МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук |  |  |  | А. В. Толстиков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ |
| РАЗРАБОТКА ИЭТР НА БАЗЕ ГРАФИЧЕСКОГО ЯДРА C3D |
| по дисциплине: ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 181-326 |  |  |  | А. М. Фролов |
| СТУДЕНТ ГР. № | 181-326 |  |  |  | Р. А. Волобуев |
| СТУДЕНТ ГР. № | 181-326 |  |  |  | А. В. Серяков |
| СТУДЕНТ ГР. № | 181-325 |  |  |  | М. Б. Новохатский |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Москва 2020

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 2

ЦЕЛЬ ПРОЕКТА 3

РЕЗУЛЬТАТЫ ПРОЕКТА (СПИСОК ПРИМЕРОВ)

1. Пример №1 4
2. Пример №2 7
3. Пример №2.1 11
4. Пример №3 12
5. Пример №4 15
6. Пример №5 18
7. Пример №6 20
8. Пример №7 23

ЗАКЛЮЧЕНИЕ 26

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc516226223) 27

# ВВЕДЕНИЕ

# Геометрическое ядро – это сложное программное обеспечение, включающее в себя множество функция и алгоритмов. При интеграции ядра в САПР роль управления ядром берет на себя интерфейс САПР. В данном случае не требуется доскональных знаний всех особенностей геометрического ядра, интерфейс позволяет полностью сконцентрироваться непосредственно на моделировании. В случае же, когда программист занимается разработкой или модифицирование геометрического моделировщика, требуются знания архитектуры, API и алгоритмов конкретного ядра. Для успешной работы создатели ядра должны иметь возможность предоставить программисту набор примеров, раскрывающих возможности и базовые алгоритмы геометрического ядра.

# ЦЕЛЬ ПРОЕКТА

Целью данного проекта является создание мини-программ, демонстрирующих основные принципы работы геометрического ядра C3D. Каждая программа реализует несколько базовых алгоритмов ядра. Также каждая программа дополняется аннотацией, описывающей принцип работы программы.

Программы в сумме своей описывают следующие возможности ядра:

1. Создание в пространстве точек, кривых, плоскостей;
2. Создание замкнутых контуров (эскизов);
3. Создание поверхностей путем выдавливания, вращения контура;
4. Создание таких элементов, как сопряжения, фаски.

Каждая аннотация выполняет следующие задачи:

1. Демонстрирует готовый код программы;
2. Описывает принцип работы программы;
3. Описывает те функции и алгоритмы, которые использовались в программе.

**РЕЗУЛЬТАТЫ ПРОЕКТА**

**ПРИМЕР №1**

Задача: реализация 2D эскиза. Создание контура с использованием отрезков, дуг, скругления и фаски.

**Поэтапное построение эскиза.**

1. Создаем локальную СК:  
   MbPlacement3Dpl;
2. Затем создаем массив точек, вершин ломаной, и объявляем их.

Создаем массив точек:

SArray<MbCartPoint>arrPnts(19);(возможно, тут надо описать, какое слово что значит)

Добавляем в массив точки:

arrPnts.Add(MbCartPoint(30, 40));

arrPnts.Add(MbCartPoint(30, 20));

arrPnts.Add(MbCartPoint(0, 20));

arrPnts.Add(MbCartPoint(0, 0));

arrPnts.Add(MbCartPoint(100, 0));

arrPnts.Add(MbCartPoint(100, 20));

arrPnts.Add(MbCartPoint(70, 20));

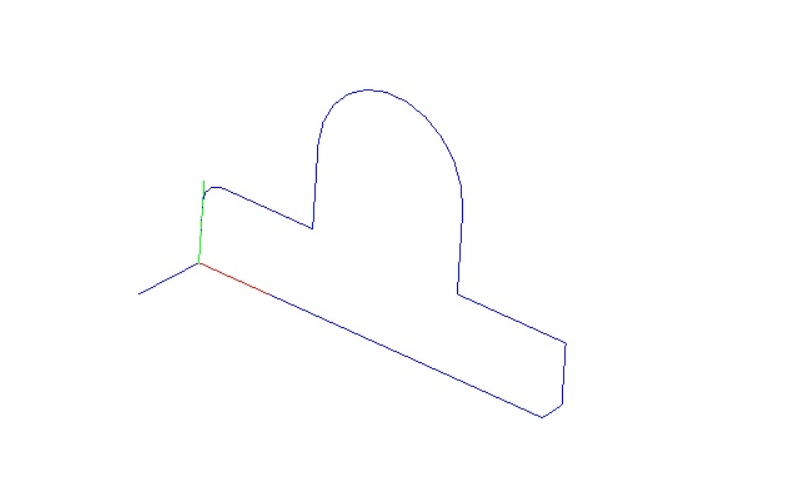
arrPnts.Add(MbCartPoint(70, 40));

1. Создаем ломаную:

MbPolyline\* pPolyline = newMbPolyline(arrPnts, false/\* Флаг незамкнутой линии \*/);

1. Создаем фаску:  
   ChamferPolyContour(pPolyline, 5,5, false, false, arrPnts[4]);
2. Создаем контур:  
   MbContour\* pContour = newMbContour(\*pPolyline, true);
3. Создаем скругление:  
   FilletPolyContour(pPolyline, 5, false, arrPnts[2], pContour);
4. Создаем дугу (Арку):  
   MbCartPointarcCenter(50, 40);   
   const double RADIUS = 20;  
   MbArc\* pArc = new MbArc(arcCenter, RADIUS, arrPnts[7], arrPnts[0], 1 /\*initSense\*/);
5. Добавляем арку в контур:  
   pContour->AddSegment(pArc);
6. Добавляем контур в сцену:  
   viewManager->AddObject(Style(1, RGB(0, 0, 255)), pContour, &pl);
7. Уменьшаем счетчик ссылок динамически созданных объектов ядра:  
   ::DeleteItem(pPolyline);  
   ::DeleteItem(pArc);  
   ::DeleteItem(pContour);

**Результат построения.**



**Код программы.**

void MakeUserCommand0()

{

MbPlacement 3D pl; // Локальная СК (по умолчанию совпадает с мировой СК)

// Вершины ломаной

// Создание массива точек

SArray<MbCartPoint>arrPnts(19);

arrPnts.Add(MbCartPoint(30, 40));

arrPnts.Add(MbCartPoint(30, 20));

arrPnts.Add(MbCartPoint(0, 20));

arrPnts.Add(MbCartPoint(0, 0));

arrPnts.Add(MbCartPoint(100, 0));

arrPnts.Add(MbCartPoint(100, 20));

arrPnts.Add(MbCartPoint(70, 20));

arrPnts.Add(MbCartPoint(70, 40));

// ЛоманаялиниясвершинамиarrPnts

MbPolyline\* pPolyline = new MbPolyline(arrPnts, false/\* Флагнезамкнутойлинии \*/);

ChamferPolyContour(pPolyline, 5,5, false, false, arrPnts[4]);

MbContour\* pContour = new MbContour(\*pPolyline, true);

FilletPolyContour(pPolyline, 5, false, arrPnts[2], pContour);

// Дуга окружности для замыкания ломаной.

// При построении указывается центр, радиус, начальная и конечная точки и

// направление обхода дуги между этими точками (значение initSense>0 соответствует

// обходу против часовой стрелки, а initSense<0 – по часовой стрелке).

MbCartPointarc Center(50, 40);

Constdouble RADIUS = 20;

MbArc\* pArc = new MbArc(arcCenter, RADIUS, arrPnts[7], arrPnts[0], 1 /\*initSense\*/);

// Контуриздвухсегментов

pContour->AddSegment(pArc)

// Отображениеконтура

if (pContour)

viewManager->AddObject(Style(1, RGB(0, 0, 255)), pContour, &pl);

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem(pPolyline);

::DeleteItem(pArc);

::DeleteItem(pContour);

}

**ПРИМЕР №2**

Задача: построение эскиза и выдавливание под углом (Усеченная пирамида).

**Поэтапное построение эскиза.**

1. Создаем функцию CreateSketch:

voidCreateSketch(RPArray<MbContour>&\_arrContours)

{

...

}

1. Затем описываем CreateSketch (Все функции, которые мы применили объяснены в комментариях).

// Создание массива точек квадрата, к которому в дальнейшем добавятся скругления.

// Размер массива - 8 точек для учета точек четырех сегментов скруглений.

