
# Workshop (London)

Back to Basics

Thursday, Jul 31st - 6:30 PM

SkillsMatter, 116-120 Goswell Road, London

# Carl Bateman

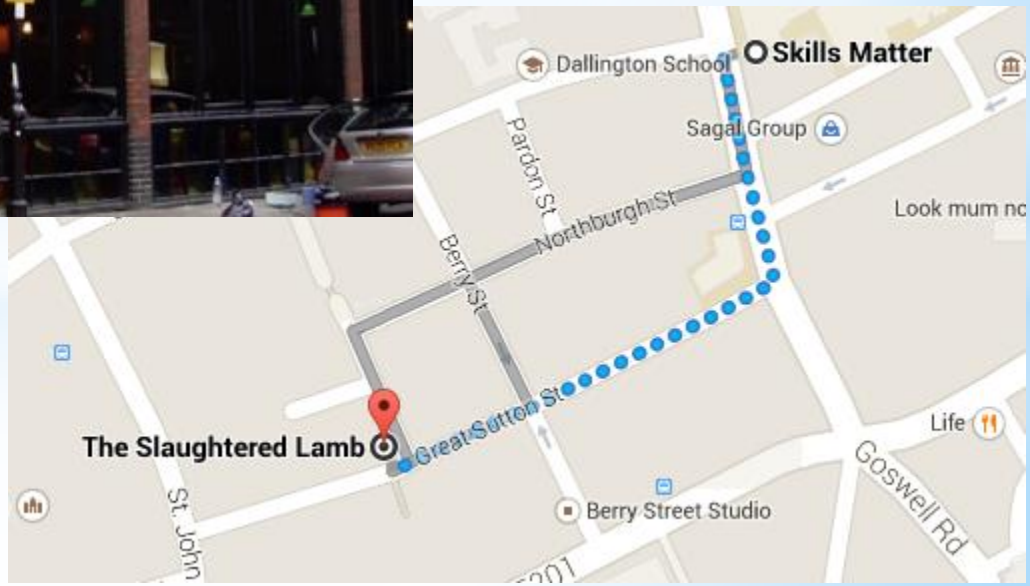
- By day --
  - Software Engineer (desktop)
  - C#, C++, VB, MySQL, .NET, Linq, blah, blah, blah
- By night --
  - Software Engineer (desktop)
  - OpenGL, Unity, JavaScript, PHP, CSS, HTML and, of course, WebGL
- Love 3d and game jams

# Thanks to…

- SkillsMatter (venue)
  - With whom I am in no way affiliated
- Pearson
  - eBook 45% discount

# WebGL Workshop

- After Workshop Drinkies @ The Slaughtered Lamb

# WebGL Workshop

- Not a lecture
- A resumé of my WebGL experience
- Question? Ask!
- A few books – many, many, many web pages
- Frankencode
- A lot to cover

# Who knows...?

- OpenGL
- Shaders
- C or C++
- Web
- Desktop
- 3D Maths (vectors, matrices, quaternions)

