Mali Tools

Mali GPU Tools

Performance Analysis, Debug, and Software Development

Mali Graphics Debugger

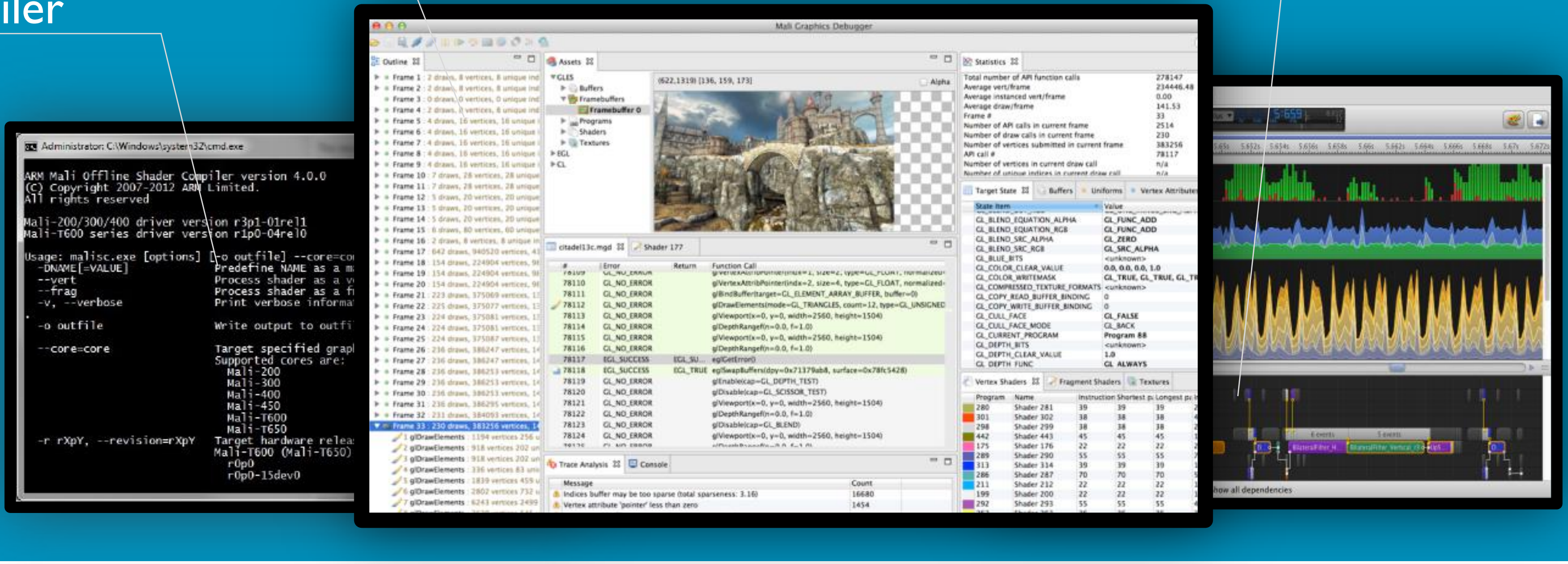
- API Trace & Debug
- OpenGL ES, OpenCL
- Debug and improve performance at frame level

Mali Offline Compiler

- Analyze shader performance
- Command line tool
- Number of cycles
- Registers utilization

ARM DS-5 Streamline

Profile CPUs and Mali GPUs
Timeline
HW Counters
OpenCL visualizer



Administrator: C:\Windows\system32\cmd.exe

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

Mali-200/300/400 driver version r3p1-01rel1
Mali-T600 series driver version r1p0-04rel0

Usage: malisc.exe [options] [-o outfile] --core=co
  -DNAME[=VALUE]              Predefine NAME as a m
  --vert                      Process shader as a v
  --frag                      Process shader as a f
  -v, --verbose               Print verbose informa
.
  -o outfile                  Write output to outfi

  --core=core                 Target specified grap
                              Supported cores are:
                                Mali-200
                                Mali-300
                                Mali-400
                                Mali-450
                                Mali-T600
                                Mali-T650
  -r rXpY, --revision=rXpY    Target hardware relea
                              Mali-T600 (Mali-T650)
                                r0p0
                                r0p0-15dev0
```

OpenGL ES Emulator

- Emulate OpenGL ES 2.0, 3.1
- Supports Android Extension Pack
- Windows and Linux
- Benchmarked against Khronos Conformance Suite

Texture Compression Tool

- Command line and GUI
- ETC, ETC2, ASTC, 3D textures

ASTC encoder

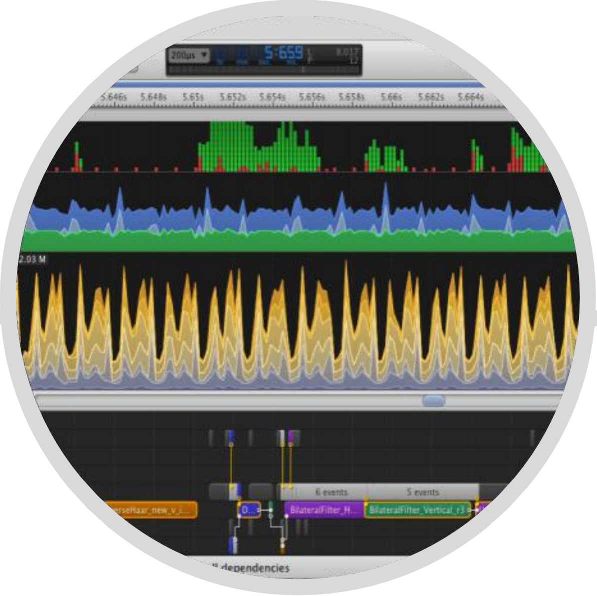
- Available on GitHub

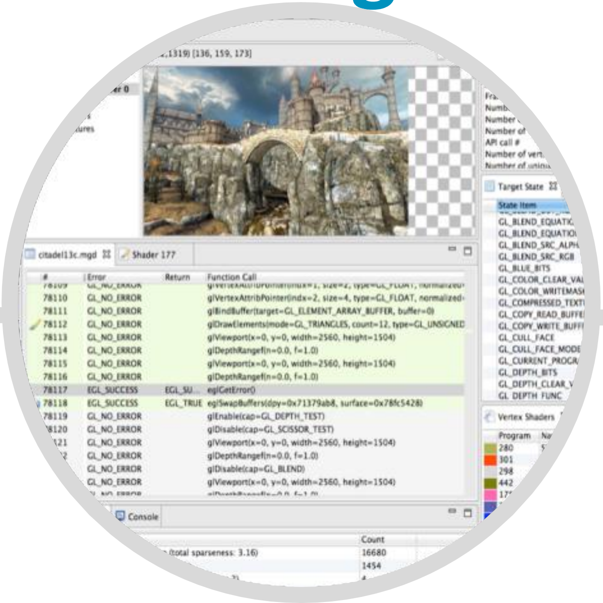
Performance Analysis and Debug with tools from ARM

Analyze

DS-5 Streamline

- Profile CPUs and Mali GPUs
- Timeline
- HW Counters
- OpenCL visualizer





#	Error	Return	Function Call
78110	GL_NO_ERROR		glVertexAttribPointer(indx=2, size=4, type=GL_FLOAT, normalized=
78111	GL_NO_ERROR		glBindBuffer(target=GL_ELEMENT_ARRAY_BUFFER, buffer=0)
78112	GL_NO_ERROR		glDrawElements(mode=GL_TRIANGLES, count=12, type=GL_UNSIGNED
78113	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)
78114	GL_NO_ERROR		glDepthRangef(n=0.0, f=1.0)
78115	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)
78116	GL_NO_ERROR		glDepthRangef(n=0.0, f=1.0)
78117	EGL_SUCCESS	EGL_SU...	eglGetError()
78118	EGL_SUCCESS	EGL_TRUE	eglSwapBuffers(dpy=0x71379ab8, surface=0x78fc5428)
78119	GL_NO_ERROR		glEnable(cap=GL_DEPTH_TEST)
78120	GL_NO_ERROR		glDisable(cap=GL_SCISSOR_TEST)
78121	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)
	GL_NO_ERROR		glDepthRangef(n=0.0, f=1.0)
	GL_NO_ERROR		glDisable(cap=GL_BLEND)
	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)

GL_BLEND_EQUATIO
GL_BLEND_EQUATIO
GL_BLEND_SRC_ALPH
GL_BLEND_SRC_RGB
GL_BLUE_BITS
GL_COLOR_CLEAR_VAL
GL_COLOR_WRITEMAS
GL_COMPRESSED_TEXT
GL_COPY_READ_BUFFE
GL_COPY_WRITE_BUFF
GL_CULL_FACE
GL_CULL_FACE_MODE
GL_CURRENT_PROGRA
GL_DEPTH_BITS
GL_DEPTH_CLEAR_V
GL_DEPTH_FUNC

Program
280
301
298
442

	Count
(total sparseness: 3.16)	16680
	1454

Debug

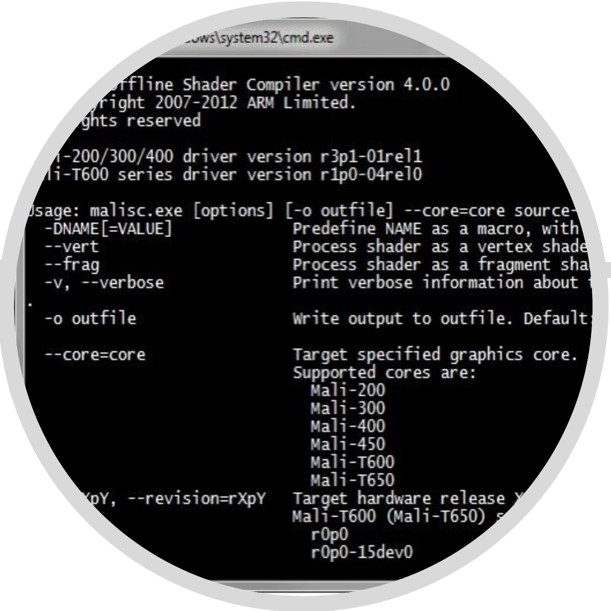
Mali Graphics Debugger

- API Trace & Debug
- OpenGL ES, OpenCL
- Debug and improve performance at frame level

Optimize

Mali Offline Compiler

- Analyze shader performance
- Command line tool
- Number of cycles
- Registers utilization



```
ffline Shader Compiler version 4.0.0
yright 2007-2012 ARM Limited.
ghts reserved

i-200/300/400 driver version r3p1-01rel1
li-T600 series driver version r1p0-04rel0

Usage: malisc.exe [options] [-o outfile] --core=core source-
  -DNAME[=VALUE]             Predefine NAME as a macro, with
  --vert                     Process shader as a vertex shade
  --frag                     Process shader as a fragment sha
  -v, --verbose              Print verbose information about 
.
  -o outfile                 Write output to outfile. Default

