**Project 7: Putting It All Together**

Atuhaire Ambala and Ricardo Escarcega

Prof. Ricardo Citro

CST 310

11/06/24

## 

#### **Project Overview**

This project is an exploration of 3D animation, lighting, and user interactivity using OpenGL. It consists of two parts, each emphasizing different aspects of OpenGL functionalities:

1. **CheckeredTriangles**: This program displays a rotating pattern of triangles with a checkered design, featuring user-controlled movement, zooming, and stopping of the rotation.
2. **ColorCubeFlyby**: This interactive 3D scene features multiple colored cubes, adjustable brightness, and illumination. Users can control rotation, zoom, and the positioning of cubes within the environment, including the addition of bounding planes.

The objective is to demonstrate OpenGL skills by integrating keyboard controls for interactivity, exploring lighting and brightness effects, and understanding the impact of transformations in a 3D space.

### **Part 1: CheckeredTriangles**

#### **Description**

The **CheckeredTriangles** program visualizes a pattern of spinning triangles in a checkered arrangement. The user can control the spin, movement, and zoom, making it an interactive experience.

#### **Features**

1. **360-Degree Rotation**: Triangles spin continuously in a 3D space, allowing a full view of the checkered pattern.
2. **User Interactivity**:
   * **Start/Stop Rotation**: Press c to start spinning and p to stop the motion.
   * **Vertical Movement**: Use u to move the pattern upward and d to move it downward.
   * **Horizontal Movement**: Use L to move left and R to move right.
   * **Zoom Control**: Press + to zoom in and - to zoom out.

#### **Technical Details**

* **Core Functions**:
  + renderCheckeredPattern(): Draws the checkered triangles.
  + processKeyboardInput(): Handles keyboard events to control the animation and position.
* **Dependencies**: This part uses OpenGL for rendering and GLUT for managing window and input events.

### **Part 2: ColorCubeFlyby**

#### **Description**

The **ColorCubeFlyby** program displays a scene containing multiple cubes with distinct colors and brightness. It also includes two planes to contain the bouncing motion of the cubes. Users can interact by rotating, zooming, and repositioning the cubes, and by modifying lighting to observe the effects on each cube's appearance.

#### **Features**

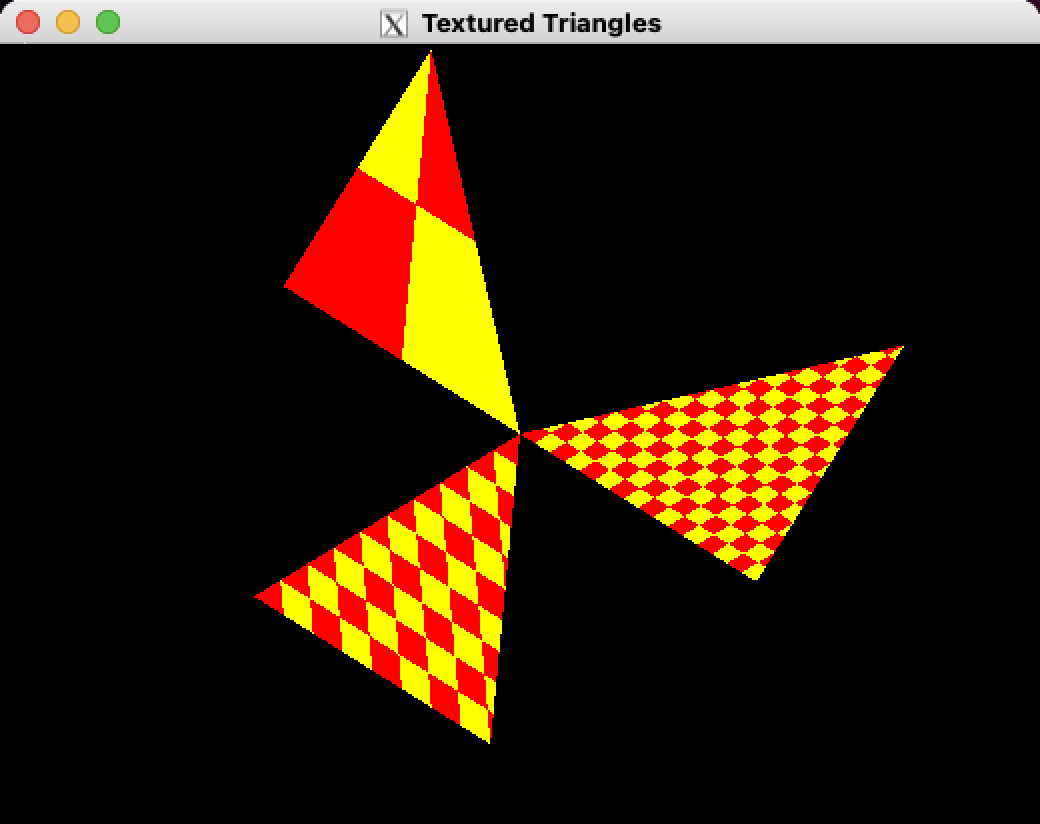
1. **Rotation Control**: Users can rotate the scene with the r key to view it from different angles.
2. **User Interactivity**:
   * **Start/Stop Cube Movement**: Press s to halt motion and c to resume.
   * **Vertical Movement**: Use u to move up and d to move down.
   * **Zoom Control**: + to zoom in and - to zoom out.
3. **Enhanced Visuals**:
   * Additional cubes with varied colors.
   * Two vertical planes on each side to create a boundary for the cube’s motion, creating a "bouncing" effect.

