**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи №1 з дисципліни

«Проєктування вбудованих систем»

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Головня Олександр Ростиславович*

**Перевірив**

(прізвище, ім'я, по батькові)

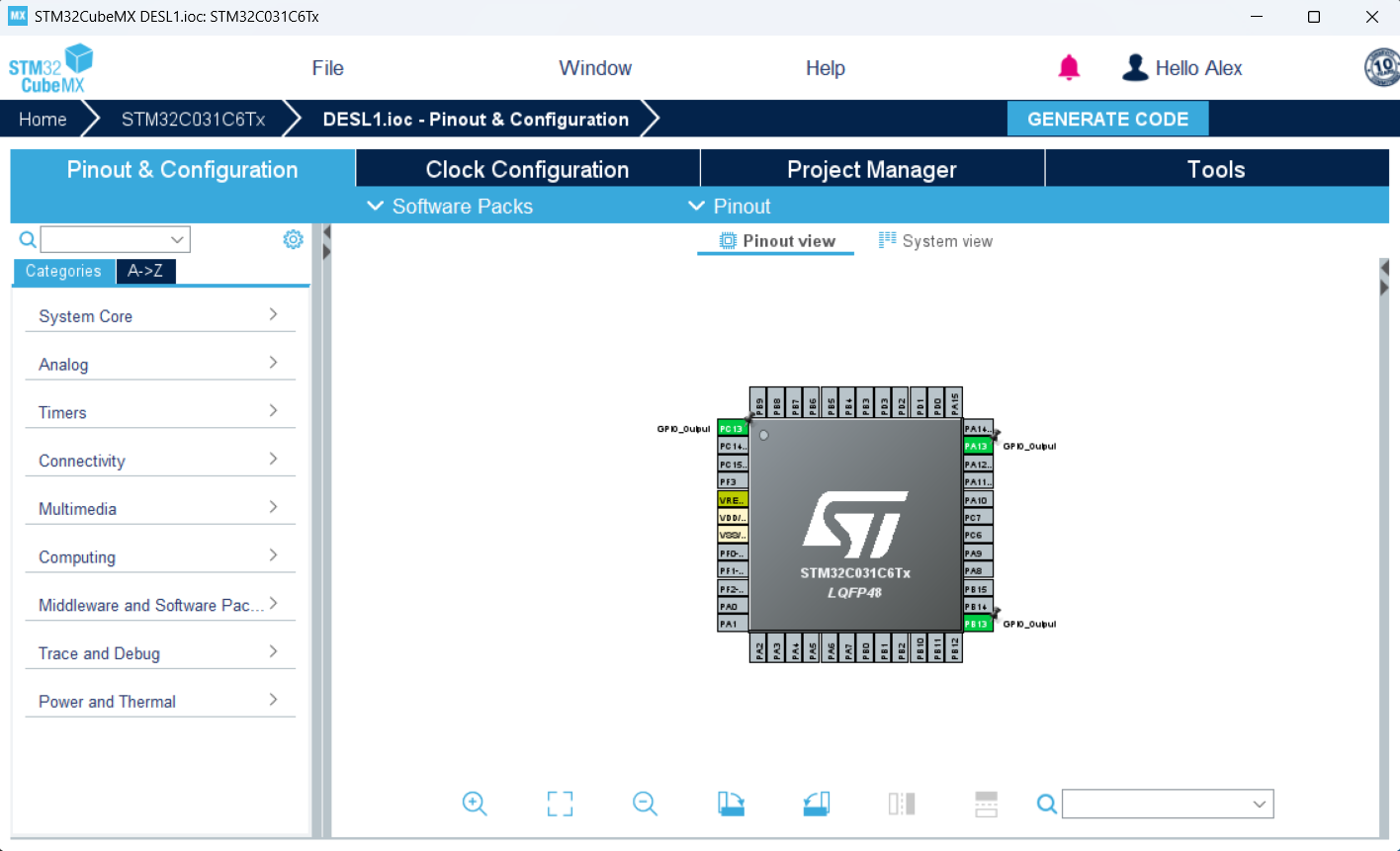
Київ 2025

**Лабораторна робота No1. Знайомство з базовими засобами комп’ютерної графіки деяких операційних платформ:**

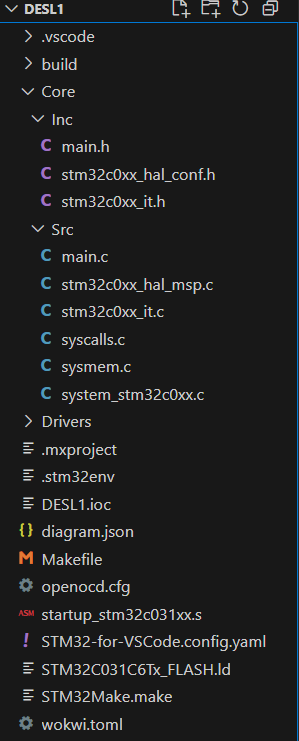
Мета: навчитись працювати з середовищем для розробки та проектування вбудованих систем на основі симулятора вбудованих систем Wokwi.

У цій лабораторній роботі виконані наступні кроки:

* Встановити VS code
* Встановити Wokwi в VS code - <https://docs.wokwi.com/vscode/getting-started> (сертифікат безкоштовний для non - commercial використання)
* Встановити Stm32cubemx та створити проєкт на основі STM32C031C6



* Відкрити проєкт у VS code



* Встановити наступний додаток, щоб легко компілювати проєкт - <https://marketplace.visualstudio.com/items?itemName=bmd.stm32-for-vscode>
* Створити та запустити програму, яка блимає світлодіодом (у звіті продемонструвати код та скріншоти симулятора з виконанням програми)
* Ознайомитись з <https://docs.wokwi.com/parts/>, розглянути у лабораторній роботі 2 модулі на вибір (описати для чого цей модуль можна використати, у яких сферах його можна застосувати, у поєднанні з якими іншими модулями він може бути корисним)

