ГОУВПО «Воронежский государственный технический университет»

Кафедра компьютерных интеллектуальных технологий проектирования

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам по дисциплине «Автоматизация конструкторского и технологического проектирования» для студентов специальности 230104 «Системы автоматизированного проектирования» очной формы обучения

Воронеж 2011

Составители: д-р техн. наук М.И. Чижов,
к.т.н., доцент М.В. Паринов,
ст. преп. В.А. Рыжков,
И.А. Еремин

УДК 681.3

Методические указания к лабораторным работам по дисциплине «Автоматизация конструкторского и технологического проектирования» для студентов специальности 230104 «Системы автоматизированного проектирования» очной формы обучения / ГОУВПО «Воронежский государственный технический университет»; сост. М.И. Чижов, М.В. Паринов, В.А. Рыжков, И.А. Еремин. Воронеж, 2011. 43с.

Издание содержит указания по организации лабораторных работ студентов в соответствии с основным содержанием первого раздела курса «Автоматизация конструкторского и технологического проектирования». В методических указаниях приведены теоретические и практические разделы по выполнению лабораторных работ курса.

Предназначены для студентов четвертого курса.

Ил. 10.
Рецензент:

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Печатается по решению редакционно-издательского совета Воронежского государственного технического университета

Введение

В методических указания: ассмотрены основы работы с функциями построения 3D моделей и сборок NXOpen/API. Структура лабораторных работ подразумевает создание 3D сборки пневмоцилиндра средствами API функций NX 7.5. Сборочная модель формируется по принципу “сборка снизу”: сначала формируются отдельные детали, после чего осуществляется их сборка.

Рассматриваемый цилиндр состоит из 3 деталей: корпуса, штока и крышки (рис. 1).

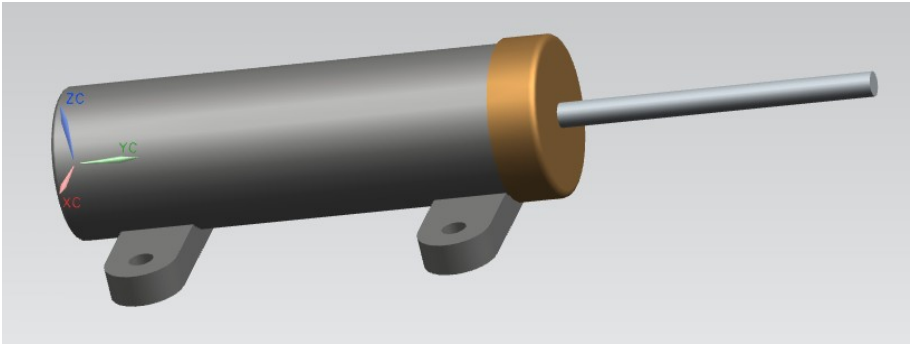

Рисунок 1 - Сборочная 3D модель цилиндра

Шток (рис. 2) выполнен с помощью операции “Вращение”.

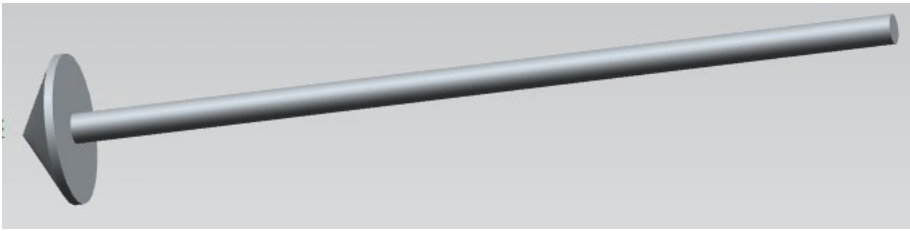

Рисунок 2 – Шток

Крышка (рис. 3) выполнена с помощью операции

“Вращение”. Скругление торцевого ребра осуществляется отдельно соответствующей операцией. Фаски выполняются в единой операции “Фаска”; в качестве входных данных указываются два ребра, на которых требуется выполнение фасок.

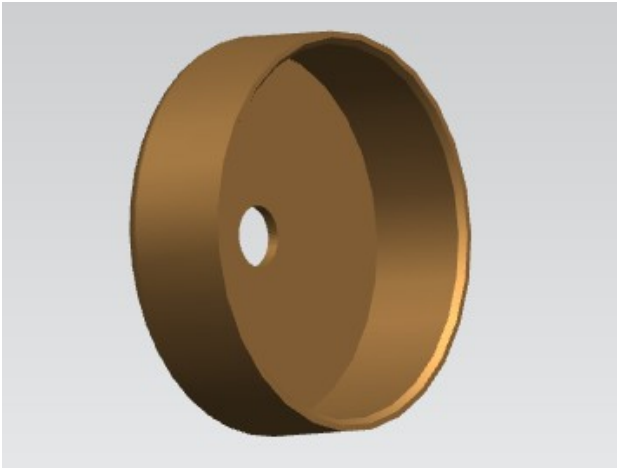

Рисунок 3 - Крышка

Наибольшую сложность представляет корпус цилиндра (рис. 4). Авторами рекомендуется следующая последовательность формирования 3D модели:

1. Создание цилиндрического тела со сквозным отверстием, диаметр которого соответствует диаметру штуцера, устанавливаемого в торец цилиндра, выполняется операцией “Вращение”.
2. Формирование крепежных лап цилиндра с помощью операции “Выдавливание”; с целью получения единого тела необходимо в опциях операции указать булево свойство “Объединение”.
3. Создание внутренней полости цилиндра (в которой перемещается шток) с помощью операции “Вращение”; операция отличается от аналогичной, рассмотренной в п. 1 тем, что указывается булево свойство “Вычитание”.

4. С помощью специализированной операции выполнить скругление.

5. С помощью специализированной операции выполнить фаску.

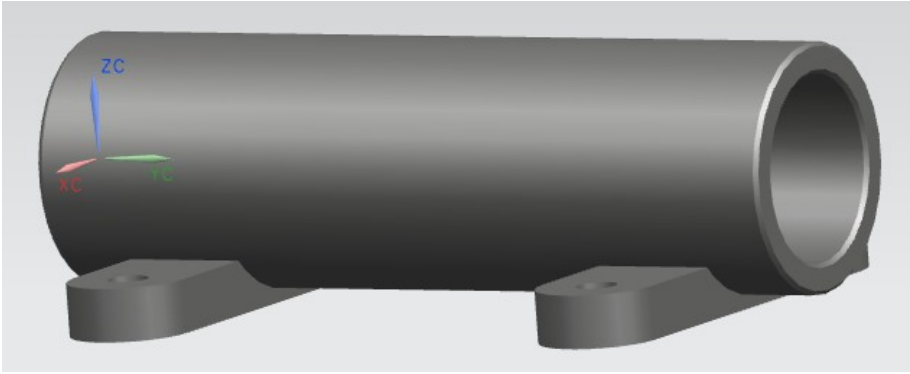

Рисунок 4 - Корпус

В качестве инструмента разработчика в методических указаниях используется C# в среде Microsoft Visual Studio 2008.

