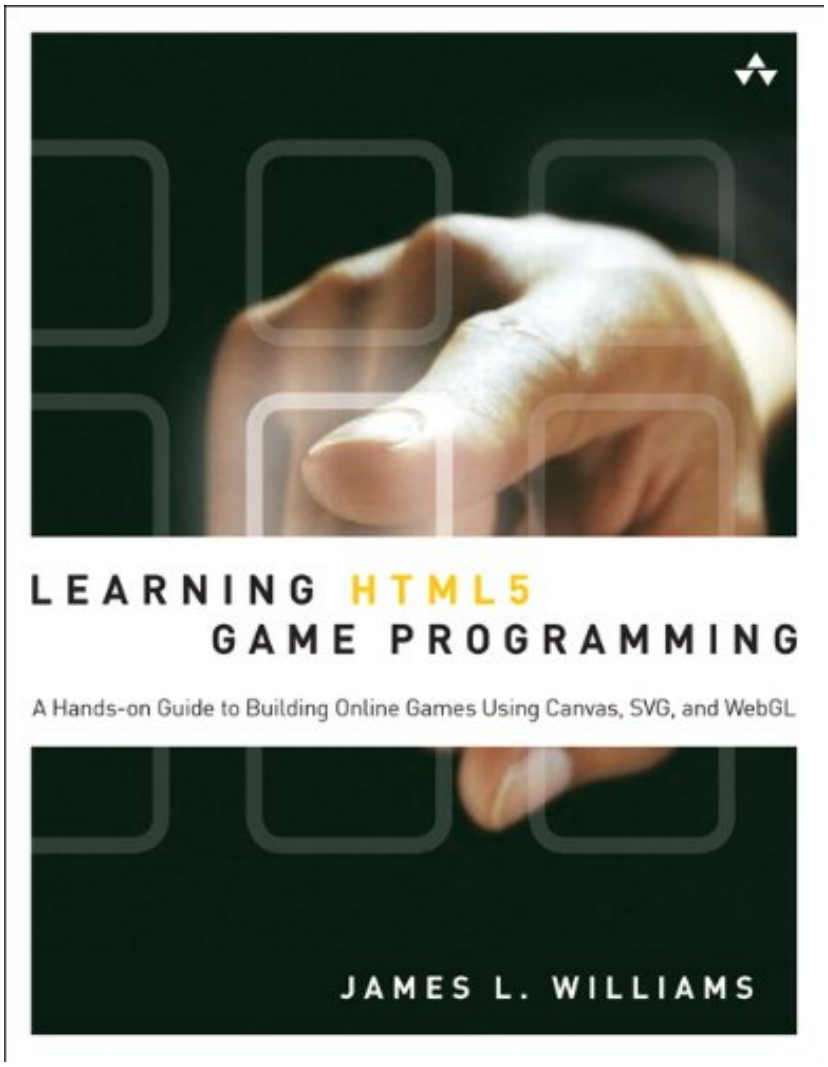
Intro to WebGL/Three.js

About Me

- Author of Learning HTML5 Game Programming
- Blog:



  http://jameswilliams.be/blog
- Twitter: @ecspike
- Google+: http://jameswilliams.be/+

Agenda

- What is WebGL/Three.js?
- Shaders
- Lighting
- Materials
- Creating Meshes
- GLSL
- Exporters
- Animation
- Debugging
- Demos

What is WebGL?

- Hardware accelerated
- Low-level 3D graphics context using the canvas tag
- Supported by most modern browsers
- Syntax is based on OpenGL ES 2.0
- Supports OpenGL Shading language(GLSL)
- Books on OpenGL ES 2.0 can help

Three.js

Three.js

- 3D scenegraph engine
- Abstraction layer over WebGL

  Capable of rendering to
  - Canvas 2D
  - WebGL
  - SVG

  Exporters for popular 3D modeling applications/formats:
  Blender, 3DS Max, OBJ, FBX

  Industry demos:
  3 Dreams of Black
  Ginger Facial Rigging
- Github:https://github.com/mrdoob/three.js

