МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт

З лабораторної роботи № 1 з дисципліни
«Програмування комп'ютерної графіки»

«Знайомство з базовими засобами комп’ютерної графіки деяких операційних платформ»

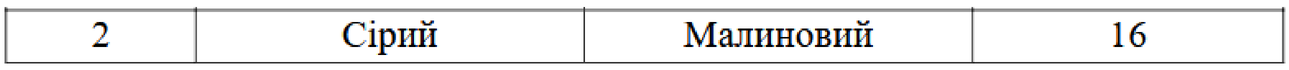
Виконав(ла)	*ІП-13 Бабіч Денис*	
	(шифр, прізвище, ім'я, по батькові)	
Перевірив(ла)	*Порєв В. М.*	
	(посада, прізвище, ім'я, по батькові)	

Київ 2025

ОСНОВНА ЧАСТИНА

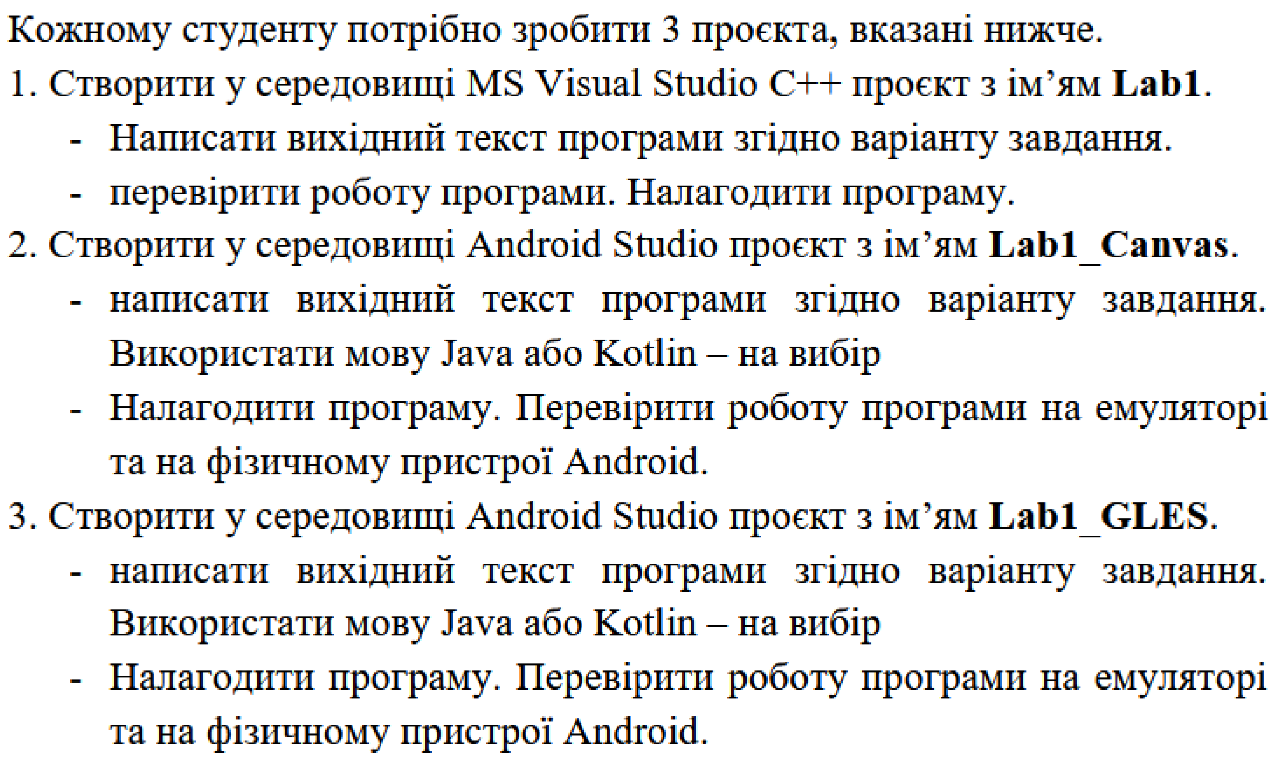
Мета роботи: Отримати перші навички створення програм то набути знання щодо базових засобів відображення графіки для різних операційних платформ.

Завдання:



2	Сірий	Малиновий	16

Рисунок 1.1 – Варіант індивідуального завдання (1402 % 8 = 2)



Кожному студенту потрібно зробити 3 проєкта, вказані нижче.

1. Створити у середовищі MS Visual Studio C++ проєкт з ім'ям **Lab1**.
   - Написати вихідний текст програми згідно варіанту завдання.
   - перевірити роботу програми. Налагодити програму.
2. Створити у середовищі Android Studio проєкт з ім'ям **Lab1_Canvas**.
   - написати вихідний текст програми згідно варіанту завдання. Використати мову Java або Kotlin – на вибір
   - Налагодити програму. Перевірити роботу програми на емуляторі та на фізичному пристрої Android.
3. Створити у середовищі Android Studio проєкт з ім'ям **Lab1_GLES**.
   - написати вихідний текст програми згідно варіанту завдання. Використати мову Java або Kotlin – на вибір
   - Налагодити програму. Перевірити роботу програми на емуляторі та на фізичному пристрої Android.

Рисунок 1.2 – Завдання лабораторного практикуму

1. Завдання 1.1 (GDI Windows)

