Федеральное государственное автономное
образовательное учреждение
высшего профессионального образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий

Кафедра “Системы искусственного интеллекта”

УТВЕРЖДАЮ
Заведующий кафедрой

_____ _____
подпись инициалы, фамилия
« ____» _____ 20 __ г.

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

Удаление невидимых поверхностей с помощью алгоритма Z-буфер

Студент, КИ 08-05 _____ О.В. Дрозд
номер группы подпись, дата инициалы, фамилия
Преподаватель _____ Э.И. Сиротин
подпись, дата инициалы, фамилия

Красноярск 2011

Цель:

Научиться строить параметрически заданные полигональные поверхностные модели простейших трехмерных объектов, и осуществлять их визуализацию с удалением невидимых поверхностей с помощью алгоритма Z-буфер.

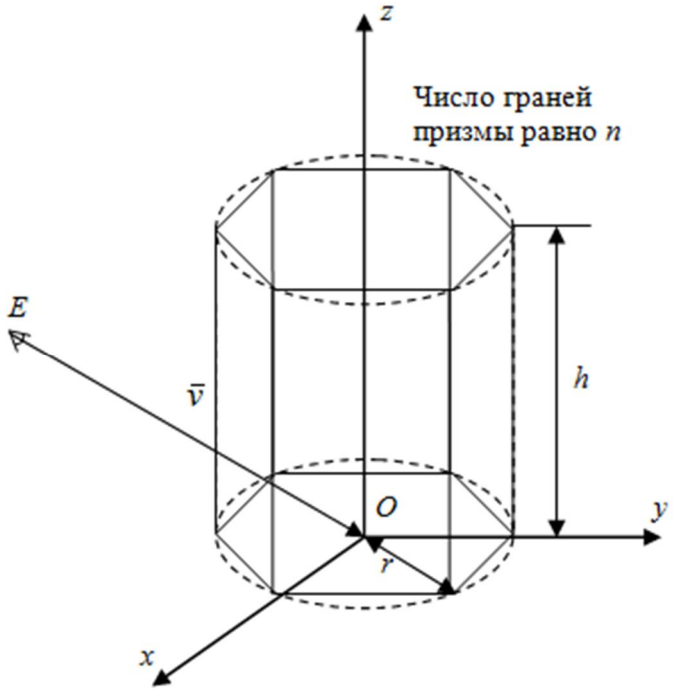
Задача:

На основе исходных параметров построить полигональную поверхностную модель трехмерного объекта, и визуализировать полученную сцену, используя параллельное и перспективное проецирования и удаление невидимых поверхностей с помощью алгоритма Z-буфер.

Вариант задания:

Необходимо взять трехмерную сцену из лабораторной работы №3, и визуализировать ее, используя алгоритм удаления невидимых поверхностей z-буфер.

Объектом является цилиндр, аппроксимируемый правильной *n*-гранной призмой. Основания цилиндра параллельны плоскости *xy*. Нижнее основание лежит в плоскости *xy*. Центр нижнего основания совпадает с точкой *O* начала мировой системы координат. Число граней призмы *n*, радиус описанной вокруг основания окружности *r*, высота призмы *h*, координаты точки наблюдения *E* и расстояние до экрана *d* задаются пользователем. Вектор наблюдения $\bar{v}$ направлен в точку *O*. Необходимо построить параллельную и перспективную проекции данной сцены.



Теоретически е сведения:

Закраска произвольного треугольника, (алгоритм 1)

Рассмотрим изображение треугольника на экране. Оно представляет собой множество точек, упорядоченных по вертикали и горизонтали рис. 1.

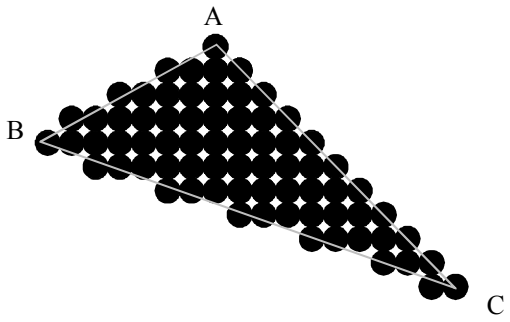


Рисунок 1. Попиксельное представление закрашенного треугольника

Будем отрисовывать треугольник последовательно, точка за точкой сверху вниз и слева направо. Для этого надо пройти по всем строкам экрана, которые пересекают треугольник, (т.е от минимального до максимального значения *y*- координат вершин треугольника) и нарисовать соответствующие горизонтальные отрезки (рис. 2).

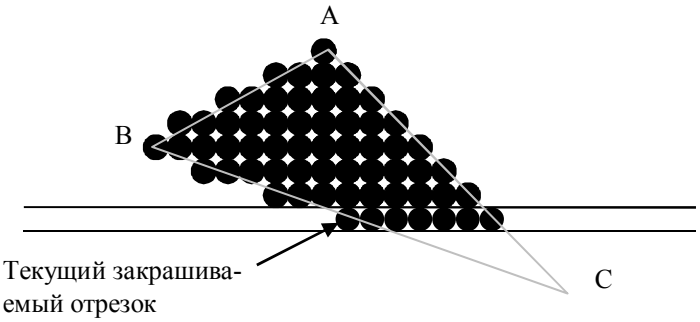


Рисунок 2. Последовательная закраска треугольника.

Для этого, сначала отсортируем вершины треугольника так, чтобы А была верхней а С – нижней. Тогда, $y_{\min} = Ay$, а $y_{\max} = Cy$, и нам нужно пройтись по всем строкам от Ay до Cy. Рассмотрим некоторую строку sy, $Ay \le sy \le Cy$. Если $sy < By$, то эта строка пересекает стороны АВ и АС. Если $By \le sy \le Cy$, то строка пересекает ВС и АС.

Поскольку нам известны координаты всех вершин треугольника, мы можем, написав уравнения прямых, содержащих стороны треугольника, найти точки пересечения прямой $y = sy$ со сторонами треугольника. Нарисуем горизонтальный отрезок, соединяющий эти точки, и повторим для всех строк от Ay до Cy. Так мы нарисуем весь треугольник.

Рассмотрим подробно нахождение точки пересечения прямой $y = sy$ и стороны треугольника, например AB. Для этого напишем уравнение прямой AB в виде $x = ky + b$:

$$x = \frac{Bx - Ax}{By - Ay}(y - Ay) + Ax$$

Подставим в него известное значение $y = sy$:

$$x = \frac{Bx - Ax}{By - Ay}(sy - Ay) + Ax$$

Для других сторон точки пересечения ищутся аналогично. Так для BC:

$$x = \frac{Cx - Bx}{Cy - By}(sy - By) + Bx$$

Для AC:

$$x = \frac{Cx - Ax}{Cy - Ay}(sy - Ay) + Ax$$

Необходимо отслеживать случаи $By = Cy$, $By = Ay$ и $Ay = Cy$, поскольку будет происходить деление на 0.