Лабораторная работа №1
Создание простейшей прикладной библиотеки для NX 7.5 на языке C# в среде MS Visual Studio 2008

Цель работы: интегрировать мастер разработки прикладных подпрограмм для NX 7.5 в Visual Studio, научиться создавать простейшие библиотеки и запускать их на выполнение в NX.

1. Теоретическая часть

Для разработки прикладных пользовательских программ для NX 7.5 SIEMENS рекомендует использовать среду разработку Microsoft Visual Studio 2008. Допускается (но при этом не гарантируется работоспособность) использование более поздних версий.

В составе установленного NX 7.5 присутствуют файлы мастеров для Visual Studio 2008, которые позволяют создавать под него прикладные программы. Поддерживаются 3 языка: C++, C#, Visual Basic. Для получения доступа к мастерам Visual Studio необходимо выполнить следующую инструкцию:

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 рабочий каталог Visual Studio. Путь к каталогу по умолчанию: C:\Program Files\Microsoft Visual Studio 9.0. В процессе копирования необходимо подтвердить желание при копировании заменить папки и файлы с совпадающими именами. Внимание: в процессе копирования Visual Studio должна быть выгружена из памяти.

При запуске Visual Studio после выполнения представленной выше инструкции в диалоге создания проектов будут включены соответствующие мастера (3 штуки: по одному для каждого языка). Рассмотрим работу с мастером под C# .

В качестве типа проектов выберете C# (рис. 1.1). В открывшемся справа меню выберете NX7 Open C# Wizard. Далее задайте имя и расположение проекта, затем нажмите кнопку OK.

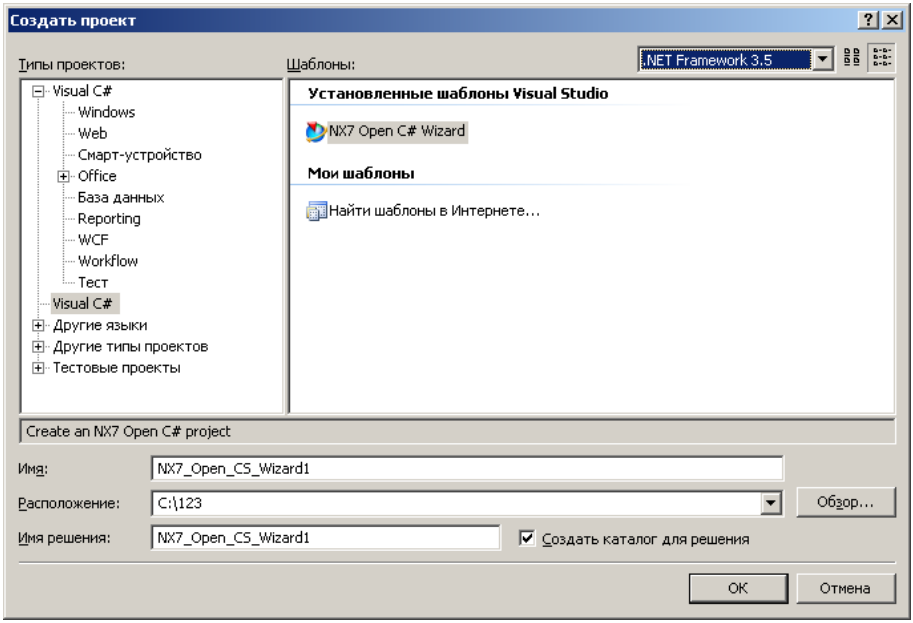

Рисунок 1.1 - Мастер создания проекта

В открывшемся окне мастера нажмите кнопку Next. В следующем диалоговом окне (рис. 1.2) предлагается выбрать 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 “Use APIs” установите галочки напротив двух предлагаемых вариантов. Нажмите кнопку Next.

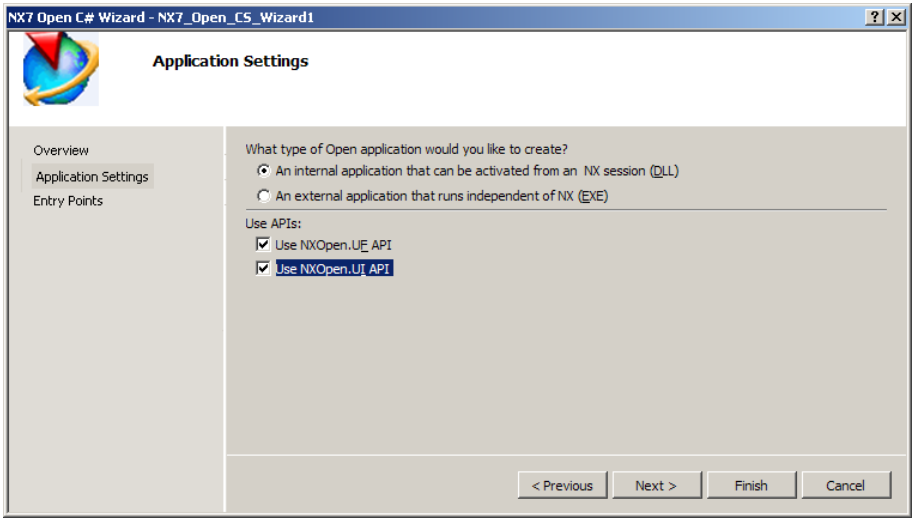

Рисунок 1.2 - Настройка типа приложения и используемых API

В открывшемся диалоговом окне (рис. 1.3) необходимо выбрать опции загрузки и выгрузки разрабатываемой библиотеки. Выберем в качестве опции загрузки “Explicitly (Main)”, что соответствует загрузке приложения через вызываемый пользователем диалог. Также доступны следующие варианты: при загрузке NX и согласно установленному событию. В качестве опции выгрузки выберем “Automatically, when the NX session terminates”. В данном случае библиотека будет выгружена вместе с NX. Другие доступные варианты позволяют выгружать библиотеку по завершению ее работы или через специальный диалог.

По завершению настройки нажмите кнопку Finish, после чего среда разработки автоматически переместит вас в окно написания программного кода.