  --core=core                Target specified graphics core.
                             Supported cores are:
                               Mali-200
                               Mali-300
                               Mali-400
                               Mali-450
                               Mali-T600
                               Mali-T650
 XpY, --revision=rXpY        Target hardware release X
                             Mali-T600 (Mali-T650) s
                               r0p0
                               r0p0-15dev0
```

Mali Graphics Debugger

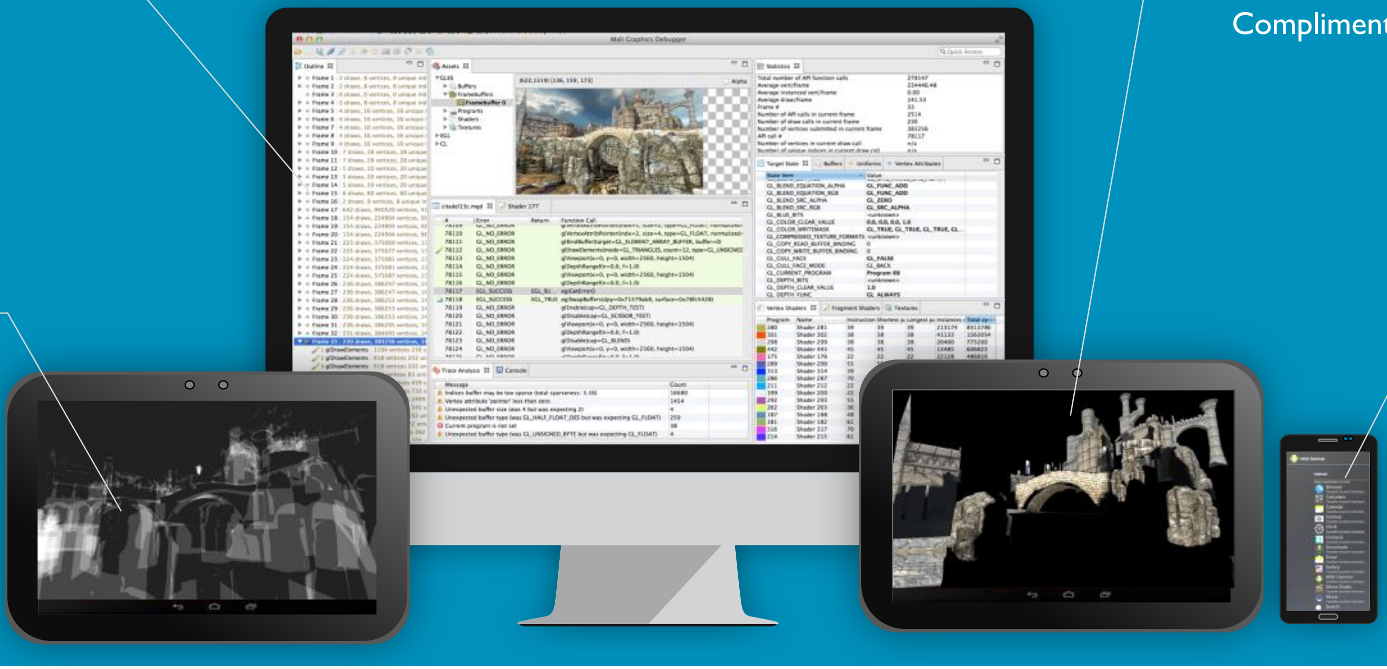
Trace graphics and compute applications to debug issues and analyze the performance

Advanced API Debugger

- Graphics debugging for content developers
- OpenGL® ES 1.1, 2.0, 3.1, EGL, OpenCL™

Advanced Drawing Modes

- Native mode
- Overdraw mode
- Shader map mode
- Fragment count mode



Frame Analyzer

Understand issues and causes at frame level
Complimentary to ARM® DS-5 Streamline

Android Application

Start/stop daemon
List all the debuggable processes
Launch application to debug

Graphics State Visibility

- Shows the current state of the API at any point of the trace
- Discover when and how a certain state was changed

Analyse shaders and kernels

- All the shaders being used by the application are reported
- Shader statistics
- Each shader is compiled with the Mali Offline Compiler and is statically analyzed

Flexible and cross platform

- Runs on Windows, Linux and Mac
- Traces from Android and Linux targets

ARM DS-5

Streamline

Performance Analyzer

Speed Up Your Code

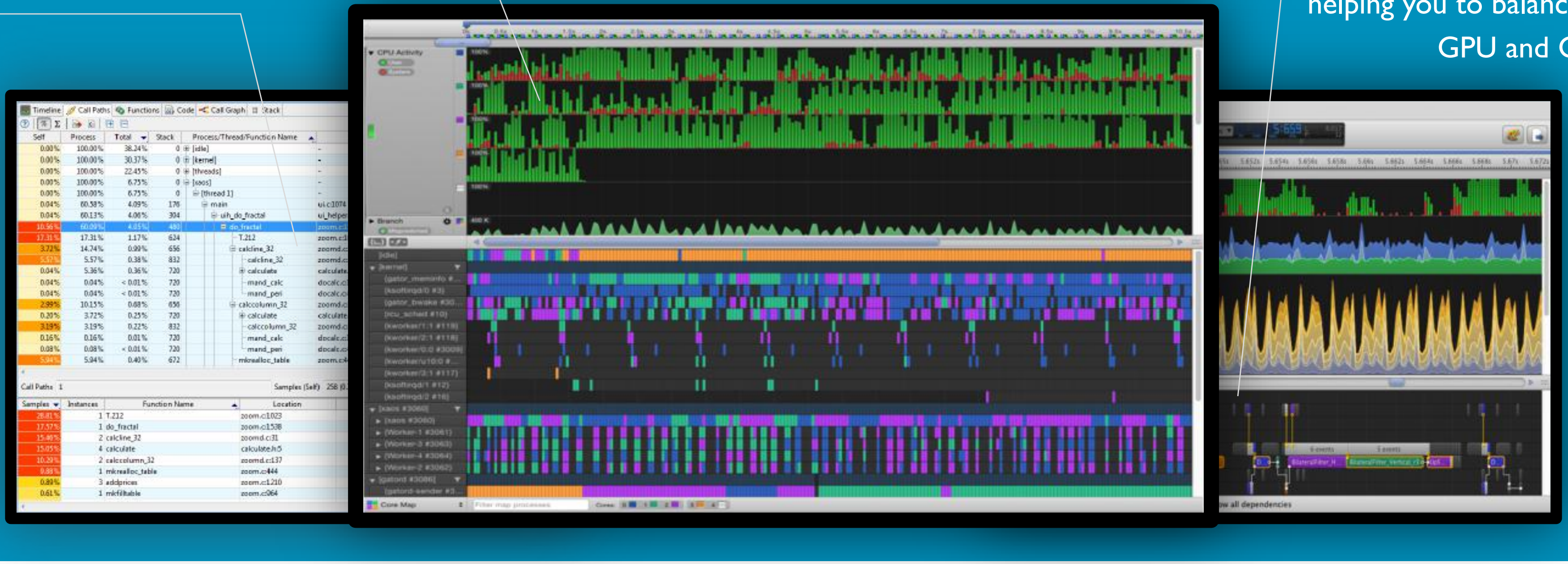
- Find out where the CPU is spending the most time
- Tune code for optimal cache usage

Drill down to the Source Code

- Break performance down by function
- View it alongside the disassembly

OpenCL™ Visualizer

Visualization of OpenCL dependencies, helping you to balance resources between GPU and CPU better than ever



Self	Process	Total	Stack	Process/Thread/Function Name	
0.00%	100.00%	38.24%	0	[idle]	-
0.00%	100.00%	30.37%	0	[kernel]	-
0.00%	100.00%	22.45%	0	[threads]	-
0.00%	100.00%	6.75%	0	[xaos]	-
0.00%	100.00%	6.75%	0	[thread 1]	-
0.04%	60.58%	4.09%	176	main	ui.c:1074
0.04%	60.13%	4.06%	304	uih_do_fractal	ui_helper
10.56%	60.09%	4.05%	480	do_fractal	zoom.c:1
17.31%	17.31%	1.17%	624	T.212	zoom.c:1
3.72%	14.74%	0.99%	656	calcline_32	zoomd.c
5.57%	5.57%	0.38%	832	calcline_32	zoomd.c
0.04%	5.36%	0.36%	720	calculate	calculate.
0.04%	0.04%	< 0.01%	720	mand_calc	docalc.c
0.04%	0.04%	< 0.01%	720	mand_peri	docalc.c
2.99%	10.15%	0.68%	656	calccolumn_32	zoomd.c
0.20%	3.72%	0.25%	720	calculate	calculate
3.19%	3.19%	0.22%	832	calccolumn_32	zoomd.c
0.16%	0.16%	0.01%	720	mand_calc	docalc.c
0.08%	0.08%	< 0.01%	720	mand_peri	docalc.c
5.94%	5.94%	0.40%	672	mkrealloc_table	zoom.c:4

Call Paths 1 Samples (Self) 258 (0.

Samples	Instances	Function Name	Location
28.81%	1	T.212	zoom.c:1023
17.57%	1	do_fractal	zoom.c:1538
15.40%	2	calcline_32	zoomd.c:31
15.05%	4	calculate	calculate.h:5
10.29%	2	calccolumn_32	zoomd.c:137
9.88%	1	mkrealloc_table	zoom.c:444
0.89%	3	addprices	zoom.c:1210
0.61%	1	mkfilltable	zoom.c:964

Mali GPU Support

- Analyze and optimize Mali™ GPU utilization
- Monitor CPU and GPU cache usage

Optimize energy efficiency

- Monitor actual power consumption with the ARM Energy Probe
- Correlate software execution to actual power consumption

Customize it for Your System

- Flexible architecture permits easy addition of new counters
- Open source driver and daemon gives developers ultimate flexibility

arm

Mali Offline Compiler

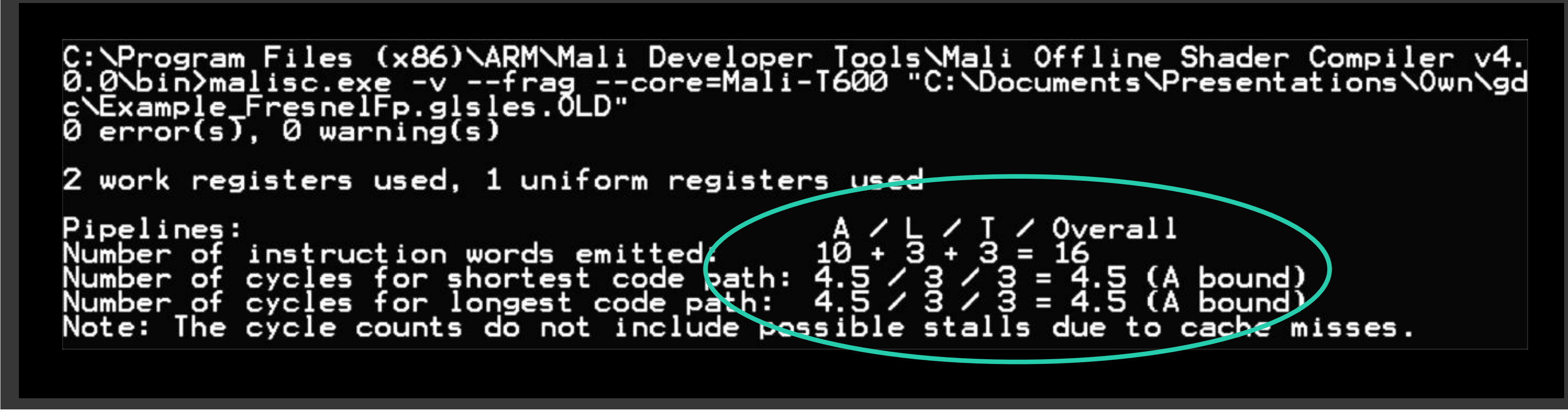
Compile and statically analyze your shaders ahead of execution

Compiles shader code written in OpenGL ES Shading Language (ESSL) offline

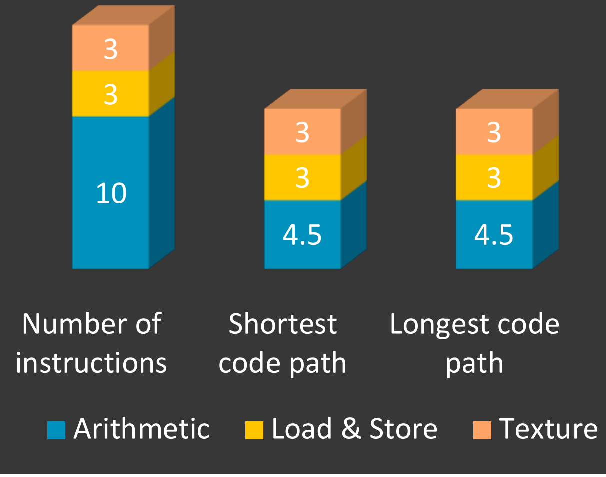
Provides verbose shader performance & error messages for optimization and debug

Support for Mali-300, Mali-400, Mali-450, Mali-T604, Mali-T628, Mali-T760, Mali-T880, Mali-G71

- Integration with Mali Graphics Debugger
- Integration with OpenGL ES Emulator

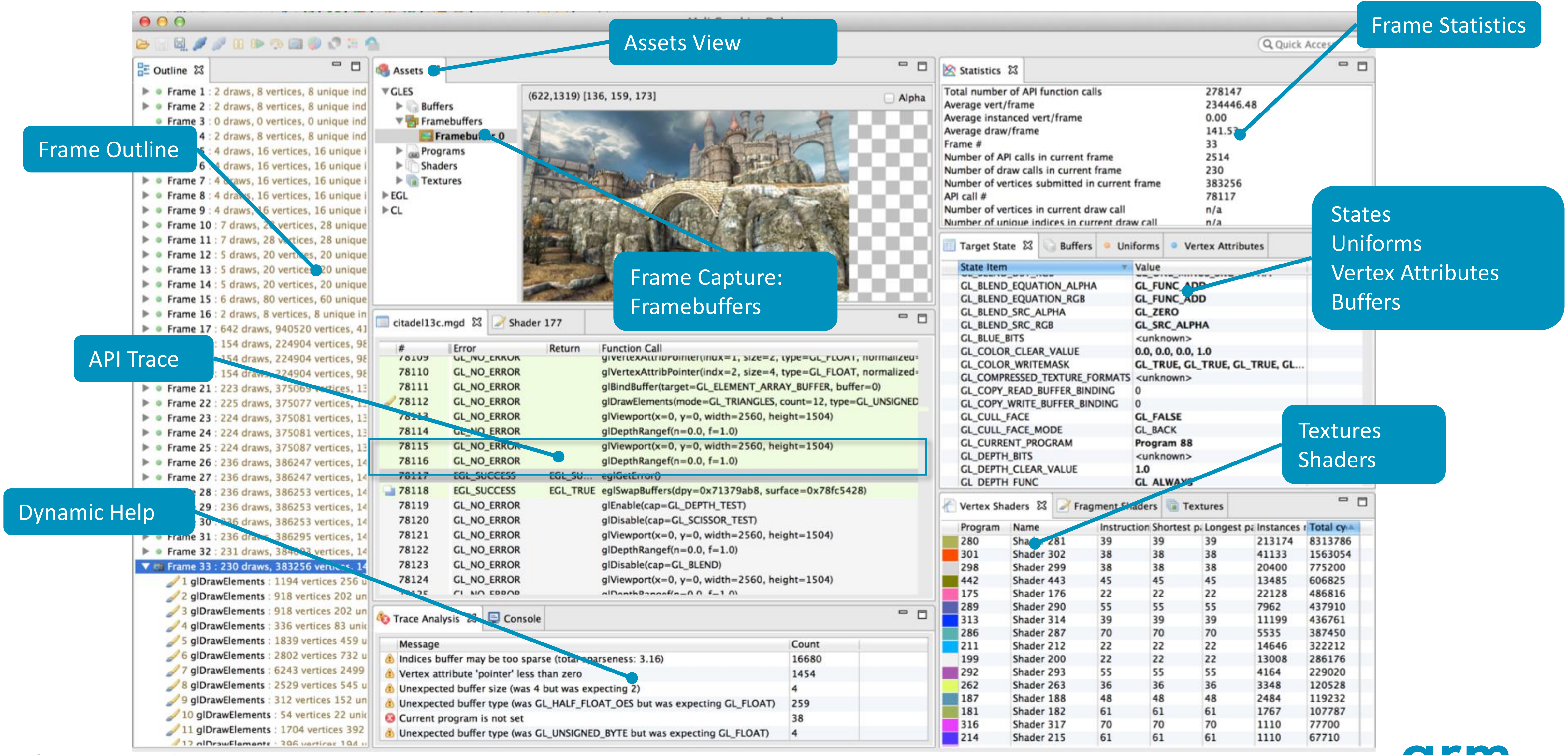


```
C:\Program Files (x86)\ARM\Mali Developer Tools\Mali Offline Shader Compiler v4.
0.0\bin>malisc.exe -v --frag --core=Mali-T600 "C:\Documents\Presentations\Own\gd
c\Example_FresnelFp.glsles.OLD"
0 error(s), 0 warning(s)

