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 №2
"примитивы OpenGL ES 1"
по дисциплине
«Программирование мобильных устройств»

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

Проверила:
Павлова У.В.

Новосибирск 2021

Оглавление

Задание

Необходимо создать классы прорисовки квадрата, куба, сферы.

Выполнение

Для выполнения данной лабораторной работы, мною были реализованы класс рендера и 3 класса с описанием фигуры.

Класс рендера имплементирует методы класса GLSurfaceView.Renderer.

Это 3 метода:

void onSurfaceCreated – метод вызываемый при создании объекта имплементирующего методы класса GLSurfaceView.Renderer.

void onSurfaceChanged – метод вызываемый при изменении поверхности во время выполнения приложения

void onDrawFrame – метод вызываемый перед отрисовкой кадра, здесь в основном располагается вся реализация для отображения какого-либо объекта.

Я реализовал 3 классы с рендером фигур:

Square.java – Класс ответственный за отрисовку квадрата.

Cube.java – Класс реализующий отрисовку куба.

Sphere.java – Класс реализующий отрисовку сферы.

Листинг проекта

MainActivity.java

```
package com.example.lab2;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.WindowManager;

public class MainActivity extends Activity {
    private GLSurfaceView g;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        super.onCreate(savedInstanceState);

        g = new GLSurfaceView(this);
        g.setEGLConfigChooser(8,8,8,8,16,1);
        g.setRenderer(new MyRenderer(this));
        g.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
        setContentView(g);
    }

    @Override
    protected void onPause() {
        super.onPause();
        g.onPause();
    }
    @Override
    protected void onResume() {
        super.onResume();
        g.onResume();
    }
}
```

MyRender.java

```
package com.example.lab2;

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

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


public class MyRenderer implements GLSurfaceView.Renderer {
    Context context;

    private Square mSquare;
    private Cube mCube;
    private Sphere mSphere;

    private static float angleCube = 0;
    private static float speedCube = -1.5f;
    private float mTransY = 0f;
```

```
    private float mAngle = 0;

    public MyRenderer(Context context){
        this.context = context;
        mSquare = new Square();
        mCube = new Cube();
        mSphere = new Sphere(5);
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        gl.glClearDepthf(1.0f);
        gl.glEnable(GL10.GL_DEPTH_TEST);
        gl.glDepthFunc(GL10.GL_LEQUAL);
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
        gl.glShadeModel(GL10.GL_SMOOTH);
        gl.glDisable(GL10.GL_DITHER);
    }

    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        if (height == 0) height = 1;
        float aspect = (float)width / height;
        gl.glViewport(0, 0, width, height);
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        GLU.gluPerspective(gl, 45, aspect, 0.1f, 100.f);
        gl.glMatrixMode(GL10.GL_MODELVIEW); // Select model-view matrix
        gl.glLoadIdentity();
    }

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        gl.glLoadIdentity();
        gl.glTranslatef(0.0f, 0.0f, -5.0f);
        gl.glRotatef(mAngle, 0, 0, -1);
        mSquare.draw(gl);

        gl.glLoadIdentity();
        gl.glTranslatef(0.0f, 0.0f, -4.0f);
        gl.glScalef(0.2f, 0.2f, 0.2f);
        gl.glRotatef(angleCube, 1.0f, 1.0f, 1.0f);
        mCube.draw(gl);

        gl.glLoadIdentity();
        gl.glTranslatef((float)Math.cos(mTransY), (float)Math.sin(mTransY) + 0.0f, -4.5f);
        gl.glScalef(0.05f, 0.05f, 0.05f);
        gl.glRotatef(mAngle, -1, -1, 0);
        mSphere.draw(gl);

        angleCube += speedCube;
        mTransY+=.05f;
        mAngle+=1.8;
    }

}
```

