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»

Студент: К. М. Воронов
Преподаватель: А. В. Морозов
Группа: М8О-307Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №1

Построение изображений 2D - кривых

Задача: Написать и отладить программу, строящую изображение заданной замечательной кривой.

Вариант задания: $x^{2/3} + y^{2/3} = a^{2/3}$

Описание

Для удобства переведём координаты в полярные координаты: $x = acos^3\phi$, $y = asin^3\phi$. В итоге параметризация происходит по ϕ: 2π делится на заданное пользователем число и рисовка происходит с соответствующим шагом. Также здесь реализоованы сдвиг, поворот графика, автомасштаб/масштаб и рисовка осей. Для корректного отображения к соответствующим координатам прибавлены половины от высоты/ширины окна.

Исходный код

Основные фрагменты кода.

Генерация точек

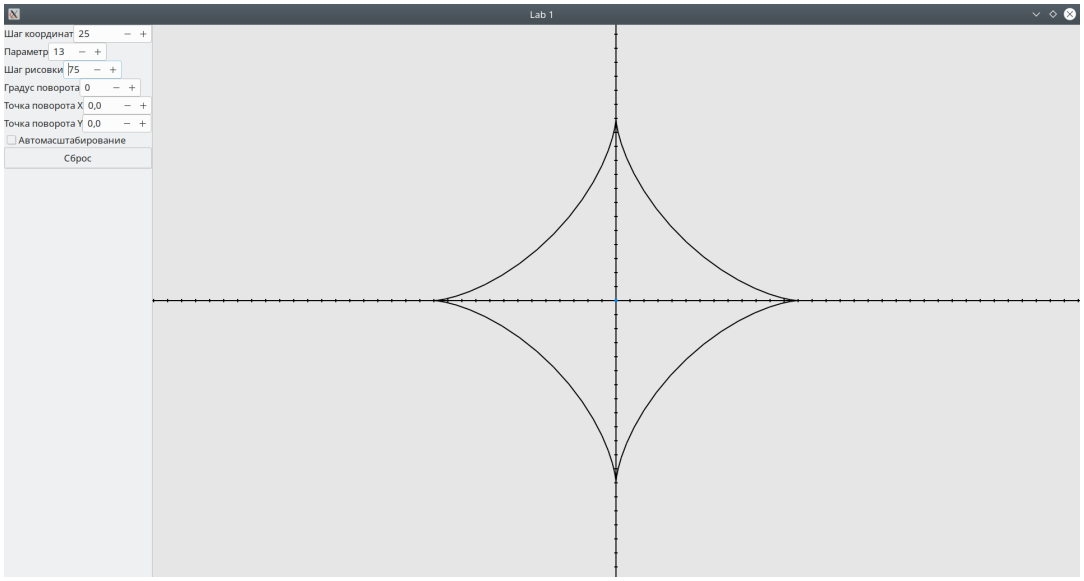
```
private void Redro(object? sender, EventArgs e)
{
    points.Clear();
    paramPhi = (int)_phi.Value;
    double k = 2 * Math.PI / paramPhi;
    for (double i = 0; i <= 2 * Math.PI; i += k)
    {
        Point point = new Point();
        point.x = paramA * Math.Cos(i) * Math.Cos(i) * Math.Cos(i) * Globalgo;
        point.y = paramA * Math.Sin(i) * Math.Sin(i) * Math.Sin(i) * Globalgo;
        points.Add(point);
    }
    points.Add(points[0]);
    _drawingArea.QueueDraw();
}
```

