Getting started with C3D Modeler

LESSON 7
Operations for constructing solids of arbitrary shape

Table of Contents

1. Introduction

In the previous lesson, the simplest methods for constructing solid models were considered: the construction of elementary bodies, extrusion and rotation bodies. In addition to these operations, the operations of constructing kinematic bodies and the construction of bodies by cross sections are often used. Similar operations for the construction of surfaces were considered in the Lesson 5 "Construction of swept surfaces".

Kinematic bodies are formed as a result of the movement of a certain contour (generating curve) along a given trajectory, which is described with a guide curve. The resulting body is a volume filled ("swept out") by the generating curve as the guide moves. Therefore, kinematic bodies are also called swept solids. The extrusion and rotation bodies considered in the previous lesson are special cases of swept solids. In the case of an extrusion body, the guide curve is a straight line segment, and in the case of a body of revolution, an arc of a circle or a circle.

Although swept solids make it possible to obtain solid models of a more complex shape than extrusion and rotation bodies, they are also not universal. Body sweeping is convenient for modeling bodies with a constant cross section - both solid and thin-walled. If the section of the body is not constant, the operation of constructing the body in cross sections can be used. By selecting the number of sections, their location and shape, you can achieve the required accuracy of the representation of the simulated body. The criterion of satisfactory performance, as a rule, depends on the characteristics of the problem being solved. For example, for illustrative purposes, the quality of modeling can be assessed visually, and when building industrial models based on numerical values of geometric properties.

As the body construction methods become more versatile — the extrusion body, the rotation body, the sweeping body and the body constructed over sections — the interface of the corresponding C3D functions becomes more complex to accommodate an increasing number of possible parameters. But the general structure of the function prototypes and the assignment of parameters remain similar for all operations of constructing solids.

In practice, even the most common operations for constructing solids usually do not allow one to obtain the required geometrical model through a single operation. This may require not only basic solid modeling operations, but also Boolean operations of combining solids (discussed in Lesson 8, as well as specialized operations (for example, chamfering and filleting, Lesson 10).

2. Swept solid

A swept solid is a body constructed by moving a generator curve along a guide curve. An arbitrary curve can be specified as a guide curve. The generating curve is defined as a set of contours that should not intersect with each other. In the simplest case, the contour may be only one, but it is permissible to specify a certain set of contours, each of which is a certain closed curve. Such an opportunity may be necessary, for example, to construct a model of a body with shaped holes or a body consisting of several parts.

In C3D Modeler, the utility function ::EvolutionSolid (the action_solid.h header file) is used to build swept solids. This function stores the MbCurveEvolutionSolid construction object in the build log. The prototype of the build swept solid function is:

```
MbResultType EvolutionSolid(
                  const MbSweptData&          sweptData,
                  const MbCurve3D&            spine,
                  const EvolutionValues&      params,
                  const MbSNameMaker&         operNames,
                  const PArray<MbSNameMaker>& contoursNames,
                  const MbSNameMaker&         spineNames,
                  MbSolid*&                   result );
```

Input function parameters:

1) sweptData	– generating curve data;
2) spine	– guide curve data;
3) params	– operation parameters;
4) operNames	– operation name maker;
5) contoursNames	– name maker of the segments of the generating contour;
6) spineNames	– name maker of the generatrix

Output data:

1) Return value is a rt_Success in case of successful construction or the result code of the MbResultType operation, explaining the error that occurred.
2) result – built solid.

The sweptData generating curve is described using the MbSweptData class. This class has several constructors that allow you to define the contours in the composition of the generator in various ways - on a plane, in three-dimensional space, on a given surface.

Parameters of the operation (params) are represented as an object of the EvolutionValues class. This class is inherited from SweptValues and due to such inheritance it receives the attributes thickness1, thickness2 and shellClosed. The attributes thickness1 and thickness2 allow you to specify the wall thickness when building a thin-walled body sweeping in two directions - inward and outward from the generator curve. The shellClosed attribute is a sign of the closure of the resulting body. The EvolutionValues class also contains its own attributes. The EvolutionValues::parallel attribute sets the motion properties of the generator curve along the guide. Three options are possible:

- parallel = 0 – the generating curve moves by parallel transference;
- parallel = 1 – the generating curve moves with the original angle relative to the guide;
- parallel = 2 – the generating curve moves with perpendicularity of the guide.

Example 2.1 demonstrates the construction of a swept solid using various variants of the movement of the generatrix along the guide. The guide is defined as a 4-th order three-dimensional NURBS spline, and the generator is defined as a square with rounded corners, as in Example 4.1 from Lesson 6.

Example 2.1. Constructing a swept solid by moving one closed loop along a NURBS curve (Figures 1-3).

```
#include <vector>
#include "alg_curve_fillet.h"
#include "cur_nurbs3d.h"
#include "action_solid.h"

using namespace std;
using namespace c3d;
using namespace TestVariables;


// Auxiliary function for constructing the generator curve
void CreateSketch(RPArray<MbContour>& _arrContours)
{
  // The contents of this function should be copied from Example 4.1, “Construction of
Extrusion Solids,” from Lesson 6.
}


void MakeUserCommand0()
{
  // Local SC (default coincides with the global SC)
  MbPlacement3D pl;

  // Control point array for building NURBS spline
  vector<MbCartPoint3D> vecPnts = { { 25, 0, 0 }, { 25, 40, -50 }, { 25, 10, -100 },
                                  { 25, 80, -200 }, { 25, 30, -250 }, { 25, 0, -300 } };
  SArray<MbCartPoint3D> arrPnts( vecPnts );
  // Construction of the guide curve in the form of an open 4-th order NURBS
  // spline by control points
  MbNurbs3D* pSpline = MbNurbs3D::Create( 4, arrPnts, false );

  // Description of the generating curve in the form of a plane contour on the XY plane
  // of the global SC
  MbPlane* pPlaneXY = new MbPlane(MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),
                                MbCartPoint3D(0, 1, 0));
  // Construction of generating curve using the CreateSketch auxiliary function
  RPArray<MbContour> arrContours;
  CreateSketch(arrContours);
  MbSweptData sweptData( *pPlaneXY, arrContours  );

  // An object with parameters for the operation of body sweeping
  EvolutionValues params;
  // Option of plane-parallel movement of the generatrix along the guide
  params.parallel = 0;

  // Service name objects for calling a geometric operation
  MbSNameMaker operNames(ct_CurveEvolutionSolid, MbSNameMaker::i_SideNone, 0);
  MbSNameMaker cNames(ct_CurveSweptSolid, MbSNameMaker::i_SideNone, 0);
  PArray<MbSNameMaker> contourNames(1, 0, false);
  contourNames.Add(&cNames);
  MbSNameMaker splineNames(ct_CurveSweptSolid, MbSNameMaker::i_SideNone, 0);

  // Calling an operation of construction of the swept solid
  MbSolid* pSolid = NULL;
  MbResultType res = ::EvolutionSolid( sweptData, *pSpline, params, operNames,
                                       contourNames, splineNames, pSolid );

  // Display of the constructed body
  if (res == rt_Success)
    viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

  // Displays a guide curve with an offset along the Y axis (in order to display, so
  // that the displayed curve is offset from the body surface).
  pSpline->Move(MbVector3D(MbCartPoint3D(0, 0, 0), MbCartPoint3D(0, -50, 0)));
  viewManager->AddObject(Style(3, RGB(0, 0, 255)), pSpline, &pl);

  // Reducing kernel object reference counts
  ::DeleteItem(pSolid);
  ::DeleteItem(pPlaneXY);
  ::DeleteItem(pSpline);
}
```

