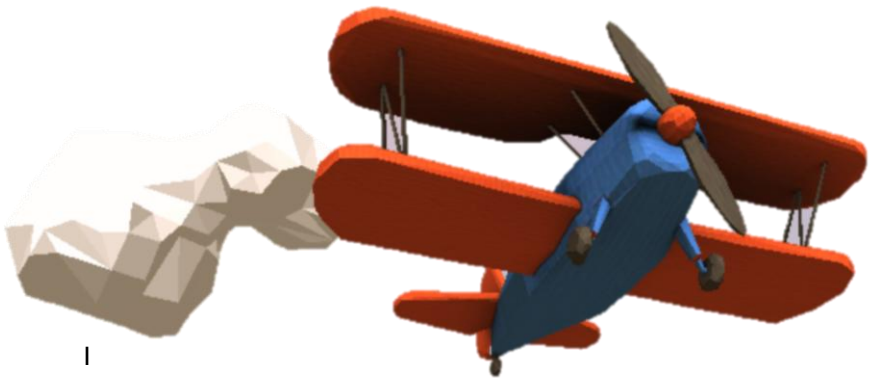
WebGL™ Optimizations for Mobile

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

ARM

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)
  - Google (Chrome™ browser)
  - Mozilla (Firefox® browser)
  - Opera (Opera™ browser)

ARM

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.

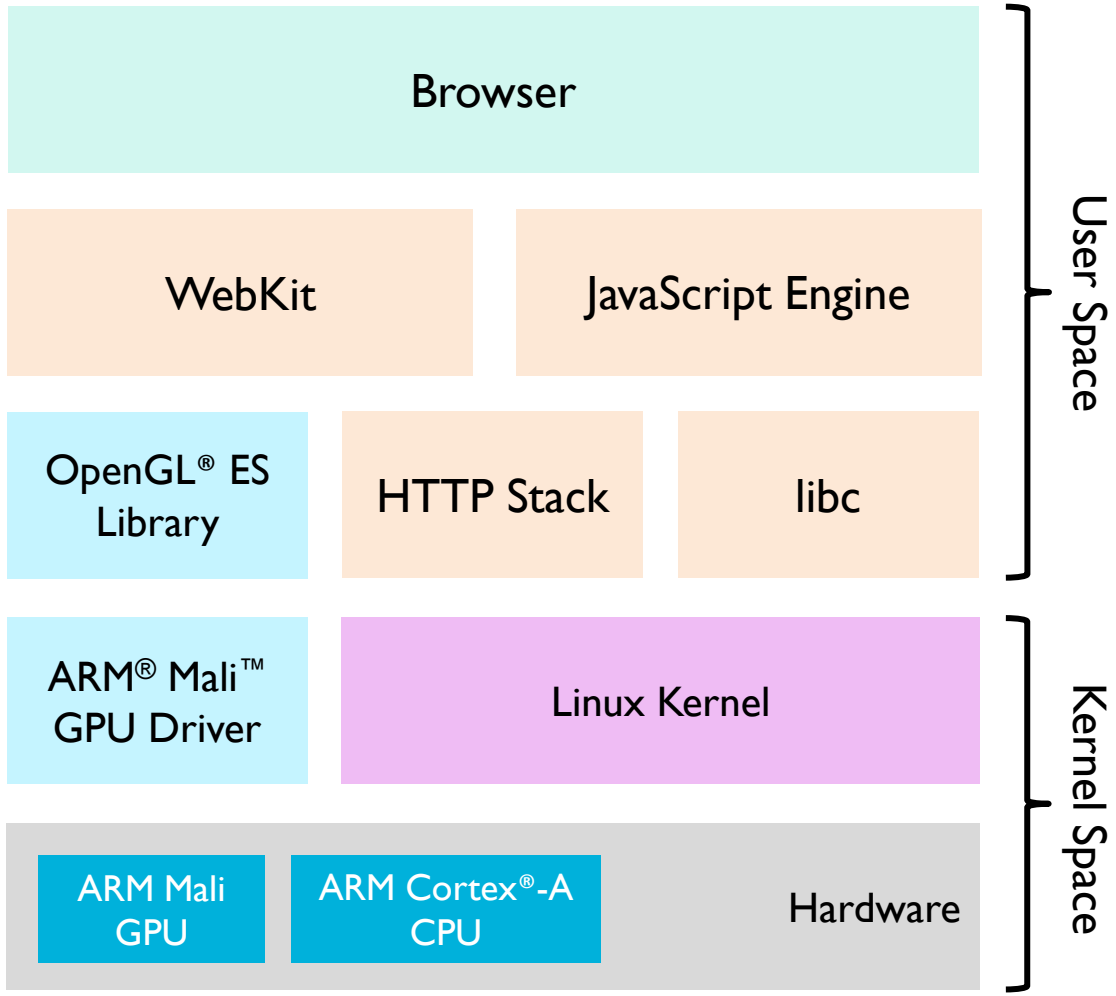
- What do you need to start creating graphics?
  - Obtain WebGLrenderingContext object for a given HTMLCanvasElement.
  - It creates a drawing buffer into which the API calls are rendered.