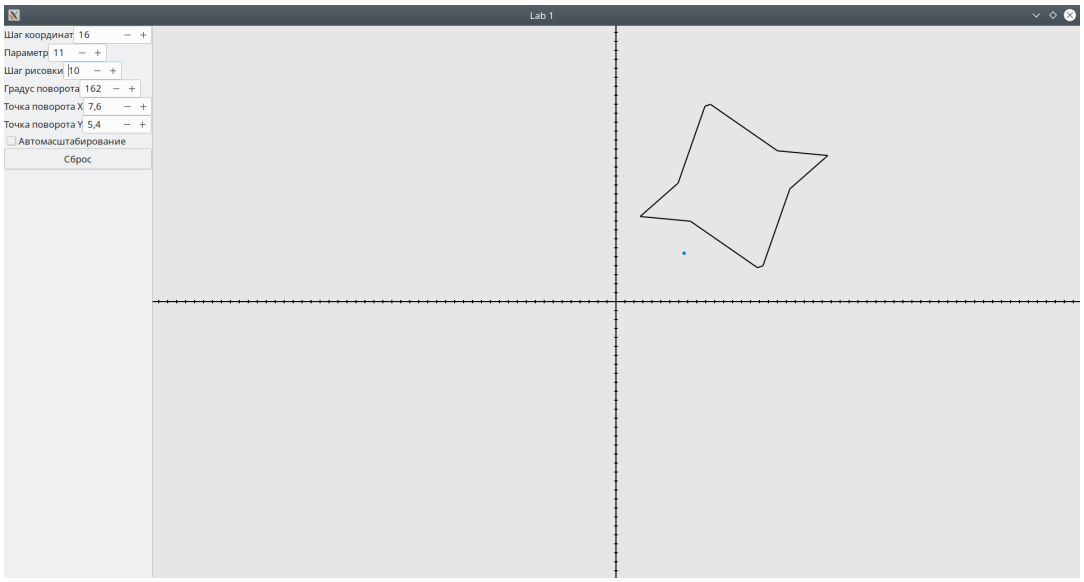
Рисовка осей и графика

```

public void DrawCoordinates(Context context, int point1, int point2, double go)
{
    context.MoveTo(0, point2 / 2 + dy);
    context.LineTo(point1, point2 / 2 + dy);
    context.Stroke();

    context.MoveTo(point1 / 2 + dx, 0);
    context.LineTo(point1 / 2 + dx, point2);
    context.Stroke();

    for (double i = point1 / 2 + dx; i < point1; i += go)
    {
        context.MoveTo(i, point2 / 2 + dy - 3);
        context.LineTo(i, point2 / 2 + dy + 3);
        context.Stroke();
    }

    for (double i = point1 / 2 + dx; i > 0; i -= go)
    {
        context.MoveTo(i, point2 / 2 + dy - 3);
        context.LineTo(i, point2 / 2 + dy + 3);
        context.Stroke();
    }

    for (double i = point2 / 2 + dy; i < point2; i += go)
    {
```

```
43          context.MoveTo(point1 / 2 + dx - 3, i);
44          context.LineTo(point1 / 2 + dx + 3, i);
45          context.Stroke();
46      }
47
48      for (double i = point2 / 2 + dy; i > 0; i -= go)
49      {
50          context.MoveTo(point1 / 2 + dx - 3, i);
51          context.LineTo(point1 / 2 + dx + 3, i);
52          context.Stroke();
53      }
54
55      RotateGraph();
56
57      if (_auto.Active)
58      {
59          _adj.Value *= (0.5 * Math.Min(point1 / (2 * maxFuncX), point2 / (2 * maxFuncY))
                ;
60          _drawingArea.QueueDraw();
61      }
62
63      for (int i = 0; i < RotationPoints.Count - 1; ++i)
64      {
65          context.MoveTo(RotationPoints[i].x + point1 / 2 + dx, RotationPoints[i].y +
                point2 / 2 + dy);
66          context.LineTo(RotationPoints[i + 1].x + point1 / 2 + dx, RotationPoints[i +
                1].y + point2 / 2 + dy);
67          context.Stroke();
68      }
69  }
```

Скриншоты работы программы





Лабораторная работа №2

Каркасная визуализация выпуклого многогранника. Удаление невидимых линий

Задача: 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.

Вариант задания: Обелиск (усеченный клин).

Описание

Для хранения фигуры используется два класса: класс вершин и класс полигонов. В каждом полигоне хранятся все вершины, из которых он состоит, а в каждой вершине - все полигоны, с которыми она соприкасается. Для каждого полигона вычисляется нормаль, с помощью которой определяется, нужно ли рисовать данный полигон (если нормаль смотрит в противоположную сторону от экрана - полигон пропускается). Для поворота, сдвига и масштабирования координаты умножаются на соответствующие матрицы. Есть возможность раскрасить фигуру.

Исходный код

Основные фрагменты кода.

Подсчёт матриц

```
70  void CalculateMatrix()
71  {
72      WorldMatrix = Matrix4x4.Identity;
73
74      WorldMatrix *= Matrix4x4.CreateScale((float)_scaleX.Value, (float)_scaleY.Value, (
            float)_scaleZ.Value);
75
76      if (_proj.ActiveText == "")
77      {
78          WorldMatrix *= Matrix4x4.CreateRotationY((float) (-45 * Math.PI / 180)) *
79                        Matrix4x4.CreateRotationX((float) (35 * Math.PI / 180));
80      }
81      else
82      {
83          WorldMatrix *= Matrix4x4.CreateRotationX((float) (_rotationX.Value * Math.PI /
                180) );
84          WorldMatrix *= Matrix4x4.CreateRotationY((float) (_rotationY.Value * Math.PI /
                180));
85          WorldMatrix *= Matrix4x4.CreateRotationZ((float) (_rotationZ.Value * Math.PI /
                180));
86      }
87
88      WorldMatrix *= Matrix4x4.CreateTranslation((float) _shiftX.Value, (float) _shiftY.
            Value, (float)_shiftZ.Value);
89
90      Matrix4x4 projectionMatrix = Matrix4x4.Identity;
91
92      if (_proj.ActiveText == " ")
93      {
94          projectionMatrix.M22 = 0;
95      }
96
97      if (_proj.ActiveText == " ")
98      {
99          projectionMatrix.M11 = 0;
100     }
101
102     if (_proj.ActiveText == " ")
103     {
104         projectionMatrix.M33 = 0;
105     }
106
107     WorldMatrix = projectionMatrix * WorldMatrix;
108
```

