Metal Performance Optimization Techniques

Session 610

Philip Bennett  GPU Software Performance
Serhat Tekin  GPU Software Developer Technologies
Metal at WWDC

What’s New in Metal, Part 1
- Metal in Review
- New Features
- Metal and App Thinning

What’s New in Metal, Part 2
- Introducing MetalKit
- Metal Performance Shaders

Metal Performance Optimization Techniques
- Metal System Trace Tool
- Metal Best Practices
Metal at WWDC

What’s New in Metal, Part 1
- Metal in Review
- New Features
- Metal and App Thinning

What’s New in Metal, Part 2
- Introducing MetalKit
- Metal Performance Shaders

Metal Performance Optimization Techniques
- Metal System Trace Tool
- Metal Best Practices
Metal at WWDC

What’s New in Metal, Part 1
- Metal in Review
- New Features
- Metal and App Thinning

What’s New in Metal, Part 2
- Introducing MetalKit
- Metal Performance Shaders

Metal Performance Optimization Techniques
- Metal System Trace Tool
- Metal Best Practices
Metal Debugging and Profiling Tools
Metal Tools in Xcode

- Visual frame debugger
- Resource and state viewer
- Shader profiling
- Shader edit-and-continue
- Debug mode for Metal framework
- Integrated offline Metal compiler
FPS
60 FPS

0 0x10032dc60 <- [MTLayer nextDrawable]
1 0x100340320 <- [MTDrawable texture]
2 0x100344680 <- [0x10032/0a0 commandBuffer]
CommandBuffer 0x100344680
  shadow buffer 8.49 µs
  g-buffer
    17 g-buffer <- [renderCommandEncoderWithDescriptor:<data>]
    g-buffer pass
      18 [pushDebugGroup:"g-buffer pass"]
      19 [setLabel:"g-buffer"]
      20 [setDepthStencilState:0x1007448d0]
    skybox 1.83 ms
  structure
    34 [pushDebugGroup:"structure"]
    35 [setRenderPipelineState:GBuffer Render]
    36 [setCullMode:2]
    37 [setDepthStencilState:0x100744ef0]
    38 [setStencilReferenceValue:128]
    39 [setVertexBuffer:0x10074ad00 offset:0 atindex:0]
    40 [setVertexBuffer:0x10074cb70 offset:0 atindex:1]
    41 [setFragmentBuffer:0x10074c150 offset:0 atindex:1]
    42 [setFragmentTexture:0x1003409d0 atindex:0]
    43 [setFragmentTexture:0x10033d690 atindex:1]
    44 [setFragmentTexture:0x100754540 atindex:2]
    45 [setFragmentTexture:0x100333290 atindex:3]
    46 [drawIndexedPrimitives:3 indexCount:61266 indexType:... 5.01 ms
    47 [setFragmentTexture:0x100344aa0 atindex:0]
    48 [setFragmentTexture:0x100755c70 atindex:1]
    49 [setFragmentTexture:0x100757360 atindex:1]
    50 [setFragmentTexture:0x100333290 atindex:3]
    51 [drawIndexedPrimitives:3 indexCount:21006 indexType... 1.11 ms
    52 [setFragmentTexture:0x10034bb90 atindex:0]
    53 [setFragmentTexture:0x100347170 atindex:1]
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>1.107226</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000020</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000040</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000060</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>-1.793836</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000080</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000100</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>-1.793836</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000120</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000140</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>-1.030359</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000160</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000180</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>-1.030359</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000001a0</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000001c0</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>0.493481</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000001e0</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000200</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>-0.617417</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000220</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000240</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000260</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000280</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000002a0</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000002c0</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000002e0</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000300</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>-0.416737</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000320</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000340</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000360</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000380</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>-0.844773</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000003a0</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000003c0</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000003e0</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000400</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>-0.486163</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000420</td>
<td>0.386180</td>
<td>0.082199</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000440</td>
<td>-0.317592</td>
<td>0.099551</td>
<td>0.373220</td>
<td>0.373220</td>
<td>0.420851</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000460</td>
<td>0.671940</td>
<td>0.107124</td>
<td>0.308022</td>
<td>0.306770</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000480</td>
<td>-0.551585</td>
<td>-0.130258</td>
<td>0.374520</td>
<td>0.373220</td>
<td>0.242005</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8 columns ▼ Float ▲
GPU Report

Frames Per Second: 60

Utilization:
- TILER: 67%
- RENDERER: 61%
- DEVICE: 67%

Frame Time:
- CPU: 16.3 ms
- GPU: 11.1 ms

Pipeline Performance

Program | Frame % | Current ms
--------|---------|-----------
Render Pipeline "GBuffer Render"
  46 [drawIndexedPrimitives: 3 indexCount: 61266 indexT... | 55.2% | 6.12 |
  51 [drawIndexedPrimitives: 3 indexCount: 21006 indexTy... | 45.2% | 5.01 |
  56 [drawIndexedPrimitives: 3 indexCount: 48 indexType:... | 10.0% | 1.11 |
Render Pipeline "Skybox Render" | 16.6% | 1.83 |
Render Pipeline "Composition Render" | 15.3% | 1.69 |
Render Pipeline "Light Color Render" | 12.8% | 1.42 |
Render Pipeline "Shadow Render" | 0.1% | 0.01 |
Total | | 11.07 |
output.v_view = (matrices->mvMatrix * tempPosition).xyz;

return output;

}

fragment FragOutput lightFrag(VertexOutput in [[stage_in]],
constant LightFragmentInputs *lightData [[buffer(0)]],
FragOutput gBuffers)
{

float3 n_s = gBuffers.normal.rgb;

float scene_z = gBuffers.depth;

float3 n = n_s * 2.0 - 1.0;

// Derive the view-space position of the scene fragment in the G-buffer
// Since the light primitive and the G-buffer were rendered with the same view-projection matrix,
// we can treat the view-space position of the current light primitive fragment as a ray from the origin,
// and derive the view-space position of the scene by projecting along the ray with
// (scene_z / v.view.z).
// Our scene view-space position is also the view-vector to the scene fragment.
float3 v = in.v_view * (scene_z / in.v_view.z);

