Class

MTKView

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

iOS 9.0+ | iPadOS 9.0+ | Mac Catalyst 13.1+ | macOS 10.11+ | tvOS 9.0+ | visionOS 1.0+

```
@MainActor
class MTKView
```

Overview

The `MTKView` class provides a default implementation of a Metal-aware view that you can use to render graphics using Metal and display them onscreen. When asked, the view provides a `MTLRenderPassDescriptor` object that points at a texture for you to render new contents into. Optionally, an `MTKView` can create depth and stencil textures for you and any intermediate textures needed for antialiasing. The view uses a `CAMetalLayer` to manage the Metal drawable objects.

The view requires a `MTLDevice` object to manage the Metal objects it creates for you. You must set the `device` property and, optionally, modify the view's drawable properties before drawing.

Configuring the Drawing Behavior

The MTKView class supports three drawing modes:

- Timed updates: The view redraws its contents based on an internal timer. In this case, which is the default behavior, both `isPaused` and `enableSetNeedsDisplay` are set to `false`. Use this mode for games and other animated content that's regularly updated.
- Draw notifications: The view redraws itself when something invalidates its contents, usually because of a call to `setNeedsDisplay()` or some other view-related behavior. In this case, set `isPaused` and `enableSetNeedsDisplay` to `true`. Use this mode for apps with a more

traditional workflow, where updates happen when data changes, but not on a regular timed interval.

- Explicit drawing: The view redraws its contents only when you explicitly call the `draw()` method. In this case, set `isPaused` to `true` and `enableSetNeedsDisplay` to `false`. Use this mode to create your own custom workflow.

Drawing the View's Contents

Regardless of drawing mode, when the view needs to update its contents, it calls the `draw(_:)` method when that method has been overridden by a subclass, or `draw(in:)` on the view's delegate if the subclass doesn't override it. You should either subclass `MTKView` or provide a delegate, but not both.

In your drawing method, you obtain a render pass descriptor from the view, render into it, and then present the associated drawable.

Obtaining a Drawable from a MetalKit View

Each `MTKView` is backed by a `CAMetalLayer`. In your renderer, implement the `MTKViewDelegate` protocol to interact with a MetalKit view. Call the MetalKit view's `currentRenderPassDescriptor` property to obtain a render pass descriptor configured for the current frame:

Swift Objective-C

```
// BEGIN encoding your onscreen render pass.
// Obtain a render pass descriptor generated from the drawable's texture.
// (`currentRenderPassDescriptor` implicitly obtains the current drawable.)
// If there's a valid render pass descriptor, use it to render to the current drawab
if let onscreenDescriptor = view.currentRenderPassDescriptor
```

When you read this property, Core Animation implicitly obtains a drawable for the current frame and stores it in the `currentDrawable` property. It then configures a render pass descriptor to draw into that drawable, including any depth, stencil, and antialiasing textures as necessary. The view configures this render pass using the default store and load actions. You can adjust the descriptor further before using it to create a `MTLRenderCommandEncoder`.

Obtain drawables as late as possible; preferably, immediately before encoding your onscreen render pass.

Registering the Drawable's Presentation

After rendering the contents, you must present the drawable to update the view's contents. The most convenient way to present the content is to call the `present(_:)` method on the command buffer. Then, call the `commit()` method to submit the command buffer to a GPU:

Swift | Objective-C

```
if let onscreenDescriptor = view.currentRenderPassDescriptor,
let onscreenCommandEncoder = onscreenCommandBuffer.makeRenderCommandEncoder(descript
    /* Set render state and resources.
       ...
     */
    /* Issue draw calls.
       ...
     */
    onscreenCommandEncoder.endEncoding()
    // END encoding your onscreen render pass.

    // Register the drawable's presentation.
    if let currentDrawable = view.currentDrawable {
        onscreenCommandBuffer.present(currentDrawable)
    }
}

// Finalize your onscreen CPU work and commit the command buffer to a GPU.
onscreenCommandBuffer.commit()
```

When a command queue schedules a command buffer for execution, the drawable tracks all render or write requests on itself in that command buffer. The operating system doesn't present the drawable onscreen until the commands have finished executing. By asking the command buffer to present the drawable, you guarantee that presentation happens after the command queue has scheduled this command buffer. Don't wait for the command buffer to finish executing before registering the drawable's presentation.

