МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И РОБОТОТЕХНИКИ

Кафедра “Системы автоматизированного проектирования”

**Пояснительная записка к курсовой работе**

по курсу “Геометрическое моделирование”

**«Получение реалистичного изображения объекта, представленного полигональной сеткой, на базе трёхмерной геометрической модели»**

**Исполнитель:** студент гр.10702314

Азаревич В. А.

**Руководитель:** Носкова Л.А.

Минск 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ……………………………………………………………………………………....4

ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ…………………………………………………………….6

ПОСТАНОВКА ЗАДАЧИ……………………………………………………………………….7

ОПИСАНИЕ ОБЪЕКТА. ПОЛУЧЕНИЕ ГЕОМЕТРИЧЕСКОЙ МОДЕЛИ…………………8

РЕДАКТИРОВАНИЕ ОБЪЕКТА……………………………………………………………...10

ПОЛУЧЕНИЕ ВСЕХ ВИДОВ ПРОЕКЦИЙ…………………………………………………..14

ЗАКЛЮЧЕНИЕ…………………………………………………………………………………19

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА………………………………………………………..20

ПРИЛОЖЕНИЕ………………………………………………………………………………....21

# ВВЕДЕНИЕ

Геометрическое моделирование и компьютерная графика – наиболее важные и динамично развивающиеся области информационных технологий. Они находят свое применение в различных сферах деятельности: от компьютерных игр и спецэффектов для кино до сложных научных симуляций.

Особое место компьютерная графика занимает в промышленных инженерных пакетах. Она используется при создании и редактировании деталей и сборок в системах автоматизированного проектирования (CAD), при моделировании различных процессов в системах инженерного анализа (CAE), а также для визуализации работы станов с ЧПУ в системах технологической подготовки производства (CAM).

Кроме этих основных направлений существуют также некоторые другие: например, промышленный дизайн, крэш-тесты и тому подобное.

По способам задания изображений компьютерную графику можно разделить на двумерную и трехмерную.

Двумерная компьютерная графика классифицируется по типу представления графической информации, и следующими из него алгоритмами обработки изображений. Обычно компьютерную графику разделяют на векторную и растровую.

[Векторная графика](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) представляет изображение как набор геометрических примитивов. Обычно в качестве них выбираются точки, прямые, окружности, прямоугольники, а также как общий случай, [сплайны](http://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD) некоторого порядка. Объектам присваиваются некоторые атрибуты, например, толщина линий, цвет заполнения. Рисунок хранится как набор координат, векторов и других чисел, характеризующих набор примитивов. При воспроизведении перекрывающихся объектов имеет значение их порядок. Следует отметить, что не всякое изображение можно представить как набор из примитивов. Такой способ представления хорош для схем, используется для масштабируемых шрифтов, деловой графики, очень широко используется для создания мультфильмов и просто роликов разного содержания.

[Растровая графика](http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) всегда оперирует двумерным массивом (матрицей) [пикселов](http://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D1%81%D0%B5%D0%BB). Каждому пикселю сопоставляется значение — яркости, цвета, прозрачности — или комбинация этих значений. Растровый образ имеет некоторое число строк и столбцов.

В растровом виде представимо любое изображение, однако этот способ хранения имеет свои недостатки: больший объём памяти, необходимый для работы с изображениями, потери при редактировании.

[Трёхмерная графика](http://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) оперирует с объектами в трёхмерном пространстве. Обычно результаты представляют собой плоскую картинку, [проекцию](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F%29). Трёхмерная компьютерная графика широко используется в кино, компьютерных играх.

В трёхмерной компьютерной графике все объекты обычно представляются как набор поверхностей или частиц. Минимальную поверхность называют [полигоном](http://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%BE%D0%BB%D0%B8%D0%B3%D0%BE%D0%BD_%28%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0%29&action=edit&redlink=1). В качестве полигона обычно выбирают треугольники.

Всеми визуальными преобразованиями в 3D-графике управляют [матрицы](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0). Так в компьютерной графике используется три вида матриц:

* [матрица поворота](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BF%D0%BE%D0%B2%D0%BE%D1%80%D0%BE%D1%82%D0%B0)
* матрица сдвига
* матрица масштабирования

Любой полигон можно представить в виде набора из координат его вершин. [Координаты](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D1%8B) каждой вершины представляют собой [вектор](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80) (x, y, z). Умножив вектор на соответствующую матрицу, мы получим новый вектор. Сделав такое преобразование со всеми вершинами полигона, получим новый полигон, а преобразовав все полигоны, получим новый объект, повёрнутый/сдвинутый/промасштабированный относительно исходного.

Задача трехмерного моделирования состоит в том, что нужно определить структуру данных, описывающую трехмерное тело, на основе полученной структуры получить возможность изменять объект, изменять его положение в пространстве. Также к задачам трехмерного моделирования относятся задачи проекционного черчения, реализации реалистического изображения объекта с учетом таких факторов как свет, материал, шероховатость поверхностей и т.д.

Решение этой задачи облегчает работу пользователя (конструктора, инженера), связанную с проектированием каких-либо объектов, т.к. традиционное проекционное черчение не дает реального изображения предмета. Реальное изображение объекта легче воспринимается человеком, что делает работу более комфортной, а, следовательно, и более продуктивной.

Системы геометрического моделирования делятся на два больших класса: твердотельные и поверхностные. В данном курсовом проекте реализована система поверхностного моделирования, так как она значительно проще и является вполне адекватной в рамках поставленной задачи.

# 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

Данный курсовой проект реализован в среде Microsoft Visual Studio 2013 ultimate на языке C#.

Язык C# является наиболее известной новинкой в области создания языков программирования. В отличие от 60-х годов XX века - периода бурного языкотворчества - в нынешнее время языки создаются крайне редко. Явившись на свет в недрах Microsoft, будучи наследником C++, он с первых своих шагов получил мощную поддержку. Однако этого явно недостаточно для настоящего признания достоинств языка. Попробуем разобраться, имеет ли он большое будущее?

