В.Ф. Барабанов А.М. Нужный
В.В. Сафронов Н.И. Гребенникова

ПАРАМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ NX API

Учебно-методическое пособие

Воронеж 2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «Воронежский государственный
технический университет»

В.Ф. Барабанов А.М. Нужный
В.В. Сафронов Н.И. Гребенникова

ПАРАМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ NX API

Утверждено учебно-методическим советом
университета в качестве учебно-методического пособия

Воронеж 2017

УДК 681.3.06(07)
ББК 32.973-018я7
Б 24

Параметрическое моделирование с использованием NX API: учебно-метод. пособие [Электронный ресурс]. - Электрон, текстовые и граф. данные (1,9 Мб) / В.Ф. Барабанов, А.М. Нужный, В.В. Сафронов, Н.И. Гребенникова. - Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2017. - 1 электрон. опт. диск (CD-ROM): цв. - Систем. требования: ПК 500 и выше; 256 Мб ОЗУ; Windows XP; SVGA с разрешением 1024x768; Adobe Acrobat; CD-ROM дисковод; мышь. - Загл. с экрана.

В учебно-методическом пособии рассматриваются основные приемы использования API NX Open для решения задач трехмерного моделирования, приводятся задания по темам лабораторных работ.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего образования по направлению подготовки бакалавров 09.03.01 «Информатика и вычислительная техника», профилям «Вычислительные машины, комплексы, системы и сети», «Системы автоматизированного проектирования», «Системы автоматизированного проектирования в машиностроении», дисциплине «Компьютерная графика».

Ил. 32. Библиогр.: 11 назв.

Рецензенты: кафедра вычислительной техники
и информационных систем
Воронежского государственного
лесотехнического университета
им. Г.Ф. Морозова (зав. кафедрой
д-р техн. наук, проф. В.К. Зольников);
д-р техн. наук, проф. А.М. Литвиненко

ВВЕДЕНИЕ

Современный рынок программного обеспечения предлагает множество универсальных систем автоматизированного проектирования, обладающих мощными средствами 3D-моделирования, и позволяющих значительно повысить эффективность работы конструкторов и технологов в самых различных сферах проектирования и производства.

Одной из наиболее востребованных систем подобного класса является Siemens NX, представляющий собой интерактивную систему автоматизации проектирования, расчета и изготовления изделий, предназначенную для использования в различных отраслях промышленности.

NX относится к так называемым системам высокого уровня автоматизированного проектирования и обладает широким набором инструментальных средств. NX широко распространена во всем мире и используется для разработки продукции ведущими мировыми производителями, такими как Daimler, Bosh, Boeing, NASA Jet Propulsion Laboratory, «ОКБ им. Сухого», «ГКНПЦ им. Хруничева», ПАО «КАМАЗ» и пр. Основная задача системы в конечном итоге состоит в сокращении стоимости создания изделия, улучшении его качества и сокращении сроков выхода на рынок.

Для обозначения систем этого класса используется аббревиатура CAD/CAM/CAE (Computer-Aided Design, Computer-Aided Manufacturing, Computer-aided Engineering), что дословно переводится как «Проектирование с Помощью Компьютера», «Изготовление с Помощью Компьютера», «Компьютерная поддержка инженерных расчетов». В русском языке для обозначения вышеперечисленных разновидностей программ используется термин САПР (системы автоматизированного проектирования).

Подсистема CAD предназначена для автоматизации проектных, конструкторских и чертежных работ и разработки проектно-конструкторской документации (моделирование деталей и сборок, создание чертежей, анализ, оптимизация

конструкции и т.д.). Система NX CAD позволяет выполнить моделирование деталей и сборок изделия, провести анализ пересечений и расчёт массы, подготовить 2D-документацию - чертежи или 3D-документацию с использованием PMI (размеры и аннотации наносятся на 3D-модель). С помощью инструментария приложений моделирования деталей и сборочных единиц, пользователь может создать полный цифровой аналог разрабатываемого узла или единичной детали, содержащий точную геометрию, рассчитанные массово-инерционные характеристики, свойства материалов, а также все требования необходимые для изготовления и контроля.

Возможности системы позволяют моделировать изделия любой степени сложности и размерности – от бытовой техники до изделий корабельной и авиакосмической промышленностей. Электронные модели, создаваемые в приложениях NX CAD используются далее в модулях инженерного анализа и технологической подготовки производства.

Подсистема CAM обеспечивает автоматизированную подготовку управляющих программ для оборудования с ЧПУ на основе математической модели детали, созданной в CAD-системе.

Модуль NX CAM поддерживает различные виды обработки: токарную обработку, фрезерную обработку на 3—5-осевых станках с ЧПУ, токарно-фрезерную, электроэрозионную проволочную обработку. Содержит встроенный модуль симуляции обработки на станке, работающий в кодах управляющей программы (G-кодах), который используется для анализа УП и обеспечивает контроль столкновений.

Функции инженерного анализа (CAE) обеспечивают анализ изделия, сборки и возможность симуляции процессов с помощью широкой гаммы инженерных приложений.

Набор средств инженерного анализа в системе NX представляет собой приложение пре- и постпроцессинга

(Pre/Post) и подключаемых к интерфейсу расчётных решателей. В качестве решателей может выступать как пакет NX Nastran, так и программные пакеты других разработчиков. Среда инженерного анализа может работать как независимо, так и в интеграции с PLM системой Teamcenter. В последнем случае все расчетные данные сохраняются в PLM системы и управляются с точки зрения прав доступа, ревизионности, процессов выпуска и согласования, и т.д.

Функции NX разделены по "приложениям" и возможностям. Все функциональные модули NX вызываются из управляющего модуля (ядро системы), который называется Базовый модуль NX - Gateway. Все остальные модули являются необязательными и могут быть подобраны согласно специфике работы пользователя.

NX - это полностью трехмерная система, которая позволяет идеально воспроизвести почти любую геометрическую форму, оперируя числами с удвоенной точностью. Как правило, работа начинается с построения геометрии, описывающей изделие или деталь конструкции. На основании созданной трехмерной геометрической модели изделия впоследствии могут быть получены:

- полностью образмеренные чертежи;
- команды для станков с ЧПУ производящих обработку и выполняющих производственные процессы;
- исходные данные для решения задач инженерного анализа, например, дискретные модели для метода конечных элементов.

При этом, вне зависимости от уровня САПР и ее инструментальной насыщенности, практически всегда существует ряд задач, автоматизация которых не представляется возможной с использованием стандартных средств системы. К их числу можно отнести автоматизацию рутинных функций, таких как построение массивов однотипных элементов, отличающихся базовыми размерами, анализ геометрии и корректировку значительных объемов

ранее созданных моделей с целью их унификации или устранения ошибок, создание модулей для размещения и анализа в чертеже неграфических данных, имеющих отношение к предметной области объекта проектирования.

Наиболее эффективным средством решения подобных задач является использование интерфейса прикладного программирования (API), представляющего собой библиотеки классов, структур, процедур и функций, позволяющие получить программный доступ к объектной модели системы и призванный упростить создание пользовательских программных модулей, интегрированных с базовым программным продуктом. Использование API позволяет осуществлять эффективную адаптацию системы проектирования к самым различным потребностям, значительно расширять ее функционал, что позволяет существенно сократить затраты на проектирование изделий.

В системе NX набор средств API получил название API NX Open.

Перечень основных инструментов NX Open приведен на рис. 1.

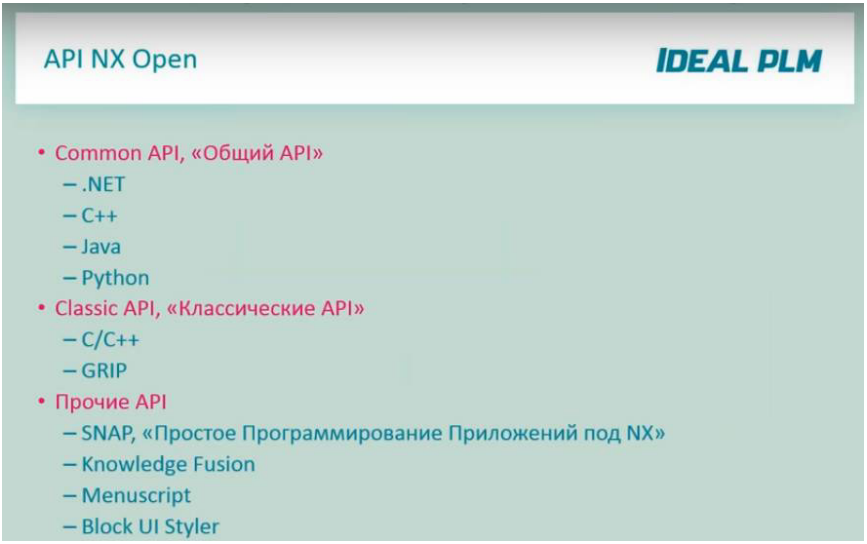


Рис. 1. Инструменты NX Open

Основным инструментарием в настоящее время является Common API, включающий поддержку языков .NET (VB и C#), C++, Java, и, начиная с NX 10 – Python.

Классический API поддерживает C++ с более старым набором функций и GRIP, унаследованный NX от UNIGRAPHICS.

Помимо этого имеется набор средств, не являющихся NX Open, но осуществляющих поддержку основных инструментов. К их числу относится SNAP – упрощенный API, представляющий собой современную версию GRIP, Knowledge Fusiuon – набор инструментов, позволяющих включить логику в разрабатываемую модель, Menuscript - средства адаптации интерфейса системы, Block UI Styler – инструмент для разработки приложений, использующих стандартный интерфейс NX.

До появления «Общего API» были разработаны три ранних API, которые в настоящее время поддерживаются, но не развиваются. Это Open C, Open C++ и NX Oen GRIP. В настоящее время средства Common API используют значительное количество функций (около 5000), реализованных в Open C.

Наиболее существенные характеристики Common API приведены на рис. 2.

Common API представляет собой современную унифицированную среду разработки, интегрирующую в себе все возможности по разработке путем использования общей объектной модели.

На рис. 3 представлен перечень языков программирования и средств разработки, поддерживаемых Common API.

Целью данного методического пособия является обучение студентов основным приемам использования API NX Open для решения задач трехмерного моделирования.

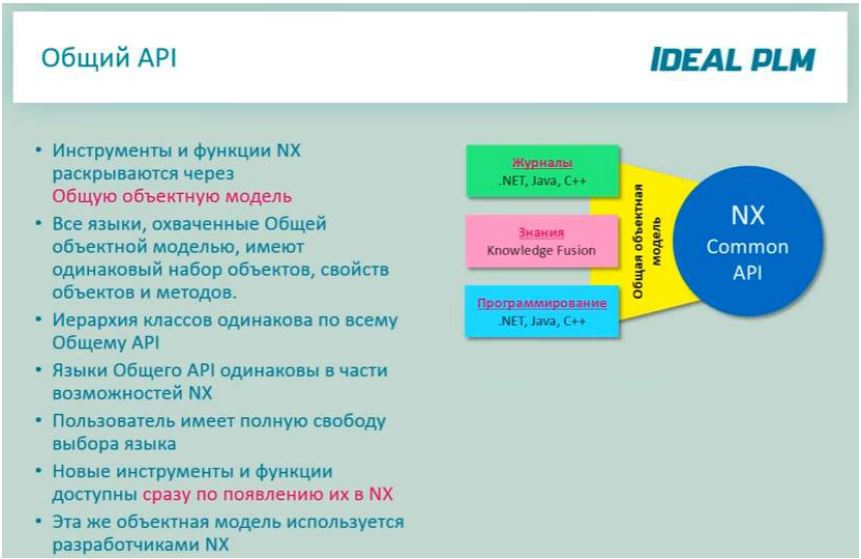


Рис. 2. Характеристики Common API

Рис. 3. Перечень поддерживаемых Common API языков программирования и средств разработки

1. ЛАБОРАТОРНАЯ РАБОТА № 1
СОЗДАНИЕ ПРОСТЫХ МОДЕЛЕЙ В NX

Цель работы: Научиться создавать простейшие библиотеки в Visual Studio для NX 7.5 и запускать их на выполнение в NX, научиться рисовать простейшие примитивы (линия, дуга), 2D-модели (прямоугольник, окружность), а также 3D-модели (цилиндр, параллелепипед, шар) на их основе с помощью операций «выдавливание» и «вращение».

1.1. Теоретическая часть

Для разработки прикладных пользовательских программ для NX 7.5 целесообразно использовать среду разработки Microsoft Visual Studio 2008, как наиболее эффективно поддерживающую шаблоны NX7 Open. Для NX 8.0 рекомендуется использовать Microsoft Visual Studio 2010 или более поздние версии программного продукта.

Для получения доступа к мастеру создания пользовательских программ для NX 7.5 в Visual Studio необходимо выполнить следующие действия:

1. Открыть папку “vs_files”, которая находится в папке установленного NX. По умолчанию путь к ней: C:\Program Files\UGS\NX7.5\UGOPEN\vs_files.
2. Скопировать содержимое папки “vs_files” в каталог Visual Studio. Путь к каталогу по умолчанию: C:\Program Files\Microsoft Visual Studio 9.0. При копировании заменить существующие файлы и папки.

При запуске Visual Studio после выполнения вышеописанных действий в диалог создания проектов добавляются три «мастера». Рассмотрим работу с «мастером» для C# . В качестве типа проектов выберем C# (рис. 1.1). В открывшемся справа меню выберем NX7 Open C# Wizard. Далее зададим имя и путь сохранения проекта, нажмем кнопку ОК.

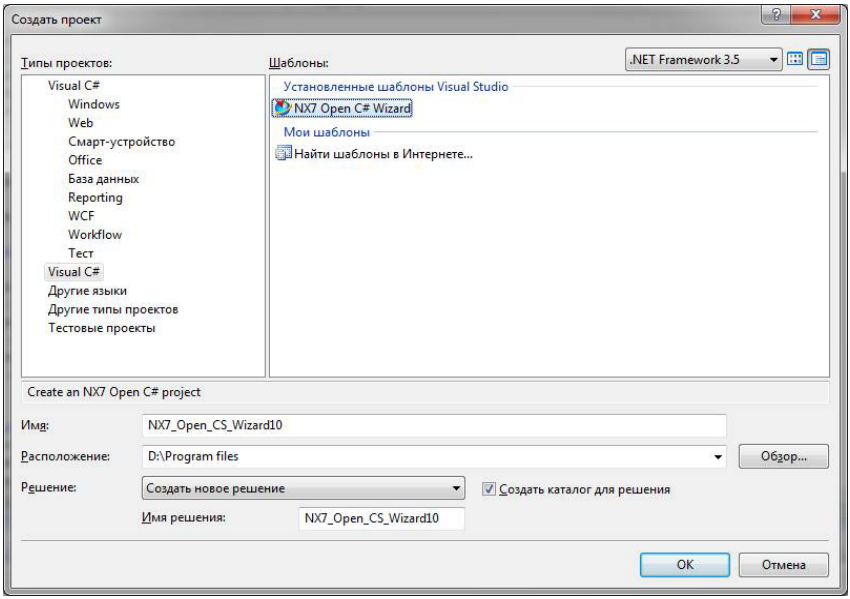


Рис. 1.1. Мастер создания проекта

В открывшемся окне нажмем кнопку Next. В следующем окне (рис. 1.2) выбираем тип создаваемого приложения и типы используемых API. Выберем «An internal application that can be activated from an NX session» (DLL), что соответствует созданию прикладной подпрограммы в виде динамической библиотеки. В пункте «UseAPIs» установим галочки напротив двух предлагаемых вариантов и нажмем кнопку Next.

В открывшемся окне (рис. 1.3) выберем опции загрузки и выгрузки разрабатываемой библиотеки. Выберем в качестве опции загрузки «Explicitly (Main)». В качестве опции выгрузки выберем «Automatically, when the application completes». В данном случае библиотека будет выгружена вместе с NX. Другие варианты позволяют выгружать библиотеку после завершения ее работы или через специальный диалог.

После завершения настройки нажмем кнопку Finish, после чего откроется окно написания программного кода.

