Delivering Optimized Metal Apps and Games

Guillem Viñals Gangoletts  GPU Software
Samuel Colbran  GPU Software
Ubaka Onyechi  GPU Software
Metal OK Practices
Metal OK Practices
Metal OK Practices
Metal Acceptable Practices
Metal OK Practices
Metal Acceptable Practices
Metal OK Practices
Metal Acceptable Practices
Metal Practices That Will Do the Trick
Metal OK Practices
Metal Acceptable Practices
Metal Practices That Will Do the Trick
Metal Best Practices
Afterpulse — Elite Army
Digital Legends Entertainment

Cascaded shadow maps
Deferred shading
・Physically-based HDR rendering
・Image based lighting
・Parallax cubemap reflections
Post-process
・Bloom, tonemap, anti-aliasing, and more
General Performance
Memory Bandwidth
Memory Footprint
General Performance

Memory Bandwidth

Memory Footprint
Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
Best Practices
1. Choose the right resolution

Each effect may require a different resolution

Best practice
• Consider the image quality and performance trade-off
• Composite the game UI at native resolution
Depth Pass
742x1608

Deferred Pass
742x1608

UI Pass
1124x2436

Shadow Maps
1024x1024

SSAO Passes
562x1218

Shadow Maps
1024x1024

Deferred Pass
742x1608

UI Pass
1124x2436
Depth Pass 742x1608

Deferred Pass 742x1608

SSAO Passes 562x1218

Shadow Maps 1024x1024

UI Pass 1124x2436
Best Practices

2. Minimize non-opaque overdraw

Processing multiple fragments per pixel causes overdraw

iOS GPUs are very efficient at reducing opaque overdraw

Best practice
• Render opaque meshes first and translucent meshes later
• Don't render invisible (fully transparent) meshes
### Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Median</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Time</td>
<td>591 ns</td>
<td>9.19 ms</td>
</tr>
<tr>
<td>Vertex Stage Time</td>
<td>65.69%</td>
<td>75.27%</td>
</tr>
<tr>
<td>Fragment Stage Time</td>
<td>34.32%</td>
<td>95.04%</td>
</tr>
</tbody>
</table>

### Performance Limiters

<table>
<thead>
<tr>
<th>Performance Limiter</th>
<th>Medal</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top Performance Limiter</td>
<td>15.78%</td>
<td>96.73%</td>
</tr>
<tr>
<td>ALU Limiter</td>
<td>15.78%</td>
<td>61.55%</td>
</tr>
<tr>
<td>Texture Read Limiter</td>
<td>0%</td>
<td>56.16%</td>
</tr>
<tr>
<td>Texture Write Limiter</td>
<td>0.01%</td>
<td>96.73%</td>
</tr>
<tr>
<td>Buffer Read Limiter</td>
<td>6.49%</td>
<td>25.30%</td>
</tr>
<tr>
<td>Buffer Write Limiter</td>
<td>0.01%</td>
<td>0.02%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Read Limiter</td>
<td>0%</td>
<td>0.01%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Write Limiter</td>
<td>0%</td>
<td>24.85%</td>
</tr>
<tr>
<td>FS Interpretation Limiter</td>
<td>0%</td>
<td>14.8%</td>
</tr>
</tbody>
</table>

### Performance Utilization

<table>
<thead>
<tr>
<th>Utilization</th>
<th>Medal</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>F32 Utilization</td>
<td>6.21%</td>
<td>61.55%</td>
</tr>
<tr>
<td>F16 Utilization</td>
<td>0%</td>
<td>1.93%</td>
</tr>
<tr>
<td>Texture Read Utilization</td>
<td>0%</td>
<td>19.18%</td>
</tr>
<tr>
<td>Texture Write Utilization</td>
<td>0%</td>
<td>38.34%</td>
</tr>
<tr>
<td>Buffer Read Utilization</td>
<td>5.44%</td>
<td>19.73%</td>
</tr>
<tr>
<td>Buffer Write Utilization</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Read Utilization</td>
<td>0%</td>
<td>0.01%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Write Utilization</td>
<td>0%</td>
<td>19.98%</td>
</tr>
</tbody>
</table>

### Vertices

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Medal</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>480</td>
<td>41,964</td>
</tr>
<tr>
<td>Vertices Reused</td>
<td>59.1%</td>
<td>81.86%</td>
</tr>
<tr>
<td>Pixel per Vertex</td>
<td>685,120</td>
<td></td>
</tr>
<tr>
<td>Vertices per Second</td>
<td>219,819,900</td>
<td>615,340,390</td>
</tr>
</tbody>
</table>

### Vertex Shader

<table>
<thead>
<tr>
<th>Vertex Shader</th>
<th>Medal</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Shader Time</td>
<td>62.03%</td>
<td>70.99%</td>
</tr>
<tr>
<td>VS Invocations</td>
<td>176</td>
<td>25,852</td>
</tr>
<tr>
<td>VS Occupancy</td>
<td>3.99%</td>
<td>3.99%</td>
</tr>
<tr>
<td>VS ALU Instructions</td>
<td>10,608</td>
<td>3,283,204</td>
</tr>
<tr>
<td>VS ALU Float Instructions</td>
<td>40.30%</td>
<td>44.89%</td>
</tr>
<tr>
<td>VS ALU Integer Instructions</td>
<td>1.93%</td>
<td>25%</td>
</tr>
<tr>
<td>VS ALU Limit</td>
<td>15.09%</td>
<td>15.23%</td>
</tr>
<tr>
<td>VS Buffer Read Limit</td>
<td>5.69%</td>
<td>13.12%</td>
</tr>
<tr>
<td>VS Buffer Write Limit</td>
<td>0.01%</td>
<td>0.02%</td>
</tr>
<tr>
<td>VS Bytes Read From Main Memory</td>
<td>4.69 MB</td>
<td>14.74 MB</td>
</tr>
</tbody>
</table>

