#WWDC19

Delivering Optimized Metal Apps and Games

Guillem Viñals Gangolells — 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

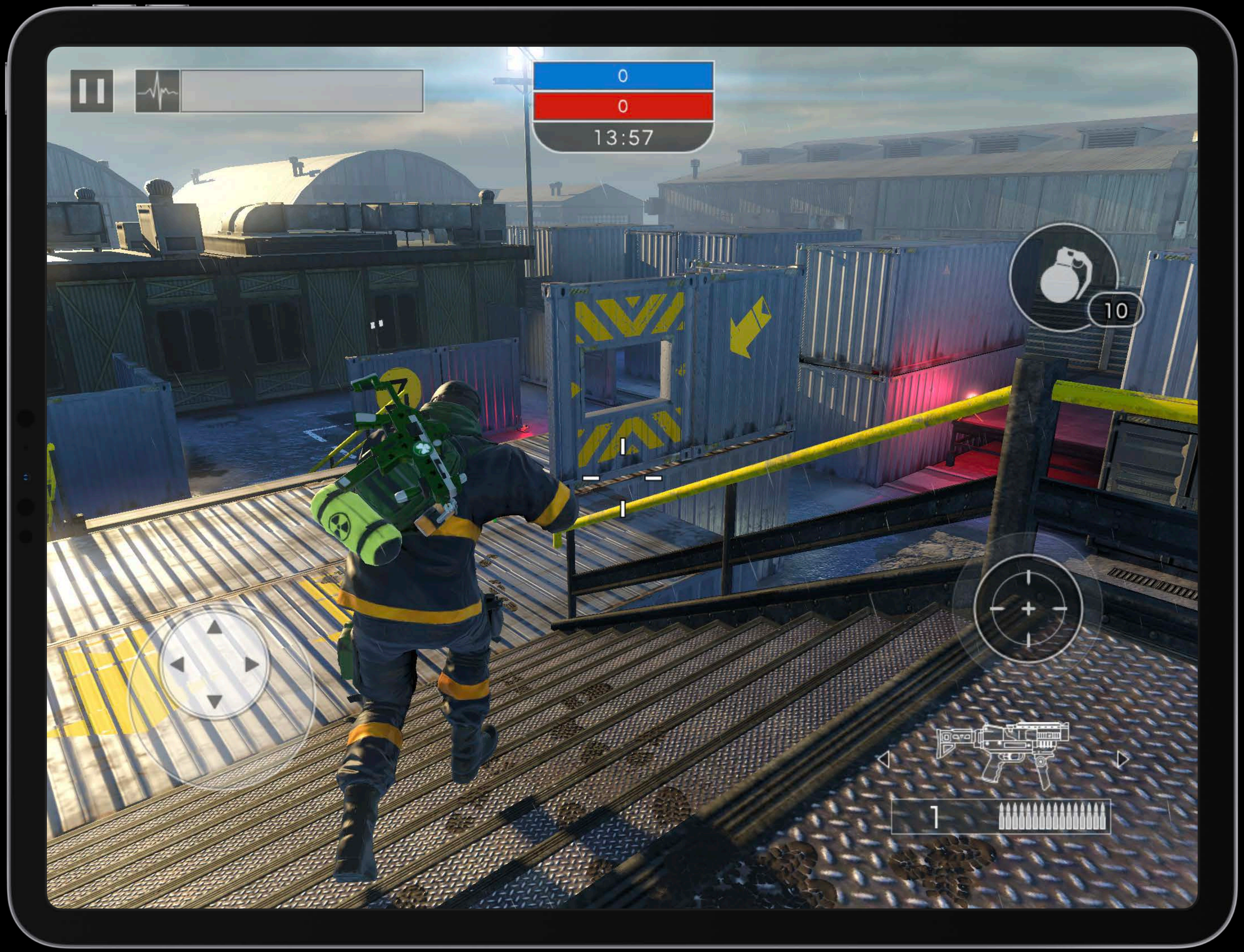
Cascaded shadow maps

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

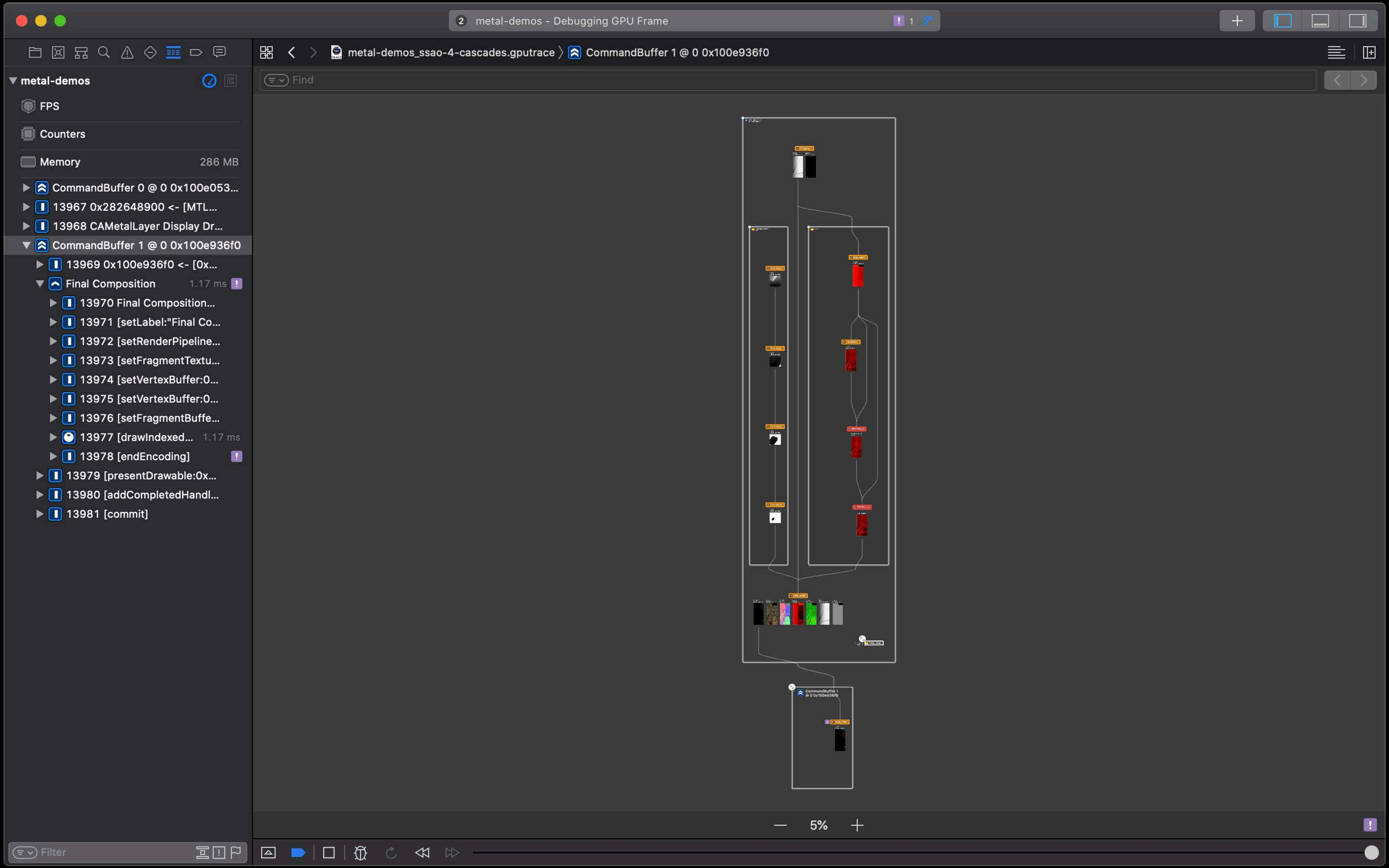
Best Practices

1. Choose the right resolution

Each effect may require a different resolution

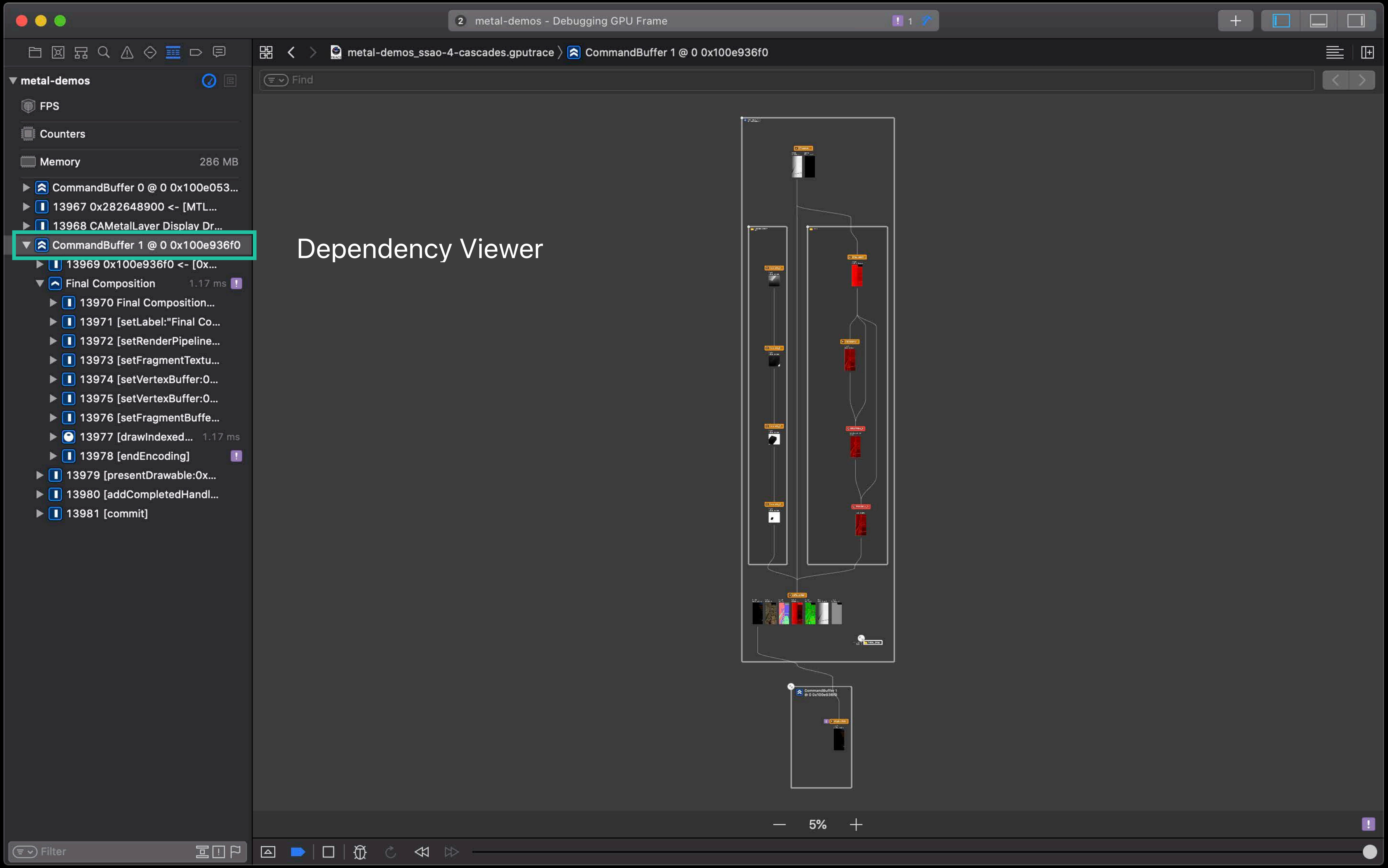
Best practice

- Consider the image quality and performance trade-off
- Composite the game UI at native resolution

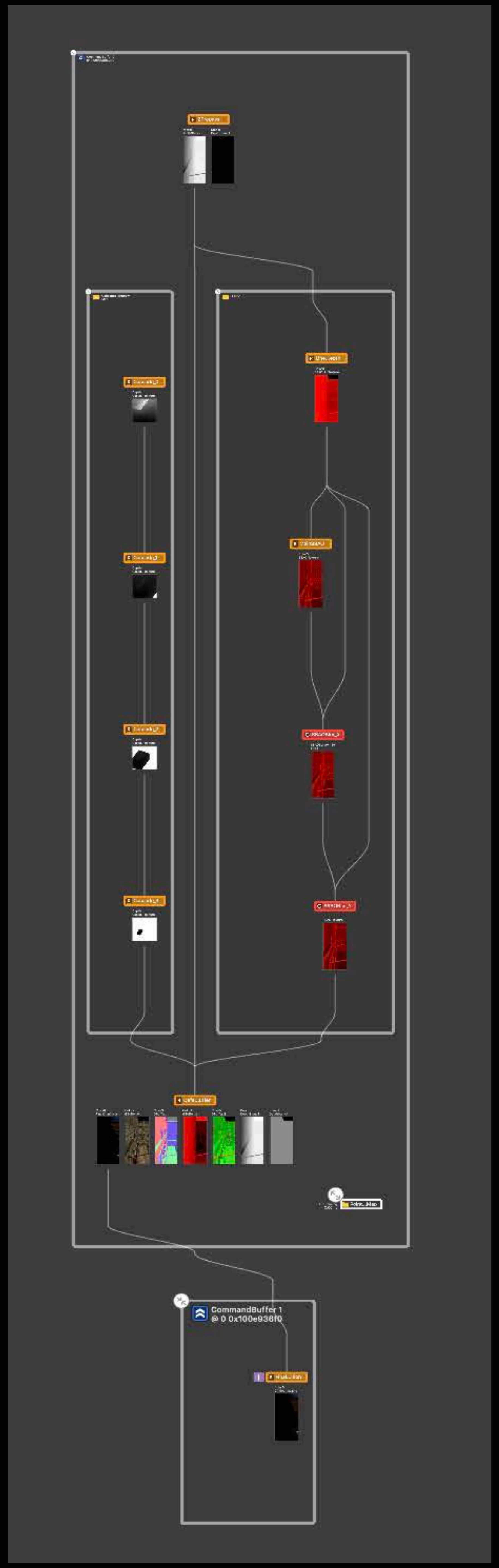


Sponza Atrium © 2009 Frank Meinl, Crytek, https://casual-effects.com/data/ Rendered with Metal, https://creativecommons.org/licenses/by/3.0/

Dependency Viewer



Sponza Atrium © 2009 Frank Meinl, Crytek, https://casual-effects.com/data/ Rendered with Metal, https://creativecommons.org/licenses/by/3.0/



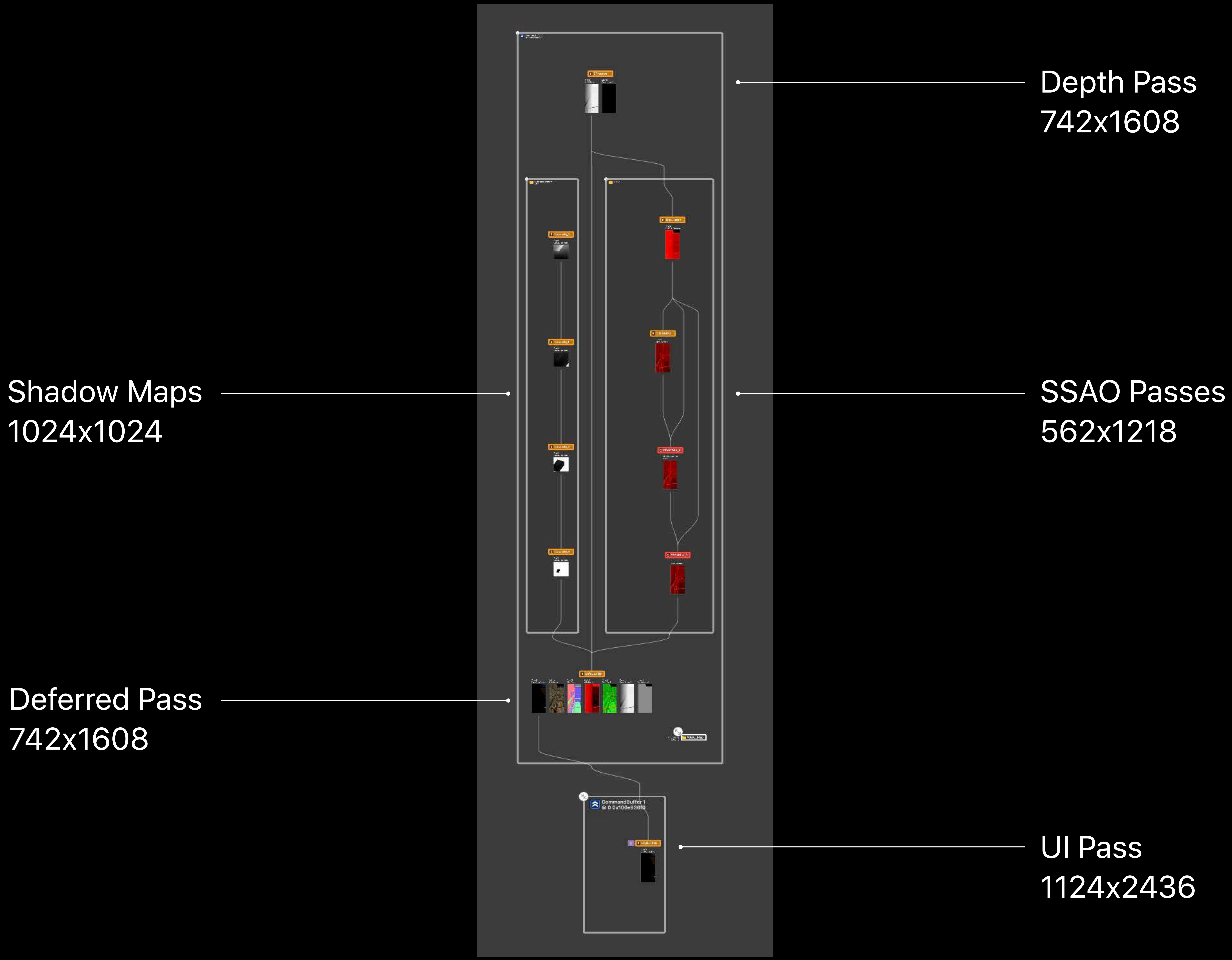
Depth Pass
742x1608

Shadow Maps
1024x1024

SSAO Passes
562x1218

Deferred Pass
742x1608

UI Pass
1124x2436


Depth Pass
742x1608
Shadow Maps
1024x1024
SSAO Passes
562x1218
Deferred Pass
742x1608
UI Pass
1124x2436

Best Practices

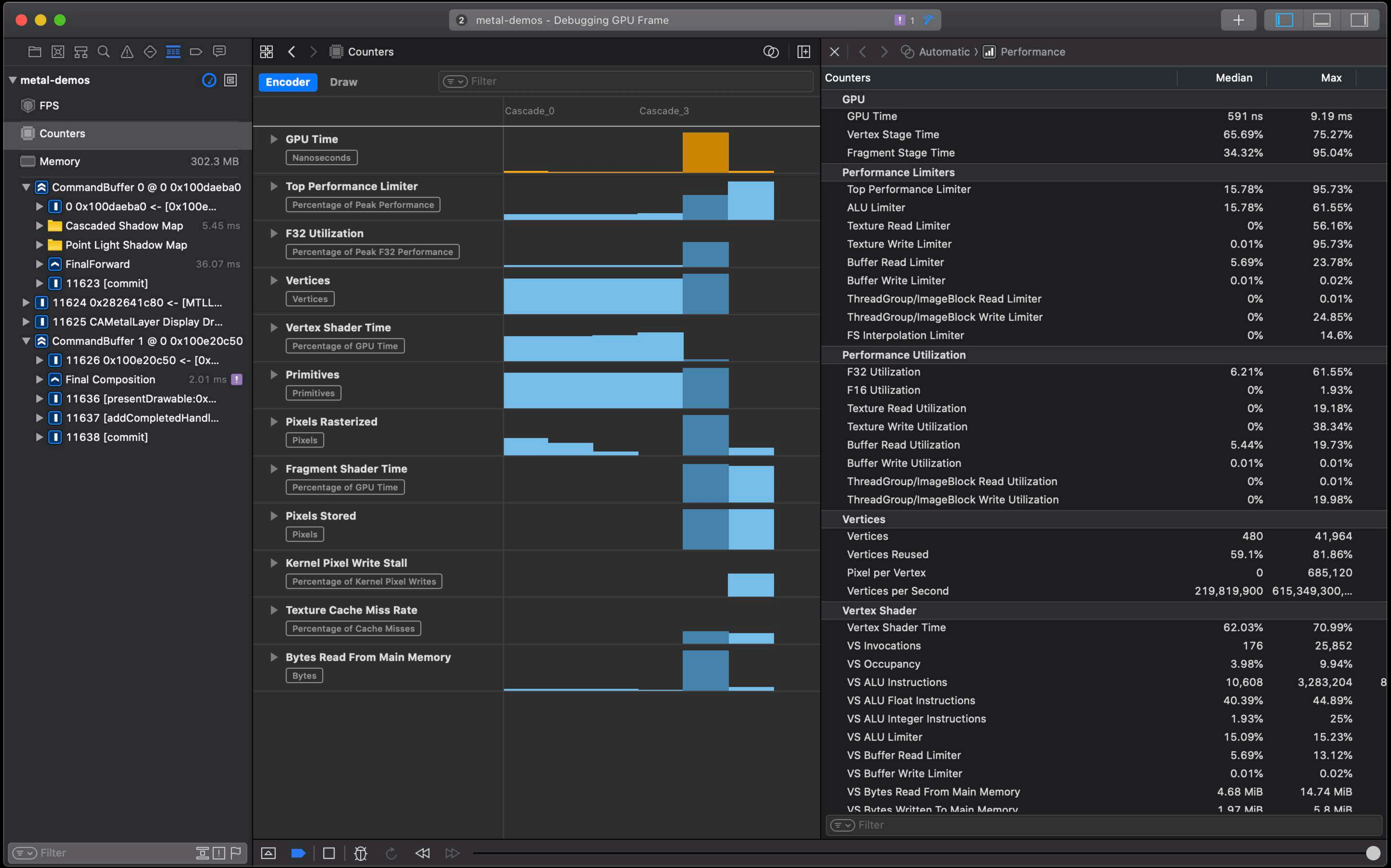
2. Minimize non-opaque overdraw

Processing multiple fragments per pixel causes overdraw

iOS GPUs are very efficient at reducing opaque overdraw

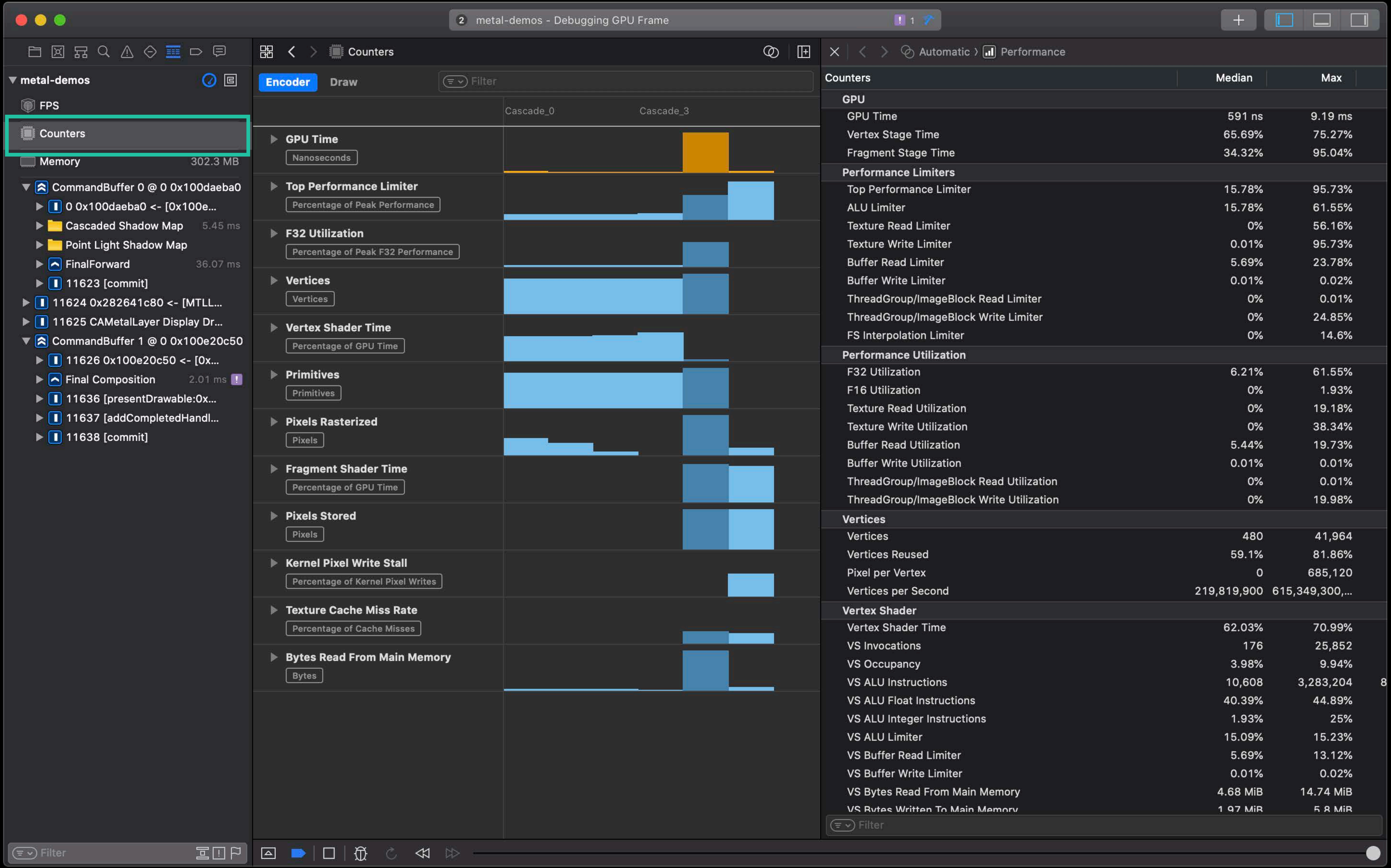
Best practice

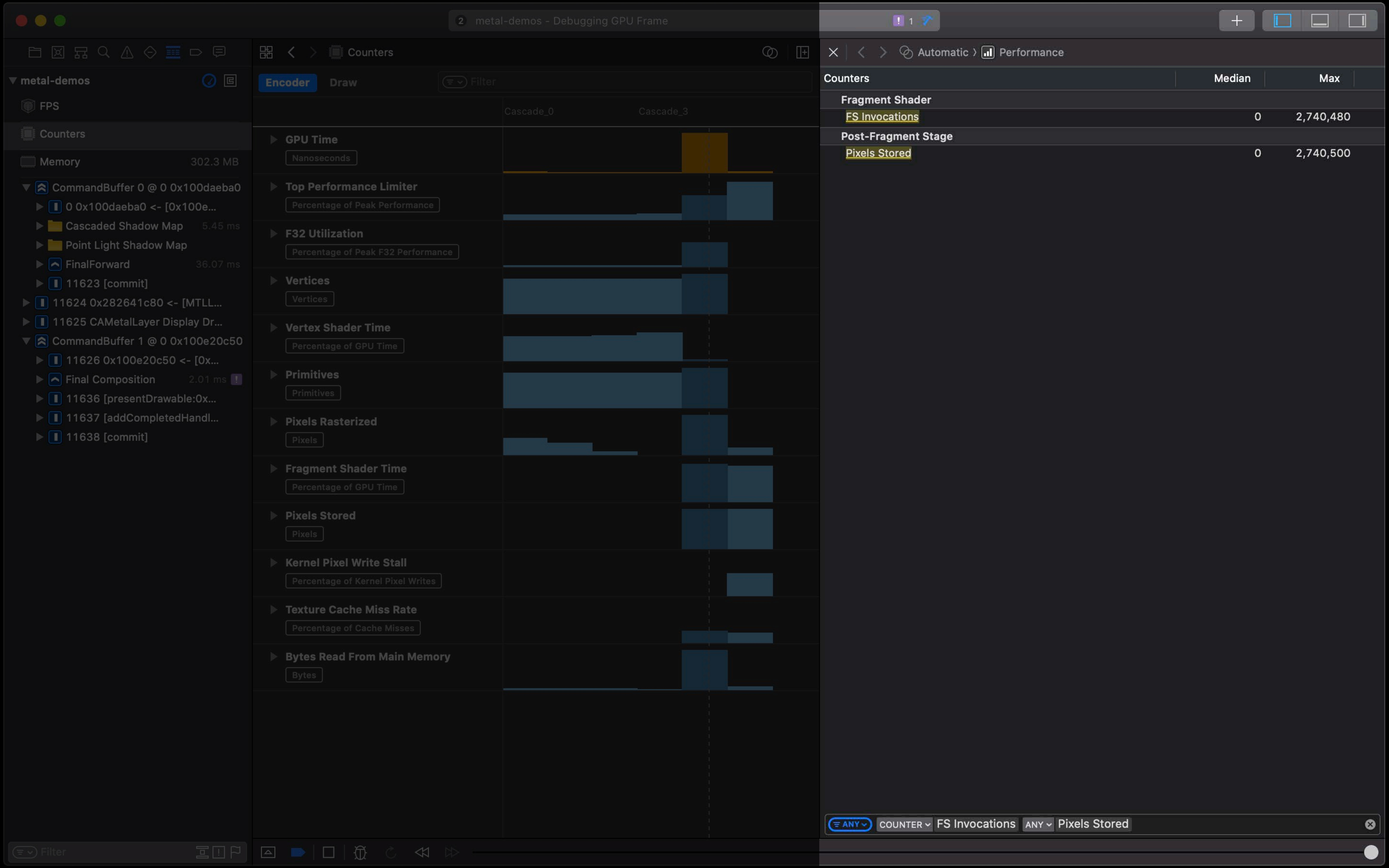
- Render opaque meshes first and translucent meshes later
- Don't render invisible (fully transparent) meshes