Camera

- Eye Point
- Field of Vision
- Near/Far Planes
- Target(LookAt) Point
- Up Vector

```
camera = new THREE.[Perspective]Camera(FOV,
ASPECT, NEAR, FAR, [target]);
```

Advanced Camera Types

Creating Meshes

- Geometry

  Built-in geometries:

  - Sphere
  - Plane
  - Cylinder
  - Cube
  - Text
  - Torus
  - Path

- Mesh

Creating Meshes - Code

```
geometry = new THREE.Geometry();
geometry.vertices.push(new THREE.Vertex(new
THREE.Vector3(0, 10, 0)));
geometry.vertices.push(new THREE.Vertex(new
THREE.Vector3(-10, -10, 0)));
geometry.vertices.push(new THREE.Vertex(new
THREE.Vector3(10, -10, 0)));
geometry.faces.push(new THREE.Face3(0,1,2));
var triangle = new THREE.Mesh(geometry,
        new THREE.MeshBasicMaterial( { color:
0x00ff00 } )
);

plane = new THREE.Mesh( new THREE.Plane( 200, 200
),
        new THREE.MeshBasicMaterial( { color:
0xe0e0e0 } )
);

plane.overdraw = true;

scene.addObject( plane );
scene.addChild(triangle);
```

Materials, Lighting, and Shaders

Lighting

- Ambient
- Point
- Directional
- SpotLight

```
                          new
THREE.AmbientLight(hexColor);
new THREE.PointLight(hexColor, [intensity],
[distance]);
new THREE.DirectionalLight(hexColor, [intensity],
[distance], [castShadow]);
new THREE.SpotLight(hexColor, [intensity],
[distance], [castShadow]);
```

Shading

- Flat
- Lambertian
- Gouraud
- Phong

Materials

- MeshBasicMaterial
- MeshLambertMaterial
- MeshPhongMaterial
- MeshShaderMaterial

Common Properties

- color
- ambient
- specular
- shininess
- opacity
- mappings: map(texture), envMap
- shading
- wireframe
- blending

GLSL

What is GLSL?

- Targets the GPU and graphics pipeline
- High level language with C-like syntax
- Passed around as strings
- Can be generated and compiled at run-time
- Referred to as programs (the combination of a vertex and fragment shader)

Vertex Shaders

- Run once per vertex in a mesh
- Can alter color, position, or texture coordinates

Example vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    #ifdef GL_ES
    precision highp float;
    #endif

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

Frament(Pixel) Shaders

- Run on every pixel in a mesh
- Can produce effects such as bump mapping and shadowing
- Only knows* about the pixel it is working on

Example fragment shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    #ifdef GL_ES
    precision highp float;
    #endif

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

Shader Demo Code

```
function drawTriangle() {
        var geometry, geoMaterial;

        geoMaterial = new
THREE.MeshLambertMaterial({
                        color:0xFF00FF
        });
        shaderMaterial = new
THREE.MeshShaderMaterial({
                        vertexShader: $('#geom-
vertexShader').get(0).innerHTML,
                        fragmentShader: $('#geom-
fragmentShader').get(0).innerHTML,
                        vertexColors: true
        });

        geometry = new THREE.Geometry();
        geometry.vertices.push(new
THREE.Vertex(new THREE.Vector3(0, 10, 0)));
        geometry.vertices.push(new
THREE.Vertex(new THREE.Vector3(-10, -10, 0)));
        geometry.vertices.push(new
THREE.Vertex(new THREE.Vector3(10, -10, 0)));
        geometry.faces.push(new
THREE.Face3(0,1,2));
        var triangle = new THREE.Mesh(geometry,
shaderMaterial);

        plane = new THREE.Mesh( new THREE.Plane(
200, 200 ), new THREE.MeshBasicMaterial( { color:
0xe0e0e0 } ) );
```

```
        //plane.rotation.x = - 90 * ( Math.PI /
180 );
        plane.overdraw = true;
        scene.addObject( plane );

        scene.addChild(triangle);
}
```

Shader Variable Types

- uniform
- varying
- attribute

- bool
- int
- float
- vec2 / vec3 / vec4
- mat 2 / 3 / 4
- sampler1D / 2D / 3D

Constructing New Shader Types

```
struct MyMaterial {
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    float shininess;
};
```

