# Approximation of a Sphere with WebGL

Stephan Peters

Colorado State University Global

24FB-CSC405-1: Graphics and Visualization

Jennifer Marquez

22 September 2024

## **Approximation of a Sphere with WebGL**

In this project, I use WebGL with specific code provided in section 6.6 of Angel and Shreiner’s 8th edition of *Interactive Computer Graphics*. A curved surface in a WebGL solid model is a mesh of polygons, in the case of this recursive sphere, of 3d tetrahedrons. This process is called tessellation. This is true of almost all solid modeling applications. Sophisticated models, such as in VR, motion pictures, or high-resolution 3D games can have millions of triangles to create one still image (Nießner et al., 2015).

## Discussion of the Sphere Object

In the context of shading calculations, a sphere serves as an ideal curved surface to demonstrate the application of various shading techniques. Shading algorithms, such as flat shading, Gouraud shading, and Phong shading, calculate the color of each pixel on the sphere's surface based on its position, orientation, and lighting conditions. These techniques simulate realistic visual effects by varying the intensity and direction of reflected light, resulting in smooth transitions between colors and highlights. The visual appearance of a sphere on a 2D surface (monitor or screen) using this WebGL program is accomplished by creating a tetrahedron, then dividing that tetrahedron recursively until the appearance of a sphere is shown on the screen. The surfaces of the sphere are not truly curved, but the final product makes the 2D image appear curved. Figure 1 shows the creation of a sphere using a Discrete Global Grid System (Lei et al., 2020). While this is not what I used to create the sphere in this example, it is a good visual representation of how the sphere is created recursively in my example:

### Figure 1

*The 4-fold tetrahedron version Discrete Global Grid Systems (DGGSs)*

A blue sphere with green lines

Description automatically generated

## Implementing the Sphere in WebGL

In this program, the sphere is approximated using a recursive subdivision algorithm, which starts with a tetrahedron and divides each face into smaller triangles. The code generates the sphere by calling the tetrahedron function, which in turn calls the divideTriangle() function to recursively divide each face of the tetrahedron. The divideTriangle() function recursively divides a given triangle into smaller triangles until the specified subdivision level is reached. I also included event listeners for user interactions, such as adjusting material properties (color), phi and theta (which make the sphere appear ovaloid if they are not uniformly moved) and number of subdivisions (recursions). The render() function is responsible for rendering the shaded sphere using WebGL, updating the model-view and projection matrices, and drawing the sphere triangles. The shading calculations in the provided WebGL code make the sphere appear more realistic by applying lighting effects to the vertices of the triangles that form the sphere. The calculations are based on the Phong reflection model, which combines ambient, diffuse, and specular lighting components. The user can see the effects of some of the matrices and variables using sliders in the included HTML.