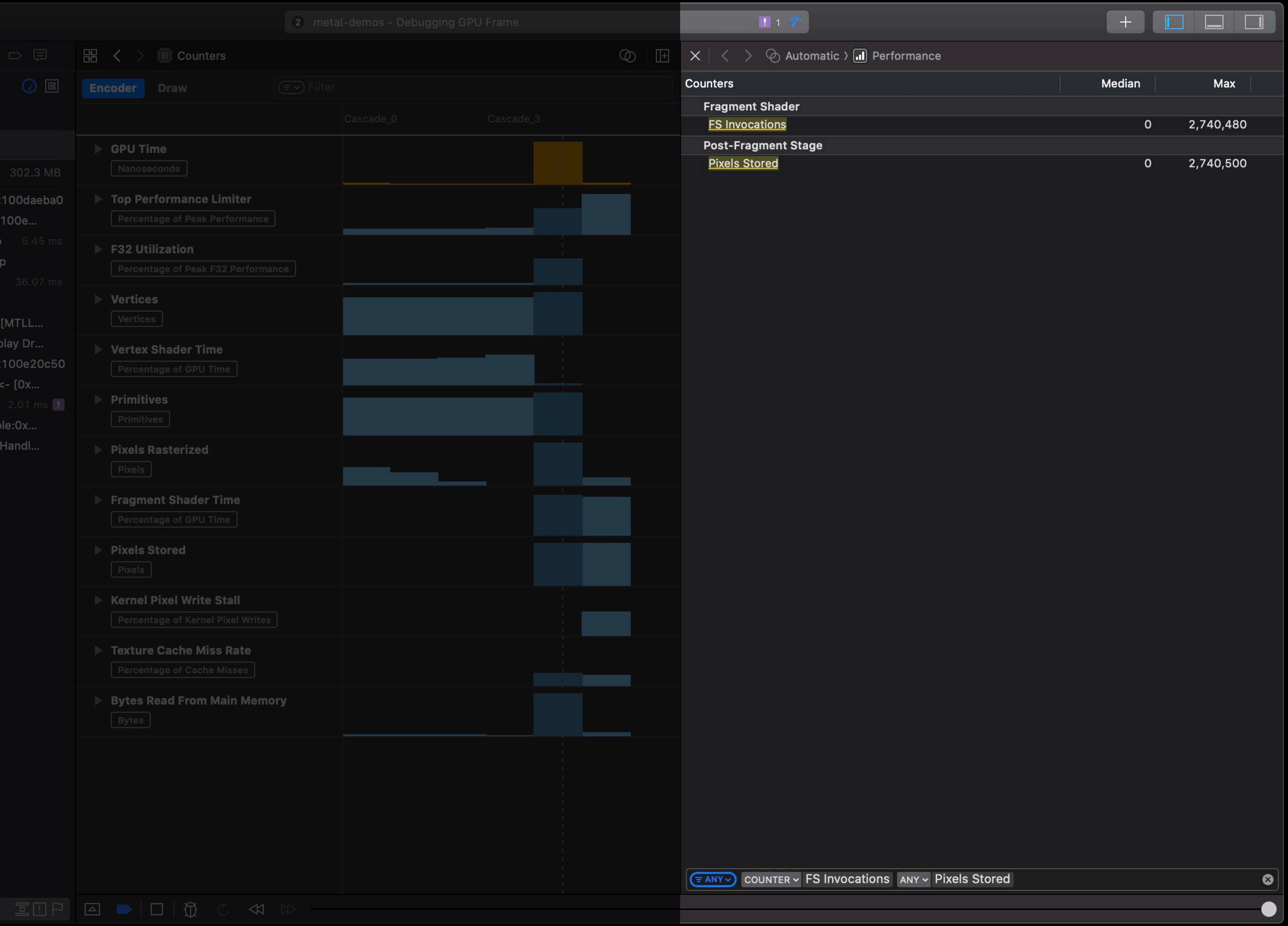


metal-demos - Debugging GPU Frame
metal-demos
FPS
Counters
Memory 302.3 MB
CommandBuffer 0 @ 0 0x100daeba0
0 0x100daeba0 <- [0x100e...
Cascaded Shadow Map 5.45 ms
Point Light Shadow Map
FinalForward 36.07 ms
11623 [commit]
11624 0x282641c80 <- [MTLL...
11625 CAMetalLayer Display Dr...
CommandBuffer 1 @ 0 0x100e20c50
11626 0x100e20c50 <- [0x...
Final Composition 2.01 ms
11636 [presentDrawable:0x...
11637 [addCompletedHandl...
11638 [commit]
Filter
Counters
Encoder
Draw
Filter
Cascade_0
Cascade_3
GPU Time
Nanoseconds
Top Performance Limiter
Percentage of Peak Performance
F32 Utilization
Percentage of Peak F32 Performance
Vertices
Vertices
Vertex Shader Time
Percentage of GPU Time
Primitives
Primitives
Pixels Rasterized
Pixels
Fragment Shader Time
Percentage of GPU Time
Pixels Stored
Pixels
Kernel Pixel Write Stall
Percentage of Kernel Pixel Writes
Texture Cache Miss Rate
Percentage of Cache Misses
Bytes Read From Main Memory
Bytes
Automatic
Performance
Counters Median Max
GPU
GPU Time 591 ns 9.19 ms
Vertex Stage Time 65.69% 75.27%
Fragment Stage Time 34.32% 95.04%
Performance Limiters
Top Performance Limiter 15.78% 95.73%
ALU Limiter 15.78% 61.55%
Texture Read Limiter 0% 56.16%
Texture Write Limiter 0.01% 95.73%
Buffer Read Limiter 5.69% 23.78%
Buffer Write Limiter 0.01% 0.02%
ThreadGroup/ImageBlock Read Limiter 0% 0.01%
ThreadGroup/ImageBlock Write Limiter 0% 24.85%
FS Interpolation Limiter 0% 14.6%
Performance Utilization
F32 Utilization 6.21% 61.55%
F16 Utilization 0% 1.93%
Texture Read Utilization 0% 19.18%
Texture Write Utilization 0% 38.34%
Buffer Read Utilization 5.44% 19.73%
Buffer Write Utilization 0.01% 0.01%
ThreadGroup/ImageBlock Read Utilization 0% 0.01%
ThreadGroup/ImageBlock Write Utilization 0% 19.98%
Vertices
Vertices 480 41,964
Vertices Reused 59.1% 81.86%
Pixel per Vertex 0 685,120
Vertices per Second 219,819,900 615,349,300,...
Vertex Shader
Vertex Shader Time 62.03% 70.99%
VS Invocations 176 25,852
VS Occupancy 3.98% 9.94%
VS ALU Instructions 10,608 3,283,204
VS ALU Float Instructions 40.39% 44.89%
VS ALU Integer Instructions 1.93% 25%
VS ALU Limiter 15.09% 15.23%
VS Buffer Read Limiter 5.69% 13.12%
VS Buffer Write Limiter 0.01% 0.02%
VS Bytes Read From Main Memory 4.68 MiB 14.74 MiB
Filter

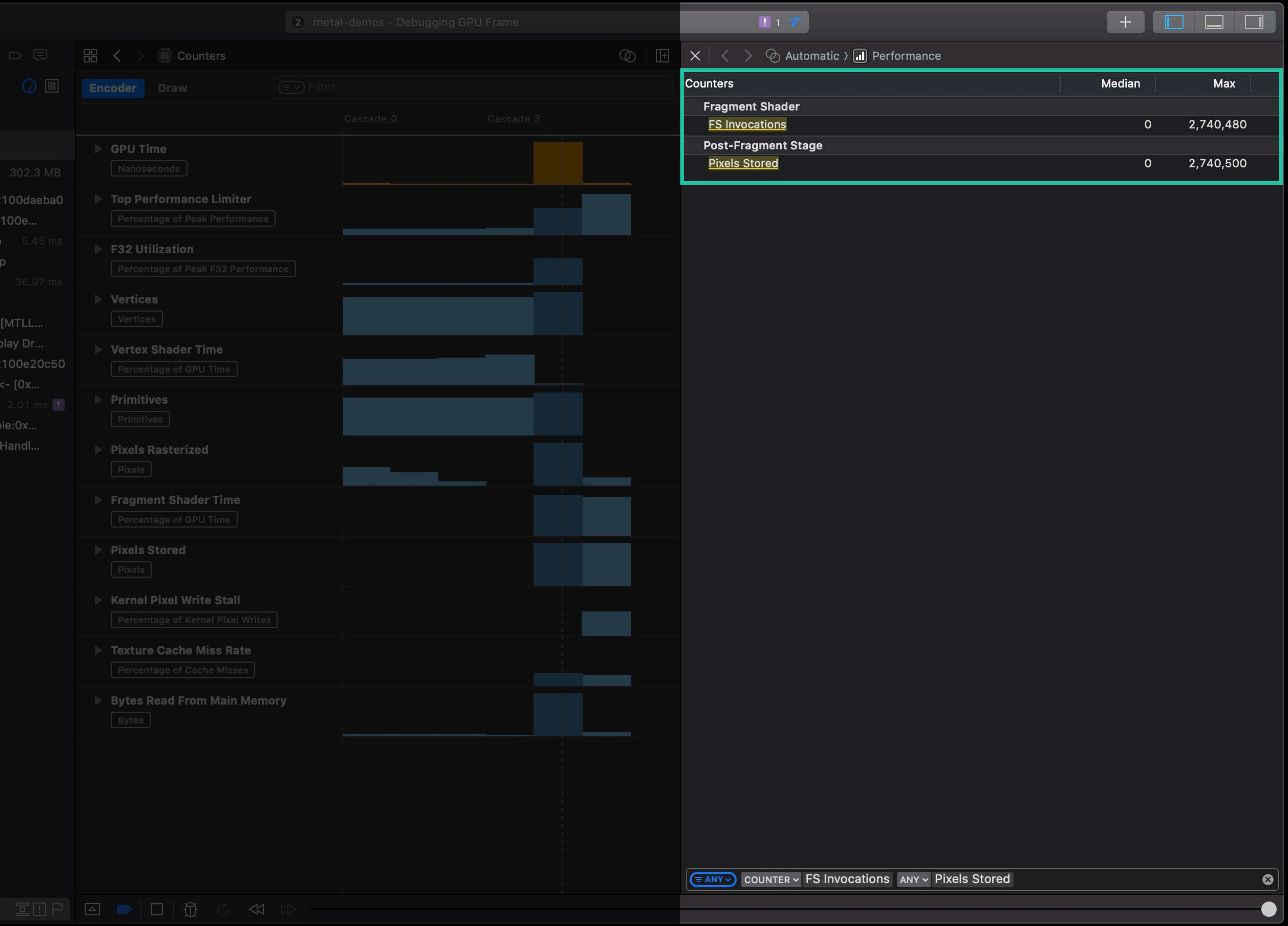
GPU Counters




metal-demos - Debugging GPU Frame
Counters
Automatic
Performance
metal-demos
FPS
Counters
Memory 302.3 MB
CommandBuffer 0 @ 0 0x100daeba0
0 0x100daeba0 <- [0x100e...
Cascaded Shadow Map 5.45 ms
Point Light Shadow Map
FinalForward 36.07 ms
11623 [commit]
11624 0x282641c80 <- [MTLL...
11625 CAMetalLayer Display Dr...
CommandBuffer 1 @ 0 0x100e20c50
11626 0x100e20c50 <- [0x...
Final Composition 2.01 ms
11636 [presentDrawable:0x...
11637 [addCompletedHandl...
11638 [commit]
Encoder
Draw
Filter
Cascade_0
Cascade_3
GPU Time
Nanoseconds
Top Performance Limiter
Percentage of Peak Performance
F32 Utilization
Percentage of Peak F32 Performance
Vertices
Vertices
Vertex Shader Time
Percentage of GPU Time
Primitives
Primitives
Pixels Rasterized
Pixels
Fragment Shader Time
Percentage of GPU Time
Pixels Stored
Pixels
Kernel Pixel Write Stall
Percentage of Kernel Pixel Writes
Texture Cache Miss Rate
Percentage of Cache Misses
Bytes Read From Main Memory
Bytes
Counters
Median
Max
Fragment Shader
FS Invocations 0 2,740,480
Post-Fragment Stage
Pixels Stored 0 2,740,500
ANY
COUNTER
FS Invocations
ANY
Pixels Stored
Filter



Overdraw = 2740480 / 2740500
= ~1.00



$$\text{Overdraw} = 2740480 / 2740500$$
$$= \sim 1.00$$

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

Demo

```swift
// 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()
```

```swift
// 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()
```

```swift
// 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

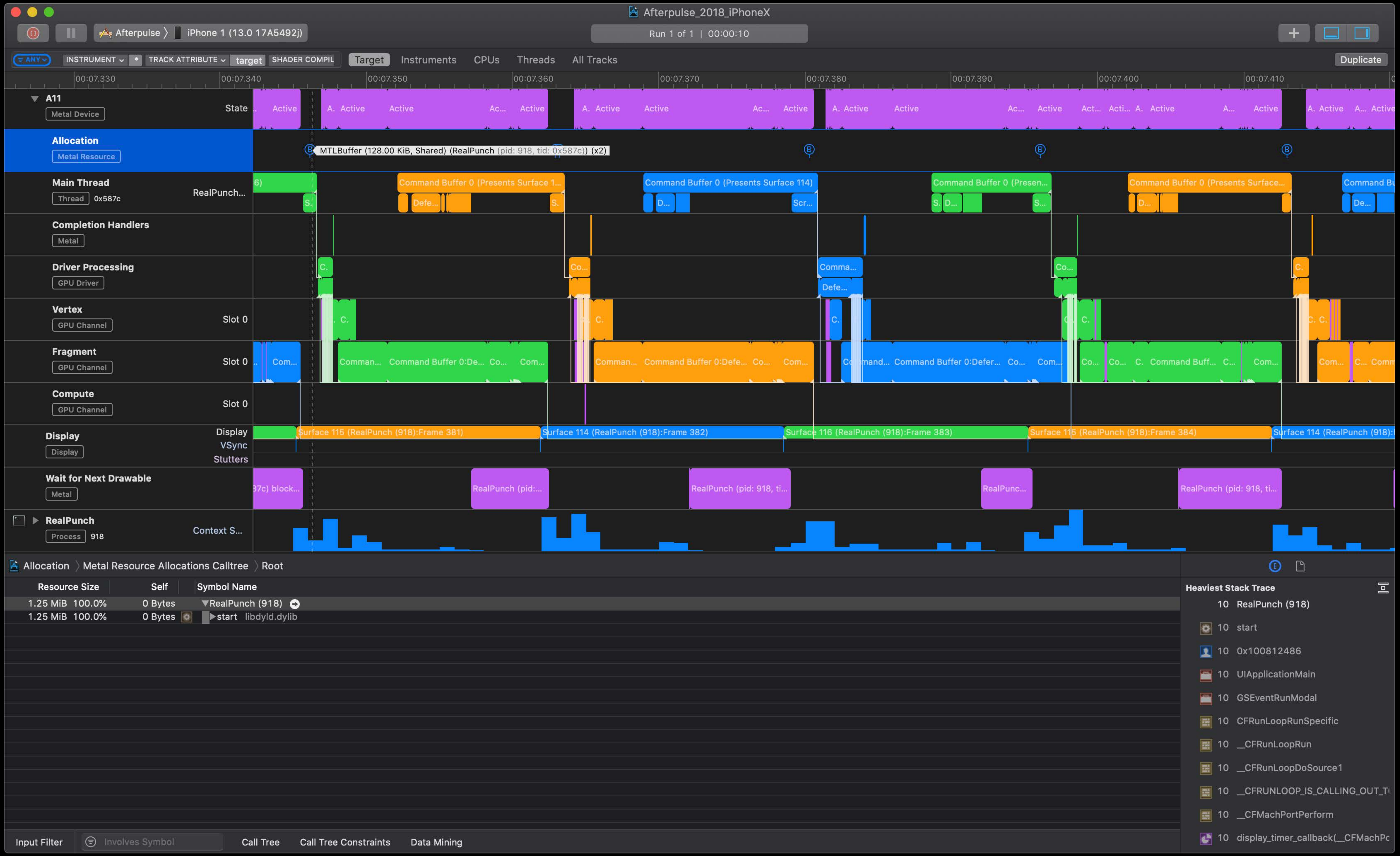
Streaming resources from the render thread may cause stalls

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

NEW



Metal System Trace

NEW

Best Practices

5. Design for sustained performance

Designing your game for sustained performance

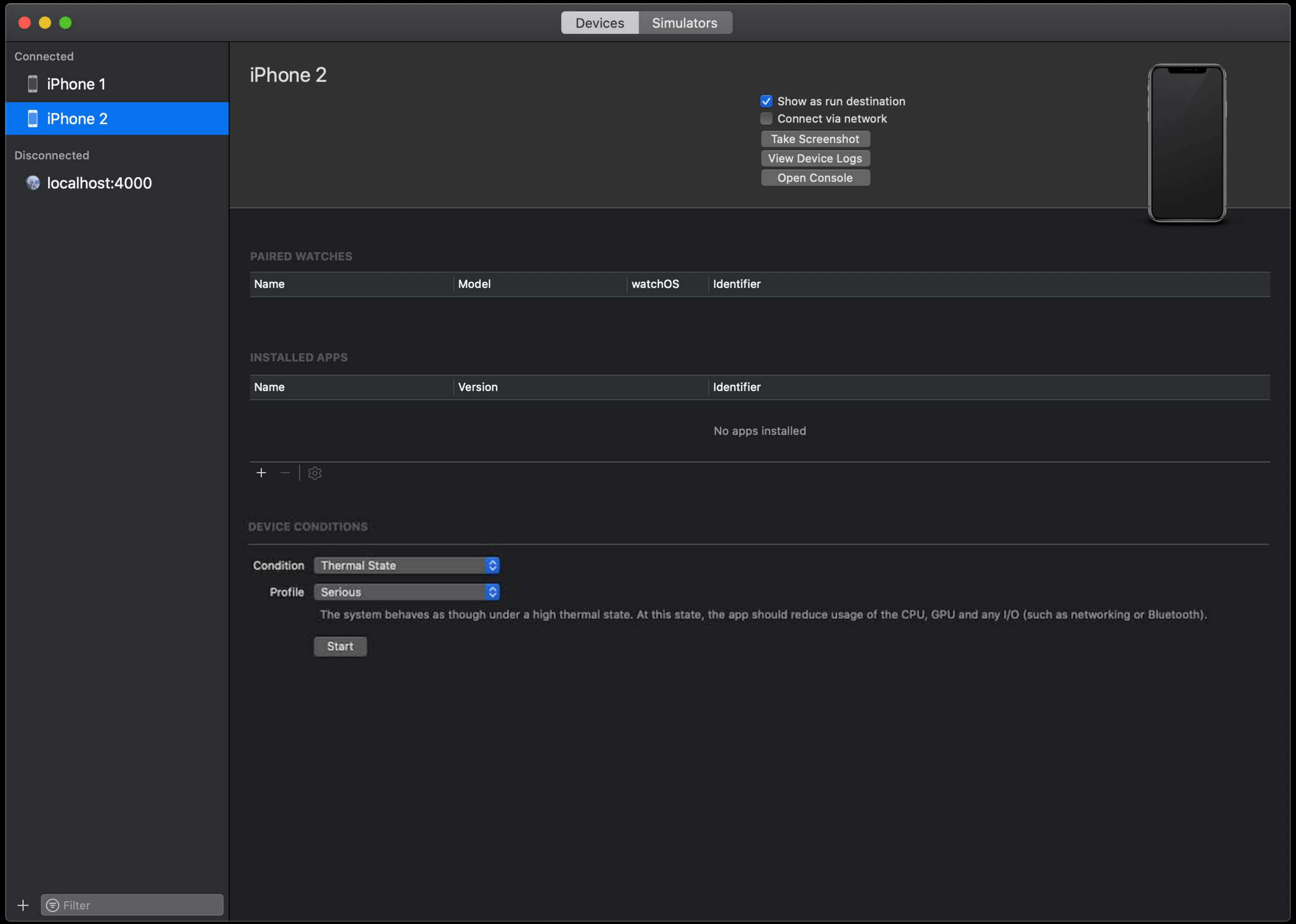
- Improves thermals
- Improves stability and responsiveness

Best practice

- Test your game under `serious` thermal state
- Consider tuning your game for `serious` thermal state

Device Conditions

NEW

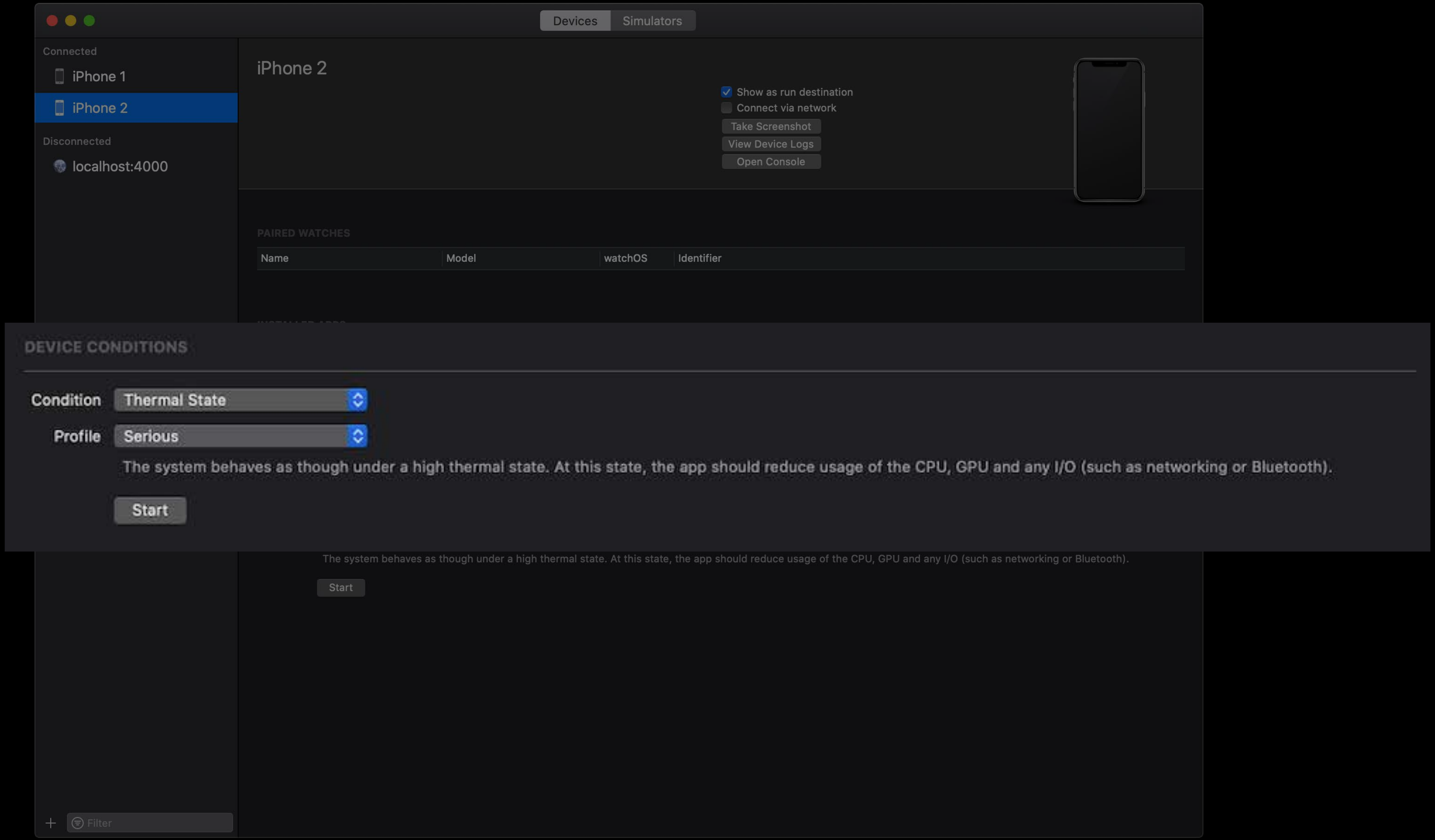


Designing for Adverse Network and Temperature Conditions

Friday, 4:20

Device Conditions

NEW



Designing for Adverse Network and Temperature Conditions

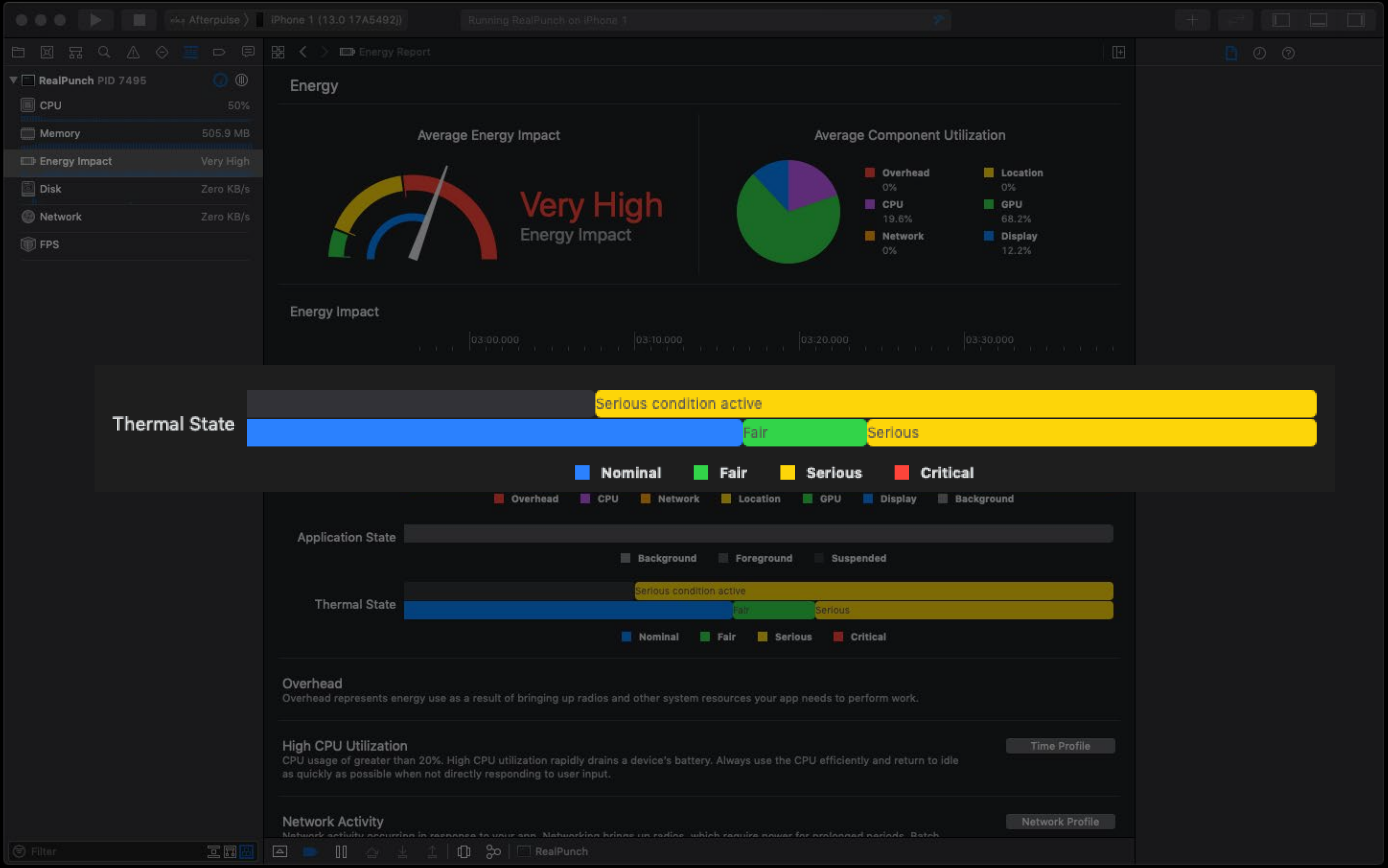
Friday, 4:20

Xcode Energy Gauge

NEW

Xcode Energy Gauge

NEW



General Performance

Memory Bandwidth

Memory Footprint

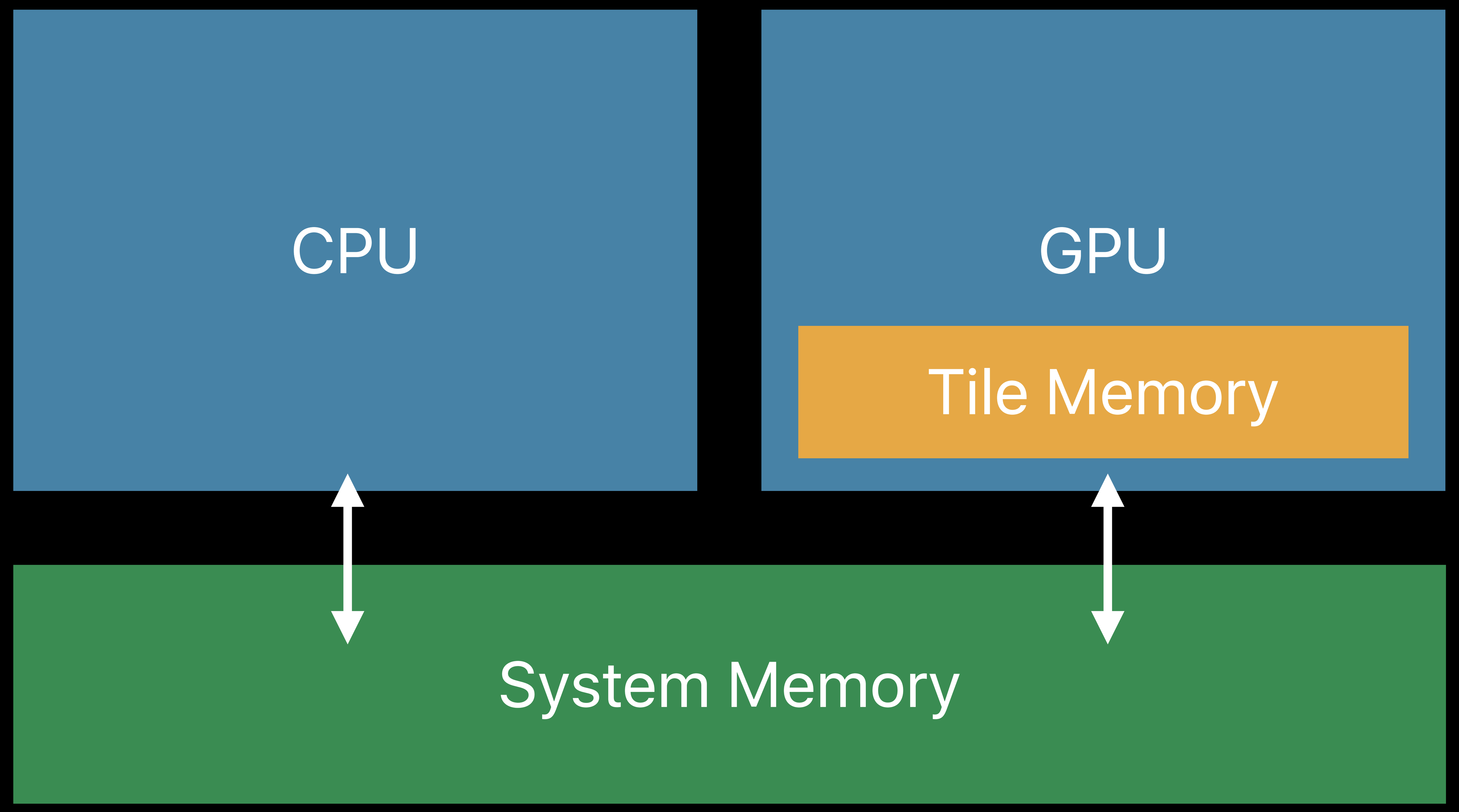
Why Bandwidth?

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

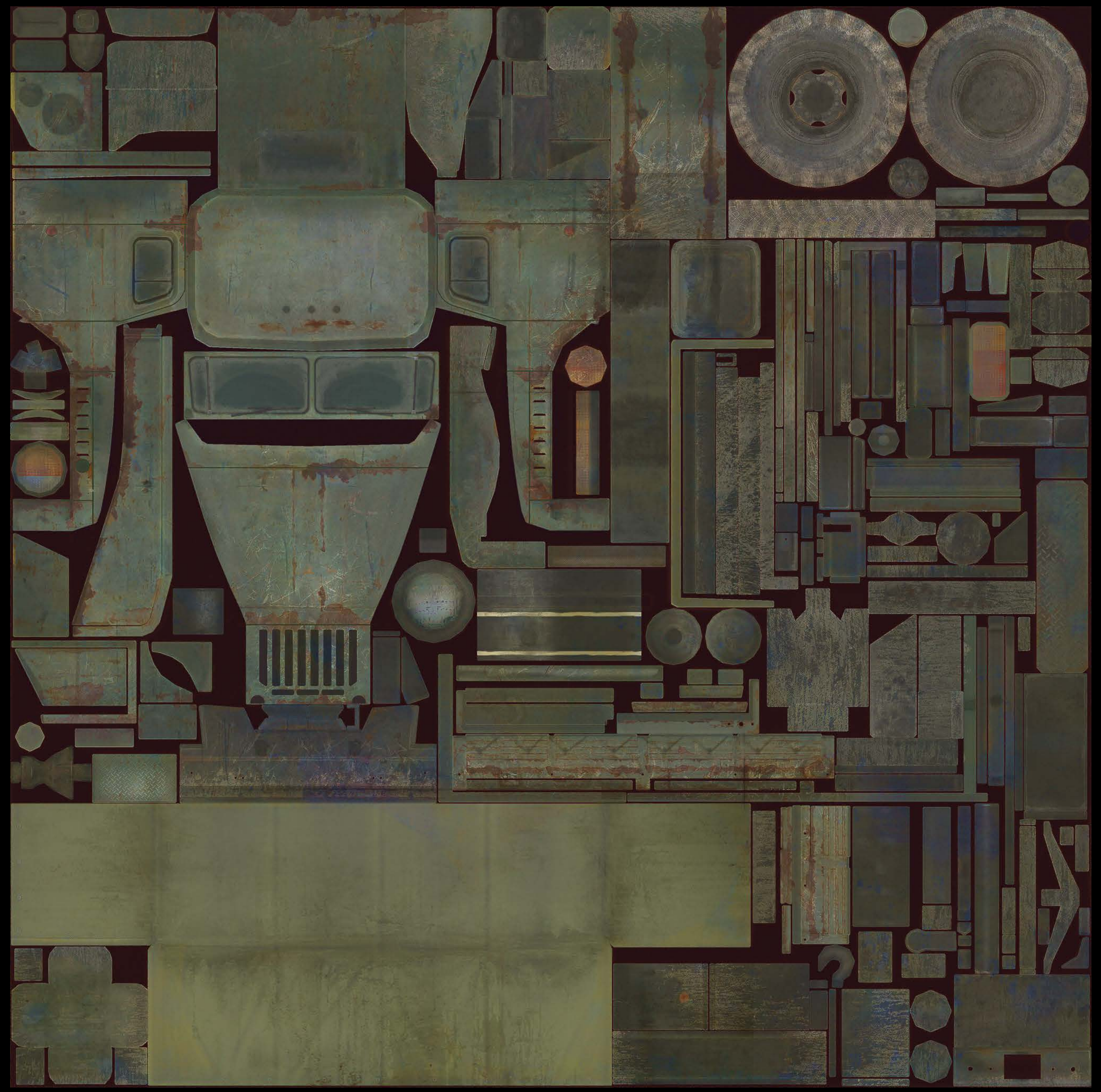
6. Compress texture assets

Sampling large textures may be inefficient

Best practice

- Compress all texture assets — ASTC, PVRTC, and more
- Generate mipmaps for textures that may be minified

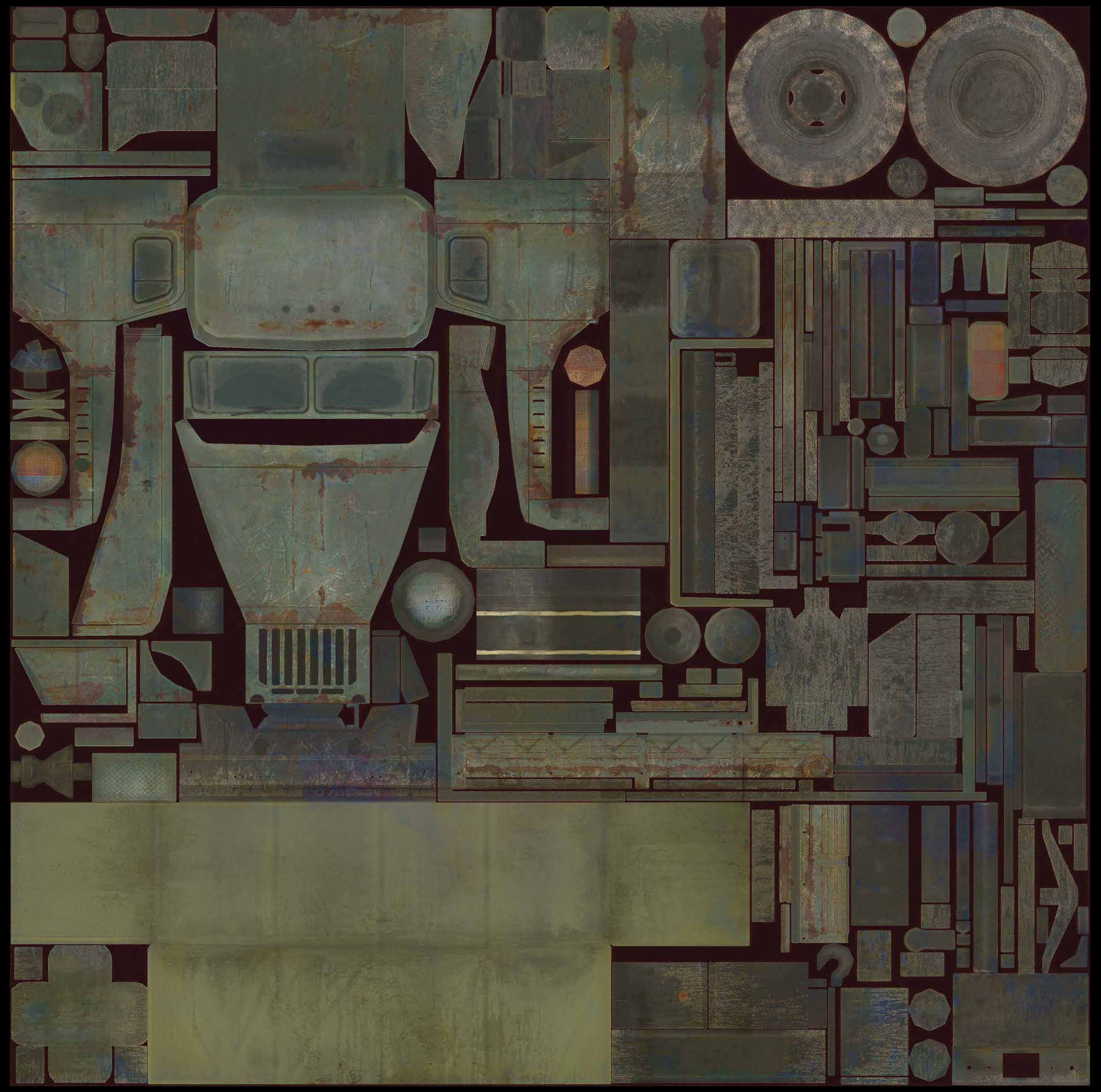
Memory Savings of Texture Compression



2048x2048
RGBA8Unorm

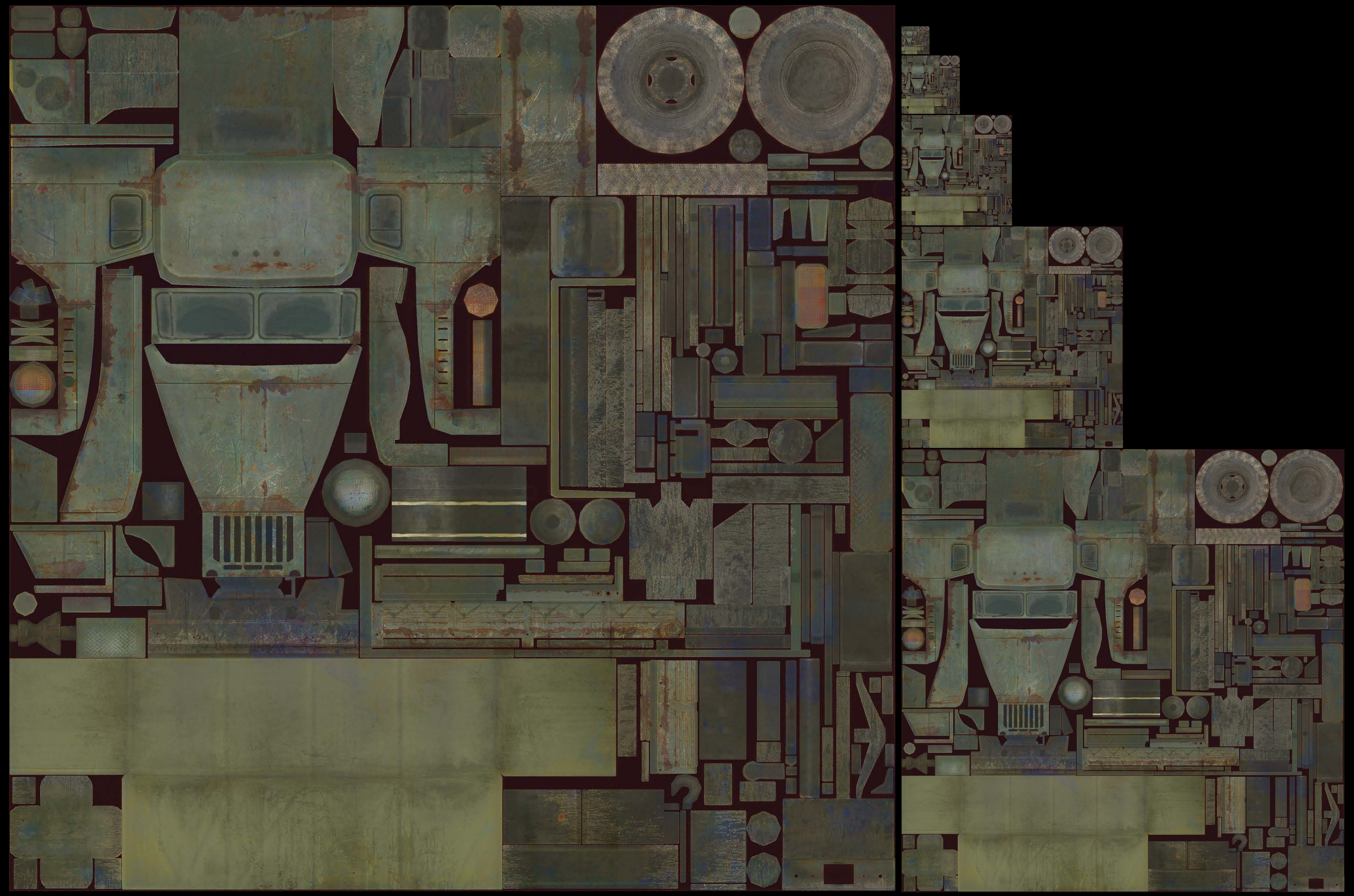
16MB

Memory Savings of Texture Compression



2048x2048
RGBA8Unorm

16MB

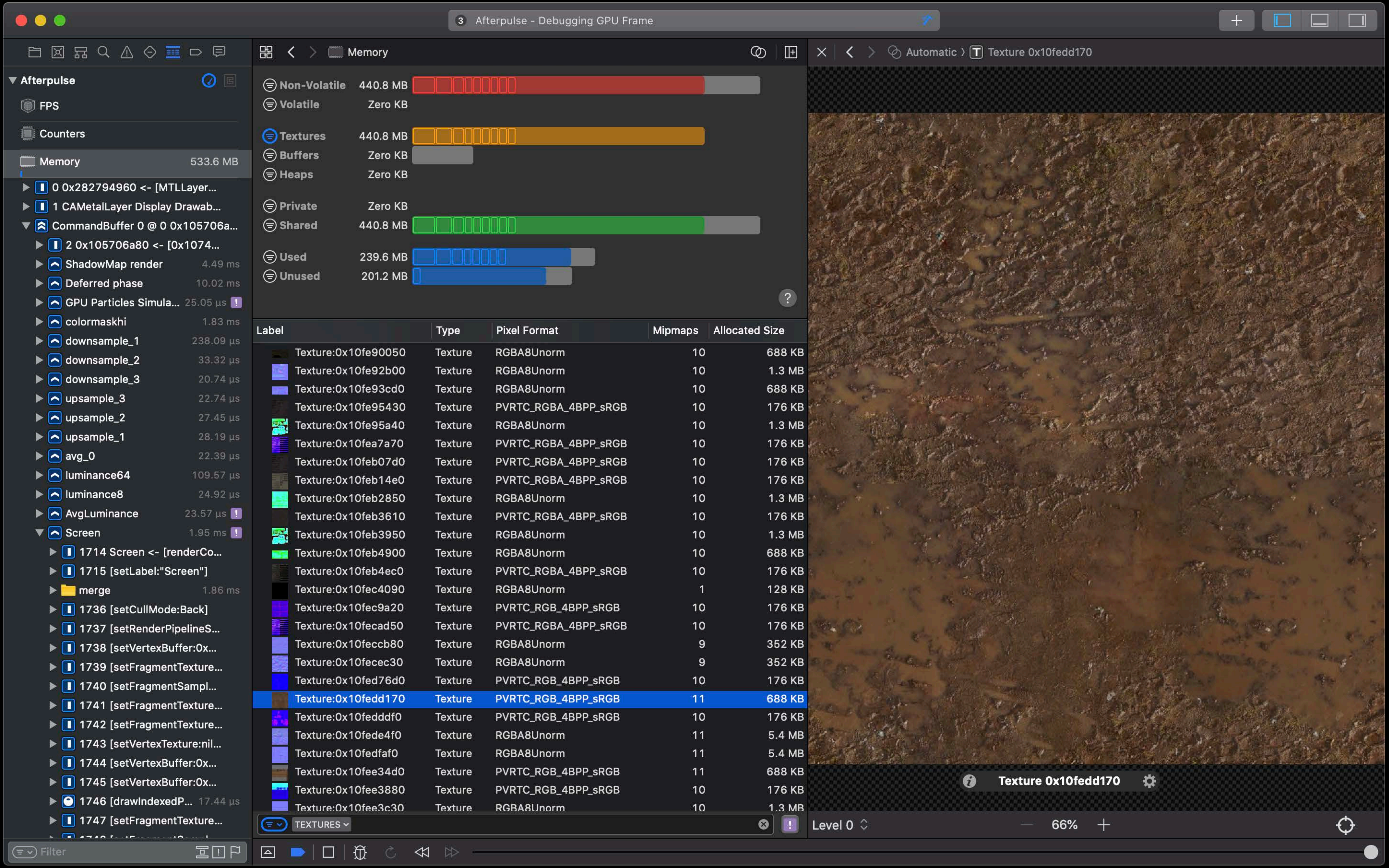


2048x2048
PVRTC_RGB_4BPP (mip mapped)

2.7MB

Metal Memory Viewer

NEW



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

