

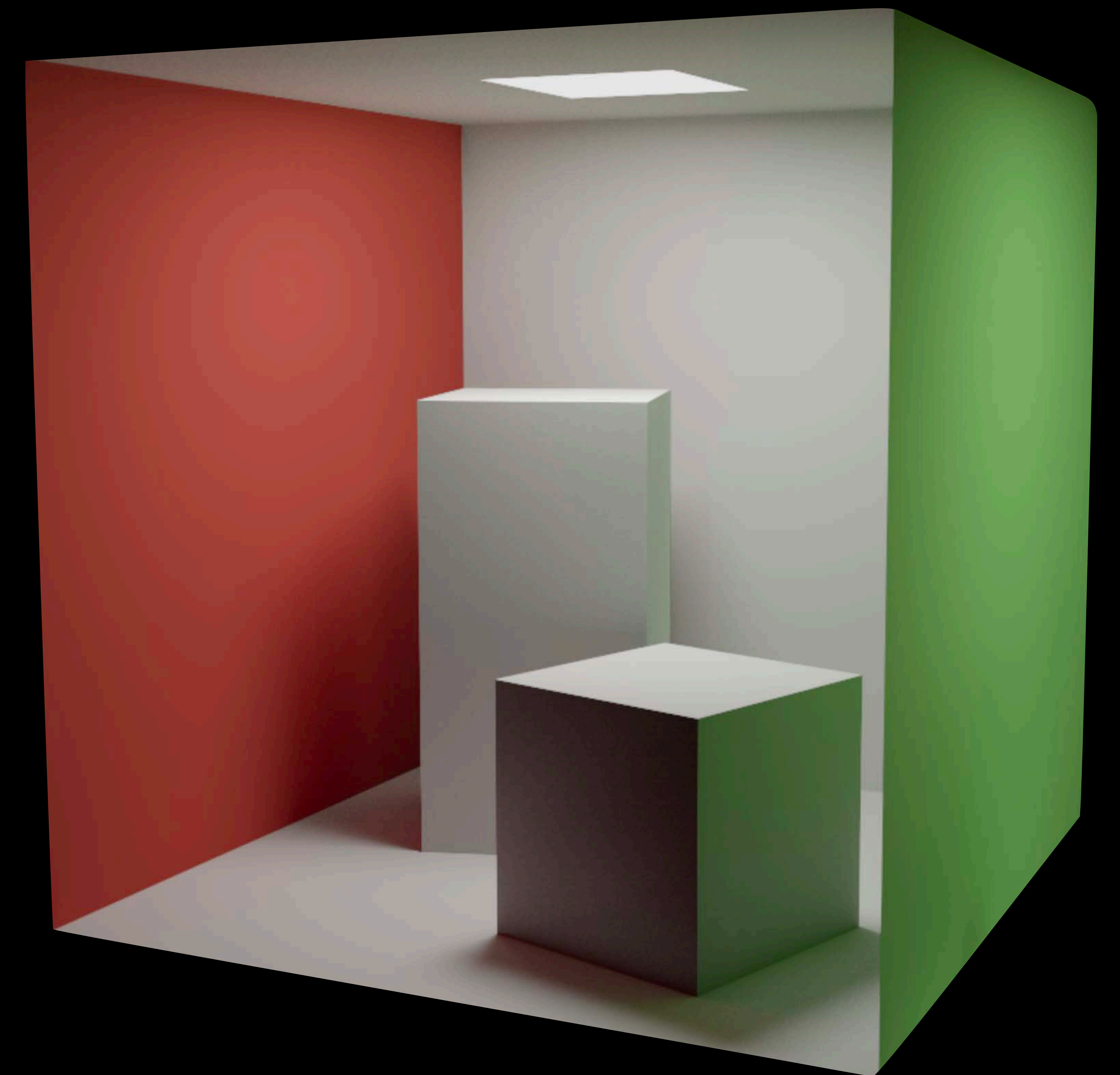
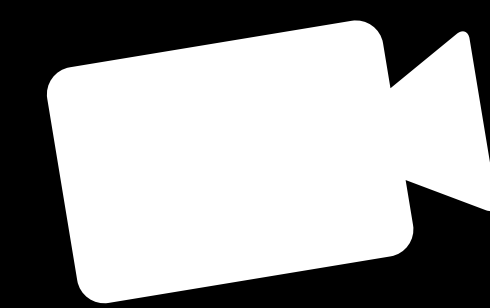
#WWDC19

Ray Tracing with Metal

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

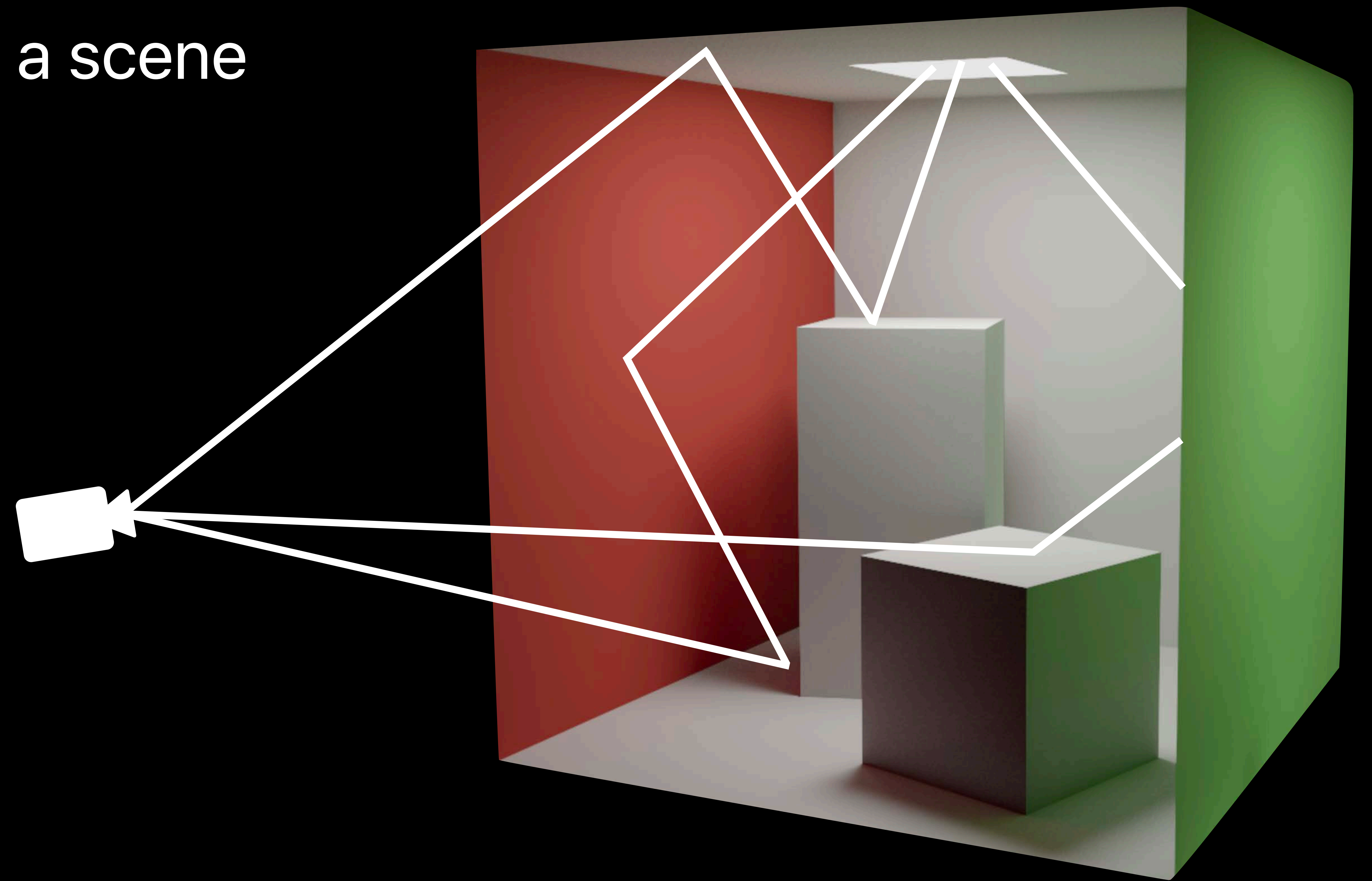
Ray Tracing

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



Ray Tracing

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

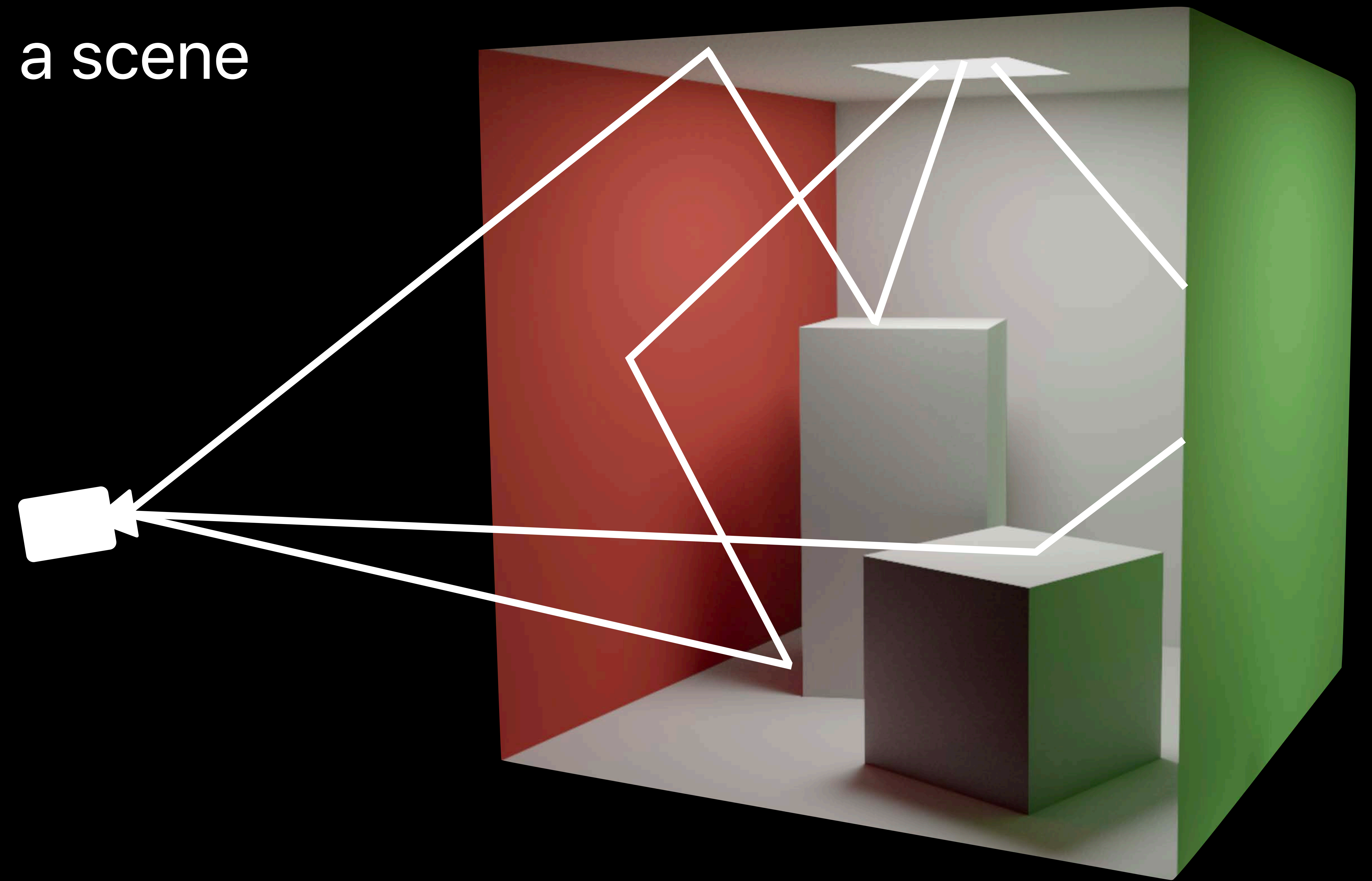


Ray Tracing

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

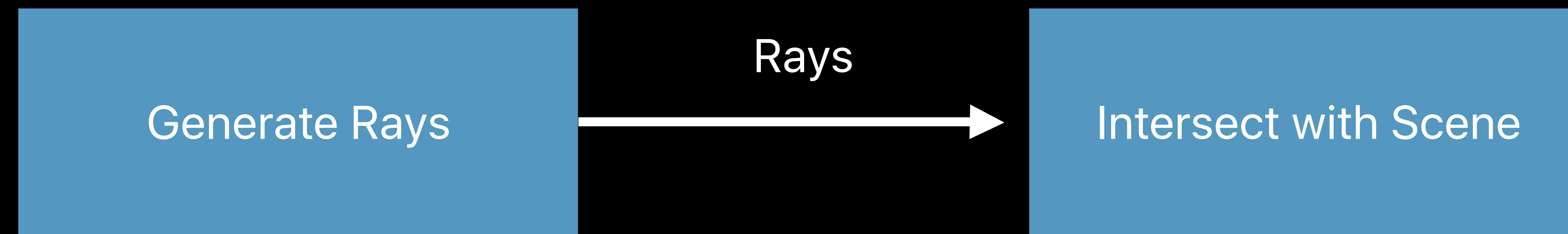
Denoising

Ray Tracing with Metal

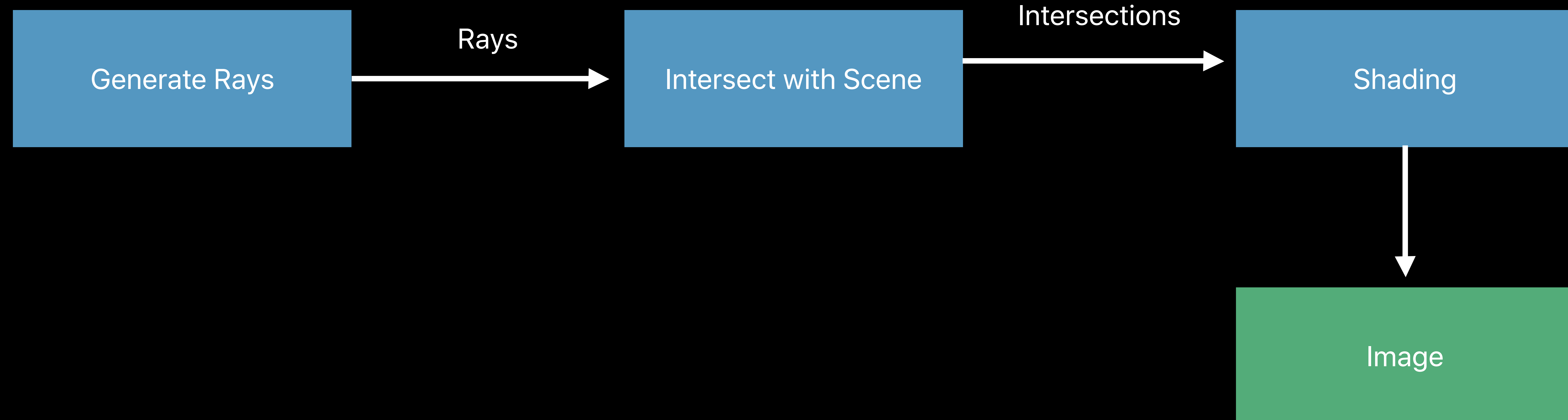
Ray Tracing with Metal

Generate Rays

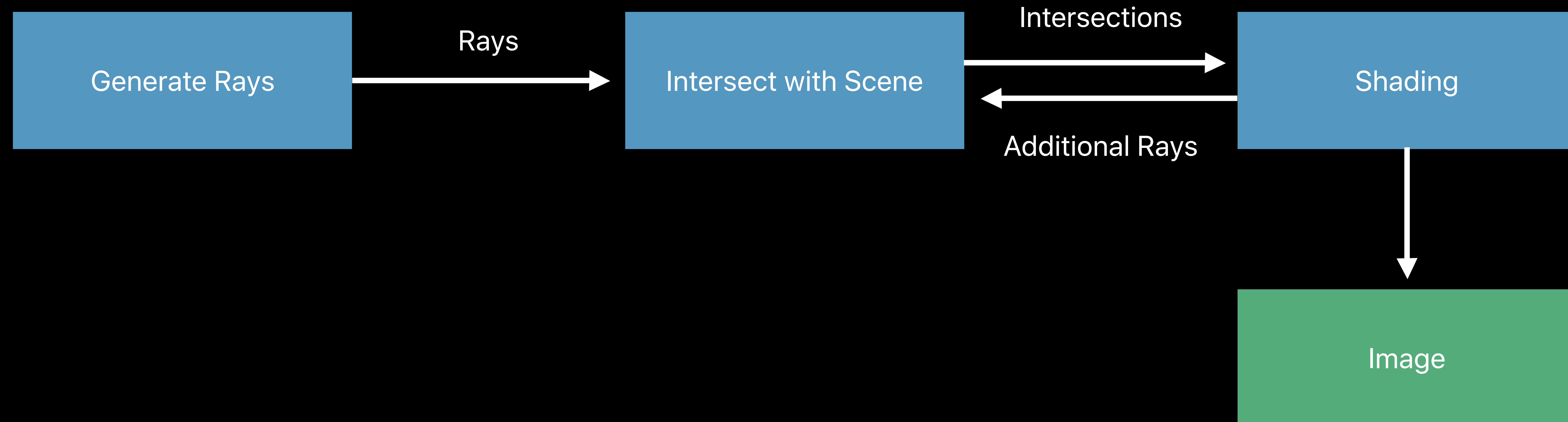
Ray Tracing with Metal



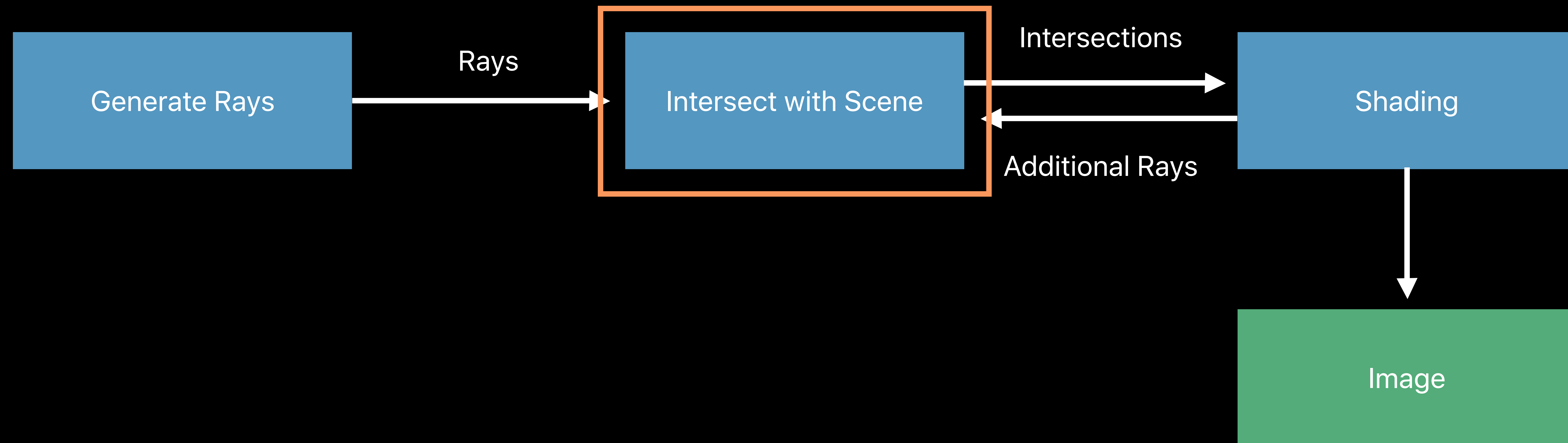
Ray Tracing with Metal



Ray Tracing with Metal



Ray Tracing with Metal



MPSRayIntersector

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



Intersector

MPSRayIntersector

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

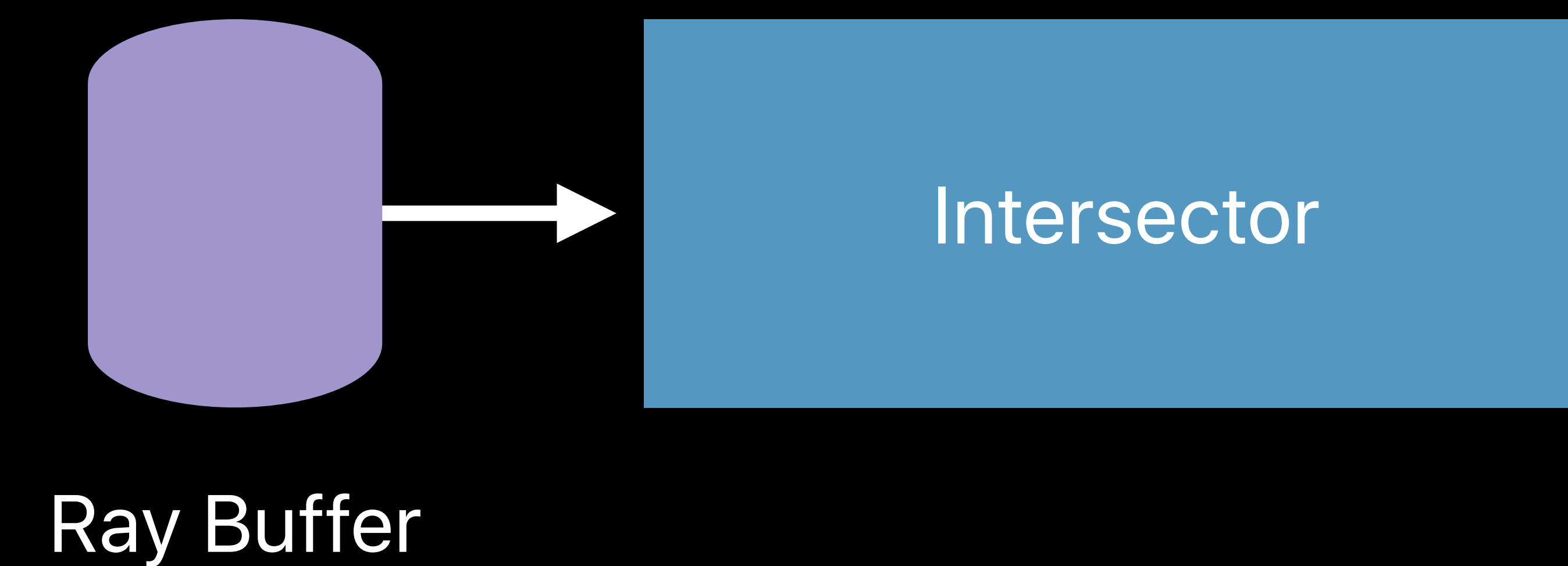


Intersector

MPSRayIntersector

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

Accepts batches of rays in a Metal buffer

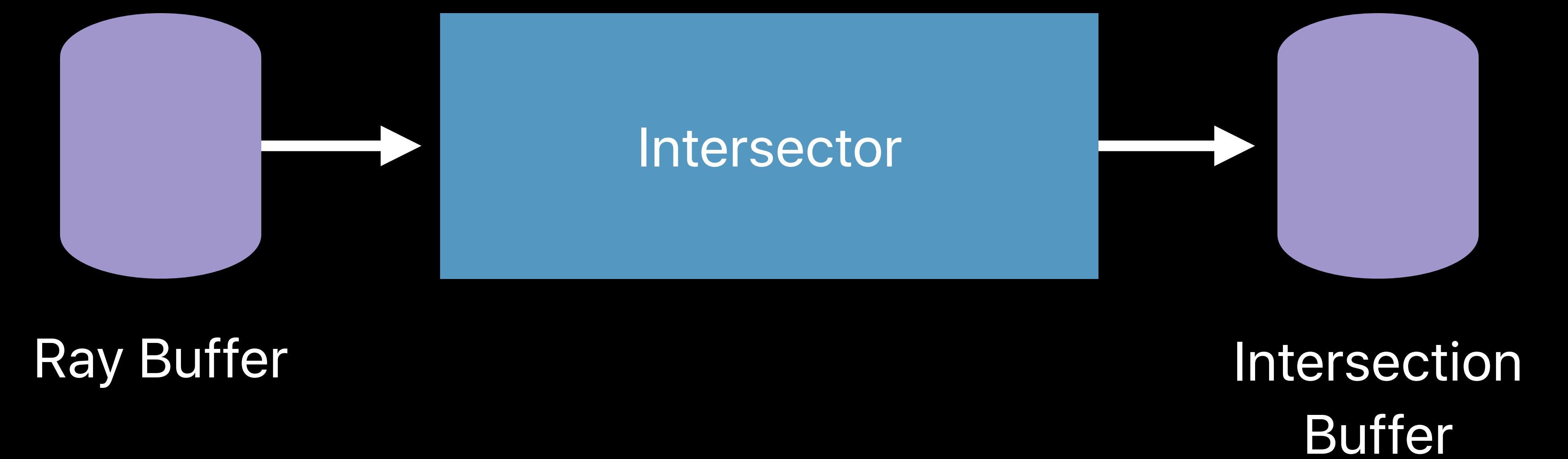


MPSRayIntersector

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

Accepts batches of rays in a Metal buffer

Returns one intersection per ray



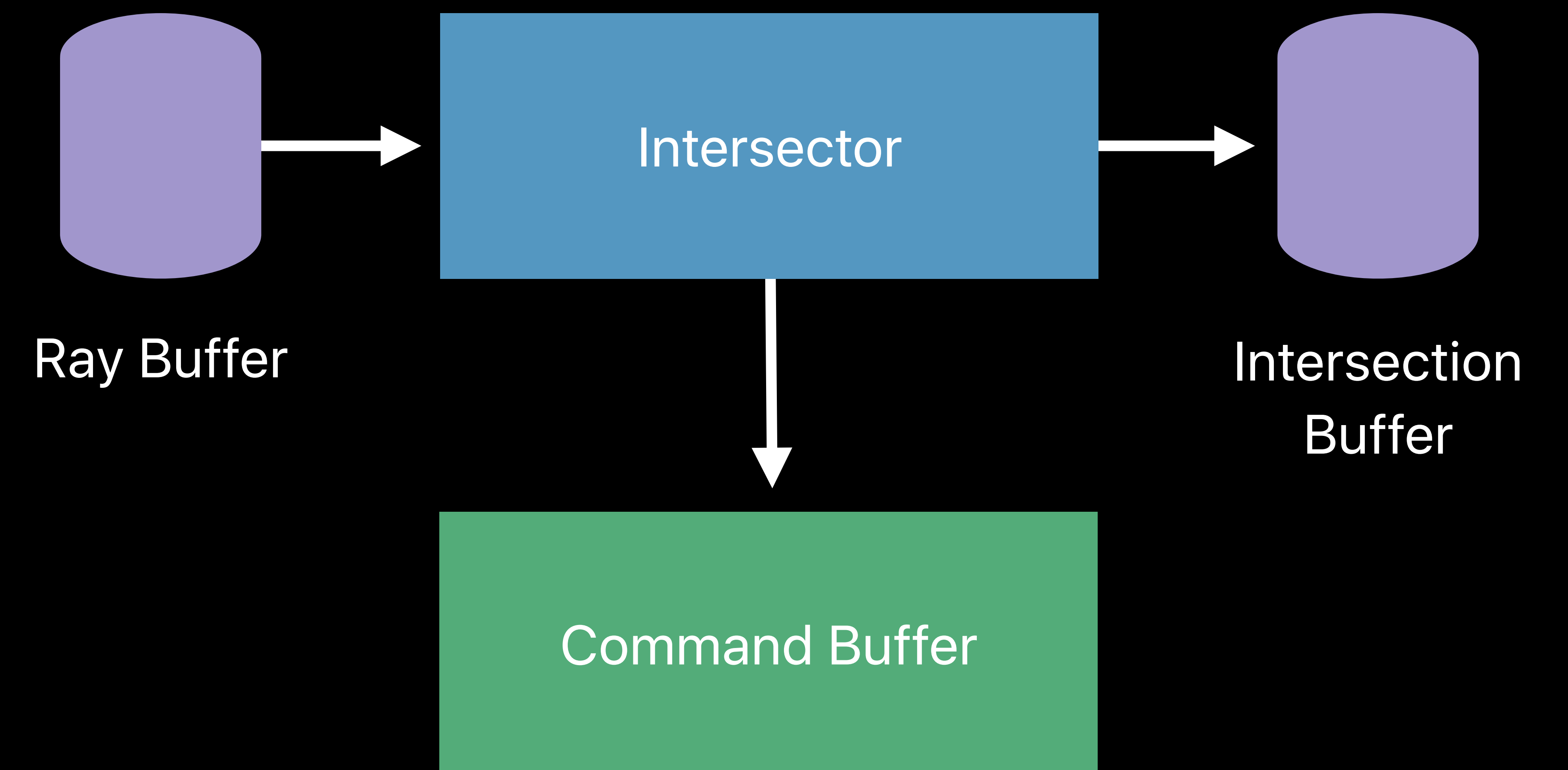
MPSRayIntersector

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

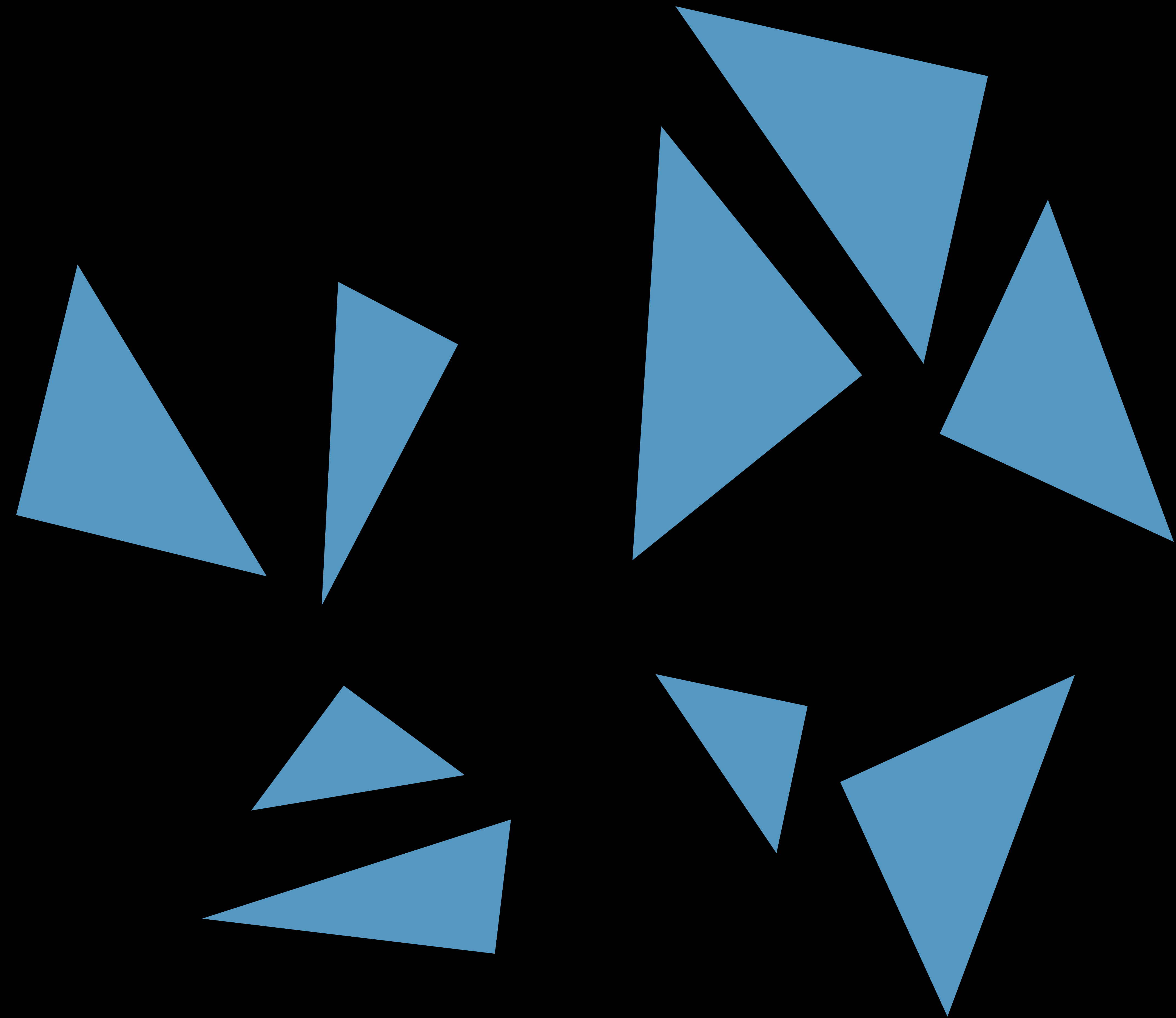
Accepts batches of rays in a Metal buffer

