API Collection

GPU counters and counter sample buffers

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

Overview

A GPU *counter* (`MTLCounter`) is typically a hardware feature that tracks a specific performance metric, such as timestamps before and after an important rendering stage. A *counter set* (`MTLCounterSet`) is a collection of related counters. A *counter sample buffer* (`MTLCounterSampleBuffer`) represents the memory where a GPU device stores the data for a specific counter set.

You can retrieve and inspect data from a GPU's counter set with the following steps:

1. Inspect which GPU counter sets a GPU device supports (see Confirming which counters and counter sets a GPU supports).
2. Make a counter sample buffer to store the data (see Creating a counter sample buffer to store a GPU's counter data during a pass).
3. Instruct the GPU to save the counter set data to the buffer during a pass or an immediate mode command (see Sampling GPU data into counter sample buffers).
4. Transform the counter set data into a standard type (see Converting a GPU's counter data into a readable format).

If you're sampling data from a timestamp counter set (`timestamp`), you may need to convert the timestamps from the GPU's clock to the CPU's clock. See Converting GPU timestamps into CPU time for more information.

Topics

Counters and counter sets

Confirming which counters and counter sets a GPU supports

Check whether a GPU produces the runtime performance data you want to sample.

`protocol` `MTLCounterSet`

A collection of individual counters a GPU device supports for a counter set.

`struct` `MTLCommonCounterSet`

The name of a specific counter set that a GPU device can support.

`protocol` `MTLCounter`

An individual counter a GPU device lists within one of its counter sets.

`struct` `MTLCommonCounter`

The name of a specific counter that can appear in a GPU device's counter sets.

Counter sample buffers

Creating a counter sample buffer to store a GPU's counter data during a pass

Make a buffer that provides a place for a GPU to save its runtime performance metrics as it runs a pass.

`class` `MTLCounterSampleBufferDescriptor`

A group of properties that configures the counter sample buffers you create with it.

`protocol` `MTLCounterSampleBuffer`

A specialized memory buffer that stores a GPU's counter set data.

Sampling GPU data into counter sample buffers

Retrieve a GPU's counter data at a time the GPU supports.

`var` `MTLCounterDontSample: Int`

A sentinel value that instructs an encoder to skip sampling a counter as the GPU runs the encoder's pass.

Counter sample data output

Converting a GPU's counter data into a readable format

Inspect and use the data within a GPU's counter sample buffer by resolving it into a standard format.

`struct MTLCounterResultTimestamp`

The data structure for storing the data you resolve from a timestamp counter set.

`struct MTLCounterResultStatistic`

The data structure for storing the data you resolve from a statistic counter set.

`struct MTLCounterResultStageUtilization`

The data structure for storing the data you resolve from a stage-utilization counter set.

`var MTLCounterErrorValue: UInt64`

A sentinel value for an entry in a counter sample buffer that indicates the entry's data is invalid.

Timestamp data

Converting GPU timestamps into CPU time

Correlate GPU events with CPU timelines by calculating the CPU time equivalents for GPU timestamps.

`typealias MTLTimestamp`

The number of nanoseconds for a point in absolute time or Mach absolute time.

Counter sample buffer errors

`struct MTLCounterSampleBufferError`

The error codes that indicate why a GPU driver can't create a counter sample buffer.

See Also

Developer tools

{} 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.

- Metal debugging types

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