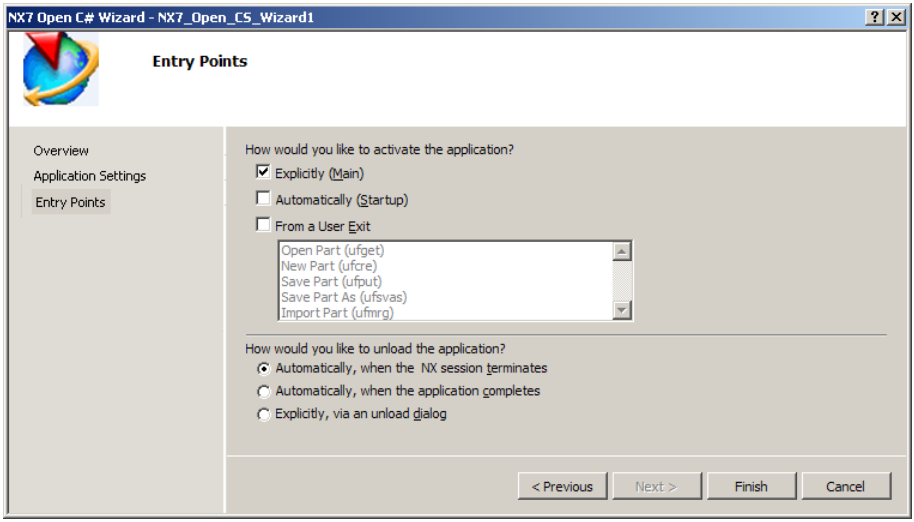

Рисунок 1.3 - Настройка загрузки и выгрузки библиотеки

Найдите следующую процедуру:

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

Вместо строки комментария “//TODO: Add your application code here” введите программный код разрабатываемого приложения.

В качестве первой программы предлагается вывести типовое сообщение NX. Для этого введите следующую строку:

```
UI.GetUI().NXMessageBox.Show("Message",
NXMessageBox.DialogType.Information, "Изучаем NXOpen/API");
```

Здесь “Message” – заголовок сообщения, NXMessageBox.DialogType.Information – тип окна сообщения (в данном случае информационный), “Изучаем NXOpen/API” – текст сообщения.

Далее необходимо откомпилировать проект. Для этого можно воспользоваться клавишей F6. Если все проделано верно в окне ошибок и предупреждений будет выведено сообщение о 0 ошибок. Вероятно вам будет сделано несколько предупреждений. Обычно они не препятствуют нормальной работе приложения, однако рекомендуется тщательно с ними ознакомиться и по возможности их устранить.

Результатом компиляции является файл с расширением dll и именем, которое вы задали при создании проекта. Файл следует искать в катеlоге “Debug”, который в свою очередь лежит 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, 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 OK.

Внимание: в NX 7.5 существует понятие “Роль”. В зависимости от выбора роли интерфейс пользователя системы значительно изменяется. Возможность запуска пользовательских прикладных подпрограмм доступна не во всех ролях. Рекомендуем выбирать роль “Расширенные с полным меню”.

Результат выполнения созданной ранее библиотеки показан на рисунке 1.4.

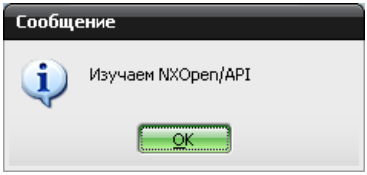

Рисунок 1.4 - Результат работы учебной библиотеки

Так как выгрузка библиотеки осуществляется в момент закрытия NX, попытка перекомпилировать проект будет терпеть неудачу, пока не выгрузите NX.

2. Практиеская часть

2.1. Вопросы для самостоятельного контроля знаний

1. Что такое API?
2. Что такое динамичесие библиотеки?
3. Основные отличия внешних и внутренних прикладных подпрограмм для NX?
4. Что такое Entry Points?

2.2. Задание на лабораторную работу

Задание выдается преподавателем по вариантам.

2.3. 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 на русском или английском языке
5. Выводы

Лабораторная работа №2
Создание детали с помощью операции “Вращение” средствами NXOpen/API на примере крышки пневмоцилиндра

Цель работы: освоить основы твердотельного моделирования в NX 7.5 посредствам API функций, изучить методику создания 3D модели, работу с отдельными 2D примитивами, освоить операцию “Вращение”.

1. Теоретическая часть

В данной лабораторной работе предлагается выполнить крышку, представленную на рис. 3 с некоторыми упрощениями: на текущем этапе из детали исключаются фаски и скругление. Таким образом, деталь приобретает вид, показанный на рис. 2.1.

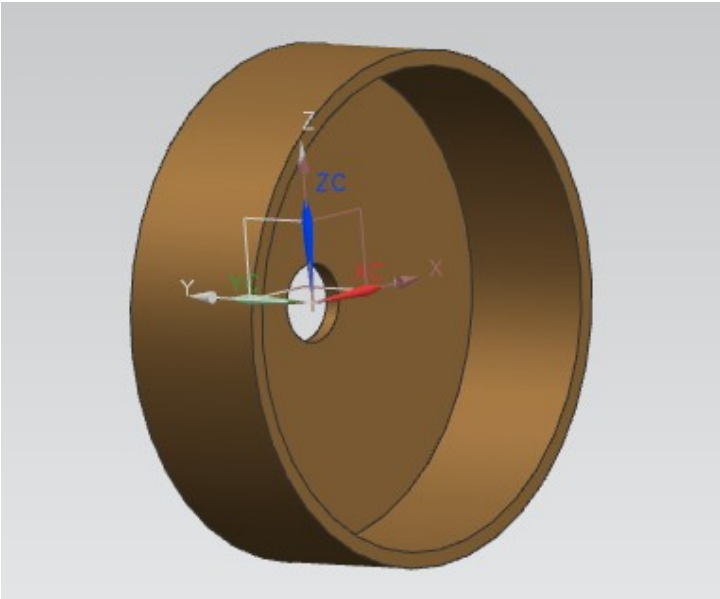

Рисунок 2.1 - Упрощенная модель крышки

Для создания рассматриваемой модели необходимо

осуществить выполнить операцию “Вращение” для эскиза, представленного на рис. 2.2 относительно оси X.

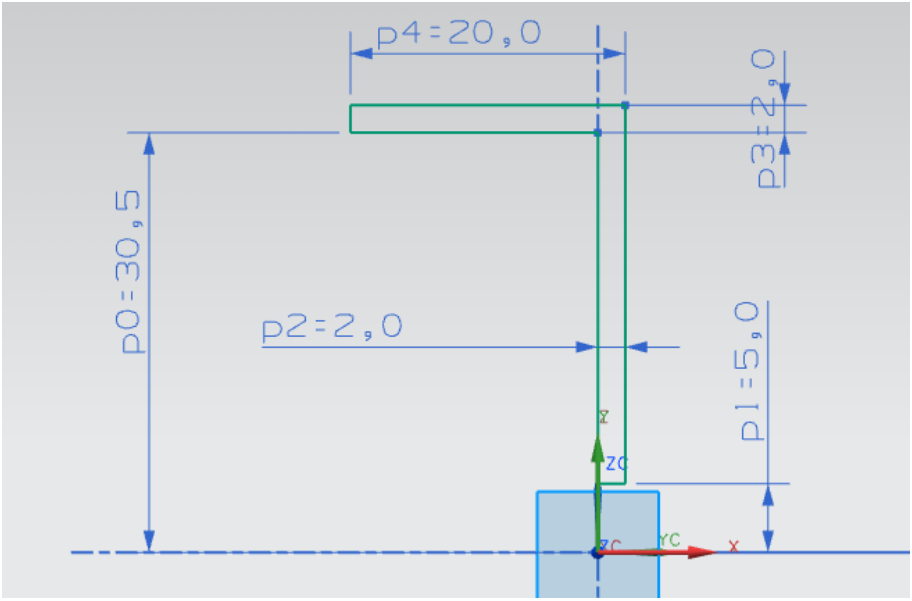

Рисунок 2.2 - Эскиз для операции “Вращение”

Начало работы по созданию данной библиотеки аналогично лабораторной работе №1. Вместо строки, ответственной за вывод сообщения, предлагается вставить программный код, представленный далее. Программный код приведен в полном объеме с подробными комментариями. Последовательность описания кода соответствует действительности. При корректном копировании программного кода из текущего описания в создаваемую библиотеку, будет получена работающая пользовательская подпрограмма.