Создателем языка является сотрудник Microsoft Андреас Хейлсберг. Он стал известным в мире программистов задолго до того, как пришел в Microsoft. Как отмечал сам Андреас Хейлсберг, C# создавался как язык компонентного программирования, и в этом одно из главных достоинств языка, направленное на возможность повторного использования созданных компонентов. Из других объективных факторов отмеследующие:

* C# создавался параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности - как FCL, так и CLR;
* C# является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами;
* C# является мощным объектным языком с возможностями наследования и универсализации;
* C# является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования. Общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от С++ к C#;
* язык стал проще и надежнее. Простота и надежность, главным образом, связаны с тем, что на C# хотя и допускаются, но не поощряются такие опасные свойства С++ как указатели, адресация, разыменование, адресная арифметика;
* благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Эффективность кода даже повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода;
* мощная библиотека каркаса поддерживает удобство построения различных типов приложений на C#, позволяя легко строить Web-службы, другие виды компонентов, достаточно просто сохранять и получать информацию из базы данных и других хранилищ данных;
* реализация, сочетающая построение надежного и эффективного кода, является немаловажным фактором, способствующим успеху C#.

Как уже отмечалось, Visual Studio .Net для языка C# предлагает 12 возможных видов проектов. Среди них есть пустой проект, в котором изначально не содержится никакой функциональности; есть также проект, ориентированный на создание Web-служб.

Решение содержит один или несколько проектов, ресурсы, необходимые этим проектам, возможно, дополнительные файлы, не входящие в проекты. Один из проектоврешения должен быть выделен и назначен стартовым проектом. Проект состоит из классов, собранных в одном или нескольких пространствах имен. Пространства имен позволяют структурироватьпроекты, содержащие большое число классов, объединяя в одну группу близкие классы. Если над проектом работает несколько исполнителей, то, как правило, каждый из них создает свое пространство имен. Помимо структуризации, это дает возможность присваивать классам имена, не задумываясь об их уникальности. В разных пространствах имен могут существовать одноименные классы.

# 2. ПОСТАНОВКА ЗАДАЧИ

1. Получить 3-х мерную модель объекта, представляющую цилиндр с отверстием в форме пирамиды. Реализовать возможность ввода и изменений пользователем необходимых параметров модели (высота цилиндра, радиус цилиндра; радиус окружности описанной вокруг основания пирамиды, степень аппроксимации);
2. Реализовать возможность геометрических преобразований объекта: перемещение, поворот, масштабирование;
3. Получить все виды проекций данной модели: фронтальная, горизонтальная, профильная, аксонометрическая, косоугольная и перспективная, с возможностью изменения параметров аксонометрической (углы ϕ и ψ), косоугольной (угол α и параметрL) и перспективной проекций (сферические координаты  и параметр d – масштабный коэффициент);

В данном курсовом проекте выбрана левосторонняя система координат( Рис. 2.1 ).

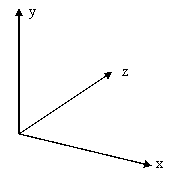


Рис.2.1 – Левосторонняя cистема координат

# 3. ОПИСАНИЕ ОБЪЕКТА. ПОЛУЧЕНИЕ ГЕОМЕТРИЧЕСКОЙ МОДЕЛИ.

3.1 Описание объекта

Модель состоит из цилиндра с призматическим отверстием. (Рис. 3.1)

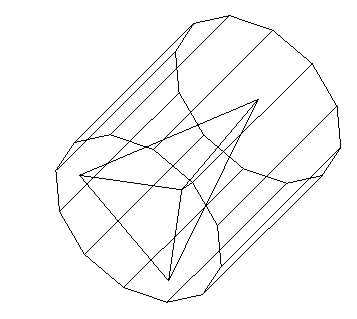


Рис. 3.1 Проектируемый объект

3.1.1 Размеры цилиндра

В данной курсовой работе, реализована возможность изменения геометрических параметров модели: радиус цилиндра, высота цилиндра, степень аппроксимации.(Рис. 3.2)

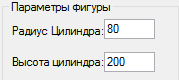
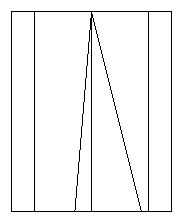
 

Рис. 3.2 Изменение параметров цилиндра

3.1.2 Размеры пирамиды и аппроксимация

В данной курсовой работе, реализована возможность изменения геометрических параметров конуса: радиус окружности описанной вокруг основания пирамиды(Рис. 3.3).

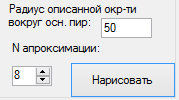
.

Рис. 3.3 Изменение параметров пирамиды и аппроксимации

* 1. Получение геометрической модели

3.2.1 Выбор мировой системы координат

Для удобства работы с координатами вершин, систему координат расположим согласно рисунку(Рис. 3.4):

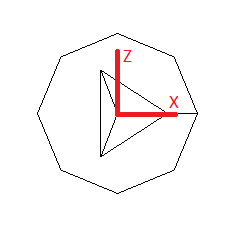
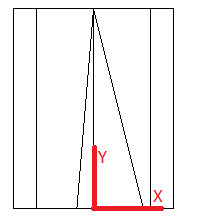


