Framework

Metal

Render advanced 3D graphics and compute data in parallel with graphics processors.

iOS 8.0+ | iPadOS 8.0+ | Mac Catalyst 13.0+ | macOS 10.11+ | tvOS 9.0+ | visionOS 1.0+

Overview

The Metal framework gives your app direct access to a device's graphics processing unit (GPU). With Metal, apps can leverage a GPU to quickly render complex scenes and run computational tasks in parallel. For example, apps in these categories use Metal to maximize their performance:

- Games that render sophisticated 2D or 3D environments
- Video processing apps, like Final Cut Pro
- Scientific research apps that analyze and process large datasets
- Fully immersive visionOS apps

Metal works hand-in-hand with other frameworks that supplement its capability. For example, MetalFX upscales your renderings in less time than rendering them natively, and MetalKit simplifies the tasks that display your Metal content onscreen. The Metal Performance Shaders framework provides a large library of optimized compute and rendering shaders that take advantage of each GPU's unique hardware. In visionOS, create fully immersive stereoscopic content with the help of the Compositor Services framework.

Many high-level Apple frameworks leverage the performance of Metal, including RealityKit, SpriteKit, and Core Image. These high-level frameworks implement the GPU programming details for you. However, you can typically get better performance by writing your own custom Metal and shader code. See the Metal Shading Language Specification for shader implementation details.

Topics

Essentials

Begin with the Metal fundamentals.

- Understanding the Metal 4 core API
  Discover the features and functionality in the Metal 4 foundational APIs.
- {} Drawing a triangle with Metal 4
  Render a colorful, rotating 2D triangle by running draw commands with a render pipeline on a GPU.
- {} Performing calculations on a GPU
  Use Metal to find GPUs and perform calculations on them.
- {} Using Metal to draw a view's contents
  Create a MetalKit view and a render pass to draw the view's contents.

Samples

Discover graphics techniques and Metal features through sample code projects.

- Metal sample code library
  Explore the complete set of Metal samples.

GPU devices

Start with a Metal device instance to begin working with the GPU it represents.

- GPU devices and work submission
  Find any available GPU, submit work to it with command buffers, suspend work, and coordinate between multiple GPUs.

Command encoders

Send work to a GPU by issuing commands and configuring the pipeline states for those commands.

- Render passes
  Encode a render pass to draw graphics into an image.
- Compute passes

Encode a compute pass that runs computations in parallel on a thread grid, processing and manipulating Metal resource data on multiple cores of a GPU.

- Machine-learning passes

  Add machine-learning model inference to your Metal app's GPU workflow.

- Blit passes

  Encode a block information transfer pass to adjust and copy data to and from GPU resources, such as buffers and textures.

- Indirect command encoding

  Store draw commands in Metal buffers and run them at a later time on the GPU, either once or repeatedly.

- Ray tracing with acceleration structures

  Build a representation of your scene's geometry using triangles and bounding volumes to quickly trace rays through the scene.

Resources

Store data in buffers and textures, and optionally manage the underlying GPU memory yourself.

- Resource fundamentals

  Control the common attributes of all Metal memory resources, including buffers and textures, and how to configure their underlying memory.

- Buffers

  Create and manage untyped data your app uses to exchange information with its shader functions.

- Textures

  Create and manage typed data your app uses to exchange information with its shader functions.

- Memory heaps

  Take control of your app's GPU memory management by creating a large memory allocation for various buffers, textures, and other resources.

- Resource loading

  Load assets in your games and apps quickly by running a dedicated input/output queue alongside your GPU tasks.

- Resource synchronization

Prevent multiple commands that can access the same resources simultaneously by coordinating those accesses with barriers, fences, or events.

Shader compilation and libraries

Compile and organize shaders, the GPU functions that run on a Metal device's execution units.

- Using the Metal 4 compilation API
  Control when and how you compile an app's shaders.
- Shader libraries
  Manage and load your app's Metal shaders.
- Using function specialization to build pipeline variants
  Create pipelines for different levels of detail from a common shader source.

Presentation

Display standard or high-dynamic-range content on a device's display with Core Animation or MetalKit, in standard or high dynamic range.

- Managing your game window for Metal in macOS
  Set up a window and view for optimally displaying your Metal content.
- Managing your Metal app window in iPadOS
  Set up a window that handles dynamically resizing your Metal content.
- Adapting your game interface for smaller screens
  Make text legible on all devices the player chooses to run your game on.
- Onscreen presentation
  Show the output from a GPU's rendering pass to the user in your app.
- HDR content
  Take advantage of high dynamic range to present more vibrant colors in your apps and games.

Developer tools

Identify and fix issues with your app's Metal API calls, shader code, resources, and performance during development by using Metal Debugger.

{} Supporting Simulator in a Metal app

Configure alternative render paths in your Metal app to enable running your app in Simulator.

{} Capturing Metal commands programmatically

Invoke a Metal frame capture from your app, then save the resulting GPU trace to a file or view it in Xcode.

Logging shader debug messages

Print debugging messages that a shader generates using shader logging.

Developing Metal apps that run in Simulator

Prototype and test your Metal apps in Simulator.

Improving your game's graphics performance and settings

Fix performance glitches and develop default settings for smooth experiences on Apple platforms using the powerful suite of Metal development tools.

Metal debugger

Debug and profile your Metal workload with a GPU trace.

Metal developer workflows

Locate and fix issues related to your app's use of the Metal API and GPU functions.

GPU counters and counter sample buffers

Retrieve runtime data from a GPU device by sampling one or more of its counters.

Metal debugging types

Create capture managers and capture scopes, and review a GPU device's log after it runs a command buffer.

Apple silicon

Take advantage of the unique architecture of Apple silicon GPUs.

Porting your Metal code to Apple silicon

Create a version of your Metal app that runs on both Apple silicon and Intel-based Mac computers.

Tailor your apps for Apple GPUs and tile-based deferred rendering

Learn about characteristic Apple GPU features, including imageblocks, tile shaders, and raster order groups.

Reference

- Metal structures
- Metal enumerations
- Metal constants
- Metal data types
- Metal variables

See Also

Related Documentation

Metal Programming Guide

Metal Best Practices Guide