**Результати виконання коду:**

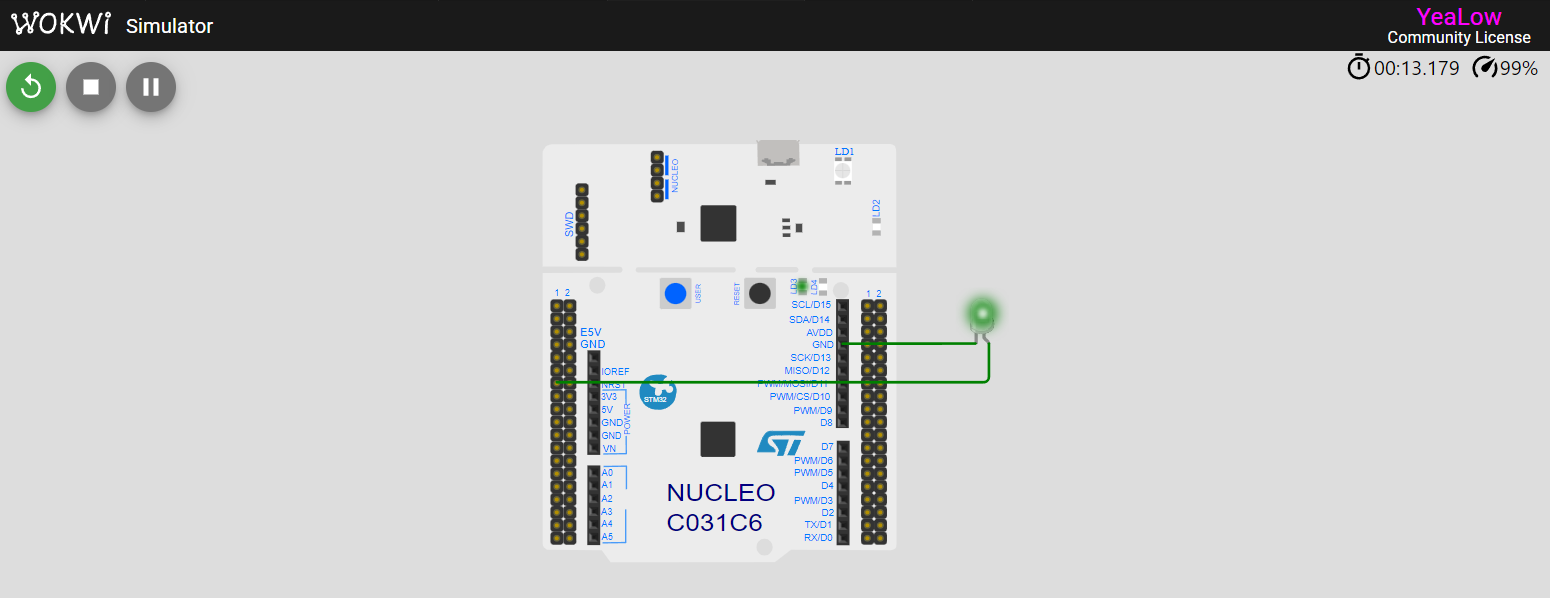
****

Рисунок 1. – Результат виконання програми в симуляторі

**Лістинг коду main.c**

#include "stm32c0xx.h"

void SystemClock\_Config(void);

void GPIO\_Init(void);

int main(void)

{

    // Ініціалізація системного годинника

    SystemClock\_Config();

    // Ініціалізація GPIO

    GPIO\_Init();

    // Основний цикл

    while (1)

    {

        // Включити світлодіод (встановлюємо високий рівень на пін PA13)

        GPIOA->ODR |= (1 << 13); // Встановлюємо біт 13 на 1 для включення світлодіода

        // Затримка

        for (volatile int i = 0; i < 100000; i++);

        // Вимкнути світлодіод (встановлюємо низький рівень на пін PA13)

        GPIOA->ODR &= ~(1 << 13); // Встановлюємо біт 13 на 0 для вимкнення світлодіода

        // Затримка

        for (volatile int i = 0; i < 100000; i++);

    }

}

void SystemClock\_Config(void)

{

    // Налаштування системного годинника (для простоти використовуємо стандартне налаштування)

}

void GPIO\_Init(void)

{

    // Увімкнути тактуючу лінію для порту A

    RCC->IOPENR |= RCC\_IOPENR\_GPIOAEN;

    // Налаштування піну PA13 як вихід

    GPIOA->MODER &= ~GPIO\_MODER\_MODE13;    // Очистити біт

    GPIOA->MODER |= GPIO\_MODER\_MODE13\_0;   // Налаштувати як вихід

    GPIOA->OTYPER &= ~GPIO\_OTYPER\_OT13;    // Налаштувати як вивід push-pull

    GPIOA->OSPEEDR &= ~GPIO\_OSPEEDR\_OSPEED13;  // Налаштувати на низьку швидкість

    GPIOA->PUPDR &= ~GPIO\_PUPDR\_PUPD13;   // Вимкнути підтягувачі

}

