Reflection:

Module-4 Critical Thinking – 3D Colored Rotating Cube

Alejandro Ricciardi

Colorado State University Global

CSC405: Graphics and Visualization

Professor: Dr Jennifer Marquez

September 9, 2024

Reflection:

Module-4 Critical Thinking – 3D Rotating Cube

This reflection is part of Module 4 Critical Thinking – Colored Cube from CSC405: Graphics and Visualization at Colorado State University Global. It provides an overview and reflection on the program's functionality and output screenshots. As well as the steps I took to create the colored cube, an explanation of the role of shaders, buffers, and transformations in achieving the final result, and I reflect on what I have learned through this exercise. The program is titled "3D Rotating Cube" and it is based on WebGL and is coded using GLES 3, JavaScript, and HTML. It is a very simple WebGL application that generates and displays a 3D animation of a colored cube where the user can rotate the cube along the X, Y, and Z axes and move it up, down, left, and right. The user can also pause and restart the rotation while moving the cube. This program visits the concepts of transformation in computer graphics, more specifically rotation and transtlation.

The Transformations and Their Role

I implemented two types of transformations, a quaternion rotation transformation on the x, y, and z axis, and a translation transformation onlong the x and y. The transformations concatenate by type; for example, a rotation about the x-axis can concatenate with a translation along the y-axis or x-axis but not with a rotation about the z-axis.

Rotation

The quaternion rotation transformation is performed by the cube rotation animation. The animation is on an infinite loop; however, it can be paused by the user by clicking the stop button and restarted by clicking the start button. The cube rotation is based on quaternion-based rotations for each axis. The mathematical concept of quaternion is used in 3D computer graphics to compute

smooth rotation animations. It is a four-dimensional complex number of the form $q = w + xi + yj + zk$, where w, x, y, and z are real numbers, and i, j, and k are quaternions, usually vectors. (Angel & Shreiner). Using quaternion to implement a 3D rotation helps with issues like gimbal lock in Euler angles. It smooths object rotation animations. The role of the quaternion rotation transformation is to rotate the cube about the x, y, or z axis in a smooth fashion.

Translation

The translation transformation is the action of moving an object from one position to another in a space. This program is performed on the x-axis, it is controlled by the user by clicking on the Left or Right buttons, and on the y-axis, it is controlled by the user by clicking on the Up or Down buttons. In the program, the translation transformation is performed by adjusting the translation vector in the vertex shader. The role of translation is to move the cube a long the x or y-axis.

The Buttons

The buttons add a way for the user to interact with the program. The user can control the transformations by using the following buttons:

- The Reset Button: it resets the cube's position and rotation to its original state.
- The Rotate X Button: it rotates the cube around the x-axis.
- The Rotate Y Button: it rotates the cube around the y-axis.
- The Rotate Z Button: it rotates the cube around the z-axis.
- The Pause Rotation Button: It pauses the cube's rotation. Note that the cube can be moved up and down and left and right while the rotation animation is paused or while the rotation animation is running.
- The Restart Rotation Button: It restarts the cube's rotation.

- The Move Up Button: It moves up the cube, along the y-axis.
- The Move Down Button: It moves down the cube, along the y-axis.
- The Move Left Button: It moves the cube right, along the x-axis.
- The Move Left Button: It moves the cube left, along the x-axis.

Please see the following YouTube video, where I showcase the program functionality: 3D Rotating Cube WebGL (Ricciardi, 2024)

My Steps to Create the Rotating Cube

The first step that I took was to define the cube's geometry and location in 512 by 512 canvas in a 3D space. I also set the view, ‘viewport’, to the size of the canvas. This required setting up the vertex buffers to store the positions and colors of the cube's vertices. The geometry of the cube was initialized in the cube.js file, the cube is made of six faces each is made of two GLSL primitive triangles, and each triangle is made of three vertices or points. Then I define the transformations and the user controls Here is a detailed list describing the steps I took in chronological order:

- I set up the HTML code, to set the structure of the webpage. I defined the canvas tag with a width and height of 512x512 pixels to contain the generated graphics from WebGL. I also set the button tags knowing in advance what their functionality was going to be.
- I defined the vertex and fragment shaders script tabs and initialized the WebGl context and the GLSL code in the shader.
- I define the cube geometry, the cube is made of six faces each is made of two GLSL primitive triangles, and each triangle is made of three vertices or points. I defined the colorCube() function in the cube.js file. The function defined the vertices of the

cube’s geometry and assigned a different color to each face. I used the function quad() to define each face, each quad object represents a four-sided polygon, a square in this program. The vertex positions were stored in a buffer (vBuffer variable in the code), and the vertex colors were stored in a separate buffer (cBuffer variable in the code).

- I bound the buffers and passed the data to the GPU, that is the buffers are bound to the ARRAY_BUFFER and to sent to the GPU. Then I defined the formats of the vertex position buffer (positionLoc) and the vertex color buffer (colorLoc) to be bused during rendering.

- I set up the uniform Variables for the rotation (uThetha) and translation transformation (uTranslation), they are used to control the cube’s rotation and translation. Uniform variables are global values passed from the CPU-side application (JavaScript), their values are the same for all vertices in a single draw call. A draw call is a command issued by the CPU to the GPU to render (or "draw"). In other words, uniform variables are WebGL constants, they are set once per draw call from the JavaScript code (CPU-side application), they remain the same for all vertices in that specific draw call.

- I added event listeners in the JavaScript file that I linked to the buttons tab in the HTML file. This implemented a user interface making it possible for the user to control the cube's graphical transformations, that is pause-start rotation, up-down, and right-left cube movement.

- And finally, I implemented the render function responsible for drawing the cube on the canvas. The function is recursive creating a loop that is redrawn at every frame based on user input and the cube rotation.

I planned each step in advance by drawing program diagrams and tried the transformations one at a time ensuring that they function properly.

The Roles of Shaders and Buffers

The vertex shader handles the transformation of the cube. It computes the vertex's position. It computes the rotation using quaternion mathematics and translation vectors to move the cube along the x and y axes. The fragment shader handles the cube's face colors. After the vertex shader computes the positions of the vertices, the fragment shader allocates the assigned colors to each vertex and determines the color of each pixel that makes up the cube face surfaces. On the other hand, the buffer transfers the data to the GPU before being computed by the shaders. The shaders compute the data using the GPU. In this program, I used a position buffer (vBuffer): to store the cube's vertex positions and a color buffer (cBuffer) to store the colors assigned to each face of the cube. Note that the buffer used to transfer the data to the GPU is the ARRAY_BUFFER and both the position and color buffer data a transferred to this buffer before transferring.

What I Learned

I learned quite a bit during the process of making this program. I learned how to implement transformations in 3D space using WebGL. The quaternion rotation transformation was challenging but also very rewarding I also acquired a better understanding of the role the buffer plays and how WebGL handles the CPU and GPU levels, and how the vertex and fragment shaders work together

to create 3D graphics and animation. Adding the buttons to the project has enhanced my understanding of how to link animation and user interface. Overall, the process of making the program was challenging but it was incredibly educative. It has augmented my skills in programming 3D graphics and provided me with the foundational knowledge necessary to create more advanced projects in graphics and visualization.

References

Angel, E., & Shreiner, D. (2020). Chapter 4 geometric objects and transformations. *Interactive computer graphics*. 8th edition. Pearson Education, Inc. ISBN: 9780135258262

Ricciardi A. (2024, August 6). *3D rotating cube WebGL* [Video]. YouTube. https://www.youtube.com/watch?v=peIjP2O0FTU