Закраска произвольного треугольника (алгоритм 2)

Рассмотренный ранее алгоритм позволяет закрасить любой треугольник, однако его быстродействие оставляет желать лучшего. В этом повинны как поточечное рисование на поверхности окна (поскольку оно чрезвычайно медленное), так и использование операций с плавающей точкой над вещественными числами. От поточечного рисования нам избавиться не удастся, поскольку необходимо иметь возможность задавать значение цвета для каждой точки треугольника. Однако API Windows предлагает метод, при использовании которого, поточечное рисование может быть ускорено в несколько раз.

Суть его заключается в том, чтобы создать в памяти битовую карту изображения, нарисовать треугольник в ней, и потом эту карту отобразить на поверхности окна. Поскольку рисование будет производиться в оперативной памяти, а не непосредственно на экране, то быстродействие алгоритма существенно возрастает.

Все необходимые средства для работы с битовой картой предоставляет класс TBitmap. Прежде чем начать работать с битовой картой, необходимо создать его экземпляр, и задать размеры битовой карты, и формат пикселей (т.е. глубину цвета). Для этого необходимо сделать следующие действия:

```
CBitmap * MyBitMap = new CBitmap();
MyBitMap->CreateCompatibleBitmap(myDC,800,600);

CDC * MyCDC = new CDC();
MyCDC->CreateCompatibleDC(myDC);
CBitmap* pOldBitmap = MyCDC->SelectObject(MyBitMap)
```

Поточечное рисование на битовой карте осуществляется с помощью строк сканирования. Вся битовая карта представляется как одномерный массив горизонтальных строк, называемых строками сканирования. Каждая из этих строк представляет собой одномерный массив, каждый элемент которого для глубины цвета 32 бита, содержит значение цвета соответствующей точки. Это значение представляет собой 32 разрядное целое число, в котором первый байт задает интенсивность синего цвета, второй байт – зеленого, третий байт – красного, а четвертый – значение прозрачности (канал альфа).

После того, как необходимость в битовой карте пропадет, необходимо уничтожить ее экземпляр.

```
DeleteObject(MyCDC);
DeleteObject(myDC);
MyBitMap->DeleteObject()
```

Кроме использования битовых карт, можно дополнительно увеличить быстродействие алгоритма, отказавшись от использования вещественных чисел и операций с плавающей точкой.

Оказывается, можно разработать другой алгоритм закраски треугольника, в котором используются только целые числа, и отсутствует операция деления.

Сначала представим стороны треугольника в виде отрезков прямых линий (см. рис. 3).

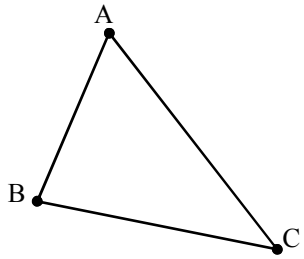


Рисунок 3. Представление сторон треугольника в виде набора отрезков.

Поскольку изображение на экране состоит из совокупности точек, то и изображения этих отрезков также будет состоять из точек. Если рассмотреть с увеличением окрестность вершины А, можно увидеть примерно следующую картину (см. рис. 4.).

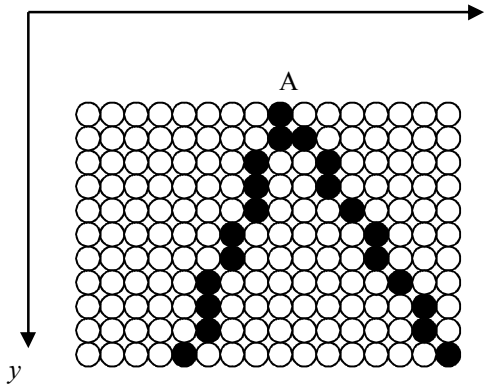


Рисунок 4. Вид окрестности точки А.

Значит, если мы сможем разработать алгоритм последовательного нахождения координат точек отрезков A B, A C, B C, то сможем нарисовать закрашенный треугольник просто рисуя горизонтальные отрезки соединяющие точки двух сторон с одинаковыми координатами *y* (см. рис. 6.).

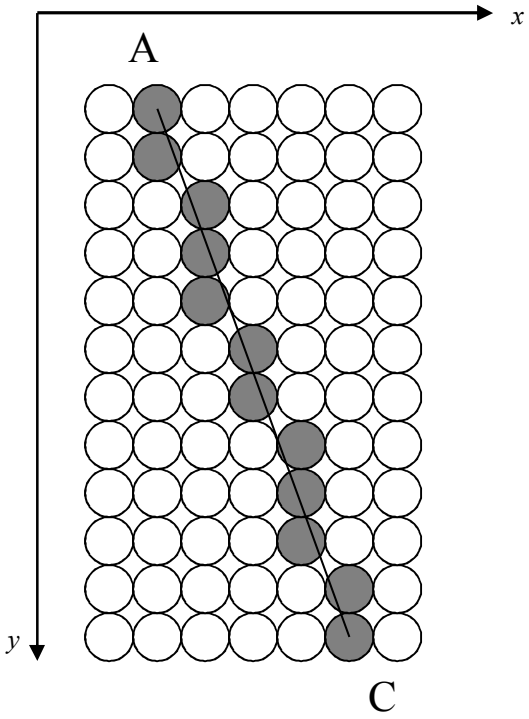


Рисунок 6. Выбор точек отрезка AC.

```
for (i=0; i<kol_t/2; i++)
      {
            if (i!=kol_t/2-1)
            {
                  //основание
                  t[i].SetA(b[n+1].x, b[n+1].y, b[n+1].z, 1);
                  t[i].SetB(b[i].x, b[i].y, b[i].z, 1);
                  t[i].SetC(b[i+1].x, b[i+1].y, b[i+1].z, 1);

                  //острие
                  t[i+kol_t/2].SetA(b[n].x, b[n].y, b[n].z, 1);
                  t[i+kol_t/2].SetB(b[i].x, b[i].y, b[i].z, 1);
                  t[i+kol_t/2].SetC(b[i+1].x, b[i+1].y, b[i+1].z, 1);
            }
            else
            {
                  //основание
                  t[i].SetA(b[n+1].x, b[n+1].y, b[n+1].z, 1);
                  t[i].SetB(b[i].x, b[i].y, b[i].z, 1);
                  t[i].SetC(b[0].x, b[0].y, b[0].z, 1);

                  //острие
                  t[i+kol_t/2].SetA(b[n].x, b[n].y, b[n].z, 1);
                  t[i+kol_t/2].SetB(b[i].x, b[i].y, b[i].z, 1);
                  t[i+kol_t/2].SetC(b[0].x, b[0].y, b[0].z, 1);
            }
      }
```

Это означает, что расстояние по горизонтали между новой выбранной точкой и этой линией не должно превышать значения 0.5.

Введем переменную *d* для обозначения этого расстояния. Потребуем, чтобы $-0.5 < d \le 0.5$.

Последнее неравенство обеспечивает условие для определения необходимости давать приращение переменной *x*.

