

OpenGL ES Tools and Techniques

Get faster, fast!

Session 514

Seth Sowerby

Manager, GPU Software Developer Technologies

These are confidential sessions—please refrain from streaming, blogging, or taking pictures

Introduction

- iOS is a great platform for games
- OpenGL ES is a great technology for powering games on iOS
- Apple provides OpenGL ES tools to help you make great games



What You Will Learn

- What OpenGL ES tools are available to you
- Why you need these tools
- What new features are introduced in Xcode 4.5 and iOS 6.0 SDK
- Workflows and methods to get the most from these tools

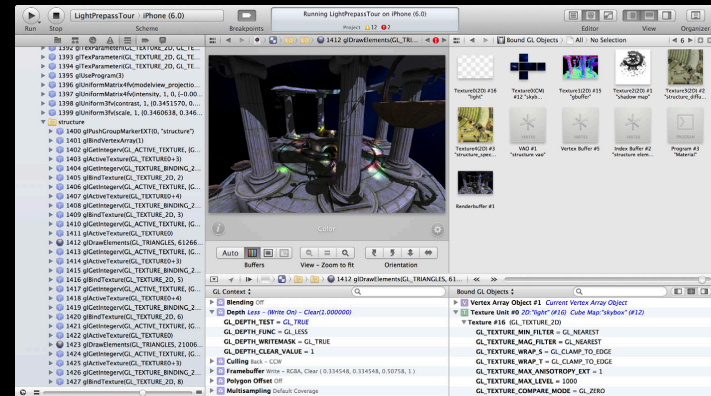
OpenGL ES Tools Recap

The story so far

OpenGL ES Frame Debugger

Integrated into Xcode

- OpenGL ES debugging built into Xcode
- Launch your app from Xcode
- Capture a frame
- Debug!



OpenGL ES Frame Debugger

The screenshot displays the OpenGL ES Frame Debugger interface, showing a 3D scene rendered on an iPhone (6.0). The interface is divided into several panels:

- Command List:** A list of OpenGL ES commands and their parameters, such as `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)` and `glDrawElements(GL_TRIANGLES, 61266, GL_UNSIGNED_INT, 0, 0)`.
- 3D Viewport:** A central window showing the rendered 3D scene, which appears to be a classical building interior with columns and a dome.
- Texture Browser:** A panel on the right showing a grid of textures, including `Texture0(2D) #16 "light"`, `Texture1(2D) #15 "gbuffer"`, and `Texture2(2D) #1 "shadow map"`.
- GL Context:** A panel at the bottom left showing the current state of the GL context, including `Blending Off`, `Depth Less - (Write On) - Clear(1.000000)`, and `Culling Back - CCW`.
- Bound GL Objects:** A panel at the bottom right showing the current state of bound GL objects, including `Vertex Array Object #1 Current Vertex Array Object` and `Texture Unit #0 2D: "light" (#16) Cube Map: "skybox" (#12)`.

OpenGL ES Frame Debugger

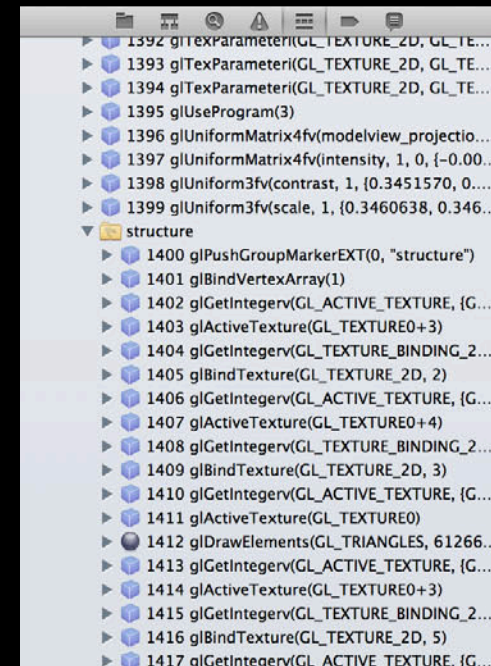
The screenshot displays the OpenGL ES Frame Debugger interface, which is used for analyzing and debugging OpenGL ES rendering. The interface is divided into several panels:

- Command List:** A tree view on the left showing a sequence of OpenGL ES commands, such as `glTexParameteri`, `glUniformMatrix4fv`, and `glDrawElements`. The current command being debugged is `1412 glDrawElements(GL_TRIANGLES, 61266, ...)`.
- 3D Viewport:** A central window showing a 3D scene with classical architecture, including columns and a dome. The scene is rendered with various textures and lighting effects.
- Color and Buffers:** A panel below the viewport showing the current color and buffer settings. The color is set to `Auto`, and the buffers are `GL_DEPTH_TEST`, `GL_DEPTH_FUNC`, and `GL_DEPTH_CLEAR_VALUE`.
- GL Context:** A panel at the bottom left showing the current GL context settings, including `Blending Off`, `Depth Less - (Write On) - Clear(1.000000)`, `GL_DEPTH_TEST = GL_TRUE`, `GL_DEPTH_FUNC = GL_LESS`, `GL_DEPTH_WRITEMASK = GL_TRUE`, `GL_DEPTH_CLEAR_VALUE = 1`, `Culling Back - CCW`, `Framebuffer Write - RGBA, Clear (0.334548, 0.334548, 0.50758, 1)`, `Polygon Offset Off`, and `Multisampling Default Coverage`.
- Bound GL Objects:** A panel at the bottom right showing the current bound GL objects, including `Vertex Array Object #1 Current Vertex Array Object` and `Texture Unit #0 2D: "light" (#16) Cube Map: "skybox" (#12)`. The texture unit settings are `GL_TEXTURE_MIN_FILTER = GL_NEAREST`, `GL_TEXTURE_MAG_FILTER = GL_NEAREST`, `GL_TEXTURE_WRAP_S = GL_CLAMP_TO_EDGE`, `GL_TEXTURE_WRAP_T = GL_CLAMP_TO_EDGE`, `GL_TEXTURE_MAX_ANISOTROPY_EXT = 1`, `GL_TEXTURE_MAX_LEVEL = 1000`, and `GL_TEXTURE_COMPARE_MODE = GL_ZERO`.

OpenGL ES Frame Debugger

Navigate your frame

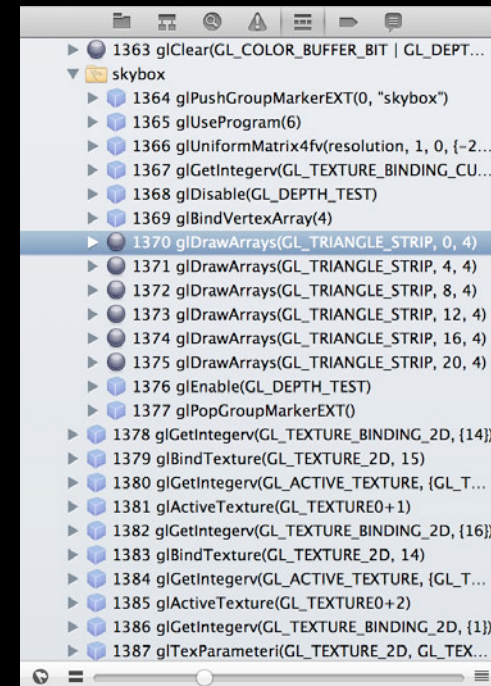
- All the GL calls from your frame displayed in the Debug Navigator
- Select a call to inspect GL at the point
- Annotate your frame with `glPushGroupMarkerEXT` / `glPopGroupMarkerEXT`
- Disclose a call to inspect the call stack



OpenGL ES Frame Debugger

Navigate your frame

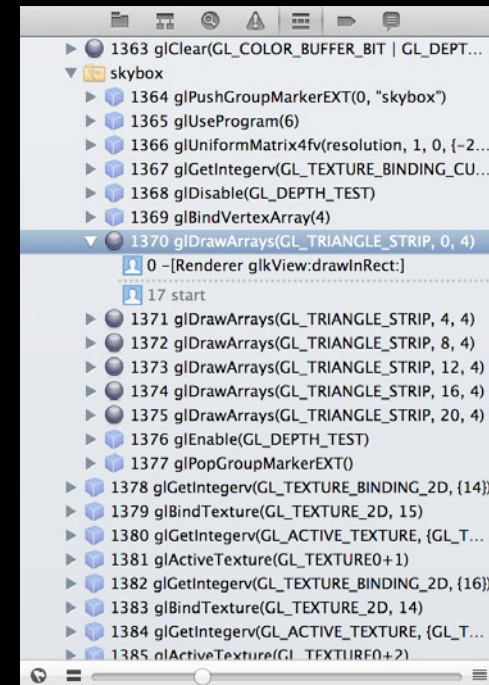
- All the GL calls from your frame displayed in the Debug Navigator
- Select a call to inspect GL at the point
- Annotate your frame with `glPushGroupMarkerEXT` / `glPopGroupMarkerEXT`
- Disclose a call to inspect the call stack



OpenGL ES Frame Debugger

Navigate your frame

- All the GL calls from your frame displayed in the Debug Navigator
- Select a call to inspect GL at the point
- Annotate your frame with `glPushGroupMarkerEXT` / `glPopGroupMarkerEXT`
- Disclose a call to inspect the call stack



OpenGL ES Frame Debugger

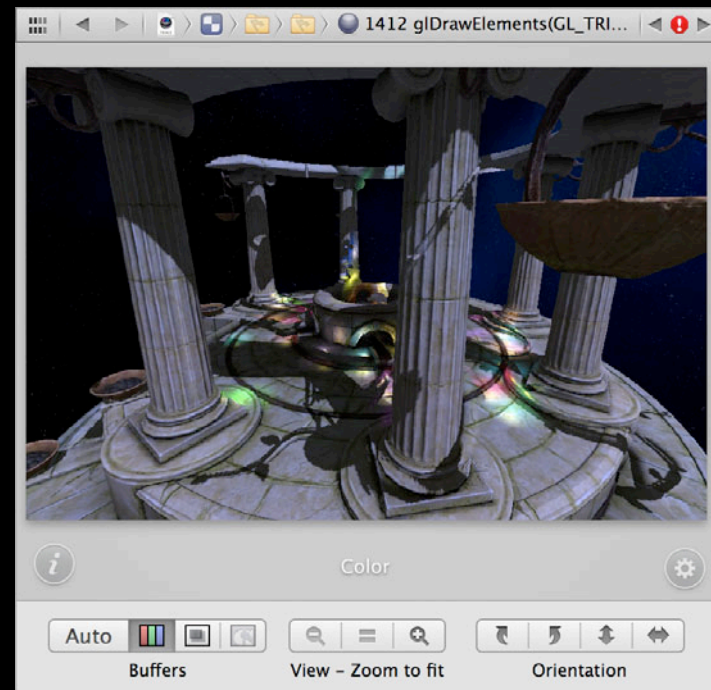
The screenshot displays the OpenGL ES Frame Debugger interface, which is used for analyzing and debugging graphics applications. The interface is divided into several main sections:

- Command List (Left):** A tree view showing the sequence of OpenGL ES commands executed during the current frame. The list includes commands such as `glTexParameteri`, `glUniformMatrix4fv`, `glBindVertexArray`, `glActiveTexture`, `glBindTexture`, `glDrawElements`, and `glBindTexture`. The current command being inspected is `1412 glDrawElements(GL_TRIANGLES, 61266, ...)`.
- 3D View (Center):** A 3D rendering of the scene, showing a classical architectural structure with columns and a dome. The scene is rendered with various textures and lighting effects.
- Texture Browser (Right):** A grid of texture thumbnails, including `Texture0(2D) #16 "light"`, `Texture0(CM) #12 "skyb..."`, `Texture1(2D) #15 "gbuffer"`, `Texture2(2D) #1 "shadow map"`, `Texture3(2D) #2 "structure_diffu..."`, `Texture4(2D) #3 "structure_spec..."`, `VAO #1 "structure vao"`, `Vertex Buffer #5`, `Index Buffer #2 "structure elem..."`, `Program #3 "Material"`, and `Renderbuffer #1`.
- GL Context (Bottom Left):** A panel showing the current state of the OpenGL ES context. It includes settings for `Blending Off`, `Depth Less - (Write On) - Clear(1.000000)`, `GL_DEPTH_TEST = GL_TRUE`, `GL_DEPTH_FUNC = GL_LESS`, `GL_DEPTH_WRITEMASK = GL_TRUE`, `GL_DEPTH_CLEAR_VALUE = 1`, `Culling Back - CCW`, `Framebuffer Write - RGBA, Clear (0.334548, 0.334548, 0.50758, 1)`, `Polygon Offset Off`, and `Multisampling Default Coverage`.
- Bound GL Objects (Bottom Right):** A panel showing the current state of the bound GL objects. It includes `Vertex Array Object #1 Current Vertex Array Object` and `Texture Unit #0 2D: "light" (#16) Cube Map: "skybox" (#12)`. The texture unit settings are: `Texture #16 (GL_TEXTURE_2D)`, `GL_TEXTURE_MIN_FILTER = GL_NEAREST`, `GL_TEXTURE_MAG_FILTER = GL_NEAREST`, `GL_TEXTURE_WRAP_S = GL_CLAMP_TO_EDGE`, `GL_TEXTURE_WRAP_T = GL_CLAMP_TO_EDGE`, `GL_TEXTURE_MAX_ANISOTROPY_EXT = 1`, `GL_TEXTURE_MAX_LEVEL = 1000`, and `GL_TEXTURE_COMPARE_MODE = GL_ZERO`.

OpenGL ES Frame Debugger

Inspect your framebuffer

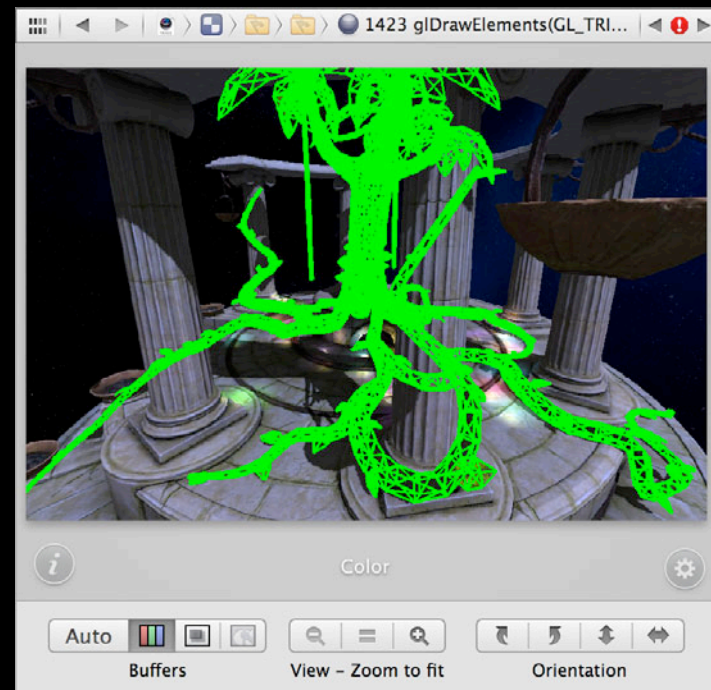
- Inspect the contents of the color, depth, and stencil buffers
- Current draw call highlighted as wireframe



OpenGL ES Frame Debugger

Inspect your framebuffer

- Inspect the contents of the color, depth, and stencil buffers
- Current draw call highlighted as wireframe



OpenGL ES Frame Debugger

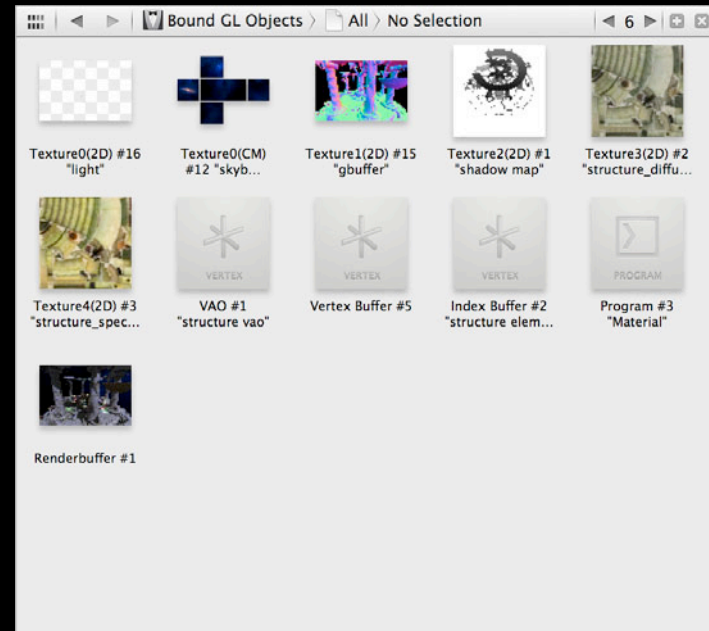
The screenshot displays the OpenGL ES Frame Debugger interface, which is used for analyzing and debugging OpenGL ES rendering. The interface is divided into several panels:

- Command List:** A list of OpenGL ES commands and their parameters, such as `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)` and `glDrawElements(GL_TRIANGLES, 61266, GL_UNSIGNED_INT, 0, 0)`.
- 3D Viewport:** A central window showing a 3D scene with a classical building structure, including columns and a dome, rendered in a dark environment.
- Texture Browser:** A panel on the right showing a grid of textures, including `Texture0(CM) #12 "skybox"`, `Texture1(2D) #15 "gbuffer"`, and `Texture2(2D) #1 "shadow map"`.
- GL Context:** A panel at the bottom left showing the current state of the GL context, including `Blending Off`, `Depth Less - (Write On) - Clear(1.000000)`, `GL_DEPTH_TEST = GL_TRUE`, `GL_DEPTH_FUNC = GL_LESS`, `GL_DEPTH_WRITEMASK = GL_TRUE`, `GL_DEPTH_CLEAR_VALUE = 1`, `Culling Back - CCW`, `Framebuffer Write - RGBA, Clear (0.334548, 0.334548, 0.50758, 1)`, `Polygon Offset Off`, and `Multisampling Default Coverage`.
- Bound GL Objects:** A panel at the bottom right showing the current state of bound GL objects, including `Vertex Array Object #1 Current Vertex Array Object` and `Texture Unit #0 2D:"light" (#16) Cube Map:"skybox" (#12)`.

OpenGL ES Frame Debugger

Inspect GL objects

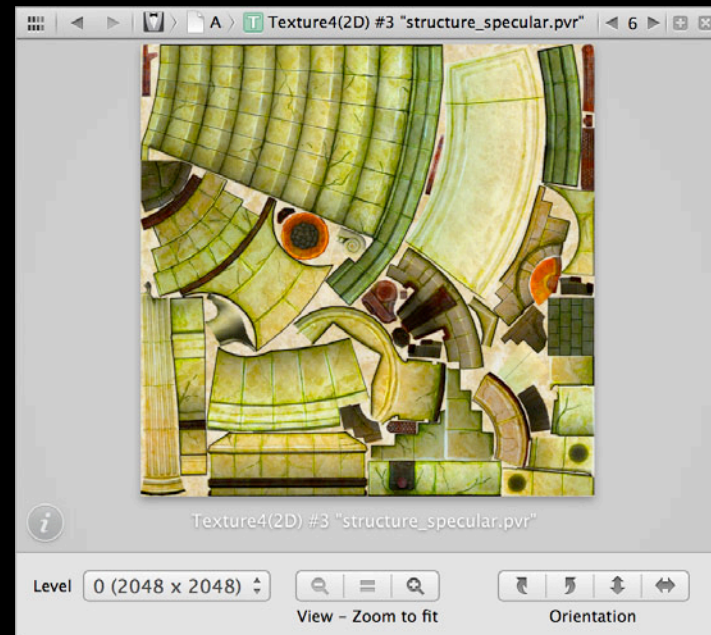
- Inspect all OpenGL ES object types
- Filter by bound objects or all
- Select an object to view in detail
- Label your resources with `glLabelObjectEXT`



OpenGL ES Frame Debugger

Inspect GL objects

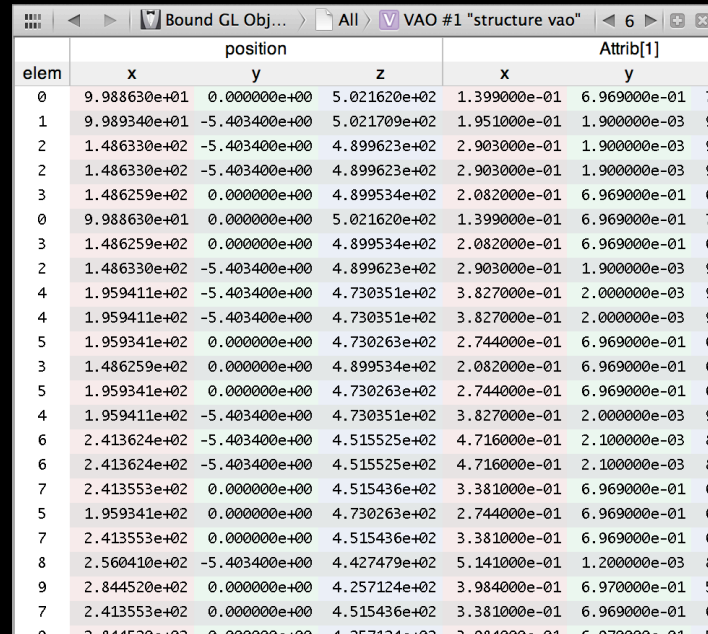
- Inspect all OpenGL ES object types
- Filter by bound objects or all
- Select an object to view in detail
- Label your resources with `glLabelObjectEXT`



OpenGL ES Frame Debugger

Inspect GL objects

- Inspect all OpenGL ES object types
- Filter by bound objects or all
- Select an object to view in detail
- Label your resources with `glLabelObjectEXT`



