МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторные работы № 1 - 7
по курсу «Компьютерная графика»

Группа: М8О-307Б-19
Студент: Бирюков В. В.
Преподаватель: Морозов А. В.
Оценка:
Дата:

Москва, 2021

Лабораторная работа №1

Постановка задачи

Тема: Построение изображений 2D- кривых.
Задание: Написать и отладить программу, строящую изображение заданной замечательной кривой. Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.
Вариант: $(x^2+y^2)^2=a^2(x^2-y^2)$

Описание

Аппроксимируем кривую кусочно-гладкой функцией. Изменяя x с заданным шагом аппроксимации будем вычислять значения функции. В данном варианте x изменяется в пределах от -a до a и каждому значению соответствует два значения y.
Полученные точки необходимо перевести в экранные координаты, применив к ним аффинные преобразования, задаваемые пользователем, сдвинув в центр экрана и обратив ось Y. Затем соединяем точки при помощи стандартных функций Cairo.

Исходный код

Генерация точек

```
private void CalculateDots()
{
    _dots.Clear();

    float start = (float) -_a.Value, end = (float) _a.Value;
    float d = (end - start) / (float) _stepsCount.Value;
    for (float x = start; x <= end; x += d)
    {
        float t = 0.5f * (float) (-Math.Pow(_a.Value, 2) - 2 * x * x +
          Math.Sqrt(Math.Pow(_a.Value, 4) + 8 * Math.Pow(_a.Value, 2) * x * x));
        if (Math.Abs(t) < 1e-6) t = 0;
        _dots.Add(new Vector2(x, (float) Math.Sqrt(t)));
        _dots.Insert(0, new Vector2(x, (float) -Math.Sqrt(t)));
    }
    _dots.Add(_dots[0]);
}
```

Трансформирование точки в экранные координаты

```
private Vector2 TransformDot(Vector2 dot, bool scale = true,
                             bool shiftToCenter = true, bool shiftXY = true,
                             bool rotate = true, bool fixY = true)
{
 var scaleMatrix = Matrix3x2.Identity * (float)_scale.Value;
 Vector2 center = new Vector2((float)_rCenterX.Value,(float)_rCenterY.Value);
 float cos = (float) Math.Cos(_angle.Value * Math.PI / 180);
 float sin = (float) Math.Sin(_angle.Value * Math.PI / 180);
 var rotationMatrix = new Matrix3x2(cos, sin, -sin, cos, 0, 0);
 var shift = new Vector2((float) -_shiftX.Value, (float) -_shiftY.Value);
```

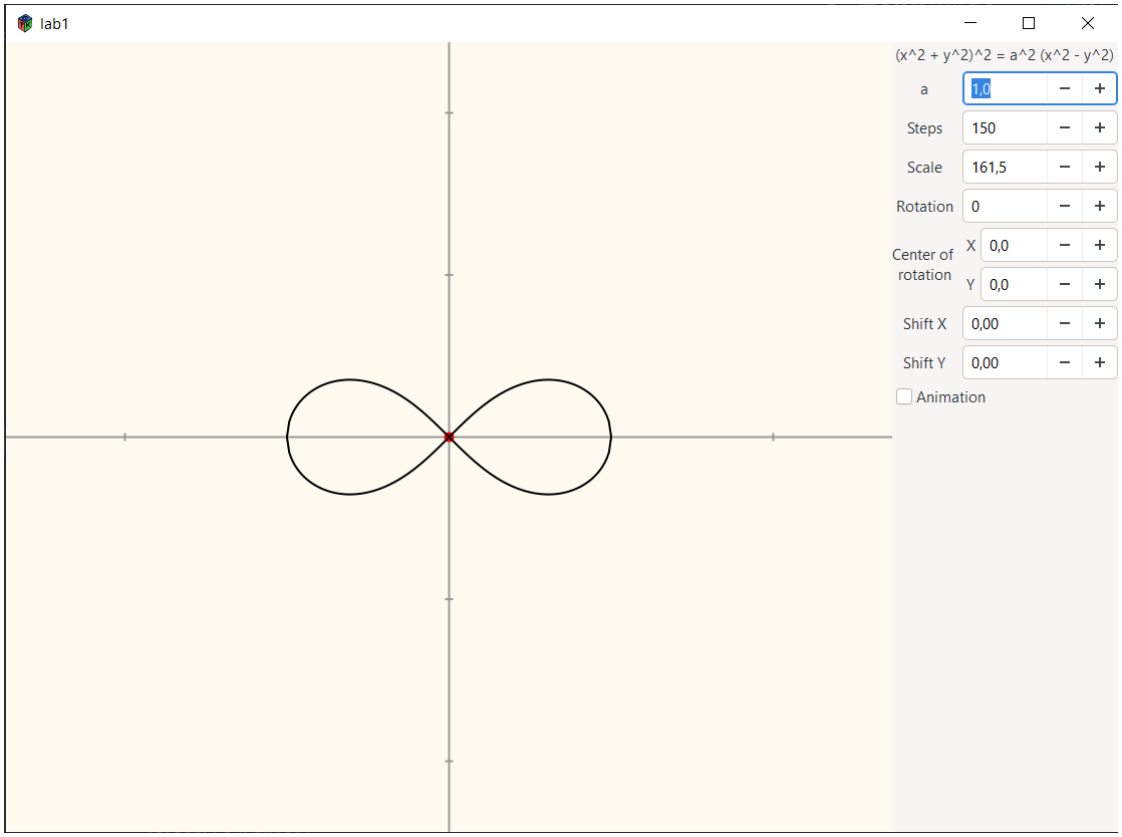
```
    if (rotate) dot = Vector2.Transform(dot - center, rotationMatrix) + center;
    if (shiftXY) dot += shift;
    if (scale) dot = Vector2.Transform(dot, scaleMatrix);
    if (fixY) dot.Y *= -1;
    if (shiftToCenter) dot += _canvasSize * 0.5f;
    return dot;
}
```

Отрисовка кривой

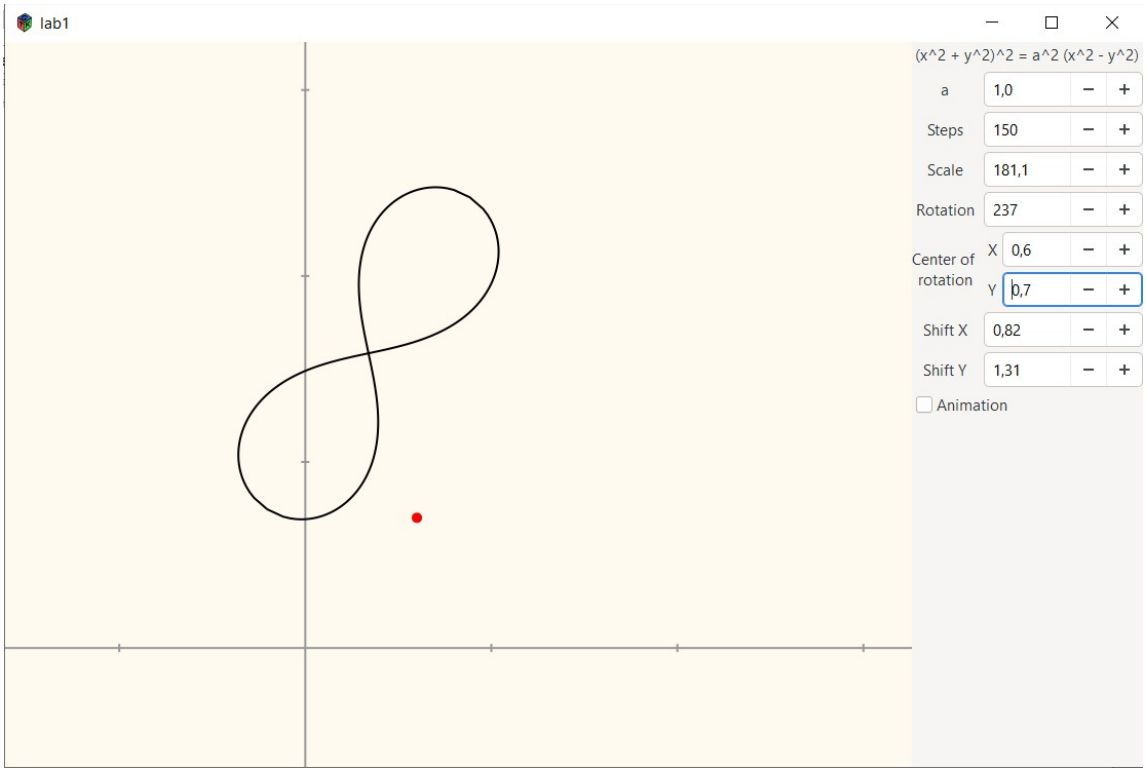
```
private void CanvasOnDrawn(object o, DrawnArgs args)
{
    var cr = args.Cr;
    cr.Antialias = Antialias.Subpixel;

    cr.SetSourceRGB(1, 0.98, 0.94);
    cr.Paint();

    DrawAxis(cr);
    DrawRotationCenter(cr);

    cr.SetSourceRGB(0, 0, 0);
    cr.MoveTo(TransformDot(_dots[0]));
    int max = _animateButton.Active ? _maxSteps : _dots.Count;
    for (int i = 0; i < Math.Min(max, _dots.Count); ++i)
    {
        cr.LineTo(TransformDot(_dots[i]));
    }
    cr.Stroke();
}
private void DrawRotationCenter(Context cr)
{
    Vector2 center = TransformDot(new Vector2((float)_rCenterX.Value,
                                              (float)_rCenterY.Value));
    cr.SetSourceRGB(1, 0, 0);
    cr.Arc(center.X, center.Y, 5, 0, 2 * Math.PI);
    cr.Fill();
    cr.Stroke();
}
private void DrawAxis(Context cr)
{
    cr.SetSourceRGB(0.6, 0.6, 0.6);

    var halfX = _canvasSize.X / 2 - _shiftX.Value * _scale.Value;
    var halfY = _canvasSize.Y / 2 + _shiftY.Value * _scale.Value;
    cr.MoveTo(0, halfY);
    cr.LineTo(_canvasSize.X, halfY);
    cr.MoveTo(halfX, 0);
    cr.LineTo(halfX, _canvasSize.Y);
    cr.Stroke();
    double size = 8;
    for (var x = halfX + _scale.Value; x < _canvasSize.X; x += _scale.Value)
        cr.Line(x, halfY + size / 2, x, halfY - size / 2);
    for (var x = halfX - _scale.Value; x >= 0 ; x -= _scale.Value)
        cr.Line(x, halfY + size / 2, x, halfY - size / 2);}
    for (var y = halfY + _scale.Value; y < _canvasSize.Y; y += _scale.Value)
        cr.Line(halfX + size / 2, y, halfX - size / 2, y);
    for (var y = halfY - _scale.Value; y >= 0 ; y -= _scale.Value)
        cr.Line(halfX + size / 2, y, halfX - size / 2, y);
}
```

