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

Курсовая работа
по дисциплине
«Программирование мобильных устройств»

Выполнил: студент 4 курса
ИВТ, гр. ИП-013
Копытина Т. А.

Проверил: ст. пр. кафедры ПМиК
Павлова У.В.

Новосибирск 2023

Оглавление

Задание

Создайте программу в которой нарисован стол на OpenGL ES 2.0.

На столе лежат различные фрукты/овощи (не менее 4 различных), стакан с напитком. Имеется освещение по модели Фонга.

Решение поставленной задачи

Для реализации задания на курсовую работу были разработаны данные файлы:

MainActivity.kt:

Она наследуется от AppCompatActivity и отвечает за создание и управление OpenGL-контекстом, определяет, поддерживается ли устройство OpenGL ES 2.0. Создает и настраивает GLSurfaceView, который предоставляет возможность отрисовки 3D-сцены, а также реализует методы onPause и onResume для приостановки и возобновления работы OpenGL приложения при изменении активности.

MyRenderer.kt:

Этот класс реализует интерфейс GLSurfaceView.Renderer и является основным классом для рендеринга 3D-сцены. В конструкторе инициализируется список objects, представляющий объекты 3D на сцене, а также задается цвет фона сцены. Реализует методы для инициализации OpenGL, настройки сцены, и отрисовки моделей. Метод onSurfaceCreated настраивает параметры OpenGL, включает буфер глубины и устанавливает режим смешивания для прозрачности. Метод onSurfaceChanged устанавливает размер отображаемой области при изменении размеров экрана. Метод onDrawFrame отвечает за отрисовку моделей, включая анимацию и взаимодействие с освещением.

Object.kt:

Этот класс представляет 3D-объекты и реализует методы для настройки параметров объекта, таких как позиция, масштаб, цвет и текстуры. Содержит методы для инициализации объектов, настройки вершинных и текстурных буферов, а также метод onDrawFrame для отрисовки объекта.

TextureUtils.kt:

Этот класс предоставляет утилиты для работы с текстурами, включая загрузку текстур из изображений.

Utils.kt:

Этот класс содержит утилитарные методы, такие как создание буферов и преобразование InputStream в строку.

ShaderUtils.kt:

Этот класс предоставляет утилиты для загрузки и компиляции шейдеров OpenGL ES 2.0.

MatrixUtils.kt:

Этот класс реализует само освещение. Оно строится от главного объекта(свечи), до других объектов расположенных на поверхности.