main.c

```
#define _USE_MATH_DEFINES
#include <math.h>
#include "framework.h"
#include "WindowsGDI.h"

#define MAX_LOADSTRING 100
#define APPLICATION_WINDOW_WIDTH 400
#define APPLICATION_WINDOW_HEIGHT 600

#define PREFERENCES_SUN_OFFSET_Y 125;
#define PREFERENCES_PYRAMID_OFFSET_X 75;
#define PREFERENCES_PYRAMID_OFFSET_Y 100;
#define PREFERENCES_PEN_FOREGROUND_THICKNESS 2
#define PREFERENCES_COLOR_BACKGROUND RGB(27, 27, 27)
#define PREFERENCES_COLOR_FOREGROUND RGB(255, 20, 147)

#define SHAPE_LINE_LENGTH 90
#define SHAPE_TRIANGLE_SIDE_SIZE 70
#define SHAPE_TRIANGLE_VERTICES_COUNT 3
#define SHAPE_POLYGON_RADIUS 45
#define SHAPE_POLYGON_VERTICES_COUNT 16

HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];

RECT clientRect;
```

```
HPEN penForeground;
HBRUSH brushForeground;
HBRUSH brushBackground;

ATOM MyRegisterClass(HINSTANCE);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

void OnWmPaint(const HWND);
void HandleWmPaintDrawing(const HWND, const HDC);
void FillBackground(const HWND, const HDC, const HBRUSH);
void DrawTriangle(const HWND, const HDC, const HBRUSH);
void DrawPolygon(const HWND, const HDC, const HBRUSH);
void DrawRays(const HWND, const HDC, const HPEN);
inline POINT GetClientCenter(const RECT *const);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE
hPrevInstance, _In_ LPWSTR lpCmdLine, _In_ int nCmdShow)
{
	UNREFERENCED_PARAMETER(hPrevInstance);
	UNREFERENCED_PARAMETER(lpCmdLine);

	LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadStringW(hInstance, IDC_WINDOWSGDI, szWindowClass,
MAX_LOADSTRING);
	MyRegisterClass(hInstance);

	if (!InitInstance(hInstance, nCmdShow))
	return FALSE;
```

```
    HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_WINDOWSGDI));

    MSG msg;

    while (GetMessage(&msg, NULL, 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.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
```

```
        wcex.lpszMenuName = NULL;
        wcex.hbrBackground = NULL;
        wcex.hInstance = hInstance;
        wcex.lpfnWndProc = WndProc;
        wcex.lpszClassName = szWindowClass;
        wcex.style = CS_HREDRAW | CS_VREDRAW;
        wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
        wcex.hIcon = LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_WINDOWSGDI));
        wcex.hIconSm = LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));

        brushBackground =
CreateSolidBrush(PREFERENCES_COLOR_BACKGROUND);
        brushForeground =
CreateSolidBrush(PREFERENCES_COLOR_FOREGROUND);
        penForeground = CreatePen(PS_SOLID,
PREFERENCES_PEN_FOREGROUND_THICKNESS,
PREFERENCES_COLOR_FOREGROUND);

        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;

        DWORD windowStyles = WS_OVERLAPPED | WS_CAPTION |
WS_SYSMENU | WS_MINIMIZEBOX;
        HWND hWnd = CreateWindowW(szWindowClass, szTitle, windowStyles,
CW_USEDEFAULT, 0, APPLICATION_WINDOW_WIDTH,
APPLICATION_WINDOW_HEIGHT, NULL, NULL, hInstance, NULL);

        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_PAINT      - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
        switch (message)
        {
        case WM_PAINT:
        OnWmPaint(hWnd);
        break;
        case WM_DESTROY:
        DeleteObject(penForeground);
        DeleteObject(brushBackground);
        PostQuitMessage(EXIT_SUCCESS);
        break;
        default:
        return DefWindowProc(hWnd, message, wParam, lParam);
        }

        return EXIT_SUCCESS;
}

void __stdcall OnWmPaint(const HWND hWnd)
{
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hWnd, &ps);
        GetClientRect(hWnd, &clientRect);
        HandleWmPaintDrawing(hWnd, hdc);
        EndPaint(hWnd, &ps);
```

```
}

void __stdcall HandleWmPaintDrawing(const HWND hWnd, const HDC hdc)
{
	SelectObject(hdc, GetStockObject(NULL_PEN));
	SelectObject(hdc, GetStockObject(NULL_BRUSH));

	FillBackground(hWnd, hdc, brushBackground);
	DrawTriangle(hWnd, hdc, brushForeground);
	DrawPolygon(hWnd, hdc, brushForeground);
	DrawRays(hWnd, hdc, penForeground);
}

void __stdcall FillBackground(const HWND hWnd, const HDC hdc, const HBRUSH
brush)
{
	SelectObject(hdc, brush);
	FillRect(hdc, &clientRect, brush);
}

void __stdcall DrawTriangle(const HWND hWnd, const HDC hdc, const HBRUSH
brush)
{
	POINT center = GetClientCenter(&clientRect);
	POINT vertices[SHAPE_TRIANGLE_VERTICES_COUNT];

	center.x += PREFERENCES_PYRAMID_OFFSET_X;
	center.y += PREFERENCES_PYRAMID_OFFSET_Y;
```

```
        // Top vertex
        vertices[0].x = center.x;
        vertices[0].y = center.y - SHAPE_TRIANGLE_SIDE_SIZE;

        // Bottom left
        vertices[1].x = center.x - SHAPE_TRIANGLE_SIDE_SIZE;
        vertices[1].y = center.y + SHAPE_TRIANGLE_SIDE_SIZE;

        // Bottom right
        vertices[2].x = center.x + SHAPE_TRIANGLE_SIDE_SIZE;
        vertices[2].y = center.y + SHAPE_TRIANGLE_SIDE_SIZE;

        SelectObject(hdc, brush);
        Polygon(hdc, vertices, SHAPE_TRIANGLE_VERTICES_COUNT);
}

void __stdcall DrawPolygon(const HWND hWnd, const HDC hdc, const HBRUSH
brush)
{
        POINT center = GetClientCenter(&clientRect);
        center.y -= PREFERENCES_SUN_OFFSET_Y;

        POINT vertices[SHAPE_POLYGON_VERTICES_COUNT];

        for (size_t i = 0; i < SHAPE_POLYGON_VERTICES_COUNT; ++i)
        {
        vertices[i].x = SHAPE_POLYGON_RADIUS * cos(2 * M_PI * i /
SHAPE_POLYGON_VERTICES_COUNT) + center.x;
```

```
        vertices[i].y = SHAPE_POLYGON_RADIUS * sin(2 * M_PI * i /
SHAPE_POLYGON_VERTICES_COUNT) + center.y;
        }

        SelectObject(hdc, brush);
        Polygon(hdc, vertices, SHAPE_POLYGON_VERTICES_COUNT);
}

void __stdcall DrawRays(const HWND hWnd, const HDC hdc, const HPEN pen)
{
        float angle;
        int lineEndX;
        int lineEndY;

        POINT center = GetClientCenter(&clientRect);
        center.y -= PREFERENCES_SUN_OFFSET_Y;

        SelectObject(hdc, pen);

        for (size_t i = 0; i < SHAPE_POLYGON_VERTICES_COUNT; ++i)
        {
        angle = 2 * M_PI * i / SHAPE_POLYGON_VERTICES_COUNT;
        lineEndX = center.x + SHAPE_LINE_LENGTH * cos(angle);
        lineEndY = center.y + SHAPE_LINE_LENGTH * sin(angle);

        MoveToEx(hdc, center.x, center.y, NULL);
        LineTo(hdc, lineEndX, lineEndY);
        }
}
```

```
inline POINT GetClientCenter(const RECT *const clientRect)
{
	POINT center;
	center.x = (clientRect->right - clientRect->left) / 2;
	center.y = (clientRect->bottom - clientRect->top) / 2;
	return center;
}
```

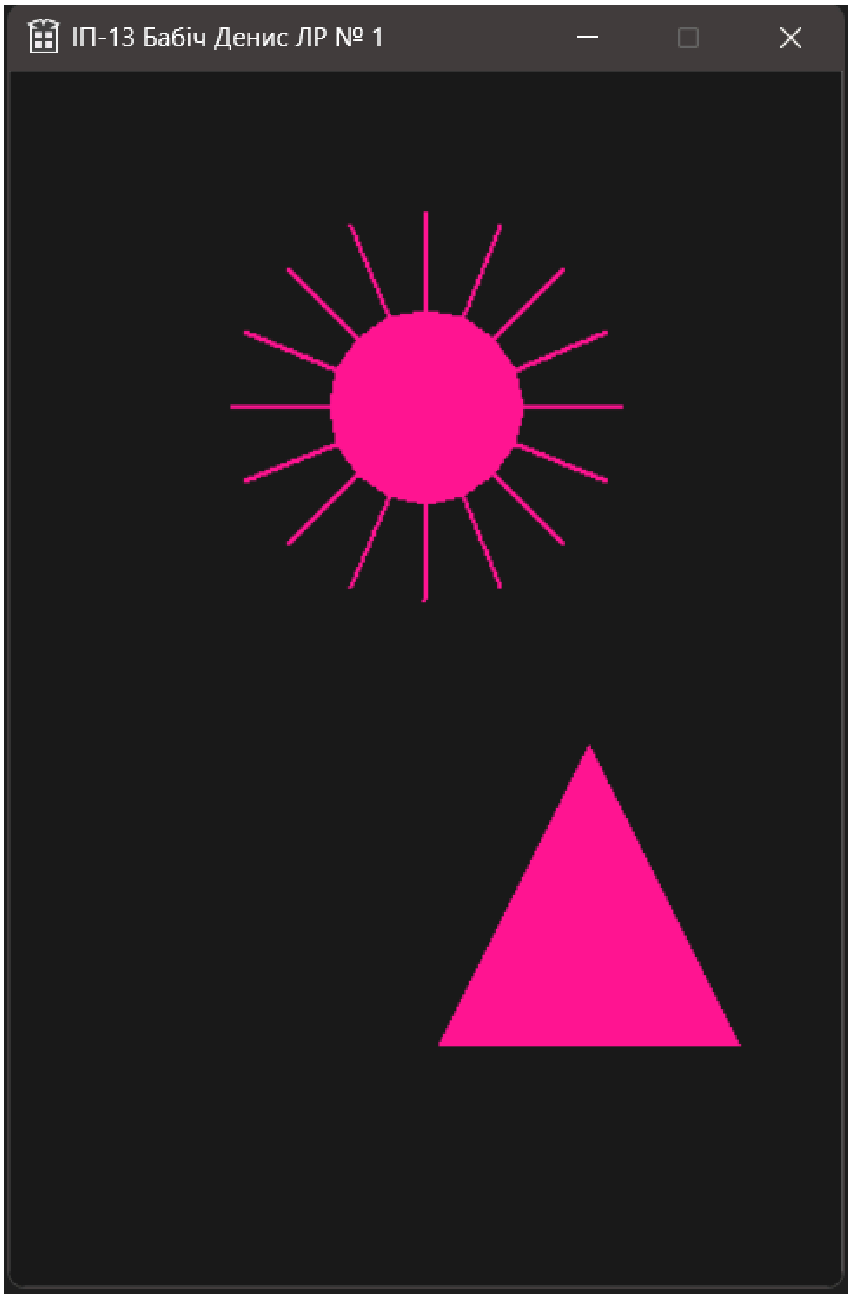


Рисунок 1.3 – Результат роботи за допомогою Windows GDI

2. Завдання 1.2 (Android Graphics Canvas)