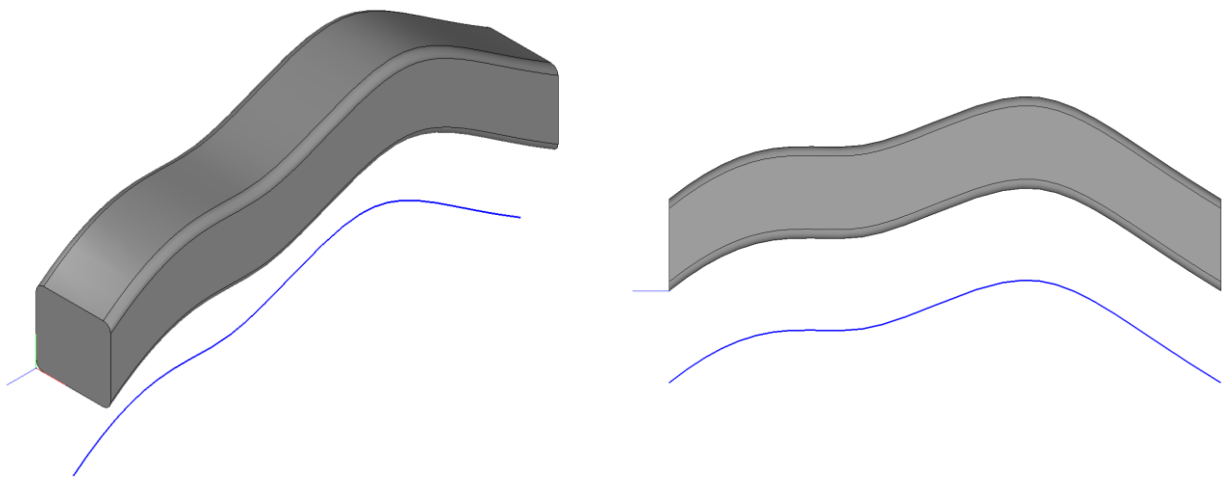

Fig. 1. An example of constructing a body of sweeping using Example 2.1 when specifying a variant of plane-parallel movement of a parallel = 0 generator.

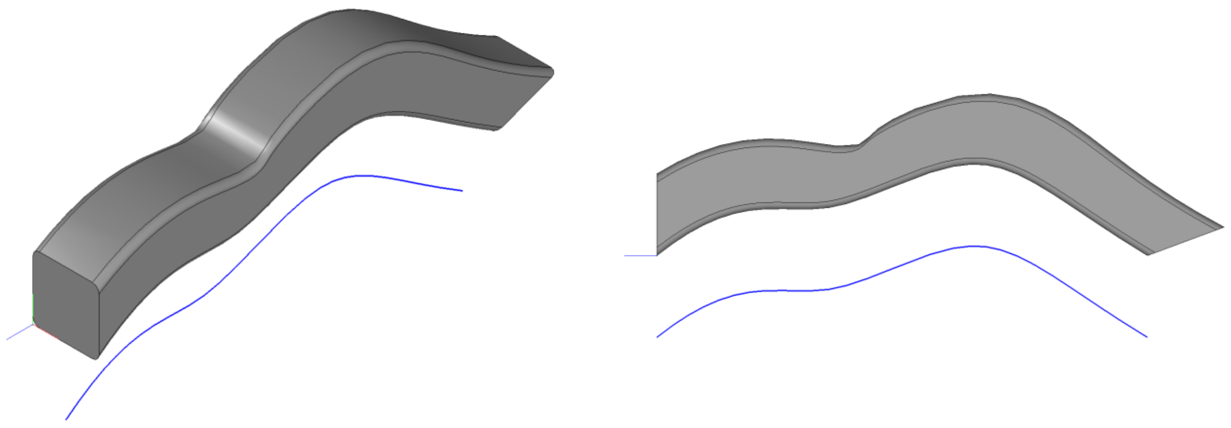

Fig. 2. An example of constructing a body of sweeping using Example 2.1 with the indication of the variant of the motion of the generator with the original angle parallel = 1.

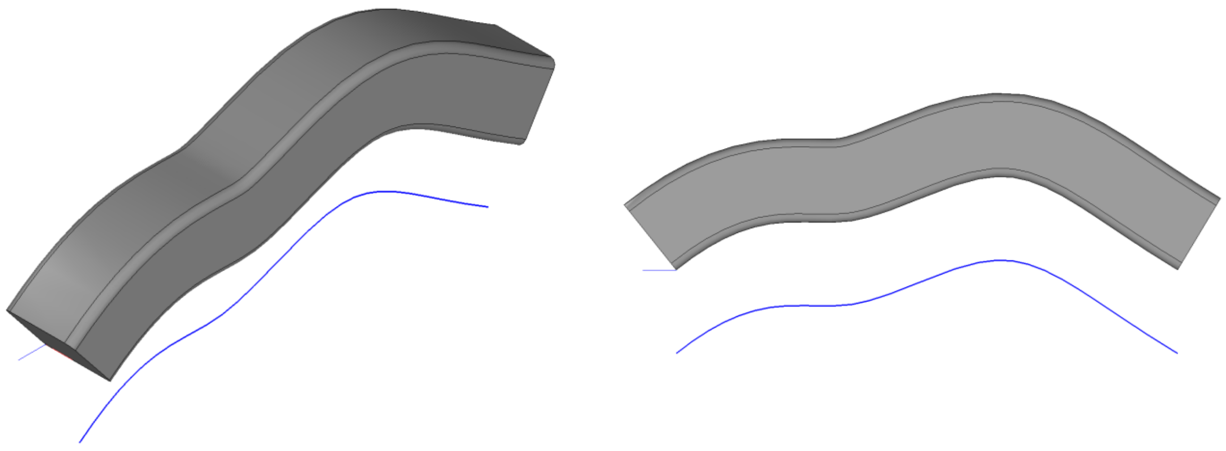

Fig. 3. An example of constructing the body of sweeping using Example 2.1 with the indication of the variant of the orthogonal motion of the generator parallel = 2.

