Московский авиационный институт

(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Компьютерная графика»

Лабораторная работа № 6

Тема: Создание шейдерных анимационных эффектов в OpenGL

Студент: Махмудов Орхан
Группа: О8-305
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

.

Тема: Создание шейдерных анимационных эффектов в OpenGL 2.1.

Задание: Для поверхности, созданной в л.р. №5, обеспечить выполнение заданного преподавателем шейдерного эффекта.

Вариант №17: Анимация. Вращение относительно оси OZ Скорость вращения меняется по синусоиде.

2. Описание программы

Используемая среда: Visual Studio 2019

Используемые библиотеки: System.Windows.Media.Media3D; Glut; OpenGl

Язык программирования: C#

Используемые структуры данных: массивы, двумерные массивы.

Ввод: Все параметры задаются через консоль. Фигура также отображается в окне консоли.

Вывод: 3d-фигура прямой цилиндр с основанием - сектором параболы.

Краткая инструкция для пользователя: при запуске программы перед пользователем появляется окно консоли, в котором он вводит входные данные. А именно - пользователем задается коэффициент аппроксимации, радиус и высота фигуры, а также коэффициенты a, отвечающие за широту основания фигуры. После настройки параметров пользователь видит результирующую фигуру, которая автоматически вращается по заданному закону.

Задача аппроксимации в этом случае - определить, сколько сторон минимально должен иметь правильный многоугольник, чтобы соответствовать коэффициенту аппроксимации. По коэффициенту аппроксимации и задаваемому радиусу цилиндра можно найти радиус окружности, в которую можно вписать аппроксимированные многоугольниками основания цилиндра. Далее, через формулы описанной и вписанной окружностей (где r = a 2tg(π/n) R = a 2sin(π/n) a - длина стороны правильного многоугольника) можно найти искомое количество сторон многоугольника n = , которым аппроксимируются основания цилиндра.

3. Набор тестов

№ теста	**коэффициент аппроксимации**	**радиус цилиндра**	**высота цилиндра**	**коэффициент a**	**коэффициент b**
1	0,99	1	1	1	1
2	0,8	1	1,5	0,8	1

4. Результаты выполнения тестов

№ теста	коэффициент аппроксимации	радиус цилиндра	высота цилиндра	коэффициент a	коэффициент b
1	0,99	1	1	1	1

рисунок №1

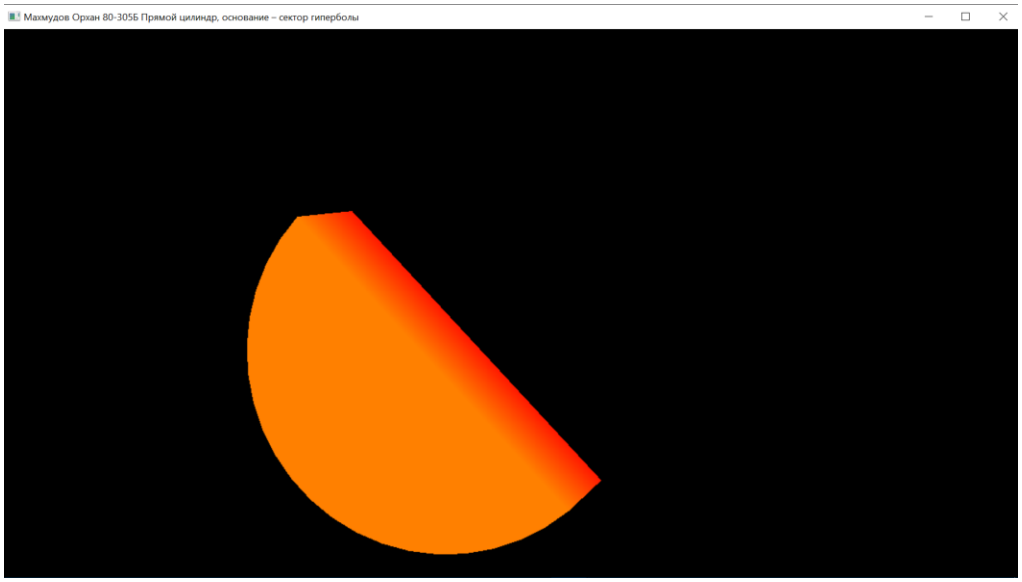

Рисунок №2

№ теста	коэффициент аппроксимации	радиус цилиндра	высота цилиндра	коэффициент a	коэффициент b
2	0,8	1	2	2	1

