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. В. Серяков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Москва 2020

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 2

ЦЕЛЬ ПРОЕКТА 3

РЕЗУЛЬТАТЫ ПРОЕКТА (СПИСОК ПРИМЕРОВ)

1. Пример №3 4
2. Пример №4 13
3. Пример №5 20
4. Пример №6 23
5. Пример №7 25
6. Пример №8 28
7. Пример №9 31
8. Пример №10 35
9. Пример №11 38

ЗАКЛЮЧЕНИЕ 41

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc516226223) 42

# 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 №3**

Задача: реализация 3D модели болта

**Поэтапное построение эскиза.**

1. Создаем эскиз шестиугольника:

void CreateSketch6(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 18;

// Количество сторон многоугольника

const int SIDE\_CNT = 6;

// Радиус описывающей окружности многоугольника

const double RAD1 = 45.0;

// Массив для хранения вершин ломаной

SArray<MbCartPoint> arrPnts(SIDE\_CNT);

// Вычисление вершин ломаной равномерным делением окружности

for (int i = 0; i < SIDE\_CNT; i++)

{

// Угловое положение i-й вершины на описывающей окружности.

// Угловое положение начальной вершины - M\_PI/2 (эта вершина

// расположена на вертикальной оси).

double angle = M\_PI / 2 + 2 \* M\_PI / SIDE\_CNT \* i;

MbCartPoint pnt(RAD1 \* cos(angle), RAD1 \* sin(angle));

arrPnts.Add(pnt);

}

// Построение единой ломаной внешнего контура по точкам

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

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

\_arrContours.push\_back(pContour);

}

1. Создаем эскиз окружности:

void CreateSketch7(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 16.5;

// Функция, отвечающая за построение окружности

MbArc\* pCircle = new MbArc(centerCircle, RAD);

MbContour\* pContourCircle = new MbContour(\*pCircle, true);

\_arrContours.push\_back(pContourCircle);

}

1. Создаем эскиз резьбы:

void CreateSketch8(RPArray<MbContour>& \_arrContours)

{

double R = 18;

double l = 2;

// Массив точек для создания контура

SArray<MbCartPoint>arrPnts(4);

arrPnts.Add(MbCartPoint(-l \* 0.1 + l, l + R));

arrPnts.Add(MbCartPoint(l \* 0.1 + l, l + R));

arrPnts.Add(MbCartPoint(2 \* l, -l + R));

arrPnts.Add(MbCartPoint(0, -l + R));

// Функция, отвечаюшая за построение линии из двух точек

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 эскиз еще одной окружности:

void CreateSketch9(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 20.25;

MbArc\* pCircle = new MbArc(centerCircle, RAD);

MbContour\* pContourCircle = new MbContour(\*pCircle, true);

\_arrContours.push\_back(pContourCircle);

}

1. Переходим к основному коду:

void MakeUserCommand9()

{

MbPlacement3D pl;

RPArray<MbContour> arrContours;

CreateSketch6(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

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 = 30.0, 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);

MbPlacement3D pl1;

RPArray<MbContour> arrContours1;

CreateSketch7(arrContours1);

// Отображение образующей (в плоскости XY глобальной СК)

for (int i = 0; i < arrContours1.size(); i++)

viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours1[i], &pl1);

// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ

MbPlane\* pPlaneXY1 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

// Объект, хранящий параметры образующей

MbSweptData sweptData1(\*pPlaneXY1, arrContours1);

// Направляющий вектор для операции выдавливания

MbVector3D dir1(0, 0, 1);

const double HEIGHT\_FORWARD1 = 100.0, HEIGHT\_BACKWARD1 = 0.0;

// Параметры операции выдавливания, задающие свойства тела для построения:

// расстояние выдавливания в прямом и в обратном направлении вдоль

// направляющего вектора

ExtrusionValues extrusionParam1(HEIGHT\_FORWARD1, HEIGHT\_BACKWARD1);

// Служебный объект для именования элементов модели твердого тела

MbSNameMaker operNames1(ct\_CurveExtrusionSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames1(0, 1, false);

// Построение твердого тела выдавливания

MbSolid\* pSolid1 = NULL;

MbResultType res1 = ::ExtrusionSolid(sweptData1, dir1, NULL, NULL, false,

extrusionParam1, operNames1, cNames1, pSolid1);

MbPlacement3D pl3;

RPArray<MbContour> arrContours3;

CreateSketch9(arrContours3);

// Отображение образующей (в плоскости XY глобальной СК)

for (int i = 0; i < arrContours3.size(); i++)

viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours3[i], &pl3);

// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ

MbPlane\* pPlaneXY3 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

// Объект, хранящий параметры образующей

MbSweptData sweptData3(\*pPlaneXY3, arrContours3);

// Направляющий вектор для операции выдавливания

MbVector3D dir3(0, 0, 1);

const double HEIGHT\_FORWARD2 = 10.0, HEIGHT\_BACKWARD2 = 0.0;

// Параметры операции выдавливания, задающие свойства тела для построения:

// расстояние выдавливания в прямом и в обратном направлении вдоль

// направляющего вектора

ExtrusionValues extrusionParam3(HEIGHT\_FORWARD2, HEIGHT\_BACKWARD2);

// Служебный объект для именования элементов модели твердого тела

MbSNameMaker operNames3(ct\_CurveExtrusionSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames3(0, 1, false);

// Построение твердого тела выдавливания

MbSolid\* pSolid3 = NULL;

MbResultType res3 = ::ExtrusionSolid(sweptData3, dir3, NULL, NULL, false,

extrusionParam3, operNames3, cNames3, pSolid3);

// Отображение построенного тела

if (res == rt\_Success)

{

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

}

if (res1 == rt\_Success)

{

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid1);

}

if (res1 == rt\_Success)

{

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid3);

}

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

MbPlacement3D pl2;

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 4-го порядка по контрольным точкам

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

MbCartPoint3D(0, 100, 0), 5, false);

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

MbPlane\* pPlaneXY2 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 0, -50),

MbCartPoint3D(0, 50, 0));

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours2;

CreateSketch8(arrContours2);

MbSweptData sweptData2(\*pPlaneXY2, arrContours2);

// Объект с параметрами операции построения тела заметания

EvolutionValues params2;

// Вариант плоскопараллельного движения образующей вдоль направляющей

params2.parallel = 1;

// Служебные объекты-именователи для вызова геометрической операции

MbSNameMaker operNames2(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames2(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames2(1, 0, false);

contourNames2.Add(&cNames2);

MbSNameMaker splineNames2(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

// Вызов операции построения тела заметания

MbSolid\* pSolid2 = NULL;

MbResultType res2 = ::EvolutionSolid(sweptData2, \*pSpiral, params2, operNames2,

contourNames2, splineNames2, pSolid2);

// Отображение построенного тела

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

if (res2 == rt\_Success)

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem(pSolid);

::DeleteItem(pPlaneXY);

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

::DeleteItem(arrContours[i]);

::DeleteItem(pSolid1);

::DeleteItem(pPlaneXY1);

for (int i = 0; i < arrContours1.size(); i++)

::DeleteItem(arrContours1[i]);

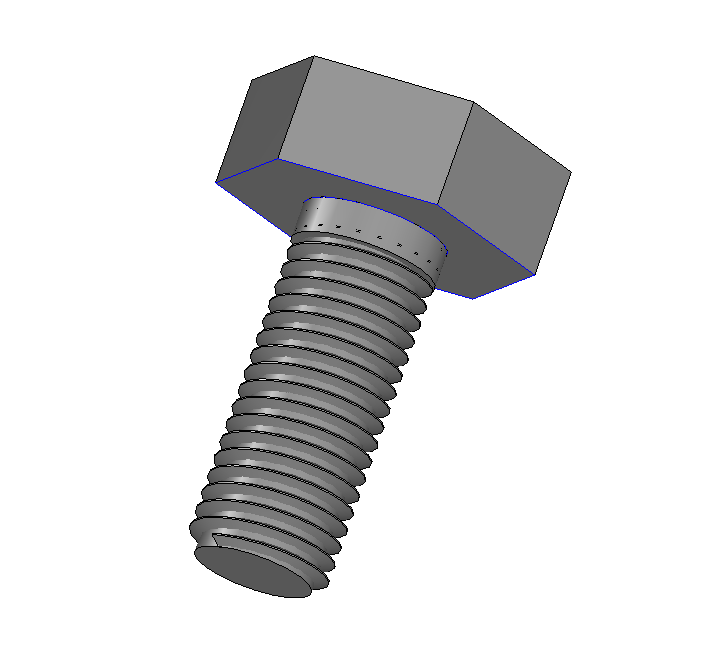
::DeleteItem(pSolid2);