SArray<MbCartPoint>arrPnts(8);

arrPnts.Add(MbCartPoint(0, 0));

arrPnts.Add(MbCartPoint(50, 0));

arrPnts.Add(MbCartPoint(50, 50));

arrPnts.Add(MbCartPoint(0, 50));

// Построение единой ломаной внешнего контура по точкам

MbPolyline\* pPolyline = newMbPolyline(arrPnts, true);

MbContour\* pContourPolyline = NULL;

// ЗаданиескруглениясиспользованиемфункцииFilletPolyContour

::FilletPolyContour(pPolyline, 5, false, arrPnts[4], pContourPolyline);

// Задание индексов точек, в которых будет задаваться скругление с учетом

// добавления новой точки при скруглении с использованием функции FilletTwoSegments

ptrdiff\_t idxSideRight1 = 0;

ptrdiff\_t idxSideRight2 = 2;

ptrdiff\_t idxSideRight3 = 4;

// Добавлениескруглений

pContourPolyline->FilletTwoSegments(idxSideRight1, 5);

pContourPolyline->FilletTwoSegments(idxSideRight2, 5);

pContourPolyline->FilletTwoSegments(idxSideRight3, 5);

\_arrContours.push\_back(pContourPolyline);

1. Переходим к описанию основной функции.

void MakeUserCommand1()

{

...

}

1. Создаем локальную систему координат.

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3Dpl;

1. Далее, мы вызываем функцию, отвечающую за создание эскиза.

// Создание образующей для тела выдавливания

RPArray<MbContour>arrContours;

CreateSketch(arrContours);

1. После, мы отображаем данный эскиз.

// Отображение образующей (в плоскости XY глобальной СК)

for (inti = 0; i<arrContours.size(); i++)

viewManager->AddObject(Style(1, LIGHTRED), arrContours[i], &pl);

1. Ниже будет представлен код, отвечающий за построение твердого тела с помощью выдавливания.

// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ

// Образующая размещается на плоскости XY глобальной СК.

// Важное замечание: объект-плоскость должен создаваться динамически,

// поскольку он продолжает использоваться в объекте-твердом теле после

// выхода из данной функции.

MbPlane\* pPlaneXY = newMbPlane(MbCartPoint3D(0, 0, 0),

MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

// Объект, хранящий параметры образующей

MbSweptDatasweptData(\*pPlaneXY, arrContours);

// Направляющий вектор для операции выдавливания

MbVector3Ddir(0, 0, -1);

// Параметры операции выдавливания, задающие свойства тела для построения в прямом

// и в обратном направлении вдоль (глубина выдавливания и уклон).

constdouble HEIGHT\_FORWARD = 60.0, HEIGHT\_BACKWARD = 0.0;

constdouble ANGLE\_FORWARD\_DEGREE = 15.0;

ExtrusionValuesextrusionParams(HEIGHT\_FORWARD, HEIGHT\_BACKWARD);

// Указание уклона для выдавливания в прямом направлении

extrusionParams.side1.rake = ANGLE\_FORWARD\_DEGREE \* M\_PI / 180.0;

// Именователи элементов модели твердого тела и контуров образующей

MbSNameMakeroperNames(1, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker>cNames(0, 1, false);

// Вызов функции-утилиты для построения твердого тела выдавливания

MbSolid\* pSolid = NULL;

MbResultType res = ::ExtrusionSolid(sweptData, dir, NULL, NULL, false,

extrusionParams, operNames, cNames, pSolid);

// Отображениепостроенноготела

if (res == rt\_Success)

{

// Смещение тела по оси Y, чтобы при отображении оно не накладывалось на образующую

pSolid->Move(MbVector3D(0, 0, 0));

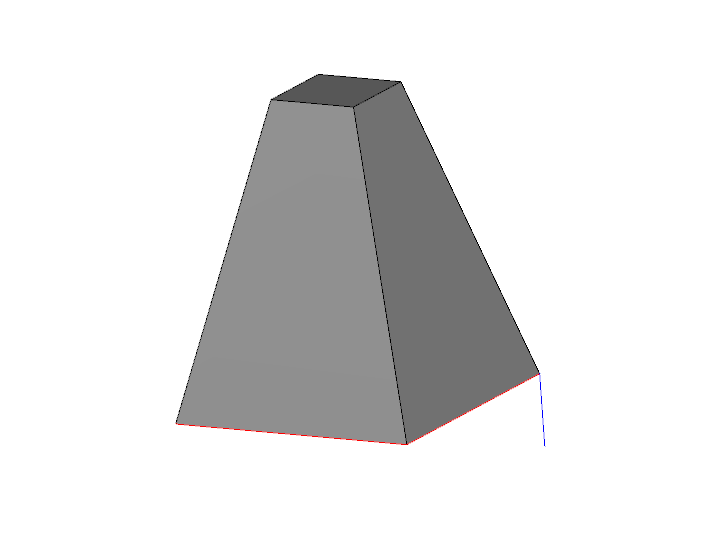
viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

}

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem(pSolid);

**Результат построения**



**Код программы**

voidCreateSketch(RPArray<MbContour>&\_arrContours)

{

// Создание массива точек квадрата, к которому в дальнейшем добавятся скругления.

// Размер массива - 8 точек для учета точек четырех сегментов скруглений.

SArray<MbCartPoint>arrPnts(8);

arrPnts.Add(MbCartPoint(0, 0));

arrPnts.Add(MbCartPoint(50, 0));

arrPnts.Add(MbCartPoint(50, 50));

arrPnts.Add(MbCartPoint(0, 50));

// Построение единой ломаной внешнего контура по точкам

MbPolyline\* pPolyline = newMbPolyline(arrPnts, true);

MbContour\* pContourPolyline = NULL;

// ЗаданиескруглениясиспользованиемфункцииFilletPolyContour

::FilletPolyContour(pPolyline, 5, false, arrPnts[4], pContourPolyline);

// Задание индексов точек, в которых будет задаваться скругление с учетом

// добавления новой точки при скруглении с использованием функции FilletTwoSegments

ptrdiff\_t idxSideRight1 = 0;

ptrdiff\_t idxSideRight2 = 2;

ptrdiff\_t idxSideRight3 = 4;

// Добавлениескруглений

pContourPolyline->FilletTwoSegments(idxSideRight1, 5);

pContourPolyline->FilletTwoSegments(idxSideRight2, 5);

pContourPolyline->FilletTwoSegments(idxSideRight3, 5);

\_arrContours.push\_back(pContourPolyline);

}

void MakeUserCommand1()

{

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3Dpl;

// Создание образующей для тела выдавливания

RPArray<MbContour>arrContours;

CreateSketch(arrContours);

// Отображение образующей (в плоскости XY глобальной СК)

for (inti = 0; i<arrContours.size(); i++)

viewManager->AddObject(Style(1, LIGHTRED), arrContours[i], &pl);

// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ

// Образующая размещается на плоскости XY глобальной СК.

// Важное замечание: объект-плоскость должен создаваться динамически,

// поскольку он продолжает использоваться в объекте-твердом теле после

// выхода из данной функции.

MbPlane\* pPlaneXY = newMbPlane(MbCartPoint3D(0, 0, 0),

MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

// Объект, хранящий параметры образующей

MbSweptDatasweptData(\*pPlaneXY, arrContours);

// Направляющий вектор для операции выдавливания

MbVector3Ddir(0, 0, -1);

// Параметры операции выдавливания, задающие свойства тела для построения в прямом

// и в обратном направлении вдоль (глубина выдавливания и уклон).

constdouble HEIGHT\_FORWARD = 60.0, HEIGHT\_BACKWARD = 0.0;

constdouble ANGLE\_FORWARD\_DEGREE = 15.0;

ExtrusionValuesextrusionParams(HEIGHT\_FORWARD, HEIGHT\_BACKWARD);

// Указание уклона для выдавливания в прямом направлении

extrusionParams.side1.rake = ANGLE\_FORWARD\_DEGREE \* M\_PI / 180.0;

// Именователи элементов модели твердого тела и контуров образующей

MbSNameMakeroperNames(1, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker>cNames(0, 1, false);

// Вызов функции-утилиты для построения твердого тела выдавливания

MbSolid\* pSolid = NULL;

MbResultType res = ::ExtrusionSolid(sweptData, dir, NULL, NULL, false,

extrusionParams, operNames, cNames, pSolid);

// Отображениепостроенноготела

if (res == rt\_Success)

{

// Смещение тела по оси Y, чтобы при отображении оно не накладывалось на образующую

pSolid->Move(MbVector3D(0, 0, 0));

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

}

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem(pSolid);

}

**ПРИМЕР №2.1**

Задача: построение эскиза в пространстве (Усеченная пирамида).

**Поэтапное построение эскиза.**

1. Первое что мы делам, создаем локальную систему координат.

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3Dpl;

1. После мы создаем массив из 8 точек.

