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 № 3 з дисципліни
«Програмування комп'ютерної графіки»

**«Перетворення координат та проєкції.
Анімація. Керування за допомогою сенсорів вводу»**

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 відображення тривимірних об’єктів засобами графіки OpenGL ES.

Завдання:

1. Застосунок **Lab3_GLES** для вибору режиму роботи повинен мати меню з двома пунктами:
   - Pyramid rotation
   - Nine Cubes

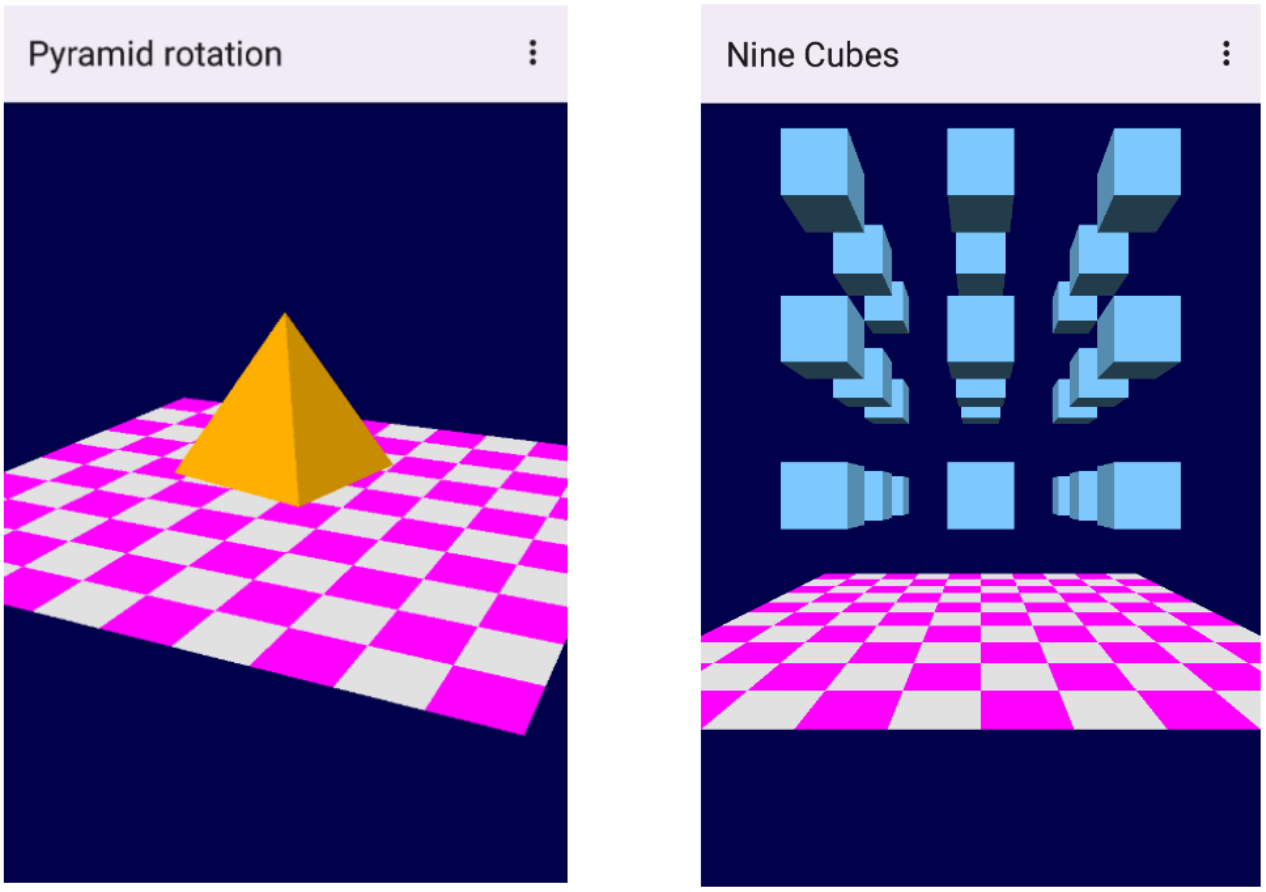

Рисунок 1.1 – Завдання лабораторної роботи

2. Обрати кольори фону, шахового поля, піраміди, кубів, вертикальний кут конусу огляду – на свій розсуд, продемонструвавши, як їх можна змінити.

3. У режимі Pyramid rotation забезпечити безперервне обертання піраміди вертикалі над шаховим полем. Рендеринг має бути у режимі *RENDERMODE_CONTINUOUSLY*.

4. Запрограмувати, щоб у режимі Pyramid rotation можна було б за допомогою пересування стілуса (пальця) по екрану змінювати ракурс показу сцени наступним чином:

- обертати камеру навколо вертикальної осі (змінювати кут α)
- наближати-віддаляти камеру відносно центру сцени

При будь-яких змінах ракурсу показу камера постійно повинна дивитися у центр сцени.

Рисунок 1.2 – Завдання лабораторного практикуму

5. У режимі Nine Cubes показ сцени статичний, з постійним ракурсом – якщо не торкатися сенсорного екрану. Сцена складається з шахового поля та решіткі з 27 кубів. Рендеринг у режимі *RENDERMODE_WHEN_DIRTY*.

6. Запрограмувати, щоб у режимі Nine Cubes можна було б за допомогою пересування стілуса (пальця) по екрану змінювати ракурс показу сцени наступним чином (імітувати рух на літальному апараті):

- рухатися вперед-назад вздовж напрямку зору камери
- робити повороти вправо-вліво,
- змінювати нахил камери уверх-вниз і потім відповідно рухатися вздовж нового напрямку зору камери

7. У режимі Nine Cubes продемонструвати проходження користувача програми серед кубів без наїзду на них.

