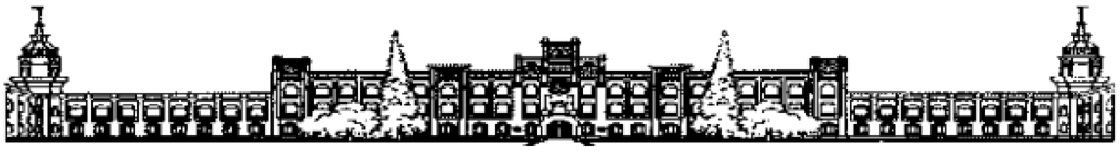


MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Victor Porev

Computer Graphics Programming

Laboratory Work 4

Lighting modeling

Kulubecioglu Mehmet

IM-14 FIOT

Class Number: 12

Kyiv

IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE

2025

1. Introduction

This project demonstrates the creation and lighting of various 3D objects (such as cylinders, toruses, spheres, pyramids, and a chessboard-like plane) on Android using OpenGL ES 3.0+. The project implements the Phong lighting model, combining ambient, diffuse, and specular components, and allows interactive control of the light source's position via touch events. In addition, multiple scene modes (e.g., CustomCylinderMode, CustomTorusMode, ShapesRenderer, Pyramid mode, and Nine Cubes mode) are provided, each displaying a different set of 3D geometries and lighting effects.

2. Project Objectives and Scope

- **3D Object Creation:**
  - Generate geometry for cylinders, toruses, spheres, pyramids, and a chessboard-like plane using programmatically computed vertex data.
- **Lighting Model:**
  - Apply the Phong lighting model by calculating ambient, diffuse, and specular components to achieve realistic shading.
  - Enable interactive control of the light source's position through touch events.
- **User Interaction:**
  - Allow the user to change the light's position and adjust the scene by touch and drag events.
  - Provide a menu for selecting different scene modes (CustomCylinderMode, CustomTorusMode, ShapesRenderer, Pyramid, Nine Cubes).

3. Theoretical Background

3.1 OpenGL ES and Shaders

- **OpenGL ES: A graphics API used for hardware-accelerated graphics on mobile devices.**
- **Vertex Shader:** Processes each vertex by applying transformations (model, view, and projection matrices) to compute its final position.
- **Fragment Shader:** Calculates the color of each pixel (fragment) based on lighting and texture data; the Phong lighting model is implemented in the fragment shader.

3.2 Phong Lighting Model

The Phong model considers three main components of light:

1. **Ambient Light:** Represents the general low-level illumination present in the scene.
2. **Diffuse Light (Lambertian):** Calculated as the dot product between the surface normal (N) and the light direction (L), i.e., max(N·L,0)\max(N \cdot L, 0)max(N · L,0).
3. **Specular Light:** Produces highlights on shiny surfaces based on the dot product between the reflected light vector (R) and the view vector (V), raised to a power (p) for shininess: max(R·V,0)p\max(R \cdot V, 0)^pmax(R · V,0)p.

The total illumination is computed by summing these components.

3.3 Touch Interaction

- **ACTION_DOWN:** The initial touch on the screen records the starting coordinates.
- **ACTION_MOVE:** As the finger drags, the light source's (or camera's) position is updated based on the movement, resulting in real-time changes in scene illumination.

4. Project Architecture and Code Structure

4.1 Main Classes

- ❖ **BaseRendererMode:**
  - ➢ Serves as the base class that handles shader compilation, VAO/VBO creation, and basic touch event handling.
  - ➢ Contains methods such as **`useProgramForDrawing()`**, **`bindVertexArray()`**, and **`compileAndAttachShaders()`**.
- ❖ **LampDiffuseSpecMode (extends BaseRendererMode):**
  - ➢ Adds variables for light color, object color, and light position.
  - ➢ Implements the Phong lighting model by setting uniform variables (vLightPos, vEyePos, vLightColor, etc.).
  - ➢ Contains the **`drawLampSymbol()`** method to render the light source as a point (lamp symbol).
- ❖ **CustomCylinderMode, CustomTorusMode, ShapesRenderer, Pyramid Mode, Nine Cubes Mode:**
  - ➢ Each mode's **`initScene()`** method creates the vertex data for the specific geometries.
  - ➢ In their **`drawScene()`** (or **`useProgramForDrawing()`**) methods, they first render the main objects (with vColorMode = 1) and then call **`drawLampSymbol()`** (with vColorMode = 0) to draw the light source point.