Returns one intersection per ray

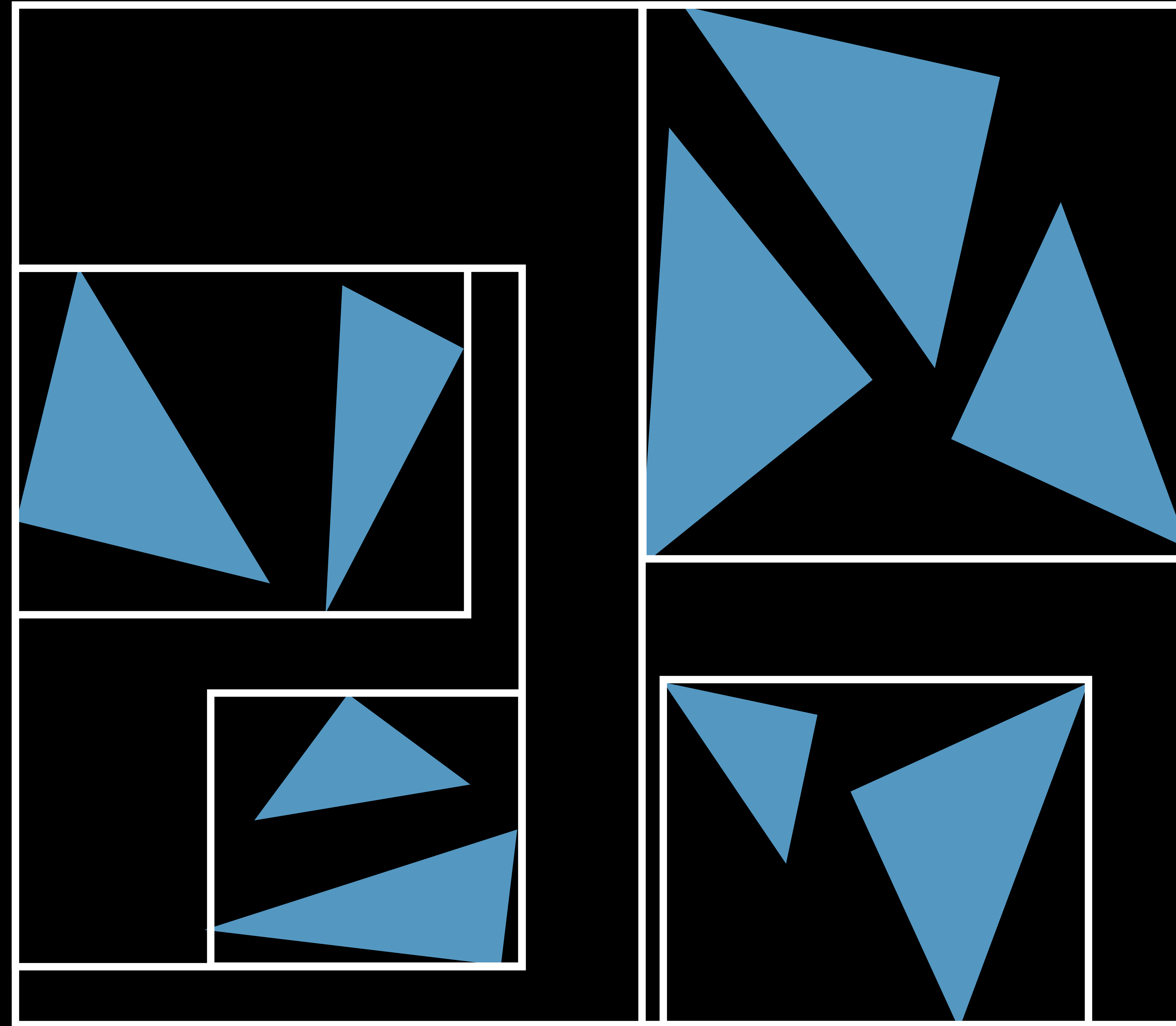
Encodes into a Metal command buffer



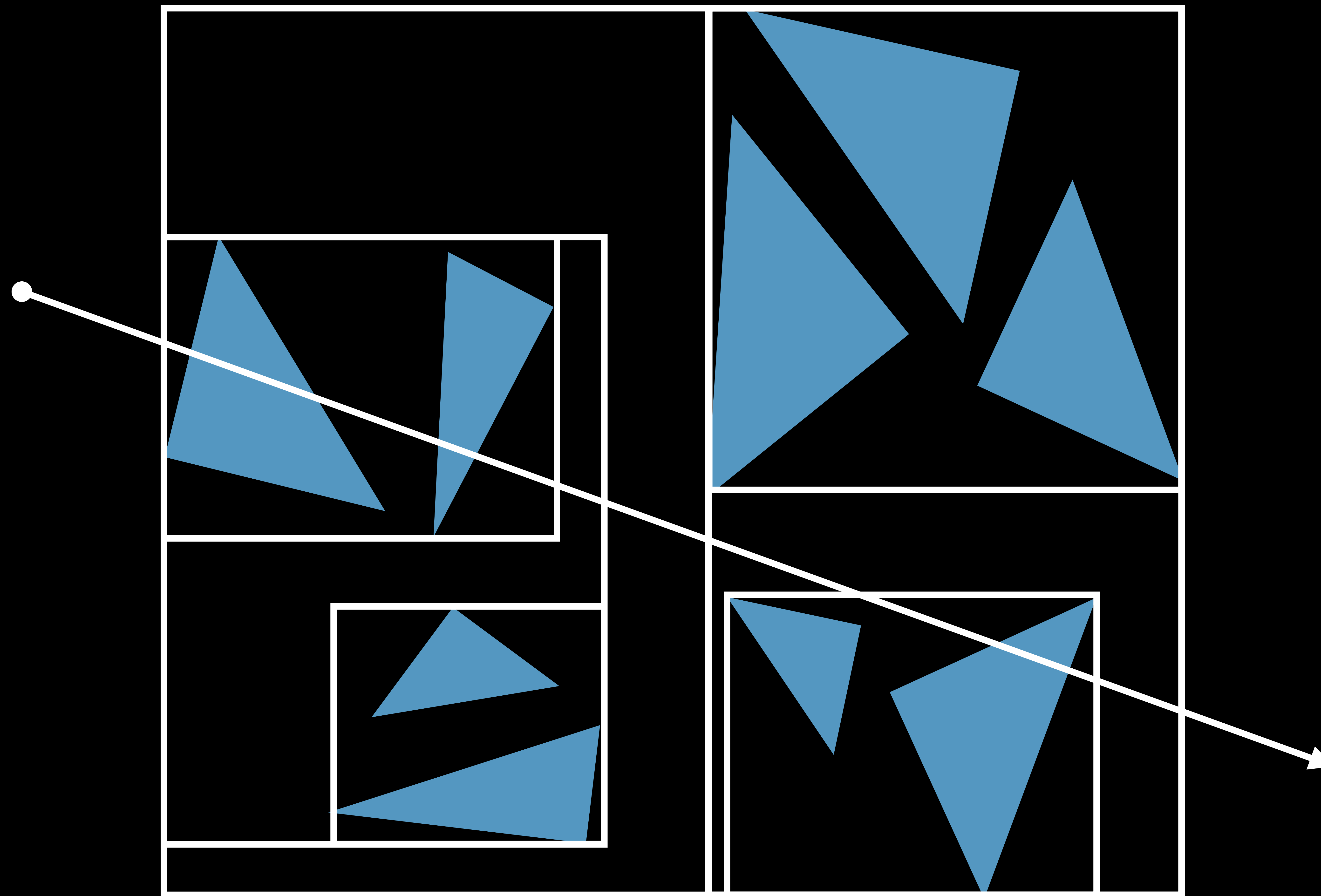
MPSRayIntersector



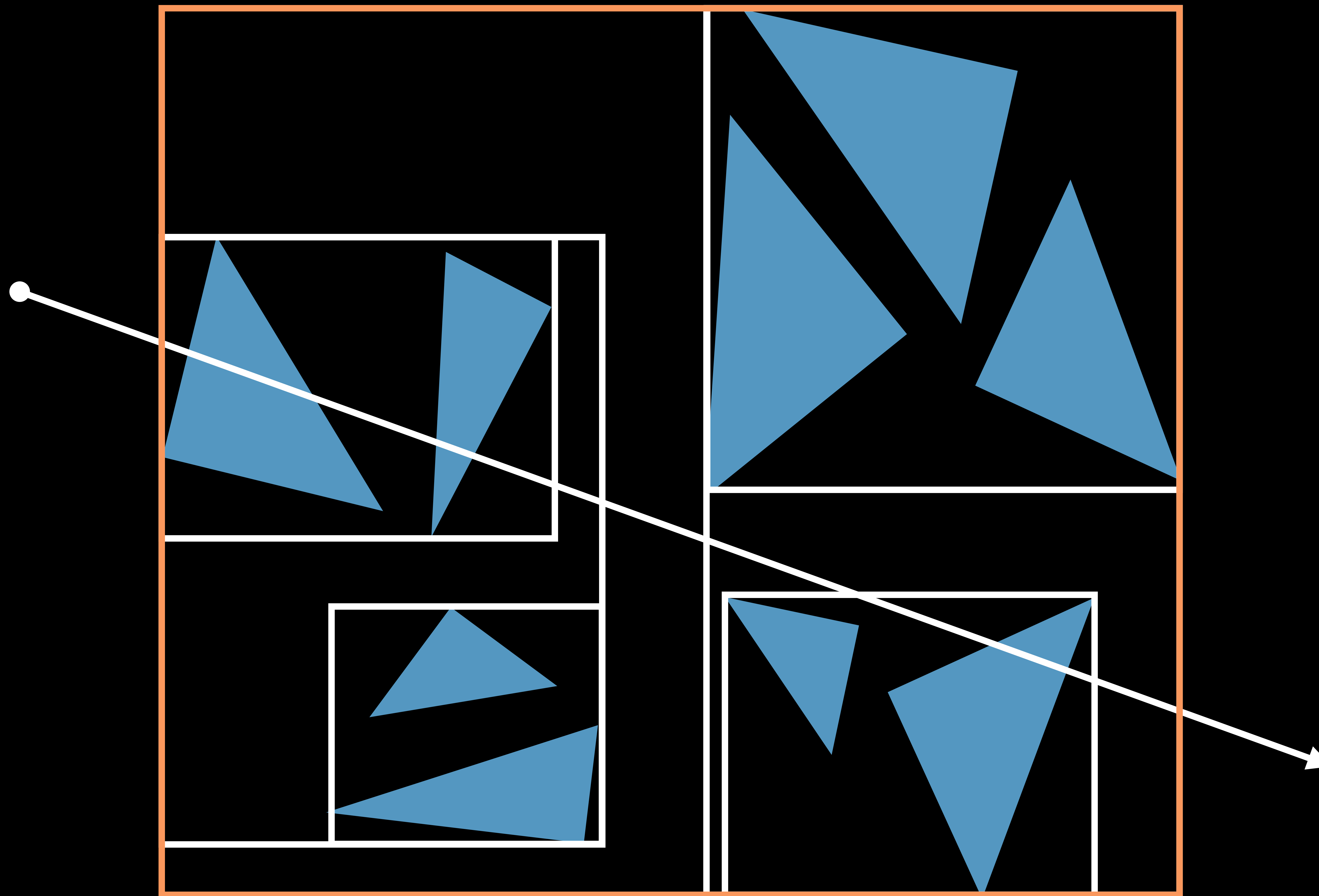
MPSRayIntersector



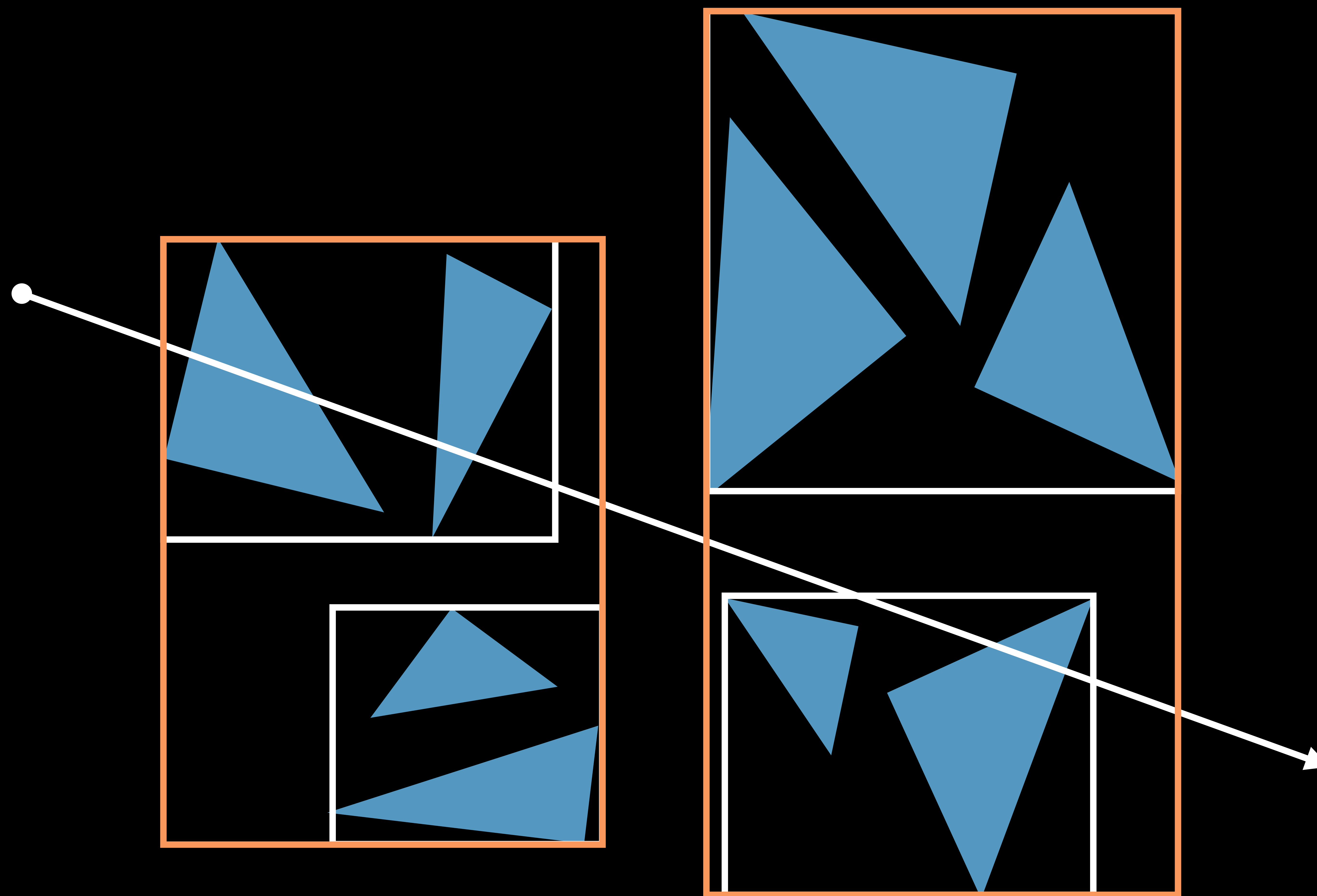
MPSRayIntersector



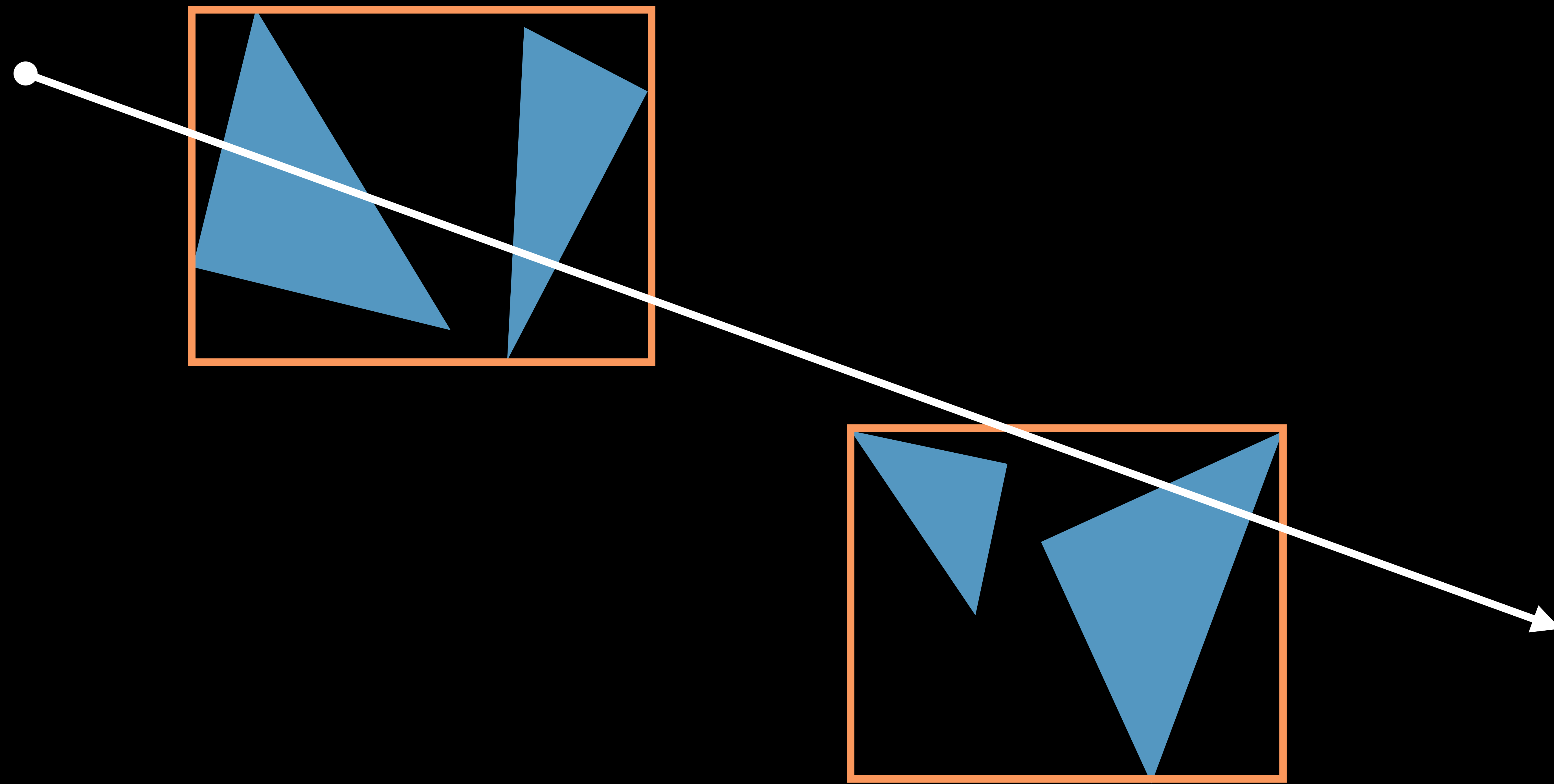
MPSRayIntersector



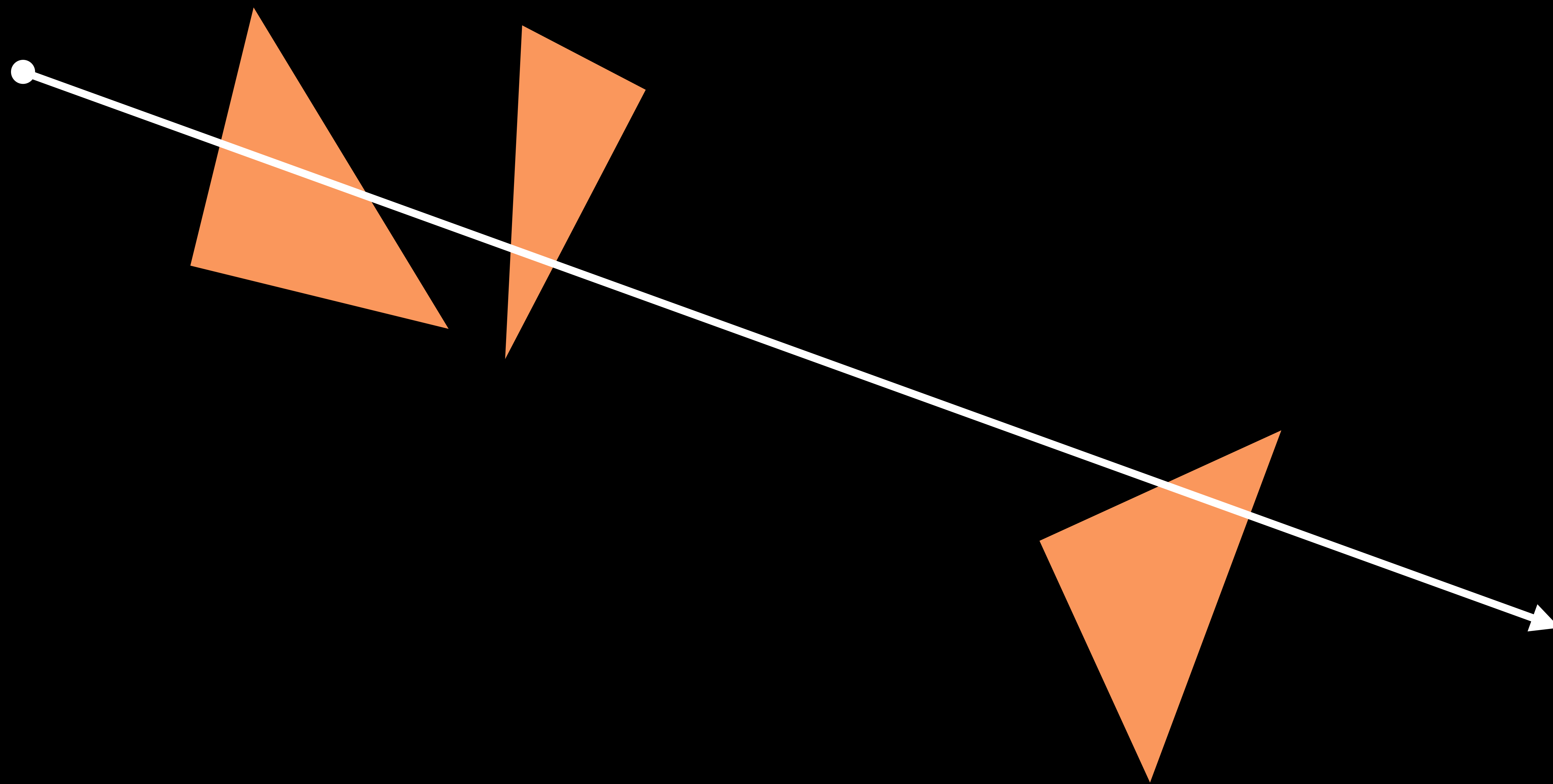
MPSRayIntersector



MPSRayIntersector

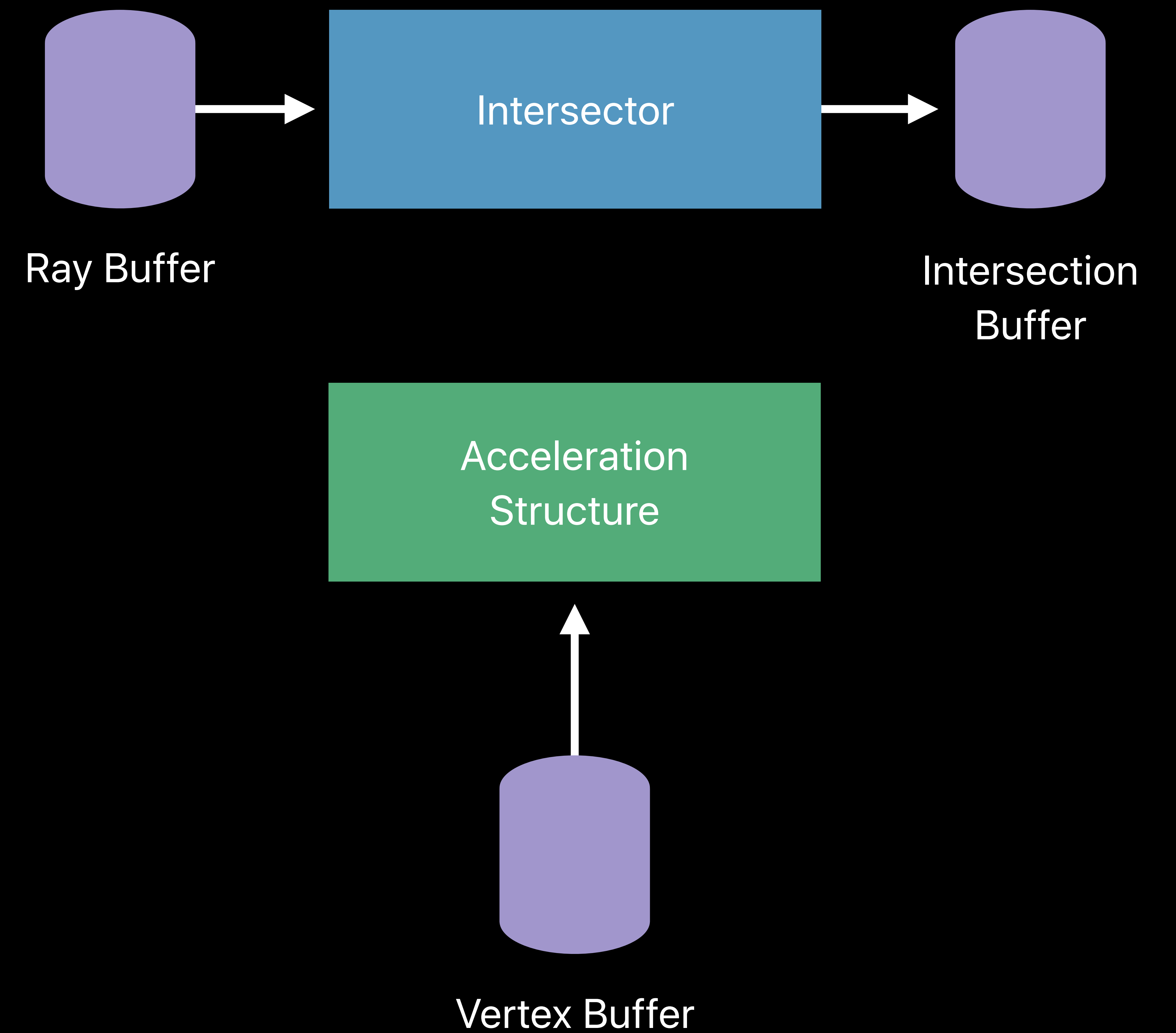


MPSRayIntersector



MPSRayIntersector

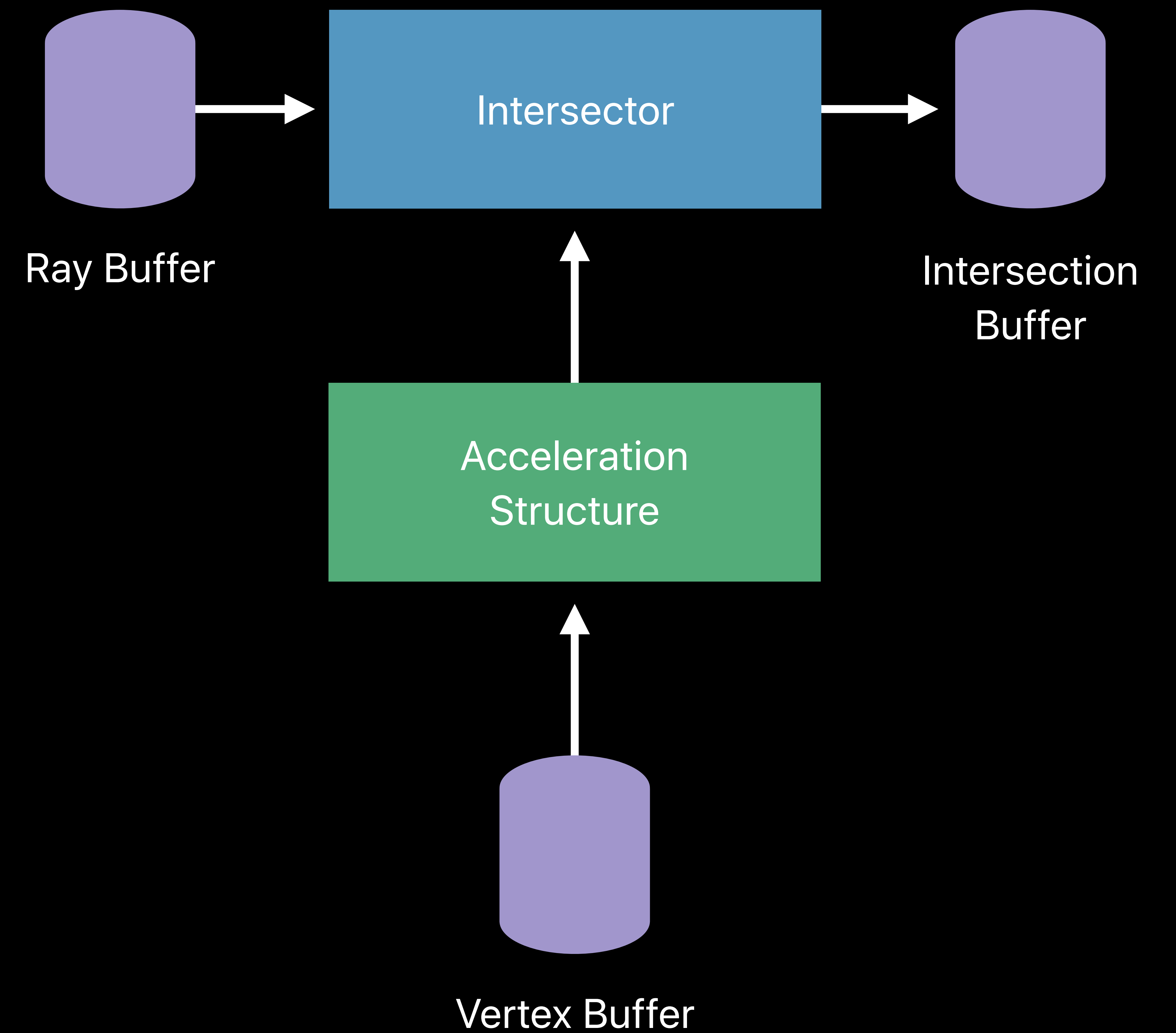
Build an **acceleration structure** over triangles in a vertex buffer



MPSRayIntersector

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

Pass acceleration structure to intersector

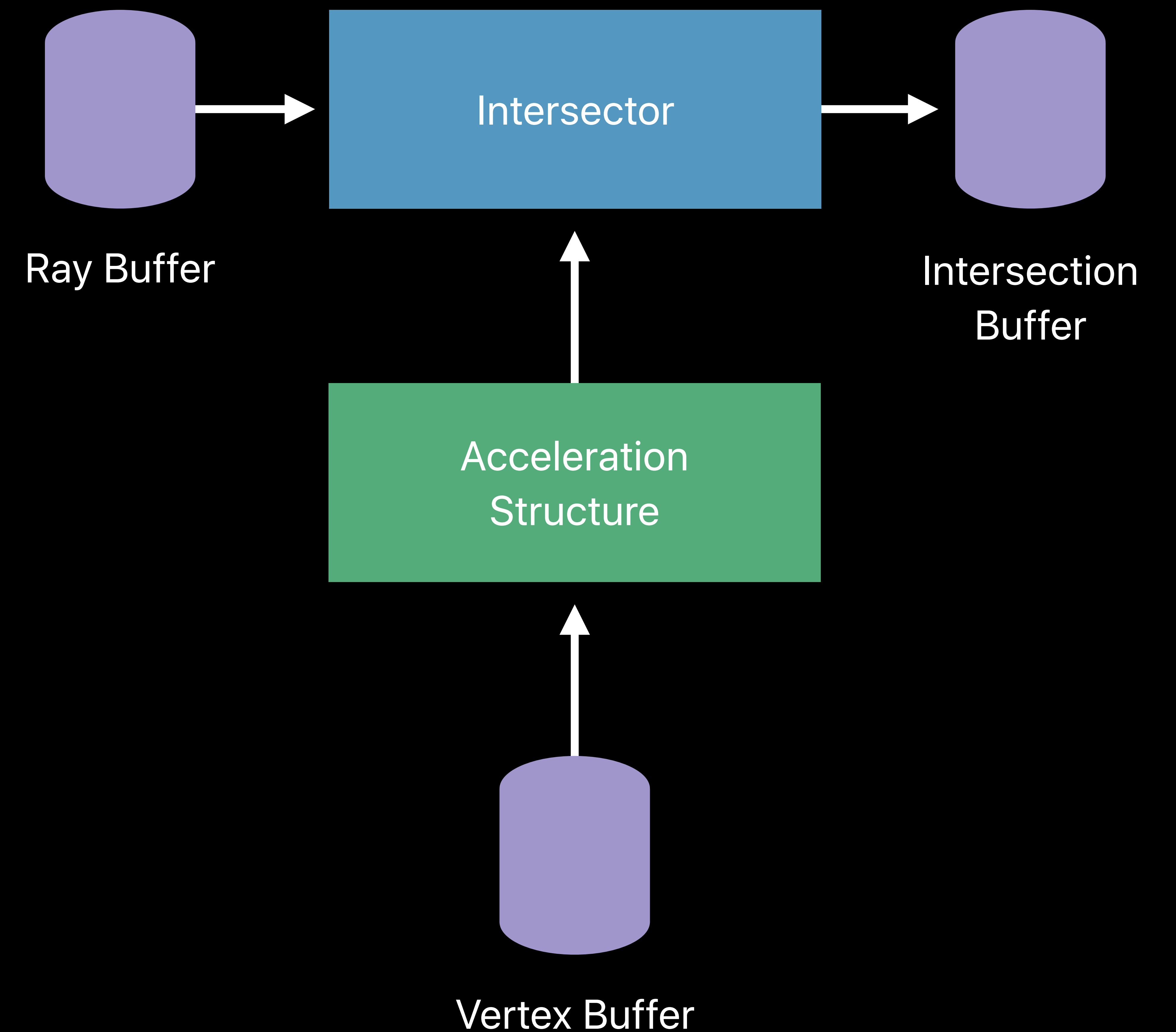


MPSRayIntersector

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

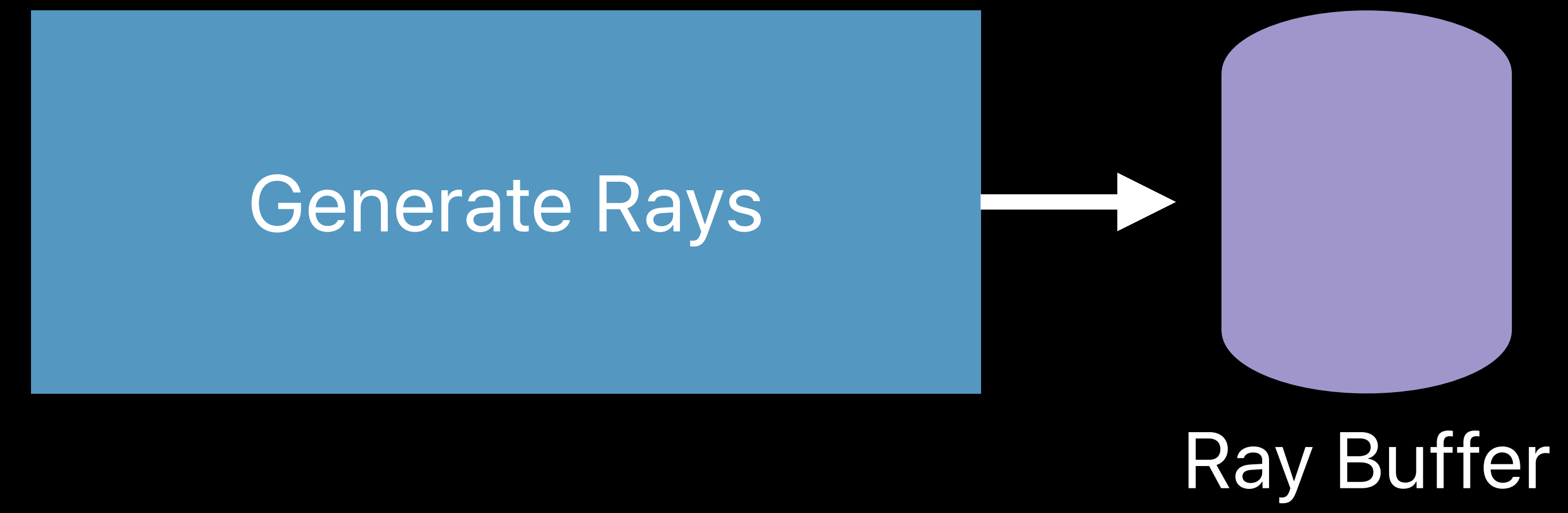
Pass acceleration structure to intersector

