GLSL

WebGL is a combination of JavaScript and GLSL providing commands that allow access to and control the GPU via shader programs.

What is a Shader?

A program which is compiled and run on the GPU.

Shaders are written in GLSL, are text strings or files, are *compiled* in the browser and sent to the GPU where they are *executed*. The GPU side WebGL code should run as fast as other native code.

Shader programs are massively parallel, working on many *individual* vertices or pixels at once. While data can be passed from the browser and from the vertex shader to the fragment shader, data *cannot* be passed directly between instances of the shader program. A workaround of sorts is possible with textures but the result of one shader program cannot be passed another.

Shader Program

In WebGL† a shader program **must** have a vertex shader and a fragment shader. The vertex shader works on vertex data, while the fragment shader works on pixel data.

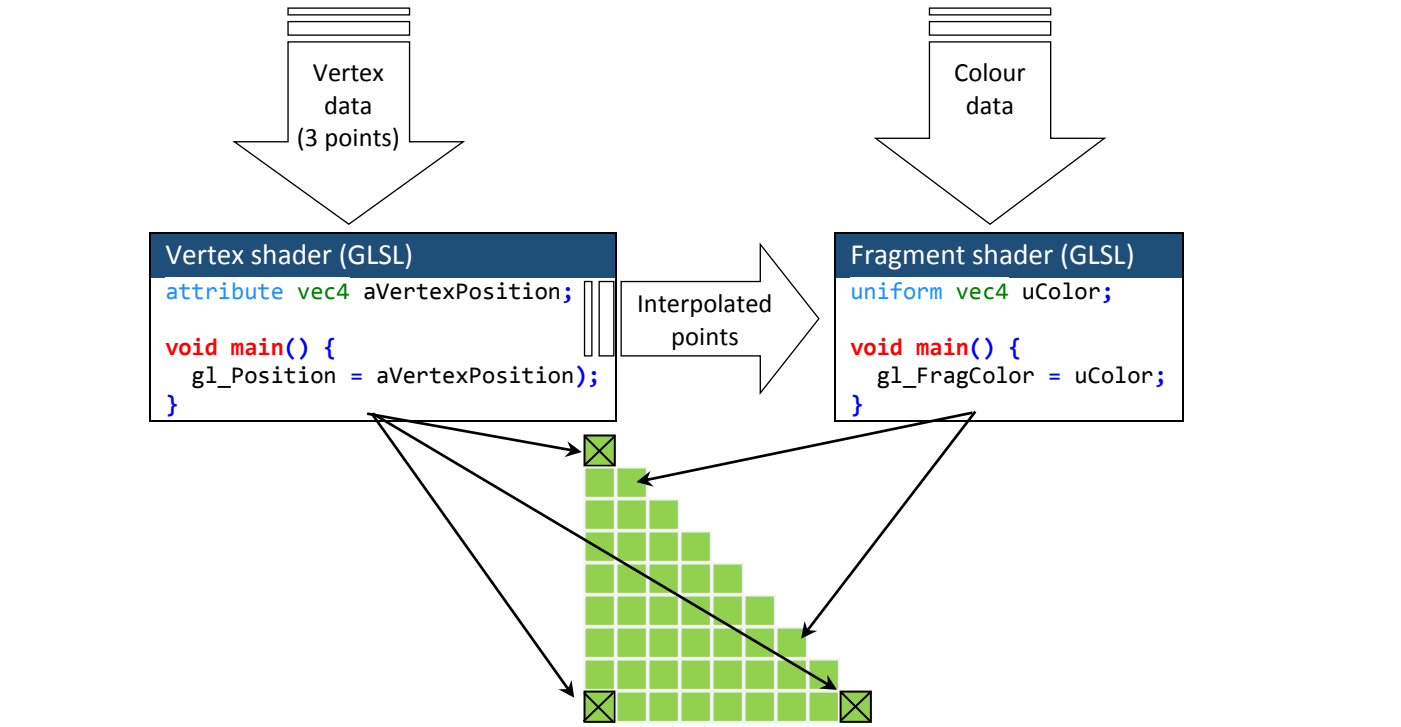
†OpenGL also has geometry and compute shaders.

e.g.