### Buffer

<table>
<thead>
<tr>
<th>Byte</th>
<th>Medal</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>1.97 MB</td>
<td>A MB</td>
</tr>
</tbody>
</table>
### GPU Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Median</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPU Time</strong></td>
<td>591 ns</td>
<td>9.19 ms</td>
</tr>
<tr>
<td><strong>Vertex Stage Time</strong></td>
<td>66.69%</td>
<td>75.27%</td>
</tr>
<tr>
<td><strong>Fragment Stage Time</strong></td>
<td>34.32%</td>
<td>95.04%</td>
</tr>
<tr>
<td><strong>Performance Limiters</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top Performance Limiter</td>
<td>15.78%</td>
<td>96.73%</td>
</tr>
<tr>
<td>ALU Limiter</td>
<td>15.78%</td>
<td>61.55%</td>
</tr>
<tr>
<td>Texture Read Limiter</td>
<td>0%</td>
<td>56.16%</td>
</tr>
<tr>
<td>Texture Write Limiter</td>
<td>0.01%</td>
<td>95.73%</td>
</tr>
<tr>
<td>Buffer Read Limiter</td>
<td>6.49%</td>
<td>25.30%</td>
</tr>
<tr>
<td>Buffer Write Limiter</td>
<td>0.01%</td>
<td>0.03%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Read Limiter</td>
<td>0%</td>
<td>0.01%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Write Limiter</td>
<td>0%</td>
<td>24.85%</td>
</tr>
<tr>
<td>FS Interpretation Limiter</td>
<td>0%</td>
<td>14.86%</td>
</tr>
<tr>
<td><strong>Performance Utilization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F32 Utilization</td>
<td>6.21%</td>
<td>61.55%</td>
</tr>
<tr>
<td>F16 Utilization</td>
<td>0%</td>
<td>1.93%</td>
</tr>
<tr>
<td>Texture Read Utilization</td>
<td>0%</td>
<td>19.18%</td>
</tr>
<tr>
<td>Texture Write Utilization</td>
<td>0%</td>
<td>36.34%</td>
</tr>
<tr>
<td>Buffer Read Utilization</td>
<td>5.44%</td>
<td>19.73%</td>
</tr>
<tr>
<td>Buffer Write Utilization</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Read Utilization</td>
<td>0%</td>
<td>0.01%</td>
</tr>
<tr>
<td>ThreadGroup/ImageBlock Write Utilization</td>
<td>0%</td>
<td>19.98%</td>
</tr>
<tr>
<td><strong>Vertices</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertices</td>
<td>480</td>
<td>41.964</td>
</tr>
<tr>
<td>Vertices Reused</td>
<td>59.1%</td>
<td>81.86%</td>
</tr>
<tr>
<td>Pixel per Vertex</td>
<td>0</td>
<td>685,120</td>
</tr>
<tr>
<td>Vertices per Second</td>
<td>219,819,900</td>
<td>615,340,320</td>
</tr>
<tr>
<td><strong>Vertex Shader</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex Shader Time</td>
<td>62.03%</td>
<td>70.99%</td>
</tr>
<tr>
<td><strong>VS Invocations</strong></td>
<td>176</td>
<td>25,862</td>
</tr>
<tr>
<td><strong>VS Occupancy</strong></td>
<td>3.94%</td>
<td>3.94%</td>
</tr>
<tr>
<td><strong>VS ALU Instructions</strong></td>
<td>10,608</td>
<td>3,283,204</td>
</tr>
<tr>
<td><strong>VS ALU Float Instructions</strong></td>
<td>40.39%</td>
<td>44.89%</td>
</tr>
<tr>
<td><strong>VS ALU Integer Instructions</strong></td>
<td>1.93%</td>
<td>25%</td>
</tr>
<tr>
<td><strong>VS ALU Limit</strong></td>
<td>15.09%</td>
<td>15.23%</td>
</tr>
<tr>
<td><strong>VS Buffer Read Limit</strong></td>
<td>8.60%</td>
<td>13.12%</td>
</tr>
<tr>
<td><strong>VS Buffer Write Limit</strong></td>
<td>0.01%</td>
<td>0.02%</td>
</tr>
<tr>
<td><strong>VS Bytes Read From Main Memory</strong></td>
<td>4.68 MB</td>
<td></td>
</tr>
</tbody>
</table>

**GPU Time** represents the total time spent on GPU operations. The **Vertex Stage Time** and **Fragment Stage Time** indicate the time spent processing vertices and fragments, respectively. The **Performance Limiters** show the percentage of time spent on various operations such as ALU, Texture, and Buffer. The **Performance Utilization** section provides utilizations for various stages such as F32 and F16. The **Vertices** section shows statistics on vertex operations, including invocations, occupancy, and buffer usage.
Overdraw = \frac{2740480}{2740500} = \approx 1.00
Overdraw = \frac{2740480}{2740500} = \sim 1.00
Best Practices

3. Submit GPU work early

Scheduling all offscreen GPU work early
- Improves latency and responsiveness
- Allows the system to adapt to the workload

Best practice
- Submit all offscreen GPU work early
- Get the drawable as late in the frame as possible
// Off-screen command buffer

let offscreenCb = commandQueue.makeCommandBuffer()!

// ... Encode off-screen work ...

offscreenCb.commit()

let drawable = caMetalLayer.nextDrawable()!

// On-screen command buffer

let onscreenCb = commandQueue.makeCommandBuffer()!

// .. Encode on-screen work ...

onscreenCb.present(drawable, afterMinimumDuration: 33.0 / 1000)
onscreenCb.commit()
// Off-screen command buffer

let offscreenCb = commandQueue.makeCommandBuffer()!

// ... Encode off-screen work ...

offscreenCb.commit()

let drawable = caMetalLayer.nextDrawable()!

// On-screen command buffer

let onscreenCb = commandQueue.makeCommandBuffer()!

// .. Encode on-screen work ...

onscreenCb.present(drawable, afterMinimumDuration: 33.0 / 1000)
onscreenCb.commit()
// Off-screen command buffer

let offscreenCb = commandQueue.makeCommandBuffer()!

// ... Encode off-screen work ...

offscreenCb.commit()

let drawable = caMetalLayer.nextDrawable()!

// On-screen command buffer

let onscreenCb = commandQueue.makeCommandBuffer()!

// .. Encode on-screen work ...

onscreenCb.present(drawable, afterMinimumDuration: 33.0 / 1000)
onscreenCb.commit()
// Off-screen command buffer

let offscreenCb = commandQueue.makeCommandBuffer()!

// ... Encode off-screen work ...

offscreenCb.commit()

let drawable = caMetalLayer.nextDrawable()!

// On-screen command buffer

let onscreenCb = commandQueue.makeCommandBuffer()!

// .. Encode on-screen work ...

onscreenCb.present(drawable, afterMinimumDuration: 33.0 / 1000)
onscreenCb.commit()
Best Practices
4. Stream resources efficiently

Allocating resources takes time

Streaming resources from the render thread may cause stalls

Best practice
• Consider the memory and performance trade-off
• Allocate and load GPU resources at launch time
• Allocate and stream new resources from a dedicated thread
Metal System Trace
Metal System Trace
Best Practices
5. Design for sustained performance

Designing your game for sustained performance
• Improves thermals
• Improves stability and responsiveness

Best practice
• Test your game under serious thermal state
• Consider tuning your game for serious thermal state
Device Conditions

Designing for Adverse Network and Temperature Conditions
Device Conditions

Designing for Adverse Network and Temperature Conditions

Friday, 4:20
Xcode Energy Gauge

Energy Impact

Component Utilization

Application State

Thermal State

Overhead

High CPU Utilization

Network Activity
Xcode Energy Gauge

![Diagram of Energy Gauge with values and labels]
General Performance
Memory Bandwidth
Memory Footprint
Why Bandwidth?