Рисунок 1.3 – Завдання лабораторного практикуму

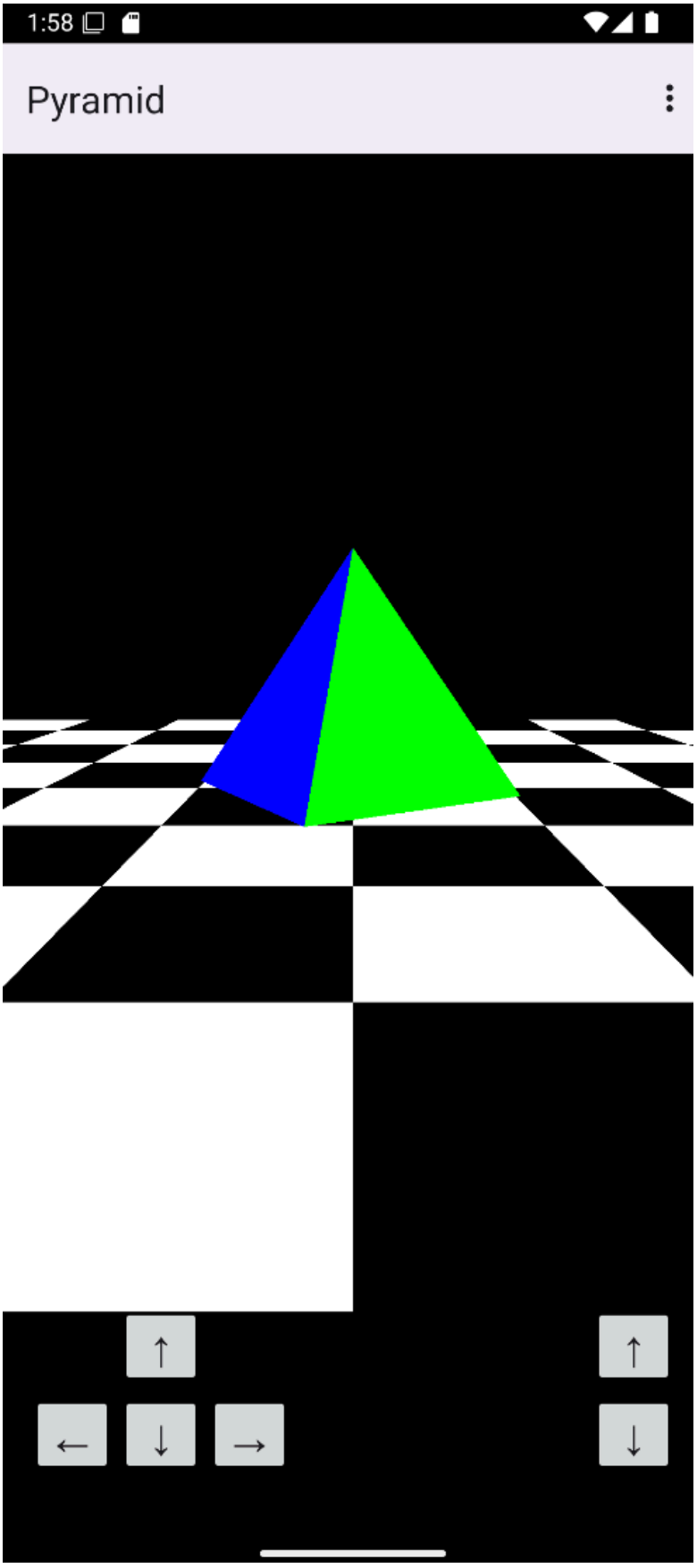

Рисунок 1.4 – Сцена з пірамідою

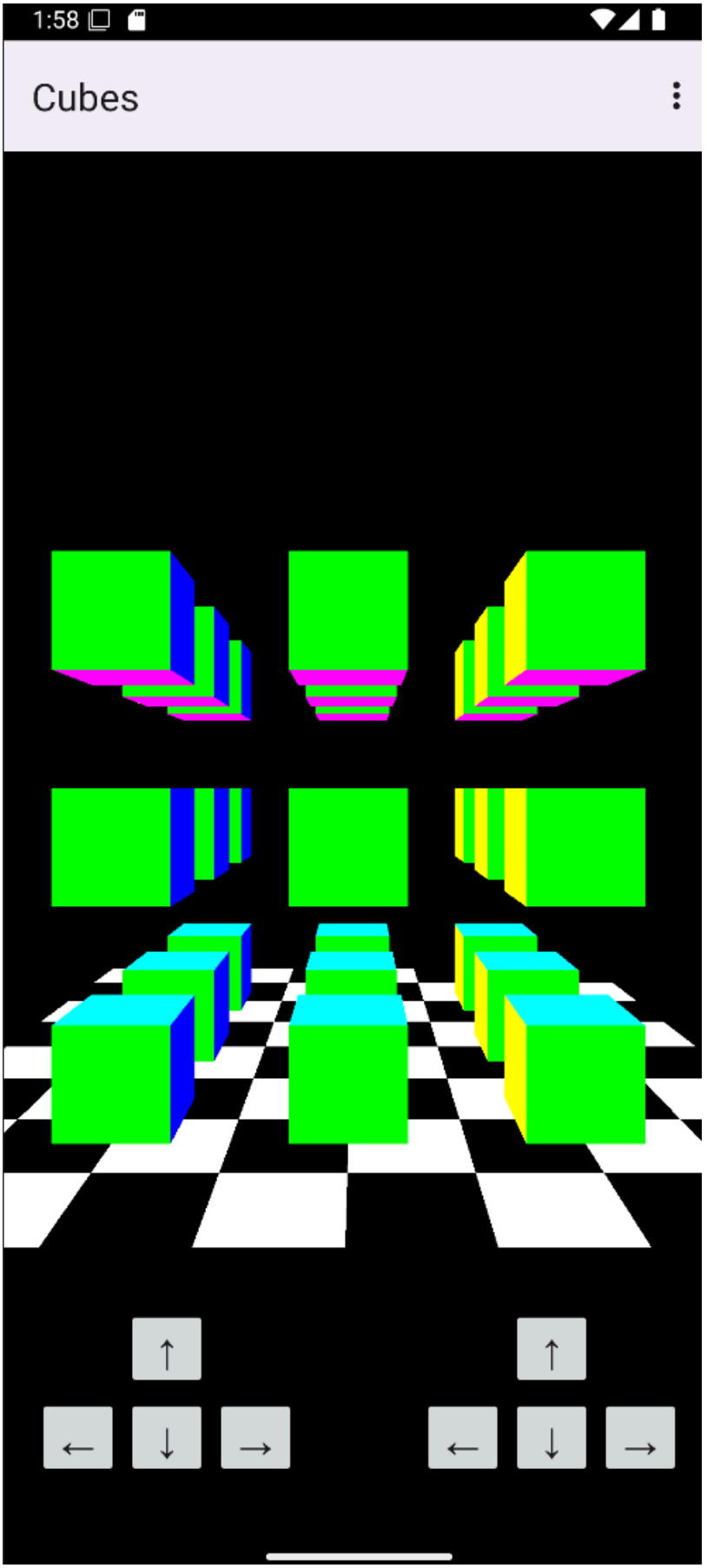

Рисунок 1.5 – Сцена з кубами

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.animationsexample.core.components.concrete;

import android.opengl.GLES32;
import android.opengl.Matrix;
import android.opengl.GLSurfaceView;
import com.labwork.animationsexample.runtime.Framework;
import com.labwork.animationsexample.core.general.Color;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.general.Vector3;

public final class CameraPerspectiveComponent extends CameraComponent {
  private final Vector3 target;
  private final GLSurfaceView viewport;

  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.viewport = Framework.getInstance().getSurfaceView();
        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;
    }

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

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

```
  @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(float deltaTime) {
    this.setAspectRatio((float)this.viewport.getWidth() / this.viewport.getHeight());
              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());
  }
}
```

RotationComponent.java

```
package com.labwork.animationsexample.demo.components;

import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.components.common.Component;
import
com.labwork.animationsexample.core.components.concrete.TransformComponent;

public final class RotationComponent extends Component {
  private final float speed = 300.0f;

  private float angle;
  private TransformComponent transform;

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

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

  @Override
  public void onUpdate(float deltaTime) {
    this.angle += this.speed * deltaTime;
    this.transform.getRotation().setY(this.angle);
  }
}
```

RenderingComponent.java

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

import android.opengl.GLES32;
import com.labwork.animationsexample.core.general.Mesh;
import com.labwork.animationsexample.core.general.Color;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.general.Shader;
import com.labwork.animationsexample.core.general.Material;
import com.labwork.animationsexample.core.components.common.Component;

public final class RenderingComponent extends Component {
  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;
  }

  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) {
        Color color = this.material.getColorAlbedo();
        Shader shader = this.material.getShader(renderPass);

            GLES32.glUniformMatrix4fv(shader.getVariableHandler("uMatrixModel"), 1,
false, this.transform.getMatrixModel(), 0);
                GLES32.glUniform4f(shader.getVariableHandler("uMaterialAlbedoColor"),
color.getRNormalized(), color.getGNormalized(), color.getBNormalized(),
color.getANormalized());

        this.mesh.draw();
    }
}
```

TransformComponent.java

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

import android.opengl.Matrix;
import com.labwork.animationsexample.core.general.Axis;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.general.Vector3;
import com.labwork.animationsexample.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,
}
```