JavaScript

```
aVertexPosition = gl.getAttribLocation(shaderProgram,
"aVertexPosition");
gl.enableVertexAttribArray(aVertexPosition);
...
gl.vertexAttribPointer(itemSize, gl.FLOAT, false, 0, 0);
```

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



gl_Position and gl_FragColor (along with gl_PointSize and gl_FragData[]) are built-in variables and represent the output of the shader (gl_FragCoord, gl_PointCoord, gl_FrontFacing are built-in inputs).

Exercises

We'll be using Three.js as it provides the use of custom shaders and takes care of the heavy lifting. (Babylon.js and Clara.io also provide the use of custom shaders.)

Show clipping volume

Demo vertex displacement cube to sphere & animate

Demo vertex displacement wavy cube & animate

(shadows won't work)

SIMD

Texture (vertex shader, fragment shader)

Transparency, iridescence, multiple materials

Procedural texture, static, animated

Light and anisotropic material - teapot

Normals

Mandelbrot

Image processing

GPGPU, render to texture, multiple passes, post-processing and ping ponging

Babylon.js

Task 1 – (Raw WebGL) Draw a Triangle

Not really, this is just for reference, to show how to set up and use WebGL without any libraries, SDKs, etc..

However, note the shaders at lines 18 and 26.

```
1   <html>
2   <head>
3     <title>Base</title>
4     <style> canvas { width: 100%; height: 100% } </style>
5   </head>
6   <body onload="main()">
7   </body>
8
9   <script id="vertex-shader" type="x-shader/x-vertex">
10    precision highp float;
11    attribute vec3 aVertexPosition;
12    void main() {
13      gl_Position = vec4(aVertexPosition, 1.0);
14    }
15  </script>
16
17  <script id="fragment-shader" type="x-shader/x-fragment">
18    precision highp float;
19    uniform vec4 uColor;
20    void main() {
21      gl_FragColor = uColor;
22    }
23  </script>
24
25  <script>
26    var shaderProgram;
27    var cubeVertexPositionBuffer;
28    var vertices;
29
30    function init() {
31      initWebGL();
32      initShaderProgram();
33      initGeometry();
34      animate();
35    }
36
37    function initWebGL() {
38      canvas = document.getElementById("mycanvas");
39      gl = canvas.getContext("webgl");
40
41      var names = ["webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];
42      for (var i = 0; i < names.length; ++i) {
43        try {
44          gl = canvas.getContext(names[i]);
45        }
46        catch (e) {}
47        if (gl) break;
48      }
49
50      gl.viewportWidth = canvas.width;
51      gl.viewportHeight = canvas.height;
52    }
53
54    function initShaderProgram() {
55      var v = document.getElementById("vertex").firstChild.nodeValue;
56      var f = document.getElementById("fragment").firstChild.nodeValue;
57
58      var vs = gl.createShader(gl.VERTEX_SHADER);
59      gl.shaderSource(vs, v);
60      gl.compileShader(vs);
61
62      var fs = gl.createShader(gl.FRAGMENT_SHADER);
63      gl.shaderSource(fs, f);
64      gl.compileShader(fs);
65
66      shaderProgram = gl.createProgram();
```

```
67        gl.attachShader(shaderProgram, vs);
68        gl.attachShader(shaderProgram, fs);
69        gl.linkProgram(shaderProgram);
70
71        if (!gl.getShaderParameter(vs, gl.COMPILE_STATUS))
72          console.log(gl.getShaderInfoLog(vs));
73
74        if (!gl.getShaderParameter(fs, gl.COMPILE_STATUS))
75          console.log(gl.getShaderInfoLog(fs));
76
77        if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
78          console.log(gl.getProgramInfoLog(shaderProgram));
79
80        gl.useProgram(shaderProgram);
81
82        shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");
83        gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);
84
85        shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
86        gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
87      }
88
89      function initGeometry() {
90        vertices = new Float32Array([-0.5,  0.5, 0.0,
91                                      0.5, -0.5, 0.0,
92                                     -0.5, -0.5, 0.0]);
93
94        cubeVertexPositionBuffer = gl.createBuffer();
95        gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
96        gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
97
98        cubeVertexPositionBuffer.itemSize = 3;
99        cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
100     }
101
102     function draw() {
103       gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
    gl.FLOAT, false, 0, 0);
104
105       gl.clearColor(0, 0.5, 0, 1);
106       gl.clear(gl.COLOR_BUFFER_BIT);
107
108       gl.drawArrays(gl.TRIANGLES, 0, cubeVertexPositionBuffer.numItems);
109     }
110
111     function animate() {
112       requestAnimationFrame(animate);
113       draw();
114     }
115   </script>
116   </html>
```

Vertex Shader

Task 2 – Make a Box in THREE.js (base for rest of exercises)

Again, not really, note the shaders (lines 9 and 18) and the `shaderMaterial` (line 75). The shaders take the same form as those is plain WebGL. This is where we'll be focusing our attention.