**Висновок:** У ході виконання лабораторної роботи було встановлено та налаштовано середовище розробки VS Code, а також додано емулятор Wokwi для роботи з мікроконтролерами. Була створена тестова програма для мікроконтролера STM32C031C6, яка здійснює миготіння світлодіода, а її виконання перевірено за допомогою симулятора.

Отримані результати дозволяють зробити висновок, що використання середовища Wokwi у поєднанні з VS Code спрощує розробку та тестування програм для мікроконтролерів, особливо на початкових етапах проектування. Це надає можливість швидко перевіряти працездатність коду без необхідності фізичного обладнання, що є зручним для навчальних і дослідницьких цілей.

**Додаток А (Вихідні тексти файлів \*.cpp, \*.java)**

**Lab1\_GDI**

// Holovnia\_Lab1.cpp : Defines the entry point for the application.

//

#include "framework.h"

#include "Holovnia\_Lab1.h"

#include <math.h>

#define MAX\_LOADSTRING 100

#define PI 3.1415

// Global Variables:

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

// Initialize global strings

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_HOLOVNIALAB1, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_HOLOVNIALAB1));

MSG msg;

// Main message loop:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// PURPOSE: Registers the window class.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_HOLOVNIALAB1));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_HOLOVNIALAB1);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// FUNCTION: InitInstance(HINSTANCE, int)

//

// PURPOSE: Saves instance handle and creates main window

//

// COMMENTS:

//

// In this function, we save the instance handle in a global variable and

// create and display the main program window.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: Processes messages for the main window.