4.2 GraphicsPrimitives Class

This class provides helper methods for generating vertex data for various geometries:

- addCylinderWithNormal(): Generates vertex and normal data for a cylinder based on radius, height, and segment count.
- **addTorusWithNormal():** Computes vertex data for a torus using majorRadius, minorRadius, and segment counts.
- **addSphereWithNormal():** Creates a sphere using latitude/longitude segmentation.
- **addChessBoardWithNormal():** Creates a chessboard-like plane by adding quads in a grid pattern.
- **addPointWithNormal():** Adds a single point (used for the light source symbol).

Each method uses a two-pass approach:

- In the first pass (when the array parameter is null), it computes the total number of floats required.
- In the second pass, the actual vertex data is written.

4.3 Light Source (Lamp Symbol)

- The light source is added as the first object in the scene (index 0) using **`addPointWithNormal()`**.
- The **`drawLampSymbol()`** method in LampDiffuseSpecMode renders this point by setting **`vColorMode`** to 0, resetting the model matrix (using **`Matrix.setIdentityM()`**), applying a translation to the light's position, and then drawing the object with index 0.
- This method ensures that the light source appears as a clearly visible point (typically white) regardless of the lighting calculations applied to other objects.

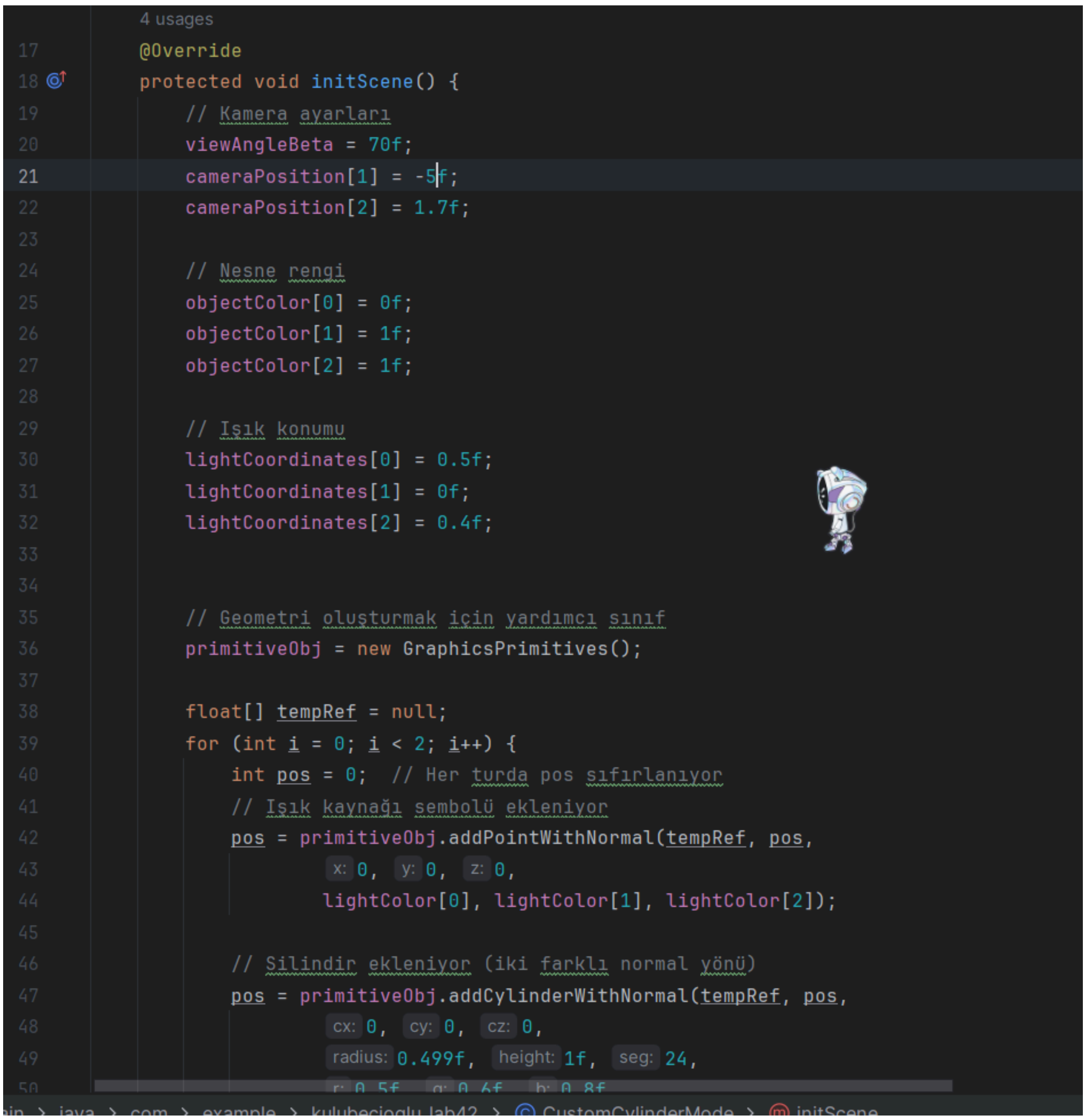
4.4 Touch Events

- The **`onTouchDown()`** method records the initial touch coordinates and adjusts the light's z-coordinate if the touch occurs near the screen edges.

❖ The **`onTouchMove()`** method updates the light's x and y coordinates based on the movement, enabling interactive repositioning of the light source.

