Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: «Вычислительная математика и программирование»
Дисциплина: «Компьютерная графика»

Лабораторная работа №4-6 по курсу «Компьютерная графика»

Тема: Ознакомление с технологией OpenGL и языком GLSL. Создание шейдерных анимационных эффектов в OpenGL 2.1.

Студент: Тимофеев А. В.
Преподаватель: Морозов А. В.
Группа: М80-307Б
Дата:
Оценка:
Подпись:

Москва, 2022

Постановка задачи

Задание: 1)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.
2)Для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта:
Вариант: 5. Анимация. Координата X изменяется по закону X*cos(t), координата Y изменяется по закону Y = Ysin(X+t).

Решение задачи

Возьмем за основу способ генерации фигуры и вычисление интенсивностей цвета такие же, какие они были в Л.Р. 3. При этом важно понимать, что затенение Фонга отличается от Гуро тем, что при его использовании для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей - это будет осуществляться автоматически при загрузке нормалей как выходные атрибуты в вершинном шейдере OpenGL. Стоит также отметить, что все вычисления, связанные с освещением, будут происходить исключительно во фрагментном шейдере OpenGL. Для корректной отрисовки фигуры в среде OpenGL необходимо задать матрицу проекций (свойство OpenGL.GL_PROJECTION) и объектно-видовую матрицу (свойство OpenGL.GL_MODELVIEW) - их следует предварительно вычислить и только затем подгрузить в OpenGL. После задания этих матриц будет автоматически производиться следующее преобразование для координат некоторой вершины P :

$$P_{transformed} = M_{projection} * M_{modelview} * P$$

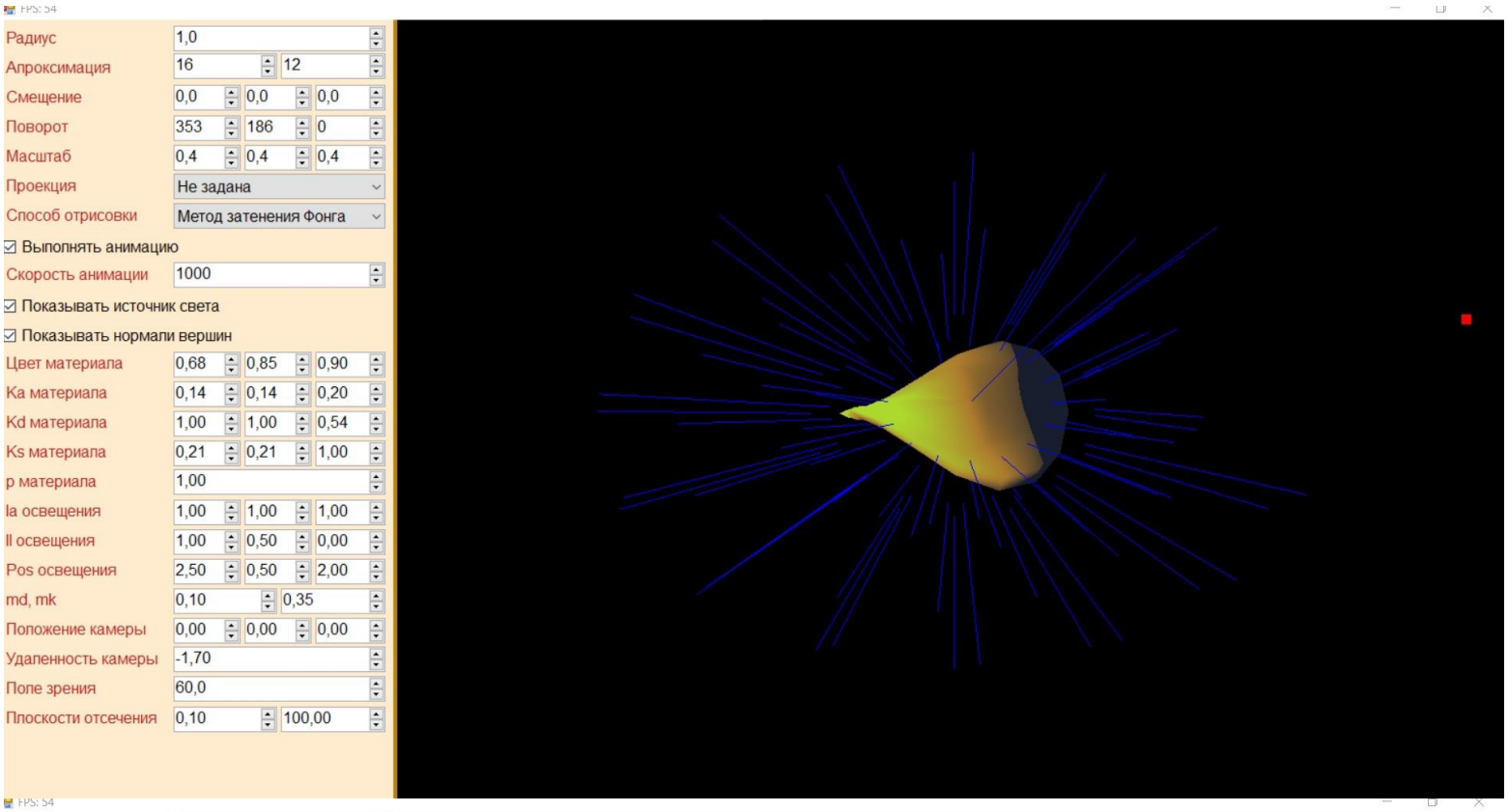
Чтобы вся информация о фигуре хранилось исключительно в GPU необходимо загружать её в буфер вершин VBO. При этом для вызова метода отрисовки OpenGL необходимо знать, в каком порядке следуют вершины в массиве. При работе с прямым эллиптическим цилиндром будем использоваться порядок обхода GL_TRIANGLE_FAN - для оснований, и GL_TRIANGLE_STRIP - для боковой поверхности (вызывая этот метод отрисовки таким обходом для каждого этажа соответственно).