#### **Technical Details**

* **Core Functions**:
  + renderCubesWithPlanes(): Renders multiple cubes and vertical planes with specified lighting.
  + processCubeControls(): Manages input events to control cube behavior and positioning.
* **Lighting and Shading**: The program uses a lighting model to add realism, with adjustable brightness levels to highlight the effects of illumination on each cube.

### **Requirements and Setup**

1. **Dependencies**:
   * OpenGL for 3D rendering.
   * GLUT for handling user input and window creation.
2. **How to Run**:
   * Compile each .cpp file using an OpenGL-compatible compiler.
   * Ensure that any required assets, such as shaders or textures, are in the appropriate directory.
3. **Keyboard Controls**:
   * **CheckeredTriangles**:
     + Start/Stop Rotation: c to start, p to stop.
     + Move Up/Down: u for up, d for down.
     + Move Left/Right: L for left, R for right.
     + Zoom: + to zoom in, - to zoom out.
   * **ColorCubeFlyby**:
     + Rotate Scene: r key.
     + Start/Stop Cube Motion: s to stop, c to continue.
     + Move Up/Down: u to go up, d to go down.
     + Zoom: + to zoom in, - to zoom out.
4. **Screenshots**:



A screenshot of a computer

Description automatically generated

### **Mesh Creation Methodology**

### Creating a 3D mesh in OpenGL involves defining the vertices, setting up buffers, and configuring shaders to render the object on the screen. This project includes mesh creation for both triangles (Part 1) and cubes (Part 2), each designed to exhibit different lighting and animation effects.

#### Steps to Create a Mesh

### Define Vertex Data:

### The vertex data includes coordinates for each point in the mesh, along with any additional attributes, such as normals for lighting calculations.

### For example, in the CheckeredTriangles program, vertices are defined to form a triangular pattern. In ColorCubeFlyby, vertices are defined for each face of the cube, ensuring each face can interact independently with the lighting.

### Set Up Vertex Buffer Object (VBO) and Vertex Array Object (VAO):

### A Vertex Buffer Object (VBO) is created to store vertex data in GPU memory, which allows efficient access during rendering.

### A Vertex Array Object (VAO) is used to manage multiple VBOs, enabling the GPU to process the mesh data in one structure.

### In this project, each VBO stores the positions and normals of the vertices, while the VAO links these buffers for use during rendering.

### Normal Calculation:

### Normals are essential for lighting calculations, as they determine how light interacts with the mesh’s surface. Normals are typically defined per vertex, specifying the direction perpendicular to the surface.

### For the ColorCubeFlyby program, each cube face has its normal vectors, allowing accurate specular and diffuse lighting effects based on the cube’s orientation.

### Configure Shaders:

### Shaders are programs that run on the GPU, controlling how each vertex and fragment (pixel) is processed. The vertex shader applies transformations to position the mesh in 3D space, while the fragment shader calculates lighting effects.

### In this project, custom shaders are used to implement the Phong lighting model, which includes ambient, diffuse, and specular lighting. These shaders make the mesh appear realistic under dynamic lighting conditions.

### Apply Transformations:

### Transformations, including translation, rotation, and scaling, are applied to position the mesh within the scene. These transformations are managed in the vertex shader, where matrices are used to control the object’s movement and orientation.

