

OpenGL ES Tuning and Optimization

More frames per second

Alex Kan and Jean-François Roy GPU Software

Session Overview

- OpenGL ES Analyzer
- Tuning the graphics pipeline
- Analyzer demo

OpenGL ES Analyzer Instrument Developer preview

Jean-François Roy GPU Software Developer Technologies

Ś I	Instruments File Edit View	Instrument	Window Help				🕓 🕴 🎅 🖌 💽 Tue 10	0:25 AM Q
							and the second se	
	000				Instruments	2		
							0.100	
		wei_1 •			Bun 1 of		Ge instrument Detail	
	Record Targe	et .	Inspecti	on Range	itali 1 or .	View Library	Search	
	Instruments	00:00						
				and the second states				
	Openicit is Analyzer 7	# Redundant	State C Unoptimized MS	AA Resolve				
			Suboptimal MSA	A buffer resolve ope	eration 🔘			
	· · · · · · · · · · · · · · · · · · ·	# GL Calls						
						and the production of the second s		
	and the second se	# Eatches			Done			
					Done			
		# Triangles R	endered					\prec
	DenGL ES							
	I I]						
	🖾 OpenGL ES Analyzer	🗧 🖽 Frame Sta	atistics 🗢					
	Verrides	Frame #▲	# Triangles Rendered	# Batches	# GL Calls	# Redundant State Changes		
	 None 	0	12	6	365	3		
	O Eliminate all OpenGL ES operations	1	12	6	48	0		
	Eliminate rendering operations only	2	12	6	48	0		18/7/8
	Simplify all shader processing	4	12	6	48	0		
	O Minimize utilized texture handwidth	5	12	6	48	0		
	C mininize utilized texture bundwidth					0		
	Minimize number of pixels rendered	6	12	6	48	ő		
	Minimize number of pixels rendered Call Tree	6	12	6	48 48	0		
	Call Tree Separate by Category	6 7 8	12 12 12	6 6 6	48 48 48	0		
	Call Tree Call Call Call Call Call Call Call Call	6 7 8 9	12 12 12	6 6 6	48 48 48 48	0		
	Minimize number of pixels rendered Call Tree Separate by Category Separate by Thread Invert Call Tree Mide Miscines Sembols	6 7 8 9 10	12 12 12 12 12 12	6 6 6 6	48 48 48 48 48 48 48			
	Minimize number of pixels rendered Minimize number of pixels rendered Call Tree Separate by Category Separate by Thread Invert Call Tree Hide Missing Symbols Hide Symbols	6 7 8 9 10 11 12	12 12 12 12 12 12 12 12 12	6 6 6 6 6	48 48 48 48 48 48 48 48 48			
	Minimize number of pixels rendered Genaric by Category Separate by Category Separate by Thread Hide Missing Symbols Hide System Libraries Show Oh-C Only	6 7 8 9 10 11 12 13	12 12 12 12 12 12 12 12 12 12	6 6 6 6 6 6	48 48 48 48 48 48 48 48 48 48			
	Minimize number of pixels rendered Call Tree Separate by Category Separate by Thread Invert Call Tree Hide Missing Symbols Hide System Libraries Show Obj-C Only Flatter Recursion	6 7 8 9 10 11 12 13 14	12 12 12 12 12 12 12 12 12 12 12	6 6 6 6 6 6 8	48 48 48 48 48 48 48 48 48 55			
	Minimize number of pixels rendered Call Tree Separate by Thread Invert Call Tree Hide Missing Symbols Hide System Libraries Show Obj-C Only Flatten Recursion Call Tree Constraints	6 7 8 9 10 11 12 13 14 15	12 12 12 12 12 12 12 12 12 12 16 16	6 6 6 6 6 8 8	48 48 48 48 48 48 48 48 55 55			





Available on the Attendees Site

OpenGL ES Analyzer Instrument Developer preview



- Available as a developer preview with Xcode 4
 - Preview only supports PowerVR SGX devices with iOS 4
 - Preview cannot attach to a running application
- We want your feedback!

Why an Instrument?

OpenGL ES Analyzer Instrument The power of many

- Correlate data from multiple instruments
- Powerful data mining
- Well-understood tool and interface



Activity Monitor

- Traces all OpenGL ES activity
- Provides key statistics

Overrides

- Disable specific parts of the graphics pipeline
- Helpful for finding bottlenecks





Measuring OpenGL ES Activity

Measuring OpenGL ES Activity Understanding the problem

- Your application may be doing...
 - A lot more work than you thought
 - Work in an unexpected order
 - The wrong kind of work



Measuring OpenGL ES Activity Activity monitor

- Records a trace of all OpenGL ES activity
- Four main hubs of information
 - Frame statistics
 - API statistics
 - Command trace
 - Call tree



