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 № 2 з дисципліни
«Програмування комп'ютерної графіки»

«Вказування кольорів об’єктів засобами OpenGL ES та організація інтерфейсу користувача застосунку»

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

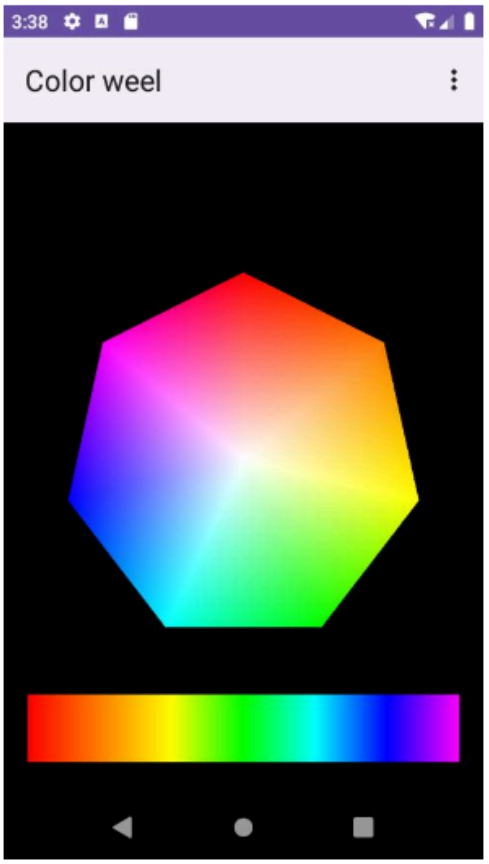
Мета роботи: Отримати навички програмування кольорів об’єктів для графіки OpenGL ES та меню користувача.

Завдання:

- для парних номерів залікових книжок варіант 1: анімоване змінне затемнення лише у семикутника, а у стрічки веселки зображення незмінне;

Рисунок 1.1 – Варіант індивідуального завдання (1402 % 2 = 0)

1. Застосунок **Lab2_GLES** для вибору режиму роботи повинен мати меню з двома пунктами:
- Color Weel
- Color Weel animation



2. У режимі **Color Weel** потрібно відображати статичну картинку – на чорному фоні семикутник та стрічку веселки знизу.

3. У режимі **Color Weel animation** також відображаються ці два об'єкти, але один з них у режимі анімації плавно затемнюється до суцільного чорного і потім плавно відновлює первісну яскравість. Так повторюється до кінця роботи застосунку, або доти, поки не буде обрано інший режим роботи застосунку. Який з об’єктів має анімовану змінну яскравість – це визначається варіантом завдання.