Отклонение, вначале устанавливается равным нулю и изменяется на каждом шаге цикла. Поскольку оно показывает, насколько левее точной прямой линии лежит вычисленная точка, то значение *d* увеличивается на значение наклона, если *y* увеличивается на 1 и *x* остается без изменения. Это условие не выполняется, если значение *d* превышает 0.5. В этот момент нужно увеличить значение *x* на 1. Соответственно, и отклонение *d* должно быть уменьшено на единицу.

Теперь посмотрим, как можно избавиться от вещественных значений переменных *t* и *d*.

Значение переменной *t* вычисляется следующим образом:

$$t = \frac{CX - AX}{CY - AY}$$

где числитель и знаменатель представляют собой целые числа.

Величина *d* вычисляется как конечная сумма элементов, каждый из которых равен либо *t*, либо -1. Поэтому *d* также можно записать в виде частного со знаменателем равным $CY - AY$.

Значит, можно перейти к целочисленным переменным *t* и *d*, путем умножения на значение знаменателя. Также просто избавиться от константы 0.5. Для этого нужно дополнительно умножить значение знаменателя на 2. Таким образом $t = 2lx$, где *lx=CX-AX*.

Однако наша функция будет правильно работать только для случая, когда $lx \le ly$, и просто поменять местами *x* и *y* в противном случае нам не удастся (треугольник должен отрисовываться последовательно сверху – вниз).

Рассмотрим следующий рисунок:

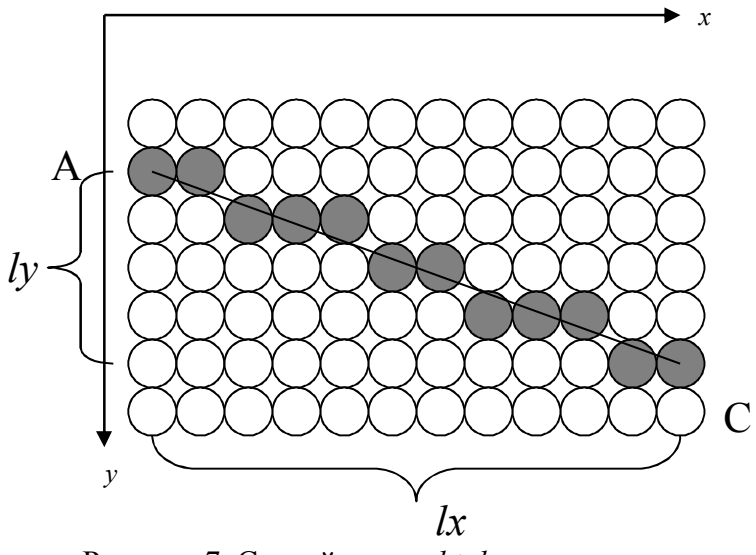


Рисунок 7. Случай, когда *lx>ly*.

На нем хорошо видно, что в этом случае одному значению y может соответствовать несколько значений x. Но нас интересует из этих значений лишь максимальное. А это значит, что значение отклонения от прямой линии в этой точке будет максимальное, но не превысит 0.5. Применяя тот же подход что был рассмотрен выше, и эти соображения можно написать следующий вариант функции для данного случая:

```
for (sy=t.A.y+1; sy<=t.C.y; sy++)
      {
                              x1=k1*(sy-t.A.y)+t.A.x;//поиск пересечения с AC
            if (sy<=t.B.y)    x2=k2*(sy-t.A.y)+t.A.x;//поиск пересечения с AB
            else              x2=k3*(sy-t.B.y)+t.B.x;//поиск пересечения с BC
            if (x1>x2)
            {
                  for (tmp=x2; tmp<=x1; tmp++)
                  {
                        if ((tmp<win_width)
                                          && (tmp>=0)
                                          && (sy<win_height)
                                          && (sy>=0))
                        {
                              if (Prov(t, tmp, sy, z))
                                    mDC->SetPixel(tmp, sy, Colour);
                        }
                  }
            }
            else
            {
                  for (tmp=x1; tmp<=x2; tmp++)
                  {
                        if ((tmp<win_width)
                                          && (tmp>=0)
                                          && (sy<win_height)
                                          && (sy>=0))
                        {
                              if (Prov(t, tmp, sy, z))
                                    mDC->SetPixel(tmp, sy, Colour);
                        }
                  }
            }
      }
      }
```

Для других сторон треугольника координаты точек прямых линий ищутся аналогично.

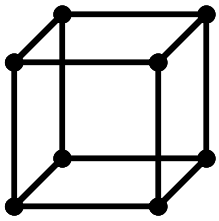
Обнуление Z буфера

```
for (i=0; i<600; i++)//обнуление Z-буфера
      for (j=0; j<800; j++)
            z[i][j] = 0;
```

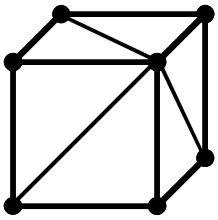
Поверхностная полигональная модель трехмерного объекта и удаление невидимых поверхностей

В данной лабораторной работе объект рассматривается как сплошная поверхность, состоящая из множества граней. Для простоты вычислений будем представлять каждую грань треугольником. Т.е. каждая грань задается координатами трех вершин треугольника. Треугольник выбран потому, что

это выпуклая фигура, все точки которой всегда лежат в одной плоскости. Если нарисовать после видового и перспективного преобразования все грани объекта (все треугольники) по аналогии с рисованием каркасной модели, то результат окажется не тем, который ожидается. Вместо изображения объекта мы получим хаотическое нагромождение треугольников, в котором разглядеть объект будет крайне сложно. Это происходит потому, что теперь объект представляет собой сплошное тело, а это значит, что те грани, которые находятся ближе к наблюдателю, заслоняют собой грани находящиеся дальше от него рис. 8.б.



а) Каркасная модель объекта



б) Поверхностная модель объекта с аппроксимацией поверхности треугольными гранями

Рисунок 8. Различные модели представления объекта.

Следовательно, нам необходимо в процессе рисования граней нарисовать только те точки поверхности граней, которые видны наблюдателю. Такая задача решается с помощью алгоритмов удаления невидимых поверхностей. В данной лабораторной работе необходимо использовать алгоритм под названием Z-буфер.

Алгоритм удаления невидимых поверхностей Z-буфер

уть этого алгоритма заключается в построении рельефа по глубине наблюдаемой сцены, и использования этой информации для рисования только видимых точек поверхности объекта.

Информацию о глубине точек поверхности объекта несет координата *Z* этих точек, так как ось *Z* после преобразований у нас направлена в глубь экрана, и совпадает с направлением наблюдения.

Заведем двумерный массив (собственно Z-буфер) размером с экран, и заполним все его элементы каким-то большим числом, настолько большим, что координаты *z* для всех точек сцены заведомо меньше. Для каждой рисуемой точки подсчитаем значение ее координаты *z*. Если оно больше, чем значение в *Z*-буфере (точка закрыта какой-то другой точкой), или меньше, чем 0 (точка находится за камерой), то переходим к следующей точке. Если меньше, то рисуем точку на экране, а в *z*-буфер записываем значение координаты *z* текущей точки.