Measuring OpenGL ES Activity Frame statistics

- Get an idea of your per-frame workload
- Navigate the trace by frame
- Select range of frames in the timeline

🗄 Frame Sta	atistics 🗢				
Frame # 🔺	# Triangles Rendered	# Batches	# GL Calls	# Redundant State Changes	# Render Passes
0	25296	6	1220	32	1
1	41732	8	239	65	1
2	41732	8	218	65	1
3	41732	8	218	65	1
4	41732	8	218	69	1
5	41732	8	218	69	1
6	41732	8	218	69	1
7	41732	8	218	69	1
8	41732	8	218	69	1
9	41732	8	218	69	1
10	41732	8	218	69	1
11	41732	8	218	69	1
12	41732	8	218	69	1
13	41732	8	218	69	1
14	41732	8	218	69	1
15	41732	8	218	69	1
16	41732	8	218	69	1
17	41732	8	218	69	1
18	41732	8	218	69	1
19	41732				_

Primitives and batches

- Maximize ratio of primitives to batches (draw commands)
- Minimize batches
- Find your most costly frame w/r to geometry

🛛 🎛 Frame S	tatistics 🗢				
Frame #	# Triangles Rendered 🔺	# Batches 🔺	# GL Calls	# Redundant State Changes	# Render Passes
0	25296	6	1220	32	1
1	41732	8	239	65	1
2	41732	8	218	65	1
3	41732	8	218	65	1
4	41732	8	218	69	1
5	41732	8	218	69	1
6	41732	8	218	69	1
7	41732	8	218	69	1
8	41732	8	218	69	1
9	41732	8	218	69	1
10	41732	8	218	69	1
11	41732	8	218	69	1
12	41732	8	218	69	1
13	41732	8	218	69	1
14	41732	8	218	69	1
15	41732	8	218	69	1
16	41732	8	218	69	1
17	41732	8			1
18	41732	8			
	41732				

OpenGL commands

- Minimize how many commands you issue per frame
- Ratio of batches to GL commands should approach 1
 - If small, try sorting the geometry by state

🛛 🎛 Frame Sta	itistics 🗢				
Frame #	# Triangles Rendered	# Batches 🔺	# GL Calls ▲	# Redundant State Changes	# Render Passes
0	25296	6	1220	32	1
1	41732	8	239	65	1
2	41732	8	218	65	1
3	41732	8	218	65	1
4	41732	8	218	69	1
5	41732	8	218	69	1
6	41732	8	218	69	1
7	41732	8	218	69	1
8	41732	8	218	69	1
9	41732	8	218	69	1
10	41732	8	218	69	1
11	41732	8	218	69	1
12	41732	8	218	69	1
13	41732	8	218	69	1
14	41732	8	218	69	1
15	41732	8	218	69	1
16	41732	8	218	69	1
17	41732	8	218		1
	41732	8	218		

Redundant state changes

- How many commands were issued to change state to the same value
 - Trigger work in the driver nonetheless
- Don't do them!

🛛 🎛 Frame Sta	atistics 🗢				
Frame #	# Triangles Rendered	# Batches	# GL Calls	# Redundant State Changes 🔺	# Render Passes
0	25296	6	1220	32	
1	41732	8	239	65	
2	41732	8	218	65	
3	41732	8	218	65	
4	41732	8	218	69	
5	41732	8	218	69	
6	41732	8	218	69	
7	41732	8	218	69	
8	41732	8	218	69	
9	41732	8	218	69	
10	41732	8	218	69	
11	41732	8	218	69	
12	41732	8	218	69	
13	41732	8	218	69	
14	41732	8	218	69	
15	41732	8	218	69	
16	41732	8	218	69	
17	41732	8	218	69	
				69	

Render passes

- The graphics hardware operates in render passes
- Aim for only one render pass per frame
- Some commands can force the hardware to end the current pass

🛛 🎛 Frame S	tatistics 🗢				
Frame #	# Triangles Rendered	# Batches	# GL Calls	# Redundant State Changes	# Render Passes 4
0	25296	6	1220	32	1
1	41732	8	239	65	1
2	41732	8	218	65	1
3	41732	8	218	65	1
4	41732	8	218	69	1
5	41732	8	218	69	1
6	41732	8	218	69	1
7	41732	8	218	69	1
8	41732	8	218	69	1
9	41732	8	218	69	1
10	41732	8	218	69	1
11	41732	8	218	69	1
12	41732	8	218	69	1
13	41732	8	218	69	1
14	41732	8	218	69	1
15	41732	8	218	69	1
16	41732	8	218	69	1
17	41732	8		69	1
					1

Measuring OpenGL ES Activity API statistics

- API statistics allow you to see what commands
 - Are used most frequently
 - Cost the most