Now builds on the GPU

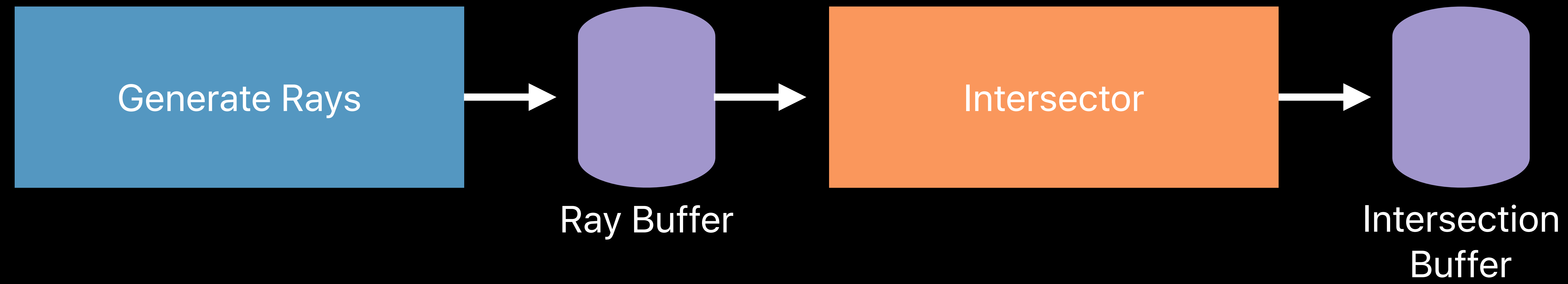


Ray Tracing with Metal

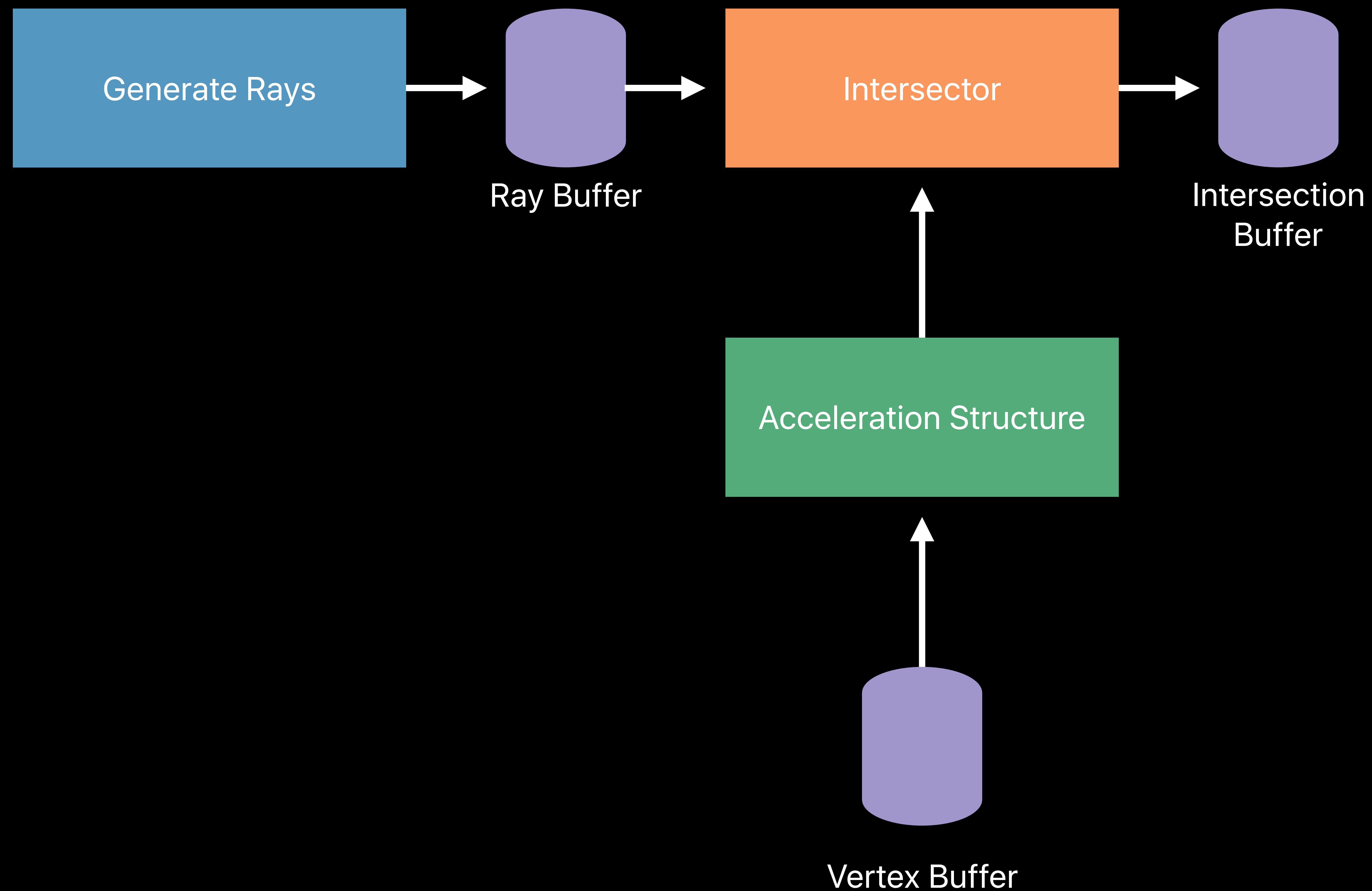
Ray Tracing with Metal



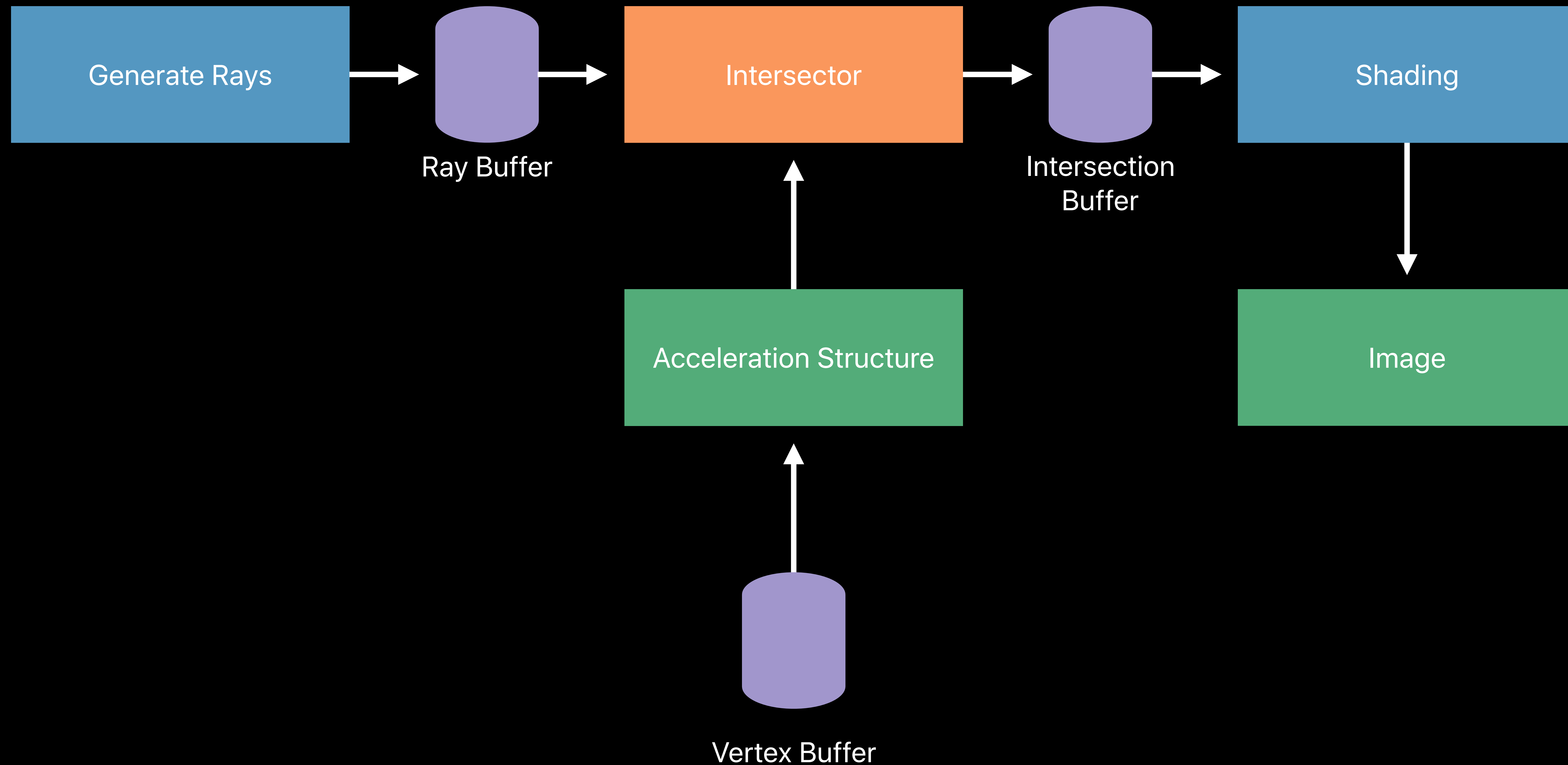
Ray Tracing with Metal



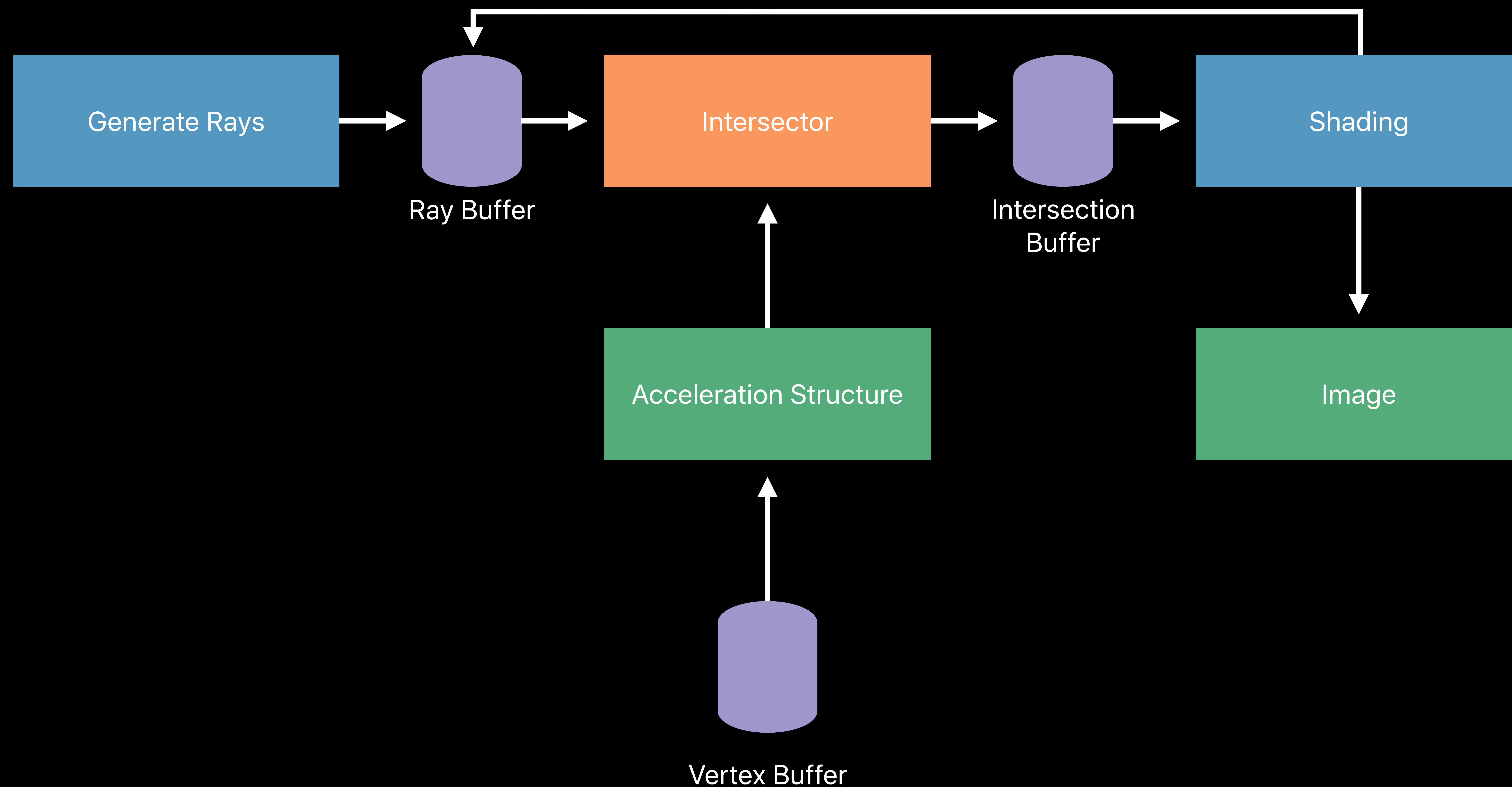
Ray Tracing with Metal



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



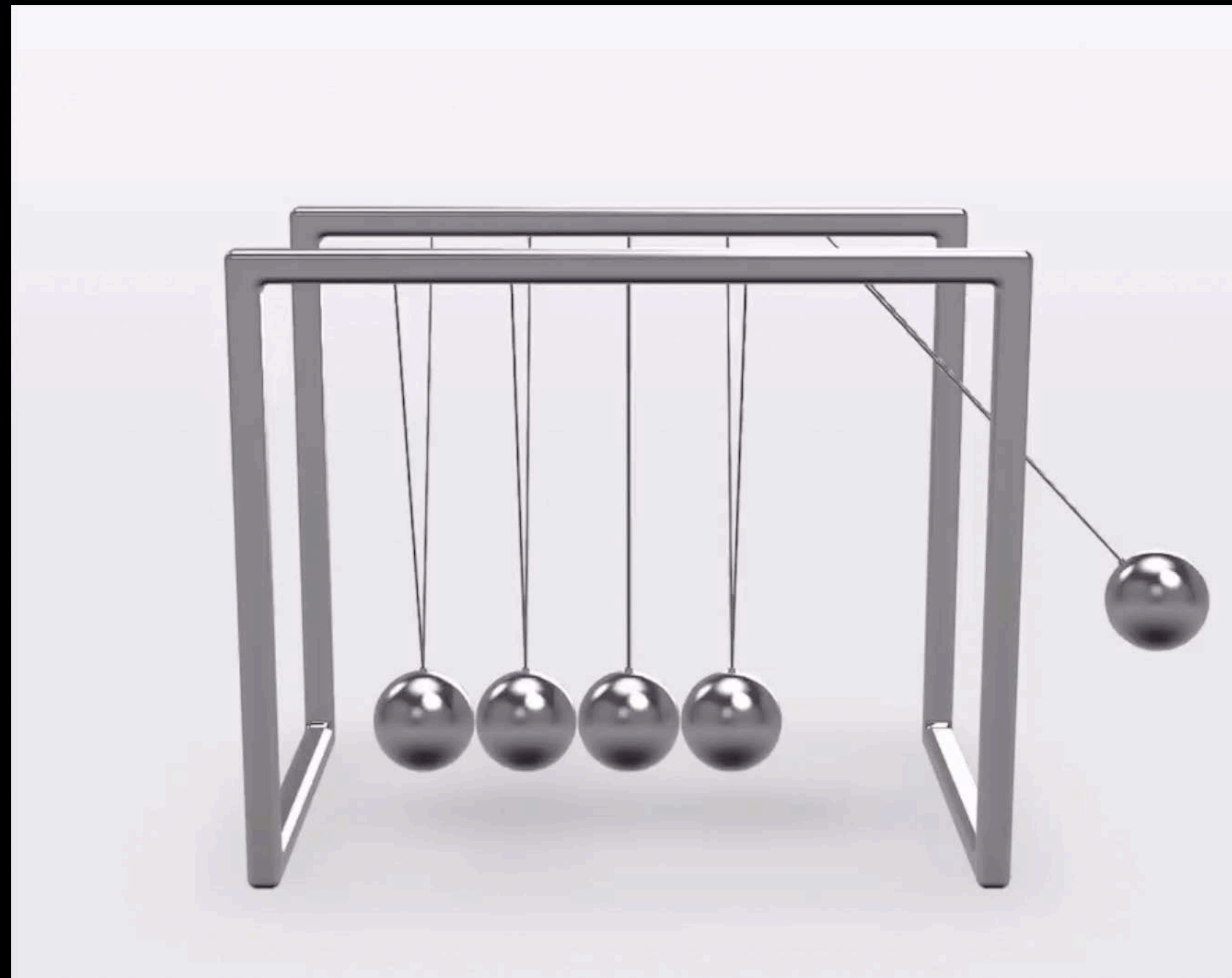
Ray Tracing in AR Quick Look



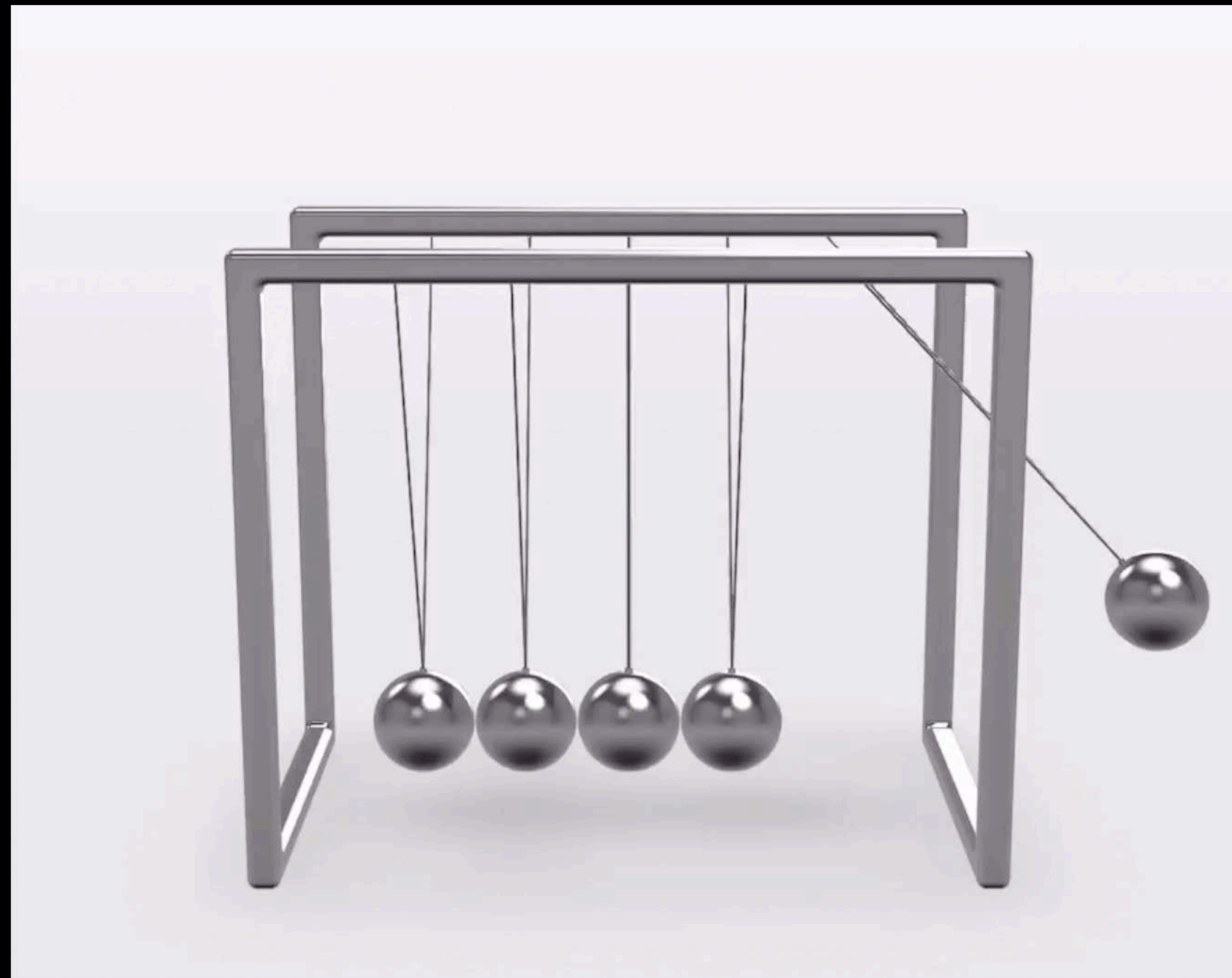
Ray Tracing in AR Quick Look



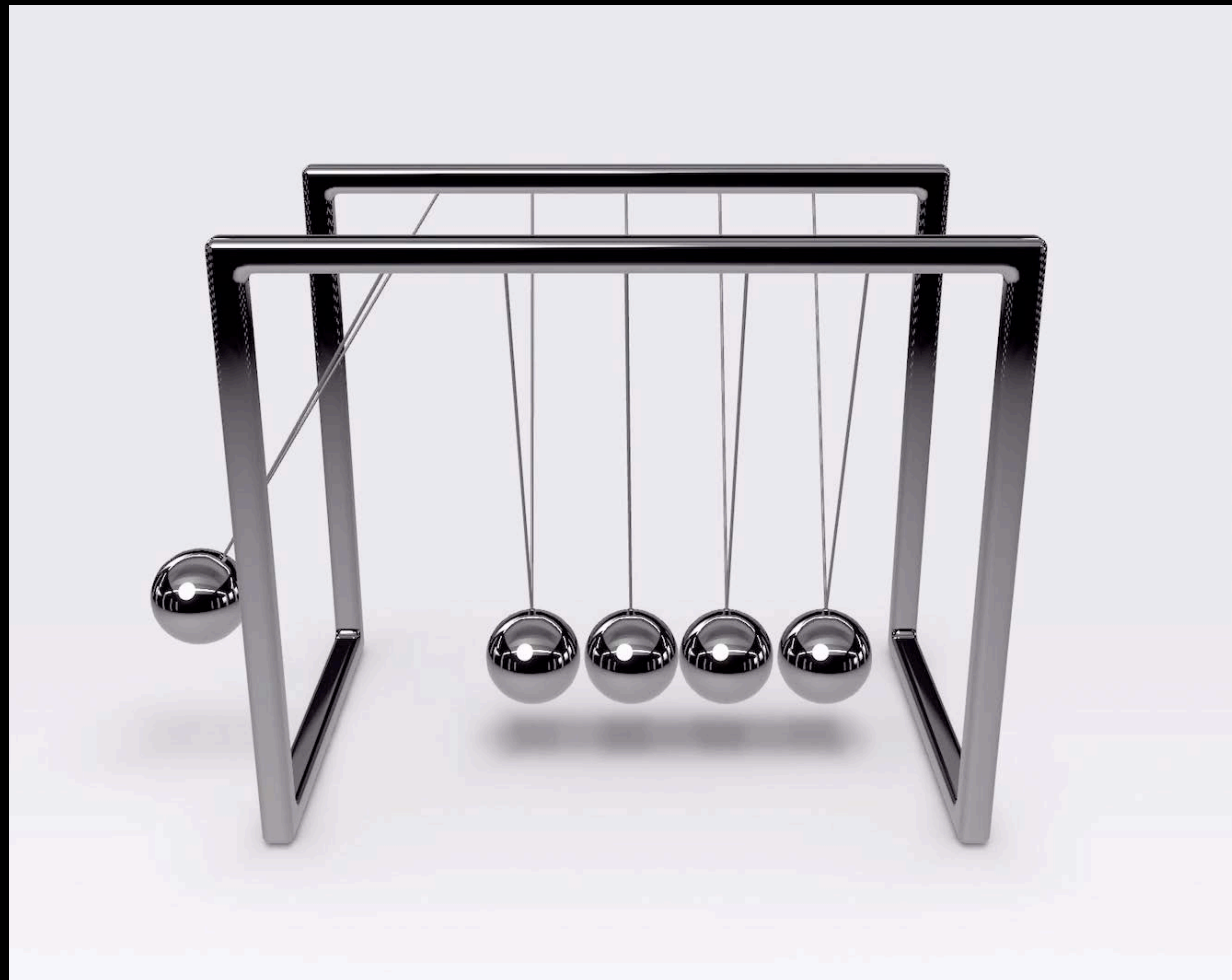
Ray Tracing in AR Quick Look



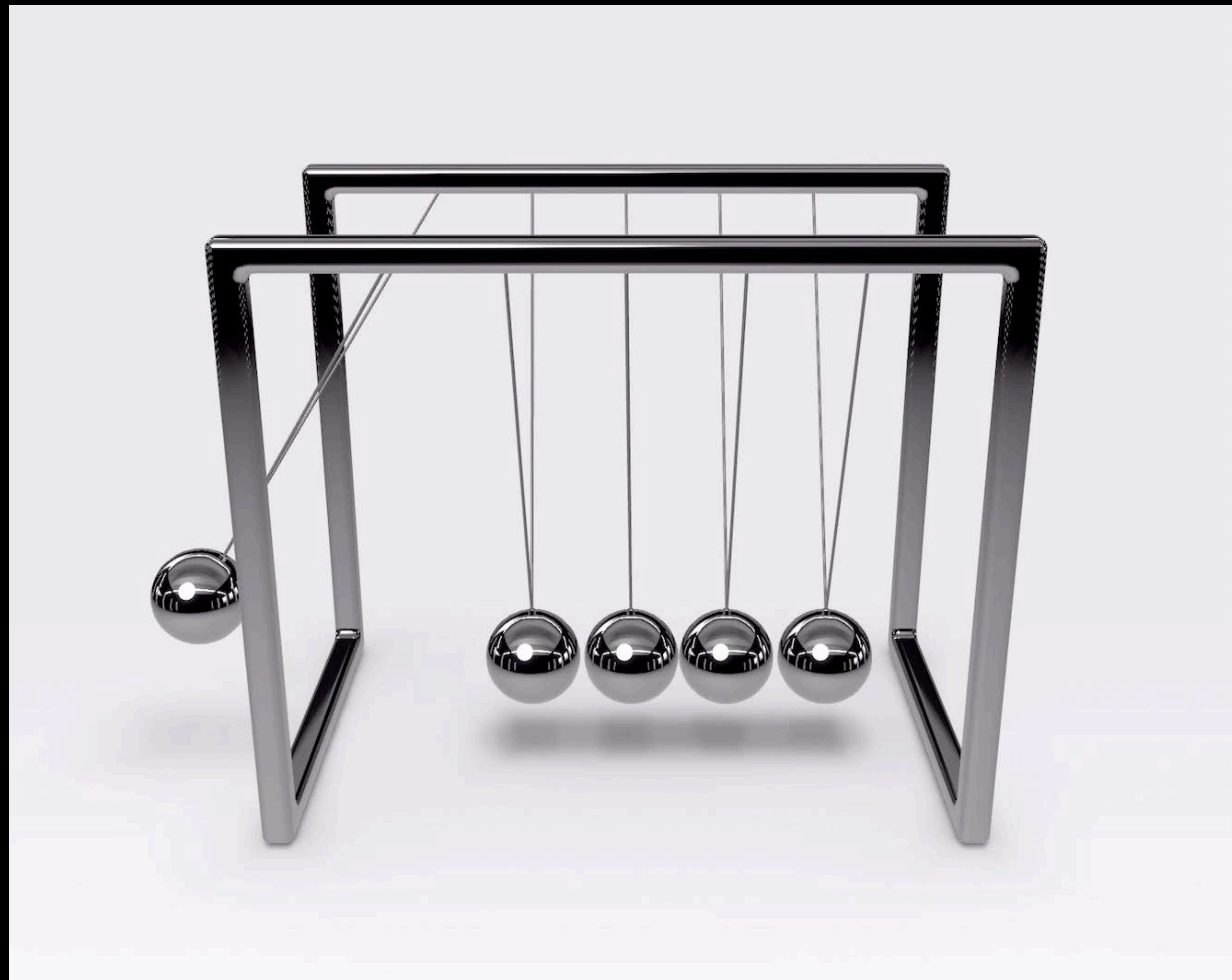
Ray Tracing in AR Quick Look



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

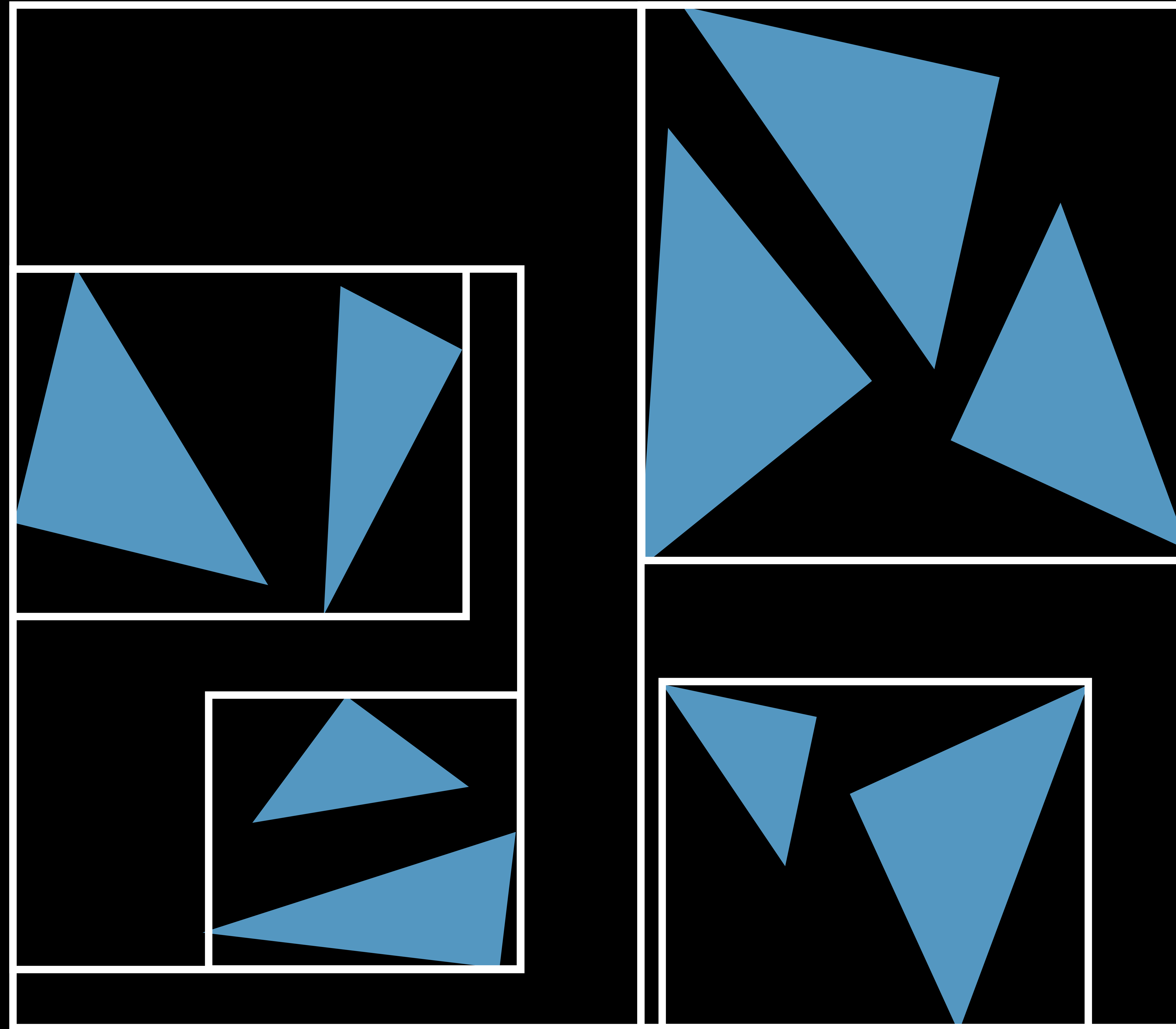
Deformation and skinned animation

Need to update acceleration structure

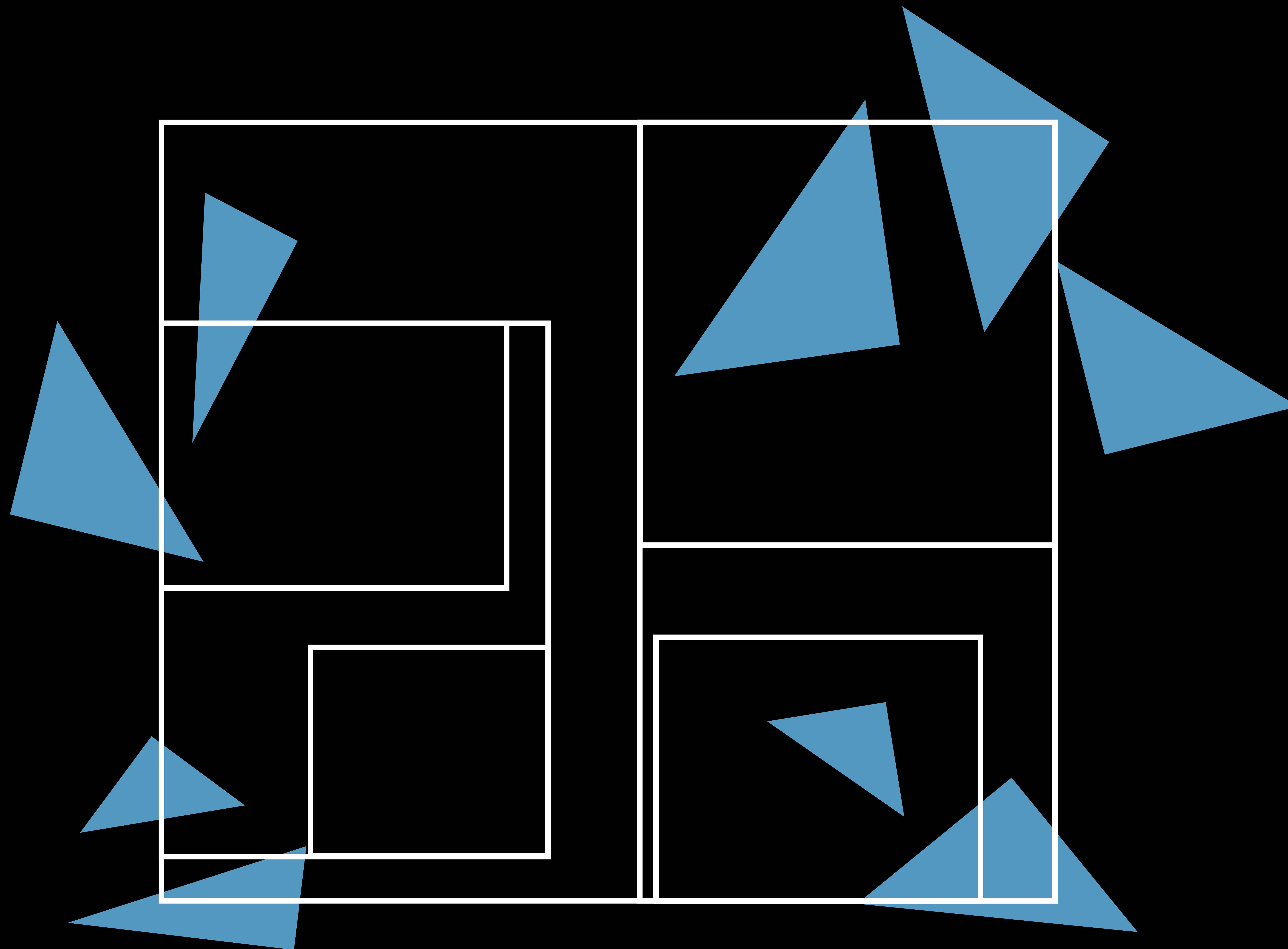
Objects tend to retain their shape



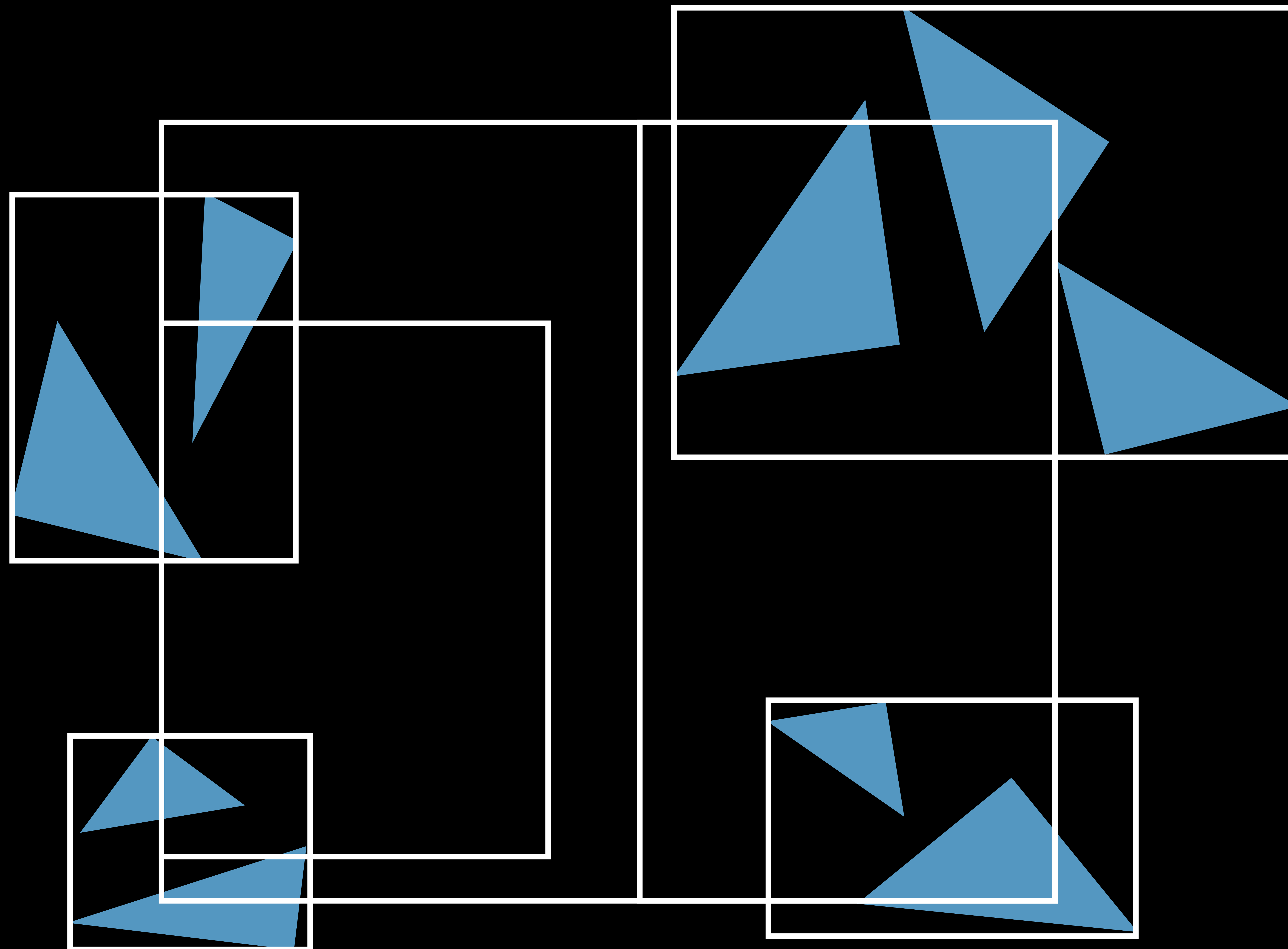
Vertex Animation



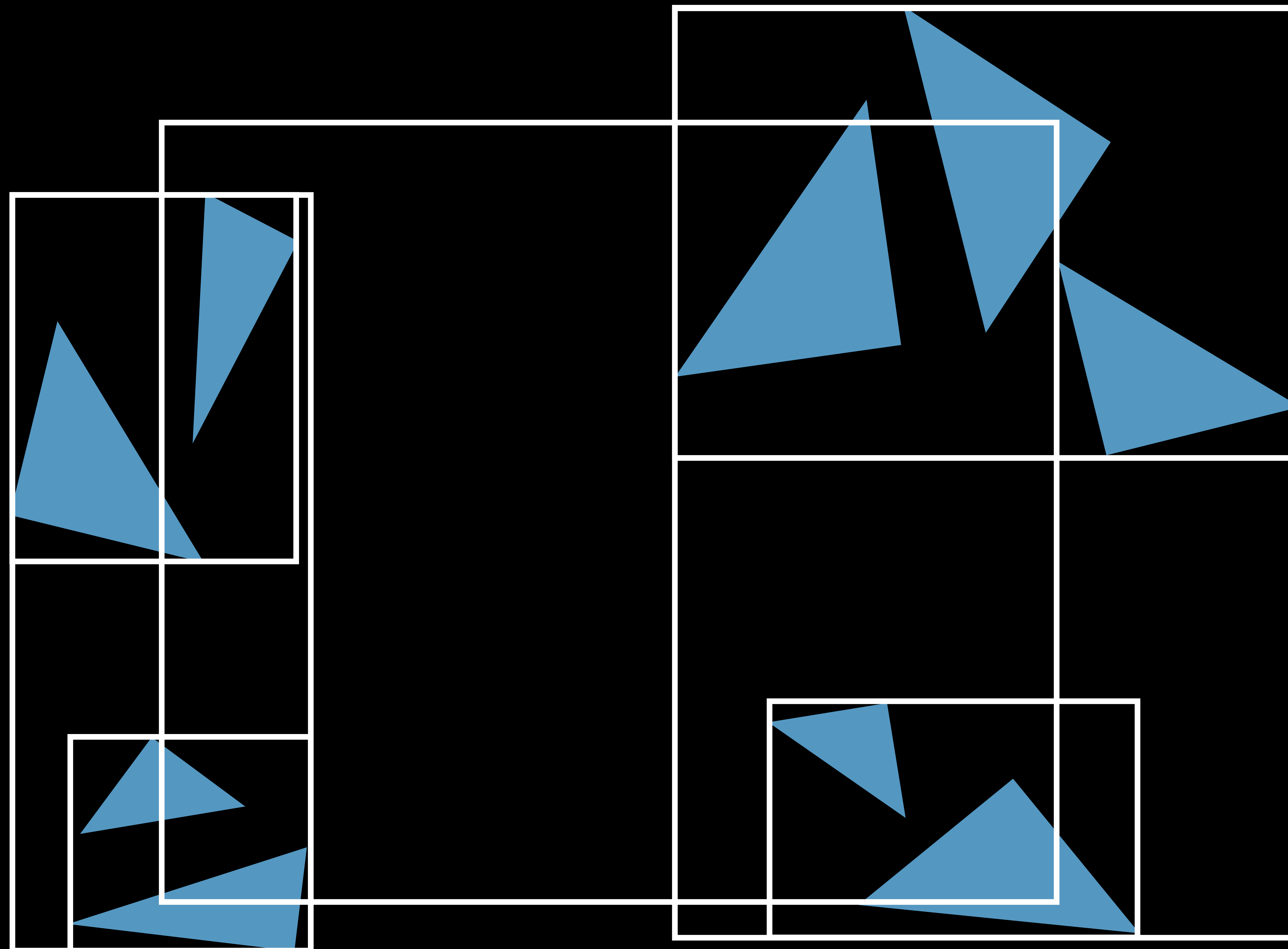
Vertex Animation



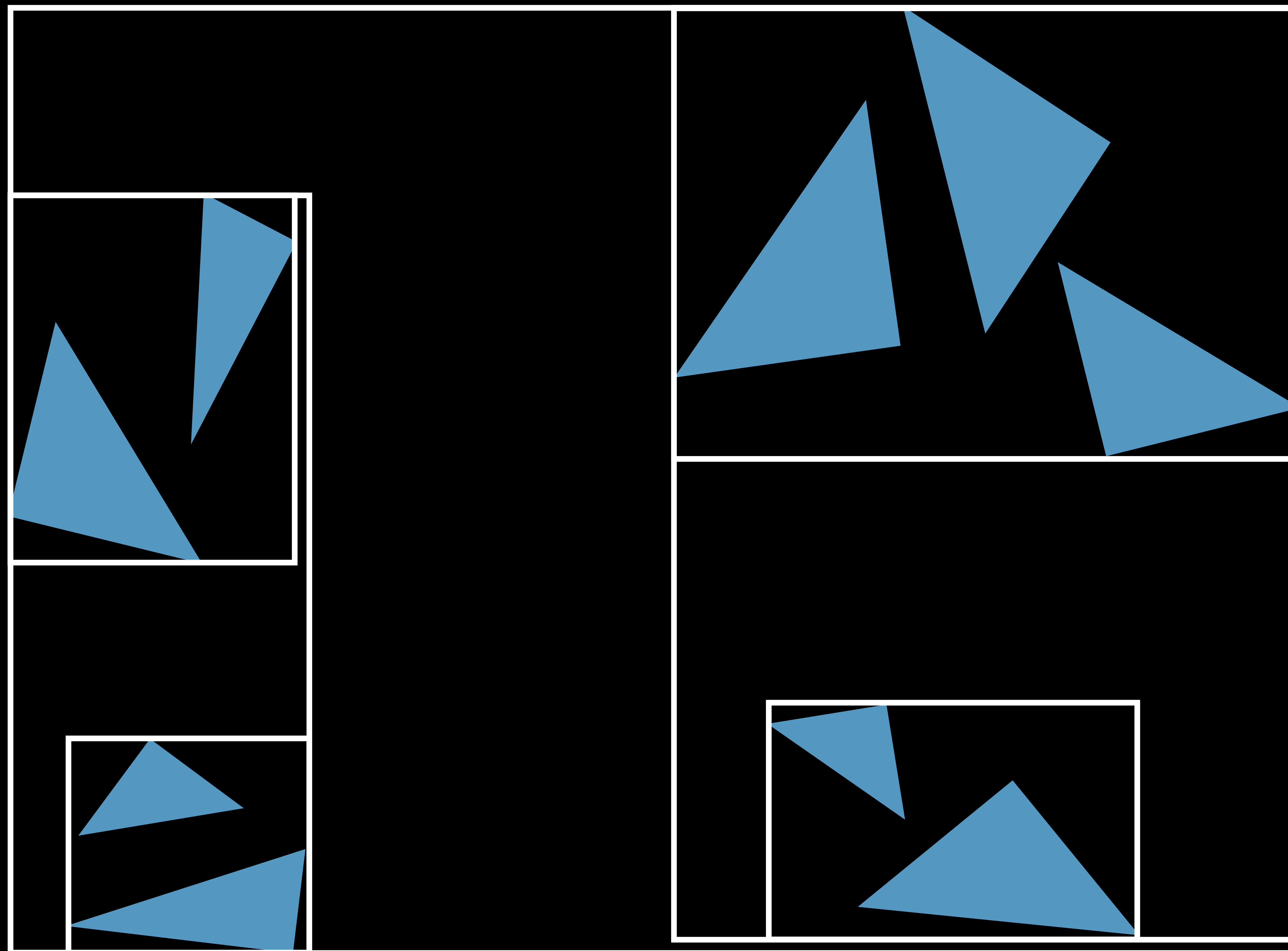
Vertex Animation



Vertex Animation



Vertex Animation



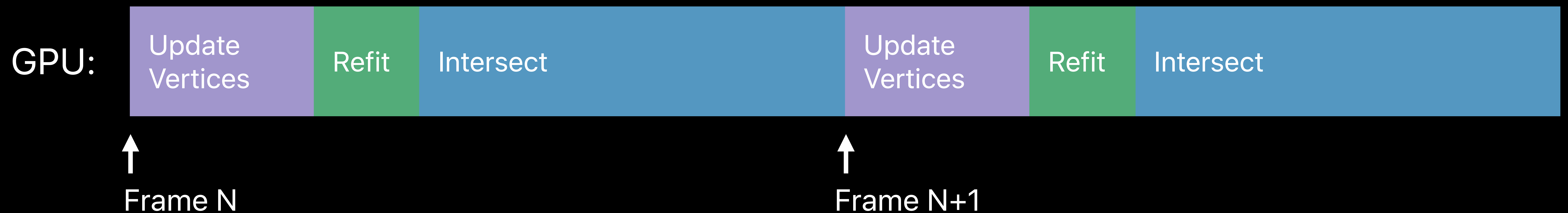
Refitting

Much faster than building from scratch

Refitting

Much faster than building from scratch

Runs on the GPU

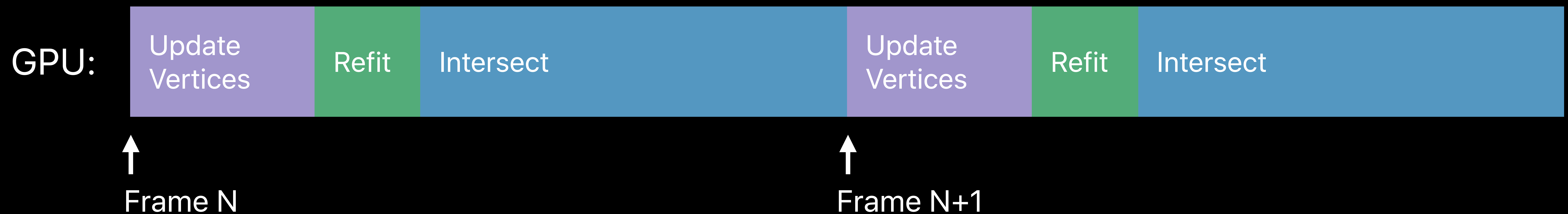


Refitting

Much faster than building from scratch

Runs on the GPU

Can't add or remove geometry



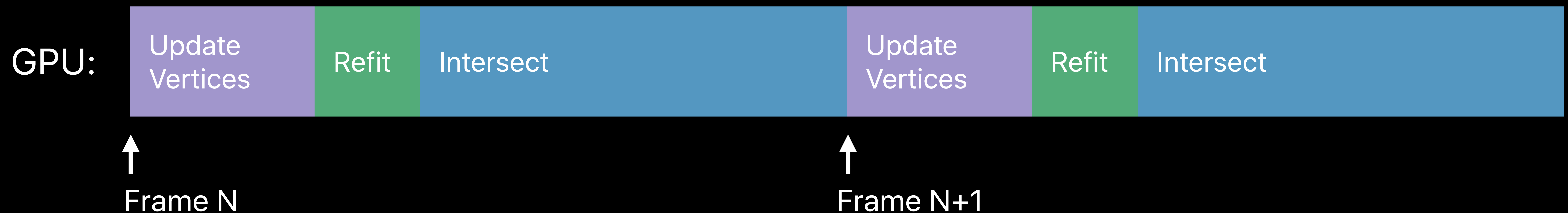
Refitting

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

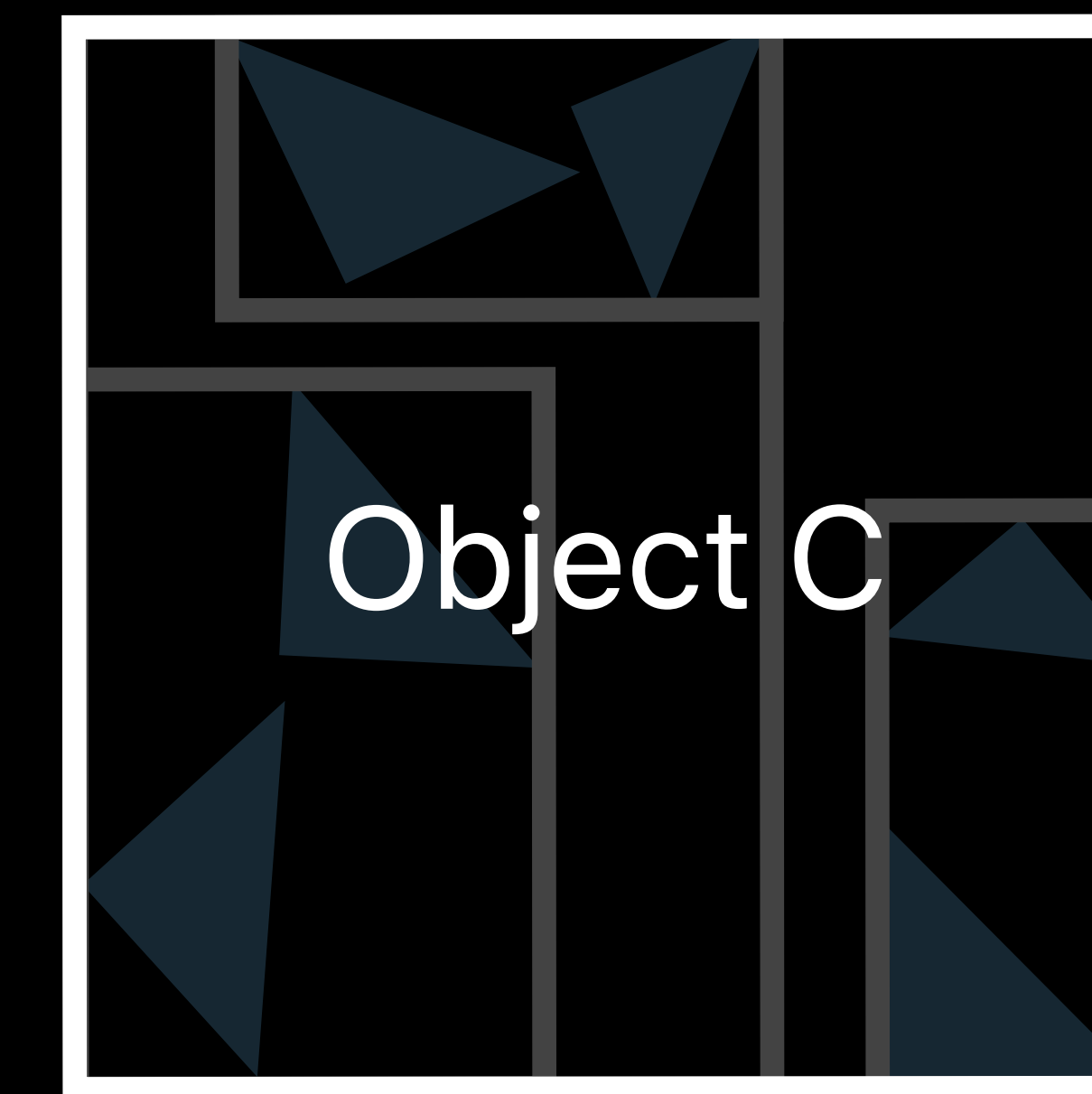
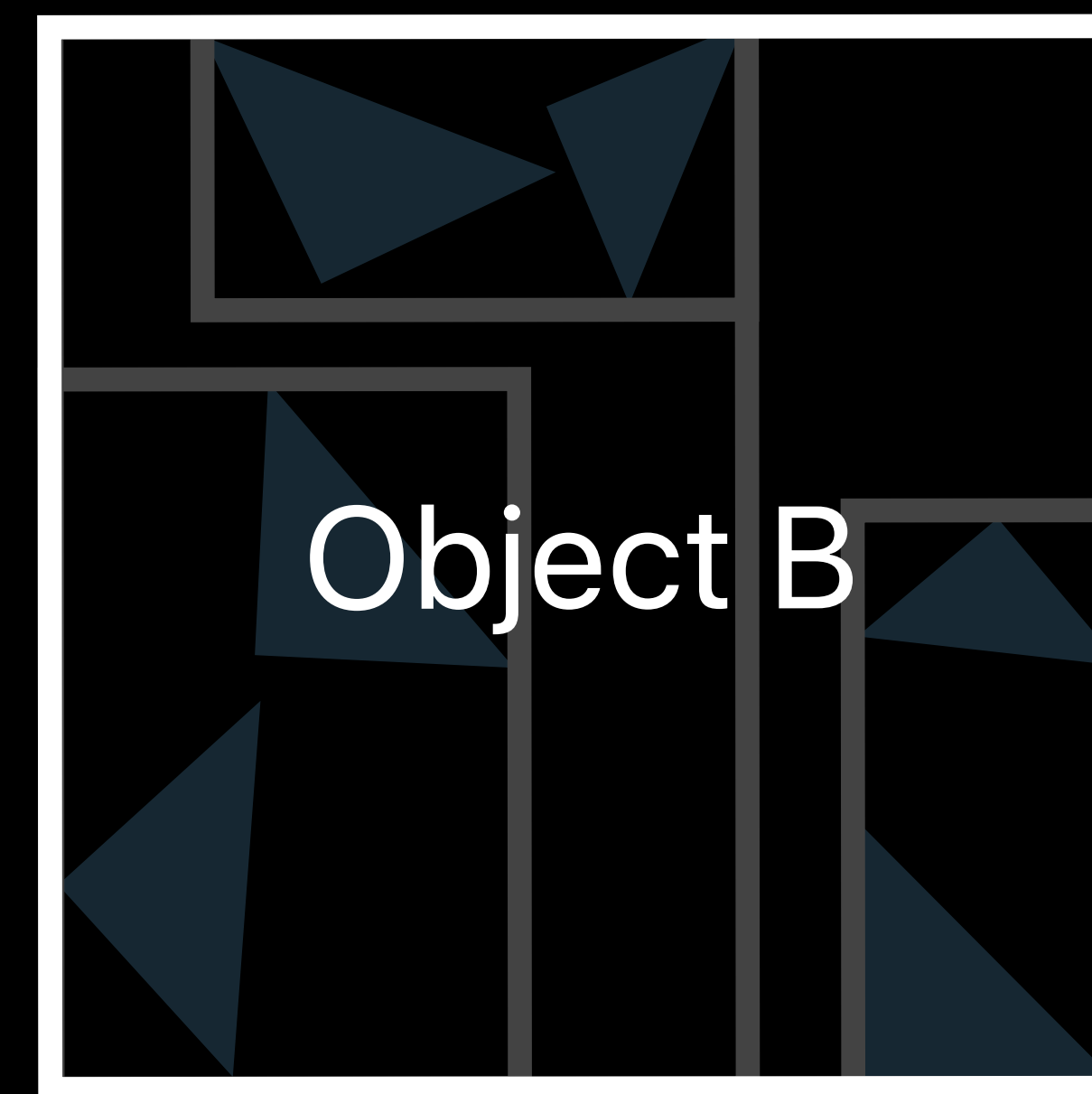
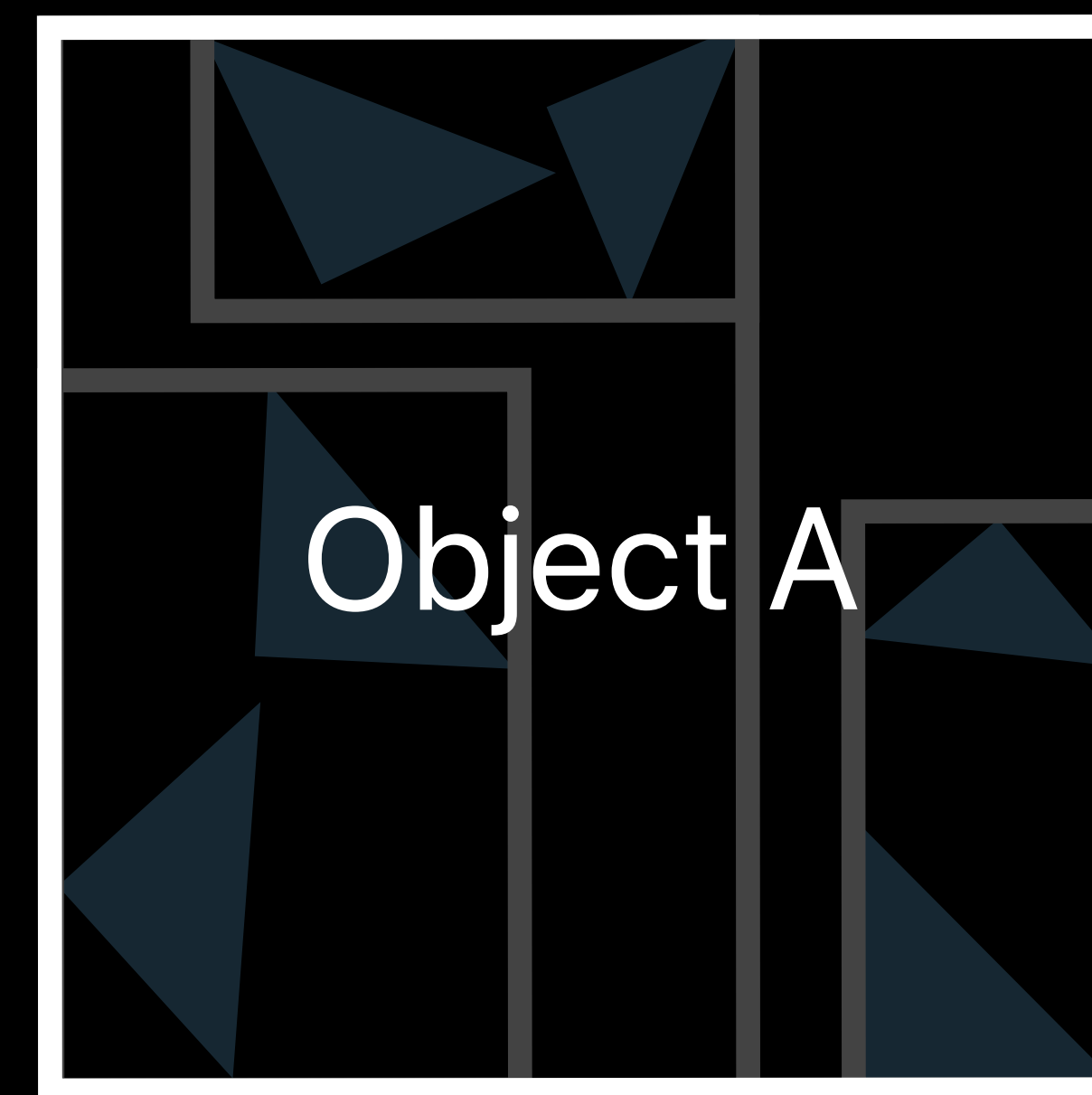


Rigid Body Animation

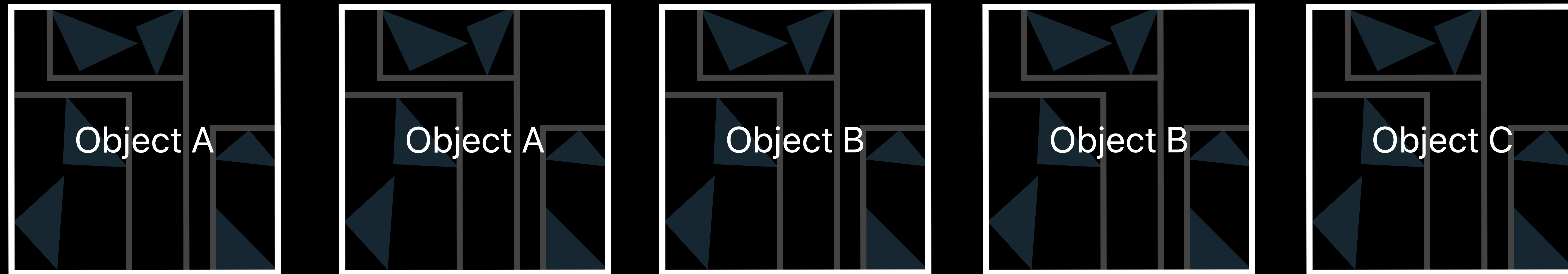
Most geometry only moves rigidly or not at all
May have multiple copies of the same objects



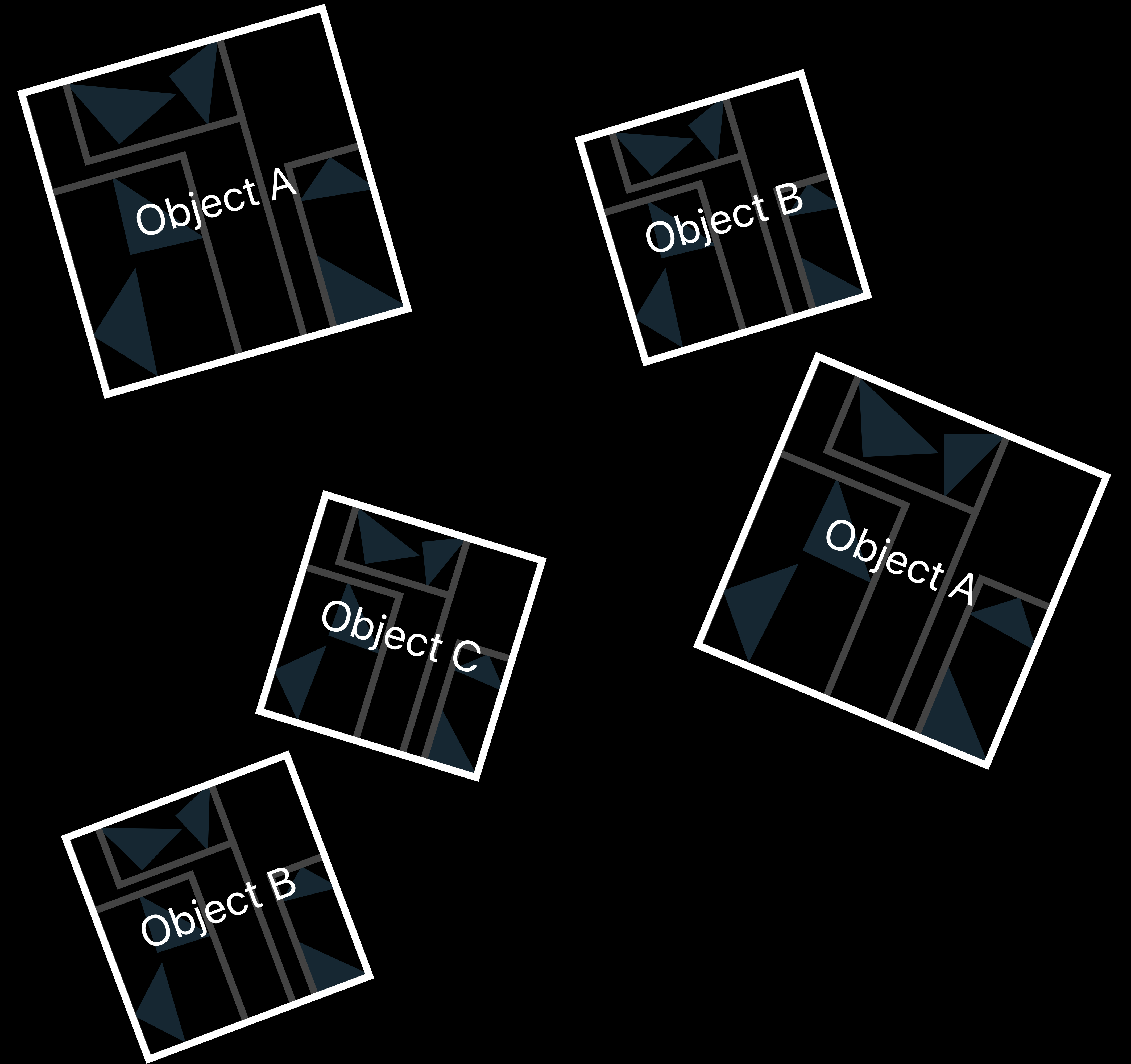
Two-Level Acceleration Structures



Two-Level Acceleration Structures



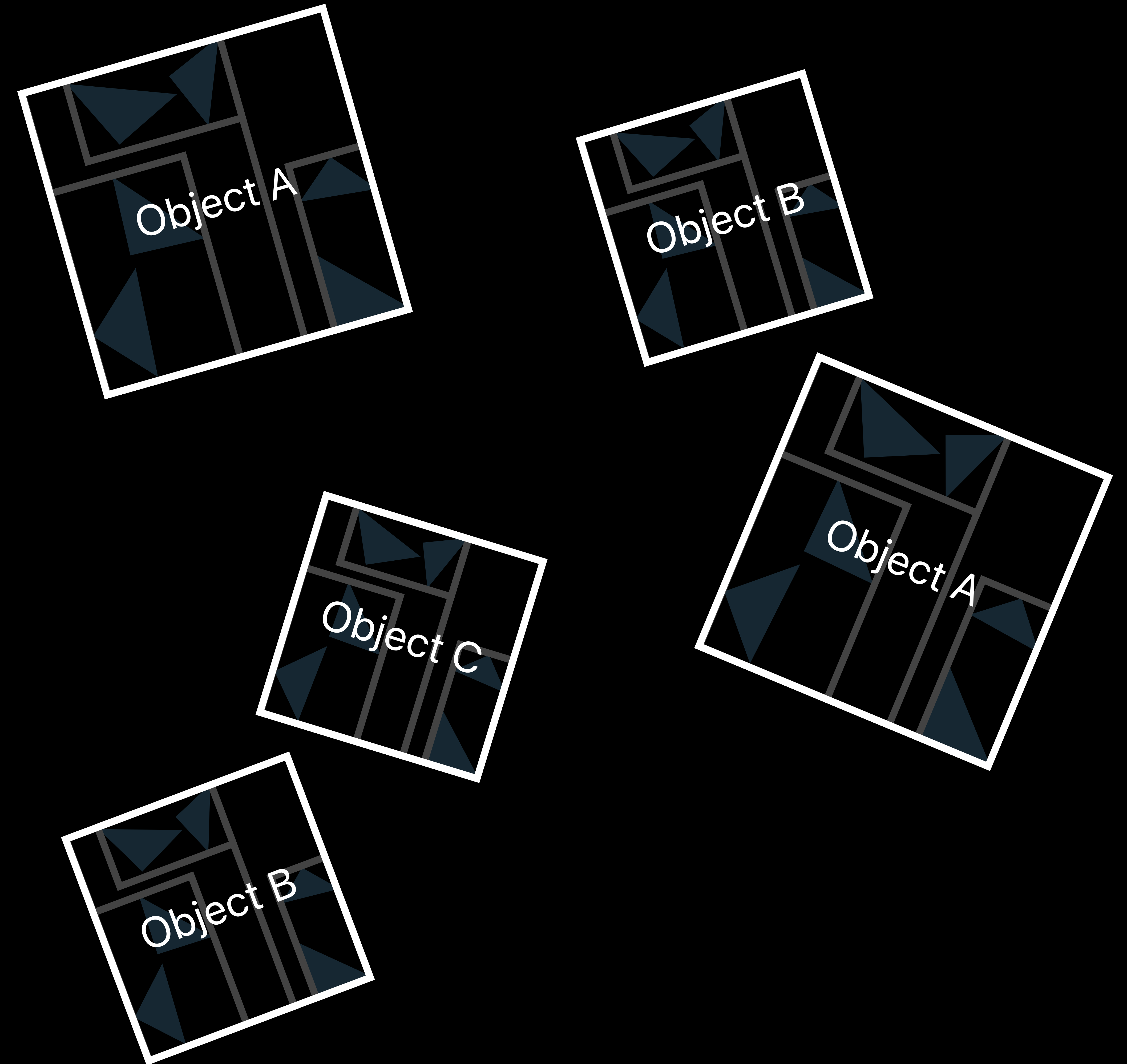
Two-Level Acceleration Structures



Two-Level Acceleration Structures

Transformation matrices:

float4x4	float4x4	float4x4	float4x4	float4x4
----------	----------	----------	----------	----------

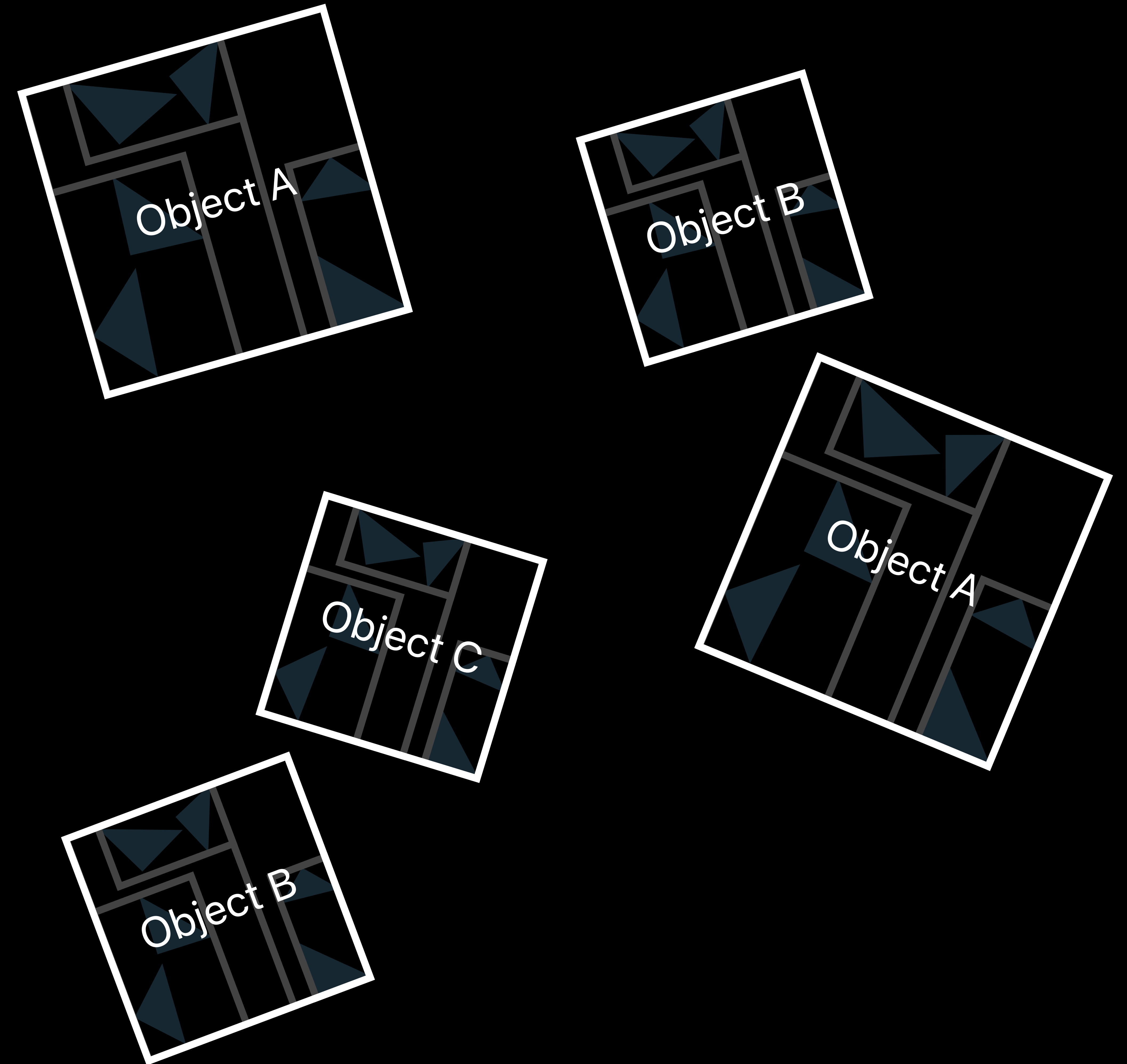
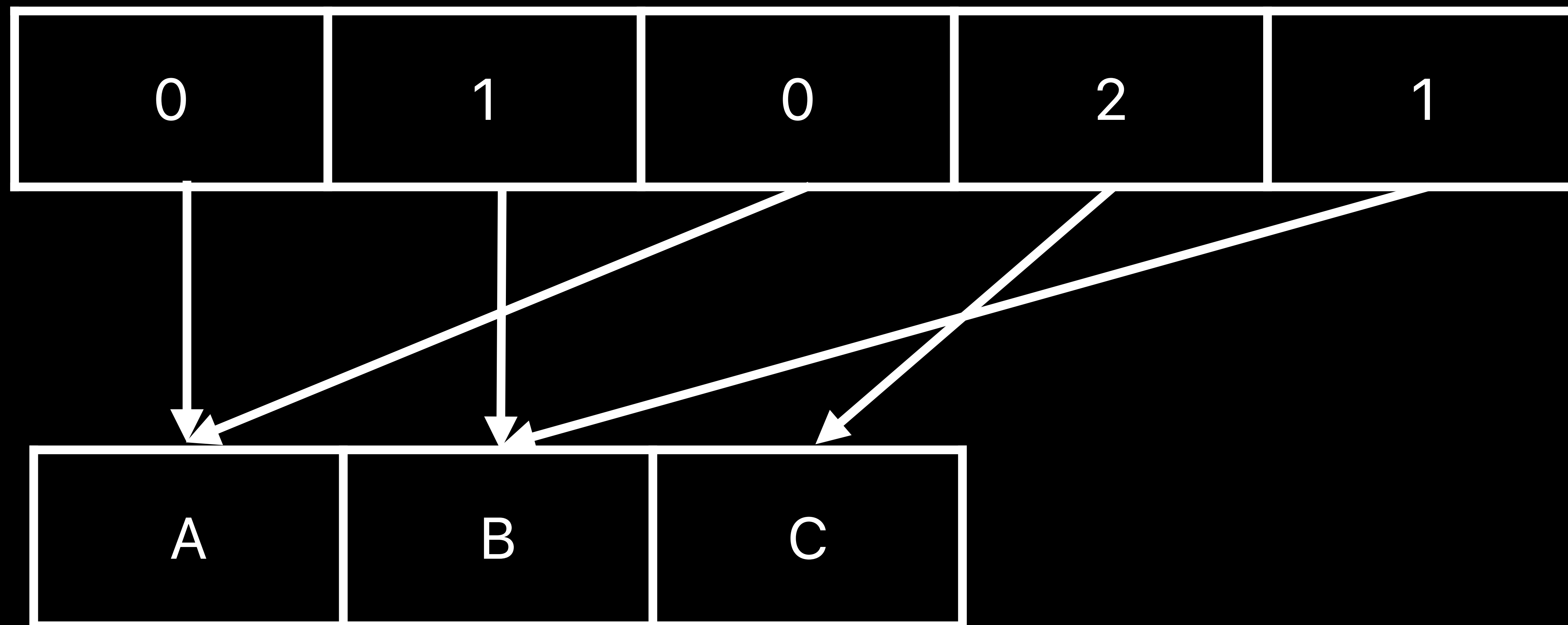


Two-Level Acceleration Structures

Transformation matrices:



Acceleration structure indices:

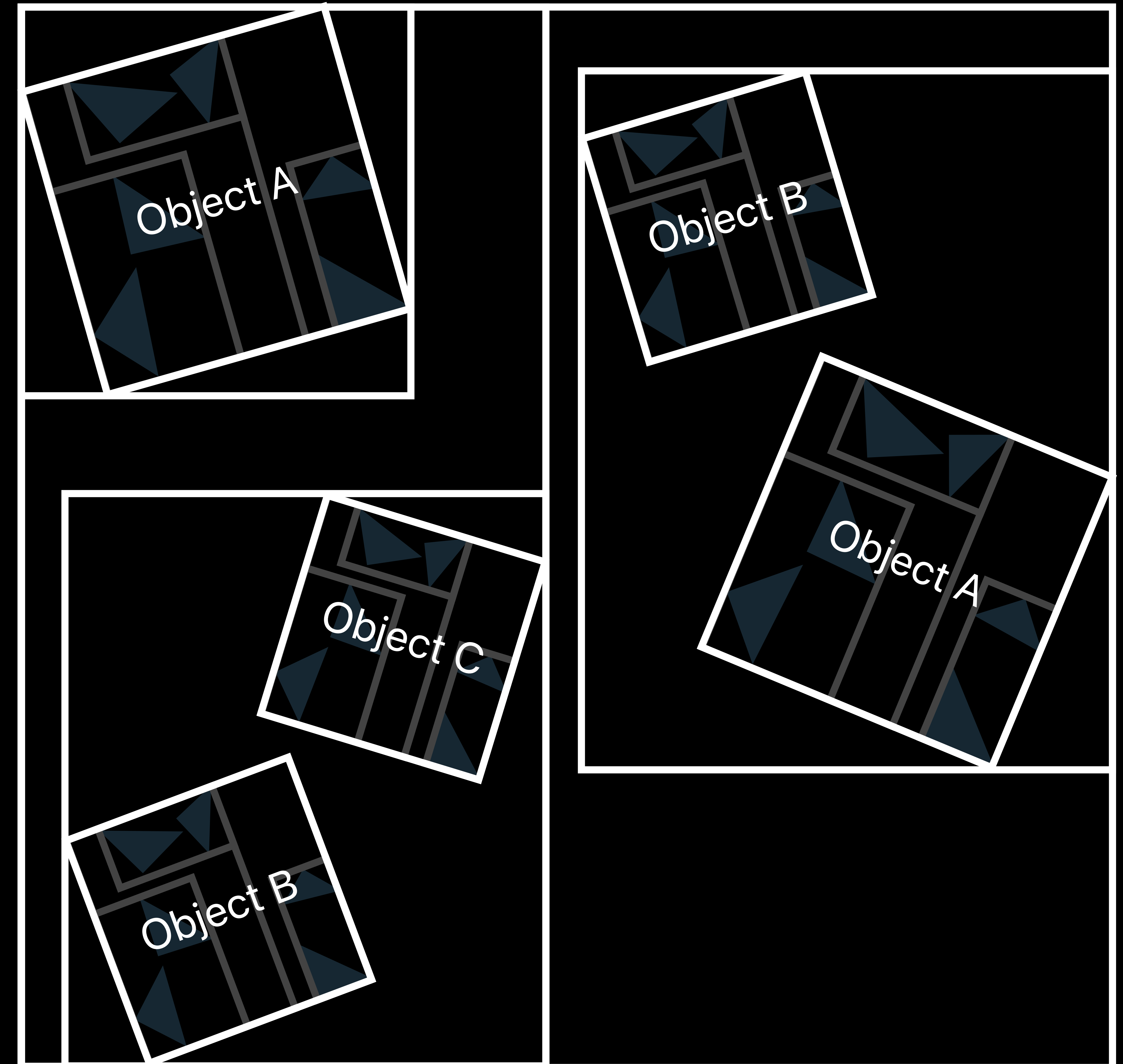
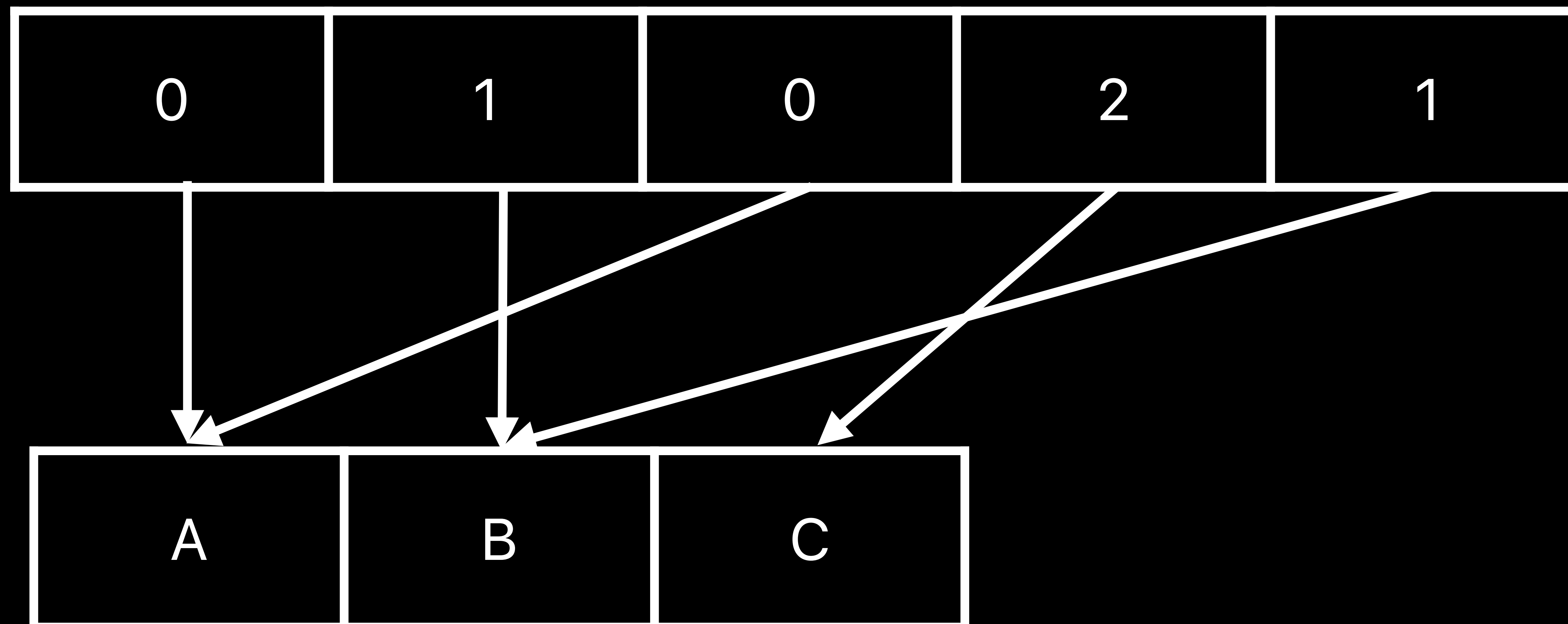


