**Colored Cube**

Camden Birkeland

Colorado State University Global

CSC405: Graphics & Visualization

Muhammad Khan

5/12/2024

**Colored Cube**

To create the colored cube, we first initialized OpenGL and GLFW to set up a rendering context. We created a GLFW window with a specified OpenGL version and core profile. Following this, we initialized GLEW to handle the OpenGL extensions. This setup ensures that the necessary tools are available for rendering. We then wrote vertex and fragment shaders in GLSL (OpenGL Shading Language). The vertex shader processes each vertex, handling transformations to position vertices correctly in the 3D space. The fragment shader determines the color of each fragment (pixel) based on the interpolated vertex colors. Compiling and linking these shaders into a shader program enabled their use in rendering operations.

Next, we defined the vertex data for a cube, specifying positions and colors for each vertex. We created Vertex Buffer Objects (VBOs) and Vertex Array Objects (VAOs) to manage this data efficiently. The VBO stores vertex attributes such as positions and colors in the GPU memory, while the VAO keeps track of the vertex attribute configurations. Additionally, we used an Element Buffer Object (EBO) to store the indices that define the cube’s faces. This method allows for efficient rendering by reusing vertices and minimizing the amount of data sent to the GPU. Binding these buffers and configuring vertex attribute pointers enabled the GPU to interpret the data correctly.

Finally, we set up transformations to animate the cube. The model matrix handles the cube’s rotation, the view matrix simulates a camera by moving the scene, and the projection matrix applies perspective to give a 3D effect. In the render loop, these matrices are updated continuously, creating the animation of a rotating cube. The updated matrices are sent to the shaders as uniform variables. Within the render loop, we clear the screen, apply the shader program, and draw the cube using glDrawElements. Enabling depth testing ensures that the cube's faces are rendered correctly concerning their spatial depth. This combination of shaders, buffers, and transformations results in a smoothly rotating, colored 3D cube rendered on the screen.

**Output**

