Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Курсовая работа по дисциплине

Программирование мобильных устройств

Выполнил: студент 4 курса

группы ИП-112

Притула А.Д.

старший преподаватель
кафедры ПМиК

Осипова У.В.

Новосибирск, 2024

Содержание

Постановка задачи

Создайте программу, в которой нарисован стол на OpenGL ES 2.0.

На столе лежат различные фрукты/овощи, стакан с напитком.
Имеется освещение по модели Фонга.

Описание основных блоков программы

MainActivity.java – основной класс программы.

Происходит создание экземпляра GLSurfaceView, который будет использоваться для отображения графики, установка OpenGL 2.0.

Создаётся экземпляр класса MyRenderer, где реализован интерфейс GLSurfaceView.Renderer, который отвечает за рендеринг графики.

Устанавливается содержимое Activity, чтобы GLSurfaceView стал основным элементом интерфейса.

MyRenderer.java

onSurfaceCreated – создаются объекты TableObject для каждого 3D-объекта, который будет отрисован, далее шейдеры загружаются и компилируются в ShadersWork.

onSurfaceChanged – устанавливается область просмотра, устанавливаются матрицы проекции для камеры и света.

onDrawFrame – получаем дескриптор текущей программы шейдеров, устанавливаем матрицу вида, которая определяет положение и ориентацию камеры в пространстве, получение расположений переменных в программе шейдеров, вычисляется текущая позиция света и инициируется вызов функции draw_picture для отрисовки объекта.

draw_picture - очищаем буферы цвета и глубины (для удаления предыдущего содержимого на экране). Вычисление матриц для использования в шейдерах для правильного расчёта вершин, освещения и теней. Отправляем все объекты рендериться с помощью .render

ShadersWork.java

createProgram – вызывает loadShader для загрузки и компиляции шейдеров. Создаём программу, и привязываем шейдеры к программе.

loadShader - создаём шейдер и компилируем его, возвращаем идентификатор шейдера.

getProgramm – геттер для дескриптора программы.

TableObject.java

Читает объект из файла, включаем атрибуты вершин, нормалей и цветов, отрисовываем объекты.

Демонстрация работы

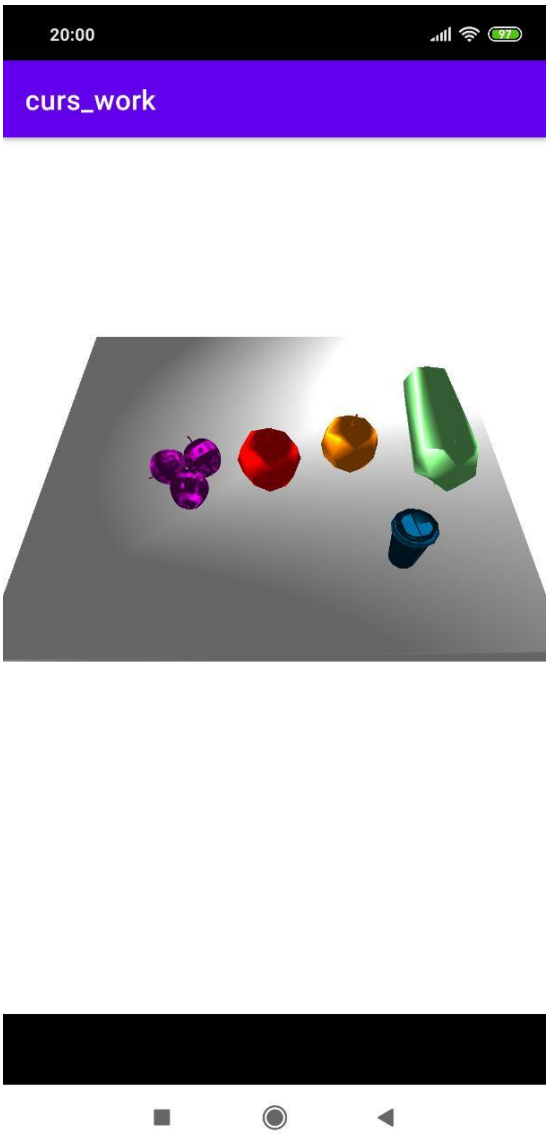

Рисунок 1 – Демонстрация работы программы

Исходный код

MainActivity.java

```
package com.example.curs_work;


import androidx.appcompat.app.AppCompatActivity;


import android.opengl.GLSurfaceView;

import android.os.Bundle;


public class MainActivity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        GLSurfaceView mGLSurfaceView = new GLSurfaceView(this);


        mGLSurfaceView.setEGLContextClientVersion(2); //устанавливаем версию OPENGL2


        MyRenderer renderer = new MyRenderer(this); //создаем свой рендерер

        mGLSurfaceView.setRenderer(renderer);


        setContentView(mGLSurfaceView);

    }

}
```

