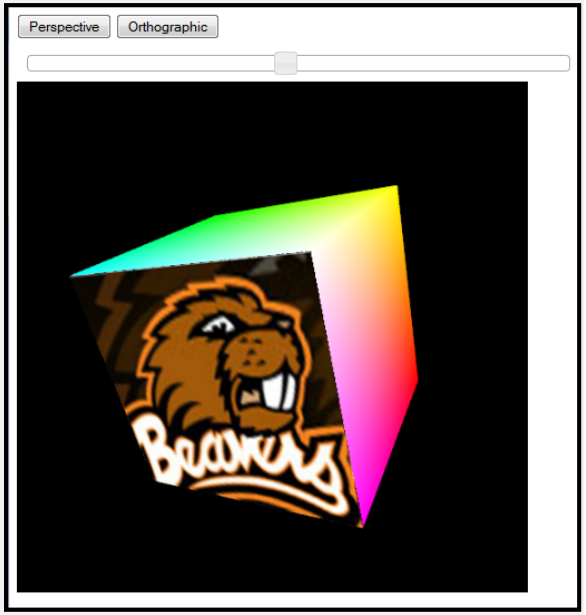
WebGL

http://cs.oregonstate.edu/~mjb/webgl

Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu



Oregon State University
Computer Graphics

What is "WebGL"?

From WikiPedia:

WebGL (*Web Graphics Library*) is a JavaScript API for rendering interactive 3D graphics and 2D graphics within any compatible web browser without the use of plug-ins. WebGL is integrated completely into all the web standards of the browser allowing GPU accelerated usage of physics and image processing and effects as part of the web page canvas. WebGL elements can be mixed with other HTML elements and composited with other parts of the page or page background. WebGL programs consist of control code written in JavaScript and shader code that is executed on a computer's Graphics Processing Unit (GPU). WebGL is designed and maintained by the non-profit Khronos Group.

[… of which OSU is a member]

Oregon State University
Computer Graphics

What are the Important Things to Know about WebGL?

1. WebGL programs are written in JavaScript. This means that the code is downloaded and run by the browser.

2. WebGL programs are written in JavaScript. This means that the code is interpreted, not compiled. This makes the code slower than the equivalent C/C++.

3. The JavaScript API for WebGL talks to the C-OpenGL library which in turn talks to the graphics hardware on the web browser client's system. This gives you access to all of the speed and capability that the hardware provides. Use it! Especially use the vertex buffer object capability to store your displayable geometry on the graphics card, and be sure to bind your textures into GPU memory. Anything that you can put on the GPU side minimizes the amount of code necessary to get it drawn.

4. WebGL uses the OpenGL-ES 2.0 as its graphics API. "ES" stands for "Embedded Systems". This is the same flavor of OpenGL that mobile devices use.

What are the Important Things to Know about OpenGL-ES 2.0?

