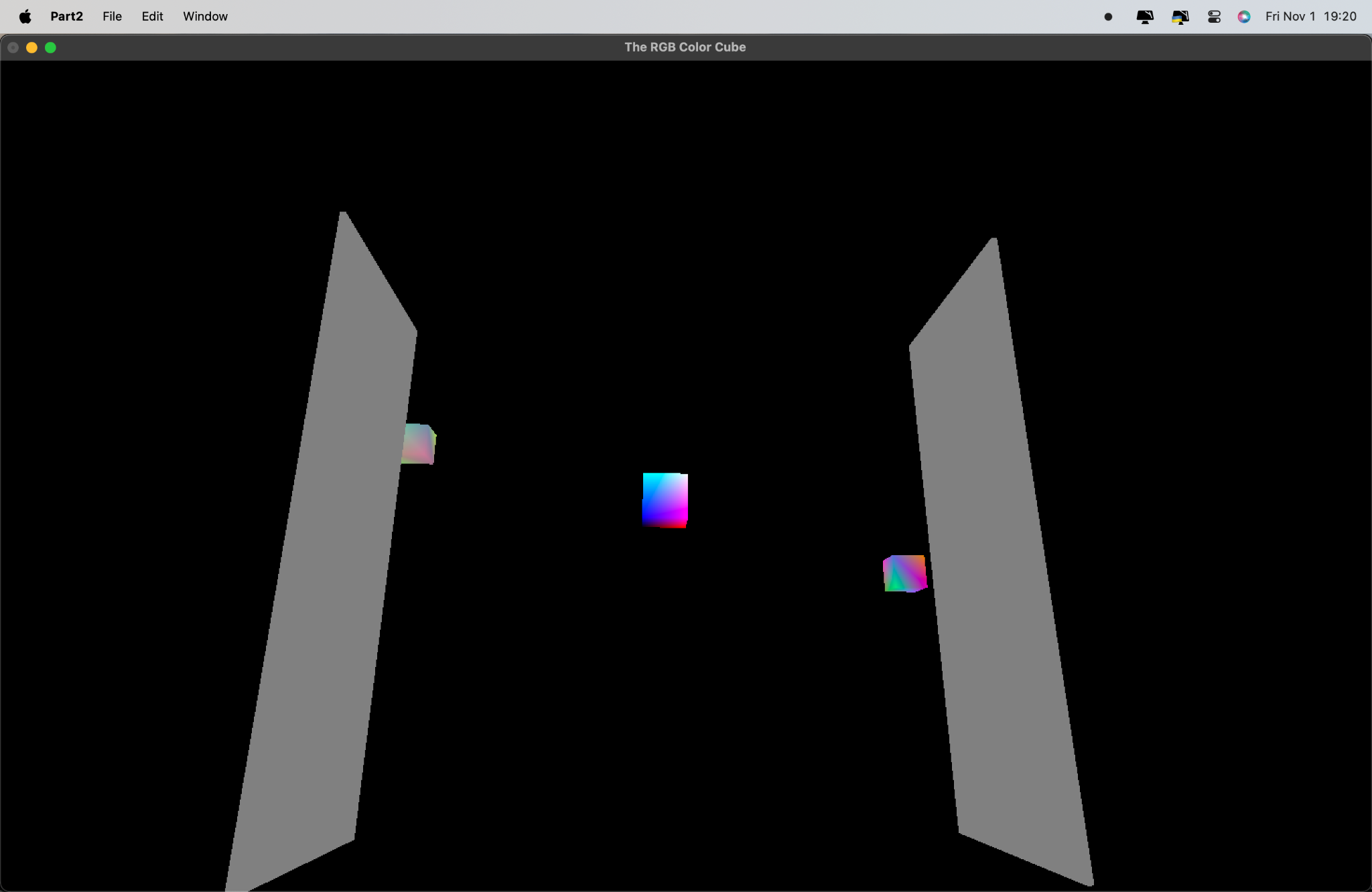
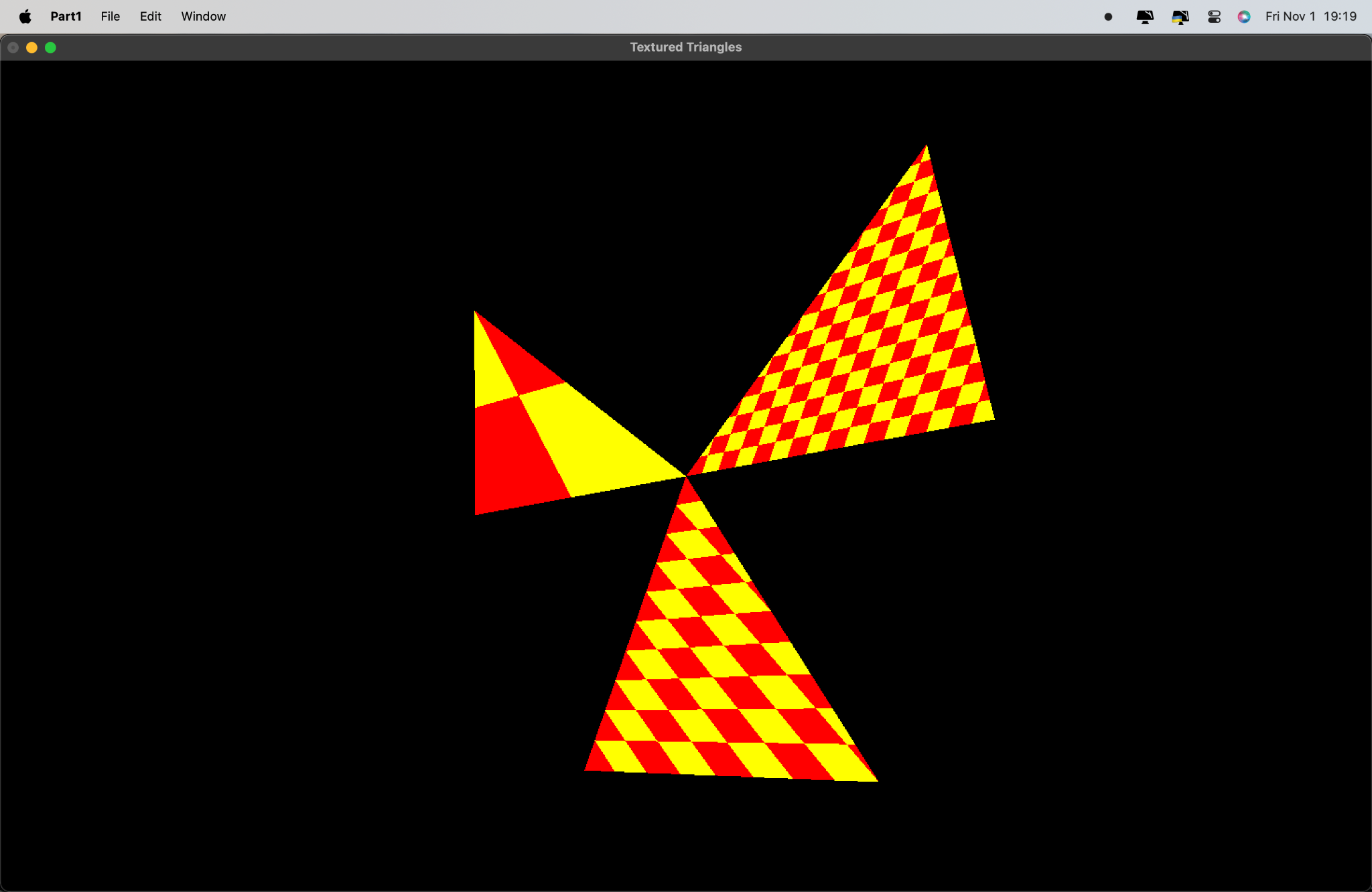
Project 7: Putting It All Together