TableObjects.java

```
package com.example.curs_work;


import java.util.List;

import java.util.Scanner;

import java.nio.ByteOrder;

import java.io.IOException;

import java.nio.ByteBuffer;

import java.nio.FloatBuffer;
```

```
import java.nio.ShortBuffer;
import java.util.ArrayList;

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

class TableObjects {
    private FloatBuffer normalBuffer; // буфер нормали
    private FloatBuffer verticesBuffer; // буфер вершин
    private final FloatBuffer colorBuffer; // буфер цвета
    private ShortBuffer facesVertexBuffer; // буфер "передних" вершин
    private ShortBuffer facesNormalBuffer; // буфер "передних" координат нормали
    private final List < String > facesList; // список "передних" координат из объектов

    TableObjects(Context c, float[] color, String ObjName) {
        // Считываем координаты объектов в списки
        facesList = new ArrayList < > ();
        List < String > verticesList = new ArrayList < > ();
        List < String > normalList = new ArrayList < > ();
        try {
            Scanner scanner = new Scanner(c.getAssets().open(ObjName));
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                if (line.startsWith("v ")) {
                    verticesList.add(line); // добавляем координаты вершин
                } else if (line.startsWith("f ")) {
                    facesList.add(line); // добавляем координаты вершин
                } else if (line.startsWith("vn ")) {
                    normalList.add(line); // добавляем координаты нормалей
                }
            }

            // Заполняем буферы считанными координатами
```

```
ByteBuffer buffer1 = ByteBuffer.allocateDirect(verticesList.size() * 3 * 4);
buffer1.order(ByteOrder.nativeOrder());
verticesBuffer = buffer1.asFloatBuffer();

ByteBuffer buffer2 = ByteBuffer.allocateDirect(normalList.size() * 3 * 4);
buffer2.order(ByteOrder.nativeOrder());
normalBuffer = buffer2.asFloatBuffer();

ByteBuffer buffer3 = ByteBuffer.allocateDirect(facesList.size() * 3 * 2);
buffer3.order(ByteOrder.nativeOrder());
facesVertexBuffer = buffer3.asShortBuffer();

ByteBuffer buffer4 = ByteBuffer.allocateDirect(facesList.size() * 3 * 2);
buffer4.order(ByteOrder.nativeOrder());
facesNormalBuffer = buffer4.asShortBuffer();

// Парсим координаты в float и размечаем данные буферов последовательно по координатам
for (String vertex: verticesList) {
    String[] coords = vertex.split(" ");
    float x = Float.parseFloat(coords[1]);
    float y = Float.parseFloat(coords[2]);
    float z = Float.parseFloat(coords[3]);
    verticesBuffer.put(x);
    verticesBuffer.put(y);
    verticesBuffer.put(z);
}
verticesBuffer.position(0);

for (String vertex: normalList) {
    String[] coords = vertex.split(" ");
    float x = Float.parseFloat(coords[1]);
    float y = Float.parseFloat(coords[2]);
    float z = Float.parseFloat(coords[3]);
```

```
            normalBuffer.put(x);
            normalBuffer.put(y);
            normalBuffer.put(z);
        }
        normalBuffer.position(0);

        for (String face: facesList) {
            String[] vertexIndices = face.split(" ");
            for (int i = 1; i <= 3; i++) {
                String[] indices = vertexIndices[i].split("/");
                short vertexIndex = Short.parseShort(indices[0]);
                short normalIndex = Short.parseShort(indices[2]);
                facesVertexBuffer.put((short)(vertexIndex - 1));
                facesNormalBuffer.put((short)(normalIndex - 1));
            }
        }
        facesVertexBuffer.position(0);
        facesNormalBuffer.position(0);

        verticesList.clear();
        normalList.clear();

        scanner.close();
    } catch (IOException e) {
        e.printStackTrace();
        throw new RuntimeException("Failed to load model: " + ObjName, e);
    }

    // Размечаем буфер цветов
    float[] colorData = new float[facesList.size() * 4];
    for (int v = 0; v < facesList.size(); v++) {
        colorData[4 * v] = color[0];
        colorData[4 * v + 1] = color[1];
```

```
            colorData[4 * v + 2] = color[2];
            colorData[4 * v + 3] = color[3];
        }

        ByteBuffer bColor = ByteBuffer.allocateDirect(colorData.length * 4);
        bColor.order(ByteOrder.nativeOrder());
        colorBuffer = bColor.asFloatBuffer();
        colorBuffer.put(colorData).position(0);
    }

    void render(int positionAttribute, int normalAttribute, int colorAttribute) {
        // Устанавливаем буферы в позицию
        colorBuffer.position(0);
        normalBuffer.position(0);
        verticesBuffer.position(0);
        facesVertexBuffer.position(0);
        facesNormalBuffer.position(0);

        // Определяем массивы для буферов вершин, нормали и цвета и "подключаем" их
        GLES20.glVertexAttribPointer(positionAttribute, 3, GLES20.GL_FLOAT, false,
            0, verticesBuffer);
        GLES20.glEnableVertexAttribArray(positionAttribute);

        GLES20.glVertexAttribPointer(normalAttribute, 3, GLES20.GL_FLOAT, false,
            0, normalBuffer);
        GLES20.glEnableVertexAttribArray(normalAttribute);

        GLES20.glVertexAttribPointer(colorAttribute, 4, GLES20.GL_FLOAT, false,
            0, colorBuffer);
        GLES20.glEnableVertexAttribArray(colorAttribute);

        // Рисуем треугольниками объекты
        GLES20.glDrawElements(GLES20.GL_TRIANGLES, facesList.size() * 3,
```

```
            GLES20.GL_UNSIGNED_SHORT, facesVertexBuffer);
    }
}
```

