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 № 5 з дисципліни
«Програмування комп'ютерної графіки»

«**Використання текстур**»

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 текстур тривимірних об’єктів засобами графіки OpenGL ES.

Завдання:

1. Застосунок **Lab5_GLES** для вибору режиму роботи повинен мати меню з трьома пунктами:
   - Torus
   - Earth
   - Seaview
2. Меню має забезпечувати вибір потрібного режиму роботи

3. У режимі Torus потрібно запрограмувати рендеринг текстурованого тора та чотирикутника шахової дошки.

Рисунок 1.1 – Завдання лабораторної роботи

4. У режимі Earth потрібно запрограмувати текстуроване зображення планети.

Рисунок 1.2 – Завдання лабораторного практикуму

5. У режимі Seaview запрограмувати куб Skybox. Центр кубу – у центрі системи координат. Усередині кубу Skybox розташувати тор із шаховою дошкою (п. 3).

Рисунок 1.3 – Завдання лабораторного практикуму

6. В усіх режимах потрібно забезпечити керування ракурсом показу. Потрібно зробити керування поворотами по горизонталі та вертикалі а також для наближення-віддалення (збільшення-зменшення).

Рисунок 1.4 – Завдання лабораторного практикуму

Рисунок 1.5 – Отриманий рендер планети

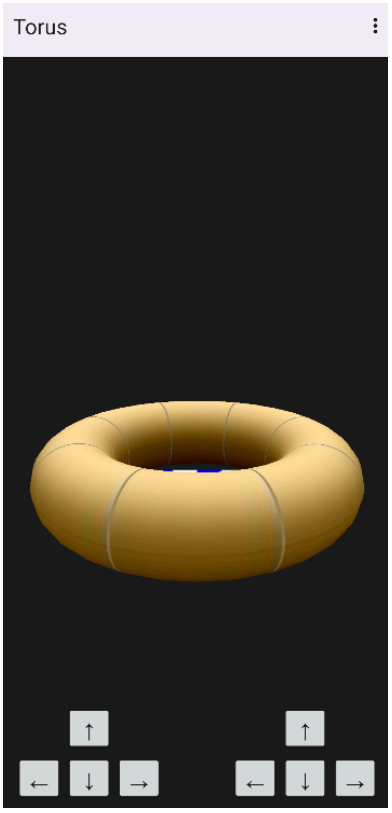

Рисунок 1.6 – Отриманий рендер торуса

Рисунок 1.7 – Отриманий рендер торуса зі скайбоксом

Component.java

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

import com.labwork.texturesexample.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(float deltaTime) {}

    public void onDestroy() {}
}
```

CameraComponent.java

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

import android.opengl.Matrix;
import com.labwork.texturesexample.core.general.Color;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.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;
    }
}
```

CameraPerspectiveComponent.java

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

import android.opengl.GLES32;
import android.opengl.Matrix;
import android.opengl.GLSurfaceView;
import com.labwork.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Color;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.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();
    }

    @Override
    public void onUpdate(float deltaTime) {
        this.setAspectRatio((float)this.viewport.getWidth() / this.viewport.getHeight());
                        GLES32.glClearColor(super.backgroundColor.getRNormalized(),
super.backgroundColor.getGNormalized(),
super.backgroundColor.getBNormalized(),
super.backgroundColor.getANormalized());

                    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());
    }
}
```

OpaqueRenderingComponent.java

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

import android.opengl.GLES32;
import com.labwork.texturesexample.core.general.Mesh;
import com.labwork.texturesexample.core.general.Color;
import com.labwork.texturesexample.core.general.Entity;
```

```
import com.labwork.texturesexample.core.general.Shader;
import com.labwork.texturesexample.core.general.Material;
import com.labwork.texturesexample.core.components.common.Component;

public final class OpaqueRenderingComponent extends Component {
  private static final int TEXTURE_UNIT_INDEX_OPAQUE = 0;

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

  public OpaqueRenderingComponent(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() {
    Shader shader = this.material.getShader();

      GLES32.glUniformMatrix4fv(shader.getVariableHandler("uMatrixModel"), 1, false, this.transform.getMatrixModel(), 0);

    Color color = this.material.getColor();
        GLES32.glUniform4f(shader.getVariableHandler("uMaterialColorRGBA"), color.getRNormalized(), color.getGNormalized(), color.getBNormalized(), color.getANormalized());

                            GLES32.glBindTexture(GLES32.GL_TEXTURE_2D, this.material.getTextureAlbedo().getId());
                GLES32.glUniform1i(shader.getVariableHandler("uTextureAlbedo"), OpaqueRenderingComponent.TEXTURE_UNIT_INDEX_OPAQUE);

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

SkyboxRenderingComponent.java

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

import android.opengl.GLES32;
import com.labwork.texturesexample.core.general.Mesh;
import com.labwork.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Color;
import com.labwork.texturesexample.core.general.Shader;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.core.general.Material;
import com.labwork.texturesexample.core.general.Texture3D;
import com.labwork.texturesexample.core.components.common.Component;

public final class SkyboxRenderingComponent extends Component {
  private static final int TEXTURE_UNIT_INDEX_SKYBOX = 0;

  private final Mesh mesh;
  private final Texture3D cubeTexture;

  private Material material;
  private TransformComponent transform;

   public SkyboxRenderingComponent(Entity entity, Mesh mesh, Material material,
Texture3D cubeTexture) {
      super(entity);
      this.mesh = mesh;
      this.material = material;
      this.cubeTexture = cubeTexture;
  }
```

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

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

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

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

  public void render() {
     Shader shader = this.material.getShader();

        GLES32.glUniformMatrix4fv(shader.getVariableHandler("uMatrixModel"), 1,
false, this.transform.getMatrixModel(), 0);

     Color color = this.material.getColor();
           GLES32.glUniform4f(shader.getVariableHandler("uMaterialColorRGBA"),
color.getRNormalized(), color.getGNormalized(), color.getBNormalized(),
color.getANormalized());
```

```
                GLES32.glBindTexture(GLES32.GL_TEXTURE_CUBE_MAP,
this.cubeTexture.getId());
        GLES32.glUniform1i(shader.getVariableHandler("uCubemapTextureAlbedo"),
SkyboxRenderingComponent.TEXTURE_UNIT_INDEX_SKYBOX);

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

TransformComponent.java

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

import android.opengl.Matrix;
import com.labwork.texturesexample.core.general.Axis;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.core.general.Vector3;
import com.labwork.texturesexample.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 void setScale(Vector3 value) {
    this.scale.setX(value.getX());
    this.scale.setY(value.getY());
    this.scale.setZ(value.getZ());
}

