# A Colored Cube with WebGL

Stephan Peters

Colorado State University Global

24FB-CSC405-1: Graphics and Visualization

Jennifer Marquez

8 September 2024

## **A Colored Cube in WebGL**

In this project, I use WebGL to create a colored cube that appears to rotate. I’ve also implemented some mouse-drag and arrow key events. When the mouse is dragged, the cube’s rotation will follow the movement of the mouse. The arrow keys resize the cube and make it appear to come closer or move farther away.

## Vertices, Shapes, Shaders, Buffers and Transformation

Vertices define object shapes. In this application, the DEFAULT\_VERT array contains the vertex positions and colors of the cube. Each vertex is represented by a position (x, y, z) and a color, red, green and blue (r, g, b). The vertices are organized in a specific order to form triangles that define the faces of the cube. The shaders control appearance. This application includes both a vertex shader and a fragment shader. The vertex shader (vertex.glsl) processes each vertex and calculates the position of the vertex in normalized device coordinates. It also passes the color of the vertex to the fragment shader. The fragment shader (fragment.glsl) receives the color of each fragment (a small part of a triangle) and determines the final color of the fragment. Buffers hold data: The initVertexBuffers() function creates a buffer object, binds it, and copies the vertex data (DEFAULT\_VERT) into the buffer. The buffer is then associated with the vertex attributes (aPosition and aColor) in the vertex shader. Transformations position and orient objects: The draw function calculates the Model-View-Projection (MVP) matrix, which transforms the vertex positions from model space to normalized device coordinates. The MVP matrix is then passed to the vertex shader as a uniform variable (uMVPMatrix). The vertex shader applies the MVP matrix to each vertex position, resulting in the final position of the vertex in the 3D scene. The updateState() function continuously updates the rotation angles (state.app.angle.x and state.app.angle.y) to create the rotating effect.

## The Vertex Shader, Fragment Shader, Buffers, and Building the Cube

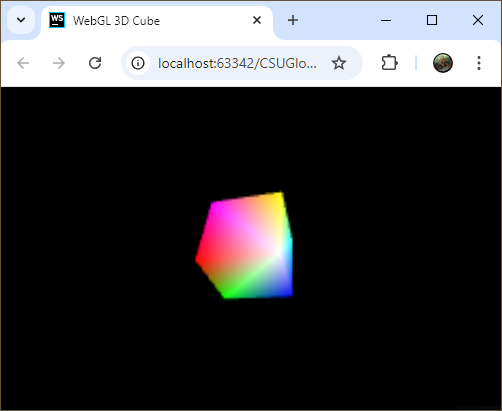
The Vertex and Fragment Shaders in this application are very simple. The Vertex Shader does not perform any additional calculations or transformations on the vertex positions or colors. It simply passes the transformed vertex positions and original colors to the Fragment Shader as varying variables (aPosition and aColor). The calculations are performed within the JavaScript. The Fragment Shader in this application receives the interpolated vertex colors from the Vertex Shader and sets the output color of the fragment to the interpolated color. In this case, the Fragment Shader also does not perform any additional calculations or transformations on the colors, the colors are also manipulated in the JavaScript. The varying vec4 vColor; declaration declares a varying variable (vColor) to hold the color passed from the Vertex Shader to the Fragment Shader. This variable will be interpolated across the fragments produced by rasterization. The void main() function is the main entry point of the Fragment Shader. Inside this function, the output color of the fragment (gl\_FragColor) is set to the interpolated color passed from the Vertex Shader (vColor). This color will be used for rendering the fragments on the screen. The initVertexBuffers() function initializes the vertex buffers and attributes and sends them to the shaders, which are then used in the draw function to render the 3D objects. The buffers that hold and communicate data between the CPU and GPU are managed in the ./common/glUtils.program.renderBuffers() function. The actual cube is created using triangles in the DEFAULT\_VERT and DEFAULT\_INDICES arrays, DEFAULT\_VERT gives the initial xyz coordinates and rgb color information, and DEFAULT\_VERT creates the cube faces using triangles.

## Implementing a loop to Rotate the Cube

The updateState() function acts as a render loop that continuously updates and redraws the scene in the WebGL application. It is called repeatedly within the animate() function, which creates an animation loop. updateState() is called at the beginning of each frame in the animate() function. It calculates the time difference (timeDifference) between the current frame and the last frame, then it updates the angles (state.app.angle.x and state.app.angle.y) based on the time difference and rotation speeds. Finally, it updates the last update time (state.animation.lastUpdateTime) to the current time. updateState() is also responsible for handling arrow key-press events.

### Figure 1

*Example of the rotating cube.*

*.*

## Conclusion

This was an interesting experiment in creating animation of a shape using WebGL. The ability to control the movement of the cube using event handlers (built into signals.js) was also rewarding. WebGL is very powerful.

## References

Angel, E., & Shreiner, D. (2020). *Interactive computer graphics* (8th ed.). Pearson.

Parker, D. W. (2016). *ProgrammingTIL-WebGL*. GitHub. <https://github.com/davidwparker/programmingtil-webgl>

Parker, D. W. (2017, January 24). *How to draw a 3D cube in WebGL - ProgrammingTIL #118 WebGL tutorial video screencast 0061*. YouTube <https://www.youtube.com/watch?v=qmUXezq_tE8>