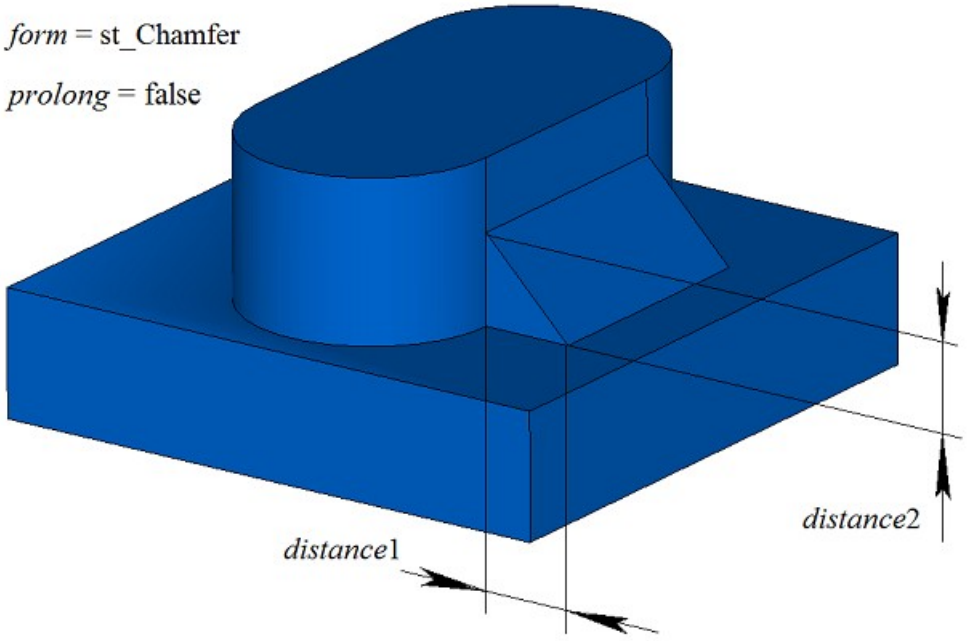


Рис. М.2.12.6.

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

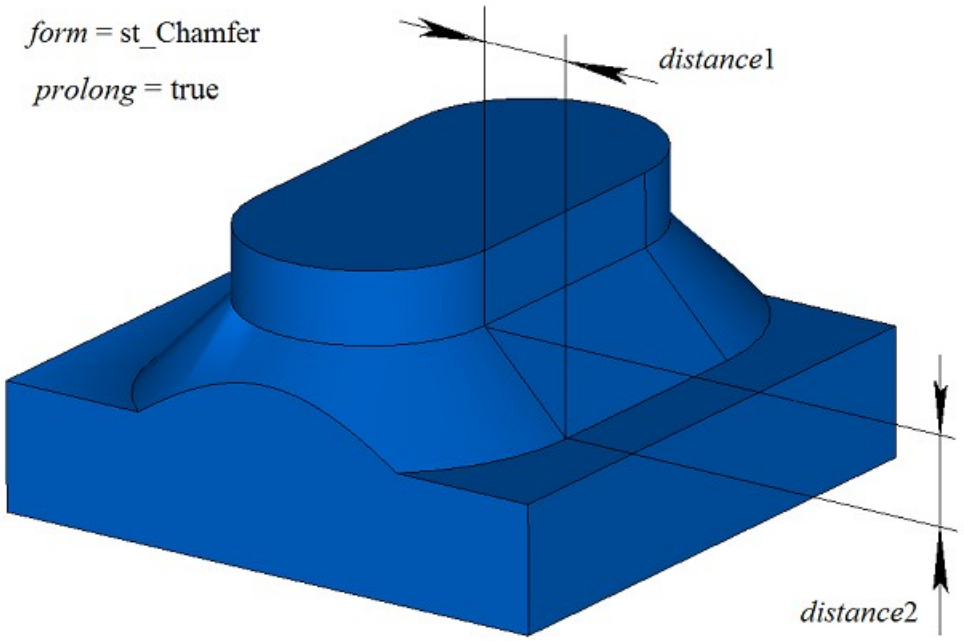


Рис. М.2.12.7.

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

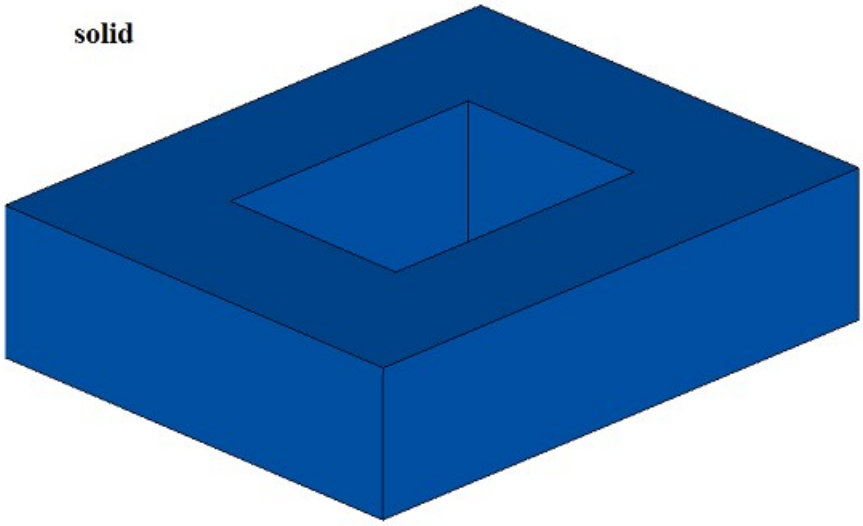


Рис. М.2.12.8.

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.

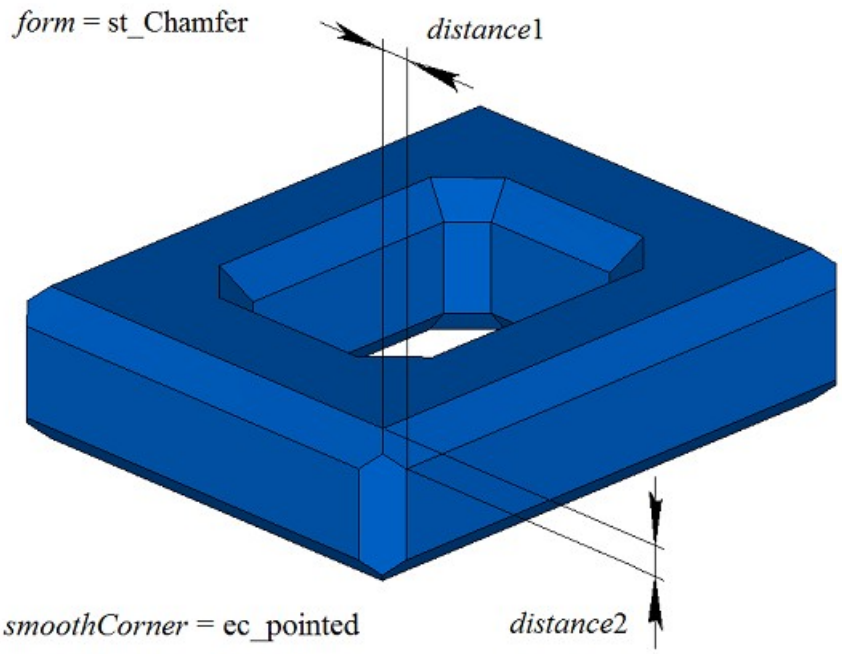


Рис. М.2.12.9.

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.

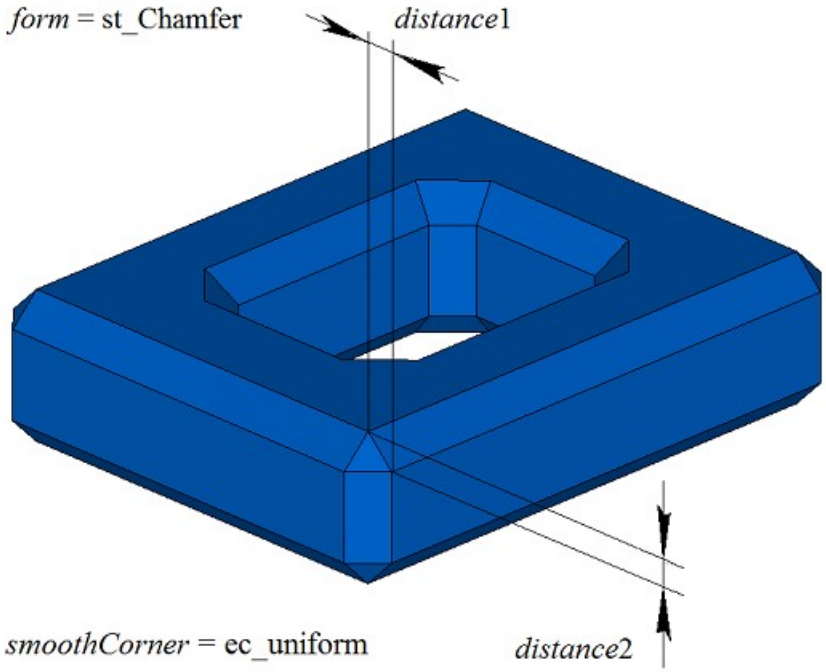


Рис. М.2.12.10.

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.

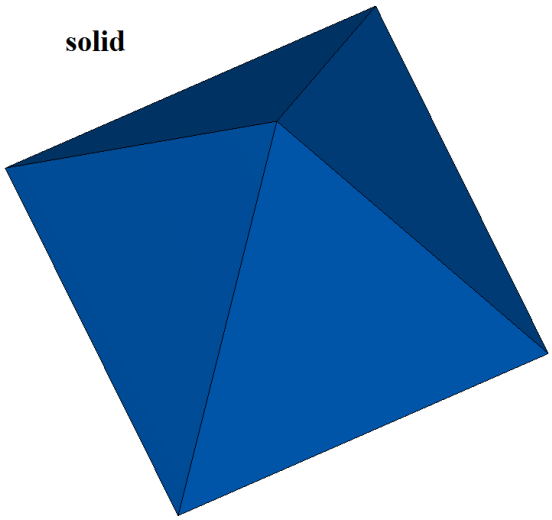


Рис. М.2.12.11.

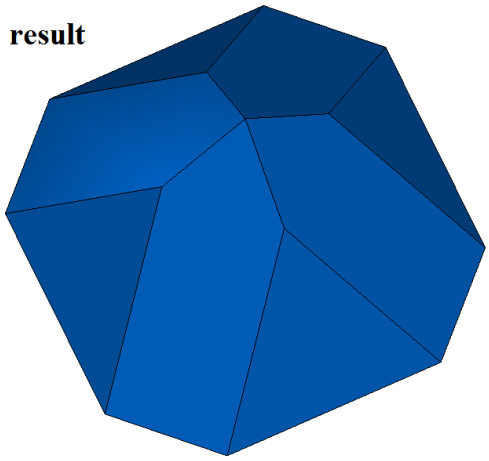


Рис. М.2.12.12.

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.

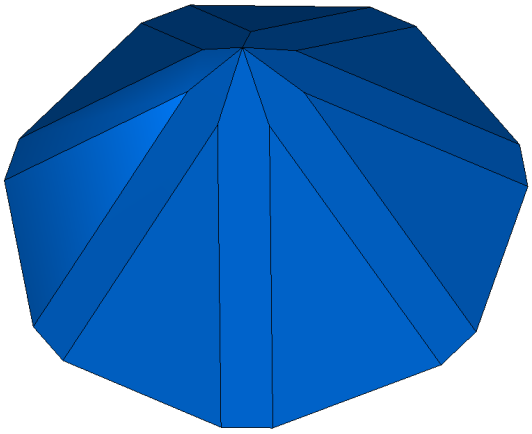


Рис. М.2.12.13.

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

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

М.2.13. Построение тонкостенного тела

Метод

MbResultType

ThinSolid (MbSolid & **solid**,
MbeCopyMode *sameShell*,
RPArray<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 – 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 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.thickness*1 в положительном направлении нормали грани или *params.thickness*2 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.7.9. Копирование множества граней MbFaceShell.

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

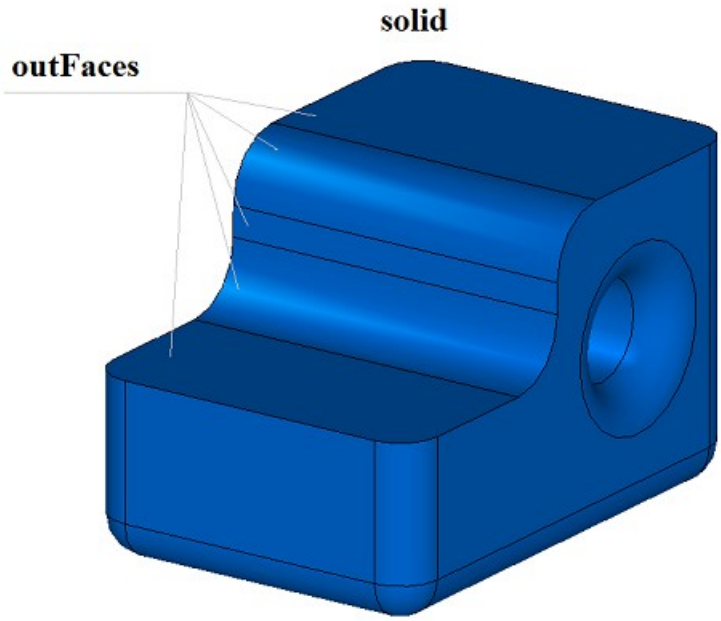


Рис. М.2.13.1.

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

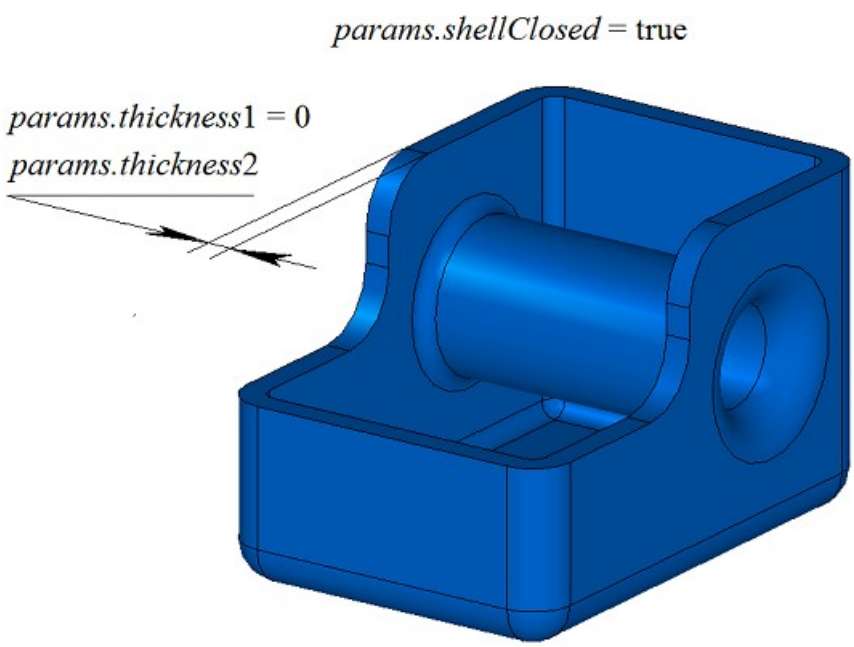


Рис. М.2.13.2.

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

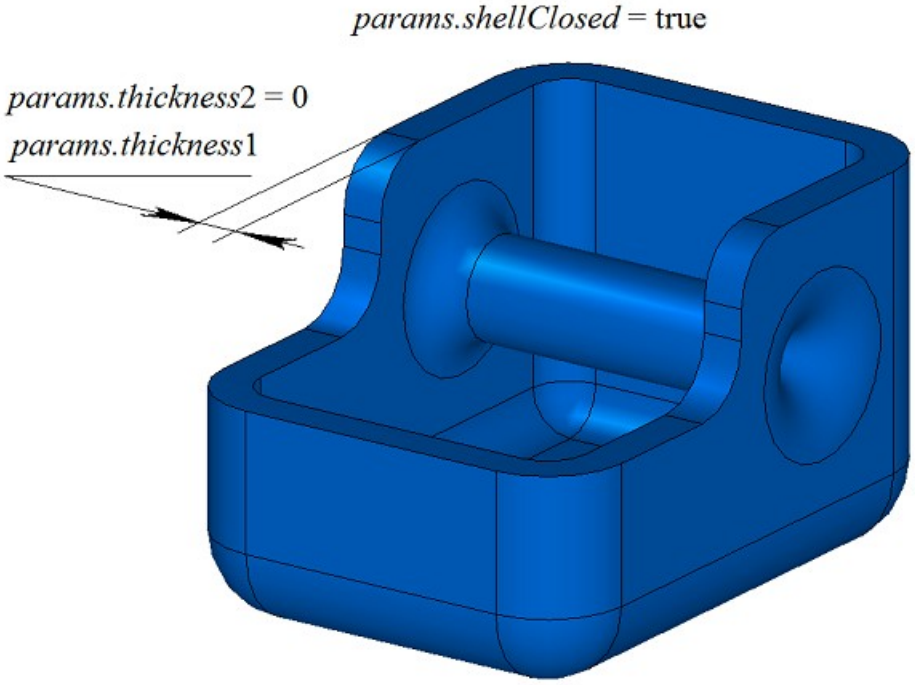


Рис. М.2.13.3.

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

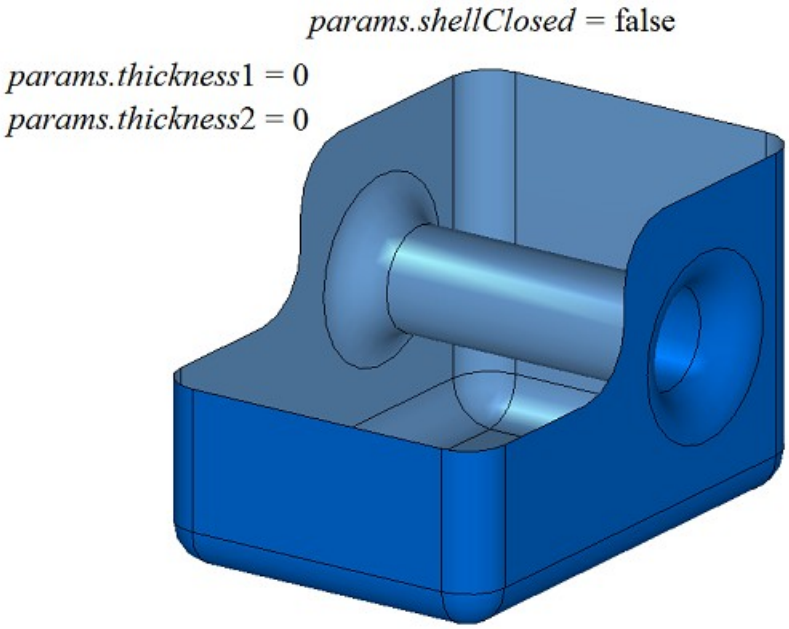


Рис. М.2.13.4.

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

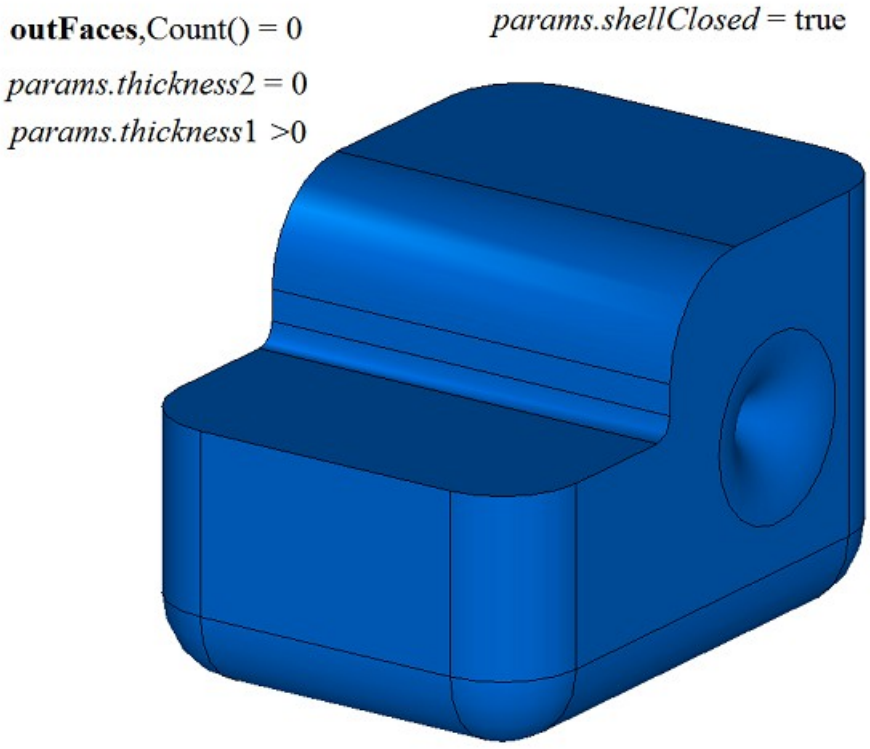


Рис. М.2.13.5.

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*,
RPArray<MbFace> & **outFaces**,
RPArray<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 – 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 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.thickness*1 в положительном направлении нормали грани или *params.thickness*2 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.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*.

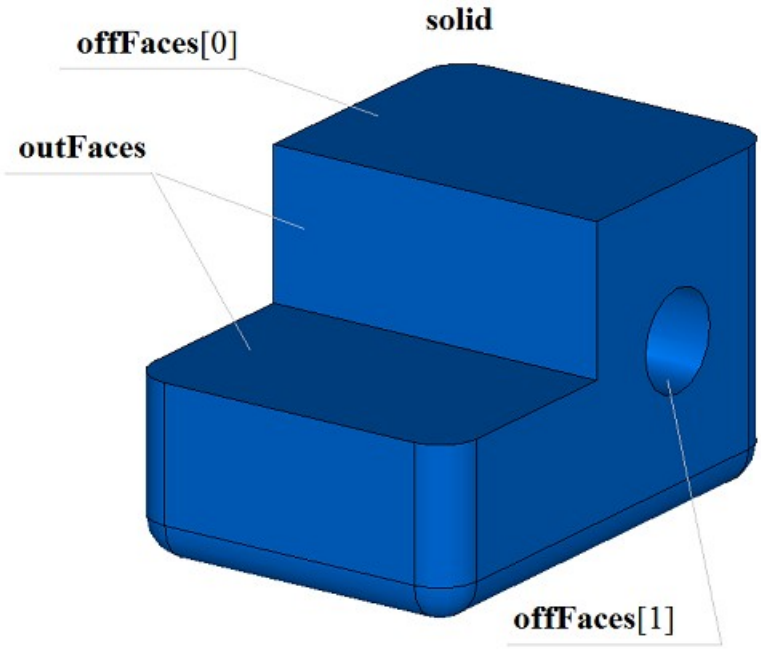


Рис. М.2.14.1.

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

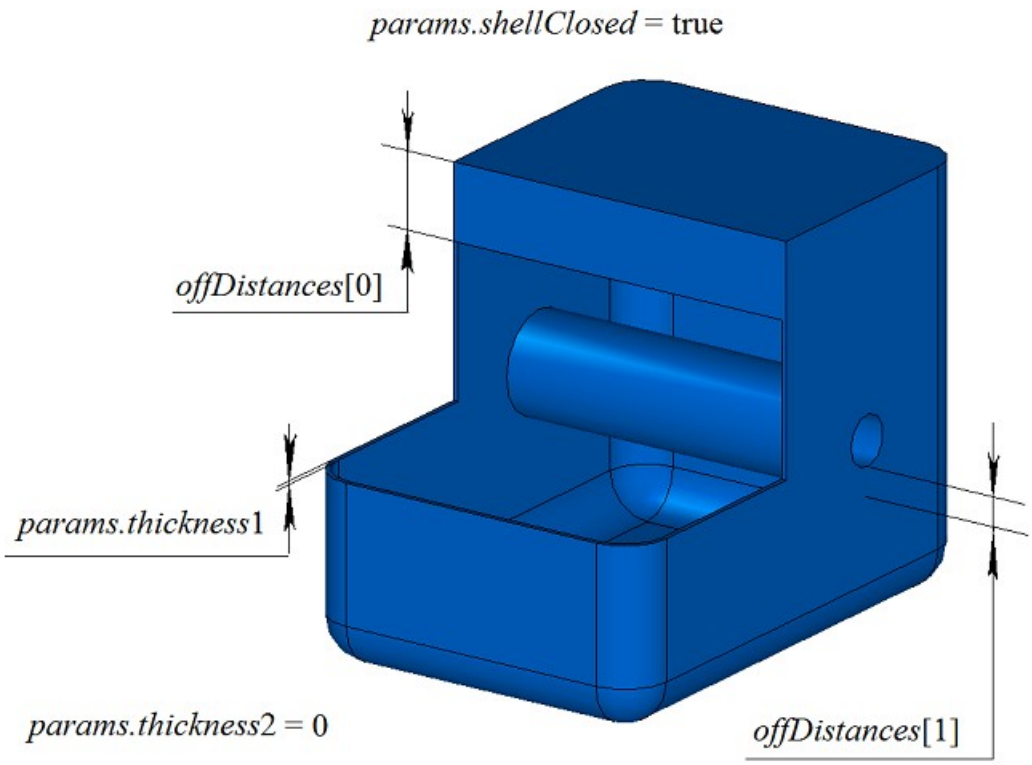


Рис. М.2.14.2.

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.thickness*1 в положительном направлении нормали грани или *params.thickness*2 в отрицательном направлении нормали грани. Параметры names и name обеспечивают именование граней построенного тела.

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

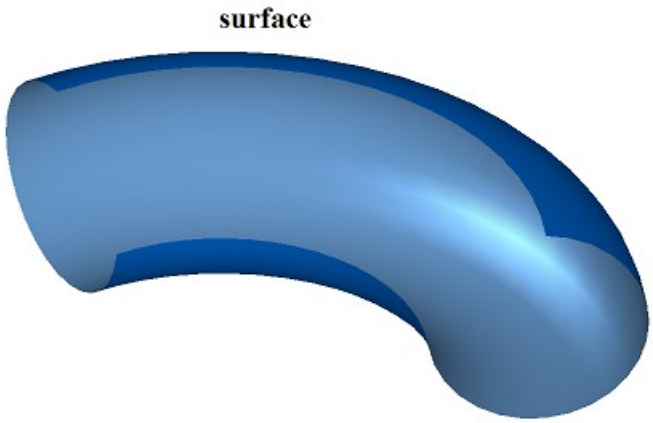


Рис. М.2.15.1.

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

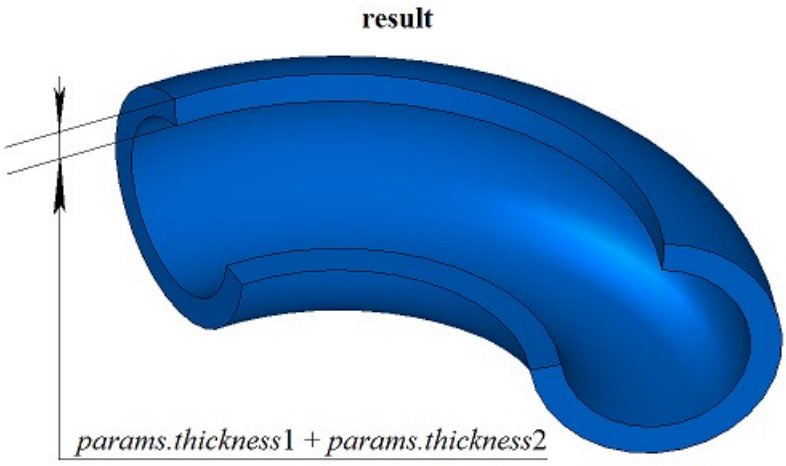


Рис. М.2.15.2.

Метод **ThinSolid** добавляет в журнал построенного тела строитель MbShellSolid, который содержит все необходимые данные для выполнения операции. Строитель MbShellSolid объявлен в файле cr_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 – именователь 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 именование граней построенного тела.

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

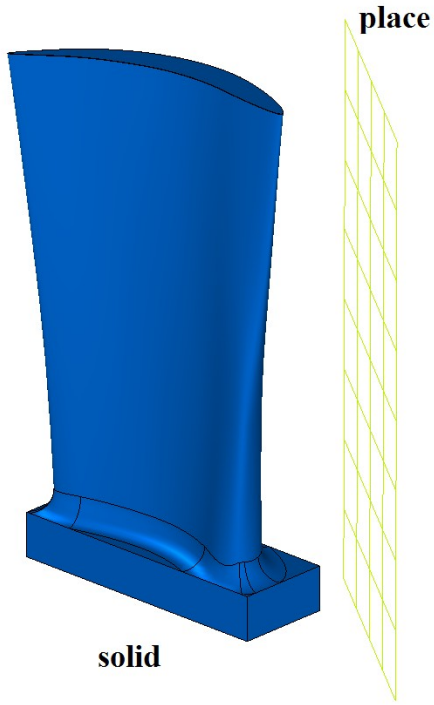


Рис. М.2.16.1.

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

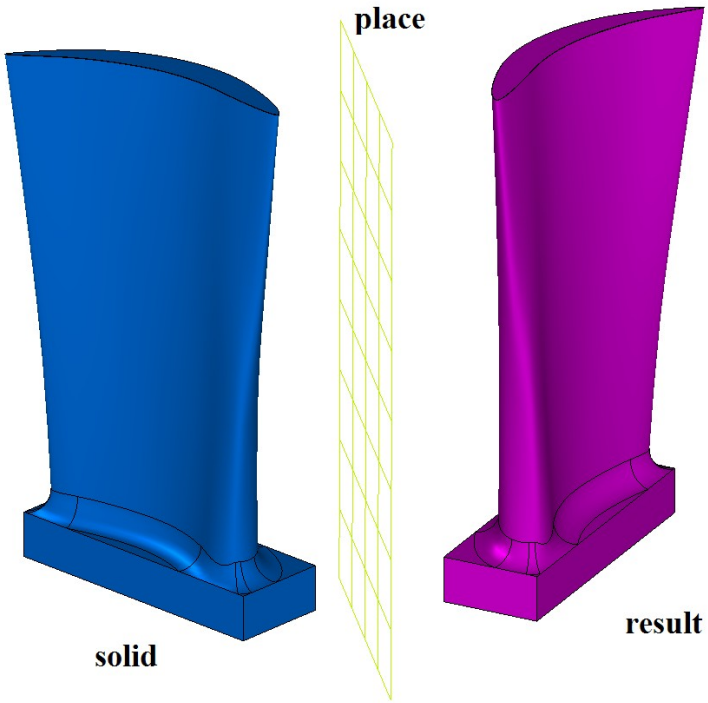


Рис. М.2.16.2.

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 симметричное тело командой меню «Создать->Тело->На базе тел->Симметричное».

М.2.17. Булева операция тела и множества тел

Метод

MbResultType
UnionResult (MbSolid * **solid**,
MbeCopyMode *sameShell*,
RPArray<MbSolid> & **solids**,
MbeCopyMode *sameShells*,
OperationType *oType*,
bool *checkIntersect*,
bool *mergeFaces*,
const MbSNameMaker & names,
bool *isArray*,
MbSolid *& **result**,
RPArray<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 – именователь 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*. При значении *cm_Copy* тело **solid** (тела **solids**) полностью копируется, поэтому построенное тело **result** и тело **solid** (тела **solids**) не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и тело **solid** (тела **solids**) будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и тело **solid** (тела **solids**) будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты тела **solid** (**solids**), поэтому после построения тело **solid** (тела **solids**) должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.7.9. Копирование множества граней MbFaceShell.

Параметр OperationType *oType* определяет тип былевой 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 перекладыванием 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**.

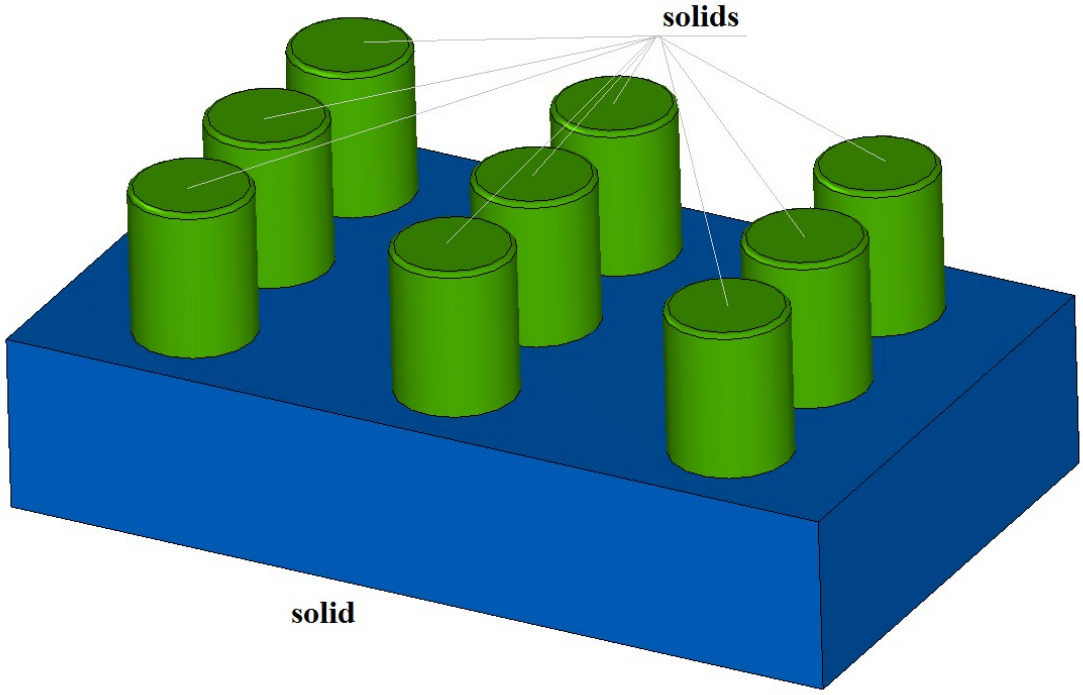


Рис. М.2.17.1.

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.

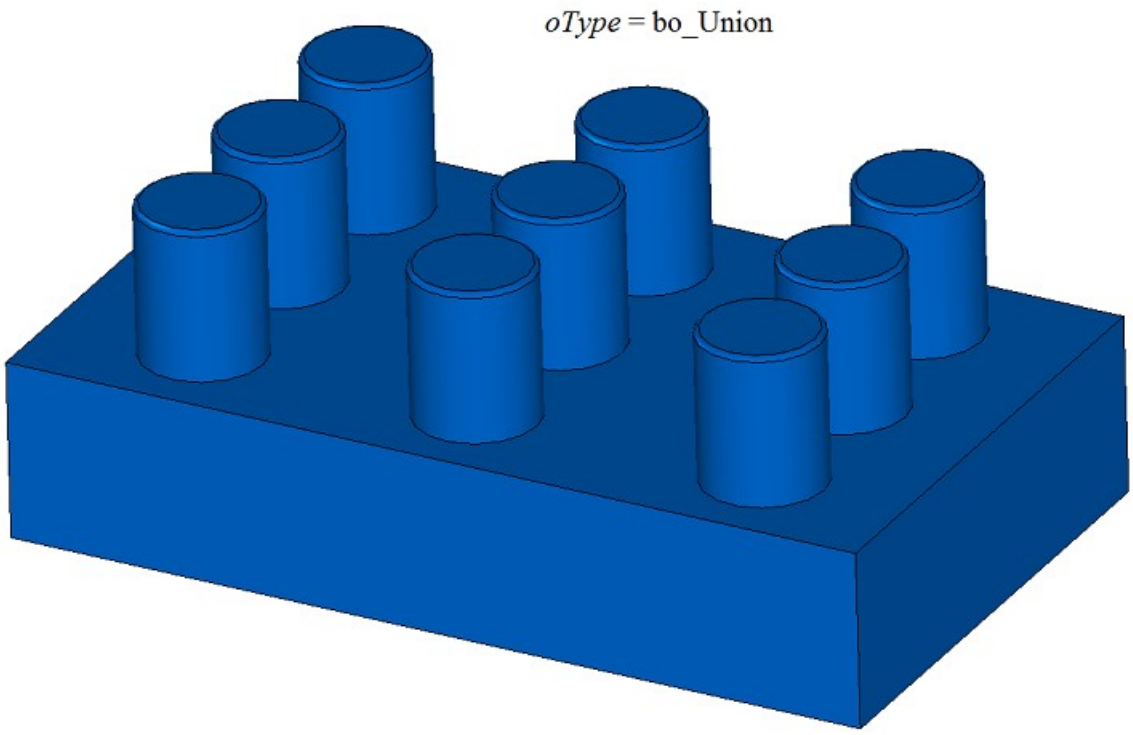


Рис. М.2.17.2.

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

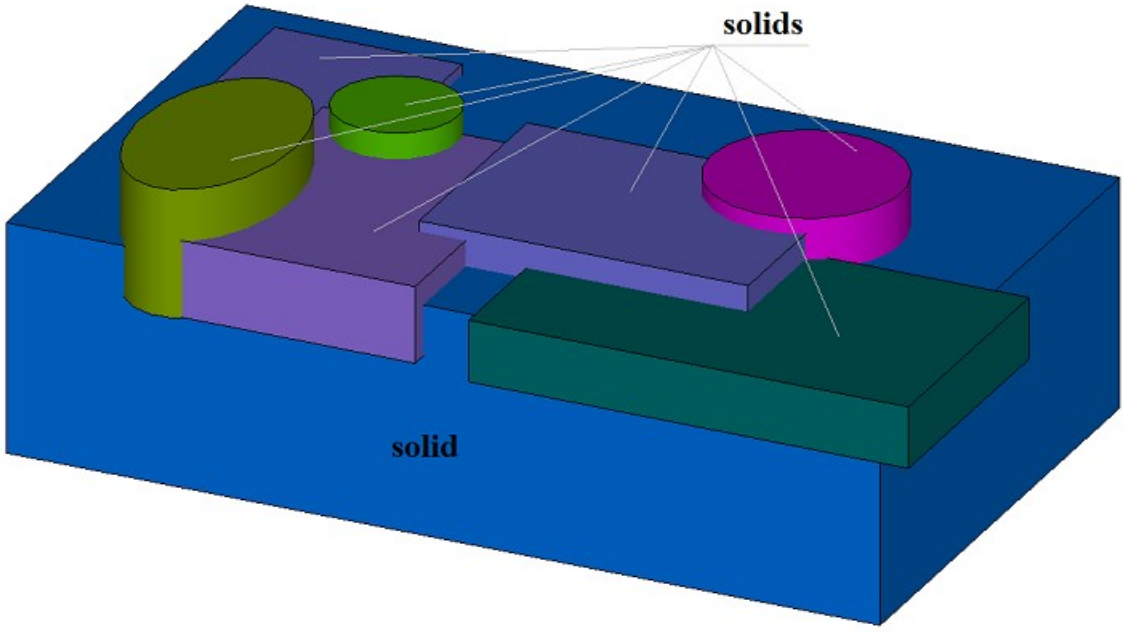


Рис. М.2.17.3.

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.

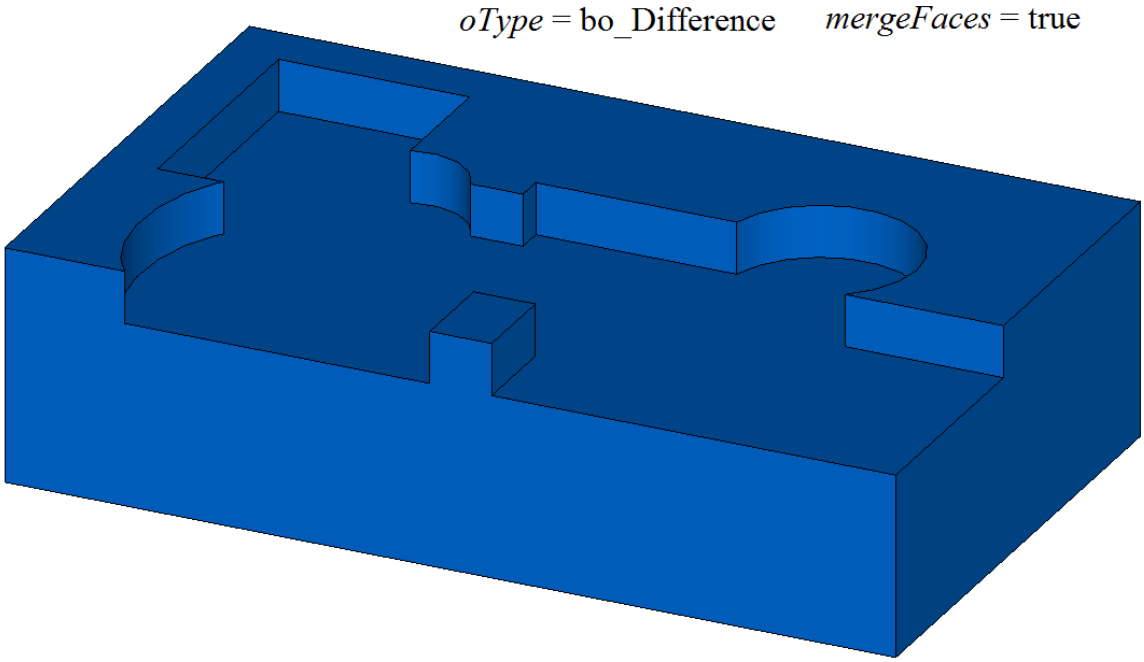


Рис. М.2.17.4.

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

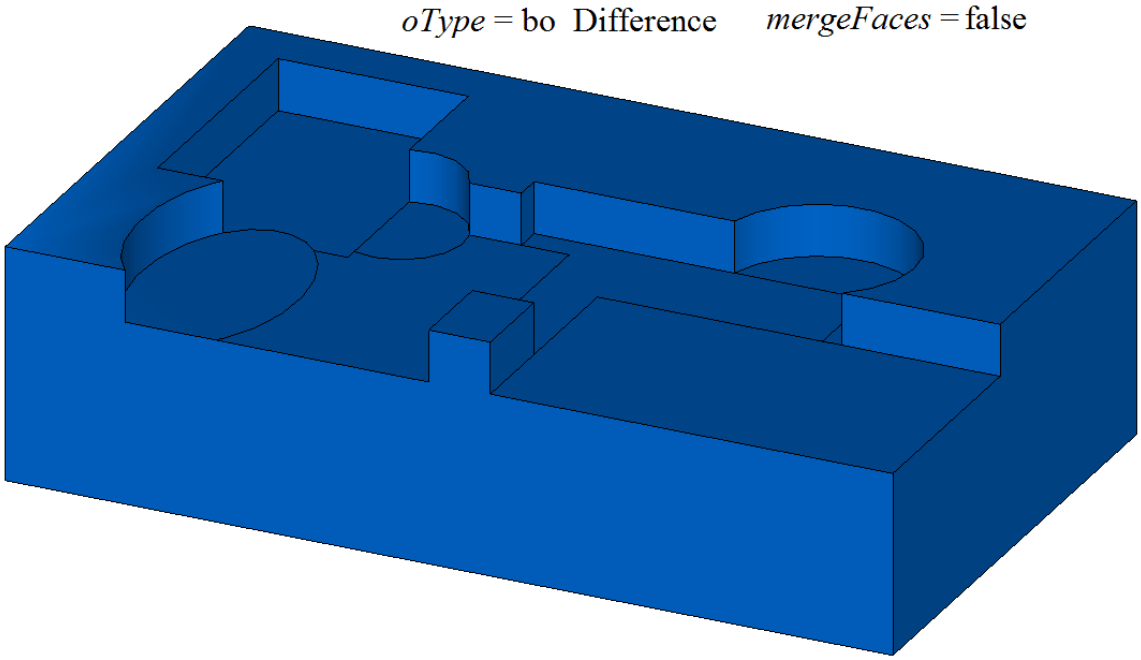


Рис. М.2.17.5.

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.