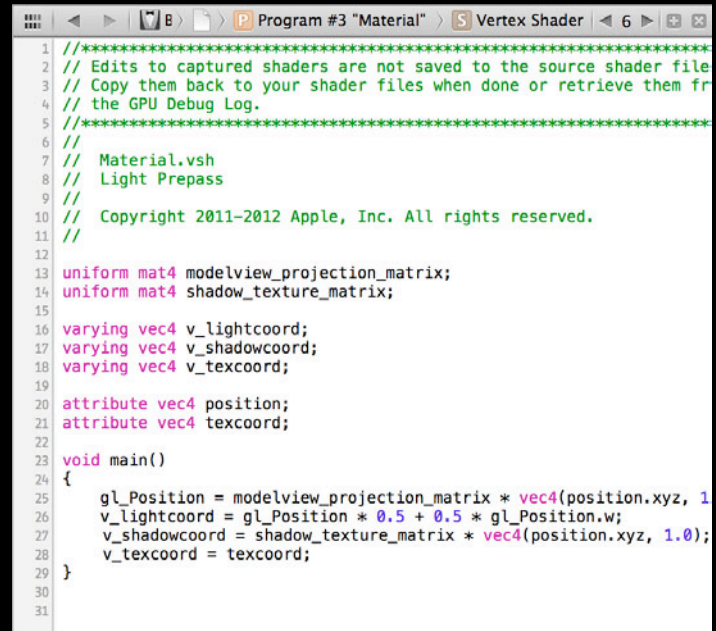
The screenshot shows a window titled "Bound GL Obj..." with a dropdown menu set to "VAO #1 'structure vao'". The table below displays the details of the bound objects.

elem	position			Attrib[1]	
	x	y	z	x	y
0	9.988630e+01	0.000000e+00	5.021620e+02	1.399000e-01	6.969000e-01
1	9.989340e+01	-5.403400e+00	5.021709e+02	1.951000e-01	1.900000e-03
2	1.486330e+02	-5.403400e+00	4.899623e+02	2.903000e-01	1.900000e-03
2	1.486330e+02	-5.403400e+00	4.899623e+02	2.903000e-01	1.900000e-03
3	1.486259e+02	0.000000e+00	4.899534e+02	2.082000e-01	6.969000e-01
0	9.988630e+01	0.000000e+00	5.021620e+02	1.399000e-01	6.969000e-01
3	1.486259e+02	0.000000e+00	4.899534e+02	2.082000e-01	6.969000e-01
2	1.486330e+02	-5.403400e+00	4.899623e+02	2.903000e-01	1.900000e-03
4	1.959411e+02	-5.403400e+00	4.730351e+02	3.827000e-01	2.000000e-03
4	1.959411e+02	-5.403400e+00	4.730351e+02	3.827000e-01	2.000000e-03
5	1.959341e+02	0.000000e+00	4.730263e+02	2.744000e-01	6.969000e-01
3	1.486259e+02	0.000000e+00	4.899534e+02	2.082000e-01	6.969000e-01
5	1.959341e+02	0.000000e+00	4.730263e+02	2.744000e-01	6.969000e-01
4	1.959411e+02	-5.403400e+00	4.730351e+02	3.827000e-01	2.000000e-03
6	2.413624e+02	-5.403400e+00	4.515525e+02	4.716000e-01	2.100000e-03
6	2.413624e+02	-5.403400e+00	4.515525e+02	4.716000e-01	2.100000e-03
7	2.413553e+02	0.000000e+00	4.515436e+02	3.381000e-01	6.969000e-01
5	1.959341e+02	0.000000e+00	4.730263e+02	2.744000e-01	6.969000e-01
7	2.413553e+02	0.000000e+00	4.515436e+02	3.381000e-01	6.969000e-01
8	2.560410e+02	-5.403400e+00	4.427479e+02	5.141000e-01	1.200000e-03
9	2.844520e+02	0.000000e+00	4.257124e+02	3.984000e-01	6.970000e-01
7	2.413553e+02	0.000000e+00	4.515436e+02	3.381000e-01	6.969000e-01

OpenGL ES Frame Debugger

Inspect GL objects

- Inspect all OpenGL ES object types
- Filter by bound objects or all
- Select an object to view in detail
- Label your resources with `glLabelObjectEXT`

A screenshot of a code editor window titled "Program #3 'Material'". The editor shows a vertex shader program with the following code:

```
1 //*****
2 // Edits to captured shaders are not saved to the source shader file
3 // Copy them back to your shader files when done or retrieve them fr
4 // the GPU Debug Log.
5 //*****
6 //
7 // Material.vsh
8 // Light Prepass
9 //
10 // Copyright 2011-2012 Apple, Inc. All rights reserved.
11 //
12
13 uniform mat4 modelview_projection_matrix;
14 uniform mat4 shadow_texture_matrix;
15
16 varying vec4 v_lightcoord;
17 varying vec4 v_shadowcoord;
18 varying vec4 v_texcoord;
19
20 attribute vec4 position;
21 attribute vec4 texcoord;
22
23 void main()
24 {
25     gl_Position = modelview_projection_matrix * vec4(position.xyz, 1
26     v_lightcoord = gl_Position * 0.5 + 0.5 * gl_Position.w;
27     v_shadowcoord = shadow_texture_matrix * vec4(position.xyz, 1.0);
28     v_texcoord = texcoord;
29 }
30
31
```

OpenGL ES Frame Debugger

The screenshot displays the OpenGL ES Frame Debugger interface, which is used for analyzing and debugging graphics applications. The interface is divided into several main sections:

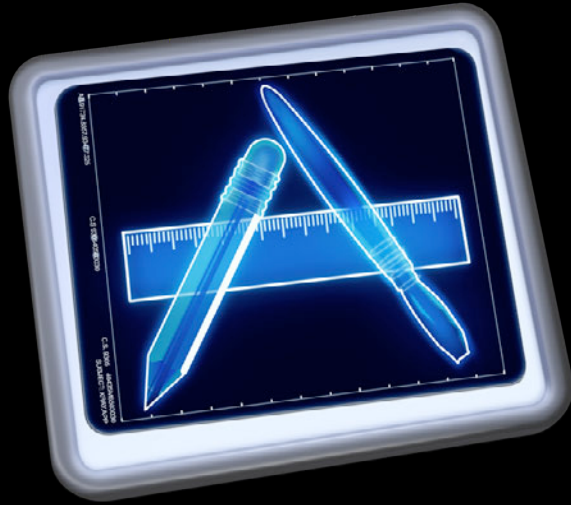
- Command List (Left):** A list of OpenGL ES commands and their parameters, such as `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)` and `glDrawElements(GL_TRIANGLES, 61266, GL_UNSIGNED_INT, 0, 0)`. The list is filtered to show the current frame's commands.
- 3D View (Center):** A 3D rendering of a scene with classical architecture, including columns and a dome. The scene is rendered with various textures and lighting effects.
- Color (Bottom Center):** A color selection tool with a color picker and a color swatch.
- Buffers (Bottom Center):** A panel for viewing and manipulating buffers, including a color picker and a zoom-to-fit button.
- Orientation (Bottom Center):** A panel for selecting the orientation of the 3D view, with options for Auto, View, and Zoom to fit.
- GL Context (Bottom Left):** A panel showing the current GL context settings, including `Blending Off`, `Depth Less - (Write On) - Clear(1.000000)`, `Culling Back - CCW`, `Framebuffer Write - RGBA, Clear (0.334548, 0.334548, 0.50758, 1)`, `Polygon Offset Off`, and `Multisampling Default Coverage`.
- Bound GL Objects (Bottom Right):** A panel showing the current bound GL objects, including `Vertex Array Object #1 Current Vertex Array Object` and `Texture Unit #0 2D: "light" (#16) Cube Map: "skybox" (#12)`. The texture unit settings are displayed, such as `GL_TEXTURE_MIN_FILTER = GL_NEAREST`, `GL_TEXTURE_MAG_FILTER = GL_NEAREST`, `GL_TEXTURE_WRAP_S = GL_CLAMP_TO_EDGE`, `GL_TEXTURE_WRAP_T = GL_CLAMP_TO_EDGE`, `GL_TEXTURE_MAX_ANISOTROPY_EXT = 1`, `GL_TEXTURE_MAX_LEVEL = 1000`, and `GL_TEXTURE_COMPARE_MODE = GL_ZERO`.

OpenGL ES Frame Debugger

Inspect GL state

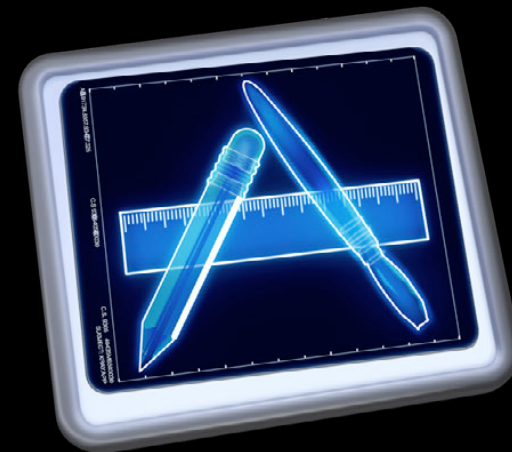
- Inspect all GL state
- Context state
- Object state
- Renderer capabilities
- Summarizes related states
- Highlights changes states



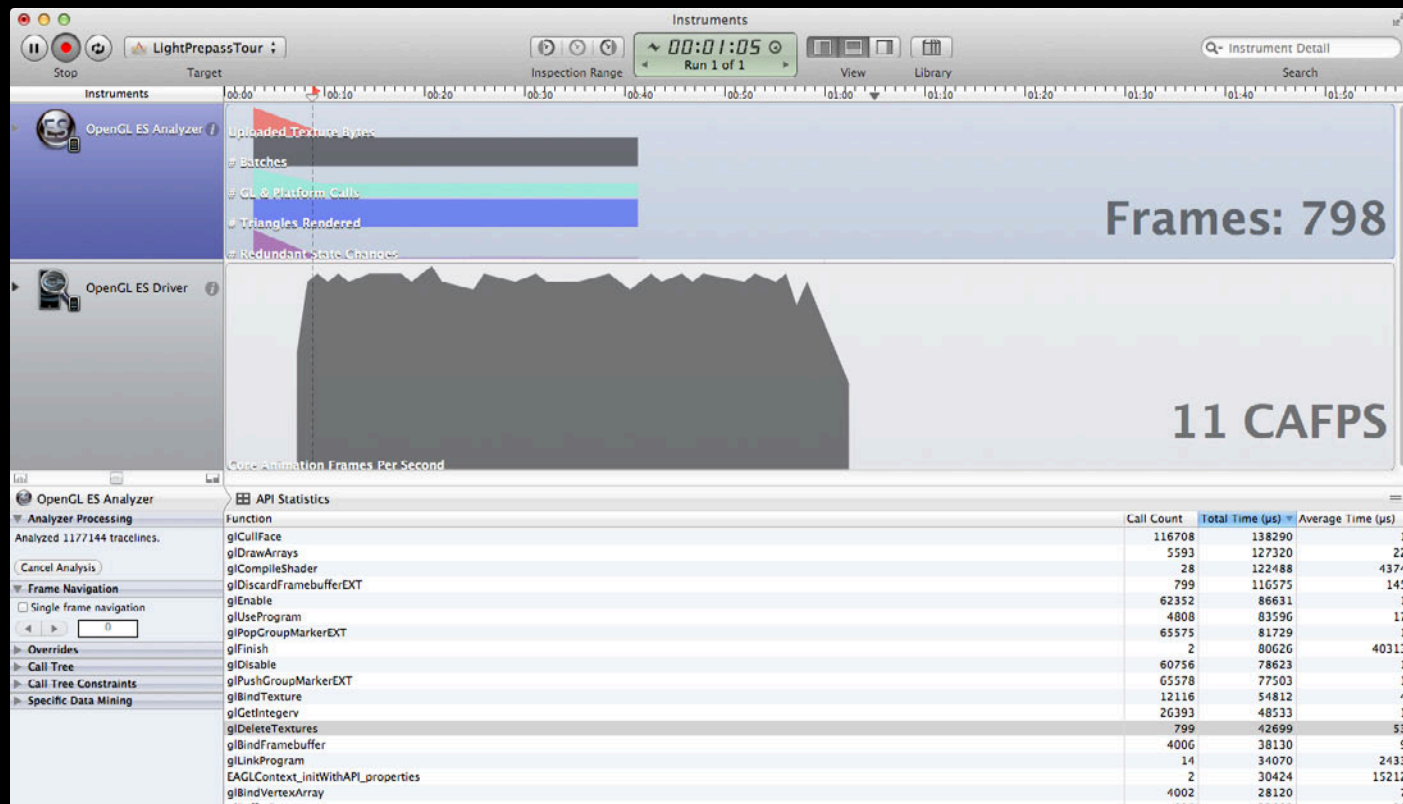


OpenGL ES Analyzer Instrument

- Part of Instruments
- Graph representation of GL performance data
- Use alongside CPU profiling etc.