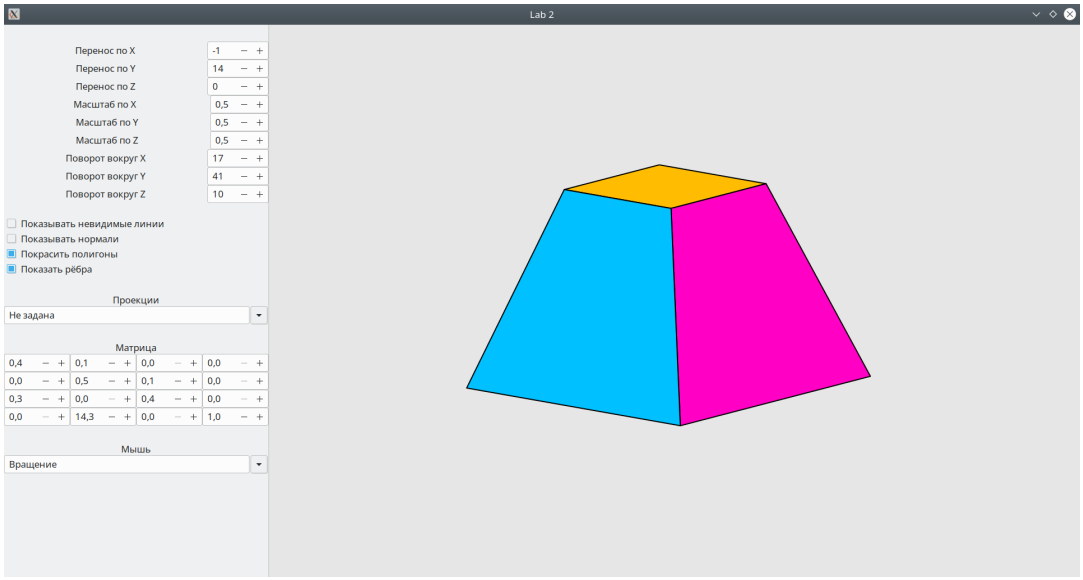
```
109     _m11.Value = WorldMatrix.M11;
110     _m12.Value = WorldMatrix.M12;
111     _m13.Value = WorldMatrix.M13;
112     _m14.Value = WorldMatrix.M14;
113     _m21.Value = WorldMatrix.M21;
114     _m22.Value = WorldMatrix.M22;
115     _m23.Value = WorldMatrix.M23;
116     _m24.Value = WorldMatrix.M24;
117     _m31.Value = WorldMatrix.M31;
118     _m32.Value = WorldMatrix.M32;
119     _m33.Value = WorldMatrix.M33;
120     _m34.Value = WorldMatrix.M34;
121     _m41.Value = WorldMatrix.M41;
122     _m42.Value = WorldMatrix.M42;
123     _m43.Value = WorldMatrix.M43;
124     _m44.Value = WorldMatrix.M44;
125 
126     WMatrix();
127     _drawingArea.QueueDraw();
128 }
```

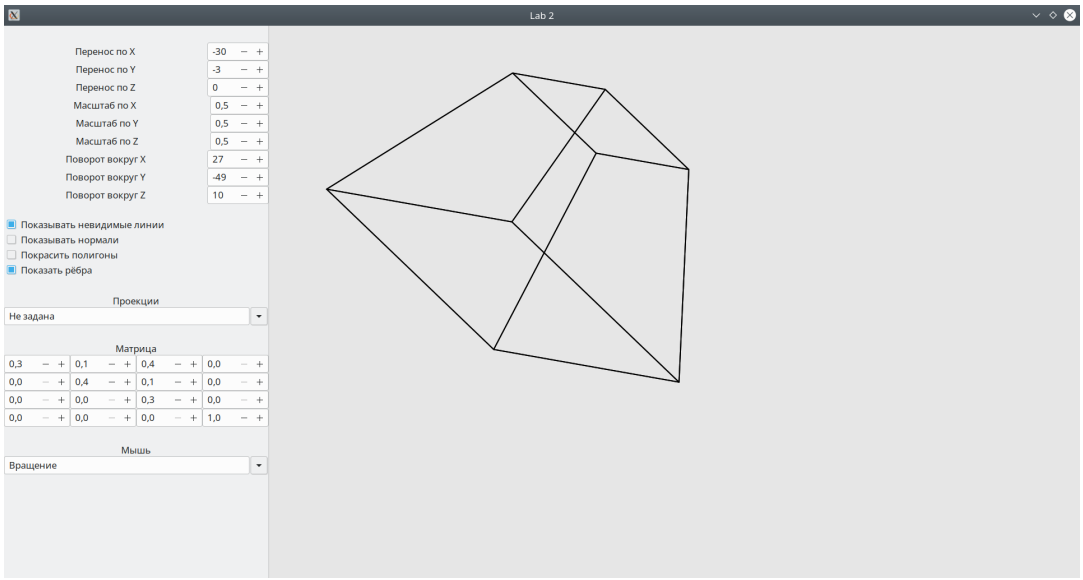
Функция, определяющая, надо ли рисовать полигон

```
129 bool Hide(Polygon a)
130 {
131     Vector3 vector1 = DifferenceVector3(a.points[0].PointInWorldSpace,a.points[1].
            PointInWorldSpace) ;
132     Vector3 vector2 = DifferenceVector3(a.points[2].PointInWorldSpace,a.points[1].
            PointInWorldSpace) ;
133 
134 
135     Vector3 vector3 = new Vector3(vector1.Y * vector2.Z - vector1.Z * vector2.Y,
136         vector1.Z * vector2.X - vector1.X * vector2.Z, vector1.X * vector2.Y - vector1.
                Y * vector2.X);
137 
138     vector3 = Normalize(vector3);
139 
140     Vector2 start = new Vector2((a.points[0].PointInWorldSpace.X + a.points[1].
            PointInWorldSpace.X + a.points[2].PointInWorldSpace.X + a.points[3].
            PointInWorldSpace.X) / 4, (a.points[0].PointInWorldSpace.Y + a.points[1].
            PointInWorldSpace.Y + a.points[2].PointInWorldSpace.Y + a.points[3].
            PointInWorldSpace.Y) / 4 );
141 
142     vector3.X = vector3.X + start.X;
143     vector3.Y = vector3.Y + start.Y;
144 
145 
146     if (vector3.Z > 0)
147     {
148         return true;
```

```
149     }
150     return false;
151 }
```

Скриншоты работы программы





Лабораторная работа №3

Основы построения фотореалистичных изображений

Задача: Используя результаты предыдущей лабораторной работы, 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 задания: Эллипсоид.