Рисунок 1.2 – Завдання лабораторного практикуму

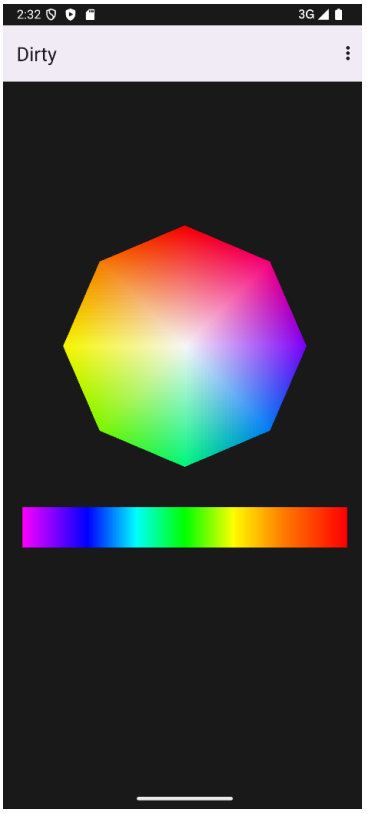


Рисунок 1.3 – Виведення статичного кадру

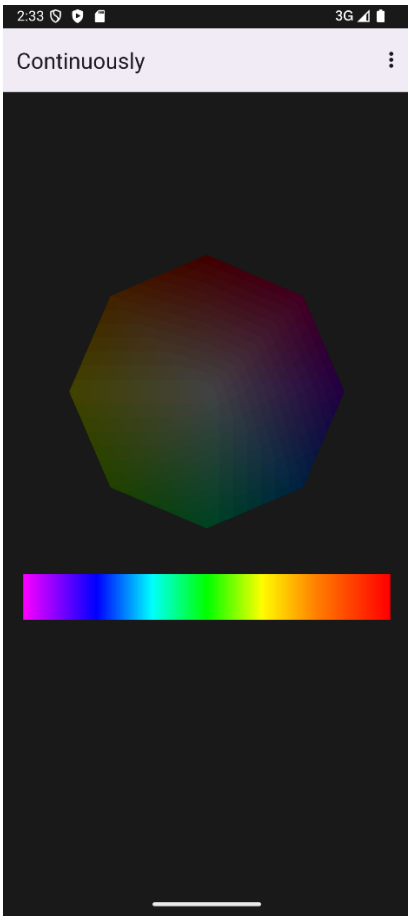


Рисунок 1.4 – Демонстрація затемнення полігону

Component.java

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

import com.labwork.newtoncolorwheel.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.newtoncolorwheel.core.components.concrete;

import android.opengl.Matrix;
import com.labwork.newtoncolorwheel.core.general.Color;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.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.newtoncolorwheel.core.components.concrete;

import android.opengl.GLES32;
import android.opengl.Matrix;
import com.labwork.newtoncolorwheel.core.general.Color;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.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.newtoncolorwheel.core.components.concrete;

import android.opengl.GLES32;
import android.opengl.Matrix;
import com.labwork.newtoncolorwheel.core.general.Color;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.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.getRNormalized(),
super.backgroundColor.getGNormalized(),
super.backgroundColor.getBNormalized(),
super.backgroundColor.getANormalized());
  }

  @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());
  }
}
```

ColorShiftingComponent.java

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

```
import com.labwork.newtoncolorwheel.core.general.Color;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.core.components.common.Component;

public class ColorShiftingComponent extends Component {

  private final int colorChannelMinValue = 0;
  private final int colorChannelMaxValue = 255;

  private Color color;
  private int r, g, b;
  private int step = 3;

  public ColorShiftingComponent(Entity entity) {
     super(entity);
  }

  @Override
  public void onStart() {
                                                          this.color =
super.getEntity().getComponent(RenderingComponent.class).getMaterial().getColor
Albedo();
     this.r = this.color.getR();
     this.g = this.color.getG();
     this.b = this.color.getB();
  }

  @Override
  public void onUpdate() {
```

```
        this.r -= this.step;
        this.g -= this.step;
        this.b -= this.step;

        if (this.r >= this.colorChannelMaxValue || this.r <= this.colorChannelMinValue)
            this.step = -this.step;

        this.color.setR(this.r);
        this.color.setG(this.g);
        this.color.setB(this.b);
    }
}
```

RenderingComponent.java

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

import android.opengl.GLES32;
import android.opengl.Matrix;
import com.labwork.newtoncolorwheel.runtime.Framework;
import com.labwork.newtoncolorwheel.core.general.Mesh;
import com.labwork.newtoncolorwheel.core.general.Color;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.core.general.Shader;
import com.labwork.newtoncolorwheel.core.general.Material;
import com.labwork.newtoncolorwheel.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 void setMesh(Mesh value) {
        this.mesh = value;
    }

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

```
  public void setMaterial(Material value) {
     this.material = value;
  }

  @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());

     Color color = this.material.getColorAlbedo();
     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);
      GLES32.glUniformMatrix4fv(shader.getHandlerUniformMatrixMVP(), 1, false,
this.matrixProjectionViewModel, 0);
           GLES32.glUniform4f(shader.getHandlerUniformMaterialAlbedoColor(),
color.getRNormalized(), color.getGNormalized(), color.getBNormalized(),
color.getANormalized());

     this.mesh.draw();

     GLES32.glUseProgram(0);
```

```
    }
}
```

TransformComponent.java

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

import android.opengl.Matrix;
import com.labwork.newtoncolorwheel.core.general.Axis;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.core.general.Vector3;
import com.labwork.newtoncolorwheel.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.newtoncolorwheel.core.general;

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

Color.java

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

public final class Color {

  private static final float MAX_CHANNEL_VALUE = 255.0f;

  private int r, g, b, a;
  private float rNormalized, gNormalized, bNormalized, aNormalized;

  public Color(int r, int g, int b, int a) {
     this.r = r;
     this.g = g;
     this.b = b;
     this.a = a;
     this.rNormalized = r / Color.MAX_CHANNEL_VALUE;
     this.gNormalized = g / Color.MAX_CHANNEL_VALUE;
     this.bNormalized = b / Color.MAX_CHANNEL_VALUE;
     this.aNormalized = a / Color.MAX_CHANNEL_VALUE;
  }

  public int getR() {
     return this.r;
  }

  public void setR(int value) {
     this.r = value;
     this.rNormalized = value / Color.MAX_CHANNEL_VALUE;
  }
```

```
public float getRNormalized() {
    return this.rNormalized;
}

public int getG() {
    return this.g;
}

public void setG(int value) {
    this.g = value;
    this.gNormalized = value / Color.MAX_CHANNEL_VALUE;
}

public float getGNormalized() {
    return this.gNormalized;
}

public int getB() {
    return this.b;
}

public void setB(int value) {
    this.b = value;
    this.bNormalized = value / Color.MAX_CHANNEL_VALUE;
}

public float getBNormalized() {
    return this.bNormalized;
```

```
    }

    public int getA() {
        return this.a;
    }

    public void setA(int value) {
        this.a = value;
        this.aNormalized = value / Color.MAX_CHANNEL_VALUE;
    }

    public float getANormalized() {
        return this.aNormalized;
    }
}
```

