Национальный исследовательский институт
«Московский Авиационный Институт»

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ**

**Кафедра вычислительной математики и программирования**

# Курсовой проект

## по курсу «Компьютерная графика»

Тема: «Каркасная визуализация поверхности»

Студент: Куликов А.В.

Группа: М80-308Б

Преподаватель: Морозов А.В.

Оценка:

Москва-2019

**Постановка задачи**

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. 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.

Вариант 8: Линейчатая поверхность Кунса (границы – кубические кривые Безье 3D)

**Решение задачи**

***Поверхность Кунса***

Поверхность Кунса задается четырьмя контурными линиями $\mathrm{c}_0(s)$, $\mathrm{c}_1(s)$, $d_0(t)$ , $d_1(t)$, имеющими точки пересечения $\mathrm{c}_0(0) = d_0(0), \mathrm{c}_0(1) = d_1(0), \mathrm{c}_1(0) = d_0(1)$, $\mathrm{c}_1(1) = d_1(1)$ (угловые точки). Сама же поверхность Кунса образуется как сумма двух линейчатых поверхностей $L_c$, $L_d$ за вычетом поверхности $B$, построенной по точкам пересечения контурных линий. Дополнительная поверхность вычитается для компенсации того, что при сложении поверхностей граничные точки учитываются дважды, нарушая таким образом требование “натянутости” поверхности на граничные кривые.

Линейчатые поверхности $L_c$, $L_d$ представляют собой поверхности, строящиеся движением прямой линии по двум противолежащим контурным линиям. Т.о. для вычисления линейчатых поверхностей

Пользуемся линейной интерполяцией:

$$L_c(s,t) = (1-t)c_0(s) + tc_1(s)$$

$$L_d(s,t) = (1-s)d_0(t) + sd_1(t)$$

Результирующая поверхность должна проходить через граничные кривые и угловые точки, а т.к. при сложении поверхностей значения там были учтены дважды, то дополнительная вычитаемая поверхность должна совпадать с значениями с о значениями на границе и в угловых точках. Этого можно добиться применив билинейную интерполяцию:

$$B(s,t) = c_0(0)(1-s)(1-t) + c_0(1)s(1-t) + c_1(0)(1-s)t + c_1(1)st$$

Результирующую поверхность получаем следующим образом:

$$\mathrm{C}(s,t) = L_c(s,t) + L_d(s,t) - B(s,t)$$

***Кривые Безье***

Кривая Безье в общем виде задается многоугольником и имеет математическое параметрическое представление вида:

$$\sum_{i=0}^{n} B_i J_{n,i}(t), 0 \le t \le 1$$

где базис Безье или Бернштейна, или функция аппроксимации

$$J_{n,i} = \binom{n}{i} t^i (1-t)^{n-i}$$

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!}$$

$J_{n,i}$- это i-тая функция базиса Бернштейна порядка n. Здесь i — порядковый номер опорной вершины, n — порядок определяющей функции базиса Бернштейна — и, следовательно, сегмента полиномиальной кривой, на единицу меньше количества точек определяющего многоугольника.

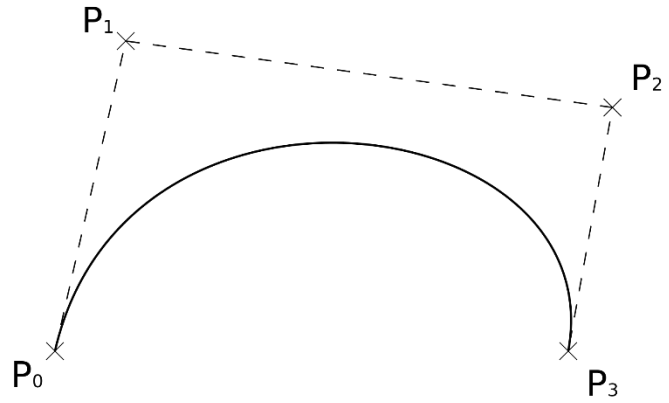



Рис. 2.1. Кривая Безье и, образующий ее, многоугольник

$\binom{n}{i}$ - биномиальные коэффициенты.
$B_i$ - функция компонент векторов опорных вершин (координаты вершин многоугольника Безье).

В данной работе был использован частный случай кривых Безье – кубические кривые.

В параметрической форме они задаются следующим образом:

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3, t \in [0,1],$$

где $P_0$, $P_1$, $P_2$, $P_3$ – опорные точки (вектора в 3-х мерном пространстве), задающие многоугольник, определяющий форму кривой.

Линия стартует в точке $P_0$, двигается в направлении $P_1$,отклоняясь к $P_2$, и, наконец, заканчивается в точке $P_3$. Кривая не проходит через точки $P_1, P_2$. Они используются для указания направления кривой.

***Аффинные преобразования***

Аффи́нное преобразование — отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся — в пересекающиеся, скрещивающиеся — в скрещивающиеся.