🖽 API Statistics 🗢			
OpenGL ES Function	Count	Total Time (µs) 🔻	Average Time (µs)
glClear	954	5025619	487160
glDrawElements	7622	1683719	24750
EAGLPresentRenderBuffer	953	1092747	104377
glUseProgram	10488	203210	1720
glBindBuffer	40132	188810	370
glUniformMatrix4fv	17150	155419	768
glBufferData	113	146701	1298
glCompileShader	8	134188	16773
glVertexAttribPointer	23820	115560	414
glBindTexture	23843	113148	399
glDiscardFramebufferEXT	953	100226	9458
glEnableVertexAttribArray	23820	68953	223
glUniform4f	5716	65583	1000
glActiveTexture	23821	65155	208
alUniform4fy	- 954	39133	3691

Cost in time

- Ideally, draw commands should dominate after loading
- Simple commands with can be costly when called frequently
- Minimize usage of expensive commands during gameplay or do them on another thread

API Statistics 🗢			
OpenGL ES Function	Count		Average Time (µs)
glClear	954	5025619	487160
glDrawElements	7622	1683719	24750
EAGLPresentRenderBuffer	953	1092747	104377
glUseProgram	10488	203210	1720
glBindBuffer	40132	188810	370
glUniformMatrix4fv	17150	155419	768
glBufferData	113	146701	1298
glCompileShader	8	134188	16773
glVertexAttribPointer	23820	115560	414
glBindTexture	23843	113148	399
glDiscardFramebufferEXT	953	100226	9458

Cost in time

- Ideally, draw commands should dominate after loading
- Simple commands can be costly when called frequently
- Minimize usage of expensive commands during gameplay or do them on another thread

API Statistics			
OpenGL ES Function	Count	Total Time (us)	Average Time (us)
glClear	1101	5693504	551033
EAGLPresentRenderBuffer	1100	1290681	123059
EAGLInitWithAPIProperties	1	32043	32043
glDrawElements	8800	1848703	26701
glCompileShader	8	134188	16773
glDiscardFramebufferEXT	1100	115167	10876
EAGLRenderbufferStorageFromDrawable	2	11360	5680
glLinkProgram	5	22830	4566
glUniform4fv	1101	45596	4302
glBindFramebuffer	1105	28533	2636
glCheckFramebufferStatus	2	5160	2580
glUseProgram	12107	244640	2071
glBufferData	113	146701	1298

Frequency

- Make sure you're not issuing commands more than you need to
 - Likely that many of them are redundant
- Use performance extensions to reduce command count

E API Statistics 🗢			
OpenGL ES Function	Count 🔻	Total Time (µs)	Average Time (µs)
glBindBuffer	42342	202249	397
glBindTexture	25158	125878	446
glVertexAttribPointer	25138	119847	429
glEnableVertexAttribArray	25138	74719	242
glActiveTexture	25136	67135	213
glUniformMatrix4fv	18099	164720	813
glUseProgram	11066	216278	1831
glDrawElements	8043	1742220	25440
glGetFloatv	6034	21129	284
glUniform4f	6033	68349	1040
	4050	13859	
	4023	- 14420	
	3016	24012	
		17587	

Time range filtering

- API statistics are re-computed to match the selected time range
 - Use the frame statistics table to select a range of frames
 - Region of interest highlighted by other instruments

API Statistics 🗘			
OpenGL ES Function	Count	Total Time (µs) 🔻	Average Time (µs)
glClear	954	5025619	487160
glDrawElements	7622	1683719	24750
EAGLPresentRenderBuffer	953	1092747	104377
glUseProgram	10488	203210	1720
glBindBuffer	40132	188810	370
glUniformMatrix4fv	17150	155419	768
glBufferData	113	146701	1298
glCompileShader	8	134188	16773
glVertexAttribPointer	23820	115560	414
glBindTexture	23843	113148	399
glDiscardFramebufferEXT	953	100226	9458
glEnableVertexAttribArray	23820	68953	223
glUniform4f	5716	65583	1000
glActiveTexture	23821	65155	
alUniform4fv	954	39133	

Measuring OpenGL ES Activity Command trace

- The complete list of every OpenGL command issued by your application
- Useful in conjunction with the other hubs

