Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Компьютерная графика»
Тема: Поверхность вращения. Образующая – Cardinal Spline 3D.

Студент: Д. С. Ляшун
Преподаватель: А. В. Морозов
Группа: М8О-307Б

Дата:
Оценка:
Подпись:

Москва, 2022

1 Постановка задачи

Цель: Знакомство с построением сложных 3D фигур, закрепление полученных знаний по курсу.

Задание: Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и переключаются из графического интерфейса. Должна быть обеспечена возможность тестирования программы на различных наборах подготовленных исходных данных и их изменение. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах или буфер вершин OpenGL. Реализовать возможность отображения опорных точек, направляющих и других данных по которым формируется порция поверхности и отключения каркасной визуализации.

Вариант тела: Поверхность вращения. Образующая - Cardinal Spline 3D.

2 Теоретические сведения

Реализация точек образующей будет производиться по следующему алгоритму. Для начала вычислим коэффициенты для отрисовки каждого сегмента, состоящего из четырёх точек, по следующим формулам:

$$\begin{cases} k_{4i} = 2(\frac{i}{s_{cnt}})^3 - 3(\frac{i}{s_{cnt}})^2 + 1 \\ k_{4i+1} = 3(\frac{i}{s_{cnt}})^2 - 2(\frac{i}{s_{cnt}})^3 \\ k_{4i+2} = (\frac{i}{s_{cnt}})^3 - 2(\frac{i}{s_{cnt}})^2 + \frac{i}{s_{cnt}} \\ k_{4i+3} = (\frac{i}{s_{cnt}})^3 - (\frac{i}{s_{cnt}})^2 \end{cases}$$

где s_{cnt} - общее количество сегментов, i - номер текущего сегмента, $i = 1..s_{cnt}$. Полученные коэффициенты используются при вычислении точек сплайна по следующей формуле:

$$point_{res} = k_{4t}p_1 + k_{4t+1}p_2 + k_{4t+2}p_3 + k_{4t+3}p_4,$$

$$p_1 = point_i$$

$$p_2 = point_{i+1}$$

$$p_3 = (p_2 - point_{i-1})tension$$

$$p_4 = (point_{i+2} - p_1)tension$$

где $point$ - набор точек из исходного набора образующих, p_3, p_4 - образующие вектора по этим точкам, $tension$ - коэффициент приближения от линии к сплайну.

Стоит отметить, что исходные наборы точек-образующих будут задаваться как пресеты в текстовых файлах с возможностью редактирования, сами пресеты интерактивно могут переключаться в ходе работы программы.

После вычисления всех точек сегментов необходимо провести операцию поворота сплайна для создания 3D фигуры. Для этого достаточно в цикле путем аффинного преобразования - поворота по Y-оси поворачивать ранее сгенерированную область по кругу - число поворотов можно считать как параметр аппроксимации.

В программе будет доступно несколько вариантов отрисовки фигуры:

1. Одним цветом, задаваемым как параметр - трёхкомпонентный вектор в программном окне.

2. Каркасная визуализация - без отображения полигонов, только образующие ребра фигуры.

3. Случайные цвета - каждый полигон случайным образом окрашивается в какой-то цвет.

4. Затенение по Фонгу - с возможностью полной настройки необходимых параметров освещения и позиции источника света.

Все основные варианты аффинных преобразований - смещение, поворот и масштабирование, а также проекции - спереди, сверху, сбоку и изометрия, будут реализованы в рабочей программе.

Сама фигура будет отрисовываться с помощью средств OpenGL и храниться в буфере вершин VBO. Помимо самой фигуры будут также показываться её нормали и образующие точки. При проецировании фигуры на плоскость используется матрицы проекций с некоторыми редактируемыми значениями - поле зрения и плоскости отсечения.

