**Функций из лаб. раб.**

1. **Двумерные точки.**

void MakeUserCommand0()

{

// Создание двумерной точки в виде объекта MbCartPoint ядра C3D

MbCartPoint pnt( 1, 0 );

// Вызов метода отображения у объекта тестового приложения

viewManager->AddObject( PPOINT\_Style, pnt );

}

В данном случае добавляем команду пространства имен с переменными и константами:

using namespace c3d;

using namespace TestVariables

1. **Построение двумерной точки в локальной СК(Системе координат)**

void MakeUserCommand0()

{

// Локальная система координат - по умолчанию совпадает с мировой СК

MbPlacement3D pl;

// Создание двумерной точки в виде объекта MbCartPoint ядра C3D

MbCartPoint pnt( 1, 0 );

// Добавление точки в геометрическую модель тестового приложения

viewManager->AddObject( PPOINT\_Style, pnt, &pl );

}

Для того чтобы мы могли использовать класс MbPlacement3D мы добавляем команду

#include "mb\_placement3d.h"

1. **Построение двумерных точек с указанием стиля отображения**

void MakeUserCommand0()

{

// Система координат - по умолчанию совпадает с мировой СК

MbPlacement3D pl;

// Создание двух двумерных точек на осях X и Y

MbCartPoint pntX( 1, 0 ), pntY( 0, 1 );

// Добавление точек в модель c назначением им стилей отображения различными цветами

viewManager->AddObject( Style(5, RGB(255,0,0)), pntX, &pl );

viewManager->AddObject( Style(4, RGB(0,255,0)), pntY, &pl );

}

В данном примере заданы две точки и их стиль отображения, для точки pntX красный цвет и толщина 5 пикселей, для pntY синий цвет и толщина 4 пикселя.

1. **Построение двумерных точек в локальной СК, повернутой относительной мировой СК на 45 градусов во круг оси Z.**

void MakeUserCommand0()

{

// Создание локальной СК, повернутой на 45 градусов относительно оси Z мировой СК

MbPlacement3D pl;

MbAxis3D axisZ( MbVector3D(0, 0, 1) );

pl.Rotate( axisZ, 45\*M\_PI/180 );

// Создание двух двумерных точек на осях X и Y

MbCartPoint pntX( 10, 0 ), pntY( 0, 10 );

// Добавление точек в модель c указанием локальной СК

viewManager->AddObject( Style(5, RGB(255,0,0)), pntX, &pl );

viewManager->AddObject( Style(5, RGB(0,255,0)), pntY, &pl );

}

В данном примере рассматривается новая СК, которая повернута относительно мировой СК по оси Z на 45 градусов. В данном случает за поворот СК отвечает часть кода

MbPlacement3D pl;

MbAxis3D axisZ( MbVector3D(0, 0, 1) );

pl.Rotate( axisZ, 45\*M\_PI/180 );

где axisZ- это наша ось Z, 45\*M\_PI/180 – это на сколько мы будем поворачивать СК.

1. **Построение трехмерных точек**

void MakeUserCommand0()

{

// Создание трех трехмерных точек на осях X, Y и Z

MbCartPoint3D pntX( 1, 0, 0 ), pntY( 0, 1, 0 ), pntZ( 0, 0, 1 );

// Добавление точек в модель c назначением им стилей отображения различными цветами

// Каждая точка добавляется в виде "вырожденного" точечного каркаса, состоящего

// из единственной точки

viewManager->AddObject( Style(5, RGB(255,0,0)), new MbPointFrame(pntX) );

viewManager->AddObject( Style(5, RGB(0,255,0)), new MbPointFrame(pntY) );

viewManager->AddObject( Style(5, RGB(0,0,255)), new MbPointFrame(pntZ) );

}

Для задачи координат точки в трехмерном пространстве необходимо написать следующую строчку кода

MbCartPoint3D pntX( 1, 0, 0 );

1. **Построение набора трехмерных точек в виде одного точечного каркаса.**

void MakeUserCommand0()

{

// Динамическое создание геометрического объекта "Точечный каркас"

MbPointFrame\* pPnts = new MbPointFrame();

// Добавление точек в каркас pPnts - равномерная выборка одного периода синусоиды

const int STEP\_COUNT = 10;

for (int i = 0; i <= STEP\_COUNT; i++)

{

double x = 2\*M\_PI/STEP\_COUNT \* i;

MbCartPoint3D pnt( x, sin(x), 0 );

pPnts->AddVertex( pnt );

}

// Добавление точечного каркаса в геометрическую модель

viewManager->AddObject( Style(5, RGB(255,0,0)), pPnts );

}

Данный кусок кода строит точки по sin(x) через цикл for, с помощью которого строиться 10 точек.

1. **Построение прямой линии в двухмерном пространстве**

void MakeUserCommand0()

{

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

// ПОСТРОЕНИЕ ДВУМЕРНОЙ ПРЯМОЙ – ПО ДВУМ ТОЧКАМ

MbCartPoint p1(0, 0), p2(1, 1); // Пара точек для построения прямой

// Динамическое создание объекта-прямой посредством вызова конструктора MbLine

MbLine\* pLine1 = new MbLine( p1, p2 );

// В случае успешного создания объекта-прямой этот объект передается для сохранения

// в геометрической модели тестового приложения и для последующего отображения

// (будет отображаться красным цветом)

if ( pLine1 != NULL )

viewManager->AddObject( Style( 1, RGB(255,0,0)), pLine1, &pl );

// ПРИМЕЧАНИЕ: удаление pLine1 не производится - за это отвечает

// объект - геометрическая модель внутри тестового приложения

// ПОСТРОЕНИЕ ДВУМЕРНОЙ ПРЯМОЙ – ПО ТОЧКЕ И УГЛУ

// Коэффициент для преобразования значений углов из градусов в радианы

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

// Точка на оси Y

MbCartPoint pnt( 0, 5 );

// Направление (нормированный вектор) 15 градусов относительно оси X

MbDirection dir( 15\*DEG\_TO\_RAD );

// Создание объекта-прямой с указанием точки и угла

MbLine\* pLine2 = new MbLine( pnt, dir );

// Передача объекта-прямой тестовому приложению (будет отображаться синим цветом)

if ( pLine2 )

viewManager->AddObject( Style( 1, RGB(0,0,255)), pLine2, &pl );

}

1. **Построение двумерного отрезка**

#include "cur\_line\_segment.h" // Класс двумерного отрезка MbLineSegment

void MakeUserCommand0()

{

// Локальная СК (расположена так, чтобы ее плоскость XY совпадала с

// плоскостью XZ мировой СК)

MbPlacement3D pl;

pl.SetAxisY( MbVector3D(0, 0, 1) );

// Вершины отрезка

MbCartPoint p1(5, 5), p2(15, 7);

// Динамическое создание объекта-отрезка

MbLineSegment\* pSeg = new MbLineSegment( p1, p2 );

// Сохранение и отображение в тестовом приложении

// LINE\_SEG\_Style - стиль, принятый в тестовом приложении для отображения отрезков

if ( pSeg )

viewManager->AddObject( LINE\_SEG\_Style, pSeg, &pl );

}

1. **Построение прямы линий в трехмерном пространстве**

// Построение пары прямых по двум точкам и проверка их пересечения

void MakeUserCommand0()

{

// Построение трехмерной прямой по двум точкам

MbCartPoint3D p1(0, 5, 0), p2(3, 7, 5);

MbLine3D\* pLine1 = new MbLine3D( p1, p2 );

// Прямая и точки, по которым она построена, отображаются синим цветом

if (pLine1)

{

viewManager->AddObject( Style( 2, BLUE ), pLine1 );

viewManager->AddObject( Style( 1, BLUE ), new MbPointFrame(p1) );

viewManager->AddObject( Style( 1, BLUE ), new MbPointFrame(p2) );

}

// Построение второй прямой по двум точкам

MbCartPoint3D p3(0, 3, 0), p4(3, -7, 5);

MbLine3D\* pLine2 = new MbLine3D( p3, p4 );

// Отображение второй прямой и пары ее точек пурпурным цветом

if ( pLine2 )

{

viewManager->AddObject( Style( 2, LIGHTMAGENTA ), pLine2 );

viewManager->AddObject( Style( 1, LIGHTMAGENTA ), new MbPointFrame(p3) );

viewManager->AddObject( Style( 1, LIGHTMAGENTA ), new MbPointFrame(p4) );

}

// Применение алгоритма для нахождения ближайших точек двух прямых и для

// вычисления расстояния между ними

MbCartPoint3D pntRes1, pntRes2;

double distMin = LineLineNearestPoints( \*pLine1, \*pLine2, pntRes1, pntRes2 );

// Поскольку точность компьютерных вычислений конечна, то для проверки

// пересечения прямых минимальное расстояние следует сравнивать не с

// точным нулевым значением 0.0, а с малой конечной величиной - допуском

if (distMin > 1e-10 )

{

// Отображение красным цветом ближайших точек пары прямых

viewManager->AddObject( Style( 1, LIGHTRED ), new MbPointFrame(pntRes1) );

viewManager->AddObject( Style( 1, LIGHTRED ), new MbPointFrame(pntRes2) );

// Формирование сообщения для информационного окна

TCHAR sBuf[200];

\_stprintf( sBuf, \_T("Расстояние между прямыми: %.3lf"), distMin );

MessageBoxEx( sBuf, \_T("Прямые не пересекаются") );

}

else

{

// Отображение красным цветом точки пересечения прямых

viewManager->AddObject( Style( 1, LIGHTRED ), new MbPointFrame(pntRes1) );

MessageBoxEx( \_T("Прямые пересекаются"), \_T("Проверка пересечения") );

}

}

1. **Построение эллиптических дуг**

void MakeUserCommand0()

{

// Коэффициент для преобразования значений углов из градусов в радианы

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

// Построение окружности в плоскости XY с центром в точке (5, 0) и радиусом 2

MbCartPoint3D pntCenter( 5, 0, 0 );

// Пара точек на окружности

MbCartPoint3D pntOnCircle1( 7, 0, 0 );

MbCartPoint3D pntOnCircle2( 5, 2, 0 );

// Вызываемый конструктор позволяет строить как окружности, так и дуги окружностей

// (в зависимости от передаваемых параметров)

// Четвертый параметр определяет смысл первого параметра – при n=0 его следует

// рассматривать в качестве центра окружности

// Пятый параметр при closed=true задает построение замкнутой окружности (а не дуги)

MbArc3D\* pCircle = new MbArc3D(pntCenter, pntOnCircle1, pntOnCircle2,

0 /\* n \*/, true /\* closed \*/ );

if (pCircle)

viewManager->AddObject( Style( 1, LIGHTBLUE ), pCircle );

// Построение эллипса в плоскости XY с поворотом осей на 45 градусов

// pl - локальная система координат, в которой эллипс имеет каноническую форму

// (в ней оси эллипса параллельны осям координат)

MbPlacement3D pl;

pl.Rotate( MbAxis3D(pl.GetAxisZ()), 45\*DEG\_TO\_RAD );

// Построение эллипса с указанием локальной СК и величин полуосей

// Последний параметр задает конец дуги в случае построения эллиптической дуги,

// для замкнутого эллипса должно быть angle=0

MbArc3D\* pEllipse = new MbArc3D( pl, 10, 5, 0 /\* angle \*/ );

if (pEllipse)

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

// Построение четверти эллиптической дуги для уже построенного эллипса.

// Начало и конец дуги расположены под углами 60 и 150 градусов относительно

// большой полуоси эллипса.

// Последний параметр initSense=1 задает направление движения против часовой стрелки

// от начальной до конечной точки дуги.

MbArc3D\* pQuarterArc = new MbArc3D( \*pEllipse, 60\*DEG\_TO\_RAD,

150\*DEG\_TO\_RAD, 1 /\* initSense \*/ );

if (pQuarterArc)

viewManager->AddObject( Style( 2, LIGHTMAGENTA ), pQuarterArc );

}

1. **Построение элементарных поверхностей**

#include "mb\_axis3d.h" // Ось в пространстве MbAxis3D

#include "mb\_placement3d.h" // Система координат MbPlacement3D

#include "point\_frame.h" // Точечный каркас MbPointFrame

#include "cur\_line\_segment3d.h" // Отрезок MbLineSegment3D

#include "surf\_plane.h" // Плоскость MbPlane

#include "surf\_sphere\_surface.h" // Сферическая поверхность MbSphereSurface

#include "surf\_cylinder\_surface.h" // Цилиндрическая поверхность MbCylinderSurface

#include "surf\_cone\_surface.h" // Коническая поверхность MbConeSurface

#include "action\_surface.h" // Функции построения поверхностей

void MakeUserCommand0()

{

// Коэффициент для преобразования значений углов из градусов в радианы

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

// Построение плоскости, совпадающей с координатной плоскостью XZ.

// Производится вызов конструктора плоскости по трем точкам.

// В данном случае могут быть переданы произвольные точки, лежащие в плоскости XZ.

// Значения координат влияют на размеры плоскости при отображении в

// тестовом приложении.

MbCartPoint3D planePnt1( 0, 0, 0 );

MbCartPoint3D planePnt2( 50, 0, 0 );

MbCartPoint3D planePnt3( 0, 0, 50 );

MbPlane\* pPlane = new MbPlane( planePnt1, planePnt2, planePnt3 );

// Отображение плоскости синим цветом

if ( pPlane )

viewManager->AddObject( Style( 1, LIGHTBLUE ), pPlane );

// Далее производится построение элементарных поверхностей, касающихся плоскости XZ

// и лежащих в ее положительном полупространстве.

// Построение сферы посредством явного вызова конструктора с указанием центра и

// радиуса сферической поверхности

double radius\_Sph1 = 10;

MbCartPoint3D centerPnt\_Sph1( -radius\_Sph1\*3, radius\_Sph1, -radius\_Sph1\*3 );

MbSphereSurface\* pSphere1 = new MbSphereSurface( centerPnt\_Sph1, radius\_Sph1 );

if ( pSphere1 )

viewManager->AddObject( Style( 1, LIGHTRED ), pSphere1 );

// Построение сферы вызовом функции ядра для создания элементарной поверхности

// Назначение точек-параметров:

// 1) начало локальной СК поверхности;

// 2) точка, определяющая направление оси Х локальной СК и радиус поверхности;

// 3) точка, определяющая направление оси Y локальной СК.

// Параметры задаются так, чтобы центры двух сфер располагались по разные стороны и

// на равном расстоянии от оси OY мировой СК

MbSurface\* pSphere2 = NULL;

double radius\_Sph2 = 7;

MbCartPoint3D centerPnt\_Sph2( -centerPnt\_Sph1.x, radius\_Sph2, -centerPnt\_Sph1.z );

MbCartPoint3D pntOX\_Sph2 = centerPnt\_Sph2 + MbVector3D( radius\_Sph2, 0, 0 );

MbCartPoint3D pntOY\_Sph2 = centerPnt\_Sph2 + MbVector3D( 0, radius\_Sph2, 0 );

MbResultType resSph2 = ::ElementarySurface( centerPnt\_Sph2, pntOX\_Sph2,

pntOY\_Sph2, st\_SphereSurface, pSphere2 );

if ( resSph2 == rt\_Success )

viewManager->AddObject( Style( 1, LIGHTGREEN ), pSphere2 );

// Построение первой цилиндрической поверхности

// Вызов конструктора с указанием центра локальной СК, радиуса основания и высоты.