# Back to Basics

- Slides and stuff at

webglworkshop.com/08

# Back to Basics

- OpenGL ES 2.0 for the web
- Access to GPU via shaders

- Any browser

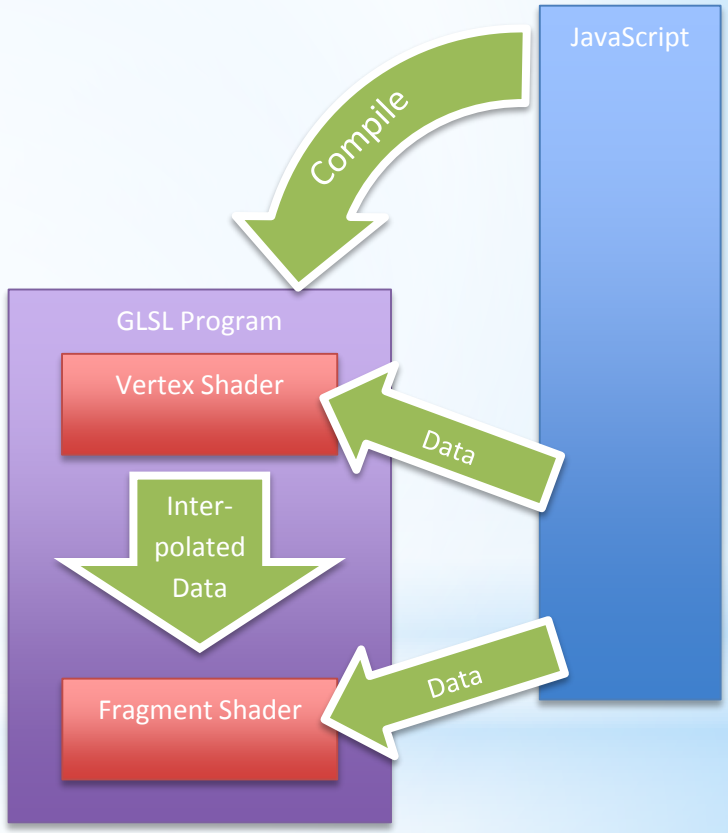
- Check support
  - get.webgl.org
  - caniuse.com/webgl
  - browserleaks.com/webgl
  - webglreport.com
  - doesmybrowsersupportwebgl.com

# Back to Basics

- OpenGL
  - Fast language
  - Fast GPU
- OpenGL ES 2.0
  - Fast language
  - Slow GPU
- WebGL
  - Slow language
  - Fast GPU (desktop) – offload to GPU
  - Slow GPU (mobile) - ??? – kill battery

# Back to Basics

- JavaScript
  - Not typed
- GLSL
  - Strongly typed



# HTML Template - Self-explanatory?

```
<html>
  <title> </title>
  <head>
```

```
  <style>
    body {background-color:#b0c4de;}
    canvas {background-color:#c4deb0;}
  </style>
```

Simple styling to differentiate body and canvas

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

Fragment and vertex shader

```
  <script type="text/javascript">
    function init() {  }
  </script>
```

JavaScript code

```
  </head>
  <body onload="init()">
```

```
  <canvas id="glCanvas" width="500" height="500">
```

<canvas> element to hold WebGL context

```
 </body>
</html>
```

# Hello Triangle 1 - Shaders

```
<script id="vertex" type="x-shader">
  attribute vec2 aVertexPosition;

  void main() {
    gl_Position = vec4(aVertexPosition, 0.0, 1.0);
  }
</script>
```

Vertex position

```
<script id="fragment" type="x-shader">
  precision highp float;
  uniform vec4 uColor;

  void main() {
    gl_FragColor = uColor;
  }
</script>
```

Fragment (pixel) colour

# GLSL

- C like, GPU specific
- Vectors, matrices – native type
- Swizzles (.xyz, .zzz, .rgb, .brg, ~~.xbz~~)

- Functions
- Dot, cross, length, normalize, mix, reflect, etc.

# Hello Triangle 2 – get and clear context

```
var shaderProgram;
var cubeVertexPositionBuffer;
```

Global variables

```
function initWebGL() {
  canvas = document.getElementById(“myCanvas");
```

Get glCanvas element

```
  var names = ["webgl", "experimental-webgl",
"webkit-3d", "moz-webgl"];
  for (var i = 0; i < names.length; ++i)  {
    try {
      gl = canvas.getContext(names[i]);
    }
    catch (e) { }
    if (gl) break;
  }
```

Context name can differ depending on browser

Store possible context names in array then try each

Can become very complicated

Context covers entire canvas

WebGL methods / functions, constants, etc. accessed through context i.e.“`gl.`” (by convention)

```
  gl.viewportWidth = canvas.width;
  gl.viewportHeight = canvas.height;
```

Save viewport dimensions

# Hello Triangle 3 – build shaders

```
var v = document.getElementById("vertex").
firstChild.nodeValue;
var f = document.getElementById("fragment").
firstChild.nodeValue;
```

```
var vs = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vs, v);
gl.compileShader(vs);
```

```
var fs = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fs, f);
gl.compileShader(fs);
```

```
program = gl.createProgram();
gl.attachShader(program, vs);
gl.attachShader(program, fs);
gl.linkProgram(program);
```