Memory transfers are costly

iOS devices have
- Shared memory between the CPU and GPU
- Dedicated memory for the GPU

Metal is designed to help you leverage both
Textures
Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
6. Compress texture assets
7. Optimize for faster GPU access
8. Choose the right pixel format
Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
6. Compress texture assets
7. Optimize for faster GPU access
8. Choose the right pixel format
Best Practices
6. Compress texture assets

Sampling large textures may be inefficient

Best practice
• Compress all texture assets — ASTC, PVRTC, and more
• Generate mipmaps for textures that may be minified
Memory Savings of Texture Compression

2048x2048
RGBA8Unorm

16MB
Memory Savings of Texture Compression

2048x2048 RGBA8Unorm
16MB

2048x2048 PVRTC_RGB_4BPP (mip mapped)
2.7MB
Metal Memory Viewer
Lossless Texture Compression

Some textures can’t be compressed ahead of time
• Render targets
• Generated at runtime

A12 and later support lossless texture compression
Best Practices
7. Optimize for faster GPU access

Configuring textures correctly allows faster GPU access

Best practice
• Use **private** storage mode
• Don’t set the **unknown** usage flag
• Don’t set unnecessary usage flags (i.e. **shaderWrite** or **pixelView**)
• For **shared** textures, explicitly optimize after any CPU updates
// Create a texture with optimal GPU access

// ...

textureDescriptor.storageMode = .private
textureDescriptor.usage = [.shaderRead, .renderTarget]

let texture = device.makeTexture(descriptor: textureDescriptor)
// Create a texture with optimal GPU access

// ...

textureDescriptor.storageMode = .private
textureDescriptor.usage = [.shaderRead, .renderTarget]

let texture = device.makeTexture(descriptor: textureDescriptor)
// Create a texture with optimal GPU access

// ...
textureDescriptor.storageMode = .private
textureDescriptor.usage = [.shaderRead, .renderTarget]

let texture = device.makeTexture(descriptor: textureDescriptor)
// Optimize Shared texture after CPU update

// ...
textureDescriptor.storageMode = .shared
textureDescriptor.usage = .shaderRead

let texture = device.makeTexture(descriptor: textureDescriptor)!

// ...
texture.replace(region: region, mipmapLevel: 0, withBytes: bytes, bytesPerRow: bytesPerRow)

let blitCommandEncoder = commandBuffer.makeBlitCommandEncoder()!
blitCommandEncoder.optimizeContentsForGPUAccess(texture: texture)
blitCommandEncoder.endEncoding()}
// Optimize Shared texture after CPU update

// ...

let textureDescriptor = 

    textureDescriptor.storageMode = .shared
    textureDescriptor.usage = .shaderRead

let texture = device.makeTexture(descriptor: textureDescriptor)!

// ...

texture.replace(region: region, mipmapLevel: 0, withBytes: bytes, bytesPerRow: bytesPerRow)

let blitCommandEncoder = commandBuffer.makeBlitCommandEncoder()!
blitCommandEncoder.optimizeContentsForGPUAccess(texture: texture)
blitCommandEncoder.endEncoding()
// Optimize Shared texture after CPU update

// ...
textureDescriptor.storageMode = .shared
textureDescriptor.usage = .shaderRead

let texture = device.makeTexture(descriptor: textureDescriptor)!

// ...
texture.replace(region: region, mipmapLevel: 0, withBytes: bytes, bytesPerRow: bytesPerRow)

let blitCommandEncoder = commandBuffer.makeBlitCommandEncoder()!
blitCommandEncoder.optimizeContentsForGPUAccess(texture: texture)
blitCommandEncoder.endEncoding()
// Optimize Shared texture after CPU update

// ...
textureDescriptor.storageMode = .shared
textureDescriptor.usage = .shaderRead

let texture = device.makeTexture(descriptor: textureDescriptor)!

// ...
texture.replace(region: region, mipmapLevel: 0, withBytes: bytes, bytesPerRow: bytesPerRow)

let blitCommandEncoder = commandBuffer.makeBlitCommandEncoder()!
blitCommandEncoder.optimizeContentsForGPUAccess(texture: texture)
blitCommandEncoder.endEncoding()
Metal Memory Viewer
Best Practices
8. Choose the right pixel format

Larger pixel formats use more bandwidth

Sampling rate depends on pixel format

Best practice
• Avoid pixel formats with unnecessary channels
  • For example, using RGBA16 to store 2-component data
• Use lower precision whenever possible
Conventional 32-bit formats

YCbCr (GBGR422 and BGRG422)
RG11B10Float
RGB9E5Float

64-bit formats

128-bit formats

Sampling Rate (A12 and later)

- Conventional 32-bit formats: 1.0x
- YCbCr (GBGR422 and BGRG422): 0.5x
- 64-bit formats: 0.5x
- 128-bit formats: 0.25x
Metal Memory Viewer
Metal Memory Viewer
Render Targets
Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
6. Compress texture assets
7. Optimize for faster GPU access
8. Choose the right pixel format
9. Optimize load and store actions
10. Optimize multi-sampled textures
11. Leverage tile memory
Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
6. Compress texture assets
7. Optimize for faster GPU access
8. Choose the right pixel format
9. Optimize load and store actions
10. Optimize multi-sampled textures
11. Leverage tile memory
Best Practices
9. Optimize load and store actions

Loading or storing render targets costs bandwidth

Suboptimal configuration may create false dependencies

Best practice
• Don't load or store render targets unless it's necessary
// Configure color attachment 1 as transient

// ...
renderPassDescriptor.colorAttachments[1].texture = texture
renderPassDescriptor.colorAttachments[1].loadAction = .clear
renderPassDescriptor.colorAttachments[1].storeAction = .dontCare
// Configure color attachment 1 as transient

// ...

renderPassDescriptor.colorAttachments[1].texture = texture
renderPassDescriptor.colorAttachments[1].loadAction = .clear
renderPassDescriptor.colorAttachments[1].storeAction = .dontCare
<table>
<thead>
<tr>
<th>Color0: FAll0...nColor</th>
<th>Color1: BloomColor</th>
<th>Depth: DepthStencil</th>
<th>Stencil: DepthStencil</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Format</td>
<td>BGRA8Unorm</td>
<td>Dimensions</td>
<td>1125x2436</td>
</tr>
<tr>
<td>Allocated Size</td>
<td>10.97 MiB</td>
<td>Load Action</td>
<td>Clear</td>
</tr>
<tr>
<td>Store Action</td>
<td>Store</td>
<td>Store Action</td>
<td>Don'tCare</td>
</tr>
<tr>
<td>Pixel Format</td>
<td>BGRA8Unorm</td>
<td>Dimensions</td>
<td>1125x2436</td>
</tr>
<tr>
<td>Allocated Size</td>
<td>10.97 MiB</td>
<td>Load Action</td>
<td>Clear</td>
</tr>
<tr>
<td>Store Action</td>
<td>Store</td>
<td>Store Action</td>
<td>Don'tCare</td>
</tr>
<tr>
<td>Pixel Format</td>
<td>Depth...encil8</td>
<td>Dimensions</td>
<td>1125x2436</td>
</tr>
<tr>
<td>Allocated Size</td>
<td>13.78 MiB</td>
<td>Load Action</td>
<td>Clear</td>
</tr>
<tr>
<td>Store Action</td>
<td>Don'tCare</td>
<td>Store Action</td>
<td>Don'tCare</td>
</tr>
<tr>
<td>Pixel Format</td>
<td>Depth...encil8</td>
<td>Dimensions</td>
<td>1125x2436</td>
</tr>
<tr>
<td>Allocated Size</td>
<td>13.78 MiB</td>
<td>Load Action</td>
<td>Clear</td>
</tr>
<tr>
<td>Store Action</td>
<td>Don'tCare</td>
<td>Store Action</td>
<td>Don'tCare</td>
</tr>
</tbody>
</table>
Best Practices

10. Optimize multi-sampled textures

iOS devices have very efficient MSAA
• Resolve from tile memory
• Explicit color coverage control, custom resolves, and more

Best practice
• Consider MSAA over native resolution
• Don’t load or store the multisample texture
• Set the storage mode of the multisample texture to memoryless
// Create a multisample texture

// ...
textureDescriptor.textureType = .type2DMultisample
textureDescriptor.sampleCount = 4
textureDescriptor.storageMode = .memoryless

let msaaTexture = device.makeTexture(descriptor: textureDescriptor)!