In fig. 1-3, you can notice the effect of the EvolutionValues::parallel parameter on the shape of the resulting sweeping body. It is clearly seen how the value of this parameter is related to the initial and final position of the generator. However, it may not be easy to imagine the shape of the resulting sweeping body in advance. For example, in fig. 2 when parallel = 1, the generator maintains a constant angle relative to the guide, and at a distance of about a third of its overall size, a construction feature is observed - a narrowing of the cross section of the sweeping body. This is due to a change in the spatial position of the generator curve - in this case, it not only moves in parallel, but also rotates around the X axis of its local coordinate system. From this example, we can conclude that the shape of the resulting body can very significantly depend on the shape of the guide curve and the method of movement of the generatrix. In general, the more the slope of the guide curve changes, the more the cross section of the resultant sweeping body can change in those areas in which the generator rotates with greatly varying speed.

All bodies shown in fig. 1-3 are closed. To build a body with an open shell, you can specify the parameter shellClosed = false. The result is shown in Fig. 4. An unclosed body resembles a thin-walled with a wall of zero thickness - in a sense, it is the “lateral surface” of a sweeping body.

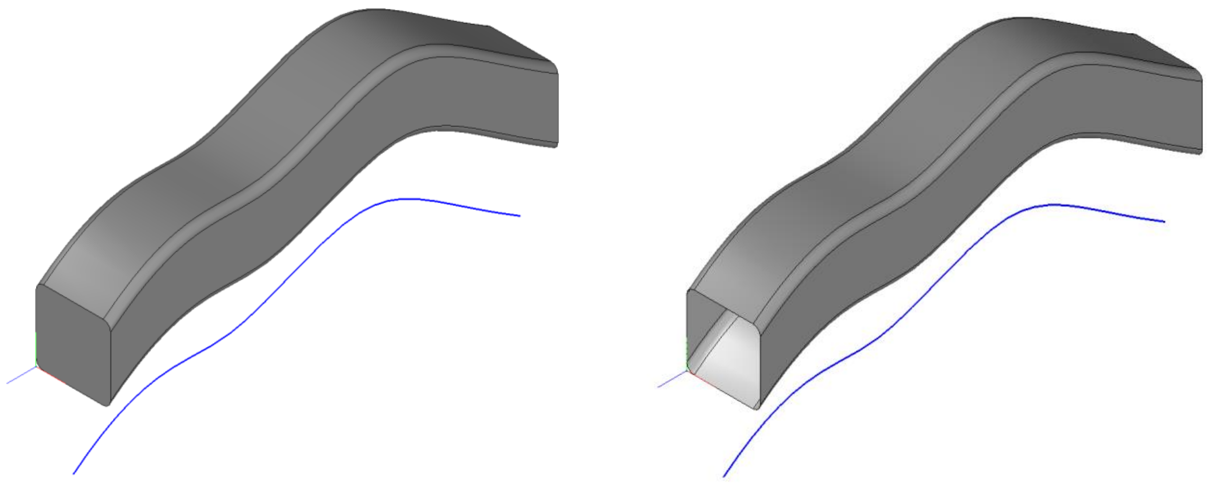

Fig. 4. The sweeping body during plane-parallel motion of the generatrix, parallel = 0. The closed sweeping body is shown on the left. This body is the same as pic. 1 and is built by default (params.shellClosed value = true). The right is the open body, obtained by explicitly specifying shellClosed = false.

In fig. 5 demonstrates the options for constructing a swept solid using a generatrix in the form of two concentric circles. The corresponding function for constructing the generator is shown in Example 2.2. The CreateSketch function from this example can be used in conjunction with Example 2.1. In fig. 5, it can be noted that when constructing a solid, the EvolutionSolid function classified one circuit as external, and the other as internal. When constructing a thin-walled solid, a resultant solid was obtained consisting of two closed parts.

Example 2.2. The function to build a generator in the form of two concentric circles (Fig. 5).

```
// Auxiliary function for constructing the generator curve (for example 2.1)
void CreateSketch(RPArray<MbContour>& _arrContours)
{
  // Building a generator in the form of two concentric circles

  const double RAD_EXT = 35.0;         // Outer circle radius
  const double RAD_INT = 15.0;         // Inner circle radius

  MbArc* pCircleExt = new MbArc( MbCartPoint(0,0), RAD_EXT );
  MbArc* pCircleInt = new MbArc( MbCartPoint(0,0), RAD_INT );

_arrContours.push_back( new MbContour(*pCircleExt, true) );
```

```
  _arrContours.push_back( new MbContour(*pCircleInt, true) );
}
```

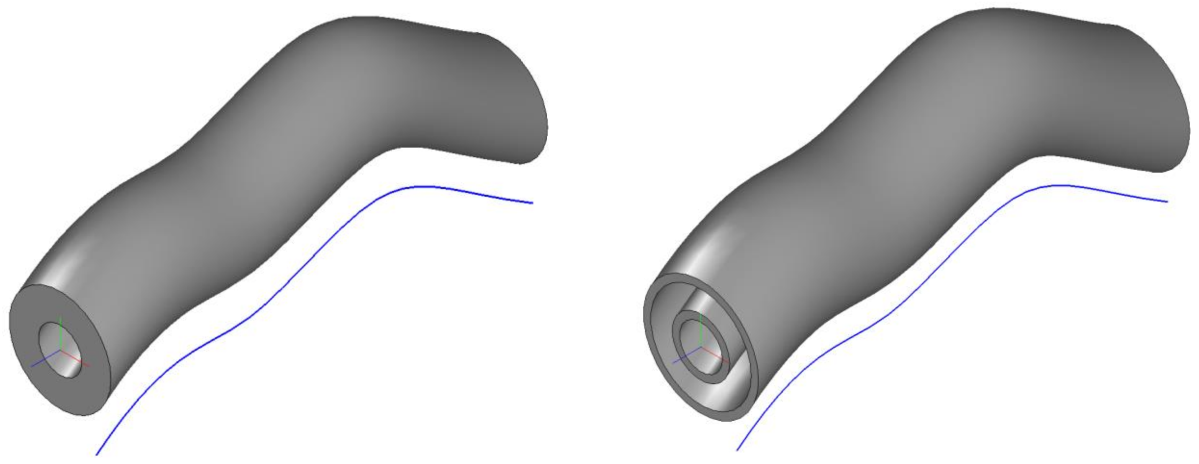

Fig. 5. The swept solid with plane-parallel motion (parallel = 0) forms in the form of two concentric circles (Example 2.2). (Left) Solid (construction by default). (Right) The thin-walled solid (thickness1 = 5, thickness2 = 0) consists of two parts.

The swept solid from several parts can be obtained not only as a result of the construction of thin-walled solids along the generators from nested contours, but also by explicitly specifying the generator in the form of several non-intersecting contours. Example 2.3 shows the function of constructing a generator in the form of four non-intersecting circles with centers located at the vertices of a square. The corresponding swept solid is shown in fig. 6.