# Hello Triangle 4 – check shaders

```
if (!gl.getShaderParameter(vs, gl.COMPILE_STATUS))
  console.log(gl.getShaderInfoLog(vs));
```

Check status

```
if (!gl.getShaderParameter(fs, gl.COMPILE_STATUS))
  console.log(gl.getShaderInfoLog(fs));
```

```
if (!gl.getProgramParameter(program, gl.LINK_STATUS))
  console.log(gl.getProgramInfoLog(program));
```

```
 gl.useProgram(shaderProgram);
```

```
 shaderProgram.uColor =
gl.getUniformLocation(shaderProgram, "uColor");
  gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0,
1.0]);
```

Get position of uniform “uColor”

```
 shaderProgram.aVertexPosition =
gl.getAttribLocation(shaderProgram, "aVertexPosition");
```

Get position of attribute “aVertexPosition”

```
gl.enableVertexAttribArray(shaderProgram.
aVertexPosition);
```

Enable vertex array

# Hello Triangle 5 – define geometry

```
function initGeometry() {
```

```
 var vertices = new Float32Array([-0.5, 0.5, 0.5, -0.5,
-0.5, -0.5]);
```

Define vertices

```
 cubeVertexPositionBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER,
cubeVertexPositionBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices,
gl.STATIC_DRAW);
```

Create buffer and bind data

```
  cubeVertexPositionBuffer.itemSize = 2;
  cubeVertexPositionBuffer.numItems = vertices.length /
cubeVertexPositionBuffer.itemSize;
```

Itemsize: ordinates in vertex

# Hello Triangle 6 – connect to GPU and draw

```
function draw() {
  gl.vertexAttribPointer(shaderProgram.aVertexPosition,
cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0,
0);
```

Set pointer to vertices

```
  gl.clearColor(0, 0.5, 0, 1);
```

Clear background to pale blue

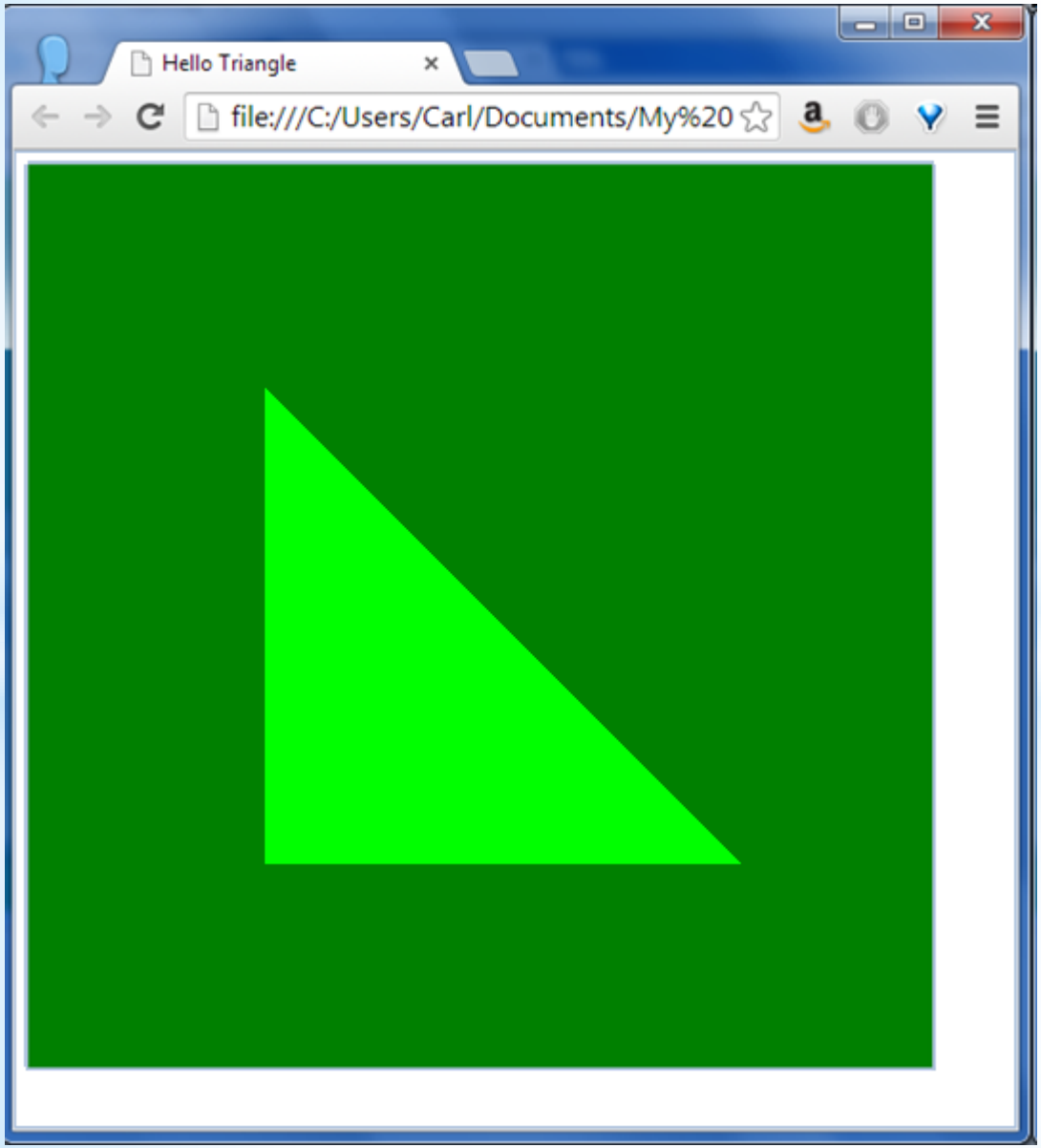
```
  gl.clear(gl.COLOR_BUFFER_BIT);
```