  - For example:

```
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

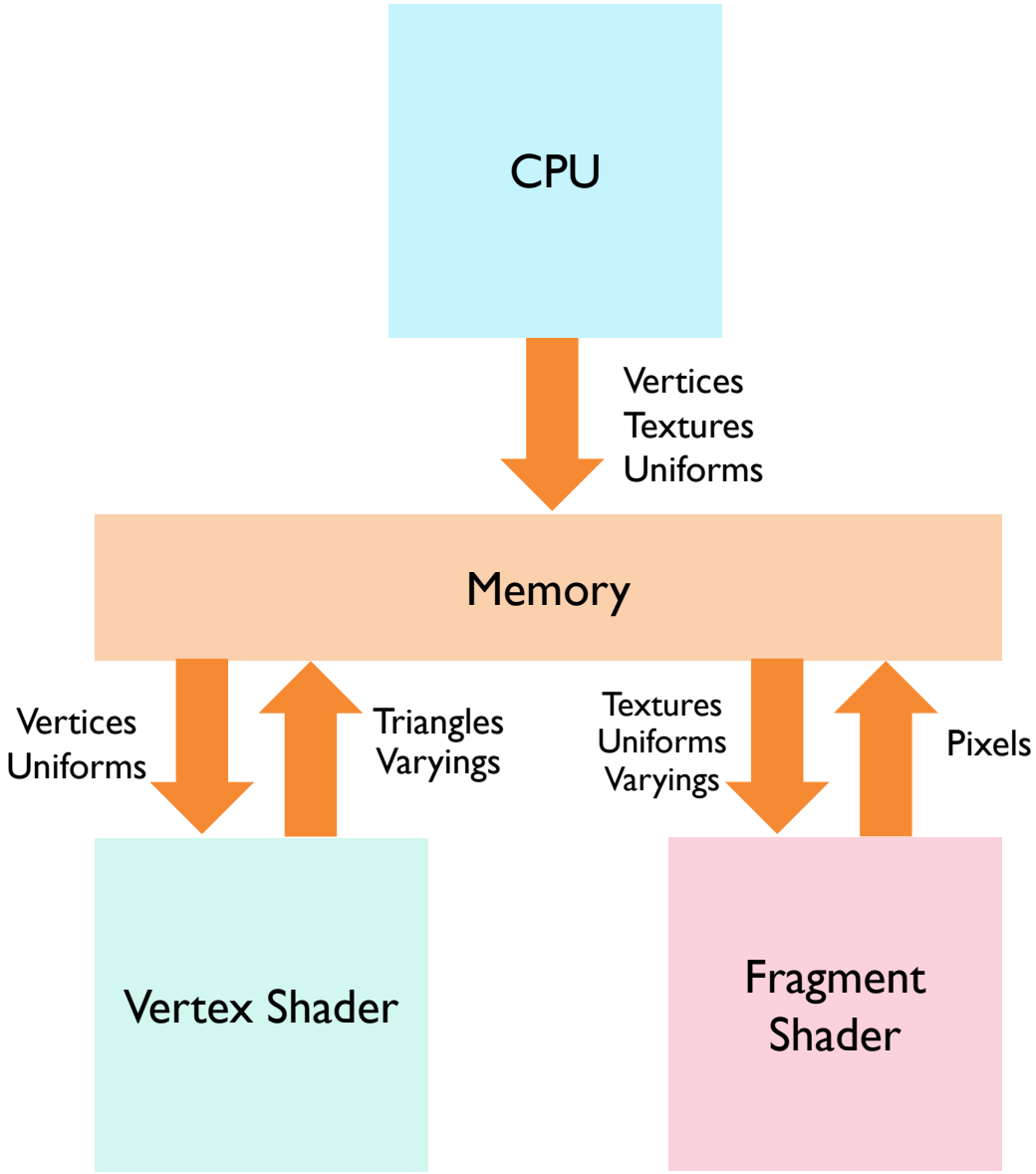


See Chromium Rendering Stack:
http://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome

ARM

Locate the Bottleneck

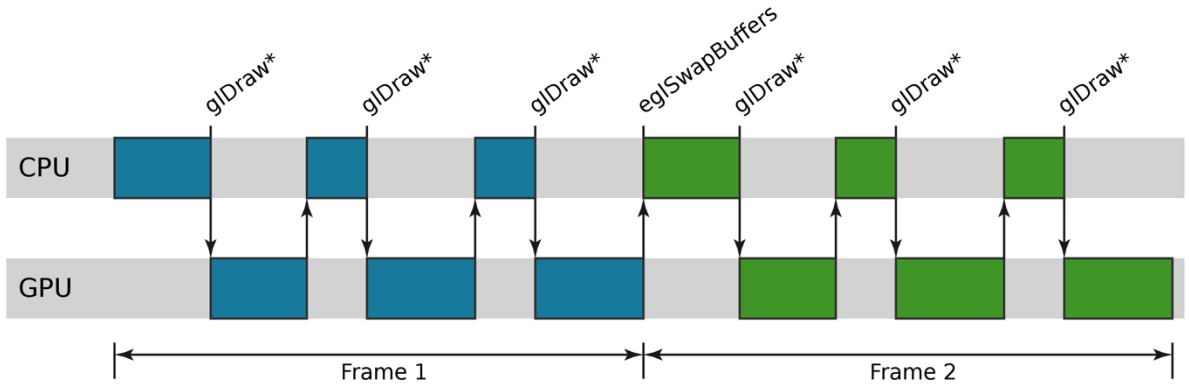
- The frame rate of a particular WebGL™ application could be limited by:
  - CPU
  - Vertex Shader
  - Fragment Shader
  - Bandwidth