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

public void setRotation(Vector3 value) {
    this.rotation.setX(value.getX());
    this.rotation.setY(value.getY());
    this.rotation.setZ(value.getZ());
}

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

public void setPosition(Vector3 value) {
    this.position.setX(value.getX());
    this.position.setY(value.getY());
    this.position.setZ(value.getZ());
}

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.getZ(), 0.0f, 0.0f, 1.0f);
      Matrix.rotateM(this.matrixModel, 0, this.rotation.getY(), 0.0f, 1.0f, 0.0f);
      Matrix.rotateM(this.matrixModel, 0, this.rotation.getX(), 1.0f, 0.0f, 0.0f);
       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()]);
      Vector3.normalize(this.vectorUp, this.vectorUp);
      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()]);
      Vector3.normalize(this.vectorRight, this.vectorRight);
      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()]);
    Vector3.normalize(this.vectorForward, this.vectorForward);
    return this.vectorForward;
  }

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

Axis.java

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

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

Color.java

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

import java.util.Map;
import java.util.HashMap;
import java.util.Collection;
import com.labwork.texturesexample.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(float deltaTime) {
        for (Component component : this.components.values())
            component.onUpdate(deltaTime);
    }

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

Material.java

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

public final class Material {
   private Color color;
   private Shader shader;
   private float propertyAmbient;
   private float propertyDiffuse;
   private float propertySpecular;
   private Texture2D textureAlbedo;


      public Material(Shader shader, Color color, Texture2D textureAlbedo, float propertyAmbient, float propertyDiffuse, float propertySpecular) {
      this.color = color;
      this.shader = shader;
      this.textureAlbedo = textureAlbedo;
      this.propertyAmbient = propertyAmbient;
      this.propertyDiffuse = propertyDiffuse;
      this.propertySpecular = propertySpecular;
   }

   public Color getColor() {
      return this.color;
   }

   public void setColor(Color value) {
      this.color= value;
   }
```

```
public Shader getShader() {
    return this.shader;
}

public void setShader(Shader value) {
    this.shader = value;
}

public Texture2D getTextureAlbedo() {
    return this.textureAlbedo;
}

public void setTextureAlbedo(Texture2D value) {
    this.textureAlbedo = value;
}

public float getPropertyAmbient() {
    return this.propertyAmbient;
}

public void setPropertyAmbient(float value) {
    this.propertyAmbient = value;
}

public float getPropertyDiffuse() {
    return this.propertyDiffuse;
}
```

```
  public void setPropertyDiffuse(float value) {
    this.propertyDiffuse = value;
  }

  public float getPropertySpecular() {
    return this.propertySpecular;
  }

  public void setPropertySpecular(float value) {
    this.propertySpecular = value;
  }
}
```

Mesh.java

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

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

public final class Mesh {
  private static final int HANDLERS_COUNT = 3;

  private static final int HANDLER_INDEX_VAO = 0;
  private static final int HANDLER_INDEX_VBO = 1;
  private static final int HANDLER_INDEX_EBO = 2;

  public static final int PAYLOAD_POSITION_SIZE = 3;
```

```
  public static final int PAYLOAD_POSITION_INDEX = 0;
  public static final int PAYLOAD_POSITION_OFFSET = 0;

  public static final int PAYLOAD_TEXTURE_SIZE = 2;
  public static final int PAYLOAD_TEXTURE_INDEX = 1;
        public static final int PAYLOAD_TEXTURE_OFFSET =
(Mesh.PAYLOAD_POSITION_SIZE) * Float.BYTES;

  public static final int PAYLOAD_NORMAL_SIZE = 3;
  public static final int PAYLOAD_NORMAL_INDEX = 2;
        public static final int PAYLOAD_NORMAL_OFFSET =
(Mesh.PAYLOAD_POSITION_SIZE + Mesh.PAYLOAD_TEXTURE_SIZE) *
Float.BYTES;

  public static final int PAYLOAD_COLOR_SIZE = 3;
  public static final int PAYLOAD_COLOR_INDEX = 3;
        public static final int PAYLOAD_COLOR_OFFSET =
(Mesh.PAYLOAD_POSITION_SIZE + Mesh.PAYLOAD_TEXTURE_SIZE +
Mesh.PAYLOAD_NORMAL_SIZE) * Float.BYTES;

  public static final int PAYLOAD_STRIDE = (Mesh.PAYLOAD_POSITION_SIZE
+ Mesh.PAYLOAD_TEXTURE_SIZE + Mesh.PAYLOAD_NORMAL_SIZE +
Mesh.PAYLOAD_COLOR_SIZE) * Float.BYTES;

  private final int[] handlers;
  private final int drawingMode;
  private final int indicesCount;

  public Mesh(float[] vertices, int[] indices, int drawingMode) {
```

```
        this.drawingMode = drawingMode;
        this.indicesCount = indices.length;
        this.handlers = new int[Mesh.HANDLERS_COUNT];

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

                IntBuffer indexBuffer = ByteBuffer.allocateDirect(indices.length *
Integer.BYTES).order(ByteOrder.nativeOrder()).asIntBuffer();
        indexBuffer.put(indices).position(0);

        GLES32.glGenVertexArrays(1, this.handlers, Mesh.HANDLER_INDEX_VAO);
        GLES32.glGenBuffers(1, this.handlers, Mesh.HANDLER_INDEX_VBO);
        GLES32.glGenBuffers(1, this.handlers, Mesh.HANDLER_INDEX_EBO);

        GLES32.glBindVertexArray(this.handlers[Mesh.HANDLER_INDEX_VAO]);

                                GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER,
this.handlers[Mesh.HANDLER_INDEX_VBO]);
                GLES32.glBufferData(GLES32.GL_ARRAY_BUFFER, vertices.length *
Float.BYTES, vertexBuffer, GLES32.GL_STATIC_DRAW);

                        GLES32.glBindBuffer(GLES32.GL_ELEMENT_ARRAY_BUFFER,
this.handlers[Mesh.HANDLER_INDEX_EBO]);
                        GLES32.glBufferData(GLES32.GL_ELEMENT_ARRAY_BUFFER,
indices.length * Integer.BYTES, indexBuffer, GLES32.GL_STATIC_DRAW);
```

```
            GLES32.glVertexAttribPointer(Mesh.PAYLOAD_POSITION_INDEX,
Mesh.PAYLOAD_POSITION_SIZE,  GLES32.GL_FLOAT,  false,
Mesh.PAYLOAD_STRIDE, Mesh.PAYLOAD_POSITION_OFFSET);
    GLES32.glEnableVertexAttribArray(Mesh.PAYLOAD_POSITION_INDEX);

            GLES32.glVertexAttribPointer(Mesh.PAYLOAD_TEXTURE_INDEX,
Mesh.PAYLOAD_TEXTURE_SIZE,  GLES32.GL_FLOAT,  false,
Mesh.PAYLOAD_STRIDE, Mesh.PAYLOAD_TEXTURE_OFFSET);
    GLES32.glEnableVertexAttribArray(Mesh.PAYLOAD_TEXTURE_INDEX);

            GLES32.glVertexAttribPointer(Mesh.PAYLOAD_NORMAL_INDEX,
Mesh.PAYLOAD_NORMAL_SIZE,  GLES32.GL_FLOAT,  false,
Mesh.PAYLOAD_STRIDE, Mesh.PAYLOAD_NORMAL_OFFSET);
    GLES32.glEnableVertexAttribArray(Mesh.PAYLOAD_NORMAL_INDEX);

              GLES32.glVertexAttribPointer(Mesh.PAYLOAD_COLOR_INDEX,
Mesh.PAYLOAD_COLOR_SIZE,  GLES32.GL_FLOAT,  false,
Mesh.PAYLOAD_STRIDE, Mesh.PAYLOAD_COLOR_OFFSET);
    GLES32.glEnableVertexAttribArray(Mesh.PAYLOAD_COLOR_INDEX);

    GLES32.glBindVertexArray(0);
    GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER, 0);
    GLES32.glBindBuffer(GLES32.GL_ELEMENT_ARRAY_BUFFER, 0);
  }

  public int getId() {
    return this.handlers[Mesh.HANDLER_INDEX_VAO];
  }
```

```
  public void draw() {
    GLES32.glBindVertexArray(this.handlers[Mesh.HANDLER_INDEX_VAO]);
                GLES32.glDrawElements(this.drawingMode,  this.indicesCount,
GLES32.GL_UNSIGNED_INT, 0);
    GLES32.glBindVertexArray(0);
  }

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

Scene.java

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

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

public final class Scene {
  private final List<Entity> entities;

  private CameraComponent camera;
  private SkyboxRenderingComponent skybox;
```

```
public Scene() {
    this.entities = new ArrayList<>();
}

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

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

public SkyboxRenderingComponent getSkybox() {
    return this.skybox;
}

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;
        }
        if (component instanceof SkyboxRenderingComponent) {
            this.skybox = (SkyboxRenderingComponent) component;
        }
    }
```

```
    }

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

Shader.java