В общем виде могут быть заданы в виде матрицы преобразования.

$$f(x) = M \cdot x + v,$$

где M – матрица преобразования, v – вектор переноса.

На практике часто используются их следующие виды: преобразование поворота, преобразование масштабирования, преобразование переноса и т.д.

И, конечно же, они повсеместно используются в компьютерной графике.

**Описание программы**

В программе реализованы два основных класса BezierSpline, CoonsSurface, предоставляющие методы для вычисления координаты точки на поверхности, отрисовки как самих поверхностей и кривых, так и каркаса их задающего. На основе функционала этих классов и происходит расчет сетки поверхности для отображения. Так же для взаимодействия с пользователем реализовано управление мышью.

**Описание работы программы**

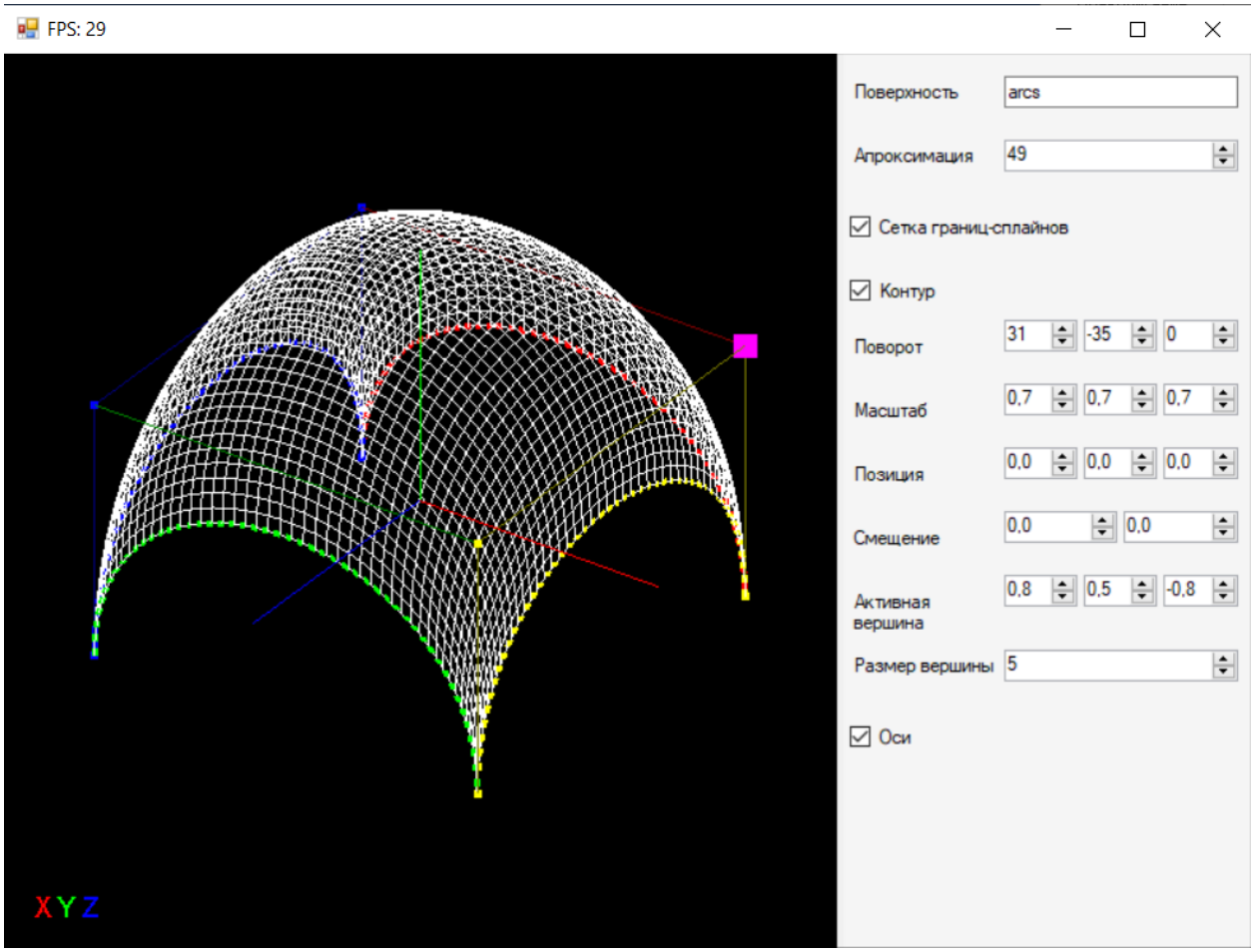
Программа позволяет визуализировать линейчатую поверхность Кунса, ограниченную кубическими кривыми Безье. Так же