NoClipControllerComponent.java

```
package com.labwork.animationsexample.demo.components;

import android.view.View;
import android.view.MotionEvent;
import android.widget.Button;
import android.widget.RelativeLayout;
import android.widget.RelativeLayout.LayoutParams;
import com.labwork.animationsexample.runtime.Framework;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.general.Vector3;
import com.labwork.animationsexample.core.components.common.Component;
import
com.labwork.animationsexample.core.components.concrete.TransformComponent;

public final class NoClipControllerComponent extends Component {
  private static final float MOVEMENT_SPEED = 1.0f;
  private static final float ROTATION_SPEED = 45.0f;

  private TransformComponent transform;

  private boolean isMovingLeft;
  private boolean isMovingRight;
  private boolean isMovingForward;
  private boolean isMovingBackward;

  private boolean isRotatingUp;
  private boolean isRotatingDown;
  private boolean isRotatingLeft;
```

```
    private boolean isRotatingRight;

    private final Button buttonMoveLeft;
    private final Button buttonMoveRight;
    private final Button buttonMoveForward;
    private final Button buttonMoveBackward;

    private final Button buttonRotateUp;
    private final Button buttonRotateDown;
    private final Button buttonRotateLeft;
    private final Button buttonRotateRight;

    private final Vector3 tempVector = new Vector3(0, 0, 0);
    private final Vector3 moveDirection = new Vector3(0, 0, 0);

    public NoClipControllerComponent(Entity entity, Button buttonMoveForward, Button buttonMoveBackward, Button buttonMoveLeft, Button buttonMoveRight, Button buttonRotateUp, Button buttonRotateDown, Button buttonRotateLeft, Button buttonRotateRight) {
        super(entity);

        int spacing = 10;
        int leftOffset = 50;
        int rightOffset = 50;
        int buttonSize = 125;
        int bottomOffset = 150;
        float textSize = 30.0f;

        buttonMoveLeft.setVisibility(View.INVISIBLE);
```

```
        buttonMoveRight.setVisibility(View.INVISIBLE);
        buttonMoveForward.setVisibility(View.INVISIBLE);
        buttonMoveBackward.setVisibility(View.INVISIBLE);
        buttonRotateUp.setVisibility(View.INVISIBLE);
        buttonRotateDown.setVisibility(View.INVISIBLE);
        buttonRotateLeft.setVisibility(View.INVISIBLE);
        buttonRotateRight.setVisibility(View.INVISIBLE);

        this.buttonMoveLeft = buttonMoveLeft;
        buttonMoveLeft.setId(View.generateViewId());
        buttonMoveLeft.setPadding(0, 0, 0, 0);
        buttonMoveLeft.setText("←");
        buttonMoveLeft.setTextSize(textSize);
        LayoutParams paramsMoveLeft = new LayoutParams(buttonSize, buttonSize);
        paramsMoveLeft.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
        paramsMoveLeft.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
        paramsMoveLeft.leftMargin = leftOffset;
        paramsMoveLeft.bottomMargin = bottomOffset;
        buttonMoveLeft.setLayoutParams(paramsMoveLeft);
        buttonMoveLeft.setOnTouchListener(this::handleMoveLeftButtonTouch);

        this.buttonMoveBackward = buttonMoveBackward;
        buttonMoveBackward.setId(View.generateViewId());
        buttonMoveBackward.setPadding(0, 0, 0, 0);
        buttonMoveBackward.setText("↓");
        buttonMoveBackward.setTextSize(textSize);
        LayoutParams paramsMoveDown = new LayoutParams(buttonSize, buttonSize);
                                paramsMoveDown.addRule(RelativeLayout.RIGHT_OF,
buttonMoveLeft.getId());
```

```
        paramsMoveDown.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
        paramsMoveDown.leftMargin = spacing;
        paramsMoveDown.bottomMargin = bottomOffset;
        buttonMoveBackward.setLayoutParams(paramsMoveDown);

buttonMoveBackward.setOnTouchListener(this::handleMoveBackwardButtonTouch)
;

        this.buttonMoveForward = buttonMoveForward;
        buttonMoveForward.setId(View.generateViewId());
        buttonMoveForward.setPadding(0, 0, 0, 0);
        buttonMoveForward.setText("↑");
        buttonMoveForward.setTextSize(textSize);
        LayoutParams paramsMoveUp = new LayoutParams(buttonSize, buttonSize);
                                        paramsMoveUp.addRule(RelativeLayout.ABOVE,
buttonMoveBackward.getId());
                                    paramsMoveUp.addRule(RelativeLayout.ALIGN_LEFT,
buttonMoveBackward.getId());
        paramsMoveUp.bottomMargin = spacing;
        buttonMoveForward.setLayoutParams(paramsMoveUp);

buttonMoveForward.setOnTouchListener(this::handleMoveForwardButtonTouch);

        this.buttonMoveRight = buttonMoveRight;
        buttonMoveRight.setId(View.generateViewId());
        buttonMoveRight.setPadding(0, 0, 0, 0);
        buttonMoveRight.setText("→");
        buttonMoveRight.setTextSize(textSize);
        LayoutParams paramsMoveRight = new LayoutParams(buttonSize, buttonSize);
```

```
                paramsMoveRight.addRule(RelativeLayout.RIGHT_OF,
buttonMoveBackward.getId());
                paramsMoveRight.addRule(RelativeLayout.ALIGN_TOP,
buttonMoveBackward.getId());
    paramsMoveRight.leftMargin = spacing;
    buttonMoveRight.setLayoutParams(paramsMoveRight);
    buttonMoveRight.setOnTouchListener(this::handleMoveRightButtonTouch);

    this.buttonRotateRight = buttonRotateRight;
    buttonRotateRight.setId(View.generateViewId());
    buttonRotateRight.setPadding(0, 0, 0, 0);
    buttonRotateRight.setText("→");
    buttonRotateRight.setTextSize(textSize);
    LayoutParams paramsRotateRight = new LayoutParams(buttonSize, buttonSize);
    paramsRotateRight.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
    paramsRotateRight.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
    paramsRotateRight.rightMargin = rightOffset;
    paramsRotateRight.bottomMargin = bottomOffset;
    buttonRotateRight.setLayoutParams(paramsRotateRight);
    buttonRotateRight.setOnTouchListener(this::handleRotateRightButtonTouch);

    this.buttonRotateDown = buttonRotateDown;
    buttonRotateDown.setId(View.generateViewId());
    buttonRotateDown.setPadding(0, 0, 0, 0);
    buttonRotateDown.setText("↓");
    buttonRotateDown.setTextSize(textSize);
        LayoutParams paramsRotateDown = new LayoutParams(buttonSize,
buttonSize);
```

```
                paramsRotateDown.addRule(RelativeLayout.LEFT_OF,
buttonRotateRight.getId());
    paramsRotateDown.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
    paramsRotateDown.rightMargin = spacing;
    paramsRotateDown.bottomMargin = bottomOffset;
    buttonRotateDown.setLayoutParams(paramsRotateDown);
    buttonRotateDown.setOnTouchListener(this::handleRotateDownButtonTouch);

    this.buttonRotateLeft = buttonRotateLeft;
    buttonRotateLeft.setId(View.generateViewId());
    buttonRotateLeft.setPadding(0, 0, 0, 0);
    buttonRotateLeft.setText("←");
    buttonRotateLeft.setTextSize(textSize);