Here the colour of the mesh is controlled by the uniform variable uColour (line 77). You might wish to try changing the value.

An orbit control is added just to allow interaction and make things a bit more interesting. Since there is no lighting in the scene (yet), wireframe is used to make things clearer.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4     <title>GLSL - Three.js Cube</title>
5
6     <script src="three.js"></script>
7     <script src="OrbitControls.js"></script>
8
9     <script id="vertexShader" type="x-shader/x-vertex">
10      varying vec2 vUv;
11
12      void main() {
13        vUv = uv;
14        gl_Position = projectionMatrix * modelViewMatrix * vec4( position, 1.0 );
15      }
16    </script>
17
18    <script id="fragmentShader" type="x-shader/x-fragment">
19      varying vec2 vUv;
20      uniform vec4 uColour;
21
22      void main() {
23        gl_FragColor = uColour;
24      }
25    </script>
26
27    <script>
28      window.addEventListener ?
29      window.addEventListener("load", init, false) :
30      window.attachEvent && window.attachEvent("onload", init);
31
32      function init() {
33        var container;
34        var renderer;
35        var camera;
36
37        var scene = new THREE.Scene();
38        var clock = new THREE.Clock();
39        var orbitControl;
40
41        container = document.getElementById("renderTarget");
42        renderer = createRenderer(container);
43        camera = createCamera(container);
44        var shape = addShape(scene);
45
46        orbitControl = new THREE.OrbitControls(camera, renderer.domElement);
47        orbitControl.object.position.z *= 1.25;
48
49        window.addEventListener('resize', function (event) {
50          var w = container.clientWidth;
51          var h = container.clientHeight;
52          renderer.setSize(w, h);
53          camera.aspect = w / h;
54          camera.updateProjectionMatrix();
55        });
56
57        var render = function () {
```

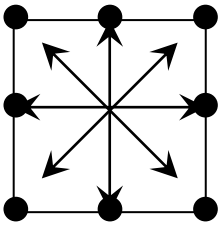
```
58          orbitControl.update(clock.getDelta());
59          renderer.render(scene, camera);
60          requestAnimationFrame(render);
61        };
62
63        render();
64      }
65
66      function addShape(scene) {
67        var geometry = new THREE.BoxGeometry(15, 15, 15, 10, 10, 10);
68        var material = createMaterial();
69        material.wireframe = true;
70        var shape = new THREE.Mesh(geometry, material);
71        scene.add(shape);
72      }
73
74      function createMaterial() {
75        var shaderMaterial = new THREE.ShaderMaterial({
76          uniforms: {
77            uColour: { type: "v4", value: new THREE.Vector4(1, 0, 0, 1) },
78          },
79          //attributes: {},
80          vertexShader: document.getElementById('vertexShader').textContent,
81          fragmentShader: document.getElementById('fragmentShader').textContent
82        });
83
84        return shaderMaterial;
85      }
86
87      function createCamera(container) {
88        var w = container.clientWidth;
89        var h = container.clientHeight;
90        var camera = new THREE.PerspectiveCamera(75, w / h, 0.1, 1000);
91        camera.position.z = 30;
92        return camera;
93      }
94
95      function createRenderer(container) {
96        var renderer = new THREE.WebGLRenderer({ alpha: true });
97        renderer.setPixelRatio(window.devicePixelRatio);
98        container.appendChild(renderer.domElement);
99        renderer.setSize(container.clientWidth, container.clientHeight);
100       return renderer;
101     }
102   </script>
103 </head>
104 <body>
105   <div style="position: absolute; width: 98%; height: 98%;" id="renderTarget"></div>
106 </body>
107 </html>
```

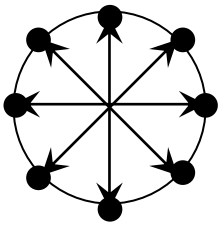
Task 3 – Make the Box Spherical

Static

Now let's make the box spherical.

All points on the surface of a sphere are equidistant from the centre, so we need to move all the boxes' vertices so that they are.





For each point, find its unit vector and multiply by the desired radius.

1. Change the ***vertex shader*** to do the following:
   a. Add a uniform (uRadius) for the radius of the sphere
   b. For a vertex, find its unit position vector (the vector from the centre to the vertex)
   c. Multiply the vector by the radius
2. Add a uniform for the radius (uRadius) to the list of uniforms in createMaterial

Solution file -- 11.Threejs-Spherise.html

Change over time

Let's transform between the cube and sphere over time. The control logic will be split over JavaScript and GLSL.

1. Change the ***vertex shader*** to do the following:
   a. Add a uniform (uLimit)
   b. Add an "if" statement, such that if the vertex position is greater than the radius apply the "sphere" logic as above else leave it unchanged.
2. Change the ***JavaScript*** render function (~line 66) to the following:

```
var radius = 0, min = 10, off = 5;
var render = function () {
  var delta = clock.getDelta();
  orbitControl.update(delta);

  radius += delta / 5;
  radius %= 10;
  material.uniforms.uRadius.value = min + off * Math.sin(2 * Math.PI * radius);

  renderer.render(scene, camera);
  requestAnimationFrame(render);
};
```

Note: using sin guarantees values between 1 and -1.

Rewrite without "if" statement

Warning, Will Robinson! Warning! GLSL uses a SIMD (Single Instruction, Multiple Data) model, a single instruction is run on the data in each thread (not a CPU thread), so when an if statement is encountered if the result can vary between threads BOTH branches are executed (e.g. for a single uniform if(uValue == 3) would only a single branch would be executed, whereas for a varying if(vValue == 3) both branches would be executed). So, here using "if" is bad.

To avoid this the body of the main becomes:

```
void main() {
  vUv = uv;
  float mag = min(length(position), uRadius);
  vec3 t = max(normalize(position) * uRadius, position);
  gl_Position = projectionMatrix * modelViewMatrix * vec4( mag * normalize(position), 1.0 );
}
```

Try changing **min** to **max** for a different effect.

Solution file -- 12.Threejs-Spherise-animated.html

Task 4 – Make the Cube Wobbly

For a slightly different animation, let's make the box wobbly.

Offset all points along one axis by some amount at right angles.

1. Change the ***vertex shader*** to do the following:

```
void main() {
  vUv = uv;
  tPosition = position;
  tPosition.x += cos(uRadius + tPosition.y);
  gl_Position = projectionMatrix * modelViewMatrix * vec4( tPosition, 1.0 );
}
```

2. In the ***JavaScript*** render function change:

```
  material.uniforms.uRadius.value = min + off * Math.sin(2 * Math.PI * radius);
```

   to

```
  material.uniforms.uRadius.value = radius * 5;
```

Solution file -- 13.Threejs-Spherise-wave.html

Task 5 – Display an Image

The sampler2D type allows the use of textures in GLSL. It can be used in vertex and fragment shaders. Here, we'll demo its use in a vertex shader. It's more properly used in fragment shaders but can be used as a method to get data into a vertex shader.

1. In the header include the smiley and dm js files (these hold the base 64 encoded images)

```
<script src="smiley.js"></script>
<script src="dm.js"></script>
```

2. In the ***vertex*** shader:

   a. Add a sampler2d and a varying

   b. Read a colour for the current from the sampler

```
uniform sampler2D uTexture;
varying vec4 vColour;

void main() {
  vColour = texture2D( uTexture, uv );
  gl_Position = projectionMatrix * modelViewMatrix * vec4( position, 1.0 );
}
```

3. In the ***fragment*** shader:

   a. Assign the varying to gl_FragColor

```
varying vec4 vColour;

void main() {
  gl_FragColor = vColour;
}
```

4. In ***JavaScript*** in the addShape function:

   a. Revert the render function to its simpler form

```
var render = function () {
```

```
  orbitControl.update(clock.getDelta());
  requestAnimationFrame(render);
  renderer.render(scene, camera);
};
```

b. Change BoxGeometry to PlaneGeometry