### For the CheckeredTriangles program, rotation transformations allow the triangles to spin, while translation matrices control the cubes’ movement in the ColorCubeFlyby program.

### Rendering Loop:

### In each frame, the mesh is drawn by binding the VAO, activating the shaders, and calling the glDrawArrays or glDrawElements function.

### Lighting calculations are applied in real-time within the fragment shader, providing dynamic reflections and shading based on the mesh’s position relative to the light source and camera.

### 

### **Mathematical Concepts**

This project applies several mathematical concepts to achieve realistic 3D animations, lighting, and interactions in an OpenGL environment. These concepts are foundational to understanding how 3D objects are rendered and illuminated in real-time graphics. Below is an explanation of these concepts, including definitions, formulas, relationships between variables, and their significance in the project:

1. **Rotation and Transformation Matrices**:
   * To animate and rotate objects, transformation matrices are used. A rotation matrix defines how an object rotates around the x, y, or z-axis. For instance, a rotation matrix for a z-axis rotation is given by:
   * Here, θ\thetaθ represents the rotation angle. By applying this matrix to object coordinates, we achieve rotation around the specified axis.

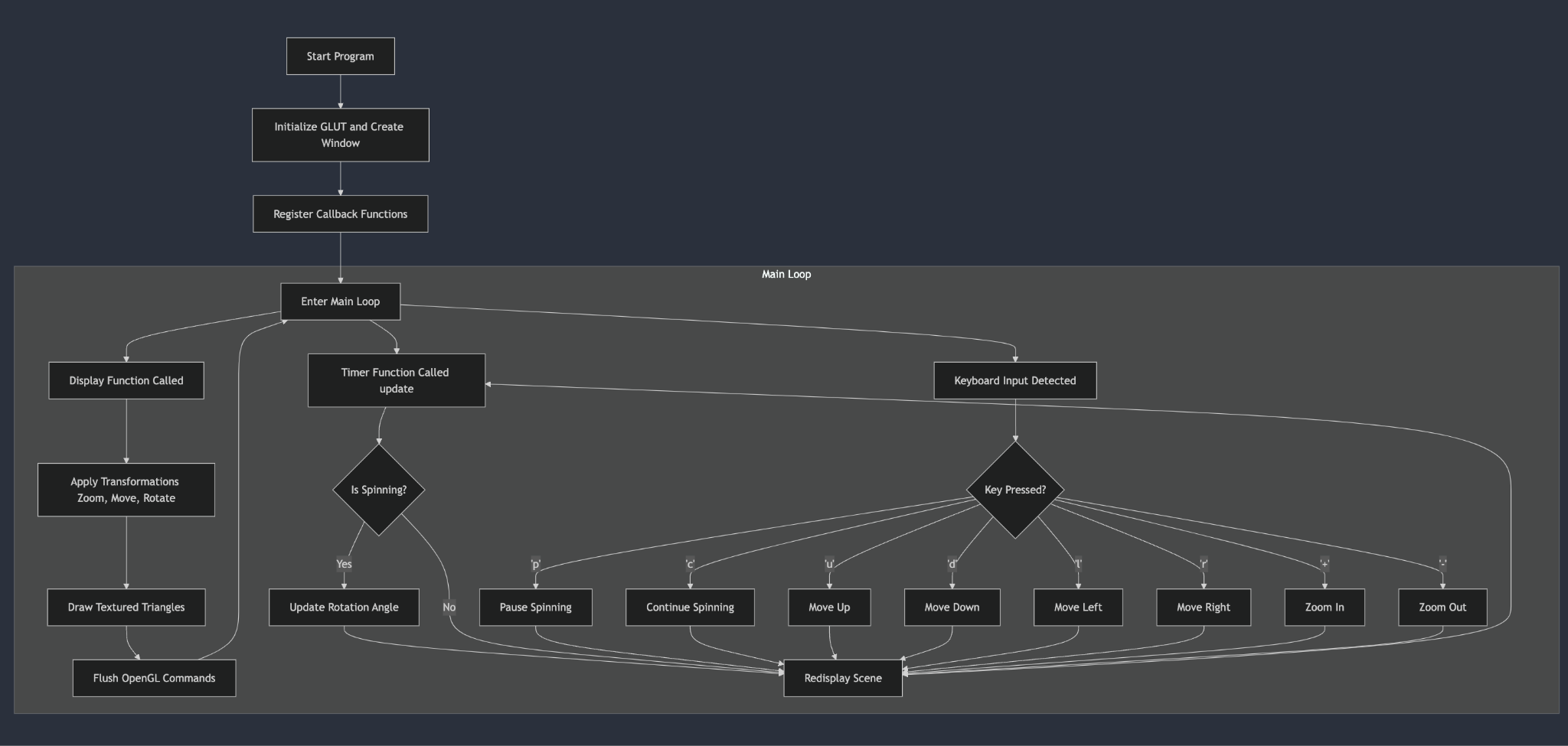
### **Method for Reading and Processing the Codes**

