Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Лабораторная работа по теме: «Матрицы модели-вида OpenGL ES 1»

Выполнили:

студентки 4 курса

ИВТ, гр. ИП-712

Гервас А.В.

Онищенко А.В.

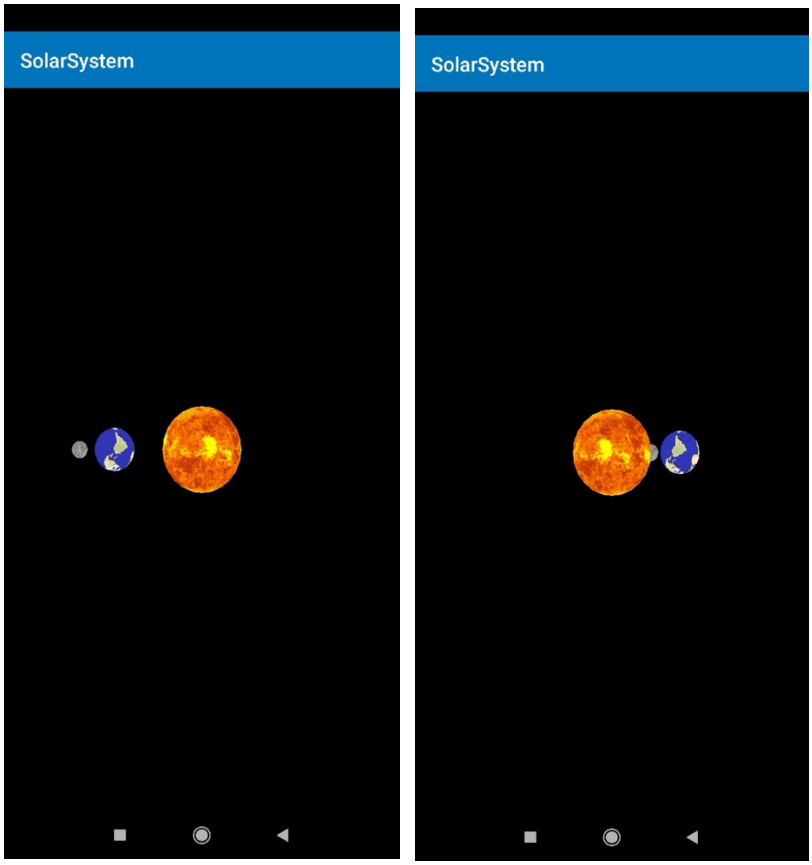
Новосибирск 2020

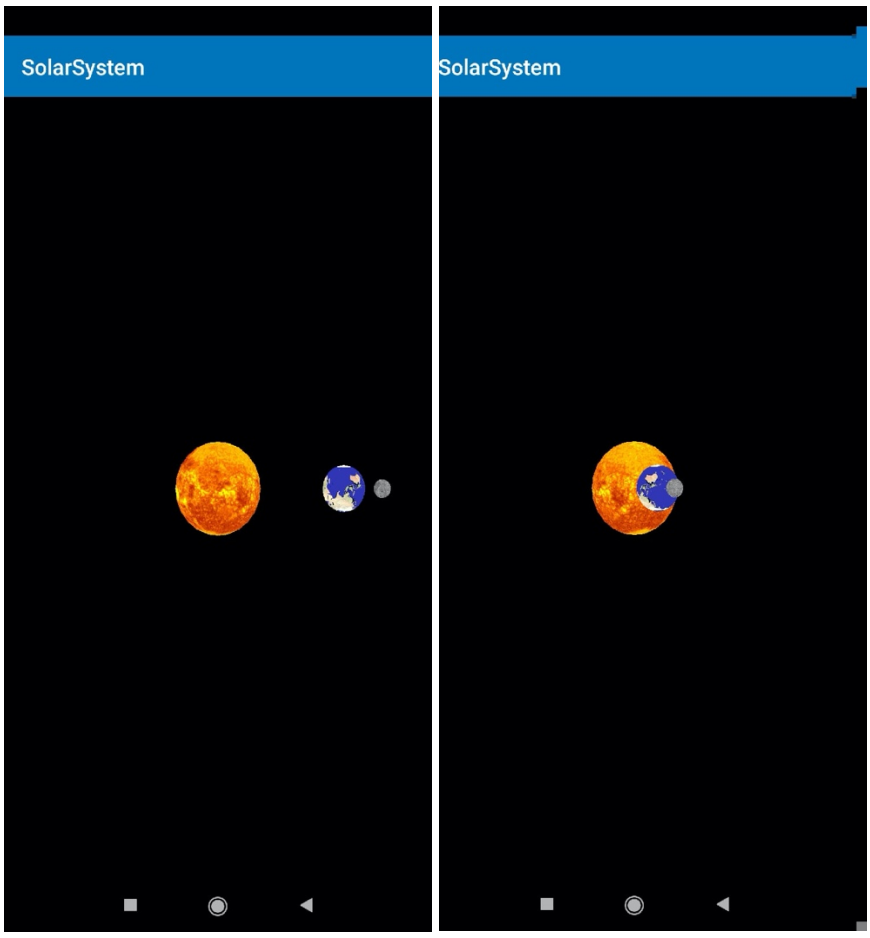
Оглавление

Задание

Необходимо создать модель Солнце и вращающиеся Земля и Луна. Текстуры взять из интернета.

Скриншоты





Листинг кода

Приложение написано на языке Java.

MainActivity.java

```
package ru.sibsutis.solarsystem;

import androidx.appcompat.app.AppCompatActivity;

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

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

```
        GLSurfaceView view = new GLSurfaceView(this);
        view.setRenderer(new SolarSystemRenderer(this));

view.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
        setContentView(view);
    }
}
```

SolarSystemRenderer.java

```
package ru.sibsutis.solarsystem;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLSurfaceView;
import android.opengl.GLUtils;

import java.io.InputStream;

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

import static java.lang.Math.cos;
import static java.lang.Math.sin;

public class SolarSystemRenderer implements GLSurfaceView.Renderer {

    static public int[] texture_name = {R.drawable.sun, R.drawable.arth_light,
R.drawable.moon};
    static public int[] textures = new int[texture_name.length];

    Context c;
    private Planet Sun = new Planet(2f);
    private Planet Earth = new Planet(1f);
    private Planet Moon = new Planet(0.4f);

    private float p = 0.0f;
    private float angle = 40.0f;

    public SolarSystemRenderer(Context context) {
        c = context;