2 work registers used, 1 uniform registers used

Pipelines:                              A / L / T / Overall
Number of instruction words emitted:   10 + 3 + 3 = 16
Number of cycles for shortest code path: 4.5 / 3 / 3 = 4.5 (A bound)
Number of cycles for longest code path:  4.5 / 3 / 3 = 4.5 (A bound)
Note: The cycle counts do not include possible stalls due to cache misses.
```



Section 2: Deep Dive MGD and Streamline

Overview of the ARM Mali Graphics Debugger



Assets View

Frame Statistics

Frame Outline

States
Uniforms
Vertex Attributes
Buffers

Frame Capture: Framebuffers

API Trace

Textures
Shaders

Dynamic Help

Outline

- Frame 1 : 2 draws, 8 vertices, 8 unique ind
- Frame 2 : 2 draws, 8 vertices, 8 unique ind
- Frame 3 : 0 draws, 0 vertices, 0 unique ind
- 4 : 2 draws, 8 vertices, 8 unique ind
- : 4 draws, 16 vertices, 16 unique i
- 6 : 4 draws, 16 vertices, 16 unique i
- Frame 7 : 4 draws, 16 vertices, 16 unique i
- Frame 8 : 4 draws, 16 vertices, 16 unique i
- Frame 9 : 4 draws, 16 vertices, 16 unique i
- Frame 10 : 7 draws, 28 vertices, 28 unique
- Frame 11 : 7 draws, 28 vertices, 28 unique
- Frame 12 : 5 draws, 20 vertices, 20 unique
- Frame 13 : 5 draws, 20 vertices, 20 unique
- Frame 14 : 5 draws, 20 vertices, 20 unique
- Frame 15 : 6 draws, 80 vertices, 60 unique
- Frame 16 : 2 draws, 8 vertices, 8 unique in
- Frame 17 : 642 draws, 940520 vertices, 41
- : 154 draws, 224904 vertices, 98
- 154 draws, 224904 vertices, 98
- : 154 draws, 224904 vertices, 98
- Frame 21 : 223 draws, 375069 vertices, 13
- Frame 22 : 225 draws, 375077 vertices, 1
- Frame 23 : 224 draws, 375081 vertices, 13
- Frame 24 : 224 draws, 375081 vertices, 13
- Frame 25 : 224 draws, 375087 vertices, 13
- Frame 26 : 236 draws, 386247 vertices, 14
- Frame 27 : 236 draws, 386247 vertices, 14
- 28 : 236 draws, 386253 vertices, 14
- 29 : 236 draws, 386253 vertices, 14
- 30 : 236 draws, 386253 vertices, 14
- Frame 31 : 236 draws, 386295 vertices, 14
- Frame 32 : 231 draws, 384093 vertices, 14
- Frame 33 : 230 draws, 383256 vertices, 14
  - 1 glDrawElements : 1194 vertices 256 u
  - 2 glDrawElements : 918 vertices 202 un
  - 3 glDrawElements : 918 vertices 202 un
  - 4 glDrawElements : 336 vertices 83 unic
  - 5 glDrawElements : 1839 vertices 459 u
  - 6 glDrawElements : 2802 vertices 732 u
  - 7 glDrawElements : 6243 vertices 2499
  - 8 glDrawElements : 2529 vertices 545 u
  - 9 glDrawElements : 312 vertices 152 un
  - 10 glDrawElements : 54 vertices 22 unic
  - 11 glDrawElements : 1704 vertices 392
  - 12 glDrawElements : 396 vertices 194 u

Assets

- GLES
  - Buffers
  - Framebuffers
    - Framebuffer 0
  - Programs
  - Shaders
  - Textures
- EGL
- CL

(622,1319) [136, 159, 173] □ Alpha

citadel13c.mgd | Shader 177

#	Error	Return	Function Call
78109	GL_NO_ERROR		glVertexAttribPointer(indx=1, size=2, type=GL_FLOAT, normalized=
78110	GL_NO_ERROR		glVertexAttribPointer(indx=2, size=4, type=GL_FLOAT, normalized=
78111	GL_NO_ERROR		glBindBuffer(target=GL_ELEMENT_ARRAY_BUFFER, buffer=0)
78112	GL_NO_ERROR		glDrawElements(mode=GL_TRIANGLES, count=12, type=GL_UNSIGNED
78113	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)
78114	GL_NO_ERROR		glDepthRangef(n=0.0, f=1.0)
78115	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)
78116	GL_NO_ERROR		glDepthRangef(n=0.0, f=1.0)
78117	EGL_SUCCESS	EGL_SU...	eglGetError()
78118	EGL_SUCCESS	EGL_TRUE	eglSwapBuffers(dpy=0x71379ab8, surface=0x78fc5428)
78119	GL_NO_ERROR		glEnable(cap=GL_DEPTH_TEST)
78120	GL_NO_ERROR		glDisable(cap=GL_SCISSOR_TEST)
78121	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)
78122	GL_NO_ERROR		glDepthRangef(n=0.0, f=1.0)
78123	GL_NO_ERROR		glDisable(cap=GL_BLEND)
78124	GL_NO_ERROR		glViewport(x=0, y=0, width=2560, height=1504)
78125	GL_NO_ERROR		glDepthRangef(n=0.0, f=1.0)

Trace Analysis | Console

Message	Count
Indices buffer may be too sparse (total sparseness: 3.16)	16680
Vertex attribute 'pointer' less than zero	1454
Unexpected buffer size (was 4 but was expecting 2)	4
Unexpected buffer type (was GL_HALF_FLOAT_OES but was expecting GL_FLOAT)	259
Current program is not set	38
Unexpected buffer type (was GL_UNSIGNED_BYTE but was expecting GL_FLOAT)	4

Quick Access

Statistics

Total number of API function calls	278147
Average vert/frame	234446.48
Average instanced vert/frame	0.00
Average draw/frame	141.53
Frame #	33
Number of API calls in current frame	2514
Number of draw calls in current frame	230
Number of vertices submitted in current frame	383256
API call #	78117
Number of vertices in current draw call	n/a
Number of unique indices in current draw call	n/a

Target State | Buffers | Uniforms | Vertex Attributes

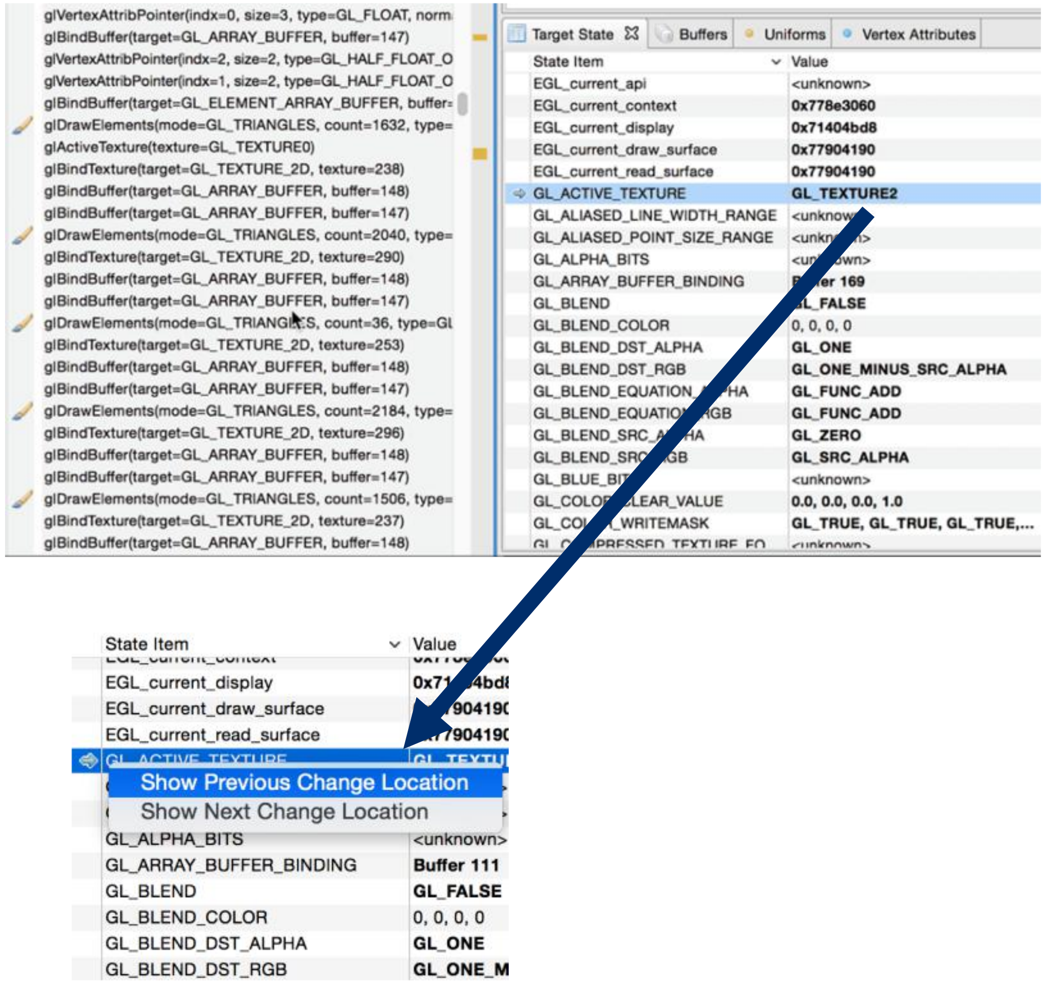
State Item	Value
GL_BLEND_EQUATION_ALPHA	GL_FUNC_ADD
GL_BLEND_EQUATION_RGB	GL_FUNC_ADD
GL_BLEND_SRC_ALPHA	GL_ZERO
GL_BLEND_SRC_RGB	GL_SRC_ALPHA
GL_BLUE_BITS	<unknown>
GL_COLOR_CLEAR_VALUE	0.0, 0.0, 0.0, 1.0
GL_COLOR_WRITEMASK	GL_TRUE, GL_TRUE, GL_TRUE, GL...
GL_COMPRESSED_TEXTURE_FORMATS	<unknown>
GL_COPY_READ_BUFFER_BINDING	0
GL_COPY_WRITE_BUFFER_BINDING	0
GL_CULL_FACE	GL_FALSE
GL_CULL_FACE_MODE	GL_BACK
GL_CURRENT_PROGRAM	Program 88
GL_DEPTH_BITS	<unknown>
GL_DEPTH_CLEAR_VALUE	1.0
GL_DEPTH_FUNC	GL_ALWAYS

Vertex Shaders | Fragment Shaders | Textures

Program	Name	Instruction	Shortest p	Longest pa	Instances	Total cy
280	Shader 281	39	39	39	213174	8313786
301	Shader 302	38	38	38	41133	1563054
298	Shader 299	38	38	38	20400	775200
442	Shader 443	45	45	45	13485	606825
175	Shader 176	22	22	22	22128	486816
289	Shader 290	55	55	55	7962	437910
313	Shader 314	39	39	39	11199	436761
286	Shader 287	70	70	70	5535	387450
211	Shader 212	22	22	22	14646	322212
199	Shader 200	22	22	22	13008	286176
292	Shader 293	55	55	55	4164	229020
262	Shader 263	36	36	36	3348	120528
187	Shader 188	48	48	48	2484	119232
181	Shader 182	61	61	61	1767	107787
316	Shader 317	70	70	70	1110	77700
214	Shader 215	61	61	61	1110	67710

Mali Graphics Debugger

Target state