- Fortunately we have tools to understand which one is the culprit



ARM

Frame Rendering Time

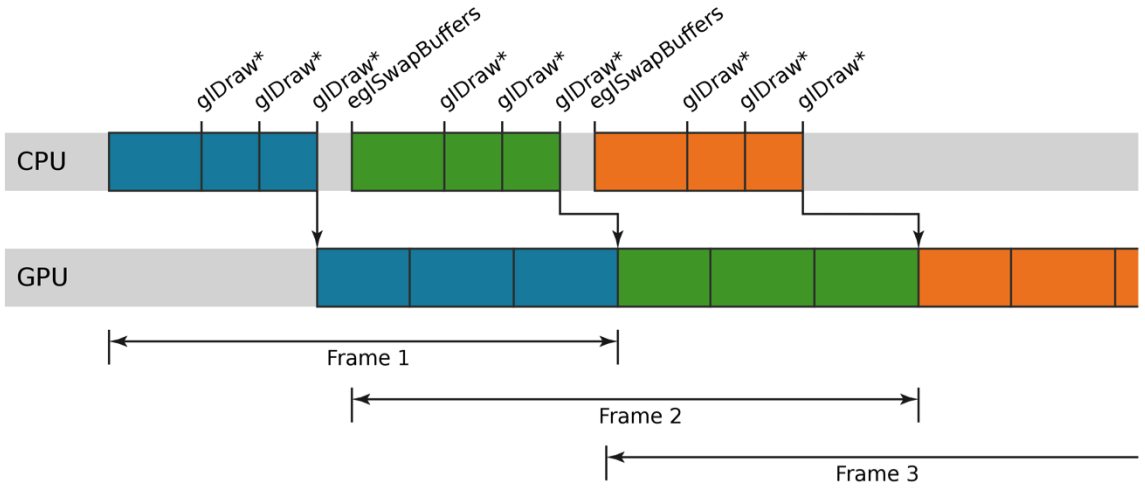
Synchronous Rendering




```
// THIS DOES NOT MEASURE GPU RENDERING

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

Deferred Rendering

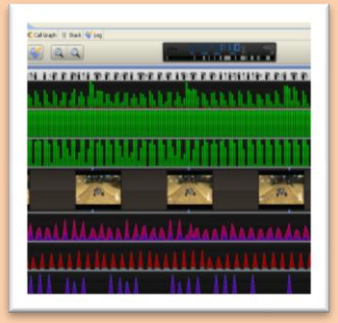



```
// THIS FORCES SYNCHRONOUS RENDERING
// (BAD PRACTICE)

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

