

WebGL™ Optimizations for Mobile

Lorenzo Dal Col
Senior Software Engineer, ARM



Agenda

1. Introduction to WebGL™ on mobile
 - Rendering Pipeline
 - Locate the bottleneck
2. Performance analysis and debugging tools for WebGL
 - Generic optimization tips
3. PlayCanvas experience
 - WebGL Inspector
4. Use case: PlayCanvas Swooop
 - ARM® DS-5 Streamline
 - ARM Mali™ Graphics Debugger
5. Q & A



Bring the Power of OpenGL[®] ES to Mobile Browsers

What is WebGL[™]?

- A cross-platform, royalty free web standard
- Low-level 3D graphics API
- Based on OpenGL[®] ES 2.0
- A shader based API using GLSL (OpenGL Shading Language)
- Some concessions made to JavaScript[™] (memory management)

Why WebGL?

- It brings plug-in free 3D to the web, implemented right into the browser.
- Major browser vendors are members of the WebGL Working Group:
 - Apple (Safari[®] browser)
 - Mozilla (Firefox[®] browser)
 - Google (Chrome[™] browser)
 - Opera (Opera[™] browser)



Introduction to WebGL™

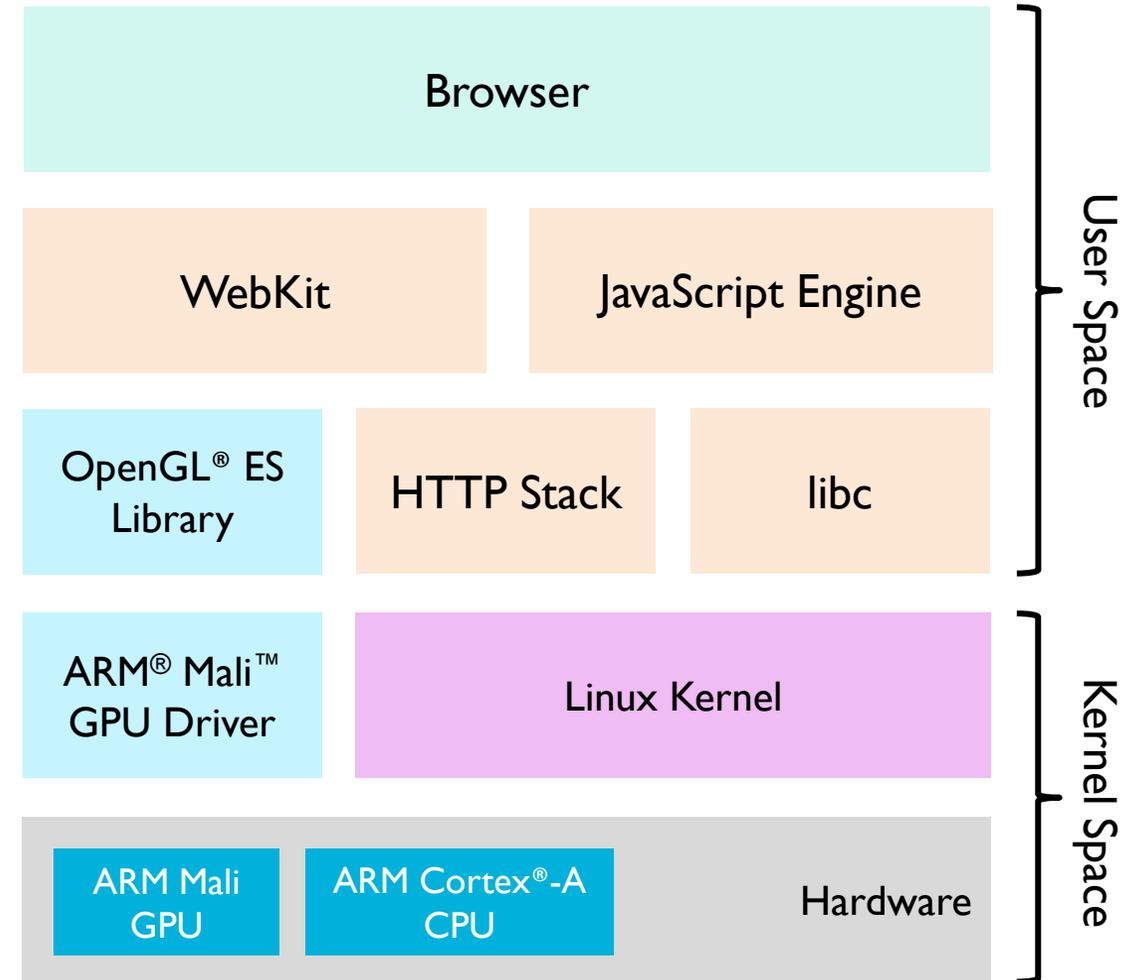
- How does it fit in a web browser?
 - You use JavaScript™ to control it.
 - Your JavaScript is embedded in HTML5 and uses its Canvas element to draw on.
- What do you need to start creating graphics?
 - Obtain WebGLRenderingContext object for a given HTMLCanvasElement.
 - It creates a drawing buffer into which the API calls are rendered.
 - For example:

```
var canvas = document.getElementById('canvas1');  
var gl = canvas.getContext('webgl');  
canvas.width = newWidth;  
canvas.height = newHeight;  
gl.viewport(0, 0, canvas.width, canvas.height);
```

WebGL™ Stack

What is happening when a WebGL page is loaded

- User enters URL
- HTTP stack requests the HTML page
- Additional requests will be necessary to get JavaScript™ code and other resources
- JavaScript code will be pre-parsed while loading other assets and the DOM tree is built
- JavaScript code will contain calls to the WebGL API
 - They will go back to WebKit®, which calls OpenGL® ES 2.0 library
 - Shaders are compiled
 - Textures, vertex buffers & uniforms must be loaded to the GPU
 - Rendering can start