```swift
// Create a texture with optimal GPU access

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

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

```swift
// Create a texture with optimal GPU access

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

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

```swift
// Create a texture with optimal GPU access

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

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

```swift
// 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()
```

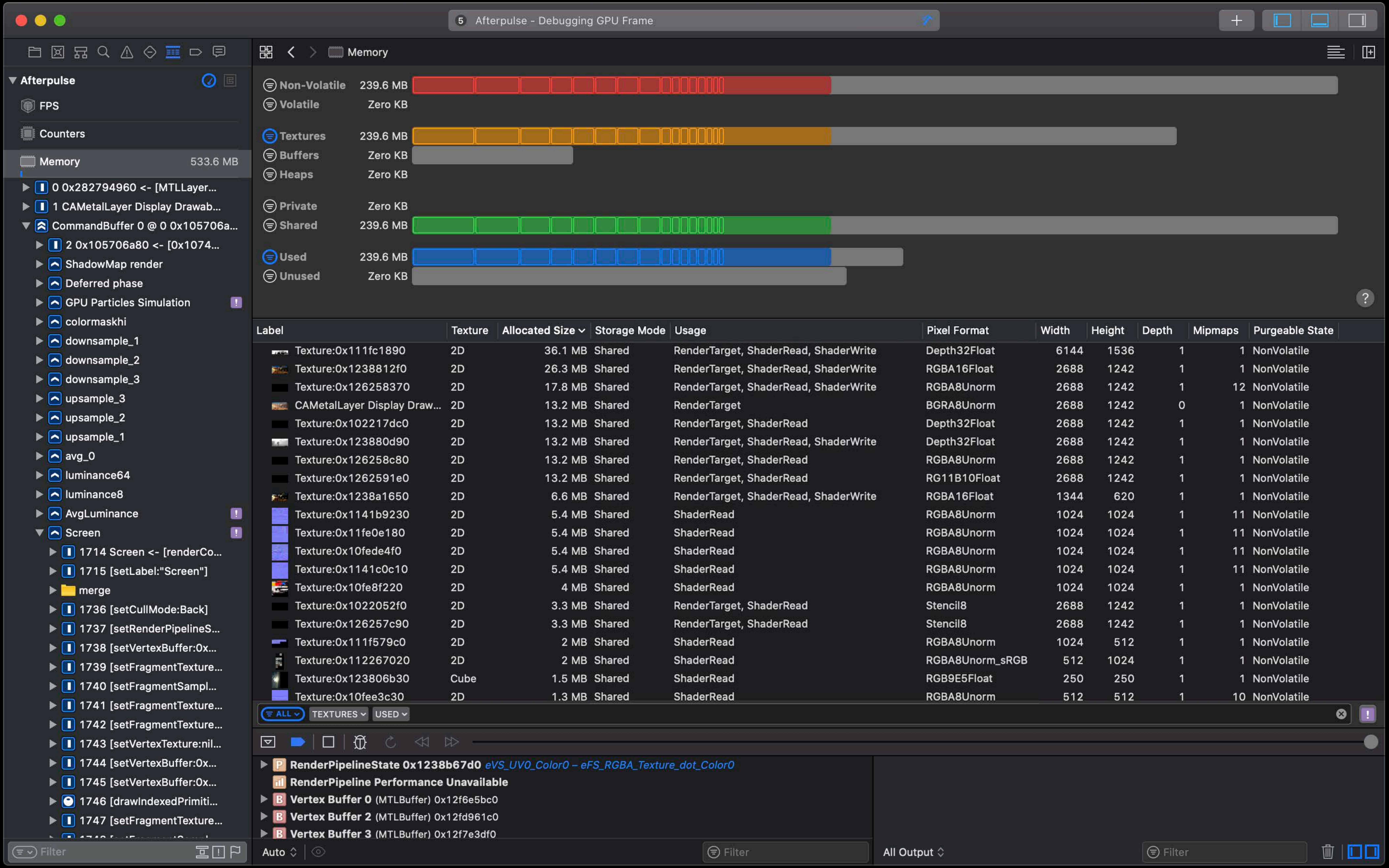
```swift
// Optimize Shared texture after CPU update

// ...
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()
```

```swift
// 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()
```

```swift
// 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()
```

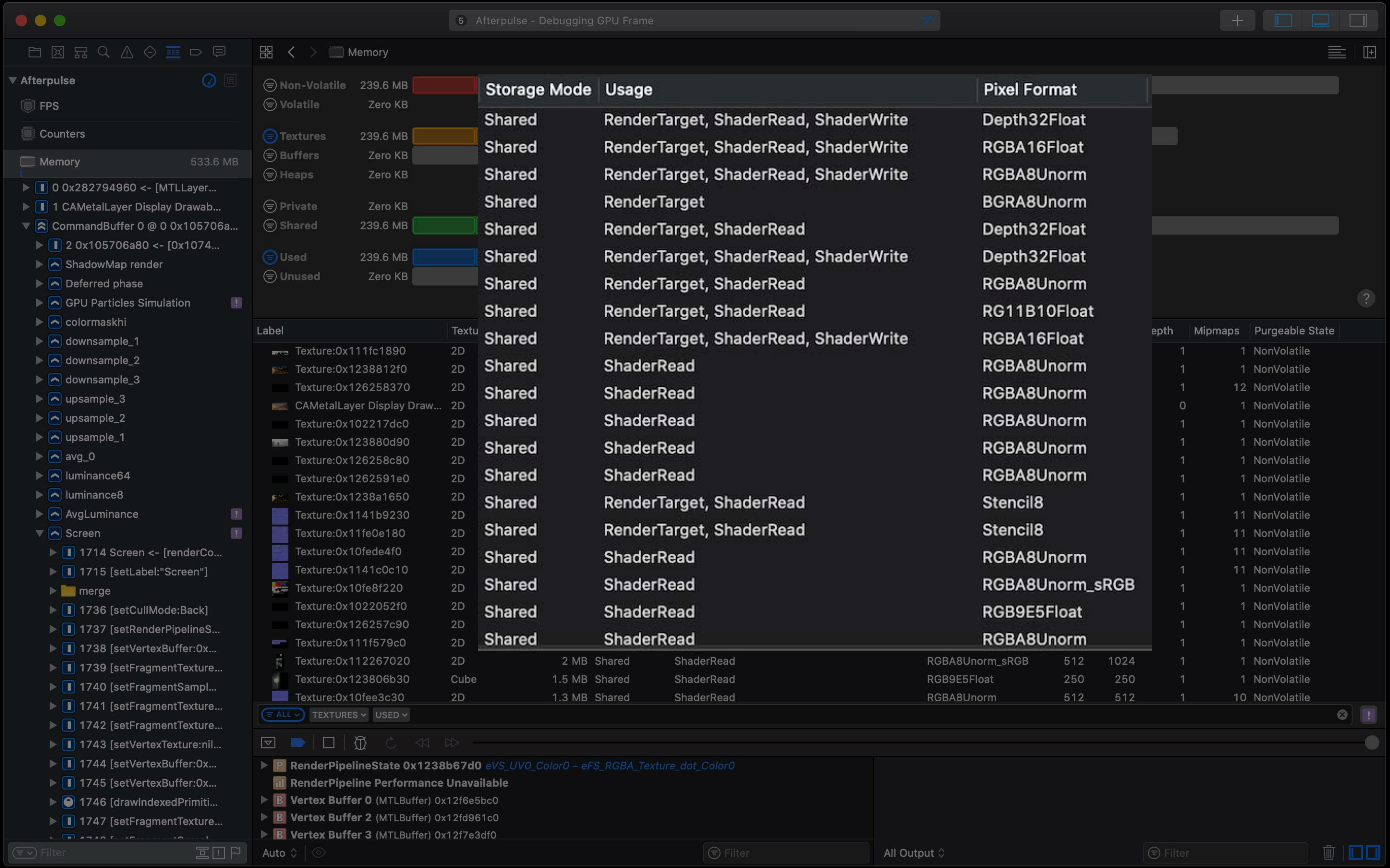
Metal Memory Viewer

NEW



Metal Memory Viewer

NEW



Best Practices

8. Choose the right pixel format

Larger pixel formats use more bandwidth

Sampling rate depends on pixel format

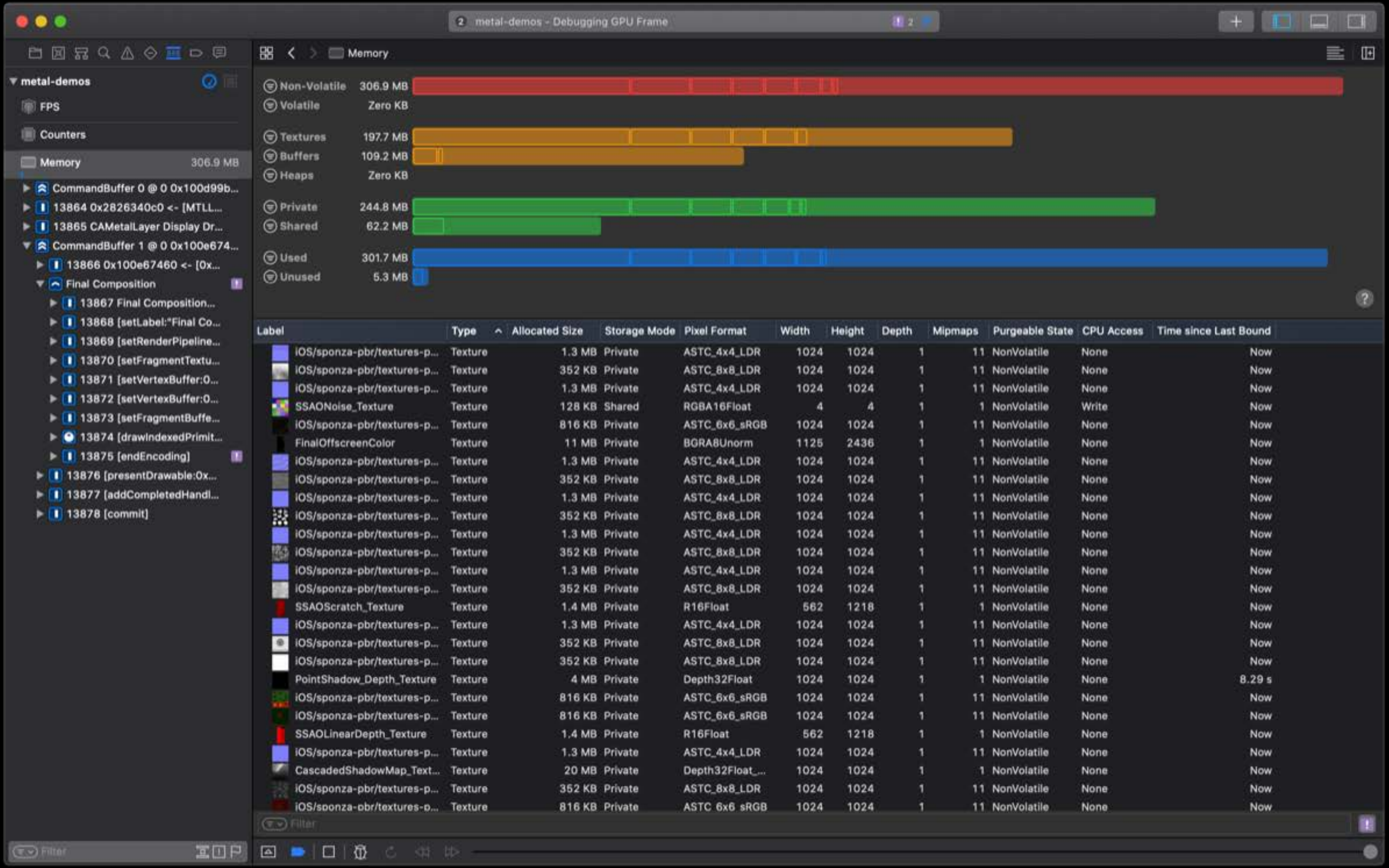
Best practice

- Avoid pixel formats with unnecessary channels
  - For example, using RGBA16 to store 2-component data
- Use lower precision whenever possible


Conventional
32-bit formats
1.0x
YCbCr (GBGR422 and BGRG422)
RG11B10Float
RGB9E5Float
64-bit formats
.5x
128-bit formats
.25x
Sampling Rate (A12 and later)

