Protocol

MTLTexture

A resource that holds formatted image data.

iOS 8.0+ | iPadOS 8.0+ | Mac Catalyst 13.1+ | macOS 10.11+ | tvOS | visionOS 1.0+

```
protocol MTLTexture : MTLResource
```

Mentioned in

- Understanding the Metal 4 core API
- Improving CPU performance by using argument buffers
- Setting resource storage modes
- Synchronizing a managed resource in macOS
- Choosing a resource storage mode for Apple GPUs

Overview

Don't implement this protocol yourself; instead, use one of the following methods to create an `MTLTexture` instance:

- Create an `MTLTextureDescriptor` instance to describe the texture's properties and then call the `makeTexture(descriptor:)` method of the `MTLDevice` protocol to create the texture.
- To create a texture that uses an existing `IOSurface` to hold the texture data, create an `MTLTextureDescriptor` instance to describe the image data in the surface. Call the `makeTexture(descriptor:iosurface:plane:)` method to create the texture.
- To create a texture that reinterprets another texture's data as if it has a different format, call one of the following texture methods:

    - `makeTextureView(pixelFormat:)`
    - `makeTextureView(pixelFormat:textureType:levels:slices:)` (Swift)
    - `newTextureViewWithPixelFormat:textureType:levels:slices:` (Objective-C)

  You need to choose a pixel format for the new texture compatible with the source texture's pixel format. The new texture shares the same storage allocation as the source texture. If you make changes to the new texture, the source texture reflects those changes, and vice versa.

- To create a texture that uses an `MTLBuffer` instance's contents to hold pixel data, create an `MTLTextureDescriptor` instance to describe the texture's properties. Then call the `makeTexture(descriptor:offset:bytesPerRow:)` method on the buffer instance. The new texture instance shares the storage allocation of the source buffer instance. If you make changes to the texture, the buffer reflects those changes, and vice versa.

After you create an `MTLTexture` instance, most of its characteristics, such as its size, type, and pixel format are all immutable. Only the texture's pixel data is mutable.

To copy pixel data from system memory into the texture, call `replace(region:mipmapLevel:slice:withBytes:bytesPerRow:bytesPerImage:)` or `replace(region:mipmapLevel:withBytes:bytesPerRow:)`.

To copy pixel data back to system memory, call `getBytes(_:bytesPerRow:bytesPerImage:from:mipmapLevel:slice:)` or `getBytes(_:bytesPerRow:from:mipmapLevel:)`.

Topics

Copying data into a texture image

```
func replace(region: MTLRegion, mipmapLevel: Int, slice: Int, withBytes: UnsafeRawPointer, bytesPerRow: Int, bytesPerImage: Int)
```

Copies pixel data into a section of a texture slice.

Required

```
func replace(region: MTLRegion, mipmapLevel: Int, withBytes: UnsafeRawPointer, bytesPerRow: Int)
```

Copies a block of pixels into a section of texture slice 0.

Required

Copying data from a texture image

```
func getBytes(UnsafeMutableRawPointer, bytesPerRow: Int, bytesPerImage: Int, from: MTLRegion, mipmapLevel: Int, slice: Int)
```

Copies pixel data from the texture to a buffer in system memory.

Required

```
func getBytes(UnsafeMutableRawPointer, bytesPerRow: Int, from:
MTLRegion, mipmapLevel: Int)
```

Copies pixel data from the first slice of the texture to a buffer in system memory.

Required

Creating textures by reinterpreting existing texture data

```
func makeTextureView(pixelFormat: MTLPixelFormat) -> (any MTLTexture)?
```

Creates a new view of the texture, reinterpreting its data using a different pixel format.

Required

```
func makeTextureView(pixelFormat: MTLPixelFormat, textureType:
MTLTextureType, levels: Range<Int>, slices: Range<Int>) -> (any
MTLTexture)?
```

Creates a new view of the texture, reinterpreting a subset of its data using a different type and pixel format.

```
func makeTextureView(pixelFormat: MTLPixelFormat, textureType:
MTLTextureType, levels: Range<Int>, slices: Range<Int>, swizzle:
MTLTextureSwizzleChannels) -> (any MTLTexture)?
```

Creates a new view of the texture, reinterpreting a subset of its data using a different type, pixel format, and swizzle pattern.