田 Trace	◆ \race
	Trace
0	EAGLInitWithAPIProperties(0x00000000000533940, 2ul, 0x00533950, 0x00533a40)
1	glGetString(GL_EXTENSIONS)
2	glGetIntegerv(GL_RENDERBUFFER_BINDING, 0x2fffd6bc)
3	glGetIntegerv(GL_DRAW_FRAMEBUFFER_BINDING_EXT, 0x2fffd6b8)
4	glGenRenderbuffers(1, {1u})
5	glBindRenderbuffer(GL_RENDERBUFFER, 1u)
6	[0x00000000533940 renderbufferStorage:GL_RENDERBUFFER fromDrawable:0x0051e270]
7	glGenFramebuffers(1, {1u})
8	glBindFramebuffer(GL_FRAMEBUFFER, 1u)
9	gIFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENTO, GL_RENDERBUFFER, 1u)
10	glGenRenderbuffers(1, {2u})
11	glBindRenderbuffer(GL_RENDERBUFFER, 2u)
12	glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, 320, 480)
13	glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 2u)
14	glGetError()
15	glCheckFramebufferStatus(GL_FRAMEBUFFER)
16	glBindFramebuffer(GL_FRAMEBUFFER, 0u)
17	glBindRenderbuffer(GL_RENDERBUFFER, 0u)
18	glGetError()
19	glGetIntegerv(GL_MAX_TEXTURE_SIZE, 0x2fffd63c)
	alCatintagen/CL_MAY_TEXTURE_SIZE_0v2fffd6e8)

Measuring OpenGL ES Activity Call tree

- Standard Instruments call tree view but focused onOpenGL commands
- Allows you to see which OpenGL commands took the most amount of time and from where they were called

E Call Trees ♦		Call Tree						
Running Time		Running Count			Symbol Name			
4463.7ms	52.5%	845	0.4%	-	▶glClear OpenGLES →			
1554.9ms	18.3%	6756	3.6%	-	▶glDrawElements OpenGLES			
962.9ms	11.3%	844	0.4%	C	CA::Display::DisplayLink::dispatch(unsigned long long, unsigned long long) QuartzCore			
182.2ms	2.1%	9295	4.9%	-	▶glUseProgram OpenGLES			
171.5ms	2.0%	35586	19.1%	-	▶glBindBuffer OpenGLES			
146.7ms	1.7%	113	0.0%	-	▶glBufferData OpenGLES			
140.1ms	1.6%	15203	8.1%	-	▶glUniformMatrix4fv OpenGLES			
134.2ms	1.5%	8	0.0%	-	▶glCompileShader OpenGLES			
105.6ms	1.2%	21116	11.3%	-	▶glVertexAttribPointer OpenGLES			
104.2ms	1.2%	21136	11.3%	-	▶glBindTexture OpenGLES			
91.6ms	1.0%	844	0.4%	-	▶glDiscardFramebufferEXT OpenGLES			
63.9ms	0.7%	21116	11.3%	-	▶glEnableVertexAttribArray OpenGLES			
60.6ms	0.7%	21114	11.3%	-	▶glActiveTexture OpenGLES			
59.8ms	0.7%	5068	2.7%	-	▶glUniform4f OpenGLES			
34.1ms	0.4%	845	0.4%		▶glUniform4fv OpenGLES			
23.1ms	0.2%	2533	1.3%		▶glDepthMask OpenGLES			
22.8ms	0.2%	5	0.0%		▶glLinkProgram OpenGLES			
21.6ms	0.2%	106			▶glCompressedTeximage2D_OpenGLES			







Tuning the Graphics Pipeline

Alex Kan Embedded Graphics Acceleration

How a Frame Is Rendered The basics

- CPU encodes rendering commands for GPU
- GPU reads and processes vertices
- GPU shades fragments for primitives
- Core Animation composites rendered results to framebuffer

СРИ	Vertex	Fragment	CA

• Not all stages take the same amount of time

CPU	Vertex	Fragment	СА

• Not all stages take the same amount of time

CPU	Vertex	Fragment	СА

- Not all stages take the same amount of time
- Most stages can run in parallel with each other



- Not all stages take the same amount of time
- Most stages can run in parallel with each other


How a Frame Is Rendered Pipelining

- Not all stages take the same amount of time
- Most stages can run in parallel with each other



How a Frame Is Rendered Bottlenecks

- The slowest stage determines how long your frame takes
- Optimizing the slowest stage will produce the largest gains



Finding Bottlenecks Using Overrides Strategy

- Examine CPU/GPU utilization with CPU Sampler and OpenGL ES Driver instruments
- Apply various combinations of overrides with OpenGL ES Analyzer to see the effect on performance





Optimizing the Pipeline Stage by stage

Optimizing the Pipeline Topics

- CPU bottlenecks
- GPU bottlenecks
 - Bandwidth, computation, and workload size
 - Vertex bottlenecks
 - Fragment bottlenecks

Identification and classification

- GPU utilization well below 100%
- Framerate unchanged when simplifying draw calls
- CPU mostly in GL framework, in either of two states:
 - Fully utilized
 - Frequently blocked

High CPU utilization

- Usual culprit is handling state changes
 - Analyzer state statistics can tell you if state changes are redundant
- Some state changes are more expensive than others