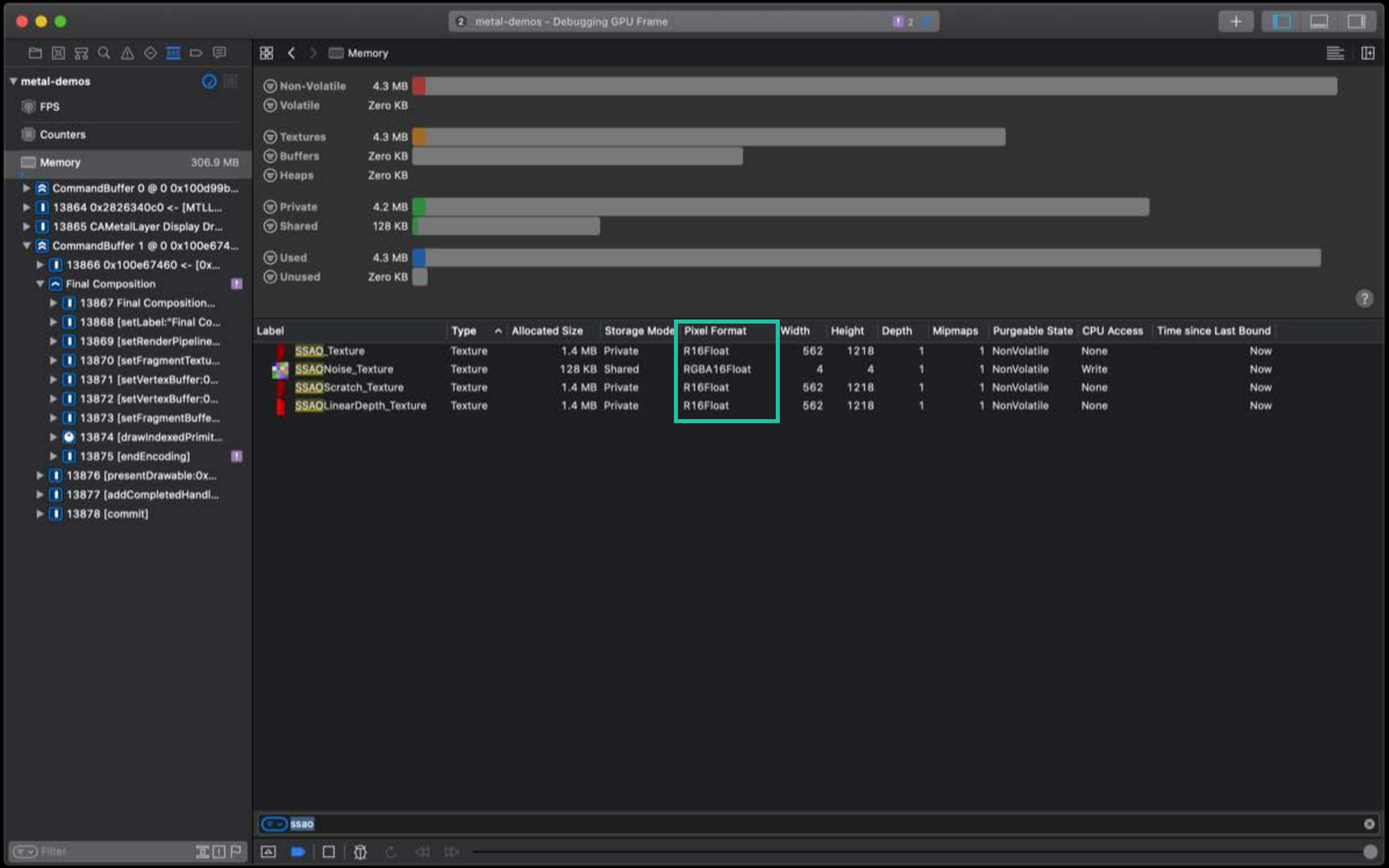
Metal Memory Viewer

NEW



Metal Memory Viewer

NEW



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

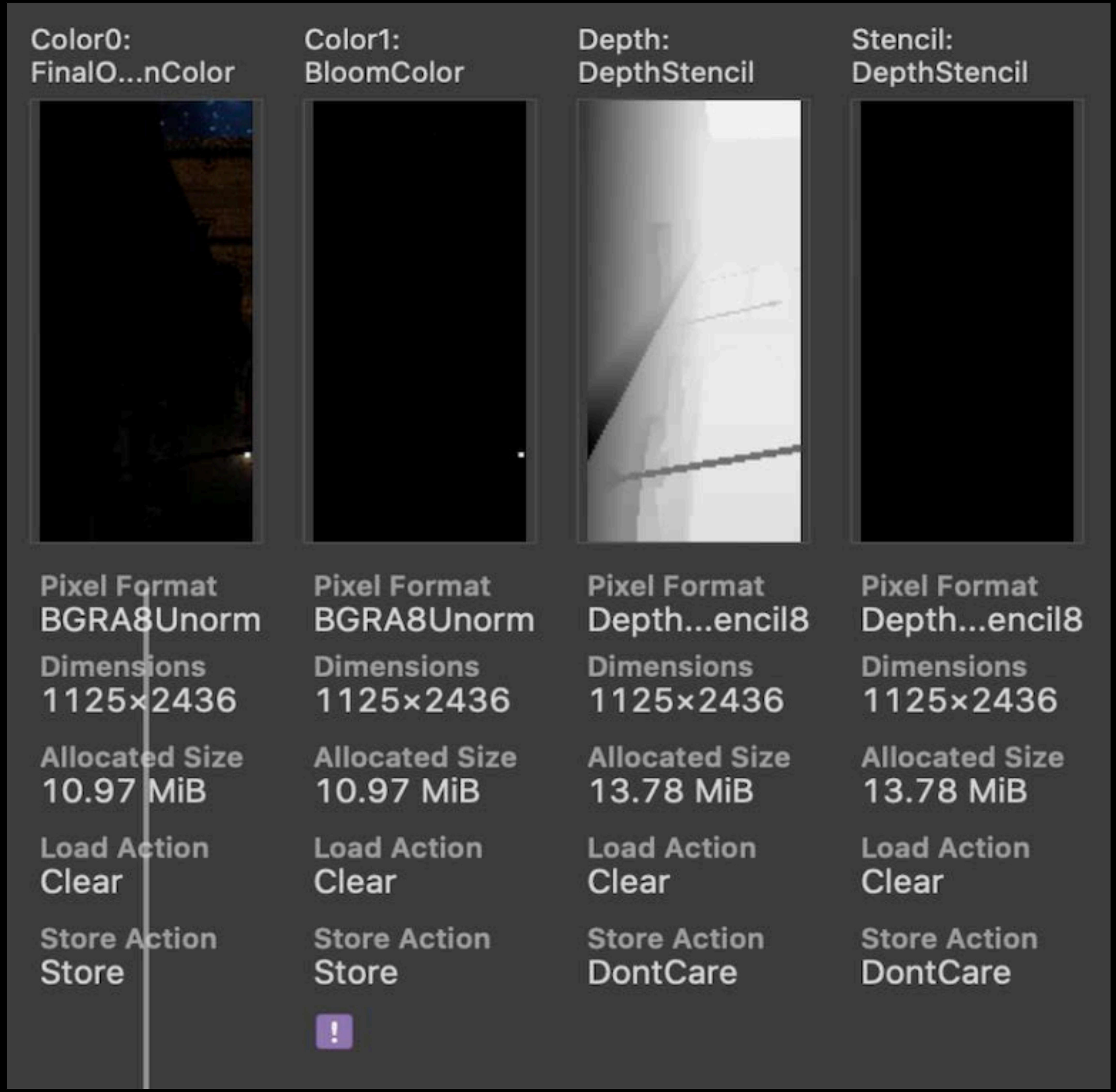
Suboptimal configuration may create false dependencies

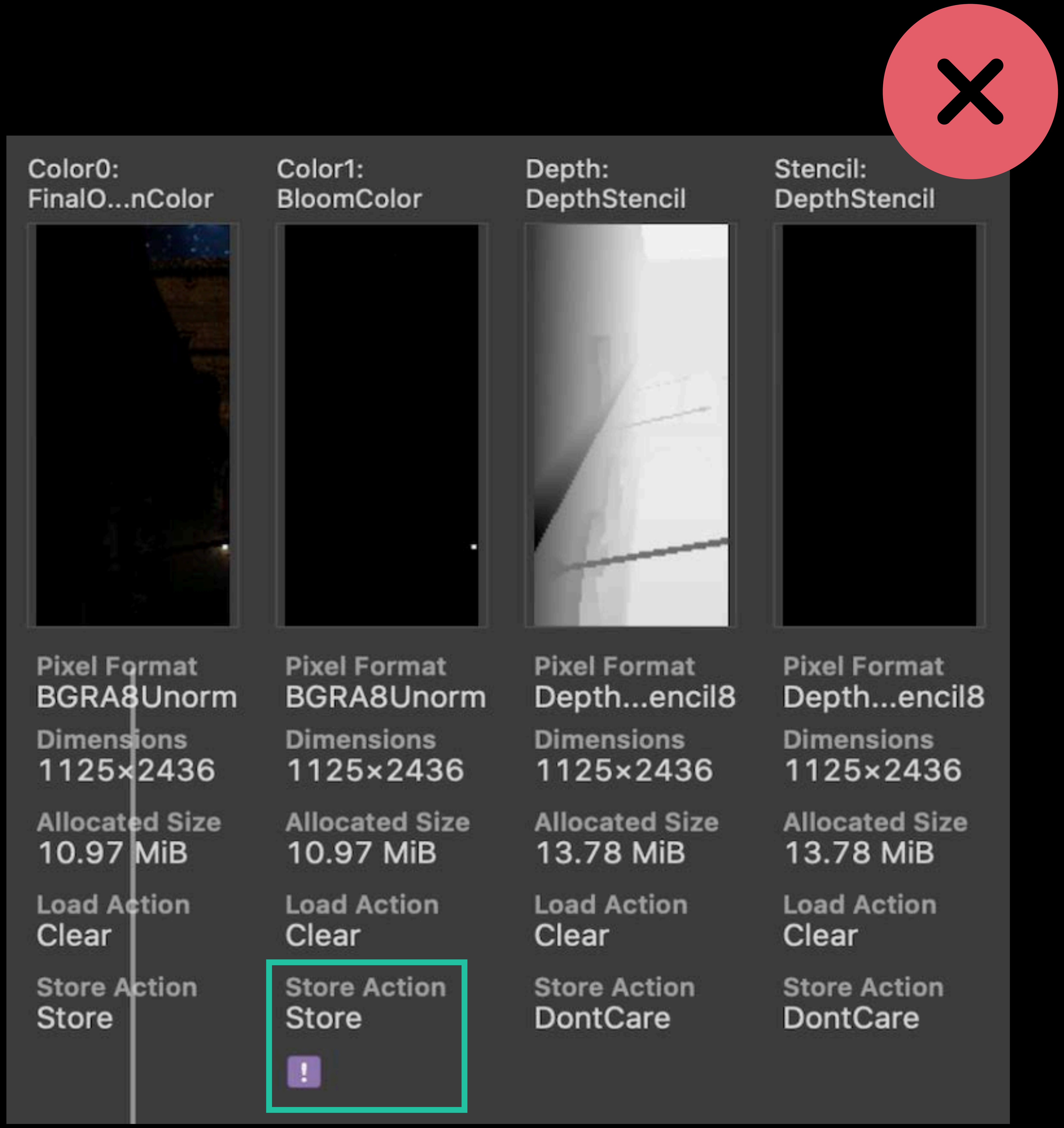
Best practice

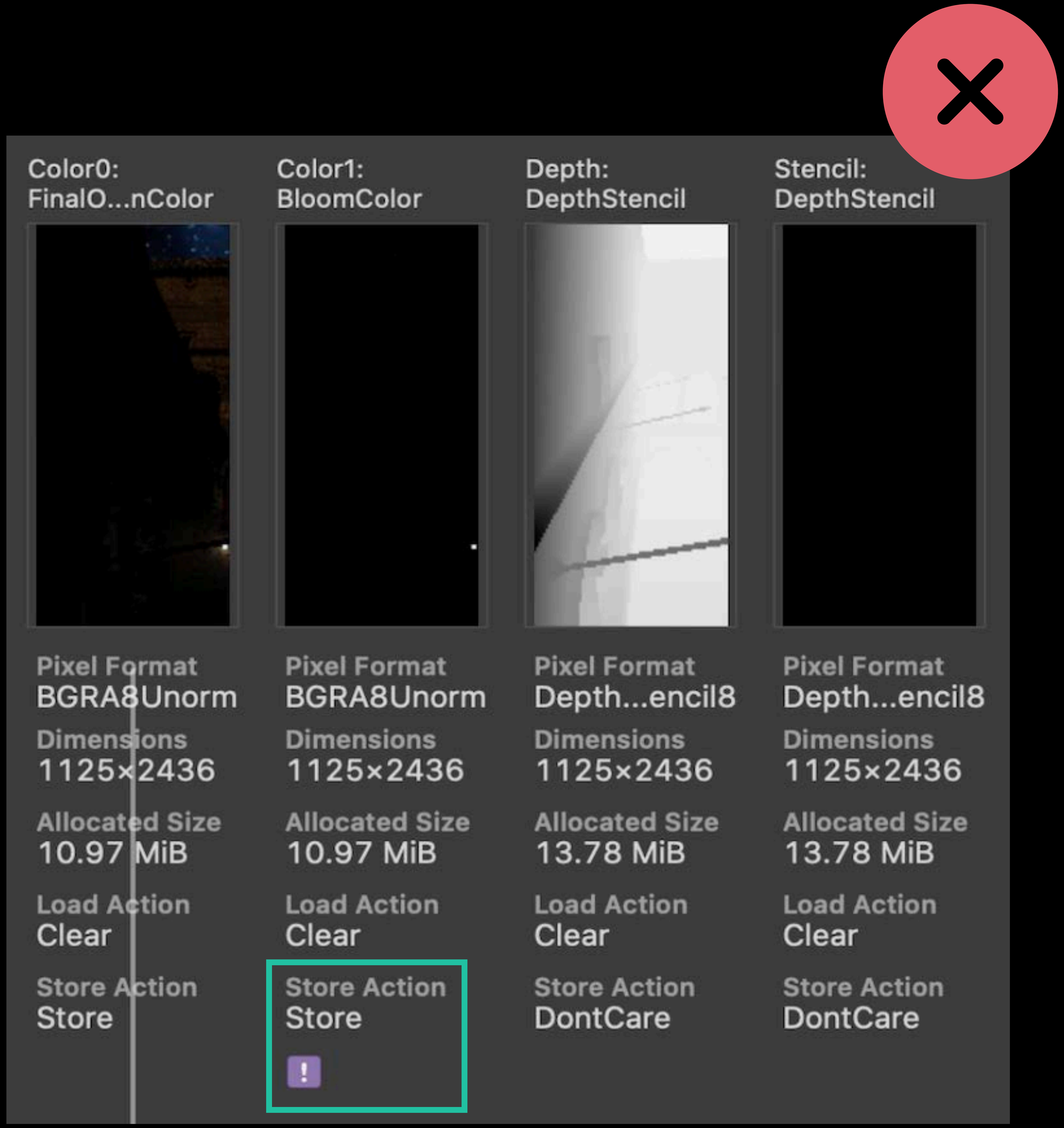
- Don't load or store render targets unless it's necessary

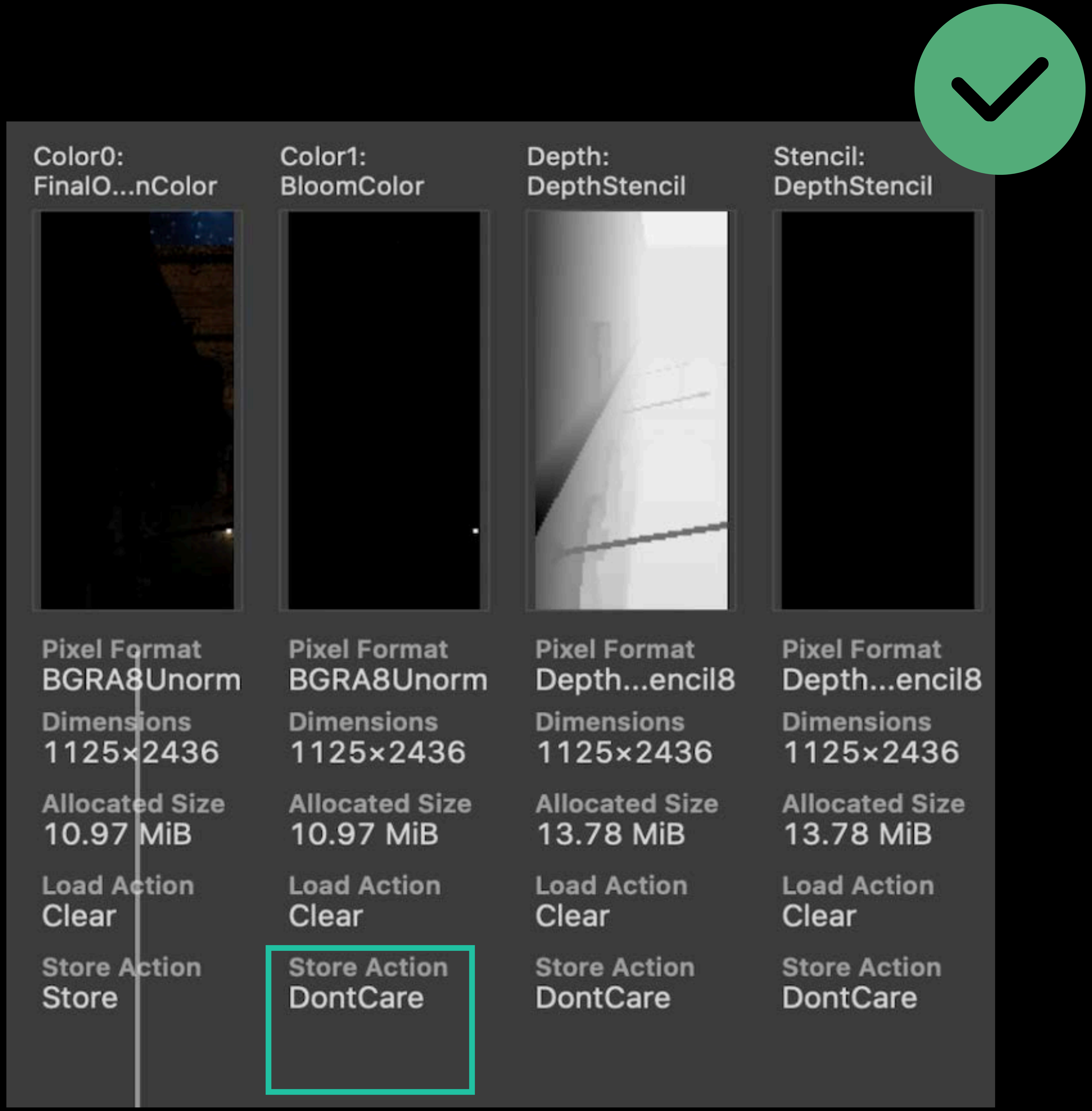
```swift
// Configure color attachment 1 as transient

// ...
renderPassDescriptor.colorAttachments[1].texture     = texture
renderPassDescriptor.colorAttachments[1].loadAction  = .clear
renderPassDescriptor.colorAttachments[1].storeAction = .dontCare
```

```swift
// Configure color attachment 1 as transient

// ...
renderPassDescriptor.colorAttachments[1].texture     = texture
renderPassDescriptor.colorAttachments[1].loadAction  = .clear
renderPassDescriptor.colorAttachments[1].storeAction = .dontCare
```


Color0:
FinalO...nColor
Color1:
BloomColor
Depth:
DepthStencil
Stencil:
DepthStencil
Pixel Format
BGRA8Unorm
Dimensions
1125×2436
Allocated Size
10.97 MiB
Load Action
Clear
Store Action
Store
Pixel Format
BGRA8Unorm
Dimensions
1125×2436
Allocated Size
10.97 MiB
Load Action
Clear
Store Action
Store
Pixel Format
Depth...encil8
Dimensions
1125×2436
Allocated Size
13.78 MiB
Load Action
Clear
Store Action
DontCare
Pixel Format
Depth...encil8
Dimensions
1125×2436
Allocated Size
13.78 MiB
Load Action
Clear
Store Action
DontCare


Color0:
FinalO...nColor
Color1:
BloomColor
Depth:
DepthStencil
Stencil:
DepthStencil
Pixel Format
BGRA8Unorm
Pixel Format
BGRA8Unorm
Pixel Format
Depth...encil8
Pixel Format
Depth...encil8
Dimensions
1125×2436
Dimensions
1125×2436
Dimensions
1125×2436
Dimensions
1125×2436
Allocated Size
10.97 MiB
Allocated Size
10.97 MiB
Allocated Size
13.78 MiB
Allocated Size
13.78 MiB
Load Action
Clear
Load Action
Clear
Load Action
Clear
Load Action
Clear
Store Action
Store
Store Action
Store
Store Action
DontCare
Store Action
DontCare


Color0: FinalO...nColor
Color1: BloomColor
Depth: DepthStencil
Stencil: DepthStencil
Pixel Format BGRA8Unorm
Pixel Format BGRA8Unorm
Pixel Format Depth...encil8
Pixel Format Depth...encil8
Dimensions 1125×2436
Dimensions 1125×2436
Dimensions 1125×2436
Dimensions 1125×2436
Allocated Size 10.97 MiB
Allocated Size 10.97 MiB
Allocated Size 13.78 MiB
Allocated Size 13.78 MiB
Load Action Clear
Load Action Clear
Load Action Clear
Load Action Clear
Store Action Store
Store Action Store
Store Action DontCare
Store Action DontCare


Color0: FinalO...nColor
Color1: BloomColor
Depth: DepthStencil
Stencil: DepthStencil
Pixel Format BGRA8Unorm
Pixel Format BGRA8Unorm
Pixel Format Depth...encil8
Pixel Format Depth...encil8
Dimensions 1125×2436
Dimensions 1125×2436
Dimensions 1125×2436
Dimensions 1125×2436
Allocated Size 10.97 MiB
Allocated Size 10.97 MiB
Allocated Size 13.78 MiB
Allocated Size 13.78 MiB
Load Action Clear
Load Action Clear
Load Action Clear
Load Action Clear
Store Action Store
Store Action DontCare
Store Action DontCare
Store Action DontCare

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

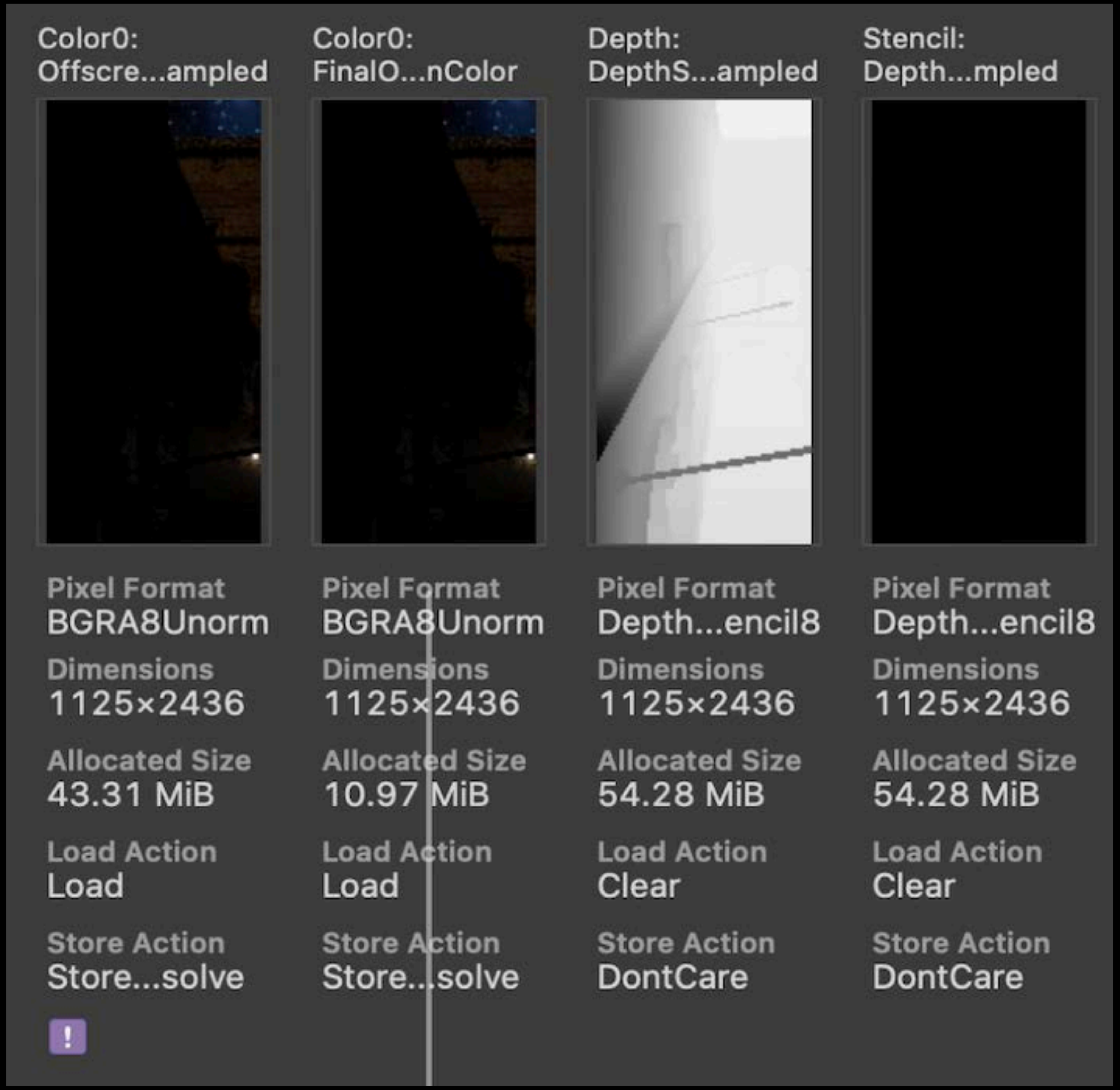
Best practice

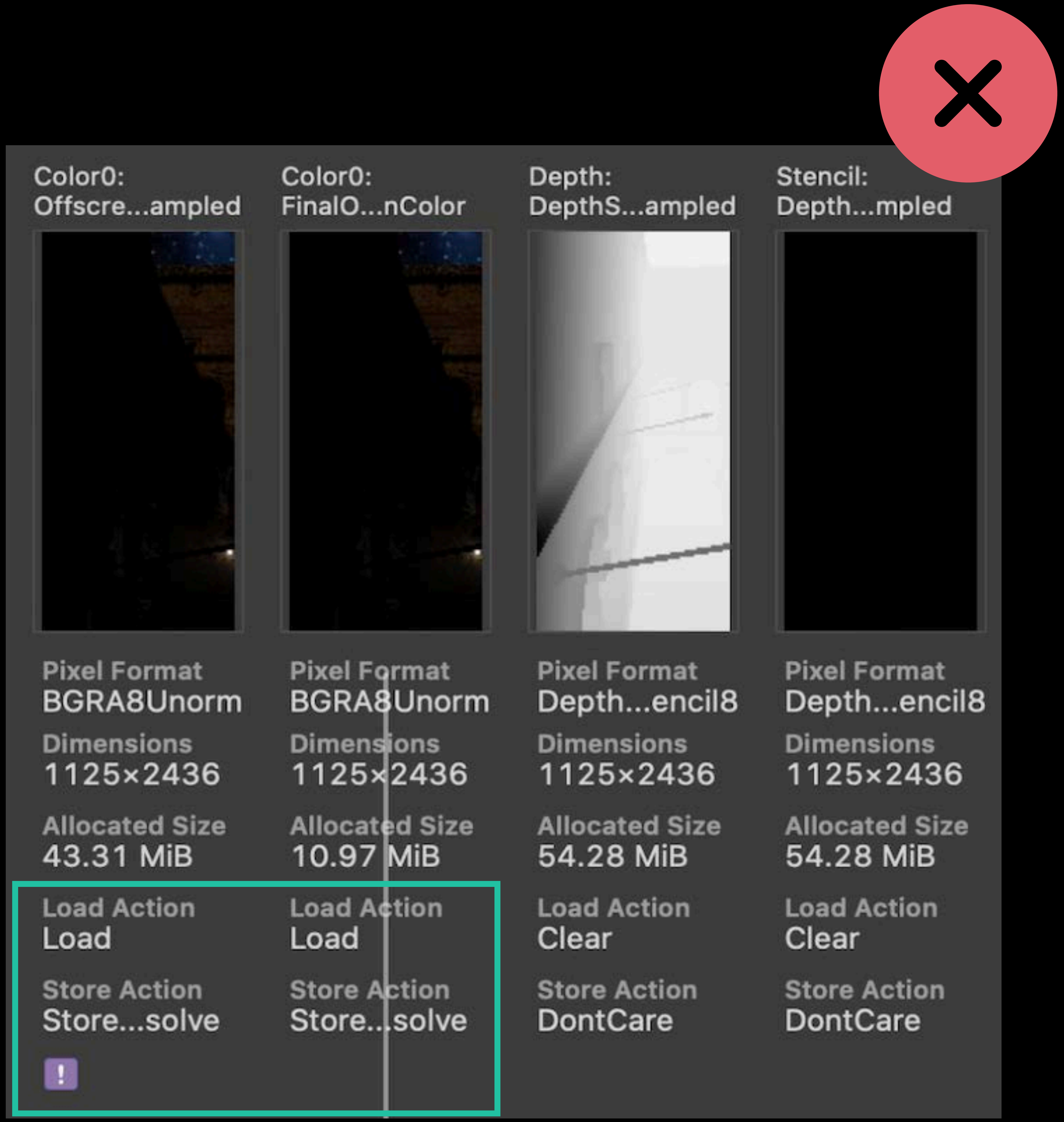
- Consider MSAA over native resolution
- Don’t load or store the multisample texture
- Set the storage mode of the multisample texture to `memoryless`

```swift
// 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
```

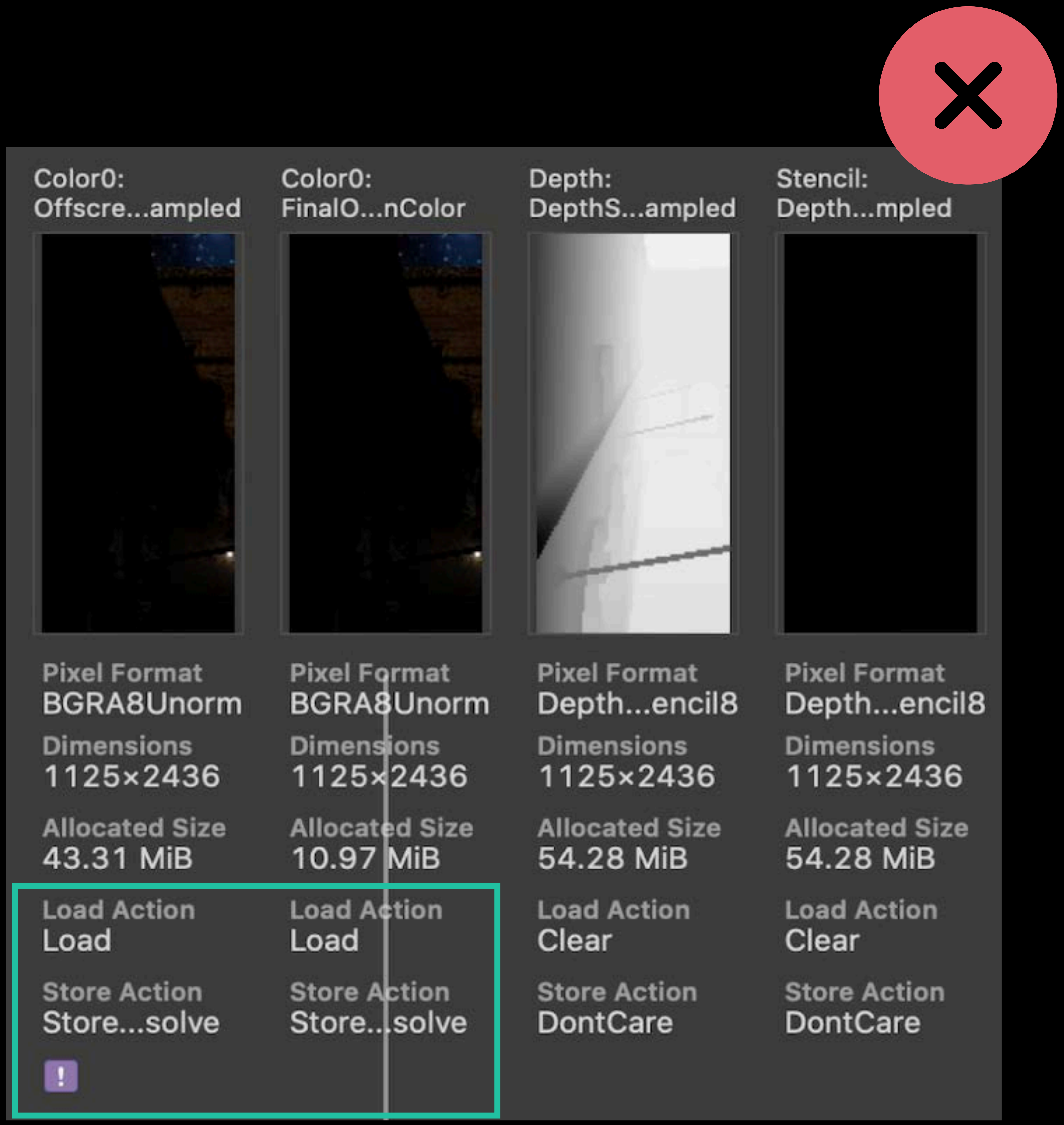
```swift
// Create a multisample texture

// ...
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
```

```swift
// 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
```