```
glVertexAttribPointer(indx=0, size=3, type=GL_FLOAT, norm
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=147)
glVertexAttribPointer(indx=2, size=2, type=GL_HALF_FLOAT_O
glVertexAttribPointer(indx=1, size=2, type=GL_HALF_FLOAT_O
glBindBuffer(target=GL_ELEMENT_ARRAY_BUFFER, buffer=
glDrawElements(mode=GL_TRIANGLES, count=1632, type=
glActiveTexture(texture=GL_TEXTURE0)
glBindTexture(target=GL_TEXTURE_2D, texture=238)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=148)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=147)
glDrawElements(mode=GL_TRIANGLES, count=2040, type=
glBindTexture(target=GL_TEXTURE_2D, texture=290)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=148)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=147)
glDrawElements(mode=GL_TRIANGLES, count=36, type=GL
glBindTexture(target=GL_TEXTURE_2D, texture=253)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=148)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=147)
glDrawElements(mode=GL_TRIANGLES, count=2184, type=
glBindTexture(target=GL_TEXTURE_2D, texture=296)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=148)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=147)
glDrawElements(mode=GL_TRIANGLES, count=1506, type=
glBindTexture(target=GL_TEXTURE_2D, texture=237)
glBindBuffer(target=GL_ARRAY_BUFFER, buffer=148)
```

State Item	Value
EGL_current_api	<unknown>
EGL_current_context	0x778e3060
EGL_current_display	0x71404bd8
EGL_current_draw_surface	0x77904190
EGL_current_read_surface	0x77904190
GL_ACTIVE_TEXTURE	GL_TEXTURE2
GL_ALIASED_LINE_WIDTH_RANGE	<unknown>
GL_ALIASED_POINT_SIZE_RANGE	<unknown>
GL_ALPHA_BITS	<unknown>
GL_ARRAY_BUFFER_BINDING	Buffer 169
GL_BLEND	GL_FALSE
GL_BLEND_COLOR	0, 0, 0, 0
GL_BLEND_DST_ALPHA	GL_ONE
GL_BLEND_DST_RGB	GL_ONE_MINUS_SRC_ALPHA
GL_BLEND_EQUATION_ALPHA	GL_FUNC_ADD
GL_BLEND_EQUATION_RGB	GL_FUNC_ADD
GL_BLEND_SRC_ALPHA	GL_ZERO
GL_BLEND_SRC_RGB	GL_SRC_ALPHA
GL_BLUE_BITS	<unknown>
GL_COLOR_CLEAR_VALUE	0.0, 0.0, 0.0, 1.0
GL_COLOR_WRITEMASK	GL_TRUE, GL_TRUE, GL_TRUE,...
GL_COMPRESSED_TEXTURE_FO	<unknown>

State Item	Value
EGL_current_context	0x778e
EGL_current_display	0x71404bd8
EGL_current_draw_surface	0x77904190
EGL_current_read_surface	0x77904190
GL_ACTIVE_TEXTURE	GL_TEXTU
GL_ALPHA_BITS	<unknown>
GL_ARRAY_BUFFER_BINDING	Buffer 111
GL_BLEND	GL_FALSE
GL_BLEND_COLOR	0, 0, 0, 0
GL_BLEND_DST_ALPHA	GL_ONE
GL_BLEND_DST_RGB	GL_ONE_M

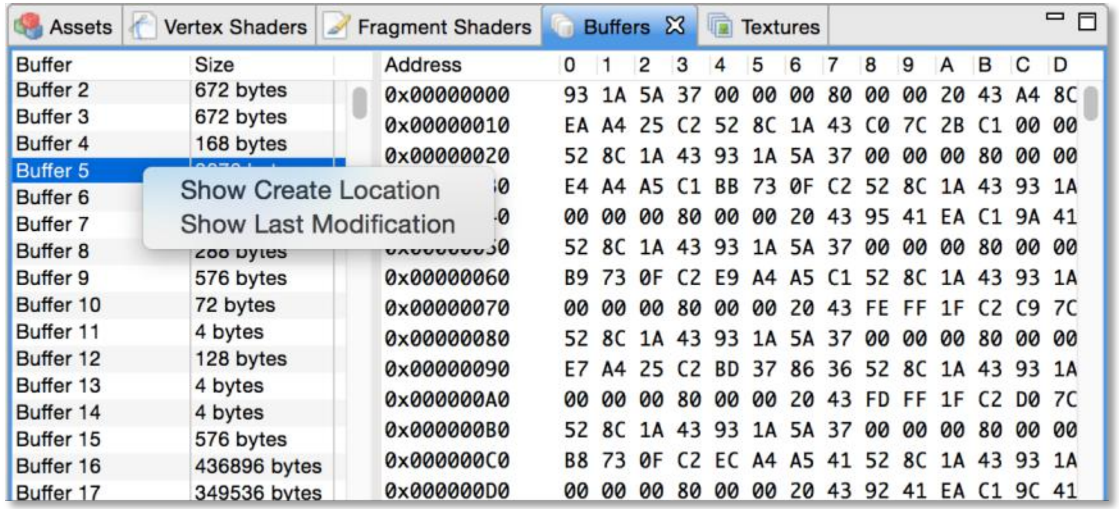
Shows the current state of the API at any point of the trace

- Every time a new API call is selected in the trace the state is updated
- Useful to debug problems and understand causes for performance issues

Discover when and how a certain state was changed

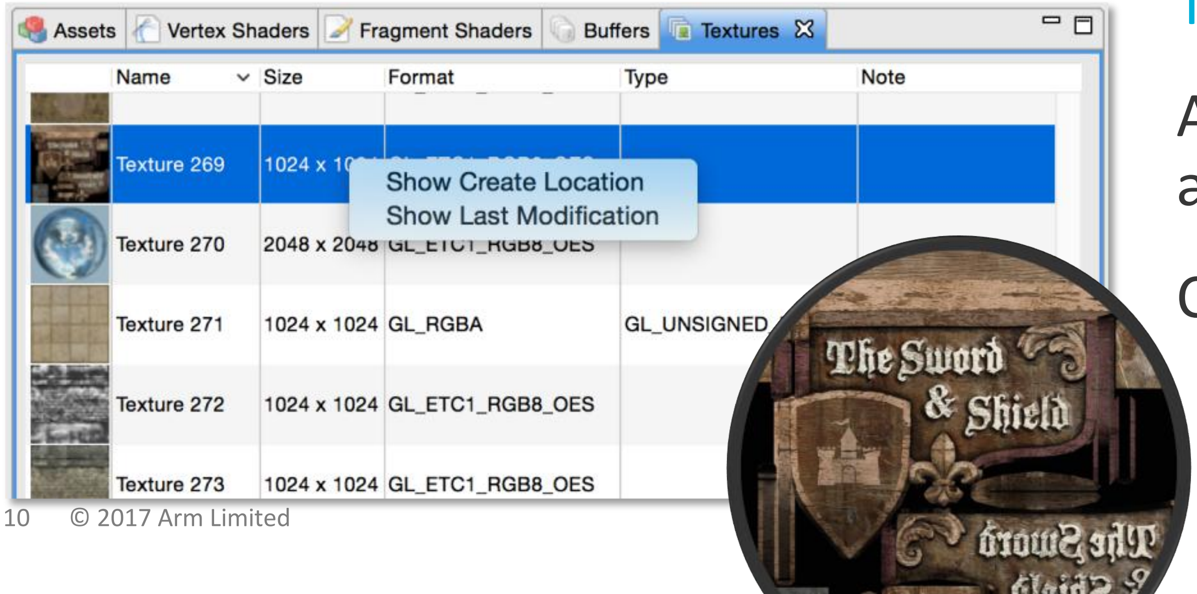
Mali Graphics Debugger

All the heavy assets are available for debugging, including data buffers and textures



Buffer	Size
Buffer 2	672 bytes
Buffer 3	672 bytes
Buffer 4	168 bytes
Buffer 5	[illegible]
Buffer 6	
Buffer 7	
Buffer 8	288 bytes
Buffer 9	576 bytes
Buffer 10	72 bytes
Buffer 11	4 bytes
Buffer 12	128 bytes
Buffer 13	4 bytes
Buffer 14	4 bytes
Buffer 15	576 bytes
Buffer 16	436896 bytes
Buffer 17	349536 bytes

```
Address     0  1  2  3  4  5  6  7  8  9  A  B  C  D
0x00000000  93 1A 5A 37 00 00 00 80 00 00 20 43 A4 8C
0x00000010  EA A4 25 C2 52 8C 1A 43 C0 7C 2B C1 00 00
0x00000020  52 8C 1A 43 93 1A 5A 37 00 00 00 80 00 00
         0  E4 A4 A5 C1 BB 73 0F C2 52 8C 1A 43 93 1A
         0  00 00 00 80 00 00 20 43 95 41 EA C1 9A 41
0x00000050  52 8C 1A 43 93 1A 5A 37 00 00 00 80 00 00
0x00000060  B9 73 0F C2 E9 A4 A5 C1 52 8C 1A 43 93 1A
0x00000070  00 00 00 80 00 00 20 43 FE FF 1F C2 C9 7C
0x00000080  52 8C 1A 43 93 1A 5A 37 00 00 00 80 00 00
0x00000090  E7 A4 25 C2 BD 37 86 36 52 8C 1A 43 93 1A
0x000000A0  00 00 00 80 00 00 20 43 FD FF 1F C2 D0 7C
0x000000B0  52 8C 1A 43 93 1A 5A 37 00 00 00 80 00 00
0x000000C0  B8 73 0F C2 EC A4 A5 41 52 8C 1A 43 93 1A
0x000000D0  00 00 00 80 00 00 20 43 92 41 EA C1 9C 41
```



Name	Size	Format	Type	Note
Texture 269	1024 x 10			
Texture 270	2048 x 2048	GL_ETC1_RGB8_OES		
Texture 271	1024 x 1024	GL_RGBA	GL_UNSIGNED	
Texture 272	1024 x 1024	GL_ETC1_RGB8_OES		
Texture 273	1024 x 1024	GL_ETC1_RGB8_OES		

Buffers

Client and server side buffers are captured every time they change

See how each API call affects them

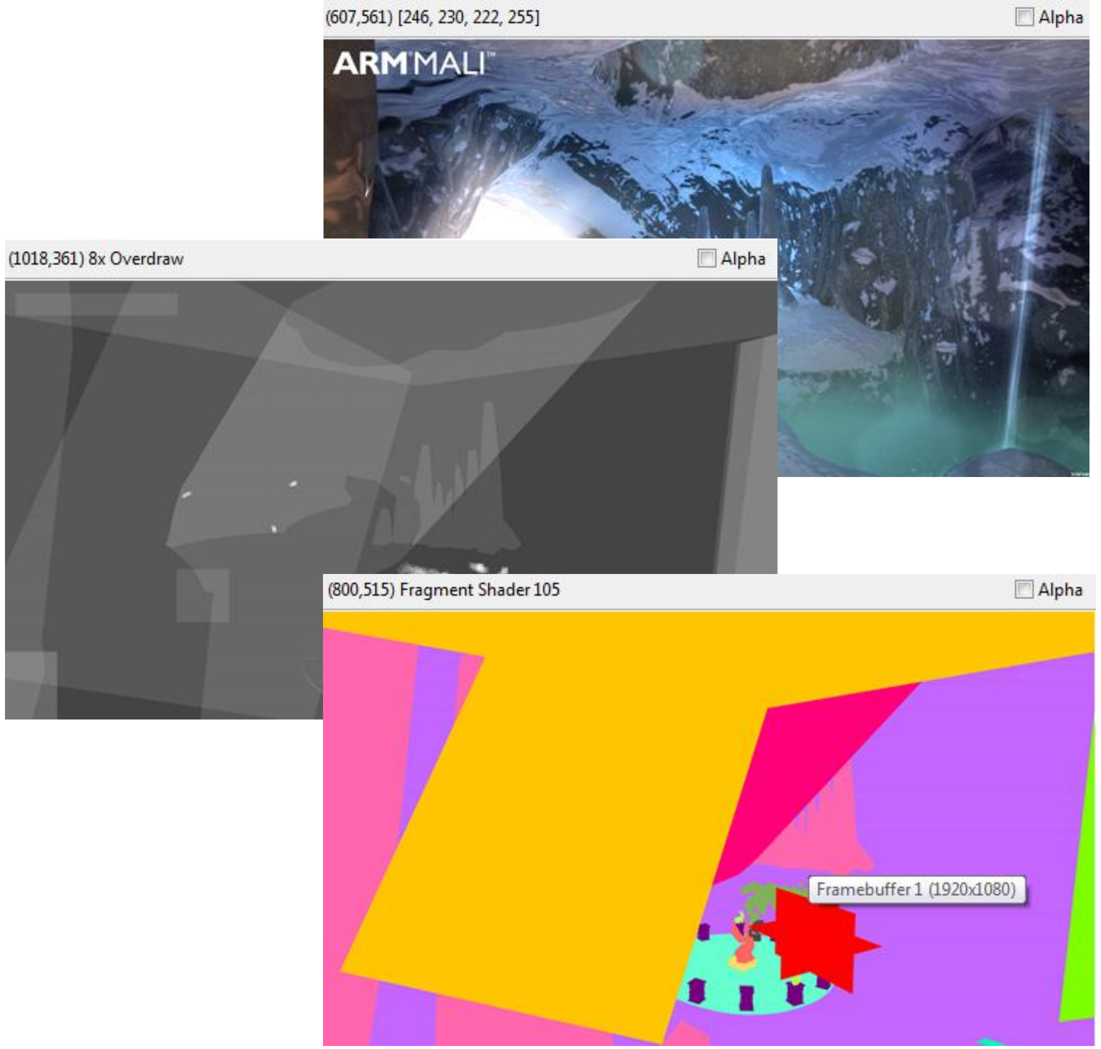
Textures

All the textures being uploaded are captured at native resolution

Check their size, format and type

Mali Graphics Debugger

Alternative drawing modes



Different drawing modes can be forced and used both for live rendering and frame captures

Native mode

- Frames are rendered with the original shaders

Overdraw mode

- Highlights where overdraw happens (ie. objects are drawn on top of each other)

Shader map mode

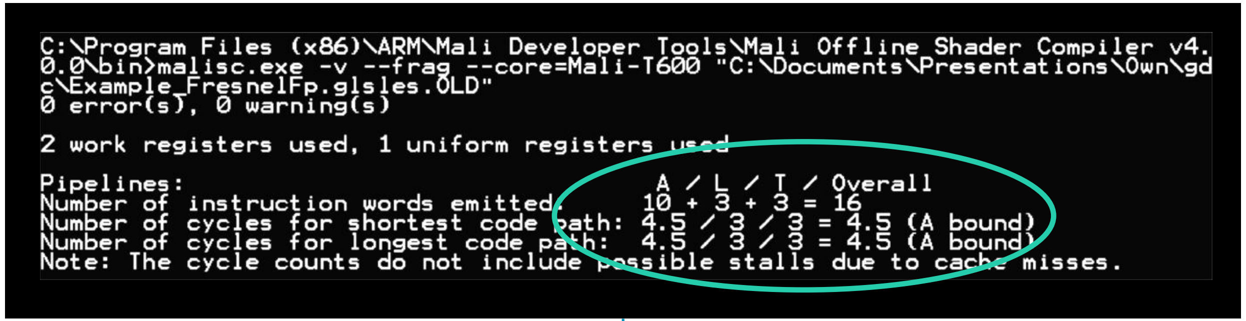
- Native shaders are replaced with different solid colors

Fragment count mode

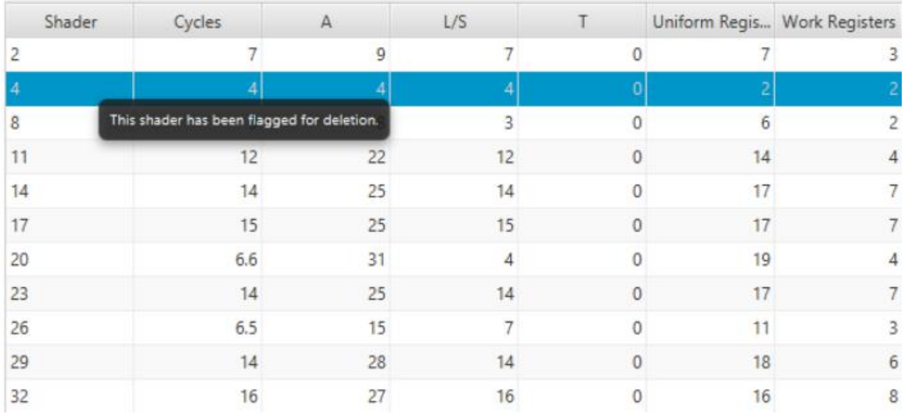
- All the fragments that are processed by each frame are counted

Running through offline Compiler

- The Mali Offline Compiler is integrated into MGD.
- Every shader that is in your application is automatically ran through the compiler.
- The results are placed in a table so you can quickly see which shaders are the most expensive.