Демонстрация работы программы

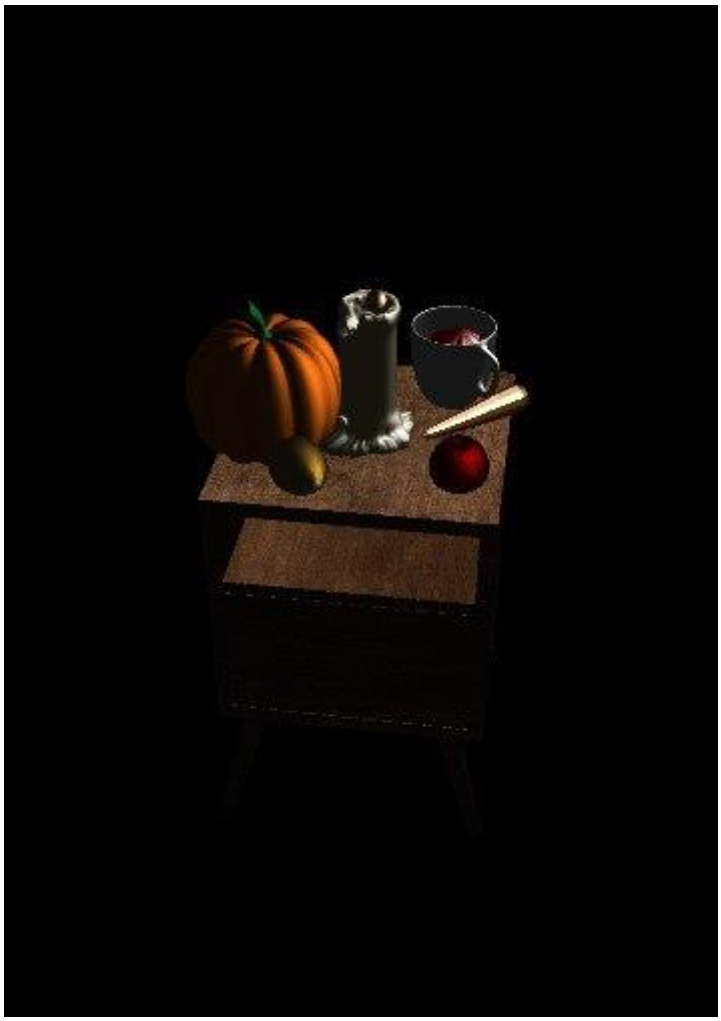

Рис 1. Демонстрация работы программы

Листинг

Object.kt:

```
package ru.lyovkin.CURS


import android.opengl.GLES20.GL_FLOAT
import android.opengl.GLES20.GL_TEXTURE0
import android.opengl.GLES20.GL_TEXTURE_2D
import android.opengl.GLES20.GL_TRIANGLES
import android.opengl.GLES20.GL_UNSIGNED_INT
import android.opengl.GLES20.glActiveTexture
import android.opengl.GLES20.glBindTexture
import android.opengl.GLES20.glDrawElements
import android.opengl.GLES20.glEnableVertexAttribArray
import android.opengl.GLES20.glGetAttribLocation
import android.opengl.GLES20.glGetUniformLocation
import android.opengl.GLES20.glUniform1f
import android.opengl.GLES20.glUniform1i
import android.opengl.GLES20.glUniform3fv
import android.opengl.GLES20.glUniformMatrix4fv
import android.opengl.GLES20.glUseProgram
import android.opengl.GLES20.glVertexAttribPointer
import android.opengl.GLES20.glViewport
import android.opengl.Matrix
import android.util.Log
import de.javagl.obj.Obj
import de.javagl.obj.ObjData
import de.javagl.obj.ObjReader
import de.javagl.obj.ObjUtils
import java.io.InputStream
import java.nio.FloatBuffer
import java.nio.IntBuffer
import javax.microedition.khronos.egl.EGLConfig
import javax.microedition.khronos.opengles.GL10
import kotlin.math.cos
import kotlin.math.sin


//import kotlin.math.cos
//import kotlin.math.sin


class Camera {
    private var viewMatrix = FloatArray(16)


    fun setLookAtM(eyeX: Float, eyeY: Float, eyeZ: Float, centerX: Float, centerY: Float, centerZ: Float, upX: Float, upY: Float, upZ: Float) {
        Matrix.setLookAtM(viewMatrix, 0, eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
    }
```

```
    fun getMatrix(): FloatArray {
        return viewMatrix
    }
}
class Object(
    private val indicesBuffer: IntBuffer,
    private val verticesBuffer: FloatBuffer,
    private val texCordsBuffer: FloatBuffer,
    private val normalBuffer: FloatBuffer,

    private var vertexShader: String = "",
    private var fragmentShader: String = "",

    private val texture: InputStream? = null,

    private var colorBuffer: FloatBuffer = verticesBuffer,

    private var alpha: Float = 1f,

    private val camera: Camera = Camera()



) : RenderObject {
    private val tag = this.javaClass.simpleName

    companion object {
        private fun fromObj(
            vertexShader: String,
            fragmentShader: String,
            obj: Obj,
            texture: InputStream? = null
        ): Object {
            val indices = ObjData.getFaceVertexIndices(obj)
            val vertices = ObjData.getVertices(obj)
            val coords = ObjData.getTexCoords(obj, 2)
            val normals = ObjData.getNormals(obj)

            return Object(
                indices,
                vertices,
                coords,
                normals,
                vertexShader,
                fragmentShader,
                texture,
            )
        }

        fun fromInputStream(
            vertexShader: String,
            fragmentShader: String,
            model: InputStream,
            texture: InputStream? = null
```

```
    ): Object {
        return fromObj(
            vertexShader,
            fragmentShader,
            ObjUtils.convertToRenderable(
                ObjReader.read(model)
            ),
            texture,