Демонстрация работы программы

Изображение кривой



Вращение относительно произвольного центра



Лабораторная работа №2

Постановка задачи

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.
Задание: Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.
Варианты: Правильный октаэдр.

Описание

Для представления трехмерного объекта используется класс вершин и класс полигонов. Каждая вершина хранит свои координаты в локальном и мировом пространстве. Каждый полигон хранит все вершины, из который он состоит, а также вектор нормали в локальном и мировом пространстве.
Для преобразования в экранные координаты все точки умножаются на матрицу модели и видовую матрицу, которая производит корректировку центра и оси Y.
Для удаления невидимых линий, не происходит отрисовка полигонов, чья нормаль сонаправлена направлению взгляда наблюдателя, которое принимается за отрицательное направление оси Z.
Правильный октаэдр можно задать при помощи 6 точек и 8 граней.
Также в программе реализована отрисовка нормалей и преобразование из формата obj во внутреннее представление, что позволяет загрузить любой трехмерный объект.

Исходный код

Составление матрицы модели

```
private void CalculateWorldMatrix()
{
    Matrix4x4 rotation;
    if (_projections.Active == (int)Projection.Isometric)
    {
        rotation = Matrix4x4.CreateRotationY((float) (45 * Math.PI / 180)) *
                   Matrix4x4.CreateRotationX((float) (35 * Math.PI / 180));
    }
    else if (_projections.Active == (int)Projection.Dimetric)
    {
        rotation = Matrix4x4.CreateRotationY((float) (26 * Math.PI / 180)) *
                   Matrix4x4.CreateRotationX((float) (30 * Math.PI / 180));
    }
    else
    {
        rotation = Matrix4x4.CreateRotationX((float)(_xAngle.Value*Math.PI/180))
                * Matrix4x4.CreateRotationY((float)(_yAngle.Value*Math.PI/180))
                * Matrix4x4.CreateRotationZ((float)(_zAngle.Value*Math.PI/180));
    }
```

```
    var translation = Matrix4x4.CreateTranslation((float) _xShift.Value,
                          (float) _yShift.Value, (float) _zShift.Value);
    var scale = Matrix4x4.CreateScale((float) _xScale.Value,
                          (float) _yScale.Value, (float) _zScale.Value);

    var projectionMatrix = Matrix4x4.Identity;
    if (_projections.Active == (int)Projection.Front)
        projectionMatrix.M33 = 0;
    else if (_projections.Active == (int)Projection.Top)
        projectionMatrix.M22 = 0;
    else if (_projections.Active == (int)Projection.Right)
        projectionMatrix.M11 = 0;

    _worldMatrix = projectionMatrix * scale * rotation * translation;

    UpdateMatrixView();
    TransformToWorld();
    _canvas.QueueDraw();
}
```

Преобразование в мировое и видовое пространства

```
private void TransformToWorld()
{
    foreach (var vertex in _vertices)
    {
        vertex.PointInWorld = Vector4.Transform(vertex.Point, _worldMatrix);
    }

    var normalMatrix = TransposeInvert(_worldMatrix);
    foreach (var polygon in _polygons)
    {
        polygon.NormalInWorld = Vector4.Transform(polygon.Normal, normalMatrix);
    }

    if (_zBuffer.Active)
        _polygons = _polygons.OrderBy((p) =>
                    p.Vertices.Select((v) => v.PointInWorld.Z).Max()).ToList();
}

private Vector2 TransformToView(Vector4 point)
{
    Vector4 dot = Vector4.Transform(point, _viewMatrix);
    return new Vector2(dot.X, dot.Y);
}
```

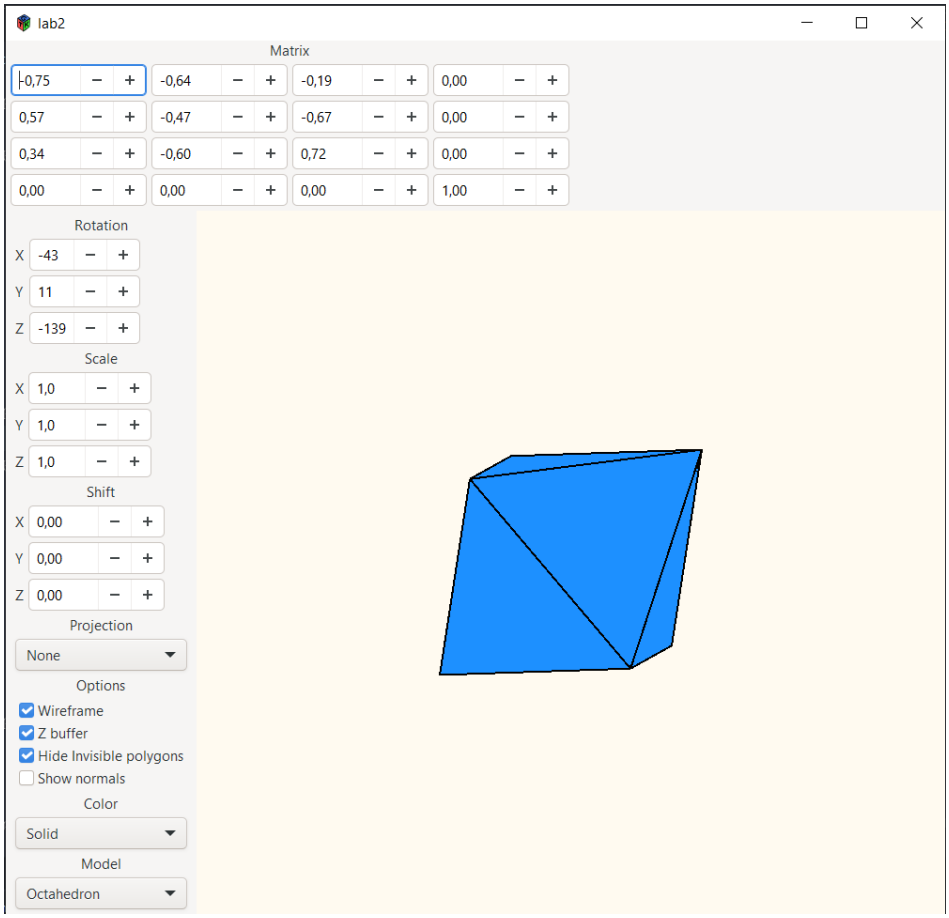
Отрисовка

```
private readonly Vector4 _viewDirection = new Vector4(0, 0, -1, 0);
private const int NormalLength = 20;

private void CanvasDrawnHandler(object o, DrawnArgs args)
{
    var cr = args.Cr;
    cr.Antialias = Antialias.None;
    cr.LineJoin = LineJoin.Bevel;

    cr.SetSourceColor(BACKGROUND_COLOR);
    cr.Paint();
```