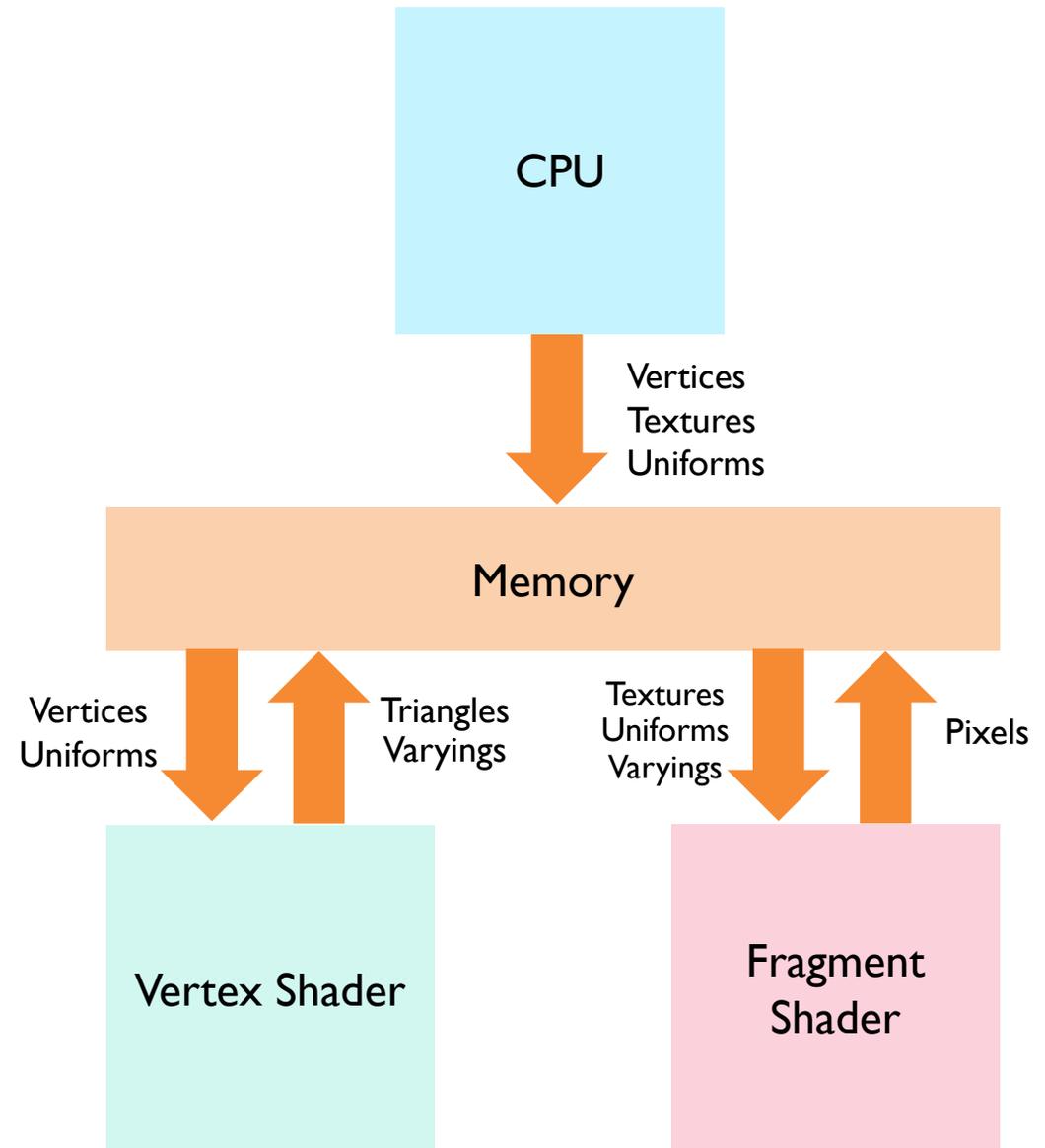


See Chromium Rendering Stack:
<http://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome>



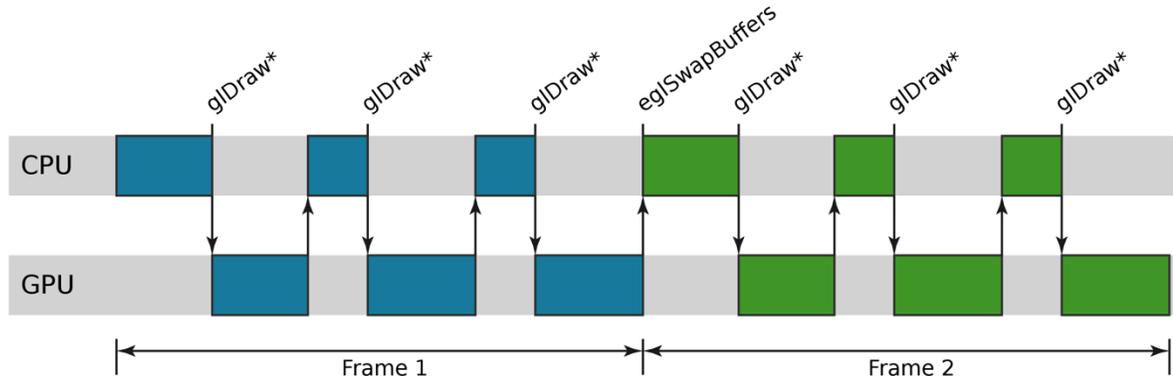
Locate the Bottleneck

- The frame rate of a particular WebGL™ application could be limited by:
 - CPU
 - Vertex Shader
 - Fragment Shader
 - Bandwidth
- Fortunately we have tools to understand which one is the culprit



