Article

Simplifying GPU resource management with residency sets

Organize your resources into groups and influence when they become accessible to the GPU.

Overview

Metal apps typically create resources, such as textures and buffers, so that their shaders can work with data as they run on the GPU. These resources need to be in memory that's accessible to the GPU, or *resident*, so the shaders can access their data.

A *residency set* is one way you tell Metal which resources your app needs to make resident. You do this by creating `MTLResidencySet` instances, managing which resource *allocations* they contain, and attaching them to command buffers or command queues. Resource allocation types conform to the `MTLAllocation` protocol, including `MTLBuffer`, `MTLTexture`, and `MTLHeap`.

The other way to tell Metal which resources it needs to make resident is by calling a command encoder's methods. However, these methods can impact an app's runtime performance because each call incurs some CPU overhead. Additionally, Metal makes those resources resident right after your app commits the command buffer, which can delay when the GPU starts working on it. This overhead adds up as the number of resources increases, especially in apps that use many resources for each frame, such as games.

Residency sets help you mitigate these performance issues and delays. With a residency set, your app can:

- Add multiple allocations with less CPU overhead than with a command encoder's methods
- Make its allocations resident at the same time
- Request that Metal make its resources resident ahead of time
- Keep allocations resident indefinitely

- Remove all allocations, or a selection of them, which Metal marks as candidates that it can make nonresident, if necessary

You can attach each residency set to a command buffer or an entire command queue. Attaching a residency set to a command buffer removes the need to tell each of its command encoders which resources they need to use. Similarly, attaching a residency set to a command queue removes the need to attach that residency set to each of its command buffers.

Make a residency set and add allocations to it

Create a residency set by configuring an `MTLResidencySetDescriptor` instance and passing it to the `makeResidencySet(descriptor:)` method of an `MTLDevice`.

Swift Objective-C

```
let setDescriptor = MTLResidencySetDescriptor()
setDescriptor.label = "Primary residency set"
setDescriptor.initialCapacity = 42

let residencySet = try device.makeResidencySet(descriptor: setDescriptor)
```

Add an individual allocation to the `MTLResidencySet` instance by calling its `addAllocation(_:)` method, or add multiple allocations with its `addAllocations(_:)` method.

Swift Objective-C

```
let residencySet = try device.makeResidencySet(descriptor: setDescriptor)

residencySet.addAllocation(buffer0)
residencySet.addAllocation(buffer1)
residencySet.addAllocation(texture0)
residencySet.addAllocation(texture1)
residencySet.addAllocation(heap)

let allocations = [buffer2,
                   texture2,
                   argumentBufferHeap,
                   textureHeap]

residencySet.addAllocations(allocations)
```

A residency set handles redundant allocations by ignoring instances that already have an entry in the set.

> **Important**
>
> Adding a resource allocation that originates from an `MTLHeap` to a residency set makes that entire heap resident.

Finalize and apply the pending changes to the residency set by calling its `commit()` method.

Swift Objective-C

```
residencySet.commit()
```

See `MTLResidencySet` for information about working with residency sets, including:

- Inspecting current allocations
- Adding and removing allocations over time
- Accounting for resource hazards

Attach a residency set to a command buffer

Connect an `MTLCommandBuffer` instance to a residency set's resource allocations by attaching the set to the command buffer with the `useResidencySet(_:)` or `useResidencySets(_:)` method. Every command buffer can maintain a list of up to 32 different residency sets.

Swift Objective-C

```
commandBuffer.useResidencySet(residencySet)
```

Metal makes the allocations in the set resident before the GPU runs the passes in the command buffer. This includes all resources that come from an `MTLHeap` allocation that's in the residency set.

You don't need to call the following methods for any allocation in a residency set that you associate with the command buffer:

MTLRenderCommandEncoder	MTLComputeCommandEncoder
useResource(_:usage:stages:)	useResource(_:usage:)
useResources(_:usage:stages:)	useResources(_:usage:)
useHeap(_:stages:)	useHeap(_:)
useHeaps(_:stages:)	useHeaps(_:)

Attaching a residency set to a command buffer takes less CPU runtime and overhead than calling these methods for each encoder within a command buffer.

Attach a residency set to a command queue and its command buffers

Connect an MTLCommandQueue instance to a residency set's resource allocations by attaching the set to the queue with its addResidencySet(_:) or addResidencySets(_:) method. Every command queue can maintain a list of up to 32 different residency sets.

Swift Objective-C

```
commandQueue.addResidencySet(residencySet)
```

When your app calls a command buffer's commit() method, Metal automatically attaches the owning queue's current residency sets to the command buffer.

Tip

Attach a residency set to a command queue for resources the GPU needs access to frequently, or for the lifetime of your app.

Attaching a residency set to a command queue is more efficient than attaching that residency set to multiple command buffers from that queue.

Detach a residency set from a command queue

When your command queue doesn't need the resources of a residency set, disconnect it from the queue by calling the removeResidencySet(_:) or removeResidencySets(_:) method.

Swift   Objective-C

```
commandQueue.removeResidencySet(residencySet)
```

The residency set remains attached to any of the queue's command buffers already in-flight with a status equal to `MTLCommandBufferStatus.committed` or `MTLCommandBufferStatus.scheduled`.

Request residency ahead of time

To make allocations in a residency set resident (for allocations that aren't already resident), the Metal framework needs to do some work on the CPU. By default, Metal does this work when you call the `commit()` method of the first command buffer that's using the residency set. Making the allocations resident at this time can delay the graphics driver from submitting the command buffer to the GPU.

To help minimize the time between committing a command buffer and when the GPU starts working on it, ask Metal to do the work beforehand. You do this by calling a residency set's `requestResidency()` method.

Swift   Objective-C

```
residencySet.requestResidency()
```

Call this method at any time before you commit the first command buffer that relies on the allocations in the residency set. This can be any noncritical moment when your app can afford the CPU time the framework needs to prepare the applicable allocations for residency. For example, you can call this method at launch or during an app state change.

> **Note**
>
> The `requestResidency()` method may postpone some of the necessary steps to make allocations resident in scenarios where other apps have competing memory needs.

Conclude residency for the resources

When your app no longer needs a residency set's allocations to be accessible to the GPU, call the `endResidency()` method, which effectively releases them.

Swift Objective-C

```
residencySet.endResidency()
```

The method tells Metal that it can reuse the memory backing that residency set’s allocations for your app’s other residency sets, or for another app.

See Also

Residency sets

`protocol MTLResidencySet`

A collection of resource allocations that can move in and out of resident memory.

`class MTLResidencySetDescriptor`

A configuration that customizes the behavior for a residency set.