Metal for Ray Tracing Acceleration

Session 606

Sean James, GPU Software Engineer
Wayne Lister, GPU Software Engineer
Metal Performance Shaders

GPU-accelerated primitives, optimized for iOS and macOS

- Image processing
- Linear algebra
- Machine learning—infERENCE

<table>
<thead>
<tr>
<th>Using Metal 2 for Compute</th>
<th>WWDC 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in Metal, Part 2</td>
<td>WWDC 2016</td>
</tr>
</tbody>
</table>
Metal Performance Shaders

GPU-accelerated primitives, optimized for iOS and macOS

• Image processing
• Linear algebra
• Machine learning—inference and training

Metal for Accelerating Machine Learning  Hall 3  Thursday 4:00PM
Metal Performance Shaders

GPU-accelerated primitives, optimized for iOS and macOS

• Image processing
• Linear algebra
• Machine learning—infERENCE AND TRAINING
• Ray tracing
Ray Tracing

Tracing a ray’s path as it interacts with a scene
Ray Tracing

Tracing a ray’s path as it interacts with a scene
Ray Tracing

Tracing a ray’s path as it interacts with a scene

Applications

- Rendering
- Audio and physics simulation
- Collision detection
- AI and pathfinding
Rasterization

Projects triangles onto the screen one at a time
Rasterization

Projects triangles onto the screen one at a time

Fast—method of choice for games and real-time applications
Rasterization

Projects triangles onto the screen one at a time

Fast—method of choice for games and real-time applications

Difficult to model behavior of light
Ray Tracing

Reflections

Can be computed accurately with ray tracing
Ray Tracing
Soft shadows

Can be computed directly with ray tracing

Realistic transition from hard to soft shadows
Ray Tracing
Global illumination

Naturally modeled with ray tracing
Ray Tracing

Many other effects—ambient occlusion, refraction, area lights, depth of field, motion blur
Ray Tracing

Many other effects—ambient occlusion, refraction, area lights, depth of field, motion blur

Method of choice for photorealistic, offline rendering
Ray Tracing

Many other effects—ambient occlusion, refraction, area lights, depth of field, motion blur

Method of choice for photorealistic, offline rendering

Significantly more computationally expensive—doing more work to simulate physical effects
Rendering with Ray Tracing

Work backwards from camera to light source
Rendering with Ray Tracing

Work backwards from camera to light source

Cast *primary rays* from the camera into the scene
Rendering with Ray Tracing

Work backwards from camera to light source

Cast primary rays from the camera into the scene

Compute shading at intersection points
Rendering with Ray Tracing

Compute direct lighting contribution
Rendering with Ray Tracing

Compute direct lighting contribution

Cast shadow rays from intersection point to light source
Rendering with Ray Tracing

Compute direct lighting contribution

Cast *shadow rays* from intersection point to light source

In shadow if shadow ray does not reach light
Rendering with Ray Tracing

Cast secondary rays from intersection point in random directions
Rendering with Ray Tracing

Cast secondary rays from intersection point in random directions
Rendering with Ray Tracing

Cast *secondary rays* from intersection point in random directions
Rendering with Ray Tracing

Cast secondary rays from intersection point in random directions

Add direct lighting reflected from secondary intersection point
Rendering with Ray Tracing

Cast **secondary rays** from intersection point in random directions

Add direct lighting reflected from secondary intersection point
Rendering with Ray Tracing

Cast **secondary rays** from intersection point in random directions

Add direct lighting reflected from secondary intersection point

Repeat to simulate light bouncing
Rendering with Ray Tracing

Ray count grows exponentially with number of bounces
Rendering with Ray Tracing

Ray count grows exponentially with number of bounces

Avoid growth—one shadow ray and secondary ray per bounce
Rendering with Ray Tracing

Ray count grows exponentially with number of bounces

Avoid growth—one shadow ray and secondary ray per bounce
Rendering with Ray Tracing

Ray count grows exponentially with number of bounces

Avoid growth—one shadow ray and secondary ray per bounce

Average over multiple frames
Rendering with Ray Tracing
Generate Primary Rays
Rendering with Ray Tracing

1. Generate Primary Rays
2. Intersect with Scene
Rendering with Ray Tracing

1. Generate Primary Rays
2. Intersect with Scene
3. Intersections
4. Shading

Primary Rays → Intersect with Scene → Intersections → Shading
Rendering with Ray Tracing

