Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК

Лабораторная работа №4
по дисциплине
«Программирование мобильных устройств»

Выполнил:
студент гр. ИП-813
Бурдуковский И.А.

Проверила:
Павлова У.В.

Новосибирск 2021

Оглавление

Задание

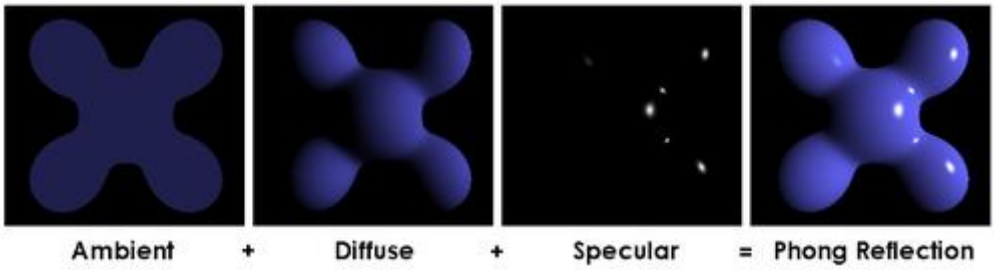
Лабораторная работа Модель Фонга

Создать сферу произвольного цвета, освещенную по модели Фонга.

Выполнение

В данной лабораторной работе, было необходимо реализовать освещение по модели Фонга.

Модель Фонга - модель расчёта освещения трёхмерных объектов, в том числе полигональных моделей и примитивов, а также метод интерполяции освещения по всему объекту.



Модель Фонга рассматривает освещение исходя из трех составляющих, это: основное затенение (Ambient), lиффузное рассеивание (Diffuse), блик (Specular);

Способ расчета данной модели освещения:

$$I = K_a I_a + K_d(\vec{n}, \vec{l}) + K_s(\vec{n}, \vec{h})^p$$

где

$\vec{n}$ — вектор нормали к поверхности в точке

$\vec{l}$ — падающий луч (направление на источник света)

$\vec{h}$ — отраженный луч (направление идеально отраженного от поверхности луча)