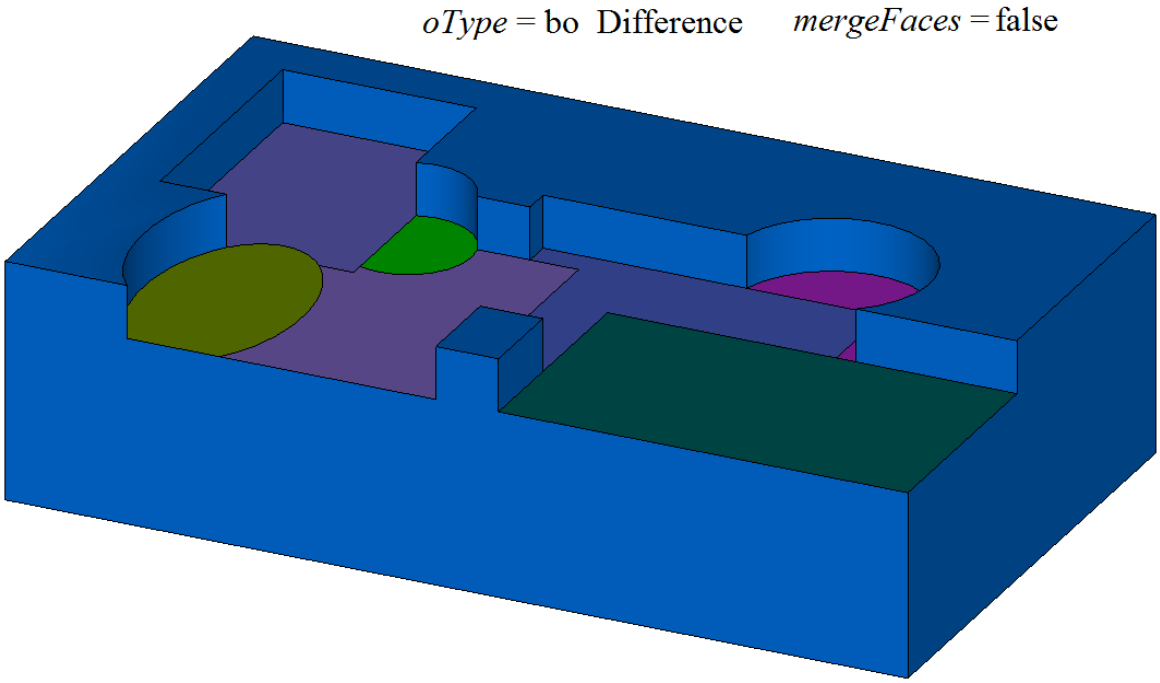


Рис. М.2.17.6.

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

М.2.18. Объединение множества тел

Метод
MbResultType
UnionSolid (RPArray<MbSolid> & **solids**,
MbeCopyMode *sameShells*,
bool *checkIntersect*,
const MbSNameMaker & names,
bool *isArray*,
MbSolid*& **result**,
RPArray<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 – именователь 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*. При значении *cm_Copy* тела **solids** полностью копируются, поэтому построенное тело **result** и тела **solids** не будут связаны между собой. При значении *cm_KeepSurface* построенное тело **result** и тела **solids** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и тела **solids** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты тел **solids**, поэтому после построения тела **solids** должны быть удалены. Перечисление MbeCopyMode описано в параграфе О.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. М.2.18.1 приведены исходные тела **solids**.

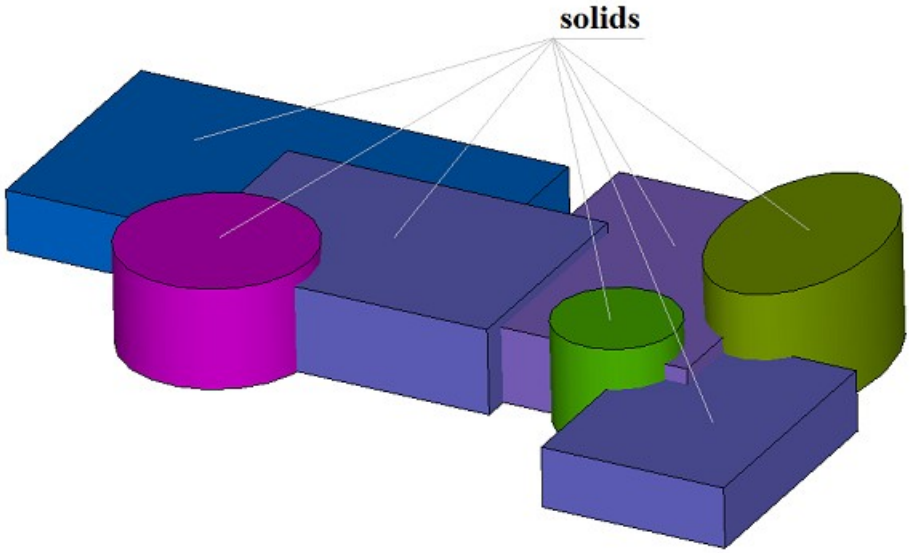


Рис. М.2.18.1.

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.

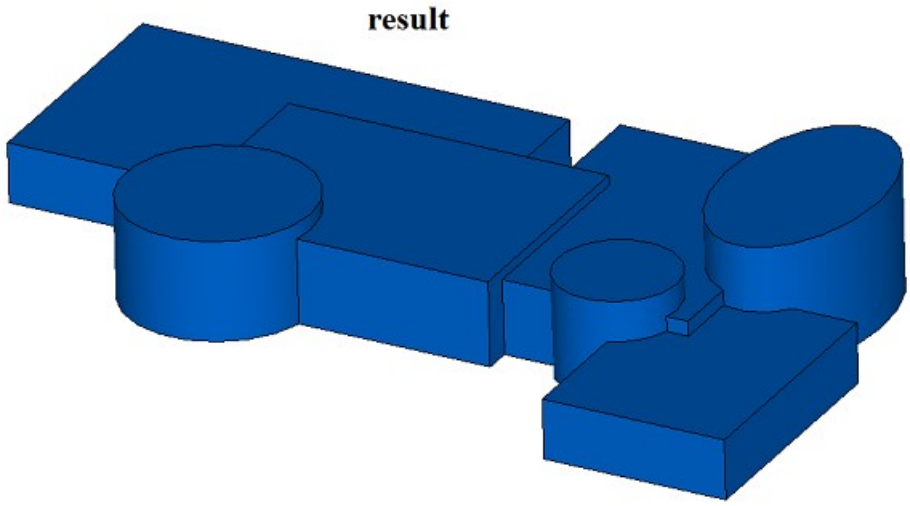


Рис. М.2.18.2.

Метод
MbResultType
UnionSolid (const RPArray<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**,
RPArray<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 отделенных частей.

Метод объявлен в файле action_solid.h.

После вычитания тела **solid2** из тела **solid2**, 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**.

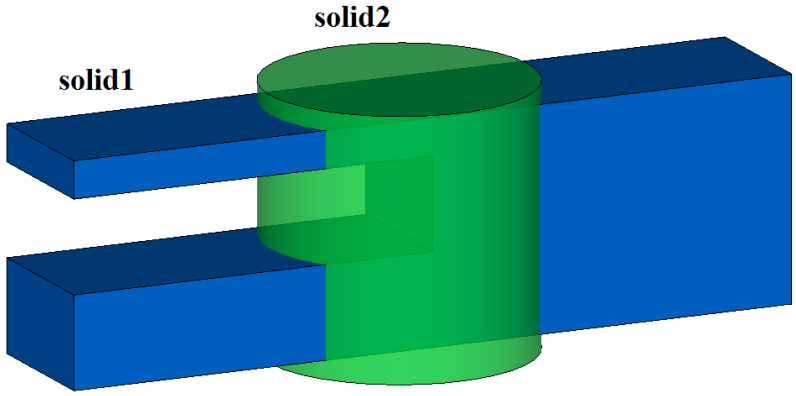


Рис. М.2.19.1.

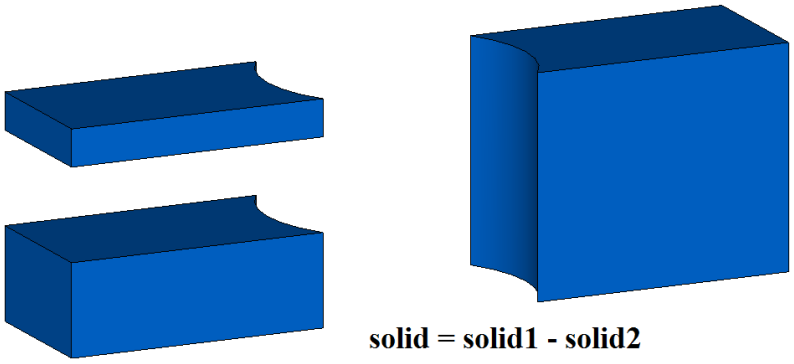


Рис. М.2.19.2.

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.

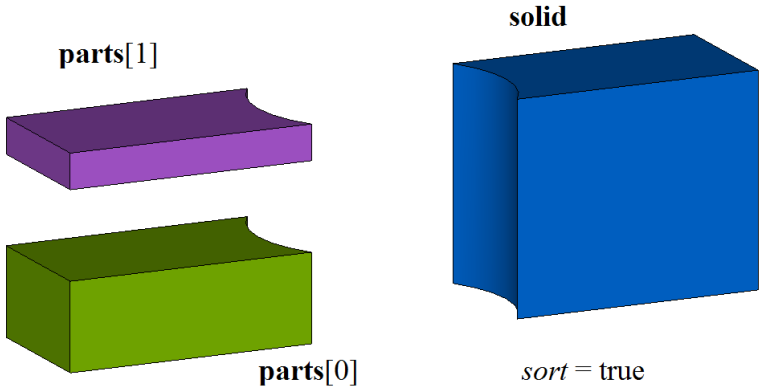


Рис. М.2.19.3.

Параметр names обеспечивает именование граней построенного тела и версионирование операции.

Метод

unsigned int

CreateParts (const MbSolid & **solid**,
RPArray<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.

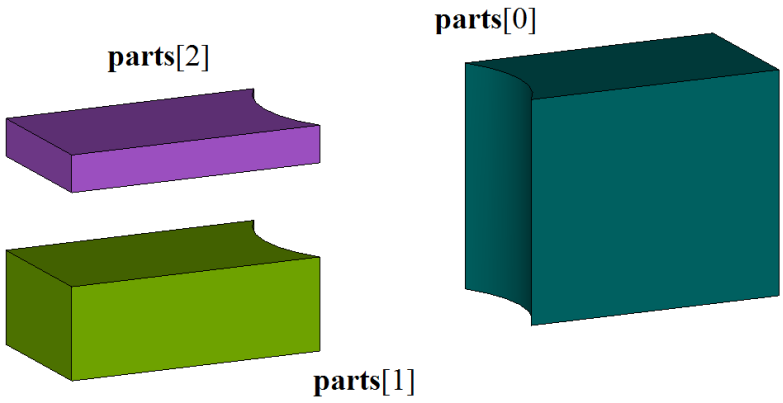


Рис. М.2.19.4.

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 cr_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 не связанной 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.

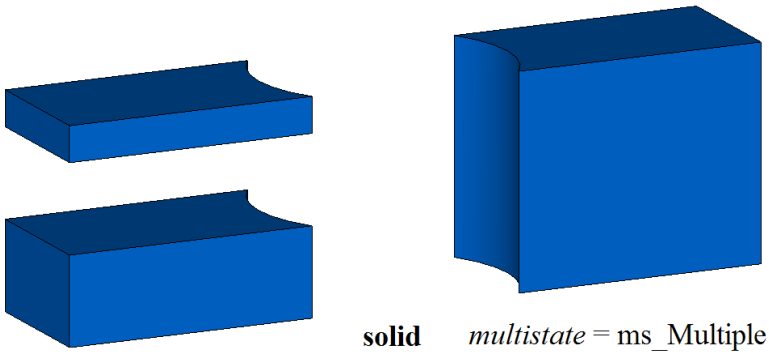


Рис. М.2.20.1.

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

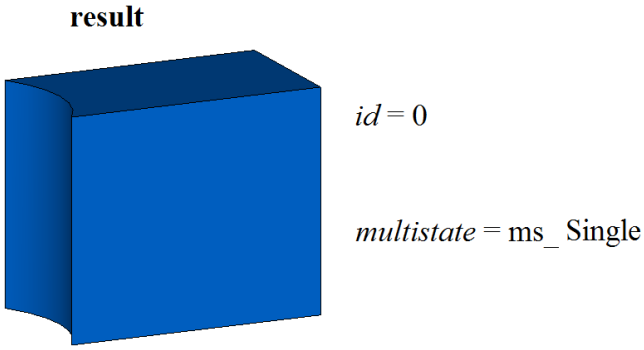


Рис. М.2.20.2.

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 cr_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 RPArray<MbSpaceItem> & items,
             bool  same,
             RPArray<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 – 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 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. 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].

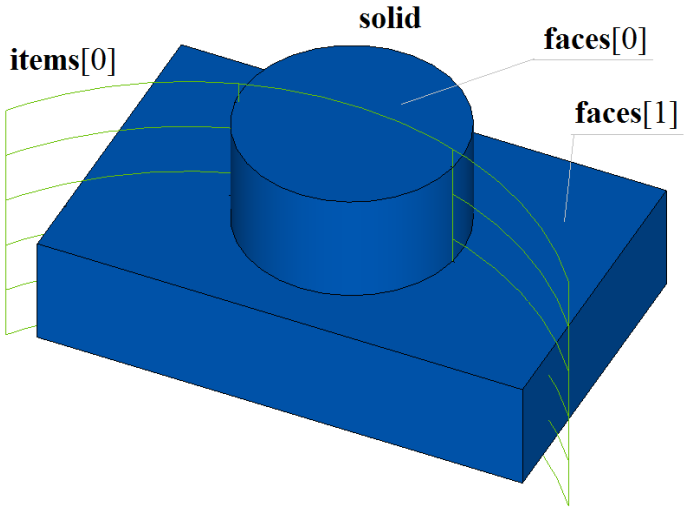


Рис. М.2.21.1.

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.

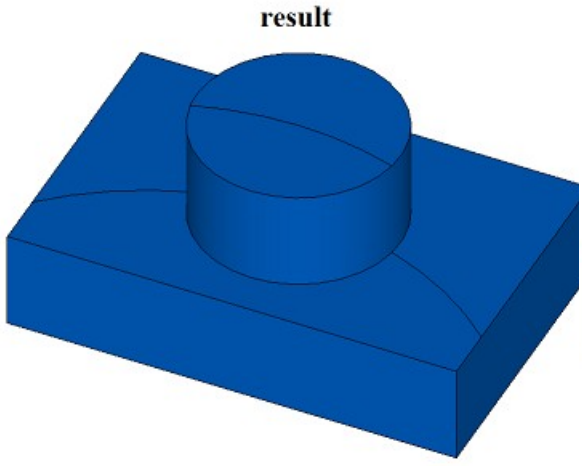


Рис. М.2.21.2.

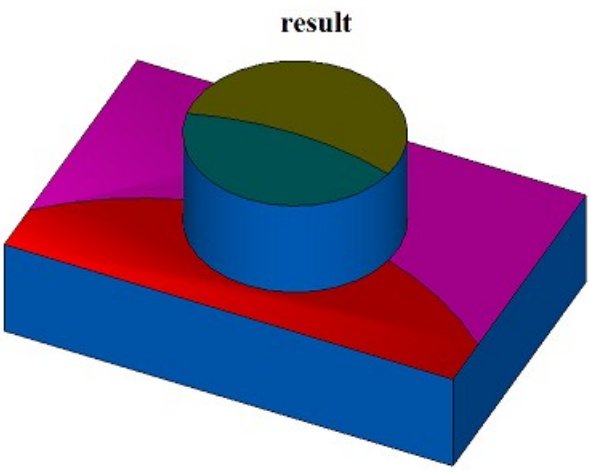


Рис. М.2.21.3.

Метод
MbResultType
SplitSolid (MbSolid & **solid**,
MbeCopyMode *sameShell*,
const MbPlacement3D & **place**,
MbeSenseValue *type*,
const RPArray<MbContour> & **contours**,
bool *same*,
RPArray<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 cr_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 – 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_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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому

после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе .

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.

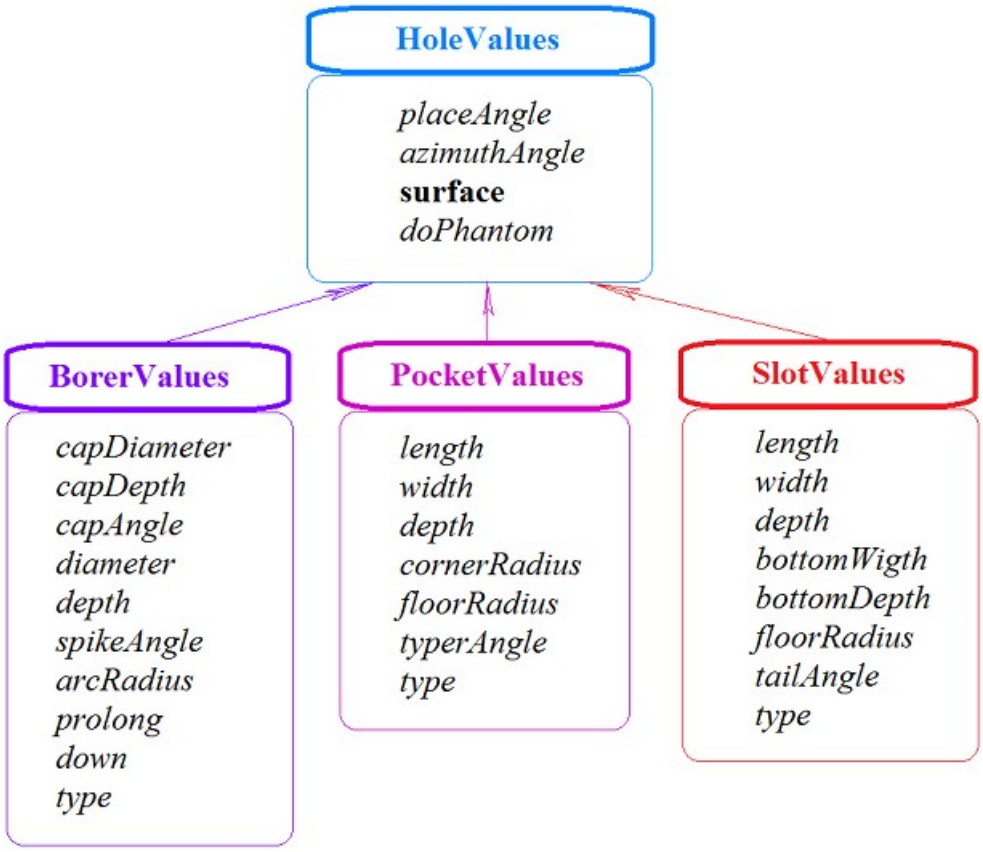
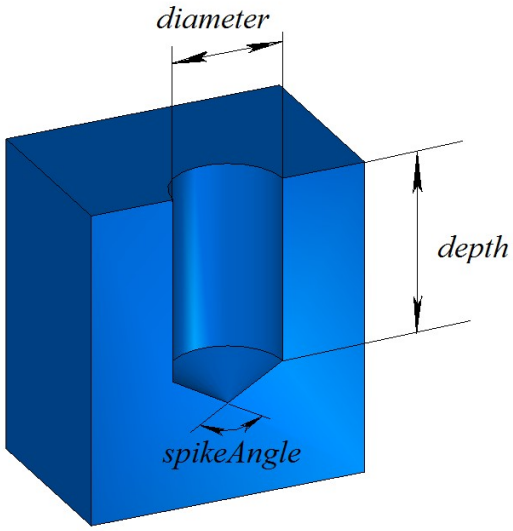


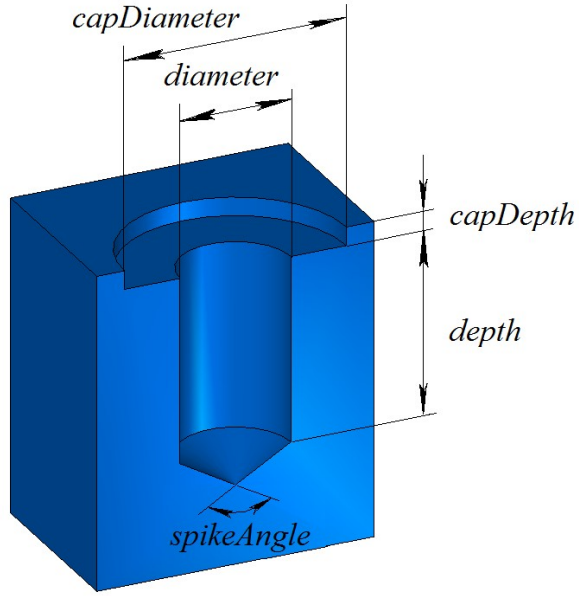
Рис. М.2.22.1.

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.



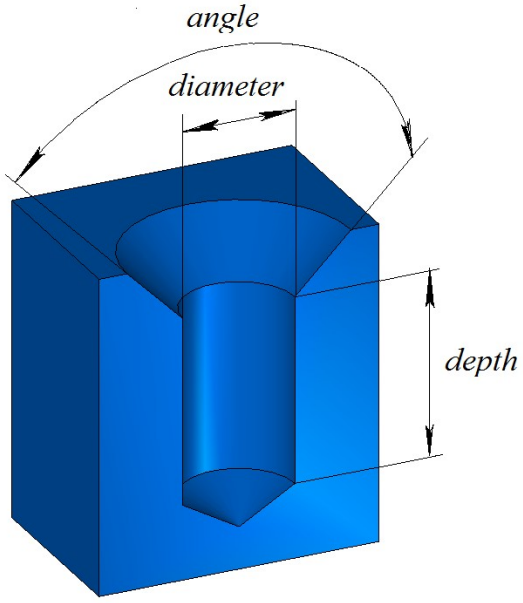
BorerValues::*type* = bt_SimpleCylinder

Рис. М.2.22.2.



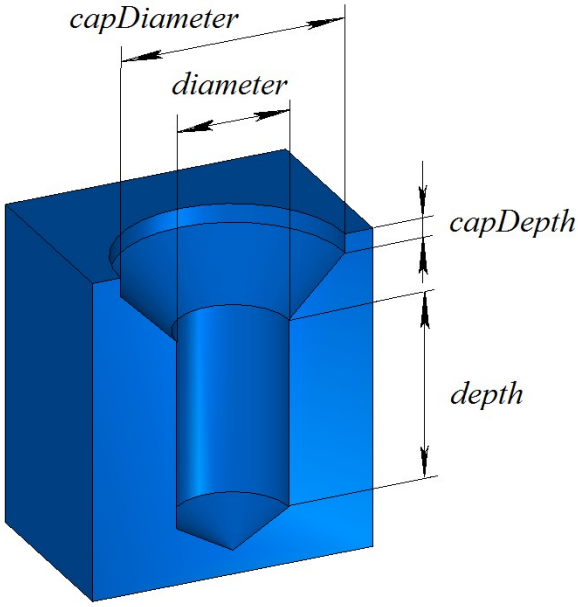
BorerValues::*type* = bt_TwofoldCylinder

Рис. М.2.22.3.



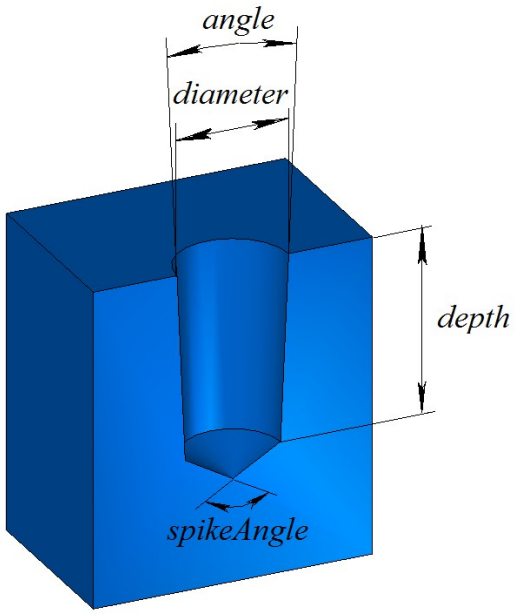
BorerValues::*type* = bt_ChamferCylinder

Рис. М.2.22.4.



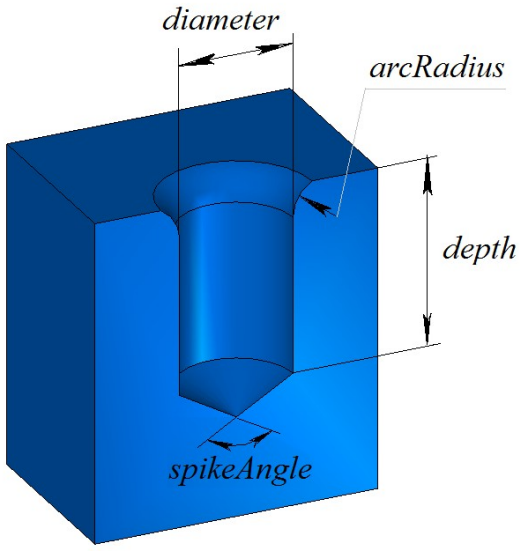
BorerValues::*type* = bt_ComplexCylinder

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.

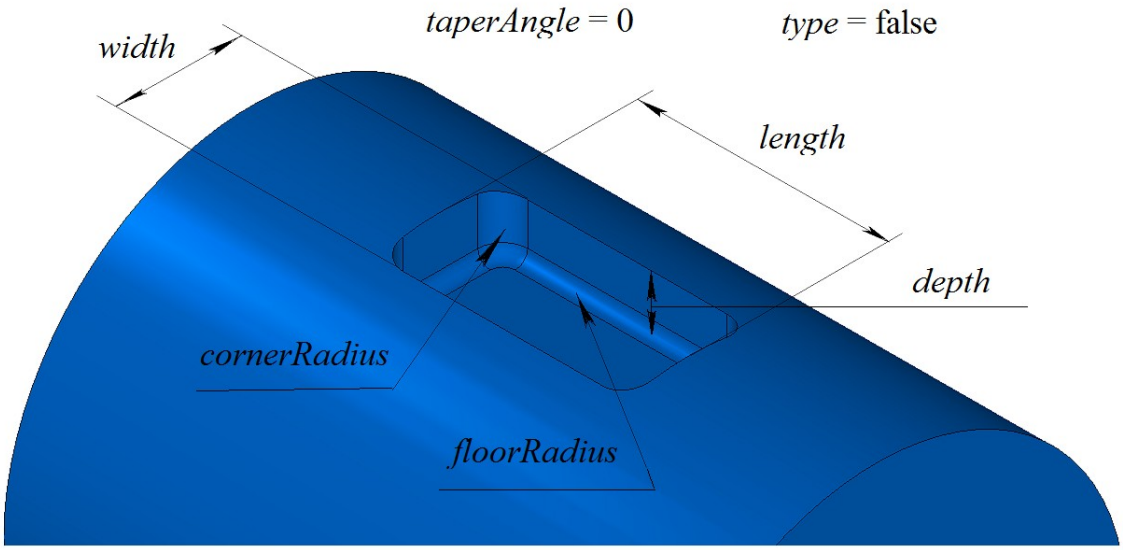


Рис. М.2.22.8.

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

Параметр перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.7.9. Копирование множества граней MbFaceShell.

На рис. М.2.23.1 показаны исходное тело **solid**, локальная система координат **place**, в плоскости XY которой расположен контур **contour**.

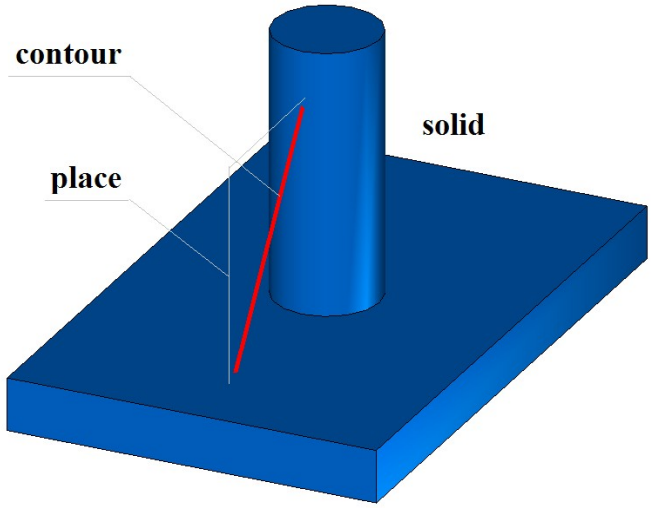


Рис. М.2.23.1.

На рис. М.2.23.2 приведен результат построения ребра жесткости без уклона боковых граней ребра жесткости.

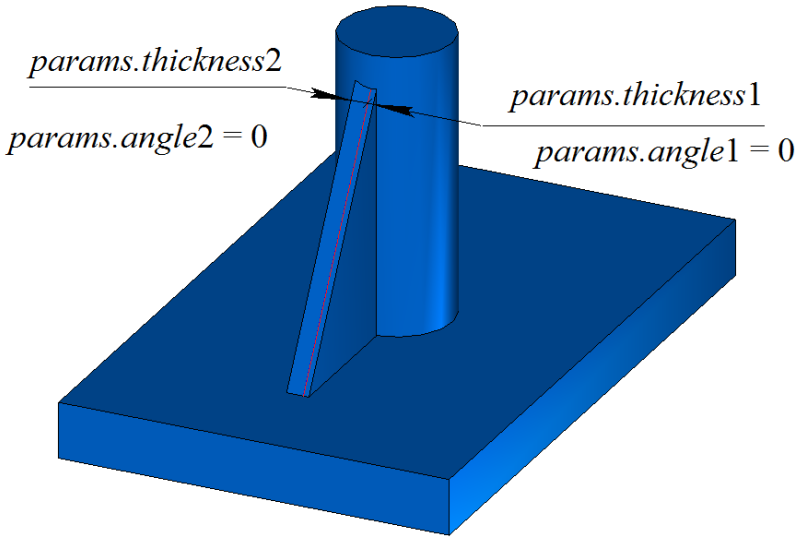


Рис. М.2.23.2.

На рис. М.2.23.3 приведен результат построения ребра жесткости с уклоном боковых граней ребра жесткости.

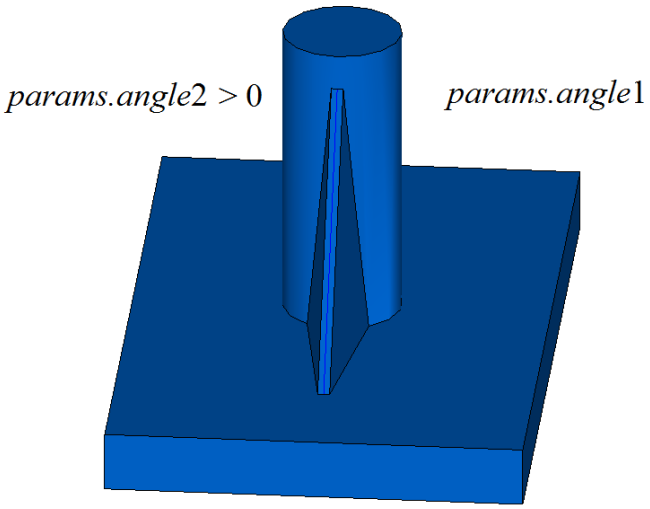


Рис. М.2.23.3.

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 RPArray<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 – 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 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 перечисление *sameShell* может принимать одно из четырех значений: *cm_Copy*, *cm_KeepSurface*, *cm_KeepHistory*, *cm_Same*. При значении *cm_Copy* исходное тело **solid** полностью копируется, поэтому построенное тело **result** и исходное тело **solid** не будут связаны между собой, то есть не будут иметь общих объектов. При значении *cm_KeepSurface* построенное тело **result** и исходное тело **solid** будут иметь одни и те же базовые поверхности граней. При значении *cm_KeepHistory* построенное тело **result** и исходное тело **solid** будут иметь одни и те же вершины, базовые поверхности граней и грани, не затронутые операцией. При значении *cm_Same* в построенное тело **result** будут перенесены все необходимые объекты исходного тела **solid**, поэтому после построения исходное тело должно быть удалено. Перечисление MbeCopyMode описано в параграфе О.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**.

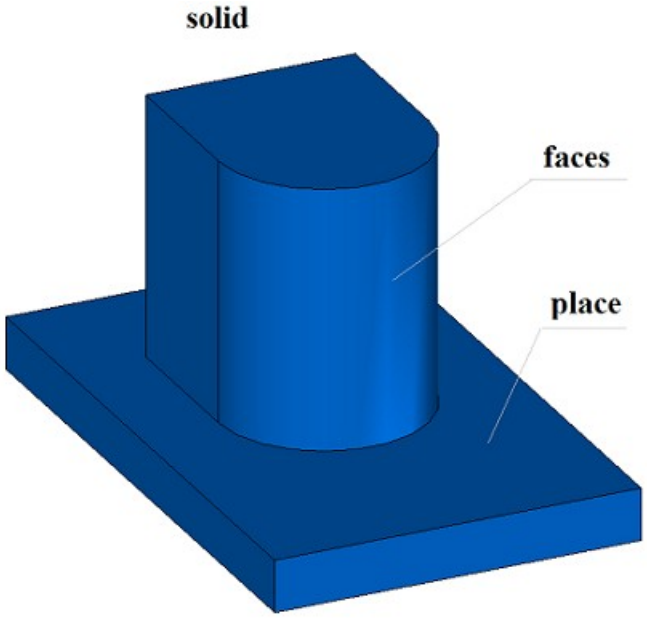


Рис. М.2.24.1.

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

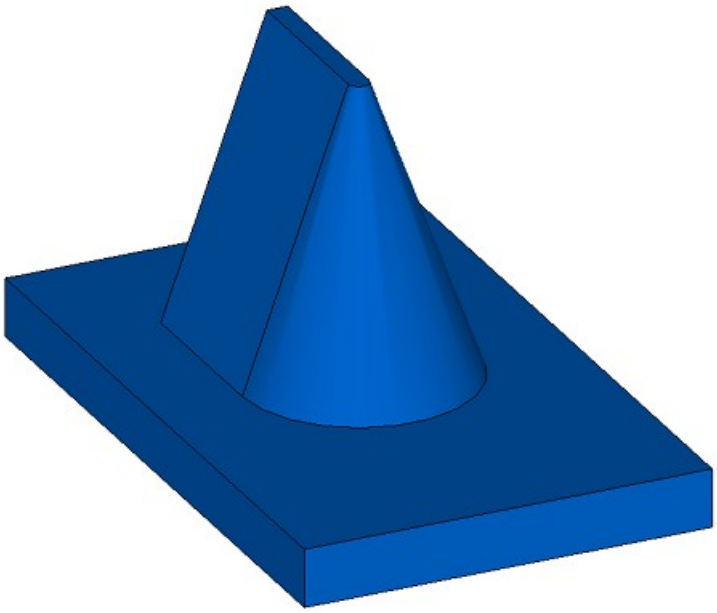


Рис. М.2.24.2.

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

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

М.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 – 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_solid.h.

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

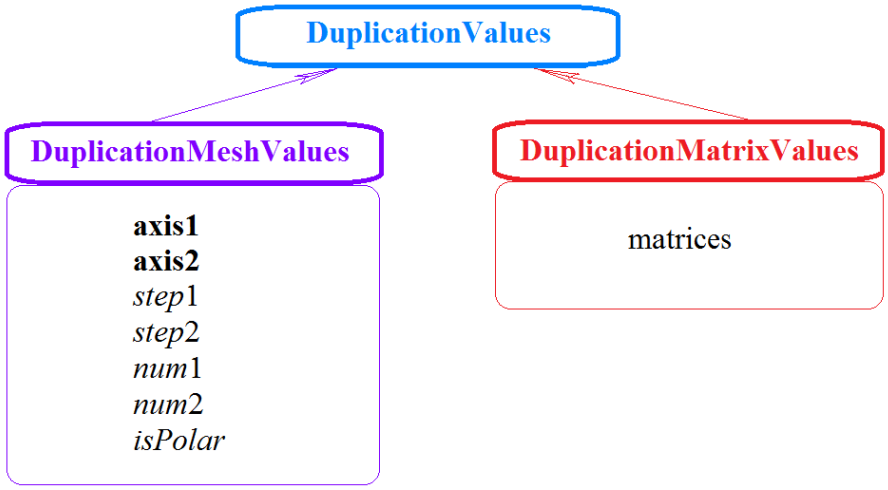
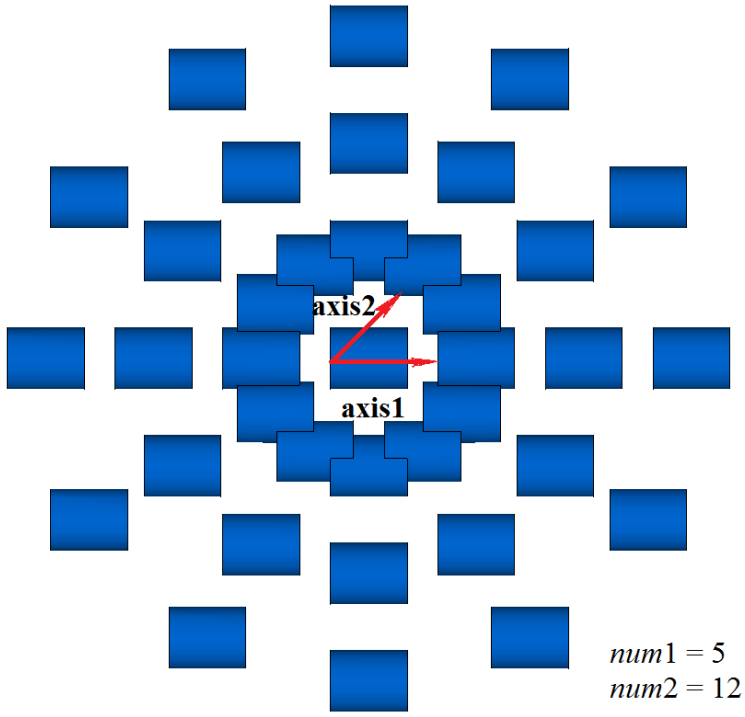


Рис. М.2.25.1.

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 **axis1** и **axis2**. Началом отсчета служит исходное тело. Вдоль оси **axis1** располагаются *num*1 копий тела с шагом *step*1, вдоль оси **axis2** располагаются *num*2 копий тела с шагом *step*2, включая исходное тело. Если *parameters.isPolar*=true, построенные копии располагаются в узлах полярной сетки. Началом отсчета служит исходное тело. Радиальное направление сетки определяется вектором **axis1**, а ось вращения определяется векторным произведением **axis1** и **axis2**. Вдоль радиальных направлений располагаются *num*1 копий тела с шагом *step*1, на каждой окружности располагаются *num*2 копий тела с угловым шагом *step*2.

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.



DuplicationMeshValues::*isPolar* = true

Рис. M.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 cr_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».

Часть О. ОБЪЕКТЫ ГЕОМЕТРИЧЕСКОГО ЯДРА C3D

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 куб, 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 некомпланарными векторами **axisX**, **axisY**, **axisZ**, рис. О.1.4.1.

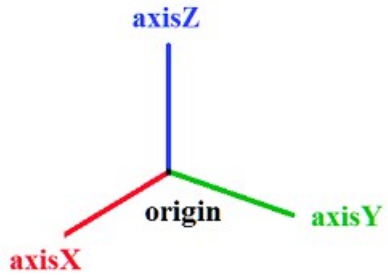


Рис. О.1.4.1.

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 афинной (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 **r** с координатами r_1, r_2, r_3 и базисными векторами **a**=[a_1 a_2 a_3], **b**=[b_1 b_2 b_3], **c**=[c_1 c_2 c_3]. Векторы **a**, **b**, **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 **M**. Заметим, что каждому базисному вектору **a**, **b**, **c** и начальной точке **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 куб в трёхмерном пространстве MbCube

Класс MbCube объявлен в файле mb_cube.h.

Габаритный куб MbCube описывает габариты протяжённого объекта (кривой, поверхности, тела, нескольких тел) в трёхмерном пространстве и определяется двумя точками **pmin** и **pmax**. Грани габаритного куба 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.

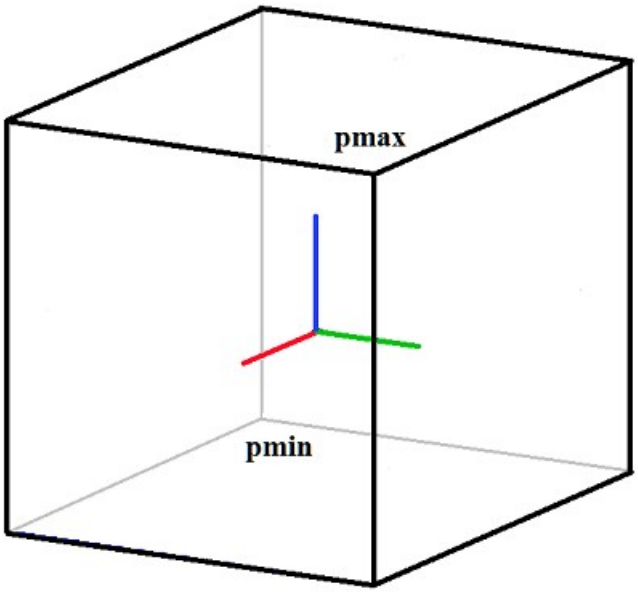


Рис. О.1.6.1.

Если габариты протяжённого объекта не определены, то габаритный куб 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,

$$\boldsymbol{vector} = [x \;\; y].$$

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 остью OX и нормализованным вектором. Таким образом, *ax*=cos(α), *ay*=sin(α), 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,

$$\boldsymbol{point} = [x \;\; y].$$

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

$$x = \frac{x_w}{w}, \;\; y = \frac{y_w}{w}.$$

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.

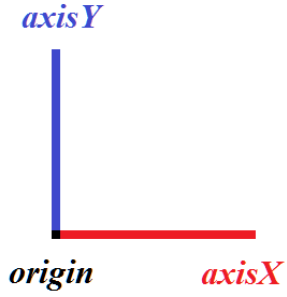


Рис. О.1.12.1.

В большинстве случаев система координат является правой, а векторы системы ортонормированы. 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 афинной (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 ***r*** с координатами r_1, r_2 и базисными векторами ***a***=[a_1 a_2], ***b***=[b_1 b_2]. Векторы ***a*** и ***b*** не должны быть колинеарными, но могут быть не ортогональными друг другу и иметь произвольную длину. Матрица MbMatrix преобразования координат из локальной системы в выбранную систему координат имеет вид

$$\boldsymbol{M} = \begin{bmatrix} a_1 & a_2 & 0 \\ b_1 & b_2 & 0 \\ r_1 & r_2 & 1 \end{bmatrix}.$$

Для обозначения расширенных матриц в двумерном пространстве будем использовать прописные буквы латинского алфавита, выделенные полужирным наклонным шрифтом, например ***M***. Заметим, что каждому базисному вектору ***a***, ***b*** и начальной точке ***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

$$\begin{bmatrix} p_1 & p_2 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & 0 \\ b_1 & b_2 & 0 \\ r_1 & r_2 & 1 \end{bmatrix}.$$

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

$$\begin{bmatrix} r_1 & r_2 & 0 \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & 0 \\ b_1 & b_2 & 0 \\ r_1 & r_2 & 1 \end{bmatrix}.$$

Заметим, что двумерный вектор умножается на расширенную матрицу MbMatrix справа.

О.1.14. Габаритный прямоугольник в двухмерном 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 прямоугольник а 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.

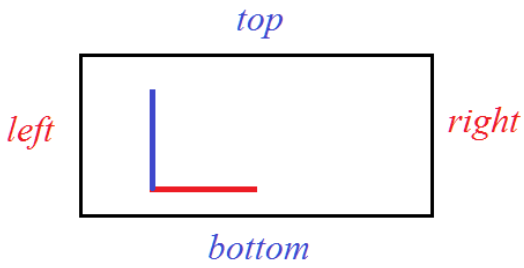


Рис. О.1.14.1.

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.

О.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.

О.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. О.2.1.1.

