Onscreen presentation

Show the output from a GPU's rendering pass to the user in your app.

Overview

A texture contains visual data that you may want to display onscreen, such as a filtered image or a frame of animation in a game. To display a texture on a device's screen, you need to create or acquire a drawable texture from Core Animation.

In Metal, a *drawable* is a texture that bridges the display subsystem within Core Animation to Metal. You can use a drawable as the output of a render pass and then present it to a display with its `MTLDrawable` protocol.

> **Note**
>
> You can't present a texture you directly create in Metal unless you copy its content to a drawable using a blit pass (see Blit passes).

To display content on a device's screen, add one of these to your Metal app:

- A `CAMetalLayer` instance within a custom view class
- An `MTKView` instance

With a `CAMetalLayer` instance, your app can get drawable instances by calling its `nextDrawable()` method. Each drawable conforms to the `CAMetalDrawable` protocol, which has a `texture` property that a render pass can use as its output. See Creating a custom Metal view for more information.

Alternatively, your app can use an `MTKView` and its `currentDrawable` property. This MetalKit class provides a default implementation of a Metal-aware view that uses an underlying `CAMetalLayer` instance. A MetalKit view provides a quick and easy way to present your app's content, but gives you less control than using a `CAMetalLayer` directly. See Using Metal to draw a view's contents for more information.

Whichever mechanism you choose, pay close attention to how your app handles drawables. Each drawable comes from a limited and reusable resource pool. A drawable may not always be available when your app requests one. When that happens, Core Animation blocks the calling thread until a drawable becomes available — usually at the display's next refresh interval.

Topics

Presenting with core animation

{} Creating a custom Metal view

Implement a lightweight view for Metal rendering that's customized to your app's needs.

{} Reading pixel data from a drawable texture

Access texture data from the CPU by copying it to a buffer.

{} Achieving smooth frame rates with a Metal display link

Pace rendering with minimal input latency while providing essential information to the operating system for power-efficient rendering, thermal mitigation, and the scheduling of sustainable workloads.

`class CAMetalLayer`

A Core Animation layer that Metal can render into, typically displayed onscreen.

`protocol CAMetalDrawable : MTLDrawable`

A Metal drawable associated with a Core Animation layer.

Presenting with MetalKit

{} Using Metal to draw a view's contents

Create a MetalKit view and a render pass to draw the view's contents.

`@MainActor class MTKView`

A specialized view that creates, configures, and displays Metal objects.

`protocol MTKViewDelegate : NSObjectProtocol`

Methods for responding to a MetalKit view's drawing and resizing events.

See Also

Presentation

- Managing your game window for Metal in macOS

  Set up a window and view for optimally displaying your Metal content.

- Managing your Metal app window in iPadOS

  Set up a window that handles dynamically resizing your Metal content.

- Adapting your game interface for smaller screens

  Make text legible on all devices the player chooses to run your game on.

- HDR content

  Take advantage of high dynamic range to present more vibrant colors in your apps and games.