//

// WM\_COMMAND - process the application menu

// WM\_PAINT - Paint the main window

// WM\_DESTROY - post a quit message and return

//

//

void DrawSun(HDC hdc, int x, int y, int radius, int N = 22) {//22

double angleStep = 2 \* PI / N;

POINT points[22]; // Масив точок для N-кутника (22)

// Створюємо чорний пензель і чорну обводку

HBRUSH hBrush = CreateSolidBrush(RGB(0, 0, 0)); // Заповнення чорним

HBRUSH oldBrush = (HBRUSH)SelectObject(hdc, hBrush);

// чорний пен для малювання

HPEN hPen = CreatePen(PS\_SOLID, 2, RGB(0, 0, 0));

HPEN oldPen = (HPEN)SelectObject(hdc, hPen);

// Визначаємо координати точок Nкутника

for (int i = 0; i < N; i++) {

double angle = i \* angleStep;

points[i].x = x + (int)(radius \* cos(angle));

points[i].y = y + (int)(radius \* sin(angle));

}

// Малюємо N-кутник

Polygon(hdc, points, N);

// Малюємо промені

for (int i = 0; i < N; i++) {

double angle = i \* angleStep;

int x1 = x + (int)(radius \* cos(angle));

int y1 = y + (int)(radius \* sin(angle));

int x2 = x + (int)((radius + 40) \* cos(angle)); // Довжина променя

int y2 = y + (int)((radius + 40) \* sin(angle));

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, x2, y2);

}

// Відновлюємо старий пен і видаляємо створений

SelectObject(hdc, oldPen);

DeleteObject(hPen);

}

void DrawTriangle(HDC hdc, int x, int y, int size)

{

POINT triangle[3] = {

{x, y - size},

{x - size, y + size},

{x + size, y + size}

};

HPEN hPen = CreatePen(PS\_SOLID, 2, RGB(0, 0, 0));

HBRUSH hBrush = CreateSolidBrush(RGB(0, 0, 0));

HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

Polygon(hdc, triangle, 3);

SelectObject(hdc, hOldPen);

SelectObject(hdc, hOldBrush);

DeleteObject(hPen);

DeleteObject(hBrush);

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Add any drawing code that uses hdc here...

RECT rect;

GetClientRect(hWnd, &rect);

// Змінюємо фон на світло-рожевий

HBRUSH hBrush = CreateSolidBrush(RGB(255, 182, 193));

FillRect(hdc, &rect, hBrush);

DeleteObject(hBrush);

int centerX = (rect.right - rect.left) / 2;

int centerY = (rect.bottom - rect.top) / 2;

DrawSun(hdc, centerX, centerY - 100, 50);

DrawTriangle(hdc, centerX + 100, centerY + 100, 50);

// END: Add any drawing code that uses hdc here...

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Message handler for about box.

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

**Lab1\_Canvas:**package com.example.holovnia\_lab1;

import android.content.Context;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.graphics.Path;

import android.util.AttributeSet;

import android.view.View;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(new CustomView(this));

}

public class CustomView extends View {

private Paint paint;

public CustomView(Context context) {

super(context);

init();

}

private void init() {

paint = new Paint();

paint.setAntiAlias(true);

setBackgroundColor(Color.rgb(255, 182, 193)); // Світло-рожевий

}

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

int width = getWidth();

int height = getHeight();

int centerX = width / 2;

int centerY = height / 2;

drawPolygon(canvas, centerX, centerY - 200, 100, 22);

drawSunRays(canvas, centerX, centerY - 200, 50, 150, 22);

drawTriangle(canvas, centerX + 200, centerY + 200, 100);

}

private void drawPolygon(Canvas canvas, int x, int y, int radius, int numSides) {

paint.setColor(Color.BLACK);

paint.setStyle(Paint.Style.FILL);

Path path = new Path();

for (int i = 0; i < numSides; i++) {

double angle = 2 \* Math.PI \* i / numSides;

int px = x + (int) (radius \* Math.cos(angle));

int py = y + (int) (radius \* Math.sin(angle));

if (i == 0) {

path.moveTo(px, py);

} else {

path.lineTo(px, py);

}

}

path.close();

canvas.drawPath(path, paint);