// Now, we have everything we need to calculate our view-space lighting vectors.
float3 l = (lightData->view_light_position.xyz - v);

float n_ls = dot(n, l);

float v_ls = dot(v, l);

float l_ls = dot(l, l);

float3 h = (l * rsqrt(l_ls / v_ls)) - v);

float h_ls = dot(h, l);

float nLt = dot(n, l) * rsqrt(n_ls * l_ls);

float nh = dot(n, h) * rsqrt(n_hs * h_ls);

float d_atten = sqrt(l_ls);

float atten = fmax(1.0 - d_atten / lightData->light_color_radius.w, 0.0);

float diffuse = fmax(nLt, 0.0) * atten;

float4 light = gBuffers.light;

light.rgb = lightData->light_color_radius.xyz * diffuse;

light.a = pow(fmax(nh, 0.0), 32.0) * step(0.0, nLt) * atten * 1.0001;

FragOutput output;
output.albedo = gBuffers.albedo;
output.normal = gBuffers.normal;
output.depth = gBuffers.depth;
output.light = light;

return output;
Metal Tools in Instruments

Metal System Trace
Metal Tools in Instruments
Metal System Trace

New instrument for iOS 9
Profile Metal apps across CPU and GPU
null
Demo

Metal System Trace

Serhat Tekin
GPU Software Developer Technologies
Metal Performance Best Practices
Stalled: 22 ms
Command Buffer 0

Active: 25 ms

Stalled: 22 ms
Starved: 25 ms
Stalled: 22 ms
Active: 25 ms
Create Expensive Objects Upfront and Reuse
Shaders and State

In a legacy app

- Content loading
- Rendering loop
Shaders and State

In a legacy app

- Shader compilation
- Content loading
- Rendering loop
Shaders and State

In a legacy app

- Shader compilation
- Shader “warming”

Content loading

Rendering loop
Shaders and State

In a legacy app

<table>
<thead>
<tr>
<th>Content loading</th>
<th>Rendering loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shader compilation</td>
<td>State definition</td>
</tr>
<tr>
<td>Shader “warming”</td>
<td></td>
</tr>
</tbody>
</table>
Shaders and State

In a legacy app

Content loading

- Shader compilation
- Shader “warming”

Rendering loop

- State definition
- Draw calls
Shaders and State

In a Metal app

Content loading

- Shader compilation
- Shader “warming”

Rendering loop

- State definition
- Draw calls
Shaders and State

In a Metal app

Build time

Content loading

Shader compilation

Shader “warming”

State definition

Draw calls

Rendering loop
Shaders and State

In a Metal app

- Shader compilation
- Shader “warming”
- State definition
- Draw calls

Build time  Content loading  Rendering loop
Shaders and State

In a Metal app

- Shader compilation
- State definition
  - Shader “warming”
- Draw calls

Build time
Content loading
Rendering loop
Shaders and State
In a Metal app

- Shader compilation
- State definition
- Draw calls

Build time
Content loading
Rendering loop
Shaders and State

In a Metal app

- Shader compilation
- State definition
- Draw calls

Build time
Content loading
Rendering loop
Shaders and State
In a Metal app

- Shader compilation
- State definition
- Draw calls
- Draw calls
- Draw calls
- ...
- Build time
- Content loading
- Rendering loop
Metal State Management

Fundamentals
Metal State Management
Fundamentals

Metal facilitates upfront state definition

- Decouples state validation and compilation from draw commands
- Allows expensive operations to be moved out of rendering loop
Metal State Management

Fundamentals

Metal facilitates upfront state definition

• Decouples state validation and compilation from draw commands
• Allows expensive operations to be moved out of rendering loop

Expensive to create state encapsulated in immutable state objects

• Intended to be created upfront and reused many times
lightVert <- [0x1007c1af0 newFunctionWithNamed: "lightVert"]
compositionVertex <- [0x1007c1af0 newFunctionWithNamed: "compositionVertex"]
fairyVertex <- [0x1007c1af0 newFunctionWithNamed: "fairyVertex"]
fairyFragmentAlt <- [0x1007c1af0 newFunctionWithNamed: "fairyFragmentAlt"]
lightFragAlt <- [0x1007c1af0 newFunctionWithNamed: "lightFragAlt"]
compositionFragAlt <- [0x1007c1af0 newFunctionWithNamed: "compositionFragAlt"]
UIVert <- [0x1007c1af0 newFunctionWithNamed: "UIVert"]
UIFrag <- [0x1007c1af0 newFunctionWithNamed: "UIFrag"]
Light Mask Render <- [Device newRenderPipelineStateWithDescriptor: <data> error:0]
Light Color Render <- [Device newRenderPipelineStateWithDescriptor: <data> error:0]
Composition Render <- [Device newRenderPipelineStateWithDescriptor: <data> error:0]
Fairy Sprites <- [Device newRenderPipelineStateWithDescriptor: <data> error:0]
UI <- [Device newRenderPipelineStateWithDescriptor: <data> error:0]
0x100358aa0 <- [MTLLayer nextDrawable]
0x100346d50 <- [MTLDrawable texture]
0x10121b700 <- [0x1007c0cf0 commandBuffer]
8 lightVert <- [0x1007c1af0 newFunctionWithName:"lightVert"]
9 compositionVertex <- [0x1007c1af0 newFunctionWithName:"compositionVertex"]
10 fairyVertex <- [0x1007c1af0 newFunctionWithName:"fairyVertex"]
11 fairyFragmentAlt <- [0x1007c1af0 newFunctionWithName:"fairyFragmentAlt"]
12 lightFragAlt <- [0x1007c1af0 newFunctionWithName:"lightFragAlt"]
13 compositionFragAlt <- [0x1007c1af0 newFunctionWithName:"compositionFragAlt"]
14 UIVert <- [0x1007c1af0 newFunctionWithName:"UIVert"]
15 UIFrag <- [0x1007c1af0 newFunctionWithName:"UIFrag"]
16 Light Mask Render <- [Device newRenderPipelineStateWithDescriptor:s<
17 Light Color Render <- [Device newRenderPipelineStateWithDescriptor:s<
18 Composition Render <- [Device newRenderPipelineStateWithDescriptor:s<
19 Fairy Sprites <- [Device newRenderPipelineStateWithDescriptor:s<
20 UI <- [Device newRenderPipelineStateWithDescriptor:s<
21 0x100358aa0 <- [MTLLayer nextDrawable]
22 0x100346d50 <- [MTLDrawable texture]
23 0x10121b700 <- [0x1007c0cf0 commandBuffer]
Shader Library
MTLLibrary
Shader Library