3 Листинг программы

Генерация фигуры:

```
public static DMatrix4 Scaling(float x, float y, float z)
{
    DMatrix4 r = DMatrix4.Identity;

    r[1, 1] = x;
    r[2, 2] = y;
    r[3, 3] = z;
    return r;
}

public static DMatrix4 RotateY(float ang)
{
    return RotateAxis(ang, 1, 3);
}

public static DMatrix4 RotateAxis(float ang, int ax1, int ax2)
{
    DMatrix4 r = DMatrix4.Identity;

    float sn = (float)Math.Sin(ang),
        cs = (float)Math.Cos(ang);

    r[ax1, ax1] = cs;
    r[ax1, ax2] = sn;
    r[ax2, ax1] = -sn;
    r[ax2, ax2] = cs;

    return r;
}


List<DVector3> Points = new List<DVector3>();
List<DVector3> SplinePoints = new List<DVector3>();

// cardinal spline
public void calcSpline(ref List<DVector3> res, float tension, int numOfSeg)
{
    res.Clear();
    if (Points.Count < 4) return;

    var cache = new float[(numOfSeg + 1) * 4];
    int i, cachePtr;

    cache[0] = 1;
    cachePtr = 4;
    for (i = 1; i < numOfSeg; i++)
```

```
47      {
48          float st = i * 1.0f / numOfSeg,
49              st2 = st * st,
50              st3 = st2 * st,
51              st23 = st3 * 2,
52              st32 = st2 * 3;
53
54          cache[cachePtr++] = st23 - st32 + 1; // c1
55          cache[cachePtr++] = st32 - st23; // c2
56          cache[cachePtr++] = st3 - 2 * st2 + st; // c3
57          cache[cachePtr++] = st3 - st2; // c4
58      }
59
60      cache[++cachePtr] = 1;
61
62      List<DVector3> pts = new List<DVector3>(Points);
63      pts.Insert(0, Points[0]);
64      pts.Add(Points[Points.Count - 1]);
65
66      for (i = 1; i < Points.Count; i++)
67      {
68          DVector3
69              pt1 = pts[i],
70              pt3 = pts[i + 1];
71
72          DVector3
73              t1 = (pt3 - pts[i - 1]) * tension,
74              t2 = (pts[i + 2] - pt1) * tension;
75
76          for (int t = 0; t < numOfSeg; t++)
77          {
78
79              var c = t * 4;
80
81              float
82                  c1 = cache[c],
83                  c2 = cache[c + 1],
84                  c3 = cache[c + 2],
85                  c4 = cache[c + 3];
86
87              res.Add(c1 * pt1 + c2 * pt3 + c3 * t1 + c4 * t2);
88          }
89      }
90  }
91
92  DVector4[,] mPoints;
93
94  public void calcModel(int w)
95  {
```

```
    int x, y, h;
    float xm, ym;

    DMatrix4 m1;

    h = SplinePoints.Count;

    mPoints = new DVector4[w + 1, h];

    xm = 1;
    ym = 1; // 0.5f;
    float scale = 0.05f;

    for (x = 0; x <= w; x++)
    {
        m1 = Scaling(scale * xm, scale, scale * ym) *
            RotateY((float)(x * 2.0 / w * Math.PI));

        for (y = 0; y < h; y++)
        {
            DVector4 p4 = new DVector4(SplinePoints[y], 1);
            mPoints[x, y] = m1 * p4;
        }
    }
    return;
}


int[,] quad =
{
        { 0, 0 },
        { 1, 0 },
        { 1, 1 },

        { 0, 0 },
        { 1, 1 },
        { 0, 1 },
};

public void finishCalcModel(int w, DeviceArgs e)
{
    var random = new Random();
    int h, x, y, i;

    h = SplinePoints.Count;

    vertices = new Vertex[(w) * h];
    indices = new uint[w * (h - 1) * 6];