::DeleteItem(pPlaneXY2);

::DeleteItem(pSpiral);

}

**Результат построения.**



**Код программы.**

void CreateSketch9(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 20.25;

MbArc\* pCircle = new MbArc(centerCircle, RAD);

MbContour\* pContourCircle = new MbContour(\*pCircle, true);

\_arrContours.push\_back(pContourCircle);

}

void CreateSketch8(RPArray<MbContour>& \_arrContours)

{

double R = 18;

double l = 2;

SArray<MbCartPoint>arrPnts(4);

arrPnts.Add(MbCartPoint(-l \* 0.1 + l, l + R));

arrPnts.Add(MbCartPoint(l \* 0.1 + l, l + R));

arrPnts.Add(MbCartPoint(2 \* l, -l + R));

arrPnts.Add(MbCartPoint(0, -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 CreateSketch7(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 16.5;

MbArc\* pCircle = new MbArc(centerCircle, RAD);

MbContour\* pContourCircle = new MbContour(\*pCircle, true);

\_arrContours.push\_back(pContourCircle);

}

void CreateSketch6(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 18;

// Количество сторон многоугольника

const int SIDE\_CNT = 6;

// Радиус описывающей окружности многоугольника

const double RAD1 = 45.0;

// Массив для хранения вершин ломаной

SArray<MbCartPoint> arrPnts(SIDE\_CNT);

// Вычисление вершин ломаной равномерным делением окружности

for (int i = 0; i < SIDE\_CNT; i++)

{

// Угловое положение i-й вершины на описывающей окружности.

// Угловое положение начальной вершины - M\_PI/2 (эта вершина

// расположена на вертикальной оси).

double angle = M\_PI / 2 + 2 \* M\_PI / SIDE\_CNT \* i;

MbCartPoint pnt(RAD1 \* cos(angle), RAD1 \* sin(angle));

arrPnts.Add(pnt);

}

// Построение единой ломаной внешнего контура по точкам

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

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

\_arrContours.push\_back(pContour);

}

void MakeUserCommand9()

{

MbPlacement3D pl;

RPArray<MbContour> arrContours;

CreateSketch6(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

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 = 30.0, 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);

MbPlacement3D pl1;

RPArray<MbContour> arrContours1;

CreateSketch7(arrContours1);

// Отображение образующей (в плоскости XY глобальной СК)

for (int i = 0; i < arrContours1.size(); i++)

viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours1[i], &pl1);

// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ

MbPlane\* pPlaneXY1 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

// Объект, хранящий параметры образующей

MbSweptData sweptData1(\*pPlaneXY1, arrContours1);

// Направляющий вектор для операции выдавливания

MbVector3D dir1(0, 0, 1);

const double HEIGHT\_FORWARD1 = 100.0, HEIGHT\_BACKWARD1 = 0.0;

// Параметры операции выдавливания, задающие свойства тела для построения:

// расстояние выдавливания в прямом и в обратном направлении вдоль

// направляющего вектора

ExtrusionValues extrusionParam1(HEIGHT\_FORWARD1, HEIGHT\_BACKWARD1);

// Служебный объект для именования элементов модели твердого тела

MbSNameMaker operNames1(ct\_CurveExtrusionSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames1(0, 1, false);

// Построение твердого тела выдавливания

MbSolid\* pSolid1 = NULL;

MbResultType res1 = ::ExtrusionSolid(sweptData1, dir1, NULL, NULL, false,

extrusionParam1, operNames1, cNames1, pSolid1);

MbPlacement3D pl3;

RPArray<MbContour> arrContours3;

CreateSketch9(arrContours3);

// Отображение образующей (в плоскости XY глобальной СК)

for (int i = 0; i < arrContours3.size(); i++)

viewManager->AddObject(Style(1, RGB(0, 0, 255)), arrContours3[i], &pl3);

// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ

MbPlane\* pPlaneXY3 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),

MbCartPoint3D(0, 1, 0));

// Объект, хранящий параметры образующей

MbSweptData sweptData3(\*pPlaneXY3, arrContours3);

// Направляющий вектор для операции выдавливания

MbVector3D dir3(0, 0, 1);

const double HEIGHT\_FORWARD2 = 10.0, HEIGHT\_BACKWARD2 = 0.0;

// Параметры операции выдавливания, задающие свойства тела для построения:

// расстояние выдавливания в прямом и в обратном направлении вдоль

// направляющего вектора

ExtrusionValues extrusionParam3(HEIGHT\_FORWARD2, HEIGHT\_BACKWARD2);

// Служебный объект для именования элементов модели твердого тела

MbSNameMaker operNames3(ct\_CurveExtrusionSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> cNames3(0, 1, false);

// Построение твердого тела выдавливания

MbSolid\* pSolid3 = NULL;

MbResultType res3 = ::ExtrusionSolid(sweptData3, dir3, NULL, NULL, false,

extrusionParam3, operNames3, cNames3, pSolid3);

// Отображение построенного тела

if (res == rt\_Success)

{

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

}

if (res1 == rt\_Success)

{

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid1);

}

if (res1 == rt\_Success)

{

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid3);

}

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

MbPlacement3D pl2;

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 4-го порядка по контрольным точкам

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

MbCartPoint3D(0, 100, 0), 5, false);

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

MbPlane\* pPlaneXY2 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 0, -50),

MbCartPoint3D(0, 50, 0));

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours2;

CreateSketch8(arrContours2);

MbSweptData sweptData2(\*pPlaneXY2, arrContours2);

// Объект с параметрами операции построения тела заметания

EvolutionValues params2;

// Вариант плоскопараллельного движения образующей вдоль направляющей

params2.parallel = 1;

// Служебные объекты-именователи для вызова геометрической операции

MbSNameMaker operNames2(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames2(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

PArray<MbSNameMaker> contourNames2(1, 0, false);

contourNames2.Add(&cNames2);

MbSNameMaker splineNames2(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

// Вызов операции построения тела заметания

MbSolid\* pSolid2 = NULL;

MbResultType res2 = ::EvolutionSolid(sweptData2, \*pSpiral, params2, operNames2,

contourNames2, splineNames2, pSolid2);

// Отображение построенного тела

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

if (res2 == rt\_Success)

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem(pSolid);

::DeleteItem(pPlaneXY);

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

::DeleteItem(arrContours[i]);

::DeleteItem(pSolid1);

::DeleteItem(pPlaneXY1);

for (int i = 0; i < arrContours1.size(); i++)

::DeleteItem(arrContours1[i]);

::DeleteItem(pSolid2);

::DeleteItem(pPlaneXY2);

::DeleteItem(pSpiral);

}

**ПРИМЕР №4**

Задача: реализация 3D модели гайки.

**Поэтапное построение эскиза.**

1. Создаем эскиз шестиугольника, описанного около окружности:

void CreateSketch2(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 20;

// Количество сторон многоугольника

const int SIDE\_CNT = 6;

// Радиус описывающей окружности многоугольника

const double RAD1 = 45.0;

// Массив для хранения вершин ломаной

SArray<MbCartPoint> arrPnts(SIDE\_CNT);

// Вычисление вершин ломаной равномерным делением окружности

for (int i = 0; i < SIDE\_CNT; i++)

{

// Угловое положение i-й вершины на описывающей окружности.