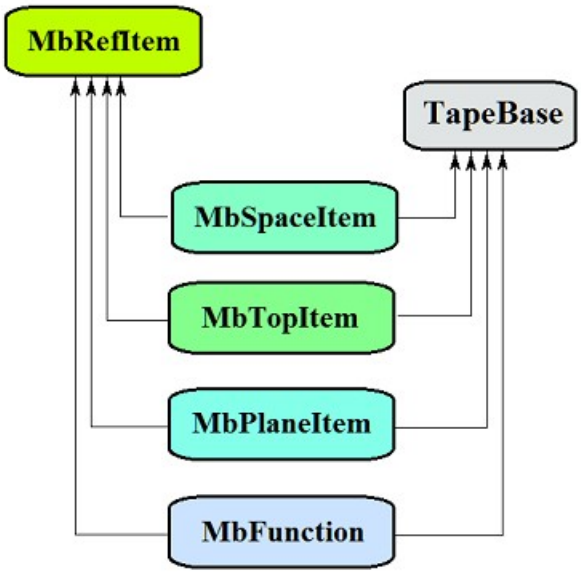


Рис О.2.1.1.

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.

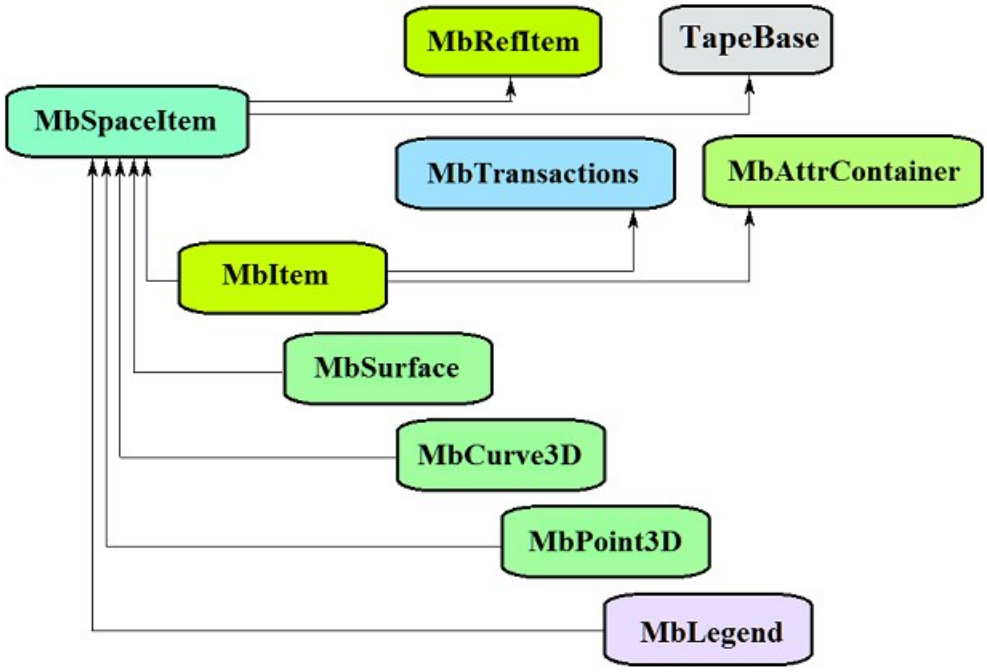


Рис О.2.2.1.

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

MbeSpaceType **IsA**(),
MbeSpaceType **Type**(),
MbeSpaceType **Family**().

возвращающими тип из перечисления трёхмерных 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.

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, TapeBase 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.

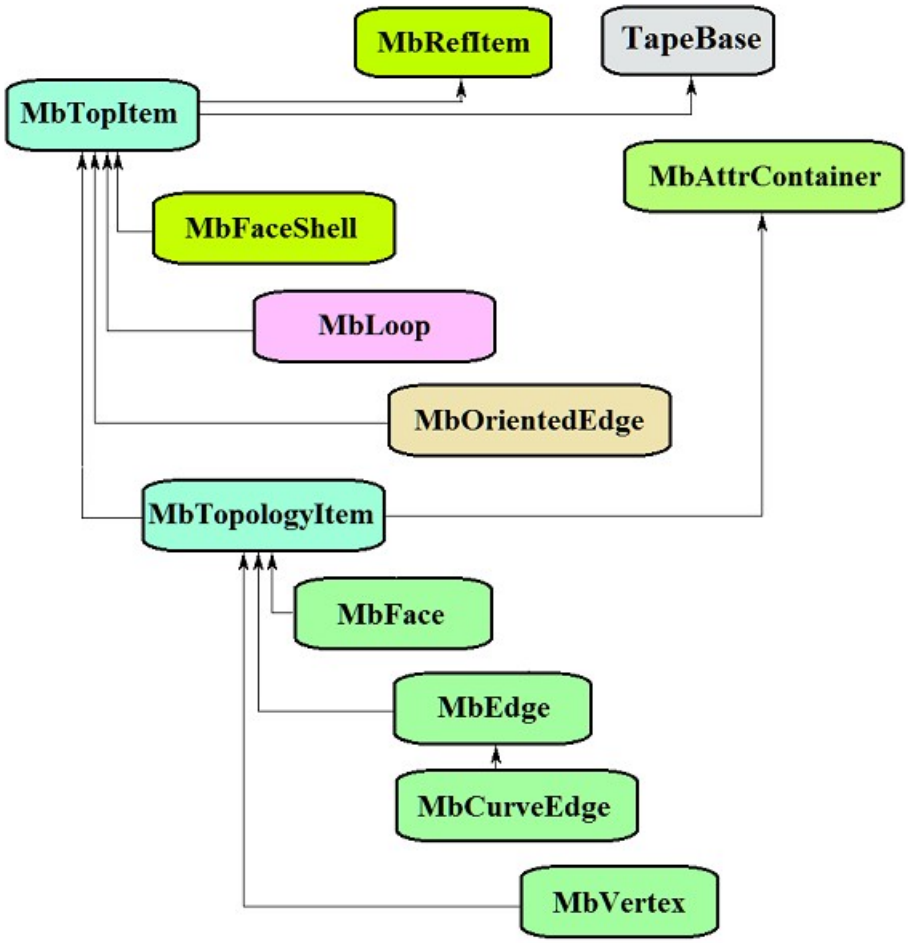


Рис О.2.3.1.

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.

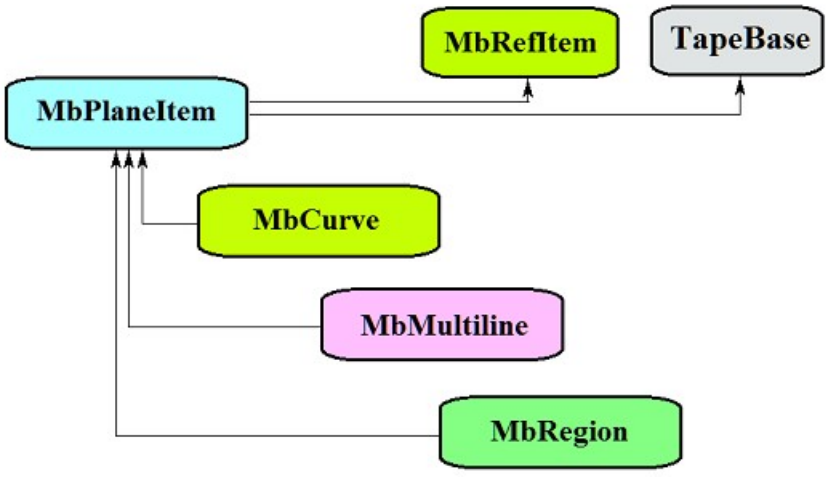


Рис О.2.4.1.

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.

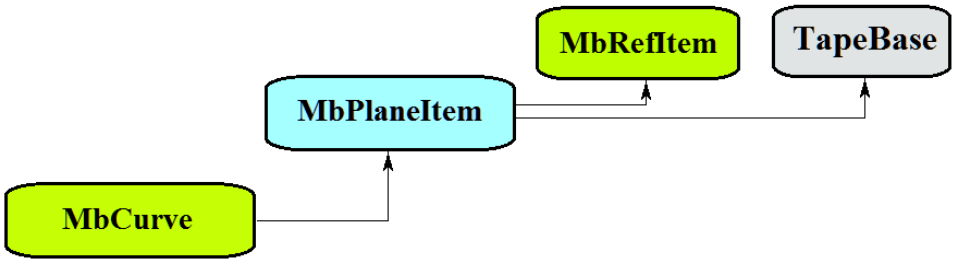


Рис. О.3.1.1.

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

$$\mathrm{curve}(t)=\begin{bmatrix} u(t) & v(t) \end{bmatrix}$$

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 & $\boldsymbol{r}_t$),
void **SecondDer**(double & t, MbVector & $\boldsymbol{r}_{tt}$),
void **ThirdDer**(double & t, MbVector & $\boldsymbol{r}_{ttt}$)
выдают соответственно первую $\boldsymbol{r}_t$, вторую $\boldsymbol{r}_{tt}$ и третью $\boldsymbol{r}_{ttt}$ 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 продолжаются циклически.
Методы
void **_FirstDer**(double t, MbVector & $\boldsymbol{r}_t$),
void **_SecondDer**(double t, MbVector & $\boldsymbol{r}_{tt}$),
void **_ThirdDer**(double t, MbVector & $\boldsymbol{r}_{ttt}$)
выдают соответственно первую $\boldsymbol{r}_t$, вторую $\boldsymbol{r}_{tt}$ и третью $\boldsymbol{r}_{ttt}$ 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,
- MbePlaneType **IsA**(),
- MbePlaneType **Type**(),
- MbePlaneType **Family**(),

методы, обеспечивающие выдачу и редактирование внутренних данных объекта,
- MbProperty & **CreateProperty**(MbePrompt 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.

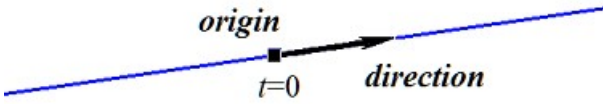


Рис. О.3.2.1.

В методе **PointOn**(double & *t*, MbCartPoint & ***r***) радиус-вектор прямой ***r*** описывается векторной функцией

$$\boldsymbol{r}(t) = \boldsymbol{origin} + t\,\boldsymbol{direction}.$$

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

$$\boldsymbol{r}(t) = (1 - t)\,\boldsymbol{point1} + t\,\boldsymbol{point2}.$$

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 *trim*1 и *trim*2 и направлением *sense*, заданными в локальной системе координат MbPlacement ***position***.

Углы *trim*1 и *trim*2 отсчитываются по дуге в направлении движения от вектора ***position.axisX*** к вектору ***position.axisY***. Углы *trim*1 и *trim*2 будем называть параметрами усечения. Значения параметров усечения, равные нулю и 2π, соответствуют точке на оси ***position.axisX***. Параметр кривой *t* принимает значения на отрезке: $0 \le t \le |trim2 - trim1|$. 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 *trim*1<*trim*2 и дуга строится от параметра *trimm*1 в сторону возрастания угла. Если *sense*=–1, то *trim*1>*trim*2 и дуга строится от параметра *trimm*1 в сторону уменьшения угла.

В методе **PointOn**(double & *t*, MbCartPoint & ***r***) радиус-вектор кривой ***r*** описывается векторной функцией

$$\boldsymbol{r}(t) = \boldsymbol{position.origin} + a\cos(trim1+(sense)t)\,\boldsymbol{position.axisX} + b\sin(trim1+(sense)t)\,\boldsymbol{position.axisY}.$$

Дуга эллипса приведена на рис. О.3.4.1.

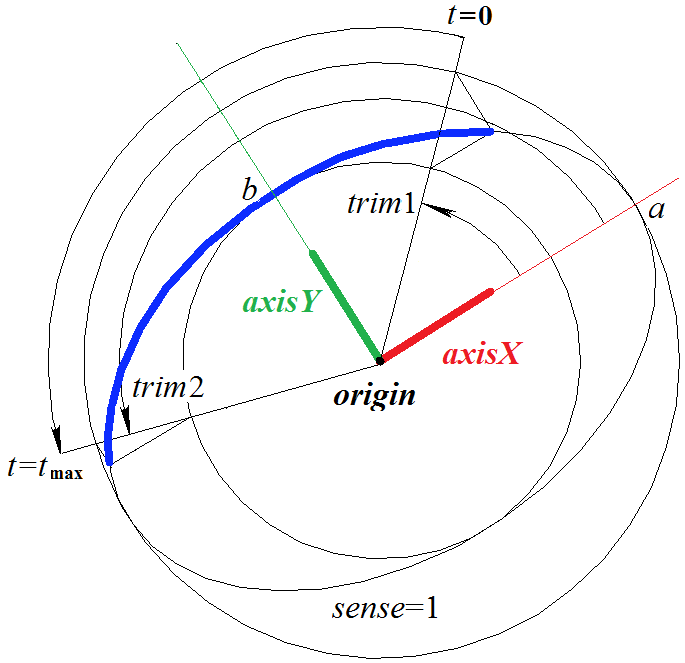


Рис. О.3.4.1.

Радиусы кривой должны быть больше нуля: $a>0$, $b>0$. Для параметров усечения должны соблюдаться неравенства: $trim1<trim2$ при $sense=1$ и $trim\ 1> trim\ 2$ 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 SArray<MbCartPoint>***pointList*** и признаком периодичности кривой *closed*.

Кривая проходит через множество точек ***pointList***[i], $i=0,...,segmentsCount$. при значениях параметра $t=0,...,segmentsCount$. 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: $0\le t\le segmentsCount$.

В методе **PointOn**(double & t, MbCartPoint & ***r***) радиус-вектор кривой ***r*** описывается векторной функцией

$$\boldsymbol{r}(t) = \boldsymbol{pointList}[i]\,(1-w) + \boldsymbol{pointList}[i+1]\,w,$$

где $w=\dfrac{t-t_i}{t_{i+1}-t_i}$, а $t_i\le t\le t_{i+1}$. 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.

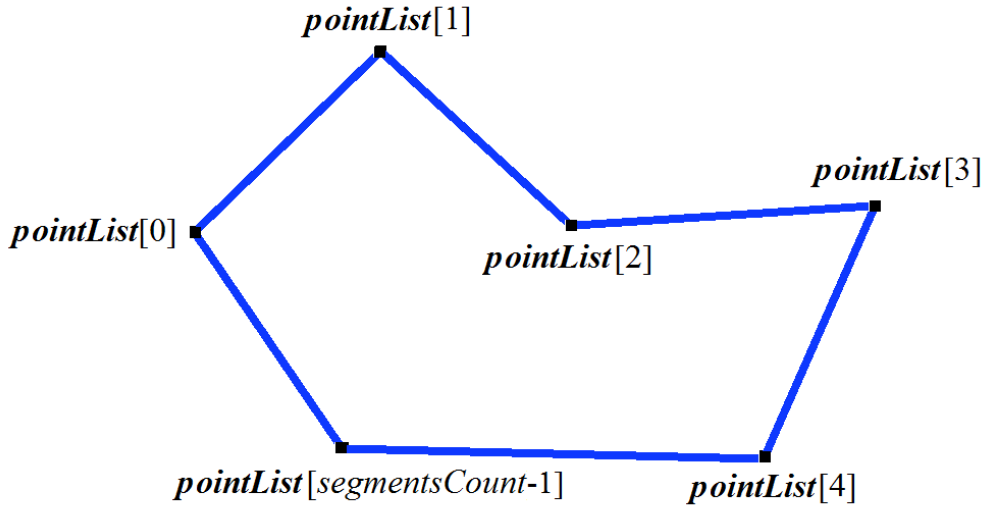


Рис. О.3.5.1.

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.

B-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 B-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 B-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*+2*degree*–1.

В методе **PointOn**(double & *t*, MbCartPoint & ***r***) радиус-вектор кривой ***r*** описывается векторной функцией

$$r(t)=\frac{\sum_{j=0}^{pointsCount-1} N_{j}^{degree}(t)\,weight[j]\,\mathrm{pointList}[j]}{\sum_{j=0}^{pointsCount-1} N_{j}^{degree}(t)\,weight[j]},$$

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.

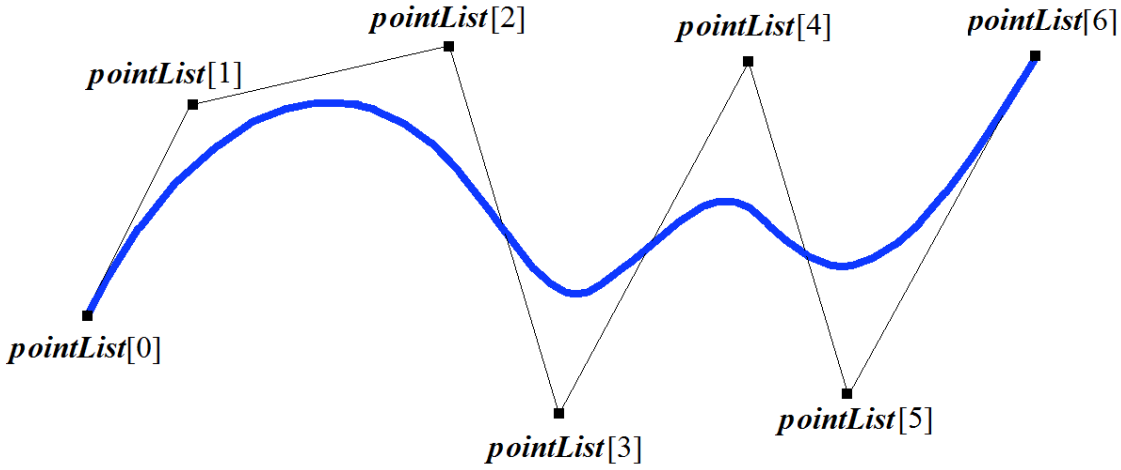


Рис. О.3.6.1.

Кривая может быть периодической. Периодическая NURBS-кривая приведена на рис. О.3.6.2.

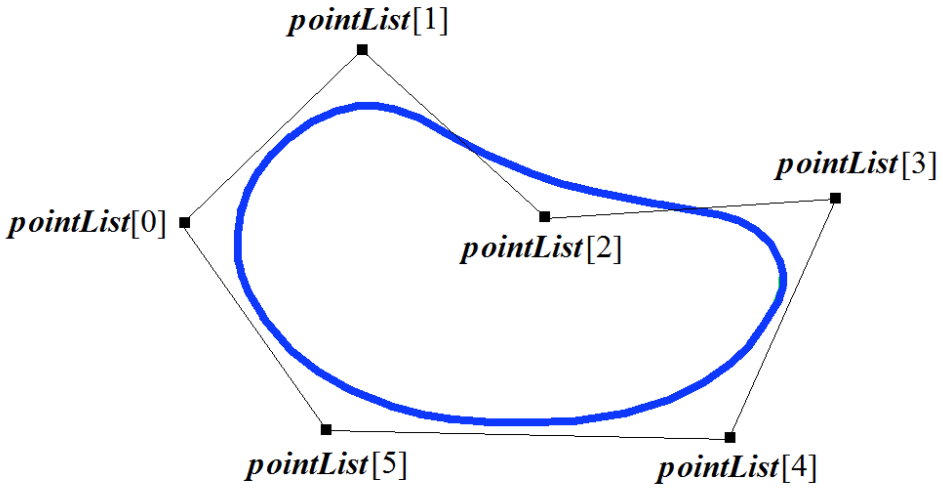


Рис. О.3.6.2.

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*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,...,*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,...,*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*]≤*t*≤*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*:

$$r(t)=(1-3w^2+2w^3)\,\text{pointList}[i]+(3w^2+2w^3)\,\text{pointList}[i+1]+$$
$$+\big((w-2w^2+w^3)\,\text{vectorList}[i]+(-w^2+w^3)\,\text{vectorList}[i+1]\big)\big(tList[i+1]-tList[i]\big)\ ,$$

где $w=\dfrac{t-tList[i]}{tList[i+1]-tList[i]}$, а *tList*[*i*]≤*t*≤*tList*[*i*+1]. Кривая Эрмита приведена на рис. О.3.7.1.

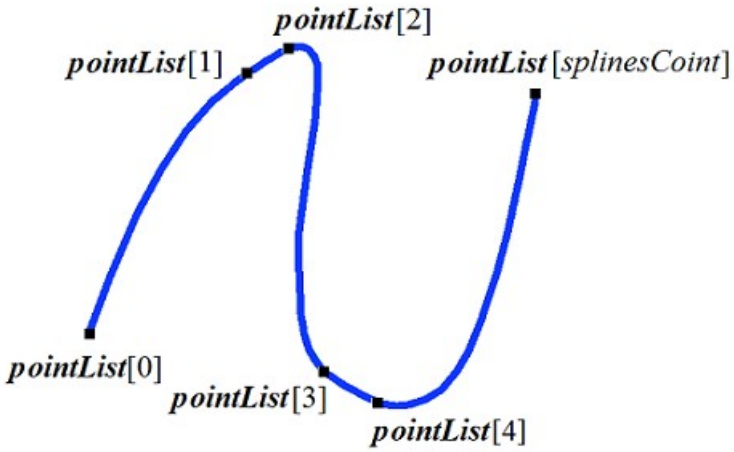


Рис. О.3.7.1.

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*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,...,*splinesCount*, изменяются пропорционально расстоянию между точками, а производные ***vectorList***[*i*], *i*=1,2,...,*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(*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*, MbCartPoint & ***r***) радиус-вектор кривой ***r*** описывается векторной функцией найденного участка для его локального параметра *w*:

$$r(t)=\sum_{j=0}^{3}\frac{3!}{j!(3-j)!}w^{j}(1-w)^{3-j}\,\mathrm{pointList}[3n+j]\,,$$

где $w=t-n$, $n\le t\le n+1$, $0\le w\le 1$, $B_j^3(w)=\frac{3!}{j!(3-j)!}w^j(1-w)^{3-j}$ – функции Бернштейна третьей степени для *j*–ой, *j*=0,1,2,3, контрольной точки ***pointList***[3*n*+*j*] найденного участка с номером *n*.. Составная кривая Безье приведена на рис. О.3.8.1.

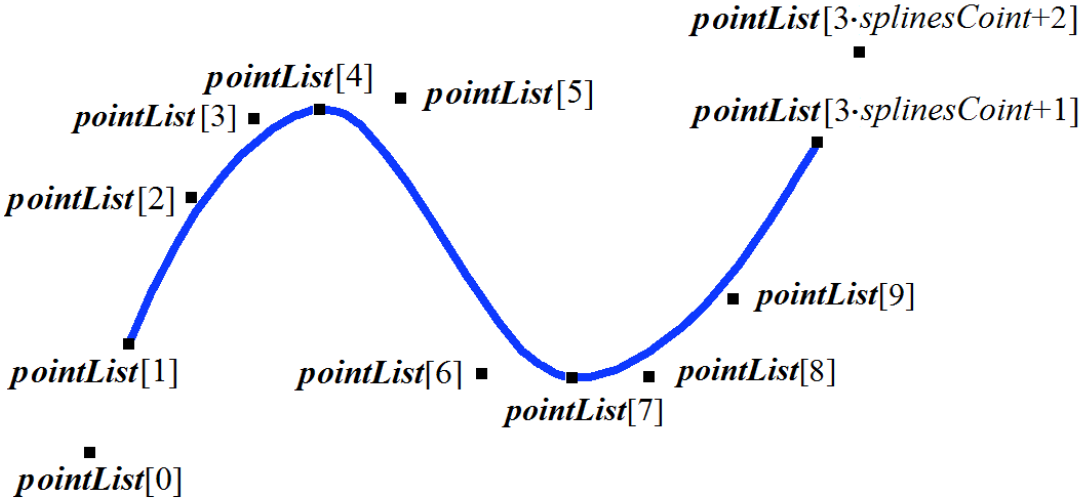


Рис. О.3.8.1.

Параметр кривой *t* принимает значения на отрезке: $0\le t\le splinesCount$. Кривая может быть периодической. Период периодической кривой равен *splinesCount*.

При целочисленных значениях параметра кривая проходит через контрольные точки, например, при параметре *t=n* кривая пройдёт через контрольную точку ***pointList***[3*n*], *n*=0,1,...,*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,...,*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*]≤*t*≤*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

$$r(t)=(1-w)\,\boldsymbol{pointList}[i]+w\,\boldsymbol{pointList}[i+1]+$$
$$+\left((-2w+3w^2-w^3)\,\mathrm{vectorList}[i]+(-w+w^3)\,\boldsymbol{vectorList}[i+1]\right)\frac{(tList[i+1]-tList[i])^2}{6},$$

где $w=\dfrac{t-tList[i]}{tList[i+1]-tList[i]}$, а $tList[i]\le t\le tList[i+1]$. Кубический сплайн, построенный по тем же контрольным точкам, что и составная кривая Эрмита, приведён на рис. О.3.9.1.

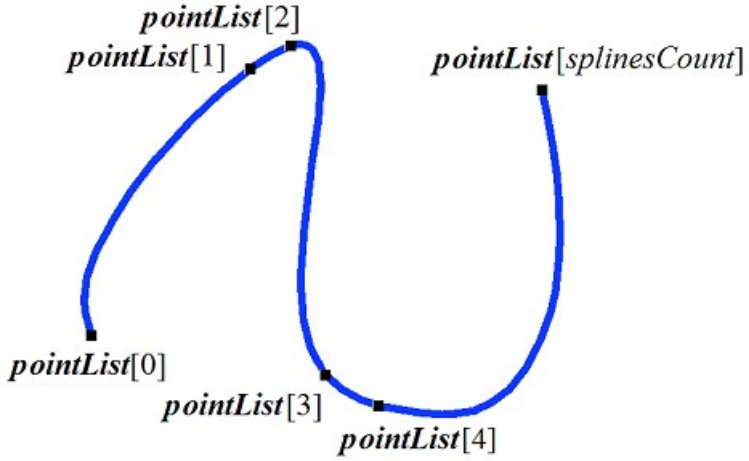


Рис. О.3.9.1.

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*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,...,*splinesCount*, изменяются пропорционально расстоянию между точками, а вторые производные ***vectorList***[*i*], *i*=1,2,...,*splinesCount*–1 вычисляются путём решение 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 *trim*1, конечным параметром усечения базовой кривой *trim*2 и признаком совпадения направлений базовой кривой и усечённой кривой *sense*.

Усечённая кривая совпадает с базовой кривой на участке, определённом параметрами *trim*1 и *trim*2, но может иметь противоположное с ним направление. Если *sense*=1, то *trim*1<*trim*2 и усечённая кривая совпадает по направлению с базовой кривой. Если *sense*=–1, то *trim*2<*trim*1 и усечённая кривая направлена против базовой кривой.

В методе **PointOn**(double & *t*, MbCartPoint & ***r***) радиус-вектор кривой ***r*** описывается векторной функцией

$$\boldsymbol{r}(t) = \boldsymbol{basisCurve}(trim1+sense\cdot t).$$

Усеченная кривая приведена на рис. О.3.10.1.

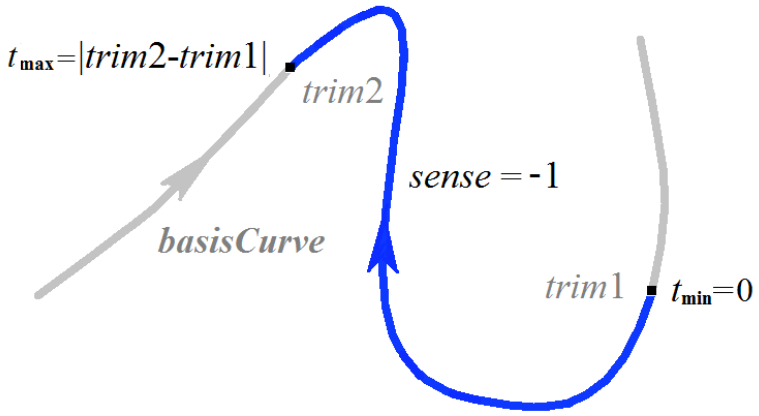


Рис. О.3.10.1.

Параметр кривой *t* принимает значения на отрезке: $0 \le t \le sense(trim2 - trim1)$.

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 *trim*2=*trim*1+*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 *t*1, double *t*2, 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 *tmin* 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

$$\boldsymbol{r}(t) = \boldsymbol{basisCurve}(v(t)),$$

где $v(t) = b\min\frac{trim2 - t}{trim2 - trim1} + b\max\frac{t - trim1}{trim2 - trim1}$, *bmin*, *bmax* – граничные значения области определения параметра базовой кривой.

Параметр кривой *t* принимает значения на отрезке: $tmin \le t \le tmax$.

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.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

$$\boldsymbol{r}(t) = \boldsymbol{basisCurve}(t) + \boldsymbol{normal}(t) \cdot distance,$$

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.

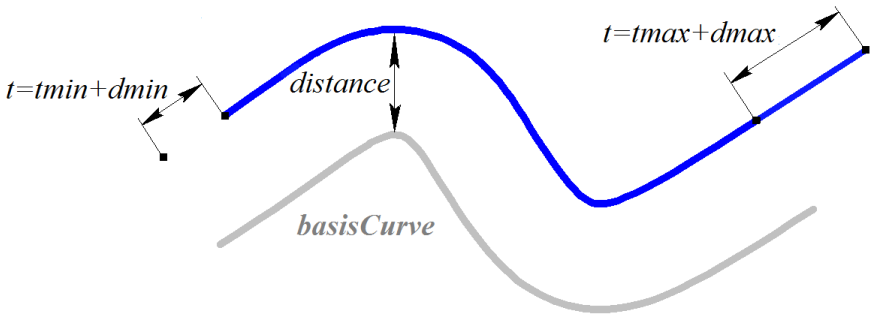


Рис. О.3.12.1.

Параметр кривой *t* принимает значения на отрезке: *tmin+dmin*≤*t*≤*tmax+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*≤*t*≤*tmax*.

В методе **PointOn**(double & *t*, MbCartPoint & ***r***) радиус-вектор кривой ***r*** описывается векторной функцией

$$\boldsymbol{r}(t) = [xFunction(t) \quad yFunction(t)].$$

Символьная кривая приведена на рис. О.3.13.1.

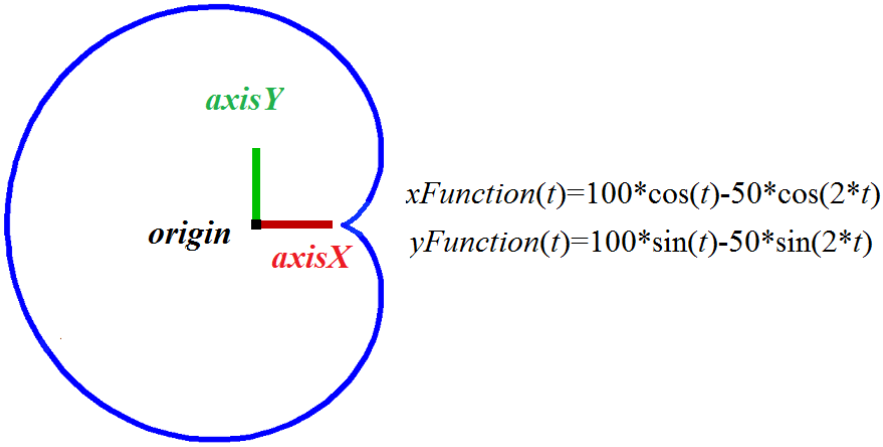


Рис. О.3.13.1.

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

$$\boldsymbol{r}(t) = \boldsymbol{position.origin} + (((tmin+t-phase) / frequency)\ \boldsymbol{position.axisX}) + (amplitude \cos(tmin+t)\ \boldsymbol{position.axisY}).$$

Косинусоида приведена на рис. О.3.14.1.

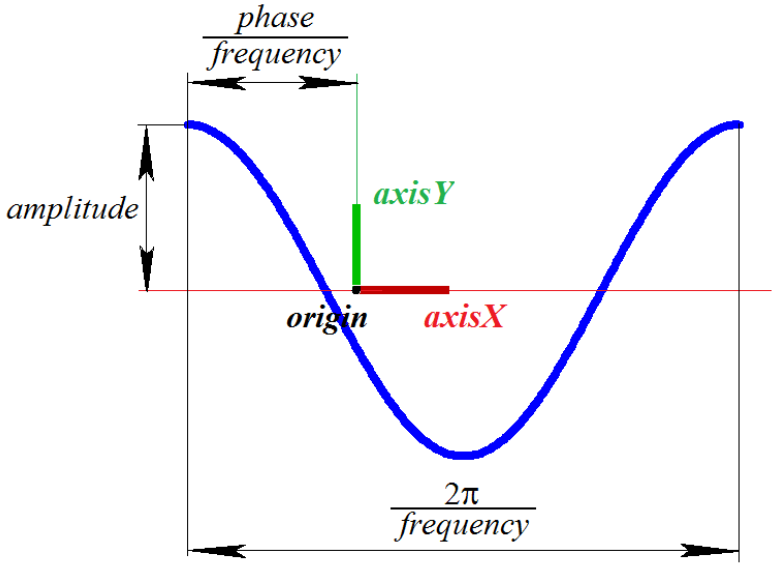


Рис. О.3.14.1.

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*tmax*. Для параметров должно соблюдаться неравенство: *tmin*<*tmax*. 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.

О.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 *tmin*, максимальным параметром кривой *tmax* и признаком периодичности кривой *closed.*

В методе **PointOn**(double & *t*, MbCartPoint & ***r***) радиус-вектор кривой ***r*** описывается векторной функцией

$$\boldsymbol{r}(t) = \boldsymbol{point}.$$

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*tmax*. Кривая может быть периодической. Для параметров должно соблюдаться неравенство: *tmin*<*tmax*.

Двумерная кривая-точка используется в паре с другой двумерной кривой для описания пересечения поверхностей, одна из которых имеет особую точку, например, полюс. Параметры *tmin*, *tmax*, *closed* кривой-точки соответствуют параметрам двумерной кривой, используемой в паре с кривой-точкой.

О.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 точках. 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

$$\boldsymbol{r}(t) = [u \;\; v],$$

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

$$\begin{aligned} \textbf{deriveU} \cdot (\textbf{spaceCurve}(t) - \textbf{surface}(\, u, v\,)) = 0, \\ \textbf{deriveV} \cdot (\textbf{spaceCurve}(t) - \textbf{surface}(\, u, v\,)) = 0, \end{aligned}$$

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.

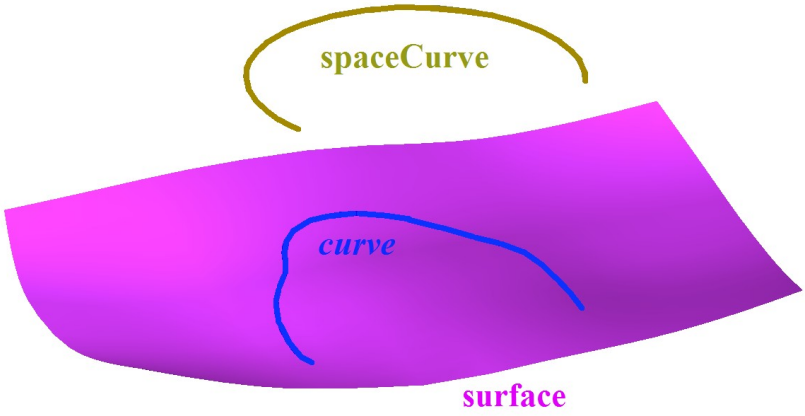


Рис. О.3.16.1.

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 RPArray<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: $t_{max}=\sum\left(w_{i\max}-w_{i\min}\right)$, где w_{imin} и w_{imax} – 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

$$\boldsymbol{r}(t) = \boldsymbol{segments}[k](w_k),$$

где ***segments***$[k](w_k)$ – рабочий сегмент контура с индексом k, w_k – параметр рабочего сегмента, равный: $w_k=w_{k\min}+t-\sum_{i=0}^{k-1}\left(w_{i\max}-w_{i\min}\right)$. Сегмент с индексом k определяется по значению параметра контура t из условия $\sum_{i=0}^{k-1}\left(w_{i\max}-w_{i\min}\right)\le t<\sum_{i=0}^{k}\left(w_{i\max}-w_{i\min}\right)$, где w_{imin} и w_{imax} – минимальное и максимальное значение параметра i–го сегмента. Контур приведён на рис. О.3.17.1.

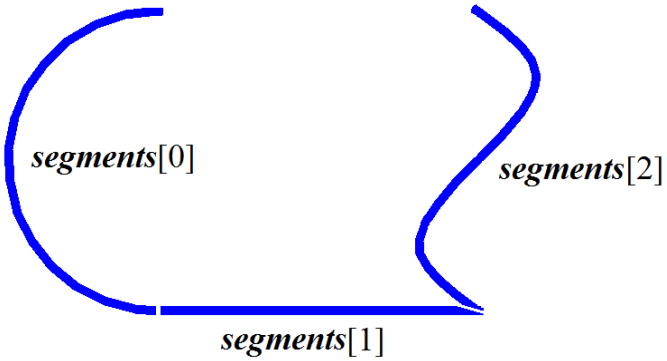


Рис. О.3.17.1.

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.

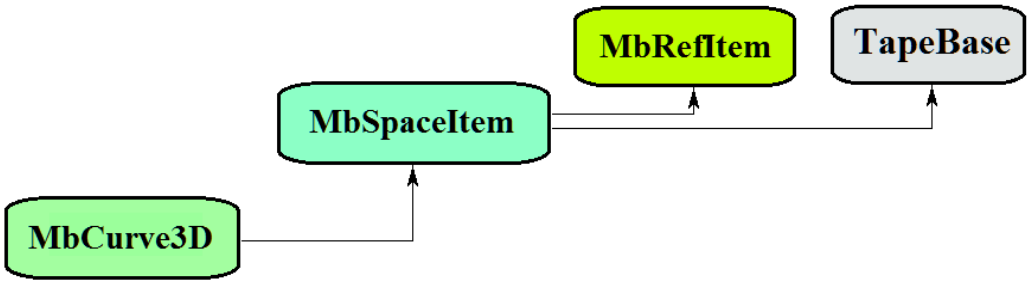


Рис. О.4.1.1.

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

$$\mathbf{curve}(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}$$

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 **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, MbCartPoint3D & **r**).
Он выдаёт радиус-вектор **r** точки кривой для заданного параметра t. Методы
void **FirstDer**(double & t, MbVector3D & $\mathbf{r}_t$),
void **SecondDer**(double & t, MbVector3D & $\mathbf{r}_{tt}$),
void **ThirdDer**(double & t, MbVector3D & $\mathbf{r}_{ttt}$)
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 **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 продолжаются циклически. Методы
void **_FirstDer**(double t, MbVector3D & $\mathbf{r}_t$),
void **_SecondDer**(double t, MbVector3D & $\mathbf{r}_{tt}$),
void **_ThirdDer**(double t, MbVector3D & $\mathbf{r}_{ttt}$)
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 MbLinet3D 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.

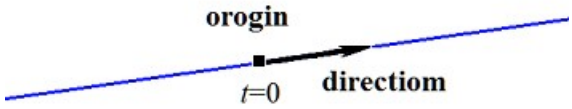


Рис. О.4.2.1.

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 *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.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 *trim*1 и *trim*2, заданными в локальной системе координат MbPlacement3D **position**.

Углы *trim*1 и *trim*2 отсчитываются по дуге в направлении движения от вектора **position.axisX** к вектору **position.axisY**. Углы *trim*1 и *trim*2 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: $0 \le t \le trim2 - trim1$. 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

$$\mathbf{r}(t) = \mathbf{position.origin} + a\cos(trim1+t)\,\mathbf{position.axisX} + b\sin(trim1+t)\,\mathbf{position.axisY}.$$

Дуга эллипса приведена на рис. О.4.4.1.

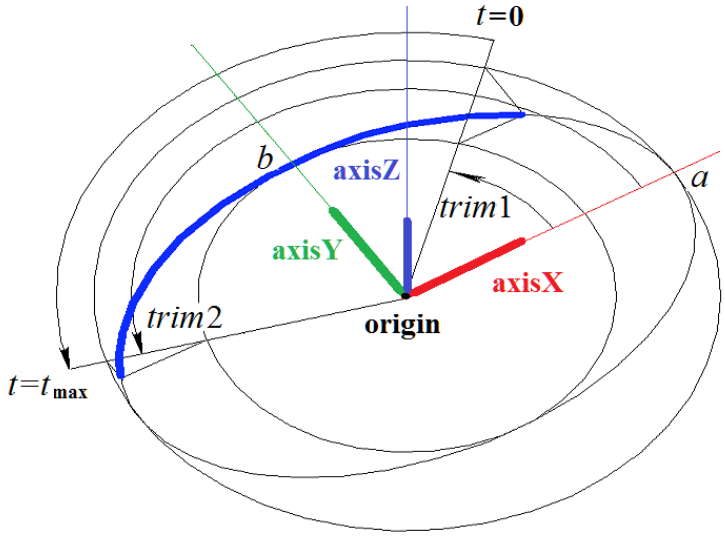


Рис. О.4.4.1.

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], $i=0,...,segmentsCount$. при значениях параметра $t=0,...,segmentsCount$. 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: $0\le t\le segmentsCount$.

В методе **PointOn**(double & t, MbCartPoint3D & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{pointList}[i]\,(1-w) + \mathbf{pointList}[i+1]\,w,$$

где $w = \dfrac{t - t_i}{t_{i+1} - t_i}$, а $t_i\le t\le t_{i+1}$. Ломаная является простейшей кривой, построенной по набору точек. Она состоит из отрезков, последовательно соединяющих контрольные точки. Ломаная приведена на рис. О.4.5.1.

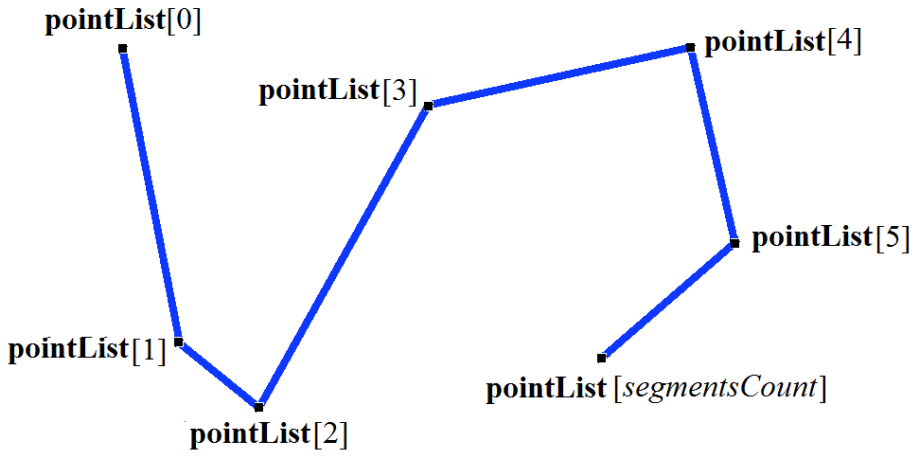


Рис. О.4.5.1.

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 B-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 B-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*+2*degree*–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.

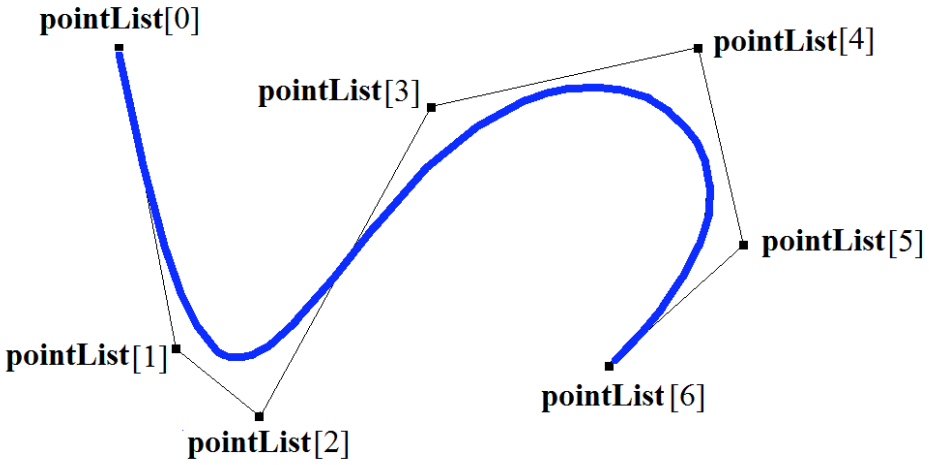


Рис. О.4.6.1.

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*tmax*, где *tmin*=*knots*[*degree*–1], *tmax*=*knots*[*knotsCount*–*degree*]. Кривая может быть периодической. Периодическая NURBS-кривая приведена на рис. О.4.6.2.

