

Performance Optimization and Debug Tools for mobile games with PlayCanvas

Jonathan Kirkham, *Senior Software Engineer, ARM*
Will Eastcott, *CEO, PlayCanvas*



Introduction

Jonathan Kirkham, ARM

- Worked with ARM technology and graphics at University
- Joined ARM in 2011 to work on 3D graphics
- Developing performance analysis tools and debuggers for the Mali GPUs

Agenda

1. Introduction to WebGL™ on mobile
 - Rendering Pipeline
2. PlayCanvas experience
 - WebGL Inspector
3. Performance analysis and debugging tools for WebGL
 - Generic optimization tips
4. 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™

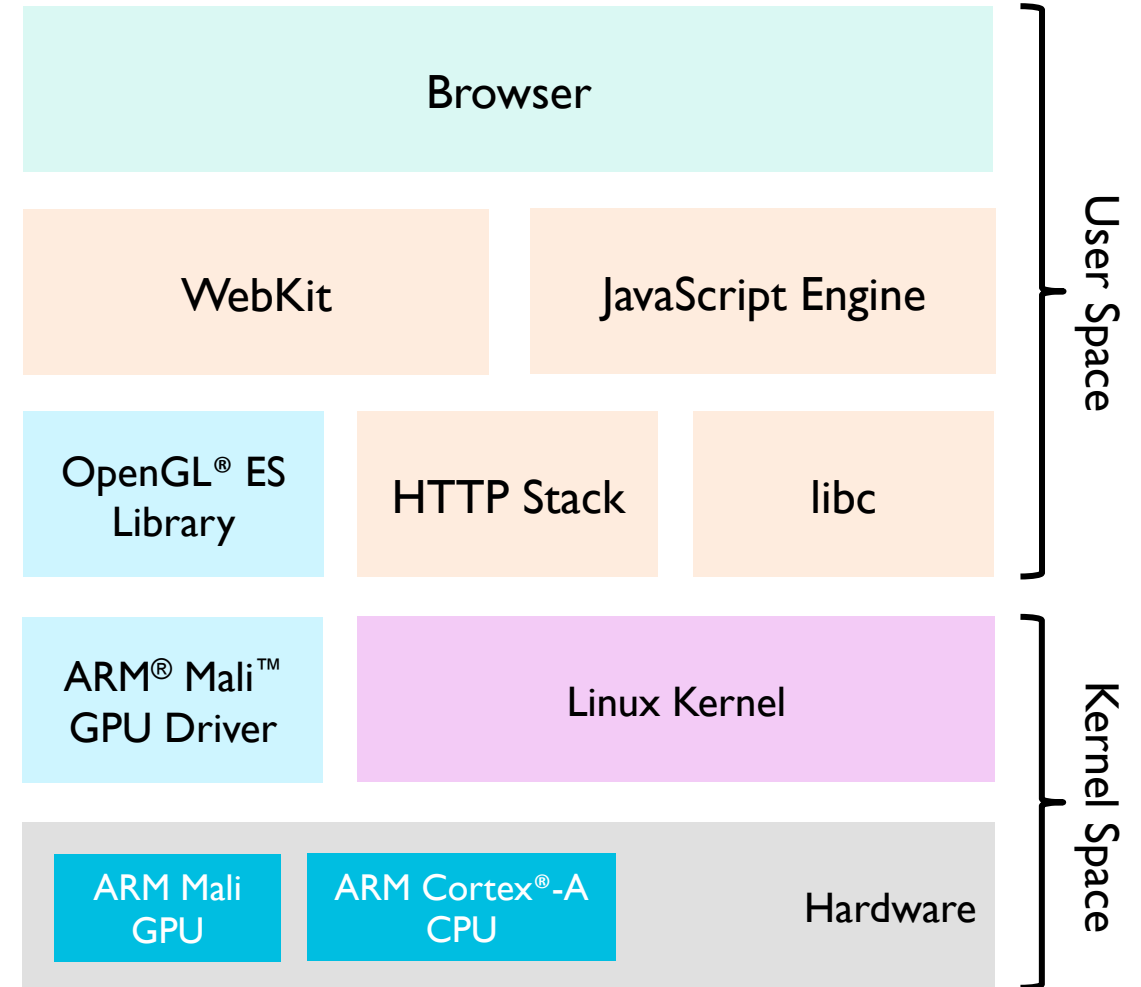
- How does it fit in a web browser?
 - You use JavaScript™ to control it.
 - Your JavaScript is embedded in HTML5 and uses its Canvas element to draw on.
- What do you need to start creating graphics?
 - Obtain WebGLRenderingContext object for a given HTMLCanvasElement.
 - It creates a drawing buffer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



Introducing Me...

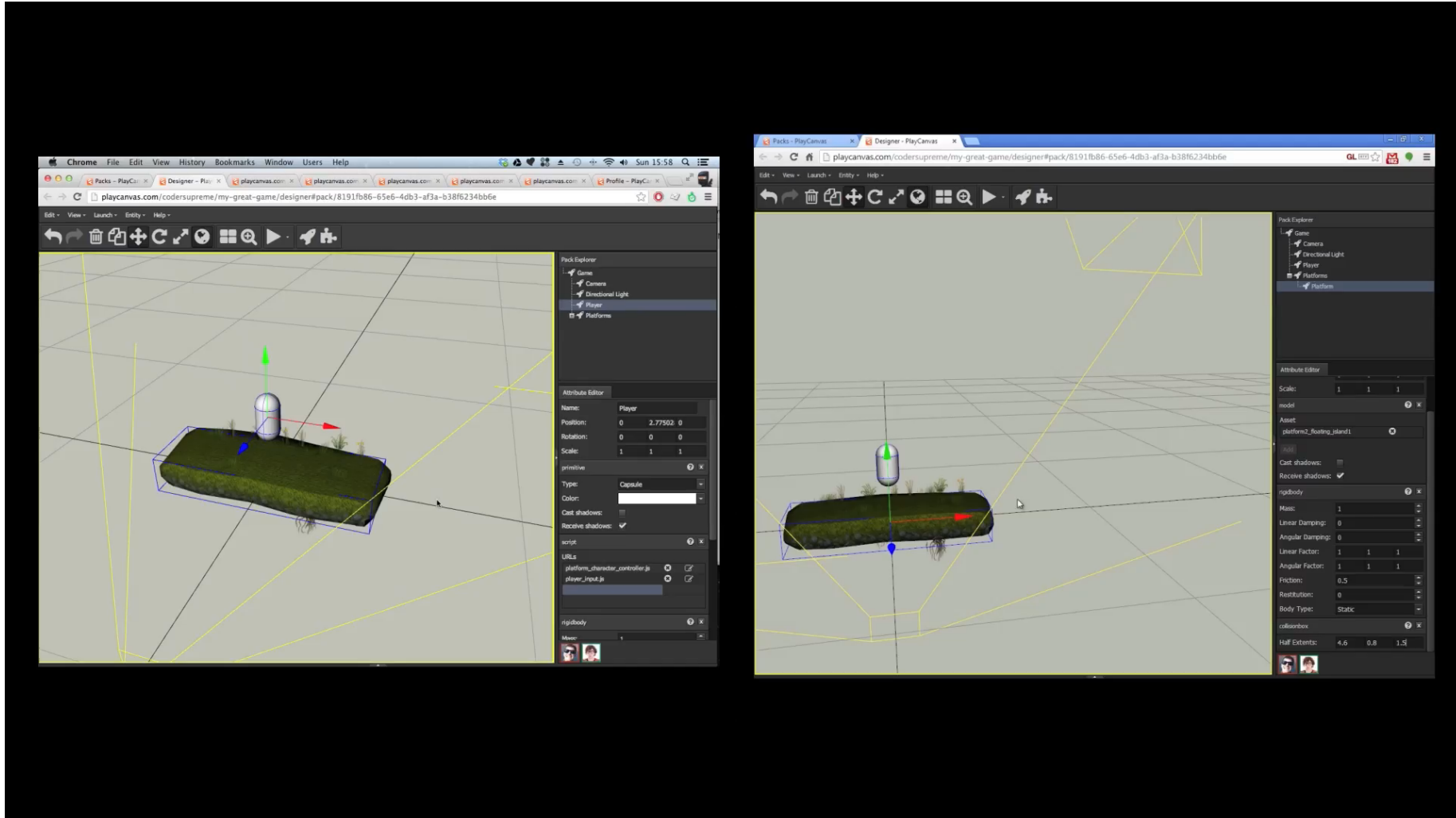


...and PlayCanvas



PlayCanvas is the world's easiest to use WebGL Game Engine. It's free, it's open source and it's backed by amazing developer tools.

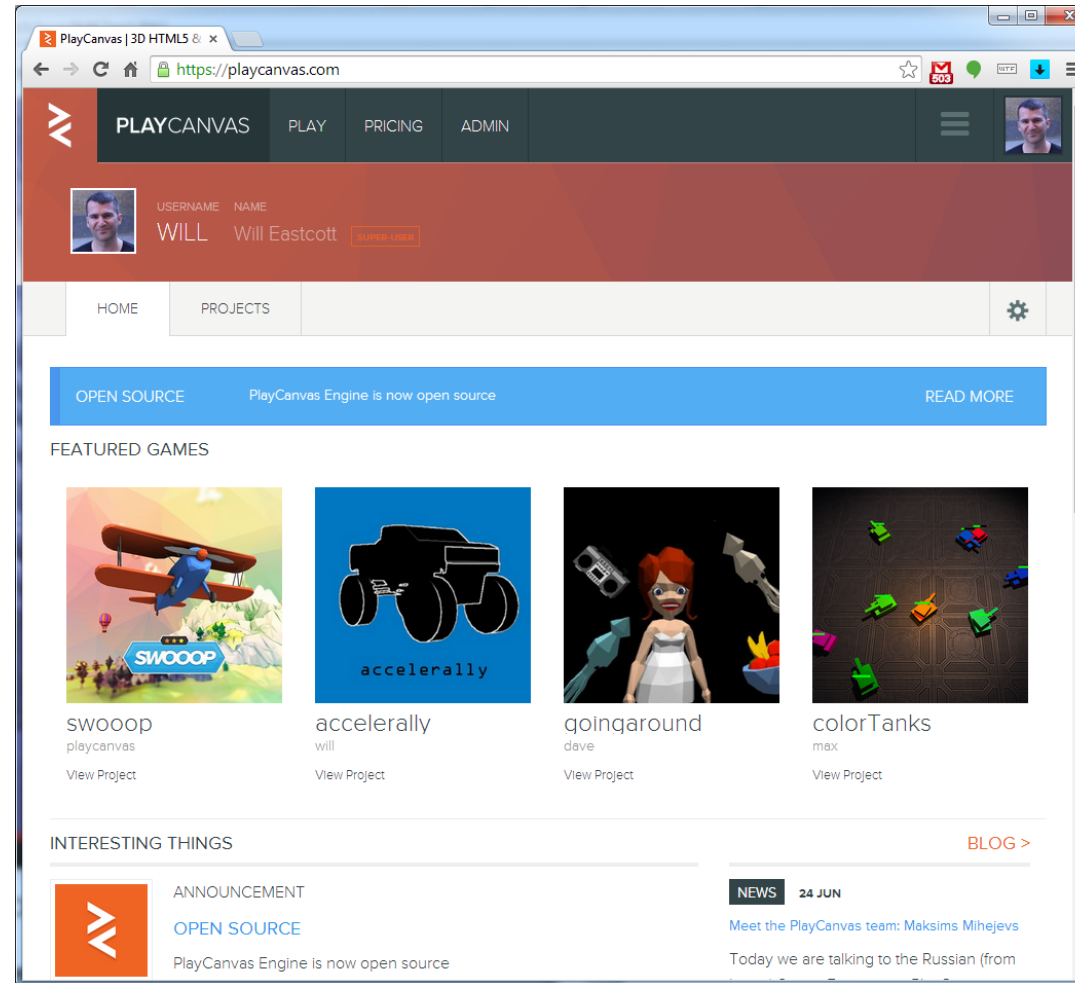
Google Docs for Games: Realtime Collaboration