// Угловое положение начальной вершины - M\_PI/2 (эта вершина

// расположена на вертикальной оси).

double angle = M\_PI / 2 + 2 \* M\_PI / SIDE\_CNT \* i;

MbCartPoint pnt(RAD1 \* cos(angle), RAD1 \* sin(angle));

arrPnts.Add(pnt);

}

// Построение единой ломаной внешнего контура по точкам

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

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

// Построение окружности

MbArc\* pCircle = new MbArc(centerCircle, RAD);

MbContour\* pContourCircle = new MbContour(\*pCircle, true);

\_arrContours.push\_back(pContour);

\_arrContours.push\_back(pContourCircle);

}

1. Построение эскиза резьбы:

void CreateSketch01(RPArray<MbContour>& \_arrContours)

{

// Параметры для эскиза

double R = 18;

double l = 2;

// Массив точек для массив

SArray<MbCartPoint>arrPnts(4);

arrPnts.Add(MbCartPoint(-l \* 0.1 + l, -l + R));

arrPnts.Add(MbCartPoint(l \* 0.1 + l, -l + R));

arrPnts.Add(MbCartPoint(2\*l, l + R));

arrPnts.Add(MbCartPoint(0, l + R));

// Создание единого контура

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. Переходим к основному коду:

void MakeUserCommand0()

{

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

MbPlacement3D pl;

// СОЗДАНИЕ КОНТУРОВ ДЛЯ ОБРАЗУЮЩЕЙ

RPArray<MbContour> arrContours;

CreateSketch2(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 = 30.0, 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);

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

MbPlacement3D pl1;

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 4-го порядка по контрольным точкам

MbConeSpiral\* pSpiral = new MbConeSpiral(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 0, -26), MbCartPoint3D(0, 26, 0), 4, false);

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

MbPlane\* pPlaneXY1 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 0, -50),

MbCartPoint3D(0, 50, 0));

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours1;

CreateSketch01(arrContours1);

MbSweptData sweptData1(\*pPlaneXY1, arrContours1);

// 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 operNames1(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames1(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

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

contourNames.Add(&cNames1);

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

// Вызов операции построения тела заметания

MbSolid\* pSolid1 = NULL;

MbResultType res1 = ::EvolutionSolid(sweptData1, \*pSpiral, params, operNames1,

contourNames, splineNames, pSolid1);

// Отображение построенного тела

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid1);

if (res1 == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid1);

// Отображение построенного тела

if (res == rt\_Success)

{

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]);

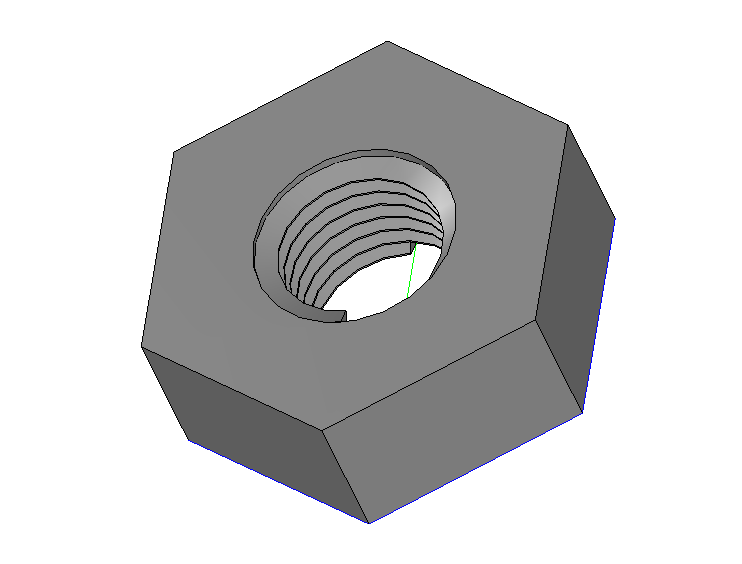
::DeleteItem(pSolid1);

::DeleteItem(pPlaneXY1);

::DeleteItem(pSpiral);

}

**Результат построения.**



**Код программы.**

void CreateSketch2(RPArray<MbContour>& \_arrContours)

{

// Центры и радиусы окружностей, дуги которых входят в контур

const MbCartPoint centerCircle(0, 0);

const double RAD = 20;

// Количество сторон многоугольника

const int SIDE\_CNT = 6;

// Радиус описывающей окружности многоугольника

const double RAD1 = 45.0;

// Массив для хранения вершин ломаной

SArray<MbCartPoint> arrPnts(SIDE\_CNT);

// Вычисление вершин ломаной равномерным делением окружности

for (int i = 0; i < SIDE\_CNT; i++)

{

// Угловое положение i-й вершины на описывающей окружности.

// Угловое положение начальной вершины - M\_PI/2 (эта вершина

// расположена на вертикальной оси).

double angle = M\_PI / 2 + 2 \* M\_PI / SIDE\_CNT \* i;

MbCartPoint pnt(RAD1 \* cos(angle), RAD1 \* sin(angle));

arrPnts.Add(pnt);

}

// Построение единой ломаной внешнего контура по точкам

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

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

MbArc\* pCircle = new MbArc(centerCircle, RAD);

MbContour\* pContourCircle = new MbContour(\*pCircle, true);

\_arrContours.push\_back(pContour);

\_arrContours.push\_back(pContourCircle);

}

void CreateSketch01(RPArray<MbContour>& \_arrContours)

{

double R = 18;

double l = 2;

SArray<MbCartPoint>arrPnts(4);

arrPnts.Add(MbCartPoint(-l \* 0.1 + l, -l + R));

arrPnts.Add(MbCartPoint(l \* 0.1 + l, -l + R));

arrPnts.Add(MbCartPoint(2\*l, l + R));

arrPnts.Add(MbCartPoint(0, 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 MakeUserCommand0()

{

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

// СОЗДАНИЕ КОНТУРОВ ДЛЯ ОБРАЗУЮЩЕЙ

RPArray<MbContour> arrContours;

CreateSketch2(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 = 30.0, 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);

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

MbPlacement3D pl1;

// Построение направляющей кривой в виде незамкнутого NURBS-сплайна

// 4-го порядка по контрольным точкам

MbConeSpiral\* pSpiral = new MbConeSpiral(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 0, -26),

MbCartPoint3D(0, 26, 0), 4, false);

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

MbPlane\* pPlaneXY1 = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 0, -50),

MbCartPoint3D(0, 50, 0));

// Построение образующей кривой с помощью вспомогательной функции CreateSketch

RPArray<MbContour> arrContours1;

CreateSketch01(arrContours1);

MbSweptData sweptData1(\*pPlaneXY1, arrContours1);

// 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 operNames1(ct\_CurveEvolutionSolid, MbSNameMaker::i\_SideNone, 0);

MbSNameMaker cNames1(ct\_CurveSweptSolid, MbSNameMaker::i\_SideNone, 0);

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

contourNames.Add(&cNames1);

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

// Вызов операции построения тела заметания

MbSolid\* pSolid1 = NULL;

MbResultType res1 = ::EvolutionSolid(sweptData1, \*pSpiral, params, operNames1,

contourNames, splineNames, pSolid1);

// Отображение построенного тела

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid1);

if (res1 == rt\_Success)

viewManager->AddObject(Style(1, LIGHTGRAY), pSolid1);

// Отображение построенного тела

if (res == rt\_Success)

{

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]);

::DeleteItem(pSolid1);

::DeleteItem(pPlaneXY1);

::DeleteItem(pSpiral);

}

**ПРИМЕР №5**

Задача: реализация 3D модели вазы

**Поэтапное построение эскиза.**

1. Описываем основной код:

void MakeUserCommand8()

{

// Базовая поверхность вращения: построение по образующей, оси вращения и углу

SArray<MbCartPoint3D> arrGenPnts(7);

arrGenPnts.Add(MbCartPoint3D(0, -4.5, 0));

arrGenPnts.Add(MbCartPoint3D(4, 10, 0));

arrGenPnts.Add(MbCartPoint3D(0, 15, 0));

MbBezier3D\* pGenCurve = new MbBezier3D(arrGenPnts, false);

MbAxis3D axRev(MbCartPoint3D(0, 0, 0), MbVector3D(0, 1, 0));

MbRevolutionSurface\* pBaseSurf = new MbRevolutionSurface(

\*pGenCurve, axRev, 2 \* M\_PI, false);

// Построение эквидистантной поверхности

MbOffsetSurface\* pOffSurf = new MbOffsetSurface(\*pBaseSurf, 7, false);