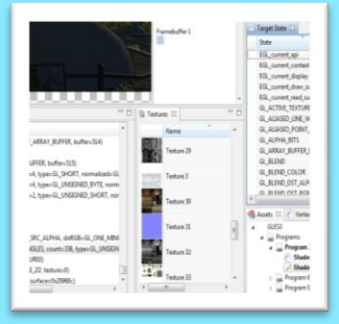
ARM

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, 2.0, 3.0 and OpenCL™ 1.1

Offline Compilers

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

ARM

PlayCanvas

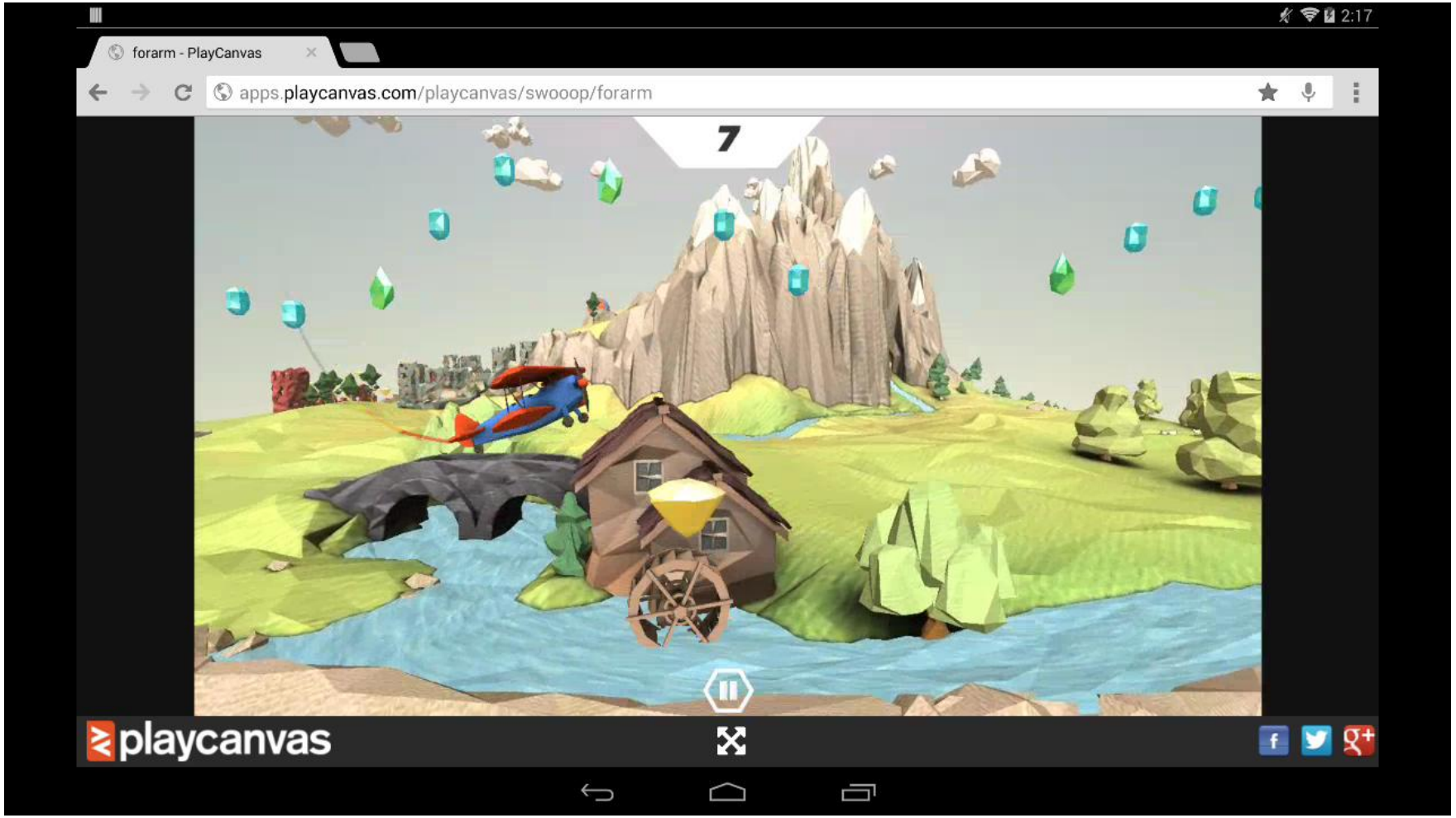
SWOOOP

- 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

ARM

PlayCanvas Swooop Gameplay

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