```

```
145     for (x = 0; x < w; ++x)
146     {
147         for (y = 0; y < h; ++y)
148         {
149             var v = mPoints[x, y];
150             vertices[x*h + y] = new Vertex((float)v[0], (float)v[1], (float)v[2], (
                    float)v[3], (byte)random.Next(256), (byte)random.Next(256), (byte)random
                    .Next(256));
151         }
152     }
153 
154     int ind_indx = 0;
155 
156     for (x = 0; x < w; x++)
157     {
158         for (y = 0; y < h - 1; y++)
159         {
160             var p1 = mPoints[x + 1, y] - mPoints[x, y];
161             var p2 = mPoints[x, y + 1] - mPoints[x, y];
162             var n = -(p1 * p2);
163 
164             for (i = 0; i < 6; i++)
165             {
166                 var indx = (x + quad[i, 0]) * h + (y + quad[i, 1]);
167                 if (indx >= h * w) indx -= h * w;
168                 ChangeNormale(ref vertices[indx], n);
169                 indices[ind_indx++] = (uint) indx;
170             }
171         }
172     }
173 
174     for (int j = 0; j < vertices.Length; ++j)
175     {
176         DVector4 vec4 = new DVector4(vertices[j].nx, vertices[j].ny, vertices[j].nz,
                vertices[j].nw);
177         vec4.Normalize();
178         vertices[j].nx = (float)vec4.X;
179         vertices[j].ny = (float)vec4.Y;
180         vertices[j].nz = (float)vec4.Z;
181         vertices[j].nw = (float)vec4.W;
182     }
183 
184     normalPoints = new DVector4[((w) * h) * 2];
185     normalIndices = new uint[((w) * h) * 2];
186 
187     double normalLength = 0.25;
188     for (i = 0; i < vertices.Length; ++i)
189     {
```

```
190         normalPoints[2 * i] = new DVector4(vertices[i].vx, vertices[i].vy, vertices[i].
                vz, vertices[i].vw);
191         normalPoints[2 * i + 1] = new DVector4(vertices[i].vx + normalLength * vertices
                [i].nx,
192             vertices[i].vy + normalLength * vertices[i].ny,
193             vertices[i].vz + normalLength * vertices[i].nz,
194             vertices[i].vw + normalLength * vertices[i].nw);
195     }
196     for (i = 0; i < normalIndices.Length; ++i)
197     {
198         normalIndices[i] = (uint)i;
199     }
200 
201     headPoints = new DVector4[Points.Count];
202     headIndicies = new uint[Points.Count];
203 
204     for (i = 0; i < headPoints.Length; ++i)
205     {
206         headPoints[i] = new DVector4(Points[i], 1);
207         headPoints[i] = Scaling(0.05f, 0.05f, 0.05f) * headPoints[i];
208         headIndicies[i] = (uint)i;
209     }
210 
211     var gl = e.gl;
212     unsafe
213     {
214         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vertexBuffer[0]);
215         fixed (Vertex* ptr = &vertices[0])
216         {
217             gl.BufferData(OpenGL.GL_ARRAY_BUFFER, vertices.Length * sizeof(Vertex), (
                    IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
218         }
219 
220         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, indexBuffer[0]);
221         fixed (uint* ptr = &indices[0])
222         {
223             gl.BufferData(OpenGL.GL_ELEMENT_ARRAY_BUFFER, indices.Length * sizeof(uint)
                    , (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
224         }
225 
226         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, normalDataBuffer[0]);
227         fixed (DVector4* ptr = &normalPoints[0])
228         {
229             gl.BufferData(OpenGL.GL_ARRAY_BUFFER, normalPoints.Length * sizeof(DVector4
                    ), (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
230         }
231 
232         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, normalIndexBuffer[0]);
233         fixed (uint* ptr = &normalIndices[0])
```

```
234         {
235             gl.BufferData(OpenGL.GL_ELEMENT_ARRAY_BUFFER, normalIndices.Length * sizeof
                (uint), (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
236         }
237 
238         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, headBuffer[0]);
239         fixed (DVector4* ptr = &headPoints[0])
240         {
241             gl.BufferData(OpenGL.GL_ARRAY_BUFFER, headPoints.Length * sizeof(DVector4),
                 (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
242         }
243 
244         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, headIndexBuffer[0]);
245         fixed (uint* ptr = &headIndicies[0])
246         {
247             gl.BufferData(OpenGL.GL_ELEMENT_ARRAY_BUFFER, headIndicies.Length * sizeof(
                uint), (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
248         }
249     }
250 }
251 
252 
253 void Generate(DeviceArgs e)
254 {
255     StreamReader file = new StreamReader("preset1.txt");
256     if (CurPreset == Presets.Preset1)
257     {
258         file = new StreamReader("preset1.txt");
259     }
260     else if (CurPreset == Presets.Preset2)
261     {
262         file = new StreamReader("preset2.txt");
263     }
264     else if (CurPreset == Presets.Preset3)
265     {
266         file = new StreamReader("preset3.txt");
267     }
268     else if (CurPreset == Presets.Preset4)
269     {
270         file = new StreamReader("preset4.txt");
271     }
272 
273     string[] words = file.ReadToEnd().Split(' ', '\n');
274     Points = new List<DVector3>();
275     for (int i = 0; i < words.Length; i += 2)
276     {
277         Points.Add(new DVector3(Convert.ToInt32(words[i]), Convert.ToInt32(words[i +
                1]), 0));
278     }
```

```
279     file.Close();
280
281     int numSeg = NumSeg;
282
283     calcSpline(ref SplinePoints, 0.5f, numSeg);
284     calcModel(numSeg);
285     finishCalcModel(numSeg, e);
286 }
```