```
C:\Program Files (x86)\ARM\Mali Developer Tools\Mali Offline Shader Compiler v4.
0.0\bin>malisc.exe -v --frag --core=Mali-T600 "C:\Documents\Presentations\Own\gd
c\Example_FresnelFp.glsles.OLD"
0 error(s), 0 warning(s)

2 work registers used, 1 uniform registers used

Pipelines:                                A / L / T / Overall
Number of instruction words emitted:     10 + 3 + 3 = 16
Number of cycles for shortest code path: 4.5 / 3 / 3 = 4.5 (A bound)
Number of cycles for longest code path:  4.5 / 3 / 3 = 4.5 (A bound)
Note: The cycle counts do not include possible stalls due to cache misses.
```



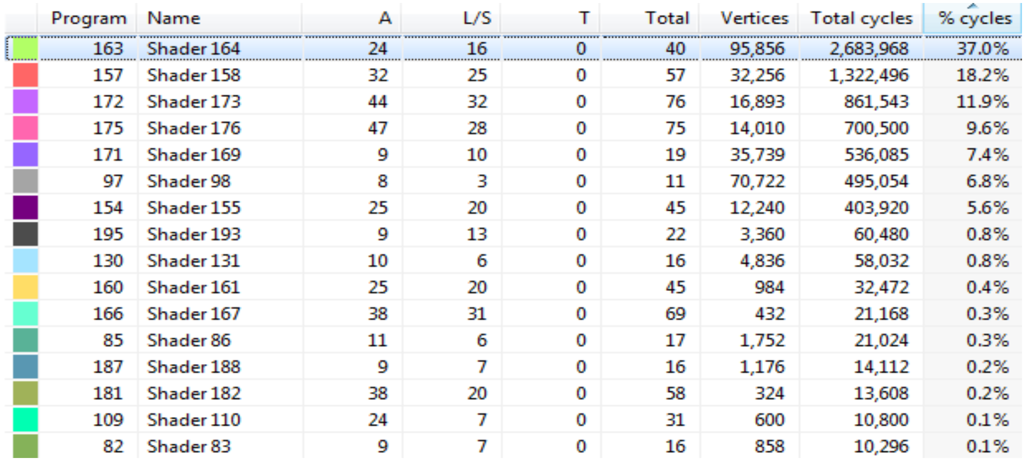
Shader	Cycles	A	L/S	T	Uniform Regis...	Work Registers
2	7	9	7	0	7	3
4	4	4	4	0	2	2
8	[illegible]	[illegible]	3	0	6	2
11	12	22	12	0	14	4
14	14	25	14	0	17	7
17	15	25	15	0	17	7
20	6.6	31	4	0	19	4
23	14	25	14	0	17	7
26	6.5	15	7	0	11	3
29	14	28	14	0	18	6
32	16	27	16	0	16	8

This shader has been flagged for deletion.

Estimation of Vertex and Fragment Cost

Vertex Count

- For every drawcall MGD knows how many vertices are being drawn.
- Therefore it can estimate how expensive each shader
- Report this to the user in a sortable table.

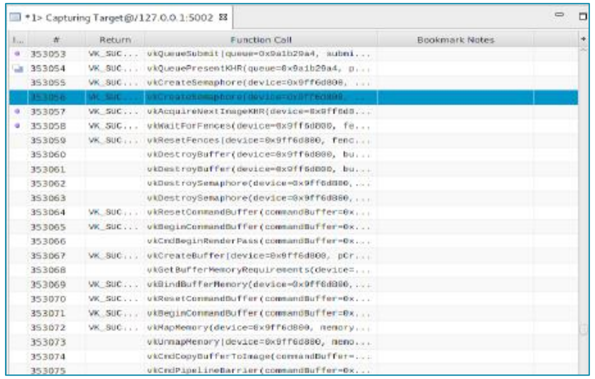


	Program	Name	A	L/S	T	Total	Vertices	Total cycles	% cycles
🟩	163	Shader 164	24	16	0	40	95,856	2,683,968	37.0%
🟥	157	Shader 158	32	25	0	57	32,256	1,322,496	18.2%
🟪	172	Shader 173	44	32	0	76	16,893	861,543	11.9%
🟪	175	Shader 176	47	28	0	75	14,010	700,500	9.6%
🟪	171	Shader 169	9	10	0	19	35,739	536,085	7.4%
⬜	97	Shader 98	8	3	0	11	70,722	495,054	6.8%
🟪	154	Shader 155	25	20	0	45	12,240	403,920	5.6%
⬛	195	Shader 193	9	13	0	22	3,360	60,480	0.8%
🟦	130	Shader 131	10	6	0	16	4,836	58,032	0.8%
🟨	160	Shader 161	25	20	0	45	984	32,472	0.4%
🟩	166	Shader 167	38	31	0	69	432	21,168	0.3%
🟩	85	Shader 86	11	6	0	17	1,752	21,024	0.3%
🟦	187	Shader 188	9	7	0	16	1,176	14,112	0.2%
🟩	181	Shader 182	38	20	0	58	324	13,608	0.2%
🟩	109	Shader 110	24	7	0	31	600	10,800	0.1%
🟩	82	Shader 83	9	7	0	16	858	10,296	0.1%

Fragment Count

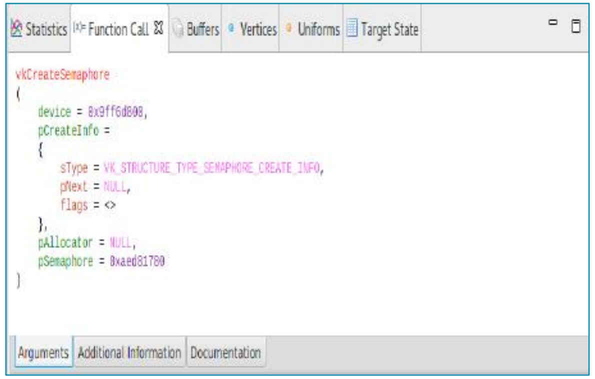
- At API level there is no way of finding out how many fragments are drawn.
- MGD does several calculations on the Framebuffer to detect how many fragments have been drawn
- This can produce an estimated cost much like the vertex count

Mali Graphics Debugger – Vulkan Support



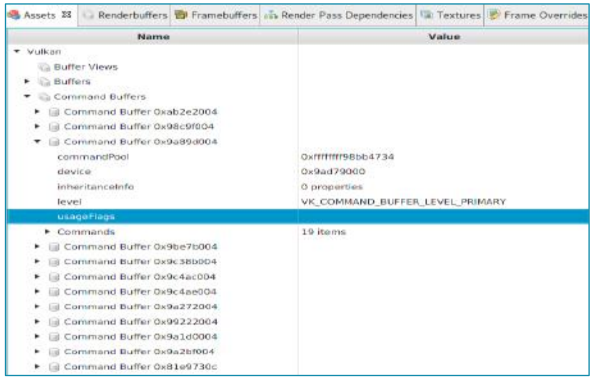
I...	#	Return	Function Call	Bookmark Notes
	353053	VK_SUC...	vkQueueSubmit(queue=0x9a1b29a4, submi...	
	353054	VK_SUC...	vkQueuePresentKHR(queue=0x9a1b29a4, p...	
	353055	VK_SUC...	vkCreateSemaphore(device=0x9ff6d808, ...	
	353056	VK_SUC...	vkCreateSemaphore(device=0x9ff6d808, ...	
	353057	VK_SUC...	vkAcquireNextImageKHR(device=0x9ff6d8...	
	353058	VK_SUC...	vkWaitForFences(device=0x9ff6d808, fe...	
	353059	VK_SUC...	vkResetFences(device=0x9ff6d808, fenc...	
	353060		vkDestroyBuffer(device=0x9ff6d808, bu...	
	353061		vkDestroyBuffer(device=0x9ff6d808, bu...	
	353062		vkDestroySemaphore(device=0x9ff6d808,...	
	353063		vkDestroySemaphore(device=0x9ff6d808,...	
	353064	VK_SUC...	vkResetCommandBuffer(commandBuffer=0x...	
	353065	VK_SUC...	vkBeginCommandBuffer(commandBuffer=0x...	
	353066		vkCmdBeginRenderPass(commandBuffer=0x...	
	353067	VK_SUC...	vkCreateBuffer(device=0x9ff6d808, pCr...	
	353068		vkGetBufferMemoryRequirements(device=...	
	353069	VK_SUC...	vkBindBufferMemory(device=0x9ff6d808,...	
	353070	VK_SUC...	vkResetCommandBuffer(commandBuffer=0x...	
	353071	VK_SUC...	vkBeginCommandBuffer(commandBuffer=0x...	
	353072	VK_SUC...	vkMapMemory(device=0x9ff6d808, memory...	
	353073		vkUnmapMemory(device=0x9ff6d808, memo...	
	353074		vkCmdCopyBufferToImage(commandBuffer=...	
	353075		vkCmdPipelineBarrier(commandBuffer=0x...	

Trace



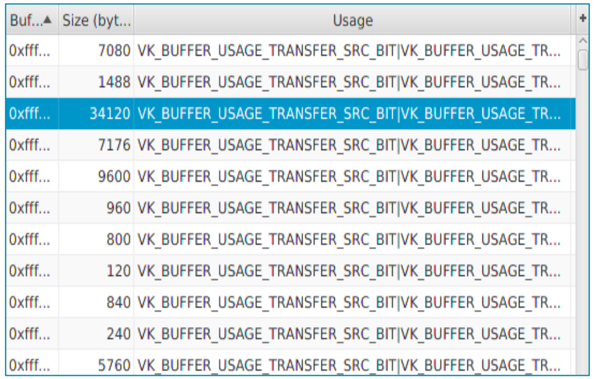
```
vkCreateSemaphore
(
    device = 0x9ff6d808,
    pCreateInfo =
    {
        sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO,
        pNext = NULL,
        flags = <>
    },
    pAllocator = NULL,
    pSemaphore = 0xaed81780
)
```

Function Parameters



Name	Value
▾ Vulkan	
Buffer Views	
▸ Buffers	
▾ Command Buffers	
▸ Command Buffer 0xab2e2004	
▸ Command Buffer 0x98c9f004	
▾ Command Buffer 0x9a89d004	
commandPool	0xffffffff98bb4734
device	0x9ad79000
inheritanceInfo	0 properties
level	VK_COMMAND_BUFFER_LEVEL_PRIMARY
usageFlags	
▸ Commands	19 items
▸ Command Buffer 0x9be7b004	
▸ Command Buffer 0x9c38b004	
▸ Command Buffer 0x9c4ac004	
▸ Command Buffer 0x9c4ae004	
▸ Command Buffer 0x9a272004	
▸ Command Buffer 0x99222004	
▸ Command Buffer 0x9a1d0004	
▸ Command Buffer 0x9a2bf004	
▸ Command Buffer 0x81e9730c	

Assets View



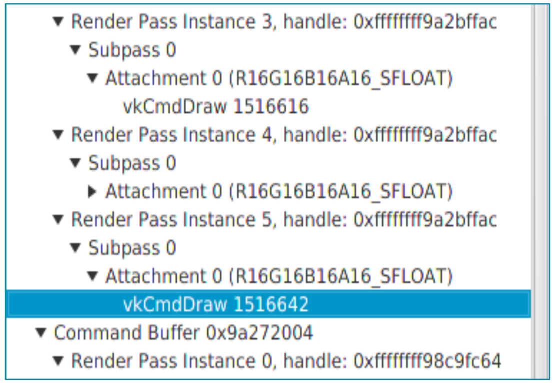
Buf... ▲	Size (byt...	Usage
0xfff...	7080	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	1488	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	34120	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	7176	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	9600	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	960	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	800	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	120	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	840	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	240	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...