Communicating with the Shader

```
                        var uniforms;
```

```
                        uniforms = {
                                time: {type:"f",
value:0.0}
                        }
```

```
                shaderMaterial = new
THREE.MeshShaderMaterial({
                        // attributes: attributes,
                        uniforms: uniforms,
                        vertexShader: $('#geom-
vertexShader').get(0).innerHTML,
                        fragmentShader: $('#geom-
fragmentShader').get(0).innerHTML,
        });
```

Custom Shader

```
        #ifdef GL_ES
precision highp float;
#endif

uniform float time;

void main(){
    float r = cos(time);
    float g = sin(time);
    float b = tan(time);

    gl_FragColor = vec4(r, 1.0 - g , b, 1.0);
}
```

Custom Shader

```
#ifdef GL_ES
precision highp float;
#endif

uniform float time;
float r, g, b;

void computeColors() {
        r = cos(time);
        g = sin(time);
        b = tan(time);
}

void main(){

    gl_FragColor = vec4(r, 1.0 - g , b, 1.0);
}
```

Shader Toy Demo

http://www.iquilezles.org/apps/shadertoy/

Texturing/UV Mapping

Texturing Demo

```
function drawScene() {
        var texture =
THREE.ImageUtils.loadTexture("200407-
bluemarble.jpg" );
        var material = new
THREE.MeshBasicMaterial( {
                        color: 0xFFFFFF, ambient:
0xFFFFFF, map:texture
        } );
        sphere = new THREE.Mesh(new
THREE.Sphere(32, 32, 32), material);
        scene.addObject(sphere);
}
```

Creating and Loading Assets

Blender

Free and open source 3D modeling application

Built-in Python API for scripting

Vibrant plugin community

Can import/export from many formats including Three.js

Advanced features

Uses in the Media Industry

http://blender.org

Loading Assets

```
function drawCube() {
    var loader = new THREE.JSONLoader();
    loader.load( {model: "cube.js", callback:
createScene1 });
}

function createScene1(obj) {
    obj.materials[0][0].shading =
THREE.FlatShading;
    mesh = THREE.SceneUtils.addMesh( scene, obj,
250, 400, 0, 0, 0, 0, 0,
        obj.materials[0] );
}
```

Three.js JSON Model Format

```
var model = {

    "version" : 2,
    "scale" : 1.000000,
    "materials": [... ],
    "vertices": [ ... ],
    "morphTargets": [],
    "normals": [],
    "colors": [],
    "uvs": [[]],
    "faces": [],
    "edges" : []
};
```

Animation

Armature - 3D representation of bones, ligaments, and tendons

Forward kinematics

Inverse kinematics

Keyframes/Morph targets

Optimizing and Debugging WebGL

Why setTimeout is bad

- can peg the processor
- not always consistent
- not optimized for drawing the canvas

RequestAnimationFrame

- requestAnimationFrame
- mozRequestAnimationFrame
- webkitRequestAnimationFrame
- msRequestAnimationFrame
- oRequestAnimationFrame

  requestAnimFrame shim

```
window.requestAnimFrame = (function(){
      return  window.requestAnimationFrame
||
              window.webkitRequestAnimationFrame
||
              window.mozRequestAnimationFrame
||
              window.oRequestAnimationFrame
||
              window.msRequestAnimationFrame
||
              function(/* function */ callback, /*
DOMElement */ element){
                window.setTimeout(callback, 1000 /
60);
              };
    })();
```

Src: http://paulirish.com/2011/requestanimationframe-for-smart-animating/

Stats.js

FPS - frames per second

MS - how many millis it took to render the frame

MB - the allocated megabytes

Github: https://github.com/mrdoob/stats.js

```
var stats = new Stats()
$("body").append(stats.domElement);

//... in your render function
stats.update();
```

WebGL Inspector

- Allows you to incrementally step through rendering
- View texture assets and GLSL programs
- Permits capturing individual frames
- Can be embedded or installed as a Chrome/Webkit extension
- Github: http://benvanik.github.com/WebGL-Inspector/

Thanks for coming!