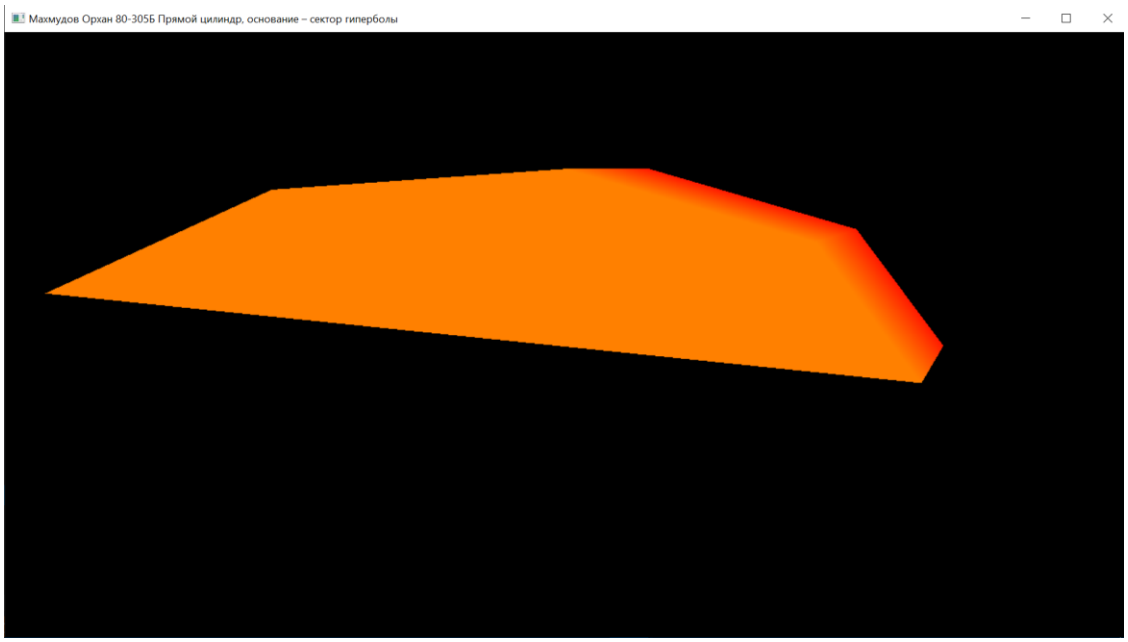

рисунок №1

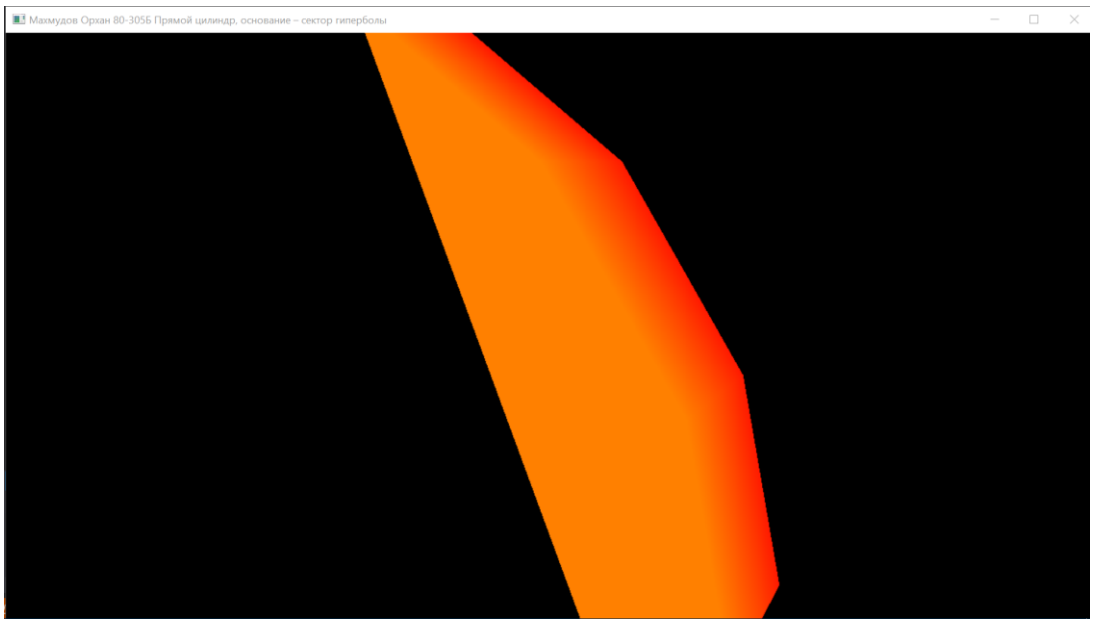

рисунок №2

5. Листинг программы

Program.cs

```
// Махмудов Орхан. 80-305Б Прямой цилиндр, основание – сектор гиперболы
// Анимация. Вращение относительно оси OZ Скорость вращения меняется по синусоиде.

using System;
using Tao.FreeGlut;
using OpenGL;

namespace CG_Lab_6
{
   class Program
   {
      public static int width = 1900, height = 1080;
      public static ShaderProgram program;

      private static int angle_number = 3; // кол-во углов пирамиды
      private static int level_number = 1; // общее кол-во пирамид
      private static double r = 1; // радиус цилиндра
      private static double Height = 1;
      private static double Approx = 0.7;
      private static double r_a = 1;
      private static double r_b = 1;

      private static Point[] bottom_points; // массив вершин нижнего основания
      private static Point[] top_points; // массив вершин верхнего основания

      private static System.Diagnostics.Stopwatch watch; // таймер
      private static float angle; // угол поворота

      // программа шейдера вершин
      public static string VertexShader = @"
      #version 130
      in vec3 vertexPosition;
      in vec3 vertexColor;
      out vec3 color;
      uniform mat4 projection_matrix;
      uniform mat4 view_matrix;
      uniform mat4 model_matrix;
      void main(void)
      {
         color = vertexColor;
         gl_Position = projection_matrix * view_matrix * model_matrix * vec4(vertexPosition, 1);
      }";
```

```
public static string FragmentShader = @"
#version 130
in vec3 color;
out vec4 fragment;
void main(void)
{
   fragment = vec4(color, 1);
}";

static void Main(string[] args)
{
   Console.Write("enter coefficient of approximation (0,6 - 0,99)  ");
   Approx = Convert.ToDouble(Console.ReadLine());
   if (Approx > 0.99)
      Approx = 0.99;
   if (Approx < 0.3)
      Approx = 0.6;

   angle_number = Convert.ToInt32(Math.PI / Math.Acos(Approx));
   if (angle_number < 3)
      angle_number = 3;

   Console.Write("enter radius (0,1 - 5)  ");
   r = Convert.ToDouble(Console.ReadLine());
   if (r > 5)
      r = 5;
