Ray Tracing with Metal

Sean James, GPU Software Engineer
Wayne Lister, GPU Software Engineer
Matt Kaplan, GPU Software Engineer
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
Ray Tracing

Used in offline rendering to model individual rays of light

- Reflections and refraction
- Shadows
- Ambient occlusion
- Global illumination

Sponza Atrium © 2009 Frank Meinl, Crytek, https://casual-effects.com/data/  Ray traced with Metal https://creativecommons.org/licenses/by/3.0/
Ray Tracing in Real-Time

Dynamic scenes
Ray Tracing in Real-Time

Dynamic scenes

Performance
Ray Tracing in Real-Time

Dynamic scenes
Performance
Denoising
Ray Tracing with Metal

Generate Rays
Ray Tracing with Metal

Generate Rays → Intersect with Scene
Ray Tracing with Metal

1. Generate Rays
2. Intersect with Scene
3. Shading
4. Image
Ray Tracing with Metal

Generate Rays -> Intersect with Scene

Intersect with Scene -> Shading

Shading -> Image

Rays

Intersections

Additional Rays
Ray Tracing with Metal

1. Generate Rays
2. Intersect with Scene
3. Shading
4. Image
MPSRayIntersector

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

Build an acceleration structure over triangles in a vertex buffer
MPSRayIntersector

Build an *acceleration structure* over triangles in a vertex buffer

Pass acceleration structure to intersector
MPSRayIntersector

Build an *acceleration structure* over triangles in a vertex buffer

Pass acceleration structure to intersector

Now builds on the GPU
Ray Tracing with Metal

Generate Rays → Ray Buffer
Ray Tracing with Metal

Generate Rays → Ray Buffer → Intersector → Intersection Buffer
Ray Tracing with Metal

- Generate Rays
- Ray Buffer
- Intersector
  - Intersector Buffer
  - Acceleration Structure
  - Vertex Buffer
Ray Tracing with Metal

- Generate Rays
- Intersector
- Shading
- Acceleration Structure
- Image
- Vertex Buffer

Flow:
- Generate Rays → Ray Buffer → Intersector → Intersection Buffer → Shading
- Intersector uses Acceleration Structure
- Shading produces Image
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look

Advances in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Ray Tracing in AR Quick Look
Dynamic Scenes

Three types of animation
Dynamic Scenes

Three types of animation

• Camera movement
Dynamic Scenes

Three types of animation

• Camera movement
• Vertex animation
Dynamic Scenes

Three types of animation
• Camera movement
• Vertex animation
• Rigid body animation
Vertex Animation

Deformation and skinned animation
Vertex Animation

Deformation and skinned animation
Vertex Animation

Deformation and skinned animation

Need to update acceleration structure
Vertex Animation

Deformation and skinned animation

Need to update acceleration structure

Objects tend to retain their shape
Vertex Animation
Vertex Animation
Vertex Animation
Refitting

Much faster than building from scratch
Refitting

Much faster than building from scratch

Runs on the GPU
Refitting

Much faster than building from scratch

Runs on the GPU

Can’t add or remove geometry
Refitting

Much faster than building from scratch

Runs on the GPU

Can’t add or remove geometry

Potentially degrades acceleration structure quality
Refitting

Enable refitting before building acceleration structure:

```swift
accelerationStructure.usage = .refit
```
Refitting

Enable refitting before building acceleration structure:

```swift
accelerationStructure.usage = .refit
```

Encode refit operation into a command buffer:

```swift
accelerationStructure.encodeRefit(commandBuffer: commandBuffer)
```
Rigid Body Animation

Most geometry only moves rigidly or not at all
Rigid Body Animation

Most geometry only moves rigidly or not at all
Rigid Body Animation

Most geometry only moves rigidly or not at all
May have multiple copies of the same objects
Two-Level Acceleration Structures

Object A

Object B

Object C
Two-Level Acceleration Structures

Object A

Object A

Object B

Object B

Object C
Two-Level Acceleration Structures
Two-Level Acceleration Structures

Transformation matrices:

<table>
<thead>
<tr>
<th>float4x4</th>
<th>float4x4</th>
<th>float4x4</th>
<th>float4x4</th>
<th>float4x4</th>
</tr>
</thead>
</table>

