Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Программирование мобильных устройств

Лабораторная работа №4

Модель Фонга

Выполнил: студент 4 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверил: доцент кафедры ПМиК

Нечта Иван Васильевич

Новосибирск, 2020

**Содержание**

1. **Постановка задачи..………………………………………………………3**
2. **Исходный код……..……………………………………………………….3**
3. **Результаты работы…..………………………………………………….10**

**Постановка задачи**

Создать сферу произвольного цвета, освещенную по модели Фонга

**Исходный код**

**MyGL20Renderer.java – отрисовка и инициализация сферы**

public class MyGL20Renderer implements GLSurfaceView.Renderer

{

private Sphere sphere;

float[] vm\_matrix = new float[16];

float[] mvp\_matrix = new float[16];

float[] pvm\_matrix = new float[16];

float[] invert\_matrix = new float[16];

float[] normal\_matrix = new float[16];

float[] rotation\_x\_matrix = new float[16];

float[] rotation\_y\_matrix = new float[16];

private final static long TIME = 10000;

private final float[] projection\_matrix = new float[16];

@Override

public void onSurfaceCreated(GL10 unused, EGLConfig cfg)

{

sphere = new Sphere(30, 60);

GLES20.glClearColor(215, 230, 45, 1.0f);

GLES20.glEnable(GL10.GL\_DEPTH\_TEST);

GLES20.glDepthFunc(GL10.GL\_LEQUAL);

}

@Override

public void onSurfaceChanged(GL10 unused, int width, int height)

{

GLES20.glViewport(0, 0, width, height);

Matrix.frustumM(projection\_matrix, 0, -(float) width / height, (float) width / height, -1, 1, 1, 20);

}

@Override

public void onDrawFrame(GL10 unused)

{

float[] scale\_matrix = new float[16];

float[] final\_matrix = new float[16];

float[] temporary\_matrix = new float[16];

float angle = (float) (Math.cos((float)(SystemClock.uptimeMillis() % TIME) / TIME \* 2 \* 3.1415926f) \* 4f);

GLES20.glClear(GLES20.GL\_COLOR\_BUFFER\_BIT|GLES20.GL\_DEPTH\_BUFFER\_BIT);

Matrix.setLookAtM(vm\_matrix, 0, -1, 0, -1f, -1f, 0f, 0f, 25.0f, 5.0f, 25.0f);

Matrix.multiplyMM(pvm\_matrix, 0, projection\_matrix, 0, vm\_matrix, 0);

Matrix.setRotateM(rotation\_x\_matrix, 0, 0, 5.0f, 5.0f, 0f);

Matrix.setRotateM(rotation\_y\_matrix, 0,0, 5.0f, 5.0f, 0);

Matrix.multiplyMM(temporary\_matrix, 0, pvm\_matrix, 0, rotation\_x\_matrix, 0);

Matrix.multiplyMM(mvp\_matrix, 0, temporary\_matrix, 0, rotation\_y\_matrix, 0);

Matrix.multiplyMM(temporary\_matrix, 0, vm\_matrix, 0, rotation\_x\_matrix, 0);

Matrix.multiplyMM(invert\_matrix, 0, temporary\_matrix, 0, rotation\_y\_matrix, 0);

Matrix.invertM(temporary\_matrix, 0, invert\_matrix, 0);

Matrix.transposeM(normal\_matrix, 0, temporary\_matrix, 0);

Matrix.setLookAtM(vm\_matrix, 0, angle, 1, angle, 0, 0, 0, 1, 1, 1);

Matrix.setIdentityM(scale\_matrix, 0);

Matrix.scaleM(scale\_matrix, 0, 1.5f, 1.5f, 0.5f);

Matrix.multiplyMM(temporary\_matrix, 0, mvp\_matrix, 0, scale\_matrix, 0);

Matrix.translateM(final\_matrix, 0, temporary\_matrix, 0, -1.7f, 0f, 5f);

sphere.draw(final\_matrix, normal\_matrix, vm\_matrix);

}

}

**ShaderUtils.java – класс работы с шейдерами**

public class ShaderUtils

{

public static int shader\_init(int vertex\_shader, int fragment\_shader)