Рассматриваемая библиотека начинается со следующей конструкции:

```
Tag UFPart1;
string name1 = "model_k";
int units1 = 1;
```

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

Первые 3 строки текста применяются для описания переменных. В данном контексте переменная UFPart1 является деталью, 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 12 однотипных переменных, соответствующих конечным точкам отрезков эскиза.

```
double[] l1_endpt1 = { 0, 5, 0.00 };
double[] l1_endpt2 = { 2, 5, 0.00 };
double[] l2_endpt1 = { 2, 5, 0.00 };
double[] l2_endpt2 = { 2, 32.5, 0.00 };
double[] l3_endpt1 = { 2, 32.5, 0.00 };
double[] l3_endpt2 = { -18, 32.5, 0.00 };
double[] l4_endpt1 = { -18, 32.5, 0.00 };
double[] l4_endpt2 = { -18, 30.5, 0.00 };
double[] l5_endpt1 = { -18, 30.5, 0.00 };
double[] l5_endpt2 = { 0, 30.5, 0.00 };
double[] l6_endpt1 = { 0, 30.5, 0.00 };
double[] l6_endpt2 = { 0, 5, 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 6 новых структур.

```
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();
```

Структуры line1 – line6 относятся к специальному типу NX, соответствующему такому объекту, как отрезок.

В последующем фрагменте программного кода задаются конечные точки отрезков.

```
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];
```

Первая и аналогичные ей строки создают массивы вещественных чисел, в которые будут записываться тройки координат точек отрезка. Строки 2 – 4 и аналогичные им задают координаты по X, Y, Z начальных точек отрезков. Строки 6 – 8 и аналогичные им соответствующим образом задают конечные точки отрезков.

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

```
    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]);
```

Первая строка создает переменную objarray1, представляющую собой массив тэгов из 7 элементов. 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 objarray1.

Дальнейший элемент задает переменные и их значения.

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

double[] direction1 = { 1.00, 0.00, 0.00 };
string[] limit1 = { "0", "360" };
Tag[] features1;
```

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. direction1 – 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 limit1 – массив строкового типа, в который заносится начальный и конечный угол для операции вращения, features1 – массив тэгов, выходной информации для операции “Вращение”.

Последующая строка отвечает за операцию “Вращение”

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

Аргументами данной операции являются:

1. objarray1 – массив элементов эскиза операции;

2. limit1 – начальный и конечный угол вращения;
3. ref_pt1 – базовая точка;
4. direction1 – вектор, относительно которого осуществляется вращение;
5. FeatureSigns.Nullsign – задает булеву операцию, в данном случае операция отсутствует.

Строка

```
theUfSession.Part.Save();
```

сохраняет деталь в файл с именем, заданным переменной name1, по умолчанию путь к сохраненному файлу находится по адресу: C:\Program Files\UGS\NX7.5\UGII.

2. Практическая часть

2.1. Вопросы для самостоятельного контроля знаний

1. Что такое тэг?
2. Как осуществляется работа с динамическими массивами в C#?
3. Назовите основные параметры операции “Вращение”?
4. Что такое булевы операции в 3D инженерной графике?
5. Какова функция вектора в операции “Вращение”?

2.2. Задание на лабораторную работу

Задание выдается преподавателем по вариантам.

2.3. 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 на русском или

английском языке

5. Скриншоты 3D модели, сформированной Вашей библиотекой
6. Выводы.

Лабораторная работа №3
Создание фасок и скруглений с помощью соответствующих операций средствами NXOpen/API на примере крышки пневмоцилиндра

Цель работы: освоить методику создания скруглений и фасок на детали посредствам API функций NX.

1. Теоретическая часть

В данной лабораторной работе предлагается доработать деталь, представленную на рис. 2.1, до состояния детали, показанной на рис. 3. Для этого необходимо создать две фаски и скругление:

Внимание: теоретически имеется возможность создания фасок и скруглений в эскизе, что сокращает количество операций. Данный подход в корне неверен, так как значительно увеличивается потребление вычислительных ресурсов и снижается гибкость создаваемой модели. Поэтому фаски и скругления всегда следует создавать на твердом теле с помощью специализированных операций.

Рассмотренный далее программный код целесообразно разместить перед строкой, выполняющей сохранение детали.

Начальный блок программного кода отвечает за объявление переменных. Все типы представленных здесь переменных рассмотрены в предыдущей лабораторной работе.

```
Tag feat = features1[0];
      Tag cyl_tag, obj_id_camf, blend1;
      Tag[] Edge_array_cyl, list1, list2;
int ecount;
```

Интерес представляет первая трока. В ней из массива тэгов features1 выбирается нулевой тэг, который записывается в переменную feat, теперь в переменной feat находится тэг нулевого тела детали. В рассматриваемом случае он единственный. Однако возможности NX позволяют сохранять в

оном файле детали несколько отдельных тел.

Далее следует 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);
```

Первая строка выделяет из тэга тела тэг объектов тела, который являются входными данными во второй строке. Выходными данными второй строки является массив тэгов ребер объекта, используемый в качестве аргумента в третьей строке. Выходной информацией третьей строки является количество объектов в массиве (в нашем случае число ребер детали).

Далее создаем два объекта типа ArrayList.

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

Первый объект в дальнейшем будет использован для сохранения в него ребер, на которых будут выполнены фаски, второй по аналогии используется для скругления.

Представленный ниже цикл повторяется количество раз, равное число ребер рассматриваемого тела.

```
    for (int ii = 0; ii < ecount; ii++)
        {
           Tag edge;
            theUfSession.Modl.AskListItem(Edge_array_cyl, ii,
out edge);
           if ((ii == 1) || (ii == 2))
               {
                arr_list1.Add(edge);
               }
           if (ii == 0)
               {
                arr_list2.Add(edge);
               }    }
```

Вторая строка тела цикла выбирает из массива ребер каждое отдельное ребро с порядковым номером в массиве ii (номера ребер перебираются в цикле последовательно). Далее следуют две конструкции оператора if. При выполнении условия первого оператора выбранное на текущем шаге цикла ребро добавляется в arr_list1 (то есть на нем будет создана фаска). Второй оператор if аналогично реализует конструкцию формирования листа ребер для скруглений.

Последующие две строки текста осуществляют конвертирование типов данных.