Objects:
- Object A
- Object B
- Object C
Two-Level Acceleration Structures

Transformation matrices:

<table>
<thead>
<tr>
<th>float4x4</th>
<th>float4x4</th>
<th>float4x4</th>
<th>float4x4</th>
<th>float4x4</th>
</tr>
</thead>
</table>

Acceleration structure indices:

| 0 | 1 | 0 | 2 | 1 |

A  B  C

Objects:

- Object A
- Object B
- Object C
Two-Level Acceleration Structures

Transformation matrices:

| float4x4 | float4x4 | float4x4 | float4x4 | float4x4 |

Acceleration structure indices:

| 0 | 1 | 0 | 2 | 1 |

A  B  C
Two-Level Acceleration Structures

Build triangle acceleration structures:

```swift
let group = MPSAccelerationStructureGroup(device: device)
var accelerationStructures : [MPSTriangleAccelerationStructure] = []

// for each unique object:
    let triangleAccelerationStructure = MPSTriangleAccelerationStructure(group: group)
    // configure properties...
    triangleAccelerationStructure.rebuild()

accelerationStructures.append(triangleAccelerationStructure)
```
Two-Level Acceleration Structures

Build triangle acceleration structures:

```swift
let group = MPSAccelerationStructureGroup(device: device)

var accelerationStructures : [MPSTriangleAccelerationStructure] = []

// for each unique object:
    let triangleAccelerationStructure = MPSTriangleAccelerationStructure(group: group)
    // configure properties...
    triangleAccelerationStructure.rebuild()

    accelerationStructures.append(triangleAccelerationStructure)
```
Two-Level Acceleration Structures

Build triangle acceleration structures:

```swift
let group = MPSAccelerationStructureGroup(device: device)
var accelerationStructures : [MPSTriangleAccelerationStructure] = []

// for each unique object:
let triangleAccelerationStructure = MPSTriangleAccelerationStructure(group: group)
// configure properties...
triangleAccelerationStructure.rebuild()

accelerationStructures.append(triangleAccelerationStructure)
```
Two-Level Acceleration Structures

Build triangle acceleration structures:

```swift
let group = MPSAccelerationStructureGroup(device: device)
var accelerationStructures : [MPSTriangleAccelerationStructure] = []

// for each unique object:
let triangleAccelerationStructure = MPSTriangleAccelerationStructure(group: group)
// configure properties...
triangleAccelerationStructure.rebuild()

accelerationStructures.append(triangleAccelerationStructure)
```
Two-Level Acceleration Structures

Create instance acceleration structure:

```swift
let instanceAccelerationStructure = MPSInstanceAccelerationStructure(group: group)

instanceAccelerationStructure.accelerationStructures = accelerationStructures
instanceAccelerationStructure.transformBuffer = transformBuffer
instanceAccelerationStructure.instanceBuffer = instanceBuffer
instanceAccelerationStructure.instanceCount = instanceCount
```
Two-Level Acceleration Structures

Create instance acceleration structure:

```swift
let instanceAccelerationStructure = MPSInstanceAccelerationStructure(group: group)

instanceAccelerationStructure.accelerationStructures = accelerationStructures
instanceAccelerationStructure.transformBuffer = transformBuffer
instanceAccelerationStructure.instanceBuffer = instanceBuffer
instanceAccelerationStructure.instanceCount = instanceCount
```
Create instance acceleration structure:

```swift
let instanceAccelerationStructure = MPSInstanceAccelerationStructure(group: group)

instanceAccelerationStructure.accelerationStructures = accelerationStructures
instanceAccelerationStructure.transformBuffer = transformBuffer
instanceAccelerationStructure.instanceBuffer = instanceBuffer
instanceAccelerationStructure.instanceCount = instanceCount
```
Two-Level Acceleration Structures

Create instance acceleration structure:

```swift
let instanceAccelerationStructure = MPSInstanceAccelerationStructure(group: group)

instanceAccelerationStructure.accelerationStructures = accelerationStructures
instanceAccelerationStructure.transformBuffer = transformBuffer
instanceAccelerationStructure.instanceBuffer = instanceBuffer
instanceAccelerationStructure.instanceCount = instanceCount
```
Two-Level Acceleration Structures