```
package com.labwork.texturesexample.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<?> renderFeature;

    public Shader(Class<?> renderFeature, String sourceVert, String sourceFrag) {
        this.renderFeature = renderFeature;
        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_COLOR_INDEX, "aVertexColorRGB");
                                        GLES32.glBindAttribLocation(this.programId,
Mesh.PAYLOAD_NORMAL_INDEX, "aVertexNormalLocal");
                                        GLES32.glBindAttribLocation(this.programId,
Mesh.PAYLOAD_POSITION_INDEX, "aVertexPositionLocal");
                                        GLES32.glBindAttribLocation(this.programId,
Mesh.PAYLOAD_TEXTURE_INDEX, "aVertexTextureCoordinate");

        GLES32.glLinkProgram(this.programId);
    }

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

    public Class<?> getRenderFeature() {
        return this.renderFeature;
    }

    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);
  }
}
```

Texture2D.java

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

import android.opengl.GLES32;
import android.opengl.GLUtils;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;

public final class Texture2D {
  private static final int HANDLERS_COUNT = 1;
  private static final int HANDLER_INDEX_ID = 0;

  private final int[] handlers;

  public Texture2D(Context context, int resourceId, int wrap, int filter) {
    this.handlers = new int[Texture2D.HANDLERS_COUNT];

    GLES32.glGenTextures(1, this.handlers, 0);

    final BitmapFactory.Options options = new BitmapFactory.Options();
```

```
        options.inScaled = false;

        final Bitmap bitmap = BitmapFactory.decodeResource(context.getResources(),
resourceId, options);

                                GLES32.glBindTexture(GLES32.GL_TEXTURE_2D,
this.handlers[Texture2D.HANDLER_INDEX_ID]);

                                GLES32.glTexParameteri(GLES32.GL_TEXTURE_2D,
GLES32.GL_TEXTURE_WRAP_S, wrap);
                                GLES32.glTexParameteri(GLES32.GL_TEXTURE_2D,
GLES32.GL_TEXTURE_WRAP_T, wrap);
                                GLES32.glTexParameteri(GLES32.GL_TEXTURE_2D,
GLES32.GL_TEXTURE_MIN_FILTER, filter);
                                GLES32.glTexParameteri(GLES32.GL_TEXTURE_2D,
GLES32.GL_TEXTURE_MAG_FILTER, filter);

        GLUtils.texImage2D(GLES32.GL_TEXTURE_2D, 0, bitmap, 0);

        bitmap.recycle();

        GLES32.glBindTexture(GLES32.GL_TEXTURE_2D, 0);
    }

    public int getId() {
        return this.handlers[Texture2D.HANDLER_INDEX_ID];
    }
}
```

Texture3D.java

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

import android.opengl.GLES32;
import android.opengl.GLUtils;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;

public final class Texture3D {
  private static final int HANDLERS_COUNT = 1;
  private static final int HANDLER_INDEX_ID = 0;

  private final int[] handlers;

  public Texture3D(Context context, int[] resourceIds, int wrap, int filter) {
     this.handlers = new int[Texture3D.HANDLERS_COUNT];

     GLES32.glGenTextures(1, this.handlers, 0);
     GLES32.glBindTexture(GLES32.GL_TEXTURE_CUBE_MAP,
this.handlers[Texture3D.HANDLER_INDEX_ID]);

     final BitmapFactory.Options options = new BitmapFactory.Options();
     options.inScaled = false;

     int[] cubeTargets = {
           GLES32.GL_TEXTURE_CUBE_MAP_POSITIVE_X, // Right
           GLES32.GL_TEXTURE_CUBE_MAP_NEGATIVE_X, // Left
           GLES32.GL_TEXTURE_CUBE_MAP_POSITIVE_Y, // Top
           GLES32.GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, // Bottom
           GLES32.GL_TEXTURE_CUBE_MAP_POSITIVE_Z, // Front
           GLES32.GL_TEXTURE_CUBE_MAP_NEGATIVE_Z  // Back
     };

     for (int i = 0; i < 6; ++i) {
        final Bitmap bitmap =
BitmapFactory.decodeResource(context.getResources(), resourceIds[i], options);
        GLUtils.texImage2D(cubeTargets[i], 0, bitmap, 0);
        bitmap.recycle();
     }