    LayoutParams paramsRotateLeft = new LayoutParams(buttonSize, buttonSize);
                    paramsRotateLeft.addRule(RelativeLayout.LEFT_OF,
buttonRotateDown.getId());
                paramsRotateLeft.addRule(RelativeLayout.ALIGN_TOP,
buttonRotateDown.getId());
    paramsRotateLeft.rightMargin = spacing;
    buttonRotateLeft.setLayoutParams(paramsRotateLeft);
    buttonRotateLeft.setOnTouchListener(this::handleRotateLeftButtonTouch);

    this.buttonRotateUp = buttonRotateUp;
    buttonRotateUp.setId(View.generateViewId());
    buttonRotateUp.setPadding(0, 0, 0, 0);
    buttonRotateUp.setText("↑");
    buttonRotateUp.setTextSize(textSize);
    LayoutParams paramsRotateUp = new LayoutParams(buttonSize, buttonSize);
    paramsRotateUp.addRule(RelativeLayout.ABOVE, buttonRotateDown.getId());
```

```
                paramsRotateUp.addRule(RelativeLayout.ALIGN_LEFT,
buttonRotateDown.getId());
    paramsRotateUp.bottomMargin = spacing;
    buttonRotateUp.setLayoutParams(paramsRotateUp);
    buttonRotateUp.setOnTouchListener(this::handleRotateUpButtonTouch);

    Framework.getInstance().getViewport().register(buttonMoveLeft);
    Framework.getInstance().getViewport().register(buttonMoveRight);
    Framework.getInstance().getViewport().register(buttonMoveForward);
    Framework.getInstance().getViewport().register(buttonMoveBackward);
    Framework.getInstance().getViewport().register(buttonRotateUp);
    Framework.getInstance().getViewport().register(buttonRotateDown);
    Framework.getInstance().getViewport().register(buttonRotateLeft);
    Framework.getInstance().getViewport().register(buttonRotateRight);
  }

  private boolean handleMoveForwardButtonTouch(View view, MotionEvent event)
{
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isMovingForward = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isMovingForward = false;
            return true;
        default:
            return false;
    }
```

```
    }

    private boolean handleMoveBackwardButtonTouch(View view, MotionEvent
event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                this.isMovingBackward = true;
                return true;
            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL:
                this.isMovingBackward = false;
                return true;
            default:
                return false;
        }
    }

    private boolean handleMoveLeftButtonTouch(View view, MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                this.isMovingLeft = true;
                return true;
            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL:
                this.isMovingLeft = false;
                return true;
            default:
                return false;
        }
```

```
}

private boolean handleMoveRightButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isMovingRight = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isMovingRight = false;
            return true;
        default:
            return false;
    }
}

private boolean handleRotateUpButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isRotatingUp = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isRotatingUp = false;
            return true;
        default:
            return false;
    }
}
```

```
private boolean handleRotateDownButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isRotatingDown = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isRotatingDown = false;
            return true;
        default:
            return false;
    }
}

private boolean handleRotateLeftButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isRotatingLeft = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isRotatingLeft = false;
            return true;
        default:
            return false;
    }
}
```

```
private boolean handleRotateRightButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isRotatingRight = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isRotatingRight = false;
            return true;
        default:
            return false;
    }
}

@Override
public void onStart() {
    this.buttonMoveLeft.setVisibility(View.VISIBLE);
    this.buttonMoveRight.setVisibility(View.VISIBLE);
    this.buttonMoveForward.setVisibility(View.VISIBLE);
    this.buttonMoveBackward.setVisibility(View.VISIBLE);
    this.buttonRotateUp.setVisibility(View.VISIBLE);
    this.buttonRotateDown.setVisibility(View.VISIBLE);
    this.buttonRotateLeft.setVisibility(View.VISIBLE);
    this.buttonRotateRight.setVisibility(View.VISIBLE);

    this.transform = super.getEntity().getComponent(TransformComponent.class);
}

@Override
```

```
  public void onUpdate(float deltaTime) {
     Vector3 position = this.transform.getPosition();
     Vector3 rotation = this.transform.getRotation();
          float moveSpeed = NoClipControllerComponent.MOVEMENT_SPEED *
deltaTime;
           float rotateSpeed = NoClipControllerComponent.ROTATION_SPEED *
deltaTime;

     if (this.isRotatingUp) {
        rotation.setX(rotation.getX() - rotateSpeed);
     }
     if (this.isRotatingDown) {
        rotation.setX(rotation.getX() + rotateSpeed);
     }
     if (this.isRotatingLeft) {
        rotation.setY(rotation.getY() + rotateSpeed);
     }
     if (this.isRotatingRight) {
        rotation.setY(rotation.getY() - rotateSpeed);
     }

     float pitch = rotation.getX();

     if (pitch > 90.0f)
        pitch = 90.0f;

     if (pitch < -90.0f)
        pitch = -90.0f;
```

```
    rotation.setX(pitch);

    Vector3 forward = this.transform.getForward();
    Vector3 right = this.transform.getRight();

    this.moveDirection.setXYZ(0, 0, 0);

    if (this.isMovingForward) {
        Vector3.add(this.moveDirection, forward, this.moveDirection);
    }
    if (this.isMovingBackward) {
        Vector3.subtract(this.moveDirection, forward, this.moveDirection);
    }
    if (this.isMovingRight) {
        Vector3.subtract(this.moveDirection, right, this.moveDirection);
    }
    if (this.isMovingLeft) {
        Vector3.add(this.moveDirection, right, this.moveDirection);
    }

    if (this.moveDirection.magnitude() > 0) {
        Vector3.normalize(this.moveDirection, this.tempVector);
        Vector3.multiply(this.tempVector, moveSpeed, this.moveDirection);
        position.setX(position.getX() + this.moveDirection.getX());
        position.setY(position.getY() + this.moveDirection.getY());
        position.setZ(position.getZ() + this.moveDirection.getZ());
    }
}
```

```
  @Override
  public void onDestroy() {
     this.buttonMoveLeft.setVisibility(View.INVISIBLE);
     this.buttonMoveRight.setVisibility(View.INVISIBLE);
     this.buttonMoveForward.setVisibility(View.INVISIBLE);
     this.buttonMoveBackward.setVisibility(View.INVISIBLE);
     this.buttonRotateUp.setVisibility(View.INVISIBLE);
     this.buttonRotateDown.setVisibility(View.INVISIBLE);
     this.buttonRotateLeft.setVisibility(View.INVISIBLE);
     this.buttonRotateRight.setVisibility(View.INVISIBLE);
  }
}
```