Entity.java

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

import java.util.Map;
import java.util.HashMap;
import java.util.Collection;
import com.labwork.newtoncolorwheel.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.newtoncolorwheel.core.general;

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

public final class Material {

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

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

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

  public Color getColorAlbedo() {
     return this.colorAlbedo;
  }

  public void setColorAlbedo(Color value) {
     this.colorAlbedo = value;
  }

  public void setShader(Shader shader) {
     this.shaders.put(shader.getRenderPass(), shader);
```

```
  }

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

Mesh.java

```
package com.labwork.newtoncolorwheel.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;

  public static final int PAYLOAD_VERTEX_POSITION_SIZE = 3;
  public static final int PAYLOAD_VERTEX_POSITION_INDEX = 0;
  public static final int PAYLOAD_VERTEX_POSITION_OFFSET = 0;
  public static final int PAYLOAD_VERTEX_COLOR_SIZE = 4;
  public static final int PAYLOAD_VERTEX_COLOR_INDEX = 1;
      public static final int PAYLOAD_VERTEX_COLOR_OFFSET =
Mesh.PAYLOAD_VERTEX_POSITION_SIZE * Float.BYTES;
```

```
        public static final int PAYLOAD_STRIDE =
(Mesh.PAYLOAD_VERTEX_POSITION_SIZE +
Mesh.PAYLOAD_VERTEX_COLOR_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.PAYLOAD_VERTEX_POSITION_SIZE +
Mesh.PAYLOAD_VERTEX_COLOR_SIZE);

     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.glVertexAttribPointer(Mesh.PAYLOAD_VERTEX_COLOR_INDEX,
Mesh.PAYLOAD_VERTEX_COLOR_SIZE, GLES32.GL_FLOAT, false,
Mesh.PAYLOAD_STRIDE, Mesh.PAYLOAD_VERTEX_COLOR_OFFSET);

GLES32.glEnableVertexAttribArray(Mesh.PAYLOAD_VERTEX_COLOR_INDEX)
;

    GLES32.glBindVertexArray(0);
    GLES32.glEnableVertexAttribArray(0);
    GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER, 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.newtoncolorwheel.core.general;

import java.util.List;
import java.util.ArrayList;
import java.util.Collection;
import com.labwork.newtoncolorwheel.core.components.common.Component;
import
com.labwork.newtoncolorwheel.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.newtoncolorwheel.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;

private int handlerUniformMatrixMVP = -1;
private int handlerUniformMaterialAlbedoColor = -1;

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 int getHandlerUniformMatrixMVP() {
    return this.handlerUniformMatrixMVP;
}
```

```
  public int getHandlerUniformMaterialAlbedoColor() {
    return this.handlerUniformMaterialAlbedoColor;
  }

  public void compile() {
    GLES32.glCompileShader(this.vertId);
    GLES32.glCompileShader(this.fragId);

    GLES32.glAttachShader(this.programId, this.vertId);
    GLES32.glAttachShader(this.programId, this.fragId);

                                        GLES32.glBindAttribLocation(this.programId,
Mesh.PAYLOAD_VERTEX_POSITION_INDEX, "inVertexPosition");
                                        GLES32.glBindAttribLocation(this.programId,
Mesh.PAYLOAD_VERTEX_COLOR_INDEX, "inVertexColor");

    GLES32.glLinkProgram(this.programId);

                                            this.handlerUniformMatrixMVP        =
GLES32.glGetUniformLocation(this.programId, "uMatrixMVP");
                                        this.handlerUniformMaterialAlbedoColor      =
GLES32.glGetUniformLocation(this.programId, "uMaterialAlbedoColor");
  }

  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.newtoncolorwheel.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.newtoncolorwheel.demo;

public final class Standalone {

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