MTLLibrary

A library of compiled shaders
Shader Library

MTLLibrary

A library of compiled shaders

Create offline using Xcode

• Automatically generates default library
• Accessed with [MTLDevice newDefaultLibrary]
Shader Library

MTLLibrary

A library of compiled shaders

Create offline using Xcode

- Automatically generates default library
- Accessed with `[MTLDevice newDefaultLibrary]`

Use command-line tools with shader creation pipeline

- Developer/Platforms/iPhoneOS.platform/usr/bin/metal, metal-ar, metallib
- Accessed with `[MTLDevice newLibraryWithFile:error:]`
Shader Library

MTLLibrary

A library of compiled shaders

Create offline using Xcode

• Automatically generates default library
• Accessed with `[MTLDevice newDefaultLibrary]`

Use command-line tools with shader creation pipeline

• Developer/Platforms/iPhoneOS.platform/usr/bin/metal, metal-ar, metallib
• Accessed with `[MTLDevice newLibraryWithFile:error:]`

Create asynchronously if runtime creation unavoidable

• `[MTLDevice newLibraryWithSource:options:completionHandler:]`
Device and Command Queue

MTLDevice and MTLCommandQueue
Device and Command Queue

MTLDevice and MTLCommandQueue

Represent a single GPU and queue of command buffers awaiting execution
Device and Command Queue

MTLDevice and MTLCommandQueue

Represent a single GPU and queue of command buffers awaiting execution

Create during app initialization and reuse throughout app lifetime
Device and Command Queue

MTLDevice and MTLCommandQueue

Represent a single GPU and queue of command buffers awaiting execution

Create during app initialization and reuse throughout app lifetime
Create one per GPU used
Render and Compute Pipeline State

MTLRenderPipelineState and MTLComputePipelineState
Render and Compute Pipeline State

MTLRenderPipelineState and MTLComputePipelineState

Encapsulate programmable GPU pipeline state
Render and Compute Pipeline State

MTLRenderPipelineState and MTLComputePipelineState

Encapsulate programmable GPU pipeline state

Create during content loading and reuse often
Render and Compute Pipeline State

MTLRenderPipelineState and MTLComputePipelineState

Encapsulate programmable GPU pipeline state

Create during content loading and reuse often

Can be created asynchronously

- [MTLDevice newRenderPipelineStateWithDescriptor:options:completionHandler:]
- MTLDevice newComputePipelineStateWithDescriptor:options:completionHandler:
Render and Compute Pipeline State

MTLRenderPipelineState and MTLComputePipelineState

Encapsulate programmable GPU pipeline state

Create during content loading and reuse often

Can be created asynchronously

• `[MTLDevice newRenderPipelineStateWithDescriptor:options:completionHandler:]`
• `[MTLDevice newComputePipelineStateWithDescriptor:options:completionHandler:]`

Don't obtain reflection data if not required
Depth/Stencil and Sampler State

MTLDepthStencilState and MTLSamplerState
Depth/Stencil and Sampler State

MTLDepthStencilState and MTLSamplerState

Encapsulate fixed-function GPU pipeline state
Depth/Stencil and Sampler State

MTLDepthStencilState and MTLSamplerState

Encapsulate fixed-function GPU pipeline state

Create during content loading along with pipeline state
Depth/Stencil and Sampler State

MTLDepthStencilState and MTLSamplerState

Encapsulate fixed-function GPU pipeline state

Create during content loading along with pipeline state

Some Metal implementations may hash state objects to avoid duplication
Buffer and Texture Resources

MTLBuffer and MTLTexture
Buffer and Texture Resources
MTLBuffer and MTLTexture

Data consumed by GPU
Buffer and Texture Resources

MTLBuffer and MTLTexture

Data consumed by GPU

Create during content loading and reuse often
Buffer and Texture Resources

MTLBuffer and MTLTexture

Data consumed by GPU

Create during content loading and reuse often

Create multiple copies of dynamic resources in order to buffer updates
<table>
<thead>
<tr>
<th>State</th>
<th>When to Create</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>When to Create</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
</tr>
<tr>
<td>MTLLibrary</td>
<td>App building</td>
</tr>
<tr>
<td>State</td>
<td>When to Create</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>MTLLibrary</td>
<td>App building</td>
</tr>
<tr>
<td>MTLDevice</td>
<td>App initialization</td>
</tr>
<tr>
<td>MTLCommandQueue</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>When to Create</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>MTLLibrary</td>
<td>App building</td>
</tr>
<tr>
<td>MTLDevice</td>
<td>App initialization</td>
</tr>
<tr>
<td>MTLCommandQueue</td>
<td></td>
</tr>
<tr>
<td>MTLRenderPipelineState</td>
<td>Content loading</td>
</tr>
<tr>
<td>MTLComputePipelineState</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>When to Create</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>MTLLibrary</td>
<td>App building</td>
</tr>
<tr>
<td>MTLDevice</td>
<td>App initialization</td>
</tr>
<tr>
<td>MTLCommandQueue</td>
<td></td>
</tr>
<tr>
<td>MTLRenderPipelineState</td>
<td>Content loading</td>
</tr>
<tr>
<td>MTLComputePipelineState</td>
<td></td>
</tr>
<tr>
<td>MTLDepthStencilState</td>
<td>Content loading</td>
</tr>
<tr>
<td>MTLSamplerState</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>When to Create</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>MTLLibrary</td>
<td>App building</td>
</tr>
<tr>
<td>MTLDevice</td>
<td>App initialization</td>
</tr>
<tr>
<td>MTLCommandQueue</td>
<td></td>
</tr>
<tr>
<td>MTLRenderPipelineState</td>
<td>Content loading</td>
</tr>
<tr>
<td>MTLComputePipelineState</td>
<td></td>
</tr>
<tr>
<td>MTLDepthStencilState</td>
<td>Content loading</td>
</tr>
<tr>
<td>MTLSamplerState</td>
<td></td>
</tr>
<tr>
<td>MTLTexture</td>
<td>Content loading</td>
</tr>
<tr>
<td>MTLBuffer</td>
<td></td>
</tr>
</tbody>
</table>
Before
Create state and objects upfront and reuse