```
list1 = (Tag[])arr_list1.ToArray(typeof(Tag));
list2 = (Tag[])arr_list2.ToArray(typeof(Tag));
```

Полученные переменные list1 и list 2 относятся к типу массив тэгов.

Далее следует фрагмент текста, в котором создается скругление и задается ряд его параметров.

```
int allow_smooth = 0;
int allow_cliff = 0;
int allow_notch = 0;
double vrb_tol = 0.0;

theUfSession.Modl.CreateBlend("3", list2, allow_smooth,
allow_cliff, allow_notch, vrb_tol, out blend1);
```

Аргументами для скругления являются:

1. “3” – радиус;
2. list2 – массив ребер, на которых необходимо выполнить скругление;
3. allow_smooth - Smooth overflow/prevent flag: 0 = Allow this type of blend; 1 = Prevent this type of blend;
4. allow_smooth - Cliffedge overflow/prevent flag: 0 = Allow this type of blend; 1 = Prevent this type of blend;
5. allow_notch - Notch overflow/prevent flag: 0 = Allow this

type of blend; 1 = Prevent this type of blend;

6. vrb_tol - Variable radius blend tolerance.

Последующий текст содержит операцию “Фаска” и задает ее отдельные параметры.

```
string offset1 = "1";
string offset2 = "1";
string ang = "45";
theUfSession.Modl.CreateChamfer(3, offset1, offset2, ang,
list1, out obj_id_camf);
```

Аргументами для фаски являются:

1. 3 – тип входных данных (в данном случае по стороне и углу);
2. offset1 и offset2 – стороны фаски;
3. ang – угол фаски в градусах;
4. list1 – массив ребер, на которых необходимо выполнить фаски.

2. Практическая часть

2.1. Вопросы для самостоятельного контроля знаний

1. Что такое ребро, грань, поверхность?
2. Синтаксис оператора if в C#?
3. Какие способы задания фасок вы знаете?

2.2. Задание на лабораторную работу

Задание выдается преподавателем по вариантам.

2.3. 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 на русском или английском языке

5. Скриншоты 3D модели, сформированной Вашей библиотекой

6. Выводы

Лабораторная работа №4

Создание детали с помощью операции “Выдавливание” средствами NXOpen/API на примере корпуса пневмоцилиндра

Цель работы: Освоение методики построения операции выдавливания средствами NX Open API на примере цилиндра.

1. Теоретическая часть

1.1. Общий вид и порядок построения детали «Цилиндр»

Общий вид детали «Цилиндр» в разрезе представлен на рисунке 4.1.

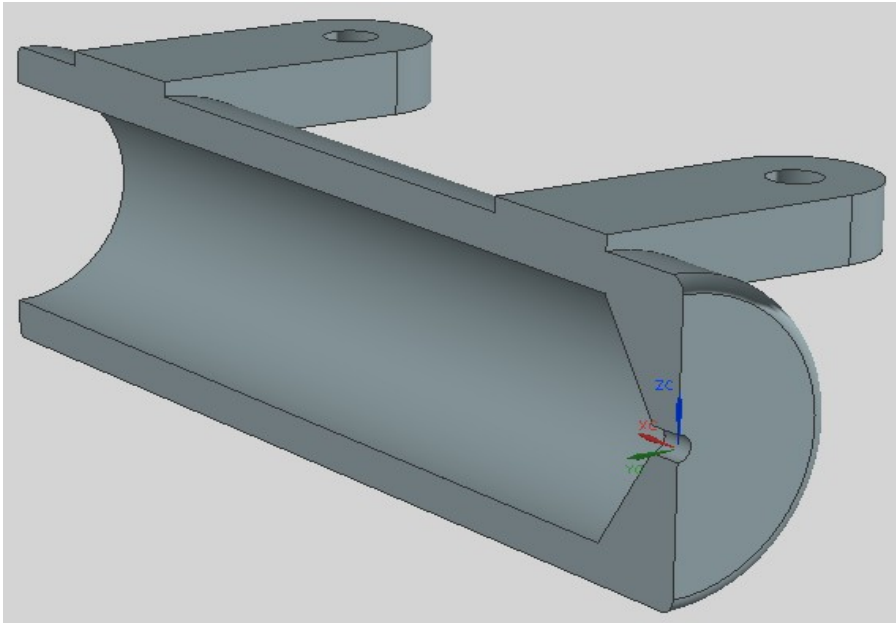

Рисунок 4.1 – Общий вид детали «Цилиндр» в разрезе

Приведем эскизы, входящие в последовательность проектирования 3D модели детали.

Во первых выполняется операция вращения эскиза образующего поверхность основной формы детали. Данный

эскиз с размерами и положением осей представлен на рисунке 4.2.

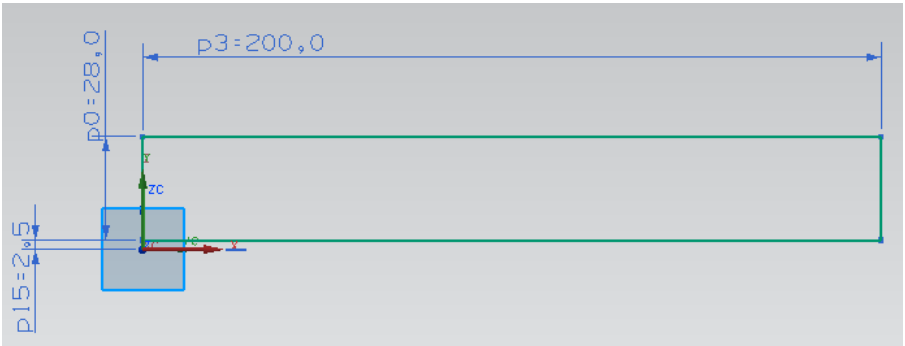

Рисунок 4.2 – Эскиз операции «Вращения»

Создание эскиза данной операции при помощи API функций NX происходит в той же последовательности, что и аналогичная операция, описанная в лабораторной работе №2.

1.2. Программная реализация операции «Выдавливания»

Далее, рассмотрим параметры операции выдавливания, которые ранее не рассматривались в данном методическом пособии. В данной детали операция выдавливания представлена в виде двух опор цилиндра, с отверстиями под крепление. При этом, обе из этих опор строятся последовательно, и имеют одинаковый профиль, эскиз которого изображен на рисунке 4.3.

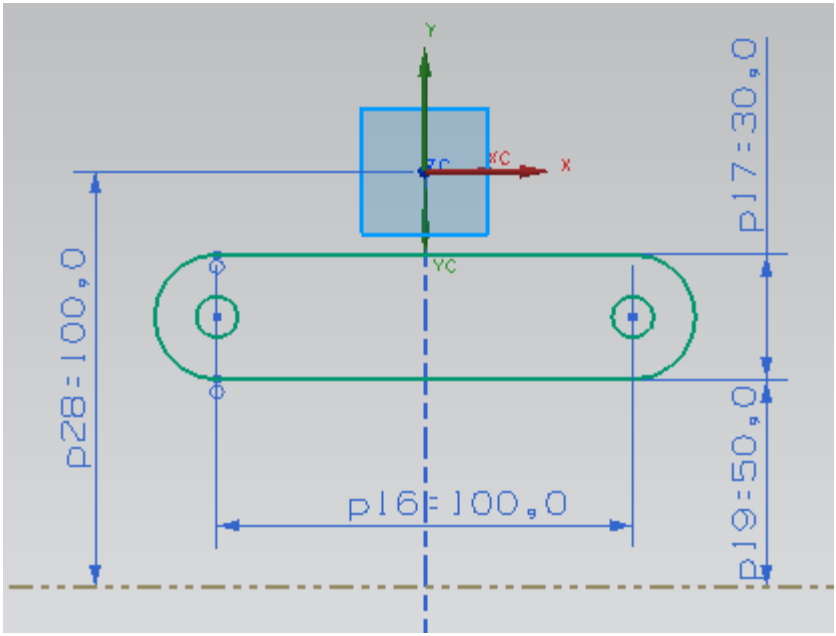

Рисунок 4.3 – Эскиз операции «Выдавливания»

Данный эскиз, определяет операцию выдавливания, образующую одну из опор цилиндра. Вторая опора расположена симметрично горизонтальной оси представленной на рисунке 3.3.

Рассмотрим более подробно построение данной операции на примере. Функция, задающая построение одной из опор начинается с определения требуемых переменных.