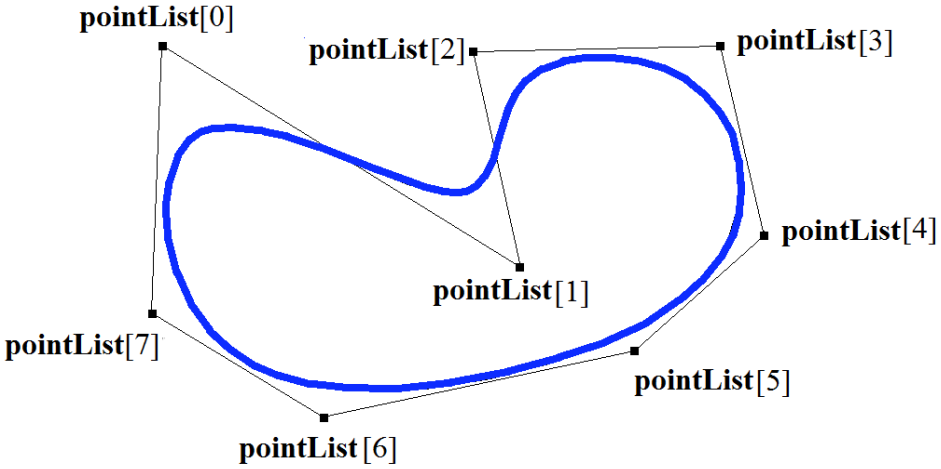


Рис. О.4.6.2.

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,...,*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,...,*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*]≤*t*≤*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*:

$$\mathbf{r}(t) = (1-3w^2+2w^3)\mathbf{pointList}[i] + (3w^2+2w^3)\mathbf{pointList}[i+1] +$$
$$+\left((w-2w^2+w^3)\mathbf{vectorList}[i] + (-w^2+w^3)\mathbf{vectorList}[i+1]\right)(tList[i+1]-tList[i]),$$

где $w = \dfrac{t-tList[i]}{tList[i+1]-tList[i]}$, а *tList*[*i*]≤*t*≤*tList*[*i*+1]. Кривая Эрмита приведена на рис. О.4.7.1.

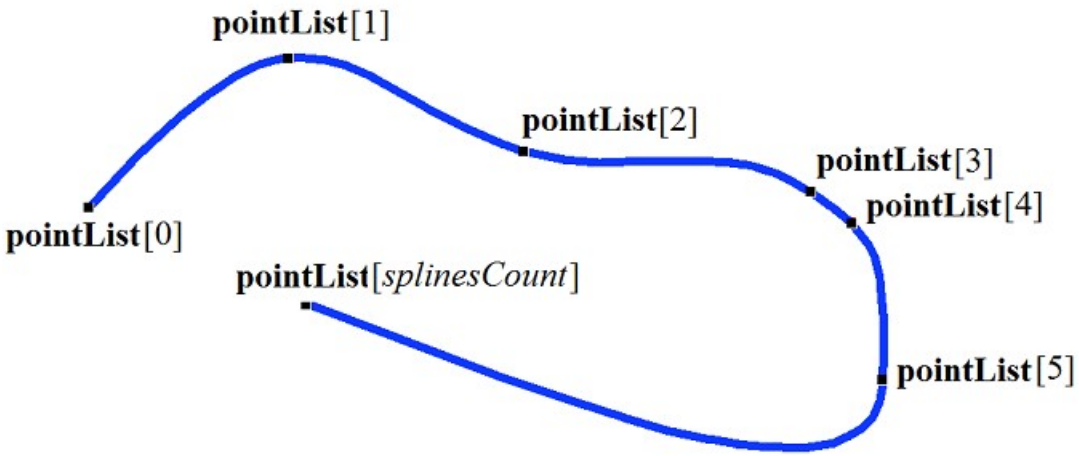


Рис. О.4.7.1.

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*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,...,*splinesCount*, изменяются пропорционально расстоянию между точками, а производные **vectorList**[*i*], *i*=1,2,...,*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 $w=t-n$, $n\le t\le n+1$, $0\le w\le 1$, $B_j^3(w) = \frac{3!}{j!(3-j)!} w^j (1-w)^{3-j}$ – 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.

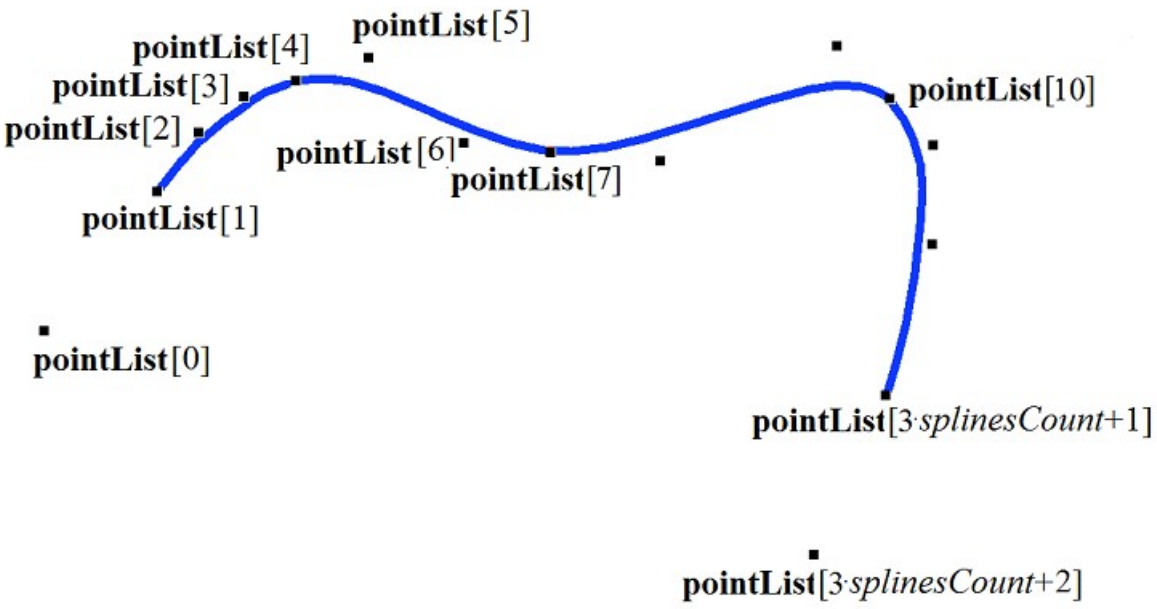


Рис. О.4.8.1.

Параметр кривой t принимает значения на отрезке: $0\le t\le splinesCount$. Кривая может быть периодической. Период периодической кривой равен *splinesCount*.

При целочисленных значениях параметра кривая проходит через контрольные точки, например, при параметре $t=n$ кривая пройдёт через контрольную точку **pointList**[3n], n=0,1,...,*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,...,*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*]≤*t*≤*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 $w = \dfrac{t - tList[i]}{tList[i+1] - tList[i]}$, а *tList*[*i*]≤*t*≤*tList*[*i*+1]. Кубический сплайн, построенный по тем же контрольным точкам, что и составная кривая Эрмита, приведён на рис. О.4.9.1.

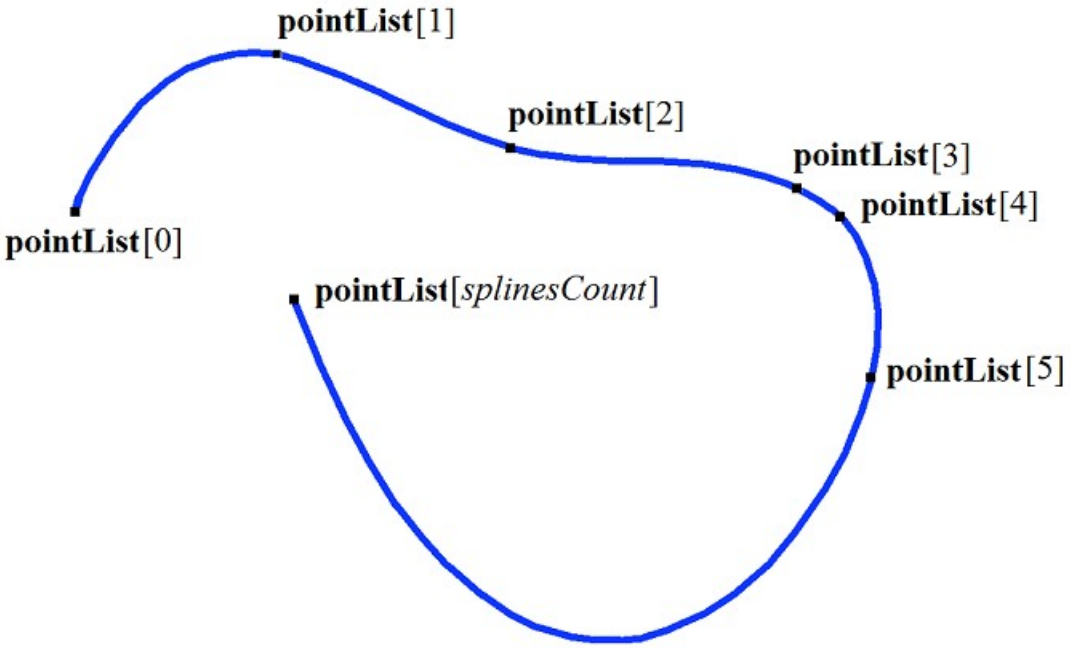


Рис. О.4.9.1.

Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*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,...,*splinesCount*, изменяются пропорционально расстоянию между точками, а вторые производные **vectorList**[*i*], *i*=1,2,...,*splinesCount*–1 вычисляются путём решение 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 *trim*1, конечным параметром усечения базовой кривой *trim*2 и признаком совпадения направлений базовой кривой и усечённой кривой *sense*.

Усечённая кривая совпадает с базовой кривой на участке, определённом параметрами *trim*1 и *trim*2, но может иметь противоположное с ним направление. Если *sense*=1, то *trim*1<*trim*2 и усечённая кривая совпадает по направлению с базовой кривой. Если *sense*=–1, то *trim*2<*trim*1 и усечённая кривая направлена против базовой кривой.

В методе **PointOn**(double & *t*, MbCartPoint3D & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{basisCurve}(trim1+sense\cdot t).$$

Усеченная кривая приведена на рис. О.4.10.1.

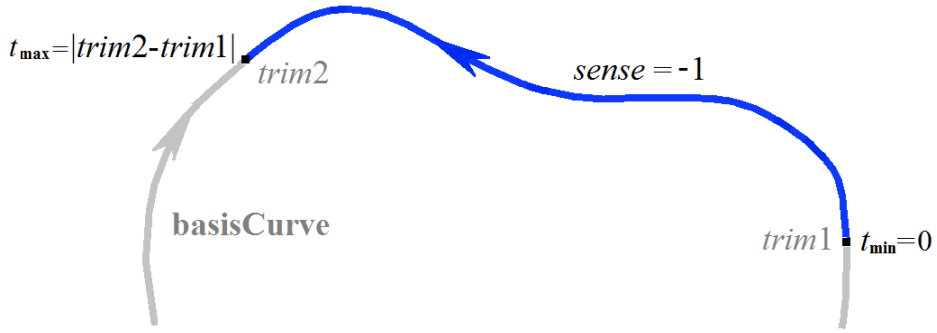


Рис. О.4.10.1.

Параметр кривой *t* принимает значения на отрезке: $0 \le t \le sense(trim2 - trim1)$.

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 *t*1, double *t*2, 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 *tmin* 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 & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{basisCurve}(v(t)),$$

где $v(t) = bmin\dfrac{trim2 - t}{trim2 - trim1} + bmax\dfrac{t - trim1}{trim2 - trim1}$, *bmin*, *bmax* – граничные значения области определения параметра базовой кривой.

Параметр кривой *t* принимает значения на отрезке: $tmin \le t \le tmax$.

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.

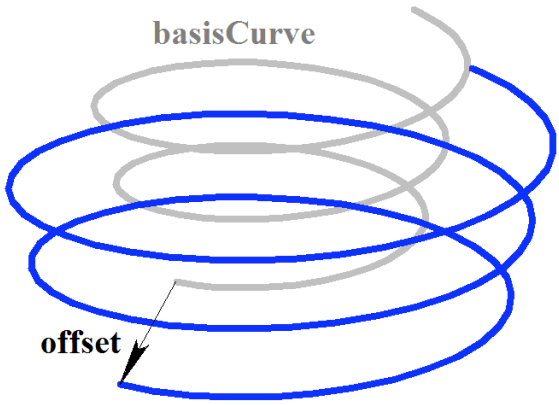


Рис. О.4.12.1.

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.

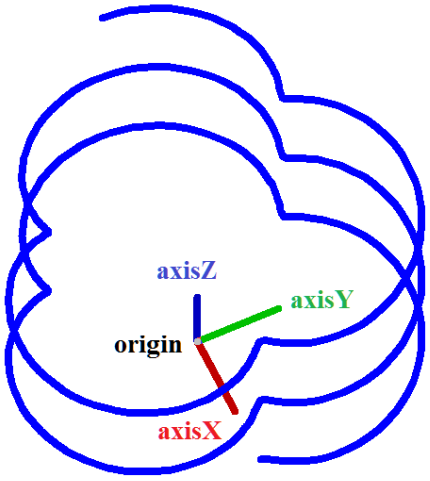
Координатные функции *xFunction*(*t*), *yFunction*(*t*), *zFunction*(*t*) символьной кривой представляют собой скалярные функции общего параметра *t* и заданы в виде символьных выражений. Для каждого символьного выражения выполнен лексический анализ и построено дерево, которое вычисляет

значение символьного выражения для заданного параметра и производные символьного выражения по параметру. Параметр кривой *t* принимает значения на отрезке: *tmin*≤*t*≤*tmax*.

В методе **PointOn**(double & *t*, MbCartPoint3D & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = [xFunction(t) \quad yFunction(t) \quad zFunction(t)].$$

Символьная кривая приведена на рис. О.4.13.1.



xFunction(*t*)=100*cos(*t*)-20*cos(4**t*)

yFunction(*t*)=100*sin(*t*)-20*sin(4**t*)

zFunction(*t*)=*t*10

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*≤*t*≤*tmax*.

В методе **PointOn**(double & *t*, MbCartPoint3D & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = \mathbf{position.origin} + (radius + t\ tgAlpha\ step\_2pi)(\cos(t)\ \mathbf{position.axisX} + \sin(t)\ \mathbf{position.axisY}) + ((t\ step\_2pi)\ \mathbf{position.axisZ}).$$

Коническая спираль приведена на рис. О.4.14.1.

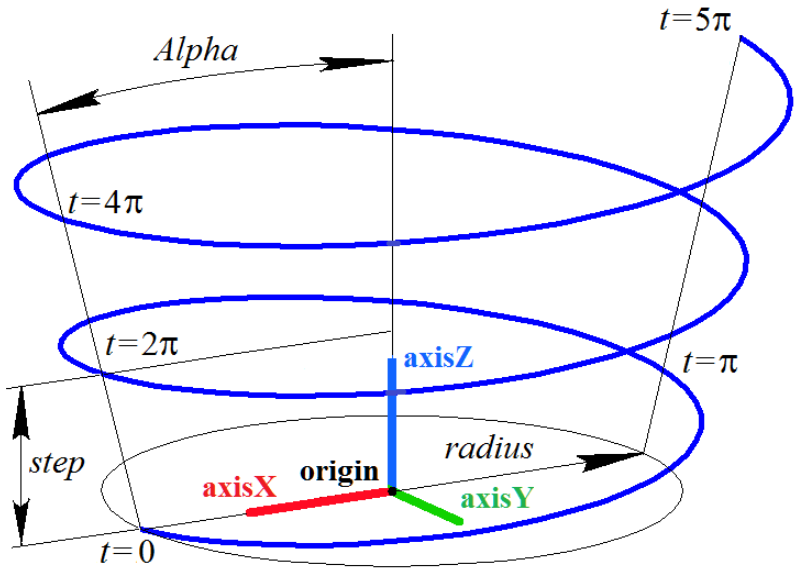


Рис. О.4.14.1.

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 *tgAlpha*=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

$$\mathbf{r}(t) = \mathbf{position.origin} + radius(t)\,(\cos(t)\,\mathbf{position.axisX} + \sin(t)\,\mathbf{position.axisY}) + ((t \cdot step / 2\pi)\,\mathbf{position.axisZ}),$$

где *radius*(*t*) – локальный радиус. Локальный радиус *radius*(*t*) вычисляется следующим образом. По заданному параметру *t* определяем абсциссу *t*·*step*/2π искомой двумерной точки кривой. Далее определяем точку пересечения кривой ***curve*** и вертикальной прямой, пересекающей ось абсцисс в точке *t*·*step*/2π. 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. О.4.15.1.

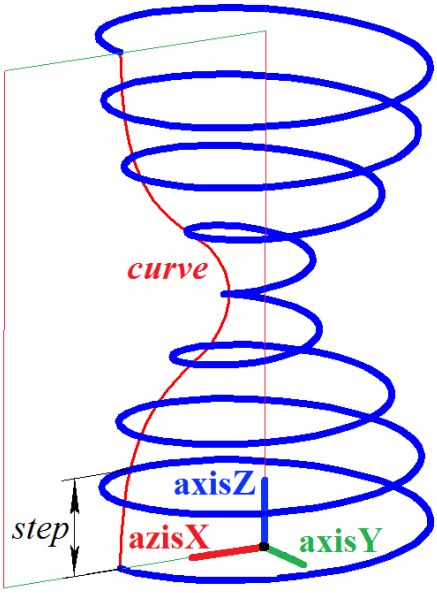


Рис. О.4.15.1.

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.

О.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

$$\mathbf{r}(t) = \mathbf{position.origin} + ((\boldsymbol{point}.y + \mathit{radius}\cos(t)\ \boldsymbol{normal}.y)\ \mathbf{position.axisX}) + (\mathit{radius}\sin(t)\ \mathbf{position.axisY}) + ((\boldsymbol{point}.x + \mathit{radius}\ \mathit{cos(t)}\ \boldsymbol{normal}.x)\ \mathbf{position.axisZ}),$$

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. О.4.16.1.

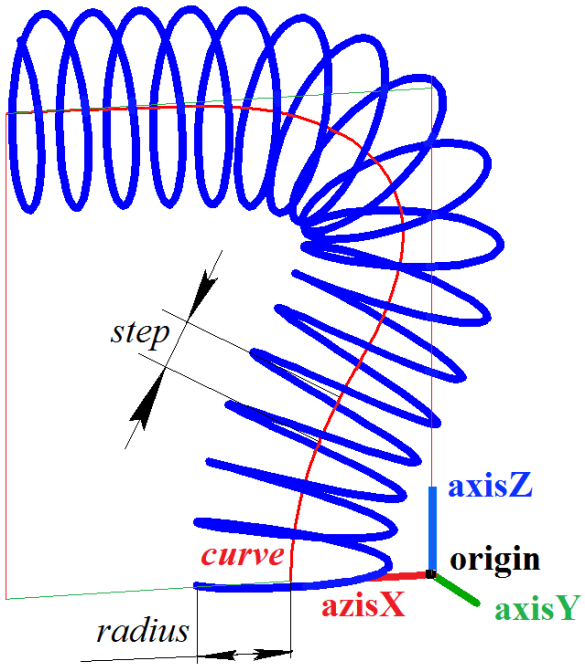


Рис. О.4.16.1.

Минимальный радиус кривизны кривой ***curve*** не должен быть меньше радиуса спирали. Для граничных параметров должны соблюдаться неравенства: *tmin*<*tmax*. Кривая не может быть периодической.

Локальная система координат **position** может быть как правой, так и левой.

О.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 *param*1 и *param*2, признаками *sense*1 и *sense*2 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 *param*1 и *param*2. Направление соединительной кривой в этих точках определяют параметры *sense*1 и *sense*2. 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: *tmin*≤*t*≤*tmax*.

В методе **PointOn**(double & *t*, MbCartPoint3D & **r**) радиус-вектор кривой **r** описывается векторной функцией

$$\mathbf{r}(t) = (1-3w^2+2w^3)\mathbf{point1} + (3w^2+2w^3)\mathbf{point2} + \\ + \left((w-2w^2+w^3)\mathbf{derive1} + (-w^2+w^3)\mathbf{derive2}\right)(tmax-tmin),$$

где $w = \dfrac{t-tmin}{tmax-tmin}$ – относительное значение параметра, **point1** – точка кривой **curve1**, которая вычисляется методом **curve1**–>**PointOn**(*param*1,**point1**), **point2** – точка кривой **curve2**, которая вычисляется методом **curve2**–>**PointOn**(*param*2,**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 *tmax*–*tmin*. Соединительная кривая приведена на рис. О.4.17.1.

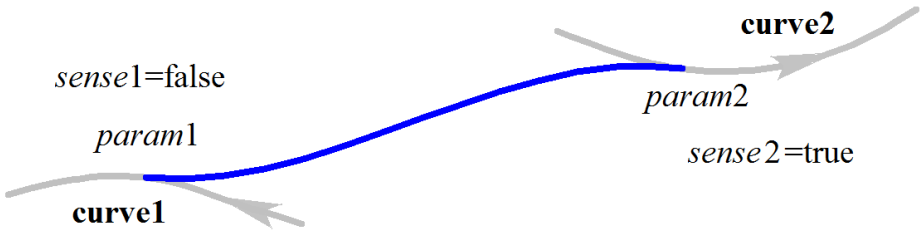


Рис. О.4.17.1.

Форма кривой зависит от расположения крайних точек и направлений соединяемых кривых в этих точках. Для граничных параметров должны соблюдаться неравенства: *tmin<tmax*. 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 друз да 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_{imin} и w_{imax} – 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

$$\mathbf{r}(t) = \mathbf{segments}[k](w_k),$$

где **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 $\sum_{i=0}^{k-1} (w_{i\,max} - w_{i\,min}) \le t < \sum_{i=0}^{k} (w_{i\,max} - w_{i\,min})$, где w_{imin} и w_{imax} – минимальное и максимальное значение параметра *i*–го сегмента. Контур приведён на рис. О.4.18.1.

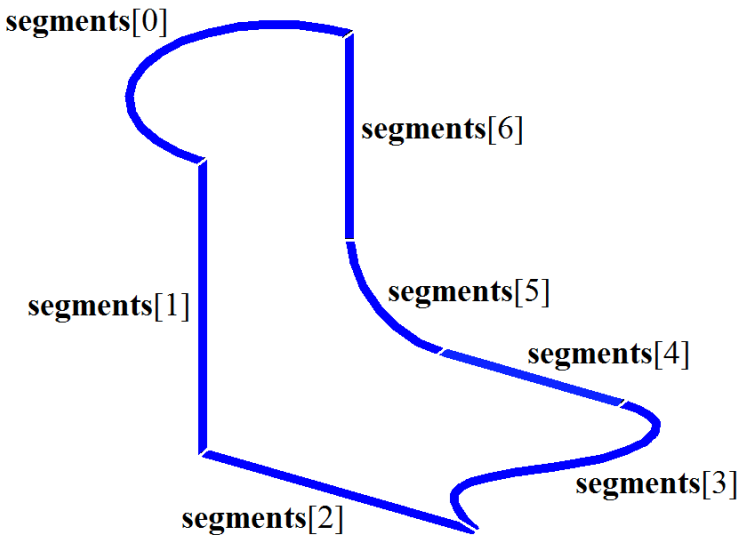


Рис. О.4.18.1.

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

$$\mathbf{r}(t) = \mathbf{position.origin} + (\boldsymbol{point}.x\ \mathbf{position.axisX}) + (\boldsymbol{point}.y\ \mathbf{position.axisY}) ,$$

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.

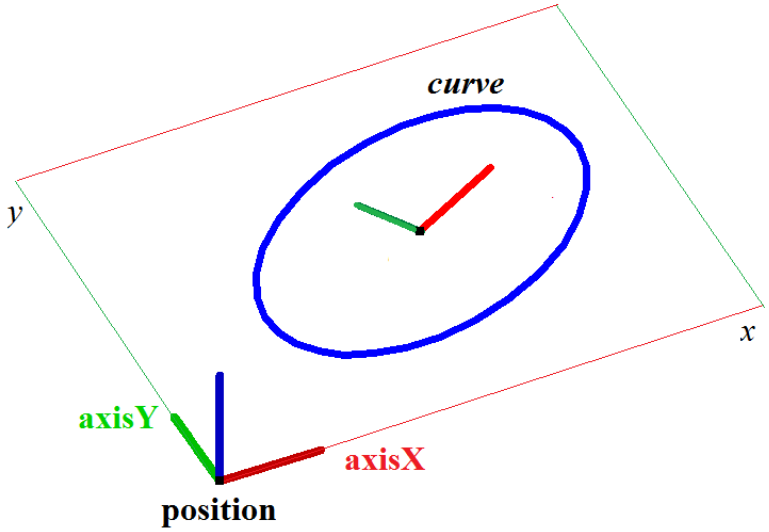


Рис. О.4.19.1.

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

$$\mathbf{r}(t) = \mathbf{surface}(\ \boldsymbol{point}.x, \boldsymbol{point}.y\ ),$$

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.

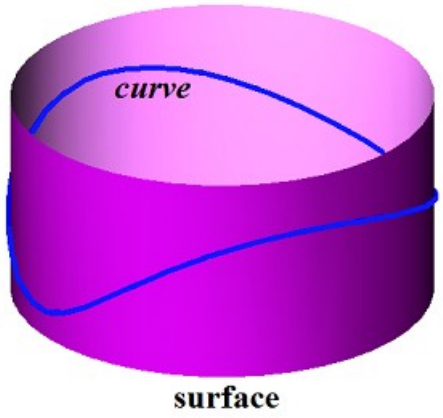


Рис. О.4.20.1.

Двумерная кривая в области определения параметров поверхности приведена на рис. О.4.20.2.

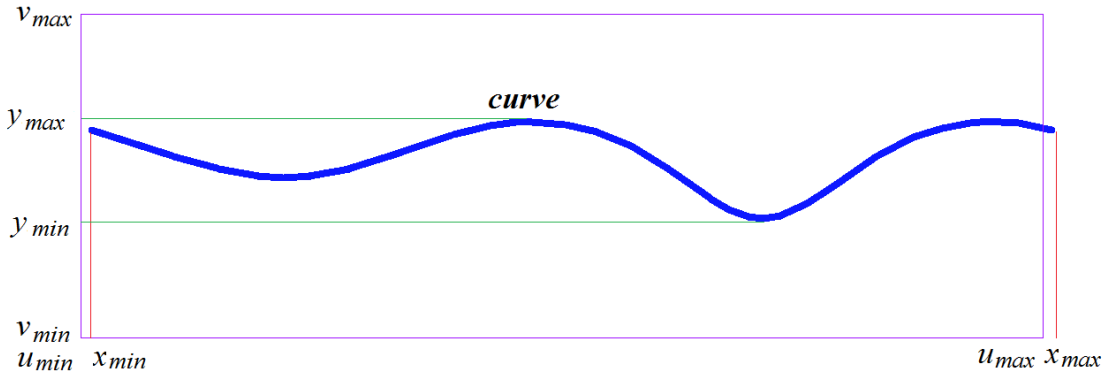


Рис. О.4.20.2.

Производная кривой на поверхности вычисляется как сложная функция

$$\frac{d\mathbf{r}(t)}{dt} = \frac{\partial \mathbf{surface}(u,v)}{\partial u}\boldsymbol{derive}.x + \frac{\partial \mathbf{surface}(u,v)}{\partial v}\boldsymbol{derive}.y\ ,$$

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

$$\mathbf{r}(t) = \mathbf{surface}(\, u, v \,),$$

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.

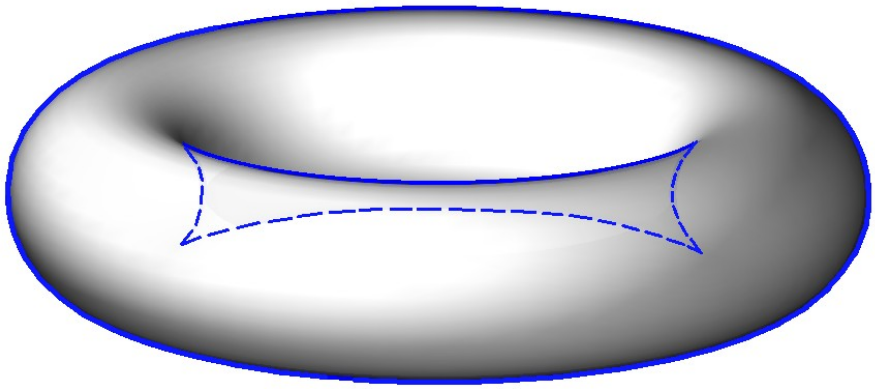


Рис. О.4.21.1.

Силуэтная кривая поверхности тора с другого направления приведена на рис. О.4.21.2.

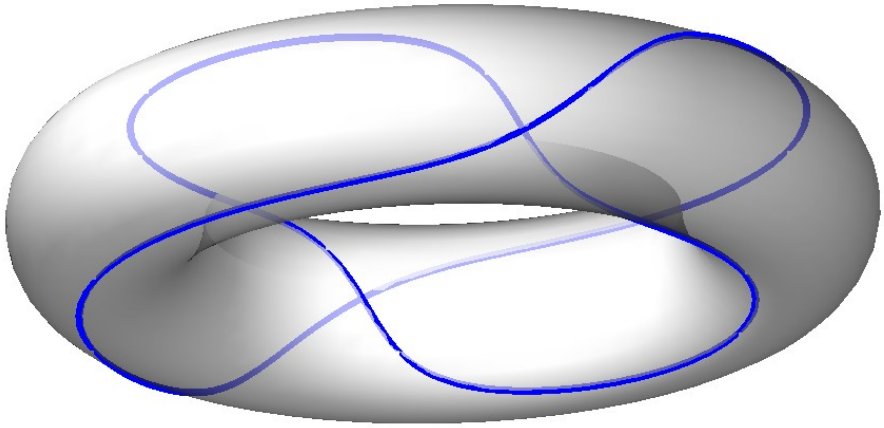


Рис. О.4.21.2.

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

$$\mathbf{r}(t) = \mathbf{surface}(\ \boldsymbol{point}.x, \boldsymbol{point}.y\ ),$$

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

$$\frac{d\mathbf{r}(t)}{dt} = \frac{\partial \mathbf{surface}(u,v)}{\partial u}\boldsymbol{derive}.x + \frac{\partial \mathbf{surface}(u,v)}{\partial v}\boldsymbol{derive}.y\ ,$$

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.

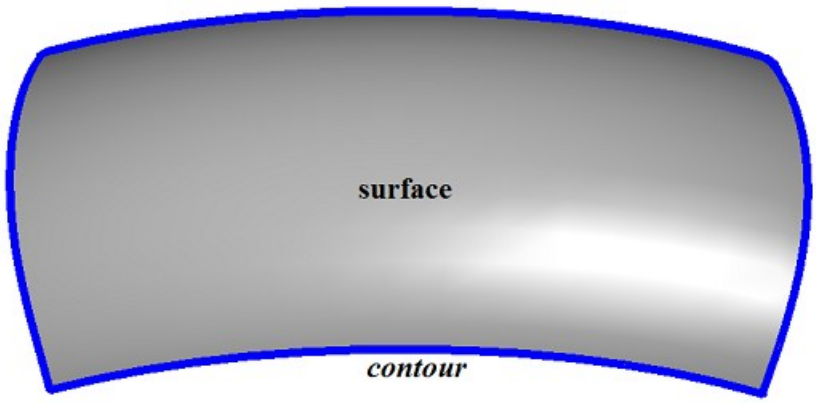


Рис. О.4.22.1.

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

r(*t*) = **position.origin** + (***point***.*x* **position.axisX**) + (***point***.*y* **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.

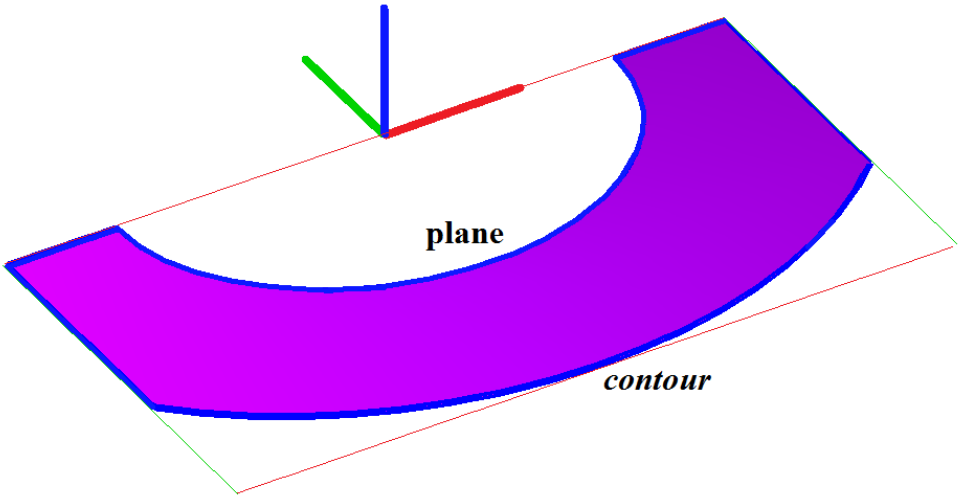


Рис. О.4.23.1.

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.

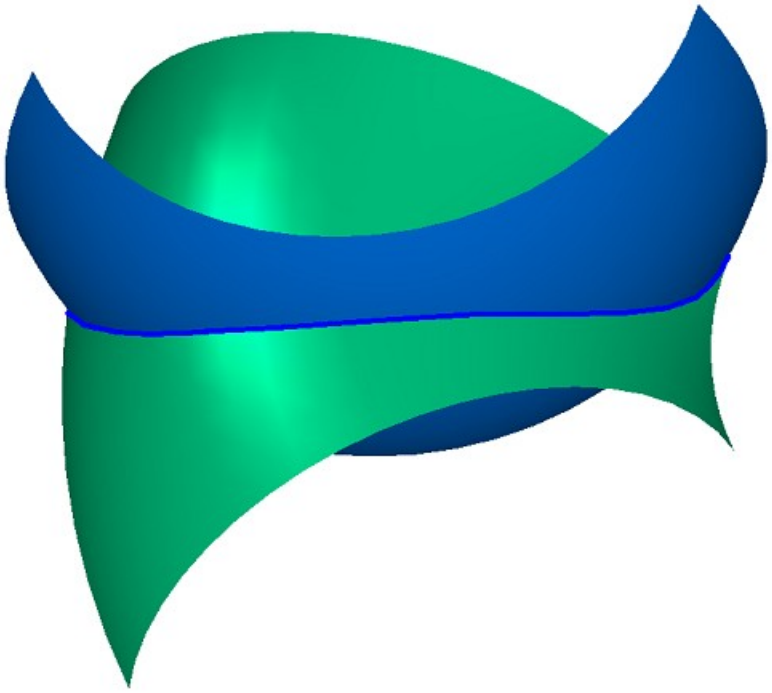


Рис. О.4.24.1.

На рис. О.4.24.2 и О.4.24.3 приведены кривые на поверхностях, из которых строится кривая пересечения.

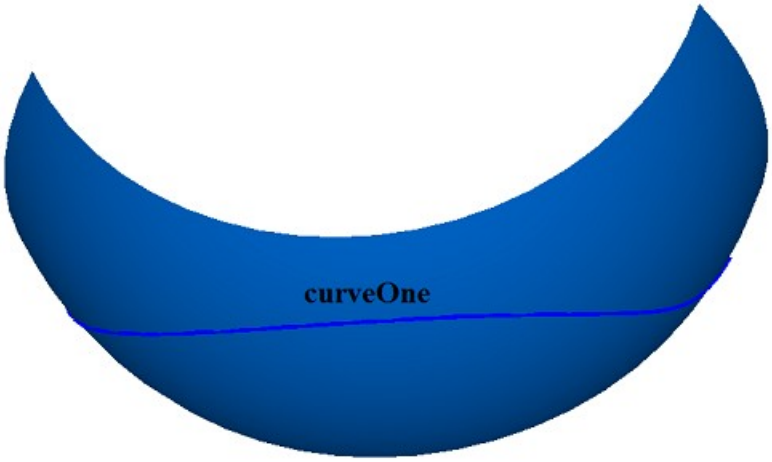


Рис. О.4.24.2.

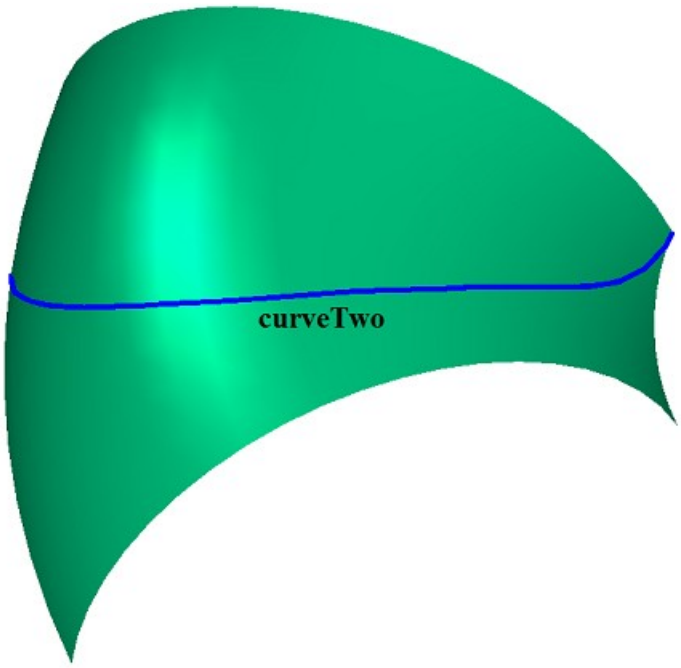


Рис. О.4.24.3.

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.

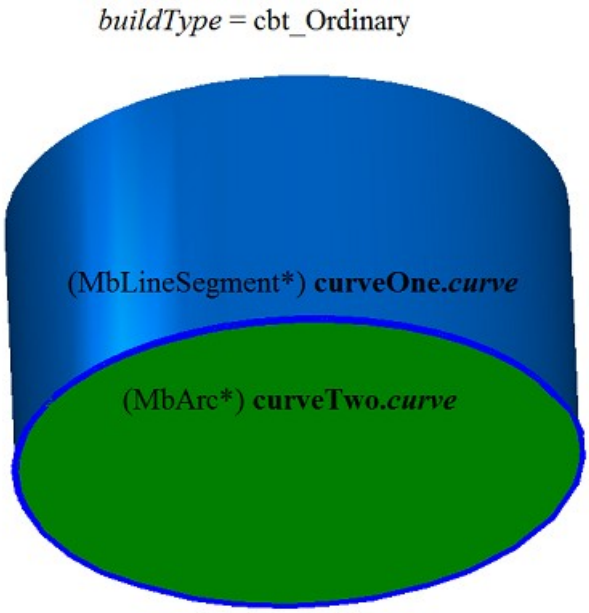


Рис. О.4.24.4.

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**.

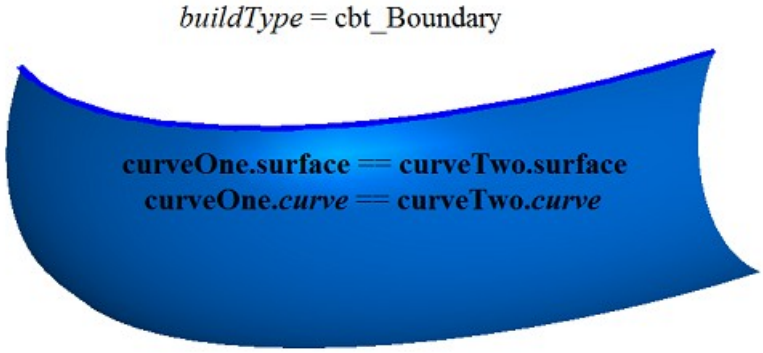


Рис. О.4.24.5.

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*,***point***1), u_1=***point***1.*x*, v_1=***point***1.*y*, u_2, v_2 – координаты двумерной точки, начальное приближение которой вычисляется методом **curveTwo.*curve*–**

>**PointOn**(*t*,***point***2), u_2=***point***2.*x*, v_2=***point***2.*y*. Далее в общем случае (*buildType*=cbt_Specific) параметры u_1, v_1, u_2, v_2 уточняются итерационным методом, использующим уравнения

$$\mathbf{curveOne.surface}(u_1,v_1) = \mathbf{plane}(x,y),$$
$$\mathbf{curveTwo.surface}(u_2,v_2) = \mathbf{plane}(x,y),$$

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.

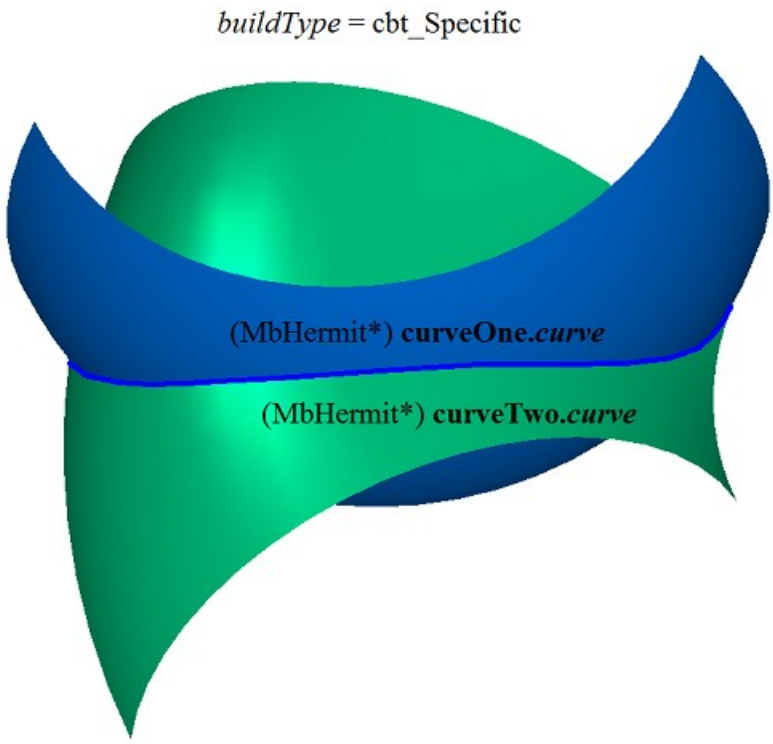


Рис. О.4.24.6.

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.

О.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.

О.5.1. Поверхность MbSurface

Класс MbSurface объявлен в файле surface.h.

Поверхность MbSurface является наследником класса MbSpaceItem, рис. О.5.1.1.

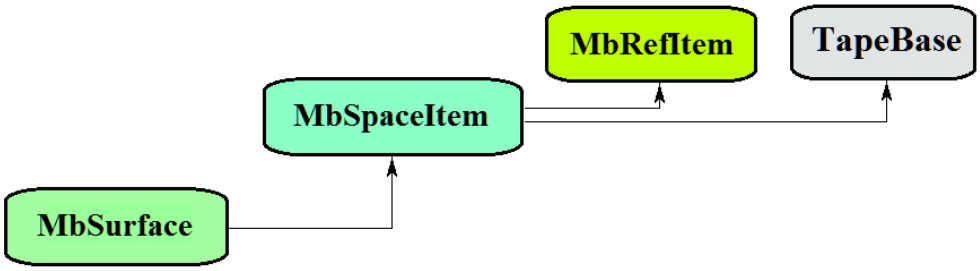


Рис. О.5.1.1.

