<canvas id="canvas"></canvas>

<!-- vertex shader -->

<script  id="vertex-shader-3d" type="x-shader/x-vertex">

attribute vec4 a\_position;

attribute vec2 a\_texcoord;

uniform mat4 u\_matrix;

varying vec2 v\_texcoord;

void main() {

  // Multiply the position by the matrix.

  gl\_Position = u\_matrix \* a\_position;

  // Pass the texcoord to the fragment shader.

  v\_texcoord = a\_texcoord;

}

</script>

<!-- fragment shader -->

<script  id="fragment-shader-3d" type="x-shader/x-fragment">

precision mediump float;

// Passed in from the vertex shader.

varying vec2 v\_texcoord;

// The texture.

uniform sampler2D u\_texture;

void main() {

   gl\_FragColor = texture2D(u\_texture, v\_texcoord);

}

</script><!--

for most samples webgl-utils only provides shader compiling/linking and

canvas resizing because why clutter the examples with code that's the same in every sample.

See https://webglfundamentals.org/webgl/lessons/webgl-boilerplate.html

and https://webglfundamentals.org/webgl/lessons/webgl-resizing-the-canvas.html

for webgl-utils, m3, m4, and webgl-lessons-ui.

-->

<script src="https://webglfundamentals.org/webgl/resources/webgl-utils.js"></script>

<script src="https://webglfundamentals.org/webgl/resources/m4.js"></script>

@import url("https://webglfundamentals.org/webgl/resources/webgl-tutorials.css");

body {

  margin: 0;

}

canvas {

  width: 100vw;

  height: 100vh;

  display: block;

}

"use strict";

function main() {

  // Get A WebGL context

  /\*\* @type {HTMLCanvasElement} \*/

  var canvas = document.querySelector("#canvas");

  var gl = canvas.getContext("webgl");

  if (!gl) {

    return;

  }

  // setup GLSL program

  var program = webglUtils.createProgramFromScripts(gl, ["vertex-shader-3d", "fragment-shader-3d"]);

  // look up where the vertex data needs to go.

  var positionLocation = gl.getAttribLocation(program, "a\_position");

  var texcoordLocation = gl.getAttribLocation(program, "a\_texcoord");

  // lookup uniforms

  var matrixLocation = gl.getUniformLocation(program, "u\_matrix");

  var textureLocation = gl.getUniformLocation(program, "u\_texture");

  // Create a buffer for positions

  var positionBuffer = gl.createBuffer();

  // Bind it to ARRAY\_BUFFER (think of it as ARRAY\_BUFFER = positionBuffer)

  gl.bindBuffer(gl.ARRAY\_BUFFER, positionBuffer);

  // Put the positions in the buffer

  setGeometry(gl);

  // provide texture coordinates for the rectangle.

  var texcoordBuffer = gl.createBuffer();

  gl.bindBuffer(gl.ARRAY\_BUFFER, texcoordBuffer);

  // Set Texcoords.

  setTexcoords(gl);

  // Create a texture.

  var texture = gl.createTexture();

  gl.bindTexture(gl.TEXTURE\_2D, texture);

  // Fill the texture with a 1x1 blue pixel.

  gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED\_BYTE,

                new Uint8Array([0, 0, 255, 255]));

  // Asynchronously load an image

  var image = new Image();

  image.src = "https://webglfundamentals.org/webgl/resources/f-texture.png";

  image.addEventListener('load', function() {

    // Now that the image has loaded make copy it to the texture.

    gl.bindTexture(gl.TEXTURE\_2D, texture);

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED\_BYTE, image);

    gl.generateMipmap(gl.TEXTURE\_2D);

  });

  function radToDeg(r) {

    return r \* 180 / Math.PI;

  }

  function degToRad(d) {

    return d \* Math.PI / 180;

  }

  var fieldOfViewRadians = degToRad(60);

  var modelXRotationRadians = degToRad(0);

  var modelYRotationRadians = degToRad(0);

  var then = 0;

  requestAnimationFrame(drawScene);

  // Draw the scene.

  function drawScene(now) {

    // Convert to seconds

    now \*= 0.001;

    // Subtract the previous time from the current time

    var deltaTime = now - then;

    // Remember the current time for the next frame.

    then = now;

    webglUtils.resizeCanvasToDisplaySize(gl.canvas);

    // Tell WebGL how to convert from clip space to pixels

    gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

    gl.enable(gl.CULL\_FACE);

    gl.enable(gl.DEPTH\_TEST);

    // Animate the rotation

    modelXRotationRadians += 1.2 \* deltaTime;

    modelYRotationRadians += 0.7 \* deltaTime;

    // Clear the canvas AND the depth buffer.