Имеет смысл считать значения не z, а $w = 1/z$. Тогда условия чуть изменяются – точка загорожена другой, если значение w меньше значения в Z-буфере; и точка находится за камерой, если w < 0. Буфер инициализируем нулями. Тогда не нужна проверка на положительность w - точка попадает в Z-буфер и на экран, только, если w больше текущего значения, и поэтому точки, для которых $w < 0$ в буфер никогда не попадут.

Для рисования граней объекта пригодны алгоритмы закрашивания по точкам многоугольников, рассмотренные выше. Единственное, что необходимо в них добавить, это работу с Z-буфером и расчет координаты Z для текущей точки поверхности треугольника по ее уже рассчитанным координатам X и Y и координатам трех вершин треугольника.

Этапы решения задачи

Сгенерировать 2 n-угольника радиуса a. Все точки первого многоугольника будут иметь координату z = 0, все точки второго z = h. Генерацию правильного многоугольника будем осуществлять следующим образом. Как известно, сумма углов правильного n-угольника *S = 180 * (n – 2).* Тогда каждый угол *φ = 180 * (n – 2) / n.* В таком случае, *i*-я вершина многоугольника *pi, (0 <=i < n)* будет иметь координаты *pi = (acos(φi), asin(φi), 0, 1).*

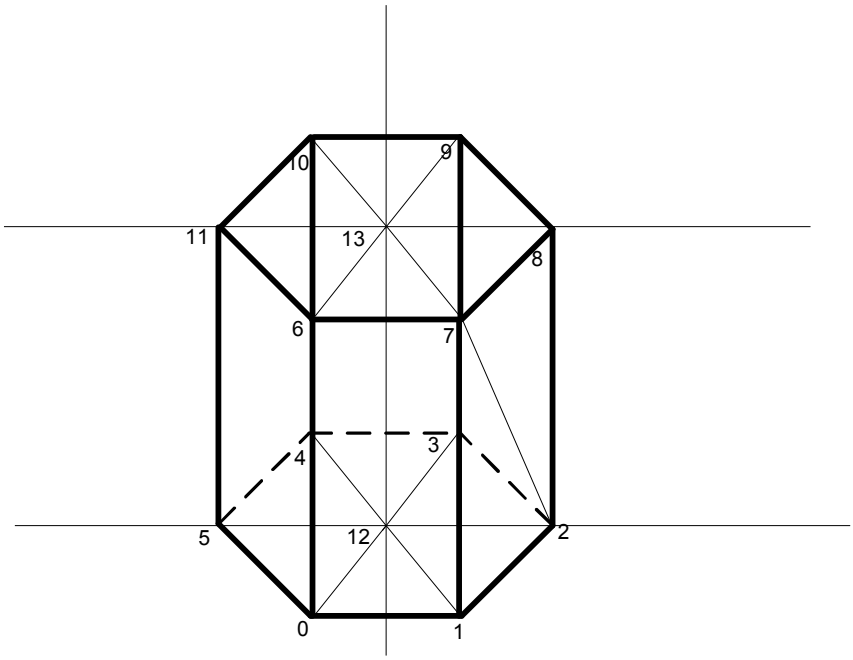
Далее необходимо сгенерировать 4 * n треугольников. Пронумеруем вершины призмы следующим образом: 0, 1, …, n – 1 – вершины нижнего основания призмы, n, n + 1, …, 2 * n – 1 – вершины верхнего основания призмы, точка (0, 0, 0) имеет номер 2 * n, а (0, 0, h) – 2 * n + 1.

Каждый треугольник представляет собой тройку (i, j, k), где i, j, k – номера вершин. Сначала сгенерируем n треугольников вида (2 * n, 0, 1), (2 * n, 2, 3) и так далее до (2 * n, n – 1, 1).

Аналогично генерируются треугольники вида (2 * n + 1, n, n + 1), (2 * n + 1, n + 1, n + 2) и так далее до (2 * n + 1, 2 * n – 1, n). Вышеуказанные треугольники «замощают» основания призмы.

Затем сгенерируем n пар треугольников, представляющих боковые грани призмы. Они будут иметь вид (0, 1, n) | (n, n + 1, 1), (1, 2, n + 1) | (n + 1, n + 2, 2) и так далее до (n – 1, 0, 2 * n – 1) | (2 * n – 2, 2 * n – 1, 0).

Ниже представлен пример.



Отображение проекции многогранника на экран. Для этого необходимо произвести видовое преобразование над всеми вершинами многогранника. (см. лабораторная работа №3).

Цвет закраски граней выбирается рандомно. Цвет представляется в терминах каналов альфа, красного, зеленого и синего (ARGB). Значения Red, Green, Blue выбираются случайно в диапазоне от 55 до 255.

```
        Color color = Color.FromArgb(alpha, random.Next(55, 255), random.Next(55, 255), random.Next(55, 255));
```

Закраска треугольника проводится по алгоритму №2. Закрашенный треугольник представляется в виде совокупности горизонтальных отрезков, соединяющих точки двух сторон с одинаковыми координатами *y*. После выполнения видового преобразования необходимо растеризовать и закрасить треугольники и выполнить удаление невидимых поверхностей при помощи Z-буфера.

Матрицы геометрических перобразований (Matrix.cs):

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Lab4
{
    class Matrix
    {
        public static double[] VectorMatrixMlp(double[] vector, double[][] matrix, int n)
        {
            double[] ret = new double[n];
            for (int i = 0; i < n; i++)
            {
```

```
                ret[i] = 0;
                for (int j = 0; j < n; j++)
                    ret[i] += vector[j] * matrix[j][i];
            }
            return ret;
        }
        public static void MatrixMatrixMlp(double[][] matrix1, double[][]
matrix2, int n, double[][] ret)
        {
            for (int i = 0; i < n; i++)
            {

                for (int j = 0; j < n; j++)
                {
                    ret[i][j] = 0;
                    for (int k = 0; k < n; k++)
                        ret[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }
        public static void Make3DRotationZ(double cos, double sin, double[][]
ret)
        {
            ret[0][0] = cos;
            ret[0][1] = sin;
            ret[0][2] = ret[0][3] = 0;
            ret[1][0] = -ret[0][1];
            ret[1][1] = ret[0][0];
            ret[1][2] = ret[1][3] = 0;
            ret[2][0] = ret[2][1] = ret[2][3] = 0;
            ret[2][2] = ret[3][3] = 1;
            ret[3][0] = ret[3][1] = ret[3][2] = 0;
        }
        public static void Make3DRotationY(double cos, double sin, double[][]
ret)
        {
            ret[0][0] = cos;
            ret[0][1] = 0;
            ret[0][2] = -sin;
            ret[0][3] = 0;
            ret[1][0] = 0;
            ret[1][1] = 1;
            ret[1][2] = 0;
            ret[1][3] = 0;
            ret[2][0] = -ret[0][2];
            ret[2][1] = 0;
            ret[2][2] = ret[0][0];
            ret[2][3] = 0;
            ret[3][0] = ret[3][1] = ret[3][2] = 0;
            ret[3][3] = 1;
        }
        public static void Make3DScaling(double sx, double sy, double sz,
double[][] ret)
        {
            ret[0][0] = sx;
            ret[1][1] = sy;
            ret[2][2] = sz;
            ret[0][1] = ret[0][2] = ret[0][3] = 0;
            ret[1][0] = ret[1][2] = ret[1][3] = 0;
            ret[2][0] = ret[2][1] = ret[2][3] = 0;
            ret[3][3] = 1;
        }
        public static void Make3DTranslation(double dx, double dy, double dz,
double[][] ret)
```