SArray<MbCartPoint3D> arrPnts(8);

1. После мы заносим в массив координаты точек

arrPnts.Add(MbCartPoint3D(0, 0, 0));

arrPnts.Add(MbCartPoint3D(0, 50, 0));

arrPnts.Add(MbCartPoint3D(50, 50, 0));

arrPnts.Add(MbCartPoint3D(50, 0, 0));

arrPnts.Add(MbCartPoint3D(15, 15, 40));

arrPnts.Add(MbCartPoint3D(15, 35, 40));

arrPnts.Add(MbCartPoint3D(35, 35, 40));

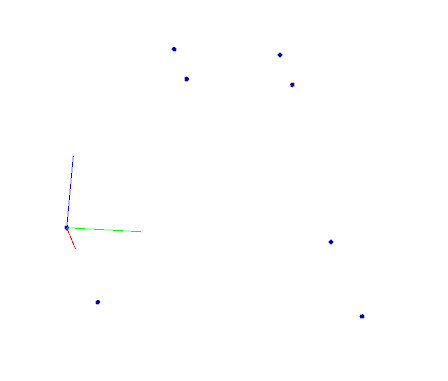
arrPnts.Add(MbCartPoint3D(35, 15, 40));

1. И в конце, мы задаем отображение наших точек синим цветом.

//Выделения массива точек синим цветом

viewManager->AddObject(Style(1, BLUE), new MbPointFrame(arrPnts));

**Результат работы**



**Код программы**

void MakeUserCommand2()

{

MbPlacement3D pl; // Локальная СК (по умолчанию совпадает с мировой СК)

//Массив из точек

SArray<MbCartPoint3D> arrPnts(8);

arrPnts.Add(MbCartPoint3D(0, 0, 0));

arrPnts.Add(MbCartPoint3D(0, 50, 0));

arrPnts.Add(MbCartPoint3D(50, 50, 0));

arrPnts.Add(MbCartPoint3D(50, 0, 0));

arrPnts.Add(MbCartPoint3D(15, 15, 40));

arrPnts.Add(MbCartPoint3D(15, 35, 40));

arrPnts.Add(MbCartPoint3D(35, 35, 40));

arrPnts.Add(MbCartPoint3D(35, 15, 40));

//Выделения массива точек синим цветом

viewManager->AddObject(Style(1, BLUE), new MbPointFrame(arrPnts));

}

**ПРИМЕР №3**

Задача: построение параллелепипеда с отверстием при помощи инструмента выдавливания со смещением эскиза.

**Поэтапное построение эскиза.**

1. Создадим отдельную функцию для создания эскиза.  
   Создадим контур квадрата по массиву точек.

SArray<MbCartPoint> arrPnts(8);  
 arrPnts.Add(MbCartPoint(0, 0));  
 arrPnts.Add(MbCartPoint(50, 0));  
 arrPnts.Add(MbCartPoint(50, 50));  
 arrPnts.Add(MbCartPoint(0, 50));  
MbPolyline\* pPolyline = new MbPolyline(arrPnts, true);  
MbContour\* pContour = new MbContour(\*pPolyline, true);

Создадим контур круглого отверстия при помощи функции окружности.   
const MbCartPoint centerCircle(25, 25);  
const double RAD = 10;  
MbArc\* pCircle = new MbArc(centerCircle, RAD);  
MbContour\* pContourCircle = new MbContour([\*pCircle](https://vk.com/pcircle), true);

1. Создадим локальную систему координат.

MbPlacement3D pl;

1. Создадим контур для образующей.

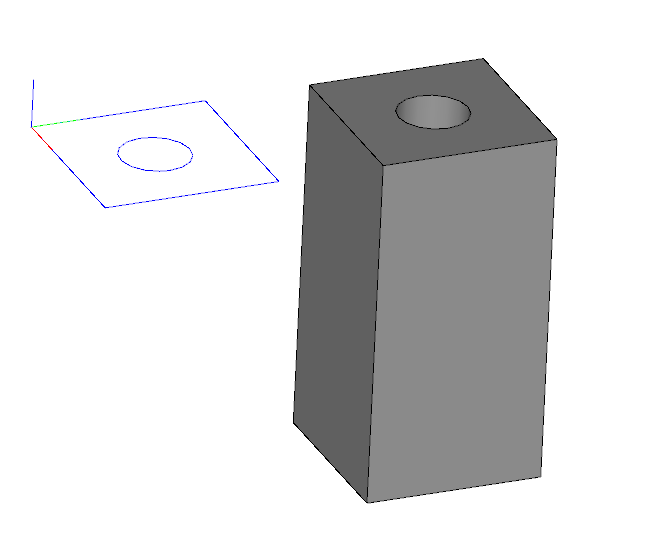
RPArray<MbContour> arrContours;  
CreateSketch(arrContours);  
  
Сделаем отображение данного контура.  
for (int i = 0; i < arrContours.size(); i++)  
viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours[i], &pl);

1. Создадим плоскость, в которой будет располагаться образующая.

MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),  
MbCartPoint3D(0, 1, 0));

1. Создадим объект для хранения параметров образующей.  
   MbSweptData sweptData(\*pPlaneXY, arrContours);
2. Определим параметры для операции выдавливания.  
   Создадим направляющий вектор.  
   MbVector3D dir(0, 0, -1);  
     
   Зададим параметры операции выдавливания.  
   const double HEIGHT\_FORWARD = 100.0, HEIGHT\_BACKWARD = 0.0;  
   ExtrusionValues extrusionParam(HEIGHT\_FORWARD, HEIGHT\_BACKWARD);
3. Создадим объект для именования элементов модели твердого тела.  
   MbSNameMaker operNames(ct\_CurveExtrusionSolid, MbSNameMaker::i\_SideNone, 0);  
   PArray<MbSNameMaker> cNames(0, 1, false);
4. Создадим твердое тело.  
   MbSolid\* pSolid = NULL;  
   MbResultType res = ::ExtrusionSolid(sweptData, dir, NULL, NULL, false,  
   extrusionParam, operNames, cNames, pSolid);
5. Отобразим модель твердого тела.  
   if (res == rt\_Success)  
    {  
    viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);  
    }
6. Уменьшение счетчиков ссылок динамически созданных объектов ядра.  
   ::DeleteItem(pSolid);  
   ::DeleteItem(pPlaneXY);  
   for (int i = 0; i < arrContours.size(); i++)  
    ::DeleteItem(arrContours[i]);

**Результат построения**



**Код программы**

void CreateSketch(RPArray<MbContour>& \_arrContours)  
{  
// Создание массива точек квадрата, к которому в дальнейшем добавятся скругления.  
// Размер массива - 8 точек для учета точек четырех сегментов скруглений.  
 SArray<MbCartPoint> arrPnts(8);  
 arrPnts.Add(MbCartPoint(0, 0));  
 arrPnts.Add(MbCartPoint(50, 0));  
 arrPnts.Add(MbCartPoint(50, 50));  
 arrPnts.Add(MbCartPoint(0, 50));  
// Построение единой ломаной внешнего контура по точкам  
 MbPolyline\* pPolyline = new MbPolyline(arrPnts, true);  
 MbContour\* pContour = new MbContour(\*pPolyline, true);  
// Центры и радиусы окружностей, дуги которых входят в контур  
 const MbCartPoint centerCircle(25, 25);  
 const double RAD = 10;  
 MbArc\* pCircle = new MbArc(centerCircle, RAD);  
 MbContour\* pContourCircle = new MbContour([\*pCircle](https://vk.com/pcircle), true);  
 \_arrContours.push\_back(pContour);  
 \_arrContours.push\_back(pContourCircle);  
}  
void MakeUserCommand0()  
{  
 MbPlacement3D pl; // Локальная СК (по умолчанию совпадает с мировой СК)  
// СОЗДАНИЕ КОНТУРОВ ДЛЯ ОБРАЗУЮЩЕЙ  
 RPArray<MbContour> arrContours;  
 CreateSketch(arrContours);  
  
// Отображение образующей (в плоскости XY глобальной СК)  
 for (int i = 0; i < arrContours.size(); i++)  
 viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours[i], &pl);  
// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ  
// Образующая размещается на плоскости XY глобальной СК.  
// Важное замечание: объект-плоскость должен создаваться динамически,  
// поскольку он продолжает использоваться в объекте-твердом теле после  
// выхода из данной функции.  
 MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),  
 MbCartPoint3D(0, 1, 0));  
// Объект, хранящий параметры образующей  
 MbSweptData sweptData(\*pPlaneXY, arrContours);  
// Направляющий вектор для операции выдавливания  
 MbVector3D dir(0, 0, -1);  
// Параметры операции выдавливания, задающие свойства тела для построения:  
// расстояние выдавливания в прямом и в обратном направлении вдоль  
// направляющего вектора  
 const double HEIGHT\_FORWARD = 100.0, HEIGHT\_BACKWARD = 0.0;  
 ExtrusionValues extrusionParam(HEIGHT\_FORWARD, HEIGHT\_BACKWARD);  
// Служебный объект для именования элементов модели твердого тела  
 MbSNameMaker operNames(ct\_CurveExtrusionSolid, MbSNameMaker::i\_SideNone, 0);  
 PArray<MbSNameMaker> cNames(0, 1, false);  
// Построение твердого тела выдавливания  
 MbSolid\* pSolid = NULL;  
 MbResultType res = ::ExtrusionSolid(sweptData, dir, NULL, NULL, false,  
 extrusionParam, operNames, cNames, pSolid);  
// Отображение построенного тела  
 if (res == rt\_Success)  
 {  
// Смещение тела по оси Y, чтобы при отображении оно не накладывалось на образующую  
 pSolid->Move(MbVector3D(0, 80, 0));  
 viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);  
 }  
// Уменьшение счетчиков ссылок динамически созданных объектов ядра  
 ::DeleteItem(pSolid);  
 ::DeleteItem(pPlaneXY);  
 for (int i = 0; i < arrContours.size(); i++)  
 ::DeleteItem(arrContours[i]);  
}

**ПРИМЕР №4**

Задача: построение цилиндра с помощью функции вращения

**Поэтапное построение эскиза.**

1. Создаем функцию CreateSketch:

void CreateSketch(RPArray<MbContour>& \_arrContours)

{

...

