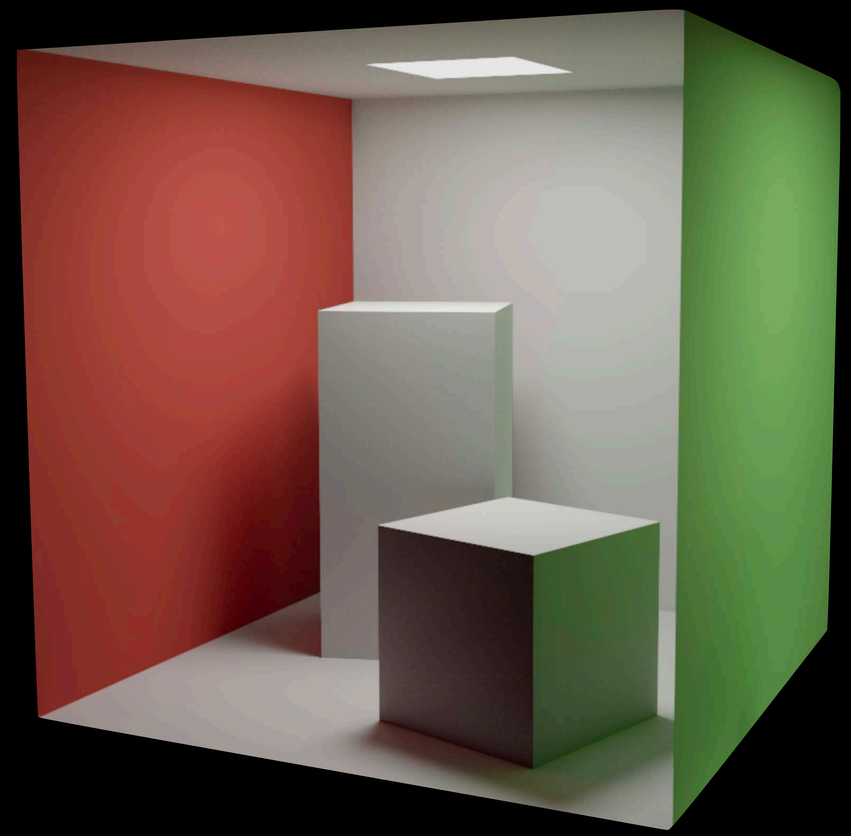
#WWDC19

Ray Tracing with Metal

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

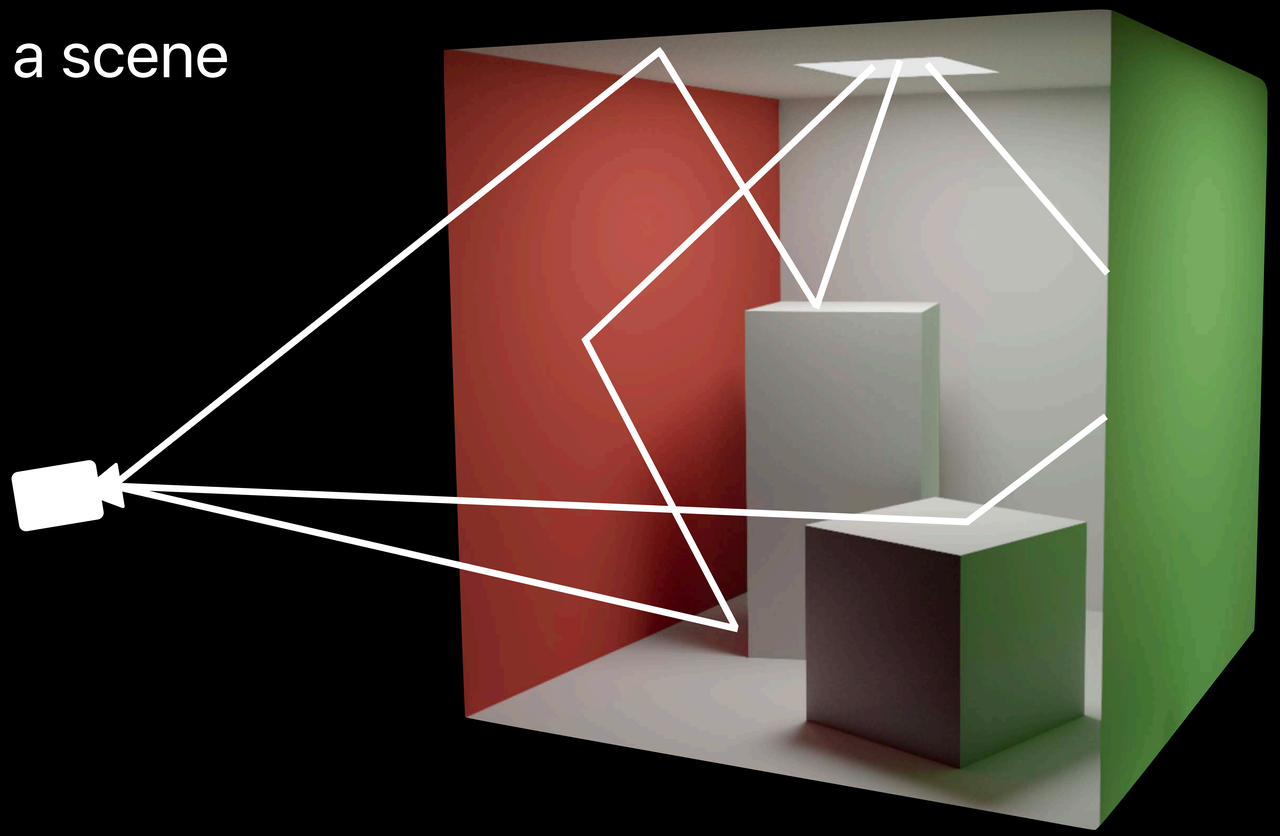
Ray Tracing

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



Ray Tracing

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

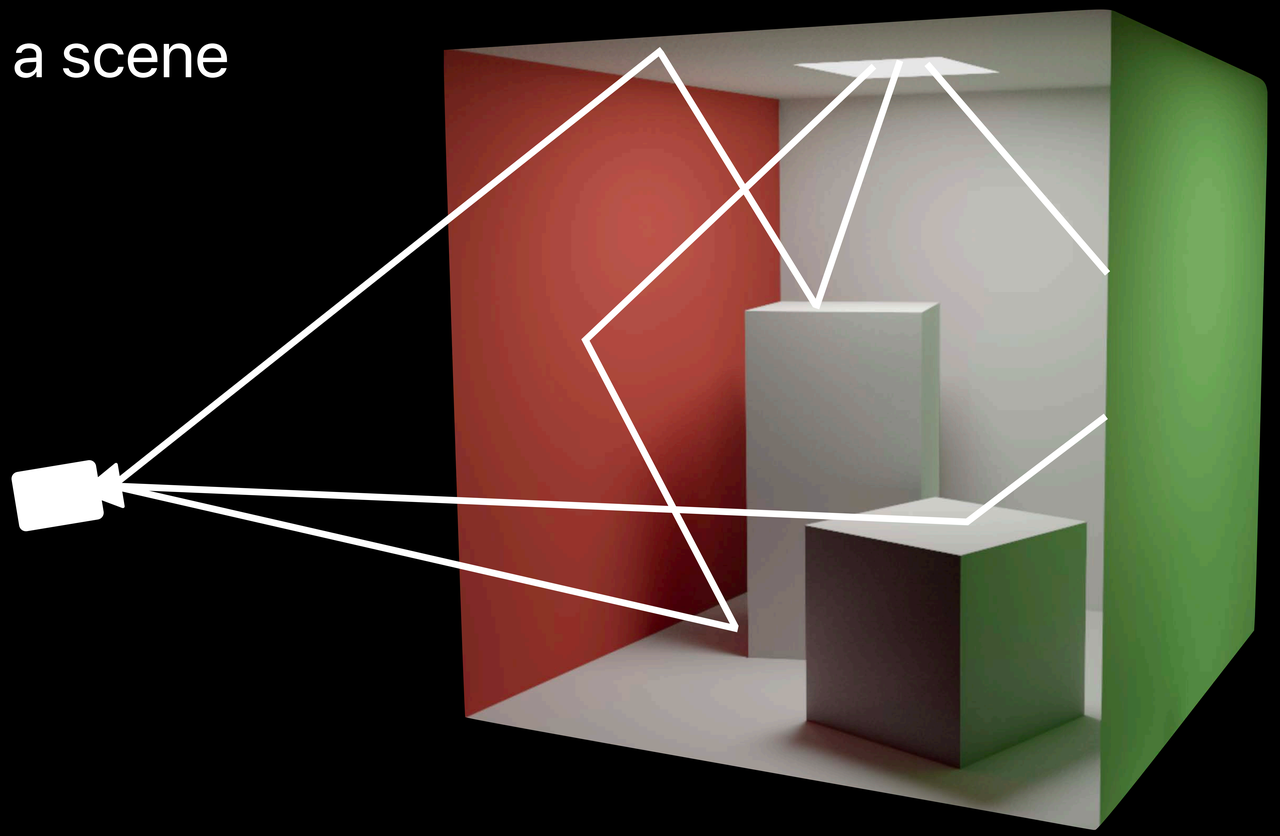


Ray Tracing

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

Ray Tracing in Real-Time

Dynamic scenes

Ray Tracing in Real-Time

Dynamic scenes

Performance

Ray Tracing in Real-Time

Dynamic scenes

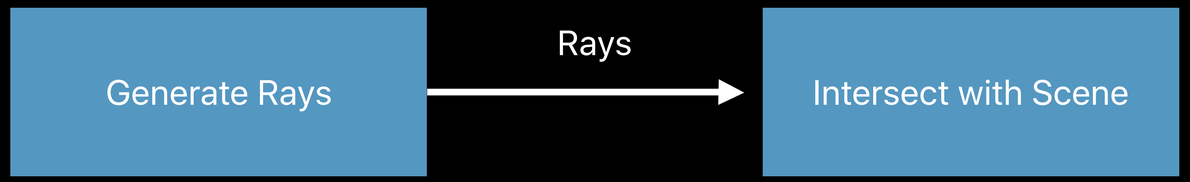
Performance

Denoising

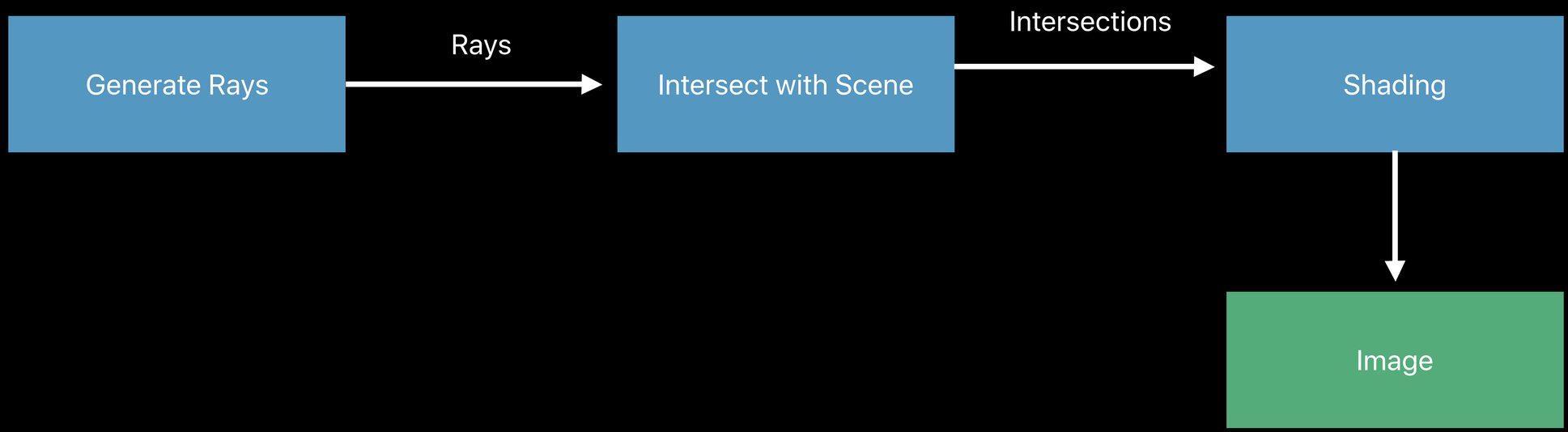
Ray Tracing with Metal

Ray Tracing with Metal

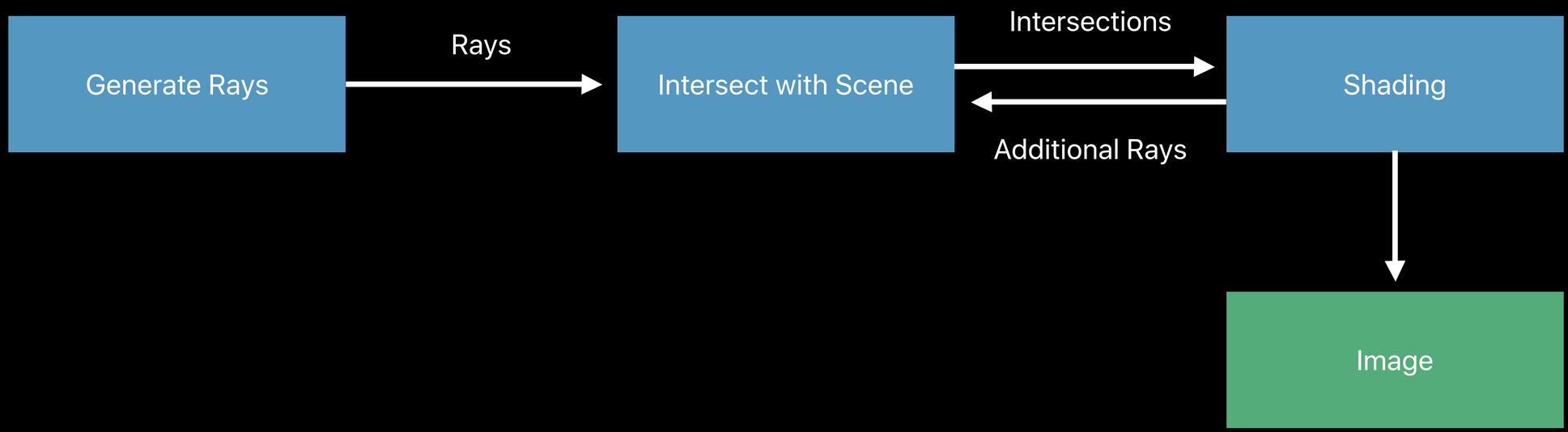
Ray Tracing with Metal



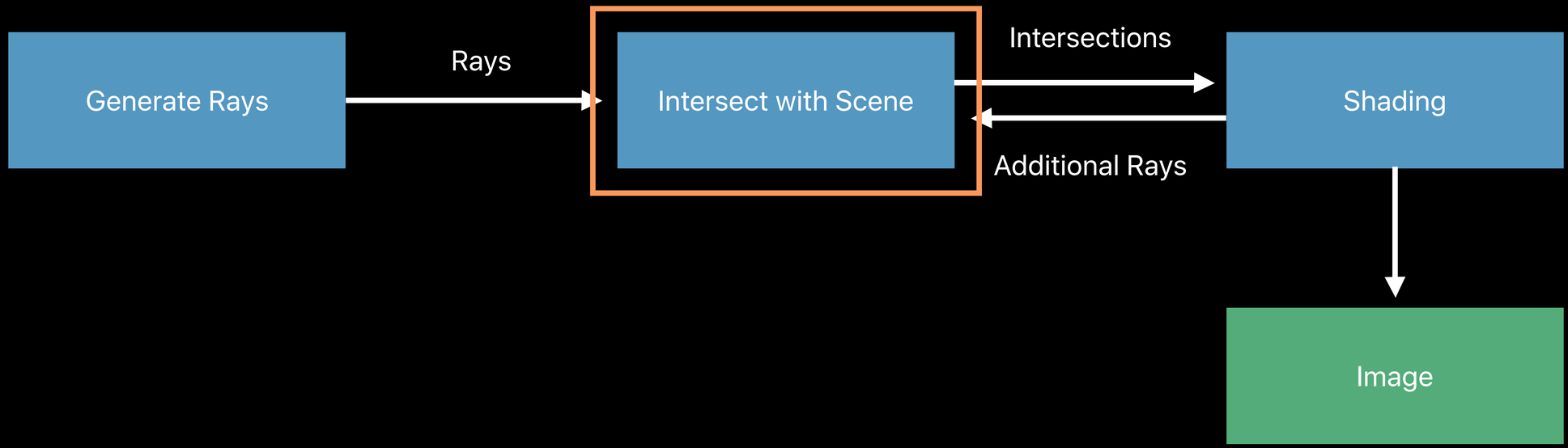
Ray Tracing with Metal



Ray Tracing with Metal



Ray Tracing with Metal



MPSRayIntersector

Ray intersector accelerates ray/triangle intersection tests on the GPU

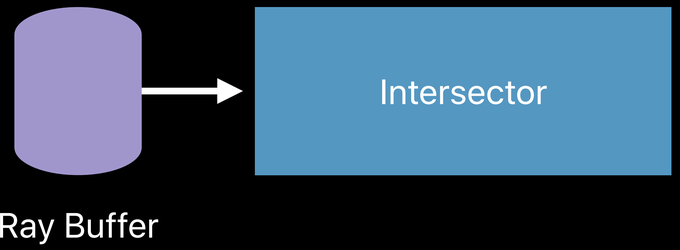
MPSRayIntersector

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

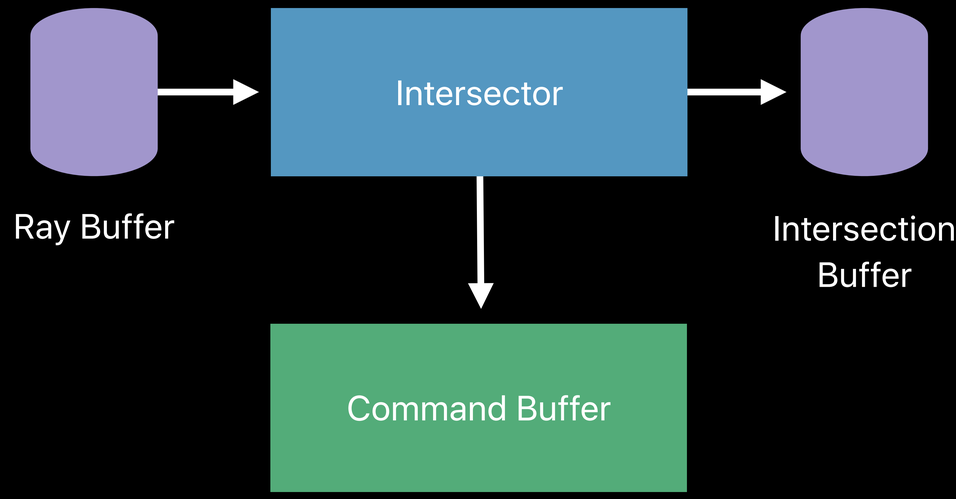
MPSRayIntersector

Ray intersector accelerates ray/triangle intersection tests on the GPU

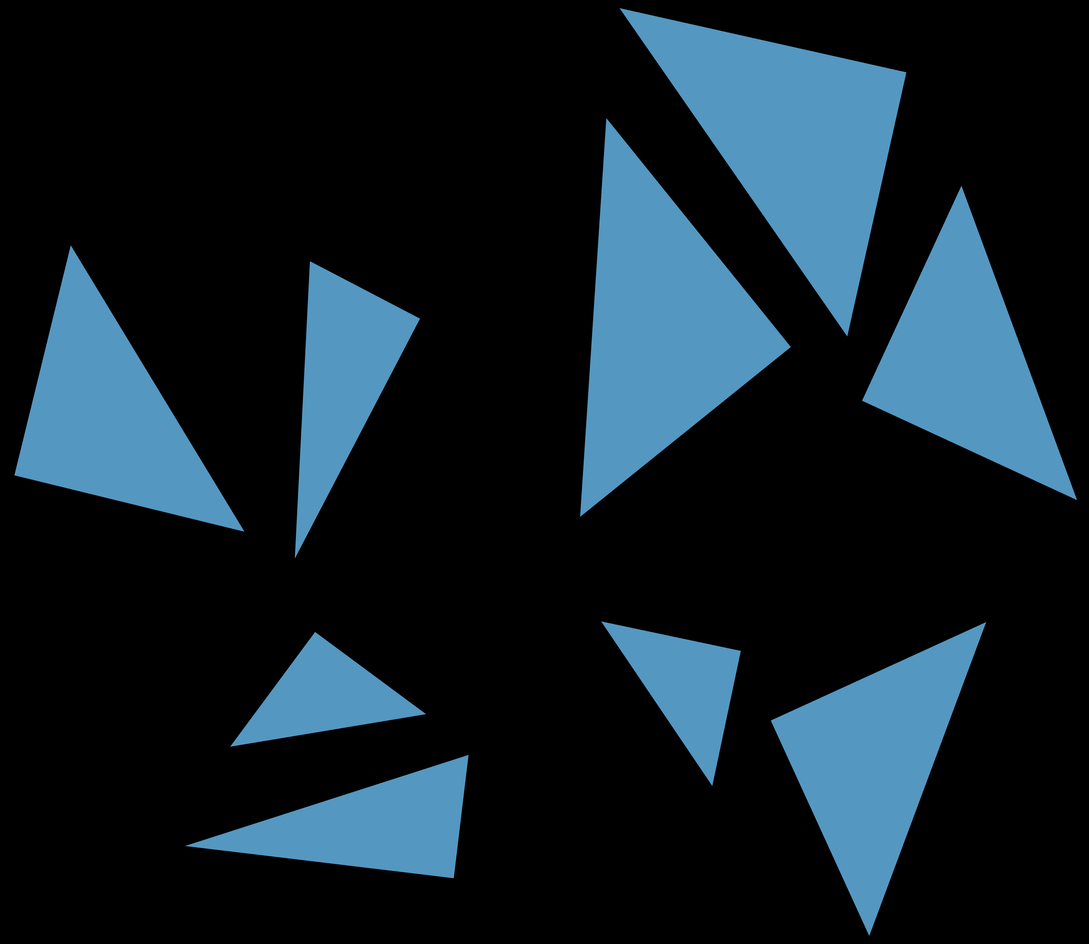
Accepts batches of rays in a Metal buffer

Returns one intersection per ray

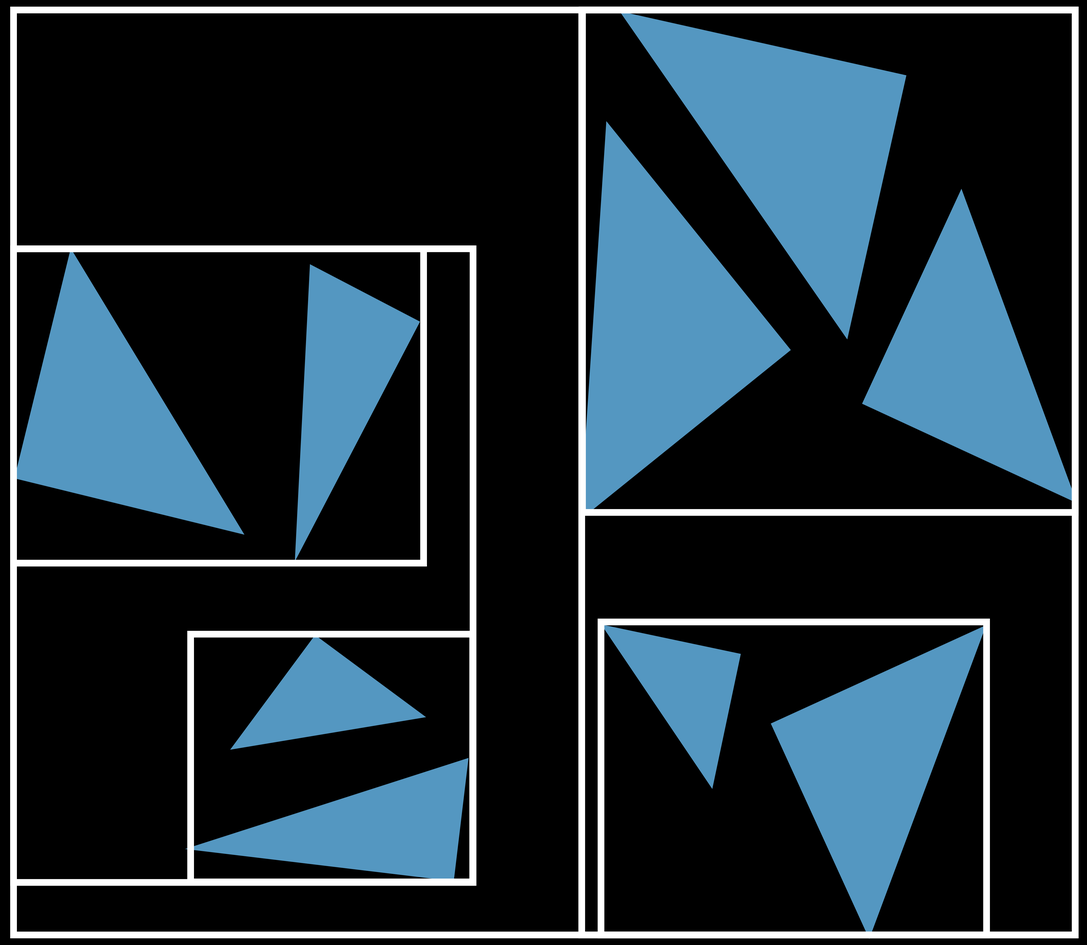
Encodes into a Metal command buffer



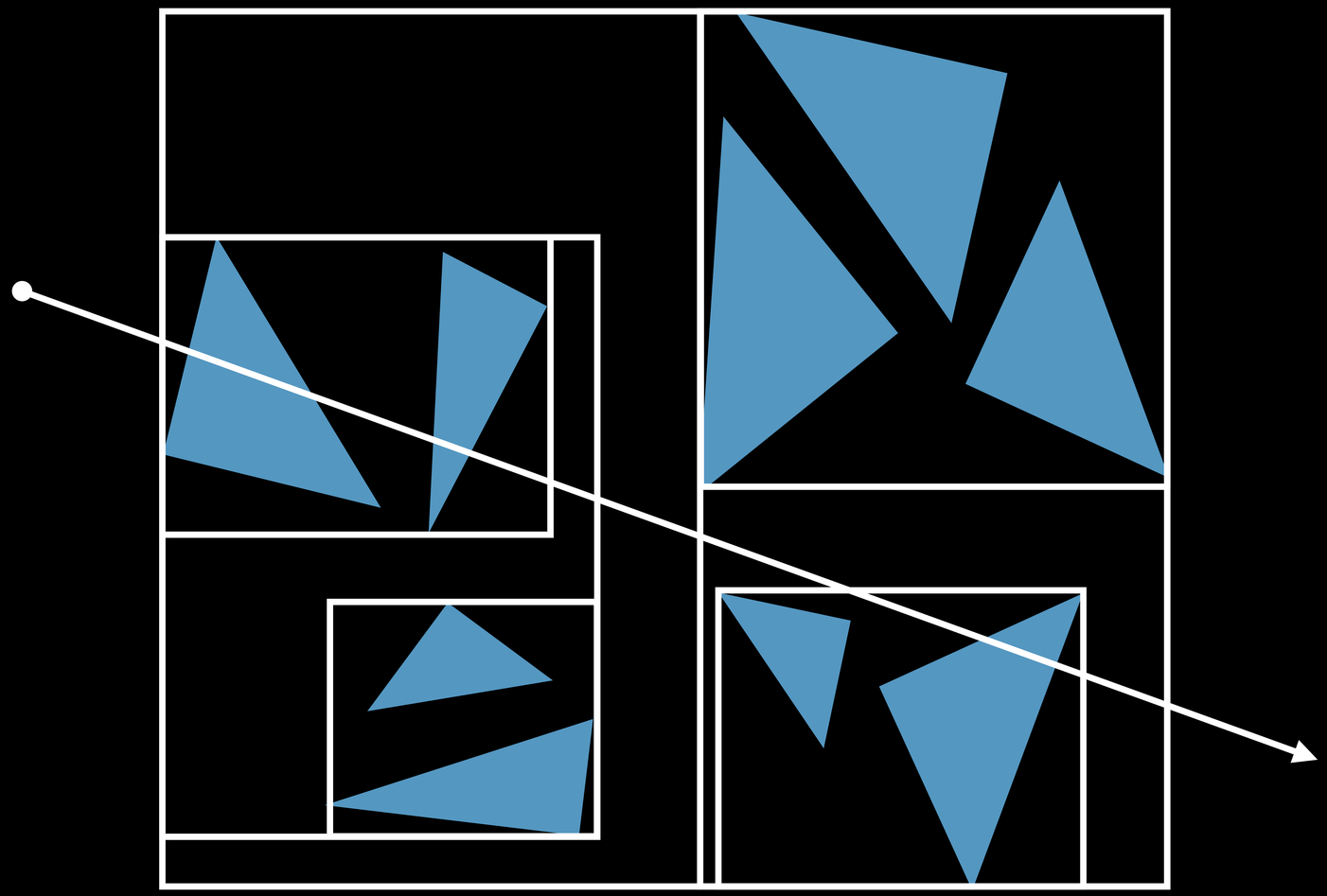
MPSRayIntersector



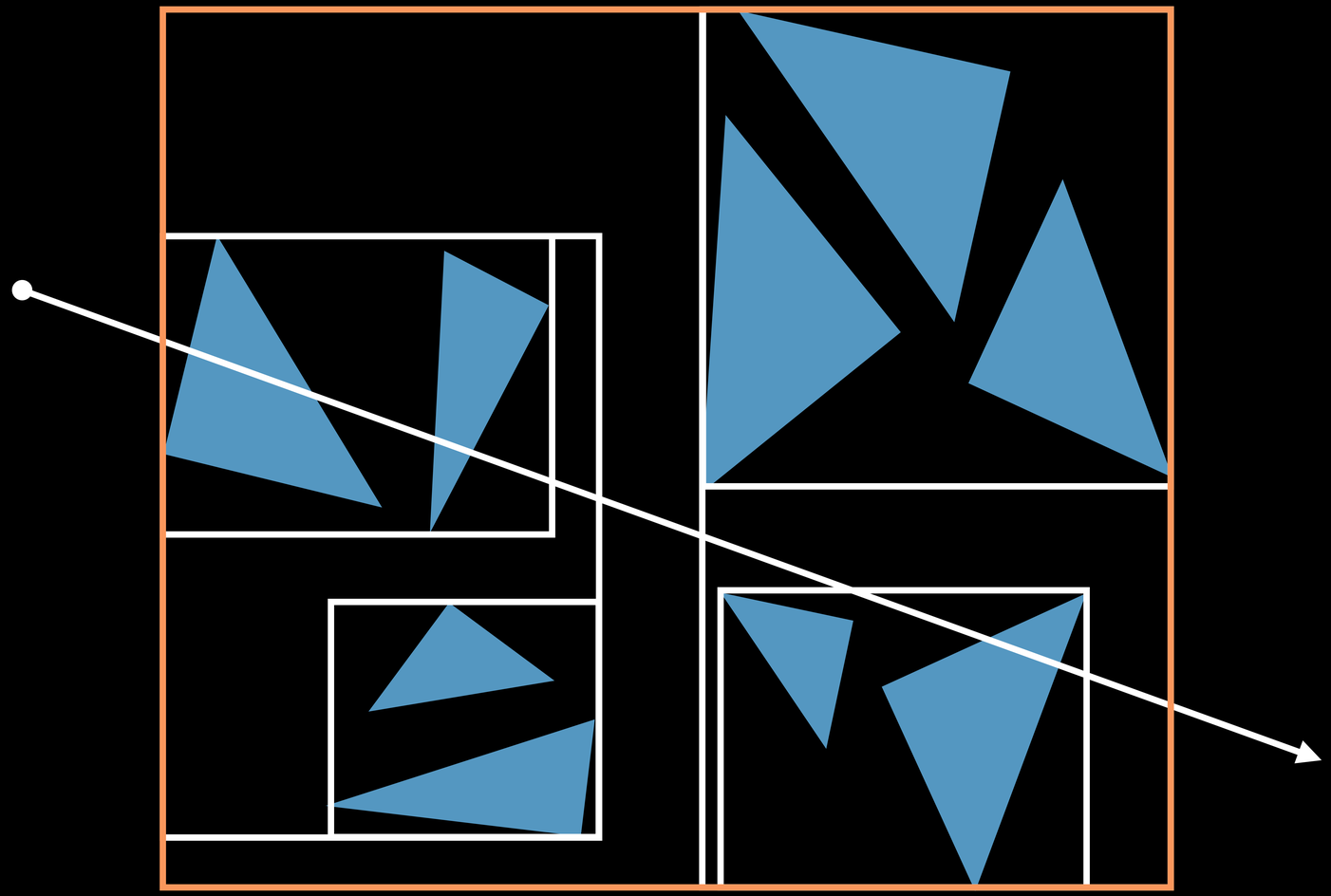
MPSRayIntersector



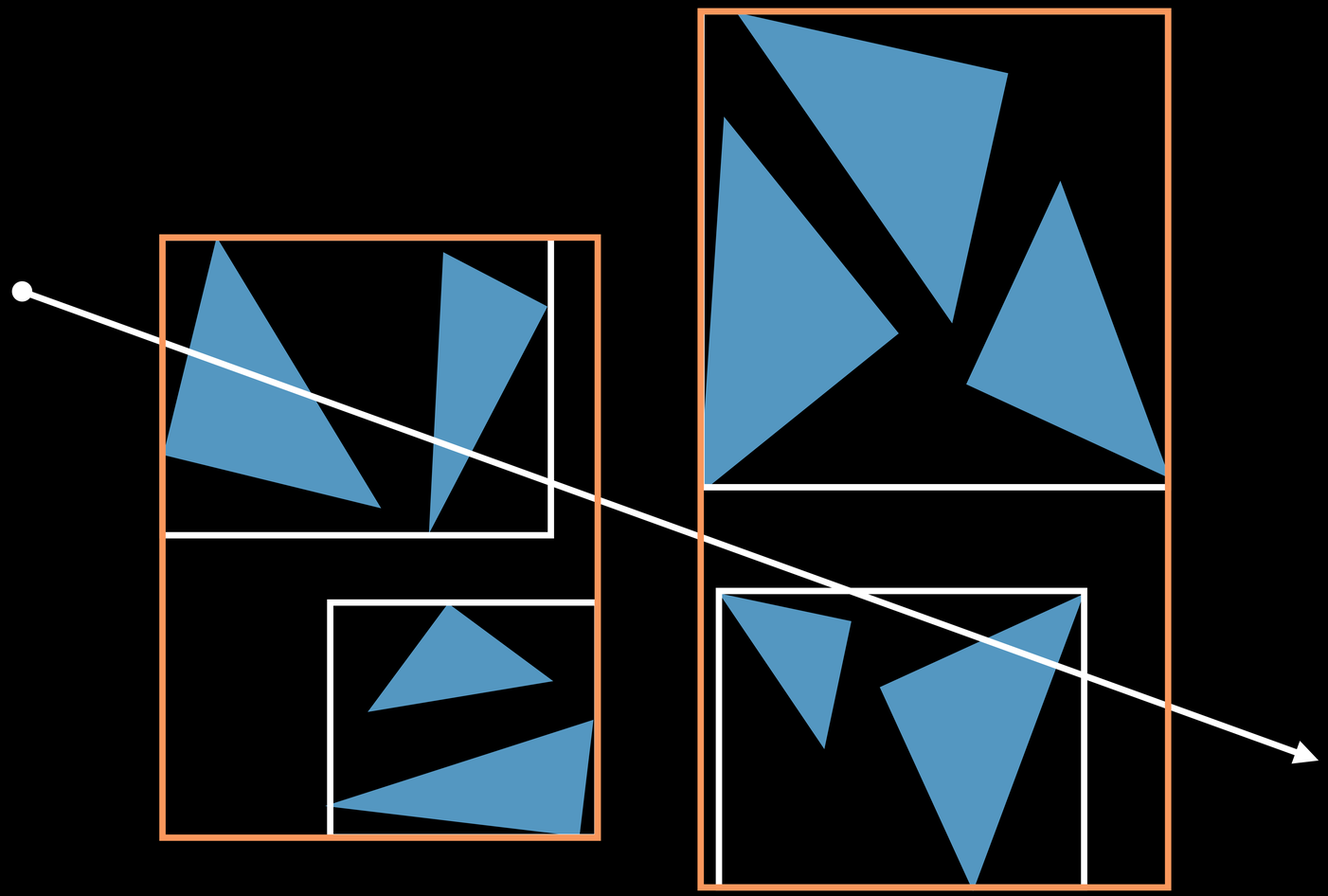
MPSRayIntersector



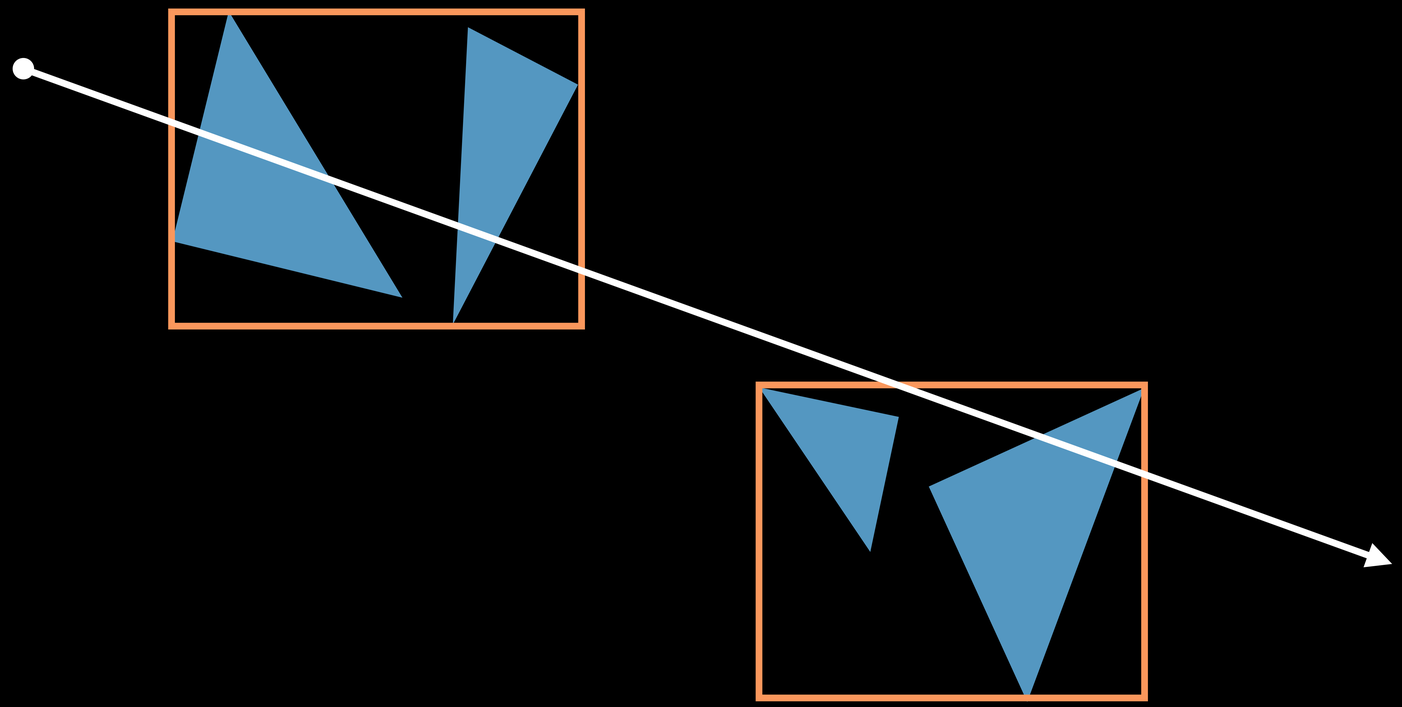
MPSRayIntersector



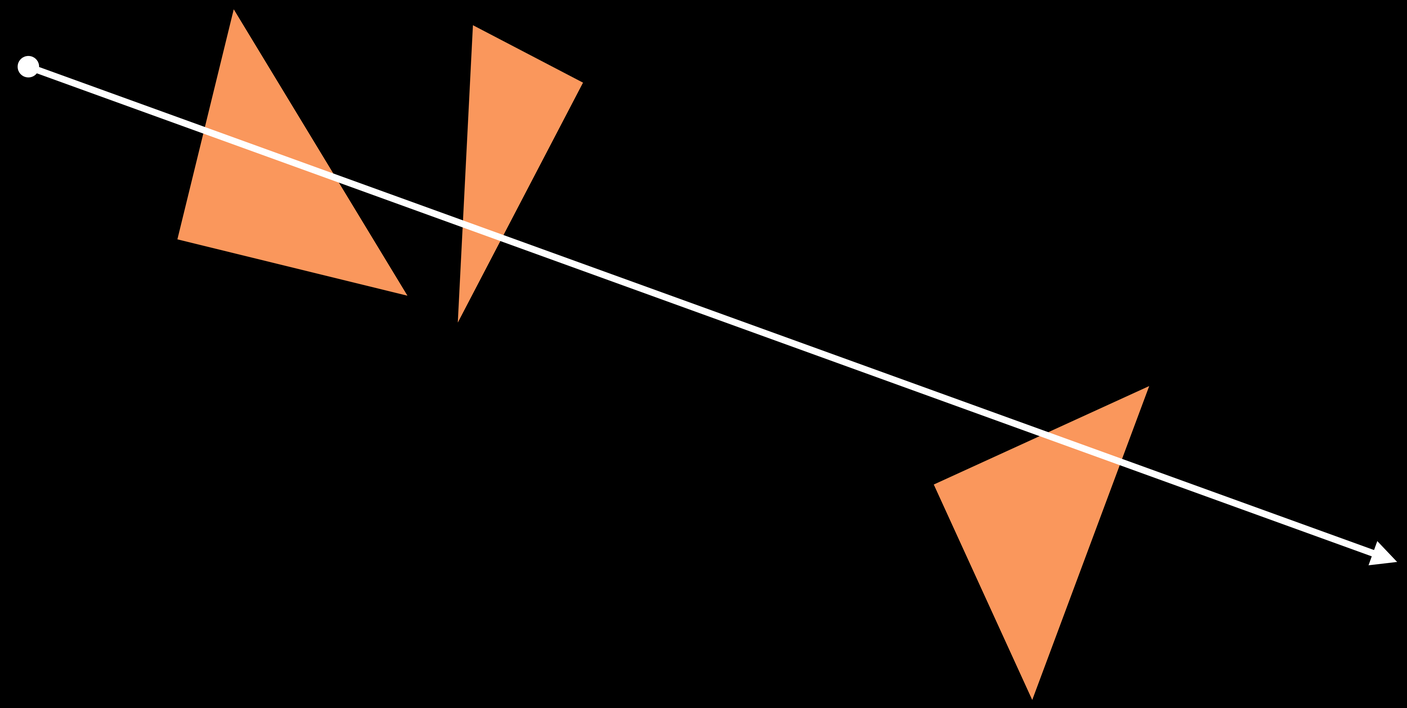
MPSRayIntersector



MPSRayIntersector

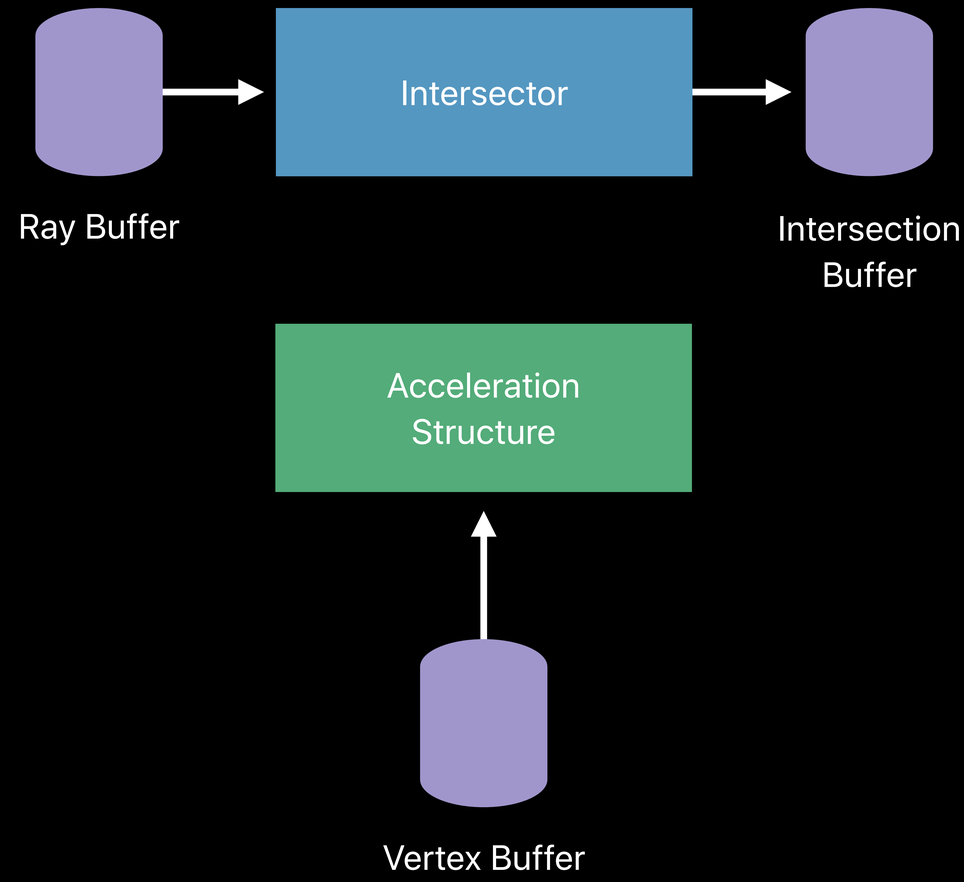


MPSRayIntersector



MPSRayIntersector

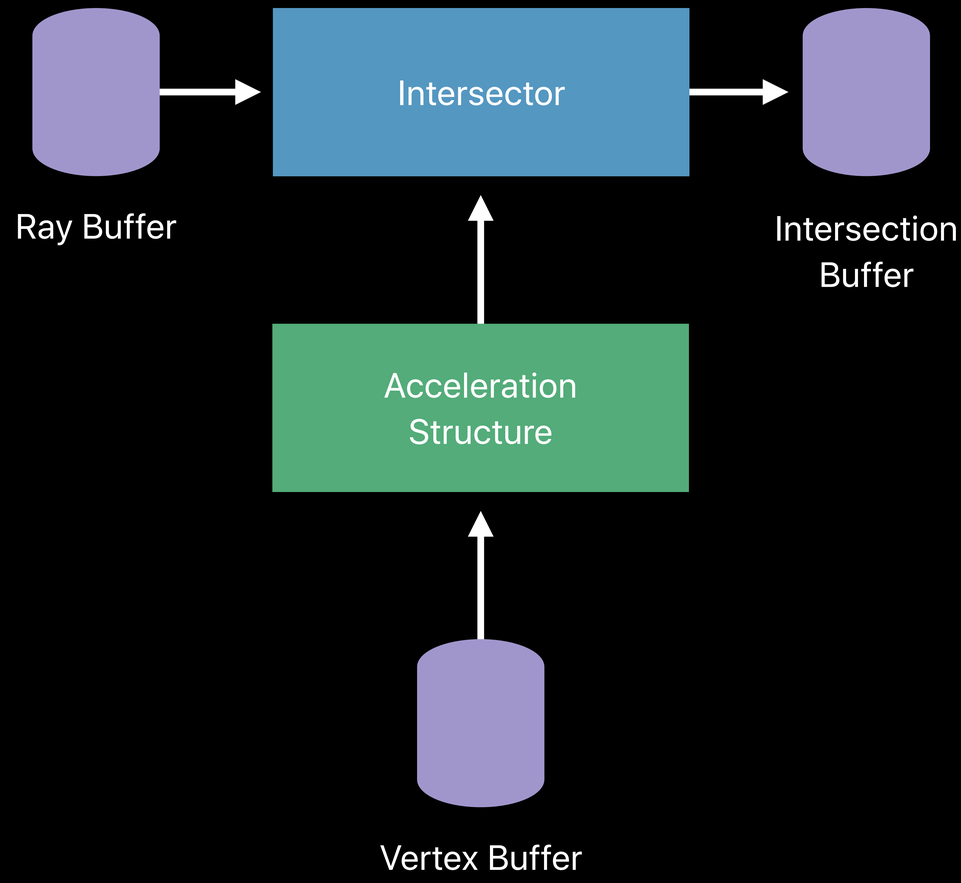
Build an acceleration structure over triangles in a vertex buffer



MPSRayIntersector

Build an acceleration structure over triangles in a vertex buffer

Pass acceleration structure to intersector

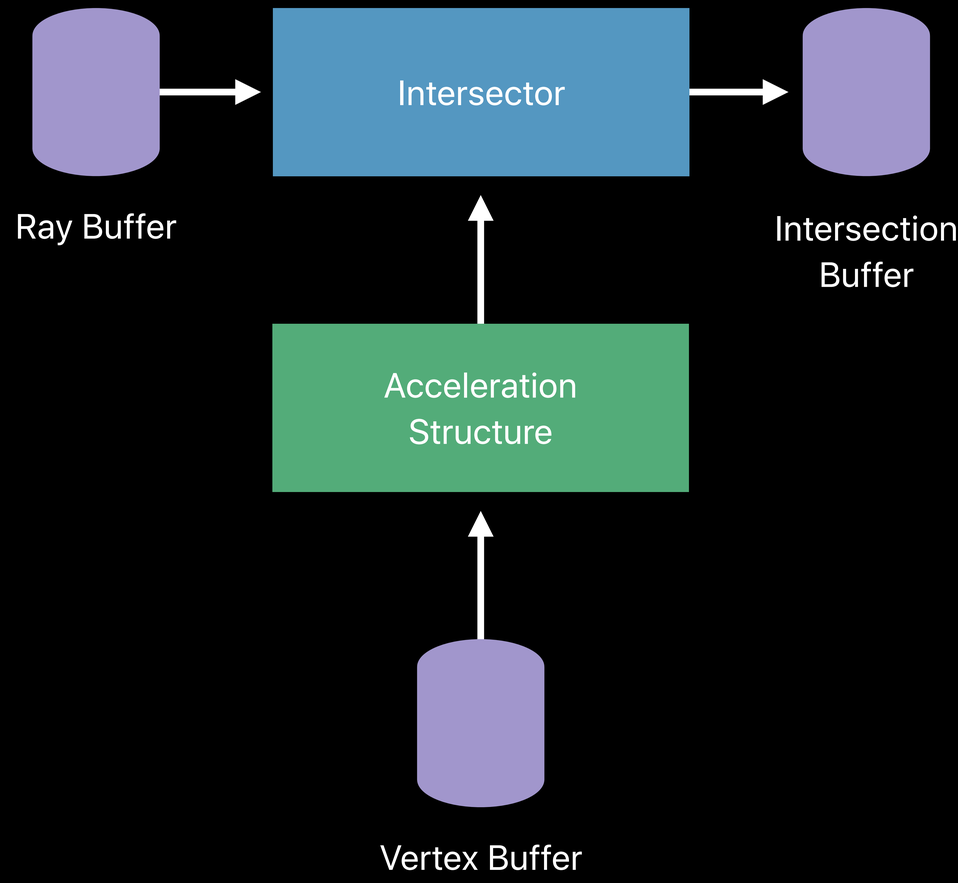


MPSRayIntersector

Build an acceleration structure over triangles in a vertex buffer

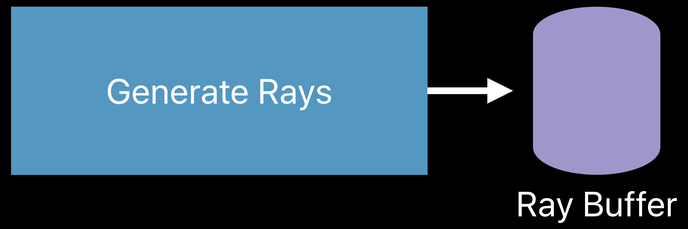
Pass acceleration structure to intersector