// Построение цилиндрической поверхности

double height\_Cyl = -0.1;

double radius\_Cyl = 5.5;

MbCartPoint3D baseCenter1(0, 0, 0); // Центр первого основания

MbCartPoint3D baseCenter2(0, height\_Cyl, 0); // Центр второго основания

// Точка на втором основании для указания радиуса цилиндра

MbCartPoint3D pntOnBase2(radius\_Cyl, height\_Cyl, 0);

// Вызов функции ядра для создания элементарной поверхности

MbSurface\* pCylSurf = NULL;

MbResultType resCylSurf = ::ElementarySurface(baseCenter1, baseCenter2, pntOnBase2,

st\_CylinderSurface, pCylSurf);

// Построение цилиндрического тела

MbSolid\* pCyl = NULL;

if (resCylSurf == rt\_Success)

{

// Вспомогательный объект для именования составных элементов твердого тела

MbSNameMaker namesCyl(ct\_ElementarySolid, MbSNameMaker::i\_SideNone, 0);

// Вызов функции ядра для построения тела на основе элементарной поверхности

MbResultType resSolid = ::ElementarySolid(\*pCylSurf, namesCyl, pCyl);

if (resSolid == rt\_Success)

viewManager->AddObject(Style(1, LIGHTRED), pCyl);

}

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pGenCurve);

::DeleteItem(pBaseSurf);

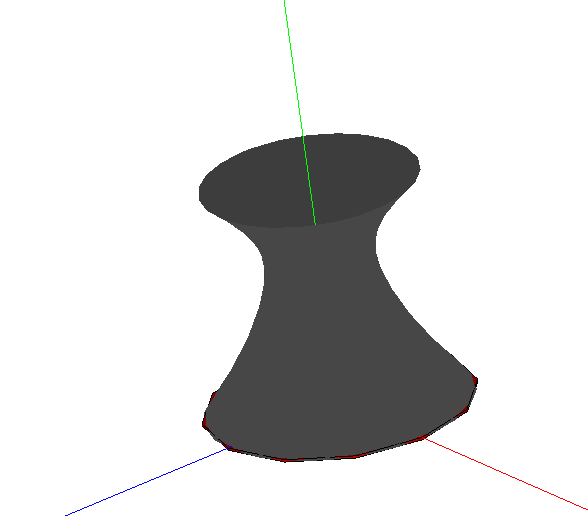
::DeleteItem(pOffSurf);

::DeleteItem(pCylSurf);

::DeleteItem(pCyl);

}

**Результат построения.**



**Код программы.**

void MakeUserCommand8()

{

// Базовая поверхность вращения: построение по образующей, оси вращения и углу

SArray<MbCartPoint3D> arrGenPnts(7);

arrGenPnts.Add(MbCartPoint3D(0, -4.5, 0));

arrGenPnts.Add(MbCartPoint3D(4, 10, 0));

arrGenPnts.Add(MbCartPoint3D(0, 15, 0));

MbBezier3D\* pGenCurve = new MbBezier3D(arrGenPnts, false);

MbAxis3D axRev(MbCartPoint3D(0, 0, 0), MbVector3D(0, 1, 0));

MbRevolutionSurface\* pBaseSurf = new MbRevolutionSurface(

\*pGenCurve, axRev, 2 \* M\_PI, false);

// Построение эквидистантной поверхности

MbOffsetSurface\* pOffSurf = new MbOffsetSurface(\*pBaseSurf, 7, false);

// Построение цилиндрической поверхности

double height\_Cyl = -0.1;

double radius\_Cyl = 5.5;

MbCartPoint3D baseCenter1(0, 0, 0); // Центр первого основания

MbCartPoint3D baseCenter2(0, height\_Cyl, 0); // Центр второго основания

// Точка на втором основании для указания радиуса цилиндра

MbCartPoint3D pntOnBase2(radius\_Cyl, height\_Cyl, 0);

// Вызов функции ядра для создания элементарной поверхности

MbSurface\* pCylSurf = NULL;

MbResultType resCylSurf = ::ElementarySurface(baseCenter1, baseCenter2, pntOnBase2,

st\_CylinderSurface, pCylSurf);

// Построение цилиндрического тела

MbSolid\* pCyl = NULL;

if (resCylSurf == rt\_Success)

{

// Вспомогательный объект для именования составных элементов твердого тела

MbSNameMaker namesCyl(ct\_ElementarySolid, MbSNameMaker::i\_SideNone, 0);

// Вызов функции ядра для построения тела на основе элементарной поверхности

MbResultType resSolid = ::ElementarySolid(\*pCylSurf, namesCyl, pCyl);

if (resSolid == rt\_Success)

viewManager->AddObject(Style(1, LIGHTRED), pCyl);

}

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pGenCurve);

::DeleteItem(pBaseSurf);

::DeleteItem(pOffSurf);

::DeleteItem(pCylSurf);

::DeleteItem(pCyl);

}

**ПРИМЕР №6**

Задача: реализация 3D модели “эллипсоид”

**Поэтапное построение эскиза.**

1. Описание основного кода:

void MakeUserCommand7()

{

// Базовая поверхность вращения: построение по образующей, оси вращения и углу

SArray<MbCartPoint3D> arrGenPnts(7);

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

arrGenPnts.Add(MbCartPoint3D(7, 10, 0));

arrGenPnts.Add(MbCartPoint3D(0, 20, 0));

// Сплайн Безье в трёхмерном пространстве.

MbBezier3D\* pGenCurve = new MbBezier3D(arrGenPnts, false);

// Ось - вектор, привязанный к фиксированной точке

MbAxis3D axRev(MbCartPoint3D(0, 0, 0), MbVector3D(0, 1, 0));

// Поверхность вращения.

MbRevolutionSurface\* pBaseSurf = new MbRevolutionSurface(\*pGenCurve, axRev, 2 \* M\_PI, false);

// Отображение поверхностей

viewManager->AddObject(Style(1, YELLOW), pBaseSurf);

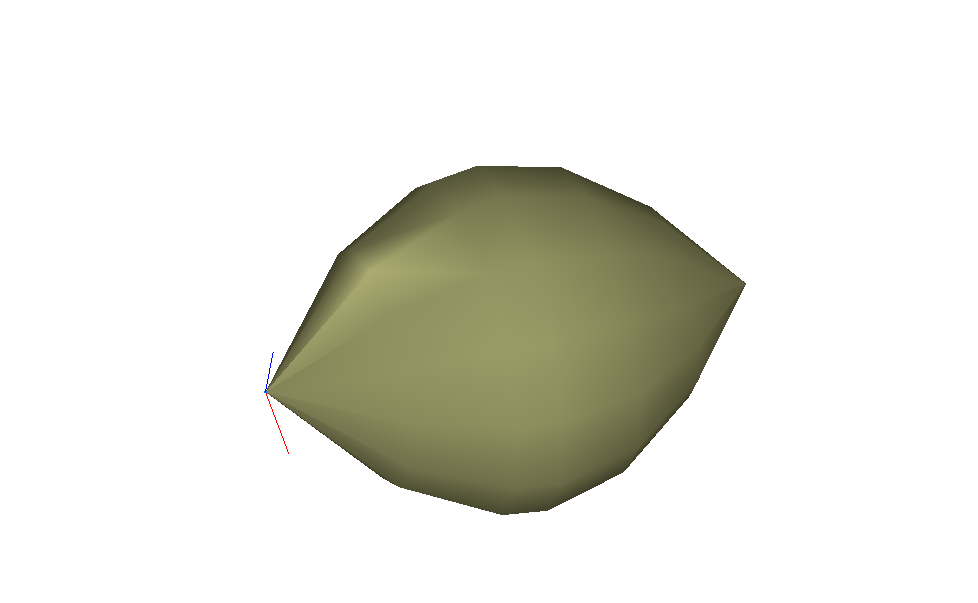
// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pGenCurve);

::DeleteItem(pBaseSurf);

}

**Результат построения.**



**Код программы.**

void MakeUserCommand7()

{

// Базовая поверхность вращения: построение по образующей, оси вращения и углу

SArray<MbCartPoint3D> arrGenPnts(7);

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

arrGenPnts.Add(MbCartPoint3D(7, 10, 0));

arrGenPnts.Add(MbCartPoint3D(0, 20, 0));

MbBezier3D\* pGenCurve = new MbBezier3D(arrGenPnts, false);

MbAxis3D axRev(MbCartPoint3D(0, 0, 0), MbVector3D(0, 1, 0));

MbRevolutionSurface\* pBaseSurf = new MbRevolutionSurface(

\*pGenCurve, axRev, 2 \* M\_PI, false);

// Отображение поверхностей

viewManager->AddObject(Style(1, YELLOW), pBaseSurf);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pGenCurve);

::DeleteItem(pBaseSurf);

}

**ПРИМЕР №7**

Задача: реализация 3D модели криволинейная поверхность

**Поэтапное построение эскиза.**

1. Описание основного кода:

void MakeUserCommand6()

{

// Создание пересекающихся кривых pCurve0, pCurve1, pCurve2 - границ поверхности.