1. Generate Primary Rays
2. Intersect with Scene
3. Shadow, Secondary Rays
4. Shading
Rendering with Ray Tracing

- Generate Primary Rays
- Intersect with Scene
- Intersections → Shadow, Secondary Rays
- Shading
Rendering with Ray Tracing

- Generate Primary Rays
- Intersect with Scene
- Shading
- Image

Primary Rays → Intersect with Scene → Shading

Intersections → Shadow, Secondary Rays
Rendering with Ray Tracing

1. Generate Primary Rays
2. Intersect with Scene
   - Intersections
   - Shadow, Secondary Rays
3. Shading
4. Image
MPSRayIntersector

Ray intersector accelerates ray/triangle intersection tests on the GPU
MPSRayIntersector

Ray intersector accelerates ray/triangle intersection tests on the GPU

Accepts batches of rays in a Metal buffer
MPSRayIntersector

Ray *intersector* accelerates ray/triangle intersection tests on the GPU

Accepts batches of rays in a Metal buffer

Returns one intersection per ray
MPSRayIntersector

Ray intersector accelerates ray/triangle intersection tests on the GPU

Accepts batches of rays in a Metal buffer

Returns one intersection per ray

Encodes into a Metal command buffer
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
Accelerating Ray/Scene Intersection
MPSRayIntersector

Scene represented by triangle vertices in a **vertex buffer**
MPSRayIntersector

Scene represented by triangle vertices in a vertex buffer

Build an acceleration structure over the vertex buffer
MPSRayIntersector

Scene represented by triangle vertices in a **vertex buffer**

Build an **acceleration structure** over the vertex buffer

Pass acceleration structure to intersector
Ray Tracing with Metal Performance Shaders
Ray Tracing with Metal Performance Shaders

Primary Rays and Shading
Ray Tracing with Metal Performance Shaders

Primary Rays and Shading   Shadow Rays
Ray Tracing with Metal Performance Shaders

- Primary Rays and Shading
- Shadow Rays
- Secondary Rays
Ray Tracing with Metal Performance Shaders

Primary Rays and Shading

Shadow Rays

Secondary Rays
Primary Rays and Shading
Primary Rays and Shading

Intersector
Primary Rays and Shading

- Vertex Buffer
- Acceleration Structure
- Intersector
Primary Rays and Shading

Generate Primary Rays → Ray Buffer → Intersector → Acceleration Structure → Vertex Buffer
Primary Rays and Shading

- Generate Primary Rays
- Ray Buffer
- Intersector
- Intersection Buffer
- Acceleration Structure
- Vertex Buffer
Primary Rays and Shading

- Generate Primary Rays
- Ray Buffer
- Intersector
- Intersection Buffer
- Acceleration Structure
- Image
- Vertex Buffer
- Shading
Creating an Intersector

Create an MPSRayIntersector with a Metal device:

```swift
let intersector = MPSRayIntersector(device: device)
```
Primary Rays and Shading

Generate Primary Rays -> Ray Buffer -> Intersector -> Intersection Buffer -> Shading

Acceleration Structure

Image

Vertex Buffer
Primary Rays and Shading

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading

Acceleration Structure

Vertex Buffer

Image
Creating an Acceleration Structure

Create an MPSTriangleAccelerationStructure with a Metal device:

```swift
let accelerationStructure = MPSTriangleAccelerationStructure(device: device)
```
Creating an Acceleration Structure

Create an MPSTriangleAccelerationStructure with a Metal device:

```swift
let accelerationStructure = MPSTriangleAccelerationStructure(device: device)
```

Assign vertex buffer:

```swift
accelerationStructure.vertexBuffer = vertexBuffer
accelerationStructure.triangleCount = triangleCount
```
Creating an Acceleration Structure

Create an MPSTriangleAccelerationStructure with a Metal device:

```swift
let accelerationStructure = MPSTriangleAccelerationStructure(device: device)
accelerationStructure.vertexBuffer = vertexBuffer
accelerationStructure.triangleCount = triangleCount
```

Assign vertex buffer:

```swift
accelerationStructure.vertexBuffer = vertexBuffer
accelerationStructure.triangleCount = triangleCount
```

Build acceleration structure:

```swift
accelerationStructure.rebuild()
```
Primary Rays and Shading

1. Generate Primary Rays
2. Ray Buffer
3. Intersector
4. Intersection Buffer
5. Acceleration Structure
6. Shading
7. Image
8. Vertex Buffer
Primary Rays and Shading