Рис. 3.4 Выбор мировой системы координат

3.2.2 Расчёт координат

Координаты призмы:

pPri[0].X = 0; pPri[0].Y = 0; pPri[0].Z = priR;

pPri[1].X = priR \* Math.Sin(2 \* Math.PI / 5); pPri[1].Y = 0; pPri[1].Z = priR \* Math.Cos(2 \* Math.PI / 5);

pPri[2].X = priR \* Math.Sin(2 \* Math.PI /10); pPri[2].Y = 0; pPri[2].Z = -priR \* Math.Cos(2 \* Math.PI / 10);

pPri[3].X = -priR \* Math.Sin(2 \* Math.PI / 10); pPri[3].Y = 0; pPri[3].Z = -priR \* Math.Cos(2 \* Math.PI / 10);

pPri[4].X = -priR \* Math.Sin(2 \* Math.PI / 5); pPri[4].Y = 0; pPri[4].Z = priR \* Math.Cos(2 \* Math.PI / 5);

pPri[5].X = 0; pPri[5].Y = inpriC; pPri[5].Z = priR;

pPri[6].X = priR \* Math.Sin(2 \* Math.PI / 5); pPri[6].Y = inpriC; pPri[6].Z = priR \* Math.Cos(2 \* Math.PI / 5);

pPri[7].X = priR \* Math.Sin(2 \* Math.PI / 10); pPri[7].Y = inpriC; pPri[7].Z = -priR \* Math.Cos(2 \* Math.PI / 10);

pPri[8].X = -priR \* Math.Sin(2 \* Math.PI / 10); pPri[8].Y = inpriC; pPri[8].Z = -priR \* Math.Cos(2 \* Math.PI / 10);

pPri[9].X = -priR \* Math.Sin(2 \* Math.PI / 5); pPri[9].Y = inpriC; pPri[9].Z = priR \* Math.Cos(2 \* Math.PI / 5);

Координаты цилиндра:

for (i = 0; i < gCil.Length; i++)

gCil[i].Points.Clear();

for (i = 0; i < n - 1; i++)

{

gCil[i].Points.Add(upperCircle[i]); gCil[i].Points.Add(upperCircle[i+1]);

gCil[i].Points.Add(lowerCircle[i + 1]); gCil[i].Points.Add(lowerCircle[i]);

}

gCil[n - 1].Points.Add(upperCircle[n - 1]); gCil[n-1].Points.Add(upperCircle[0]);

gCil[n - 1].Points.Add(lowerCircle[0]); gCil[n - 1].Points.Add(lowerCircle[n - 1]);

3.2.3 Получение модели

Чтобы решить поставленную задачу необходимо получить трехмерную геометрическую модель выше описанного тела. В данной работе будем использовать совокупность двух моделей: каркасной и поверхностной. В каркасной модели описываются вершины (их координаты) и ребра, как отрезки между вершинами. Это делается следующим образом:

*V1={X1, Y1, Z1}* – вершина, заданная координатами;

*E1={V1, V2}* – ребро, заданное двумя вершинами,

где *V1* и *V2* – это указатели на конкретные вершины.

На основе такой модели можно получать проекции, но нельзя удалять невидимые линии. Поэтому необходимо также и поверхностная модель, в которой нужно описать каждую грань нашего объекта.

Зададим каждую грань через вершины:

*P1={V1, V2, V3, ...},*

где *{V1, V2, V3, ...}* – список всех вершин многоугольника, ограничивающего грань, причем вершины в этом списке следуют в направлении обхода против часовой стрелке.

# 4. РЕДАКТИРОВАНИЕ ОБЪЕКТА

Для возможности геометрических преобразований модели объекта необходимо применять соответствующие матрицы преобразований.

## 4.1 Перенос

Трехмерный перенос является простым расширением двумерного:





.

В программе данная матрица реализована следующим образом:

Matrix R = new Matrix(4, 4);

R[0, 0] = 1; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 1; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = 1; R[2, 3] = 0;

R[3, 0] = disX; R[3, 1] = disY; R[3, 2] = disZ; R[3, 3] = 1;

disX- значение переноса относительно оси *OX*.

disY- значение переноса относительно оси *OY*.

disZ- значение переноса относительно оси *OZ*.

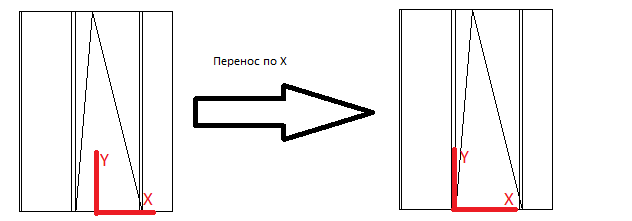


Рис.4.1 - Перенос объекта по оси *OX*

## 4.2 Поворот

Поворот вокруг оси Xосуществляется с помощью метода RotateX.

Матрица поворота вокруг оси :



где θ - угол поворота относительно оси *OX.* В программе данная матрица имеет следующий вид:

R[0, 0] = 1;

R[0, 1] = 0;

R[0, 2] = 0;

R[0, 3] = 0;

R[1, 0] = 0;

R[1, 1] = Math.Cos(radians);

R[1, 2] = Math.Sin(radians);

R[1, 3] = 0;

R[2, 0] = 0;

R[2, 1] = -Math.Sin(radians);

R[2, 2] = Math.Cos(radians);

R[2, 3] = 0;

R[3, 0] = 0;

R[3, 1] = 0;

R[3, 2] = 0;

R[3, 3] = 1;

Где radians – значение введенного угла поворота в радианах.

Поворот вокруг оси Y осуществляется с помощью метода RotateY.

Матрица поворота вокруг оси :



где θ - угол поворота относительно оси *OY.*

Поворот вокруг оси Z осуществляется с помощью метода RotateZ.