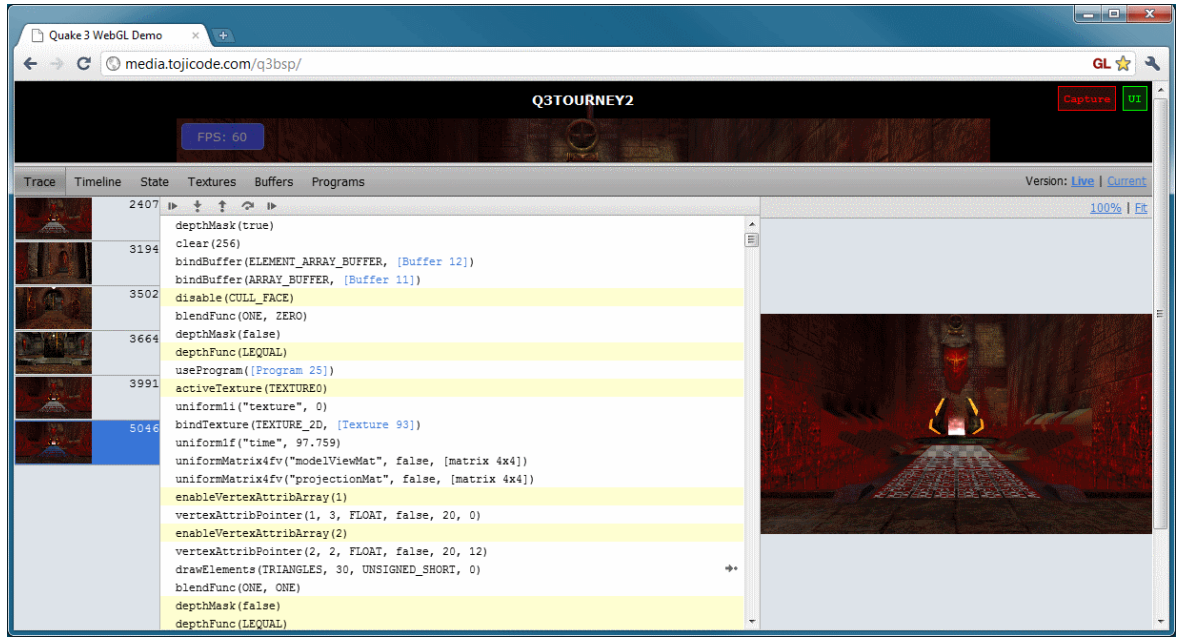


http://swooop.playcanvas.com/

ARM

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)

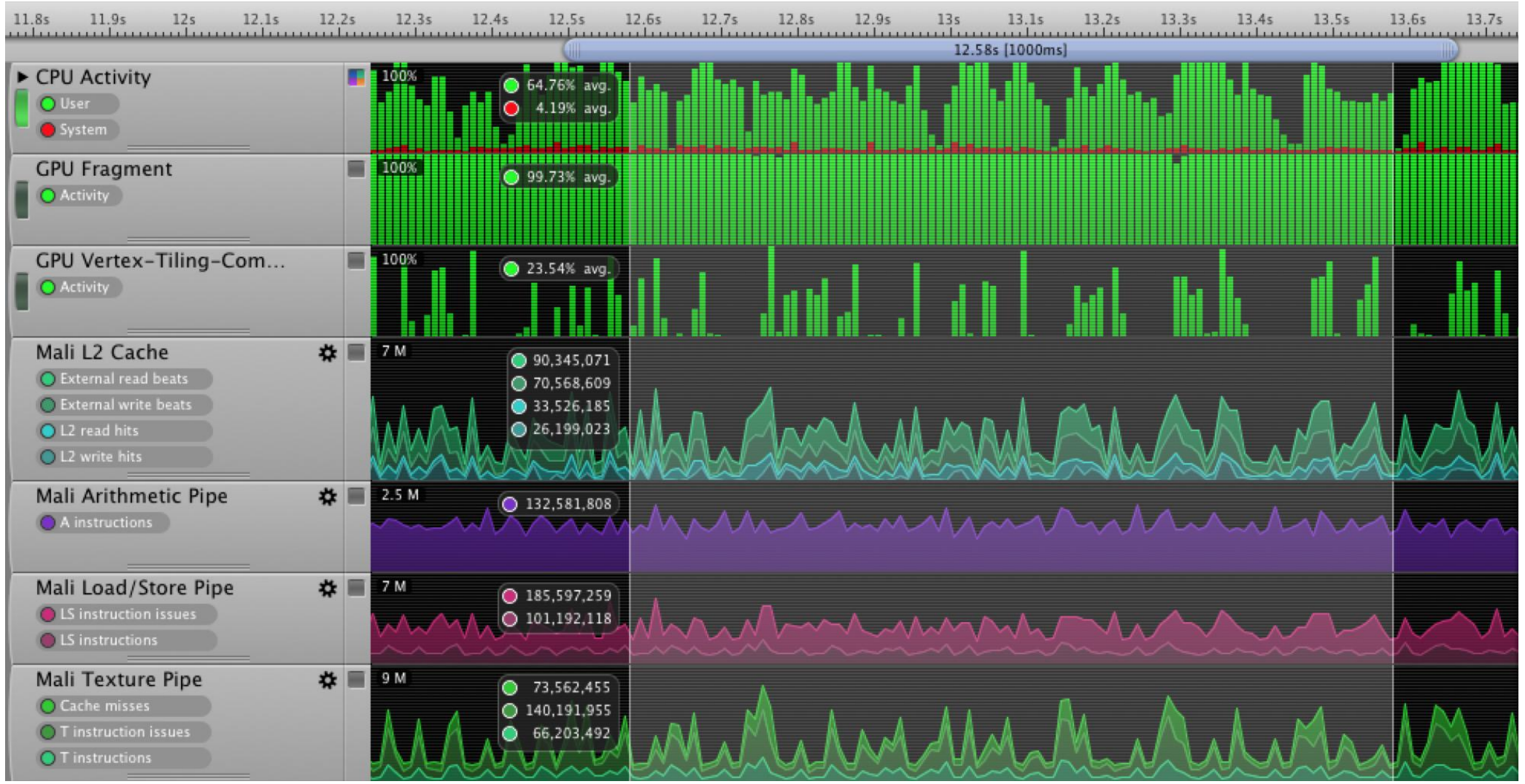


- GLSL Optimizer has been used to check that the GLSL that PlayCanvas procedurally generates is reasonably optimal

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

ARM

ARM® DS-5 Streamline



ARM

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