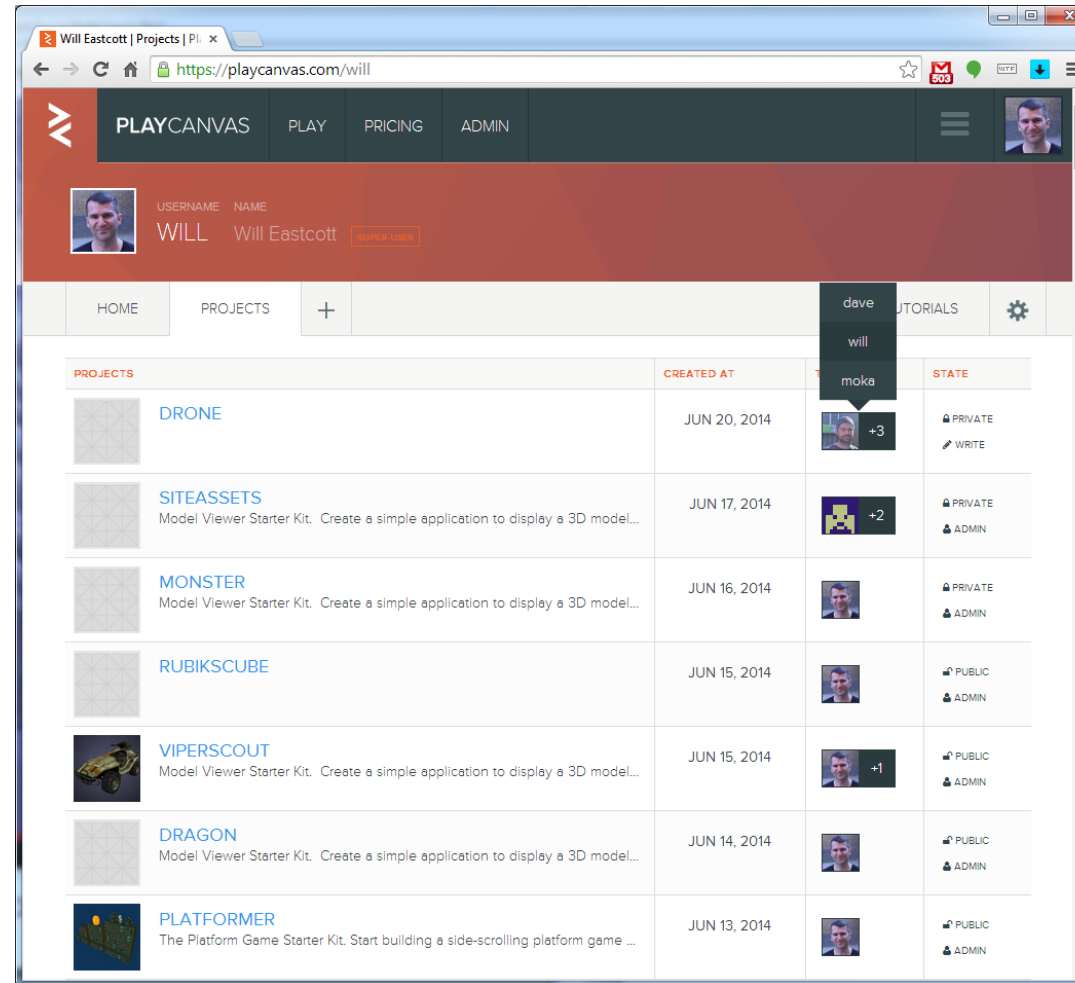
Game Development Goes Mobile on ARM










At Last: A Real Community for Game Dev



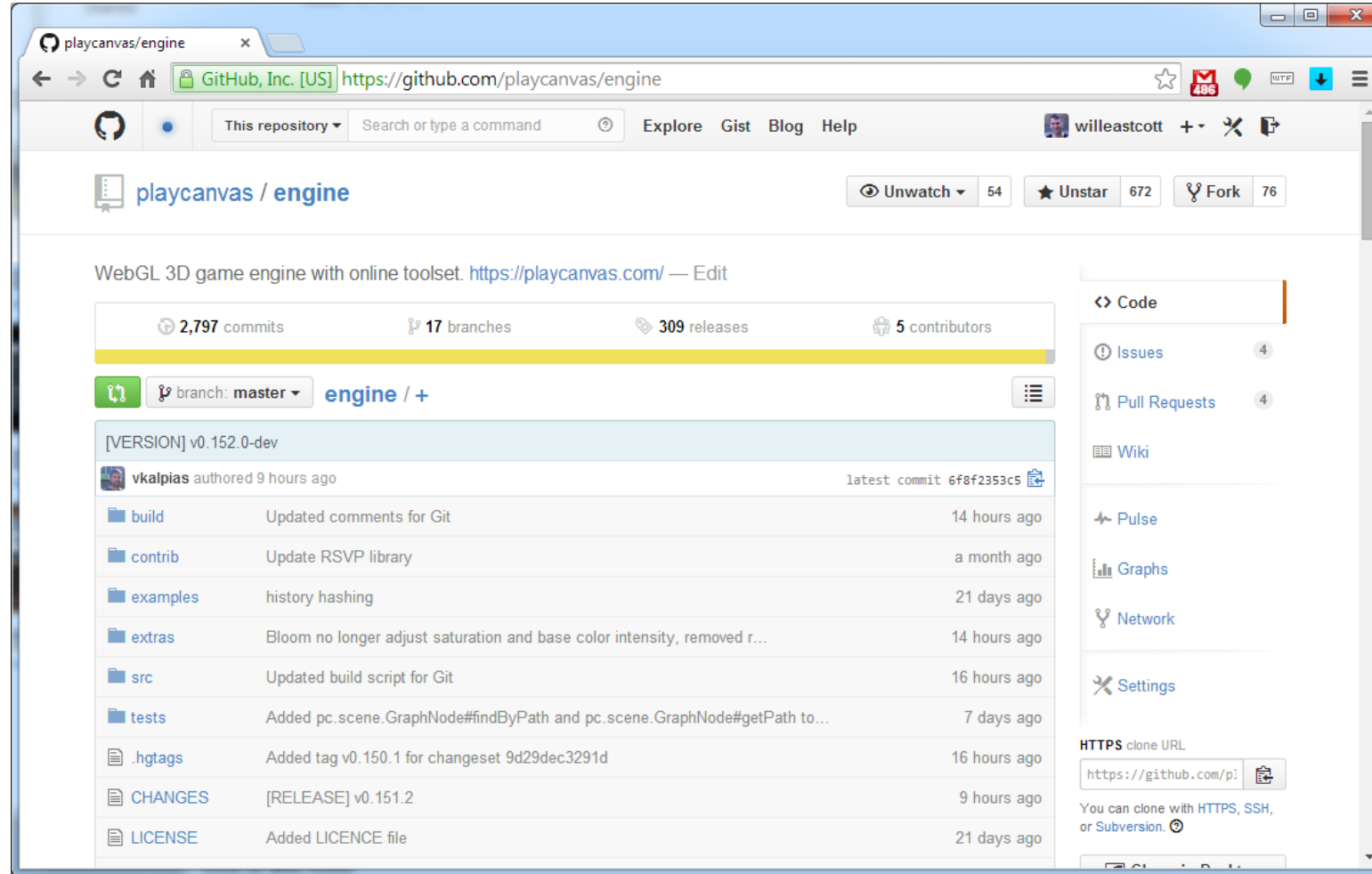
At Last: A Real Community for Game Dev



The screenshot shows a web browser window displaying the PlayCanvas website. The browser's address bar shows the URL <https://playcanvas.com/will>. The website's navigation bar includes links for PLAYCANVAS, PLAY, PRICING, and ADMIN. A user profile for 'WILL' (Will Eastcott) is visible, with a 'SUPER-USER' badge. Below the navigation bar, there are tabs for HOME, PROJECTS, and TUTORIALS. A dropdown menu is open over the PROJECTS tab, showing a list of users: dave, will, and moka. The main content area displays a table of projects with columns for PROJECTS, CREATED AT, and STATE. The projects listed are DRONE, SITEASSETS, MONSTER, RUBIKSCUBE, VIPERSCOUT, DRAGON, and PLATFORMER. Each project entry includes a thumbnail, a title, a description, a creation date, a user icon, and a state (PRIVATE or PUBLIC).

PROJECTS	CREATED AT	STATE
 DRONE	JUN 20, 2014	PRIVATE
 SITEASSETS Model Viewer Starter Kit. Create a simple application to display a 3D model...	JUN 17, 2014	PRIVATE
 MONSTER Model Viewer Starter Kit. Create a simple application to display a 3D model...	JUN 16, 2014	PRIVATE
 RUBIKSCUBE	JUN 15, 2014	PUBLIC
 VIPERSCOUT Model Viewer Starter Kit. Create a simple application to display a 3D model...	JUN 15, 2014	PUBLIC
 DRAGON Model Viewer Starter Kit. Create a simple application to display a 3D model...	JUN 14, 2014	PUBLIC
 PLATFORMER The Platform Game Starter Kit. Start building a side-scrolling platform game ...	JUN 13, 2014	PUBLIC

Open Sourced: <https://github.com/playcanvas/engine>



The screenshot shows the GitHub repository page for `playcanvas/engine`. The repository is described as a "WebGL 3D game engine with online toolset" and is available at <https://playcanvas.com/>. It has 2,797 commits, 17 branches, 309 releases, and 5 contributors. The current branch is `master`, and the repository is named `engine`. The commit history shows the latest commit by `vkalpias` 9 hours ago, with a commit hash of `6f8f2353c5`. The commit message is "[VERSION] v0.152.0-dev". The commit history table is as follows:

File	Commit Message	Time Ago
<code>build</code>	Updated comments for Git	14 hours ago
<code>contrib</code>	Update RSVP library	a month ago
<code>examples</code>	history hashing	21 days ago
<code>extras</code>	Bloom no longer adjust saturation and base color intensity, removed r...	14 hours ago
<code>src</code>	Updated build script for Git	16 hours ago
<code>tests</code>	Added pc.scene.GraphNode#findByPath and pc.scene.GraphNode#getPath to...	7 days ago
<code>.hgtags</code>	Added tag v0.150.1 for changeset 9d29dec3291d	16 hours ago
<code>CHANGES</code>	[RELEASE] v0.151.2	9 hours ago
<code>LICENSE</code>	Added LICENCE file	21 days ago

The right sidebar contains navigation links for `Code`, `Issues` (4), `Pull Requests` (4), `Wiki`, `Pulse`, `Graphs`, `Network`, and `Settings`. The `HTTPS clone URL` is `https://github.com/p...`. The page also indicates that the repository can be cloned using HTTPS, SSH, or Subversion.

The Building of a WebGL Game: SWOOOP



What We Did Right

- We didn't use physics
- We didn't use realtime shadows
- We didn't use post effects
- We adopted an art style which only required low res texturing
- We kept the number of draw calls below 150
- We added visual flare with cheap GPU based effects like particles and UV scrolling