To understand and process **CheckeredTriangles.cpp** and **ColorCubeFlyby.cpp**, a structured approach was used:

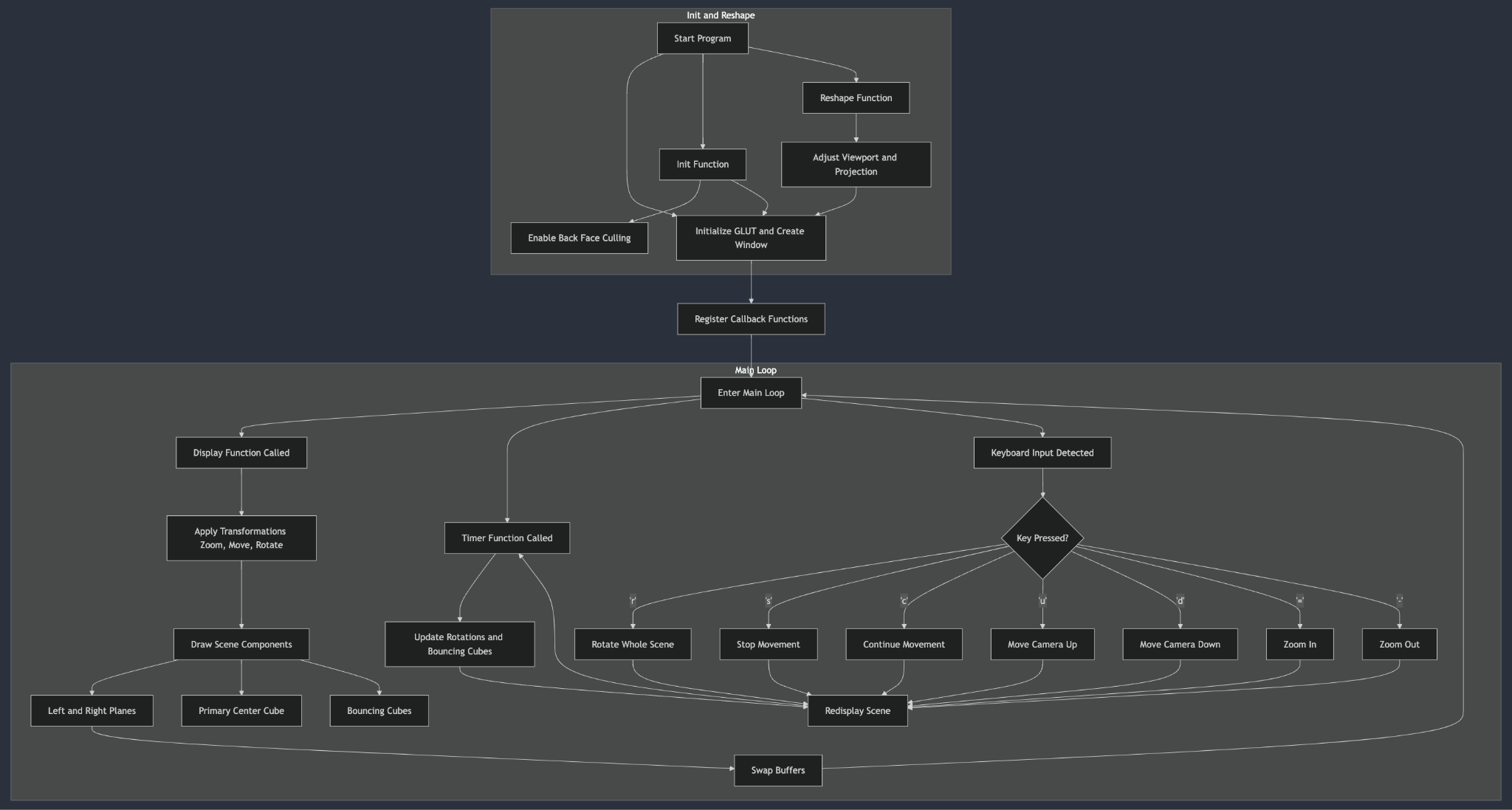
1. **Code Structure Overview**:
   * Each file’s overall structure was reviewed to understand the main functionalities. **CheckeredTriangles.cpp** focuses on rotating and transforming triangles, while **ColorCubeFlyby.cpp** handles multiple cubes with different colors and lighting effects.
2. **Key Functions and Controls**:
   * Essential functions for rendering, input handling, and transformations were identified. This included rotation and zoom controls in **CheckeredTriangles** and interactive lighting and movement in **ColorCubeFlyby**.
3. **Shader and Lighting Setup**:
   * The shader configurations and lighting calculations were analyzed, focusing on how the Phong lighting model was implemented to create realistic lighting effects for both triangles and cubes.
4. **User Interaction**:
   * Keyboard controls were examined to understand how user inputs affect animation and movement. Each control was tested to verify its functionality and impact on the visual output.
5. **Testing and Documentation**:
   * Each feature was tested for accuracy, with notes added to clarify code sections and document behavior. This approach ensured a clear understanding of each program’s components and interactions.