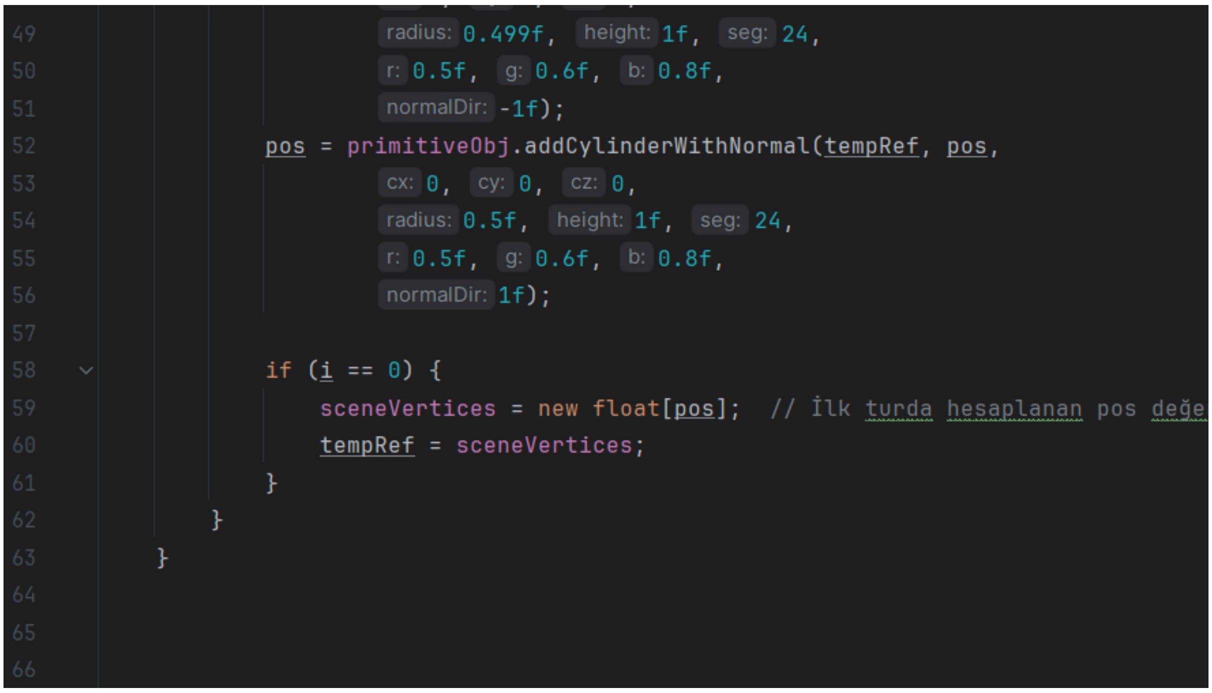
5. Code Samples

5.1. initScene() in ShapesRenderer



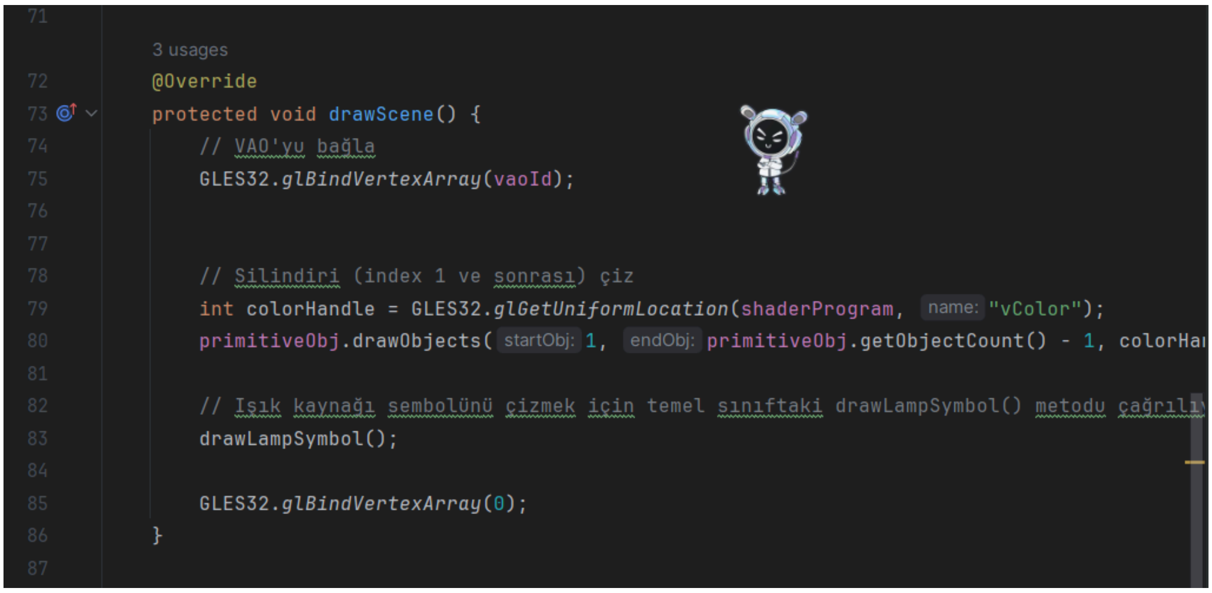
```
4 usages
@Override
protected void initScene() {
    // Kamera ayarları
    viewAngleBeta = 70f;
    cameraPosition[1] = -5f;
    cameraPosition[2] = 1.7f;

    // Nesne rengi
    objectColor[0] = 0f;
    objectColor[1] = 1f;
    objectColor[2] = 1f;

    // Işık konumu
    lightCoordinates[0] = 0.5f;
    lightCoordinates[1] = 0f;
    lightCoordinates[2] = 0.4f;


    // Geometri oluşturmak için yardımcı sınıf
    primitiveObj = new GraphicsPrimitives();

    float[] tempRef = null;
    for (int i = 0; i < 2; i++) {
        int pos = 0;  // Her turda pos sıfırlanıyor
        // Işık kaynağı sembolü ekleniyor
        pos = primitiveObj.addPointWithNormal(tempRef, pos,
                x: 0, y: 0, z: 0,
                lightColor[0], lightColor[1], lightColor[2]);

        // Silindir ekleniyor (iki farklı normal yönü)
        pos = primitiveObj.addCylinderWithNormal(tempRef, pos,
                cx: 0, cy: 0, cz: 0,
                radius: 0.499f, height: 1f, seg: 24,
                r: 0.5f, g: 0.4f, b: 0.8f
```