// Configure MSAA render pass descriptor

// ...
renderPassDescriptor.colorAttachments[0].texture = msaaTexture
renderPassDescriptor.colorAttachments[0].loadAction = .clear
renderPassDescriptor.colorAttachments[0].storeAction = .multisampleResolve
// Create a multisample texture

// ...
textureDescriptor.textureType = .type2DMultisample
textureDescriptor.sampleCount = 4
textureDescriptor.storageMode = .memoryless

let msaaTexture = device.makeTexture(descriptor: textureDescriptor)!

// Configure MSAA render pass descriptor

// ...
renderPassDescriptor.colorAttachments[0].texture = msaaTexture
renderPassDescriptor.colorAttachments[0].loadAction = .clear
renderPassDescriptor.colorAttachments[0].storeAction = .multisampleResolve
// Create a multisample texture

// ...

textureDescriptor.textureType = .type2DMultisample
textureDescriptor.sampleCount = 4
textureDescriptor.storageMode = .memoryless

let msaaTexture = device.makeTexture(descriptor: textureDescriptor)!

// Configure MSAA render pass descriptor

// ...

renderPassDescriptor.colorAttachments[0].texture = msaaTexture
renderPassDescriptor.colorAttachments[0].loadAction = .clear
renderPassDescriptor.colorAttachments[0].storeAction = .multisampleResolve
~97 MB footprint
~97 MB bandwidth
The diagram on the right is correct.

- **Color0:** Offscreen, sampled
- **Depth:** Depth, sampled
- **Stencil:** Depth, sampled

Pixel Format: BGRA8Unorm
Dimensions: 1125x2436
Allocated Size: 10.97 MiB

Load Action: Clear
Store Action: Multi...solve

~11 MB footprint
~11 MB bandwidth
Best Practices

11. Leverage tile memory

Metal provides access to tile memory
• Programmable blending
• Image blocks
• Tile shaders

Best practice
• Explicitly utilize tile memory
Traditional Deferred Shading

GPU
- Render Pass 1: Geometry
- Render Pass 2: Lighting

System Memory
- Depth
- Normal
- Albedo
- Properties
- Lit
- Scene

Scene Render Pass 2: Lighting
Traditional Deferred Shading

- **GPU**
  - Render Pass 1: Geometry
  - Render Pass 2: Lighting

- **System Memory**
  - Depth
  - Normal
  - Albedo
  - Properties
  - Lit Scene

**Scene**
Traditional Deferred Shading

**GPU**
- Render Pass 1: Geometry
- Render Pass 2: Lighting

**System Memory**
- Depth
- Normal
- Albedo
- Properties
- Lit
- Scene
Single Pass Deferred Shading

- GPU
  - Geometry
  - Lighting

- Tile Memory
  - Depth
  - Normal
  - Albedo
  - Properties

- System Memory
  - Lit Scene
<table>
<thead>
<tr>
<th>Color0: 0x11cddcd20</th>
<th>Color1: 0x11cdd7b90</th>
<th>Color2: 0x11cdd7f30</th>
<th>Color3: 0x11cdd82d0</th>
<th>Color4: 0x11cdd8670</th>
<th>Depth: 0x11cddcc6b0</th>
<th>Stencil: 0x11cdd7670</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGBA16Float</td>
<td>RGBA8Unorm</td>
<td>RGBA8Unorm</td>
<td>RG11...0Float</td>
<td>RGBA16Float</td>
<td>Depth32Float</td>
<td>Stencil8</td>
</tr>
<tr>
<td>Dimensions</td>
<td>Dimensions</td>
<td>Dimensions</td>
<td>Dimensions</td>
<td>Dimensions</td>
<td>Dimensions</td>
<td>Dimensions</td>
</tr>
<tr>
<td>2436×1125</td>
<td>2436×1125</td>
<td>2436×1125</td>
<td>2436×1125</td>
<td>2436×1125</td>
<td>2436×1125</td>
<td>2436×1125</td>
</tr>
<tr>
<td>Allocated Size</td>
<td>Allocated Size</td>
<td>Allocated Size</td>
<td>Allocated Size</td>
<td>Allocated Size</td>
<td>Allocated Size</td>
<td>Allocated Size</td>
</tr>
<tr>
<td>21.66 MiB</td>
<td>Memoryless</td>
<td>Memoryless</td>
<td>Memoryless</td>
<td>Memoryless</td>
<td>Memoryless</td>
<td>Memoryless</td>
</tr>
<tr>
<td>Load Action</td>
<td>Load Action</td>
<td>Load Action</td>
<td>Load Action</td>
<td>Load Action</td>
<td>Load Action</td>
<td>Load Action</td>
</tr>
<tr>
<td>Clear</td>
<td>DontCare</td>
<td>DontCare</td>
<td>DontCare</td>
<td>Clear</td>
<td>Clear</td>
<td>Clear</td>
</tr>
<tr>
<td>Store Action</td>
<td>Store Action</td>
<td>Store Action</td>
<td>Store Action</td>
<td>Store Action</td>
<td>Store Action</td>
<td>Store Action</td>
</tr>
<tr>
<td>Store</td>
<td>DontCare</td>
<td>DontCare</td>
<td>DontCare</td>
<td>Store</td>
<td>Store</td>
<td>DontCare</td>
</tr>
</tbody>
</table>
Demo

Samuel Colbran, GPU Software
General Performance
Memory Bandwidth
Memory Footprint
App Memory Limit

iOS enforces an app memory limit
• Allows more apps to remain in memory
• Allows the system to stay responsive
• Enables fast switching between apps
iOS 12

Upcoming App Store Submission Requirements

March 20, 2019

iOS 12 is now running on more than 80% of devices worldwide. Make sure your app delivers a great user experience by seamlessly integrating with the latest advances in iOS. Starting March 27, 2019, all new apps and app updates for iPhone or iPad, including universal apps, must be built with the iOS 12.1 SDK or later and support iPhone XS Max or the 12.9-inch iPad Pro (3rd generation). Screenshots for these devices will also be required. All new apps and app updates for Apple Watch will need to be built with the watchOS 5.1 SDK or later and support Apple Watch Series 4.