OpenGL ES Analyzer Instrument



OpenGL ES Analyzer Instrument

OpenGL ES API Statistics

Function	Call Count	Total Time (μs) ▾	Average Time (μs)
glDrawElements	254583	6519618	25
glGenerateMipmap	1	2275145	2275145
EAGLContext_presentRenderBuffer	1663	1559831	937
glUniform3fv	242936	1079557	4
glUniformMatrix4fv	251253	914282	3
glCompressedTexImage2D	105	489800	4664
glClear	6655	405659	60
glDiscardFramebufferEXT	1663	346054	208
glStencilOp	239610	325154	1
glStencilFunc	239610	318970	1
glColorMask	239610	317806	1
glCullFace	242938	306760	1
glDrawArrays	11641	259941	22
glTexImage2D	1679	219948	130
glUseProgram	9992	184302	18
glEnable	129787	178038	1
glPopGroupMarkerEXT	136460	172754	1

OpenGL ES Analyzer Instrument

OpenGL ES Expert

Expert		Categories		
Severity	Total...	Unique...	Category	Summary
■	1862	1	Validation Error	Invalid OpenGL ES API usage detected
■	1862	1	GL Error: Invalid Operation	GL Error: Invalid Operation
▲	2	2	CPU wait on GPU for Finish	Finish caused CPU wait for GPU
▲	2	2	Unnecessary Logical Buffer Store	Unnecessary logical buffer store operation
▲	32002	19	Redundant Call	Redundant state call
▲	1862	1	Report Opaque After Transparent	Opaque geometry rendered after transparent geometry
▲	1861	1	Fragment Shader Dynamic Branching + Memory Access	Dynamic branching and main memory access in fragment shader
▲	1	1	GenerateMipmap Texture Synchronization	GenerateMipmap caused texture synchronization
▲	7444	2	Dependent Texture Sampling	Dependent texture samples in fragment shader
▲	1862	1	Inefficient State Update	Inefficient state update
▲	109	3	Mip-Linear Filtering Might Not Be Needed	Mip-Linear filtering might not be needed
▲	1863	2	Logical Buffer Load	Slow framebuffer load
▲	1867	4	Texture Format Compactness	8-bit per channel texture format
▲	1859	1	State Query Call Count	Numerous state query calls in current frame
▲	1	1	Recommend TexParameter Setup Before Upload	Mipmap usage changed after uploading texture data
▲	1862	2	Mipmapping Usage	Possible mipmapping usage scenario



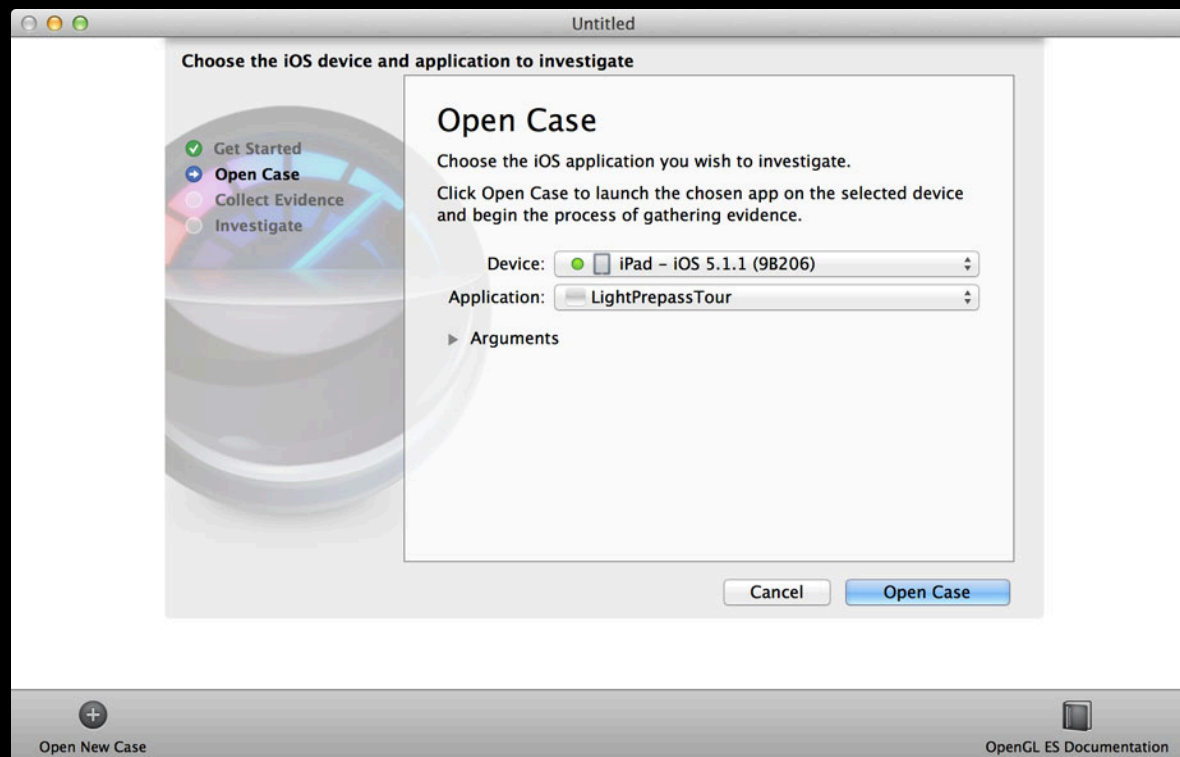
OpenGL ES Performance Detective

- Automated performance analysis of your scene
- Runs rich set of experiments to find OpenGL ES performance bottlenecks
- Gives actionable advice to address performance