FixedOrientationControllerComponent.java

```
package com.labwork.animationsexample.demo.components;

import android.view.View;
import android.view.MotionEvent;
import android.widget.Button;
import android.widget.RelativeLayout;
import android.widget.RelativeLayout.LayoutParams;
import com.labwork.animationsexample.runtime.Framework;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.general.Vector3;
import com.labwork.animationsexample.core.components.common.Component;
import
com.labwork.animationsexample.core.components.concrete.TransformComponent;

public final class FixedOrientationControllerComponent extends Component {
  private static final float MOVEMENT_SPEED = 1.0f;
```

```
    private TransformComponent transform;

    private boolean isMovingLeft;
    private boolean isMovingRight;
    private boolean isMovingForward;
    private boolean isMovingBackward;
    private boolean isMovingUp;
    private boolean isMovingDown;

    private final Button buttonMoveLeft;
    private final Button buttonMoveRight;
    private final Button buttonMoveForward;
    private final Button buttonMoveBackward;
    private final Button buttonMoveUp;
    private final Button buttonMoveDown;

    private final Vector3 up = new Vector3(0.0f, 1.0f, 0.0f);
    private final Vector3 right = new Vector3(0.0f, 0.0f, 0.0f);
    private final Vector3 toOrigin = new Vector3(0.0f, 0.0f, 0.0f);
    private final Vector3 newToOrigin = new Vector3(0.0f, 0.0f, 0.0f);

        public FixedOrientationControllerComponent(Entity entity, Button
buttonMoveForward, Button buttonMoveBackward, Button buttonMoveLeft, Button
buttonMoveRight, Button buttonMoveUp, Button buttonMoveDown) {
        super(entity);

        int spacing = 10;
        int leftOffset = 50;
```

```
int rightOffset = 50;
int buttonSize = 125;
int bottomOffset = 150;
float textSize = 30.0f;

buttonMoveLeft.setVisibility(View.INVISIBLE);
buttonMoveRight.setVisibility(View.INVISIBLE);
buttonMoveForward.setVisibility(View.INVISIBLE);
buttonMoveBackward.setVisibility(View.INVISIBLE);
buttonMoveUp.setVisibility(View.INVISIBLE);
buttonMoveDown.setVisibility(View.INVISIBLE);

this.buttonMoveLeft = buttonMoveLeft;
buttonMoveLeft.setId(View.generateViewId());
buttonMoveLeft.setPadding(0, 0, 0, 0);
buttonMoveLeft.setText("←");
buttonMoveLeft.setTextSize(textSize);
LayoutParams paramsMoveLeft = new LayoutParams(buttonSize, buttonSize);
paramsMoveLeft.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
paramsMoveLeft.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
paramsMoveLeft.leftMargin = leftOffset;
paramsMoveLeft.bottomMargin = bottomOffset;
buttonMoveLeft.setLayoutParams(paramsMoveLeft);
buttonMoveLeft.setOnTouchListener(this::handleMoveLeftButtonTouch);

this.buttonMoveBackward = buttonMoveBackward;
buttonMoveBackward.setId(View.generateViewId());
buttonMoveBackward.setPadding(0, 0, 0, 0);
buttonMoveBackward.setText("↓");
```

```
        buttonMoveBackward.setTextSize(textSize);
        LayoutParams paramsMoveDown = new LayoutParams(buttonSize, buttonSize);
                                    paramsMoveDown.addRule(RelativeLayout.RIGHT_OF,
buttonMoveLeft.getId());
        paramsMoveDown.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
        paramsMoveDown.leftMargin = spacing;
        paramsMoveDown.bottomMargin = bottomOffset;
        buttonMoveBackward.setLayoutParams(paramsMoveDown);

buttonMoveBackward.setOnTouchListener(this::handleMoveBackwardButtonTouch)
;

        this.buttonMoveForward = buttonMoveForward;
        buttonMoveForward.setId(View.generateViewId());
        buttonMoveForward.setPadding(0, 0, 0, 0);
        buttonMoveForward.setText("↑");
        buttonMoveForward.setTextSize(textSize);
        LayoutParams paramsMoveUp = new LayoutParams(buttonSize, buttonSize);
                                        paramsMoveUp.addRule(RelativeLayout.ABOVE,
buttonMoveBackward.getId());
                                    paramsMoveUp.addRule(RelativeLayout.ALIGN_LEFT,
buttonMoveBackward.getId());
        paramsMoveUp.bottomMargin = spacing;
        buttonMoveForward.setLayoutParams(paramsMoveUp);

buttonMoveForward.setOnTouchListener(this::handleMoveForwardButtonTouch);

        this.buttonMoveRight = buttonMoveRight;
        buttonMoveRight.setId(View.generateViewId());
```

```
        buttonMoveRight.setPadding(0, 0, 0, 0);
        buttonMoveRight.setText("→");
        buttonMoveRight.setTextSize(textSize);
        LayoutParams paramsMoveRight = new LayoutParams(buttonSize, buttonSize);
                                    paramsMoveRight.addRule(RelativeLayout.RIGHT_OF,
buttonMoveBackward.getId());
                                   paramsMoveRight.addRule(RelativeLayout.ALIGN_TOP,
buttonMoveBackward.getId());
        paramsMoveRight.leftMargin = spacing;
        buttonMoveRight.setLayoutParams(paramsMoveRight);
        buttonMoveRight.setOnTouchListener(this::handleMoveRightButtonTouch);

        this.buttonMoveDown = buttonMoveDown;
        buttonMoveDown.setId(View.generateViewId());
        buttonMoveDown.setPadding(0, 0, 0, 0);
        buttonMoveDown.setText("↓");
        buttonMoveDown.setTextSize(textSize);
            LayoutParams paramsMoveDownRight = new LayoutParams(buttonSize,
buttonSize);

paramsMoveDownRight.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
        paramsMoveDownRight.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
        paramsMoveDownRight.rightMargin = rightOffset;
        paramsMoveDownRight.bottomMargin = bottomOffset;
        buttonMoveDown.setLayoutParams(paramsMoveDownRight);
        buttonMoveDown.setOnTouchListener(this::handleMoveDownButtonTouch);

        this.buttonMoveUp = buttonMoveUp;
        buttonMoveUp.setId(View.generateViewId());
```

```
        buttonMoveUp.setPadding(0, 0, 0, 0);
        buttonMoveUp.setText("↑");
        buttonMoveUp.setTextSize(textSize);
                LayoutParams paramsMoveUpRight = new LayoutParams(buttonSize,
buttonSize);
                                paramsMoveUpRight.addRule(RelativeLayout.ABOVE,
buttonMoveDown.getId());
                            paramsMoveUpRight.addRule(RelativeLayout.ALIGN_LEFT,
buttonMoveDown.getId());
        paramsMoveUpRight.bottomMargin = spacing;
        buttonMoveUp.setLayoutParams(paramsMoveUpRight);
        buttonMoveUp.setOnTouchListener(this::handleMoveUpButtonTouch);

        Framework.getInstance().getViewport().register(buttonMoveLeft);
        Framework.getInstance().getViewport().register(buttonMoveRight);
        Framework.getInstance().getViewport().register(buttonMoveForward);
        Framework.getInstance().getViewport().register(buttonMoveBackward);
        Framework.getInstance().getViewport().register(buttonMoveUp);
        Framework.getInstance().getViewport().register(buttonMoveDown);
    }

    private boolean handleMoveForwardButtonTouch(View view, MotionEvent event)
{
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                this.isMovingForward = true;
                return true;
            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL:
```

```
            this.isMovingForward = false;
            return true;
        default:
            return false;
    }
  }

    private boolean handleMoveBackwardButtonTouch(View view, MotionEvent
event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isMovingBackward = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isMovingBackward = false;
            return true;
        default:
            return false;
    }
  }

  private boolean handleMoveLeftButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isMovingLeft = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
```

```
            this.isMovingLeft = false;
            return true;
        default:
            return false;
    }
}

private boolean handleMoveRightButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isMovingRight = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isMovingRight = false;
            return true;
        default:
            return false;
    }
}

private boolean handleMoveUpButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isMovingUp = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isMovingUp = false;
```

```
            return true;
        default:
            return false;
    }
}

private boolean handleMoveDownButtonTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            this.isMovingDown = true;
            return true;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            this.isMovingDown = false;
            return true;
        default:
            return false;
    }
}

@Override
public void onStart() {
    this.buttonMoveLeft.setVisibility(View.VISIBLE);
    this.buttonMoveRight.setVisibility(View.VISIBLE);
    this.buttonMoveForward.setVisibility(View.VISIBLE);
    this.buttonMoveBackward.setVisibility(View.VISIBLE);
    this.buttonMoveUp.setVisibility(View.VISIBLE);
    this.buttonMoveDown.setVisibility(View.VISIBLE);
```

```
        this.transform = super.getEntity().getComponent(TransformComponent.class);
    }

    @Override
    public void onUpdate(float deltaTime) {
        Vector3 position = this.transform.getPosition();
        Vector3 rotation = this.transform.getRotation();
                                                    float moveSpeed =
FixedOrientationControllerComponent.MOVEMENT_SPEED * deltaTime;

        this.toOrigin.setXYZ(0, 0, 0);
        Vector3.subtract(this.toOrigin, position, this.toOrigin);
        Vector3.normalize(this.toOrigin, this.toOrigin);

        this.right.setXYZ(0, 0, 0);
        Vector3.cross(this.toOrigin, this.up,this.right);
        Vector3.normalize(this.right, this.right);

        if (this.isMovingForward) {
            position.setX(position.getX() + this.toOrigin.getX() * moveSpeed);
            position.setY(position.getY() + this.toOrigin.getY() * moveSpeed);
            position.setZ(position.getZ() + this.toOrigin.getZ() * moveSpeed);
        }

        if (this.isMovingBackward) {
            position.setX(position.getX() - this.toOrigin.getX() * moveSpeed);
            position.setY(position.getY() - this.toOrigin.getY() * moveSpeed);
            position.setZ(position.getZ() - this.toOrigin.getZ() * moveSpeed);
        }
```

```
if (this.isMovingLeft) {
    position.setX(position.getX() - this.right.getX() * moveSpeed);
    position.setY(position.getY() - this.right.getY() * moveSpeed);
    position.setZ(position.getZ() - this.right.getZ() * moveSpeed);
}

if (this.isMovingRight) {
    position.setX(position.getX() + this.right.getX() * moveSpeed);
    position.setY(position.getY() + this.right.getY() * moveSpeed);
    position.setZ(position.getZ() + this.right.getZ() * moveSpeed);
}

if (this.isMovingUp) {
    position.setY(position.getY() + moveSpeed);
}

if (this.isMovingDown) {
    position.setY(position.getY() - moveSpeed);
}

this.newToOrigin.setXYZ(0, 0, 0);
Vector3.subtract(this.newToOrigin, position, this.newToOrigin);
Vector3.normalize(this.newToOrigin, this.newToOrigin);

    float yaw = (float) Math.toDegrees(Math.atan2(this.newToOrigin.getX(),
this.newToOrigin.getZ()));
float pitch = (float) Math.toDegrees(Math.asin(-this.newToOrigin.getY()));
```

```
        rotation.setX(pitch);
        rotation.setY(yaw);
        rotation.setZ(0);
    }

    @Override
    public void onDestroy() {
        this.buttonMoveLeft.setVisibility(View.INVISIBLE);
        this.buttonMoveRight.setVisibility(View.INVISIBLE);
        this.buttonMoveForward.setVisibility(View.INVISIBLE);
        this.buttonMoveBackward.setVisibility(View.INVISIBLE);
        this.buttonMoveUp.setVisibility(View.INVISIBLE);
        this.buttonMoveDown.setVisibility(View.INVISIBLE);
    }
}
```