- Compile shader source offline
- Keep the rendering loop free of object creation
Buffer Dynamic Shared Resources
Dynamic Resources
Dynamic Resources

GPU resources updated by CPU many times after creation

- Shader constants
- Vertex and index buffers for dynamic geometry
- Dynamic textures
Shared Storage Mode

MTLStorageModeShared
Shared Storage Mode

MTLStorageModeShared

Creates resource in memory shared by CPU and GPU
Shared Storage Mode
MTLStorageModeShared

Creates resource in memory shared by CPU and GPU
Default storage mode on iOS
Shared Storage Mode
MTLStorageModeShared

Creates resource in memory shared by CPU and GPU
Default storage mode on iOS
Allows unsynchronized CPU access
• Data modified through persistent pointer
• App responsible for not overwriting data in use by GPU
Single Buffer

Time

CPU

GPU
Single Buffer

Time

CPU
- Update
- Constants Buffer

GPU
Single Buffer

Time

CPU
- Update
- Constants Buffer

GPU
- Execute
- Constants Buffer
Single Buffer

Time

CPU

Update

Constants Buffer

GPU

Execute

Constants Buffer

waitUntilCompleted
Single Buffer

CPU
- Update
  - Constants Buffer

GPU
- Execute
  - Constants Buffer

execute

Time

waitUntilCompleted
Single Buffer

- CPU: Update -> waitUntilCompleted -> Update
- GPU: Execute -> Constants Buffer
- Starved

Constants Buffer
Creating New Buffers per Frame

Time

CPU

GPU
Creating New Buffers per Frame
Triple Buffering Scheme

Time

CPU
- Update Buffer 1
- Update Buffer 2
- Update Buffer 3

GPU
- Execute Buffer 1
- Execute Buffer 2
- Execute Buffer 3
Triple Buffering Scheme

Time

CPU

- Update
- Buffer 2
- Buffer 1

GPU

- Execute
- Buffer 2
- Buffer 3
- Buffer 1
Triple Buffering Scheme

Time

CPU
- Update
- Update
- Update
- Update
- Update

GPU
- Execute
- Execute
- Execute
- Execute
- Execute

Buffer 3
Buffer 1
Buffer 2
Sample Code
MetalUniformStreaming

Metal Optimization Best Practices

Buffer dynamic shared resources

- Most efficient way of updating dynamic resources between frames
- Enforce safety by using semaphore to stop CPU getting too far ahead of GPU
Acquire the Drawable at the Latest Opportunity
Drawable Surfaces
Drawable Surfaces

Surface into which app draws its visible output
Drawable Surfaces

Surface into which app draws its visible output

CAMetalLayer has a limited pool of CAMetalDrawable objects

• Free drawables returned to pool at display intervals
Drawable Surfaces

Surface into which app draws its visible output
CAMetalLayer has a limited pool of CAMetalDrawable objects
  • Free drawables returned to pool at display intervals
Drawables may be used by multiple display stages at once:
  • Your application
  • GPU
  • Core Animation (if compositing enabled)
  • Display hardware
Acquiring a Drawable Surface
Acquiring a Drawable Surface

Drawable acquired with \texttt{nextDrawable} method
Acquiring a Drawable Surface

Drawable acquired with `nextDrawable` method
Method returns once a drawable is available
• Calling thread will block until at least next display interval if none available
MetalApp

FPS

0x1003a6680 <-> [MTLLayer nextDrawable]

1 0x10104b7e0 <-> [MTLDrawable texture]

2 0x10039d0c0 <-> [0x10103b540 commandBuffer]

CommandBuffer 0x10039d0c0

Shadow Buffer

G-buffer

G-buffer

Sun light

Fairy lights

Fairy sprites

UI

1009 [addCompletedHandler:0x16fd0f9d0]

1010 [presentDrawable:0x1003a6680]

1011 [commit]
CAMetalLayer `nextDrawable Called Early`

⚠️ Your application called CAMetalLayer `nextDrawable` earlier than needed. This may block execution until a CAMetalDrawable is available. Delay the call to `nextDrawable` until the drawable is needed for encoding.

: function index of NextDrawable = 20, function index of violating function = 21

AAPLRenderer.m
Your application called CAMetalLayer nextDrawable earlier than needed. This may block execution until a CAMetalDrawable is available. Delay the call to nextDrawable until the drawable is needed for encoding.

: function index of NextDrawable = 20, function index of violating function = 21

AAPLRenderer.mm
When to Acquire Your Drawable

1 Frame

- Command Buffer
- Shadow Pass
- G-buffer Pass
- Light Pass 1
- Light Pass 2
- Light Sprites
- UI Pass
- presentDrawable
- commit
When to Acquire Your Drawable

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Buffer</td>
<td></td>
</tr>
<tr>
<td>Shadow Pass</td>
<td></td>
</tr>
<tr>
<td>G-buffer Pass</td>
<td></td>
</tr>
<tr>
<td>Light Pass 1</td>
<td></td>
</tr>
<tr>
<td>Light Pass 2</td>
<td></td>
</tr>
<tr>
<td>Light Sprites</td>
<td></td>
</tr>
<tr>
<td>UI Pass</td>
<td></td>
</tr>
<tr>
<td>presentDrawable</td>
<td>commit</td>
</tr>
</tbody>
</table>