Описание

Общая модель взята из предыдущей работы. Координаты в сферические: $x = acos\phi sin\theta$, $y = bsin\theta sin\phi$, $z = ccos\theta$. Параматризация происходит по ϕ и по θ. Реализованы две модели освещения: плоское затенение и затенение Гуро. Они состоят из трёх составляющих: фоновой, рассеивающей и зеркальной. Для модели освещения Гуро считаются нормали для вершин, как суммы нормалей для полигонов.

Исходный код

Основные фрагменты кода.

Фоновая составляющая

```
152  Vector3 Background()
153 {
154     Vector3 ka = new Vector3((float)_kar.Value, (float)_kag.Value, (float)_kab.Value);
155     Vector3 ia = new Vector3((float)_iar.Value, (float)_iag.Value, (float)_iab.Value);
156     return Multiplication(ka, ia);
157 }
```

Рассеивающая составляющая для плоского затенения

```
158 Vector3 Diffuse( Polygon a)
159 {
160     Vector3 ans = new Vector3();
161     Vector3 kd = new Vector3((float)_kdr.Value, (float)_kdg.Value, (float)_kdb.Value);
162     Vector3 il = new Vector3((float)_ilr.Value, (float)_ilg.Value, (float)_ilb.Value);
163 
164     ans = Multiplication(kd, il);
165 
166      Vector3 start = new Vector3((a.points[0].PointInWorldSpace.X + a.points[1].
            PointInWorldSpace.X + a.points[2].PointInWorldSpace.X) / 3, (a.points[0].
            PointInWorldSpace.Y + a.points[1].PointInWorldSpace.Y + a.points[2].
            PointInWorldSpace.Y) / 3, (a.points[0].PointInWorldSpace.Z + a.points[1].
            PointInWorldSpace.Z + a.points[2].PointInWorldSpace.Z) / 3 );
167     Vector3 dot = new Vector3((float)_lx.Value - start.X, (float)_ly.Value - start.Y,(
            float)_lz.Value - start.Z);
168 
169     dot = Normalize(dot);
170     ans *= Vector3.Dot(dot, a.NormalInWorldSpace);
171     ans.X = Math.Max(0, ans.X);
172     ans.Y = Math.Max(0, ans.Y);
173     ans.Z = Math.Max(0, ans.Z);
174 
175     dot.X = (float)_lx.Value - a.NormalInWorldSpace.X - start.X;
176     dot.Y = (float)_ly.Value - a.NormalInWorldSpace.Y - start.Y;
177     dot.Z = (float)_lz.Value - a.NormalInWorldSpace.Z - start.Z;
178     return ans/ (float)(dot.Length() / 50 + _k.Value);
179 }
```

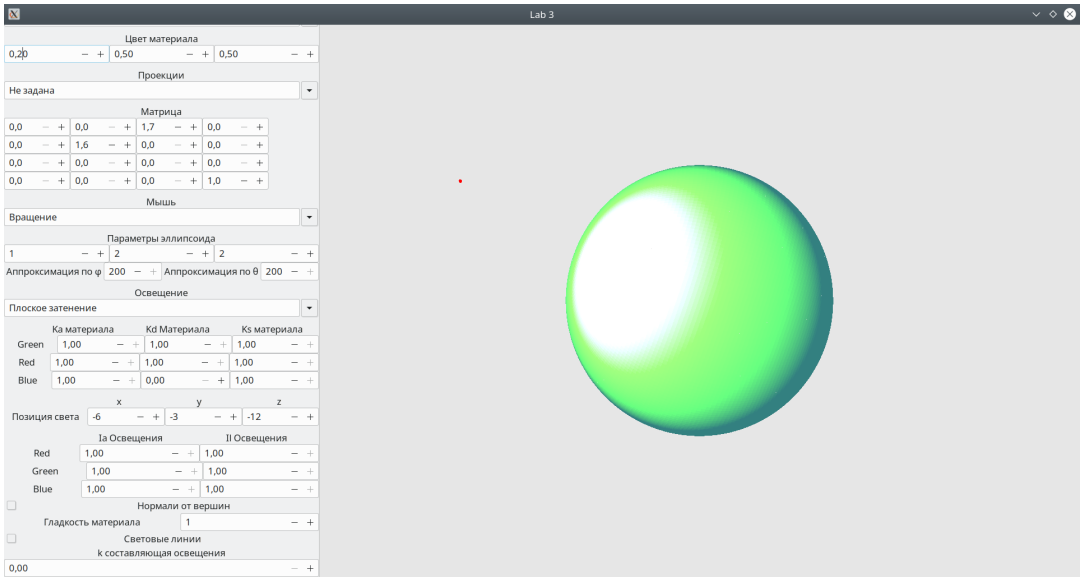
Зеркальная составляющая для затенения Гуро