    }

    private void loadGLTexture(GL10 gl) {
```

```
        gl.glGenTextures(3, textures, 0);
        for (int i = 0; i < texture_name.length; ++i) {
            gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[i]); // привязка
текстуры
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
            InputStream is = c.getResources().openRawResource(texture_name[i]);
            Bitmap bitmap = BitmapFactory.decodeStream(is);  //создание объекта
bitmap
            GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);  // загрузка
текстуры
            bitmap.recycle();  // используется для освобождения памяти
        }
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0, 1.0f);
        gl.glClearDepthf(1);
        gl.glEnable(GL10.GL_DEPTH_TEST); // разрешение глубины
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();  // Матрица идентичности сбрасывает матрицу
обратно в состояние по умолчанию
        gl.glOrthof(-10, 10, -10, 10, -10, 10); // применяет орфографическую
проекцию
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glScalef(1, 0.6f, 1);
        loadGLTexture(gl);
    }

    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {

    }

    @Override
    public void onDrawFrame(GL10 gl) {

        float RotationOffset, RotationSpeed;
        p = (p == 360) ? 0 : p + 2;
        angle = (angle == 360) ? 0 : angle + 0.15f;
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
GL10.GL_DEPTH_BUFFER_BIT);
        gl.glEnable(GL10.GL_TEXTURE_2D);  // подключение текстур
```

```
        gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);  // привязка
        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY); //
устанавливают состояние клиентской части