Too early
When to Acquire Your Drawable

1 Frame

- Command Buffer
- Shadow Pass
- G-buffer Pass
- Light Pass 1
- Light Pass 2
- Light Sprites
- UI Pass
- presentDrawable
- commit
- nextDrawable
Before
Metal Optimization Best Practices

Acquire the drawable at the latest opportunity

• Immediately before the render pass in which it is used
• Hides long latency if no drawables available
4

Don't Waste Render Command Encoders
Render Command Encoder
Render Command Encoder

Generates hardware commands for single rendering pass

• Specifies states for vertex and fragment stages of 3D pipeline
• Interleaves resources, state changes, and draw calls
• All rendering to a single set of render attachments
Render Encoders and iOS Device GPUs
Render Encoders and iOS Device GPUs

Tile-Based Deferred Renderer
Render Encoders and iOS Device GPUs

Tile-Based Deferred Renderer

Each render command encoder results in two GPU phases
Render Encoders and iOS Device GPUs

Tile-Based Deferred Renderer

Each render command encoder results in two GPU phases
• Vertex phase transforms and bins encoded draws into tiles
Tile-Based Deferred Renderer

Each render command encoder results in two GPU phases

- Vertex phase transforms and bins encoded draws into tiles
- Fragment phase processes visible pixels in each tile
Render Encoders and iOS Device GPUs

Tile-Based Deferred Renderer

Each render command encoder results in two GPU phases

- Vertex phase transforms and bins encoded draws into tiles
- Fragment phase processes visible pixels in each tile
  - Processing performed in fast on-chip tile buffers
MetalApp

FPS

0x10046fd0 <- [MTLLayer nextDrawable]
1 0x100457070 <- [MTLDrawable texture]
2 0x100447ca0 <- [0x100430bc0 commandBuffer]

CommandBuffer 0x100447ca0

Shadow Buffer

G-buffer

15 G-buffer <- [renderCommandEncoderWithDescriptor:<data>]
16 [setLabel:"G-buffer"]
17 [setDepthStencilState:0x10043adc0]
18 [endEncoding]
Before

After

19 ms
22 ms

19 ms
22 ms
Render Passes in Example App
Render Passes in Example App

- Shadow Pass
- G-Buffer
  - Albedo
  - Normals
  - Linear Depth
  - Stencil Buffer
  - Depth Buffer
- Light Pass 1
- Light Pass 2
- Light Output
- Light Sprites
- UI
- Drawables
Inefficient Use of Command Encoders

Your application created separate command encoders which can be combined into a single encoder for better performance.

AAPLRenderer.mm

- RenderCommandEncoder Sun light (0x1007a5530)
- RenderCommandEncoder Fairy lights (0x1007a5740)
Your application created separate command encoders which can be combined into a single encoder for better performance.
Inefficient Use of Command Encoders

Your application created separate command encoders which can be combined into a single encoder for better performance.

AAPLRenderer.mm

- RenderCommandEncoder Sun light (0x1007a5530)
- RenderCommandEncoder Fairy lights (0x1007a5740)
Merging Passes

- Shadow Pass
  - G-Buffer
    - Albedo
    - Normals
    - Linear Depth
      - Stencil Buffer
        - Shadow Buffer
      - Depth Buffer
    - Light Pass 1
      - Stencil Buffer
        - Light Pass 2
      - Depth Buffer
    - Light Output
  - Light Pass 1
  - Light Output
  - Light Pass 2
  - Light Output
  - Light Sprites
  - UI
  - Drawable
Merging Passes

Shadow Pass

G-Buffer

Albedo

Normals

Linear Depth

Stencil Buffer

Depth Buffer

Light Pass 1

Light Output

Light Pass 2

Light Output

Light Sprites

Light Output

Drawable
From Six Passes to Four Passes

- Shadow Pass
- G-Buffer
  - Albedo
  - Normals
  - Linear Depth
  - Stencil Buffer
  - Depth Buffer
- Light Pass
- Light Output
- Drawble
- UI
Six passes: 21 ms

Four passes: 18 ms
More Merging

- Shadow Pass
- G-Buffer
  - Albedo
  - Normals
  - Linear Depth
  - Stencil Buffer
  - Depth Buffer
- Light Pass
- Light Output
- UI
- Drawable
More Merging

- Shadow Pass
- G-Buffer
  - Albedo
  - Normals
  - Linear Depth
  - Stencil Buffer
  - Depth Buffer
- Light Pass
- Light Output
- Drawble
- UI
Down to Three Passes

- Shadow Pass
- G-Buffer and Light Pass
- Light Output
- UI
- Drawable
G-buffer and lighting
Four passes: 18 ms

Three passes: 17 ms
Color Attachment 0

Buffer:0x... 4.52 ms

Fragment Texture 0 0x10046e1100 - 2048 x 2048, RGBA8Unorm
Fragment Texture 1 0x1003d0f80 - 2048 x 2048, RGBA8Unorm
Fragment Texture 2 0x10045f70 - 2048 x 2048, RGBA8Unorm
Fragment Texture 3 0x10042f6a0 - 1024 x 1024, Depth32Float
Vertex Buffer 0 0x100504c10
Vertex Buffer 1 0x100513c40
Index Buffer 0x10050580, Offset=0x000282c0

"G-Buffer and lighting" (0x1003e0660)

Encoder Statistics 267 draw calls, 111MB load/store bandwidth
- Draw Calls = 267
- Load Bandwidth = 48MB
- Store Bandwidth = 63MB

Frame Statistics 271 draw calls, 127MB load/store bandwidth
- Frame Rate = 59.9fps
- Draw Calls = 271
- Command Buffers = 1
- Render Encoders = 3
- Load Bandwidth = 48MB
- Store Bandwidth = 79MB

MTL Command Encoder 0

Fragment Texture 0 0x10042e90 - 1024 x 1024, Depth32Float
Fragment Buffer 0 0x10050d780
Vertex Buffer 0 0x100504c10
Vertex Buffer 1 0x100513c40

RenderPassDescriptor 1536x2048 Color(x4)/Depth/Stencil
Visibility Result Buffer 0x0 Does not exist