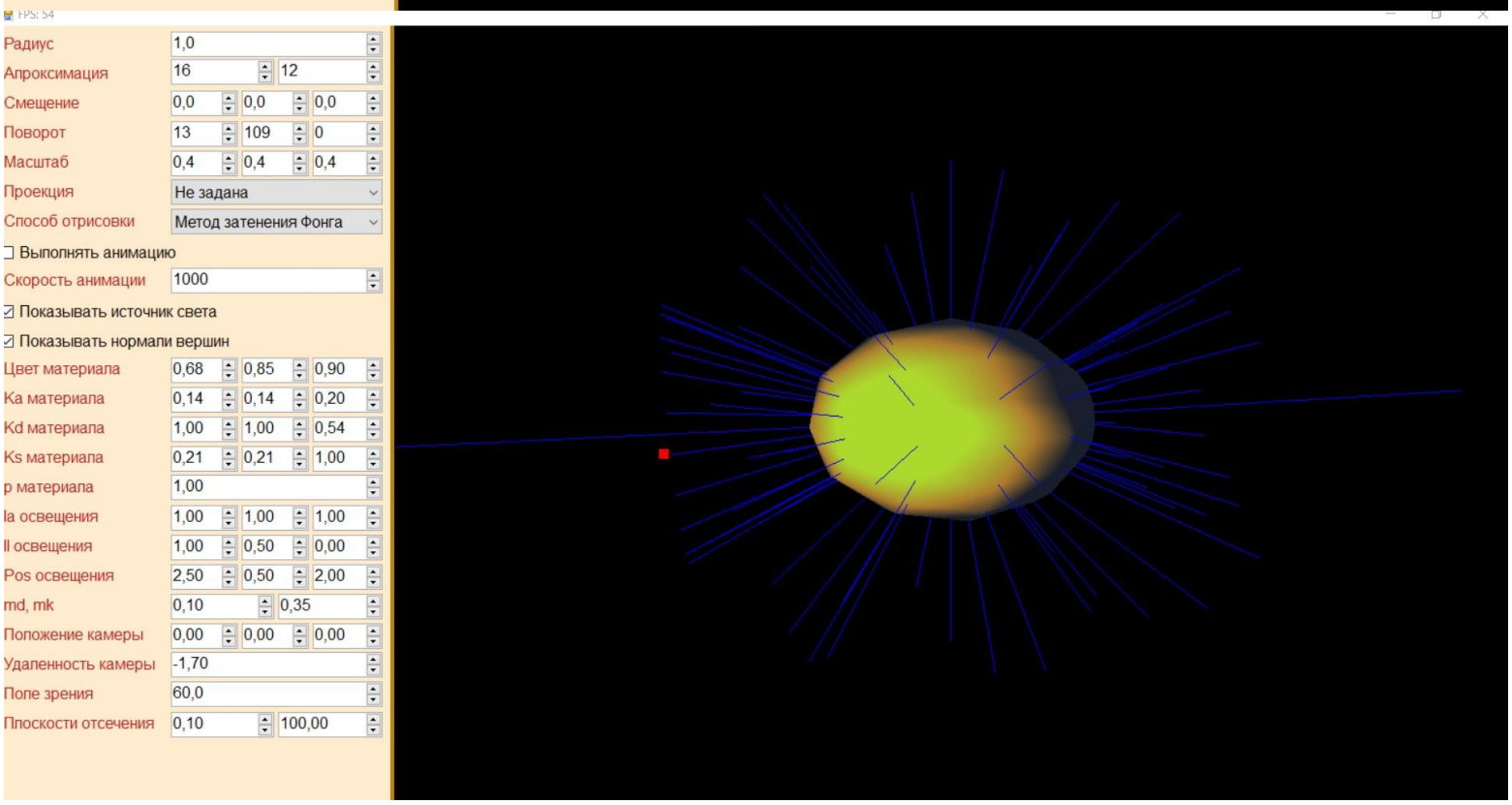
Для создания требуемого шейдерного эффекта достаточно немного изменить вершинный шейдер - менять указанным способом координаты Y загружаемых вершин:

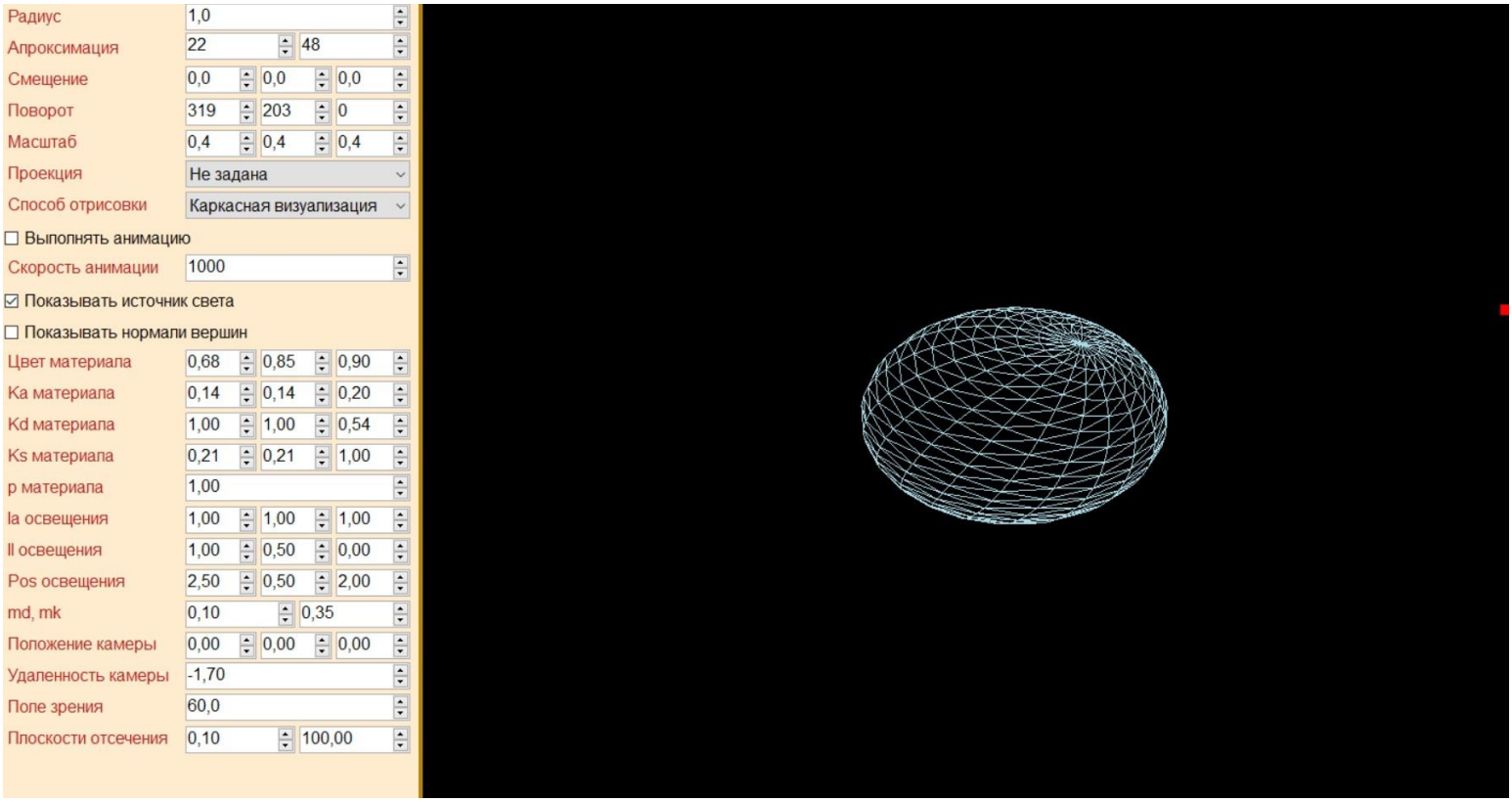
$$Y = Y \cdot \cos(t + Y)$$

Обновленная величина времени t будет подгружаться в шейдер каждый раз при выполнении основного цикла работы программы. Стоит отметить, что при изменении координат вершин не требуется пересчитывать все ранее вычисленные нормали фигуры - изменения при анимации незначительные и в принципе этого не требуют.

Пример работы


FPS: 54
Радиус 1,0
Апроксимация 16 12
Смещение 0,0 0,0 0,0
Поворот 353 186 0
Масштаб 0,4 0,4 0,4
Проекция Не задана
Способ отрисовки Метод затенения Фонга
Выполнять анимацию
Скорость анимации 1000
Показывать источник света
Показывать нормали вершин
Цвет материала 0,68 0,85 0,90
Ka материала 0,14 0,14 0,20
Kd материала 1,00 1,00 0,54
Ks материала 0,21 0,21 1,00
p материала 1,00
Ia освещения 1,00 1,00 1,00
Il освещения 1,00 0,50 0,00
Pos освещения 2,50 0,50 2,00
md, mk 0,10 0,35
Положение камеры 0,00 0,00 0,00
Удаленность камеры -1,70
Поле зрения 60,0
Плоскости отсечения 0,10 100,00


FPS: 54
Радиус 1,0
Апроксимация 16 12
Смещение 0,0 0,0 0,0
Поворот 13 109 0