// Незамкнутая кривая Безье для приближенного представления одного периода

// синусоиды в плоскости XY (по SIN\_PNT\_CNT точкам).

SArray<MbCartPoint3D> arrPntsBezier;

const double SIN\_WIDTH = 7;

const int SIN\_PNT\_CNT = 15;

for (int i = 0; i < SIN\_PNT\_CNT; i++)

{

double x = i \* 2 \* M\_PI / SIN\_PNT\_CNT;

double y = 5 \* sin(x);

arrPntsBezier.push\_back(MbCartPoint3D(x \* SIN\_WIDTH, y, 0));

}

MbBezier3D\* pCurve0 = new MbBezier3D(arrPntsBezier, false);

// Отрезок в плоскости XZ (начальная точка совпадает с начальной точкой pCurve0)

MbLineSegment3D\* pCurve1 = new MbLineSegment3D(arrPntsBezier[0],

MbCartPoint3D(3, -1, 25));

// Дуга окружности, соединяющая pCurve0 и pCurve1 по конечным точкам

// Дуга строится по трем точкам. Начальная точка pc1 и конечная pc2 совпадают

// с конечными точками pCurve0 и pCurve1.

// Промежуточная точка дуги в этом примере вычисляется произвольным образом

// (точка - центр отрезка pc1-pc3 смещается вниз по оси y).

MbCartPoint3D pc1 = arrPntsBezier[SIN\_PNT\_CNT - 1];

MbCartPoint3D pc3 = pCurve1->GetPoint2();

MbCartPoint3D pc2;

pc2.x = (pc1.x + pc3.x) / 2;

pc2.y = (pc1.y + pc3.y) / 2 - 10;

pc2.z = (pc1.z + pc3.z) / 2;

MbArc3D\* pCurve2 = new MbArc3D(pc1, pc2, pc3, 1, false);

// Построение поверхности

MbCornerSurface\* pSurf = new MbCornerSurface(\*pCurve0, \*pCurve1, \*pCurve2);

// Отображение поверхности

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

// Отображение граничных кривых

viewManager->AddObject(Style(5, LIGHTRED), pCurve0);

viewManager->AddObject(Style(5, LIGHTRED), pCurve1);

viewManager->AddObject(Style(5, LIGHTRED), pCurve2);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf);

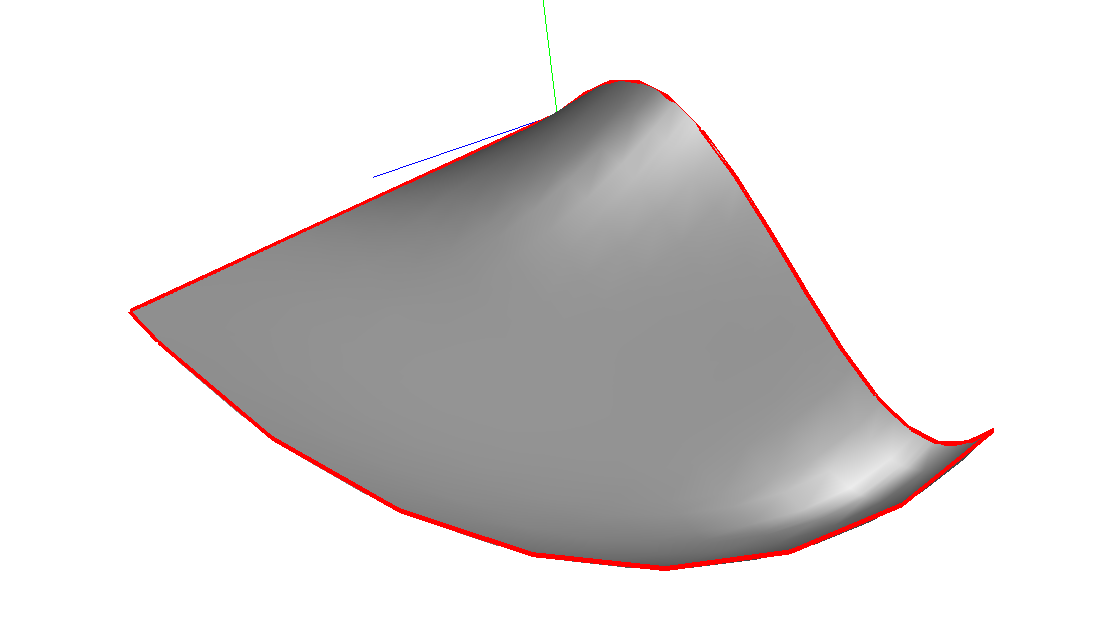
::DeleteItem(pCurve0);

::DeleteItem(pCurve1);

::DeleteItem(pCurve2);

}

**Результат построения.**



**Код программы.**

void MakeUserCommand6()

{

// Создание пересекающихся кривых pCurve0, pCurve1, pCurve2 - границ поверхности.

// Незамкнутая кривая Безье для приближенного представления одного периода

// синусоиды в плоскости XY (по SIN\_PNT\_CNT точкам).

SArray<MbCartPoint3D> arrPntsBezier;

const double SIN\_WIDTH = 7;

const int SIN\_PNT\_CNT = 15;

for (int i = 0; i < SIN\_PNT\_CNT; i++)

{

double x = i \* 2 \* M\_PI / SIN\_PNT\_CNT;

double y = 5 \* sin(x);

arrPntsBezier.push\_back(MbCartPoint3D(x \* SIN\_WIDTH, y, 0));

}

MbBezier3D\* pCurve0 = new MbBezier3D(arrPntsBezier, false);

// Отрезок в плоскости XZ (начальная точка совпадает с начальной точкой pCurve0)

MbLineSegment3D\* pCurve1 = new MbLineSegment3D(arrPntsBezier[0],

MbCartPoint3D(3, -1, 25));

// Дуга окружности, соединяющая pCurve0 и pCurve1 по конечным точкам

// Дуга строится по трем точкам. Начальная точка pc1 и конечная pc2 совпадают

// с конечными точками pCurve0 и pCurve1.

// Промежуточная точка дуги в этом примере вычисляется произвольным образом

// (точка - центр отрезка pc1-pc3 смещается вниз по оси y).