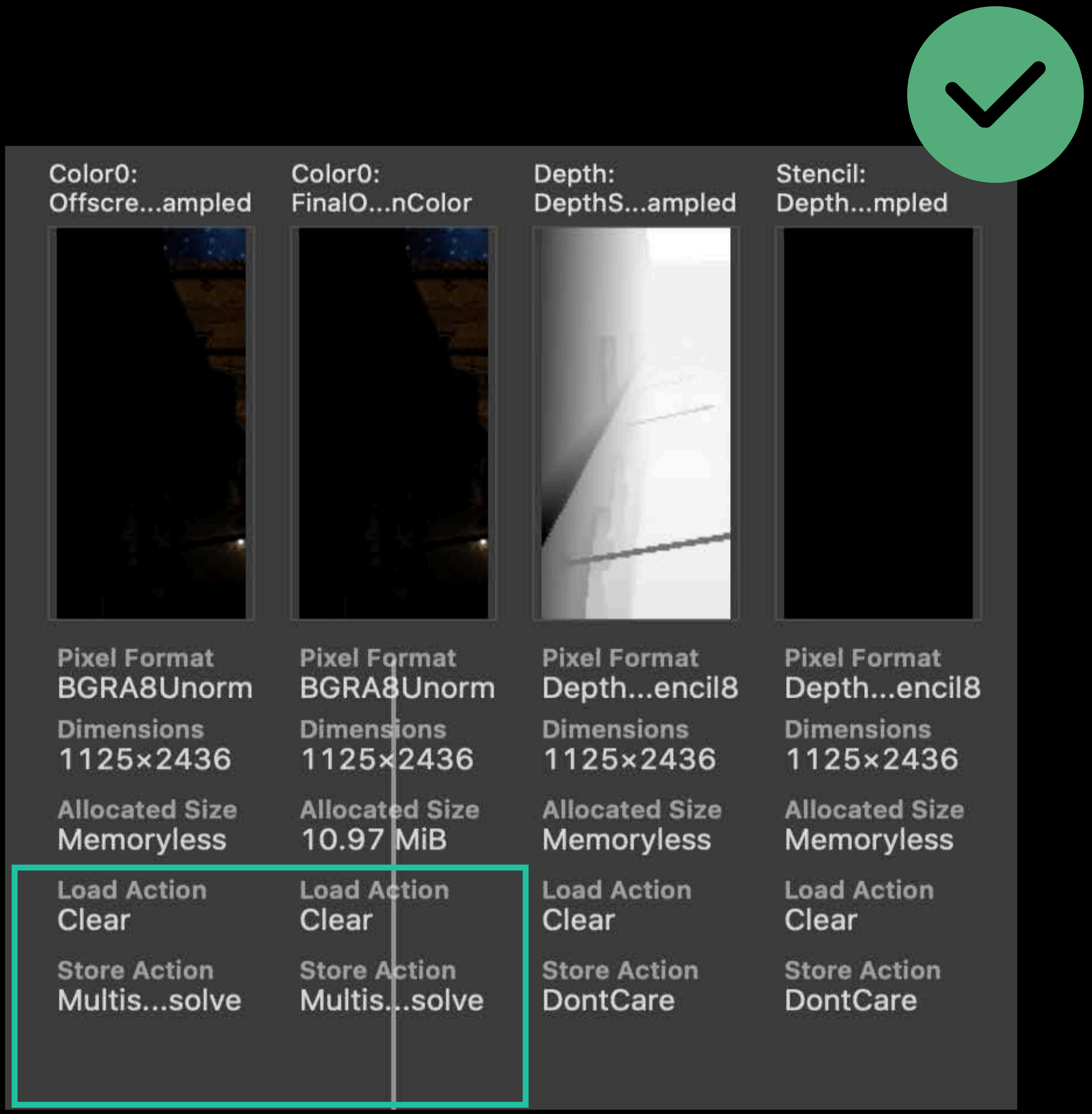

Color0:
Offscre...ampled
Color0:
FinalO...nColor
Depth:
DepthS...ampled
Stencil:
Depth...mpled
Pixel Format
BGRA8Unorm
Dimensions
1125×2436
Allocated Size
43.31 MiB
Load Action
Load
Store Action
Store...solve
Pixel Format
BGRA8Unorm
Dimensions
1125×2436
Allocated Size
10.97 MiB
Load Action
Load
Store Action
Store...solve
Pixel Format
Depth...encil8
Dimensions
1125×2436
Allocated Size
54.28 MiB
Load Action
Clear
Store Action
DontCare
Pixel Format
Depth...encil8
Dimensions
1125×2436
Allocated Size
54.28 MiB
Load Action
Clear
Store Action
DontCare



~97 MB footprint
~97 MB bandwidth



~97 MB footprint
~97 MB bandwidth



~11 MB footprint
~11 MB bandwidth

Best Practices

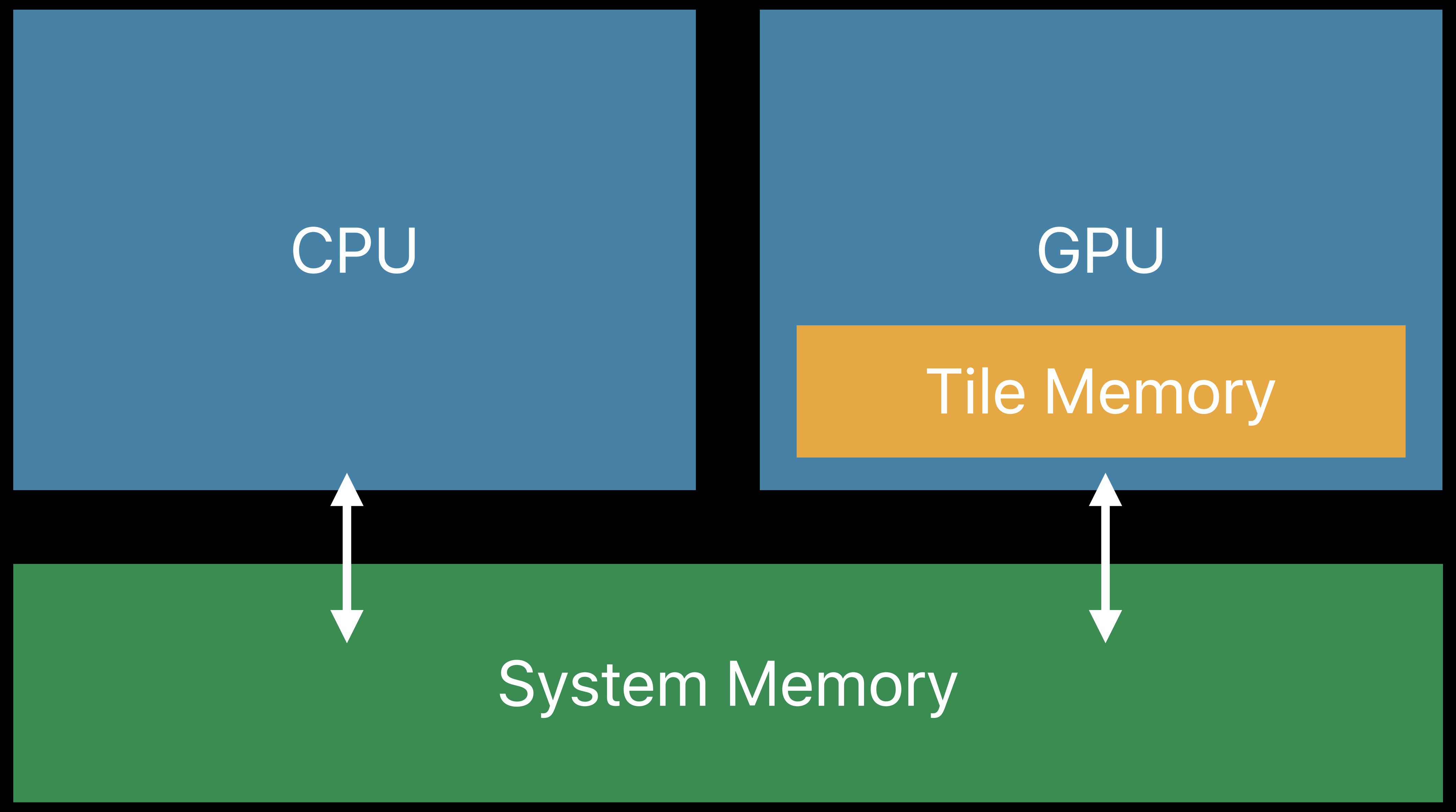
11. Leverage tile memory

Metal provides access to tile memory

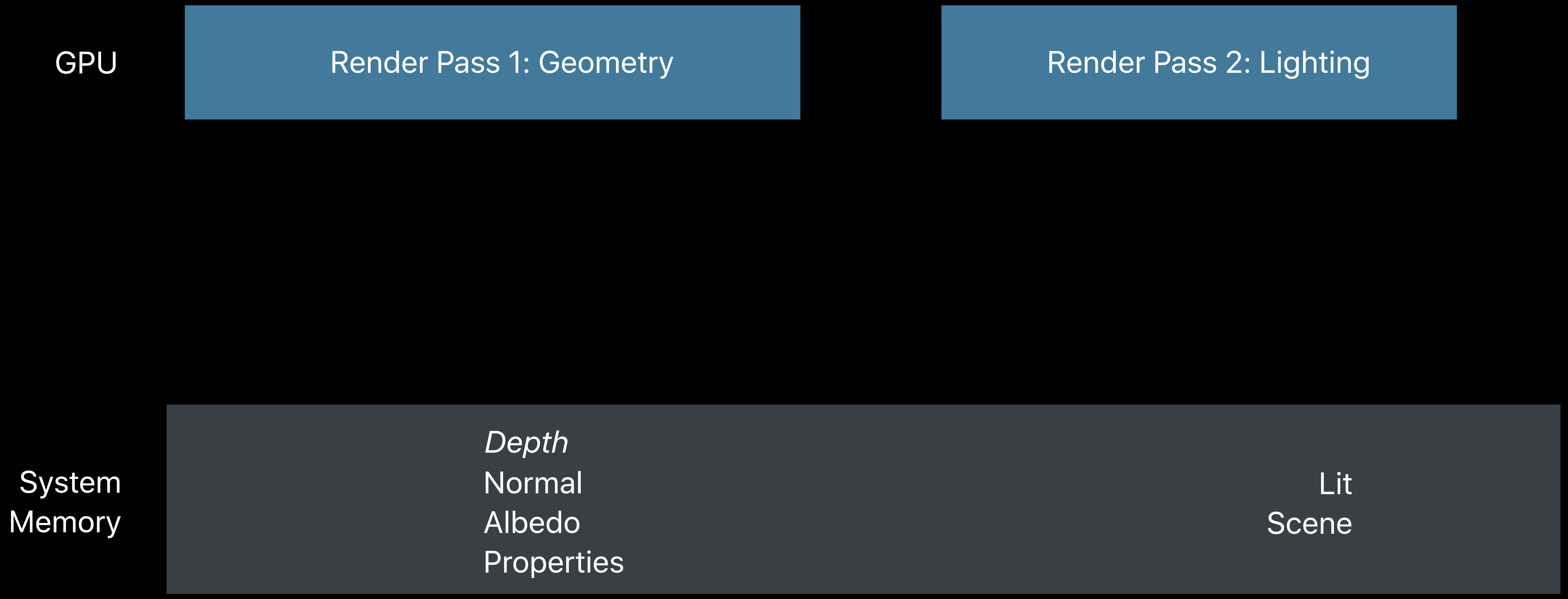
- Programmable blending
- Image blocks
- Tile shaders

Best practice

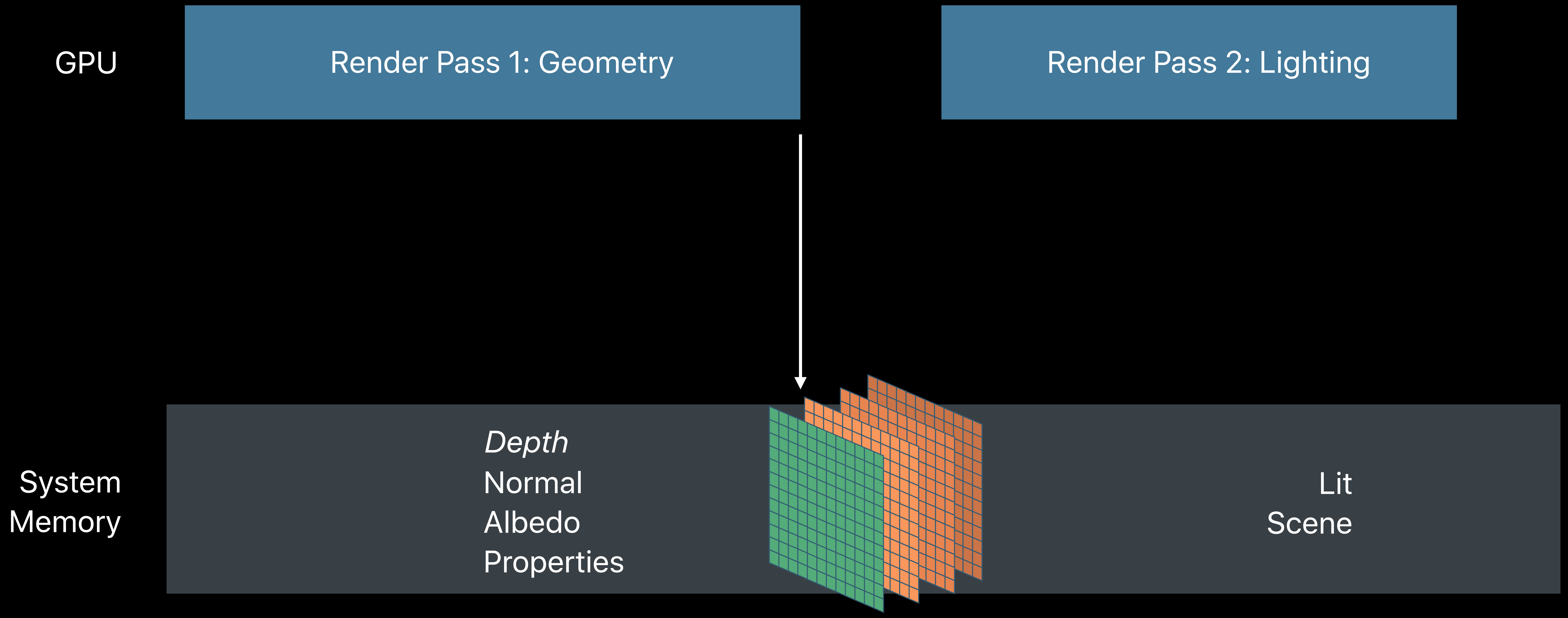
- Explicitly utilize tile memory



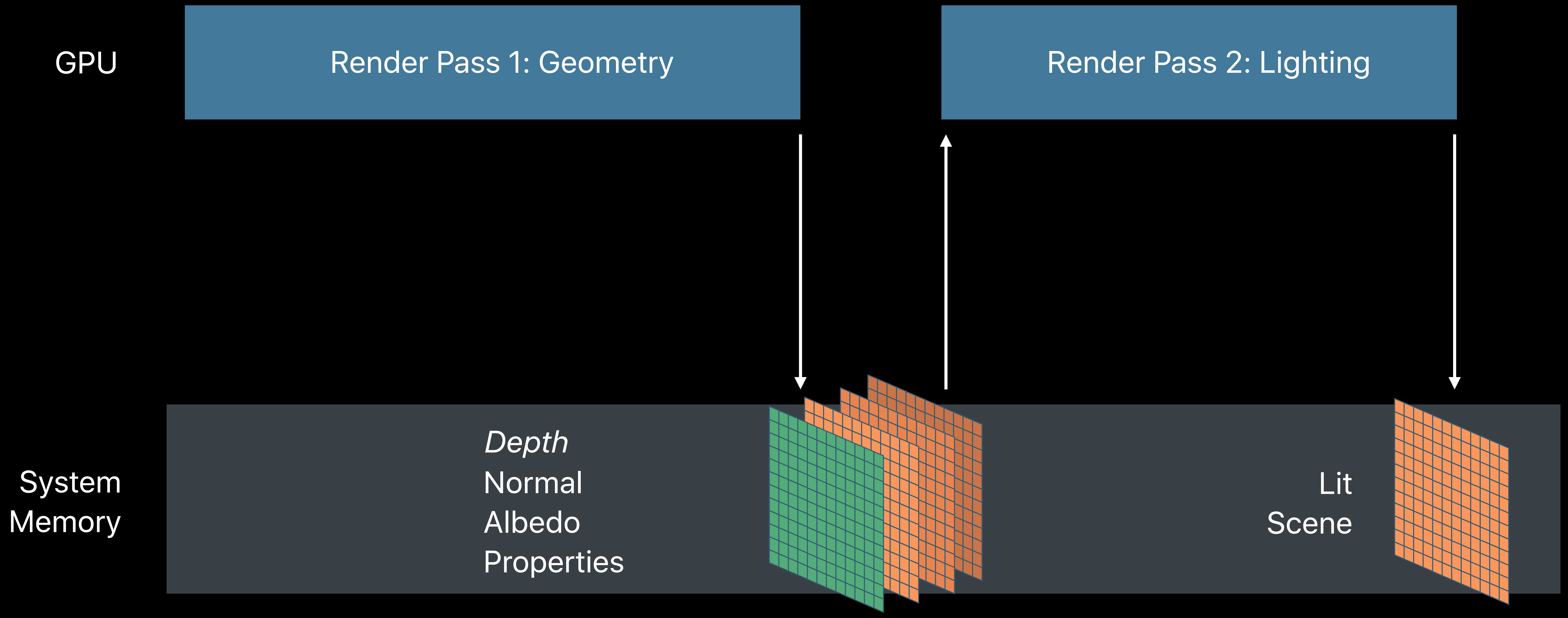
Traditional Deferred Shading



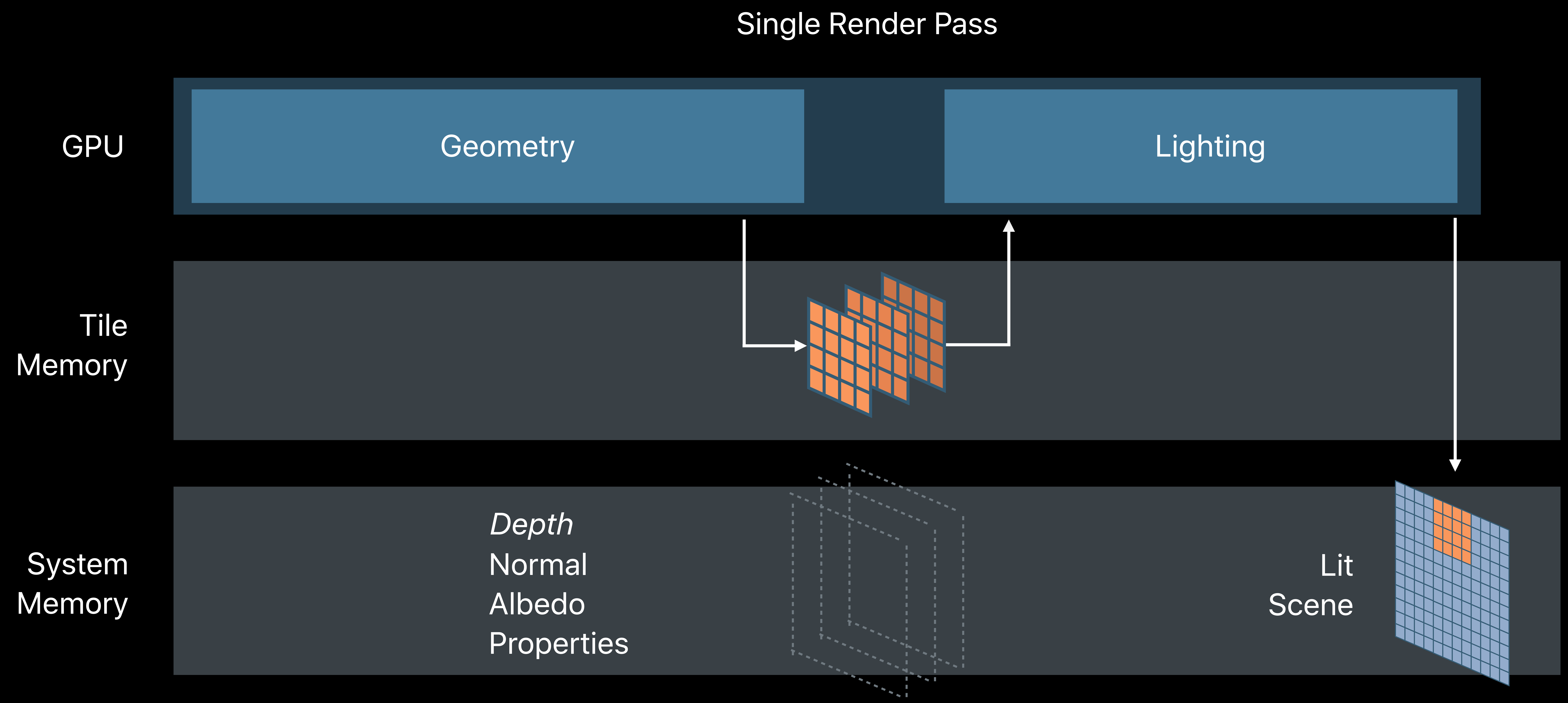
Traditional Deferred Shading



Traditional Deferred Shading



Single Pass Deferred Shading



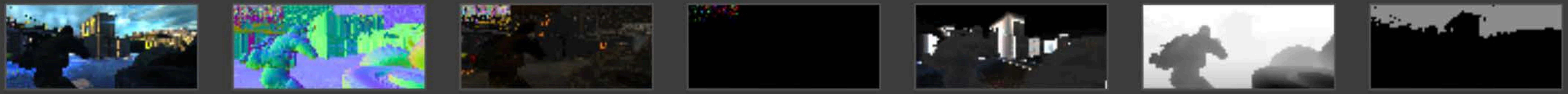
GPU Timing
15.94 ms

Draw Calls
163

Dispatches
0

Vertices
226158

R Deferred phase



	Color0: 0x11cdccd20	Color1: 0x11cdd7b90	Color2: 0x11cdd7f30	Color3: 0x11cdd82d0	Color4: 0x11cdd8670	Depth: 0x11cdcc6b0	Stencil: 0x11cdd7670
Pixel Format	RGBA16Float	RGBA8Unorm	RGBA8Unorm	RG11...0Float	RGBA16Float	Depth32Float	Stencil8
Dimensions	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125
Allocated Size	21.66 MiB	Memoryless	Memoryless	Memoryless	Memoryless	10.97 MiB	Memoryless
Load Action	Clear	DontCare	DontCare	DontCare	Clear	Clear	Clear
Store Action	Store	DontCare	DontCare	DontCare	DontCare	Store	DontCare

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 ›

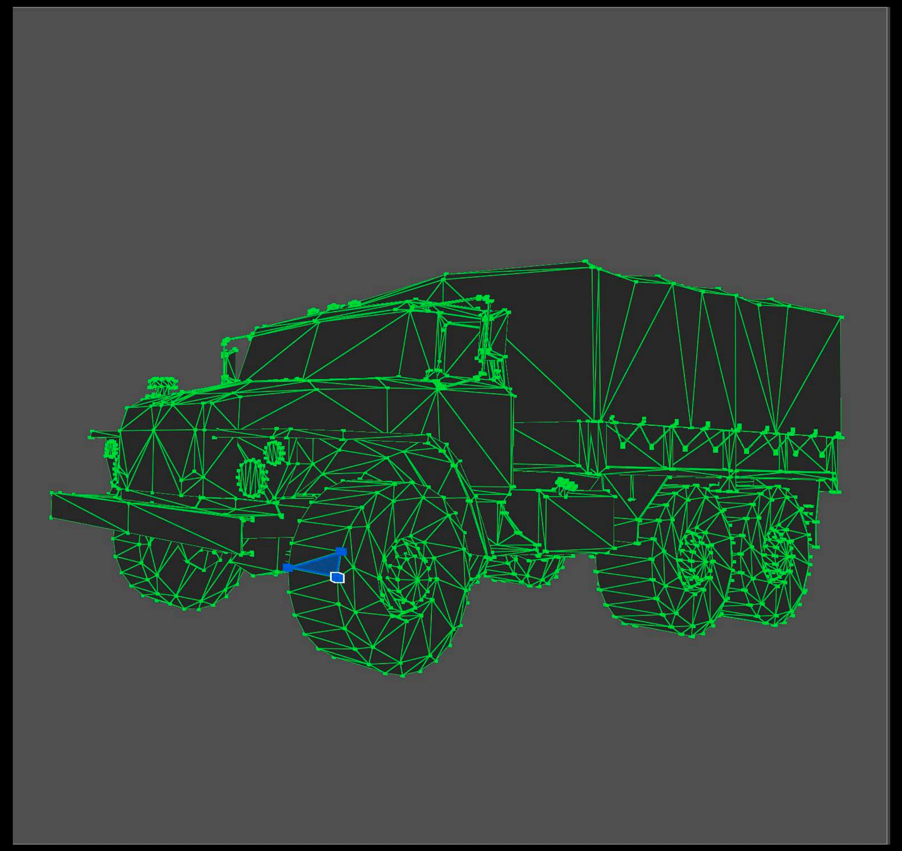
Metal Resources

iOS 12 accounting changes affect

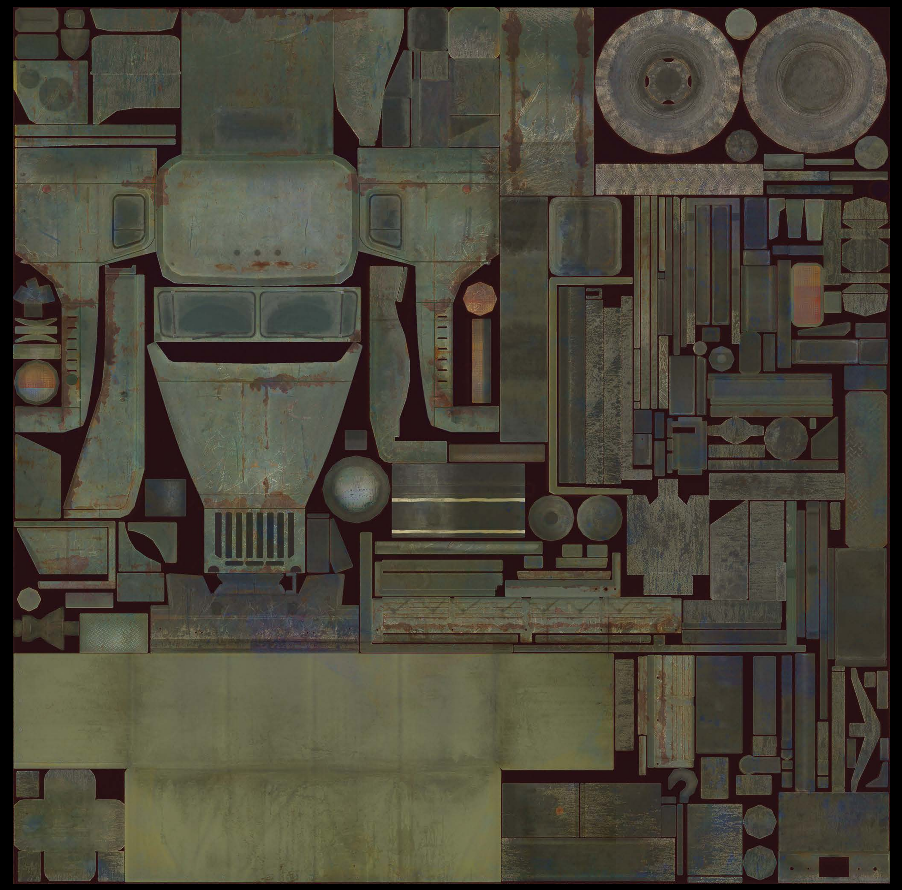
- Metal buffers
- Metal textures

These are used for

- Game assets (meshes, materials, etc.)
- Post process effects (shadows, blur, etc.)