1. WebGL uses OpenGL-ES 2.0 as its graphics API.

2. There is no fixed-function pipeline -- you must use vertex and fragment shaders.

3. You can't use **glBegin…glEnd** to draw things. You must use VBOs.

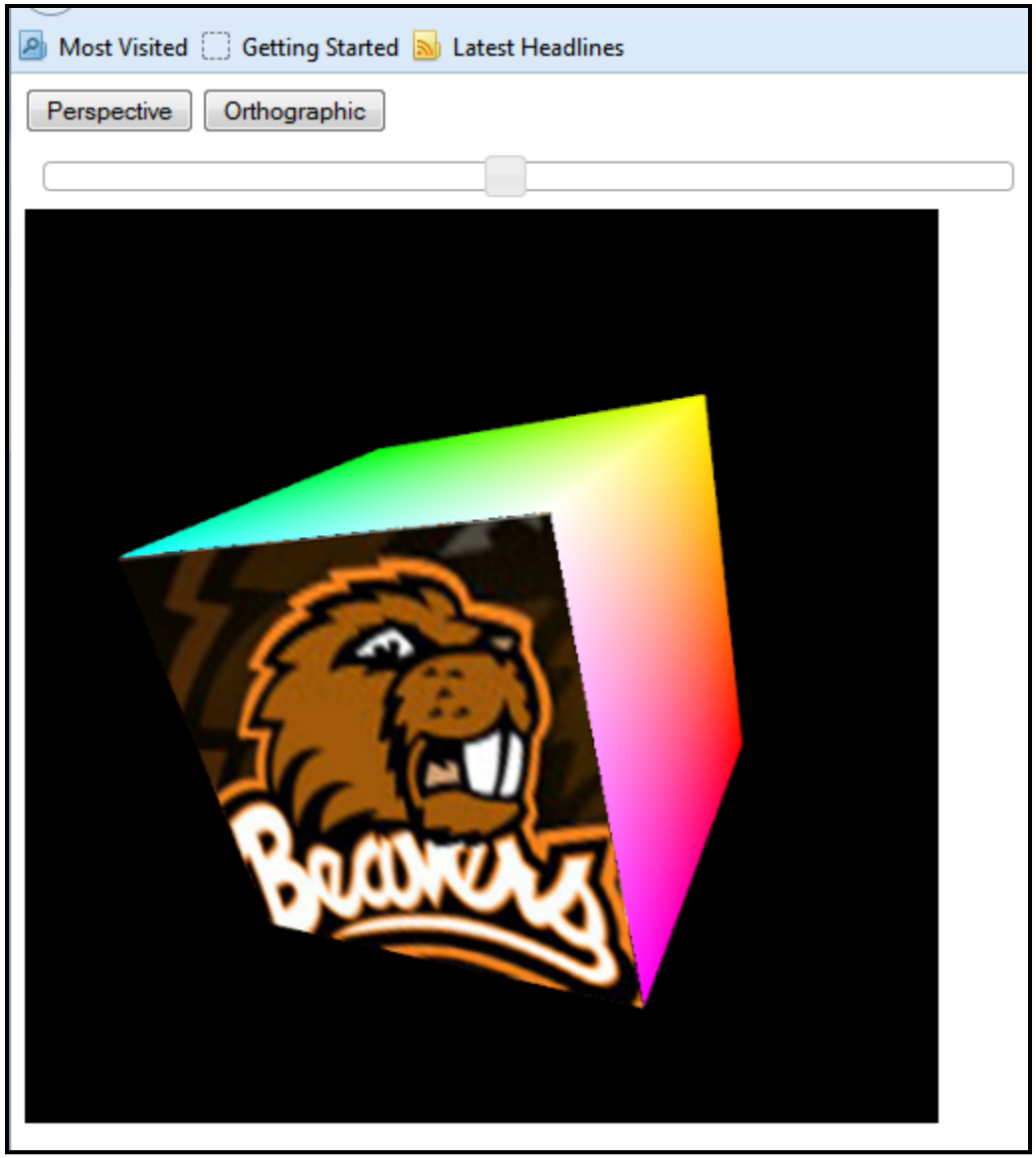
4. You can't use built-in transformation functions (e.g., **glRotatef**, **glScalef**, **glTranslatef, gluLookAt, glOrtho, gluPerspective**). You must use your own matrices passed in as vertex shader attribute variables.

5. Attribute variables for the vertex shader are declared as "attribute", not "in".

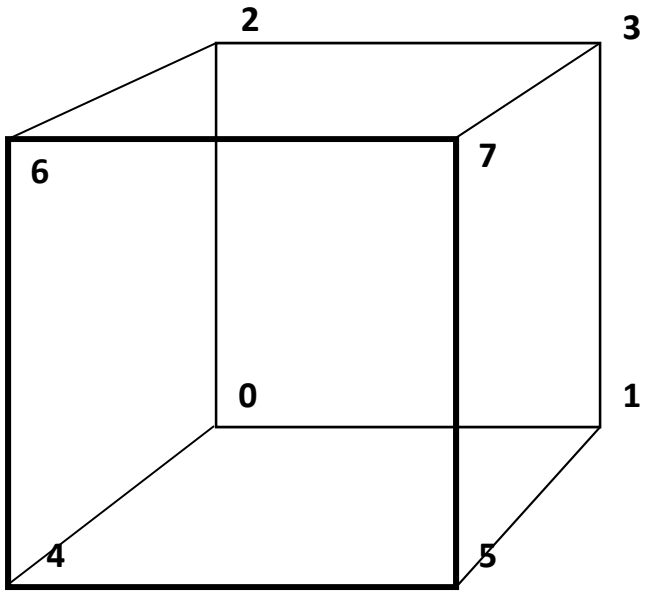
6. Output variables from the vertex shader which are then rasterized to become input variables for the fragment shader are declared as "varying", not "out" and "in".