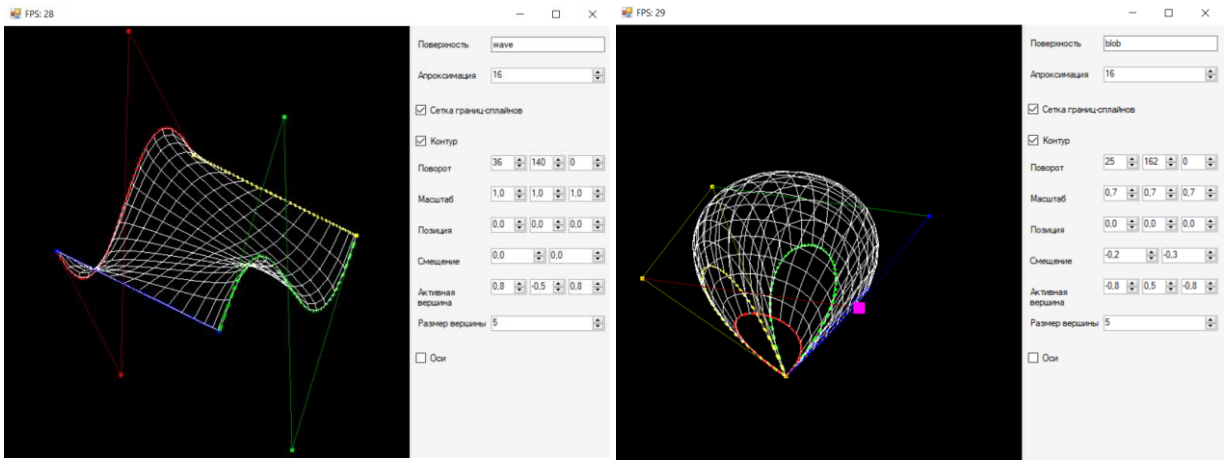
Чтобы изменить направления кривых, и саму поверхность соответственно, нужно кликом мышки по желаемой вершине выделить ее (подсветится розовым) и передвинуть используя поле “Активная вершина” на панели управления.

Так же программа позволяет изменить точность аппроксимации поверхности при помощи поле “Апроксимация”. Она отражается в количестве и плотности параметрических линий, образующих поверхность.

Программа предоставляет возможность аффинных преобразований поверхности: поворот, масштабирование, перемещение в пространстве при помощи полей “Поворот”, “Масштаб”, “Позиция” соответственно.