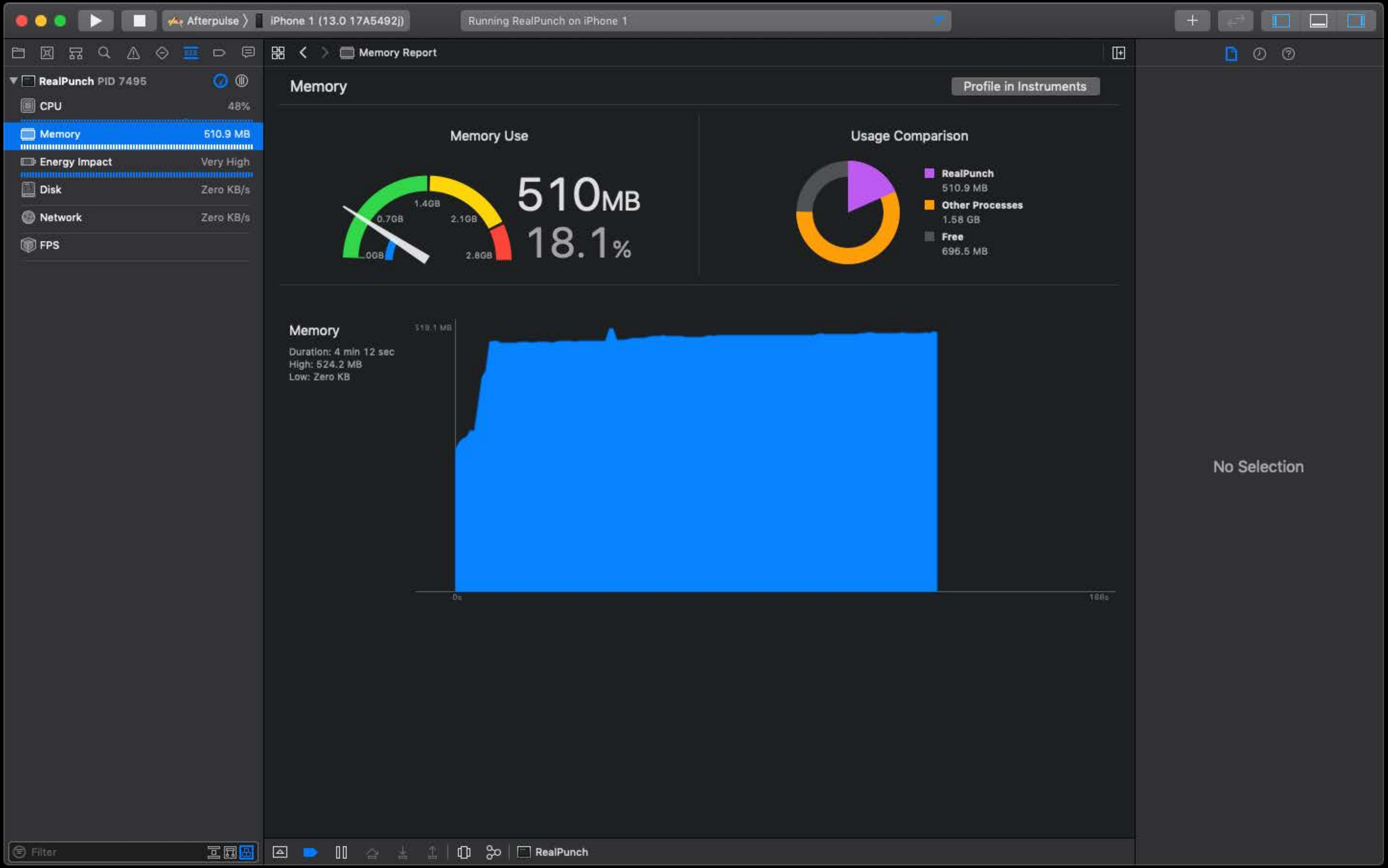


Buffer



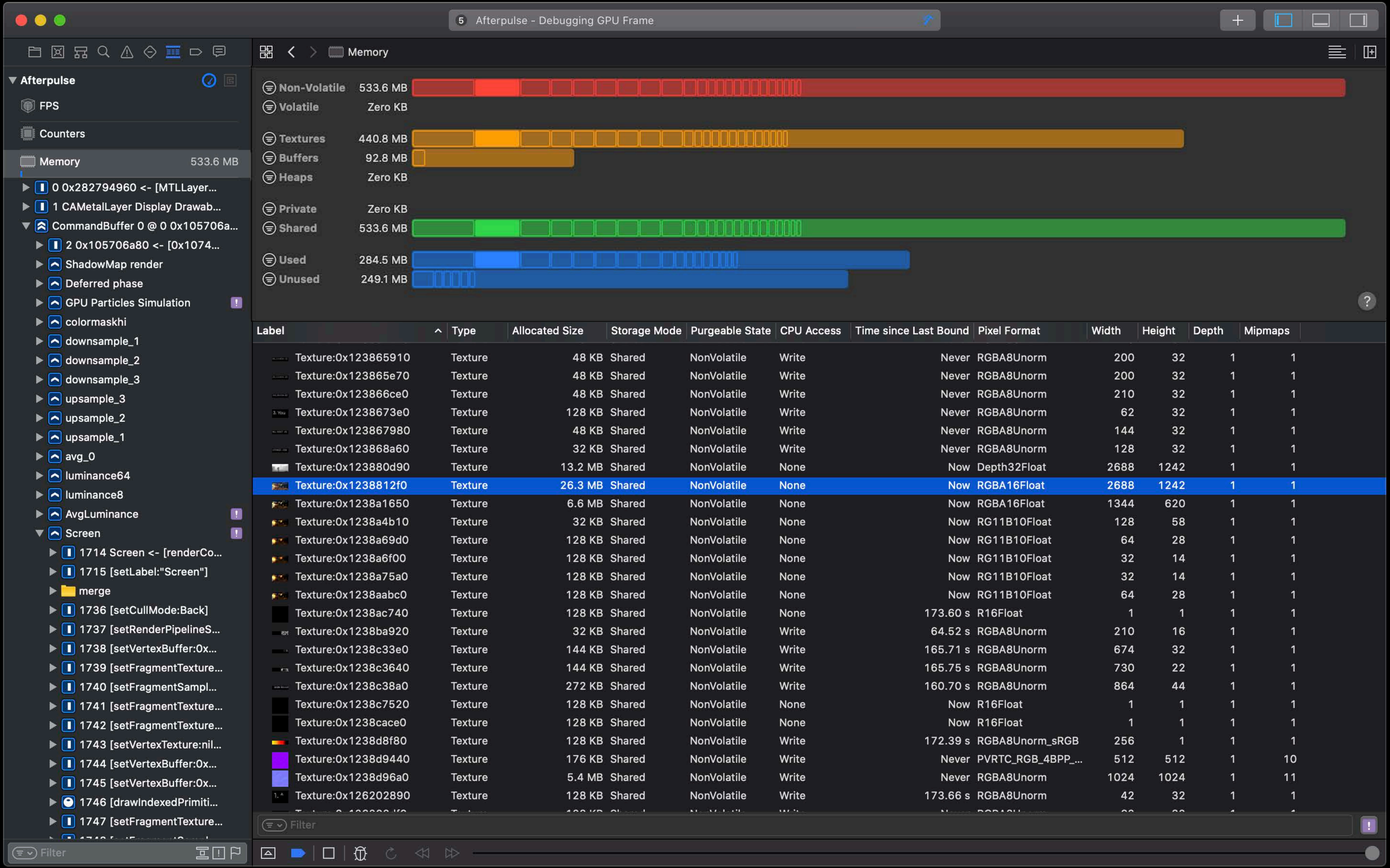
Texture

Xcode 11


Afterpulse
iPhone 1 (13.0 17A5492j)
Running RealPunch on iPhone 1
Memory Report
RealPunch PID 7495
CPU 48%
Memory 510.9 MB
Energy Impact Very High
Disk Zero KB/s
Network Zero KB/s
FPS
Memory
Profile in Instruments
Memory Use
1.4GB
0.7GB
2.1GB
0GB
2.8GB
510MB
18.1%
Usage Comparison
RealPunch
510.9 MB
Other Processes
1.58 GB
Free
696.5 MB
Memory
Duration: 4 min 12 sec
High: 524.2 MB
Low: Zero KB
No Selection
Filter
RealPunch

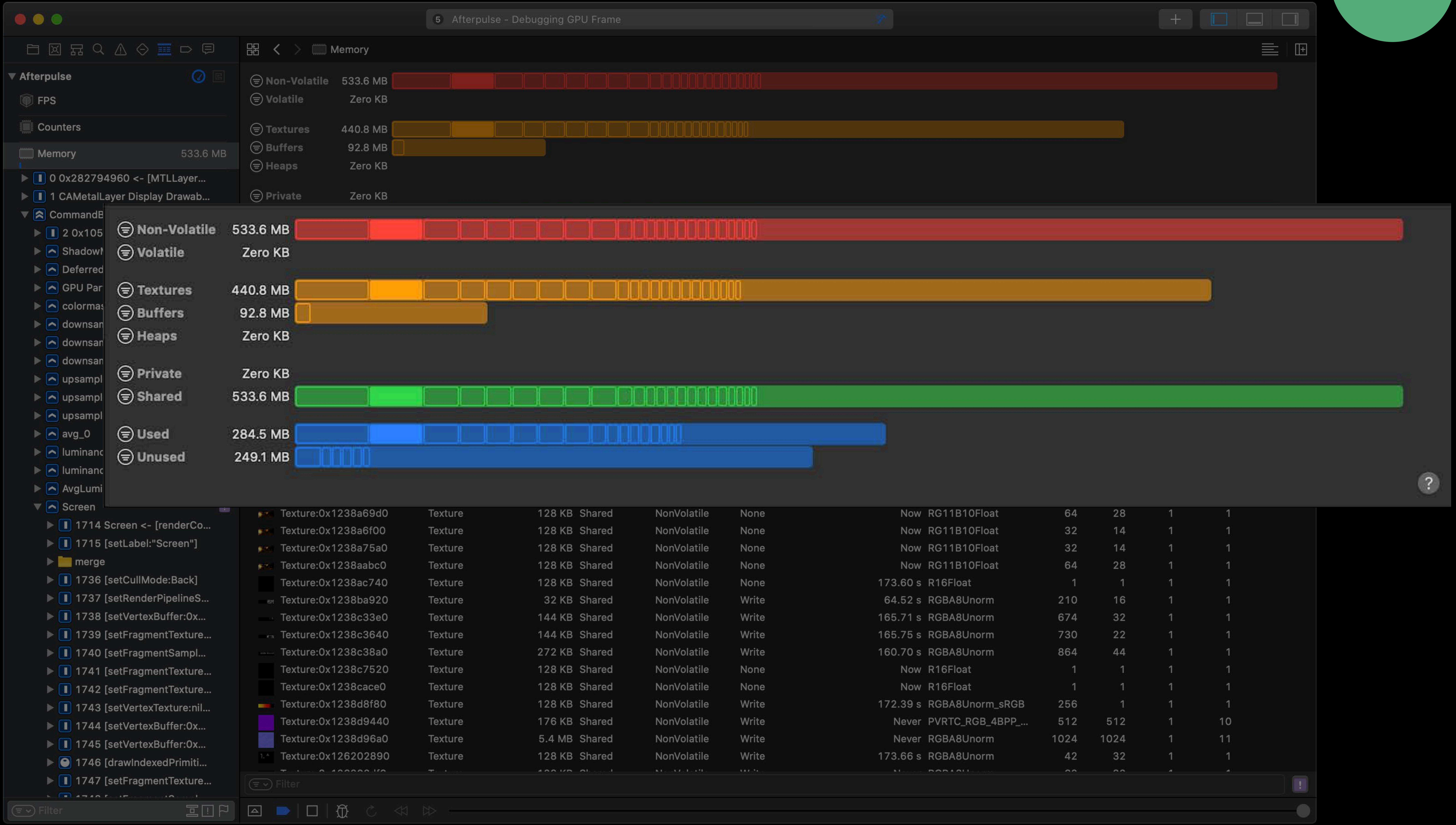
Metal Memory Viewer

NEW



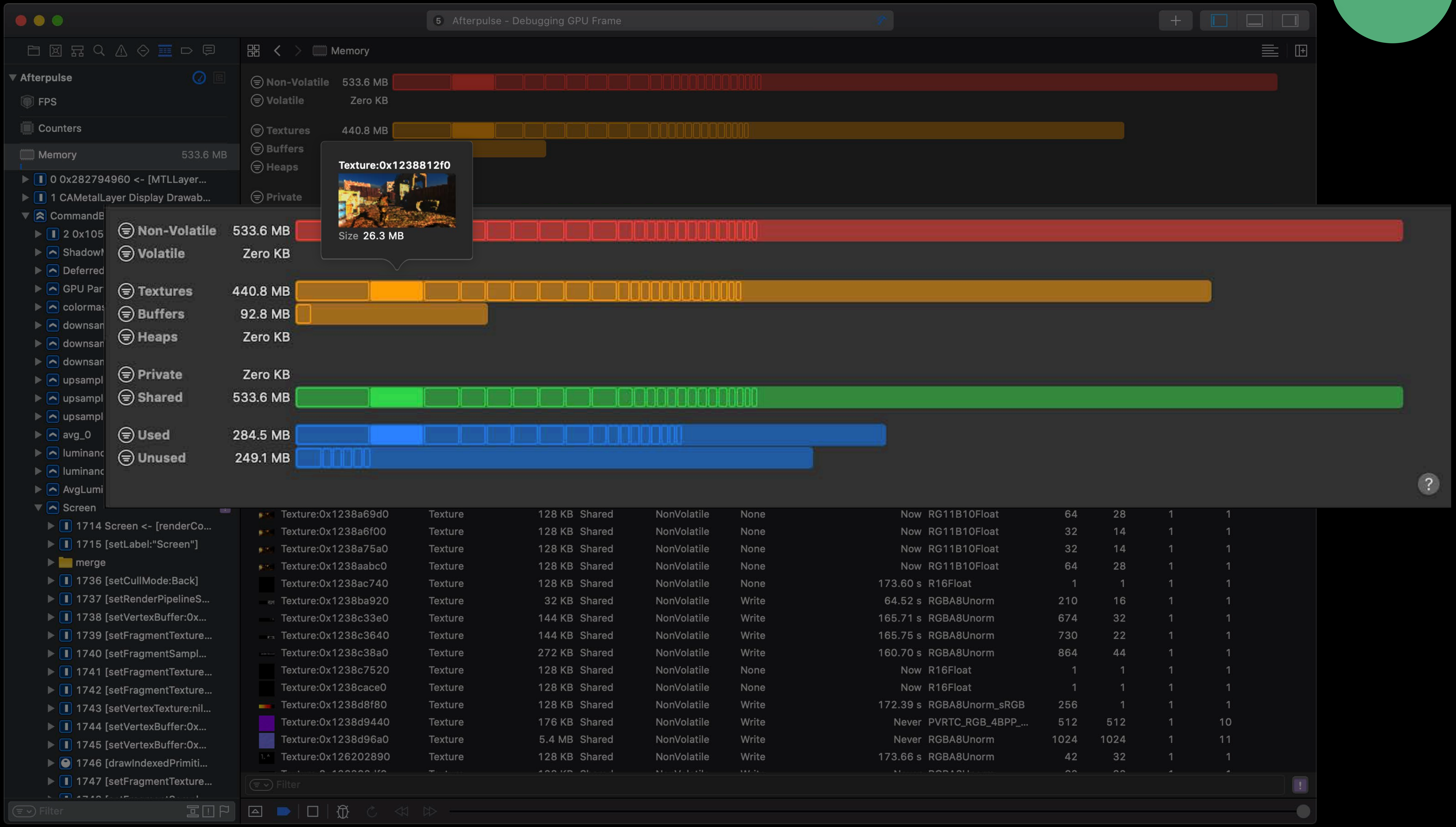
Metal Memory Viewer

NEW



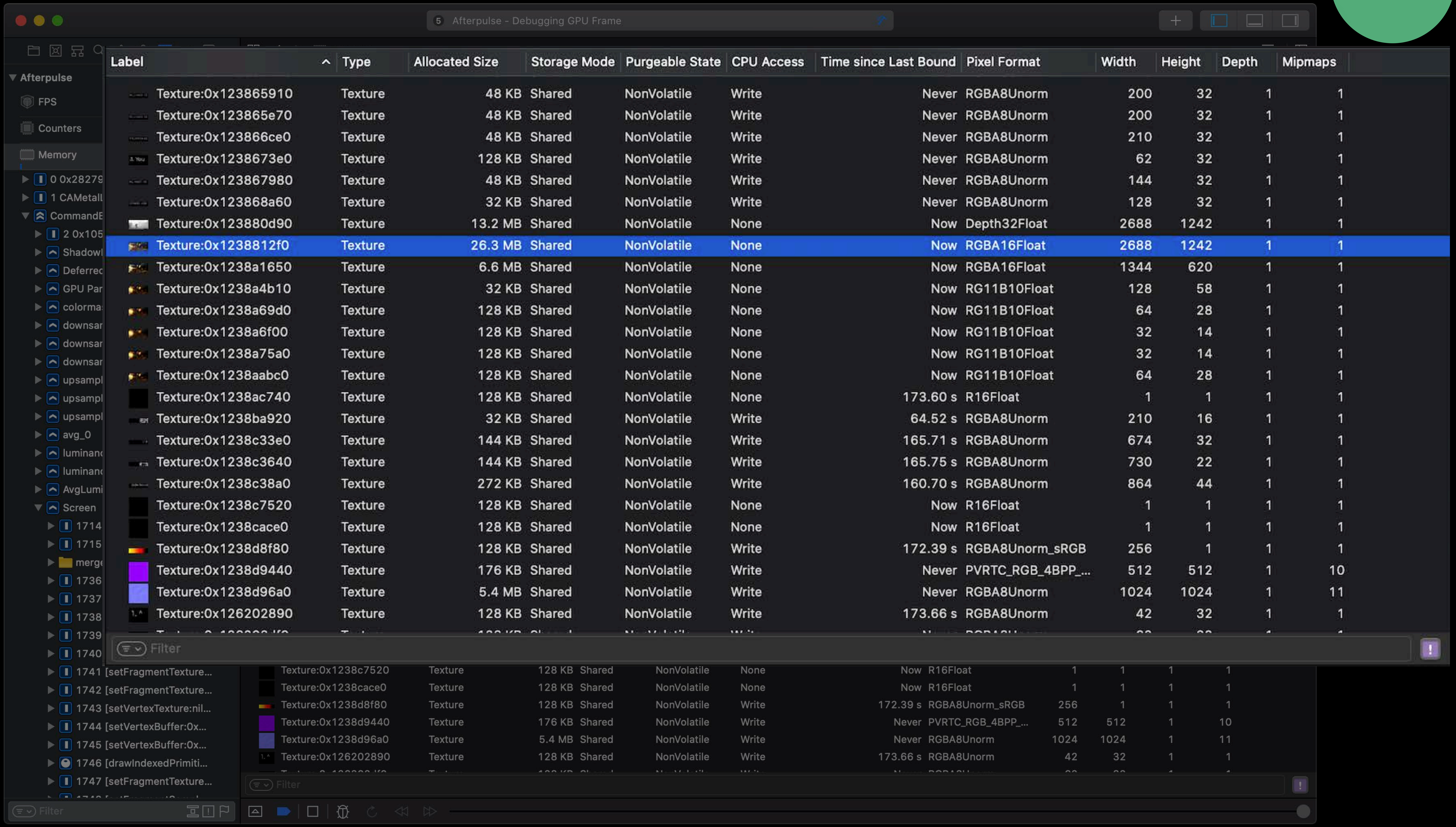
Metal Memory Viewer

NEW



Metal Memory Viewer

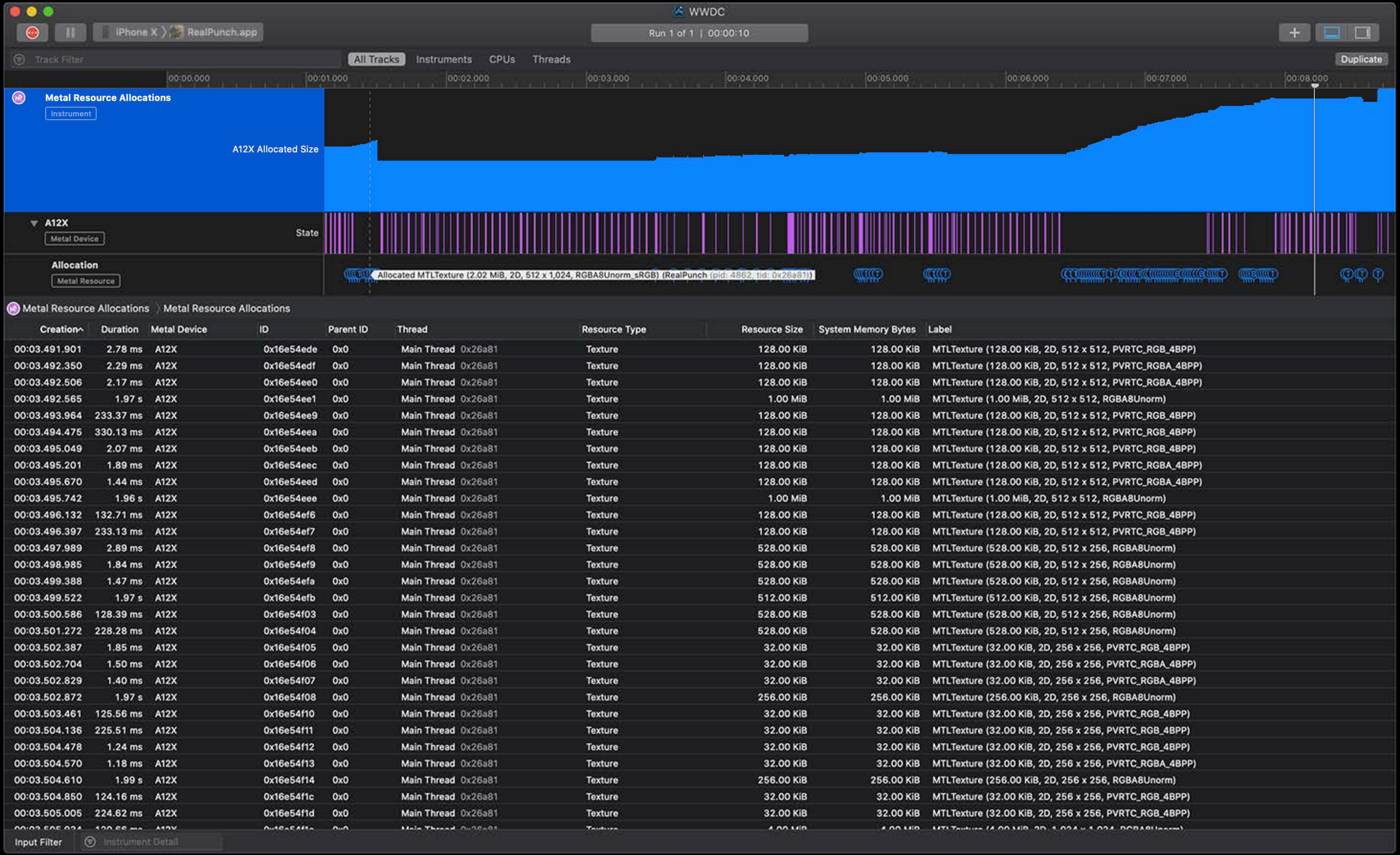
NEW



Label	Type	Allocated Size	Storage Mode	Purgeable State	CPU Access	Time since Last Bound	Pixel Format	Width	Height	Depth	Mipmaps
Texture:0x123865910	Texture	48 KB	Shared	NonVolatile	Write	Never	RGBA8Unorm	200	32	1	1
Texture:0x123865e70	Texture	48 KB	Shared	NonVolatile	Write	Never	RGBA8Unorm	200	32	1	1
Texture:0x123866ce0	Texture	48 KB	Shared	NonVolatile	Write	Never	RGBA8Unorm	210	32	1	1
Texture:0x1238673e0	Texture	128 KB	Shared	NonVolatile	Write	Never	RGBA8Unorm	62	32	1	1
Texture:0x123867980	Texture	48 KB	Shared	NonVolatile	Write	Never	RGBA8Unorm	144	32	1	1
Texture:0x123868a60	Texture	32 KB	Shared	NonVolatile	Write	Never	RGBA8Unorm	128	32	1	1
Texture:0x123880d90	Texture	13.2 MB	Shared	NonVolatile	None	Now	Depth32Float	2688	1242	1	1
Texture:0x1238812f0	Texture	26.3 MB	Shared	NonVolatile	None	Now	RGBA16Float	2688	1242	1	1
Texture:0x1238a1650	Texture	6.6 MB	Shared	NonVolatile	None	Now	RGBA16Float	1344	620	1	1
Texture:0x1238a4b10	Texture	32 KB	Shared	NonVolatile	None	Now	RG11B10Float	128	58	1	1
Texture:0x1238a69d0	Texture	128 KB	Shared	NonVolatile	None	Now	RG11B10Float	64	28	1	1
Texture:0x1238a6f00	Texture	128 KB	Shared	NonVolatile	None	Now	RG11B10Float	32	14	1	1
Texture:0x1238a75a0	Texture	128 KB	Shared	NonVolatile	None	Now	RG11B10Float	32	14	1	1
Texture:0x1238aabc0	Texture	128 KB	Shared	NonVolatile	None	Now	RG11B10Float	64	28	1	1
Texture:0x1238ac740	Texture	128 KB	Shared	NonVolatile	None	173.60 s	R16Float	1	1	1	1
Texture:0x1238ba920	Texture	32 KB	Shared	NonVolatile	Write	64.52 s	RGBA8Unorm	210	16	1	1
Texture:0x1238c33e0	Texture	144 KB	Shared	NonVolatile	Write	165.71 s	RGBA8Unorm	674	32	1	1
Texture:0x1238c3640	Texture	144 KB	Shared	NonVolatile	Write	165.75 s	RGBA8Unorm	730	22	1	1
Texture:0x1238c38a0	Texture	272 KB	Shared	NonVolatile	Write	160.70 s	RGBA8Unorm	864	44	1	1
Texture:0x1238c7520	Texture	128 KB	Shared	NonVolatile	None	Now	R16Float	1	1	1	1
Texture:0x1238cace0	Texture	128 KB	Shared	NonVolatile	None	Now	R16Float	1	1	1	1
Texture:0x1238d8f80	Texture	128 KB	Shared	NonVolatile	Write	172.39 s	RGBA8Unorm_sRGB	256	1	1	1
Texture:0x1238d9440	Texture	176 KB	Shared	NonVolatile	Write	Never	PVRTC_RGB_4BPP_...	512	512	1	10
Texture:0x1238d96a0	Texture	5.4 MB	Shared	NonVolatile	Write	Never	RGBA8Unorm	1024	1024	1	11
Texture:0x126202890	Texture	128 KB	Shared	NonVolatile	Write	173.66 s	RGBA8Unorm	42	32	1	1

Metal Resource Allocations Instrument

NEW



Creation	Duration	Metal Device	ID	Parent ID	Thread	Resource Type	Resource Size	System Memory Bytes	Label
00:03.491.901	2.78 ms	A12X	0x16e54ede	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.492.350	2.29 ms	A12X	0x16e54edf	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.492.506	2.17 ms	A12X	0x16e54ee0	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.492.565	1.97 s	A12X	0x16e54ee1	0x0	Main Thread 0x26a81	Texture	1.00 MiB	1.00 MiB	MTLTexture (1.00 MiB, 2D, 512 x 512, RGBA8Unorm)
00:03.493.964	233.37 ms	A12X	0x16e54ee9	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.494.475	330.13 ms	A12X	0x16e54eea	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.495.049	2.07 ms	A12X	0x16e54eeb	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.495.201	1.89 ms	A12X	0x16e54eec	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.495.670	1.44 ms	A12X	0x16e54eed	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.495.742	1.96 s	A12X	0x16e54eee	0x0	Main Thread 0x26a81	Texture	1.00 MiB	1.00 MiB	MTLTexture (1.00 MiB, 2D, 512 x 512, RGBA8Unorm)
00:03.496.132	132.71 ms	A12X	0x16e54ef6	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.496.397	233.13 ms	A12X	0x16e54ef7	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.497.989	2.89 ms	A12X	0x16e54ef8	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.498.985	1.84 ms	A12X	0x16e54ef9	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.499.388	1.47 ms	A12X	0x16e54efa	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.499.522	1.97 s	A12X	0x16e54efb	0x0	Main Thread 0x26a81	Texture	512.00 KiB	512.00 KiB	MTLTexture (512.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.500.586	128.39 ms	A12X	0x16e54f03	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.501.272	228.28 ms	A12X	0x16e54f04	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.502.387	1.85 ms	A12X	0x16e54f05	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.502.704	1.50 ms	A12X	0x16e54f06	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGBA_4BPP)
00:03.502.829	1.40 ms	A12X	0x16e54f07	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGBA_4BPP)
00:03.502.872	1.97 s	A12X	0x16e54f08	0x0	Main Thread 0x26a81	Texture	256.00 KiB	256.00 KiB	MTLTexture (256.00 KiB, 2D, 256 x 256, RGBA8Unorm)
00:03.503.461	125.56 ms	A12X	0x16e54f10	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.136	225.51 ms	A12X	0x16e54f11	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.478	1.24 ms	A12X	0x16e54f12	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.570	1.18 ms	A12X	0x16e54f13	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGBA_4BPP)
00:03.504.610	1.99 s	A12X	0x16e54f14	0x0	Main Thread 0x26a81	Texture	256.00 KiB	256.00 KiB	MTLTexture (256.00 KiB, 2D, 256 x 256, RGBA8Unorm)
00:03.504.850	124.16 ms	A12X	0x16e54f1c	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.505.005	224.62 ms	A12X	0x16e54f1d	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)