Enable colour clearing

```
  gl.drawArrays(gl.TRIANGLES, 0,
cubeVertexPositionBuffer.numItems);
```

Draw array

# Hello Triangle - result



# Steps

Triangle → 3D Square

3D Square → Coloured Cube

Coloured Cube → Rotated Cube

Rotated Cube → Rotatable Cube

Rotatable Cube → Dice

# Steps

## Triangle → 3D Square

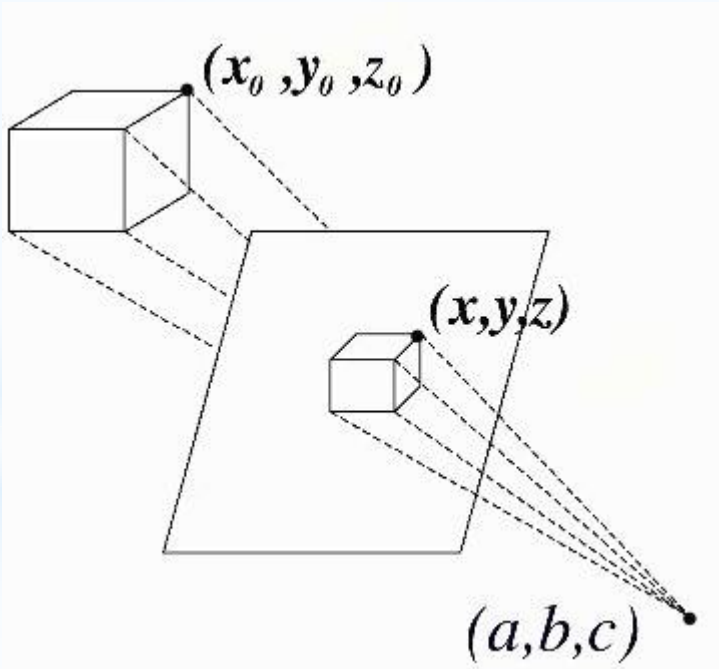
## Note: included libraries

```
m3c[0].x = m3a[0].x * m3b[0].x + m3a[1].x * m3b[0].y
                               + m3a[2].x * m3b[0].z;
m3c[1].x = m3a[0].x * m3b[1].x + m3a[1].x * m3b[1].y
                               + m3a[2].x * m3b[1].z;
m3c[2].x = m3a[0].x * m3b[2].x + m3a[1].x * m3b[2].y
                               + m3a[2].x * m3b[2].z;
m3c[0].y = m3a[0].y * m3b[0].x + m3a[1].y * m3b[0].y
                               + m3a[2].y * m3b[0].z;
m3c[1].y = m3a[0].y * m3b[1].x + m3a[1].y * m3b[1].y
                               + m3a[2].y * m3b[1].z;
m3c[2].y = m3a[0].y * m3b[2].x + m3a[1].y * m3b[2].y
                               + m3a[2].y * m3b[2].z;
m3c[0].z = m3a[0].z * m3b[0].x + m3a[1].z * m3b[0].y
                               + m3a[2].z * m3b[0].z;
m3c[1].z = m3a[0].z * m3b[1].x + m3a[1].z * m3b[1].y
                               + m3a[2].z * m3b[1].z;
m3c[2].z = m3a[0].z * m3b[2].x + m3a[1].z * m3b[2].y
                               + m3a[2].z * m3b[2].z;
```