$\vec{h} = 2(\vec{l} * \vec{n})\vec{n} - \vec{l}$

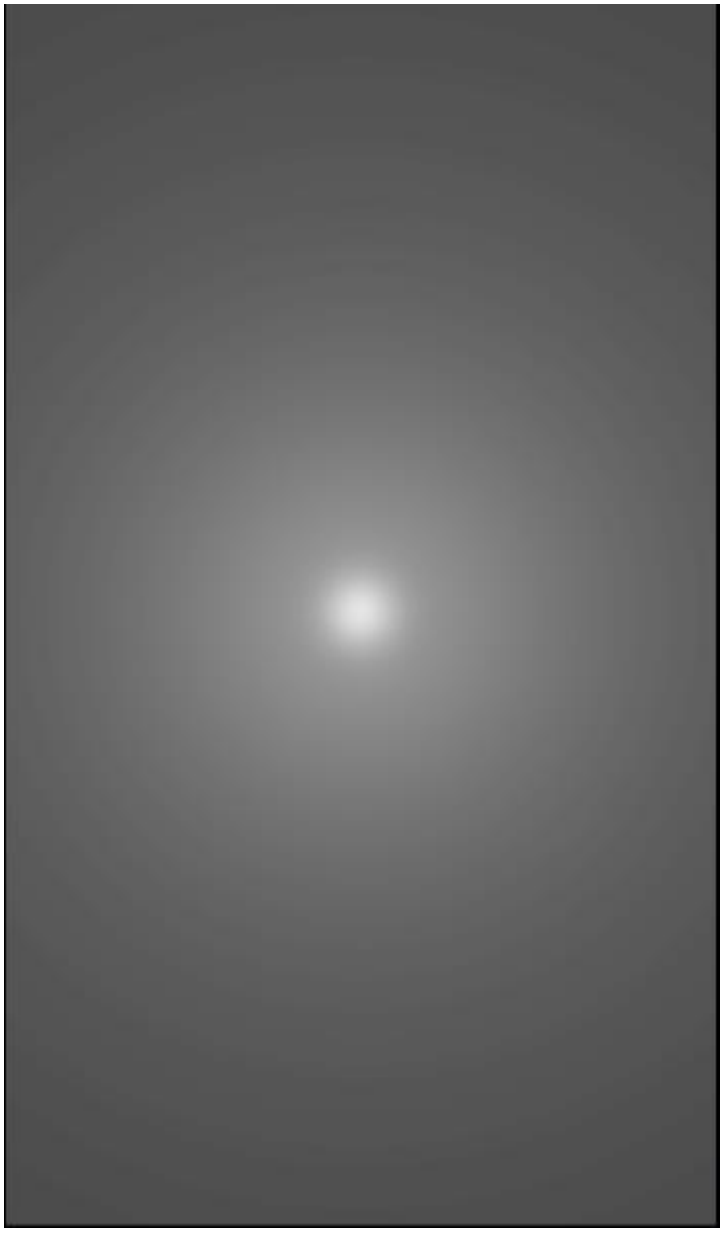
K_a — коэффициент фонового освещения

K_s — коэффициент зеркального освещения

K_d — коэффициент диффузного освещения

Таким образом нам необходимо высчитать интенсивность освещения для каждой точки на поверхности сферы, а затем отобразить это через фрагментный шейдер и получить результат.

Результат:



Листинг проекта

MainActivity.java

```
package com.example.lab4;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    private MySurfaceView mGLSurfaceView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mGLSurfaceView = new MySurfaceView(this);
        setContentView(mGLSurfaceView);
    }
}
```

MySyrfaceView.java

```
package com.example.lab4;

import android.content.Context;
import android.opengl.GLSurfaceView;

public class MySurfaceView extends GLSurfaceView {
    private MyRenderer renderer;

    public MySurfaceView(Context context) {
        super(context);
        setEGLContextClientVersion(2);
        renderer = new MyRenderer(context);
        setRenderer(renderer);
        setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
    }

}
```