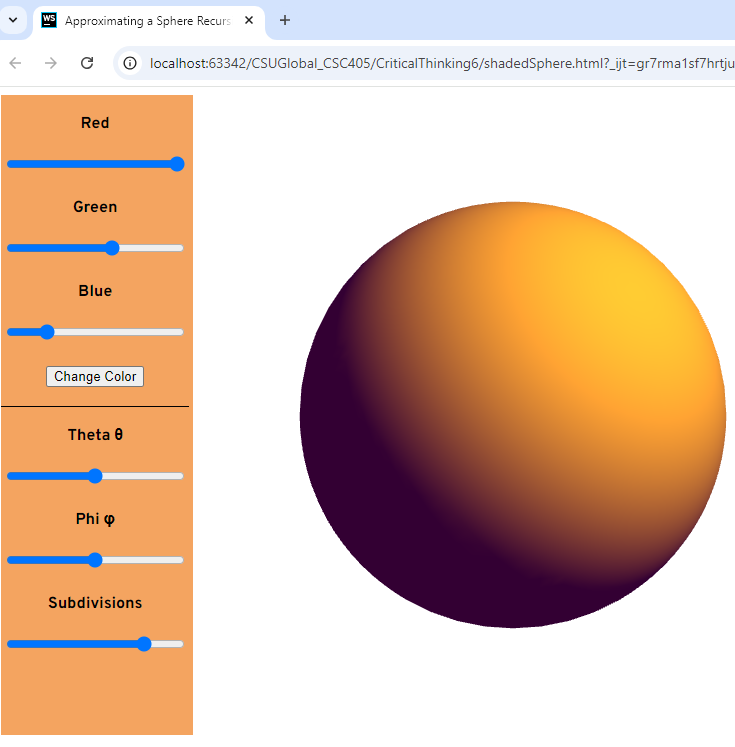
## Challenges Faced

While creating the program, there were several challenges. One challenge was initially for changing the colors I called init() as part of the slider listeners. This resulted in an unresponsive web page as the color sliders were manipulated. So, instead of refreshing continuously as the sliders are manipulated causing more and more of the machine’s graphic resources to be used, I placed a simple button that says, “change color” which causes the init() function to operate independently of the sliders. Another challenge I faced was the webpage had an unappealing user interface. I did most of the fixing using simple in-line CSS, and this made the application look much better. While it still is not perfect, the page looks acceptable now.

Figures 2 through 4 provide some examples of running the program.