        )
    }
}

/* GL ids */
private var programId = 0
private var textureId = 0

/* Vertex shader */
private var vertexLocation = 0
private var vertexColorLocation = 0
private var vertexAlphaLocation = 0
private var textureCordLocation = 0
private var vertexNormalLocation = 0
private var lightDirection = 0
private var projectionLocation = 0
private var modelViewLocation = 0

private var projectionMatrix = MatrixUtils.getIdentityMatrix()
private var modelViewMatrix = MatrixUtils.getIdentityMatrix()

/* Fragment shader */
private var samplerLocation = 0

/* Object fields */
private var _rotate = floatArrayOf(0f, 0f, 0f)
private var _position = floatArrayOf(0f, 0f, 0f)
private var _scale = floatArrayOf(1f, 1f, 1f)
private var _lightDirection = floatArrayOf(0f, 0f, 0f)

override var x: Float
    get() = _position[0]
    set(value) {
        _position[0] = value
    }

override var y: Float
    get() = _position[1]
    set(value) {
        _position[1] = value
    }

override var z: Float
    get() = _position[2]
    set(value) {
        _position[2] = value
```

```
    }

override var rotateX: Float
    get() = _rotate[0]
    set(value) {
        _rotate[0] = value
    }

override var rotateY: Float
    get() = _rotate[1]
    set(value) {
        _rotate[1] = value
    }

override var rotateZ: Float
    get() = _rotate[2]
    set(value) {
        _rotate[2] = value
    }

override var scaleX: Float
    get() = _scale[0]
    set(value) {
        _scale[0] = value
    }

override var scaleY: Float
    get() = _scale[1]
    set(value) {
        _scale[1] = value
    }

override var scaleZ: Float
    get() = _scale[2]
    set(value) {
        _scale[2] = value
    }

init {
    setColor(0.8f, 0.8f, 0.9f)
}

private fun setupGraphics(width: Int, height: Int): Boolean {
    programId = ShaderUtils.createProgram(vertexShader, fragmentShader)
    if (programId == 0) {
        Log.e(tag, "Could not create program")
        return false
    }

    if (texture != null) {
        textureId = TextureUtils.loadTexture(texture)
    }

    vertexLocation = glGetAttribLocation(programId, "vertexPosition")
```

```
        vertexColorLocation = glGetAttribLocation(programId, "vertexColor")
        vertexAlphaLocation = glGetUniformLocation(programId, "vertexAlpha")
        textureCordLocation = glGetAttribLocation(programId, "vertexTextureCord")
        vertexNormalLocation = glGetAttribLocation(programId, "vertexNormal")
        lightDirection = glGetUniformLocation(programId, "lightDirection")
        projectionLocation = glGetUniformLocation(programId, "projection")
        modelViewLocation = glGetUniformLocation(programId, "modelView")
        samplerLocation = glGetUniformLocation(programId, "texture")

        MatrixUtils.matrixPerspective(
            projectionMatrix,
            45f,
            width.toFloat() / height.toFloat(),
            0.1f,
            100f
        )

        glViewport(0, 0, width, height)

        return textureId != 0
    }

    override fun setColor(r: Float, g: Float, b: Float, a: Float) {
        val color = FloatArray(colorBuffer.capacity()) { 0f }
        for (i in 0 until color.indices.count() step 3) {
            color[i] = r
            color[i + 1] = g
            color[i + 2] = b
        }
        colorBuffer = Utils.createBuffer(color)
        alpha = a
    }