# Steps

## Triangle → 3D Square

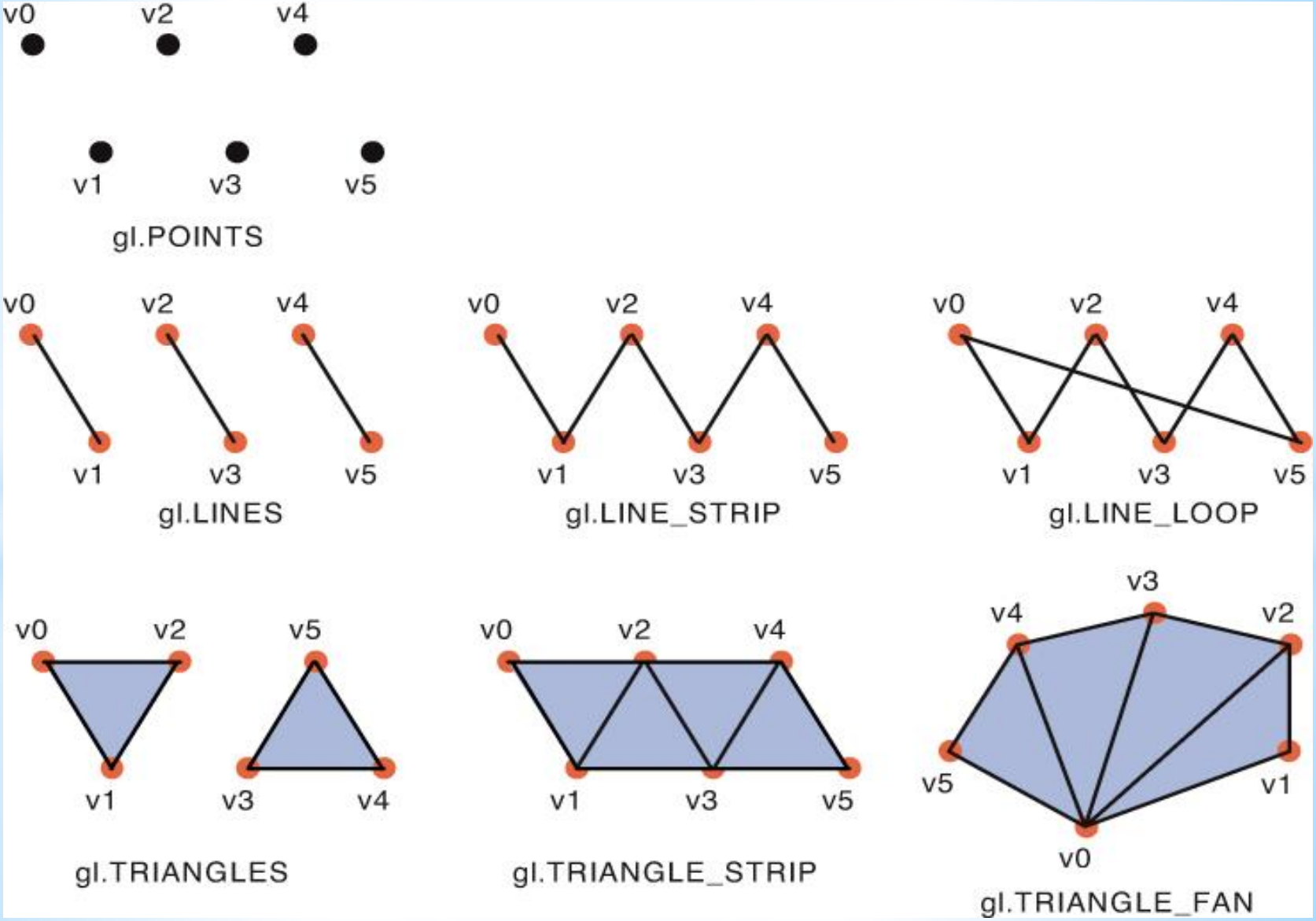
Add perspective projection and model matrices