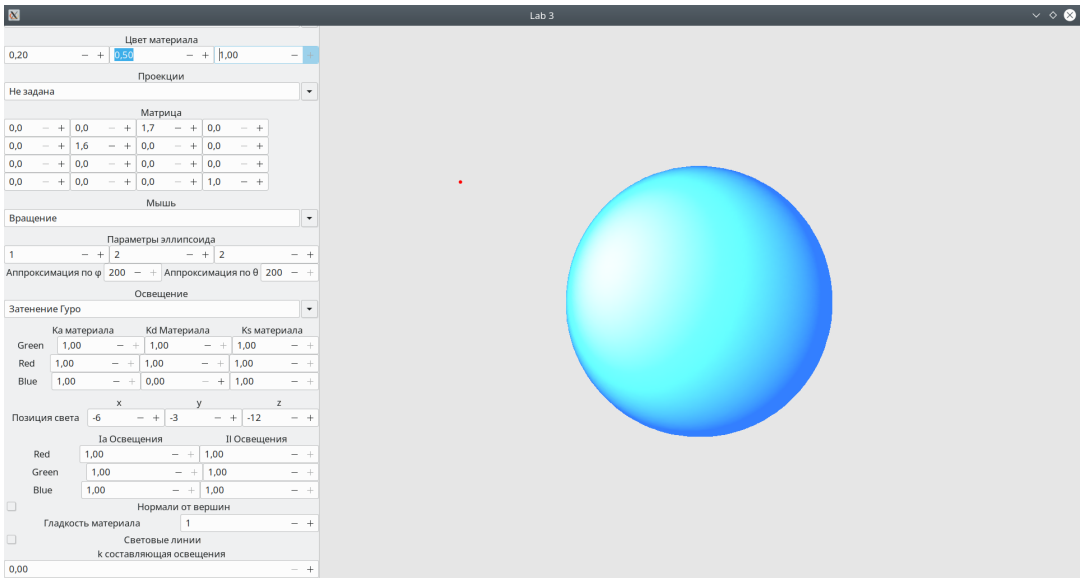
```
180 void Specular()
181 {
182     for (int i = 0; i < Verticies.Count; ++i)
183     {
184         for (int j = 0; j < Verticies[i].Count; ++j)
185         {
```

```
186             Vector3 ans = new Vector3();
187             Vector3 ks = new Vector3((float)_ksr.Value, (float)_ksg.Value, (float)_ksb.
                    Value);
188             Vector3 il = new Vector3((float)_ilr.Value, (float)_ilg.Value, (float)_ilb.
                    Value);
189
190             ans = Multiplication(ks, il);
191
192             Vector3 start = new Vector3(Verticies[i][j].PointInWorldSpace.X, Verticies[
                    i][j].PointInWorldSpace.Y, Verticies[i][j].PointInWorldSpace.Z);
193             Vector3 l = new Vector3((float)_lx.Value - start.X, (float)_ly.Value -
                    start.Y,(float)_lz.Value - start.Z);
194             l = Normalize(l);
195
196             if (Vector3.Dot(l, Verticies[i][j].NormalInWorldSpace) < 1e-6)
197             {
198                 continue;
199             }
200
201             Vector3 r = new Vector3();
202
203             r = 2 * Verticies[i][j].NormalInWorldSpace * (Vector3.Dot(l, Verticies[i][j
                    ].NormalInWorldSpace) / Vector3.Dot(Verticies[i][j].NormalInWorldSpace,
                    Verticies[i][j].NormalInWorldSpace));
204             r = r - l;
205
206             r = Normalize(r);
207
208             float cosRL = 1;
209
210             bool end = false;
211             for (int k = 0; k < _p.Value; ++k)
212             {
213                 double t = Vector3.Dot(r, l);
214                 if (t > -1e-6)
215                 {
216                     cosRL *= Vector3.Dot(r, l);
217                 }
218                 else
219                 {
220                     end = true;
221                     break;
222                 }
223             }
224
225             if (end)
226             {
227                 continue;
228             }
```

```
229 ||
230 ||            ans *= cosRL;
231 ||            ans.X = Math.Max(0, ans.X);
232 ||            ans.Y = Math.Max(0, ans.Y);
233 ||            ans.Z = Math.Max(0, ans.Z);
234 ||
235 ||            l.X = (float)_lx.Value - start.X;
236 ||            l.Y = (float)_ly.Value - start.Y;
237 ||            l.Z = (float)_lz.Value - start.Z;
238 ||            Verticies[i][j].Int += (ans / (float)(l.Length() / 30 + _k.Value));
239 ||        }
240 ||    }
241 || }
```

Скриншоты работы программы





Лабораторные работы №4-5

Ознакомление с технологией OpenGL

Задача: Создать графическое приложение с использованием OpenGL. Используя результаты предыдущей лабораторной работы, изобразить заданное тело 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.

Вариант задания: Эллипсоид.

Описание

Данная программа написана с использованием OpenGl. Параметризация фигуры происходит также, как в прошлой лабе. Реализовано затенение Фонга. Также реализованы 2 шейдера: вершинный и фрагментный. Основные составляющие освещения реализованы во фрагментном шейдере. Также есть возможности, реализованные в прошлых лабах: сдвиг, поворот, масштаб. Можно показать источник света.

Исходный код

Основные фрагменты кода.

Фрагментный шейдер

```
#version 330 core

struct Material {
    vec3 ka;
    vec3 kd;
    vec3 ks;
    float p;
};

struct Light {
    vec3 ia;
    vec3 il;
    vec3 position;
};

in vec3 normalnf;
in vec3 fragCoord;

out vec4 color;

uniform mat4 view4f;

uniform float k;
uniform vec3 c;
uniform vec3 camera;
uniform Material m;
uniform Light l;

void main() {

    vec3 position = vec3(view4f * vec4(l.position, 1) );

    vec3 background = m.ka * l.ia;
    vec3 diffuse = m.kd * l.il;

    vec3 toLight = normalize(position - fragCoord);
    diffuse *= max(dot(toLight, normalnf), 0);

    vec3 specular = m.ks * l.il;

    if (dot(toLight, normalnf) > 1e-3) {
        vec3 r = reflect(-toLight, normalnf);
        specular *= pow(max(dot(r, normalize(camera - fragCoord)), 0), m.p);
    } else {
```