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_INDEX_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_INDEX);

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.animationsexample.core.general;

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

```
import
com.labwork.animationsexample.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;
        }
```

```
        }
    }

    public void onUnloaded() {
        for (Entity entity: this.entities)
            entity.onDestroy();
    }
}
```

Shader.java

```
package com.labwork.animationsexample.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);
```

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

        GLES32.glLinkProgram(this.programId);
    }

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

    public Class<?> getRenderPass() {
        return this.renderPass;
    }

    public int getVariableHandler(String identifier) {
        return GLES32.glGetUniformLocation(this.programId, identifier);
    }

    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.animationsexample.rendering.passes.concrete;

import java.util.List;
import android.opengl.GLES32;
import com.labwork.animationsexample.runtime.Framework;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.general.Shader;
import com.labwork.animationsexample.rendering.passes.common.RenderPass;
import
com.labwork.animationsexample.core.components.concrete.CameraComponent;
import
com.labwork.animationsexample.core.components.concrete.RenderingComponent;

public final class OpaqueRenderPass extends RenderPass {

  public OpaqueRenderPass(Shader shader) {
     super(shader);
  }

  @Override
  public final void execute(List<Entity> dispatchedEntities) {
     CameraComponent camera = Framework.getInstance().getScene().getCamera();

     GLES32.glEnable(GLES32.GL_DEPTH_TEST);

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

```
        GLES32.glUseProgram(super.shader.getId());
        GLES32.glUniformMatrix4fv(super.shader.getVariableHandler("uMatrixView"),
1, false, camera.getMatrixView(), 0);

GLES32.glUniformMatrix4fv(super.shader.getVariableHandler("uMatrixProjection"),
1, false, camera.getMatrixProjection(), 0);

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

                                                    if (rendering == null ||
rendering.getMaterial().getShader(OpaqueRenderPass.class) == null)
                continue;

            rendering.render(OpaqueRenderPass.class);
        }

        GLES32.glUseProgram(0);

        GLES32.glDisable(GLES32.GL_DEPTH_TEST);
    }
}
```

SimpleProgrammableRenderer.java

```
package com.labwork.animationsexample.rendering.renderer.concrete;

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 com.labwork.animationsexample.runtime.Framework;
import com.labwork.animationsexample.core.general.Scene;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.rendering.passes.common.RenderPass;
import
com.labwork.animationsexample.rendering.renderer.common.RendererProgrammabl
e;

public final class SimpleProgrammableRenderer implements RendererProgrammable
{
  private final List<RenderPass> passes;
  private final List<Entity> dispatchedEntities;
  private final Runnable initializationCallback;

  private float deltaTime;
  private float timestampCurrent;
  private float timestampPrevious;

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

  public void onDrawFrame(GL10 unused) {
     this.timestampCurrent = System.nanoTime();
```

```
            this.deltaTime = (this.timestampCurrent - this.timestampPrevious) /
1_000_000_000.0f;
    this.timestampPrevious = this.timestampCurrent;

    if (this.deltaTime > 0.95f) {
      this.deltaTime = 0.95f;
    }

    if (Framework.getInstance().getScene() == null)
      return;

    this.dispatchedEntities.clear();

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

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

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

  public void onSurfaceCreated(GL10 unused, EGLConfig config) {
    this.initializationCallback.run();
    this.timestampPrevious = System.nanoTime();
```

```
    }

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

    public void loadScene(Scene scene) {
        List<Entity> entities = Framework.getInstance().getScene().getEntities();

        for (Entity entity : entities)
            entity.onStart();
    }

    public void registerRenderPass(RenderPass pass) {
        this.passes.add(pass);
    }
}
```

Viewport.java

```
package com.labwork.animationsexample.rendering.viewport.concrete;

import android.content.Context;
import android.opengl.GLSurfaceView;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.RelativeLayout.LayoutParams;
import
com.labwork.animationsexample.rendering.renderer.common.RendererProgrammabl
e;
```

```
import
com.labwork.animationsexample.rendering.viewport.common.ViewportConfigurable
;

public final class Viewport extends GLSurfaceView implements
ViewportConfigurable {
  private final RelativeLayout layout;

  public Viewport(Context context) {
     super(context);
     super.setEGLContextClientVersion(3);
     this.layout = new RelativeLayout(context);
      this.layout.addView(this, new LayoutParams(LayoutParams.MATCH_PARENT,
LayoutParams.MATCH_PARENT));
  }

  public RelativeLayout getLayout() {
     return this.layout;
  }

  public GLSurfaceView getSurfaceView() {
     return this;
  }

  public void register(View view) {
     this.layout.post(() -> {
        this.layout.addView(view);
     });
  }
```

```
  public void initialize(RendererProgrammable renderer) {
      super.setFocusable(true);
      super.setRenderer(renderer);
      super.setFocusableInTouchMode(true);
      super.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
  }
}
```