Querying texture attributes

```
var textureType: MTLTextureType
```

The dimension and arrangement of the texture image data.

Required

```
var pixelFormat: MTLPixelFormat
```

The format of pixels in the texture.

Required

```
var width: Int
```

The width of the texture image for the base level mipmap, in pixels.

Required

```
var height: Int
```

The height of the texture image for the base level mipmap, in pixels.

Required

`var depth: Int`

The depth of the texture image for the base level mipmap, in pixels.

Required

`var mipmapLevelCount: Int`

The number of mipmap levels in the texture.

Required

`var arrayLength: Int`

The number of slices in the texture array.

Required

`var sampleCount: Int`

The number of samples in each pixel.

Required

`var isFramebufferOnly: Bool`

A Boolean value that indicates whether the texture can only be used as a render target.

Required

`var usage: MTLTextureUsage`

Options that determine how you can use the texture.

Required

`var allowGPUOptimizedContents: Bool`

A Boolean value indicating whether the GPU is allowed to adjust the contents of the texture to improve GPU performance.

Required

`var isShareable: Bool`

A Boolean indicating whether this texture can be shared with other processes.

Required

`var swizzle: MTLTextureSwizzleChannels`

The pattern that the GPU applies to pixels when you read or sample pixels from the texture.

Required

`enum MTLTextureType`

The dimension of each image, including whether multiple images are arranged into an array or a cube.

`struct MTLTextureUsage`

An enumeration for the various options that determine how you can use a texture.

Getting information about the IOSurface the texture was created from

`var iosurface: IOSurfaceRef?`

A reference to the underlying surface instance for the texture, if applicable.

Required

`var iosurfacePlane: Int`

The number of a plane within the underlying surface instance for the texture, if applicable.

Required

Getting information about ancestor resources

`var parent: (any MTLTexture)?`

The parent texture used to create this texture, if any.

Required

`var parentRelativeLevel: Int`

The base level of the parent texture used to create this texture.

Required

`var parentRelativeSlice: Int`

The base slice of the parent texture used to create this texture.

Required

`var buffer: (any MTLBuffer)?`

The source buffer used to create this texture, if any.

Required

`var bufferOffset: Int`

The offset in the source buffer where the texture’s data comes from.

Required

`var bufferBytesPerRow: Int`

The number of bytes in each row of the texture’s source buffer.

Required

~~`var rootResource: (any MTLResource)?`~~

The resource that owns the storage for this texture.

Required

Deprecated

Creating a shared texture handle

`func makeSharedTextureHandle() -> MTLSharedTextureHandle?`

Creates a new texture handle from a shareable texture.

Required

Creating views of textures on other GPUs

`func makeRemoteTextureView(any MTLDevice) -> (any MTLTexture)?`

Creates a remote texture view for another GPU in the same peer group.

Required

`var remoteStorageTexture: (any MTLTexture)?`

The texture on another GPU that the texture was created from, if any.

Required

Querying sparse properties

`var isSparse: Bool`

A Boolean value that indicates whether this is a sparse texture.

`var firstMipmapInTail: Int`

The index of the first mipmap in the tail.

`var tailSizeInBytes: Int`

The size of the sparse texture tail, in bytes.

Instance Properties

`var compressionType: MTLTextureCompressionType`

Required

`var gpuResourceID: MTLResourceID`

Required

`var sparseTextureTier: MTLTextureSparseTier`

Required

Instance Methods

```
func newTextureView(with: MTLTextureViewDescriptor) -> (any MTLTexture
)?
```

Required

Relationships

Inherits From

`MTLAllocation`, `MTLResource`, `NSObjectProtocol`

See Also

Texture basics

Understanding color-renderable pixel format sizes

Know the size limits of color render targets in Apple GPUs based on the target’s pixel format.

Optimizing texture data

Optimize a texture’s data to improve GPU or CPU access.

`enum MTLTextureCompressionType`

`class MTLTextureDescriptor`

An instance that you use to configure new Metal texture instances.

`class MTKTextureLoader`

An object that creates textures from existing data in common image formats.

`class MTLSharedTextureHandle`

A texture handle that can be shared across process address space boundaries.

`enum MTLPixelFormat`

The data formats that describe the organization and characteristics of individual pixels in a texture.