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    // Tell it to use our program (pair of shaders)

    gl.useProgram(program);

    // Turn on the position attribute

    gl.enableVertexAttribArray(positionLocation);

    // Bind the position buffer.

    gl.bindBuffer(gl.ARRAY\_BUFFER, positionBuffer);

    // Tell the position attribute how to get data out of positionBuffer (ARRAY\_BUFFER)

    var size = 3;          // 3 components per iteration

    var type = gl.FLOAT;   // the data is 32bit floats

    var normalize = false; // don't normalize the data

    var stride = 0;        // 0 = move forward size \* sizeof(type) each iteration to get the next position

    var offset = 0;        // start at the beginning of the buffer

    gl.vertexAttribPointer(

        positionLocation, size, type, normalize, stride, offset);

    // Turn on the texcoord attribute

    gl.enableVertexAttribArray(texcoordLocation);

    // bind the texcoord buffer.

    gl.bindBuffer(gl.ARRAY\_BUFFER, texcoordBuffer);

    // Tell the texcoord attribute how to get data out of texcoordBuffer (ARRAY\_BUFFER)

    var size = 2;          // 2 components per iteration

    var type = gl.FLOAT;   // the data is 32bit floats

    var normalize = false; // don't normalize the data

    var stride = 0;        // 0 = move forward size \* sizeof(type) each iteration to get the next position

    var offset = 0;        // start at the beginning of the buffer

    gl.vertexAttribPointer(

        texcoordLocation, size, type, normalize, stride, offset);

    // Compute the projection matrix

    var aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;

    var projectionMatrix =

        m4.perspective(fieldOfViewRadians, aspect, 1, 2000);

    var cameraPosition = [0, 0, 200];

    var up = [0, 1, 0];

    var target = [0, 0, 0];

    // Compute the camera's matrix using look at.

    var cameraMatrix = m4.lookAt(cameraPosition, target, up);

    // Make a view matrix from the camera matrix.

    var viewMatrix = m4.inverse(cameraMatrix);

    var viewProjectionMatrix = m4.multiply(projectionMatrix, viewMatrix);

    var matrix = m4.xRotate(viewProjectionMatrix, modelXRotationRadians);

    matrix = m4.yRotate(matrix, modelYRotationRadians);

    // Set the matrix.

    gl.uniformMatrix4fv(matrixLocation, false, matrix);

    // Tell the shader to use texture unit 0 for u\_texture

    gl.uniform1i(textureLocation, 0);

    // Draw the geometry.

    gl.drawArrays(gl.TRIANGLES, 0, 16 \* 6);

    requestAnimationFrame(drawScene);

  }

}

// Fill the buffer with the values that define a letter 'F'.

function setGeometry(gl) {

  var positions = new Float32Array([

          // left column front

          0,   0,  0,

          0, 150,  0,

          30,   0,  0,

          0, 150,  0,

          30, 150,  0,

          30,   0,  0,

          // top rung front

          30,   0,  0,

          30,  30,  0,

          100,   0,  0,

          30,  30,  0,

          100,  30,  0,

          100,   0,  0,

          // middle rung front

          30,  60,  0,

          30,  90,  0,

          67,  60,  0,

          30,  90,  0,

          67,  90,  0,

          67,  60,  0,

          // left column back

            0,   0,  30,

           30,   0,  30,

            0, 150,  30,

            0, 150,  30,

           30,   0,  30,

           30, 150,  30,

          // top rung back

           30,   0,  30,

          100,   0,  30,

           30,  30,  30,

           30,  30,  30,

          100,   0,  30,

          100,  30,  30,

          // middle rung back

           30,  60,  30,

           67,  60,  30,

           30,  90,  30,

           30,  90,  30,

           67,  60,  30,

           67,  90,  30,

          // top

            0,   0,   0,

          100,   0,   0,

          100,   0,  30,

            0,   0,   0,

          100,   0,  30,

            0,   0,  30,

          // top rung right

          100,   0,   0,

          100,  30,   0,

          100,  30,  30,

          100,   0,   0,

          100,  30,  30,

          100,   0,  30,

          // under top rung

          30,   30,   0,

          30,   30,  30,

          100,  30,  30,

          30,   30,   0,

          100,  30,  30,

          100,  30,   0,

          // between top rung and middle

          30,   30,   0,

          30,   60,  30,

          30,   30,  30,

          30,   30,   0,

          30,   60,   0,

          30,   60,  30,

          // top of middle rung

          30,   60,   0,

          67,   60,  30,

          30,   60,  30,

          30,   60,   0,

          67,   60,   0,

          67,   60,  30,

          // right of middle rung

          67,   60,   0,

          67,   90,  30,

          67,   60,  30,

          67,   60,   0,

          67,   90,   0,

          67,   90,  30,

          // bottom of middle rung.