}

2. Затем описываем CreateSketch (Все функции, которые мы применили объяснены в комментариях).

// Создание массива точек квадрата, к которому в дальнейшем добавятся скругления.

// Размер массива - 8 точек для учета точек четырех сегментов скруглений.

SArray<MbCartPoint> arrPnts(4);

arrPnts.Add(MbCartPoint(0, 0));

arrPnts.Add(MbCartPoint(50, 0));

arrPnts.Add(MbCartPoint(50, 50));

arrPnts.Add(MbCartPoint(0, 50));

// Построение единой ломаной внешнего контура по точкам

MbPolyline\* pPolyline = new MbPolyline(arrPnts, true);

MbContour\* pContourPolyline = new MbContour(\*pPolyline, true);

\_arrContours.push\_back(pContourPolyline);

3. Переходим к описанию основной функции.

void MakeUserCommand0()

{

...

}

4. Первое что мы делам, создаем локальную систему координат и множитель для преобразования угловых значений.

// Множитель для преобразования угловых значений из градусов в радианы

const double DEG\_TO\_RAD = M\_PI / 180.0;

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3D pl;

5. Далее, мы вызываем функцию отвечающую за создание эскиза.

// Вызов функции для построения образующей

RPArray<MbContour> arrContours;

CreateSketch(arrContours);

// Отображение образующей (в плоскости XY глобальной СК)

for (int i = 0; i < arrContours.size(); i++)

viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours[i], &pl);

7. Далее будет представлен код по построению твердого тела с помощью функции вращения.

// Подготовка параметров для вызова функции построения тела вращения.

// sweptData - объект, в котором хранятся сведения об образующей.

MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

MbSweptData sweptData(\*pPlaneXY, arrContours);

// Объект параметров для операции построения тела вращения.

// Первые два параметра конструктора задают углы вращения в прямом и обратном

// направлении (по нормали к образующей или противоположно этой нормали).

// В данном примере вращение осуществляется на 120 градусов в прямом направлении.

// Третий параметр задает форму топологии твердого тела.

// При s = 0 строится тело с топологией сферы, при s = 1 - с топологией тора.

// Если образующая - незамкнутая кривая, и ось вращения лежит в плоскости кривой,

// то при s = 0 при построении образующая будет автоматически продлена до оси

// вращения. В данном примере различий между значениями s нет, т.к. образующая

// имеет вид замкнутого контура.

RevolutionValues revParms(360 \* DEG\_TO\_RAD, 0, 0);

// Именователи для операции построения тела вращения и для контуров образующей

MbSNameMaker operNames(1, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames(0, 1, false);

// Ось вращения для построения тела:

// ось Y мировой СК смещается на -50 единиц вдоль оси X.

MbAxis3D axis(pl.GetAxisY());

// Вызов функции-утилиты для построения твердого тела вращения

MbSolid\* pSolid = NULL;

MbResultType res = ::RevolutionSolid(sweptData, axis, revParms,

operNames, cNames, pSolid);

// Отображение построенного тела

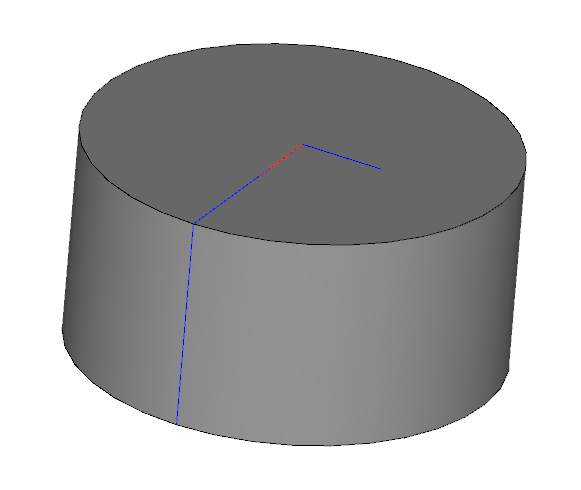
if (res == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

// Уменьшение счетчиков ссылок динамических объектов ядра

::DeleteItem(pSolid);

**Результаты построения**



**Код программы**

void CreateSketch(RPArray<MbContour>& \_arrContours)

{

// Создание массива точек квадрата, к которому в дальнейшем добавятся скругления.

// Размер массива - 8 точек для учета точек четырех сегментов скруглений.

SArray<MbCartPoint> arrPnts(4);

arrPnts.Add(MbCartPoint(0, 0));

arrPnts.Add(MbCartPoint(50, 0));

arrPnts.Add(MbCartPoint(50, 50));

arrPnts.Add(MbCartPoint(0, 50));

// Построение единой ломаной внешнего контура по точкам

MbPolyline\* pPolyline = new MbPolyline(arrPnts, true);

MbContour\* pContourPolyline = new MbContour(\*pPolyline, true);

\_arrContours.push\_back(pContourPolyline);

}

void MakeUserCommand0()

{

// Множитель для преобразования угловых значений из градусов в радианы

const double DEG\_TO\_RAD = M\_PI / 180.0;

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3D pl;

// Вызов функции для построения образующей (из примера 4.1)

RPArray<MbContour> arrContours;

CreateSketch(arrContours);

// Отображение образующей (в плоскости XY глобальной СК)

for (int i = 0; i < arrContours.size(); i++)

viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours[i], &pl);

// Подготовка параметров для вызова функции построения тела вращения.

// sweptData - объект, в котором хранятся сведения об образующей.

MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

MbSweptData sweptData(\*pPlaneXY, arrContours);

// Объект параметров для операции построения тела вращения.

// Первые два параметра конструктора задают углы вращения в прямом и обратном

// направлении (по нормали к образующей или противоположно этой нормали).

// В данном примере вращение осуществляется на 120 градусов в прямом направлении.

// Третий параметр задает форму топологии твердого тела.

// При s = 0 строится тело с топологией сферы, при s = 1 - с топологией тора.

// Если образующая - незамкнутая кривая, и ось вращения лежит в плоскости кривой,

// то при s = 0 при построении образующая будет автоматически продлена до оси

// вращения. В данном примере различий между значениями s нет, т.к. образующая

// имеет вид замкнутого контура.

RevolutionValues revParms(360 \* DEG\_TO\_RAD, 0, 0);

// Именователи для операции построения тела вращения и для контуров образующей

MbSNameMaker operNames(1, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames(0, 1, false);

// Ось вращения для построения тела:

// ось Y мировой СК смещается на -50 единиц вдоль оси X.

MbAxis3D axis(pl.GetAxisY());

// Вызов функции-утилиты для построения твердого тела вращения

MbSolid\* pSolid = NULL;

MbResultType res = ::RevolutionSolid(sweptData, axis, revParms,

operNames, cNames, pSolid);

// Отображение построенного тела

if (res == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

// Уменьшение счетчиков ссылок динамических объектов ядра

::DeleteItem(pSolid);

}

**ПРИМЕР №5**

Задача: построение простых тел.

**Поэтапное построение эскиза.**

1. Создаем с помощью функции MbSphereSurface сферу, для этого мы задаем координаты центра сферы в пространстве и радиус.

MbSphereSurface\* pSurf = new MbSphereSurface(MbCartPoint3D(50, 0, 0), 20);

1. После делаем отображение сферы в пространстве.  
   viewManager->AddObject(Style(1, LIGHTGRAY), pSurf);
2. Создаем цилиндрическую поверхность. Для этого необходимо задать центр локальной оси, после задать два вектора по оси Z и Y.

// Цилиндрическая поверхность - построение по трем точкам

const MbCartPoint3D p0(50, 0, 0); // Центр локальной СК цилиндра (центр основания)

const MbCartPoint3D p1(0, 20, 0); // Вектор p0-p1 задает ось Z локальной СК

// и высоту цилиндра

const MbCartPoint3D p2(10, 0, 0); // Вектор p0-p2 задает ось X локальной СК и

// радиус основания

1. Следующий этап вызов функции MbCylinderSurface для построения поверхности цилиндра и отображаем его в пространстве.