Example 2.3. The function to build a generator in the form of four non-intersecting circles (Fig. 6).

```
// Auxiliary function for constructing the generator curve (for example 2.1)
void CreateSketch(RPArray<MbContour>& _arrContours)
{
  // Construction of a generator in the form of four non-intersecting circles of radius
  // RAD, the centers of which lie at the vertices of a square with side SIDE.
  const double RAD  = 30.0;
  const double SIDE = 100.0;

  MbArc* pCircle1 = new MbArc( MbCartPoint(0, 0), RAD );
  MbArc* pCircle2 = new MbArc( MbCartPoint(SIDE, 0), RAD );
  MbArc* pCircle3 = new MbArc( MbCartPoint(SIDE, SIDE), RAD );
  MbArc* pCircle4 = new MbArc( MbCartPoint(0, SIDE), RAD );

  _arrContours.push_back( new MbContour(*pCircle1, true) );
  _arrContours.push_back( new MbContour(*pCircle2, true) );
  _arrContours.push_back( new MbContour(*pCircle3, true) );
  _arrContours.push_back( new MbContour(*pCircle4, true) );
}
```

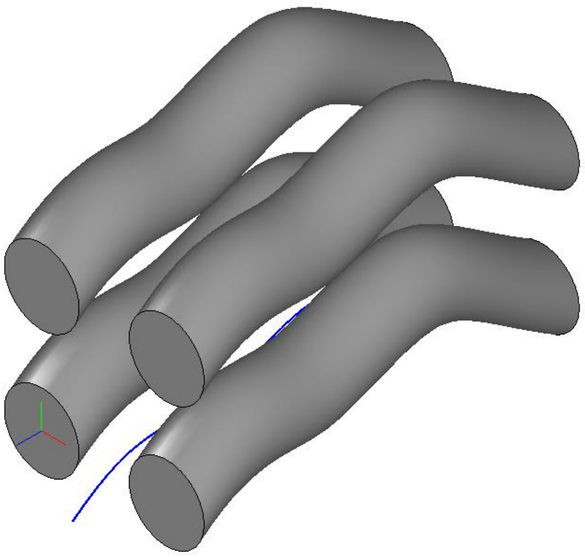

Fig. 6. Four-part swept solid with the generatrix from Example 2.3 (the generatrix moves in parallel, parallel = 0).

2.1 Tasks

1) Build a swept solid with a guide curve in the form of a fifth-order NURBS spline passing through the following control five control points:

   (0,0,0); (0,-2,-16); (0,-15,-20); (0,-22.5,-30); (0,-15,-40).

   Use a circle with a radius of 3 as a generator. Construct several variants of the swept solid with different variants of the generator movement (for different values of the parallel parameter) and with different values of the thickness1 and thickness2 parameters. In fig. 7 shows examples of the resulting bodies of remarks with the values of the parameter parallel = 0, 1, 2.

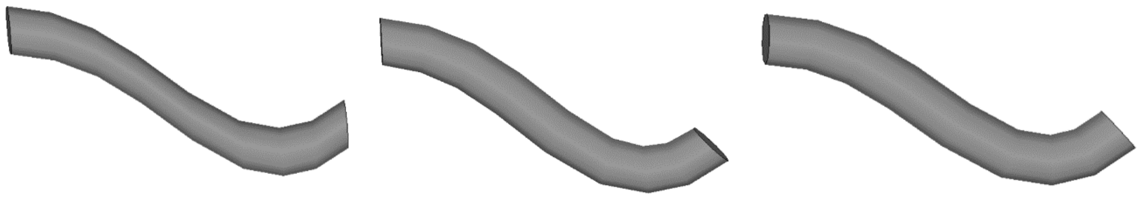

Fig. 7. Swept solids for the task 2.1.1.

Considering Example 2.2 (Fig. 5), it was shown that the EvolutionSolid function, like the extrusion body building and rotation body building functions, independently divides the composite contour into internal and external segments in the case when the contour of the generator contains a set of disjoint curves. Build up the swept solids shown in fig. 8. The constituents of these solids are a combination of generators from Example 2.1 and several circles.

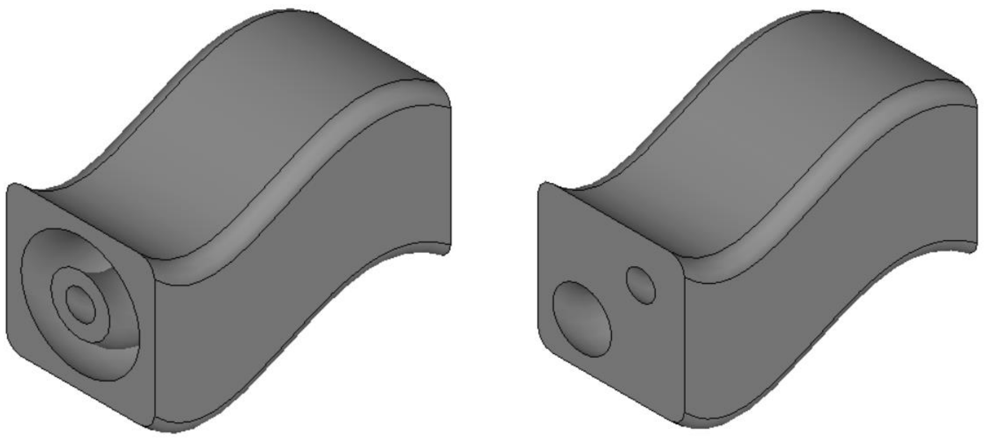

Fig. 8. Swept solids for the task 2.1.2.

Use a 4-rth order NURBS spline passing through the following control points as a guide curve:
(25,0,0); (25,30,40); (25,-30,70); (25,0,100).
For the plane-parallel motion of the generator, accept the value of the parameter parallel = 0.

2) Construct a wept solid, resembling a round spring, with a spiral guide with parallel = 1 (Fig. 9). To present the guide, use the MbConeSpiral class. The axis of the spiral is directed (by default) along the Z axis of the global SC. Forming in the form of a circle lies in the XZ plane of the world SC, and its center coincides with the starting point of the spiral curve. The starting point of the spiral curve can be calculated using the MbConeSpiral::PointOn method with the value of the parameter t = 0. The position of the generatrix is shown in an enlarged view on the right-hand side of Fig. 9.