**Flowcharts**

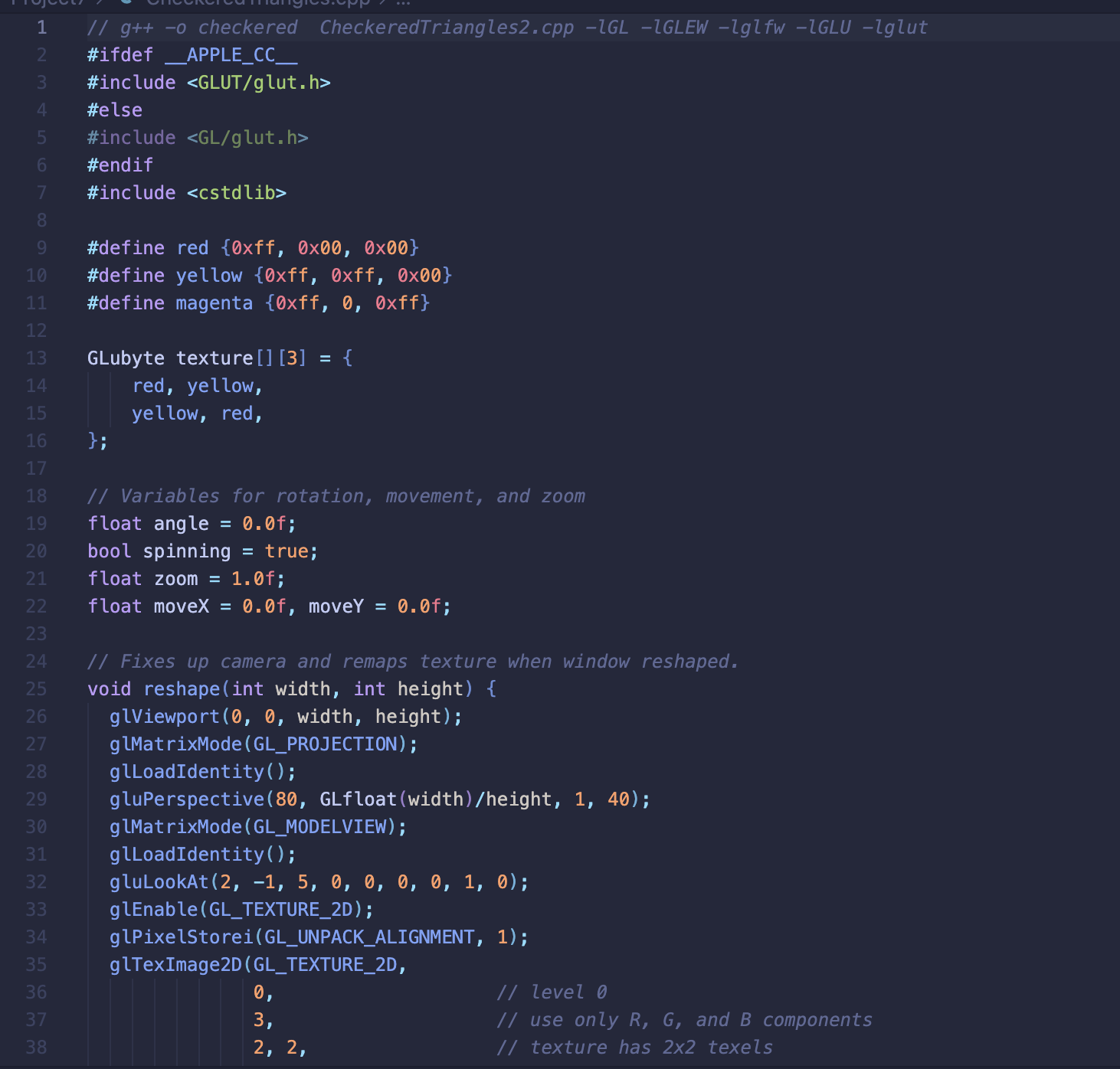
**Checkered Triangles**