    override fun setPosition(x: Float, y: Float, z: Float) {
        this.x = x
        this.y = y
        this.z = z
    }

    override fun setRotation(x: Float, y: Float, z: Float) {
        this.rotateX = x
        this.rotateY = y
        this.rotateZ = z
    }

    override fun setScale(x: Float, y: Float, z: Float) {
```

```
        this.scaleX = x
        this.scaleY = y
        this.scaleZ = z
    }

    override fun setLightDirection(x: Float, y: Float, z: Float) {
        this._lightDirection[0] = x
        this._lightDirection[1] = y
        this._lightDirection[2] = z
    }

    override fun onSurfaceCreated(gl: GL10, config: EGLConfig) {
    }

    override fun onSurfaceChanged(gl: GL10, width: Int, height: Int) {
        setupGraphics(width, height)
    }

    private var angle: Float = 0f
    private var radius = 4f
    private var centerX = 0f
    private var centerY = 0f
    private var centerZ = 0f
    override fun onDrawFrame(gl: GL10) {
        /* Matrix setup */
MatrixUtils.matrixIdentityFunction(modelViewMatrix)

MatrixUtils.matrixRotateX(modelViewMatrix, rotateX)

MatrixUtils.matrixRotateY(modelViewMatrix, rotateY)

MatrixUtils.matrixRotateZ(modelViewMatrix, rotateZ)
        MatrixUtils.matrixScale(modelViewMatrix, scaleX, scaleY, scaleZ)

MatrixUtils.matrixTranslate(modelViewMatrix, x, y, z)

        //x0 = x0 + r * cos a
        //y0 = y0 + r * sin a
        //angle += 0.002f
        angle = 180f
        val y = centerY + radius * cos(angle)
        val z = centerZ + radius * sin(angle)

        /* Установка матрицы вида из камеры */
        camera.setLookAtM(3f, y, z, centerX, centerY, centerZ, 1f, 0f, 0f);
        val viewMatrix = camera.getMatrix()

        /* Умножение матрицы модели-вида на матрицу вида */
```

```
        Matrix.multiplyMM(modelViewMatrix,      0,
viewMatrix, 0, modelViewMatrix, 0)

        /* Установка шейдерной программы */
        glUseProgram(programId)

        /* Передача позиции */
        glVertexAttribPointer(vertexLocation,      3,
GL_FLOAT, false, 0, verticesBuffer)
        glEnableVertexAttribArray(vertexLocation)

        /* Передача цвета */
        glVertexAttribPointer(vertexColorLocation,
3, GL_FLOAT, false, 0, colorBuffer)

glEnableVertexAttribArray(vertexColorLocation)

        glUniform1f(vertexAlphaLocation, alpha)

        /* Передача текстурных координат */
        glVertexAttribPointer(textureCordLocation,
2, GL_FLOAT, false, 0, texCordsBuffer)

glEnableVertexAttribArray(textureCordLocation)

        /* Передача нормалей, если они есть */
        if (normalBuffer.capacity() > 0) {

glVertexAttribPointer(vertexNormalLocation,     3,
GL_FLOAT, false, 0, normalBuffer)

glEnableVertexAttribArray(vertexNormalLocation
)
        }

        /* Передача направления света */
        glUniform3fv(lightDirection,               1,
_lightDirection, 0)

        /* Активация текстуры и передача в шейдер
*/
        glActiveTexture(GL_TEXTURE0)
        glBindTexture(GL_TEXTURE_2D,
textureId)
        glUniform1i(samplerLocation, 0)

        /* Передача матрицы проекции и модели-
вида в шейдер */
        glUniformMatrix4fv(projectionLocation,     1,
false, projectionMatrix, 0)
        glUniformMatrix4fv(modelViewLocation,   1,
false, modelViewMatrix, 0)

        /* Отрисовка */
        glDrawElements(GL_TRIANGLES,
indicesBuffer.capacity(),    GL_UNSIGNED_INT,
indicesBuffer)
    }
}
```