Now builds on the GPU

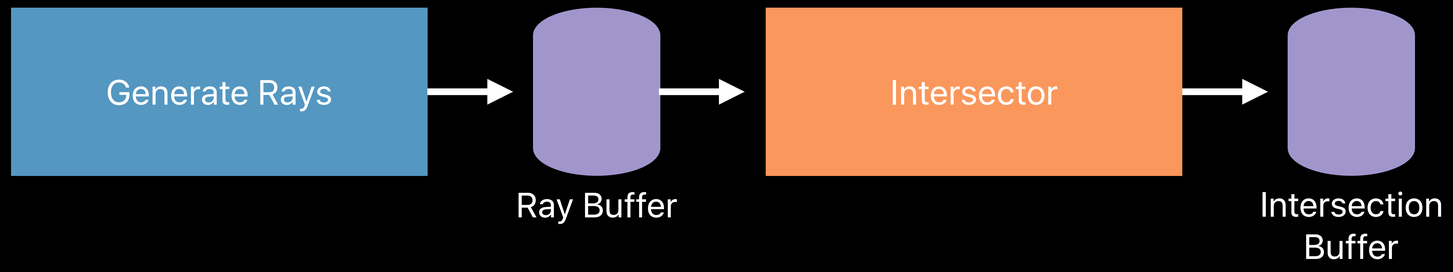


Ray Tracing with Metal

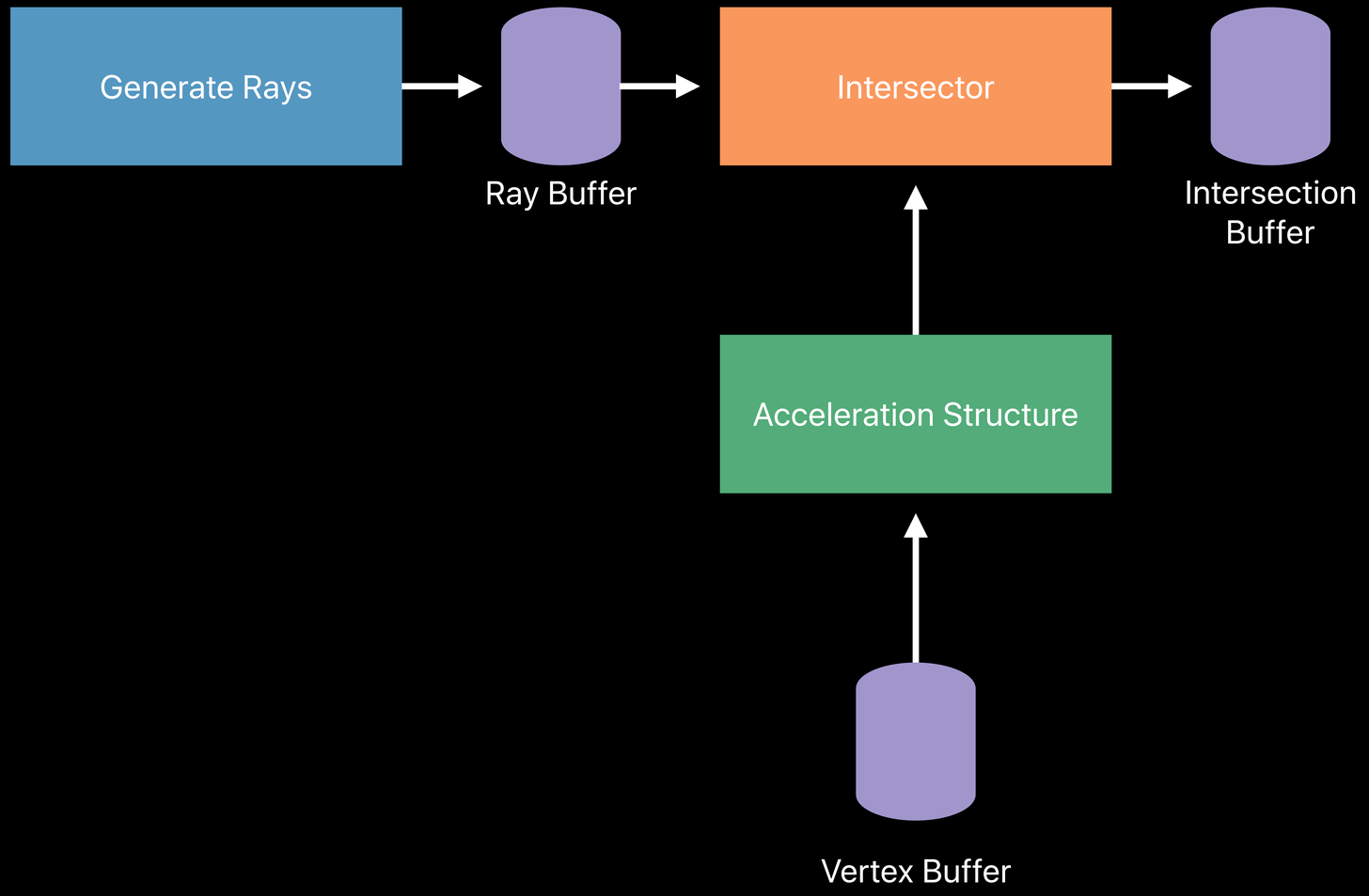
Ray Tracing with Metal



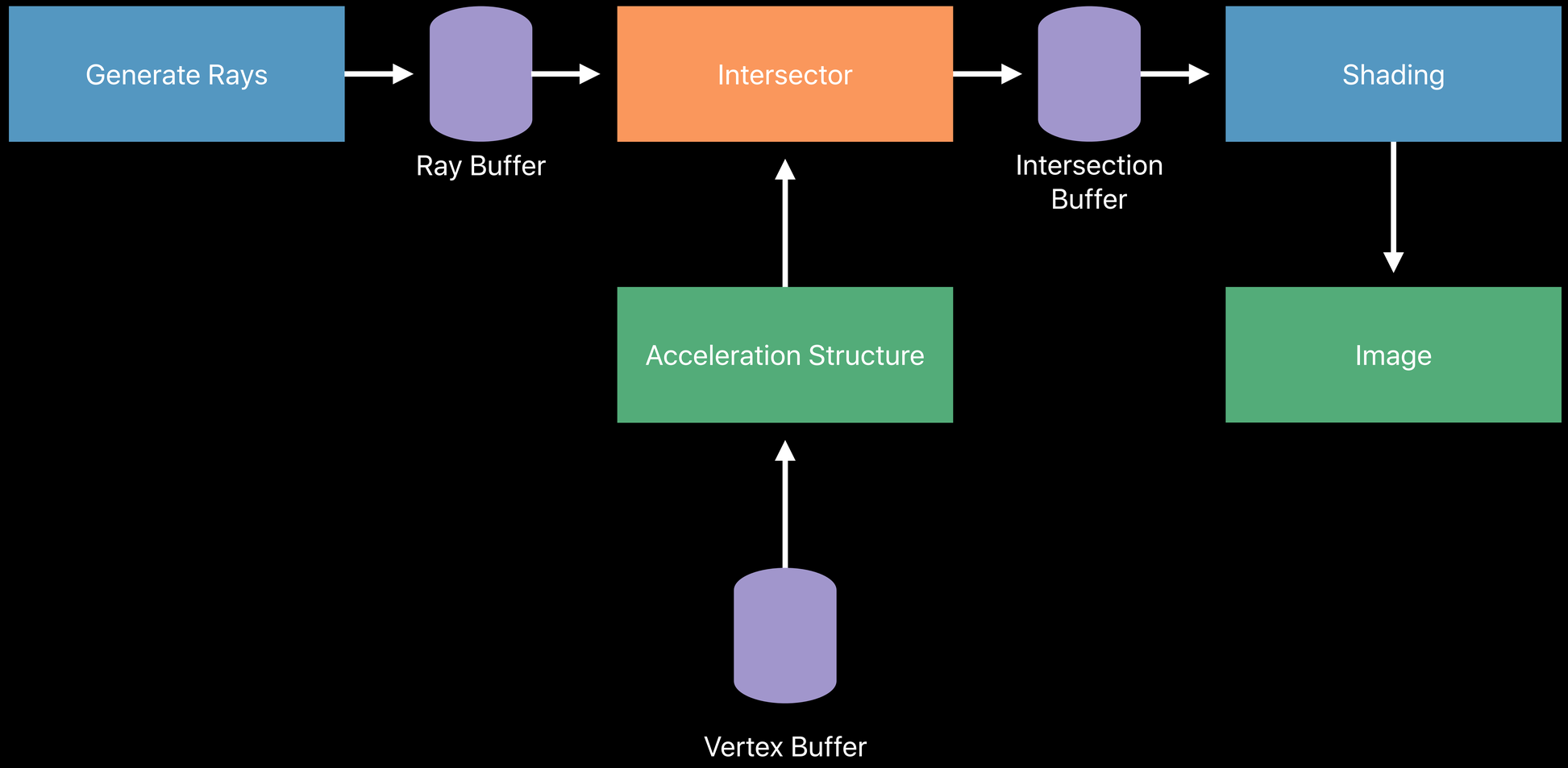
Ray Tracing with Metal



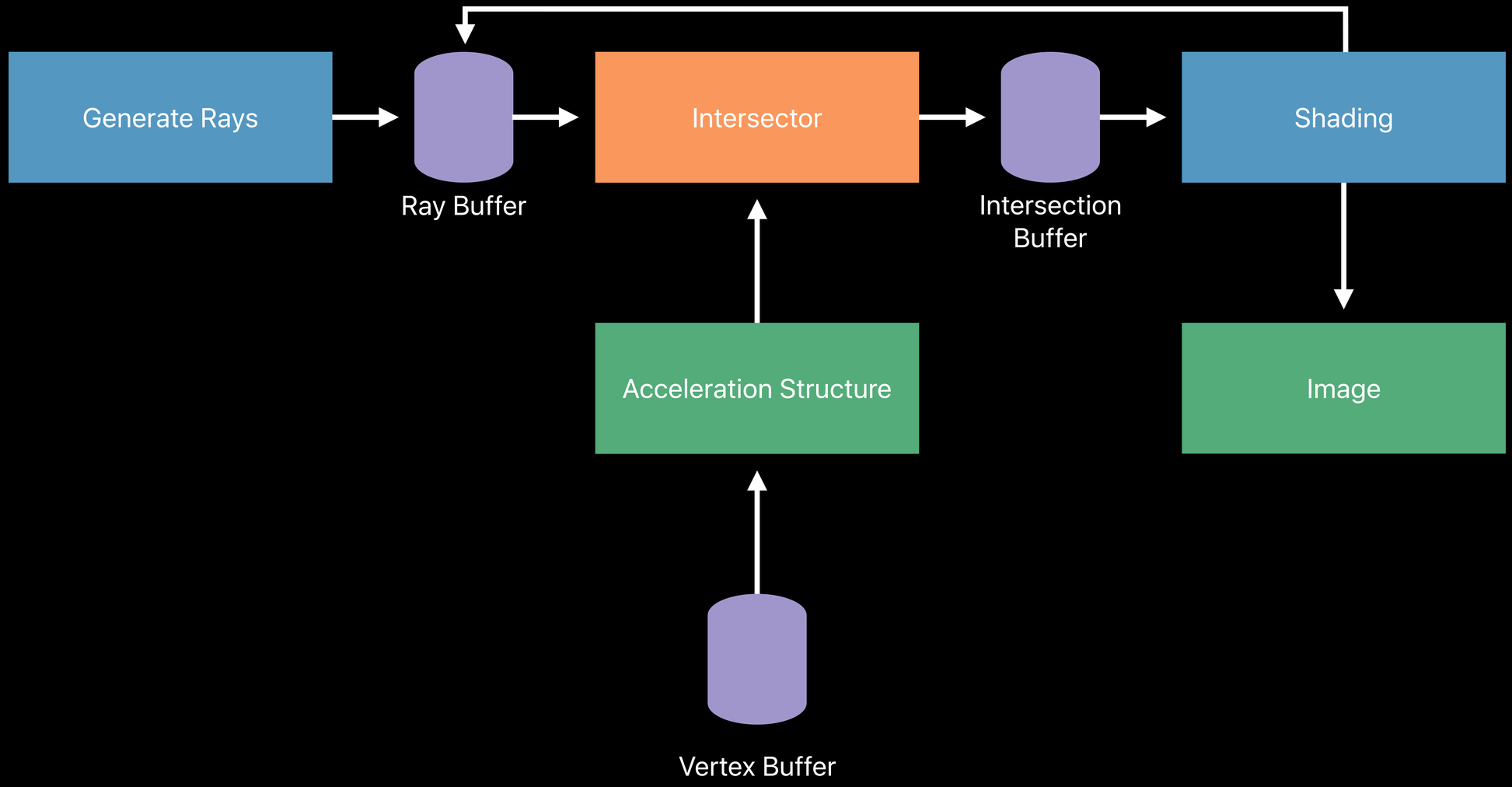
Ray Tracing with Metal



Ray Tracing with Metal



Ray Tracing with Metal



Ray Tracing in AR Quick Look

Ray Tracing in AR Quick Look

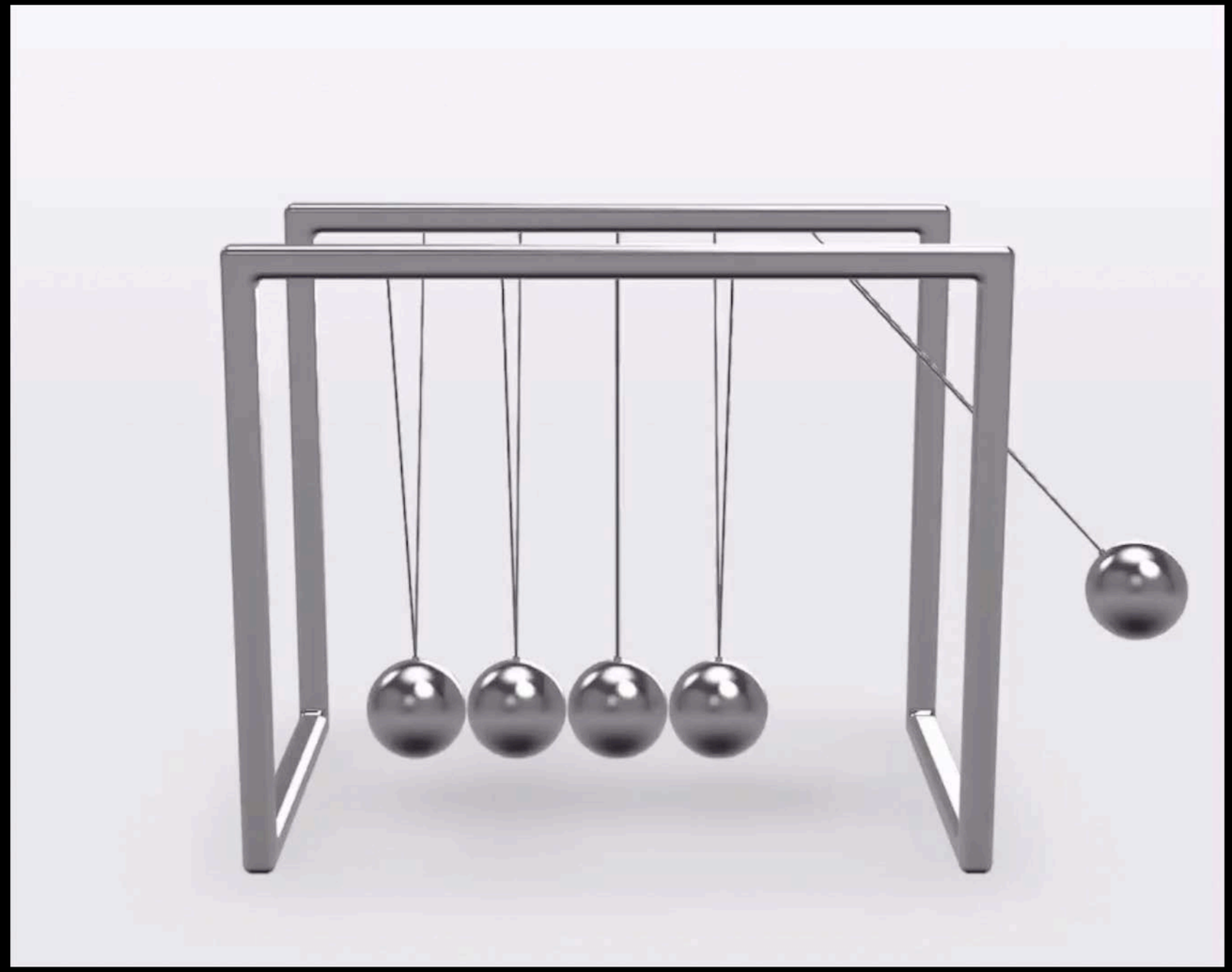
Ray Tracing in AR Quick Look

Ray Tracing in AR Quick Look

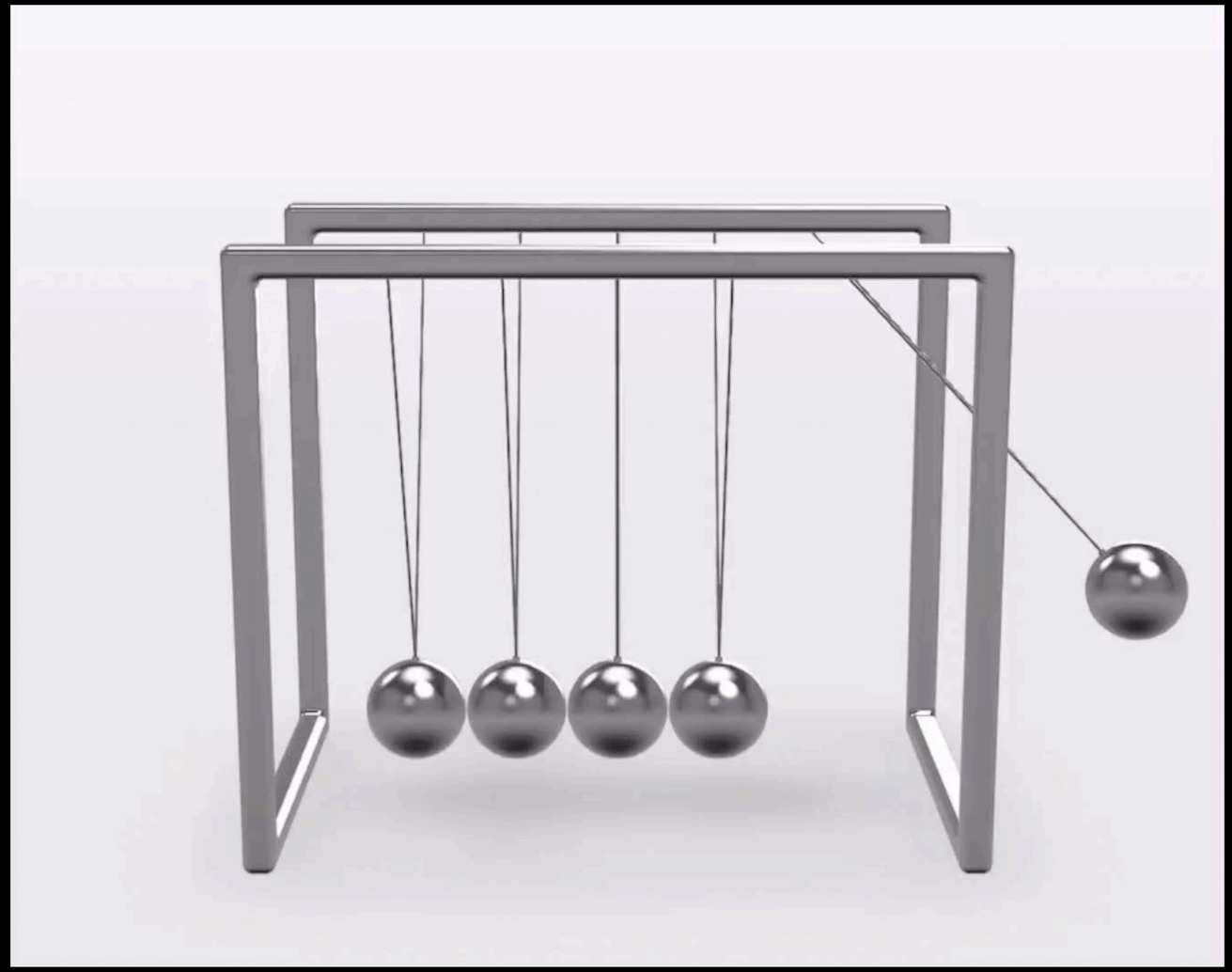
Ray Tracing in AR Quick Look

Ray Tracing in AR Quick Look

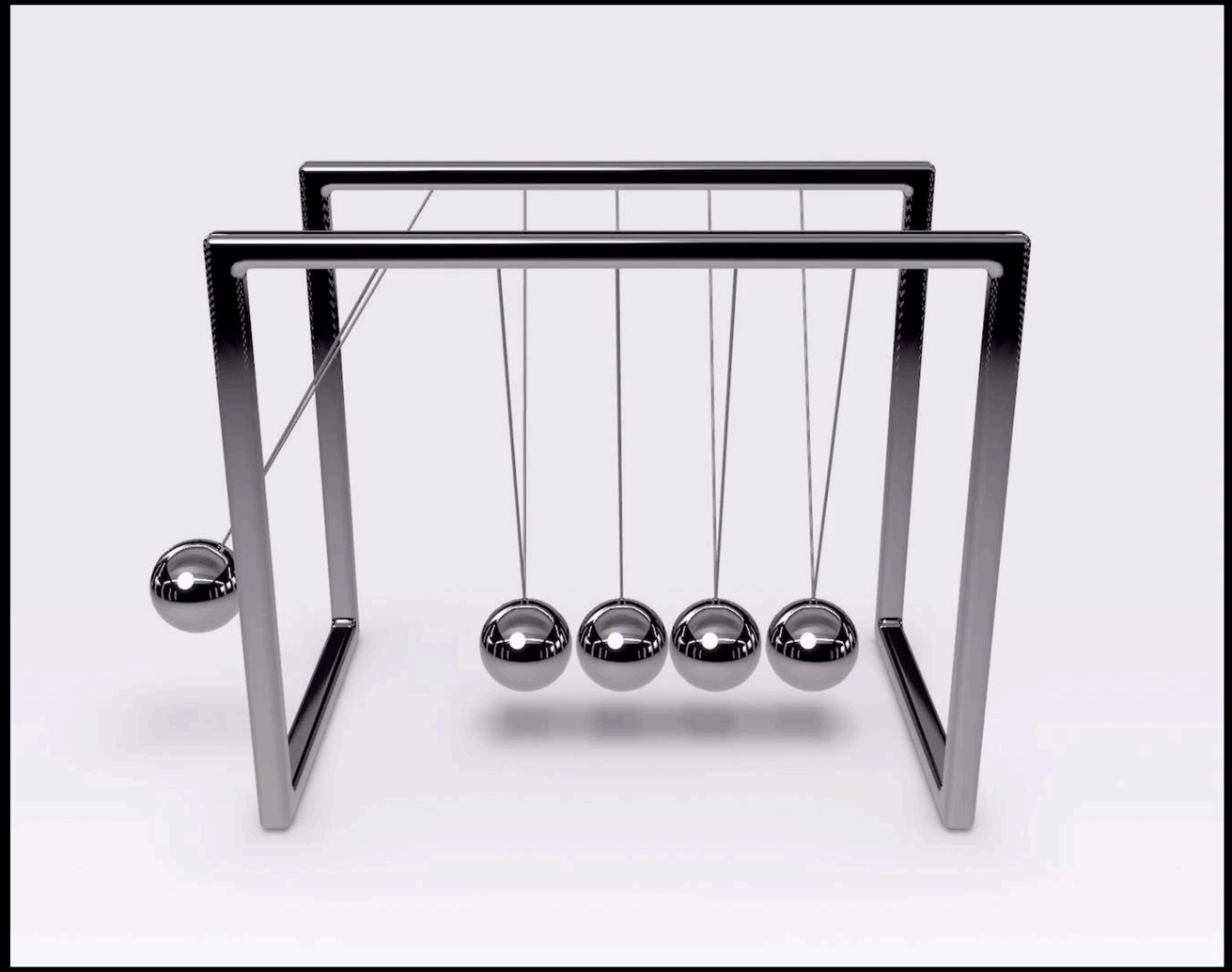
Ray Tracing in AR Quick Look



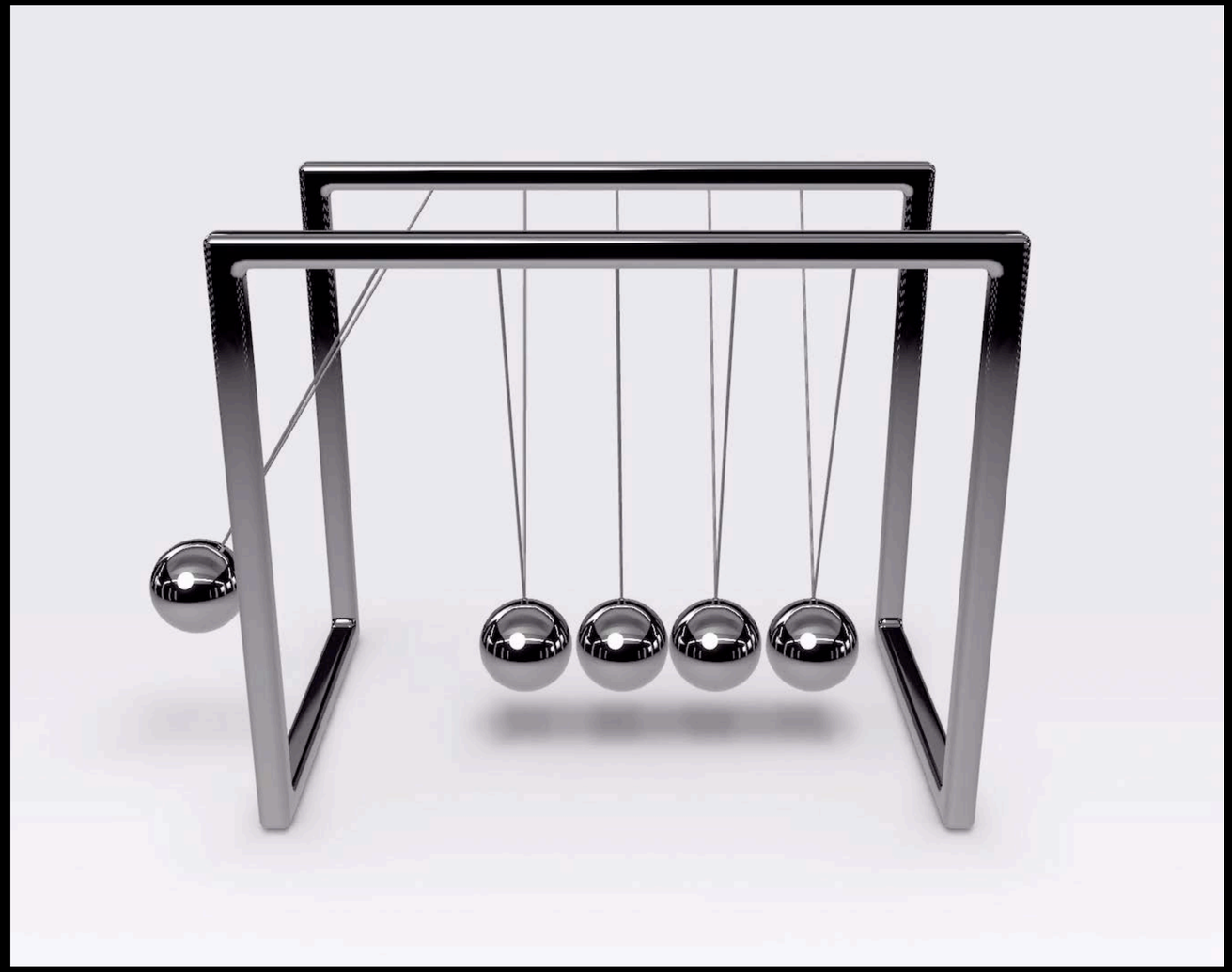
Ray Tracing in AR Quick Look



Ray Tracing in AR Quick Look



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

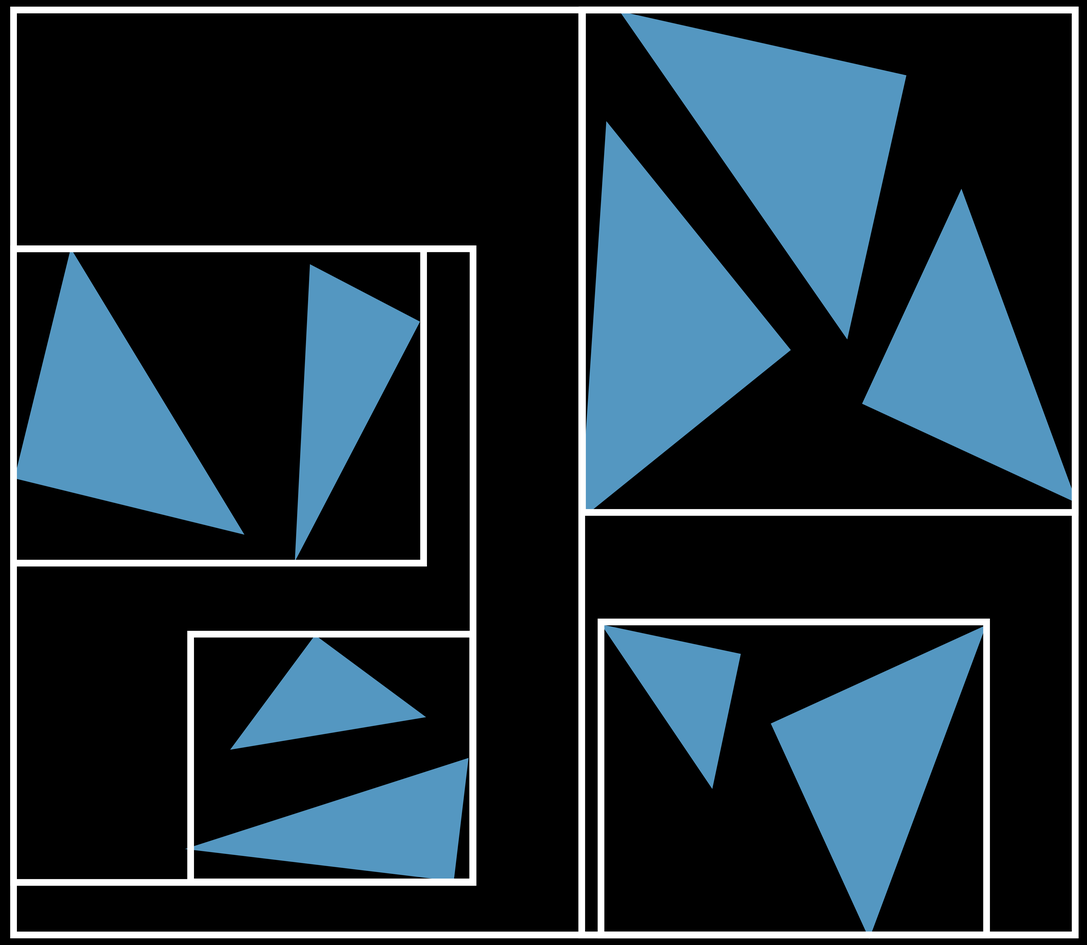
Vertex Animation

Deformation and skinned animation

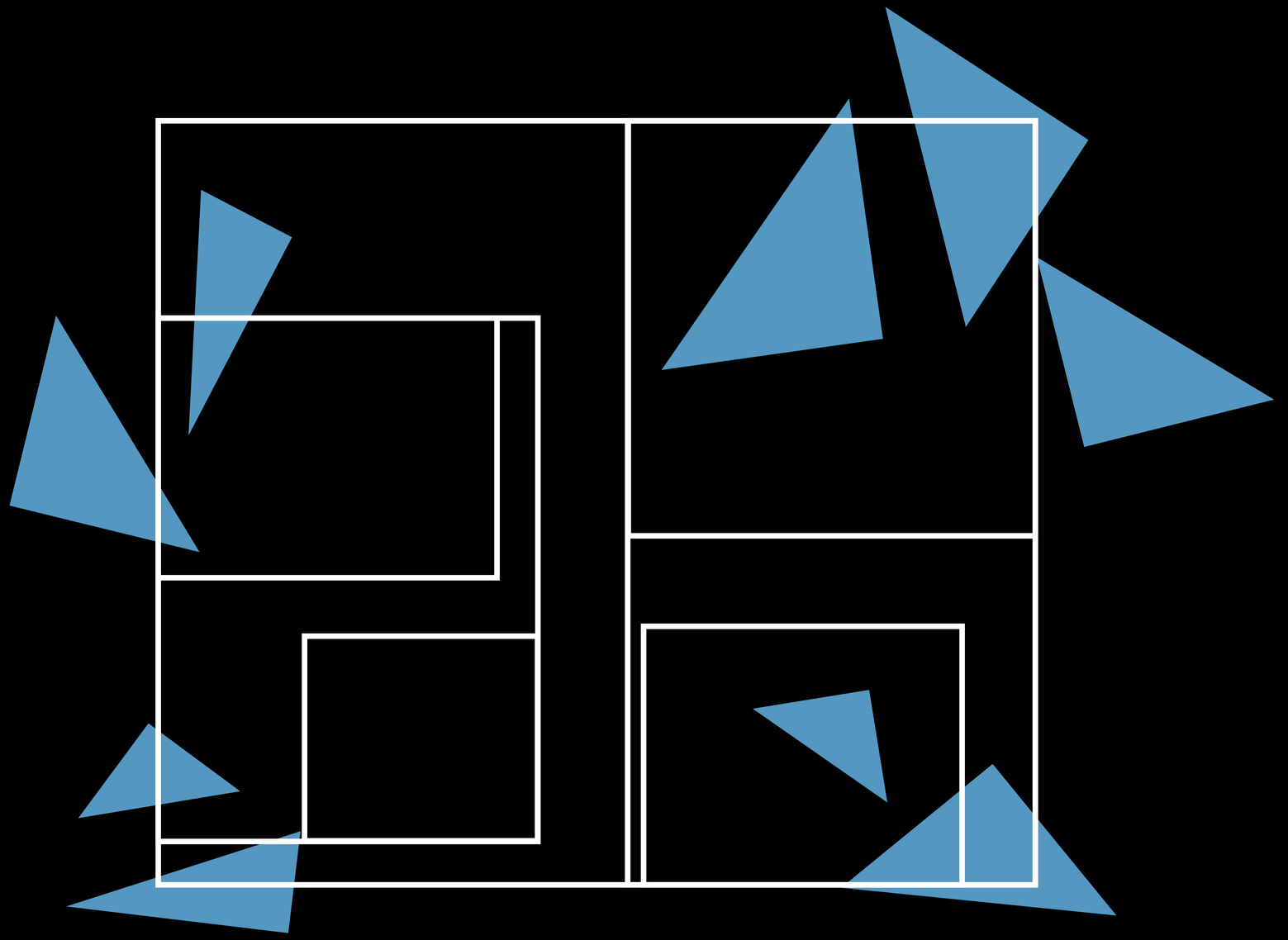
Need to update acceleration structure

Objects tend to retain their shape

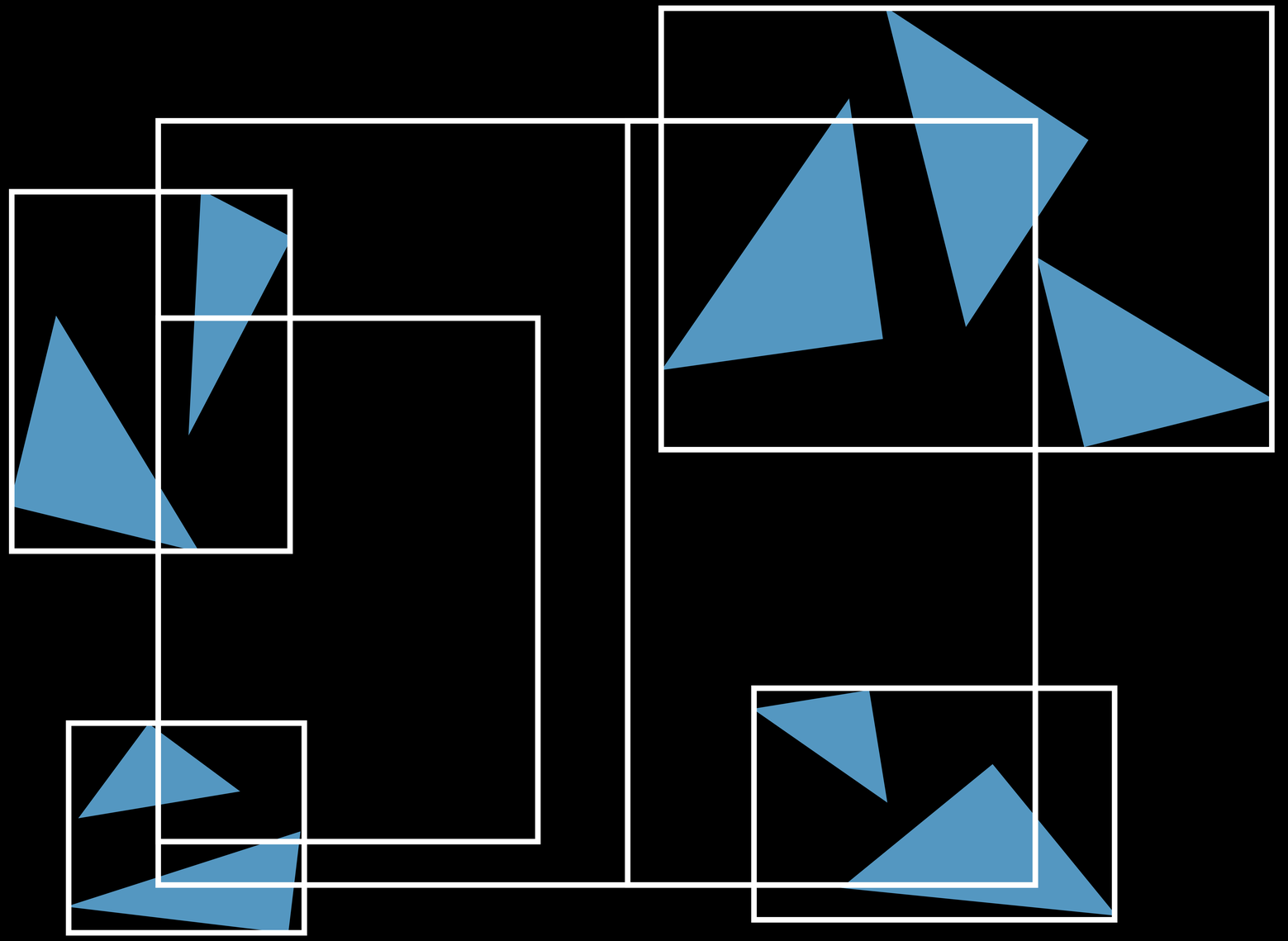
Vertex Animation



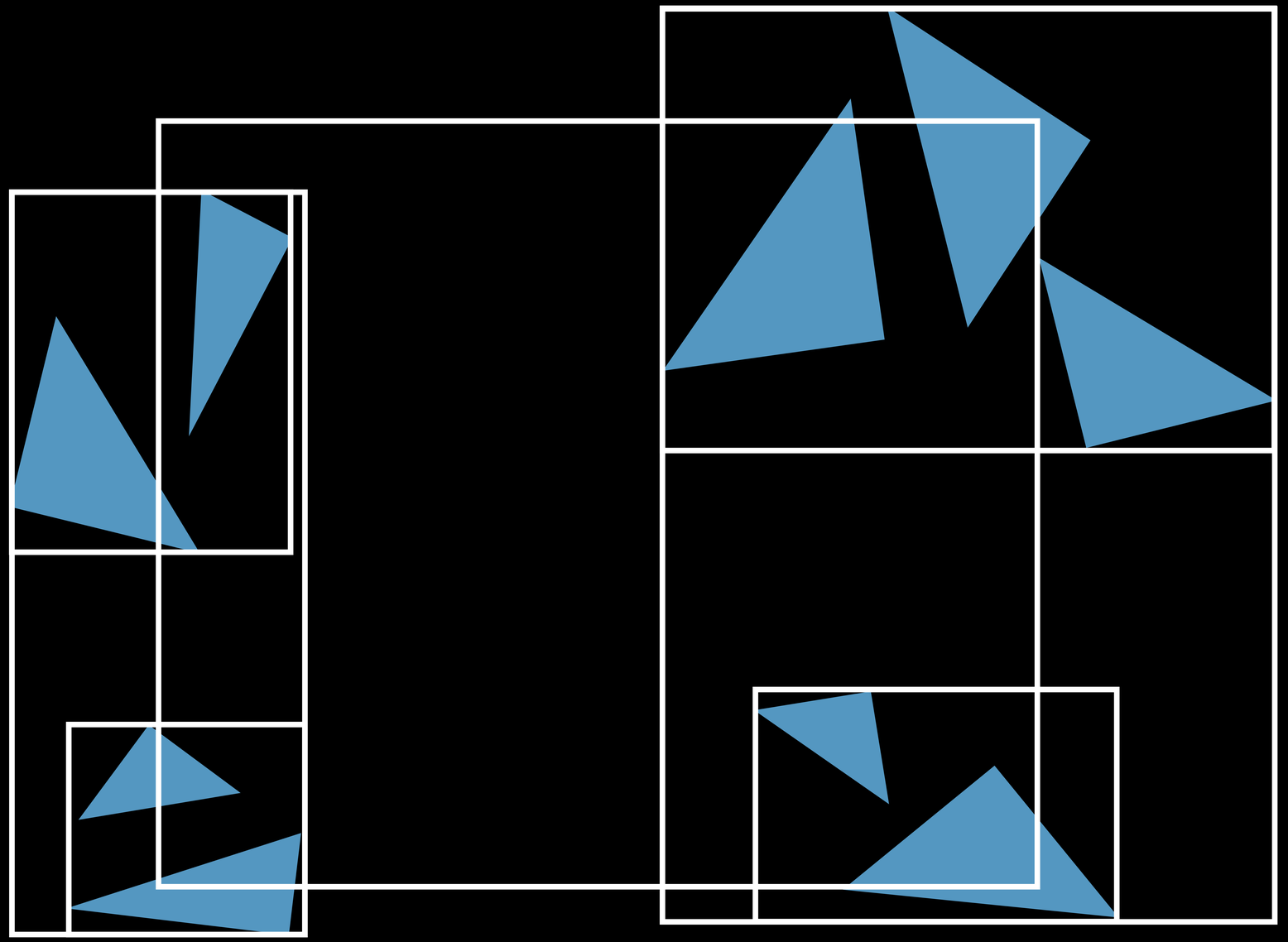
Vertex Animation



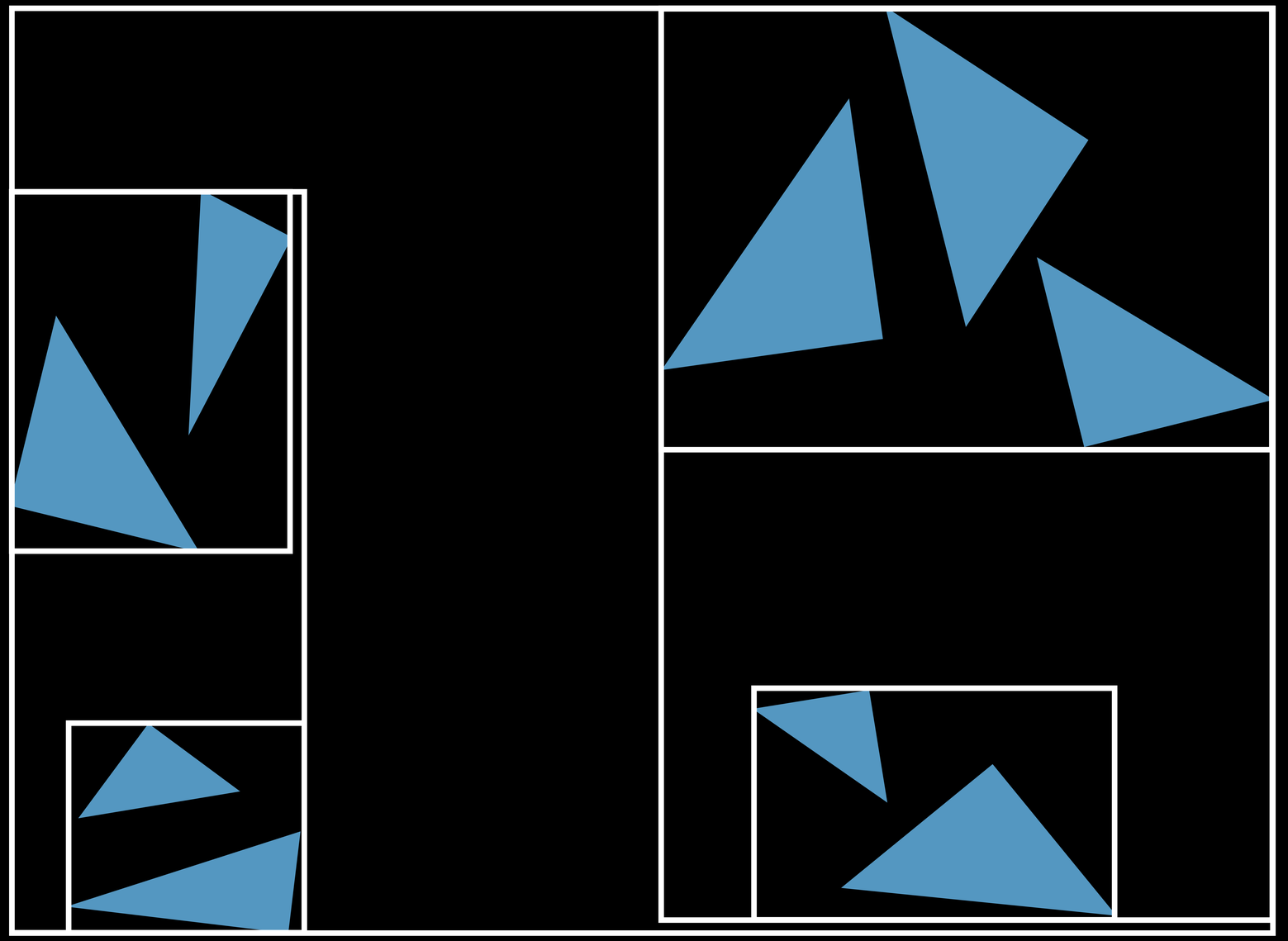
Vertex Animation



Vertex Animation



Vertex Animation



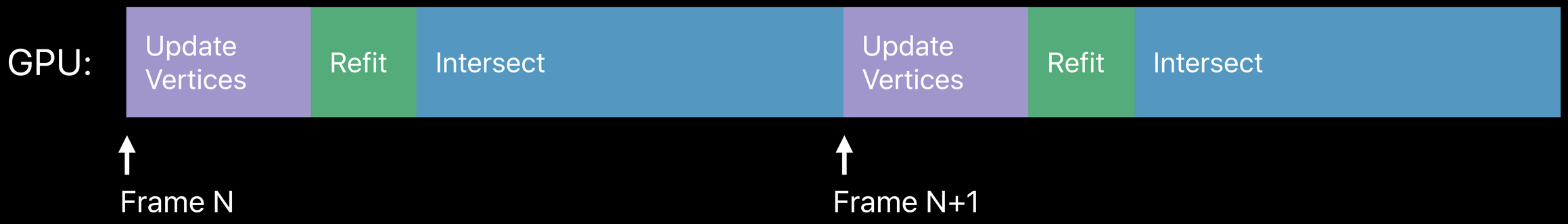
Refitting

Much faster than building from scratch

Refitting

Much faster than building from scratch

Runs on the GPU

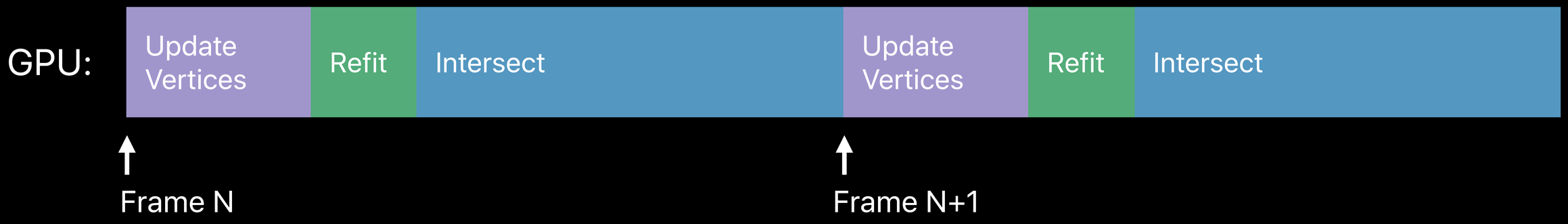


Refitting

Much faster than building from scratch

Runs on the GPU

Can’t add or remove geometry



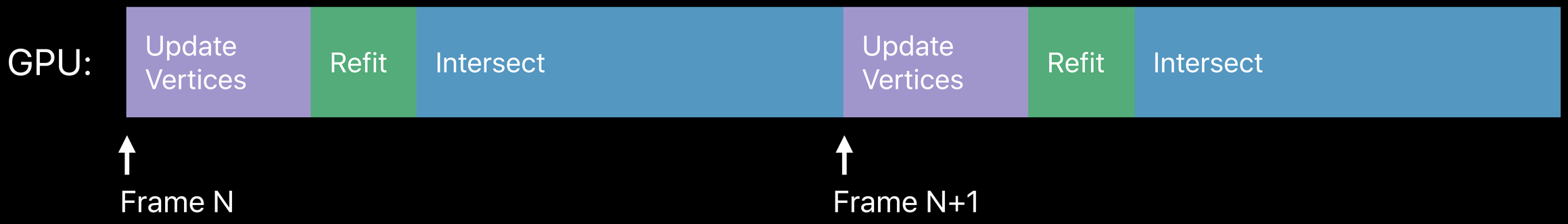
Refitting

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:

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

Refitting

Enable refitting before building acceleration structure:

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

Encode refit operation into a command buffer:

```
accelerationStructure.encodeRefit(commandBuffer: commandBuffer)
```

Rigid Body Animation

Most geometry only moves rigidly or not at all

Rigid Body Animation

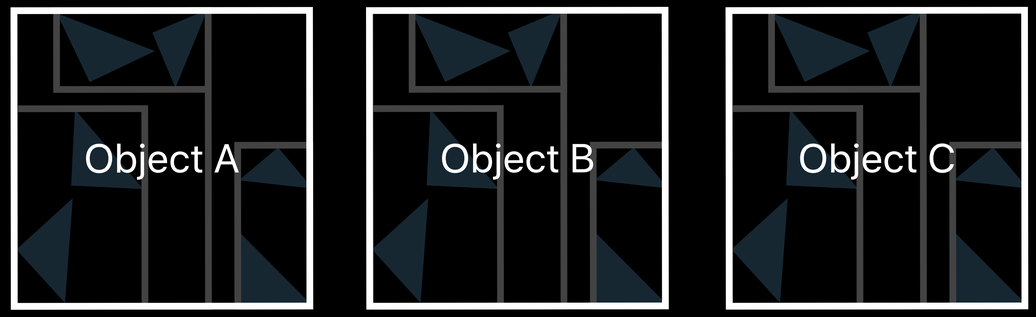
Most geometry only moves rigidly or not at all

Rigid Body Animation

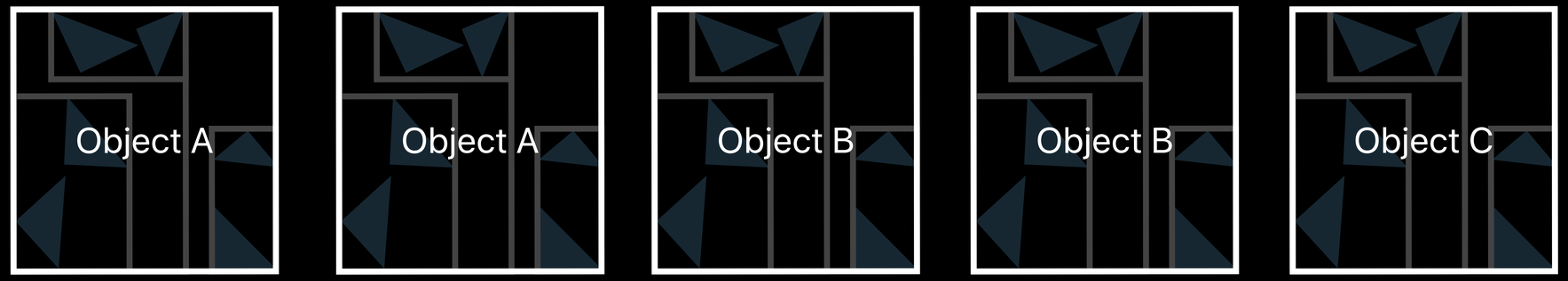
Most geometry only moves rigidly or not at all

May have multiple copies of the same objects

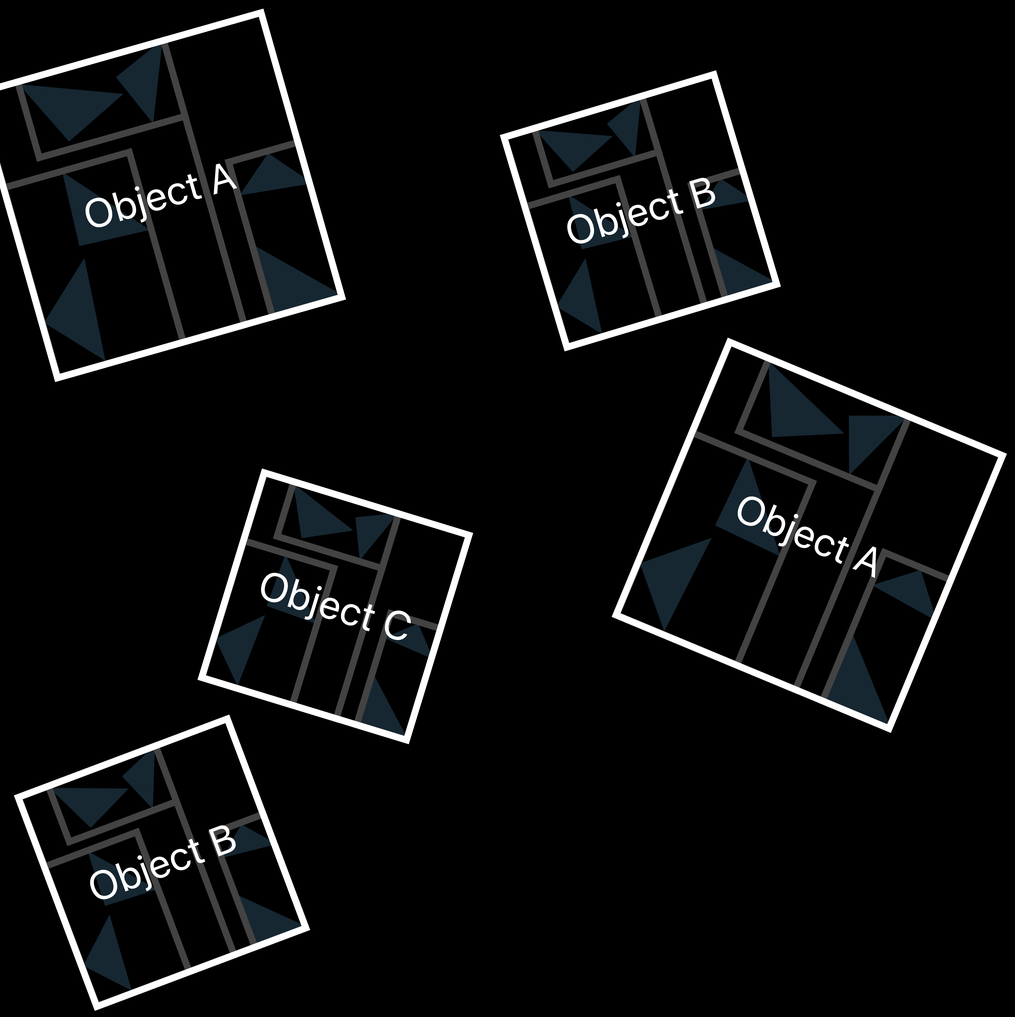
Two-Level Acceleration Structures



Two-Level Acceleration Structures



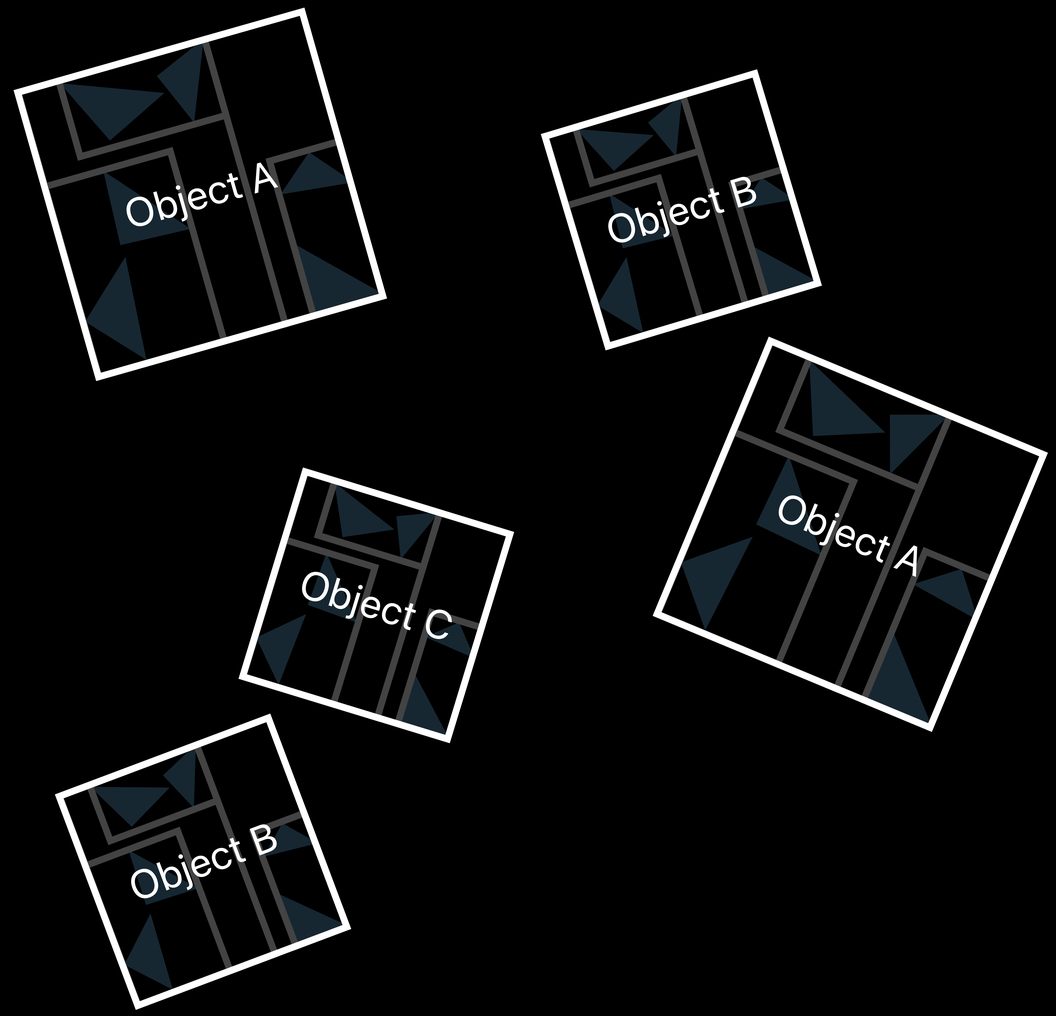
Two-Level Acceleration Structures



Two-Level Acceleration Structures

Transformation matrices:

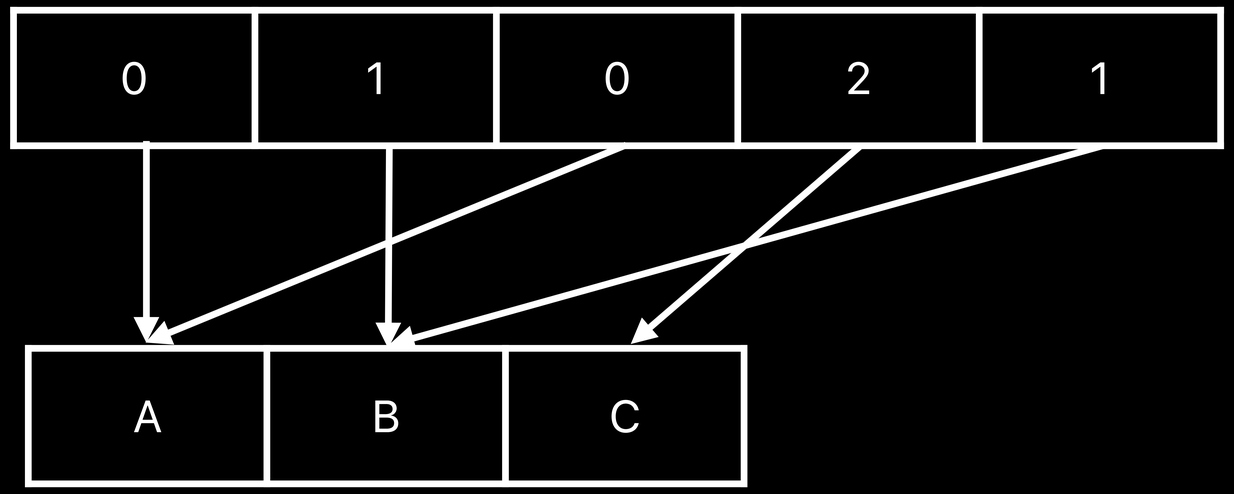
float4x4	float4x4	float4x4	float4x4	float4x4

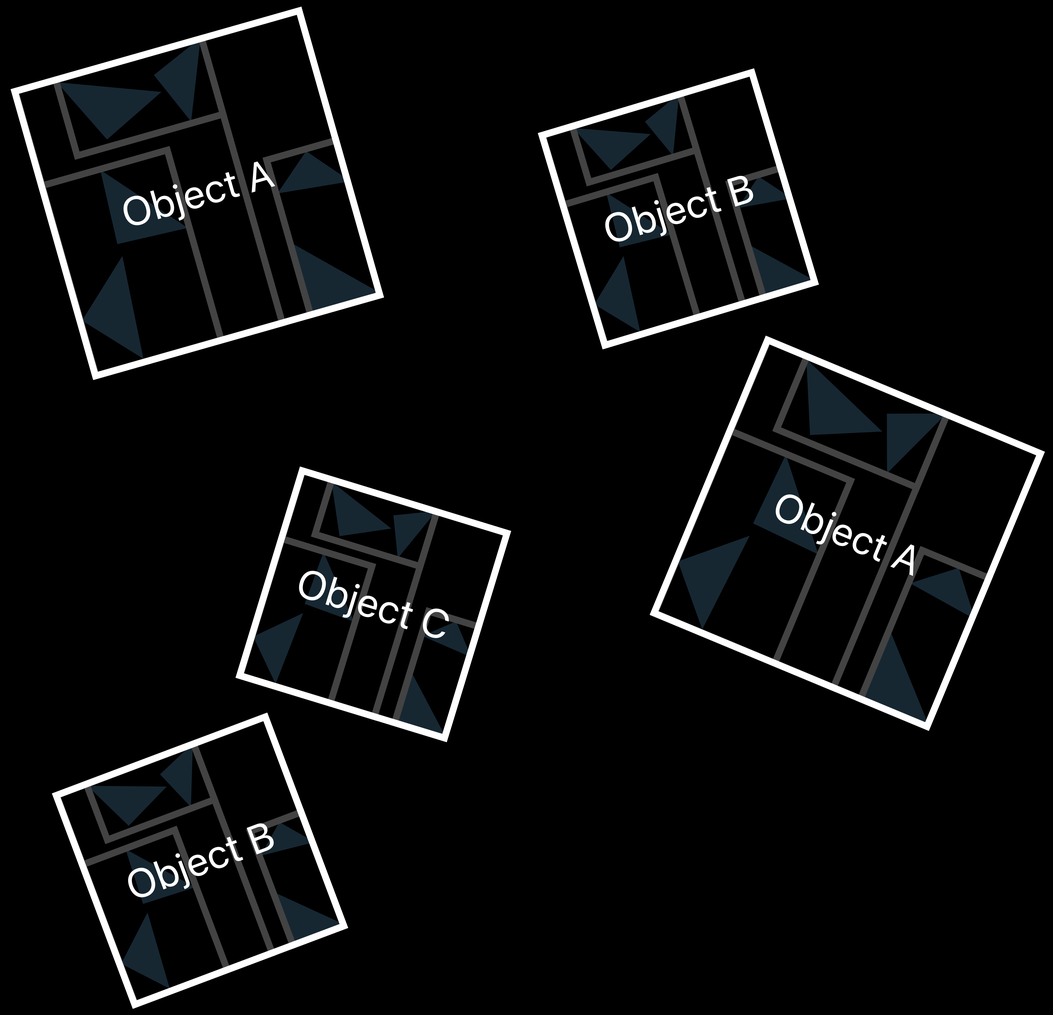


Two-Level Acceleration Structures

Transformation matrices:

Acceleration structure indices:

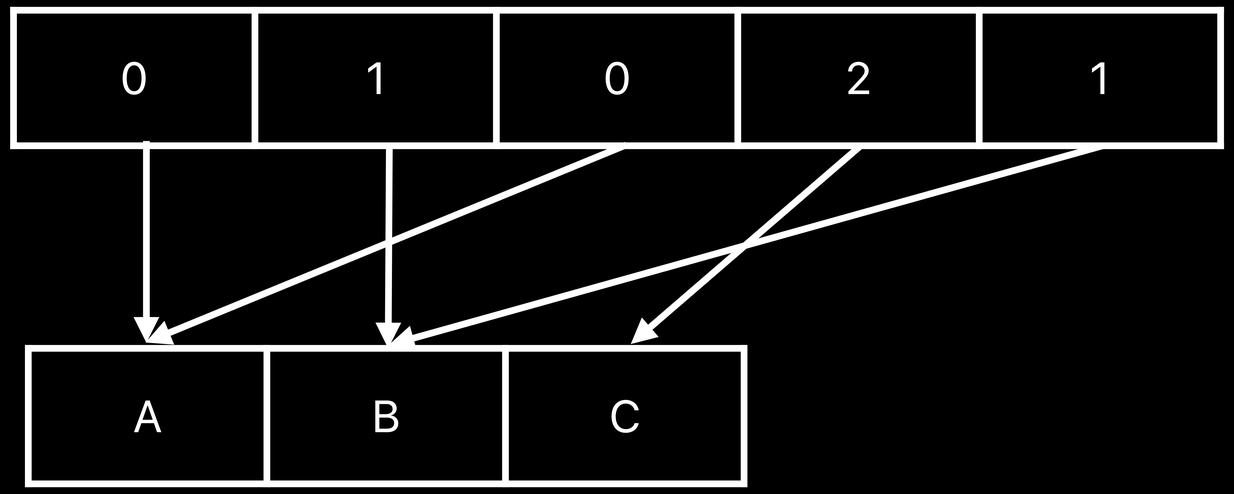




Two-Level Acceleration Structures

Transformation matrices:

Acceleration structure indices:



Two-Level Acceleration Structures

Build triangle acceleration structures:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

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

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

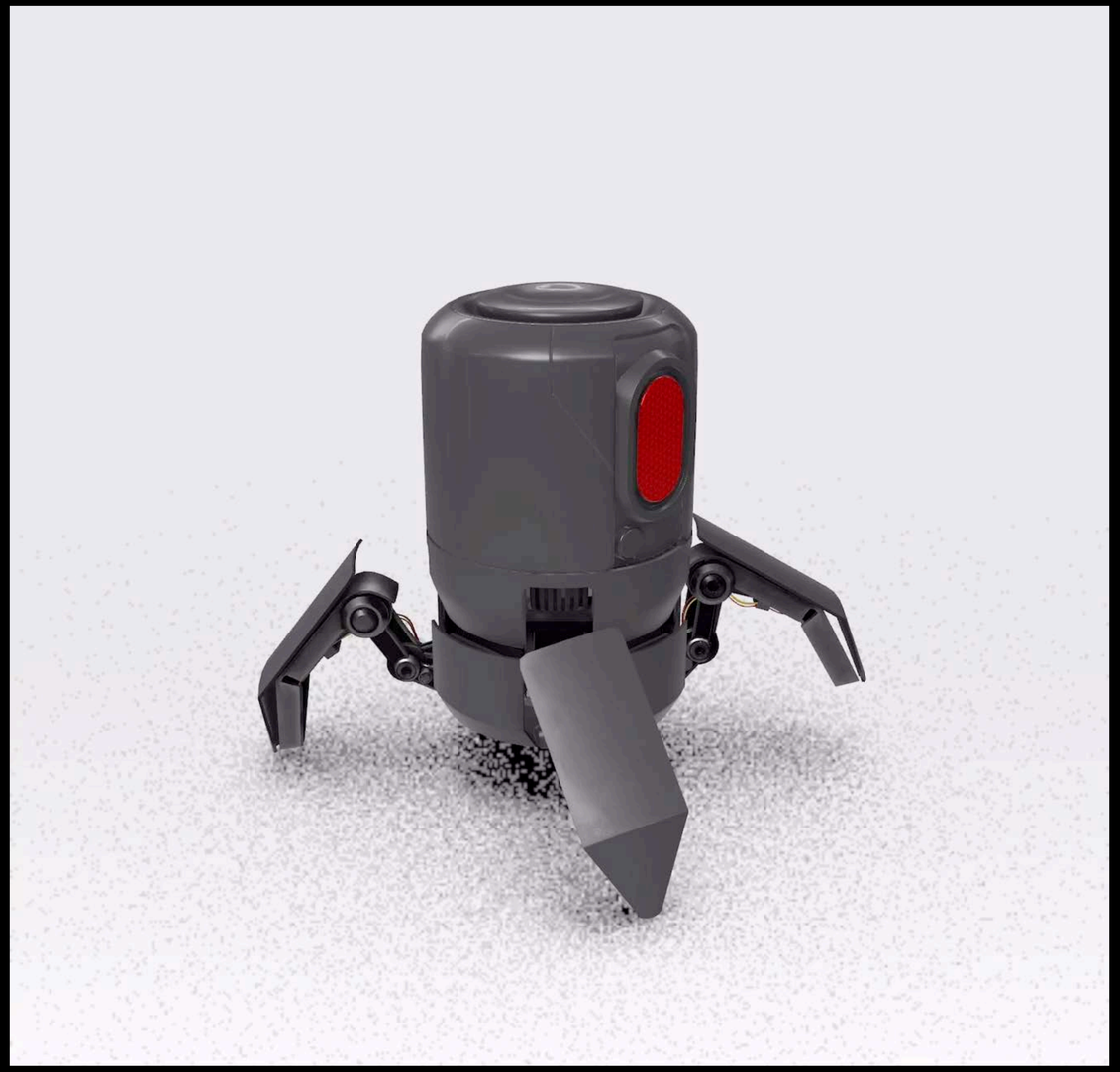
Rebuild when scene changes:

```
instanceAccelerationStructure.rebuild()
```

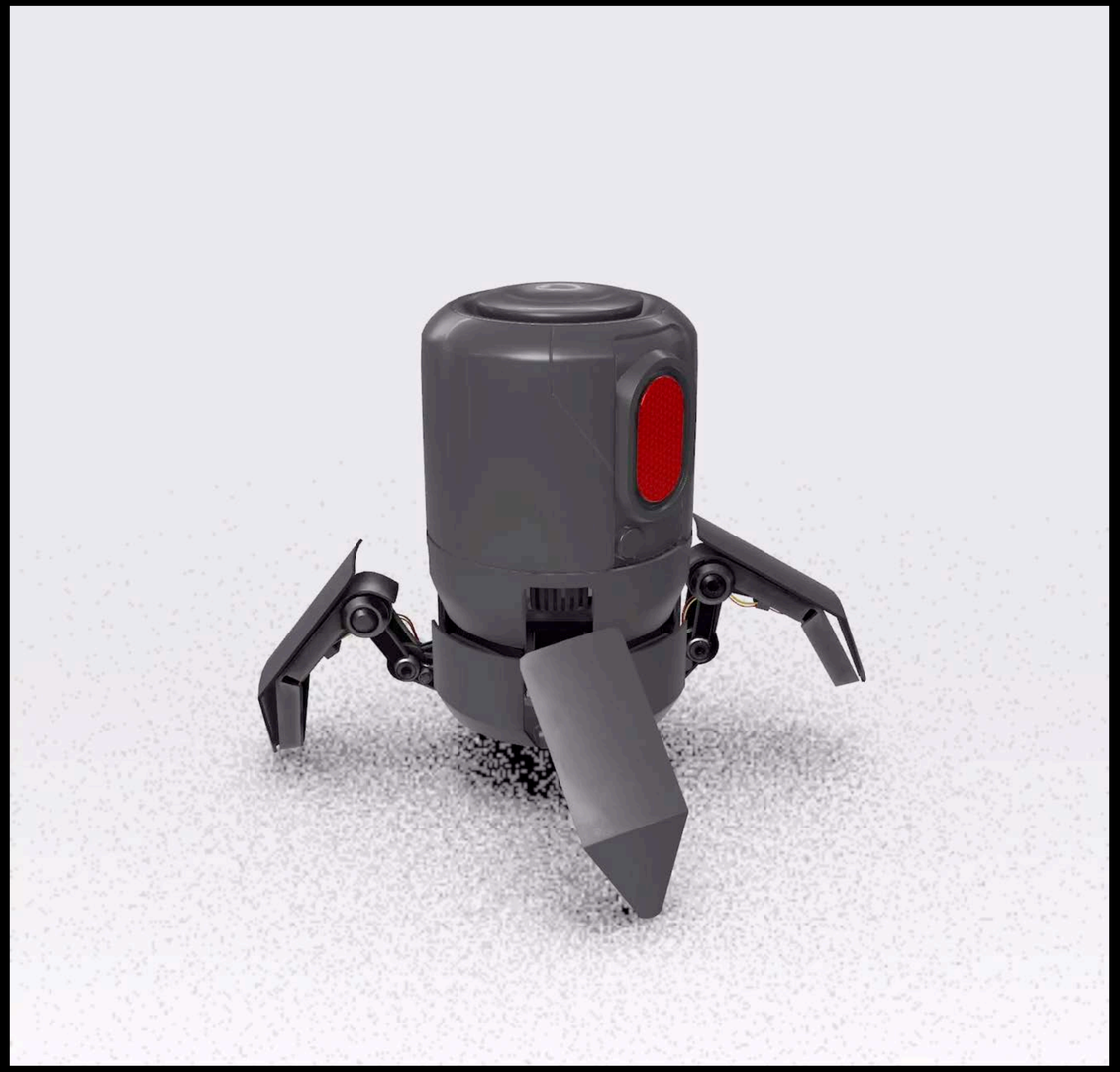
Denoising

Denoising

Denoising

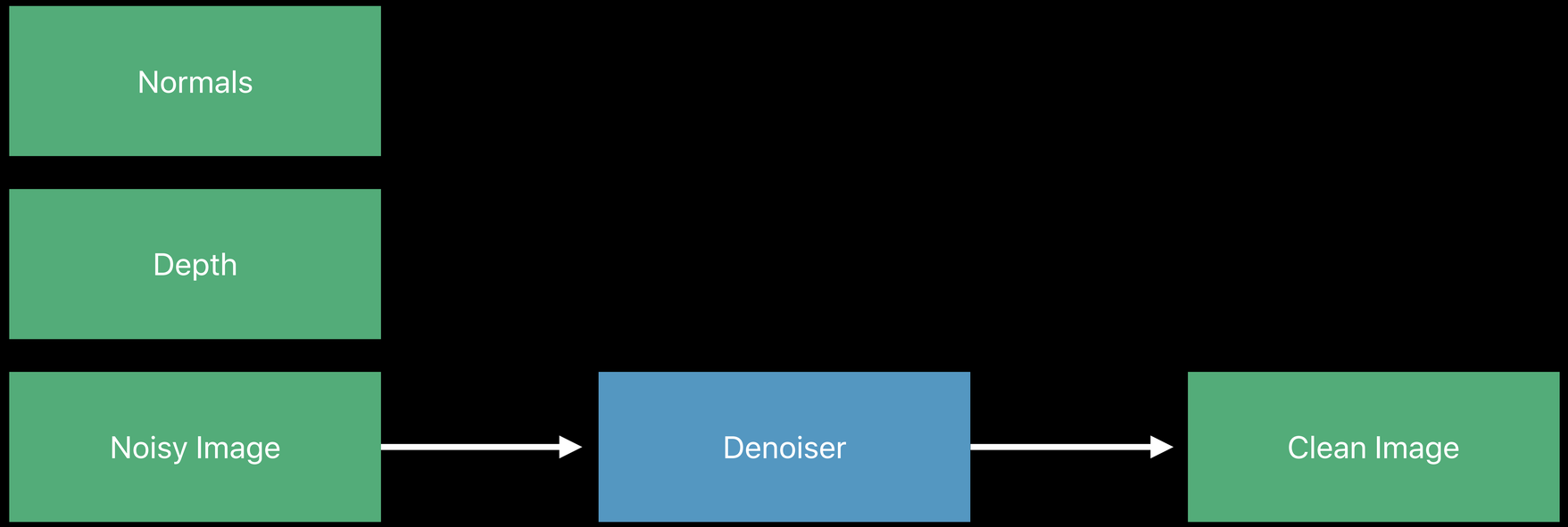


Denoising



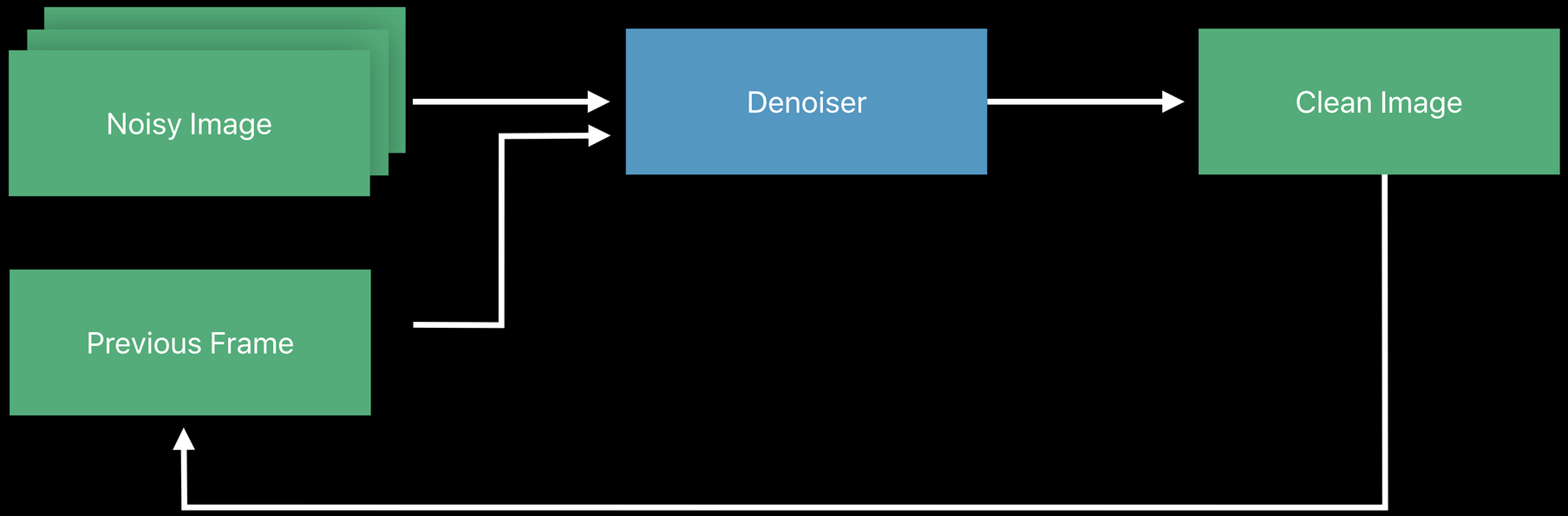
Denoising

Denoising

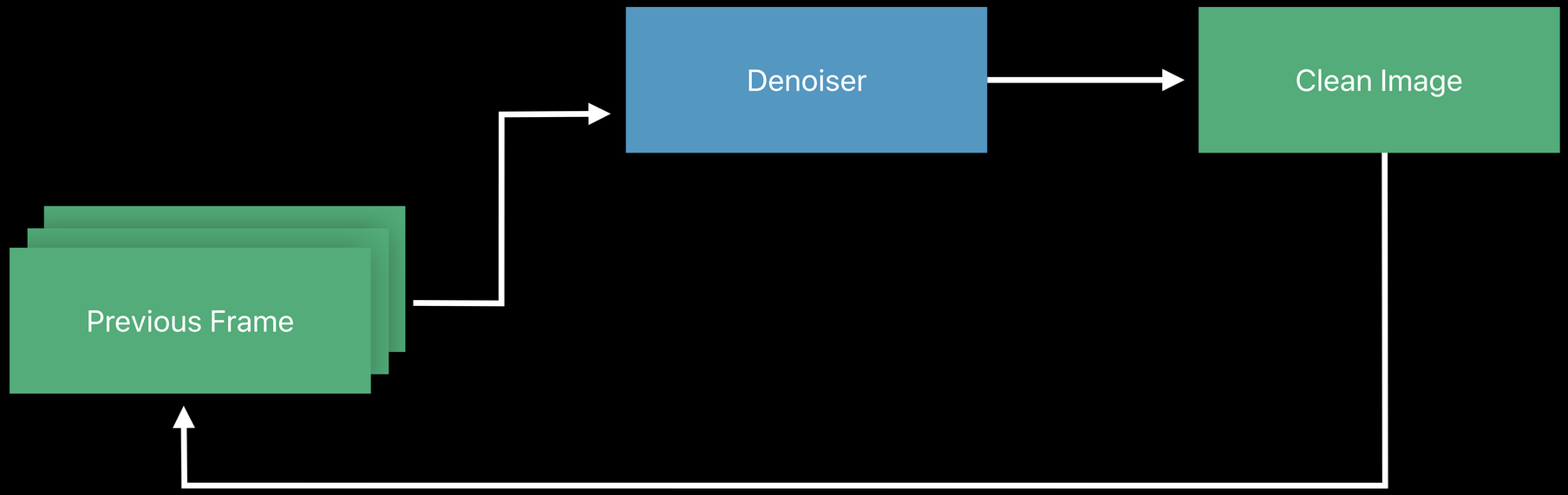


Denoising

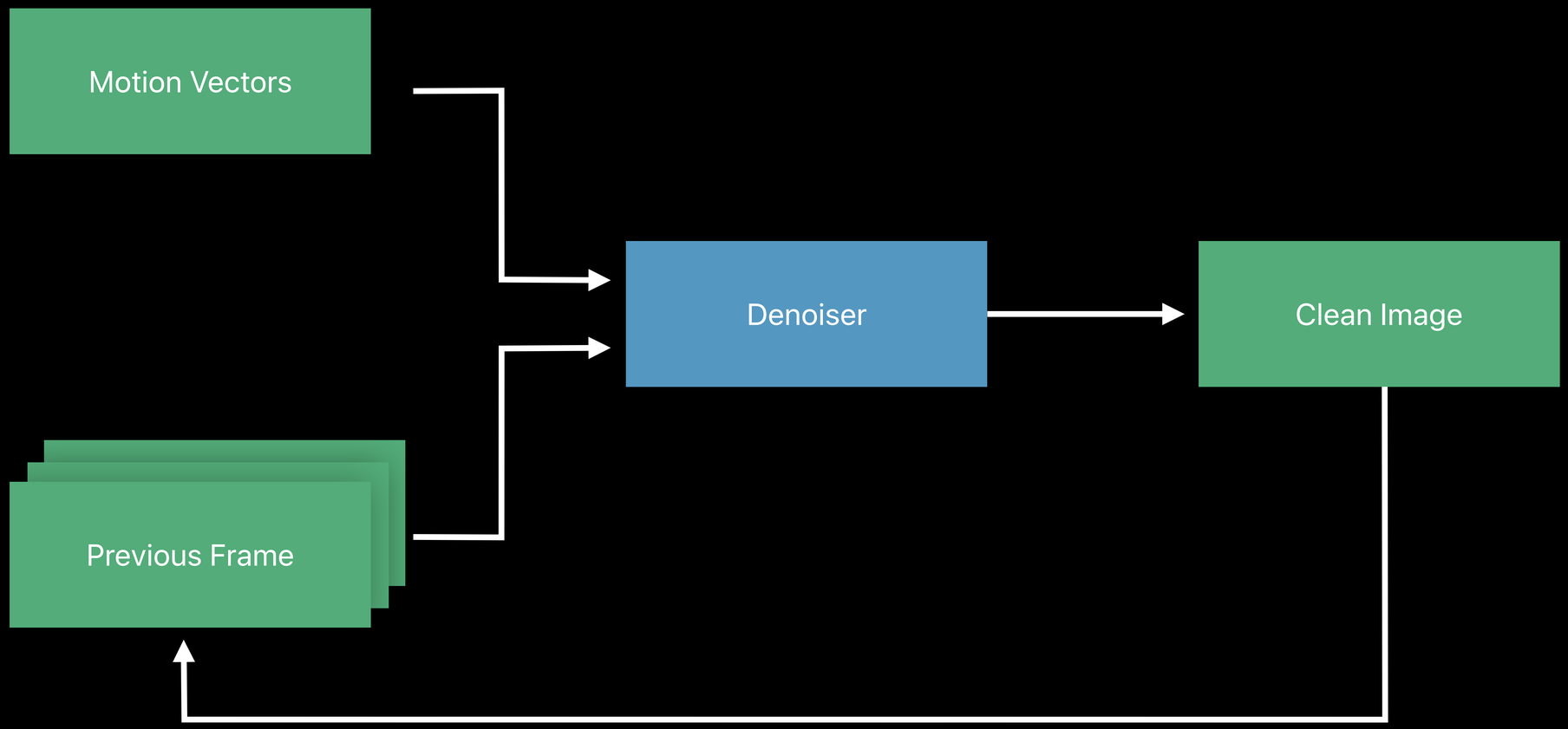
Denoising



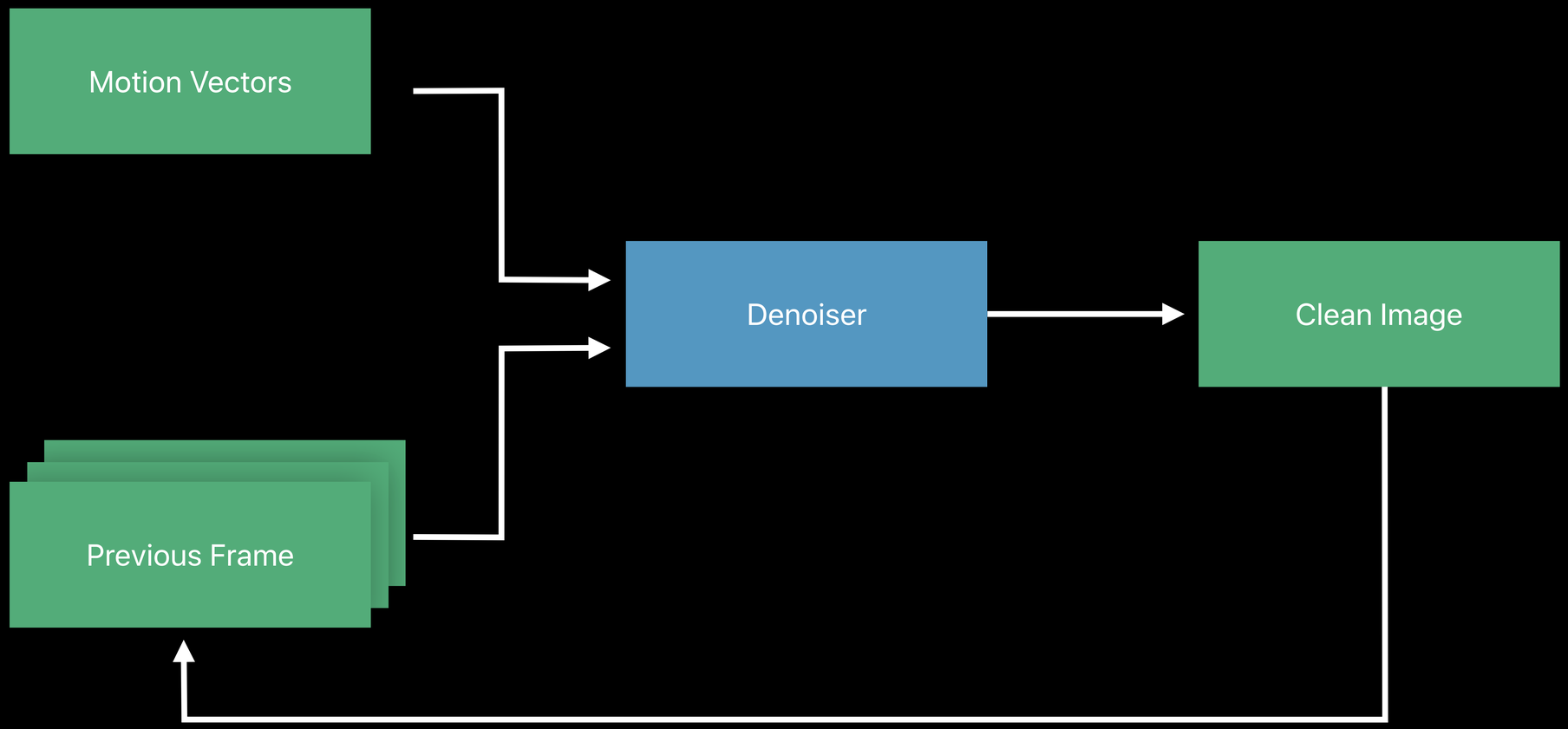
Denoising



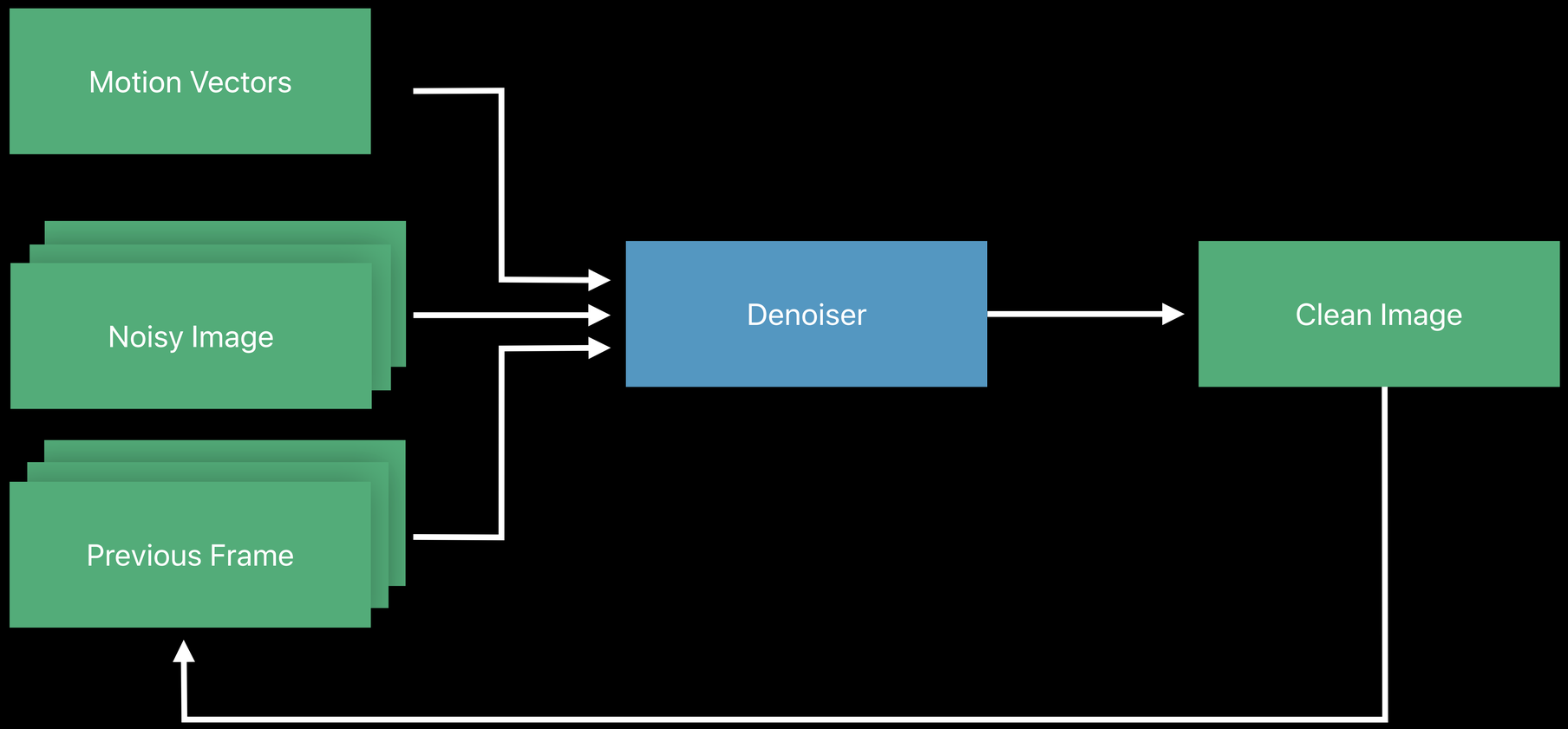
Denoising



Denoising



Denoising


Motion Vectors
Noisy Image
Previous Frame
Denoiser
Clean Image

MPSSVGF

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

MPSSVGF

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

MPSSVGFDenoiser coordinates denoising process

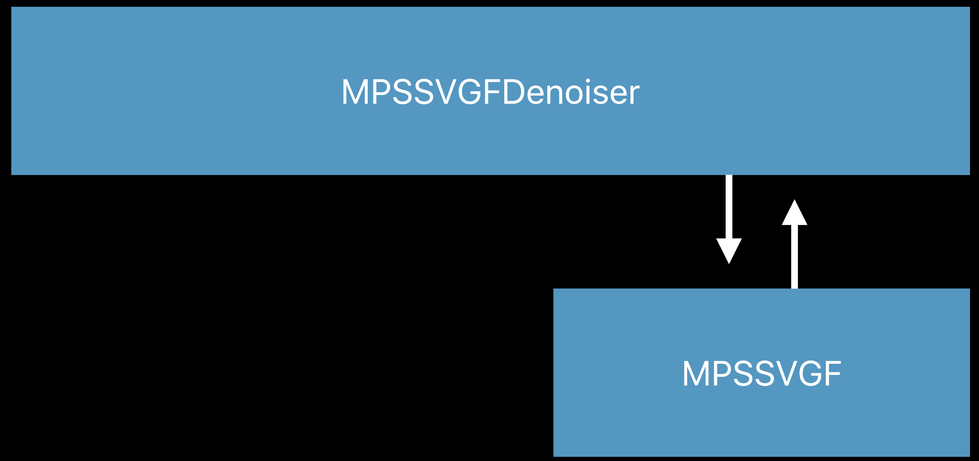
MPSSVGFDenoiser

MPSSVGF

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

MPSSVGFDenoiser coordinates denoising process

Low-level control

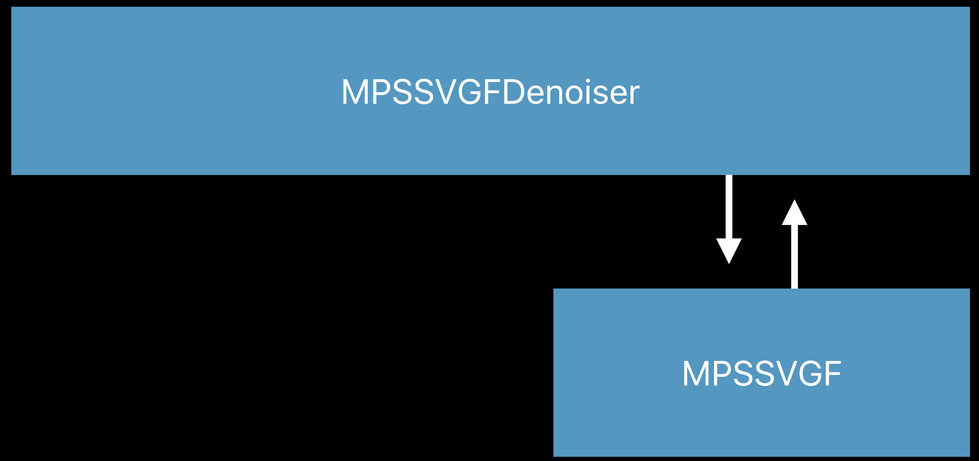


MPSSVGF

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

MPSSVGFDenoiser coordinates denoising process

Low-level control

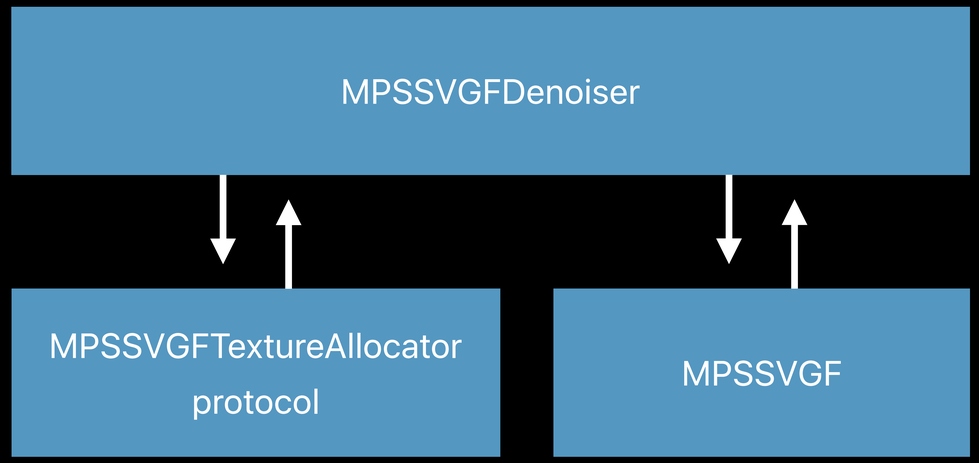


MPSSVGF

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

MPSSVGFDenoiser coordinates denoising process

Low-level control



MPSSVGFDenoiser

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 = MPSSVGFDenoiser(SVGF: svgf, textureAllocator: textureAllocator)
```

MPSSVGFDenoiser

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 = MPSSVGFDenoiser(SVGF: svgf, textureAllocator: textureAllocator)
```

MPSSVGFDenoiser

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 = MPSSVGFDenoiser(SVGF: svgf, textureAllocator: textureAllocator)
```

MPSSVGFDenoiser

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 = MPSSVGFDenoiser(SVGF: svgf, textureAllocator: textureAllocator)
```

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
```

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

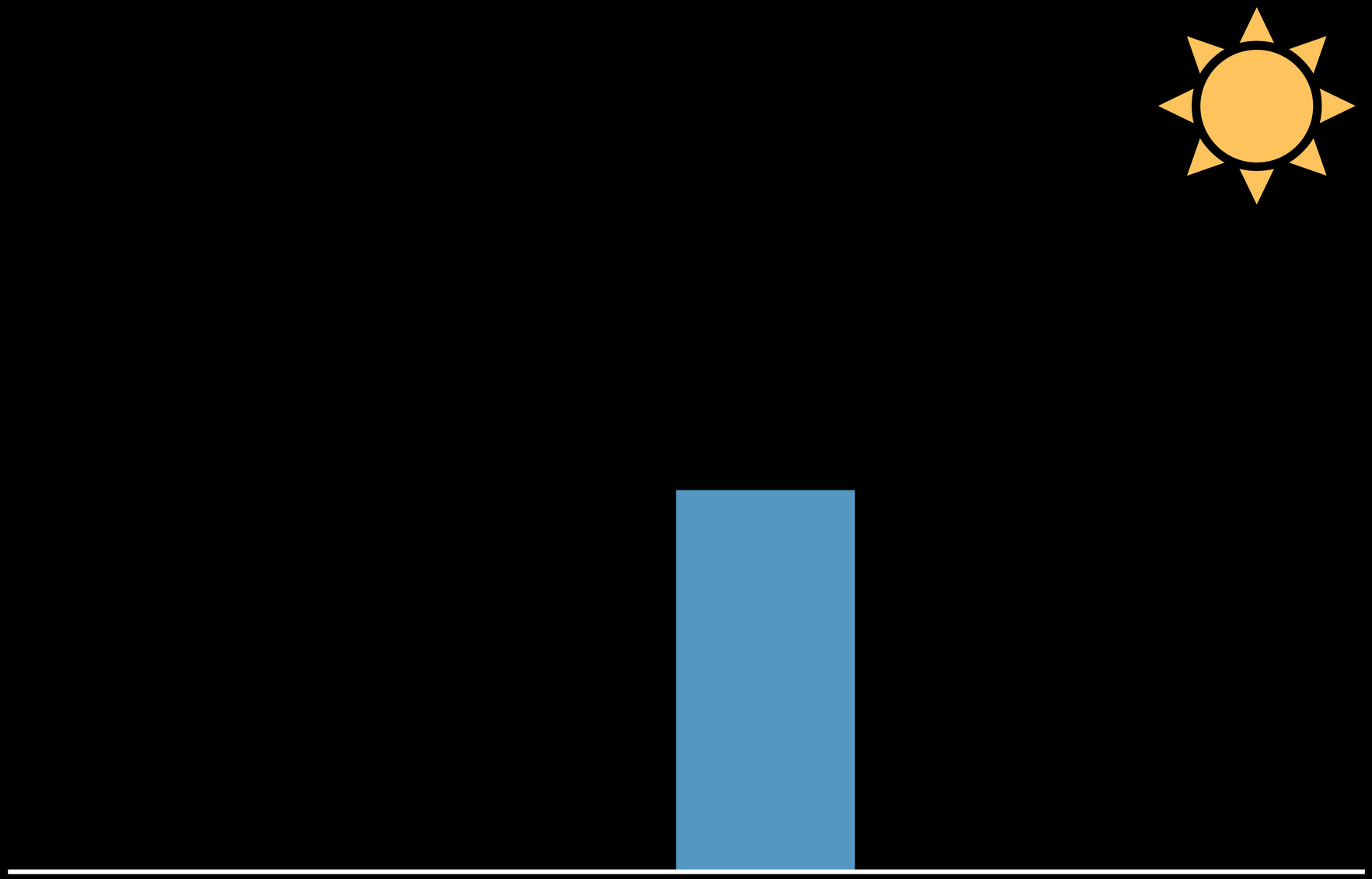
Ray/triangle intersection

Dynamic scenes

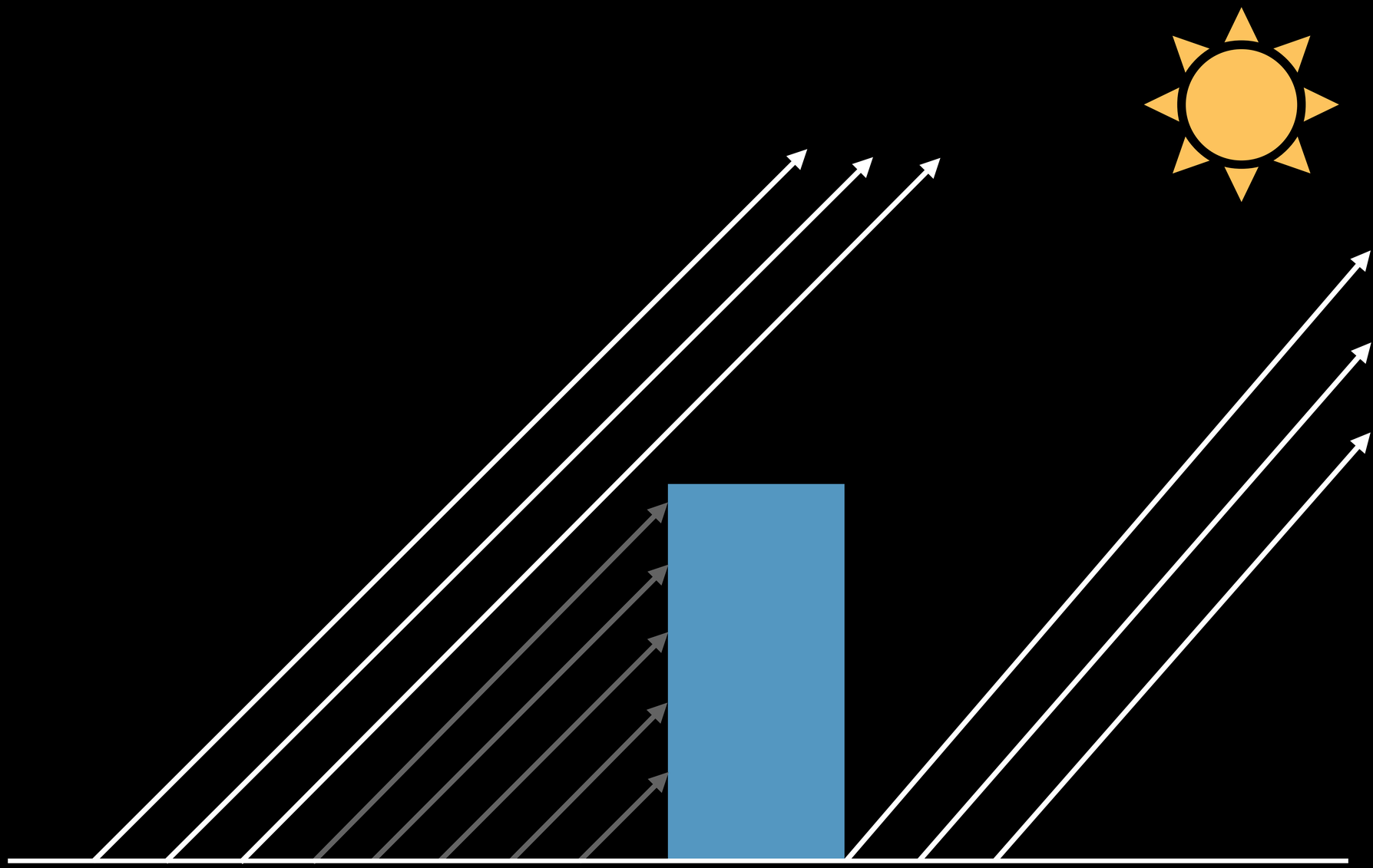
Denoising

Ray Tracing in Practice

Hard Shadows



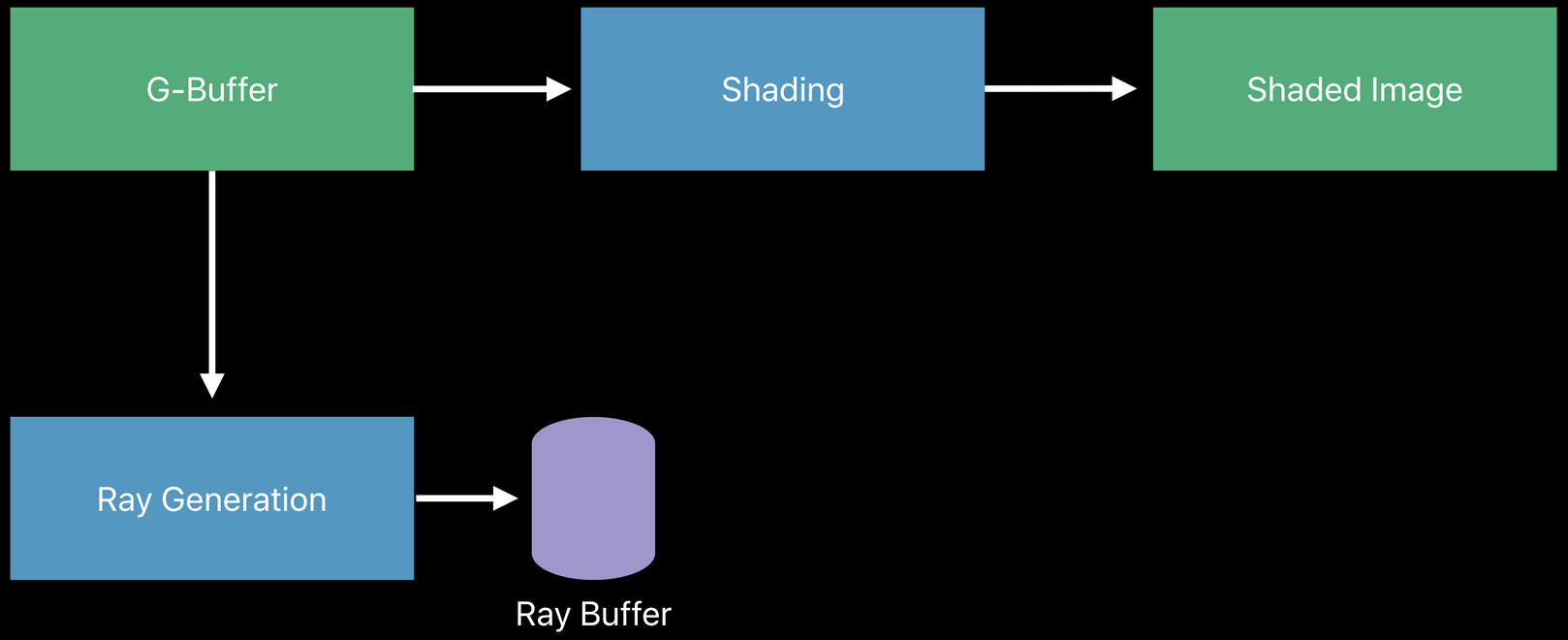
Hard Shadows



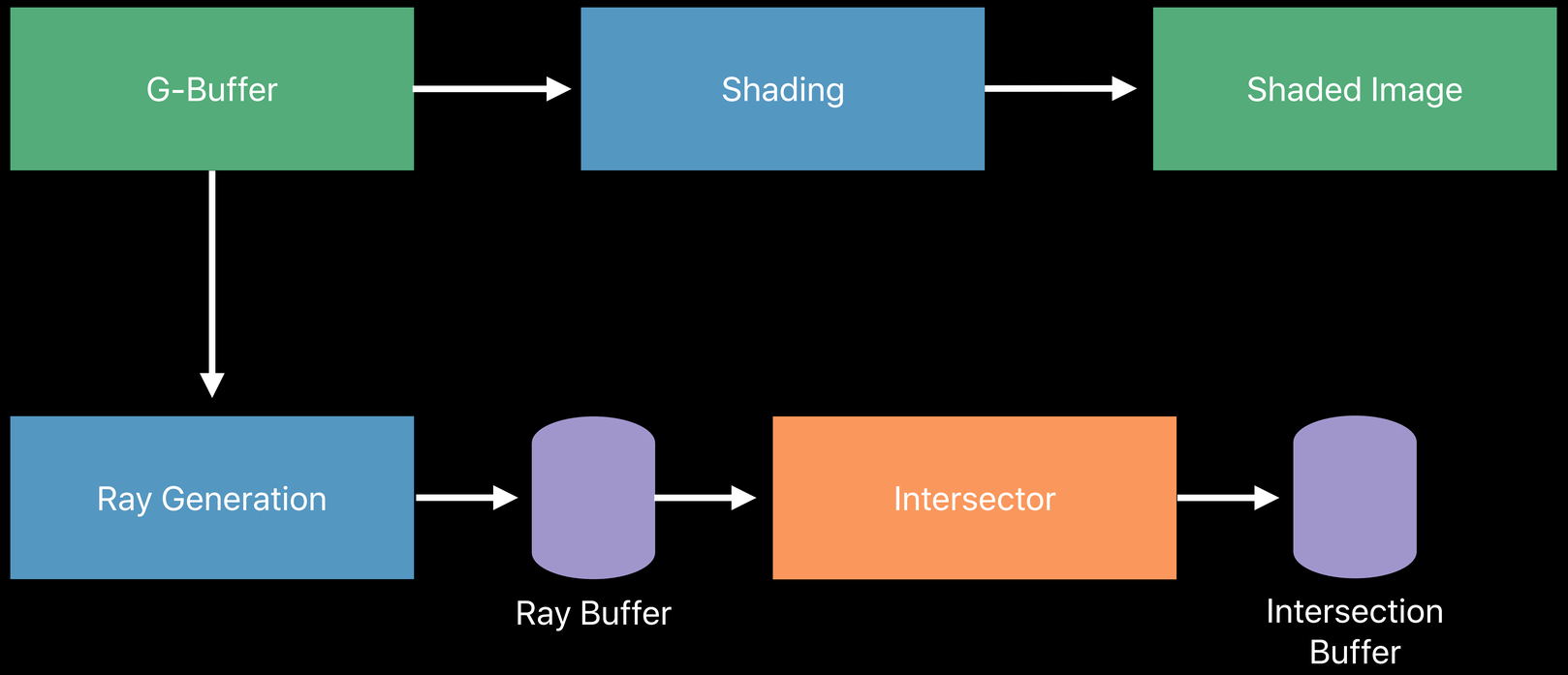
Hybrid Rendering

Hybrid Rendering

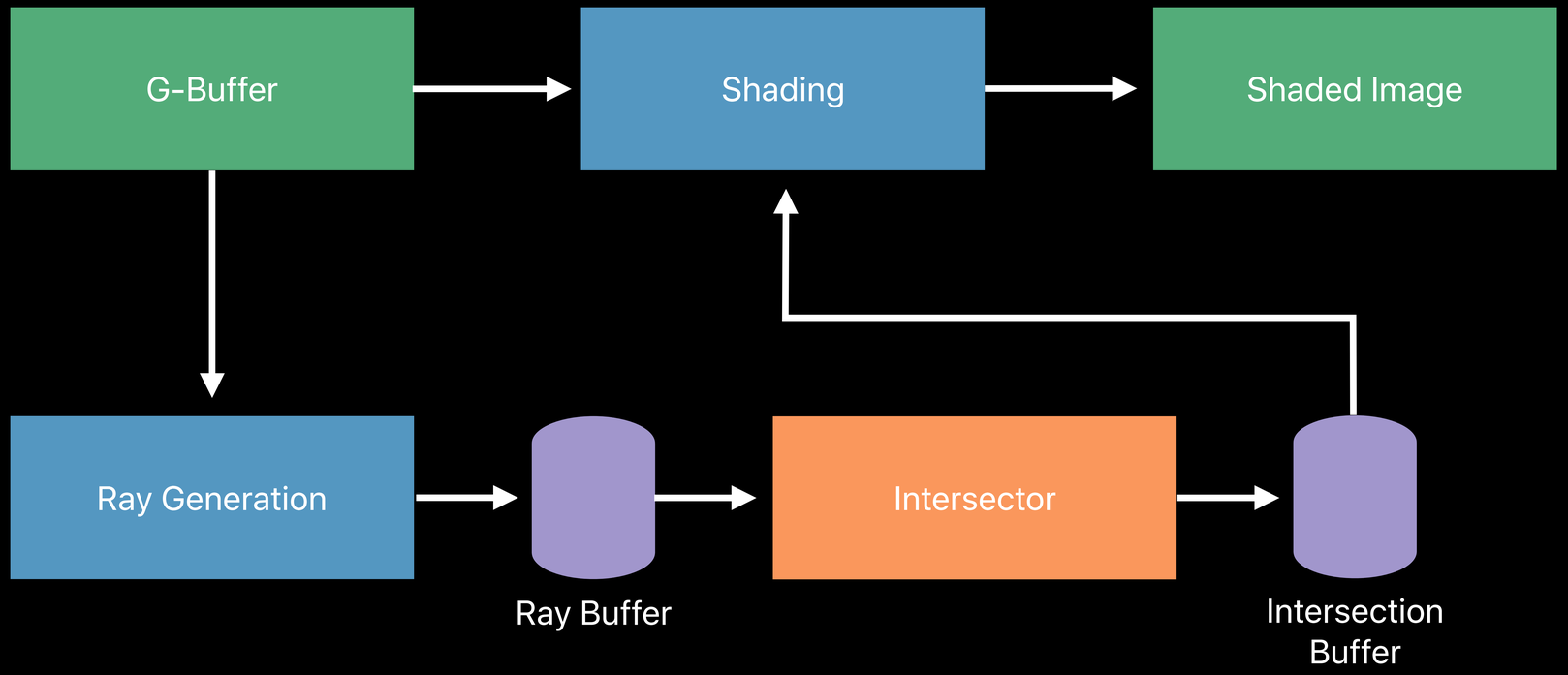
Hybrid Rendering



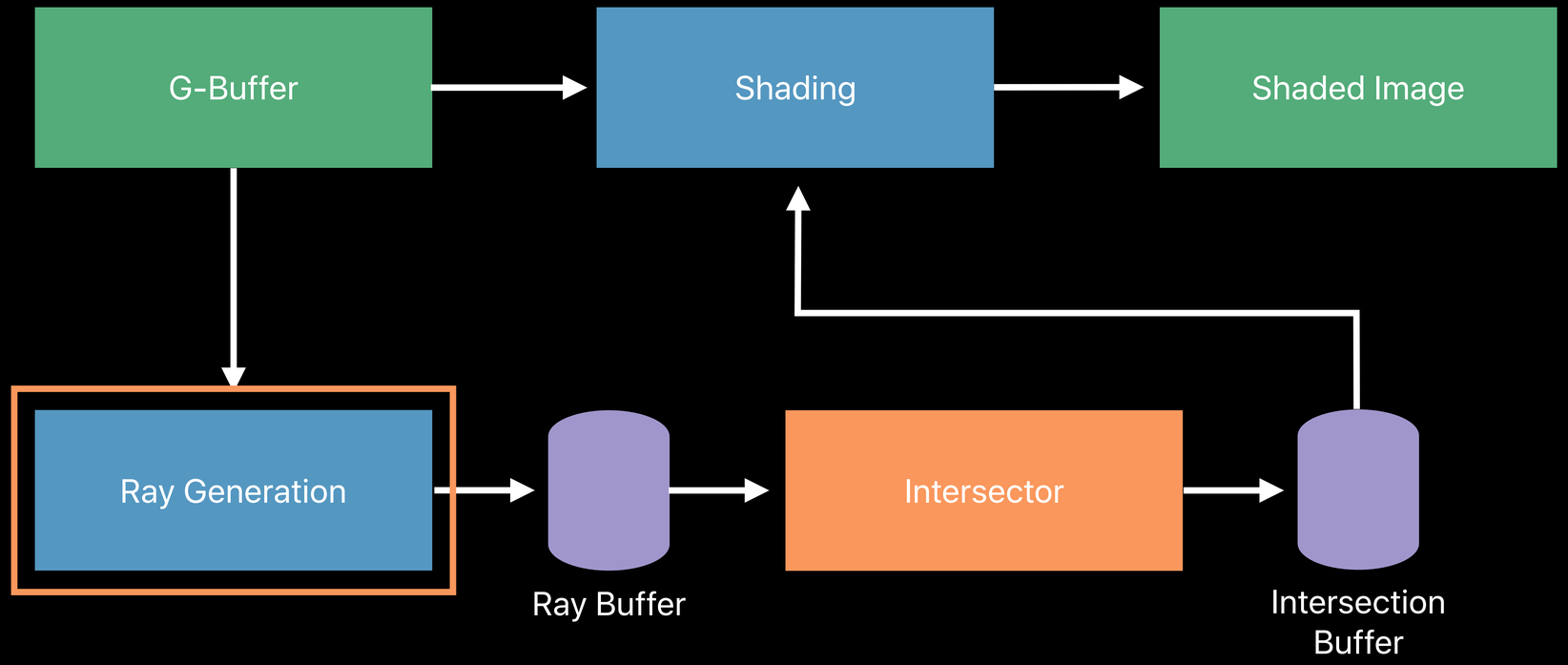
Hybrid Rendering



Hybrid Rendering



Hybrid Rendering



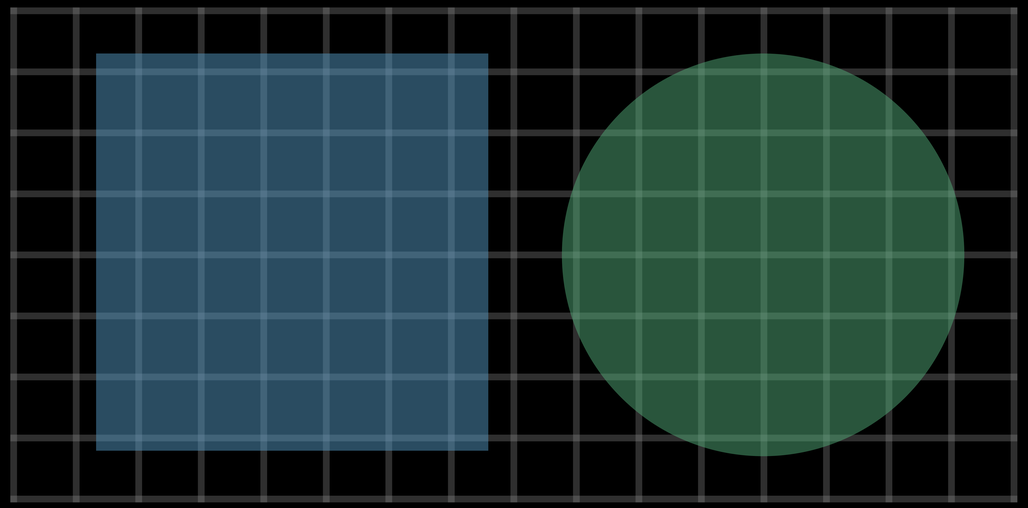
Ray Generation