     GLES32.glTexParameteri(GLES32.GL_TEXTURE_CUBE_MAP,
GLES32.GL_TEXTURE_WRAP_S, wrap);
```

```
        GLES32.glTexParameteri(GLES32.GL_TEXTURE_CUBE_MAP,
GLES32.GL_TEXTURE_WRAP_T, wrap);
        GLES32.glTexParameteri(GLES32.GL_TEXTURE_CUBE_MAP,
GLES32.GL_TEXTURE_WRAP_R, wrap);
        GLES32.glTexParameteri(GLES32.GL_TEXTURE_CUBE_MAP,
GLES32.GL_TEXTURE_MIN_FILTER, filter);
        GLES32.glTexParameteri(GLES32.GL_TEXTURE_CUBE_MAP,
GLES32.GL_TEXTURE_MAG_FILTER, filter);

        GLES32.glBindTexture(GLES32.GL_TEXTURE_CUBE_MAP, 0);
    }

    public int getId() {
        return this.handlers[Texture3D.HANDLER_INDEX_ID];
    }
}
```

Vector3.java

```
package com.labwork.texturesexample.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 void setXYZ(float x, float y, float z) {
    this.x = x;
    this.y = y;
    this.z = z;
}

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

DynamicLightControllerComponent.java

```
package com.labwork.texturesexample.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.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.core.general.Vector3;
import com.labwork.texturesexample.core.components.common.Component;
import
com.labwork.texturesexample.core.components.concrete.TransformComponent;

public final class DynamicLightControllerComponent 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;

        public DynamicLightControllerComponent(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();
         float moveSpeed = DynamicLightControllerComponent.MOVEMENT_SPEED
* deltaTime;

        if (this.isMovingForward) {
            position.setZ(position.getZ() + moveSpeed);
        }
        if (this.isMovingBackward) {
            position.setZ(position.getZ() - moveSpeed);
        }
        if (this.isMovingLeft) {
            position.setX(position.getX() + moveSpeed);
        }
        if (this.isMovingRight) {
            position.setX(position.getX() - moveSpeed);
        }
        if (this.isMovingUp) {
            position.setY(position.getY() + moveSpeed);
        }
        if (this.isMovingDown) {
            position.setY(position.getY() - moveSpeed);
        }
    }

    @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);
    }
}
```

NoClipControllerComponent.java

```
package com.labwork.texturesexample.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.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.core.general.Vector3;
import com.labwork.texturesexample.core.components.common.Component;
import
com.labwork.texturesexample.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 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);

    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;

    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);
    }

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

    if (this.isMovingLeft) {
```

```
                    Vector3.add(this.moveDirection,  this.transform.getRight(),
this.moveDirection);
    }
    if (this.isMovingRight) {
                    Vector3.subtract(this.moveDirection,  this.transform.getRight(),
this.moveDirection);
    }
    if (this.isMovingForward) {
                    Vector3.add(this.moveDirection,  this.transform.getForward(),
this.moveDirection);
    }
    if (this.isMovingBackward) {
                    Vector3.subtract(this.moveDirection,  this.transform.getForward(),
this.moveDirection);
    }

    if (this.moveDirection.getMagnitude() > 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);
    }
}
```

Standalone.java

```
package com.labwork.texturesexample.demo.shaders;

public final class Standalone {

    public static final String SHADER_OPAQUE_VERT_SOURCE =
            "#version 300 es\n" +

            "in vec3 aVertexColorRGB;\n" +
            "in vec3 aVertexNormalLocal;\n" +
            "in vec3 aVertexPositionLocal;\n" +
            "in vec2 aVertexTextureCoordinate;\n" +

            "uniform mat4 uMatrixView;\n" +
            "uniform mat4 uMatrixModel;\n" +
            "uniform mat4 uMatrixProjection;\n" +

             "out vec3 vVertexColorRGB;\n" +
             "out vec2 vVertexTextureCoordinate;\n" +

             "void main() {\n" +
```

```
            "    gl_Position = uMatrixProjection * uMatrixView * uMatrixModel *
vec4(aVertexPositionLocal, 1.0);\n" +

        "    vVertexColorRGB = aVertexColorRGB;\n" +
        "    vVertexTextureCoordinate = aVertexTextureCoordinate;\n" +
        "}\n";

  public static final String SHADER_OPAQUE_FRAG_SOURCE =
        "#version 300 es\n" +
        "precision mediump float;\n" +

        "in vec3 vVertexColorRGB;\n" +
        "in vec2 vVertexTextureCoordinate;\n" +

        "uniform vec4 uMaterialColorRGBA;\n" +
        "uniform sampler2D uTextureAlbedo;\n" +

        "out vec4 outFragmentColor;\n" +

        "void main() {\n" +
                        "        outFragmentColor = texture(uTextureAlbedo,
vVertexTextureCoordinate);\n" +
        "}\n";

  public static final String SHADER_SKYBOX_VERT_SOURCE =
        "#version 300 es\n" +

        "in vec3 aVertexColorRGB;\n" +
        "in vec3 aVertexNormalLocal;\n" +
```

```
        "in vec3 aVertexPositionLocal;\n" +
        "in vec2 aVertexTextureCoordinate;\n" +

        "uniform mat4 uMatrixView;\n" +
        "uniform mat4 uMatrixModel;\n" +
        "uniform mat4 uMatrixProjection;\n" +

        "out vec3 vTextureCoordinate;\n" +

        "void main() {\n" +
        "    mat4 viewNoTranslation = mat4(mat3(uMatrixView));\n" +
          "     gl_Position = uMatrixProjection * viewNoTranslation * uMatrixModel *
vec4(aVertexPositionLocal, 1.0);\n" +

        "    vTextureCoordinate = aVertexPositionLocal;\n" +
        "}\n";

  public static final String SHADER_SKYBOX_FRAG_SOURCE =
        "#version 300 es\n" +
        "precision mediump float;\n" +

        "in vec3 vTextureCoordinate;\n" +

        "uniform samplerCube uCubeTexture;\n" +

        "out vec4 outFragmentColor;\n" +

        "void main() {\n" +
        "    outFragmentColor = texture(uCubeTexture, vTextureCoordinate);\n" +
```

```
        "}\n";
}
```