Create instance acceleration structure:

```swift
let instanceAccelerationStructure = MPSInstanceAccelerationStructure(group: group)

instanceAccelerationStructure.accelerationStructures = accelerationStructures
instanceAccelerationStructure.transformBuffer = transformBuffer
instanceAccelerationStructure.instanceBuffer = instanceBuffer
instanceAccelerationStructure.instanceCount = instanceCount
```
Two-Level Acceleration Structures

Create instance acceleration structure:

```swift
let instanceAccelerationStructure = MPSInstanceAccelerationStructure(group: group)

instanceAccelerationStructure.accelerationStructures = accelerationStructures
instanceAccelerationStructure.transformBuffer = transformBuffer
instanceAccelerationStructure.instanceBuffer = instanceBuffer
instanceAccelerationStructure.instanceCount = instanceCount
```

Rebuild when scene changes:

```swift
instanceAccelerationStructure.rebuild()
```
Denoising
Denoising
Denoising
Denoising

Noisy Image -> Denoiser -> Clean Image
Denoising

- Normals
- Depth
- Noisy Image
- Denoiser
- Clean Image
Denoising

Noisy Image → Denoiser → Clean Image
Denoising
Denoising

- Motion Vectors
- Previous Frame

Denoiser

Clean Image
Denoising

- Motion Vectors
- Noisy Image
- Previous Frame

Denoiser

Clean Image
High-quality “Spatiotemporal Variance-Guided Filtering” denoising algorithm
High-quality “Spatiotemporal Variance-Guided Filtering” denoising algorithm

MPSSVGFDenoiser coordinates denoising process
MPSSVGF

High-quality “Spatiotemporal Variance-Guided Filtering” denoising algorithm

MPSSVGF Denoiser coordinates denoising process

Low-level control
High-quality “Spatiotemporal Variance-Guided Filtering” denoising algorithm

MPSSVGFDenoiser coordinates denoising process

Low-level control
High-quality "Spatiotemporal Variance-Guided Filtering" denoising algorithm

MPSSVGF D denoiser coordinates denoising process

Low-level control
Create denoiser:

// Allocate the denoising kernels
let svgf = MPSSVGF(device: device)

// Configure SVGF properties

// Create a custom texture allocator or use the default allocator
let textureAllocator = MPSSVGFDefaultTextureAllocator(device: device)

// Create the denoiser object
let denoiser = MPSSVGFDDenoiser(SVGF: svgf, textureAllocator: textureAllocator)
Create denoiser:

```swift
// Allocate the denoising kernels
let svgf = MPSSVGF(device: device)

// Configure SVGF properties

// Create a custom texture allocator or use the default allocator
let textureAllocator = MPSSVGFDefaultTextureAllocator(device: device)

// Create the denoiser object
let denoiser = MPSSVGFDDenoiser(SVGF: svgf, textureAllocator: textureAllocator)
```
Create denoiser:

// Allocate the denoising kernels
let svgf = MPSSVGF(device: device)

// Configure SVGF properties

// Create a custom texture allocator or use the default allocator
let textureAllocator = MPSSVGFDefaultTextureAllocator(device: device)

// Create the denoiser object
let denoiser = MPSSVGFDDenoiser(SVGF: svgf, textureAllocator: textureAllocator)
Create denoiser:

```swift
// Allocate the denoising kernels
let svgf = MPSSVGF(device: device)

// Configure SVGF properties

// Create a custom texture allocator or use the default allocator
let textureAllocator = MPSSVGFDefaultTextureAllocator(device: device)

// Create the denoiser object
let denoiser = MPSSVGFDEnoiser(SVGF: svgf, textureAllocator: textureAllocator)
```
Encode to command buffer:

denoiser.sourceTexture = textureToDenoise
denoiser.depthNormalTexture = depthNormalTexture
denoiser.previousDepthNormalTexture = previousDepthNormalTexture
denoiser.motionVectorTexture = motionVectorTexture

denoiser.encode(commandBuffer: commandBuffer)

let denoisedTexture = denoiser.destinationTexture
MPSSVGFDenoiser

Encode to command buffer:

denoiser.sourceTexture = textureToDenoise
denoiser.depthNormalTexture = depthNormalTexture
denoiser.previousDepthNormalTexture = previousDepthNormalTexture
denoiser.motionVectorTexture = motionVectorTexture

denoiser.encode(commandBuffer: commandBuffer)

let denoisedTexture = denoiser.destinationTexture
MPSSVGFDenoiser

Encode to command buffer:

denoiser.sourceTexture = textureToDenoise
denoiser.depthNormalTexture = depthNormalTexture
denoiser.previousDepthNormalTexture = previousDepthNormalTexture
denoiser.motionVectorTexture = motionVectorTexture

denoiser.encode(commandBuffer: commandBuffer)

let denoisedTexture = denoiser.destinationTexture
MPSSVGFDenoiser

Encode to command buffer:

denoiser.sourceTexture = textureToDenoise
denoiser.depthNormalTexture = depthNormalTexture
denoiser.previousDepthNormalTexture = previousDepthNormalTexture
denoiser.motionVectorTexture = motionVectorTexture

denoiser.encode(commandBuffer: commandBuffer)

let denoisedTexture = denoiser.destinationTexture
Ray Tracing with Metal

Ray/triangle intersection
Dynamic scenes
Denoising
Ray Tracing in Practice
Hard Shadows
Hard Shadows
Hybrid Rendering
Hybrid Rendering

G-Buffer → Shading → Shaded Image
Hybrid Rendering

G-Buffer → Ray Generation → Ray Buffer → Shading → Shaded Image
Hybrid Rendering

- G-Buffer
- Ray Generation
- Ray Buffer
- Intersector
- Intersection Buffer
- Shading
- Shaded Image
Hybrid Rendering

- G-Buffer
- Shading
- Shaded Image
- Ray Generation
- Intersector
- Intersection Buffer
- Ray Buffer
Hybrid Rendering

G-Buffer → Shading → Shaded Image

Ray Generation → Ray Buffer → Intersector → Intersection Buffer
Ray Generation

```cpp
MPSRayOriginDirection ray;

ray.origin = worldPosition + worldNormal * SURFACE_BIAS;
ray.direction = directionToLight;

rayBuffer[outputIndex] = ray;
```
Ray Coherency

Metal processes rays in the order you specify them.

Block linear layout can improve performance.

Row Linear
Ray Coherency

Metal processes rays in the order you specify them.

Block linear layout can improve performance.

Row Linear
Ray Coherency

Metal processes rays in the order you specify them

Block linear layout can improve performance
Disabling Rays

Not all pixels need a shadow ray
• Background pixels
• Surfaces facing away from the Sun

Disable rays by setting `maxDistance < 0.0`
Hard Shadows
Soft Shadows
Soft Shadows

Extend a cone from the surface towards the Sun

Generate ray directions randomly within this cone
Soft Shadows

Extend a cone from the surface towards the Sun

Generate ray directions randomly within this cone
Soft Shadows

Extend a cone from the surface towards the Sun

Generate ray directions randomly within this cone
Shadow Term
Ambient Occlusion
Use rays to estimate how much ambient light reaches a surface.

Falloff based on angle and intersection distance.
Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

Falloff based on angle and intersection distance
Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

Falloff based on angle and intersection distance
Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

Falloff based on angle and intersection distance
Importance Sampling

Use importance sampling to generate rays
Fewer rays for same visual quality
Importance Sampling

Hemisphere sampling
Importance Sampling

Cosine sampling
Importance Sampling

Cosine sampling
Importance Sampling

Cosine sampling
Distance sampling
Importance Sampling

Cosine sampling

Distance sampling
Parameter Space

Points in 2D parameter space map to 3D ray distributions
Low Discrepancy Sequences

Distribute points evenly in parameter space

Low discrepancy sequence (e.g. Halton, Sobol)
Low Discrepancy Sequences

Distribute points evenly in parameter space

Low discrepancy sequence (e.g. Halton, Sobol)
Low Discrepancy Sequences

Distribute points evenly in parameter space

Low discrepancy sequence (e.g. Halton, Sobol)
Low Discrepancy Sequences

Distribute points evenly in parameter space

Low discrepancy sequence (e.g. Halton, Sobol)

Random

Halton (2, 3)
Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels
Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels
Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels
Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels

White Noise
Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels

White Noise

Blue Noise
Ray Traced Result
Ray Traced Result
Global Illumination
Global Illumination

What is Global Illumination?
Memory
Ray Lifetime
Debugging
Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging
Global Illumination
Global Illumination
Global Illumination

1. Multiple light sources scatter light throughout the scene, illuminating objects.
2. Light reflection and refraction on surfaces create indirect illumination.
3. Shadow cast by objects and light blocking boundaries.
4. Diffuse, glossy, and metallic reflections enhance realism in the rendered image.
Global Illumination

1. [Diagram showing light sources and their impact on objects]
2. [Diagram showing light sources and their impact on objects]
3. [Diagram showing light sources and their impact on objects]
4. [Diagram showing light sources and their impact on objects]
Global Illumination
Global Illumination
Global Illumination
Global Illumination
Global Illumination
Global Illumination
Global Illumination

1. Generate Rays
2. Intersector
3. Process Results
4. Generate Shadow Rays
5. Intersector
6. Add Light to Final Image
7. Final Image
Global Illumination

Generate Rays → Intersector → Process Results

Generate Shadow Rays → Intersector → Add Light to Final Image → Final Image
Global Illumination

What is Global Illumination?

Memory
Ray Lifetime
Debugging
Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging
Data Requirements

Passing data between pipeline iterations
Hybrid methods didn’t need to do this
Can still use constant indexing for all the buffers

| Ray Position |
| Ray Direction |
| Ray Type |
| Index of Refraction |
| Hit Surface Properties |
| Ray Color |

...
Memory Usage

Memory footprint grows quickly
The ray buffer alone for a 4K image is 250MB
Our demo uses 80B per ray
Can quickly exceed available GPU memory
Memory Usage

Solution: Batch your rays up into smaller groups or tiles

Limit the number of rays you launch simultaneously
Bandwidth Overload

Paging data in and out is a major limiting factor.

For a 4K image — 8,294,400 rays per pass

5GB of data per iteration at 80B per ray!

May use more with supersampling
Reducing Bandwidth Usage

- Coalesce loads and stores
- Use smaller data types where possible
- Split structs
Reducing Bandwidth Usage

Counterintuitive — use your own origin and direction buffers

Avoid load/stores of structs members you don’t need

```c
struct MPSRayOriginMinDistanceDirectionMaxDistance {
    packed_float3 origin;
    float minDistance;
    packed_float3 direction;
    float maxDistance;
};
```

```c
packed_half3 *origin;
packed_half3 *direction;
```
Reduce register pressure

- Track simultaneously live variables
- Don’t hold onto structs
- Be careful with loop counters, function calls
Textures

Can’t predict what surface a ray will intersect

Sponza scene has 76 textures

Quickly run out of binding slots
struct Material {
    texture2d<float> texture;
    // ...
};

kernel void shadingKernel(const device Material *materials,
                           const device Intersection *intersections,
                           /* ... */)
{
    unsigned int primitiveIndex = intersections[tid].primitiveIndex;
    const device Material & material = materials[primitiveIndex];
    texture2d<float> texture = material.texture;
}
Argument Buffers

```cpp
struct Material {
    texture2d<float> texture;
    // ...
};

kernel void shadingKernel(const device Material *materials,
                          const device Intersection *intersections,
                          /* ... */)
{
    unsigned int primitiveIndex = intersections[tid].primitiveIndex;
    const device Material & material = materials[primitiveIndex];
    texture2d<float> texture = material.texture;
}
```
**Argument Buffers**

```cpp
struct Material {
    texture2d<float> texture;
    // ...
};

kernel void shadingKernel(const device Material *materials,
                           const device Intersection *intersections,
                           /* ... */)
{
    unsigned int primitiveIndex = intersections[tid].primitiveIndex;
    const device Material & material = materials[primitiveIndex];
    texture2d<float> texture = material.texture;
}
```
Argument Buffers

```c
struct Material {
    texture2d<float> texture;
    // ...
};

kernel void shadingKernel(const device Material *materials,
                           const device Intersection *intersections,
                           /* ... */)
{
    unsigned int primitiveIndex = intersections[tid].primitiveIndex;
    const device Material & material = materials[primitiveIndex];
    texture2d<float> texture = material.texture;
}
```
struct Material {
    texture2d<float> texture;
    // ...
};