0xfff...	5760	VK_BUFFER_USAGE_TRANSFER_SRC_BIT|VK_BUFFER_USAGE_TR...

Buffers

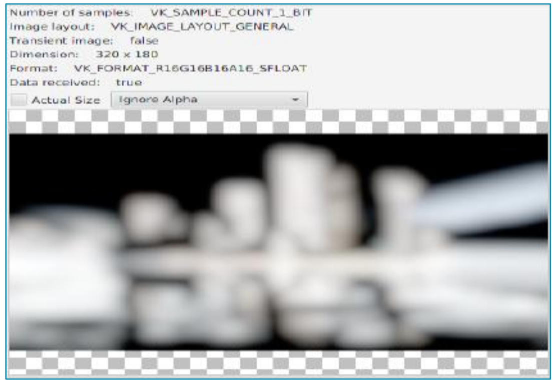
- Allows full API trace of Vulkan applications.
- Shows you easily all of the function parameters available.
- Implemented as a Vulkan layer which shows easy loading.
- Also shows Asset data and buffer information.
- Implemented perspectives to hide information that isn’t related to a particular API

Mali Graphics Debugger – Vulkan Frame Capture

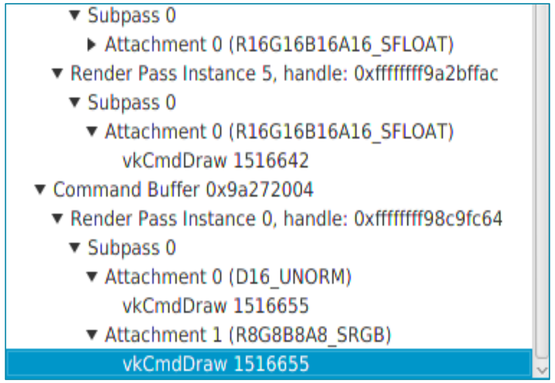
- MGD will now let you do a Frame Capture just like OpenGL ES.
- Frame Capture allows you to see how the scene is composed draw call by draw call.
- Great to catch any render abnormalities in the scene.



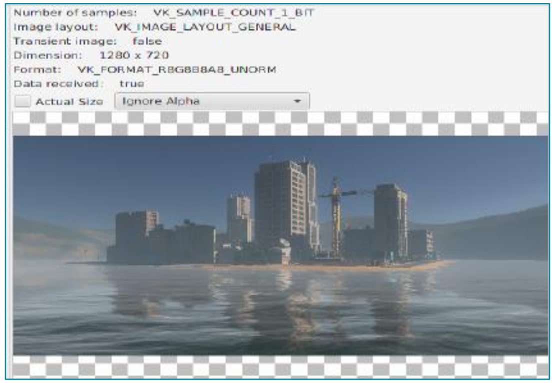
```
  ▼ Render Pass Instance 3, handle: 0xffffffff9a2bffac
    ▼ Subpass 0
      ▼ Attachment 0 (R16G16B16A16_SFLOAT)
          vkCmdDraw 1516616
  ▼ Render Pass Instance 4, handle: 0xffffffff9a2bffac
    ▼ Subpass 0
      ▶ Attachment 0 (R16G16B16A16_SFLOAT)
  ▼ Render Pass Instance 5, handle: 0xffffffff9a2bffac
    ▼ Subpass 0
      ▼ Attachment 0 (R16G16B16A16_SFLOAT)
          vkCmdDraw 1516642
▼ Command Buffer 0x9a272004
  ▼ Render Pass Instance 0, handle: 0xffffffff98c9fc64
```



Number of samples: VK_SAMPLE_COUNT_1_BIT
Image layout: VK_IMAGE_LAYOUT_GENERAL
Transient image: false
Dimension: 320 x 180
Format: VK_FORMAT_R16G16B16A16_SFLOAT
Data received: true



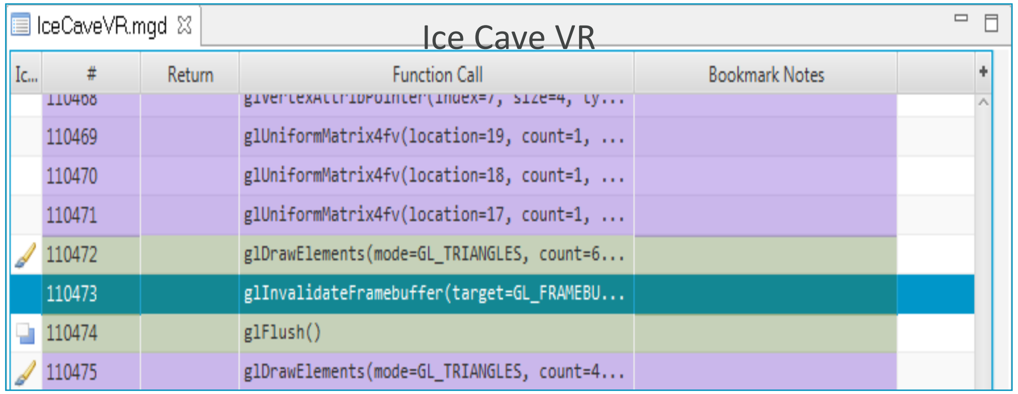
```
    ▼ Subpass 0
      ▶ Attachment 0 (R16G16B16A16_SFLOAT)
  ▼ Render Pass Instance 5, handle: 0xffffffff9a2bffac
    ▼ Subpass 0
      ▼ Attachment 0 (R16G16B16A16_SFLOAT)
          vkCmdDraw 1516642
▼ Command Buffer 0x9a272004
  ▼ Render Pass Instance 0, handle: 0xffffffff98c9fc64
    ▼ Subpass 0
      ▼ Attachment 0 (D16_UNORM)
          vkCmdDraw 1516655
      ▼ Attachment 1 (R8G8B8A8_SRGB)
          vkCmdDraw 1516655
```



Number of samples: VK_SAMPLE_COUNT_1_BIT
Image layout: VK_IMAGE_LAYOUT_GENERAL
Transient image: false
Dimension: 1280 x 720
Format: VK_FORMAT_R8G8B8A8_UNORM
Data received: true

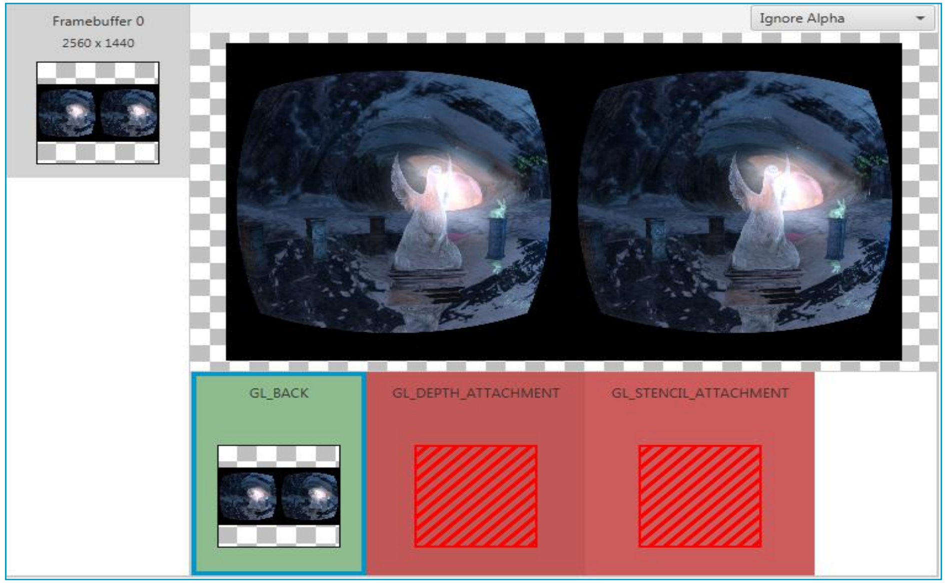
Mali Graphics Debugger – VR Update

- VR is becoming more and more popular with every month.
- MGD has supported VR with Oculus Gear VR and the Oculus SDK for over a year now.
- In V4.6 of MGD support for daydream has now been added and works as well as Samsung Gear VR.
- It works by using a series of heuristics to understand whether to treat glFlush as the end of Frame or not.
- Works with Unity, Unreal and internal demos



Ice Cave VR

Ic...	#	Return	Function Call	Bookmark Notes
	110468		glVertexAttribPointer(index=7, size=4, ty...	
	110469		glUniformMatrix4fv(location=19, count=1, ...	
	110470		glUniformMatrix4fv(location=18, count=1, ...	
	110471		glUniformMatrix4fv(location=17, count=1, ...	
	110472		glDrawElements(mode=GL_TRIANGLES, count=6...	
	110473		glInvalidateFramebuffer(target=GL_FRAMEBU...	
	110474		glFlush()	
	110475		glDrawElements(mode=GL_TRIANGLES, count=4...	

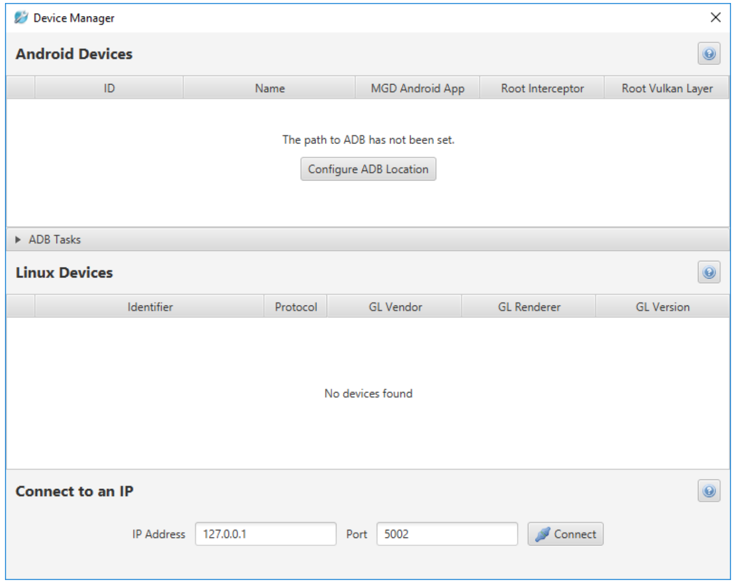


How to use MGD

Rooted Setup Mode

Step 1: Connection

- Tell MGD where adb is located on the system.



Device Manager

Android Devices

ID	Name	MGD Android App	Root Interceptor	Root Vulkan Layer

The path to ADB has not been set.

Configure ADB Location

▸ ADB Tasks

Linux Devices

Identifier	Protocol	GL Vendor	GL Renderer	GL Version

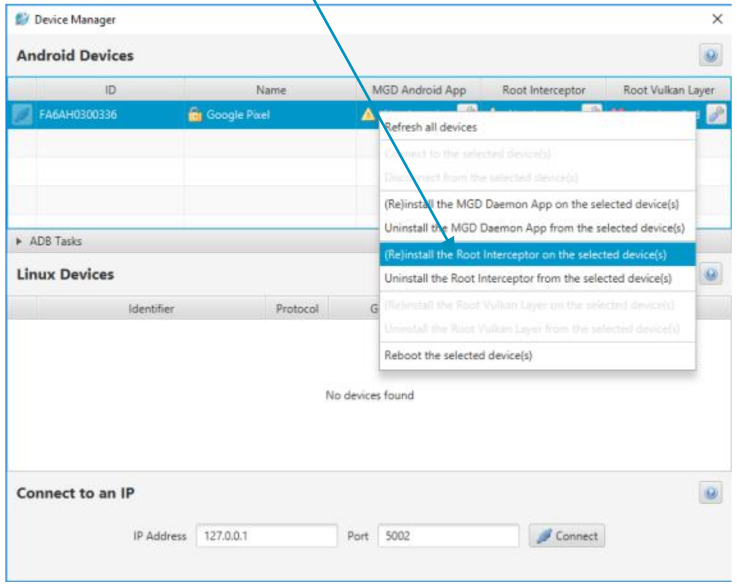
No devices found

Connect to an IP

IP Address 127.0.0.1 Port 5002 Connect

Step 2:

- Click on the device you want to install MGD onto



Device Manager

Android Devices

ID	Name	MGD Android App	Root Interceptor	Root Vulkan Layer
FA6AH0300336	Google Pixel			

Refresh all devices
Connect to the selected device(s)
Disconnect from the selected device(s)
(Re)install the MGD Daemon App on the selected device(s)
Uninstall the MGD Daemon App from the selected device(s)
(Re)install the Root Interceptor on the selected device(s)
Uninstall the Root Interceptor from the selected device(s)
(Re)install the Root Vulkan Layer on the selected device(s)
Uninstall the Root Vulkan Layer from the selected device(s)
Reboot the selected device(s)

▸ ADB Tasks

Linux Devices

Identifier	Protocol	G

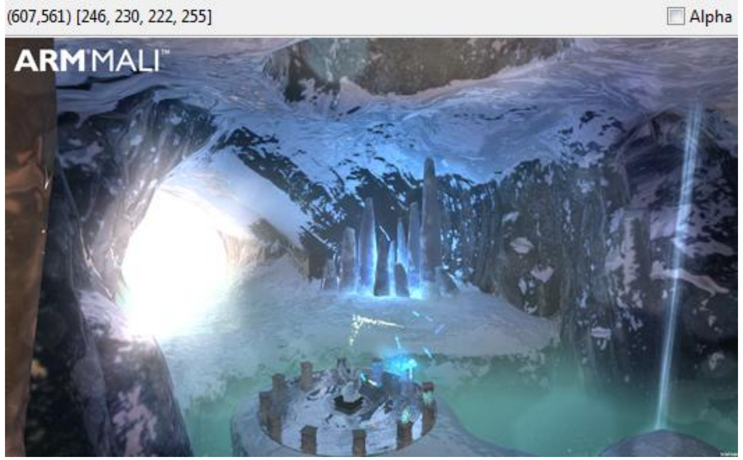
No devices found

Connect to an IP

IP Address 127.0.0.1 Port 5002 Connect

Step 3:

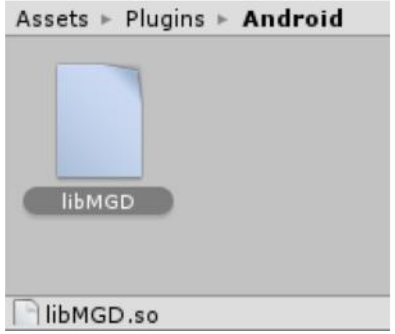
- Connect and then select the application you want to trace



(607,561) [246, 230, 222, 255] ☐ Alpha

Unity Setup

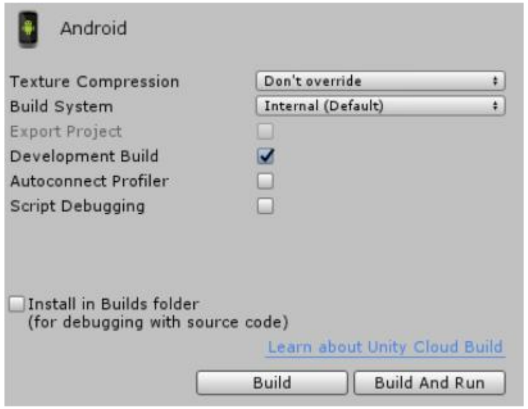
OpenGL ES



Assets ▸ Plugins ▸ **Android**

libMGD

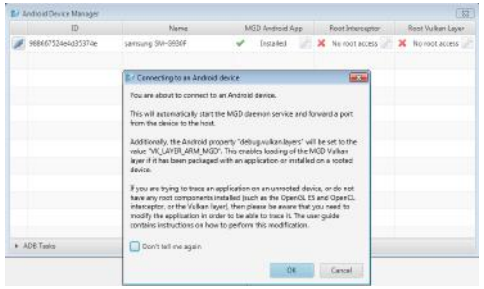
libMGD.so



Android

Texture Compression	Don't override
Build System	Internal (Default)
Export Project	□
Development Build	☑