```
function addShape(scene) {
  var geometry = new THREE.PlaneGeometry(15, 15, 10, 10);
```

c. Change the createMaterial function to load and use an image

```
function createMaterial() {
  var image = new Image();
  var texture = new THREE.Texture(image);
  texture.needsUpdate = true;

  var shaderMaterial = new THREE.ShaderMaterial({
    uniforms: {
      uTexture: { type: "t", value: texture }
    },
    //attributes: {},
    vertexShader: document.getElementById('vertexShader').textContent,
    fragmentShader: document.getElementById('fragmentShader').textContent
  });

  image.onload = function () {
    texture.needsUpdate = true;
  };
  image.src = imgDM;// imgSmiley;

  return shaderMaterial;
}
```

The image will initially look like pants. Increasing the height and width segment count for the PlaneGeometry will improve it.

Fragment Shader

Task 6 – Display an Image (properly)

When displaying images, a fragment shader should really be used (the exception is volume rendering or similar). So let's do it properly, by moving the sampler2D to fragment shader.

1. In the ***vertex*** shader:

```
varying vec4 vUv;

void main() {
  vUv = uv;
  gl_Position = projectionMatrix * modelViewMatrix * vec4( position, 1.0 );
}
```

2. In the ***fragment*** shader:

```
varying vec4 vUv;
uniform sampler2D uTexture;

void main() {
  gl_FragColor = texture2D( uTexture, uv );
}
```

3. The vertex density is now irrelevant to image quality (and the wireframe looks the same). So, in ***JavaScript*** in the addShape function, reduce the height and width segments:

```
function addShape(scene) {
  var geometry = new THREE.PlaneGeometry(15, 15, 1, 1);
```

Play around with the colours and transparency, rotate/mirror the image.

Solution file -- 21.Threejs-image-fragment.html

Task 7 – Image Processing

We've done some simple image processing, but using a convolution kernel (just a 3×3 or 5×5 matrix) it's possible to achieve a variety of more sophisticated effects. For instance, to perform edge detection, for each pixel its neighbouring pixels are weighted and summed.

1. In the ***fragment*** shader:

```
varying vec2 vUv;

uniform sampler2D uTexture;
uniform vec2 uTextureSize;
uniform float uKernel[9];

void main() {
  vec2 onePixel = vec2(1.0, 1.0) / uTextureSize;
  vec4 colorSum = texture2D(uTexture, vUv + onePixel * vec2(-1, -1)) * uKernel[0] +
                  texture2D(uTexture, vUv + onePixel * vec2( 0, -1)) * uKernel[1] +
                  texture2D(uTexture, vUv + onePixel * vec2( 1, -1)) * uKernel[2] +
                  texture2D(uTexture, vUv + onePixel * vec2(-1,  0)) * uKernel[3] +
                  texture2D(uTexture, vUv + onePixel * vec2( 0,  0)) * uKernel[4] +
                  texture2D(uTexture, vUv + onePixel * vec2( 1,  0)) * uKernel[5] +
                  texture2D(uTexture, vUv + onePixel * vec2(-1,  1)) * uKernel[6] +
                  texture2D(uTexture, vUv + onePixel * vec2( 0,  1)) * uKernel[7] +
                  texture2D(uTexture, vUv + onePixel * vec2( 1,  1)) * uKernel[8] ;

  float kernelWeight = uKernel[0] +
                       uKernel[1] +
                       uKernel[2] +
                       uKernel[3] +
                       uKernel[4] +
                       uKernel[5] +
                       uKernel[6] +
                       uKernel[7] +
                       uKernel[8] ;

  if (kernelWeight <= 0.0) {
    kernelWeight = 1.0;
  }

  gl_FragColor = vec4((colorSum / kernelWeight).rgb, 1);
}
```

2. In ***JavaScript*** in the createMaterial function:

   a. In the createMaterial function, add uniforms for uTextureSize, uKernel and uTexture to the ShaderMaterial and assign values to them