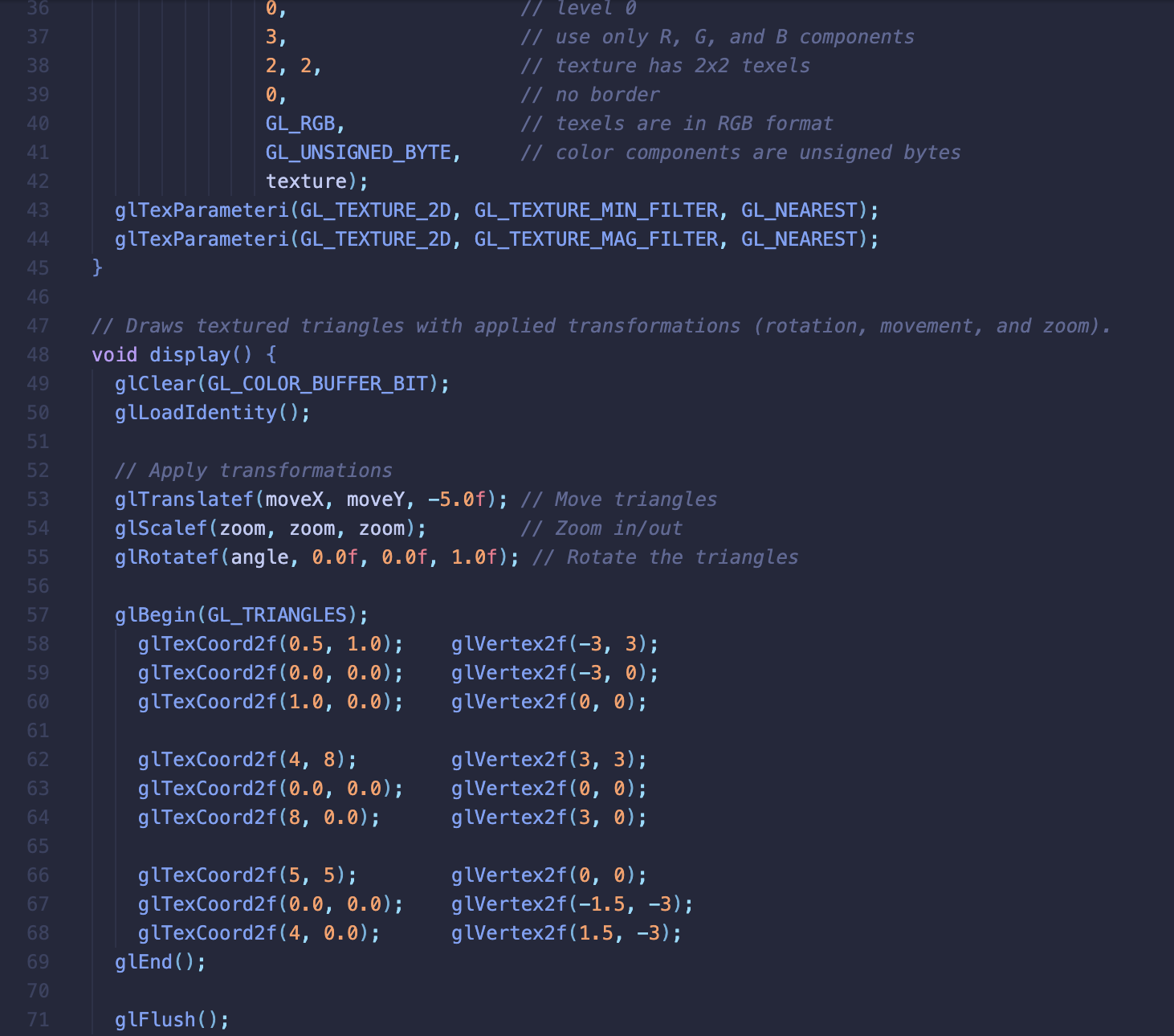
****

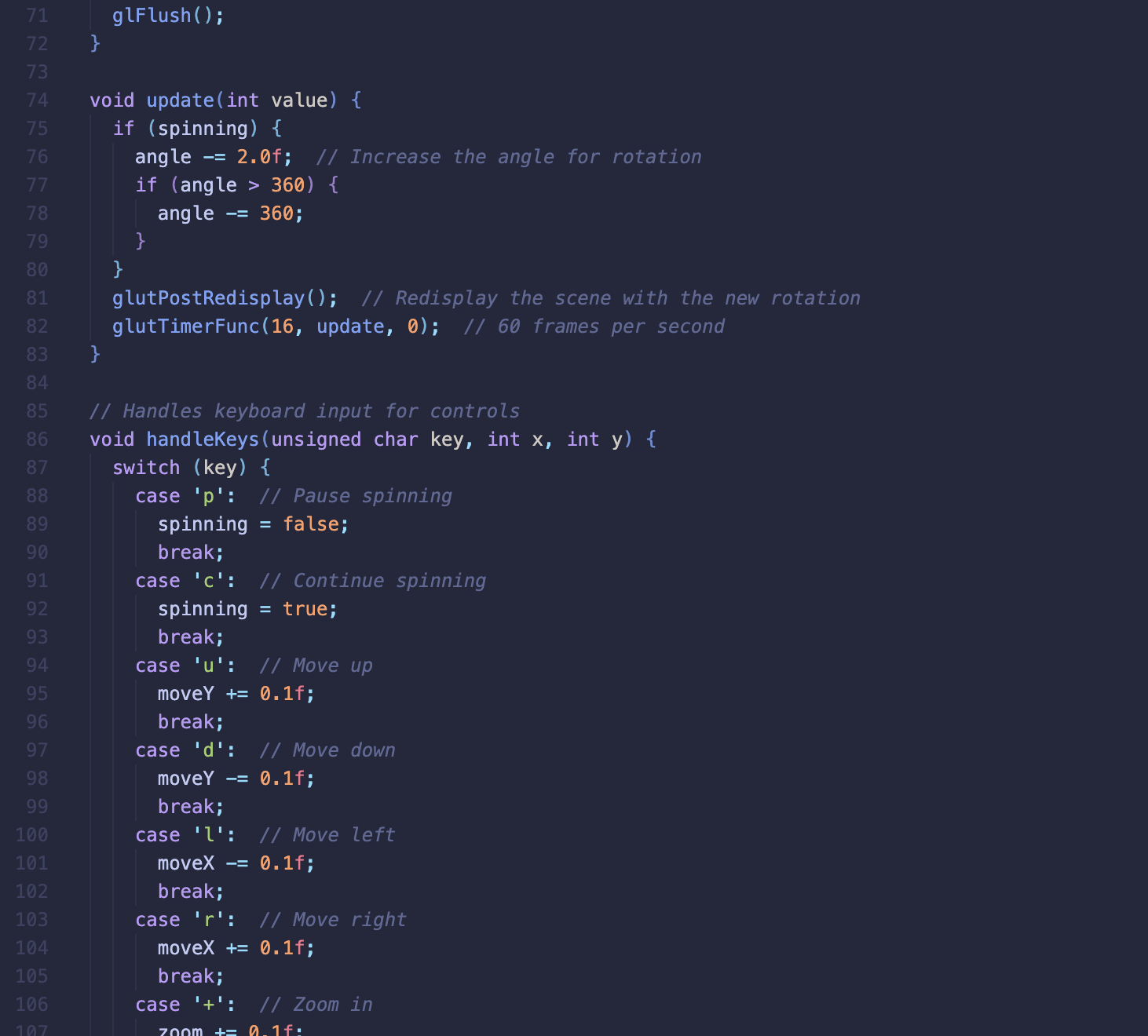
**Color Cubes Fly By**



**Checkered Triangles Code:**









**Color Cube Fly By Code**