{

final int init\_id = glCreateProgram();

glAttachShader(init\_id, vertex\_shader);

glAttachShader(init\_id, fragment\_shader);

glLinkProgram(init\_id);

final int[] linkStatus = new int[1];

glGetProgramiv(init\_id, GL\_LINK\_STATUS, linkStatus, 0);

if (linkStatus[0] == 0)

{

glDeleteProgram(init\_id);

return -2;

}

return init\_id;

}

static int shader\_handler(int type, String shader)

{

final int shader\_id = glCreateShader(type);

if (shader\_id == 0)

{

return -1;

}

glShaderSource(shader\_id, shader);

glCompileShader(shader\_id);

final int[] is\_shader\_good = new int[1];

glGetShaderiv(shader\_id, GL\_COMPILE\_STATUS, is\_shader\_good, 0);

if (is\_shader\_good[0] == 0)

{

glDeleteShader(shader\_id);

return -2;

}

return shader\_id;

}

public static int create\_shaders()

{

String vertex\_shader = "uniform mat4 uMVPMatrix, uMVMatrix, uNormalMat;" +

"attribute vec4 vPosition;" +

"attribute vec4 vColor;" +

"attribute vec3 vNormal;" +

"varying vec4 varyingColor;" +

"varying vec3 varyingNormal;" +

"varying vec3 varyingPos;" +

"void main()" +

"{" +

"varyingColor = vColor;" +

"varyingNormal= vec3(uNormalMat \* vec4(vNormal, 0.0));" +

"varyingPos = vec3(uMVMatrix \* vPosition);" +

"gl\_Position = uMVPMatrix \* vPosition;" +

"}";

String fragment\_shader = "precision mediump float;" +

"varying vec4 varyingColor; varying vec3 varyingNormal;" +

"varying vec3 varyingPos;" +

"uniform vec3 lightDir;" +

"void main()" +

"{" +

"float Ns = 40.0;" +

"float kd = 0.9, ks = 0.9;" +

"vec4 light = vec4(1.0, 1.0, 1.0, 1.0);" +

"vec4 lightS = vec4(1.0, 1.0, 1.0, 1.0);" +

"vec3 Nn = normalize(varyingNormal);" +

"vec3 Ln = normalize(lightDir);" +

"vec4 diffuse = kd \* light \* max(dot(Nn, Ln), 0.0);" +

"vec3 Ref = reflect(Nn, Ln);" +

"float spec = pow(max(dot(Ref, normalize(varyingPos)), 0.0), Ns);" +

"vec4 specular = lightS \* ks \* spec;" +

"gl\_FragColor = varyingColor \* diffuse + specular;" +

"}";

int vertex\_shader\_id = shader\_handler(GL\_VERTEX\_SHADER, vertex\_shader);

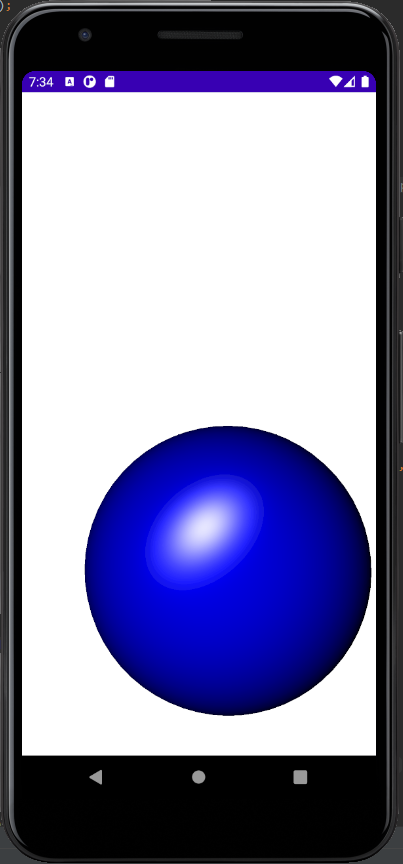
int fragment\_shader\_id = shader\_handler(GL\_FRAGMENT\_SHADER, fragment\_shader);

return shader\_init(vertex\_shader\_id, fragment\_shader\_id);

}

}

**Результаты работы**

****