7. Fragment shaders must set the precision to be used, for example:
   precision highp float;

Cube Example



Cube Example



Cube Vertices =

```
{ -1., -1., -1. },
{  1., -1., -1. },
{ -1.,  1., -1. },
{  1.,  1., -1. },
{ -1., -1.,  1. },
{  1., -1.,  1. },
{ -1.,  1.,  1. },
{  1.,  1.,  1. }
```

Cube Colors =

```
{ 0., 0., 0. },
{ 1., 0., 0. },
{ 0., 1., 0. },
{ 1., 1., 0. },
{ 0., 0., 1. },
{ 1., 0., 1. },
{ 0., 1., 1. },
{ 1., 1., 1. }
```

Cube Indices =

```
{ 1, 0, 2, 3 },
{ 4, 5, 7, 6 },
{ 5, 1, 3, 7 },
{ 0, 4, 6, 2 },
{ 6, 7, 3, 2 },
{ 0, 1, 5, 4 }
```

A Sample WebGL Program

Some things to note:

1. This code displays a cube with 5 colored sides and one textured side. The cube is rotated in 3D by holding down the left mouse button and moving the mouse. It is scaled by using a jQuery-generated slider. The display is toggled between perspective and orthographic with HTML buttons.

2. This code is written to be clear-to-understand. It is not necessarily written in the most efficient style using the very best practices.

3. In computer graphics, vertex properties come in two flavors: (1) those that are the same for that vertex no matter what face that vertex is on, and (2) those that are different for that vertex depending on what face the vertex is on. This program demonstrates both. The colors are the first flavor. The texture coordinates are the second.

The Vertex Shader Source Code in the HTML File

sample.html

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

uniform mat4    uModelViewMatrix;
uniform mat4    uProjectionMatrix;

attribute vec3  aVertex;
attribute vec3  aColor;
attribute vec2  aTexCoord0;

varying vec3    vColor;
varying vec2    vST;
varying float   vZ;

void
main( )
{
        vColor = aColor;
        vST    = aTexCoord0;
        vZ     = aVertex.z;
        gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aVertex,1.);
}
</script>
```

Announces that this is a vertex shader

The name of the vertex shader

Uniform variables

Per-vertex attribute variables

Vertex shader-created output variables to be interpolated through the rasterizer

Set varying variables and transform the vertex

The Fragment Shader Source Code in the HTML File

sample.html

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

precision highp float;


uniform sampler2D      uTexture;

varying vec3    vColor;
varying vec2    vST;
varying float   vZ;

void
main( )
{
      if( vZ <= 0.99 )
      {
            gl_FragColor = vec4( vColor, 1. );
      }
      else
      {
            vec4 rgba = texture2D( uTexture, vST );
            gl_FragColor = vec4( rgba.rgb, 1. );
      }
}
</script>
```

Announces that this is a fragment shader

The name of the fragment shader

Must set the precision in OpenGL-ES

Uniform variable

Fragment shader input variables that were interpolated through the rasterizer

Used to identify which face to texture

Decide to use the color or the texture, based on the Z model coordinate

The General Format of the HTML File

sample.html

```
<link rel="stylesheet" href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css">
<style>#slider { margin: 10px; }  </style>
<script src="http://code.jquery.com/jquery-1.8.3.js"></script>
<script src="http://code.jquery.com/ui/1.9.2/jquery-ui.js"></script>
```

Get the CSS style sheet and get the jQuery user interface JavaScript code

The General Format of the HTML File

sample.html

```
<button id = "PerspButton">Perspective</button>
<button id = "OrthoButton">Orthographic</button>
<p></p>

<div id="slider">
<script>
$( "#slider" ).slider( );
$( "#slider" ).slider( "option", "min",   0.1  );
$( "#slider" ).slider( "option", "max",   2.0  );
$( "#slider" ).slider( "option", "value", 1.0  );
$( "#slider" ).slider( "option", "step",  0.01 );
$( "#slider" ).slider( "option", "orientation", "horizontal" );
$( "#slider" ).slider( "enable" );
</script>
</div>

<canvas id="gl-canvas" width="512"" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>

<script type="text/javascript" src="http://cs.oregonstate.edu/~mjb/WebGL/Webgl-Utils.js"></script>
<script type="text/javascript" src="http://cs.oregonstate.edu/~mjb/WebGL/InitShaders.js"></script>
<script type="text/javascript" src="http://cs.oregonstate.edu/~mjb/WebGL/GlMatrix.js"></script>
<script type="text/javascript" src="sampledata.js"></script>
<script type="text/javascript" src="sample.js"></script>
```

Setup the buttons

Setup the jQuery slider parameters

Bring in all of the support code and the main program

What is in Webgl-Utils.js?


```
<script type="text/javascript" src="http://cs.oregonstate.edu/~mjb/WebGL/Webgl-Utils.js"></script>
```

Webgl-Utils.js is a Google-supplied set of Javascript to setup the WebGL window, canvas, and context.

