Project 5: Perspective

Introduction

Now we know enough to add perspective-correct rendering. Let's keep it simple with just a spinning cube.

Goals:

* Draw a cube mesh using elements.
* Make it spin clockwise at an eighth-turn (.125 tau) per second
* Make it perspective correct with a field of view of 90 degrees (.25 tau)
* Either use the shader from the "Hints" section to show the depth, or make every vertex of the cube a different color.
* Your results should look like "right.gif" (I'm using the shader, but you don't have to)

Steps:

* Use the slides to create a function that creates a frustum projection matrix.
* Then, use the slides to create a function that creates a perspective matrix using the frustum function.
* Use this matrix to project your spinning cube with perspective.

Hints

If you don't want to write cube-mesh code, you can make an .obj file by hand with the correct vertices and indices, and then use your .obj project.

you can use this shader to help debug your depth buffer:

let fragment\_source =   
    `   #version 300 es  
        precision mediump float;  
  
        in vec4 v\_color;  
  
        out vec4 f\_color;  
  
        void main( void ) {  
            /\* we can test depth values with this.\*/  
            f\_color = vec4(vec3(gl\_FragCoord.z), 1.0);   
        }  
    `;

Frustum Code

You can use this code to create a frustum. You will be using our FOV to frustum conversion math to generate the correct perspective matrix.

//located inside of the Mat4 class  
static frustum( left, right, bottom, top, near, far ) {   
        // these scalars will scale x,y values to the near plane  
        let scale\_x = 2 \* near / ( right - left );  
        let scale\_y = 2 \* near / ( top - bottom );  
  
        // shift the eye depending on the right/left and top/bottom planes.  
        // only really used for VR (left eye and right eye shifted differently).    
        let t\_x = ( right + left ) / ( right - left );  
        let t\_y = ( top + bottom ) / ( top - bottom );  
  
        // map z into the range [ -1, 1 ] linearly  
        const linear\_c2 = 1 / ( far - near );  
        const linear\_c1 = near / ( far - near );  
        // remember that the w coordinate will always be 1 before being fed to the vertex shader.  
        // therefore, anything we put in row 3, col 4 of the matrix will be added to the z.  
  
        // map z into the range [ -1, 1], but with a non-linear ramp  
        // see: https://learnopengl.com/Advanced-OpenGL/Depth-testing and  
        // https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix and  
        // http://learnwebgl.brown37.net/08\_projections/projections\_perspective.html  
        // for more info. (note, I'm using left-handed coordinates. Some sources use right-handed).  
        const nonlin\_c2 = (far + near) / (far - near);  
        const nonlin\_c1 = 2 \* far \* near / (far - near);  
  
        let c1 = nonlin\_c1;  
        let c2 = nonlin\_c2;  
  
        return new Mat4( [  
            scale\_x,    0,          t\_x, 0,  
            0,          scale\_y,    t\_y, 0,  
            0,          0,          c2, -c1,  
            0,          0,          1, 0,   
        ] );  
    }