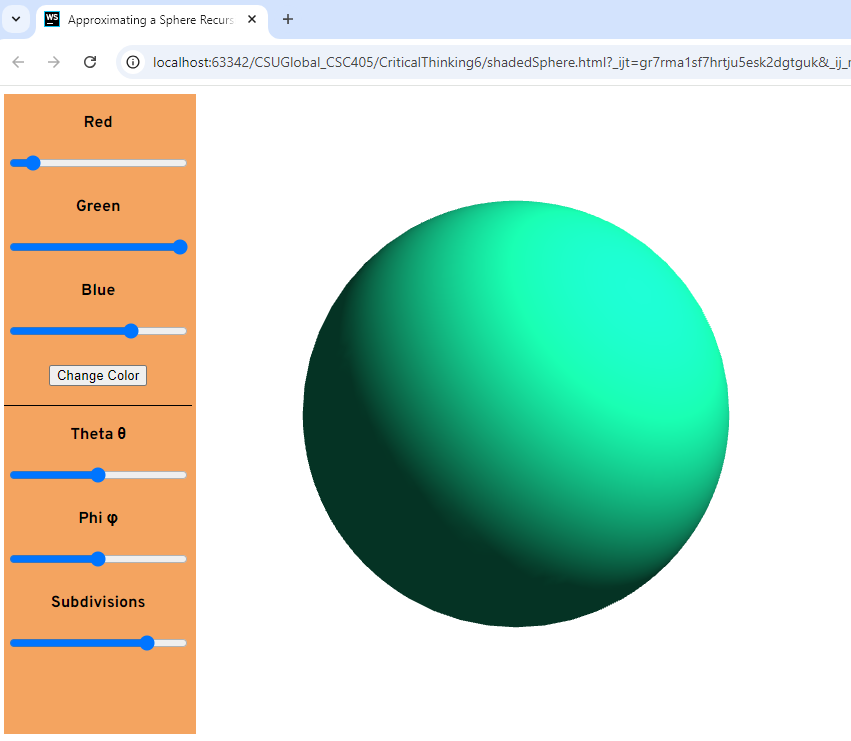
### Figure 1

*Example of the program running.*

*.*

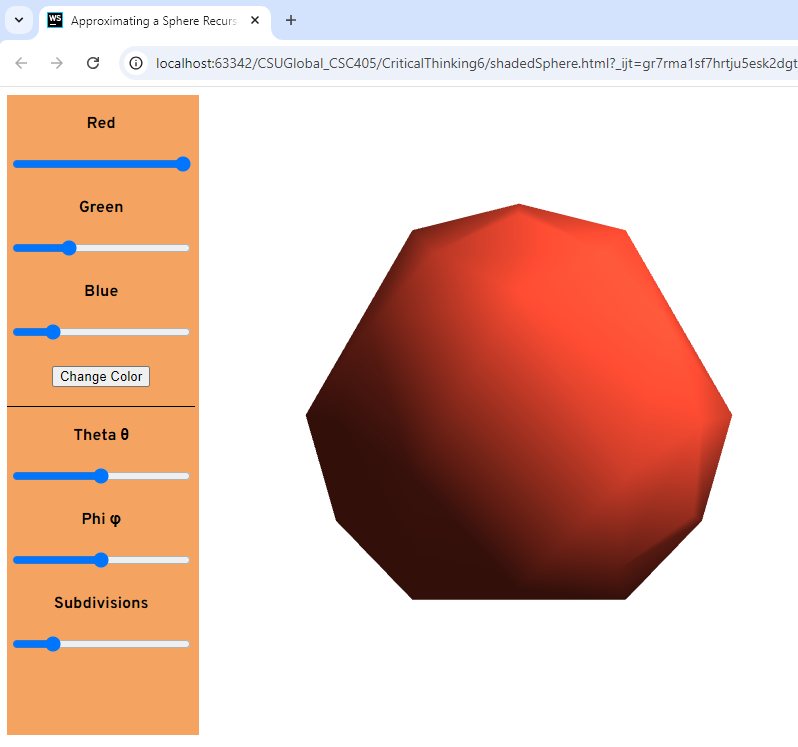
### Figure 2

*Another Example of the program running.*

*.*

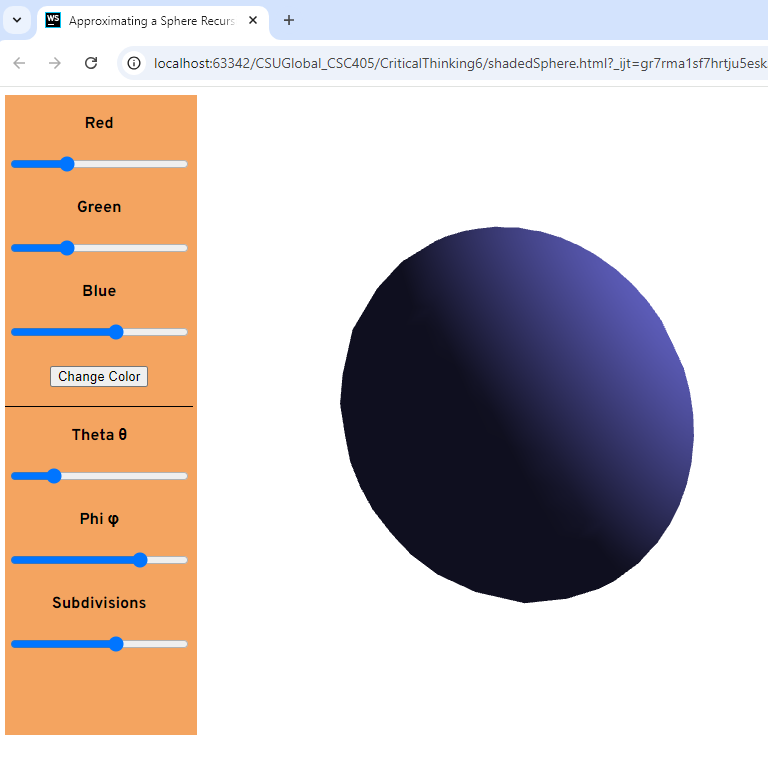
### Figure 3

*Another Example of the program running with less subdivisions (recursions).*

*.*

### Figure 4

*Another Example of the program running showing an ovaloid sphere with theta less than phi.*

*.*

## Conclusion

Curved surfaces are replicated in WebGL using straight sided polygons. The more polygons there are, and the smaller the polygons are, the better the curves will be rendered to a human eye on a 2D screen. Using light sources, shadows, and reflection adds to the realism of a rendered 3D object.

## References

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

Lei, K., Qi, D., & Tian, X. (2020). A new coordinate system for constructing spherical grid systems. *Applied Sciences*, *10*(2), 655. <https://doi.org/10.3390/app10020655>

Nießner, M., Keinert, B., Fisher, M., Stamminger, M., Loop, C., & Schäfer, H. (2015). Real-Time Rendering Techniques with Hardware Tessellation. *Computer Graphics Forum*, *35*(1), 113–137. <https://doi.org/10.1111/cgf.12714>