Матрица поворота вокруг осиZ:



где θ - угол поворота относительно оси *OZ.*

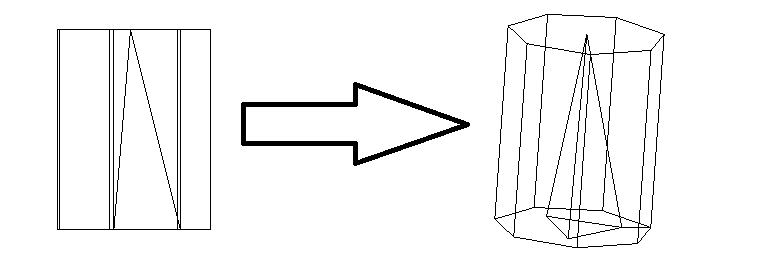


Рис. 4.2 – Поворот объекта относительно оси *OX* и *OY* на углы в 15°

## 4.3 Масштабирование

Масштабирование осуществляется с помощью методов ScaleAll и Scale

Матрица масштабирования выглядит следующим образом:



, или .

Данная матрица в программе реализована следующим способом:

R[0, 0] = scX; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = scY; R[1, 2] = 0; R[1, 3] = 0;

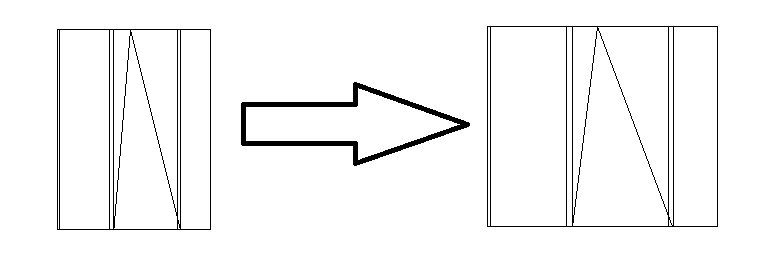
R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = scZ; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

scX- коэффициент масштабирования относительно оси *OX*.

scY- коэффициент масштабирования относительно оси *OY*.

scZ- коэффициент масштабирования относительно оси *OZ*.



После масштабирования

До масштабирования

Рис. 4.3 – Пример масштабирования объекта (scX=1,5)

# 5. ПОЛУЧЕНИЕ ВСЕХ ВИДОВ ПРОЕКЦИЙ

Различают следующие виды проекций: параллельные и центральные. Данные проекции в свою очередь делятся следующим образом, параллельные – на прямоугольные и косоугольные, центральные – на одноточечные, двухточечные и трехточечные центральные проекции.

В данной программе реализована возможность получения следующих видов проекций: фронтальной, горизонтальной, профильной, аксонометрической, косоугольной и перспективной проекций.

## 5.1 Фронтальная проекция

При построении фронтальной проекции принимаем, что наша проекционная плоскость - Z=0, а направление проецирования равно направлению оси Z (). И для получения **фронтальной** проекции нашего объекта матрицу точек надо умножить на матрицу фронтальной проекции .

Матрица фронтальной проекции имеет следующий вид:

Matrix R = new Matrix(4, 4);

R[0, 0] = 1; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 1; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = 0; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

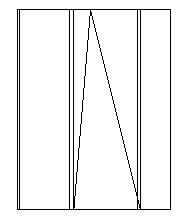


Рис.5.1 – Фронтальная проекция объекта

## 5.2 Горизонтальная проекция

Для получения **горизонтальной** проекции матрицу точек надо умножить на матрицу горизонтальной проекции  (в данном случае мы принимаем за проекционную плоскость Y = 0, а за направление проецирования – направление оси Y).

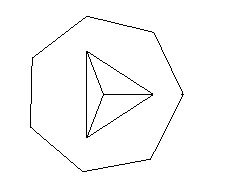


Рис. 5.2 – Горизонтальная проекция объекта

## 5.3 Профильная проекция

Для получения **профильной** проекции матрицу точек надо умножить на матрицу профильной проекции  (за проекционную плоскость X=0, направление проецирования – направление оси X).

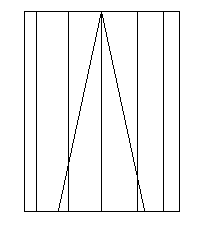


Рис. 5.3 – Профильная проекция объекта

## Аксонометрическая проекция

Аксонометрическая проекция получается при повороте на угол относительно оси У и на угол  относительно оси Х с последующим проецированием вдоль оси Z.

Матрицы точек объекта надо умножить на матрицу аксонометрической проекции:



При выборе аксонометрической проекции необходимо ввести углы  и .

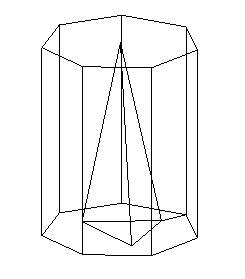


Рис. 5.4 –Построение аксонометрической проекции модели

## 5.5 Косоугольная проекция

Для построения косоугольной проекции матрицу точек нашего объекта надо умножить на матрицу косоугольной проекции:

.

При этом необходимо ввести параметры Lи уголα. Значение данных примеров видно из данного рисунка.

27

Рис. 5.5 – Значение параметров Lи углаα на примере единичного куба

В программе матрица косоугольной проекции имеет следующий вид:

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = 1; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 1; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = L \* Math.Cos(alfa); R[2, 1] = L \* Math.Sin(alfa); R[2, 2] = 0; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

Применение матрицы  приводит к сдвигу и последующему проецированию объекта. Плоскости с постоянной координатой *z=z1* переносятся в направлении *х* на *,* в направлении *y* — на ** и затем проецируется на плоскость *z=0.* Сдвиг сохраняет параллельность прямых, а также углы и расстояния в плоскостях, параллельных оси *z*.

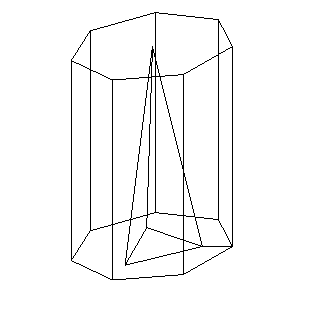


Рис.5.6- Пример построение косоугольной проекции

## 5.5 Перспективная проекция

Для получения перспективной проекции необходимо разделить на *Z* все точки объекта и умножить их на *d,* где *d –* своеобразный масштабный коэффициент, расстояние между точкой наблюденияи экраном.Это расстояние является фактором, приводящим к тому, что удаленные объекты выглядят мельче. Допустимы все значения *z*, кроме *z=0*.