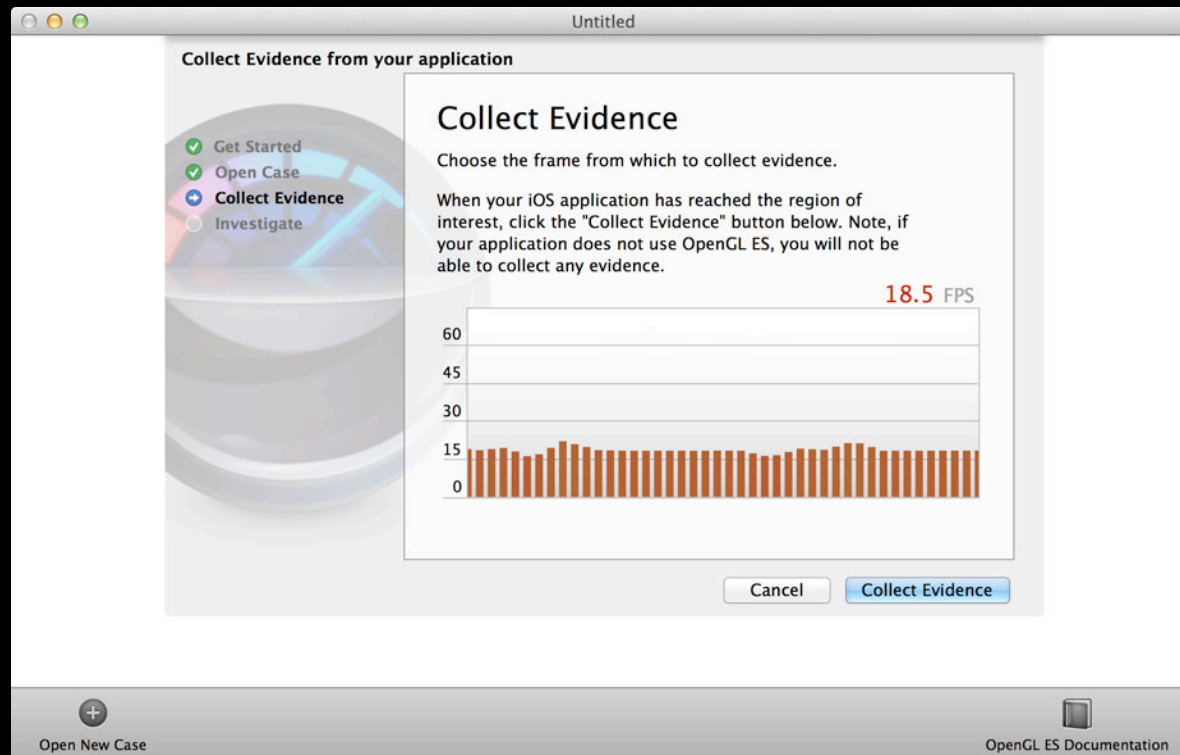
OpenGL ES Performance Detective

Select your app



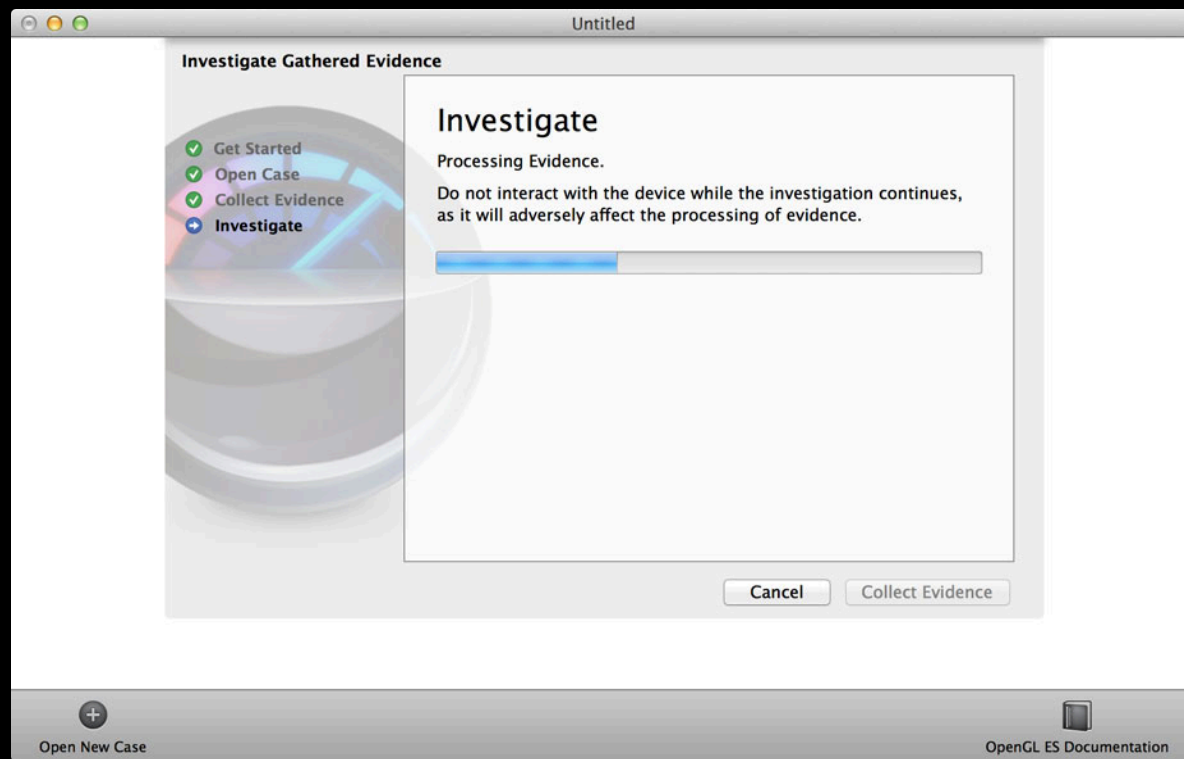
OpenGL ES Performance Detective

Trigger when to investigate



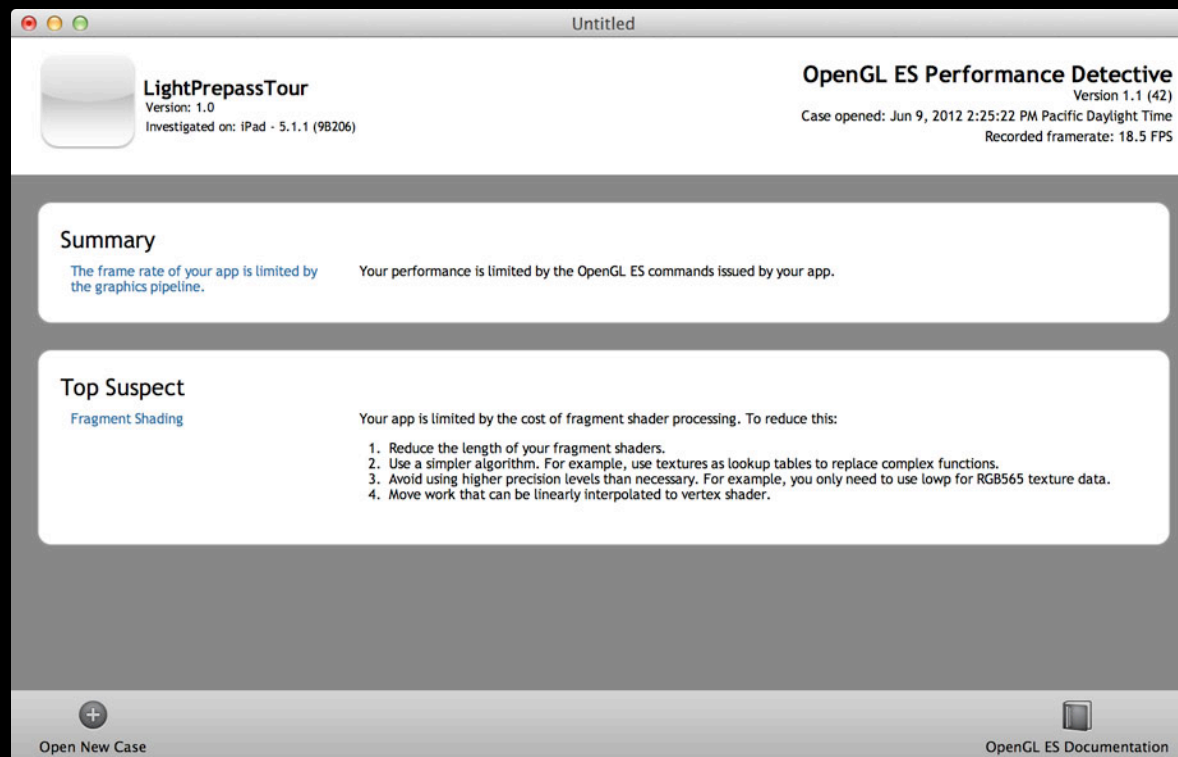
OpenGL ES Performance Detective

Let investigations ensue



OpenGL ES Performance Detective

Get actionable performance analysis



The screenshot shows the OpenGL ES Performance Detective application window. The window title is "Untitled". The application name is "LightPrepassTour" with version "1.0" and "Investigated on: iPad - 5.1.1 (9B206)". The application being analyzed is "OpenGL ES Performance Detective" with version "1.1 (42)". The case was opened on "Jun 9, 2012 2:25:22 PM Pacific Daylight Time" and the recorded framerate is "18.5 FPS".

Summary
The frame rate of your app is limited by the graphics pipeline. Your performance is limited by the OpenGL ES commands issued by your app.

Top Suspect
Fragment Shading
Your app is limited by the cost of fragment shader processing. To reduce this:

1. Reduce the length of your fragment shaders.
2. Use a simpler algorithm. For example, use textures as lookup tables to replace complex functions.
3. Avoid using higher precision levels than necessary. For example, you only need to use lowp for RGB565 texture data.
4. Move work that can be linearly interpolated to vertex shader.

At the bottom of the window, there is a button labeled "Open New Case" and a link labeled "OpenGL ES Documentation".

What's New in Xcode 4.5

Improvements for OpenGL ES

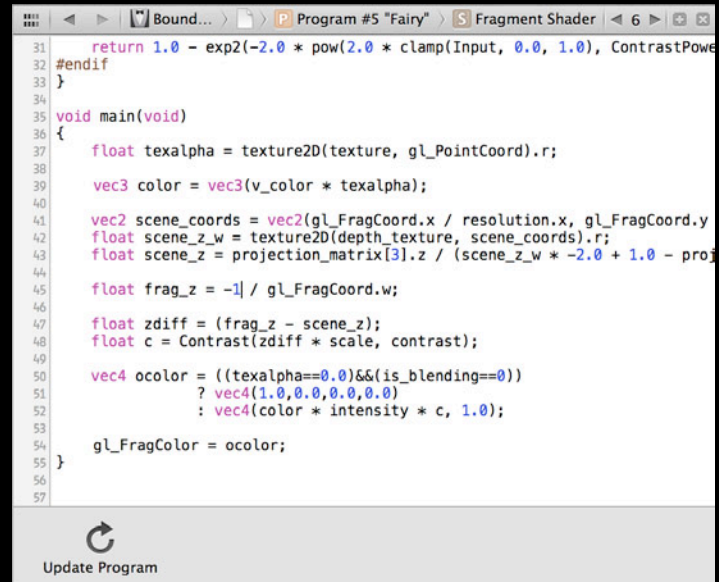
8 Great New Features

1. Shader edit and continue
2. Integrated OpenGL ES Expert
3. Multi-context debugging
4. Save and load frame captures
5. Performance monitoring
6. Integrated OpenGL ES Performance Analysis
7. OpenGL ES Performance Analysis 2.0
8. Find your slow shaders

1

Shader Edit and Continue

- Edit the GLSL for your programs
- See compilation and link errors inline
- See the effect of your changes instantly
- Edits saved to a shader edit log
- Resume the app to see your changes live

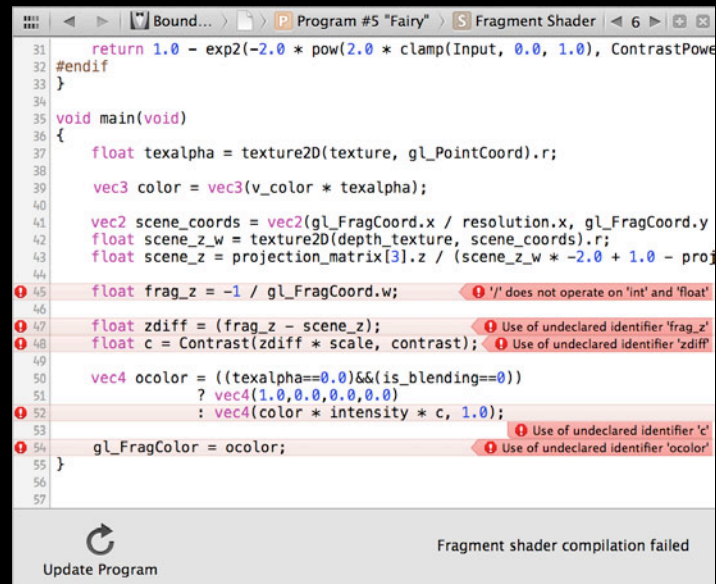


```
31     return 1.0 - exp2(-2.0 * pow(2.0 * clamp(Input, 0.0, 1.0), ContrastPower
32 #endif
33 }
34
35 void main(void)
36 {
37     float texalpha = texture2D(texture, gl_PointCoord).r;
38
39     vec3 color = vec3(v_color * texalpha);
40
41     vec2 scene_coords = vec2(gl_FragCoord.x / resolution.x, gl_FragCoord.y
42     float scene_z_w = texture2D(depth_texture, scene_coords).r;
43     float scene_z = projection_matrix[3].z / (scene_z_w * -2.0 + 1.0 - proj
44
45     float frag_z = -1 / gl_FragCoord.w;
46
47     float zdiff = (frag_z - scene_z);
48     float c = Contrast(zdiff * scale, contrast);
49
50     vec4 ocolor = ((texalpha==0.0)&&(is_blending==0))
51     ? vec4(1.0,0.0,0.0,0.0)
52     : vec4(color * intensity * c, 1.0);
53
54     gl_FragColor = ocolor;
55 }
56
57
```

Update Program

Shader Edit and Continue

- Edit the GLSL for your programs
- See compilation and link errors inline
- See the effect of your changes instantly
- Edits saved to a shader edit log
- Resume the app to see your changes live



```
31     return 1.0 - exp2(-2.0 * pow(2.0 * clamp(Input, 0.0, 1.0), ContrastPow
32 #endif
33 }
34
35 void main(void)
36 {
37     float texalpha = texture2D(texture, gl_PointCoord).r;
38
39     vec3 color = vec3(v_color * texalpha);
40
41     vec2 scene_coords = vec2(gl_FragCoord.x / resolution.x, gl_FragCoord.y
42     float scene_z_w = texture2D(depth_texture, scene_coords).r;
43     float scene_z = projection_matrix[3].z / (scene_z_w * -2.0 + 1.0 - pro
44
45     float frag_z = -1 / gl_FragCoord.w; // does not operate on 'int' and 'float'
46
47     float zdiff = (frag_z - scene_z); Use of undeclared identifier 'frag_z'
48     float c = Contrast(zdiff * scale, contrast); Use of undeclared identifier 'zdiff'
49
50     vec4 ocolor = ((texalpha==0.0)&&(is_blending==0))
51     ? vec4(1.0,0.0,0.0,0.0)
52     : vec4(color * intensity * c, 1.0); Use of undeclared identifier 'c'
53
54     gl_FragColor = ocolor; Use of undeclared identifier 'ocolor'
55 }
56
57
```