ARM

Fragment Bound and Bandwidth Optimizations

Reduce Bandwidth Usage

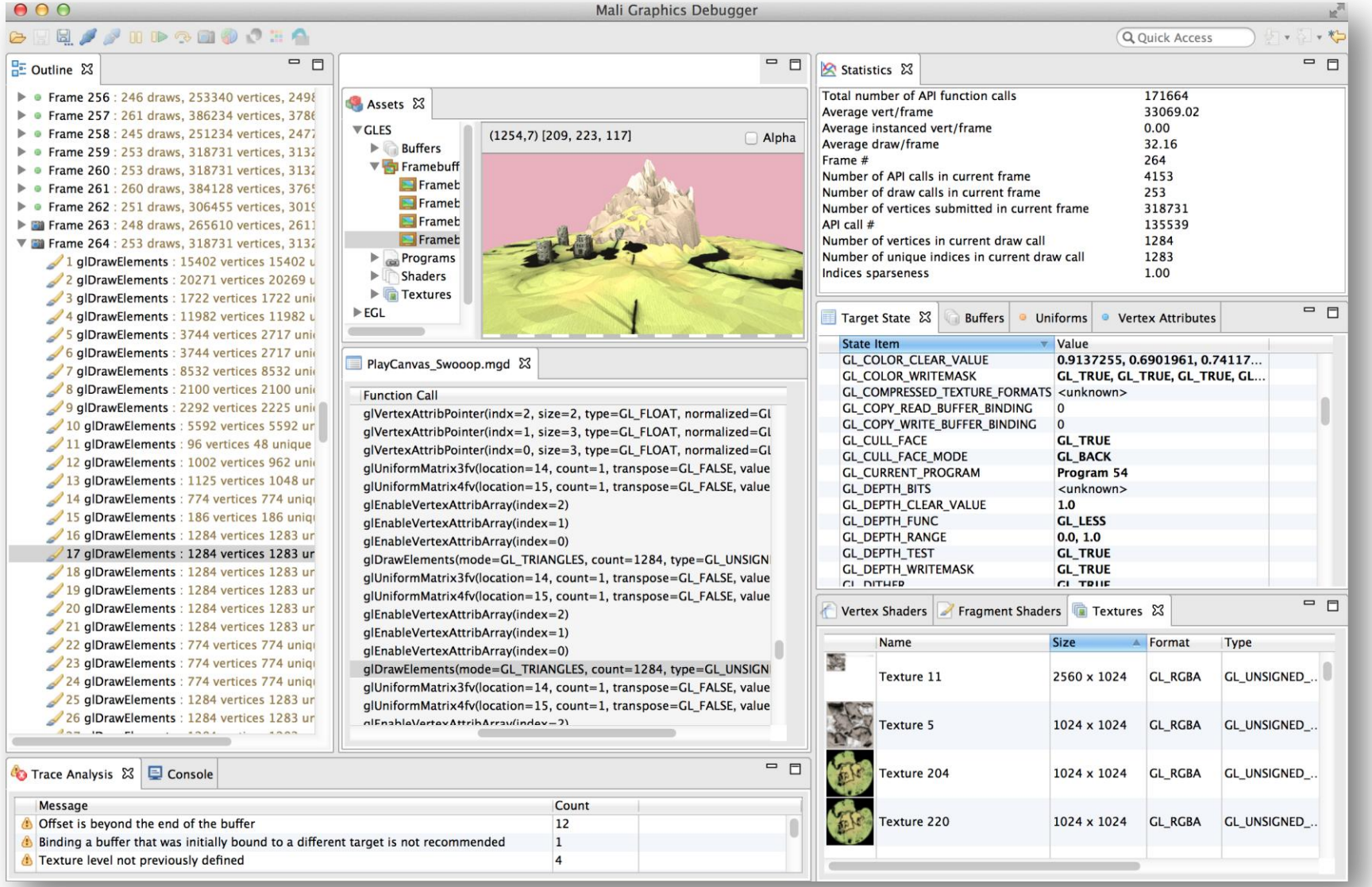
- Use texture mipmapping
- Reduce the size of the textures
- Reduce the number of vertices and varyings
- Interleave vertices, normals, texture coordinates

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

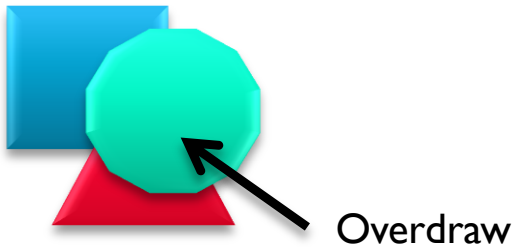


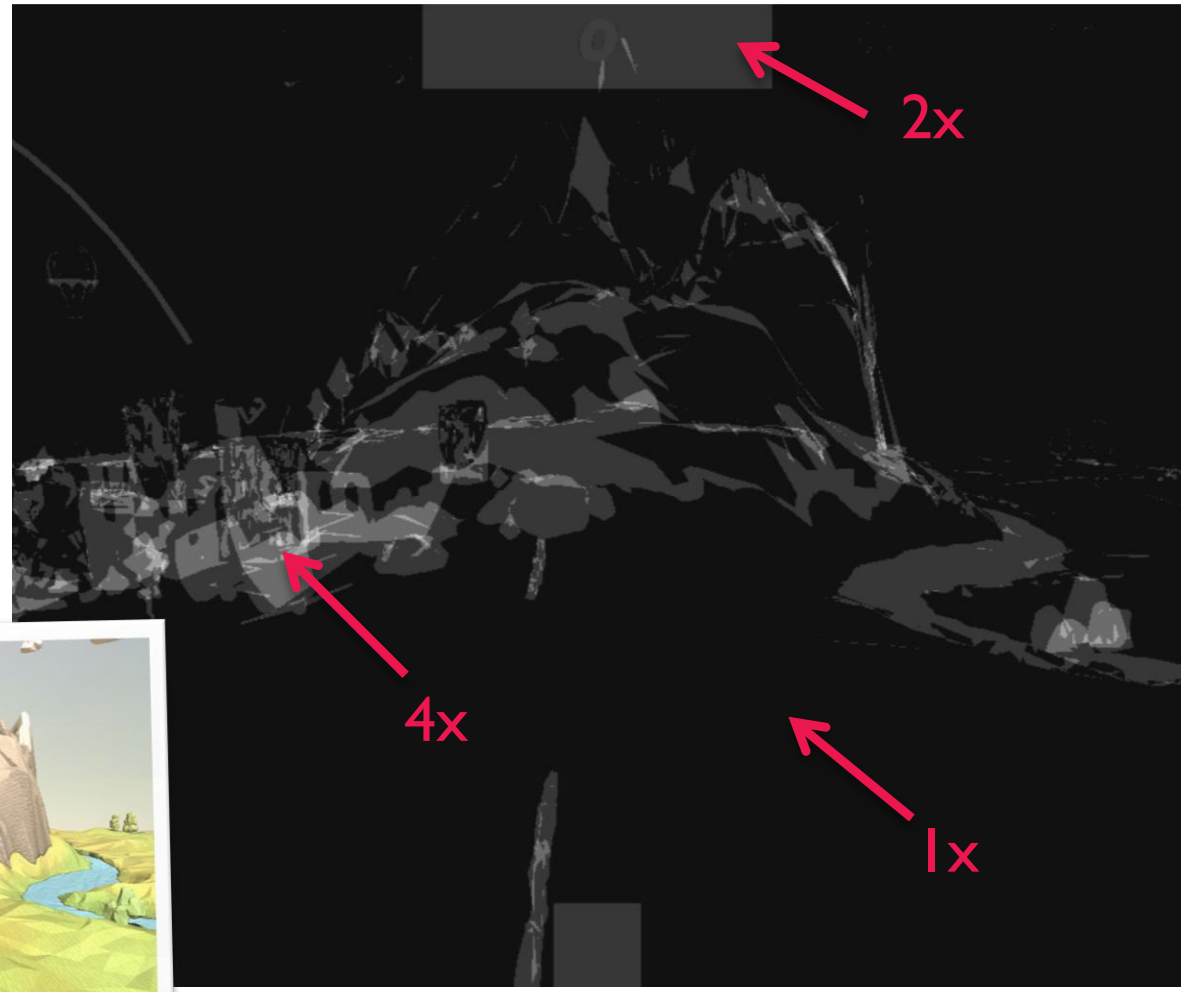