CustomGraphicsView.java

```
package com.labwork.examplecanvas;

import android.view.View;
import android.graphics.Path;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Canvas;
import android.content.Context;

final class CustomGraphicsView extends View {

  private static final int PREFERENCES_SUN_OFFSET_Y = 600;
  private static final int PREFERENCES_PYRAMID_OFFSET_X = 200;
  private static final int PREFERENCES_PYRAMID_OFFSET_Y = 50;
  private static final float PREFERENCES_PEN_FOREGROUND_THICKNESS =
2f;
  private static final int PREFERENCES_COLOR_BACKGROUND = Color.rgb(27,
27, 27);
  private static final int PREFERENCES_COLOR_FOREGROUND =
Color.rgb(255, 20, 147);

  private static final int SHAPE_LINE_LENGTH = 225;
  private static final int SHAPE_TRIANGLE_SIDE_SIZE = 200;
  private static final int SHAPE_POLYGON_RADIUS = 125;
  private static final int SHAPE_POLYGON_VERTICES_COUNT = 16;

  private final Path polygonPath;
  private final Path trianglePath;

  private final Paint backgroundPaint;
  private final Paint foregroundPaint;

  public CustomGraphicsView(Context context) {
     super(context);

     this.polygonPath = new Path();
     this.trianglePath = new Path();

     this.backgroundPaint = new Paint();
     this.backgroundPaint.setStyle(Paint.Style.FILL);

this.backgroundPaint.setColor(CustomGraphicsView.PREFERENCES_COLOR_BA
CKGROUND);

     this.foregroundPaint = new Paint();
     this.foregroundPaint.setStyle(Paint.Style.FILL);

this.foregroundPaint.setColor(CustomGraphicsView.PREFERENCES_COLOR_FOR
EGROUND);

this.foregroundPaint.setStrokeWidth(CustomGraphicsView.PREFERENCES_PEN_F
OREGROUND_THICKNESS);
  }

  @Override
  protected final void onDraw(Canvas canvas) {
     if (canvas == null) {
       throw new IllegalArgumentException("canvas cannot be null");
     }

     super.onDraw(canvas);

     float centerX = getWidth() / 2f;
     float centerY = getHeight() / 2f;

                 canvas.drawRect(0, 0, super.getWidth(), super.getHeight(),
this.backgroundPaint);

     this.drawTriangle(canvas, centerX, centerY);
     this.drawPolygon(canvas, centerX, centerY);
     this.drawRays(canvas, centerX, centerY);
  }

  private void drawTriangle(Canvas canvas, float centerX, float centerY) {
     if (canvas == null) {
       throw new IllegalArgumentException("canvas cannot be null");
     }

     this.trianglePath.reset();

     centerX += CustomGraphicsView.PREFERENCES_PYRAMID_OFFSET_X;
     centerY += CustomGraphicsView.PREFERENCES_PYRAMID_OFFSET_Y;

     // Top vertex
```

```
                    trianglePath.moveTo(centerX,       centerY       -
CustomGraphicsView.SHAPE_TRIANGLE_SIDE_SIZE);
    // Bottom left
                              trianglePath.lineTo(centerX            -
CustomGraphicsView.SHAPE_TRIANGLE_SIDE_SIZE,
        centerY + CustomGraphicsView.SHAPE_TRIANGLE_SIDE_SIZE);
    // Bottom right
                              trianglePath.lineTo(centerX            +
CustomGraphicsView.SHAPE_TRIANGLE_SIDE_SIZE,
        centerY + CustomGraphicsView.SHAPE_TRIANGLE_SIDE_SIZE);

    trianglePath.close();

    canvas.drawPath(trianglePath, this.foregroundPaint);
  }

  private void drawPolygon(Canvas canvas, float centerX, float centerY) {
    if (canvas == null) {
      throw new IllegalArgumentException("canvas cannot be null");
    }

    this.polygonPath.reset();

    centerY -= CustomGraphicsView.PREFERENCES_SUN_OFFSET_Y;

                    for    (int    i    =    0;    i    <
CustomGraphicsView.SHAPE_POLYGON_VERTICES_COUNT; ++i) {
               float  angle  =  (float)  (2  *  Math.PI  *  i  /
CustomGraphicsView.SHAPE_POLYGON_VERTICES_COUNT);
```

```
            float x = (float) (CustomGraphicsView.SHAPE_POLYGON_RADIUS *
Math.cos(angle)) + centerX;
            float y = (float) (CustomGraphicsView.SHAPE_POLYGON_RADIUS *
Math.sin(angle)) + centerY;

        if (i == 0) {
            polygonPath.moveTo(x, y);
        }else {
            polygonPath.lineTo(x, y);
        }
    }

    polygonPath.close();

    canvas.drawPath(polygonPath, this.foregroundPaint);
  }

  private void drawRays(Canvas canvas, float centerX, float centerY) {
    if (canvas == null) {
        throw new IllegalArgumentException("canvas cannot be null");
    }

    centerY -= CustomGraphicsView.PREFERENCES_SUN_OFFSET_Y;

                    for (int i = 0; i <
CustomGraphicsView.SHAPE_POLYGON_VERTICES_COUNT; ++i) {
                float angle = (float) (2 * Math.PI * i /
CustomGraphicsView.SHAPE_POLYGON_VERTICES_COUNT);
```

```
                        float endX = (float) (centerX +
CustomGraphicsView.SHAPE_LINE_LENGTH * Math.cos(angle));
                        float endY = (float) (centerY +
CustomGraphicsView.SHAPE_LINE_LENGTH * Math.sin(angle));

            canvas.drawLine(centerX, centerY, endX, endY, this.foregroundPaint);
        }
    }
}
```

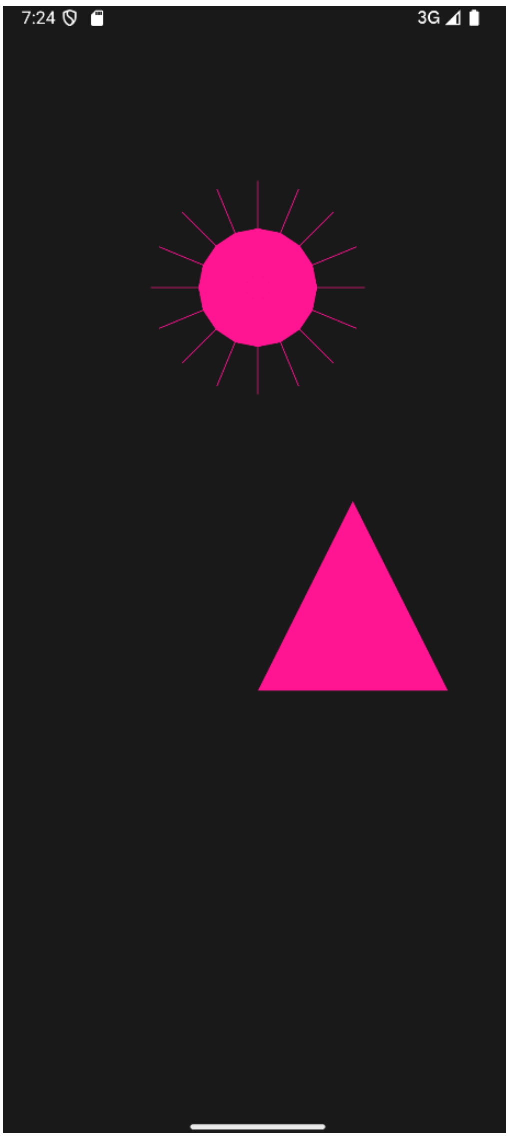


Рисунок 1.4 – Результат роботи Android Graphics Canvas

3. Завдання 1.1 (Android OpenGLES)

Component.java

```
package com.labwork.exampleopengles.core.components.common;

import com.labwork.exampleopengles.core.general.Entity;

public class Component {

  private static int nextId;

  private final int id;
  private final Entity entity;

  private boolean isActive;

  public Component(Entity entity) {
     this.entity = entity;
     this.id = ++Component.nextId;
  }

  public int getId() {
     return this.id;
  }

  public Entity getEntity() {
     return this.entity;
  }

  public boolean getIsActive() {
     return this.isActive;
  }

  public void setIsActive(boolean value) {
     this.isActive = value;
  }

  public void onStart() {}

  public void onUpdate() {}

  public void onDestroy() {}
}
```