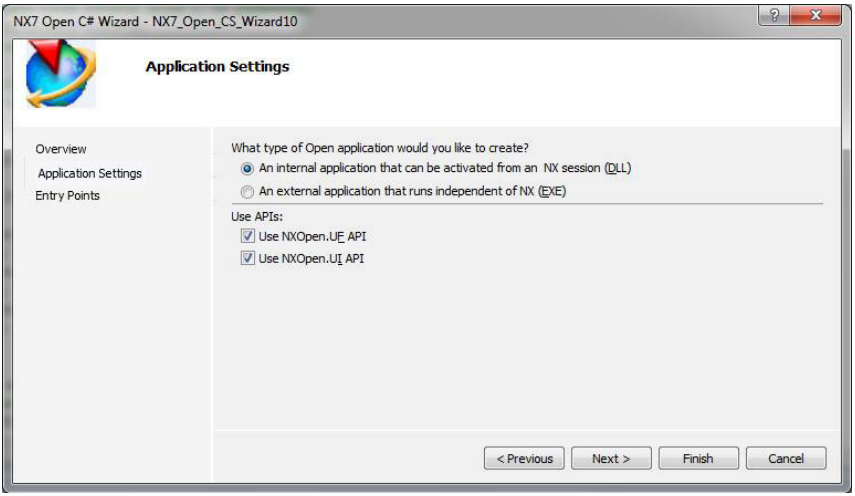


Рис. 1.2. Выбор типа создаваемого приложения и типа используемых API

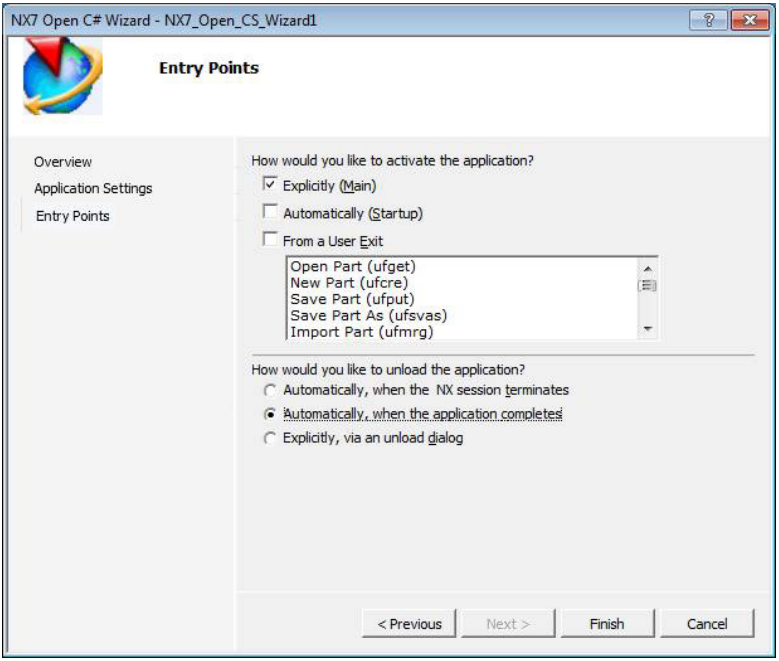


Рис. 1.3. Настройка загрузки и выгрузки библиотеки

В Visual Studio найдите процедуру Main, являющуюся точкой входа запускаемого приложения:

```
//------------------------------------------------------------
//Explicit Activation
//This entry point is used to activate the application explicitly
//------------------------------------------------------------
public static int Main(string[] args)
{
intretValue = 0;
try
{
theProgram = new Program();
//TODO: Add your application code here
theProgram.Dispose();
}
catch(NXOpen.NXExceptionex)
{
// ---- Enter your exception handling code here -
}
Return retValue;
}
```

Вместо строки «//TODO: Add your application code here» вводится программный код вашего приложения.

Приступим к созданию примитива «отрезок». Рассматриваемая библиотека начинается со следующей конструкции:

```
Tag UFPart1;
String name1 = "1";
Int units1 = 1;
theUfSession.Part.New(name1, units1, out UFPart1);
```

Первые три строки текста применяются для описания переменных. В данном контексте переменная UFPart1 является деталью (создается новый проект), ее тип задается как тэг (типовой объект NX). name1 - строковая переменная, которая задает имя файла детали. units1 – переменная

целочисленного типа, определяющая тип системы мер (1 – метрическая система, 2 – английская). Четвертая строка отвечает за создание новой детали. Переменные name1 и units1 для нее являются входными, а UFPart1 выходными данными.

В последующем блоке описываются 2 однотипные переменные, соответствующие конечным точкам отрезка.

```
double[] l1_endpt1 = { 0, 0, 0.00 };
double[] l1_endpt2 = { -100, 0, 0.00 };
```

Тип переменных – массив вещественных чисел. Заполнение массивов осуществляется тройками чисел, являющимися координатами точек в 3D пространстве; координаты указываются в порядке X, Y, Z. Так как эскиз плоский, координата по третьей оси (Z) постоянна (в данном случае равна 0).

Последующий блок создает новую структуру.

```
UFCurve.Line l1 = new UFCurve.Line();
```

Структура l1 относится к специальному типу NX, соответствующему такому объекту, как отрезок. В последующем фрагменте программного кода задаются конечные точки отрезка.

```
l1.start_point = new double[3];
l1.start_point[0] = l1_endpt1[0];
l1.start_point[1] = l1_endpt1[1];
l1.start_point[2] = l1_endpt1[2];
l1.end_point = new double[3];
l1.end_point[0] = l1_endpt2[0];
l1.end_point[1] = l1_endpt2[1];
l1.end_point[2] = l1_endpt2[2];
```

Первая строка создает массивы вещественных чисел, в которые будут записываться тройки координат точек отрезка. Строки 2 – 4 задают координаты по X, Y, Z начальных точек отрезков. Строки 6 – 8 задают конечные точки отрезков.

Текст, приведенный ниже, отвечает за создание отрезка в 3D-пространстве.

```
Tag[] objarray = new Tag[1];
theUfSession.Curve.CreateLine(ref l1, out objarray[0]);
```

Первая строка создает переменную objarray, представляющую собой массив тэгов. Каждая последующая строка создает отрезок в 3D-пространстве с последующим его отображением на экране. В качестве входных данных используются координаты конечных точек отрезка. Выходная информация, представляющая тэг каждого отрезка, записывается поэлементно в массив objarray.

Для сохранения результатов построений можно использовать команду:

theUfSession.Part.Save(); - сохраняет деталь в файл с именем, заданным переменной name1, по умолчанию путь к сохраненному файлу находится по адресу: C:\ProgramFiles\UGS\NX7.5\UGII. При сохранении файла может потребоваться его ручное удаление при повторных запусках библиотеки.

Далее необходимо откомпилировать проект. Для этого можно воспользоваться клавишей F6. Если все сделано верно, в окне ошибок и предупреждений будет выведено сообщение об отсутствии ошибок. Могут присутствовать предупреждения, не препятствующие нормальной работе приложения, однако рекомендуется тщательно с ними ознакомиться и по возможности их устранить. Результатом компиляции является файл с расширением .dll и именем, заданным при создании проекта. Файл размещается в каталоге «Debug», расположенном в папке «Bin».

Пример пути:
C:\Projects\NX7_Open_CS_Wizard1\NX7_Open_CS_Wizard1\bin\Debug.

Внимание: NX 7.5 не поддерживает работу с кириллическими файловыми именами, поэтому все пути и

имена, с которыми работает NX, должны быть выполнены исключительно латиницей. Для запуска библиотеки необходимо загрузить NX от имени администратора, после чего в главном меню выбрать последовательность команд «Файл – Выполнить – NX функция пользователя» или воспользоваться сочетанием клавиш Ctrl-U. В запустившемся диалоге открытия файла следует указать требуемую библиотеку и нажать ОК.

Результат выполнения созданной ранее библиотеки показан на рис. 1.4.

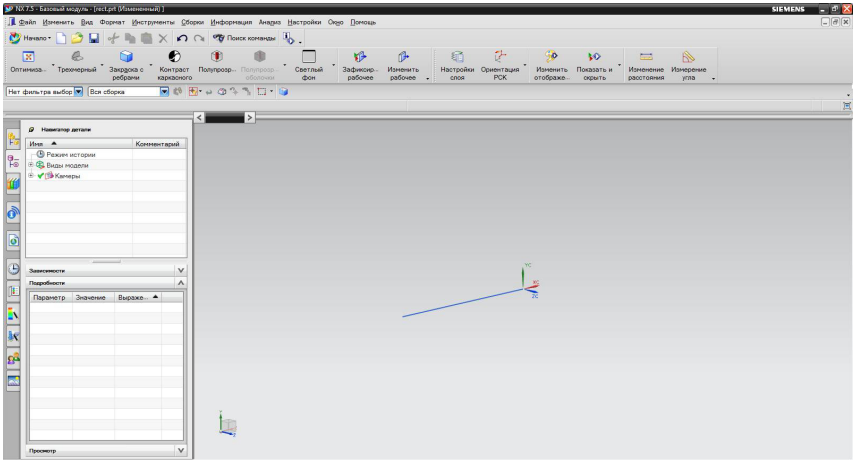


Рис. 1.4

Далее рассмотрим рисование 2D-фигуры «прямоугольник», состоящей из 4 отрезков, соединенных между собой. Необходимо открыть новый проект и создать отрезки.

```
//создание нового проекта
Tag UFPart1;

string name1 = "rect";
int units1 = 1;
```

```
theUfSession.Part.New(name1, units1, out UFPart1);

double[] l1_endpt1 = { 0, 0, 0.00 }; //начальная точка
double[] l1_endpt2 = { -100, 0, 0.00 }; //конечная точка
double[] l2_endpt1 = { -100, 0, 0.00 };
double[] l2_endpt2 = { -100, 30, 0.00 };
double[] l3_endpt1 = { -100, 30, 0.00 };
double[] l3_endpt2 = { 0, 30, 0.00 };
double[] l4_endpt1 = { 0, 30, 0.00 };
double[] l4_endpt2 = { 0, 0, 0.00 };

UFCurve.Line l1 = new UFCurve.Line();
UFCurve.Line l2 = new UFCurve.Line();
UFCurve.Line l3 = new UFCurve.Line();
UFCurve.Line l4 = new UFCurve.Line();

l1.start_point = new double[3];
l1.start_point[0] = l1_endpt1[0];
l1.start_point[1] = l1_endpt1[1];
l1.start_point[2] = l1_endpt1[2];
l1.end_point = new double[3];
l1.end_point[0] = l1_endpt2[0];
l1.end_point[1] = l1_endpt2[1];
l1.end_point[2] = l1_endpt2[2];
l2.start_point = new double[3];
l2.start_point[0] = l2_endpt1[0];
l2.start_point[1] = l2_endpt1[1];
l2.start_point[2] = l2_endpt1[2];
l2.end_point = new double[3];
l2.end_point[0] = l2_endpt2[0];
l2.end_point[1] = l2_endpt2[1];
l2.end_point[2] = l2_endpt2[2];
l3.start_point = new double[3];
l3.start_point[0] = l3_endpt1[0];
l3.start_point[1] = l3_endpt1[1];
l3.start_point[2] = l3_endpt1[2];
```

```
l3.end_point = new double[3];
l3.end_point[0] = l3_endpt2[0];
l3.end_point[1] = l3_endpt2[1];
l3.end_point[2] = l3_endpt2[2];
l4.start_point = new double[3];
l4.start_point[0] = l4_endpt1[0];
l4.start_point[1] = l4_endpt1[1];
l4.start_point[2] = l4_endpt1[2];
l4.end_point = new double[3];
l4.end_point[0] = l4_endpt2[0];
l4.end_point[1] = l4_endpt2[1];
l4.end_point[2] = l4_endpt2[2];

Tag[] objarray1 = new Tag[4];

theUfSession.Curve.CreateLine(ref l1, out objarray1[0]);
theUfSession.Curve.CreateLine(ref l2, out objarray1[1]);
theUfSession.Curve.CreateLine(ref l3, out objarray1[2]);
theUfSession.Curve.CreateLine(ref l4, out objarray1[3]);
```

Сохраним проект, откомпилируем его и запустим в NX. Результат выполнения приведен на рис. 1.5.

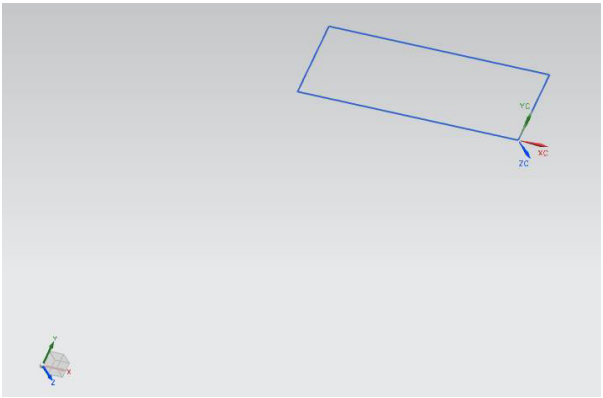


Рис. 1.5

Теперь освоим операцию «выдавливание» для создания параллелепипеда. Откроем уже созданный и сохраненный проект, в котором был построен прямоугольник. Внесем в него несколько изменений (представленный ниже код вставляется после создания прямоугольника).

```
double[] direction = { 0.0, 0.0, 1.0 };
//Переменная, задающая значения направления выдавливания
//ось CZ
double[] ref_pt = new double[3];
//Требуемая, но не используемая переменная
string taper_angle = "0.0";
//Переменная, определяющая значение уклона при
//выдавливании
string[] limit = { "0", "80" };
//Переменная, определяющая параметры начала и конца
//операции выдавливания
Tag[] f;
//f – переменная для записи указателя на объект,
//получившийся в результате операции выдавливания
theUfSession.Modl.CreateExtruded(objarray1, taper_angle, limit,
ref_pt, direction, FeatureSigns.Nullsign, out f);
```

Параметры операции «Выдавливания»:

1. objarray - массив объектов выдавливания;
2. taper_angle - угол уклона;
3. limit – начало и конец выдавливания;
4. ref_pt – пустой параметр;
5. direction – направление выдавливания (ось x,y или z);
6. FeatureSigns.Nullsign - задает булеву операцию, в данном случае операция отсутствует;
7. f - выходной параметр - указатель на результат операции.

Результат выполнения приведен на рис. 1.6.

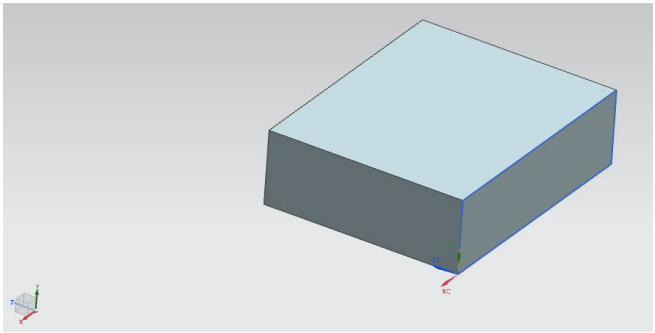


Рис. 1.6

Создадим примитив «дуга». Для этого в новый проект вставим код, представленный ниже.

```
//создание нового проекта
Tag UFPart1;
string name1 = "rect";
int units1 = 1;
theUfSession.Part.New(name1, units1, out UFPart1);

Tag[] objarray = new Tag[7];
Tag wcs, matrix;

double[] arc1_centerpt = { 0, 0, 0 };
//Переменная, содержащая значения координат центра дуги
{x,y,z}
double arc1_start_ang = 0;
//Переменная, содержащая значение угла начала дуги (в
//радианах)
double arc1_end_ang = 3.14159265358979324
/*полуокружность*/;
//Переменная, содержащая значение угла конца дуги (в
//радианах)
double arc1_radius = 50;
//Переменная, содержащая значение радиуса дуги (в радианах)
UFCurve.Arc arc1 = new UFCurve.Arc();
arc1.start_angle = arc1_start_ang;
```

```
//Начальный угол
arc1.end_angle = arc1_end_ang;
//Конечныйугол
arc1.arc_center = new double[3];
//Центр дуги 1
arc1.arc_center[0] = arc1_centerpt[0];
//Координата центра дуги по X
arc1.arc_center[1] = arc1_centerpt[1];
//Координата центра дуги по Y
arc1.arc_center[2] = arc1_centerpt[2];
//Координата центра дуги по Z
arc1.radius = arc1_radius;
//радиусдуги

theUfSession.Csys.AskWcs(out wcs);
//Получение указателя на активную систему координат
theUfSession.Csys.AskMatrixOfObject(wcs, out matrix);
//Получение идентификатора матрицы, связанного с объектом,
//указатель на который содержится в wcs
arc1.matrix_tag = matrix;
//Определение указателя матрицы дуги

theUfSession.Curve.CreateArc(ref arc1, out objarray[0]);
//рисование дуги
```

Результат выполенения программы представлен на рис. 1.7.