MyRenderer.java

```
package com.example.lab4;

import android.content.Context;
import android.opengl.GLES20;
import android.opengl.GLSurfaceView;
import android.opengl.Matrix;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

public class MyRenderer implements GLSurfaceView.Renderer {

    private Context context;
    private float xCamera, yCamera, zCamera;
```

```
private float xLightPosition, yLightPosition, zLightPosition;
private float[] modelMatrix;
private float[] viewMatrix;
private float[] modelViewMatrix;
private float[] projectionMatrix;
private float[] modelViewProjectionMatrix;
private FloatBuffer vertexBuffer;
private FloatBuffer normalBuffer;
private FloatBuffer colorBuffer;
private Shader mShader;

public MyRenderer(Context context) {
   this.context = context;
   xLightPosition = 0f;
   yLightPosition = 0.5f;
   zLightPosition = 0f;
   modelMatrix = new float[16];
   viewMatrix = new float[16];
   modelViewMatrix = new float[16];
   projectionMatrix = new float[16];
   modelViewProjectionMatrix = new float[16];

   Matrix.setIdentityM(modelMatrix, 0);
   xCamera = 0f;
   yCamera = 8f;
   zCamera = 0.01f;

   Matrix.setLookAtM(
         viewMatrix, 0, xCamera, yCamera, zCamera, 0, 0, 0, 0, 2, 0);
   Matrix.multiplyMM(modelViewMatrix, 0, viewMatrix, 0, modelMatrix, 0);

   float x1 = -2;
   float y1 = 0;
   float z1 = -2;

   float x2 = -2;
   float y2 = 0;
   float z2 = 2;

   float x3 = 2;
   float y3 = 0;
   float z3 = -2;

   float x4 = 2;
   float y4 = 0;
   float z4 = 2;
   float[] vertexArray = {x1,y1,z1, x2,y2,z2, x3,y3,z3, x4,y4,z4};

   ByteBuffer bvertex = ByteBuffer.allocateDirect(vertexArray.length * 4);
   bvertex.order(ByteOrder.nativeOrder());
   vertexBuffer = bvertex.asFloatBuffer();
   vertexBuffer.position(0);

   vertexBuffer.put(vertexArray);
   vertexBuffer.position(0);

   float nx = 0;
   float ny = 1;
   float nz = 0;
   float[] normalArray = {nx, ny, nz,   nx, ny, nz,   nx, ny, nz,   nx, ny, nz};
```

```
    ByteBuffer bnormal = ByteBuffer.allocateDirect(normalArray.length*4);
    bnormal.order(ByteOrder.nativeOrder());
    normalBuffer = bnormal.asFloatBuffer();
    normalBuffer.position(0);
    normalBuffer.put(normalArray);
    normalBuffer.position(0);

    float[] colorArray = {
            0f, 0f, 0f, 0,
            0f, 0f, 0f, 0,
            0f, 0, 0f, 0,
            0f, 0f, 0f, 0
    };
    ByteBuffer bcolor = ByteBuffer.allocateDirect(colorArray.length * 4);
    bcolor.order(ByteOrder.nativeOrder());
    colorBuffer = bcolor.asFloatBuffer();
    colorBuffer.position(0);
    colorBuffer.put(colorArray);
    colorBuffer.position(0);
}

@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    GLES20.glEnable(GLES20.GL_DEPTH_TEST);
    GLES20.glEnable(GLES20.GL_CULL_FACE);
    GLES20.glHint(GLES20.GL_GENERATE_MIPMAP_HINT, GLES20.GL_NICEST);

    String vertexShaderCode=
            "uniform mat4 u_modelViewProjectionMatrix;"+
                    "attribute vec3 a_vertex;"+
                    "attribute vec3 a_normal;"+
                    "attribute vec4 a_color;"+
                    "varying vec3 v_vertex;"+
                    "varying vec3 v_normal;"+
                    "varying vec4 v_color;"+
                    "void main() {"+
                    "v_vertex=a_vertex;"+
                    "vec3 n_normal=normalize(a_normal);"+
                    "v_normal=n_normal;"+
                    "v_color=a_color;"+
                    "gl_Position = u_modelViewProjectionMatrix * vec4(a_vertex,1.0);"+
                    "}";
    String fragmentShaderCode=
            "precision mediump float;"+
                    "uniform vec3 u_camera;"+
                    "uniform vec3 u_lightPosition;"+
                    "varying vec3 v_vertex;"+
                    "varying vec3 v_normal;"+
                    "varying vec4 v_color;"+
                    "void main() {"+
                    "vec3 n_normal = normalize(v_normal);"+
                    "vec3 lightvector = normalize(u_lightPosition - v_vertex);"+
                    "vec3 lookvector = normalize(u_camera - v_vertex);"+
                    "float ambient = 0.4;"+
                    "float k_diffuse = 0.8;"+
                    "float k_specular = 0.6;"+
                    "float diffuse = k_diffuse * max(dot(n_normal, lightvector), 0.0);"+
                    "vec3 reflectvector = reflect(-lightvector, n_normal);"+
                    "float specular = k_specular * pow(max(dot(lookvector, reflectvector), 0.0), 40.0);"+
                    "vec4 one = vec4(1.0,1.0,1.0,1.0);"+
```

```
                "vec4 lightColor = (ambient + diffuse + specular) * one;"+
                "gl_FragColor = mix(lightColor, v_color, 0.5);"+
                "}";

        mShader = new Shader(vertexShaderCode, fragmentShaderCode);
        mShader.linkVertexBuffer(vertexBuffer);
        mShader.linkNormalBuffer(normalBuffer);
        mShader.linkColorBuffer(colorBuffer);
    }

    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        GLES20.glViewport(0, 0, width, height);
        float ratio = (float) width / height;
        float k = 0.055f;
        float left = -k * ratio;
        float right = k * ratio;
        float bottom = -k;
        float near = 0.25f;
        float far = 9.0f;
        Matrix.frustumM(projectionMatrix, 0, left, right, bottom, k, near, far);
        Matrix.multiplyMM(modelViewProjectionMatrix,
                0, projectionMatrix, 0, modelViewMatrix, 0);
    }

    @Override
    public void onDrawFrame(GL10 gl) {
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT | GLES20.GL_DEPTH_BUFFER_BIT);
        mShader.linkModelViewProjectionMatrix(modelViewProjectionMatrix);
        mShader.linkCamera(xCamera, yCamera, zCamera);
        mShader.linkLightSource(xLightPosition, yLightPosition, zLightPosition);
        mShader.useProgram();
        GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4);
    }
}
```