// Ось цилиндра ориентируется вдоль оси Z локальной СК

double radius\_Cyl1 = 4;

double height\_Cyl1 = 15;

MbPlacement3D plCyl1;

// Локальная СК цилиндра: мировая СК смещается на заданный вектор и поворачивается

// так, чтобы ось Z локальной СК была параллельна оси OY мировой СК

plCyl1.Rotate(MbAxis3D(MbVector3D(1,0,0)), -90\*DEG\_TO\_RAD );

plCyl1.Move( MbVector3D( radius\_Cyl1\*5, 0, -radius\_Cyl1\*5 ) );

MbCylinderSurface\* pCyl1 = new MbCylinderSurface( plCyl1, radius\_Cyl1, height\_Cyl1 );

if ( pCyl1 )

viewManager->AddObject( Style( 1, LIGHTRED ), pCyl1 );

// Построение второй цилиндрической поверхности

// Вызов функции ядра для создания элементарной поверхности

MbSurface\* pCyl2 = NULL;

double radius\_Cyl2 = 2.5;

double height\_Cyl2 = 10;

// Центр нижнего основания цилиндра (начало координат локальной СК) получаем

// отражением центра основания первого цилиндра относительно оси OY мировой СК

MbCartPoint3D centerPnt\_Cyl2 = plCyl1.GetOrigin();

centerPnt\_Cyl2.x = -centerPnt\_Cyl2.x;

centerPnt\_Cyl2.z = -centerPnt\_Cyl2.z;

// Точка, задающая ось X локальной СК и определяющая высоту цилиндра

// В данном случае ось X локальной СК параллельна оси Y мировой СК

MbCartPoint3D pntOX\_Cyl2 = centerPnt\_Cyl2 + MbVector3D( 0, height\_Cyl2, 0 );

// Точка, задающая ось Y локальной СК и определяющая радиус основания

// В данном случае ось Y локальной СК параллельна оси Z мировой СК

MbCartPoint3D pntOY\_Cyl2 = centerPnt\_Cyl2 + MbVector3D( 0, 0, radius\_Cyl2 );

MbResultType resCyl2 = ::ElementarySurface( centerPnt\_Cyl2, pntOX\_Cyl2,

pntOY\_Cyl2, st\_CylinderSurface, pCyl2 );

if ( resCyl2 == rt\_Success )

viewManager->AddObject( Style( 1, LIGHTGREEN ), pCyl2 );

// Построение первой конической поверхности

// Эта поверхность касается плоскости XZ мировой СК в одной точке

// Вызывается конструктор конической поверхности по трем точкам.

double height\_Cone1 = 15;

double radius\_Cone1 = 7.5;

// Вершина конической поверхности

MbCartPoint3D pntCone1\_origin( 10, 0, 0 );

// Центр основания

MbCartPoint3D pntCone1\_base = pntCone1\_origin + MbVector3D(0, height\_Cone1, 0 );

// Точка на боковой поверхности в плоскости основания

MbCartPoint3D pntCone1\_surf = pntCone1\_base + MbVector3D(radius\_Cone1, 0, 0 );

MbConeSurface\* pCone1 = new MbConeSurface( pntCone1\_origin, pntCone1\_base,

pntCone1\_surf );

if (pCone1)

viewManager->AddObject( Style( 1, LIGHTRED ), pCone1 );

// Построение второй конической поверхности

// Основание этой поверхности лежит на плоскости XZ мировой СК

double height\_Cone2 = 8;

double radius\_Cone2 = 4;

// Вершина конической поверхности

MbCartPoint3D pntCone2\_origin( -pntCone1\_origin.x, height\_Cone2, 0 );

// Центр основания

MbCartPoint3D pntCone2\_base = pntCone2\_origin + MbVector3D(0, -height\_Cone2, 0 );

// Точка на боковой поверхности в плоскости основания

MbCartPoint3D pntCone2\_surf = pntCone2\_base + MbVector3D(radius\_Cone2, 0, 0 );

MbSurface\* pCone2 = NULL;

MbResultType resCone2 = ::ElementarySurface( pntCone2\_origin, pntCone2\_base,

pntCone2\_surf, st\_ConeSurface, pCone2 );

if ( resCone2 == rt\_Success )

viewManager->AddObject( Style( 1, LIGHTGREEN ), pCone2 );

// Уменьшение счетчика ссылок на динамически созданные объекты.

// При достижении счетчиком 0 объект будет удален вызовом деструктора.

::DeleteItem( pPlane );

::DeleteItem( pSphere1 );

::DeleteItem( pSphere2 );

::DeleteItem( pCyl1 );

::DeleteItem( pCyl2 );

::DeleteItem( pCone1 );

::DeleteItem( pCone2 );

}

1. **Построение элементарных твердых тел – цилиндр и пирамида**

#include "surface.h" // Поверхность MbSurface

#include "solid.h" // Твердое тело MbSolid

#include "action\_surface.h" // Функции построения поверхностей

#include "action\_solid.h" // Функции построения твердых тел

#include "creator.h" // Построитель элементов твердотельных моделей

void MakeUserCommand0()

{

// Построение цилиндрической поверхности

double height\_Cyl = 10;

double radius\_Cyl = 3;

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 );

}

// Построение тела по характерным точкам (четырехгранная равносторонняя пирамида)

MbSolid\* pPyr = NULL;

// Массив характерных точек – 4 вершины основания + вершина пирамиды

SArray<MbCartPoint3D> arrPnts(5);

arrPnts.Add(MbCartPoint3D( -5, 0, -5 ));

arrPnts.Add(MbCartPoint3D( 5, 0, -5 ));

arrPnts.Add(MbCartPoint3D( 5, 0, 5 ));

arrPnts.Add(MbCartPoint3D( -5, 0, 5 ));

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

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

// Вызов функции ядра для построения тела типа et\_Pyramid по массиву точек

MbResultType resPyr = ::ElementarySolid( arrPnts, et\_Pyramid, namesPyr, pPyr );

if (resPyr == rt\_Success)

{

// Смещение твердого тела на заданный вектор

pPyr->Move( MbVector3D( 15, 0, 0 ) );

viewManager->AddObject( Style( 1, LIGHTBLUE ), pPyr );

}

// Уменьшение счетчиков ссылок на динамически созданные объекты

::DeleteItem( pCylSurf );

::DeleteItem( pCyl );

::DeleteItem( pPyr );

}

1. **Построение прямой линии в двумерном пространстве**

#include "cur\_line.h" // MbLine - Прямая в двумерном пространстве

void MakeUserCommand0()

{

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

// Коэффициент для преобразования значений углов из градусов в радианы

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

// ПОСТРОЕНИЕ ДВУМЕРНОЙ ПРЯМОЙ – ПО ТОЧКЕ И НАПРАВЛЯЮЩЕМУ ВЕКТОРУ

MbCartPoint p1(0, 0);

MbDirection dir( DEG\_TO\_RAD \* 45 );

// Динамическое создание объекта-прямой посредством вызова конструктора MbLine

MbLine\* pLine = new MbLine( p1, dir);

// Отображение прямой

if ( pLine != NULL )

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

// Получение информации о типе геометрического объекта

MbePlaneType type\_obj = pLine->IsA(); // pt\_Line

// Код группового типа, обозначающий родительский класс

MbePlaneType type\_parent = pLine->Type(); // pt\_Curve

// Код семейства, обозначающий базовый класс, унаследованный от MbPlaneItem

MbePlaneType type\_family = pLine->Family(); // pt\_Curve

// Получение характеристик области определения параметра t

// Получение предельных значений параметра кривой

double tMin = pLine->GetTMin(); // -50000000

double tMax = pLine->GetTMax(); // 50000000

// Проверка, является ли кривая замкнутой

bool bClosed = pLine->IsClosed(); // false

// Проверка, является ли замкнутая кривая периодической

bool bIsPeriodic = pLine->IsPeriodic(); // false

// Получение периода для периодической кривой

double tPeriod = pLine->GetPeriod(); // 0.0

// Вычислить метрическую длину кривой

double length = 0.0;

bool bHasLength = pLine->HasLength( length ); // false

// Проверка, является ли кривая ограниченной

bool bIsBounded = pLine->IsBounded(); // false

// Проверка, является ли кривая прямолинейной

bool bIsStraight = pLine->IsStraight(); // true

// Уменьшение счетчика ссылок динамического объекта-прямой

::DeleteItem( pLine );

}

1. **Построение концентрических окр и дуг окр**

#include "cur\_arc.h" // MbArc - Эллиптическая дуга в двумерном пространстве

#include <vector> // std::vector - динамический массив из стандартной библиотеки C++

void MakeUserCommand0()

{

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

// Центр окружностей и дуг в локальной СК

MbCartPoint cnt( 5, 5 );

// Построение окружности по центру и радиусу

const double RAD1 = 16;

MbArc\* pc1 = new MbArc(cnt, RAD1);

// Окружность по центру и по точке на окружности

const double RAD2 = 14;

MbArc\* pc2 = new MbArc( cnt, MbCartPoint(cnt.x + RAD2, cnt.y) );

// Дуга окружности: задается центр, радиус, точки-концы дуги и направление

// обхода от первой до второй концевой точки (-1 - по часовой стрелке, +1 - против).

// Координаты точек-концов дуги вычисляются с использованием координатных функций:

// x(t) = R\*cos(t), y(t) = R\*sin(t)

// Радиус-векторы концов дуги выбраны расположенными под углом 30 и 60 градусов к

// горизонтальной оси.

// Дуга pc3 строится по часовой стрелке, дуга pc4 - против

const double RAD3 = 12;

MbCartPoint p31(cnt.x + RAD3\*cos(M\_PI/6), cnt.y + RAD3\*sin(M\_PI/6) );

MbCartPoint p32(cnt.x + RAD3\*cos(M\_PI/3), cnt.y + RAD3\*sin(M\_PI/3) );

MbArc\* pc3 = new MbArc( cnt, RAD3, p31, p32, -1 );

const double RAD4 = 10;

MbCartPoint p41(cnt.x + RAD4\*cos(M\_PI/6), cnt.y + RAD4\*sin(M\_PI/6) );

MbCartPoint p42(cnt.x + RAD4\*cos(M\_PI/3), cnt.y + RAD4\*sin(M\_PI/3) );

MbArc\* pc4 = new MbArc( cnt, RAD4, p41, p42, 1 );

// Дуга окружности: задается центр, радиус, параметры точек-концов дуги и

// направление обхода дуги: -1 - по часовой стрелке, +1 - против

const double RAD5 = 8;

MbArc\* pc5 = new MbArc( cnt, RAD5, M\_PI/2, 3\*M\_PI/2, -1 );

const double RAD6 = 6;

MbArc\* pc6 = new MbArc( cnt, RAD6, M\_PI/2, 3\*M\_PI/2, 1 );

// Дуга окружности: по трем точкам.

// Координаты точек p71, p72 и p73, принадлежащих дуге окружностей,

// вычисляются в локальной СК окружности.

const double RAD7 = 4;

MbCartPoint p71( RAD7\*cos(M\_PI/6), RAD7\*sin(M\_PI/6) );

MbCartPoint p72( RAD7\*cos(M\_PI/2), RAD7\*sin(M\_PI/2) );

MbCartPoint p73( RAD7\*cos(3\*M\_PI/2), RAD7\*sin(3\*M\_PI/2) );

// Смещение координат точек для перемещения центра СК окружности в точку cnt

p71.Move( cnt );

p72.Move( cnt );

p73.Move( cnt );

MbArc\* pc7 = new MbArc( p71, p72, p73 );

// Помещение указателей на окружности в массив для отображения с помощью цикла

std::vector<MbArc\*> arrArcs = { pc1, pc2, pc3, pc4, pc5, pc6, pc7 };

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

{

viewManager->AddObject( ARC\_Style, arrArcs[i], &pl );

::DeleteItem( arrArcs[i] );

}

}

1. **Построение окр с использованием координатных функций в символьном виде**

// MbCharacterCurve - кривая, заданная в символическом виде

#include "cur\_character\_curve.h"

#include "function.h" // MbFunction - скалярная функция одной переменной

#include "function\_factory.h" // MbFunctionFactory – генератор объектов-функций

void MakeUserCommand0()

{

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

// Объект – генератор функций (осуществляет синтаксический разбор выражения

// функции и готовит служебные структуры данных для вычисления значений функции).

MbFunctionFactory ff;

// Координатные функции в символическом виде для окружности

// с центром в начале координат и радиусом 10

c3d::string\_t sXFunc = \_T("10\*cos(t)");

c3d::string\_t sYFunc = \_T("10\*sin(t)");

// Предельные значения параметра t

const double T\_MIN = 0.0, T\_MAX = 2\*M\_PI;

// Координатные функции в виде объектов MbFunction

MbFunction\* pXFunc = ff.CreateAnalyticalFunction( sXFunc, \_T("t"), T\_MIN, T\_MAX );

MbFunction\* pYFunc = ff.CreateAnalyticalFunction( sYFunc, \_T("t"), T\_MIN, T\_MAX );

if ( pXFunc != NULL && pYFunc != NULL )

{

// Локальная двумерная СК кривой - совпадает с XOY трехмерной системы pl

MbPlacement plCurve( MbCartPoint(0,0), MbDirection(0.0) );

// Объект-кривая

MbCharacterCurve\* pCurve = new MbCharacterCurve( \*pXFunc, \*pYFunc, ls\_CartesSystem,

plCurve, T\_MIN, T\_MAX );

// Отображение кривой

if ( pCurve )

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

::DeleteItem( pCurve );

}

// Удаление динамически созданных объектов-функций

::DeleteItem( pXFunc );

::DeleteItem( pYFunc );

}

1. **Построение многоугольника, вписанного в окр**

#include "cur\_polyline.h" // MbPolyline - ломаная линия

void MakeUserCommand0()

{

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

// Количество сторон многоугольника

const int SIDE\_CNT = 5;

// Радиус описывающей окружности многоугольника

const double RAD = 5.0;

// 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( RAD\*cos(angle), RAD\*sin(angle) );

arrPnts.Add( pnt );

}

// Замкнутая ломаная линия с вершинами arrPnts

MbPolyline\* pPolyline = new MbPolyline( arrPnts, true /\* Флаг замкнутой линии \*/ );

// Отображение ломаной

if ( pPolyline )

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

::DeleteItem( pPolyline );

}

**17.Построение контура**

#include "cur\_contour.h" // MbContour - контур в двумерном пространстве

#include "cur\_polyline.h" // MbPolyline - Ломаная линия

#include "cur\_arc.h" // MbArc - Эллиптическая дуга

void MakeUserCommand0()

{

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

// Вершины ломаной

SArray<MbCartPoint> arrPnts(8);

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) );

// Ломаная линия с вершинами arrPnts

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

// 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).

MbCartPoint arcCenter( 50, 40 );

const double RADIUS = 20;

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

// Контур из двух сегментов

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

pContour->AddSegment( pArc );

// Отображение контура

if (pContour )

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

// Вызовы информационных методов контура

size\_t segmentsCount = pContour->GetSegmentsCount(); // 2

double tMin = pContour->GetTMin(); // 0.0

double tMax = pContour->GetTMax(); // 10.14159

// Параметр IsSmoothConnected - допуск для классификации угловых точек на контуре

bool isSmooth = pContour->IsSmoothConnected(0.001); // false

double length = pContour->CalculateLength( tMin, tMax ); // 302.83

bool isClosed = pContour->IsClosedContinuousC0(); // true

bool isCurvilinear = pContour->IsAnyCurvilinear(); // true

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pPolyline );

::DeleteItem( pArc );

::DeleteItem( pContour );

}

**18.Выдавливание контура**

// Заголовочные файлы, дополнительные к примеру 2.1

#include "surf\_plane.h" // MbPlane - плоскость

#include "cur\_contour\_on\_plane.h" // MbContourOnPlane - Контур на плоскости

// (кривая в трехмерном пространстве)

#include "action\_surface.h" // Функции- алгоритмы построения поверхностей

void MakeUserCommand0()

{

// ... сюда следует поместить часть примера 2.1 до первого вызова ::DeleteItem

// Из скопированной части удалите фрагмент, связанный с отображением геометрических

// объектов (вызов viewManager->AddObject() и предшествующую проверку if).

// ПОСТРОЕНИЕ ПОВЕРХНОСТИ ВЫДАВЛИВАНИЯ

// Контур-образующую разместим в плоскости ZY глобальной системы координат.

// Эта плоскость представляется в виде объекта MbPlane и строится по точке-началу

// координат и двум векторам, задающим оси локальной системы координат

// плоскости – в данном случае это оси Z и Y глобальной СК.

MbPlane\* pPlaneZY = new MbPlane( MbCartPoint3D(0,0,0), MbVector3D(0,0,1),

MbVector3D(0,1,0) );

// Трехмерная кривая "контур на плоскости", которая будет использоваться в качестве

// образующей для операции выдавливания

MbContourOnPlane\* pc = new MbContourOnPlane( \*pPlaneZY, \*pContour, true );

// Вектор, задающий направление и величину выдавливания (на 15 единиц в

// отрицательном направлении оси X глобальной СК)

MbVector3D dir( -15, 0, 0 );

// Вызов функции для построения поверхности выдавливания

MbSurface\* pSurf = NULL;

MbResultType res = ::ExtrusionSurface(

\*pc, dir, true /\* разрешение на упрощение поверхности \*/, pSurf );

// Отображение построенной поверхности

if ( res == rt\_Success )

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pPlaneZY );

::DeleteItem( pc );

::DeleteItem( pSurf );

::DeleteItem( pPolyline );

::DeleteItem( pArc );

::DeleteItem( pContour );

}

**19.Построение твердого тела**

// Заголовочные файлы, дополнительные к примеру 2.1

#include "solid.h" // MbSolid - твердое тело

#include "creator.h" // Классы для построения геометрических моделей

#include "action\_solid.h" // Функции-алгоритмы построения твердых тел

void MakeUserCommand0()

{

// ... сюда следует поместить часть примера 2.1 до первого вызова ::DeleteItem

// Из скопированной части удалите фрагмент, связанный с отображением геометрических

// объектов (вызов viewManager->AddObject() и предшествующую проверку if).

// ПОСТРОЕНИЕ ТЕЛА ВЫДАВЛИВАНИЯ

// Локальная СК в трехмерном пространстве, у которой ось X совпадает с

// осью Z глобальной СК. Строится по двум осям X и Y и началу координат.