Frame Rendering Time

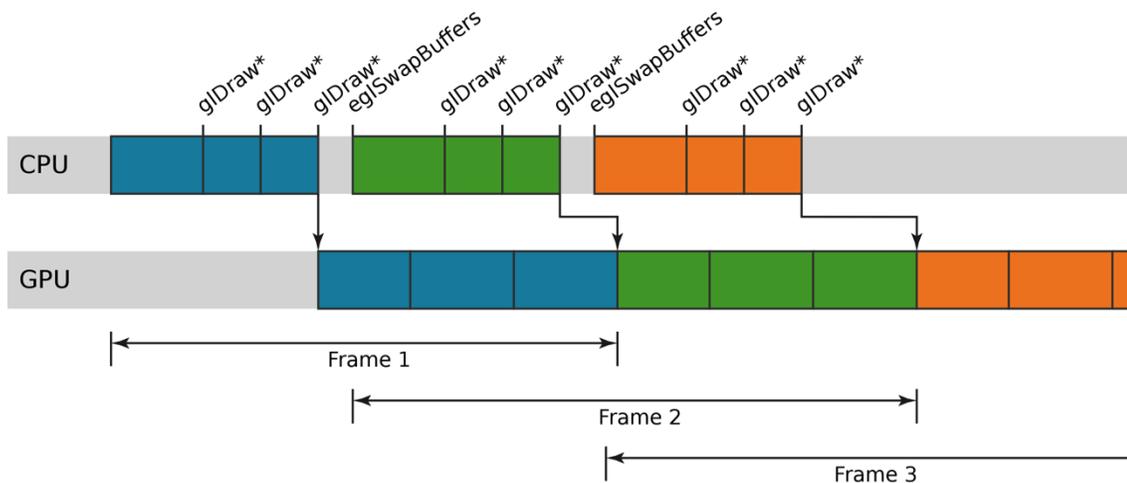
Synchronous Rendering



```
// THIS DOES NOT MEASURE GPU RENDERING
```

```
var start = new Date().getTime();  
gl.drawElements(gl.TRIANGLE, ...);  
var time = new Date().getTime() - start;
```

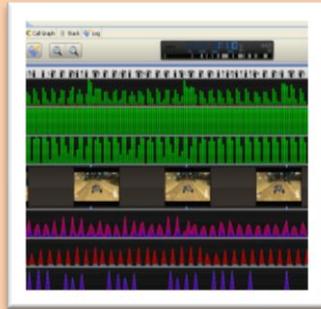
Deferred Rendering



```
// THIS FORCES SYNCHRONOUS RENDERING  
// (BAD PRACTICE)
```

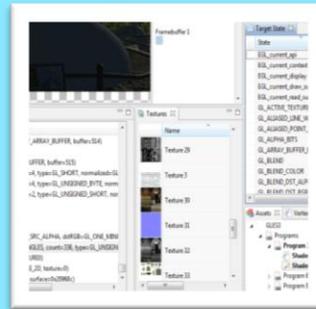
```
var start = new Date().getTime();  
gl.drawElements(gl.TRIANGLE, ...);  
gl.finish(); // or gl.readPixels...  
var time = new Date().getTime() - start;
```

Performance Analysis & Debug



DS-5 Streamline

- System-wide performance analysis
- Combined ARM® Cortex® Processors and Mali™ GPU visibility
- Optimize for performance & power across the system



Mali Graphics Debugger

- API Trace & Debug Tool
- Understand graphics and compute issues at the API level
- Debug and improve performance at frame level
- Support for OpenGL® ES 1, 1.1, 2.0, 3.0 and OpenCL™ 1.1

```
ARM Mali Offline Shader Compiler version 4.0.0
(C) Copyright 2007-2012 ARM Limited.
All rights reserved.

Mali-200/300/400 driver version r1p0-01r11
Mali-T600 series driver version r1p0-04r10

usage: malioc.exe [options] [-o outfile] --core=core
       -NAME[-VALUE]      Predefine NAME as a
       --vert             Process shader as a
       --frag             Process shader as a
       -V, --verbose      Print verbose info
       -o outfile         write output to outfile
       --core=core       Target specified gra
Supported cores are:
Mali-200
Mali-300
Mali-400
Mali-450
Mali-T600
Mali-T650
Target hardware rele
Mali-T600 (Mali-T650
r1p0
r1p0-15dev0
```

Offline Compilers

- Understand complexity of GLSL shaders and CL kernels
- Support for Mali-4xx and Mali-T6xx GPU families

PlayCanvas

SWOOOP

- HTML5/WebGL™ game built with PlayCanvas
- Demonstration that high-quality arcade gaming is possible with HTML5+WebGL across desktop, tablets and smartphones
- Cross platform touch, mouse and keyboard controls
- *Low poly* art style
 - Flat shaded surfaces with ambient occlusion combined with diffuse color