What is in InitShaders.js?


```
<script type="text/javascript" src="http://cs.oregonstate.edu/~mjb/WebGL/InitShaders.js"></script>
```

InitShaders.js contains the calls to **gl.createShader, gl.shaderSource, gl.compileShader, gl.createProgram, gl.attachShader,** and **gl.linkProgram** to create the shader program from the vertex and fragment shader source.

The logic is exactly the same as it is in C, but written in Javascript.

What is in GlMatrix.js?


```
<script type="text/javascript" src="http://cs.oregonstate.edu/~mjb/WebGL/GlMatrix.js"></script>
```

GlMatrix.js came from Brandon Jones and Ed Angel. It contains vec2, vec3, vec4, mat2, mat3, and mat4 data types along with methods to create transformation matrices to pass into vertex shaders as attribute variables.

Basically, it acts as a Javascript GLM.

Take a look through it sometime. It is very readable.

The General Format of the JavaScript File


```
var canvas;
var gl;

var Program;

var Vertices;
var Color;

var NumPoints;
var VertexArray;
var ColorArray;
var TexArray;

var MouseDown = false;
var LastMouseX;
var LastMouseY;
var Left, Middle, Right;
var Perspective;
var SaveScale = 1.;

var MvMatrix = mat4.create( );
var PMatrix  = mat4.create( );
var MvLoc;
var PLoc;
var TLoc;
var SampleTexture;
var ModelMatrix = mat4.create( );

var ST00, ST01, ST10, ST11;

window.onload = InitGraphics;   // function to call first
```

sample.js

Compiled and linked shader program

OpenGL arrays for vertices, colors, and texture coordinates

Mouse and transformation information

Matrix and texture information

Texture coordinates

Function to call first

The General Format of the JavaScript File

sample.js

```
function DrawTriangle( i, a, b, c, sta, stb, stc )
{
	VertexArray[i+0] = Vertices[a];
	ColorArray[i+0]  = Colors[a];
	TexArray[i+0]    = sta;

	VertexArray[i+1] = Vertices[b];
	ColorArray[i+1]  = Colors[b];
	TexArray[i+1]    = stb;

	VertexArray[i+2] = Vertices[c];
	ColorArray[i+2]  = Colors[c];
	TexArray[i+2]    = stc;

	return i+3;
}


function DrawQuad( i, a, b, c, d )
{
	i = DrawTriangle( i, a, b, c,  ST00, ST10, ST11 );
	i = DrawTriangle( i, a, c, d,  ST00, ST11, ST01 );
	return i ;
}
```

DrawQuad( ) calls this twice to draw two triangles per quad

Call this to draw a quadrilateral

The General Format of the JavaScript File

sample.js

```
function InitGraphics( )
{
	canvas = document.getElementById( "gl-canvas" );

	gl = WebGLUtils.setupWebGL( canvas );
	if( ! gl )
	{
		alert( "WebGL isn't available" );
	}

	canvas.onmousedown	= HandleMouseDown;
	document.onmouseup	= HandleMouseUp;
	document.onmousemove = HandleMouseMove;

	// set some handy constants for later:

	ST00 = vec2.create(  [ 0., 0. ]  );
	ST01 = vec2.create(  [ 0., 1. ]  );
	ST10 = vec2.create(  [ 1., 0. ]  );
	ST11 = vec2.create(  [ 1., 1. ]  );

	// set globals:

	Perspective = true;
	mat4.identity( ModelMatrix );
```

InitGraphics( ) sets everything up

Quadrilateral texture coordinates

The General Format of the JavaScript File

sample.js

```
//  load shaders:

        Program = InitShaders( gl, "vertex-shader", "fragment-shader" );
        gl.useProgram( Program );

        MvLoc = gl.getUniformLocation( Program, "uModelViewMatrix" );
        CheckError( “MvLoc " );
        PLoc  = gl.getUniformLocation( Program, "uProjectionMatrix" );
        CheckError( "PLoc " );
        TLoc  = gl.getUniformLocation( Program, "uTexture" );
        CheckError( "TLoc " );

        // setup the texture:

        SampleTexture = gl.createTexture( );
        SampleTexture.image = new Image( );
        SampleTexture.image.onload = function( )
        {
                HandleLoadedTexture( SampleTexture );
        }

        SampleTexture.image.src = "http://cs.oregonstate.edu/~mjb/webgl/beaver.bmp";
        CheckError( "Texture src " );

        // setup ui:

        var b1 = document.getElementById( "PerspButton" );
        b1.addEventListener( "click", function( ) { Perspective =  true; Display( ); }, false );

        b2 = document.getElementById( "OrthoButton" )
        b2.addEventListener( "click", function( ) { Perspective = false; Display( ); }, false );
```