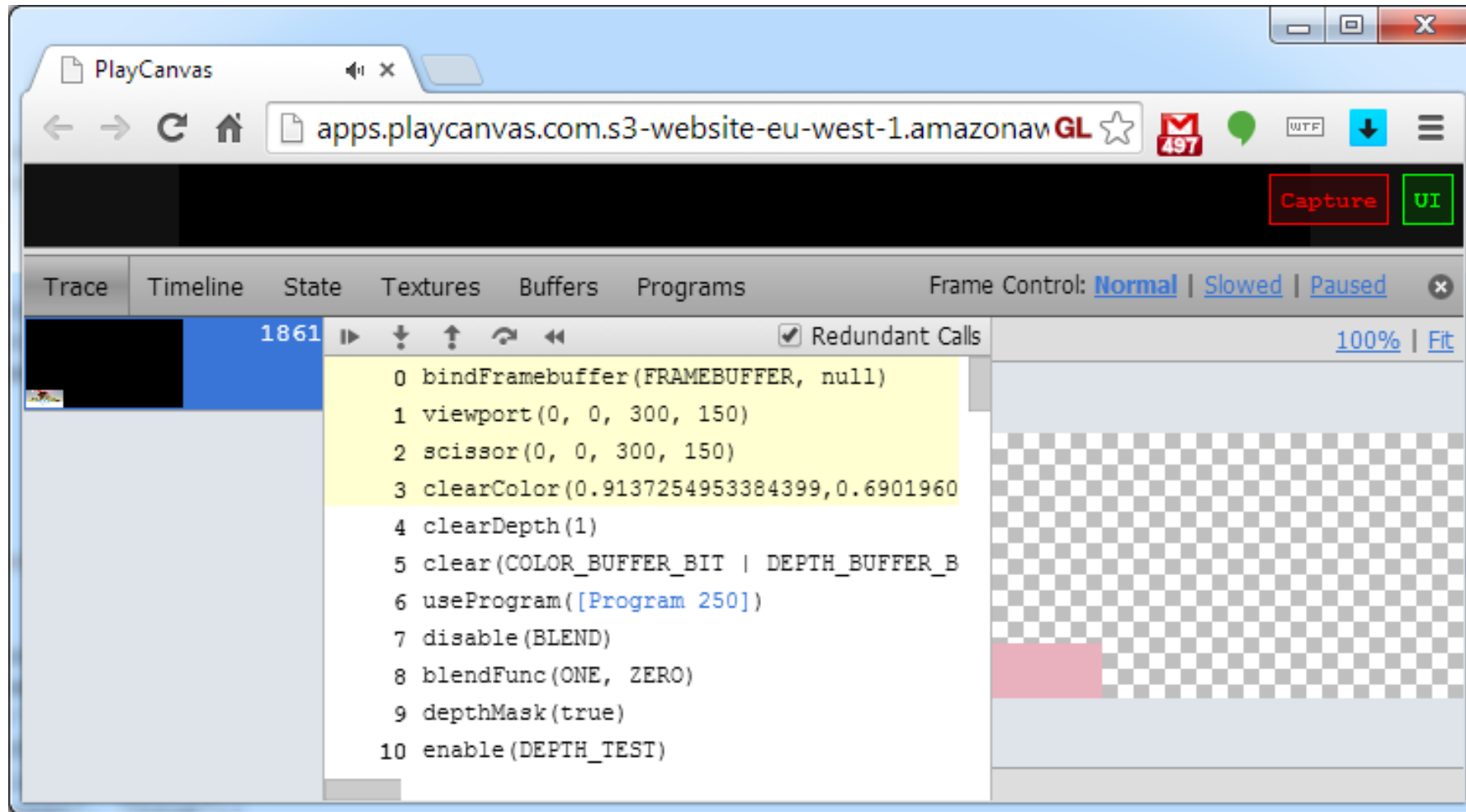
What We Did Wrong

- We adopted an art style which generated a lot of vertex data
- Keeping draw calls low generated more vertex data
- We used realtime lighting on the environment
- Each gem was a separate draw call

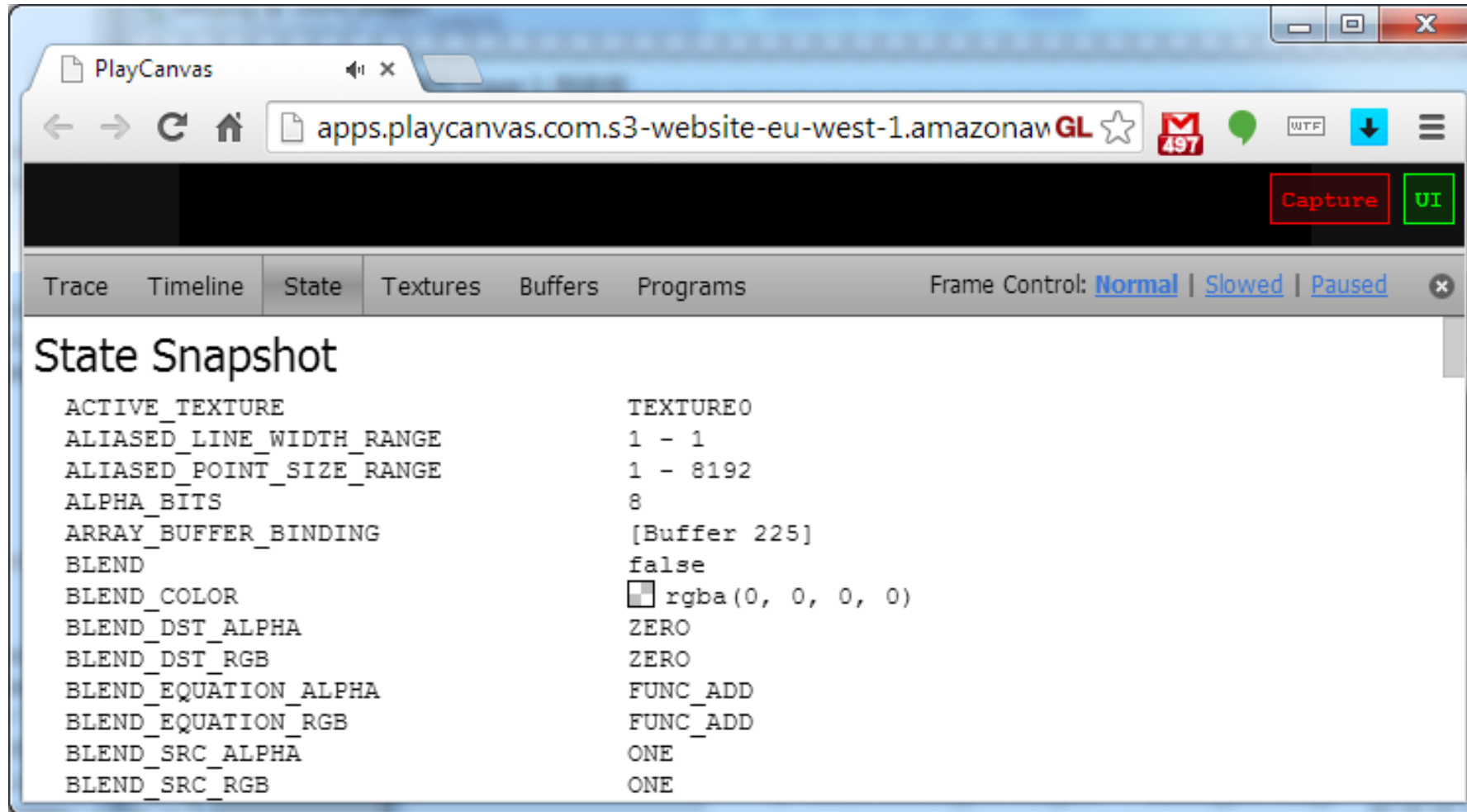
Optimizing Your WebGL Code: Options

- Learn from the Open Source community
- In-browser Developer Tools
- GLSL Optimizer (<https://github.com/aras-p/glsl-optimizer>)
- WebGL Inspector (<http://benvanik.github.io/WebGL-Inspector/>)

WebGL Inspector: Function Tracing



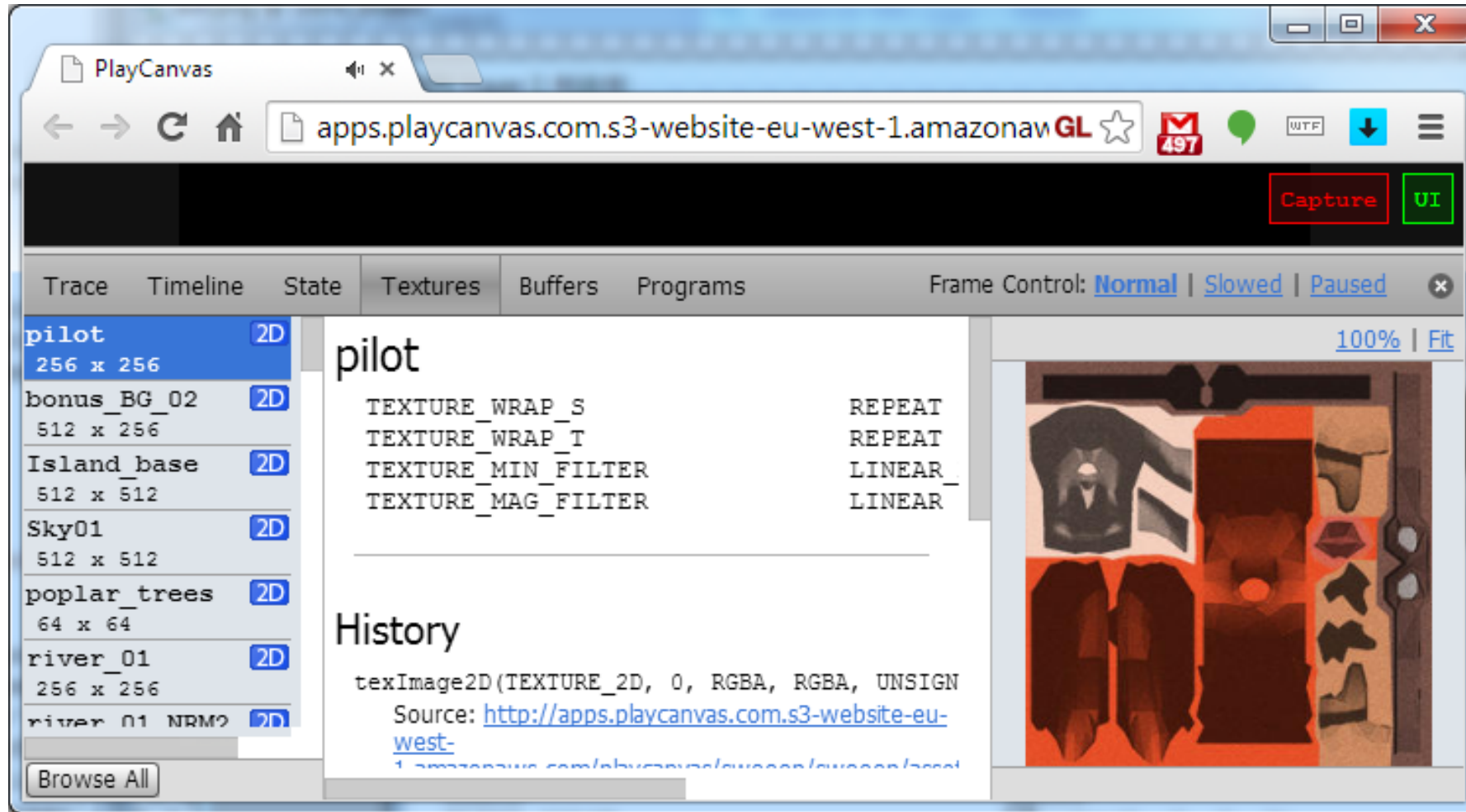
WebGL Inspector: Render State



The screenshot shows the WebGL Inspector interface. The browser tab is titled "PlayCanvas" and the address bar shows "apps.playcanvas.com.s3-website-eu-west-1.amazonaws.com". The interface includes a "Capture" button (highlighted in red) and a "UI" button (highlighted in green). The "State" panel is selected, displaying a "State Snapshot" with the following render state parameters:

ACTIVE_TEXTURE	TEXTURE0
ALIASED_LINE_WIDTH_RANGE	1 - 1
ALIASED_POINT_SIZE_RANGE	1 - 8192
ALPHA_BITS	8
ARRAY_BUFFER_BINDING	[Buffer 225]
BLEND	false
BLEND_COLOR	<input type="checkbox"/> rgba(0, 0, 0, 0)
BLEND_DST_ALPHA	ZERO
BLEND_DST_RGB	ZERO
BLEND_EQUATION_ALPHA	FUNC_ADD
BLEND_EQUATION_RGB	FUNC_ADD
BLEND_SRC_ALPHA	ONE
BLEND_SRC_RGB	ONE

WebGL Inspector: Textures



The screenshot shows the WebGL Inspector interface for a browser window titled "PlayCanvas". The address bar displays the URL "apps.playcanvas.com.s3-website-eu-west-1.amazonaws.com". The interface includes a "Capture" button (highlighted in red) and a "UI" button (highlighted in green). The main panel is divided into several sections:

- Texture List:** A list of textures on the left, with "pilot" selected. Other textures include "bonus_BG_02", "Island_base", "Sky01", "poplar_trees", "river_01", and "river_01 NRM?".
- Texture Details:** The "pilot" texture is shown with dimensions "256 x 256".
- Texture Properties:** A table of properties for the "pilot" texture:

TEXTURE_WRAP_S	REPEAT
TEXTURE_WRAP_T	REPEAT
TEXTURE_MIN_FILTER	LINEAR
TEXTURE_MAG_FILTER	LINEAR
- History:** A log entry showing "texImage2D(TEXTURE_2D, 0, RGBA, RGBA, UNSIGN)" with the source URL "http://apps.playcanvas.com.s3-website-eu-west-1.amazonaws.com/...".
- Texture Preview:** A small image showing a 2D texture of a pilot's uniform.

WebGL Inspector: Vertex and Index Buffers

The screenshot shows the WebGL Inspector interface for a browser window titled "PlayCanvas". The address bar shows the URL "apps.playcanvas.com.s3-website-eu-west-1.amazonaws.com". The interface includes a "Capture" button (highlighted in red) and a "UI" button (highlighted in green). Below the browser window, there are tabs for "Trace", "Timeline", "State", "Textures", "Buffers", and "Programs". The "Buffers" tab is selected, and the "Frame Control" is set to "Normal".