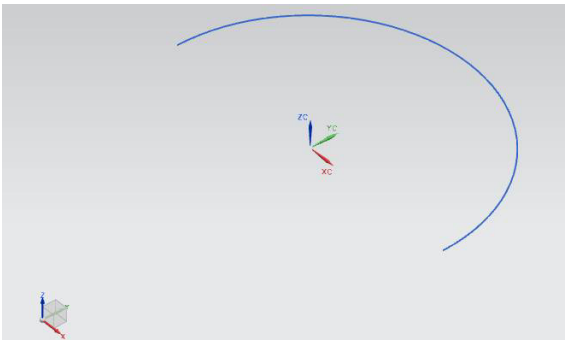


Рис. 1.7

Для того, чтобы нарисовать примитив «шар», необходимо к уже созданной дуге (полуокружности) применить операцию «вращение».

```
double[] ref_pt = new double[3];
ref_pt[0] = 0.00;
ref_pt[1] = 0.00;
ref_pt[2] = 0.00;
double[] direction = { 1.00, 0.00, 0.00 };
string[] limit = { "0", "360" };
Tag[] f;

theUfSession.Modl.CreateRevolved(objarray, limit, ref_pt,
direction, FeatureSigns.Nullsign, out f);
```

Строки 1 – 4 создают массив из 3 вещественных чисел и заполняют его нулями. В дальнейшем он будет использоваться для задания точки с нулевыми координатами, относительно которой будет осуществляться вращение. direction – аналогичный массив из трех элементов. В данной программе он отвечает за вектор (точнее его конечную точку), относительно которого осуществляется вращение. Начало вектора находится в начале координат. Переменная limit – массив строкового типа, в который заносится начальный и конечный угол для операции вращения, f – массив тэгов, выходной информации для операции «вращение». Последняя строка отвечает за операцию «вращение».

Аргументами данной операции являются:

1. objarray – массив элементов эскиза операции;
2. limit – начальный и конечный угол вращения;
3. ref_pt – базовая точка;
4. direction – вектор, относительно которого осуществляется вращение;
5. FeatureSigns.Nullsign – задает булеву операцию, в данном случае операция отсутствует.
6. f – массив тэгов, выходная информация

Результат выполнения операции представлен на рис. 1.8.

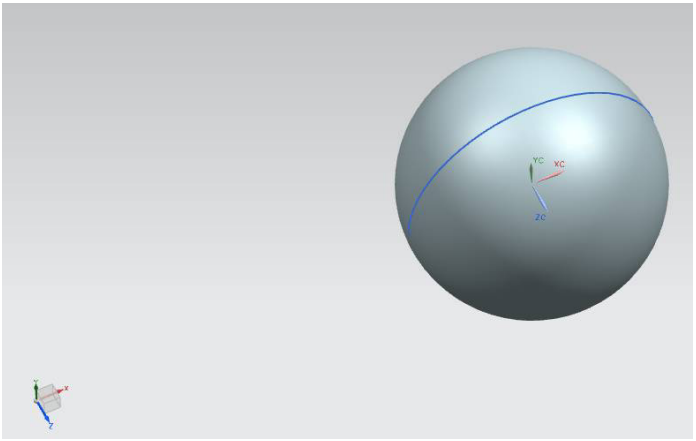


Рис. 1.8

Для сохранения детали можно использовать:

theUfSession.Part.Save(); - сохраняет деталь в файл с именем, заданным переменной name1, по умолчанию путь к сохраненному файлу находится по адресу: C:\ProgramFiles\UGS\NX7.5\UGII.

Теперь освоим методику построения цилиндра с помощью операции «выдавливания». Для этого необходимо создать новый проект и нарисовать окружность.

```
//создание нового проекта
Tag UFPart1;
string name1 = "rect";
int units1 = 1;
theUfSession.Part.New(name1, units1, out UFPart1);


Tag[] objarray = new Tag[7];
Tag wcs, matrix;

double[] arc1_centerpt = { 0, 0, 0 };
```

```
//Переменная, содержащая значения координат центра дуги
//{x,y,z}
double arc1_start_ang = 0;
//Переменная, содержащая значение угла начала дуги (в
//радианах)
Double arc1_end_ang =
3.14159265358979324*2/*окружность*/; //Переменная,
//содержащая значение угла конца дуги (в радианах)
double arc1_radius = 50;
//Переменная, содержащая значение радиуса дуги (в радианах)
UFCurve.Arc arc1 = new UFCurve.Arc();
arc1.start_angle = arc1_start_ang;
//Начальный угол
arc1.end_angle = arc1_end_ang;
//Конечный угол
arc1.arc_center = new double[3];
//Центрдуги 1
arc1.arc_center[0] = arc1_centerpt[0];
//Координата центра дуги по X
arc1.arc_center[1] = arc1_centerpt[1];
//Координата центра дуги по Y
arc1.arc_center[2] = arc1_centerpt[2];
//Координата центра дуги по Z
arc1.radius = arc1_radius;
//радиус дуги

theUfSession.Csys.AskWcs(out wcs);
//Получения указателя на активную систему координат
theUfSession.Csys.AskMatrixOfObject(wcs, out matrix);
//Получение идентификатора матрицы, связанного с объектом,
//указатель на который содержится в wcs
arc1.matrix_tag = matrix;
//Определение указателя матрицы дуги

theUfSession.Curve.CreateArc(ref arc1, out objarray[0]);
//рисование дуги
```

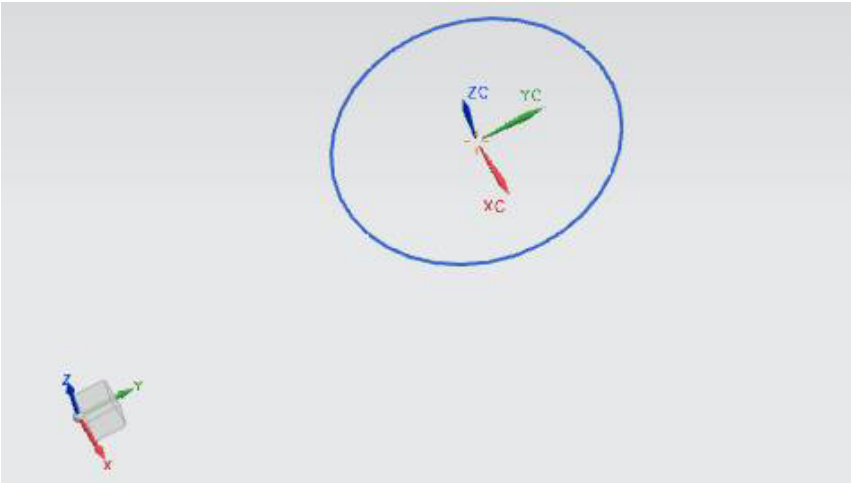


Рис. 1.9

Применим к созданной окружности уже знакомую операцию «выдавливания»:

```
double[] direction = { 0.0, 0.0, 1.0 };
//Переменная, задающая значения направления выдавливания
//ось CZ
double[] ref_pt = new double[3];
//Требуемая, но не используемая переменная
string taper_angle = "0.0";
//Переменная, определяющая значение уклона при
//выдавливании
string[] limit = { "0", "80" };
//Переменная, определяющая параметры начала и конца
//операции выдавливания
Tag[] f;
//f – переменная для записи указателя на объект,
//получившийся в результате операции выдавливания
theUfSession.Modl.CreateExtruded(objarray, taper_angle, limit,
ref_pt, direction, FeatureSigns.Nullsign, out f);
```

Результат представлен на рисунке 10.

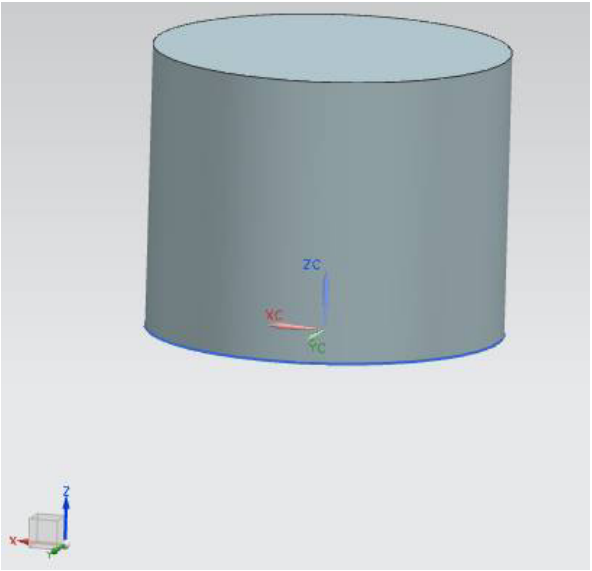


Рис. 1.10

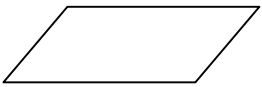
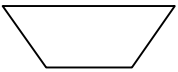
Можно создать цилиндр и другим способом: отрисовать прямоугольник, а затем применить к нему операцию «вращения».

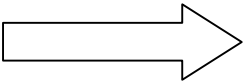
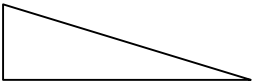
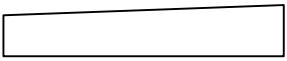
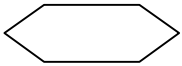
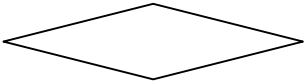
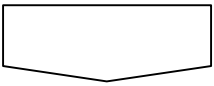
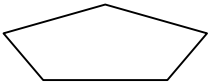
1.2. Практическая часть

1.2.1. Задание на лабораторную работу

Задание выдается преподавателем по вариантам.

Из плоской геометрической фигуры, представленной в таблице, путем «выдавливания», получить объемную фигуру.

Вариант	Геометрическая фигура	Угол выдавливаия отновительно плоскости YOZ
1		0^0
2		15^0

3		10^0
4		20^0
5		11^0
6		10^0
7		20^0
8		16^0
9		10^0
10		17^0

1.2.2. Содержание отчета по лабораторной работе

1. Название и цель работы.
2. Скриншоты с кратким описанием, соответствующие основным шагам выполненной работы.
3. Скриншоты, демонстрирующие работоспособность созданной библиотеки.
4. Листинг прграммы с комментариями.
5. Скриншоты 3D модели, сформированной Вашей библиотекой.
6. Выводы.

2. ЛАБОРАТОРНАЯ РАБОТА № 2
СОЗДАНИЕ СЛОЖНЫХ МОДЕЛЕЙ В NX

Цель работы: научиться рисовать сложные детали в NX с помощью операций «вращение», «выдавливание», освоить операцию «вырезание вращением».

2.1. Теоретическая часть

2.1.1. Общий вид и порядок построения детали

Вид с положением осей представлен на рис. 2.1.

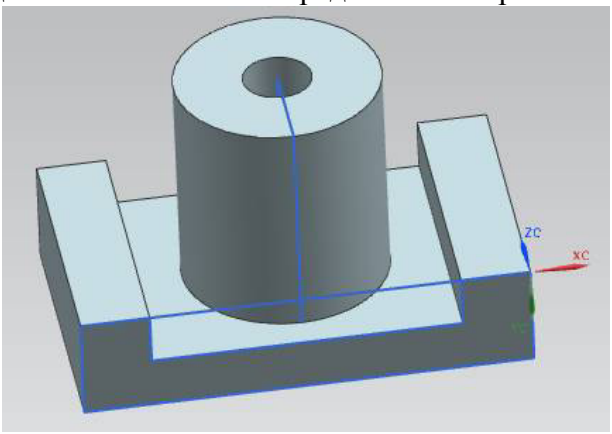


Рис. 2.1

Размеры выбираются произвольно. Цилиндр реализуется с помощью вращения прямоугольника вокруг своей оси.

Порядок создания детали:

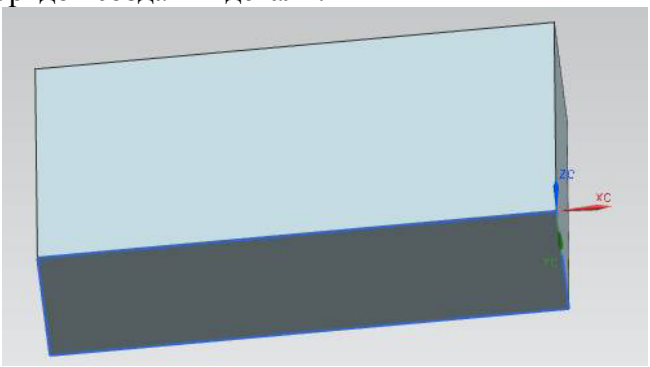


Рис. 2.2. Первый шаг

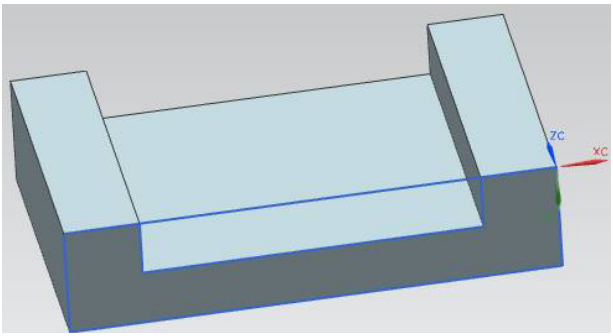


Рис. 2.3. Второй шаг

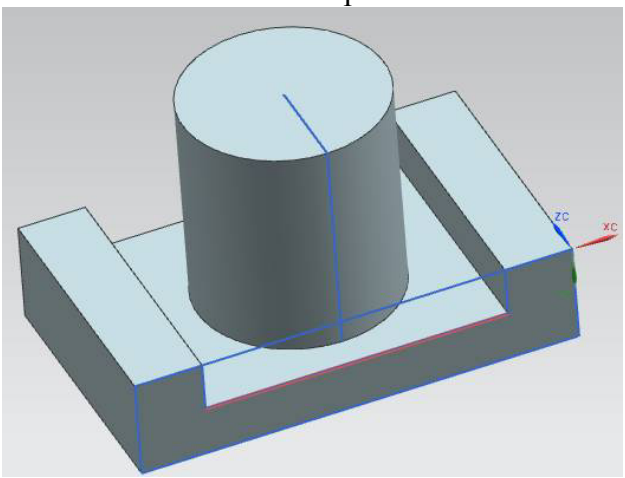


Рис. 2.4. Третий шаг

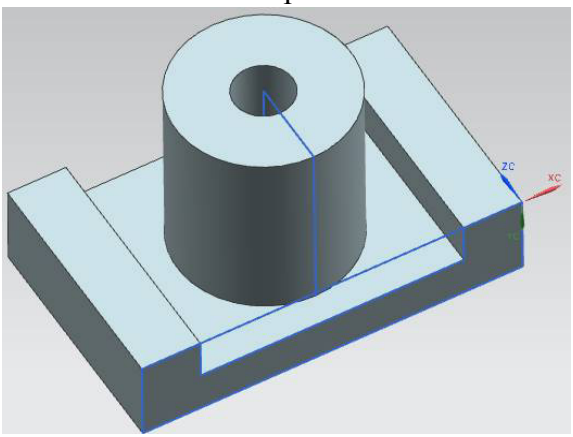


Рис. 2.5. Четвертый шаг

```
double[] l1_endpt1 = { 0, 0, 0.00 };//Координаты начальной
//точки отрезка 1
double[] l1_endpt2 = { -130, 0, 0.00 };//Координаты конечной
//точки отрезка 1
double[] l2_endpt1 = { -130, 0, 0.00 };
double[] l2_endpt2 = { -130, 30, 0.00 };
double[] l3_endpt1 = { -130, 30, 0.00 };
double[] l3_endpt2 = { 0, 30, 0.00 };
double[] l4_endpt1 = { 0, 30, 0.00 };
double[] l4_endpt2 = { 0, 0, 0.00 };
double[] l5_endpt1 = { -20, 0, 0.00 };
double[] l5_endpt2 = { -20, 15, 0.00 };
double[] l6_endpt1 = { -20, 15, 0.00 };
double[] l6_endpt2 = { -110, 15, 0.00 };
double[] l7_endpt1 = { -110, 15, 0.00 };
double[] l7_endpt2 = { -110, 0, 0.00 };
double[] l8_endpt1 = { -110, 0, 0.00 };
double[] l8_endpt2 = { -20, 0, 0.00 };
double[] l9_endpt1 = { -65, 15, 40 };
double[] l9_endpt2 = { -65, 15, 10 };
double[] l10_endpt1 = { -65, 15, 10 };
double[] l10_endpt2 = { -65, -50, 10 };
double[] l11_endpt1 = { -65, -50, 10 };
double[] l11_endpt2 = { -65, -50, 40 };
double[] l12_endpt1 = { -65, -50, 40 };
double[] l12_endpt2 = { -65, 15, 40 };
double[] l13_endpt1 = { -65, 15, 40 };
double[] l13_endpt2 = { -65, 15, 30 };
double[] l14_endpt1 = { -65, 15, 30 };
double[] l14_endpt2 = { -65, -50, 30 };
double[] l15_endpt1 = { -65, -50, 30 };
double[] l15_endpt2 = { -65, -50, 40 };
double[] l16_endpt1 = { -65, -50, 40 };
double[] l16_endpt2 = { -65, 15, 40 };

UFCurve.Line l1 = new UFCurve.Line();
```

```
UFCurve.Line l2 = new UFCurve.Line();
UFCurve.Line l3 = new UFCurve.Line();
UFCurve.Line l4 = new UFCurve.Line();
UFCurve.Line l5 = new UFCurve.Line();
UFCurve.Line l6 = new UFCurve.Line();
UFCurve.Line l7 = new UFCurve.Line();
UFCurve.Line l8 = new UFCurve.Line();
UFCurve.Line l9 = new UFCurve.Line();
UFCurve.Line l10 = new UFCurve.Line();
UFCurve.Line l11 = new UFCurve.Line();
UFCurve.Line l12 = new UFCurve.Line();
UFCurve.Line l13 = new UFCurve.Line();
UFCurve.Line l14 = new UFCurve.Line();
UFCurve.Line l15 = new UFCurve.Line();
UFCurve.Line l16 = new UFCurve.Line();

l1.start_point = new double[3];
l1.start_point[0] = l1_endpt1[0];
l1.start_point[1] = l1_endpt1[1];
l1.start_point[2] = l1_endpt1[2];
l1.end_point = new double[3];
l1.end_point[0] = l1_endpt2[0];
l1.end_point[1] = l1_endpt2[1];
l1.end_point[2] = l1_endpt2[2];
l2.start_point = new double[3];
l2.start_point[0] = l2_endpt1[0];
l2.start_point[1] = l2_endpt1[1];
l2.start_point[2] = l2_endpt1[2];
l2.end_point = new double[3];
l2.end_point[0] = l2_endpt2[0];
l2.end_point[1] = l2_endpt2[1];
l2.end_point[2] = l2_endpt2[2];
l3.start_point = new double[3];
l3.start_point[0] = l3_endpt1[0];
l3.start_point[1] = l3_endpt1[1];
l3.start_point[2] = l3_endpt1[2];
```

```
l3.end_point = new double[3];
l3.end_point[0] = l3_endpt2[0];
l3.end_point[1] = l3_endpt2[1];
l3.end_point[2] = l3_endpt2[2];
l4.start_point = new double[3];
l4.start_point[0] = l4_endpt1[0];
l4.start_point[1] = l4_endpt1[1];
l4.start_point[2] = l4_endpt1[2];
l4.end_point = new double[3];
l4.end_point[0] = l4_endpt2[0];
l4.end_point[1] = l4_endpt2[1];
l4.end_point[2] = l4_endpt2[2];
l5.start_point = new double[3];
l5.start_point[0] = l5_endpt1[0];
l5.start_point[1] = l5_endpt1[1];
l5.start_point[2] = l5_endpt1[2];
l5.end_point = new double[3];
l5.end_point[0] = l5_endpt2[0];
l5.end_point[1] = l5_endpt2[1];
l5.end_point[2] = l5_endpt2[2];
l6.start_point = new double[3];
l6.start_point[0] = l6_endpt1[0];
l6.start_point[1] = l6_endpt1[1];
l6.start_point[2] = l6_endpt1[2];
l6.end_point = new double[3];
l6.end_point[0] = l6_endpt2[0];
l6.end_point[1] = l6_endpt2[1];
l6.end_point[2] = l6_endpt2[2];
l7.start_point = new double[3];
l7.start_point[0] = l7_endpt1[0];
l7.start_point[1] = l7_endpt1[1];
l7.start_point[2] = l7_endpt1[2];
l7.end_point = new double[3];
l7.end_point[0] = l7_endpt2[0];
l7.end_point[1] = l7_endpt2[1];
l7.end_point[2] = l7_endpt2[2];
```

```
l8.start_point = new double[3];
l8.start_point[0] = l8_endpt1[0];
l8.start_point[1] = l8_endpt1[1];
l8.start_point[2] = l8_endpt1[2];
l8.end_point = new double[3];
l8.end_point[0] = l8_endpt2[0];
l8.end_point[1] = l8_endpt2[1];
l8.end_point[2] = l8_endpt2[2];
l9.start_point = new double[3];
l9.start_point[0] = l9_endpt1[0];
l9.start_point[1] = l9_endpt1[1];
l9.start_point[2] = l9_endpt1[2];
l9.end_point = new double[3];
l9.end_point[0] = l9_endpt2[0];
l9.end_point[1] = l9_endpt2[1];
l9.end_point[2] = l9_endpt2[2];
l10.start_point = new double[3];
l10.start_point[0] = l10_endpt1[0];
l10.start_point[1] = l10_endpt1[1];
l10.start_point[2] = l10_endpt1[2];
l10.end_point = new double[3];
l10.end_point[0] = l10_endpt2[0];
l10.end_point[1] = l10_endpt2[1];
l10.end_point[2] = l10_endpt2[2];
l11.start_point = new double[3];
l11.start_point[0] = l11_endpt1[0];
l11.start_point[1] = l11_endpt1[1];
l11.start_point[2] = l11_endpt1[2];
l11.end_point = new double[3];
l11.end_point[0] = l11_endpt2[0];
l11.end_point[1] = l11_endpt2[1];
l11.end_point[2] = l11_endpt2[2];
l12.start_point = new double[3];
l12.start_point[0] = l12_endpt1[0];
l12.start_point[1] = l12_endpt1[1];
l12.start_point[2] = l12_endpt1[2];
```

```
l12.end_point = new double[3];
l12.end_point[0] = l12_endpt2[0];
l12.end_point[1] = l12_endpt2[1];
l12.end_point[2] = l12_endpt2[2];
l13.start_point = new double[3];
l13.start_point[0] = l13_endpt1[0];
l13.start_point[1] = l13_endpt1[1];
l13.start_point[2] = l13_endpt1[2];
l13.end_point = new double[3];
l13.end_point[0] = l13_endpt2[0];
l13.end_point[1] = l13_endpt2[1];
l13.end_point[2] = l13_endpt2[2];
l14.start_point = new double[3];
l14.start_point[0] = l14_endpt1[0];
l14.start_point[1] = l14_endpt1[1];
l14.start_point[2] = l14_endpt1[2];
l14.end_point = new double[3];
l14.end_point[0] = l14_endpt2[0];
l14.end_point[1] = l14_endpt2[1];
l14.end_point[2] = l14_endpt2[2];
l15.start_point = new double[3];
l15.start_point[0] = l15_endpt1[0];
l15.start_point[1] = l15_endpt1[1];
l15.start_point[2] = l15_endpt1[2];
l15.end_point = new double[3];
l15.end_point[0] = l15_endpt2[0];
l15.end_point[1] = l15_endpt2[1];
l15.end_point[2] = l15_endpt2[2];
l16.start_point = new double[3];
l16.start_point[0] = l16_endpt1[0];
l16.start_point[1] = l16_endpt1[1];
l16.start_point[2] = l16_endpt1[2];
l16.end_point = new double[3];
l16.end_point[0] = l16_endpt2[0];
l16.end_point[1] = l16_endpt2[1];
l16.end_point[2] = l16_endpt2[2];
```

```
//создание первого прямоугольника для основания
Tag[] objarray1 = new Tag[4];
theUfSession.Curve.CreateLine(ref l1, out objarray1[0]);
theUfSession.Curve.CreateLine(ref l2, out objarray1[1]);
theUfSession.Curve.CreateLine(ref l3, out objarray1[2]);
theUfSession.Curve.CreateLine(ref l4, out objarray1[3]);
//создание второго прямоугольника для выемки в основании
Tag[] objarray2 = new Tag[4];
theUfSession.Curve.CreateLine(ref l5, out objarray2[0]);
theUfSession.Curve.CreateLine(ref l6, out objarray2[1]);
theUfSession.Curve.CreateLine(ref l7, out objarray2[2]);
theUfSession.Curve.CreateLine(ref l8, out objarray2[3]);
//создание прямоугольника для рисования цилиндра
Tag[] objarray3 = new Tag[4];
theUfSession.Curve.CreateLine(ref l9, out objarray3[0]);
theUfSession.Curve.CreateLine(ref l10, out objarray3[1]);
theUfSession.Curve.CreateLine(ref l11, out objarray3[2]);
theUfSession.Curve.CreateLine(ref l12, out objarray3[3]);
//создание прямоугольника для выемки
Tag[] objarray4 = new Tag[4];
theUfSession.Curve.CreateLine(ref l13, out objarray4[0]);
theUfSession.Curve.CreateLine(ref l14, out objarray4[1]);
theUfSession.Curve.CreateLine(ref l15, out objarray4[2]);
theUfSession.Curve.CreateLine(ref l16, out objarray4[3]);

//параллелепипеда для основания
double[] direction1 = { 0.0, 0.0, 1.0 };
//Переменная, задающая значения направления выдавливания
//ось CZ
double[] ref_pt1 = new double[3];
//Требуемая, но не используемая переменная
string taper_angle1 = "0.0";
//Переменная, определяющая значение уклона при
//выдавливании
string[] limit1 = { "0", "80" };
```

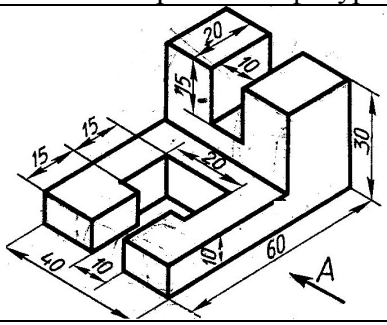
```
//Переменная, определяющая начало и конец выдавливания
Tag[] f1;
//Указатель на результат операции
theUfSession.Modl.CreateExtruded(objarray1, taper_angle1,
limit1, ref_pt1, direction1, FeatureSigns.Nullsign, out f1);

//создание параллелепипеда для выемки в основании
double[] direction2 = { 0.0, 0.0, 1.0 };
//Переменная задающая значения направления выдавливания
//ось CZ
double[] ref_pt2 = new double[3];
//Требуемая, но не используемая переменная
string taper_angle2 = "0.0";
//Переменная, определяющая значение уклона при
//выдавливании
string[] limit2 = { "0", "80" };
//Переменная, определяющая начало и конец выдавливания
Tag[] f2;
//Указатель на результат операции выдавливания
theUfSession.Modl.CreateExtruded(objarray2, taper_angle2,
limit2, ref_pt2, direction2, FeatureSigns.Negative, out f2);

//вращения прямоугольника для создания цилиндра
double[] ref_pt3 = new double[3];
//задание точки, относительно которой будет выполняться
//вращение
ref_pt3[0] = -65;
ref_pt3[1] = 0.00;
ref_pt3[2] = 40;
double[] direction3 = { 0.00, 1.00, 0.00 };
//направление вращения
string[] limit3 = { "0", "360" };
//начальный и конечный угол вращения
Tag[] f3;
//указатель на результат операции вращения
```

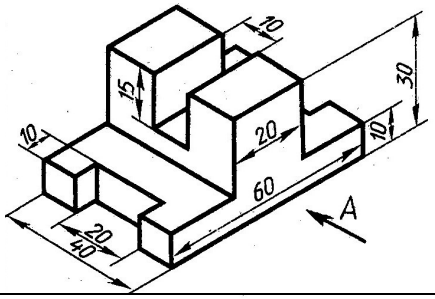
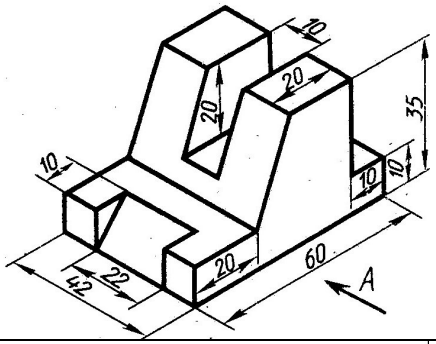
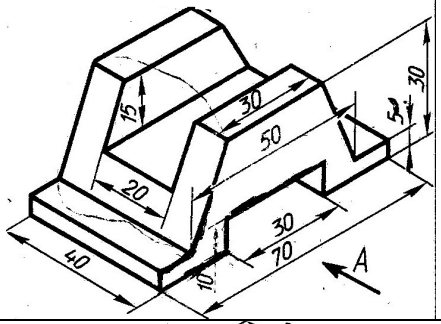
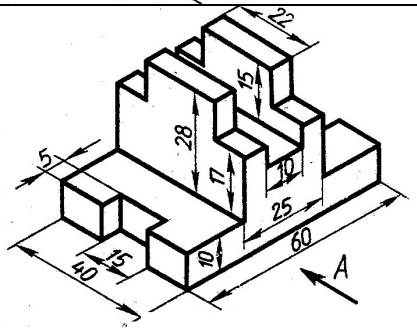
```
theUfSession.Modl.CreateRevolved(objarray3, limit3, ref_pt3,
direction3, FeatureSigns.Nullsign, out f3);

//Создание операции "вырезания вращением"
double[] ref_pt4 = new double[3];
ref_pt4[0] = -65;
ref_pt4[1] = 0.00;
ref_pt4[2] = 40;
double[] direction4 = { 0.00, 1.00, 0.00 };
string[] limit4 = { "0", "360" };
Tag[] f4;
//Параметр FeatureSigns функции CreateRevolved должен быть
//определен как Negative (вычитание)
theUfSession.Modl.CreateRevolved(objarray4, limit4, ref_pt4,
direction4, FeatureSigns.Negative, out f4);
```

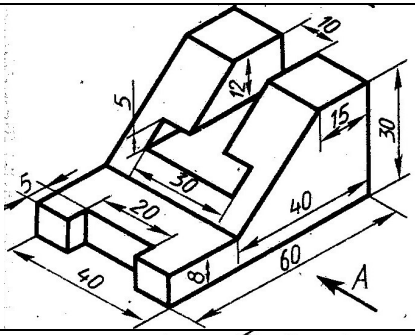
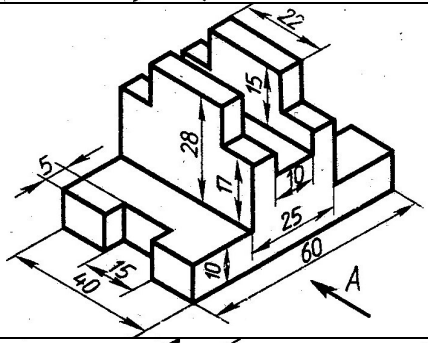
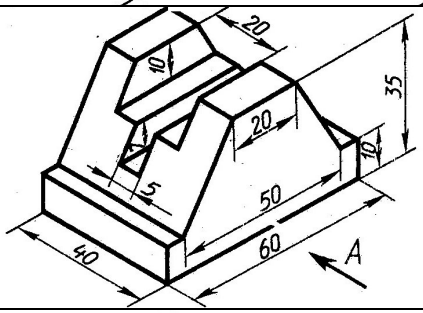
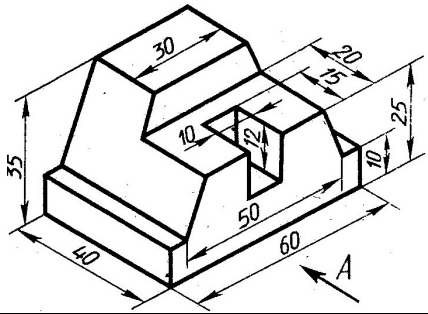
2.2. Практическая часть

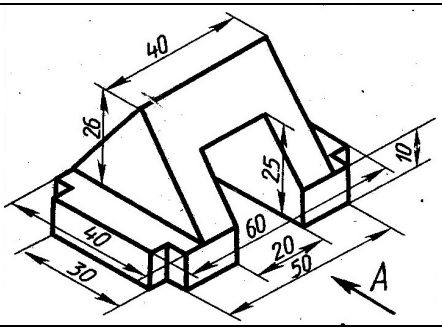
2.2.1. Задание на лабораторную работу

Получить следующие геометрические фигуры

Вариант	Геометрическая фигура
1	

2	 10 15 30 10 20 10 60 A 20 40
3	 10 20 20 35 10 10 20 60 A 22 42
4	 15 30 50 5 30 20 30 70 A 40 10
5	 15 28 5 17 10 25 60 A 10 15 40

6	 10 12 5 15 30 5 30 20 40 8 60 40 A
7	 22 15 28 5 17 10 25 60 15 10 40 A
8	 20 10 35 20 10 5 50 40 60 A
9	 30 20 15 25 10 12 10 35 50 40 60 A

10	

2.2.2. Содержание отчета по лабораторной работе

1. Название и цель работы.
2. Скриншоты с кратким описанием, соответствующие основным шагам выполненной работы.
3. Скриншоты, демонстрирующие работоспособность созданной библиотеки.
4. Листинг прграммы с комментариями.
5. Скриншоты 3D модели, сформированной Вашей библиотекой.
6. Выводы.