MbPlacement3D plZY( MbVector3D(0,0,1), MbVector3D(0,1,0), MbCartPoint3D(0,0,0) );

// Объект, хранящий параметры образующей (контур и локальная СК).

// Контур размещается на плоскость XY локальной СК plZY.

MbSweptData sweptData( plZY, \*pContour );

// Направляющий вектор для операции выдавливания

MbVector3D dirV( -1, 0, 0 );

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

// расстояние выдавливания в прямом и в обратном направлении вдоль

// направляющего вектора

const double HEIGHT\_FORWARD = 10.0, 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 resCode = ::ExtrusionSolid( sweptData, dirV, NULL, NULL, false,

extrusionParam, operNames, cNames, pSolid);

// Отображение построенного тела

if ( resCode == rt\_Success )

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pPolyline );

::DeleteItem( pArc );

::DeleteItem( pContour );

::DeleteItem( pSolid );

}

**20.Построение скруглений и фасок**

#include "cur\_polyline.h" // MbPolyline - ломаная линия

#include "cur\_contour.h" // MbContour - контур

#include "alg\_curve\_fillet.h" // Функции для построения скруглений и фасок

void MakeUserCommand0()

{

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

// Ломаная линия, изображающая квадрат.

// Вершины перечисляются по часовой стрелке, начиная с левого нижнего угла.

const double SQUARE\_SIDE = 50;

SArray<MbCartPoint> arrPnts(4);

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

arrPnts.Add( MbCartPoint(0, SQUARE\_SIDE) );

arrPnts.Add( MbCartPoint(SQUARE\_SIDE, SQUARE\_SIDE) );

arrPnts.Add( MbCartPoint(SQUARE\_SIDE, 0 ) );

MbPolyline\* pSquare = new MbPolyline( arrPnts, true /\* Флаг замкнутой линии \*/);

// Построение нового контура со скруглениями на основе ломаной-квадрата.

// В качестве входных параметров указывается радиус скругления,

// флаг построения только в одной вершине, и точка, задающая вершину для скругления.

// Функция возвращает новую кривую-контур со скруглением в виде

// выходного параметра pFillets.

MbContour\* pFillets = NULL;

::FilletPolyContour(pSquare, 5, false, arrPnts[1], pFillets );

// Построение еще трех скруглений с помощью методов MbContour.

// Указывается индекс первого из скругляемых сегментов и

// радиус скругления. При указании индексов учитывается, что

// каждое скругление добавляет один сегмент в контур.

// После вызова метода для построения скругления значения индексов

// изменяются и соответствуют построенным сегментам-дугам.

ptrdiff\_t idxSideTop = 2; // Индекс верхней стороны квадрата

ptrdiff\_t idxSideRight = 4; // Индекс правой стороны

ptrdiff\_t idxSideBottom = 6; // Индекс нижней стороны

pFillets->FilletTwoSegments( idxSideTop, 10 /\* радиус скругления \*/ );

pFillets->FilletTwoSegments( idxSideRight, 15 );

pFillets->FilletTwoSegments( idxSideBottom, 20 );

// Сдвиг контура по горизонтали, чтобы он не накладывался на исходный квадрат

pFillets->Move( MbVector( SQUARE\_SIDE\*3/2, 0 ) );

// Создание копии ломаной-квадрата для построения ломаной с фасками

MbPolyline\* pChamfers = new MbPolyline( \*pSquare );

// Сдвиг ломаной, чтобы она не накладывалась на уже построенные кривые.

pChamfers->Move( MbVector( 2 \* SQUARE\_SIDE\*3/2, 0 ) );

// Длины фасок (начиная с левого нижнего угла квадрата)

const double arrLen[] = { 20, 5, 10, 15 };

// Построение фасок в углах ломаной, начиная с левого нижнего угла

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

for (int i = 0; i < 4; i++)

{

// Индекс очередной вершины, в которой строится фаска. Множитель 2 нужен для

// учета того, что каждая фаска добавляет один сегмент к ломаной.

int idxPnt = i\*2;

// Координаты очередной вершины квадрата.

MbCartPoint pntVert;

pChamfers->GetPoint( idxPnt, pntVert );

// Функция построения фаски модифицирует исходную кривую посредством

// добавления отрезка длиной arrLen[i] с направлением 45 градусов.

::ChamferPolyContour(pChamfers, arrLen[i], 45\*DEG\_TO\_RAD, true, false, pntVert );

}

// Отображение ломаных и контура

if ( pSquare )

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

if ( pFillets )

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

if ( pChamfers )

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

// Пример запроса количества сегментов у построенных кривых

int segCount\_Square = pSquare->GetSegmentsCount(); // 4

int segCount\_Fillets = pFillets->GetSegmentsCount(); // 8

int segCount\_Chamfers = pChamfers->GetSegmentsCount(); // 8

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pSquare );

::DeleteItem( pFillets );

::DeleteItem( pChamfers );

}

**21. Использование контура со скруглениями в качестве образующей тела выдавливания.**

#include "cur\_contour.h" // MbContour - контур

#include "cur\_arc.h" // MbArc - дуга эллипса/окружности

#include "cur\_line.h" // MbLine - прямая в двумерном пространстве

#include "cur\_line\_segment.h" // MbLineSegment - класс двумерного отрезка

#include "mb\_cross\_point.h" // MbCrossPoint - точка пересечения пары кривых

#include "surf\_plane.h" // MbPlane - плоскость

#include "solid.h" // MbSolid - твердое тело

#include "creator.h" // Классы для построения геометрических моделей

#include "alg\_curve\_fillet.h" // Функции для построения скруглений и фасок

#include "action\_point.h" // Функции-алгоритмы для операций с точками

#include "action\_solid.h" // Функции- алгоритмы построения твердых тел

// ФУНКЦИЯ ПОСТРОЕНИЯ КРИВЫХ ДЛЯ ОБРАЗУЮЩЕЙ ТЕЛА ВЫДАВЛИВАНИЯ.

// Три кривые возвращаются в массиве в виде указателей на контуры.

void CreateSketch( RPArray<MbContour>& \_arrContours )

{

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

const MbCartPoint centerCircleLeft( 0, 0 );

const MbCartPoint centerCircleRight( 100, 0 );

const double RAD\_LEFT = 25;

const double RAD\_RIGHT = 15;

// Вспомогательные горизонтальные прямые, содержащие отрезки контура

// Значения горизонтальных координат могут быть любыми.

MbLine\* pLineTop = new MbLine(MbCartPoint( 0, 7.5), MbCartPoint( 10, 7.5 ) );

MbLine\* pLineBtm = new MbLine(MbCartPoint( 0, -7.5), MbCartPoint( 10, -7.5 ) );

// Вычисление точек пересечения вспомогательных прямых с окружностями (рис.6)

MbCrossPoint pntsEA[2]; // Пересечения верхней прямой и левой окружности

MbCrossPoint pntsDG[2]; // Верхняя прямая и правая окружность

MbCrossPoint pntsFB[2]; // Нижняя прямая и левая окружность

MbCrossPoint pntsCH[2]; // Нижняя прямая и правая окружность

::LineCircle( \*pLineTop, centerCircleLeft, RAD\_LEFT, pntsEA );

::LineCircle( \*pLineTop, centerCircleRight, RAD\_RIGHT, pntsDG );

::LineCircle( \*pLineBtm, centerCircleLeft, RAD\_LEFT, pntsFB );

::LineCircle( \*pLineBtm, centerCircleRight, RAD\_RIGHT, pntsCH );

// Упорядочение точек пересечения по горизонтали, чтобы было pnts[0].x < pnts[1].x

if (pntsEA[0].p.x > pntsEA[1].p.x )

std::swap(pntsEA[0], pntsEA[1] );

if (pntsDG[0].p.x > pntsDG[1].p.x )

std::swap(pntsDG[0], pntsDG[1] );

if (pntsFB[0].p.x > pntsFB[1].p.x )

std::swap(pntsFB[0], pntsFB[1] );

if (pntsCH[0].p.x > pntsCH[1].p.x )

std::swap(pntsCH[0], pntsCH[1] );

// Дуги окружностей для контура

MbArc\* pArcAB = new MbArc(centerCircleLeft, RAD\_LEFT, pntsEA[1].p, pntsFB[1].p, 1);

MbArc\* pArcCD = new MbArc(centerCircleRight, RAD\_RIGHT, pntsCH[0].p, pntsDG[0].p, 1);

// Отрезки для контура

MbLineSegment\* pSegBC = new MbLineSegment(pntsFB[1].p, pntsCH[0].p);

MbLineSegment\* pSegDA = new MbLineSegment(pntsDG[0].p, pntsEA[1].p );

// Построение контура

MbContour\* pContour = new MbContour();

pContour->AddSegment( pArcAB );

pContour->AddSegment( pSegBC );

pContour->AddSegment( pArcCD );

pContour->AddSegment( pSegDA );

// Построение скруглений во всех узлах контура (содержит четыре узла)

pContour->Fillet( 10 );

// Построение внутренних окружностей образующей и преобразование их в контуры

const double INNER\_RAD\_LEFT = 12.5;

const double INNER\_RAD\_RIGHT = 7.5;

MbArc\* pCircleLeft = new MbArc(centerCircleLeft, INNER\_RAD\_LEFT);

MbArc\* pCircleRight = new MbArc(centerCircleRight, INNER\_RAD\_RIGHT);

MbContour\* pContourCircleLeft = new MbContour( \*pCircleLeft, true );

MbContour\* pContourCircleRight = new MbContour( \*pCircleRight, true );

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pLineTop );

::DeleteItem( pLineBtm );

::DeleteItem( pArcAB );

::DeleteItem( pArcCD );

::DeleteItem( pSegBC );

::DeleteItem( pSegDA );

// Сохранение указателей на возвращаемые контуры в выходном массиве

// У возвращаемых объектов счетчик ссылок уменьшать не надо - это должно быть

// сделано в месте вызова, после того, как эти объекты больше не будут нужны.

\_arrContours.push\_back( pContour );

\_arrContours.push\_back( pContourCircleLeft );

\_arrContours.push\_back( pContourCircleRight );

}

// ОБРАБОТЧИК ПОЛЬЗОВАТЕЛЬСКИЙ КОМАНДЫ ТЕСТОВОГО ПРИЛОЖЕНИЯ

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 = 10.0, 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] );

}

**22. Построение кубического сплайна и ломаной**

#include "cur\_polyline.h" // MbPolyline - Ломаная линия

#include "cur\_cubic\_spline.h" // MbCubicSpline - кубический сплайн в 2D

void MakeUserCommand0()

{

// Контрольные точки сплайна

SArray<MbCartPoint> arrPnts(8);

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

arrPnts.Add( MbCartPoint(-7, 20) );

arrPnts.Add( MbCartPoint(8.5, 30) );

arrPnts.Add( MbCartPoint(33, 31) );

arrPnts.Add( MbCartPoint(30.5,25.5) );

arrPnts.Add( MbCartPoint(45, 9.5) );

arrPnts.Add( MbCartPoint(50.5, 15) );

arrPnts.Add( MbCartPoint(72.5, 2) );

// Построение незамкнутого кубического сплайна по контрольным точкам

MbCubicSpline\* pSpline = new MbCubicSpline( arrPnts, false );

// Построение ломаной линии по контрольным точкам сплайна

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

// Отображение контрольных точек

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

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

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

// Отображение ломаной

viewManager->AddObject( Style(2, RGB(0,255,0)), pPolyline, &pl );

// Отображение сплайна

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pSpline );

::DeleteItem( pPolyline );

}

**23. Применение скруглений для сопряжения кубического сплайна с другими кривыми**

#include "cur\_line\_segment.h" // MbLineSegment - класс двумерного отрезка

#include "cur\_cubic\_spline.h" // MbCubicSpline - кубический сплайн в 2D

#include "cur\_contour.h" // MbContour - контур

void MakeUserCommand0()

{

// Контрольные точки сплайна

SArray<MbCartPoint> arrPnts(8);

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

arrPnts.Add( MbCartPoint(-7, 20) );

arrPnts.Add( MbCartPoint(8.5, 30) );

arrPnts.Add( MbCartPoint(33, 31) );

arrPnts.Add( MbCartPoint(30.5, 25.5) );

arrPnts.Add( MbCartPoint(45, 9.5) );

arrPnts.Add( MbCartPoint(50.5, 15) );

arrPnts.Add( MbCartPoint(72.5, 2) );

// Построение незамкнутого кубического сплайна по контрольным точкам

MbCubicSpline\* pSpline = new MbCubicSpline( arrPnts, false );

// Два отрезка, соединяющихся со сплайном в концевых точках

MbLineSegment\* pSeg1 = new MbLineSegment(MbCartPoint(0, -50), MbCartPoint(0, 0));

MbLineSegment\* pSeg2 = new MbLineSegment(MbCartPoint(72.5,2), MbCartPoint(130,-45));

// Построение контура из отрезков и сплайна

MbContour\* pContour = new MbContour();

pContour->AddSegment( pSeg1 );

pContour->AddSegment( pSpline );

pContour->AddSegment( pSeg2 );

// Скругление в угловых точках контура

const int FILLET\_RAD = 10;

ptrdiff\_t idxSegLeft = 0;

pContour->FilletTwoSegments( idxSegLeft, FILLET\_RAD );

ptrdiff\_t idxSegRight = pContour->GetSegmentsCount()-2;

pContour->FilletTwoSegments( idxSegRight, FILLET\_RAD );

// Отображение контрольных точек

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

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

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

// Отображение контура

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pSpline );

::DeleteItem( pSeg1 );

::DeleteItem( pSeg2 );

::DeleteItem( pContour );

}

**24. Построение кубической кривой Безье по контрольным точкам**

#include "cur\_bezier.h" // MbBezier - кубическая кривая Безье

#include "cur\_line\_segment.h" // MbLineSegment - отрезок

void MakeUserCommand0()

{

// Контрольные точки

SArray<MbCartPoint> arrPnts(4);

// Для рис. 19(а)

/\*

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

arrPnts.Add( MbCartPoint(0,30) );

arrPnts.Add( MbCartPoint(100, 30) );

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

\*/

// Для рис. 19(б)

/\*

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

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

arrPnts.Add( MbCartPoint(80, 30) );

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

\*/

// Для рис. 19(в)

/\*

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

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

arrPnts.Add( MbCartPoint(60, 30) );

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

\*/

// Для рис. 19(г)

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

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

arrPnts.Add( MbCartPoint(70, -30) );

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

// Построение незамкнутого кубического сплайна Безье

// Используется конструктор для построения по четырем контрольным точкам

MbBezier\* pSpline = new MbBezier( arrPnts );

// Отображение контрольных точек

MbPlacement3D pl;

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

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

// Отображение сплайна

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

// Отображение контрольной ломаной сплайна

for (int i = 0; i < arrPnts.size()-1; i++)

{

MbLineSegment\* pSeg = new MbLineSegment(arrPnts[i], arrPnts[i+1]);

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

::DeleteItem( pSeg );

}

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pSpline );

}

**25. Отображение локальной системы координат в виде набора элементарных поверхностей**

#include "surf\_sphere\_surface.h" // MbSphereSurface – сферическая поверхность

#include "surf\_cylinder\_surface.h" // MbCylinderSurface – цилиндрическая поверхность

#include "surf\_cone\_surface.h" // MbConeSurface - коническая поверхность"

// Функция для отображения локальной СК \_cs в виде набора элементарных поверхностей

void DrawCoordSystem( const MbPlacement3D& \_cs, const double \_axesLen )

{

// Центр и оси СК \_cs

MbCartPoint3D csCnt( \_cs.GetOrigin() );

// При создании осей в виде объектов MbAxis3D необходимо корректно указывать

// не только направление, но и начало оси – иначе повороты вокруг этих

// осей будут вычисляться некорректно.

MbAxis3D csAxX( csCnt, \_cs.GetAxisX() );

MbAxis3D csAxY( csCnt, \_cs.GetAxisY() );

MbAxis3D csAxZ( csCnt, \_cs.GetAxisZ() );

// Характеристики цилиндрических поверхностей, изображающих оси координат:

// - высота цилиндров - \_axesLen

// - радиус цилиндров - 5% от высоты, но не менее MIN\_CYL\_RAD

double cylHeight = \_axesLen;

const double MIN\_CYL\_RAD = 0.5;

double cylRadius = std::max<double>( cylHeight\*0.05, MIN\_CYL\_RAD );

// Вычисление локальных СК для цилиндров, изображающих

// координатные оси: plCylX, plCylY, plCylZ.

// Центры СК цилиндров совпадают с центром \_cs.

// В локальной СК каждого цилиндра ось Z ориентирована вдоль одной из осей \_cs,

// поэтому plCyZ совпадает с \_cs, а plCylX и plCylY вычисляются посредством

// поворота \_cs вокруг одной из осей на 90 градусов.