```
        {
            ret[0][0] = ret[1][1] = ret[2][2] = ret[3][3] = 1;
            ret[0][1] = ret[0][2] = ret[0][3] = 0;
            ret[1][0] = ret[1][2] = ret[1][3] = 0;
            ret[2][0] = ret[2][1] = ret[2][3] = 0;
            ret[3][0] = dx;
            ret[3][1] = dy;
            ret[3][2] = dz;
        }
    }
}
```

Структура описания треугольника (Triangle.cs):

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Lab4
{
    struct Triangle
    {
        int[] vertices;
        public Triangle(int i1, int i2, int i3)
        {
            vertices = new int[3];
            vertices[0] = i1;
            vertices[1] = i2;
            vertices[2] = i3;
            Array.Sort(vertices);
        }
        public static bool operator !=(Triangle a, Triangle b)
        {
            return !(a == b);
        }
        public static bool operator == (Triangle a, Triangle b)
        {
            for (int i = 0; i < 3; i++)
                if (a.vertices[i] != b.vertices[i])
                    return false;
            return true;
        }
        public int GetI(int i)
        {
            return vertices[i];
        }
    }
}
```

UniformPoint.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Lab4
{
    public struct UniformPoint
    {
        double[] coordinates;
        public UniformPoint(UniformPoint p)
        {
```

```
        coordinates = new double[4];
        p.coordinates.CopyTo(coordinates, 0);
    }
    public UniformPoint(double x, double y, double z, double w)
    {
        coordinates = new double[4];
        coordinates[0] = x;
        coordinates[1] = y;
        coordinates[2] = z;
        coordinates[3] = w;
    }
    public void Shift(double dx, double dy, double dz)
    {
        coordinates[0] += dx;
        coordinates[1] = dy - coordinates[1];
        coordinates[2] += dz;
    }
    public void Transform(double[][] t, ref UniformPoint p)
    {
        p.coordinates = Matrix.VectorMatrixMlp(coordinates, t, 4);
    }
    public void Transform(double[][] t)
    {
        coordinates = Matrix.VectorMatrixMlp(coordinates, t, 4);
    }
    public float GetX()
    {
        return (float)coordinates[0];
    }
    public float GetY()
    {
        return (float)coordinates[1];
    }
    public float GetZ()
    {
        return (float)coordinates[2];
    }
    public int GetIX()
    {
        return (int)Math.Round(coordinates[0]);
    }
    public int GetIY()
    {
        return (int)Math.Round(coordinates[1]);
    }
    public int GetIZ()
    {
        return (int)Math.Round(coordinates[2]);
    }
    public void MakePerspectiveTransformation(double d)
    {
       coordinates[0] = coordinates[0] * d / coordinates[2];
       coordinates[1] = coordinates[1] * d / coordinates[2];
    }
    public static UniformPoint operator -(UniformPoint a, UniformPoint b)
    {
        UniformPoint ret = new UniformPoint(0, 0, 0, 1);
        for (int i = 0; i < 3; i++)
            ret.coordinates[i] = a.coordinates[i] - b.coordinates[i];
        return ret;
    }
    public static double Sqr(double x)
    {
        return x * x;
```

```
        }
        public double Abs()
        {
            return Math.Sqrt(Sqr(coordinates[0]) + Sqr(coordinates[1]) +
Sqr(coordinates[2]));
        }
        public static UniformPoint CrossProduct(UniformPoint a, UniformPoint
b)
        {
            double x1 = a.coordinates[0];
            double y1 = a.coordinates[1];
            double z1 = a.coordinates[2];
            double x2 = b.coordinates[0];
            double y2 = b.coordinates[1];
            double z2 = b.coordinates[2];
            return new UniformPoint(y1 * z2 - y2 * z1, z1 * x2 - x1 * z2, x1
* y2 - y1 * x2, 1);
        }
        public void Normalize()
        {
            double abs = this.Abs();
            for (int i = 0; i < 3; i++)
                coordinates[i] /= abs;
        }
        public static double DotProduct(UniformPoint a, UniformPoint b)
        {
            double ret = 0;
            for (int i = 0; i < 3; i++)
                ret += a.coordinates[i] * b.coordinates[i];
            return ret;
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace Lab4
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Код основного приложения:

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Lab4
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            fade = new List<bool>();
            spectrum = new List<byte[]>();
            light = new List<UniformPoint>();
        }

        void Form1_Paint(object sender, PaintEventArgs e)
        {
            Draw();
        }
        Random random;
        Triangle[] triangles;
        Color[] colors;
        UniformPoint[] vertices;
        double[,] zBuffer;
        double pd = 1;
        List<bool> fade;
        List<byte[]> spectrum;
        List<UniformPoint> light;