MbCartPoint3D pc1 = arrPntsBezier[SIN\_PNT\_CNT - 1];

MbCartPoint3D pc3 = pCurve1->GetPoint2();

MbCartPoint3D pc2;

pc2.x = (pc1.x + pc3.x) / 2;

pc2.y = (pc1.y + pc3.y) / 2 - 10;

pc2.z = (pc1.z + pc3.z) / 2;

MbArc3D\* pCurve2 = new MbArc3D(pc1, pc2, pc3, 1, false);

// Построение поверхности

MbCornerSurface\* pSurf = new MbCornerSurface(\*pCurve0, \*pCurve1, \*pCurve2);

// Отображение поверхности

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

// Отображение граничных кривых

viewManager->AddObject(Style(5, LIGHTRED), pCurve0);

viewManager->AddObject(Style(5, LIGHTRED), pCurve1);

viewManager->AddObject(Style(5, LIGHTRED), pCurve2);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf);

::DeleteItem(pCurve0);

::DeleteItem(pCurve1);

::DeleteItem(pCurve2);

}

**ПРИМЕР №8**

Задача: реализация 3D модели построение эквидистантной поверхности

**Поэтапное построение эскиза.**

1. Описание основного кода:

void MakeUserCommand1()

{

// Базовая поверхность вращения: построение по образующей, оси вращения и углу

// Массив точек для эскиза

SArray<MbCartPoint3D> arrGenPnts(7);

arrGenPnts.Add(MbCartPoint3D(2.5, 20, 0));

arrGenPnts.Add(MbCartPoint3D(4, 10, 0));

arrGenPnts.Add(MbCartPoint3D(9.7, 7.4, 0));

arrGenPnts.Add(MbCartPoint3D(9.7, 4, 0));

arrGenPnts.Add(MbCartPoint3D(7.6, 2.2, 0));

arrGenPnts.Add(MbCartPoint3D(10, -7, 0));

arrGenPnts.Add(MbCartPoint3D(9.8, -11, 0));

MbBezier3D\* pGenCurve = new MbBezier3D(arrGenPnts, false);

MbAxis3D axRev(MbCartPoint3D(0, 0, 0), MbVector3D(0, 1, 0));

MbRevolutionSurface\* pBaseSurf = new MbRevolutionSurface(

\*pGenCurve, axRev, 2 \* M\_PI, false);

// Построение эквидистантной поверхности

MbOffsetSurface\* pOffSurf = new MbOffsetSurface(\*pBaseSurf, 5, false);

// Смещение pOffSurf вдоль оси X для удобства отображения

pOffSurf->Move(MbVector3D(45, 0, 0));

// Отображение поверхностей

viewManager->AddObject(Style(1, GREEN), pBaseSurf);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

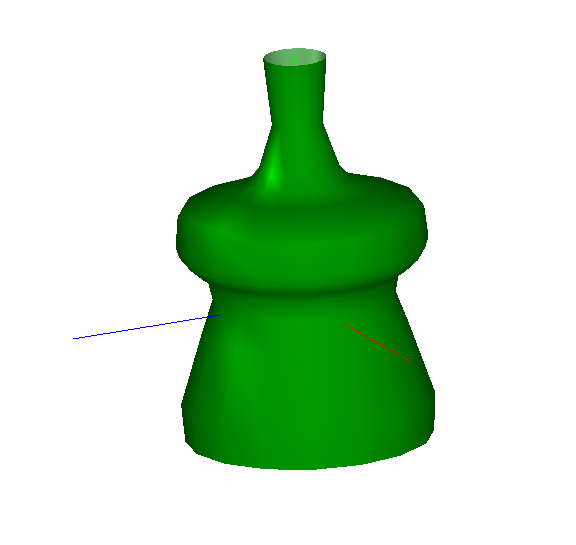
::DeleteItem(pGenCurve);

::DeleteItem(pBaseSurf);

::DeleteItem(pOffSurf);

}

**Результат построения.**



**Код программы.**

void MakeUserCommand1()

{

// Базовая поверхность вращения: построение по образующей, оси вращения и углу

SArray<MbCartPoint3D> arrGenPnts(7);

arrGenPnts.Add(MbCartPoint3D(2.5, 20, 0));

arrGenPnts.Add(MbCartPoint3D(4, 10, 0));

arrGenPnts.Add(MbCartPoint3D(9.7, 7.4, 0));

arrGenPnts.Add(MbCartPoint3D(9.7, 4, 0));

arrGenPnts.Add(MbCartPoint3D(7.6, 2.2, 0));

arrGenPnts.Add(MbCartPoint3D(10, -7, 0));

arrGenPnts.Add(MbCartPoint3D(9.8, -11, 0));

MbBezier3D\* pGenCurve = new MbBezier3D(arrGenPnts, false);

MbAxis3D axRev(MbCartPoint3D(0, 0, 0), MbVector3D(0, 1, 0));

MbRevolutionSurface\* pBaseSurf = new MbRevolutionSurface(

\*pGenCurve, axRev, 2 \* M\_PI, false);

// Построение эквидистантной поверхности

MbOffsetSurface\* pOffSurf = new MbOffsetSurface(\*pBaseSurf, 5, false);

// Смещение pOffSurf вдоль оси X для удобства отображения

pOffSurf->Move(MbVector3D(45, 0, 0));

// Отображение поверхностей

viewManager->AddObject(Style(1, GREEN), pBaseSurf);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pGenCurve);

::DeleteItem(pBaseSurf);

::DeleteItem(pOffSurf);

}

**ПРИМЕР №9**

Задача: реализация 3D модели Сеточная поверхность (Лист изогнутый)

**Поэтапное построение эскиза.**

1. Описание основного кода:

void MakeUserCommand2()

{

// Первое семейство координатных кривых:

// - кривые Безье, представляющие два периода синусоиды (растянутые

// на отрезок U\_SIN\_DX).

// - U\_CURVES\_CNT кривых расположены вдоль оси Z на равных промежутках

// в интервале шириной U\_DZ

const int U\_CURVES\_CNT = 10;

const double U\_DZ = 30;

const int U\_SIN\_PERIOD\_CNT = 2;

const double U\_SIN\_DX = 15;

const int U\_BEZIER\_PNT\_CNT = 15;

RPArray<MbCurve3D> arrCurvesU;

for (int i = 0; i < U\_CURVES\_CNT; i++)

{

SArray<MbCartPoint3D> arrPnts;

for (int n = 0; n < U\_BEZIER\_PNT\_CNT; n++)

{

MbCartPoint3D p;

p.x = n \* U\_SIN\_DX / (U\_BEZIER\_PNT\_CNT - 1);

p.y = sin(p.x / U\_SIN\_DX \* 2 \* M\_PI \* U\_SIN\_PERIOD\_CNT);

p.z = i \* U\_DZ / (U\_CURVES\_CNT - 1);

arrPnts.push\_back(p);

}

MbBezier3D\* pc = new MbBezier3D(arrPnts, false);

arrCurvesU.Add(pc);

}

// Второе семейство координатных кривых:

// - кривые Безье, представляющие три периода синусоиды (растянутые

// на отрезок V\_SIN\_DZ).

// - V\_CURVES\_CNT кривых расположены вдоль оси X на равных промежутках

// в интервале шириной V\_DX

const int V\_CURVES\_CNT = 5;

const double V\_DX = U\_SIN\_DX;

const int V\_SIN\_PERIOD\_CNT = 3;

const double V\_SIN\_DZ = U\_DZ;

const int V\_BEZIER\_PNT\_CNT = 15;

RPArray<MbCurve3D> arrCurvesV;

for (int i = 0; i < V\_CURVES\_CNT; i++)

{

SArray<MbCartPoint3D> arrPnts;

for (int n = 0; n < V\_BEZIER\_PNT\_CNT; n++)

{

MbCartPoint3D p;

p.x = i \* V\_DX / (V\_CURVES\_CNT - 1);

p.z = n \* V\_SIN\_DZ / (V\_BEZIER\_PNT\_CNT - 1);

p.y = sin(p.z / V\_SIN\_DZ \* 2 \* M\_PI \* V\_SIN\_PERIOD\_CNT);

arrPnts.push\_back(p);

}

MbBezier3D\* pc = new MbBezier3D(arrPnts, false);

arrCurvesV.Add(pc);

}

// Построение поверхности

MbMeshSurface\* pSurf = new MbMeshSurface(arrCurvesU, arrCurvesV,

false, false, false);

// Отображение поверхности

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

// Отображение координатных кривых

for (int i = 0, iLim = arrCurvesU.size(); i < iLim; i++)

viewManager->AddObject(Style(5, LIGHTRED), arrCurvesU[i]);

for (int i = 0, iLim = arrCurvesV.size(); i < iLim; i++)

viewManager->AddObject(Style(5, LIGHTBLUE), arrCurvesV[i]);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf);

for (int i = 0, iLim = arrCurvesU.size(); i < iLim; i++)