RenderFeature.java

```
package com.labwork.texturesexample.rendering.features.common;

import java.util.List;
import com.labwork.texturesexample.core.general.Shader;
import com.labwork.texturesexample.core.general.Entity;

public abstract class RenderFeature {
  protected Shader shader;

  public RenderFeature(Shader shader) {
     this.shader = shader;
  }

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

OpaqueRenderPass.java

```
package com.labwork.texturesexample.rendering.features.concrete;

import java.util.List;
import android.opengl.GLES32;
import com.labwork.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.core.general.Shader;
import com.labwork.texturesexample.rendering.features.common.RenderFeature;
import com.labwork.texturesexample.core.components.concrete.CameraComponent;
```

```
import
com.labwork.texturesexample.core.components.concrete.OpaqueRenderingCompone
nt;

public final class OpaqueRenderFeature extends RenderFeature {

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

  @Override
  public final void execute(List<Entity> dispatchedEntities) {
     GLES32.glEnable(GLES32.GL_DEPTH_TEST);

     GLES32.glUseProgram(super.shader.getId());
     GLES32.glActiveTexture(GLES32.GL_TEXTURE0);

     CameraComponent camera = Framework.getInstance().getScene().getCamera();
      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) {
                                        OpaqueRenderingComponent rendering =
entity.getComponent(OpaqueRenderingComponent.class);

        if (rendering == null)
```

```
            continue;

                        if (rendering.getMaterial().getShader().getRenderFeature() ==
OpaqueRenderFeature.class) {
            rendering.render();
        }
    }

    GLES32.glUseProgram(0);

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

SkyboxRednerFetature.java

```
package com.labwork.texturesexample.rendering.features.concrete;

import java.util.List;
import android.opengl.GLES32;
import com.labwork.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.core.general.Shader;
import com.labwork.texturesexample.rendering.features.common.RenderFeature;
import com.labwork.texturesexample.core.components.concrete.CameraComponent;
import
com.labwork.texturesexample.core.components.concrete.SkyboxRenderingCompone
nt;

public final class SkyboxRenderFeature extends RenderFeature {
```

```
  public SkyboxRenderFeature(Shader shader) {
     super(shader);
  }

  @Override
  public final void execute(List<Entity> dispatchedEntities) {
                                GLES32.glClear(GLES32.GL_COLOR_BUFFER_BIT     |
GLES32.GL_DEPTH_BUFFER_BIT);

     GLES32.glDepthMask(false);

     GLES32.glUseProgram(super.shader.getId());
     GLES32.glActiveTexture(GLES32.GL_TEXTURE0);

     CameraComponent camera = Framework.getInstance().getScene().getCamera();
      GLES32.glUniformMatrix4fv(super.shader.getVariableHandler("uMatrixView"),
1, false, camera.getMatrixView(), 0);

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

                                        SkyboxRenderingComponent      skybox      =
Framework.getInstance().getScene().getSkybox();

     if (skybox != null) {
        skybox.render();
     }

     GLES32.glUseProgram(0);
```

```
        GLES32.glDepthMask(true);
    }
}
```

RenderProgrammable.java

```
package com.labwork.texturesexample.rendering.renderer.common;

import android.opengl.GLSurfaceView.Renderer;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import com.labwork.texturesexample.core.general.Scene;
import com.labwork.texturesexample.rendering.features.common.RenderFeature;

public interface RendererProgrammable extends Renderer {
    void onDrawFrame(GL10 unused);
    void onSurfaceCreated(GL10 unused, EGLConfig config);
    void onSurfaceChanged(GL10 unused, int width, int height);
    void loadScene(Scene scene);
    void registerRenderFeature(RenderFeature feature);
}
```

ForwardRenderer.java

```
package com.labwork.texturesexample.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.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Scene;
import com.labwork.texturesexample.core.general.Entity;
import com.labwork.texturesexample.rendering.features.common.RenderFeature;
import
com.labwork.texturesexample.rendering.renderer.common.RendererProgrammable;

public final class ForwardRenderer implements RendererProgrammable {
  private final List<RenderFeature> features;
  private final List<Entity> dispatchedEntities;
  private final Runnable initializationCallback;

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

  public ForwardRenderer(Runnable initializationCallback) {
    this.features = 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 (RenderFeature feature : this.features)
            feature.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 registerRenderFeature(RenderFeature feature) {
     this.features.add(feature);
  }
}
```

ViewportConfigurable.java

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

import android.view.View;
import android.widget.RelativeLayout;
import android.opengl.GLSurfaceView;
import
com.labwork.texturesexample.rendering.renderer.common.RendererProgrammable;

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

Viewport.java

```
package com.labwork.texturesexample.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.texturesexample.rendering.renderer.common.RendererProgrammable;
import
com.labwork.texturesexample.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);
  }
}
```