MbCylinderSurface\* pSurf1 = new MbCylinderSurface(p0, p1, p2);

// Отображение поверхности

viewManager->AddObject(Style(1, LIGHTGRAY), pSurf1);

1. Следующее строим коническую поверхность. На понадобится локальная СК, радиус основания конуса, высота конуса и Угол между осью Z локальной СК и боковой образующей.

// Коническая поверхность - конструктор по локальной системе координат,

// радиусу, углу и высоте.

const MbPlacement3D pl1; // Локальная СК совпадает с мировой

const double radius = 0; // Радиус в плоскости XY локальной СК

// (если 0 - строится коническая поверхность конуса,

// если >0 - строится поверхность усеченного конуса)

// Угол между осью Z локальной СК и боковой образующей

const double angle = 22.5 \* M\_PI / 180.0;

const double height = 10; // Высота конуса (вдоль оси Z)

1. И также вызываем функцию MbConeSurfaсе и делаем конус видимым в пространстве.

MbConeSurface\* pSurf2 = new MbConeSurface(pl1, radius, angle, height);

// Отображение поверхности

viewManager->AddObject(Style(1, LIGHTGRAY), pSurf2);

1. Делаем тородиальную поверхность.

// Тороидальная поверхность: конструктор по локальной СК и двум радиусам

// Ось Z локальной СК является осью вращательной симметрии торической поверхности

const MbPlacement3D pl2; // Локальная СК совпадает с мировой.

const double radius1 = 10; // Радиус центров трубки тора

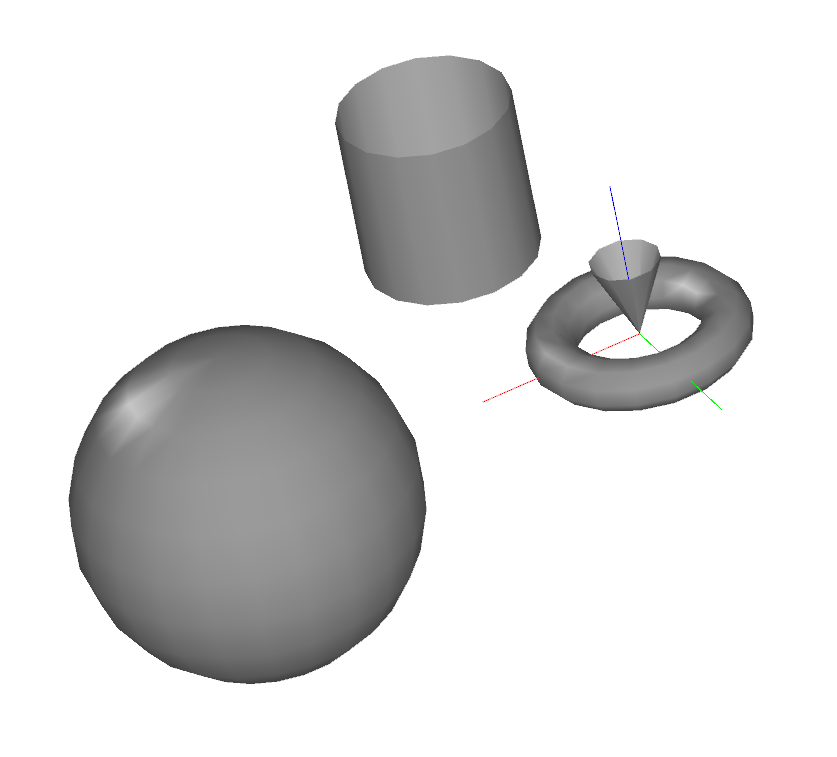
const double radius2 = 3; // Радиус трубки тора

MbTorusSurface\* pSurf3 = new MbTorusSurface(pl2, radius1, radius2);

// Отображение поверхности

viewManager->AddObject(Style(1, LIGHTGRAY), pSurf3);

**Результат построения**



**Код программы**

void MakeUserCommand6()

{

// Сферическая поверхность - построение по центру и радиусу

MbSphereSurface\* pSurf = new MbSphereSurface(MbCartPoint3D(50, 0, 0), 20);

// Отображение поверхности

viewManager->AddObject(Style(1, LIGHTGRAY), pSurf);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf);

// Цилиндрическая поверхность - построение по трем точкам

const MbCartPoint3D p0(50, 0, 0); // Центр локальной СК цилиндра (центр основания)

const MbCartPoint3D p1(0, 20, 0); // Вектор p0-p1 задает ось Z локальной СК и высоту цилиндра

const MbCartPoint3D p2(10, 0, 0); // Вектор p0-p2 задает ось X локальной СК и

// радиус основания

MbCylinderSurface\* pSurf1 = new MbCylinderSurface(p0, p1, p2);

// Отображение поверхности

viewManager->AddObject(Style(1, LIGHTGRAY), pSurf1);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf1);

// Коническая поверхность - конструктор по локальной системе координат,

// радиусу, углу и высоте.

const MbPlacement3D pl1; // Локальная СК совпадает с мировой

const double radius = 0; // Радиус в плоскости XY локальной СК

// (если 0 - строится коническая поверхность конуса, если >0 - строится поверхность усеченного конуса)

// Угол между осью Z локальной СК и боковой образующей

const double angle = 22.5 \* M\_PI / 180.0;

const double height = 10; // Высота конуса (вдоль оси Z)

MbConeSurface\* pSurf2 = new MbConeSurface(pl1, radius, angle, height);

// Отображение поверхности

viewManager->AddObject(Style(1, LIGHTGRAY), pSurf2);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf2);

// Тороидальная поверхность: конструктор по локальной СК и двум радиусам

// Ось Z локальной СК является осью вращательной симметрии торической поверхности

const MbPlacement3D pl2; // Локальная СК совпадает с мировой.

const double radius1 = 10; // Радиус центров трубки тора

const double radius2 = 3; // Радиус трубки тора

MbTorusSurface\* pSurf3 = new MbTorusSurface(pl2, radius1, radius2);

// Отображение поверхности

viewManager->AddObject(Style(1, LIGHTGRAY), pSurf3);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf3);

}

**ПРИМЕР №6**

Задача: построение крюка (лофт).

**Поэтапное построение эскиза.**

1. Создаем функцию CreateSketch.

void CreateSketch(RPArray<MbSurface>& \_arrSurfaces, RPArray<MbContour>& \_arrContours, double vert,double hor, double OFS\_X, double OFS\_Y, double OFS\_Z, double d)

{

...

}

2. Затем описываем CreateSketch (Все функции, которые мы применили объяснены в комментариях).

double l = d \* 0.4472138576667362; //константа

SArray<MbCartPoint>arrPnts(4); //создание массива точек

arrPnts.Add(MbCartPoint(-l / 2, l));

arrPnts.Add(MbCartPoint(l / 2, l));

arrPnts.Add(MbCartPoint(-l / 2, -l));

arrPnts.Add(MbCartPoint(l / 2, -l));

MbLineSegment\* pS1 = new MbLineSegment(arrPnts[0], arrPnts[1]); //создание линии по двум точкам из массива

MbContour\* pContour = new MbContour(\*pS1, true); //создание контура, состоящего из линии pS1

MbArc\* pArc1 = new MbArc(MbCartPoint(0, 0), arrPnts[1], arrPnts[3], 0); //создание арки по центру и двум точкам из массива

pContour->AddSegment(pArc1); //добавление арки в контур

MbLineSegment\* pS2 = new MbLineSegment(arrPnts[3], arrPnts[2]);

pContour->AddSegment(pS2);

MbArc\* pArc2 = new MbArc(MbCartPoint(0, 0), arrPnts[2], arrPnts[0], 0);

pContour->AddSegment(pArc2);

MbPlacement3D plSurf(MbCartPoint3D(0 + OFS\_X, 0 + OFS\_Y, 0 + OFS\_Z), //создание локальной системы координат

MbCartPoint3D(hor + OFS\_X, vert + OFS\_Y, 0 + OFS\_Z), //создание точки в трехмерном пространстве

MbCartPoint3D(0 + OFS\_X, 0 + OFS\_Y, 1 + OFS\_Z));

MbSurface\* pSurf1 = new MbPlane(plSurf); //создание поверхности в локальной системе координат

\_arrSurfaces.push\_back(pSurf1);

\_arrContours.push\_back(pContour);

3. Переходим к описанию основной функции.

void MakeUserCommand0()

{

...

}

4. Первое что мы делам, создаем набор скетчей, они будут выступать образующими .

// Получение образующей в виде массивов поверхностей и контуров на них с помощью

// вспомогательной функции.

RPArray<MbSurface> arrSurfaces;

RPArray<MbContour> arrContours;

CreateSketch(arrSurfaces, arrContours, 0, 1, 30, 55, 0, 12);

CreateSketch(arrSurfaces, arrContours, 0, 1, 30, 40, 0, 12);

CreateSketch(arrSurfaces, arrContours, 0, 1, 40, 23, 0, 15);

CreateSketch(arrSurfaces, arrContours, 1, 0, 22, 10, 0, 15);

CreateSketch(arrSurfaces, arrContours, 0, 1, 5, 25, 0, 10);

CreateSketch(arrSurfaces, arrContours, 0, 1, 5, 30, 0, 3);

5. Создаем объект для операции лофта.

