

Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная
математика»

Кафедра 806 «Вычислительная математика и
программирование»

Лабораторные работы №4-5 по курсу «Компьютерная графика»

Студент: М. А. Инютин
Преподаватель: А. В. Морозов
Группа: М8О-307Б-19
Дата:
Оценка:
Подпись:

Москва, 2022

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: Шаровой сектор.

1 Описание

OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Для отображения на экран необходимо использовать Vertex Buffer. Этот метод хранит информацию об объекте непосредственно в видеопамяти. Напрямую обратиться к ней нельзя, OpenGL предоставляет возможность создания буферов вершин и индексов, куда следует скопировать данные об объекте.

Шейдер (Shader) — программа для процессора графической карты (GPU), управляющая поведением шейдерной стадии графического конвейера и занимающаяся обработкой соответствующих входных данных.

В работе используются вершинный и фрагментный шейдеры. Первый трансформирует вершины из локального пространства в пространство камеры. Второй отвечает за закраску геометрического объекта, интерполирует параметры и рассчитывает цвет отдельно взятого пикселя.

Шейдеры можно писать на GLSL (Graphics Library Shader Language) — высокоуровневом языке используемом для шейдеров OpenGL. Синтаксис языка очень похож на синтаксис C.

2 Исходный код

Файл *Figure.cs* содержит код для генерации фигуры. После генерации точек, образующих поверхность, задаются полигоны, вычисляются вектора нормали к вершинам.

```
1 public Figure(int paramPhi, int paramTheta, double paramR, double maxTheta, Misc.  
   Colour col)  
2 {  
3     ParamPhi = paramPhi;  
4     ParamTheta = paramTheta - 1;  
5     R = paramR;  
6     MaxTheta = Misc.ToRadians(maxTheta);  
7     DeltaPhi = Misc.ToRadians(Misc.MAX_DEG / (double)(ParamPhi));  
8     DeltaTheta = MaxTheta / (double)(ParamTheta);  
9     FigureColour = col;  
10    GenVertices();  
11    GenFigure();  
12    GenVertexNormals();  
13 }  
14  
15 private void GenVertices()  
16 {  
17     CenterLow = new Vertex(0, 0, R * Math.Cos(MaxTheta), FigureColour);  
18     CenterHigh = new Vertex(0, 0, R, FigureColour);  
19     Vertices = new List<Vertex>();  
20     Vertices.Add(CenterLow);  
21     Vertices.Add(CenterHigh);  
22     LayerVertices = new List< List<Vertex> >();  
23     double stepTheta = DeltaTheta;  
24     for (int i = 0; i < ParamTheta; ++i)  
25     {  
26         double curZ = R * Math.Cos(stepTheta);  
27         double curXY = R * Math.Sin(stepTheta);  
28         double stepPhi = 0;  
29         List<Vertex> curLayerVertices = new List<Vertex>();  
30         for (int j = 0; j < ParamPhi; ++j)  
31         {  
32             Vertex curVertex = new Vertex(curXY * Math.Cos(stepPhi), curXY * Math.Sin(  
                 stepPhi), curZ, FigureColour);  
33             curLayerVertices.Add(curVertex);  
34             Vertices.Add(curVertex);  
35             stepPhi = stepPhi + DeltaPhi;  
36         }  
37         LayerVertices.Add(curLayerVertices);  
38         stepTheta = stepTheta + DeltaTheta;  
39     }  
40     for (int i = 0; i < Vertices.Count; ++i)  
41     {  
42         Vertices[i].Id = i;
```

```

43     }
44 }
45
46 private void GenFigure()
47 {
48     Polygons = new List<Polygon>();
49     GenLow();
50     GenHigh();
51     GenSphere();
52 }
53
54 private void GenLow()
55 {
56     int lastLayerInd = LayerVertices.Count - 1;
57     for (int j = 0; j < ParamPhi; ++j)
58     {
59         Vertex a = LayerVertices[lastLayerInd][j];
60         Vertex b = LayerVertices[lastLayerInd][(j + 1) % ParamPhi];
61         Polygons.Add(new Polygon(CenterLow, b, a));
62     }
63 }
64
65 private void GenSphere()
66 {
67     for (int i = 0; i < ParamTheta - 1; ++i)
68     {
69         for (int j = 0; j < ParamPhi; ++j)
70         {
71             Vertex a = LayerVertices[i][j];
72             Vertex b = LayerVertices[i][(j + 1) % ParamPhi];
73             Vertex c = LayerVertices[i + 1][(j + 1) % ParamPhi];
74             Vertex d = LayerVertices[i + 1][j];
75             Polygons.Add(new Polygon(a, c, b));
76             Polygons.Add(new Polygon(c, a, d));
77         }
78     }
79 }
80
81 private void GenHigh()
82 {
83     for (int j = 0; j < ParamPhi; ++j)
84     {
85         Vertex a = LayerVertices[0][j];
86         Vertex b = LayerVertices[0][(j + 1) % ParamPhi];
87         Polygons.Add(new Polygon(CenterHigh, a, b));
88     }
89 }
90
91 public void GenVertexNormals()