Change vec2 to vec3

Change assignment to gl_Position

# WebGL Primitives

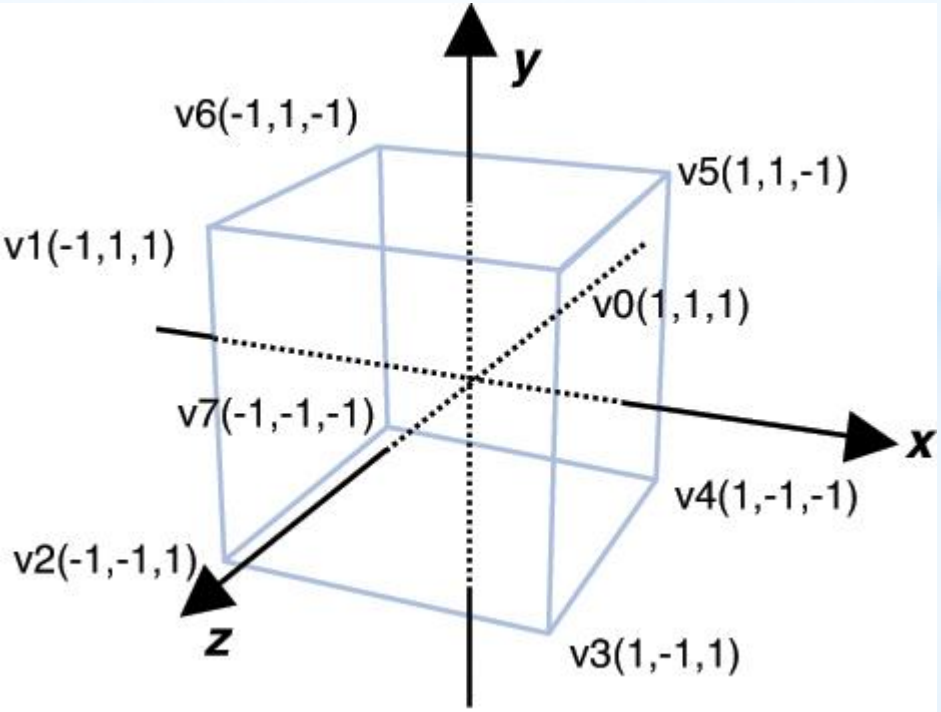


# Steps

3D Square → Coloured Cube (1)