E Frame Statistics 🗢												
Frame # 🔺	# Triangles Rendered	# Batches	# GL Calls	# Redundant State Changes	# Render Passes							
0	25296	6	1220	32	1							
1	41732	8	239	65	1							
2	41732	8	218	65	1							
3	41732	8	218	65	1							
4	41732	8	218	69	1							
5	41732	8	218	69	1							
6	41732	8	218	69	1							
7	41732	8	218	69	1							
8	41732	8	218	69	1							
9	41732	8	218	69	1							
10	41732	8	218	69	1							
11	41732	8	218	69	1							
12	41732	8	218	69	1							
13	41732	8	218	69	1							
14	41732	8	218	69	1							
15	41732	8	218	69	1							
16	41732	8	218	69	1							

Reducing CPU utilization

- Minimize state changes
- Take advantage of per-object state
 - Textures
 - Programs
 - Vertex arrays

Pipeline stalls

- CPU may need to synchronize to access/modify an object in flight
 - Wait as long as possible to retrieve rendering results
 - Avoid modifying textures and VBOs after using them in a frame
 - Consider double- or triple-buffering objects if you need partial updates

Summary

- Reduce state changes, particularly redundant ones
- Avoid making CPU wait for GPU
- Instruments can help you identify both these cases

Workload size, bandwidth and computation



Vertex Processing Bottlenecks

Identification and classification

Tiler utilization at or near 100%

- Workload size: Number of vertices
 - Frame rate increases when using simplified models

- Cost per vertex
 - Data: Fetching of vertex data
 - Computation: Vertex transformation/ lighting/shading
 - Frame rate increases when doing less calculation

Vertex Fetching Bottlenecks Getting vertex data to the GPU quickly

- Use vertex buffer objects (VBOs)
 - Indicate usage pattern with storage hints
 - GL_STATIC_DRAW: update once, draw repeatedly
 - GL_DYNAMIC_DRAW: update repeatedly, draw repeatedly
 - GL_STREAM_DRAW: update once, draw at most a few times
- Use indexed draw calls where possible
 - Improves performance if vertices are reused

Vertex Fetching Bottlenecks

Getting vertex data to the GPU quickly

- Interleave your vertex attributes
 - Align vertex attributes and strides to 4-byte boundaries



V1	N1	TC1	V2	N2	TC2	V3	N3	ТСЗ		
									ľ	7
`	,									2
	Draw Array									

Vertex Processing Bottlenecks Summary

- Take advantage of VBOs and VAOs
- Size/structure your vertex data for efficient data transfer

Fragment Processing Bottlenecks

Identification and classification

Renderer utilization at or near 100%

• Workload size: Number of visible fragments

Overrides

🔘 None

- Eliminate all OpenGL ES operations
- O Eliminate rendering operations only
- O Simplify fragment shader processing
- O Simplify all shader processing
- O Minimize utilized texture bandwidth
- Minimize number of pixels rendered

• Cost per fragment

- Data: Framebuffer and texture bandwidth
- Computation: Fragment shading

Processing Fewer Pixels

Dealing with overdraw

- Hidden surface removal operates on groups of opaque objects
 - Maximum efficiency by drawing opaque objects together first
- Sort order:
 - Draw all opaque objects
 - Draw any objects using "discard" keyword
 - Draw all alpha blended objects

Processing Fewer Pixels Overdraw and blending/discard

- Blending affects every pixel in a quad
 - Even transparent ones
- Alpha-testing is generally expensive on embedded hardware
- Wasted pixels can add up as number of layers increase



Processing Fewer Pixels Sprite trimming

- Reduce area by using a shape that tightly encloses sprite
- Trades smaller shape for extra vertex processing



Processing Pixels Faster What's my limit?

- Bandwidth
 - Performance increases with smaller textures/lower bit depth
- Computation
 - Performance increases with simplified fragment shader
- Analyzer has overrides for both of these situations



- If you don't reuse your buffer contents from frame to frame:
 - Do a full-screen clear of all buffers at the start of the frame
 - Discard non-color buffers at the end of the frame

```
GLenum attachments[] = { GL_DEPTH_ATTACHMENT };
glDiscardFramebufferEXT(GL_READ_FRAMEBUFFER_APPLE, 1, attachments);
```

```
glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);
[context presentRenderbuffer:GL_RENDERBUFFER_0ES];
```



- If you don't reuse your buffer contents from frame to frame:
 - Do a full-screen clear of all buffers at the start of the frame
 - Discard non-color buffers at the end of the frame

```
GLenum attachments[] = { GL_DEPTH_ATTACHMENT };
glDiscardFramebufferEXT(GL_READ_FRAMEBUFFER_APPLE, 1, attachments);
```

```
glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);
[context presentRenderbuffer:GL_RENDERBUFFER_0ES];
```



• Even more important when performing multisample rendering



• Even more important when performing multisample rendering



- Use the smallest texture format and type suitable for your assets
 - PVRTC, if your content is suitable
 - Single-channel luminance, alpha
 - 16-bit formats: RGBA4444, RGBA5551, RGB565
- Size your textures appropriately for display
 - Use mipmaps if your textures will be drawn at many different scales