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading

Acceleration Structure → Image

Vertex Buffer
Generating Primary Rays

Launch one thread per pixel
Generating Primary Rays

Launch one thread per pixel

Write Ray struct to ray buffer:

```c
struct Ray {
    float3 origin;
    float3 direction;
};
```
Generating Primary Rays

Launch one thread per pixel

Write Ray struct to ray buffer:

```c
struct Ray {
    float3 origin;
    float3 direction;
};
```
Generating Primary Rays

Launch one thread per pixel

Write Ray struct to ray buffer:

```c
struct Ray {
    float3 origin;
    float3 direction;
};
```
Generating Primary Rays

Launch one thread per pixel

Write Ray struct to ray buffer:

```c
struct Ray {
    float3 origin;
    float3 direction;
};
```
Primary Rays and Shading

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading

Acceleration Structure → Image → Vertex Buffer
Primary Rays and Shading

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading

Acceleration Structure

Image

Vertex Buffer
Finding Intersections with the Scene

Encode intersection test into a command buffer:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: 0,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: 0,
    rayCount: rayCount,
    accelerationStructure: accelerationStructure)
```
Finding Intersections with the Scene

Encode intersection test into a command buffer:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
intersectionType: .nearest,
rayBuffer: rayBuffer,
rayBufferOffset: 0,
intersectionBuffer: intersectionBuffer,
intersectionBufferOffset: 0,
rayCount: rayCount,
accelerationStructure: accelerationStructure)
```
Finding Intersections with the Scene

Encode intersection test into a command buffer:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
                               intersectionType: .nearest,
                               rayBuffer: rayBuffer,
                               rayBufferOffset: 0,
                               intersectionBuffer: intersectionBuffer,
                               intersectionBufferOffset: 0,
                               rayCount: rayCount,
                               accelerationStructure: accelerationStructure)
```
Finding Intersections with the Scene

Encode intersection test into a command buffer:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: 0,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: 0,
    rayCount: rayCount,
    accelerationStructure: accelerationStructure)
```
Finding Intersections with the Scene

Encode intersection test into a command buffer:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: 0,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: 0,
    rayCount: rayCount,
    accelerationStructure: accelerationStructure)
```
Finding Intersections with the Scene

Encode intersection test into a command buffer:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: 0,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: 0,
    rayCount: rayCount,
    accelerationStructure: accelerationStructure)
```
Primary Rays and Shading

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading

Acceleration Structure

Vertex Buffer → Image
Shading

Apply lighting and textures similar to fragment shader

Depends on intersection point and vertex attributes
```c
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};
```

Shading

Intersection point
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};
Shading

```c
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};
```
Shading

```c
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};
```
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};

Barycentric coordinates $u$ and $v$
Shading

```c
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};
```

Barycentric coordinates $u$ and $v$

Third coordinate: $w = 1 - u - v$
Shading

Barycentric coordinates \( u \) and \( v \)

Third coordinate: \( w = 1 - u - v \)

Interpolated: \( uV_0 + vV_1 + wV_2 \)

```c
struct Intersection {
    float distance;
    int primitiveIndex;
    float2 coordinates;
};
```
Primary Rays and Shading

1. Generate Primary Rays
2. Intersector
3. Intersection Buffer
4. Acceleration Structure
5. Shading
6. Image
7. Vertex Buffer

Ray Buffer
Primary Rays and Shading

Shadow Rays

Secondary Rays
Primary Rays and Shading

Shadow Rays

Secondary Rays
Shadow Rays

Check if shading point is in shadow before adding color to image
Shadow Rays

Check if shading point is in shadow before adding color to image

Cast a ray from shading point to light source
Shadow Rays

Check if shading point is in shadow before adding color to image

Cast a ray from shading point to light source

If shadow ray does not reach light, shading point is in shadow
Shadow Rays

Check if shading point is in shadow before adding color to image

Cast a ray from shading point to light source

If shadow ray does not reach light, shading point is in shadow
Shadow Rays

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading

Acceleration Structure

Image
Shadow Rays

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading
Shadow Rays

- Generate Primary Rays
- Intersector
- Shading

Flow:
- Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading
- Shadow Ray Buffer