        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, Sun.textureBuffer); //
загрузка текстуры
        gl.glPushMatrix();  // копирует верхнюю матрицу и помещает ее в стек
        gl.glRotatef(90, 1, 0, 0);
        gl.glRotatef(p, 0, 0, 0.1f);
        gl.glColor4f(1, 1, 0, 1);
        Sun.onDrawFrame(gl);
        gl.glPopMatrix();  // извлекает верхнюю матрицу из стека
        RotationOffset = 6.0f;  // смещение вращения
        RotationSpeed = 0.05f;  // скорость вращения
        gl.glPushMatrix();  // копирует верхнюю матрицу и помещает ее в стек

        gl.glTranslatef(RotationOffset * (float) (cos(angle * RotationSpeed)), 0f,
                RotationOffset * (float) (sin(angle * RotationSpeed)));

        gl.glRotatef(90, 1, 0, 0);
        gl.glRotatef(p, 0, 0, 1);
        gl.glPushMatrix();

        gl.glEnable(GL10.GL_TEXTURE_2D);  // подключение текстуры
        gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[1]);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, Earth.textureBuffer);
        gl.glColor4f(1, 1, 1, 1);
        Earth.onDrawFrame(gl);
        RotationOffset = 1.5f;
        RotationSpeed = 0.2f;

        gl.glTranslatef(RotationOffset * (float) (cos(0 * RotationSpeed)) + 0.3f, 0f,
                RotationOffset * (float) (sin(0 * RotationSpeed)));

        gl.glEnable(GL10.GL_TEXTURE_2D);  // подключение текстуры
        gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[2]);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, Moon.textureBuffer);
        gl.glColor4f(1, 1, 1, 1);
        Moon.onDrawFrame(gl);
        gl.glRotatef(p, 0, 0, 1);

        gl.glPopMatrix();
        gl.glPopMatrix();
        gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        gl.glDisable(GL10.GL_TEXTURE_2D); // отключ
```

```
    }
}
```

Planet.java

```
package ru.sibsutis.solarsystem;

import android.opengl.GLSurfaceView;

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 Planet implements GLSurfaceView.Renderer {

    private int n;
    private FloatBuffer mVertexBuffer;
    public FloatBuffer textureBuffer;

    public Planet(float R) {
        n = 0;
        int dtheta = 15, dphi = 15;
        float DTOR = (float) (Math.PI / 180.0f);
        ByteBuffer byteBuf = ByteBuffer.allocateDirect(5000 * 3 * 4);  // выделение
памяти из основной кучи JVM
        byteBuf.order(ByteOrder.nativeOrder()); // извлекает собственный порядок
байтов базовой платформы
        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) {
                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);

                double cosM = Math.cos((theta + dtheta) * DTOR);
```

```
            mVertexBuffer.put((float) (cosM * Math.cos(phi * DTOR)) * R);
            mVertexBuffer.put((float) (cosM * Math.sin(phi * DTOR)) * R);

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

            textureBuffer.put((float) (phi / 360.0f));
            textureBuffer.put((float) ((90 + theta) / 180.0f));
            textureBuffer.put((float) (phi / 360.0f));
            textureBuffer.put((float) ((90 + theta + dtheta) / 180.0f));
            textureBuffer.put((float) ((phi + dphi) / 360.0f));
            textureBuffer.put((float) ((90 + theta + dtheta) / 180.0f));
            textureBuffer.put((float) ((phi + dphi) / 360.0f));
            textureBuffer.put((float) ((90 + theta) / 180.0f));
          }
        }

        mVertexBuffer.position(0);
        textureBuffer.position(0);
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {

    }

    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {

    }

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glEnable(GL10.GL_BLEND); // разрешение на наложение цветов
```

```
        gl.glBlendFunc(GL10.GL_SRC_ALPHA,
GL10.GL_ONE_MINUS_SRC_ALPHA); //алгоритм смешения с масштабными
коэффициентами
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY); // разрешить массив
вершин
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mVertexBuffer); // определяет
массив данных вершин, хранит в памяти видеокарты
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, textureBuffer);   // с
текстурами

        for (int i = 0; i < n; i += 4)
           gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, i, 4);

        // рендер примитивов из массива
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisable(GL10.GL_BLEND);
    }
}
```