Основной цикл работы:

```
1  protected override void OnDeviceUpdate(object s, DeviceArgs e)
2  {
3      var gl = e.gl;
4
5      gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT | OpenGL.
           GL_STENCIL_BUFFER_BIT);
6
7      if (0 != ((int)_Commands & (int)Commands.ChangeProjectionMatrix))
8      {
9          _Commands ^= Commands.ChangeProjectionMatrix;
10         UpdateProjectionMatrix(e);
11         _Commands |= Commands.Transform;
12     }
13
14     if (0 != ((int)_Commands & (int)Commands.NewFigure))
15     {
16         _Commands ^= Commands.NewFigure;
17         Generate(e);
18         _Commands |= Commands.FigureChange;
19     }
20
21     if (0 != ((int)_Commands & (int)Commands.FigureChange))
22     {
23         _Commands ^= Commands.FigureChange;
24         Generate(e);
25     }
26
27     if (0 != ((int)_Commands & (int)Commands.Transform))
28     {
29         _Commands ^= Commands.Transform;
30         UpdateModelViewMatrix(e);
31     }
32
33     if (0 != ((int)_Commands & (int) Commands.ChangeLightPos))
34     {
35         _Commands ^= Commands.ChangeLightPos;
36
37         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, LightVertexBuffer[0]);
38         DVector4 LightPosV4 = new DVector4(LightPos, 1);
```

```
39          unsafe
40          {
41              LightVertexArray = LightPosV4.ToArray();
42
43              fixed (double* ptr = &LightVertexArray[0])
44              {
45                  gl.BufferData(OpenGL.GL_ARRAY_BUFFER, LightVertexArray.Length * sizeof(
                        double), (IntPtr) ptr, OpenGL.GL_STATIC_DRAW);
46              }
47          }
48
49          gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, LightIndexBuffer[0]);
50          unsafe
51          {
52              fixed (uint* ptr = &LightIndexValues[0])
53              {
54                  gl.BufferData(OpenGL.GL_ELEMENT_ARRAY_BUFFER, LightIndexValues.Length *
                        sizeof(uint), (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
55              }
56          }
57
58          LightPos_InWorldSpace = _PointTransform * LightPosV4;
59
60      }
61
62      if (CurVisual == Visualization.OneColor)
63      {
64          gl.PolygonMode(OpenGL.GL_FRONT_AND_BACK, OpenGL.GL_FILL);
65          gl.Color(MaterialColor.X, MaterialColor.Y, MaterialColor.Z);
66      }
67      else if (CurVisual == Visualization.RandomColor)
68      {
69          gl.PolygonMode(OpenGL.GL_FRONT_AND_BACK, OpenGL.GL_FILL);
70          gl.EnableClientState(OpenGL.GL_COLOR_ARRAY);
71          unsafe
72          {
73              gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vertexBuffer[0]);
74              gl.ColorPointer(3, OpenGL.GL_BYTE, sizeof(Vertex), (IntPtr)(sizeof(float) *
                     8));
75          }
76      }
77      else if (CurVisual == Visualization.NoPolygons)
78      {
79          gl.PolygonMode(OpenGL.GL_FRONT_AND_BACK, OpenGL.GL_LINE);
80          gl.Color(MaterialColor.X, MaterialColor.Y, MaterialColor.Z);
81      }
82      else if (CurVisual == Visualization.PhongShading)
83      {
84          gl.PolygonMode(OpenGL.GL_FRONT_AND_BACK, OpenGL.GL_FILL);
```

```
85      }
86
87      gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vertexBuffer[0]);
88      gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, indexBuffer[0]);
89
90      if (CurVisual == Visualization.PhongShading)
91      {
92          gl.UseProgram(prog_shader);
93
94          UpdateLightValues(e);
95          gl.UniformMatrix4(uniform_ModelView, 1, true, ConvertToFloatArray(
                ModelViewMatrix));
96          gl.UniformMatrix4(uniform_Projection, 1, true, ConvertToFloatArray(pMatrix));
97          gl.UniformMatrix4(uniform_NormalMatrix, 1, true, ConvertToFloatArray(
                NormalMatrix));
98          gl.UniformMatrix4(uniform_PointMatrix, 1, true, ConvertToFloatArray(
                _PointTransform));
99
100         gl.EnableVertexAttribArray((uint)attribute_normale);