```
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

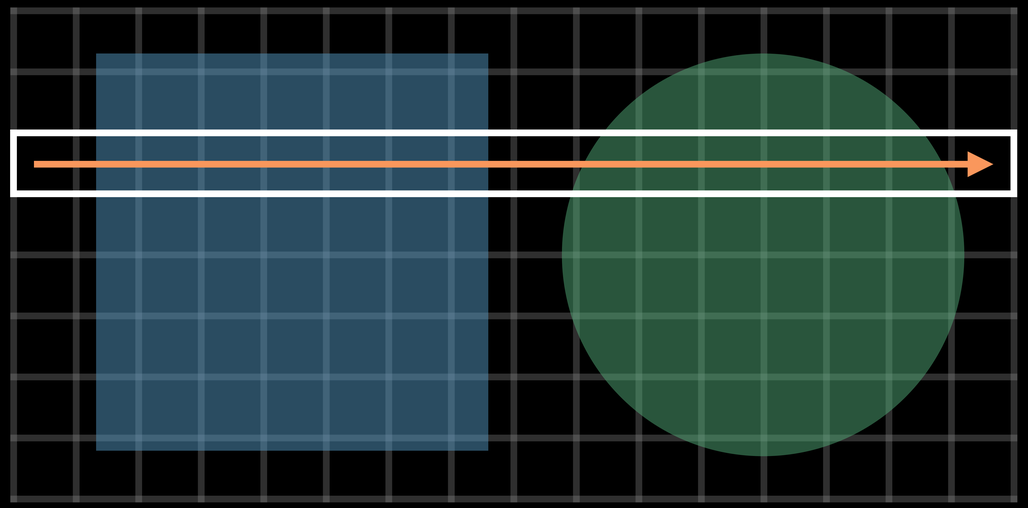


Row Linear

Ray Coherency

Metal processes rays in the order you specify them

Block linear layout can improve performance

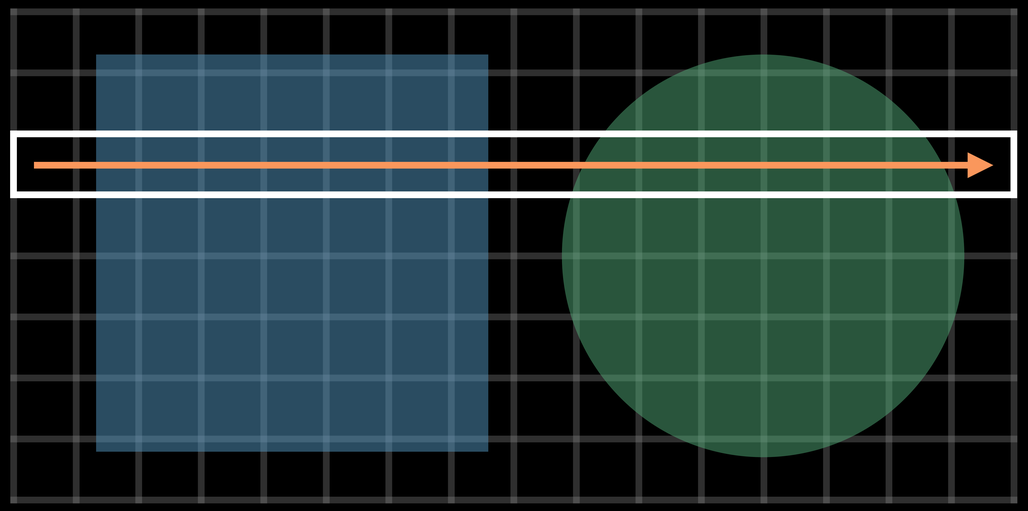


Row Linear

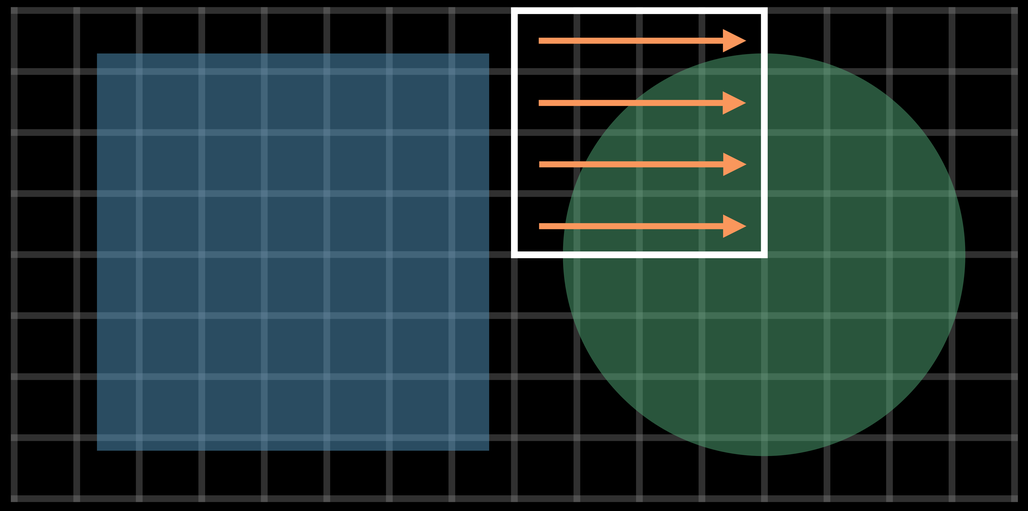
Ray Coherency

Metal processes rays in the order you specify them

Block linear layout can improve performance



Row Linear



Block Linear

Disabling Rays

Not all pixels need a shadow ray

- Background pixels
- Surfaces facing away from the Sun

Disable rays by setting maxDistance < 0.0

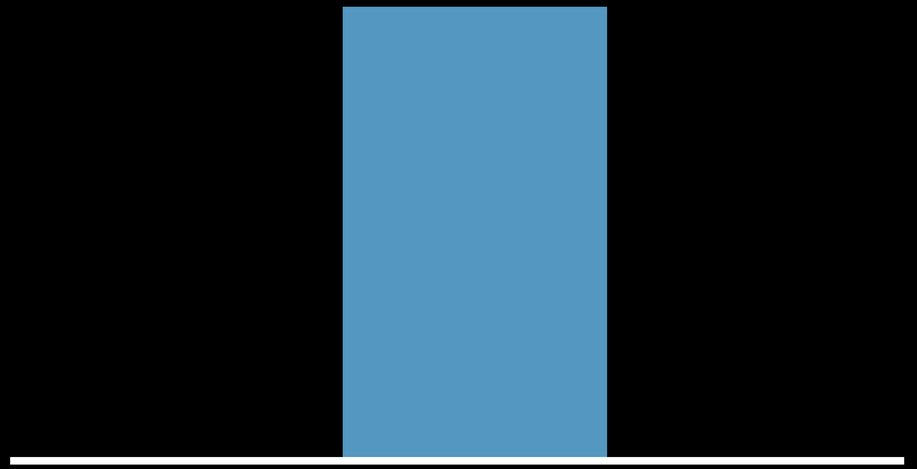
Hard Shadows

Soft Shadows

Soft Shadows

Extend a cone from the surface towards the Sun

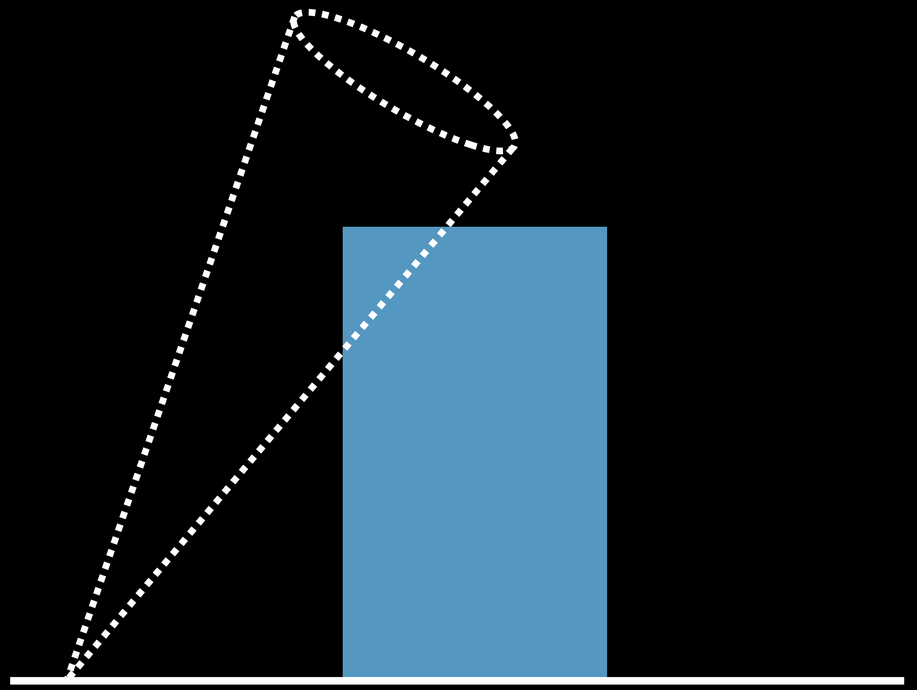
Generate ray directions randomly within this cone



Soft Shadows

Extend a cone from the surface towards the Sun

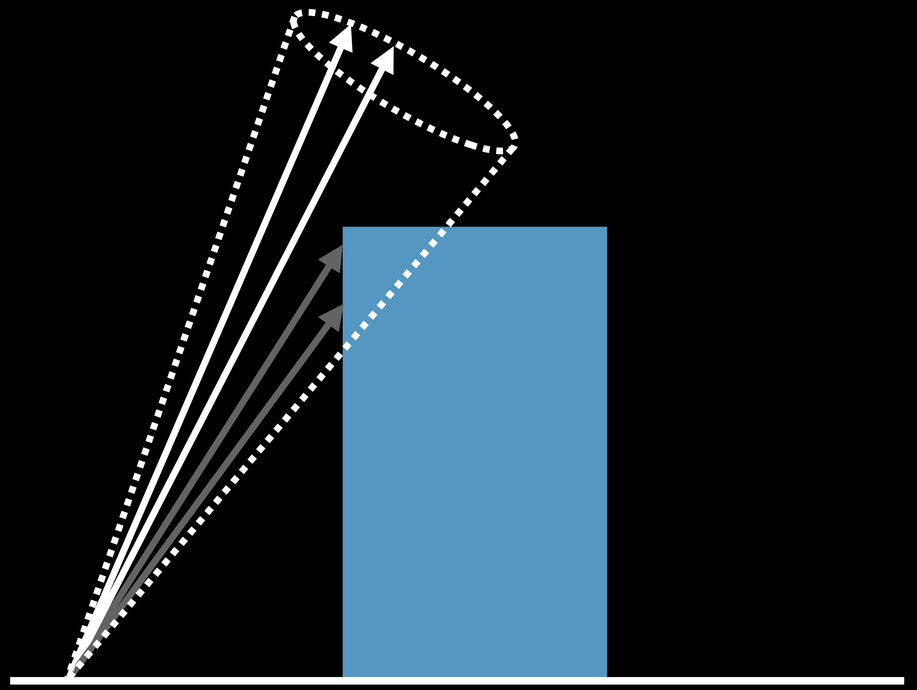
Generate ray directions randomly within this cone



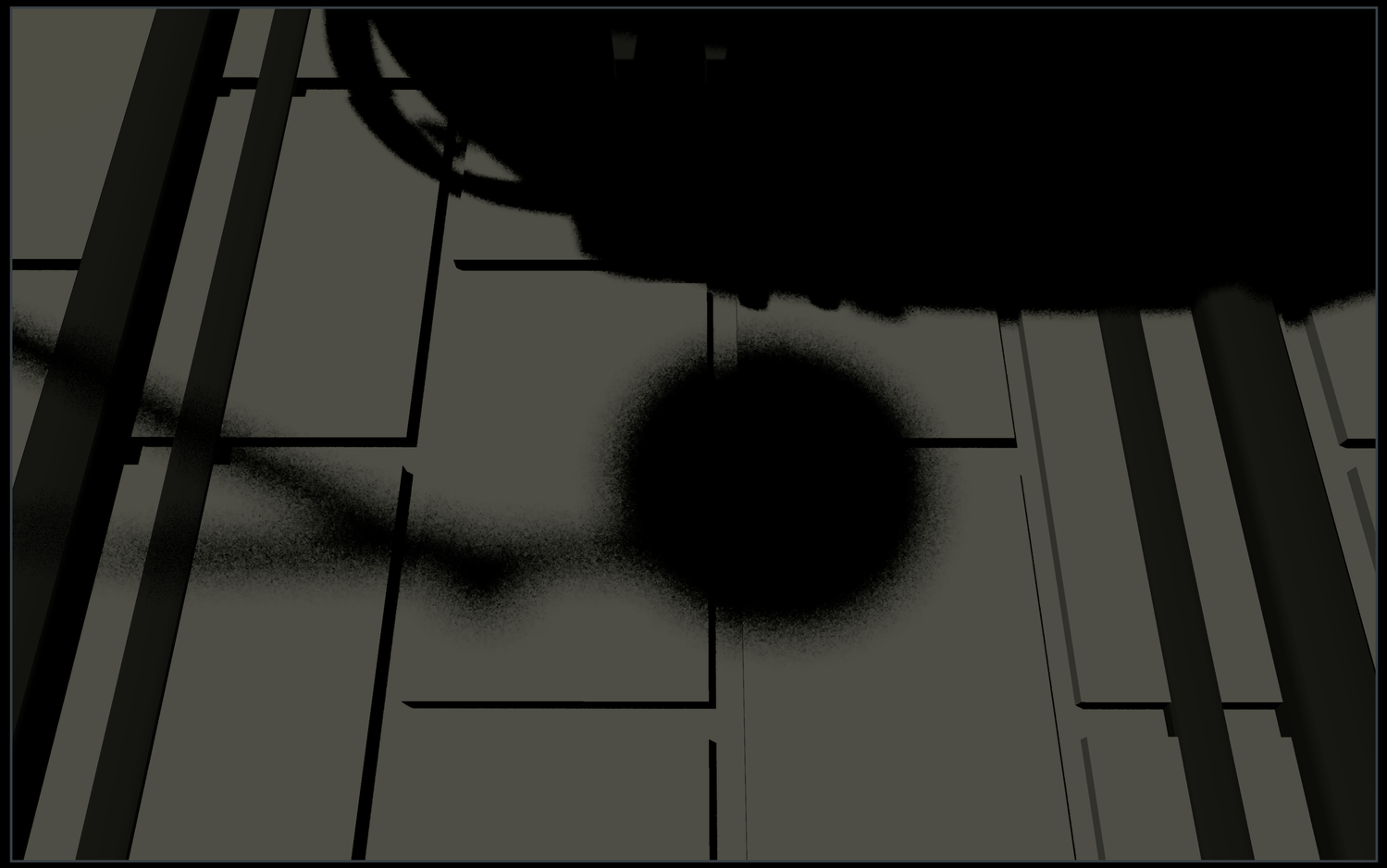
Soft Shadows

Extend a cone from the surface towards the Sun

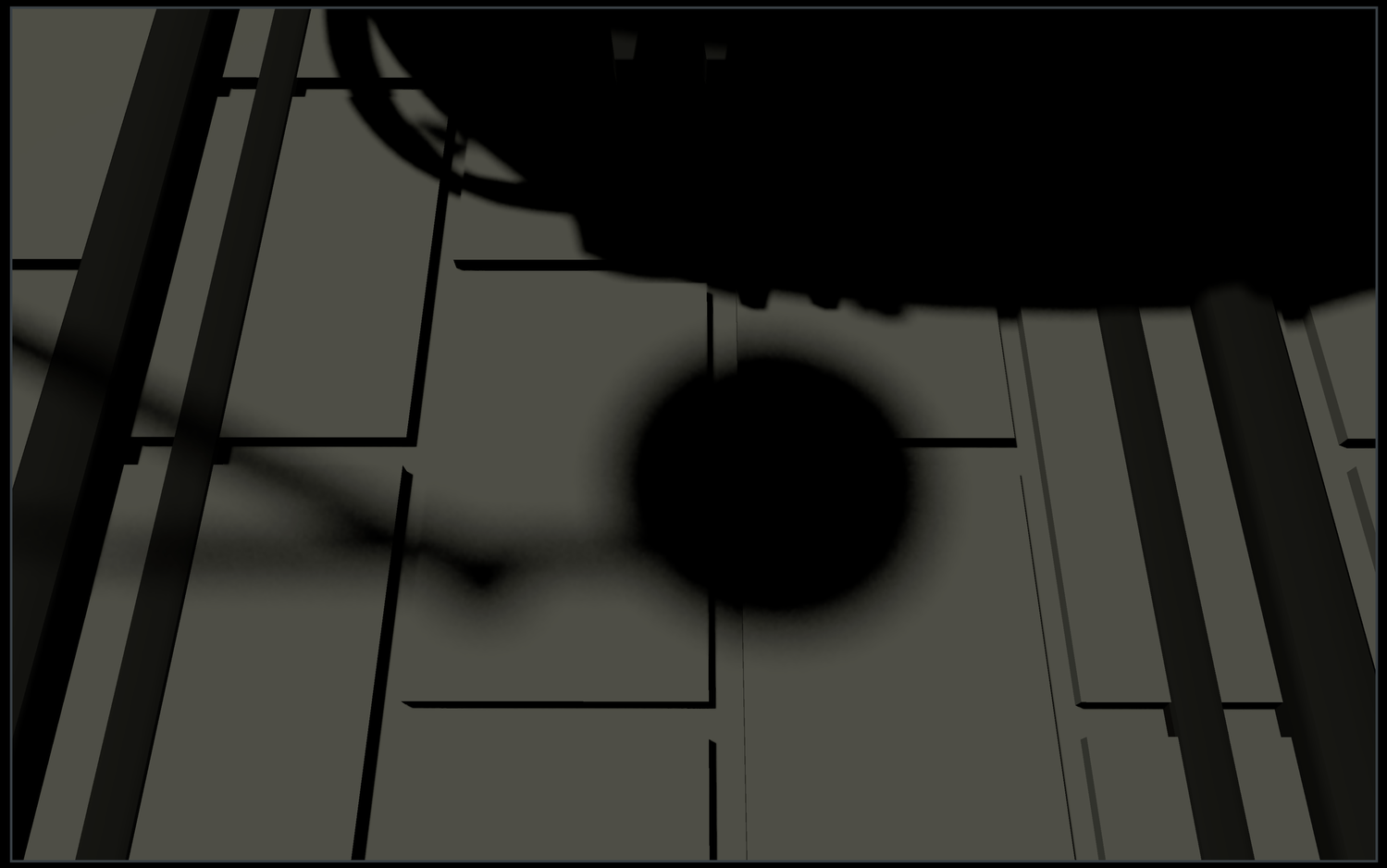
Generate ray directions randomly within this cone



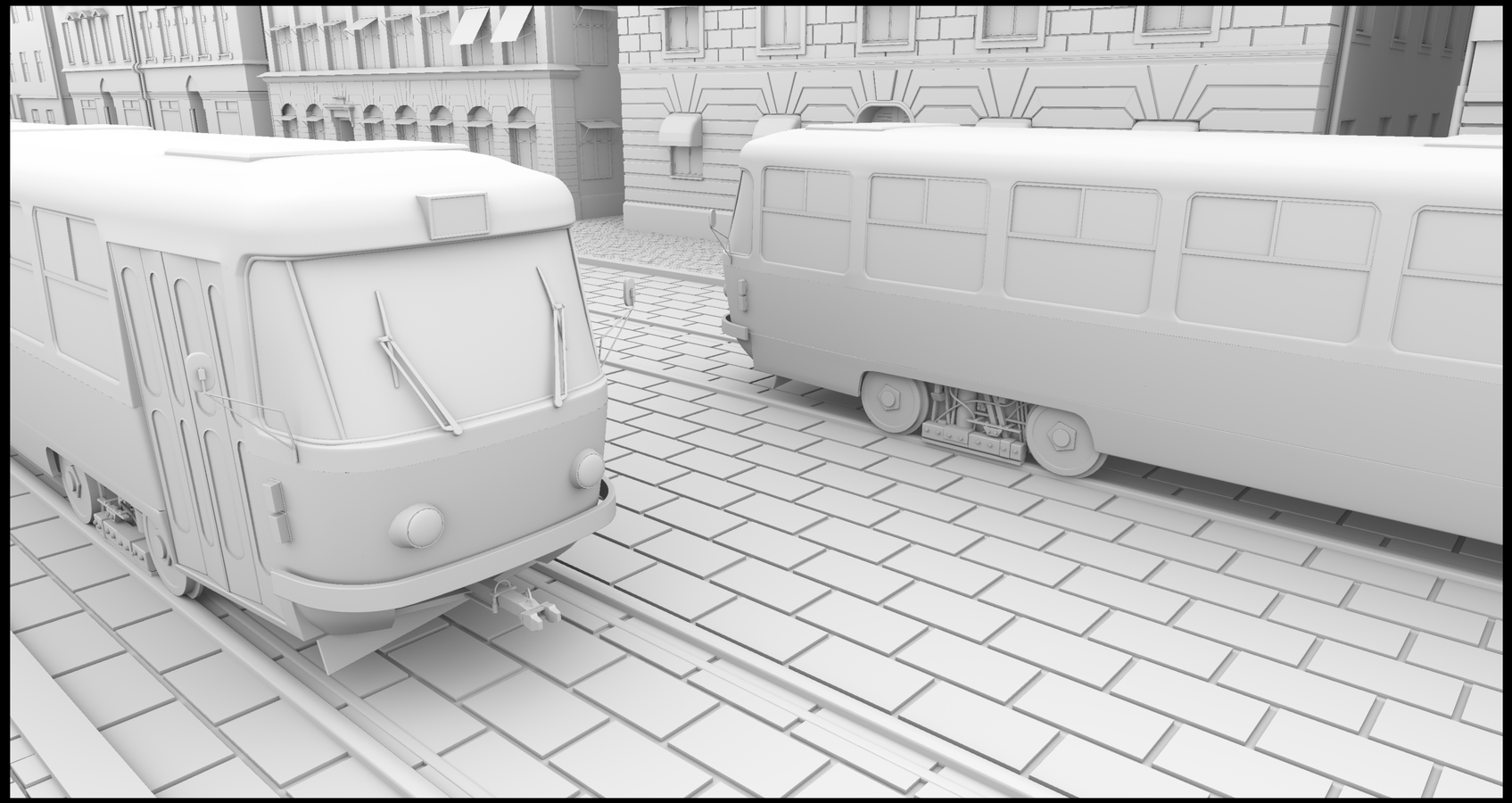
Shadow Term



Denoised



Ambient Occlusion



Ambient Occlusion

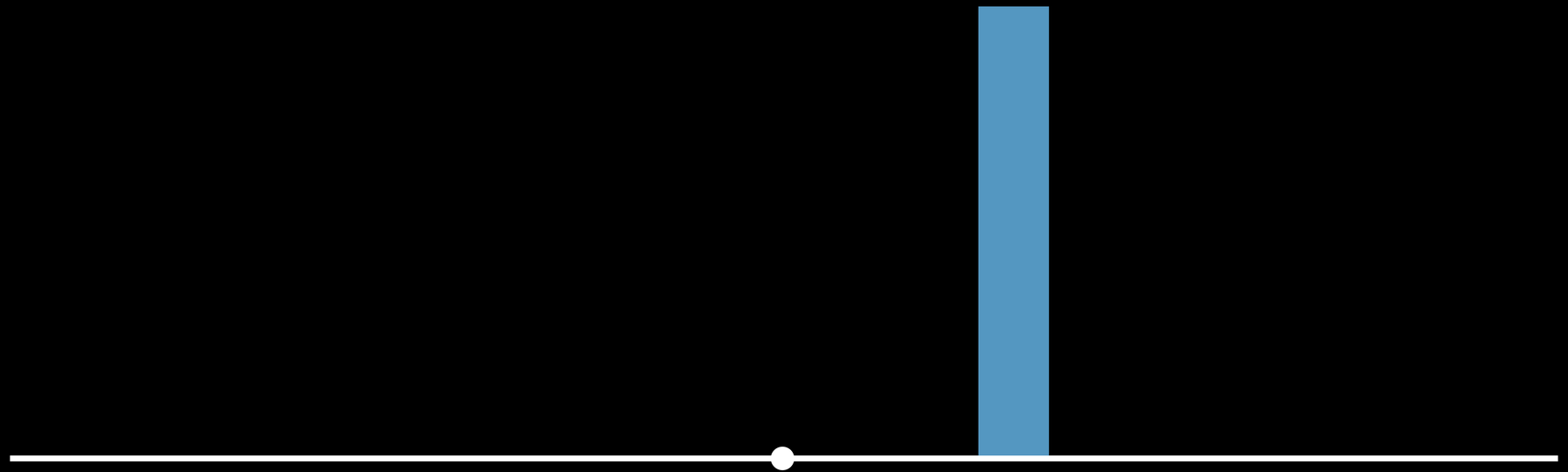
Use rays to estimate how much ambient light reaches a surface

Falloff based on angle and intersection distance

Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

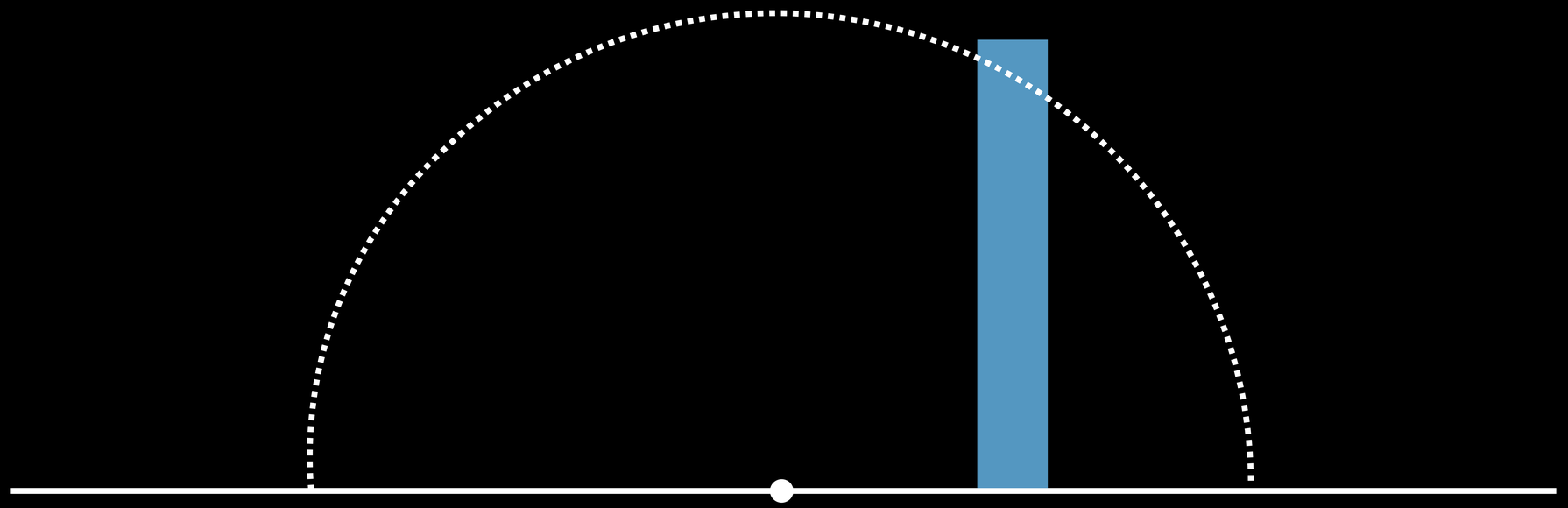
Falloff based on angle and intersection distance



Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

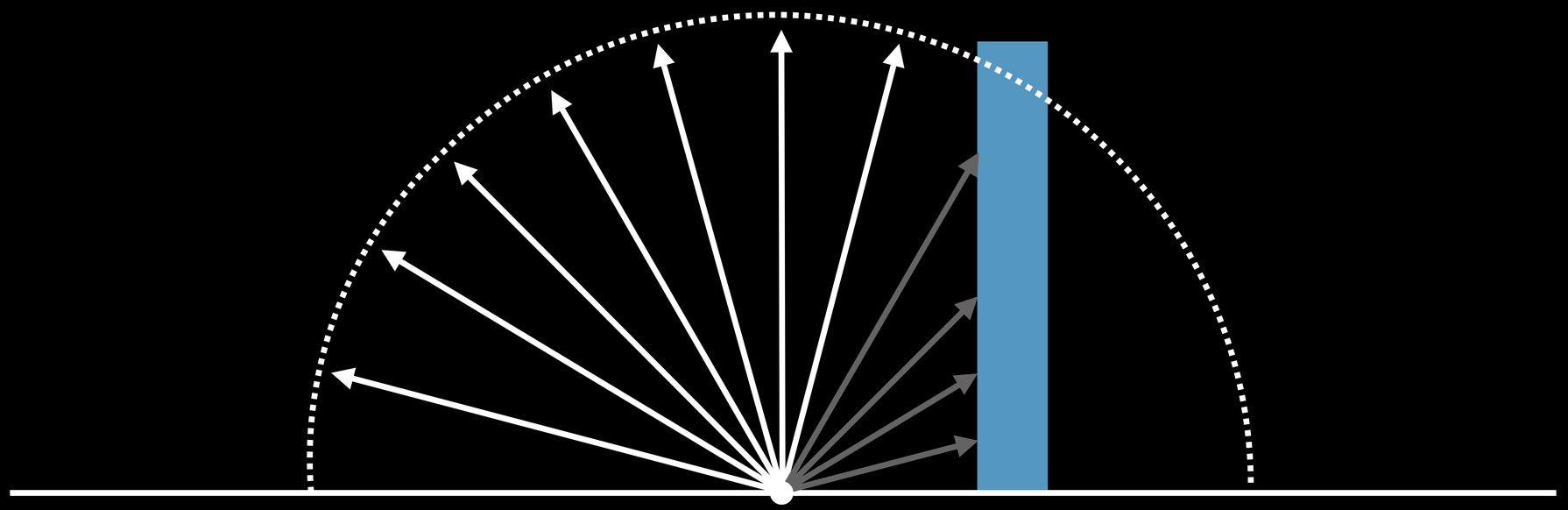
Falloff based on angle and intersection distance



Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

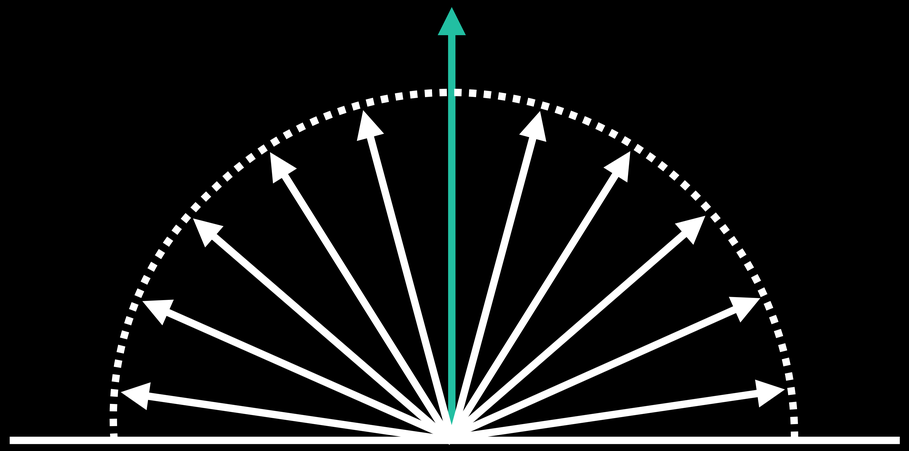
Falloff based on angle and intersection distance



Importance Sampling

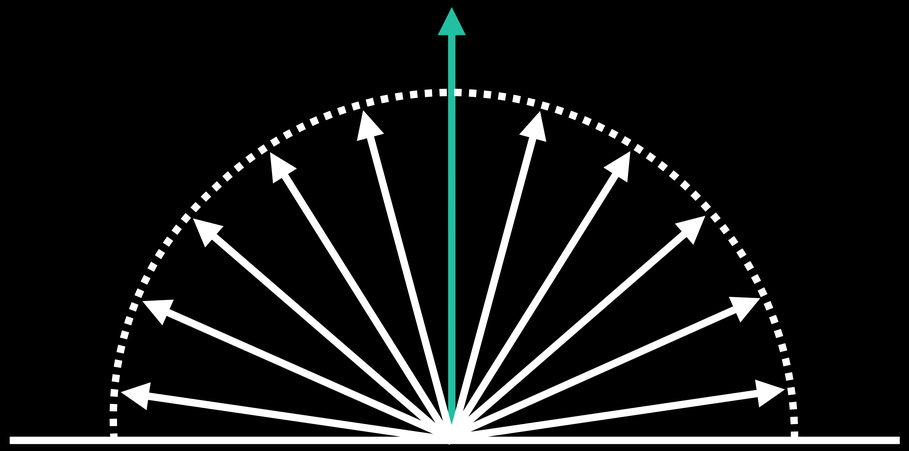
Use importance sampling to generate rays

Fewer rays for same visual quality



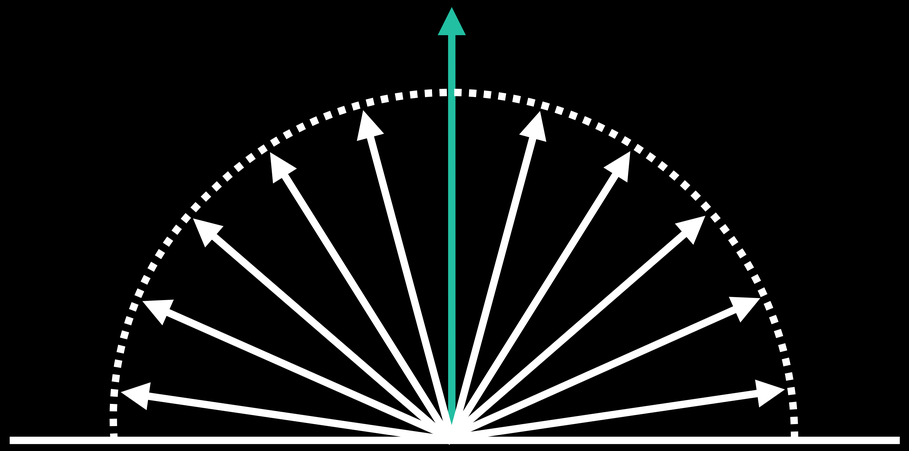
Importance Sampling

Hemisphere sampling



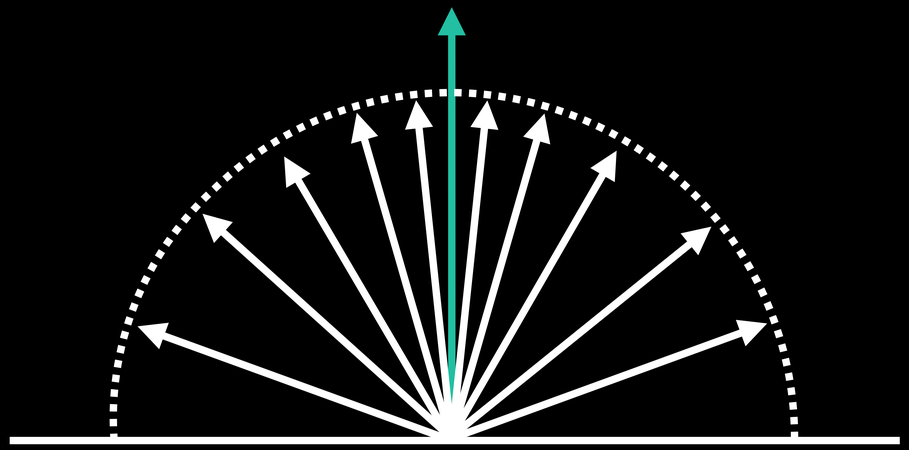
Importance Sampling

Cosine sampling



Importance Sampling

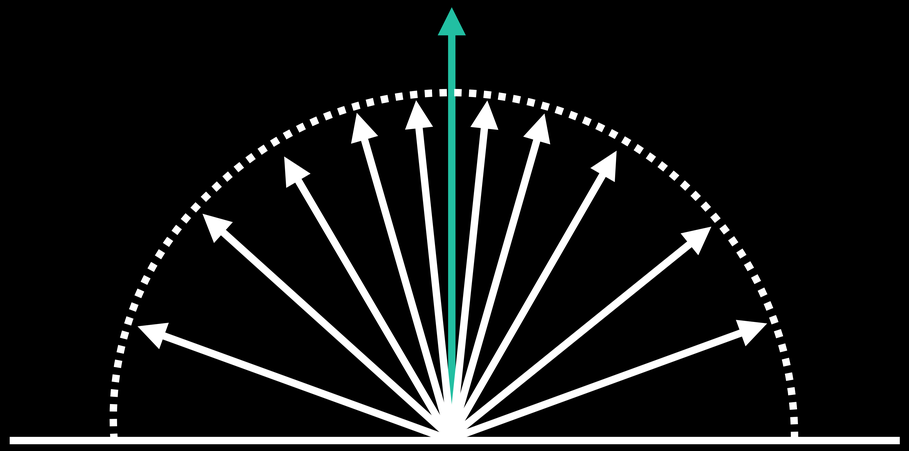
Cosine sampling



Importance Sampling

Cosine sampling

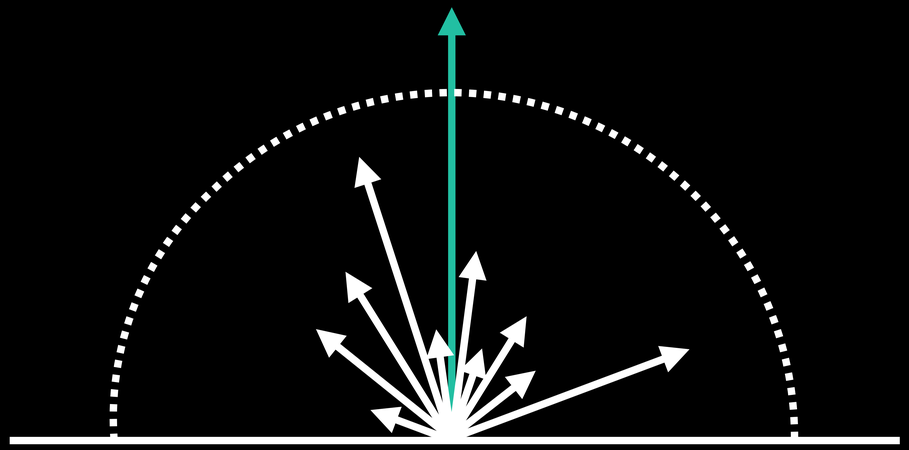
Distance sampling



Importance Sampling

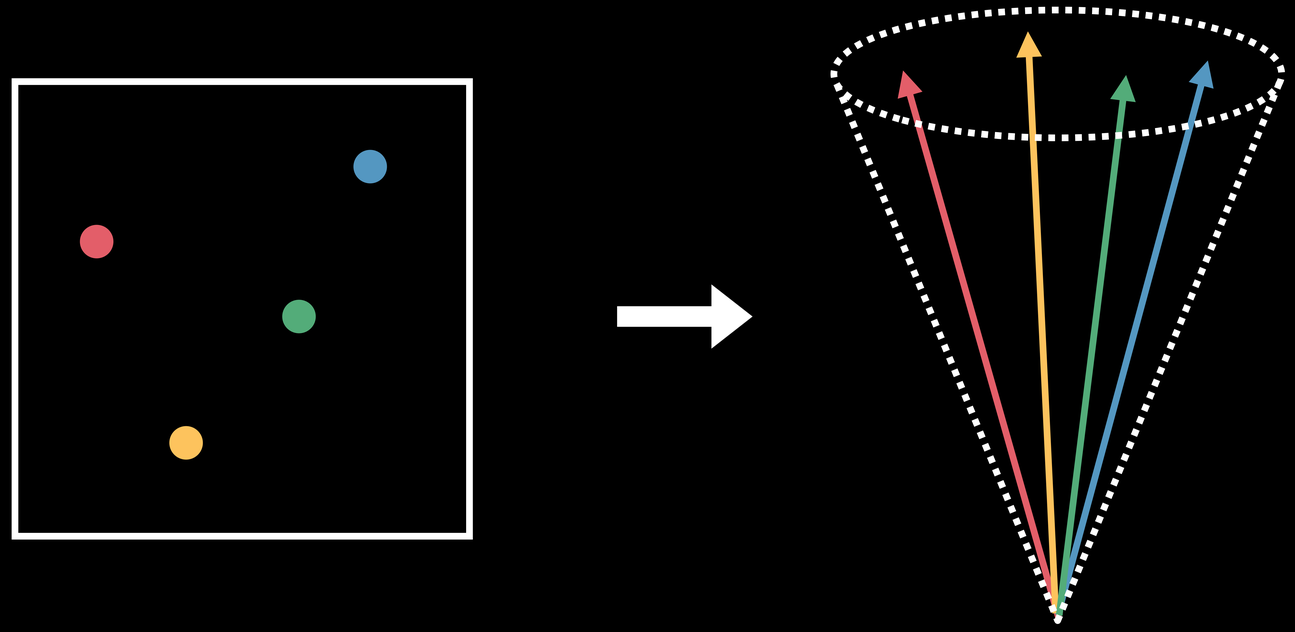
Cosine sampling

Distance sampling



Parameter Space

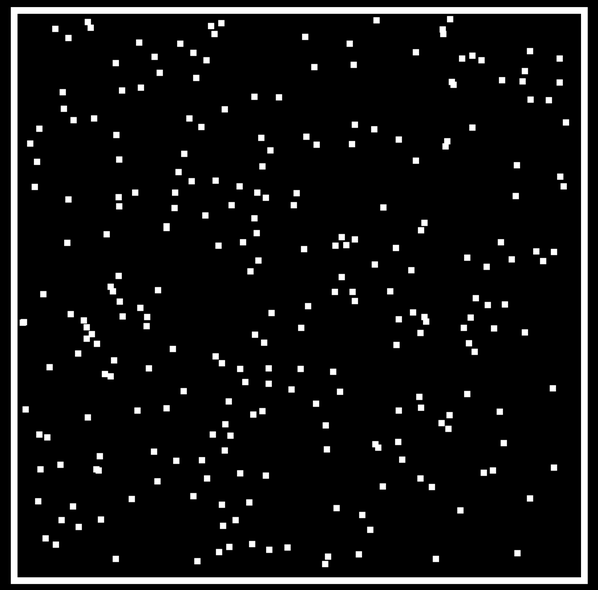
Points in 2D parameter space map to 3D ray distributions



Low Discrepancy Sequences

Distribute points evenly in parameter space

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

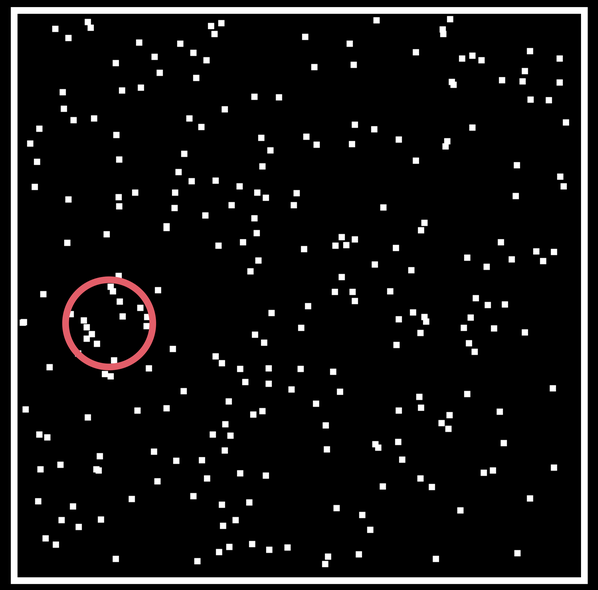


Random

Low Discrepancy Sequences

Distribute points evenly in parameter space

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

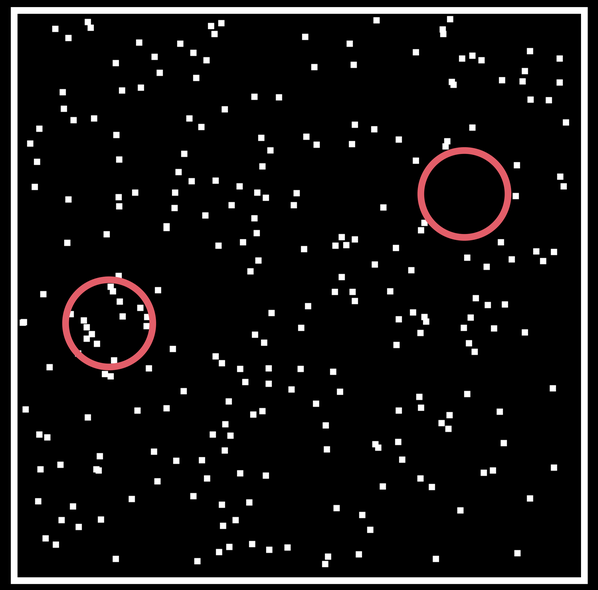


Random

Low Discrepancy Sequences

Distribute points evenly in parameter space

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

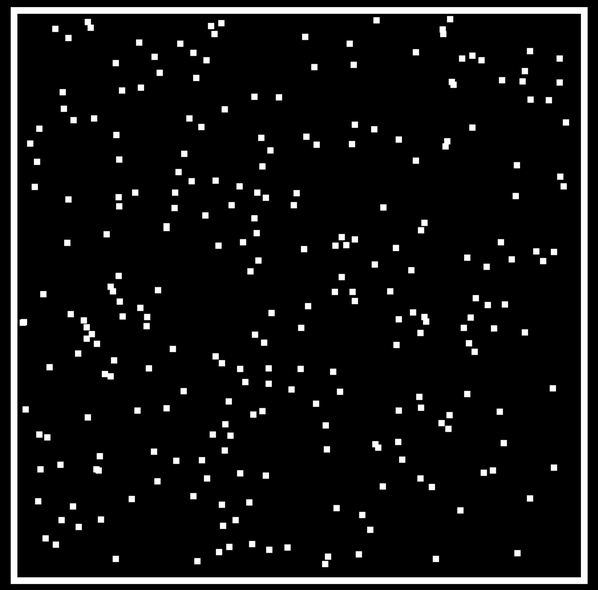


Random

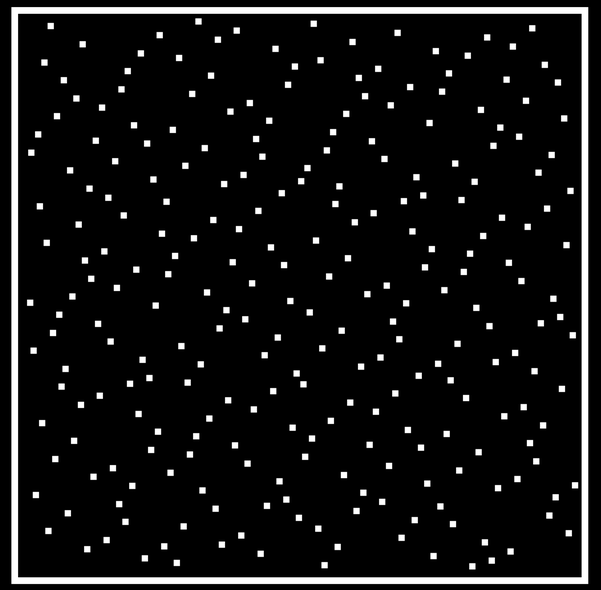
Low Discrepancy Sequences

Distribute points evenly in parameter space

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



Random

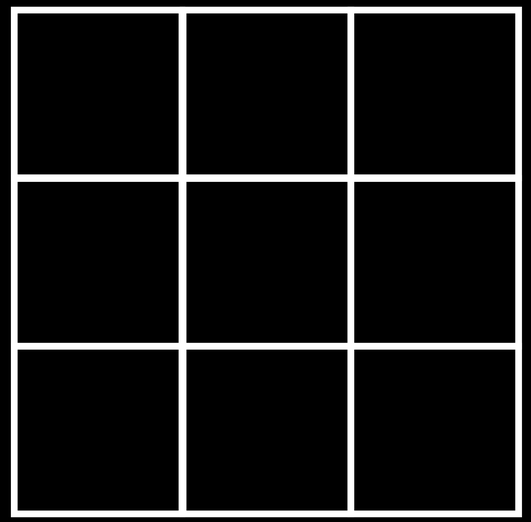


Halton (2, 3)