```
{
        double[] direction4 = { 0.0, 0.0, 1.0 };
//Переменная задающая значения направления выдавливания ось CZ
        double[] ref_pt4 = new double[3];
//Требуемая, но не используемая переменная

        string taper_angle4 = "0.0";
//Переменная, определяющая значение уклона при выдавливании
        string[] limit4 = { "-10", "3" };
//Переменная, определяющая параметры начала и конца операции
выдавливания
```

```
        int i4, count4 = 6;
// Переменная счетчик и число объектов в эскизе

        Tag[] objarray4 = new Tag[7];
// Массив объектов из 7 элементов. Заполняется указателями на
элементы эскиза выдавливания при их построении (линии и дуги)

        Tag wcs_tag1, matrix_tag1, wcs_tag2, matrix_tag2,
wcs_tag3, matrix_tag3, wcs_tag4, matrix_tag4;
//Переменные wcs_tag1 – для записи указателя на текущую систему
координат; matrix_tag1 – для записи идентификатора матрицы
связанного с объектом и т.д.

        Tag[] features4;
//features4 – переменная для записи указателя на объект,
получившийся в результате операции выдавливания

        double[] arc1_centerpt1 = { 35, -50, 30.5};
//Переменная содержащая значения координат центра дуги 1{x,y,z}
        double arc1_start_ang1 = 0;
//Переменная содержащая значение угла начала дуги 1 (в радианах)
        double arc1_end_ang1 = 3.14159265358979324 * 2;
//Переменная содержащая значение угла конца дуги 1 (в радианах)
        double arc1_rad1 = 5;
//Переменная содержащая значение радиуса дуги 1 (в радианах)

        UFCurve.Arc arc1 = new UFCurve.Arc();
//Создание структуры NX соответствующей дуге 1
//Установка параметров дуги 1
        arc1.start_angle = arc1_start_ang1;
//Начальный угол
        arc1.end_angle = arc1_end_ang1;
//Конечный угол
        arc1.arc_center = new double[3];
//Центр дуги 1
        arc1.arc_center[0] = arc1_centerpt1[0];
//Координата центра дуги 1 по X
        arc1.arc_center[1] = arc1_centerpt1[1];
//Координата центра дуги 1 по Y
        arc1.arc_center[2] = arc1_centerpt1[2];
//Координата центра дуги 1 по Z
        arc1.radius = arc1_rad1;
//Радиус дуги 1
```

```
        theUFSession.Csys.AskWcs(out wcs_tag1);
//Получения указателя на активную систему координат
        theUFSession.Csys.AskMatrixOfObject(wcs_tag1, out
matrix_tag1);
//Получение идентификатора матрицы, связанного с объектом,
указатель на который содержится в wcs_tag1

        arc1.matrix_tag = matrix_tag1;
//Определение указателя матрицы дуги 1
/*-------------------------------------------------------------*/
/**********************Аналогично дуге 1********************/
        double[] arc2_centerpt2 = { 35, 50, 30.5 };
        double arc2_start_ang2 = 0;
        double arc2_end_ang2 = 3.14159265358979324 * 2;
        double arc2_rad2 = 5;

        UFCurve.Arc arc2 = new UFCurve.Arc();
        arc2.start_angle = arc2_start_ang2;
        arc2.end_angle = arc2_end_ang2;
        arc2.arc_center = new double[3];
        arc2.arc_center[0] = arc2_centerpt2[0];
        arc2.arc_center[1] = arc2_centerpt2[1];
        arc2.arc_center[2] = arc2_centerpt2[2];
        arc2.radius = arc2_rad2;

        theUFSession.Csys.AskWcs(out wcs_tag2);

theUFSession.Csys.AskMatrixOfObject(wcs_tag2, out matrix_tag2);

        arc2.matrix_tag = matrix_tag2;
/*-------------------------------------------------------------*/

/**********************Аналогично дуге 1********************/
        double[] arc3_centerpt3 = { 35, -50, 30.5 };
        double arc3_start_ang3 = -3.14159265358979324;
        double arc3_end_ang3 = 0;
        double arc3_rad3 = 15;

        UFCurve.Arc arc3 = new UFCurve.Arc();
        arc3.start_angle = arc3_start_ang3;
        arc3.end_angle = arc3_end_ang3;
        arc3.arc_center = new double[3];
        arc3.arc_center[0] = arc3_centerpt3[0];
```

```
        arc3.arc_center[1] = arc3_centerpt3[1];
        arc3.arc_center[2] = arc3_centerpt3[2];
        arc3.radius = arc3_rad3;

        theUFSession.Csys.AskWcs(out wcs_tag3);

theUFSession.Csys.AskMatrixOfObject(wcs_tag3, out matrix_tag3);

        arc3.matrix_tag = matrix_tag3;
/*-------------------------------------------------------*/

/**********************Аналогично дуге 1*******************/
        double[] arc4_centerpt4 = { 35, 50, 30.5 };
        double arc4_start_ang4 = 0;
        double arc4_end_ang4 = 3.14159265358979324;
        double arc4_rad4 = 15;