ViewportConfigurable.java

```
package com.labwork.animationsexample.rendering.viewport.common;

import android.view.View;
import android.widget.RelativeLayout;
import android.opengl.GLSurfaceView;
import
com.labwork.animationsexample.rendering.renderer.common.RendererProgrammabl
e;

public interface ViewportConfigurable {
  RelativeLayout getLayout();
  GLSurfaceView getSurfaceView();
  void register(View view);
  void initialize(RendererProgrammable renderer);
}
```

Framework.java

```
package com.labwork.animationsexample.runtime;

import android.opengl.GLSurfaceView;
```

```
import com.labwork.animationsexample.core.general.Scene;
import
com.labwork.animationsexample.rendering.renderer.common.RendererProgrammabl
e;
import
com.labwork.animationsexample.rendering.viewport.common.ViewportConfigurable
;

public final class Framework {
  private static final Framework INSTANCE = new Framework();

  private Scene scene;
  private GLSurfaceView surfaceView;
  private ViewportConfigurable viewport;
  private RendererProgrammable renderer;

  private Framework() { }

  public static Framework getInstance() {
     return Framework.INSTANCE;
  }

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

  public GLSurfaceView getSurfaceView() {
     return this.surfaceView;
  }
```

```
  public ViewportConfigurable getViewport() {
     return this.viewport;
  }

  public RendererProgrammable getRenderer() {
     return this.renderer;
  }

  public void loadScene(Scene scene) {
     if (this.scene != null)
        this.scene.onUnloaded();

     this.scene = scene;
     this.renderer.loadScene(scene);
  }

     public void initialize(RendererProgrammable renderer, ViewportConfigurable
viewport) {
     viewport.initialize(renderer);
     this.renderer = renderer;
     this.viewport = viewport;
     this.surfaceView = viewport.getSurfaceView();
  }
}
```

MainActivity.java

```
package com.labwork.animationsexample;

import android.opengl.GLES32;
```

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

import android.widget.Button;

import
com.labwork.animationsexample.core.components.concrete.CameraPerspectiveCom
ponent;
import
com.labwork.animationsexample.core.components.concrete.RenderingComponent;
import com.labwork.animationsexample.demo.components.RotationComponent;
import
com.labwork.animationsexample.core.components.concrete.TransformComponent;
import com.labwork.animationsexample.core.general.Color;
import com.labwork.animationsexample.core.general.Entity;
import com.labwork.animationsexample.core.general.Material;
import com.labwork.animationsexample.core.general.Mesh;
import com.labwork.animationsexample.core.general.Shader;
import
com.labwork.animationsexample.demo.components.FixedOrientationControllerCom
ponent;
import com.labwork.animationsexample.demo.shaders.Standalone;
import
com.labwork.animationsexample.rendering.passes.concrete.OpaqueRenderPass;
import
com.labwork.animationsexample.rendering.renderer.common.RendererProgrammabl
e;
```

```
import
com.labwork.animationsexample.rendering.renderer.concrete.SimpleProgrammableR
enderer;
import
com.labwork.animationsexample.rendering.viewport.common.ViewportConfigurable
;
import com.labwork.animationsexample.rendering.viewport.concrete.Viewport;
import com.labwork.animationsexample.runtime.Framework;

import com.labwork.animationsexample.core.general.Scene;
import
com.labwork.animationsexample.demo.components.NoClipControllerComponent;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
  private static final int MENU_ITEM_SCENE_CUBES = 1;
  private static final int MENU_ITEM_SCENE_PYRAMID = 2;

  private Shader shader;
  private Scene cubesScene;
  private Scene pyramidScene;

  @Override
  protected final void onCreate(Bundle savedInstanceState) {
     ViewportConfigurable viewport = new Viewport(this);
                                    RendererProgrammable renderer = new
SimpleProgrammableRenderer(this::initializeAssets);
```

```
        super.onCreate(savedInstanceState);
        super.setContentView(viewport.getLayout());
        Framework.getInstance().initialize(renderer, viewport);
    }

    private void initializeAssets() {
                        this.shader = new Shader(OpaqueRenderPass.class,
Standalone.SHADER_VERT_SOURCE, Standalone.SHADER_FRAG_SOURCE);
                    Framework.getInstance().getRenderer().registerRenderPass(new
OpaqueRenderPass(this.shader));
        this.pyramidScene = this.initializePyramidScene();
        this.cubesScene = this.initializeCubesScene();
    }

    private Scene initializeCubesScene() {
        Scene scene = new Scene();
        Material material = new Material(new Color(255, 255, 255, 0), this.shader);

        Entity chessboard = new Entity();
        chessboard.addComponent(new TransformComponent(chessboard));
            Mesh chessboardMesh = new Mesh(this.generateChessboardVertices(),
GLES32.GL_TRIANGLES);
                chessboard.addComponent(new RenderingComponent(chessboard,
chessboardMesh, material));
        scene.addEntity(chessboard);

        float spacing = 2.0f;
```

```
for (int x = 0; x < 3; x++) {
    for (int y = 0; y < 3; y++) {
        for (int z = 0; z < 3; z++) {
            Entity cube = new Entity();
            TransformComponent transform = new TransformComponent(cube);
            cube.addComponent(transform);
                    Mesh cubeMesh = new Mesh(this.generateCubeVertices(),
GLES32.GL_TRIANGLES);
                cube.addComponent(new RenderingComponent(cube, cubeMesh,
material));

            float startOffset = -spacing;
            transform.getPosition().setX(startOffset + x * spacing);
            transform.getPosition().setY(0.5f + y * spacing);
            transform.getPosition().setZ(startOffset + z * spacing);

            scene.addEntity(cube);
        }
    }
}

Entity camera = new Entity();
camera.addComponent(new TransformComponent(camera));
        camera.addComponent(new NoClipControllerComponent(camera, new
Button(this), new Button(this), new Button(this), new Button(this), new Button(this),
new Button(this), new Button(this), new Button(this)));
        camera.addComponent(new CameraPerspectiveComponent(camera, new
Color(27, 27, 27, 255), 0.001f, 100.0f, 90.0f, 90.0f));
camera.getComponent(TransformComponent.class).getPosition().setY(3.0f);
```

```
        camera.getComponent(TransformComponent.class).getPosition().setZ(-7.0f);
        scene.addEntity(camera);

        return scene;
    }

    private Scene initializePyramidScene() {
        Scene scene = new Scene();
        Material material = new Material(new Color(255, 255, 255, 0), this.shader);

        Entity pyramid = new Entity();
        pyramid.addComponent(new RotationComponent(pyramid));
        pyramid.addComponent(new TransformComponent(pyramid));
                Mesh pyramidMesh = new Mesh(this.generatePyramidVertices(),
GLES32.GL_TRIANGLES);
            pyramid.addComponent(new RenderingComponent(pyramid, pyramidMesh,
material));

        pyramid.getComponent(TransformComponent.class).getPosition().setY(0.5f);

        Entity chessboard = new Entity();
        chessboard.addComponent(new TransformComponent(chessboard));
              Mesh chessboardMesh = new Mesh(this.generateChessboardVertices(),
GLES32.GL_TRIANGLES);
                    chessboard.addComponent(new RenderingComponent(chessboard,
chessboardMesh, material));