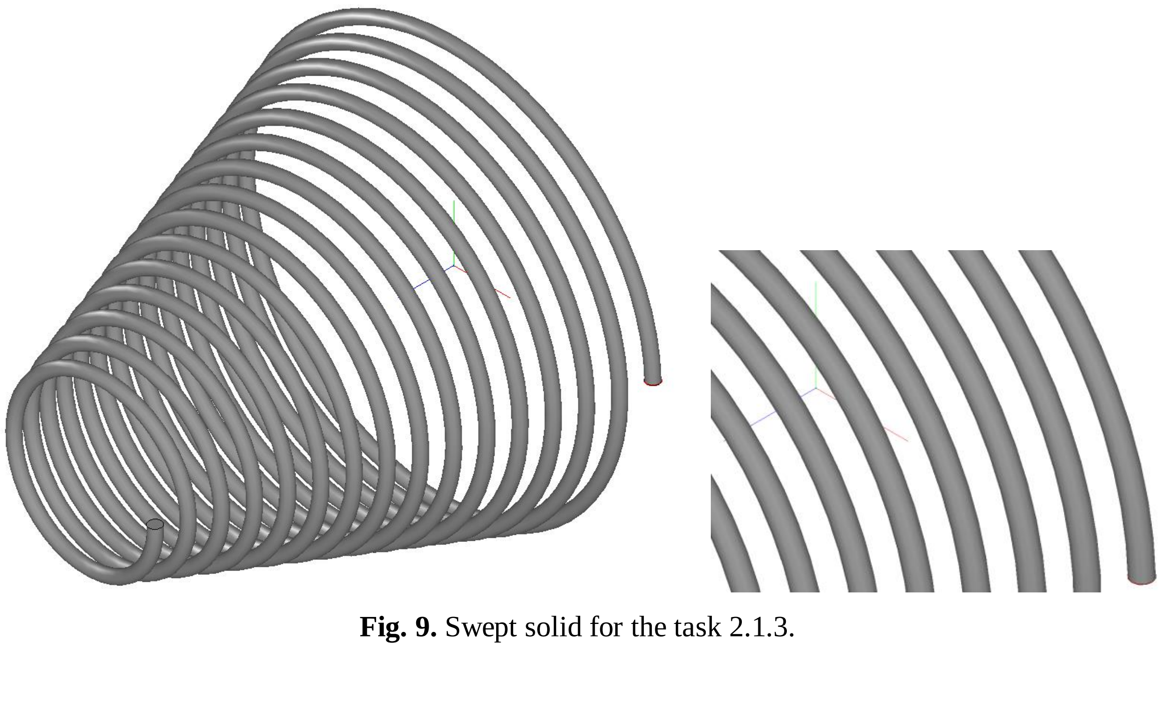

Fig. 9. Swept solid for the task 2.1.3.

3. Construction of solid by plane sections

The operation of constructing solids on a set of sections allows you to build bodies of arbitrary shape. By increasing the number of sections, it is possible to increase the accuracy of the approximate representation of the required geometric model. When performing this operation, a surface is calculated that passes through all specified cross sections of the body. In C3D Modeler, the operation of construction of solid by plane sections is performed using the LoftedSolid utility function (like all previously considered solid construction operations, it is described in the action_solid.h header file). When performing this operation, the builder of the MbCurveLoftedSolid class is placed in the solid build log. The prototype of the LoftedSolid function has a structure resembling the operations of constructing solids considered earlier. In this function, as a parameter, you must pass a description of the generator, the parameters of the operation and the objects named by it. Forming for this operation is an array of cross sections of the body. As a result, the numeric code and the constructed solid body are returned.

```
MbResultType LoftedSolid(
                  RPArray<MbSurface>&          surfs,
                  RPArray<MbContour>&          contours,
                  const MbCurve3D*             spine,
                  const LoftedValues&          params,
                  RPArray<MbCurve3D>*          guideCurves,
                  SArray<MbCartPoint3D>*       points,
                  const MbSNameMaker&          names,
                  RPArray<MbSNameMaker>&       namesContours,
                  MbSolid*&                    result );
```

Required input parameters of the function:

1) surfs – array of surfaces defining cross-sections (contours);
2) contours – an array of cross sections — contours of a generatrix curve (each contour in this array must correspond to a surface in the array surfs);
3) params – operation parameters;
4) names – face name maker;
5) namesContours – name maker of the contours of the generatrix.

Optional input parameters that provide additional ways to control the shape of the resulting body:

1) spine – guide curve;
2) guideCurves – a set of guiding curves defining the trajectories of points (points) of contours (contours);
3) points – an array of control points that allows you to control the joining of curves that pass through a multitude of contours. Each point in this array corresponds to a contour with the same index in the arrays of contours and surfaces.

Output data:

1) Return value is a rt_Success in case of successful construction or the result code of the MbResultType operation, explaining the error that occurred.
2) result – built solid body.

Example 3.1 demonstrates the use of the LoftedSolid function to build a body along four planar sections (Fig. 10). Circles are used as sections, and the resultant body has the form of a body of revolution. When you change the position of the center of one or several circles-sections (i.e. the elements of the arrCenters array in the auxiliary function CreateSketch), the shape of the body will change.

Example 3.1. Solid construction by sections (Fig. 10).

```
#include "action_solid.h"

using namespace c3d;
using namespace TestVariables;
```

```
// Auxiliary function to build a generator in the form of four circles.
// As a result, an array of surfaces is returned - planes (on which the circles lie)
// and an array of circles.
void CreateSketch( RPArray<MbSurface>& _arrPlanes, RPArray<MbContour>& _arrCircles )
{
  // Radii of four circles
  const int CIRCLES_CNT = 4;
  const double RAD[] = { 20, 50, 20, 40 };

  // The centers of the circles in the local CS of the corresponding planes.
  // Although the coordinates of all the centers are the same, they are represented as
  // an array so that they can be easily changed.
  const MbCartPoint arrCenters[CIRCLES_CNT] = {
    MbCartPoint(0, 0), MbCartPoint(0, 0), MbCartPoint(0, 0), MbCartPoint(0, 0)
  };

  // 2D curves - circles centered on centerArc
  MbArc* pArc2D[CIRCLES_CNT] = { NULL };
  for (int i = 0; i < CIRCLES_CNT; i++ )
    pArc2D[i] = new MbArc(arrCenters[i], RAD[i]);

  // Coordinate systems defining the position of the planes of circles in space.
  // The 0th circle is located in the XZ plane of the global SC, the planes for the
  // remaining circles are located above the XZ plane.
  MbPlacement3D plArc[CIRCLES_CNT] = {
       MbPlacement3D( MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),
                      MbCartPoint3D(0, 0, 1) ),
       MbPlacement3D( MbCartPoint3D(0, 60, 0), MbCartPoint3D(1, 60, 0),
                      MbCartPoint3D(0, 60, 1) ),
       MbPlacement3D( MbCartPoint3D(0, 100, 0), MbCartPoint3D(1, 100, 0),
                      MbCartPoint3D(0, 100, 1) ),
       MbPlacement3D( MbCartPoint3D(0, 130, 0), MbCartPoint3D(1, 130, 0),
                      MbCartPoint3D(0, 130, 1) )
  };

  // Surfaces - planes containing circles.
  // The position of the planes is given by local coordinate systems plArc.
  MbSurface* pPlanes[CIRCLES_CNT] = { NULL };
  for (int i = 0; i < CIRCLES_CNT; i++ )
    pPlanes[i] = new MbPlane(plArc[i]);

  // Saving planes and circles (as contours) in returned arrays.
  for (int i = 0; i < CIRCLES_CNT; i++ )
  {
    _arrCircles.push_back( new MbContour( *pArc2D[i], true ) );
    _arrPlanes.push_back( pPlanes[i] );
  }

  // Reducing kernel object reference counts
  for (int i = 0; i < CIRCLES_CNT; i++ )
  ::DeleteItem( pArc2D[i] );
}


void MakeUserCommand0()
{
  // Getting generators in the form of arrays of surfaces and contours on them using an
  // auxiliary function.
  RPArray<MbSurface> arrSurfaces;
  RPArray<MbContour> arrContours;
  CreateSketch( arrSurfaces, arrContours );
  // Object with sweep operation parameters.
  LoftedValues params;

  // Objects for naming solid model elements.
  MbSNameMaker names(ct_CurveLoftedSolid, MbSNameMaker::i_SideNone, 0);
  PArray<MbSNameMaker> contourNames(0, 1, false);

  // Build a swept solid
  MbSolid* pSolid = NULL;
  MbResultType res = ::LoftedSolid( arrSurfaces, arrContours, NULL, params,
                                    NULL, NULL, names, contourNames, pSolid );

  // Display of the constructed body
  if (res == rt_Success)
    viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

  // Reduced body reference count
  ::DeleteItem(pSolid);
}
```