Color Attachment 0 Load/Store Texture 0x1003be40
- Texture 0x1003be40 1536 x 2048, BGRA8Unorm
  - TextureLevel = 0
  - TextureSlice = 0
  - TextureDepthPlane = 0
  - Resolve Texture 0x0 Does not exist
    - ResolveTextureLevel = 0
    - ResolveTextureSlice = 0
    - ResolveTextureDepthPlane = 0
  - LoadAction = Load
  - StoreAction = Store
  - ClearColorRed = 0.82
  - ClearColorGreen = 0.73
  - ClearColorBlue = 0.64
  - ClearColorAlpha = 1.00

Color Attachment 1 Load/Store Texture 0x1003c9b10
Color Attachment 2 Load/Store Texture 0x1003a2640
Color Attachment 3 Load/Store Texture 0x1003d1bd0
Depth Attachment Clear/Store Texture 0x1003cd0a0
Stencil Attachment Clear/Store Texture 0x1003c6bb0
GBuffer Render (0x10042ee90) gBufferVert/gBufferFrag

RenderPipeline Performance: 5.87 ms (38.7%)

Fragment Texture 0 0x100461100 - 2048 x 2048, RGBA8Unorm
Fragment Texture 1 0x1003d0f80 - 2048 x 2048, RGBA8Unorm
Fragment Texture 2 0x10045fd70 - 2048 x 2048, RGBA8Unorm
Fragment Texture 3 0x10042aaa0 - 1024 x 1024, Depth32Float

Vertex Buffer 0 0x10050d780
Vertex Buffer 1 0x100513c40
Index Buffer index Buffer 0x100505580, Offset=0x000282c0

G-buffer and lighting (0x1003e0660)

Encoder Statistics: 267 draw calls, 111MB load/store bandwidth
- Draw Calls = 267
- Load Bandwidth = 48MB
- Store Bandwidth = 63MB

Frame Statistics: 271 draw calls, 127MB load/store bandwidth
- Frame Rate = 59.9fps
- Draw Calls = 271
- Command Buffers = 1
- Render Encoders = 3
- Load Bandwidth = 48MB
- Store Bandwidth = 79MB

Fragment Texture 3 0x10042aa8a0 - 1024 x 1024, Depth32Float
Fragment Buffer 0 0x10050d780
Fragment Buffer 1 0x10050d780
Vertex Buffer 0 0x10050d780
Vertex Buffer 1 0x100513c40

RenderPassDescriptor 1536x2048 Color(x4)/Depth/Stencil
Visibility Result Buffer 0x0 Does not exist

Color Attachment 0 Load/Store Texture 0x1003bfe40
- Texture 0x1003bfe40 1536 x 2048, BGRAB8Unorm
  - TextureLevel = 0
  - TextureSlice = 0
  - TextureDepthPlane = 0
  - Resolve Texture 0x0 Does not exist
  - ResolveTextureLevel = 0
  - ResolveTextureSlice = 0
  - ResolveTextureDepthPlane = 0
  - LoadAction = Load
  - StoreAction = Store
  -ClearColorRed = 0.82
  -ClearColorGreen = 0.73
  -ClearColorBlue = 0.64
  -ClearColorAlpha = 1.00

Color Attachment 1 Load/Store Texture 0x1003c9b10
Color Attachment 2 Load/Store Texture 0x1003a2640
Color Attachment 3 Load/Store Texture 0x1003d1bd0
Depth Attachment Clear/Store Texture 0x1003cd0a0
Stencil Attachment Clear/Store Texture 0x1003c8bb0

MTL Command Encoder
Load Bandwidth = 48MB
Store Bandwidth = 63MB

Frame Rate = 59.9fps
Draw Calls = 271
Command Buffers = 1
Render Encoders = 3
Load Bandwidth = 48MB
Store Bandwidth = 79MB

LoadAction = Load
StoreAction = Store

Load/Store Texture 0
Load/Store Texture 1
Load/Store Texture 2
Load/Store Texture 3
Load/Store Texture 4
Load/Store Texture 5
Load/Store Texture 6
Load/Store Texture 7
Clear/Store Texture 0
Clear/Store Texture 1
Clear/Store Texture 2
Clear/Store Texture 3
Clear/Store Texture 4
Clear/Store Texture 5
Clear/Store Texture 6
Clear/Store Texture 7

MTL Command Encoder
Load Bandwidth = 0B
Store Bandwidth = 12MB

Frame Rate = 60.0fps
Draw Calls = 271
Command Buffers = 1
Render Encoders = 3
Load Bandwidth = 0B
Store Bandwidth = 28MB

LoadAction = Clear
StoreAction = Store

MTL Command Encoder
Metal Optimization Best Practices

Don't waste render encoders
  • Don't start encoding if there's nothing to draw
  • Merge encoders that render to the same attachments
Consider Multithreading if Still CPU-Bound
Metal Multithreading
Metal Multithreading

Metal designed to facilitate multithreading
Metal Multithreading

Metal designed to facilitate multithreading
Thread-safe, efficient, and scalable implementation
Metal Multithreading

Metal designed to facilitate multithreading
Thread-safe, efficient, and scalable implementation
Multiple command buffers can be encoded in parallel
  • App controls execution order
Time

Thread 1
- Render Pass 1

Thread 2
- Render Pass 2

GPU
- Render Pass 1
- Render Pass 2

1 Frame
Thread 1  Thread 2  Thread 3

Command Buffer  Command Buffer  Command Buffer

Command Queue  Device
Thread 1

- Render Command Encoder

- Command Buffer

Thread 2

- Render Command Encoder

- Command Buffer

Thread 3

- Render Command Encoder
- Compute Command Encoder
- Blit Command Encoder

- Command Buffer

Command Queue

Device
Setting Up Threaded Passes

// create one command buffer per thread
id <MTLCommandBuffer> commandBuffer1 = [commandQueue commandBuffer];
id <MTLCommandBuffer> commandBuffer2 = [commandQueue commandBuffer];

// create and initialize render pass descriptors
// reserve each buffer a place in command queue to enforce GPU submission order
[commandBuffer1 enqueue];
[commandBuffer2 enqueue];