RenderObject.kt:

```
package ru.lyovkin.CURS


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


interface RenderObject {
    var x: Float
    var y: Float
    var z: Float
    var rotateX: Float
    var rotateY: Float
    var rotateZ: Float
    var scaleX: Float
    var scaleY: Float
    var scaleZ: Float
    fun setColor(r: Float, g: Float, b: Float, a: Float = 1f)
    fun setPosition(x: Float, y: Float, z: Float)
    fun setRotation(x: Float, y: Float, z: Float)
    fun setScale(x: Float, y: Float, z: Float)
    fun setLightDirection(x: Float, y: Float, z: Float)
    fun onSurfaceCreated(gl: GL10, config: EGLConfig)
    fun onSurfaceChanged(gl: GL10, width: Int, height: Int)
    fun onDrawFrame(gl: GL10)
}
```

ShaderUtils.kt:

```
package ru.lyovkin.CURS

import android.opengl.GLES20
import android.util.Log

class ShaderUtils {
    companion object {
        private val tag = ShaderUtils::class.java.simpleName

        private fun loadShader(shaderType: Int, shaderSource: String): Int {
            val shader = GLES20.glCreateShader(shaderType)
            if (shader == 0) {
                return shader
            }

            GLES20.glShaderSource(shader, shaderSource)
            GLES20.glCompileShader(shader)

            val compiled = IntArray(1)
            GLES20.glGetShaderiv(shader, GLES20.GL_COMPILE_STATUS, compiled, 0)
            if (compiled[0] != 0) {
                return shader
            }

            val infoLen = IntArray(1)
            GLES20.glGetShaderiv(shader, GLES20.GL_INFO_LOG_LENGTH, infoLen, 0)
            if (infoLen[0] != 0) {
                val logStr = GLES20.glGetShaderInfoLog(shader)
                Log.e(tag, "Type: $shaderType\n$logStr")
            }

            GLES20.glDeleteShader(shader)
            return 0
        }

        fun createProgram(vertexSource: String, fragmentSource: String): Int {
            val vertexShader = loadShader(GLES20.GL_VERTEX_SHADER, vertexSource)
            if (vertexShader == 0) {
                return 0
            }

            val fragmentShader = loadShader(GLES20.GL_FRAGMENT_SHADER, fragmentSource)
            if (fragmentShader == 0) {
                return 0
            }
```

```
        var program = GLES20.glCreateProgram()
        GLES20.glAttachShader(program, vertexShader)
        GLES20.glAttachShader(program, fragmentShader)

        GLES20.glLinkProgram(program)
        val linkStatus = intArrayOf(GLES20.GL_FALSE)
        GLES20.glGetProgramiv(program, GLES20.GL_LINK_STATUS, linkStatus, 0)

        if (linkStatus[0] == GLES20.GL_TRUE) {
            return program
        }

        val bufLength = IntArray(1)
        GLES20.glGetProgramiv(program, GLES20.GL_INFO_LOG_LENGTH, bufLength, 0)
        if (bufLength[0] == 0) {
            val logStr = GLES20.glGetProgramInfoLog(program)
            Log.e(tag, logStr)
        }

        GLES20.glDeleteProgram(program)
        return 0
    }
  }
}
```