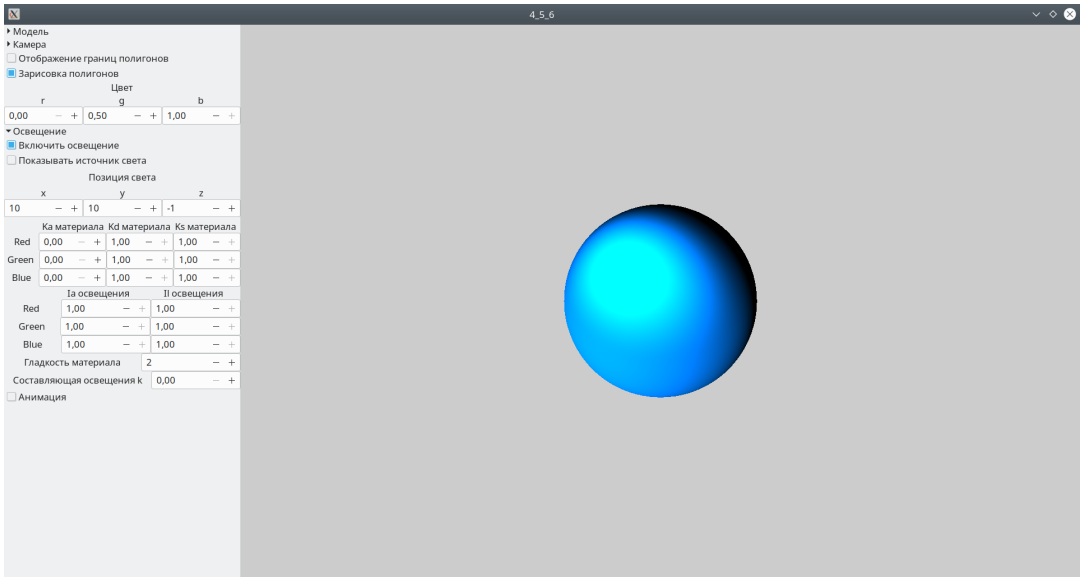
```
        specular *= 0;
    }

   color = vec4(c * background + (c * diffuse + c * specular) / (k + length(position -
       fragCoord) / 20), 1);
}
```

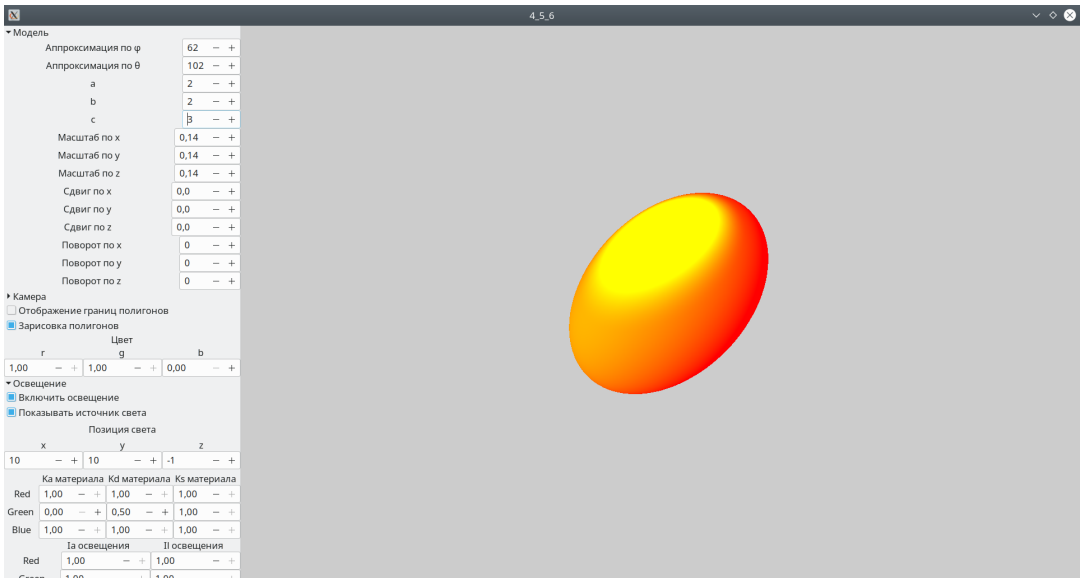
Компиляция шейдеров

```
vertexShader = gl.CreateShader(OpenGL.GL_VERTEX_SHADER);
string s = ReadFromRes("lab4-5-6.VertexShader.glsl");
gl.ShaderSource(vertexShader, s);
gl.CompileShader(vertexShader);

System.Text.StringBuilder txt = new System.Text.StringBuilder(512);
gl.GetShaderInfoLog(vertexShader, 512, (IntPtr)0, txt);
//Console.WriteLine(txt);

var glsl_tmp = new int[1];
gl.GetShader(vertexShader, OpenGL.GL_COMPILE_STATUS, glsl_tmp);
Debug.Assert(glsl_tmp[0] == OpenGL.GL_TRUE, "Shader compilation failed");

uint fragmentShader;
fragmentShader = gl.CreateShader(OpenGL.GL_FRAGMENT_SHADER);
s = ReadFromRes("lab4-5-6.FragmentShader.glsl");
gl.ShaderSource(fragmentShader, s);
gl.CompileShader(fragmentShader);

txt = new System.Text.StringBuilder(512);
gl.GetShaderInfoLog(fragmentShader, 512, (IntPtr)0, txt);
// Console.WriteLine(txt);
gl.GetShader(fragmentShader, OpenGL.GL_COMPILE_STATUS, glsl_tmp);
Debug.Assert(glsl_tmp[0] == OpenGL.GL_TRUE, "Shader compilation failed");


uint fragmentLight;
fragmentLight = gl.CreateShader(OpenGL.GL_FRAGMENT_SHADER);
s = ReadFromRes("lab4-5-6.FragmentLight.glsl");
gl.ShaderSource(fragmentLight, s);
gl.CompileShader(fragmentLight);

txt = new System.Text.StringBuilder(512);
gl.GetShaderInfoLog(fragmentLight, 512, (IntPtr)0, txt);
//Console.WriteLine(txt);
gl.GetShader(fragmentLight, OpenGL.GL_COMPILE_STATUS, glsl_tmp);
Debug.Assert(glsl_tmp[0] == OpenGL.GL_TRUE, "Shader compilation failed");

uint shaderProgram;
shaderProgram = gl.CreateProgram();
gl.AttachShader(shaderProgram, vertexShader);
```

```
332 gl.AttachShader(shaderProgram, fragmentShader);
333 gl.LinkProgram(shaderProgram);
334
335 gl.GetProgram(shaderProgram, OpenGL.GL_LINK_STATUS, glsl_tmp);
336 Debug.Assert(glsl_tmp[0] == OpenGL.GL_TRUE, "Shader program link failed");
337
338 uint lightProgram;
339 lightProgram = gl.CreateProgram();
340 gl.AttachShader(lightProgram, vertexShader);
341 gl.AttachShader(lightProgram, fragmentLight);
342 gl.LinkProgram(lightProgram);
343
344 gl.GetProgram(lightProgram, OpenGL.GL_LINK_STATUS, glsl_tmp);
345 Debug.Assert(glsl_tmp[0] == OpenGL.GL_TRUE, "Shader program link failed");
```