kernel void shadingKernel(const device Material *materials,
               const device Intersection *intersections,
               /* ... */)
{
    unsigned int primitiveIndex = intersections[tid].primitiveIndex;
    const device Material & material = materials[primitiveIndex];
    texture2d<float> texture = material.texture;
}
struct Material {
    texture2d<float> texture;
    // ...
};

kernel void shadingKernel(const device Material *materials,
                           const device Intersection *intersections,
                           /* ... */)
{
    unsigned int primitiveIndex = intersections[tid].primitiveIndex;
    const device Material & material = materials[primitiveIndex];
    texture2d<float> texture = material.texture;
}
Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging
Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging
Eliminating Inactive Rays

Rays can stop contributing

• Leave the scene

• Ray no longer carries enough light to make a measurable impact

• Total internal reflection for transparent surfaces
Inactive Rays

First iteration: 100% of rays active

0 1 2 3 4 6 6 7
Ray Buffer

Venice Sunset Environment Map © Hdr Haven: hdrhaven.com/hdr/?c=nature&h=venice_sunset, creativecommons.org/publicdomain/zero/1.0/
Inactive Rays

Second iteration: 57% of rays active

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>

| 0 | 2 | 4 | 5 | 7 |
Inactive Rays

Third iteration: 43% of rays active

```
0 1 2 3 4 5 6 7
```

```
0   2 4 5 7
```

```
2   4 5 7
```
Inactive Rays

Fourth iteration: 32% of rays active

0 1 2 3 4 5 6 7
0 2 4 5 7
2 4 5 7
2 4 7
Inactive Rays

Fifth iteration: 23% of rays active
Inactive Rays

Threadgroups become sparsely utilized

Ray intersector must still process inactive rays

Control flow statements to cull inactive rays
Ray Compaction

Only add active rays to the next ray buffer

Threadgroups are fully utilized

Also works for shadow rays
Ray Compaction

Only add active rays to the next ray buffer

Threadgroups are fully utilized

Also works for shadow rays
Ray Compaction

Buffer indices no longer map to constant pixel locations

Need to track pixel coordinates for each ray
Ray Compaction

Use atomics to get a unique index per ray in the output buffer:

```c
kernel void compactionKernel(device atomic_uint & outgoingRayCount,
                               device Ray & outgoingRays,
                               /* ... */)
{
    unsigned int outgoingRayIndex =
        atomic_fetch_add_explicit(&outgoingRayCount, 1, memory_order_relaxed);

    // Setup ray

    outgoingRays[outgoingRayIndex] = ray;
}
```
Ray Compaction

Use atomics to get a unique index per ray in the output buffer:

```c
kernel void compactionKernel(device atomic_uint & outgoingRayCount,
    device Ray & outgoingRays,
    /* ... */)
{
    unsigned int outgoingRayIndex =
        atomic_fetch_add_explicit(&outgoingRayCount, 1, memory_order_relaxed);

    // Setup ray
    outgoingRays[outgoingRayIndex] = ray;
}
```
Ray Compaction

Use atomics to get a unique index per ray in the output buffer:

```c
kernel void compactionKernel(device atomic_uint & outgoingRayCount,
                               device Ray & outgoingRays,
                               /* ... */)
{
    unsigned int outgoingRayIndex =
        atomic_fetch_add_explicit(&outgoingRayCount, 1, memory_order_relaxed);

    // Setup ray
    outgoingRays[outgoingRayIndex] = ray;
}
```
Ray Compaction

Launch one thread per ray using indirect dispatch:

```swift
// Fill out MTLDispatchThreadgroupsIndirectArguments in indirectBuffer in a compute kernel

computeEncoder.dispatchThreadgroups(indirectBuffer: indirectBuffer,
                                     indirectBufferOffset: indirectBufferOffset,
                                     threadsPerThreadgroup: threadsPerThreadgroup)
```
Launch one thread per ray using indirect dispatch:

```swift
// Fill out MTLDispatchThreadgroupsIndirectArguments in indirectBuffer in a compute kernel
computeEncoder.dispatchThreadgroups(indirectBuffer: indirectBuffer,
                                      indirectBufferOffset: indirectBufferOffset,
                                      threadsPerThreadgroup: threadsPerThreadgroup)
```
Ray Compaction