ain > java > com > example > kulubecioglu.lab42 > CustomCylinderMode > initScene



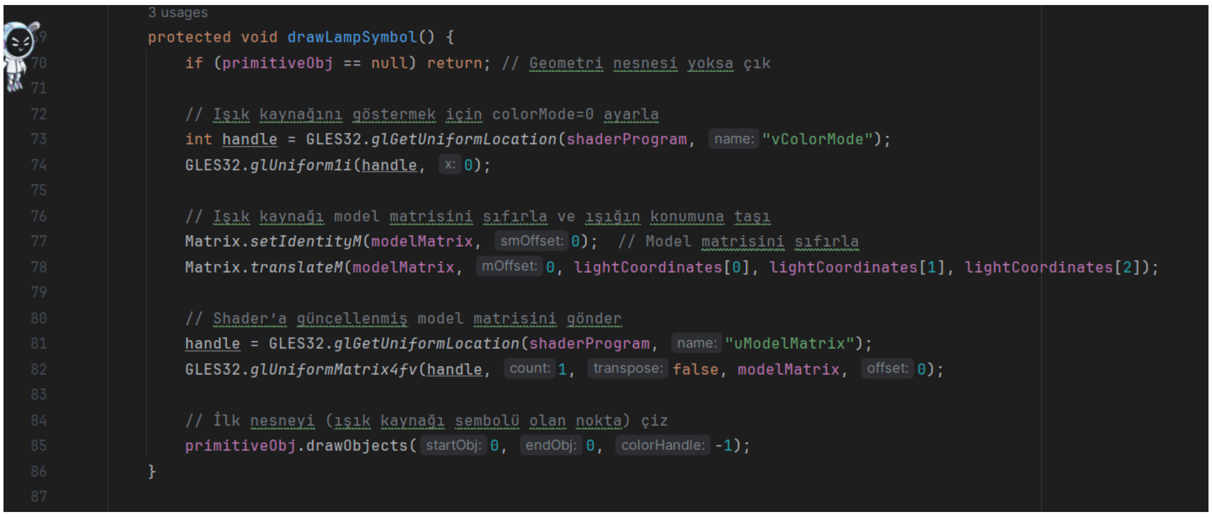
```
                    radius: 0.499f, height: 1f, seg: 24,
                    r: 0.5f, g: 0.6f, b: 0.8f,
                    normalDir: -1f);
            pos = primitiveObj.addCylinderWithNormal(tempRef, pos,
                    cx: 0, cy: 0, cz: 0,
                    radius: 0.5f, height: 1f, seg: 24,
                    r: 0.5f, g: 0.6f, b: 0.8f,
                    normalDir: 1f);

            if (i == 0) {
                sceneVertices = new float[pos];  // İlk turda hesaplanan pos değe
                tempRef = sceneVertices;
            }
        }
    }
```