Metal Resource Allocations Instrument

NEW

Metal Resource Allocations Instrument

NEW

Metal Resource Allocations Instrument

NEW

WWDC

iPhone X 〉 RealPunch.app

Run 1 of 1 | 00:00:10

Metal Resource Allocations 〉 Metal Resource Allocations

Creation	Duration	Metal Device	ID	Parent ID	Thread	Resource Type	Resource Size	System Memory Bytes	Label
00:03.491.901	2.78 ms	A12X	0x16e54ede	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.492.350	2.29 ms	A12X	0x16e54edf	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.492.506	2.17 ms	A12X	0x16e54ee0	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.492.565	1.97 s	A12X	0x16e54ee1	0x0	Main Thread 0x26a81	Texture	1.00 MiB	1.00 MiB	MTLTexture (1.00 MiB, 2D, 512 x 512, RGBA8Unorm)
00:03.493.964	233.37 ms	A12X	0x16e54ee9	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.494.475	330.13 ms	A12X	0x16e54eea	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.495.049	2.07 ms	A12X	0x16e54eeb	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.495.201	1.89 ms	A12X	0x16e54eec	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.495.670	1.44 ms	A12X	0x16e54eed	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGBA_4BPP)
00:03.495.742	1.96 s	A12X	0x16e54eee	0x0	Main Thread 0x26a81	Texture	1.00 MiB	1.00 MiB	MTLTexture (1.00 MiB, 2D, 512 x 512, RGBA8Unorm)
00:03.496.132	132.71 ms	A12X	0x16e54ef6	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.496.397	233.13 ms	A12X	0x16e54ef7	0x0	Main Thread 0x26a81	Texture	128.00 KiB	128.00 KiB	MTLTexture (128.00 KiB, 2D, 512 x 512, PVRTC_RGB_4BPP)
00:03.497.989	2.89 ms	A12X	0x16e54ef8	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.498.985	1.84 ms	A12X	0x16e54ef9	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.499.388	1.47 ms	A12X	0x16e54efa	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.499.522	1.97 s	A12X	0x16e54efb	0x0	Main Thread 0x26a81	Texture	512.00 KiB	512.00 KiB	MTLTexture (512.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.500.586	128.39 ms	A12X	0x16e54f03	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.501.272	228.28 ms	A12X	0x16e54f04	0x0	Main Thread 0x26a81	Texture	528.00 KiB	528.00 KiB	MTLTexture (528.00 KiB, 2D, 512 x 256, RGBA8Unorm)
00:03.502.387	1.85 ms	A12X	0x16e54f05	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.502.704	1.50 ms	A12X	0x16e54f06	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGBA_4BPP)
00:03.502.829	1.40 ms	A12X	0x16e54f07	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGBA_4BPP)
00:03.502.872	1.97 s	A12X	0x16e54f08	0x0	Main Thread 0x26a81	Texture	256.00 KiB	256.00 KiB	MTLTexture (256.00 KiB, 2D, 256 x 256, RGBA8Unorm)
00:03.503.461	125.56 ms	A12X	0x16e54f10	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.136	225.51 ms	A12X	0x16e54f11	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.478	1.24 ms	A12X	0x16e54f12	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.570	1.18 ms	A12X	0x16e54f13	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGBA_4BPP)
00:03.504.610	1.99 s	A12X	0x16e54f14	0x0	Main Thread 0x26a81	Texture	256.00 KiB	256.00 KiB	MTLTexture (256.00 KiB, 2D, 256 x 256, RGBA8Unorm)
00:03.504.850	124.16 ms	A12X	0x16e54f1c	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.505.005	224.62 ms	A12X	0x16e54f1d	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)

00:03.504.136	225.51 ms	A12X	0x16e54f11	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.478	1.24 ms	A12X	0x16e54f12	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.504.570	1.18 ms	A12X	0x16e54f13	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGBA_4BPP)
00:03.504.610	1.99 s	A12X	0x16e54f14	0x0	Main Thread 0x26a81	Texture	256.00 KiB	256.00 KiB	MTLTexture (256.00 KiB, 2D, 256 x 256, RGBA8Unorm)
00:03.504.850	124.16 ms	A12X	0x16e54f1c	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)
00:03.505.005	224.62 ms	A12X	0x16e54f1d	0x0	Main Thread 0x26a81	Texture	32.00 KiB	32.00 KiB	MTLTexture (32.00 KiB, 2D, 256 x 256, PVRTC_RGB_4BPP)

Input Filter

Instrument Detail

Available Memory

NEW

```
#include <os/proc.h>

size_t os_proc_available_memory(void)
```

New C-based API to query available memory

Enables games to

- Stream resources more effectively
- Avoid memory spikes

Available Memory

NEW

```
#include <os/proc.h>

size_t os_proc_available_memory(void)
```

New C-based API to query available memory

Enables games to

- Stream resources more effectively
- Avoid memory spikes

On-Device GPU Capture

NEW

Programmatically trigger a GPU capture

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

To enable, add `MetalCaptureEnabled` to info.plist

```swift
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 ...
}
```

```swift
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 ...
}
```

```swift
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 ...
}
```

```swift
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 ...
}
```

```swift
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

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 extures
11. Leverage tile memory
12. Use memoryless render targets
13. Avoid loading unused assets
14. Use smaller assets
15. Simplify memory-intensive effects

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 extures
11. Leverage tile memory
12. Use memoryless render targets
13. Avoid loading unused assets
14. Use smaller assets
15. Simplify memory-intensive effects

Best Practices

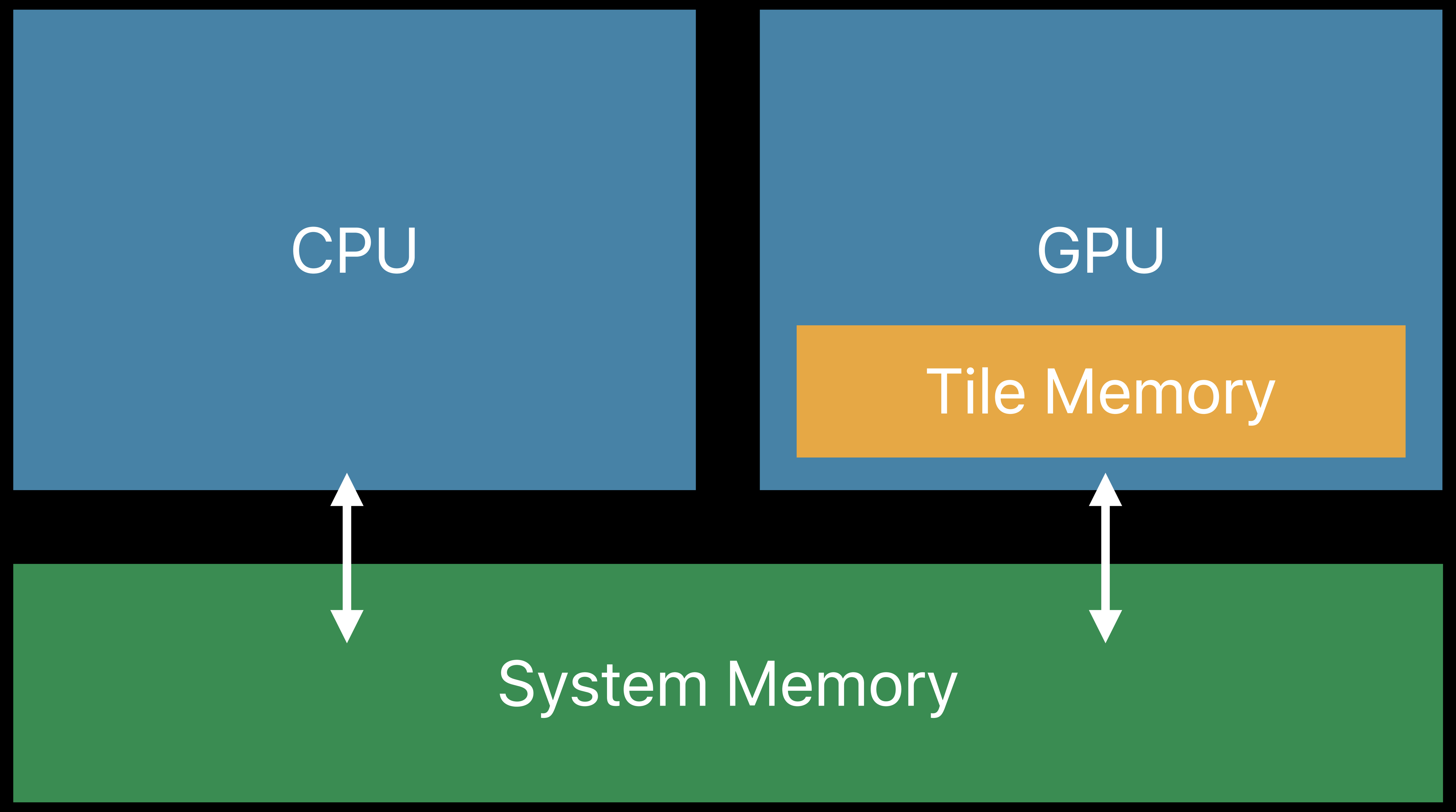
12. Use memoryless render targets

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!

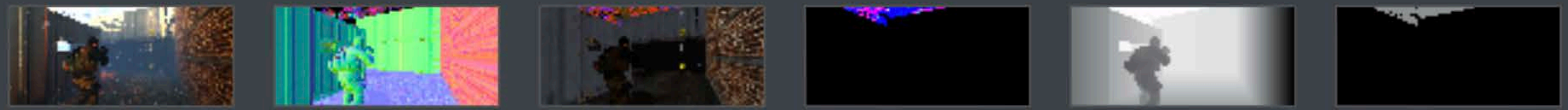


```swift
// Configure the GBuffer textures as memoryless render targets
// ...
gBufferTextureDescriptor.storageMode = .memoryless
gBufferTextureDescriptor.usage       = [ .shaderRead, .renderTarget ]

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

    renderPassDescritptor.colorAttachments[i].texture     = gBufferTextures[i]
    renderPassDescritptor.colorAttachments[i].loadAction  = .clear
    renderPassDescritptor.colorAttachments[i].storeAction = .dontCare
}
```

```swift
// 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)!

    renderPassDescritptor.colorAttachments[i].texture     = gBufferTextures[i]
    renderPassDescritptor.colorAttachments[i].loadAction  = .clear
    renderPassDescritptor.colorAttachments[i].storeAction = .dontCare
}
```

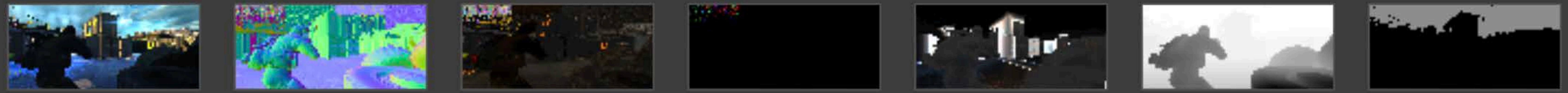
```swift
// Configure the GBuffer textures as memoryless render targets
// ...
gBufferTextureDescriptor.storageMode = .memoryless
gBufferTextureDescriptor.usage       = [ .shaderRead, .renderTarget ]

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

    renderPassDescritptor.colorAttachments[i].texture     = gBufferTextures[i]
    renderPassDescritptor.colorAttachments[i].loadAction  = .clear
    renderPassDescritptor.colorAttachments[i].storeAction = .dontCare
}
```

Color0: 0x10141d1c0	Color1: 0x1014200a0	Color2: 0x101420640	Color3: 0x10141d3d0	Depth: 0x101316060	Stencil: 0x10141f2d0



	Color0	Color1	Color2	Color3	Depth	Stencil
Pixel Format	RGBA16Float	RGBA8Unorm	RGBA8Unorm	RG11...0Float	Depth32Float	Stencil8
Dimensions	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125
Allocated Size	21.66 MiB	14.94 MiB	10.97 MiB	10.97 MiB	10.97 MiB	2.81 MiB
Load Action	Clear	DontCare	DontCare	DontCare	Clear	Clear
Store Action	Store	DontCare	DontCare	DontCare	Store	DontCare

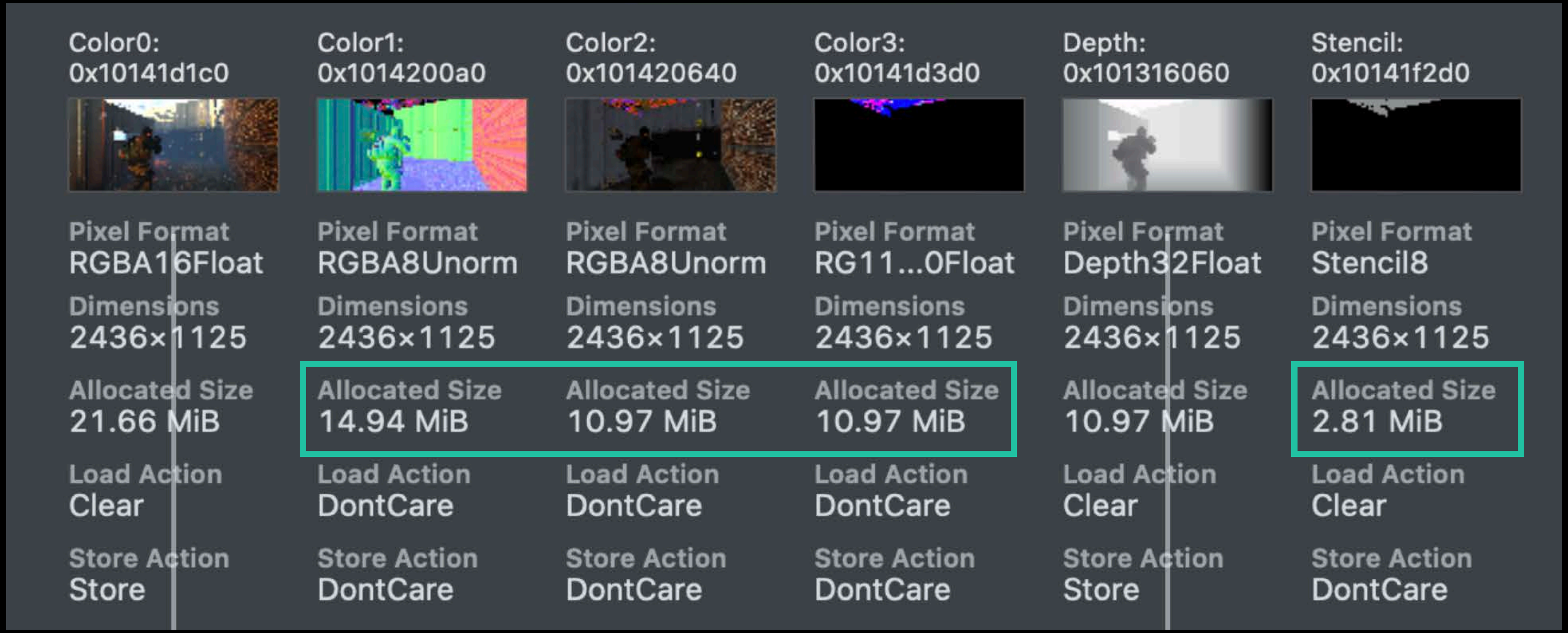
Color0: 0x11cdccd20	Color1: 0x11cdd7b90	Color2: 0x11cdd7f30	Color3: 0x11cdd82d0	Color4: 0x11cdd8670	Depth: 0x11cdcc6b0	Stencil: 0x11cdd7670

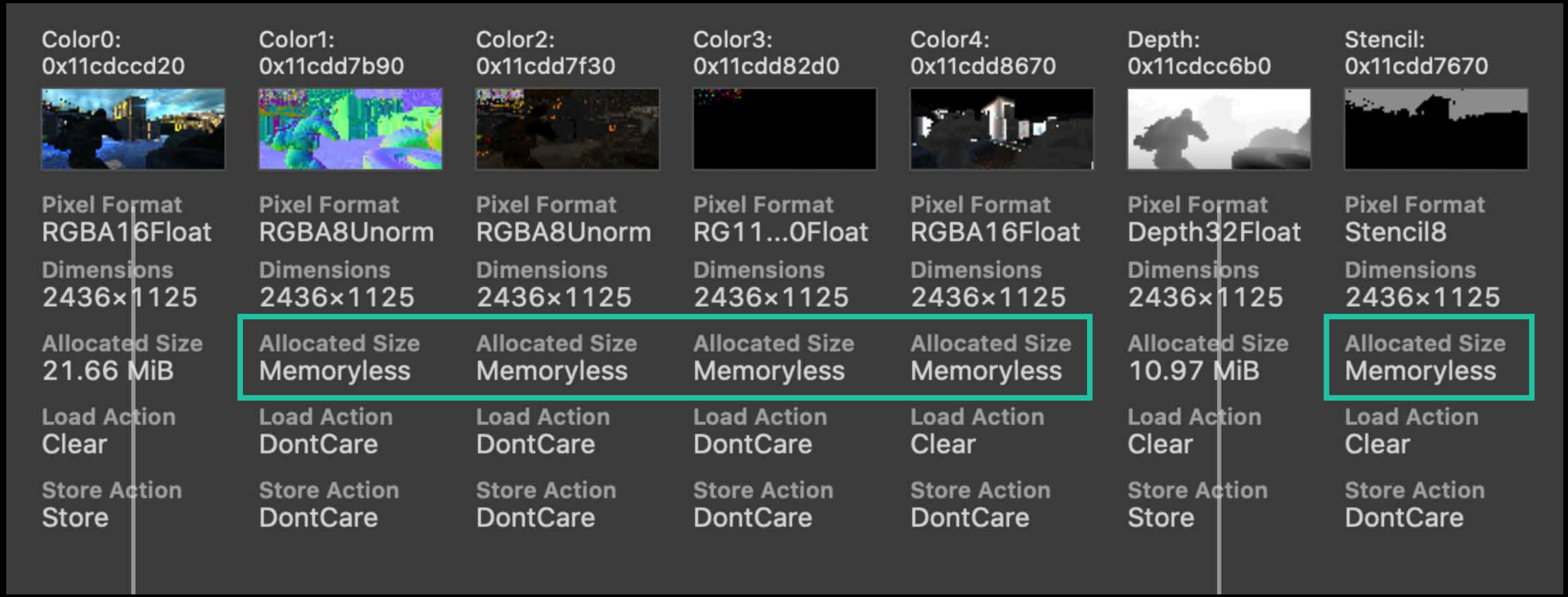


	Color0	Color1	Color2	Color3	Color4	Depth	Stencil
Pixel Format	RGBA16Float	RGBA8Unorm	RGBA8Unorm	RG11...0Float	RGBA16Float	Depth32Float	Stencil8
Dimensions	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125	2436×1125
Allocated Size	21.66 MiB	Memoryless	Memoryless	Memoryless	Memoryless	10.97 MiB	Memoryless
Load Action	Clear	DontCare	DontCare	DontCare	Clear	Clear	Clear
Store Action	Store	DontCare	DontCare	DontCare	DontCare	Store	DontCare

Color0: 0x10141d1c0	Color1: 0x1014200a0	Color2: 0x101420640	Color3: 0x10141d3d0	Depth: 0x101316060	Stencil: 0x10141f2d0
Pixel Format RGBA16Float	Pixel Format RGBA8Unorm	Pixel Format RGBA8Unorm	Pixel Format RG11...0Float	Pixel Format Depth32Float	Pixel Format Stencil8
Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125
Allocated Size 21.66 MiB	Allocated Size 14.94 MiB	Allocated Size 10.97 MiB	Allocated Size 10.97 MiB	Allocated Size 10.97 MiB	Allocated Size 2.81 MiB
Load Action Clear	Load Action DontCare	Load Action DontCare	Load Action DontCare	Load Action Clear	Load Action Clear
Store Action Store	Store Action DontCare	Store Action DontCare	Store Action DontCare	Store Action Store	Store Action DontCare

Color0: 0x11cdccd20	Color1: 0x11cdd7b90	Color2: 0x11cdd7f30	Color3: 0x11cdd82d0	Color4: 0x11cdd8670	Depth: 0x11cdcc6b0	Stencil: 0x11cdd7670
Pixel Format RGBA16Float	Pixel Format RGBA8Unorm	Pixel Format RGBA8Unorm	Pixel Format RG11...0Float	Pixel Format RGBA16Float	Pixel Format Depth32Float	Pixel Format Stencil8
Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125	Dimensions 2436×1125
Allocated Size 21.66 MiB	Allocated Size Memoryless	Allocated Size Memoryless	Allocated Size Memoryless	Allocated Size Memoryless	Allocated Size 10.97 MiB	Allocated Size Memoryless
Load Action Clear	Load Action DontCare	Load Action DontCare	Load Action DontCare	Load Action Clear	Load Action Clear	Load Action Clear
Store Action Store	Store Action DontCare	Store Action DontCare	Store Action DontCare	Store Action DontCare	Store Action Store	Store Action DontCare


Color0: 0x10141d1c0
Pixel Format RGBA16Float
Dimensions 2436×1125
Allocated Size 21.66 MiB
Load Action Clear
Store Action Store
Color1: 0x1014200a0
Pixel Format RGBA8Unorm
Dimensions 2436×1125
Allocated Size 14.94 MiB
Load Action DontCare
Store Action DontCare
Color2: 0x101420640
Pixel Format RGBA8Unorm
Dimensions 2436×1125
Allocated Size 10.97 MiB
Load Action DontCare
Store Action DontCare
Color3: 0x10141d3d0
Pixel Format RG11...0Float
Dimensions 2436×1125
Allocated Size 10.97 MiB
Load Action DontCare
Store Action DontCare
Depth: 0x101316060
Pixel Format Depth32Float
Dimensions 2436×1125
Allocated Size 10.97 MiB
Load Action Clear
Store Action Store
Stencil: 0x10141f2d0
Pixel Format Stencil8
Dimensions 2436×1125
Allocated Size 2.81 MiB
Load Action Clear
Store Action DontCare


Color0: 0x11cdccd20
Pixel Format RGBA16Float
Dimensions 2436×1125
Allocated Size 21.66 MiB
Load Action Clear
Store Action Store
Color1: 0x11cdd7b90
Pixel Format RGBA8Unorm
Dimensions 2436×1125
Allocated Size Memoryless
Load Action DontCare
Store Action DontCare
Color2: 0x11cdd7f30
Pixel Format RGBA8Unorm
Dimensions 2436×1125
Allocated Size Memoryless
Load Action DontCare
Store Action DontCare
Color3: 0x11cdd82d0
Pixel Format RG11...0Float
Dimensions 2436×1125
Allocated Size Memoryless
Load Action DontCare
Store Action DontCare
Color4: 0x11cdd8670
Pixel Format RGBA16Float
Dimensions 2436×1125
Allocated Size Memoryless
Load Action Clear
Store Action DontCare
Depth: 0x11cdcc6b0
Pixel Format Depth32Float
Dimensions 2436×1125
Allocated Size 10.97 MiB
Load Action Clear
Store Action Store
Stencil: 0x11cdd7670
Pixel Format Stencil8
Dimensions 2436×1125
Allocated Size Memoryless
Load Action Clear
Store Action DontCare

Best Practices

13. Avoid loading unused resources

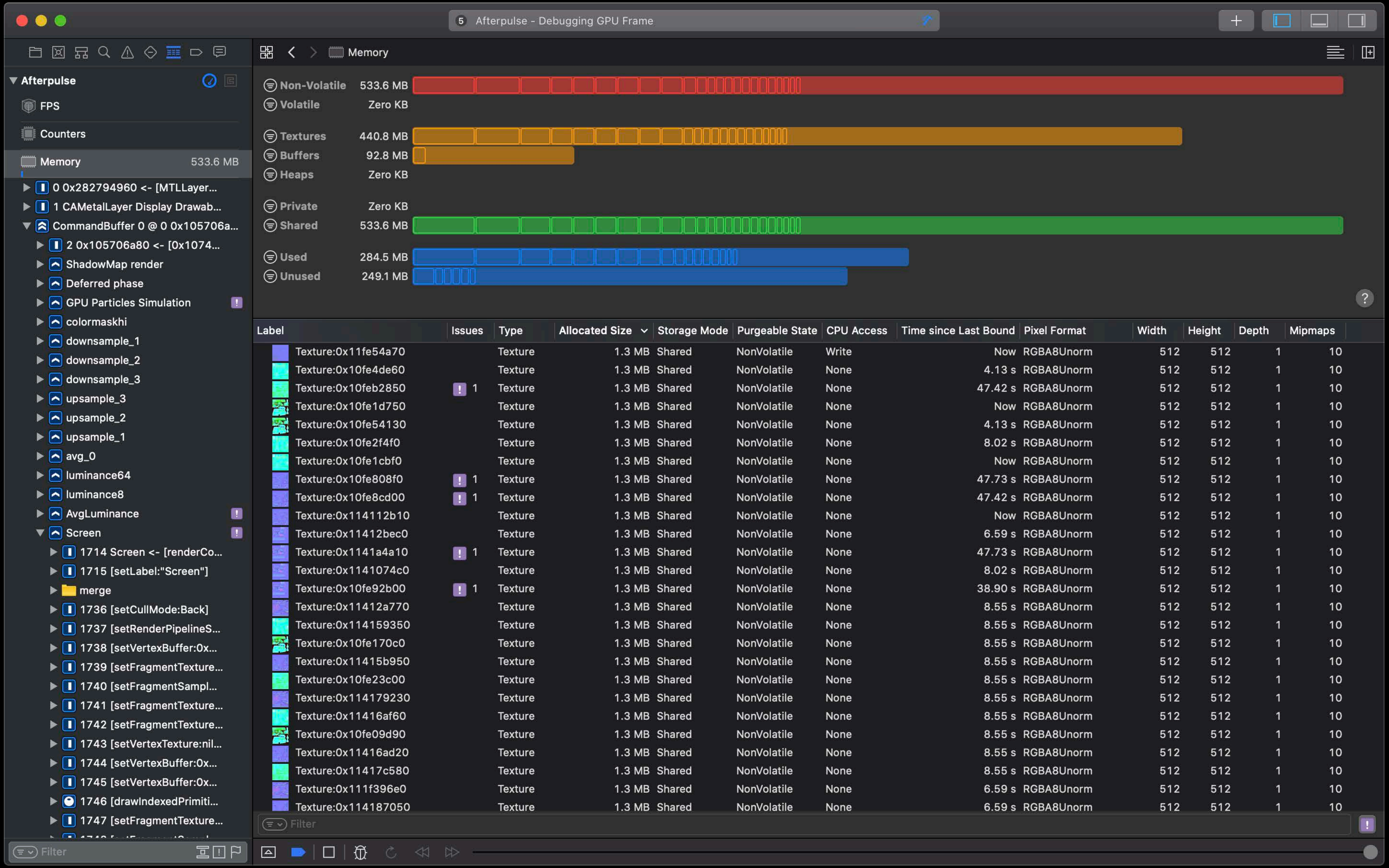
Loading all the assets into memory will increase memory footprint

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

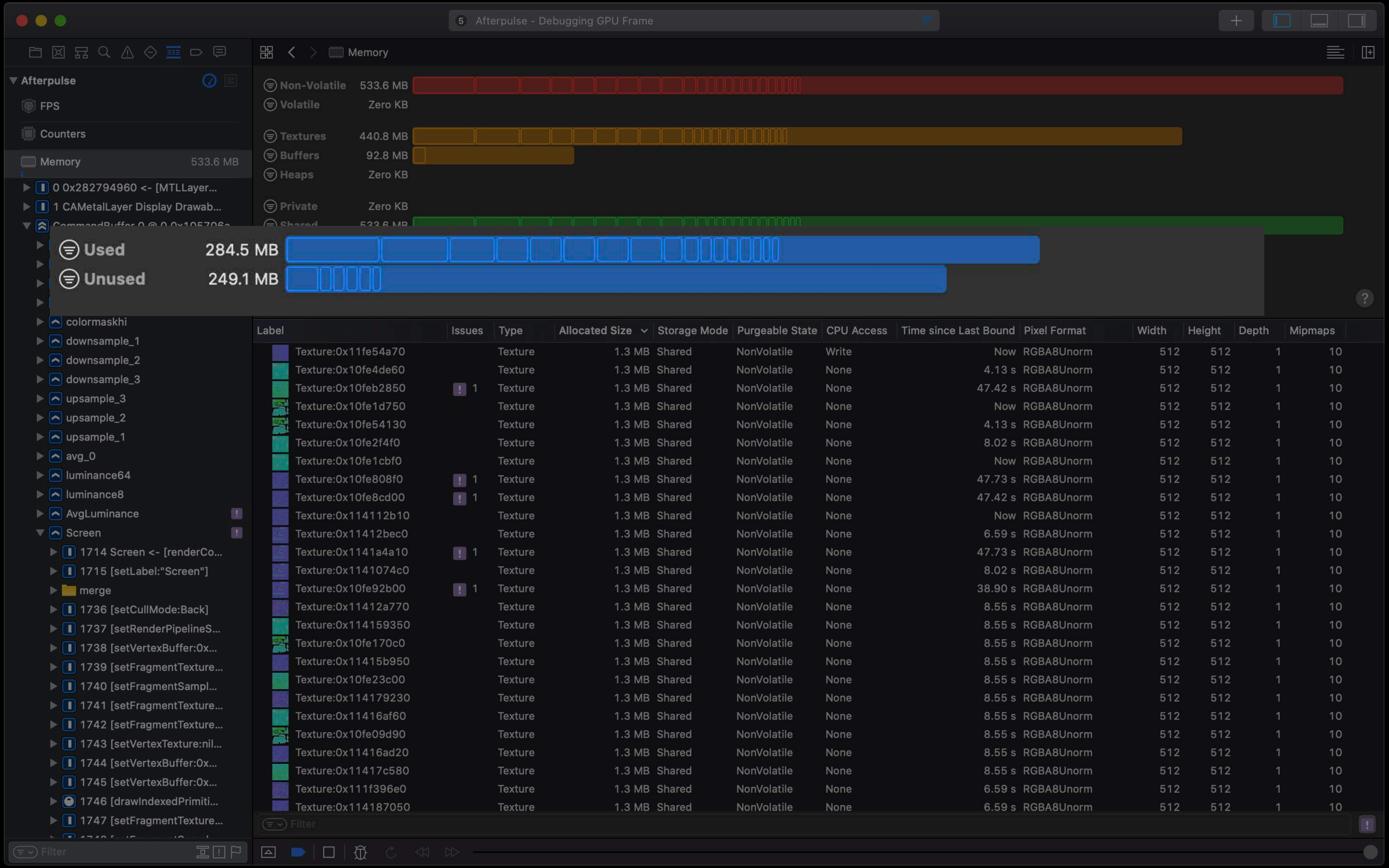
Metal Memory Viewer

NEW



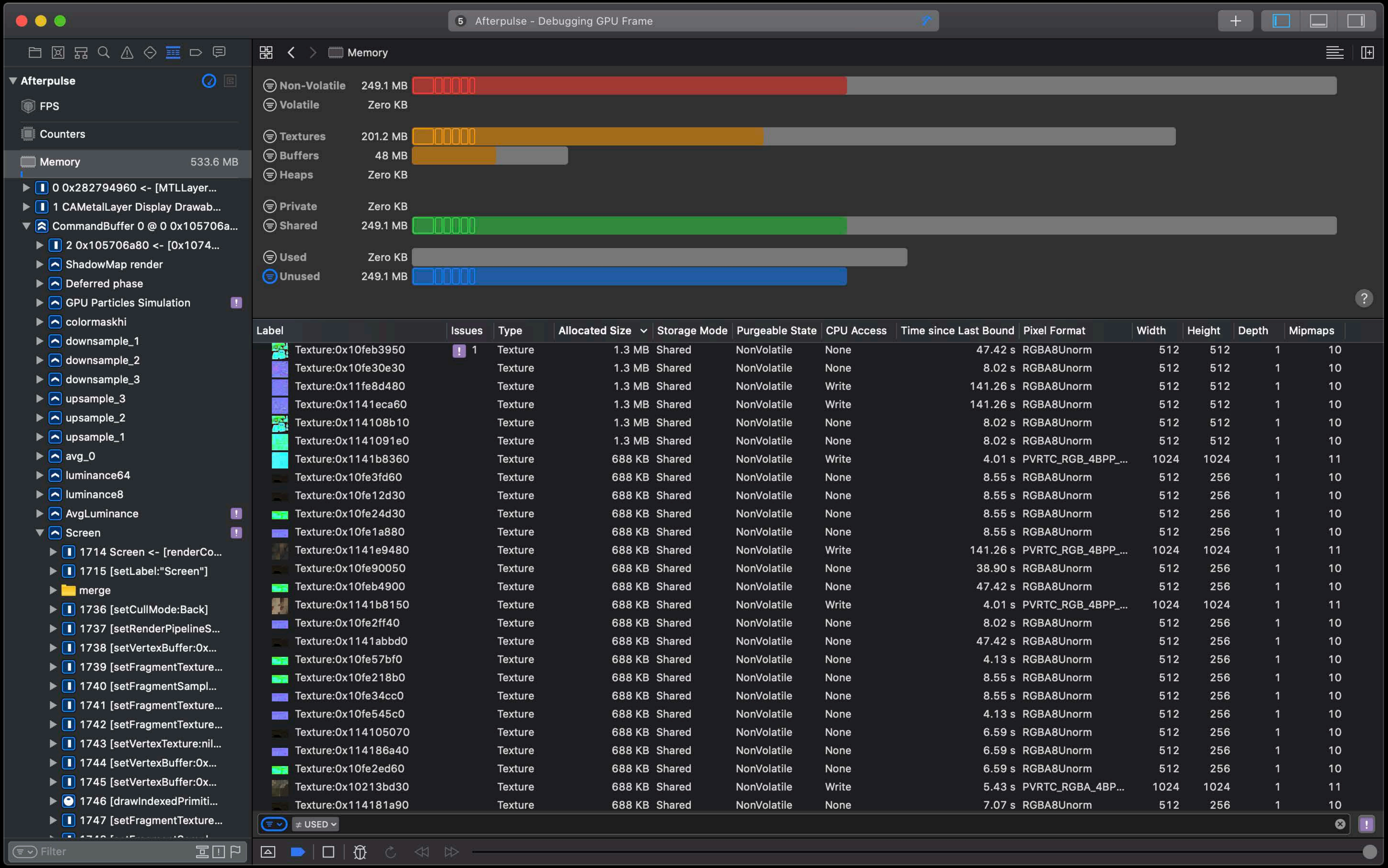
Metal Memory Viewer

NEW



Metal Memory Viewer

NEW



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

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

Metal Resource Heaps

Best Practices

16. Advanced — Use metal resource heaps

Rendering a frame may require a lot of intermediate memory

Resource heaps allow

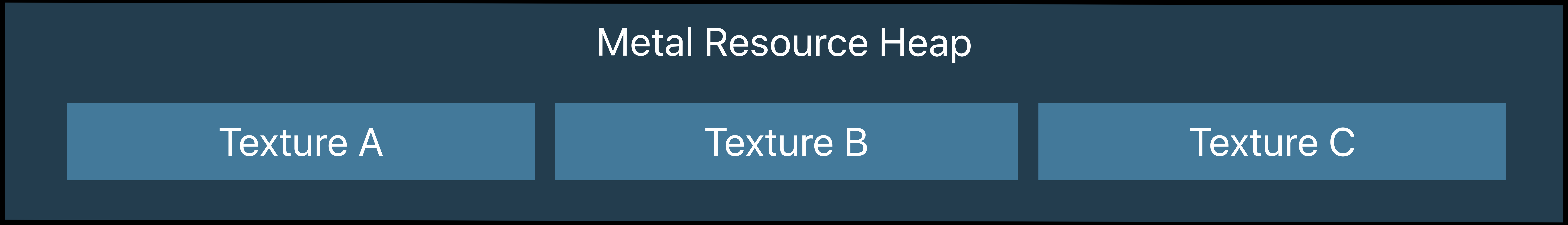
- Creation of multiple resources from single allocation
- Aliasing resources

Best practice

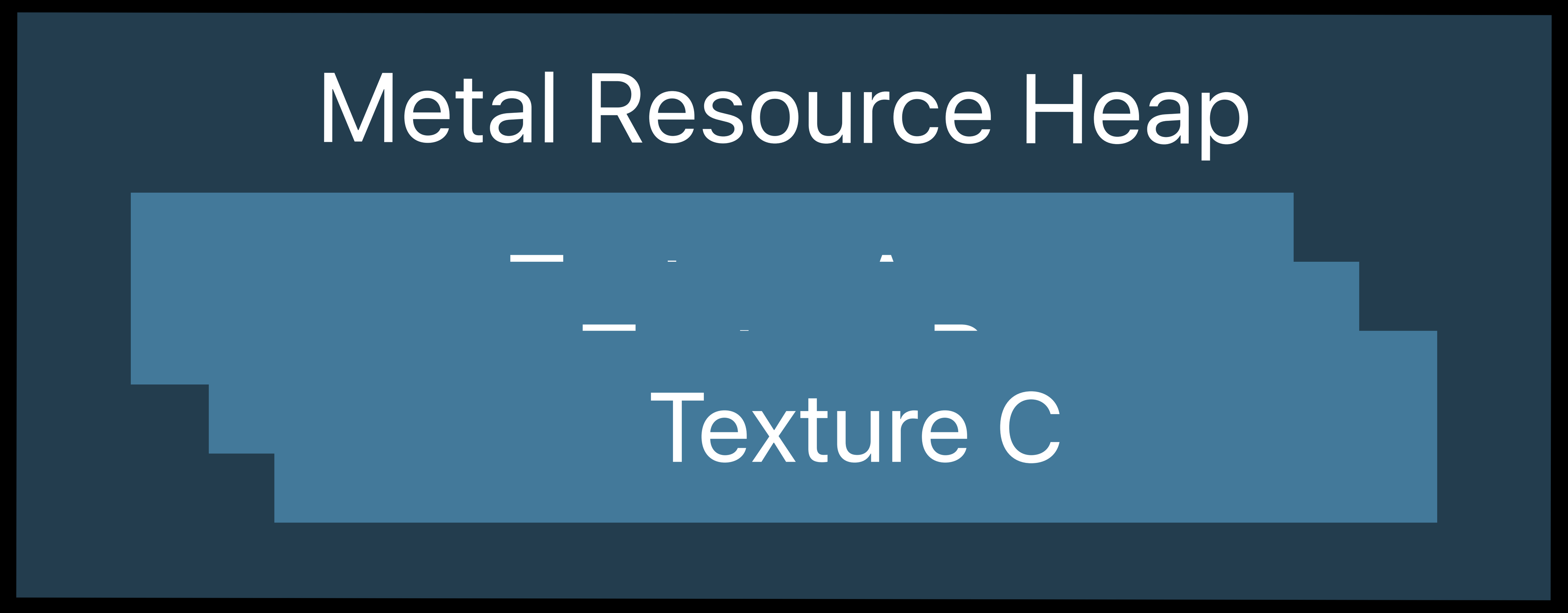
- Alias intermediate resources that have no dependencies (i.e. SSAO and DoF)

Metal Resource Heaps

Metal Resource Heaps



Metal Resource Heaps



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