   if (r < 0.1)
      r = 0.1;

   Console.Write("enter height (0,1 - 5)  ");
   Height = Convert.ToDouble(Console.ReadLine());
   if (Height > 5)
      Height = 5;
   if (Height < 0.1)
      Height = 0.1;

   Console.Write("enter coefficient a (0,1 - 5)  ");
   r_a = Convert.ToDouble(Console.ReadLine());
   if (r_a > 5)
      r_a = 5;
   if (r_a < 0.1)
      r_a = 0.1;

   Console.Write("enter coefficient a (0,1 - 5)  ");
   r_b = Convert.ToDouble(Console.ReadLine());
   if (r_b > 5)
      r_b = 5;
   if (r_b < 0.1)
      r_b = 0.1;
```

```
            watch = System.Diagnostics.Stopwatch.StartNew();
            InitOpenGL(); // инициализация OpenGL
        }

        private static void InitOpenGL()
        {
            Glut.glutInit();
            Glut.glutInitDisplayMode(Glut.GLUT_DOUBLE | Glut.GLUT_DEPTH);

            Glut.glutInitWindowSize(width, height);
            Glut.glutCreateWindow("Махмудов Орхан 80-305Б Прямой цилиндр, основание – сектор гиперболы");

            Glut.glutIdleFunc(OnRenderFrame);
            Glut.glutDisplayFunc(OnDisplay);
            Glut.glutCloseFunc(OnClose);

            Gl.Enable(EnableCap.DepthTest);

            program = new ShaderProgram(VertexShader, FragmentShader);

            // задаем положение изображения в окне
            program.Use();
            program["projection_matrix"].SetValue(Matrix4.CreatePerspectiveFieldOfView(0.45f, (float)width / height, 0.1f, 1000f));
            program["view_matrix"].SetValue(Matrix4.LookAt(new Vector3(0, 0, 10), new Vector3(-1,0,0), new Vector3(0, 1, 0)));
            program["model_matrix"].SetValue(Matrix4.CreateRotationX(180) * Matrix4.CreateTranslation(new Vector3(-1.5f, 0, 0)));

            Glut.glutMainLoop(); // старт
        }

        private static void OnClose()
        {
            program.DisposeChildren = true;
            program.Dispose();
        }

        private static void OnDisplay()
        {

        }

        private static void OnRenderFrame()
        {
            // изменяем угол поворота относительно таймера
            watch.Stop();
            float deltaTime = (float)watch.ElapsedTicks / System.Diagnostics.Stopwatch.Frequency;
```