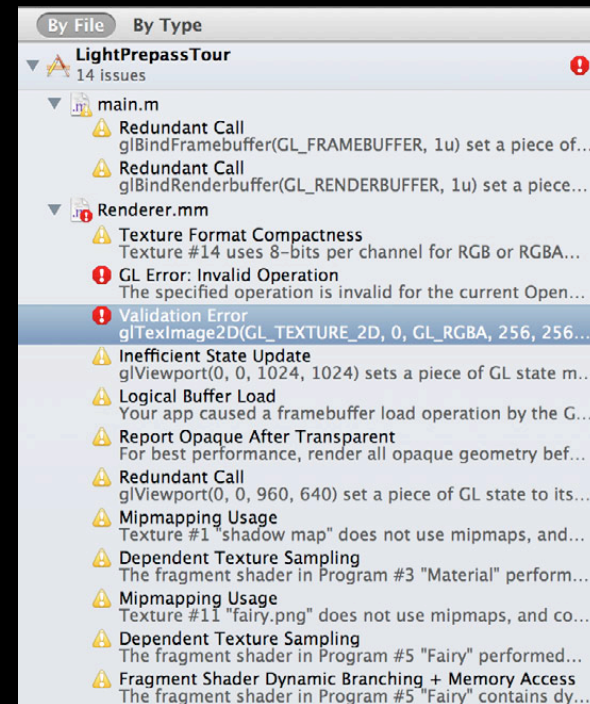
Fragment shader compilation failed

Update Program

2

Integrated OpenGL ES Expert

- GL issues for your captured frame in the frame debugger
- Catch bugs early and often
- No need to call glGetError all the time
- Go straight to your code to fix in place



By File By Type

LightPrepassTour
14 issues

- main.m
 - Redundant Call
glBindFramebuffer(GL_FRAMEBUFFER, 1u) set a piece of...
 - Redundant Call
glBindRenderbuffer(GL_RENDERBUFFER, 1u) set a piece...
- Renderer.mm
 - Texture Format Compactness
Texture #14 uses 8-bits per channel for RGB or RGBA...
 - GL Error: Invalid Operation
The specified operation is invalid for the current Open...
 - Validation Error
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256...
 - Inefficient State Update
glViewport(0, 0, 1024, 1024) sets a piece of GL state m...
 - Logical Buffer Load
Your app caused a framebuffer load operation by the G...
 - Report Opaque After Transparent
For best performance, render all opaque geometry bef...
 - Redundant Call
glViewport(0, 0, 960, 640) set a piece of GL state to its...
 - Mipmapping Usage
Texture #1 "shadow map" does not use mipmaps, and...
 - Dependent Texture Sampling
The fragment shader in Program #3 "Material" perform...
 - Mipmapping Usage
Texture #11 "fairy.png" does not use mipmaps, and co...
 - Dependent Texture Sampling
The fragment shader in Program #5 "Fairy" performed...
 - Fragment Shader Dynamic Branching + Memory Access
The fragment shader in Program #5 "Fairy" contains dy...

By File By Type

LightPrepassTour
14 issues

- main.m
 - Redundant Call
glBindFramebuffer(GL_FRAMEBUFFER, 1u) set a piece of...
 - Redundant Call
glBindRenderbuffer(GL_RENDERBUFFER, 1u) set a piece...
- Renderer.mm
 - Texture Format Compactness
Texture #14 uses 8-bits per channel for RGB or RGBA...
 - GL Error: Invalid Operation
The specified operation is invalid for the current Open...
 - Validation Error
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256...
 - Inefficient State Update
glViewport(0, 0, 1024, 1024) sets a piece of GL state m...
 - Logical Buffer Load
Your app caused a framebuffer load operation by the G...
 - Report Opaque After Transparent
For best performance, render all opaque geometry bef...
 - Redundant Call
glViewport(0, 0, 960, 640) set a piece of GL state to its...
 - Mipmapping Usage
Texture #1 "shadow map" does not use mipmaps, and...
 - Dependent Texture Sampling
The fragment shader in Program #3 "Material" perform...
 - Mipmapping Usage
Texture #11 "fairy.png" does not use mipmaps, and co...
 - Dependent Texture Sampling
The fragment shader in Program #5 "Fairy" performed...
 - Fragment Shader Dynamic Branching + Memory Access
The fragment shader in Program #5 "Fairy" contains dy...

Integrated OpenGL ES Expert

The screenshot displays an IDE interface with a project named "LightPrepassTour" containing 14 issues. The issues are categorized by file and type. The "Renderer.mm" file has several warnings, including "Texture Format Compactness", "GL Error: Invalid Operation", and "Validation Error". The "Validation Error" is highlighted, showing the code snippet: `glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);`. The code editor shows the corresponding code in "Renderer.mm" with line numbers 1450 to 1482. The code includes a multi-context guard, texture management, and rendering logic. A red tooltip is visible over the `glTexImage2D` call, indicating an invalid operation for the current OpenGL state.

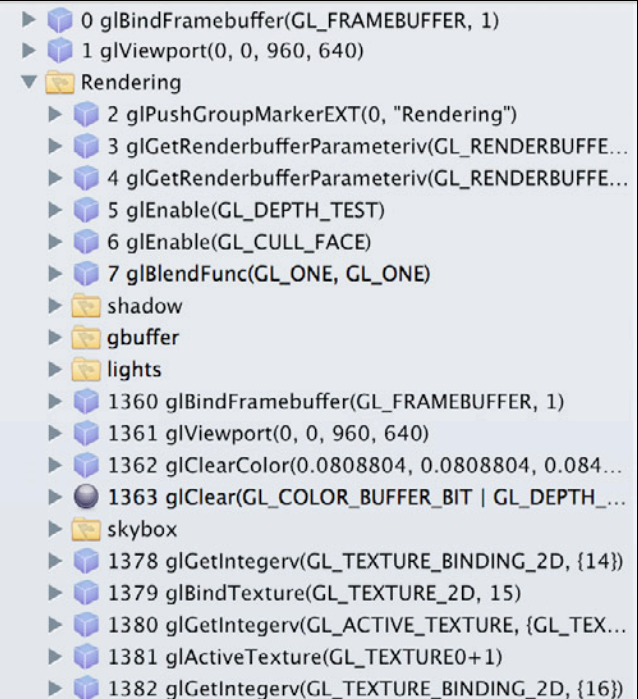
By File	By Type	Line
LightPrepassTour	14 issues	1450
main.m	Redundant Call	1451
main.m	Redundant Call	1452
Renderer.mm	Texture Format Compactness	1453
Renderer.mm	GL Error: Invalid Operation	1454
Renderer.mm	Validation Error	1455
Renderer.mm	Inefficient State Update	1456
Renderer.mm	Logical Buffer Load	1457
Renderer.mm	Report Opaque After Transparent	1458
Renderer.mm	Redundant Call	1459
Renderer.mm	Mipmapping Usage	1460
Renderer.mm	Dependent Texture Sampling	1461
Renderer.mm	Mipmapping Usage	1462
Renderer.mm	Dependent Texture Sampling	1463
Renderer.mm	Fragment Shader Dynamic Branching + Memory Access	1464

```
1450         farPlane];
1451         loaded = YES;
1452     }
1453 #ifdef MULTI_CONTEXT
1454     [EAGLContext setCurrentContext:backgroundContext];
1455
1456     if(lastTexture != 0)
1457     {
1458         glDeleteTextures(1, &lastTexture);
1459     }
1460
1461     glGenTextures(1, &lastTexture);
1462
1463     glActiveTexture (GL_TEXTURE0);
1464     glBindTexture(GL_TEXTURE_2D, lastTexture);
1465
1466     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGB,
1467                 GL_UNSIGNED_BYTE, NULL);
1468     [glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, nullptr)...
1469     #endif
1470
1471     [self updateLightsForTime:frameTime];
1472     [lpView resizeRTTBuffers];
1473
1474     glEnable(GL_DEPTH_TEST);
1475     glEnable(GL_CULL_FACE);
1476     glBlendFunc(GL_ONE, GL_ONE);
1477
1478     if (shadowMode != kShadowModeNone)
1479     {
1480         [self lpView:lpView renderShadowBufferForTime:frameTime];
1481         if (debugView == kDebugViewShadow)
1482         {
```

3

Multi-Context Debugging

- GL context switches shown within the Debug Navigator
- Quickly diagnose multi-context and threading issues
- Meets the needs of the upcoming wave of multi-context apps
[EAGLContext setDebugLabel:]

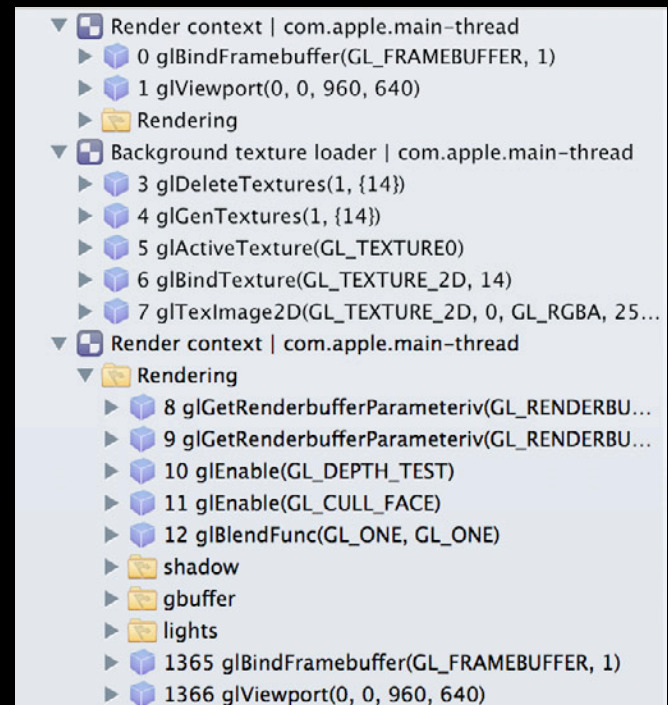


A screenshot of a debug navigator showing a list of OpenGL calls and context switches. The list is organized into a tree structure with expandable/collapsible icons. The calls include:

- 0 glBindFramebuffer(GL_FRAMEBUFFER, 1)
- 1 glViewport(0, 0, 960, 640)
- Rendering (expanded folder)
 - 2 glPushGroupMarkerEXT(0, "Rendering")
 - 3 glGetRenderbufferParameteriv(GL_RENDERBUFFER...
 - 4 glGetRenderbufferParameteriv(GL_RENDERBUFFER...
 - 5 glEnable(GL_DEPTH_TEST)
 - 6 glEnable(GL_CULL_FACE)
 - 7 glBlendFunc(GL_ONE, GL_ONE)
 - shadow (expanded folder)
 - gbuffer (expanded folder)
 - lights (expanded folder)
- 1360 glBindFramebuffer(GL_FRAMEBUFFER, 1)
- 1361 glViewport(0, 0, 960, 640)
- 1362 glClearColor(0.0808804, 0.0808804, 0.084...
- 1363 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_...
- skybox (expanded folder)
 - 1378 glGetIntegerv(GL_TEXTURE_BINDING_2D, {14})
 - 1379 glBindTexture(GL_TEXTURE_2D, 15)
 - 1380 glGetIntegerv(GL_ACTIVE_TEXTURE, {GL_TEX...
 - 1381 glActiveTexture(GL_TEXTURE0+1)
 - 1382 glGetIntegerv(GL_TEXTURE_BINDING_2D, {16})

Multi-Context Debugging

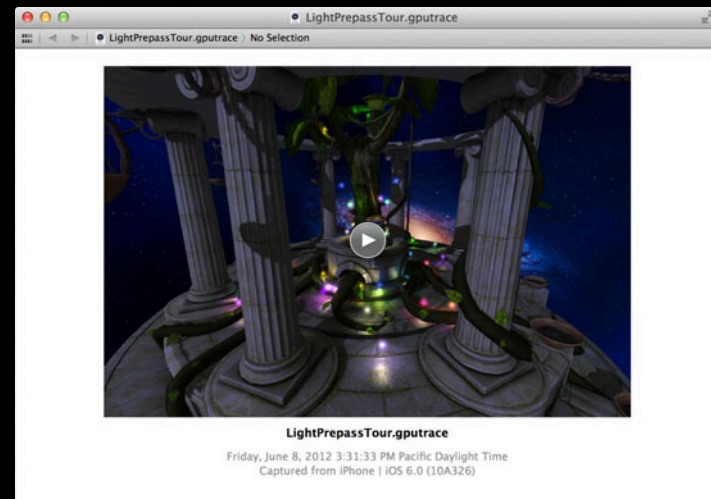
- GL context switches shown within the Debug Navigator
- Quickly diagnose multi-context and threading issues
- Meets the needs of the upcoming wave of multi-context apps
[EAGLContext setDebugLabel:]



4

Save and Load Captured Frames

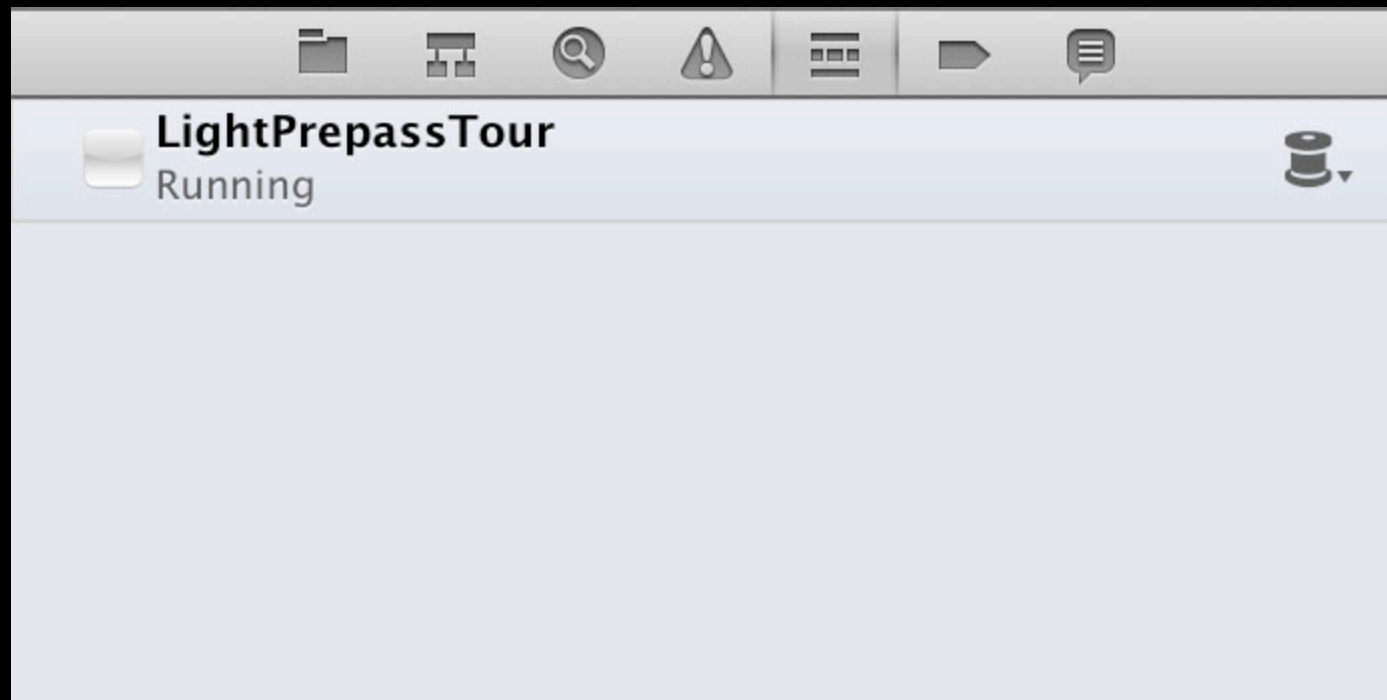
- Export the current frame while in the Frame Debugger
- Reload later to continue your work
- Or to test on another device
- Or send to someone else for their input



