# An Interactive Viewer with WebGL

Stephan Peters

Colorado State University Global

24FB-CSC405-1: Graphics and Visualization

Jennifer Marquez

15 September 2024

## **An Interactive Viewer with WebGL**

In this project, I use WebGL with specific code provided in section 5.4.6 of Angel and Shreiner’s 8th edition of *Interactive Computer Graphics*. I couldn’t use their render() function provided in the textbook with the interactive cube application I presented for Critical Thinking 4, as the WebGL utilities (in the common directory) are incompatible without major modification to the source files.

## User Controls

The sliders enable users to interactively explore different viewing angles and distances for the 3D cube. The corresponding JavaScript code in the slider.js file handles the event listeners by creating an array for each slider to be handled, then running the elements of the array through a for each loop, then it updates the WebGL rendering accordingly.

function *getSlider*(sliderId) {

return document.getElementById(sliderId);

}

*// Create an array of slider IDs*

const sliderIds = ['distanceSlider', 'thetaSlider', 'phiSlider',

'aspectSlider', 'fovSlider'];

*// Create an array of slider elements*

const sliders = sliderIds.map(*getSlider*);

*// Add a universal event listener to each slider element*

sliders.forEach((slider) => {

const sliderCircle = document.getElementById('sliderCircle');

const sliderPath = document.getElementById('sliderPath');

slider.addEventListener('input', (event) => {

const angle = (event.target.value - slider.min) /

(slider.max - slider.min) \* 360;

const radius = 40;

const x = 50 + radius \* Math.cos(angle \* Math.PI / 180);

const y = 50 + radius \* Math.sin(angle \* Math.PI / 180);

sliderCircle.setAttribute('cx', x);

sliderCircle.setAttribute('cy', y);

});

});

The specific sliders and their corresponding viewing functions are as follows:

* Distance Slider: This slider controls the distance of the cube from the origin in the 3D space. The radius value is used to calculate the position of the cube in the scene.
* Theta Slider: This slider controls the rotation angle around the y-axis. The theta value is used to update the model-view matrix to rotate the cube.
* Phi Slider: This slider controls the rotation angle around the x-axis. The phi value is used to update the model-view matrix to rotate the cube.
* Field of View (FOV) Slider: This slider controls the field of view of the perspective projection. The FOV value is used to calculate the projection matrix.
* Aspect Slider: This slider controls the aspect ratio of the canvas. The aspect ratio is used to calculate the projection matrix

## The Vertex Shader and Fragment Shaders

The vertex shader provided in the textbook were very simple and created a very “Windows 3.11” look in the browser, as they did no shading of the elements.

attribute vec4 vPosition;

attribute vec4 vColor;

varying vec4 fColor;

uniform mat4 modelViewMatrix;

uniform mat4 projectionMatrix;

void main()

{

gl\_Position = projectionMatrix \* modelViewMatrix \*

vPosition;

fColor = vColor;

}

So, I modified it to include a light source and some shading.

...

uniform vec3 lightPosition;

...

void main()

{

vec4 transformedPosition = modelViewMatrix \* vPosition;

vec3 lightDirection =

normalize(lightPosition – transformedPosition.xyz);

float diffuse = max(dot(lightDirection,

vec3(0.0, 0.0, 1.0)), 0.0);

vec4 diffuseColor = vec4(vColor.rgb \*

diffuse, vColor.a);

gl\_Position = projectionMatrix \* transformedPosition;

fColor = diffuseColor;

}

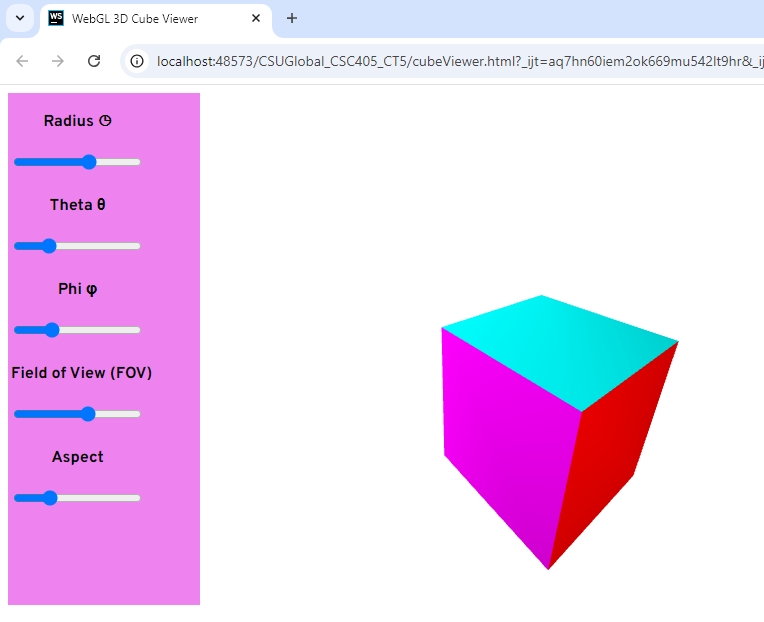
This made the cube look a lot nicer as it added subtle shading to it.

## Running the Program

Running the program produces some interesting effects. It is interesting to modify the cube using the sliders and watching the perspective change. It does not change as a viewer would expect if their eyes were at the camera point, the viewer can see what the object would look like from a different perspective with vanishing points, but the viewer is not always looking at the “near” face of the cube. It gives me a lot of respect for the designers who design for movie animation or 3D games. Examples of the program running can be seen in the figures on the following page:

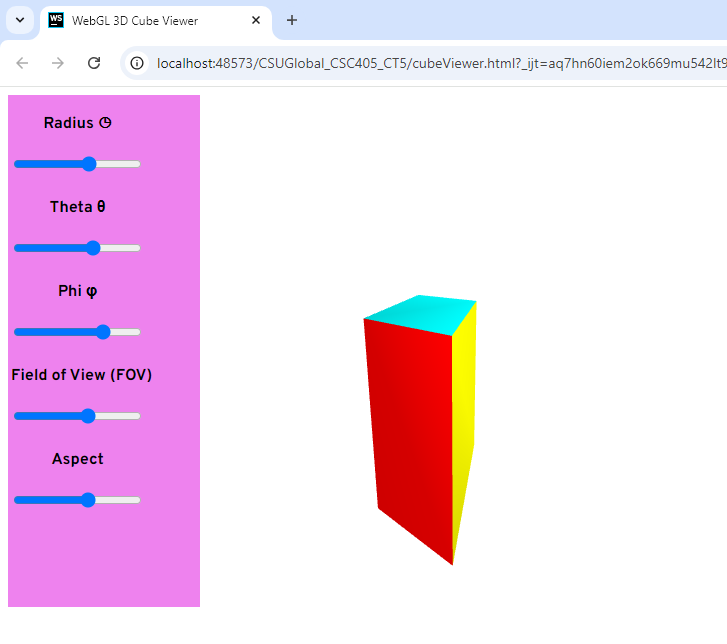
### Figure 1

*Example of the program running.*

*.*

### Figure 2

*Another Example of the program running.*

*.*

## Conclusion

While I would have much rather used a different WebGL interface than the one provided by Angel and Shreiner, it was a good exercise in making a project work with a provided codebase that may not be optimal. This is a true real-world situation.

## References

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