MbPlacement3D plCylX(\_cs), plCylY(\_cs), plCylZ(\_cs);

plCylX.Rotate( csAxY, M\_PI/2 );

plCylY.Rotate( csAxX, -M\_PI/2 );

// Цилиндрические поверхности, изображающие оси X, Y и Z системы координат \_cs

MbCylinderSurface\* pCylX = new MbCylinderSurface( plCylX, cylRadius, cylHeight );

MbCylinderSurface\* pCylY = new MbCylinderSurface( plCylY, cylRadius, cylHeight );

MbCylinderSurface\* pCylZ = new MbCylinderSurface( plCylZ, cylRadius, cylHeight );

// Сферическая поверхность, изображающая начало координат.

// Радиус сферы превышает радиус цилиндров, чтобы сферу было видно при отображении.

double sphereRadius = cylRadius\*2;

MbSphereSurface\* pSphere = new MbSphereSurface( \_cs.GetOrigin(), sphereRadius );

// Характеристики конических поверхностей, изображающих стрелки на осях координат:

// - радиусы конусов совпадают с радиусом сферы, изображающей начало координат;

// - высота конусов - 20% от высоты цилиндров;

// - угол между образующей и высотой конуса определяется радиусом и высотой.

double coneRadius = sphereRadius;

double coneHeight = cylHeight\*0.2;

double coneAngle = atan( coneRadius/coneHeight );

// Вычисление локальных СК конических поверхностей, изображающих

// стрелки осей координат: plConeX, plConeY и plConeZ.

// Ось конуса в его локальной СК направлена в сторону "-Z", а вершина конуса

// располагается в начале этой СК.

// Для вычисления локальной СК каждого конуса производятся действия:

// 1) СК конуса совмещается с СК цилиндра;

// 2) СК поворачивается вокруг своей оси X, чтобы изменить направление

// оси Z на противоположное;

// 3) СК смещается вдоль оси Z на величину -coneOfs так, чтобы основание

// конуса располагалось на "конце оси".

MbPlacement3D plConeX = plCylX;

double coneOfs = (cylHeight + coneHeight);

MbAxis3D cylX\_axX(plCylX.GetOrigin(), plCylX.GetAxisX());

plConeX.Rotate(cylX\_axX, M\_PI);

plConeX.Move( plConeX.GetAxisZ() \* -coneOfs );

MbPlacement3D plConeY = plCylY;

MbAxis3D cylY\_axX(plCylY.GetOrigin(), plCylY.GetAxisX());

plConeY.Rotate(cylY\_axX, M\_PI);

plConeY.Move( plConeY.GetAxisZ() \* -coneOfs );

MbPlacement3D plConeZ = plCylZ;

MbAxis3D cylZ\_axX(plCylZ.GetOrigin(), plCylZ.GetAxisX());

plConeZ.Rotate(cylZ\_axX, M\_PI);

plConeZ.Move( plConeZ.GetAxisZ() \* -coneOfs );

// При построении конических поверхностей в качестве радиуса указывается 0 - в таком

// случае строится конус с вершиной. Если указывается значение радиуса >0 - то

// строится поверхность усеченного конуса.

MbConeSurface\* pConeX = new MbConeSurface( plConeX, 0.0, coneAngle, coneHeight );

MbConeSurface\* pConeY = new MbConeSurface( plConeY, 0.0, coneAngle, coneHeight );

MbConeSurface\* pConeZ = new MbConeSurface( plConeZ, 0.0, coneAngle, coneHeight );

// Отображение поверхностей

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

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

viewManager->AddObject(Style(1, LIGHTGREEN), pCylY );

viewManager->AddObject(Style(1, LIGHTGREEN), pConeY );

viewManager->AddObject(Style(1, LIGHTBLUE), pCylZ );

viewManager->AddObject(Style(1, LIGHTBLUE), pConeZ );

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pCylX );

::DeleteItem( pCylY );

::DeleteItem( pCylZ );

::DeleteItem( pConeX );

::DeleteItem( pConeY );

::DeleteItem( pConeZ );

::DeleteItem( pSphere );

}

void MakeUserCommand0()

{

// Отображение мировой СК с осями в виде цилиндров длиной 10.0

MbPlacement3D pl;

DrawCoordSystem( pl, 10.0 );

// Локальная СК, смещенная относительно мировой вдоль оси Y

MbPlacement3D pl1 = pl;

pl1.Move(MbVector3D(0, 15, 0));

DrawCoordSystem( pl1, 5.0 );

// Локальная СК, полученная преобразованиями относительно мировой СК:

// - поворот вокруг Z;

// - смещение вдоль оси Z.

MbPlacement3D pl2 = pl;

MbAxis3D global\_axZ( MbCartPoint3D(0,0,0), MbVector3D(0,0,1) );

pl2.Rotate( global\_axZ, M\_PI/4 );

pl2.Move(MbVector3D(0, 0, 25));

DrawCoordSystem( pl2, 5.0 );

// Локальная СК, полученная преобразованиями относительно мировой СК:

// - поворот вокруг оси X

// - поворот вокруг оси Y

// - смещение вдоль направления (1, 1, 1)

// Замечание: При отображении этой СК в тестовом приложении в изометрической

// проекции может показаться, что СК отображается неправильно и ее начало

// координат совпадает с началом мировой СК.

// Чтобы убедиться в корректности построения, измените вид проекции с

// помощью команды меню тестового приложения Окно->Ориентировать или

// поверните модель нажатием комбинации клавиш Ctrl+стрелка.

MbPlacement3D pl3 = pl;

MbAxis3D global\_axX( MbCartPoint3D(0,0,0), MbVector3D(1,0,0) );

MbAxis3D global\_axY( MbCartPoint3D(0,0,0), MbVector3D(0,1,0) );

pl3.Rotate( global\_axX, M\_PI/3 );

pl3.Rotate( global\_axY, M\_PI/3 );

pl3.Move(40 \* MbVector3D(1, 1, 1));

DrawCoordSystem( pl3, 5.0 );

}

**26. MbPlane – плоскость**

#include "surf\_plane.h"

void MakeUserCommand0()

{

// Плоскость - построение по трем точкам

MbCartPoint3D p1( -15, 0, 0 ), p2( 15, 0, 0 ), p3( 0, 0, 15 );

MbPlane\* pPlane = new MbPlane( p1, p2, p3 );

// Отображение поверхности

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pPlane );

}

**27. MbSphereSurface – сферическая поверхность**

#include "surf\_sphere\_surface.h"

void MakeUserCommand0()

{

// Сферическая поверхность - построение по центру и радиусу

MbSphereSurface\* pSurf = new MbSphereSurface( MbCartPoint3D(0,0,0), 10 );

// Отображение поверхности

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pSurf );

}

**28. MbCylinderSurface – цилиндрическая поверхность**

#include "surf\_cylinder\_surface.h"

void MakeUserCommand0()

{

// Цилиндрическая поверхность - построение по трем точкам

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

const MbCartPoint3D p1( 0, 10, 0 ); // Вектор p0-p1 задает ось Z локальной СК

// и высоту цилиндра

const MbCartPoint3D p2( 2, 0, 0 ); // Вектор p0-p2 задает ось X локальной СК и

// радиус основания

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

// Отображение поверхности

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pSurf);

}

**29. MbConeSurface – коническая поверхность**

#include "surf\_cone\_surface.h"

void MakeUserCommand0()

{

// Коническая поверхность - конструктор по локальной системе координат,

// радиусу, углу и высоте.

const MbPlacement3D pl; // 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\* pSurf = new MbConeSurface( pl, radius, angle, height );

// Отображение поверхности

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pSurf );

}

**30. MbTorusSurface – тороидальная поверхность**

#include "surf\_torus\_surface.h"

void MakeUserCommand0()

{

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

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

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

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

const double radius2 = 3; // Радиус трубки тора

MbTorusSurface\* pSurf = new MbTorusSurface( pl, radius1, radius2 );

// Отображение поверхности

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pSurf );

}

**31. MbRevolutionSurface – поверхность вращения**

#include "surf\_revolution\_surface.h"

#include "surf\_plane.h"

#include "cur\_surface\_intersection.h"

#include "cur\_bezier3d.h"

#include "mb\_axis3d.h"

#include "action\_surface\_curve.h"

#include "name\_item.h"

#include "wire\_frame.h"

void MakeUserCommand0()

{

// Построение поверхности вращения 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\* pSurf = new MbRevolutionSurface(

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

// Вычисление сечения поверхности pSurf горизонтальной плоскостью

MbPlane\* pPlane = new MbPlane( MbCartPoint3D( -20, 0, 0 ),

MbCartPoint3D( 20, 0, -20 ), MbCartPoint3D( 20, 0, 20 ) );

MbSNameMaker names( ct\_Curve3DCreator, MbSNameMaker::i\_SideNone, 0 );

MbWireFrame\* pFrCurve = new MbWireFrame();

// Получение кривых пересечения поверхности pSurf плоскостью pPlane

IntersectionCurve( \*pSurf, \*pPlane, names, pFrCurve );

RPArray<MbCurve3D> arrCurves( 4 /\* Начальная емкость массива \*/ );

::ExtractCurvesDeleteFrame( pFrCurve, arrCurves );

// Кривая pC, изображающая направляющую - это вычисленная кривая сечения,

// к которой применяется преобразование масштабирования, чтобы

// это сечение было видно поверх поверхности вращения

MbCurve3D\* pC = ((MbSurfaceIntersectionCurve\*)(arrCurves[0]))->MakeCurve();

MbMatrix3D mScale = MbMatrix3D::identity;

mScale.Scale( 1.05, 1, 1.05 );

pC->Transform( mScale );

// Отображение образующей, направляющей и поверхности

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

viewManager->AddObject(Style(5, LIGHTBLUE), pC );

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pGenCurve);

::DeleteItem( pFrCurve);

::DeleteItem( pSurf );

}

**32. MbExtrusionSurface – поверхность выдавливания**

#include "surf\_extrusion\_surface.h"

#include "cur\_line\_segment3d.h"

#include "cur\_bezier3d.h"

#include "mb\_axis3d.h"

void MakeUserCommand0()

{

// Построение поверхности выдавливания по образующей и направлению выдавливания

SArray<MbCartPoint3D> arrGenPnts(9);

arrGenPnts.Add(MbCartPoint3D(14.2875, -88.9000, 0));

arrGenPnts.Add(MbCartPoint3D(17.4625, -63.5000, 0));

arrGenPnts.Add(MbCartPoint3D(41.2750, -63.5000, 0));

arrGenPnts.Add(MbCartPoint3D(47.6250, -44.7145, 0));

arrGenPnts.Add(MbCartPoint3D(58.7375, -31.4854, 0));

arrGenPnts.Add(MbCartPoint3D(74.0833, -37.0417, 0));

arrGenPnts.Add(MbCartPoint3D(87.5771, -24.8708, 0));

arrGenPnts.Add(MbCartPoint3D(80.1687, -16.9333, 0));

arrGenPnts.Add(MbCartPoint3D(85.9896, -9.2604, 0));

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

MbVector3D dir( 0, 0, 50 );

MbExtrusionSurface\* pSurf = new MbExtrusionSurface( \*pGenCurve, dir, false );

// Отрезок, представляющий направляющую выдавливания

MbLineSegment3D\* pTrajCurve = new MbLineSegment3D(arrGenPnts.back(),

arrGenPnts.back() + dir );

// Отображение образующей, направляющей и поверхности

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

viewManager->AddObject(Style(5, LIGHTBLUE), pTrajCurve);

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pGenCurve );

::DeleteItem( pTrajCurve );

::DeleteItem( pSurf );

}

**33. MbExpansionSurface – поверхность плоскопараллельного движения**

#include "surf\_expansion\_surface.h"

#include "cur\_bezier3d.h"

#include "cur\_polyline3d.h"

#include "mb\_axis3d.h"

void MakeUserCommand0()

{

// Построение поверхности плоскопараллельного движения по образующей и направляющей

// Образующая строится в виде кривой Безье, проходящей через заданные точки

SArray<MbCartPoint3D> arrGenPnts(7);

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

arrGenPnts.Add(MbCartPoint3D(25.4, -7.94, 0));

arrGenPnts.Add(MbCartPoint3D(43.39, 6.09, 0));

arrGenPnts.Add(MbCartPoint3D(44.19, 33.34, 0));

arrGenPnts.Add(MbCartPoint3D(30.69, 56.62, 0));

arrGenPnts.Add(MbCartPoint3D(51.85, 79.40, 0));

arrGenPnts.Add(MbCartPoint3D(57.41, 94.75, 0));

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

// Направляющая строится в виде ломаной линии

SArray<MbCartPoint3D> arrTrajPnts (6);

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

arrTrajPnts.Add(MbCartPoint3D(10, 0, 0));

arrTrajPnts.Add(MbCartPoint3D(10, 0, 20));

arrTrajPnts.Add(MbCartPoint3D(5, 0, 20));

arrTrajPnts.Add(MbCartPoint3D(5, 0, 40));

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

MbPolyline3D\* pTrajCurve = new MbPolyline3D( arrTrajPnts, false );

// Построение поверхности

MbExpansionSurface\* pSurf = new MbExpansionSurface( \*pGenCurve, \*pTrajCurve,

false, false, NULL );

// Отображение образующей, направляющей и поверхности

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

viewManager->AddObject(Style(5, LIGHTBLUE), pTrajCurve);

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pGenCurve);

::DeleteItem( pTrajCurve);

::DeleteItem( pSurf );

}

**34. MbSpiralSurface – спиральная поверхность**

#include "surf\_spiral\_surface.h"

#include "cur\_arc3d.h"

#include "cur\_cone\_spiral.h"

#include "mb\_cart\_point3d.h"

void MakeUserCommand0()

{

// Построение образующей и направляющей

MbArc3D\* pGenCurve = new MbArc3D(

MbCartPoint3D(5, 0, 0), MbCartPoint3D(6, 0, 0),

MbCartPoint3D(5, 0, -1), 0, true );

MbPlacement3D pl( MbCartPoint3D(0,0,0 ) );

MbConeSpiral\* pCS = new MbConeSpiral( pl, 5, 5, 17, 3 );

// Построение спиральной поверхности

MbSpiralSurface\* pSurf = new MbSpiralSurface( \*pGenCurve, \*pCS, false );

// Отображение образующей, направляющей и поверхности

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

viewManager->AddObject(Style(5, LIGHTBLUE), pCS );

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pCS );

::DeleteItem( pGenCurve);

::DeleteItem( pSurf );

}

**35. MbEvolutionSurface – кинематическая поверхность (Что-то похожее на Loft в Inventor)**

#include "surf\_evolution\_surface.h"

#include "cur\_arc3d.h"

#include "cur\_bezier3d.h"

#include "mb\_cart\_point3d.h"

void MakeUserCommand0()

{

// Построение образующей (окружность по трем точкам)

MbArc3D\* pGenCurve = new MbArc3D( MbCartPoint3D(6.6, 1.3, 0),

MbCartPoint3D(0, 0, 0),

MbCartPoint3D(11.7, 6.1, 0));

// Построение направляющей (кривая Безье по точкам)

SArray<MbCartPoint3D> arrTrajPnts(7);

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

arrTrajPnts.Add(MbCartPoint3D(7.6, 0, 18.2));

arrTrajPnts.Add(MbCartPoint3D(18.5, 0, 20.6));

arrTrajPnts.Add(MbCartPoint3D(36.0, 0, 14.7));

arrTrajPnts.Add(MbCartPoint3D(40.1, 0.0, 0));

arrTrajPnts.Add(MbCartPoint3D(69.7, 0.0, 0));

arrTrajPnts.Add(MbCartPoint3D(80.4, 0.0, 0));

MbBezier3D\* pTrajCurve = new MbBezier3D(arrTrajPnts, false);

// Описание направляющей для кинематической поверхности

MbSpine::Create(\*pTrajCurve, false, false);

// Построение кинематической поверхности

MbEvolutionSurface\* pSurf = new MbEvolutionSurface(\*pGenCurve,

MbSpine::Create(\*pTrajCurve, false, false), false, false);

// Отображение образующей, направляющей и поверхности

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

viewManager->AddObject(Style(5, LIGHTBLUE), pTrajCurve);

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pGenCurve);

::DeleteItem(pTrajCurve);

::DeleteItem(pSurf);

}

**36. MbExactionSurface – кинематическая поверхность с адаптацией**

#include "surf\_exaction\_surface.h"

#include "cur\_arc3d.h"

#include "cur\_bezier3d.h"

#include "mb\_cart\_point3d.h"

void MakeUserCommand0()

{

// Построение образующей (окружность по трем точкам)

MbArc3D\* pGenCurve = new MbArc3D(MbCartPoint3D(6.6, 1.3, 0),

MbCartPoint3D(0, 0, 0), MbCartPoint3D(11.7, 6.1, 0));

// Построение направляющей (кривая Безье по точкам)

SArray<MbCartPoint3D> arrTrajPnts(7);

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

arrTrajPnts.Add(MbCartPoint3D(7.6, 0, 18.2));

arrTrajPnts.Add(MbCartPoint3D(18.5, 0, 20.6));

arrTrajPnts.Add(MbCartPoint3D(36.0, 0, 14.7));

arrTrajPnts.Add(MbCartPoint3D(40.1, 0.0, 0));

arrTrajPnts.Add(MbCartPoint3D(69.7, 0.0, 0));

arrTrajPnts.Add(MbCartPoint3D(80.4, 0.0, 0));

// Описание направляющей для кинематической поверхности.