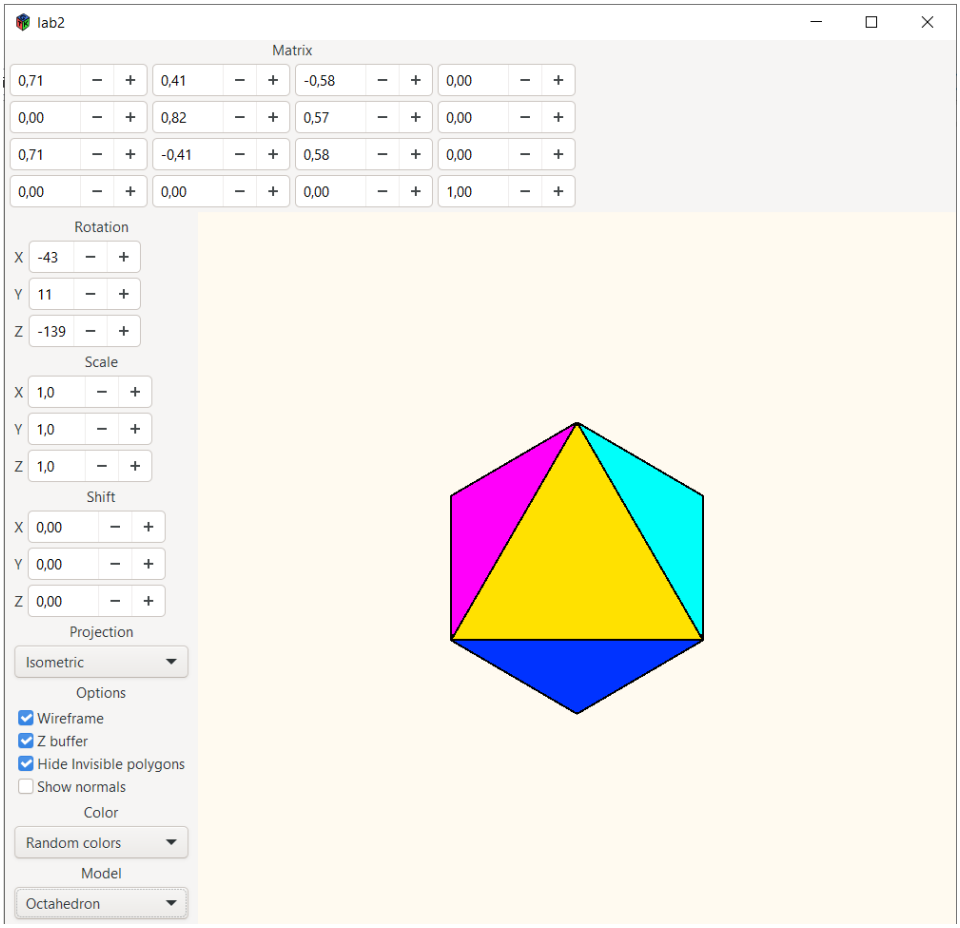
```
    cr.SetSourceColor(LINE_COLOR);
    foreach (var polygon in _polygons)
    {
        if (Vector4.Dot(polygon.NormalInWorld, _viewDirection) > 0
            && _hideInvisible.Active)
            continue;

        cr.MoveTo(TransformToView(polygon.Vertices[0].PointInWorld));
        for (int i = 1; i < polygon.Vertices.Length; ++i)
        {
            cr.LineTo(TransformToView(polygon.Vertices[i].PointInWorld));
        }
        cr.ClosePath();
        cr.Save();
        if (_fillPolygons)
        {
            cr.SetSourceRGB(polygon.Color.X, polygon.Color.Y, polygon.Color.Z);
            cr.FillPreserve();
        }
        cr.Restore();
        if (_wireframe.Active)
            cr.Stroke();
        else
            cr.NewPath();

        if (_showNormals.Active)
        {
            cr.Save();
            cr.SetSourceColor(NORMAL_COLOR);
            Vector4 center = Vector4.Zero;
            foreach (var vertex in polygon.Vertices)
            {
                center += vertex.PointInWorld;
            }

            center /= polygon.Vertices.Length;
            cr.MoveTo(TransformToView(center));
            var viewNormal = TransformToView(polygon.NormalInWorld);
            if (viewNormal.Length() > NormalLength)
            {
                viewNormal = Vector2.Normalize(viewNormal) * NormalLength;
            }
            cr.RelLineTo(viewNormal);
            cr.Stroke();
            cr.Restore();
        }
    }
}
```

Демонстрация работы программы

Октаэдр



Изометрическая проекция



Лабораторная работа №3

Постановка задачи

Тема: Основы построения фотореалистичных изображений.
Задание: Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.
Вариант: Прямой эллиптический цилиндр.

Описание

Аппроксимируем цилиндр прямой призмой с правильный основанием. Параметры аппроксимации задают количество боковых граней и число разбиений этих граней.
Класс вершины теперь хранит также нормаль и полигоны, в которые входит вершина.
Световые свойства объекта хранятся в классе материала.
Используем простую модель освещения. $I = I_a + I_d + I_s$, где I_a - фоновая составляющая, I_d - рассеянная составляющая, I_s - зеркальная составляющая.
В программе реализована плоская модель затенения, при которой каждый полигон закрашивается в цвет, вычисляемый в его центре, и модель затенения Гуро, в которой каждый треугольный полигон закрашивается трехцветным градиентом по цвету вершин.
Так как полигоны могут иметь больше трех вершин, при отрисовке выполняется их триангуляция.

Исходный код

Генерация цилиндра

```
public static void Prism(int sidesX, int sidesY, float height, float radius,
Mesh mesh, Material material)
{
    mesh.Vertices.Clear();
    mesh.Polygons.Clear();
    var dh = height / sidesY;
    var dphi = 2 * Math.PI / sidesX;
    height /= 2;
    var rotation = Matrix4x4.CreateRotationY((float) dphi);
    var highCenter = new Vertex(0, height, 0);
    var lowCenter = new Vertex(0, height - dh * sidesY, 0);
    mesh.Vertices.Add(new Vertex(radius, height, 0));
    for (int i = 1; i < sidesX; ++i)
    {
        mesh.Vertices.Add(new Vertex(Vector4.Transform(
                          mesh.Vertices.Last().Point, rotation)));
        mesh.Polygons.Add(new Polygon(mesh.Vertices[i-1],
                          mesh.Vertices[i], highCenter, material));
    }
    mesh.Polygons.Add(new Polygon(mesh.Vertices[sidesX-1],
```

```
                                    mesh.Vertices[0], highCenter, material));

    for (int i = 1; i < sidesY+1; ++i)
    {
        for (int j = 0; j < sidesX; ++j)
        {
            mesh.Vertices.Add(new Vertex(
                                        mesh.Vertices[(i-1)*sidesX + j].Point));
            mesh.Vertices[^1].Point.Y -= dh;
        }
    }

    for (int i = 1; i < sidesY+1; ++i)
    {
        for (int j = 1; j < sidesX; ++j)
        {
            mesh.Polygons.Add(new Polygon(mesh.Vertices[(i-1)*sidesX + j],
               mesh.Vertices[(i-1)*sidesX + j-1], mesh.Vertices[i*sidesX + j-1],
               mesh.Vertices[i*sidesX + j], material));
        }
        mesh.Polygons.Add(new Polygon(mesh.Vertices[(i-1)*sidesX],
                           mesh.Vertices[(i-1)*sidesX + sidesX-1],
                           mesh.Vertices[i*sidesX + sidesX-1],
                           mesh.Vertices[i*sidesX], material));
    }

    for (int i = 1; i < sidesX; ++i)
    {
        mesh.Polygons.Add(new Polygon(mesh.Vertices[^i],
                          mesh.Vertices[^(i+1)], lowCenter, material));
    }
    mesh.Polygons.Add(new Polygon(mesh.Vertices[^sidesX],
                      mesh.Vertices[^1], lowCenter, material));
    mesh.Vertices.Add(highCenter);
    mesh.Vertices.Add(lowCenter);
    mesh.CalculateVerticesNormals();
}
```

Вычисление цвета в точке

```
private Vector3 GetPointColor(Vector4 point, Vector4 normal, Material material)
{
    Vector3 ambient = _ambientLight.Intensity * material.Ka;

    Vector4 light = Vector4.Transform(_pointLight.Point, _worldMatrix);
    Vector4 toLight = Vector4.Normalize(light - point);
    Vector3 diffuse = _pointLight.Intensity * material.Kd *
                      Math.Max(Vector4.Dot(toLight, normal), 0);

    Vector4 reflect = Vector4.Normalize(2 * Vector4.Dot(toLight, normal) *
                                        normal - toLight);
    Vector4 toViewer = Vector4.Normalize(-_viewDirection);
    Vector3 specular = (Vector4.Dot(toLight, normal) > 1e-6 ? 1 : 0) *
                       _pointLight.Intensity * material.Ks *
                       (float)Math.Pow(Math.Max(Vector4.Dot(reflect, toViewer),
                                                           0), material.P);

    float attenuation = 1 / (1 + _pointLight.Attenuation *
                                        (light - point).LengthSquared());
```

```
    return material.Color * (ambient + (diffuse + specular) * attenuation);
}
```