3. ЛАБОРАТОРНАЯ РАБОТА № 3
СОЗДАНИЕ ПАРАМЕТРИЧЕСКИ ЗАДАННЫХ ОБЪЕКТОВ

Цель работы: научиться параметрически задавать параметры объектов, освоить операции «скругление» и «резьба».

3.1. Теоретическая часть

3.1.1. Общий вид и порядок построения детали

Вид с положением осей представлен на рис. 3.1.

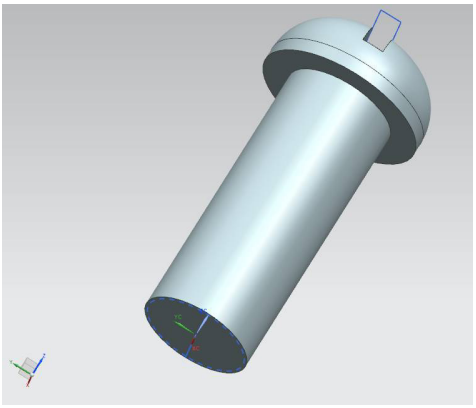


Рис. 3.1. 3D-модель объекта

Пунктирная окружность синего цвета, изображенная с нижнего торца болта, является отображением резьбы.

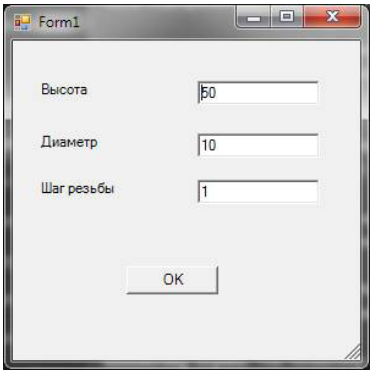


Рис. 3.2. Форма задания параметров болта

После создания нового проекта в Visual Studio необходимо добавить в него форму. Для этого нужно в окне Solution Explorer правой кнопкой мыши щелкнуть по названию проекта и выполнить последовательность Add – WindowsForm.

После этого следует добавить на форму необходимые элементы управления из окна ToolBox (панель элементов, находится в левой части интегрированной среды разработки Visual Studio). Добавим три элемента textBox, которые используются для ввода данных, три элемента Label, и кнопку.

В основном файле проекта Programm.cs добавим пространство имен:

```
using System.Collections;
```

для использования объекта ArrayList.

Ниже добавим переменные, значения которых будут вводиться с формы.

```
public static double height;//Высота, мм
public static double Radius;//радиус, мм
public static double step;//шаг резьбы, мм
```

Так будет выглядеть начало файла Programm.cs:

```
using System;
using NXOpen;
using NXOpen.UF;
using System.Collections;

public class Program
{
// class members
private static Session theSession;
private static UI theUI;
private static UFSession theUfSession;
public static Program theProgram;
public static bool isDisposeCalled;
```

```
public static double height;//Высота, мм
public static double Radius;//радиус, мм
public static double step;//шагрезьбы, мм
```

Дважды щелкнув по кнопке на форме, создастся код обработчика события нажатия на кнопку, в нем добавим следующий код:

```
namespace NX7_Open_CS_Wizard6
{
public partial class Form1 : Form
   {
public Form1()
      {
         InitializeComponent();
      }
private void button1_Click(object sender, EventArgs e)
      {
Program.height = Convert.ToDouble(textBox1.Text);
Program.Radius = Convert.ToDouble(textBox2.Text);
Program.step = Convert.ToDouble(textBox3.Text);
Close();
      }
   }
}
```

В файле Programm.cs, в процедуре Main, перед кодом, описывающим геометрические построения, необходимо задать команду открытия формы при запуске приложения:

```
NX7_Open_CS_Wizard6.Form1 f = new
NX7_Open_CS_Wizard6.Form1();
f.ShowDialog();
```

(Обратите внимание на цифру в конце имени проекта, котрая должна соответствовать ввашему номеру проекта).

При открытии DLL файла в NX откроется форма, в которую необходимо ввести необходимые значения и нажать

на кнопку. Далее будут использоваться переменные height, Radius, stepсодержащие введенные значения.

Создадим четырехугольник из которого вращением будет создан цилиндр (рис. 3.3).

```
//создание нового проекта
Tag UFPart1;
string name1 = "bolt";
int units1 = 1;
theUfSession.Part.New(name1, units1, out UFPart1);

UFCurve.Line segment0 = new UFCurve.Line();
UFCurve.Line segment1 = new UFCurve.Line();
UFCurve.Line segment2 = new UFCurve.Line();
UFCurve.Line segment3 = new UFCurve.Line();

segment0.start_point = new double[3];
segment0.start_point[0] = 0.00;
segment0.start_point[1] = 0.00;
segment0.start_point[2] = 0.00;
segment0.end_point = new double[3];
segment0.end_point[0] = 0.00;
segment0.end_point[1] = 0.00;
segment0.end_point[2] = height;

segment1.start_point = new double[3];
segment1.start_point[0] = 0.00;
segment1.start_point[1] = 0.00;
segment1.start_point[2] = height;
segment1.end_point = new double[3];
segment1.end_point[0] = Radius;
segment1.end_point[1] = 0.00;
segment1.end_point[2] = height;

segment2.start_point = new double[3];
segment2.start_point[0] = Radius;
segment2.start_point[1] = 0.00;
```

```
segment2.start_point[2] = height;
segment2.end_point = new double[3];
segment2.end_point[0] = Radius;
segment2.end_point[1] = 0.00;
segment2.end_point[2] = 0.00;

segment3.start_point = new double[3];
segment3.start_point[0] = Radius;
segment3.start_point[1] = 0.00;
segment3.start_point[2] = 0.00;
segment3.end_point = new double[3];
segment3.end_point[0] = 0.00;
segment3.end_point[1] = 0.00;
segment3.end_point[2] = 0.00;

Tag[] BoltArray = new Tag[5];
theUfSession.Curve.CreateLine(ref segment0, out BoltArray[0]);
theUfSession.Curve.CreateLine(ref segment1, out BoltArray[1]);
theUfSession.Curve.CreateLine(ref segment2, out BoltArray[2]);
theUfSession.Curve.CreateLine(ref segment3, out BoltArray[3]);
double[] ref_pt1 = new double[3];
ref_pt1[0] = 0.00;
ref_pt1[1] = 0.00;
ref_pt1[2] = 0.00;

double[] direction1 = { 0.00, 0.00, 1.00 };
string[] limit1 = { "0", "360" };
Tag[] features1;
theUfSession.Modl.CreateRevolved(BoltArray, limit1, ref_pt1,
direction1, FeatureSigns.Nullsign, out features1);
```

Рис. 3.3. Результат работы программного кода

Создадим ещё один четырехугольник над цилидром, вращая который, получим заготовку под «шляпку» болта (рис. 3.4). Операция «вращение» выполним с параметром Positive, чтобы соединить две части.

```
Tag[] features2;

UFCurve.Line segment4 = new UFCurve.Line();
UFCurve.Line segment5 = new UFCurve.Line();
UFCurve.Line segment6 = new UFCurve.Line();
UFCurve.Line segment7 = new UFCurve.Line();

segment4.start_point = new double[3];
segment4.start_point[0] = 0.00;
segment4.start_point[1] = 0.00;
segment4.start_point[2] = height;
segment4.end_point = new double[3];
segment4.end_point[0] = Radius * 1.5;
```

```
segment4.end_point[1] = 0.00;
segment4.end_point[2] = height;

segment5.start_point = new double[3];
segment5.start_point[0] = Radius * 1.5;
segment5.start_point[1] = 0.00;
segment5.start_point[2] = height;
segment5.end_point = new double[3];
segment5.end_point[0] = Radius * 1.5;
segment5.end_point[1] = 0.00;
segment5.end_point[2] = height * 1.25;

segment6.start_point = new double[3];
segment6.start_point[0] = Radius * 1.5;
segment6.start_point[1] = 0.00;
segment6.start_point[2] = height * 1.25;
segment6.end_point = new double[3];
segment6.end_point[0] = 0.00;
segment6.end_point[1] = 0.00;
segment6.end_point[2] = height * 1.25;

segment7.start_point = new double[3];
segment7.start_point[0] = 0.00;
segment7.start_point[1] = 0.00;
segment7.start_point[2] = height * 1.25;
segment7.end_point = new double[3];
segment7.end_point[0] = 0.00;
segment7.end_point[1] = 0.00;
segment7.end_point[2] = height;

Tag[] BoltArray1 = new Tag[5];
theUfSession.Curve.CreateLine(ref segment4, out BoltArray1[0]);
theUfSession.Curve.CreateLine(ref segment5, out BoltArray1[1]);
theUfSession.Curve.CreateLine(ref segment6, out BoltArray1[2]);
theUfSession.Curve.CreateLine(ref segment7, out BoltArray1[3]);
```

```
double[] ref_pt2 = new double[3];
ref_pt1[0] = 0.00;
ref_pt1[1] = 0.00;
ref_pt1[2] = height;

double[] direction2 = { 0.00, 0.00, 1.00 };
string[] limit2 = { "0", "360" };

theUfSession.Modl.CreateRevolved(BoltArray1, limit2, ref_pt2,
direction2, FeatureSigns.Positive, out features2);
```

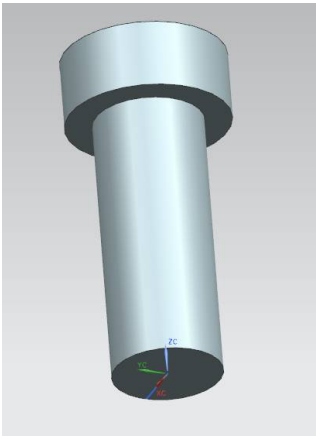


Рис. 3.4. Построенная программно «шляпка» болта

Следующая команда используется для скругления «шляпки» болта (рис. 3.5).

```
Tag feat1 = features2[0];
Tag cyl_tag, obj_id_camf, blend1;
Tag[] Edge_array_cyl, list1, list2;
int ecount;

theUfSession.Modl.AskFeatBody(feat1, out cyl_tag);
theUfSession.Modl.AskBodyEdges(cyl_tag, out
Edge_array_cyl);
theUfSession.Modl.AskListCount(Edge_array_cyl, out
```

```
ecount);
ArrayList arr_list1 = new ArrayList();
ArrayList arr_list2 = new ArrayList();

for (int ii = 0; ii < ecount; ii++)
{
Tag edge;
   theUfSession.Modl.AskListItem(Edge_array_cyl, ii, out edge);
if (ii == 2)
   {
      arr_list2.Add(edge);
   }
}
list2 = (Tag[])arr_list2.ToArray(typeof(Tag));

int allow_smooth = 0;
int allow_cliff = 0;
int allow_notch = 0;
double vrb_tol = 0.0;
string rad1 = Convert.ToInt32(Radius).ToString();
theUfSession.Modl.CreateBlend(rad1, list2, allow_smooth,
            allow_cliff, allow_notch, vrb_tol, out blend1);
```

Рис. 3.5. Скругление «шляпки» болта

Теперь необходимо добавить символическую резьбу на цилидр меньшего радиуса. Для этого добавим следующий код.

```
Tag feat = features1[0];
Tag[] FeatFaces;
int FacesCount, FaceType, FaceNormDir;
Tag face, s_face, c_face, feature_eid;
double[] point = new double[3];
double[] dir = new double[3];
double[] box = new double[6];
double radius, rad;

        s_face = new Tag();
        c_face = new Tag();

        theUfSession.Modl.AskFeatFaces(feat, out FeatFaces);
        theUfSession.Modl.AskListCount(FeatFaces, out
FacesCount);

UFModl.SymbThreadData thread = new
UFModl.SymbThreadData();

for (int i = 0; i < FacesCount; i++)
        {
          theUfSession.Modl.AskListItem(FeatFaces, i, out face);
          theUfSession.Modl.AskFaceData(face, out FaceType,
point, dir, box, out radius, out rad, out FaceNormDir);

if (FaceType == 22)
            {
              s_face = face;
//UI.GetUI().NXMessageBox.Show("Message",
NXMessageBox.DialogType.Information, "Planar face");
            }
if (FaceType == 16)
            {
```

```
                c_face = face;
//UI.GetUI().NXMessageBox.Show("Message",
NXMessageBox.DialogType.Information, "Cylindrical face");
            }
        }
double[] thread_direction = { 0.00, 0.00, 1.00 };
// направление резьбы
        thread.cyl_face = c_face;
        thread.start_face = s_face;
        thread.axis_direction = thread_direction;
        thread.rotation = 1;
        thread.num_starts = 1;
        thread.length = Convert.ToString(height);// длина резьбы
        thread.form = "Metric";// метрическая система
        thread.major_dia = Convert.ToString(2 * Radius);
//внешний диаметр резьбы
        thread.minor_dia = Convert.ToString((2 * Radius) -1);
//внутренний диаметр резьбы
        thread.tapped_dia = Convert.ToString(2 *
Radius);//диметр резьбы
        thread.pitch = Convert.ToString(step);//шаг резьбы
        thread.angle = "60";//угол резьбы

try
        {
          theUfSession.Modl.CreateSymbThread(ref thread, out
feat);
        }
catch (NXOpen.NXException ex)
        {
UI.GetUI().NXMessageBox.Show("Message",
NXMessageBox.DialogType.Error, ex.Message);
}
```