Diagram Flow:
Shadow Rays

Generate Primary Rays → Intersector → Shading

Ray Buffer → Intersection Buffer

Shadow Ray Buffer → Intersector → Intersection Buffer
Shadow Rays

1. Generate Primary Rays
2. Intersector
3. Ray Buffer
4. Intersection Buffer
5. Shading

6. Shadow Ray Buffer
7. Intersector
8. Intersection Buffer
9. Add Color to Image

Image
Shadow Rays

Generate Primary Rays → Ray Buffer → Intersector → Intersection Buffer → Shading

Shadow Ray Buffer

Add Color to Image → Intersection Buffer

Image
Shadow Rays versus Primary Rays
Shadow Rays versus Primary Rays

Maximum intersection distance
Shadow Rays versus Primary Rays

Maximum intersection distance

Don’t need triangle index or barycentric coordinates
Shadow Rays versus Primary Rays

Maximum intersection distance

Don’t need triangle index or barycentric coordinates

Propagate color from shading kernel to final kernel
Customizing the Ray Struct

Ray type is configurable

```c
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```
Customizing the Ray Struct

Ray type is configurable

Choose what data is provided to the intersector

```c
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```
Customizing the Ray Struct

Ray type is configurable
Choose what data is provided to the intersector
Append app-specific data

```c
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```
Customizing the Ray Struct

Ray type is configurable

Choose what data is provided to the intersector

Append app-specific data

```c
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```

Configure MPSRayIntersector for new Ray struct type:

```c
intersector.rayDataType = .originMinDistanceDirectionMaxDistance
intersector.rayStride = MemoryLayout<Ray>.stride
```
Customizing the Ray Struct

Ray type is configurable

Choose what data is provided to the intersector

Append app-specific data

Configure MPSRayIntersector for new Ray struct type:

```cpp
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```

```cpp
intersector.rayDataType = .originMinDistanceDirectionMaxDistance
intersector.rayStride = MemoryLayout<Ray>.stride
```
Customizing the Ray Struct

Ray type is configurable

Choose what data is provided to the intersector

Append app-specific data

```cpp
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```

Configure MPSRayIntersector for new Ray struct type:

```cpp
intersector.rayDataType = .originMinDistanceDirectionMaxDistance
intersector.rayStride = MemoryLayout<Ray>.stride
```
Shadow Rays

Generate Primary Rays → Intersector → Intersection Buffer → Shading

Intersection Buffer

Shadow Ray Buffer → Intersector → Intersection Buffer → Add Color to Image → Image
Shadow Rays

1. Generate Primary Rays
2. Intersector
3. Ray Buffer
4. Shading
5. Intersection Buffer
6. Add Color to Image
7. Image
8. Shadow Ray Buffer
9. Intersector
10. Intersection Buffer
Finding Shadow Ray Intersections

Modify call to MPSRayIntersector:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: 0,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: 0,
    rayCount: rayCount,
    accelerationStructure: accelerationStructure)
```
Finding Shadow Ray Intersections

Modify call to MPSRayIntersector:

```swift
intersector.intersectionDataType = .distance

intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: 0,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: 0,
    rayCount: rayCount,
    accelerationStructure: accelerationStructure)
```
Finding Shadow Ray Intersections

Modify call to MPSRayIntersector:

```swift
intersector.intersectionDataType = .distance

intersector.encodeIntersection(commandBuffer: commandBuffer,
      intersectionType: .nearest,
      rayBuffer: rayBuffer,
      rayBufferOffset: 0,
      intersectionBuffer: intersectionBuffer,
      intersectionBufferOffset: 0,
      rayCount: rayCount,
      accelerationStructure: accelerationStructure)
```
Finding Shadow Ray Intersections

Modify call to MPSRayIntersector:

```swift
intersector.intersectionDataType = .distance

intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: 0,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: 0,
    rayCount: rayCount,
    accelerationStructure: accelerationStructure)
```
Finding Shadow Ray Intersections

Modify call to MPSRayIntersector:

```swift
intersector.intersectionDataType = .distance

intersector.encodeIntersection(commandBuffer: commandBuffer,
intersectionType: .any,
rayBuffer: rayBuffer,
rayBufferOffset: 0,
intersectionBuffer: intersectionBuffer,
intersectionBufferOffset: 0,
rayCount: rayCount,
accelerationStructure: accelerationStructure)
```
Shadow Rays

Generate Primary Rays → Intersector → Ray Buffer → Intersection Buffer → Shading

Intersection Buffer → Add Color to Image → Image

Shadow Ray Buffer → Intersector
Shadow Rays

Generate Primary Rays → Intersector → Ray Buffer → Intersector → Intersection Buffer → Shading

Add Color to Image

Image
Adding Ray Color to the Image

- Shadow Ray Buffer
- Intersection Buffer
Adding Ray Color to the Image