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

$$\mathbf{suface}(u,v) = \begin{bmatrix} x(u,v) & y(u,v) & z(u,v) \end{bmatrix}$$

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 $u_{min} \le u \le u_{max}$, $v_{min} \le v \le v_{max}$. В общем случае область Ω описывается двумерными кривыми. Координаты $x(u,v)$, $y(u,v)$, $z(u,v)$ точки поверхности **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 **surface**$(u \pm k p_u, v)$=**surface**(u,v), где k – целое число. Поверхность будем называть периодической по второму параметру, если существует $p_v>0$, такое, что **surface**$(u, v \pm k p_v)$=**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

$$\mathbf{s}_u = \frac{\partial \mathbf{surface}(u,v)}{\partial u}; \ \mathbf{s}_v = \frac{\partial \mathbf{surface}(u,v)}{\partial v};$$

$$\mathbf{s}_{uu} = \frac{\partial^2 \mathbf{surface}(u,v)}{\partial u^2}; \ \mathbf{s}_{vv} = \frac{\partial^2 \mathbf{surface}(u,v)}{\partial v^2}; \ \mathbf{s}_{uv} = \frac{\partial^2 \mathbf{surface}(u,v)}{\partial u \partial v} = \frac{\partial^2 \mathbf{surface}(u,v)}{\partial v \partial u};$$

$$\mathbf{s}_{uuu} = \frac{\partial^3 \mathbf{surface}(u,v)}{\partial u^3}; \ \mathbf{s}_{uuv} = \frac{\partial^3 \mathbf{surface}(u,v)}{\partial u \partial u \partial v}; \ \mathbf{s}_{uvv} = \frac{\partial^3 \mathbf{surface}(u,v)}{\partial u \partial v \partial v}; \ \mathbf{s}_{vvv} = \frac{\partial^3 \mathbf{surface}(u,v)}{\partial v^3};$$

для частных производных поверхности по её параметрам.

Основным методом поверхности является метод

void **PointOn**(double & u, double & v, MbCartPoint3D & **s**).

Он выдаёт радиус-вектор **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 & $\mathbf{s}_u$),
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* 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,
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,
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 прямоугольную область определения параметров.

О.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. О.5.2.1.

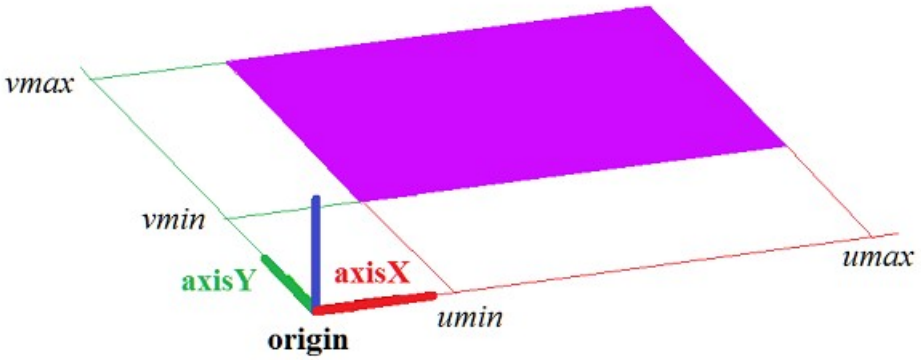


Рис. О.5.2.1.

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**.

О.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: $umin \le u \le umax$. 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: $vmin \le v \le vmax$. 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

$$\mathbf{s}(u,v) = \mathbf{position.origin} + radius\ (\cos(u)\ \mathbf{position.axisX} + \sin(u)\ \mathbf{position.axisY}) + height\ v\ \mathbf{position.axisZ}.$$

Цилиндрическая поверхность приведена на рис. О.5.3.1.

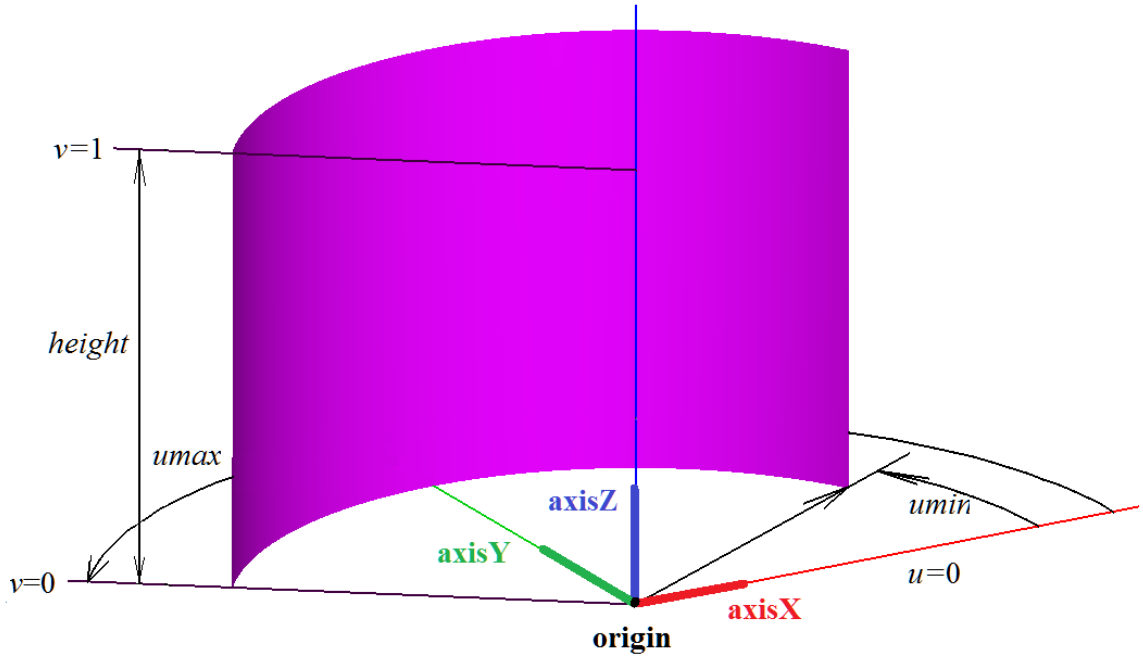


Рис. О.5.3.1.

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).

О.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*≤*u*≤*umax*. Значения *u*=0 и *u*=2π соответствуют точке на плоскости XZ. Поверхность может быть периодической по первому параметру. У периодической поверхности *umax*-*umin*=2π, у не периодической поверхности *umax*-*umin*<2π.

Второй параметр поверхности отсчитывается по прямой вдоль вектора **position.axisZ**. Второй параметр поверхности *v* принимает значения на отрезке: *vmin*≤*v*≤*vmax*. Значенит *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

$$\mathbf{s}(u,v) = \mathbf{position.origin} + (radius + height\ v\ tg(angle))\ (\cos(u)\ \mathbf{position.axisX} + \sin(u)\ \mathbf{position.axisY}) + height\ v\ \mathbf{position.axisZ}.$$

Коническая поверхность приведена на рис. О.5.4.1.

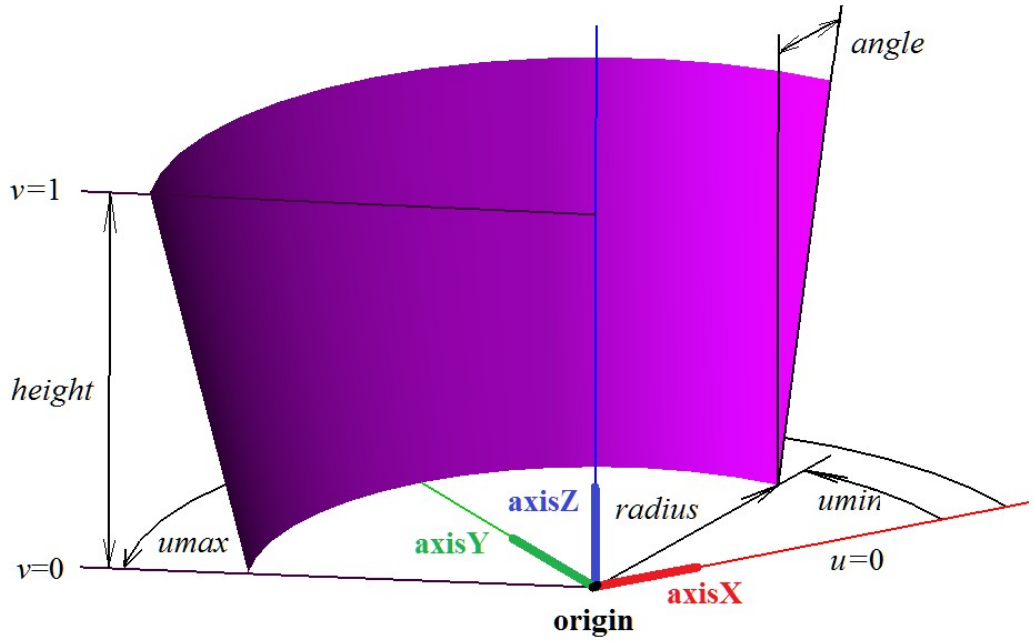


Рис. О.5.4.1.

Радиус и высота должны быть больше нуля, а угол по модулю не должен превышать π/2: *radius*>0, *height*>0, -π/2<*angle*<π/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* *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*≤*u*≤*umax*. Значения *u*=0 и *u*=2π соответствуют точке на плоскости XZ. Поверхность может быть периодической по первому параметру. У периодической поверхности *umax*-*umin*=2π, у не периодической поверхности *umax*-*umin*<2π.

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*≤*v*≤*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

$$\mathbf{s}(u,v) = \mathbf{position.origin} + radius\,(\cos(u)\,\mathbf{position.axisX} + \sin(u)\,\mathbf{position.axisY}) + radius\,\sin(v)\,\mathbf{position.axisZ}.$$

Сфера приведена на рис. О.5.5.1.

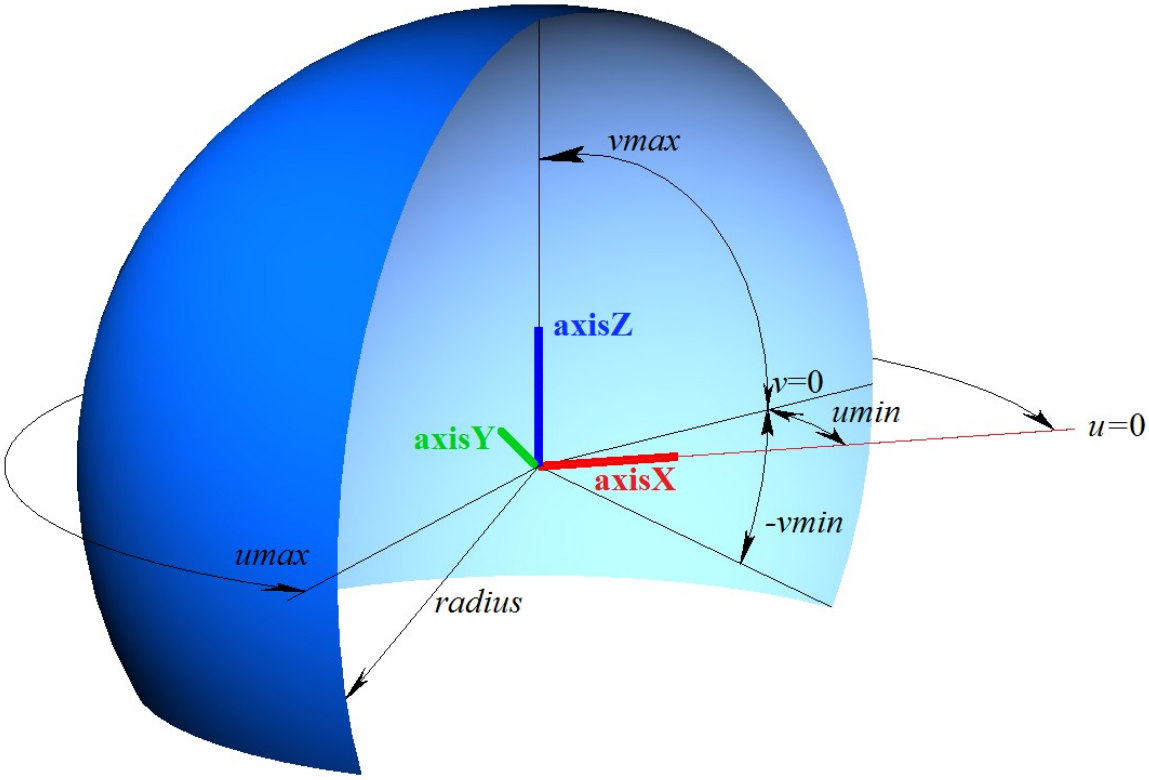


Рис. О.5.5.1.

Радиус сферы должен быть больше нуля: *radius*>0. Сфера имеет полюсы для параметра *v*=π/2 и *v*=–π/2. Для граничных параметров поверхности должны соблюдаться неравенства: *umin*<*umax*, *vmin*<*vmax*, *vmax*<=π/2, *vmin*>=–π/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.

О.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*≤*u*≤*umax*. Значения *u*=0 и *u*=2π соответствуют точке на на плоскости XZ. Поверхность может быть периодической по первому параметру. У периодической поверхности *umax*-*umin*=2π, у не периодической поверхности *umax*-*umin*<2π.

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*≤*v*≤*vmax*. Значения *v*=0 и *v*=2π соответствуют точке на плоскости XY локальной системы координат поверхности. Поверхность может быть периодической по второму параметру при *majorRadius*>*minorRadius*. У периодической поверхности *vmax*-*vmin*=2π, у не периодической поверхности *vmax*-*vmin*<2π.

В методе **PointOn**(double *u*, double *v*, MbCartPoint3D & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{position.origin} + (majorRadius + (minorRadius\ \cos(v))\ (\cos(u)\ \mathbf{position.axisX} + \sin(u)\ \mathbf{position.axisY}) + minorRadius\ \sin(v)\ \mathbf{position.axisZ}.$$

Поверхность тора приведена на рис. О.5.6.1.

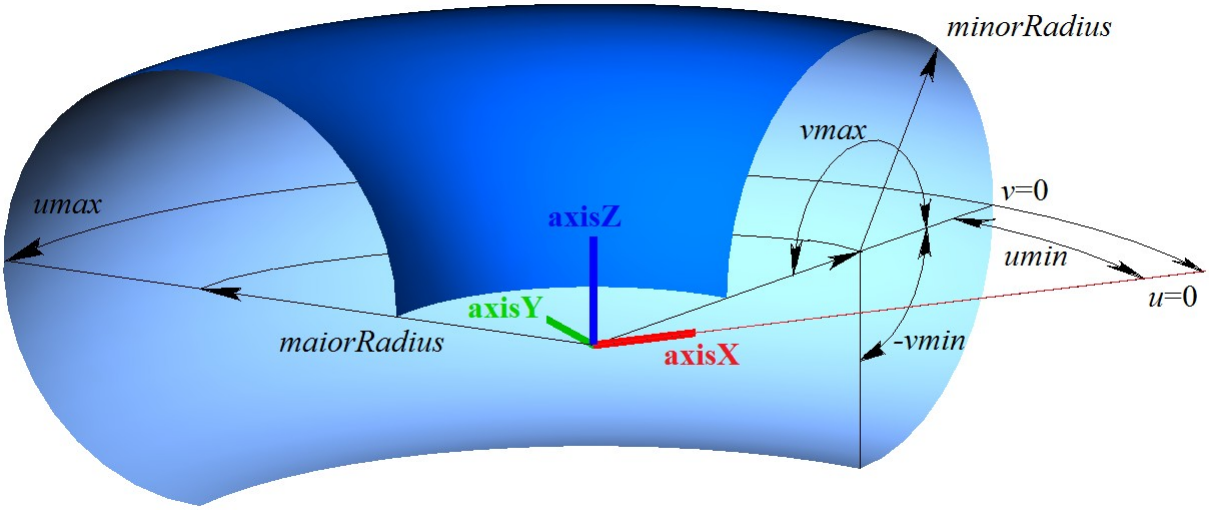


Рис. О.5.6.1.

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*=2π–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*≤*u*≤*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*≤*v*≤*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

$$\mathbf{s}(u,v) = \mathbf{curve}(u) + (\mathbf{direction}\ distance\ v).$$

Поверхность выдавливания приведена на рис. О.5.7.1.

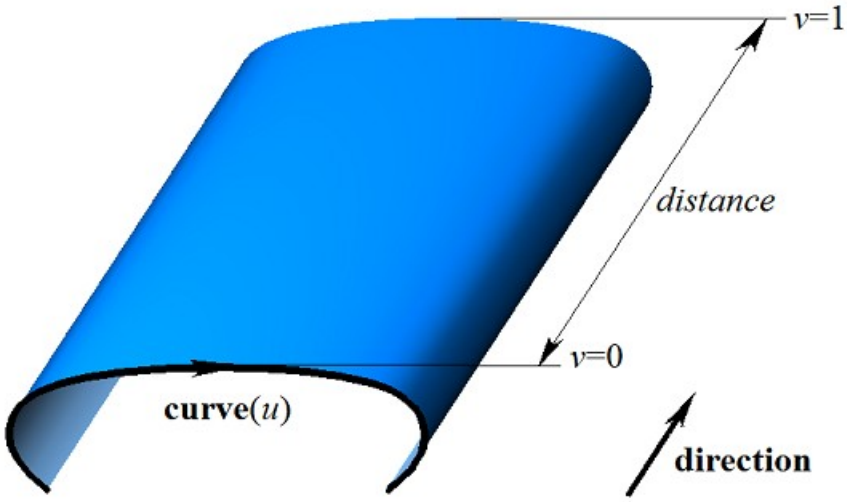


Рис. О.5.7.1.

Для граничных значений второго параметра поверхности должно соблюдаться неравенство: *vmin*<*vmax*.

О.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.axizZ** 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 *umin*≤*u*≤*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*≤*v*≤*vmax*. Значения *v*=0 и *v*=2π соответствуют точке на образующей кривой. Поверхность может быть периодической по второму параметру. У периодической поверхности *vmax*-*vmin*=2π, у не периодической поверхности *vmax*-*vmin*<2π.

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 **(curve**(*u*)–**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. О.5.8.1.

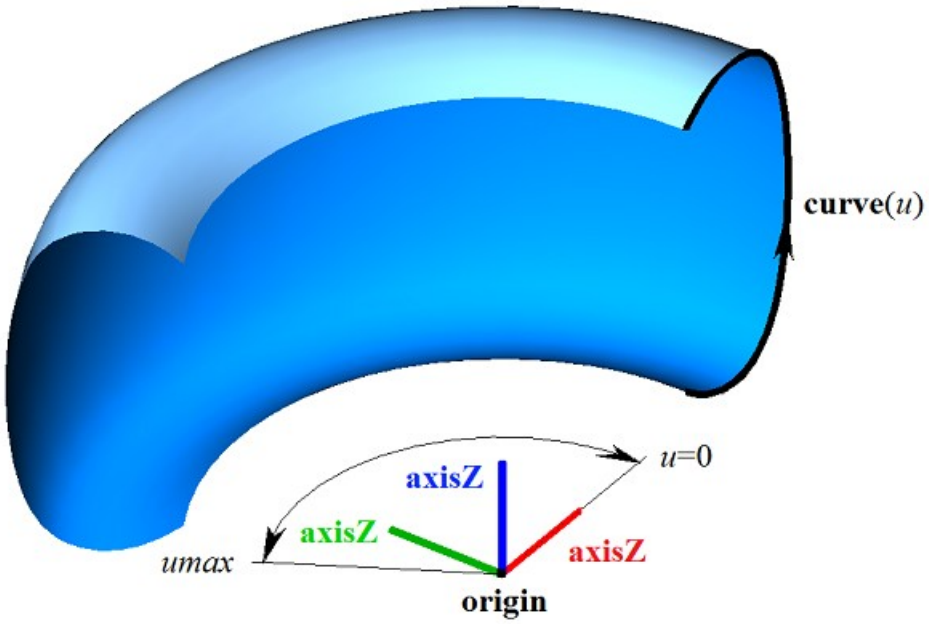


Рис. О.5.8.1.

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.axizZ** от вектора **position.axisX** в направлении вектора **position.axisY**. Локальная система координат **position** может быть как правой, так и левой.

О.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*≤*u*≤*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*≤*v*≤*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

$$\mathbf{s}(u,v) = \mathbf{spine}(v) + \mathbf{curve}(u) - \mathbf{origin}.$$

Поверхность перемещения в общем случае приведена на рис. О.5.9.1.

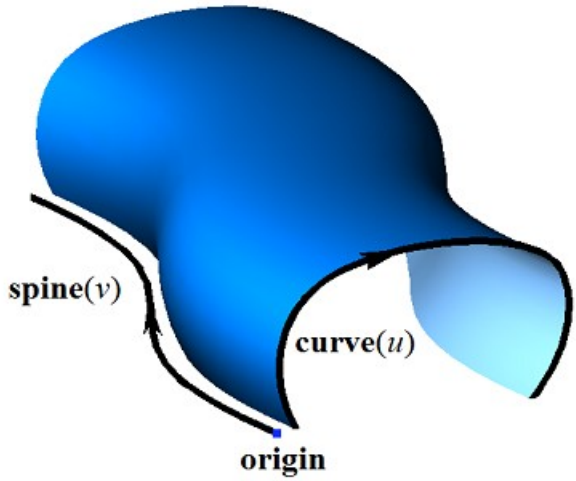


Рис. О.5.9.1.

В методе **PointOn**(double *u*, double *v*, MbCartPoint3D & **s**) в частном случае радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{spine}(v) + (\mathbf{curve}(u) - \mathbf{origin})\,(1-w) + (\mathbf{brink}(t) - \mathbf{ending})\,w,$$

где $w = \dfrac{v - vmin}{vmax - vmin}$, $t = tmin + (u - umin)dt$. Поверхность перемещения в частном случае приведена на рис. О.5.9.2.

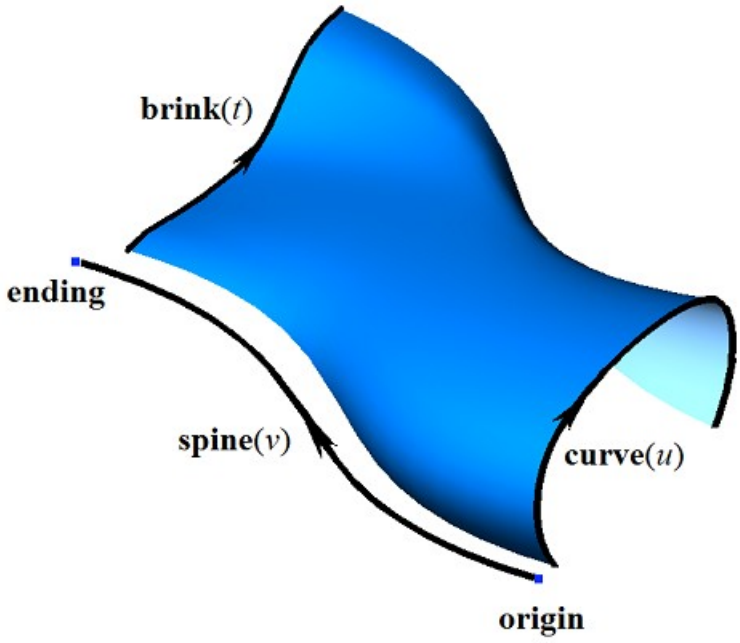


Рис. О.5.9.2.

Для отсутствия самопересечений поверхности образующая и направляющие кривые не должны иметь параллельных друг другу участков. В определенных случаях поверхность перемещения может иметь особые точки.

О.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.axizZ** 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 $umin \le u \le umax$, 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: $vmin \le v \le vmax$. 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

$$\mathbf{s}(u,v) = \mathbf{position.origin} + radius\,(\cos(t)\,\mathbf{position.axisX} + \sin(t)\,\mathbf{position.axisY}) + ((t\ step/2\pi)\,\mathbf{position.axisZ}) + (\mathbf{curve}(u) - \mathbf{origin})\,\mathbf{M}(v),$$

где $\mathbf{M}(v)$ – матрица вращения. Заметим, что умножение вектора (**curve**(*u*)–**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 $\mathbf{M}(v)$ переводит вектор **curve(*u*)–origin** в локальную систему координат, поворачивает его в ней на угол *v* вокруг оси вращения и возвращает повёрнутый вектор обратно в глобальную систему координат. Спиральная поверхность приведена на рис. О.5.10.1.

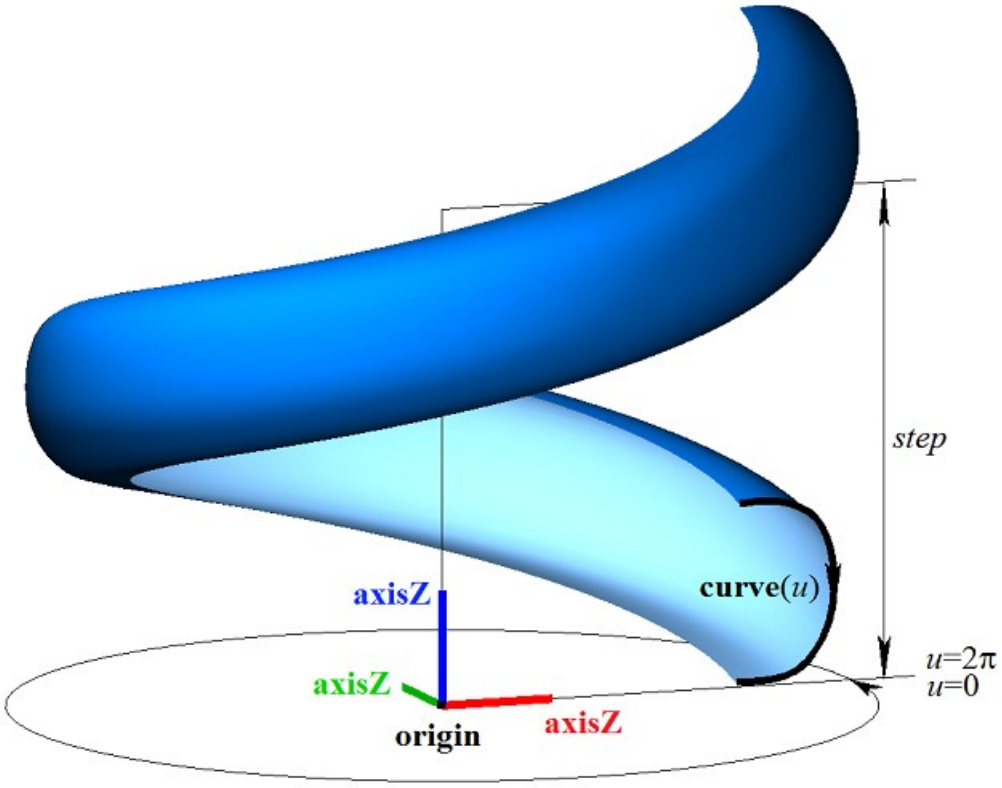


Рис. О.5.10.1.

Для граничных значений второго параметра поверхности должно соблюдаться неравенство: *vmin*<*vmax*.

О.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 $umin \le u \le umax$, 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 $vmin \le v \le vmax$, 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

$$\mathbf{s}(u,v) = \mathbf{spine}(v) + (\mathbf{curve}(u) - \mathbf{origin})\ \mathbf{M}(v),$$

где $\mathbf{M}(v)$ – матрица, связанная с направляющей кривой. Заметим, что умножение вектора **(curve**(*u*)–**origin**) на матрицу $\mathbf{M}(v)$ выполняется справа. Матрица $\mathbf{M}(v)$ имеет вид

$$\mathbf{M}(v) = \mathbf{A}^{-1}(vmin) \cdot \mathbf{A}(v),$$

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.

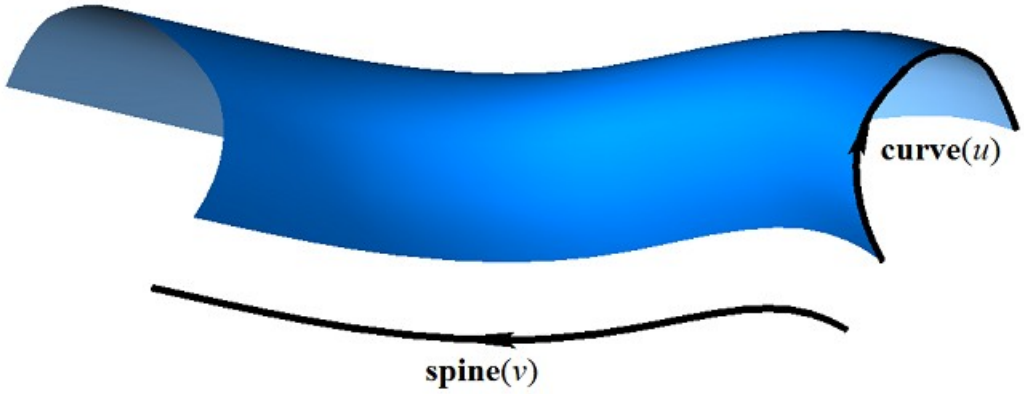


Рис. О.5.11.1.

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.

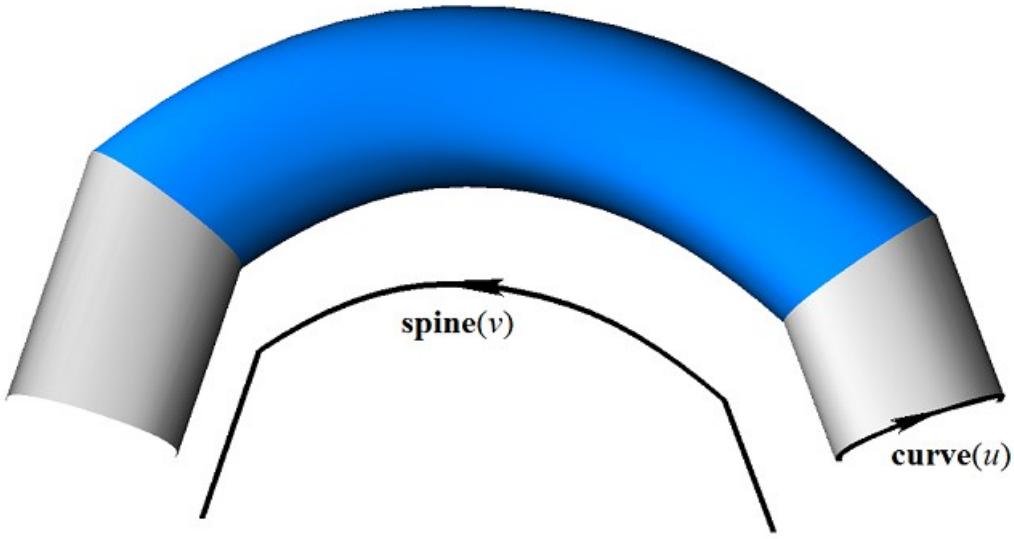


Рис. О.5.12.1.

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*≤*u*≤*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*≤*v*≤*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 $w = \dfrac{v - vmin}{vmax - vmin}$. Секториальная поверхность приведена на рис. О.5.13.1.

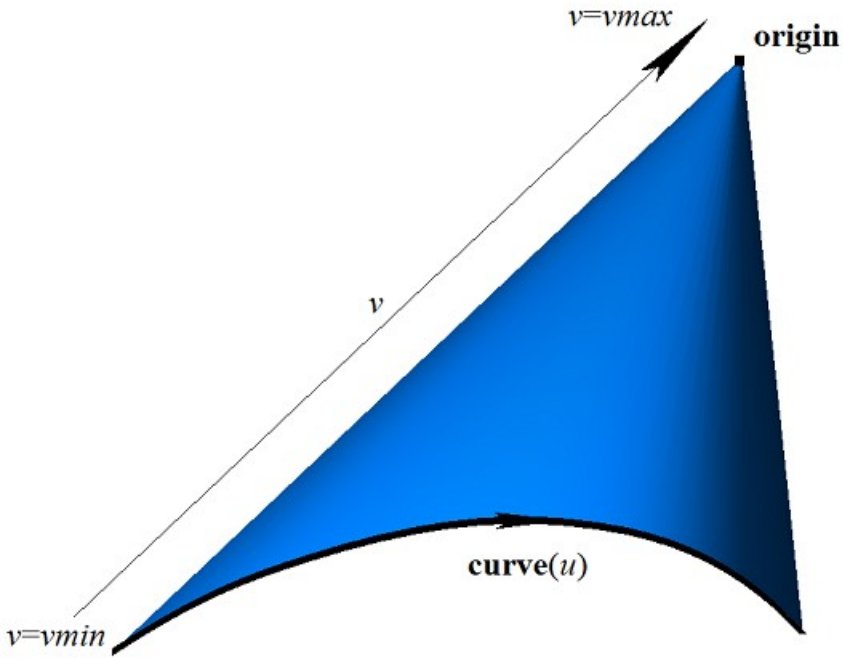


Рис. О.5.13.1.

Кривые поверхности **s**(*const*,*v*) являются отрезками прямой. Поверхность имеет полюс в точке **origin** при *v*=*vmax*. Секториальная поверхность является частным случаем линейчатой поверхности.

О.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*≤*u*≤*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*≤*v*≤*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

$$\mathbf{s}(u,v) = \mathbf{curve}(u)\,(1-w) + \mathbf{sline}(t)\,w,$$

где $w = \frac{v - vmin}{vmax - vmin}$, $t = tmin + (u - umin)dt$. Линейчатая поверхность приведена на рис. О.5.14.1.

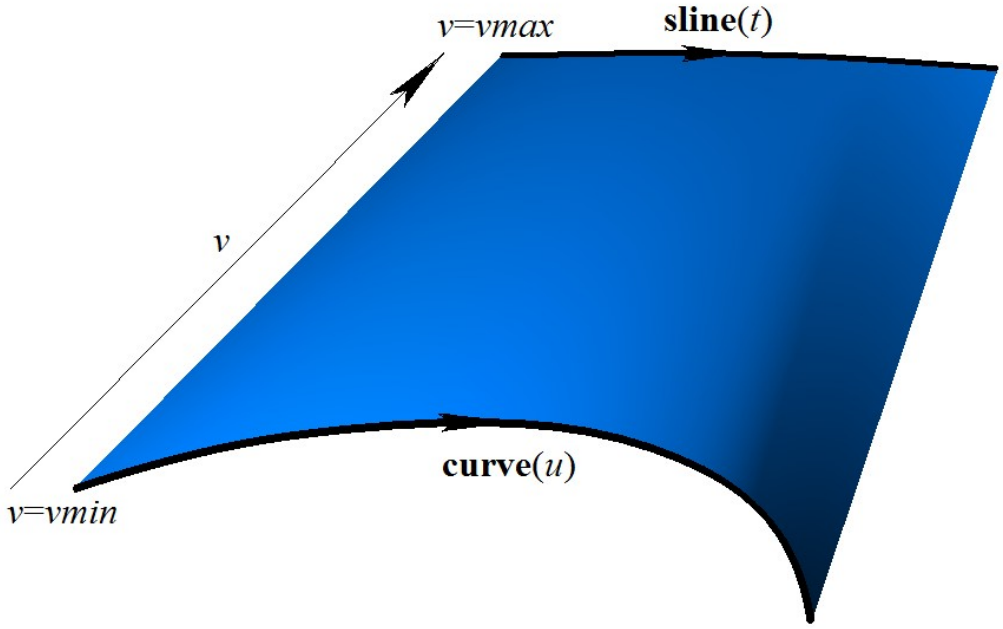


Рис. О.5.14.1.

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 RPArray<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 **derive**1, вектором направления непериодической поверхности при *vmax* MbVector3D **derive**2, 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 MbReperamCurve3D. Первый параметр поверхности принимает значения на отрезке *umin*≤*u*≤*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: $vmin \le v \le vmax$. 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 $w = \dfrac{v - vParams[i]}{vParams[i+1] - vParams[i]}$, **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 $vParams[i] \le v \le vParams[i+1]$. Если среди соседних элементов множества *vLabels* есть одинаковые значения, то между соответствующими кривыми поверхность эквивалентна поверхности выдавливания. Поверхность на семействе кривых приведена на рис. О.5.15.1.

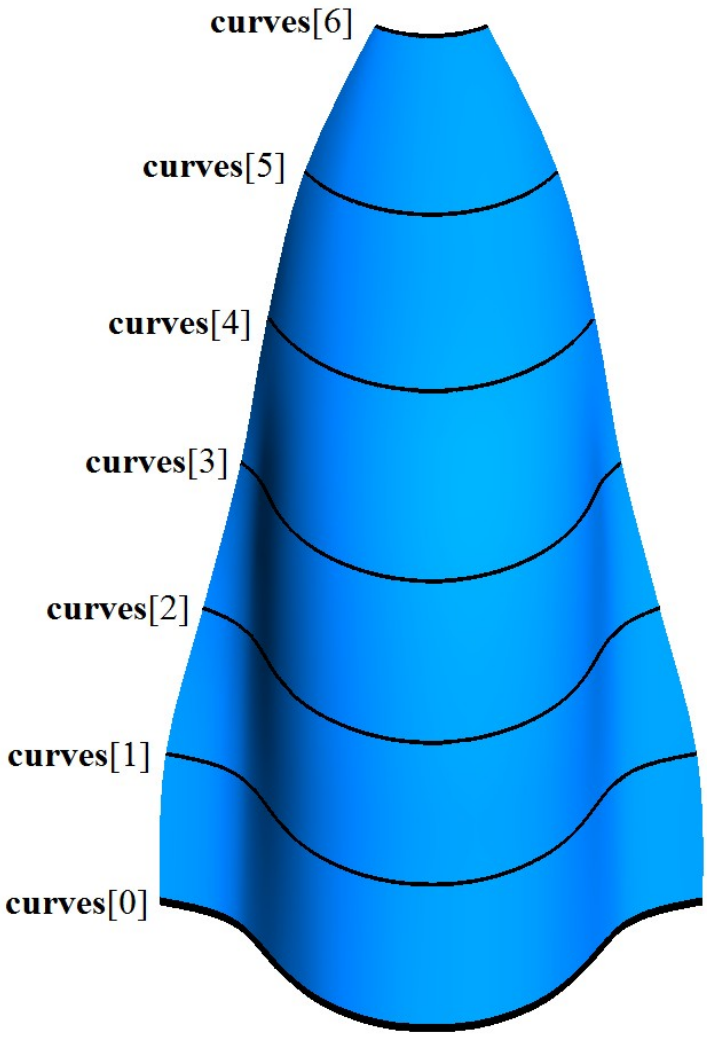


Рис. О.5.15.1.

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 180. 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.

О.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 RPArray<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 MbReperamCurve3D. Первый параметр поверхности принимает значения на отрезке $umin \le u \le umax$, 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 $vmin \le v \le vmax$, 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. О.5.16.1.

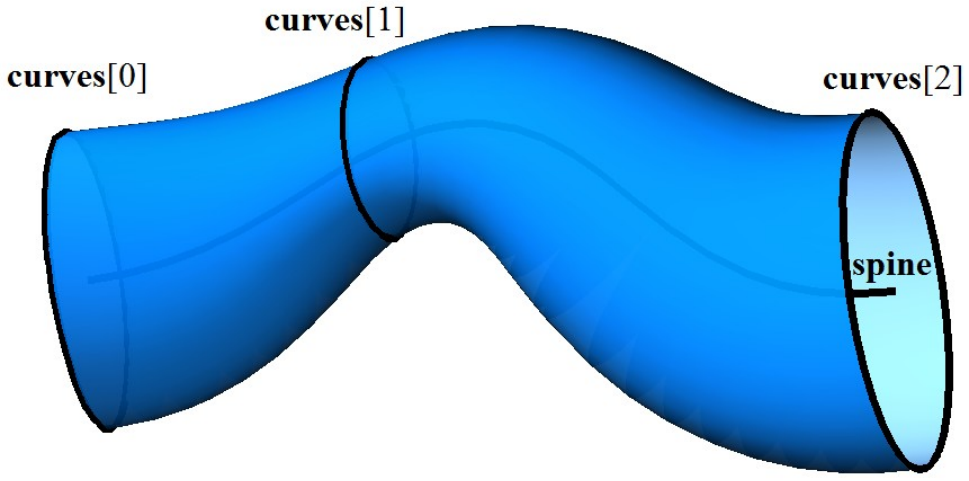


Рис. О.5.16.1.

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.

О.5.17. Поверхность на трёх кривых MbCornerSurface

Класс MbCornerSurface объявлен в файле surf_corner_surface.h.

Поверхность на трёх кривых MbCornerSurface описывается кривыми MbCurve3D* **curve**0, **curve**1, **curve**2, тремя точками MbCartPoint3D **vertex**[3] и тремя парами граничных значений параметров соответствующих кривых: *t*0*min*, *t*0*max*, *t*1*min*, *t*1*max*, *t*2*min*, *t*2*max*. У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Первый параметр поверхности *u* принимает значения на отрезке $0 \le u \le 1$. 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 $0 \le v \le 1$. Поверхность не может быть периодической по второму параметру.

Кривые **curve**0, **curve**1, **curve**2 должна иметь точки пересечения или точки скрещения друг с другом. Параметры кривых *t*0*min*, *t*0*max*, *t*1*min*, *t*1*max*, *t*2*min*, *t*2*max* 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

$$\begin{aligned} \mathbf{s}(u,v) \;=\; & w0\,(\mathbf{curve}2(t2) + \mathbf{curve}1(s1) - \mathbf{vertex}[0]) + \\ & + w1\,(\mathbf{curve}0(t0) + \mathbf{curve}2(s2) - \mathbf{vertex}[1]) + \\ & + w2\,(\mathbf{curve}1(t1) + \mathbf{curve}0(s0) - \mathbf{vertex}[2]), \end{aligned}$$

где $w0=1-u$, $w1=0.5(u-uv)$, $w2=0.5(u+uv)$ – барицентрические координаты поверхности, $t0=w2\cdot t0min+(1-w2)\cdot t0max$, $s0=(1-w1)\cdot t0min+w1\cdot t0max$ – параметры кривой **curve**0, $t1=w0\cdot t1min+(1-w0)\cdot t1max$, $s1=(1-w2)\cdot t1min+w2\cdot t1max$ – параметры кривой **curve**1, $t2=w1\cdot t2min+(1-w1)\cdot t2max$, $s2=(1-w0)\cdot t2min+w0\cdot t2max$ – параметры кривой **curve**2. Поверхность на трех скрещивающихся кривых приведена на рис. О.5.17.1.

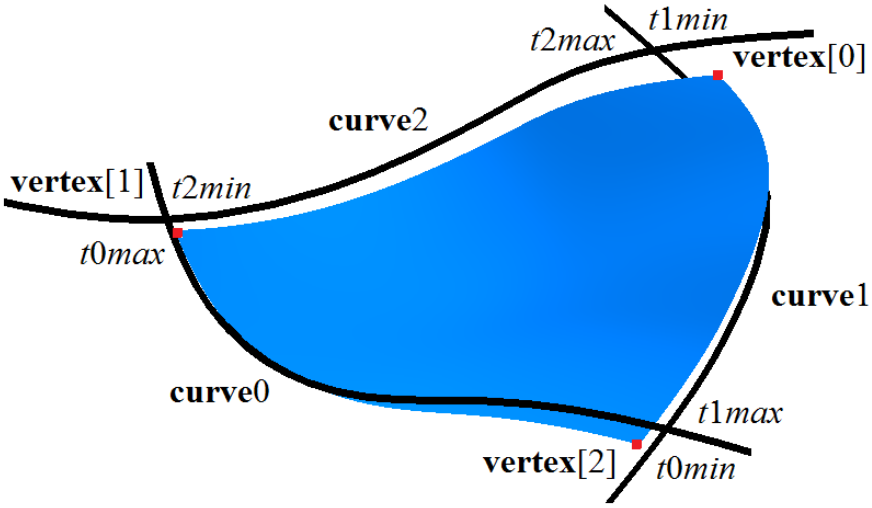


Рис. О.5.17.1.

На рис. О.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.

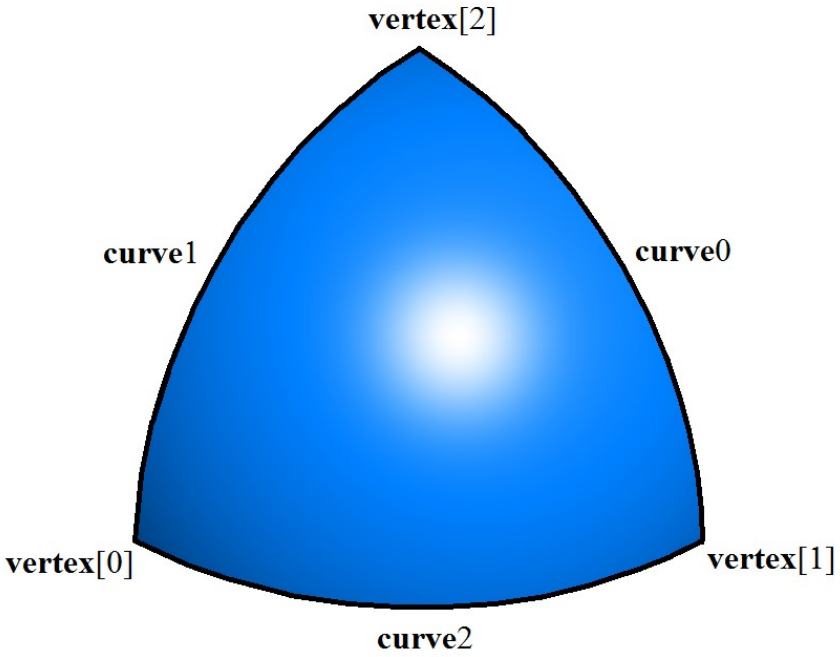


Рис. О.5.17.2.