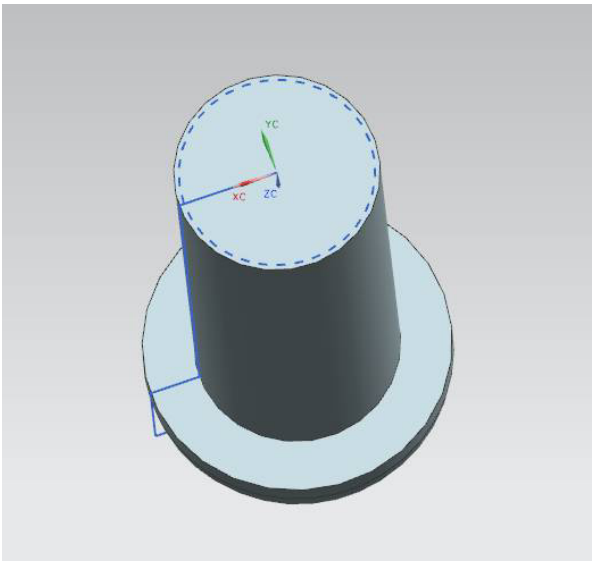


Рис. 3.6. Символическая резьба на цилидре

Осталось только добавить прорезь под шлицевую отвертку в «шляпке», для этого построим прямоугольник над шляпкой и продавим его вниз с параметром Negative.

```
UFCurve.Line segment8 = new UFCurve.Line();
UFCurve.Line segment9 = new UFCurve.Line();
UFCurve.Line segment10 = new UFCurve.Line();
UFCurve.Line segment11 = new UFCurve.Line();

segment8.start_point = new double[3];
segment8.start_point[0] = Radius * 1.5;
segment8.start_point[1] = Radius * 0.2;
segment8.start_point[2] = height * 1.25;
segment8.end_point = new double[3];
segment8.end_point[0] = Radius * 1.5;
segment8.end_point[1] = -(Radius * 0.2);
segment8.end_point[2] = height * 1.25;

segment9.start_point = new double[3];
segment9.start_point[0] = Radius * 1.5;
```

```
segment9.start_point[1] = -(Radius * 0.2);
segment9.start_point[2] = height * 1.25;
segment9.end_point = new double[3];
segment9.end_point[0] = -(Radius * 1.5);
segment9.end_point[1] = -(Radius * 0.2);
segment9.end_point[2] = height * 1.25;

segment10.start_point = new double[3];
segment10.start_point[0] = -(Radius * 1.5);
segment10.start_point[1] = -(Radius * 0.2);
segment10.start_point[2] = height * 1.25;
segment10.end_point = new double[3];
segment10.end_point[0] = -(Radius * 1.5);
segment10.end_point[1] = Radius * 0.2;
segment10.end_point[2] = height * 1.25;

segment11.start_point = new double[3];
segment11.start_point[0] = -(Radius * 1.5);
segment11.start_point[1] = Radius * 0.2;
segment11.start_point[2] = height * 1.25;
segment11.end_point = new double[3];
segment11.end_point[0] = Radius * 1.5;
segment11.end_point[1] = Radius * 0.2;
segment11.end_point[2] = height * 1.25;

Tag[] BoltArray2 = new Tag[5];
theUfSession.Curve.CreateLine(ref segment8, out BoltArray2[0]);
theUfSession.Curve.CreateLine(ref segment9, out BoltArray2[1]);
theUfSession.Curve.CreateLine(ref segment10, out
BoltArray2[2]);
theUfSession.Curve.CreateLine(ref segment11, out
BoltArray2[3]);

string glub = Convert.ToInt32(height * 0.125).ToString();
string taper_angle = "0.0";
string[] limit3 = { "0", glub };
double[] ref_pt3 = new double[3];
```

```
double[] direction3 = { 0.0, 0.0, -1.0 };

theUfSession.Modl.CreateExtruded(BoltArray2, taper_angle,
limit3, ref_pt3, direction3,FeatureSigns.Negative, out features2);
```

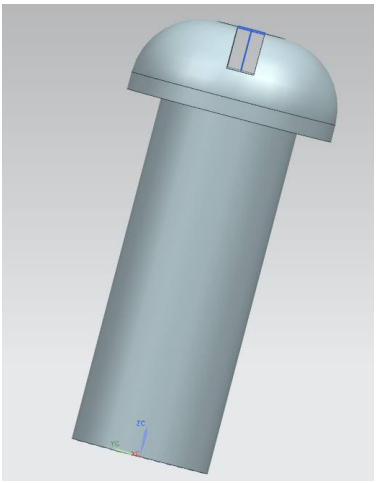
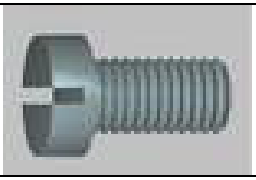
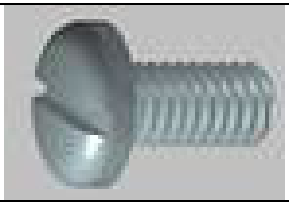
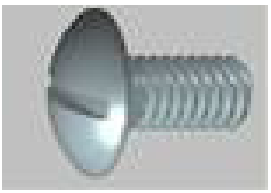


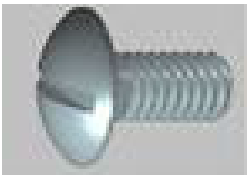
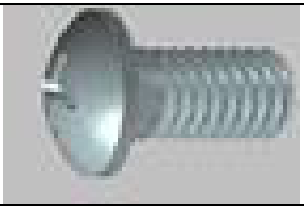
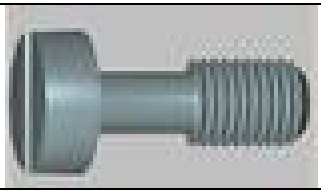
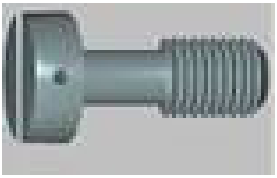
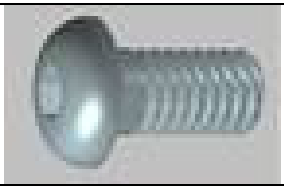
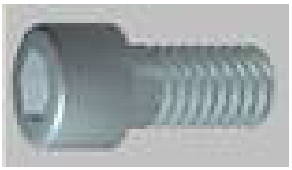
Рис. 3.7. Добавленная прорезь на «шляпке»

3.2. Практическая часть

3.2.1. Задание на лабораторную работу

Получить следующие объекты болтов.

Вариант, геометрическая фигура	
1.	6.
	
2.	7.
	

3.	8.
	
4.	9.
	
5.	10.
	

3.2.2. Содержание отчета по лабораторной работе

1. Название и цель работы.
2. Скриншоты с кратким описанием, соответствующие основным шагам выполненной работы.
3. Скриншоты, демонстрирующие работоспособность созданной библиотеки.
4. Листинг прграммы с комментариями.
5. Скриншоты 3D-модели, сформированной Вашей библиотекой.
6. Выводы.

4. ЛАБОРАТОРНАЯ РАБОТА № 4
СОЗДАНИЕ СБОРКИ КОНСТРУКЦИИ ИЗ ОТДЕЛЬНЫХ ДЕТАЛЕЙ

Цель работы: освоение методики размещения моделей деталей в абсолютной системе координат пространства сборочной модели, освоение операции «фаска».

4.1. Теоретическая часть

В данной лабораторной работе предлагается выполнить сборочную модель фильтра, представленную на рис. 4.1. Таким образом, деталь приобретает конечный вид.

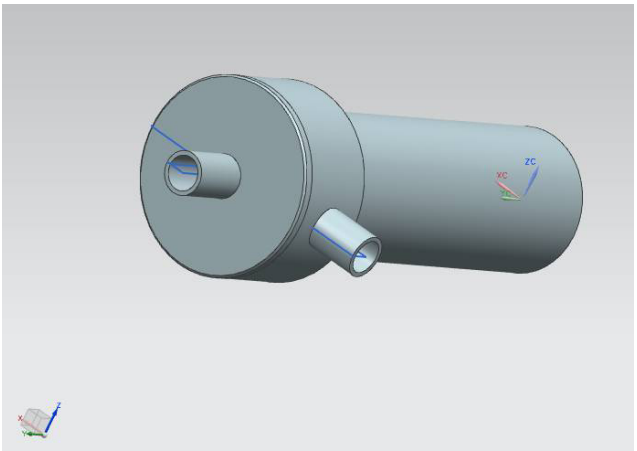


Рис. 4.1. Сборочная модель фильтра

Приведем подробное описание программной реализации построения сборочной 3D-модели.

Модель состоит из трех деталей (корпус, крышка, фильтрующий элемент), каждая из которых оформляется как отдельный документ. На завершающем этапе производится сборка всех деталей в одно изделие. Все детали на момент осуществления сборки, могут быть сохранены в файлы на диске или быть представленными в виде открытых проектов.

В начале программы создадим новую модель - корпус для фильтра, которая представлена на рис. 4.2.

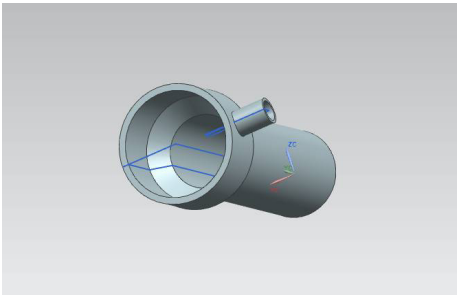


Рис. 4.2. Корпус для фильтра

```
Tag UFPart1;
string name1 = "Korpus";
int units1 = 1;
theUfSession.Part.New(name1, units1, out UFPart1);

double[] l1_endpt1 = { 0, 0, 0.00 };
double[] l1_endpt2 = { 32.5, 0, 0.00 };

double[] l2_endpt1 = { 32.5, 0, 0.00 };
double[] l2_endpt2 = { 32.5, 170, 0.00 };

double[] l3_endpt1 = { 32.5, 170, 0.00 };
double[] l3_endpt2 = { 45, 185, 0.00 };

double[] l4_endpt1 = { 45, 185, 0.00 };
double[] l4_endpt2 = { 45, 225, 0.00 };

double[] l5_endpt1 = { 45, 225, 0.00 };
double[] l5_endpt2 = { 0, 225, 0.00 };

double[] l6_endpt1 = { 0, 225, 0.00 };
double[] l6_endpt2 = { 0, 0, 0.00 };

UFCurve.Line line1 = new UFCurve.Line();
UFCurve.Line line2 = new UFCurve.Line();
```

```
UFCurve.Line line3 = new UFCurve.Line();
UFCurve.Line line4 = new UFCurve.Line();
UFCurve.Line line5 = new UFCurve.Line();
UFCurve.Line line6 = new UFCurve.Line();

line1.start_point = new double[3];
line1.start_point[0] = l1_endpt1[0];
line1.start_point[1] = l1_endpt1[1];
line1.start_point[2] = l1_endpt1[2];
line1.end_point = new double[3];
line1.end_point[0] = l1_endpt2[0];
line1.end_point[1] = l1_endpt2[1];
line1.end_point[2] = l1_endpt2[2];
line2.start_point = new double[3];
line2.start_point[0] = l2_endpt1[0];
line2.start_point[1] = l2_endpt1[1];
line2.start_point[2] = l2_endpt1[2];
line2.end_point = new double[3];
line2.end_point[0] = l2_endpt2[0];
line2.end_point[1] = l2_endpt2[1];
line2.end_point[2] = l2_endpt2[2];
line3.start_point = new double[3];
line3.start_point[0] = l3_endpt1[0];
line3.start_point[1] = l3_endpt1[1];
line3.start_point[2] = l3_endpt1[2];
line3.end_point = new double[3];
line3.end_point[0] = l3_endpt2[0];
line3.end_point[1] = l3_endpt2[1];
line3.end_point[2] = l3_endpt2[2];
line4.start_point = new double[3];
line4.start_point[0] = l4_endpt1[0];
line4.start_point[1] = l4_endpt1[1];
line4.start_point[2] = l4_endpt1[2];
line4.end_point = new double[3];
line4.end_point[0] = l4_endpt2[0];
line4.end_point[1] = l4_endpt2[1];
```

```
line4.end_point[2] = l4_endpt2[2];
line5.start_point = new double[3];
line5.start_point[0] = l5_endpt1[0];
line5.start_point[1] = l5_endpt1[1];
line5.start_point[2] = l5_endpt1[2];
line5.end_point = new double[3];
line5.end_point[0] = l5_endpt2[0];
line5.end_point[1] = l5_endpt2[1];
line5.end_point[2] = l5_endpt2[2];
line6.start_point = new double[3];
line6.start_point[0] = l6_endpt1[0];
line6.start_point[1] = l6_endpt1[1];
line6.start_point[2] = l6_endpt1[2];
line6.end_point = new double[3];
line6.end_point[0] = l6_endpt2[0];
line6.end_point[1] = l6_endpt2[1];
line6.end_point[2] = l6_endpt2[2];

Tag[] objarray1 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line1, out objarray1[0]);
theUfSession.Curve.CreateLine(ref  line2, out objarray1[1]);
theUfSession.Curve.CreateLine(ref  line3, out objarray1[2]);
theUfSession.Curve.CreateLine(ref  line4, out objarray1[3]);
theUfSession.Curve.CreateLine(ref  line5, out objarray1[4]);
theUfSession.Curve.CreateLine(ref  line6, out objarray1[5]);

double[] ref_pt1 = new double[3];
ref_pt1[0] = 0.00;
ref_pt1[1] = 0.00;
ref_pt1[2] = 0.00;
double[] direction1 = { 0.00, 1.00, 0.00 };
string[] limit1 = { "0", "360" };
Tag[] features1;

theUfSession.Modl.CreateRevolved(objarray1, limit1,
ref_pt1, direction1, FeatureSigns.Nullsign, out features1);
```

```
//////////////////////////////////////////

double[] l13_endpt1 = { -70, 205, 0.00 };
double[] l13_endpt2 = { -20, 205, 0.00 };

double[] l14_endpt1 = l13_endpt2;
double[] l14_endpt2 = { -20, 215, 0.00 };

double[] l15_endpt1 = l14_endpt2;
double[] l15_endpt2 = { -70, 215, 0.00 };

double[] l16_endpt1 = l15_endpt2;
double[] l16_endpt2 = { -70, 205, 0.00 };

UFCurve.Line line13 = new UFCurve.Line();
UFCurve.Line line14 = new UFCurve.Line();
UFCurve.Line line15 = new UFCurve.Line();
UFCurve.Line line16 = new UFCurve.Line();

line13.start_point = new double[3];
line13.start_point[0] = l13_endpt1[0];
line13.start_point[1] = l13_endpt1[1];
line13.start_point[2] = l13_endpt1[2];
line13.end_point = new double[3];
line13.end_point[0] = l13_endpt2[0];
line13.end_point[1] = l13_endpt2[1];
line13.end_point[2] = l13_endpt2[2];
line14.start_point = new double[3];
line14.start_point[0] = l14_endpt1[0];
line14.start_point[1] = l14_endpt1[1];
line14.start_point[2] = l14_endpt1[2];
line14.end_point = new double[3];
line14.end_point[0] = l14_endpt2[0];
line14.end_point[1] = l14_endpt2[1];
line14.end_point[2] = l14_endpt2[2];
line15.start_point = new double[3];
```

```
line15.start_point[0] = l15_endpt1[0];
line15.start_point[1] = l15_endpt1[1];
line15.start_point[2] = l15_endpt1[2];
line15.end_point = new double[3];
line15.end_point[0] = l15_endpt2[0];
line15.end_point[1] = l15_endpt2[1];
line15.end_point[2] = l15_endpt2[2];
line16.start_point = new double[3];
line16.start_point[0] = l16_endpt1[0];
line16.start_point[1] = l16_endpt1[1];
line16.start_point[2] = l16_endpt1[2];
line16.end_point = new double[3];
line16.end_point[0] = l16_endpt2[0];
line16.end_point[1] = l16_endpt2[1];
line16.end_point[2] = l16_endpt2[2];

Tag[] objarray3 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line13, out objarray3[0]);
theUfSession.Curve.CreateLine(ref  line14, out objarray3[1]);
theUfSession.Curve.CreateLine(ref  line15, out objarray3[2]);
theUfSession.Curve.CreateLine(ref  line16, out objarray3[3]);
double[] ref_pt2 = new double[3];
ref_pt2[0] = -50.00;
ref_pt2[1] = 205.00;
ref_pt2[2] = 0.00;
double[] direction2 = { 1.00, 0.00, 0.00 };
theUfSession.Modl.CreateRevolved(objarray3, limit1,
ref_pt2, direction2, FeatureSigns.Positive, out features1);

//////////////////////////////////////////
double[] l7_endpt1 = { 0, 5, 0.00 };
double[] l7_endpt2 = { 27.5, 5, 0.00 };

double[] l8_endpt1 = l7_endpt2;
double[] l8_endpt2 = { 27.5, 170, 0.00 };
```

```
double[] l9_endpt1 = l8_endpt2;
double[] l9_endpt2 = { 40, 185, 0.00 };

double[] l10_endpt1 = l9_endpt2;
double[] l10_endpt2 = { 40, 225, 0.00 };

double[] l11_endpt1 = l10_endpt2;
double[] l11_endpt2 = { 0, 225, 0.00 };

double[] l12_endpt1 = l11_endpt2;
double[] l12_endpt2 = { 0, 5, 0.00 };

UFCurve.Line line7 = new UFCurve.Line();
UFCurve.Line line8 = new UFCurve.Line();
UFCurve.Line line9 = new UFCurve.Line();
UFCurve.Line line10 = new UFCurve.Line();
UFCurve.Line line11 = new UFCurve.Line();
UFCurve.Line line12 = new UFCurve.Line();

line7.start_point = new double[3];
line7.start_point[0] = l7_endpt1[0];
line7.start_point[1] = l7_endpt1[1];
line7.start_point[2] = l7_endpt1[2];
line7.end_point = new double[3];
line7.end_point[0] = l7_endpt2[0];
line7.end_point[1] = l7_endpt2[1];
line7.end_point[2] = l7_endpt2[2];
line8.start_point = new double[3];
line8.start_point[0] = l8_endpt1[0];
line8.start_point[1] = l8_endpt1[1];
line8.start_point[2] = l8_endpt1[2];
line8.end_point = new double[3];
line8.end_point[0] = l8_endpt2[0];
line8.end_point[1] = l8_endpt2[1];
line8.end_point[2] = l8_endpt2[2];
line9.start_point = new double[3];
```

```
line9.start_point[0] = l9_endpt1[0];
line9.start_point[1] = l9_endpt1[1];
line9.start_point[2] = l9_endpt1[2];
line9.end_point = new double[3];
line9.end_point[0] = l9_endpt2[0];
line9.end_point[1] = l9_endpt2[1];
line9.end_point[2] = l9_endpt2[2];
line10.start_point = new double[3];
line10.start_point[0] = l10_endpt1[0];
line10.start_point[1] = l10_endpt1[1];
line10.start_point[2] = l10_endpt1[2];
line10.end_point = new double[3];
line10.end_point[0] = l10_endpt2[0];
line10.end_point[1] = l10_endpt2[1];
line10.end_point[2] = l10_endpt2[2];
line11.start_point = new double[3];
line11.start_point[0] = l11_endpt1[0];
line11.start_point[1] = l11_endpt1[1];
line11.start_point[2] = l11_endpt1[2];
line11.end_point = new double[3];
line11.end_point[0] = l11_endpt2[0];
line11.end_point[1] = l11_endpt2[1];
line11.end_point[2] = l11_endpt2[2];
line12.start_point = new double[3];
line12.start_point[0] = l12_endpt1[0];
line12.start_point[1] = l12_endpt1[1];
line12.start_point[2] = l12_endpt1[2];
line12.end_point = new double[3];
line12.end_point[0] = l12_endpt2[0];
line12.end_point[1] = l12_endpt2[1];
line12.end_point[2] = l12_endpt2[2];

Tag[] objarray2 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line7, out objarray2[0]);
theUfSession.Curve.CreateLine(ref  line8, out objarray2[1]);
theUfSession.Curve.CreateLine(ref  line9, out objarray2[2]);
```

```
theUfSession.Curve.CreateLine(ref  line10, out objarray2[3]);
theUfSession.Curve.CreateLine(ref  line11, out objarray2[4]);
theUfSession.Curve.CreateLine(ref  line12, out objarray2[5]);

theUfSession.Modl.CreateRevolved(objarray2, limit1, ref_pt1,
direction1, FeatureSigns.Negative, out features1);

double[] l17_endpt1 = { -70, 205, 0.00 };
double[] l17_endpt2 = { -20, 205, 0.00 };

double[] l18_endpt1 = l17_endpt2;
double[] l18_endpt2 = { -20, 213, 0.00 };

double[] l19_endpt1 = l18_endpt2;
double[] l19_endpt2 = { -70, 213, 0.00 };

double[] l20_endpt1 = l19_endpt2;
double[] l20_endpt2 = { -70, 205, 0.00 };

UFCurve.Line line17 = new UFCurve.Line();
UFCurve.Line line18 = new UFCurve.Line();
UFCurve.Line line19 = new UFCurve.Line();
UFCurve.Line line20 = new UFCurve.Line();

line17.start_point = new double[3];
line17.start_point[0] = l17_endpt1[0];
line17.start_point[1] = l17_endpt1[1];
line17.start_point[2] = l17_endpt1[2];
line17.end_point = new double[3];
line17.end_point[0] = l17_endpt2[0];
line17.end_point[1] = l17_endpt2[1];
line17.end_point[2] = l17_endpt2[2];
line18.start_point = new double[3];
line18.start_point[0] = l18_endpt1[0];
line18.start_point[1] = l18_endpt1[1];
line18.start_point[2] = l18_endpt1[2];
```

```
line18.end_point = new double[3];
line18.end_point[0] = l18_endpt2[0];
line18.end_point[1] = l18_endpt2[1];
line18.end_point[2] = l18_endpt2[2];
line19.start_point = new double[3];
line19.start_point[0] = l19_endpt1[0];
line19.start_point[1] = l19_endpt1[1];
line19.start_point[2] = l19_endpt1[2];
line19.end_point = new double[3];
line19.end_point[0] = l19_endpt2[0];
line19.end_point[1] = l19_endpt2[1];
line19.end_point[2] = l19_endpt2[2];
line20.start_point = new double[3];
line20.start_point[0] = l20_endpt1[0];
line20.start_point[1] = l20_endpt1[1];
line20.start_point[2] = l20_endpt1[2];
line20.end_point = new double[3];
line20.end_point[0] = l20_endpt2[0];
line20.end_point[1] = l20_endpt2[1];
line20.end_point[2] = l20_endpt2[2];

Tag[] objarray4 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line17, out objarray4[0]);
theUfSession.Curve.CreateLine(ref  line18, out objarray4[1]);
theUfSession.Curve.CreateLine(ref  line19, out objarray4[2]);
theUfSession.Curve.CreateLine(ref  line20, out objarray4[3]);

theUfSession.Modl.CreateRevolved(objarray4,  limit1,  ref_pt2,
direction2, FeatureSigns.Negative, out features1);
```