MbBezier3D\* pTrajCurve = new MbBezier3D(arrTrajPnts, false);

// Построение кинематической поверхности с адаптацией

// Для поверхности с адаптацией с помощью векторов нормали задается ориентация

// сечений на границах поверхности

MbVector3D n0(0, 5, -5);

MbVector3D n1(5, 5, 0);

MbExactionSurface\* pSurf = new MbExactionSurface(\*pGenCurve,

MbSpine::Create(\*pTrajCurve, false, false),n0, n1, M\_PI / 4, M\_PI / 4, false);

// Отображение образующей, направляющей и поверхности

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

viewManager->AddObject(Style(5, LIGHTBLUE), pTrajCurve);

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pGenCurve);

::DeleteItem(pTrajCurve);

::DeleteItem(pSurf);

}

**37. MbRuledSurface – линейная поверхность**

#include "surf\_ruled\_surface.h"

#include "cur\_polyline3d.h"

#include "cur\_arc3d.h"

void MakeUserCommand0()

{

// Первая граничная кривая – дуга окружности

MbCartPoint3D p1(-1.5, 0, 0), p2(1, 2, 0), p3(3.5, 0, 0);

MbArc3D\* pC1 = new MbArc3D(p1, p2, p3, 1, false);

// Вторая граничная кривая – ломаная по точкам

SArray<MbCartPoint3D> arrPnts(8);

arrPnts.Add(MbCartPoint3D(-3, 0, -5));

arrPnts.Add(MbCartPoint3D(-3, 6, -5));

arrPnts.Add(MbCartPoint3D(0, 6, -5));

arrPnts.Add(MbCartPoint3D(0, 4, -5));

arrPnts.Add(MbCartPoint3D(2, 4, -5));

arrPnts.Add(MbCartPoint3D(2, 6, -5));

arrPnts.Add(MbCartPoint3D(5, 6, -5));

arrPnts.Add(MbCartPoint3D(5, 0, -5));

MbPolyline3D\* pC2 = new MbPolyline3D(arrPnts, false);

// Построение линейчатой поверхности по двум граничным кривым

MbRuledSurface\* pSurf = new MbRuledSurface(\*pC1, \*pC2, false);

// Отображение граничных кривых и поверхности

viewManager->AddObject(Style(5, LIGHTRED), pC1);

viewManager->AddObject(Style(5, LIGHTRED), pC2);

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem(pC1);

::DeleteItem(pC2);

::DeleteItem(pSurf);

}

**38. MbSectorSurface – секториальная поверхность**

#include "surf\_sector\_surface.h"

#include "cur\_polyline3d.h"

#include "cur\_arc3d.h"

#include "point\_frame.h"

void MakeUserCommand0()

{

// Граничная кривая – ломаная по точкам

SArray<MbCartPoint3D> arrPnts(8);

arrPnts.Add(MbCartPoint3D(-3, 0, -5));

arrPnts.Add(MbCartPoint3D(-3, 6, -5));

arrPnts.Add(MbCartPoint3D(0, 6, -5));

arrPnts.Add(MbCartPoint3D(0, 4, -5));

arrPnts.Add(MbCartPoint3D(2, 4, -5));

arrPnts.Add(MbCartPoint3D(2, 6, -5));

arrPnts.Add(MbCartPoint3D(5, 6, -5));

arrPnts.Add(MbCartPoint3D(5, 0, -5));

MbPolyline3D\* pC = new MbPolyline3D( arrPnts, false );

// Построение секториальной поверхности по граничной кривой и точке

MbSectorSurface\* pSurf = new MbSectorSurface( \*pC, MbCartPoint3D( 1, 0, 0 ), false );

// Отображение граничной кривой, точки и поверхности

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

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

new MbPointFrame(MbCartPoint3D(1, 0, 0)) );

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pC );

::DeleteItem( pSurf );

}

**39. MbLoftedSurface – поверхность, проходящая через заданное семейство кривых**

#include "surf\_lofted\_surface.h"

#include "cur\_polyline3d.h"

#include "cur\_arc3d.h"

void MakeUserCommand0()

{

// 5 кривых сечений: дуга окружности, ломаная, дуга, ломаная, дуга

MbArc3D\* pC1 = new MbArc3D(MbCartPoint3D(-3, 0, 0),

MbCartPoint3D(0, 2, 0), MbCartPoint3D(3, 0, 0), 1, false );

std::vector<MbCartPoint3D> arrPnts2 = { { -3, 0, -5 }, { -3, 2, -5 },

{ 3, 2, -5 }, { 3, 0, -5 } };

MbPolyline3D\* pC2 = new MbPolyline3D( arrPnts2, false );

MbArc3D\* pC3 = new MbArc3D(MbCartPoint3D(-3, 5, -10),

MbCartPoint3D(0, 7, -10), MbCartPoint3D(3, 5, -10), 1, false );

std::vector<MbCartPoint3D> arrPnts4 = { { -3, 0, -15 }, { -3, 2, -15 },

{ 3, 2, -15 }, { 3, 0, -15 } };

MbPolyline3D\* pC4 = new MbPolyline3D( arrPnts4, false );

MbArc3D\* pC5 = new MbArc3D(MbCartPoint3D(-3, 0, -20),

MbCartPoint3D(0, 2, -20), MbCartPoint3D(3, 0, -20), 1, false );

// Помещение указателей на сечения в массив

RPArray<MbCurve3D> arrCurves;

arrCurves.Add( pC1 );

arrCurves.Add( pC2 );

arrCurves.Add( pC3 );

arrCurves.Add( pC4 );

arrCurves.Add( pC5 );

// Построение поверхности по сечениям

MbVector3D v1( 0, 0, 1 );

MbVector3D v2( 0, 0, -1 );

MbLoftedSurface\* pSurf = new MbLoftedSurface( arrCurves, false, v1, v2, false );

// Отображение кривых-сечений

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

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

// Отображение поверхности

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

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

::DeleteItem(arrCurves[i]);

::DeleteItem( pSurf );

}

**40. MbElevationSurface – поверхность, проходящая через семейство кривых с заданной направляющей**

#include "surf\_elevation\_surface.h"

#include "cur\_polyline3d.h"

#include "cur\_bezier3d.h"

#include "cur\_arc3d.h"

void MakeUserCommand0()

{

// 5 кривых сечений: дуга окружности, ломаная, дуга, ломаная, дуга

MbArc3D\* pC1 = new MbArc3D(MbCartPoint3D(-3, 0, 0),

MbCartPoint3D(0, 2, 0), MbCartPoint3D(3, 0, 0), 1, false );

std::vector<MbCartPoint3D> arrPnts2 = { { -3, 0, -5 }, { -3, 2, -5 },

{ 3, 2, -5 }, { 3, 0, -5 } };

MbPolyline3D\* pC2 = new MbPolyline3D( arrPnts2, false );

MbArc3D\* pC3 = new MbArc3D(MbCartPoint3D(-3, 5, -10),

MbCartPoint3D(0, 7, -10), MbCartPoint3D(3, 5, -10), 1, false );

std::vector<MbCartPoint3D> arrPnts4 = { { -3, 0, -15 }, { -3, 2, -15 },

{ 3, 2, -15 }, { 3, 0, -15 } };

MbPolyline3D\* pC4 = new MbPolyline3D( arrPnts4, false );

MbArc3D\* pC5 = new MbArc3D(MbCartPoint3D(-3, 0, -20),

MbCartPoint3D(0, 2, -20), MbCartPoint3D(3, 0, -20), 1, false );

// Помещение указателей на сечения в массив

RPArray<MbCurve3D> arrCurves;

arrCurves.Add( pC1 );

arrCurves.Add( pC2 );

arrCurves.Add( pC3 );

arrCurves.Add( pC4 );

arrCurves.Add( pC5 );

// Направляющая кривая – кривая Безье

SArray<MbCartPoint3D> arrPnts(9);

arrPnts.Add(MbCartPoint3D(0, 2, 0));

arrPnts.Add(MbCartPoint3D(0, 3, -2.5));

arrPnts.Add(MbCartPoint3D(0, 2, -5));

arrPnts.Add(MbCartPoint3D(0, 6.0, -7.5));

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

arrPnts.Add(MbCartPoint3D(0, 6.0, -12.5));

arrPnts.Add(MbCartPoint3D(0, 2, -15));

arrPnts.Add(MbCartPoint3D(0, 3, -17.5));

arrPnts.Add(MbCartPoint3D(0, 2, -20));

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

// Построение поверхности по сечениям с заданной направляющей

MbElevationSurface\* pSurf = new MbElevationSurface(arrCurves, false, \*pSpine, false);

// Отображение сечений

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

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

// Отображение направляющей

viewManager->AddObject(Style(5, LIGHTBLUE), pSpine );

// Отображение поверхности

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

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

::DeleteItem(arrCurves[i]);

::DeleteItem( pSurf );

}

**41. MbJoinSurface – поверхность соединения**

#include "surf\_extrusion\_surface.h" // MbExtrusionSurface - Поверхность выдавливания

// MbJoinSurface - Поверхность сопряжения по набору кривых

#include "surf\_join\_surface.h"

#include "cur\_line\_segment3d.h" // MbLineSegment3D - Отрезок в трехмерном пространстве

#include "cur\_bezier3d.h" // MbBezier3D - Сплайновая кривая Безье

void MakeUserCommand0()

{

// Фрагменты плоскостей (прямоугольники), для которых надо построить сопрягающую

// поверхность.

// Поскольку класс "Плоскость" MbPlane предназначен для представления

// бесконечной плоскости, то прямоугольник удобнее построить как поверхность

// выдавливания с указанием отрезка в качестве образующей.

MbLineSegment3D\* pBoundSeg1 = new MbLineSegment3D(

MbCartPoint3D( -5, 0, 0 ), MbCartPoint3D ( 5, 0, 0 ) );

MbVector3D vecGen1( 0, 0, 15 );

MbExtrusionSurface\* pPlane1 = new MbExtrusionSurface( \*pBoundSeg1, vecGen1, false );

MbLineSegment3D\* pBoundSeg2 = new MbLineSegment3D(

MbCartPoint3D( -5, 5, -10 ), MbCartPoint3D ( 5, 5, -10 ) );

MbVector3D vecGen2( 0, 15, -5 );

MbExtrusionSurface\* pPlane2 = new MbExtrusionSurface( \*pBoundSeg2, vecGen2, false );

// Сопрягающая поверхность по набору кривых MbJoinSurface.

// Для ее построения указывается набор кривых, содержащий граничные кривые

// сопрягаемых поверхностей, а также промежуточные кривые, регулирующие форму

// сопряжения (в данном случае это пара незамкнутых кривых Безье,

// напоминающих волнистую линию).

SArray<MbCartPoint3D> arrPnts(7);

arrPnts.Add(MbCartPoint3D(-5, 2, -2));

arrPnts.Add(MbCartPoint3D(-4, 3, -2));

arrPnts.Add(MbCartPoint3D(-3, 2, -2));

arrPnts.Add(MbCartPoint3D(0, 1, -2));

arrPnts.Add(MbCartPoint3D(3, 2, -2));

arrPnts.Add(MbCartPoint3D(4, 3, -2));

arrPnts.Add(MbCartPoint3D(5, 2, -2));

MbBezier3D\* pCurve1 = new MbBezier3D( arrPnts, false );

MbBezier3D\* pCurve2 = new MbBezier3D( arrPnts, false );

pCurve2->Move(MbVector3D( 0, 3, -3 ));

RPArray<MbCurve3D> arrCurves;

arrCurves.Add( pBoundSeg1 );

arrCurves.Add( pCurve1 );

arrCurves.Add( pCurve2 );

arrCurves.Add( pBoundSeg2 );

MbJoinSurface\* pSurf = new MbJoinSurface( arrCurves, false );

// Отображение сопрягаемых плоскостей

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane1 );

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane2 );

// Отображение сопрягающей поверхности

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

// Отображение граничных кривых (в данном примере - отрезков)

// сопрягаемых поверхностей

viewManager->AddObject(Style(5, LIGHTRED), pBoundSeg1 );

viewManager->AddObject(Style(5, LIGHTRED), pBoundSeg2 );

// Отображение кривых, используемых для задания формы сопрягающей поверхности

// (за исключением первой и последней - они совпадают с граничными кривыми)

for (int i = 1, iLim = arrCurves.size()-1; i<iLim; i++)

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

// (pBoundSeg1 и pBoundSeg1 освобождаются в составе arrCurves)

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

::DeleteItem( arrCurves[i] );

::DeleteItem( pPlane1 );

::DeleteItem( pPlane2 );

::DeleteItem( pSurf );

}

**42. MbChamferSurface – поверхность фаски**

#include "surf\_extrusion\_surface.h" // MbExtrusionSurface - Поверхность выдавливания

// MbChamferSurface - Сопрягающая поверхность фаски

#include "surf\_chamfer\_surface.h"

#include "cur\_bezier3d.h" // MbBezier3D - Кривая Безье в трехмерном пространстве

#include "cur\_surface\_curve.h" // MbSurfaceCurve - Кривая на базовой поверхности

// Алгоритмы для построения кривых в трехмерном пространстве

#include "action\_surface\_curve.h"

void MakeUserCommand0()

{

// Фрагменты плоскостей, для которых надо построить сопрягающую поверхность.

// Эти фрагменты строятся как поверхности выдавливания с образующими в виде

// кривых Безье.

SArray<MbCartPoint3D> arrPnts1(4);

arrPnts1.Add(MbCartPoint3D(-5, 0, 0));

arrPnts1.Add(MbCartPoint3D(0, 0, 0));

arrPnts1.Add(MbCartPoint3D(0, 0, 3));

arrPnts1.Add(MbCartPoint3D(5, 0, 0));

MbBezier3D\* pBoundBezier1 = new MbBezier3D( arrPnts1, false );

MbVector3D vecGen1( 0, 0, 15 );

MbExtrusionSurface\* pPlane1 = new MbExtrusionSurface(\*pBoundBezier1, vecGen1, false);

SArray<MbCartPoint3D> arrPnts2(4);

arrPnts2.Add(MbCartPoint3D(-5, 5, -10));

arrPnts2.Add(MbCartPoint3D(0, 4, -11));

arrPnts2.Add(MbCartPoint3D(1, 5, -12));

arrPnts2.Add(MbCartPoint3D(5, 5, -10));

MbBezier3D\* pBoundBezier2 = new MbBezier3D( arrPnts2, false );

MbVector3D vecGen2( 0, 15, -5 );

MbExtrusionSurface\* pPlane2 = new MbExtrusionSurface(\*pBoundBezier2, vecGen2, false);

// Сопрягаемые граничные кривые областей pPlane1 и pPlane2 необходимо

// представить в виде объектов MbSurfaceCurve - "Кривая на поверхности".

// Эти кривые рассчитываются как проекции pBoundBezier1 и pBoundBezier2 на

// pPlane1 и pPlane2.

// Проекционные кривые совпадают с проецируемыми кривыми Безье, но представляются

// в виде необходимых объектов MbSurfaceCurve.

RPArray<MbCurve3D> arrCurves1;

CurveProjection( \*pPlane1, \*pBoundBezier1, NULL, true, true, arrCurves1 );

RPArray<MbCurve3D> arrCurves2;

CurveProjection( \*pPlane2, \*pBoundBezier2, NULL, true, true, arrCurves2 );

MbSurfaceCurve\* pBoundCurve1 = static\_cast<MbSurfaceCurve\*>(arrCurves1[0]);

MbSurfaceCurve\* pBoundCurve2 = static\_cast<MbSurfaceCurve\*>(arrCurves2[0]);

// Сопрягающая поверхность фаски MbChamferSurface.

