Sample Code

Adjusting the level of detail using Metal mesh shaders

Choose and render meshes with several levels of detail using object and mesh shaders.

Download

iOS 16.0+ | iPadOS 16.0+ | macOS 13.0+ | Xcode 14.0+

Overview

Note

This sample code project is associated with WWDC22 session 10162: Transform your geometry with Metal mesh shaders.

Configure the sample code project

To run this sample, you need Xcode 14 or later, and a physical device that supports `MTLGPUFamily.mac2` or `MTLGPUFamily.apple7`, such as:

- A Mac running macOS 13 or later
- An iOS device with an A15 chip or later running iOS 16 or later

This sample can only run on a physical device because it uses mesh shader features, which Simulator doesn't support.

See Also

Render workflows

{} Using Metal to draw a view's contents

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

{} Drawing a triangle with Metal 4

Render a colorful, rotating 2D triangle by running draw commands with a render pipeline on a GPU.

{} Selecting device objects for graphics rendering

Switch dynamically between multiple GPUs to efficiently render to a display.

{} Customizing render pass setup

Render into an offscreen texture by creating a custom render pass.

{} Creating a custom Metal view

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

{} Calculating primitive visibility using depth testing

Determine which pixels are visible in a scene by using a depth texture.

{} Encoding indirect command buffers on the CPU

Reduce CPU overhead and simplify your command execution by reusing commands.

{} Implementing order-independent transparency with image blocks

Draw overlapping, transparent surfaces in any order by using tile shaders and image blocks.

{} Loading textures and models using Metal fast resource loading

Stream texture and buffer data directly from disk into Metal resources using fast resource loading.

{} Creating a 3D application with hydra rendering

Build a 3D application that integrates with Hydra and USD.

{} Culling occluded geometry using the visibility result buffer

Draw a scene without rendering hidden geometry by checking whether each object in the scene is visible.

{} Improving edge-rendering quality with multisample antialiasing (MSAA)

Apply MSAA to enhance the rendering of edges with custom resolve options and immediate and tile-based resolve paths.

{} 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.