          30,   90,   0,

          30,   90,  30,

          67,   90,  30,

          30,   90,   0,

          67,   90,  30,

          67,   90,   0,

          // right of bottom

          30,   90,   0,

          30,  150,  30,

          30,   90,  30,

          30,   90,   0,

          30,  150,   0,

          30,  150,  30,

          // bottom

          0,   150,   0,

          0,   150,  30,

          30,  150,  30,

          0,   150,   0,

          30,  150,  30,

          30,  150,   0,

          // left side

          0,   0,   0,

          0,   0,  30,

          0, 150,  30,

          0,   0,   0,

          0, 150,  30,

          0, 150,   0]);

  // Center the F around the origin and Flip it around. We do this because

  // we're in 3D now with and +Y is up where as before when we started with 2D

  // we had +Y as down.

  // We could do by changing all the values above but I'm lazy.

  // We could also do it with a matrix at draw time but you should

  // never do stuff at draw time if you can do it at init time.

  var matrix = m4.identity();// m4.xRotation(Math.PI);

  matrix = m4.translate(matrix, -50, -75, -15);

  for (var ii = 0; ii < positions.length; ii += 3) {

    var vector = m4.transformVector(matrix, [positions[ii + 0], positions[ii + 1], positions[ii + 2], 1]);

    positions[ii + 0] = vector[0];

    positions[ii + 1] = vector[1];

    positions[ii + 2] = vector[2];

  }

  gl.bufferData(gl.ARRAY\_BUFFER, positions, gl.STATIC\_DRAW);

}

// Fill the buffer with texture coordinates the F.

function setTexcoords(gl) {

  gl.bufferData(

      gl.ARRAY\_BUFFER,

      new Float32Array([

        // left column front

        0, 0,

        0, 1,

        1, 0,

        0, 1,

        1, 1,

        1, 0,

        // top rung front

        0, 0,

        0, 1,

        1, 0,

        0, 1,

        1, 1,

        1, 0,

        // middle rung front

        0, 0,

        0, 1,

        1, 0,

        0, 1,

        1, 1,

        1, 0,

        // left column back

        0, 0,

        1, 0,

        0, 1,

        0, 1,

        1, 0,

        1, 1,

        // top rung back

        0, 0,

        1, 0,

        0, 1,

        0, 1,

        1, 0,

        1, 1,

        // middle rung back

        0, 0,

        1, 0,

        0, 1,

        0, 1,

        1, 0,

        1, 1,

        // top

        0, 0,

        1, 0,

        1, 1,

        0, 0,

        1, 1,

        0, 1,

        // top rung right

        0, 0,

        1, 0,

        1, 1,

        0, 0,

        1, 1,

        0, 1,

        // under top rung

        0, 0,

        0, 1,

        1, 1,

        0, 0,

        1, 1,

        1, 0,

        // between top rung and middle

        0, 0,

        1, 1,

        0, 1,

        0, 0,

        1, 0,

        1, 1,

        // top of middle rung

        0, 0,

        1, 1,

        0, 1,

        0, 0,

        1, 0,

        1, 1,

        // right of middle rung

        0, 0,

        1, 1,

        0, 1,

        0, 0,

        1, 0,

        1, 1,

        // bottom of middle rung.

        0, 0,

        0, 1,

        1, 1,

        0, 0,

        1, 1,

        1, 0,

        // right of bottom

        0, 0,

        1, 1,

        0, 1,

        0, 0,

        1, 0,

        1, 1,

        // bottom

        0, 0,

        0, 1,

        1, 1,

        0, 0,

        1, 1,

        1, 0,

        // left side

        0, 0,

        0, 1,

        1, 1,

        0, 0,

        1, 1,

        1, 0]),

      gl.STATIC\_DRAW);

}

main();