PlayCanvas Swoop Gameplay

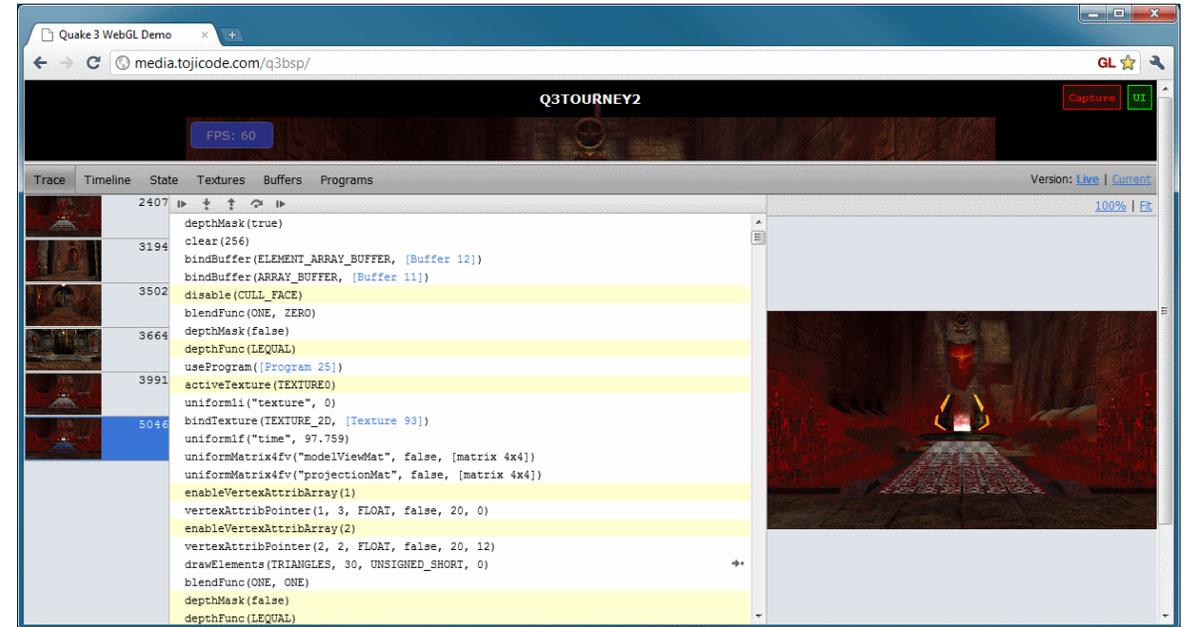
Running in the Chrome™ browser on a Google Nexus 10 with Android™ 4.4



<http://swoop.playcanvas.com/>

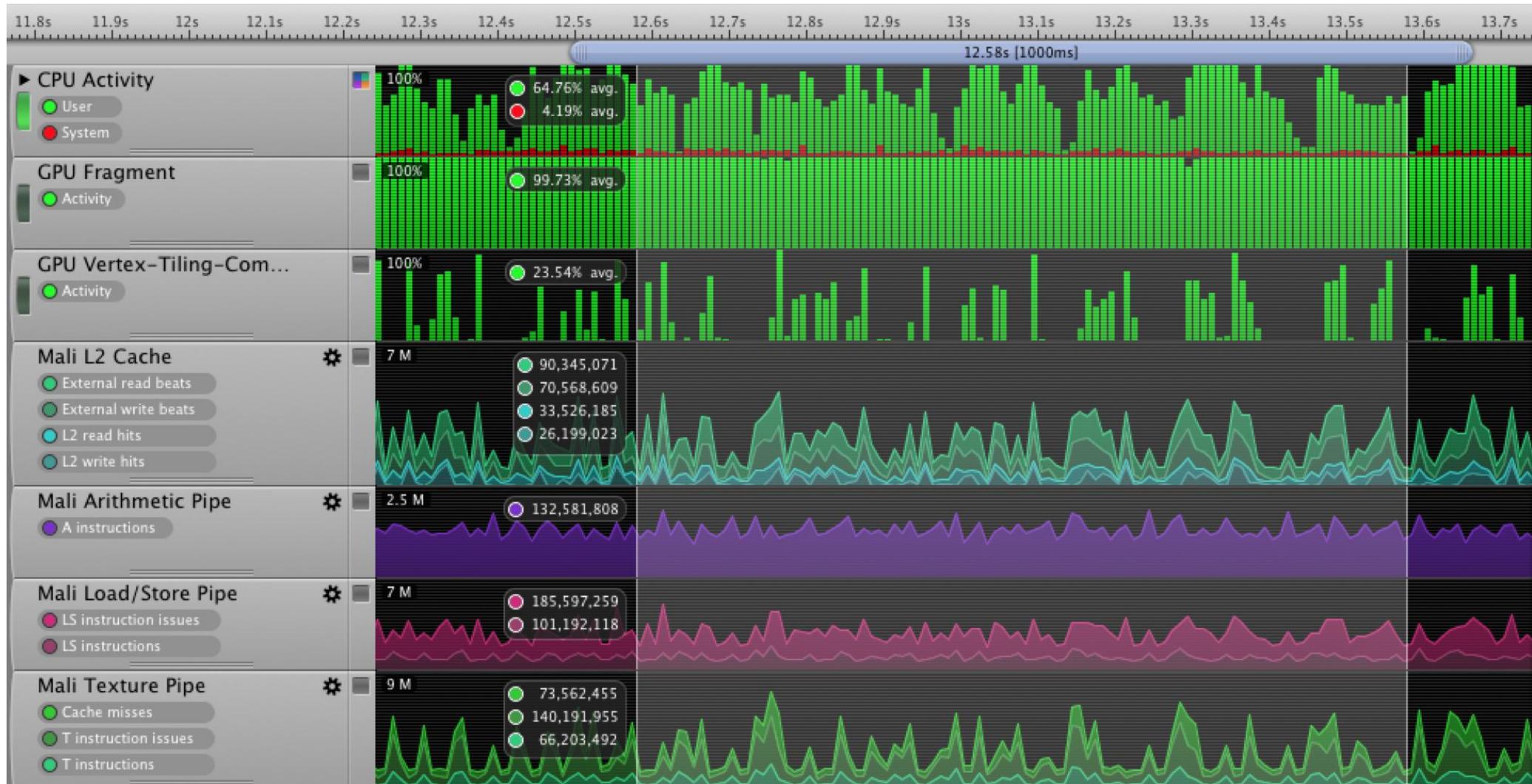
PlayCanvas Experience

- **WebGL Inspector** can be very useful to optimize the stream of commands that are submitted to WebGL™
 - It's very good at highlighting redundant calls
 - It's also an important debugging tool (i.e. debugging draw order and render state problems)
- **GLSL Optimizer** has been used to check that the GLSL that PlayCanvas procedurally generates is reasonably optimal