CameraComponent.java

```
package com.labwork.exampleopengles.core.components.concrete;

import android.opengl.Matrix;
import com.labwork.exampleopengles.core.general.Color;
import com.labwork.exampleopengles.core.general.Entity;
import com.labwork.exampleopengles.core.components.common.Component;

public class CameraComponent extends Component {

    private static final int MATRIX_DIMENSIONS_COUNT = 16;

    protected final float[] matrixView;
    protected final float[] matrixProjection;

    protected Color backgroundColor;
```

```
  protected float farClippingPlane;
  protected float nearClippingPlane;

  public CameraComponent(Entity entity, Color color, float nearClippingPlane, float
farClippingPlane) {
     super(entity);
     this.backgroundColor = color;
     this.farClippingPlane = farClippingPlane;
     this.nearClippingPlane = nearClippingPlane;
                                         this.matrixView         =         new
float[CameraComponent.MATRIX_DIMENSIONS_COUNT];
                                     this.matrixProjection        =        new
float[CameraComponent.MATRIX_DIMENSIONS_COUNT];
     Matrix.setIdentityM(this.matrixView, 0);
     Matrix.setIdentityM(this.matrixProjection, 0);
  }

  public float[] getMatrixView() {
     return this.matrixView;
  }

  public float[] getMatrixProjection() {
     return this.matrixProjection;
  }

  public Color getBackgroundColor() {
     return this.backgroundColor;
  }
  public void setBackgroundColor(Color value) {
    this.backgroundColor = value;
  }

  public float getFarClippingPlane() {
    return this.farClippingPlane;
  }

  public void setFarClippingPlane(float value) {
    this.farClippingPlane = value;
  }

  public float getNearClippingPlane() {
    return this.nearClippingPlane;
  }

  public void setNearClippingPlane(float value) {
    this.nearClippingPlane = value;
  }
}
```

CameraOrthographicComponent.java

```
package com.labwork.exampleopengles.core.components.concrete;

import android.opengl.GLES32;
import android.opengl.Matrix;
import com.labwork.exampleopengles.core.general.Color;
import com.labwork.exampleopengles.core.general.Entity;
import com.labwork.exampleopengles.core.general.Vector3;
```

```
public final class CameraOrthographicComponent extends CameraComponent {

  private final Vector3 target;

  private Vector3 up;
  private Vector3 position;
  private TransformComponent transform;
  private float left, right, bottom, top;

    public CameraOrthographicComponent(Entity entity, Color color, float
nearClippingPlane, float farClippingPlane, float left, float right, float bottom, float
top) {
    super(entity, color, nearClippingPlane, farClippingPlane);
    this.top = top;
    this.left = left;
    this.right = right;
    this.bottom = bottom;
    this.up = new Vector3(0.0f, 1.0f, 0.0f);
    this.target = new Vector3(0.0f, 0.0f, -1.0f);
    this.position = new Vector3(0.0f, 0.0f, 0.0f);
  }

  public float getTop() {
    return top;
  }

  public float getLeft() {
    return left;
  }
  public float getRight() {
    return right;
  }

  public float getBottom() {
    return bottom;
  }

  public void setBounds(float left, float right, float bottom, float top) {
    this.top = top;
    this.left = left;
    this.right = right;
    this.bottom = bottom;
        Matrix.orthoM(super.matrixProjection, 0, left, right, bottom, top,
super.nearClippingPlane, super.farClippingPlane);
  }

  @Override
  public void onStart() {
    this.transform = super.getEntity().getComponent(TransformComponent.class);
    this.up = this.transform.getUp();
    this.position = this.transform.getPosition();
      Matrix.orthoM(super.matrixProjection, 0, this.left, this.right, this.bottom,
this.top, super.nearClippingPlane, super.farClippingPlane);
                    GLES32.glClearColor(super.backgroundColor.getR(),
super.backgroundColor.getG(),                    super.backgroundColor.getB(),
super.backgroundColor.getA());
  }
  @Override
  public void onUpdate() {
        Vector3.add(this.transform.getPosition(), this.transform.getForward(), this.target);
        Matrix.orthoM(super.matrixProjection, 0, this.left, this.right, this.bottom, this.top, super.nearClippingPlane, super.farClippingPlane);
        Matrix.setLookAtM(super.matrixView, 0, this.position.getX(), this.position.getY(), this.position.getZ(), this.target.getX(), this.target.getY(), this.target.getZ(), this.up.getX(), this.up.getY(), this.up.getZ());
  }
}
```

CameraPerspectiveComponent.java

```
package com.labwork.exampleopengles.core.components.concrete;

import android.opengl.GLES32;
import android.opengl.Matrix;
import com.labwork.exampleopengles.core.general.Color;
import com.labwork.exampleopengles.core.general.Entity;
import com.labwork.exampleopengles.core.general.Vector3;

public final class CameraPerspectiveComponent extends CameraComponent {

  private final Vector3 target;

  private Vector3 up;
  private Vector3 position;
  private float aspectRatio;
  private float fieldOfView;
  private TransformComponent transform;

      public CameraPerspectiveComponent(Entity entity, Color color, float nearClippingPlane, float farClippingPlane, float aspectRatio, float fieldOfView) {
    super(entity, color, nearClippingPlane, farClippingPlane);
    this.fieldOfView = fieldOfView;
    this.aspectRatio = aspectRatio;
    this.up = new Vector3(0.0f, 1.0f, 0.0f);
    this.target = new Vector3(0.0f, 0.0f, -1.0f);
    this.position = new Vector3(0.0f, 0.0f, 0.0f);
  }

  public float getAspectRatio() {
    return this.aspectRatio;
  }

  public void setAspectRatio(float value) {
    this.aspectRatio = value;
            Matrix.perspectiveM(super.matrixProjection, 0, this.fieldOfView, this.aspectRatio, super.nearClippingPlane, super.farClippingPlane);
  }

  public float getFieldOfView() {
    return this.fieldOfView;
  }

  public void setFieldOfView(float value) {
    this.fieldOfView = value;
```

```
        Matrix.perspectiveM(super.matrixProjection, 0, this.fieldOfView, this.aspectRatio, super.nearClippingPlane, super.farClippingPlane);
  }

  @Override
  public void onStart() {
    this.transform = super.getEntity().getComponent(TransformComponent.class);
    this.up = this.transform.getUp();
    this.position = this.transform.getPosition();
        Matrix.perspectiveM(super.matrixProjection, 0, this.fieldOfView, this.aspectRatio, super.nearClippingPlane, super.farClippingPlane);
        GLES32.glClearColor(super.backgroundColor.getR(), super.backgroundColor.getG(), super.backgroundColor.getB(), super.backgroundColor.getA());
  }

  @Override
  public void onUpdate() {
        Vector3.add(this.transform.getPosition(), this.transform.getForward(), this.target);
        Matrix.perspectiveM(super.matrixProjection, 0, this.fieldOfView, this.aspectRatio, super.nearClippingPlane, super.farClippingPlane);
        Matrix.setLookAtM(super.matrixView, 0, this.position.getX(), this.position.getY(), this.position.getZ(), this.target.getX(), this.target.getY(), this.target.getZ(), this.up.getX(), this.up.getY(), this.up.getZ());
  }
}
```