Load, compile, and link the shaders

Obtain the symbol table location of the uniform variables

Setup the texture

Here's where the texture can be found – setting this causes the texture to be loaded

What to do when a button is pressed

The General Format of the JavaScript File

sample.js

```
// initialize the data:

InitData( );

// put the data in opengl buffers:

var vertexBufferId = gl.createBuffer( );
gl.bindBuffer( gl.ARRAY_BUFFER, vertexBufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(VertexArray), gl.STATIC_DRAW );
var vLoc = gl.getAttribLocation( Program, "aVertex" );
gl.vertexAttribPointer( vLoc, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vLoc );

var colorBufferId = gl.createBuffer( );
gl.bindBuffer( gl.ARRAY_BUFFER, colorBufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(ColorArray), gl.STATIC_DRAW );
var cLoc = gl.getAttribLocation( Program, "aColor" );
gl.vertexAttribPointer( cLoc, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( cLoc );

var texBufferId = gl.createBuffer( );
gl.bindBuffer( gl.ARRAY_BUFFER, texBufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(TexArray), gl.STATIC_DRAW );
var tcLoc = gl.getAttribLocation( Program, "aTexCoord0" );
gl.vertexAttribPointer( tcLoc, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( tcLoc );

// get everything running:

Animate( );
}
```

Fill the arrays with data

Setup the vertex buffer to map to the vertex shader attribute variable "aVertex"

Setup the color buffer to map to the vertex shader attribute variable "aColor"

Setup the texture coordinate buffer to map to the vertex shader attribute variable "aTexCoord0"

Start the display

The General Format of the JavaScript File

sample.js

```
function Animate( )
{
	requestAnimFrame( Animate );
	Display( );
}


function Display( )
{
	gl.clearColor( 0.0, 0.0, 0.0, 1.0 );
	gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT );
	gl.viewport( 0, 0, canvas.width, canvas.height );
	gl.enable( gl.DEPTH_TEST );

	// projection matrix:

	if( Perspective )
	{
		PMatrix = mat4.perspective( 60., 1., 0.1, 100.0 );
	}
	else
	{
		PMatrix = mat4.ortho( -2., 2.,  -2., 2.,  0.1, 100. );
	}
```

Ask for the next display and then render the scene

Typical OpenGL start-up

Use the correct projection matrix

The General Format of the JavaScript File

sample.js

```
// read the scaling slider:

        var s = $( "#slider" ).slider( "value" );
        if( s != SaveScale )
        {
                var newScaleMatrix = mat4.create( );
                mat4.identity( newScaleMatrix );
                var s2 = s / SaveScale;
                mat4.scale( newScaleMatrix, [ s2, s2, s2 ] );
                mat4.multiply( newScaleMatrix, ModelMatrix, ModelMatrix );
                SaveScale = s;
        }

        // modelview and projection matrices:

        gl.useProgram( Program );
        mat4.identity( MvMatrix );
        mat4.translate( MvMatrix, [0, 0, -4] );        // viewing
        mat4.multiply( MvMatrix, ModelMatrix );   // modeling
        gl.uniformMatrix4fv( MvLoc, false, MvMatrix );
        gl.uniformMatrix4fv( PLoc,  false, PMatrix );

        // texture sampler:

        gl.activeTexture( gl.TEXTURE6 );
        gl.bindTexture( gl.TEXTURE_2D, SampleTexture );
        gl.uniform1i( TLoc, 6 );

        // do the drawing:

        gl.drawArrays( gl.TRIANGLES, 0, NumPoints );
}
```