Отрисовка

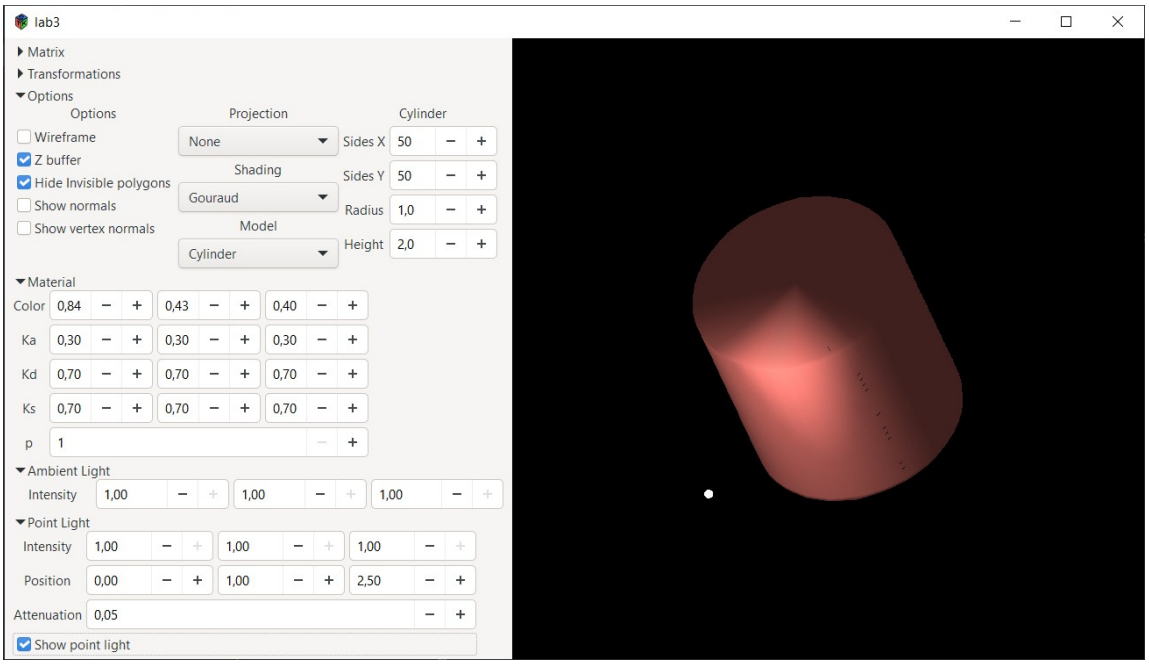
```
private void CanvasDrawnHandler(object o, DrawnArgs args)
{
    var cr = args.Cr;
    cr.Antialias = Antialias.None;
    cr.LineJoin = LineJoin.Bevel;

    cr.SetSourceColor(BACKGROUND_COLOR);
    cr.Paint();

    if (_shading.Active == (int) Shading.Gouraud)
        _surface.BeginUpdate(cr);

    foreach (var polygon in _object.Polygons)
    {
        if (Vector4.Dot(polygon.NormalInWorld, _viewDirection) > 0
            && _hideInvisible.Active)
            continue;

        Vector4 center = polygon.Vertices[0].PointInWorld;
        cr.MoveTo(TransformToView(polygon.Vertices[0].PointInWorld));
        for (int i = 1; i < polygon.Vertices.Length; ++i)
        {
            center += polygon.Vertices[i].PointInWorld;
            cr.LineTo(TransformToView(polygon.Vertices[i].PointInWorld));
        }
        center /= polygon.Vertices.Length;
        cr.ClosePath();
        cr.SetSourceColor(LINE_COLOR);

        if (_fillPolygons)
        {
            if (_shading.Active == (int) Shading.None)
            {
                cr.SetSourceRGB(polygon.Material.Color.X,
                                polygon.Material.Color.Y,
                                polygon.Material.Color.Z);
                cr.Fill();
            }
            else if (_shading.Active == (int) Shading.Flat)
            {
                var color = GetPointColor(center, polygon.NormalInWorld,
                                          polygon.Material);
                cr.SetSourceRGB(color.X, color.Y, color.Z);
                cr.Fill();
            }
            else if (_shading.Active == (int) Shading.Gouraud)
            {
                cr.NewPath();
                List<Vector3> colors = new();
                foreach (Vertex vertex in polygon.Vertices)
                {
                    colors.Add(GetPointColor(vertex.PointInWorld,
                                             vertex.NormalInWorld,
                                             polygon.Material));
                }
```

```
            var point1 = TransformToView(polygon.Vertices[0].PointInWorld);
            for (int i = 1; i < polygon.Vertices.Length - 1; ++i)
            {
                var point2 =
                        TransformToView(polygon.Vertices[i].PointInWorld);
                var point3 =
                        TransformToView(polygon.Vertices[i+1].PointInWorld);
                _surface.DrawTriangle(colors[0], point1, colors[i], point2,
                                      colors[i+1], point3);
            }
        }
    }
}
if (_shading.Active == (int) Shading.Gouraud)
    _surface.EndUpdate();

foreach (var polygon in _object.Polygons)
{
    if (Vector4.Dot(polygon.NormalInWorld, _viewDirection) > 0
        && _hideInvisible.Active)
        continue;

    Vector4 center = polygon.Vertices[0].PointInWorld;
    cr.MoveTo(TransformToView(polygon.Vertices[0].PointInWorld));
    for (int i = 1; i < polygon.Vertices.Length; ++i)
    {
        center += polygon.Vertices[i].PointInWorld;
        cr.LineTo(TransformToView(polygon.Vertices[i].PointInWorld));
    }
    center /= polygon.Vertices.Length;
    cr.ClosePath();
    cr.SetSourceColor(LINE_COLOR);

    if (_wireframe.Active)
        cr.Stroke();
    else
        cr.NewPath();

    if (_showNormals.Active)
    {
        cr.Save();
        cr.SetSourceColor(NORMAL_COLOR);

        cr.MoveTo(TransformToView(center));
        var viewNormal = TransformToView(polygon.NormalInWorld);
        if (viewNormal.Length() > NormalLength)
        {
            viewNormal = Vector2.Normalize(viewNormal) * NormalLength;
        }
        cr.RelLineTo(viewNormal);
        cr.Stroke();
        cr.Restore();
    }
}

if (_showVertexNormals.Active)
{
    cr.SetSourceColor(NORMAL_COLOR);
    foreach (Vertex vertex in _object.Vertices)
    {
        bool visible = false;
```

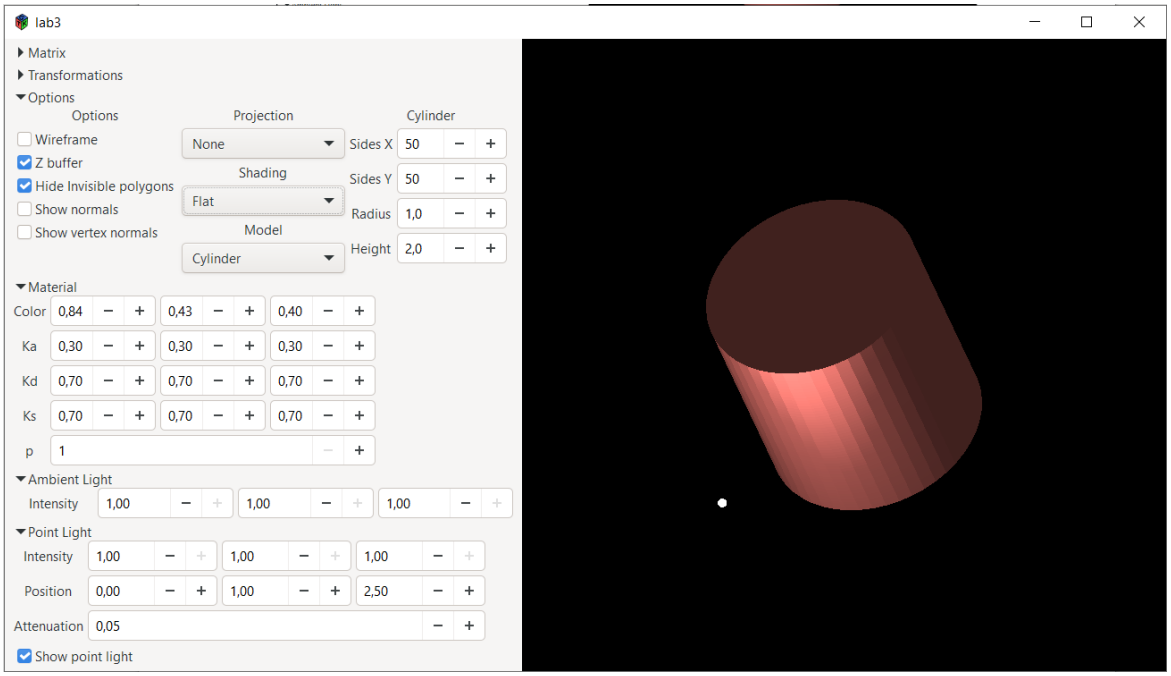
```
            foreach (Polygon polygon in vertex.Polygons)
            {
                if (Vector4.Dot(_viewDirection, polygon.NormalInWorld) <= 0)
                {
                    visible = true;
                    break;
                }
            }

            if (!visible)
                continue;

            cr.MoveTo(TransformToView(vertex.PointInWorld));
            var viewNormal = TransformToView(vertex.NormalInWorld);
            if (viewNormal.Length() > NormalLength)
            {
                viewNormal = Vector2.Normalize(viewNormal) * NormalLength;
            }
            cr.RelLineTo(viewNormal);
            cr.Stroke();
        }
    }

    if (_showPointLight.Active)
    {
        cr.SetSourceRGB(1, 1, 1);
        Vector2 point = TransformToView(Vector4.Transform(_pointLight.Point,
                                                          _worldMatrix));
        cr.Arc(point.X, point.Y, 5, 0, 2 * Math.PI);
        cr.ClosePath();
        cr.Fill();
        cr.Stroke();
    }
}
```