Frame Capture

ARM

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



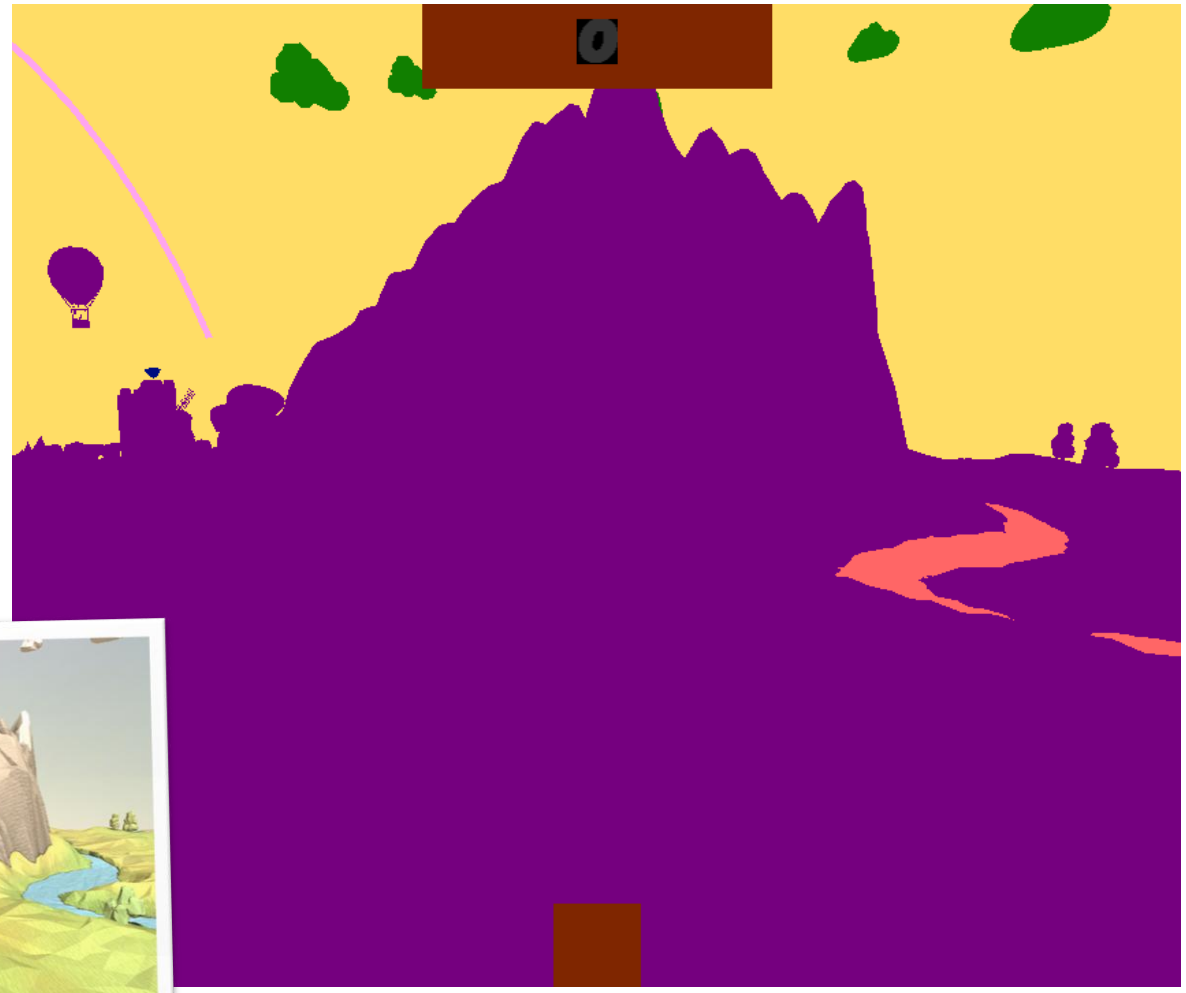


ARM

Shader Map and Fragment Count

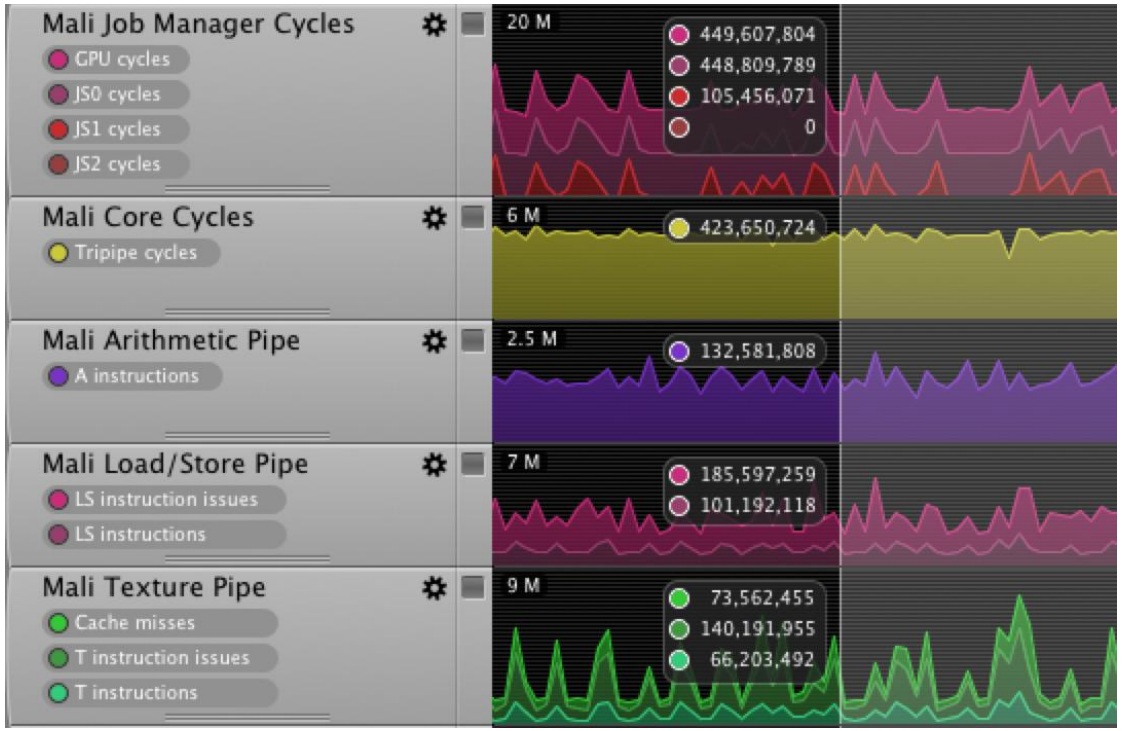
Vertex Shaders | Fragment Shaders | Textures

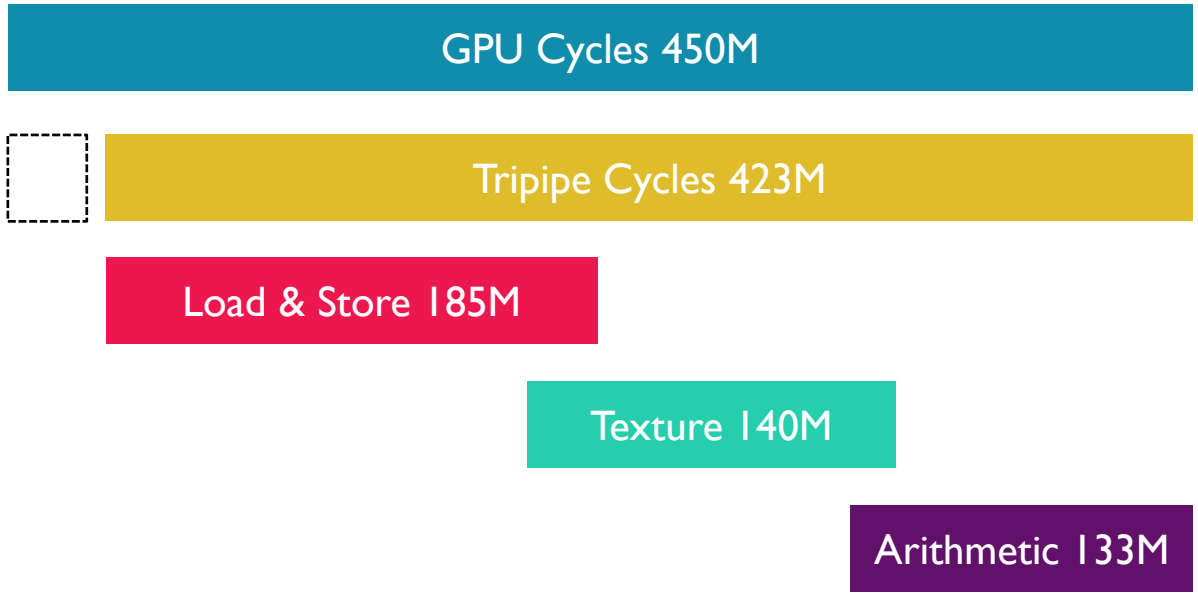
Program	Name	Instructions	Shortest path	Longest pa	Instances r	Total cycles
54	Shader 53	20	15	19	730781	12423277
60	Shader 59	18	13	17	326609	4899135