Framework.java

```
package com.labwork.texturesexample.runtime;

import android.opengl.GLSurfaceView;
import com.labwork.texturesexample.core.general.Scene;
import
com.labwork.texturesexample.rendering.renderer.common.RendererProgrammable;
import
com.labwork.texturesexample.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.texturesexample;

import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Button;
import android.opengl.GLES32;
import androidx.appcompat.app.AppCompatActivity;
import com.labwork.texturesexample.runtime.Framework;
import com.labwork.texturesexample.core.general.Mesh;
import com.labwork.texturesexample.core.general.Color;
import com.labwork.texturesexample.core.general.Scene;
import com.labwork.texturesexample.core.general.Entity;
```

```
import com.labwork.texturesexample.core.general.Shader;
import com.labwork.texturesexample.core.general.Texture2D;
import com.labwork.texturesexample.core.general.Texture3D;
import com.labwork.texturesexample.core.general.Material;
import
com.labwork.texturesexample.demo.components.NoClipControllerComponent;
import
com.labwork.texturesexample.core.components.concrete.TransformComponent;
import
com.labwork.texturesexample.core.components.concrete.SkyboxRenderingCompone
nt;
import
com.labwork.texturesexample.core.components.concrete.OpaqueRenderingCompone
nt;
import
com.labwork.texturesexample.core.components.concrete.CameraPerspectiveCompon
ent;
import com.labwork.texturesexample.demo.shaders.Standalone;
import com.labwork.texturesexample.rendering.renderer.concrete.ForwardRenderer;
import
com.labwork.texturesexample.rendering.renderer.common.RendererProgrammable;
import
com.labwork.texturesexample.rendering.features.concrete.OpaqueRenderFeature;
import
com.labwork.texturesexample.rendering.features.concrete.SkyboxRenderFeature;
import com.labwork.texturesexample.rendering.viewport.concrete.Viewport;
import
com.labwork.texturesexample.rendering.viewport.common.ViewportConfigurable;
```

```
public class MainActivity extends AppCompatActivity {
  private static final int MENU_ITEM_SCENE_EARTH = 1;
  private static final int MENU_ITEM_SCENE_TORUS = 2;
  private static final int MENU_ITEM_SCENE_SKYBOX = 3;

  private Scene earthScene;
  private Scene torusScene;
  private Scene skyboxScene;
  private Shader opaqueShader;
  private Shader skyboxShader;

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

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

  private void initializeAssets() {
                this.opaqueShader = new Shader(OpaqueRenderFeature.class,
Standalone.SHADER_OPAQUE_VERT_SOURCE,
Standalone.SHADER_OPAQUE_FRAG_SOURCE);
                this.skyboxShader = new Shader(SkyboxRenderFeature.class,
Standalone.SHADER_SKYBOX_VERT_SOURCE,
Standalone.SHADER_SKYBOX_FRAG_SOURCE);
```

```
                Framework.getInstance().getRenderer().registerRenderFeature(new
SkyboxRenderFeature(this.skyboxShader));
                Framework.getInstance().getRenderer().registerRenderFeature(new
OpaqueRenderFeature(this.opaqueShader));

    this.earthScene = this.initializeEarthScene();
    this.torusScene = this.initializeTorusScene();
    this.skyboxScene = this.initializeSkyboxScene();
  }

  private Scene initializeEarthScene() {
    Scene scene = new Scene();
     Material material = new Material(this.opaqueShader, new Color(255, 255, 255,
0), new Texture2D(this, R.drawable.earth, GLES32.GL_CLAMP_TO_EDGE,
GLES32.GL_LINEAR), 0.3f, 0.7f, 0.0f);

    Entity sphere = new Entity();
    sphere.addComponent(new TransformComponent(sphere));
                Mesh sphereMesh = new Mesh(this.generateEarthVertices(),
this.generateEarthIndices(), GLES32.GL_TRIANGLES);
     sphere.addComponent(new OpaqueRenderingComponent(sphere, sphereMesh,
material));
    scene.addEntity(sphere);

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

```
            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)));
    scene.addEntity(camera);

    camera.getComponent(TransformComponent.class).getPosition().setY(1.5f);
    camera.getComponent(TransformComponent.class).getPosition().setZ(-3.0f);
    camera.getComponent(TransformComponent.class).getRotation().setX(15.0f);

    return scene;
  }

  private Scene initializeTorusScene() {
    Scene scene = new Scene();

     Material torusMaterial = new Material(this.opaqueShader, new Color(255, 255,
255, 0), new Texture2D(this, R.drawable.torus, GLES32.GL_CLAMP_TO_EDGE,
GLES32.GL_LINEAR), 0.3f, 0.7f, 0.0f);
    Entity torus = new Entity();
    torus.addComponent(new TransformComponent(torus));
                Mesh torusMesh = new Mesh(this.generateTorusVertices(),
this.generateTorusIndices(), GLES32.GL_TRIANGLES);
          torus.addComponent(new OpaqueRenderingComponent(torus, torusMesh,
torusMaterial));
    scene.addEntity(torus);
```

```
        Material planeMaterial = new Material(this.opaqueShader, new Color(255, 255, 255, 0), new Texture2D(this, R.drawable.chess, GLES32.GL_REPEAT, GLES32.GL_LINEAR), 0.3f, 0.7f, 0.0f);
        Entity plane = new Entity();
        plane.addComponent(new TransformComponent(plane));
                        Mesh planeMesh = new Mesh(this.generatePlaneVertices(), this.generatePlaneIndices(), GLES32.GL_TRIANGLES);
            plane.addComponent(new OpaqueRenderingComponent(plane, planeMesh, planeMaterial));
        scene.addEntity(plane);

        plane.getComponent(TransformComponent.class).getPosition().setY(-0.5f);

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

        camera.getComponent(TransformComponent.class).getPosition().setY(1.5f);
        camera.getComponent(TransformComponent.class).getPosition().setZ(-3.0f);
        camera.getComponent(TransformComponent.class).getRotation().setX(15.0f);

        return scene;
    }
```

```
private Scene initializeSkyboxScene() {
    Scene scene = new Scene();

     Material torusMaterial = new Material(this.opaqueShader, new Color(255, 255, 255, 0), new Texture2D(this, R.drawable.torus, GLES32.GL_CLAMP_TO_EDGE, GLES32.GL_LINEAR), 0.3f, 0.7f, 0.0f);
    Entity torus = new Entity();
    torus.addComponent(new TransformComponent(torus));
                Mesh torusMesh = new Mesh(this.generateTorusVertices(), this.generateTorusIndices(), GLES32.GL_TRIANGLES);
        torus.addComponent(new OpaqueRenderingComponent(torus, torusMesh, torusMaterial));
    scene.addEntity(torus);

     Material planeMaterial = new Material(this.opaqueShader, new Color(255, 255, 255, 0), new Texture2D(this, R.drawable.chess, GLES32.GL_REPEAT, GLES32.GL_LINEAR), 0.3f, 0.7f, 0.0f);
    Entity plane = new Entity();
    plane.addComponent(new TransformComponent(plane));
                Mesh planeMesh = new Mesh(this.generatePlaneVertices(), this.generatePlaneIndices(), GLES32.GL_TRIANGLES);
        plane.addComponent(new OpaqueRenderingComponent(plane, planeMesh, planeMaterial));
    scene.addEntity(plane);
    plane.getComponent(TransformComponent.class).getPosition().setY(-0.5f);

    int[] skyboxTextures = {
        R.drawable.skybox_right,
        R.drawable.skybox_left,
```

```
            R.drawable.skybox_top,
            R.drawable.skybox_bottom,
            R.drawable.skybox_front,
            R.drawable.skybox_back
        };

                Texture3D skyboxTexture = new Texture3D(this, skyboxTextures, GLES32.GL_CLAMP_TO_EDGE, GLES32.GL_LINEAR);
            Material skyboxMaterial = new Material(this.skyboxShader, new Color(255, 255, 255, 255), null, 0.0f, 0.0f, 0.0f);
                    Mesh skyboxMesh = new Mesh(generateSkyboxCubeVertices(), generateSkyboxCubeIndices(), GLES32.GL_TRIANGLES);
        Entity skybox = new Entity();
        skybox.addComponent(new TransformComponent(skybox));
        skybox.addComponent(new SkyboxRenderingComponent(skybox, skyboxMesh, skyboxMaterial, skyboxTexture));
        scene.addEntity(skybox);