MatrixUtils.kt:

```
package ru.lyovkin.CURS

import kotlin.math.PI
import kotlin.math.cos
import kotlin.math.sin
import kotlin.math.tan

class MatrixUtils {
    companion object {
        fun getIdentityMatrix(): FloatArray {
            val tempMatrix = FloatArray(16)
            matrixIdentityFunction(tempMatrix)
            return tempMatrix
        }

        fun matrixIdentityFunction(matrix: FloatArray) {
            matrix[0] = 1.0f
            matrix[1] = 0.0f
            matrix[2] = 0.0f
            matrix[3] = 0.0f
            matrix[4] = 0.0f
            matrix[5] = 1.0f
            matrix[6] = 0.0f
            matrix[7] = 0.0f
            matrix[8] = 0.0f
            matrix[9] = 0.0f
            matrix[10] = 1.0f
            matrix[11] = 0.0f
            matrix[12] = 0.0f
            matrix[13] = 0.0f
            matrix[14] = 0.0f
            matrix[15] = 1.0f
        }

        fun matrixMultiply(destination: FloatArray, operand1: FloatArray, operand2: FloatArray) {
            val result = FloatArray(16)

            for (i in 0..3) {
                for (j in 0..3) {
                    result[4 * i + j] =
                        operand1[j] * operand2[4 * i] + operand1[4 + j] * operand2[4 * i + 1] +
                                operand1[8 + j] * operand2[4 * i + 2] + operand1[12 + j] * operand2[4 * i + 3]
                }
            }

            for (i in 0..15) {
                destination[i] = result[i]
            }
        }

        fun matrixRotateX(matrix: FloatArray, angle: Float) {
```

```
        val tempMatrix = FloatArray(16)
        matrixIdentityFunction(tempMatrix)

        tempMatrix[5] = cos(matrixDegreesToRadians(angle))
        tempMatrix[9] = -sin(matrixDegreesToRadians(angle))
        tempMatrix[6] = sin(matrixDegreesToRadians(angle))
        tempMatrix[10] = cos(matrixDegreesToRadians(angle))

        matrixMultiply(matrix, tempMatrix, matrix)
    }

    fun matrixRotateY(matrix: FloatArray, angle: Float) {
        val tempMatrix = FloatArray(16)
        matrixIdentityFunction(tempMatrix)

        tempMatrix[0] = cos(matrixDegreesToRadians(angle))
        tempMatrix[8] = sin(matrixDegreesToRadians(angle))
        tempMatrix[2] = -sin(matrixDegreesToRadians(angle))
        tempMatrix[10] = cos(matrixDegreesToRadians(angle))

        matrixMultiply(matrix, tempMatrix, matrix)
    }

    fun matrixRotateZ(matrix: FloatArray, angle: Float) {
        val tempMatrix = FloatArray(16)
        matrixIdentityFunction(tempMatrix)

        tempMatrix[0] = cos(matrixDegreesToRadians(angle))
        tempMatrix[4] = -sin(matrixDegreesToRadians(angle))
        tempMatrix[1] = sin(matrixDegreesToRadians(angle))
        tempMatrix[5] = cos(matrixDegreesToRadians(angle))

        matrixMultiply(matrix, tempMatrix, matrix)
    }

    fun matrixTranslate(matrix: FloatArray, x: Float, y: Float, z: Float) {
        val tempMatrix = FloatArray(16)
        matrixIdentityFunction(tempMatrix)

        tempMatrix[12] = x
        tempMatrix[13] = y
        tempMatrix[14] = z
```

```
        matrixMultiply(matrix, tempMatrix, matrix)
    }

    fun matrixPerspective(
        matrix: FloatArray,
        fieldOfView: Float,
        aspectRatio: Float,
        zNear: Float,
        zFar: Float
    ) {
        val yMax: Float = (zNear * tan(fieldOfView * PI / 360.0)).toFloat()
        val xMax: Float = yMax * aspectRatio
        matrixFrustum(matrix, -xMax, xMax, -yMax, yMax, zNear, zFar)
    }

    private fun matrixFrustum(
        matrix: FloatArray,
        left: Float,
        right: Float,
        bottom: Float,
        top: Float,
        zNear: Float,
        zFar: Float
    ) {
        val temp: Float = 2.0f * zNear
        val xDistance: Float = right - left
        val yDistance: Float = top - bottom
        val zDistance: Float = zFar - zNear
        matrixIdentityFunction(matrix)
        matrix[0] = temp / xDistance
        matrix[5] = temp / yDistance
        matrix[8] = (right + left) / xDistance
        matrix[9] = (top + bottom) / yDistance
        matrix[10] = (-zFar - zNear) / zDistance
        matrix[11] = -1.0f
        matrix[14] = (-temp * zFar) / zDistance
        matrix[15] = 0.0f
    }
    fun matrixScale(matrix: FloatArray, x: Float, y: Float, z: Float) {
        val tempMatrix = FloatArray(16)
        matrixIdentityFunction(tempMatrix)

        tempMatrix[0] = x
        tempMatrix[5] = y
        tempMatrix[10] = z
        matrixMultiply(matrix, tempMatrix, matrix);
    }
    fun matrixDegreesToRadians(degrees: Float): Float {
        return (PI * degrees / 180.0f).toFloat()
    }
}}
```