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 $w0=0$ поверхность проходит по участку кривой **curve**0, при $w1=0$ поверхность проходит по участку кривой **curve**1, при $w2=0$ поверхность проходит по участку кривой **curve**2.

О.5.18. Поверхность Кунса MbCoverSurface

Класс MbCoverSurface объявлен в файле surf_cover_surface.h.

Поверхность Кунса MbCoverSurface описывается кривыми MbCurve3D* **curve**0, **curve**1, **curve**2, **curve**3, четырьмя точками MbCartPoint3D **vertex**[4], четырьмя парами граничных значений параметров соответствующих кривых: *t*0*min*, *t*0*max*, *t*1*min*, *t*1*max, t*2*min, t*2*max*, *t*3*min*, *t*3*max*, 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 $0 \le u \le 1$. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve**0 и **curve**2.

Второй параметр поверхности v принимает значения на отрезке $0 \le v \le 1$. Поверхность может быть периодической по второму параметру, если периодическими являются кривые **curve**1 и **curve**3.

Соседние кривые **curve**0, **curve**1, **curve**2, **curve**3 должна иметь точки пересечения или точки скрещения друг с другом. Параметры кривых *t*0*min*, *t*0*max*, *t*1*min*, *t*1*max*, *t*2*min*, *t*2*max*, *t*3*min*, *t*3*max* 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

$$\begin{aligned} \mathbf{s}(u,v) = {} & (1-v)\,(\mathbf{curve}0(t0) - (1-u)\,\mathbf{vertex}[0]) + \\ & + u \quad (\mathbf{curve}1(t1) - (1-v)\,\mathbf{vertex}[1]) + \\ & + v \quad (\mathbf{curve}2(t2) - u \quad \mathbf{vertex}[2]) + \\ & + (1-u)\,(\mathbf{curve}3(t3) - v \quad \mathbf{vertex}[3]), \end{aligned}$$

где $w0=1-u$, $w1=0.5(u-uv)$, $w2=0.5(u+uv)$ – барицентрические координаты поверхности, $t0=(1-u)\cdot t0min+u\cdot t0max$ – параметр кривой **curve**0, $t1=(1-v)\cdot t1min+v t1max$ – параметр кривой **curve**1, $t2=(1-u)\cdot t2min+u\cdot t2max$ – параметр кривой **curve**2, $t3=(1-v)\cdot t3min+v t3max$ – параметр кривой **curve**3. Поверхность Кунса на четырёх скрещивающихся кривых приведена на рис. О.5.18.1.

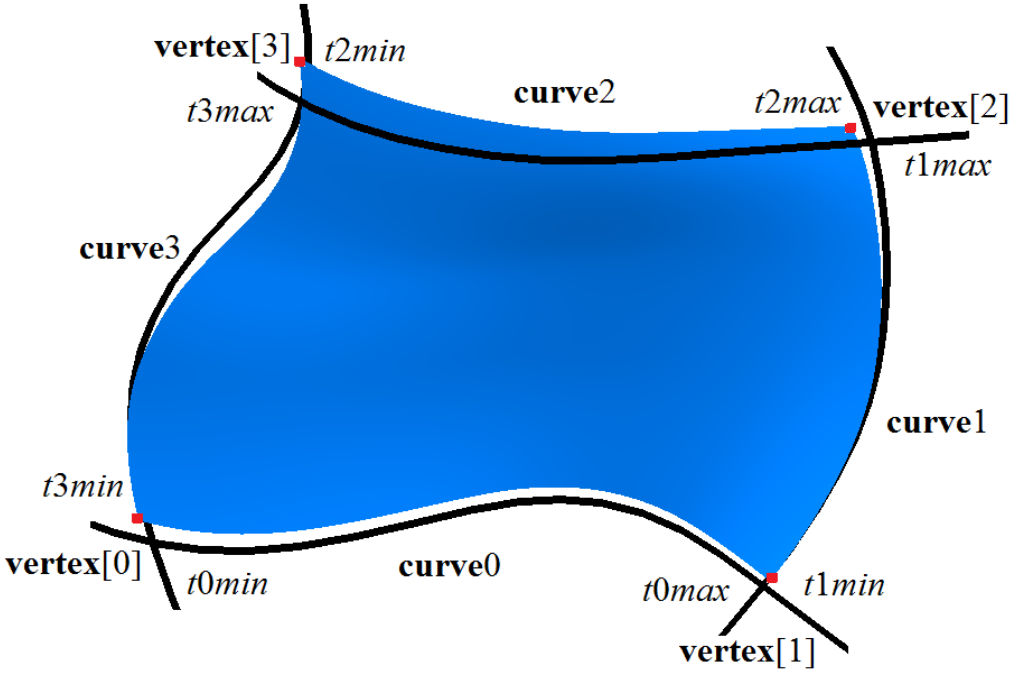


Рис. О.5.18.1.

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 **curve**3, при $v=0$ поверхность проходит по участку кривой **curve**0, при $u=1$ поверхность проходит по участку кривой **curve**1, при $v=1$ поверхность проходит по участку кривой **curve**2.

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* **curve**0, **curve**1, **curve**2, **curve**3, производной **curve**0V радиуса-вектора поверхности вдоль кривой **curve**0 по второму параметру поверхности, производной **curve**1U радиуса-вектора поверхности вдоль кривой **curve**1 по первому параметру поверхности, производной **curve**2V радиуса-вектора поверхности вдоль кривой **curve**2 по второму параметру поверхности, производной **curve**3U радиуса-вектора поверхности вдоль кривой **curve**3 по первому параметру поверхности, четырьмя угловыми точками **vertex**[4], четырьмя производными по первому параметру поверхности в угловых точках **vertex**U[4], четырьмя производными по второму параметру поверхности в угловых точках **vertex**V[4], четырьмя смешанными производными по первому и второму параметрам поверхности в угловых точках **vertex**UV[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 $0\le u\le 1$. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve**0, **curve**2, **curve**V0, **curve**V2, а кривые **curve**U1 и **curve**U3 совпадают.

Второй параметр поверхности v принимает значения на отрезке $0\le v\le 1$. Поверхность может быть периодической по второму параметру, если периодическими являются кривые **curve**1 и **curve**3, **curve**U1, **curve**U3, а кривые **curve**V0 и **curve**V2 совпадают.

Соседние кривые **curve**0, **curve**1, **curve**2, **curve**3 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], **vertex**U[4] , **vertex**V[4] , **vertex**UV[4]. Направления кривых для поверхности не имеют значения, но параметризация пар кривых **curve**0 и **curve**V0, **curve**2 и **curve**V2, **curve**1 и **curve**U1, **curve**3 и **curve**U3 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.

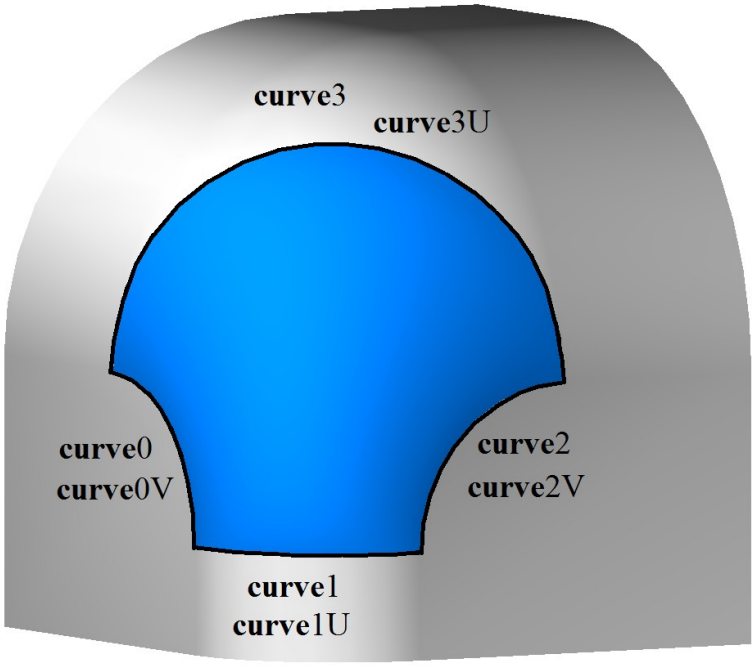


Рис. О.5.19.1.

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 **curve**3, при *v*=0 поверхность проходит по участку кривой **curve**0, при *u*=1 поверхность проходит по участку кривой **curve**1, при *v*=1 поверхность проходит по участку кривой **curve**2.

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 *type*0 сопряжения поверхности на краю, соответствующем второму параметру *vmin*, типом *type*1 сопряжения поверхности на краю, соответствующем первому параметру *umin*, типом *type*2 сопряжения поверхности на краю, соответствующем второму параметру *vmax*, типом *type*3 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]≤*v*≤*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.

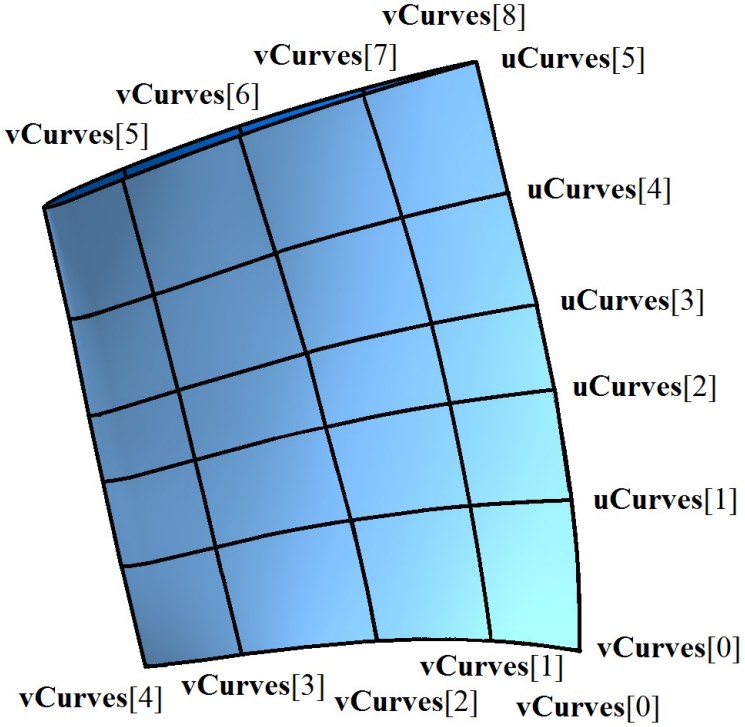


Рис. О.5.20.1.

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 *isPoleUmin*, *isPoleUmax*.

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*≤*u*≤*umax*, который соответствует области определения кривых **curves**. Поверхность может быть периодической по первому параметру, если периодическими являются все кривые **curves**. Пусть узловой вектор *knots* содержит *knotsCount* элементов, а количество кривых в множестве **curves** равно

curvesCount. Количества элементов в множествах связаны равенством *curvesCount*+*degree*=*knotsCount*.

Второй параметр поверхности *v* принимает значения на отрезке *vmin*≤*v*≤*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

$$\mathbf{s}(u,v) = \frac{\sum_{j=0}^{curvesCount-1} N_j^{degree}(v)\,\mathbf{curves}[j](u)}{\sum_{j=0}^{curvesCount-1} N_j^{degree}(v)},$$

где $N_j^{degree}(v)$ – B-сплайны порядка *degree* для *j*–ой кривой **curves**[*j*]. Поверхность соединения приведена на рис. О.5.21.1.

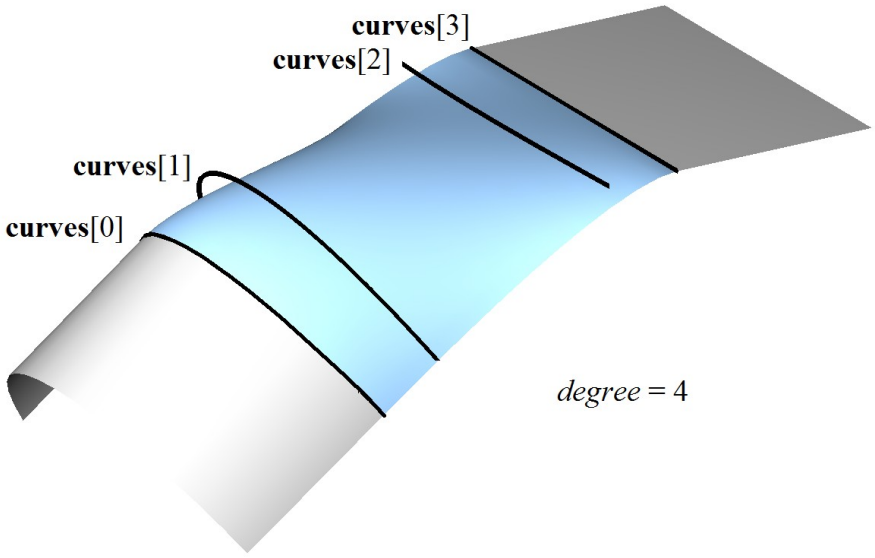


Рис. О.5.21.1.

Каждая кривая **s**(*const*,*v*) с фиксированным первым параметром поверхности *u*=const является NURBS-кривой порядка *degree*, построенной по точкам **curves**[*i*](*const*).

О.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,...,*vcount*–1, *j*=0,1,...,*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 *ucount*+*degree*=*uknotsCount*. Для периодической NURBS-поверхности количество элементов в множествах связаны равенством *knotsCount*+2*degree*–1=*knotsCount*.

В методе **PointOn**(double *u*, double *v*, MbCartPoint3D & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{r}(u,v) = \frac{\sum_{i=0}^{vcount-1}\sum_{j=0}^{ucount-1} N_i^{\,vdgree}(v)\,N_j^{\,udgree}(u)\,weight[i][j]\,\mathbf{points}[i][j]}{\sum_{i=0}^{vcount-1}\sum_{j=0}^{ucount-1} N_i^{\,vdgree}(v)\,N_j^{\,udgree}(u)\,weight[i][j]},$$

где $N_i^{vdegree}(v)$ и $N_j^{udegree}(u)$ – B-сплайны. На рис. О.5.22.1 приведены отрезки, связывающие соседние контрольные точки **points**[*i*][*j*].

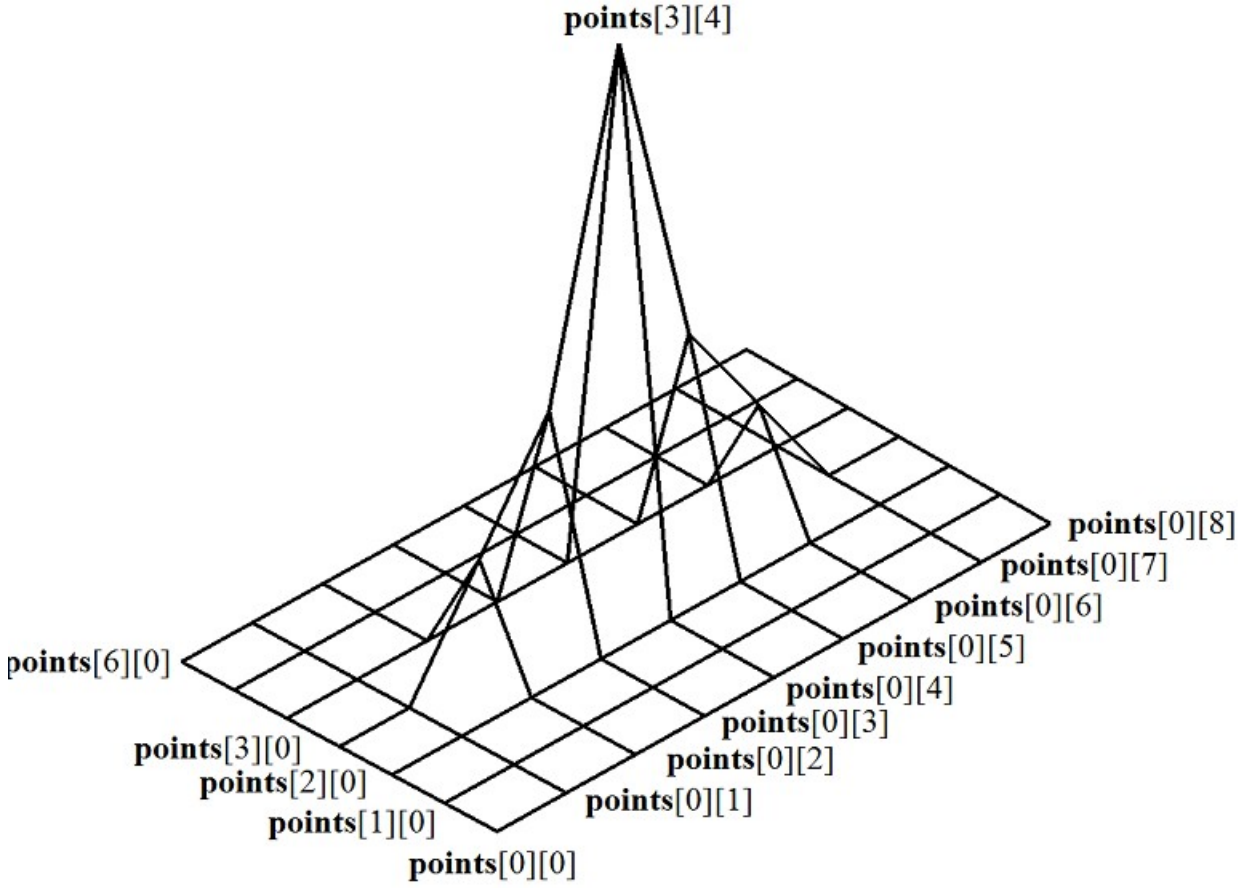


Рис. О.5.22.1.

Рис. О.5.22.1 демонстрирует условное расположение контрольных точек в узлах прямоугольной таблицы. NURBS-поверхность, построенная на тех же контрольных точках, приведена на рис. О.5.22.2.

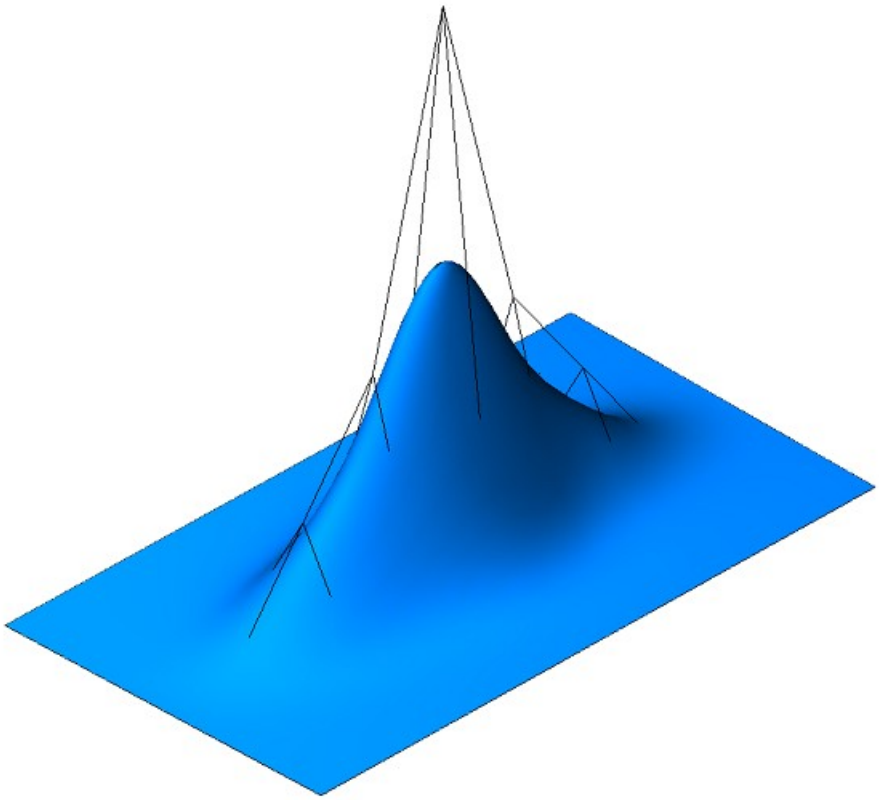


Рис. О.5.22.2.

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]≤*u*≤*uknots*[*ucount*], *vknots*[*vdegree*–1]≤*v*≤*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]≤*u*≤*uknots*[*ucount*+*udegree*–1], *vknots*[*vdegree*–1]≤*v*≤*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 *u*0*min*, *u*0*max*, граничными значениями второго параметра базовой поверхности *v*0*min*, *v*0*max*, признаками периодичности базовой поверхности *u*0*closed*, *v*0*closed*, 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 **normal**(*u*,*v*) – нормаль поверхности в заданной точке. Эквидистантная поверхность и её базовая поверхность приведены на рис. О.5.23.1.

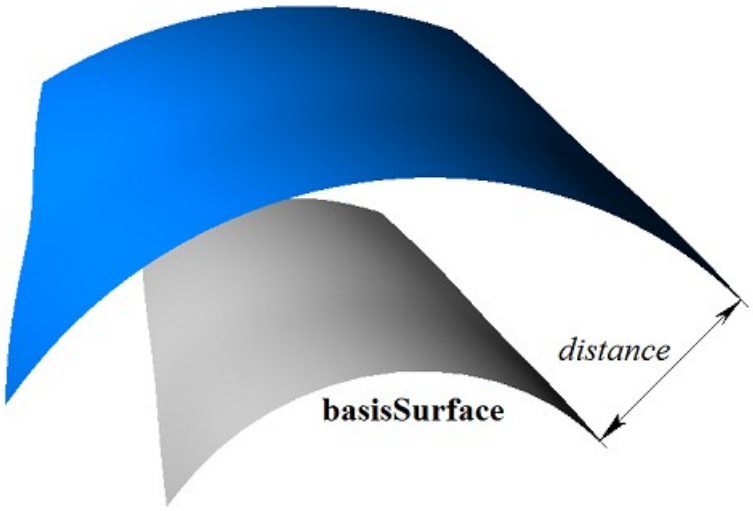


Рис. О.5.23.1.

Область изменения параметра эквидистантной поверхности может отличаться от области изменения параметров базовой поверхности. Область изменения первого параметра эквидистантной поверхности определяется неравенствами: *u*0*min*+*dumin*≤*u*≤*u*0*max*+*dumax*. Область изменения второго параметра эквидистантной поверхности определяется неравенствами: *v*0*min*+*dvmin*≤*v*≤*v*0*max*+*dvmax*. Эквидистантная поверхность с отрицательным смещением и увеличенной областью определения параметров и её базовая поверхность приведены на рис. О.5.23.2.

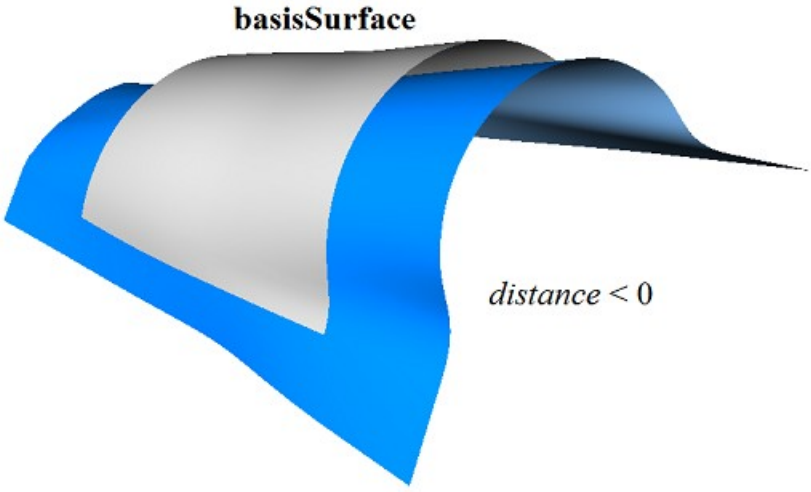


Рис. О.5.23.2.

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* **curve**1, кривой на второй сопрягаемой поверхности MbSurfaceCurve* **curve**2, способом построения фаски *form*, катетами фаски *distance*1 и *distance*2, граничными значениями *umin* и *umax*

параметра кривых **curve**1 и **curve**2, 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 **curve**1 и **curve**2 согласованы друг с другом и имеют одинаковую область определения параметра. Первый параметр поверхности *u* совпадает с параметром кривых **curve**1 и **curve**2, который является общим для них. Первый параметр поверхности принимает значения на отрезке *umin*≤*u*≤*umax*, который соответствует области определения кривых **curve**1 и **curve**2. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve**1 и **curve**2.

Второй параметр поверхности *v* принимает значения на отрезке: *vmin*≤*v*≤*vmax*. Значение *v*=*vmin* соответствует точке на кривой **curve**1, значение *v*=*vmax* соответствует точке на кривой **curve**2. Поверхность не может быть периодической по второму параметру.

В методе **PointOn**(double *u*, double *v*, MbCartPoint3D & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \mathbf{curve}1(u)\,(1-w) + \mathbf{curve}2(u)\,w,$$

где $w = \dfrac{v - vmin}{vmax - vmin}$. Поверхность фаски приведена на рис. О.5.24.1.

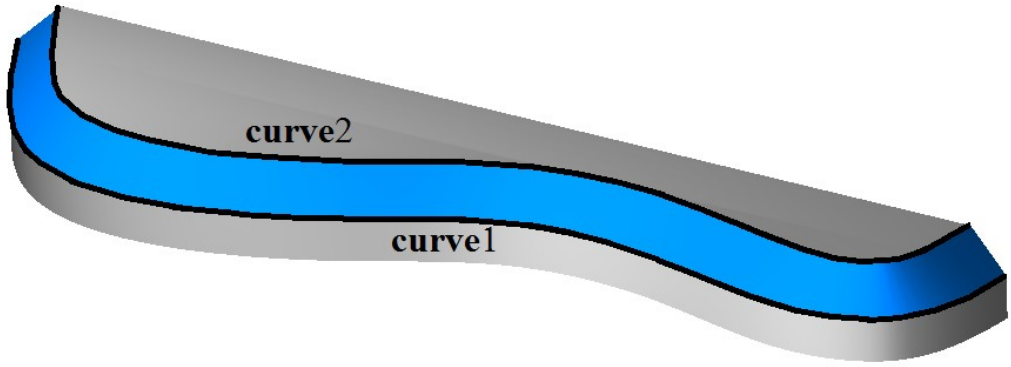


Рис. О.5.24.1.

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 **curve**1 и **curve**2 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* **curve**1, кривой на второй сопрягаемой поверхности MbSurfaceCurve* **curve**2, кривой MbCurve3D* **curve**0, функцией MbFunction* **weights**0 веса кривой **curve**0, способом построения скругления *form*, радиусами скругления *distance*1 и *distance*2, коэффициентом формы *conic*, граничными значениями *umin* и *umax* параметра кривых **curve**1, **curve**2, **curve**0 и функции **weights**0, 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 *byCurve*1 наличия кромки вдоль кривой **curve**2 или **curve**1 (*equable*=false). У поверхности есть ещё некоторые данные, которые не обязательны и служат для ускорения работы методов поверхности.

Кривые **curve**1, **curve**2, **curve**0 и функция **weights**0 согласованы друг с другом и имеют одинаковую область определения параметра. Первый параметр поверхности *u* совпадает с параметром кривых **curve**1, **curve**2, **curve**0 и функции **weights**0, который является общим для них. Первый параметр поверхности принимает значения на отрезке *umin*≤*u*≤*umax*, который соответствует области определения кривых **curve**1, **curve**2, **curve**0 и функции **weights**0. Поверхность может быть

периодической по первому параметру, если периодическими являются кривые **curve**1, **curve**2, **curve**0 и функция **weights**0.

Второй параметр поверхности *v* принимает значения на отрезке: *vmin*≤*v*≤*vmax*. Значение *v*=*vmin* соответствует точке на кривой **curve**1, значение *v*=*vmax* соответствует точке на кривой **curve**2. Поверхность не может быть периодической по второму параметру.

В методе **PointOn**(double *u*, double *v*, MbCartPoint3D & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v) = \frac{(1-v)^2\,\mathbf{curve}1(u) + 2t(1-v)weight0(u)\mathbf{curve}0(u) + v^2\mathbf{curve}2(u)}{(1-v)^2 + 2v(1-v)weight0(u) + v^2}.$$

Каждая кривая **s**(*const*,*v*) с фиксированным первым параметром поверхности *u*=const является NURBS-кривой третьего порядка, построенной по точкам **curve**1(*u*), **curve**0(*u*), **curve**2(*u*), у которой крайние точки имеют единичный вес, а средняя точка **curve**0(*u*) имеет вес **weights**0(*u*). При *conic*=_ARC_ функция **weights**0(*u*) вычисляется из условия, что все кривые **s**(*const*,*v*) являются дугами окружности. При *conic*≠_ARC_ функция **weights**0 является константой и равна коэффициенту формы *conic*. Поверхность скругления приведена на рис. О.5.25.1.

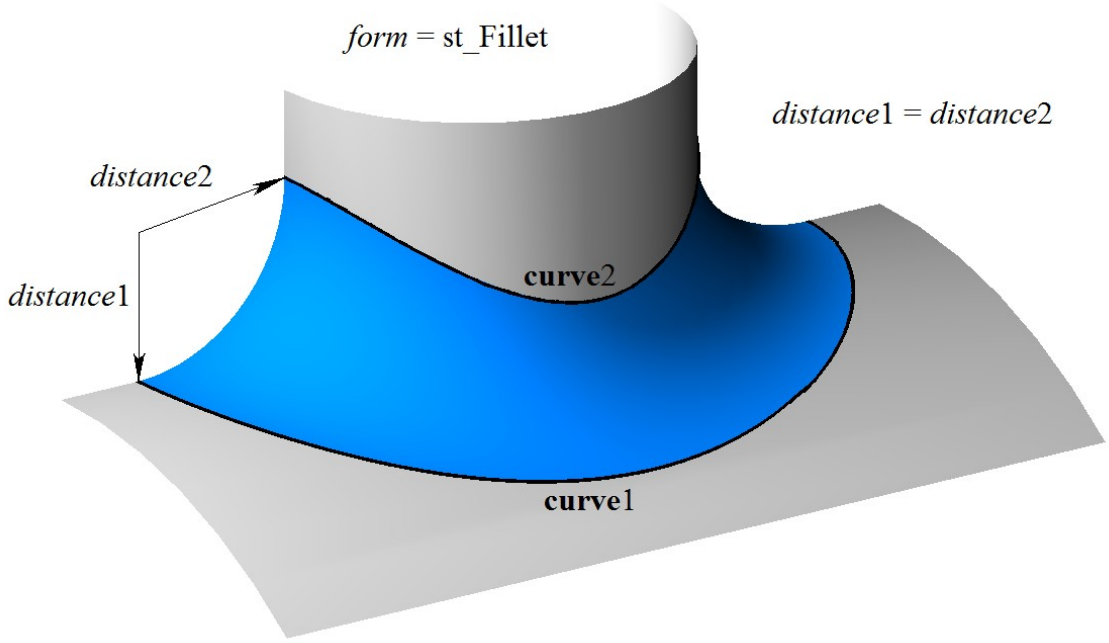


Рис. О.5.25.1.

В общем случае поверхность гладко сопрягается с поверхностями, на которых расположены кривые **curve**1(*u*) и **curve**2(*u*). В этом случае параметр *equable*=true. Если *equable*=false, то по одной из кривых **curve**1 или **curve**2 сопряжение является гладким, а другая кривая является кромкой. При *byCurve*1=true сопряжение является гладким по кривой **curve**1, в противном случае сопряжение является гладким по кривой **curve**2. Поверхность скругления с сохранением кромки приведена на рис. О.5.25.2.

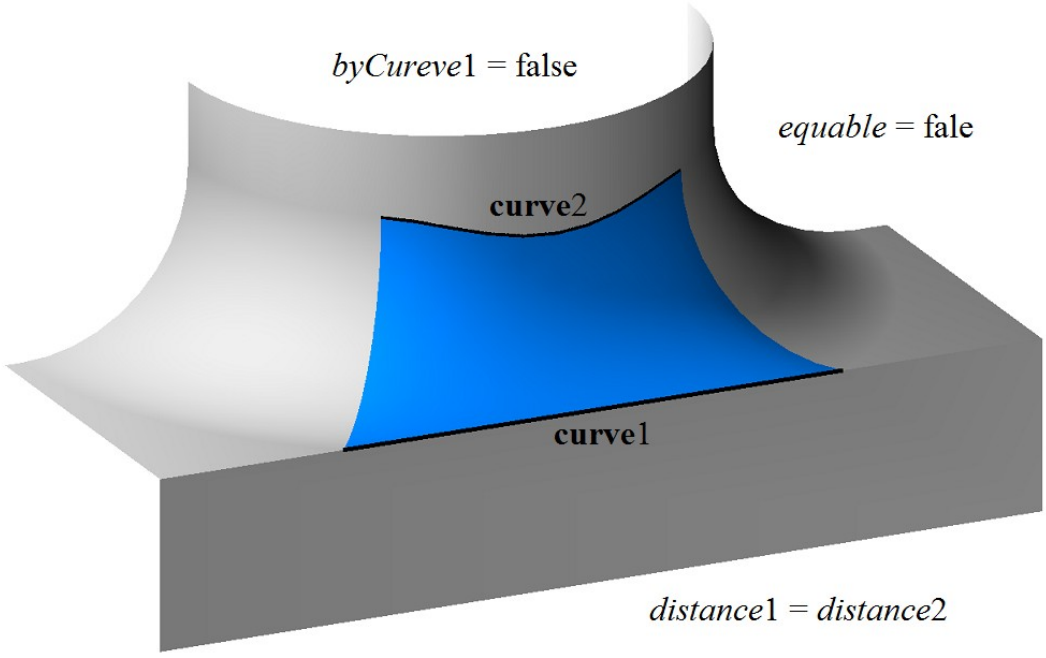


Рис. О.5.25.2.

Если радиусы скругления *distance*1 и *distance*2 не равны, то поверхность скругления в сечении **s**(*const*,*v*) с параметрами *u*=const описывает эллипс. Эллиптическая поверхность скругления приведена на рис. О.5.25.3.

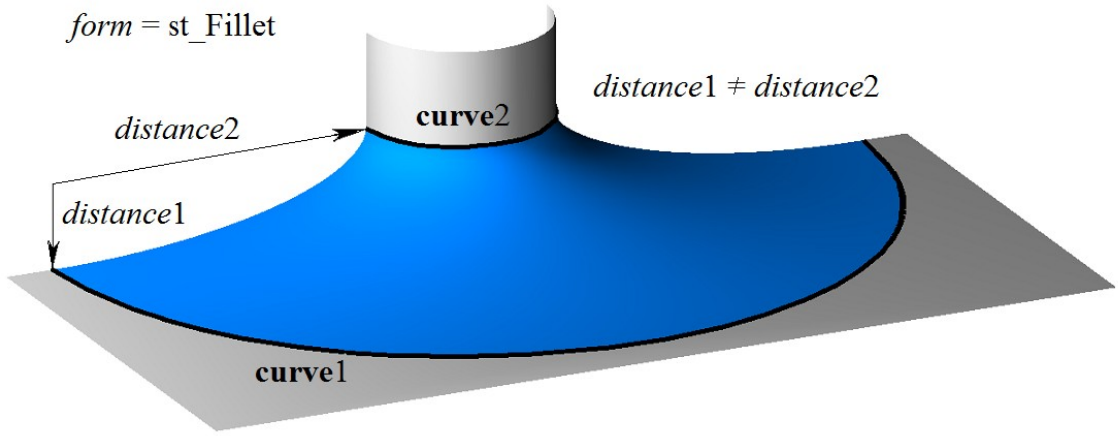


Рис. О.5.25.3.

В общем случае способ скругления *form*=st_Fillet. Если способ скругления *form*=st_Span, то *distance*1=*distance*2 и равны расстоянию между кривыми **curve**1 и **curve**2, а радиус поверхности скругления является переменным. Поверхность скругления с сохранением расстояния между опорными кривыми приведена на рис. О.5.25.4.

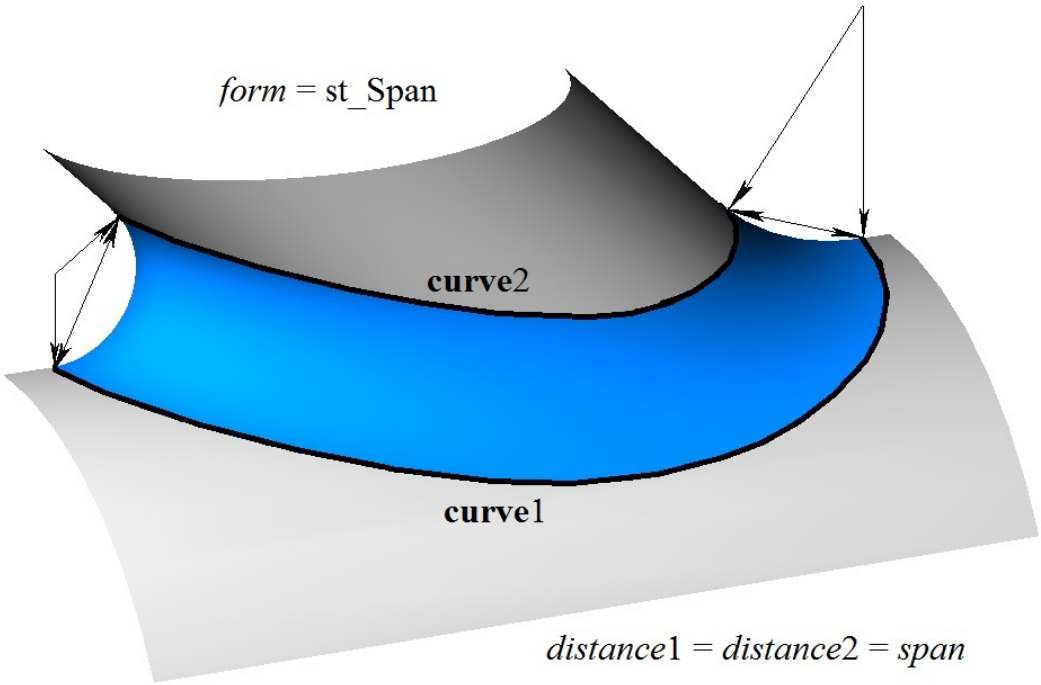


Рис. О.5.25.4.

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 **curve**0, **curve**1 и **curve**2 совпадают.

О.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* **curve**1, кривой на второй сопрягаемой поверхности MbSurfaceCurve* **curve**2, кривой MbCurve3D* **curve**0, функцией MbFunction* **weights**0 веса кривой **curve**0, функцией изменения радиуса MbFunction* **function**, способом построения скругления *form*, радиусами скругления *distance*1 и *distance*2, коэффициентом формы *conic*, граничными значениями *umin* и *umax* параметра кривых **curve**1, **curve**2, **curve**0 и функций **weights**0 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 **curve**1, **curve**2, **curve**0 и функции **weights**0, **function** согласованы друг с другом и имеют одинаковую область определения параметра. Первый параметр поверхности *u* совпадает с параметром кривых **curve**1, **curve**2, **curve**0 и функций **weights**0 и **function**, который является общим для них. Первый параметр поверхности принимает значения на отрезке *umin*≤*u*≤*umax*, который соответствует области определения кривых **curve**1, **curve**2, **curve**0 и функций **weights**0 и **function**. Поверхность может быть периодической по первому параметру, если периодическими являются кривые **curve**1, **curve**2, **curve**0 и функции **weights**0 и **function**.

Второй параметр поверхности *v* принимает значения на отрезке: *vmin*≤*v*≤*vmax*. Значение *v*=*vmin* соответствует точке на кривой **curve**1, значение *v*=*vmax* соответствует точке на кривой **curve**2. Поверхность не может быть периодической по второму параметру.

В методе **PointOn**(double *u*, double *v*, MbCartPoint3D & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v)=\frac{(1-v)^2\,\mathbf{curve}1(u)+2t(1-v)weight0(u)\mathbf{curve}0(u)+v^2\mathbf{curve}2(u)}{(1-v)^2+2v(1-v)weight0(u)+v^2}.$$

Каждая кривая **s**(*const*,*v*) с фиксированным первым параметром поверхности *u*=const является NURBS-кривой третьего порядка, построенной по точкам **curve**1(*u*), **curve**0(*u*), **curve**2(*u*), у которой крайние точки имеют единичный вес, а средняя точка **curve**0(*u*) имеет вес **weights**0(*u*). При изменении первого параметра поверхности радиусы скругления меняются по законам R1(*u*)=*distance*1·**function**(*u*) и R2(*u*)=*distance*2·**function**(*u*). При *conic*=_ARC_ функция **weights**0(*u*) вычисляется из условия, что все кривые **s**(*const*,*v*) являются дугами окружности. При *conic*≠_ARC_ функция **weights**0 является константой и равна коэффициенту формы *conic*. Поверхность скругления с переменным радиусом приведена на рис. О.5.26.1.

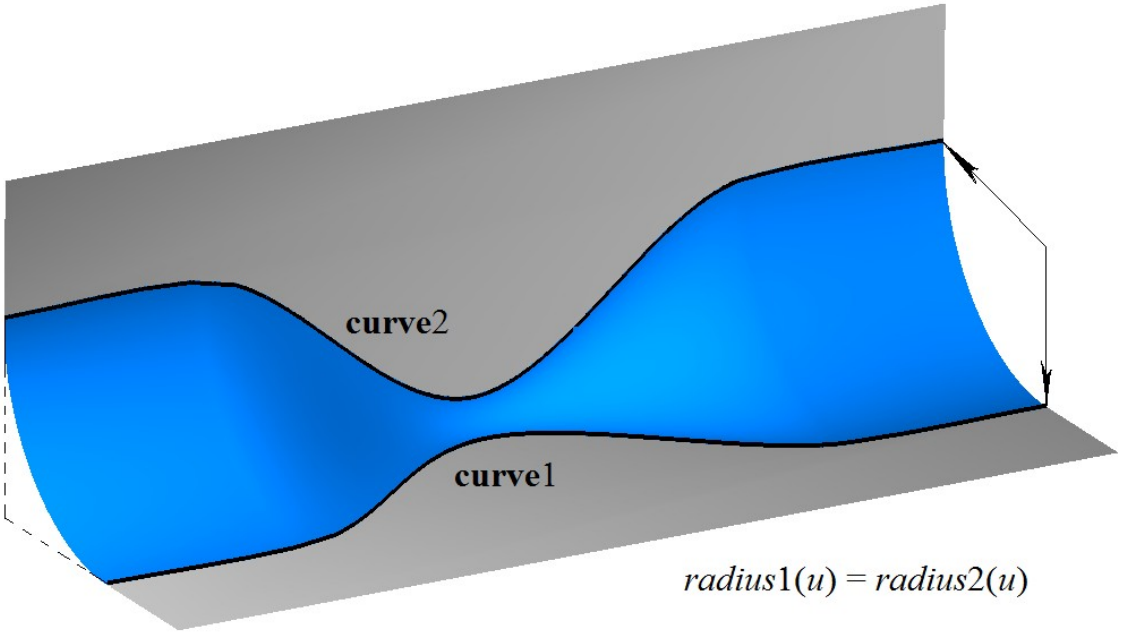


Рис. О.5.26.1.

Поверхность скругления с переменным радиусом гладко сопрягается с поверхностями, на которых расположены кривые **curve**1(*u*) и **curve**2(*u*), при этом параметр *equable*=true и *form*=st_Fillet.

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 **curve**0, **curve**1 и **curve**2 совпадают.