```

```

92 | {
93 |     foreach (Vertex vert in Vertices)
94 |     {
95 |         vert.Normal = new Vector4D();
96 |         vert.Normal.W = 0;
97 |     }
98 |     foreach (Polygon poly in Polygons)
99 |     {
100 |         Vector4D polyN = poly.NormalVector();
101 |         polyN.Normalize();
102 |         foreach (Vertex vert in poly.Data)
103 |         {
104 |             vert.Normal = vert.Normal + polyN;
105 |             vert.Normal.W += 1;
106 |         }
107 |     }
108 |     foreach (Vertex vert in Vertices)
109 |     {
110 |         vert.Normal.X = vert.Normal.X / vert.Normal.W;
111 |         vert.Normal.Y = vert.Normal.Y / vert.Normal.W;
112 |         vert.Normal.Z = vert.Normal.Z / vert.Normal.W;
113 |         vert.Normal.W = 0;
114 |         vert.Normal.Normalize();
115 |     }
116 | }

```

Geometry.cs содержит классы векторов и матрицы, методы для работы с ними. В *Misc.cs* заданы константы, используемые во всей программе.

Вся работа с интерфейсом, компиляция шейдеров, создание буфера вершин описаны в *MainWindow.cs*. Компиляция шейдеров:

```
117 private void CompileShaders()
118 {
119     int[] success = new int[1];
120     System.Text.StringBuilder txt = new System.Text.StringBuilder(512);
121
122     vertexShader = gl.CreateShader(OpenGL.GL_VERTEX_SHADER);
123     gl.ShaderSource(vertexShader, HelpUtils.ReadFromRes("Phong.vert"));
124     gl.CompileShader(vertexShader);
125
126     gl.GetShader(vertexShader, OpenGL.GL_COMPILE_STATUS, success);
127     if (success[0] == 0)
128     {
129         gl.GetShaderInfoLog(vertexShader, 512, (IntPtr)0, txt);
130         Console.WriteLine("Vertex shader compilation failed!\n" + txt);
131     }
132
133     fragmentShader = gl.CreateShader(OpenGL.GL_FRAGMENT_SHADER);
134     gl.ShaderSource(fragmentShader, HelpUtils.ReadFromRes("Phong.frag"));
135     gl.CompileShader(fragmentShader);
136
137     gl.GetShader(fragmentShader, OpenGL.GL_COMPILE_STATUS, success);
138     if (success[0] == 0)
139     {
140         gl.GetShaderInfoLog(fragmentShader, 512, (IntPtr)0, txt);
141         Console.WriteLine("Fragment shader compilation failed!\n" + txt);
142     }
143
144     shaderProgram = gl.CreateProgram();
145     gl.AttachShader(shaderProgram, vertexShader);
146     gl.AttachShader(shaderProgram, fragmentShader);
147     gl.LinkProgram(shaderProgram);
148
149     gl.GetProgram(vertexShader, OpenGL.GL_LINK_STATUS, success);
150     if (success[0] == 0)
151     {
152         gl.GetProgramInfoLog(vertexShader, 512, (IntPtr)0, txt);
153         Console.WriteLine("Shader program linking failed!\n" + txt);
154     }
155 }
```

Вершинный шейдер *Phong.vert* преобразует локальные координаты в видовые, затем передаёт их фрагментному шейдеру.