**Пример работы программы**






**Листинг кода**

Program.cs

```
using System;
using SharpGL;
using CGLabPlatform;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.IO;
using System.Linq;
using System.Text;

public abstract class CGLabDemoOGL : OGLApplicationTemplate<CGLabDemoOGL>
{
   [STAThread] static void Main() { RunApplication(); }

   #region Свойства

   [DisplayTextBoxProperty("arcs", "Поверхность")]
   public virtual string Prefix
   {
      get { return Get<string>(); }
      set { if (Set(value)) if (File.Exists(value + ".txt")) surface = new CoonsSurface(value + ".txt");
surface.ComputeSurface(ApproxLevel); ActiveVertex = null; }
   }

   [DisplayNumericProperty(4, 1, "Апроксимация", 3, 100)]
   public virtual int ApproxLevel
   {
      get { return Get<int>(); }
      set { if (Set(value)) if (surface != null) surface.ComputeSurface(value); }
   }

   [DisplayCheckerProperty(false, "Сетка границ-сплайнов")]
   public virtual bool Wireframe { get; set; }

   [DisplayCheckerProperty(false, "Контур")]
   public virtual bool Border { get; set; }

   [DisplayNumericProperty(new[] { 0d, 0d, 0d }, 1, "Поворот")]
   public virtual DVector3 Rotation
   {
      get { return Get<DVector3>(); }
      set { if (Set(value)) UpdateModelViewMatrix(); }
```

```
    }

    [DisplayNumericProperty(new[] { 1d, 1d, 1d }, 0.1, "Масштаб", 0.0)]
    public virtual DVector3 Scale
    {
       get { return Get<DVector3>(); }
       set { if (Set(value)) UpdateModelViewMatrix(); }
    }

    [DisplayNumericProperty(new[] { 0d, 0d, 0d }, 0.1, "Позиция")]
    public virtual DVector3 Position
    {
       get { return Get<DVector3>(); }
       set { if (Set(value)) UpdateModelViewMatrix(); }
    }

    [DisplayNumericProperty(new[] { 0d, 0d }, 0.1, "Смещение")]
    public virtual DVector2 Shift
    {
       get { return Get<DVector2>(); }
       set { if (Set(value)) UpdateModelViewMatrix(); }
    }

    [DisplayNumericProperty(new[] { 0d, 0d, 0d }, 0.1, "Активная вершина")]
    public virtual DVector3 ActiveVertexPosition
    {
       get { return Get<DVector3>(); }
       set { if (Set(value)) if (ActiveVertex != null)
            {
                ActiveVertex.Point = value;
                surface.ComputeSurface(ApproxLevel);
            }
       }
    }

    [DisplayNumericProperty(5, 1, "Размер вершины", 3, 20)]
    public virtual int VertexSize
    {
       get { return Get<int>(); }
       set { if (Set(value)) BezierSpline.VertexSize = value; }
    }

    [DisplayCheckerProperty(false, "Оси")]
    public virtual bool Axis { get; set; }

    #endregion

    public class Polygon
    {
       public DVector4 _Normal;
       public DVector4 Normal;

       public List<Vertex> Vertex;

       public int Color;

       public Polygon()
       {
          Vertex = new List<Vertex>();
```

```
    }
    public Polygon(List<Vertex> verts)
    {
        Vertex = verts;
        _Normal = CrossProduct(verts[0]._Point - verts[1]._Point, verts[1]._Point - verts[2]._Point);
        _Normal /= _Normal.GetLength();

        foreach (Vertex v in verts)
        {
            v.Polygon.Add(this);
        }
    }
}

private static DVector4 CrossProduct(DVector4 a, DVector4 b)
{
    double X = a.Y * b.Z - a.Z * b.Y;
    double Y = a.Z * b.X - a.X * b.Z;
    double Z = a.X * b.Y - a.Y * b.X;

    return new DVector4(X, Y, Z, 0.0);
}

public class Vertex
{
    public DVector4 _Point; // точка в локальной системе координат
    public DVector4 Point; // точка в мировой\видовой сиситеме координат

    public List<Polygon> Polygon;

    public DVector4 _Normal;
    public DVector4 Normal;

    public Vertex(DVector3 point)
    {
        Polygon = new List<Polygon>();
        _Point = new DVector4(point, 1.0);
        _Normal = DVector4.Zero;
    }
}

public class Vertex3
{
    public DVector3 Point { get; set; }
    public Vertex3(double x, double y, double z)
    {
        Point = new DVector3(x, y, z);
    }
}

public static DVector3 ReflectByPoint(DVector3 toReflect, DVector3 by)
{
    DVector3 v = by - toReflect;

    return by + v;
}

public class CoonsSurface {
    public BezierSpline border0;
```

```
public BezierSpline border1;
public BezierSpline border2;
public BezierSpline border3;

public Vertex[] vertices = null;
public Polygon[] polygons = null;

/*
      b0
    +-----+
    |     |
 b2 |     | b3
