Shader Time - HW 1 - 1 / 27 / 2026

Prompt

Create a reactive 3D mesh. Use p5.js and WebGL. Use the version of vertex() that includes the z parameter. Use both position and color attributes. Make your mesh react to user input. This can be using the mouse, keyboard, microphone, or any other input you can think of. If you have an idea but are not sure how to do it, ask on Discord and we can break it down and figure it out together.

p5 Docs on Vertex Object

Implementation Plan

1. Create Canvas (Grey Background, autosizes to the browser)
2. Create 3 red buttons which flash blue when clicked/triggered:
   - Clear – Removes all vertices and edges
   - Close – draws edges between vertices
   - Step-Up – steps up the drawing from 2D to 3D to 4D projection (loops back to 2D)
3. Click event handler
   - If cursor is over a button, don't draw a vertex on user click gesture
   - Else, if cursor is over the canvas and the user clicks, draw a vertex there (vertices should be little white dots)
     - Vertices should be stored in a data structure. This data structure contains the index (first vertex drawn, second vertex drawn, etc…), and the coordinates in 2D space of that vertex (a tuple, containing X/Y coordinates)

4. Buttons logic
   - If the user clicks the Clear button, clear all vertex data in the data structure, and refresh the display to reflect this deletion. The Clear button can be clicked at any time.
   - If the user clicks the Close button, draw an edge between all the vertices that have been drawn in 2D. This button can only be clicked after there have been at least 2 vertices drawn. If the user has clicked the Step-Up button, and it was a valid signal, the Close button should be disabled until the Clear button is clicked. The edges should be drawn as thin, transparent, fluorescent green lines.
   - If the user clicks the Step-Up button, and the data structure reflects in 2D space at least 2 points that have been closed (that is to say, had their edges drawn), then the following process occurs:
     - Determine Z-amount: Run a recursive function which takes the average of all results of this loop: For each vertex “j”, calculate the distance between j and the previous/next vertices contained in the data structure.
     - With that Z-amount having been determined, copy the 2D graphic and offset it by that z-amount away from the camera. Close the edges between each vertex in the clone and its corresponding vertex in the original. This way, if the user has drawn a 2D square, The Step-Up button renders a 3D cube.
     - By the same logic, extrapolate a 3D projection slice of a 4D Hypercube when the user Steps-Up a 3D scene. If they click Step-Up on a 4D projection render, return to displaying the 2D scene graphic… this should loop.

- In 3D and 4D Projection renders, slowly rotate the graphic clockwise perpendicular to the screen, so as to accentuate the visual depth.

Feed implementation plan into Claude Code, asking for a file structure plan

Prompt: Read PROJECT_PROPOSAL.md. We are going to cloudflare this from a home server – propose a file structure for a basic webpage with the desired functionality using p5.js with WEBGL. When designing your implementation plan, adhere to UNIX philosophy – create a suite of short, single-purpose helper files which work together in the webpage's main loop. Anticipate any oversights in what has been proposed already, and provide a comprehensive implementation guide, as well as a proposed method for testing implementation.

User Test Claude's Generated Solution, Put on Github & Cloudflare it to the Internet ; Reflection Checkpoint

At this point, I've been working on this for about an hour. The Claude solution was clean, only with 1 bug, which was easily resolved.

This solution does not portray a true 3D projection of a 4D shape; the rotation animation is merely visual, rather than truly a mathematical tour through 4D space with a 3D lens. Mathematically and in terms of CS implementation, what would it take to move a projected 3D slice through a 4D shape and render its edges into a 2D projection on a monitor?

My plan is, now, to spend a couple of hours reviewing the code, trying to understand it, and documenting what I learn.

Understanding the Codebase

1. index.html calls the following scripts in order:
    - state, vertex, edges, buttons, stepup, projection, rotation, sketch, tests
2. state.js instantiates a class object called "State" which contains all the data and methods necessary to maintain the state of the application. This includes the following:
    - Vertex data structures for 2D, 3D, and 4D rendering (including a structure that tracks edge pairings for the 4D render)
    - Boolean flag for whether or not the vertices have been closed and edges have been rendered
    - Variables for tracking z-offset for the 3D render, w-offset for the 4D render, and rotation angle for animating 3D & 4D renders
    - Boolean flags for when buttons are pressed
3. vertex.js is an event handler for when the user clicks. It latches onto the top of the 2D vertex data structure, pushes the coordinates of the new vertex down by 1 index (storing them in the data structure), then returns 2 function calls to update the system state.
    - It should be noted here: this implementation is not using the p5.js vertex() function, nor the endShape(CLOSE) function – instead, it's recreating that functionality from scratch, so the same, mildly more complicated logic being used for drawing edges in the 4D render can be repeatedly applied in the 2D and 3D renders. Does this technically fail the assignment? Maybe, since I don't have any way of shading or palatting the faces formed by the edges drawn in this

way… it's sort of a reverse-engineer of the tools we were assigned to use… also, instead of frameCount or deltaTime, it uses dt (which I believe is another reference to deltaTime) scaled with a tickrate called SPEED in rotation.js (is this another deviation?)