RenderingComponent.java

```
package com.labwork.exampleopengles.core.components.concrete;

import android.opengl.GLES32;
import android.opengl.Matrix;
import com.labwork.exampleopengles.runtime.Framework;
import com.labwork.exampleopengles.core.general.Mesh;
import com.labwork.exampleopengles.core.general.Entity;
import com.labwork.exampleopengles.core.general.Shader;
import com.labwork.exampleopengles.core.general.Material;
import com.labwork.exampleopengles.core.components.common.Component;

public final class RenderingComponent extends Component {

  private static final int MATRIX_DIMENSIONS_COUNT = 16;

  private final float[] matrixViewModel;
  private final float[] matrixProjectionViewModel;

  private Mesh mesh;
  private Material material;
  private TransformComponent transform;

  public RenderingComponent(Entity entity, Mesh mesh, Material material) {
     super(entity);
     this.mesh = mesh;
     this.material = material;
                                   this.matrixViewModel        =        new
float[RenderingComponent.MATRIX_DIMENSIONS_COUNT];
```

```
        this.matrixProjectionViewModel = new float[RenderingComponent.MATRIX_DIMENSIONS_COUNT];
  }

  public Mesh getMesh() {
    return this.mesh;
  }

  public Material getMaterial() {
    return this.material;
  }

  @Override
  public void onStart() {
    this.transform = super.getEntity().getComponent(TransformComponent.class);
  }

  public void render(Class<?> renderPass) {
    Shader shader = this.material.getShader(renderPass);
    GLES32.glUseProgram(shader.getProgramId());

    CameraComponent camera = Framework.getInstance().getScene().getCamera();

        Matrix.multiplyMM(this.matrixViewModel, 0, camera.getMatrixView(), 0, this.transform.getMatrixModel(), 0);
        Matrix.multiplyMM(this.matrixProjectionViewModel, 0, camera.getMatrixProjection(), 0, this.matrixViewModel, 0);
```

```
int handlerUMatrixMVP = GLES32.glGetUniformLocation(shader.getProgramId(), "uMatrixMVP");
GLES32.glUniformMatrix4fv(handlerUMatrixMVP, 1, false, this.matrixProjectionViewModel, 0);

int handlerUColorBase = GLES32.glGetUniformLocation(shader.getProgramId(), "uColorBase");
GLES32.glUniform4f(handlerUColorBase, this.material.getBase().getR(), this.material.getBase().getG(), this.material.getBase().getB(), this.material.getBase().getA());

int handlerInVertexPosition = GLES32.glGetAttribLocation(shader.getProgramId(), "inVertexPosition");
GLES32.glVertexAttribPointer(handlerInVertexPosition, Mesh.PAYLOAD_VERTEX_POSITION_SIZE, GLES32.GL_FLOAT, false, Mesh.PAYLOAD_STRIDE, Mesh.PAYLOAD_VERTEX_POSITION_OFFSET);
    GLES32.glEnableVertexAttribArray(handlerInVertexPosition);

    this.mesh.draw();
    GLES32.glUseProgram(0);
  }
}
```

TransformComponent.java

```
package com.labwork.exampleopengles.core.components.concrete;

import android.opengl.Matrix;
import com.labwork.exampleopengles.core.general.Axis;
import com.labwork.exampleopengles.core.general.Entity;
import com.labwork.exampleopengles.core.general.Vector3;
import com.labwork.exampleopengles.core.components.common.Component;

public final class TransformComponent extends Component {

  private static final int MATRIX_OUTPUT_DIMENSIONS_COUNT = 16;
  private static final int MATRIX_INTERMEDIATE_DIMENSIONS_COUNT = 4;

  private static final float[] MATRIX_VECTOR_UP = { 0.0f, 1.0f, 0.0f, 0.0f };
  private static final float[] MATRIX_VECTOR_RIGHT = { 1.0f, 0.0f, 0.0f, 0.0f };
   private static final float[] MATRIX_VECTOR_FORWARD = { 0.0f, 0.0f, -1.0f,
0.0f };

  private final Vector3 scale;
  private final Vector3 rotation;
  private final Vector3 position;
  private final Vector3 vectorUp;
  private final Vector3 vectorRight;
  private final Vector3 vectorForward;

  private final float[] matrixModel;
  private final float[] matrixRotation;
  private final float[] matrixRotationOutput;

  public TransformComponent(Entity entity) {
    super(entity);
                                        this.matrixModel        =        new
float[TransformComponent.MATRIX_OUTPUT_DIMENSIONS_COUNT];
                                       this.matrixRotation        =        new
float[TransformComponent.MATRIX_OUTPUT_DIMENSIONS_COUNT];
```

```
                                        this.matrixRotationOutput          =          new
float[TransformComponent.MATRIX_INTERMEDIATE_DIMENSIONS_COUNT];
        this.scale = new Vector3(1.0f, 1.0f, 1.0f);
        this.rotation = new Vector3(0.0f, 0.0f, 0.0f);
        this.position = new Vector3(0.0f, 0.0f, 0.0f);
        this.vectorUp = new Vector3(0.0f, 0.0f, 0.0f);
        this.vectorRight = new Vector3(0.0f, 0.0f, 0.0f);
        this.vectorForward = new Vector3(0.0f, 0.0f, 0.0f);
    }

    public Vector3 getScale() {
        return this.scale;
    }

    public Vector3 getRotation() {
        return this.rotation;
    }

    public Vector3 getPosition() {
        return this.position;
    }

    public float[] getMatrixModel() {
        Matrix.setIdentityM(this.matrixModel, 0);
                Matrix.scaleM(this.matrixModel,  0,  this.scale.getX(),  this.scale.getY(),
this.scale.getZ());
        Matrix.rotateM(this.matrixModel, 0, this.rotation.getX(), 1.0f, 0.0f, 0.0f);
        Matrix.rotateM(this.matrixModel, 0, this.rotation.getY(), 0.0f, 1.0f, 0.0f);
        Matrix.rotateM(this.matrixModel, 0, this.rotation.getZ(), 0.0f, 0.0f, 1.0f);
```

```
         Matrix.translateM(this.matrixModel, 0, this.position.getX(), this.position.getY(),
this.position.getZ());
        return this.matrixModel;
    }

    public Vector3 getUp() {
          Matrix.multiplyMV(this.matrixRotationOutput, 0, this.getRotationMatrix(), 0,
TransformComponent.MATRIX_VECTOR_UP, 0);
        this.vectorUp.setX(this.matrixRotationOutput[Axis.X.ordinal()]);
        this.vectorUp.setY(this.matrixRotationOutput[Axis.Y.ordinal()]);
        this.vectorUp.setZ(this.matrixRotationOutput[Axis.Z.ordinal()]);
        return this.vectorUp;
    }

    public Vector3 getRight() {
          Matrix.multiplyMV(this.matrixRotationOutput, 0, this.getRotationMatrix(), 0,
TransformComponent.MATRIX_VECTOR_RIGHT, 0);
        this.vectorRight.setX(this.matrixRotationOutput[Axis.X.ordinal()]);
        this.vectorRight.setY(this.matrixRotationOutput[Axis.Y.ordinal()]);
        this.vectorRight.setZ(this.matrixRotationOutput[Axis.Z.ordinal()]);
        return this.vectorRight;
    }

    public Vector3 getForward() {
          Matrix.multiplyMV(this.matrixRotationOutput, 0, this.getRotationMatrix(), 0,
TransformComponent.MATRIX_VECTOR_FORWARD, 0);
        this.vectorForward.setX(this.matrixRotationOutput[Axis.X.ordinal()]);
        this.vectorForward.setY(this.matrixRotationOutput[Axis.Y.ordinal()]);
        this.vectorForward.setZ(this.matrixRotationOutput[Axis.Z.ordinal()]);
        return this.vectorForward;
    }

    private float[] getRotationMatrix() {
        Matrix.setIdentityM(this.matrixRotation, 0);
        Matrix.rotateM(this.matrixRotation, 0, this.rotation.getX(), 1.0f, 0.0f, 0.0f);
        Matrix.rotateM(this.matrixRotation, 0, this.rotation.getY(), 0.0f, 1.0f, 0.0f);
        Matrix.rotateM(this.matrixRotation, 0, this.rotation.getZ(), 0.0f, 0.0f, 1.0f);
        return this.matrixRotation;
    }
}
```

Axis.java

```
package com.labwork.exampleopengles.core.general;

public enum Axis {
    X,
    Y,
    Z,
}
```

Color.java

```
package com.labwork.exampleopengles.core.general;

public final class Color {

    private static final float MAX_FLOAT_VALUE = 255.0f;

    private int r;
    private int g;
    private int b;
    private int a;

    public Color(int r, int g, int b, int a) {
       this.r = r;
       this.g = g;
       this.b = b;
       this.a = a;
    }

    public float getR() {
       return this.r / Color.MAX_FLOAT_VALUE;
    }

    public void setR(int value) {
       this.r = value;
    }

    public float getG() {
       return this.g / Color.MAX_FLOAT_VALUE;
    }

    public void setG(int value) {
       this.g = value;
    }

    public float getB() {
       return this.b / Color.MAX_FLOAT_VALUE;
    }
```

```
  public void setB(int value) {
    this.b = value;
  }

  public float getA() {
    return this.a / Color.MAX_FLOAT_VALUE;
  }

  public void setA(int value) {
    this.a = value;
  }
}
```