5.2. drawScene() and Light Source Rendering



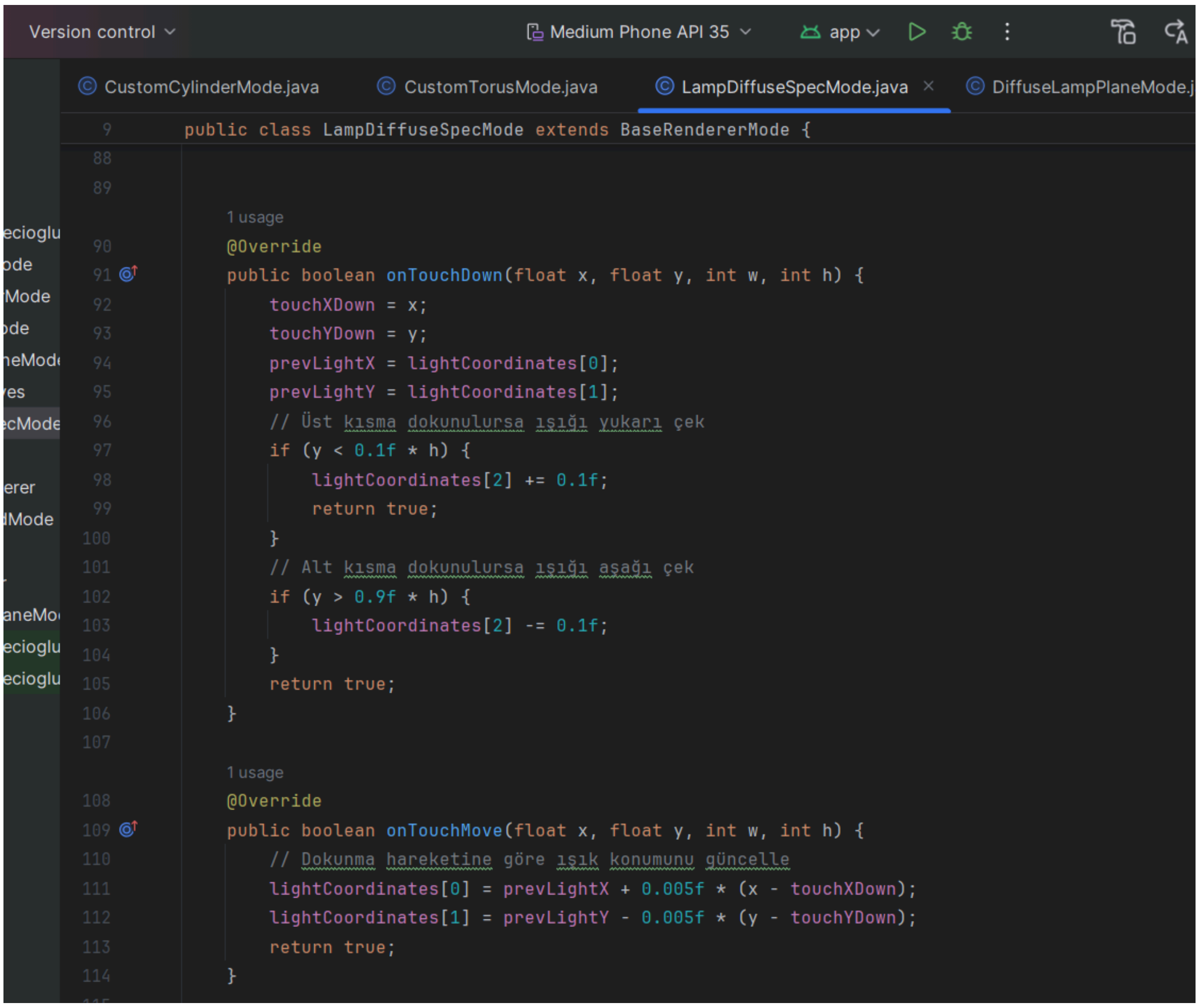
```
    3 usages
    @Override
    protected void drawScene() {
        // VAO'yu bağla
        GLES32.glBindVertexArray(vaoId);


        // Silindiri (index 1 ve sonrası) çiz
        int colorHandle = GLES32.glGetUniformLocation(shaderProgram, name: "vColor");
        primitiveObj.drawObjects( startObj: 1, endObj: primitiveObj.getObjectCount() - 1, colorHa

        // Işık kaynağı sembolünü çizmek için temel sınıftaki drawLampSymbol() metodu çağrılı
        drawLampSymbol();

        GLES32.glBindVertexArray(0);
    }
```

5.3. drawLampSymbol() in LampDiffuseSpecMode



```java
protected void drawLampSymbol() {
    if (primitiveObj == null) return; // Geometri nesnesi yoksa çık

    // Işık kaynağını göstermek için colorMode=0 ayarla
    int handle = GLES32.glGetUniformLocation(shaderProgram, "vColorMode");
    GLES32.glUniform1i(handle, 0);

    // Işık kaynağı model matrisini sıfırla ve ışığın konumuna taşı
    Matrix.setIdentityM(modelMatrix, 0);  // Model matrisini sıfırla
    Matrix.translateM(modelMatrix, 0, lightCoordinates[0], lightCoordinates[1], lightCoordinates[2]);