Масштаб 0,4 0,4 0,4
Проекция Не задана
Способ отрисовки Метод затенения Фонга
Выполнять анимацию
Скорость анимации 1000
Показывать источник света
Показывать нормали вершин
Цвет материала 0,68 0,85 0,90
Ka материала 0,14 0,14 0,20
Kd материала 1,00 1,00 0,54
Ks материала 0,21 0,21 1,00
p материала 1,00
Ia освещения 1,00 1,00 1,00
Il освещения 1,00 0,50 0,00
Pos освещения 2,50 0,50 2,00
md, mk 0,10 0,35
Положение камеры 0,00 0,00 0,00
Удаленность камеры -1,70
Поле зрения 60,0
Плоскости отсечения 0,10 100,00



Листинг программы

(классы)

Program.cs

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct Vertex{
    public Vertex(float px, float py, float pz, float pw, byte pr, byte pg, byte pb){
        vx = px;
        vy = py;
        vz = pz;
        vw = pw;
        nx = 0;
        ny = 0;
        nz = 0;
        nw = 0;
        r = pr;
        g = pg;
        b = pb;
    }
    public void SetNorm(float nx, float ny, float nz, float nw){
        this.nx = nx;
        this.nx = ny;
        this.nx = nz;
        this.nx = nw;
```