Далее приступим к созданию крышки для фильтра. Ее общий вид представлен на рис. 4.3-4.4.

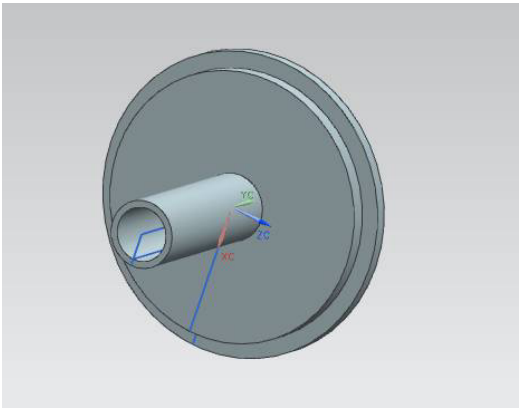


Рис. 4.3. Вид 1 крышки для фильтра

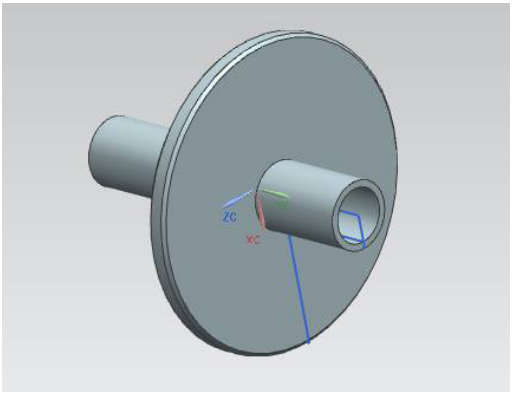


Рис. 4.4. Вид 2 крышки для фильтра

```
Tag UFPart1;
string name1 = "Krishka";
int units1 = 1;
theUfSession.Part.New(name1, units1, out UFPart1);

double[] l1_endpt1 = { 0, 0, 0.00 };
double[] l1_endpt2 = { 40, 0, 0.00 };

double[] l2_endpt1 = l1_endpt2;
double[] l2_endpt2 = { 40, 5, 0.00 };
```

```
double[] l3_endpt1 = l2_endpt2;
double[] l3_endpt2 = { 45, 5, 0.00 };

double[] l4_endpt1 = l3_endpt2;
double[] l4_endpt2 = { 45, 10, 0.00 };

double[] l5_endpt1 = l4_endpt2;
double[] l5_endpt2 = { 0, 10, 0.00 };

double[] l6_endpt1 = l5_endpt2;
double[] l6_endpt2 = { 0, 0, 0.00 };

UFCurve.Line line1 = new UFCurve.Line();
UFCurve.Line line2 = new UFCurve.Line();
UFCurve.Line line3 = new UFCurve.Line();
UFCurve.Line line4 = new UFCurve.Line();
UFCurve.Line line5 = new UFCurve.Line();
UFCurve.Line line6 = new UFCurve.Line();

line1.start_point = new double[3];
line1.start_point[0] = l1_endpt1[0];
line1.start_point[1] = l1_endpt1[1];
line1.start_point[2] = l1_endpt1[2];
line1.end_point = new double[3];
line1.end_point[0] = l1_endpt2[0];
line1.end_point[1] = l1_endpt2[1];
line1.end_point[2] = l1_endpt2[2];
line2.start_point = new double[3];
line2.start_point[0] = l2_endpt1[0];
line2.start_point[1] = l2_endpt1[1];
line2.start_point[2] = l2_endpt1[2];
line2.end_point = new double[3];
line2.end_point[0] = l2_endpt2[0];
line2.end_point[1] = l2_endpt2[1];
line2.end_point[2] = l2_endpt2[2];
line3.start_point = new double[3];
```

```
line3.start_point[0] = l3_endpt1[0];
line3.start_point[1] = l3_endpt1[1];
line3.start_point[2] = l3_endpt1[2];
line3.end_point = new double[3];
line3.end_point[0] = l3_endpt2[0];
line3.end_point[1] = l3_endpt2[1];
line3.end_point[2] = l3_endpt2[2];
line4.start_point = new double[3];
line4.start_point[0] = l4_endpt1[0];
line4.start_point[1] = l4_endpt1[1];
line4.start_point[2] = l4_endpt1[2];
line4.end_point = new double[3];
line4.end_point[0] = l4_endpt2[0];
line4.end_point[1] = l4_endpt2[1];
line4.end_point[2] = l4_endpt2[2];
line5.start_point = new double[3];
line5.start_point[0] = l5_endpt1[0];
line5.start_point[1] = l5_endpt1[1];
line5.start_point[2] = l5_endpt1[2];
line5.end_point = new double[3];
line5.end_point[0] = l5_endpt2[0];
line5.end_point[1] = l5_endpt2[1];
line5.end_point[2] = l5_endpt2[2];
line6.start_point = new double[3];
line6.start_point[0] = l6_endpt1[0];
line6.start_point[1] = l6_endpt1[1];
line6.start_point[2] = l6_endpt1[2];
line6.end_point = new double[3];
line6.end_point[0] = l6_endpt2[0];
line6.end_point[1] = l6_endpt2[1];
line6.end_point[2] = l6_endpt2[2];

Tag[] objarray1 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line1, out objarray1[0]);
theUfSession.Curve.CreateLine(ref  line2, out objarray1[1]);
theUfSession.Curve.CreateLine(ref  line3, out objarray1[2]);
```

```
theUfSession.Curve.CreateLine(ref  line4, out objarray1[3]);
theUfSession.Curve.CreateLine(ref  line5, out objarray1[4]);
theUfSession.Curve.CreateLine(ref  line6, out objarray1[5]);

double[] ref_pt1 = new double[3];
ref_pt1[0] = 0.00;
ref_pt1[1] = 0.00;
ref_pt1[2] = 0.00;
double[] direction1 = { 0.00, 1.00, 0.00 };
string[] limit1 = { "0", "360" };
Tag[] features1;

theUfSession.Modl.CreateRevolved(objarray1, limit1,
ref_pt1, direction1, FeatureSigns.Nullsign, out features1);

//////////////////////
// Создание фаски
Tag feat = features1[0];
Tag cyl_tag, obj_id_camf, blend1;
Tag[] Edge_array_cyl, list1, list2;

//из массива тэгов features1 выбирается нулевой тэг, который
//записывается в переменную feat, теперь в переменной feat
//находится тэг нулевого тела детали
int ecount;
//блок, который анализирует ребра рассматриваемой детали
theUfSession.Modl.AskFeatBody(feat, out cyl_tag);
theUfSession.Modl.AskBodyEdges(cyl_tag, out Edge_array_cyl);
theUfSession.Modl.AskListCount(Edge_array_cyl, out ecount);

// Первый объект в дальнейшем будет использован для
//сохранения в него ребер, на которых будут выполнены
//фаски, второй по аналогии используется для скругления
ArrayList arr_list1 = new ArrayList();
ArrayList arr_list2 = new ArrayList();
```

```
//цикл повторяется количество раз, равное числу ребер
//рассматриваемого тела
for (int ii = 0; ii < ecount; ii++)
{
   Tag edge;
   theUfSession.Modl.AskListItem(Edge_array_cyl, ii, out edge);
//выбирает из массива ребер каждое отдельное ребро с
//порядковым номером в массиве ii
   if ((ii == 2  ))
  {
      arr_list1.Add(edge); //ребро добавляется в arr_list1 (то есть
//на нем будет создана фаска
   }
}
//конвертирование типов данных
list1 = (Tag[])arr_list1.ToArray(typeof(Tag));
list2 = (Tag[])arr_list2.ToArray(typeof(Tag));

//параметры, задающие операцию "фаска"
string offset1 = "1";
string offset2 = "1";
string ang = "45";

theUfSession.Modl.CreateChamfer(3, offset1, offset2, ang,
list1, out obj_id_camf); //операция "фаска"

double[] l13_endpt1 = { 0, -55, 0.00 };
double[] l13_endpt2 = { 10, -55, 0.00 };

double[] l14_endpt1 = l13_endpt2;
double[] l14_endpt2 = { 10, 40, 0.00 };

double[] l15_endpt1 = l14_endpt2;
double[] l15_endpt2 = { 0, 40, 0.00 };

double[] l16_endpt1 = l15_endpt2;
double[] l16_endpt2 = { 0, -55, 0.00 };
```

```
UFCurve.Line line13 = new UFCurve.Line();
UFCurve.Line line14 = new UFCurve.Line();
UFCurve.Line line15 = new UFCurve.Line();
UFCurve.Line line16 = new UFCurve.Line();

line13.start_point = new double[3];
line13.start_point[0] = l13_endpt1[0];
line13.start_point[1] = l13_endpt1[1];
line13.start_point[2] = l13_endpt1[2];
line13.end_point = new double[3];
line13.end_point[0] = l13_endpt2[0];
line13.end_point[1] = l13_endpt2[1];
line13.end_point[2] = l13_endpt2[2];
line14.start_point = new double[3];
line14.start_point[0] = l14_endpt1[0];
line14.start_point[1] = l14_endpt1[1];
line14.start_point[2] = l14_endpt1[2];
line14.end_point = new double[3];
line14.end_point[0] = l14_endpt2[0];
line14.end_point[1] = l14_endpt2[1];
line14.end_point[2] = l14_endpt2[2];
line15.start_point = new double[3];
line15.start_point[0] = l15_endpt1[0];
line15.start_point[1] = l15_endpt1[1];
line15.start_point[2] = l15_endpt1[2];
line15.end_point = new double[3];
line15.end_point[0] = l15_endpt2[0];
line15.end_point[1] = l15_endpt2[1];
line15.end_point[2] = l15_endpt2[2];
line16.start_point = new double[3];
line16.start_point[0] = l16_endpt1[0];
line16.start_point[1] = l16_endpt1[1];
line16.start_point[2] = l16_endpt1[2];
line16.end_point = new double[3];
line16.end_point[0] = l16_endpt2[0];
line16.end_point[1] = l16_endpt2[1];
```

```
line16.end_point[2] = l16_endpt2[2];

Tag[] objarray3 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line13, out objarray3[0]);
theUfSession.Curve.CreateLine(ref  line14, out objarray3[1]);
theUfSession.Curve.CreateLine(ref  line15, out objarray3[2]);
theUfSession.Curve.CreateLine(ref  line16, out objarray3[3]);

theUfSession.Modl.CreateRevolved(objarray3, limit1,
ref_pt1, direction1, FeatureSigns.Positive, out features1);
//////////////////////////////////////////
double[] l131_endpt1 = { 0, -55, 0.00 };
double[] l131_endpt2 = { 8, -55, 0.00 };

double[] l141_endpt1 = l131_endpt2;
double[] l141_endpt2 = { 8, 40, 0.00 };

double[] l151_endpt1 = l141_endpt2;
double[] l151_endpt2 = { 0, 40, 0.00 };

double[] l161_endpt1 = l151_endpt2;
double[] l161_endpt2 = { 0, -55, 0.00 };

UFCurve.Line line131 = new UFCurve.Line();
UFCurve.Line line141 = new UFCurve.Line();
UFCurve.Line line151 = new UFCurve.Line();
UFCurve.Line line161 = new UFCurve.Line();

line131.start_point = new double[3];
line131.start_point[0] = l131_endpt1[0];
line131.start_point[1] = l131_endpt1[1];
line131.start_point[2] = l131_endpt1[2];
line131.end_point = new double[3];
line131.end_point[0] = l131_endpt2[0];
line131.end_point[1] = l131_endpt2[1];
line131.end_point[2] = l131_endpt2[2];
```

```
line141.start_point = new double[3];
line141.start_point[0] = l141_endpt1[0];
line141.start_point[1] = l141_endpt1[1];
line141.start_point[2] = l141_endpt1[2];
line141.end_point = new double[3];
line141.end_point[0] = l141_endpt2[0];
line141.end_point[1] = l141_endpt2[1];
line141.end_point[2] = l141_endpt2[2];
line151.start_point = new double[3];
line151.start_point[0] = l151_endpt1[0];
line151.start_point[1] = l151_endpt1[1];
line151.start_point[2] = l151_endpt1[2];
line151.end_point = new double[3];
line151.end_point[0] = l151_endpt2[0];
line151.end_point[1] = l151_endpt2[1];
line151.end_point[2] = l151_endpt2[2];
line161.start_point = new double[3];
line161.start_point[0] = l161_endpt1[0];
line161.start_point[1] = l161_endpt1[1];
line161.start_point[2] = l161_endpt1[2];
line161.end_point = new double[3];
line161.end_point[0] = l161_endpt2[0];
line161.end_point[1] = l161_endpt2[1];
line161.end_point[2] = l161_endpt2[2];

Tag[] objarray4 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line131, out objarray4[0]);
theUfSession.Curve.CreateLine(ref  line141, out objarray4[1]);
theUfSession.Curve.CreateLine(ref  line151, out objarray4[2]);
theUfSession.Curve.CreateLine(ref  line161, out objarray4[3]);
theUfSession.Modl.CreateRevolved(objarray4, limit1, ref_pt1,
direction1, FeatureSigns.Negative, out features1);
```