    +-----+
      b1
*/

public CoonsSurface(BezierSpline b0, BezierSpline b1, BezierSpline b2, BezierSpline b3)
{
   border0 = b0;
   border1 = b1;
   border2 = b2;
   border3 = b3;
}

public CoonsSurface(string path)
{
   StreamReader reader = new StreamReader(path);

   border0 = new BezierSpline(reader);
   border1 = new BezierSpline(reader);
   border2 = new BezierSpline(reader);
   border3 = new BezierSpline(reader);
}

DVector3 At(double s, double t)
{
   BezierSpline c0 = border0;
   BezierSpline c1 = border1;
   BezierSpline d0 = border2;
   BezierSpline d1 = border3;

   double c0Lim = c0.GetLimit();
   double c1Lim = c1.GetLimit();
   double d0Lim = d0.GetLimit();
   double d1Lim = d1.GetLimit();

   DVector3 Lc = c0.At(s * c0Lim) * (1 - t) + c1.At(s * c1Lim) * t;
   DVector3 Ld = d0.At(t * d0Lim) * (1 - s) + d1.At(t * d1Lim) * s;

   DVector3 B = c0.At(0.0 * c0Lim) * (1 - s) * (1 - t) + c0.At(1.0 * c0Lim) * s * (1 - t)
      + c1.At(0.0 * c1Lim) * (1 - s) * t + c1.At(1.0 * c1Lim) * s * t;

   return Lc + Ld - B;
}

public void ComputeSurface(int ApproxLevel)
{
   lock (locker)
   {
```

```
            double step = 1.0 / ApproxLevel;

            List<Vertex> v = new List<Vertex>();
            List<Polygon> p = new List<Polygon>();

            for (int i = 0; i <= ApproxLevel; ++i)
            {
               for (int j = 0; j <= ApproxLevel; ++j)
               {
                  DVector3 vec = At(i * step, j * step);
                  Vertex vert = new Vertex(vec);
                  v.Add(vert);
               }
            }

            for (int i = 0; i <= ApproxLevel - 1; ++i)
            {
               for (int j = 0; j <= ApproxLevel - 1; ++j)
               {
                  Polygon pol = new Polygon(new List<Vertex>() { v[i * (ApproxLevel + 1) + j], v[i * (ApproxLevel + 1)
+ j + 1], v[(i + 1) * (ApproxLevel + 1) + j + 1], v[(i + 1) * (ApproxLevel + 1) + j] });
                  p.Add(pol);
               }
            }

            vertices = v.ToArray();
            polygons = p.ToArray();
         }
      }

      object locker = new object();

      public void DrawByLines(OpenGL gl)
      {
         lock (locker)
         {
            int n = (int)Math.Sqrt(vertices.Length);
            for (int i = 0; i < n; ++i)
            {
               gl.Begin(OpenGL.GL_LINE_STRIP);
               for (int j = 0; j < n; ++j)
               {
                  DVector4 p = vertices[i * n + j]._Point;
                  gl.Vertex(p.X, p.Y, p.Z);
               }
               gl.End();
            }

            for (int i = 0; i < n; ++i)
            {
               gl.Begin(OpenGL.GL_LINE_STRIP);
               for (int j = 0; j < n; ++j)
               {
                  DVector4 p = vertices[j * n + i]._Point;
                  gl.Vertex(p.X, p.Y, p.Z);
               }
               gl.End();
            }
         }
```

```
    }

    public void DrawByQuads(OpenGL gl)
    {
        lock (locker)
        {
            gl.Begin(OpenGL.GL_QUADS);
            foreach (Polygon pol in polygons)
            {
                foreach (Vertex vert in pol.Vertex)
                {
                    DVector4 point = vert._Point;
                    gl.Vertex(point.X, point.Y, point.Z);
                }
            }
            gl.End();
        }
    }
}

static void SkipTabs(StreamReader sr)
{
    int c;
    while ((c = sr.Peek()) == ' ' || c == '\r' || c == '\t' || c == '\n')
    {
        if (c == -1)
        {
            return;
        }
        sr.Read();
    }
}

static string ReadStr(StreamReader sr)
{
    int c;
    string res = "";

    SkipTabs(sr);

    while ((c = sr.Read()) != ' ' && c != '\r' && c != '\t' && c != '\n' && c != -1)
    {
        res += Convert.ToChar(c);
    }

    SkipTabs(sr);

    return res;
}

public class BezierSpline
{
    public static float VertexSize;
    public List<Vertex3> vertices;
    public BezierSpline(List<Vertex3> verts)
    {
        vertices = verts;
    }
```

```
public BezierSpline(string path)
{
   StreamReader reader = new StreamReader(path);

   string line = ReadStr(reader);
   int n = Convert.ToInt32(line);

   List<Vertex3> v = new List<Vertex3>();
   for (int i = 0; i < n; ++i)
   {
      line = ReadStr(reader);
      double x = Convert.ToDouble(line);

      line = ReadStr(reader);