MbChamferSurface\* pSurf = new MbChamferSurface( \*pBoundCurve1, \*pBoundCurve2,

0, 0, st\_Chamfer );

// Отображение сопрягаемых плоскостей

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane1 );

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane2 );

// Отображение сопрягающей поверхности

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

// Отображение граничных кривых сопрягаемых поверхностей

viewManager->AddObject(Style(5, LIGHTRED), pBoundCurve1 );

viewManager->AddObject(Style(5, LIGHTRED), pBoundCurve2 );

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pBoundBezier1 );

::DeleteItem( pBoundBezier2 );

::DeleteItem( pBoundCurve1 );

::DeleteItem( pBoundCurve2 );

::DeleteItem( pPlane1 );

::DeleteItem( pPlane2 );

::DeleteItem( pSurf );

}

**43. MbFilletSurface – поверхность скругления**

// MbFilletSurface - Сопрягающая поверхность скругления с постоянным радиусом

#include "surf\_fillet\_surface.h"

#include "surf\_extrusion\_surface.h" // MbExtrusionSurface - Поверхность выдавливания

#include "cur\_line\_segment3d.h" // MbLineSegment3D Отрезок в трехмерном пространстве

#include "cur\_surface\_curve.h" // MbSurfaceCurve - Кривая на поверхности

// Алгоритмы для построения кривых в трехмерном пространстве

#include "action\_surface\_curve.h"

void MakeUserCommand0()

{

// Фрагменты плоскостей, для которых надо построить сопрягающую поверхность.

// Эти фрагменты (прямоугольники) строятся как поверхности выдавливания

// с образующими в виде отрезков.

MbLineSegment3D\* pBoundSeg1 = new MbLineSegment3D(

MbCartPoint3D( -5, 0, 0 ), MbCartPoint3D ( 5, 0, 0 ) );

MbVector3D vecGen1( 0, 0, 15 );

MbExtrusionSurface\* pPlane1 = new MbExtrusionSurface( \*pBoundSeg1, vecGen1, false );

MbLineSegment3D\* pBoundSeg2 = new MbLineSegment3D(

MbCartPoint3D( -5, 5, -10 ), MbCartPoint3D ( 5, 5, -10 ) );

MbVector3D vecGen2( 0, 15, -5 );

MbExtrusionSurface\* pPlane2 = new MbExtrusionSurface( \*pBoundSeg2, vecGen2, false );

// Сопрягаемые граничные кривые областей pPlane1 и pPlane2 необходимо

// представить в виде объектов MbSurfaceCurve - "Кривая на поверхности".

// Эти кривые рассчитываются как проекции pBoundSeg1 и pBoundSeg2 на

// pPlane1 и pPlane2.

// Проекционные кривые совпадают с проецируемыми отрезками, но представляются в виде

// необходимых объектов MbSurfaceCurve.

RPArray<MbCurve3D> arrCurves1;

CurveProjection( \*pPlane1, \*pBoundSeg1, NULL, true, true, arrCurves1 );

RPArray<MbCurve3D> arrCurves2;

CurveProjection( \*pPlane2, \*pBoundSeg2, NULL, true, true, arrCurves2 );

MbSurfaceCurve\* pBoundCurve1 = static\_cast<MbSurfaceCurve\*>(arrCurves1[0]);

MbSurfaceCurve\* pBoundCurve2 = static\_cast<MbSurfaceCurve\*>(arrCurves2[0]);

// Сопрягающая поверхность фаски MbChamferSurface.

MbFilletSurface\* pSurf = new MbFilletSurface( \*pBoundCurve1, \*pBoundCurve2,

-15, -15, st\_Fillet, true, true );

// Отображение сопрягаемых плоскостей

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane1 );

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane2 );

// Отображение сопрягающей поверхности

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

// Отображение граничных кривых сопрягаемых поверхностей

viewManager->AddObject(Style(5, LIGHTRED), pBoundCurve1 );

viewManager->AddObject(Style(5, LIGHTRED), pBoundCurve2 );

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pBoundSeg1 );

::DeleteItem( pBoundSeg2 );

::DeleteItem( pBoundCurve1 );

::DeleteItem( pBoundCurve2 );

::DeleteItem( pPlane1 );

::DeleteItem( pPlane2 );

::DeleteItem( pSurf );

}

**44. MbChannelSurface – поверхность скругения с пременным радиусом**

#include "surf\_extrusion\_surface.h" // MbExtrusionSurface - Поверхность выдавливания

// MbChannelSurface - Поверхность скругления с переменным радиусом

#include "surf\_channel\_surface.h"

#include "cur\_line\_segment3d.h" // MbLineSegment3D - Отрезок в трехмерном пространстве

#include "cur\_surface\_curve.h" // MbSurfaceCurve - Кривая на поверхности

// Алгоритмы для построения кривых в трехмерном пространстве

#include "action\_surface\_curve.h"

#include "func\_line\_function.h" // Объект "Линейная функция"

void MakeUserCommand0()

{

// Фрагменты плоскостей, для которых надо построить сопрягающую поверхность.

// Эти фрагменты (прямоугольники) строятся как поверхности выдавливания

// с образующими в виде отрезков.

MbLineSegment3D\* pBoundSeg1 = new MbLineSegment3D(

MbCartPoint3D( -5, 0, 0 ), MbCartPoint3D ( 5, 0, 0 ) );

MbVector3D vecGen1( 0, 0, 15 );

MbExtrusionSurface\* pPlane1 = new MbExtrusionSurface( \*pBoundSeg1, vecGen1, false );

MbLineSegment3D\* pBoundSeg2 = new MbLineSegment3D( MbCartPoint3D( -5, 5, -10 ),

MbCartPoint3D ( 5, 5, -10 ) );

MbVector3D vecGen2( 0, 15, -5 );

MbExtrusionSurface\* pPlane2 = new MbExtrusionSurface( \*pBoundSeg2, vecGen2, false );

// Сопрягаемые граничные кривые областей pPlane1 и pPlane2 необходимо

// представить в виде объектов MbSurfaceCurve - "Кривая на поверхности".

// Эти кривые рассчитываются как проекции pBoundSeg1 и pBoundSeg2 на

// pPlane1 и pPlane2.

// Проекционные кривые совпадают с проецируемыми отрезками, но представляются в виде

// необходимых объектов MbSurfaceCurve.

RPArray<MbCurve3D> arrCurves1;

CurveProjection( \*pPlane1, \*pBoundSeg1, NULL, true, true, arrCurves1 );

RPArray<MbCurve3D> arrCurves2;

CurveProjection( \*pPlane2, \*pBoundSeg2, NULL, true, true, arrCurves2 );

MbSurfaceCurve\* pBoundCurve1 = static\_cast<MbSurfaceCurve\*>(arrCurves1[0]);

MbSurfaceCurve\* pBoundCurve2 = static\_cast<MbSurfaceCurve\*>(arrCurves2[0]);

// Сопрягающая поверхность скругления с переменным радиусом MbChannelSurface.

// Радиус изменяется линейно в пределах от 0 до 2.5 по мере движения вдоль

// сопрягаемых кривых (это изменение представляется в виде объекта-линейной

// функции класса MbLineFunction).

MbLineFunction\* pFuncLinear = new MbLineFunction( 0, 2.5,

pBoundCurve1->GetTMin(), pBoundCurve1->GetTMax() );

MbChannelSurface\* pSurf = new MbChannelSurface( \*pBoundCurve1, \*pBoundCurve2,

-5, 10, st\_Fillet, \_ARC\_, \*pFuncLinear, true );

// Отображение сопрягаемых плоскостей

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane1 );

viewManager->AddObject(Style(1, LIGHTGRAY), pPlane2 );

// Отображение сопрягающей поверхности

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

// Отображение граничных кривых сопрягаемых поверхностей

viewManager->AddObject(Style(5, LIGHTRED), pBoundCurve1 );

viewManager->AddObject(Style(5, LIGHTRED), pBoundCurve2 );

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pBoundSeg1 );

::DeleteItem( pBoundSeg2 );

::DeleteItem( pBoundCurve1 );

::DeleteItem( pBoundCurve2 );

::DeleteItem( pPlane1 );

::DeleteItem( pPlane2 );

::DeleteItem( pFuncLinear );

::DeleteItem( pSurf );

}

**45. MbSplineSurface – сплайновая NURBS – поверхность**

#include "surf\_spline\_surface.h" // MbSplineSurface - сплайновая NURBS-поверхность

#include "point\_frame.h" // MbPointFrame - "Точечный каркас"

#include "action\_surface.h" // Алгоритмы построения поверхностей

void MakeUserCommand0()

{

// Двумерный массив контрольных точек.

// Для примера используются точки, расположенные в узлах равномерной квадратной

// сетки PNT\_CNTхPNT\_CNT на плоскости XZ мировой СК.

// Вертикальная координата выпуклости рассчитывается по

// формуле y = 5\*cos( r\*(M\_PI/2)/(SIZE/2) ), r=sqrt(x^2+z^2),

// чтобы контрольные точки задавали на сетке центрированную выпуклость

// ("косинусоидальную"), окруженную плоской границей.

// SIZE - размер стороны квадрата на плоскости XZ, который равномерно

// заполняется точками сетки

// BOUND\_CNT - количество точек, образующих плоский участок вокруг выпуклости.

SArray<MbCartPoint3D> arrPnts;

const int PNT\_CNT = 25;

const int BOUND\_CNT = 3;

arrPnts.reserve( PNT\_CNT\*PNT\_CNT );

// Начальная угловая точка сетки на плоскости XZ, относительно которой с равным шагом

// располагаются остальные точки сетки.

const double SIZE = 20;

const double X0 = -SIZE/2;

const double Z0 = -SIZE/2;

// Шаг между точками сетки по координатным осям X и Z

const double DX = SIZE/PNT\_CNT;

const double DZ = SIZE/PNT\_CNT;

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

for (int j = 0; j < PNT\_CNT; j++)

{

double x = X0 + DX\*j;

double z = Z0 + DZ\*i;

double y = 0;

if (i > BOUND\_CNT && i < PNT\_CNT - BOUND\_CNT &&

j > BOUND\_CNT && j < PNT\_CNT - BOUND\_CNT)

{

double r = sqrt( x\*x + z\*z );

y = 15\*cos( r\*M\_PI/2/(SIZE/2) );

}

arrPnts.push\_back( MbCartPoint3D( x, y, z ) );

}

// Свойства, влияющие на вид сплайновой поверхности: весовые коэффициенты и

// компоненты узловых векторов. В данном примере передаются пустые массивы,

// чтобы использовались значения этих свойств "по умолчанию".

SArray<double> arrWeights;

SArray<double> uKnots;

SArray<double> vKnots;

// Порядок сплайновой поверхности по направлениям U и V

const int U\_DEGREE = 3;

const int V\_DEGREE = 3;

// Построение NURBS-поверхности

MbSurface\* pSurf = 0;

::SplineSurface( arrPnts, arrWeights, PNT\_CNT, PNT\_CNT, U\_DEGREE, uKnots,

false, V\_DEGREE, vKnots, false, pSurf );

// Точечный каркас для отображения контрольных точек

MbPointFrame\* pPntsFrame = new MbPointFrame();

for ( int i=0,iLim=arrPnts.Count(); i<iLim; i++ )

pPntsFrame->AddVertex( arrPnts[i] );

// Отображение поверхности

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

// Отображение контрольных точек

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pSurf );

::DeleteItem( pPntsFrame );

}

**46. MbCornerSurface – треугольная поверхность**

// MbCornerSurface - поверхность, построенная по трем кривым

#include "surf\_corner\_surface.h"

#include "cur\_bezier3d.h"

#include "cur\_line\_segment3d.h"

#include "cur\_arc3d.h"

void MakeUserCommand0()

{

// 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 );

}

**47. MbCoverSurface – билинейная поверхность**

// MbCoverSurface - поверхность, построенная по четырем граничным кривым

#include "surf\_cover\_surface.h"

#include "cur\_arc3d.h"

void MakeUserCommand0()

{

// 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, 5, 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 );

}

**48. MbCurveBoundedSurface – поверхность, заданная замкнутыми контурами**

// MbCurveBoundedSurface - поверхность, ограниченная замкнутыми контурами

// на базовой поверхности

#include "surf\_curve\_bounded\_surface.h"

#include "surf\_cylinder\_surface.h"

#include "cur\_line\_segment.h"

void MakeUserCommand0()

{

// Базовая поверхность - цилиндрическая поверхность, построенная по трем точкам

// Центр локальной СК цилиндра (центр основания)

const MbCartPoint3D pс0( 0, 0, 0 );

// Вектор p0-p1 задает ось Z локальной СК и высоту цилиндра

const MbCartPoint3D pс1( 0, 10, 0 );

// Вектор p0-p2 задает ось X локальной СК и радиус основания

const MbCartPoint3D pс2( 5, 0, 0 );

MbCylinderSurface\* pBaseSurf = new MbCylinderSurface( pс0, pс1, pс2 );

// Точки на поверхности цилиндра с координатами (u, v) для указания на нем

// равнобедренного треугольника.

// Параметр v отсчитывается по высоте цилиндра в диапазоне [0, 1].

// Параметр u отсчитывается по окружности цилиндра в диапазоне [0, 2\*M\_PI].

// Координаты точек выбраны так, чтобы высота треугольника составляла половину

// цилиндра (с отступами по четверти высоты от краев).

// Длина основания треугольника составляет M\_PI/2, т.е., четверть

// от окружности цилиндра.

MbCartPoint p1(M\_PI/4, 0.25), p2(M\_PI/4 + M\_PI/2, 0.25), p3(M\_PI/4 + M\_PI/4, 0.75 );

MbLineSegment\* pSeg1 = new MbLineSegment( p1, p3 );

MbLineSegment\* pSeg2 = new MbLineSegment( p3, p2 );

MbLineSegment\* pSeg3 = new MbLineSegment( p2, p1 );

// Стороны треугольника объединяются в замкнутый контур.

MbContour\* pContourTriangle = new MbContour();

pContourTriangle->AddSegment( pSeg1 );

pContourTriangle->AddSegment( pSeg2 );

pContourTriangle->AddSegment( pSeg3 );

// Построение пространственной кривой "Контур на поверхности" с указанием в качестве

// базовой поверхности цилиндра pBaseSurf

MbContourOnSurface\* pContourOnCyl = new MbContourOnSurface(

\*pBaseSurf, \*pContourTriangle, false );

// Построение поверхности, ограниченной кривыми

MbCurveBoundedSurface\* pSurf = new MbCurveBoundedSurface( \*pBaseSurf );

// В качестве кривых указываются две границы: внешний контур поверхности и

// внутренний треугольный контур pContourOnCyl

pSurf->AddOuterContour();

pSurf->AddCurve( \*pContourOnCyl );

pSurf->CalculateUVLimits();

// Отображение поверхности

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

// Отображение контуров, ограничивающих поверхность

for (int i = 0; i<pSurf->GetCurvesCount(); i++)

viewManager->AddObject(Style(5, LIGHTRED), (MbSpaceItem\*)(pSurf->GetCurve(i)) );

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pBaseSurf );

::DeleteItem( pSurf );

::DeleteItem( pSeg1 );

::DeleteItem( pSeg2 );

::DeleteItem( pSeg3 );

::DeleteItem( pContourTriangle );

::DeleteItem( pContourOnCyl );

}

**49. MbMeshSurface – сеточная поверхность**

#include "surf\_mesh\_surface.h" // MbMeshSurface - сеточная поверхность

#include "cur\_bezier3d.h"

void MakeUserCommand0()

{

// 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] );

}

**50. MbGridSurface – поверхность на базе триангулированной сетки**

// MbGridSurface - поверхность, заданная триангулированной сеткой

#include "surf\_grid\_surface.h"

// MbGrid - триангулированная сетка

#include "mesh\_grid.h"

#include "surf\_sphere\_surface.h"

#include "cur\_line\_segment3d.h"

// Алгоритмы построения поверхностей

#include "action\_surface.h"

void MakeUserCommand0()

{

// Базовая поверхность - сферическая поверхность (построение по центру и радиусу)

MbSphereSurface\* pBaseSurf = new MbSphereSurface( MbCartPoint3D(0,0,0), 10 );

// Вычисление триангулированной сетки для базовой поверхности

MbGrid\* pGrid = new MbGrid;

MbStepData stepData( ist\_ParamStep, 0.5 );

pBaseSurf->CalculateSurfaceGrid( stepData, true, \*pGrid );

// Построение триангулированной поверхности по данным вычисленной сетки

MbSurface\* pSurf = NULL;

::GridSurface( \*pGrid, pSurf );

// Отображение поверхности

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

// Отображение треугольников сеточного представления

{

const int trCnt = pGrid->GetTriangles().size();

const SArray<MbFloatPoint3D>& arrPnts = pGrid->GetPoints();

for (int t = 0; t < trCnt; t++)

{

const MbTriangle& tr = pGrid->GetTriangle(t);

int idx0 = tr.GetIndex(0);

int idx1 = tr.GetIndex(1);

int idx2 = tr.GetIndex(2);

MbCartPoint3D p0, p1, p2;

arrPnts[idx0].GetCartPoint( p0 ) ;

arrPnts[idx1].GetCartPoint( p1 ) ;

arrPnts[idx2].GetCartPoint( p2 ) ;