- <http://benvanik.github.io/WebGL-Inspector/>
- <https://github.com/aras-p/glsl-optimizer>

ARM® DS-5 Streamline



Performance Optimization

How to reduce the CPU and system workload

- Reduce your number of draw calls
 - Models using the same shaders can be batched to reduce draw calls
 - Even when they have different shaders, sometimes batching makes sense
- Do not force a pipeline flush by reading back data (gl.readPixels, gl.finish, etc.)
- Move from CPU to GPU
 - Rotation matrix computation can be moved to the vertex shader (by passing a timestamp)
- Avoid unnecessary WebGL™ calls (gl.getError, redundant stage changes, etc.)
 - WebGL Inspector shows redundant calls
 - Models can be sorted to avoid state changes
- Pre-calculate positions
- Use typed arrays instead of JavaScript™ object arrays:

```
var vertices = new Array(size);
```

```
var vertices = new Float32Array(size);
```

See also: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays

Fragment Bound and Bandwidth Optimizations

Reduce Bandwidth Usage

- Use texture mipmapping
- Reduce the size of the textures
- Reduce the number of vertices and varyings
- Interleave vertices, normals, texture coordinates

Most of these optimizations will also cause a **better cache utilization**

Reduce the Fragment Activity

- Render to a smaller framebuffer
 - This will upscale the rendered frame to the size of the HTML canvas
- Move computation from the fragment to the vertex shader (use HW interpolation)
- Consider overdraw

ARM® Mali™ Graphics Debugger

The screenshot displays the ARM Mali Graphics Debugger interface with the following components:

- Outline:** A tree view of frames and draw elements. Frame 264 is selected, showing 253 draws and 318731 vertices. It lists 26 draw elements, each with a count of 1284 vertices.
- Assets:** A 3D preview window showing a landscape scene with a mountain and trees. A coordinate box indicates the position (1254,7) [209, 223, 117].
- Statistics:** A table of performance metrics:

Total number of API function calls	171664
Average vert/frame	33069.02
Average instanced vert/frame	0.00
Average draw/frame	32.16
Frame #	264
Number of API calls in current frame	4153
Number of draw calls in current frame	253
Number of vertices submitted in current frame	318731
API call #	135539
Number of vertices in current draw call	1284
Number of unique indices in current draw call	1283
Indices sparseness	1.00
- Target State:** A table of OpenGL state items:

State Item	Value
GL_COLOR_CLEAR_VALUE	0.9137255, 0.6901961, 0.74117...
GL_COLOR_WRITEMASK	GL_TRUE, GL_TRUE, GL_TRUE, GL...
GL_COMPRESSED_TEXTURE_FORMATS	<unknown>
GL_COPY_READ_BUFFER_BINDING	0
GL_COPY_WRITE_BUFFER_BINDING	0
GL_CULL_FACE	GL_TRUE
GL_CULL_FACE_MODE	GL_BACK
GL_CURRENT_PROGRAM	Program 54
GL_DEPTH_BITS	<unknown>
GL_DEPTH_CLEAR_VALUE	1.0
GL_DEPTH_FUNC	GL_LESS
GL_DEPTH_RANGE	0.0, 1.0
GL_DEPTH_TEST	GL_TRUE
GL_DEPTH_WRITEMASK	GL_TRUE
- Function Call:** A list of OpenGL function calls for the selected draw element, including `glVertexAttribPointer`, `glUniformMatrix3fv`, `glUniformMatrix4fv`, `glEnableVertexAttribArray`, and `glDrawElements`.
- Trace Analysis:** A table of messages:

Message	Count
Offset is beyond the end of the buffer	12
Binding a buffer that was initially bound to a different target is not recommended	1
Texture level not previously defined	4
- Vertex Shaders:** A table of texture information:

Name	Size	Format	Type
Texture 11	2560 x 1024	GL_RGBA	GL_UNSIGNED...
Texture 5	1024 x 1024	GL_RGBA	GL_UNSIGNED...
Texture 204	1024 x 1024	GL_RGBA	GL_UNSIGNED...
Texture 220	1024 x 1024	GL_RGBA	GL_UNSIGNED...