      double y = Convert.ToDouble(line);

      line = ReadStr(reader);
      double z = Convert.ToDouble(line);

      v.Add(new Vertex3(x, y, z));
   }

   vertices = v;
}

public BezierSpline(StreamReader reader)
{
   string line = ReadStr(reader);
   int n = Convert.ToInt32(line);

   List<Vertex3> v = new List<Vertex3>();
   for (int i = 0; i < n; ++i)
   {
      line = ReadStr(reader);
      double x = Convert.ToDouble(line);

      line = ReadStr(reader);
      double y = Convert.ToDouble(line);

      line = ReadStr(reader);
      double z = Convert.ToDouble(line);

      v.Add(new Vertex3(x, y, z));
   }

   vertices = v;
}

static private DVector3 VecAt(DVector3 p0, DVector3 p1, DVector3 p2, DVector3 p3, double t)
{
   double q = 1 - t;
   double q2 = q * q;
   double q3 = q2 * q;

   double t2 = t * t;
   double t3 = t2 * t;

   return q3 * p0 + 3 * t * q2 * p1 + 3 * t2 * q * p2 + t3 * p3;
}
```

```
public DVector3 At(double i)
{
   if (i < 0.0 || i > GetLimit())
   {
      return new DVector3(0.0, 0.0, 0.0);
   }


   if (i <= 1.0)
   {
      DVector3 p0 = vertices[0].Point;
      DVector3 p1 = vertices[1].Point;
      DVector3 p2 = vertices[2].Point;
      DVector3 p3 = vertices[3].Point;

      return VecAt(p0, p1, p2, p3, i);
   }
   else
   {
      int k = (int)i;

      if (i == k)
      {
         k--;
      }

      double t = i - k;

      DVector3 p0 = vertices[2 * k + 1].Point;
      DVector3 p1 = ReflectByPoint(vertices[2 * k].Point, p0);
      DVector3 p2 = vertices[2 * k + 2].Point;
      DVector3 p3 = vertices[2 * k + 3].Point;

      return VecAt(p0, p1, p2, p3, t);
   }
}

public double GetLimit()
{
   return (vertices.Count - 2) / 2.0;
}
public void Draw(OpenGL gl, DVector3 color)
{
   int lim = (int)GetLimit();

   gl.Color(color.X, color.Y, color.Z, 1.0);

   gl.LineWidth(4f);

   gl.Begin(OpenGL.GL_LINES);

   for (int i = 0; i <= lim * 100; ++i)
   {
      DVector3 v = At(i * 0.01);
      gl.Vertex(v.X, v.Y, v.Z);
   }

   gl.End();
```

```
      gl.LineWidth(1f);
    }

    private void DrawWF(OpenGL gl, uint mode, DVector3 color)
    {
      gl.PointSize(VertexSize);

      gl.Color(color.X, color.Y, color.Z, 1.0);

      gl.Begin(mode);

      gl.Vertex(vertices[0].Point.X, vertices[0].Point.Y, vertices[0].Point.Z);
      gl.Vertex(vertices[1].Point.X, vertices[1].Point.Y, vertices[1].Point.Z);
      gl.Vertex(vertices[2].Point.X, vertices[2].Point.Y, vertices[2].Point.Z);
      gl.Vertex(vertices[3].Point.X, vertices[3].Point.Y, vertices[3].Point.Z);

      gl.End();

      if (vertices.Count < 4)
        return;

      gl.Begin(mode);

      for (int i = 4; i < vertices.Count; i += 2)
      {
        DVector3 v0 = vertices[i - 1].Point;
        DVector3 v1 = ReflectByPoint(vertices[i - 2].Point, v0);
        DVector3 v2 = vertices[i].Point;
        DVector3 v3 = vertices[i + 1].Point;

        gl.Vertex(v0.X, v0.Y, v0.Z);
        gl.Vertex(v1.X, v1.Y, v1.Z);
        gl.Vertex(v2.X, v2.Y, v2.Z);
        gl.Vertex(v3.X, v3.Y, v3.Z);
      }

      gl.End();

      gl.Color(1.0, 1.0, 1.0, 1.0);

      gl.PointSize(1f);
    }

    public void DrawWireframe(OpenGL gl, DVector3 color)
    {
      DrawWF(gl, OpenGL.GL_LINE_STRIP, color * 0.5);
      DrawWF(gl, OpenGL.GL_POINTS, color);
    }

  }

  DVector2 NormalizeCoords(double x, double y)
  {
    double H = base.RenderDevice.Height;
    double W = base.RenderDevice.Width;

    double normX, normY;
    if (W > H)
```

```
    {
      normX = x / H * 2f - W / H;
      normY = -y / H * 2f + 1f;
    }
    else
    {
      normX = x / W * 2f - 1f;
      normY = -y / W * 2f + H / W;
    }

    return new DVector2(normX, normY);
  }

  double NormalizeDistance(double dist)
  {
    DVector2 tmp = NormalizeCoords(0, dist) - NormalizeCoords(0, 0);

    double normDist = Math.Sqrt(tmp.X * tmp.X + tmp.Y * tmp.Y);
    return normDist;
  }

  DMatrix4 transformationMatrix;

  protected override void OnMainWindowLoad(object sender, EventArgs args)
  {
    base.VSPanelWidth = 260;
    base.ValueStorage.RightColWidth = 60;
    base.RenderDevice.VSync = 1;

    #region Обработчики событий мыши и клавиатуры ----------------------------------------------------------
    RenderDevice.MouseMoveWithRightBtnDown += (s, e) => Rotation += new DVector3(e.MovDeltaY,
e.MovDeltaX, 0);

    RenderDevice.MouseMoveWithMiddleBtnDown += (s, e) => {