Демонстрация работы программы

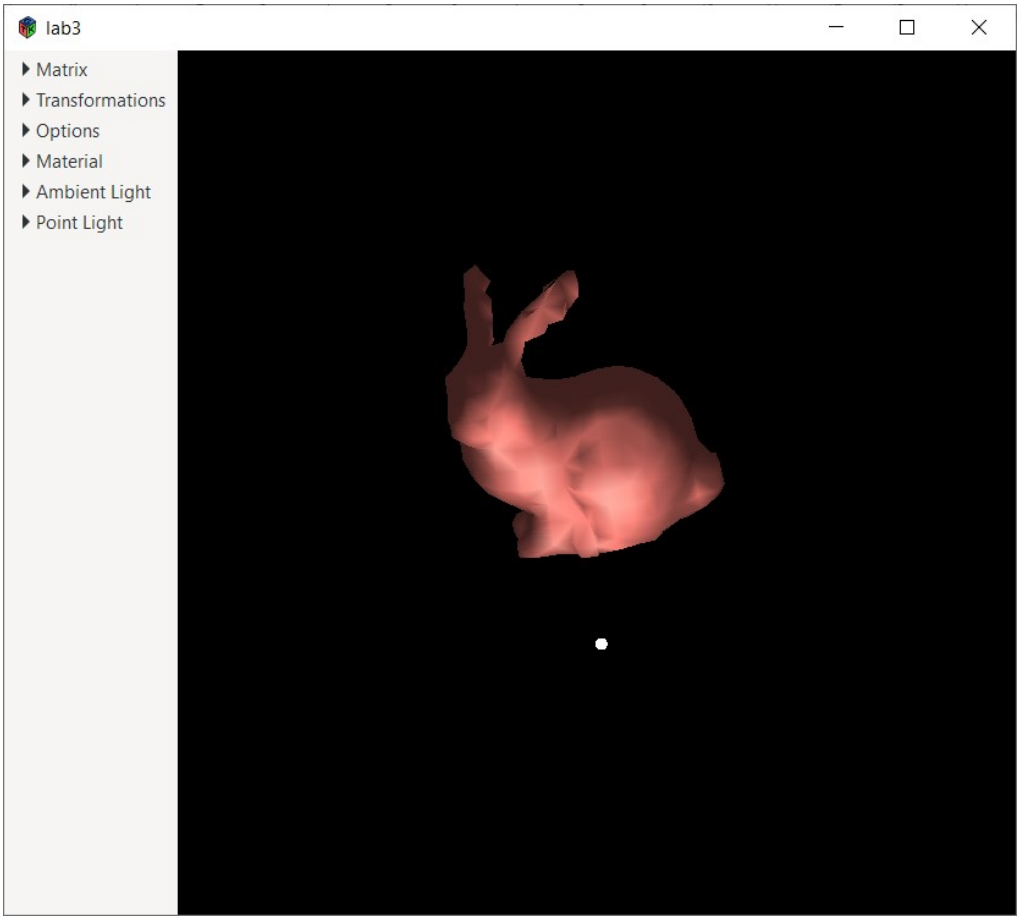
Затенение Гуро



Плоское затенение



Стенфордский кролик



Лабораторные работы №4,5,6

Постановка задачи

Тема: Ознакомление с технологией OpenGL.
Задание: Создать графическое приложение с использованием OpenGL. Используя результаты Л.Р.№3, изобразить заданное тело (то же, что и в л.р. №3) с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.
Тема: Создание шейдерных анимационных эффектов в OpenGL 2.1
Задание: Для поверхности, созданной в л.р. №5, обеспечить выполнение шейдерного эффекта.
Вариант: Анимация. Цветовые координаты изменяются по синусоидальному закону

Описание

Множество вершин и полигонов перенесено в класс меша. Для удобства работы с шейдерами реализован класс шейдера, который компилирует шейдерную программу по названиям файлов отдельных шейдеров.
В качестве моделей затенения реализована модель Фонга, в которой вычисление цвета каждой точки происходит в фрагментном шейдере, модель Гуро, в которой цвет вычисляется для каждой вершины в вершинном шейдере и модель Блинна-Фонга, для которой в фрагментном шейдере Фонга написано отдельное условие.
Реализована визуализация нормалей вершин при помощи геометрического шейдера, который для каждой вершины генерирует примитив линии.
Для просмотра объекта используется перспективная камера. Реализовано управление камерой путем перемещения ее по поверхности сферы с заданным центром, в который постоянно направлен ее вектор взгляда. Также имеется возможность регулировать угол крена камеры.

Анимационный эффект реализован в вершинном шейдере. Функция синуса поднята на 1 и сжата вдвое вдоль оси Y, таким образом её множеством значений становится [0,1]. К аргументу синуса добавляется начальная фаза, такая, чтобы при времени равном 0, значение синуса было равно исходному значению цвета. Итоговая формула:

$$C = \sin\left(\arcsin\left(2 \cdot C_0 - 1\right) + t\right) \cdot 0.5 + 0.5$$

Исходный код

Обычный вершинный шейдер

```
#version 330 core

layout (location = 0) in vec3 position;
layout (location = 1) in vec3 inColor;
layout (location = 2) in vec3 inNormal;

out vec3 normal;
out vec3 fragCoord;
out vec3 color;

uniform mat4 proj;
uniform mat4 view;
uniform mat4 model;

uniform bool animate;
uniform uint curTime;

void main()
{
    mat4 viewmodel = view * model;
    normal = normalize(vec3(viewmodel * vec4(inNormal, 0.0)));
    fragCoord = vec3(viewmodel * vec4(position, 1.0));
    if (animate) {
        color = sin(asin(2.0 * inColor - 1.0) + float(curTime) / 3000000.0)
                * 0.5 + 0.5;
    } else {
        color = inColor;
    }
    gl_Position = proj * viewmodel * vec4(position, 1.0);
}
```

Фрагментный шейдер затенения Фонга

```
#version 330 core

struct Material {
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float p;
};

struct Light {
    vec3 intensity;
    vec3 pos;
    float attenuation;
};

in vec3 normal;
in vec3 fragCoord;
in vec3 color;
out vec4 fragColor;

uniform Material material;
uniform Light light;
uniform vec3 ambientIntensity;
uniform bool blinn;
```

```
void main() {
    vec3 toLight = normalize(light.pos - fragCoord);

    vec3 ambient = ambientIntensity * material.Ka;

    float diffuseCoef = max(dot(toLight, normal), 0);
    vec3 diffuse = light.intensity * material.Kd * diffuseCoef;

    vec3 toCamera = normalize(-fragCoord);
    float specularCoef = 0;
    if (diffuseCoef > 0) {
        if (blinn) {
            vec3 halfwayDir = normalize(toLight + toCamera);
            specularCoef = pow(max(dot(halfwayDir, normal), 0), material.p);
        } else {
            vec3 reflectDir = reflect(-toLight, normal);
            specularCoef = pow(max(dot(reflectDir, toCamera), 0), material.p);
        }
    }
    vec3 specular = light.intensity * material.Ks * specularCoef;

    float distToLight = length(light.pos - fragCoord);
    float attenuation = (1 + light.attenuation * distToLight * distToLight);

    fragColor = vec4(color * (ambient + (diffuse + specular) / attenuation), 1);
}
```

Геометрический шейдер генерации нормалей

```
#version 330 core

layout (triangles) in;
layout (line_strip, max_vertices = 6) out;

in vec3 normal[];
const float MAGNITUDE = 0.4;

void GenerateLine(int index)
{
    gl_Position = gl_in[index].gl_Position;
    EmitVertex();
    gl_Position = gl_in[index].gl_Position + vec4(normal[index], 0) * MAGNITUDE;
    EmitVertex();
    EndPrimitive();
}

void main()
{
    GenerateLine(0);
    GenerateLine(1);
    GenerateLine(2);
}
```

Отрисовка

```
glArea.Render += (_, _) =>
{
 gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT);

 var projMatrix = _camera.GetProjectionMatrix();
```