```c
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```

```c
struct ShadowIntersection {
    float distance;
};
```
Adding Ray Color to the Image

```c
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};
```

```c
struct ShadowIntersection {
    float distance;
};
```
Adding Ray Color to the Image

```plaintext
struct Ray {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
    float3 color;
};

struct ShadowIntersection {
    float distance;
};
```
Adding Ray Color to the Image

```c
struct Ray {
  packed_float3 origin;
  float minDistance;
  packed_float3 direction;
  float maxDistance;
  float3 color;
};

struct ShadowIntersection {
  float distance;
};
```
Primary Rays and Shading

Shadow Rays

Secondary Rays
Primary Rays and Shading

Shadow Rays

Secondary Rays
Secondary Rays

Simulate light bouncing around the scene
Secondary Rays

Simulate light bouncing around the scene
Iterate, continuing in random direction at each step
Secondary Rays

1. Generate Primary Rays
2. Intersector
3. Ray Buffer
4. Intersection Buffer
5. Shading
6. Add Color to Image
7. Shadow Ray Buffer
8. Intersector
9. Intersection Buffer
10. Image
Secondary Rays

- Generate Primary Rays
- Intersector
- Shadow Ray Buffer
- Intersector
- Ray Buffer
- Intersector
- Intersection Buffer
- Intersection Buffer
- Add Color to Image
- Image
Secondary Rays

Generate Primary Rays → Intersector → Intersection Buffer → Shading

Generate Primary Rays → Intersector → Intersection Buffer → Add Color to Image → Image

Generate Primary Rays → Intersector → Intersection Buffer → Ray Buffer → Intersector → Intersection Buffer → Add Color to Image → Image

Generate Primary Rays → Intersector → Intersection Buffer → Ray Buffer → Intersector → Intersection Buffer → Add Color to Image → Image
Updating Secondary Rays

Update secondary rays during each iteration:

```java
ray.origin = intersectionPoint;
ray.direction = getRandomDirection(surfaceNormal);
ray.color *= surfaceColor;
```
Updating Secondary Rays

Update secondary rays during each iteration:

```java
ray.origin = intersectionPoint;
ray.direction = getRandomDirection(surfaceNormal);
ray.color *= surfaceColor;
```
Updating Secondary Rays

Update secondary rays during each iteration:

```python
ray.origin = intersectionPoint;
ray.direction = getRandomDirection(surfaceNormal);
ray.color *= surfaceColor;
```
Updating Secondary Rays

Update secondary rays during each iteration:

```python
ray.origin = intersectionPoint;
ray.direction = getRandomDirection(surfaceNormal);
ray.color *= surfaceColor;
```
Primary Rays and Shading

Shadow Rays

Secondary Rays
Sample Code

This app is available as a sample

Add your own geometry, light sources, camera model, shading model

Refer to documentation for more information
Extending to Multiple GPUs

Wayne Lister, GPU Software Engineer
Multi-GPU Ray Tracing

Split work across GPUs
Multi-GPU Ray Tracing

Split work across GPUs

Copy data between GPUs
Multi-GPU Ray Tracing

- Split work across GPUs
- Copy data between GPUs
- Synchronize execution
Splitting Work Across GPUs

Rendering
Splitting Work Across GPUs

Rendering

GPU 0

GPU 1

GPU 2
Splitting Work Across GPUs
Splitting Work Across GPUs

- Scene Data
- Acceleration Structure
- Vertex Buffer
  - GPU 0
  - GPU 1
  - GPU 2
- Rendering
- Composition
Splitting Work Across GPUs

Scene Data

Acceleration Structure
Vertex Buffer...

Rendering

GPU 0
GPU 1
GPU 2

Composition
Splitting Work Across GPUs
let copy = accelerationStructure.copy(with: nil, device: newDevice)
copy.vertexBuffer = copyBuffer(accelerationStructure.vertexBuffer, device: newDevice)
let copy = accelerationStructure.copy(with: nil, device: newDevice)
copy.vertexBuffer = copyBuffer(accelerationStructure.vertexBuffer, device: newDevice)
Copying Scene Data

```swift
let copy = accelerationStructure.copy(with: nil, device: newDevice)
copy.vertexBuffer = copyBuffer(accelerationStructure.vertexBuffer, device: newDevice)
```
Splitting Work Across GPUs
Splitting Work Across GPUs

Scene Data

Acceleration Structure
Vertex Buffer
...