        int alpha = 255;
        void Swap(ref int a, ref int b)
        {
            int c = a;
            a = b;
            b = c;
        }
        double CalculateW(int x1, int y1, int z1, int x2, int y2, int z2, int
x3, int y3, int z3, int x, int y)
        {
           int a = y1 * z2 + y2 * z3 + y3 * z1 - y3 * z2 - y2 * z1 - y1 * z3;
           int b = x1 * z2 + x2 * z3 + x3 * z1 - x3 * z2 - x2 * z1 - x1 * z3;
           int c = x1 * y2 + x2 * y3 + x3 * y1 - x3 * y2 - x2 * y1 - x1 * y3;
           int d = x1 * y2 * z3 + y1 * z2 * x3 + x2 * y3 * z1 - x3 * y2 * z1
- x2 * y1 * z3 - x1 * y3 * z2;
            double w = ((double)(b * y - a * x + d)) / c;
            return w;
        }
        void ProcessTriangle(int triangle, Bitmap image, Graphics dc)
        {
            int a = triangles[triangle].GetI(0);
            int b = triangles[triangle].GetI(1);
            int c = triangles[triangle].GetI(2);
            if (vertices[a].GetIY() > vertices[b].GetIY())
                Swap(ref a, ref b);
            if (vertices[a].GetIY() > vertices[c].GetIY())
            {
                Swap(ref a, ref c);
                Swap(ref c, ref b);
            }
            else if (vertices[b].GetIY() > vertices[c].GetIY())
                Swap(ref c, ref b);
```

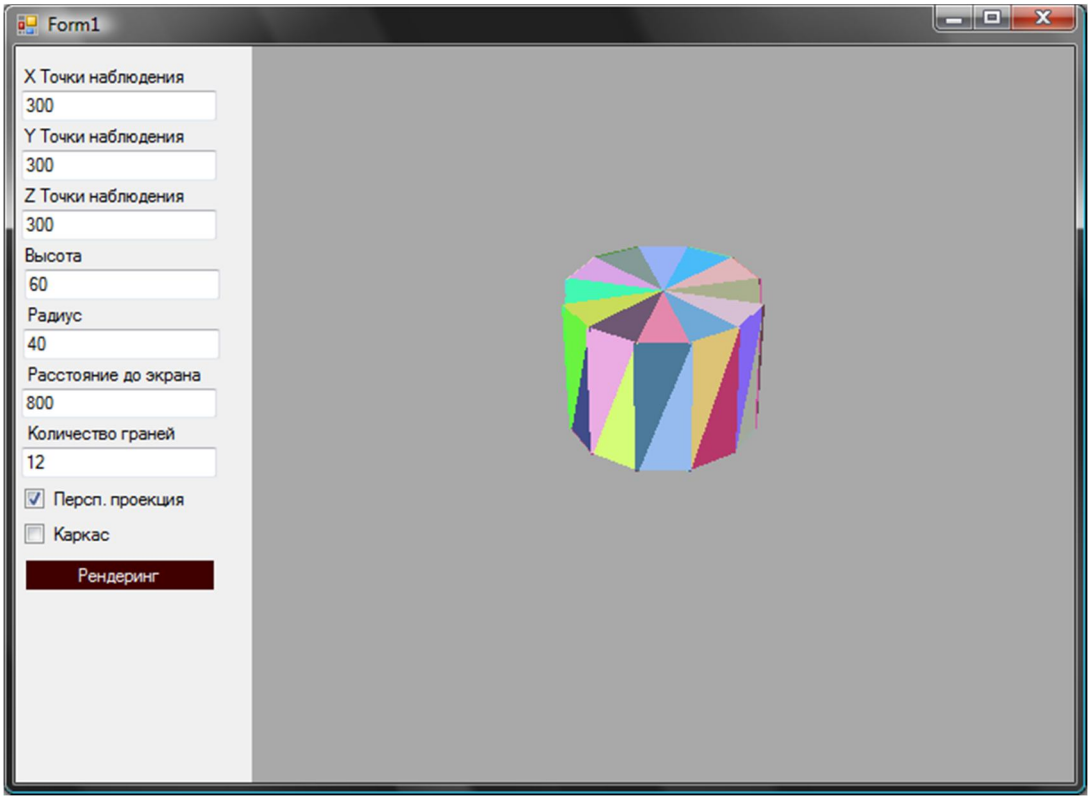
```
if (vertices[a].GetIY() == vertices[b].GetIY() &&
vertices[a].GetIX() > vertices[b].GetIX())
    Swap(ref a, ref b);
int x1 = vertices[a].GetIX();
int y1 = vertices[a].GetIY();
int z1 = vertices[a].GetIZ();
int x2 = vertices[b].GetIX();
int y2 = vertices[b].GetIY();
int z2 = vertices[b].GetIZ();
int x3 = vertices[c].GetIX();
int y3 = vertices[c].GetIY();
int z3 = vertices[c].GetIZ();
int pLx = Math.Abs(x3 - x1);
int pLy = y3 - y1;
int sLx = Math.Abs(x2 - x1);
int sLy = y2 - y1;
int pD = 0;
int sD = 0;
int pX = x1;
int pX1 = pX;
int sX = x1;
int sX1 = sX;
int pDs = pLy << 1;
int sDs = sLy << 1;
int pT = pLx << 1;
int sT = sLx << 1;
int k = x3;
int l = x1;
int m = x2;
int n = x1;
Color color = Color.FromArgb(alpha
            , random.Next(55, 255)
            , random.Next(55, 255)
            , random.Next(55, 255));
for (int y = y1; y <= y3; y++)
{
    if (y == y2)
    {
        sLy = y3 - y2;
        sLx = Math.Abs(x3 - x2);
        sD = 0;
        sDs = sLy << 1;
        sT  = sLx << 1;
        m = x3;
        n = x2;
        sX = x2;
        sX1 = sX;
    }
    pD += pT;
    sD += sT;
    while (pD > pLy && pX != k)
    {
        pD -= pDs;
        if (k > l)
            pX++;
        else pX--;
    }
    if(pT != 0)
    if (k > l )
        pX1 = pX - 1;
    else pX1 = pX + 1;
    while (sD > sLy && sX != m)
    {
        sD -= sDs;
```

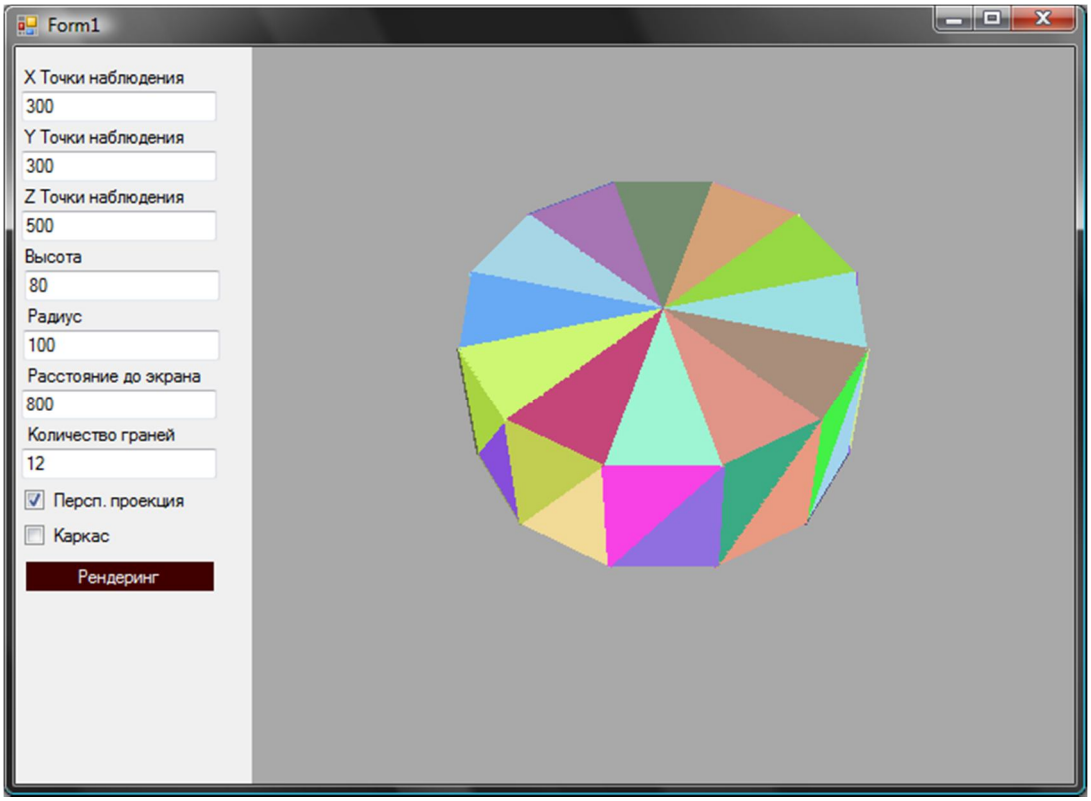
```
                    if (m > n)
                        sX++;
                    else sX--;
                }
                if(sT !=0 )
                if (m > n)
                    sX1 = sX - 1;
                else sX1 = sX + 1;