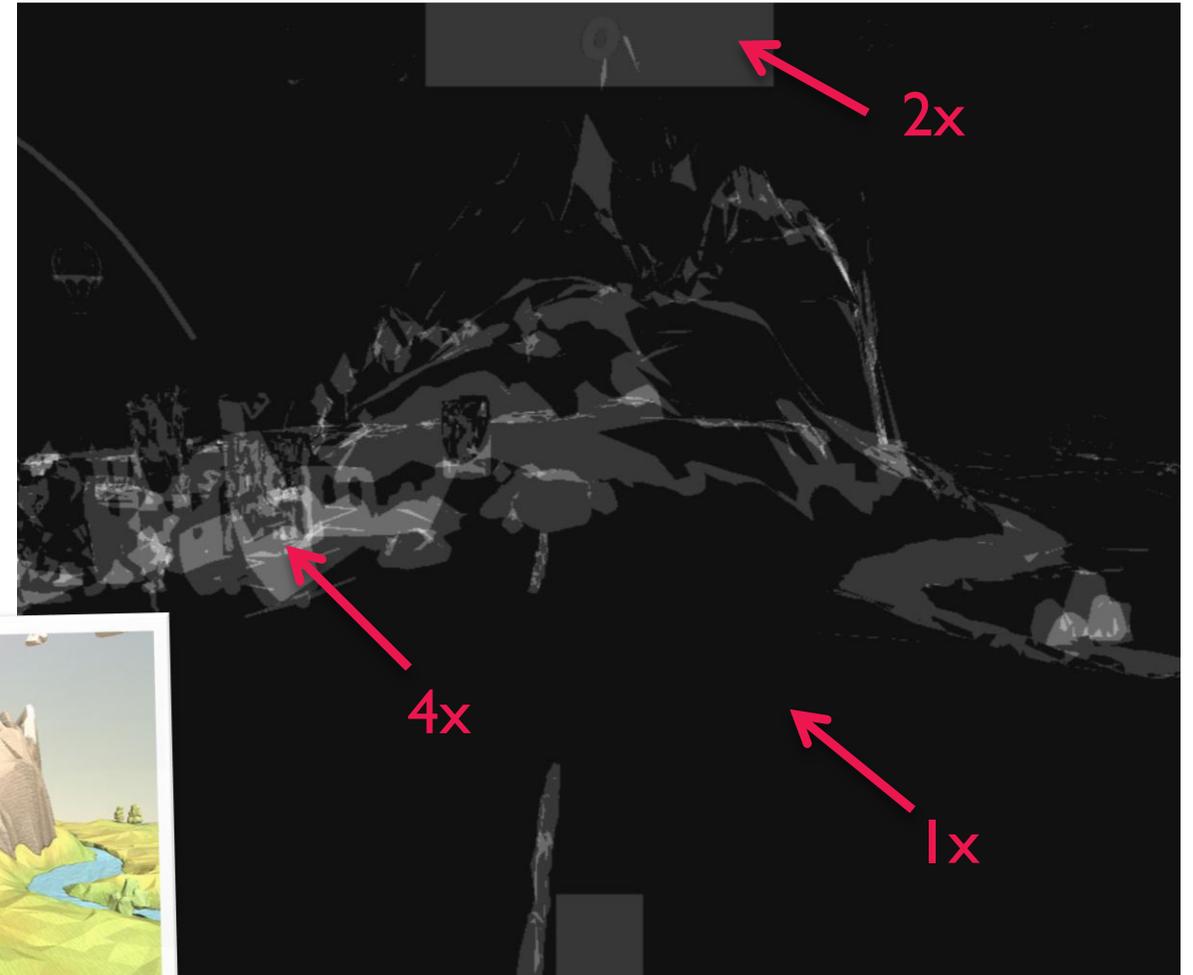
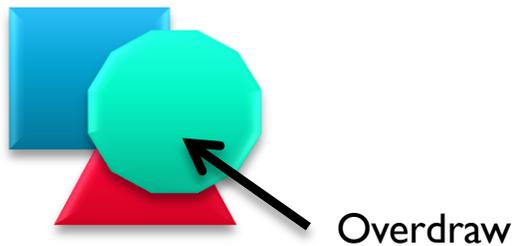
Frame Capture

Draw 001, total vertices: 15402



Overdraw

- This is when you draw to each pixel on the screen more than once
- Drawing your objects front to back instead of back to front reduces overdraw
- Also limiting the amount of transparency in the scene can help