Fragment Processing Bottlenecks Summary

- Give the GPU the smallest number of pixels to deal with
 - Take advantage of GPU's ability to cull hidden surfaces
- Minimize amount of external bandwidth consumed by GPU

Shader Tuning for GLSL ES

Shader Tuning for GLSL ES Topics

- Precision qualifiers
- Minimizing computation
- Dependent texture reads

Care and Feeding of Shader Precisions Introduction to shader precisions

- Specific to GLSL ES, not in desktop GLSL
- Varying precisions can differ between vertex and fragment stages
- Appropriate precision choices are important for good performance

Care and Feeding of Shader Precisions

- Single-precision floating point
 - Use for vertex positions and texture coordinate calculations
 - Good for texture coordinates in general

uniform highp mat4 modelviewProjection; uniform highp mat3 mapTexMatrix; attribute highp vec3 position; varying highp vec2 mapTexCoord;

mapTexCoord = (mapTexMatrix * position).st; gl_Position = modelviewProjection * vec4(position, 1.0);

Care and Feeding of Shader Precisions mediump

- Half-precision floating point
 - Potentially higher throughput
- Good for texture coordinate varyings if:
 - Texture size $< 512 \times 512$
 - Minimal wrapping/perspective

```
mediump float ndotl = max(dot(normal, objectLightDirection), 0.0);
mediump float hdotn = sign(ndotl) * dot(normal, halfAngle);
mediump litColor = ambientColor + diffuseColor * ndotl;
litColor += specularColor * pow(hdotn, specularExponent);
```

Care and Feeding of Shader Precisions

- [-2, +2] range, 8-bit fractional precision
 - Good for color, normals, and color mixing factors
- Use 3- and 4- component vectors
- Don't swizzle components

uniform lowp sampler2D tex;

varying lowp vec3 litColor;

lowp vec3 modulatedColor = texture2D(tex, coord).rgb * litColor; gl_FragColor = vec4(modulatedColor, 1.0);

Care and Feeding of Shader Precisions Choosing and using precisions

- Compiler is free to promote calculations to a higher precision
 - Keep in mind minimum required precision/range of type
 - Check implementation-specific precision and range
- Minimize conversions between precisions in calculations
 - Especially for conversions to/from lowp

Efficient Computation

Hoisting computation



Efficient Computation Expressing operations

- Operate only on the elements that you need
 - Don't coerce expressions into being vectors
- When mixing scalars and vectors, keep scalars together

```
uniform vec3 attenFactor;
```

```
mediump float ndotl = max(dot(normal, objectLightDirection), 0.0);
mediump float attenuation = attenFactor.x + attenFactor.y * objectDistance
+ attenFactor.z * (objectDistance * objectDistance);
litColor = ambientColor + diffuseColor * (ndotl / attenuation);
```

Efficient Computation Expressing operations

- Operate only on the elements that you need
 - Don't coerce expressions into being vectors
- When mixing scalars and vectors, keep scalars together

```
uniform vec3 attenFactor;
```

mediump float ndotl = max(dot(normal, objectLightDirection), 0.0); mediump float attenuation = attenFactor.x + attenFactor.y * objectDistance + attenFactor.z * (objectDistance * objectDistance); litColor = ambientColor + diffuseColor * (ndotl / attenuation);
Efficient Computation Expressing operations

- Operate only on the elements that you need
 - Don't coerce expressions into being vectors
- When mixing scalars and vectors, keep scalars together

```
uniform vec3 attenFactor;
```

```
mediump float ndotl = max(dot(normal, objectLightDirection), 0.0);
mediump float attenuation = attenFactor.x + attenFactor.y * objectDistance
+ attenFactor.z * (objectDistance * objectDistance);
litColor = ambientColor + diffuseColor * (ndotl / attenuation);
```

Efficient Computation GLSL built-ins

- Convenience functions for common functionality
- Convenient for hardware and shader compiler as well

lowp vec3 clean = texture2D(cleanTexture, coord).rgb;

```
lowp vec3 dirty = texture2D(dirtyTexture, coord).rgb;
lowp float dirtiness = texture2D(dirtMap, coord).a;
```

// BAD
lowp vec3 mixed = (1.0 - dirtiness) * clean + dirtiness * dirty;
lowp vec3 mixed = clean + (dirty - clean) * dirtiness;

// BETTER
lowp vec3 mixed = mix(clean, dirty, dirtiness);



Dependent Texture Reads What are they?

- Dependent texture reads
 - Texture samples from coordinates modified in shader
- Non-dependent texture reads
 - Texture samples from coordinates passed directly from varyings

Dependent Texture Reads

What causes them?