Two-Level Acceleration Structures

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



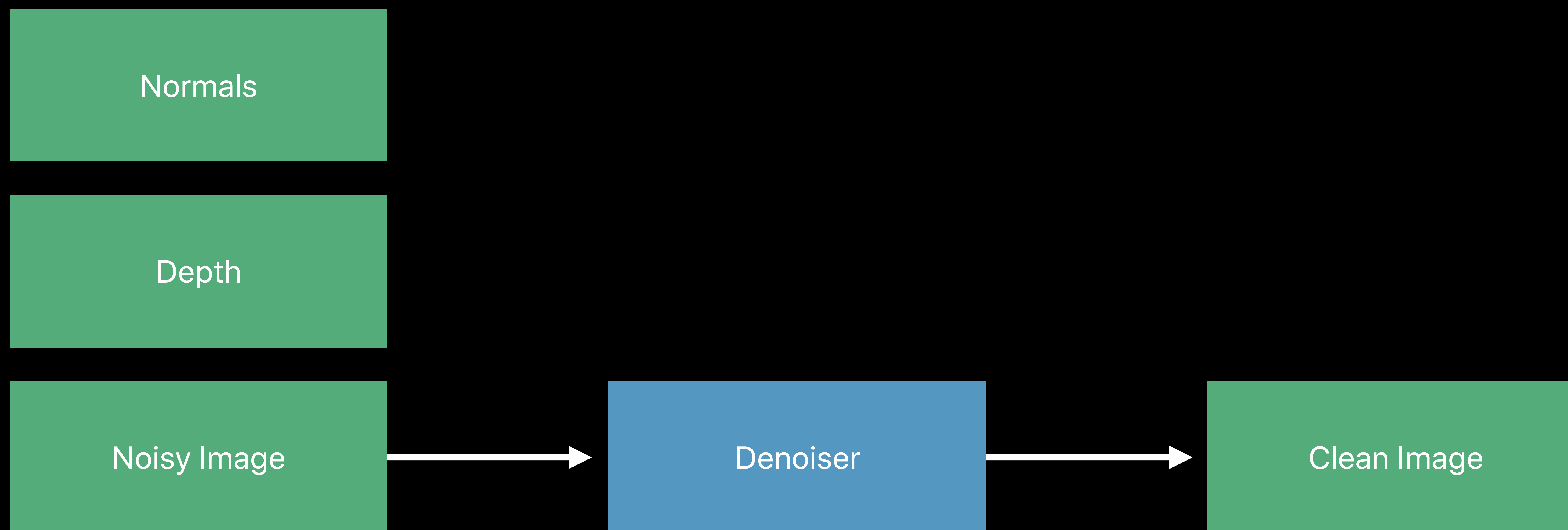
Denoising



Denoising



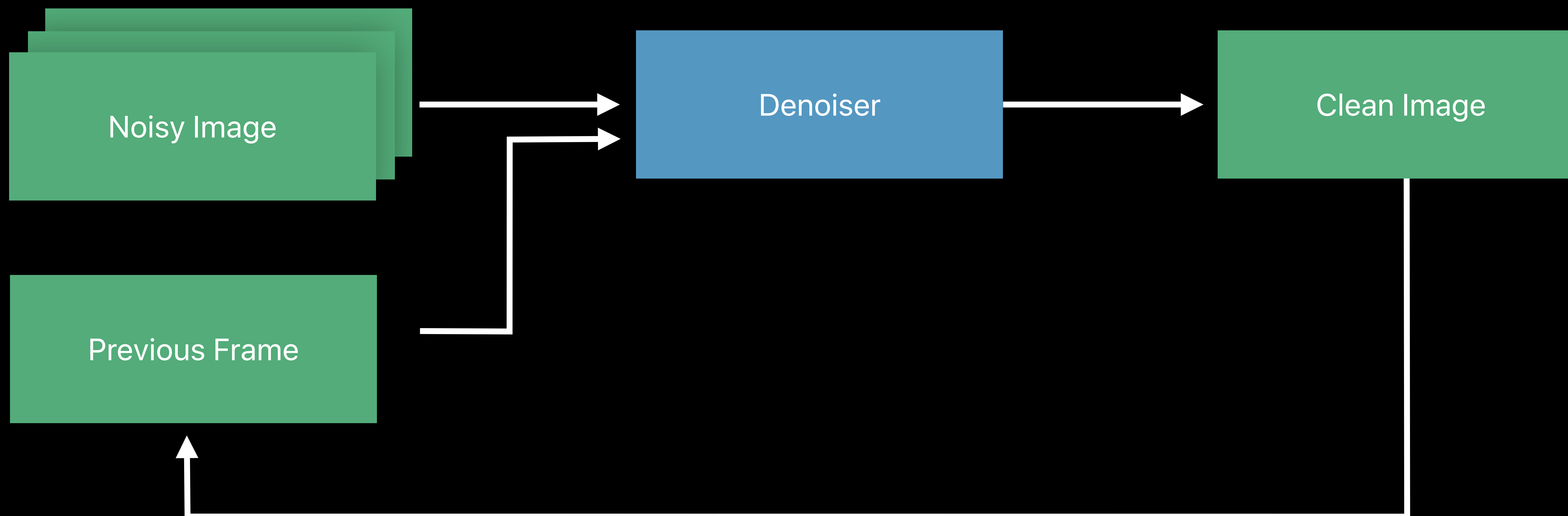
Denoising



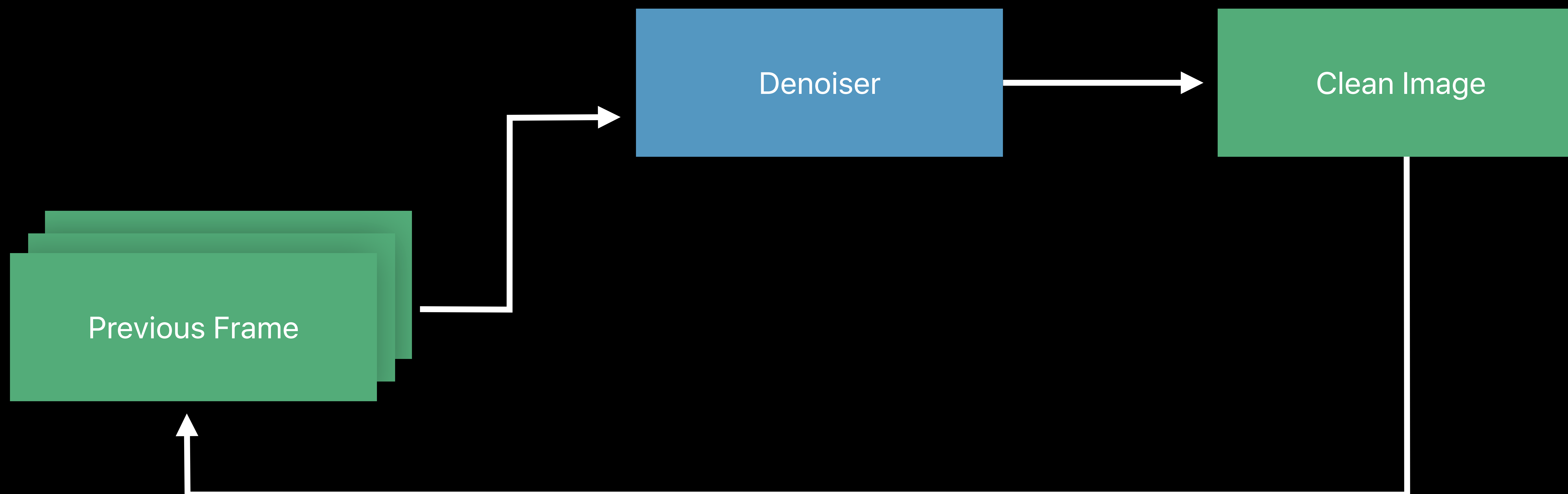
Denoising



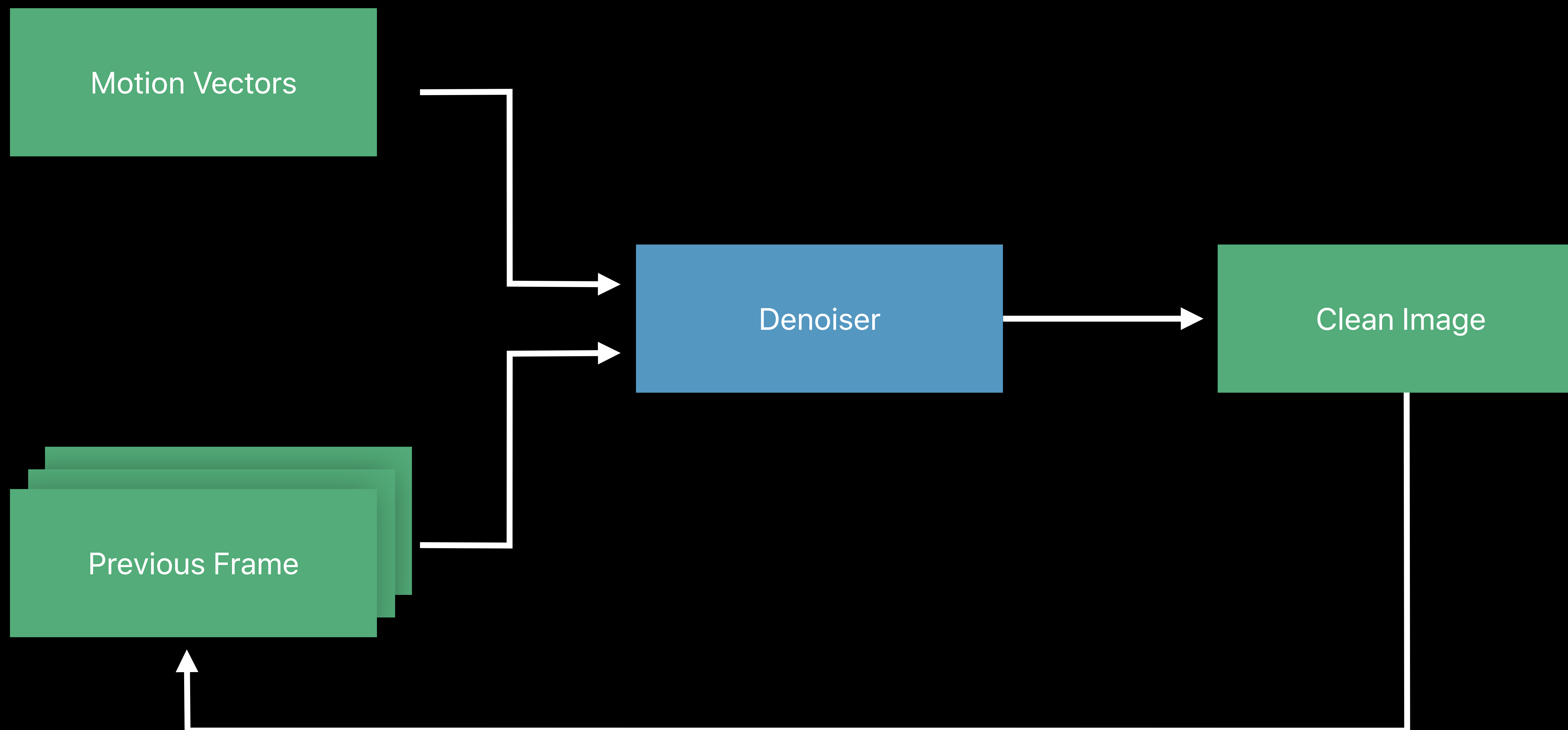
Denoising



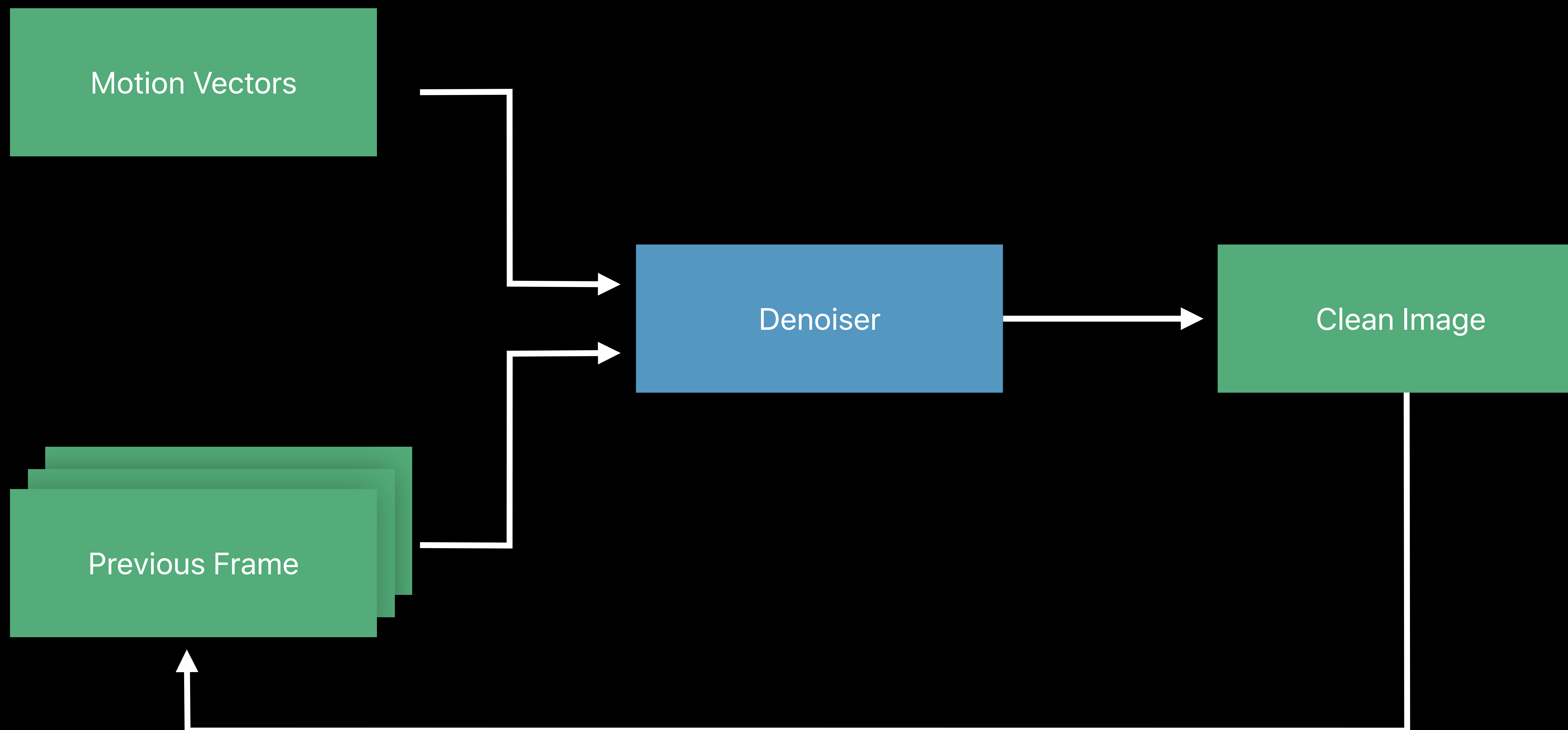
Denoising



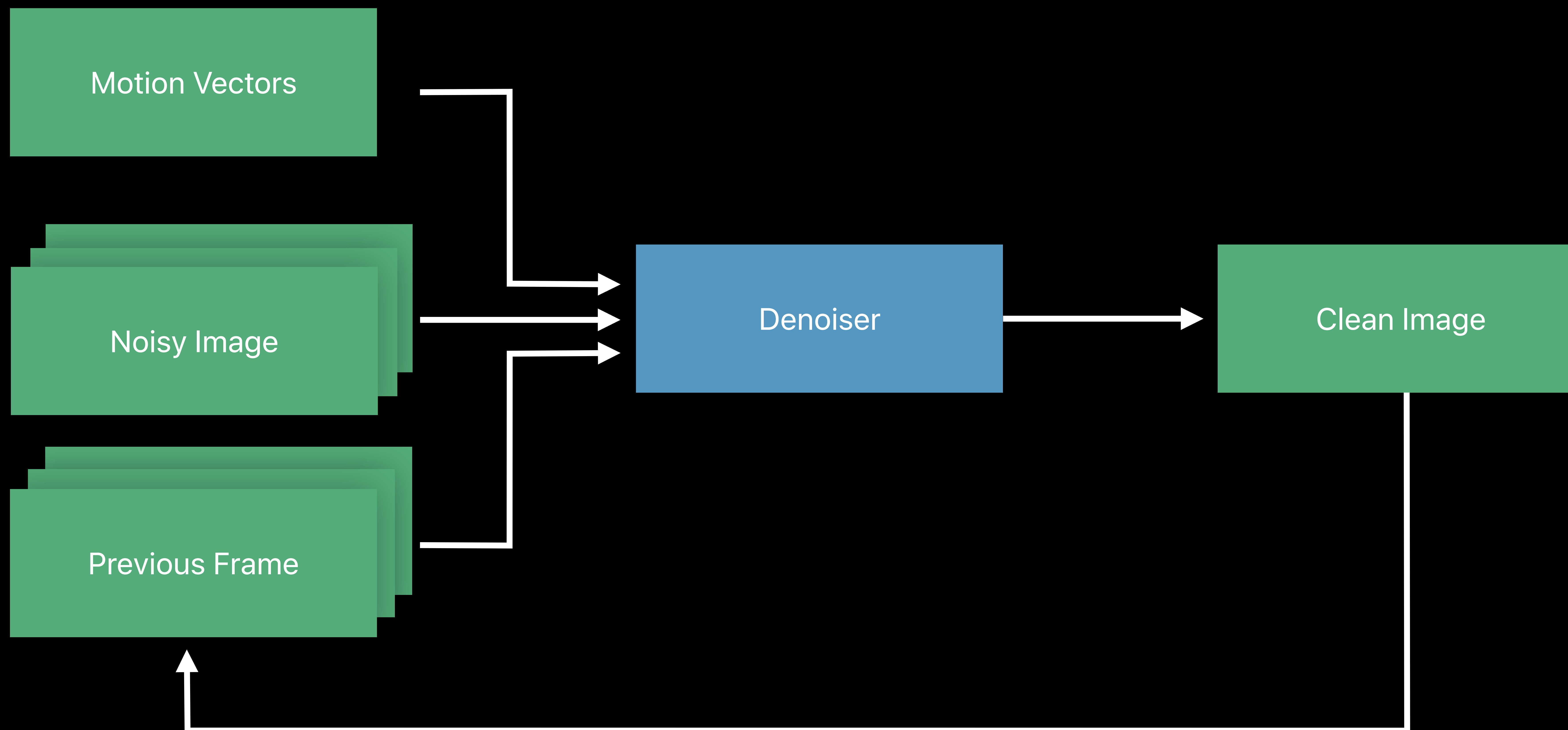
Denoising



Denoising



Denoising



MPSSVGF

High-quality "Spatiotemporal Variance-Guided Filtering" denoising algorithm

MPSSVGF

High-quality "Spatiotemporal Variance-Guided Filtering" denoising algorithm

MPSSVGFDenoisier coordinates
denoising process



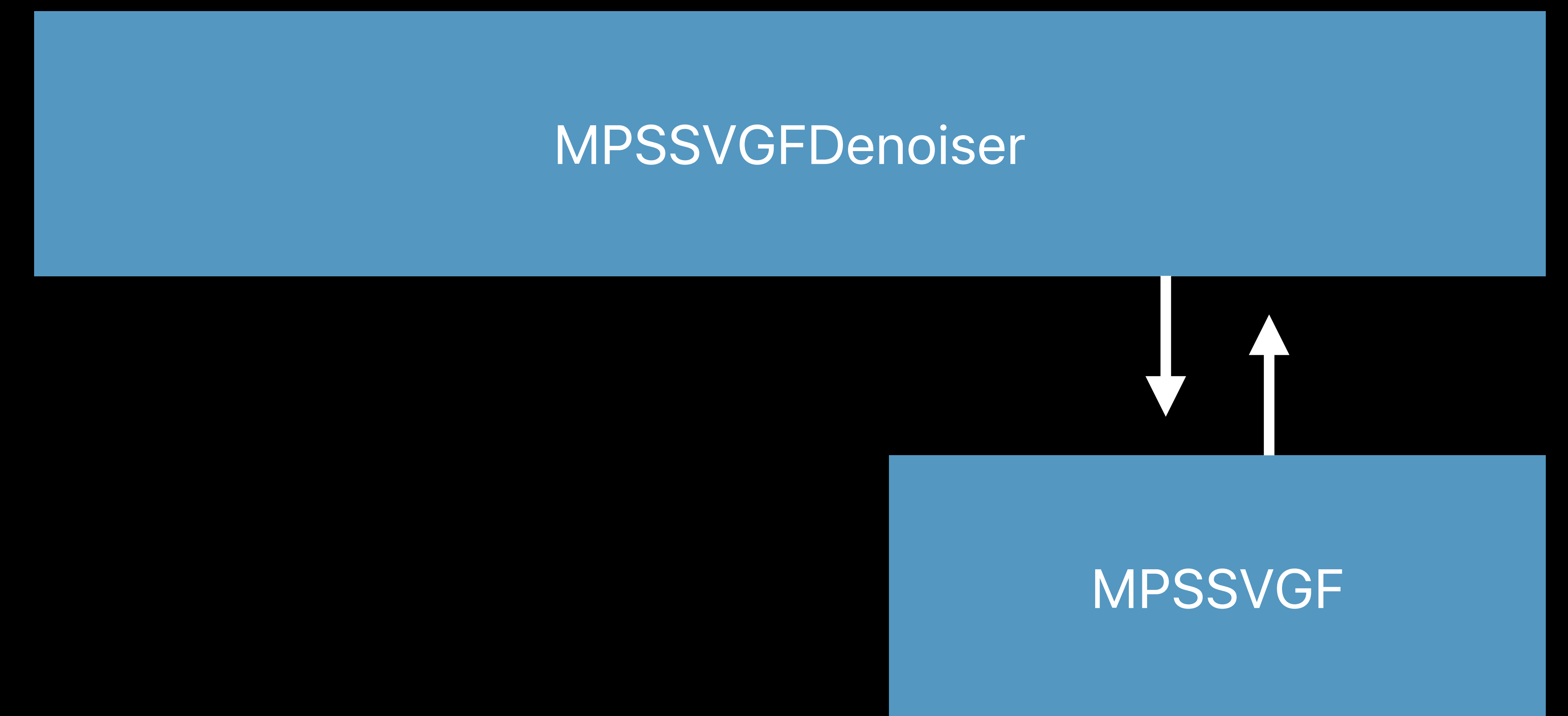
MPSSVGFDenoisier

MPSSVGF

High-quality "Spatiotemporal Variance-Guided Filtering" denoising algorithm

MPSSVGFDenoisier coordinates denoising process

Low-level control

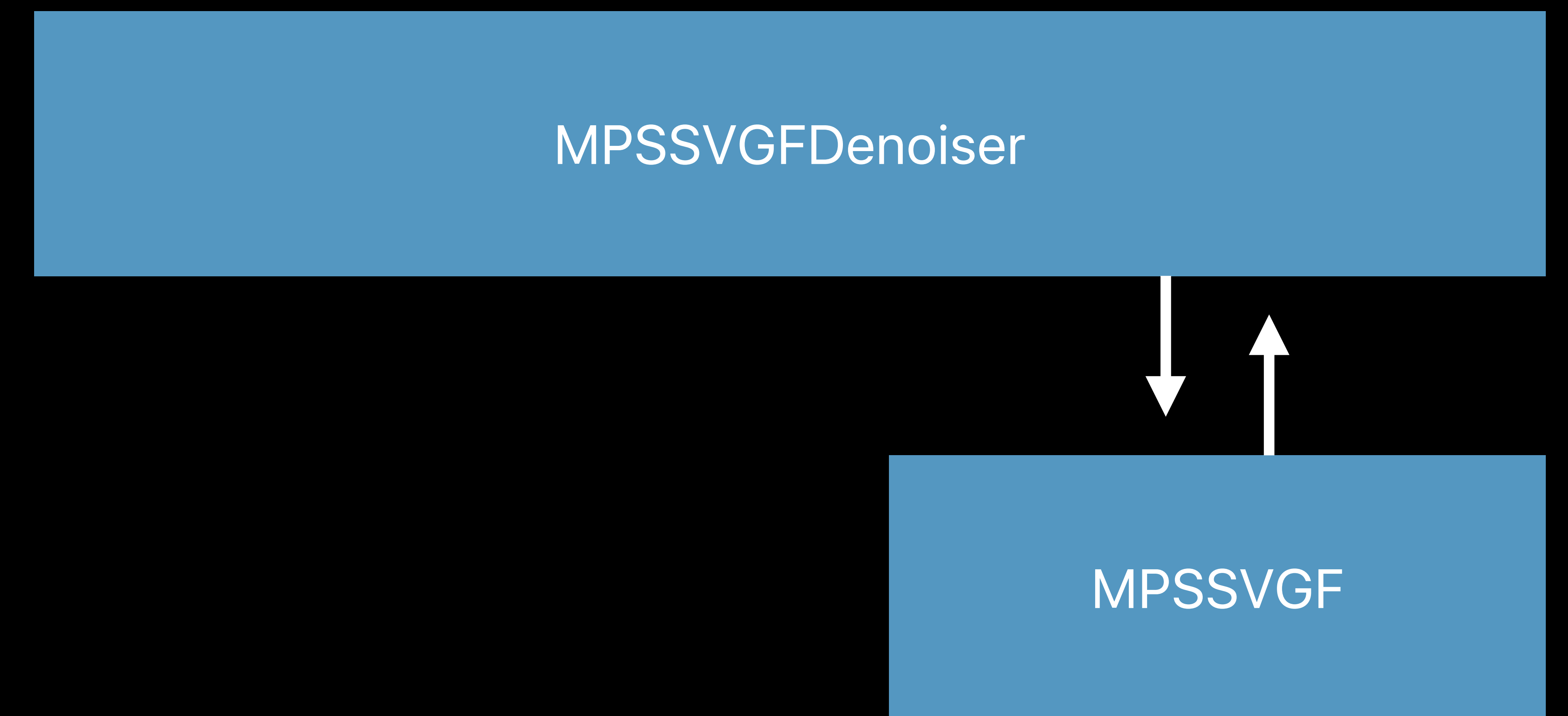


MPSSVGF

High-quality "Spatiotemporal Variance-Guided Filtering" denoising algorithm

MPSSVGFDenoisier coordinates denoising process

Low-level control

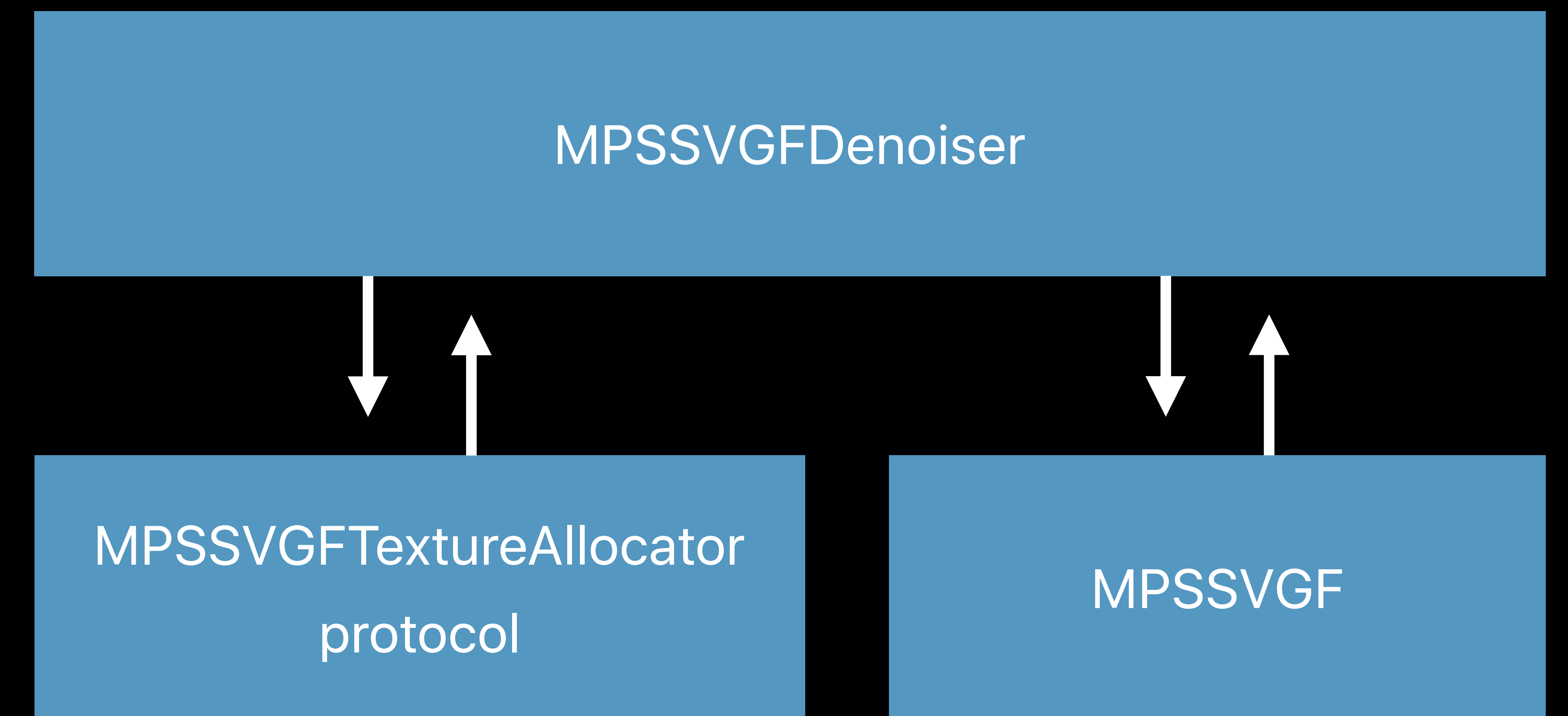


MPSSVGF

High-quality "Spatiotemporal Variance-Guided Filtering" denoising algorithm

MPSSVGFDenoisier coordinates denoising process

Low-level control



MPSSVGFDenoiser

Create denoiser:

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

// Configure SVGF properties

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

// Create the denoiser object
let denoiser = MPSSVGFDenoiser(SVGFD: 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