MbLineSegment3D\* pSeg1 = new MbLineSegment3D( p0, p1 );

MbLineSegment3D\* pSeg2 = new MbLineSegment3D( p1, p2 );

MbLineSegment3D\* pSeg3 = new MbLineSegment3D( p2, p0 );

viewManager->AddObject( Style(5, LIGHTRED), pSeg1 );

viewManager->AddObject( Style(5, LIGHTRED), pSeg2 );

viewManager->AddObject( Style(5, LIGHTRED), pSeg3 );

}

}

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pBaseSurf );

::DeleteItem( pGrid );

::DeleteItem( pSurf );

}

**51.MfOffsetSurface – эквидистантная поверхность**

// MbOffsetSurface - эквидистантная поверхность для некоторой базовой поверхности

#include "surf\_offset\_surface.h"

// MbRevolutionSurface - поверхность вращения

#include "surf\_revolution\_surface.h"

// MbAxis3D - ось в трехмерном пространстве (вектор с закрепленной начальной точкой)

#include "mb\_axis3d.h"

#include "cur\_bezier3d.h"

void MakeUserCommand0()

{

// 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);

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

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pGenCurve);

::DeleteItem( pBaseSurf );

::DeleteItem( pOffSurf );

}

**52. Построение поверхности выдавливания**

#include "surf\_extrusion\_surface.h"

#include "cur\_line\_segment3d.h"

#include "cur\_polyline3d.h"

#include "action\_surface.h"

void MakeUserCommand0()

{

// Параметры модели коробки

// Ширина, высота и глубина - размеры вдоль осей X, Y и Z

const double BOX\_WIDTH = 10, BOX\_HEIGHT = 2.5, BOX\_DEPTH = 15;

// Ширина поперечины - 20% от глубины коробки

const double BAR\_SIZE = BOX\_DEPTH/5;

// 1) Построение боковой поверхности

MbSurface\* pSurfBox = NULL;

// Вершины прямоугольника ABCD (дно коробки)

std::vector<MbCartPoint3D> arrPolyPntsBox = { { 0, 0, 0 }, { BOX\_WIDTH, 0, 0 },

{ BOX\_WIDTH, 0, BOX\_DEPTH }, { 0, 0, BOX\_DEPTH } };

// Образующая для построения боковой поверхности - контур дна коробки

// в виде ломаной линии

MbPolyline3D\* pGenCurveBox = new MbPolyline3D( arrPolyPntsBox, true );

// Направляющий вектор - параллелен оси Y, длина равна высоте коробки CC1

MbVector3D vecDirBox( 0, BOX\_HEIGHT, 0 );

// Вызов функции-утилиты для построения поверхности выдавливания

MbResultType resBox = ::ExtrusionSurface( \*pGenCurveBox, vecDirBox, true, pSurfBox );

// 2) Построение дна коробки - прямоугольника ABCD

MbSurface\* pSurfBottom = NULL;

// Образующая - отрезок AB

MbLineSegment3D\* pGenCurveBottom =

new MbLineSegment3D( arrPolyPntsBox[0], arrPolyPntsBox[1] );

// Направляющий вектор - параллелен оси Z, длина равна глубине коробки BC

MbVector3D vecDirBottom( 0, 0, BOX\_DEPTH );

MbResultType resBottom = ::ExtrusionSurface( \*pGenCurveBottom,

vecDirBottom, true, pSurfBottom );

// 3) Построение поперечного элемента

MbSurface\* pSurfBar = NULL;

// Образующая - отрезок EF - строится посредством параллельного переноса

// отрезка AB на вектор AE

MbLineSegment3D\* pGenCurveBar =

new MbLineSegment3D( arrPolyPntsBox[0], arrPolyPntsBox[1] );

pGenCurveBar->Move( MbVector3D(0, BOX\_HEIGHT, (BOX\_DEPTH - BAR\_SIZE)/2 ) );

// Направляющий вектор - параллелен оси Z, длина равна ширине поперечины

MbVector3D vecDirBar( 0, 0, BAR\_SIZE );

MbResultType resBar = ::ExtrusionSurface( \*pGenCurveBar, vecDirBar, true, pSurfBar );

// Отображение поверхностей

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

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

viewManager->AddObject(Style(1, BLUE), pSurfBar );

// Уменьшение счетчика ссылок динамически созданных объектов ядра

::DeleteItem( pGenCurveBox);

::DeleteItem( pSurfBox );

::DeleteItem( pGenCurveBottom );

::DeleteItem( pSurfBottom );

::DeleteItem( pGenCurveBar );

::DeleteItem( pSurfBar );

}

**53.Поверхность выдавливания с отверстием**

#include "surf\_extrusion\_surface.h"

#include "cur\_line\_segment3d.h"

#include "cur\_contour.h"

#include "cur\_arc3d.h"

#include "action\_surface.h"

#include "action\_curve.h"

void MakeUserCommand0()

{

// Размер стороны поверхностного фрагмента в виде квадрата

const double SIZE\_SQUARE = 5;

// Радиус внутреннего центрального отверстия

const double HOLE\_RADIUS = SIZE\_SQUARE/3;

// 1) ПОСТРОЕНИЕ БАЗОВОЙ ПОВЕРХНОСТИ ВЫДАВЛИВАНИЯ

MbSurface\* pSurfBase = 0;

{

// Образующая: отрезок на оси X длиной SIZE\_SQUARE

MbLineSegment3D\* pGenCurve =

new MbLineSegment3D(MbCartPoint3D(0, 0, 0), MbCartPoint3D(SIZE\_SQUARE,0,0));

// Направляющий вектор - направлен в сторону +Z

MbVector3D vecDir( 0, 0, SIZE\_SQUARE );

// Вызов функции-утилиты для построения поверхности выдавливания

::ExtrusionSurface( \*pGenCurve, vecDir, true, pSurfBase );

}

// 2) ПОСТРОЕНИЕ КРУГОВОГО КОНТУРА НА ПОВЕРХНОСТИ

MbContour\* pArcContour = 0;

{

// Центр отверстия и пара точек на окружности в глобальной СК

MbCartPoint3D pc( SIZE\_SQUARE/2, 0, SIZE\_SQUARE/2 );

MbCartPoint3D p1 = pc, p2 = pc;

p1.x += HOLE\_RADIUS;

p2.z += HOLE\_RADIUS;

// Вспомогательная пространственная кривая - контур отверстия

MbArc3D\* pArcCurve = new MbArc3D( pc, p1, p2, 0, true );

// Проецирование пространственной кривой на поверхность для получения

// контура на поверхности

::SurfaceBoundContour( \*pSurfBase, \*pArcCurve, Math::DefaultMathVersion(),

pArcContour );

::DeleteItem( pArcCurve );

}

// 3) ПОЛУЧЕНИЕ КОНТУРА - ВНЕШНЕЙ ГРАНИЦЫ БАЗОВОЙ ПОВЕРХНОСТИ

MbContour\* pExtContour = &pSurfBase->MakeContour(true);

// 4) ПОСТРОЕНИЕ ПОВЕРХНОСТИ, ОГРАНИЧЕННОЙ ПАРОЙ КОНТУРОВ

MbSurface\* pSurf = 0;

{

RPArray<MbCurve> arrBounds;

arrBounds.Add( pExtContour );

arrBounds.Add( pArcContour );

// Построение поверхности pSurf на базе поверхности выдавливания pSurfBase

::BoundedSurface( \*pSurfBase, arrBounds, pSurf );

}

// Отображение поверхности

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pSurfBase );

::DeleteItem( pSurf );

::DeleteItem( pArcContour );

::DeleteItem( pExtContour );

}

**54.Поверхность вращения в виде комбинации сферической и цилиндрической поверхности**

#include "cur\_line\_segment3d.h"

#include "cur\_arc3d.h"

#include "cur\_contour3d.h"

#include "action\_surface.h"

#include "surf\_revolution\_surface.h"

// Построение поверхности вращения в виде комбинации цилиндра и сферы с

// совпадающими центрами тяжести.

MbSurface\* CreateCylSphereSurface()

{

// Параметры цилиндра и сферы

const double CYL\_RAD = 2.0;

const double CYL\_HEIGHT = 20.0;

const double SPHERE\_RAD = 5.0;

// Образующая кривая: контур, состоящий из двух отрезков и дуги окружности между ними

// Точки - вершины сегментов образующей (рис. 4)

MbCartPoint3D p1( CYL\_RAD, CYL\_HEIGHT/2, 0 );

MbCartPoint3D p2( CYL\_RAD, sqrt( SPHERE\_RAD\*SPHERE\_RAD - CYL\_RAD\*CYL\_RAD ), 0 );

MbCartPoint3D p3( SPHERE\_RAD, 0, 0 );

MbCartPoint3D p4( p2.x, -p2.y, p2.z );

MbCartPoint3D p5( p1.x, -p1.y, p1.z );

// Сегменты образующей кривой

MbLineSegment3D\* pSeg1 = new MbLineSegment3D( p1, p2 );

MbArc3D\* pArc = new MbArc3D( p2, p3, p4, 1, false );

MbLineSegment3D\* pSeg2 = new MbLineSegment3D( p4, p5 );

// Построение образующей кривой в виде контура из трех сегментов

RPArray<MbCurve3D> arrCurves;

arrCurves.Add( pSeg1 );

arrCurves.Add( pArc );

arrCurves.Add( pSeg2 );

MbContour3D\* pGenContour = new MbContour3D( arrCurves, true );

// Параметры оси вращения - она совпадает с осью цилиндра и осью Y локальной СК,

// в которой задавались вершины сегментов образующей кривой p1-p5.

MbCartPoint3D axOrg( 0, 0, 0 );

MbVector3D axDir( 0, 1, 0 );

// Построение поверхности вращения

MbSurface\* pSurf = NULL;

MbResultType res = ::RevolutionSurface( \*pGenContour, axOrg, axDir,

2\*M\_PI, true, pSurf );

if ( res != rt\_Success )

{

// Возврат в случае ошибки при построении поверхности

::DeleteItem( pGenContour );

::DeleteItem( pSurf );

return NULL;

}

::DeleteItem( pGenContour );

return pSurf;

}

void MakeUserCommand0()

{

// Построение поверхности реализовано в отдельной функции

MbSurface\* pSurf = CreateCylSphereSurface();

// Отображение поверхности

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

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pSurf );

}

**55. Модель из двух поверхностей вращения и двух линейчатых поверхностей**

#include "cur\_line\_segment3d.h"

#include "cur\_polyline3d.h"

#include "cur\_contour3d.h"

#include "cur\_arc3d.h"

#include "cur\_contour.h"

#include "action\_surface.h"

#include "surf\_revolution\_surface.h"

#include "action\_surface\_curve.h"

#include "action\_curve.h"

void MakeUserCommand0()

{

// Параметры прямоугольной призмы, которая будет дополнять поверхность pCylSphere

const double PRISM\_SIDE = 5; // Сторона квадратного основания призмы

const double PRISM\_HEIGHT = 20; // Высота призмы

// 1) ПОСТРОЕНИЕ ЧАСТИ МОДЕЛИ В ВИДЕ КОМБИНАЦИИ СФЕРИЧЕСКОЙ И ЦИЛИНДРИЧЕСКОЙ

// ПОВЕРХНОСТЕЙ (вызывается функция, реализованная в примере 4.1)

MbSurface\* pCylSphere = CreateCylSphereSurface();

MbAxis3D axVert( MbVector3D( 0, 1, 0 ) );

pCylSphere->Rotate( axVert, M\_PI/2 );

// 2) ПОСТРОЕНИЕ ПАРЫ КОНТУРОВ, ОГРАНИЧИВАЮЩИХ ОДНУ ИЗ ЧАСТЕЙ ПРИЗМЫ СНАРУЖИ СФЕРЫ

// Координаты вершин основания призмы (в плоскости, параллельной ZY, и

// смещенной на половину высоты в сторону -X)

std::vector<MbCartPoint3D> arrPnts\_Base1 = {

{ -PRISM\_HEIGHT/2, PRISM\_SIDE/2, -PRISM\_SIDE/2 },

{ -PRISM\_HEIGHT/2, PRISM\_SIDE/2, PRISM\_SIDE/2 },

{ -PRISM\_HEIGHT/2, -PRISM\_SIDE/2, PRISM\_SIDE/2 },

{ -PRISM\_HEIGHT/2, -PRISM\_SIDE/2, -PRISM\_SIDE/2 }

};

// Контур - замкнутая ломаная для представления основания призмы

MbPolyline3D\* pContour\_Base1 = new MbPolyline3D( arrPnts\_Base1, true );

// Проецирование пространственного контура pContour\_Base1 на поверхность pCylSphere

RPArray< MbCurve3D > arrProjCurves\_Base1;

MbVector3D dirProj1( +1, 0, 0 );

::CurveProjection( \*pCylSphere, \*pContour\_Base1, &dirProj1, false, false,

arrProjCurves\_Base1 );

// 3) ПОСТРОЕНИЕ ПЕРВОГО ФРАГМЕНТА ПОВЕРХНОСТИ ПРИЗМЫ

// Поверхность pSurfPrism1 строится как линейчатая поверхность, заданная парой кривых

MbSurface\* pSurfPrism1 = 0;

::RuledSurface( \*pContour\_Base1, \*arrProjCurves\_Base1[0], true, pSurfPrism1 );

// 4) ПОСТРОЕНИЕ ВТОРОГО ФРАГМЕНТА ПОВЕРХНОСТИ ПРИЗМЫ

// Второе основание призмы смещено относительно первого основания на высоту призмы

MbPolyline3D\* pContour\_Base2 = new MbPolyline3D( arrPnts\_Base1, true );

pContour\_Base2->Move( MbVector3D( PRISM\_HEIGHT, 0, 0) );

// Проецирование пространственного контура pContour\_Base2 на поверхность pCylSphere

RPArray< MbCurve3D > arrProjCurves\_Base2;

MbVector3D dirProj2( -1, 0, 0 );

::CurveProjection( \*pCylSphere, \*pContour\_Base2, &dirProj2, false, false,

arrProjCurves\_Base2 );

// Построение второго фрагмента поверхности призмы как линейчатой поверхности,

// заданной парой кривых

MbSurface\* pSurfPrism2 = 0;

::RuledSurface( \*pContour\_Base2, \*arrProjCurves\_Base1[0], true, pSurfPrism2 );

// 5) ПОСТРОЕНИЕ ОТВЕРСТИЙ НА БАЗОВОЙ ПОВЕРХНОСТИ pCylSphere, СООТВЕТСТВУЮЩИХ

// ЛИНИЯМ ПЕРЕСЕЧЕНИЯ ПРИЗМЫ СО СФЕРОЙ

// Построение поверхности, ограниченной тремя контурами

MbSurface\* pSurfWithHoles = 0;

RPArray<MbCurve> arrBounds;

{

// Проецирование пространственных кривых - линий пересечения призмы со

// сферой - на базовую поверхность для получения двумерных контуров на

// базовой поверхности

MbContour\* pProjContour1 = NULL;

:SurfaceBoundContour( \*pCylSphere, \*arrProjCurves\_Base1[0],

Math::DefaultMathVersion(), pProjContour1 );

MbContour\* pProjContour2 = NULL;

::SurfaceBoundContour( \*pCylSphere, \*arrProjCurves\_Base2[0],

Math::DefaultMathVersion(), pProjContour2 );

// Получение контура - внешней границы базовой поверхности

MbContour\* pExtContour = &pCylSphere->MakeContour(true);

// Построение поверхности с отверстиями pSurfWithHoles на базе pCylSphere

arrBounds.Add( pExtContour );

arrBounds.Add( pProjContour1 );

arrBounds.Add( pProjContour2 );

::BoundedSurface( \*pCylSphere, arrBounds, pSurfWithHoles );

}

// 6) ОТОБРАЖЕНИЕ ГЕОМЕТРИЧЕСКИХ ОБЪЕКТОВ

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

viewManager->AddObject( Style(3, LIGHTRED), pContour\_Base1 );

viewManager->AddObject( Style(3, LIGHTRED), arrProjCurves\_Base1[0] );

viewManager->AddObject( Style(3, LIGHTRED), pContour\_Base2 );

viewManager->AddObject( Style(3, LIGHTRED), arrProjCurves\_Base2[0] );

viewManager->AddObject( Style(3, LIGHTGRAY), pSurfPrism1 );

viewManager->AddObject( Style(3, LIGHTGRAY), pSurfPrism2 );

// Уменьшение счетчиков ссылок динамически созданных объектов ядра

::DeleteItem( pCylSphere );

::DeleteItem( pContour\_Base1 );

for (int i=0; i<arrProjCurves\_Base1.size(); i++)

::DeleteItem(arrProjCurves\_Base1[i]);

::DeleteItem( pSurfPrism1 );

::DeleteItem( pContour\_Base2 );

for (int i=0; i<arrProjCurves\_Base2.size(); i++)

::DeleteItem(arrProjCurves\_Base2[i]);

::DeleteItem( pSurfPrism2 );

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

::DeleteItem(arrBounds[i]);

::DeleteItem( pSurfWithHoles );

}

**56. Тонкостенная твердотельная модель в виде комбинации фрагментов тел вращения и выдавливания**

#include "cur\_polyline3d.h"

#include "cur\_arc3d.h"

#include "action\_solid.h"

// Вспомогательная функция.