Но существует еще так называемое видовое преобразование. Для этого мы переносим точку наблюдении и задаем ее в сферических координатах. Далее необходимо преобразовать мировую систему координат в видовую, то есть задача сводится к преобразованию СК.

Результирующая матрица преобразования имеет следующий вид:



*ϕ* - угол поворота относительно оси *OX*

*θ* - угол поворота относительно оси*OZ*

*ρ*- расстояние от центра мировой системы координат доточки наблюдения

d- масштабный коэффициент(расстояние между точкой наблюдения и экраном).

Матрица преобразования:

MatrixR = newMatrix(4, 4);

R[0, 0] = -Math.Sin(tetta); R[0, 1] = -Math.Cos(fi) \* Math.Cos(tetta); R[0, 2] = -Math.Sin(fi) \* Math.Cos(tetta); R[0, 3] = 0;

R[1, 0] = Math.Cos(tetta); R[1, 1] = -Math.Cos(fi) \* Math.Sin(tetta); R[1, 2] = -Math.Sin(fi) \* Math.Sin(tetta); R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = Math.Sin(fi); R[2, 2] = -Math.Cos(fi); R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = ro; R[3, 3] = 1;

Как параметры влияют на отображение проекции рассмотрим на примерах:

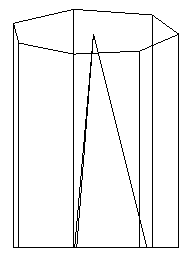


Рис. 5.7 – Пример построения профильной проекции

# ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта была получена 3-х мерная геометрическая модель цилиндра с отверстием в виде треугольной пирамиды. В программе реализована возможность получения различных проекций с изменяемыми параметрами.

# ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

1. Сиденко Л.А. Компьютерная графика и геометрическое моделирование. Питер – 2009, 224 с.
2. Э. Троелсен. Язык программирования С# 2008 и платформа .NET 3.5

**ПРИЛОЖЕНИЕ**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace MyCourseWork

{

struct Point3

{

public Point3(double inX, double inY, double inZ)

{

X = inX;

Y = inY;

Z = inZ;

}

public double X;

public double Y;

public double Z;

}

struct Rebro

{

public Point3 A;

public Point3 B;

}

class Gran

{

public Gran(Color cf)

{

Points = new List<Point3>();

ColorFill = cf;

}

public List<Point3> Points;

public Color Color;

public Color ColorFill;

}

class MyDraw

{

private Point3[] upperCircle, lowerCircle, pPir = new Point3[100];

private Rebro[] verticalRebra, horizontalRebraUpper, horizontalRebraLower, rPir = new Rebro[200];

private Gran[] gCil, gPir, gTemp = new Gran[100];

private Point3[] upperCircleBackUp, lowerCircleBackUp, pPirBackUp = new Point3[200];

private Rebro[] verticalRebraBackUp, horizontalRebraUpperBackUp, horizontalRebraLowerBackUp = new Rebro[400];

PointF Zero = new PointF(250, 250);

private Graphics gr;

double cilRad, cilVis, pirRad;

double alpha;

int n;

public MyDraw(double inCirRad, double inCilVis, double inPirRad, int inN, Graphics inGr, PointF inZero)

{

Zero = inZero;

cilRad = inCirRad;

cilVis = inCilVis;

pirRad = inPirRad;

n = inN; // Степень аппроксимирования

alpha = ((360 / n)\* Math.PI) / 180;

gr = inGr;

//temps

int i;

//Prpbably cleaning variables

upperCircle = new Point3[n];

lowerCircle = new Point3[n];

upperCircleBackUp = new Point3[n];

lowerCircleBackUp = new Point3[n];

verticalRebra = new Rebro[n];

horizontalRebraUpper = new Rebro[n+2];

horizontalRebraLower = new Rebro[n+2];

verticalRebraBackUp = new Rebro[n];

horizontalRebraLowerBackUp = new Rebro[n + 2];

horizontalRebraUpperBackUp = new Rebro[n + 2];

gCil = new Gran[n+2];

gPir = new Gran[4];

gTemp = new Gran[1];

//Coords Cilindra

for (i = 0; i < gCil.Length; i++)

gCil[i] = new Gran(Color.Blue);

for (i = 0; i < gPir.Length; i++)

gPir[i] = new Gran(Color.Blue);

//Points

for (i = 0; i < n; i++)

{

upperCircle[i].X = cilRad \* Math.Cos(alpha \* i);

upperCircle[i].Y = cilVis;

upperCircle[i].Z = cilRad \* Math.Sin(alpha \* i);

lowerCircle[i].X = cilRad \* Math.Cos(alpha \* i);

lowerCircle[i].Y = 0;

lowerCircle[i].Z = cilRad \* Math.Sin(alpha \* i);

}

//may be mistake in calculation of pir's coords lower gran

pPir[0].X = 0; pPir[0].Y = cilVis; pPir[0].Z = 0;

pPir[1].X = pirRad; pPir[1].Y = 0; pPir[1].Z = 0;

pPir[2].X = ((double)pirRad/(-3)); pPir[2].Y = 0; pPir[2].Z = (-pirRad\*Math.Sqrt(3))/2;

pPir[3].X = (double)pirRad/(-3); pPir[3].Y = 0; pPir[3].Z = ( pirRad\*Math.Sqrt(3))/2;

SetRebra();

SetGrani();

}

private void SetRebra()

{

int i;

for (i = 0; i < n; i++)

{

verticalRebra[i].A = upperCircle[i]; verticalRebra[i].B = lowerCircle[i];

