Structure

LayerRenderer.Frame

A type that provides access to the timing information and data types you need to render a single frame of content.

macOS 26.0+ | visionOS 1.0+

```
struct Frame
```

Overview

A frame represents a snapshot of your app's content at a single moment in time. In your rendering loop, you render your content into static images many times a second to create the illusion of animation. A `LayerRenderer.Frame` provides the Metal textures and information you need to render one of those images.

During each iteration of your app's rendering loop, call `queryNextFrame()` to retrieve the next frame to render. The system manages a finite number of frames and recycles them to maximize efficiency. You typically draw only one frame at a time, starting work on each new frame only after completing the previous frame.

Separate the work you do for each frame into two phases: the update phase and the encode phase. Use the update phase to perform tasks that don't depend on the current device pose. For example, you might update your app's data structures to reflect recent interactions with your content. Use the submission phase to retrieve the current pose and render the frame's content using that information. Each frame provides a `LayerRenderer.Drawable` type with access to the specific textures and rendering details for that frame.

Topics

Getting timing information

`func predictTiming() -> LayerRenderer.Frame.Timing?`

Computes and returns the predicted timing information for the frame.

`struct Timing`

A type that stores information about a frame's encoding, rendering, and presentation deadlines.

Reporting frame update times

`func startUpdate()`

Notifies Compositor Services that you started updating the app-specific content for the frame.

`func endUpdate()`

Notifies Compositor Services that you finished updating the app-specific content you need to render the frame.

Getting the drawable environment

`func queryDrawables() -> [LayerRenderer.Drawable]`

Returns the array of drawables expected to be used for the given frame. These drawables each have textures, transforms and timing information for drawing the frame.

~~`func queryDrawable() -> LayerRenderer.Drawable?`~~

Retrieves the frame's drawable, which contains the textures and drawing environment for the frame.

Deprecated

Reporting frame submission times

`func startSubmission()`

Notifies Compositor Services that you're ready to generate the Metal commands to render the specified frame.

`func endSubmission()`

Notifies Compositor Services that you finished generating the GPU commands to render the specified frame.

Getting frame-related details

```
var frameIndex: LayerFrameIndex
```

The sequential index number of a frame.

```
typealias LayerFrameIndex
```

A frame index in the layer's timeline.

```
typealias CompositorFrameIndex
```

The sequential index for a frame in the compositor's timeline.

Creating a frame

```
init()
```

Creates an uninitialized frame.

Instance Methods

```
func binocularFrustumMatrix(convention: AxisDirectionConvention, increaseTangents: SIMD4<Float>, depthRange: SIMD2<Float>) -> matrix_float4x4
```

```
func binocularFrustumMatrixForDrawableTarget(drawableTarget: LayerRenderer.Drawable.Target, convention: AxisDirectionConvention, increaseTangents: SIMD4<Float>, depthRange: SIMD2<Float>) -> matrix_float4x4
```

Returns the transform which can be used for binocular frustum culling. A matrix to convert between the device coordinate space to normalized device coordinate space. This should be acquired between starting and submitting a frame. Renderer should not utilize this transform for actual rendering output.

```
func drawableTargetViewCount(target: LayerRenderer.Drawable.Target) -> Int
```

Returns the number of view in the drawable target.

```
func monocularFrustumMatrix(convention: AxisDirectionConvention, viewIndex: Int, increaseTangents: SIMD4<Float>, depthRange: SIMD2<Float>) -> matrix_float4x4
```

```
func monocularFrustumMatrixForDrawableTarget(drawableTarget: LayerRenderer.Drawable.Target, convention: AxisDirectionConvention, viewIndex: Int, increaseTangents: SIMD4<Float>, depthRange: SIMD2<Float>) -> matrix_float4x4
```

Returns the transform which can be used for monocular frustum culling for given view. A matrix to convert between the device coordinate space to normalized device coordinate space This should be acquired between starting and submitting a frame. Renderer should not utilize this transform for actual rendering output.

Relationships

Conforms To

`BitwiseCopyable`

See Also

Render-loop setup

`class LayerRenderer`

A type that provides the Metal types and timing information you need to draw your content.