Understanding Changes in Memory Accounting

iOS 12 and tvOS 12 require apps to use memory far more efficiently than before. If you have difficulty reducing your app’s memory requirements, contact us to request an entitlement for your app to use iOS 11-style memory accounting.

Learn more about preparing your apps >
Metal Resources

iOS 12 accounting changes affect
• Metal buffers
• Metal textures

These are used for
• Game assets (meshes, materials, etc.)
• Post process effects (shadows, blur, etc.)
Xcode 11
Metal Memory Viewer

NEW
Metal Memory Viewer
Metal Memory Viewer
## Metal Memory Viewer

### Table

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Allocated Size</th>
<th>Storage Mode</th>
<th>Purgeable State</th>
<th>CPU Access</th>
<th>Time since Last Bound</th>
<th>Pixel Format</th>
<th>Width</th>
<th>Height</th>
<th>Depth</th>
<th>Mipmaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture:0x123865510</td>
<td>Texture</td>
<td>48 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>Never</td>
<td>RGBA8Unorm</td>
<td>200</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123865670</td>
<td>Texture</td>
<td>48 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>Never</td>
<td>RGBA8Unorm</td>
<td>200</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123866c60</td>
<td>Texture</td>
<td>48 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>Never</td>
<td>RGBA8Unorm</td>
<td>210</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123867ac0</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>Never</td>
<td>RGBA8Unorm</td>
<td>92</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x1238678b0</td>
<td>Texture</td>
<td>48 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>Never</td>
<td>RGBA8Unorm</td>
<td>144</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123867c60</td>
<td>Texture</td>
<td>32 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>Never</td>
<td>RGBA8Unorm</td>
<td>128</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123868e80</td>
<td>Texture</td>
<td>13.2 MB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Depth32Float</td>
<td>2688</td>
<td>1242</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Texture:0x1238692a0</td>
<td>Texture</td>
<td>28.3 MB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>RGBA16Float</td>
<td>2688</td>
<td>1242</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x12386a650</td>
<td>Texture</td>
<td>8.6 MB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>RGBA16Float</td>
<td>1344</td>
<td>820</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123864100</td>
<td>Texture</td>
<td>32 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>RG1116Float</td>
<td>128</td>
<td>58</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123869060</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>RG1116Float</td>
<td>64</td>
<td>28</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x1238668f0</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>RG1116Float</td>
<td>32</td>
<td>14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x12386c6f0</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>RG1116Float</td>
<td>32</td>
<td>14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x12386bb00</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>RG1116Float</td>
<td>64</td>
<td>28</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x12386c740</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>173.60 s</td>
<td>R16Float</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x12386a920</td>
<td>Texture</td>
<td>32 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>64.52 s</td>
<td>RGBA8Unorm</td>
<td>210</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123863e60</td>
<td>Texture</td>
<td>144 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>165.71 s</td>
<td>RGBA8Unorm</td>
<td>674</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123863640</td>
<td>Texture</td>
<td>144 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>165.76 s</td>
<td>RGBA8Unorm</td>
<td>730</td>
<td>22</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x1238638a0</td>
<td>Texture</td>
<td>272 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>Write</td>
<td>160.70 s</td>
<td>RGBA8Unorm</td>
<td>864</td>
<td>44</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123867520</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>R16Float</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x12386ba00</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Now</td>
<td>R16Float</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123868f90</td>
<td>Texture</td>
<td>128 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>172.39 s</td>
<td>RGBA8Unorm_sRGB</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x123868940</td>
<td>Texture</td>
<td>176 KB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Never</td>
<td>PVRTC_RGB_4BPP...</td>
<td>812</td>
<td>512</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Texture:0x12386a0e0</td>
<td>Texture</td>
<td>5.4 MB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>Never</td>
<td>RGBA8Unorm</td>
<td>1024</td>
<td>1024</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Texture:0x123869640</td>
<td>Texture</td>
<td>5.4 MB</td>
<td>Shared</td>
<td>NonVolatile</td>
<td>None</td>
<td>173.60 s</td>
<td>RGBA8Unorm</td>
<td>42</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Metal Resource Allocations Instrument

NEW
**Metal Resource Allocations Instrument**

New feature shown at WWDC.

![Metal Resource Allocations Instrument](image)

<table>
<thead>
<tr>
<th>Time</th>
<th>Device</th>
<th>Allocation</th>
<th>Thread</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>00:03:00.000</td>
<td>A1X</td>
<td>Os16SG640x80</td>
<td>Main Thread</td>
<td>Os16SG640x80</td>
</tr>
<tr>
<td>Metal Resource Allocations Instrument</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Creation</th>
<th>Duration</th>
<th>Metal Device</th>
<th>ID</th>
<th>Parent ID</th>
<th>Thread</th>
<th>Resource Type</th>
<th>Resource Size</th>
<th>System Memory Bytes</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:03:491.901</td>
<td>2.78 ms</td>
<td>A1TX</td>
<td>0x16e5e4dc</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:492.350</td>
<td>2.29 ms</td>
<td>A1TX</td>
<td>0x16e5e4ef</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:492.506</td>
<td>2.17 ms</td>
<td>A1TX</td>
<td>0x16e5e0e0</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:492.565</td>
<td>1.97 ms</td>
<td>A1TX</td>
<td>0x16e5e4e1</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:494.475</td>
<td>330.13 ms</td>
<td>A1TX</td>
<td>0x16e5e4ee</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:495.049</td>
<td>2.07 ms</td>
<td>A1TX</td>
<td>0x16e5e8eb</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:495.201</td>
<td>1.89 ms</td>
<td>A1TX</td>
<td>0x16e5e4ec</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:495.670</td>
<td>1.44 ms</td>
<td>A1TX</td>
<td>0x16e5e5ed</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:495.742</td>
<td>1.96 ms</td>
<td>A1TX</td>
<td>0x16e5e4ee</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:496.132</td>
<td>133.71 ms</td>
<td>A1TX</td>
<td>0x16e5e6f6</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:498.397</td>
<td>233.13 ms</td>
<td>A1TX</td>
<td>0x16e5e6f7</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:497.989</td>
<td>2.89 ms</td>
<td>A1TX</td>
<td>0x16e5e6f8</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:498.985</td>
<td>1.84 ms</td>
<td>A1TX</td>
<td>0x16e5e6f9</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:499.388</td>
<td>1.47 ms</td>
<td>A1TX</td>
<td>0x16e5e6fa</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:499.822</td>
<td>1.97 ms</td>
<td>A1TX</td>
<td>0x16e5e6fb</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:500.588</td>
<td>128.39 ms</td>
<td>A1TX</td>
<td>0x16e5e6f3</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:500.372</td>
<td>228.38 ms</td>
<td>A1TX</td>
<td>0x16e5e6f4</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:502.387</td>
<td>1.85 ms</td>
<td>A1TX</td>
<td>0x16e5e6f5</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:502.704</td>
<td>1.50 ms</td>
<td>A1TX</td>
<td>0x16e5e6f6</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:502.829</td>
<td>1.40 ms</td>
<td>A1TX</td>
<td>0x16e5e6f7</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:502.872</td>
<td>1.97 ms</td>
<td>A1TX</td>
<td>0x16e5e6f8</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:503.461</td>
<td>125.95 ms</td>
<td>A1TX</td>
<td>0x16e5e6f9</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:503.500</td>
<td>125.95 ms</td>
<td>A1TX</td>
<td>0x16e5e6fa</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:504.478</td>
<td>1.24 ms</td>
<td>A1TX</td>
<td>0x16e5e6fb</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:504.570</td>
<td>1.18 ms</td>
<td>A1TX</td>
<td>0x16e5e6fc</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:504.610</td>
<td>1.99 ms</td>
<td>A1TX</td>
<td>0x16e5e6fd</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:504.850</td>
<td>124.16 ms</td>
<td>A1TX</td>
<td>0x16e5e6fe</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
<tr>
<td>00:03:505.005</td>
<td>224.82 ms</td>
<td>A1TX</td>
<td>0x16e5e6ff</td>
<td>0x0</td>
<td>Main Thread 0x2ba81</td>
<td>Texture</td>
<td>128.00 KB</td>
<td>MTTexture (128.00 KB, 2D, 512 x 512, PVRTC_RGB8_BPP)</td>
<td></td>
</tr>
</tbody>
</table>
Available Memory