MyRenderer.kt:

```
package ru.lyovkin.CURS

import android.content.Context
import android.opengl.GLES20.*
import android.opengl.GLSurfaceView
import javax.microedition.khronos.egl.EGLConfig
import javax.microedition.khronos.opengles.GL10

class MyRenderer(ctx: Context) : GLSurfaceView.Renderer{
    private val renderObjectsArr: MutableList<RenderObject> = mutableListOf()

    private val bgColorR: Float = 0.0f
    private val bgColorG: Float = 0.0f
    private val bgColorB: Float = 0.0f
    private val bgColorA: Float = 1.0f

    init {
        val shader = Utils.inputStreamToString(ctx.assets.open("shaders/base_shader.vert"))
        val colorObject = Utils.inputStreamToString(ctx.assets.open("shaders/color_shader.frag"))
        val texture = Utils.inputStreamToString(ctx.assets.open("shaders/texture_shader.frag"))

        val tumbochka = Object.fromInputStream(
            shader,
            texture,
            ctx.assets.open("models/tumbochka.obj"),
            ctx.assets.open("models/tumbochka.png")
        )
        tumbochka.setPosition(-1f, 0f, 0f)
        tumbochka.setScale(1f, 1f, 1f)
        tumbochka.rotateZ = 270f
        tumbochka.rotateY = 45f

        val candle =
            Object.fromInputStream(shader, colorObject, ctx.assets.open("models/candle.obj"),)
        candle.setPosition(0.5f, 0f, 0f)
        candle.setScale(0.1f, 0.1f, 0.1f)
        candle.setColor(0.9f, 0.8f, 0.5f)
        candle.rotateZ = -90f

        val fire = Object.fromInputStream(shader, colorObject, ctx.assets.open("models/fire.obj"),)
        fire.setPosition(1f, 0.0f, 0.0f)
        fire.setScale(0.01f, 0.01f, 0.01f)
        fire.setColor(1f, 0.5f, 0f)

        val glassX = 0.7f
        val glassY = 0.3f
        val glassZ = -0.1f
```

```
        val glassScale = 0.15f
        val glass = Object.fromInputStream(shader, colorObject, ctx.assets.open("models/mug.obj"),)
        glass.setPosition(glassX, glassY, glassZ)
        glass.setScale(glassScale, glassScale, glassScale)
        glass.setColor(0.9f, 0.9f, 0.9f)
        glass.rotateZ = -90f

        val water =
            Object.fromInputStream(shader, colorObject, ctx.assets.open("models/candle.obj"),)
        water.setPosition(glassX+0.1f, glassY, glassZ)
        water.setScale(4/80f, 4/80f, 4/80f)
        water.setColor(1f, 0.0f, 0.0f)
        water.rotateZ = 90f

        val apple = Object.fromInputStream(
            shader,
            texture,
            ctx.assets.open("models/apple.obj"),
            ctx.assets.open("models/apple.jpg")
        )
        apple.setPosition(0.7f, 0.0f, -0.5f)
        apple.setScale(0.05f, 0.05f, 0.05f)
        apple.rotateY = -90f
        apple.rotateX = 90f

        val carrot =
            Object.fromInputStream(shader, colorObject, ctx.assets.open("models/carrot.obj"))
        carrot.setPosition(0.7f, 0.3f, -0.4f)
        carrot.setColor(1f, 0.5f, 0.0f)
        carrot.setScale(0.05f, 0.05f, 0.05f)
        //carrot.rotateZ = 90f
        //cucumber.rotateX = 30f

        val lemon = Object.fromInputStream(
            shader,
            texture,
            ctx.assets.open("models/lemon.obj"),
            ctx.assets.open("models/lemon.jpg")
        )
        lemon.setPosition(0.7f, -0.5f, -0.3f)
        lemon.setScale(0.005f, 0.005f, 0.005f)
        lemon.rotateZ = 90f

        val pumpkin = Object.fromInputStream(
            shader,
            texture,
            ctx.assets.open("models/pumpkin.obj"),
            ctx.assets.open("models/pumpkin.png")
        )
        pumpkin.setPosition(0.7f, -0.3f, 0.2f)
        pumpkin.setScale(0.3f, 0.3f, 0.3f)
        pumpkin.rotateZ = -90f

        renderObjectsArr.add(fire)
```

```
        renderObjectsArr.add(candle)
        renderObjectsArr.add(water)
        renderObjectsArr.add(glass)
        renderObjectsArr.add(apple)
        renderObjectsArr.add(carrot)
        renderObjectsArr.add(lemon)
        renderObjectsArr.add(pumpkin)
        renderObjectsArr.add(tumbochka)
    }

    override fun onSurfaceCreated(gl: GL10, config: EGLConfig) {
        glEnable(GL_DEPTH_TEST)
        glDepthFunc(GL_LEQUAL)

        glEnable(GL_BLEND)
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

        for (objectRender in renderObjectsArr) {
            objectRender.onSurfaceCreated(gl, config)
        }

    }

    override fun onSurfaceChanged(gl: GL10, width: Int, height: Int) {
        glViewport(0, 0, width, height)

        for (objectRender in renderObjectsArr) {
            objectRender.onSurfaceChanged(gl, width, height)
        }
    }

    override fun onDrawFrame(gl: GL10) {

        glClearColor(bgColorR, bgColorG, bgColorB, bgColorA)
        glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT)

        val fire = (renderObjectsArr[0] as Object)

        for (objectRender in renderObjectsArr) {
            objectRender.setLightDirection((fire.y + 0.5f) - objectRender.y, (fire.x) - objectRender.x, (fire.z) - objectRender.z)
            objectRender.onDrawFrame(gl)
        }
    }
}
```