ShaderWorks.java

```
package com.example.curs_work;

import android.opengl.GLES20;

class ShadersWork {
    private int mProgram;

    private final String mVertexS;
    private final String mFragmentS;

    ShadersWork(String vID, String fID) {
        this.mVertexS = vID;
        this.mFragmentS = fID;

        if (createProgram() != 1) {
            throw new RuntimeException("Error at creating shaders");
        }
    }

    private int createProgram() //загрузка шейдеров
    {
        int mVertexShader = loadShader(GLES20.GL_VERTEX_SHADER, mVertexS);
        if (mVertexShader == 0) {
            return 0;
        }

        int mPixelShader = loadShader(GLES20.GL_FRAGMENT_SHADER, mFragmentS);
        if (mPixelShader == 0) {
            return 0;
```

```
    }

    /*активация шейдеров*/
    mProgram = GLES20.glCreateProgram();
    if (mProgram != 0) {
        GLES20.glAttachShader(mProgram, mVertexShader);
        GLES20.glAttachShader(mProgram, mPixelShader);
        GLES20.glLinkProgram(mProgram);
        int[] linkStatus = new int[1];
        GLES20.glGetProgramiv(mProgram, GLES20.GL_LINK_STATUS, linkStatus, 0);
        if (linkStatus[0] != GLES20.GL_TRUE) {
            GLES20.glDeleteProgram(mProgram);
            mProgram = 0;
            return 0;
        }
    } else {
        return -1;
    }

    return 1;
}

private int loadShader(int shaderType, String source) //загрузка шейдеров
{
    int shader = GLES20.glCreateShader(shaderType);
    if (shader != 0) {
        GLES20.glShaderSource(shader, source);
        GLES20.glCompileShader(shader);
        int[] compiled = new int[1];
        GLES20.glGetShaderiv(shader, GLES20.GL_COMPILE_STATUS, compiled, 0);
        if (compiled[0] == 0) {
            GLES20.glDeleteShader(shader);
            shader = 0;
```

```
            }
        }
        return shader;
    }

    int getProgram() {
        return mProgram;
    }
}
```