```
function createMaterial() {
  var image = new Image();
  var texture = new THREE.Texture(image);

  var edgeDetectKernel = [ 1, -1, -1,
                           -1, +8, -1,
                           -1, -1, -1 ];

  var shaderMaterial = new THREE.ShaderMaterial({
    uniforms: {
      uTextureSize: { type: "v2", value: new THREE.Vector2(0,0) },
      uKernel: { type: "fv1", value: edgeDetectKernel },
      uTexture: { type: "t", value: texture },
    },
    //attributes: {},
    vertexShader: document.getElementById('vertexShader').textContent,
    fragmentShader: document.getElementById('fragmentShader').textContent
  });

  image.onload = function () {
    shaderMaterial.uniforms.uTextureSize.value = new THREE.Vector2(image.width, image.height);
    texture.needsUpdate = true;
  };
  image.src = imgDM;// imgSmiley;
```

```
    return shaderMaterial;
}
```

Task 8 – Procedural texturing part 1

Let's get ambitious and replace the plain colours with a procedural texture.

The texture is taken from http://glslsandbox.com/e#7868.1 (formerly http://glsl.heroku.com/e#7868.1)

1. The ***vertex*** shader:

```
void main() {
  gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0 );
}
```

2. The ***fragment*** shader:

```
uniform float time;
uniform float alpha;
uniform vec2 resolution;

#define PI 3.14159
#define TWO_PI (PI*2.0)
#define N 68.5

void main() {
  vec2 center = gl_FragCoord.xy;
  center.x=-10.12*sin(time/200.0);
  center.y=-10.12*cos(time/200.0);

  vec2 v = (gl_FragCoord.xy - resolution/20.0) / min(resolution.y,resolution.x) * 0.1;
  v.x=v.x-10.0;
  v.y=v.y-200.0;
  float col = 0.0;

  for(float i = 0.0; i < N; i++) {
    float a = i * (TWO_PI/N) * 61.95;
    col += cos(TWO_PI*(v.y * cos(a) + v.x * sin(a) + sin(time*0.004)*100.0 ));
  }

  col /= 5.0;

  gl_FragColor = vec4(col*1.0, -col*1.0, -col*4.0, 1.0);
}
```

3. In ***JavaScript*** in function createMaterial()

```
function createMaterial() {
  var shaderMaterial = new THREE.ShaderMaterial({
    uniforms: {
      time: { type: 'f', value: 0.2 },
      scale: { type: 'f', value: 0.2 },
      alpha: { type: 'f', value: 0.6 },
      resolution: { type: "v2", value: new THREE.Vector2(5, 5) }
    },
    //attributes: {},
    vertexShader: document.getElementById('vertexShader').textContent,
    fragmentShader: document.getElementById('fragmentShader').textContent
  });

  return shaderMaterial;
}
```

Task 9 – Procedural texturing part 2

The texture looks “wrong”, it doesn’t change when the cube rotates; this is because the shader uses gl_FragCoord which works in screen space and so never changes for a given pixel. For a better effect we need to use uv co-ordinates for each face.

1. In the ***vertex*** shader:
   a. Add a varying for a texture co-ordinate (i.e. vec2 vUv)
   b. Assign uv to vUv

```
varying vec2 vUv;

void main() {
  vUv = uv;
  gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0 );
}
```

2. In the ***fragment*** shader:

   a. Add a varying for a texture co-ordinate (i.e. vec2 vUv)

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

  varying vec2 vUv;
```

   b. Replace gl_FragCoord.xy with vUv:

      i. Assign (vUv.xy - resolution) / min(resolution.y,resolution.x) * 15.0 to v

```
void main() {
  vec2 center = gl_FragCoord.xy;
  center.x=-10.12*sin(time/200.0);
  center.y=-10.12*cos(time/200.0);

  vec2 v = (vUv.xy - resolution) / min(resolution.y,resolution.x) * 15.0;
```

3. In the ***Javascript*** to animate the texture:

   a. In the render function, update the time value

```
var render = function () {
  orbitControl.update(clock.getDelta());

  material.uniforms.time.value += 0.005;

  requestAnimationFrame(render);
  renderer.render(scene, camera);
};
```

Task 11 – Interacting with Light

This is a bit messy as the last couple of versions of Three.js appears to have broken the ShaderMaterial light interaction and I've had to cobble something together at the last minute.