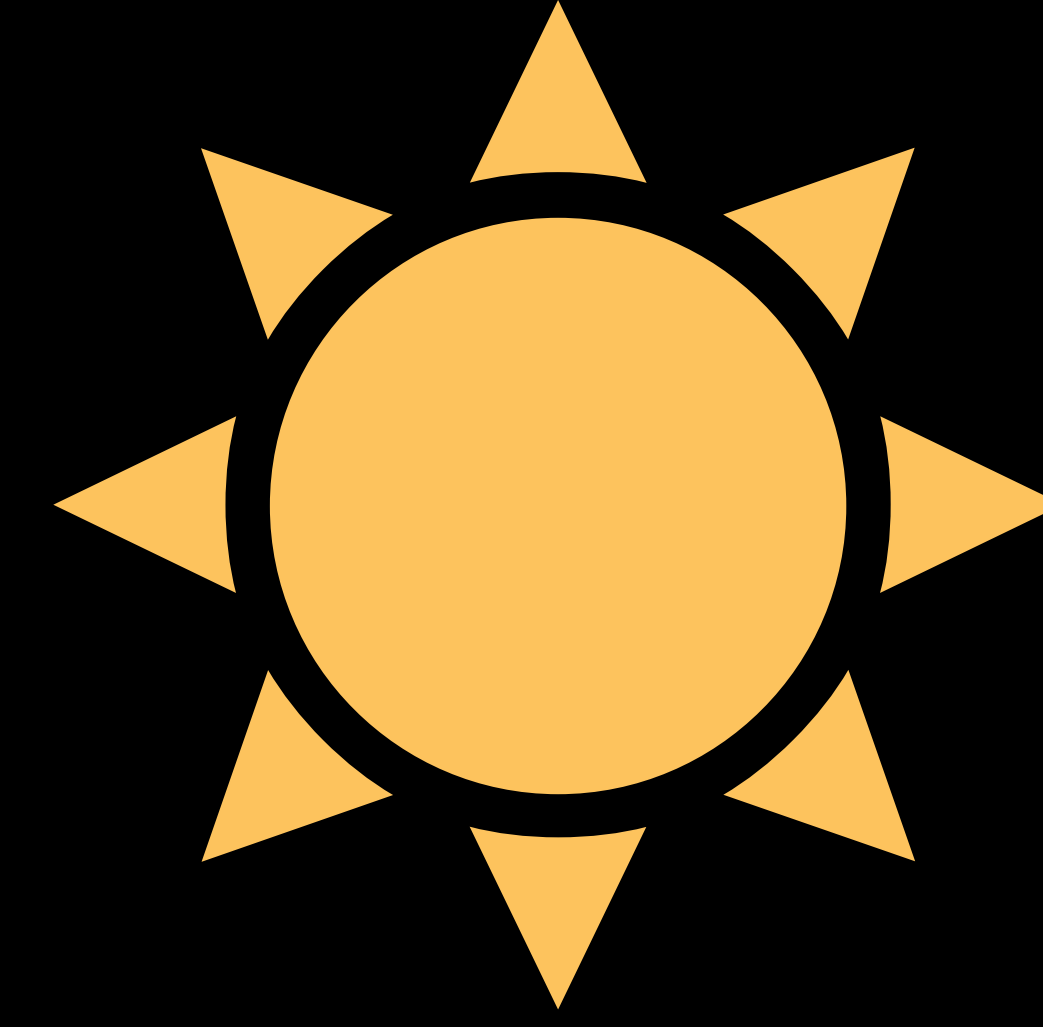
Ray/triangle intersection

Dynamic scenes

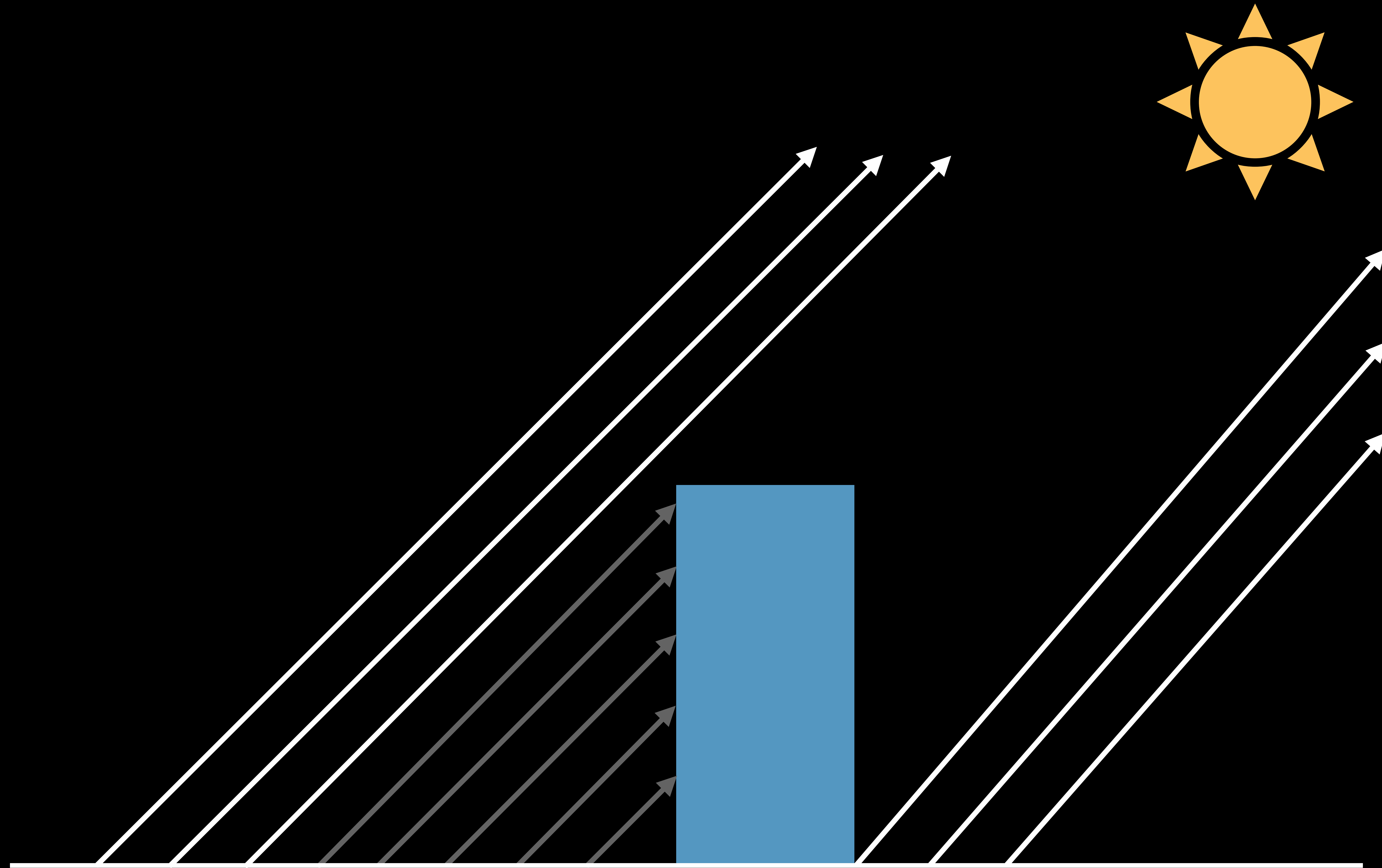
Denoising

Ray Tracing in Practice

Hard Shadows



Hard Shadows

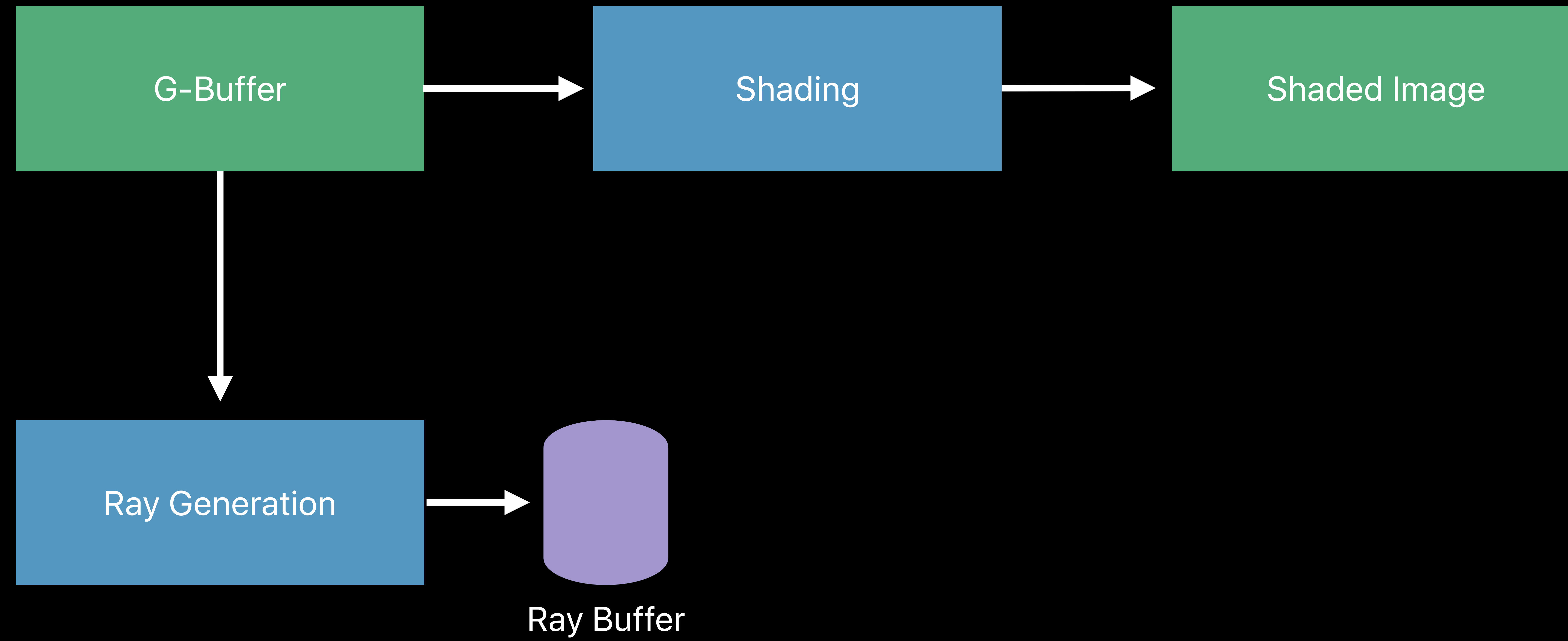


Hybrid Rendering

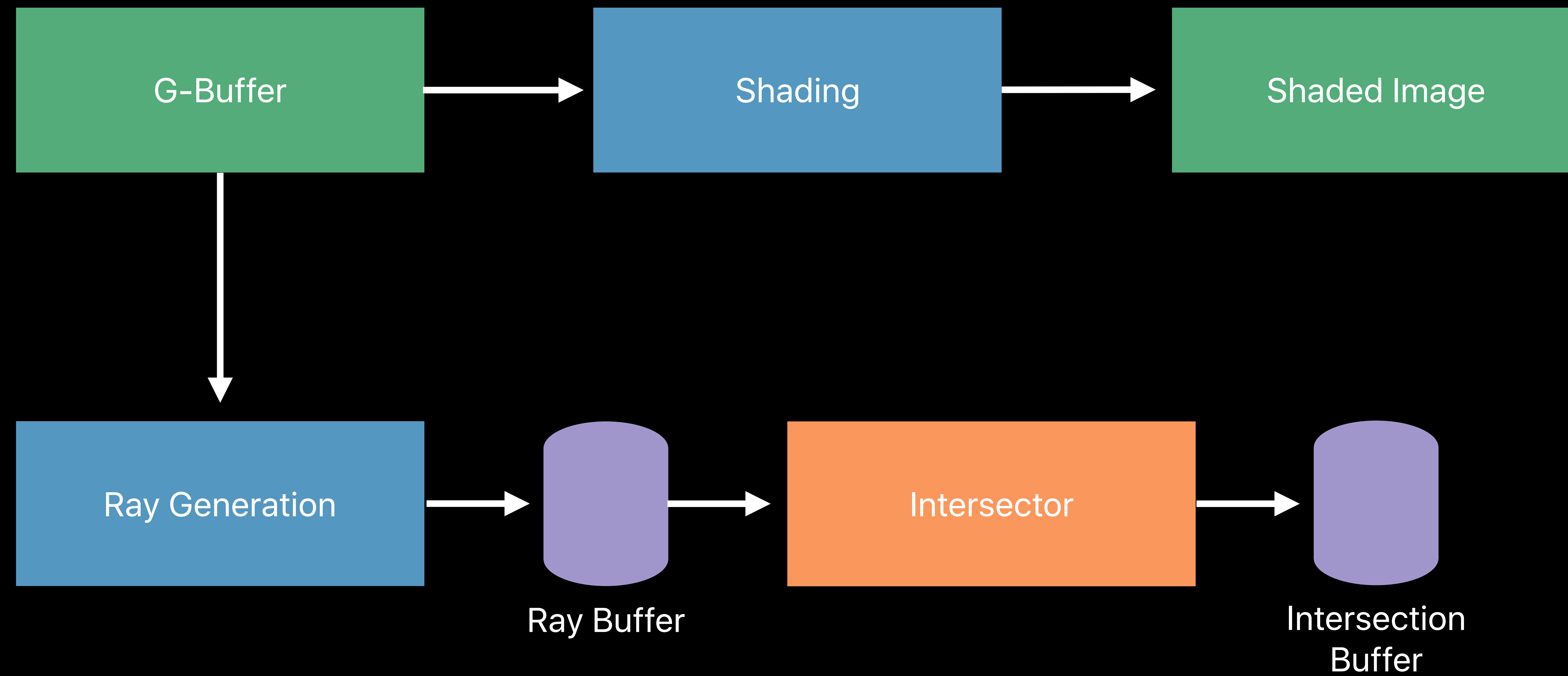
Hybrid Rendering



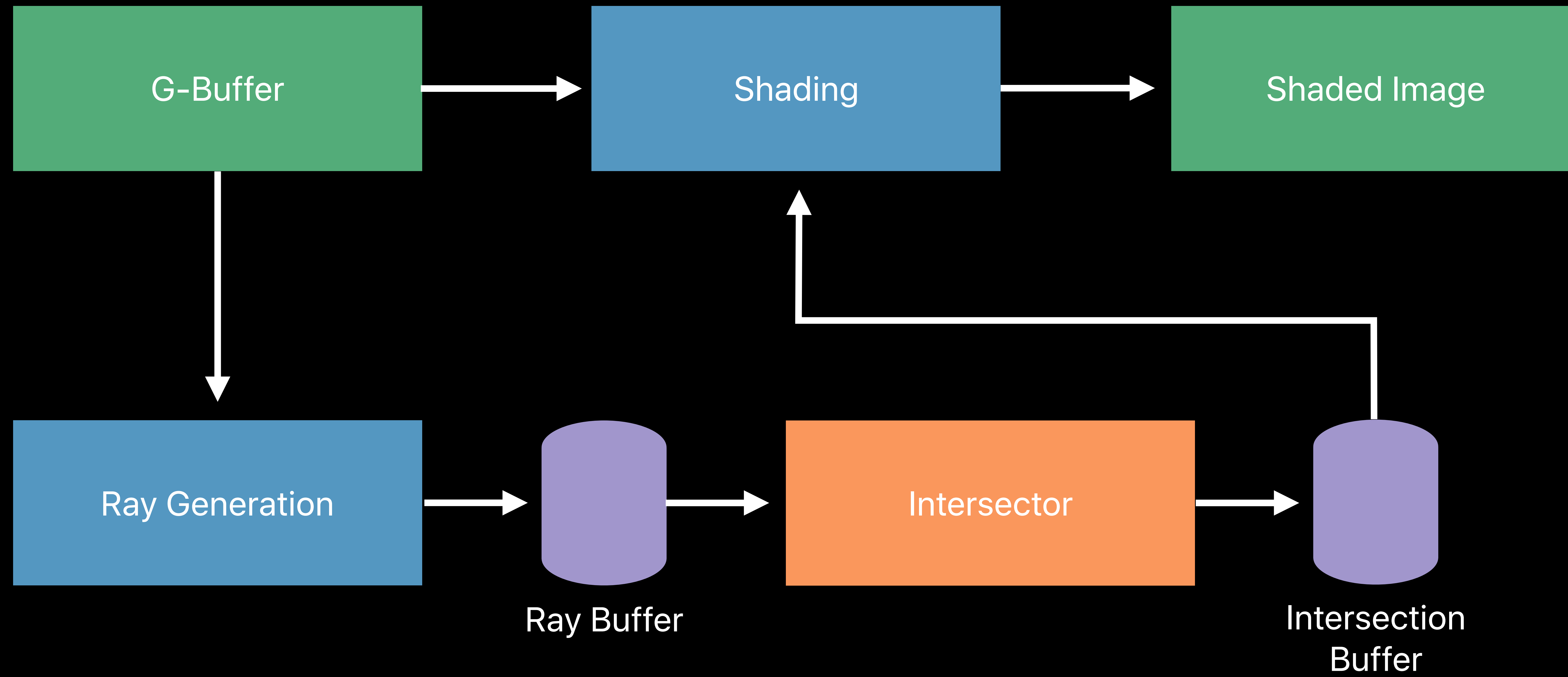
Hybrid Rendering



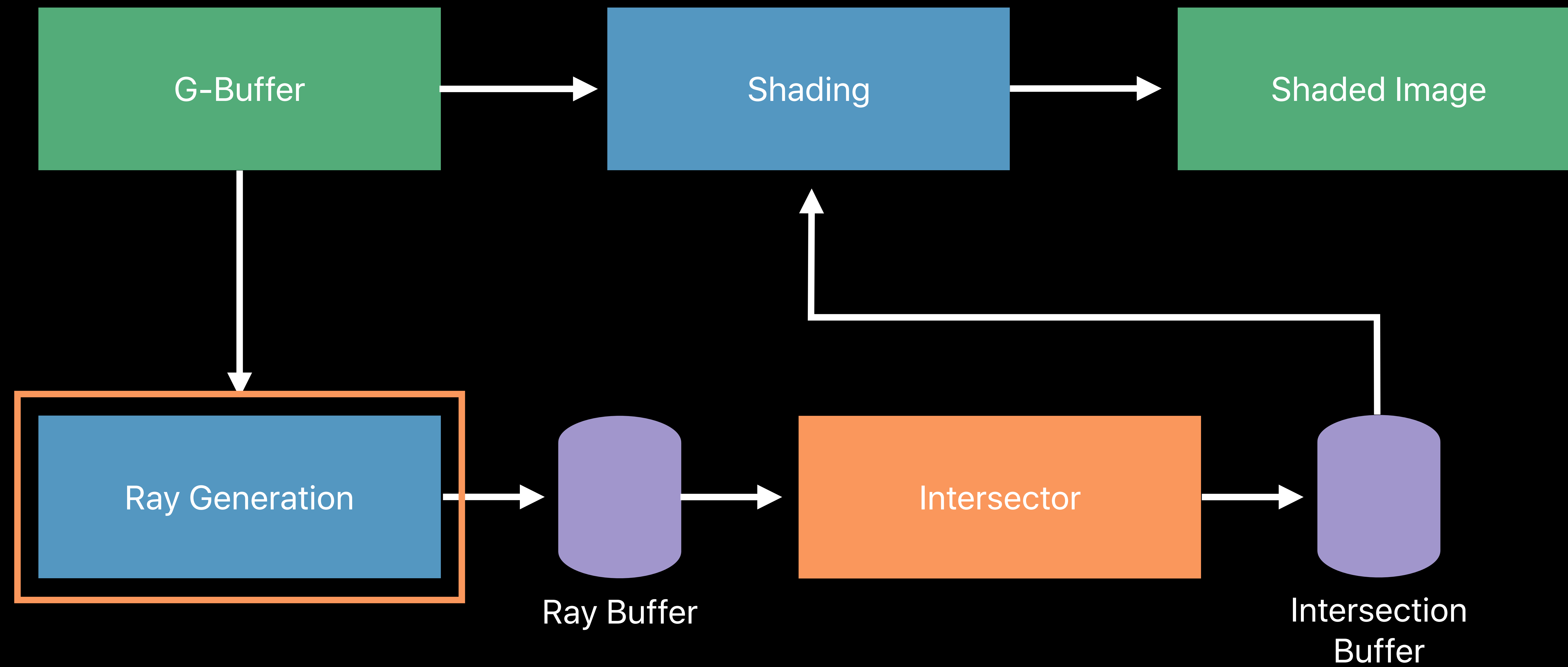
Hybrid Rendering



Hybrid Rendering



Hybrid Rendering



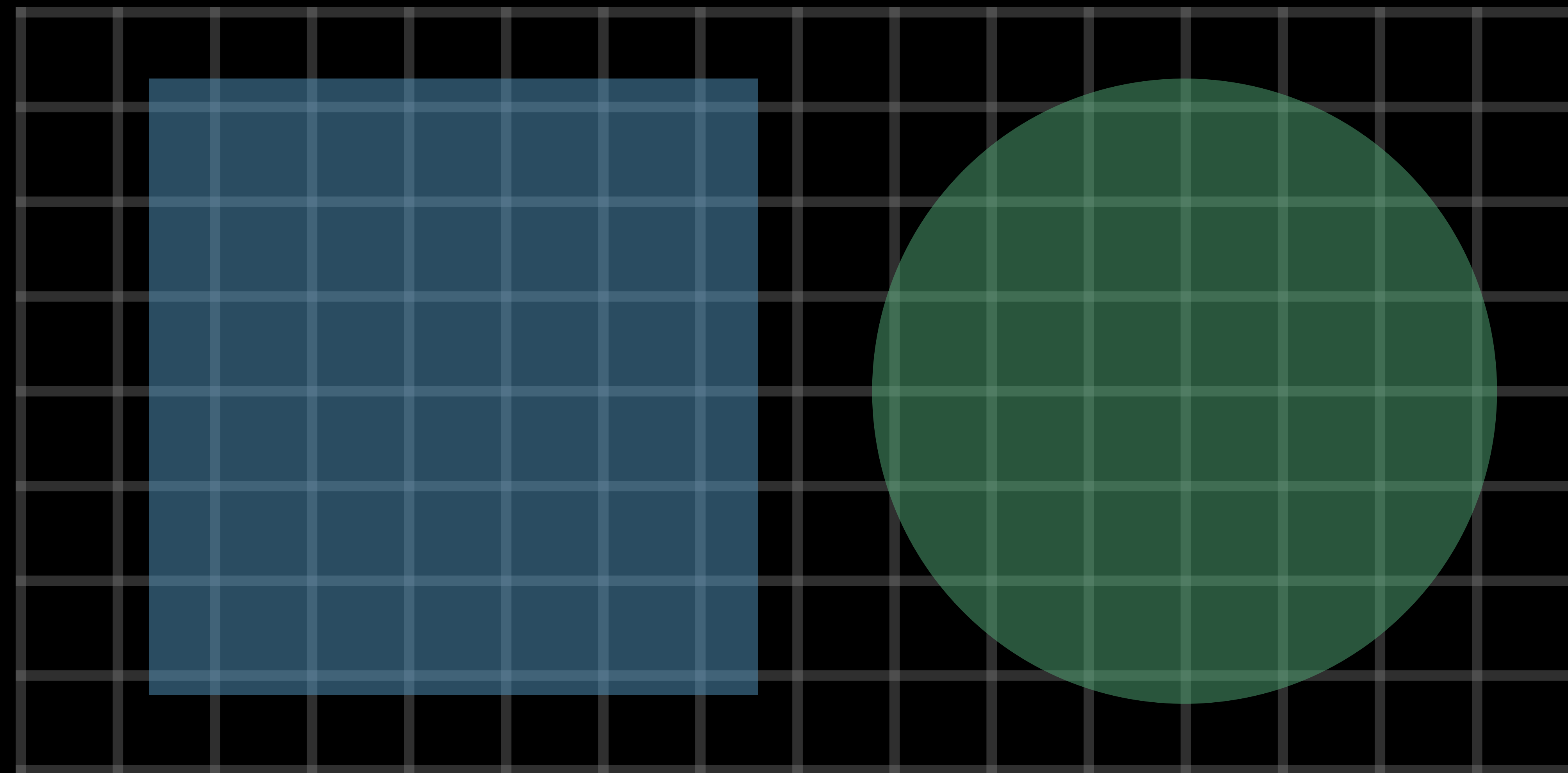
Ray Generation

```
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

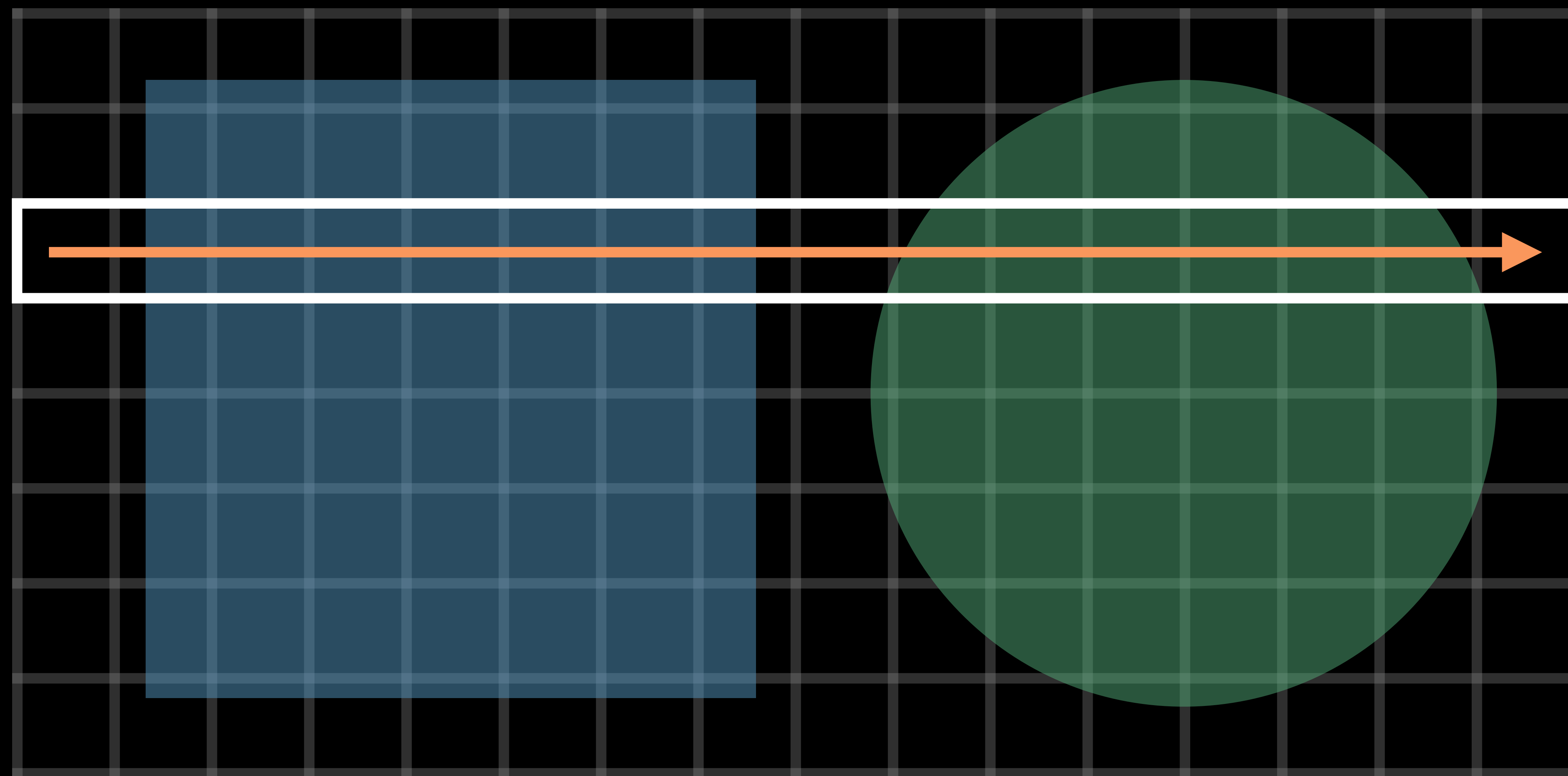


Row Linear

Ray Coherency

Metal processes rays in the order you specify them

Block linear layout can improve performance

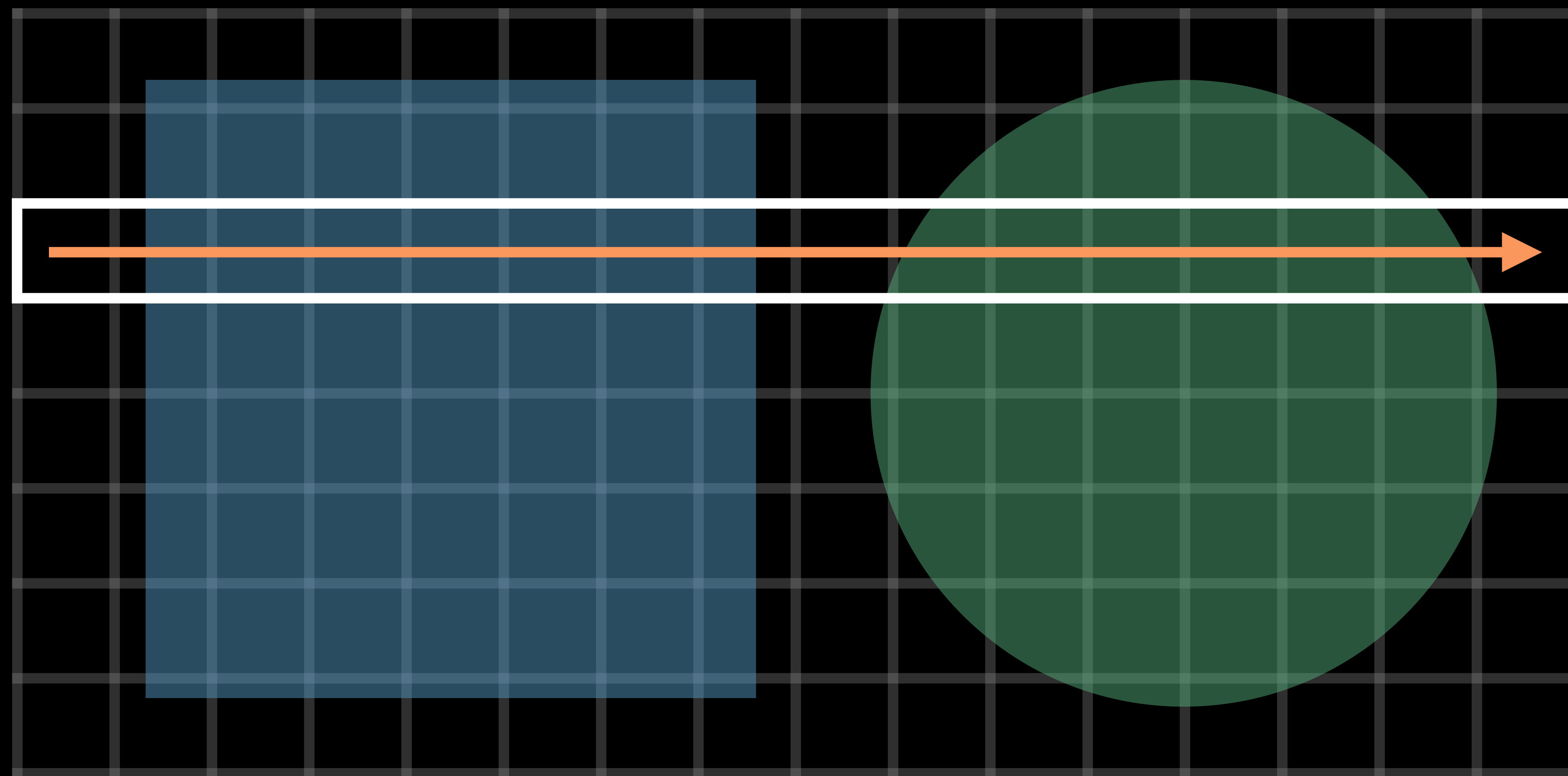


Row Linear

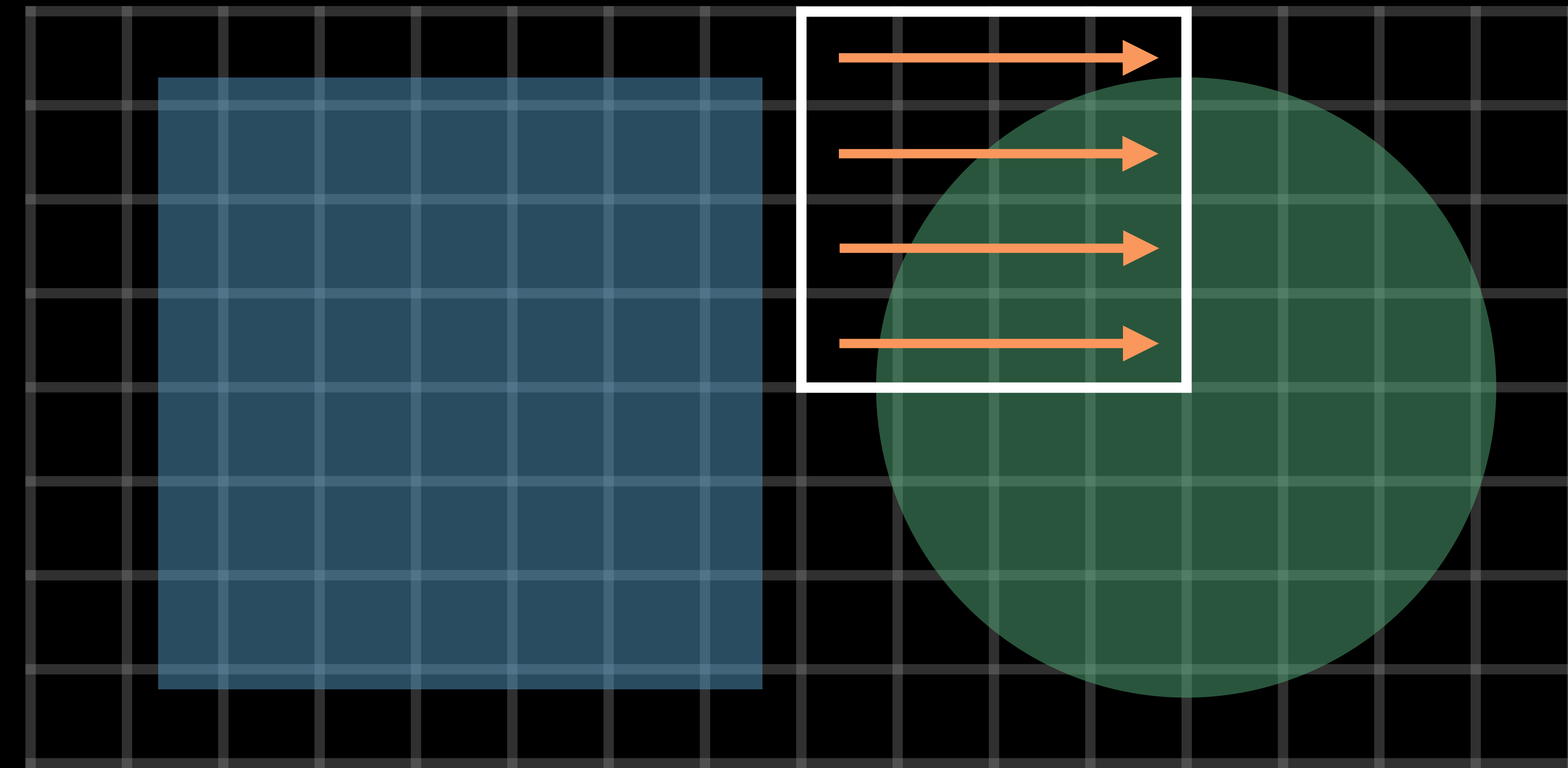
Ray Coherency

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



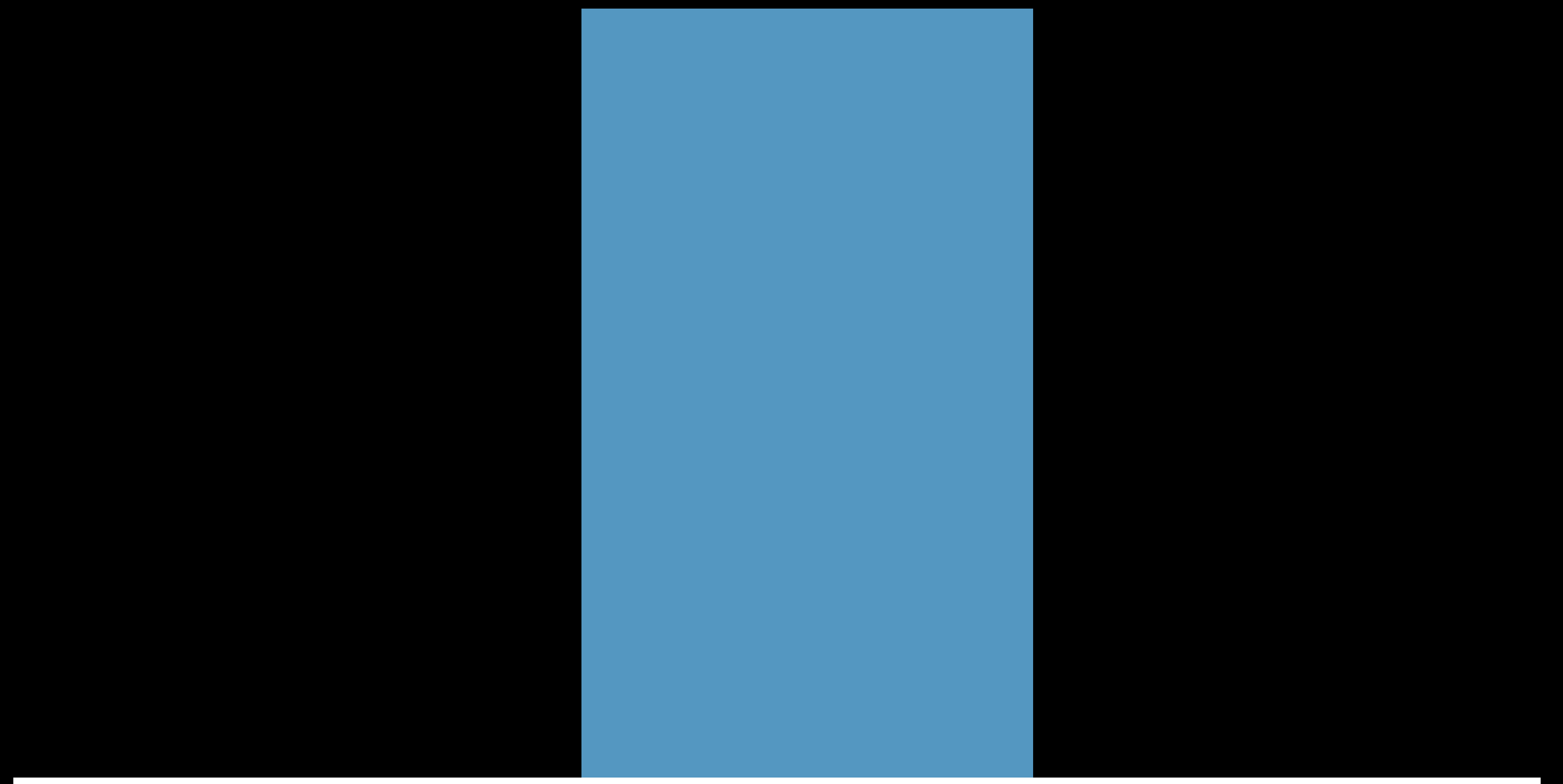
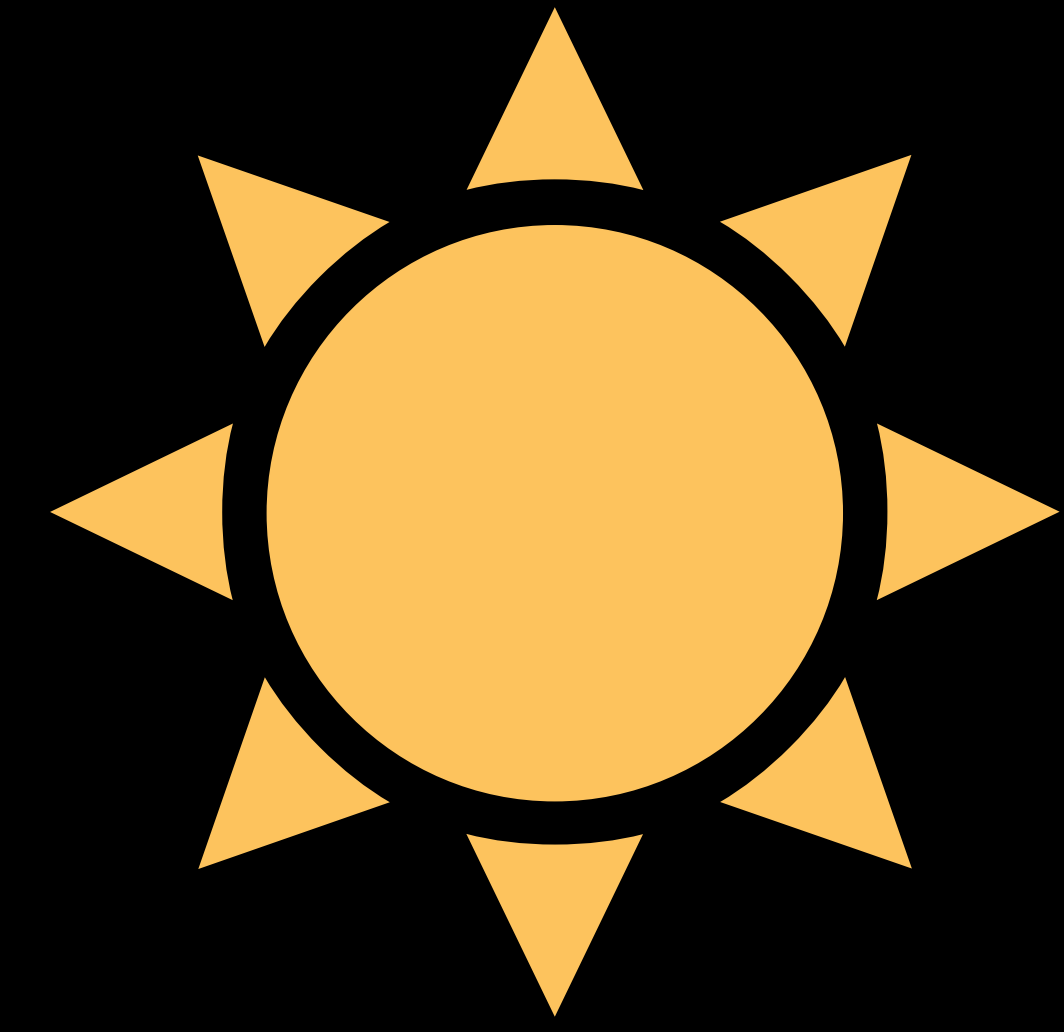
Soft Shadows



Soft Shadows

Extend a cone from the surface towards the Sun

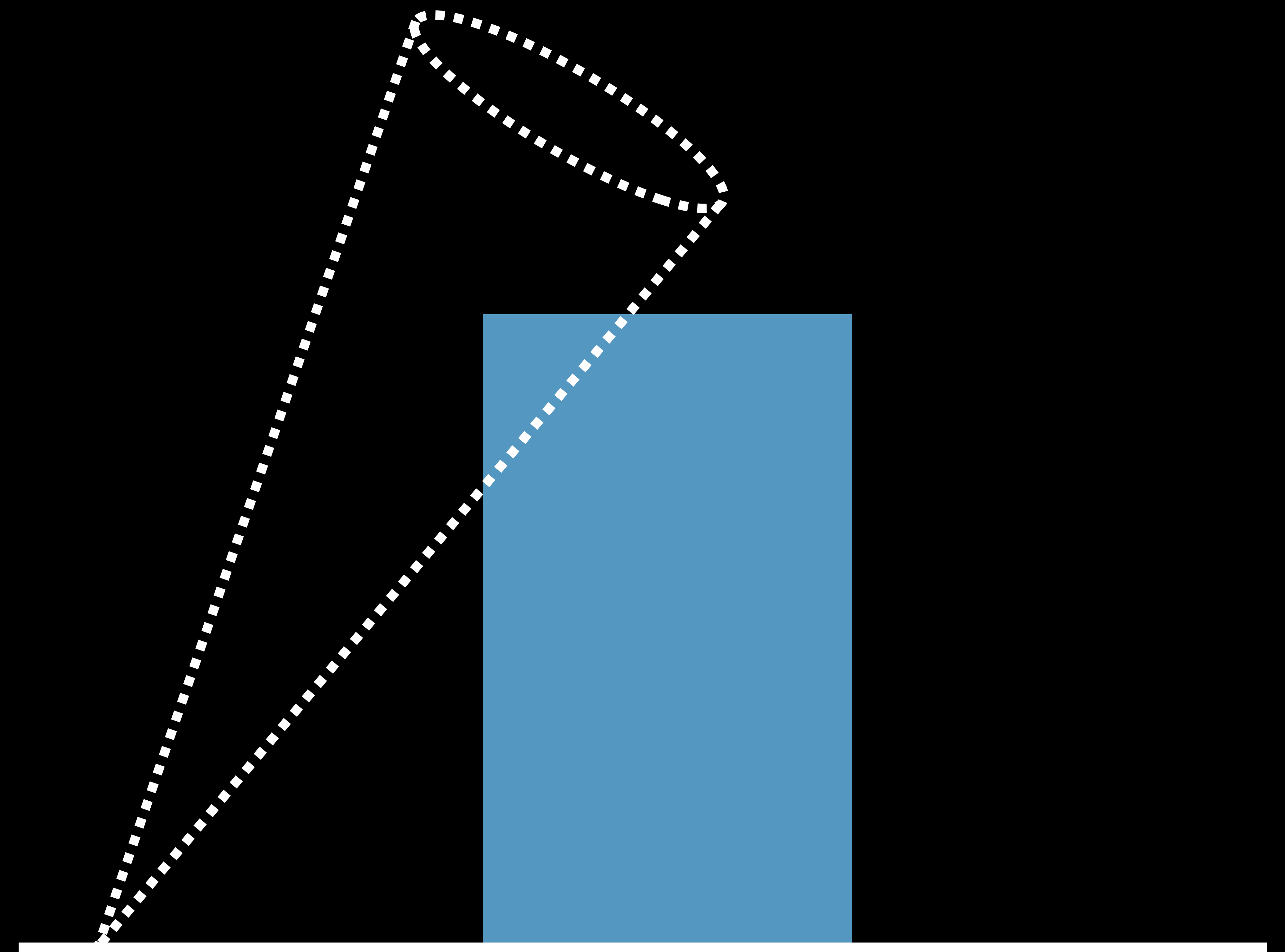
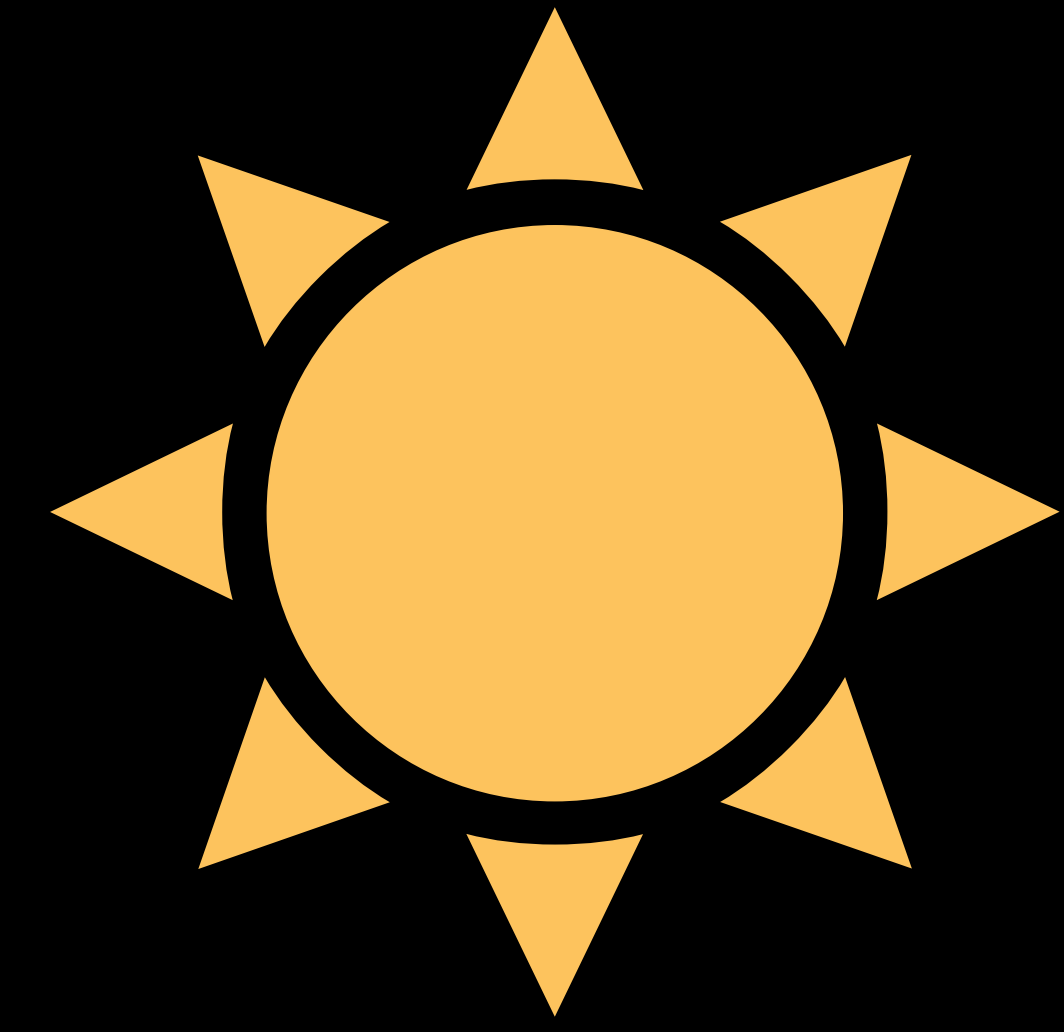
Generate ray directions randomly within
this cone



Soft Shadows

Extend a cone from the surface towards the Sun

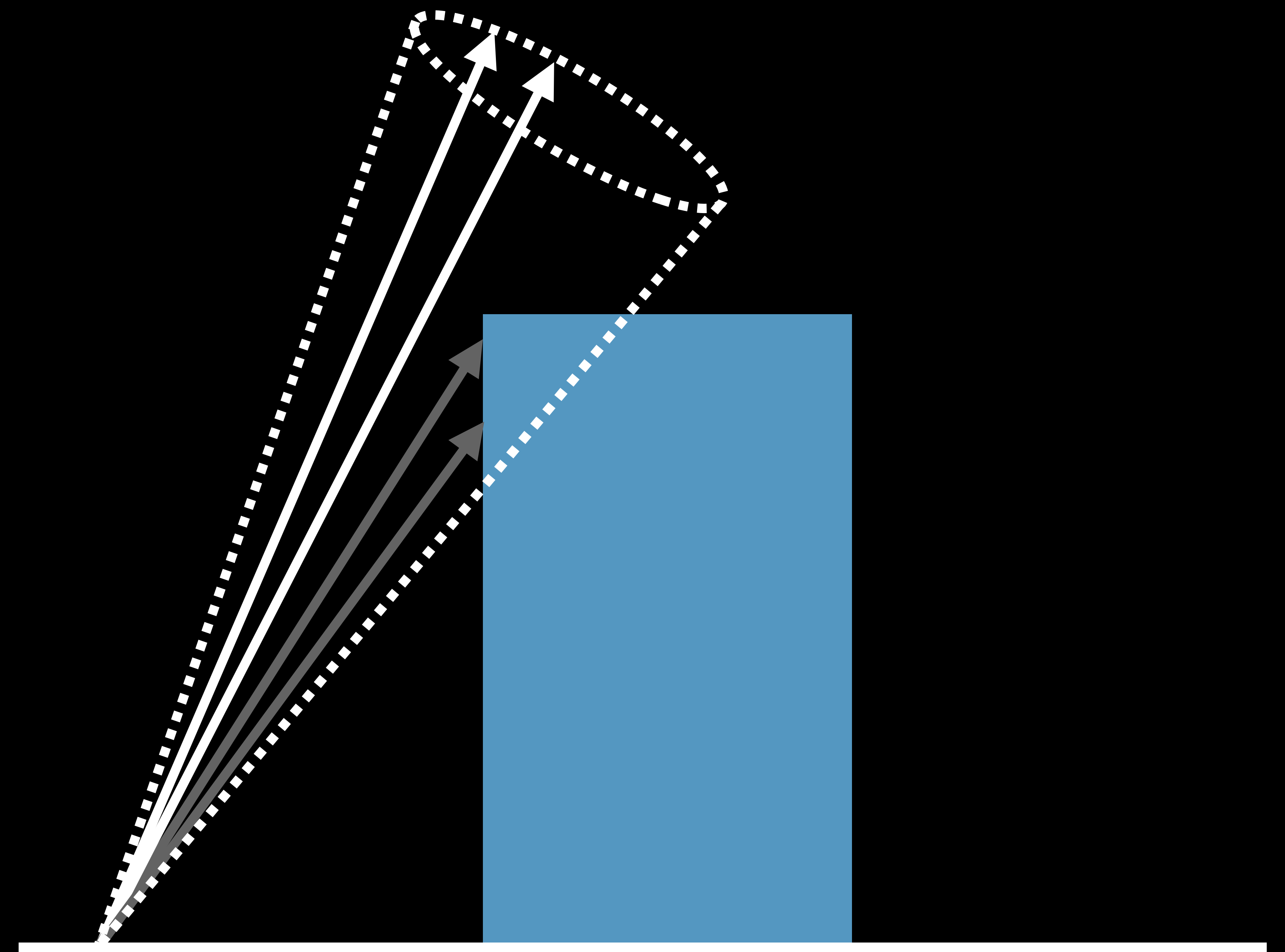
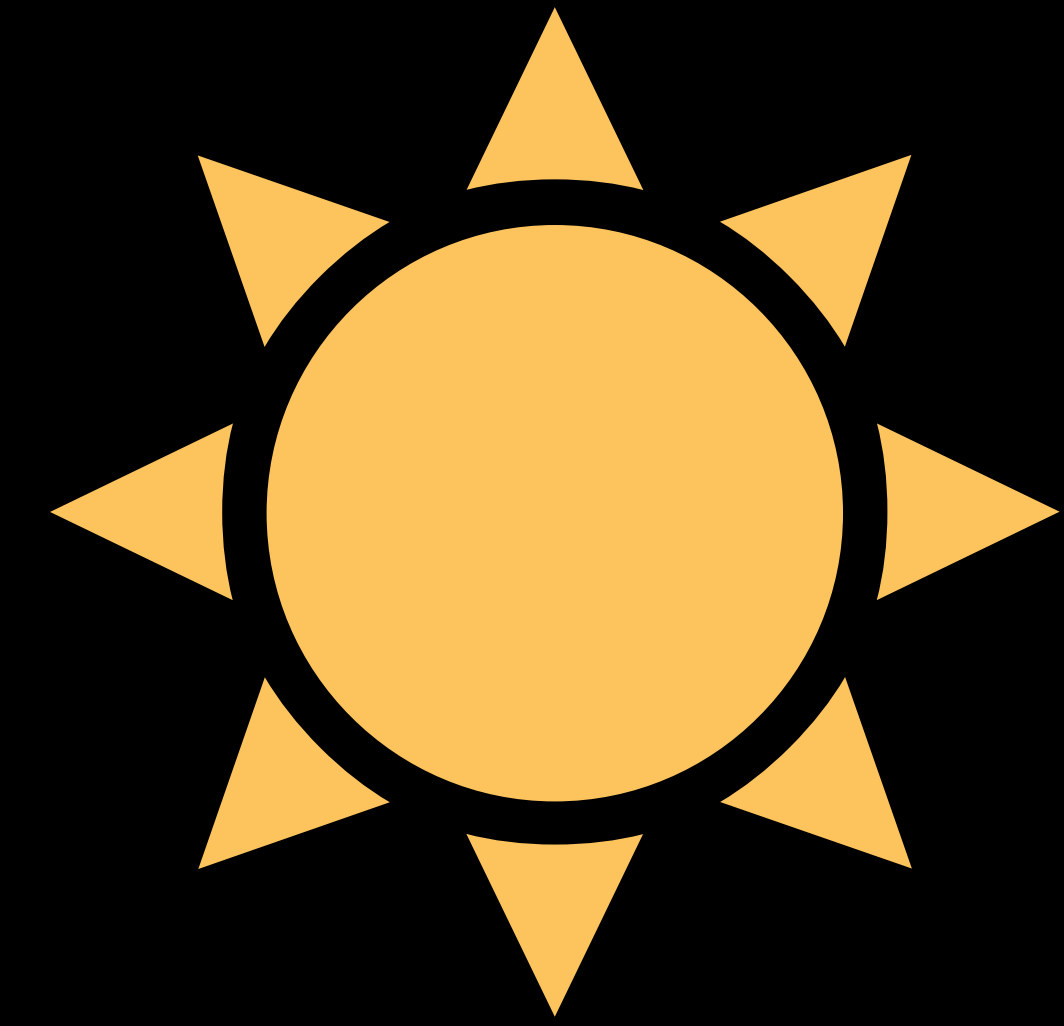
Generate ray directions randomly within
this cone



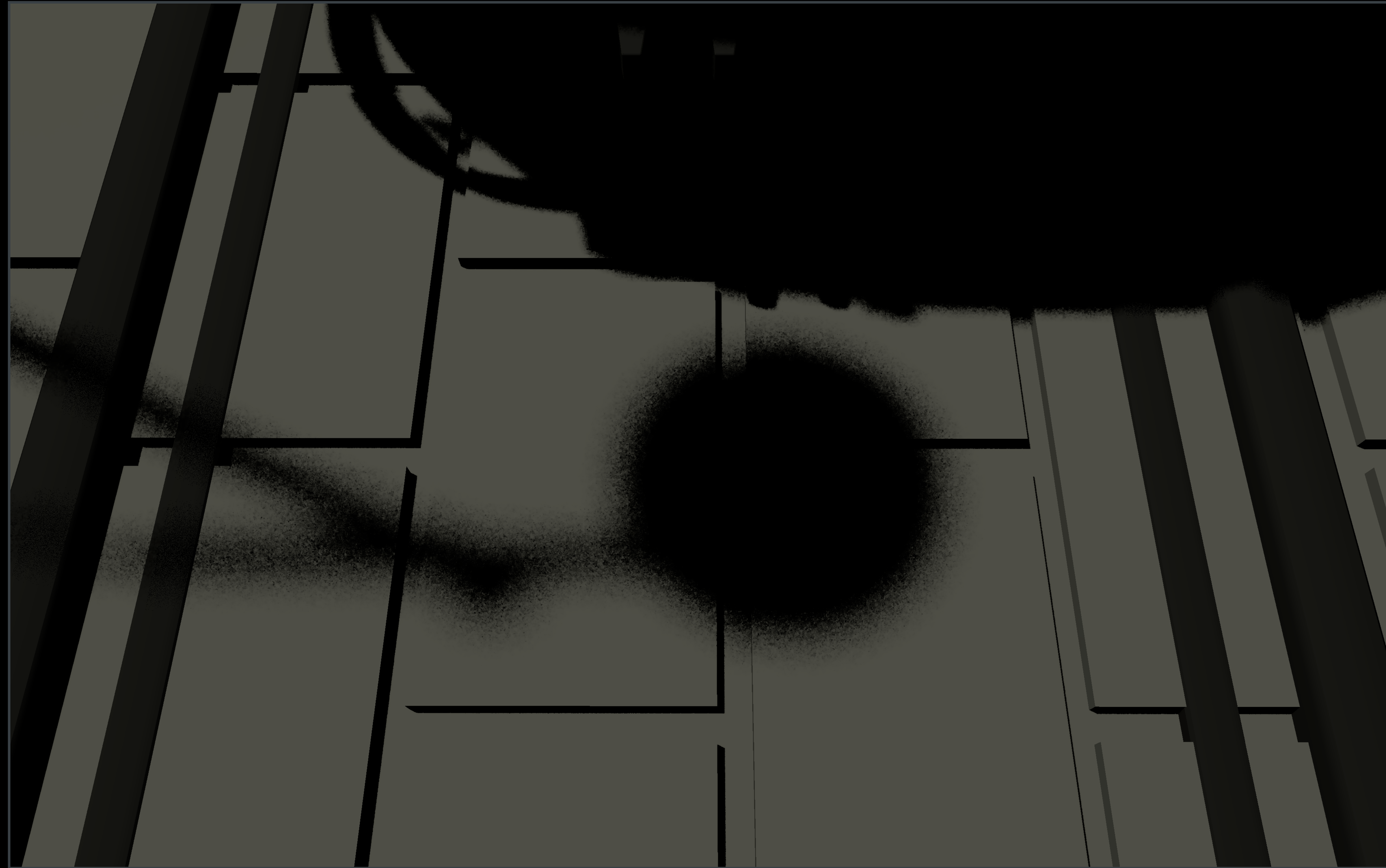