45	Shader 44	20	15	19	6358	108086
57	Shader 56	21	16	20	1389	25002
39	Shader 38	1	1	1	19840	19840
51	Shader 50	27	22	26	566	13584
36	Shader 35	7	7	7	1191	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



ARM

Inspect the Tripipe Counters



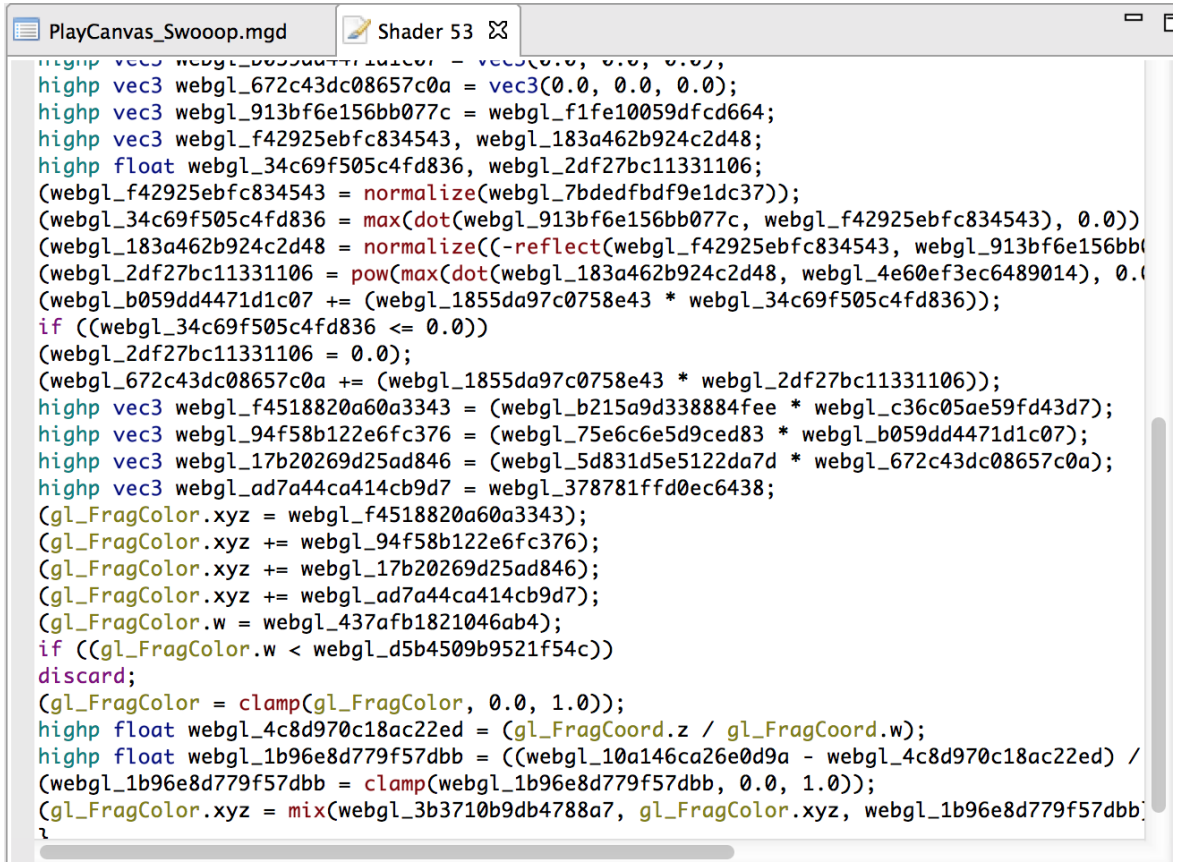


ARM

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/



ARM

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