Entity.java

```
package com.labwork.exampleopengles.core.general;

import java.util.Map;
import java.util.HashMap;
import java.util.Collection;
import com.labwork.exampleopengles.core.components.common.Component;

public class Entity {

  private static int nextId;

  private final int id;
  private final Map<Class<?>, Component> components;

  private boolean isActive;
  public Entity() {
    this.isActive = true;
    this.id = ++Entity.nextId;
    this.components = new HashMap<>();
  }

  public int getId() {
    return this.id;
  }

  public boolean getIsActive() {
    return this.isActive;
  }

  public void setIsActive(boolean value) {
    this.isActive = value;
  }

  public Collection<Component> getComponents() {
    return this.components.values();
  }

  public void addComponent(Component component) {
    if (this.components.containsKey(component.getClass()))
                throw new IllegalArgumentException("Component of type " +
component.getClass().getName() + " already exists.");

    this.components.put(component.getClass(), component);
  }

  public boolean hasComponent(Class<?> component) {
    return this.components.containsKey(component);
  }

  @SuppressWarnings("unchecked")
  public <T extends Component> T getComponent(Class<T> component) {
    return (T) this.components.getOrDefault(component, null);
  }

  public void onStart() {
    for (Component component : this.components.values())
        component.onStart();
  }

  public void onUpdate() {
    for (Component component : this.components.values())
        component.onUpdate();
  }

  public void onDestroy() {
    for (Component component : this.components.values())
        component.onDestroy();
  }
}
```

Material.java

```
package com.labwork.exampleopengles.core.general;

import java.util.Map;
import java.util.HashMap;

public final class Material {

  private Color base;
  private final Map<Class<?>, Shader> shaders;

  public Material(Color base, Shader... shaders) {
    this.base = base;
    this.shaders = new HashMap<>();

    for (Shader shader : shaders)
      this.shaders.put(shader.getRenderPass(), shader);
  }

  public Color getBase() {
    return this.base;
  }

  public Shader getShader(Class<?> renderPass) {
    return this.shaders.getOrDefault(renderPass, null);
  }
}
```

Mesh.java

```
package com.labwork.exampleopengles.core.general;

import java.nio.ByteOrder;
import java.nio.ByteBuffer;
import java.nio.FloatBuffer;
import android.opengl.GLES32;

public final class Mesh {

  private static int BINDING_HANDLERS_COUNT = 2;
  private static int BINDING_HANDLER_INDEX_VAO = 0;
  private static int BINDING_HANDLER_INDEX_VBO = 1;
  private static final int DIMENSIONS_COUNT = 3;
  private static final int PAYLOAD_VERTEX_POSITION_INDEX = 0;

  public static final int PAYLOAD_VERTEX_POSITION_SIZE = 3;
  public static final int PAYLOAD_VERTEX_POSITION_OFFSET = 0;
  public static final int PAYLOAD_STRIDE = Mesh.PAYLOAD_VERTEX_POSITION_SIZE * Float.BYTES;

  private final int drawingMode;
  private final int verticesCount;
  private final float[] verticesData;
  private final int[] bindingHandlers;

  public Mesh(float[] verticesData, int drawingMode) {
    this.drawingMode = drawingMode;
    this.verticesData = verticesData;
```

```
      this.bindingHandlers = new int[Mesh.BINDING_HANDLERS_COUNT];
      this.verticesCount = verticesData.length / Mesh.DIMENSIONS_COUNT;

       FloatBuffer vertexBuffer = ByteBuffer.allocateDirect(this.verticesData.length *
Float.BYTES).order(ByteOrder.nativeOrder()).asFloatBuffer();
      vertexBuffer.put(this.verticesData).position(0);

                              GLES32.glGenVertexArrays(1,      this.bindingHandlers,
Mesh.BINDING_HANDLER_INDEX_VAO);
                                   GLES32.glGenBuffers(1,      this.bindingHandlers,
Mesh.BINDING_HANDLER_INDEX_VBO);


GLES32.glBindVertexArray(this.bindingHandlers[Mesh.BINDING_HANDLER_IN
DEX_VAO]);
                              GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER,
this.bindingHandlers[Mesh.BINDING_HANDLER_INDEX_VBO]);
      GLES32.glBufferData(GLES32.GL_ARRAY_BUFFER, this.verticesData.length
* Float.BYTES, vertexBuffer, GLES32.GL_STATIC_DRAW);



GLES32.glVertexAttribPointer(Mesh.PAYLOAD_VERTEX_POSITION_INDEX,
Mesh.PAYLOAD_VERTEX_POSITION_SIZE,     GLES32.GL_FLOAT,     false,
Mesh.PAYLOAD_STRIDE, Mesh.PAYLOAD_VERTEX_POSITION_OFFSET);

GLES32.glEnableVertexAttribArray(Mesh.PAYLOAD_VERTEX_POSITION_INDE
X);

      GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER, 0);
```

```
        GLES32.glBindVertexArray(0);
    }

    public void draw() {

GLES32.glBindVertexArray(this.bindingHandlers[Mesh.BINDING_HANDLER_IN
DEX_VAO]);
        GLES32.glDrawArrays(this.drawingMode, 0, this.verticesCount);
        GLES32.glBindVertexArray(0);
    }

    public void delete() {
        GLES32.glDeleteBuffers(this.bindingHandlers.length, this.bindingHandlers, 0);
    }
}
```

Scene.java

```
package com.labwork.exampleopengles.core.general;

import java.util.List;
import java.util.ArrayList;
import java.util.Collection;
import com.labwork.exampleopengles.core.components.common.Component;
import com.labwork.exampleopengles.core.components.concrete.CameraComponent;

public final class Scene {

    private final List<Entity> entities;

    private CameraComponent camera;
    public Scene() {
        this.entities = new ArrayList<>();
    }

    public List<Entity> getEntities() {
        return this.entities;
    }

    public CameraComponent getCamera() {
        return this.camera;
    }

    public void addEntity(Entity entity) {
        this.entities.add(entity);

        Collection<Component> components = entity.getComponents();

        for (Component component : components) {
            if (component instanceof CameraComponent) {
                this.camera = (CameraComponent) component;
            }
        }
    }
}
```

Shader.java

```
package com.labwork.exampleopengles.core.general;
```

```
import android.opengl.GLES32;

public final class Shader {

  private final int vertId;
  private final int fragId;
  private final int programId;
  private final Class<?> renderPass;

  public Shader(Class<?> renderPass, String sourceVert, String sourceFrag) {
     this.renderPass = renderPass;
     this.programId = GLES32.glCreateProgram();

     this.vertId = GLES32.glCreateShader(GLES32.GL_VERTEX_SHADER);
     GLES32.glShaderSource(this.vertId, sourceVert);

     this.fragId = GLES32.glCreateShader(GLES32.GL_FRAGMENT_SHADER);
     GLES32.glShaderSource(this.fragId, sourceFrag);
  }

  public int getProgramId() {
     return this.programId;
  }

  public Class<?> getRenderPass() {
     return this.renderPass;
  }
  public void compile() {
     GLES32.glCompileShader(this.vertId);
     GLES32.glCompileShader(this.fragId);
     GLES32.glAttachShader(this.programId, this.vertId);
     GLES32.glAttachShader(this.programId, this.fragId);
     GLES32.glLinkProgram(this.programId);
  }

  public void delete() {
     GLES32.glDetachShader(this.programId, this.vertId);
     GLES32.glDetachShader(this.programId, this.fragId);
     GLES32.glDeleteShader(this.vertId);
     GLES32.glDeleteShader(this.fragId);
     GLES32.glDeleteProgram(this.programId);
  }
}
```

Vector3.java

```
package com.labwork.exampleopengles.core.general;

public final class Vector3 {

  private float x;
  private float y;
  private float z;

  public Vector3(float x, float y, float z) {
     this.x = x;
     this.y = y;
     this.z = z;
```

```
}

public float getX() { return this.x; }
public void setX(float value) { this.x = value; }

public float getY() { return this.y; }
public void setY(float value) { this.y = value; }

public float getZ() { return this.z; }
public void setZ(float value) { this.z = value; }

public float magnitude() {
   return (float) Math.sqrt(x * x + y * y + z * z);
}

public static float dot(Vector3 a, Vector3 b) {
   return a.x * b.x + a.y * b.y + a.z * b.z;
}

public static void add(Vector3 a, Vector3 b, Vector3 output) {
   output.x = a.x + b.x;
   output.y = a.y + b.y;
   output.z = a.z + b.z;
}

public static void subtract(Vector3 a, Vector3 b, Vector3 output) {
   output.x = a.x - b.x;
   output.y = a.y - b.y;
   output.z = a.z - b.z;
```

```
    }

    public static void multiply(Vector3 a, float scalar, Vector3 output) {
        output.x = a.x * scalar;
        output.y = a.y * scalar;
        output.z = a.z * scalar;
    }

    public static void cross(Vector3 a, Vector3 b, Vector3 output) {
        output.x = a.y * b.z - a.z * b.y;
        output.y = a.z * b.x - a.x * b.z;
        output.z = a.x * b.y - a.y * b.x;
    }

    public static void normalize(Vector3 a, Vector3 output) {
        float magnitude = (float) Math.sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
        if (magnitude == 0) {
            output.x = 0;
            output.y = 0;
            output.z = 0;
        } else {
            output.x = a.x / magnitude;
            output.y = a.y / magnitude;
            output.z = a.z / magnitude;
        }
    }
}
```