// Объект с параметрами операции заметания.

LoftedValues params;

6. Создаем объекты для наименование наших тел в тестовом приложении.

// Объекты для именования элементов модели твердого тела.

MbSNameMaker names(ct\_CurveLoftedSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames(0, 1, false);

7. Создаем твердотельную модель при помощи лофта.

// Построение твердого тела заметания

MbSolid\* pSolid = NULL;

MbResultType res = ::LoftedSolid(arrSurfaces, arrContours, NULL, params, NULL, NULL, names, contourNames, pSolid);

8. Отображение построенного тела.

// Отображение построенного тела

if (res == rt\_Success)

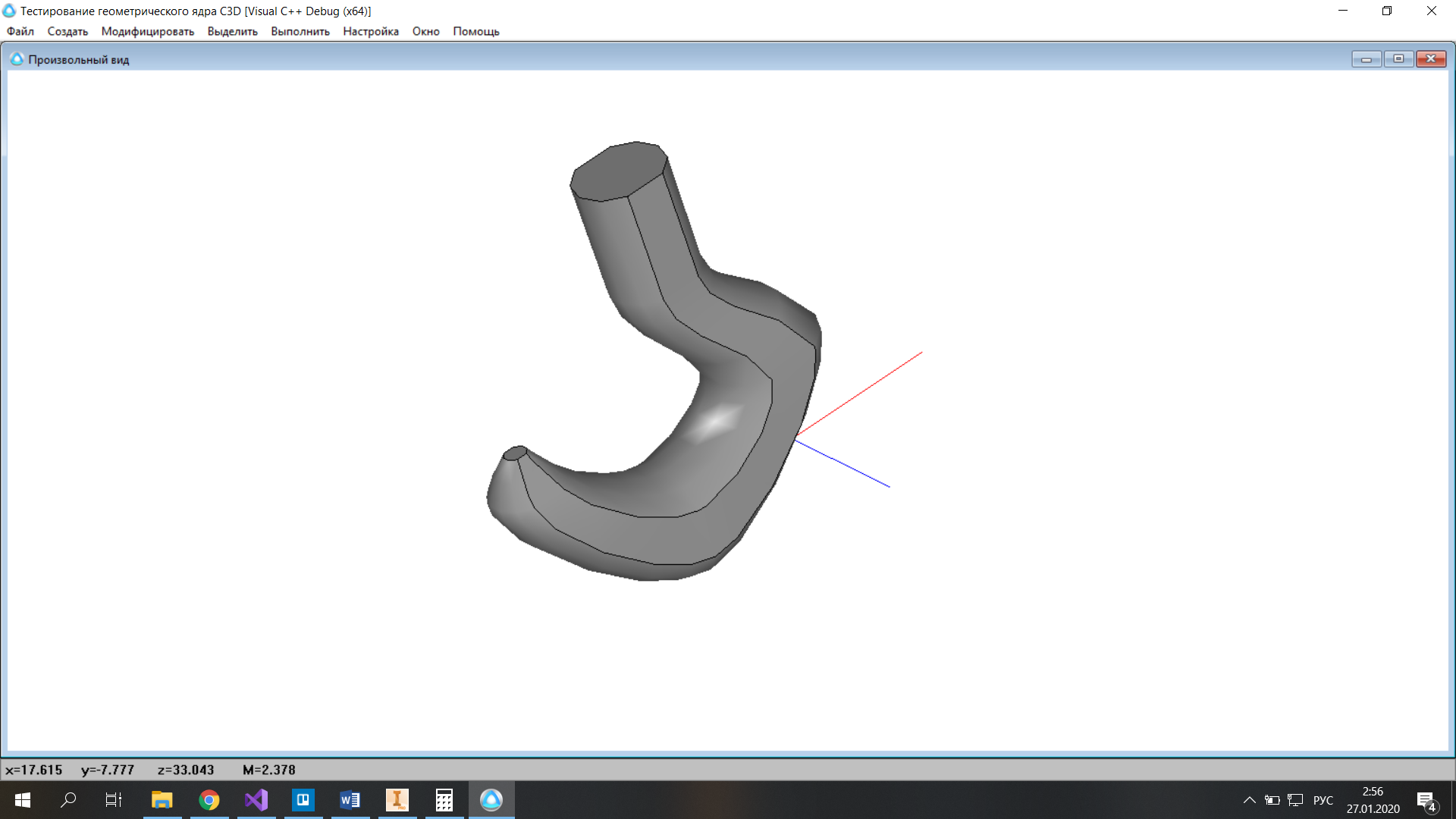
viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

9.Уменьшения счетчика ссылок.

// Уменьшение счетчика ссылок тела

::DeleteItem(pSolid);

**РЕЗУЛЬТАТ ПОСТРОЕНИЯ**



**Код программы**

void CreateSketch (RPArray<MbSurface>& \_arrSurfaces, RPArray<MbContour>& \_arrContours, double vert,double hor, double OFS\_X, double OFS\_Y, double OFS\_Z, double d)

{

double l = d \* 0.4472138576667362;

SArray<MbCartPoint>arrPnts(4);

arrPnts.Add(MbCartPoint(-l / 2, l));

arrPnts.Add(MbCartPoint(l / 2, l));

arrPnts.Add(MbCartPoint(-l / 2, -l));

arrPnts.Add(MbCartPoint(l / 2, -l));

MbLineSegment\* pS1 = new MbLineSegment(arrPnts[0], arrPnts[1]);

MbContour\* pContour = new MbContour(\*pS1, true);

MbArc\* pArc1 = new MbArc(MbCartPoint(0, 0), arrPnts[1], arrPnts[3], 0);

pContour->AddSegment(pArc1);

MbLineSegment\* pS2 = new MbLineSegment(arrPnts[3], arrPnts[2]);

pContour->AddSegment(pS2);

MbArc\* pArc2 = new MbArc(MbCartPoint(0, 0), arrPnts[2], arrPnts[0], 0);

pContour->AddSegment(pArc2);

MbPlacement3D plSurf(MbCartPoint3D(0 + OFS\_X, 0 + OFS\_Y, 0 + OFS\_Z),

MbCartPoint3D(hor + OFS\_X, vert + OFS\_Y, 0 + OFS\_Z),

MbCartPoint3D(0 + OFS\_X, 0 + OFS\_Y, 1 + OFS\_Z));

MbSurface\* pSurf1 = new MbPlane(plSurf);

\_arrSurfaces.push\_back(pSurf1);

\_arrContours.push\_back(pContour);

}

void MakeUserCommand0()

{

// Получение образующей в виде массивов поверхностей и контуров на них с помощью

// вспомогательной функции.

RPArray<MbSurface> arrSurfaces;

RPArray<MbContour> arrContours;

CreateSketch(arrSurfaces, arrContours, 0, 1, 30, 55, 0, 12);

CreateSketch(arrSurfaces, arrContours, 0, 1, 30, 40, 0, 12);

CreateSketch(arrSurfaces, arrContours, 0, 1, 40, 23, 0, 15);

CreateSketch(arrSurfaces, arrContours, 1, 0, 22, 10, 0, 15);

CreateSketch(arrSurfaces, arrContours, 0, 1, 5, 25, 0, 10);

CreateSketch(arrSurfaces, arrContours, 0, 1, 5, 30, 0, 3);

// Объект с параметрами операции заметания.

LoftedValues params;

// Объекты для именования элементов модели твердого тела.

MbSNameMaker names(ct\_CurveLoftedSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames(0, 1, false);

// Построение твердого тела заметания

MbSolid\* pSolid = NULL;

MbResultType res = ::LoftedSolid(arrSurfaces, arrContours, NULL, params,

NULL, NULL, names, contourNames, pSolid);

// Отображение построенного тела

if (res == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

// Уменьшение счетчика ссылок тела

::DeleteItem(pSolid);

}

**ПРИМЕР №7**

Задача: построение трубы (сдвиг).

**Поэтапное построение эскиза.**

1. Создаем функцию CreateSketch:

void CreateSketch(RPArray<MbContour>& \_arrContours)

{

...

}

2. Затем описываем CreateSketch (Все функции, которые мы применили объяснены в комментариях).

const double RAD\_OUTER = 35.0; // Радиус внешней окружности

const double RAD\_INNER = 30.0; // Радиус внутренней окружности

MbArc\* pCircleExt = new MbArc(MbCartPoint(0, 0), RAD\_OUTER);

MbArc\* pCircleInt = new MbArc(MbCartPoint(0, 0), RAD\_INNER);

\_arrContours.push\_back(new MbContour(\*pCircleExt, true));

\_arrContours.push\_back(new MbContour(\*pCircleInt, true));

3. Переходим к описанию основной функции.

void MakeUserCommand0()

{

...