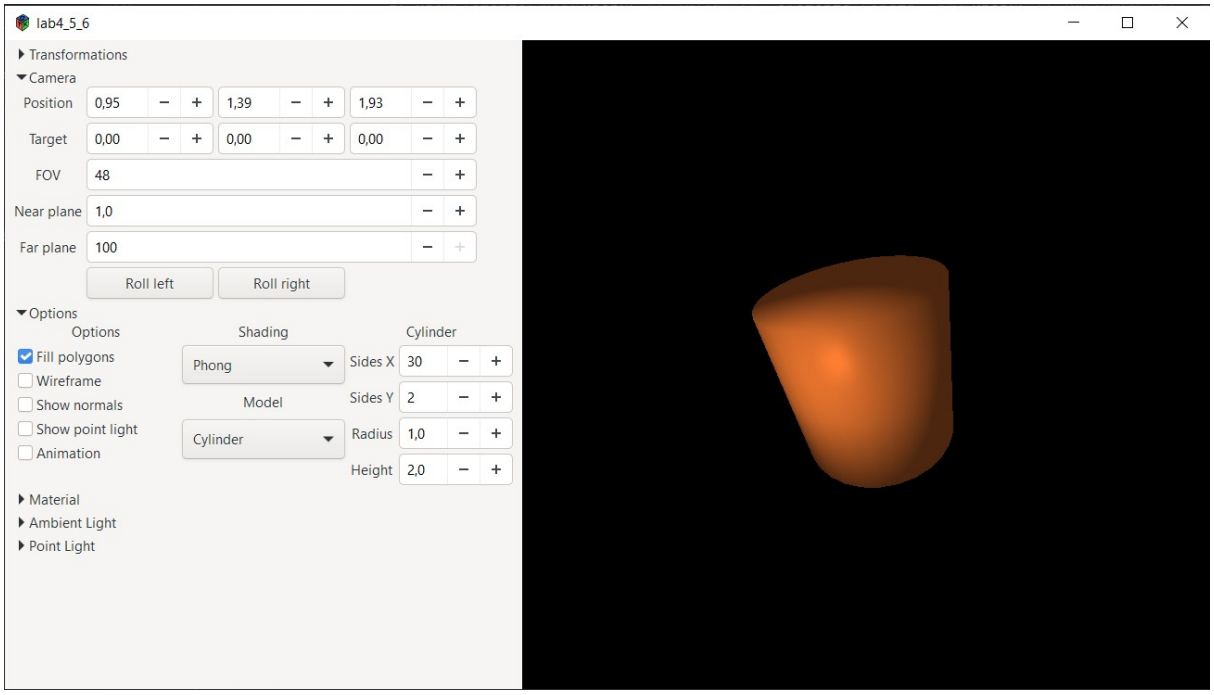
```
var viewMatrix = _camera.GetViewMatrix();
// autoscale
viewMatrix = Matrix4x4.CreateScale(Math.Min(_camera.AspectRatio, 1),
                          Math.Min(_camera.AspectRatio, 1), 1) * viewMatrix;
var modelMatrix = _object.GetModelMatrix();

gl.UseProgram(baseShader.Id);
baseShader.SetMatrix4(gl, "model", modelMatrix);
baseShader.SetMatrix4(gl, "view", viewMatrix);
baseShader.SetMatrix4(gl, "proj", projMatrix);
baseShader.SetInt(gl, "useSingleColor", 0);
baseShader.SetInt(gl, "animate", 0);
gl.UseProgram(phongShader.Id);
phongShader.SetInt(gl, "animate", 0);

if (!_animation.Active)
{
    _startTime = (uint) frame_clock.FrameTime;
}

if (_modelChanged)
{
    _modelChanged = false;
    gl.BindVertexArray(vao);
    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vbo);
    _object.ToArray(true, true, true, out vertices, out elements);
    gl.BufferData(OpenGL.GL_ARRAY_BUFFER, vertices, OpenGL.GL_DYNAMIC_DRAW);
    gl.BufferData(OpenGL.GL_ELEMENT_ARRAY_BUFFER, elements,
                  OpenGL.GL_DYNAMIC_DRAW);
    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
    gl.BindVertexArray(0);
}

if (_lightPosChanged)
{
    _lightPosChanged = false;
    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, lightVbo);
    gl.BufferData(OpenGL.GL_ARRAY_BUFFER, _pointLight.Point.ToArray(),
                  OpenGL.GL_DYNAMIC_DRAW);
    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
}

gl.BindVertexArray(vao);

if (_fillPolygons.Active)
{
    if (_shading.Active == (int) Shading.None)
    {
        gl.UseProgram(baseShader.Id);
        if (_animation.Active)
        {
            baseShader.SetInt(gl, "animate", 1);
            baseShader.SetUint(gl, "curTime",
                               (uint)frame_clock.FrameTime - _startTime);
        }
    }
    else if (_shading.Active == (int) Shading.Gouraud)
    {
        gl.UseProgram(gouraudShader.Id);
        if (_animation.Active)
        {
```

```
            gouraudShader.SetInt(gl, "animate", 1);
            gouraudShader.SetUint(gl, "curTime",
                                  (uint)frame_clock.FrameTime - _startTime);
        }
        gouraudShader.SetInt(gl, "useSingleColor", 0);
        gouraudShader.SetMatrix4(gl, "model", modelMatrix);
        gouraudShader.SetMatrix4(gl, "view", viewMatrix);
        gouraudShader.SetMatrix4(gl, "proj", projMatrix);
        gouraudShader.SetVec3(gl, "material.Ka", _material.Ka);
        gouraudShader.SetVec3(gl, "material.Kd", _material.Kd);
        gouraudShader.SetVec3(gl, "material.Ks", _material.Ks);
        gouraudShader.SetFloat(gl, "material.p", _material.P);
        gouraudShader.SetVec3(gl, "ambientIntensity",
                              _ambientLight.Intensity);
        gouraudShader.SetVec3(gl, "light.intensity", _pointLight.Intensity);
        var lightPos = Vector3.Transform(_pointLight.Point,
                                         Matrix4x4.Transpose(viewMatrix));
        gouraudShader.SetVec3(gl, "light.pos", lightPos);
        gouraudShader.SetFloat(gl, "light.attenuation",
                               _pointLight.Attenuation);
    }
    else if (_shading.Active == (int) Shading.Phong
             || _shading.Active == (int) Shading.BlinnPhong)
    {
        gl.UseProgram(phongShader.Id);
        if (_animation.Active)
        {
            phongShader.SetInt(gl, "animate", 1);
            phongShader.SetUint(gl, "curTime",
                                (uint)frame_clock.FrameTime - _startTime);
        }
        phongShader.SetMatrix4(gl, "model", modelMatrix);
        phongShader.SetMatrix4(gl, "view", viewMatrix);
        phongShader.SetMatrix4(gl, "proj", projMatrix);
        phongShader.SetVec3(gl, "material.Ka", _material.Ka);
        phongShader.SetVec3(gl, "material.Kd", _material.Kd);
        phongShader.SetVec3(gl, "material.Ks", _material.Ks);
        phongShader.SetFloat(gl, "material.p", _material.P);
        phongShader.SetVec3(gl, "ambientIntensity",
                            _ambientLight.Intensity);
        phongShader.SetVec3(gl, "light.intensity", _pointLight.Intensity);
        var lightPos = Vector3.Transform(_pointLight.Point,
                                         Matrix4x4.Transpose(viewMatrix));
        phongShader.SetVec3(gl, "light.pos", lightPos);
        phongShader.SetFloat(gl, "light.attenuation",
                             _pointLight.Attenuation);
        if (_shading.Active == (int) Shading.BlinnPhong)
            phongShader.SetInt(gl, "blinn", 1);
        else
            phongShader.SetInt(gl, "blinn", 0);
    }
    gl.PolygonMode(OpenGL.GL_FRONT_AND_BACK, OpenGL.GL_FILL);
    gl.DrawElements(OpenGL.GL_TRIANGLES, elements.Length,
                    OpenGL.GL_UNSIGNED_INT, IntPtr.Zero);
}

if (_wireframe.Active)
{
    gl.UseProgram(baseShader.Id);
    baseShader.SetInt(gl, "useSingleColor", 1);
    baseShader.SetVec3(gl, "singleColor", LINE_COLOR);
```

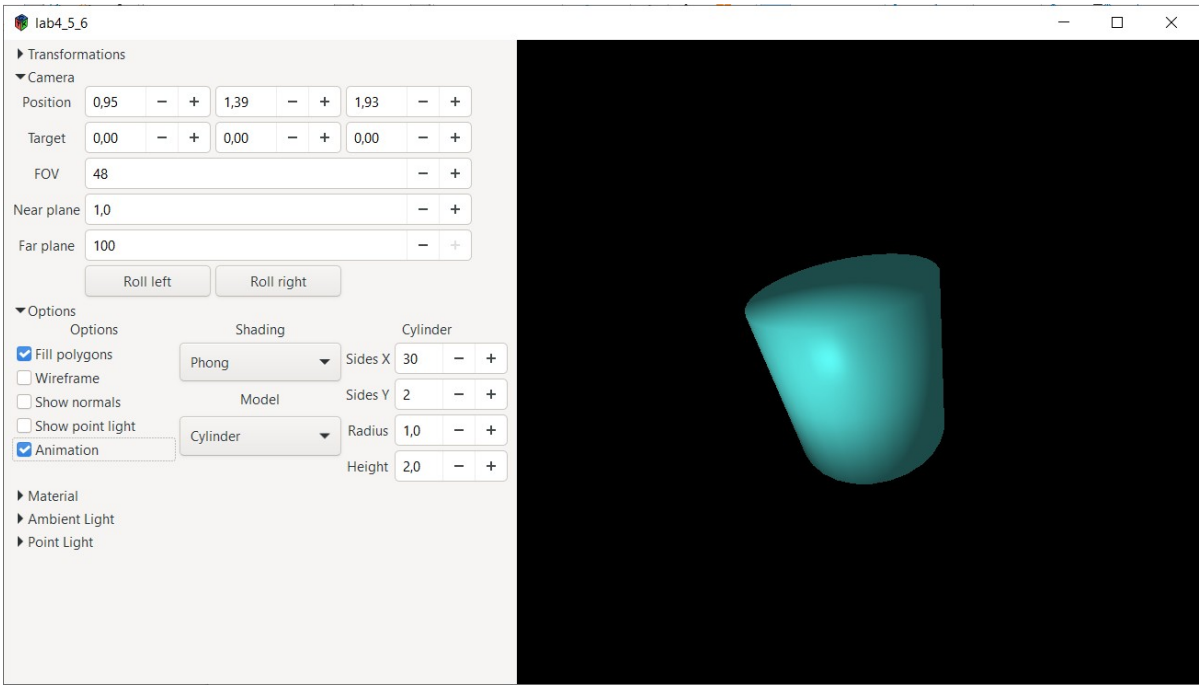
```
            baseShader.SetInt(gl, "animate", 0);
            gl.LineWidth(2);
            gl.PolygonMode(OpenGL.GL_FRONT_AND_BACK, OpenGL.GL_LINE);
            gl.DrawElements(OpenGL.GL_TRIANGLES, elements.Length,
                            OpenGL.GL_UNSIGNED_INT, IntPtr.Zero);
        }

        if (_showNormals.Active)
        {
            gl.UseProgram(normalsShader.Id);
            normalsShader.SetMatrix4(gl, "model", modelMatrix);
            normalsShader.SetMatrix4(gl, "view", viewMatrix);
            normalsShader.SetMatrix4(gl, "proj", projMatrix);
            normalsShader.SetInt(gl, "useSingleColor", 1);
            normalsShader.SetVec3(gl, "singleColor", NORMAL_COLOR);
            gl.LineWidth(2);
            gl.DrawElements(OpenGL.GL_TRIANGLES, elements.Length,
                            OpenGL.GL_UNSIGNED_INT, IntPtr.Zero);
        }

        if (_showPointLight.Active)
        {
            gl.BindVertexArray(lightVao);
            gl.UseProgram(baseShader.Id);
            baseShader.SetMatrix4(gl, "model", Matrix4x4.Identity);
            baseShader.SetInt(gl, "useSingleColor", 1);
            baseShader.SetVec3(gl, "singleColor", _pointLight.Intensity);
            baseShader.SetInt(gl, "animate", 0);
            gl.PointSize(10);
            gl.DrawArrays(OpenGL.GL_POINTS, 0, 1);
        }

        gl.BindVertexArray(0);
        gl.UseProgram(0);
    };
```

