**ПРИМЕР №3**

Задача: реализация 3D модели болта

**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));

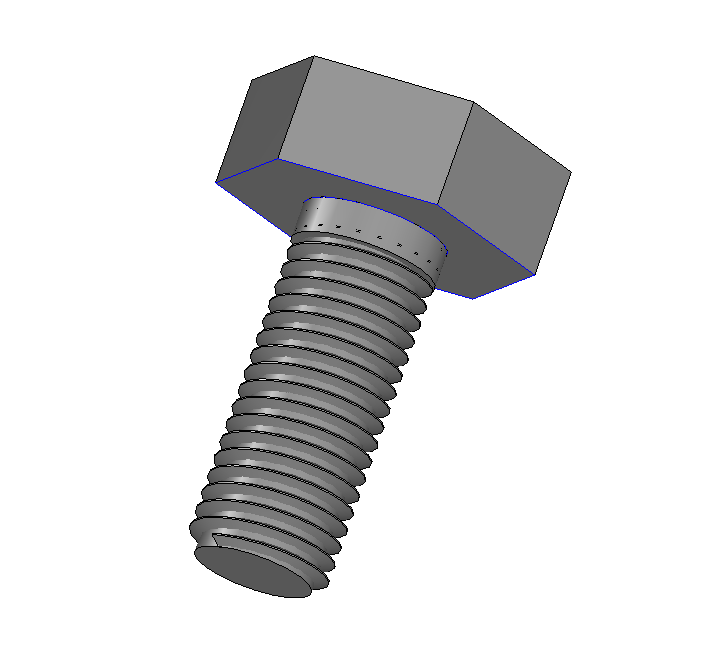
arrPnts.Add(MbCartPoint(70, 40));

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 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);

}