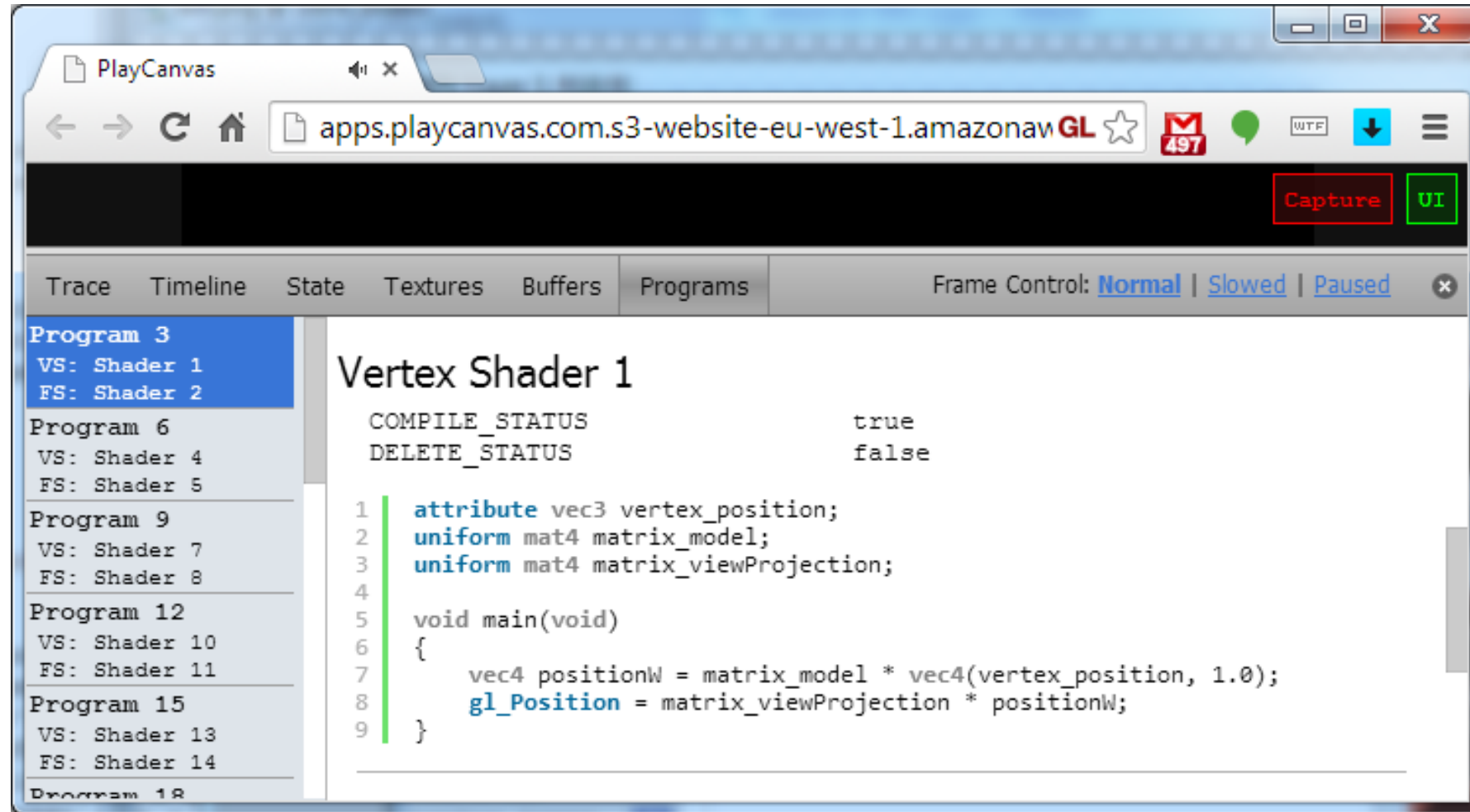
The "Usage in frame 1861" section shows the following log entries:

```
612 bindBuffer(ARRAY_BUFFER, [Buffer 61])
631 bindBuffer(ARRAY_BUFFER, [Buffer 61])
```

The "Contents" section displays a table of vertex data for Buffer 61:

0	-22.741600036621094	-59.002498626708984	76.61519622802734	-0.46354
1	-57.39680099487305	-39.403900146484375	76.61519622802734	-0.46354
2	-89.04440307617188	-67.98719787597656	-36.58169937133789	-0.46354
3	-89.04440307617188	-67.98719787597656	-36.58169937133789	-0.290937
4	-20.709699630737305	-89.47689819335938	-36.58169937133789	-0.290937

WebGL Inspector: Shader Code



The screenshot shows the WebGL Inspector interface. The browser tab is titled "PlayCanvas" and the address bar shows "apps.playcanvas.com.s3-website-eu-west-1.amazonaws.com". The interface includes a "Capture" button (highlighted in red) and a "UI" button (highlighted in green). The "Programs" tab is selected, showing a list of programs on the left and the source code for "Vertex Shader 1" in the main area. The code is as follows:

```
Vertex Shader 1
COMPILE_STATUS          true
DELETE_STATUS           false

1  attribute vec3 vertex_position;
2  uniform mat4 matrix_model;
3  uniform mat4 matrix_viewProjection;
4
5  void main(void)
6  {
7      vec4 positionW = matrix_model * vec4(vertex_position, 1.0);
8      gl_Position = matrix_viewProjection * positionW;
9  }
```

Understanding the GPU is Key

- Easy to optimize your graphics pipeline on the CPU
- Harder to know how to optimize on the GPU
- Use ARM tools to get special insight into how your graphics data is being processed

Importance of Analysis & Debug

■ Mobile Platforms

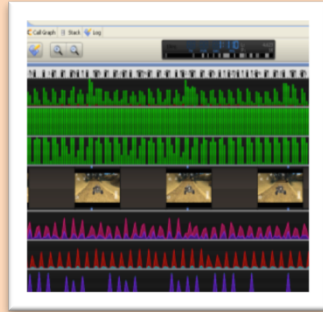
- Expectation of amazing console-like graphics and playing experience
- Screen resolution beyond HD
- Limited power budget

■ Solution

- ARM® Cortex® CPUs and Mali™ GPUs are designed for low power whilst providing innovative features to keep up performance
- Software developers can be “smart” when developing apps
- Good tools can do the heavy lifting

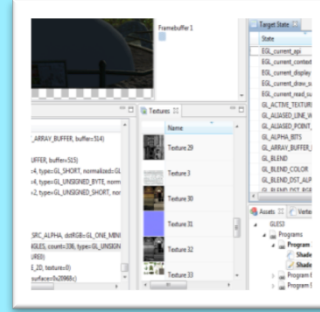


Performance Analysis & Debug



ARM® DS-5 Streamline Performance Analyzer

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



ARM 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 r3p1-04rel1
Mali-T600 series driver version r3p0-04rel0

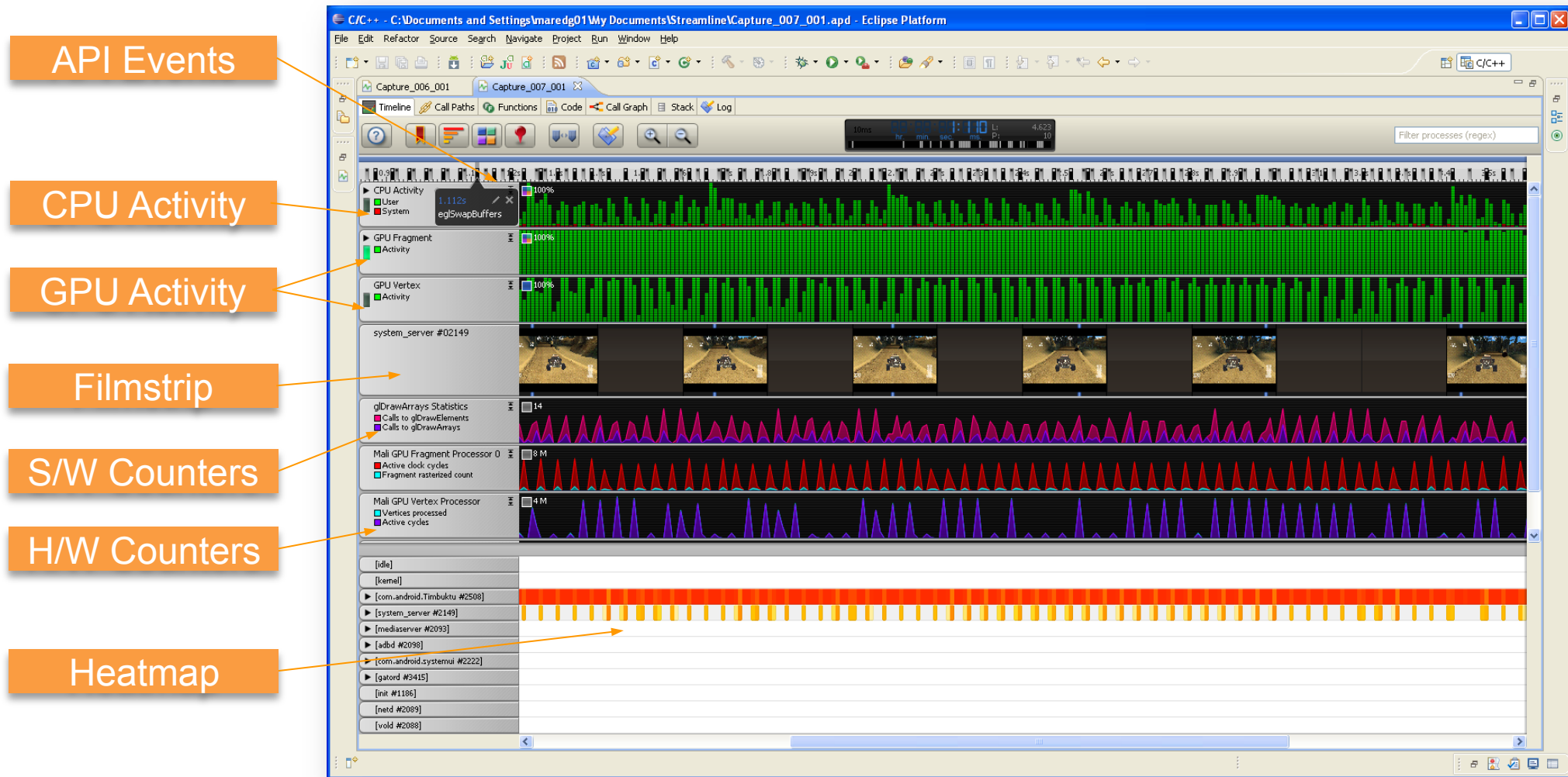
Usage: malisc.exe [options] [-o outfile] --core=core
       -DNAME [-VALUE]  Define name as a
       --vert           Process shader as a
       --frag           Process shader as a
       -v, --verbose    Print verbose inform
       -o outfile       write output to outf
       --core=core     Target specified gra
Supported cores are:
Mali-200
Mali-300
Mali-400
Mali-450
Mali-T600
Mali-T650
Mali-T600 (Mali-T650
r3p0
r3p0-1sdev0