Скриншоты работы программы





Лабораторная работа №6

Создание шейдерных анимационных эффектов в OpenGL 2.1

Задача: Для поверхности, созданной в предыдущей лабораторной работе, обеспечить выполнение шейдерного эффекта.

Вариант задания: Эллипсоид.

Анимация: Координата x изменяется по закону $x * cos(t)$, координата y изменяется по закону $y = ysin(x + t)$.

Описание

Анимация реализована в вершинном шейдере. Каждый раз в шейдер передаётся время, потом координаты нормируются относительно окна и умнжаются на косинус/синус сооответственно, после чего они восстанавливаются.

Исходный код

Основные фрагменты кода.

Вершинный шейдер

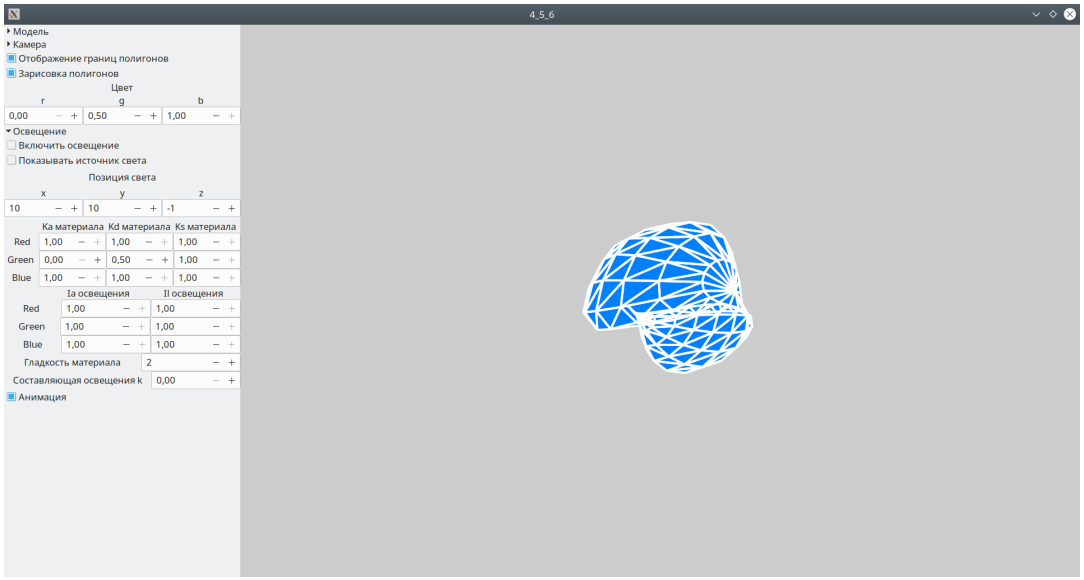
```
#version 330 core
layout (location = 0) in vec3 coord3f;
layout (location = 1) in vec3 normalf;

uniform mat4 proj4f;
uniform mat4 view4f;
uniform mat4 model4f;

uniform bool animation;
uniform float t;

out vec3 normalnf;
out vec3 fragCoord;

uniform float width;
uniform float height;

void main(void) {
    if (animation) {
        gl_Position = vec4(coord3f, 1.0);

        gl_Position.y = float(gl_Position.y / float(0.5 * float(height)));

        gl_Position.y *= sin(gl_Position.x + t);
        gl_Position.x = float(gl_Position.x / float(0.5 * float(width)));

        gl_Position.x *= cos(t);

        gl_Position.x = float(gl_Position.x * float(0.5 * float(width)));
       gl_Position.y = gl_Position.y *(0.5 * height);

        gl_Position = (proj4f * view4f * model4f) * gl_Position;
    } else {
          gl_Position = (proj4f * view4f * model4f) * vec4(coord3f, 1.0);
    }
    normalnf = normalize(vec3(view4f * model4f * vec4(normalf, 0.0)));
    fragCoord = vec3(view4f * model4f * vec4(coord3f, 1.0));
}
```