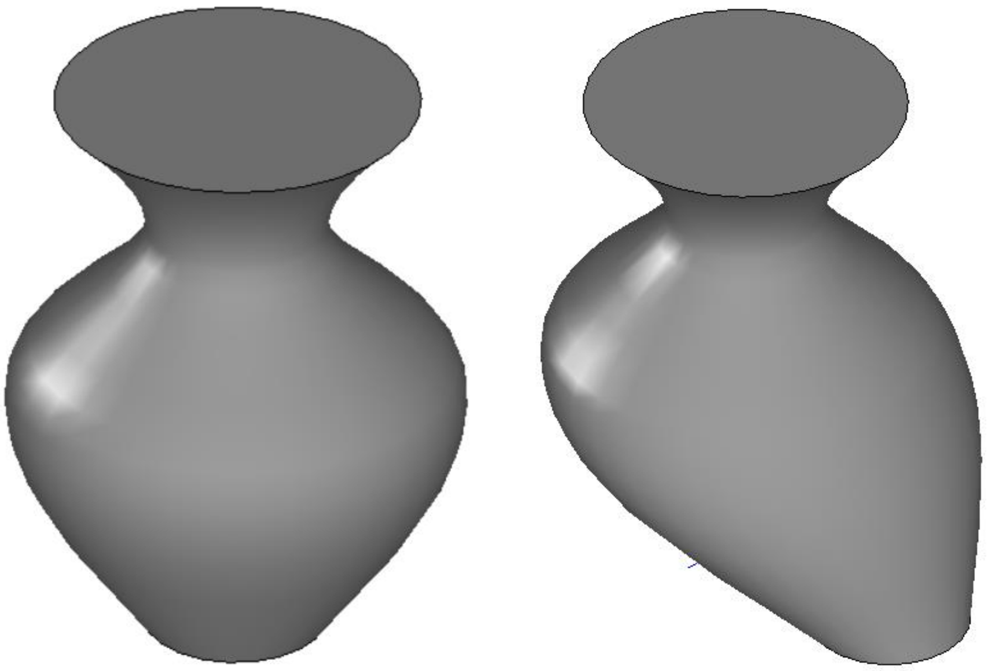

Fig. 10. Bodies constructed from sections using Example 3.1. (Left) The body of rotation obtained for the case when the centers of all sections lie on the same axis (they have the same coordinates (0, 0) in their local CS). (Right) The body obtained by displacing the center of the 0th circle-section to the point (50, 0) on the corresponding section plane.

In case the section contours contain an unequal number of segments, the LoftedSolid function automatically splits the contours to obtain an equal number of segments in them. Example 3.2 shows the CreateSketch generator function, which can be used in conjunction with Example 3.1 (as a replacement for the corresponding function). In Example 3.2, the circle, square, and triangle are specified as sections for the body. The resulting body is shown in fig. 10. It is possible to control the position of the edges connecting the vertices of different contours-sections with the help of an array of control points.

Example 3.2. Body construction with a different number of segments in plane contours

```
// Auxiliary function for constructing a generator in the form of a set of three
// sections - a circle, a square and a triangle.
void CreateSketch(RPArray<MbSurface>& _arrSurfaces, RPArray<MbContour>& _arrContours)
{
  // The first section - the circle - lies in the XZ plane of the global SC.
  const MbCartPoint circleCenter(0, 0);
  const double CIRCLE_RAD = 50;
  MbArc* pArc2D = new MbArc(circleCenter, CIRCLE_RAD);
  MbPlacement3D plCircle( MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),
                          MbCartPoint3D(0, 0, 1) );
  MbSurface* pSurfCircle = new MbPlane(plCircle);
  MbContour* pContourArc = new MbContour(*pArc2D, true);

  // The second section - a square - lies in a plane parallel to XZ, offset by 50 units
  // along the Y axis of the global SC.
  const double SQUARE_SIDE  = 50;
  const double SQUARE_OFS_Y = 50;
  SArray<MbCartPoint> arrSqVerts(4);
  arrSqVerts.Add( MbCartPoint(-SQUARE_SIDE/2, -SQUARE_SIDE/2) );
  arrSqVerts.Add( MbCartPoint(-SQUARE_SIDE/2,  SQUARE_SIDE/2) );
  arrSqVerts.Add( MbCartPoint( SQUARE_SIDE/2,  SQUARE_SIDE/2) );
  arrSqVerts.Add( MbCartPoint( SQUARE_SIDE/2, -SQUARE_SIDE/2) );
  MbPolyline* pSquarePoly = new MbPolyline(arrSqVerts, true);
  MbPlacement3D plSquare( MbCartPoint3D(0, SQUARE_OFS_Y, 0),
                          MbCartPoint3D(1, SQUARE_OFS_Y, 0),
                          MbCartPoint3D(0, SQUARE_OFS_Y, 1) );
  MbSurface* pSurfSquare    = new MbPlane(plSquare);
  MbContour* pContourSquare = new MbContour(*pSquarePoly, true);

  // The third section - a triangle - lies in a plane parallel to XZ, shifted by 100
  // units along the Y axis of the global SC.
  const double TRIANGLE_SIDE  = 70;
  const double TRIANGLE_OFS_Y = 100;
  SArray<MbCartPoint> arrTrVerts(3);
  arrTrVerts.Add( MbCartPoint( -TRIANGLE_SIDE/2, -20) );
  arrTrVerts.Add( MbCartPoint( 0, 60));
  arrTrVerts.Add( MbCartPoint( TRIANGLE_SIDE/2, -20) );
  MbPolyline* pTrianglePoly = new MbPolyline(arrTrVerts, true);
  MbPlacement3D plTriangle( MbCartPoint3D(0, TRIANGLE_OFS_Y, 0),
                            MbCartPoint3D(1, TRIANGLE_OFS_Y, 0),
                            MbCartPoint3D(0, TRIANGLE_OFS_Y, 1) );
  MbSurface* pSurfTriangle    = new MbPlane(plTriangle);
  MbContour* pContourTriangle = new MbContour(*pTrianglePoly, true);

  // Saving returned kernel objects – contours-sections and surface-planes
  // containing them.
  _arrSurfaces.push_back(pSurfCircle);
  _arrSurfaces.push_back(pSurfSquare);
  _arrSurfaces.push_back(pSurfTriangle);
  _arrContours.push_back(pContourArc);
  _arrContours.push_back(pContourSquare);
  _arrContours.push_back(pContourTriangle);

  // Decrease of two-dimensional curve counters used to create contours.
  ::DeleteItem(pArc2D);
  ::DeleteItem(pSquarePoly);
  ::DeleteItem(pTrianglePoly);
}
```