Define vertices

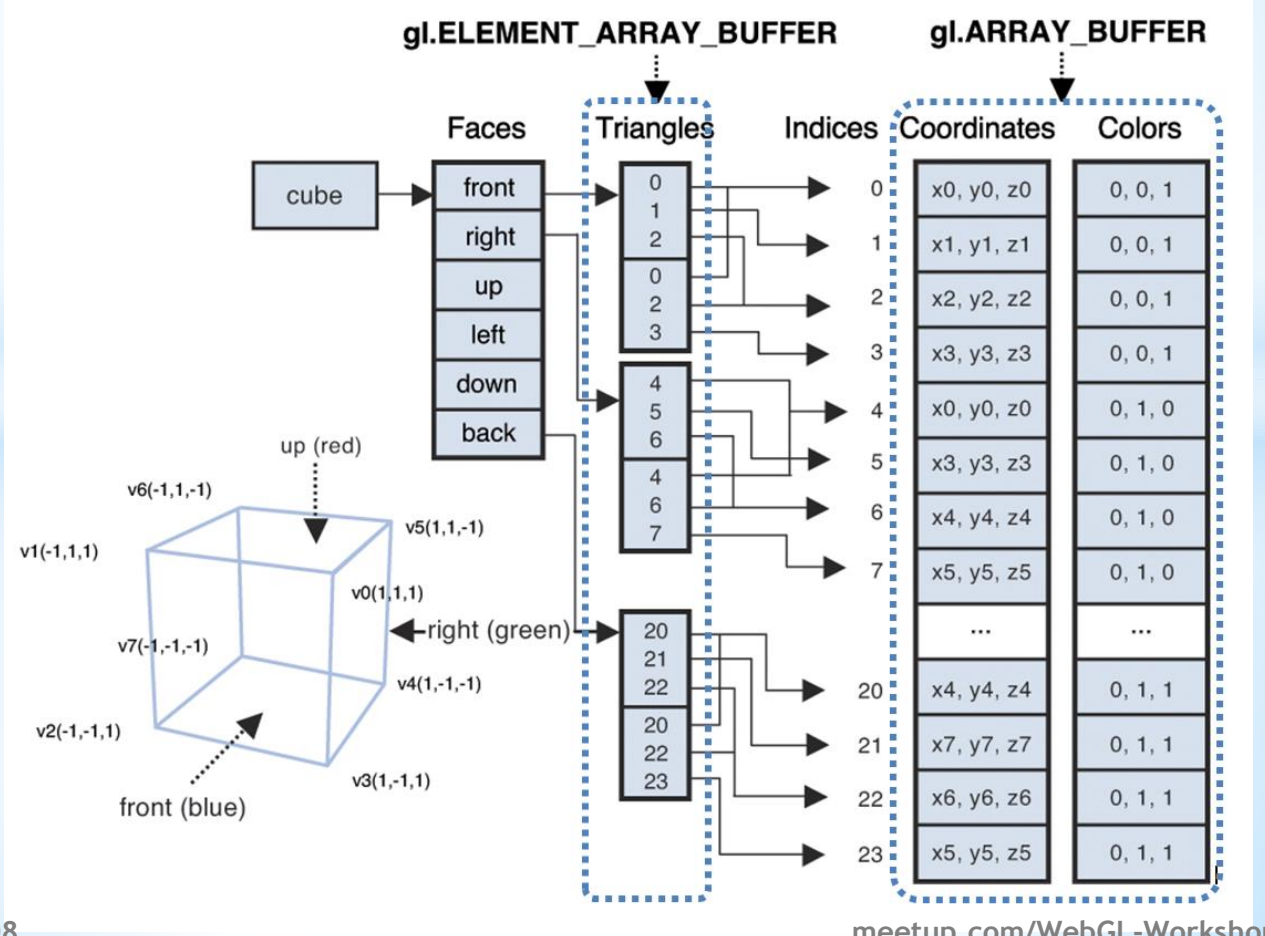
Define face colours (and for each vertex)



# Steps

## 3D Square → Coloured Cube (2)

Define face colours (and for each vertex)