Launch one thread per ray using indirect dispatch:

```swift
// Fill out MTLDiplayThreadgroupsIndirectArguments in indirectBuffer in a compute kernel

computeEncoder.dispatchThreadgroups(indirectBuffer: indirectBuffer,
indirectBufferOffset: indirectBufferOffset,
threadsPerThreadgroup: threadsPerThreadgroup)
```
Ray Compaction

Indirect ray intersection:

```swift
intersector.encodeIntersection(commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: rayBufferOffset,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: intersectionBufferOffset,
    rayCountBuffer: outgoingRayCount,
    rayCountBufferOffset: outgoingRayCountOffset,
    accelerationStructure: accelerationStructure)
```
Ray Compaction

Indirect ray intersection:

```swift
intersector.encodeIntersection(
    commandBuffer: commandBuffer,
    intersectionType: .nearest,
    rayBuffer: rayBuffer,
    rayBufferOffset: rayBufferOffset,
    intersectionBuffer: intersectionBuffer,
    intersectionBufferOffset: intersectionBufferOffset,
    rayCountBuffer: outgoingRayCount,
    rayCountBufferOffset: outgoingRayCountOffset,
    accelerationStructure: accelerationStructure)
```
Global Illumination
Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging
Global Illumination

What is Global Illumination?
Memory
Ray Lifetime
Debugging
Debugging Image Corruption

Xcode makes this a breeze to debug

Frame capture
Debugging Image Corruption

Xcode makes this a breeze to debug

Frame capture
Debugging with Xcode
Debugging with Xcode
Debugging with Xcode
Debugging with Xcode
Debugging with Xcode
Debugging with Xcode

```swift
float3 resolved = pixelAccumulation[linearAddress];
```
Debugging with Xcode

```c
Float3 resolved = pixelAccumulation[linearAddress] / pixelSampleCount[linearAddress];
```
Debugging with Xcode
Debugging with Xcode
Performance Tuning with Xcode

How to profile code changes?

Xcode profiling tools are very powerful

```c
struct Surface {
    float3 baseColor;
    float shininess;
    float roughness;
    float emissive;
    float3 transmission; // transparency
    float indexOfRefraction;
};
```
Performance Tuning With Xcode

```c
struct Surface {
    float3 baseColor;
    float shininess;
    float roughness;
    float emissive;
};

struct SurfaceRefraction {
    float3 transmission; // transparency
    float indexOfRefraction;
};
```
struct Surface {
    half3 baseColor;
    half shininess;
    half roughness;
    half emissive;
};

struct SurfaceRefraction {
    half3 transmission; // transparency
    half indexOfRefraction;
};
<table>
<thead>
<tr>
<th>Problem</th>
<th>Before Change</th>
<th>After Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Texture

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture Unit Time</td>
<td>99%</td>
<td>99%</td>
<td>2%</td>
<td>99%</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Texture Unit Stall Time</td>
<td>70%</td>
<td>70%</td>
<td>0%</td>
<td>88%</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Texture Cache Miss Rate</td>
<td>19.79%</td>
<td>19.8%</td>
<td>81.82%</td>
<td>100%</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Sampler Busy</td>
<td>37%</td>
<td>37%</td>
<td>0%</td>
<td>91%</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

### Memory

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 Throughput</td>
<td>2.92</td>
<td>2.93</td>
<td>0.02</td>
<td>3.32</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>L2 Bandwidth</td>
<td>33.39</td>
<td>33.48</td>
<td>0.31</td>
<td>57.65</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Color Block Bandwidth</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>357.43</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

### Additional Table

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture Unit Time</td>
<td>93%</td>
<td>94%</td>
<td>2%</td>
<td>99%</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Texture Unit Stall Time</td>
<td>54.01%</td>
<td>55%</td>
<td>0%</td>
<td>88%</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Texture Cache Miss Rate</td>
<td>74.59%</td>
<td>74.55%</td>
<td>81.82%</td>
<td>100%</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Sampler Busy</td>
<td>39%</td>
<td>40%</td>
<td>0%</td>
<td>91%</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