// create per-thread command encoders
id <MTLRenderCommandEncoder> pass1RCE = 
    [commandBuffer1 renderCommandEncoderWithDescriptor:renderPass1Desc];
id <MTLRenderCommandEncoder> pass2RCE = 
    [commandBuffer2 renderCommandEncoderWithDescriptor:renderPass2Desc];

// encode commands and commit buffer for each thread
[pass1RCE draw...]; [pass2RCE draw...];
[pass1RCE endEncoding]; [pass2RCE endEncoding];
[commandBuffer1 commit]; [commandBuffer2 commit];
// create one command buffer per thread
id <MTLCommandBuffer> commandBuffer1 = [commandQueue commandBuffer];
id <MTLCommandBuffer> commandBuffer2 = [commandQueue commandBuffer];

// create and initialize render pass descriptors
// reserve each buffer a place in command queue to enforce GPU submission order
[commandBuffer1 enqueue];
[commandBuffer2 enqueue];

// create per-thread command encoders
id <MTLRenderCommandEncoder> pass1RCE =
    [commandBuffer1 renderCommandEncoderWithDescriptor:renderPass1Desc];
id <MTLRenderCommandEncoder> pass2RCE =
    [commandBuffer2 renderCommandEncoderWithDescriptor:renderPass2Desc];

// encode commands and commit buffer for each thread
[pass1RCE draw...];      [pass2RCE draw...];
[pass1RCE endEncoding];  [pass2RCE endEncoding];
[commandBuffer1 commit]; [commandBuffer2 commit];
Setting Up Threaded Passes

// create one command buffer per thread
id <MTLCommandBuffer> commandBuffer1 = [commandQueue commandBuffer];
id <MTLCommandBuffer> commandBuffer2 = [commandQueue commandBuffer];

// create and initialize render pass descriptors
// reserve each buffer a place in command queue to enforce GPU submission order
[commandBuffer1 enqueue];
[commandBuffer2 enqueue];

// create per-thread command encoders
id <MTLRenderCommandEncoder> pass1RCE =
    [commandBuffer1 renderCommandEncoderWithDescriptor:renderPass1Desc];
id <MTLRenderCommandEncoder> pass2RCE =
    [commandBuffer2 renderCommandEncoderWithDescriptor:renderPass2Desc];

// encode commands and commit buffer for each thread
[pass1RCE draw...]; [pass2RCE draw...];
[pass1RCE endEncoding]; [pass2RCE endEncoding];
[commandBuffer1 commit]; [commandBuffer2 commit];
Setting Up Threaded Passes

// create one command buffer per thread
id <MTLCommandBuffer> commandBuffer1 = [commandQueue commandBuffer];
id <MTLCommandBuffer> commandBuffer2 = [commandQueue commandBuffer];

// create and initialize render pass descriptors
// reserve each buffer a place in command queue to enforce GPU submission order
[commandBuffer1 enqueue];
[commandBuffer2 enqueue];

// create per-thread command encoders
id <MTLRenderCommandEncoder> pass1RCE =
    [commandBuffer1 renderCommandEncoderWithDescriptor:renderPass1Desc];
id <MTLRenderCommandEncoder> pass2RCE =
    [commandBuffer2 renderCommandEncoderWithDescriptor:renderPass2Desc];

// encode commands and commit buffer for each thread
[pass1RCE draw...];         [pass2RCE draw...];
[pass1RCE endEncoding];     [pass2RCE endEncoding];
[commandBuffer1 commit];    [commandBuffer2 commit];
Setting Up Threaded Passes

// create one command buffer per thread
id <MTLCommandBuffer> commandBuffer1 = [commandQueue commandBuffer];
id <MTLCommandBuffer> commandBuffer2 = [commandQueue commandBuffer];

// create and initialize render pass descriptors
// reserve each buffer a place in command queue to enforce GPU submission order
[commandBuffer1 enqueue];
[commandBuffer2 enqueue];

// create per-thread command encoders
id <MTLRenderCommandEncoder> pass1RCE =
    [commandBuffer1 renderCommandEncoderWithDescriptor:renderPass1Desc];
id <MTLRenderCommandEncoder> pass2RCE =
    [commandBuffer2 renderCommandEncoderWithDescriptor:renderPass2Desc];

// encode commands and commit buffer for each thread
[pass1RCE draw...];
[pass1RCE endEncoding];
[commandBuffer1 commit];
[pass2RCE draw...];
[pass2RCE endEncoding];
[commandBuffer2 commit];
Command Queue ➔ Device
Threads 1–3
Threads 1–3

Parallel Render Command Encoder

Command Buffer

Command Queue

Device
Threads 1–3

Parallel Render Command Encoder

Command Buffer

Command Queue

Device
Setting Up Parallel Encoding

// create one command buffer per parallel render command encoder
id <MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// create and initialize render pass descriptor
// create a parallel render command encoder
id <MTLParallelRenderCommandEncoder> parallelRCE =
    [commandBuffer parallelRenderCommandEncoderWithDescriptor:renderPassDesc];

// create subordinate encoders in GPU submission order
id <MTLRenderCommandEncoder> rCE1 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE2 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE3 = [parallelRCE renderCommandEncoder];

// encode commands on each thread
[rCE1 draw...];        [rCE2 draw...];        [rCE3 draw...];
[rCE1 endEncoding];    [rCE2 endEncoding];    [rCE3 endEncoding];

// all subordinate encoders must end encoding before MTLParallelRenderCommandEncoder
[parallelRCE endEncoding];
[commandBuffer commit];
Setting Up Parallel Encoding

// create one command buffer per parallel render command encoder
id <MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// create and initialize render pass descriptor
// create a parallel render command encoder
id <MTLParallelRenderCommandEncoder> parallelRCE =
    [commandBuffer parallelRenderCommandEncoderWithDescriptor:renderPassDesc];

// create subordinate encoders in GPU submission order
id <MTLRenderCommandEncoder> rCE1 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE2 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE3 = [parallelRCE renderCommandEncoder];

// encode commands on each thread
[rCE1 draw...];        [rCE2 draw...];        [rCE3 draw...];
[rCE1 endEncoding];    [rCE2 endEncoding];    [rCE3 endEncoding];