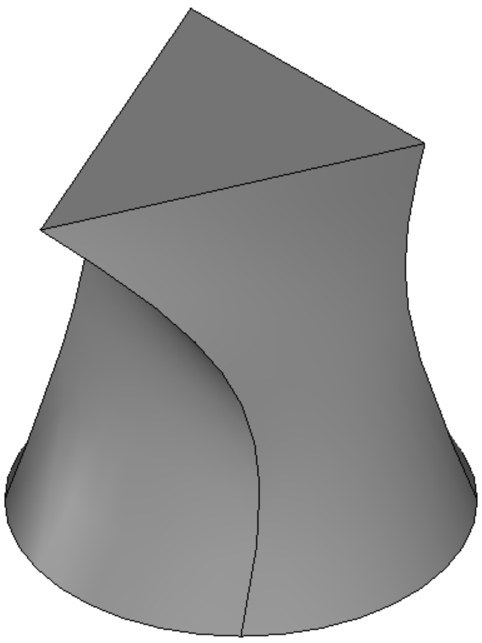

Fig. 11. Body, built on three sections of different shapes (Example 3.2).

Among the optional parameters of the LoftedSolid function are the spine and guideCurves parameters. The spine parameter allows you to explicitly specify a guide curve to build all the edges passing through the corresponding points of the contours-sections. The guideCurves parameter allows you to specify individual guide curves as an array, defining the shape of each edge. As guides can use curves of arbitrary shape. Example 3.3 shows the construction of a body in two sections. A circle is used as the initial section, and a square is used as the final section. Other intermediate sections are not specified. In fig. Figure 12 shows the resultant body for the two methods of construction — by default (with automatic edge formation) and when using the explicitly transmitted NURBS curve as a guide. When building a “default” without specifying a guideline, a straight line is used in its quality, since the origin of the local coordinate systems of the contours in this example coincide with the middle of the circle and the intersection point of the diagonals of the square. The position of the edges in this example can be changed by changing the order of defining the vertices of the second section - the square.

Example 3.3. Body construction with a guide curve

```
#include <vector>
#include "cur_nurbs3d.h"
#include "action_solid.h"

using namespace std;
using namespace c3d;
using namespace TestVariables;


// Auxiliary function to build a generator in the form of a set of two sections
// - a circle and a square.
void CreateSketch(RPArray<MbSurface>& _arrSurfaces, RPArray<MbContour>& _arrContours)
{
  // The first section is a circle.
  const MbCartPoint circleCenter(0, 0);
  const double RAD = 50;
  MbArc* pArc2D = new MbArc(circleCenter, RAD);
  MbContour* pContourCircle = new MbContour(*pArc2D, true);
  // The second section is a square
  const double SQUARE_SIDE = 50;
  SArray<MbCartPoint> arrVertsSq(4);
  arrVertsSq.Add(MbCartPoint(-SQUARE_SIDE/2, -SQUARE_SIDE/2));
  arrVertsSq.Add(MbCartPoint(-SQUARE_SIDE/2,  SQUARE_SIDE/2));
  arrVertsSq.Add(MbCartPoint( SQUARE_SIDE/2,  SQUARE_SIDE/2));
  arrVertsSq.Add(MbCartPoint( SQUARE_SIDE/2, -SQUARE_SIDE/2));
  MbPolyline* pSquarePoly   = new MbPolyline(arrVertsSq, true);
  MbContour* pContourSquare = new MbContour(*pSquarePoly, true);

  // Construction of surface-planes containing sections.
  MbPlacement3D plCircle( MbCartPoint3D(0, 0, 0), MbCartPoint3D(1, 0, 0),
                          MbCartPoint3D(0, 0, 1) );
  MbPlacement3D plSquare( MbCartPoint3D(500, 500, 0), MbCartPoint3D(501, 500, 0),
                          MbCartPoint3D(500, 500, 1) );
  MbSurface* pSurfCircle = new MbPlane(plCircle);
  MbSurface* pSurfSquare = new MbPlane(plSquare);

  // Saving returned objects in output arrays.
  _arrSurfaces.push_back(pSurfCircle);
  _arrSurfaces.push_back(pSurfSquare);
  _arrContours.push_back(pContourCircle);
  _arrContours.push_back(pContourSquare);

  // Reduction of reference counters of objects-plane curves used
  // for construction of contours.
  ::DeleteItem(pArc2D);
  ::DeleteItem(pSquarePoly);
}


void MakeUserCommand0()
{
  // SC to display the guide line, coincides with the global SC.
  MbPlacement3D pl;

  // Getting generators in the form of arrays of surfaces and contours
  // on them using an auxiliary function.
  RPArray<MbSurface> arrSurfaces;
  RPArray<MbContour> arrContours;
  CreateSketch(arrSurfaces, arrContours);

  // Object with sweep operation parameters.
  LoftedValues params;

  // Objects for naming solid model elements.
  MbSNameMaker names(ct_CurveLoftedSolid, MbSNameMaker::i_SideNone, 0);
  PArray<MbSNameMaker> contourNames(0, 1, false);

  // An array of control points for constructing a spline guide curve
  vector<MbCartPoint3D> vecPnts = { { 0, 0, 0 }, { 10, 155, 0 }, { 300, 250, 0 },
                                    { 450, 350, 0 }, { 500, 500, 0 } };
  SArray<MbCartPoint3D> arrPnts( vecPnts );

  // Building a guide in the form of an open 4-th order NURBS spline
  ptrdiff_t degree = 4;
  MbNurbs3D* pSpine = MbNurbs3D::Create(degree, arrPnts, false);
  // Guide display (with shift for easy perception).
  if ( pSpine )
  {
    pSpine->Move( MbVector3D(MbCartPoint3D(0, 0, 0),MbCartPoint3D(0, 0, 250)));
    viewManager->AddObject(Style(1, RGB(255, 0, 0)), pSpine, &pl);
  }

  // Construction of a solid body by cross sections
  MbSolid* pSolid = NULL;
  MbResultType res = ::LoftedSolid( arrSurfaces, arrContours, pSpine, params,
                                    NULL, NULL, names, contourNames, pSolid );

  // Display of the constructed body
  if (res == rt_Success)
    viewManager->AddObject(Style(1, LIGHTGRAY), pSolid);

  // Reducing kernel object reference counts
  ::DeleteItem(pSolid);
  ::DeleteItem(pSpine);
}
```