Демонстрация работы программы

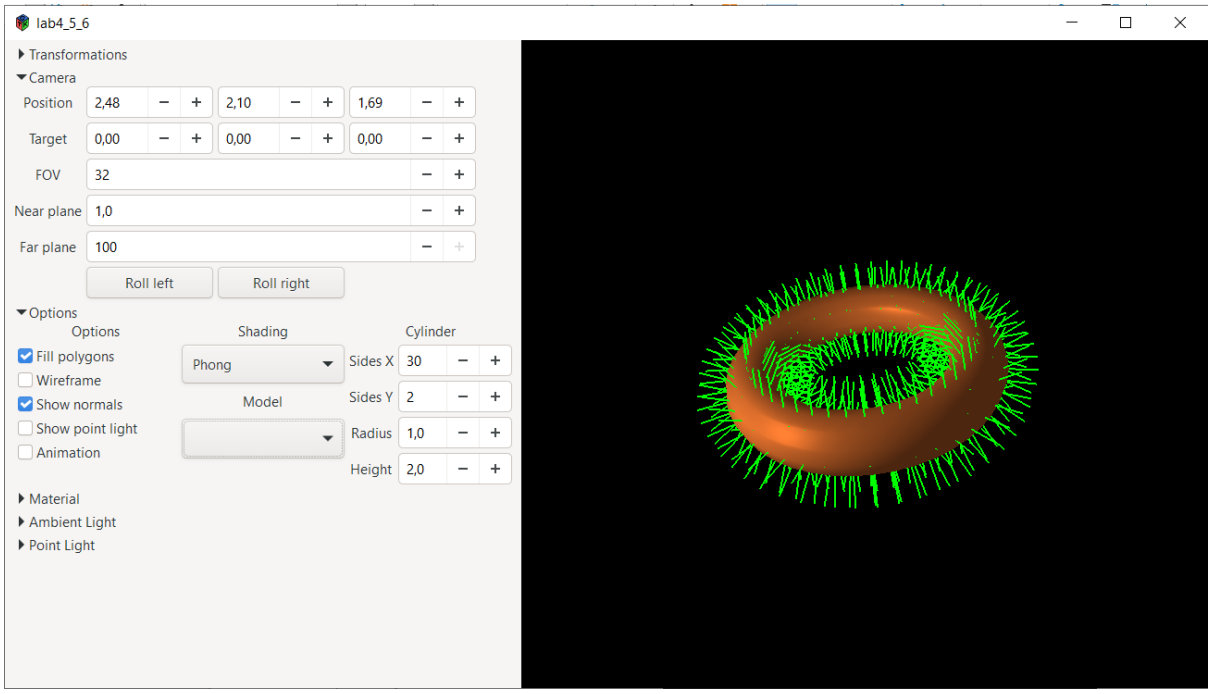
Затенение Фонга



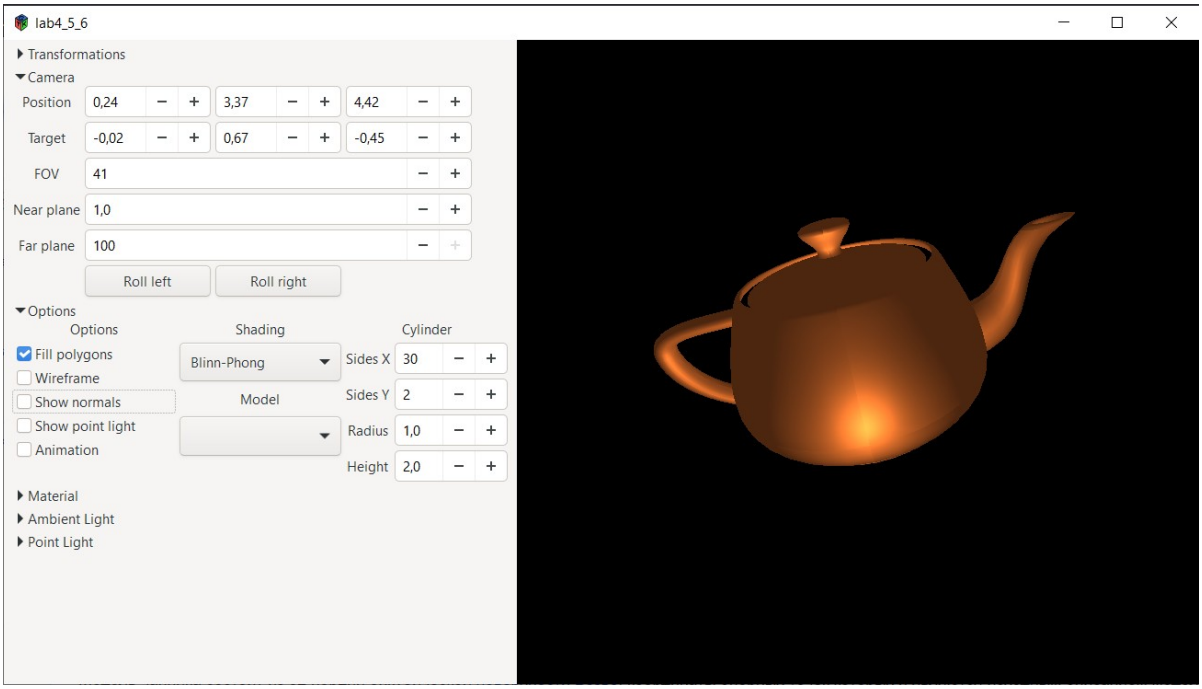
Анимация



Визуализация нормалей



Чайник из Юты, затенение Блинна-Фонга



Лабораторная работа №7

Постановка задачи

Тема: Построение плоских полиномиальных кривых.
Задание: Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек и, при необходимости, значений касательных векторов и натяжения.
Вариант: Сегмент кубического сплайна по конечным точкам и касательным

Описание

Кубический сплайн, задаваемый значениями и производными на граничных точках, представляет из себя ни что иное, как сплайн Эрмита. Элементарный сплайн Эрмита задаётся следующим уравнением.

$$P_1(2t^3-3t^2+1)+P_2(-2t^3+3t^2)+Q_1(t^3-2t^2+t)+Q_2(t^3-t^2) \quad \text{, где} \quad 0 \le t \le 1$$

P_1, P_2 – конечное точки, Q_1, Q_2 – производные в этих точках.
Саму кривую можно построить по этому уравнению, используя метод из первой лабораторной работы.
В программе реализовано построение составного сплайна Эрмита из множества элементарных, при этом производные в промежуточных точках могут контролироваться пользователем.
Реализовано изменение положения опорных точек и величины производных путем перемещения соответствующих элементов на экране, а также добавление и удаление точек.