```
            watch.Restart();
            angle += deltaTime;

            // осуществляем поворот
            program["model_matrix"].SetValue(Matrix4.CreateRotationY(angle)                    *
Matrix4.CreateRotationZ(angle) * Matrix4.CreateTranslation(new Vector3(-1.5f, 0, 0)));

            // инициализируем Viewport
            Gl.Viewport(0, 0, Program.width, Program.height);
            Gl.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

            // Алгоритм аналогичен Лаб№3
            bottom_points = new Point[angle_number];
            top_points = new Point[angle_number];

            CountBottomPoints();

            for (int level = 0; level < level_number; level++)
            {
                CountTopPoints(level);
                TruncatedPyramid pyramid = new TruncatedPyramid(bottom_points, top_points);
                pyramid.Draw();
                for (int i = 0; i < angle_number; i++) bottom_points[i] = top_points[i];
                pyramid.Dispose();
            }

            Glut.glutSwapBuffers();
        }

        // Просчитываем координаты точек нижнего основания
        private static void CountBottomPoints()
        {
            double step = Math.PI / angle_number;
            double alpha = 0;
            for (int i = 0; i < angle_number; i++)
            {
                double x = r_a*r * Math.Cos(alpha);
                double y = r_b*r * Math.Sin(alpha);
                bottom_points[i] = new Point((float)x, (float)y, 0);

                alpha += step;
            }
        }

        // Просчитываем координаты точек верхнего основания
        private static void CountTopPoints(int current_level)
        {
            double alpha = Math.PI / (2 * level_number + 1);
            for (int i = 0; i < angle_number; i++)
            {
```

```
            Point A = bottom_points[i];

            double z = Height * r * Math.Sin((current_level + 1) * alpha);
            //double k = r * Math.Cos((current_level + 1) * alpha) / Math.Sqrt(A.X * A.X + A.Y * A.Y);
            double x = A.X;
            double y = A.Y;

            top_points[i] = new Point((float)x, (float)y, (float)z);
        }
    }
}
```