// Построение тонкостенного твердого тела вращения в виде комбинации

// цилиндра и сферы с центрами тяжести в начале координат.

MbSolid\* CreateCylSphereSolid( double sphereRad, double cylRad, double cylHeight,

double thwall /\* толщина стенки \*/ )

{

// Образующая кривая: контур, состоящий из двух отрезков и дуги окружности между ними

// Точки - вершины сегментов образующей

MbCartPoint3D p1( cylRad, cylHeight/2, 0 );

MbCartPoint3D p2( cylRad, sqrt( sphereRad\*sphereRad - cylRad\*cylRad ), 0 );

MbCartPoint3D p3( sphereRad, 0, 0 );

MbCartPoint3D p4( p2.x, -p2.y, p2.z );

MbCartPoint3D p5( p1.x, -p1.y, p1.z );

// Сегменты образующей кривой

MbLineSegment3D\* pSeg1 = new MbLineSegment3D( p1, p2 );

MbArc3D\* pArc = new MbArc3D( p2, p3, p4, 1, false );

MbLineSegment3D\* pSeg2 = new MbLineSegment3D( p4, p5 );

// Построение образующей кривой в виде контура из трех сегментов

RPArray<MbCurve3D> arrCurves;

arrCurves.Add( pSeg1 );

arrCurves.Add( pArc );

arrCurves.Add( pSeg2 );

MbContour3D\* pGenContour = new MbContour3D( arrCurves, true );

// Формирование объекта-образующей sweptData

RPArray< MbContour3D > arrContours;

arrContours.Add( pGenContour );

MbSweptData sweptData(arrContours);

// Параметры оси вращения - ось направлена вдоль оси Y

MbCartPoint3D axOrg( 0, 0, 0 );

MbVector3D axDir( 0, 1, 0 );

MbAxis3D axRot( axOrg, axDir );

// Именователь граней твердого тела вращения

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

operNames.SetVersion( MbNameVersion() );

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

// Параметры операции вращения для построения тонкостенного тела

RevolutionValues params;

params.shape = 0; // Построение тела типа "сфера"

params.side1.scalarValue = 360\*M\_PI/180; // Угол поворота образующей

params.thickness1 = thwall; // Толщина стенки

// Вызов операции построения тела вращения pSolid

MbSolid\* pSolid = NULL;

MbResultType res = ::RevolutionSolid( sweptData, axRot, params,

operNames, cNames, pSolid );

if ( res != rt\_Success )

{

// Возврат в случае ошибки при построении твердого тела

::DeleteItem( pGenContour );

::DeleteItem( pSolid );

return NULL;

}

// Возврат построенного твердого тела

return pSolid;

}

// Вспомогательная функция.

// Построение тонкостенного твердого тела выдавливания, представляющего боковую

// поверхность призмы с квадратным основанием

MbSolid\* CreatePrismSolid( double prismSide, double prismHeight, double thwall )

{

// Координаты вершин основания призмы (в плоскости ZY - в среднем сечении призмы)

std::vector<MbCartPoint3D> arrPnts\_Base1 = {

{ 0, prismSide/2, -prismSide/2 },

{ 0, prismSide/2, prismSide/2 },

{ 0, -prismSide/2, prismSide/2 },

{ 0, -prismSide/2, -prismSide/2 }

};

// Контур - замкнутая ломаная для представления основания призмы

MbPolyline3D\* pContour\_Base1 = new MbPolyline3D( arrPnts\_Base1, true );

// Построение образующей кривой

RPArray<MbCurve3D> arrCurves;

arrCurves.Add( pContour\_Base1 );

MbContour3D\* pGenContour = new MbContour3D( arrCurves, true );

// Формирование объекта-образующей sweptData

RPArray< MbContour3D > arrContours;

arrContours.Add( pGenContour );

MbSweptData sweptData(arrContours);

// Направление выдавливания

MbVector3D vecDir( 1, 0, 0 );

// Именователь граней твердого тела

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

operNames.SetVersion( MbNameVersion() );

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

// Параметры операции выдавливания.

// Выдавливание производится симметрично в обе стороны от образующей.

ExtrusionValues params( prismHeight/2, prismHeight/2 );

params.thickness1 = thwall; // Толщина стенки

// Вызов операции выдавливания для построения твердого тела pSolid

MbSolid\* pSolid = NULL;

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

params, operNames, cNames, pSolid );

if ( res != rt\_Success )

{

// Возврат в случае ошибки при построении твердого тела

::DeleteItem( pGenContour );

::DeleteItem( pSolid );

return NULL;

}

// Возврат построенного твердого тела

return pSolid;

}

// Основная функция.

// Построение твердотельной тонкостенной модели в виде комбинации

// тел вращения и выдавливания

void MakeUserCommand0()

{

// 1) ПОСТРОЕНИЕ ПЕРВОГО ВСПОМОГАТЕЛЬНОГО ТОНКОСТЕННОГО ТВЕРДОГО ТЕЛА

// ВРАЩЕНИЯ – КОМБИНАЦИЯ ЦИЛИНДРА И СФЕРЫ

const double CYL\_RAD = 2.0; // Радиус цилиндрической части

const double CYL\_HEIGHT = 20.0; // Высота цилиндрической части

const double SPHERE\_RAD = 5.0; // Радиус сферической части

const double THICKNESS = 0.1; // Толщина тонкой стенки

MbSolid\* pCylSphereSolid = CreateCylSphereSolid( SPHERE\_RAD, CYL\_RAD,

CYL\_HEIGHT, THICKNESS );

// Возможный отладочный вызов для проверки выполнения шага 1)

// if ( pCylSphereSolid )

// TestVariables::viewManager->AddObject( TestVariables::SOLID\_Style,

// pCylSphereSolid );

// 2) ПОСТРОЕНИЕ ВТОРОГО ВСПОМОГАТЕЛЬНОГО ТОНКОСТЕННОГО ТВЕРДОГО ТЕЛА ВЫДАВЛИВАНИЯ –

// БОКОВАЯ ПОВЕРХНОСТЬ ПРИЗМЫ С КВАДРАТНЫМ ОСНОВАНИЕМ

const double PRISM\_SIDE = 5; // Сторона квадратного основания призмы

const double PRISM\_HEIGHT = 20; // Высота призмы

MbSolid\* pPrismSolid = CreatePrismSolid( PRISM\_SIDE, PRISM\_HEIGHT, THICKNESS );

// Возможный отладочный вызов для проверки выполнения шага 2)

// if ( pPrismSolid )

// TestVariables::viewManager->AddObject(TestVariables::SOLID\_Style, pPrismSolid);

// 3) РАЗБИЕНИЕ ПРИЗМАТИЧЕСКОГО ТЕЛА pPrismSolid НА ТРИ ЧАСТИ: ДВЕ ЧАСТИ СНАРУЖИ

// И ОДНА ВНУТРИ pCylSphereSolid.

// ВНЕШНИЕ ЧАСТИ ОТБИРАЮТСЯ ДЛЯ ИСПОЛЬЗОВАНИЯ ПРИ ПОСТРОЕНИИ РЕЗУЛЬТИРУЮШЕГО ТЕЛА

MbSolid\* pPrismPart1 = NULL;

MbSolid\* pPrismPart2 = NULL;

// Именователь граней для построения тела с помощью булевой операции

MbSNameMaker operBoolNames( ct\_BooleanSolid, MbSNameMaker::i\_SideNone, 0 );

// Флаги операции: построение замкнутого тела с объединением подобных граней и ребер

MbBooleanFlags flagsBool;

flagsBool.InitBoolean( true );

flagsBool.SetMergingFaces( true );

flagsBool.SetMergingEdges( true );

{

// Булева операция - разность тел: pSolidDiff = pPrismSolid - pCylSphereSolid

// Для выполнения разности генерируется вспомогательное тело pCylSphereSolidSmall,

// у которого радиус сферической части меньше, чем у pCylSphereSolid, на

// толщину стенки.

// Это сделано для гарантированного последующего объединения частей тел в

// дальнейшем, на шаге 5).

MbSolid\* pCylSphereSolidSmall = CreateCylSphereSolid( SPHERE\_RAD - THICKNESS,

CYL\_RAD, CYL\_HEIGHT, THICKNESS );

MbSolid\* pSolidDiff = NULL;

MbResultType res = ::BooleanResult( \*pPrismSolid, cm\_Copy, \*pCylSphereSolidSmall,

cm\_Copy, bo\_Difference, flagsBool, operBoolNames, pSolidDiff );

::DeleteObject( pCylSphereSolidSmall );

// Возможный отладочный вызов для проверки выполнения шага 3)

// if ( pSolidDiff )

// TestVariables::viewManager->AddObject( TestVariables::SOLID\_Style,

// pSolidDiff );

// Разбиение составного тела pSolidDiff на три тела: два снаружи и одно

// внутри pCylSphereSolid

if ( res == rt\_Success )

{

RPArray<MbSolid> parts;

MbSNameMaker detachNames( ct\_DetachSolid, MbSNameMaker::i\_SideNone, 0 );

// При разделении тела pSolidDiff наибольшая часть остается в pSolidDiff,

// а меньшие (partsCnt штук) помещаются в массив parts (в порядке

// убывания габаритного размера)

size\_t partsCnt = ::DetachParts( \*pSolidDiff, parts, true, detachNames );

if ( partsCnt == 2 )

{

// Отбор внешних частей тела pSolidDiff

pPrismPart1 = parts[0];

pPrismPart2 = parts[1];

// Удаление лишней части, которая находится внутри тела выдавливания

::DeleteItem( pSolidDiff );

}

}

}

// 4) ВЫРЕЗАНИЕ КВАДРАТНЫХ ОТВЕРСТИЙ ИЗ pCylSphereSolid

MbSolid\* pCylSphereHolesSolid = NULL;

{

// Булева операция - разность тел: pSolidDiff = pCylSphereSolid - pPrismSolid

MbSolid\* pSolidDiff = NULL;

MbResultType res = ::BooleanResult( \*pCylSphereSolid, cm\_Copy, \*pPrismSolid,

cm\_Copy, bo\_Difference, flagsBool, operBoolNames, pSolidDiff );

// Возможный отладочный вызов для проверки выполнения шага 4)

// if ( pSolidDiff )

// TestVariables::viewManager->AddObject(TestVariables::SOLID\_Style, pSolidDiff);

// Разбиение составного тела pSolidDiff на три тела

if ( res == rt\_Success )

{

RPArray<MbSolid> parts;

MbSNameMaker detachNames( ct\_DetachSolid, MbSNameMaker::i\_SideNone, 0 );

size\_t partsCnt = ::DetachParts( \*pSolidDiff, parts, true, detachNames );

if ( partsCnt == 2 )

{

// Сохранение части, которая будет использоваться для результирующей модели

pCylSphereHolesSolid = pSolidDiff;

// Удаление частей, соответствующих квадратным отверстиям в теле вращения

::DeleteItem( parts[0] );

::DeleteItem( parts[1] );

}

}

}

// 5) ПОСТРОЕНИЕ РЕЗУЛЬТИРУЮЩЕГО ТЕЛА pResSolid: ОБЪЕДИНЕНИЕ ТРЕХ ТЕЛ

// pCylSphereHolesSolid, pPrismPart1 И pPrismPart2

MbSolid\* pResSolid = NULL;

{

// Построение промежуточного тела: pSolid1 = pCylSphereHolesSolid + pPrismPart1

MbSolid\* pSolid1 = NULL;

::BooleanResult( \*pCylSphereHolesSolid, cm\_Copy, \*pPrismPart1, cm\_Copy, bo\_Union,

flagsBool, operBoolNames, pSolid1 );

// Построение результирующего тела: pResSolid = pSolid1 + pPrismPart2

::BooleanResult( \*pSolid1, cm\_Copy, \*pPrismPart2, cm\_Copy, bo\_Union,

flagsBool, operBoolNames, pResSolid );

// Уменьшение счетчика ссылок промежуточного тела, которое больше не потребуется

::DeleteItem( pSolid1 );

}

// ОТОБРАЖЕНИЕ РЕЗУЛЬТИРУЮЩЕГО ТВЕРДОГО ТЕЛА

if ( pResSolid )

TestVariables::viewManager->AddObject( TestVariables::SOLID\_Style, pResSolid );

// УМЕНЬШЕНИЕ СЧЕТЧИКА ССЫЛОК ДИНАМИЧЕСКИ СОЗДАННЫХ ОБЪЕКТОВ ЯДРА

::DeleteItem( pCylSphereSolid );

::DeleteItem( pPrismSolid );

::DeleteItem( pPrismPart1 );

::DeleteItem( pPrismPart2 );

::DeleteItem( pCylSphereHolesSolid );

::DeleteItem( pResSolid );

}

**57. Построение поверхности сдвига постоянного сечения**

#include <cur\_bezier.h>

#include <cur\_bezier3d.h>

void MakeUserCommand0()

{

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

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

MbPlacement3D plSpine; // СК для построения направляющей

// 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 двумерной направляющей - кривой Безье

SArray<MbCartPoint> arrPnts(4);

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

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

arrPnts.Add(MbCartPoint(70, -30));

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

// Используется конструктор для построения по четырем контрольным точкам

MbBezier\* pSpine = new MbBezier(arrPnts);

// Поворот локальной системы координат вокруг оси Y мировой системы координат до

// совмещения плоскости XY локальной СК с плоскостью XZ мировой СК

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

-90\*DEG\_TO\_RAD);

// Построение трехмерной кривой Безье на основе двумерной с использованием

// конструктора, принимающего на вход двумерную кривую и локальную

// систему координат строящейся трехмерной кривой

MbBezier3D\* pBezier = new MbBezier3D(\*pSpine, plSpine);

// Вызов функции построения поверхности сдвига

MbSurface\* pSurface = NULL;

::ExpansionSurface(\*pArc, \*pBezier, NULL, pSurface);

// Отображение полученной поверхности

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

::DeleteItem(pSurface);

::DeleteItem(pBezier);

::DeleteItem(pSpine);

::DeleteItem(pArc);

::DeleteItem(pArc2D);

}

**58. Построение поверхности сдвига временного сечения**

void MakeUserCommand0()

{

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

MbPlacement3D plArc1; // СК для построения первой образующей (совпадает с мировой)

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

MbPlacement3D plArc2; // СК для построения второй образующей (вычисляется далее)

// Построение двумерной образующей кривой - окружности

const double RAD = 10;

const MbCartPoint arcCenter(0, 0);

// Построение окружности на плоскости по центру и радиусу

MbArc\* pArc2D\_1 = new MbArc(arcCenter, RAD);

// Построение окружности в трехмерном пространстве

MbArc3D\* pArc1 = new MbArc3D(\*pArc2D\_1, plArc1);

// Построение двумерной направляющей - кривой Безье

SArray<MbCartPoint> arrPnts(4);

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

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

arrPnts.Add(MbCartPoint(70, -30));

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

// Используется конструктор для построения по четырем контрольным точкам

MbBezier\* pSpine = new MbBezier(arrPnts);

// Поворот локальной системы координат вокруг оси Y мировой системы координат до

// совмещения плоскости XY локальной СК с плоскостью XZ мировой СК

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

-90 \* DEG\_TO\_RAD);

// Построение трехмерной кривой Безье на основе двумерной двухмерной с использованием

// конструктора, принимающего на вход двумерную кривую и локальную систему

// координат строящейся трехмерной кривой

MbBezier3D\* pBezier = new MbBezier3D(\*pSpine, plSpine);

// Перенос локальной системы координат второй окружности в

// конечную точку направляющей кривой

plArc2.Move(MbVector3D(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, 0, 100)));

// Построение второй образующей - окружности в конце получаемой поверхности

MbArc\* pArc2D\_2 = new MbArc(arcCenter, RAD\*3);

MbArc3D\* pArc2 = new MbArc3D(\*pArc2D\_2, plArc2);

// Вызов функции построения поверхности сдвига

MbSurface\* pSurface = NULL;

::ExpansionSurface(\*pArc1, \*pBezier, pArc2, pSurface);

// Отображение полученной поверхности

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

::DeleteItem(pSurface);

::DeleteItem(pBezier);

::DeleteItem(pSpine);

::DeleteItem(pArc2D\_1);

::DeleteItem(pArc1);

::DeleteItem(pArc2D\_2);

::DeleteItem(pArc2);

}

**59. Построение кинематической поверхности**

void MakeUserCommand0()

{

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);

}