Soft Shadows

Extend a cone from the surface towards the Sun

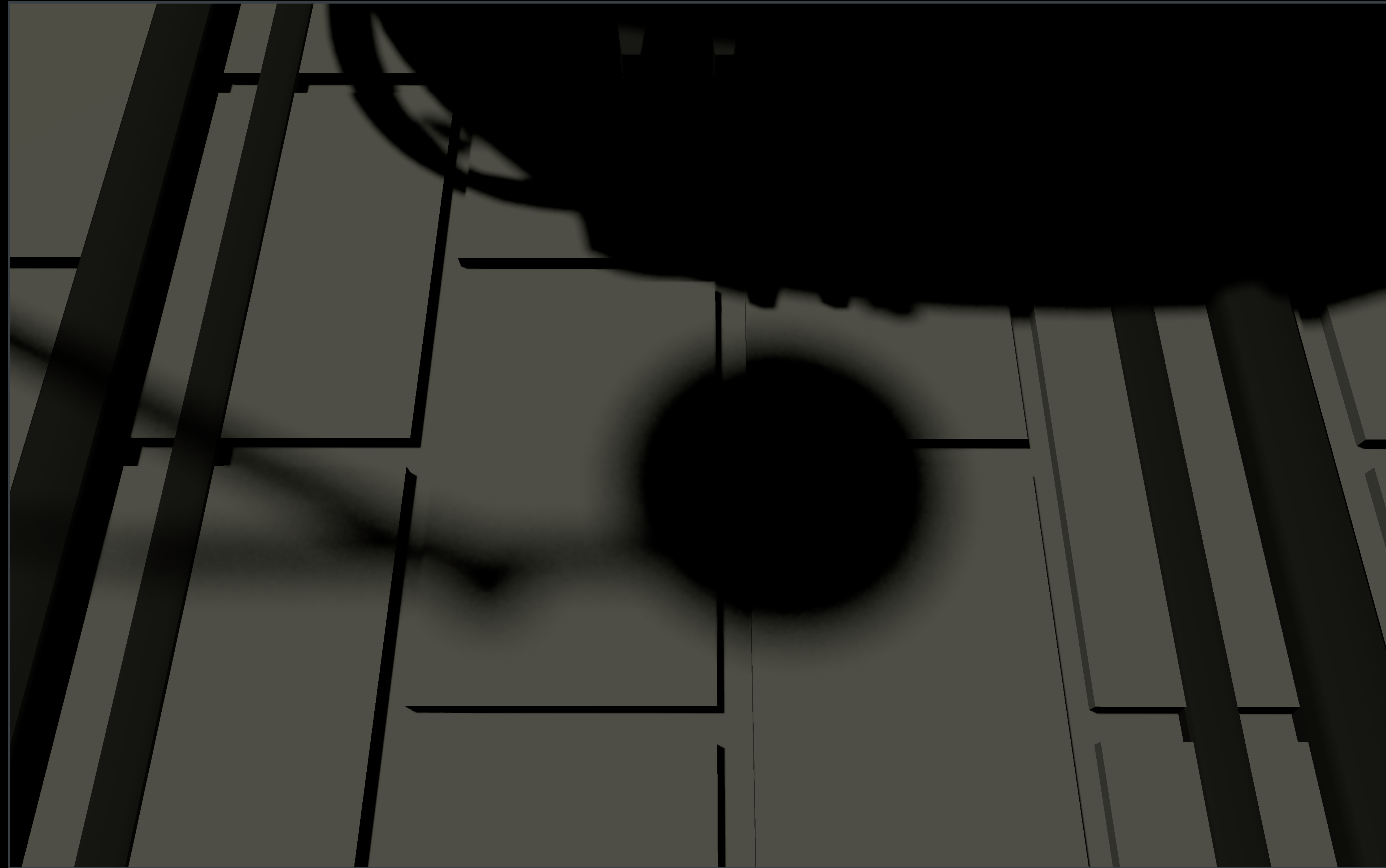
Generate ray directions randomly within this cone



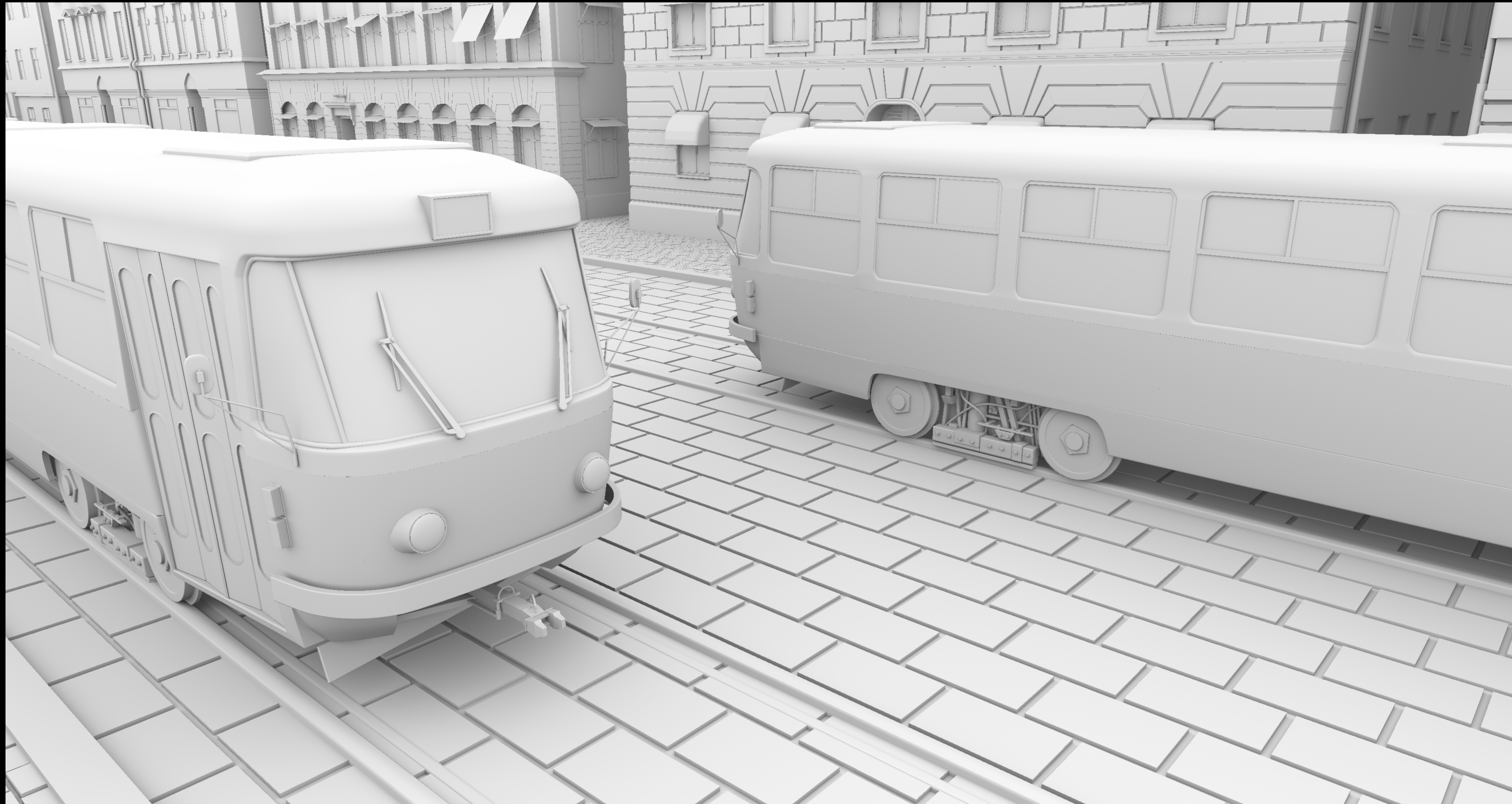
Shadow Term



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

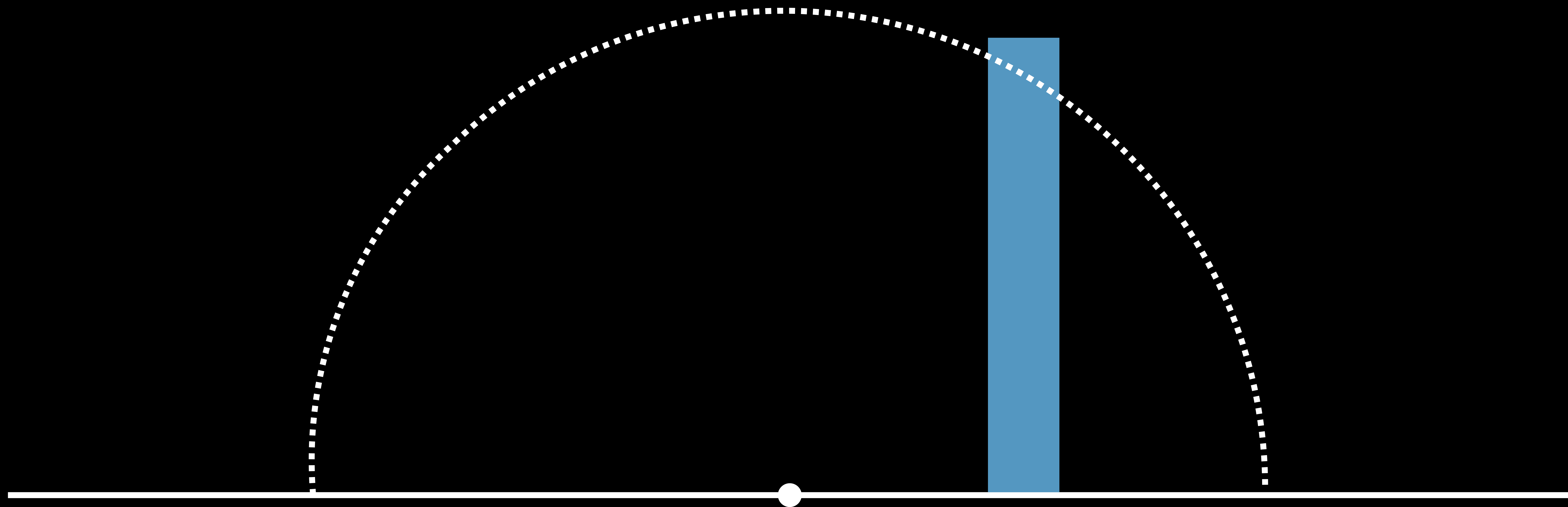
Falloff based on angle and intersection distance



Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

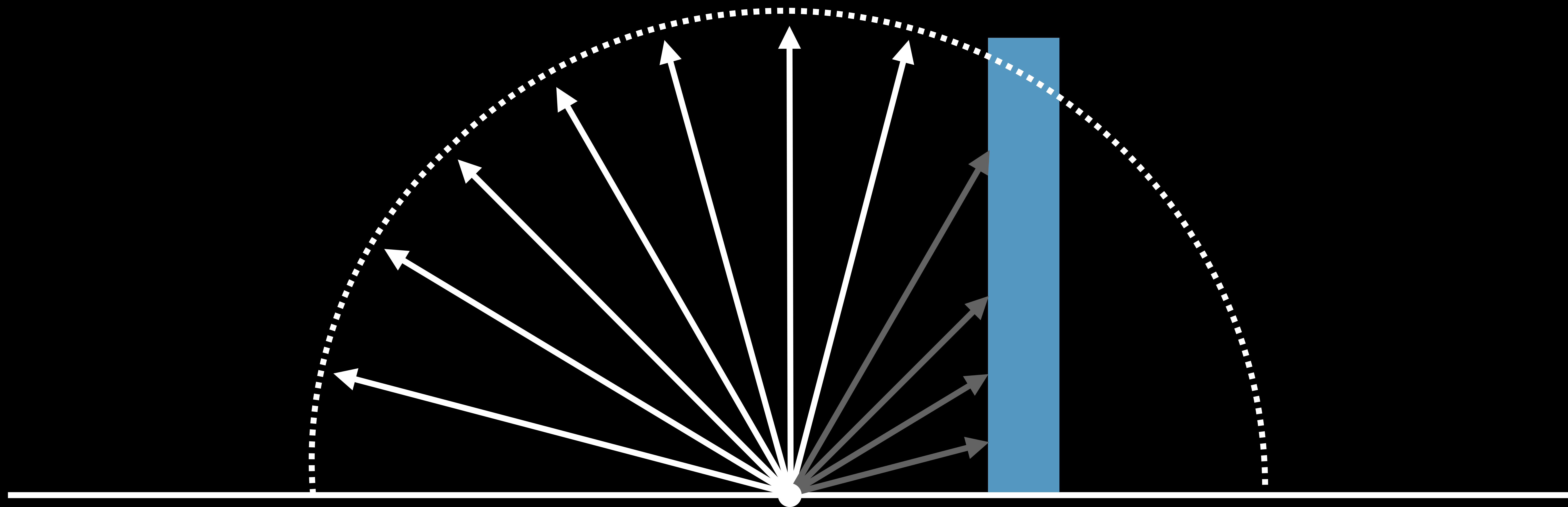
Falloff based on angle and intersection distance



Ambient Occlusion

Use rays to estimate how much ambient light reaches a surface

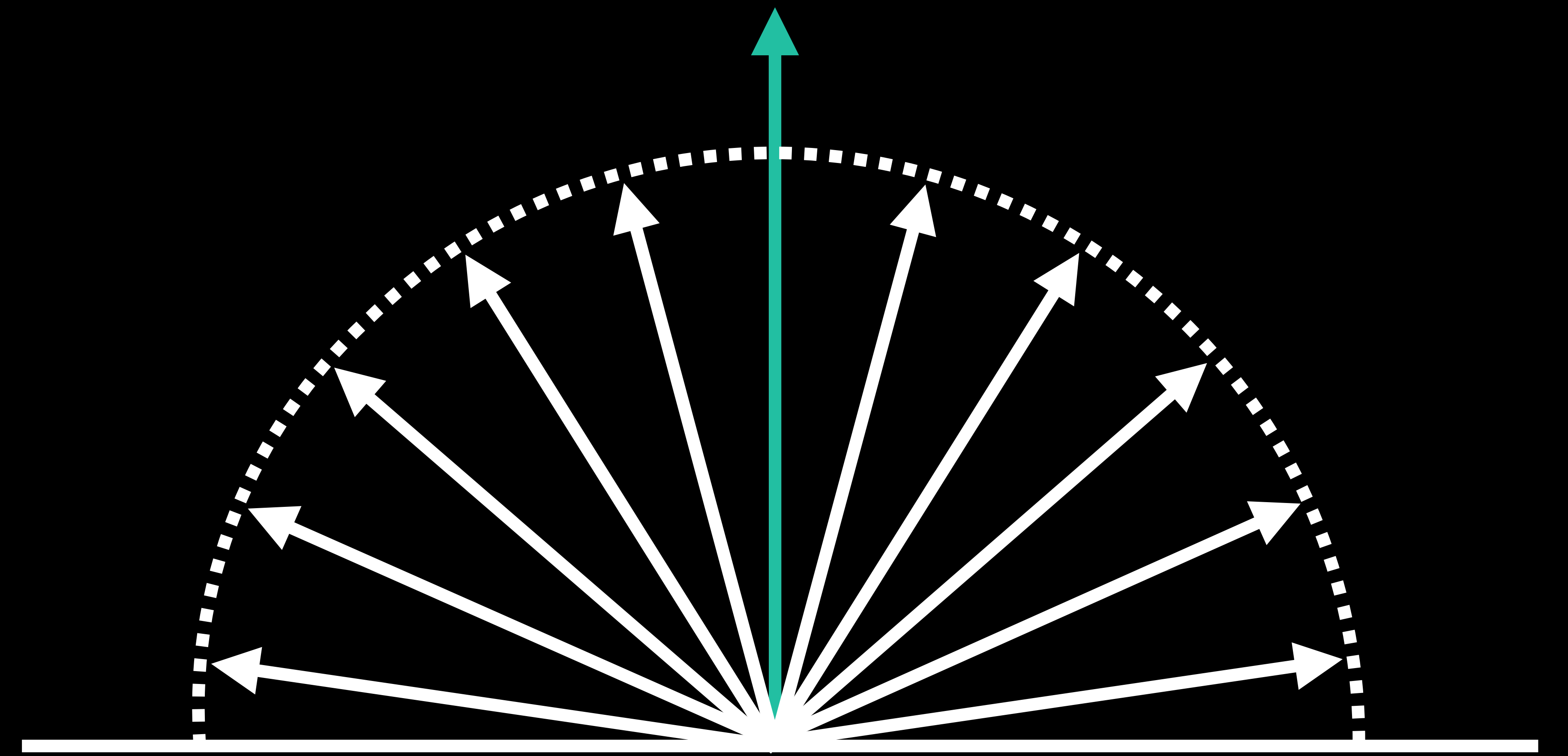
Falloff based on angle and intersection distance



Importance Sampling

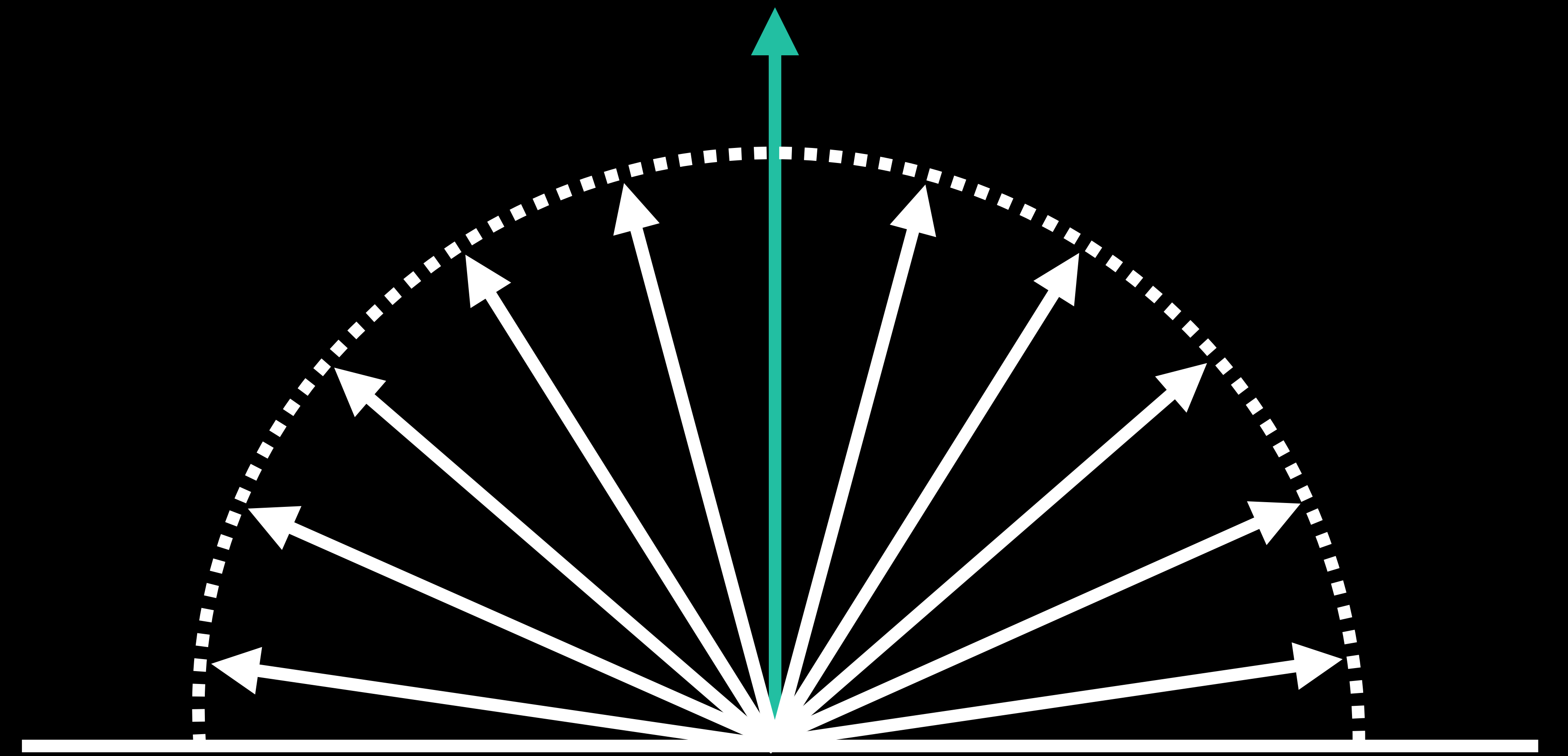
Use importance sampling to generate rays

Fewer rays for same visual quality



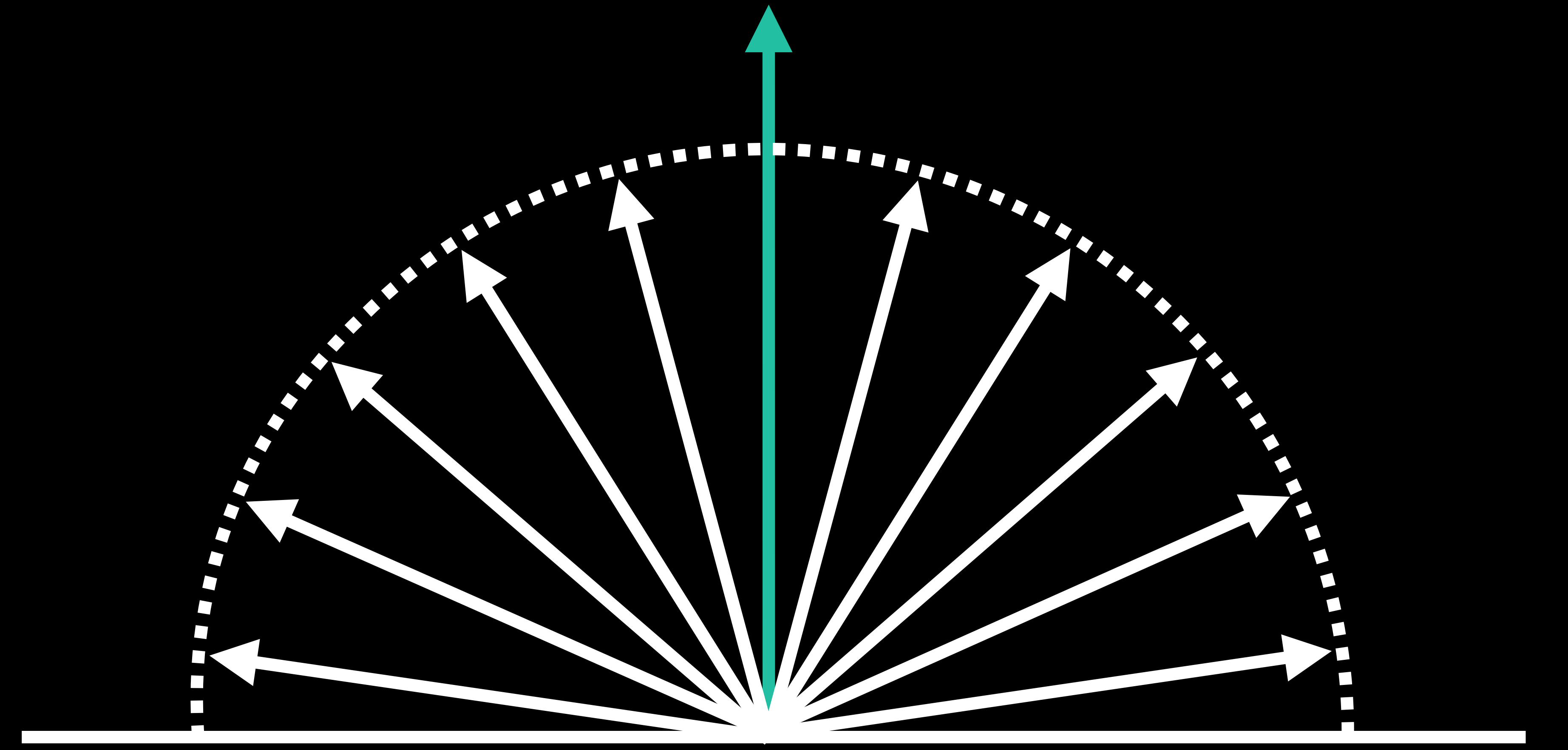
Importance Sampling

Hemisphere sampling



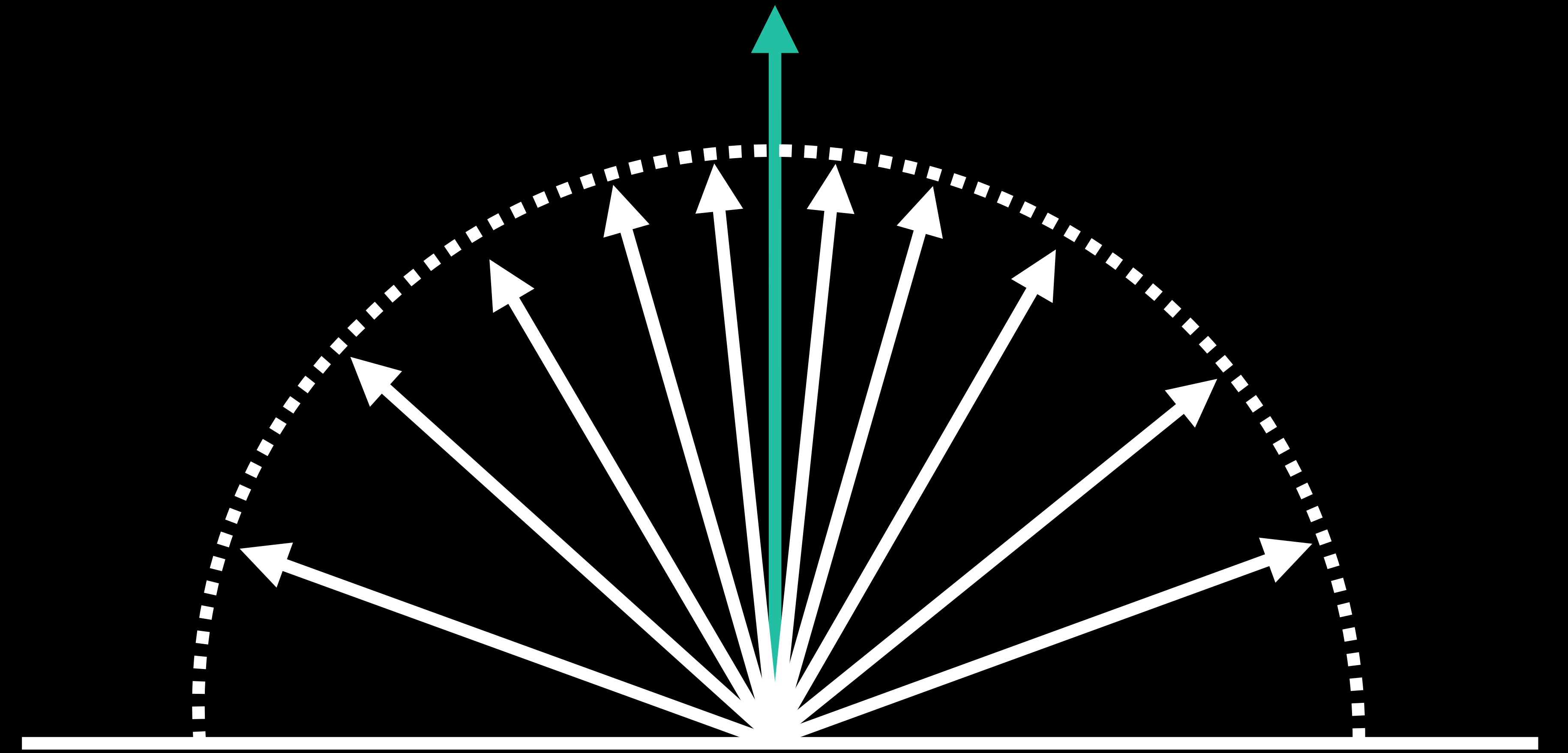
Importance Sampling

Cosine sampling



Importance Sampling

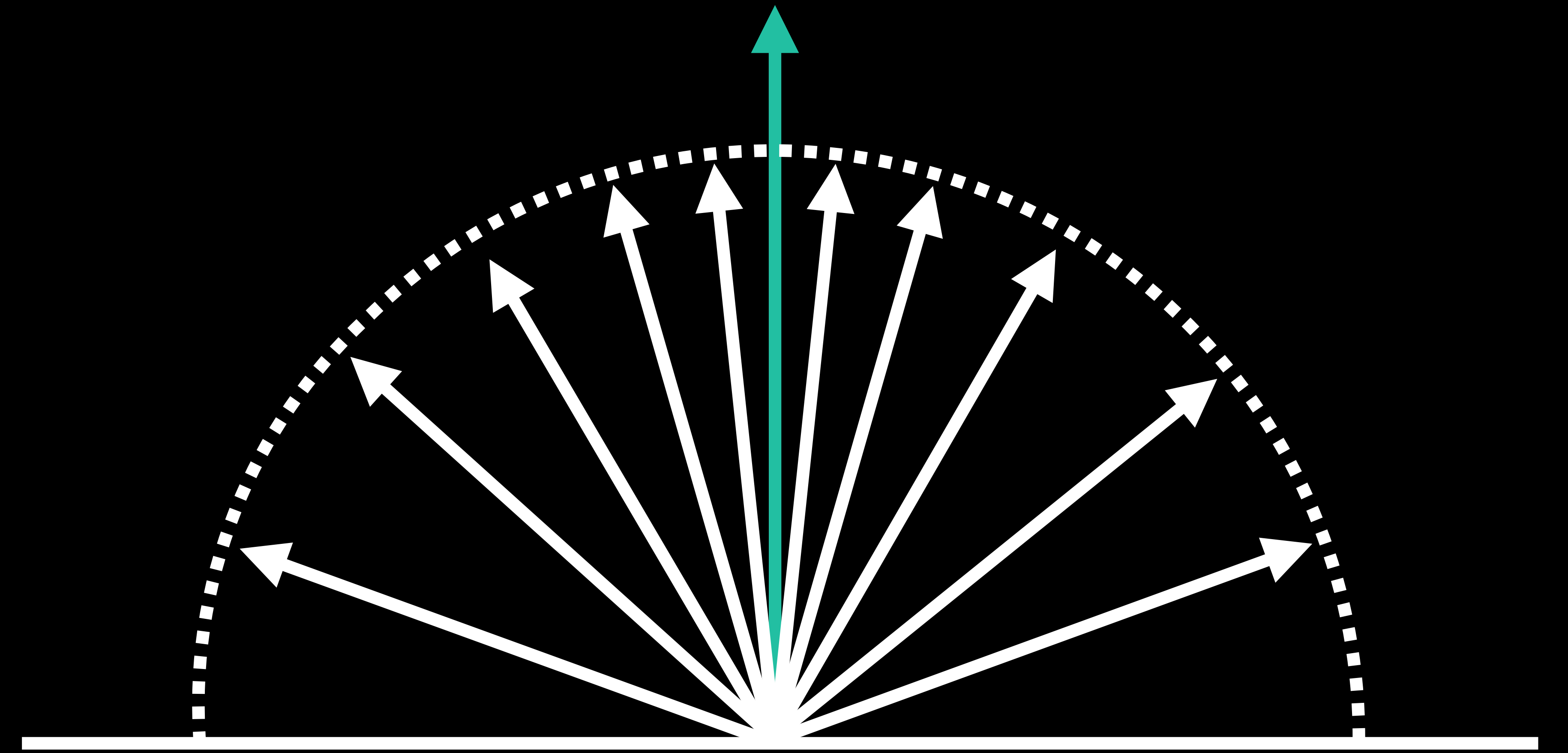
Cosine sampling



Importance Sampling

Cosine sampling

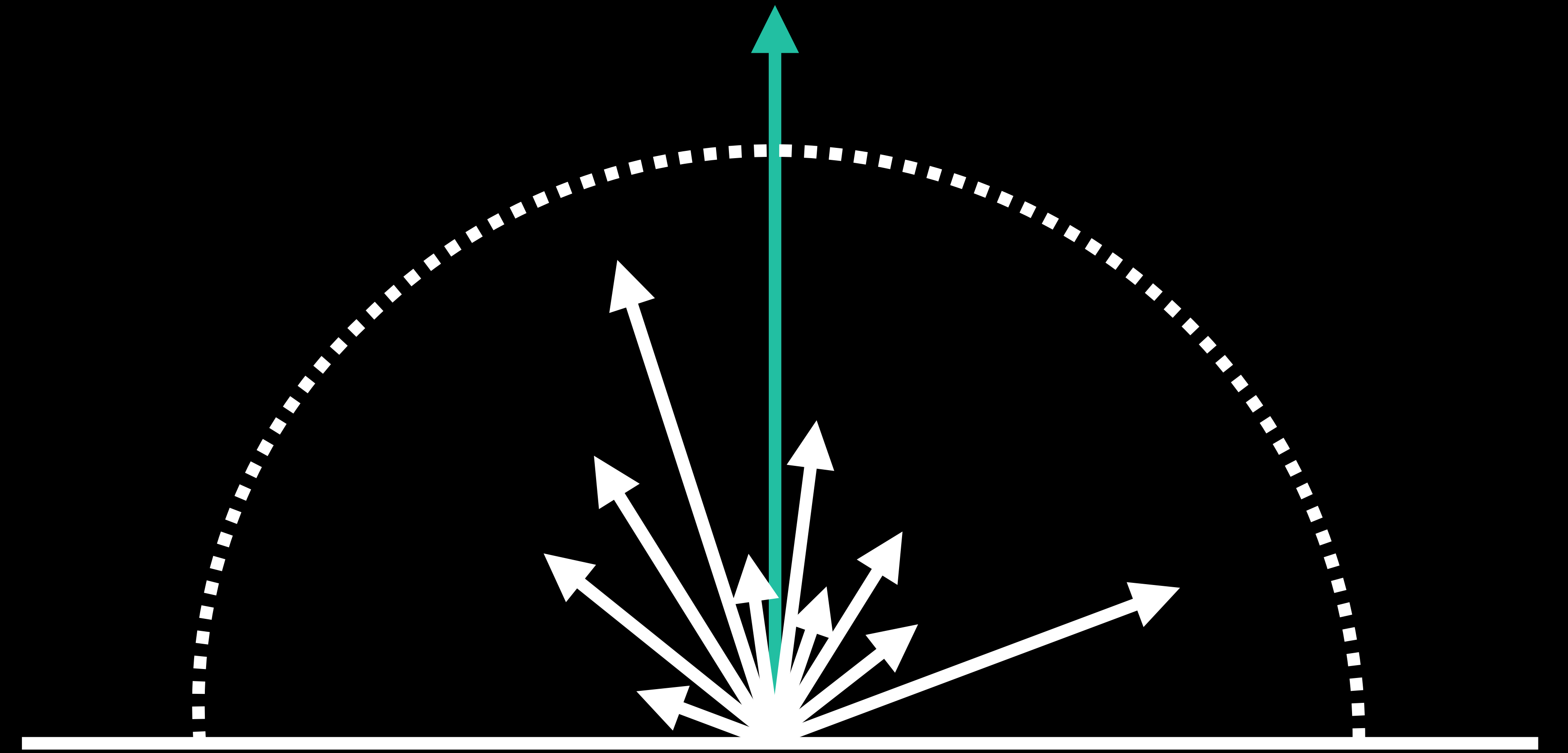
Distance sampling



Importance Sampling

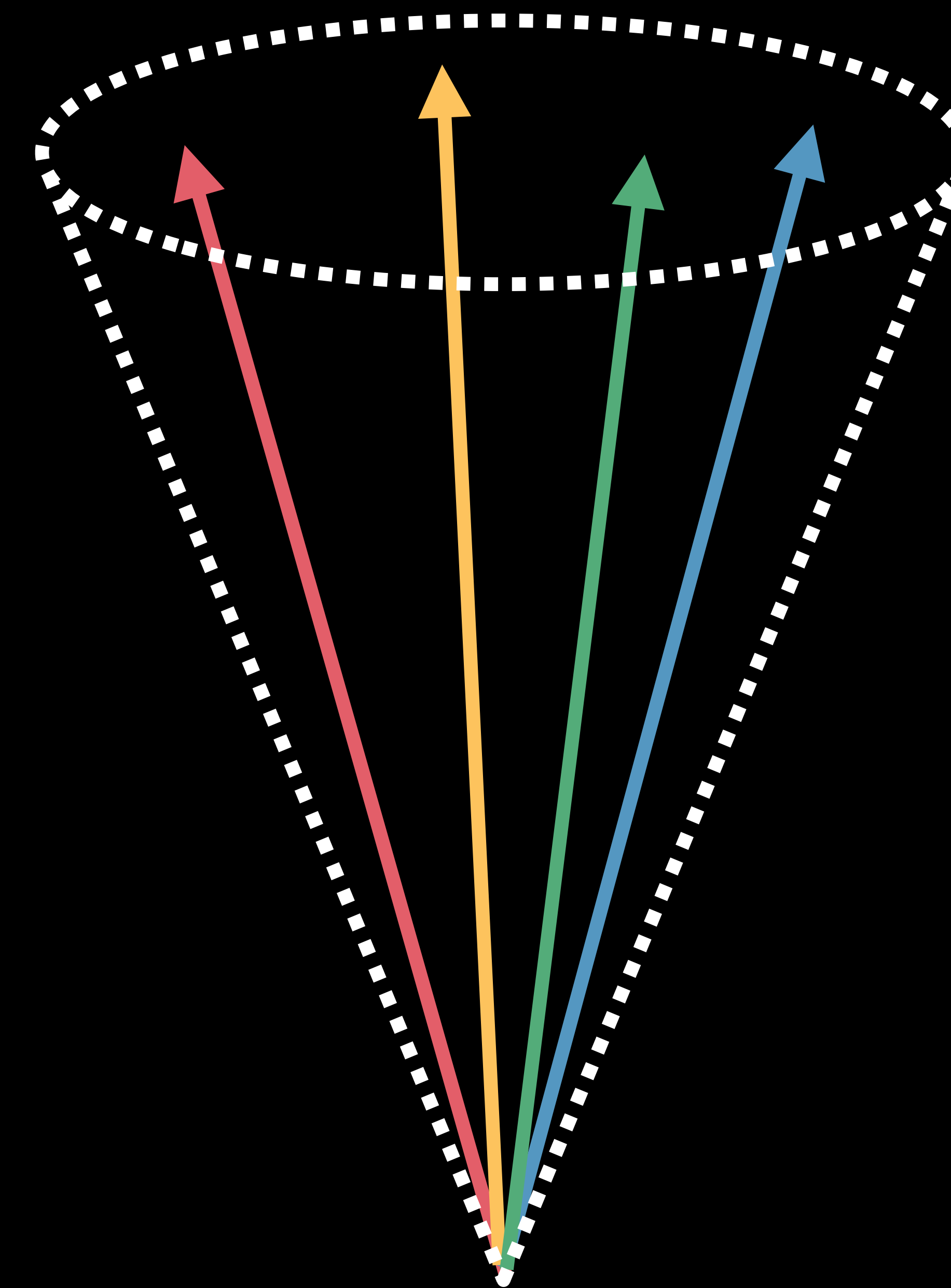
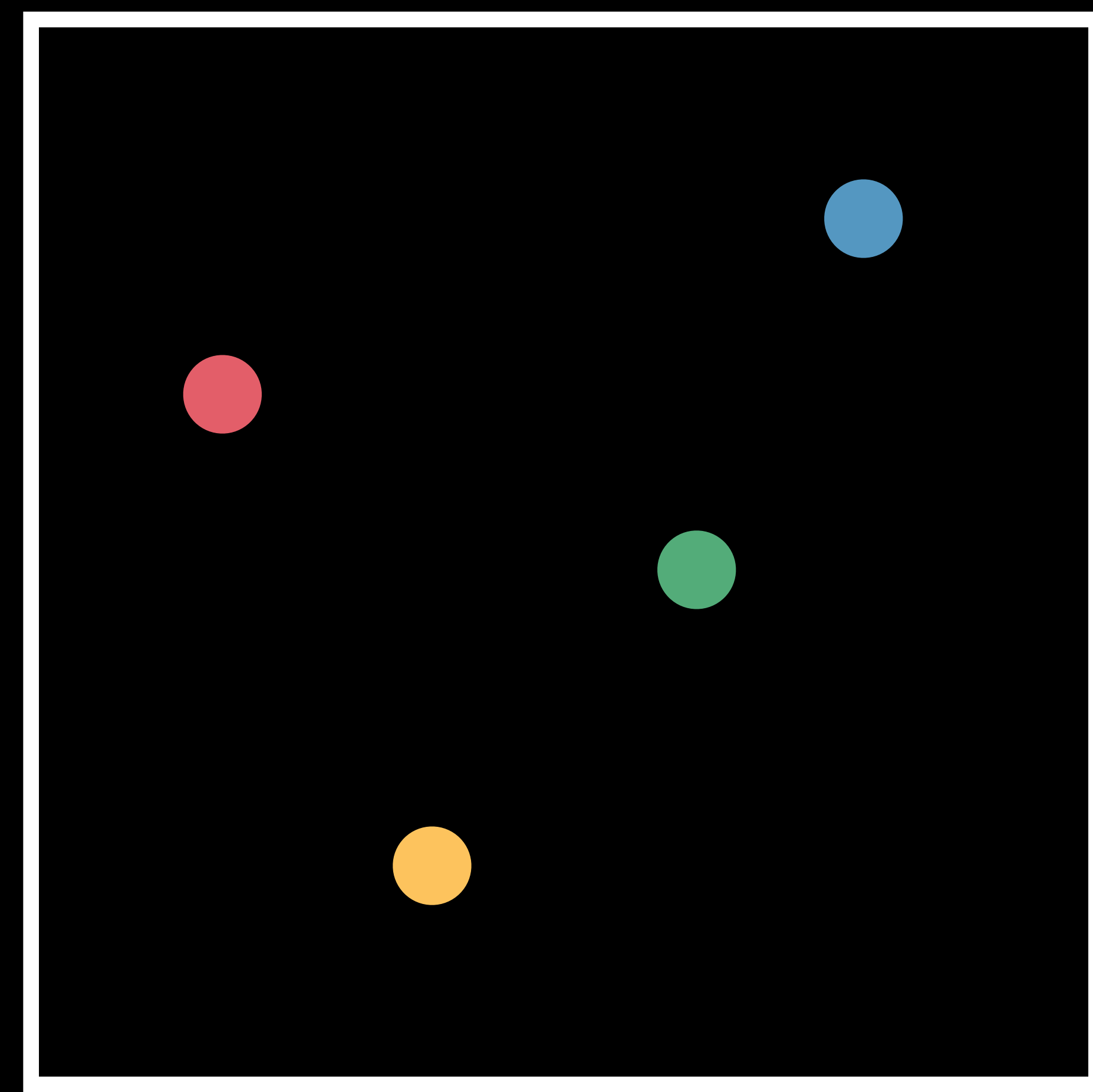
Cosine sampling

Distance sampling



Parameter Space

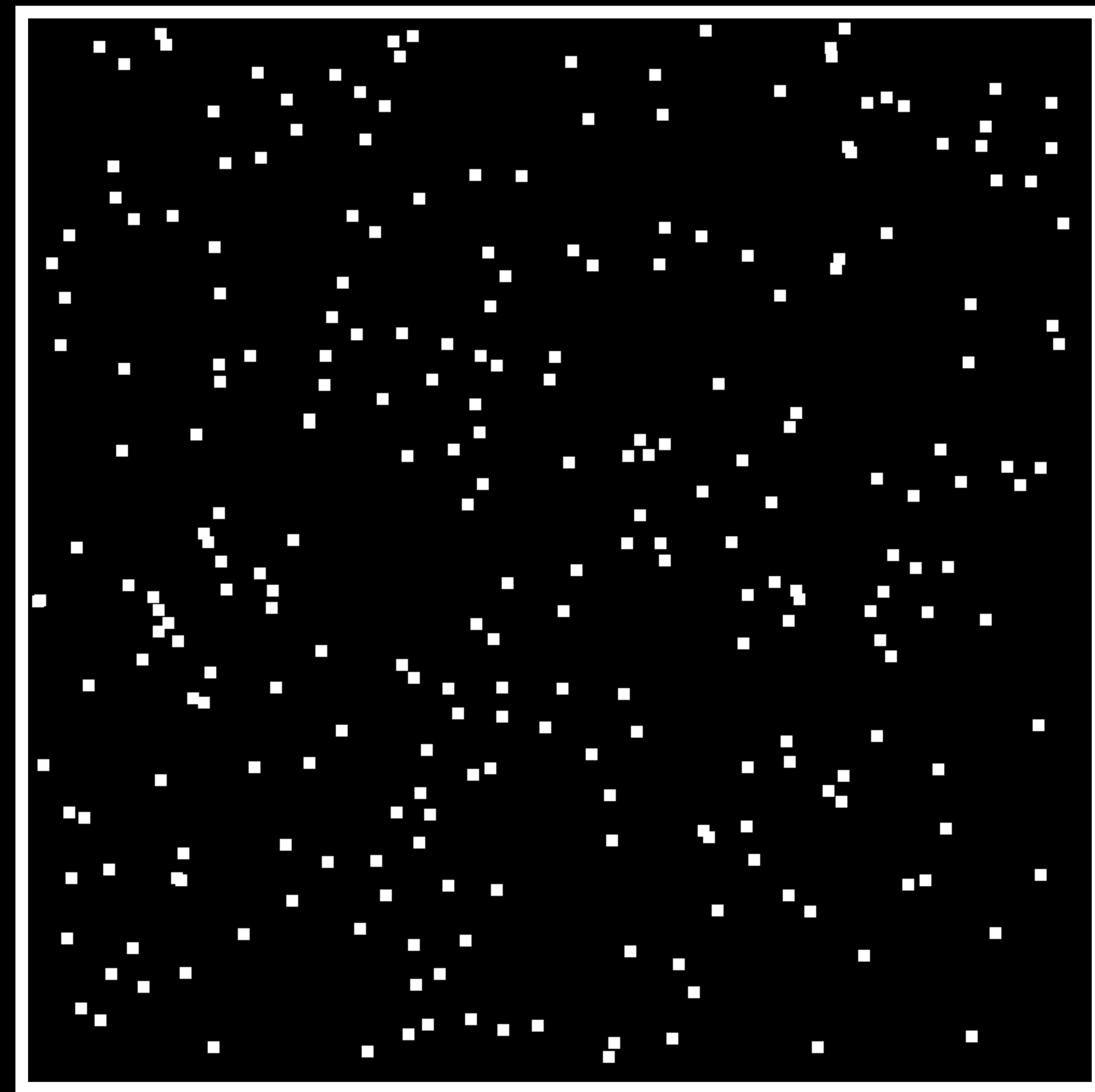
Points in 2D parameter space map to 3D ray distributions