                 for (int x = Math.Min(pX1, sX1); x <= Math.Max(pX1, sX1); x++)
                if (x >= 0 && y >= 0 && x < image.Width && y < image.Height)
                        {
                double w = CalculateW(x1, y1, z1, x2, y2, z2, x3, y3, z3, x, y);
                            if (w < zBuffer[x, y] && w > 0)
                            {
                                image.SetPixel(x, y, color);
                                zBuffer[x, y] = w;
                            }
                        }
            }
        }

        void MakePolygon(int n, double radius, double h, UniformPoint[] ret)
        {
            double addAngle = Math.PI * 2 / n;
            double angle = 0;
            for (int i = 0; i < n; i++)
            {
                ret[i] = new UniformPoint(radius * Math.Cos(angle), radius *
Math.Sin(angle), h, 1);
                angle += addAngle;
            }
        }

        void MakePrism(int edgeCount, int height, int radius)
        {
            triangles = new Triangle[edgeCount * 4];
            vertices = new UniformPoint[edgeCount * 2 + 2];
            UniformPoint[] tmp = new UniformPoint[edgeCount];
            MakePolygon(edgeCount, radius, 0, tmp);
            for (int i = 0; i < edgeCount; i++)
                vertices[i] = tmp[i];
            MakePolygon(edgeCount, radius, height, tmp);
            for (int i = 0; i < edgeCount; i++)
                vertices[i + edgeCount] = tmp[i];
            vertices[edgeCount * 2] = new UniformPoint(0, 0, 0, 1);
            vertices[edgeCount * 2 + 1] = new UniformPoint(0, 0, height, 1);

            for (int i = 0; i < triangles.Length; i++)
                triangles[i] = new Triangle(0, 0, 0);

            for (int i = 0; i < edgeCount; i++)
            {
                int v1 = i;
                int v2 = (i + 1) % edgeCount;
                int v3 = i + edgeCount;
                int v4 = i + edgeCount + 1;
                if(v4 == edgeCount * 2)
                    v4 = edgeCount;

                triangles[i] = new Triangle(edgeCount * 2, v1, v2);
                triangles[i + edgeCount] = new Triangle(edgeCount * 2 + 1,
v3, v4);
```

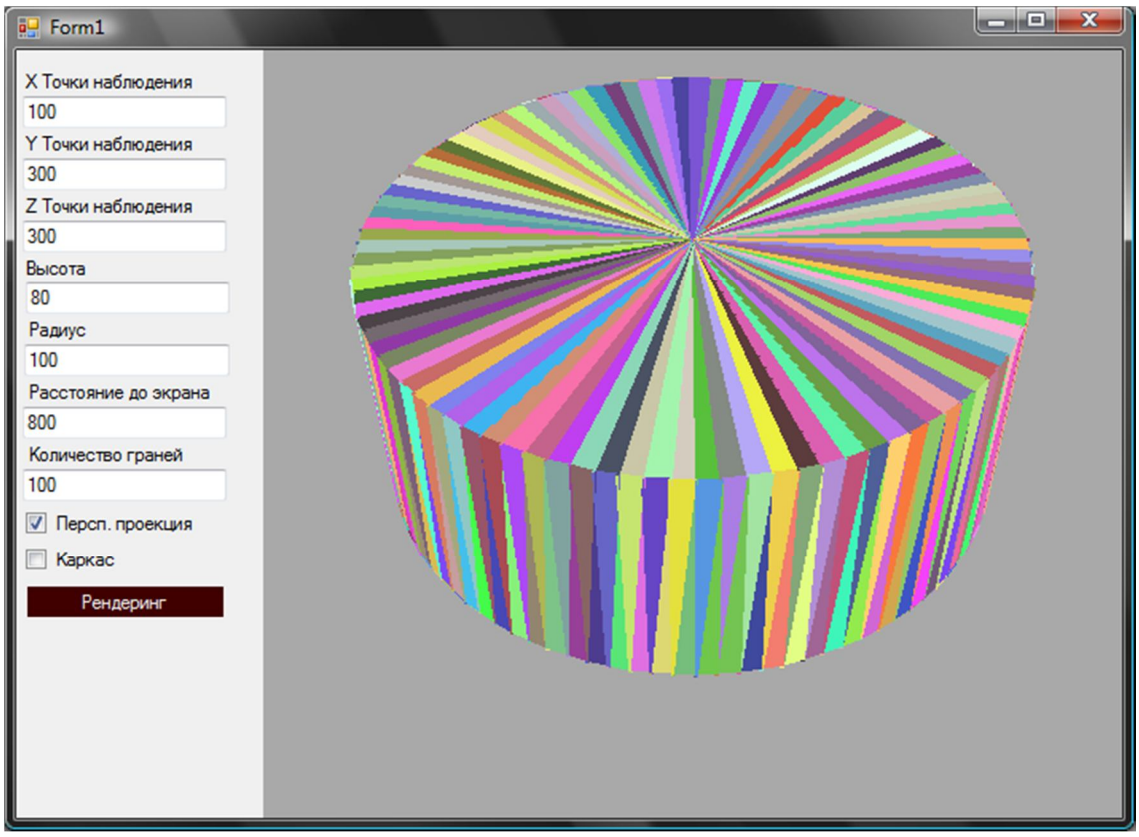
```
                triangles[i + edgeCount * 2] = new Triangle(v1, v3, v4);
                triangles[i + edgeCount * 3] = new Triangle(v1, v4, v2);
            }

        }

        void FillParaphilia(double[][] paraphilia)
        {
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    paraphilia[i][j] = 0;
            paraphilia[0][2] = -1;
            paraphilia[1][0] = 1;
            paraphilia[2][1] = 1;
            paraphilia[3][3] = 1;
        }
        void Draw()
        {
            random = new Random();
            zBuffer = new double[panel2.Width, panel2.Height];

            for (int i = 0; i < panel2.Width; i++)
                for (int j = 0; j < panel2.Height; j++)
                {
                    zBuffer[i, j] = 1 << 22;