-r rXpY, --revision=rXpY Target hardware rele
r3p0
r3p0-1sdev0
```

Offline Compilers

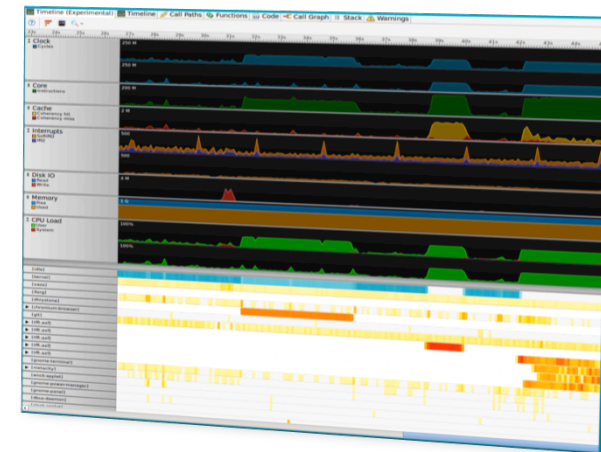
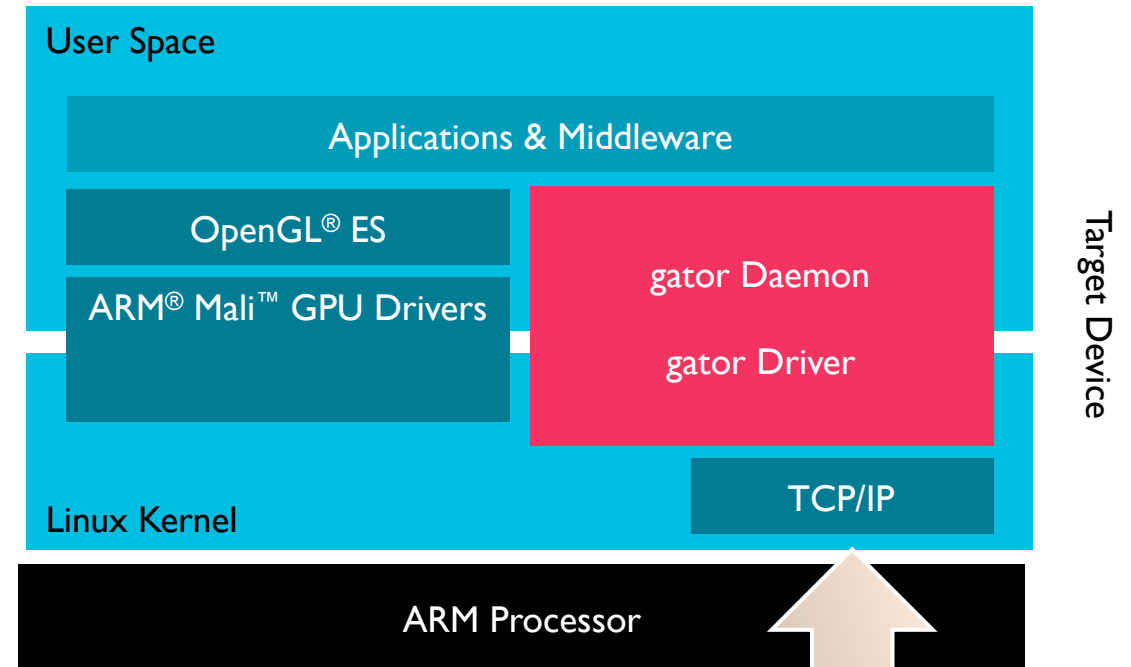
- Understand complexity of GLSL shaders and CL kernels
- Support for ARM Mali-4xx and Mali-T6xx GPU families

ARM® DS-5 Streamline Performance Analyzer



The Basics

- **Software based solution**
 - ICE/trace units not required
 - Support for Linux kernel 2.6.32+ on target
 - Eclipse plug-in or command line
- **Lightweight sample profiling**
 - Time- or event*-based sampling
 - Process to C/C++ source code profiler
 - Low probe effect; <5% typically
- **Multiple data sources**
 - CPU, GPU and Interconnect hardware counters
 - Software counters and kernel tracepoints
 - User defined counters and instrumented code
 - Power/energy measurements



ARM

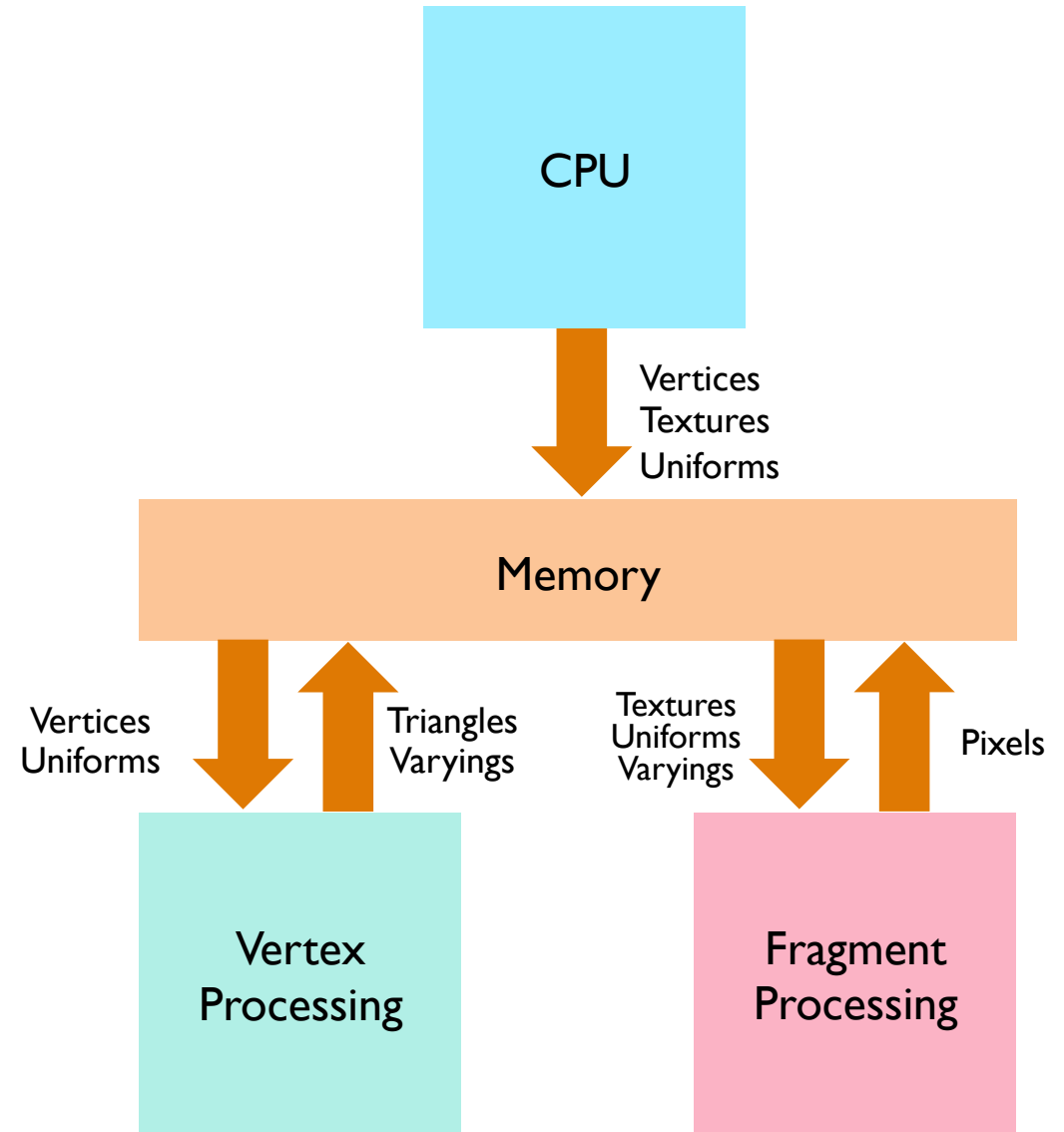
ARM® Mali™ Graphics Debugger

The screenshot shows the ARM Mali Graphics Debugger interface with several key components highlighted by callouts:

- Frame Outline:** A tree view on the left showing a list of frames (e.g., Frame 256, 257, 258, etc.) with their respective draw and vertex counts.
- Assets:** A central panel displaying a 3D scene (a mountain landscape) with a coordinate overlay (1254,7) [209, 223, 117].
- Statistics:** A panel on the right providing performance metrics such as 'Total number of API function calls' (171664) and 'Average vert/frame' (33069.02).
- Framebuffer / Render Targets:** A panel below the statistics showing the state of various render targets, including 'GL_COLOR_CLEAR_VALUE' and 'GL_DEPTH_BITS'.
- API Trace:** A large central panel displaying a detailed list of API calls, such as 'glDrawElements' and 'glVertexAttribPointer', with their parameters.
- Dynamic Help:** A panel at the bottom left showing a list of messages and their counts, such as 'Offset is beyond the end of the buffer' (12).
- Shader View:** A panel at the bottom right showing a list of textures with columns for Name, Size, Format, and Type.

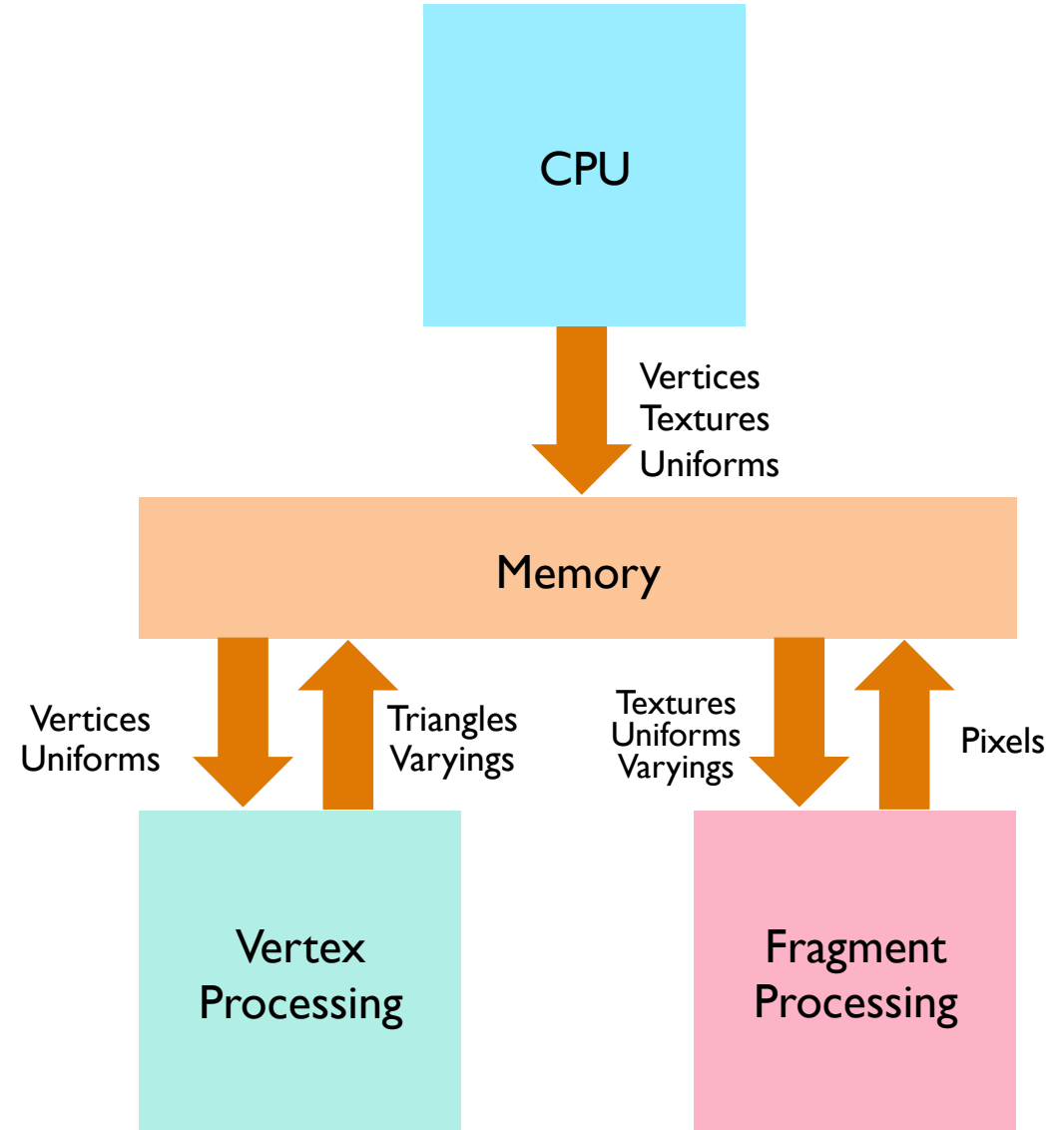
Main Bottlenecks (I)

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



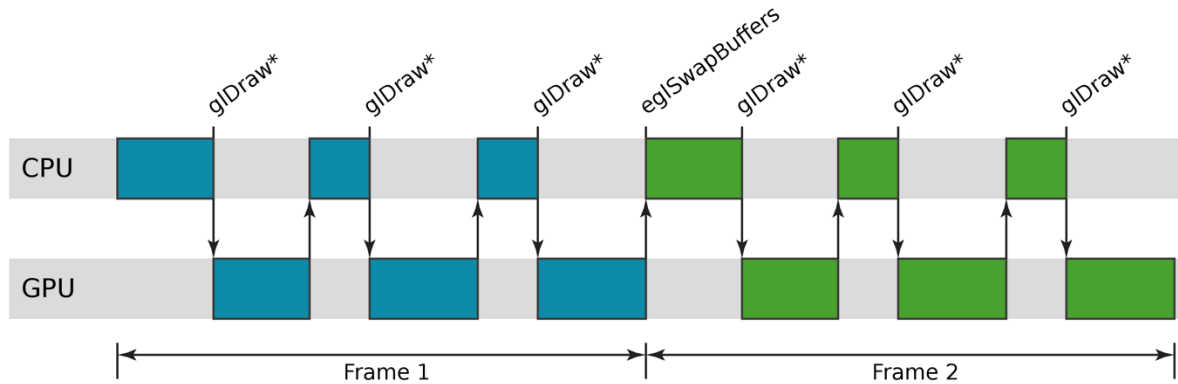
Main Bottlenecks (2)

- **CPU**
 - Too many draw calls
 - Complex physics
- **Vertex processing**
 - Too many vertices
 - Too much computation per vertex
- **Fragment processing**
 - Too many fragments, overdraw
 - Too much computation per fragment
- **Bandwidth**
 - Big and uncompressed textures
 - High resolution framebuffer



Frame Rendering Time

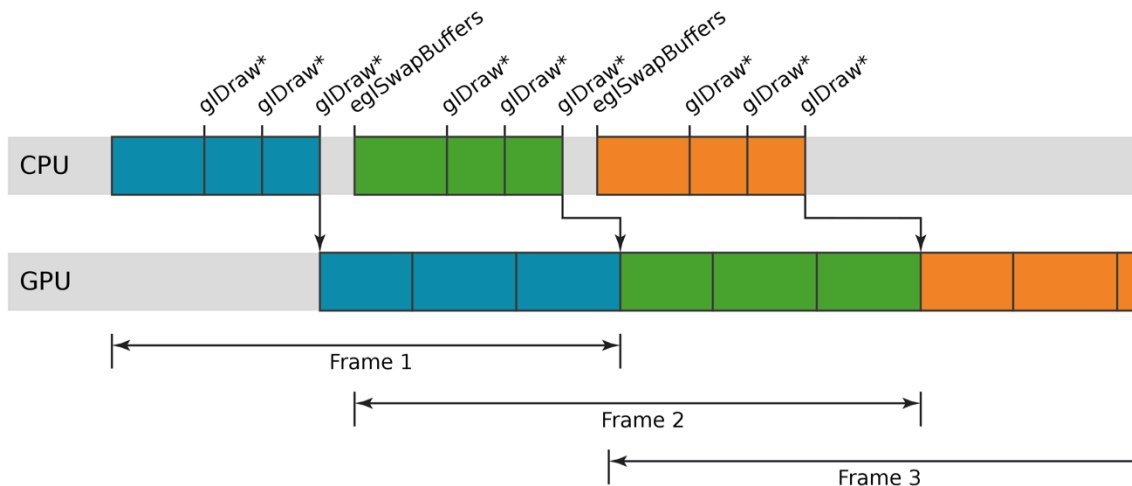
Synchronous Rendering



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

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

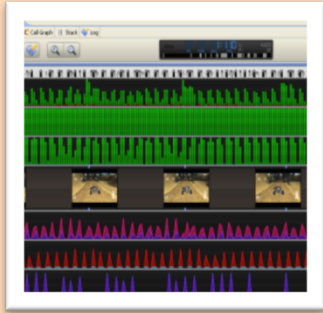
Deferred Rendering