}

4. Первое что мы делам, создаем локальную систему координат, массив контрольных точек для NURBS-сплайна .

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3D pl;

// Массив контрольных точек для построения NURBS-сплайна

std::vector<MbCartPoint3D> vecPnts = { { 0, 0, 0 }, { 0, 0, 150 } };

SArray<MbCartPoint3D> arrPnts(vecPnts);

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 2-го порядка по контрольным точкам

MbNurbs3D\* pSpline = MbNurbs3D::Create(2, arrPnts, false);

// Описание образующей кривой в виде плоского контура на плоскости XY мировой СК

MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

5. Далее, мы вызываем функцию отвечающую за создание эскиза.

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours;

CreateSketch(arrContours);

MbSweptData sweptData(\*pPlaneXY, arrContours);

// Объект с параметрами операции построения тела заметания

EvolutionValues params;

// Вариант плоскопараллельного движения образующей вдоль направляющей

params.parallel = 0;

// Служебные объекты-именователи для вызова геометрической операции

MbSNameMaker operNames(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames(1, 0, false);

contourNames.Add(&cNames);

MbSNameMaker splineNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

6. Вызываем операцию для построения тела заметания.

// Вызов операции построения тела заметания

MbSolid\* pSolid = NULL;

MbResultType res = ::EvolutionSolid(sweptData, \*pSpline, params, operNames,

contourNames, splineNames, pSolid);

// Отображение построенного тела

if (res == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

// Отображение направляющей кривой со смещением вдоль оси Y (в целях отображения,

// чтобы отображаемая кривая была смещена от поверхности тела).

pSpline->Move(MbVector3D(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, -50, 0)));

viewManager->AddObject(Style(3, RGB(0, 0, 255)), pSpline, &pl);

7. Уменьшаем счетчики ссылок.

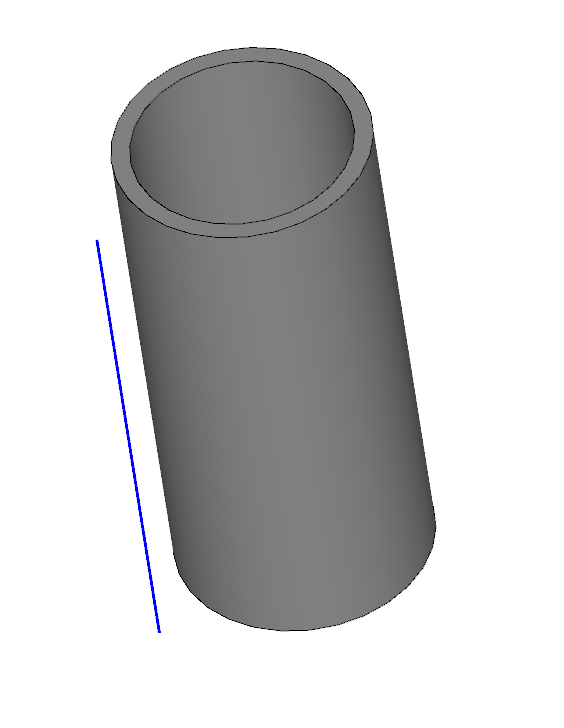
// Уменьшение счетчиков ссылок объектов ядра

::DeleteItem(pSolid);

::DeleteItem(pPlaneXY);

::DeleteItem(pSpline);

**Результаты построения**



**Код программы**

void CreateSketch(RPArray<MbContour>& \_arrContours)

{

const double RAD\_OUTER = 35.0; // Радиус внешней окружности

const double RAD\_INNER = 30.0; // Радиус внутренней окружности

MbArc\* pCircleExt = new MbArc(MbCartPoint(0, 0), RAD\_OUTER);

MbArc\* pCircleInt = new MbArc(MbCartPoint(0, 0), RAD\_INNER);

\_arrContours.push\_back(new MbContour(\*pCircleExt, true));

\_arrContours.push\_back(new MbContour(\*pCircleInt, true));

}

void MakeUserCommand0()

{

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3D pl;

// Массив контрольных точек для построения NURBS-сплайна

std::vector<MbCartPoint3D> vecPnts = { { 0, 0, 0 }, { 0, 0, 150 } };

SArray<MbCartPoint3D> arrPnts(vecPnts);

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 2-го порядка по контрольным точкам

MbNurbs3D\* pSpline = MbNurbs3D::Create(2, arrPnts, false);

// Описание образующей кривой в виде плоского контура на плоскости XY мировой СК

MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours;

CreateSketch(arrContours);

MbSweptData sweptData(\*pPlaneXY, arrContours);

// Объект с параметрами операции построения тела заметания

EvolutionValues params;

// Вариант плоскопараллельного движения образующей вдоль направляющей

params.parallel = 0;

// Служебные объекты-именователи для вызова геометрической операции

MbSNameMaker operNames(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames(1, 0, false);

contourNames.Add(&cNames);

MbSNameMaker splineNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

// Вызов операции построения тела заметания

MbSolid\* pSolid = NULL;

MbResultType res = ::EvolutionSolid(sweptData, \*pSpline, params, operNames,

contourNames, splineNames, pSolid);

// Отображение построенного тела

if (res == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

// Отображение направляющей кривой со смещением вдоль оси Y (в целях отображения,

// чтобы отображаемая кривая была смещена от поверхности тела).

pSpline->Move(MbVector3D(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, -50, 0)));

viewManager->AddObject(Style(3, RGB(0, 0, 255)), pSpline, &pl);

// Уменьшение счетчиков ссылок объектов ядра

::DeleteItem(pSolid);

::DeleteItem(pPlaneXY);

::DeleteItem(pSpline);

}

**ПРИМЕР №8**

Задача: создание червяка.

**Поэтапное построение эскиза.**

1. Создаем функцию описания контура резьбы червяка.  
   void CreateSketch01( RPArray<MbContour>& \_arrContours)

{

double R = 70;

double l = 7;

SArray<MbCartPoint>arrPnts(4);

arrPnts.Add(MbCartPoint(-l\*0.5 , l+R));

arrPnts.Add(MbCartPoint(l\*0.5 , l+R));

arrPnts.Add(MbCartPoint(l , -l+R));

arrPnts.Add(MbCartPoint(-l , -l+R));

MbLineSegment\* pS1 = new MbLineSegment(arrPnts[0], arrPnts[1]);

MbContour\* pContour = new MbContour(\*pS1, true);

MbLineSegment\* pS2 = new MbLineSegment(arrPnts[1], arrPnts[2]);

pContour->AddSegment(pS2);

MbLineSegment\* pS3 = new MbLineSegment(arrPnts[2], arrPnts[3]);

pContour->AddSegment(pS3);

MbLineSegment\* pS4 = new MbLineSegment(arrPnts[3], arrPnts[0]);

pContour->AddSegment(pS4);

\_arrContours.push\_back(pContour);

}

1. Создаем функцию описания контура цилиндра вала.

void CreateSketch02(RPArray<MbContour>& \_arrContours)

{

// Построение образующей в виде двух концентрических окружностей

const double RAD\_EXT = 66.0; // Радиус внешней окружности

const double RAD\_INT = 33.0; // Радиус внутренней окружности

MbArc\* pCircleExt = new MbArc(MbCartPoint(0, 0), RAD\_EXT);

MbArc\* pCircleInt = new MbArc(MbCartPoint(0, 0), RAD\_INT);

\_arrContours.push\_back(new MbContour(\*pCircleExt, true));

\_arrContours.push\_back(new MbContour(\*pCircleInt, true));

}

1. Переходим к описанию основной функции (Все функции, которые мы применили объяснены в комментариях).

void MakeUserCommand0()

{

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3D pl;

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 4-го порядка по контрольным точкам

MbConeSpiral\* pSpiral = new MbConeSpiral(MbCartPoint3D(0, 0, 0), MbCartPoint3D(100, 0, 0),

MbCartPoint3D(0,100, 0),20,false);

// Описание образующей кривой в виде плоского контура на плоскости XY мировой СК

MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(100, 0, 0),

MbCartPoint3D(0, 100, 0));

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours;

CreateSketch01( arrContours);

MbSweptData sweptData(\*pPlaneXY, arrContours);

// Объект с параметрами операции построения тела заметания

EvolutionValues params;

// Вариант плоскопараллельного движения образующей вдоль направляющей

params.parallel = 1;

// Служебные объекты-именователи для вызова геометрической операции

MbSNameMaker operNames(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames(1, 0, false);

contourNames.Add(&cNames);

MbSNameMaker splineNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

// Вызов операции построения тела заметания

MbSolid\* pSolid = NULL;

MbResultType res = ::EvolutionSolid(sweptData, \*pSpiral, params, operNames,

contourNames, splineNames, pSolid);

// Отображение построенного тела

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

if (res == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

//выдавливание цилиндра

MbPlane\* pPlaneXY2 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 100, 0),

MbCartPoint3D(0, 0, 100));