Autoconnect Profiler	□
Script Debugging	□

□ Install in Builds folder
(for debugging with source code)



Android Device Manager

ID	Name	MGD Android App	Root Interceptor	Root Vulkan Layer
988467524e4d35374e	samsung SM-G930F	✓ Installed	✗ No root access	✗ No root access

Connecting to an Android device

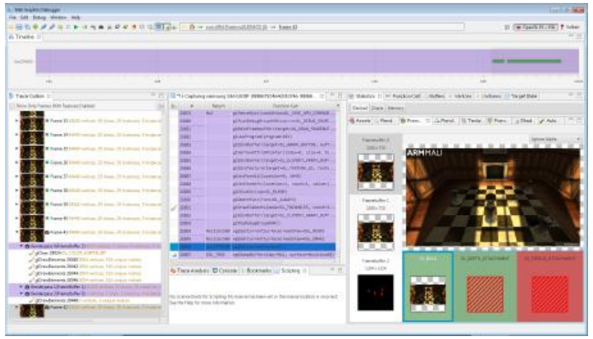
You are about to connect to an Android device.

This will automatically start the MGD daemon service and forward a port from the device to the host.

Additionally, the Android property "debug.vulkan.layers" will be set to the value "VK_LAYER_ARM_MGD". This enables loading of the MGD Vulkan layer if it has been packaged with an application or installed on a rooted device.

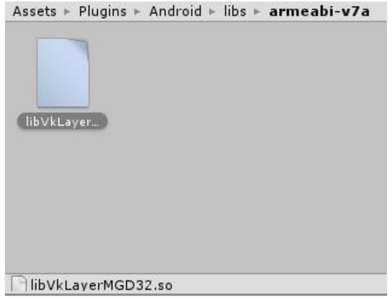
If you are trying to trace an application on an unrooted device, or do not have any root components installed (such as the OpenGL ES and OpenCL interceptor, or the Vulkan layer), then please be aware that you need to modify the application in order to be able to trace it. The user guide contains instructions on how to perform this modification.

□ Don't tell me again



Vulkan

Assets/Plugins/Android/libs/armeabi-v7a/

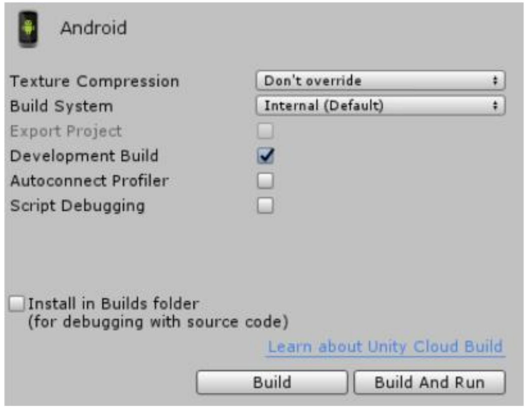


Assets ▸ Plugins ▸ Android ▸ libs ▸ **armeabi-v7a**

libVkLayer...

libVkLayerMGD32.so

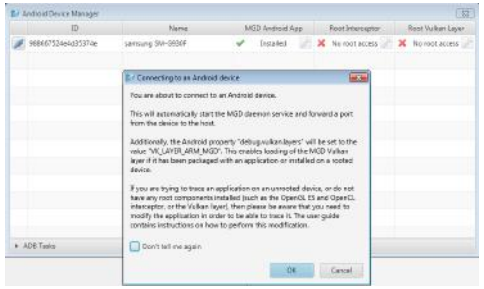
libVkLayerMGD32.so



Android

Texture Compression	Don't override
Build System	Internal (Default)
Export Project	□
Development Build	☑
Autoconnect Profiler	□
Script Debugging	□

□ Install in Builds folder
(for debugging with source code)



Android Device Manager

ID	Name	MGD Android App	Root Interceptor	Root Vulkan Layer
988467524e4d35374e	samsung SM-G930F	✓ Installed	✗ No root access	✗ No root access

Connecting to an Android device

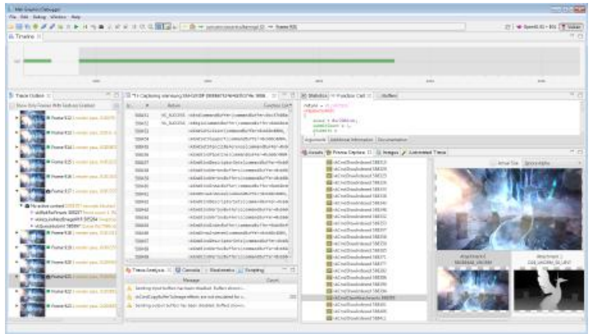
You are about to connect to an Android device.

This will automatically start the MGD daemon service and forward a port from the device to the host.

Additionally, the Android property "debug.vulkan.layers" will be set to the value "VK_LAYER_ARM_MGD". This enables loading of the MGD Vulkan layer if it has been packaged with an application or installed on a rooted device.

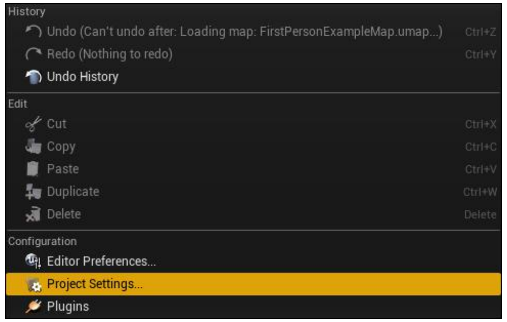
If you are trying to trace an application on an unrooted device, or do not have any root components installed (such as the OpenGL ES and OpenCL interceptor, or the Vulkan layer), then please be aware that you need to modify the application in order to be able to trace it. The user guide contains instructions on how to perform this modification.

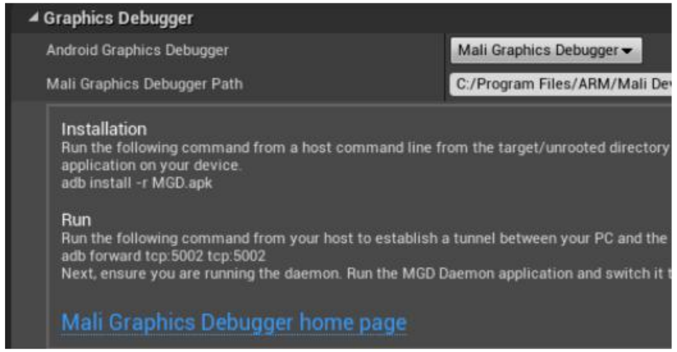
□ Don't tell me again



Unreal Setup

- Unreal is just as simple
- Just select project settings -> Android -> Graphics Debugger and select Mali Graphics Debugger.
- Give the location of your MGD installation and Unreal will do the rest.




Graphics Debugger

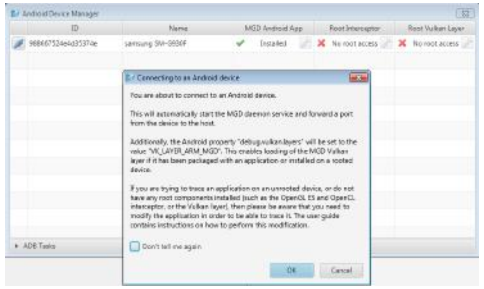
Android Graphics Debugger: Mali Graphics Debugger

Mali Graphics Debugger Path: C:/Program Files/ARM/Mali Dev

Installation
Run the following command from a host command line from the target/unrooted directory
application on your device.
adb install -r MGD.apk

Run
Run the following command from your host to establish a tunnel between your PC and the
adb forward tcp:5002 tcp:5002
Next, ensure you are running the daemon. Run the MGD Daemon application and switch it t

Mali Graphics Debugger home page


Android Device Manager

ID	Name	MGD Android App	Root Interceptor	Root Vulkan Layer
988K67524e4d35374e	samsung SM-G930F	✓ Installed	✗ No root access	✗ No root access

ADB Tasks

Connecting to an Android device

You are about to connect to an Android device.

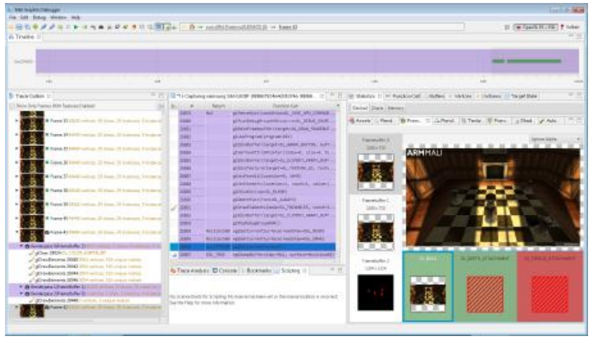
This will automatically start the MGD daemon service and forward a port from the device to the host.

Additionally, the Android property "debug.vulkan.layers" will be set to the value "VK_LAYER_ARM_MGD". This enables loading of the MGD Vulkan layer if it has been packaged with an application or installed on a rooted device.

If you are trying to trace an application on an unrooted device, or do not have any root components installed (such as the OpenGL ES and OpenCL interceptor, or the Vulkan layer), then please be aware that you need to modify the application in order to be able to trace it. The user guide contains instructions on how to perform this modification.

□ Don't tell me again

OK Cancel



How to Use Mali Graphics Debugger

1) Connect Mali Graphics Debugger from the beginning of your application

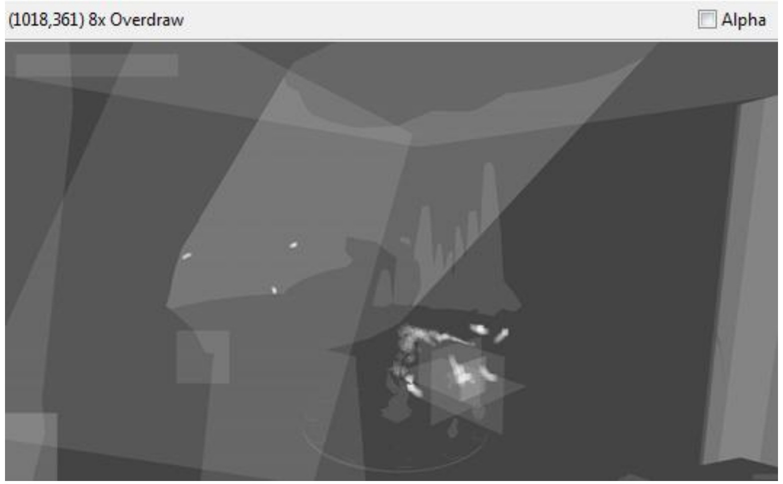
2) Get to the perceived problem area in your application

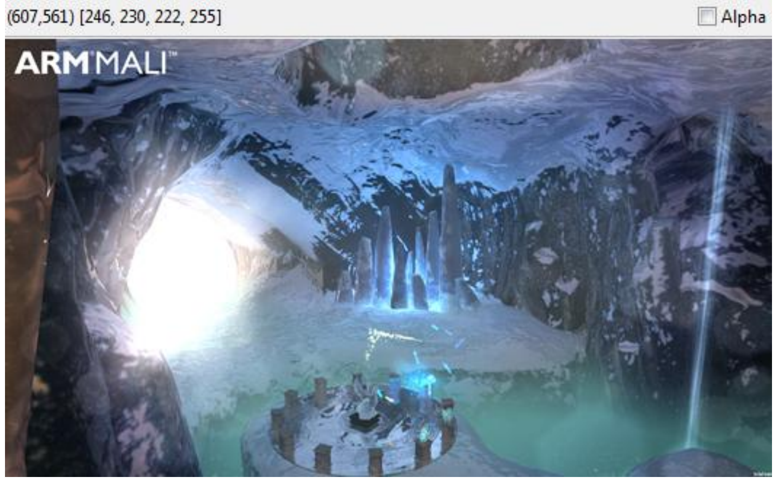
3) Pause the application

4) Run a frame capture, overdraw capture, shader map and fragment count

5) Disconnect the application as you have everything you need

Checking GPU useless jobs – excess fragments

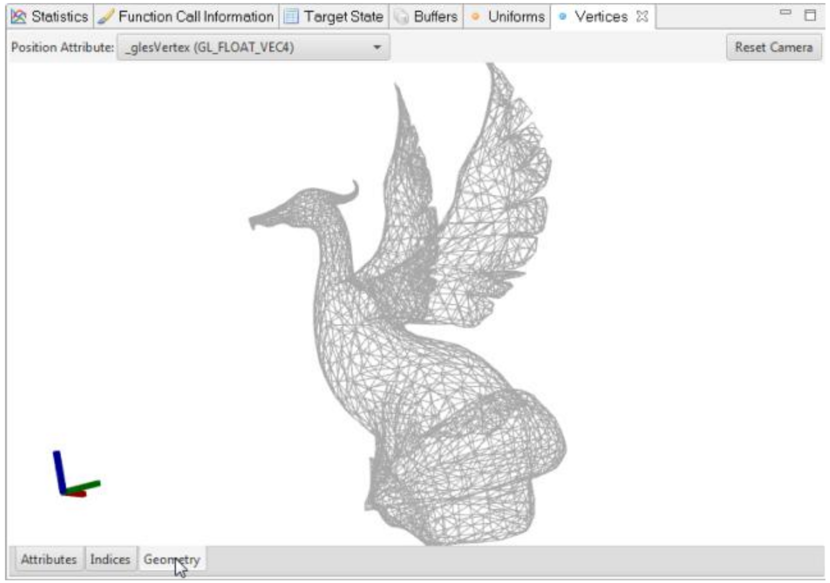




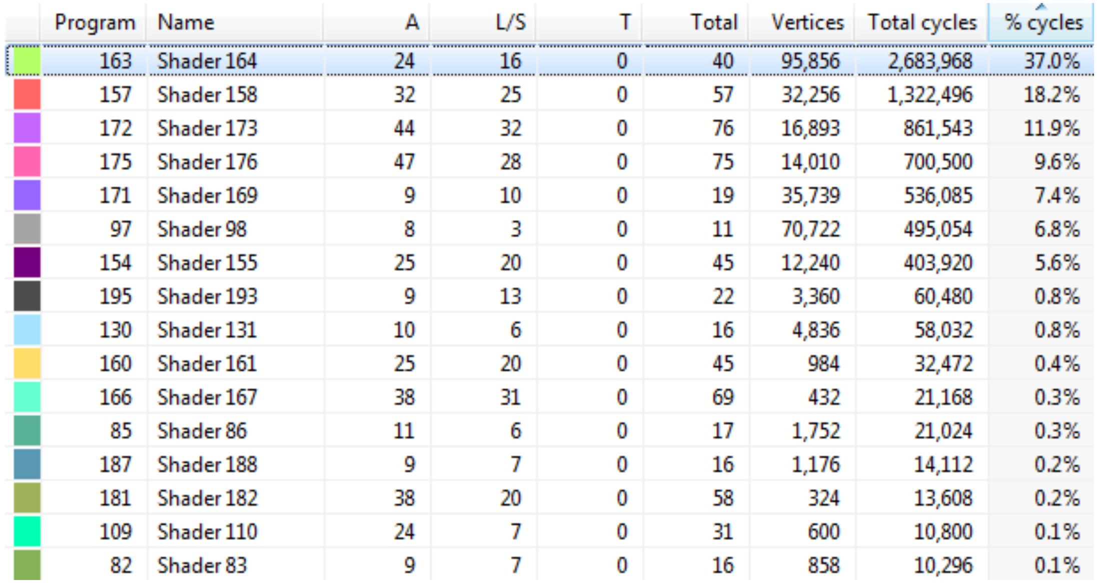
- Overdraw mode will tell you how many times the Same pixel was drawn to in a single Frame.