      double H = base.RenderDevice.Height;
      double W = base.RenderDevice.Width;

      double AspectRatio = W / H;

      if (W > H)
      {
        Shift = new DVector2(Shift.X + 2 * e.MovDeltaX / W * AspectRatio, Shift.Y - 2 * (double)e.MovDeltaY /
H);
      }
      else
      {
        Shift = new DVector2(Shift.X + 2 * e.MovDeltaX / W , Shift.Y - 2 * (double)e.MovDeltaY / H /
AspectRatio);
      }
    };

    RenderDevice.MouseLeftBtnDown += (s, e) => {

      DVector2 norm = NormalizeCoords(e.Location.X, e.Location.Y);

      double H = base.RenderDevice.Height;
      double W = base.RenderDevice.Width;
```

```
    double AspectRatio = W / H;

    double X, Y;
    if (W > H)
    {
      X = (norm.X) / AspectRatio;
      Y = norm.Y;
    }
    else
    {
      X = norm.X;
      Y = (norm.Y) * AspectRatio;
    }

    DVector4 mousePos = new DVector4(X, Y, 0.0, 0.0);

    lock (ActiveVertexLocker)
    {
      ActiveVertex = FindClosestVertex(transformationMatrix, mousePos);

      if (ActiveVertex != null)
        ActiveVertexPosition = ActiveVertex.Point;
      else
        ActiveVertexPosition = DVector3.Zero;
    }
  };

  RenderDevice.MouseLeftBtnUp += (s, e) => { };
  RenderDevice.MouseRightBtnDown += (s, e) => {

  };

  RenderDevice.MouseRightBtnUp += (s, e) =>
  {

  };
  RenderDevice.MouseMove += (s, e) =>
  {

  };

  #endregion


  #region  Инициализация OGL и параметров рендера ----------------------------------------------------
  RenderDevice.AddScheduleTask((gl, s) =>
  {
    gl.FrontFace(OpenGL.GL_CCW);
    gl.Enable(OpenGL.GL_CULL_FACE);
    gl.CullFace(OpenGL.GL_BACK);

    gl.ClearColor(0, 0, 0, 0);

    gl.Enable(OpenGL.GL_DEPTH_TEST);
    gl.DepthFunc(OpenGL.GL_LEQUAL);
    gl.ClearDepth(1.0f);    // 0 - ближе, 1 - далеко
    gl.ClearStencil(0);

  });
```

```
      #endregion

      #region Обновление матрицы проекции при изменении размеров окна и запуске приложения ----------
------
      RenderDevice.Resized += (s, e) =>
      {
         var gl = e.gl;

         UpdateProjectionMatrix(gl);
      };
      #endregion

      surface = new CoonsSurface(Prefix + ".txt");

      surface.ComputeSurface(ApproxLevel);
   }

   CoonsSurface surface;

   private void UpdateProjectionMatrix(OpenGL gl)
   {
      #region Обновление матрицы проекции ------------------------------------------------------------

      double H = gl.RenderContextProvider.Height;
      double W = gl.RenderContextProvider.Width;

      double AspectRatio = W / H;

      gl.MatrixMode(OpenGL.GL_PROJECTION);

      gl.LoadIdentity();

      if (W > H)
      {
         gl.Ortho(-1.0f * AspectRatio, 1.0 * AspectRatio, -1.0, 1.0, -100.0, 100.0);
      }
      else
      {
         gl.Ortho(-1.0f, 1.0, -1.0 / AspectRatio, 1.0 / AspectRatio, -100.0, 100.0);
      }


      #endregion
   }

   private void UpdateModelViewMatrix() // метод вызывается при измении свойств cameraAngle и
cameraDistance
   {
      #region Обновление объектно-видовой матрицы ----------------------------------------------------
      RenderDevice.AddScheduleTask((gl, s) => {

         gl.MatrixMode(OpenGL.GL_MODELVIEW);
         gl.LoadIdentity();

         gl.Translate(Shift.X, Shift.Y, 0.0);
         gl.Rotate(Rotation.X, 1.0, 0.0, 0.0);
         gl.Rotate(Rotation.Y, 0.0, 1.0, 0.0);
         gl.Rotate(Rotation.Z, 0.0, 0.0, 1.0);
         gl.Translate(Position.X, Position.Y, Position.Z);
```

```
        gl.Scale(Scale.X, Scale.Y, Scale.Z);

    });
    #endregion
}

private double ToRadians(double angle)
{
    return Math.PI * angle / 180.0;
}

private void DrawAxis(OpenGL gl)
{
    gl.MatrixMode(OpenGL.GL_MODELVIEW);
    gl.PushMatrix();

    gl.LoadIdentity();

    gl.Translate(Shift.X, Shift.Y, 0.0);
    gl.Rotate(Rotation.X, 1.0, 0.0, 0.0);
    gl.Rotate(Rotation.Y, 0.0, 1.0, 0.0);
    gl.Rotate(Rotation.Z, 0.0, 0.0, 1.0);

    gl.Disable(OpenGL.GL_DEPTH_TEST);

    gl.Begin(OpenGL.GL_LINES);

    const float AxisLength = 0.7f;

    // X-axis
    gl.Color(1f, 0f, 0f);
    gl.Vertex(0f, 0f, 0f);
    gl.Vertex(AxisLength, 0f, 0f);

    // Y-axis
    gl.Color(0f, 1f, 0f);
    gl.Vertex(0f, 0f, 0f);
    gl.Vertex(0f, AxisLength, 0f);

    // Z-axis
    gl.Color(0f, 0f, 1f);
    gl.Vertex(0f, 0f, 0f);
    gl.Vertex(0f, 0f, AxisLength);

    gl.End();

    gl.Enable(OpenGL.GL_DEPTH_TEST);

    gl.MatrixMode(OpenGL.GL_MODELVIEW);