                }
            int xView = 0;
            int yView = 0;
            int zView = 0;
            int radius = 0;
            int height = 0;
            int edgeCount = 0;
            float d = 0;
            try
            {
                xView = Convert.ToInt32(viewPointX.Text);
                yView = Convert.ToInt32(viewPointY.Text);
                zView = Convert.ToInt32(viewPointZ.Text);
                edgeCount = Convert.ToInt32(edgeCountBox.Text);
                height = Convert.ToInt32(heightBox.Text);
                radius = Convert.ToInt32(radiusBox.Text);
                d = (float)Convert.ToDouble(screenDist.Text);
            }
            catch (Exception)
            {
                MessageBox.Show("Проблема");
                return;
            }
            MakePrism(edgeCount, height, radius);
            double mv = Math.Sqrt(xView * xView + yView * yView);
            double me = Math.Sqrt(xView * xView + yView * yView + zView *
zView);
            double cost = 0;
            double sint = 1;
            if(mv != 0)
            {
                cost = (double)(xView) / mv;
                sint = (double)(yView) / mv;
            }
            double cosp = (double)(zView) / me;
            double sinp = mv / me;
            double[][] rotateZ = new double[4][];
            double[][] rotateY = new double[4][];
```

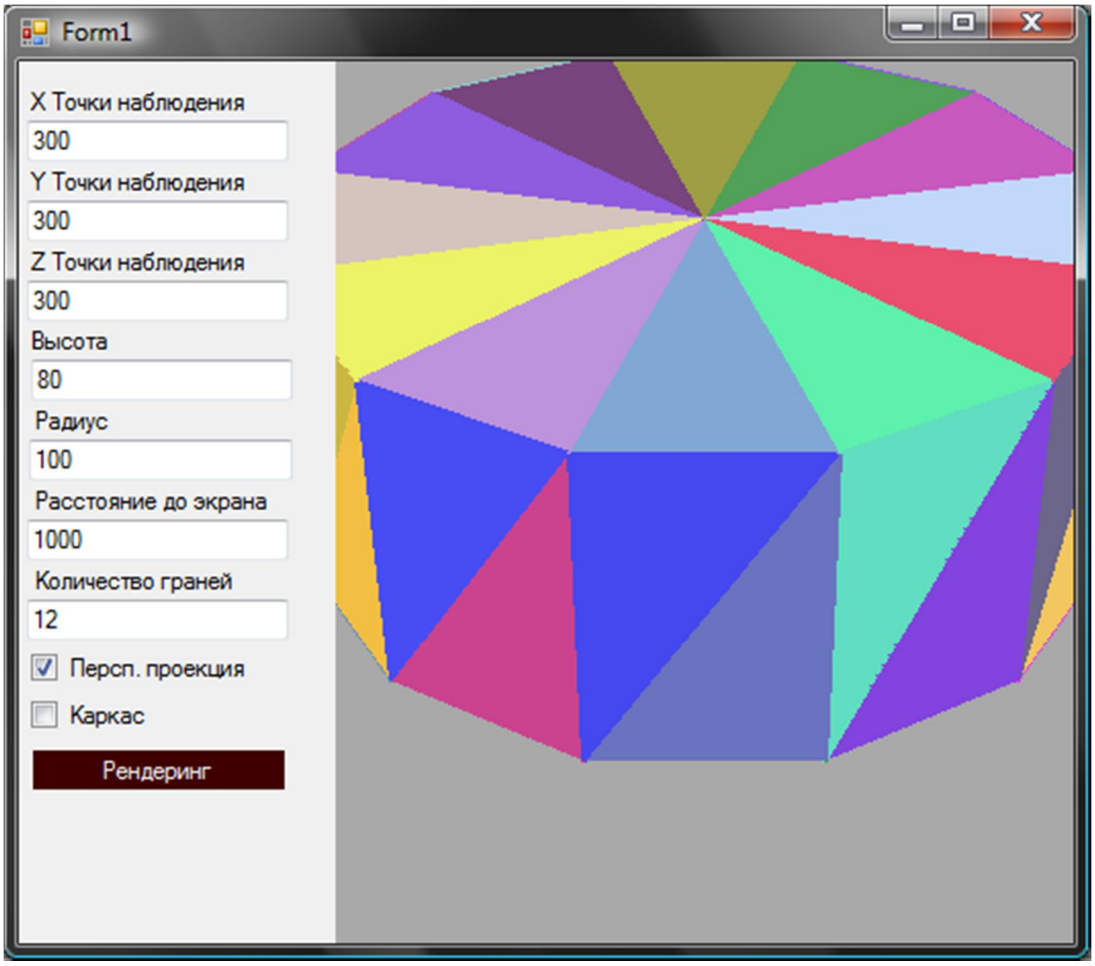
```
            double[][] tmp = new double[4][];
            double[][] tmp2 = new double[4][];
            double[][] translation = new double[4][];
            double[][] paraphilia = new double[4][];
            for (int i = 0; i < 4; i++)
            {
                tmp[i] = new double[4];
                tmp2[i] = new double[4];
                rotateY[i] = new double[4];
                rotateZ[i] = new double[4];
                translation[i] = new double[4];
                paraphilia[i] = new double[4];
            }
            FillParaphilia(paraphilia);
            Matrix.Make3DRotationY(sinp, cosp, rotateY);
            Matrix.Make3DRotationZ(cost, -sint, rotateZ);
            Matrix.Make3DTranslation(-me, 0, 0, translation);
            Matrix.MatrixMatrixMlp(rotateZ, rotateY, 4, tmp);
            Matrix.MatrixMatrixMlp(tmp, translation, 4, tmp2);
            Matrix.MatrixMatrixMlp(tmp2, paraphilia, 4, tmp);
            float dx = panel2.Width / 2;
            float dy = panel2.Height / 2;
            Graphics dc = panel2.CreateGraphics();
            dc.Clear(panel2.BackColor);
            Pen pen = new Pen(Color.White, 2);

            for (int i = 0; i < vertices.Length; i++)
            {
                vertices[i].Transform(tmp);
                if (perspective.Checked)
                    vertices[i].MakePerspectiveTransformation(d);
                vertices[i].Shift(dx, dy, 0);
            }
            if (isWireframe.Checked)
                for (int i = 0; i < triangles.Length; i++)
                {
                    float x1 = (float)vertices[triangles[i].GetI(0)].GetX();
                    float y1 = (float)vertices[triangles[i].GetI(0)].GetY();
                    float x2 = (float)vertices[triangles[i].GetI(1)].GetX();
                    float y2 = (float)vertices[triangles[i].GetI(1)].GetY();
                    float x3 = (float)vertices[triangles[i].GetI(2)].GetX();
                    float y3 = (float)vertices[triangles[i].GetI(2)].GetY();
                    dc.DrawLine(pen, x1, y1, x2, y2);
                    dc.DrawLine(pen, x1, y1, x3, y3);
                    dc.DrawLine(pen, x2, y2, x3, y3);
                }

            else
            {
                Bitmap image = new Bitmap(panel2.Width, panel2.Height);
                for (int i = 0; i < triangles.Length; i++)
                    ProcessTriangle(i, image, dc);
                dc.DrawImage(image, new Point(0, 0));
            }

        }
        private void button1_Click(object sender, EventArgs e)
        {
            Draw();
        }
    }
}
```

Демонстрация работы программы:






Form1
X Точки наблюдения
100
Y Точки наблюдения
300
Z Точки наблюдения
300
Высота
80
Радиус
100
Расстояние до экрана
800
Количество граней
100
Персп. проекция
Каркас
Рендеринг


Form1
X Точки наблюдения
300
Y Точки наблюдения
300
Z Точки наблюдения
300
Высота
80
Радиус
100
Расстояние до экрана
1000
Количество граней
12
Персп. проекция
Каркас
Рендеринг