    // Shader'a güncellenmiş model matrisini gönder
    handle = GLES32.glGetUniformLocation(shaderProgram, "uModelMatrix");
    GLES32.glUniformMatrix4fv(handle, 1, false, modelMatrix, 0);

    // İlk nesneyi (ışık kaynağı sembolü olan nokta) çiz
    primitiveObj.drawObjects( 0, 0, -1);
}
```

5.4. Touch Events



```java
public class LampDiffuseSpecMode extends BaseRendererMode {


    @Override
    public boolean onTouchDown(float x, float y, int w, int h) {
        touchXDown = x;
        touchYDown = y;
        prevLightX = lightCoordinates[0];
        prevLightY = lightCoordinates[1];
        // Üst kısma dokunulursa ışığı yukarı çek
        if (y < 0.1f * h) {
            lightCoordinates[2] += 0.1f;
            return true;
        }
        // Alt kısma dokunulursa ışığı aşağı çek
        if (y > 0.9f * h) {
            lightCoordinates[2] -= 0.1f;
        }
        return true;
    }

    @Override
    public boolean onTouchMove(float x, float y, int w, int h) {
        // Dokunma hareketine göre ışık konumunu güncelle
        lightCoordinates[0] = prevLightX + 0.005f * (x - touchXDown);
        lightCoordinates[1] = prevLightY - 0.005f * (y - touchYDown);
        return true;
    }
```