```
        }
        public readonly float vx, vy, vz, vw;
        public float nx, ny, nz, nw;
        public readonly byte r, g, b;
    }
    private void ChangeNormale(ref Vertex vertex, DVector4 normale){
        var check = new DVector4(vertex.nx, vertex.ny, vertex.nz, 0);
        check += normale;
        vertex.nx = (float) check.X;
        vertex.ny = (float) check.Y;
        vertex.nz = (float) check.Z;
    }
```

shader1.vert

```
#version 150 core
attribute vec4 Normal;
attribute vec4 Coord;
out vec3 FragNormale;
out vec3 FragVertex;
uniform mat4 Projection;
uniform mat4 ModelView;
uniform mat4 NormalMatrix;
uniform mat4 ModelMatrix;
uniform mat4 PointMatrix;
uniform float Time;
void main(void){
    vec4 Coord_now = Coord;
    if (Time > 0){
        Coord_now.x = Coord.x * cos(Time);
        Coord_now.y = Coord.y * sin(Time + Coord_now.x);
    }
    FragVertex = vec3(PointMatrix * Coord_now);
    FragNormale = vec3(NormalMatrix * Normal);
    FragNormale = normalize(FragNormale);
    gl_Position = (Projection * ModelView) * Coord_now;
}
```

shader2.frag

```
#version 150 core
in vec3 FragNormale;
in vec3 FragVertex;
```

```
uniform vec3 Ka_Material;
uniform vec3 Kd_Material;
uniform vec3 Ks_Material;
uniform float P_Material;
uniform vec3 Ia_Material;
uniform vec3 Il_Material;
uniform vec3 LightPos;
uniform vec2 Parameters;
uniform vec3 CameraPos;
uniform vec3 FragColor;
void main(void){
    vec3 L = vec3(LightPos.x - FragVertex.x, LightPos.y - FragVertex.y, LightPos.z - FragVertex.z);
    float dist = length(L);
    L = normalize(L);
    vec3 FragNormaleW = normalize(FragNormale);
    float I_red = Ia_Material.x * Ka_Material.x;
    float I_green = Ia_Material.y * Ka_Material.y;
    float I_blue = Ia_Material.z * Ka_Material.z;
    I_red += clamp(0, 1, Il_Material.x * Kd_Material.x * dot(L, FragNormaleW) / (Parameters[0] * dist + Parameters[1]));
    I_green += clamp(0, 1, Il_Material.y * Kd_Material.y * dot(L, FragNormaleW) / (Parameters[0] * dist + Parameters[1]));
    I_blue += clamp(0, 1, Il_Material.z * Kd_Material.z * dot(L, FragNormaleW) / (Parameters[0] * dist + Parameters[1]));
    if (dot(L, FragNormaleW) > 0){
        vec3 S = vec3(CameraPos.x - FragVertex.x, CameraPos.y - FragVertex.y, CameraPos.z - FragVertex.z);
        vec3 R = vec3(reflect(-L, FragNormale));
        S = normalize(S);
        R = normalize(R);
        if (dot(R, S) > 0){
            I_red += clamp(0, 1, Il_Material.x * Ks_Material.x * pow(dot(R, S), P_Material) / (Parameters[0] * dist + Parameters[1]));
            I_green += clamp(0, 1, Il_Material.y * Ks_Material.y * pow(dot(R, S), P_Material) / (Parameters[0] * dist + Parameters[1]));
            I_blue += clamp(0, 1, Il_Material.z * Ks_Material.z * pow(dot(R, S), P_Material) / (Parameters[0] * dist + Parameters[1]));
        }
    }
    I_red = min(1, I_red);
```

```
    I_green = min(1, I_green);
    I_blue = min(1, I_blue);
    vec4 result = vec4(FragColor.x * I_red, FragColor.y * I_green, FragColor.z * I_blue, 1);
    gl_FragColor = result;
}
```

Вывод

В результате выполнения данной лабораторной работы я познакомился с возможностями OpenGL поотрисовки 3D фигур, в частности, использованием буфера вершин, шейдеров, подгружаемых глобальных преобразующих матриц и многого другого. Полученные знания я применил для отрисовки прямого эллиптического цилиндра с затенением по Фонгу.