Metal sample code library

Explore the complete set of Metal samples.

Overview

Browse the topics below to find samples relevant to a concept you want to learn more about, starting with the basic computation and render workflows. The samples in the lighting and multiple technique sections demonstrate how to take advantage of the unique GPU architecture of Apple silicon.

Topics

Compute workflows

{} Performing calculations on a GPU

Use Metal to find GPUs and perform calculations on them.

{} Selecting device objects for compute processing

Switch dynamically between multiple GPUs to efficiently execute a compute-intensive simulation.

{} Customizing a TensorFlow operation

Implement a custom operation that uses Metal kernels to accelerate neural-network training performance.

{} Customizing a PyTorch operation

Implement a custom operation in PyTorch that uses Metal kernels to improve performance.

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.

{} Adjusting the level of detail using Metal mesh shaders

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

{} 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.

Textures

{} Processing a texture in a compute function

Create textures by running copy and dispatch commands in a compute pass on a GPU.

{} Reading pixel data from a drawable texture

Access texture data from the CPU by copying it to a buffer.

{} Creating and sampling textures

Load image data into a texture and apply it to a quadrangle.

{} Streaming large images with Metal sparse textures

Limit texture memory usage for large textures by loading or unloading image detail on the basis of MIP and tile region.

Argument buffers

{} Managing groups of resources with argument buffers

Create argument buffers to organize related resources.

{} Using argument buffers with resource heaps

Reduce CPU overhead by using arrays inside argument buffers and combining them with resource heaps.

{} Encoding argument buffers on the GPU

Use a compute pass to encode an argument buffer and access its arguments in a subsequent render pass.

{} Rendering terrain dynamically with argument buffers

Use argument buffers to render terrain in real time with a GPU-driven pipeline.

Shaders

{} Creating a Metal dynamic library

Compile a library of shaders and write it to a file as a dynamically linked library.

{} Using function specialization to build pipeline variants

Create pipelines for different levels of detail from a common shader source.

Synchronization

{} Synchronizing CPU and GPU work

Avoid stalls between CPU and GPU work by using multiple instances of a resource.

{} Implementing a multistage image filter using heaps and events

Use events to synchronize access to resources allocated on a heap.

{} Implementing a multistage image filter using heaps and fences

Use fences to synchronize access to resources allocated on a heap.

Lighting techniques

{} Rendering a scene with forward plus lighting using tile shaders

Implement a forward plus renderer using the latest features on Apple GPUs.

{} Rendering a scene with deferred lighting in Objective-C

Avoid expensive lighting calculations by implementing a deferred lighting renderer optimized for immediate mode and tile-based deferred renderer GPUs.

{} Rendering a scene with deferred lighting in Swift

Avoid expensive lighting calculations by implementing a deferred lighting renderer optimized for immediate mode and tile-based deferred renderer GPUs.

{} Rendering a scene with deferred lighting in C++

Avoid expensive lighting calculations by implementing a deferred lighting renderer optimized for immediate mode and tile-based deferred renderer GPUs.

{} Rendering reflections with fewer render passes

Use layer selection to reduce the number of render passes needed to generate an environment map.

Multiple techniques

{} Modern rendering with Metal

Use advanced Metal features such as indirect command buffers, sparse textures, and variable rate rasterization to implement complex rendering techniques.

{} Encoding indirect command buffers on the GPU

Maximize CPU to GPU parallelization by generating render commands on the GPU.

Ray tracing

{} Rendering reflections in real time using ray tracing

Implement realistic real-time lighting by dynamically generating reflection maps by encoding a ray-tracing compute pass.

{} Accelerating ray tracing using Metal

Implement ray-traced rendering using GPU-based parallel processing.

{} Control the ray tracing process using intersection queries

Explicitly enumerate a ray's intersections with acceleration structures by creating an intersection query object.

{} Accelerating ray tracing and motion blur using Metal

Generate ray-traced images with motion blur using GPU-based parallel processing.

{} Rendering a curve primitive in a ray tracing scene

Implement ray traced rendering using GPU-based parallel processing.

HDR

{} Processing HDR images with Metal

Implement a post-processing pipeline using the latest features on Apple GPUs.

OpenGL

{} Migrating OpenGL code to Metal

Replace your app's deprecated OpenGL code with Metal.

{} Mixing Metal and OpenGL rendering in a view

Draw with Metal and OpenGL in the same view using an interoperable texture.