7. Conclusion and Evaluation

This project successfully demonstrates the creation of multiple 3D objects using OpenGL ES on Android. By implementing the Phong lighting model (ambient, diffuse, specular), realistic shading effects are achieved. The interactive control of the light source allows users to dynamically change the lighting conditions, resulting in observable differences in specular highlights and diffuse shading across objects.

Key achievements include:

- Implementation of a modular rendering architecture using BaseRendererMode and derived classes.
- Generation of vertex data for complex objects such as cylinders, toruses, spheres, pyramids, and chessboards.
- Application of the Phong lighting model using shader programs.
- Interactive light control via touch events.
- A consistent and aesthetically pleasing color palette for various objects in the scene.

Future improvements may include:

- Applying texture mapping to objects for enhanced realism.
- Supporting multiple light sources.
- Implementing shadow mapping to add depth and realism.

Control Questions and Answers

1. **What is a light reflection model?**
   A light reflection model is a mathematical model used in computer graphics to simulate the way light interacts with surfaces. It describes how light reflects off a surface and contributes to the perceived color and brightness of that surface. Common reflection models include the **Phong reflection model**, **Blinn-Phong**, and **Lambertian (diffuse) reflection**. These models typically incorporate ambient, diffuse, and specular reflection components.

2. **How is the color of an object calculated for diffuse light reflection?**
   In diffuse light reflection, the object's color is determined by the **Lambert's cosine law**, which states that the intensity of light reflected from a surface is proportional to the cosine of the angle between the **light direction vector** and the **surface normal vector**.
   The formula:
   ```
   I_diffuse = max(dot(N, L), 0.0) * LightColor * ObjectColor;
   ```

where:

a. `N` = normal vector
b. `L` = direction vector from surface to light
c. `dot(N, L)` = cosine of the angle between them
d. `LightColor` = color/intensity of the light
e. `ObjectColor` = base color of the object

3. **What determines the color of an object for specular light reflection?**
Specular reflection depends on the viewer's position. It simulates the shiny highlights on an object. The specular color is calculated based on the **angle between the view direction and the reflection direction of the light ray**.
The formula:
```
I_specular = pow(max(dot(R, V), 0.0), shininess) * LightColor;
```

where:

a. `R` = reflection direction vector of the light
b. `V` = view direction vector
c. `shininess` = material property controlling highlight sharpness
d. `LightColor` = light's intensity/color

4. **What are the components in a light reflection model?**
A typical light reflection model includes:

a. **Ambient reflection**: Constant base light in the scene.
b. **Diffuse reflection**: Light scattered in all directions.
c. **Specular reflection**: Shiny highlights based on viewer position.
d. **Emission (optional)**: Light emitted by the object itself. These components are often combined to produce the final color seen on the object.

5. **How to program the calculation of the cosine of the angle between two vectors?**
The cosine of the angle between two vectors is calculated using the **dot product**:

float cosTheta = dot(normalize(vecA), normalize(vecB));

- **The vectors are first normalized.**
- **The dot product then gives the cosine of the angle between them.**

6. **How to find the coordinates of the direction vector of a specular reflection ray?**
   The direction vector of a specular reflection ray R can be computed using the light direction vector L and the surface normal N:

   vec3 R = reflect(-L, N);

   The `reflect()` function in GLSL performs this operation using the formula:
   `R = 2 * dot(N, L) * N - L`

7. **How to calculate the coordinates of the normal vector?**
   The normal vector N is a unit vector perpendicular to the surface at a given point. It can be calculated:

- Manually from geometry (e.g., using cross product of two edges for triangles).
- Automatically generated when creating objects with normals.
- Normalized to ensure its magnitude is 1:

  N = normalize(N);

  Accurate normals are essential for correct lighting calculations in shaders.