TruncatedPyramid.cs

```
using System;
using Tao.FreeGlut;
using OpenGL;

namespace CG_Lab_4_5
{
    class TruncatedPyramid
    {
        public VBO<Vector3> vertices;
        public VBO<Vector3> color;
        public VBO<int> elements;

        Vector3[] vector_vertices;
        Vector3[] vector_color;
        int[] array_elements;

        public TruncatedPyramid(Point[] bottom_points, Point[] top_points)
        {
            int n = top_points.Length;
            vector_vertices = new Vector3[4 * n];
            vector_color = new Vector3[6 * n * n];
            array_elements = new int[6 * n*n];

            // рисуем основания
            int j = 0;
            for(int i=2; i<n; i++)
            {
                array_elements[j] = 0;
                array_elements[j + 1] = i;
                array_elements[j + 2] = i - 1;
                array_elements[j + 3] = n;
                array_elements[j + 4] = i + n;
                array_elements[j + 5] = i - 1 + n;
                j += 6;
            }
```

```
            // рисуем боковые стороны
            for(int i=0; i<n;i++)
            {
                vector_vertices[i] = new Vector3(bottom_points[i].X, bottom_points[i].Y,
bottom_points[i].Z);
                vector_color[i] = new Vector3(2.0f, 0.1f, 0.0f);

                vector_vertices[i + n] = new Vector3(top_points[i].X, top_points[i].Y, top_points[i].Z);
                vector_color[i+n] = new Vector3(1.0f, 0.5f, 0.0f);

                array_elements[j] = i + n;
                array_elements[j + 1] = i;
                array_elements[j + 2] = i+1 ;
                array_elements[j + 3] = i+n;
                array_elements[j + 4] = i+1;
                array_elements[j + 5] = i+1+n;
                j += 6;
            }
            j -= 6;

            array_elements[j] = 2*n-1;
            array_elements[j + 1] = n-1;
            array_elements[j + 2] = 0;
            array_elements[j + 3] = 2 * n - 1;
            array_elements[j + 4] = 0;
            array_elements[j + 5] = n;

            // создаем объекты из массивов
            vertices = new VBO<Vector3>(vector_vertices);
            color = new VBO<Vector3>(vector_color);
            elements = new VBO<int>(array_elements, BufferTarget.ElementArrayBuffer);
        }

        public void Draw()
        {
            Gl.UseProgram(Program.program);

            Gl.BindBufferToShaderAttribute(vertices, Program.program, "vertexPosition");
            Gl.BindBufferToShaderAttribute(color, Program.program, "vertexColor");
            Gl.BindBuffer(elements);

            // непосредственная прорисовка
            Gl.DrawElements(BeginMode.Triangles, elements.Count, DrawElementsType.UnsignedInt,
IntPtr.Zero);
        }

        // освобождаем память
        public void Dispose()
        {
```

```
            vertices.Dispose();
            elements.Dispose();
            color.Dispose();

            Array.Clear(vector_vertices, 0, vector_vertices.Length);
            Array.Clear(vector_color, 0, vector_color.Length);
            Array.Clear(array_elements, 0, array_elements.Length);
        }
    }
}
```

Вывод

В ходе данной лабораторной работы были приобретены навыки по работе с библиотекой OpenGL , постройки тела выпуклым многогранником, была обеспечена возможность вращения многогранника по заданной траектории, удаление невидимых линий, а также была разобрана и запрограммирована простая модель закраски для случая одного источника света.

Список литературы

1. Статья “Графическая библиотека OpenGl”

   URL: https://www.rsdn.org/article/opengl/ogltut2.xml

2. Статья “Уроки OpenGl”

   URL: https://www.cyberforum.ru/opengl/thread1404219.html

3. “Уроки OpenGl + C#”

   URL: http://esate.ru/uroki/OpenGL/uroki-OpenGL-c-sharp/