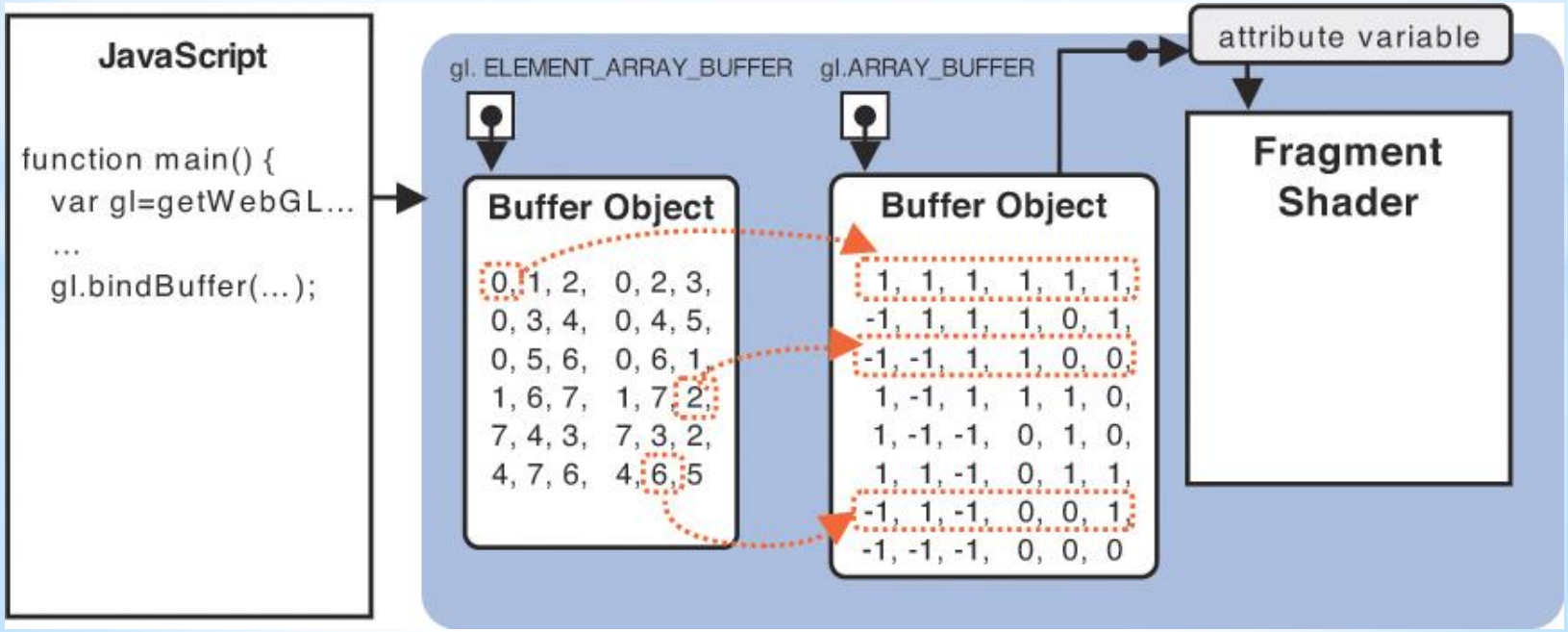


# Steps

## 3D Square → Coloured Cube (3)

Create and bind the buffers and data

Bind buffers and set attribute pointers to draw



# Steps

## Coloured Cube → Rotated Cube

Add a transformation matrix

- Rotation
- Position
- Scale
- Sheer

# Steps

## Rotated Cube → Rotatable Cube

Add mouse event handlers

Track current and previous mouse position while dragging

Difference => velocity

Update cube orientation

If not dragging reduce velocity

# Steps

Rotatable Cube → Dice ???

# Three.js

- Raw WebGL – powerful but verbose

# WebGL Workshop

References:

WebGL Programming Guide

Mozilla Developer Centre

https://developer.mozilla.org/en-US/docs/Web/WebGL

Learning WebGL blog

http://learningwebgl.com/blog/

Get started with WebGL: draw a square

http://www.creativebloq.com/javascript/get-started-webgl-draw-square-7112981

Things I hate about WebGL *(OpenGL ES 2.0 vs WebGL)*

http://www.tamats.com/blog/?p=604

Cheat sheets

# WebGL Workshop

- Help!
- Volunteers
  - Web site
  - Material
  - Presentations
- Sponsors
  - Venues
  - Door prizes
  - Speakers
  - Catering