MyRenderer.java

```
package com.example.curs_work;

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

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

public class MyRenderer implements GLSurfaceView.Renderer {
    private ShadersWork mSimpleShadowProgram;

    private int mActiveProgram; //дескриптор текущей программы

    private final float[] mMVMatrix = new float[16]; //перемноженная матрица моделей и отображения
    private final float[] mMVPMatrix = new float[16]; //перемноженная матрица моделей, отображения и проекции
    private final float[] mViewMatrix = new float[16]; //матрица отображения
    private final float[] mModelMatrix = new float[16]; //матрица моделей
    private final float[] mNormalMatrix = new float[16]; //матрица нормалей
    private final float[] mLightMvpMatrix = new float[16]; //матрица света для перемноженных матриц проекции и
отображения
    private final float[] mProjectionMatrix = new float[16]; //матрица проекции
    private final float[] mLightPosInEyeSpace = new float[16]; //матрица позиции света(?)
    private final float[] mActualLightPosition = new float[4]; //матрица текущего расположения света
```

```
private final float[] mLightProjectionMatrix = new float[16]; //матрица проекции для света

private final float[] mLightPosModel = new float[] //вектор света
{
   0.1 f, 10.0 f, 0.1 f, 1.0 f
};

private float s = 0;
private int mDisplayWidth;
private int mDisplayHeight;

/*уникальные переменные и атрибуты для матриц*/
private int scene_mvMatrixUniform;
private int scene_lightPosUniform;
private int scene_mvpMatrixUniform;
private int scene_normalMatrixUniform;
private int scene_shadowProjMatrixUniform;

private int scene_colorAttribute;
private int scene_normalAttribute;
private int scene_positionAttribute;

private final Context c;

/*объекты на экране*/
private TableObjects Cup;
private TableObjects Table;
private TableObjects Beets;
private TableObjects Apple;
private TableObjects Orange;
private TableObjects Cabbage;

MyRenderer(Context c) {
   this.c = c;
}
```

```
@Override
public void onSurfaceCreated(GL10 unused, EGLConfig config) {
    GLES20.glClearColor(1.0 f, 1.0 f, 1.0 f, 1.0 f); //очищаем буфер
    GLES20.glEnable(GLES20.GL_DEPTH_TEST); //активируем буфер глубины
    GLES20.glEnable(GLES20.GL_CULL_FACE); //удаляем задние грани

    Apple = new TableObjects(c, new float[] {
        1 f, 0 f, 0 f, 1.0 f
    }, "orange.obj");
    Cup = new TableObjects(c, new float[] {
        0.0 f, 0.2 f, 0.3 f, 1.0 f
    }, "cup.obj");
    Table = new TableObjects(c, new float[] {
        1 f, 1 f, 1 f, 1.0 f
    }, "Table.obj");
    Beets = new TableObjects(c, new float[] {
        0.5 f, 0.0 f, 0.5 f, 1.0 f
    }, "beets.obj");
    Orange = new TableObjects(c, new float[] {
        1.0 f, 0.5 f, 0 f, 1.0 f
    }, "orange.obj");
    Apple = new TableObjects(c, new float[] {
        0.5 f, 0.9 f, 0.5 f, 1.0 f
    }, "cabbage.obj");
    Cabbage = new TableObjects(c, new float[] {
        1.0 f, 0.0 f, 0.0 f, 1.0 f
    }, "apple.obj");
    /*загрузка и установление шейдеров*/
    String depth_tex_v_with_shadow_shader = "uniform mat4 uMVPMatrix;\n" +
        "uniform mat4 uMVMatrix;\n" +
        "uniform mat4 uNormalMatrix;\n" +
        "uniform mat4 uShadowProjMatrix;\n" +
        "attribute vec4 aPosition;\n" +
        "attribute vec4 aColor;\n" +
        "attribute vec3 aNormal;\n" +
        "varying vec3 vPosition;      \t\t\n" +
```

```
    "varying vec4 vColor;          \t\t\n" +