    gl.PopMatrix();
}


Vertex3 ActiveVertex = null;

Vertex3 FindClosestVertex(DMatrix4 mat, DVector4 vec)
{
    List<Vertex3> vs = new List<Vertex3>();
```

```
double radius = NormalizeDistance(VertexSize);

foreach(Vertex3 v in surface.border0.vertices)
{
   DVector4 tmp = mat * new DVector4(v.Point, 1.0);
   DVector2 d = new DVector2(tmp.X, tmp.Y) - new DVector2(vec.X, vec.Y);

   if (d.GetLength() < radius)
   {
      vs.Add(v);
   }
}

foreach (Vertex3 v in surface.border1.vertices)
{
   DVector4 tmp = mat * new DVector4(v.Point, 1.0);
   DVector2 d = new DVector2(tmp.X, tmp.Y) - new DVector2(vec.X, vec.Y);

   if (d.GetLength() < radius)
   {
      vs.Add(v);
   }
}

foreach (Vertex3 v in surface.border2.vertices)
{
   DVector4 tmp = mat * new DVector4(v.Point, 1.0);
   DVector2 d = new DVector2(tmp.X, tmp.Y) - new DVector2(vec.X, vec.Y);

   if (d.GetLength() < radius)
   {
      vs.Add(v);
   }
}

foreach (Vertex3 v in surface.border3.vertices)
{
   DVector4 tmp = mat * new DVector4(v.Point, 1.0);
   DVector2 d = new DVector2(tmp.X, tmp.Y) - new DVector2(vec.X, vec.Y);

   if (d.GetLength() < radius)
   {
      vs.Add(v);
   }
}

Vertex3 res = null;
double minDist = 100000000;

foreach (Vertex3 v in vs)
{
   DVector4 tmp = mat * new DVector4(v.Point, 1.0);
   DVector2 d = new DVector2(tmp.X, tmp.Y) - new DVector2(vec.X, vec.Y);

   double dist = d.GetLength();

   if(dist < minDist)
   {
      res = v;
```

```
            minDist = dist;
        }
    }

    return res;
}

object ActiveVertexLocker = new object();

protected unsafe override void OnDeviceUpdate(object s, OGLDeviceUpdateArgs e)
{
    var gl = e.gl;

    // Очищаем буфер экрана и буфер глубины (иначе рисоваться все будет поверх старого)
    gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT |
OpenGL.GL_STENCIL_BUFFER_BIT);

    float[] modelMat = new float[16];
    gl.GetFloat(OpenGL.GL_MODELVIEW_MATRIX, modelMat);

    float[] projectionMat = new float[16];
    gl.GetFloat(OpenGL.GL_PROJECTION_MATRIX, projectionMat);

    DMatrix4 mM = new DMatrix4(modelMat.Select(val => (double)val).ToArray());
    DMatrix4 pM = new DMatrix4(projectionMat.Select(val => (double)val).ToArray());

    mM.Transpose();
    pM.Transpose();

    transformationMatrix = pM * mM;

    if (Wireframe || Border)
    {
        DVector3 b0Color = new DVector3(1.0, 0.0, 0.0);
        DVector3 b1Color = new DVector3(0.0, 1.0, 0.0);
        DVector3 b2Color = new DVector3(0.0, 0.0, 1.0);
        DVector3 b3Color = new DVector3(1.0, 1.0, 0.0);

        if (Border)
        {
            surface.border0.Draw(gl, b0Color);
            surface.border1.Draw(gl, b1Color);
            surface.border2.Draw(gl, b2Color);
            surface.border3.Draw(gl, b3Color);
        }

        if (Wireframe)
        {
            surface.border0.DrawWireframe(gl, b0Color);
            surface.border1.DrawWireframe(gl, b1Color);
            surface.border2.DrawWireframe(gl, b2Color);
            surface.border3.DrawWireframe(gl, b3Color);
        }
    }

    gl.Color(1.0, 1.0, 1.0, 1.0);

    surface.DrawByLines(gl);
```

```
      if (Axis)
      {
         DrawAxis(gl);

         gl.DrawText(20, 20, 1.0f, 0.0f, 0.0f, "Arial", 16, "X");
         gl.DrawText(35, 20, 0.0f, 1.0f, 0.0f, "Arial", 16, "Y");
         gl.DrawText(50, 20, 0.0f, 0.0f, 1.0f, "Arial", 16, "Z");
      }

      lock (ActiveVertexLocker)
      {
         if (ActiveVertex != null)
         {
            gl.Color(1.0, 0.0, 1.0, 1.0);
            gl.PointSize(15f);
            gl.Begin(OpenGL.GL_POINTS);
            gl.Vertex(ActiveVertex.Point.X, ActiveVertex.Point.Y, ActiveVertex.Point.Z);
            gl.End();
            gl.Color(1.0, 1.0, 1.0, 1.0);
            gl.PointSize(1f);
         }
      }
      return;
   }
}
```

**Выводы**

Выполнив курсовой проект, я научился реализовывать поверхность Кунса и кубические кривые Безье. Поверхности Кунса и кривые Безье, несмотря на относительную простоту реализации, по-видимому, являются довольно мощными инструментами. Эти виды поверхностей и кривые широко применяются на практике, например, в САПРах и архитектуре.

Данный курс и в частности эта работа очень сильно помог мне разобраться во многих вещах, связанных с машинной графикой.

Приобретенные знания и умения явно не будут лишними и весьма вероятно будут использованы мной в дальнейшем.