```
// 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;
```

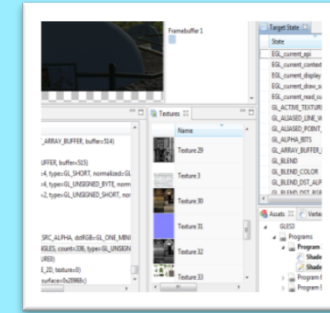
Workflow



ARM® DS-5 Streamline Performance Analyzer

What kind of problem do we have?

Detailed analysis



ARM Mali Graphics Debugger

What's causing the problem?

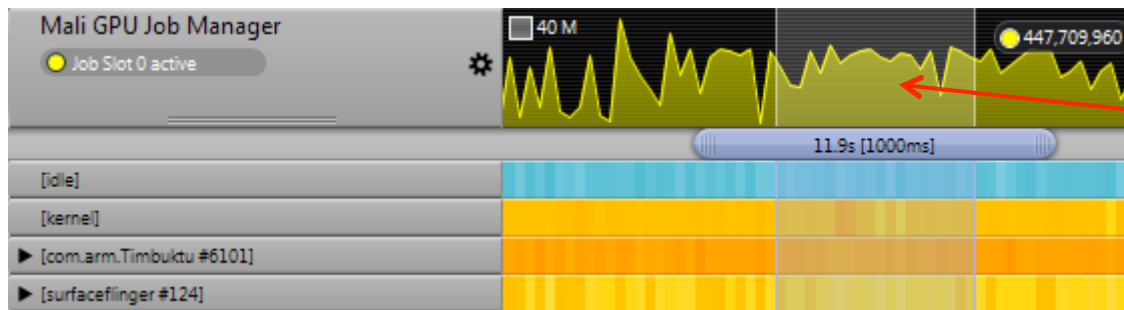
How can I fix it?

Validate

Fragment Bound

ARM DS-5 Streamline: Fragment Bound

- Involves just 1 counter and the frequency of the GPU
 - Job Slot 0 Active

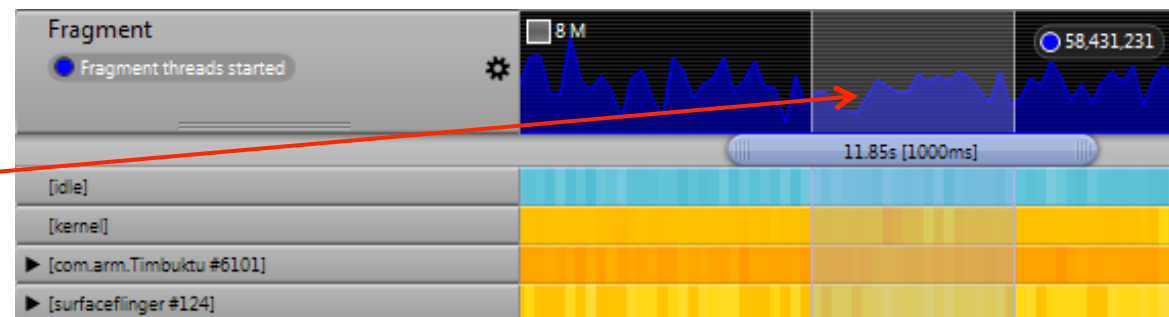


Fragment Percentage = (Job Slot 0 active / Frequency) * 100

Fragment Percentage = 84%

Overdraw = Fragment Threads Started * Number of Cores / Resolution * FPS

Overdraw = 3.9



Fragment Bound

- Resolution too high or too many effects or cycles in the shader
 - Every light and effect that you add will add to the number of cycles your shader will take
 - If you decide to run your app at native resolution be careful

Nexus 10 Native Resolution

- $2560 \times 1600 = 4,096,000$ pixels

Quad Core GPU 533Mhz

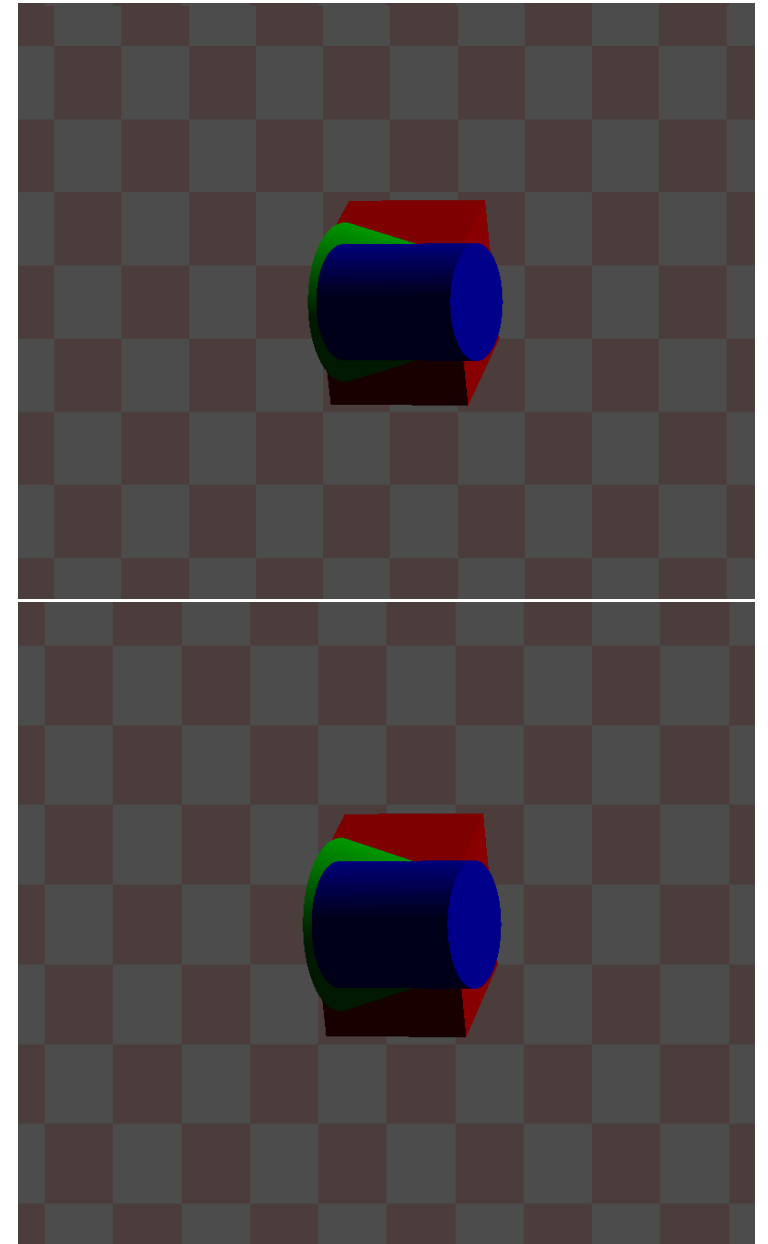
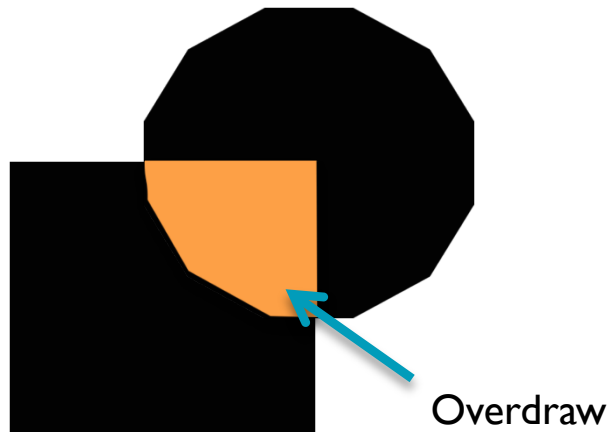
- 520 Cycles per pixel Approx.

Targeting 30 FPS

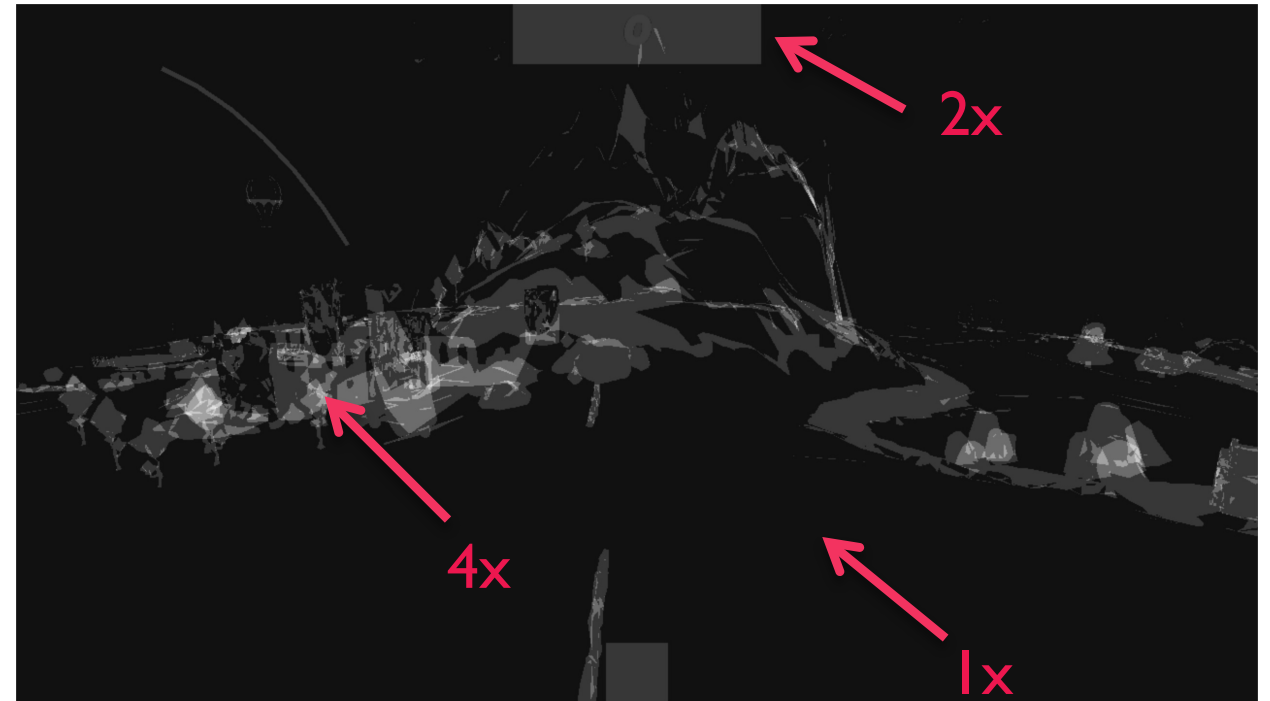
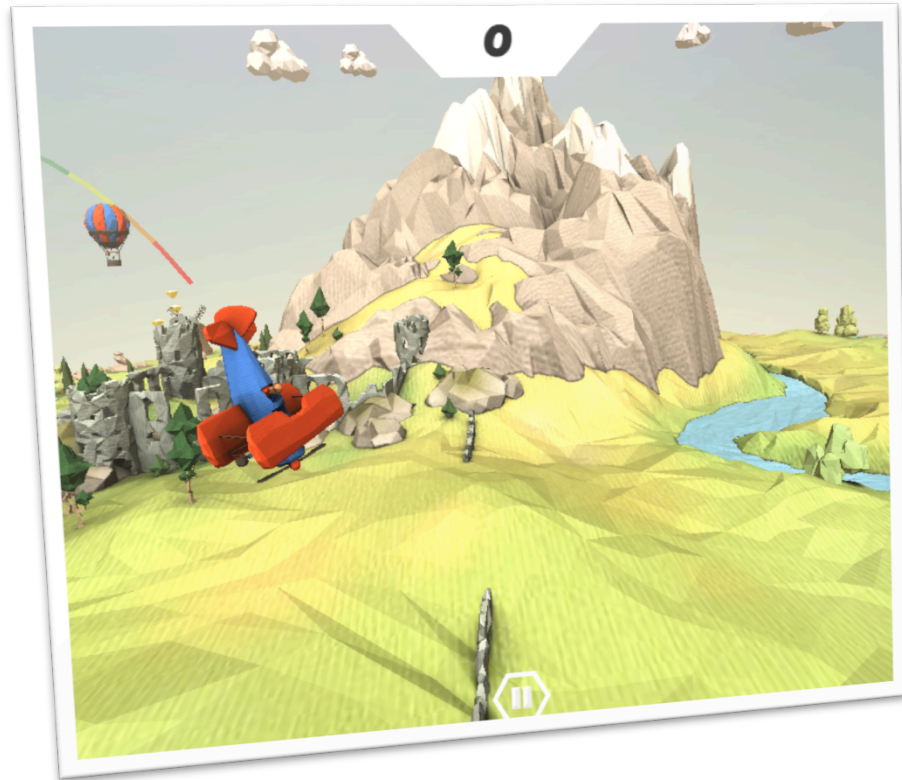
- 17 Cycles in your shader

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
- Limiting the amount of transparency in the scene can help



ARM Mali Graphics Debugger: Overdraw

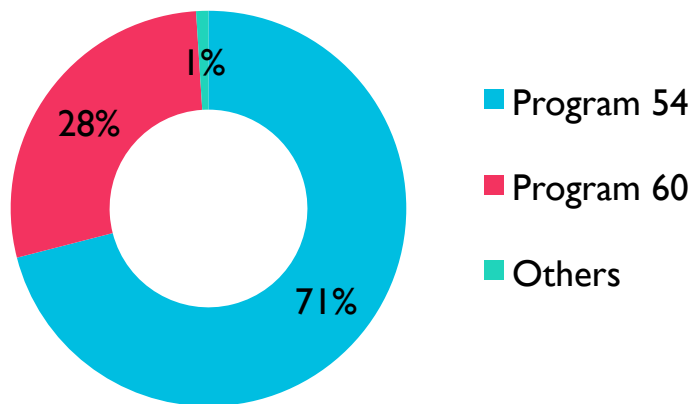


ARM Mali Graphics Debugger: Shader Map & Fragment Count

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
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



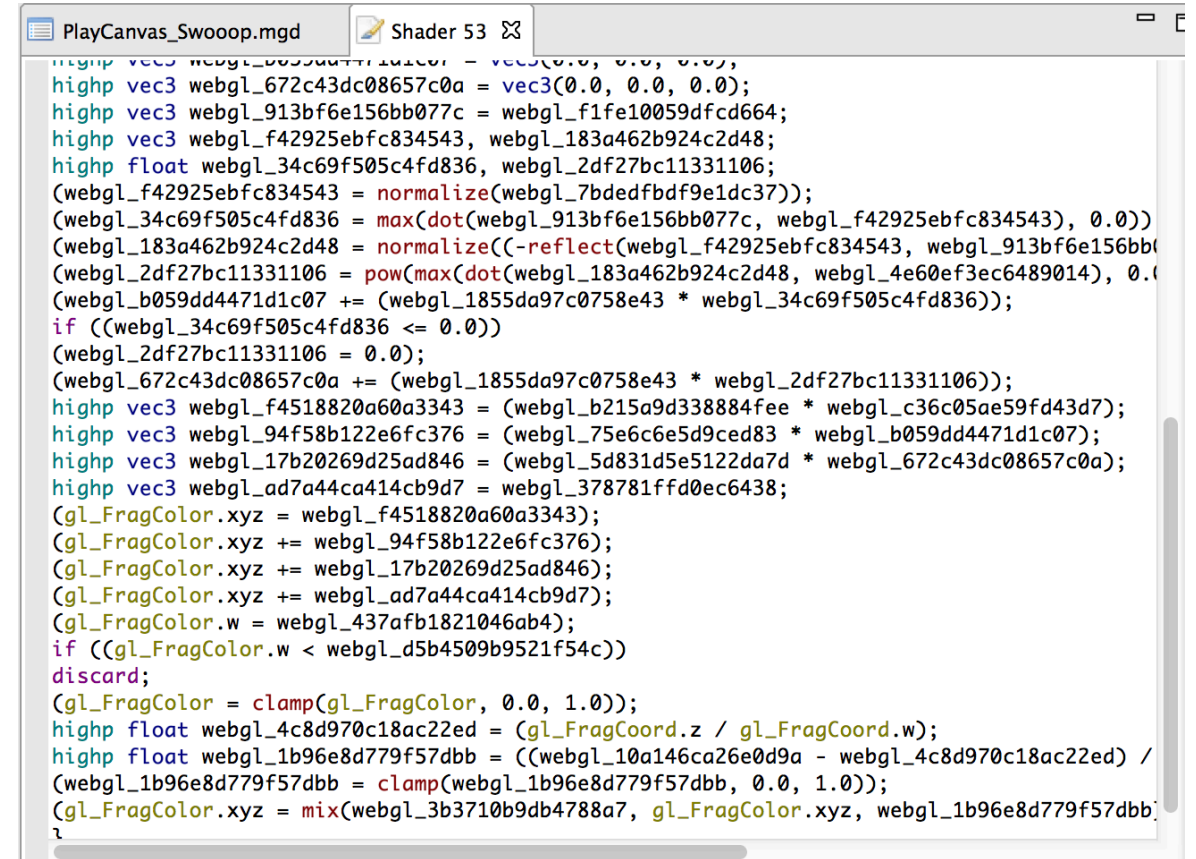
Total Cycles Per Program



Shader Optimization

- Depending on the arithmetic workload, 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);
```