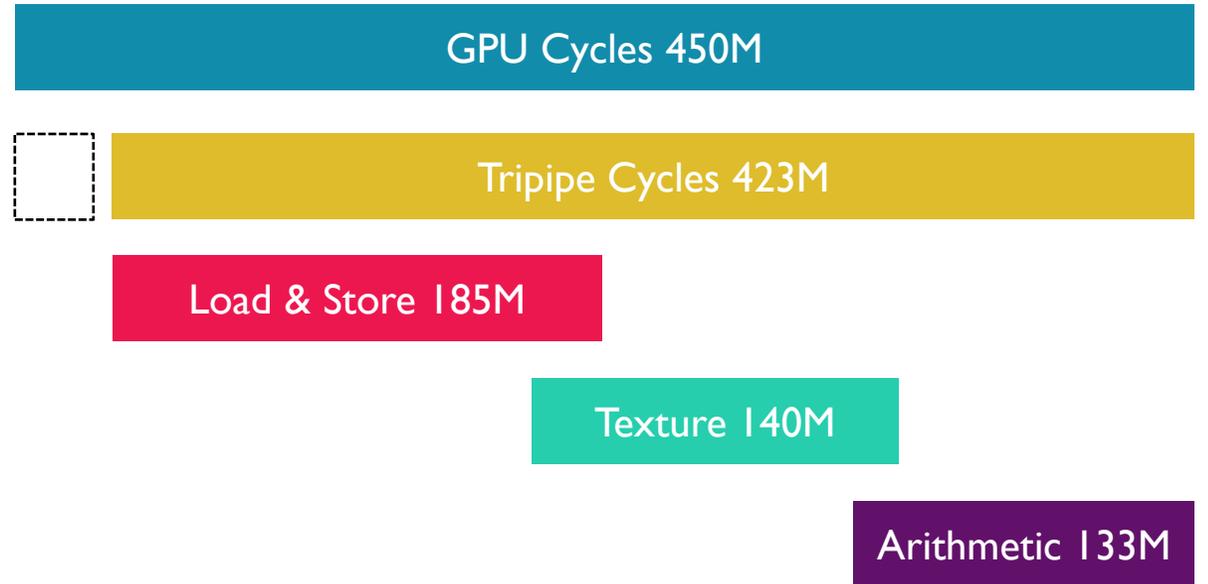
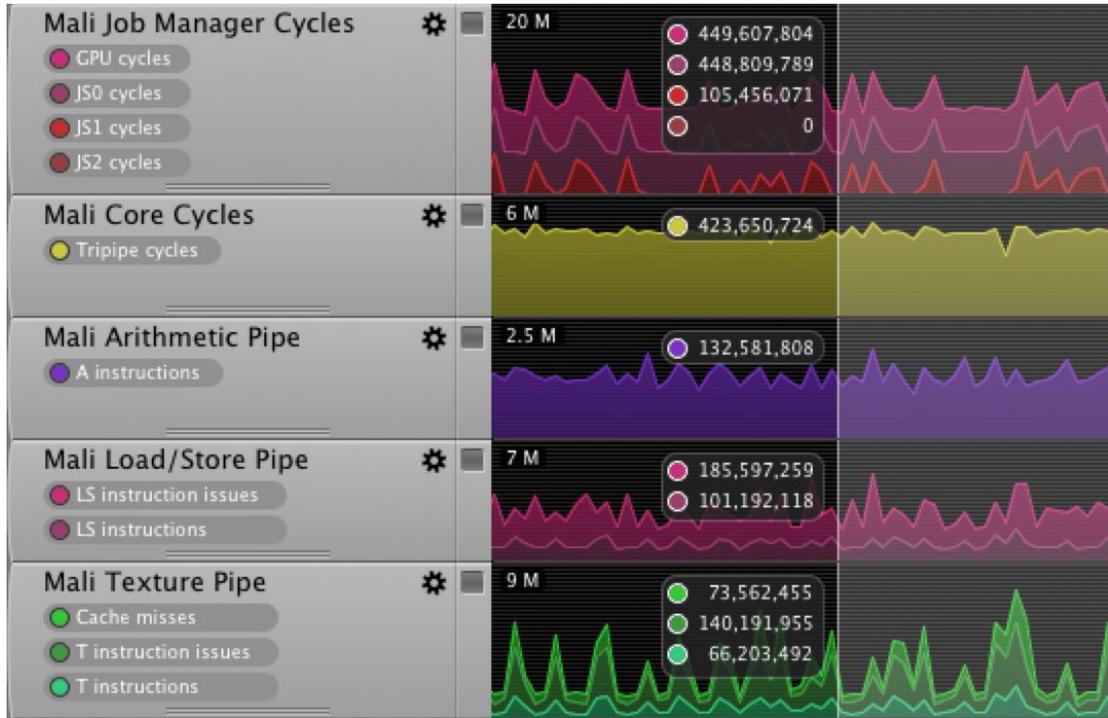


Shader Map and Fragment Count

Program Name	Instructions	Shortest path	Longest path	Instances	Total cycles
54 Shader 53	20	15	19	730781	12423277
60 Shader 59	18	13	17	326609	4899135
43 Shader 44	20	15	19	6358	108086
57 Shader 56	21	16	20	1389	25002
39 Shader 38	1	1	1	19840	19840
51 Shader 50	27	22	26	566	13584
36 Shader 35	7	7	7	1191	8337
42 Shader 41	1	1	1	1160	1160
48 Shader 47	28	23	27	0	0
15 Shader 14	2	2	2	N/A	N/A
18 Shader 17	2	2	2	N/A	N/A
21 Shader 20	4	2	3	N/A	N/A
24 Shader 23	4	2	3	N/A	N/A
27 Shader 26	7	7	7	N/A	N/A
30 Shader 29	7	7	7	N/A	N/A