Standalone.java

```
package com.labwork.exampleopengles.demo;
```

```
public final class Standalone {

  public static final String SHADER_VERT_SOURCE =
        "#version 300 es\n" +
        "in vec3 inVertexPosition;\n" +
        "uniform mat4 uMatrixMVP;\n" +
        "void main() {\n" +
        "   gl_Position = uMatrixMVP * vec4(inVertexPosition, 1.0);\n" +
        "}\n";

  public static final String SHADER_FRAG_SOURCE =
        "#version 300 es\n" +
        "precision mediump float;\n" +
        "uniform vec4 uColorBase;\n" +
        "out vec4 outColorBase;\n" +
        "void main() {\n" +
        "   outColorBase = uColorBase;\n" +
        "}\n";
}
```

RenderPass.java

```
package com.labwork.exampleopengles.rendering.passes.common;

import java.util.List;
import com.labwork.exampleopengles.core.general.Entity;

public abstract class RenderPass {

  public abstract void execute(List<Entity> dispatchedEntities);
}
```

OpaqueRenderPass.java

```
package com.labwork.exampleopengles.rendering.passes.concrete;

import java.util.List;
import android.opengl.GLES32;
import com.labwork.exampleopengles.core.general.Entity;
import com.labwork.exampleopengles.rendering.passes.common.RenderPass;
import
com.labwork.exampleopengles.core.components.concrete.RenderingComponent;

public final class OpaqueRenderPass extends RenderPass {

  @Override
  public final void execute(List<Entity> dispatchedEntities) {
    GLES32.glLineWidth(3.0f);

                              GLES32.glClear(GLES32.GL_COLOR_BUFFER_BIT     |
GLES32.GL_DEPTH_BUFFER_BIT);

    for (Entity entity: dispatchedEntities) {
                                   RenderingComponent    renderingComponent    =
entity.getComponent(RenderingComponent.class);

      if (renderingComponent == null)
        continue;

      if (renderingComponent.getMaterial().getShader(OpaqueRenderPass.class) ==
null)
```

```
            continue;

          renderingComponent.render(OpaqueRenderPass.class);
        }
    }
}
```

SimpleProgrammableRenderer.java

```
package com.labwork.exampleopengles.rendering.renderer;

import java.util.List;
import java.util.ArrayList;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLES32;
import android.opengl.GLSurfaceView.Renderer;
import com.labwork.exampleopengles.demo.Standalone;
import com.labwork.exampleopengles.runtime.Framework;
import com.labwork.exampleopengles.core.general.Scene;
import com.labwork.exampleopengles.core.general.Mesh;
import com.labwork.exampleopengles.core.general.Color;
import com.labwork.exampleopengles.core.general.Entity;
import com.labwork.exampleopengles.core.general.Shader;
import com.labwork.exampleopengles.core.general.Material;
import com.labwork.exampleopengles.rendering.passes.common.RenderPass;
import com.labwork.exampleopengles.rendering.passes.concrete.OpaqueRenderPass;
import
com.labwork.exampleopengles.core.components.concrete.RenderingComponent;
import
com.labwork.exampleopengles.core.components.concrete.TransformComponent;
```

```
import
com.labwork.exampleopengles.core.components.concrete.CameraPerspectiveCompo
nent;

public final class SimpleProgrammableRenderer implements Renderer {

  private static final int POLYGON_SIDES = 16;
  private static final float RAY_LENGTH = 2.0f;

  private final List<RenderPass> passes;
  private final List<Entity> dispatchedEntities;

  private Entity camera;
  private Entity raysEntity;
  private Entity polygonEntity;
  private Entity triangleEntity;

  private Color color;
  private Shader shader;
  private Material material;

  public SimpleProgrammableRenderer() {
    this.passes = new ArrayList<>();
    this.passes.add(new OpaqueRenderPass());
    this.dispatchedEntities = new ArrayList<>();
  }

  public void onSurfaceCreated(GL10 unused, EGLConfig config) {
    GLES32.glClearColor(0.0f, 0.0f, 0.3f, 1.0f);
```

```
Scene scene = new Scene();

this.shader = new Shader(OpaqueRenderPass.class, Standalone.SHADER_VERT_SOURCE, Standalone.SHADER_FRAG_SOURCE);
this.shader.compile();

this.color = new Color(255, 20, 147, 255);
this.material = new Material(this.color, this.shader);

this.triangleEntity = new Entity();
this.triangleEntity.addComponent(new TransformComponent(this.triangleEntity));
Mesh triangleMesh = new Mesh(this.generateTriangleVertices(), GLES32.GL_TRIANGLE_STRIP);
this.triangleEntity.addComponent(new RenderingComponent(this.triangleEntity, triangleMesh, this.material));


this.triangleEntity.getComponent(TransformComponent.class).getScale().setX(2.0f);

this.triangleEntity.getComponent(TransformComponent.class).getScale().setY(2.25f)
;

this.triangleEntity.getComponent(TransformComponent.class).getPosition().setX(0.5
f);

this.triangleEntity.getComponent(TransformComponent.class).getPosition().setY(-1.
25f);
```

```
this.triangleEntity.getComponent(TransformComponent.class).getPosition().setZ(-5.0
f);

      this.raysEntity = new Entity();
      this.raysEntity.addComponent(new TransformComponent(this.raysEntity));
            Mesh raysMesh = new Mesh(this.generateRays(POLYGON_SIDES,
RAY_LENGTH), GLES32.GL_LINES);
            this.raysEntity.addComponent(new RenderingComponent(this.raysEntity,
raysMesh, this.material));


this.raysEntity.getComponent(TransformComponent.class).getScale().setX(0.65f);

this.raysEntity.getComponent(TransformComponent.class).getScale().setY(0.65f);

this.raysEntity.getComponent(TransformComponent.class).getPosition().setY(2.5f);

this.raysEntity.getComponent(TransformComponent.class).getPosition().setZ(-5.0f);

      this.polygonEntity = new Entity();
                                    this.polygonEntity.addComponent(new
TransformComponent(this.polygonEntity));
                                    Mesh polygonMesh =new
Mesh(this.generatePolygonVertices(POLYGON_SIDES),
GLES32.GL_TRIANGLE_FAN);
                                    this.polygonEntity.addComponent(new
RenderingComponent(this.polygonEntity, polygonMesh, this.material));
```

```
this.polygonEntity.getComponent(TransformComponent.class).getScale().setX(0.65f)
;

this.polygonEntity.getComponent(TransformComponent.class).getScale().setY(0.65f)
;

this.polygonEntity.getComponent(TransformComponent.class).getPosition().setY(2.5
f);

this.polygonEntity.getComponent(TransformComponent.class).getPosition().setZ(-5.
0f);

      this.camera = new Entity();
      this.camera.addComponent(new TransformComponent(this.camera));
            this.camera.addComponent(new  CameraPerspectiveComponent(this.camera,
new Color(27, 27, 27, 255), 0.001f, 100.0f, 90.0f, 90.0f));

      scene.addEntity(this.camera);
      scene.addEntity(this.raysEntity);
      scene.addEntity(this.polygonEntity);
      scene.addEntity(this.triangleEntity);

      Framework.getInstance().loadScene(scene);

      for (Entity entity : scene.getEntities())
         entity.onStart();
   }
```

```
  public void onSurfaceChanged(GL10 unused, int width, int height) {
     GLES32.glViewport(0, 0, width, height);

this.camera.getComponent(CameraPerspectiveComponent.class).setAspectRatio((flo
at) width / height);
  }

  public void onDrawFrame(GL10 unused) {
     this.dispatchedEntities.clear();

     for (Entity entity : Framework.getInstance().getScene().getEntities()) {
        if (entity.getIsActive()) {
           entity.onUpdate();
           this.dispatchedEntities.add(entity);
        }
     }

     for (RenderPass pass : this.passes)
        pass.execute(this.dispatchedEntities);
  }

  private float[] generateTriangleVertices() {
     final float height = (float) (Math.sqrt(3) / 2);

     return new float[]{
           0.0f, height, 0.0f,  // Top vertex
           -0.5f, 0.0f, 0.0f,   // Bottom left vertex
           0.5f, 0.0f, 0.0f     // Bottom right vertex
     };
```

```
    }

    private float[] generatePolygonVertices(int edgesCount) {
        final int vertexDimensionsCount = 3;
        final float[] vertices = new float[(edgesCount + 2) * vertexDimensionsCount];

        // Center vertex
        vertices[0] = 0.0f;
        vertices[1] = 0.0f;
        vertices[2] = 0.0f;

        for (int i = 0; i <= edgesCount; ++i) {
            int index = (i + 1) * vertexDimensionsCount;
            float angle = (float) (2 * Math.PI * i / edgesCount);

            vertices[index] = (float) Math.cos(angle);
            vertices[index + 1] = (float) Math.sin(angle);
            vertices[index + 2] = 0.0f;
        }

        return vertices;
    }

    private float[] generateRays(int raysCount, float rayLength) {
        final int lineVerticesCount = 2;
        final int vertexDimensionsCount = 3;
                final float[] vertices = new float[raysCount * lineVerticesCount *
vertexDimensionsCount];
```

```
    for (int i = 0; i < raysCount; ++i) {
      int index = i * lineVerticesCount * vertexDimensionsCount;
      float angle = (float) (2 * Math.PI * i / raysCount);

      vertices[index] = (float) Math.cos(angle);
      vertices[index + 1] = (float) Math.sin(angle);
      vertices[index + 2] = 0.0f;

      vertices[index + 3] = (float) Math.cos(angle) * rayLength;
      vertices[index + 4] = (float) Math.sin(angle) * rayLength;
      vertices[index + 5] = 0.0f;
    }

    return vertices;
  }
}
```