}

private void drawSunRays(Canvas canvas, int x, int y, int innerRadius, int outerRadius, int numRays) {

paint.setColor(Color.BLACK);

paint.setStrokeWidth(5);

for (int i = 0; i < numRays; i++) {

double angle = 2 \* Math.PI \* i / numRays;

int innerX = x + (int) (innerRadius \* Math.cos(angle));

int innerY = y + (int) (innerRadius \* Math.sin(angle));

int outerX = x + (int) (outerRadius \* Math.cos(angle));

int outerY = y + (int) (outerRadius \* Math.sin(angle));

canvas.drawLine(innerX, innerY, outerX, outerY, paint);

}

}

private void drawTriangle(Canvas canvas, int x, int y, int size) {

paint.setColor(Color.BLACK);

paint.setStyle(Paint.Style.FILL);

Path path = new Path();

path.moveTo(x, y - size);

path.lineTo(x - size, y + size);

path.lineTo(x + size, y + size);

path.close();

canvas.drawPath(path, paint);

}

}

}

**Lab1\_GL**package com.example.holovnia\_lab1gl;

import android.content.Context;

import android.opengl.GLES32;

import android.opengl.GLSurfaceView;

import android.os.Bundle;

import android.util.Log;

import androidx.appcompat.app.AppCompatActivity;

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 MainActivity extends AppCompatActivity {

private GLSurfaceView gLView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

gLView = new MyGLSurfaceView(this);

setContentView(gLView);

}

public class MyGLSurfaceView extends GLSurfaceView {

public MyGLSurfaceView(Context context) {

super(context);

setEGLContextClientVersion(3);

setRenderer(new MyGLRenderer());

setRenderMode(GLSurfaceView.RENDERMODE\_CONTINUOUSLY);

}

}

}

class MyGLRenderer implements GLSurfaceView.Renderer {

private FloatBuffer triangleBuffer, polygonBuffer, sunRaysBuffer;

private int program;

private int positionHandle;

private static final int COORDS\_PER\_VERTEX = 3;

private static final int vertexStride = COORDS\_PER\_VERTEX \* 4;

private final float[] triangleCoords = {

0.25f + 0.2f, -0.25f - 0.3f, 0.0f, // Правий низ

-0.25f + 0.2f, -0.25f - 0.3f, 0.0f, // Лівий низ

0.0f + 0.2f, 0.05f - 0.3f, 0.0f // вверх

};

private final float[] polygonCoords = generatePolygonCoords(22, 0.0f, 0.2f, 0.2f); // 22-кутник

private final float[] sunRaysCoords = generateSunRays(22, 0.0f, 0.2f, 0.3f); // Промені

private final String vertexShaderCode =

"#version 300 es\n" +

"layout(location = 0) in vec4 vPosition;\n" +

"void main() {\n" +

" gl\_Position = vPosition;\n" +

"}";

private final String fragmentShaderCode =

"#version 300 es\n" +

"precision mediump float;\n" +

"out vec4 fragColor;\n" +

"void main() {\n" +

" fragColor = vec4(0.0, 0.0, 0.0, 1.0);\n" + // Чорний

"}";

@Override

public void onSurfaceCreated(GL10 unused, EGLConfig config) {

GLES32.glClearColor(1.0f, 0.8f, 0.9f, 1.0f); // Світло-рожевий фон

GLES32.glEnable(GLES32.GL\_DEPTH\_TEST);

triangleBuffer = createFloatBuffer(triangleCoords);

polygonBuffer = createFloatBuffer(polygonCoords);

sunRaysBuffer = createFloatBuffer(sunRaysCoords);

int vertexShader = loadShader(GLES32.GL\_VERTEX\_SHADER, vertexShaderCode);

int fragmentShader = loadShader(GLES32.GL\_FRAGMENT\_SHADER, fragmentShaderCode);

program = GLES32.glCreateProgram();

GLES32.glAttachShader(program, vertexShader);

GLES32.glAttachShader(program, fragmentShader);

GLES32.glLinkProgram(program);

GLES32.glUseProgram(program);

positionHandle = GLES32.glGetAttribLocation(program, "vPosition");