        UFCurve.Arc arc4 = new UFCurve.Arc();
        arc4.start_angle = arc4_start_ang4;
        arc4.end_angle = arc4_end_ang4;
        arc4.arc_center = new double[3];
        arc4.arc_center[0] = arc4_centerpt4[0];
        arc4.arc_center[1] = arc4_centerpt4[1];
        arc4.arc_center[2] = arc4_centerpt4[2];
        arc4.radius = arc4_rad4;

        theUFSession.Csys.AskWcs(out wcs_tag4);

theUFSession.Csys.AskMatrixOfObject(wcs_tag4, out matrix_tag4);

        arc4.matrix_tag = matrix_tag4;
/*-------------------------------------------------------*/
//Определение переменных содержащих координаты начальной и
конечной точек отрезков 1 и 2
        double[] l1_endpt1 = { 20, -50, 30.5 };
//Координаты начальной точки отрезка 1
        double[] l1_endpt2 = { 20, 50, 30.5 };
//Координаты конечной точки отрезка 1
        double[] l2_endpt1 = { 50, 50, 30.5 };
//Координаты начальной точки отрезка 2
        double[] l2_endpt2 = { 50, -50, 30.5 };
//Координаты конечной точки отрезка 2

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

```
//Создание переменной объекта отрезок 1
         UFCurve.Line line2 = new UFCurve.Line();
//Создание переменной объекта отрезок 2

//Задание параметров отрезков
        line1.start_point = new double[3];
        line1.start_point[0] = l1_endpt1[0];
//Координата X начальной точки отрезка 1
        line1.start_point[1] = l1_endpt1[1];
//Координата Y начальной точки отрезка 1
        line1.start_point[2] = l1_endpt1[2];
//Координата Z начальной точки отрезка 1
        line1.end_point = new double[3];
        line1.end_point[0] = l1_endpt2[0];
//Координата X конечной точки отрезка 1
        line1.end_point[1] = l1_endpt2[1];
//Координата Y конечной точки отрезка 1
        line1.end_point[2] = l1_endpt2[2];
//Координата Z конечной точки отрезка 1

        line2.start_point = new double[3];
        line2.start_point[0] = l2_endpt1[0];
//Координата X начальной точки отрезка 2
        line2.start_point[1] = l2_endpt1[1];
//Координата Y начальной точки отрезка 2
        line2.start_point[2] = l2_endpt1[2];
//Координата Z начальной точки отрезка 2
        line2.end_point = new double[3];
        line2.end_point[0] = l2_endpt2[0];
//Координата X конечной точки отрезка 2
        line2.end_point[1] = l2_endpt2[1];
//Координата Y конечной точки отрезка 2
        line2.end_point[2] = l2_endpt2[2];
//Координата Z конечной точки отрезка 2

     theUFSession.Curve.CreateArc(ref arc1/*объект дуга 1*/, out
objarray4[0]/*указатель на созданный объект дуга 1 – 0-й элемент
массива объектов выдавливания*/);
//Построение дуги 1
     theUFSession.Curve.CreateLine(ref line1, out objarray4[1]);
//Построение отрезка 1
     theUFSession.Curve.CreateLine(ref line2, out objarray4[2]);
//Построение отрезка 2
     theUFSession.Curve.CreateArc(ref arc2, out objarray4[3]);
```

```
//Построение дуги 2
     theUFSession.Curve.CreateArc(ref arc3, out objarray4[4]);
//Построение дуги 3
     theUFSession.Curve.CreateArc(ref arc4, out objarray4[5]);
//Построение дуги 4
//Создание операции выдавливания
theUFSession.Modl.CreateExtruded(
objarray4/*Массив объектов выдавливания*/,
taper_angle4/*Угол уклона*/,
limit4/*Начало и конец выдавливания*/, ъ
ref_pt4 /*Пустой параметр*/,
direction4/*Направление выдавливания*/,
FeatureSigns.Positive/*Буревая операция (ОБЕДИНЕНИЕ)*/,
out features4/*Выходной параметр - указатель на результат
операции*/);
}
```

2. Практическая часть

2.1 Вопросы для самопроверки

1. Дайте пояснение, какие переменные в примере содержат указатели на объекты
2. Расскажите механизм построения дуги
3. Как объявляется переменная содержащая объект?
4. Как задается переменная содержащая указатель на объект?
5. Поясните содержимое и типы параметров операции выдавливания?
6. Как получить матрицу параметров связанную с объектом, из указателя на данный объект?
7. Сколько параметров содержит объект дуга?
8. Сколько параметров содержит объект отрезок?
9. Опишите содержимое и типы параметров при построении примитива (дуга, линия)?

2.2 Задания для выполнения лабораторной работы

Задание выдается преподавателем по вариантам.

2.3. 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 на русском или английском языке
5. Скриншоты 3D модели, сформированной Вашей библиотекой
6. Выводы

Лабораторная работа №5
Создание детали с помощью операции “Вырезание вращением” средствами NXOpen/API на примере корпуса пневмоцилиндра

Цель работы: Освоение методики построения операции получения отверстия вырезанием средствами NX Open API на примере цилиндра.

1. Теоретическая часть

1.1. Программная реализация операции вырезания выдавливанием

Завершающим элементом построения детали цилиндр, является поверхность, образованная путем вычитания элемента вращения, образующего центральное отверстие цилиндра под поршень.

Эскиз элемента вращения представлен на рисунке 5.1.

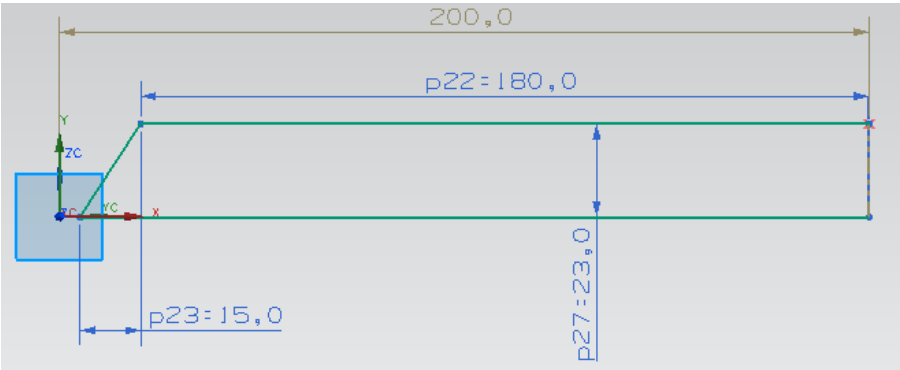

Рисунок 5.1 – Эскиз операции «Вырезания»

В данном случае, эскиз и сама операция, создается аналогично представленной выше функции построения детали «Корпус» (см. ЛР №4), с одним основным отличием в параметре, определяющем вид булевой операции при вызове функции вращения эскиза.

```
theUFSession.Modl.CreateRevolved(loop_list6, limit6, ref_pt6,
direction6, FeatureSigns.Negative, out features6);
```

При этом параметр ***FeatureSigns*** функции *CreateRevolved* должен быть определен как ***Negativ*** (вычитание), а не ***Nullsign*** (операция отсутствует).