```swift
// 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 ...
```

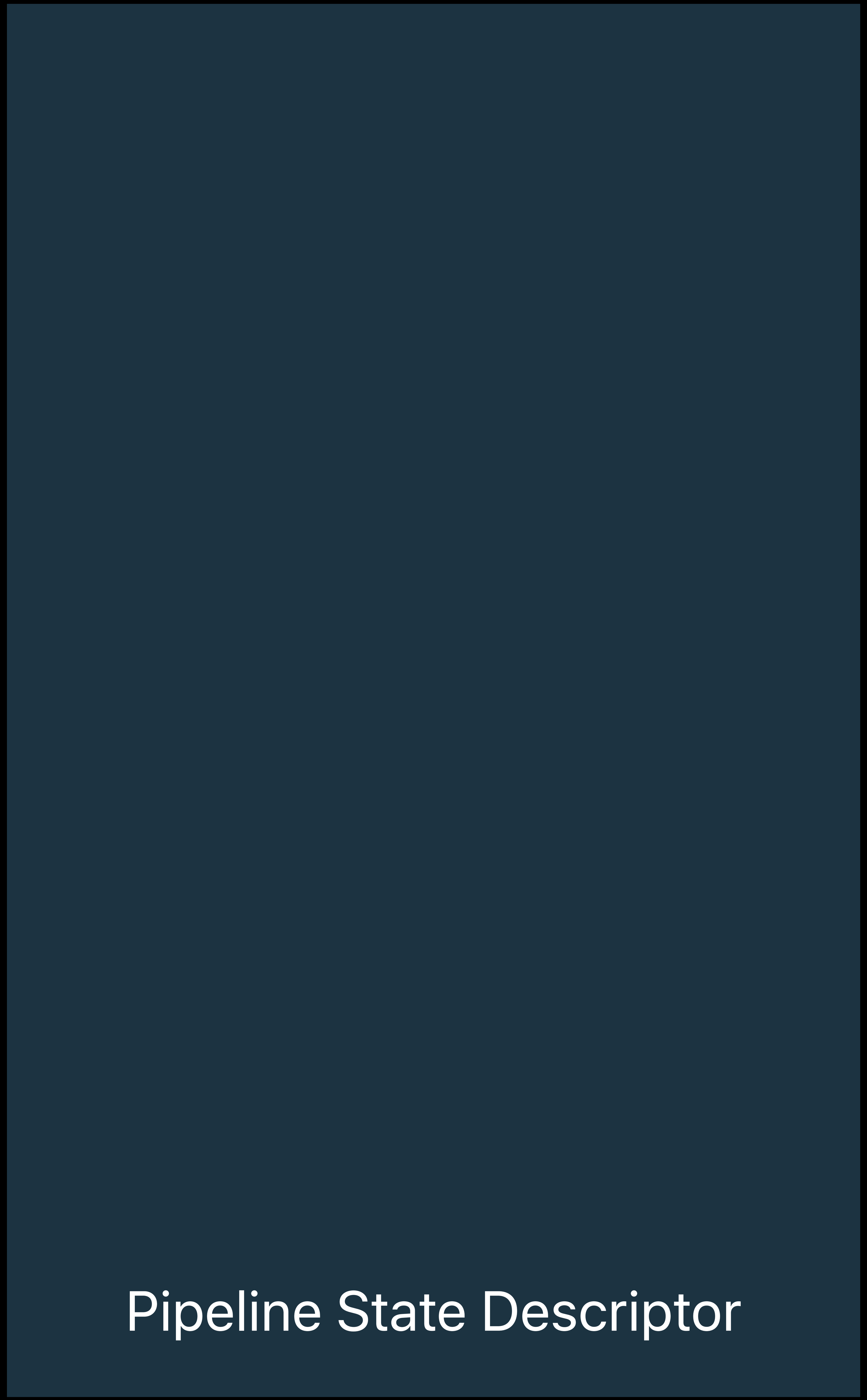
```swift
// Mark all the textures in the cache as volatile

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

// ...

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

// ... Use texture ...
```

```swift
// 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 ...
```

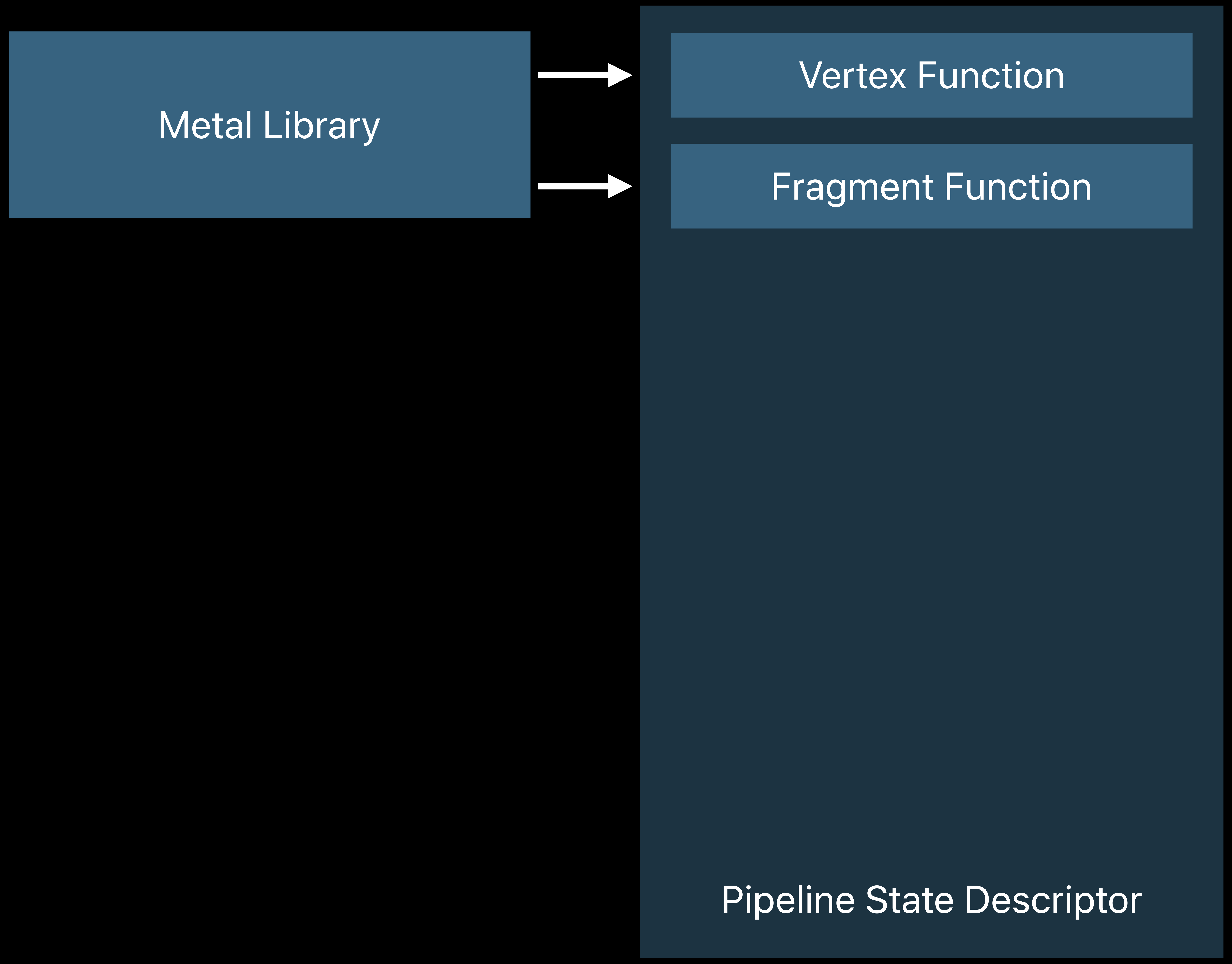
```swift
// Mark all the textures in the cache as volatile

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

// ...

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

// ... Use texture ...
```

```swift
// 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 ...
```

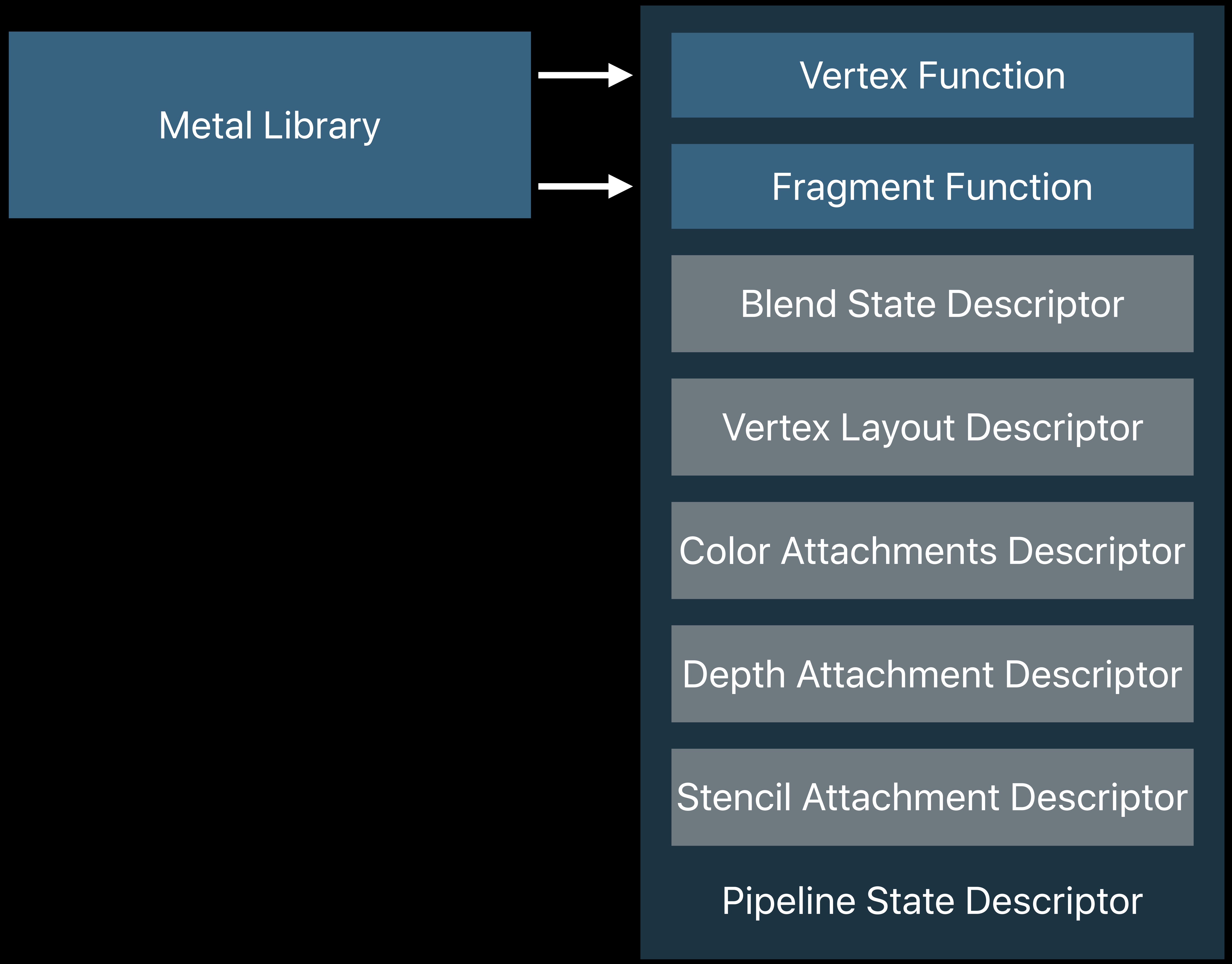
Pipeline State Objects (PSOs)



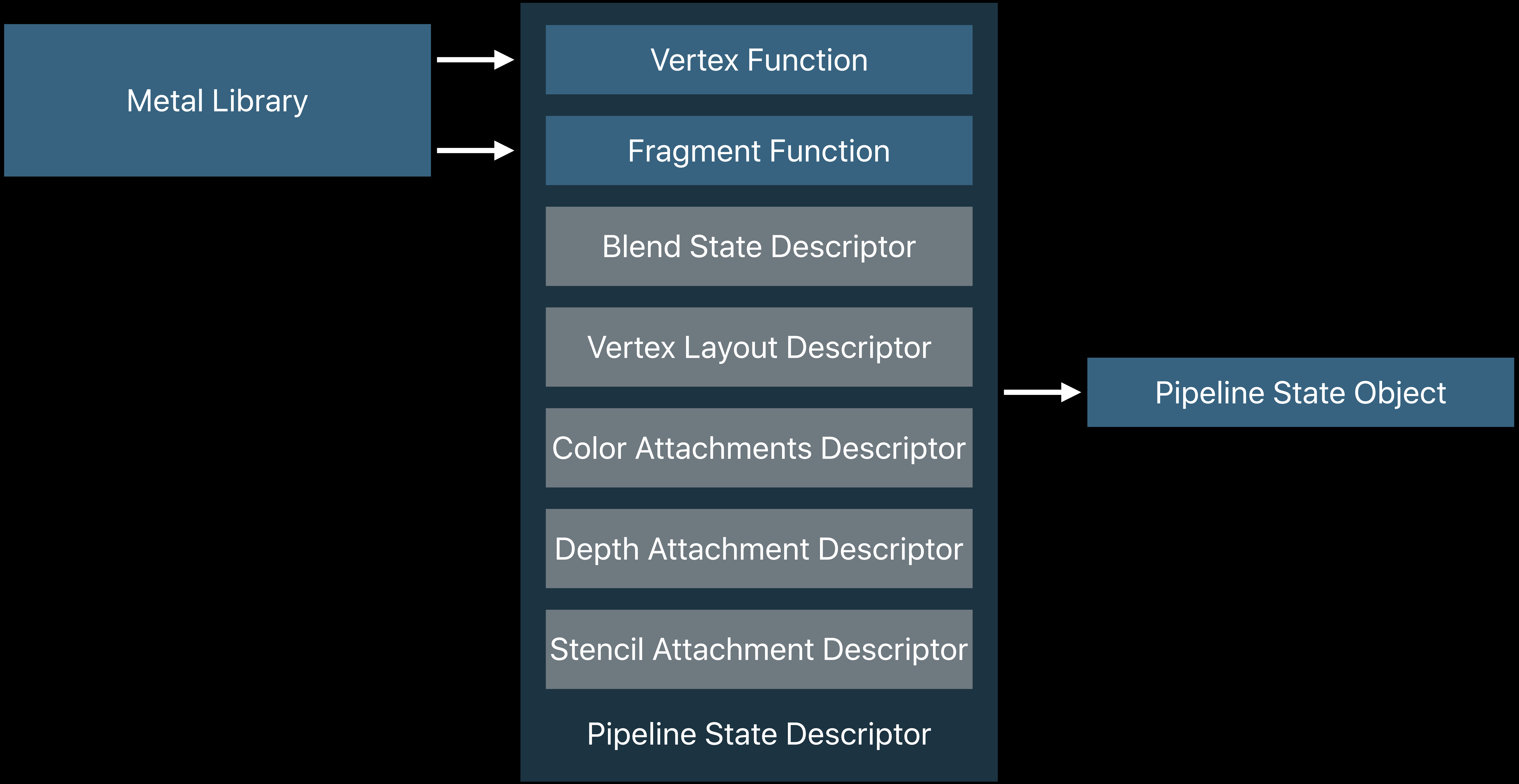
Pipeline State Objects (PSOs)



Pipeline State Objects (PSOs)



Pipeline State Objects (PSOs)



Best Practices

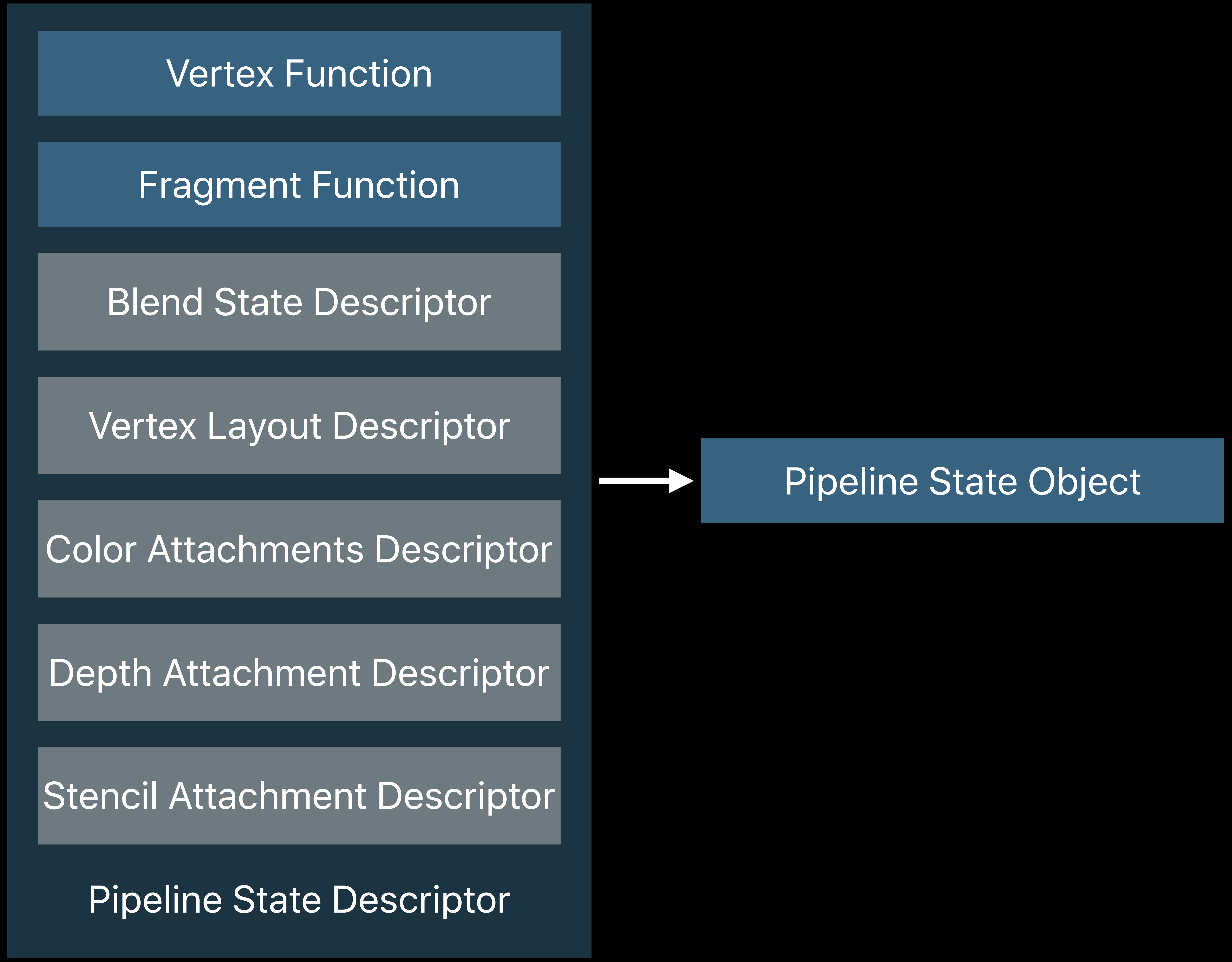
18. Advanced — Manage the Metal PSOs

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

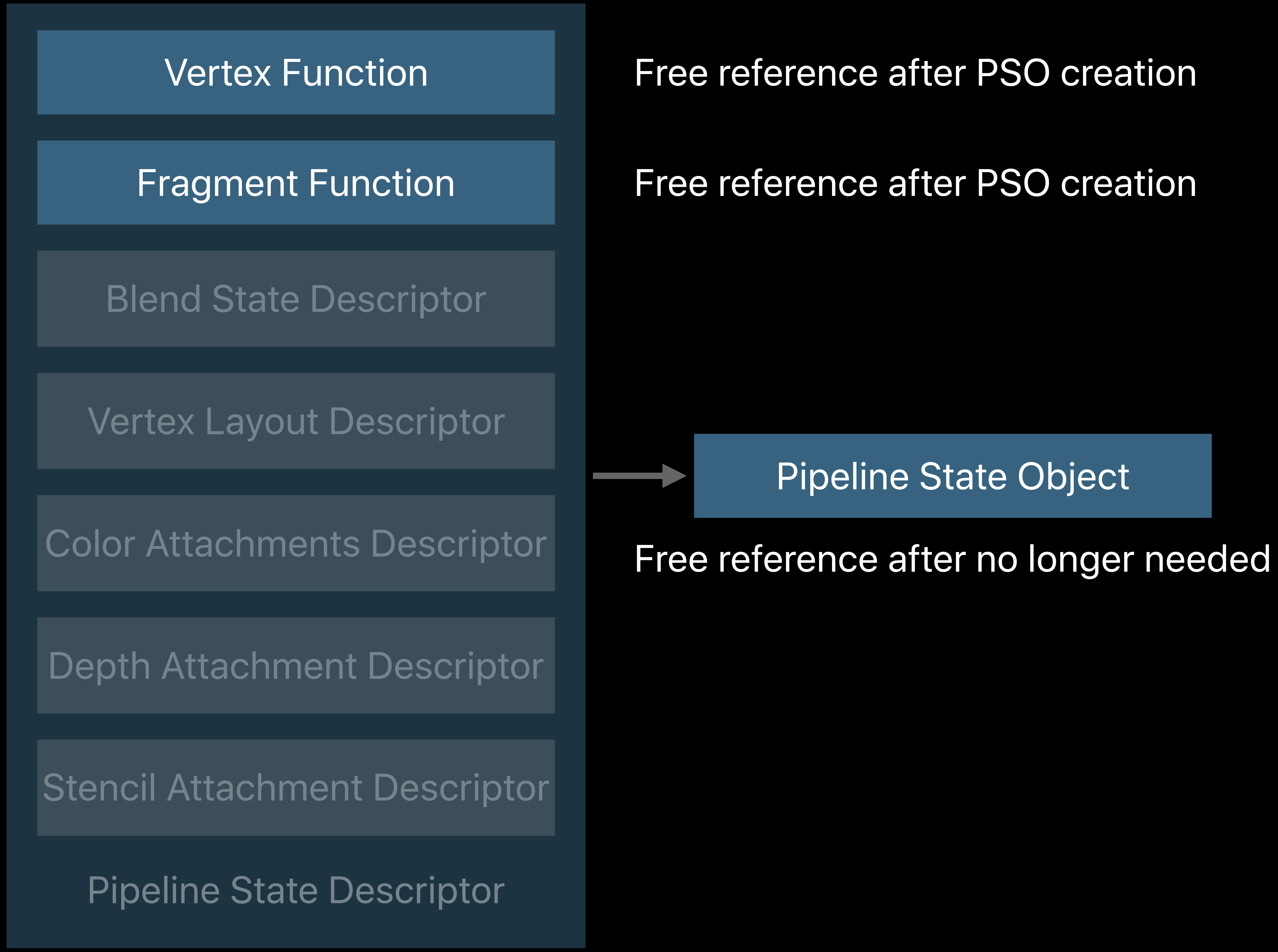
Best practice

- 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)



Pipeline State Objects (PSOs)



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

Metal Optimization Lab	Wednesday 3:00

WWDC19