1. In the ***vertex*** shader:

```
varying vec3 vNormal;
varying vec3 vViewPosition;
varying vec3 vWorldPosition;

void main() {
   gl_Position = projectionMatrix * modelViewMatrix * vec4( position, 1.0 );
   vNormal = normalize( normalMatrix * normal );
   vec4 mvPosition = modelViewMatrix * vec4( position, 1.0 );
   vViewPosition = -mvPosition.xyz;
   vWorldPosition = position;
}
```

2. In the ***fragment*** shader:

```
uniform vec3 uMaterialColor;
uniform vec3 uSpecularColor;

uniform vec3 uDirLightPos;
uniform vec3 uDirLightColor;

uniform vec3 uAmbientLightColor;

uniform float uKd;
uniform float uKs;
uniform float shininess;

uniform float uGroove;

varying vec3 vNormal;
```

```
varying vec3 vViewPosition;
varying vec3 vWorldPosition;

void main() {
   gl_FragColor = vec4( uAmbientLightColor, 1.0 );

   // compute direction to light
   vec4 lDirection = viewMatrix * vec4( uDirLightPos, 0.0 );
   vec3 lVector = normalize( lDirection.xyz );

   vec3 normal = normalize( vNormal );

   for ( int i = 0; i < 4; i++) {
      vec3 offset = vWorldPosition;

    if(i==1)
        offset = -vWorldPosition;
    else if(i==2) {
        //offset.xy = vWorldPosition.yx;
        offset.y = -vWorldPosition.y;
    } else if(i==3) {
        //offset.xy = vWorldPosition.yx;
        offset.x = -vWorldPosition.x;
    }

      offset.y = 0.0;
      vec3 jiggledNormal = normalize( normal + uGroove * normalize( offset ) );

    // diffuse: N * L. Normal must be normalized, since it's interpolated.
    float diffuse = 0.25 * max( dot( jiggledNormal, lVector ), 0.0);

    gl_FragColor.rgb += uKd * uMaterialColor * uDirLightColor * diffuse;

    // specular: N * H to a power. H is light vector + view vector
    vec3 viewPosition = normalize( vViewPosition );
    vec3 pointHalfVector = normalize( lVector + viewPosition );
    float pointDotNormalHalf = max( dot( jiggledNormal, pointHalfVector ), 0.0 );
    float specular = uKs * pow( pointDotNormalHalf, shininess );
    specular *= diffuse*(2.0 + shininess)/8.0;

    // This can give a hard termination to the highlight, but it's better than some weird sparkle.
    // Note: we don't quit here because the solution will use this code twice.
    if (diffuse <= 0.0) {
        specular = 0.0;
    }

    gl_FragColor.rgb += uDirLightColor * uSpecularColor * specular;
  }
}
```

3. In ***JavaScript*** in function createMaterial()

   a. Major changes to handle lights and their properties

```
function createMaterial() {
  var shader = {
    uniforms: {
      "uDirLightPos": { type: "v3", value: new THREE.Vector3() },
      "uDirLightColor": { type: "c", value: new THREE.Color(0xFFFFFF) },
      "uAmbientLightColor": { type: "c", value: new THREE.Color(0x050505) },
      "uMaterialColor": { type: "c", value: new THREE.Color(0xFFFFFF) },
      "uSpecularColor": { type: "c", value: new THREE.Color(0xFFFFFF) },

      uKd: {
        type: "f",
        value: 0.7
      },
      uKs: {
        type: "f",
        value: 0.3
      },
      shininess: {
```

```
        type: "f",
        value: 100.0
      },
      uGroove: {
        type: "f",
        value: 1.0
      }
    }
  };

  var u = THREE.UniformsUtils.clone(shader.uniforms);

  var vs = document.getElementById('vertexShader').text;
  var fs = document.getElementById('fragmentShader').text;

  var material = new THREE.ShaderMaterial({ uniforms: u, vertexShader: vs, fragmentShader: fs });
  return material;
}
```

b. Set the properties

```
var material = addShape(scene);
//material.wireframe = true;
material.uniforms.uDirLightPos.value = light.position;
material.uniforms.uDirLightColor.value = light.color;
material.uniforms.uAmbientLightColor.value = ambientLight.color;

var ambientLight = new THREE.AmbientLight(0x333333);
var light = new THREE.PointLight(0xffff00, 1.0);
light.position.set(100.0, 100.0, 0.1);
scene.add(light);
```

References and resources

Shader based on http://glsl.heroku.com/e#7868.1

Three.js

Resources

Online GLSL editors

Cheat sheets

https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

http://appletree.or.kr/quick_reference_cards/Web_Development/WebGL%20Cheat%20Sheet.pdf (original 404)