New C-based API to query available memory

Enables games to

- Stream resources more effectively
- Avoid memory spikes

#include <os/proc.h>

size_t os_proc_available_memory(void)
Available Memory

New C-based API to query available memory

Enables games to

• Stream resources more effectively
• Avoid memory spikes

```c
#include <os/proc.h>
size_t os_proc_available_memory(void)
```
On-Device GPU Capture

Programmatically trigger a GPU capture

- Xcode not required
- Ideal for game testers and QA process

To enable, add `MetalCaptureEnabled` to info.plist
if os_proc_available_memory() < 150 * 1024 * 1024 {
    let captureDescriptor = MTLCaptureDescriptor()
    captureDescriptor.captureObject = _device
    captureDescriptor.destination = .gpuTraceDocument
    captureDescriptor.outputURL = _outputURL
    do {
        try _captureManager.startCapture(with: captureDescriptor)
    } catch {
        // ... Handle the error ...
    }
}

// ... Render next frame ...

if _captureManager.isCapturing {
    _captureManager.stopCapture()
    // ... Handle the GPU trace ...
}
if os_proc_available_memory() < 150 * 1024 * 1024 {
    let captureDescriptor = MTLCaptureDescriptor()
    captureDescriptor.captureObject = _device
    captureDescriptor.destination = .gpuTraceDocument
    captureDescriptor.outputURL = _outputURL
    do {
        try _captureManager.startCapture(with: captureDescriptor)
    } catch {
        // ... Handle the error ...
    }
}

// ... Render next frame ...

if _captureManager.isCapturing {
    _captureManager.stopCapture()
    // ... Handle the GPU trace ...
}
if os_proc_available_memory() < 150 * 1024 * 1024 {
    let captureDescriptor = MTLCaptureDescriptor()
    captureDescriptor.captureObject = _device
    captureDescriptor.destination   = .gpuTraceDocument
    captureDescriptor.outputURL     = _outputURL
    do {
        try _captureManager.startCapture(with: captureDescriptor)
    } catch {
        // ... Handle the error ...
    }
}

// ... Render next frame ...

if _captureManager.isCapturing {
    _captureManager.stopCapture()
    // ... Handle the GPU trace ...
}
if os_proc_available_memory() < 150 * 1024 * 1024 {
    let captureDescriptor = MTLCaptureDescriptor()
    captureDescriptor.captureObject = _device
    captureDescriptor.destination   = .gpuTraceDocument
    captureDescriptor.outputURL     = _outputURL
    do {
        try _captureManager.startCapture(with: captureDescriptor)
    } catch {
        // ... Handle the error ...
    }
}

// ... Render next frame ...

if _captureManager.isCapturing {
    _captureManager.stopCapture()
    // ... Handle the GPU trace ...
}
if os_proc_available_memory() < 150 * 1024 * 1024 {
    let captureDescriptor = MTLCaptureDescriptor()
    captureDescriptor.captureObject = _device
    captureDescriptor.destination = .gpuTraceDocument
    captureDescriptor.outputURL = _outputURL
    do {
        try _captureManager.startCapture(with: captureDescriptor)
    } catch {
        // ... Handle the error ...
    }
}

// ... Render next frame ...

if _captureManager.isCapturing {
    _captureManager.stopCapture()
    // ... Handle the GPU trace ...
}

Reducing Memory Footprint
<table>
<thead>
<tr>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose the right resolution</td>
</tr>
<tr>
<td>2. Minimize non-opaque overdraw</td>
</tr>
<tr>
<td>3. Submit GPU work early</td>
</tr>
<tr>
<td>4. Stream resources efficiently</td>
</tr>
<tr>
<td>5. Design for sustained performance</td>
</tr>
<tr>
<td>6. Compress texture assets</td>
</tr>
<tr>
<td>7. Optimize for faster GPU access</td>
</tr>
<tr>
<td>8. Choose the right pixel format</td>
</tr>
<tr>
<td>9. Optimize load and store actions</td>
</tr>
<tr>
<td>10. Optimize multi-sampled extures</td>
</tr>
<tr>
<td>11. Leverage tile memory</td>
</tr>
<tr>
<td>12. Use memoryless render targets</td>
</tr>
<tr>
<td>13. Avoid loading unused assets</td>
</tr>
<tr>
<td>14. Use smaller assets</td>
</tr>
<tr>
<td>15. Simplify memory-intensive effects</td>
</tr>
</tbody>
</table>
Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
6. Compress texture assets
7. Optimize for faster GPU access
8. Choose the right pixel format
9. Optimize load and store actions
10. Optimize multi-sampled textures
11. Leverage tile memory
12. Use memoryless render targets
13. Avoid loading unused assets
14. Use smaller assets
15. Simplify memory-intensive effects
Transient render targets
• Not loaded or stored on system memory
• Don't need a memory allocation

Best practice
• Use **memoryless** storage mode
• Set for all multisampled attachments!