```
        "   vVertexColor = inVertexColor;\n" +
        "   vMaterialAlbedoColor = uMaterialAlbedoColor;\n" +
        "}\n";

  public static final String SHADER_FRAG_SOURCE =
        "#version 300 es\n" +
        "precision mediump float;\n" +
        "in vec4 vVertexColor;\n" +
        "in vec4 vMaterialAlbedoColor;\n" +
        "out vec4 outFragmentColor;\n" +
        "void main() {\n" +
        "    outFragmentColor = vVertexColor * vMaterialAlbedoColor;\n" +
        "}\n";
}
```

RenderPass.java

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

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

public abstract class RenderPass {

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

OpaqueRenderPass.java

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

import java.util.List;
```

```
import android.opengl.GLES32;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.rendering.passes.common.RenderPass;
import
com.labwork.newtoncolorwheel.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.newtoncolorwheel.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.newtoncolorwheel.demo.Standalone;
import com.labwork.newtoncolorwheel.runtime.Framework;
import com.labwork.newtoncolorwheel.core.general.Scene;
import com.labwork.newtoncolorwheel.core.general.Mesh;
import com.labwork.newtoncolorwheel.core.general.Color;
import com.labwork.newtoncolorwheel.core.general.Entity;
import com.labwork.newtoncolorwheel.core.general.Shader;
import com.labwork.newtoncolorwheel.core.general.Material;
import com.labwork.newtoncolorwheel.rendering.passes.common.RenderPass;
import
com.labwork.newtoncolorwheel.rendering.passes.concrete.OpaqueRenderPass;
import
com.labwork.newtoncolorwheel.core.components.concrete.RenderingComponent;
import
com.labwork.newtoncolorwheel.core.components.concrete.TransformComponent;
import
com.labwork.newtoncolorwheel.core.components.concrete.ColorShiftingComponent;
```

```
import
com.labwork.newtoncolorwheel.core.components.concrete.CameraPerspectiveComp
onent;

public final class SimpleProgrammableRenderer implements Renderer {

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

  private Shader shader;
  private Entity wheel;
  private Entity camera;
  private Entity rectangle;

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

  public void onSurfaceCreated(GL10 unused, EGLConfig config) {
    Scene scene = new Scene();

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

    this.rectangle = new Entity();
    this.rectangle.addComponent(new TransformComponent(this.rectangle));
```

```
            Mesh rectangleMesh = new Mesh(this.generateRectangleVertices(),
GLES32.GL_TRIANGLE_STRIP);
            this.rectangle.addComponent(new RenderingComponent(this.rectangle,
rectangleMesh, new Material(new Color(255, 255, 255, 0), this.shader)));

    this.rectangle.getComponent(TransformComponent.class).getScale().setX(2.0f);
    this.rectangle.getComponent(TransformComponent.class).getScale().setY(0.5f);

this.rectangle.getComponent(TransformComponent.class).getPosition().setY(-3.0f);

    this.wheel = new Entity();
    this.wheel.addComponent(new TransformComponent(this.wheel));
            Mesh wheelMesh = new Mesh(this.generateWheelVertices(),
GLES32.GL_TRIANGLE_FAN);
    this.wheel.addComponent(new ColorShiftingComponent(this.wheel));
      this.wheel.addComponent(new RenderingComponent(this.wheel, wheelMesh,
new Material(new Color(255, 255, 255, 255), this.shader)));

    this.wheel.getComponent(TransformComponent.class).getScale().setX(1.5f);
    this.wheel.getComponent(TransformComponent.class).getScale().setY(1.5f);
    this.wheel.getComponent(TransformComponent.class).getPosition().setY(0.5f);

    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));


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

```
        scene.addEntity(this.wheel);
        scene.addEntity(this.camera);
        scene.addEntity(this.rectangle);

        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) {
        List<Entity> entities = Framework.getInstance().getScene().getEntities();

        for (Entity entity : entities) {
            if (entity.getIsActive()) {
                entity.onUpdate();
                this.dispatchedEntities.add(entity);
            }
        }