TextureUtils.kt:

```
package ru.lyovkin.CURS

import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.graphics.Matrix
import android.graphics.Rect
import android.opengl.GLES20.*
import android.opengl.GLUtils
import java.io.InputStream


class TextureUtils {
    companion object {
        fun loadTexture(texture: InputStream): Int {
            val textureHandle = IntArray(1)
            glGenTextures(1, textureHandle, 0)

            if (textureHandle[0] == 0) {
                throw RuntimeException("Error loading texture.")
            }
            val options = BitmapFactory.Options()
            options.inScaled = false // No pre-scaling

            // Read in the resource
            var bitmap = BitmapFactory.decodeStream(texture, Rect(0, 0, 0, 0), options)
            val m = Matrix()
            m.preScale(1f, -1f)
            if (bitmap != null) {
                bitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.width, bitmap.height, m, false)
            }

            // Bind to the texture in OpenGL
            glBindTexture(GL_TEXTURE_2D, textureHandle[0])

            // Set filtering
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
            // Load the bitmap into the bound texture.
            GLUtils.texImage2D(GL_TEXTURE_2D, 0, bitmap, 0)
            // Recycle the bitmap, since its data has been loaded into OpenGL.
            bitmap?.recycle()

            return textureHandle[0]
        }
    }
}
```

MainActivity.kt:

```
package ru.lyovkin.CURS


import android.app.ActivityManager
import android.opengl.GLSurfaceView
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast


class MainActivity : AppCompatActivity() {
    private  val isProbablyEmulator: Boolean
        get() = (Build.FINGERPRINT.startsWith("generic")
                || Build.FINGERPRINT.startsWith("unknown")
                || Build.MODEL.contains("google_sdk")
                || Build.MODEL.contains("Emulator")
                || Build.MODEL.contains("Android SDK built for x86"))
    private var glSurfaceView: GLSurfaceView? = null
    private var isRendererSet = false


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val activityManager = getSystemService(ACTIVITY_SERVICE) as ActivityManager
        val configurationInfo = activityManager.deviceConfigurationInfo


        val isSupportES2 = configurationInfo.reqGlEsVersion >= 0x20000 || isProbablyEmulator


        if (!isSupportES2) {
            Toast.makeText(
                this,
                "This device does not support OpenGL ES 2.0",
                Toast.LENGTH_LONG
            ).show()
            return
        }


        glSurfaceView = GLSurfaceView(this)


        if (isProbablyEmulator) {
            glSurfaceView?.setEGLConfigChooser(
                8, 8, 8,
                8, 16, 0
            )
        }
```

```
glSurfaceView?.setEGLContextClientVersion(2)

glSurfaceView?.setRenderer(MyRenderer(this))
        glSurfaceView?.renderMode = GLSurfaceView.RENDERMODE_CONTINUOUSLY
        isRendererSet = true

        setContentView(glSurfaceView)
    }

    override fun onPause() {
        super.onPause()
        if (isRendererSet) {
            glSurfaceView?.onPause()
        }
    }

    override fun onResume() {
        super.onResume()
        if (isRendererSet) {
            glSurfaceView?.onResume()
        }
    }
}
```

Utils.kt:

```
package ru.lyovkin.CURS


import java.io.InputStream
import java.nio.*
import java.nio.charset.StandardCharsets
import java.util.stream.Collectors


class Utils {
    companion object {
        fun createBuffer(array: FloatArray): FloatBuffer = ByteBuffer
            .allocateDirect(array.size * Float.SIZE_BYTES)
            .order(ByteOrder.nativeOrder())
            .asFloatBuffer().apply {
                put(array)
                position(0)
            }


        fun inputStreamToString(inputStream: InputStream): String {
            return inputStream.bufferedReader(StandardCharsets.UTF_8).lines().collect(Collectors.joining("\n"))
        }
    }
}
```