### Additional Memory

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 Throughput</td>
<td>1.17</td>
<td>1.18</td>
<td>0.02</td>
<td>2</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>L2 Bandwidth</td>
<td>49.51</td>
<td>49.61</td>
<td>0.31</td>
<td>57.87</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Color Block Bandwidth</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>354.18</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
Potential Hotspots/Bottlenecks

Texture unit is heavily stalled - 70%

Potential Hotspots/Bottlenecks

No potential bottlenecks detected
MaxTotalThreadsPerThreadgroup = (NSUInteger) 1024
Demo
Interleaved Tiling

Divide screen into tiles, render on different GPUs
Interleaved Tiling

Divide screen into tiles, render on different GPUs
Interleaved Tiling

Divide screen into tiles, render on different GPUs
Load Balancing

Smaller tiles distribute rendering more evenly across GPUs
Load Balancing

Smaller tiles distribute rendering more evenly across GPUs
Load Balancing

Smaller tiles distribute rendering more evenly across GPUs
Load Balancing

Smaller tiles distribute rendering more evenly across GPUs

Pseudo-random assignment avoids correlation with the scene
Load Balancing

Smaller tiles distribute rendering more evenly across GPUs

Pseudo-random assignment avoids correlation with the scene

Same GPU renders the same tiles each frame
Choosing a Tile Size

How small should I make my tiles?

Experiment: vary tile size, measure performance
Choosing a Tile Size

![Bar chart showing relative performance of different tile sizes. The x-axis represents tile sizes (1x1, 2x2, 4x4, 8x8, 16x16, 32x32, 256x256, 512x512). The y-axis represents relative performance. The chart shows that larger tile sizes generally have higher relative performance.]
Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU
Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU
Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU
Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU
Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU
Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

<table>
<thead>
<tr>
<th>0.4</th>
<th>0.55</th>
<th>0.22</th>
<th>0.78</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.64</td>
<td>0.12</td>
<td>0.35</td>
<td>0.89</td>
</tr>
<tr>
<td>0.1</td>
<td>0.39</td>
<td>0.72</td>
<td>0.61</td>
</tr>
<tr>
<td>0.79</td>
<td>0.2</td>
<td>0.9</td>
<td>0.42</td>
</tr>
</tbody>
</table>

GPU 0  GPU 1  GPU 2  GPU 3
Tile Assignment

Adjust ranges to control distribution
Tile Assignment

Adjust ranges to control distribution
Tile Assignment

Adjust ranges to control distribution
Data Transfers

CPU

AMD Radeon Pro Vega II Duo

GPU 1

GPU 2

GPU 3

GPU 4

Display

Metal for Pro Apps

WWDC 2019
Data Transfers
Data Transfers

CPU

AMD Radeon Pro Vega II Duo

GPU 1

GPU 2

Infinity Fabric Link

Display

GPU 3

GPU 4

PCIe

PCIe

PCIe
Data Transfers

GPU 0
- Render tiles for frame 0
- Render tiles for frame 1
- Render tiles for frame 2
- Copy tiles for frame 0
- Copy tiles for frame 1

GPU 1
- Render tiles for frame 0
Data Transfers

GPU 0
- Render tiles for frame 0
- Render tiles for frame 1
- Render tiles for frame 2

Copy tiles for frame 0
- Copy tiles for frame 0
- Copy tiles for frame 1

GPU 1
- Render tiles for frame 0
Data Transfers

GPU 0

Render tiles for frame 0
Render tiles for frame 1
Render tiles for frame 2

Copy tiles for frame 0
Copy tiles for frame 1

GPU 1

Render tiles for frame 0
Render tiles for frame 1
Composite frame 0
Render tiles for frame 2
Composite frame 1
Summary

Ray/triangle intersection
Dynamic scenes
MPSSVGFDenoiser
Use cases
Multi-GPU
More Information

developer.apple.com/wwdc19/613

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal for Machine Learning and Ray Tracing Lab</td>
<td>Friday, 12:00</td>
</tr>
<tr>
<td>Metal for Machine Learning Session</td>
<td>Friday, 3:20</td>
</tr>
</tbody>
</table>