101         gl.EnableVertexAttribArray((uint)attribute_coord);
102         unsafe
103         {
104             gl.VertexAttribPointer((uint)attribute_normale, 4, OpenGL.GL_FLOAT, false,
                    sizeof(Vertex), (IntPtr)(4 * sizeof(float)));
105             gl.VertexAttribPointer((uint)attribute_coord, 4, OpenGL.GL_FLOAT, false,
                    sizeof(Vertex), (IntPtr)0);
106         }
107     }
108     else
109     {
110         gl.UseProgram(0);
111         gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
112         unsafe
113         {
114             gl.VertexPointer(4, OpenGL.GL_FLOAT, sizeof(Vertex), (IntPtr)0);
115         }
116     }
117
118     gl.DrawElements(OpenGL.GL_TRIANGLES, indices.Length, OpenGL.GL_UNSIGNED_INT, (
            IntPtr)0);
119
120     if (CurVisual == Visualization.PhongShading)
121     {
122         gl.DisableVertexAttribArray((uint)attribute_normale);
123         gl.DisableVertexAttribArray((uint)attribute_coord);
124         gl.UseProgram(0);
125     }
126     else
127     {
```

```
128             gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
129         }
130 
131         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
132         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
133 
134         if (CurVisual == Visualization.RandomColor)
135         {
136             gl.DisableClientState(OpenGL.GL_COLOR_ARRAY);
137         }
138 
139         if (isLightActive)
140         {
141             gl.Color(0.99, 0, 0);
142             gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, LightVertexBuffer[0]);
143             gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, LightIndexBuffer[0]);
144 
145             gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
146             unsafe
147             {
148                 gl.VertexPointer(4, OpenGL.GL_DOUBLE, sizeof(DVector4), (IntPtr)0);
149             }
150 
151             gl.PointSize(10);
152             gl.DrawElements(OpenGL.GL_POINTS, 1, OpenGL.GL_UNSIGNED_INT, (IntPtr)0);
153 
154             gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
155             gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
156             gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
157         }
158 
159         if (isNormalActive)
160         {
161             gl.Color(0, 0, 0.99);
162             gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, normalDataBuffer[0]);
163             gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, normalIndexBuffer[0]);
164 
165             gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
166             unsafe
167             {
168                 gl.VertexPointer(4, OpenGL.GL_DOUBLE, sizeof(DVector4), (IntPtr)0);
169             }
170 
171             gl.DrawElements(OpenGL.GL_LINES, normalPoints.Length, OpenGL.GL_UNSIGNED_INT, (
                    IntPtr)0);
172 
173             gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
174             gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
175             gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
```

```
176     }
177 
178     if (isHeadlActive)
179     {
180         gl.Color(0, 0.99, 0);
181         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, headBuffer[0]);
182         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, headIndexBuffer[0]);
183 
184         gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
185         unsafe
186         {
187             gl.VertexPointer(4, OpenGL.GL_DOUBLE, sizeof(DVector4), (IntPtr)0);
188         }
189 
190         gl.PointSize(10);
191         gl.DrawElements(OpenGL.GL_POINTS, headIndicies.Length, OpenGL.GL_UNSIGNED_INT,
                (IntPtr)0);
192 
193         gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
194         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
195         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
196     }
197 
198 }
```