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

        camera.getComponent(TransformComponent.class).getPosition().setY(1.5f);
        camera.getComponent(TransformComponent.class).getPosition().setZ(-3.0f);
```

```
    camera.getComponent(TransformComponent.class).getRotation().setX(15.0f);

    return scene;
}

private float[] generateSkyboxCubeVertices() {
    int index = 0;
    float size = 50.0f;
    float[] vertices = new float[24 * 11];

    vertices[index++] = -size; vertices[index++] = -size; vertices[index++] = size;
    vertices[index++] = 0.0f; vertices[index++] = 0.0f;
    vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = 1.0f;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

    vertices[index++] = size; vertices[index++] = -size; vertices[index++] = size;
    vertices[index++] = 1.0f; vertices[index++] = 0.0f;
    vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = 1.0f;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

    vertices[index++] = size; vertices[index++] = size; vertices[index++] = size;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f;
    vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = 1.0f;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

    vertices[index++] = -size; vertices[index++] = size; vertices[index++] = size;
    vertices[index++] = 0.0f; vertices[index++] = 1.0f;
    vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = 1.0f;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;
```

```
vertices[index++] = -size; vertices[index++] = -size; vertices[index++] = -size;
vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = -1.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = -size; vertices[index++] = -size;
vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = -1.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = size; vertices[index++] = -size;
vertices[index++] = 1.0f; vertices[index++] = 1.0f;
vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = -1.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = size; vertices[index++] = -size;
vertices[index++] = 0.0f; vertices[index++] = 1.0f;
vertices[index++] = 0.0f; vertices[index++] = 0.0f; vertices[index++] = -1.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = -size; vertices[index++] = -size;
vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = -1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = -size; vertices[index++] = size;
vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = -1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
```

```
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = size; vertices[index++] = size;
vertices[index++] = 1.0f; vertices[index++] = 1.0f;
vertices[index++] = -1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = size; vertices[index++] = -size;
vertices[index++] = 0.0f; vertices[index++] = 1.0f;
vertices[index++] = -1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = -size; vertices[index++] = size;
vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = -size; vertices[index++] = -size;
vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = size; vertices[index++] = -size;
vertices[index++] = 1.0f; vertices[index++] = 1.0f;
vertices[index++] = 1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = size; vertices[index++] = size;
vertices[index++] = 0.0f; vertices[index++] = 1.0f;
```

```
vertices[index++] = 1.0f; vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = size; vertices[index++] = size;
vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 0.0f; vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = size; vertices[index++] = size;
vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = 0.0f; vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = size; vertices[index++] = -size;
vertices[index++] = 1.0f; vertices[index++] = 1.0f;
vertices[index++] = 0.0f; vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = size; vertices[index++] = -size;
vertices[index++] = 0.0f; vertices[index++] = 1.0f;
vertices[index++] = 0.0f; vertices[index++] = 1.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = -size; vertices[index++] = -size; vertices[index++] = -size;
vertices[index++] = 0.0f; vertices[index++] = 0.0f;
vertices[index++] = 0.0f; vertices[index++] = -1.0f; vertices[index++] = 0.0f;
vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

vertices[index++] = size; vertices[index++] = -size; vertices[index++] = -size;
```

```
    vertices[index++] = 1.0f; vertices[index++] = 0.0f;
    vertices[index++] = 0.0f; vertices[index++] = -1.0f; vertices[index++] = 0.0f;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

    vertices[index++] = size; vertices[index++] = -size; vertices[index++] = size;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f;
    vertices[index++] = 0.0f; vertices[index++] = -1.0f; vertices[index++] = 0.0f;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

    vertices[index++] = -size; vertices[index++] = -size; vertices[index++] = size;
    vertices[index++] = 0.0f; vertices[index++] = 1.0f;
    vertices[index++] = 0.0f; vertices[index++] = -1.0f; vertices[index++] = 0.0f;
    vertices[index++] = 1.0f; vertices[index++] = 1.0f; vertices[index++] = 1.0f;

    return vertices;
}

private int[] generateSkyboxCubeIndices() {
    return new int[] {
            0, 1, 2,  0, 2, 3,      // Front
            4, 5, 6,  4, 6, 7,      // Back
            8, 9, 10, 8, 10, 11,    // Left
            12, 13, 14, 12, 14, 15, // Right
            16, 17, 18, 16, 18, 19, // Top
            20, 21, 22, 20, 22, 23  // Bottom
    };
}

private float[] generateEarthVertices() {
```

```
int latitudeBands = 32;
int longitudeBands = 64;
float radius = 1.0f;

int vertexCount = (latitudeBands + 1) * (longitudeBands + 1);
float[] vertices = new float[vertexCount * 11];

int index = 0;
for (int lat = 0; lat <= latitudeBands; lat++) {
    float theta = lat * (float)Math.PI / latitudeBands;
    float sinTheta = (float)Math.sin(theta);
    float cosTheta = (float)Math.cos(theta);

    for (int lon = 0; lon <= longitudeBands; lon++) {
        float phi = lon * 2 * (float)Math.PI / longitudeBands;
        float sinPhi = (float)Math.sin(phi);
        float cosPhi = (float)Math.cos(phi);

        float x = cosPhi * sinTheta;
        float y = cosTheta;
        float z = sinPhi * sinTheta;

        float u = 1.0f - ((float)lon / longitudeBands);
        float v = (float)lat / latitudeBands;

        vertices[index++] = x * radius;
        vertices[index++] = y * radius;
        vertices[index++] = z * radius;
        vertices[index++] = u;
```

```
            vertices[index++] = v;
            vertices[index++] = x;
            vertices[index++] = y;
            vertices[index++] = z;
            vertices[index++] = 1.0f;
            vertices[index++] = 1.0f;
            vertices[index++] = 1.0f;
        }
    }

    return vertices;
}

private int[] generateEarthIndices() {
    int latitudeBands = 32;
    int longitudeBands = 64;
    int indexCount = latitudeBands * longitudeBands * 6;
    int[] indices = new int[indexCount];
    int index = 0;

    for (int lat = 0; lat < latitudeBands; lat++) {
        for (int lon = 0; lon < longitudeBands; lon++) {
            int first = (lat * (longitudeBands + 1)) + lon;
            int second = first + longitudeBands + 1;

            indices[index++] = first;
            indices[index++] = second;
            indices[index++] = first + 1;
```