Low Discrepancy Sequences

Distribute points evenly in parameter space

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

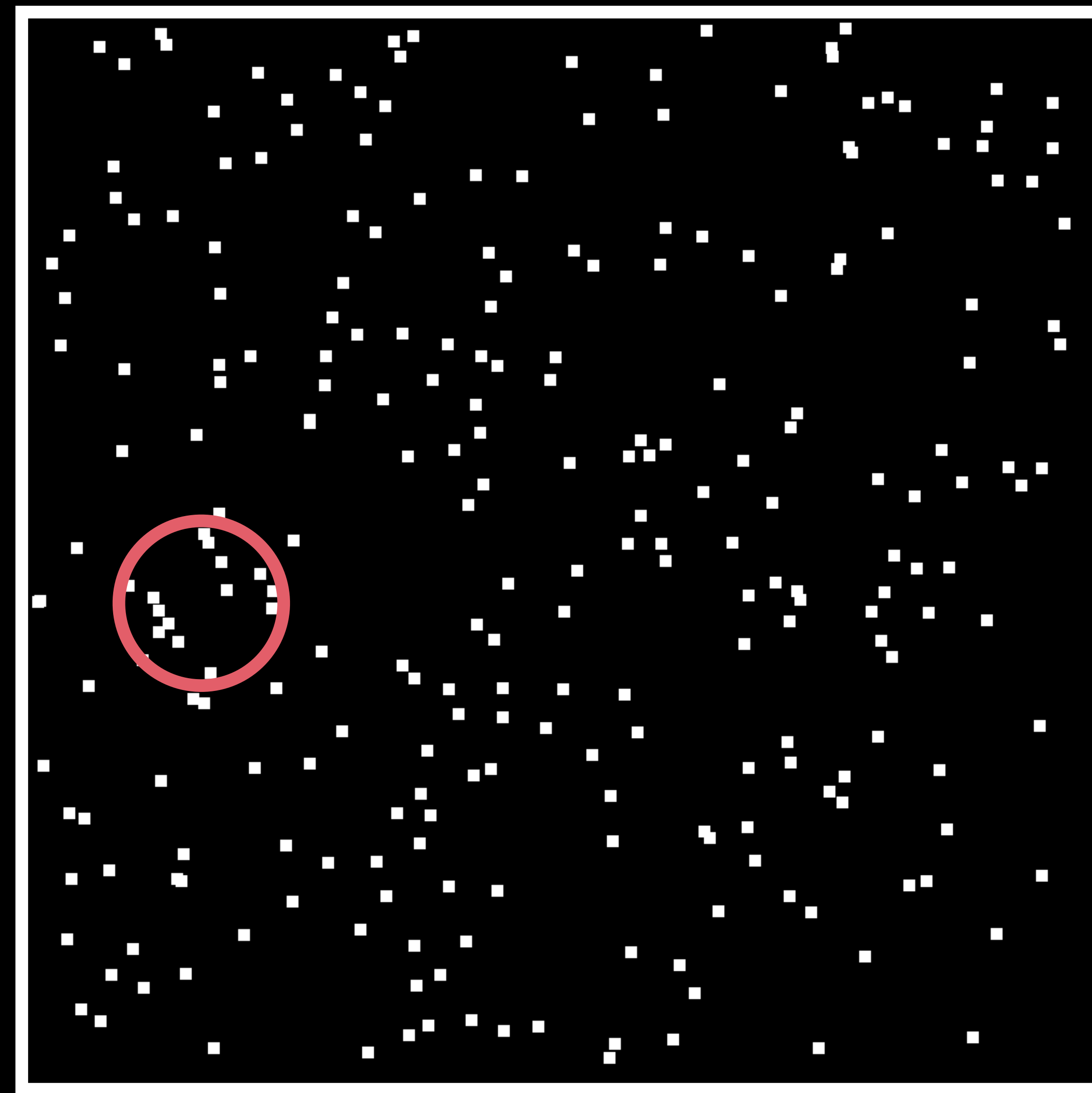


Random

Low Discrepancy Sequences

Distribute points evenly in parameter space

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

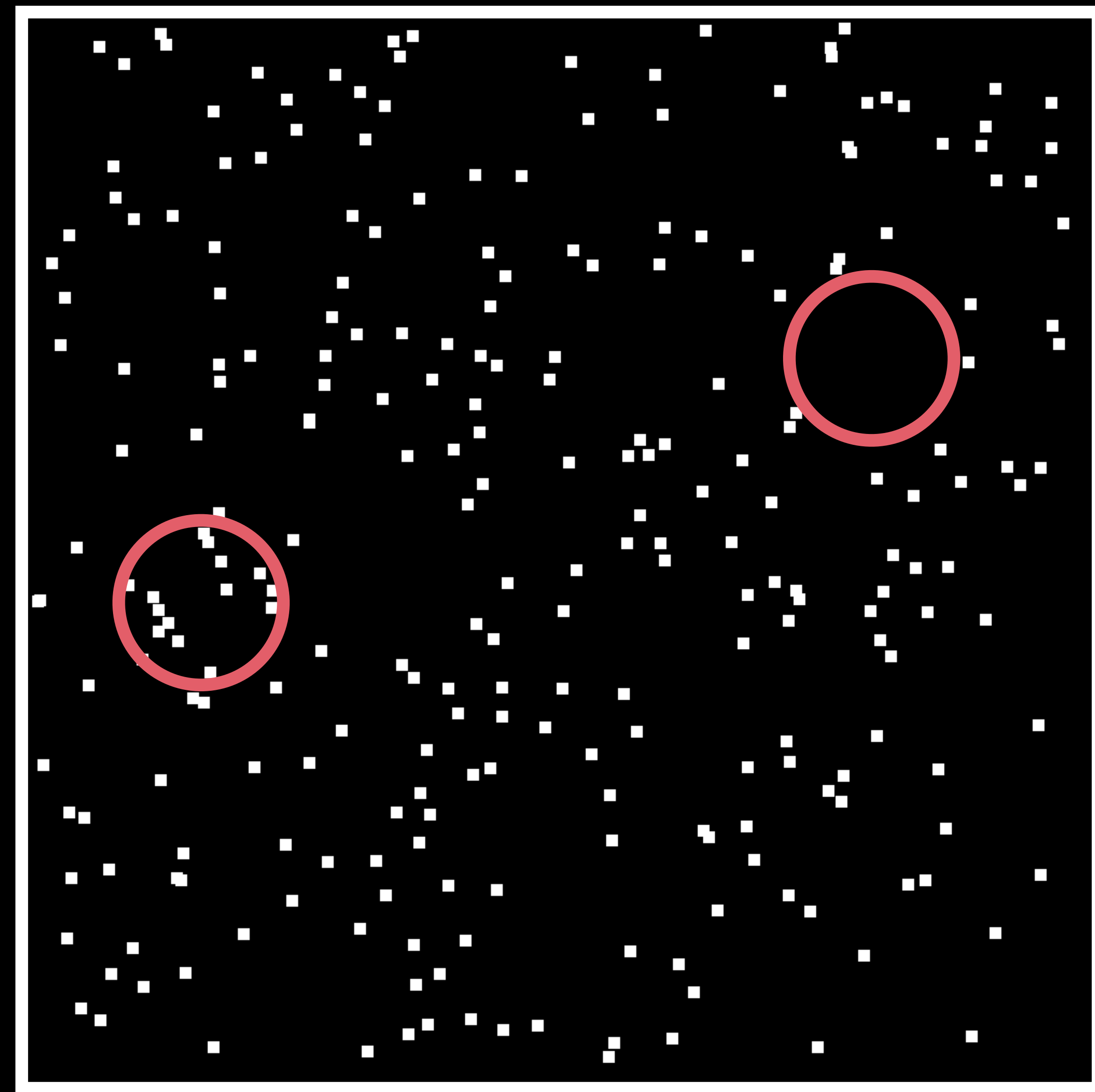


Random

Low Discrepancy Sequences

Distribute points evenly in parameter space

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

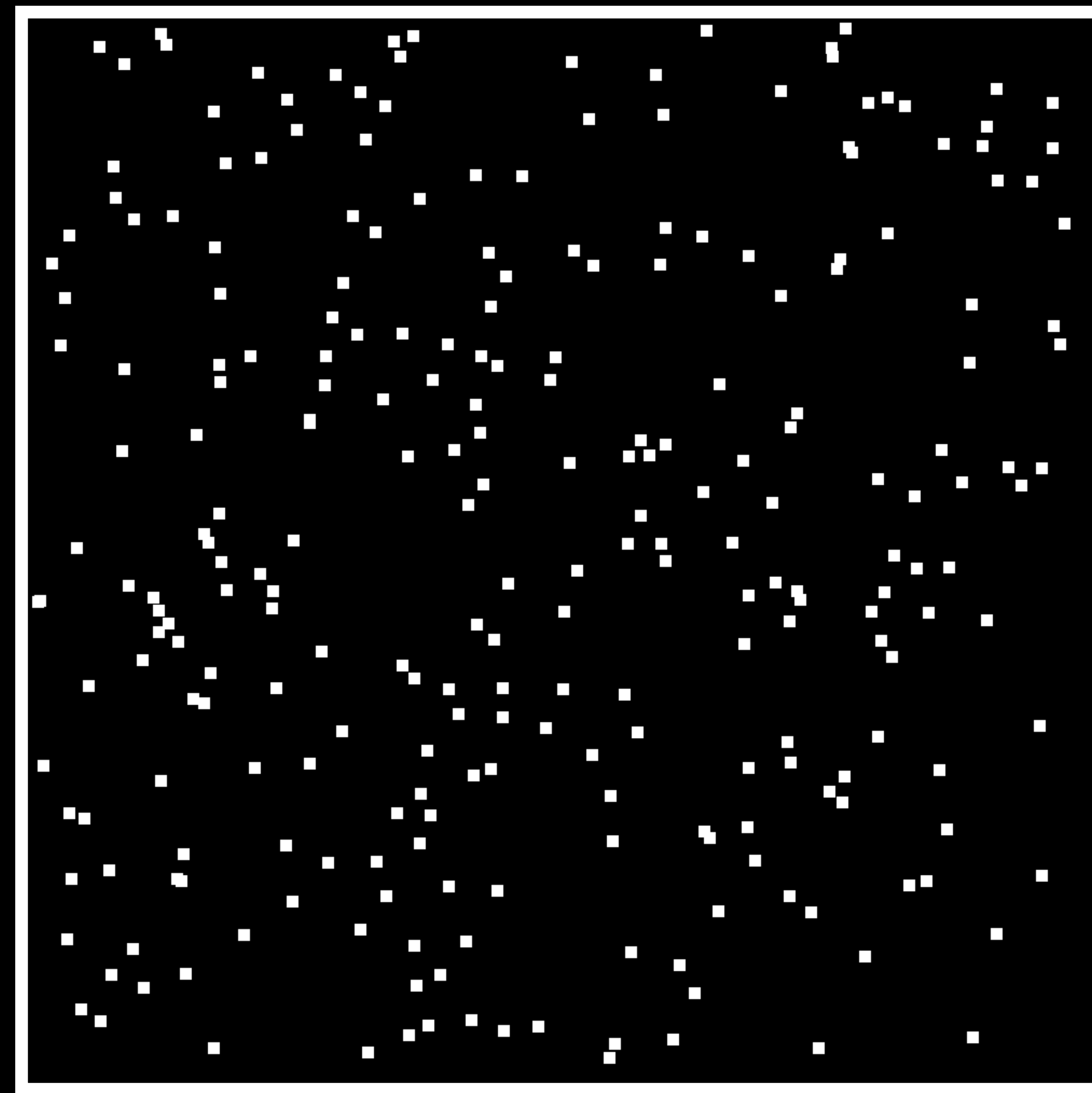


Random

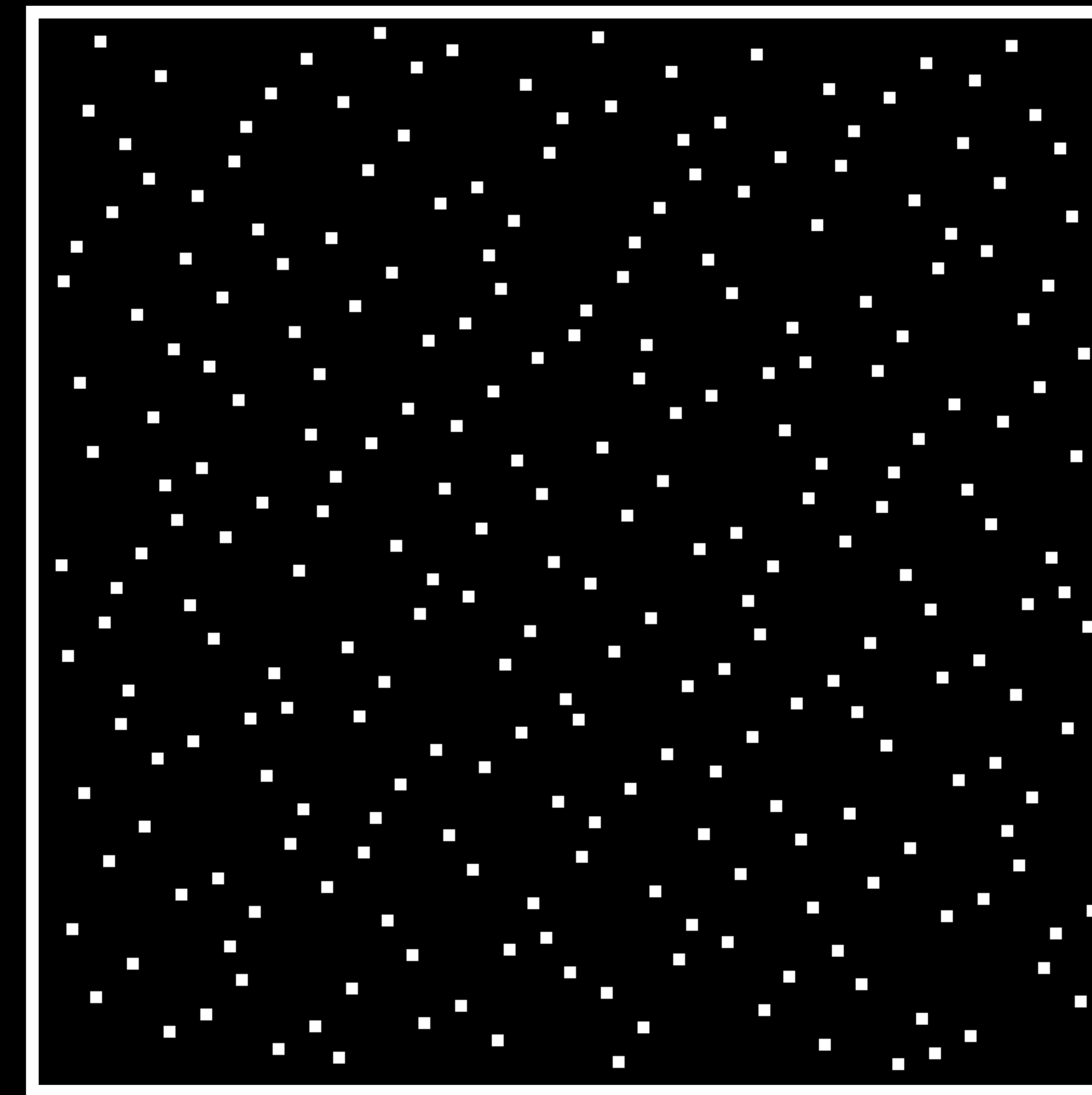
Low Discrepancy Sequences

Distribute points evenly in parameter space

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



Random

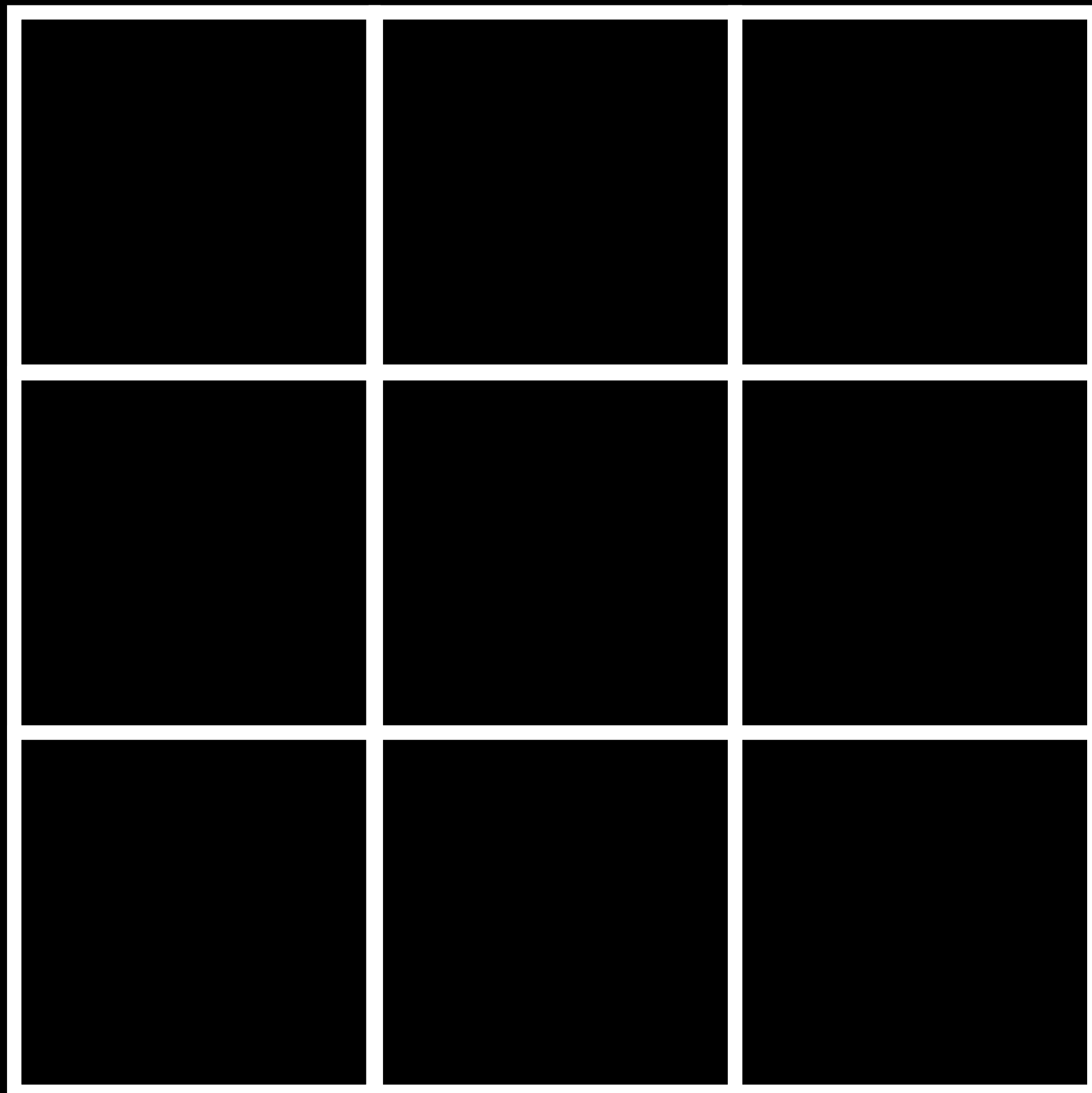


Halton (2, 3)

Pixel Decorrelation

Neighboring pixels sample different directions

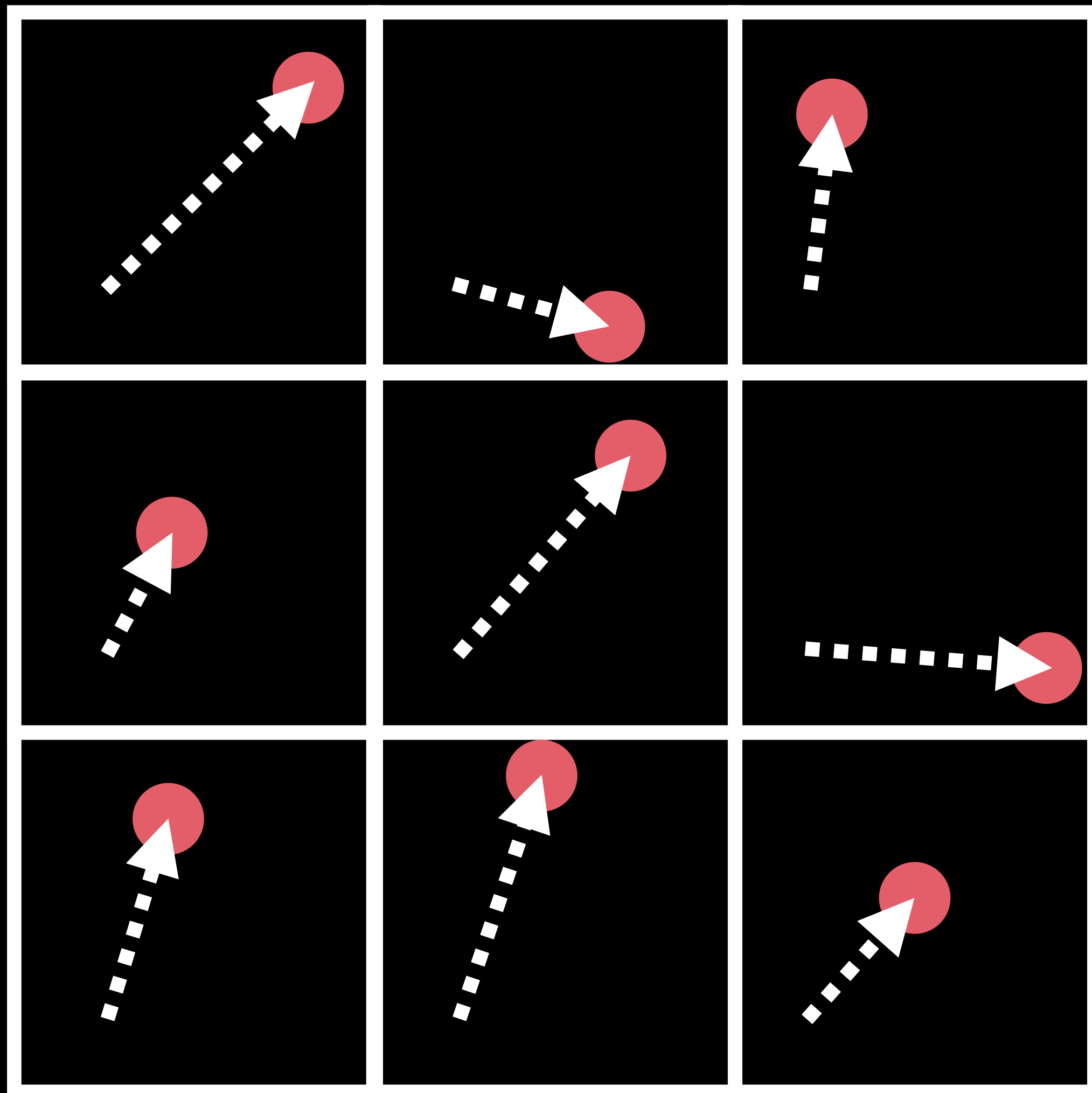
Can use same low discrepancy sample for all pixels



Pixel Decorrelation

Neighboring pixels sample different directions

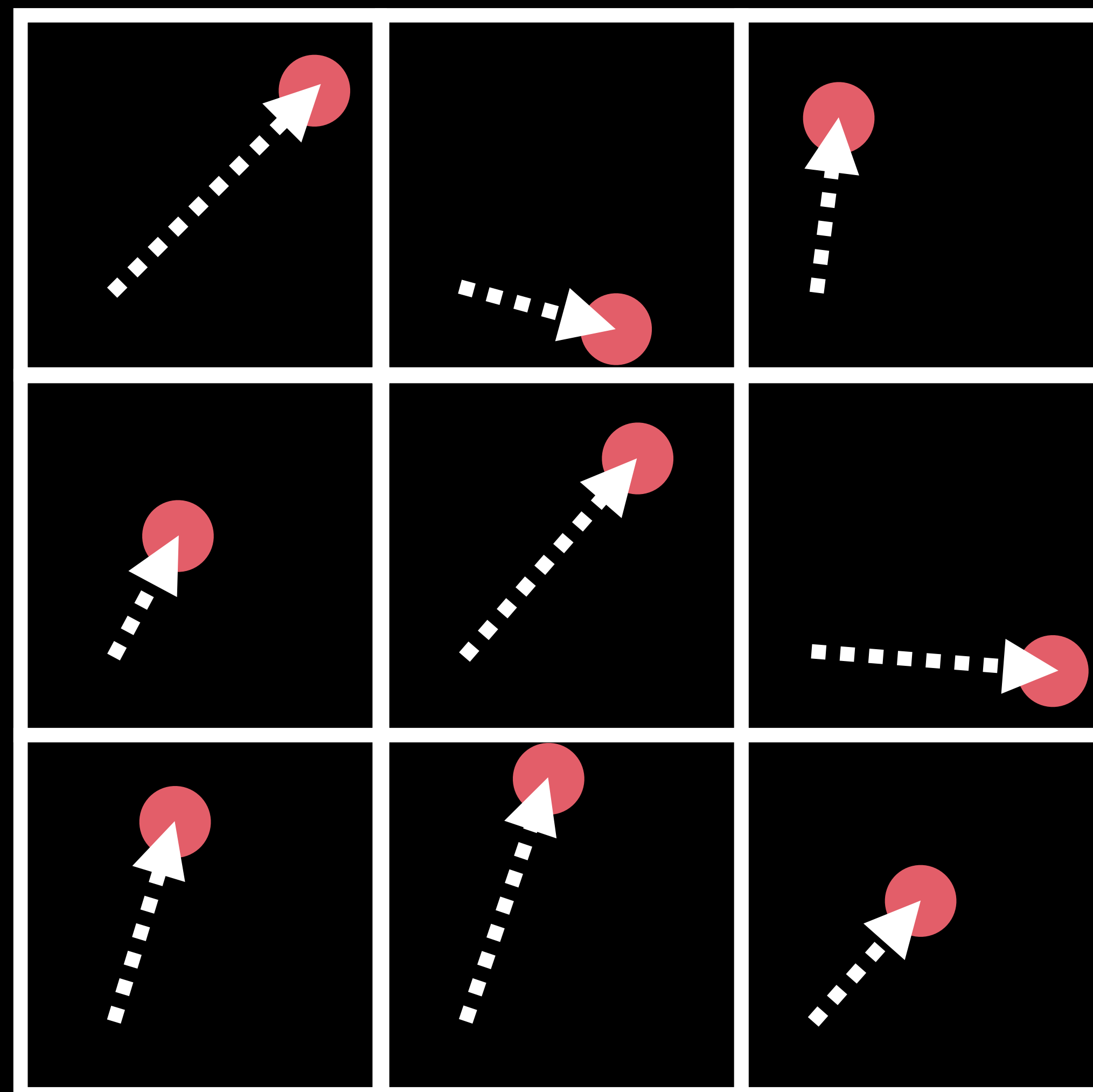
Can use same low discrepancy sample for all pixels



Pixel Decorrelation

Neighboring pixels sample different directions

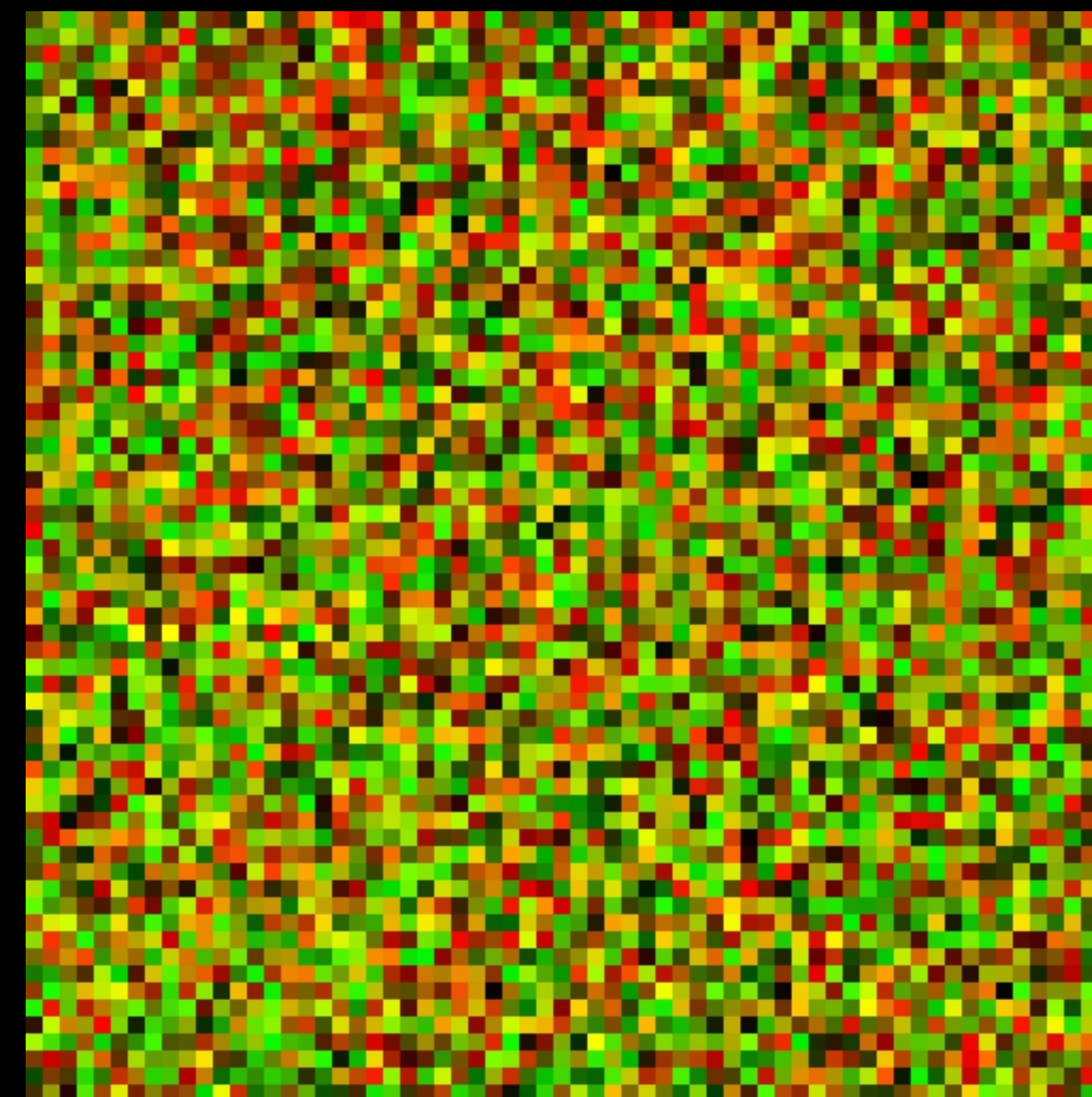
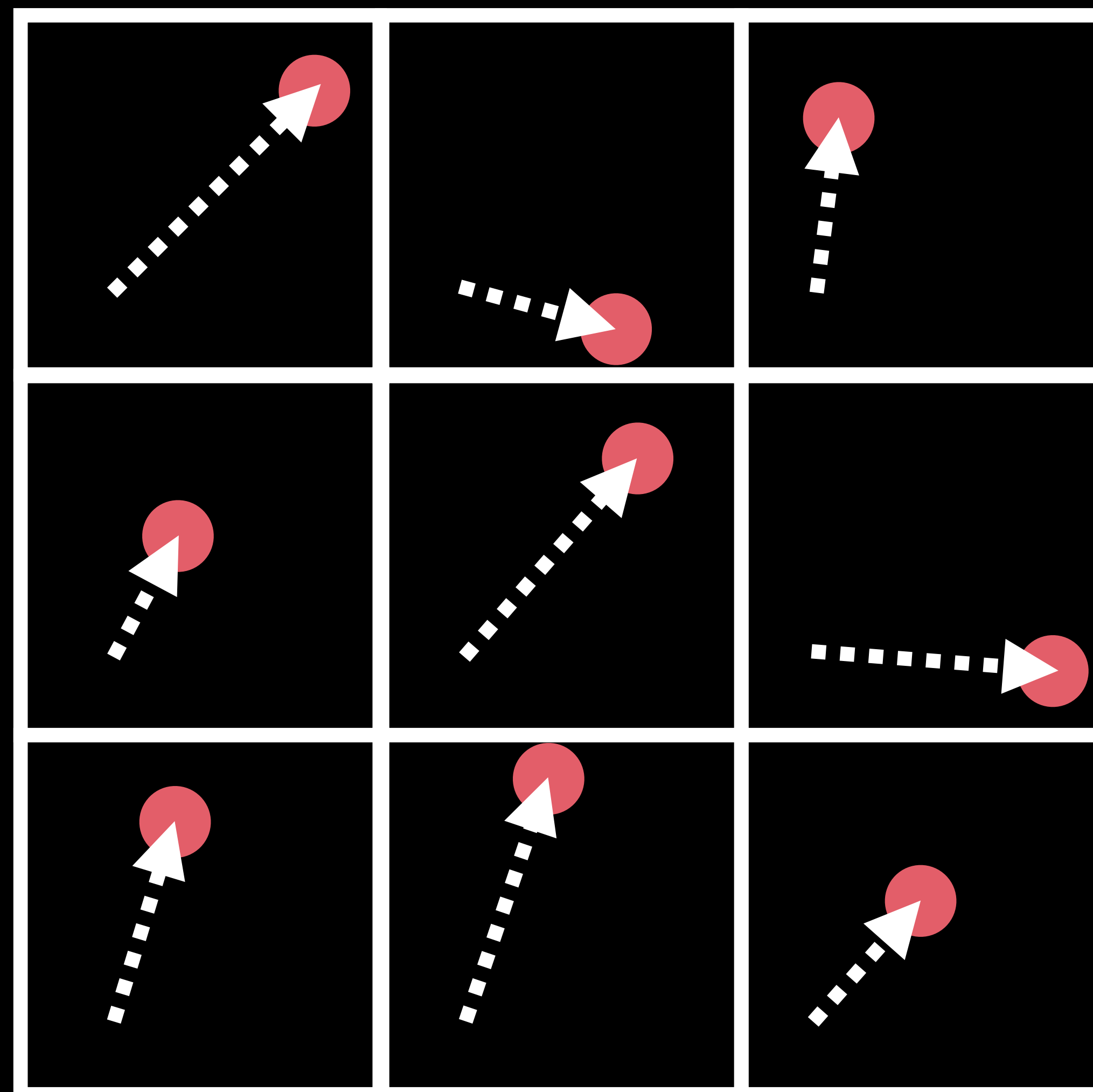
Can use same low discrepancy sample for all pixels



Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels

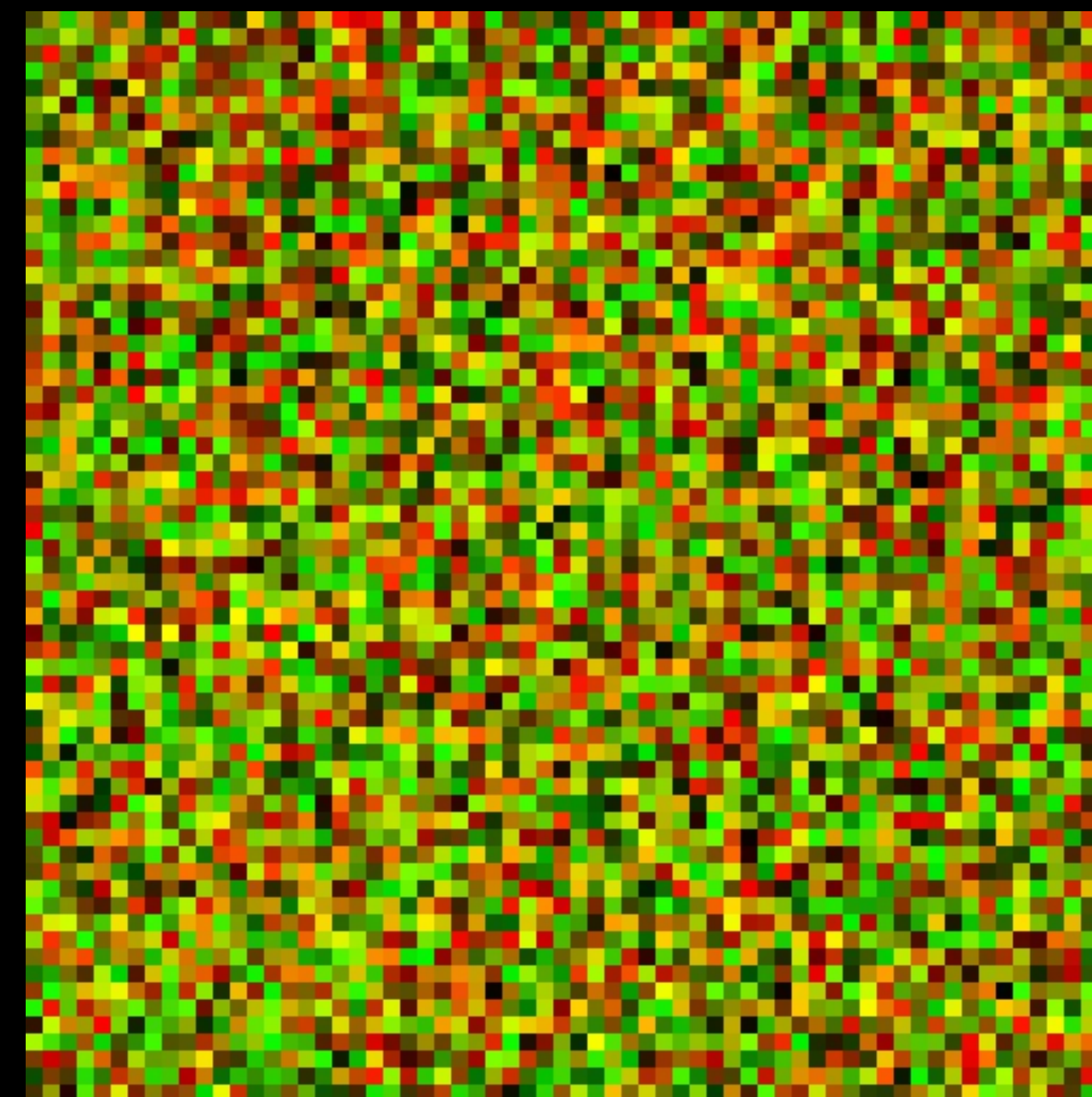
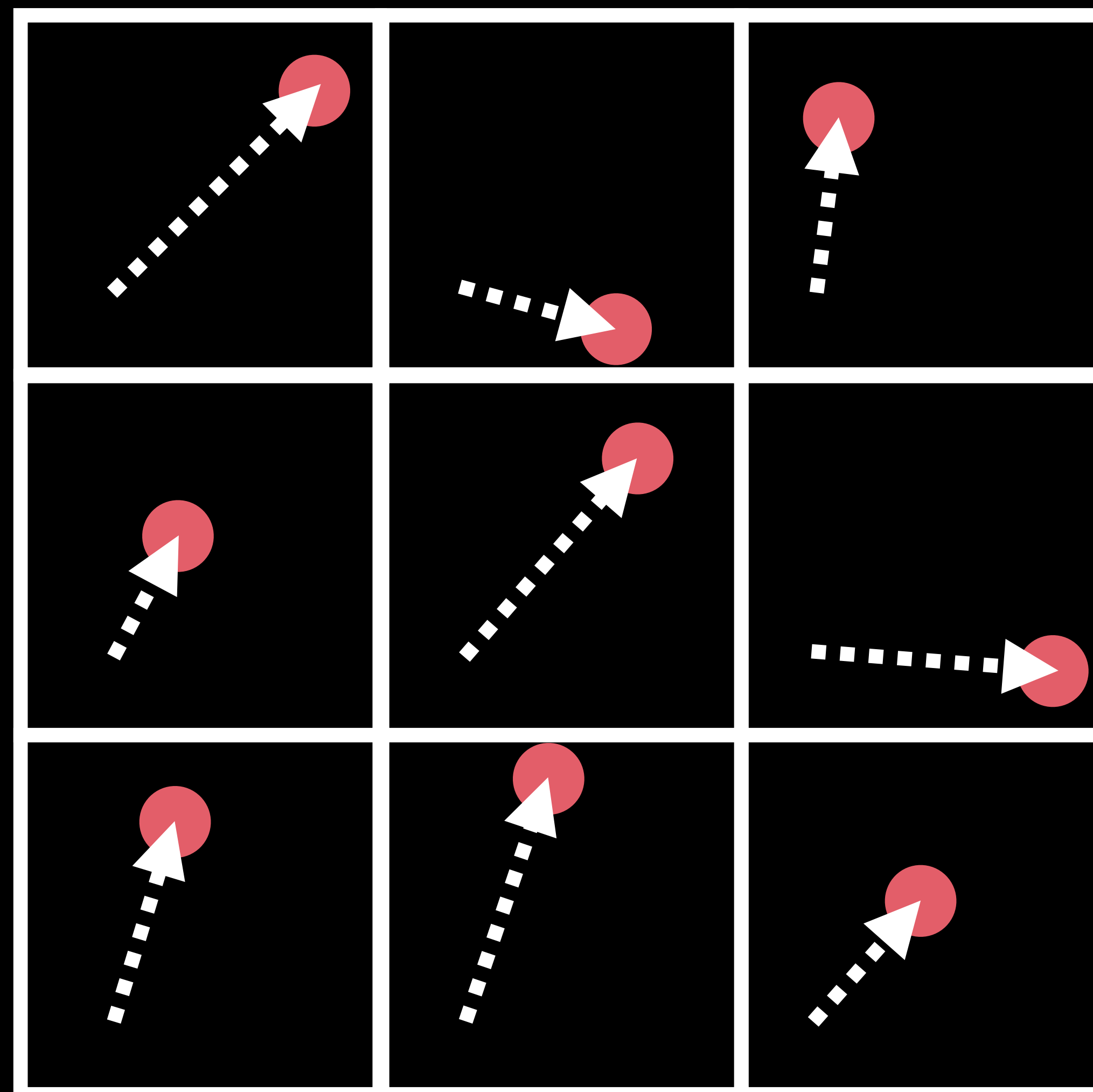