}

for (i = 0; i < n-1; i++)

{

horizontalRebraUpper[i].A = upperCircle[i]; horizontalRebraUpper[i].B = upperCircle[i+1];

horizontalRebraLower[i].A = lowerCircle[i]; horizontalRebraLower[i].B = lowerCircle[i+1];

}

horizontalRebraUpper[n-1].A = upperCircle[0]; horizontalRebraUpper[n-1].B = upperCircle[n-1];

horizontalRebraLower[n-1].A = lowerCircle[0]; horizontalRebraLower[n-1].B = lowerCircle[n-1];

rPir[0].A = pPir[0]; rPir[0].B = pPir[1];

rPir[1].A = pPir[0]; rPir[1].B = pPir[2];

rPir[2].A = pPir[0]; rPir[2].B = pPir[3];

rPir[3].A = pPir[1]; rPir[0].B = pPir[2];

rPir[4].A = pPir[2]; rPir[0].B = pPir[3];

rPir[5].A = pPir[3]; rPir[0].B = pPir[1];

}

private void SetGrani()

{

int i;

//For piramida

for (i = 0; i < gPir.Length; i++)

gPir[i].Points.Clear();

gPir[0].Points.Add(pPir[0]); gPir[0].Points.Add(pPir[1]);

gPir[0].Points.Add(pPir[2]);

gPir[1].Points.Add(pPir[0]); gPir[1].Points.Add(pPir[2]);

gPir[1].Points.Add(pPir[3]);

gPir[2].Points.Add(pPir[0]); gPir[2].Points.Add(pPir[1]);

gPir[2].Points.Add(pPir[3]);

gPir[3].Points.Add(pPir[1]); gPir[3].Points.Add(pPir[2]);

gPir[3].Points.Add(pPir[3]);

//For cilindr

for (i = 0; i < gCil.Length; i++)

gCil[i].Points.Clear();

for (i = 0; i < n - 1; i++)

{

gCil[i].Points.Add(upperCircle[i]); gCil[i].Points.Add(upperCircle[i+1]);

gCil[i].Points.Add(lowerCircle[i + 1]); gCil[i].Points.Add(lowerCircle[i]);

}

gCil[n - 1].Points.Add(upperCircle[n - 1]); gCil[n-1].Points.Add(upperCircle[0]);

gCil[n - 1].Points.Add(lowerCircle[0]); gCil[n - 1].Points.Add(lowerCircle[n - 1]);

//Add horizontal grans:( dot fogot to add!!!! hello

for (i = 0; i <n; i++)

{

gCil[n].Points.Add(upperCircle[i]);

}

//lower circle begin

gCil[n + 1].Points.Add(lowerCircle[0]);

gCil[n + 1].Points.Add(pPir[1]);

gCil[n + 1].Points.Add(pPir[2]);

gCil[n + 1].Points.Add(pPir[3]);

gCil[n + 1].Points.Add(pPir[1]);

for (i = 0; i < n; i++)

{

gCil[n].Points.Add(upperCircle[i]);

}

//lower circle end

}

public void Draw()

{

Color col = Color.White;

gr.Clear(col);

col = Color.Black;

//Needs to add some "if" about visuable gran's

DrawObject(gPir, col);

DrawObject(gCil, col);

/\*if (lowerCircle.Length > 0)

{

gTemp[0].Points.Add(lowerCircle[0]);

gTemp[0].Points.Add(pPir[1]);

gTemp[0].Points.Add(lowerCircle[0]);

col = Color.White;

DrawObject(gTemp, col);

}\*/

}

private void DrawObject(Gran[] obj, Color col)

{

foreach (Gran gran in obj)

{

#region commedted didn't finish and lagged

/\*if (isInvDel)

{

gran.Color = isProj ? gran.Color : gran.ColorFill;

if (IsVisible(gran))

{

List<PointF> lP = new List<PointF>();

foreach (Point3 point3 in gran.Points)

{

lP.Add(new PointF(Zero.X + (float)point3.X,

Zero.Y - (float)point3.Y));

}

if (isColor)

{

if (isLight)

gr.FillPolygon(new SolidBrush(gran.Color), lP.ToArray());

else

gr.FillPolygon(new SolidBrush(gran.ColorFill), lP.ToArray());

}

else

gr.FillPolygon(new SolidBrush(Color.White), lP.ToArray());

if (!isColor || !isLight)

gr.DrawPolygon(new Pen(Color.Black, 1), lP.ToArray());

}

}

else\*/

#endregion

//{

List<PointF> lP = new List<PointF>();

foreach (Point3 point3 in gran.Points)

{

lP.Add(new PointF(Zero.X + (float)point3.X,

Zero.Y - (float)point3.Y));

}

gr.DrawPolygon(new Pen(col, 1), lP.ToArray());

//}

}

}

#region Peremeschenie

public void MoveAll(int disX, int disY, int disZ)

{

int i;

Point3[] tmpP;

tmpP = Move(disX, disY, disZ, pPir);

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

pPir[i] = tmpP[i];

tmpP = Move(disX, disY, disZ, upperCircle);

for (i = 0; i < n; i++)

upperCircle[i] = tmpP[i];

tmpP = Move(disX, disY, disZ, lowerCircle);

for (i = 0; i < n; i++)

lowerCircle[i] = tmpP[i];

SetRebra();

SetGrani();

Draw();

}

private Point3[] Move(int disX, int disY, int disZ, Point3[] point3s)

{

int i;

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = 1; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 1; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = 1; R[2, 3] = 0;

R[3, 0] = disX; R[3, 1] = disY; R[3, 2] = disZ; R[3, 3] = 1;

for (i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

#endregion

#region Angle

public void RotateAll(int angleX, int angleY, int angleZ)