MyRender.java

```
package com.example.lab2;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

public class Square {
   private FloatBuffer vertexBuffer;  // Buffer for vertex-array
   private float[] vertices = {  // Vertices for the Square
         -1.0f, -1.0f,  0.0f,  // 0. left-bottom
         1.0f, -1.0f,  0.0f,  // 1. right-bottom
         -1.0f,  1.0f,  0.0f,  // 2. left-top
         1.0f,  1.0f,  0.0f   // 3. right-top
   };
   public Square() {
      ByteBuffer byteBuf = ByteBuffer.allocateDirect(vertices.length * 4);
      byteBuf.order(ByteOrder.nativeOrder());
      vertexBuffer = byteBuf.asFloatBuffer();
      vertexBuffer.put(vertices);
      vertexBuffer.position(0);
   }

   public void draw(GL10 gl) {
      gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
      gl.glEnable(GL10.GL_BLEND);
      gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
      gl.glColor4f(0.9f, 0.7f, 0.7f, 0.9f);
      gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, vertices.length / 3);
      gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
      gl.glDisable(GL10.GL_BLEND);
   }
}
```

Cube.java

```
package com.example.lab2;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

public class Cube {
   private FloatBuffer vertexBuffer;
   private int numFaces = 6;

   private float[][] colors = {  // Colors of the 6 faces
         {1.0f, 0.5f, 0.0f, 1.0f},  // 0. orange
         {1.0f, 0.0f, 1.0f, 1.0f},  // 1. violet
         {0.0f, 1.0f, 0.0f, 1.0f},  // 2. green
         {0.0f, 0.0f, 1.0f, 1.0f},  // 3. blue
         {1.0f, 0.0f, 0.0f, 1.0f},  // 4. red
         {1.0f, 1.0f, 0.0f, 1.0f}   // 5. yellow
   };

   private float[] vertices = {  // Vertices of the 6 faces
         // FRONT
```

```
            -1.0f, -1.0f,  1.0f,  // 0. left-bottom-front
            1.0f, -1.0f,  1.0f,  // 1. right-bottom-front
            -1.0f,  1.0f,  1.0f,  // 2. left-top-front
            1.0f,  1.0f,  1.0f,  // 3. right-top-front
            // BACK
            1.0f, -1.0f, -1.0f,  // 6. right-bottom-back
            -1.0f, -1.0f, -1.0f,  // 4. left-bottom-back
            1.0f,  1.0f, -1.0f,  // 7. right-top-back
            -1.0f,  1.0f, -1.0f,  // 5. left-top-back
            // LEFT
            -1.0f, -1.0f, -1.0f,  // 4. left-bottom-back
            -1.0f, -1.0f,  1.0f,  // 0. left-bottom-front
            -1.0f,  1.0f, -1.0f,  // 5. left-top-back
            -1.0f,  1.0f,  1.0f,  // 2. left-top-front
            // RIGHT
            1.0f, -1.0f,  1.0f,  // 1. right-bottom-front
            1.0f, -1.0f, -1.0f,  // 6. right-bottom-back
            1.0f,  1.0f,  1.0f,  // 3. right-top-front
            1.0f,  1.0f, -1.0f,  // 7. right-top-back
            // TOP
            -1.0f,  1.0f,  1.0f,  // 2. left-top-front
            1.0f,  1.0f,  1.0f,  // 3. right-top-front
            -1.0f,  1.0f, -1.0f,  // 5. left-top-back
            1.0f,  1.0f, -1.0f,  // 7. right-top-back
            // BOTTOM
            -1.0f, -1.0f, -1.0f,  // 4. left-bottom-back
            1.0f, -1.0f, -1.0f,  // 6. right-bottom-back
            -1.0f, -1.0f,  1.0f,  // 0. left-bottom-front
            1.0f, -1.0f,  1.0f   // 1. right-bottom-front
    };

    public Cube() {
        ByteBuffer byteBuf = ByteBuffer.allocateDirect(vertices.length * 4);
        byteBuf.order(ByteOrder.nativeOrder());
        vertexBuffer = byteBuf.asFloatBuffer();
        vertexBuffer.put(vertices);
        vertexBuffer.position(0);
    }

    public void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CCW);
        gl.glEnable(GL10.GL_CULL_FACE);
        gl.glCullFace(GL10.GL_BACK);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);

        for (int face = 0; face < numFaces; face++) {
            gl.glColor4f(colors[face][0], colors[face][1], colors[face][2], colors[face][3]);
            gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, face * 4, 4);
        }
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisable(GL10.GL_CULL_FACE);
    }

}
```