**Best Practices**
12. Use memoryless render targets
// Configure the GBuffer textures as memoryless render targets
// ...
gBufferTextureDescriptor.storageMode = .memoryless
gBufferTextureDescriptor.usage = [.shaderRead, .renderTarget]

// Configure the deferred shading render pass
// ...
for i in 1..<3 {
    gBufferTextureDescriptor.pixelFormat = gBufferPixelFormats[i]
gBufferTextures[i] = device.makeTexture(descriptor: gBufferTextureDescriptor)!

    renderPassDescriptor.colorAttachments[i].texture = gBufferTextures[i]
    renderPassDescriptor.colorAttachments[i].loadAction = .clear
    renderPassDescriptor.colorAttachments[i].storeAction = .dontCare
}
// Configure the GBuffer textures as memoryless render targets

// ...

gBufferTextureDescriptor.storageMode = .memoryless

gBufferTextureDescriptor.usage       = [.shaderRead, .renderTarget ]

// Configure the deferred shading render pass

// ...

for i in 1..<3
{
    gBufferTextureDescriptor.pixelFormat = gBufferPixelFormats[i]
gBufferTextures[i] = device.makeTexture(descriptor: gBufferTextureDescriptor)!

    renderPassDescriptor.colorAttachments[i].texture     = gBufferTextures[i]
    renderPassDescriptor.colorAttachments[i].loadAction  = .clear
    renderPassDescriptor.colorAttachments[i].storeAction = .dontCare
}
// Configure the GBuffer textures as memoryless render targets
// ...
gBufferTextureDescriptor.storageMode = .memoryless
gBufferTextureDescriptor.usage       = [ .shaderRead, .renderTarget ]

// Configure the deferred shading render pass
// ...
for i in 1..<3 {
    gBufferTextureDescriptor.pixelFormat = gBufferPixelFormats[i]
gBufferTextures[i] = device.makeTexture(descriptor: gBufferTextureDescriptor)!

    renderPassDescriptor.colorAttachments[i].texture    = gBufferTextures[i]
    renderPassDescriptor.colorAttachments[i].loadAction  = .clear
    renderPassDescriptor.colorAttachments[i].storeAction = .dontCare
}
<table>
<thead>
<tr>
<th>Color: 0x11cddcc20</th>
<th>Color: 0x11cde7e90</th>
<th>Color: 0x11cdd9630</th>
<th>Color: 0x11cde8670</th>
<th>Color: 0x11cde660</th>
<th>Depth: 0x11cdec65b0</th>
<th>Stencil: 0x11cde7670</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Format RGBA16Float</td>
<td>Pixel Format RGBA8Unorm</td>
<td>Pixel Format RGBA8Unorm</td>
<td>Pixel Format RGBA8Unorm</td>
<td>Pixel Format Depth32Float</td>
<td>Pixel Format Stencil8</td>
<td>Pixel Format Stencil8</td>
</tr>
<tr>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
</tr>
<tr>
<td>Allocated Size 21.66 MiB</td>
<td>Allocated Size Memoryless</td>
<td>Allocated Size Memoryless</td>
<td>Allocated Size Memoryless</td>
<td>Allocated Size 10.97 MiB</td>
<td>Allocated Size Memoryless</td>
<td>Allocated Size Memoryless</td>
</tr>
<tr>
<td>Load Action Clear</td>
<td>Load Action DontCare</td>
<td>Load Action DontCare</td>
<td>Load Action DontCare</td>
<td>Load Action Clear</td>
<td>Load Action Clear</td>
<td>Load Action Clear</td>
</tr>
<tr>
<td>Store Action Store</td>
<td>Store Action DontCare</td>
<td>Store Action DontCare</td>
<td>Store Action DontCare</td>
<td>Store Action Store</td>
<td>Store Action Store</td>
<td>Store Action Store</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Color: 0x10141d120</th>
<th>Color: 0x101420aa0</th>
<th>Color: 0x101420d40</th>
<th>Color: 0x10141d3d0</th>
<th>Depth: 0x101316060</th>
<th>Stencil: 0x1014112d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Format RGBA16Float</td>
<td>Pixel Format RGBA8Unorm</td>
<td>Pixel Format RGBA8Unorm</td>
<td>Pixel Format RGBA8Unorm</td>
<td>Pixel Format Depth32Float</td>
<td>Pixel Format Stencil8</td>
</tr>
<tr>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
<td>Dimensions 2436x1125</td>
</tr>
<tr>
<td>Allocated Size 14.99 MiB</td>
<td>Allocated Size 10.97 MiB</td>
<td>Allocated Size 10.97 MiB</td>
<td>Allocated Size Memoryless</td>
<td>Allocated Size 2.81 MiB</td>
<td>Allocated Size Memoryless</td>
</tr>
<tr>
<td>Load Action Clear</td>
<td>Load Action DontCare</td>
<td>Load Action DontCare</td>
<td>Load Action DontCare</td>
<td>Load Action Clear</td>
<td>Load Action Clear</td>
</tr>
<tr>
<td>Store Action Store</td>
<td>Store Action DontCare</td>
<td>Store Action DontCare</td>
<td>Store Action DontCare</td>
<td>Store Action Store</td>
<td>Store Action Store</td>
</tr>
</tbody>
</table>
Best Practices

13. Avoid loading unused resources

Loading all the assets into memory will increase memory footprint

Best practice

• Consider the memory and performance trade-off
• Load only the assets that will be used
• Free all temporary resources such as splash screen or tutorial UI
Metal Memory Viewer
Metal Memory Viewer

NEW
Metal Memory Viewer
Best Practices

14. Use smaller assets

Make assets only as large as necessary

Best practice

• Consider the image quality and memory trade-off
• Compress textures
• Compress meshes (vertex data)
• Consider loading only the smaller mip map levels
• Consider loading lower fidelity 3D models
Best Practices

15. Simplify memory-intensive effects

Some effects require large offscreen buffers
- Shadow maps, SSAO, and more

Best practice
- Consider the image quality and memory trade-off
- Lower the resolution of offscreen buffers
- Disable memory-intensive effects when memory constrained
## Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
6. Compress texture assets
7. Optimize for faster GPU access
8. Choose the right pixel format
9. Optimize load and store actions
10. Optimize multi-sampled textures
11. Leverage tile memory
12. Use memoryless render targets
13. Avoid loading unused assets
14. Use smaller assets
15. Simplify memory-intensive effects
16. (Advanced) Use Metal resource heaps
17. (Advanced) Mark resources as volatile
18. (Advanced) Manage the Metal PSOs
<table>
<thead>
<tr>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose the right resolution</td>
</tr>
<tr>
<td>2. Minimize non-opaque overdraw</td>
</tr>
<tr>
<td>3. Submit GPU work early</td>
</tr>
<tr>
<td>4. Stream resources efficiently</td>
</tr>
<tr>
<td>5. Design for sustained performance</td>
</tr>
<tr>
<td>6. Compress texture assets</td>
</tr>
<tr>
<td>7. Optimize for faster GPU access</td>
</tr>
<tr>
<td>8. Choose the right pixel format</td>
</tr>
<tr>
<td>9. Optimize load and store actions</td>
</tr>
<tr>
<td>10. Optimize multi-sampled textures</td>
</tr>
<tr>
<td>11. Leverage tile memory</td>
</tr>
<tr>
<td>12. Use memoryless render targets</td>
</tr>
<tr>
<td>13. Avoid loading unused assets</td>
</tr>
<tr>
<td>14. Use smaller assets</td>
</tr>
<tr>
<td>15. Simplify memory-intensive effects</td>
</tr>
<tr>
<td>16. <strong>(Advanced)</strong> Use Metal resource heaps</td>
</tr>
<tr>
<td>17. <strong>(Advanced)</strong> Mark resources as volatile</td>
</tr>
<tr>
<td>18. <strong>(Advanced)</strong> Manage the Metal PSOs</td>
</tr>
</tbody>
</table>
Metal Resource Heaps