О.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 **RPArray**<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 & **s**) радиус-вектор поверхности **s** описывается векторной функцией

$$\mathbf{s}(u,v)=\mathbf{basisSurface}(u,v),\qquad u,v\in\Omega,$$

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. О.5.27.1

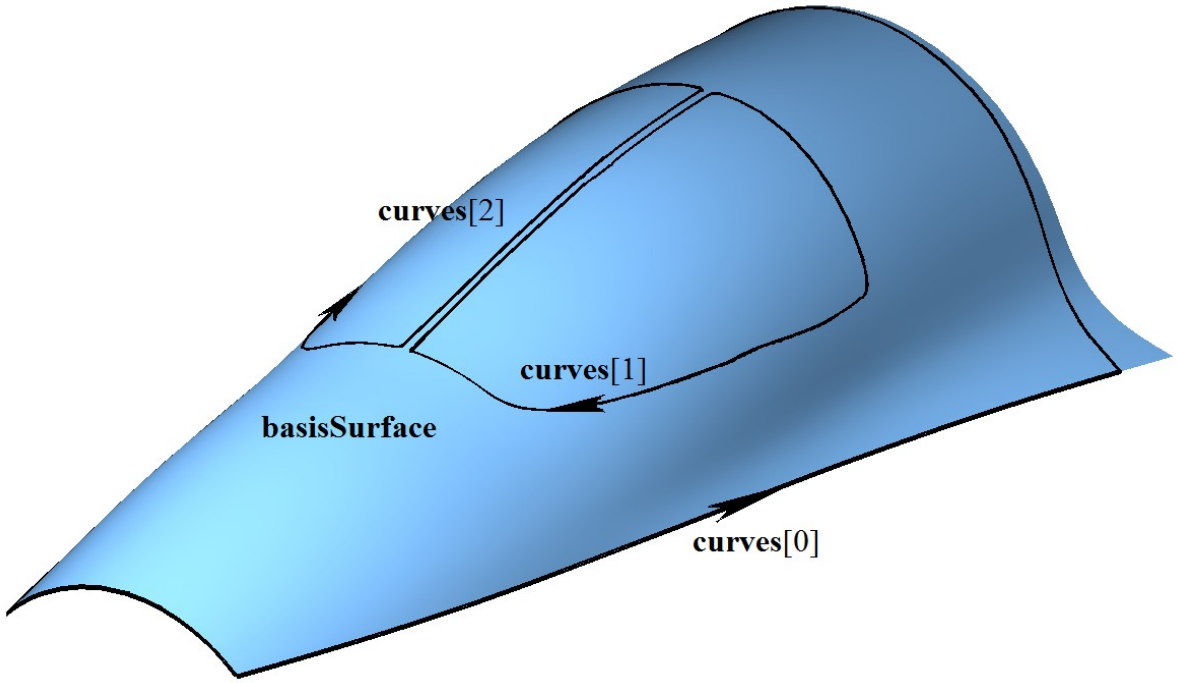


Рис. О.5.27.1

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 **r**(*u*,*v*) вычисляется методом **_PointOn**(*u*,*v*,**s**) в соответствии с правилами продолжения поверхности **basisSurface**. Поверхность с произвольной границей приведена на рис. О.5.27.2.

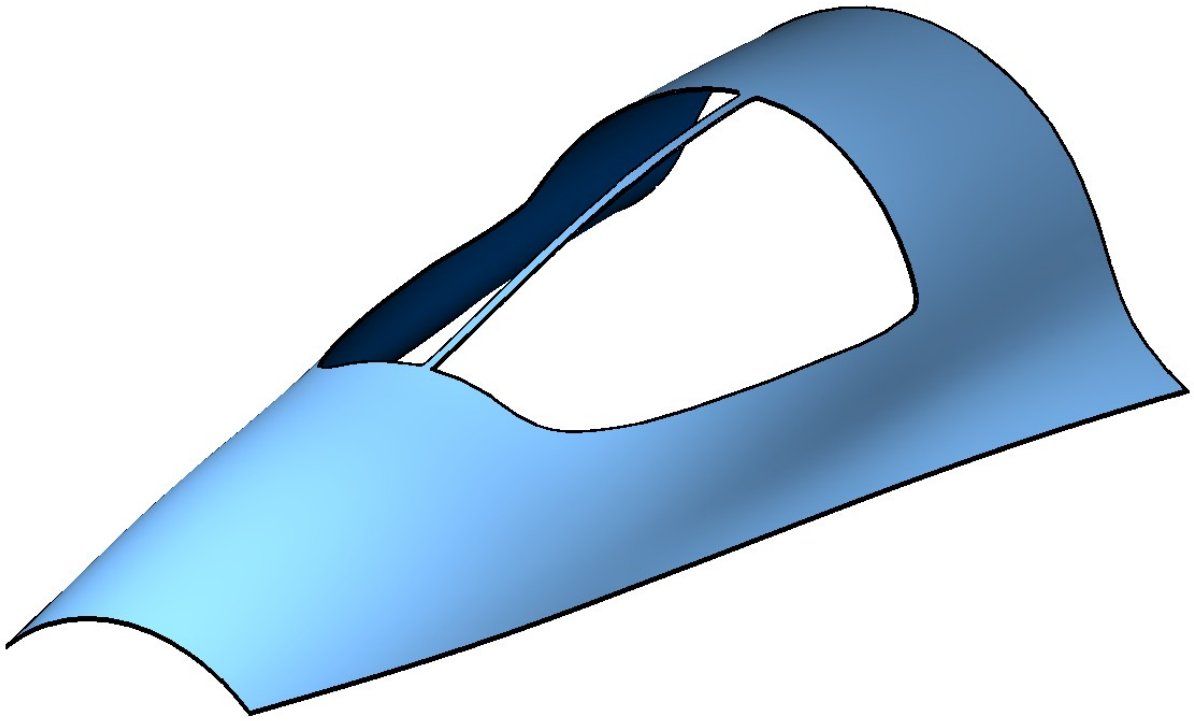


Рис. О.5.27.2.

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.

О.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.

О.6.1. Функция MbFunction

Класс MbFunction объявлен в файле function.h.

Функция MbFunction является абстрактным классом, наследником классов MbRefItem и TapeBase, рис. О.6.1.1.

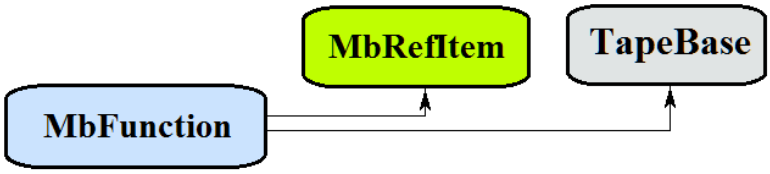


Рис. О.6.1.1.

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,
MbAnaliticalFunction – 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 продолжаются 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,
MbeFunctionType **IsA**(),
методы, обеспечивающие выдачу и редактирование внутренних данных объекта,
MbProperty & **CreateProperty**(MbePrompt name),
void **GetProperties**(MbProperties & *properties*),
void **SetProperties**(MbProperties & *properties*).

Все функции, как правило, не имеют изломов. Функция MbAnaliticalFunction, 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 $0 \le t \le 1$. 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 *value*1, *value*2 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 $tmin \le t \le tmax$. 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 *firctList*, 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,...,*splinesCount*, где *splinesCount*=*tList*.MaxIndex(), проходит через контрольную точку *valueList*[*i*] и имеет в ней производную *firctList*[*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*]≤*t*≤*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

$$f(t) = (1 - 3w^2 + 2w^3)valueList[i] + (3w^2 + 2w^3)valueList[i+1] +$$
$$+ \left((w - 2w^2 + w^3)valueList[i] + (-w^2 + w^3)valueList[i+1]\right)\left(tList[i+1] - tList[i]\right),$$

где $w = \dfrac{t - tList[i]}{tList[i+1] - tList[i]}$ - локальный параметр рабочего участка *tList*[*i*]≤*t*≤*tList*[*i*+1]. Кубическая функция Эрмита приведена на рис. О.6.4.1.

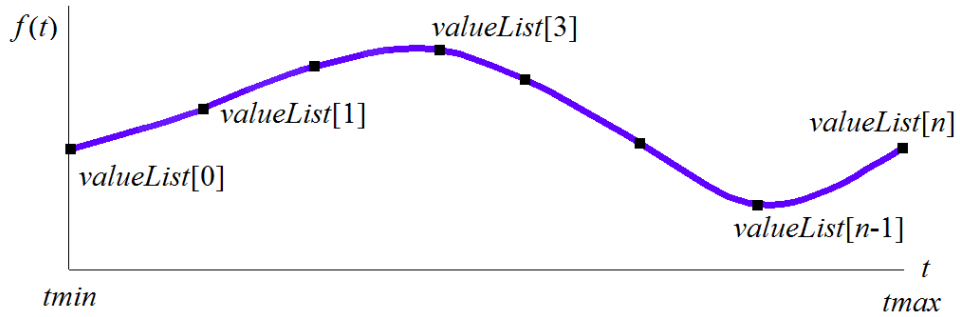


Рис. О.6.4.1.

Область определения параметра функции располагается в пределах *tmin*≤*t*≤*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 *firctList*[*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,...,*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*]≤*t*≤*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

$$\boldsymbol{r}(t) = (1 - w)valueList[i] + wvalueList[i+1] +$$

$$+\left((-2w+3w^2-w^3)valueList[i]+(-w+w^3)valueList[i+1]\right)\frac{(tList[i+1]-tList[i])^2}{6},$$

где $w=\frac{t-tList[i]}{tList[i+1]-tList[i]}$ - локальный параметр рабочего участка $tList[i]\le t\le tList[i+1]$. Кубическая сплайн-функция приведена на рис. О.6.5.1.

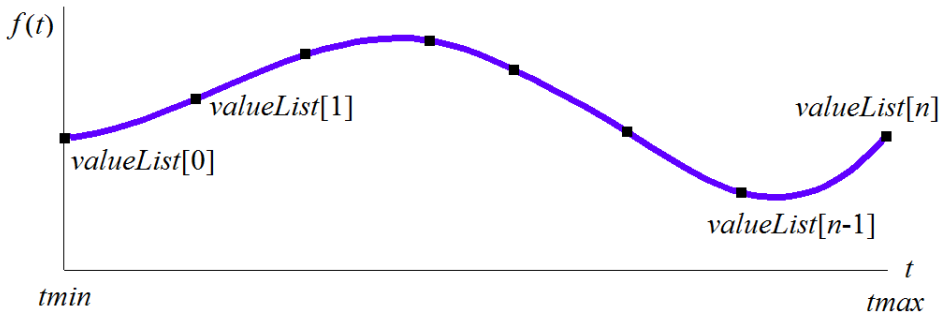


Рис. О.6.5.1.

Область определения параметра функции располагается в пределах $tmin\le t\le tmax$, 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 условии 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 $tmin\le t\le tmax$. 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.

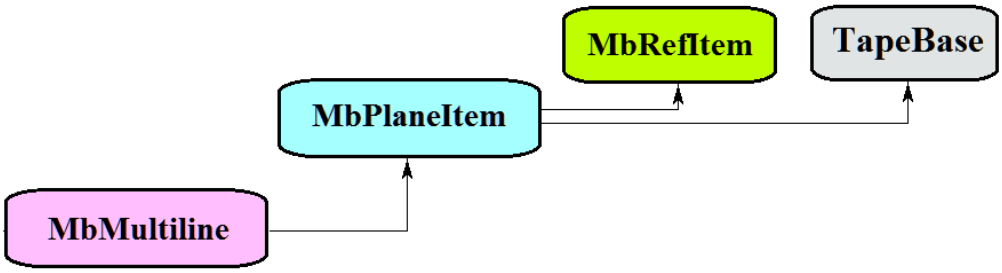


Рис. О.6.7.1.

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.

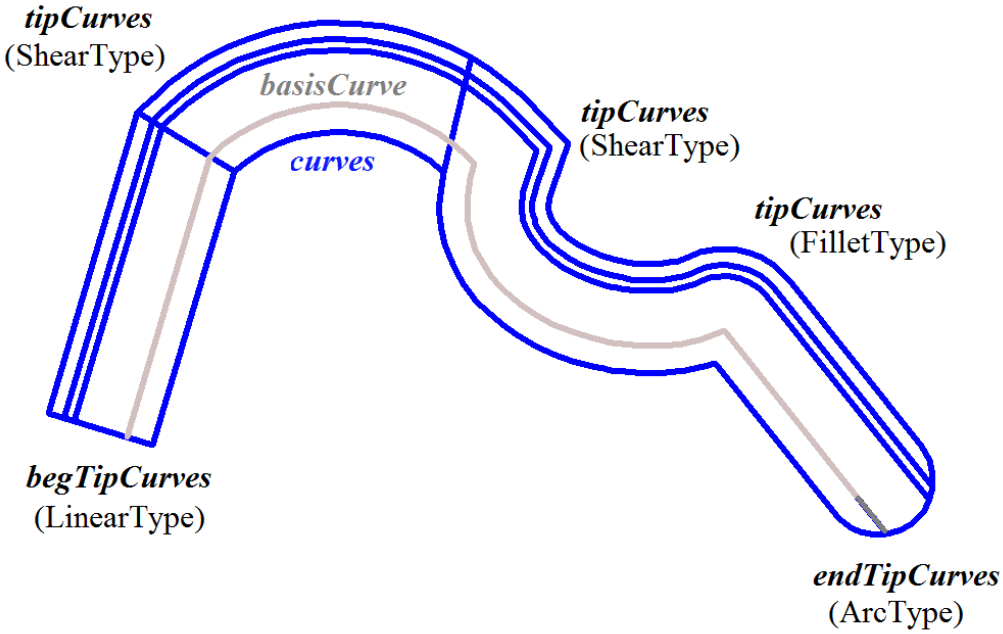


Рис. О.6.7.2.

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.

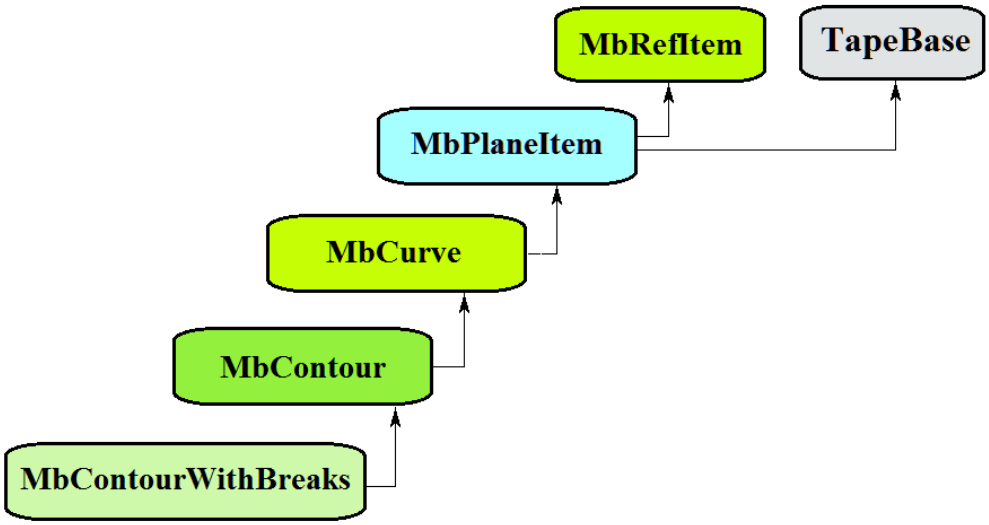


Рис. О.6.8.1.

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.

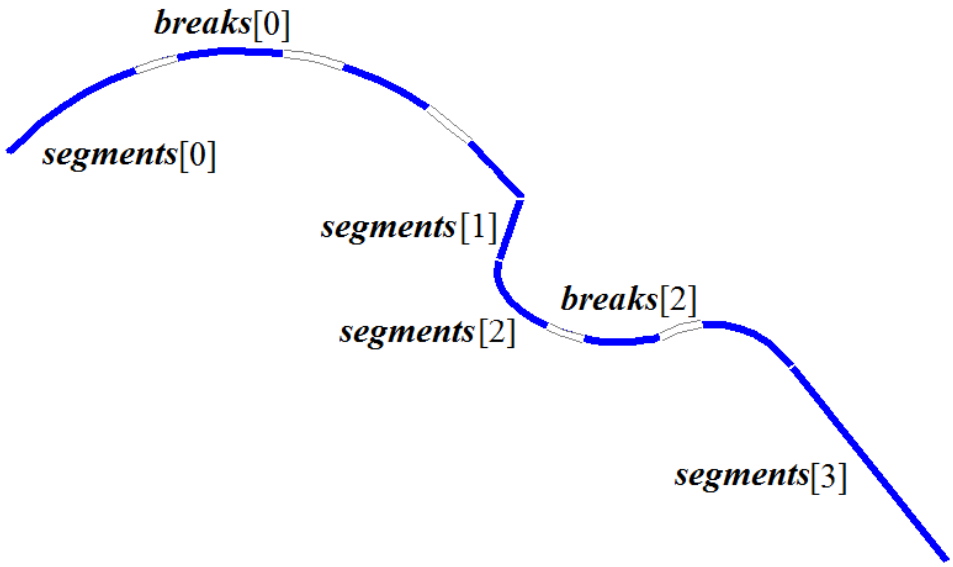


Рис. О.6.8.2.

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.

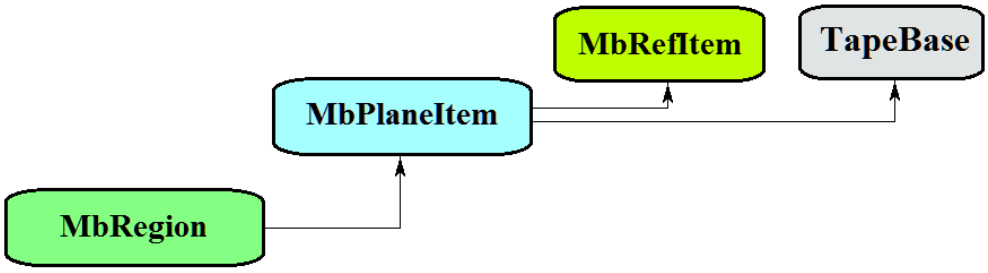


Рис. О.6.9.1.

Регион описывается множеством контуров ***contours***. Контуры 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. Контуры 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.

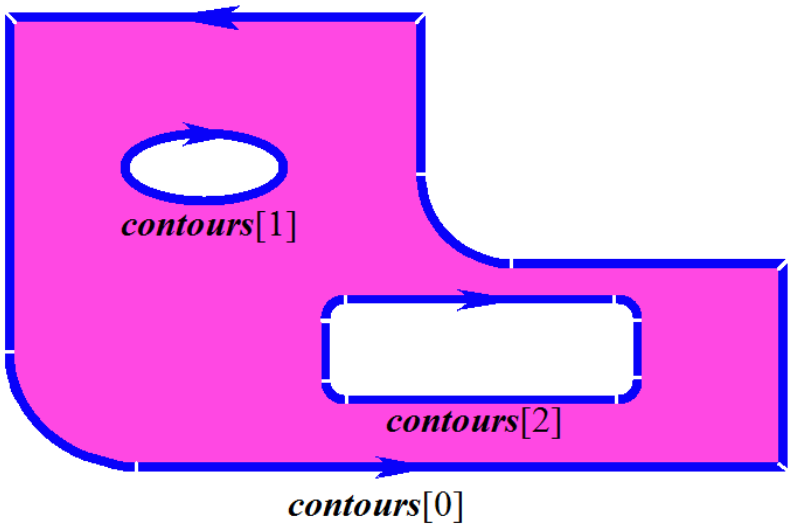


Рис. О.6.9.2.

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.

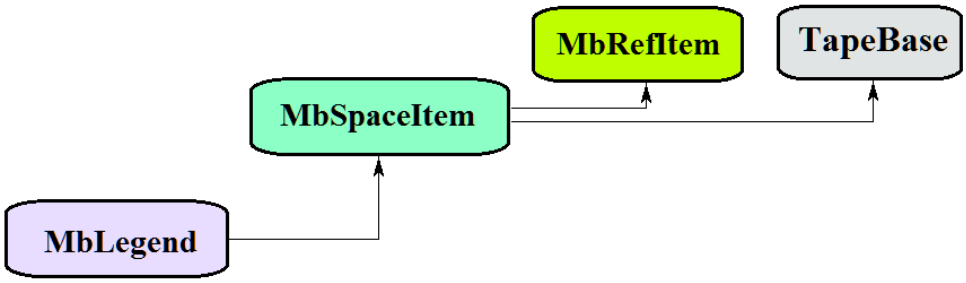


Рис. О.6.10.1.

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.

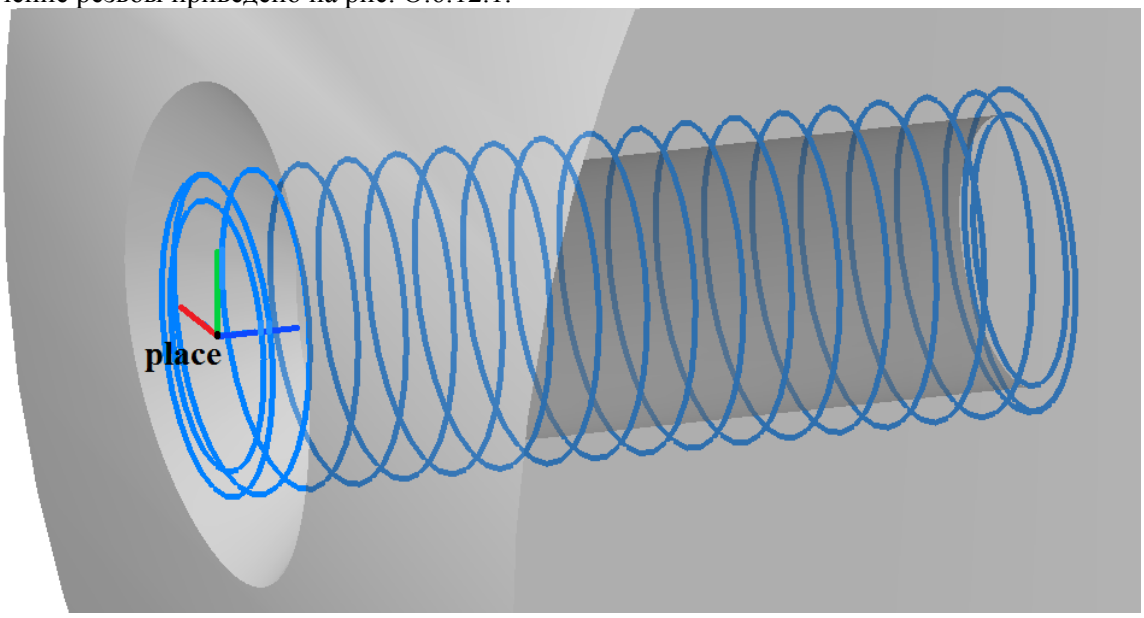


Рис. О.6.12.1.

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.

O.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**.

O.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.

О.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.

О.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. О.7.1.1.

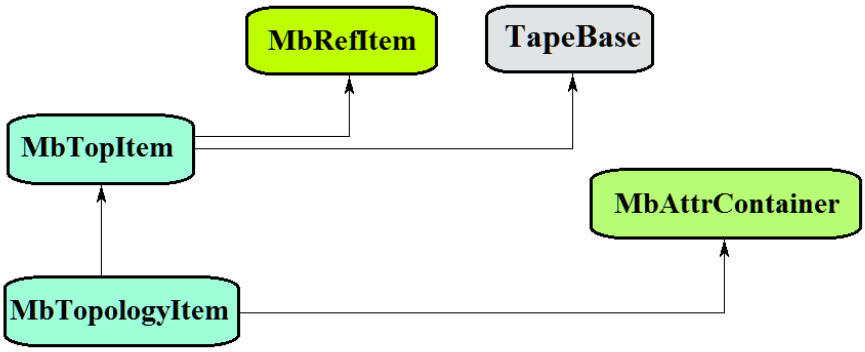


Рис. О.7.1.1

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,
MbeTopologyType **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.

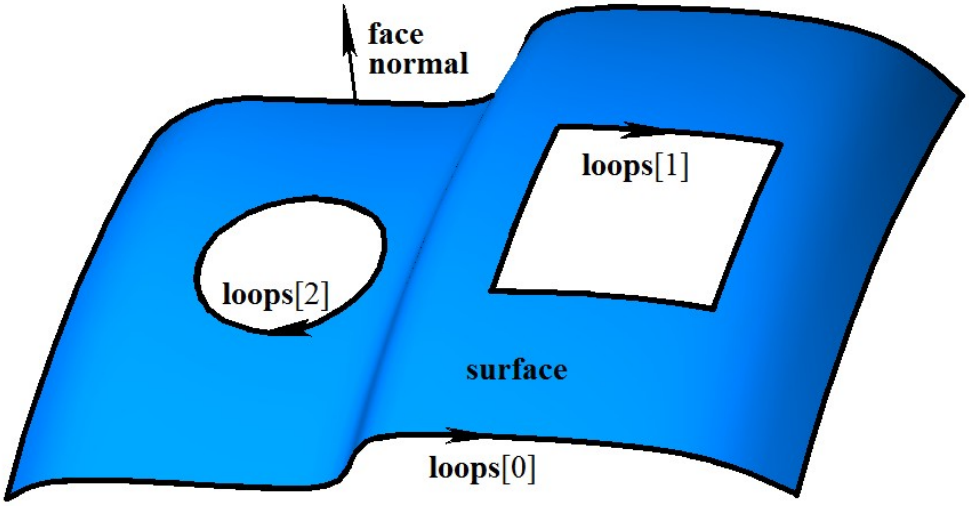


Рис. О.7.2.1.

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.

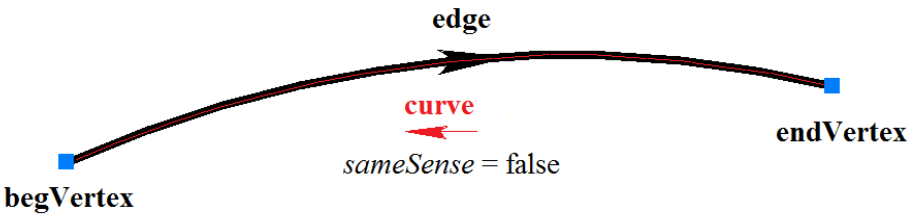


Рис. О.7.3.1.

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.

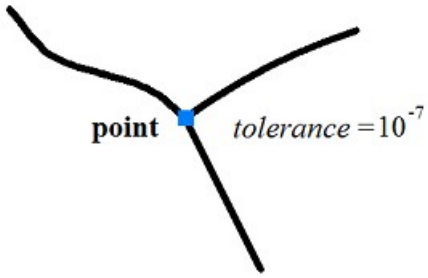


Рис. О.7.4.1.

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.

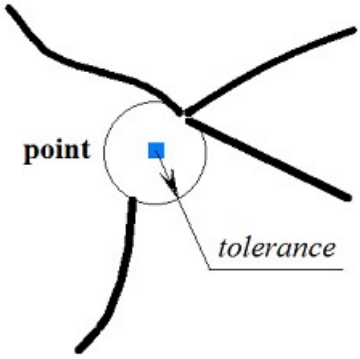


Рис. О.7.4.2.

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.

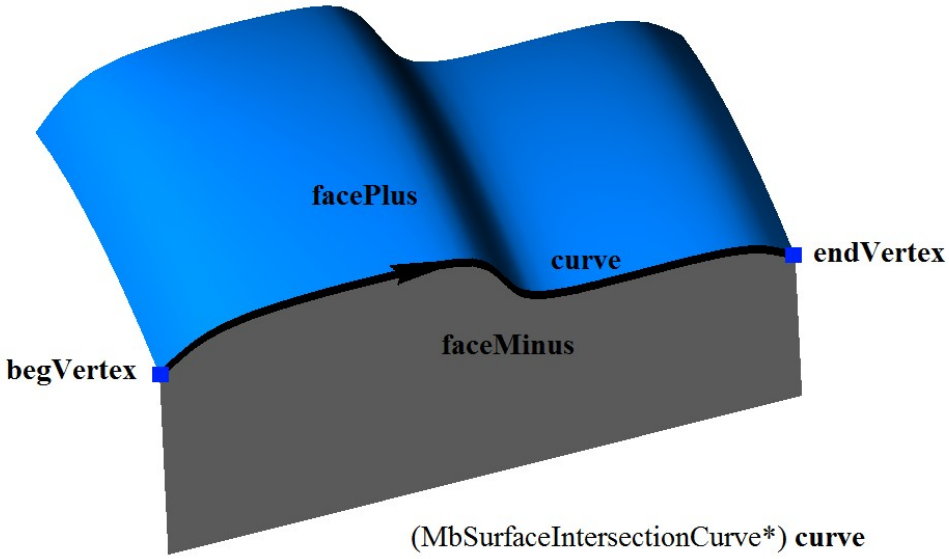


Рис. О.7.5.1.

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. О.7.5.2.

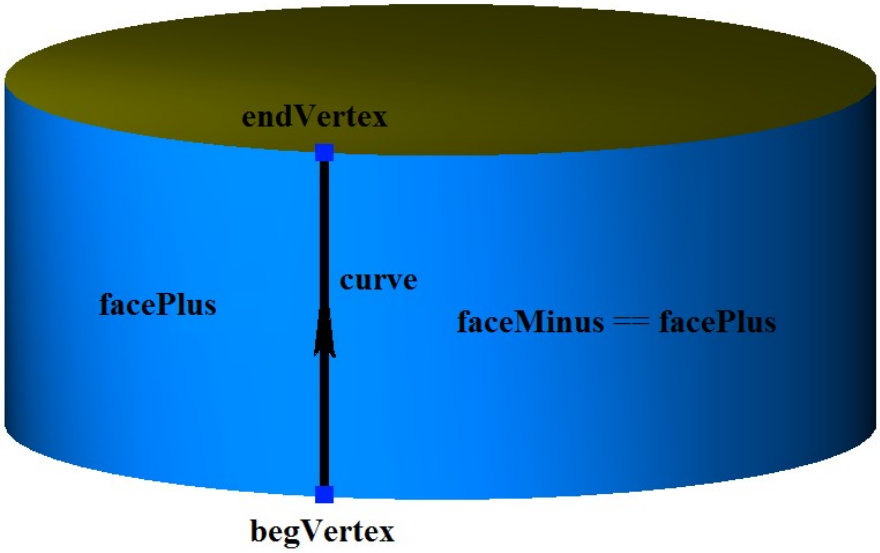


Рис. О.7.5.2.

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. О.7.5.3.

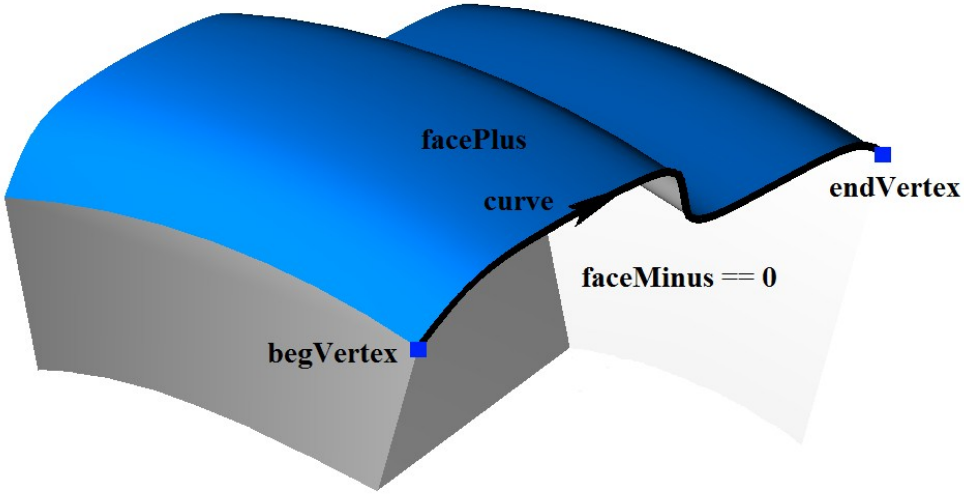


Рис. О.7.5.3.

Кривая пересечения краевого ребра имеет следующие данные:
curve->curveOne.suface == curve->curveTwo.suface и
curve->curveOne.curve == curve->curveTwo. curve.

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. О.7.5.4.

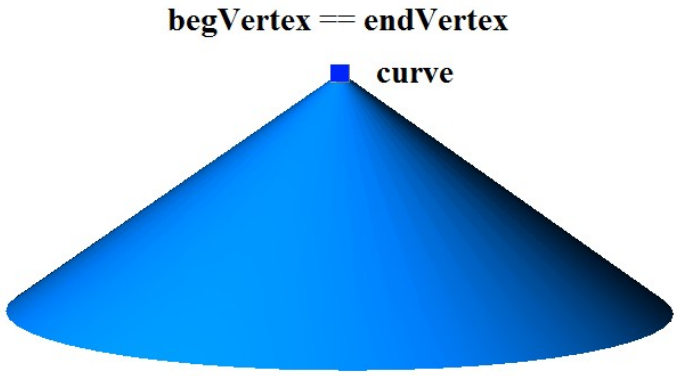


Рис. О.7.5.4.

У полюсного ребра указатель на грань слева или грань справа от ребра равен нулю. Кривая пересечения полюсного ребра имеет следующие данные:
curve->curveOne.suface == **curve->curveTwo.suface**, 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 RPArray<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.

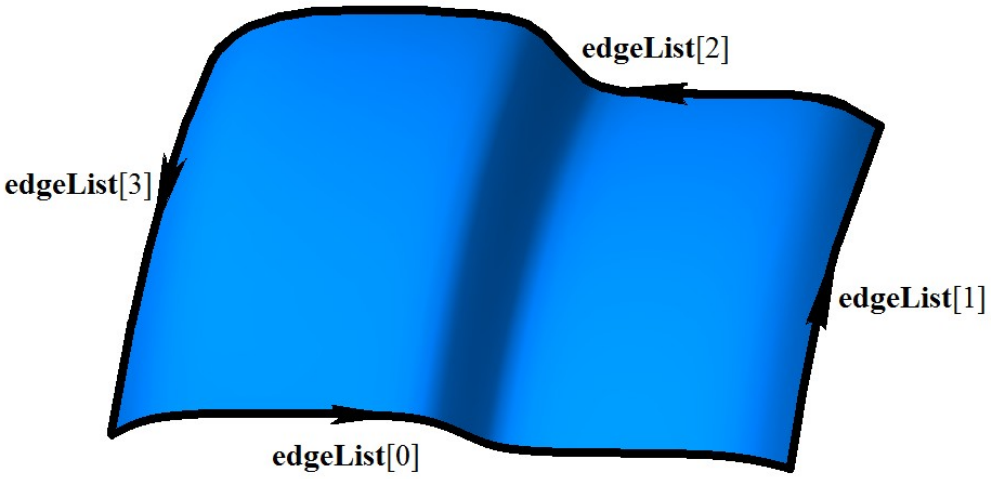


Рис. О.7.6.1.

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.

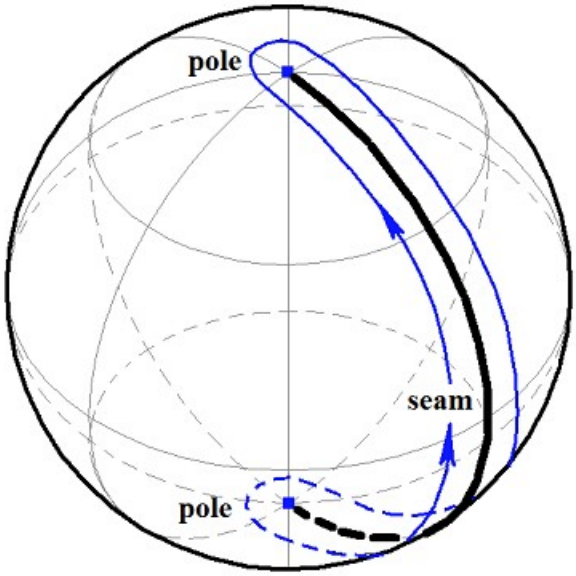


Рис. О.7.6.2.

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**.

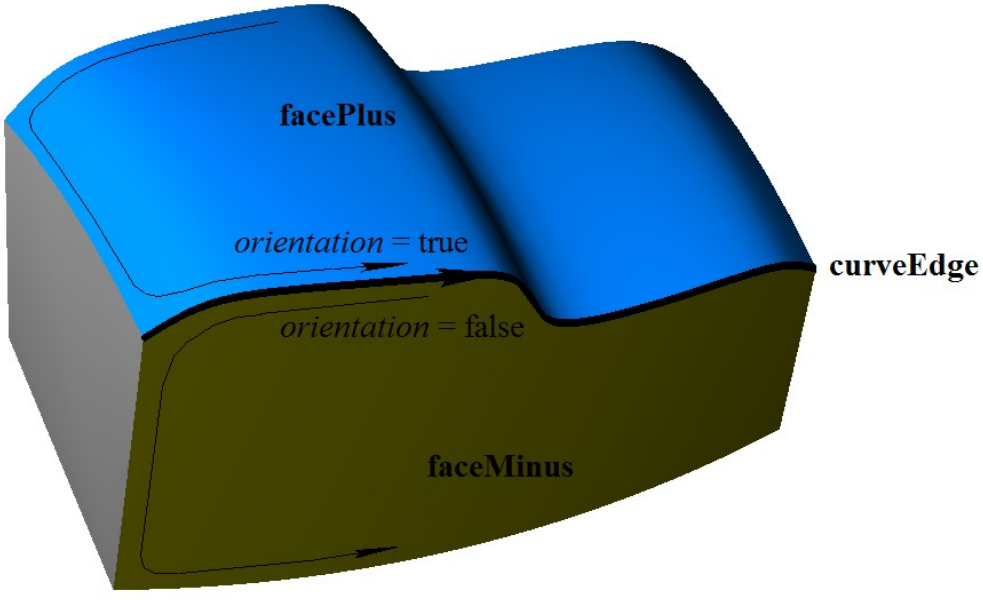


Рис. О.7.7.1.

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 пересекает сама себя.

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**.

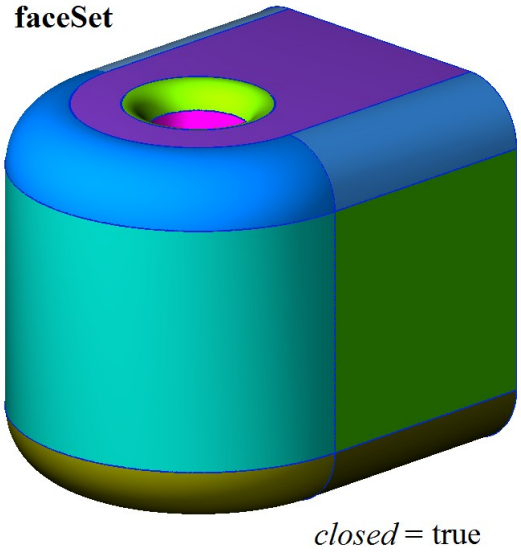


Рис. О.7.8.1.

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*. Полюсное ребро не является краевым, так как не образует край.

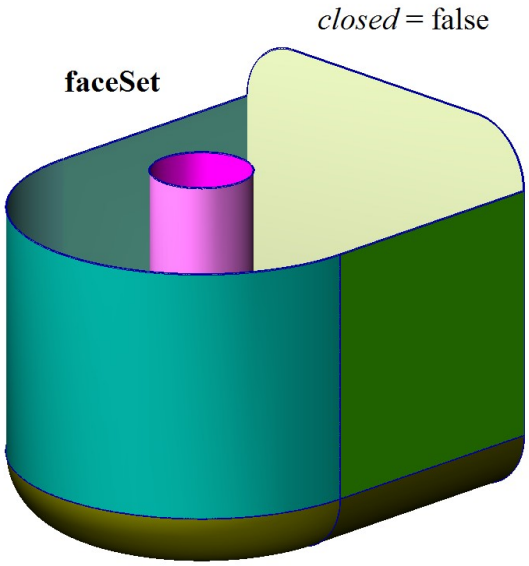


Рис. О.7.8.2.

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.

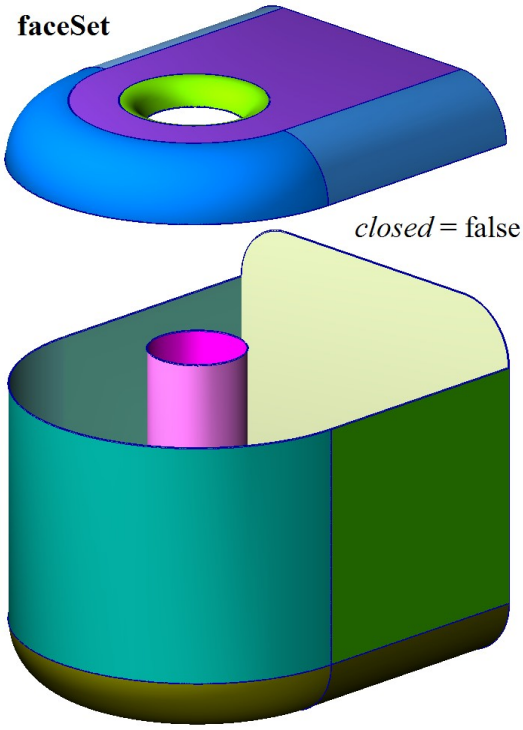


Рис. О.7.8.3.

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.

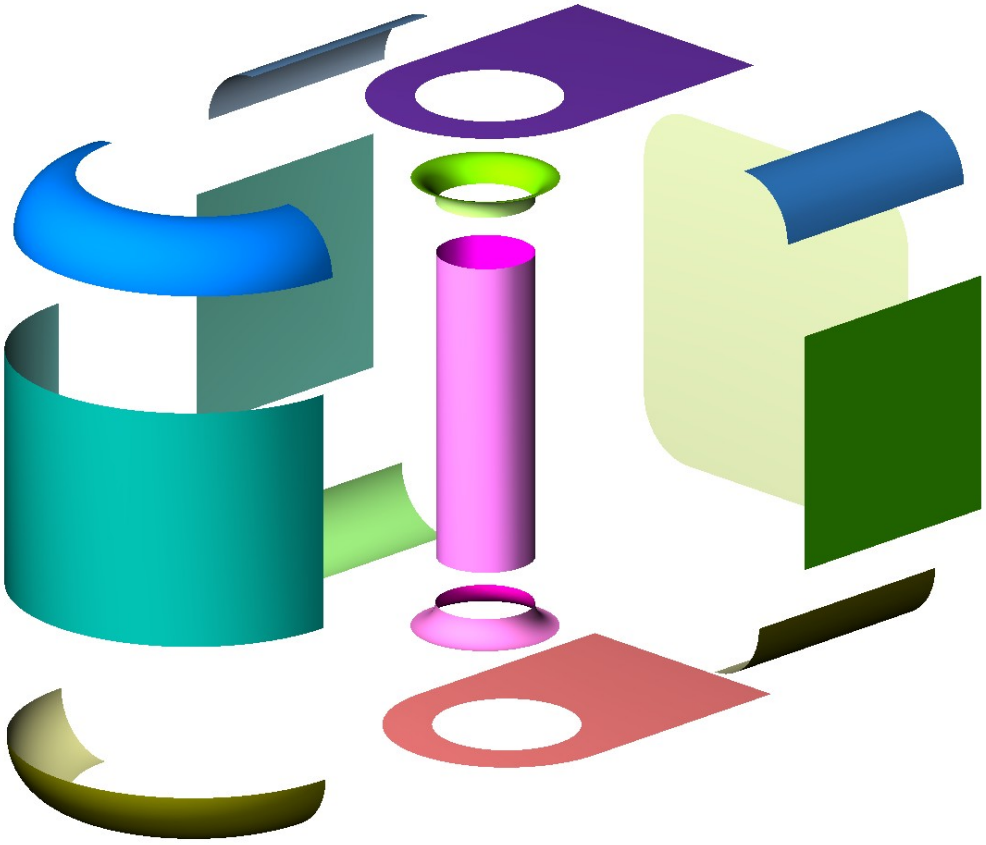


Рис. О.7.8.4.

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.

О.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 MbeCopyMode. Как правило, в методы построения вместе с модифицируемым множеством граней передается параметр типа MbeCopyMode, который управляет передачей граней, ребер и вершин от исходного объекта построенному объекту.

Перечисление MbeCopyMode объявлено в файле mb_enum.h. Параметр типа MbeCopyMode 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 значение 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 измененных 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 множества граней. Копируемое множество граней и их копия не будут связаны между собой.