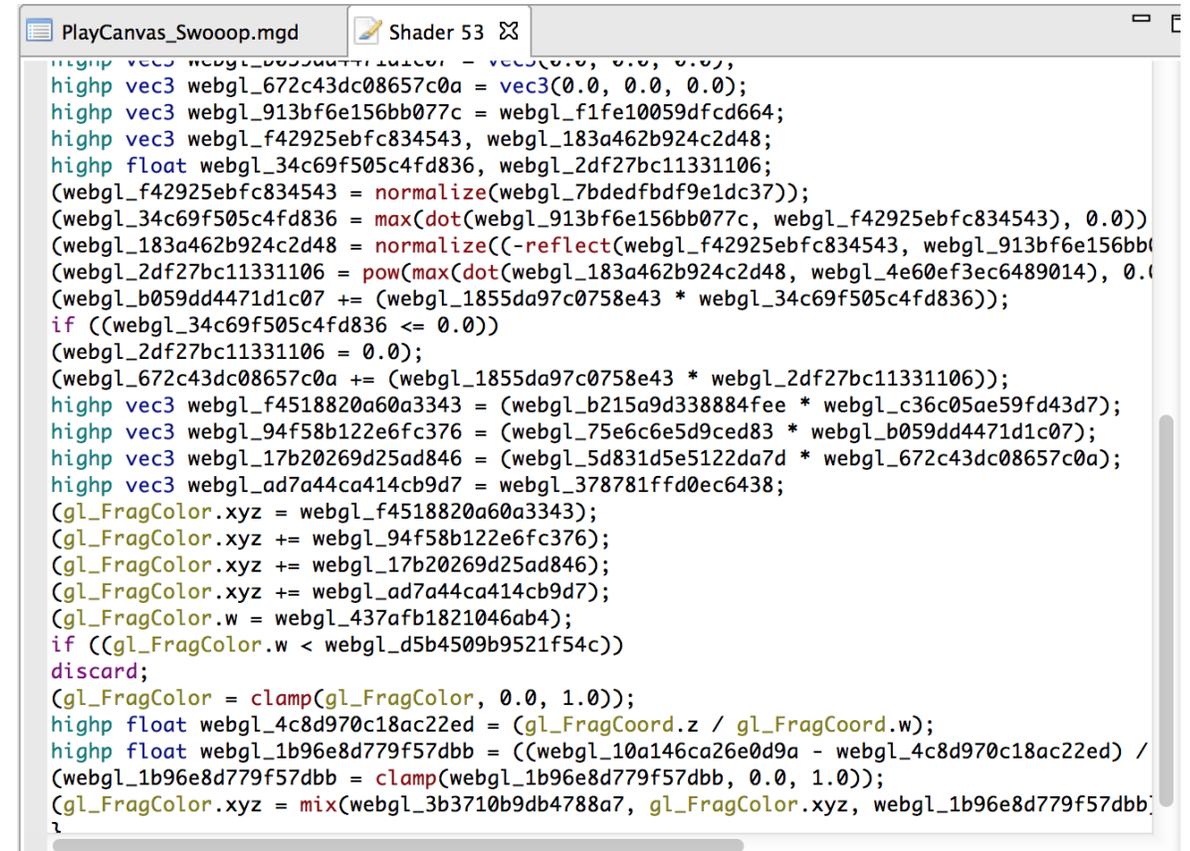
Inspect the Tripipe Counters



Shader Optimization

- Since the arithmetic workload is not very big, we could **reduce the number** of uniforms and varyings and calculate them on-the-fly
- **Reduce their size**
- **Reduce their precision:** all the varyings, uniforms and local variables are **highp**, is that really necessary?
- Use the ARM[®] Mali[™] Offline Shader Compiler!

<http://malideveloper.arm.com/develop-for-mali/tools/analysis-debug/mali-gpu-offline-shader-compiler/>



```
PlayCanvas_Swooop.mgd Shader 53
highp vec3 webgl_b059dd4471d1c07 = vec3(0.0, 0.0, 0.0);
highp vec3 webgl_672c43dc08657c0a = vec3(0.0, 0.0, 0.0);
highp vec3 webgl_913bf6e156bb077c = webgl_f1fe10059dfcd664;
highp vec3 webgl_f42925ebfc834543, webgl_183a462b924c2d48;
highp float webgl_34c69f505c4fd836, webgl_2df27bc11331106;
(webgl_f42925ebfc834543 = normalize(webgl_7bdefbdf9e1dc37));
(webgl_34c69f505c4fd836 = max(dot(webgl_913bf6e156bb077c, webgl_f42925ebfc834543), 0.0));
(webgl_183a462b924c2d48 = normalize((-reflect(webgl_f42925ebfc834543, webgl_913bf6e156bb077c), 0.0)));
(webgl_2df27bc11331106 = pow(max(dot(webgl_183a462b924c2d48, webgl_4e60ef3ec6489014), 0.0), 0.0));
(webgl_b059dd4471d1c07 += (webgl_1855da97c0758e43 * webgl_34c69f505c4fd836));
if ((webgl_34c69f505c4fd836 <= 0.0))
(webgl_2df27bc11331106 = 0.0);
(webgl_672c43dc08657c0a += (webgl_1855da97c0758e43 * webgl_2df27bc11331106));
highp vec3 webgl_f4518820a60a3343 = (webgl_b215a9d338884fee * webgl_c36c05ae59fd43d7);
highp vec3 webgl_94f58b122e6fc376 = (webgl_75e6c6e5d9ced83 * webgl_b059dd4471d1c07);
highp vec3 webgl_17b20269d25ad846 = (webgl_5d831d5e5122da7d * webgl_672c43dc08657c0a);
highp vec3 webgl_ad7a44ca414cb9d7 = webgl_378781ffd0ec6438;
gl_FragColor.xyz = webgl_f4518820a60a3343;
gl_FragColor.xyz += webgl_94f58b122e6fc376;
gl_FragColor.xyz += webgl_17b20269d25ad846;
gl_FragColor.xyz += webgl_ad7a44ca414cb9d7;
gl_FragColor.w = webgl_437afb1821046ab4;
if ((gl_FragColor.w < webgl_d5b4509b9521f54c))
discard;
gl_FragColor = clamp(gl_FragColor, 0.0, 1.0);
highp float webgl_4c8d970c18ac22ed = (gl_FragCoord.z / gl_FragCoord.w);
highp float webgl_1b96e8d779f57dbb = ((webgl_10a146ca26e0d9a - webgl_4c8d970c18ac22ed) / (webgl_1b96e8d779f57dbb = clamp(webgl_1b96e8d779f57dbb, 0.0, 1.0)));
gl_FragColor.xyz = mix(webgl_3b3710b9db4788a7, gl_FragColor.xyz, webgl_1b96e8d779f57dbb);
```

References

- *Professional WebGL Programming*, Andreas Anyuru (2012)
- *Debugging and Optimizing WebGL Applications*, Ben Vanik and Ken Russell (2011)

- Where to find more info?
 - <http://www.khronos.org/webgl/>
 - <http://en.wikipedia.org/wiki/HTML5>
 - http://en.wikipedia.org/wiki/Canvas_element
 - <http://www.khronos.org/webgl/wiki/Tutorial>
 - <https://playcanvas.com/>

Thank You

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Any other marks featured may be trademarks of their respective owners