    "varying vec3 vNormal;\n" +
    "varying vec4 vShadowCoord;\n" +
    "\n" +
    "void main() {\n" +
    "\tvPosition = vec3(uMVMatrix * aPosition);\n" +
    "\tvColor = aColor;\n" +
    "\tvNormal = vec3(uNormalMatrix * vec4(aNormal, 0.0));\n" +
    "\tvShadowCoord = uShadowProjMatrix * aPosition;\n" +
    "\tgl_Position = uMVPMatrix * aPosition;                    \n" +
    "}";
String depth_tex_f_with_simple_shadow_shader = "precision mediump float;\n" +
    "\n" +
    "uniform vec3 uLightPos;\n" +
    "uniform sampler2D uShadowTexture;\n" +
    "uniform float uxPixelOffset;\n" +
    "uniform float uyPixelOffset;\n" +
    "varying vec3 vPosition;\n" +
    "varying vec4 vColor;\n" +
    "varying vec3 vNormal;\n" +
    "varying vec4 vShadowCoord;\n" +
    "\n" +
    "float shadowSimple(){\n" +
    "\tvec4 shadowMapPosition = vShadowCoord / vShadowCoord.w;\n" +
    "\tfloat distanceFromLight = texture2D(uShadowTexture, shadowMapPosition.st).z;\n" +
    "\tfloat bias = 0.001;\n" +
    "\treturn float(distanceFromLight > shadowMapPosition.z - bias);\n" +
    "}\n" +
    "  \n" +
    "void main() {\n" +
    "\tvec3 lightVec = uLightPos - vPosition;\n" +
    "\tlightVec = normalize(lightVec);\n" +
    "\tfloat specular = pow(max(dot(vNormal, lightVec), 0.0), 5.0);\n" +
    "\tfloat diffuse = max(dot(vNormal, lightVec), 0.1);\n" +
    "\tfloat ambient = 0.3;\n" +
    "   \tfloat shadow = 1.0;\n" +
```

```
            "\t\tif (vShadowCoord.w > 0.0) {\n" +
            "\t\t\tshadow = shadowSimple();\n" +
            "\t\t\tshadow = (shadow * 0.9) + 0.9;\n" +
            "\t\t}\n" +
            "    gl_FragColor = (vColor * (diffuse + ambient + specular) * shadow);\n" +
            "}  ";
        mSimpleShadowProgram = new ShadersWork(depth_tex_v_with_shadow_shader,
depth_tex_f_with_simple_shadow_shader);
        mActiveProgram = mSimpleShadowProgram.getProgram();
    }

    @Override
    public void onSurfaceChanged(GL10 unused, int width, int height) //изменяем положение объектов
    {
        mDisplayWidth = width;
        mDisplayHeight = height;
        GLES20.glViewport(0, 0, mDisplayWidth, mDisplayHeight);

        float ratio = (float) mDisplayWidth / mDisplayHeight;

        Matrix.frustumM(mProjectionMatrix, 0, -ratio, ratio, -1.0 f, 1.0 f, 1.0 f, 100.0 f);
        Matrix.frustumM(mLightProjectionMatrix, 0, -1.1 f * ratio, 1.1 f * ratio, 1.1 f * -1.0 f, 1.1 f * 1.0 f, 1.0 f, 100.0 f);
    }

    @Override
    public void onDrawFrame(GL10 unused) //рисуем на экран
    {
        //получаем дескриптор программы
        mActiveProgram = mSimpleShadowProgram.getProgram();

        //устанавливаем камеру в позицию просмотра
        Matrix.setLookAtM(mViewMatrix, 0,
            3, 5, 0,
            0, 0, 0, -1, 0, 0);

        /*получаем расположение переменных в программе*/
```

```
    scene_mvpMatrixUniform = GLES20.glGetUniformLocation(mActiveProgram, "uMVPMatrix");
    scene_mvMatrixUniform = GLES20.glGetUniformLocation(mActiveProgram, "uMVMatrix");
    scene_normalMatrixUniform = GLES20.glGetUniformLocation(mActiveProgram, "uNormalMatrix");
    scene_lightPosUniform = GLES20.glGetUniformLocation(mActiveProgram, "uLightPos");
    scene_shadowProjMatrixUniform = GLES20.glGetUniformLocation(mActiveProgram, "uShadowProjMatrix");
    scene_positionAttribute = GLES20.glGetAttribLocation(mActiveProgram, "aPosition");
    scene_normalAttribute = GLES20.glGetAttribLocation(mActiveProgram, "aNormal");