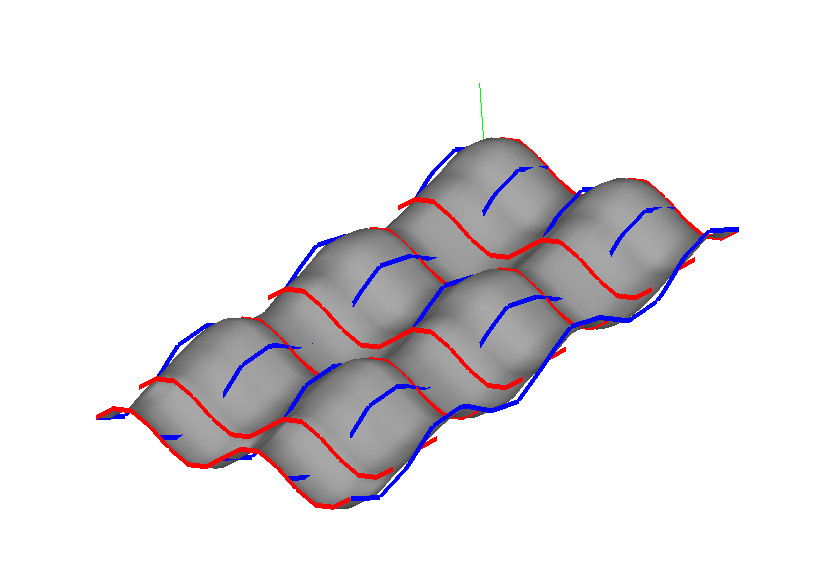
::DeleteItem(arrCurvesU[i]);

for (int i = 0, iLim = arrCurvesV.size(); i < iLim; i++)

::DeleteItem(arrCurvesV[i]);

}

**Результат построения.**



**Код программы.**

void MakeUserCommand2()

{

// Первое семейство координатных кривых:

// - кривые Безье, представляющие два периода синусоиды (растянутые

// на отрезок U\_SIN\_DX).

// - U\_CURVES\_CNT кривых расположены вдоль оси Z на равных промежутках

// в интервале шириной U\_DZ

const int U\_CURVES\_CNT = 10;

const double U\_DZ = 30;

const int U\_SIN\_PERIOD\_CNT = 2;

const double U\_SIN\_DX = 15;

const int U\_BEZIER\_PNT\_CNT = 15;

RPArray<MbCurve3D> arrCurvesU;

for (int i = 0; i < U\_CURVES\_CNT; i++)

{

SArray<MbCartPoint3D> arrPnts;

for (int n = 0; n < U\_BEZIER\_PNT\_CNT; n++)

{

MbCartPoint3D p;

p.x = n \* U\_SIN\_DX / (U\_BEZIER\_PNT\_CNT - 1);

p.y = sin(p.x / U\_SIN\_DX \* 2 \* M\_PI \* U\_SIN\_PERIOD\_CNT);

p.z = i \* U\_DZ / (U\_CURVES\_CNT - 1);

arrPnts.push\_back(p);

}

MbBezier3D\* pc = new MbBezier3D(arrPnts, false);

arrCurvesU.Add(pc);

}

// Второе семейство координатных кривых:

// - кривые Безье, представляющие три периода синусоиды (растянутые

// на отрезок V\_SIN\_DZ).

// - V\_CURVES\_CNT кривых расположены вдоль оси X на равных промежутках

// в интервале шириной V\_DX

const int V\_CURVES\_CNT = 5;

const double V\_DX = U\_SIN\_DX;

const int V\_SIN\_PERIOD\_CNT = 3;

const double V\_SIN\_DZ = U\_DZ;

const int V\_BEZIER\_PNT\_CNT = 15;

RPArray<MbCurve3D> arrCurvesV;

for (int i = 0; i < V\_CURVES\_CNT; i++)

{

SArray<MbCartPoint3D> arrPnts;

for (int n = 0; n < V\_BEZIER\_PNT\_CNT; n++)

{

MbCartPoint3D p;

p.x = i \* V\_DX / (V\_CURVES\_CNT - 1);

p.z = n \* V\_SIN\_DZ / (V\_BEZIER\_PNT\_CNT - 1);

p.y = sin(p.z / V\_SIN\_DZ \* 2 \* M\_PI \* V\_SIN\_PERIOD\_CNT);

arrPnts.push\_back(p);

}

MbBezier3D\* pc = new MbBezier3D(arrPnts, false);

arrCurvesV.Add(pc);

}

// Построение поверхности

MbMeshSurface\* pSurf = new MbMeshSurface(arrCurvesU, arrCurvesV,

false, false, false);

// Отображение поверхности

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

// Отображение координатных кривых

for (int i = 0, iLim = arrCurvesU.size(); i < iLim; i++)

viewManager->AddObject(Style(5, LIGHTRED), arrCurvesU[i]);

for (int i = 0, iLim = arrCurvesV.size(); i < iLim; i++)

viewManager->AddObject(Style(5, LIGHTBLUE), arrCurvesV[i]);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf);

for (int i = 0, iLim = arrCurvesU.size(); i < iLim; i++)

::DeleteItem(arrCurvesU[i]);

for (int i = 0, iLim = arrCurvesV.size(); i < iLim; i++)

::DeleteItem(arrCurvesV[i]);

}

**ПРИМЕР №10**

Задача: реализация 3D модели кинематическая поверхность (колено)

**Поэтапное построение эскиза.**

1. Описание основного кода:

void MakeUserCommand3()

{

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

MbPlacement3D plArc; // СК для построения первой образующей (совпадает с мировой)

MbPlacement3D plCurve; // СК для построения направляющей (вычисляется далее)

// Поворот локальной СК направляющей кривой из плоскости XY в плоскость XZ

// мировой системы координат

plCurve.Rotate(MbAxis3D(MbVector3D(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0))),

90 \* DEG\_TO\_RAD);

// Построение двумерной образующей кривой - окружности

const double RAD = 10;

const MbCartPoint arcCenter(0, 0);

// Построение окружности на плоскости по центру и радиусу

MbArc\* pArc2D = new MbArc(arcCenter, RAD);

// Построение окружности в трехмерном пространстве

MbArc3D\* pArc = new MbArc3D(\*pArc2D, plArc);

// Построение направляющей кривой - дуги окружности

// Сначала строится дуга двумерной окружности по центру окружности, радиусу,

// начальной и конечной точкам

MbArc\* pCurve2D = new MbArc(MbCartPoint(-50, 0), 50, MbCartPoint(0, 0), MbCartPoint(-50, 50), 1);

// Построение дуги трехмерной окружности

MbArc3D\* pCurve = new MbArc3D(\*pCurve2D, plCurve);

// Вызов функции построения кинематической поверхности

MbSurface\* pSurface = NULL;

::EvolutionSurface(\*pArc, \*pCurve, pSurface);

// Отображение построенной поверхности

viewManager->AddObject(Style(1, RGB(0, 0, 255)), pSurface);

// Удаление ссылок на объекты

::DeleteItem(pSurface);

::DeleteItem(pCurve2D);

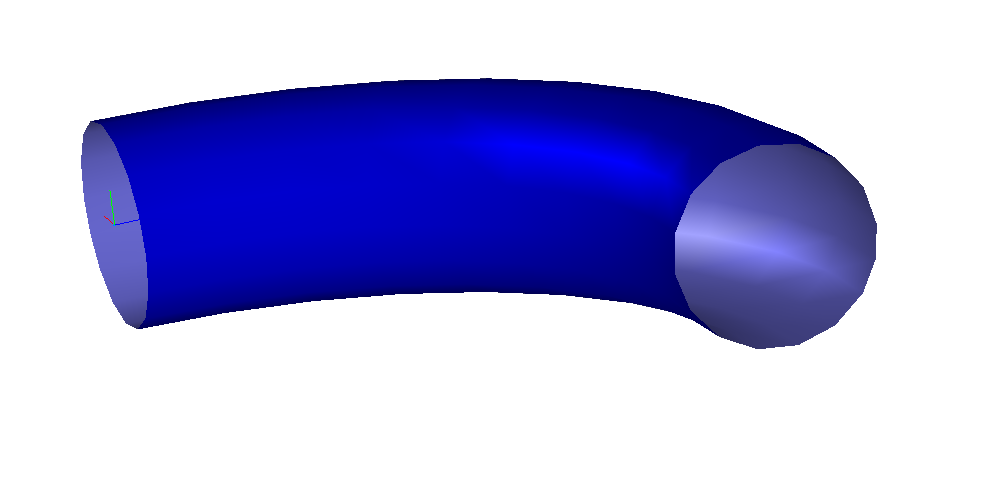
::DeleteItem(pCurve);