- Memory Allocation for A
  - Texture A
- Memory Allocation for B
  - Texture B
- Memory Allocation for C
  - Texture C

Metal Resource Heap
- Texture A
- Texture B
- Texture C
Best Practices

16. Advanced — Use metal resource heaps

Rendering a frame may require a lot of intermediate memory

Resource heaps allow

• Creation of multiple resources from single allocation
• Aliasing resources

Best practice

• Alias intermediate resources that have no dependencies (i.e. SSAO and DoF)
Metal Resource Heaps

Memory Allocation for A
  Texture A

Memory Allocation for B
  Texture B

Memory Allocation for C
  Texture C

Metal Resource Heap
  Texture A  Texture B  Texture C
Metal Resource Heaps

Metal Resource Heap

Texture A
Texture B
Texture C
Metal Resource Heaps

Metal Resource Heap

Texture C
Purgeable Memory

Purgeable memory states
• Non-Volatile — The data should not be discarded
• Volatile — The data can be discarded even when the resource may be needed
• Empty — The data has been discarded

Volatile and Empty allocations don’t count toward the application footprint
Best Practices

17. Advanced — Mark resources as volatile

Temporary resources may become a large part of your game’s footprint.

The purgeable state of Metal resources can be set explicitly.

Best practice

• Explicitly manage the purgeable state of resources
• Mark all cached resources as volatile
// Mark all the textures in the cache as volatile

for i in 0..<cacheSize {
    texturePool[i].setPurgeableState(.volatile)
}

// ...

if (texturePool[expr].setPurgeableState(.nonVolatile) == .empty) {
    // ... Regenerate texture data ...
}

// ... Use texture ...
// Mark all the textures in the cache as volatile

```swift
for i in 0..<cacheSize {
    texturePool[i].setPurgeableState(.volatile)
}
```

// ...

```swift
if (texturePool[idx].setPurgeableState(.nonVolatile) == .empty) {
    // ... Regenerate texture data ...
}
```

// ... Use texture ...
// Mark all the textures in the cache as volatile

for i in 0..<cacheSize {
    texturePool[i].setPurgeableState(.volatile)
}

// ...

if (texturePool[idx].setPurgeableState(.nonVolatile) == .empty) {
    // ... Regenerate texture data ...
}

// ... Use texture ...
// Mark all the textures in the cache as volatile

for i in 0..<cacheSize {
    texturePool[i].setPurgeableState(.volatile)
}

// ...

if (texturePool[idx].setPurgeableState(.nonVolatile) == .empty) {
    // ... Regenerate texture data ...
}

// ... Use texture ...
// Mark all the textures in the cache as volatile

for i in 0..<cacheSize {
    texturePool[i].setPurgeableState(.volatile)
}

// ...

if (texturePool[idx].setPurgeableState(.nonVolatile) == .empty) {
    // ... Regenerate texture data ...
}

// ... Use texture ...
Pipeline State Objects (PSOs)
Pipeline State Objects (PSOs)

Metal Library

→

Vertex Function

→

Fragment Function

Pipeline State Descriptor
Pipeline State Objects (PSOs)

Metal Library

- Vertex Function
- Fragment Function
- Blend State Descriptor
- Vertex Layout Descriptor
- Color Attachments Descriptor
- Depth Attachment Descriptor
- Stencil Attachment Descriptor
- Pipeline State Descriptor
Pipeline State Objects (PSOs)

- Metal Library
- Pipeline State Descriptor
  - Vertex Function
  - Fragment Function
  - Blend State Descriptor
  - Vertex Layout Descriptor
  - Color Attachments Descriptor
  - Depth Attachment Descriptor
  - Stencil Attachment Descriptor
  - Pipeline State Object
Best Practices
18. Advanced — Manage the Metal PSOs

Metal allows the application to load most of the rendering state (PSOs) up front

Best practice
• Consider the performance and memory trade-off
• Don't hold PSO references unnecessarily
• Don't hold Metal function references after PSO creation
Pipeline State Objects (PSOs)

- Vertex Function
- Fragment Function
- Blend State Descriptor
- Vertex Layout Descriptor
- Color Attachments Descriptor
- Depth Attachment Descriptor
- Stencil Attachment Descriptor
- Pipeline State Descriptor

Pipeline State Object
Pipeline State Objects (PSOs)

- Vertex Function
- Fragment Function
- Blend State Descriptor
- Vertex Layout Descriptor
- Color Attachments Descriptor
- Depth Attachment Descriptor
- Stencil Attachment Descriptor
- Pipeline State Descriptor

Free reference after PSO creation
Free reference after no longer needed
Demo

Ubaka Onyechi, GPU Software
Best Practices

1. Choose the right resolution
2. Minimize non-opaque overdraw
3. Submit GPU work early
4. Stream resources efficiently
5. Design for sustained performance
6. Compress texture assets
7. Optimize for faster GPU access
8. Choose the right pixel format
9. Optimize load and store actions
10. Optimize multi-sampled textures
11. Leverage tile memory
12. Use memoryless render targets
13. Avoid loading unused assets
14. Use smaller assets
15. Simplify memory-intensive effects
16. (Advanced) Use Metal resource heaps
17. (Advanced) Mark resources as volatile
18. (Advanced) Manage the Metal PSOs
Best Practices

- Choose the right resolution
- Minimize non-opaque overdraw
- Submit GPU work early
- Stream resources efficiently
- Design for sustained performance
- Compress texture assets
- Optimize for faster GPU access
- Choose the right pixel format
- Optimize load and store actions
- Optimize multi-sampled textures
- Leverage tile memory
- Use memoryless render targets
- Avoid loading unused assets
- Use smaller assets
- Simplify memory-intensive effects
- (Advanced) Use Metal resource heaps
- (Advanced) Mark resources as volatile
- (Advanced) Manage the Metal PSOs
Best Practices

- Choose the right resolution
- Minimize non-opaque overdraw
- Submit GPU work early
- Stream resources efficiently
- Design for sustained performance
- Compress texture assets
- Optimize for faster GPU access
- Choose the right pixel format
- Optimize load and store actions
- Optimize multi-sampled textures
- Leverage tile memory
- Use memoryless render targets
- Avoid loading unused assets
- Use smaller assets
- Simplify memory-intensive effects
- (Advanced) Use Metal resource heaps
- (Advanced) Mark resources as volatile
- (Advanced) Manage the Metal PSOs
Best Practices

- Choose the right resolution
- Minimize non-opaque overdraw
- Submit GPU work early
- Stream resources efficiently
- Design for sustained performance
- Compress texture assets
- Optimize for faster GPU access
- Choose the right pixel format
- Optimize load and store actions

- Optimize multi-sampled textures
- Leverage tile memory
- Use memoryless render targets
- Avoid loading unused assets
- Use smaller assets
- Simplify memory-intensive effects
- (Advanced) Use Metal resource heaps
- (Advanced) Mark resources as volatile
- (Advanced) Manage the Metal PSOs
More Information

developer.apple.com/wwdc19/606