О.7.10. Именование граней, рёбер и вершин

Грани, рёбра и вершины в структуре данных имеют имя MbName. Класс MbName объявлен в файле name_item.h. Именование 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. 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. 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.

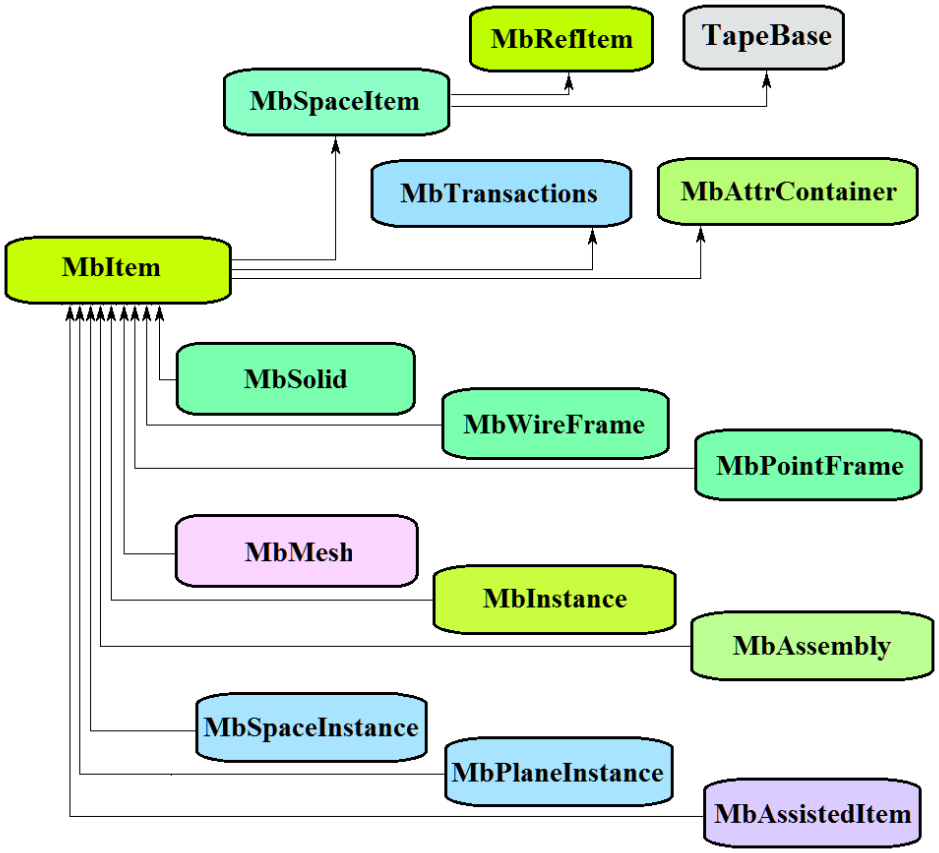


Рис О.8.1.1.

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 регистрирований 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,
MbInctance – вставка объекта геометрической модели,
MbAssembly – сборочная единица объектов геометрической модели,
MbAssistingItem – вспомогательный объект,
MbSpaceInctance – вставка трёхмерного объекта,
MbPlaneInctance – 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*).

О.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 связных множеств точек. Тип связности *multiState* сообщает о том, что тело описывает одно связное множество точек, или, что тело описывает несколько связных 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 О.8.2.1.

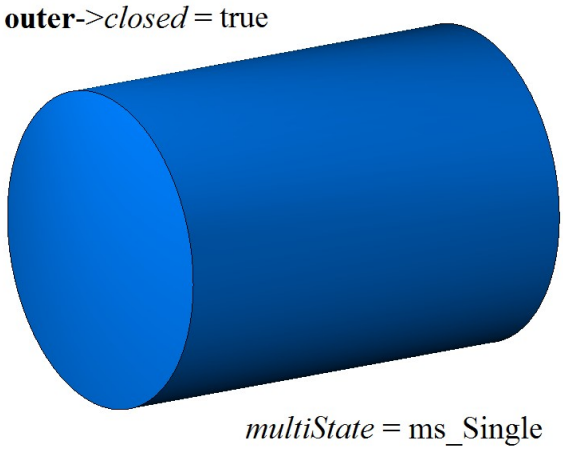


Рис. О.8.2.1.

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 О.8.2.2.

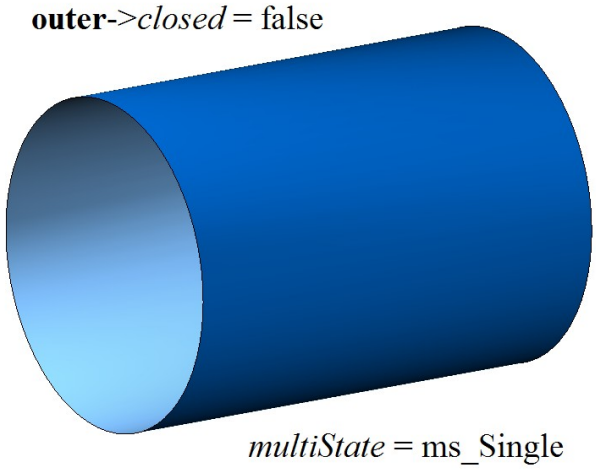


Рис. О.8.2.2.

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 О.8.2.3 приведено замкнутое тело, которое описывается двумя замкнутыми оболочками, одной внешней, а другой внутренней, располагающейся внутри внешней оболочки.

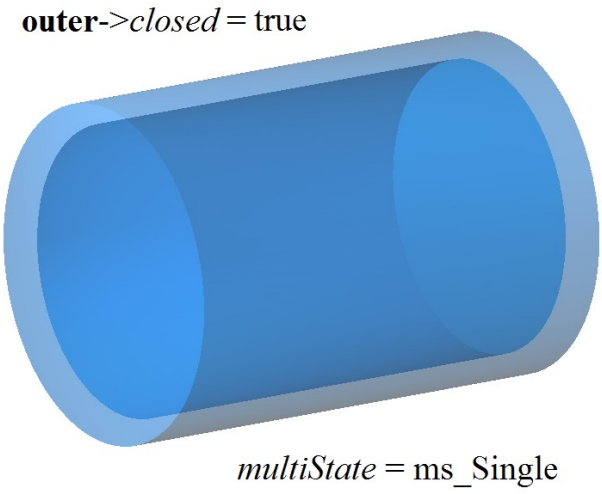


Рис. О.8.2.3.

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. О.8.2.4.

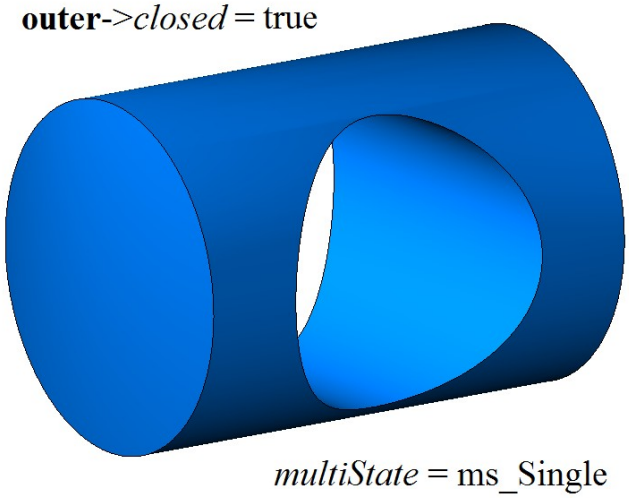


Рис. О.8.2.4.

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. О.8.2.5.

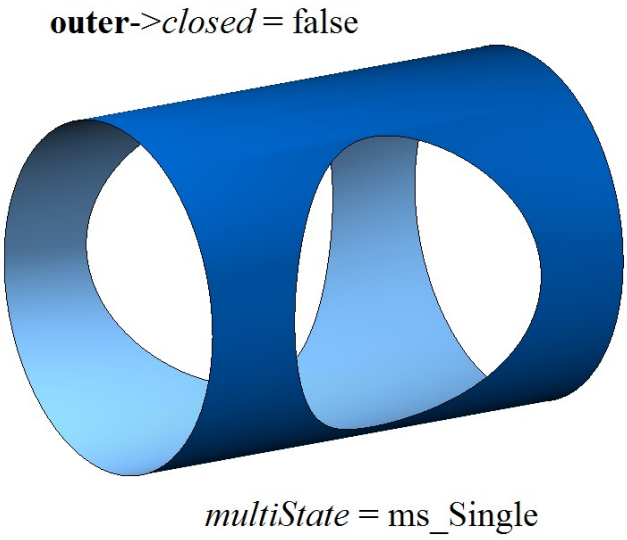


Рис. О.8.2.5.

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. О.8.2.6.

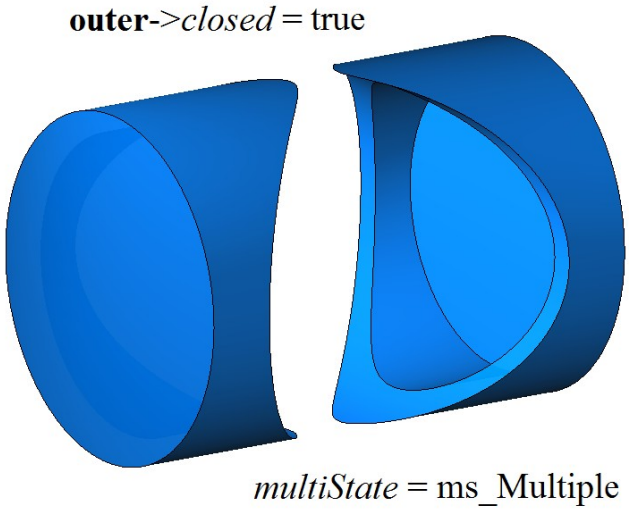


Рис. О.8.2.6.

Такое тело может быть разделено на два односвязных замкнутых тела методом **::DetachParts**(...). Тело на рис. О.8.2.6 выполнено полупрозрачным. Двусвязное тело, описываемое двумя незамкнутыми оболочками, приведено на рис. О.8.2.7.

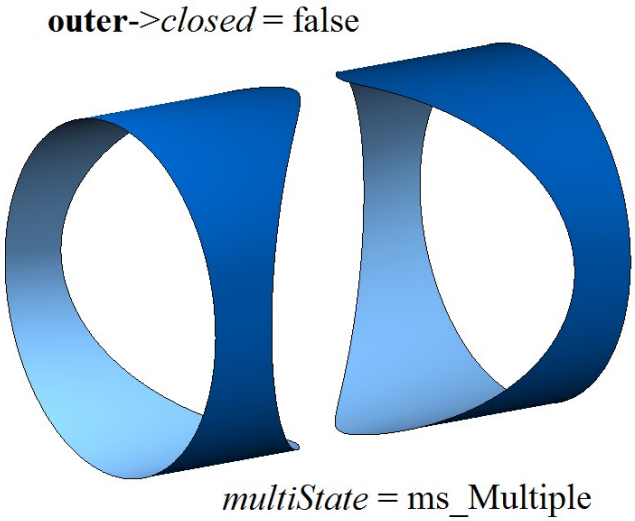


Рис. О.8.2.7.

Такое тело может быть разделено на два односвязных незамкнутых тела.

Тела создаются методами геометрического ядра C3D. 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 связных 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 связных множеств точек. Количеством связных частей *parts* сообщает о том, что каркас описывает одно связное множество точек, или, что каркас описывает несколько связных 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.

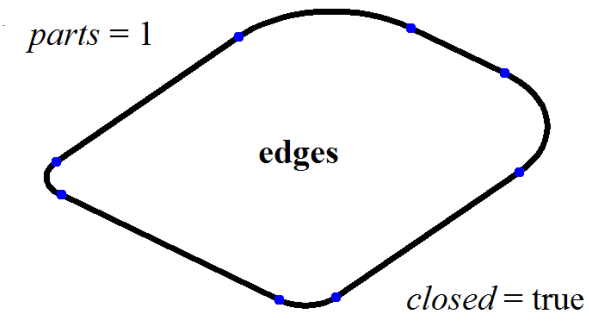


Рис. О.8.3.1.

Незамкнутый проволочный каркас приведен на рис О.8.3.2.

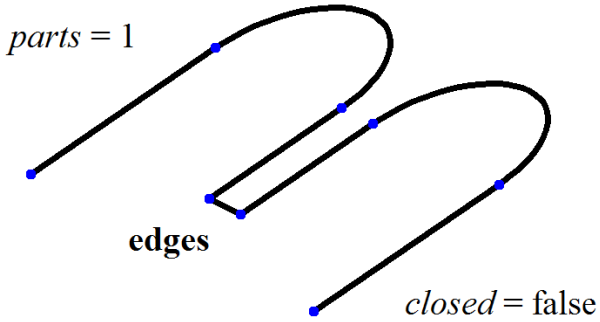


Рис. О.8.3.2.

На рис О.8.3.3 приведен незамкнутый проволочный каркас, состоящий из двух связных множеств рёбер.

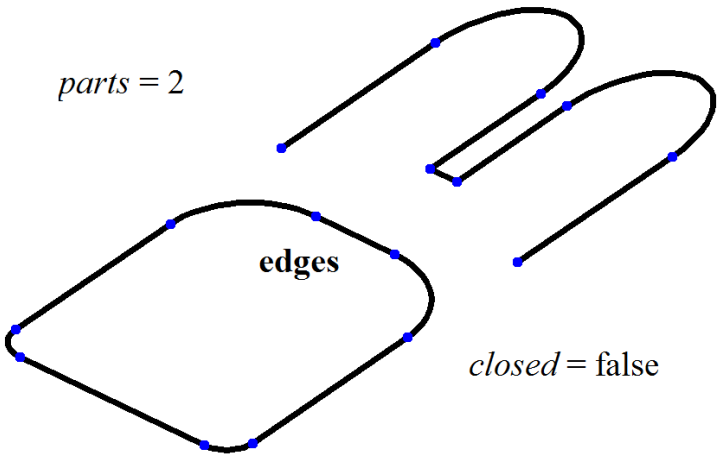


Рис. О.8.3.3.

На рис О.8.3.4 приведен замкнутый проволочный каркас, состоящий из двух связных множеств рёбер.

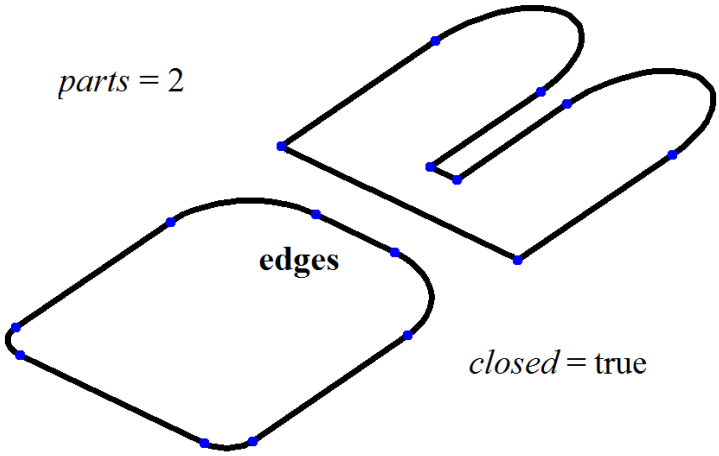


Рис. О.8.3.4.

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.

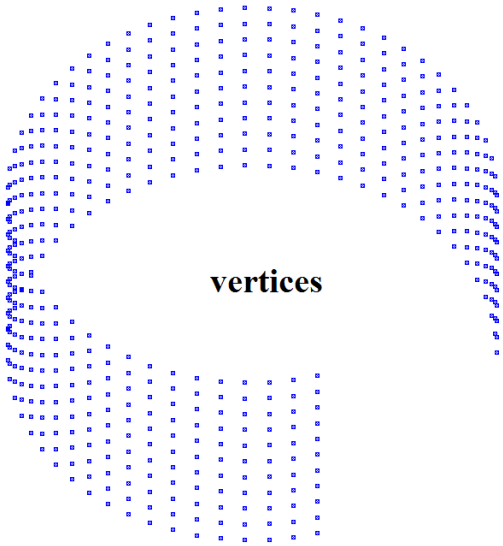


Рис. О.8.4.1.

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 RPArray<MbGrid>**grids**, множеством полигонов RPArray<MbPolygon3D>**wires**, множеством апексов RPArray<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, например, с помощью STL конвертера.

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.

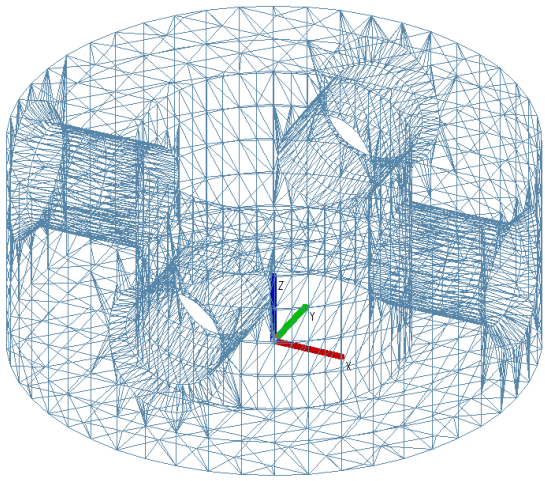


Рис. О.8.5.1.

Точечное изображение полигонального объекта приведено на рис. О.8.5.2.

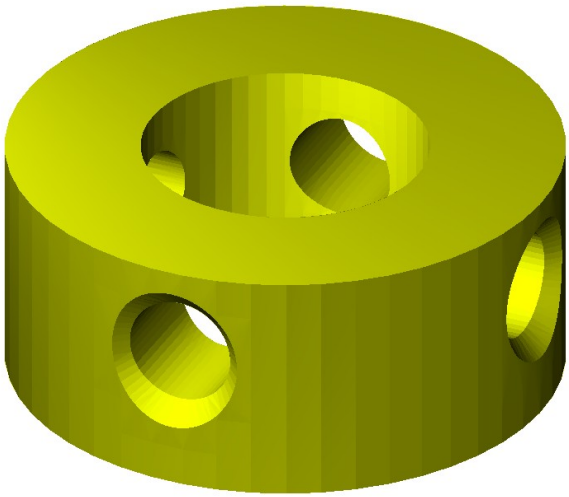


Рис. О.8.5.2.

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.

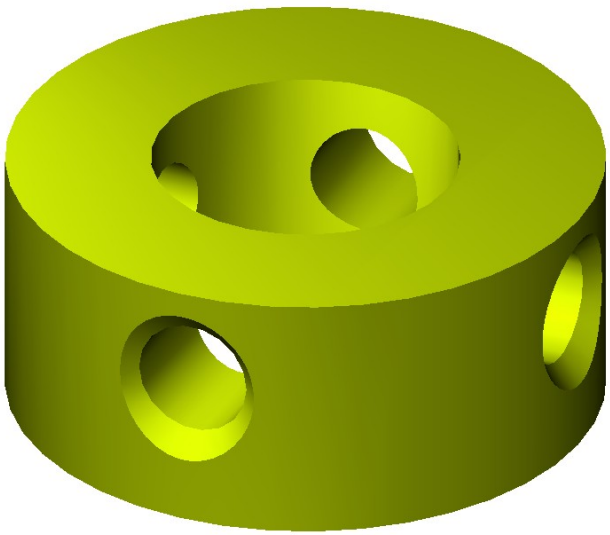


Рис. О.8.5.3.

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 MbInctance

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.

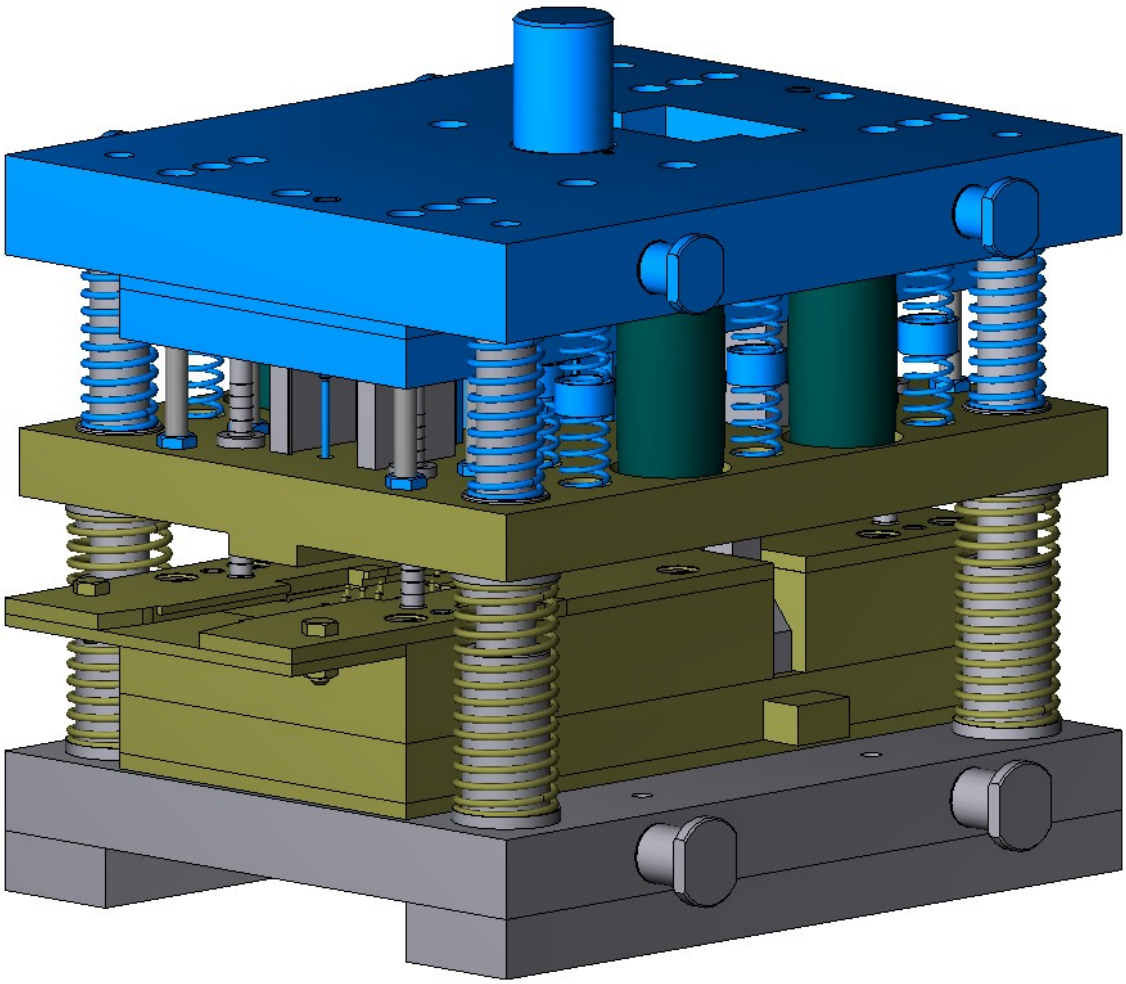


Рис. О.8.7.1.

Сборочная единица может содержать другие сборочные единицы в качестве своих объектов, то есть может иметь древовидную структуру.

О.8.8. Вставка трёхмерного объекта MbSpaceInctance

Класс MbSpaceInstance объявлен в файле space_instanse.h.

Вставка трёхмерного объекта MbSpaceInstance 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.

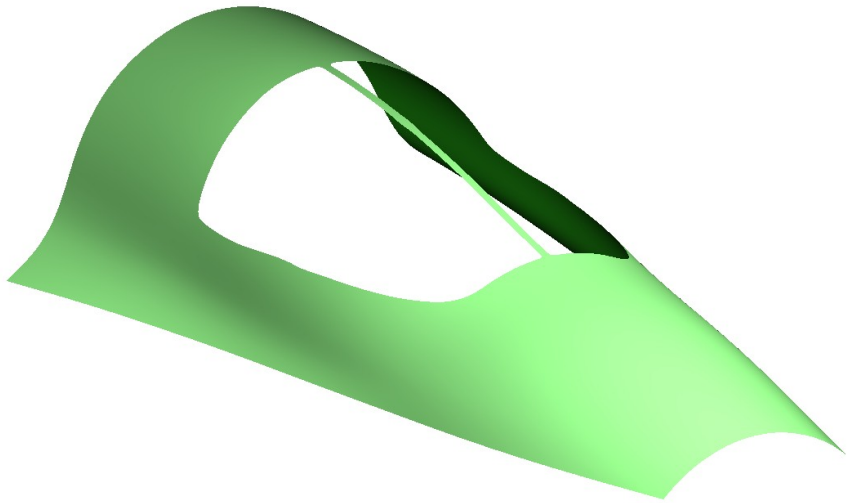


Рис. О.8.8.1.

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 MbPlaneInctance

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.

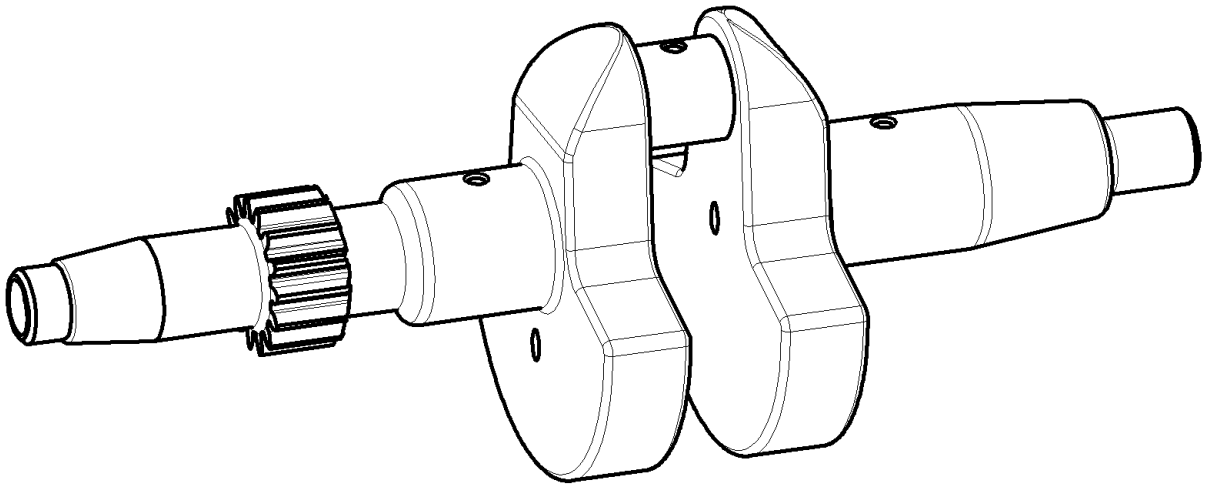


Рис. О.8.9.1.

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.

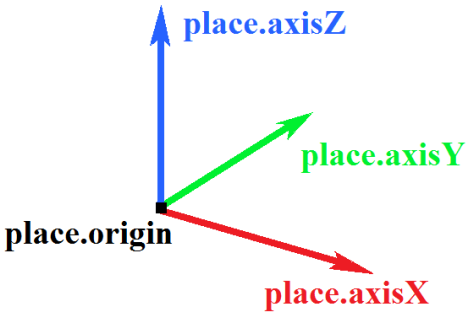


Рис. О.8.10.1.

Часть R. РАСЧЕТЫ ГЕОМЕТРИЧЕСКОГО ЯДРА C3D

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.

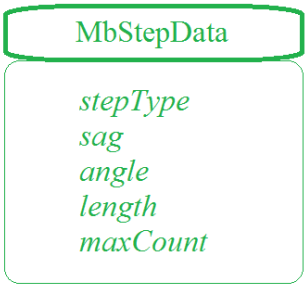


Рис. R.1.1.1.

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 густотой 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 MbeStepTypy, 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.

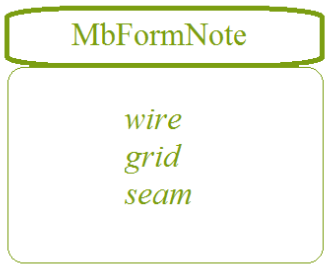


Рис. R.1.1.2.

Структура **MbFormNote** объявлена в файле mb_data.h. Структура **MbFormNote** содержит следующие данные:
bool *wire* – флаг построения полигонного объекта,
bool *grid* – ф лаг построения полигонального объекта,
bool *seam* – флаг не игнорировать шовные ребра.

Структура **MbFormNote** определяет способ построения полигонального объекта: при *wire*==true полигональный объект заполняется ломаными линиями, при *grid*==true полигональный объект заполняется триангуляциям. 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 рёбра 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 моделиMbItem*, 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.

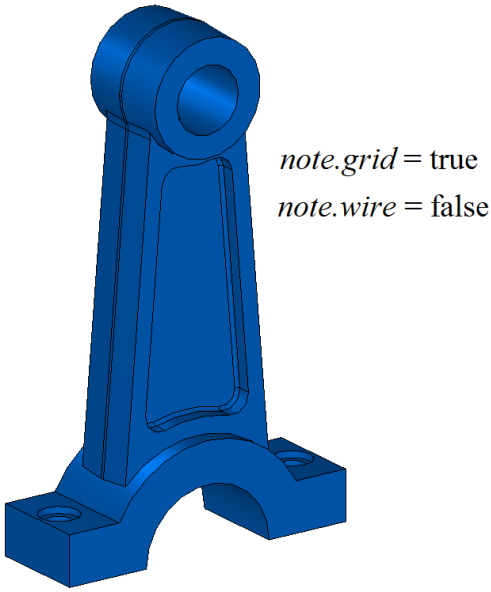


Рис. R.1.2.1.

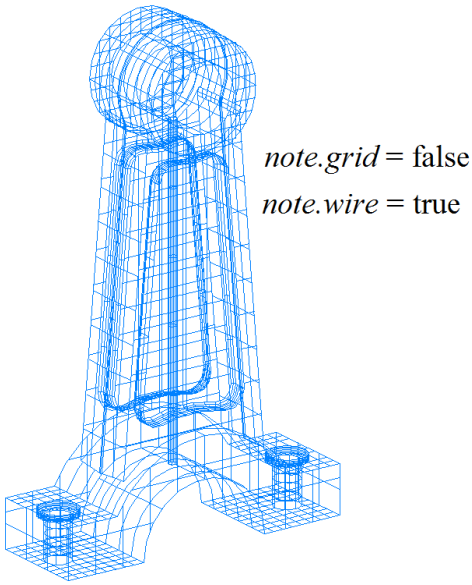


Рис. R.1.2.2.

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.

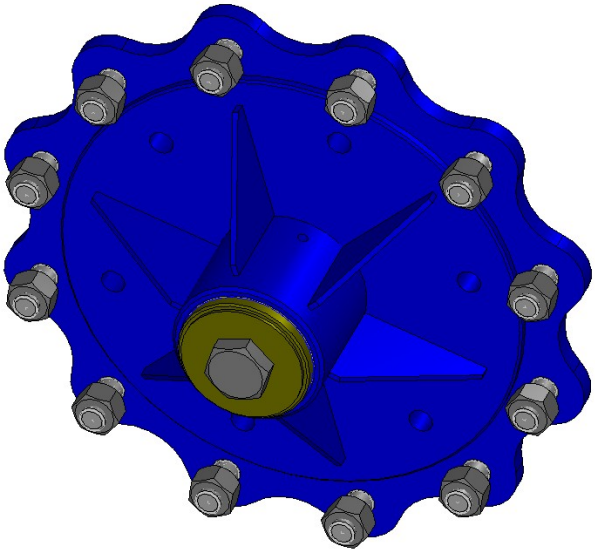


Рис. R.1.2.3.

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.

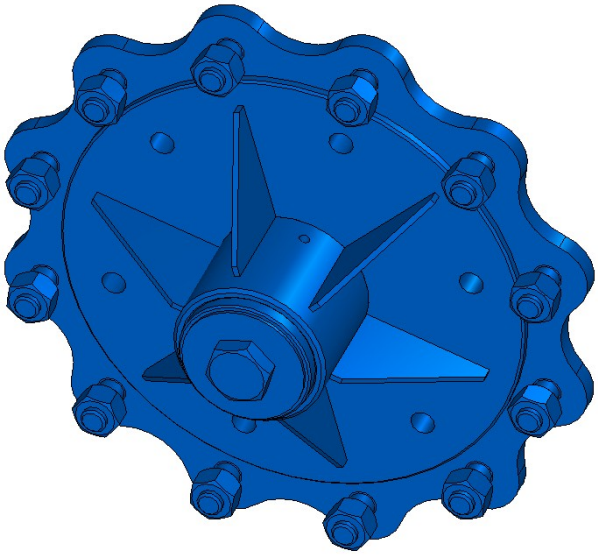


Рис. R.1.3.1.

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 O.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.

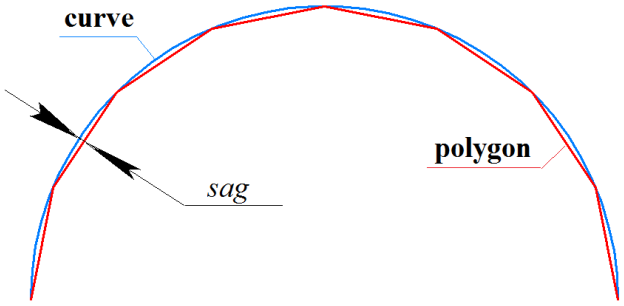


Рис. R.1.4.1.

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.

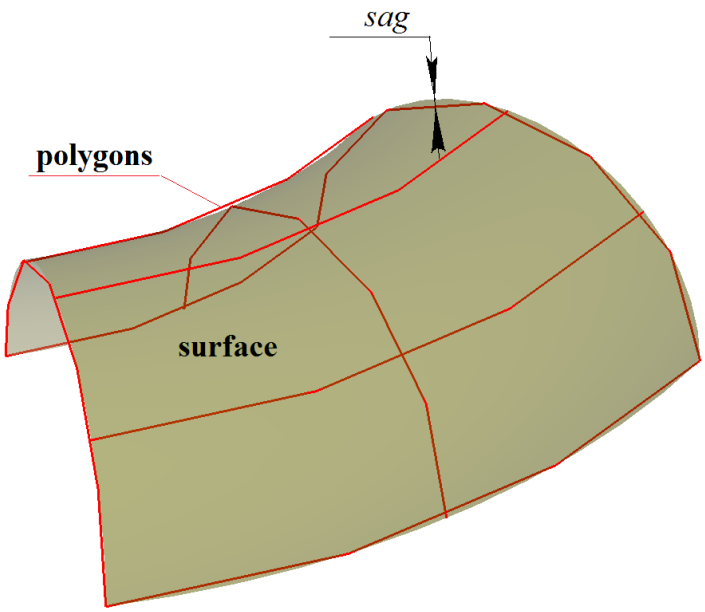


Рис. R.1.4.2.

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.

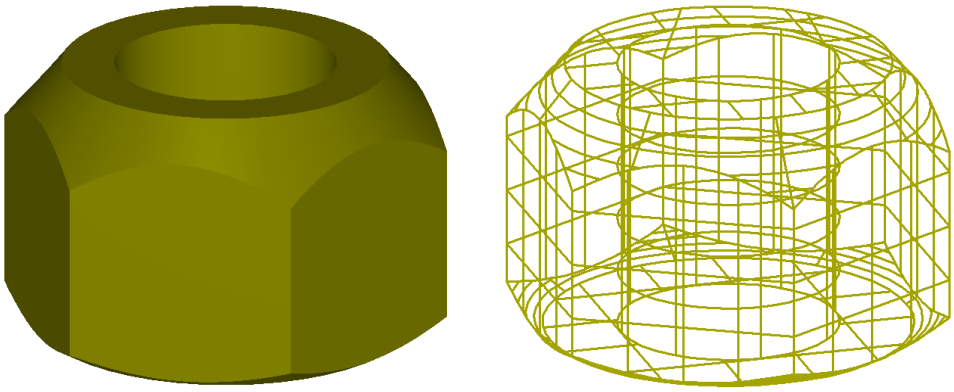


Рис. R.1.4.3.

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*.

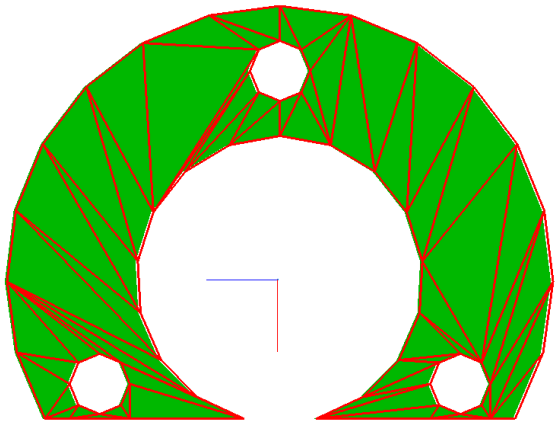


Рис. R.1.5.1.

На рис. R.1.5.2 приведена триангуляция криволинейной грани с маской *ist_SpaceStep*.

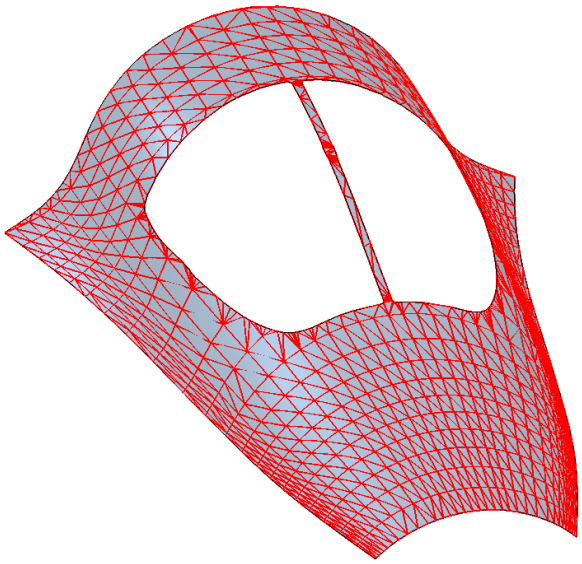


Рис. R.1.5.2.

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*,
RPArray<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 RPArray<MbSolid> & **solids**,
const MbStepData & *stepData*,
RPArray<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.

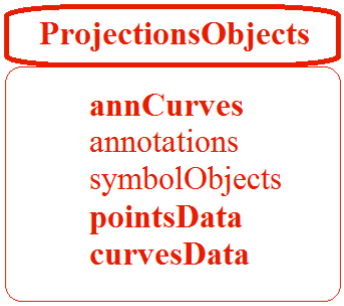


Рис. R.2.1.2.

Класс **MbProjectionsObjects** объявлен в файле map_create.h. Класс **MbProjectionsObjects** содержит следующие данные:

TPointer<PArray<MbAnnCurves> > **annCurves** – аннотационные кривые,
TPointer<RPArray<MbSimbolthThreadView> > annotations – вспомогательные объекты,
TPointer<RPArray<MbSymbol> > symbolObjects – условные обозначения,
TPointer<RPArray<MbSpacePoints> > **pointsData** – точки,
TPointer<RPArray<MbSpaceCurves> > **curvesData** – кривые.

Для передачи результатов построения плоской проекции объекта используется структура **MbVEFVestiges**, которая приведена на рис. R.2.1.3.

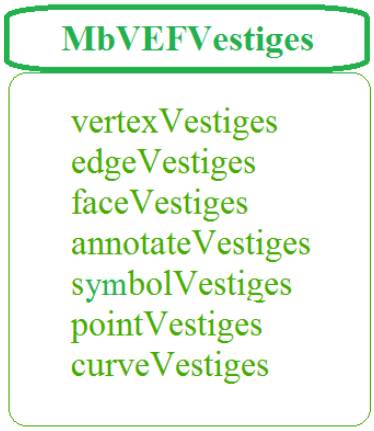


Рис. R.2.1.3.

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 RPArray<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.

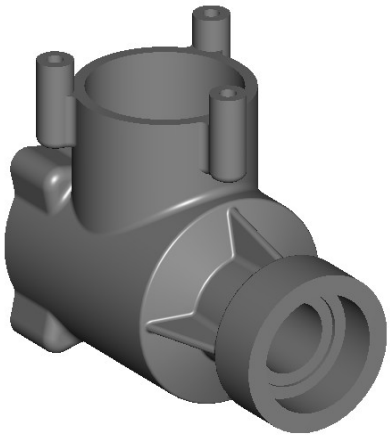


Рис. R.2.2.1.

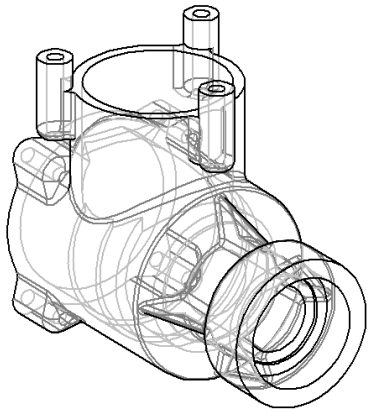


Рис. R.2.2.2.

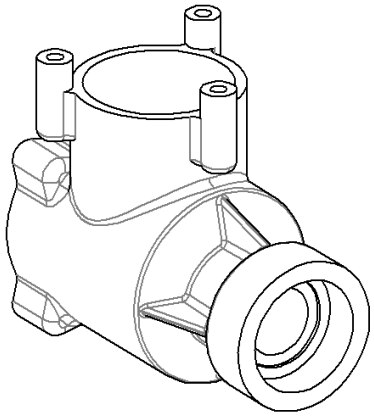


Рис. R.2.2.3.

Последний параметр рассматриваемого метода используется для поддержки предыдущих версий построения.

Метод
void
VisualLinesMapping (const MbPlacement3D & **place**,
double *znear*,
const RPArray<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 RPArray<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 RPArray<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*,
RPArray<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:
RPArray<**AssemblyMIAttire**> **assemblies** – множество сборочных единиц следующего уровня,
RPArray<**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 нормалями соседних треугольников и четырёхугольников триангуляции поверхности. Параметр *deviateAngle* управляет точностью расчёта и от его значения зависит время расчета. Параметр *deviateAngle* должен находиться в пределах 0.01<=*deviateAngle*<=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 RPArray<MbSolid> & **solids**,
const SArray<double> & *densities*,
const SArray<MbMatrix3D> & **matricies**,
const PArray<InertiaProperties> & **mpSolids**,
double *deviateAngle*,
InertiaProperties & **properties**,
IfProgressIndicator * progress = 0)

вычисляет площадь поверхности, объем, массу, центр масс и моменты инерции множества тела.

Входными параметрами метода являются:

solids – множество тел,

densities – множество плотностей или удельных масс на единицу площади,

matricies – 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 расчеты.

Метод не возвращает значения.

Метод объявлен в файле mip_solid_mass_inertia.h.

Количество элементов в множествах *densities*, **matricies**, **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 **matricies** 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 нормалями соседних треугольников и четырёхугольников триангуляции поверхности. Параметр *deviateAngle* управляет точностью расчёта и от его значения зависит время расчета. Параметр *deviateAngle* должен находиться в пределах 0.01<=*deviateAngle*<=0.35 радиан. Следует заметить, что при малых значениях *deviateAngle* для сложных моделей требуется большое время расчета.

Инерционные характеристики **properties** описаны в параграфе R.3.1. Инерционные характеристики модели. Параметр progress выдает информацию о выполнении в процессе расчета 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 расчеты.

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 насчитанные ранее инерционные характеристики и параметры управления ими. При наличии у элемента насчитанных ранее характеристик последние будут использоваться в общей сумме, что сократит общее время расчета.

Параметр *deviateAngle* управляет точностью расчёта и от его значения зависит время расчета. Параметр *deviateAngle* определяет максимально допустимый угол между нормалями соседних треугольников и четырёхугольников триангуляции поверхности. Параметр *deviateAngle* должен находиться в пределах 0.01<=*deviateAngle*<=0.35 радиан. Следует заметить, что при малых значениях *deviateAngle* для сложных моделей требуется большое время расчета.

Инерционные характеристики **properties** описаны в параграфе R.3.1. Инерционные характеристики модели. Параметр progress выдает информацию о выполнении в процессе расчета и может использоваться для остановки вычислений.

R.3.5. Вычисление площади поверхности

Метод
double
CalculateArea (const RPArray<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<=*deviateAngle*<=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 нормалями 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<=*deviateAngle*<=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<=*deviateAngle*<=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<=*deviateAngle*<=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<=*deviateAngle*<=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<=*deviateAngle*<=0.35 радиан.