// if (positionHandle < 0) {

// Log.e("OpenGL", "Error: vPosition not found in shader!");

// return;

// }

}

@Override

public void onDrawFrame(GL10 unused) {

GLES32.glClear(GLES32.GL\_COLOR\_BUFFER\_BIT | GLES32.GL\_DEPTH\_BUFFER\_BIT);

GLES32.glUseProgram(program);

// Малюємо 22-кутник (сонце)

GLES32.glEnableVertexAttribArray(positionHandle);

GLES32.glVertexAttribPointer(positionHandle, COORDS\_PER\_VERTEX, GLES32.GL\_FLOAT, false, vertexStride, polygonBuffer);

GLES32.glDrawArrays(GLES32.GL\_TRIANGLE\_FAN, 0, 22);// N = 22

GLES32.glDisableVertexAttribArray(positionHandle);

// Малюємо промені сонця

GLES32.glLineWidth(5.0f); // Товщина

GLES32.glEnableVertexAttribArray(positionHandle);

GLES32.glVertexAttribPointer(positionHandle, COORDS\_PER\_VERTEX, GLES32.GL\_FLOAT, false, vertexStride, sunRaysBuffer);

GLES32.glDrawArrays(GLES32.GL\_LINES, 0, 44); //44 = 22 лінії (промені)

GLES32.glDisableVertexAttribArray(positionHandle);

// Малюємо трикутник

GLES32.glEnableVertexAttribArray(positionHandle);

GLES32.glVertexAttribPointer(positionHandle, COORDS\_PER\_VERTEX, GLES32.GL\_FLOAT, false, vertexStride, triangleBuffer);

GLES32.glDrawArrays(GLES32.GL\_TRIANGLES, 0, 3);

GLES32.glDisableVertexAttribArray(positionHandle);

}

@Override

public void onSurfaceChanged(GL10 unused, int width, int height) {

GLES32.glViewport(0, 0, width, height);

}

private FloatBuffer createFloatBuffer(float[] array) {

ByteBuffer bb = ByteBuffer.allocateDirect(array.length \* 4);

bb.order(ByteOrder.nativeOrder());

FloatBuffer buffer = bb.asFloatBuffer();

buffer.put(array);

buffer.position(0);

return buffer;

}

private int loadShader(int type, String shaderCode) {

int shader = GLES32.glCreateShader(type);

GLES32.glShaderSource(shader, shaderCode);

GLES32.glCompileShader(shader);

int[] compileStatus = new int[1];

GLES32.glGetShaderiv(shader, GLES32.GL\_COMPILE\_STATUS, compileStatus, 0);

// if (compileStatus[0] == 0) {

// Log.e("OpenGL", "Shader compilation failed: " + GLES32.glGetShaderInfoLog(shader));

// GLES32.glDeleteShader(shader);

// return 0;

// }

return shader;

}

private float[] generatePolygonCoords(int sides, float centerX, float centerY, float radius) {

float[] coords = new float[sides \* 3];

double angleStep = 2 \* Math.PI / sides;

for (int i = 0; i < sides; i++) {

double angle = i \* angleStep;

coords[i \* 3] = centerX + (float) Math.cos(angle) \* radius;

coords[i \* 3 + 1] = centerY + (float) (Math.sin(angle) \* radius \* 0.45);//сплюснути по Y

coords[i \* 3 + 2] = 0.0f;

}

return coords;

}

private float[] generateSunRays(int sides, float centerX, float centerY, float radius) {

float[] coords = new float[sides \* 6]; // 2 координати (стартова та кінцева точка)

double angleStep = 2 \* Math.PI / sides;

for (int i = 0; i < sides; i++) {

double angle = i \* angleStep;

coords[i \* 6] = centerX; // Початок променя (центр)

coords[i \* 6 + 1] = centerY;

coords[i \* 6 + 2] = 0.0f; // Z-координата для плоскої сцени

coords[i \* 6 + 3] = centerX + (float) Math.cos(angle) \* radius; // Кінець променя

coords[i \* 6 + 4] = centerY + (float) (Math.sin(angle) \* radius \* 0.45); //сплюснути по Y;

coords[i \* 6 + 5] = 0.0f; // Z-координата

}

return coords;

}

}