```
            indices[index++] = second;
            indices[index++] = second + 1;
            indices[index++] = first + 1;
        }
    }
    return indices;
}

private float[] generateTorusVertices() {
    int radialSegments = 32;
    int tubularSegments = 64;
    float radius = 1.5f;
    float tubeRadius = 0.5f;

    int vertexCount = (radialSegments + 1) * (tubularSegments + 1);
    float[] vertices = new float[vertexCount * 11];
    int index = 0;

    for (int i = 0; i <= radialSegments; i++) {
        float u = i / (float)radialSegments * (float)Math.PI * 2;
        float cosU = (float)Math.cos(u);
        float sinU = (float)Math.sin(u);

        for (int j = 0; j <= tubularSegments; j++) {
            float v = j / (float)tubularSegments * (float)Math.PI * 2;
            float cosV = (float)Math.cos(v);
            float sinV = (float)Math.sin(v);

            float x = (radius + tubeRadius * cosV) * cosU;
```

```
            float y = tubeRadius * sinV;
            float z = (radius + tubeRadius * cosV) * sinU;

            float nx = cosV * cosU;
            float ny = sinV;
            float nz = cosV * sinU;

            float texU = (float)i / radialSegments;
            float texV = (float)j / tubularSegments;

            vertices[index++] = x;
            vertices[index++] = y;
            vertices[index++] = z;
            vertices[index++] = texU;
            vertices[index++] = texV;
            vertices[index++] = nx;
            vertices[index++] = ny;
            vertices[index++] = nz;
            vertices[index++] = 1.0f;
            vertices[index++] = 1.0f;
            vertices[index++] = 1.0f;
        }
    }

    return vertices;
}

private int[] generateTorusIndices() {
    int radialSegments = 32;
```

```
    int tubularSegments = 64;
    int indexCount = radialSegments * tubularSegments * 6;
    int[] indices = new int[indexCount];
    int index = 0;

    for (int i = 0; i < radialSegments; i++) {
        for (int j = 0; j < tubularSegments; j++) {
            int a = (tubularSegments + 1) * i + j;
            int b = (tubularSegments + 1) * (i + 1) + j;
            int c = (tubularSegments + 1) * (i + 1) + j + 1;
            int d = (tubularSegments + 1) * i + j + 1;

            indices[index++] = a;
            indices[index++] = b;
            indices[index++] = d;

            indices[index++] = b;
            indices[index++] = c;
            indices[index++] = d;
        }
    }

    return indices;
}

private float[] generatePlaneVertices() {
    float size = 2.25f;
    float repeatFactor = 1.5f;
```

```
int index = 0;
float[] vertices = new float[4 * 11];

vertices[index++] = -size / 2;
vertices[index++] = 0.0f;
vertices[index++] = -size / 2;
vertices[index++] = 0.0f;
vertices[index++] = 0.0f;
vertices[index++] = 0.0f;
vertices[index++] = 1.0f;
vertices[index++] = 0.0f;
vertices[index++] = 1.0f;
vertices[index++] = 1.0f;
vertices[index++] = 1.0f;
vertices[index++] = size / 2;
vertices[index++] = 0.0f;
vertices[index++] = -size / 2;
vertices[index++] = repeatFactor * 2;
vertices[index++] = 0.0f;
vertices[index++] = 0.0f;
vertices[index++] = 1.0f;
vertices[index++] = 0.0f;
vertices[index++] = 1.0f;
vertices[index++] = 1.0f;
vertices[index++] = 1.0f;
vertices[index++] = size / 2;
vertices[index++] = 0.0f;
vertices[index++] = size / 2;
vertices[index++] = repeatFactor * 2;
```

```
    vertices[index++] = repeatFactor * 2;
    vertices[index++] = 0.0f;
    vertices[index++] = 1.0f;
    vertices[index++] = 0.0f;
    vertices[index++] = 1.0f;
    vertices[index++] = 1.0f;
    vertices[index++] = 1.0f;
    vertices[index++] = -size / 2;
    vertices[index++] = 0.0f;
    vertices[index++] = size / 2;
    vertices[index++] = 0.0f;
    vertices[index++] = repeatFactor * 2;
    vertices[index++] = 0.0f;
    vertices[index++] = 1.0f;
    vertices[index++] = 0.0f;
    vertices[index++] = 1.0f;
    vertices[index++] = 1.0f;
    vertices[index++] = 1.0f;
    return vertices;
}

private int[] generatePlaneIndices() {
    int[] indices = new int[6];
    indices[0] = 0; // Bottom-left
    indices[1] = 1; // Bottom-right
    indices[2] = 2; // Top-right
    indices[3] = 0; // Bottom-left
    indices[4] = 2; // Top-right
    indices[5] = 3; // Top-left
```

```
        return indices;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, MENU_ITEM_SCENE_EARTH, 0, "Earth");
        menu.add(0, MENU_ITEM_SCENE_TORUS, 0, "Torus");
        menu.add(0, MENU_ITEM_SCENE_SKYBOX, 0, "Skybox");
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        super.setTitle(item.getTitle());
        switch (item.getItemId()) {
            case MENU_ITEM_SCENE_EARTH:
                Framework.getInstance().loadScene(this.earthScene);
                return true;
            case MENU_ITEM_SCENE_TORUS:
                Framework.getInstance().loadScene(this.torusScene);
                return true;
            case MENU_ITEM_SCENE_SKYBOX:
                Framework.getInstance().loadScene(this.skyboxScene);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

ВИСНОВКИ

У процесі виконання лабораторної роботи №5 створено застосунок Lab5_GLES із трьома режимами роботи: рендеринг текстурованого тору разом із чотирикутником шахової дошки, кулі з текстурою планети на основі космічних знімків та куба Skybox із тором усередині для створення ефекту навколишнього середовища. У кожному режимі реалізовано інтерактивне керування сценою – повороти по горизонталі й вертикалі та зміну відстані до об’єктів, що забезпечило гнучкість огляду. Робота охопила використання текстурних координат, циклічного повторення зображень та оптимізацію рендерингу.

Було опрацьовано ключові аспекти текстурування в OpenGL ES: завантаження текстур, створення атласів, програмування шейдерів для обробки вершин і фрагментів, а також налагодження програми на емуляторі та фізичному пристрої Android. Виконання завдань дозволило поглибити розуміння принципів роботи з текстурами та здобути практичні навички програмування реалістичної графіки тривимірних об’єктів, досягнувши поставленої мети лабораторної роботи.