Shader.java

```
package com.example.lab4;

import android.opengl.GLES20;

import java.nio.FloatBuffer;

public class Shader {
    private int program_Handle;

    public Shader(String vertexShaderCode, String fragmentShaderCode) {
        createProgram(vertexShaderCode, fragmentShaderCode);
    }

    private void createProgram(String vertexShaderCode, String fragmentShaderCode) {
        int vertexShader_Handle =
                GLES20.glCreateShader(GLES20.GL_VERTEX_SHADER);
        GLES20.glShaderSource(vertexShader_Handle, vertexShaderCode);
        GLES20.glCompileShader(vertexShader_Handle);
        int fragmentShader_Handle =
                GLES20.glCreateShader(GLES20.GL_FRAGMENT_SHADER);
        GLES20.glShaderSource(fragmentShader_Handle, fragmentShaderCode);
        GLES20.glCompileShader(fragmentShader_Handle);
```

```
        program_Handle = GLES20.glCreateProgram();
        GLES20.glAttachShader(program_Handle, vertexShader_Handle);
        GLES20.glAttachShader(program_Handle, fragmentShader_Handle);
        GLES20.glLinkProgram(program_Handle);
    }

    public void linkVertexBuffer(FloatBuffer vertexBuffer) {
        GLES20.glUseProgram(program_Handle);
        int a_vertex_Handle = GLES20.glGetAttribLocation(program_Handle, "a_vertex");
        GLES20.glEnableVertexAttribArray(a_vertex_Handle);
        GLES20.glVertexAttribPointer(
                a_vertex_Handle, 3, GLES20.GL_FLOAT, false, 0,vertexBuffer);
    }

    public void linkNormalBuffer(FloatBuffer normalBuffer) {
        GLES20.glUseProgram(program_Handle);
        int a_normal_Handle = GLES20.glGetAttribLocation(program_Handle, "a_normal");
        GLES20.glEnableVertexAttribArray(a_normal_Handle);
        GLES20.glVertexAttribPointer(
                a_normal_Handle, 3, GLES20.GL_FLOAT, false, 0,normalBuffer);
    }

    public void linkColorBuffer(FloatBuffer colorBuffer) {
        GLES20.glUseProgram(program_Handle);
        int a_color_Handle = GLES20.glGetAttribLocation(program_Handle, "a_color");
        GLES20.glEnableVertexAttribArray(a_color_Handle);
        GLES20.glVertexAttribPointer(
                a_color_Handle, 4, GLES20.GL_FLOAT, false, 0, colorBuffer);
    }

    public void linkModelViewProjectionMatrix(float [] modelViewProjectionMatrix) {
        GLES20.glUseProgram(program_Handle);
        int u_modelViewProjectionMatrix_Handle =
                GLES20.glGetUniformLocation(program_Handle, "u_modelViewProjectionMatrix");
        GLES20.glUniformMatrix4fv(
                u_modelViewProjectionMatrix_Handle, 1, false, modelViewProjectionMatrix, 0);
    }

    public void linkCamera (float xCamera, float yCamera, float zCamera) {
        GLES20.glUseProgram(program_Handle);
        int u_camera_Handle=GLES20.glGetUniformLocation(program_Handle, "u_camera");
        GLES20.glUniform3f(u_camera_Handle, xCamera, yCamera, zCamera);
    }

    public void linkLightSource (float xLightPosition, float yLightPosition, float zLightPosition) {
        GLES20.glUseProgram(program_Handle);
        int u_lightPosition_Handle=GLES20.glGetUniformLocation(program_Handle, "u_lightPosition");
        GLES20.glUniform3f(u_lightPosition_Handle, xLightPosition, yLightPosition, zLightPosition);
    }

    public void useProgram(){
        GLES20.glUseProgram(program_Handle);
    }
}
```