Read the slider; process the value if the scale factor has changed

Process and load the ModelView and Projection matrices

Tell the shader where to find the texture sampler

Draw the scene in the buffers

The General Format of the JavaScript File

sample.js

```
function HandleLoadedTexture( texture )
{
	gl.bindTexture( gl.TEXTURE_2D, texture );
	gl.pixelStorei( gl.UNPACK_FLIP_Y_WEBGL, true );
	gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, texture.image );
	gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
	gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR );
	gl.generateMipmap( gl.TEXTURE_2D );
	gl.bindTexture( gl.TEXTURE_2D, null );
	CheckError( "Loading texture " );
}


function HandleMouseDown( event )
{
	MouseDown = true;
	LastMouseX = event.clientX;
	LastMouseY = event.clientY;
	WhichButton( event );
}

function HandleMouseUp( event )
{
	MouseDown = false;
}
```

What to do with the texture after it's been loaded (usual OpenGL procedure)

Handle mouse events

The General Format of the JavaScript File

sample.js

```
function HandleMouseMove( event )
{
	if( ! MouseDown )
	{
		return;
	}
	var newX = event.clientX;
	var newY = event.clientY;

	var deltaX = newX - LastMouseX;
	var deltaY = newY - LastMouseY;

	if( Left )
	{
		var newModelMatrix = mat4.create( );
		mat4.identity( newModelMatrix );
		mat4.rotate( newModelMatrix, degToRad(deltaX / 2.), [0, 1, 0] );
		mat4.rotate( newModelMatrix, degToRad(deltaY / 2.), [1, 0, 0] );

		mat4.multiply( newModelMatrix, ModelMatrix, ModelMatrix );
	}

	LastMouseX = newX;
	LastMouseY = newY;
}
```

Handle mouse events

Here's where a mouse movement gets turned into a rotation matrix . . .

. . . and gets multiplied into the running ModelMatrix

The General Format of the JavaScript File

sample.js

```
function WhichButton( event )
{
        var b = event.button;

        Left    = ( b == 0 );
        Middle = ( b == 1 );
        Right   = ( b == 2 );
};
```

Determine which mouse button was hit

Initializing the Data

sampledata.js

```
function InitData( )
{
        // define the data:

        Vertices = new Array(8);
        Colors   = new Array(8);

        Vertices[0] = point3.create(  [ -1., -1., -1. ]  );
        Vertices[1] = point3.create(  [  1., -1., -1. ]  );
        Vertices[2] = point3.create(  [ -1.,  1., -1. ]  );
        Vertices[3] = point3.create(  [  1.,  1., -1. ]  );
        Vertices[4] = point3.create(  [ -1., -1.,  1. ]  );
        Vertices[5] = point3.create(  [  1., -1.,  1. ]  );
        Vertices[6] = point3.create(  [ -1.,  1.,  1. ]  );
        Vertices[7] = point3.create(  [  1.,  1.,  1. ]  );

        Colors[0] = color3.create(  [ 0., 0., 0. ]  );
        Colors[1] = color3.create(  [ 1., 0., 0. ]  );
        Colors[2] = color3.create(  [ 0., 1., 0. ]  );
        Colors[3] = color3.create(  [ 1., 1., 0. ]  );
        Colors[4] = color3.create(  [ 0., 0., 1. ]  );
        Colors[5] = color3.create(  [ 1., 0., 1. ]  );
        Colors[6] = color3.create(  [ 0., 1., 1. ]  );
        Colors[7] = color3.create(  [ 1., 1., 1. ]  );
```

Generate the data arrays.
(I like to put this in its own file so that I can generate this with a C/C++ program if I want)

Cube Vertices =

```
{ -1., -1., -1. },
{  1., -1., -1. },
{ -1.,  1., -1. },
{  1.,  1., -1. },
{ -1., -1.,  1. },
{  1., -1.,  1. },
{ -1.,  1.,  1. },
{  1.,  1.,  1. }
```

Cube Colors =

```
{ 0., 0., 0. },
{ 1., 0., 0. },
{ 0., 1., 0. },
{ 1., 1., 0. },
{ 0., 0., 1. },
{ 1., 0., 1. },
{ 0., 1., 1. },
{ 1., 1., 1. }
```

Initializing the Data

sampledata.js

```
        NumPoints = 6 * 2 * 3;          // sides * triangles/side * vertices/triangle
        VertexArray = new Array( NumPoints );
        ColorArray  = new Array( NumPoints );
        TexArray    = new Array( NumPoints );

        var index = 0;
        index = DrawQuad( index, 1, 0, 2, 3 );
        index = DrawQuad( index, 4, 5, 7, 6 );
        index = DrawQuad( index, 5, 1, 3, 7 );
        index = DrawQuad( index, 0, 4, 6, 2 );
        index = DrawQuad( index, 6, 7, 3, 2 );
        index = DrawQuad( index, 0, 1, 5, 4 );
}
```