Исходный код

Класс кривой

```
public class CubicSpline
{
    public List<Vector2> Points;
    public List<Vector2> Derivatives;
    public float TangentFactor;

    public CubicSpline(Vector2 a, Vector2 b, Vector2 da, Vector2 db,
                       float tangentFactor)
    {
        Points = new List<Vector2> {a, b};
        Derivatives = new List<Vector2> {da, db};
        TangentFactor = tangentFactor;
    }

    // Вычисление значения заданного элемента сплайна в заданной точке
    public Vector2 GetValue(int i, float t)
    {
        Vector2 a = Points[i + 1] - Points[i] - TangentFactor * Derivatives[i];
        Vector2 b = 2 * (Points[i] - Points[i + 1]) + TangentFactor *
                    Derivatives[i] + TangentFactor * Derivatives[i + 1];
        return Points[i] + t * (TangentFactor * Derivatives[i] + t *
```

```
                                                    (a + b * (t - 1)));
    }

    // Поиск точки (производной) по координате
    public int FindPoint(Vector2 point, float epsilon, bool points,
                         bool derivatives)
    {
        if (derivatives)
        {
            for (int i = 0; i < Derivatives.Count; ++i)
            {
                if ((Points[i] + Derivatives[i] - point).Length() < epsilon)
                {
                    return i + Points.Count;
                }

                if ((Points[i] - Derivatives[i] - point).Length() < epsilon)
                {
                    return i + Points.Count + Points.Count;
                }
            }
        }

        if (points)
        {
            for (int i = 0; i < Points.Count; ++i)
            {
                if ((Points[i] - point).Length() < epsilon)
                {
                    return i;
                }
            }
        }

        return -1;
    }

    public void AddPoint(Vector2 point)
    {
        Vector2 derivative = Vector2.Normalize(point - Points.Last()) / 4;
        Derivatives.Add(derivative);
        Points.Add(point);
    }

    public void RemovePoint(int index)
    {
        if (Points.Count == 2 || index >= Points.Count)
        {
            return;
        }

        Points.RemoveAt(index);
        Derivatives.RemoveAt(index);
    }
}
```

Отрисовка кривой

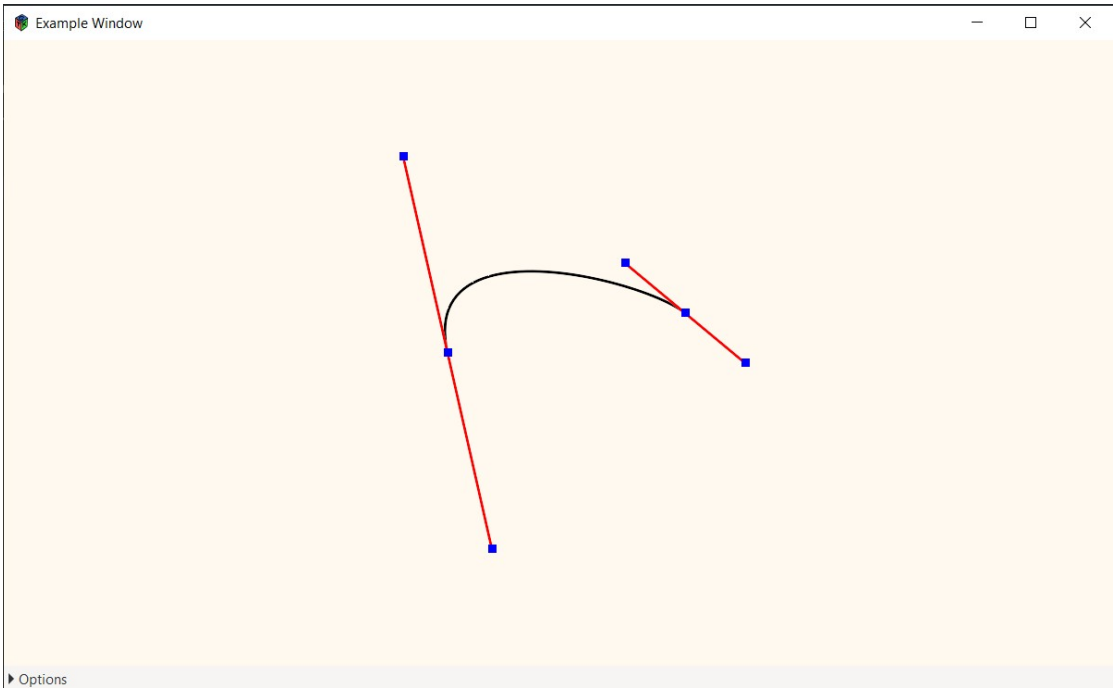
```
glArea.Render += (_, _) =>
{
    gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT);
```

```
if (_modelChanged)
{
    _modelChanged = false;
    List<float> verts = new();
    float dt = 1f / (float)_approximation.Value;
    for (int i = 0; i < _spline.Points.Count - 1; ++i)
    {
        for (float t = 0; t < 1; t += dt)
        {
            Vector2 value = _spline.GetValue(i, t);
            verts.Add(value.X);
            verts.Add(value.Y);
        }
    }
    verts.Add(_spline.Points.Last().X);
    verts.Add(_spline.Points.Last().Y);
    vertices = verts.ToArray();

    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, splineVbo);
    gl.BufferData(OpenGL.GL_ARRAY_BUFFER, vertices, OpenGL.GL_DYNAMIC_DRAW);

    float[] points = new float[_spline.Points.Count * 6];
    for (int i = 0; i < _spline.Points.Count; ++i)
    {
        points[i * 2] = _spline.Points[i].X;
        points[i * 2 + 1] = _spline.Points[i].Y;
    }

    for (int i = 0; i < _spline.Points.Count; ++i)
    {
        points[_spline.Points.Count*2 + i * 4] = _spline.Points[i].X +
                                                 _spline.Derivatives[i].X;
        points[_spline.Points.Count*2 + i * 4 + 1] = _spline.Points[i].Y +
                                                 _spline.Derivatives[i].Y;
        points[_spline.Points.Count*2 + i * 4 + 2] = _spline.Points[i].X -
                                                 _spline.Derivatives[i].X;
        points[_spline.Points.Count*2 + i * 4 + 3] = _spline.Points[i].Y -
                                                 _spline.Derivatives[i].Y;
    }
    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, pointsVbo);
    gl.BufferData(OpenGL.GL_ARRAY_BUFFER, points, OpenGL.GL_DYNAMIC_DRAW);
    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
}

gl.UseProgram(baseShader.Id);

gl.BindVertexArray(splineVao);
baseShader.SetVec3(gl, "color", LineColor);
gl.DrawArrays(OpenGL.GL_LINE_STRIP, 0, vertices.Length / 2);

if (_drawTangents.Active)
{
    gl.BindVertexArray(pointsVao);
    baseShader.SetVec3(gl, "color", TangentColor);
    gl.DrawArrays(OpenGL.GL_LINES, _spline.Points.Count,
                  _spline.Points.Count * 2);
    baseShader.SetVec3(gl, "color", PointColor);
    gl.DrawArrays(OpenGL.GL_POINTS, _spline.Points.Count,
                  _spline.Points.Count * 2);
}
```

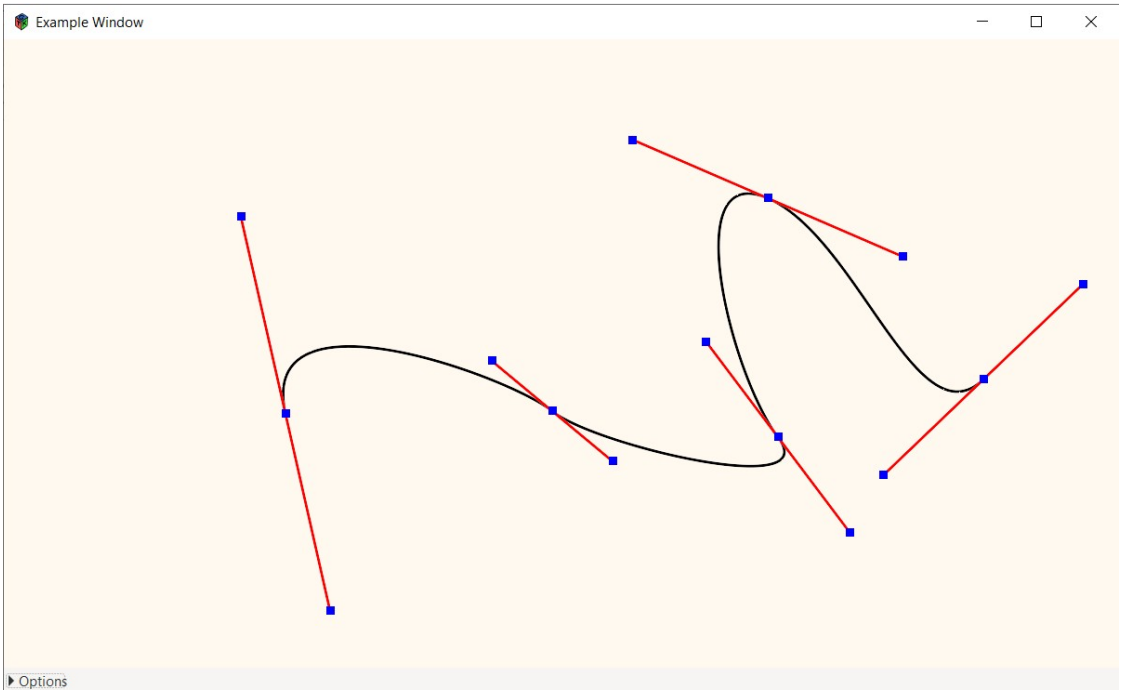
```
    if (_drawPoints.Active)
    {
        gl.BindVertexArray(pointsVao);
        baseShader.SetVec3(gl, "color", PointColor);
        gl.DrawArrays(OpenGL.GL_POINTS, 0, _spline.Points.Count);
    }

    gl.BindVertexArray(0);
    gl.UseProgram(0);
};
```

Демонстрация работы программы

Сегмент сплайна



Составной сплайн



Используемая литература

1. *Mono Documentation* – URL: http://docs.go-mono.com
2. *GTK Documentation* – URL: https://docs.gtk.org
3. *Learnopengl* – URL: https://learnopengl.com
4. *OpenGL Wiki* – URL: https://www.khronos.org/opengl/wiki
5. Шикин Е. В., Плис Л. И. *Кривые и поверхности на экране компьютера. Руководство по сплайнам для пользователей.* – М.: ДИАЛОГ-МИФИ, 1996. - 240с.