Pixel Decorrelation

Neighboring pixels sample different directions

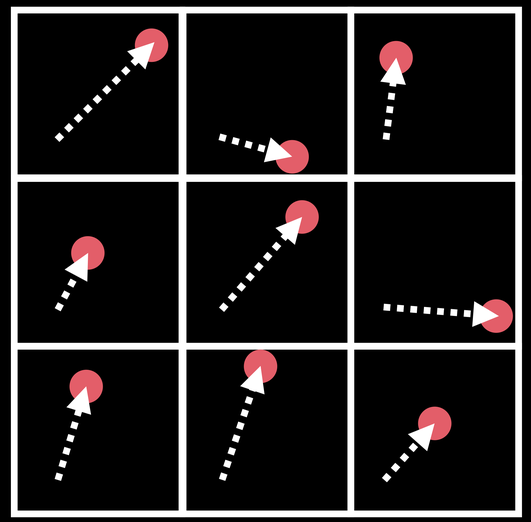
Can use same low discrepancy sample for all pixels



Pixel Decorrelation

Neighboring pixels sample different directions

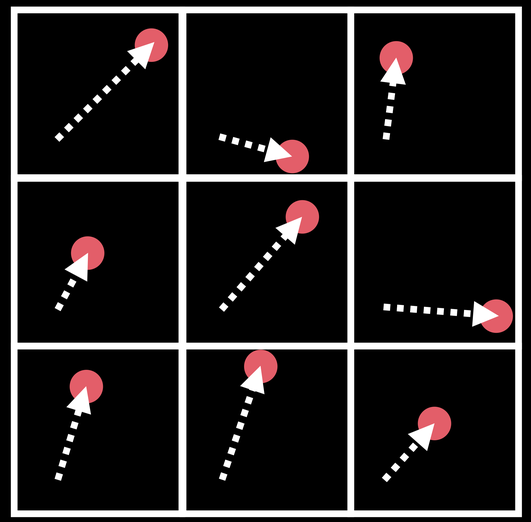
Can use same low discrepancy sample for all pixels



Pixel Decorrelation

Neighboring pixels sample different directions

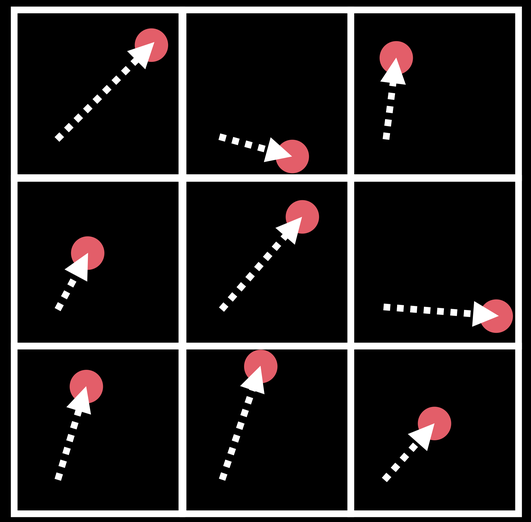
Can use same low discrepancy sample for all pixels

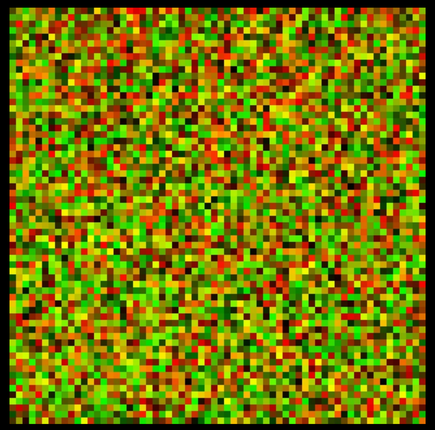


Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels



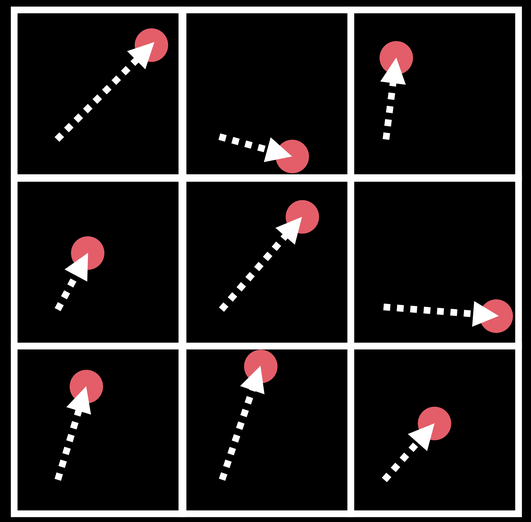


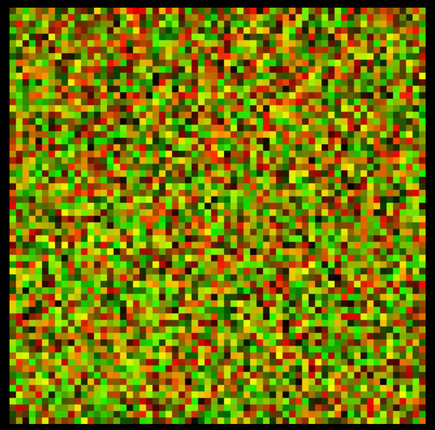
White Noise

Pixel Decorrelation

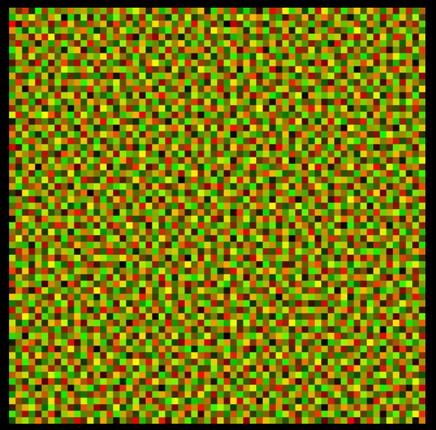
Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels



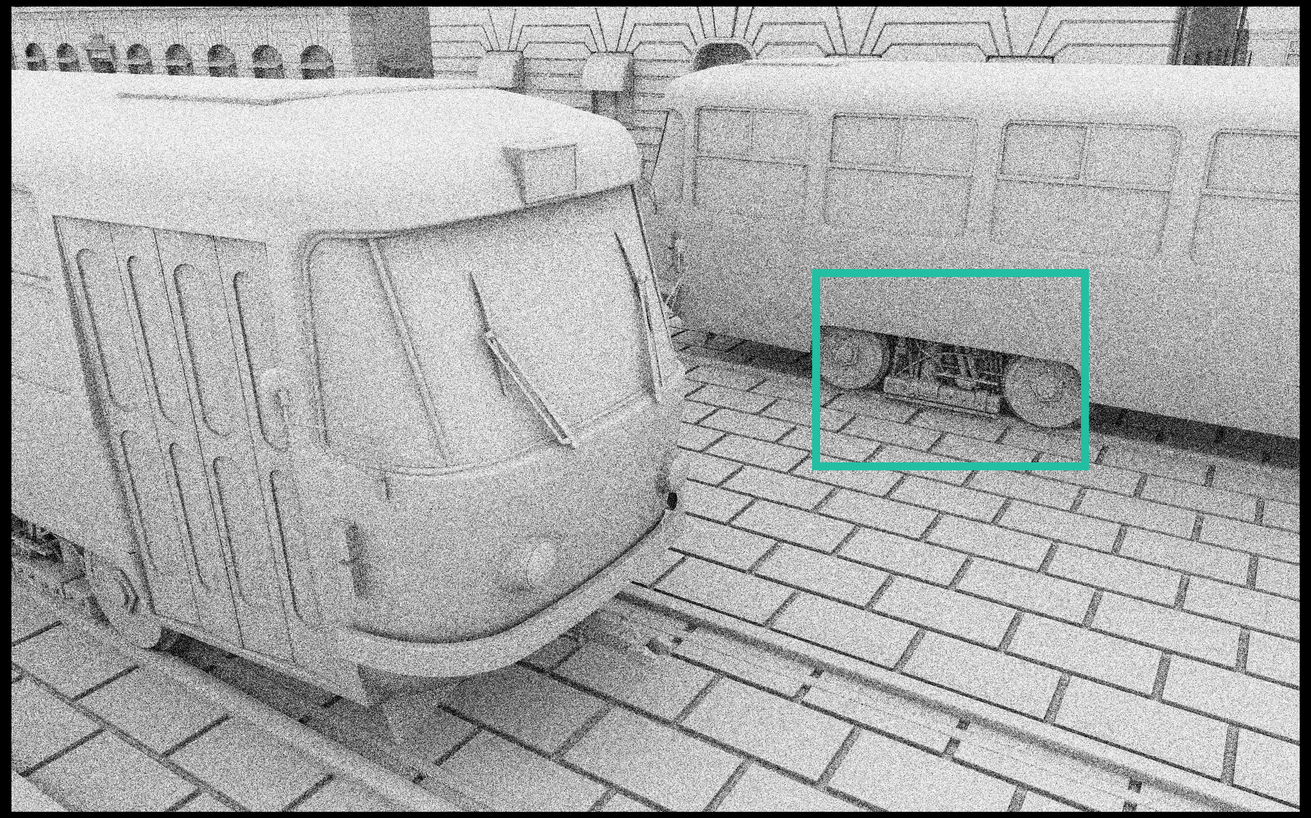


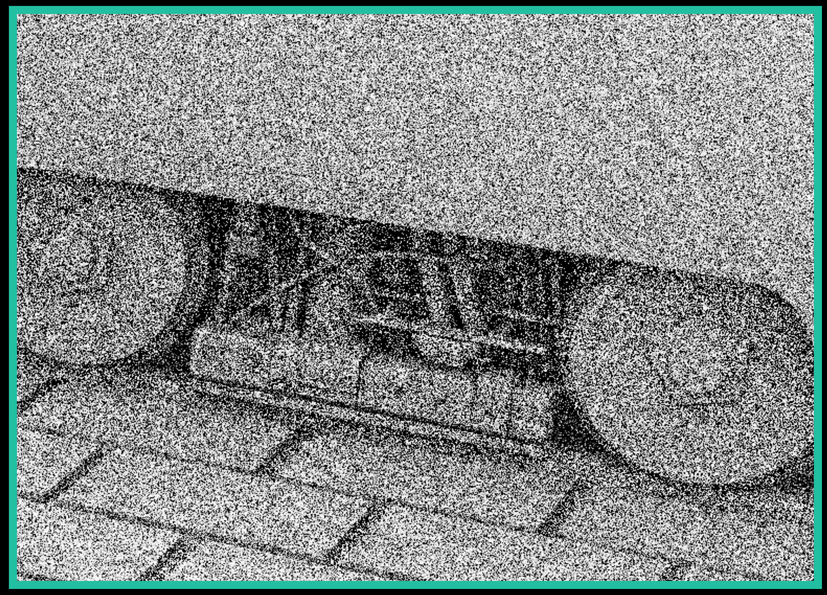
White Noise



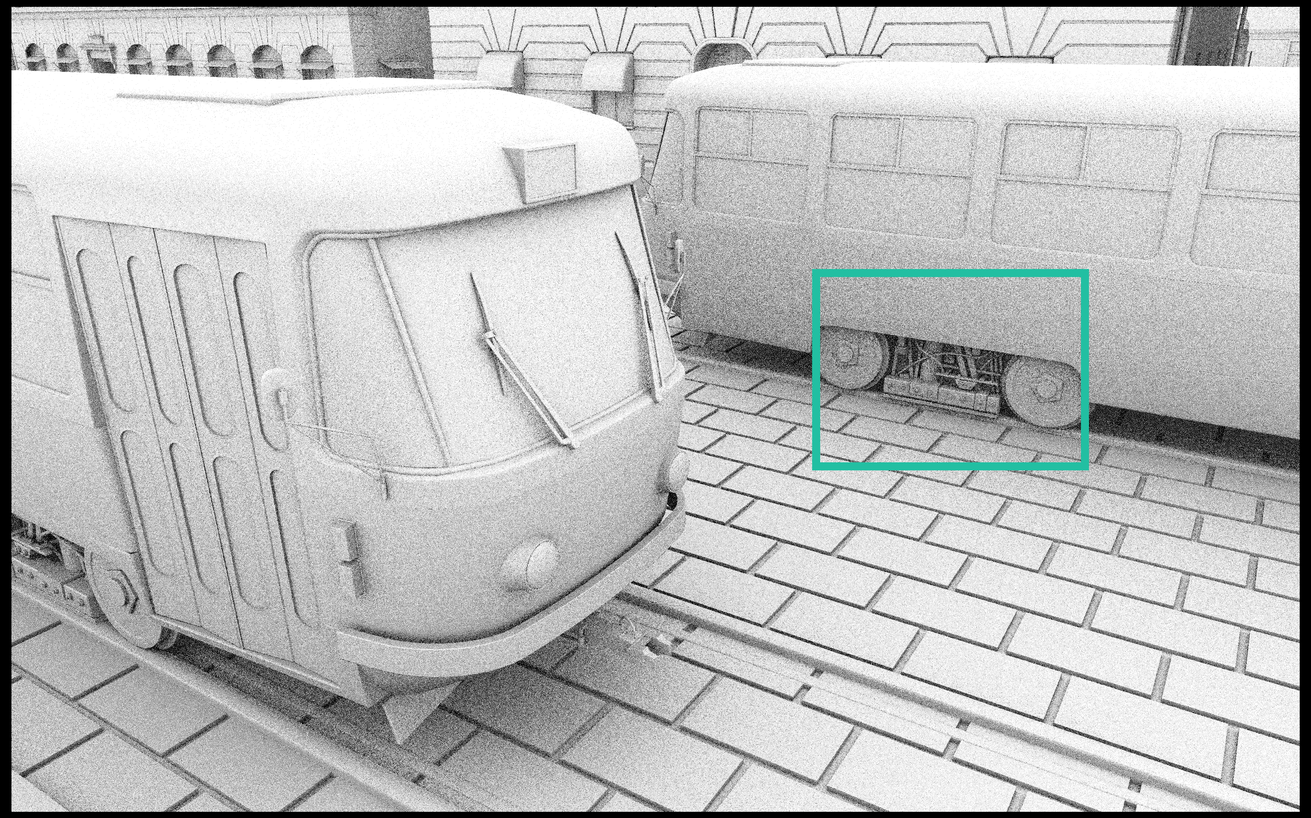
Blue Noise

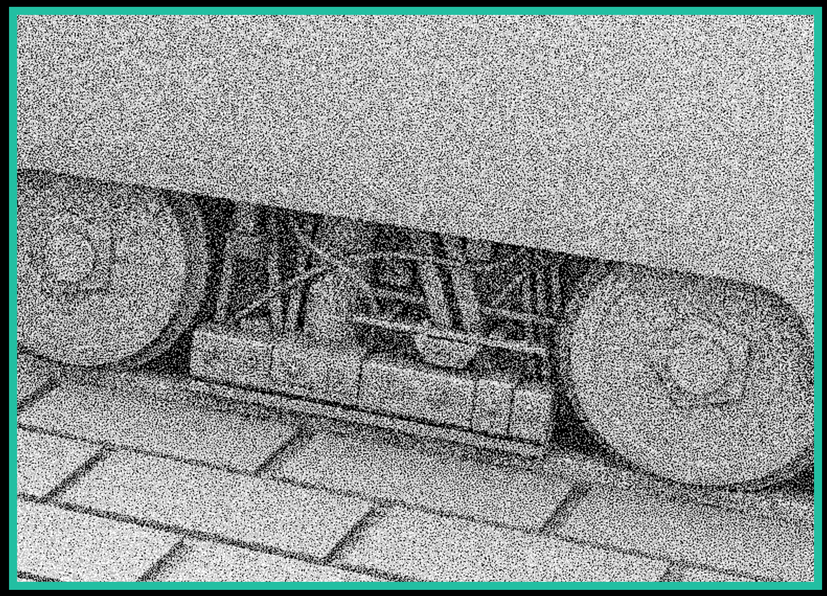
Ray Traced Result



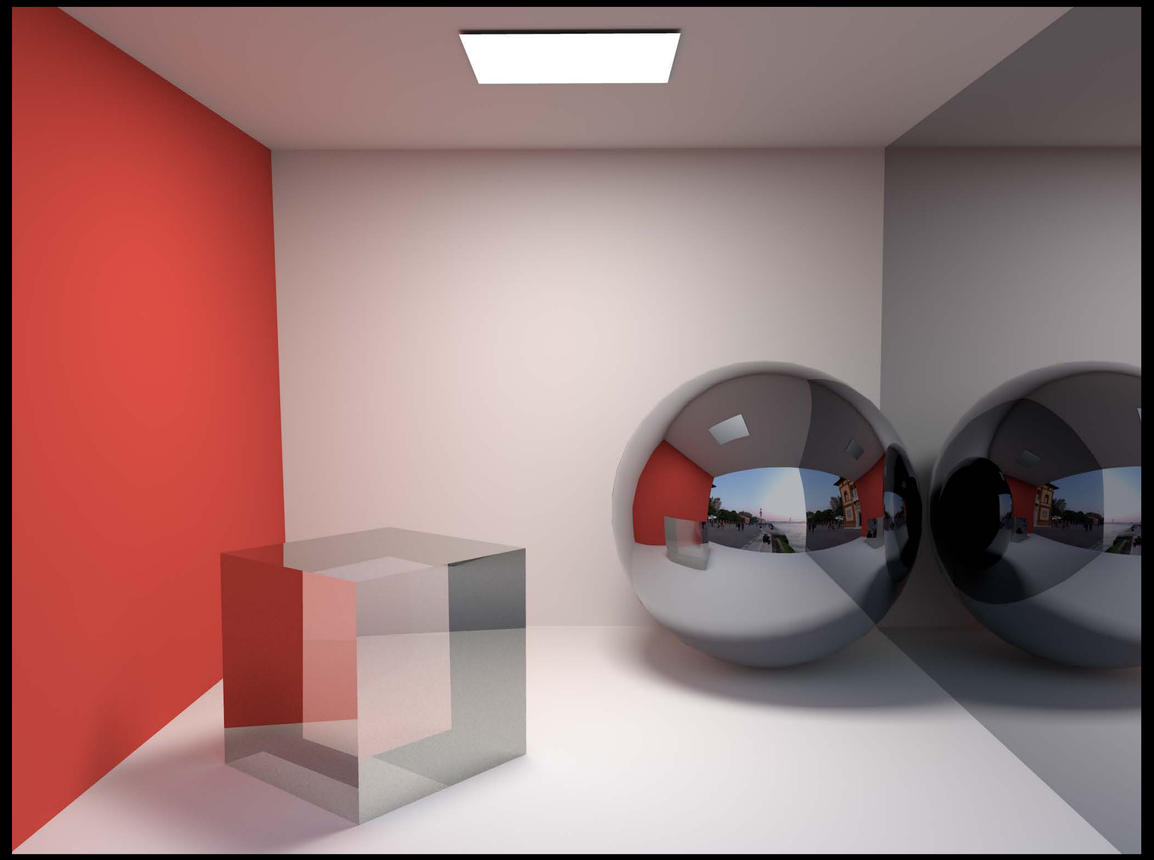


Ray Traced Result





Global Illumination



Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging

Global Illumination

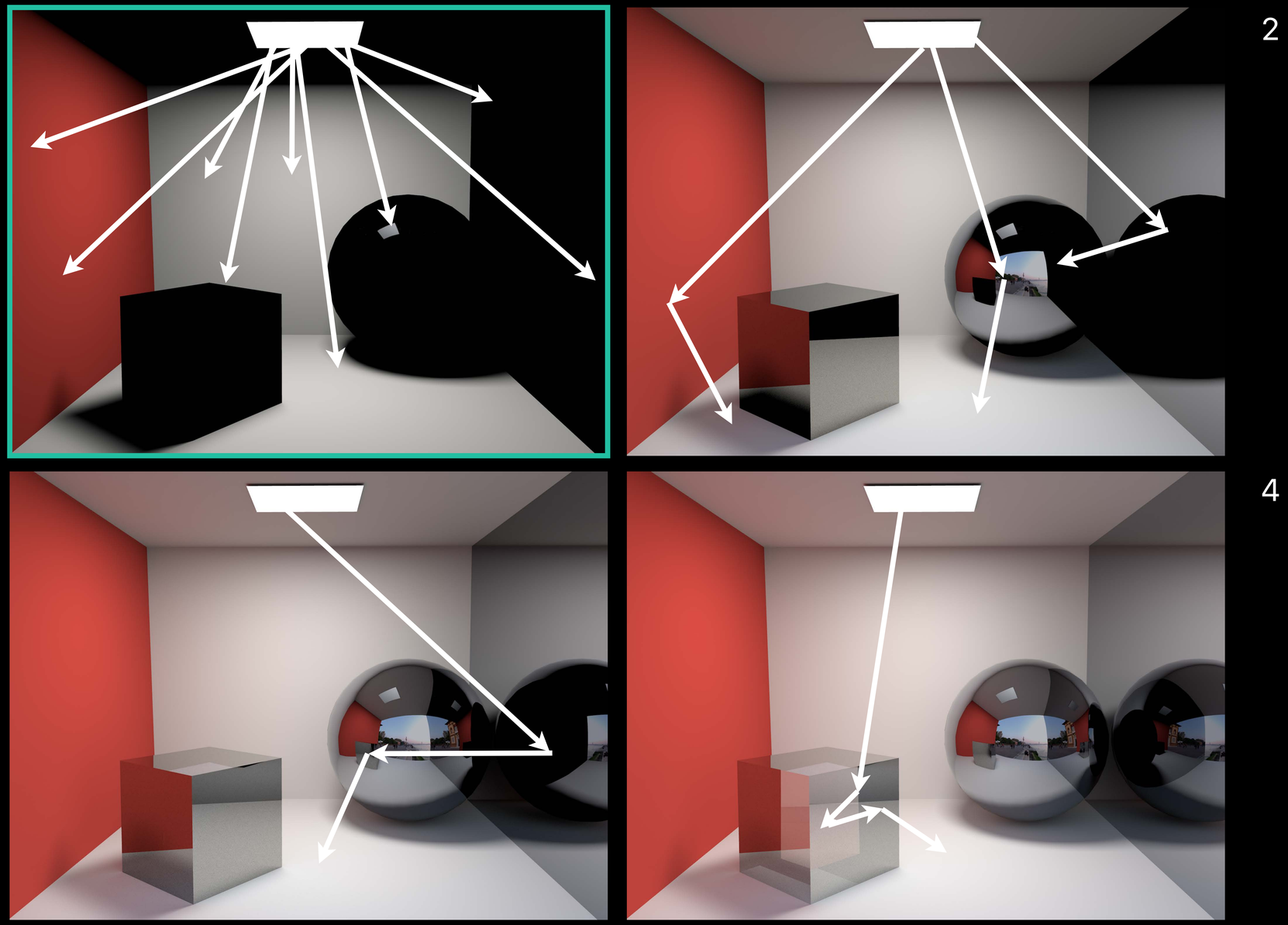
What is Global Illumination?