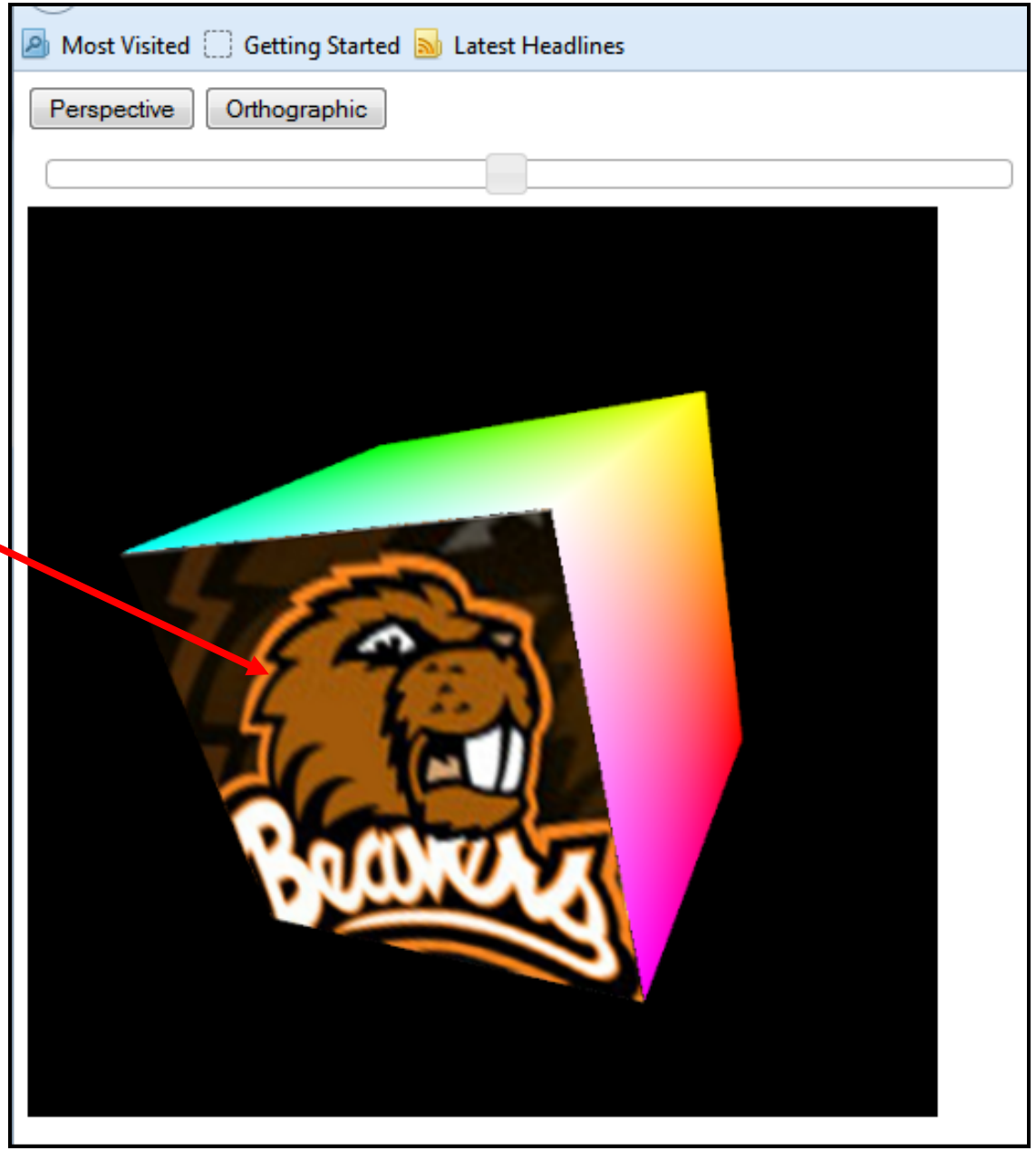
Cube Indices =

{ 1, 0, 2, 3 },
{ 4, 5, 7, 6 },
{ 5, 1, 3, 7 },
{ 0, 4, 6, 2 },
{ 6, 7, 3, 2 },
{ 0, 1, 5, 4 }

Oregon State University
Computer Graphics

The Results

The face that gets textured because its at Z = +1.0



References

The Khronos Group's WebGL Page:
http://khronos.org/webgl

Khronos Group's WebGL Quick Reference Card:
https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

Kouichi Matsuda and Rodger Lea, WebGL Programming Guide, Addison-Wesley, 2013.