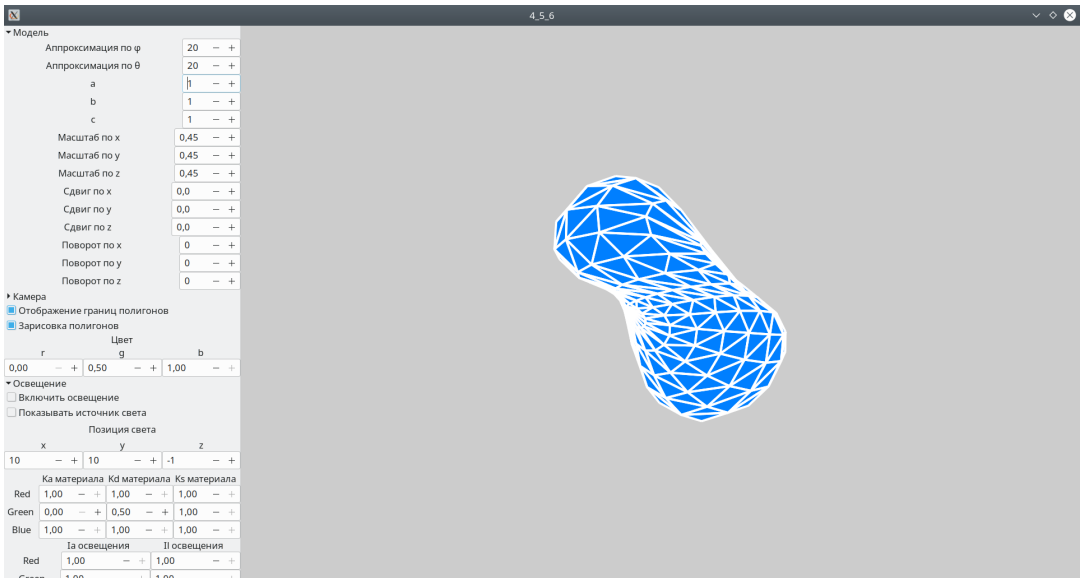
Подсчет времени и передача его в шейдер

```
var frame_clock = _drawingArea.Context.Window.FrameClock;
frame_clock.Update += (_, _) => _drawingArea.QueueRender();
frame_clock.BeginUpdating();

```

```
388 if (_animation.Active)
389 {
390     gl.UseProgram(shaderProgram);
391     loc = gl.GetUniformLocation(shaderProgram, "animation");
392     gl.Uniform1(loc, 1);
393     loc = gl.GetUniformLocation(shaderProgram, "t");
394     gl.Uniform1(loc, (float)((frame_clock.FrameTime - startTime)/1000000));
395 
396     gl.UseProgram(lightProgram);
397     loc = gl.GetUniformLocation(lightProgram, "animation");
398     gl.Uniform1(loc, 1);
399     loc = gl.GetUniformLocation(lightProgram, "t");
400     gl.Uniform1(loc, (float)((frame_clock.FrameTime - startTime)/1000000));
401 }
```

Скриншоты работы программы





Лабораторная работа №7

Построение плоских полиномиальных кривых

Задача: Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек и, при необходимости, значений касательных векторов и натяжения.

Вариант задания: Кривая Безье 2-й степени.

Описание

Кривая Безье 2-й степени строится по 3 точкам и имеет вид: $B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$, где P_0, P_1, P_2 - точки, а t - параметр от 0 до 1. Для построения составной прямой нужно чередовать точки, то есть брать первую вторую третью, третью чевертую пятую и т.д. Также есть возможность двигать ближайшие точки, для это надо пройтись по всем точкам и посчитать расстояние от них до позиции мыши, выбрав ту, у которой это расстояние будет наименьшим.

Исходный код

Генерация точек

```
402 void Figure()
403 {
404
405     int j = 0;
406
407     mas = new List<float>();
408
409     double dt = 1.0 / _tt.Value;
410     for (int i = 1; i < points.Count - 2; i += 2)
411     {
412         for (float t = 0; t < 1; t += (float)dt)
413         {
414             mas.Add(points[i].X * (1 - t) * (1 - t) + 2 * t * (1 - t) * points[i + 1].X
                      + t * t * points[i + 2].X);
415             mas.Add (points[i].Y * (1 - t) * (1 - t) + 2 * t * (1 - t) * points[i + 1].
                     Y + t * t * points[i + 2].Y);
416             j += 2;
417         }
418     }
419
420     for (float t = 0; t < 1; t += (float)dt)
421     {
422         mas.Add(points[points.Count - 1].X * (1 - t) * (1 - t) + 2 * t * (1 - t) *
                 points[0].X + t * t * points[1].X);
423         mas.Add (points[points.Count - 1].Y * (1 - t) * (1 - t) + 2 * t * (1 - t) *
                 points[0].Y + t * t * points[1].Y);
424         j += 2;
425     }
426
427
428     drawpoints = new List<float>();
429
430     for (int i = 0; i < points.Count; i += 1)
431     {
432         drawpoints.Add(points[i].X);
433         drawpoints.Add(points[i].Y);
434     }
435     setbuff = true;
436 }
```

Скриншоты работы программы