Memory

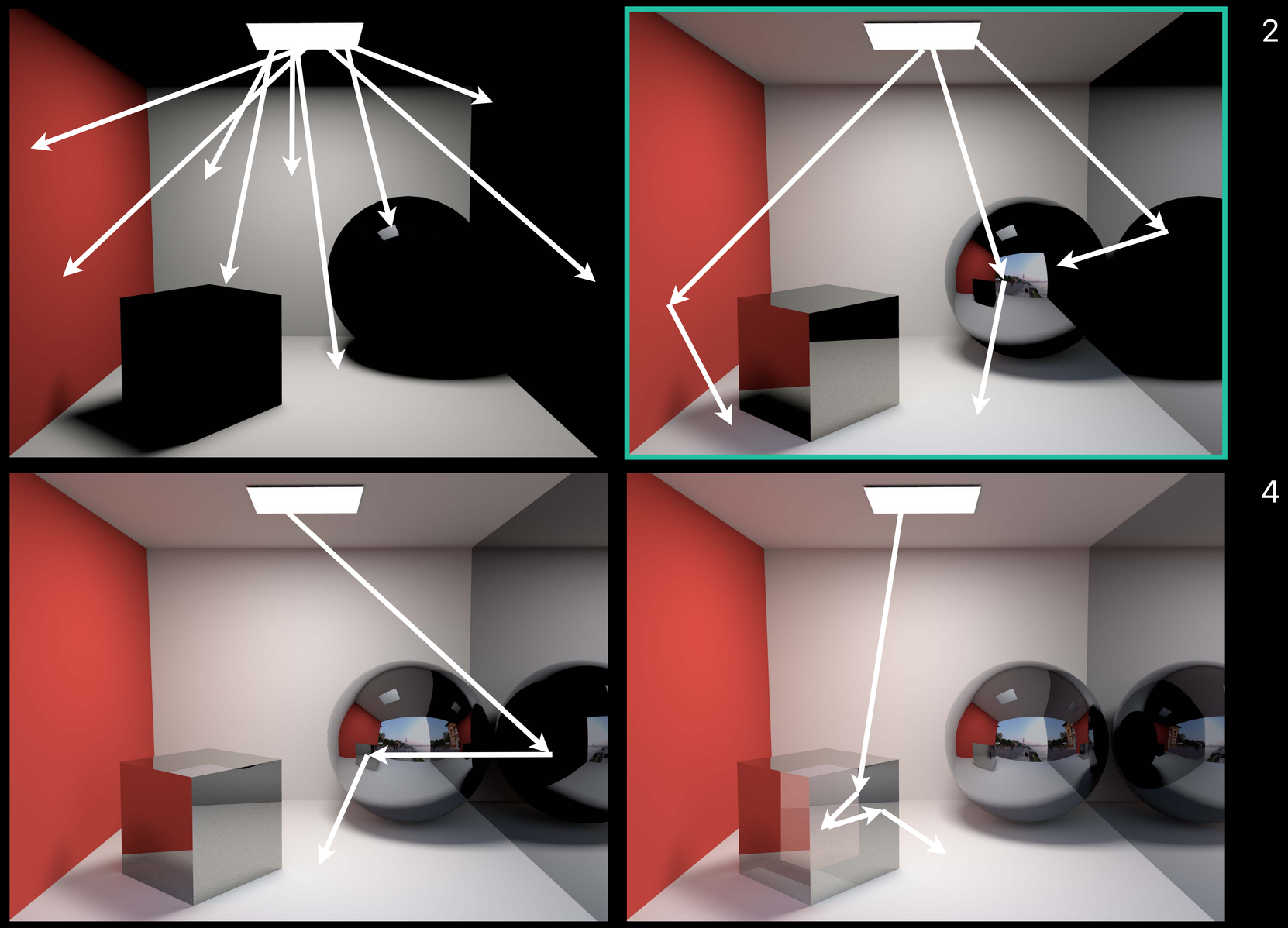
Ray Lifetime

Debugging

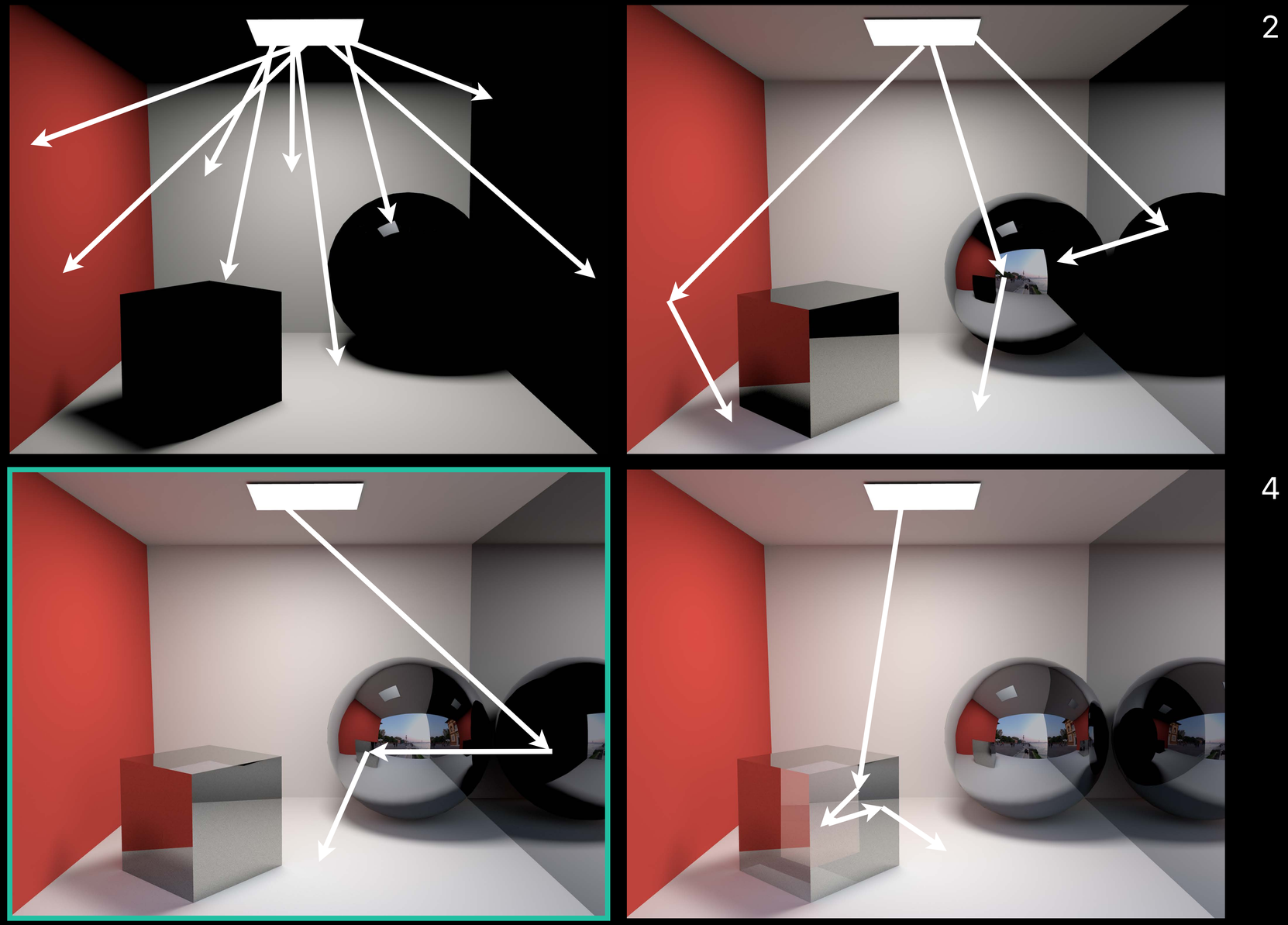
Global Illumination



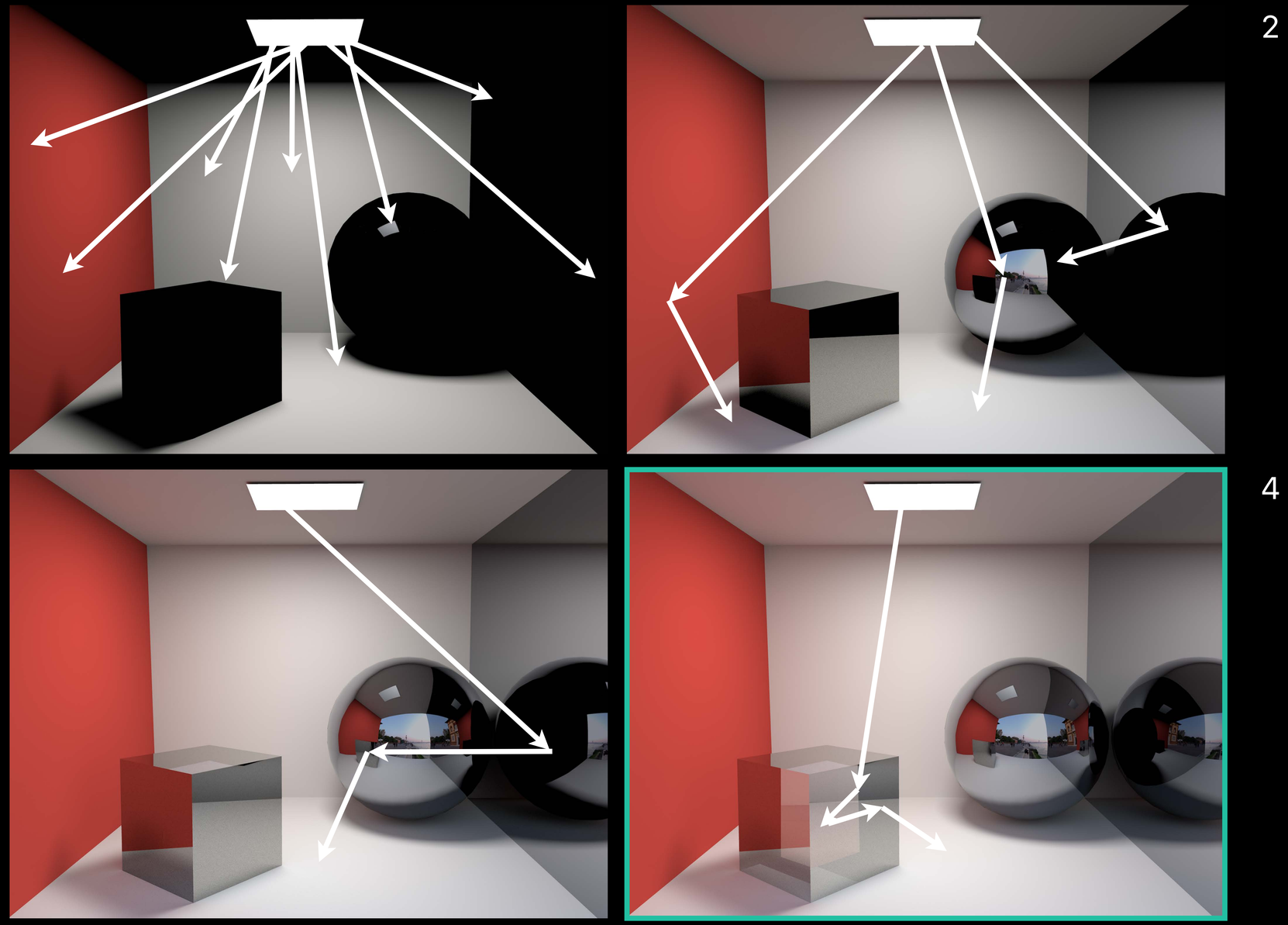
Global Illumination



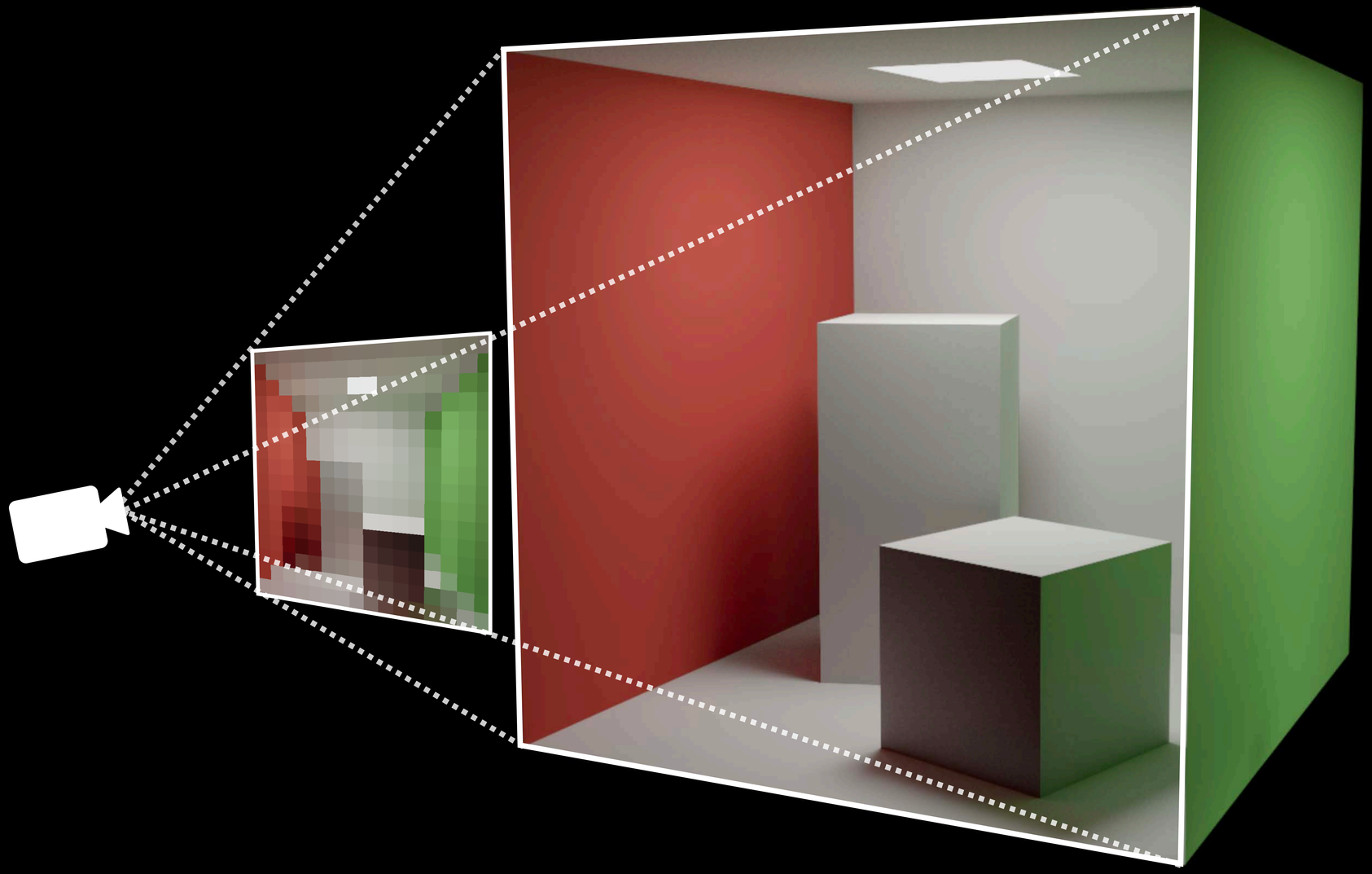
Global Illumination



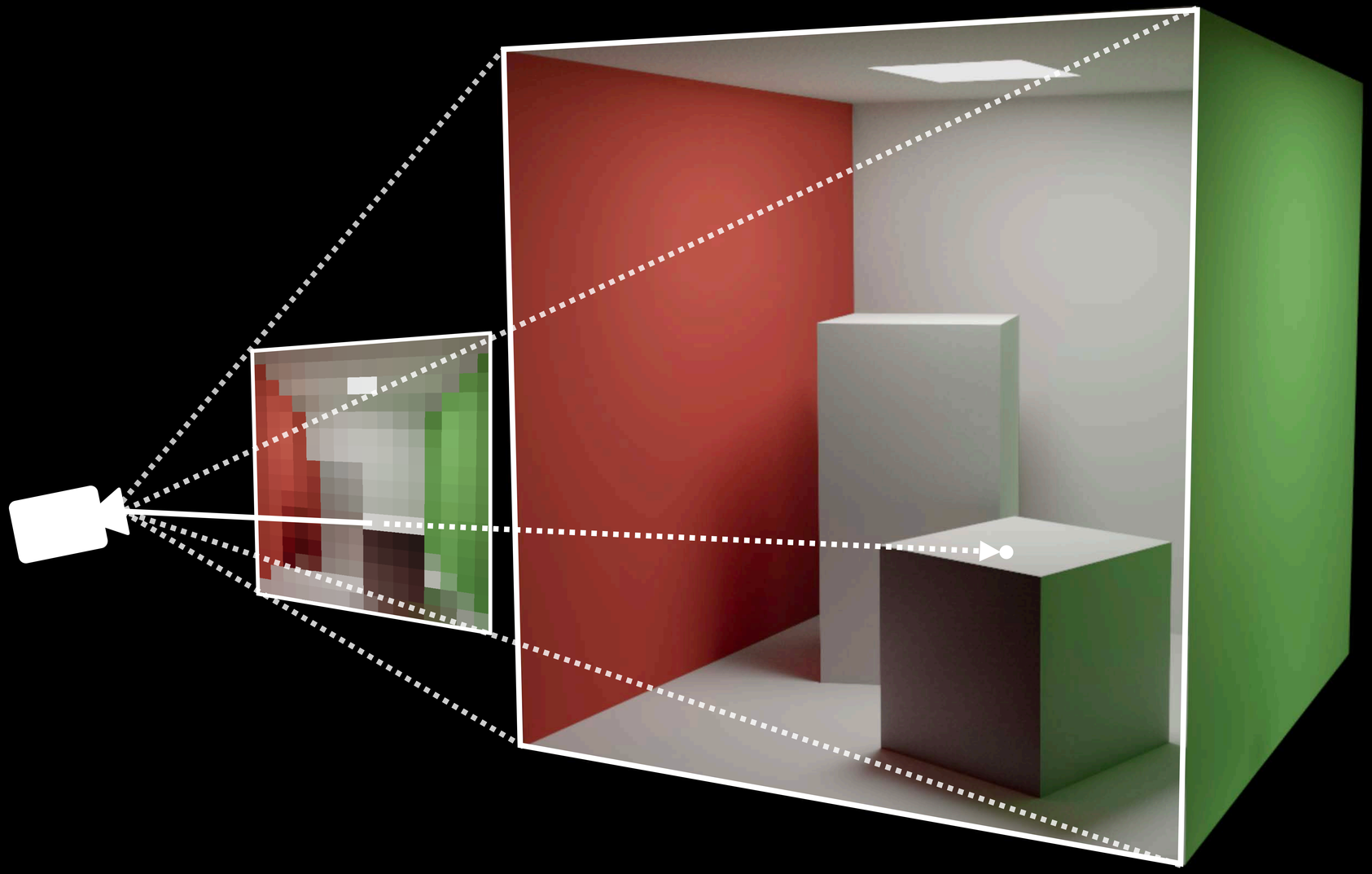
Global Illumination



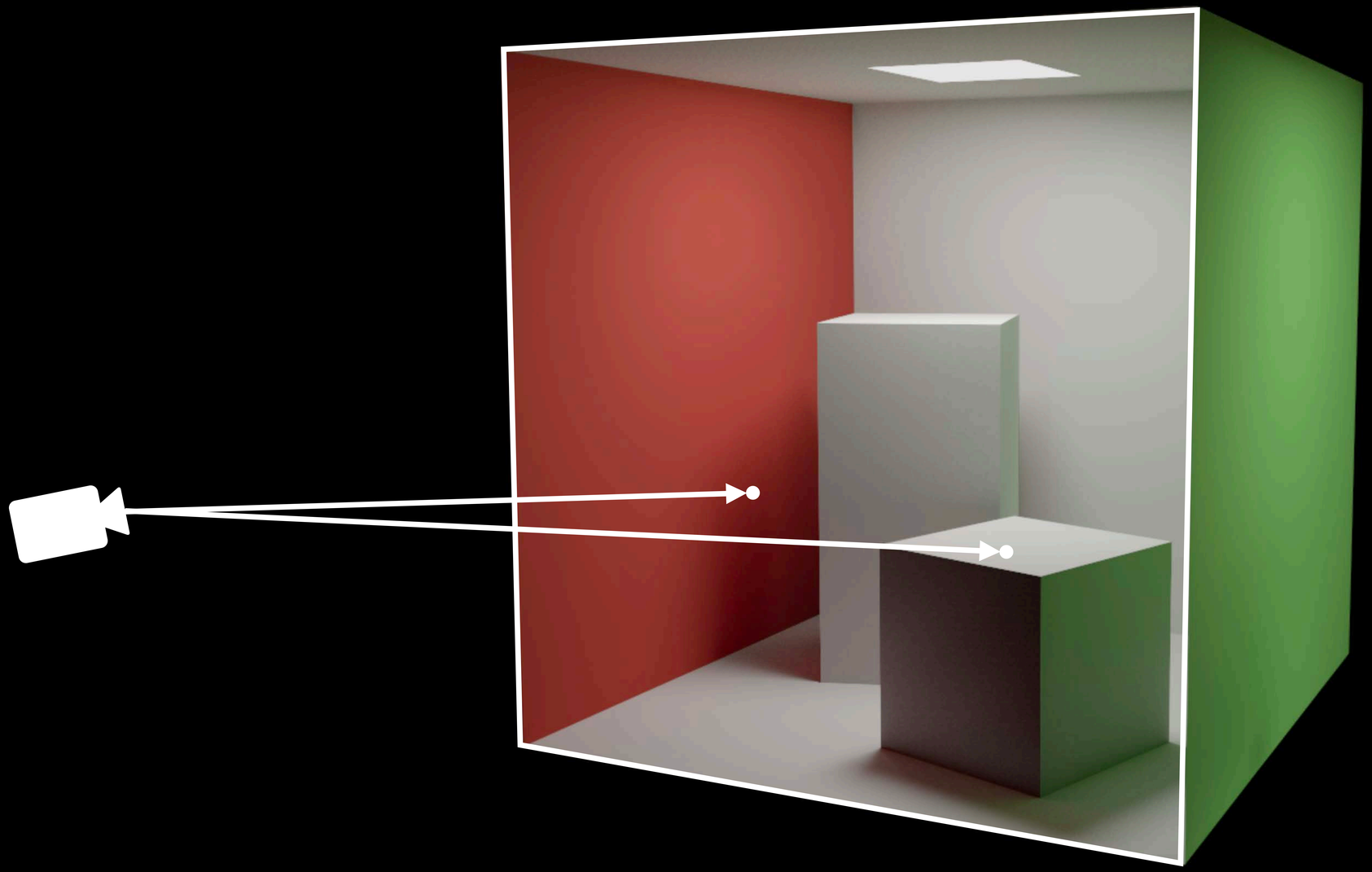
Global Illumination



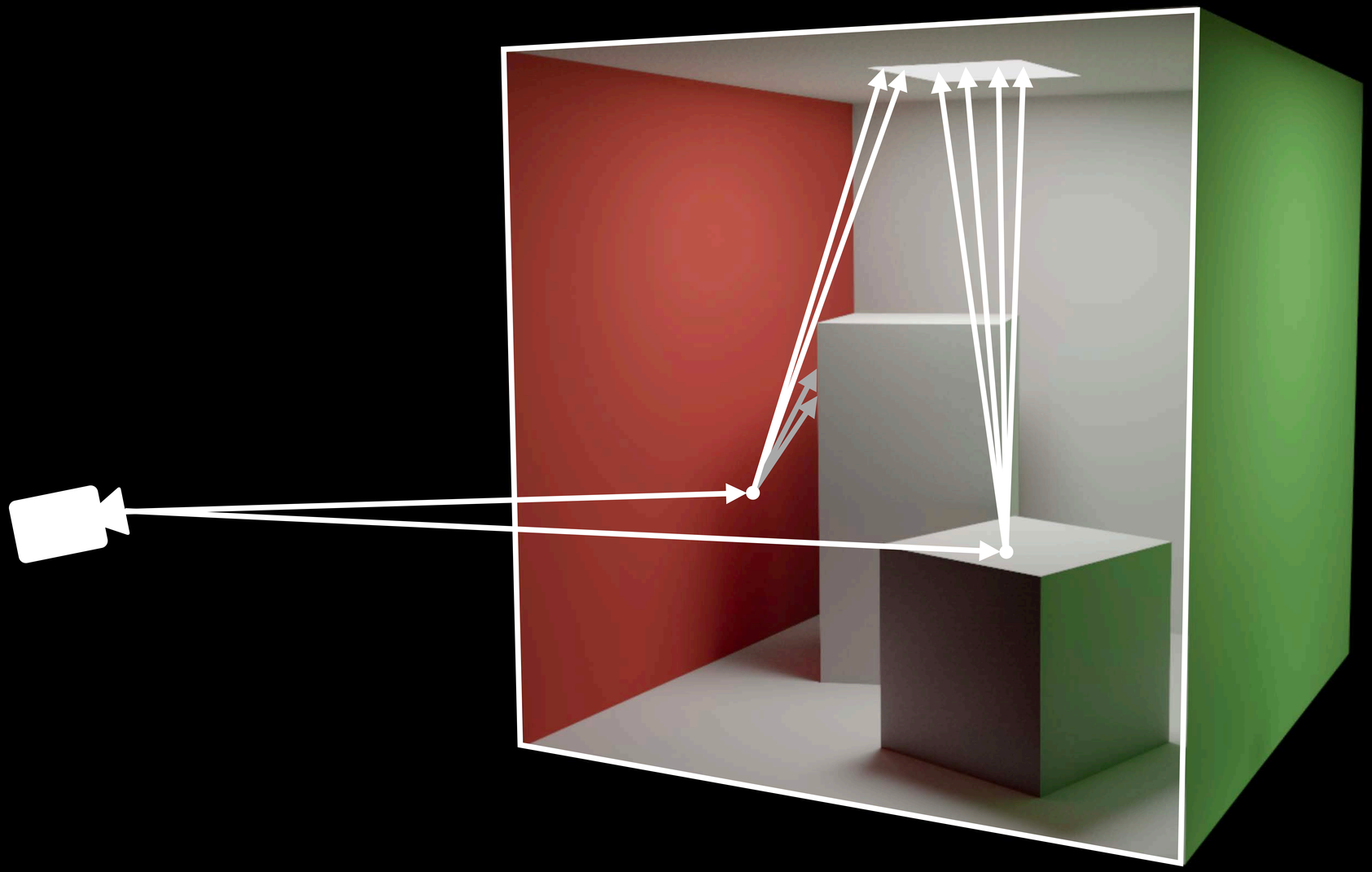
Global Illumination



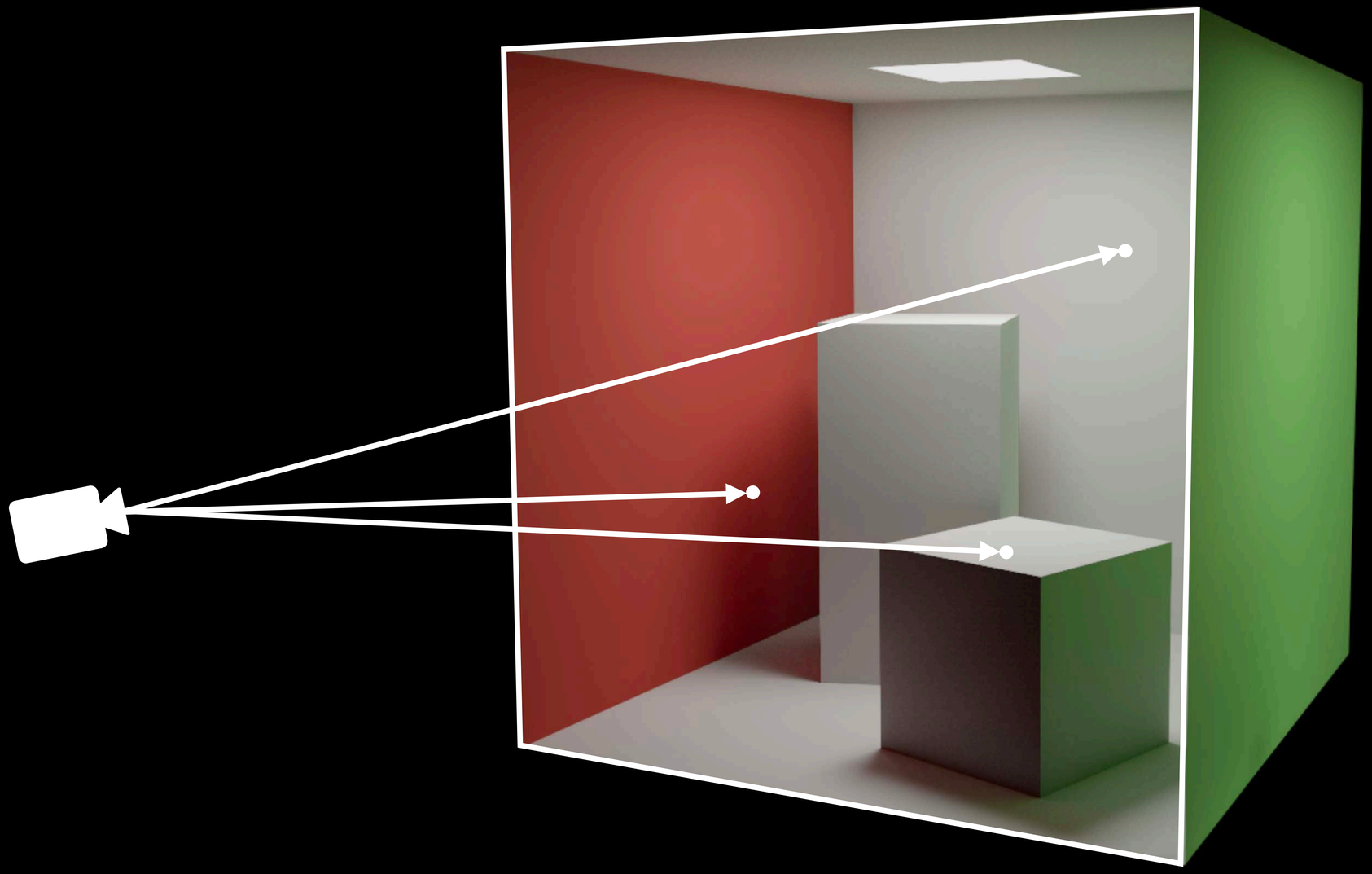
Global Illumination



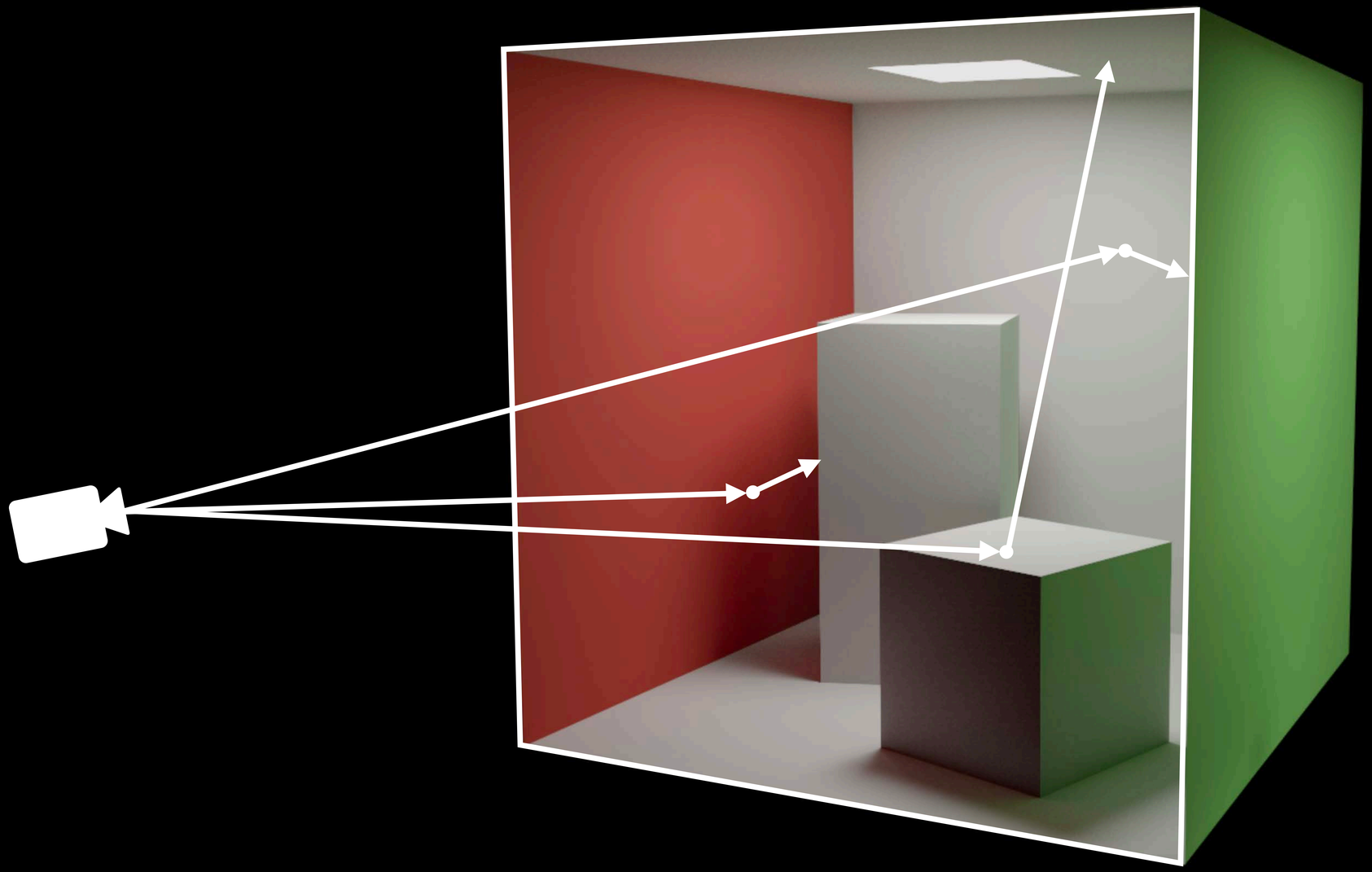
Global Illumination



Global Illumination



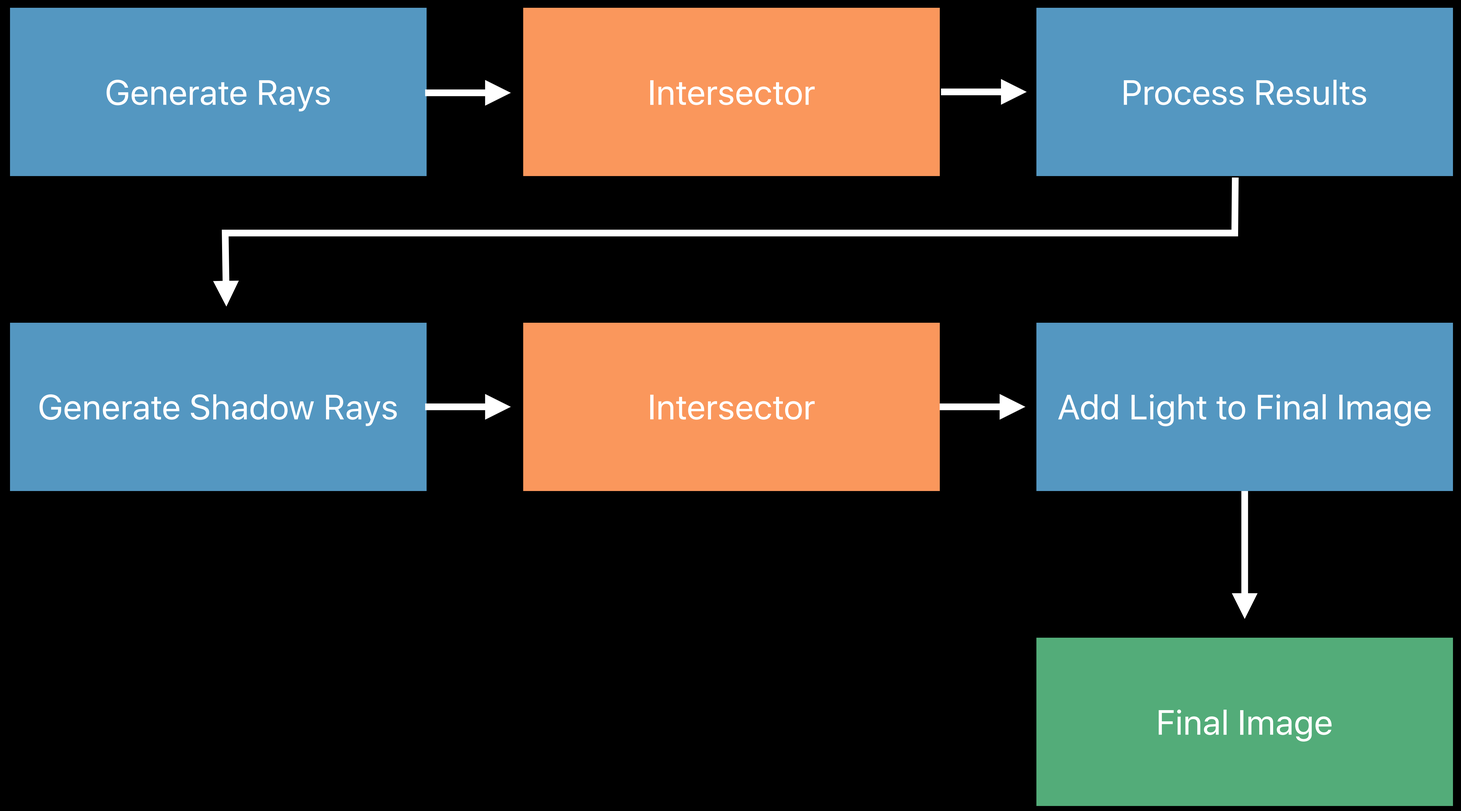
Global Illumination



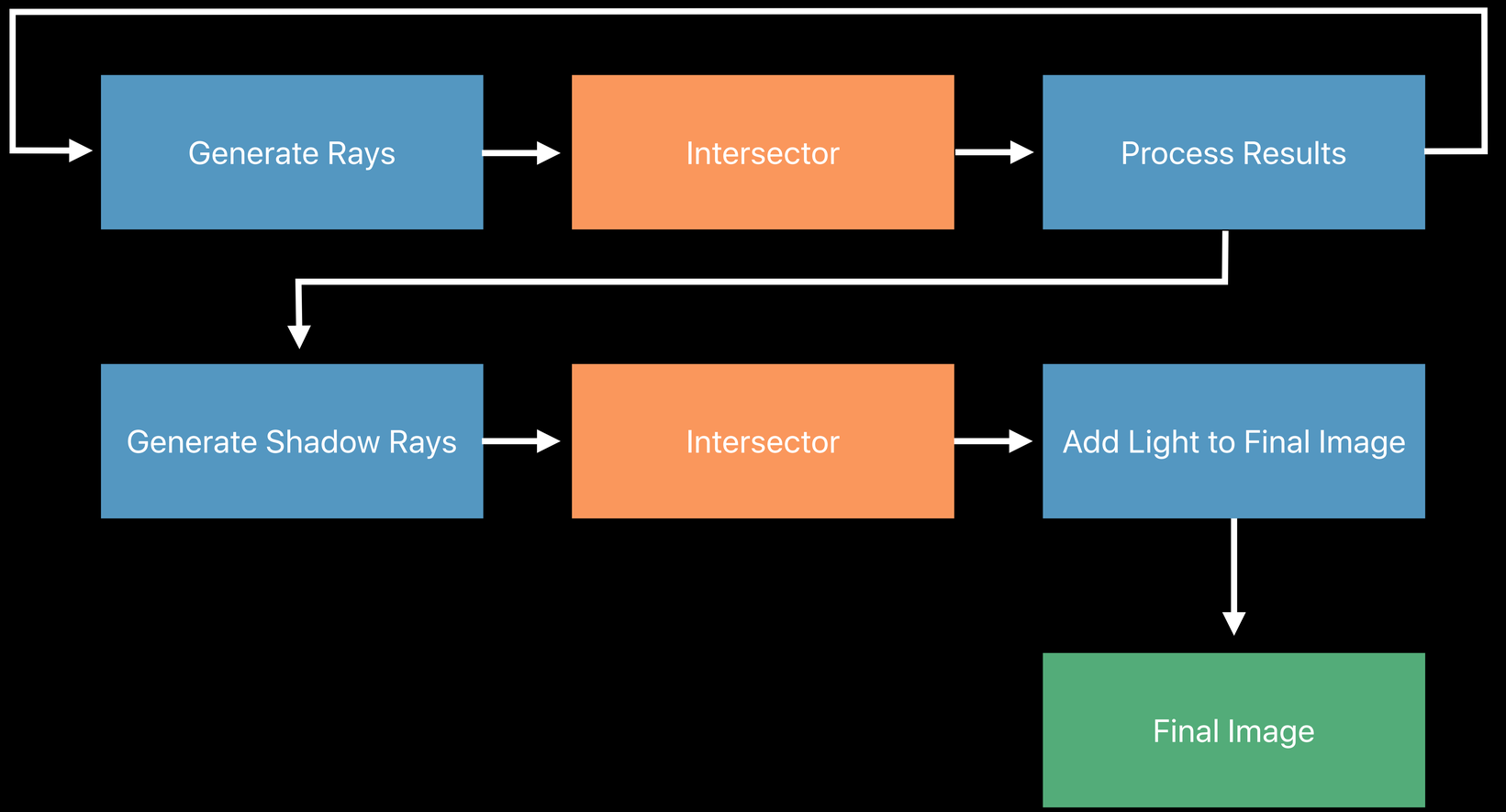
Global Illumination

Global Illumination

Global Illumination



Global Illumination



Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging

Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

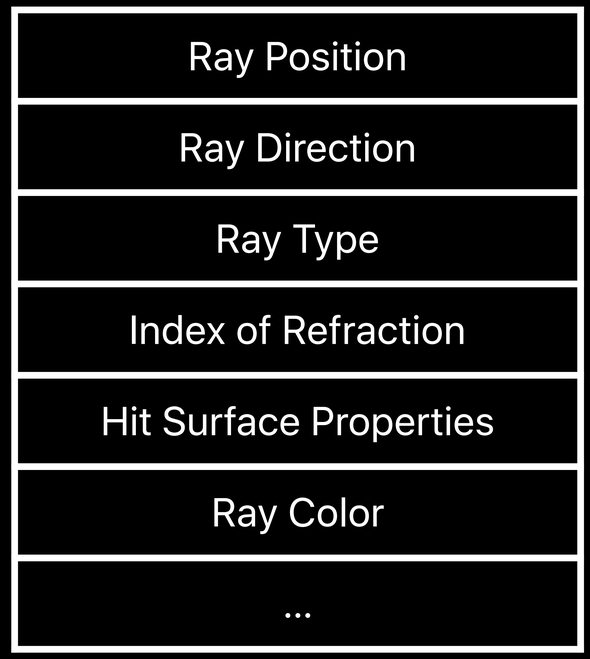
Debugging

Data Requirements

Passing data between pipeline iterations

Hybrid methods didn’t need to do this

Can still use constant indexing for all the buffers



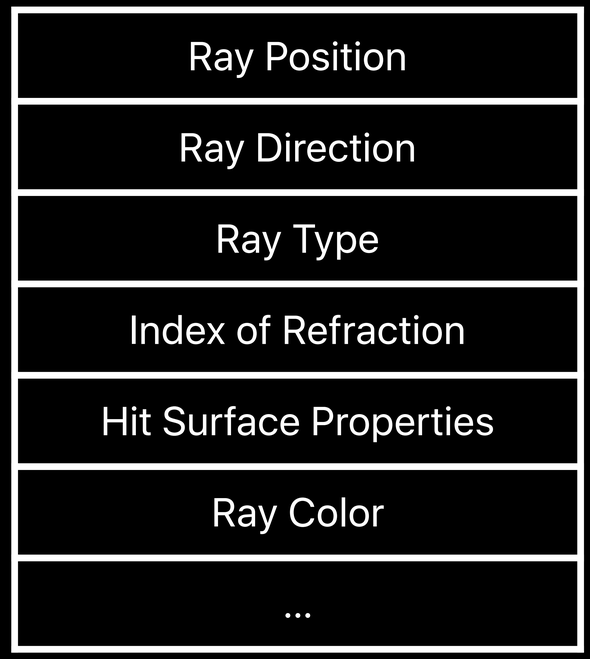
Memory Usage

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

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

```
packed_half3 *origin;
packed_half3 *direction;
```

Occupancy

Reduce register pressure

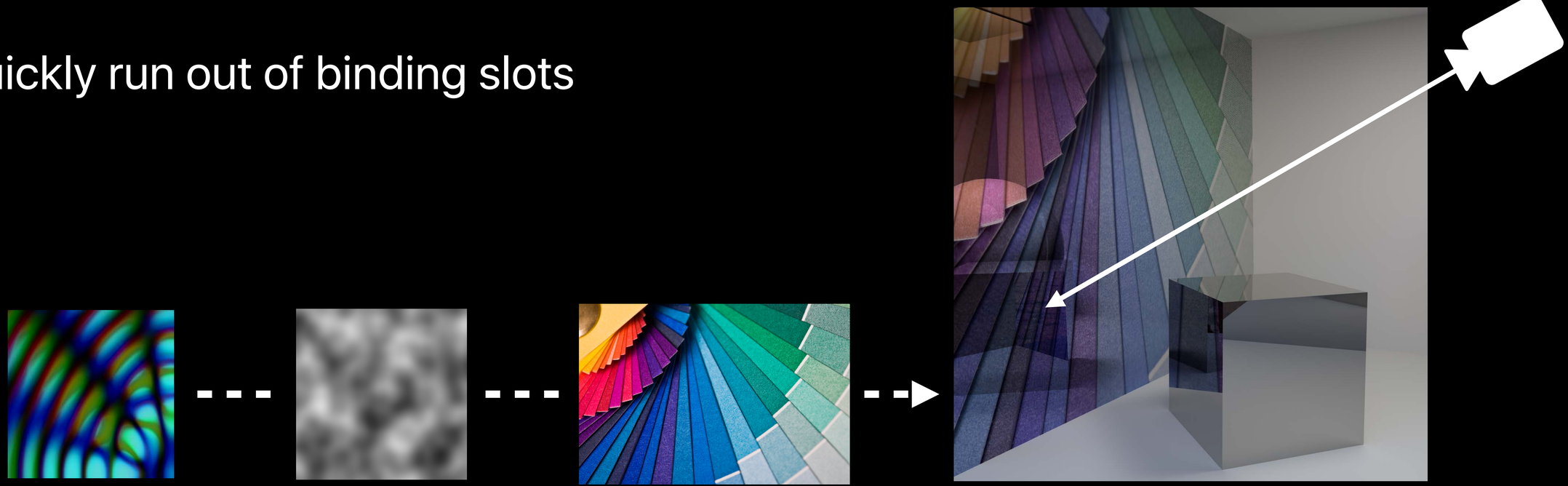
- Track simultaneously live variables
- Don’t hold onto structs
- Be careful with loop counters, function calls

Textures

Can’t predict what surface a ray will intersect

Sponza scene has 76 textures

Quickly run out of binding slots



Argument Buffers

```
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

```
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

```
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

```
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

```
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

```
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

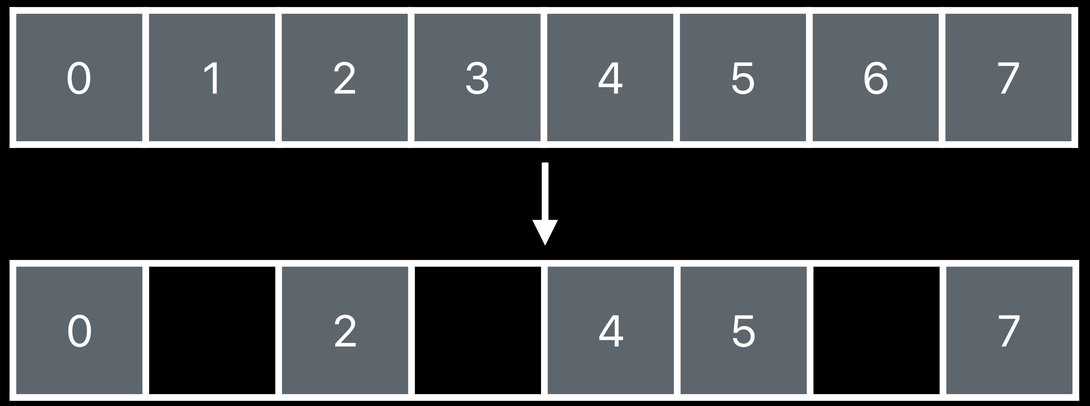
First iteration: 100% of rays active

0	1	2	3	4	6	6	7

Ray Buffer

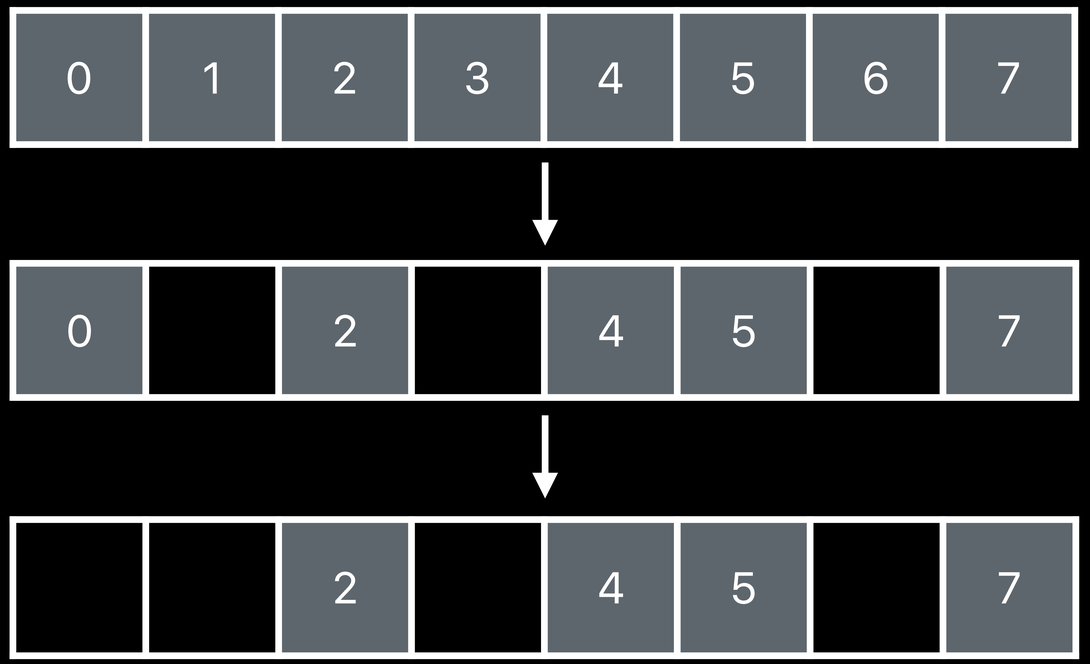
Inactive Rays

Second iteration: 57% of rays active



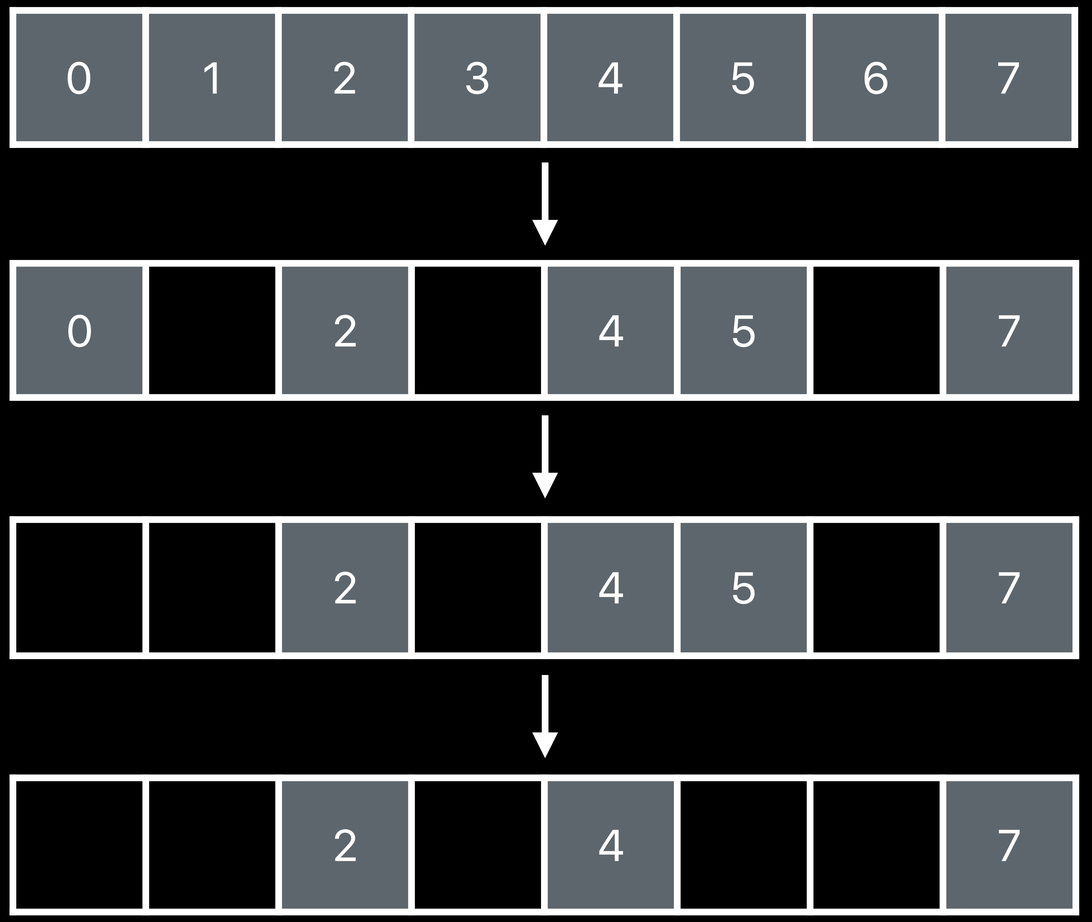
Inactive Rays

Third iteration: 43% of rays active



Inactive Rays

Fourth iteration: 32% of rays active



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

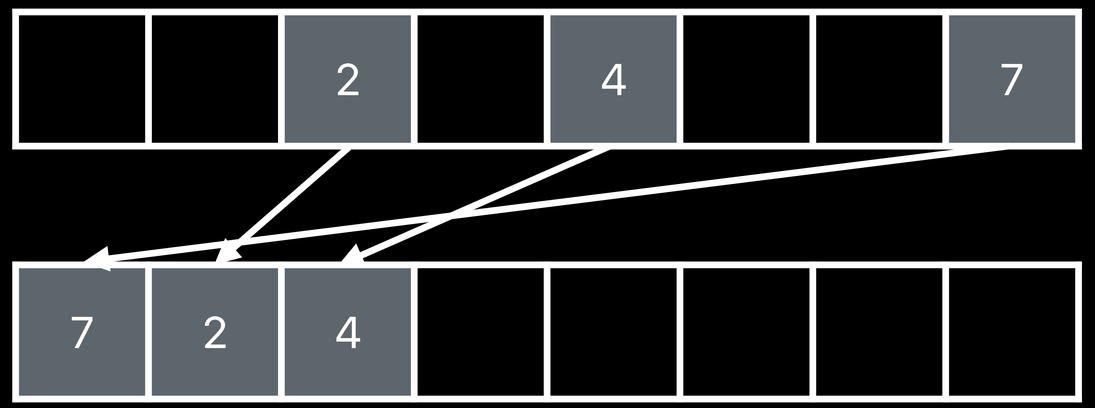
Also works for shadow rays

Ray Compaction

Only add active rays to the next ray buffer

Threadgroups are fully utilized

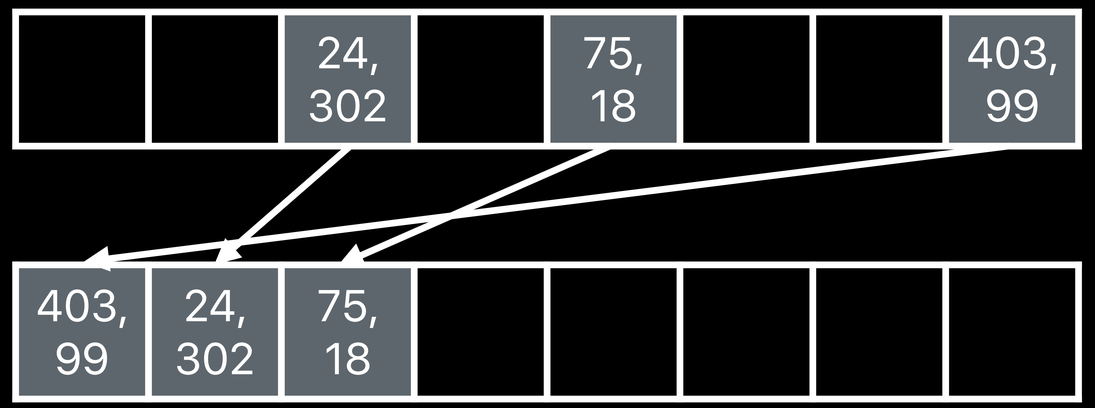
Also works for shadow rays



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:

```
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:

```
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:

```
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:

```
// 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:

```
// 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:

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

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

Ray Compaction

Indirect ray intersection:

```
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:

```
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

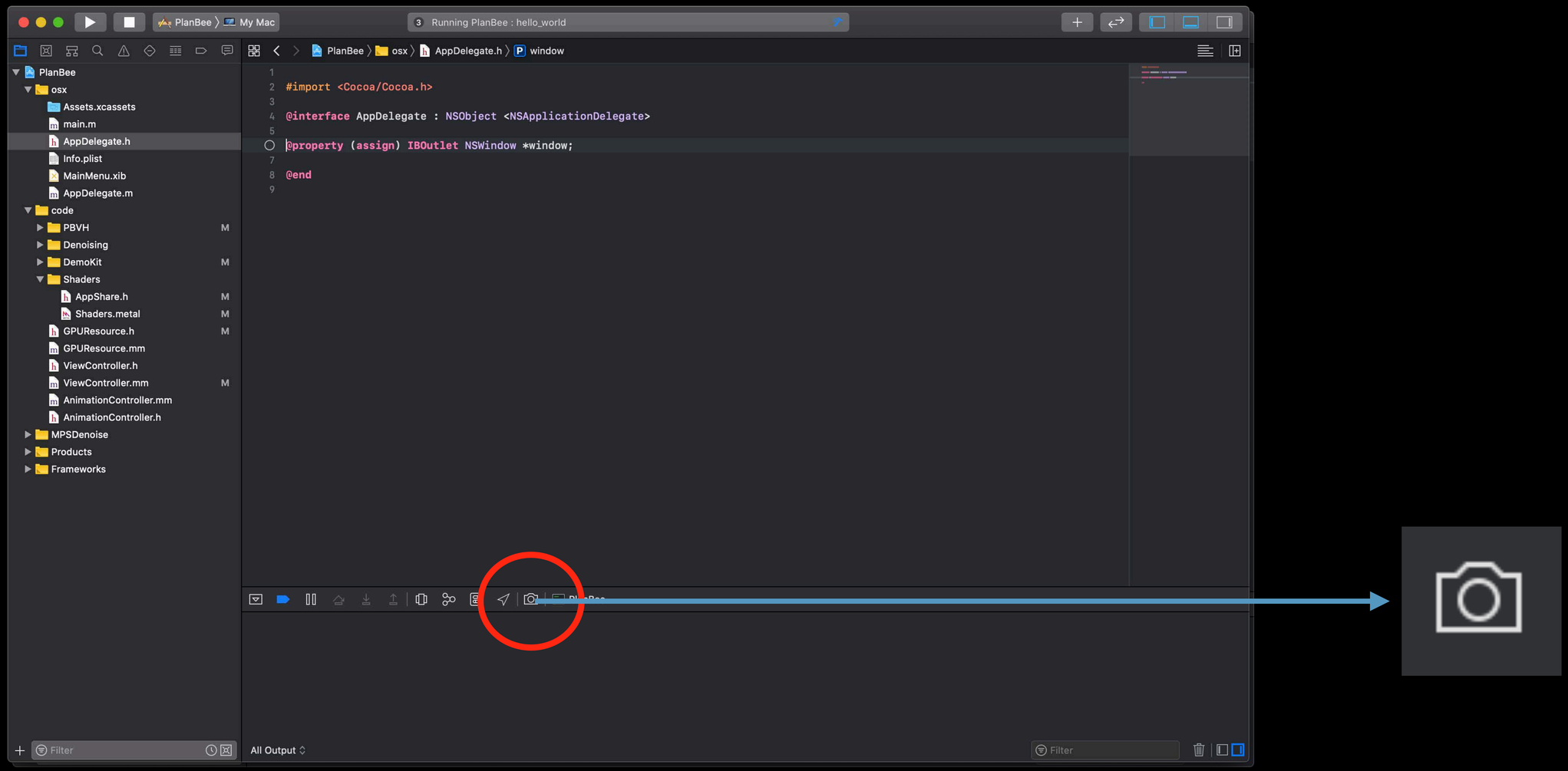
Frame capture

Debugging Image Corruption

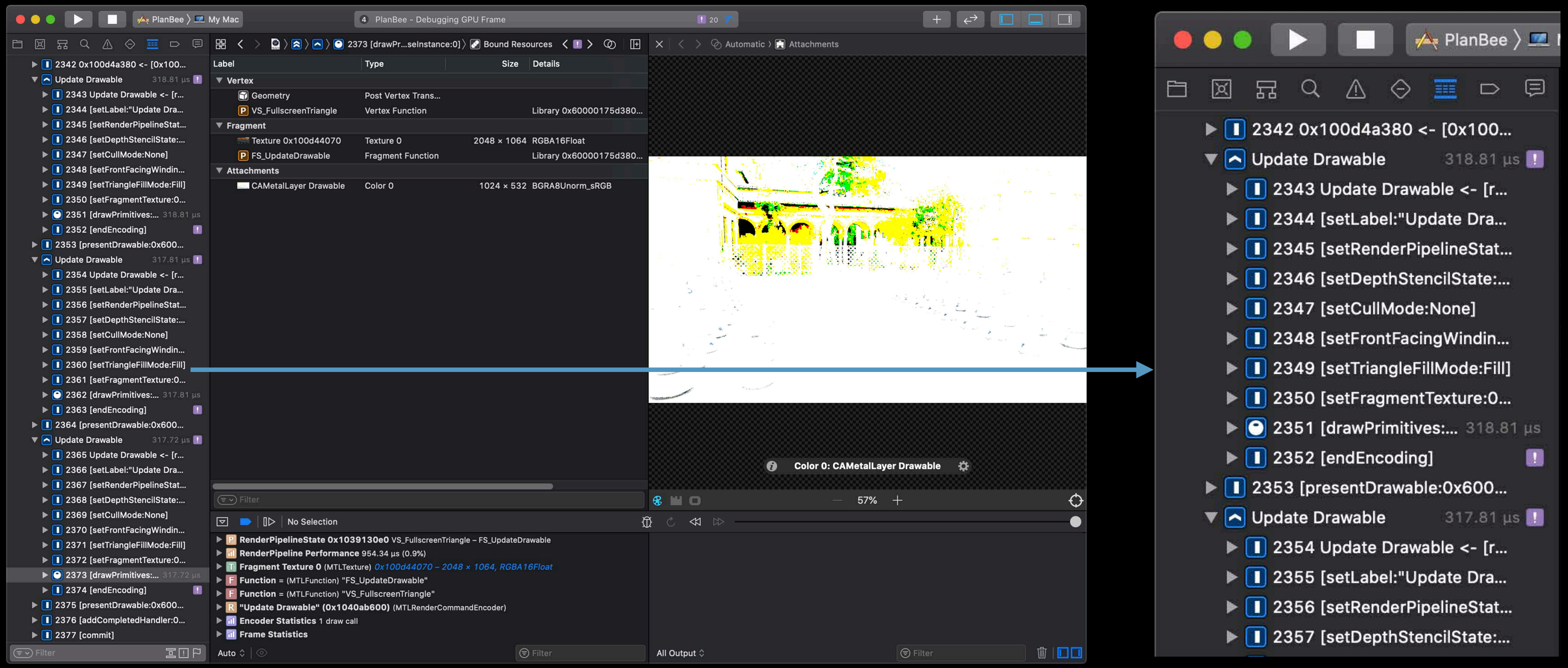
Xcode makes this a breeze to debug

Frame capture

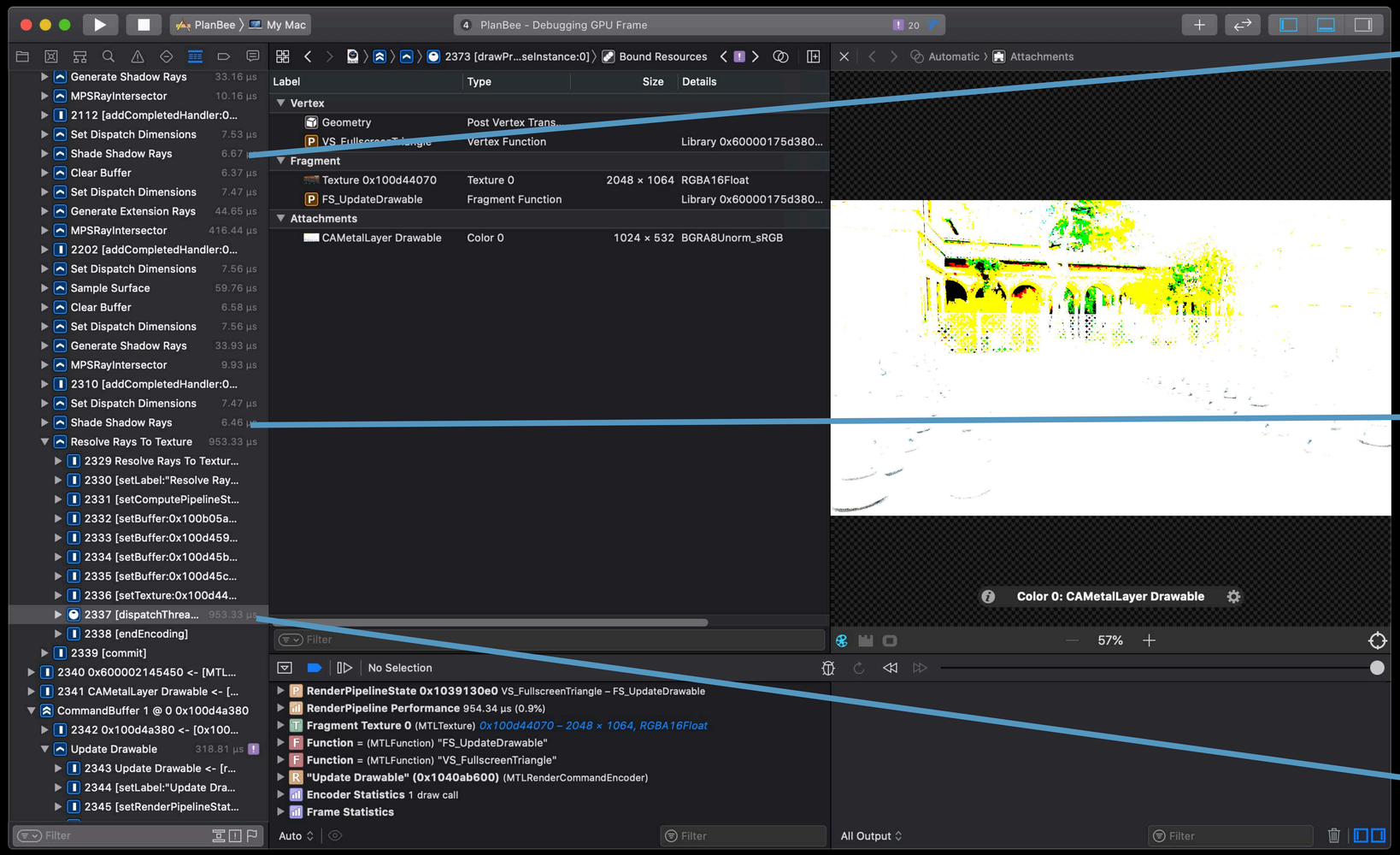
Debugging with Xcode



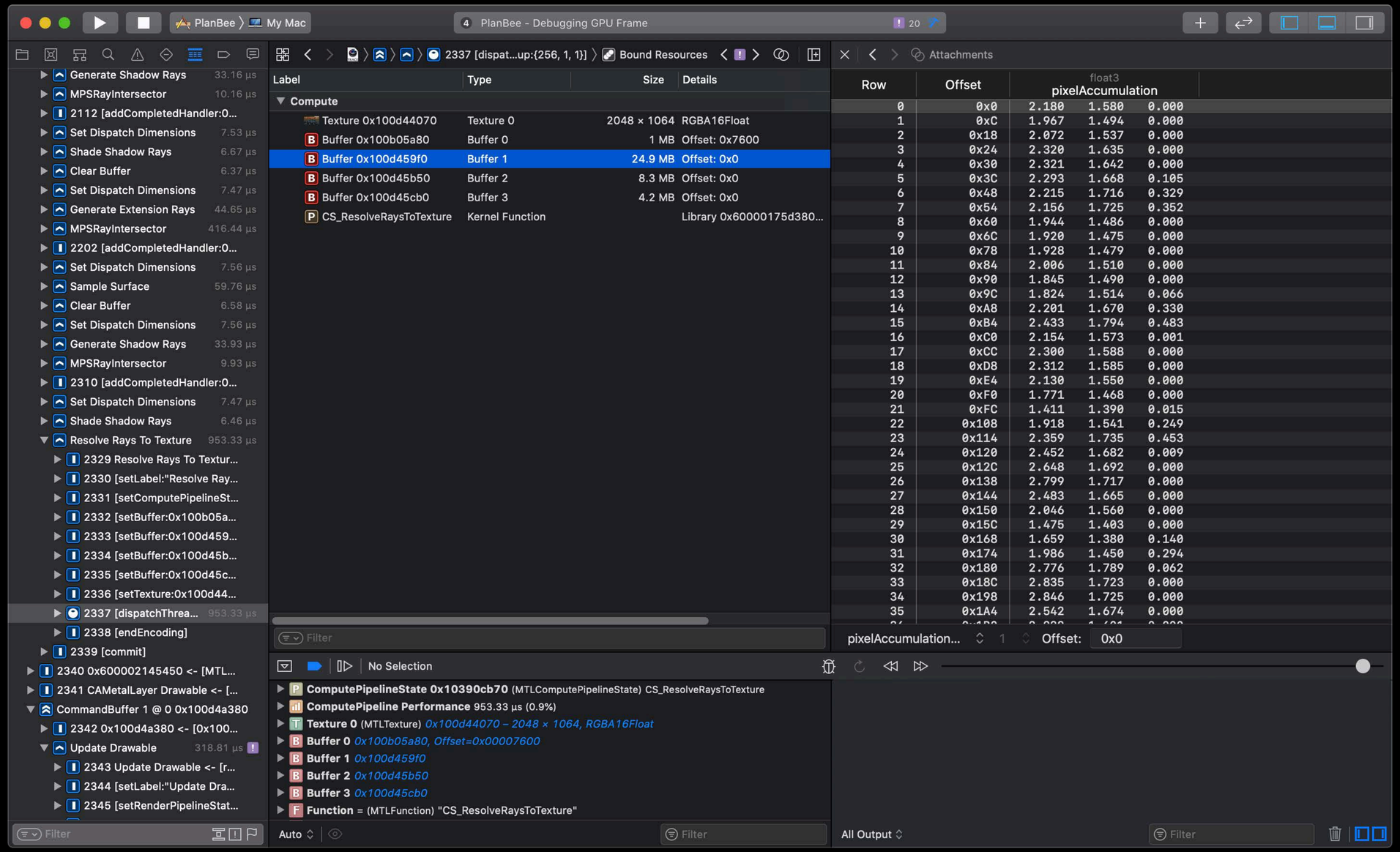
Debugging with Xcode



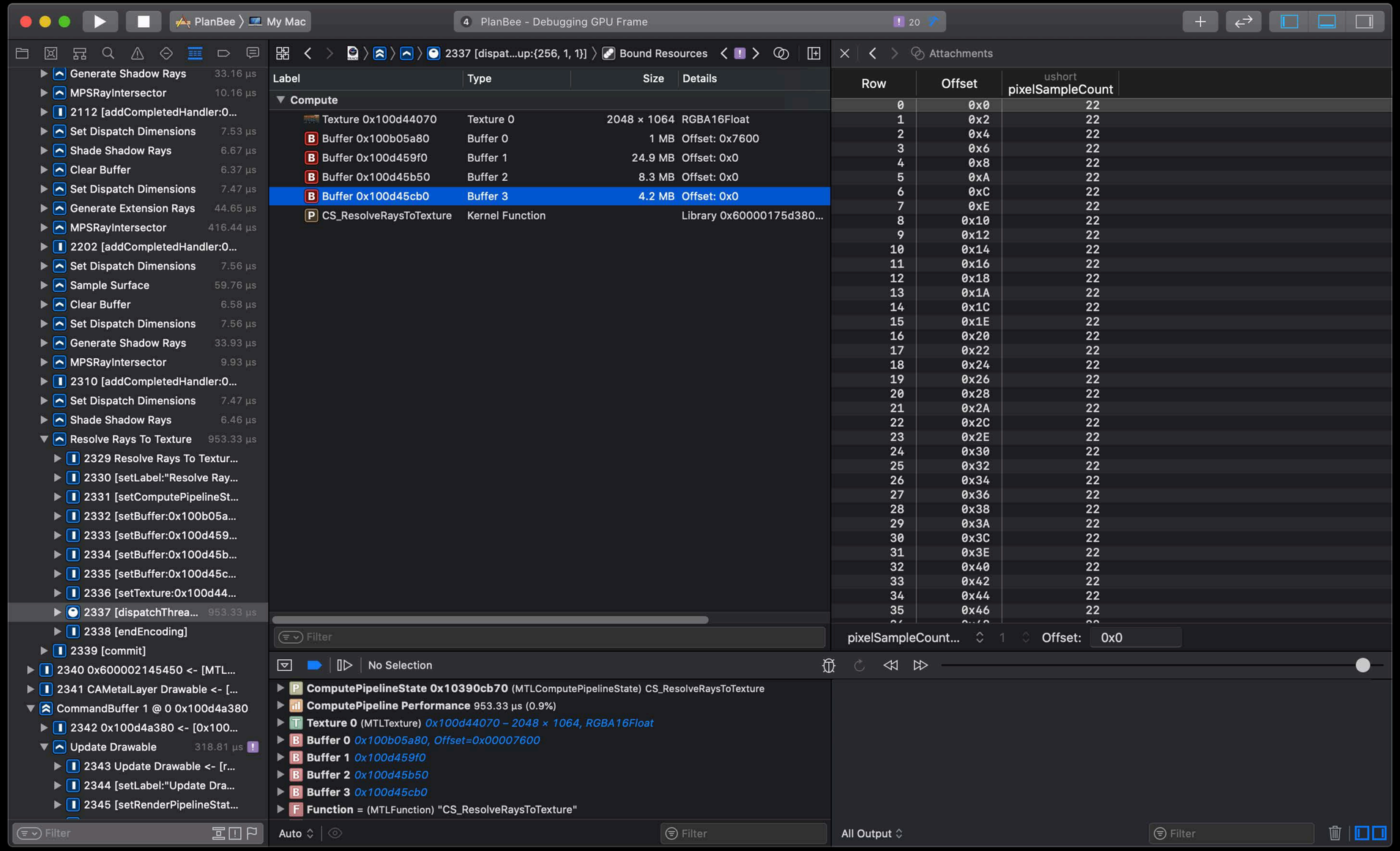
Debugging with Xcode



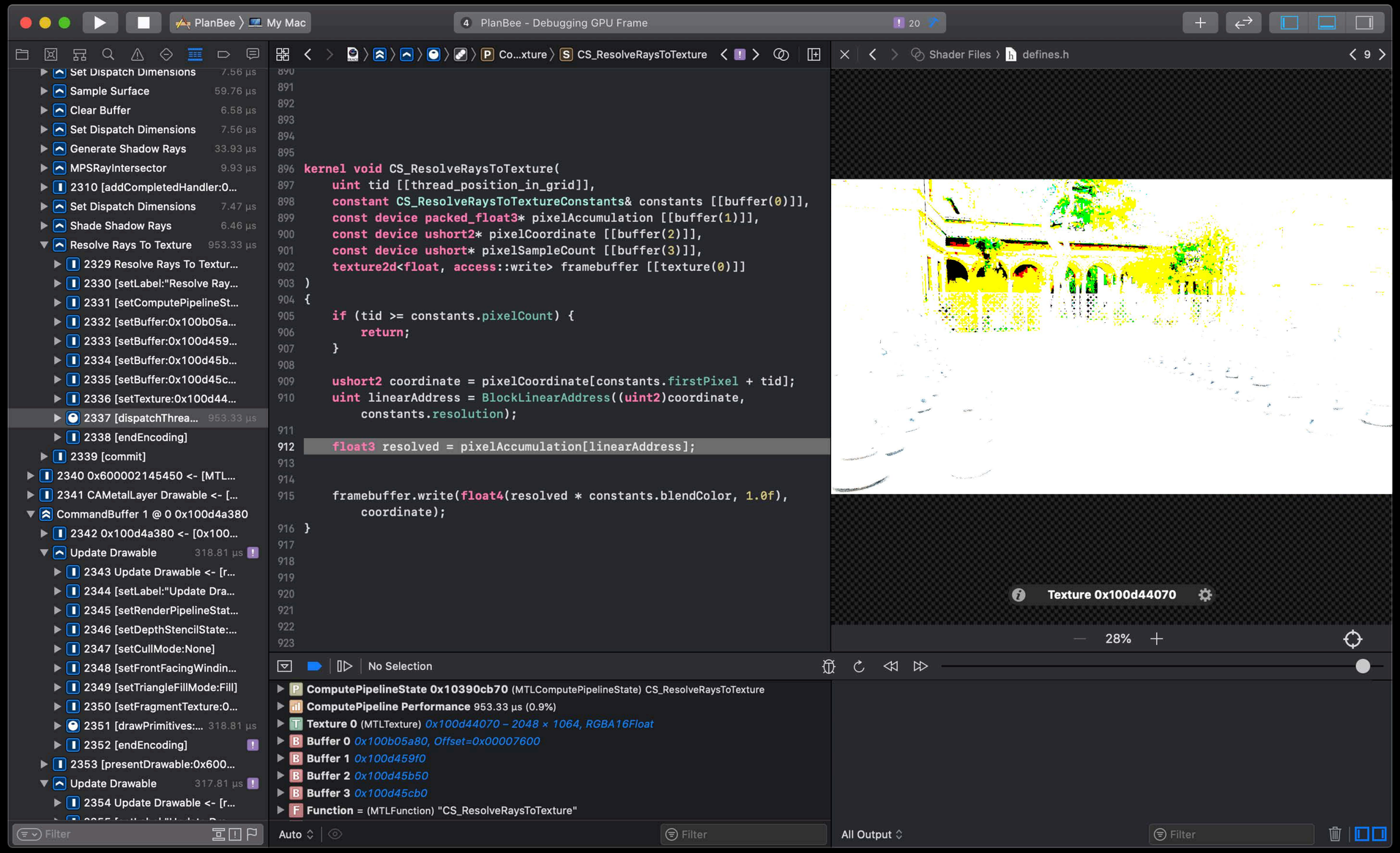
Debugging with Xcode



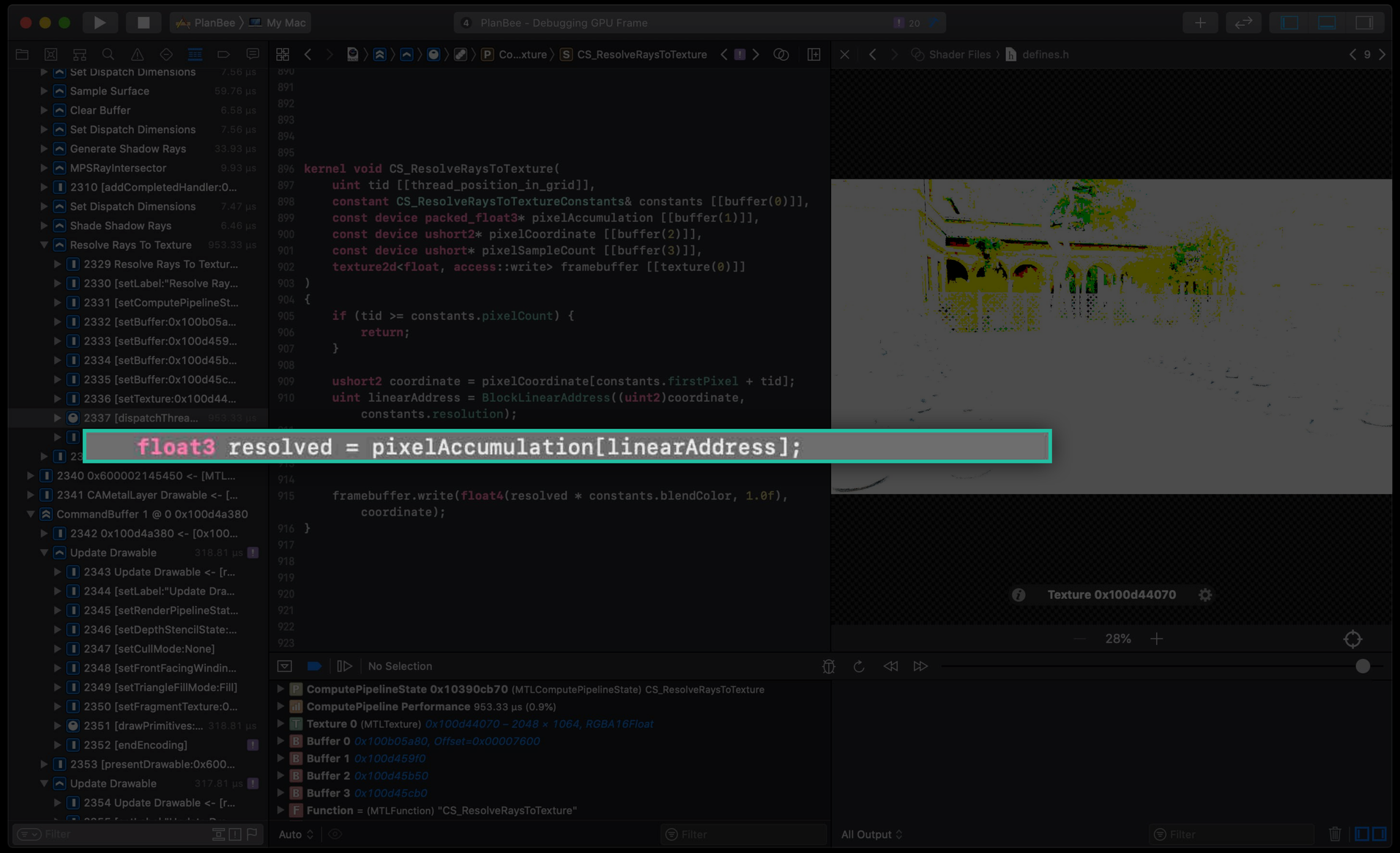
Debugging with Xcode



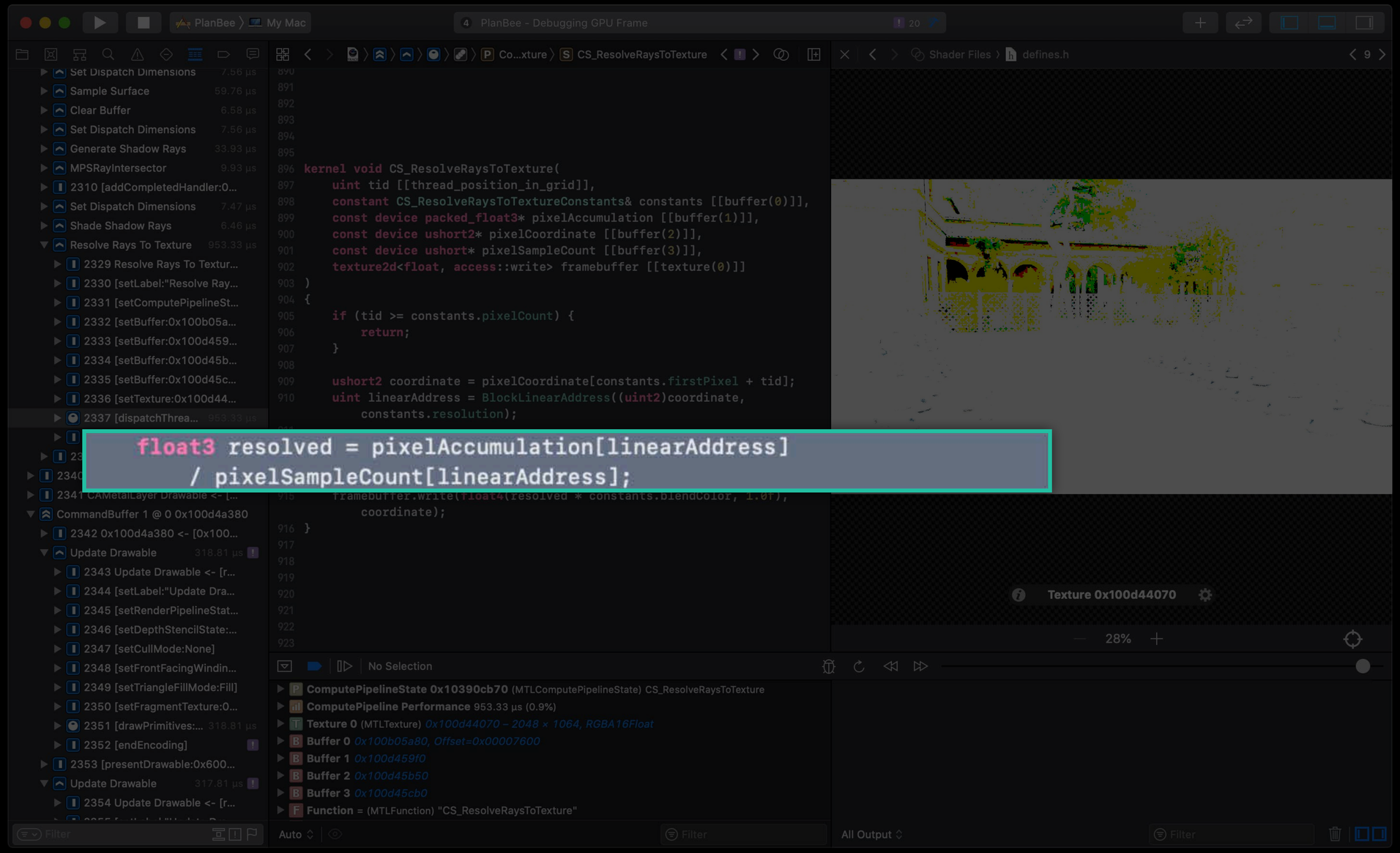
Debugging with Xcode



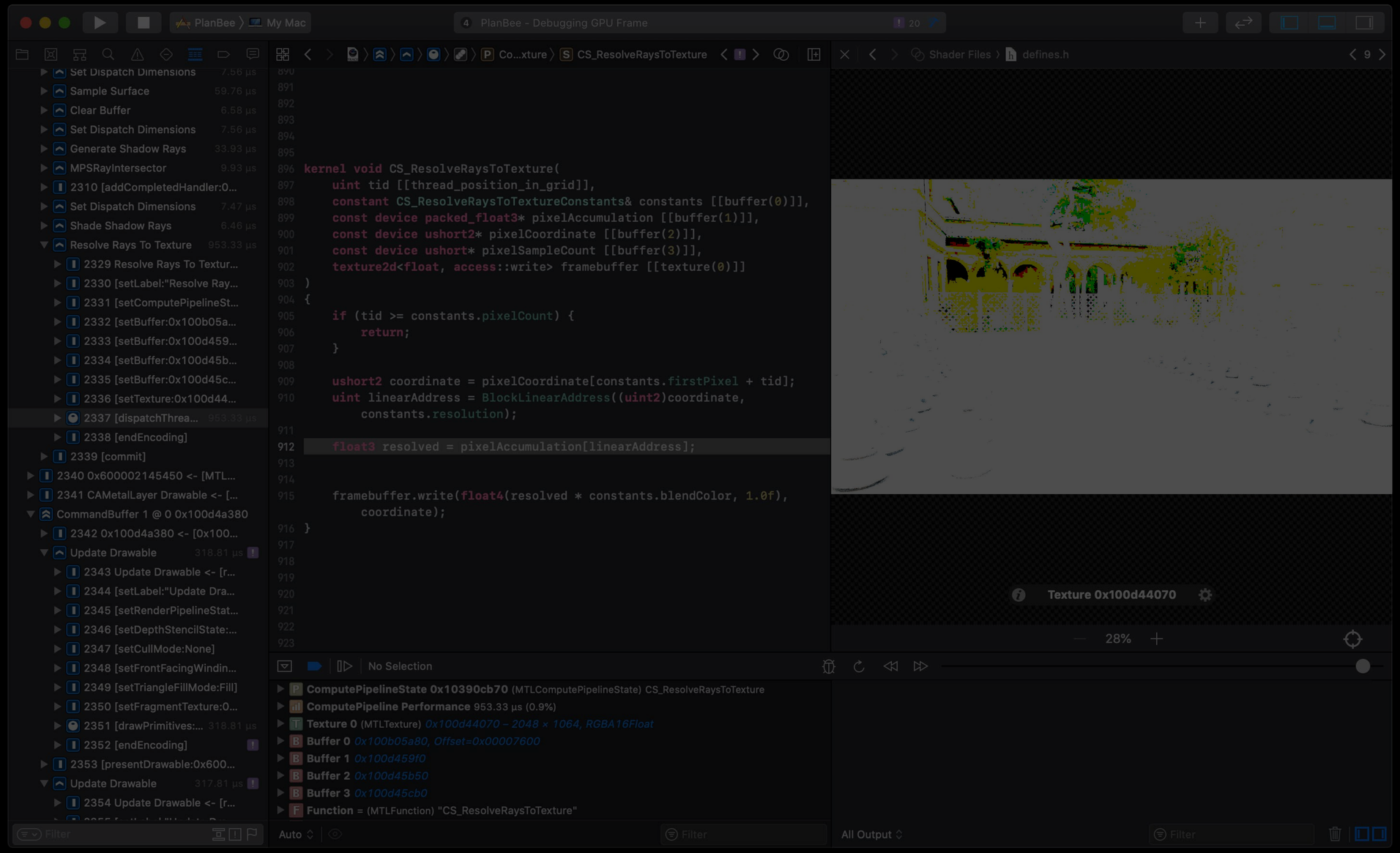
Debugging with Xcode



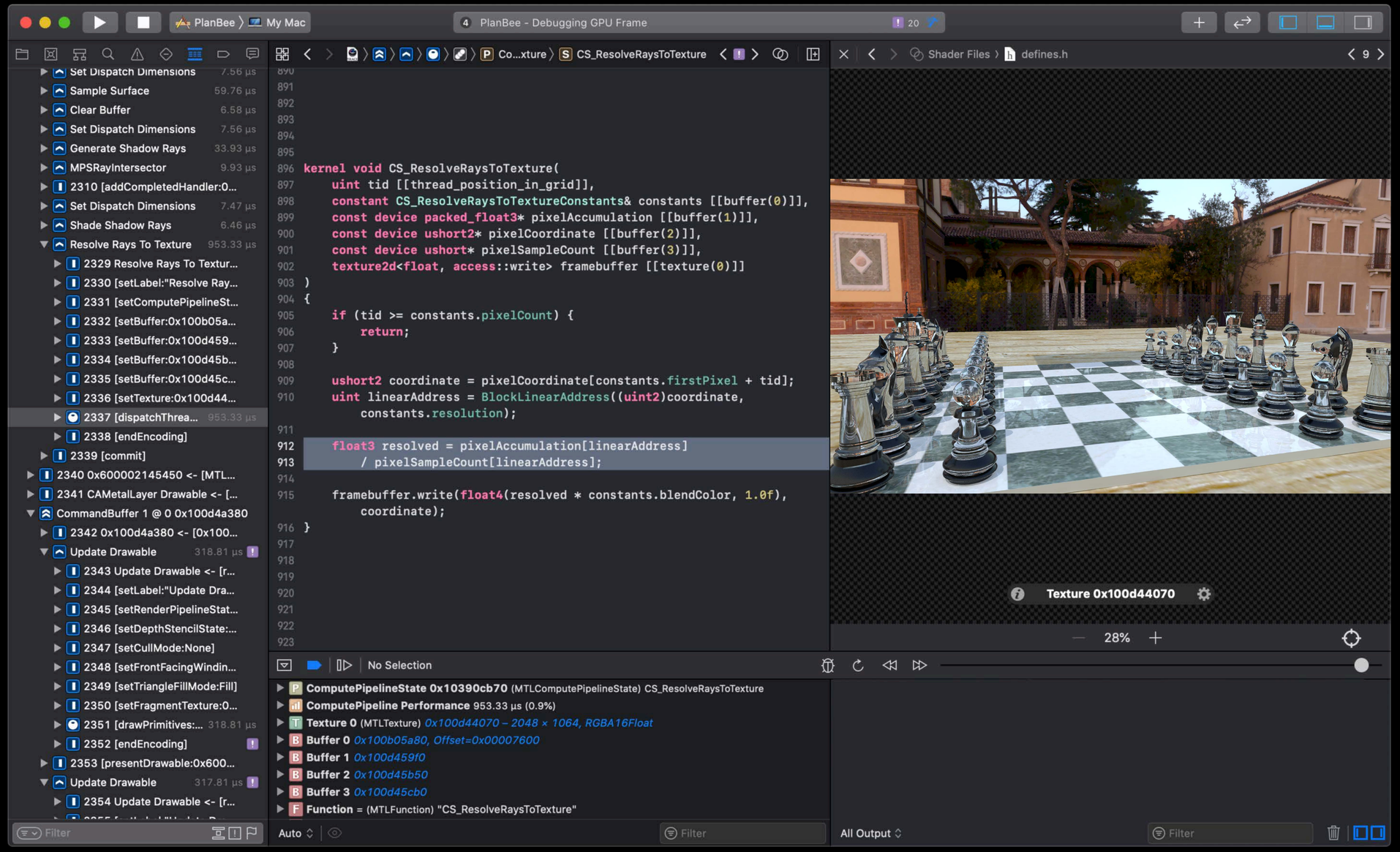
Debugging with Xcode



Debugging with Xcode



Debugging with Xcode



Performance Tuning with Xcode

How to profile code changes?

Xcode profiling tools are very powerful

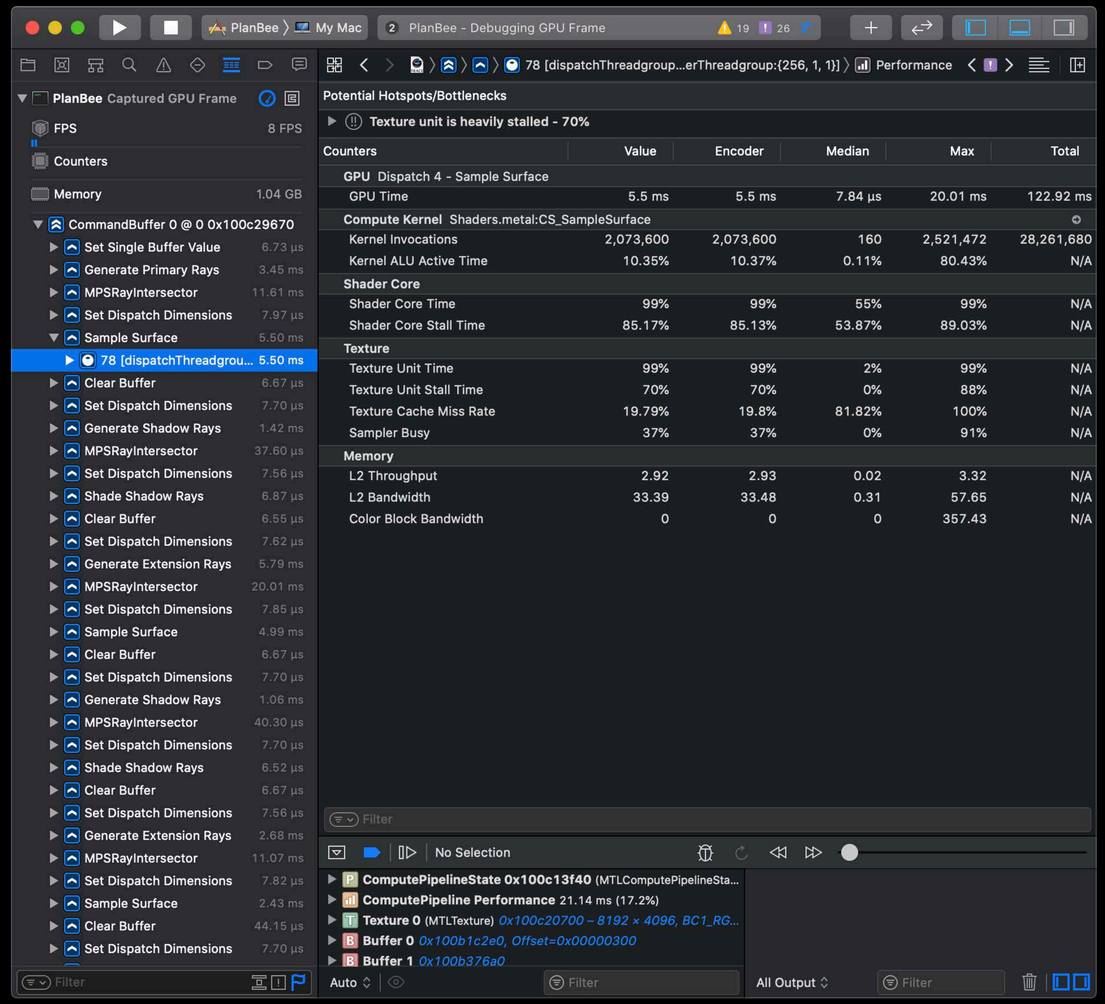
```
struct Surface {
    float3 baseColor;
    float shininess;
    float roughness;
    float emissive;
    float3 transmission; // transparency
    float indexOfRefraction;
};
```

Performance Tuning With Xcode

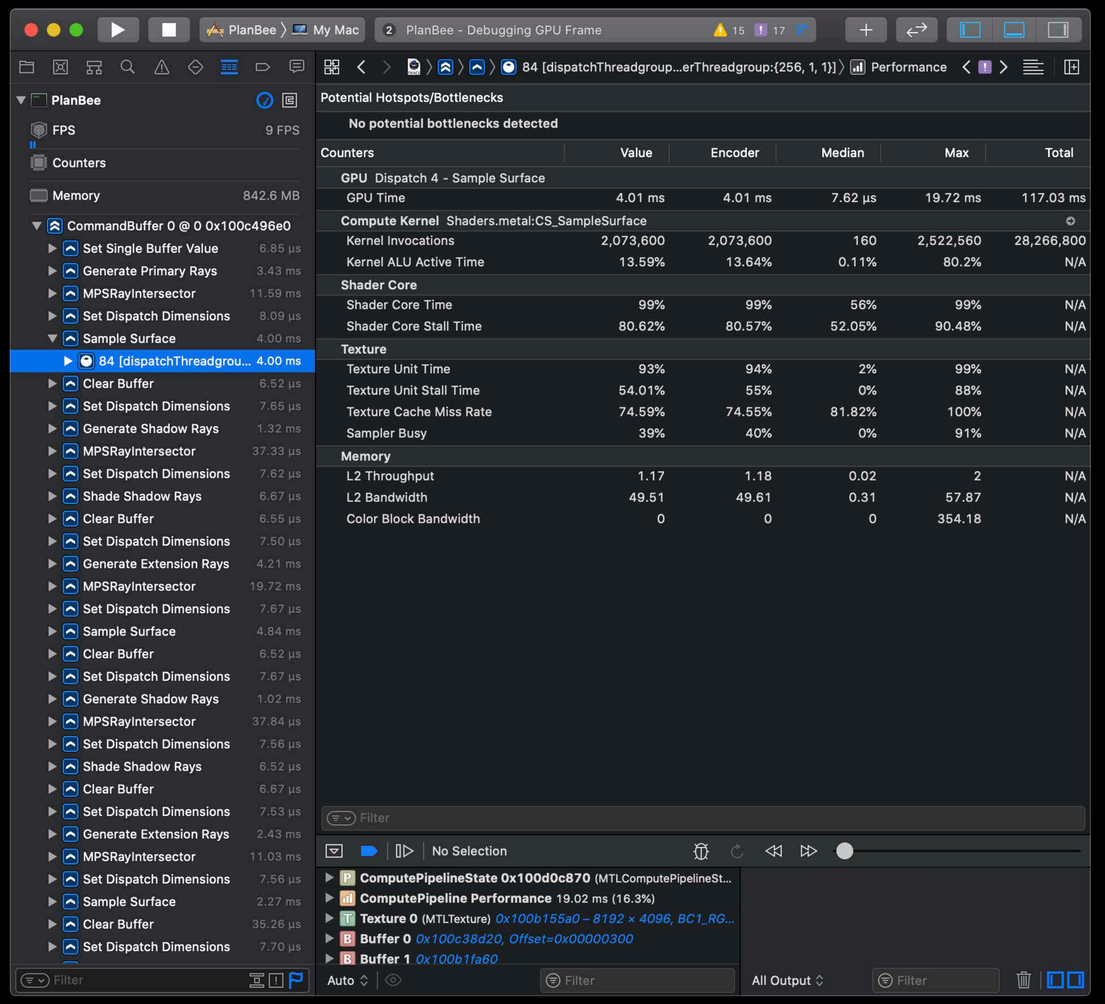
```
struct Surface {
    float3 baseColor;
    float shininess;
    float roughness;
    float emissive;
};
struct SurfaceRefraction {
    float3 transmission; // transparency
    float indexOfRefraction;
};
```

Performance Tuning With Xcode

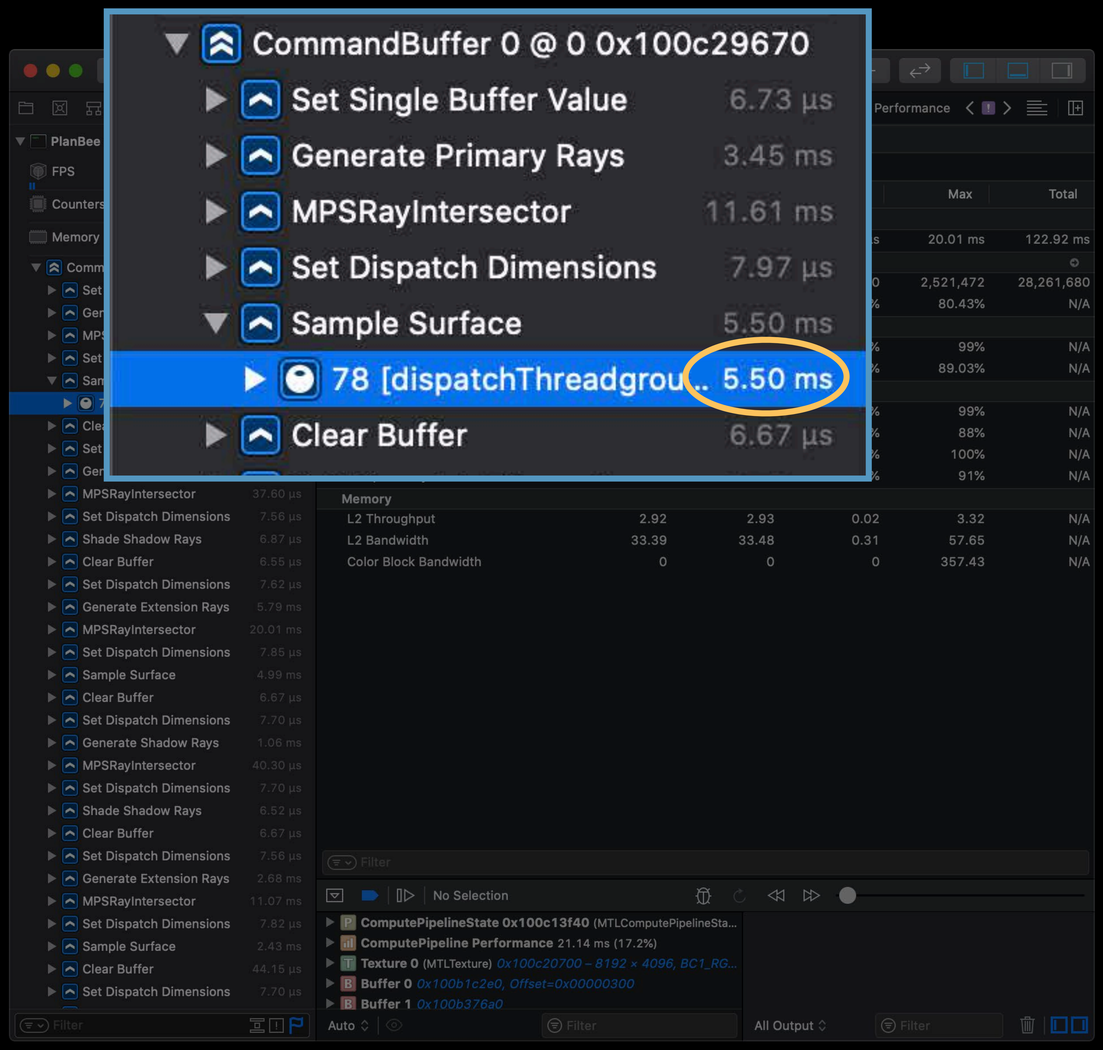
```
struct Surface {
    half3 baseColor;
    half shininess;
    half roughness;
    half emissive;
};
struct SurfaceRefraction {
    half3 transmission; // transparency
    half indexOfRefraction;
};
```

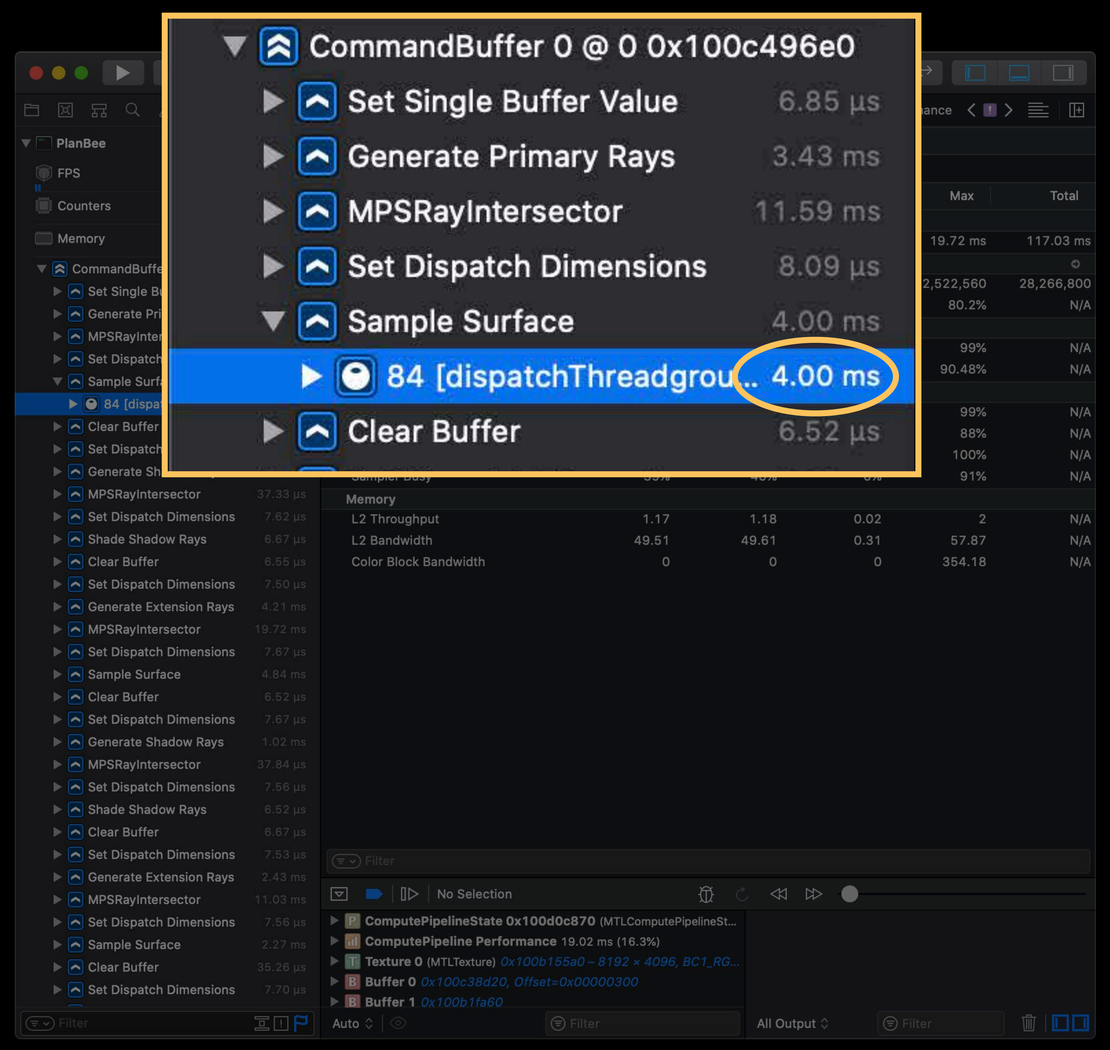


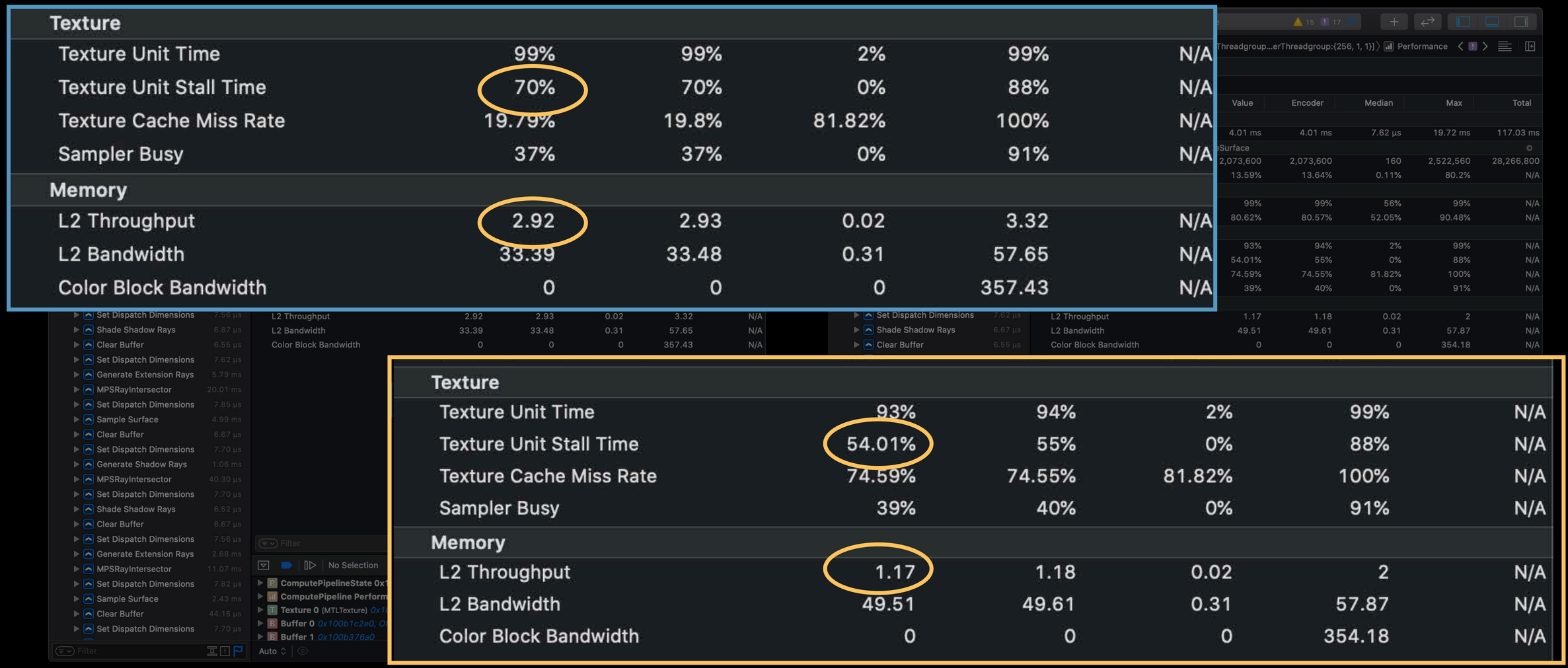
Before Change



After Change

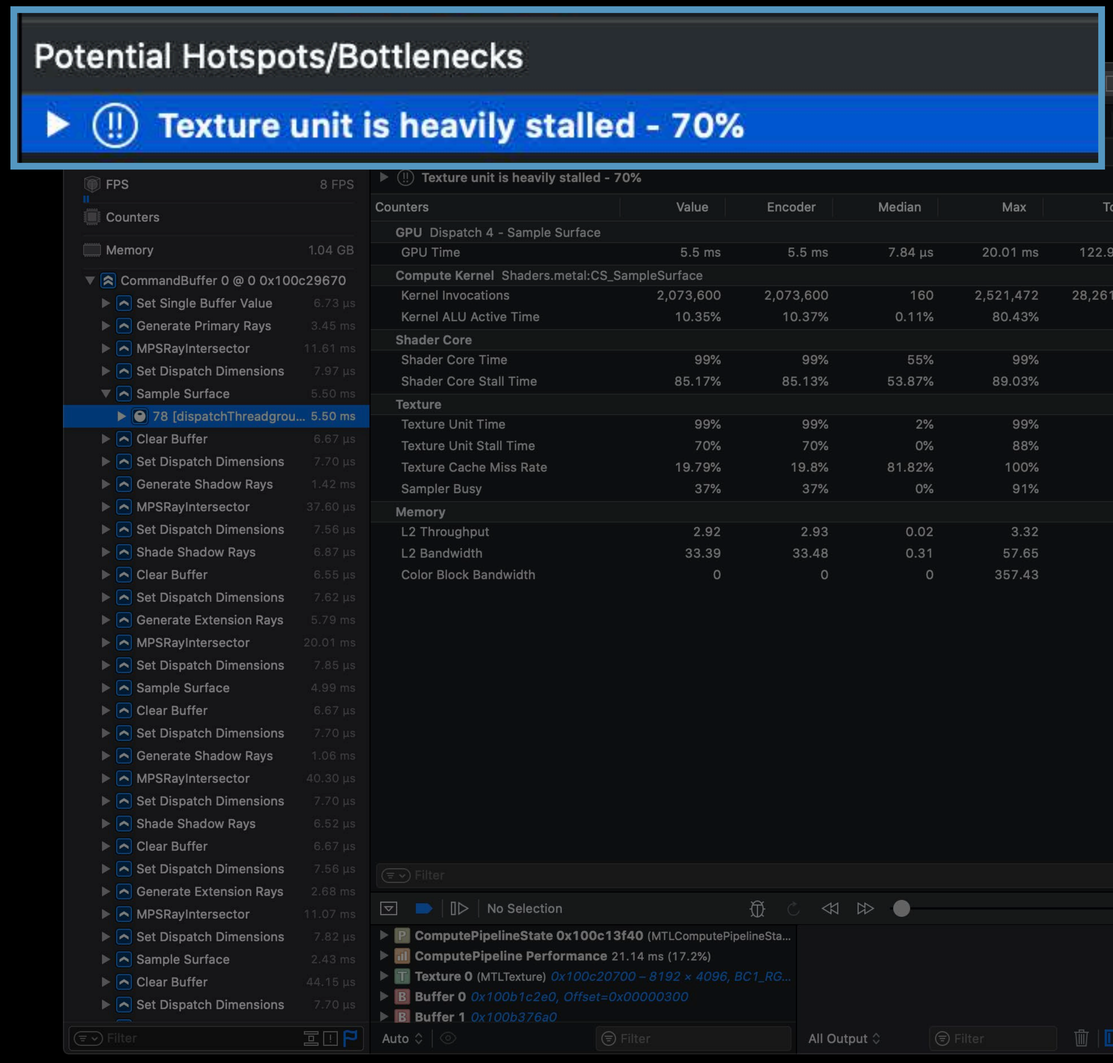

CommandBuffer 0 @ 0 0x100c29670
Set Single Buffer Value 6.73 µs
Generate Primary Rays 3.45 ms
MPSRayIntersector 11.61 ms
Set Dispatch Dimensions 7.97 µs
Sample Surface 5.50 ms
78 [dispatchThreadgrou.. 5.50 ms
Clear Buffer 6.67 µs

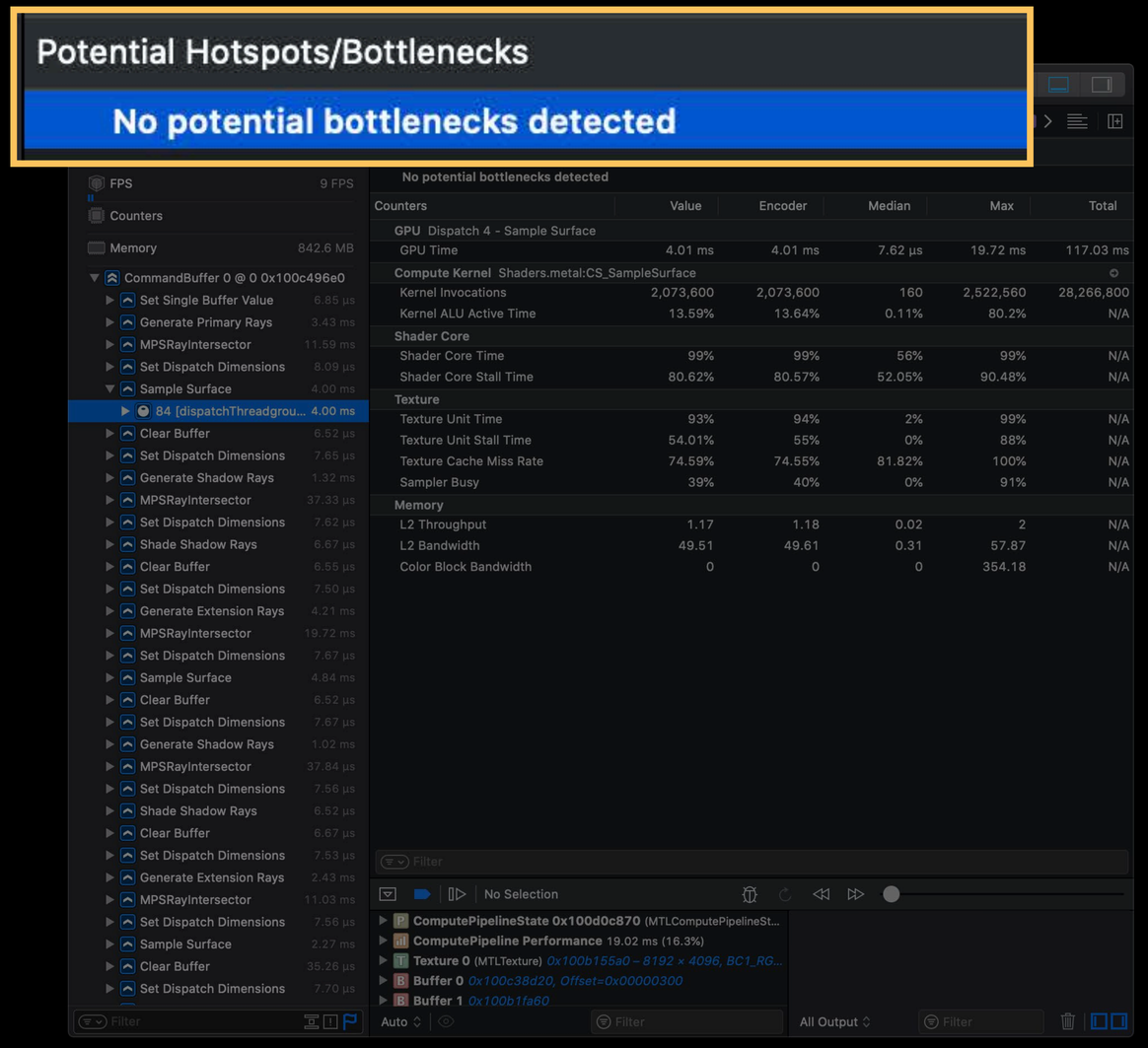

CommandBuffer 0 @ 0 0x100c496e0
Set Single Buffer Value 6.85 µs
Generate Primary Rays 3.43 ms
MPSRayIntersector 11.59 ms
Set Dispatch Dimensions 8.09 µs
Sample Surface 4.00 ms
84 [dispatchThreadgrou.. 4.00 ms
Clear Buffer 6.52 µs

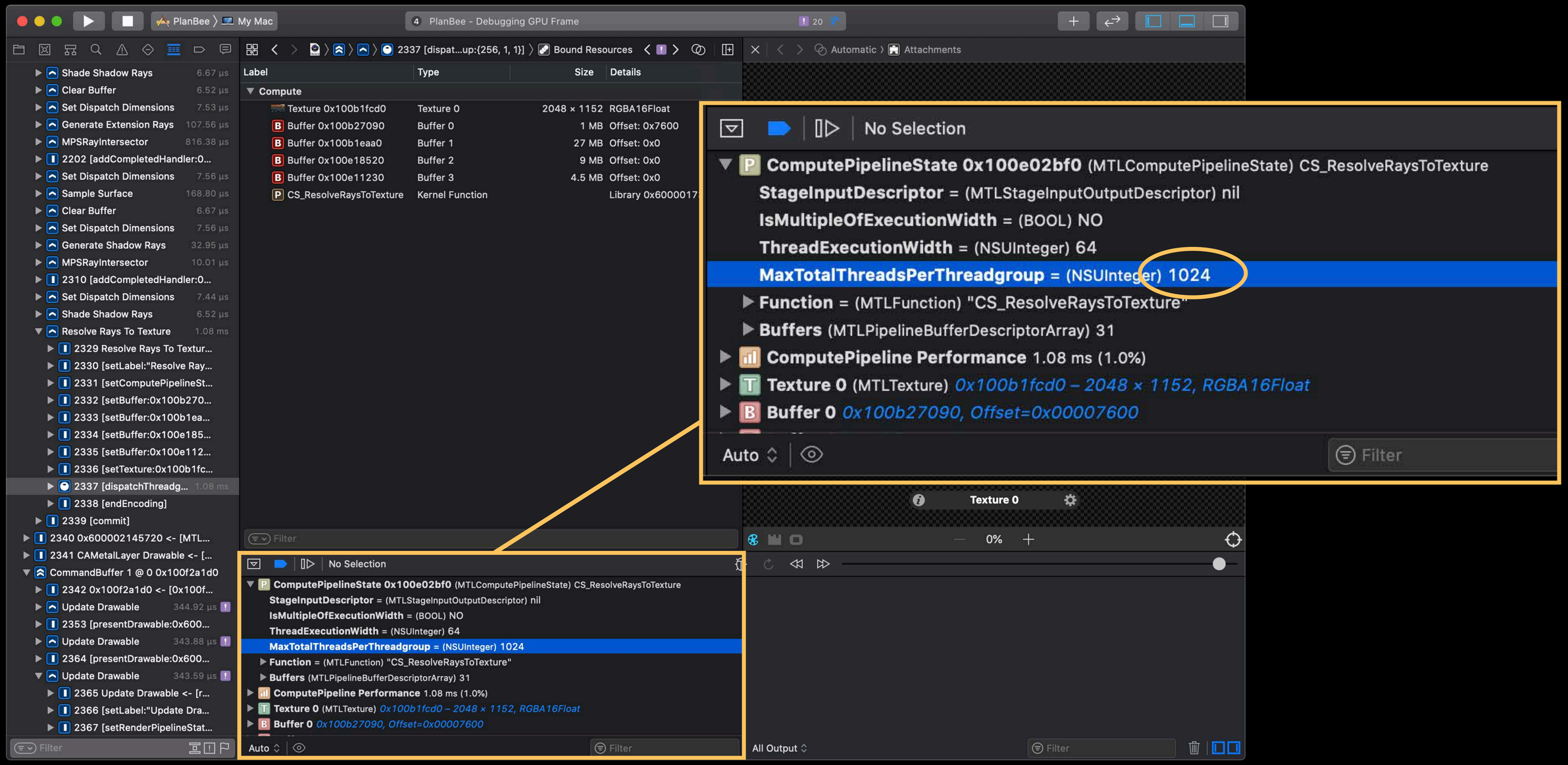


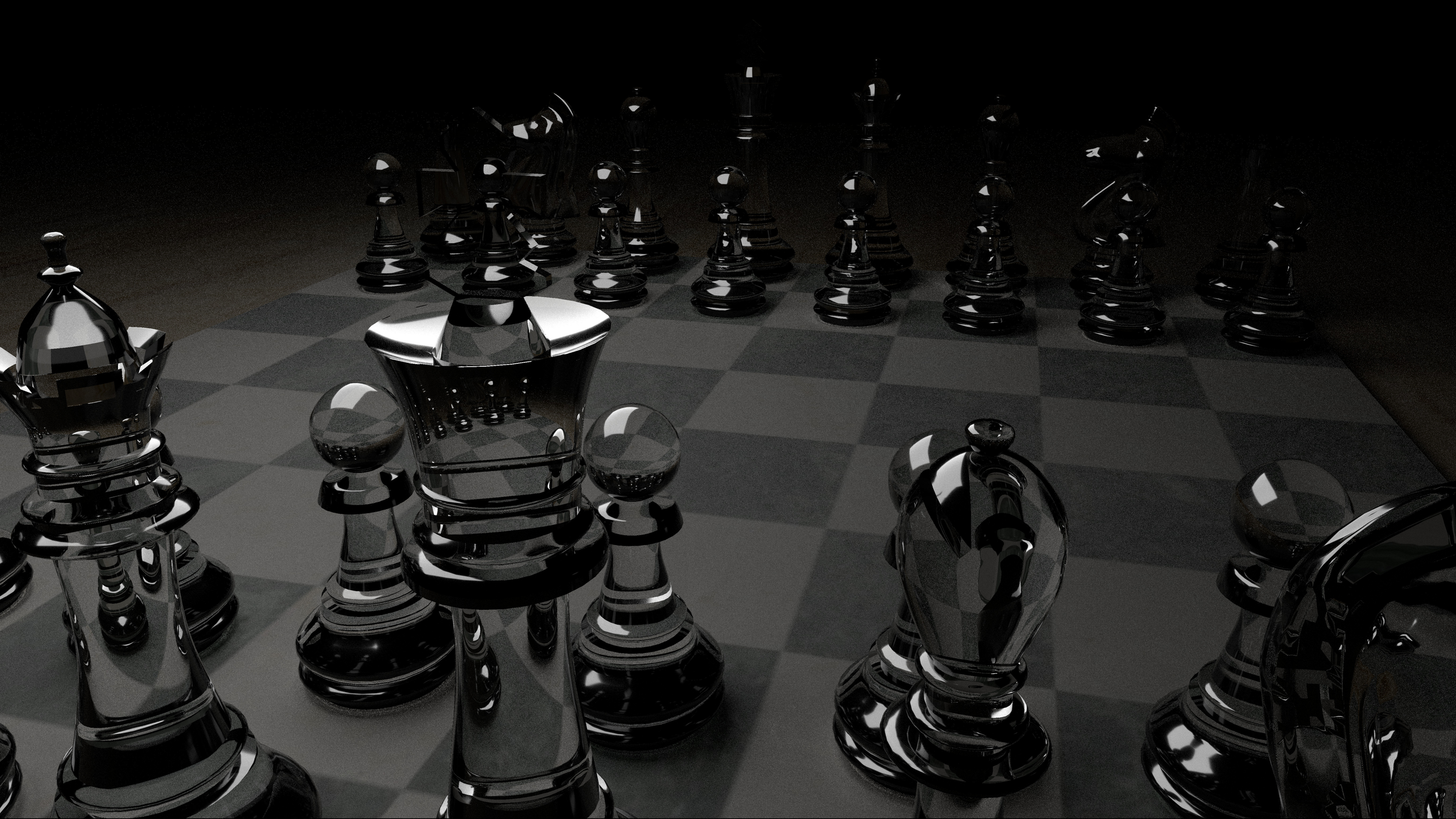
Texture					
Texture Unit Time	99%	99%	2%	99%	N/A
Texture Unit Stall Time	70%	70%	0%	88%	N/A
Texture Cache Miss Rate	19.79%	19.8%	81.82%	100%	N/A
Sampler Busy	37%	37%	0%	91%	N/A
Memory					
L2 Throughput	2.92	2.93	0.02	3.32	N/A
L2 Bandwidth	33.39	33.48	0.31	57.65	N/A
Color Block Bandwidth	0	0	0	357.43	N/A

Texture					
Texture Unit Time	93%	94%	2%	99%	N/A
Texture Unit Stall Time	54.01%	55%	0%	88%	N/A
Texture Cache Miss Rate	74.59%	74.55%	81.82%	100%	N/A
Sampler Busy	39%	40%	0%	91%	N/A
Memory					
L2 Throughput	1.17	1.18	0.02	2	N/A
L2 Bandwidth	49.51	49.61	0.31	57.87	N/A
Color Block Bandwidth	0	0	0	354.18	N/A


Potential Hotspots/Bottlenecks
Texture unit is heavily stalled - 70%
FPS 8 FPS
Counters
Memory 1.04 GB
CommandBuffer 0 @ 0 0x100c29670
Set Single Buffer Value 6.73 µs
Generate Primary Rays 3.45 ms
MPSRayIntersector 11.61 ms
Set Dispatch Dimensions 7.97 µs
Sample Surface 5.50 ms
78 [dispatchThreadgrou... 5.50 ms
Clear Buffer 6.67 µs
Set Dispatch Dimensions 7.70 µs
Generate Shadow Rays 1.42 ms
MPSRayIntersector 37.60 µs
Set Dispatch Dimensions 7.56 µs
Shade Shadow Rays 6.87 µs
Clear Buffer 6.55 µs
Set Dispatch Dimensions 7.62 µs
Generate Extension Rays 5.79 ms
MPSRayIntersector 20.01 ms
Set Dispatch Dimensions 7.85 µs
Sample Surface 4.99 ms
Clear Buffer 6.67 µs
Set Dispatch Dimensions 7.70 µs
Generate Shadow Rays 1.06 ms
MPSRayIntersector 40.30 µs
Set Dispatch Dimensions 7.70 µs
Shade Shadow Rays 6.52 µs
Clear Buffer 6.67 µs
Set Dispatch Dimensions 7.56 µs
Generate Extension Rays 2.68 ms
MPSRayIntersector 11.07 ms
Set Dispatch Dimensions 7.82 µs
Sample Surface 2.43 ms
Clear Buffer 44.15 µs
Set Dispatch Dimensions 7.70 µs
Texture unit is heavily stalled - 70%
Counters Value Encoder Median Max Total
GPU Dispatch 4 - Sample Surface
GPU Time 5.5 ms 5.5 ms 7.84 µs 20.01 ms 122.92 ms
Compute Kernel Shaders.metal:CS_SampleSurface
Kernel Invocations 2,073,600 2,073,600 160 2,521,472 28,261,680
Kernel ALU Active Time 10.35% 10.37% 0.11% 80.43% N/A
Shader Core
Shader Core Time 99% 99% 55% 99% N/A
Shader Core Stall Time 85.17% 85.13% 53.87% 89.03% N/A
Texture
Texture Unit Time 99% 99% 2% 99% N/A
Texture Unit Stall Time 70% 70% 0% 88% N/A
Texture Cache Miss Rate 19.79% 19.8% 81.82% 100% N/A
Sampler Busy 37% 37% 0% 91% N/A
Memory
L2 Throughput 2.92 2.93 0.02 3.32 N/A
L2 Bandwidth 33.39 33.48 0.31 57.65 N/A
Color Block Bandwidth 0 0 0 357.43 N/A
Filter
No Selection
ComputePipelineState 0x100c13f40 (MTLComputePipelineSta...
ComputePipeline Performance 21.14 ms (17.2%)
Texture 0 (MTLTexture) 0x100c20700 – 8192 × 4096, BC1_RG...
Buffer 0 0x100b1c2e0, Offset=0x00000300
Buffer 1 0x100b376a0
Auto
All Output
Filter


Potential Hotspots/Bottlenecks
No potential bottlenecks detected
FPS 9 FPS
Counters
Memory 842.6 MB
CommandBuffer 0 @ 0 0x100c496e0
Set Single Buffer Value 6.85 µs
Generate Primary Rays 3.43 ms
MPSRayIntersector 11.59 ms
Set Dispatch Dimensions 8.09 µs
Sample Surface 4.00 ms
84 [dispatchThreadgrou... 4.00 ms
Clear Buffer 6.52 µs
Set Dispatch Dimensions 7.65 µs
Generate Shadow Rays 1.32 ms
MPSRayIntersector 37.33 µs
Set Dispatch Dimensions 7.62 µs
Shade Shadow Rays 6.67 µs
Clear Buffer 6.55 µs
Set Dispatch Dimensions 7.50 µs
Generate Extension Rays 4.21 ms
MPSRayIntersector 19.72 ms
Set Dispatch Dimensions 7.67 µs
Sample Surface 4.84 ms
Clear Buffer 6.52 µs
Set Dispatch Dimensions 7.67 µs
Generate Shadow Rays 1.02 ms
MPSRayIntersector 37.84 µs
Set Dispatch Dimensions 7.56 µs
Shade Shadow Rays 6.52 µs
Clear Buffer 6.67 µs
Set Dispatch Dimensions 7.53 µs
Generate Extension Rays 2.43 ms
MPSRayIntersector 11.03 ms
Set Dispatch Dimensions 7.56 µs
Sample Surface 2.27 ms
Clear Buffer 35.26 µs