CST-310 Computer Graphics

Angel Velazquez, Kyler Harden

November 6, 2024

Documentation

1. Overview

This document provides an in-depth analysis and explanation of two OpenGL programs:

• Textured Triangles Program: Illustrates texture mapping on multiple triangles using a simple red and yellow checkerboard pattern.

• RGB Color Cube Program: Displays a rotating RGB color cube with additional cubes floating around the main object.

We explore the mathematical concepts, implementation details, and the logical flow for each program.

2. Mathematical Concepts

2.1 Coordinate Systems and Transformations

Both programs rely heavily on 3D transformations, including:

• Translation: Moving objects within the 3D coordinate system.

• Formula: T(x, y, z) = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ dz \\ 0 \end{bmatrix}

• Here, dx, dy, dz represent the distances to move along each axis.

• Scaling: Changing the size of objects uniformly or along specific axes.

• Formula: S(x, y, z) = \begin{bmatrix} x \cdot s\_x \\ y \cdot s\_y \\ z \cdot s\_z \\ 1 \end{bmatrix}

• s\_x, s\_y, s\_z are scale factors for each axis.

• Rotation: Rotating objects around an axis. The angle of rotation, \theta , defines the amount of rotation.

• Around the Z-axis (used in the programs):

R\_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}

2.2 Texture Mapping (Textured Triangles Program)

Texture mapping involves applying a 2D image (texture) to a 3D surface.

• Texture Coordinates: Defined in the range [0, 1], where each vertex of a triangle corresponds to a point on the texture.