Cube.java

```
package com.example.lab2;

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

import javax.microedition.khronos.opengles.GL10;

public class Sphere {
   public FloatBuffer mVertexBuffer;
   public FloatBuffer textureBuffer;
   public int n = 0, sz = 0;

   private float[][] colors = {  // Colors of the 6 faces
         {1.0f, 0.0f, 0.0f, 1.0f},  // 0. orange
         {0.95f, 0.5f, 0.5f, 1.0f},  // 1. violet
         {1.0f, 1.0f, 1.0f, 1.0f},  // 1. violet
   };

   public Sphere(float R) {
      int dtheta = 15, dphi = 15;
      float DTOR = (float) (Math.PI / 180.0f);

      ByteBuffer byteBuf = ByteBuffer.allocateDirect(5000 * 3 * 4);
      byteBuf.order(ByteOrder.nativeOrder());
      mVertexBuffer = byteBuf.asFloatBuffer();
      byteBuf = ByteBuffer.allocateDirect(5000 * 2 * 4);
      byteBuf.order(ByteOrder.nativeOrder());
      textureBuffer = byteBuf.asFloatBuffer();

      for (int theta = -90; theta <= 90 - dtheta; theta += dtheta) {
         for (int phi = 0; phi <= 360 - dphi; phi += dphi) {
            sz++;
            mVertexBuffer.put((float) (Math.cos(theta * DTOR) * Math.cos(phi * DTOR)) * R);
            mVertexBuffer.put((float) (Math.cos(theta * DTOR) * Math.sin(phi * DTOR)) * R);
            mVertexBuffer.put((float) (Math.sin(theta * DTOR)) * R);

            mVertexBuffer.put((float) (Math.cos((theta + dtheta) * DTOR) * Math.cos(phi * DTOR)) * R);
            mVertexBuffer.put((float) (Math.cos((theta + dtheta) * DTOR) * Math.sin(phi * DTOR)) * R);
            mVertexBuffer.put((float) (Math.sin((theta + dtheta) * DTOR)) * R);

            mVertexBuffer.put((float) (Math.cos((theta + dtheta) * DTOR) * Math.cos((phi + dphi) * DTOR)) * R);
            mVertexBuffer.put((float) (Math.cos((theta + dtheta) * DTOR) * Math.sin((phi + dphi) * DTOR)) * R);
            mVertexBuffer.put((float) (Math.sin((theta + dtheta) * DTOR)) * R);

            mVertexBuffer.put((float) (Math.cos(theta * DTOR) * Math.cos((phi + dphi) * DTOR)) * R);
            mVertexBuffer.put((float) (Math.cos(theta * DTOR) * Math.sin((phi + dphi) * DTOR)) * R);
            mVertexBuffer.put((float) (Math.sin(theta * DTOR)) * R);
            n += 4;
         }
      }
      mVertexBuffer.position(0);
      textureBuffer.position(0);
   }

   public void draw(GL10 gl) {
      gl.glFrontFace(GL10.GL_CCW);
      gl.glEnable(GL10.GL_CULL_FACE);
      gl.glCullFace(GL10.GL_BACK);
```

```
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mVertexBuffer);
        for (int i = 0; i < n; i += 4) {
            gl.glColor4f(colors[i % 3][0], colors[i % 3][1], colors[i % 3][2], colors[i % 3][3]);
            gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, i, 4);
        }
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisable(GL10.GL_CULL_FACE);
    }
}
```