Рассмотрим подробно программную реализацию этой функции.

```
{//Переменная задающая значения направления выдавливания ось CX
        double[] direction6 = { 1.00, 0.00, 0.00 };
//Задание координат начальной точки вращения
        double[] ref_pt6 = new double[3];
        ref_pt6[0] = 0.00;
        ref_pt6[1] = 0.00;
        ref_pt6[2] = 0.00;
//Задание пределов вращения
        string[] limit6 = { "0", "360" };
//объявление массива объектов вращения
        Tag[] objarray6 = new Tag[5];
//Объявление и определение переменны содержащих координаты точек
отрезков эскиза
        double[] l1_endpt1 = { 5, 0, 0.00 };
        double[] l1_endpt2 = { 200, 0, 0.00 };
        double[] l2_endpt1 = { 200, 0, 0.00 };
        double[] l2_endpt2 = { 200, 22.85, 0.00 };
        double[] l3_endpt1 = { 200, 22.85, 0.00 };
        double[] l3_endpt2 = { 20, 22.85, 0.00 };
        double[] l4_endpt1 = { 20, 22.85, 0.00 };
        double[] l4_endpt2 = { 5, 0, 0.00 };
// Переменная для записи указателя на объект, получившийся в
результате операции вырезания вращением
        Tag[] features6;
//Создание структур NX соответствующих отрезкам эскиза
        UFCurve.Line line1 = new UFCurve.Line();
        UFCurve.Line line2 = new UFCurve.Line();
        UFCurve.Line line3 = new UFCurve.Line();
        UFCurve.Line line4 = new UFCurve.Line();
//-----------Задаются конечные точки отрезков---------------
        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];
//------------------------------------------------
//Построение отрезков в 3D пространстве
     theUFSession.Curve.CreateLine(ref line1, out objarray6[0]);
     theUFSession.Curve.CreateLine(ref line2, out objarray6[1]);
     theUFSession.Curve.CreateLine(ref line3, out objarray6[2]);
     theUFSession.Curve.CreateLine(ref line4, out objarray6[3]);
//Создание операции вырезания вращением
  theUFSession.Modl.CreateRevolved(objarray6, limit6, ref_pt6,
direction6, FeatureSigns.Negative, out features6);
}
```

2. Практическая часть

2.1. Вопросы для самоконтроля

1) Чем определяется операция вырезание в NX?

2) Какое необходимое и достаточное количество переменных следует задать для создания операции вырезания вращением?

3) Какова функция начальной точки в операции “Вырезание вращением”?

4) Что содержит переменная, в которую записывается выходной параметр операции вырезания вращением?

2.2 Задания для выполнения лабораторной работы

Задание выдается преподавателем по вариантам.

2.3. 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 на русском или английском языке
5. Скриншоты 3D модели, сформированной Вашей библиотекой
6. Выводы

Лабораторная работа №6
Создание сборки конструкции из отдельных деталей средствами NXOpen/API на примере пневмоцилиндра

Цель работы: Освоение методики размещения моделей деталей в абсолютной системе координат пространства сборочной модели, средствами NX Open API на примере пневмоцилиндра.

1. Теоретическая часть

В данной лабораторной работе предлагается выполнить сборочную модель, представленную на рис. 1. Таким образом, деталь приобретает конечный вид.

Приведем подробное описание программной реализации построения сборочной 3D модели.

Вначале программы создадим новую модель, которая будет являться сборочной для пневмоцилиндра. Следующие три строки описываются переменные для создания новой модели.

```
Tag UFPart;
string part_name = "Cylindr";
int units = 1;
```

Далее реализуется сам процесс создания сборки. Данная структура не является новой и подробно описана в ЛР №2, поскольку абсолютно аналогична созданию простой детали.

```
{
  theUFSession.Part.New(part_name, units, out UFPart);
}
```

Приведем описание трех последующих строк.

```
{
Tag parent_part = theUFSession.Part.AskDisplayPart();
UFPart.LoadStatus error_status, error_status2, error_status3;
Tag instance, instance1, instance2;
```

Во первых, 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.

Во вторых, объявляются переменные error_status, в которые будет записываться статус загрузки моделей.

В третьих, объявление переменных в которые происходит запись tag-а нового объекта модели.

В следующем блоке программы происходит объявление и запись значений в переменные.

```
double[] origin1 = { 200, 0, 0 };
double[] matrix1 = { 1, 0, 0, 0, 1, 0 };
double[] origin2 = { 5, 0, 0 };
double[] matrix2 = { 1, 0, 0, 0, 1, 0 };
double[] origin3 = { 0, 0, 0 };
double[] matrix3 = { 1, 0, 0, 0, 1, 0 };
```

Переменные origin содержат позиции каждой из моделей в родительской сборочной модели.

Переменные matrix определяют ориентацию каждой из моделей в системе координат родительской сборочной модели.

В приведенном ниже тексте программы, описывается поочередное добавление моделей деталей пневмоцилиндра, созданных в предыдущих лабораторных работах, 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
, "model1", null, null, origin1, matrix1, 0, out instance, out
error_status);

                theUFSession.Assem.AddPartToAssembly(parent_part
, "model2", null, null, origin2, matrix2, 0, out instance1, out
error_status2);
```

```
                theUFSession.Assem.AddPartToAssembly(parent_part
, "model3", null, null, origin3, matrix3, 0, out instance2, out
error_status3);
    }
```

2. Практическая часть

2.1. Вопросы для самоконтроля

1) Чем отличаются сборочная и простая модели в NX?
2) Какие переменные определяют добавляемую деталь в сборку?
3) Без каких функций не обойтись при добавлении детали в сборку?
4) Какие условия следует соблюдать при воздании сборки данным способом?

2.2. Задания для выполнения лабораторной работы

Задание выдается преподавателем по вариантам.

2.3. 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 на русском или английском языке
5. Скриншоты 3D модели, сформированной Вашей библиотекой
6. Выводы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Интерактивное руководство в системе NX7.5 English по NX Open API.

2. Гончаров П.С. NX для конструктора-машиностроителя / Гончаров П.С., Ельцов М.Ю., Коршиков С.Б., Лаптев И.В., Осиюк В.А. - М.: ДМК-Пресс, 2010 -504 с.

3. Краснов М. Unigraphics для профессионалов / М. Краснов, Ю. Чигишев. - М.:Лори, 2004 - 141с.

4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0, 5-е изд. / Троелсен Э. .-М.:ООО “И.Д. Вильямс”, 2011 -1392 с.

5. Шилдт Г. C# 4.0: полное руководство 2010 / Шилдт Г. -М.:ООО “И.Д. Вильямс”, 2011 -1056 с.: ил.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам по дисциплине «Автоматизация конструкторского и технологического проектирования» для студентов специальности 230104 «Системы автоматизированного проектирования» очной формы обучения

Составители:
Чижов Михаил Иванович
Паринов Максим Викторович
Рыжков Владимир Анатольевич
Еремен Иван Александрович

В авторской редакции

Компьютерный набор
В.А.Рыжков, М.В. Паринов

Подписано в печать __.__.2011.
Формат 60x84/16. Бумага для множительных аппаратов.
Усл. печ. л. 2,8. Уч.-изд. л. 2,7. Тираж 50 экз. «С»
Зак. №_____

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