RPArray<MbContour> arrContours2;

CreateSketch02(arrContours2);

MbSweptData sweptData2(\*pPlaneXY2, arrContours2);

// Направляющий вектор для операции выдавливания

MbVector3D dir(1, 0, 0);

// Параметры операции выдавливания, задающие свойства тела для построения в прямом

// и в обратном направлении вдоль (глубина выдавливания и уклон).

const double HEIGHT\_FORWARD = 108.0, HEIGHT\_BACKWARD = 8.0;

ExtrusionValues extrusionParams(HEIGHT\_FORWARD, HEIGHT\_BACKWARD);

// Именователи элементов модели твердого тела и контуров образующей

MbSNameMaker operNames2(1, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames2(0, 1, false);

// Вызов функции-утилиты для построения твердого тела выдавливания

MbSolid\* pSolid2 = NULL;

MbResultType res2 = ::ExtrusionSolid(sweptData2, dir, NULL, NULL, false,

extrusionParams, operNames2, cNames2, pSolid2);

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid2);

// Отображение построенного тела

if (res2 == rt\_Success)

{

// Смещение тела по оси Y, чтобы при отображении оно не накладывалось на образующую

// pSolid->Move(MbVector3D(0, 0, 0));

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid2);

}

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem(pSolid2);

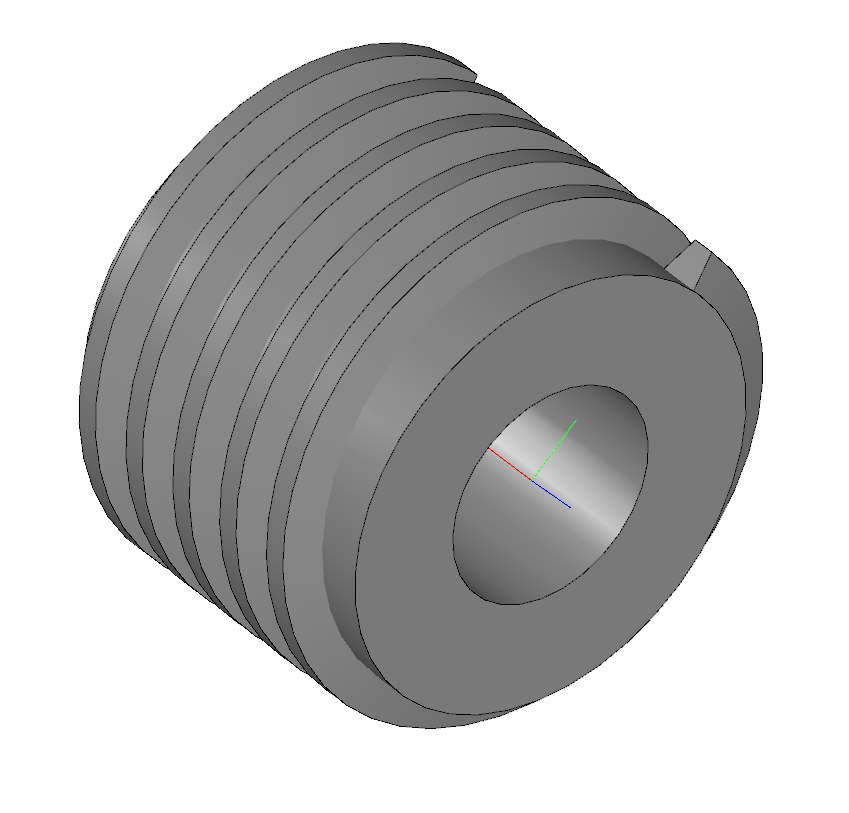
::DeleteItem(pSolid);

::DeleteItem(pPlaneXY);

::DeleteItem(pSpiral);

}

**Результаты построения**



**Код программы**

void CreateSketch01( RPArray<MbContour>& \_arrContours)

{

double R = 70;

double l = 7;

SArray<MbCartPoint>arrPnts(4);

arrPnts.Add(MbCartPoint(-l\*0.5 , l+R));

arrPnts.Add(MbCartPoint(l\*0.5 , l+R));

arrPnts.Add(MbCartPoint(l , -l+R));

arrPnts.Add(MbCartPoint(-l , -l+R));

MbLineSegment\* pS1 = new MbLineSegment(arrPnts[0], arrPnts[1]);

MbContour\* pContour = new MbContour(\*pS1, true);

MbLineSegment\* pS2 = new MbLineSegment(arrPnts[1], arrPnts[2]);

pContour->AddSegment(pS2);

MbLineSegment\* pS3 = new MbLineSegment(arrPnts [2], arrPnts[3]);

pContour->AddSegment(pS3);

MbLineSegment\* pS4 = new MbLineSegment(arrPnts[3], arrPnts[0]);

pContour->AddSegment(pS4);

\_arrContours.push\_back(pContour);

}

void CreateSketch02(RPArray<MbContour>& \_arrContours)

{

// Построение образующей в виде двух концентрических окружностей

const double RAD\_EXT = 66.0; // Радиус внешней окружности

const double RAD\_INT = 33.0; // Радиус внутренней окружности

MbArc\* pCircleExt = new MbArc(MbCartPoint(0, 0), RAD\_EXT);

MbArc\* pCircleInt = new MbArc(MbCartPoint(0, 0), RAD\_INT);

\_arrContours.push\_back(new MbContour(\*pCircleExt, true));

\_arrContours.push\_back(new MbContour(\*pCircleInt, true));

}

void MakeUserCommand0()

{

// Локальная СК (по умолчанию совпадает с мировой СК)

MbPlacement3D pl;

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 4-го порядка по контрольным точкам

MbConeSpiral\* pSpiral = new MbConeSpiral(MbCartPoint3D(0, 0, 0), MbCartPoint3D(100, 0, 0),

MbCartPoint3D(0,100, 0),20,false);

// Описание образующей кривой в виде плоского контура на плоскости XY мировой СК

MbPlane\* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(100, 0, 0),

MbCartPoint3D(0, 100, 0));

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours;

CreateSketch01( arrContours);

MbSweptData sweptData(\*pPlaneXY, arrContours);

// Объект с параметрами операции построения тела заметания

EvolutionValues params;

// Вариант плоскопараллельного движения образующей вдоль направляющей

params.parallel = 1;

// Служебные объекты-именователи для вызова геометрической операции

MbSNameMaker operNames(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames(1, 0, false);

contourNames.Add(&cNames);

MbSNameMaker splineNames(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

// Вызов операции построения тела заметания

MbSolid\* pSolid = NULL;

MbResultType res = ::EvolutionSolid(sweptData, \*pSpiral, params, operNames,

contourNames, splineNames, pSolid);

// Отображение построенного тела

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

if (res == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

//выдавливание цилиндра

MbPlane\* pPlaneXY2 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 100, 0),

MbCartPoint3D(0, 0, 100));

RPArray<MbContour> arrContours2;

CreateSketch02(arrContours2);

MbSweptData sweptData2(\*pPlaneXY2, arrContours2);

// Направляющий вектор для операции выдавливания

MbVector3D dir(1, 0, 0);

// Параметры операции выдавливания, задающие свойства тела для построения в прямом

// и в обратном направлении вдоль (глубина выдавливания и уклон).

const double HEIGHT\_FORWARD = 108.0, HEIGHT\_BACKWARD = 8.0;

ExtrusionValues extrusionParams(HEIGHT\_FORWARD, HEIGHT\_BACKWARD);

// Именователи элементов модели твердого тела и контуров образующей

MbSNameMaker operNames2(1, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames2(0, 1, false);

// Вызов функции-утилиты для построения твердого тела выдавливания

MbSolid\* pSolid2 = NULL;

MbResultType res2 = ::ExtrusionSolid(sweptData2, dir, NULL, NULL, false,

extrusionParams, operNames2, cNames2, pSolid2);

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid2);

// Отображение построенного тела

if (res2 == rt\_Success)

{

// Смещение тела по оси Y, чтобы при отображении оно не накладывалось на образующую

// pSolid->Move(MbVector3D(0, 0, 0));

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid2);

}

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem(pSolid2);

::DeleteItem(pSolid);

::DeleteItem(pPlaneXY);

::DeleteItem(pSpiral);

}

# ЗАКЛЮЧЕНИЕ

Данный проект нацелен на снижение порога вхождения для программистов, желающих работать с ядром C3D. В задачи входит описание базовых элементов и принципов работы ядра, содержательное описание каждого элемента, демонстрация кода ядра.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. C3D «Руководство разработчика»;
2. С3D, «Работа 1. Знакомство с ядром C3D»;
3. C3D, «Работа 2. Кривые в двумерном пространстве»;
4. C3D, «Работа 3. Составные кривые и сплайны в двумерном пространстве»;
5. C3D, «Работа 4. Основные типы поверхностей»;
6. Официальный сайт C3D Labs, URL: https://c3dlabs.com/en/.