    scene_colorAttribute = GLES20.glGetAttribLocation(mActiveProgram, "aColor");

    float[] basicMatrix = new float[16];

    /*делаем единичные матрицы*/
    Matrix.setIdentityM(basicMatrix, 0);
    Matrix.setIdentityM(mModelMatrix, 0);
    /*перемножаем единичную матрицу с вектором текущей позиции света*/
    Matrix.multiplyMV(mActualLightPosition, 0, basicMatrix, 0, mLightPosModel, 0);

    /*убираем задние грани объектов*/
    GLES20.glCullFace(GLES20.GL_BACK);
    /*рисуем сами объекты*/
    draw_picture();
}

private void draw_picture() //отрисовываем объекты
{
    GLES20.glBindFramebuffer(GLES20.GL_FRAMEBUFFER, 0);

    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT | GLES20.GL_DEPTH_BUFFER_BIT); //очищаем буфер

    GLES20.glUseProgram(mActiveProgram); //используем нашу программу

    GLES20.glViewport(0, 0, mDisplayWidth, mDisplayHeight); //устанавливаем точку просмотра

    float[] tempResultMatrix = new float[16]; //буферная матрица

    float[] bias = new float[] //матрица смещения
```

```
{
    0.5 f, 0.0 f, 0.0 f, 0.0 f,
    0.0 f, 0.5 f, 0.0 f, 0.0 f,
    0.0 f, 0.0 f, 0.5 f, 0.0 f,
    0.5 f, 0.5 f, 0.5 f, 1.0 f
};

float[] depthBiasMVP = new float[16]; //матрица глубины

Matrix.multiplyMM(tempResultMatrix, 0, mViewMatrix, 0, mModelMatrix, 0); //перемножаем матрицы моделей и отображения
System.arraycopy(tempResultMatrix, 0, mMVMatrix, 0, 16); //копируем результат в общую матрицу

GLES20.glUniformMatrix4fv(scene_mvMatrixUniform, 1, false, mMVMatrix, 0); //устанавливем значение переменной для матрицы

Matrix.invertM(tempResultMatrix, 0, mMVMatrix, 0); //инвертируем общую матрицу и схораняем в буфер
Matrix.transposeM(mNormalMatrix, 0, tempResultMatrix, 0); //затем транспонируем буфер и сохраняем в матрице нормалей

GLES20.glUniformMatrix4fv(scene_normalMatrixUniform, 1, false, mNormalMatrix, 0); //устанавливем значение переменной для матрицы нормалей

Matrix.multiplyMM(tempResultMatrix, 0, mProjectionMatrix, 0, mMVMatrix, 0); //перемножаем матрицы проекции и перемноженные матрицы моделей и отображения
System.arraycopy(tempResultMatrix, 0, mMVPMatrix, 0, 16); //копируем результат в БОЛЕЕ общую матрицу

GLES20.glUniformMatrix4fv(scene_mvpMatrixUniform, 1, false, mMVPMatrix, 0); //устанавливем значение переменной для MVP матрицы

Matrix.multiplyMV(mLightPosInEyeSpace, 0, mViewMatrix, 0, mActualLightPosition, 0); //перемножаем матрицу проекции и вектор света

GLES20.glUniform3f(scene_lightPosUniform, mLightPosInEyeSpace[0], mLightPosInEyeSpace[1], mLightPosInEyeSpace[2]); //устанавливаем занчение для вектора

Matrix.multiplyMM(depthBiasMVP, 0, bias, 0, mLightMvpMatrix, 0); //перемножаем смещение и матрицу света
System.arraycopy(depthBiasMVP, 0, mLightMvpMatrix, 0, 16); //сохраняем результат в матрице глубины
```

```
        GLES20.glUniformMatrix4fv(scene_shadowProjMatrixUniform, 1, false, mLightMvpMatrix, 0); //устанавливаем переменную


        /*рисование объектов*/
        Cup.render(scene_positionAttribute, scene_normalAttribute, scene_colorAttribute);
        Apple.render(scene_positionAttribute, scene_normalAttribute, scene_colorAttribute);
        Table.render(scene_positionAttribute, scene_normalAttribute, scene_colorAttribute);
        Beets.render(scene_positionAttribute, scene_normalAttribute, scene_colorAttribute);
        Orange.render(scene_positionAttribute, scene_normalAttribute, scene_colorAttribute);
        Cabbage.render(scene_positionAttribute, scene_normalAttribute, scene_colorAttribute);
    }
}
```