{

Point3[] tmpP1, tmpP2;

tmpP1 = RotateX(angleX, pPir);

tmpP2 = RotateY(angleY, tmpP1);

pPir = RotateZ(angleZ, tmpP2);

tmpP1 = RotateX(angleX, upperCircle);

tmpP2 = RotateY(angleY, tmpP1);

upperCircle = RotateZ(angleZ, tmpP2);

tmpP1 = RotateX(angleX, lowerCircle);

tmpP2 = RotateY(angleY, tmpP1);

lowerCircle = RotateZ(angleZ, tmpP2);

SetRebra();

SetGrani();

Draw();

}

private Point3[] RotateX(int ang, Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

double radians = Math.PI \* ang / 180;

Matrix R = new Matrix(4, 4);

R[0, 0] = 1;

R[0, 1] = 0;

R[0, 2] = 0;

R[0, 3] = 0;

R[1, 0] = 0;

R[1, 1] = Math.Cos(radians);

R[1, 2] = Math.Sin(radians);

R[1, 3] = 0;

R[2, 0] = 0;

R[2, 1] = -Math.Sin(radians);

R[2, 2] = Math.Cos(radians);

R[2, 3] = 0;

R[3, 0] = 0;

R[3, 1] = 0;

R[3, 2] = 0;

R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

private Point3[] RotateY(int ang, Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

double radians = Math.PI \* ang / 180;

Matrix R = new Matrix(4, 4);

R[0, 0] = Math.Cos(radians);

R[0, 1] = 0;

R[0, 2] = -Math.Sin(radians);

R[0, 3] = 0;

R[1, 0] = 0;

R[1, 1] = 1;

R[1, 2] = 0;

R[1, 3] = 0;

R[2, 0] = Math.Sin(radians);

R[2, 1] = 0;

R[2, 2] = Math.Cos(radians);

R[2, 3] = 0;

R[3, 0] = 0;

R[3, 1] = 0;

R[3, 2] = 0;

R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

private Point3[] RotateZ(int ang, Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

double radians = Math.PI \* ang / 180;

Matrix R = new Matrix(4, 4);

R[0, 0] = Math.Cos(radians);

R[0, 1] = Math.Sin(radians);

R[0, 2] = 0;

R[0, 3] = 0;

R[1, 0] = -Math.Sin(radians);

R[1, 1] = Math.Cos(radians);

R[1, 2] = 0;

R[1, 3] = 0;

R[2, 0] = 0;

R[2, 1] = 0;

R[2, 2] = 1;

R[2, 3] = 0;

R[3, 0] = 0;

R[3, 1] = 0;

R[3, 2] = 0;

R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

#endregion

#region Scale

public void ScaleAll(double scX, double scY, double scZ)

{

Point3[] tmpP;

tmpP = Scale(scX, scY, scZ, pPir);

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

pPir[i] = tmpP[i];

tmpP = Scale(scX, scY, scZ, upperCircle);

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

upperCircle[i] = tmpP[i];

tmpP = Scale(scX, scY, scZ, lowerCircle);

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

lowerCircle[i] = tmpP[i];

SetRebra();

SetGrani();

Draw();

}

private Point3[] Scale(double scX, double scY, double scZ, Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = scX; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = scY; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = scZ; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

#endregion

#region BackUpsFunction

private void SetBackup()

{

//gtemp add backup

for (int i = 0; i < pPir.Length; i++)

pPirBackUp[i] = pPir[i];

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircleBackUp[i] = lowerCircle[i];

for (int i = 0; i < upperCircle.Length; i++)

upperCircleBackUp[i] = upperCircle[i];

for (int i = 0; i < verticalRebra.Length; i++)

verticalRebraBackUp[i] = verticalRebra[i];

for (int i = 0; i < horizontalRebraUpper.Length; i++)

horizontalRebraUpperBackUp[i] = horizontalRebraUpper[i];

for (int i = 0; i < horizontalRebraLower.Length; i++)

horizontalRebraLowerBackUp[i] = horizontalRebraLower[i];

}

private void GetBackup()

{

//may be mistake in length, if something happens

//gtemp add backup

for (int i = 0; i < pPir.Length; i++)

pPir[i] = pPirBackUp[i];

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircle[i] = lowerCircleBackUp[i];

for (int i = 0; i < upperCircle.Length; i++)

upperCircle[i] = upperCircleBackUp[i];

for (int i = 0; i < verticalRebra.Length; i++)

verticalRebra[i] = verticalRebraBackUp[i];

for (int i = 0; i < horizontalRebraUpper.Length; i++)

horizontalRebraUpper[i] = horizontalRebraUpperBackUp[i];

for (int i = 0; i < horizontalRebraLower.Length; i++)

horizontalRebraLower[i] = horizontalRebraLowerBackUp[i];

}

#endregion

#region Proection

//Frontal

public void ProFrontAll()

{

SetBackup();

Point3[] tmpP, tmpP1, tmpP2 ;

tmpP = ProFront(pPir);

for (int i = 0; i < pPir.Length; i++)

pPir[i] = tmpP[i];

tmpP1 = ProFront(upperCircle);

for (int i = 0; i < upperCircle.Length; i++)

upperCircle[i] = tmpP1[i];

tmpP2 = ProFront(lowerCircle);

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircle[i] = tmpP2[i];

//lightPoint = ProFront(new Point3[] { lightPoint })[0];

SetRebra();

SetGrani();

Draw();

GetBackup();

}

private Point3[] ProFront(Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = 1; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 1; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = 0; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

//Horizontal

public void ProGorAll()

{

SetBackup();

Point3[] tmpP, tmpP1, tmpP2;

tmpP = ProGor(pPir);

for (int i = 0; i < pPir.Length; i++)

pPir[i] = tmpP[i];

tmpP1 = ProGor(upperCircle);

for (int i = 0; i < upperCircle.Length; i++)

upperCircle[i] = tmpP1[i];

tmpP2 = ProGor(lowerCircle);