Set Dispatch Dimensions 7.70 µs
No potential bottlenecks detected
Counters Value Encoder Median Max Total
GPU Dispatch 4 - Sample Surface
GPU Time 4.01 ms 4.01 ms 7.62 µs 19.72 ms 117.03 ms
Compute Kernel Shaders.metal:CS_SampleSurface
Kernel Invocations 2,073,600 2,073,600 160 2,522,560 28,266,800
Kernel ALU Active Time 13.59% 13.64% 0.11% 80.2% N/A
Shader Core
Shader Core Time 99% 99% 56% 99% N/A
Shader Core Stall Time 80.62% 80.57% 52.05% 90.48% N/A
Texture
Texture Unit Time 93% 94% 2% 99% N/A
Texture Unit Stall Time 54.01% 55% 0% 88% N/A
Texture Cache Miss Rate 74.59% 74.55% 81.82% 100% N/A
Sampler Busy 39% 40% 0% 91% N/A
Memory
L2 Throughput 1.17 1.18 0.02 2 N/A
L2 Bandwidth 49.51 49.61 0.31 57.87 N/A
Color Block Bandwidth 0 0 0 354.18 N/A
Filter
No Selection
ComputePipelineState 0x100d0c870 (MTLComputePipelineSt...
ComputePipeline Performance 19.02 ms (16.3%)
Texture 0 (MTLTexture) 0x100b155a0 – 8192 × 4096, BC1_RG...
Buffer 0 0x100c38d20, Offset=0x00000300
Buffer 1 0x100b1fa60
Auto
All Output
Filter


PlanBee 〉 My Mac
PlanBee - Debugging GPU Frame
2337 [dispat...up:{256, 1, 1}] 〉 Bound Resources
Automatic 〉 Attachments
Label
Type
Size
Details
Compute
Texture 0x100b1fcd0 Texture 0 2048 × 1152 RGBA16Float
Buffer 0x100b27090 Buffer 0 1 MB Offset: 0x7600
Buffer 0x100b1eaa0 Buffer 1 27 MB Offset: 0x0
Buffer 0x100e18520 Buffer 2 9 MB Offset: 0x0
Buffer 0x100e11230 Buffer 3 4.5 MB Offset: 0x0
CS_ResolveRaysToTexture Kernel Function
Shade Shadow Rays 6.67 µs
Clear Buffer 6.52 µs
Set Dispatch Dimensions 7.53 µs
Generate Extension Rays 107.56 µs
MPSRayIntersector 816.38 µs
2202 [addCompletedHandler:0...
Set Dispatch Dimensions 7.56 µs
Sample Surface 168.80 µs
Clear Buffer 6.67 µs
Set Dispatch Dimensions 7.56 µs
Generate Shadow Rays 32.95 µs
MPSRayIntersector 10.01 µs
2310 [addCompletedHandler:0...
Set Dispatch Dimensions 7.44 µs
Shade Shadow Rays 6.52 µs
Resolve Rays To Texture 1.08 ms
2329 Resolve Rays To Textur...
2330 [setLabel:"Resolve Ray...
2331 [setComputePipelineSt...
2332 [setBuffer:0x100b270...
2333 [setBuffer:0x100b1ea...
2334 [setBuffer:0x100e185...
2335 [setBuffer:0x100e112...
2336 [setTexture:0x100b1fc...
2337 [dispatchThreadg... 1.08 ms
2338 [endEncoding]
2339 [commit]
2340 0x600002145720 <- [MTL...
2341 CAMetalLayer Drawable <- [...
CommandBuffer 1 @ 0 0x100f2a1d0
2342 0x100f2a1d0 <- [0x100f...
Update Drawable 344.92 µs
2353 [presentDrawable:0x600...
Update Drawable 343.88 µs
2364 [presentDrawable:0x600...
Update Drawable 343.59 µs
2365 Update Drawable <- [r...
2366 [setLabel:"Update Dra...
2367 [setRenderPipelineStat...
Filter
No Selection
ComputePipelineState 0x100e02bf0 (MTLComputePipelineState) CS_ResolveRaysToTexture
StageInputDescriptor = (MTLStageInputOutputDescriptor) nil
IsMultipleOfExecutionWidth = (BOOL) NO
ThreadExecutionWidth = (NSUInteger) 64
MaxTotalThreadsPerThreadgroup = (NSUInteger) 1024
Function = (MTLFunction) "CS_ResolveRaysToTexture"
Buffers (MTLPipelineBufferDescriptorArray) 31
ComputePipeline Performance 1.08 ms (1.0%)
Texture 0 (MTLTexture) 0x100b1fcd0 – 2048 × 1152, RGBA16Float
Buffer 0 0x100b27090, Offset=0x00007600
Auto
Texture 0
0%
All Output



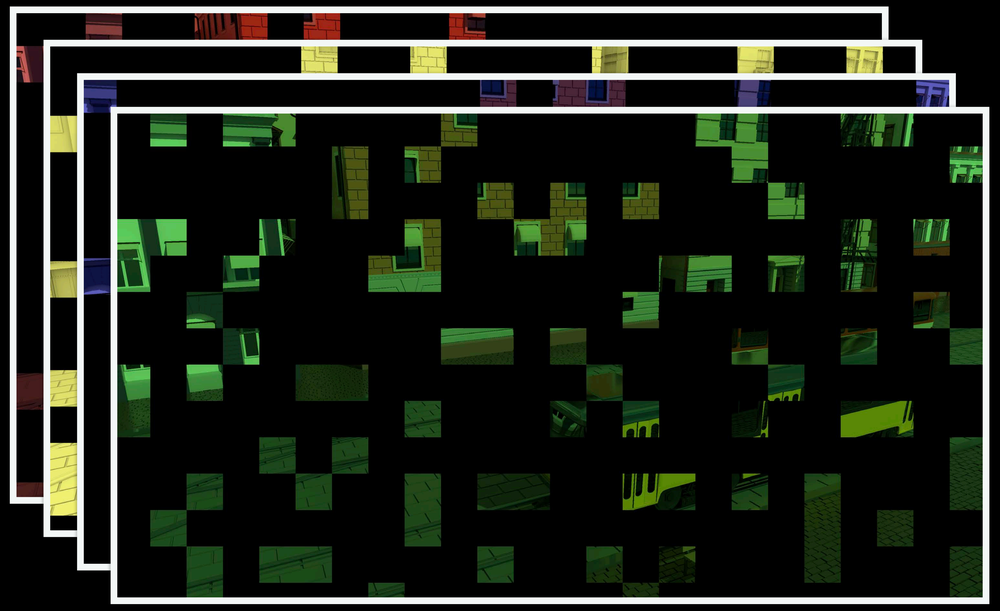
Demo

Interleaved Tiling

Divide screen into tiles, render on different GPUs

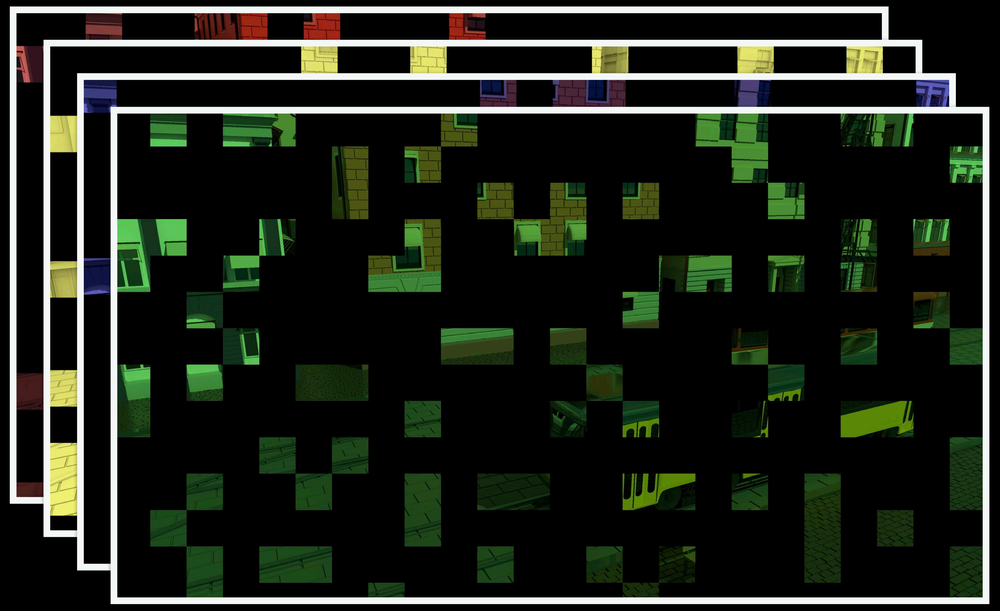
Interleaved Tiling

Divide screen into tiles, render on different GPUs



Interleaved Tiling

Divide screen into tiles, render on different GPUs



Load Balancing

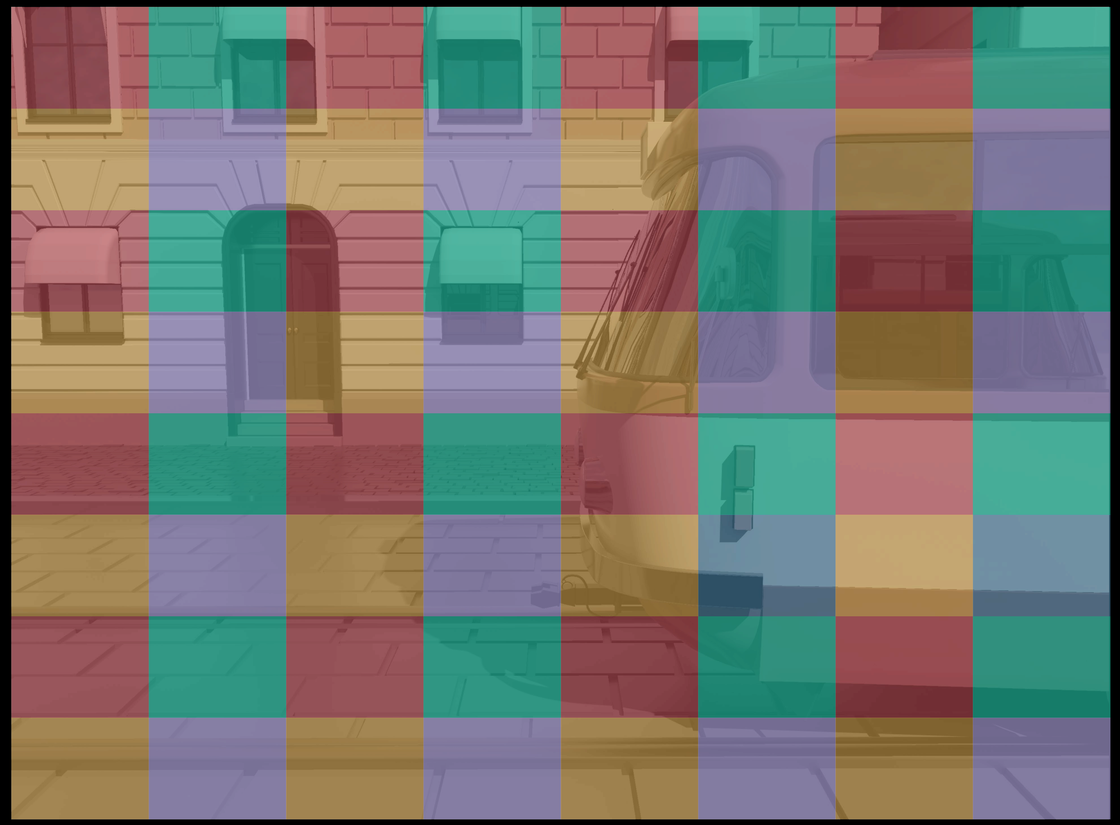
Smaller tiles distribute rendering more evenly across GPUs

Load Balancing

Smaller tiles distribute rendering more evenly across GPUs

Load Balancing

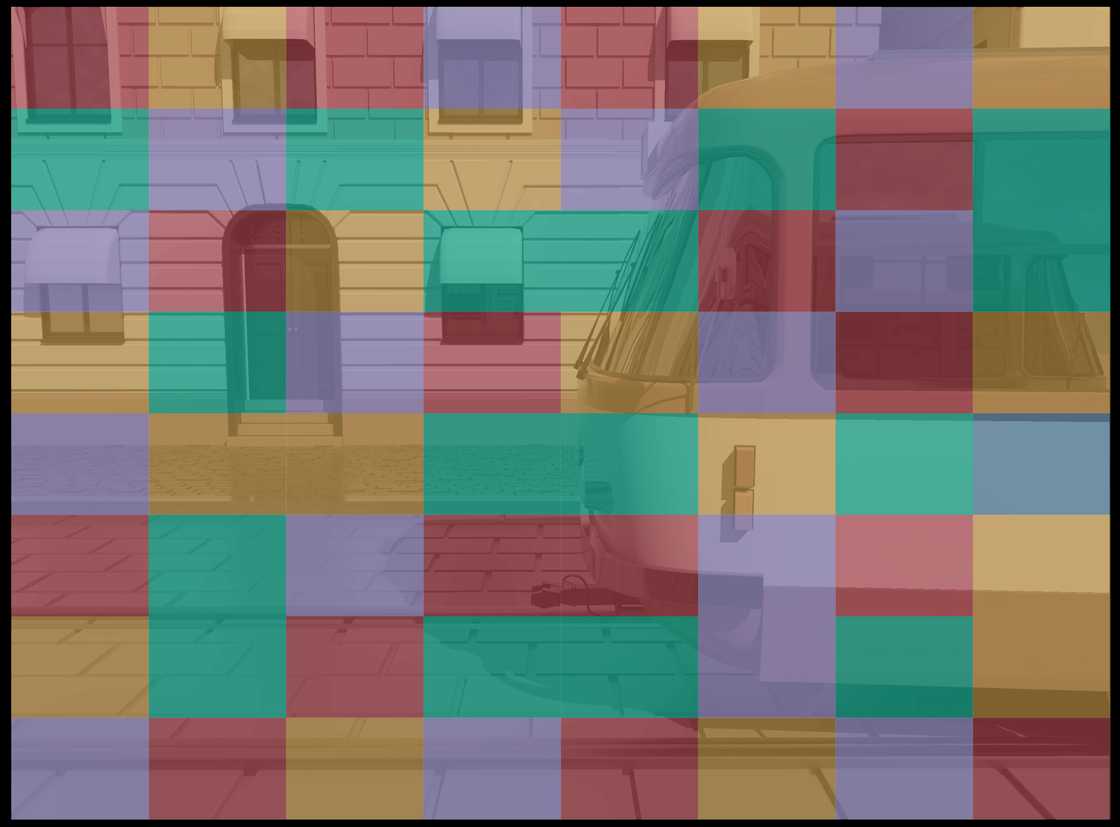
Smaller tiles distribute rendering more evenly across GPUs



Load Balancing

Smaller tiles distribute rendering more evenly across GPUs

Pseudo-random assignment avoids correlation with the scene

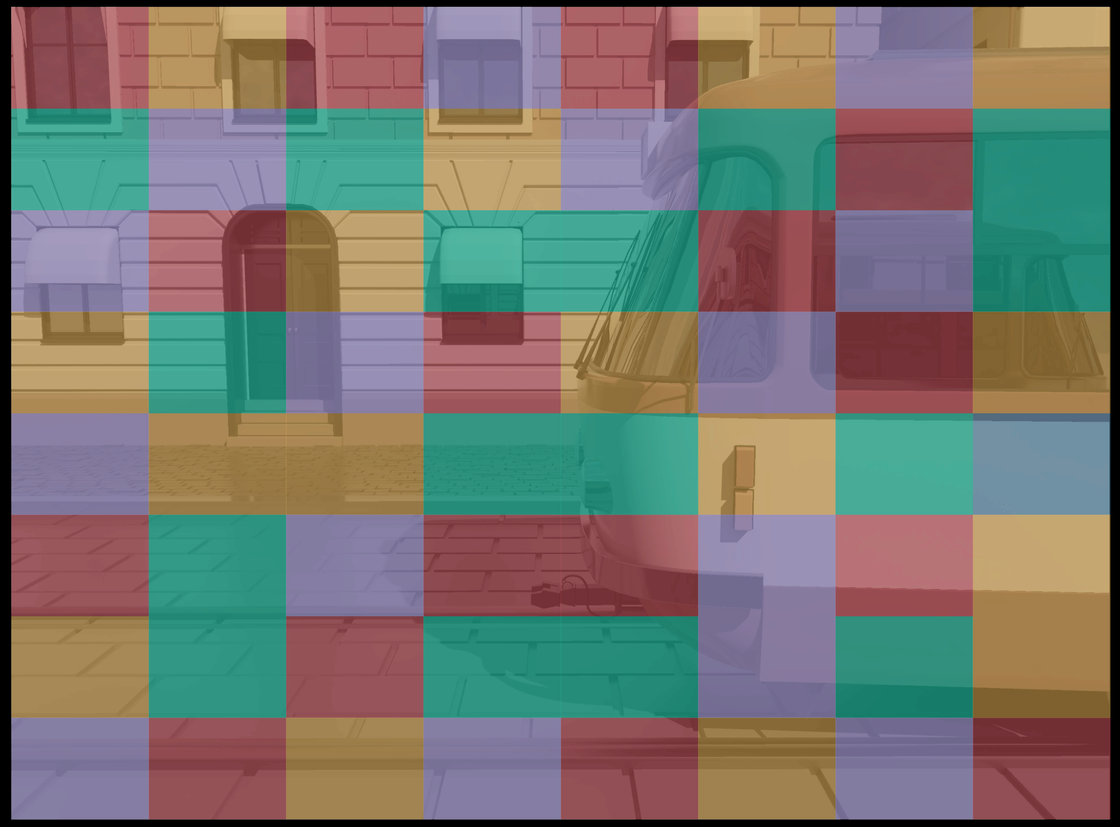


Load Balancing

Smaller tiles distribute rendering more evenly across GPUs

Pseudo-random assignment avoids correlation with the scene

Same GPU renders the same tiles each frame

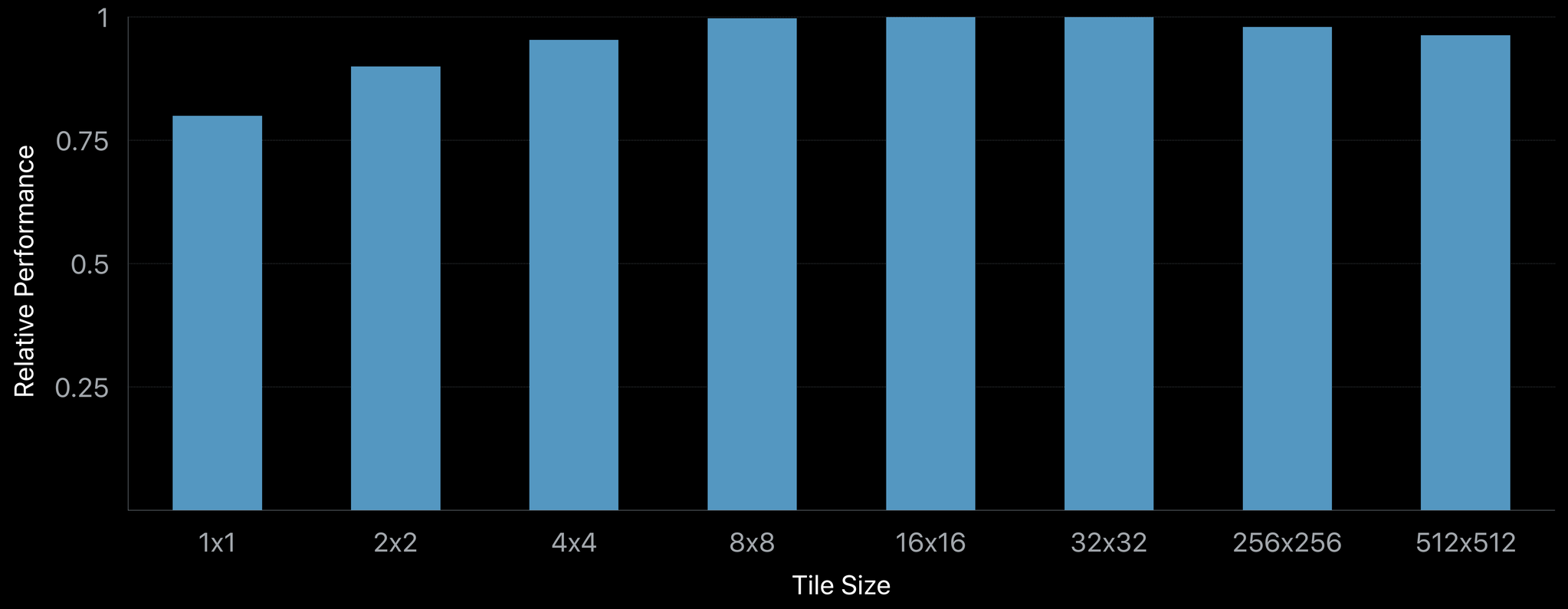


Choosing a Tile Size

How small should I make my tiles?

Experiment: vary tile size, measure performance

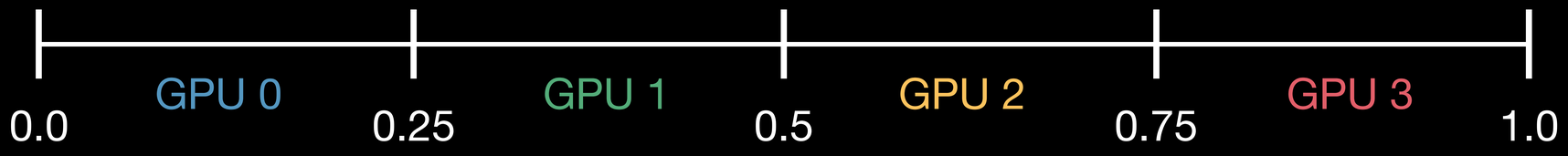
Choosing a Tile Size



Tile Assignment

Assign each tile a random number

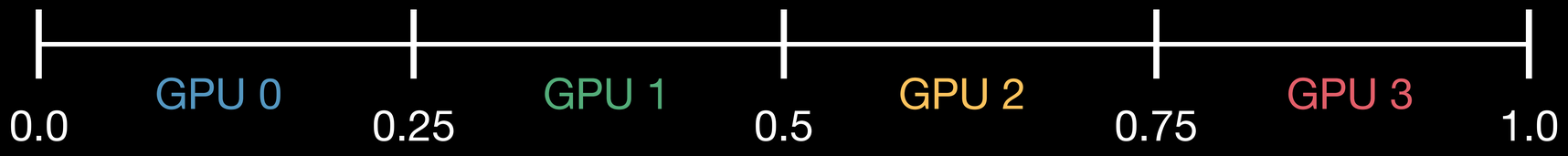
Compare against thresholds to pick GPU



Tile Assignment

Assign each tile a random number

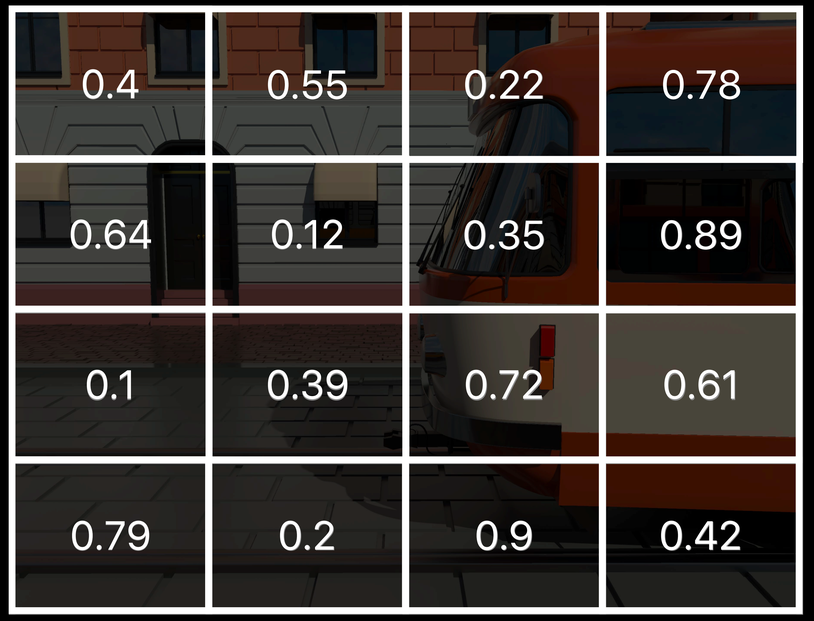
Compare against thresholds to pick GPU

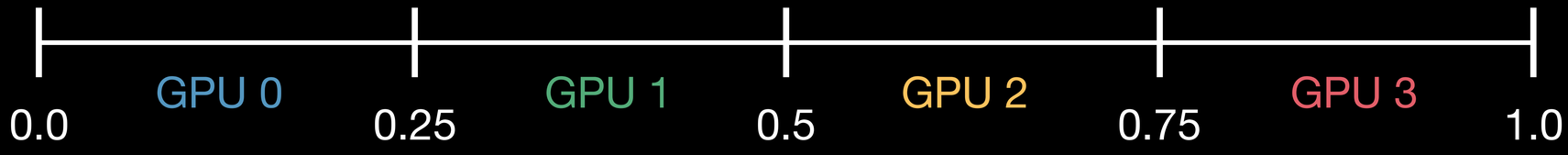


Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

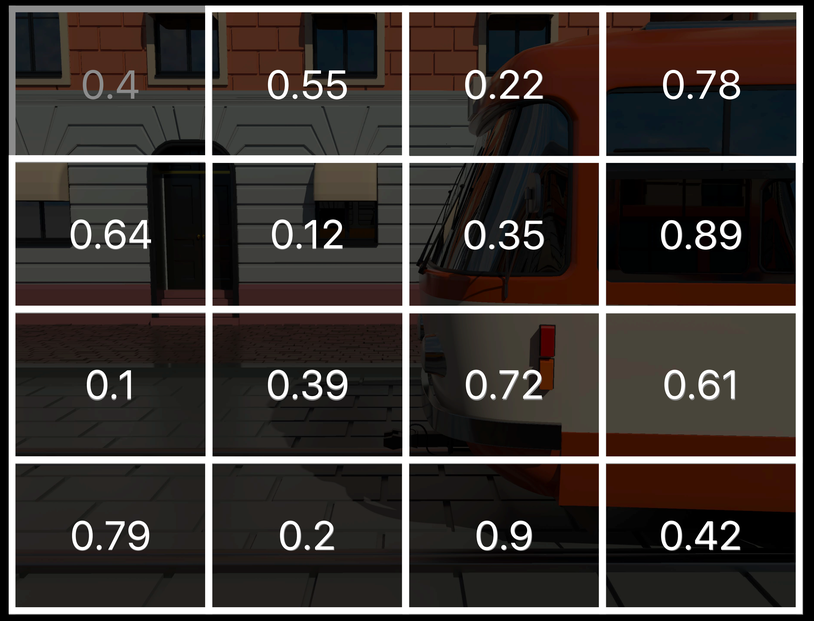


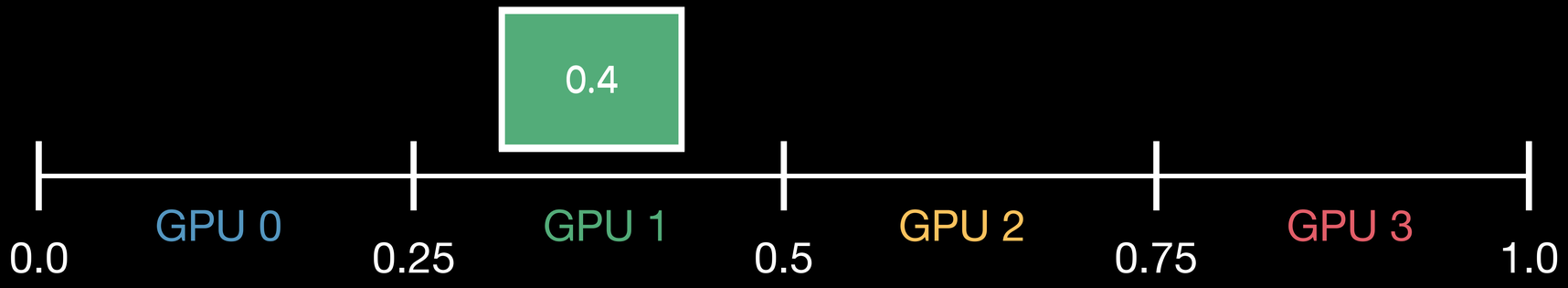


Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

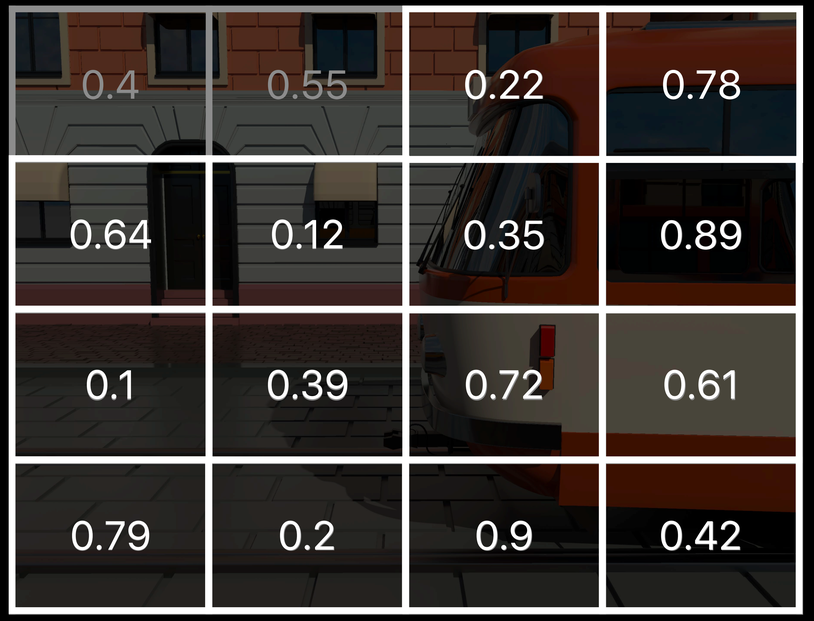


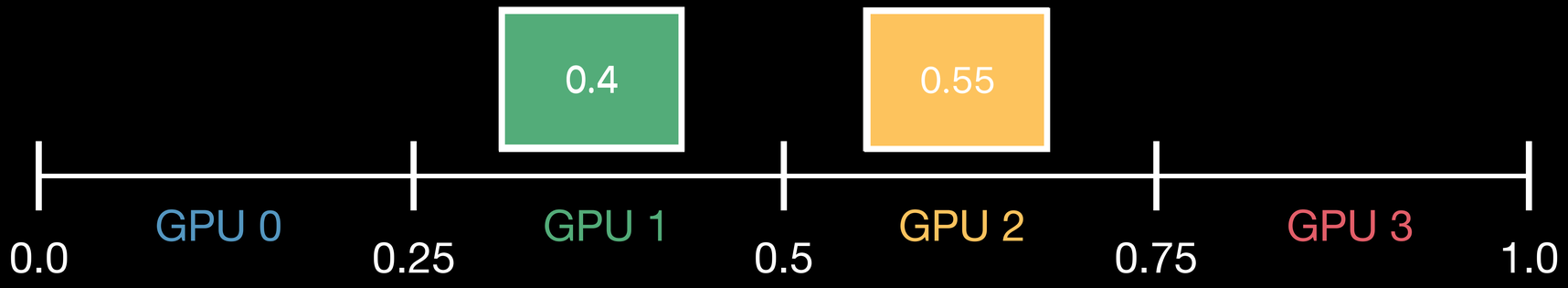


Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

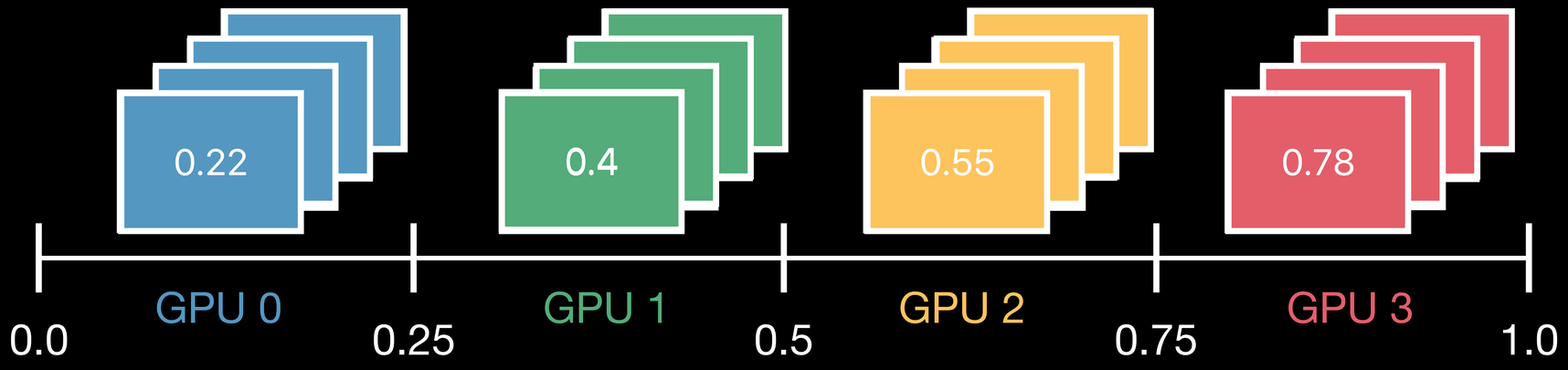




Tile Assignment

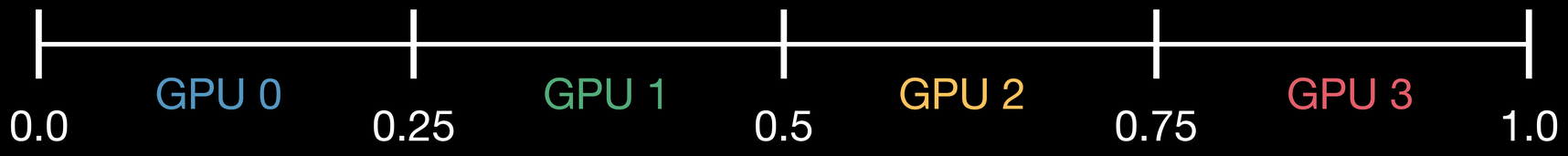
Assign each tile a random number

Compare against thresholds to pick GPU



Tile Assignment

Adjust ranges to control distribution

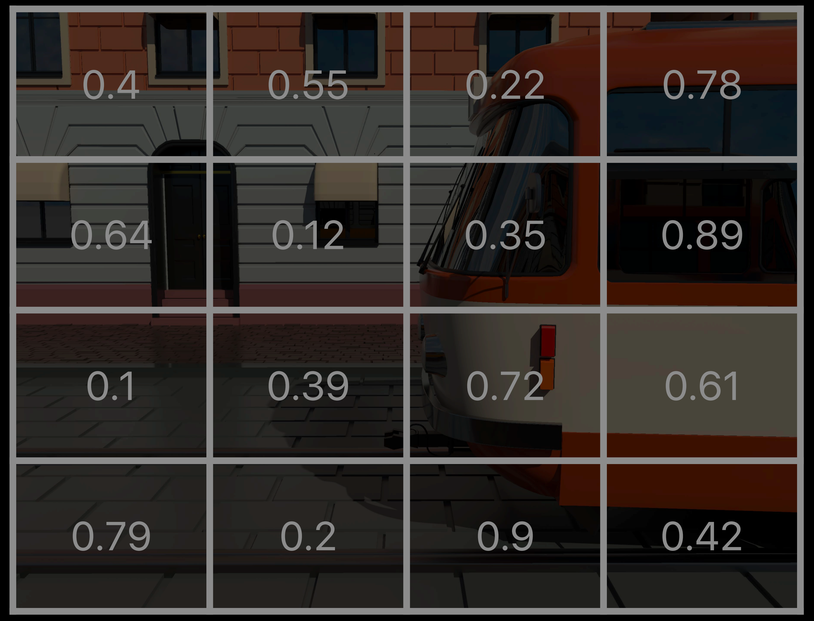


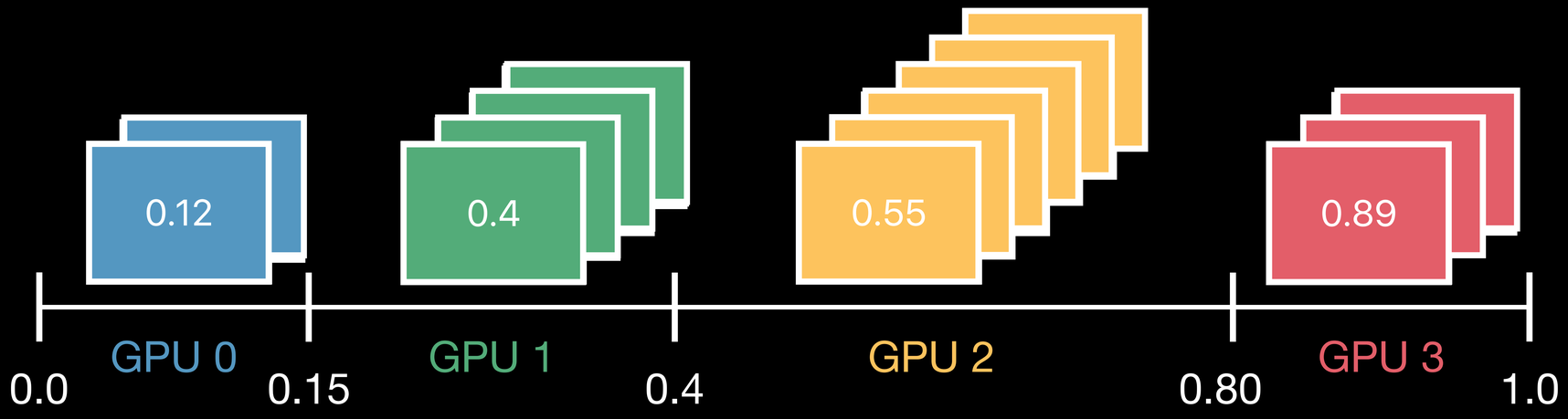
Tile Assignment

Adjust ranges to control distribution

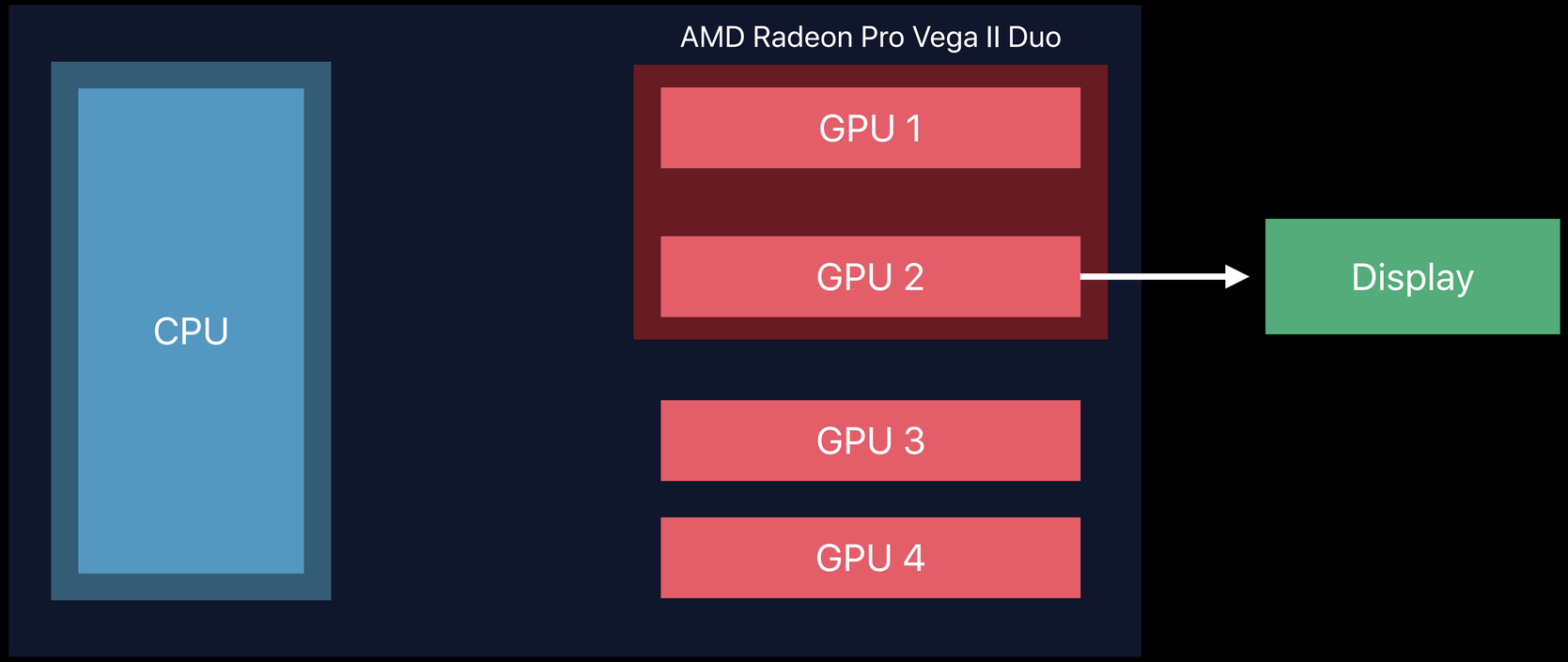
Tile Assignment

Adjust ranges to control distribution

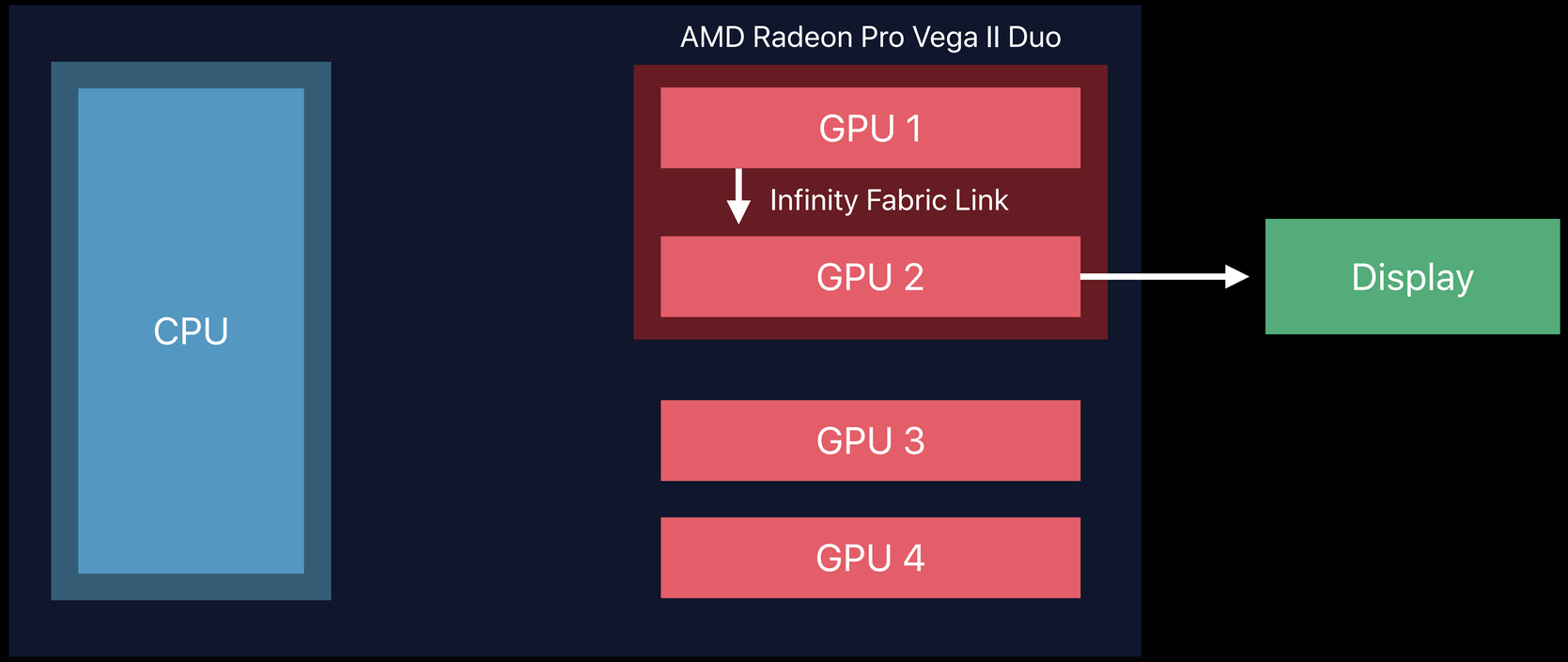




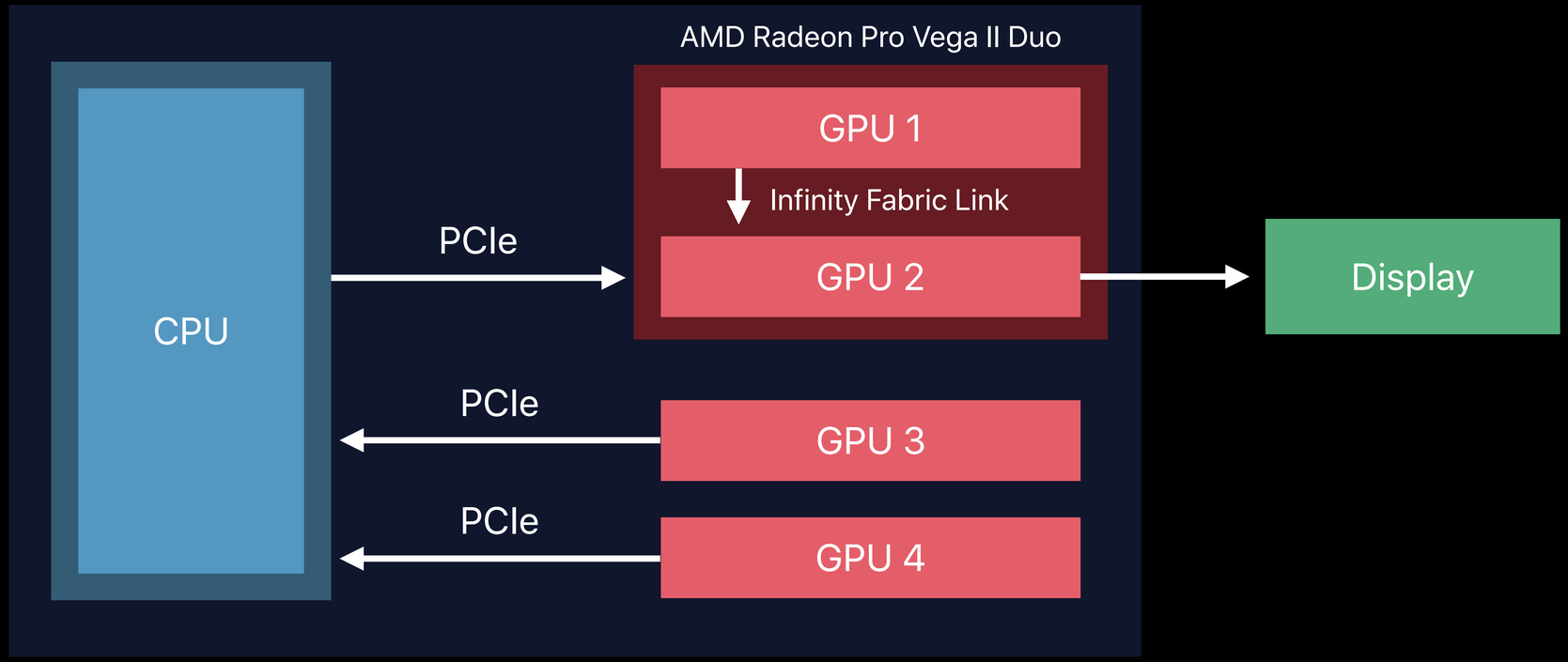
Data Transfers



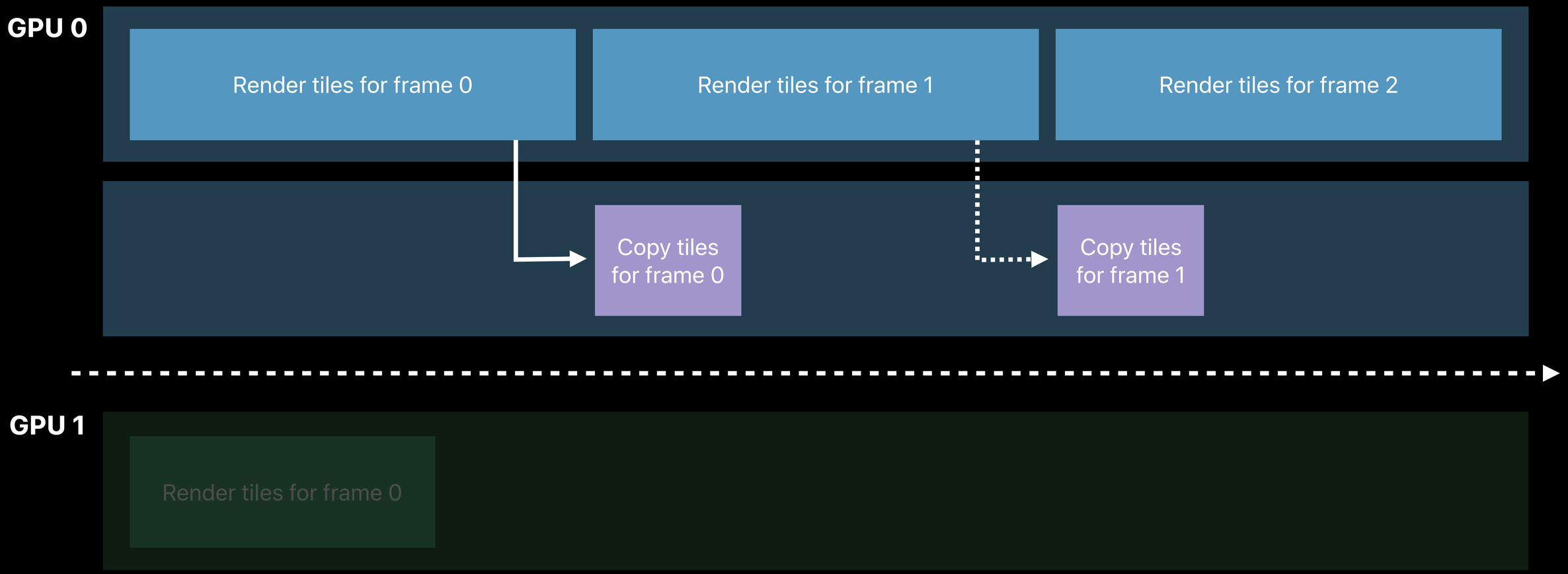
Data Transfers



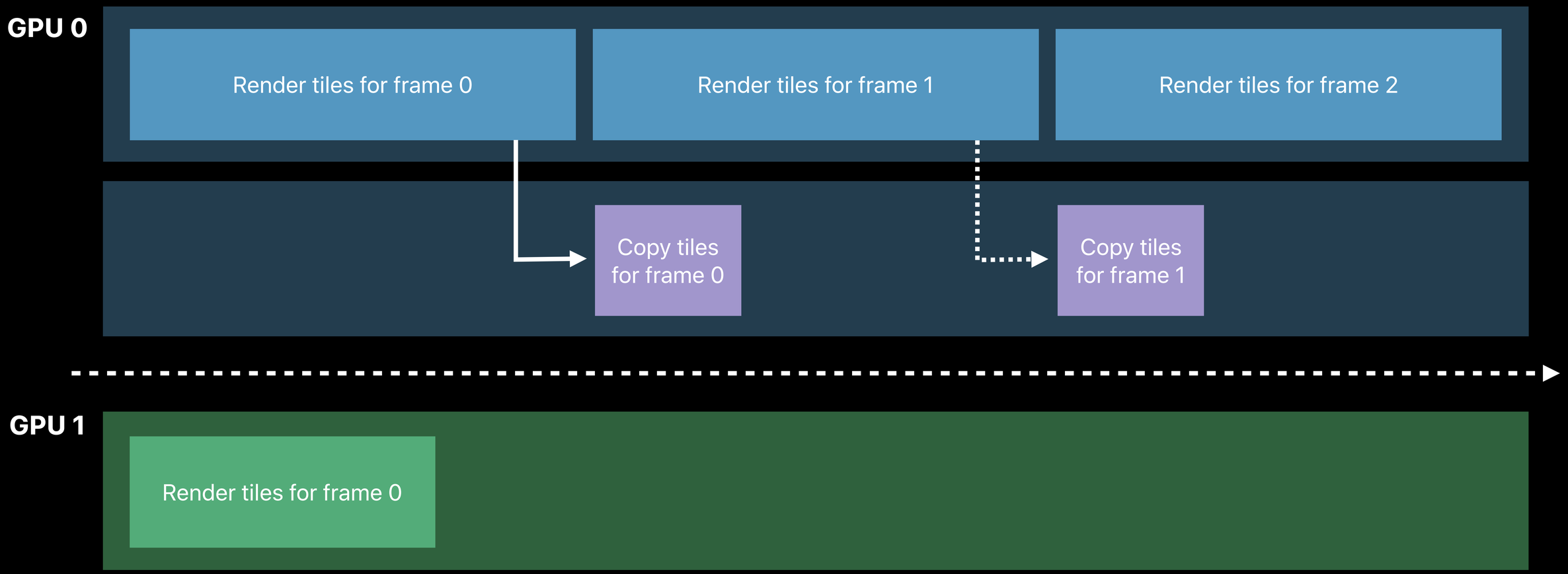
Data Transfers



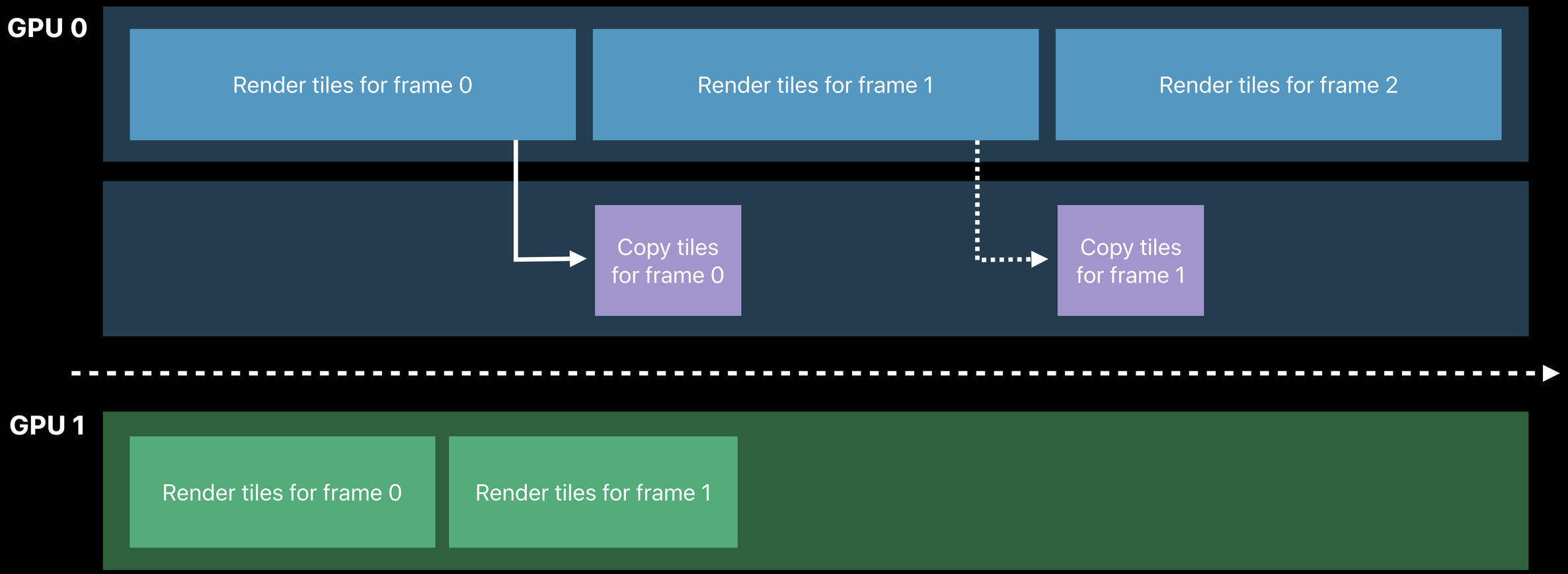
Data Transfers



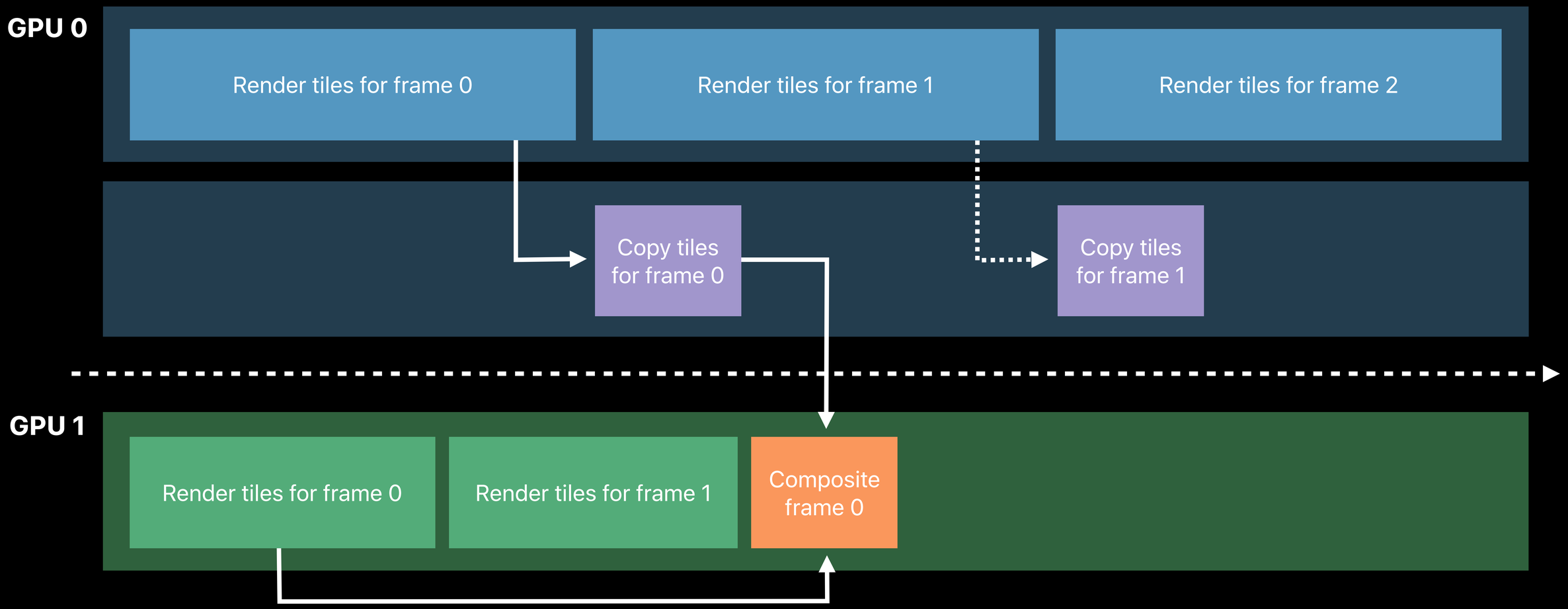
Data Transfers



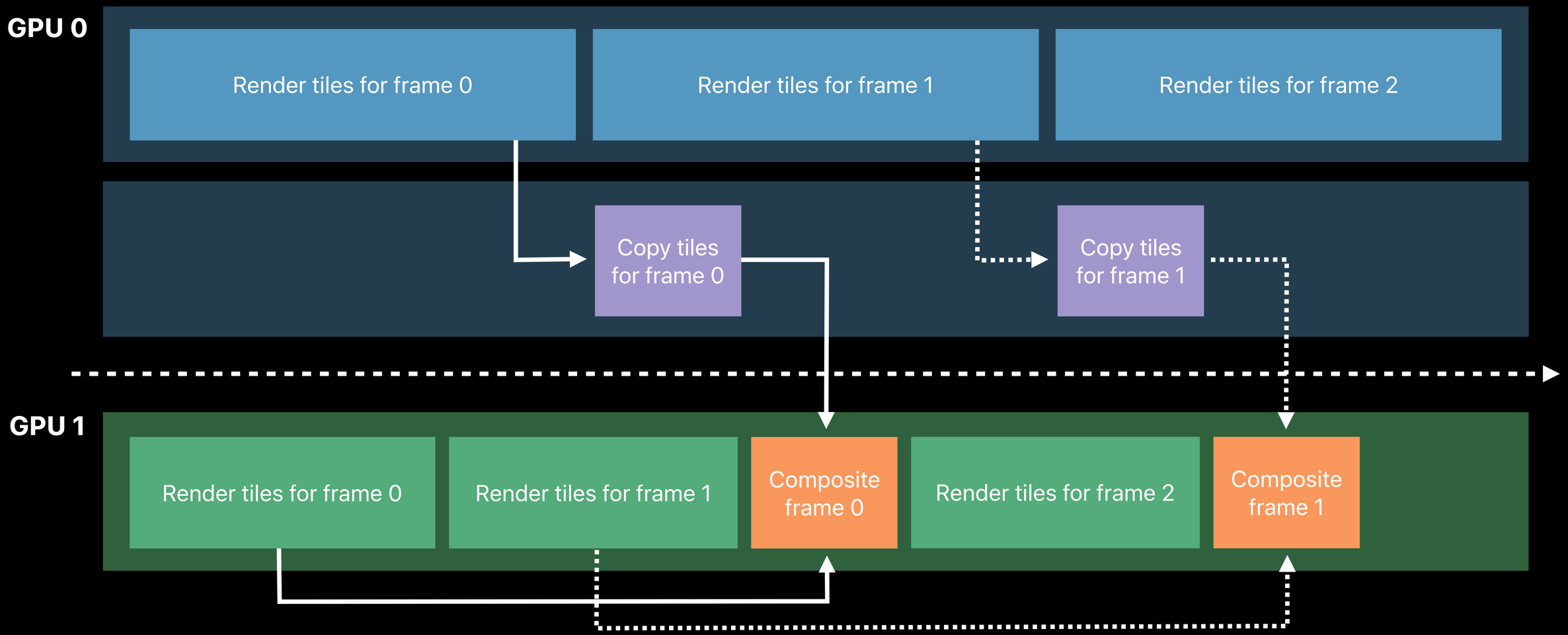
Data Transfers



Data Transfers



Data Transfers



Summary

Ray/triangle intersection

Dynamic scenes

MPSSVGFDenoiser

Use cases

Multi-GPU

More Information

developer.apple.com/wwdc19/613

Metal for Machine Learning and Ray Tracing Lab	Friday, 12:00
Metal for Machine Learning Session	Friday, 3:20


WWDC19