// all subordinate encoders must end encoding before MTLParallelRenderCommandEncoder
[parallelRCE endEncoding];
[commandBuffer commit];
Setting Up Parallel Encoding

// create one command buffer per parallel render command encoder
id <MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// create and initialize render pass descriptor
// create a parallel render command encoder
id <MTLParallelRenderCommandEncoder> parallelRCE =
    [commandBuffer parallelRenderCommandEncoderWithDescriptor:renderPassDesc];

// create subordinate encoders in GPU submission order
id <MTLRenderCommandEncoder> rCE1 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE2 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE3 = [parallelRCE renderCommandEncoder];

// encode commands on each thread
[rCE1 draw...];         [rCE2 draw...];         [rCE3 draw...];
[rCE1 endEncoding];     [rCE2 endEncoding];     [rCE3 endEncoding];

// all subordinate encoders must end encoding before MTLParallelRenderCommandEncoder
[parallelRCE endEncoding];
[commandBuffer commit];
Setting Up Parallel Encoding

// create one command buffer per parallel render command encoder
id <MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// create and initialize render pass descriptor
// create a parallel render command encoder
id <MTLParallelRenderCommandEncoder> parallelRCE =
    [commandBuffer parallelRenderCommandEncoderWithDescriptor:renderPassDesc];

// create subordinate encoders in GPU submission order
id <MTLRenderCommandEncoder> rCE1 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE2 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE3 = [parallelRCE renderCommandEncoder];

// encode commands on each thread
[rCE1 draw...];        [rCE2 draw...];        [rCE3 draw...];
[rCE1 endEncoding];    [rCE2 endEncoding];    [rCE3 endEncoding];

// all subordinate encoders must end encoding before MTLParallelRenderCommandEncoder
[parallelRCE endEncoding];
[commandBuffer commit];
Setting Up Parallel Encoding

// create one command buffer per parallel render command encoder
id <MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// create and initialize render pass descriptor
// create a parallel render command encoder
id <MTLParallelRenderCommandEncoder> parallelRCE =
    [commandBuffer parallelRenderCommandEncoderWithDescriptor:renderPassDesc];

// create subordinate encoders in GPU submission order
id <MTLRenderCommandEncoder> rCE1 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE2 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE3 = [parallelRCE renderCommandEncoder];

// encode commands on each thread
[rCE1 draw...];        [rCE2 draw...];        [rCE3 draw...];
[rCE1 endEncoding];    [rCE2 endEncoding];    [rCE3 endEncoding];

// all subordinate encoders must end encoding before MTLParallelRenderCommandEncoder
[parallelRCE endEncoding];
[commandBuffer commit];
Setting Up Parallel Encoding

// create one command buffer per parallel render command encoder
id <MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// create and initialize render pass descriptor
// create a parallel render command encoder
id <MTLParallelRenderCommandEncoder> parallelRCE =
    [commandBuffer parallelRenderCommandEncoderWithDescriptor:renderPassDesc];

// create subordinate encoders in GPU submission order
id <MTLRenderCommandEncoder> rCE1 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE2 = [parallelRCE renderCommandEncoder];
id <MTLRenderCommandEncoder> rCE3 = [parallelRCE renderCommandEncoder];

// encode commands on each thread
[rCE1 draw...];        [rCE2 draw...];        [rCE3 draw...];
[rCE1 endEncoding];    [rCE2 endEncoding];    [rCE3 endEncoding];

// all subordinate encoders must end encoding before MTLParallelRenderCommandEncoder
[parallelRCE endEncoding];
[commandBuffer commit];
Before

1 thread: 25 ms

After

2 threads: 15 ms
Metal Optimization Best Practices

Consider multithreading if still CPU-bound

• Render passes can be simultaneously encoded on separate threads
• Use MTLParallelRenderCommandEncoder to split long passes
Summary
Summary

Metal System Trace offers a new insight into Metal app performance
Summary

Metal System Trace offers a new insight into Metal app performance

• Use in conjunction with Xcode to profile early and often
Summary

Metal System Trace offers a new insight into Metal app performance
• Use in conjunction with Xcode to profile early and often

Follow Metal best practices to maximize app performance
Summary

Metal System Trace offers a new insight into Metal app performance

• Use in conjunction with Xcode to profile early and often

Follow Metal best practices to maximize app performance

• Create expensive objects upfront and reuse
Metal System Trace offers a new insight into Metal app performance

- Use in conjunction with Xcode to profile early and often

Follow Metal best practices to maximize app performance

- Create expensive objects upfront and reuse
- Buffer dynamic shared resources
Summary

Metal System Trace offers a new insight into Metal app performance

- Use in conjunction with Xcode to profile early and often

Follow Metal best practices to maximize app performance

- Create expensive objects upfront and reuse
- Buffer dynamic shared resources
- Acquire the drawable at the latest opportunity
Summary

Metal System Trace offers a new insight into Metal app performance

- Use in conjunction with Xcode to profile early and often

Follow Metal best practices to maximize app performance

- Create expensive objects upfront and reuse
- Buffer dynamic shared resources
- Acquire the drawable at the latest opportunity
- Don’t waste render command encoders
Summary

Metal System Trace offers a new insight into Metal app performance
• Use in conjunction with Xcode to profile early and often

Follow Metal best practices to maximize app performance
• Create expensive objects upfront and reuse
• Buffer dynamic shared resources
• Acquire the drawable at the latest opportunity
• Don't waste render command encoders
• Consider multithreading if still CPU-bound
Overall Progress
Overall Progress

No runtime shader compilation
Overall Progress

GPU workload within budget
Overall Progress

CPU workload within budget
Overall Progress

CPU encodes passes on multiple threads
Overall Progress

60 FPS!
More Information

Documentation and Videos
Metal Documentation
http://developer.apple.com/metal

Apple Developer Forums
http://developer.apple.com/forums

General Inquiries
Allan Schaffer, Graphics and Games Technologies Evangelist
aschaffer@apple.com
| Metal | Graphics Lab D | Friday 12:00PM |