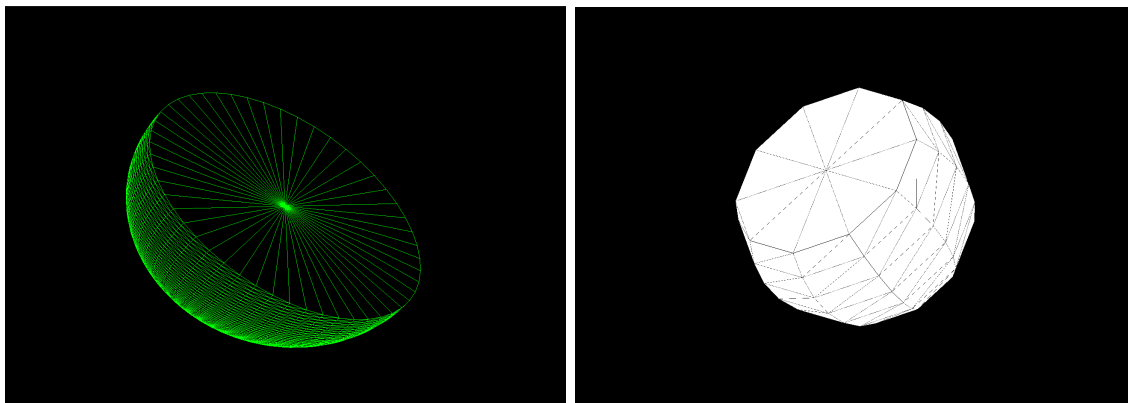
```
1 | #version 150 core
2 |
3 | in vec3 cord3f;
4 | in vec3 col3f;
5 | in vec3 norm3f;
6 | out vec3 position;
7 | out vec3 normal;
8 |
9 | uniform mat4 proj4f;
10 | uniform mat4 view4f;
11 | uniform mat4 model4f;
12 | uniform bool useSingleColor;
13 | uniform vec3 singleColor3f;
14 |
15 | void main(void) {
16 |     vec4 vertexPos = vec4(cord3f, 1.0f);
17 |     vertexPos = (proj4f * view4f * model4f) * vertexPos;
18 |     vec4 vertexNormal = vec4(norm3f, 0.0f);
19 |     vertexNormal = (view4f * model4f) * vertexNormal;
20 |     position = vertexPos.xyz;
21 |     normal = normalize(vertexNormal.xyz);
22 |     gl_Position = vertexPos;
23 | }
```


Фрагментный шейдер *Phong.frag* реализует затенение Фонга, интерполирующее нормаль к точке.

```
24 #version 150 core
25
26 in vec3 position;
27 in vec3 normal;
28 out vec4 color;
29
30 uniform bool useSingleColor;
31 uniform vec3 singleColor3f;
32 uniform vec3 ka3f;
33 uniform vec3 kd3f;
34 uniform vec3 ks3f;
35 uniform vec3 light3f;
36 uniform vec3 ia3f;
37 uniform vec3 il3f;
38 uniform float p;
39 uniform vec3 camera = vec3(0, 0, -1e9f);
40 uniform float k = 0.5;
41
42 void main(void) {
43     if (useSingleColor) {
44         color = vec4(singleColor3f, 1.0f);
45         return;
46     }
47     vec3 res = singleColor3f;
48     vec3 l = light3f - position;
49     float d = length(l);
50     l = normalize(l);
51     vec3 s = normalize(camera - position);
52     vec3 r = normalize(2 * normal * dot(normal, l) - l);
53     float diffusal = dot(l, normal);
54     float specular = dot(r, s);
55     if (diffusal < 1e-3) {
56         diffusal = 0;
57         specular = 0;
58     }
59     if (specular < 1e-3) {
60         specular = 0;
61     }
62     for (int i = 0; i < 3; ++i) {
63         res[i] = res[i] * (ia3f[i] * ka3f[i] + il3f[i] * (kd3f[i] * diffusal + ks3f[i]
64             * pow(specular, p)) / (d + k));
65     }
66     color = vec4(res, 1.0f);
67 }
```

3 Демонстрация работы программы

Каркасная визуализация шарового сектора



Освещение Фонга для одного источника