::DeleteItem(pArc2D);

::DeleteItem(pArc);

}

**Результат построения.**



**Код программы.**

void MakeUserCommand3()

{

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

MbPlacement3D plArc; // СК для построения первой образующей (совпадает с мировой)

MbPlacement3D plCurve; // СК для построения направляющей (вычисляется далее)

// Поворот локальной СК направляющей кривой из плоскости XY в плоскость XZ

// мировой системы координат

plCurve.Rotate(MbAxis3D(MbVector3D(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0))),

90 \* DEG\_TO\_RAD);

// Построение двумерной образующей кривой - окружности

const double RAD = 10;

const MbCartPoint arcCenter(0, 0);

// Построение окружности на плоскости по центру и радиусу

MbArc\* pArc2D = new MbArc(arcCenter, RAD);

// Построение окружности в трехмерном пространстве

MbArc3D\* pArc = new MbArc3D(\*pArc2D, plArc);

// Построение направляющей кривой - дуги окружности

// Сначала строится дуга двумерной окружности по центру окружности, радиусу,

// начальной и конечной точкам

MbArc\* pCurve2D = new MbArc(MbCartPoint(-50, 0), 50, MbCartPoint(0, 0),

MbCartPoint(-50, 50), 1);

// Построение дуги трехмерной окружности

MbArc3D\* pCurve = new MbArc3D(\*pCurve2D, plCurve);

// Вызов функции построения кинематической поверхности

MbSurface\* pSurface = NULL;

::EvolutionSurface(\*pArc, \*pCurve, pSurface);

// Отображение построенной поверхности

viewManager->AddObject(Style(1, RGB(0, 0, 255)), pSurface);

::DeleteItem(pSurface);

::DeleteItem(pCurve2D);

::DeleteItem(pCurve);

::DeleteItem(pArc2D);

::DeleteItem(pArc);

}

**ПРИМЕР №11**

Задача: реализация 3D Построение изогнутой поверхонсти

**Поэтапное построение эскиза.**

1. Описание основного кода:

void MakeUserCommand5()

{

// Создание границ поверхности - четырех соединяющихся круговых

// дуг pCurve0, pCurve1, pCurve2, pCurve3.

// Дуги строятся по трем точкам (используются начальная, средняя и

// конечная точка дуги).

// При задании точек надо соблюдать направление дуг, чтобы они стыковались

// в одном направлении обхода.

// Дуга в плоскости XY

MbCartPoint3D pc01(0, 0, 0), pc02(5, 3, 0), pc03(10, 0, 0);

MbArc3D\* pCurve0 = new MbArc3D(pc01, pc02, pc03, 1, false);

// Дуга в плоскости ZX, начинающаяся в конечной точке pc01 кривой pCurve0

MbCartPoint3D pc11 = pc03, pc12(7, 0, 7.5), pc13(10, 0, 15);

MbArc3D\* pCurve1 = new MbArc3D(pc11, pc12, pc13, 1, false);

// Дуга в плоскости YX (располагается ниже плоскости ZX), начинающаяся

// в конечной точке pCurve2.

MbCartPoint3D pc21 = pc13, pc22(5, -3, 15), pc23(0, 0, 15);

MbArc3D\* pCurve2 = new MbArc3D(pc21, pc22, pc23, 1, false);

// Дуга в плоскости ZX, соединяющаяся конечную точку pCurve2 с начальной

// точкой pCurve0. (Дуга pCurve3 зеркально симметрична для pCurve1)

MbCartPoint3D pc31 = pc23, pc32(3, 0, 7.5), pc33 = pc01;

MbArc3D\* pCurve3 = new MbArc3D(pc31, pc32, pc33, 1, false);

// Построение поверхности

MbCoverSurface\* pSurf = new MbCoverSurface(\*pCurve0, \*pCurve1, \*pCurve2, \*pCurve3);

// Отображение поверхности

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

// Отображение граничных кривых

viewManager->AddObject(Style(5, LIGHTRED), pCurve0);

viewManager->AddObject(Style(5, LIGHTRED), pCurve1);

viewManager->AddObject(Style(5, LIGHTRED), pCurve2);

viewManager->AddObject(Style(5, LIGHTRED), pCurve3);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf);

::DeleteItem(pCurve0);

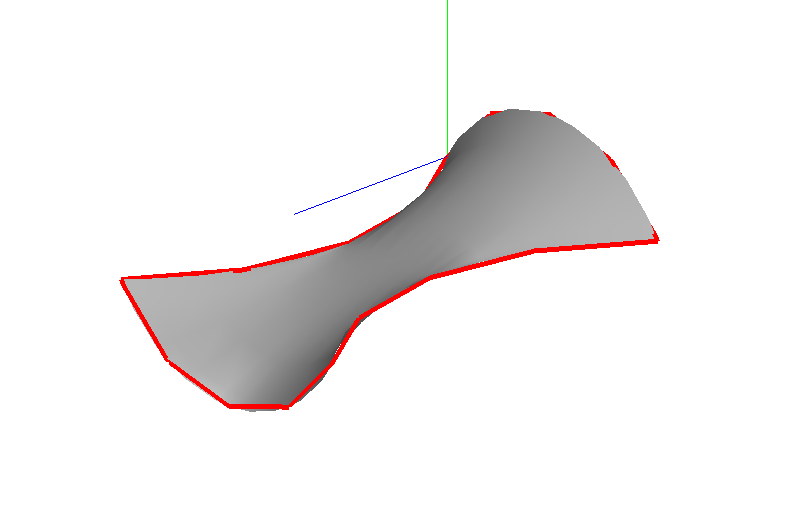
::DeleteItem(pCurve1);

::DeleteItem(pCurve2);

::DeleteItem(pCurve3);

}

**Результат построения.**



**Код программы.**

void MakeUserCommand5()

{

// Создание границ поверхности - четырех соединяющихся круговых

// дуг pCurve0, pCurve1, pCurve2, pCurve3.

// Дуги строятся по трем точкам (используются начальная, средняя и

// конечная точка дуги).

// При задании точек надо соблюдать направление дуг, чтобы они стыковались

// в одном направлении обхода.

// Дуга в плоскости XY

MbCartPoint3D pc01(0, 0, 0), pc02(5, 3, 0), pc03(10, 0, 0);

MbArc3D\* pCurve0 = new MbArc3D(pc01, pc02, pc03, 1, false);

// Дуга в плоскости ZX, начинающаяся в конечной точке pc01 кривой pCurve0

MbCartPoint3D pc11 = pc03, pc12(7, 0, 7.5), pc13(10, 0, 15);

MbArc3D\* pCurve1 = new MbArc3D(pc11, pc12, pc13, 1, false);

// Дуга в плоскости YX (располагается ниже плоскости ZX), начинающаяся

// в конечной точке pCurve2.

MbCartPoint3D pc21 = pc13, pc22(5, -3, 15), pc23(0, 0, 15);

MbArc3D\* pCurve2 = new MbArc3D(pc21, pc22, pc23, 1, false);

// Дуга в плоскости ZX, соединяющаяся конечную точку pCurve2 с начальной

// точкой pCurve0. (Дуга pCurve3 зеркально симметрична для pCurve1)

MbCartPoint3D pc31 = pc23, pc32(3, 0, 7.5), pc33 = pc01;

MbArc3D\* pCurve3 = new MbArc3D(pc31, pc32, pc33, 1, false);

// Построение поверхности

MbCoverSurface\* pSurf = new MbCoverSurface(\*pCurve0, \*pCurve1, \*pCurve2, \*pCurve3);

// Отображение поверхности

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

// Отображение граничных кривых

viewManager->AddObject(Style(5, LIGHTRED), pCurve0);

viewManager->AddObject(Style(5, LIGHTRED), pCurve1);

viewManager->AddObject(Style(5, LIGHTRED), pCurve2);

viewManager->AddObject(Style(5, LIGHTRED), pCurve3);

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pSurf);

::DeleteItem(pCurve0);

::DeleteItem(pCurve1);

::DeleteItem(pCurve2);

::DeleteItem(pCurve3);

}

# 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/.