- Ideally you should only draw to the screen once per Frame.
- Any more this and you are wasting GPU cycles.
- Sometimes you can't help this though: Clearing the screen, doing some blending effects or transparency.

- Frame capture captures the state of all Framebuffers after every draw call.
- It is a great way to see how each draw call contributes to the scene and how to match rendering to the screen with a particular draw command.
- If you click on a draw call and you can't see any addition to the screen, you should consider removing the draw call as it is extra work an isn't making any visual difference.

Vertex shading and excess vertices



Position Attribute: _glesVertex (GL_FLOAT_VEC4)

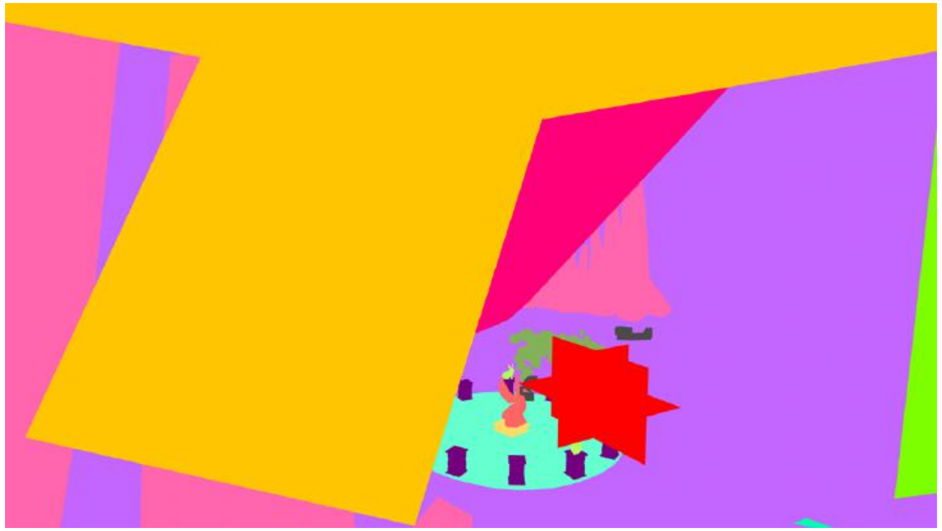


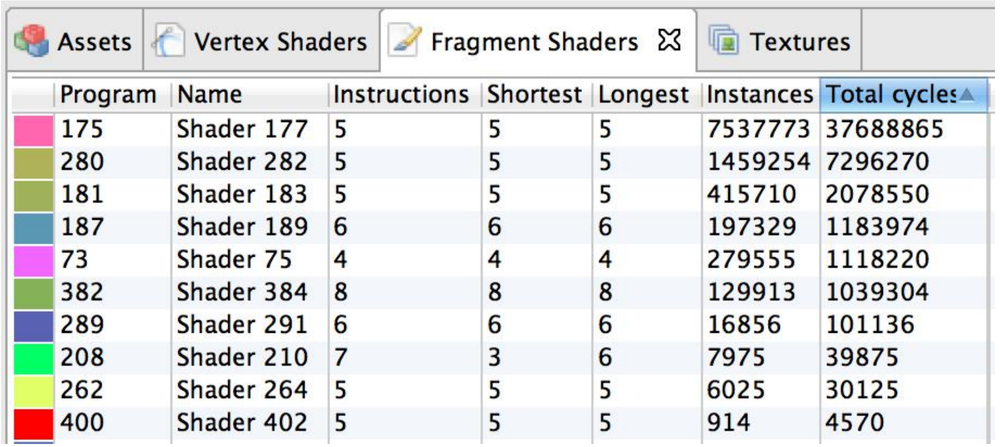
	Program	Name	A	L/S	T	Total	Vertices	Total cycles	% cycles
🟩	163	Shader 164	24	16	0	40	95,856	2,683,968	37.0%
🟥	157	Shader 158	32	25	0	57	32,256	1,322,496	18.2%
🟪	172	Shader 173	44	32	0	76	16,893	861,543	11.9%
🟥	175	Shader 176	47	28	0	75	14,010	700,500	9.6%
🟪	171	Shader 169	9	10	0	19	35,739	536,085	7.4%
⬜	97	Shader 98	8	3	0	11	70,722	495,054	6.8%
🟪	154	Shader 155	25	20	0	45	12,240	403,920	5.6%
⬛	195	Shader 193	9	13	0	22	3,360	60,480	0.8%
🟦	130	Shader 131	10	6	0	16	4,836	58,032	0.8%
🟨	160	Shader 161	25	20	0	45	984	32,472	0.4%
🟩	166	Shader 167	38	31	0	69	432	21,168	0.3%
🟩	85	Shader 86	11	6	0	17	1,752	21,024	0.3%
🟦	187	Shader 188	9	7	0	16	1,176	14,112	0.2%
🟩	181	Shader 182	38	20	0	58	324	13,608	0.2%
🟩	109	Shader 110	24	7	0	31	600	10,800	0.1%
🟩	82	Shader 83	9	7	0	16	858	10,296	0.1%

- The geometry viewer is great to see the complexity of the model you are using.
- This is a great way to see if the model you are using is fitting to its position in the scene.
- For instance this particular model wouldn't be recommends for use in the distance as it is too detailed.

- Mali Graphics Debugger automatically runs all vertex shaders through the offline compiler.
- You can then easily rank the shaders in how many cycles they contribute to the scene.
- The higher up they are in this table the more time should be spent optimizing them.

Shadermap and Fragment Count





	Program	Name	Instructions	Shortest	Longest	Instances	Total cycles
🟪	175	Shader 177	5	5	5	7537773	37688865
🟫	280	Shader 282	5	5	5	1459254	7296270
🟫	181	Shader 183	5	5	5	415710	2078550
🟦	187	Shader 189	6	6	6	197329	1183974
🟪	73	Shader 75	4	4	4	279555	1118220
🟩	382	Shader 384	8	8	8	129913	1039304
🟦	289	Shader 291	6	6	6	16856	101136
🟩	208	Shader 210	7	3	6	7975	39875
🟨	262	Shader 264	5	5	5	6025	30125
🟥	400	Shader 402	5	5	5	914	4570

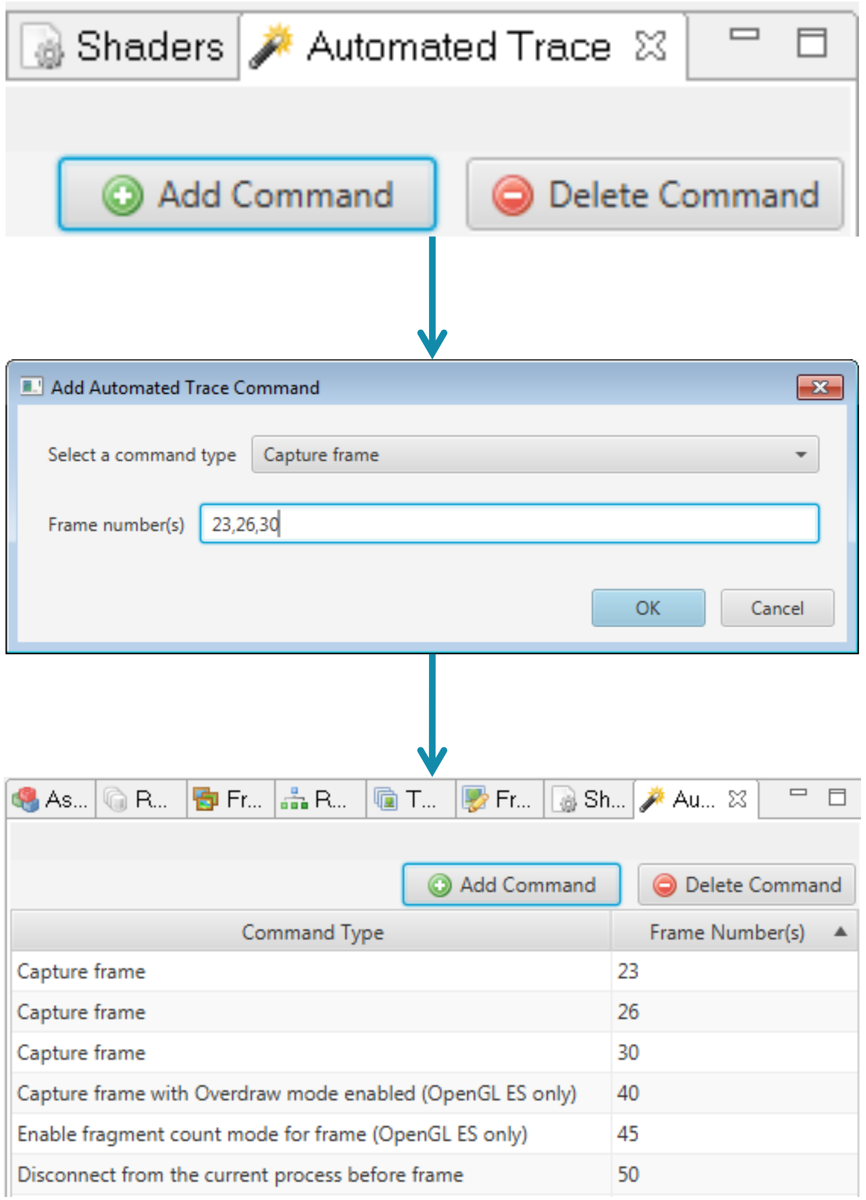
- For fragment shading we don't know how many fragments were rasterized
- This presents a problem when trying to show which shaders contributed to the scene the most.
- Using the "Fragment count" feature we can get this information.
- We can also use the "Shadermap" feature to see where on the framebuffer a particular shader drew.

Automated Trace - Introduction

- Sometimes the user knows exactly at what point they want to capture a Frame in MGD or use any of the other MGD features.
- This can be many minutes into a trace.
- The user can now set MGD to automatically do captures on a numbered Frame. When MGD traces this frame it will automatically do the desired function.
- This frees the user up to do other things while MGD generates their perfect trace for them.
- It can even disconnect the device when finished so there is no extra data captured.

How to use it

- **Step 1**: Start your trace normally
- **Step 2**: Pause the trace when tracing information starts being provided
- **Step 3**: Select the automated trace dialog and then select add command
- **Step 4**: Select the Automated trace dialog and then select add command
- **Step 5**: Select the feature you want to use and then give a comma selected list of frames you want that feature to be active for
- **Step 6**: Continue to run the application



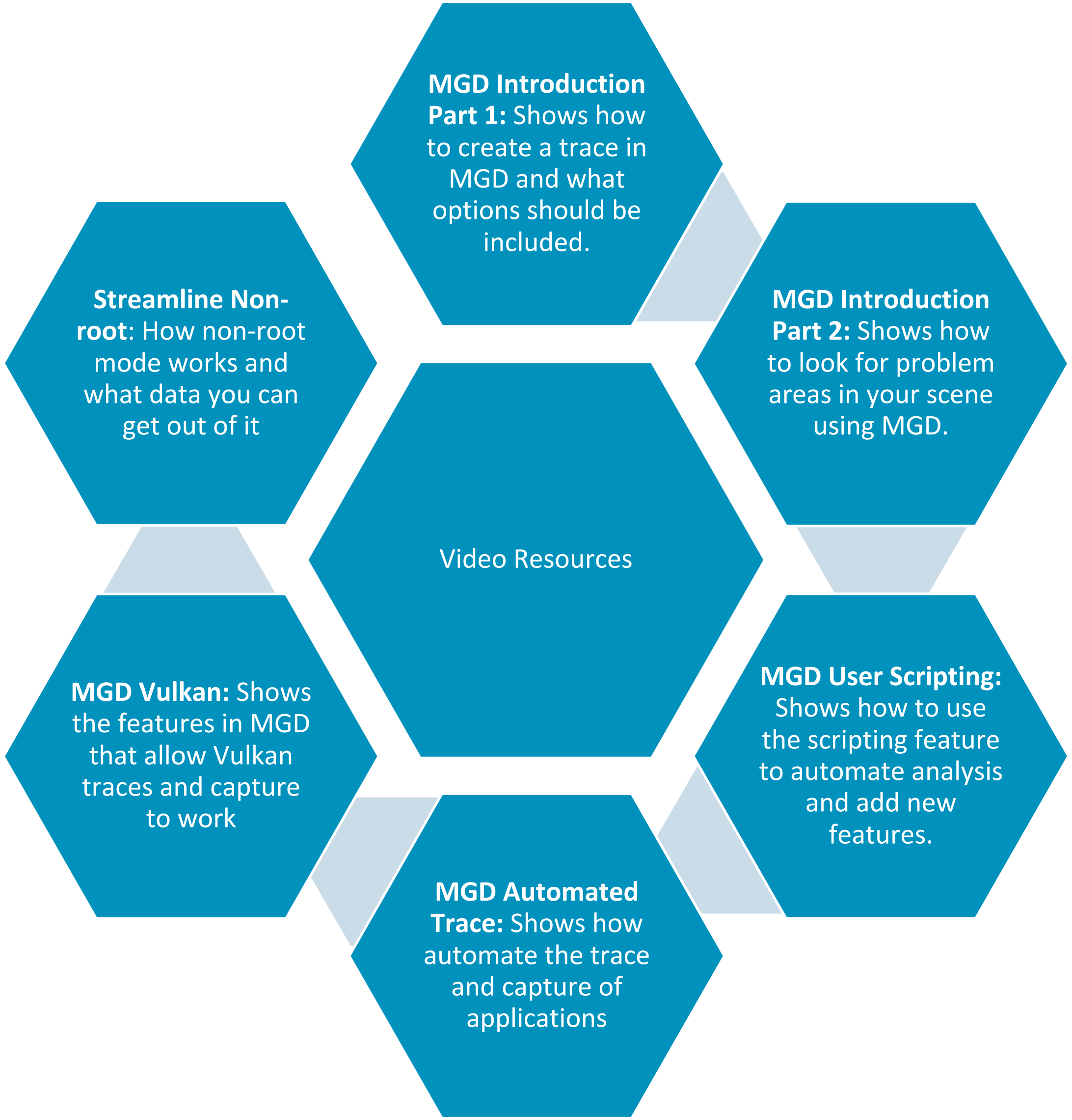
Select a command type: Capture frame

Frame number(s): 23,26,30

Command Type	Frame Number(s)
Capture frame	23
Capture frame	26
Capture frame	30
Capture frame with Overdraw mode enabled (OpenGL ES only)	40
Enable fragment count mode for frame (OpenGL ES only)	45
Disconnect from the current process before frame	50

Resources

https://developer.arm.com/products/software-development-tools/ds-5-development-studio/resources/ds-5-media-articles



Video Resources

MGD Introduction Part 1: Shows how to create a trace in MGD and what options should be included.

MGD Introduction Part 2: Shows how to look for problem areas in your scene using MGD.

MGD User Scripting: Shows how to use the scripting feature to automate analysis and add new features.

MGD Automated Trace: Shows how automate the trace and capture of applications

MGD Vulkan: Shows the features in MGD that allow Vulkan traces and capture to work

Streamline Non-root: How non-root mode works and what data you can get out of it