Rendering

GPU 0
GPU 1
GPU 2

Composition
Splitting Work Across GPUs

Scene Data

Acceleration Structure
Vertex Buffer
...

Rendering

GPU 0
GPU 1
GPU 2

Composition
Creating a Shared CPU Allocation

Device A

MTLBuffer (Private)

Device B

MTLBuffer (Private)
Creating a Shared CPU Allocation

- Device A
  - MTLBuffer (Private)
- CPU Allocation
- Device B
  - MTLBuffer (Private)
Creating a Shared CPU Allocation

Device A
MTLBuffer
(Private)

CPU Allocation

Device B
MTLBuffer
(Private)
Creating a Shared CPU Allocation
Creating a Shared CPU Allocation

Device A
MTLBuffer (Private)

MTLBuffer (Shared)

MTLBuffer (Shared)

Device B

MTLBuffer (Shared)

MTLBuffer (Private)

Blit

Blit
Creating a Shared CPU Allocation

```swift
let bufferA = deviceA.makeBuffer(length: bufferLength,
       options: .shared)!

let bufferB = deviceB.makeBuffer(bytesNoCopy: bufferA.contents(),
       length: bufferA.length,
       options: .shared
       deallocator: nil)!
```
Creating a Shared CPU Allocation

```swift
let bufferA = deviceA.makeBuffer(length: bufferLength,
    options: .shared)!

let bufferB = deviceB.makeBuffer(bytesNoCopy: bufferA.contents(),
    length: bufferA.length,
    options: .shared,
    deallocator: nil)!
```
Creating a Shared CPU Allocation

```swift
let bufferA = deviceA.makeBuffer(length: bufferLength,
    options: .shared)!

let bufferB = deviceB.makeBuffer(bytesNoCopy: bufferA.contents(),
    length: bufferA.length,
    options: .shared
    deallocator: nil)!
```
Synchronizing with Metal Events

Execution Timeline

GPU A
- Render region
- Blit region to shared buffer

GPU B
- Render region
- Blit region from shared buffer
Synchronizing with Metal Events

Execution Timeline

GPU A
- Render region
- Blit region to shared buffer

GPU B
- Render region
- Blit region from shared buffer
Synchronizing with Metal Events

Execution Timeline:

- **GPU A**
  - Render region
  - Blit region to shared buffer

- **GPU B**
  - Render region
  - Blit region from shared buffer
Synchronizing with Metal Events

Execution Timeline

GPU A
- Render region
- Blit region to shared buffer

GPU B
- Render region
- Blit region from shared buffer
Synchronizing with Metal Events

Execution Timeline

<table>
<thead>
<tr>
<th>GPU A</th>
<th>Render region</th>
<th>Blit region to shared buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU B</td>
<td>Render region</td>
<td>Blit region from shared buffer</td>
</tr>
</tbody>
</table>
Synchronizing with Metal Events

Execution Timeline

GPU A
- Render region
- Blit region to shared buffer

GPU B
- Render region
- Blit region from shared buffer
Synchronizing with Metal Events

Execution Timeline:

GPU A:
- Render region
- Blit region to shared buffer

GPU B:
- Render region
- Blit region from shared buffer
Synchronizing with Metal Events

Execution Timeline

GPU A
- Render region
- Blit region to shared buffer

GPU B
- Render region
- Wait
- Blit region from shared buffer

[commandBuffer encodeWaitForEvent:value]
Synchronizing with Metal Events

Execution Timeline

GPU A
- Render region
- Blit region to shared buffer

GPU B
- Render region
- Wait
- Blit region from shared buffer

[commandBuffer encodeSignalEvent:value]
[commandBuffer encodeWaitForEvent:value]
Load Balancing

GPUs can have different performance
Load Balancing

GPUs can have different performance
Some regions are more complex to render
Load Balancing

Adjust region sizes to keep GPUs fully utilized

Fixed Partitions
Load Balancing

Adjust region sizes to keep GPUs fully utilized
Timing GPU Work

```swift
commandBuffer.addCompletedHandler { commandBuffer in
    completionHandler[commandBuffer] = mach_absolute_time()
}
```
Demo
Summary

Accelerates ray/triangle intersection on the GPU
Summary

Accelerates ray/triangle intersection on the GPU

Optimized for macOS and iOS
Summary

Accelerates ray/triangle intersection on the GPU
Optimized for macOS and iOS
Scales across multiple GPUs
More Information

https://developer.apple.com/wwdc18/606