API Collection

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.

Overview

Ray tracing can improve your content's realism by more accurately modeling the behavior of light than traditional rendering. You can also use ray tracing to implement similar techniques that rely on line-of-sight, such as sound obstruction or visually based AI functions.

To apply ray tracing in your app:

1. Create acceleration structures that represent objects in a scene.
2. Define a ray's behavior when it collides into parts of an acceleration structure by creating either intersectors or intersection queries.
3. Generate rays into the scene from a new or existing shader.

An *intersector* uses a table of your intersection functions that define the custom behavior for each intersection type. An *intersection query* returns to your calling function to handle the custom behavior for all intersection types.

Intersectors work with compute kernels on all GPUs, and with render shaders only on Apple silicon GPUs. Alternatively, your app can use intersection queries on non-Apple GPUs, or for porting code from other graphics APIs.

Topics

Ray tracing samples

{} 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.

{} 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.

{} Rendering a curve primitive in a ray tracing scene

Implement ray traced rendering using GPU-based parallel processing.

Acceleration structures

Improving ray-tracing data access using per-primitive data

Simplify data access and improve GPU utilization by storing custom primitive data directly in the acceleration structure.

`protocol` `MTLAccelerationStructure`

A collection of model data for GPU-accelerated intersection of rays with the model.

`class` `MTL4AccelerationStructureDescriptor`

Base class for Metal 4 acceleration structure descriptors.

`class` `MTLAccelerationStructureDescriptor`

A base class for classes that define the configuration for a new acceleration structure.

`class` `MTL4PrimitiveAccelerationStructureDescriptor`

Descriptor for a primitive acceleration structure that directly references geometric shapes, such as triangles and bounding boxes.

`class` `MTLPrimitiveAccelerationStructureDescriptor`

A description of an acceleration structure that contains geometry primitives.

`class` `MTL4InstanceAccelerationStructureDescriptor`

Descriptor for an instance acceleration structure.

`class` `MTLInstanceAccelerationStructureDescriptor`

A description of an acceleration structure that derives from instances of primitive acceleration structures.

`protocol` MTLAccelerationStructureCommandEncoder

An object for encoding commands that build or refit acceleration structures.

`struct` MTLAccelerationStructureUsage

Options that affect how Metal builds an acceleration structure and the behavior of that acceleration structure.

`struct` MTLAccelerationStructureRefitOptions

Acceleration structures passes

`class` MTLAccelerationStructurePassDescriptor

`class` MTLAccelerationStructurePassSampleBufferAttachmentDescriptor

`class` MTLAccelerationStructurePassSampleBufferAttachmentDescriptorArray

Geometry descriptors

`class` MTL4AccelerationStructureGeometryDescriptor

Base class for all Metal 4 acceleration structure geometry descriptors.

`class` MTLAccelerationStructureGeometryDescriptor

A base class for descriptors that contain geometry data to convert into a ray-tracing acceleration structure.

`class` MTL4AccelerationStructureTriangleGeometryDescriptor

Describes triangle geometry suitable for ray tracing.

`class` MTLAccelerationStructureTriangleGeometryDescriptor

A description of a list of triangle primitives to turn into an acceleration structure.

`class` MTL4AccelerationStructureCurveGeometryDescriptor

Describes curve geometry suitable for ray tracing.

`class` MTLAccelerationStructureCurveGeometryDescriptor

`enum` MTLCurveType

`enum` MTLCurveBasis

`enum` MTLCurveEndCaps

`class` MTL4AccelerationStructureBoundingBoxGeometryDescriptor

Describes bounding-box geometry suitable for ray tracing.

`class` `MTLAccelerationStructureBoundingBoxGeometryDescriptor`

A description of a list of bounding boxes to turn into an acceleration structure.

Motion geometry descriptors

`class` `MTL4AccelerationStructureMotionTriangleGeometryDescriptor`

Describes motion triangle geometry, suitable for motion ray tracing.

`class` `MTLAccelerationStructureMotionTriangleGeometryDescriptor`

A description of a list of triangle primitives, as motion keyframe data, to turn into an acceleration structure.

`class` `MTL4AccelerationStructureMotionCurveGeometryDescriptor`

Describes motion curve geometry, suitable for motion ray tracing.

`class` `MTLAccelerationStructureMotionCurveGeometryDescriptor`

`class` `MTL4AccelerationStructureMotionBoundingBoxGeometryDescriptor`

Describes motion bounding box geometry, suitable for motion ray tracing.

`class` `MTLAccelerationStructureMotionBoundingBoxGeometryDescriptor`

A description of a list of bounding boxes, as motion keyframe data, to turn into an acceleration structure.

`class` `MTLMotionKeyframeData`

Geometry data for a specific keyframe to use in a moving instance.

Instance descriptors

`struct` `MTLAccelerationStructureInstanceDescriptor`

A description of an instance in an instanced geometry acceleration structure.

`struct` `MTLAccelerationStructureUserIDInstanceDescriptor`

A description of an instance in an instanced geometry acceleration structure, with the instance including a user identifier for the instance.

`struct` `MTLAccelerationStructureMotionInstanceDescriptor`

A description of an instance in an instanced geometry acceleration structure, with the instance including a user identifier and motion data for the instance.

`struct` `MTLAccelerationStructureInstanceOptions`

Options for adjusting the behavior of an instanced acceleration structure.

`class` `MTL4IndirectInstanceAccelerationStructureDescriptor`

Descriptor for an “indirect” instance acceleration structure that allows providing the instance count and motion transform count indirectly, through buffer references.

`class` `MTLIndirectInstanceAccelerationStructureDescriptor`

A description of an acceleration structure that Metal derives from instances of primitive acceleration structures that the GPU can populate.

`struct` `MTLIndirectAccelerationStructureInstanceDescriptor`

A description of an instance in an instanced geometry acceleration structure that the GPU can populate.

`struct` `MTLIndirectAccelerationStructureMotionInstanceDescriptor`

A description of an instance in an acceleration structure that the GPU can populate, with motion data for the instance.

Intersection function tables

`protocol` `MTLIntersectionFunctionTable`

A table of intersection functions that Metal calls to perform ray-tracing intersection tests.

`class` `MTLIntersectionFunctionTableDescriptor`

A specification of how to create an intersection function table.

`class` `MTLIntersectionFunctionDescriptor`

A description of an intersection function that performs an intersection test.

`struct` `MTLIntersectionFunctionSignature`

Constants for specifying different types of custom intersection functions.

`struct` `MTLIntersectionFunctionBufferArguments`

Supporting types

`typealias` `MTLAxisAlignedBoundingBox`

The bounds for an axis-aligned bounding box.

`typealias` `MTLPackedFloat3`

}

`typealias` `MTLPackedFloat4x3`

A structure that contains the top three rows of a 4x4 matrix of 32-bit floating-point values, in column-major order.

`func` `MTLPackedFloat3Make(Float, Float, Float) -> MTLPackedFloat3`

Returns a new packed vector with three floating-point values.

`struct` `MTL4BufferRange`

`func` `MTL4BufferRangeMake(MTLGPUAddress, UInt64) -> MTL4BufferRange`

See Also

Command encoders

- 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.