Tip

For better performance, only retrieve the render pass descriptor when you're ready to render the contents, and hold onto it and the related drawable object as little as possible. Release it as soon as you finish with it. For more information, see `CAMetalLayer`.

Topics

Creating a View

`init(coder: NSCoder)`

Initializes a view from data in a given unarchiver.

`init(frame: CGRect, device: (any MTLDevice)?)`

Initializes a view with the specified frame rectangle and Metal device.

Configuring the Delegate

`var delegate: (any MTKViewDelegate)?`

The view's delegate.

Configuring the Metal Device

`var device: (any MTLDevice)?`

The device object the view uses to create its Metal objects.

`var preferredDevice: (any MTLDevice)?`

The device object that the system recommends using for this view.

Configuring the Color Render Target

`var colorPixelFormat: MTLPixelFormat`

The color pixel format for the current drawable's texture.

`var colorspace: CGColorSpace?`

The color space of the rendered content.

`var framebufferOnly: Bool`

A Boolean value that determines whether the drawable's textures are used only for rendering.

`var drawableSize: CGSize`

The current size of drawable textures.

`var preferredDrawableSize: CGSize`

The recommended dimensions of the drawable.

`var autoResizeDrawable: Bool`

A Boolean value that controls whether to resize the drawable as the view changes size.

`var clearColor: MTLClearColor`

The color to use to clear the color target when creating a render pass descriptor.

Configuring the Render Target Properties

`var depthStencilPixelFormat: MTLPixelFormat`

The format used to generate the `depthStencilTexture` object.

`var depthStencilAttachmentTextureUsage: MTLTextureUsage`

The texture usage characteristics that the view uses when creating the depth and stencil textures.

`var clearDepth: Double`

The depth value to use to clear the depth target when creating a render pass descriptor.

`var clearStencil: UInt32`

The stencil value to use to clear the stencil target when creating a render pass descriptor.

Configuring Multisampling

`var sampleCount: Int`

The sample count used to generate the `multisampleColorTexture` object.

`var multisampleColorAttachmentTextureUsage: MTLTextureUsage`

The texture usage characteristics that the view uses when creating multisample textures.

Retrieving Render Target Information

`var currentRenderPassDescriptor: MTLRenderPassDescriptor?`

A render pass descriptor to draw into the current drawable.

`var currentDrawable: (any CAMetalDrawable)?`

The drawable to use for the current frame.

`var depthStencilTexture: (any MTLTexture)?`

A packed depth and stencil texture associated with the current drawable object's texture.

`var depthStencilStorageMode: MTLStorageMode`

The storage mode that the packed depth and stencil texture use.

`var multisampleColorTexture: (any MTLTexture)?`

The multisample color sample texture to render into.

Configuring Drawing Behavior

`var preferredFramesPerSecond: Int`

The rate at which the view redraws its contents.

`var isPaused: Bool`

A Boolean value that indicates whether the draw loop is paused.

`var enableSetNeedsDisplay: Bool`

A Boolean value that indicates whether the view responds to `setNeedsDisplay()`.

`func draw()`

Redraws the view's contents immediately.

`var presentsWithTransaction: Bool`

A Boolean value that determines whether the view presents its content using a Core Animation transaction.

Releasing Memory

`func releaseDrawables()`

Releases the `depthStencilTexture` and `multisampleColorTexture` objects.

Instance Properties

`var currentMTL4RenderPassDescriptor: MTL4RenderPassDescriptor?`

Relationships

Inherits From

`NSView`, `UIView`

Conforms To

CALayerDelegate
CVarArg
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSAccessibilityElementProtocol
NSAccessibilityProtocol
NSAnimatablePropertyContainer
NSAppearanceCustomization
NSCoding
NSDraggingDestination
NSObjectProtocol
NSStandardKeyBindingResponding
NSTouchBarProvider
NSUserActivityRestoring
NSUserInterfaceItemIdentification
UIAccessibilityIdentification
UIActivityItemsConfigurationProviding
UIAppearance
UIAppearanceContainer
UICoordinateSpace
UIDynamicItem
UIFocusEnvironment
UIFocusItem
UIFocusItemContainer
UILargeContentViewerItem
UIPasteConfigurationSupporting
UIPopoverPresentationControllerSourceItem
UIResponderStandardEditActions
UITraitChangeObservable
UITraitEnvironment
UIUserActivityRestoring

See Also

View Management

protocol MTKViewDelegate

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