General Tips

Fragment Bound

- 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)
- Drawing your objects **front to back** instead of back to front reduces overdraw
- Reduce the amount of transparency in the scene

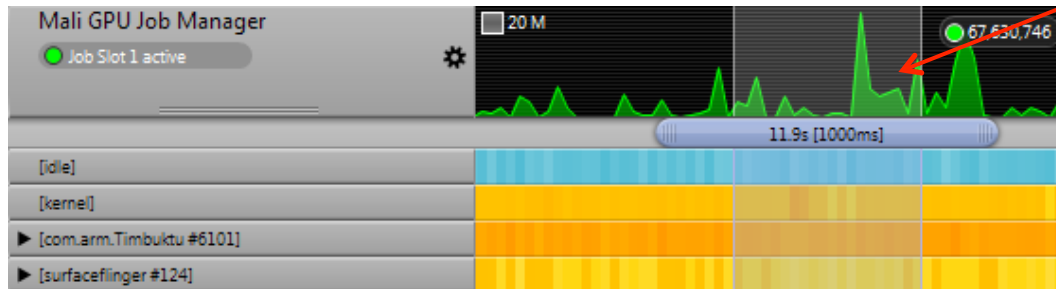


Vertex Bound

ARM DS-5 Streamline: Vertex Bound

- Involves just I counter and the frequency of the GPU
 - Job Slot I Active

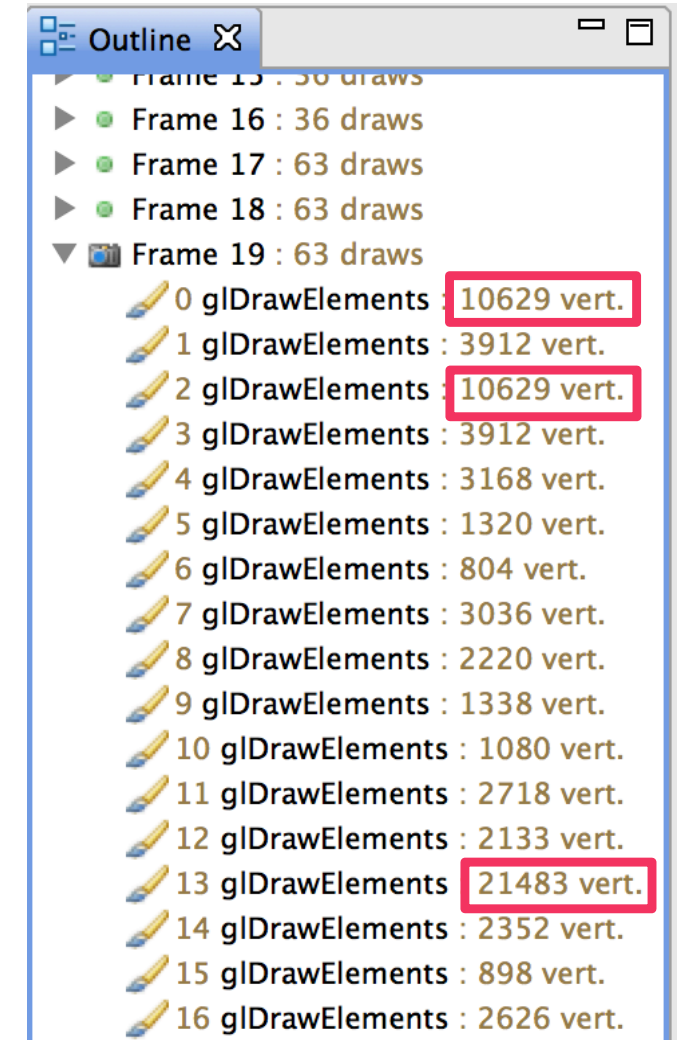
Vertex Percentage = (Job Slot I active / Frequency) * 100



Vertex Percentage = 13%

ARM Mali Graphics Debugger: Vertices Count

- Analyze the trace in Mali Graphics Debugger
- Find the draw calls with a high number of vertices
- Shader Statistics
 - Find the vertex shaders with a high number of instructions



ARM Mali Graphics Debugger: Frame Capture

Draw 001, total vertices: 15402



General Tips

Vertex Bound

- Get your artist to remove unnecessary vertices
- LOD switching
 - Only objects near the camera need to be in high detail
- Use culling
- Too many cycles in the vertex shader