4 Демонстрация работы

Рис. 1: Пресет №2, тип отрисовки - одним цветом

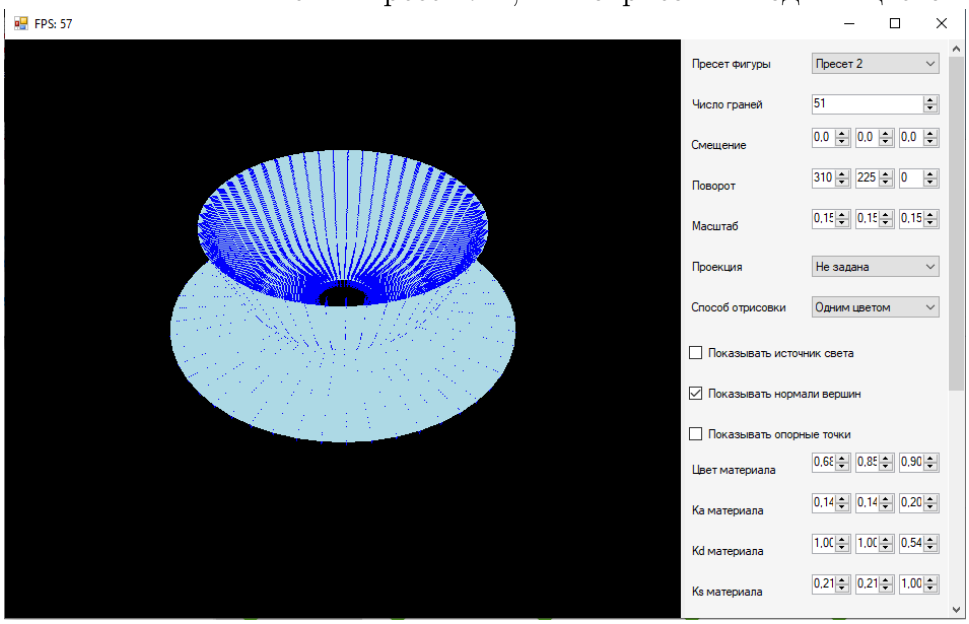


Рис. 2: Пресет №1, тип отрисовки - каркасный

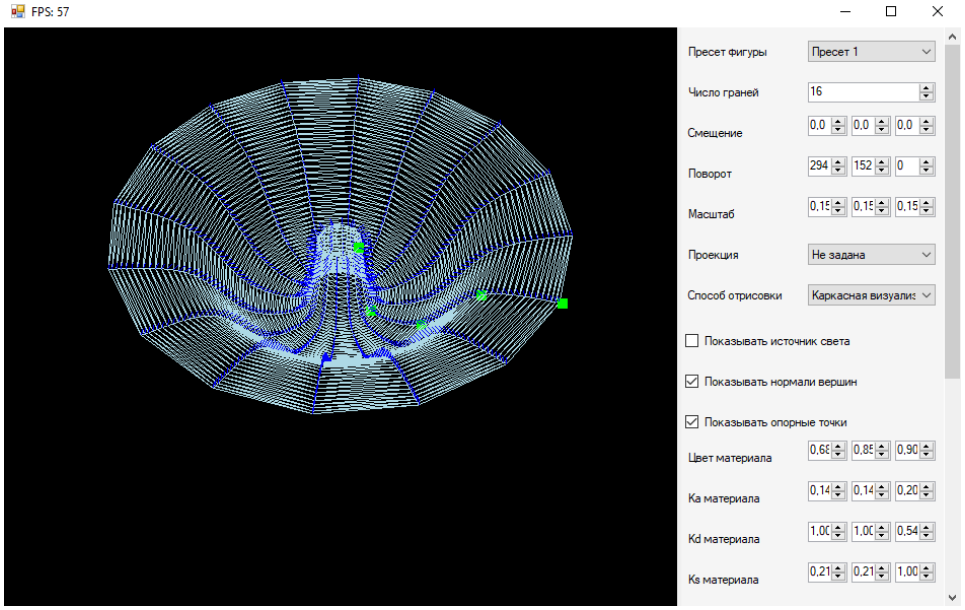


Рис. 3: Пресет №3, тип отрисовки - случайные цвета

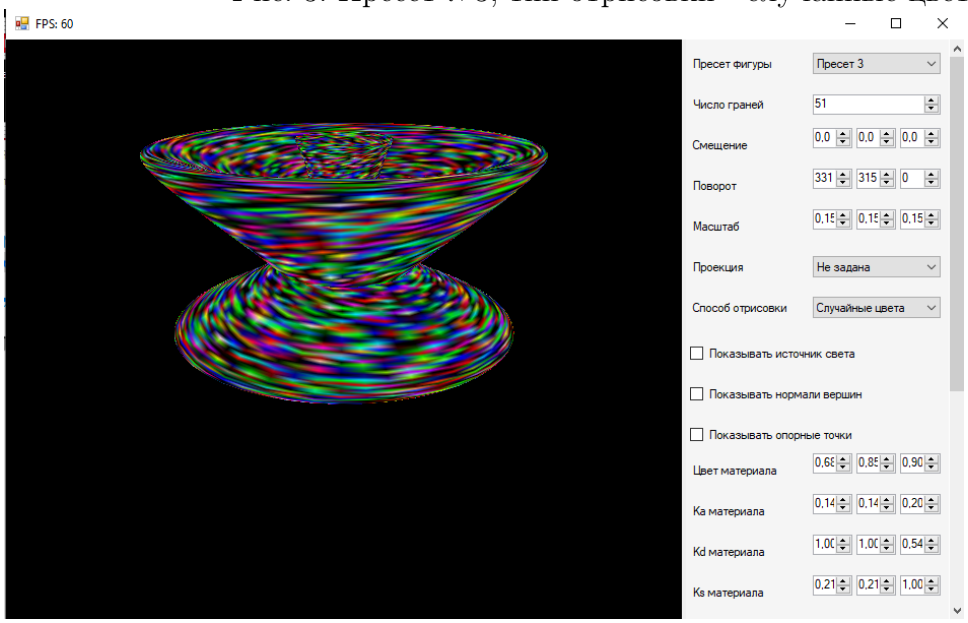
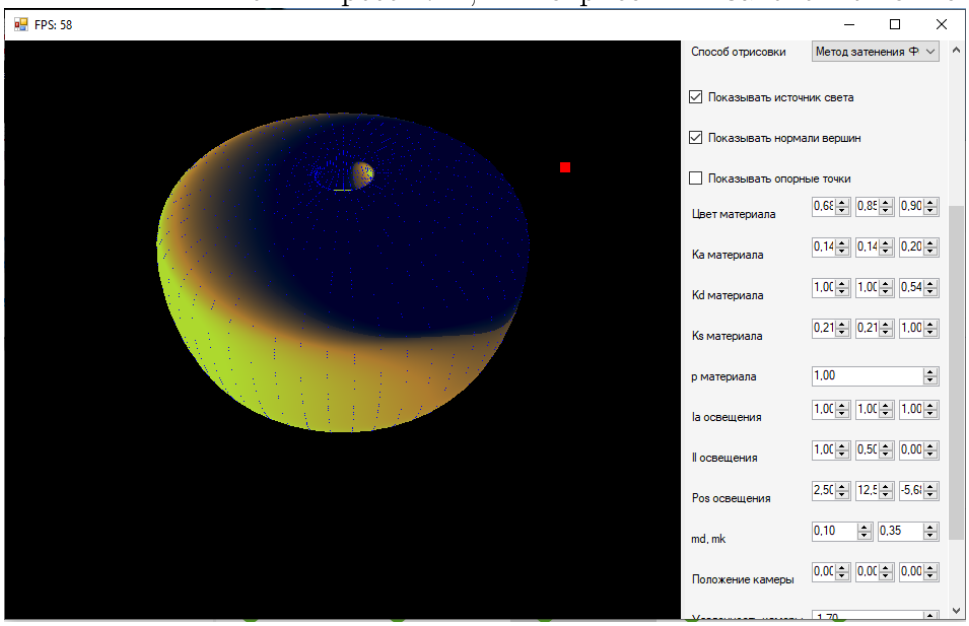


Рис. 4: Пресет №4, тип отрисовки - затенение по Фонгу



5 Выводы

В результате выполнения данной курсовой работы я познакомился с построением 3D фигур, имеющих сплайн-образующую как поверхность вращения. В ходе выполнения работы я закрепил полученные на курсе знания по аффинным преобразованиям поверхностей, проецированиям, вычислениям освещения, отрисовке с помощью средств OpenGL, в частности работе с буфером вершин и шейдерами.

Список литературы

[1] Голованов Н.Н. *Геометрическое моделирование* — Издательство Физико-математической литературы, 2002. — 472 с. (ISBN 5-94052-048-0)

[2] Е.В. Шишкин, А.И. Плис. *Кривые поверхности на экране компьютера. Руководство по сплайнам для пользователей* — Издательство Диалог-МИФИ, 1996. — 240 с. (ISBN 5-86404-080-0)