**HTML**

<canvas id="canvas"></canvas>

<div id="uiContainer">

<div id="ui">

<div id="b">Click to switch texture</div>

</div>

</div>

<!-- 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>

**CSS**

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

body {

margin: 0;

}

canvas {

width: 100vw;

height: 100vh;

display: block;

}

#ui {

width: 100px;

background-color: red;

padding: 0.25em;

}

body, document {

user-select: none;

-webkit-user-select: none;

-moz-user-select: none;

-o-user-select: none;

-ms-user-select: none;

}

#uiContainer {

left: 10px;

top: 130px;

}

**JS**

// WebGL - Textures - Mips - Depth

// from https://webglfundamentals.org/webgl/webgl-3d-textures-mips-tri-linear.html

"use strict";

var zDepth = 50;

function main() {

// Get A WebGL context

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

var gl = canvas.getContext("webgl", {antialias: false});

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 with different colored mips

var mipTexture = gl.createTexture();

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

var c = document.createElement("canvas");

var ctx = c.getContext("2d");

var mips = [

{ size: 64, color: "rgb(128,0,255)", },

{ size: 32, color: "rgb(0,0,255)", },

{ size: 16, color: "rgb(255,0,0)", },

{ size: 8, color: "rgb(255,255,0)", },

{ size: 4, color: "rgb(0,255,0)", },

{ size: 2, color: "rgb(0,255,255)", },

{ size: 1, color: "rgb(255,0,255)", },

];

mips.forEach(function(s, level) {

var size = s.size;

c.width = size;

c.height = size;

ctx.fillStyle = "rgb(255,255,255)";

ctx.fillRect(0, 0, size, size);

ctx.fillStyle = s.color;

ctx.fillRect(0, 0, size / 2, size / 2);

ctx.fillRect(size / 2, size / 2, size / 2, size / 2);

gl.texImage2D(gl.TEXTURE\_2D, level, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, c);

});

// 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();

requestCORSIfNotSameOrigin(image, "https://webglfundamentals.org/webgl/resources/mip-low-res-example.png")

image.src = "https://webglfundamentals.org/webgl/resources/mip-low-res-example.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.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

// Check if the image is a power of 2 in both dimensions.

if (isPowerOf2(image.width) && isPowerOf2(image.height)) {

// Yes, it's a power of 2. Generate mips.

gl.generateMipmap(gl.TEXTURE\_2D);

} else {

// No, it's not a power of 2. Turn of mips and set wrapping to clamp to edge

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

}

drawScene();

});

var textures = [

texture,

mipTexture,

];

var textureIndex = 0;

document.body.addEventListener('click', function() {

textureIndex = (textureIndex + 1) % textures.length;

drawScene();

});

function isPowerOf2(value) {

return (value & (value - 1)) === 0;

}

function radToDeg(r) {

return r \* 180 / Math.PI;

}

function degToRad(d) {

return d \* Math.PI / 180;

}

var fieldOfViewRadians = degToRad(60);

drawScene();

// Draw the scene.

function drawScene() {

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);

// Clear the framebuffer texture.

gl.clearColor(0, 0, 0, 1);

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 zNear = 1;

var zFar = 2000;

var projectionMatrix =

m4.perspective(fieldOfViewRadians, aspect, zNear, zFar);

var cameraPosition = [0, 0, 2];

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 settings = [

{ x: -1, y: 1, zRot: 0, magFilter: gl.NEAREST, minFilter: gl.NEAREST, },

{ x: 0, y: 1, zRot: 0, magFilter: gl.LINEAR, minFilter: gl.LINEAR, },

{ x: 1, y: 1, zRot: 0, magFilter: gl.LINEAR, minFilter: gl.NEAREST\_MIPMAP\_NEAREST, },

{ x: -1, y: -1, zRot: 1, magFilter: gl.LINEAR, minFilter: gl.LINEAR\_MIPMAP\_NEAREST, },

{ x: 0, y: -1, zRot: 1, magFilter: gl.LINEAR, minFilter: gl.NEAREST\_MIPMAP\_LINEAR, },

{ x: 1, y: -1, zRot: 1, magFilter: gl.LINEAR, minFilter: gl.LINEAR\_MIPMAP\_LINEAR, },

];

var xSpacing = 1.2;

var ySpacing = 0.7;

settings.forEach(function(s) {

gl.bindTexture(gl.TEXTURE\_2D, textures[textureIndex]);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, s.minFilter);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, s.magFilter);

var matrix = m4.translate(viewProjectionMatrix, s.x \* xSpacing, s.y \* ySpacing, -zDepth \* 0.5);

matrix = m4.zRotate(matrix, s.zRot \* Math.PI);

matrix = m4.scale(matrix, 1, 1, zDepth);

// 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, 1 \* 6);

});

}

}

// Fill the buffer with the values that define a plane.

function setGeometry(gl) {

var positions = new Float32Array(

[

-0.5, 0.5, -0.5,

0.5, 0.5, -0.5,

-0.5, 0.5, 0.5,

-0.5, 0.5, 0.5,

0.5, 0.5, -0.5,

0.5, 0.5, 0.5,

]);

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

}

// Fill the buffer with texture coordinates for a plane.

function setTexcoords(gl) {

gl.bufferData(

gl.ARRAY\_BUFFER,

new Float32Array(

[

0, 0,

1, 0,

0, zDepth,

0, zDepth,

1, 0,

1, zDepth,

]),

gl.STATIC\_DRAW);

}

main();

// This is needed if the images are not on the same domain

// NOTE: The server providing the images must give CORS permissions

// in order to be able to use the image with WebGL. Most sites

// do NOT give permission.

// See: https://webglfundamentals.org/webgl/lessons/webgl-cors-permission.html

function requestCORSIfNotSameOrigin(img, url) {

if ((new URL(url, window.location.href)).origin !== window.location.origin) {

img.crossOrigin = "";

}

}