Article

Controlling Metal rendering immersion level

Enable flexible immersive rendering by supporting progressive immersion when rendering your Metal content.

Overview

Immersion is the degeree to which your rendered content from your app replaces the viewing area. Compositor Services supports full and mixed immersion styles by default. People using your app can control the immersion level by rotating the Digital Crown on Apple Vision Pro. For some content, people may prefer progressive immersion, where the viewing area is only partially replaced by rendered content, for viewing comfort. This helps them retain the context of their environment when they're viewing content with complex scenes with movement.

Enable progressive immersion

To support progressive immersion requires some additional adoption from your app provide a mask that represents the progressive immersion level. When you attach the mask to the same render encoder where your content renders the system provides performant, single pass rendering. The mask also allows the system to avoid rendering content outside of the masked area for additional performance savings. The system provides support for smoothing and fading the edge of the progressive portal into your surroundings.

Set the possible immersion styles to ensure your immersive space supports progressive immersion.

```
ImmersiveSpace(id: "MyImmersiveSpace") {
    CompositorLayer(configuration: MyConfiguration()) { @MainActor layerRenderer in
        Renderer.startRenderLoop(layerRenderer)
```

```
    }
}.immersionStyle(selection: $immersionStyle, in: .progressive, .full)
```

After allowing both progressive and full immersion styles, your app can switch between them.

Configure the compositor layer

After your app supports progressive immersion, configure the compositor layer. Check that the layer capabilities support your preferred stencil format with `drawableRenderContextSupportedStencilformats`. Set the `drawableRenderContextStencilFormat` property for the stencil format to the layer configuration. Configure the `drawableRenderContextRasterSampleCount` with the appropriate sample count depending on your app's use of multisample antialiasing:

```
    MyConfiguration: CompositorLayerConfiguration {
        func makeConfiguration(capabilities: LayerRenderer.Capabilities,
        configuration: inout LayerRenderer.Configuration)
        // Configure other aspects of LayerRenderer.

            if configuration.layout == .layered {
                let stencilFormat: MTLPixelFormat = .stencil8
                if capabilities.drawableRenderContextSupportedStencilFormats.contair
                    configuration.drawableRenderContextStencilFormat = stencilFormat
                }
            }
        configuration.drawableRenderContextRasterSampleCount = 1
        }
    }
}
```

> **Note**
>
> The progressive immersion style only works with the layered layout configuration.

Apply a stencil mask

Apply the portal stencil mask to the layer render drawable by adding a render context to your command buffer with `addRenderContext(commandBuffer:)`. This step is a requirement for the system to efficiently draw the faded edges of the portal over your rendered 3D content. Use

the same command buffer for your subsequent drawing commands. Draw your mask on the stencil with `drawMaskOnStencilAttachment(commandEncoder:value:)` to prevent invisible pixels from rendering and then select an available stencil value. After rendering the scene, use `endEncoding(commandEncoder:)` to cease rendering, rather than directly ending the command encoder. This allows the portal effect to apply efficiently over your content.

```
struct Renderer {
    let portalStencilValue: UInt8 = 200 // Value isn't used in other stencil operati

    func renderFrame with scene: MyScene, drawable: LayerRenderer.Drawable, commandB
        let drawableRenderContext = drawable.addRenderContext(commandBuffer: commanc
        let renderEncoder = configureRenderPass(commandBuffer: commandBuffer)

        drawableRenderContext.drawMaskOnStencilAttachment(commandEncoder: renderEnco
        renderEncoder.setStencilReferenceValue(UInt32(portalStencilValue))
        scene.render(to: drawable, renderEncoder: renderEncoder)
        drawableRenderContext.endEncoding(commandEncoder: commandEncoder)
        drawable.encodePresent(commandBuffer: commandBuffer)
    }
}
```