        for (RenderPass pass : this.passes)
```

```
        pass.execute(this.dispatchedEntities);
}

private float[] generateWheelVertices() {
    return new float[] {
            0.0f,  0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, // White
            0.0f,  1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, // Red
            0.7f,  0.7f, 0.0f, 1.0f, 0.5f, 0.0f, 1.0f, // Orange
            1.0f,  0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f, // Yellow
            0.7f, -0.7f, 0.0f, 0.5f, 1.0f, 0.0f, 1.0f, // Green
            0.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.5f, 1.0f, // Cyan
           -0.7f, -0.7f, 0.0f, 0.0f, 0.5f, 1.0f, 1.0f, // Blue
           -1.0f,  0.0f, 0.0f, 0.5f, 0.0f, 1.0f, 1.0f, // Purple
           -0.7f,  0.7f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f, // Magenta
            0.0f,  1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, // Closing
    };
}

private float[] generateRectangleVertices() {
    return new float[] {
            -1.0f,  0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, // [0] Left-top (Red)
            -1.0f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, // [1] Left-bottom (Red)
            -0.6f,  0.5f, 0.0f, 1.0f, 0.5f, 0.0f, 1.0f, // [2] Next-top (Orange)
            -0.6f, -0.5f, 0.0f, 1.0f, 0.5f, 0.0f, 1.0f, // [3] Next-bottom (Orange)
            -0.3f,  0.5f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f, // [4] Next-top (Yellow)
            -0.3f, -0.5f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f, // [5] Next-bottom (Yellow)
             0.0f,  0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, // [6] Middle-top (Green)
             0.0f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, // [7] Middle-bottom (Green)
             0.3f,  0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f, // [8] Next-top (Cyan)
```

```
            0.3f, -0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f, // [9] Next-bottom (Cyan)
            0.6f,  0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, // [10] Next-top (Blue)
            0.6f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, // [11] Next-bottom (Blue)
            1.0f,  0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, // [12] Right-top (Magenta)
            1.0f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, // [13] Right-bottom (Magenta)
    };
  }
}
```

ContinuouslyGLSurfaceView.java

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

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

public final class ContinuouslyGLSurfaceView extends GLSurfaceView {

  public ContinuouslyGLSurfaceView(Context context) {
    super(context);
    super.setEGLContextClientVersion(2);
    super.setRenderer(new SimpleProgrammableRenderer());
    super.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
  }
}
```

ManualGLSurfaceView.java

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

import android.content.Context;
import android.opengl.GLSurfaceView;
import
com.labwork.newtoncolorwheel.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.newtoncolorwheel.runtime;

import com.labwork.newtoncolorwheel.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.newtoncolorwheel;

import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import androidx.appcompat.app.AppCompatActivity;
```

```
import com.labwork.newtoncolorwheel.rendering.viewport.ManualGLSurfaceView;
import
com.labwork.newtoncolorwheel.rendering.viewport.ContinuouslyGLSurfaceView;

public class MainActivity extends AppCompatActivity {

  private static final int MENU_ITEM_ID_DIRTY = 1;
  private static final int MENU_ITEM_ID_CONTINUOUSLY = 2;

  private ManualGLSurfaceView viewportManual;
  private ContinuouslyGLSurfaceView viewportContinuous;

  @Override
  protected final void onCreate(Bundle savedInstanceState) {
     this.viewportManual = new ManualGLSurfaceView(this);
     this.viewportContinuous = new ContinuouslyGLSurfaceView(this);
     super.onCreate(savedInstanceState);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
     menu.add(0, MainActivity.MENU_ITEM_ID_DIRTY, 0, "Dirty");
                menu.add(0,   MainActivity.MENU_ITEM_ID_CONTINUOUSLY,   0,
"Continuously");
     return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
```

```
        super.setTitle(item.getTitle());

        switch (item.getItemId()) {
            case MainActivity.MENU_ITEM_ID_DIRTY:
                super.setContentView(this.viewportManual);
                return true;
            case MainActivity.MENU_ITEM_ID_CONTINUOUSLY:
                super.setContentView(this.viewportContinuous);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

ВИСНОВКИ

У рамках виконання даної лабораторної роботи було розглянуто основи створення графічних зображень у режимах статичного та анімованого відображення за допомогою інтерфейсів для малювання в графічних програмах. У режимі Color Weel було реалізовано відображення статичного семикутника на чорному фоні разом із стрічкою веселки. У режимі Color Weel animation один з об’єктів за допомогою анімації плавно змінював яскравість від повного затемнення до початкового стану, що дозволило дослідити основи анімації графічних об’єктів у реальному часі.Під час роботи було опрацьовано методи управління кольоровими переходами та створення плавних анімацій, що є важливим елементом у розробці динамічних графічних інтерфейсів. В результаті було досягнуто поставлених завдань, зокрема створення двох графічних об’єктів та реалізація ефекту змінної яскравості в анімації, що допомогло здобути практичні навички у програмуванні графіки.