• Explicit texture coordinate modification in fragment shader

uniform lowp sampler2D tex; varying highp vec2 coord; varying highp vec2 warpTime;

lowp vec4 nonHoistableSample = texture2D(tex, coord + sin(warpTime)); lowp vec4 hoistableSample = texture2D(tex, coord + vec2(0.5, -0.5));

Dependent Texture Reads What causes them?

• Projective or biased texture samples (on PowerVR SGX)

```
uniform lowp sampler2D tex;
varying highp vec3 coord;
// projection
lowp vec4 proj = texture2DProj(tex, coord);
// LOD bias
lowp vec4 bias = texture2D(tex, coord.st, bias);
// LOD selection
lowp vec4 lod = texture2DLod(tex, coord.st, lod);
```

Dependent Texture Reads Why does this matter?

- Non-dependent reads are typically faster
 - Fewer shader cycles
 - Better parallelism

Shader Tuning

Summary

- Choose your precisions carefully
- Make sure your calculations are:
 - Performed in the appropriate stage
 - Including texture coordinate calculations
 - Expressed efficiently

Summary Rules of thumb

- Target the slowest pipeline stage
 - Let the tools tell you where to tune
 - Don't be afraid to do more work in other stages
- Do less
 - Take advantage of GPU hidden-surface removal
 - Use smaller data types
 - Minimize computation

OpenGL ES Analyzer Instrument Part deux

OpenGL ES Analyzer Instrument



OpenGL ES Expert It knows a thing or two

- Comprehensive expert system
- Knows the hardware and the software
- Finds problems in your app
 - ...and provides recommendations on how to address each of them



OpenGL ES Expert Categories of problems

- Redundant state changes
- Invalid framebuffer configurations
- Invalid texture configurations
- Invalid OpenGL operations
- Sub-optimal vertex data formats, layouts and storage
- Sub-optimal operation order
- Hardware-specific performance events

Demo PictureViewer, analyzed

OpenGL ES Analyzer Instrument



OpenGL ES Analyzer Instrument

📹 Instruments File Edit 🕚	View Instrument	Window Help		_			Mon 10:50 AM	Q
· · · · · · · · · · · · · · · · · · ·				Instruments	2			
(I)	Target	Inspectio	n Range	00:00:/ Run 1 of 1	View Library	Q- Instrument De Search	tail h	
	00.00		ke ja se ska		and the second			
. 🕨 🚺 🔁 OpenGL ES Analy:	zer i # Redundant S	Unoptimized MSA	A Resolve					
		Suboptimal MSAA	huffer resolve oper	ration 🔿				
	# CL Calls		surrer resource open		a in a start and a second start and the start and the second start and the second start and the second start a			
	# Eatches			Done				
	and the second second			Done				
	📜 🖉 🖉 🖉	ndered						
OpenGL ES								
OpenGL ES	 I I	istics \$						
OpenGL ES	 ∂ ■ ■ ⇒ ■ Frame Stat Frame #A 	istics ♀ # Triangles Rendered	# Batches	# GL Calls	# Redundant State Changes			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides None	 Image: State of the state of t	istics ¢ # Triangles Rendered 12	# Batches 6	# GL Calls 365	# Redundant State Changes			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides Onone Ominiate all OpenGL ES operation		istics ¢ # Triangles Rendered 12 12	# Batches 6 6	# GL Calls 365 48	# Redundant State Changes			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides One Eliminate all OpenGL ES operation Eliminate all OpenGL ES operation Simulify fragment shades noncess	Frame Stat Frame ##	istics ◆ # Triangles Rendered 12 12 12 12	# Batches 6 6 6	# GL Calls 365 48 48 48	# Redundant State Changes 3 0 0			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides None Eliminate all OpenGL ES operation Eliminate all OpenGL ES operation Simplify fragment shader procession Simplify fragment shader procession		istics ◆ # Triangles Rendered 12 12 12 12 12	# Batches 6 6 6 6 6	# GL Calls 365 48 48 48 48	# Redundant State Changes 3 0 0 0 0		=	
OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides One Eliminate all OpenGL ES operation Eliminate and OpenGL ES operation Simplify fragment shader processing Simplify all shader processing		istics ¢ # Triangles Rendered 12 12 12 12 12 12 12	# Batches 6 6 6 6 6 6 6 6	# GL Calls 365 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides OpenGL ES operation OpenGL E	Frame Stat Frame Stat Frame #A O Sing A idth S red G	istics ¢ # Triangles Rendered 12 12 12 12 12 12 12 12	# Batches 6 6 6 6 6 6 6 6 6	# GL Calls 365 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides None Eliminate all OpenGL ES operation Eliminate all OpenGL ES operation Eliminate all OpenGL ES operation Simplify flagments shader processing Minimize utilized texture bandwi Minimize number of pixels rende Call Tree		istics # Triangles Rendered 12 12 12 12 12 12 12 12 12 12	# Batches 6 6 6 6 6 6 6 6 6 6	# CL Calls 365 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides Overrides Overrides Overrides Overrides Overrides Overrides Overrides Override Overri		istics # Triangles Rendered 12 12 12 12 12 12 12 12 12 12	# Batches 6 6 6 6 6 6 6 6 6 6 6	# GL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides OpenGL ES Analyzer Overrides OpenGL ES operation OpenGL ES operation		istics \$ # Triangles Rendered 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	# Batches 6 6 6 6 6 6 6 6 6 6 6 6 6 6	# GL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenCL ES OpenCL ES OpenCL ES OpenCL ES Analyzer Overrides None Eliminate all OpenCL ES operation Eliminate all OpenCL ES operation Simplify fragment shader process Simplify all shader processing Minimize utilized texture bandwi Minimize utilized texture bandwi Minimize turber of pixels rende Separate by Thread Invert Call Tree		istics # Triangles Rendered 12 12 12 12 12 12 12 12 12 12 12 12 12 1	# Batches 6 6 6 6 6 6 6 6 6 6 6 6 6 6	# GL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides Overrides Overrides Overrides Overrides Overrides Overrides Overrides Override Overri		istics ◆ # Triangles Rendered 12 12 12 12 12 12 12 12 12 12 12 12 12	# Batches 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	# CL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides OpenGL ES Analyzer Overrides OpenGL ES operation OpenG		istics # Triangles Rendered 12 12 12 12 12 12 12 12 12 12	# Batches 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	# GL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides Ov		istics # Triangles Rendered 12 12 12 12 12 12 12 1	# Batches 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	# GL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides Overrides OpenGL ES Analyzer Overrides OpenGL ES operato OpenGL ES openato OpenGL ES operato OpenGL ES openato OpendE es openato Openato Openato		istics \$ # Triangles Rendered 12 12 12 12 12 12 12 12 12 12 12 12 12	# Batches 6 6 6 6 6 6 6 6 6 6 6 6 6 8 8	# GL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
OpenGL ES OpenGL ES OpenGL ES OpenGL ES Analyzer Overrides OpenGL ES Analyzer Overrides OpenGL ES operation Onore Osimplify Fragment shader processing Minimize utilized texture bandwil Minized texture bandwil		istics	# Batches 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 8 8 8	# GL Calls 365 48 48 48 48 48 48 48 48 48 48 48 48 48	# Redundant State Changes 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			