ManualGLSurfaceView.java

```
package com.labwork.exampleopengles.rendering.viewport;

import android.content.Context;
import android.opengl.GLSurfaceView;
import
com.labwork.exampleopengles.rendering.renderer.SimpleProgrammableRenderer;

public final class ManualGLSurfaceView extends GLSurfaceView {

  public ManualGLSurfaceView(Context context) {
    super(context);
    super.setEGLContextClientVersion(2);
```

```
        super.setRenderer(new SimpleProgrammableRenderer());
        super.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
    }
}
```

Framework.java

```
package com.labwork.exampleopengles.runtime;

import com.labwork.exampleopengles.core.general.Scene;

public final class Framework {

    private static Framework instance;

    private Scene scene;

    private Framework() {}

    public static Framework getInstance() {
        if (Framework.instance == null) {
            synchronized (Framework.class) {
                if (Framework.instance == null) {
                    Framework.instance = new Framework();
                }
            }
        }

        return Framework.instance;
    }
```

```
  public Scene getScene() {
    return this.scene;
  }

  public void loadScene(Scene scene) {
    this.scene = scene;
  }
}
```

MainActivity.java

```
package com.labwork.exampleopengles;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import com.labwork.exampleopengles.rendering.viewport.ManualGLSurfaceView;

public class MainActivity extends AppCompatActivity {

  @Override
  protected final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.setContentView(new ManualGLSurfaceView(this));
  }
}
```

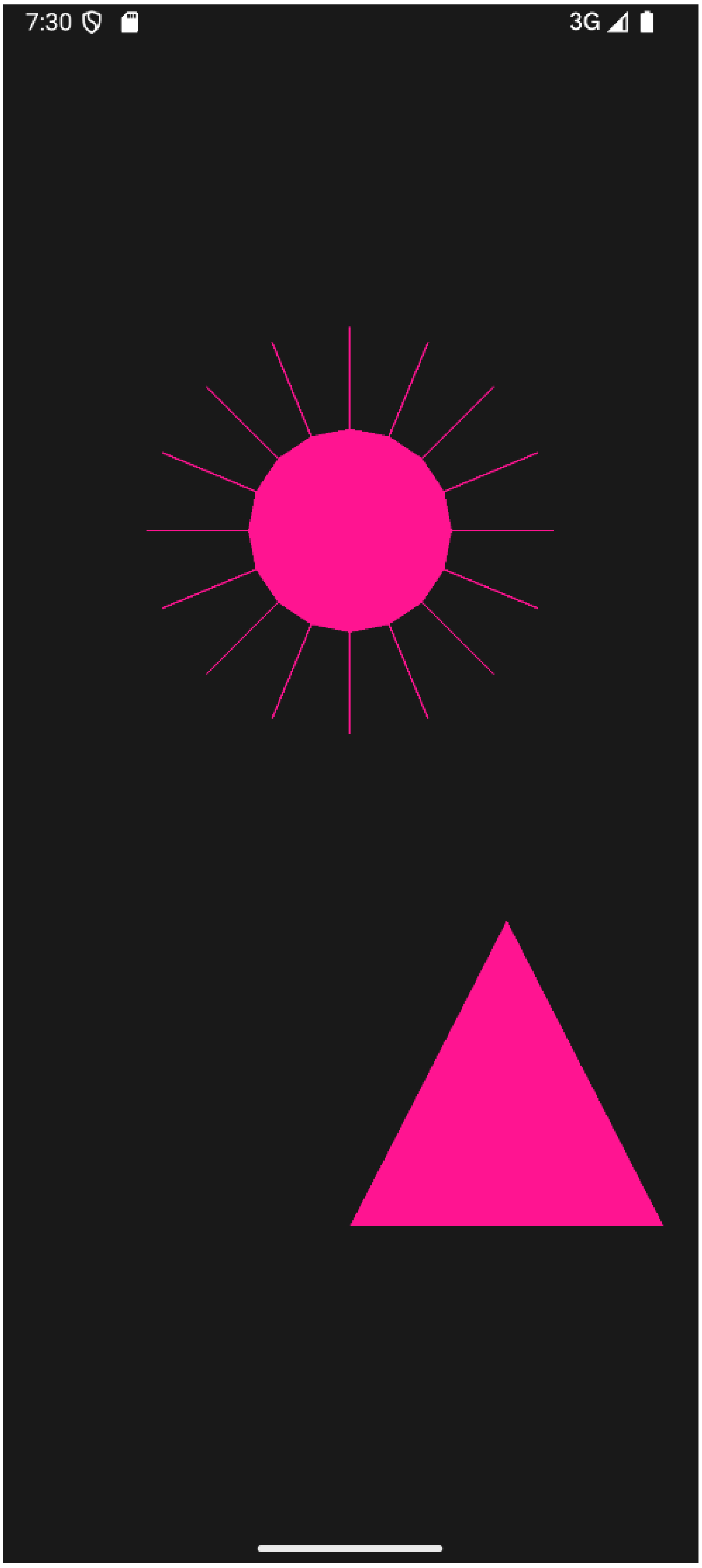


Рисунок 1.5 – Результат роботи за допомогою OpenGLES

ВИСНОВКИ

У рамках виконання лабораторної роботи було розглянуто основи роботи з графічними інтерфейсами Windows GDI, Android Graphics Canvas та OpenGL ES. Було вивчено способи отримання контексту пристрою (HDC) для малювання у Windows GDI, а також реалізовано програмування простих графічних примітивів, таких як трикутник з заповненням та без нього.

Також було досліджено використання пензлів (Brush) у GDI Windows для зафарбовування фігур, що дало змогу створювати більш виразні графічні зображення. У контексті Android Graphics Canvas розглянуто способи визначення кольору об'єктів малювання та програмування трикутників із заповненням та без нього. Окрему увагу було приділено створенню кольорового фону для області відображення у вікні MainActivity застосунку Android, що є важливим аспектом роботи з графічними інтерфейсами.

Окрім цього, було розглянуто концепцію шейдерів у OpenGL ES, які є невід'ємною частиною сучасної графіки. Шейдери дозволяють виконувати гнучке налаштування вигляду графічних об'єктів, покращуючи їхній візуальний ефект. На практиці було реалізовано програмування кола із заповненням в OpenGL ES, що дозволило зрозуміти принципи роботи з буферами та відображенням примітивів у цьому середовищі.

В результаті виконання лабораторної роботи було досягнуто поставлених цілей: освоєно основи малювання графічних примітивів у Windows GDI, Android Graphics Canvas та OpenGL ES, а також реалізовано алгоритми створення трикутників, кіл та кольорового фону. Отримані знання є важливими для подальшого вивчення комп'ютерної графіки та розробки графічних додатків на різних платформах.