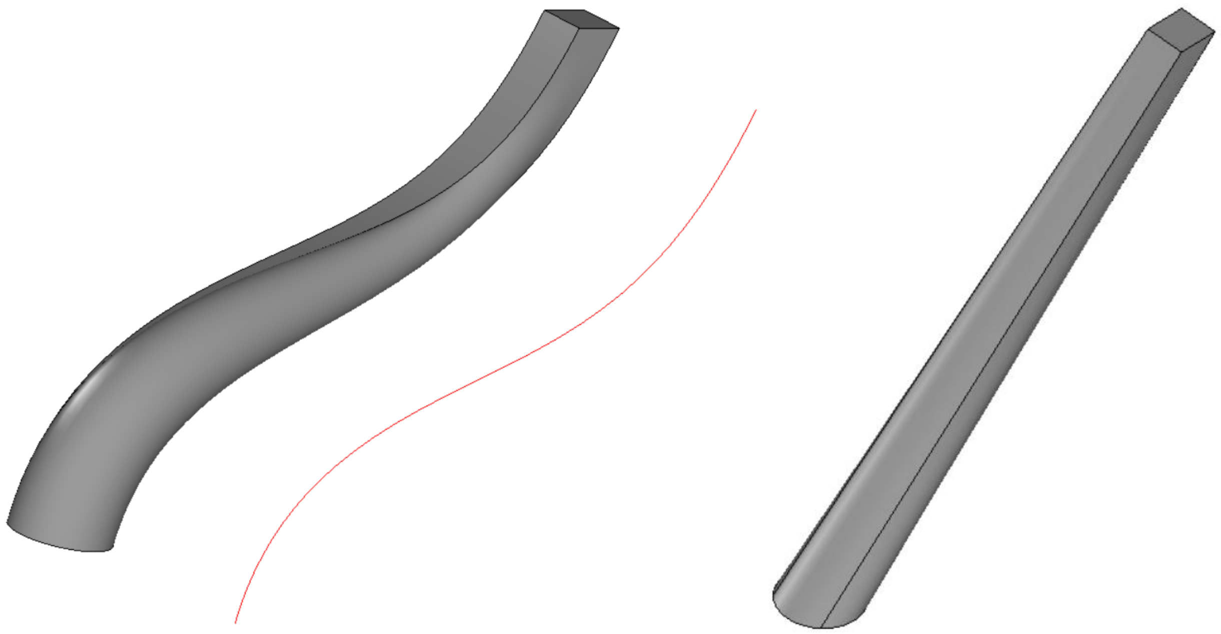

Fig. 12. The construction of the body in two sections in the form of a circle and a square (Example 3.3.). (Left) Body obtained by explicitly specifying a guide in the form of a NURBS curve. (Right) Body built by default with automatic edge generation.

3.1 Tasks

1) Build a body that represents a transition from a section — a triangle into a section — a rectangle (Fig. 13).

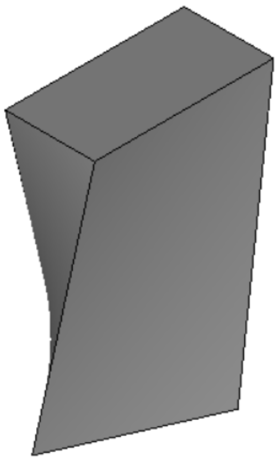

Fig. 13. Possible body variant for task 3.1.1.

2) The contours of sections for the construction of the body need not be closed. Build a body in two sections, having an arbitrary guide curve (use a NURBS – curve constructed from a set of control points). Use a circle as the initial section of the body, and a non-closed square as the final section (in Example 3.3, when building the MbPolyline polyline, you can specify the input parameter closed = false to get an open curve).
3) Bodies constructed by sections can be modified like bodies constructed using other methods by changing the parameters of construction. The parameter params is an object of the LoftedValues class and contains information on the presence and thickness of the body walls and on the closedness of the constructed body.
   Build a body by four sections, the guide curve of which is an arc of a circle with an angular size of 270°. The initial section is a circle, the final one is a square with rounded corners (as in Example 2.1), the intermediate sections are a square and a triangle. Place the two-dimensional contours-section on the guide curve with an angular step of 90° (Fig. 14).
   Set different values for the params.thickness1 parameters (offset outwards from the curve generator) and params.thickness2 (indents inside the generator curve). To avoid self-intersection of the faces of the body, set the defining dimensions of the sections much larger than the wall thickness.
   Build an open body, specifying the value of the parameter params.shellClosed = false.

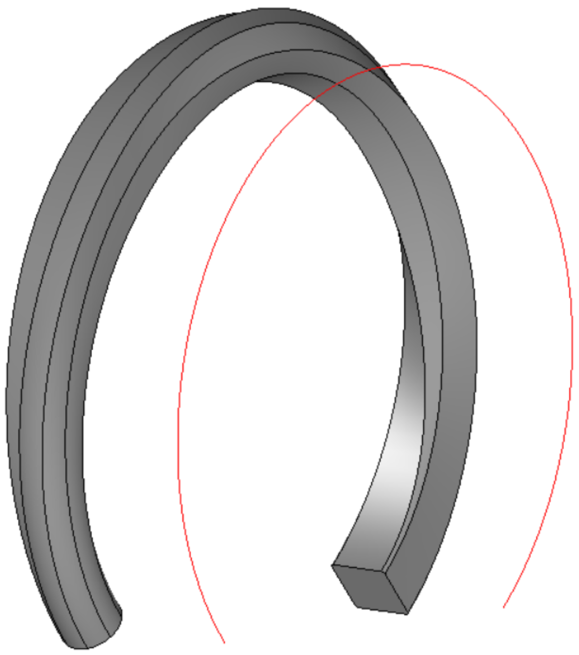

Fig. 14. Possible body option for task 3.1.3.

4. Conclusion

In this lesson, we considered the functions of C3D Modeler for constructing swept bodies (kinematic bodies) and for building bodies over sections. Together with the construction of the bodies of extrusion and rotation (Lesson 6), they form a set of four basic operations of solid modeling - in C3D these are the functions EvolutionSolid, LoftedSolid, ExtrusionSolid and RevolutionSolid.

The interfaces of the EvolutionSolid and LoftedSolid functions have a similar structure and require the specification of a generator, a guide, operation parameters, and name objects. Among the four basic modeling operations listed above, the LoftedSolid function has the greatest number of possibilities to control the shape of a solid body obtained by specifying an arbitrary number of cross sections, as well as explicitly defining the set of guide curves that define the shape of body edges. However, in practice, to speed up the computational processing and simplify the program text, you should, if possible, choose simpler modeling operations, ExtrusionSolid and RevolutionSolid, and use universal EvolutionSolid and LoftedSolid, if necessary.