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircle[i] = tmpP2[i];

SetRebra();

SetGrani();

Draw();

GetBackup();

}

private Point3[] ProGor(Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = 1; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 0; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = 1; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 2];

outMas[i].Z = outM[0, 1];

}

return outMas;

}

//Prof

public void ProProfAll()

{

SetBackup();

Point3[] tmpP, tmpP1, tmpP2;

tmpP = ProProf(pPir);

for (int i = 0; i < pPir.Length; i++)

pPir[i] = tmpP[i];

tmpP1 = ProProf(upperCircle);

for (int i = 0; i < upperCircle.Length; i++)

upperCircle[i] = tmpP1[i];

tmpP2 = ProProf(lowerCircle);

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircle[i] = tmpP2[i];

SetRebra();

SetGrani();

Draw();

GetBackup();

}

private Point3[] ProProf(Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = 0; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 1; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = 0; R[2, 2] = 1; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 2];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 0];

}

return outMas;

}

//Aksonametric

public void ProAksAll(double fi, double psi)

{

SetBackup();

Point3[] tmpP, tmpP1, tmpP2;

fi = Math.PI \* fi / 180;

psi = Math.PI \* psi / 180;

tmpP = ProAks(fi, psi, pPir);

for (int i = 0; i < pPir.Length; i++)

pPir[i] = tmpP[i];

tmpP1 = ProAks(fi, psi, upperCircle);

for (int i = 0; i < upperCircle.Length; i++)

upperCircle[i] = tmpP1[i];

tmpP2 = ProAks(fi, psi, lowerCircle);

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircle[i] = tmpP2[i];

SetRebra();

SetGrani();

Draw();

GetBackup();

}

private Point3[] ProAks(double fi, double psi, Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = Math.Cos(psi); R[0, 1] = Math.Sin(fi) \* Math.Sin(psi); R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = Math.Cos(fi); R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = Math.Sin(psi); R[2, 1] = -1 \* Math.Sin(fi) \* Math.Cos(psi); R[2, 2] = 0; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

//Kosougil

public void ProKosAll(double L, double alpfa)

{

SetBackup();

Point3[] tmpP, tmpP1, tmpP2;

alpfa = Math.PI \* alpfa / 180;

tmpP = ProKos(L, alpfa, pPir);

for (int i = 0; i < pPir.Length; i++)

pPir[i] = tmpP[i];

tmpP1 = ProKos(L, alpfa, upperCircle);

for (int i = 0; i < upperCircle.Length; i++)

upperCircle[i] = tmpP1[i];

tmpP2 = ProKos(L, alpfa, lowerCircle);

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircle[i] = tmpP2[i];

SetRebra();

SetGrani();

Draw();

GetBackup();

}

private Point3[] ProKos(double L, double alfa, Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = 1; R[0, 1] = 0; R[0, 2] = 0; R[0, 3] = 0;

R[1, 0] = 0; R[1, 1] = 1; R[1, 2] = 0; R[1, 3] = 0;

R[2, 0] = L \* Math.Cos(alfa); R[2, 1] = L \* Math.Sin(alfa); R[2, 2] = 0; R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = 0; R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

outMas[i].X = outM[0, 0];

outMas[i].Y = outM[0, 1];

outMas[i].Z = outM[0, 2];

}

return outMas;

}

//Perspect

public void ProPerAll(double d, double tetta, double fi, double ro)

{

SetBackup();

Point3[] tmpP, tmpP1, tmpP2;

fi = Math.PI \* fi / 180;

tetta = Math.PI \* tetta / 180;

tmpP = ProPer(d, tetta, fi, ro, pPir);

for (int i = 0; i < pPir.Length; i++)

pPir[i] = tmpP[i];

tmpP1 = ProPer(d, tetta, fi, ro, lowerCircle);

for (int i = 0; i < lowerCircle.Length; i++)

lowerCircle[i] = tmpP1[i];

tmpP2 = ProPer(d, tetta, fi, ro, upperCircle);

for (int i = 0; i < upperCircle.Length; i++)

upperCircle[i] = tmpP2[i];

SetRebra();

SetGrani();

Draw();

GetBackup();

}

private Point3[] ProPer(double d, double tetta, double fi, double ro, Point3[] point3s)

{

Point3[] outMas = new Point3[point3s.Length];

Matrix R = new Matrix(4, 4);

R[0, 0] = Math.Cos(tetta); R[0, 1] = -Math.Cos(fi) \* Math.Sin(tetta); R[0, 2] = -Math.Sin(fi) \* Math.Sin(tetta); R[0, 3] = 0;

R[1, 0] = Math.Sin(tetta); R[1, 1] = Math.Cos(fi) \* Math.Cos(tetta); R[1, 2] = Math.Sin(fi) \* Math.Cos(tetta); R[1, 3] = 0;

R[2, 0] = 0; R[2, 1] = Math.Sin(fi); R[2, 2] = -Math.Cos(fi); R[2, 3] = 0;

R[3, 0] = 0; R[3, 1] = 0; R[3, 2] = ro; R[3, 3] = 1;

for (int i = 0; i < point3s.Length; i++)

{

Matrix s = new Matrix(1, 4);

s[0, 0] = point3s[i].X;

s[0, 1] = point3s[i].Y;

s[0, 2] = point3s[i].Z;

s[0, 3] = 1;

Matrix outM = Matrix.Multiply(s, R);

if (outM[0, 2] == 0)

{

outM[0, 2] = 0.1;

}

outMas[i].X = outM[0, 0] \* d / outM[0, 2];

outMas[i].Y = outM[0, 1] \* d / outM[0, 2];

outMas[i].Z = outM[0, 2] \* d / outM[0, 2];

}

return outMas;

}

#endregion

}

}