Далее переходим к созданию последней части – фильтра. Общий вид представлен на рис. 4.5.

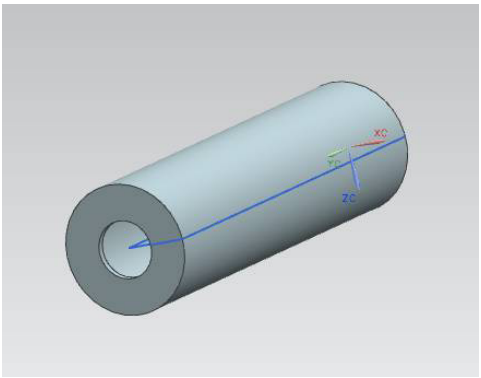


Рис. 4.5. Фильтр

```
Tag UFPart1;
string name1 = "cylindr";
int units1 = 1;
theUfSession.Part.New(name1, units1, out UFPart1);

double[] l1_endpt1 = { 10, 0, 0.00 };
double[] l1_endpt2 = { 22.5, 0, 0.00 };

double[] l2_endpt1 = l1_endpt2;
double[] l2_endpt2 = { 22.5, 160, 0.00 };

double[] l3_endpt1 = l2_endpt2;
double[] l3_endpt2 = { 10, 160, 0.00 };

double[] l4_endpt1 = l3_endpt2;
double[] l4_endpt2 = { 10, 0, 0.00 };

UFCurve.Line line1 = new UFCurve.Line();
UFCurve.Line line2 = new UFCurve.Line();
UFCurve.Line line3 = new UFCurve.Line();
UFCurve.Line line4 = new UFCurve.Line();
line1.start_point = new double[3];
line1.start_point[0] = l1_endpt1[0];
```

```
line1.start_point[1] = l1_endpt1[1];
line1.start_point[2] = l1_endpt1[2];
line1.end_point = new double[3];
line1.end_point[0] = l1_endpt2[0];
line1.end_point[1] = l1_endpt2[1];
line1.end_point[2] = l1_endpt2[2];
line2.start_point = new double[3];
line2.start_point[0] = l2_endpt1[0];
line2.start_point[1] = l2_endpt1[1];
line2.start_point[2] = l2_endpt1[2];
line2.end_point = new double[3];
line2.end_point[0] = l2_endpt2[0];
line2.end_point[1] = l2_endpt2[1];
line2.end_point[2] = l2_endpt2[2];
line3.start_point = new double[3];
line3.start_point[0] = l3_endpt1[0];
line3.start_point[1] = l3_endpt1[1];
line3.start_point[2] = l3_endpt1[2];
line3.end_point = new double[3];
line3.end_point[0] = l3_endpt2[0];
line3.end_point[1] = l3_endpt2[1];
line3.end_point[2] = l3_endpt2[2];
line4.start_point = new double[3];
line4.start_point[0] = l4_endpt1[0];
line4.start_point[1] = l4_endpt1[1];
line4.start_point[2] = l4_endpt1[2];
line4.end_point = new double[3];
line4.end_point[0] = l4_endpt2[0];
line4.end_point[1] = l4_endpt2[1];
line4.end_point[2] = l4_endpt2[2];

Tag[] objarray1 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line1, out objarray1[0]);
theUfSession.Curve.CreateLine(ref  line2, out objarray1[1]);
theUfSession.Curve.CreateLine(ref  line3, out objarray1[2]);
theUfSession.Curve.CreateLine(ref  line4, out objarray1[3]);
```

```
double[] ref_pt1 = new double[3];
ref_pt1[0] = 0.00;
ref_pt1[1] = 0.00;
ref_pt1[2] = 0.00;
double[] direction1 = { 0.00, 1.00, 0.00 };
string[] limit1 = { "0", "360" };
Tag[] features1;

theUfSession.Modl.CreateRevolved(objarray1, limit1,
ref_pt1, direction1, FeatureSigns.Nullsign, out features1);
//////////////////////////////////////////////////
double[] l11_endpt1 = { 0, 2.5, 0.00 };
double[] l11_endpt2 = { 20, 2.5, 0.00 };
double[] l21_endpt1 = l11_endpt2;
double[] l21_endpt2 = { 20, 157.5, 0.00 };
double[] l31_endpt1 = l21_endpt2;
double[] l31_endpt2 = { 0, 157.5, 0.00 };
double[] l41_endpt1 = l31_endpt2;
double[] l41_endpt2 = { 0, 2.5, 0.00 };

UFCurve.Line line11 = new UFCurve.Line();
UFCurve.Line line21 = new UFCurve.Line();
UFCurve.Line line31 = new UFCurve.Line();
UFCurve.Line line41 = new UFCurve.Line();

line11.start_point = new double[3];
line11.start_point[0] = l11_endpt1[0];
line11.start_point[1] = l11_endpt1[1];
line11.start_point[2] = l11_endpt1[2];
line11.end_point = new double[3];
line11.end_point[0] = l11_endpt2[0];
line11.end_point[1] = l11_endpt2[1];
line11.end_point[2] = l11_endpt2[2];
line21.start_point = new double[3];
line21.start_point[0] = l21_endpt1[0];
```

```
line21.start_point[1] = l21_endpt1[1];
line21.start_point[2] = l21_endpt1[2];
line21.end_point = new double[3];
line21.end_point[0] = l21_endpt2[0];
line21.end_point[1] = l21_endpt2[1];
line21.end_point[2] = l21_endpt2[2];
line31.start_point = new double[3];
line31.start_point[0] = l31_endpt1[0];
line31.start_point[1] = l31_endpt1[1];
line31.start_point[2] = l31_endpt1[2];
line31.end_point = new double[3];
line31.end_point[0] = l31_endpt2[0];
line31.end_point[1] = l31_endpt2[1];
line31.end_point[2] = l31_endpt2[2];
line41.start_point = new double[3];
line41.start_point[0] = l41_endpt1[0];
line41.start_point[1] = l41_endpt1[1];
line41.start_point[2] = l41_endpt1[2];
line41.end_point = new double[3];
line41.end_point[0] = l41_endpt2[0];
line41.end_point[1] = l41_endpt2[1];
line41.end_point[2] = l41_endpt2[2];

Tag[] objarray2 = new Tag[7];
theUfSession.Curve.CreateLine(ref  line11, out objarray2[0]);
theUfSession.Curve.CreateLine(ref  line21, out objarray2[1]);
theUfSession.Curve.CreateLine(ref  line31, out objarray2[2]);
theUfSession.Curve.CreateLine(ref  line41, out objarray2[3]);
theUfSession.Modl.CreateRevolved(objarray2, limit1, ref_pt1,
direction1, FeatureSigns.Negative, out features1);
```

Для создания сборки необходимо создать новый проект и разместить в нем код, указанный ниже.

```
Tag UFPart;
string name1 = "Filter";
int units = 1;
```

```
theUfSession.Part.New(name1, units, out UFPart);
/*В переменную parent_part, записывается результат
возвращаемый функцией AskDisplayPart. В свою очередь
данная функция возвращает tag текущей модели. Для не
сборочной модели это рабочая модель. Если в настоящее
время нет рабочей модели, то возвращается NULL_TAG. */
Tag parent_part = theUfSession.Part.AskDisplayPart();
//переменные error_status, в которые будет записываться
//статус загрузки моделей
UFPart.LoadStatus error_status, error_status2, error_status3;
//объявление переменных в которые происходит запись tag-а
//нового объекта модели
Tag instance, instance1, instance2;

/*Объявление и запись значений в переменные. Переменные
origin содержат позиции каждой из моделей в родительской
сборочной модели.
Переменные matrix определяют ориентацию каждой из
моделей в системе координат родительской сборочной
модели.*/

double[] origin1 = { 0, 220, 0 };
double[] matrix1 = { 1, 0, 0, 0, 1, 0 };
double[] origin2 = { 0, 0, 0 };
double[] matrix2 = { 1, 0, 0, 0, 1, 0 };
double[] origin3 = { 0, 7.5, 0 };
double[] matrix3 = { 1, 0, 0, 0, 1, 0 };

//поочередное добавление моделей деталей фильтра, которые
//входят в сборочную модель конструкции

/*Структура параметров функции AddPartToAssembly:
1) parent_part – tag модели для добавления деталей;
2) part – имя добавляемой модели;
3) refset_name – наименование множества частей модели для
добавления;
```

```
4) instance_name - Name of new instance;
5) origin [ 3 ] – позиция в родительской модели;
6) csys_matrix [ 6 ] – ориентация в родительской модели;
7) layer – слой (0) – текущий слой;
8) instance – tag новой детали в сборке;
9) error_status – статус ошибки добавления.
*/

theUfSession.Assem.AddPartToAssembly(parent_part, "Krishka",
null, null, origin1, matrix1, 0, out instance, out error_status);

theUfSession.Assem.AddPartToAssembly(parent_part, "Korpus",
null, null, origin2, matrix2, 0, out instance1, out error_status2);

theUfSession.Assem.AddPartToAssembly(parent_part, "cylindr",
null, null, origin3, matrix3, 0, out instance2, out error_status3);
```

4.2. Практическая часть

4.2.1. Задание на лабораторную работу

Задание выдается преподавателем по вариантам.

Пример вариантов.

Пример 1. В лабораторной работе предлагается выполнить сборочную модель штампа для жидкой штамповки, представленную на рис. 4.6.

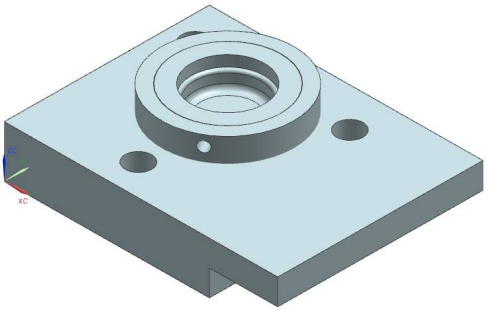


Рис. 4.6. Сборочная модель штампа для жидкой штамповки

Пример 2. В лабораторной работе предлагается выполнить сборочную модель пневмогидравлического клапана, представленную на рис. 4.7.

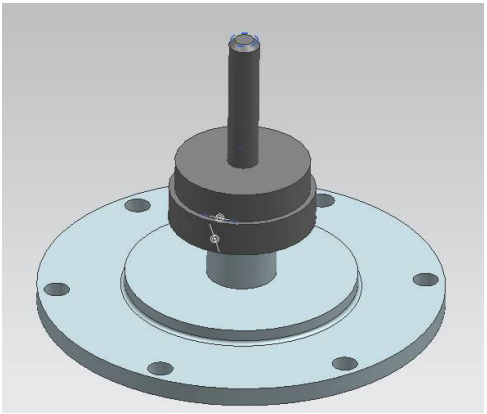


Рис. 4.7. Сборочная модель пневмогидравлического клапана

4.2.2. Содержание отчета по лабораторной работе

1. Название и цель работы.
2. Скриншоты с кратким описанием, соответствующие основным шагам выполненной работы.
3. Скриншоты, демонстрирующие работоспособность созданной библиотеки.
4. Листинг прграммы с комментариями.
5. Скриншоты 3D-модели, сформированной Вашей библиотекой.
6. Выводы.

ЗАКЛЮЧЕНИЕ

Внедрение технологий 3D-моделирования в образовательный процесс способствует более эффективному формированию уровня профессиональной подготовки для успешной реализации в трудовой деятельности. Востребованность выпускника определяется уровнем его подготовки.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ведмидь, П. Программирование обработки в NX CAM [Текст] /П. Ведмидь, А. Сулинов. - М.:ДМК Пресс, 2014. – 304 с.
2. Ведмидь, П. Основы NX CAM [Текст] / М.:ДМК Пресс. 2012 – 216 с.
3. Гончаров, П. NX для конструктора-машиностроителя [Текст] / П.С. Гончаров, М.Ю. Ельцов, С.Б. Коршиков, И.В. Лаптев, В.А.. Осиюк. - М.:ДМК Пресс.- 2016 – 500 с.
4. Siemens PLM Software [Электронный ресурс] : Режим доступа : World Wide Web. URL : http://www.plm.automation.siemens.com/ru_ru/
5. Гончаров, П. С. NX Advanced Simulation. Инженерный анализ [Текст] / П.С. Гончаров, И.А. Артамонов, Т.Ф. Халитов. — М.: ДМК Пресс, 2012. — 504 с.
6. Данилов, Ю. Практическое использование NX. . [Текст] / Ю. Данилов, И. Артамонов. — М.: ДМК Пресс, 2011. — 332 с.
7. Ельцов, М. Ю. Проектирование в NX под управлением Teamcenter [Текст] / М. Ю. Ельцов, А. А. Козлов, А.В. Седойкин— М.: ДМК Пресс, 2013. — 752 с.
8. Краснов, М. Unigraphics для профессионалов [Текст] / М. Краснов, Ю. Чигишев. — ЛОРИ, 2003. — 320 с.
9. Почекуев, Е. Н. Проектирование последовательных штампов для листовой штамповки в системе NX [Текст] / Е.Н. Почекуев, П. А. Путеев, П. Н. Шенбергер. — М.: ДМК Пресс, 2012. — 336 с.
10. Тороп, Д.Н. Teamcenter. Начало работы [Текст] / Д.Н. Тороп, В.В. Терликов— М.: ДМК Пресс, 2011. — С. 280.
11. Чижов, М.И. САПР технологического оснащения [Текст]: учеб. пособ. / М.И. Чижов, А.Ю. Мануковский. - Воронеж: ГОУВПО «Воронежский государственный технический университет», 2010. - 83 с.

ОГЛАВЛЕНИЕ

Учебное издание

Барабанов Владимир Федорович
Нужный Александр Михайлович
Сафронов Виталий Владимирович
Гребенникова Наталия Ивановна

ПАРАМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ NX API

В авторской редакции

Подписано к изданию 20.10.2017.
Объем данных 1,9 Мб.

ФГБОУ ВО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14