5

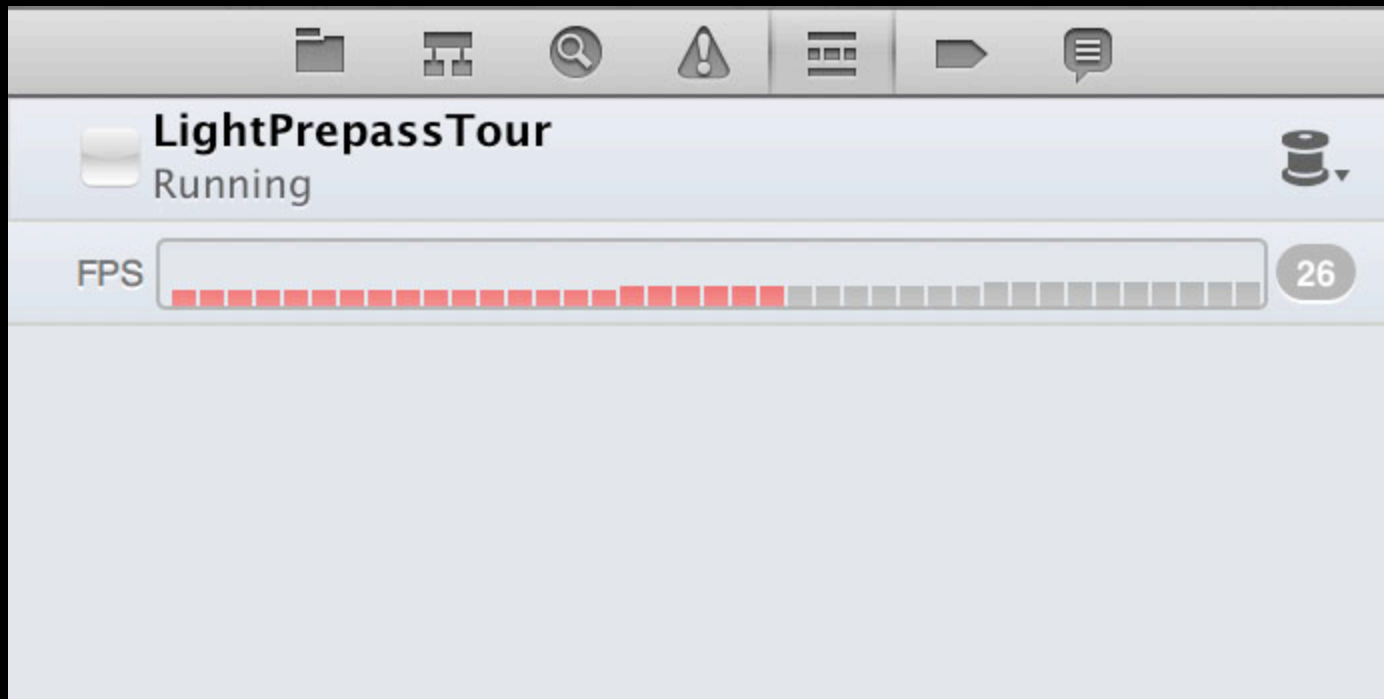
Performance Monitoring

Debug Navigator FPS

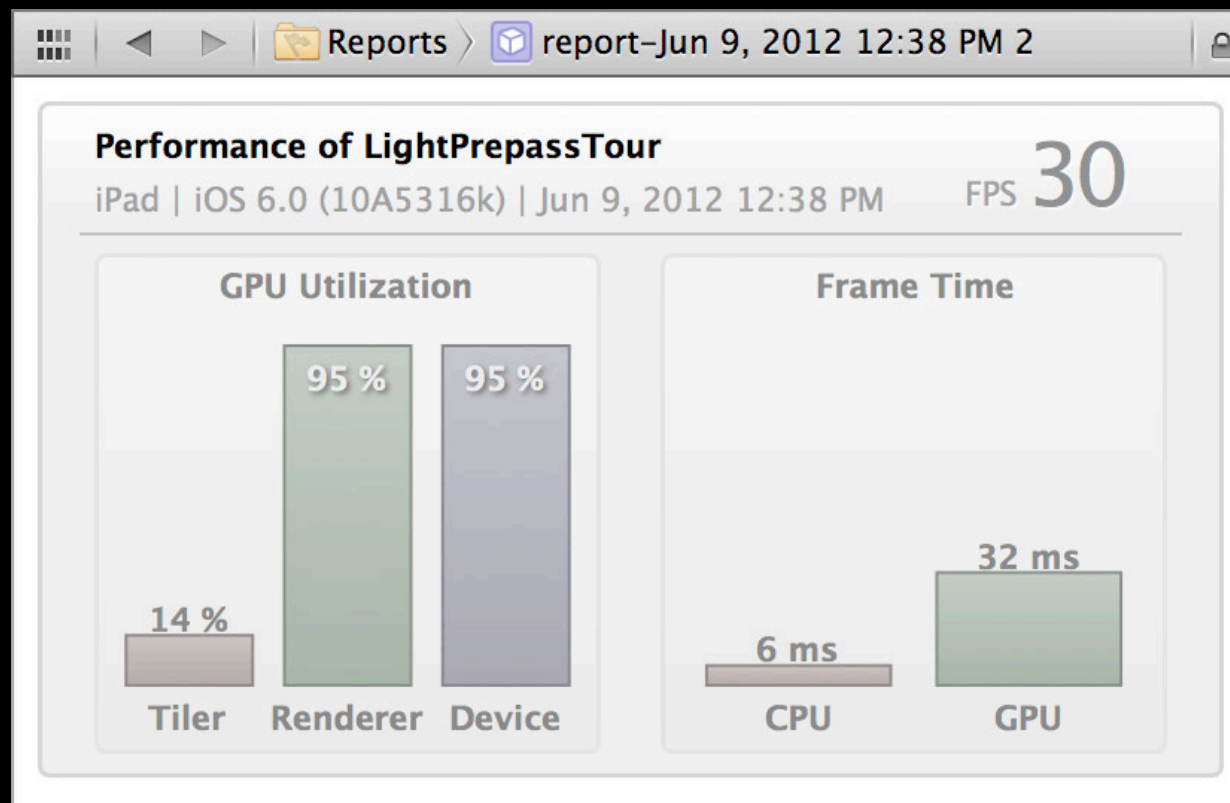


Performance Monitoring

Debug Navigator FPS



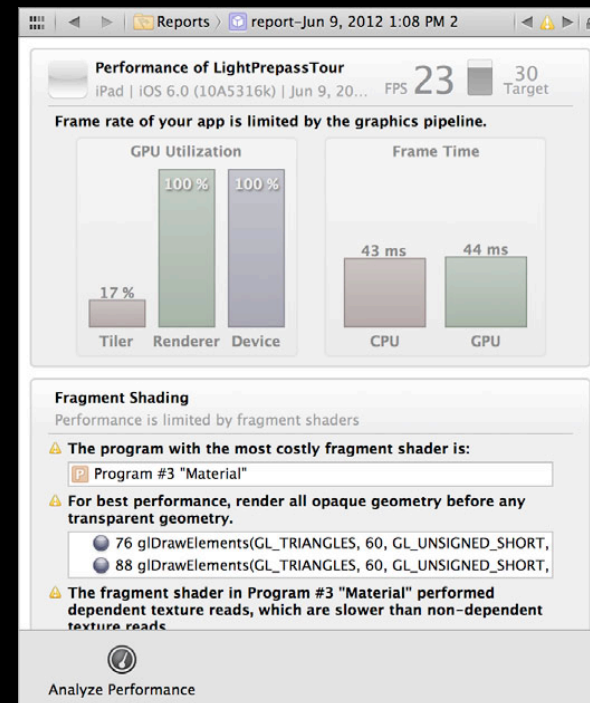
Performance Monitoring



6

Integrated OpenGL ES Performance Detective

- Performance analysis in Xcode for a cohesive workflow
- Works with the OpenGL ES Frame Debugger to let you drill down



7

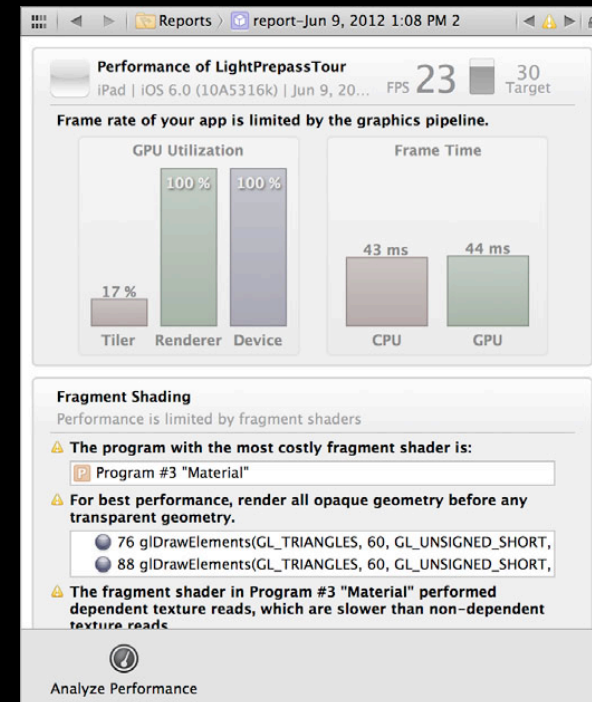
OpenGL ES Performance Analysis 2.0

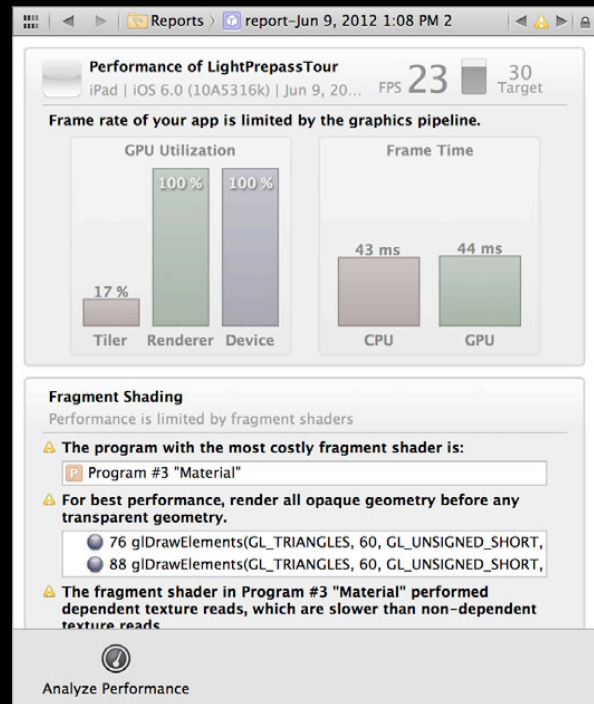
- Now detects CPU bottlenecks in OpenGL ES
 - Texture upload time
 - Vertex upload time
 - State validation time
- Detects more GPU bottlenecks
- Detects your target frame rate to guide its recommendations
- Integrated with the OpenGL Expert to relate its analysis to your GL commands

8

Finds Your Slow Shaders

- Even more analysis for fragment shader bound apps
- Finds the most costly shader
- Finds shaders with potential performance issues
- Takes you direct to the GLSL source code
- Edit your shader there and then





Reports > report-Jun 9, 2012 1:08 PM 2

Performance of LightPrepassTour

iPad | iOS 6.0 (10A5316k) | Jun 9, 20... FPS **23** Target 30

Frame rate of your app is limited by the graphics pipeline.

GPU Utilization

Tiler	17 %
Renderer	100 %
Device	100 %

Frame Time

CPU	43 ms
GPU	44 ms

Fragment Shading

Performance is limited by fragment shaders

- ⚠ The program with the most costly fragment shader is:
Program #3 "Material"
- ⚠ For best performance, render all opaque geometry before any transparent geometry.
 - 76 glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_SHORT,
 - 88 glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_SHORT,
- ⚠ The fragment shader in Program #3 "Material" performed dependent texture reads, which are slower than non-dependent texture reads.

Analyze Performance

Finds Your Slow Shaders

The screenshot shows the Xcode Performance tool interface. The top window displays the performance of the 'LightPrepassTour' app on an iPad (iOS 6.0, 10A5316k) on June 9, 2012, at 1:08 PM. The current FPS is 23, with a target of 30. A message states: 'Frame rate of your app is limited by the graphics pipeline.' Below this, two bar charts are shown: 'GPU Utilization' and 'Frame Time'. The GPU Utilization chart shows 17% for the Tiler, 100% for the Renderer, and 100% for the Device. The Frame Time chart shows 43 ms for the CPU and 44 ms for the GPU. The 'Fragment Shading' section indicates that performance is limited by fragment shaders. It identifies 'Program #3 "Material"' as the most costly fragment shader. A warning suggests rendering all opaque geometry before any transparent geometry. Two specific shader calls are listed: '76 glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_SHORT, ...)' and '88 glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_SHORT, ...)'. A final warning notes that the fragment shader in Program #3 performed dependent texture reads, which are slower than non-dependent texture reads. The bottom of the window shows an 'Analyze Performance' button.

Performance of LightPrepassTour
iPad | iOS 6.0 (10A5316k) | Jun 9, 20... FPS 23 Target 30

Frame rate of your app is limited by the graphics pipeline.

GPU Utilization

Component	Utilization
Tiler	17%
Renderer	100%
Device	100%

Frame Time

Component	Time
CPU	43 ms
GPU	44 ms

Fragment Shading
Performance is limited by fragment shaders

⚠ The program with the most costly fragment shader is:
Program #3 "Material"

⚠ For best performance, render all opaque geometry before any transparent geometry.

- 76 glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_SHORT, ...)
- 88 glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_SHORT, ...)

⚠ The fragment shader in Program #3 "Material" performed dependent texture reads, which are slower than non-dependent texture reads.

Analyze Performance

```
1 //*****  
2 // Edits to captured shaders are not saved to the source shader file.  
3 // Copy them back to your shader files when done or retrieve them from  
4 // the GPU Debug Log.  
5 //*****  
6 //  
7 // Material.fsh  
8 // Light Prepass  
9 //  
10 // Copyright 2011-2012 Apple, Inc. All rights reserved.  
11 //  
12 //  
13 //  
14 precision highp float;  
15 //  
16 #define USE_HARDWARE_PCF 1  
17 //  
18 #if USE_HARDWARE_PCF  
19 #extension GL_EXT_shadow_samplers : require  
20 uniform sampler2DShadow shadow_texture;  
21 #else  
22 uniform sampler2D shadow_texture;  
23 #endif  
24 //  
25 uniform sampler2D light_texture;  
26 uniform sampler2D gbuffer_texture;  
27 uniform sampler2D diffuse_texture;  
28 uniform sampler2D specular_texture;  
29 uniform vec3 sun_direction;  
30 uniform vec3 sun_color;  
31 //  
32 varying highp vec4 v_lightcoord;  
33 varying highp vec4 v_shadowcoord;  
34 varying highp vec4 v_texcoord;  
35 //  
36 float GetShadowValue(highp vec4 shadowCoord)  
37 {  
38 #if USE_HARDWARE_PCF  
39 return shadow2DProjEXT(shadow_texture, v_shadowcoord);  
40 }  
41 }
```

Demo

Seth Sowerby
GPU Software Developer Technologies

Demo

Michael Mayers

GPU Software Developer Technologies

Workflow Recommendations

Workflow Recommendations

- The Issue Navigator will find GL issues for you
 - Fix them early like you would compiler errors and warnings
 - Remember that GL errors can leave GL in an undefined state
- Annotate your frame with `glPushGroupMarkerEXT` / `glPopGroupMarkerEXT`
- Label your resources with `glLabelObjectEXT`

Workflow Recommendations

- Shader edit and continue let's you tweak your shaders in place
 - Great for making performance vs. quality trade-offs
- Use integrated performance analysis to find bottlenecks
 - Don't just assume you know why you're slow
 - Think about performance early—don't leave it till it's too late

More Information

Allan Schaffer

Graphics and Game Technologies Evangelist
aschaffer@apple.com

Michael Jurewitz

Developer Tools and Performance Evangelist
jury@apple.com

Apple Developer Forums

<http://devforums.apple.com>

Labs

OpenGL ES Lab

Graphics, Media & Games Lab A
Thursday 9:00AM

Summary

- Great OpenGL ES developers tools
- Even better in Xcode 4.5
- Harness them to make great games

 WWDC2012

The last 3 slides
after the logo are
intentionally left
blank for all
presentations.

The last 3 slides
after the logo are
intentionally left
blank for all
presentations.

The last 3 slides
after the logo are
intentionally left
blank for all
presentations.