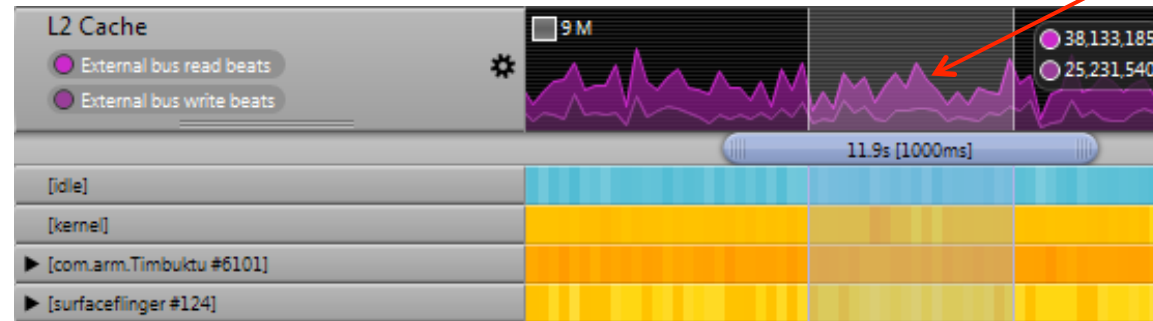


Bandwidth Bound

ARM DS-5 Streamline: Bandwidth Counters

- Involves just 2 Streamline Counters
 - External Bus Read Beats
 - External Bus Write Beats

Bandwidth = 967 MB/S

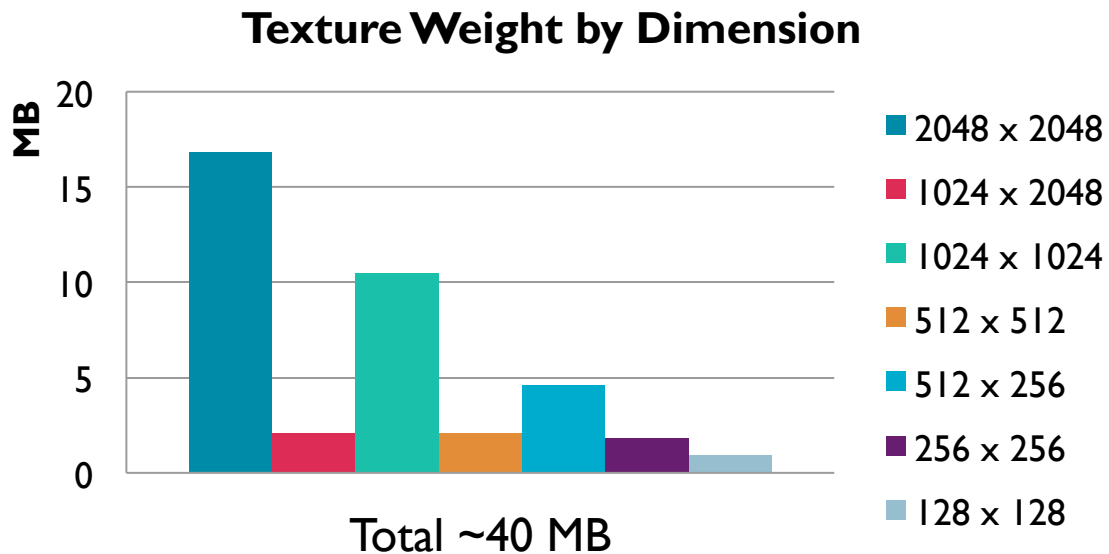


Bandwidth in Bytes = (External Bus Read Beats + External Bus Write Beats) * Bus Width

ARM Mali Graphics Debugger: Textures

Save memory and bandwidth with texture compression

- The current most popular format is ETC Texture Compression
- But ASTC (Adaptive Scalable Texture Compression) can deliver < 1 bit/pixel



Name	Size	Format	Type
Texture 1	2048 x 2048	GL_RGBA	GL_UNSIGNED_BYTE
Texture 16	2048 x 2048	GL_RGBA	GL_UNSIGNED_BYTE
Texture 17	2048 x 2048	C	
Texture 56	2048 x 2048	C	5
Texture 2	1024 x 2048	GL_RGBA	GL_UNSIGNED_BYTE
Texture 18	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE
Texture 37	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE
Texture 38	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE
Texture 40	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE
Texture 41	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE
Texture 42	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE
Texture 43	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE
Texture 60	1024 x 1024	GL_RGBA	GL_UNSIGNED_BYTE

With texture compression, the total amount could be just ~6.4 MB (ASTC 5x5 blocks)

ARM Mali Graphics Debugger: Vertex Buffer Objects

- Using Vertex Buffer Objects (VBOs) can save you a lot of time in overhead
- Every frame in your application, all of your vertices and colour information will get sent to the GPU
- A lot of the time these won't change. So there is no need to keep sending them
- Would be a much better idea to cache the data in graphics memory

The screenshot displays the ARM Mali Graphics Debugger interface. The top window, titled 'monopoly--mgd13.mgd', shows a list of function calls with columns for '#', 'Error', 'Return', and 'Function Call'. The function calls include various OpenGL ES commands such as glEnable, glCullFace, glUseProgram, glEnableVertexAttribArray, glVertexAttribPointer, glUniformMatrix4fv, glDisable, glScissor, glClearColor, and glBindTexture. The bottom window, titled 'Trace Analysis', shows a console with several error messages, including 'Offset is beyond the end of the buffer', 'Vertex attribute 'pointer' less than zero', 'Detected 8 errors returned from functions.', 'Indices buffer may be too sparse (total sparseness: NaN)', and 'Vertex attrib. array is enabled but no data is available'. A blue message at the bottom of the console states 'Detected 17140 calls to glVertexAttribPointer without a bound vertex buffer object'.

#	Error	Return	Function Call
376441	GL_NO_ERROR		glEnable(cap=GL_CULL_FACE)
376442	GL_NO_ERROR		glCullFace(mode=GL_BACK)
376443	GL_NO_ERROR		glUseProgram(program=38)
376444	GL_NO_ERROR		glEnableVertexAttribArray(index=0)
376445	GL_NO_ERROR		glEnableVertexAttribArray(index=1)
376446	GL_NO_ERROR		glEnableVertexAttribArray(index=2)
376447	GL_NO_ERROR		glVertexAttribPointer(indx=0, size=2, type=GL_FLOAT, normalized=GL_F...
376448	GL_NO_ERROR		glVertexAttribPointer(indx=1, size=4, type=GL_FLOAT, normalized=GL_F...
376449	GL_NO_ERROR		glVertexAttribPointer(indx=2, size=2, type=GL_FLOAT, normalized=GL_F...
376450	GL_NO_ERROR		glUniformMatrix4fv(location=2, count=1, transpose=GL_FALSE, value=...
376451	GL_NO_ERROR		glUniformMatrix4fv(location=3, count=1, transpose=GL_FALSE, value=...
376452	GL_NO_ERROR		glDisable(cap=GL_SCISSOR_TEST)
376453	GL_NO_ERROR		glScissor(x=0, y=0, width=2464, height=1504)
376454	GL_NO_ERROR		glClearColor(red=0.0, green=0.0, blue=0.0, alpha=0.0)
376455	GL_NO_ERROR		glDisable(cap=GL_CULL_FACE)
376456	GL_NO_ERROR		glClearColor(red=0.0, green=0.0, blue=0.0, alpha=0.0)
376457	GL_NO_ERROR		glDisable(cap=GL_CULL_FACE)
376458	GL_NO_ERROR		glClearColor(red=0.0, green=0.0, blue=0.0, alpha=0.0)
376459	GL_NO_ERROR		glDisable(cap=GL_CULL_FACE)
376460	GL_NO_ERROR		glDisable(cap=GL_BLEND)
376461	EGL_SUCCESS	EGL_SU...	eglGetError()
376462	EGL_SUCCESS	EGL_TRUE	eglSwapBuffers(dpy=0x71395be8, surface=0x78c9ef50)
376463	GL_NO_ERROR		glDisable(cap=GL_BLEND)
376464	GL_NO_ERROR	GL_NO...	glGetError()
376465	GL_NO_ERROR		glDepthMask(flag=GL_TRUE)
376466	GL_NO_ERROR	GL_NO...	glGetError()
376467	GL_NO_ERROR	GL_NO...	glGetError()
376468	GL_NO_ERROR		glBindTexture(target=GL_TEXTURE_2D, texture=0)
376469	GL_NO_ERROR	GL_NO...	glGetError()
376470	GL_NO_ERROR	GL_NO...	glGetError()
376471	GL_NO_ERROR		glUniformMatrix4fv(location=2, count=1, transpose=GL_FALSE, value=...
376472	GL_NO_ERROR	GL_NO...	glGetError()
376473	GL_NO_ERROR		glClear(mask=<0x4100>)
376474	GL_NO_ERROR	GL_NO...	glGetError()
376475	GL_NO_ERROR	GL_NO...	glGetError()
376476	GL_NO_ERROR		glBindBuffer(target=GL_ARRAY_BUFFER, buffer=3)
376477	GL_NO_ERROR	GL_NO...	glGetError()
376478	GL_NO_ERROR	GL_NO...	glGetError()

Trace Analysis Console:

- ⚠ Offset is beyond the end of the buffer
- ⚠ Vertex attribute 'pointer' less than zero
- ✖ Detected 8 errors returned from functions.
- ⚠ Indices buffer may be too sparse (total sparseness: NaN)
- ⚠ Vertex attrib. array is enabled but no data is available
- ℹ Detected 17140 calls to glVertexAttribPointer without a bound vertex buffer object
- ℹ 100.00% of the draw calls are using GL_TRIANGLES

General Tips

Bandwidth Bound

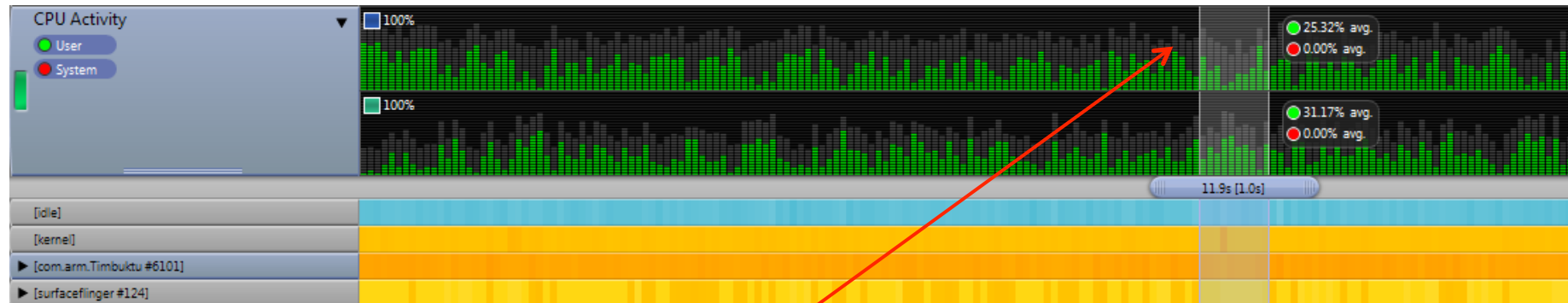
- Use texture compression
- Enable texture mipmapping
- Reduce the number of vertices and varyings
- Interleave vertices, normals, texture coordinates
- Reduce the size of the textures
- Use VBOs

This will also cause a **better cache utilization**.

CPU Bound

CPU Bound Streamline

- Easy just look at the CPU Activity
 - Remember to look at all the cores.



Some of the area is greyed out due to Streamline's ability to present per process CPU activity

General Tips

CPU Bound

- Sometimes a slow frame rate can actually be a CPU issue and not a GPU one
 - In this case optimizing your graphics won't achieve anything
- Most mobile devices have more than one core these days
 - Are you threading your application as much as possible?
- Mali GPU is a deferred architecture
 - Reduce the amount of draw calls you make
 - Try to combine your draw calls together

Summary

- Covered today:
 - Introduction to WebGL
 - PlayCanvas
 - Profiling with ARM DS-5 Streamline
 - Debugging with the ARM Mali Graphics Debugger
- ARM:
 - www.malideveloper.arm.com
 - www.ds.arm.com
 - www.community.arm.com
- WebGL & PlayCanvas:
 - <http://www.khronos.org/webgl/>
 - <https://playcanvas.com>
 - <https://github.com/playcanvas/engine>
 - <http://swoop.playcanvas.com>
 - <https://playcanvas.com/playcanvas/swoop>

Thank You
Any Questions?

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