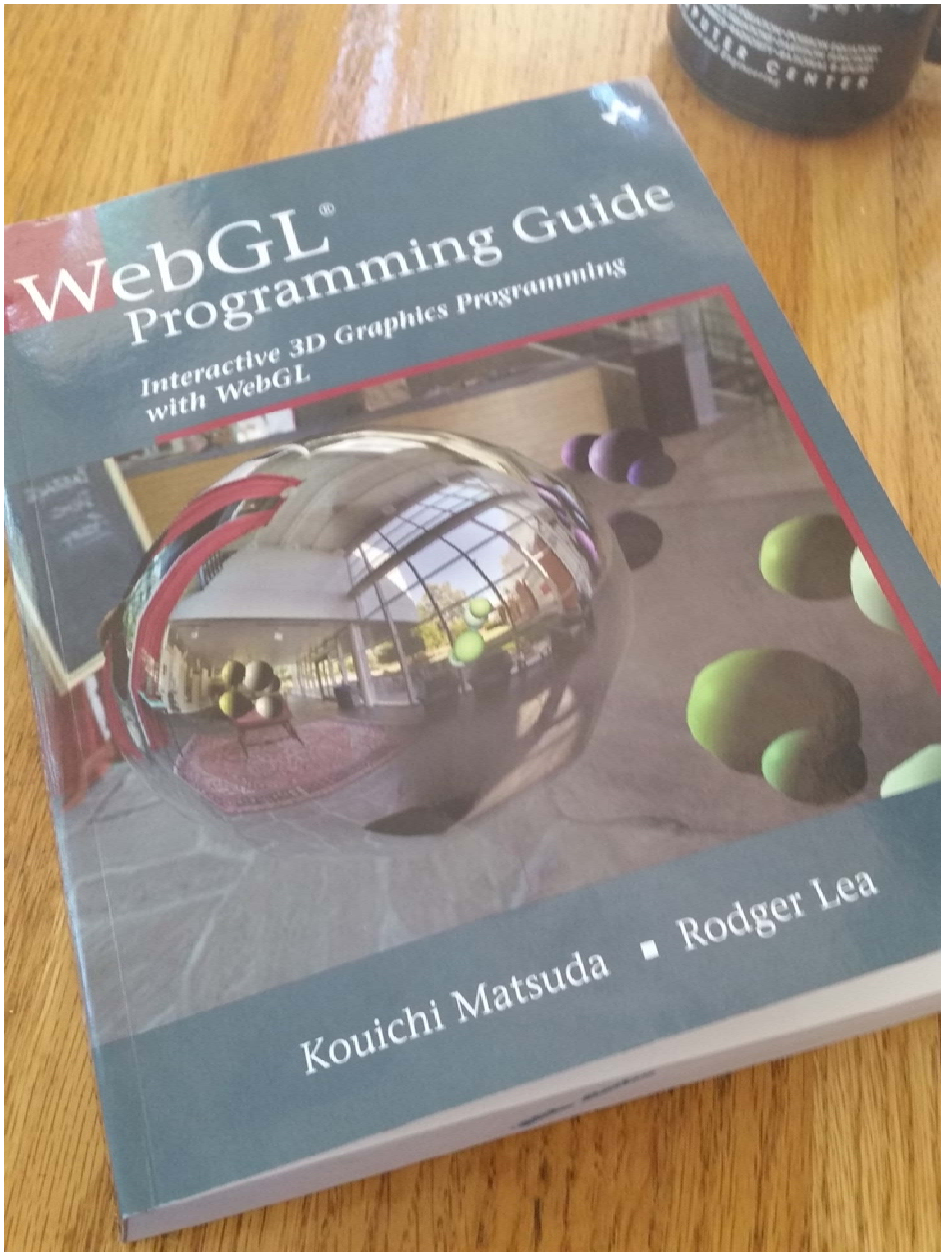
Tony Parisi, *WebGL: Up and Running*, O'Reilly, 2013.

OSU WebGL site:
http://cs.oregonstate.edu/~mjb/webgl

A set of WebGL tutorials:
http://learningwebgl.com

The jQuery user interface site:
http://api.jqueryui.com/

Ed Angel's WebGL site for his book examples:
http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/CODE/WebGL/


WebGL®
Programming Guide
Interactive 3D Graphics Programming with WebGL
Kouichi Matsuda ▪ Rodger Lea