        Entity camera = new Entity();
        camera.addComponent(new TransformComponent(camera));
```

```
        camera.addComponent(new FixedOrientationControllerComponent(camera, new Button(this), new Button(this), new Button(this), new Button(this), new Button(this), new Button(this)));
                camera.addComponent(new CameraPerspectiveComponent(camera, new Color(27, 27, 27, 255), 0.001f, 100.0f, 90.0f, 90.0f));
        camera.getComponent(TransformComponent.class).getPosition().setY(1.0f);
        camera.getComponent(TransformComponent.class).getPosition().setZ(-5.0f);

        scene.addEntity(camera);
        scene.addEntity(pyramid);
        scene.addEntity(chessboard);

        return scene;
    }

    private float[] generateCubeVertices() {
        return new float[] {
                // Front face
                -0.5f, -0.5f,  0.5f, 1.0f, 0.0f, 0.0f, 1.0f,
                0.5f, -0.5f,  0.5f, 1.0f, 0.0f, 0.0f, 1.0f,
                0.5f,  0.5f,  0.5f, 1.0f, 0.0f, 0.0f, 1.0f,
                0.5f,  0.5f,  0.5f, 1.0f, 0.0f, 0.0f, 1.0f,
                -0.5f,  0.5f,  0.5f, 1.0f, 0.0f, 0.0f, 1.0f,
                -0.5f, -0.5f,  0.5f, 1.0f, 0.0f, 0.0f, 1.0f,

                // Back face
                -0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f,
                0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f,
                0.5f,  0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f,
```

```
0.5f,  0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f,
-0.5f,  0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f,

// Left face
-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 1.0f,
-0.5f, -0.5f,  0.5f, 0.0f, 0.0f, 1.0f, 1.0f,
-0.5f,  0.5f,  0.5f, 0.0f, 0.0f, 1.0f, 1.0f,
-0.5f,  0.5f,  0.5f, 0.0f, 0.0f, 1.0f, 1.0f,
-0.5f,  0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 1.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 1.0f,

// Right face
0.5f, -0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f,
0.5f, -0.5f,  0.5f, 1.0f, 1.0f, 0.0f, 1.0f,
0.5f,  0.5f,  0.5f, 1.0f, 1.0f, 0.0f, 1.0f,
0.5f,  0.5f,  0.5f, 1.0f, 1.0f, 0.0f, 1.0f,
0.5f,  0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f,
0.5f, -0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f,

// Top face
-0.5f,  0.5f, -0.5f, 0.0f, 1.0f, 1.0f, 1.0f,
0.5f,  0.5f, -0.5f, 0.0f, 1.0f, 1.0f, 1.0f,
0.5f,  0.5f,  0.5f, 0.0f, 1.0f, 1.0f, 1.0f,
0.5f,  0.5f,  0.5f, 0.0f, 1.0f, 1.0f, 1.0f,
-0.5f,  0.5f,  0.5f, 0.0f, 1.0f, 1.0f, 1.0f,
-0.5f,  0.5f, -0.5f, 0.0f, 1.0f, 1.0f, 1.0f,

// Bottom face
```

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

private float[] generatePyramidVertices() {
    return new float[] {
            -1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  // Bottom-left
            1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  // Bottom-right
            1.0f, -1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  // Top-right
            1.0f, -1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  // Top-right
            -1.0f, -1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  // Top-left
            -1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  // Bottom-left

            -1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, 1.0f,  // Bottom-left
            1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, 1.0f,  // Bottom-right
            0.0f,  1.0f,  0.0f, 0.0f, 1.0f, 0.0f, 1.0f,  // Apex

            1.0f, -1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,  // Bottom-right
            1.0f, -1.0f,  1.0f, 0.0f, 0.0f, 1.0f, 1.0f,  // Top-right
            0.0f,  1.0f,  0.0f, 0.0f, 0.0f, 1.0f, 1.0f,  // Apex

            1.0f, -1.0f,  1.0f, 1.0f, 1.0f, 0.0f, 1.0f,  // Top-right
            -1.0f, -1.0f,  1.0f, 1.0f, 1.0f, 0.0f, 1.0f,  // Top-left
            0.0f,  1.0f,  0.0f, 1.0f, 1.0f, 0.0f, 1.0f,  // Apex
```

```
            -1.0f, -1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 0.0f,  // Top-left
            -1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 0.0f,  // Bottom-left
            0.0f,  1.0f,  0.0f, 1.0f, 0.0f, 0.0f, 0.0f   // Apex
    };
}

private float[] generateChessboardVertices() {
    List<Float> vertices = new ArrayList<>();

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            float r = (i + j) % 2 == 0 ? 1.0f : 0.0f;
            float g = (i + j) % 2 == 0 ? 1.0f : 0.0f;
            float b = (i + j) % 2 == 0 ? 1.0f : 0.0f;

            float y = 0.0f;
            float x1 = i - 4.0f;
            float x2 = i - 3.0f;
            float z1 = j - 4.0f;
            float z2 = j - 3.0f;

                    vertices.add(x1);  vertices.add(y);  vertices.add(z1);  vertices.add(r);
vertices.add(g); vertices.add(b); vertices.add(1.0f);
                    vertices.add(x2);  vertices.add(y);  vertices.add(z1);  vertices.add(r);
vertices.add(g); vertices.add(b); vertices.add(1.0f);
                    vertices.add(x2);  vertices.add(y);  vertices.add(z2);  vertices.add(r);
vertices.add(g); vertices.add(b); vertices.add(1.0f);
```

```
                    vertices.add(x2);  vertices.add(y);  vertices.add(z2);  vertices.add(r);
vertices.add(g); vertices.add(b); vertices.add(1.0f);
                    vertices.add(x1);  vertices.add(y);  vertices.add(z2);  vertices.add(r);
vertices.add(g); vertices.add(b); vertices.add(1.0f);
                    vertices.add(x1);  vertices.add(y);  vertices.add(z1);  vertices.add(r);
vertices.add(g); vertices.add(b); vertices.add(1.0f);
            }
        }

        float[] result = new float[vertices.size()];

        for (int i = 0; i < vertices.size(); i++) {
            result[i] = vertices.get(i);
        }

        return result;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, MainActivity.MENU_ITEM_SCENE_CUBES, 0, "Cubes");
        menu.add(0, MainActivity.MENU_ITEM_SCENE_PYRAMID, 0, "Pyramid");
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        super.setTitle(item.getTitle());
```

```
        switch (item.getItemId()) {
            case MainActivity.MENU_ITEM_SCENE_CUBES:
                Framework.getInstance().loadScene(this.cubesScene);
                return true;
            case MainActivity.MENU_ITEM_SCENE_PYRAMID:
                Framework.getInstance().loadScene(this.pyramidScene);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

ВИСНОВКИ

У цій лабораторній роботі було реалізовано два режими роботи: "Pyramid rotation" та "Nine Cubes". У першому режимі піраміда безперервно обертається над шаховим полем, а користувач може змінювати ракурс сцени, обертаючи камеру навколо вертикальної осі та змінюючи відстань до центру.

У режимі "Nine Cubes" сцена складається з шахового поля та решітки з 27 кубів. Користувач може переміщатися між кубами без зіткнення, змінюючи напрямок руху та нахил камери, що імітує політ літального апарата.

Було опрацьовано методи обробки сенсорного введення та використання різних режимів рендерингу. В результаті виконання роботи досягнуто поставлених завдань, отримано практичні навички програмування графіки в OpenGL ES.