OpenGL ES Analyzer Instrument A call to arms

- Go get the tool
- Improve your performance
- Send us feedback
- Make an awesome app!

				۵	
🗯 Instruments File Edit View	Instrument	Window Help	_		
				Instruments2	2
(I)	ver_1 :	Inspectio	n Range	Run 1 of 1	4 O View
OpenGL ES Analyzer ()	# Redundant S	Unoptimized MSA Suboptimal MSAA	A Resolve buffer resolve op	eration 🔘	
	# GL Calls # Batches (# Triangles Re	endered		Done	
OpenGLES	E Frame Sta	ristics 🜢			
Overrides	Frame #A	# Triangles Rendered	# Batches	# GL Calls	# Redundant State Cha
⊙ None	0	12	6	365	Pricedine and State City
O Eliminate all OpenGL ES operations	1	12	6	48	
C Eliminate rendering operations only	2	12	6	48	
 Simplify fragment shader processing 	3	12	6	48	
O Simplify all shader processing	4	12	6	48	
O Minimize utilized texture bandwidth	5	12	6	48	
Call Tree	7	12	6	48	
Separate by Category	8	12	6	48	
Separate by Category	9	12	6	48	
Invert Call Tree	10	12	6	48	
Hide Missing Symbols	11	12	6	48	
Hide System Libraries	12	12	6	48	
Show Obj-C Only	13	12	6	48	
Flatten Recursion	14	16	8	55	
Call Tree Constraints	15	16	8	55	
Specific Data Mining	16	16	8	55	
	17	16	8	\$5	

More Information

Allan Schaffer Graphics and Game Technologies Evangelist aschaffer@apple.com

Mike Jurewitz Developer Tools and Performance Evangelist jurewitz@apple.com

Documentation OpenGL ES Programming Guide for iPhone OS http://developer.apple.com/iphone

Khronos http://www.khronos.org

Apple Developer Forums http://devforums.apple.com

Related Sessions

Advanced Performance Analysis with Instruments	Mission Thursday 9:00AM
OpenGL Essential Design Practices	Pacific Heights Wednesday 11:30AM
OpenGL ES Overview for iPhone OS	Presidio Wednesday 2:00PM

Labs

OpenGL ES Lab

Graphics and Media Lab A Thursday 9:00AM to 4:15PM



