**Colored Cube Reflection**

*Andrew Barnes*

**Steps to Create Colored Cube**

Using WebGL to create a colored cube involved defining its geometry and color data by understanding vertices and faces and how they can make 3D shapes. I acquired a WebGL2 context from the canvas element to utilize the API functions used for 3D rendering. After initializing WebGL, I defined two shader programs, a vertex shader, and a fragment shader. The vertex shader obtains the locations for each vertex position and their applied transformations, and the fragment shader gives color to each pixel. Both vertex and fragment shaders are compiled and linked into a shader program to execute it on the GPU for graphics processing. Next, buffers hold data, such as positions and color, for the cube’s vertices, and indices define the vertices occupied within each face. Loading buffer data is critical because it holds the data the GPU will process. A loop is then used to draw the scene repeatedly in a function where rotations are applied. At last, the scene is drawn by taking indices from the buffer to draw each face of the cube.

**Explaining Shaders, Buffers, and Transformations**

Shaders, written in the OpenGL Shading Language (GLSL), are programs that run on the GPU for rapid parallel processing of millions of vertices and pixels. The vertex shader manipulates vertex data to position vertices in their correct positions by applying transformations, such as rotation, scaling, and translation, to transform geometry into a 3D perspective. Meanwhile, the fragment shader decides the pixel’s color and brightness based on factors like vertex colors, lighting models, and textures. Fragment shaders take geometric shapes formed from vertices and fill them with color to render the final image.

Buffers are the containerized data that the GPU must access. The vast amount of data involves vertex coordinates, color information, and indices defining faces formed from vertices. When data is organized into buffers, WebGL has a structured way to deliver it to the GPU for fast and direct access while rendering. Breaking down the data into different buffers like position buffer, color buffer, and index buffer provides WebGL a way to handle each rendering aspect more intimately and effectively. For example, the index buffer is used to recycle vertices for multiple faces, reducing the amount of data needed to be processed.

Transformations are mathematical operations applied to vertices that alter their positions. The primary operations are translation, rotation, and scaling. Translations move objects from one place to another, rotations spin objects around an axis, and scaling changes the size of an object. In WebGL, matrices modify vertex positions in the vertex shader to create transformations.

**Reflection**

From this colored cube rotation project, I’ve immersed myself in the nuances of 3D graphics to create a rendered demonstration of object transformation. Overcoming shader creating and transformation techniques provided me insight into WebGL's robustness. For each development step of geometries, shaders, buffers, and transformations, I connected them to the rendering pipeline in WebGL. The hands-on experience of the project made the complexities of 3D transformations attainable, and I could see it through to a rendering loop creating an animation. Above all, I enjoy the low-level intimacy involved throughout development because I can control every tidbit of data implemented into bringing an object to view.