• Texture Filtering: The program uses GL\_NEAREST filtering to ensure that each texel (texture pixel) is directly mapped to the closest pixel on the object.

2.3 Camera and Perspective (RGB Color Cube Program)

A perspective projection is used to simulate a realistic 3D view. The projection matrix is created using:

• Field of View (FOV): The angle of the viewing frustum in the vertical direction.

• Aspect Ratio: Ratio of the window’s width to height.

• Near and Far Clipping Planes: Limits on the depth of the scene.

The gluLookAt function is employed to simulate a “flyby” around the RGB cube, following a sinusoidal path:

x = 8 \cos(u), \quad y = 7 \cos(u) - 1, \quad z = 4 \cos\left(\frac{u}{3}\right) + 2

2.4 Color Gradients in the RGB Color Cube

Each axis of the color cube represents one of the RGB color channels:

• X-Axis: Red gradient

• Y-Axis: Green gradient

• Z-Axis: Blue gradient

3. Programming Implementation

3.1 Data Structures

• Arrays: Used to store vertices, colors, and faces of cubes.

• Example: GLint vertices[8][3] for cube vertices, GLfloat vertexColors[8][3] for color values.

• Namespaces: Each cube has its own namespace (e.g., Cube, Cube2, Cube3) to encapsulate its properties and avoid name conflicts.

3.2 Code Organization and Style

Each program follows a modular approach:

• Main Functions: main, display, reshape, handleKeys, and animate.

• Custom Functions: Each program encapsulates object-specific details in distinct namespaces (e.g., Cube::draw).

• Animation Functions: animate and timer are used to handle animation, updating rotation and position values periodically.

4. Mesh Creation

The programs create a mesh for each object using vertices and faces:

• Cube Mesh: Defined by 8 vertices and 6 faces (for each side of the cube).

• Triangle Mesh (Textured Program): Each triangle is defined with 3 vertices and associated texture coordinates.

To construct the mesh, the glBegin(GL\_QUADS) and glBegin(GL\_TRIANGLES) functions define each face, with the coordinates mapped using glVertex3iv and glVertex2f.

5. Methodology for Reading and Processing the Code

1. Initialize GLUT: Initializes the window and sets display parameters.

2. Setup Reshape Function: Adjusts the viewport and camera settings based on window dimensions.

3. Define Geometry and Colors: Vertices, colors, and textures are defined in arrays.

4. Handle Animation and Input:

• animate or timer functions control spinning and movement.

• Keyboard functions (handleKeys) allow user interaction for translating and rotating objects.

5. Rendering Loop:

• Each frame, the display function clears the screen and applies transformations.

• Texture is applied to each triangle/cube face before rendering.

Execution Flowchart

Here’s a flowchart depicting the logic of the approach for generating all features:

+---------------------+

| Program Start |

+----------+----------+

|

v

+--------------+--------------+

| Initialize GLUT and Set Up |

| Display Mode, Window, etc. |

+--------------+--------------+

|

v

+----------+----------+

| Set up Reshape |

| and Projection |

+----------+----------+

|

v

+----------+----------+

| Define Objects (Cubes|

| and Triangles) |

+----------+----------+

|

v

+----------+----------+

| Handle Keyboard |

| and Animation Input|

+----------+----------+

|

v

+----------+----------+

| Render Each Frame |

+----------+----------+

|

v

+----------+----------+

| Animate and Spin |

+----------+----------+

|

v

+------+------+

| Program |

| End |

+-------------+

6. Spinning Technique for 360-Degree Rotation

The animate function achieves 360-degree spinning by incrementing the angle variable in each frame update. This value is used in glRotated to rotate the entire object, creating a smooth spinning effect.

Explanation

• The rotation axis is defined around the Z-axis: glRotated(angle, 0, 0, 1.0f);.

• The angle variable increments every 10 milliseconds, creating a smooth rotational movement.

• The rotation wraps around after 360 degrees, continuing indefinitely.

7. Summary of Key Concepts

• 3D Transformations: Essential for positioning, rotating, and scaling objects in OpenGL.

• Texture Mapping: Demonstrates basic texture application techniques, including texture coordinates and filtering.

• Animation and Input Handling: Uses GLUT’s timer and keyboard functions for dynamic interaction.

• GLUT Reshape and Display Management: Allows objects to remain visible and centered even when the window is resized.

8. Visuals and Presentation

The program is structured to ensure clarity and modularity, allowing for easy modification and experimentation. Each function and variable serves a distinct role, contributing to the interactive 3D scene. With a simple cube and checkerboard texture, the code demonstrates fundamental OpenGL concepts effectively.

This documentation is designed to offer a complete understanding of both the code and the mathematical and programming principles underlying the programs, making it a valuable resource for further study or adaptation in related projects.