White Noise

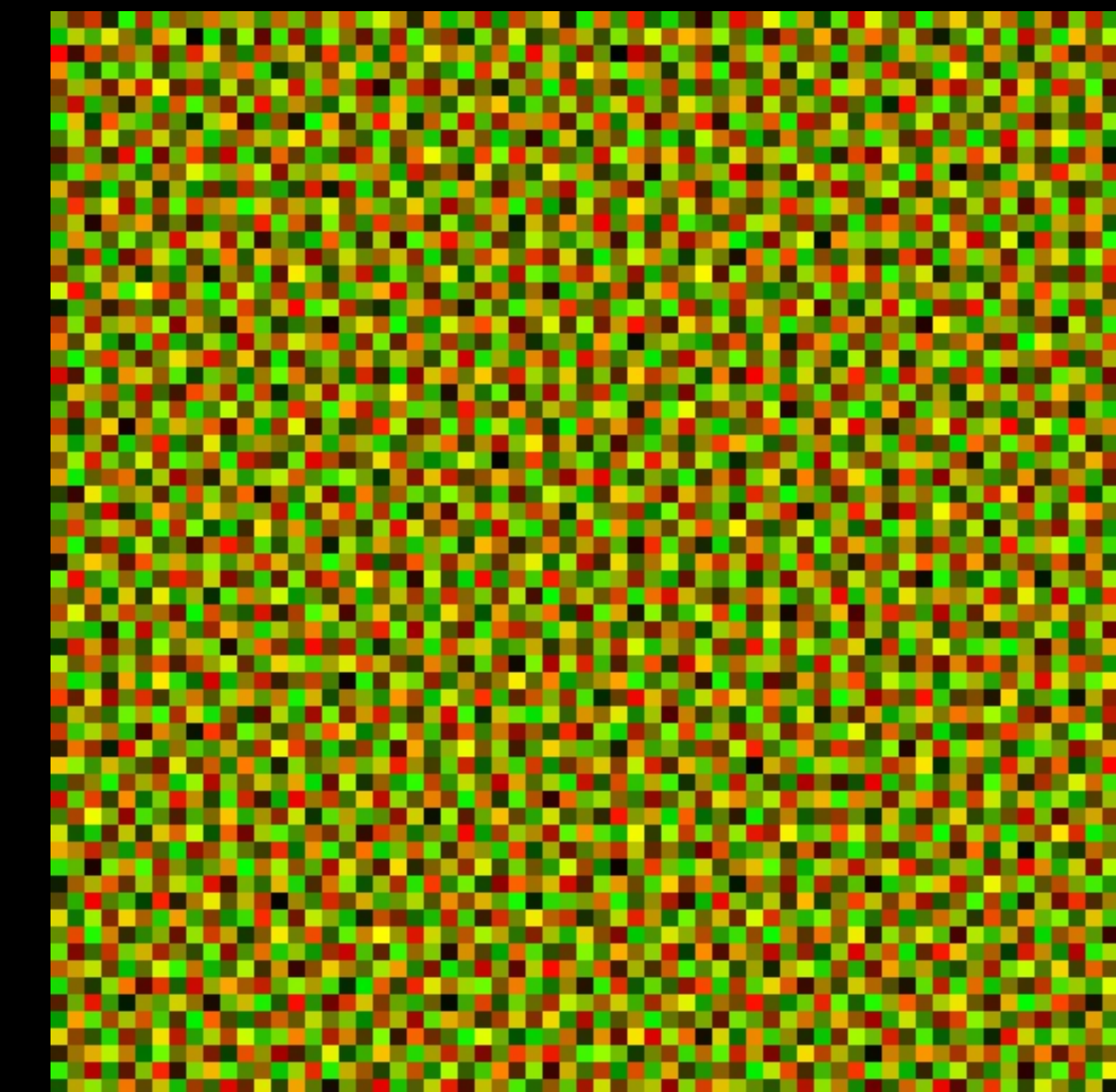
Pixel Decorrelation

Neighboring pixels sample different directions

Can use same low discrepancy sample for all pixels

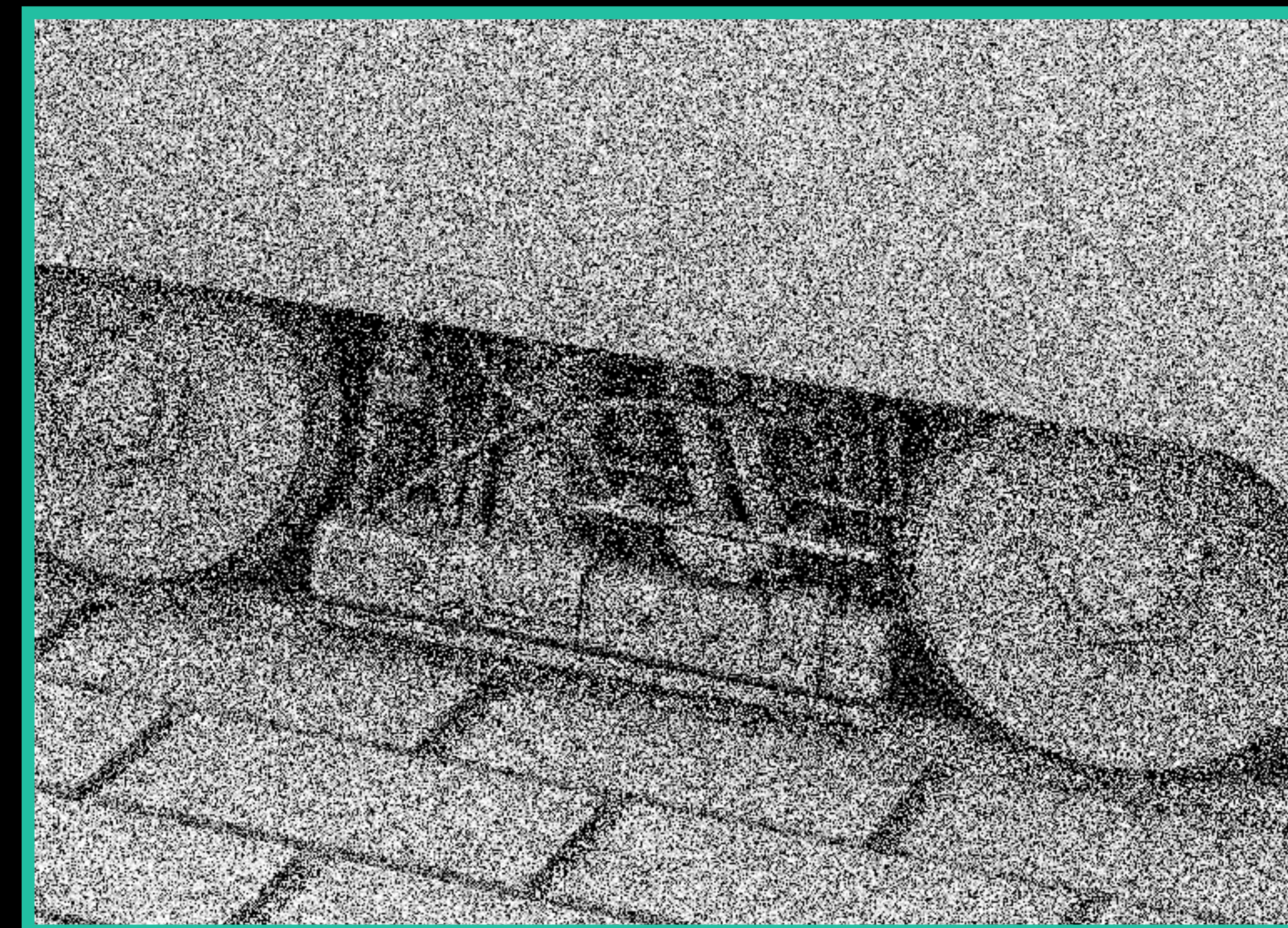
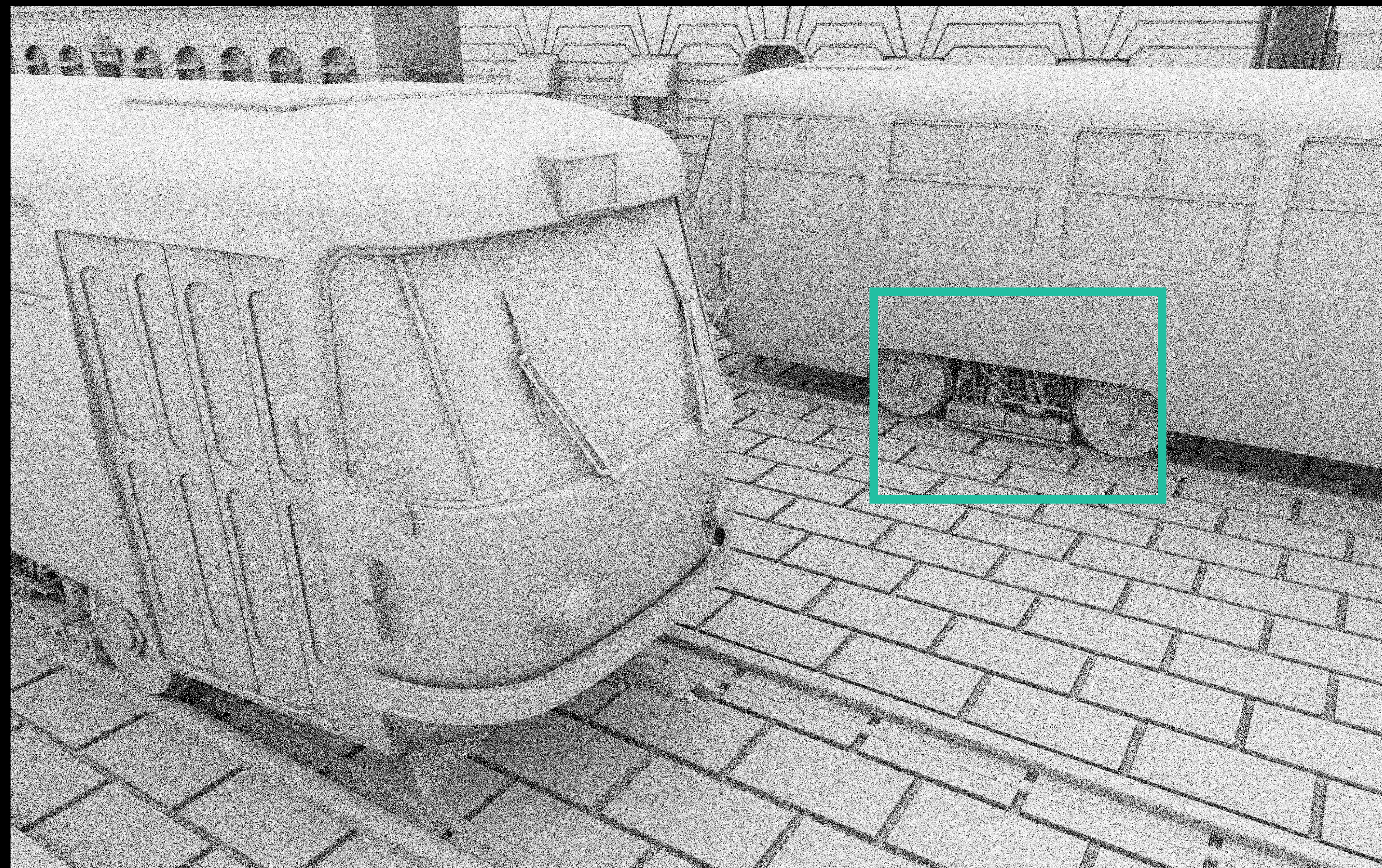


White Noise

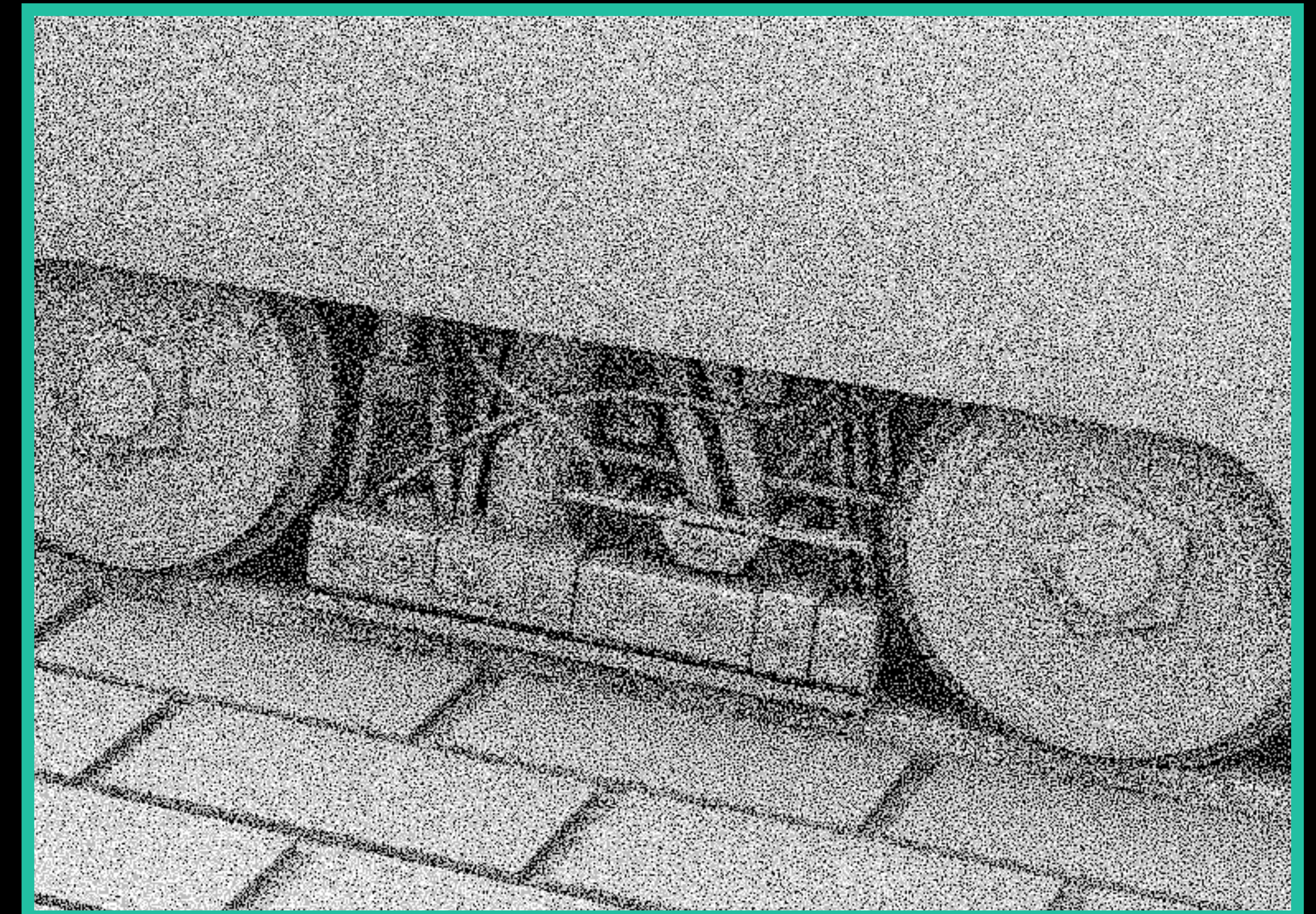
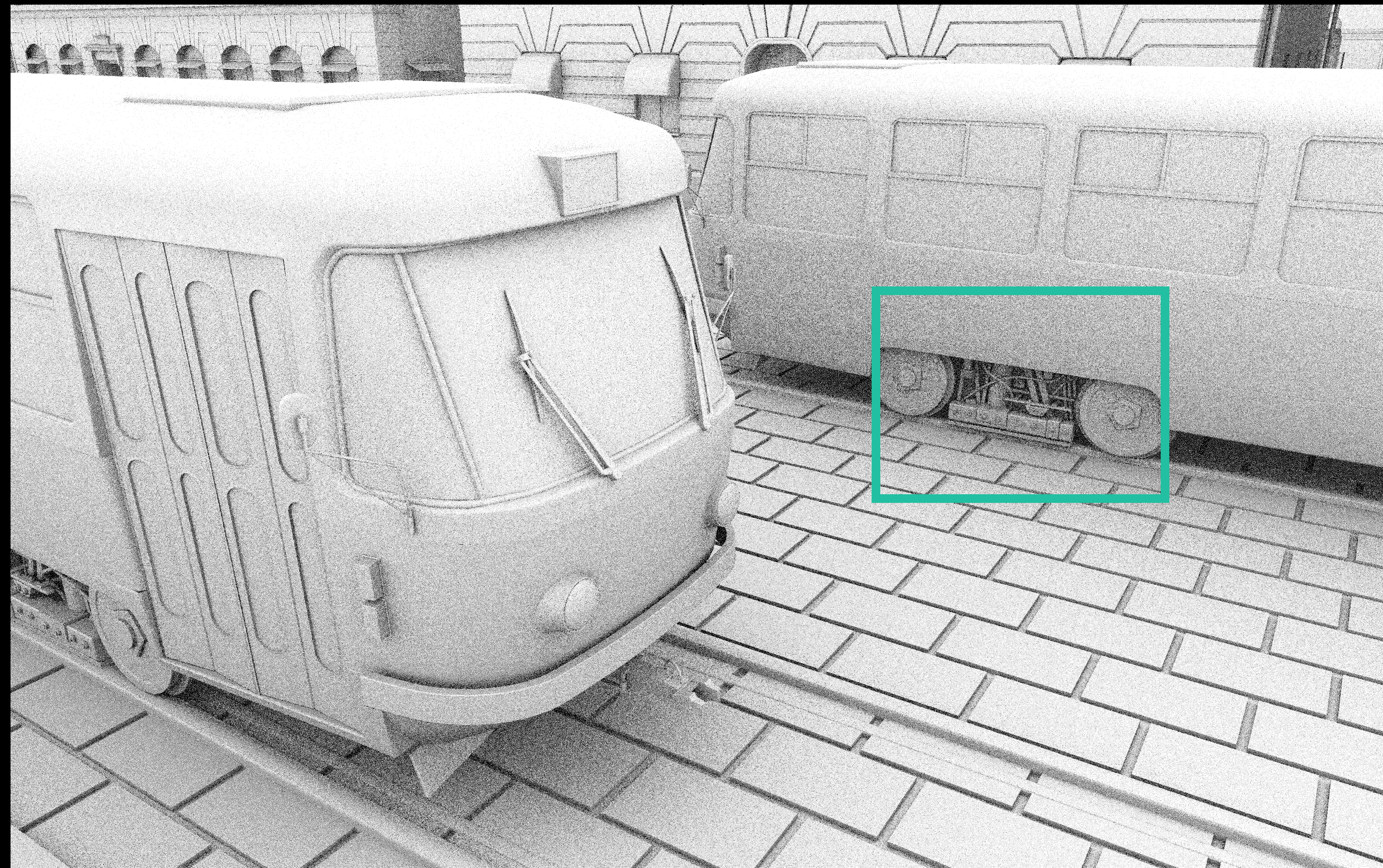


Blue Noise

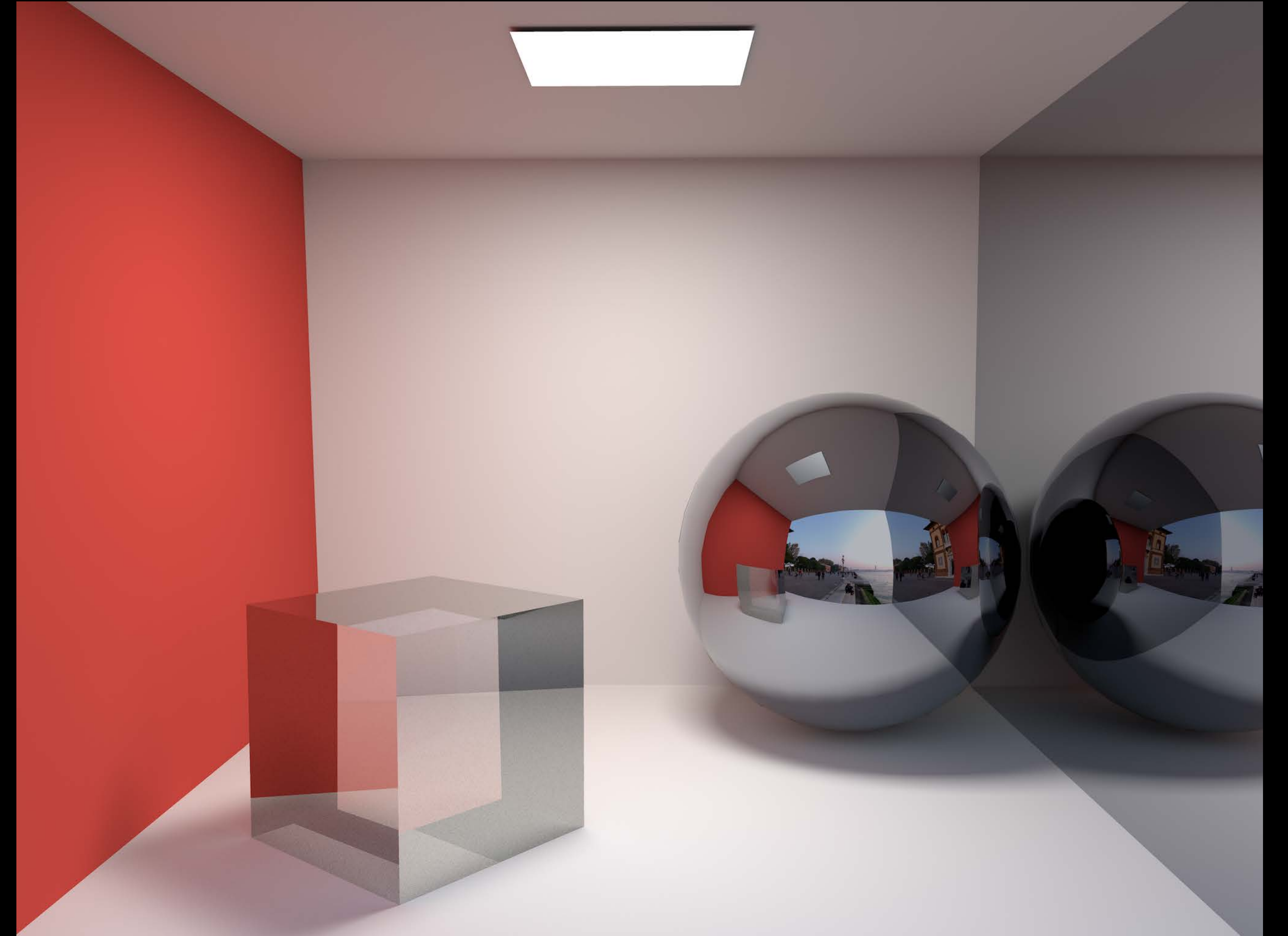
Ray Traced Result



Ray Traced Result



Global Illumination



Global Illumination

What is Global Illumination?

Memory

Ray Lifetime

Debugging

Global Illumination

What is Global Illumination?

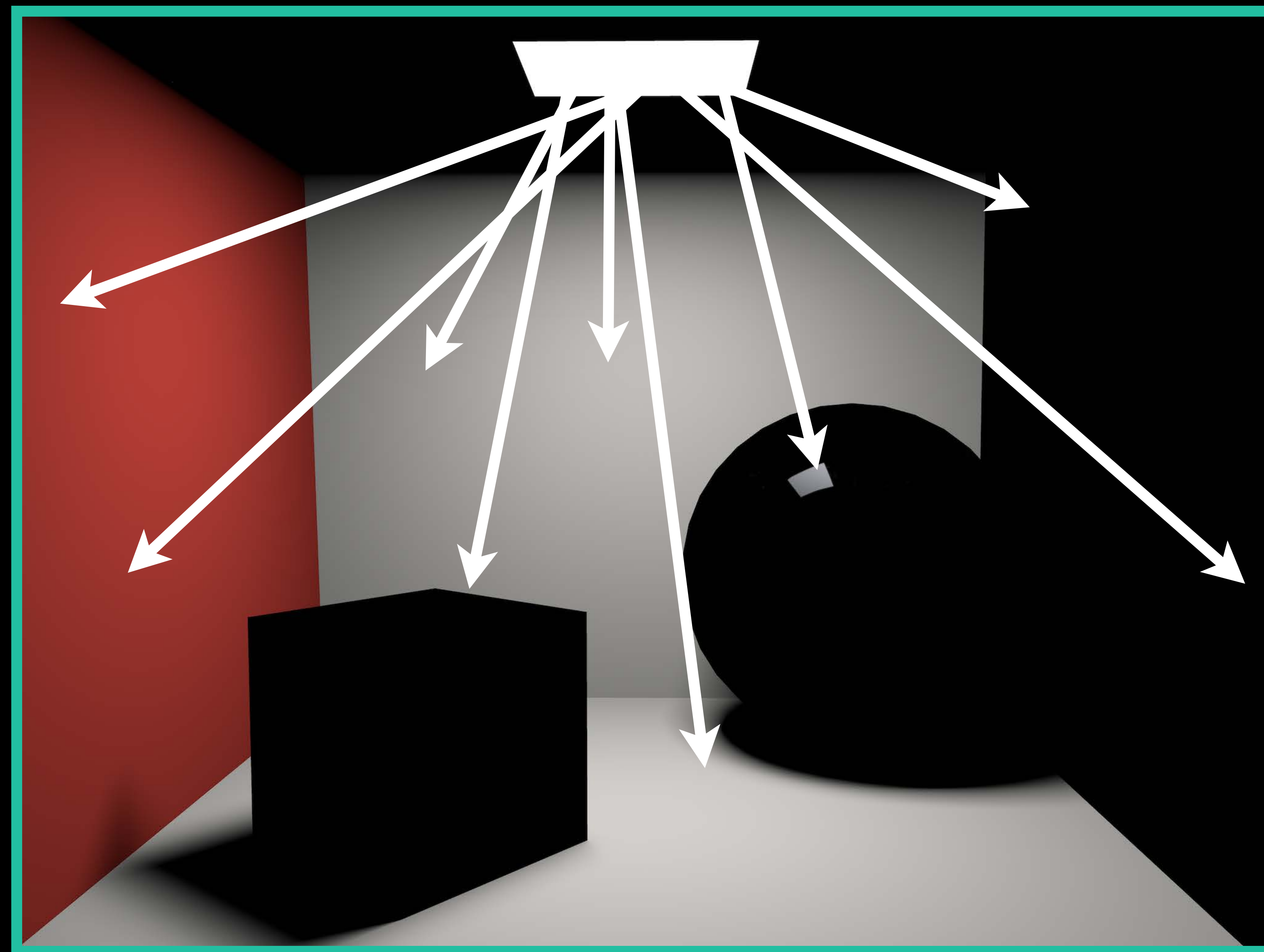
Memory

Ray Lifetime

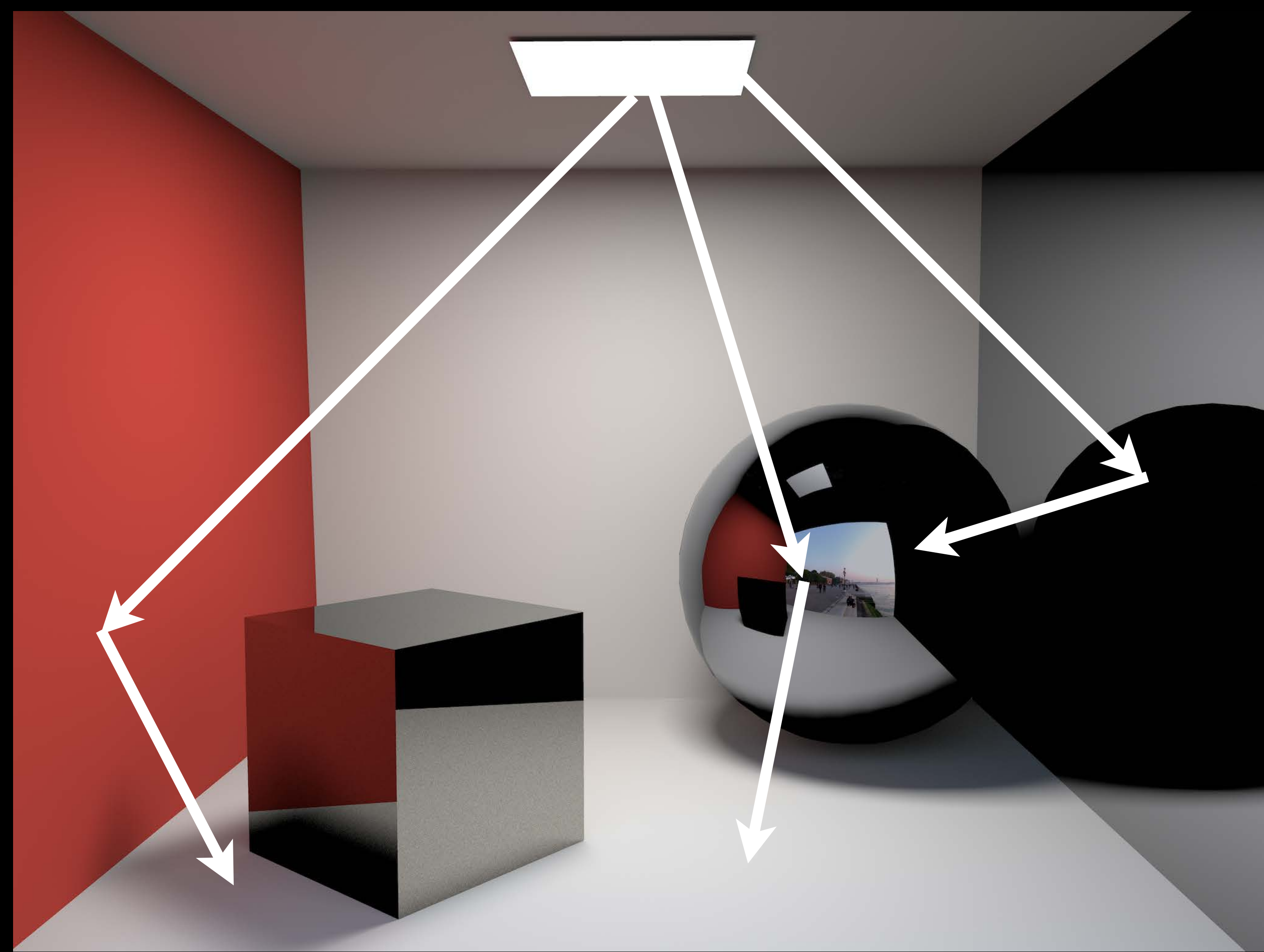
Debugging

Global Illumination

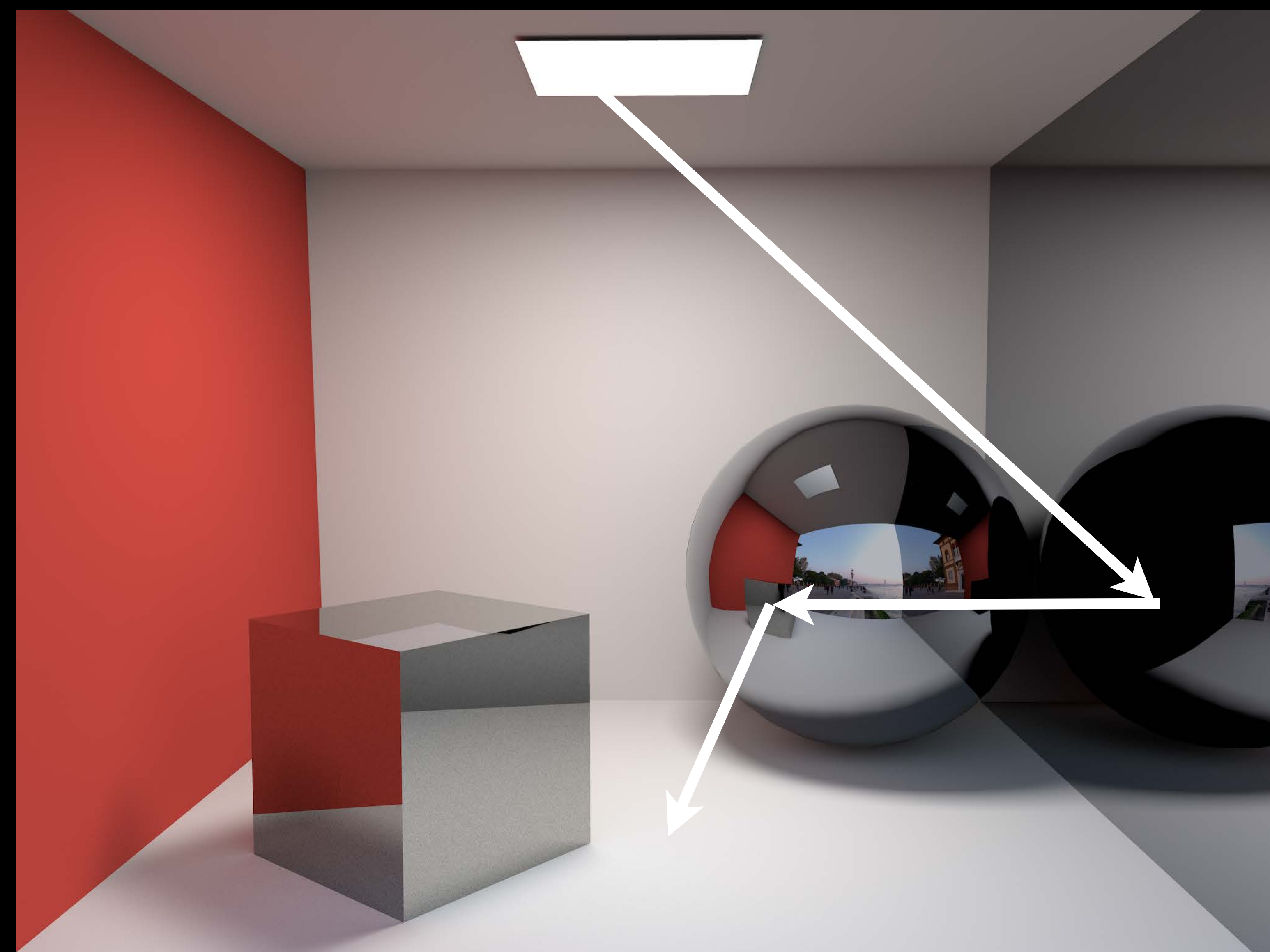
1



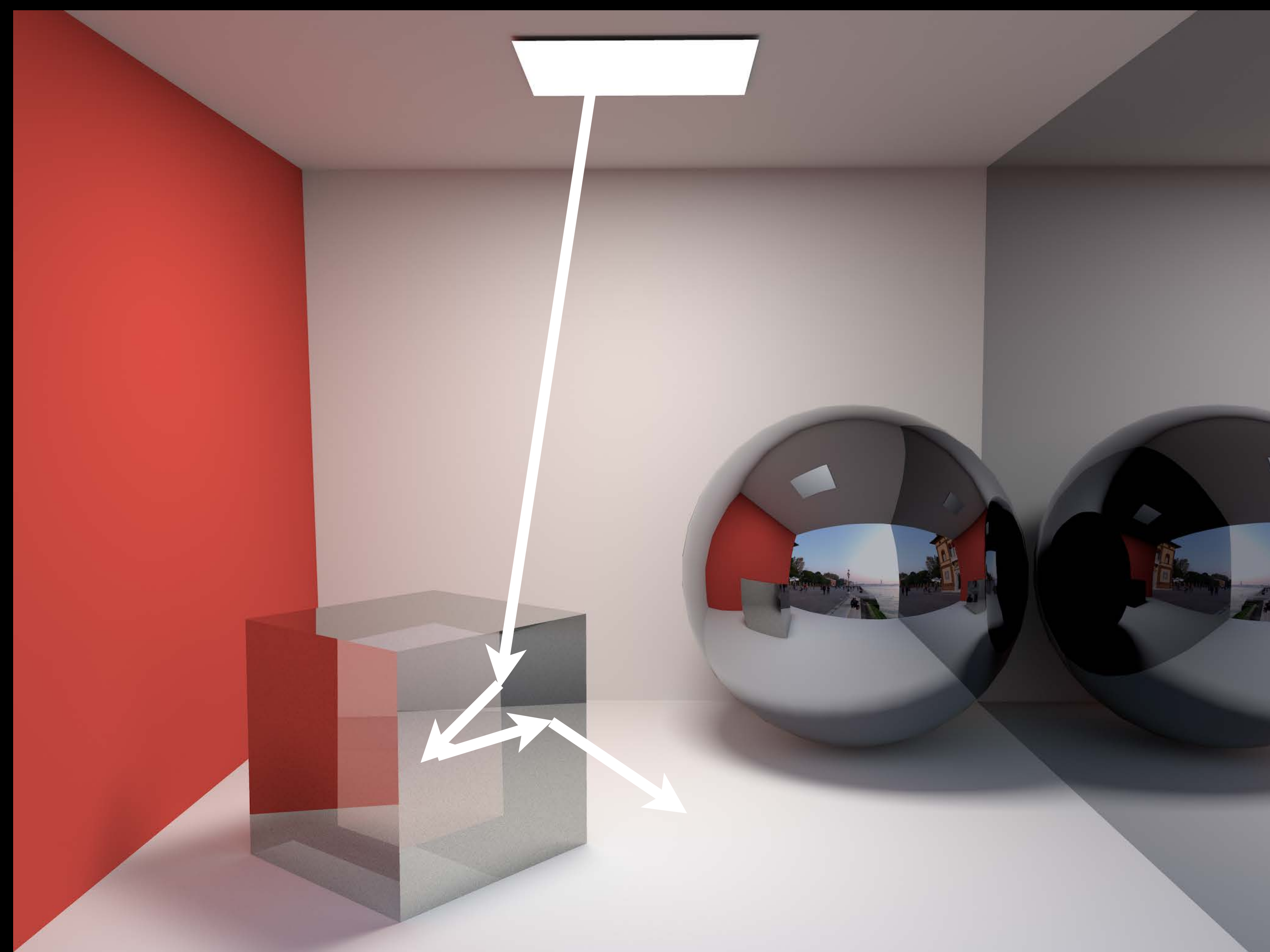
2



3

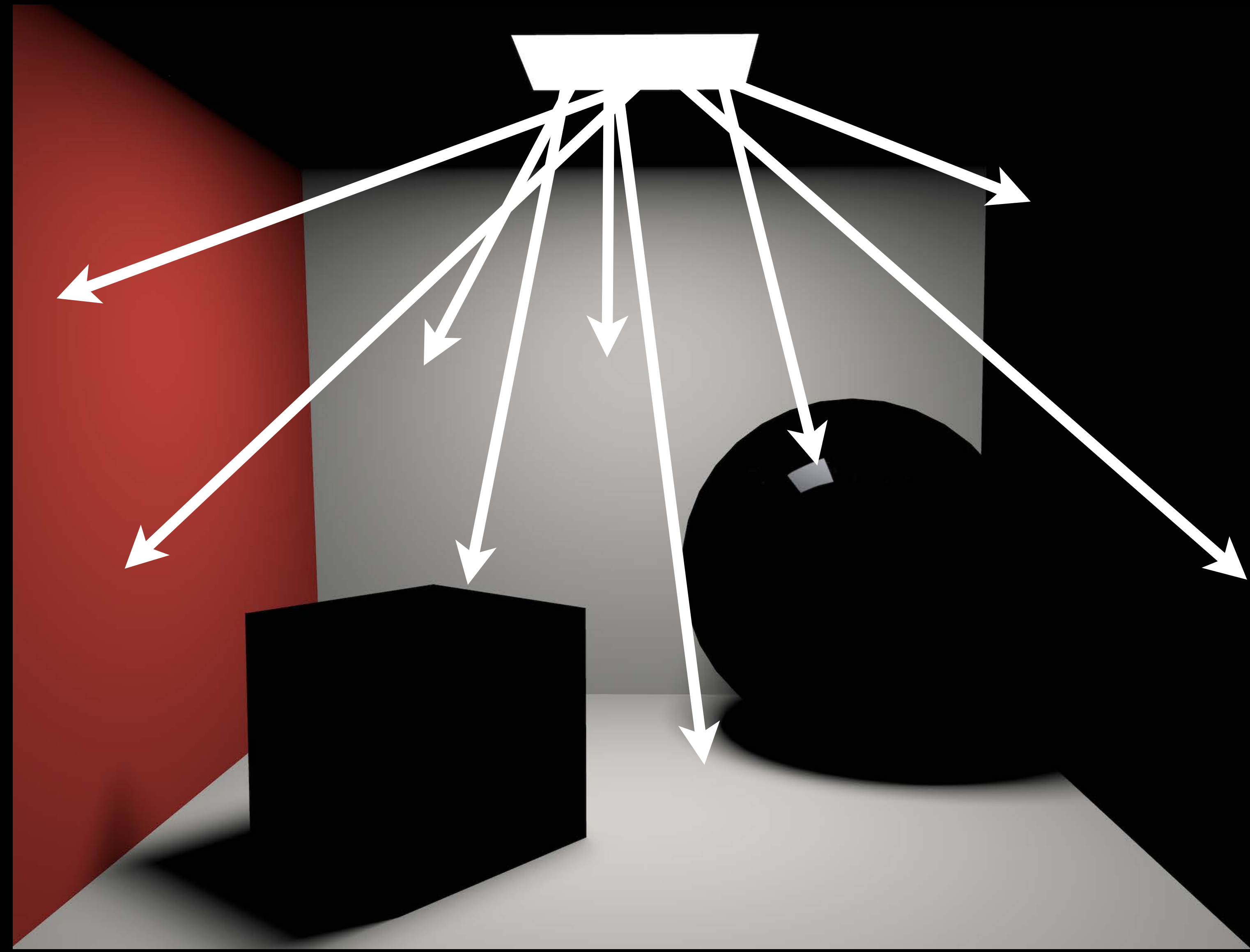


4

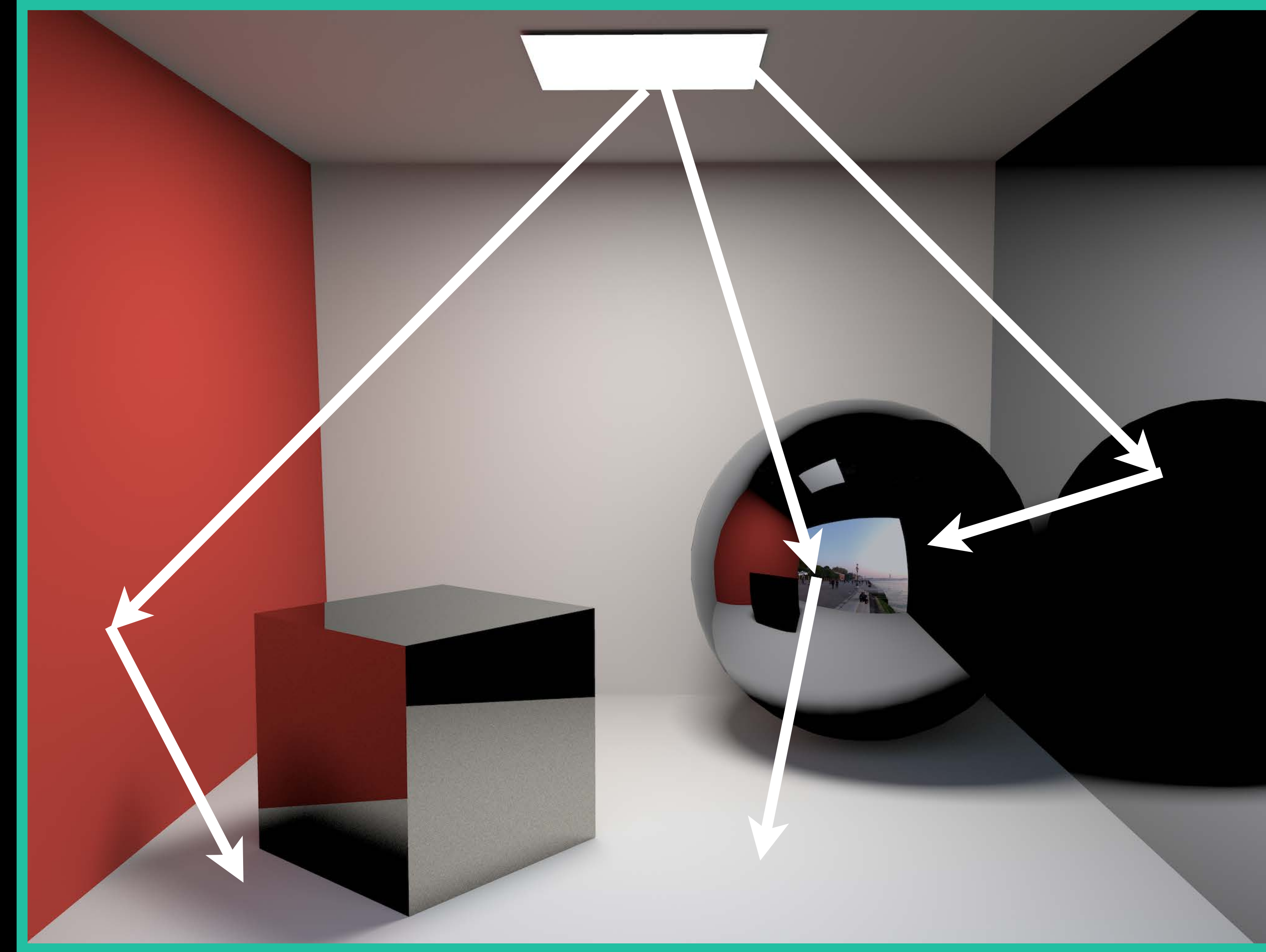


Global Illumination

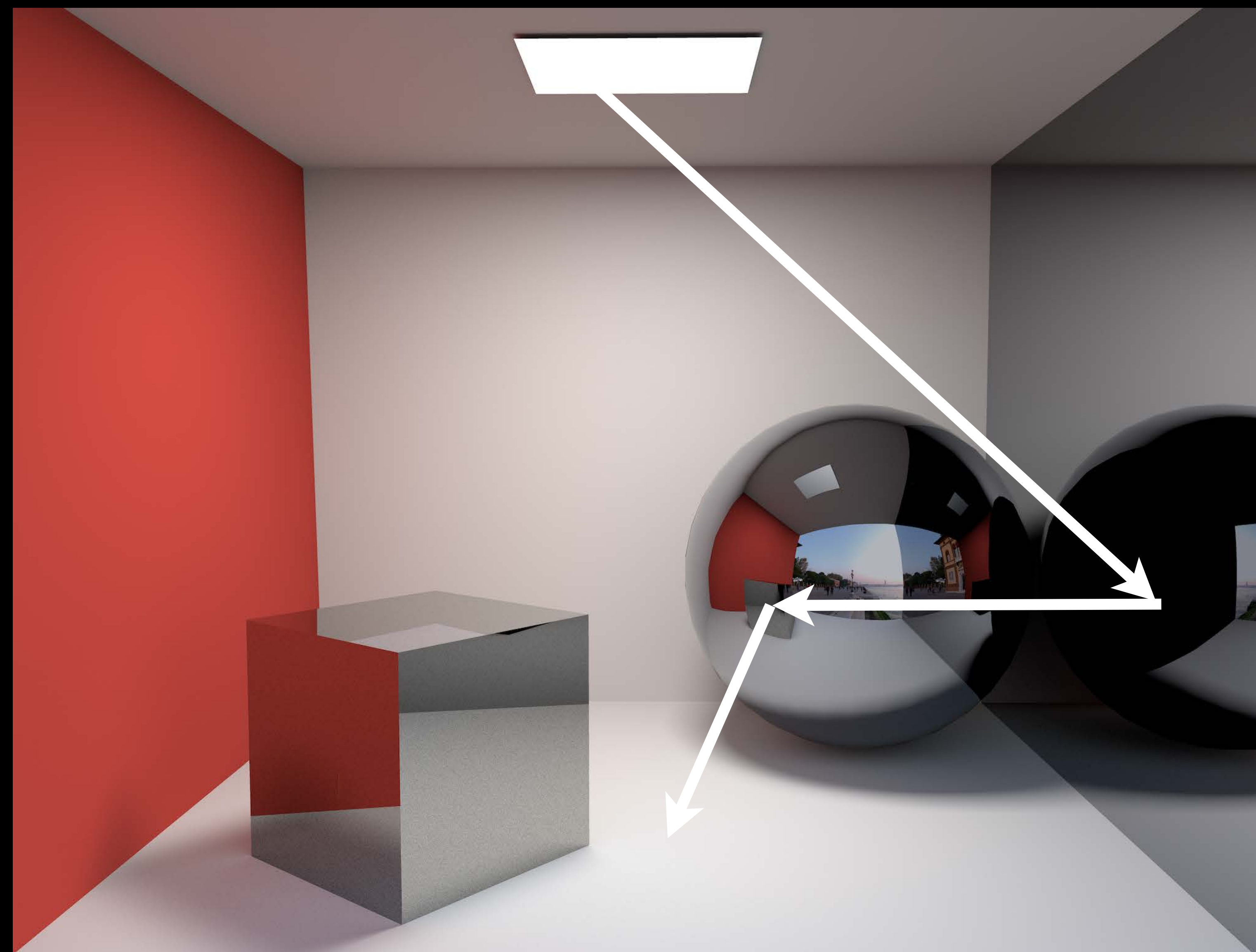
1



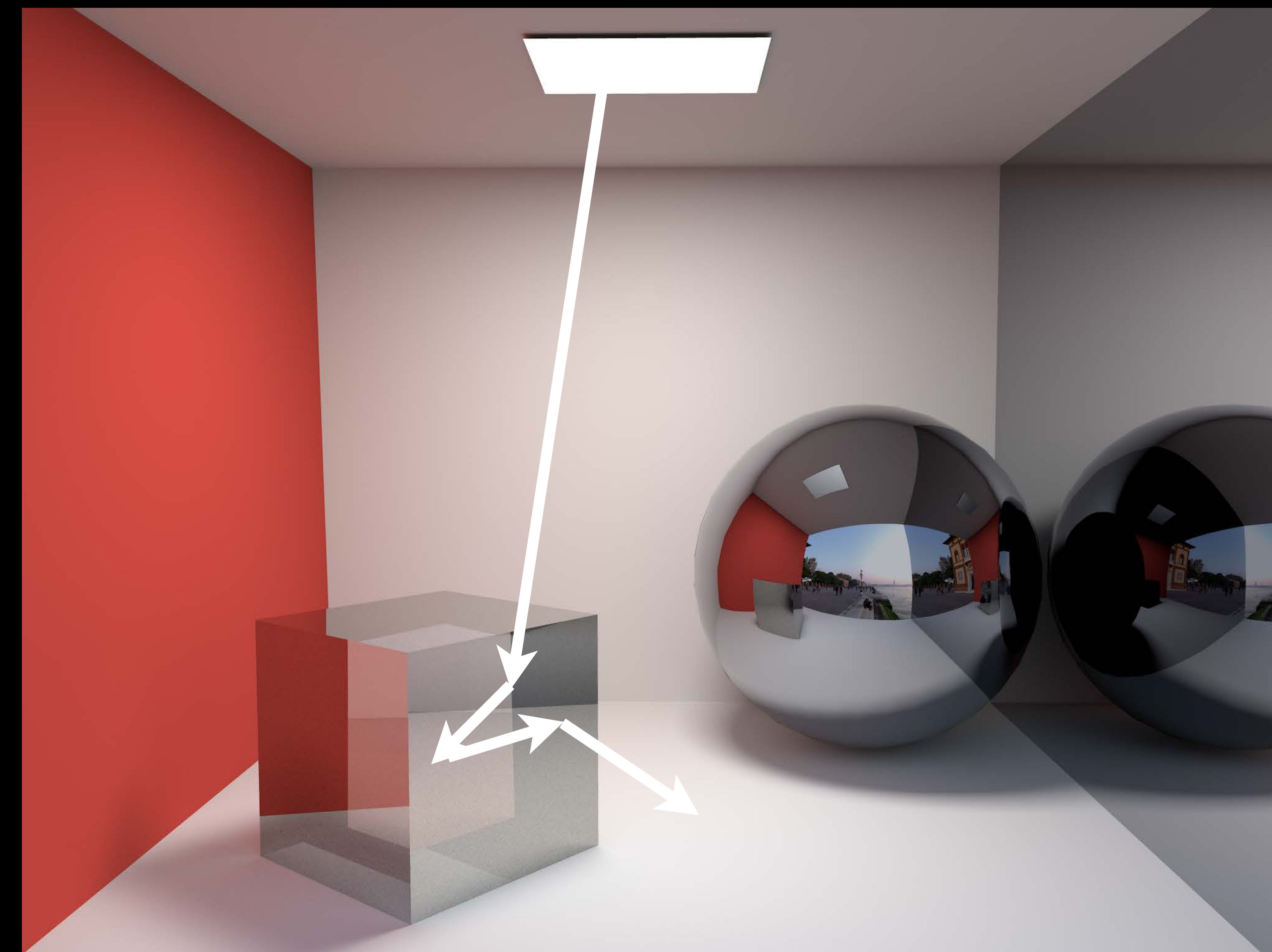
2



3

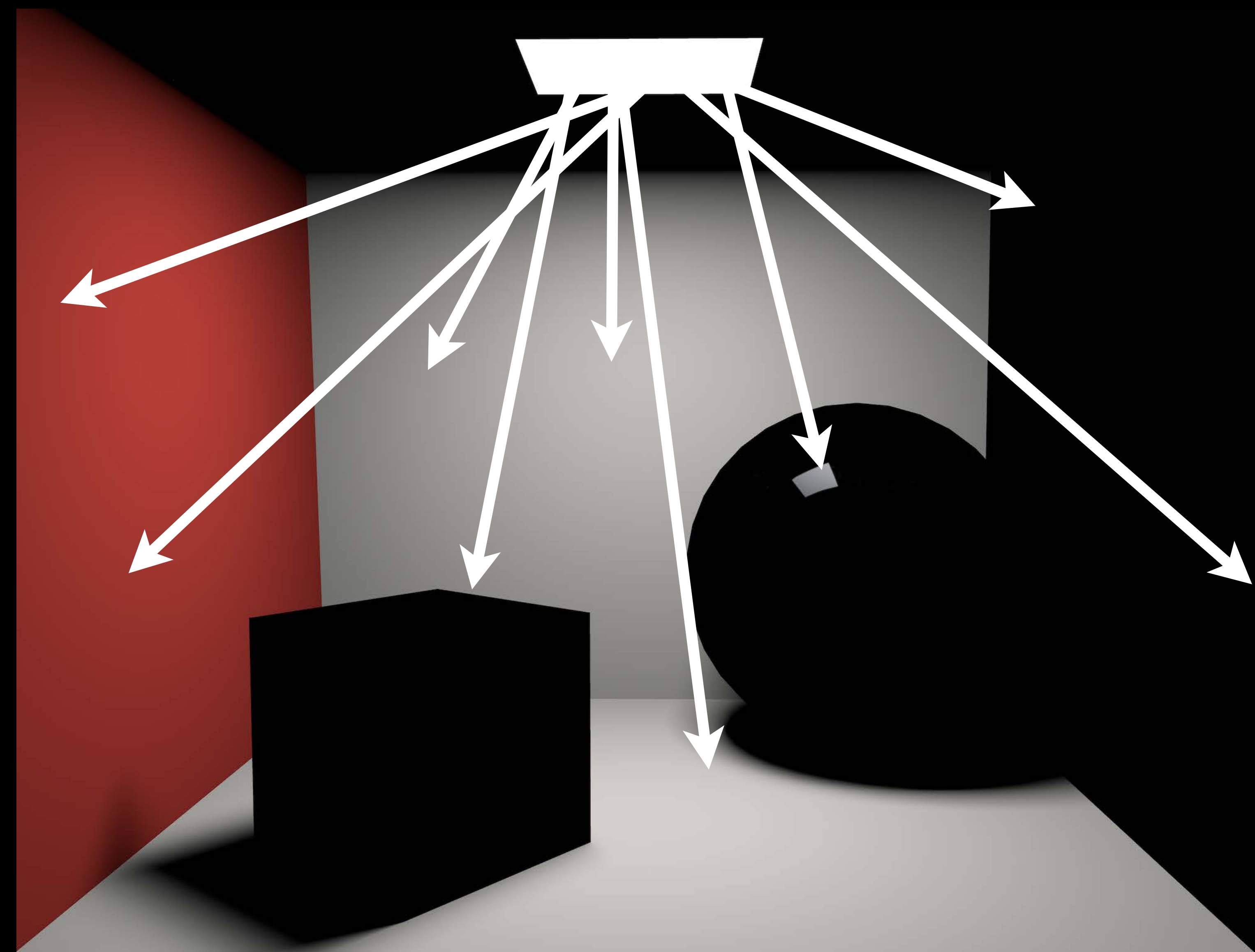


4

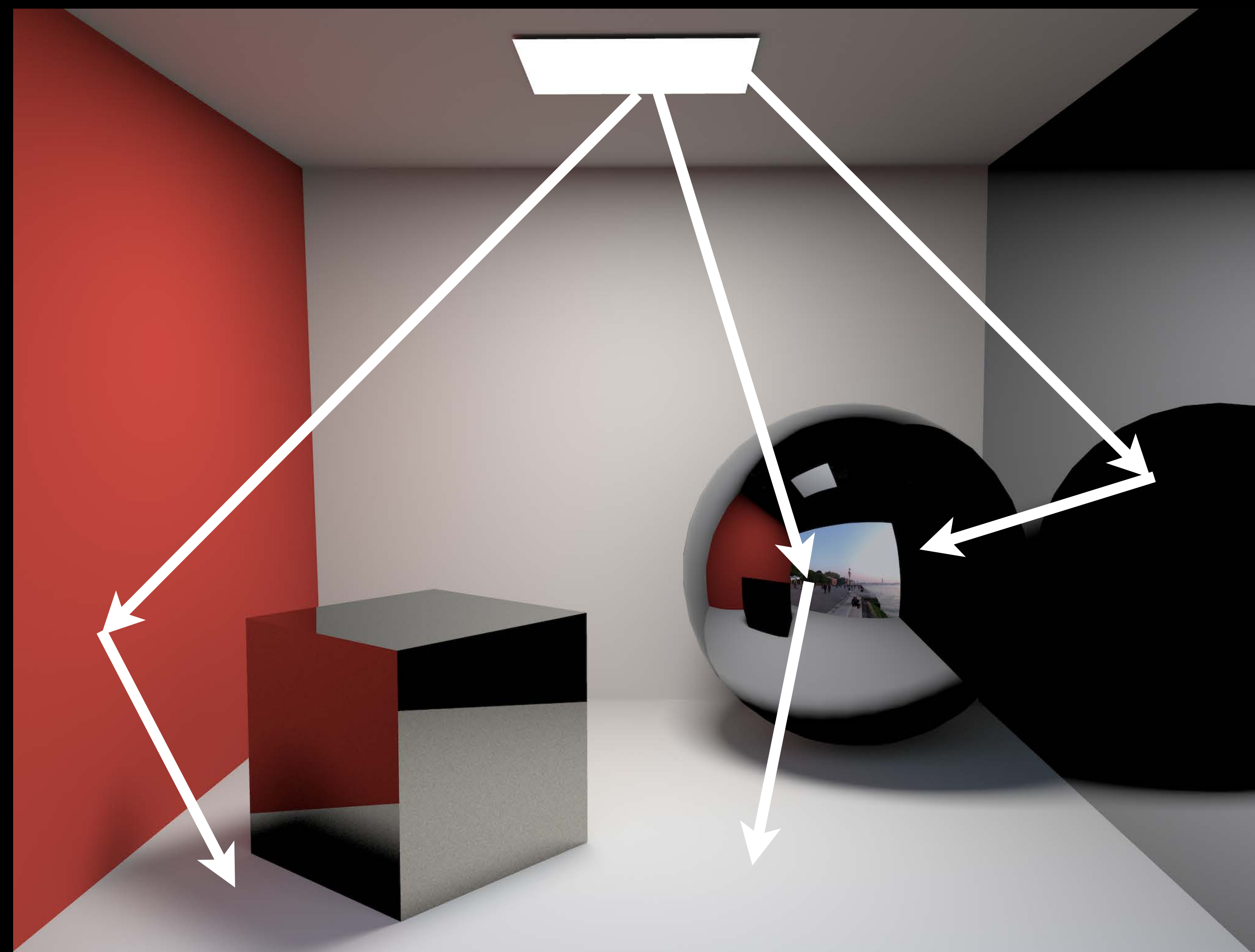


Global Illumination

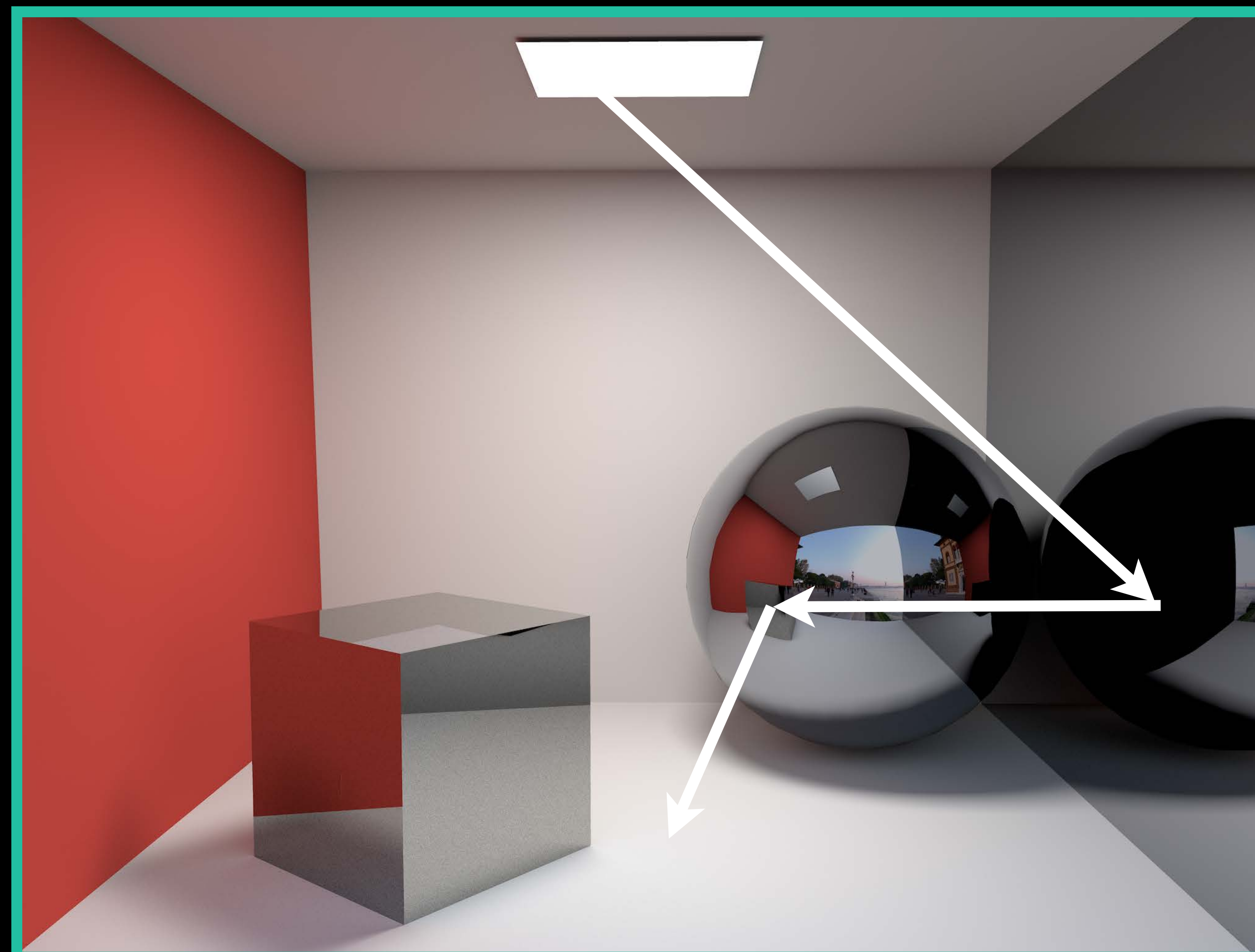
1



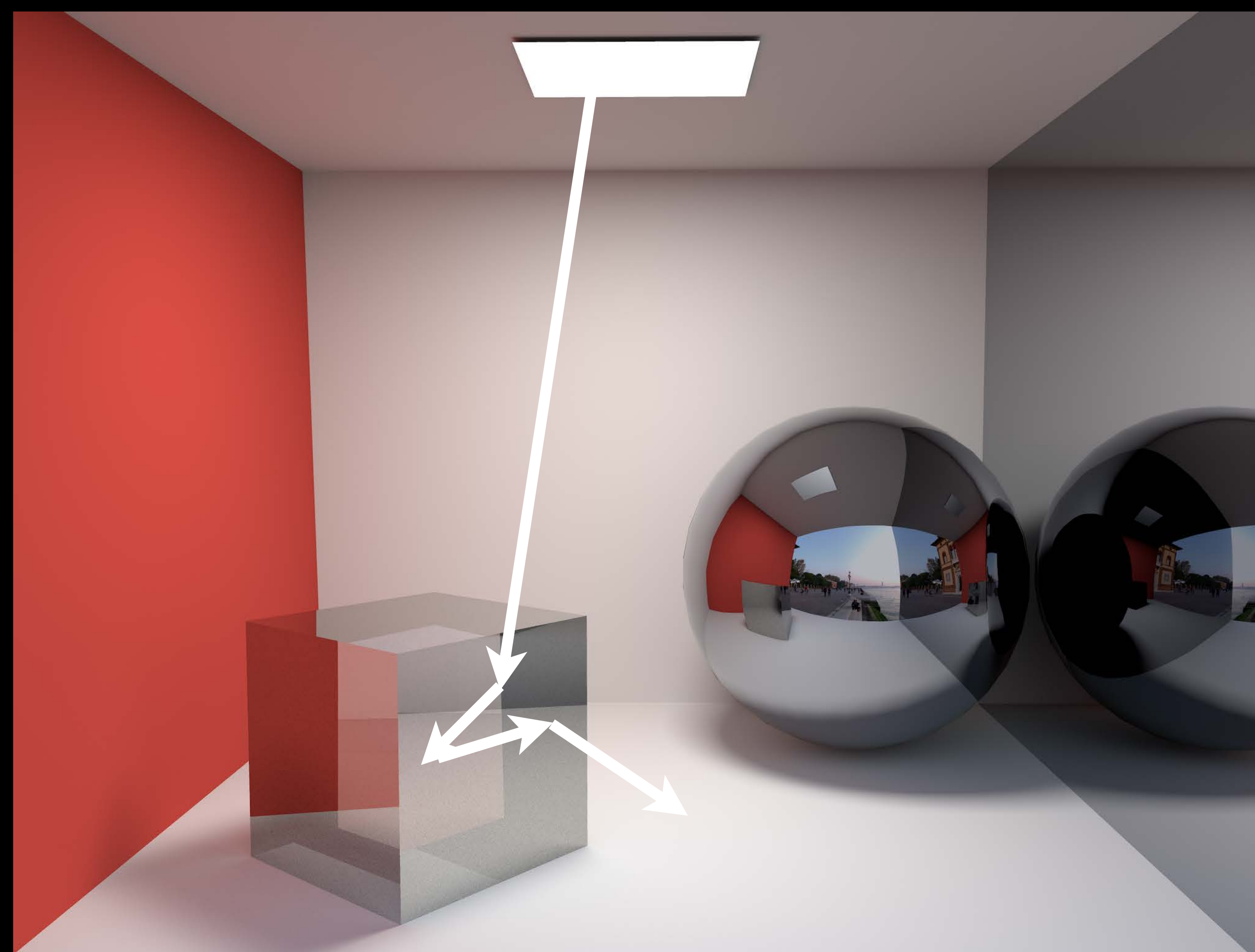
2



3

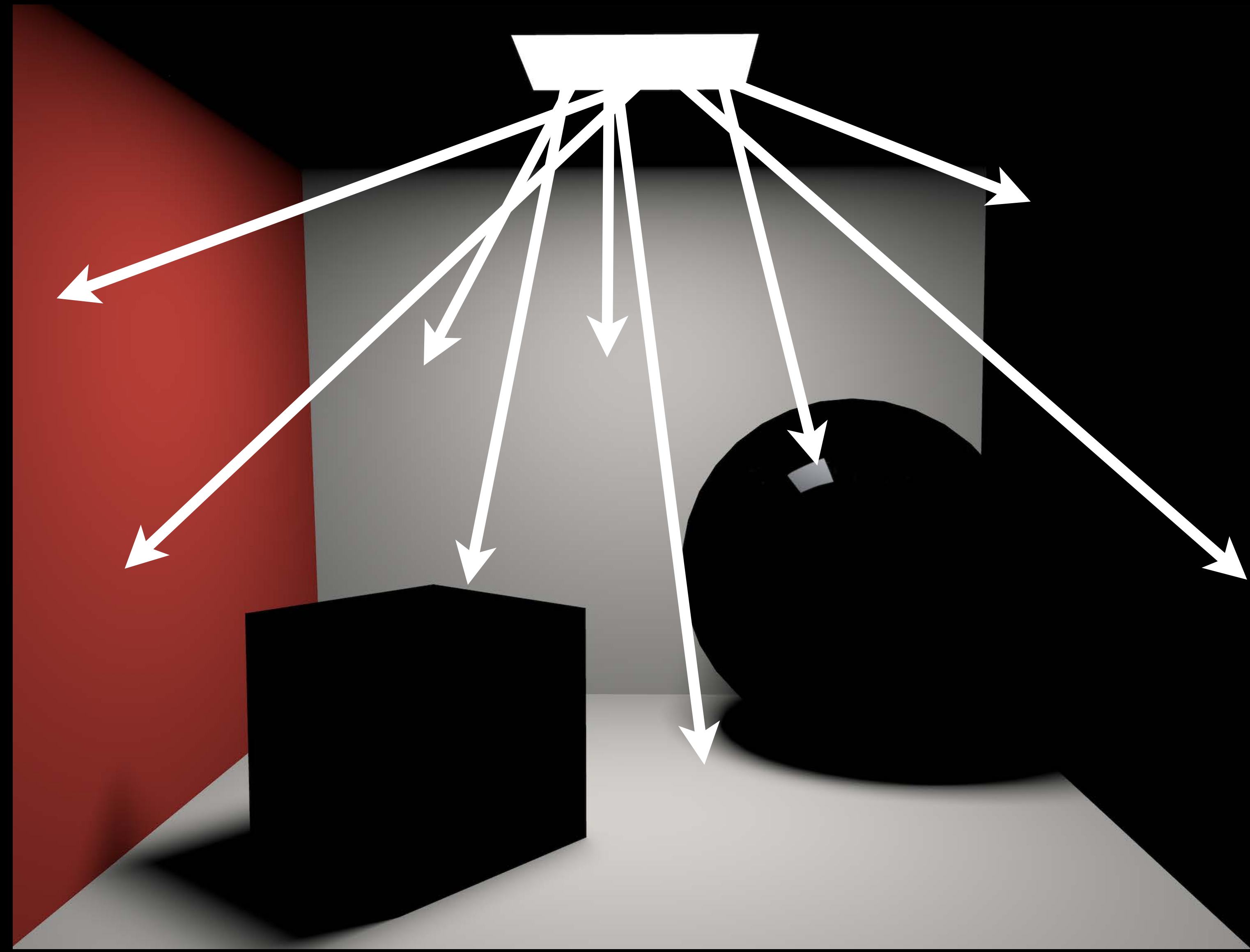


4

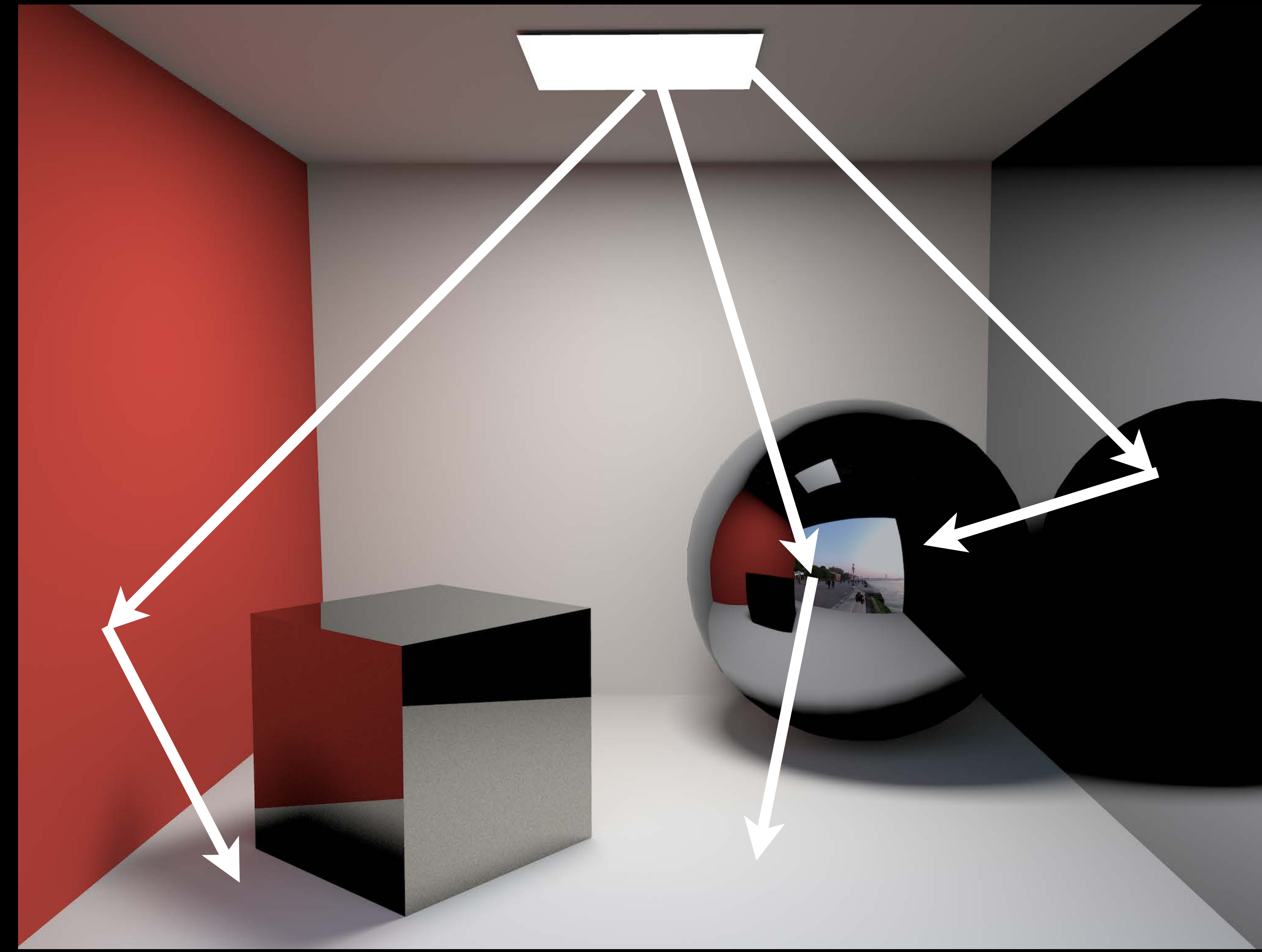


Global Illumination

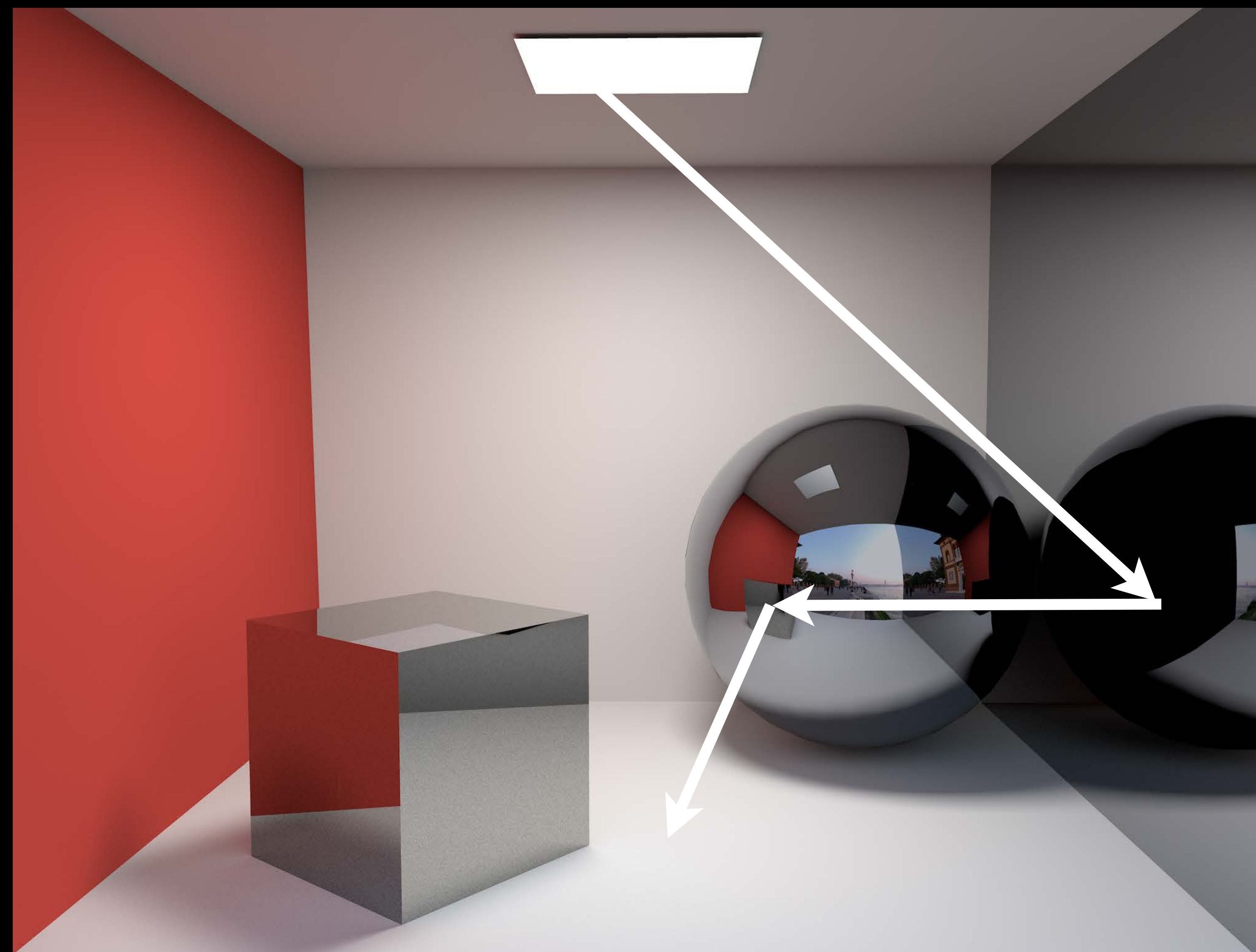
1



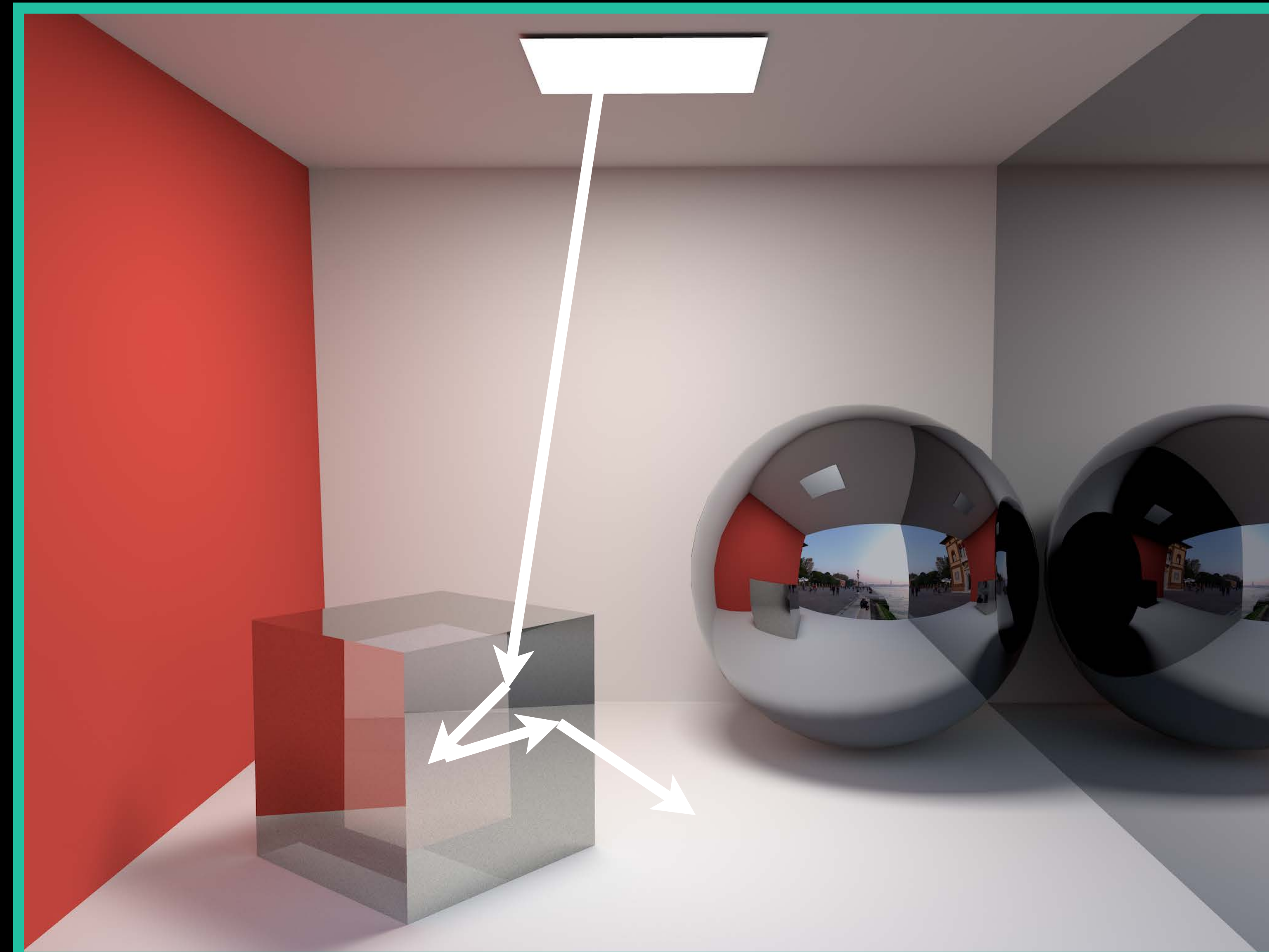
2



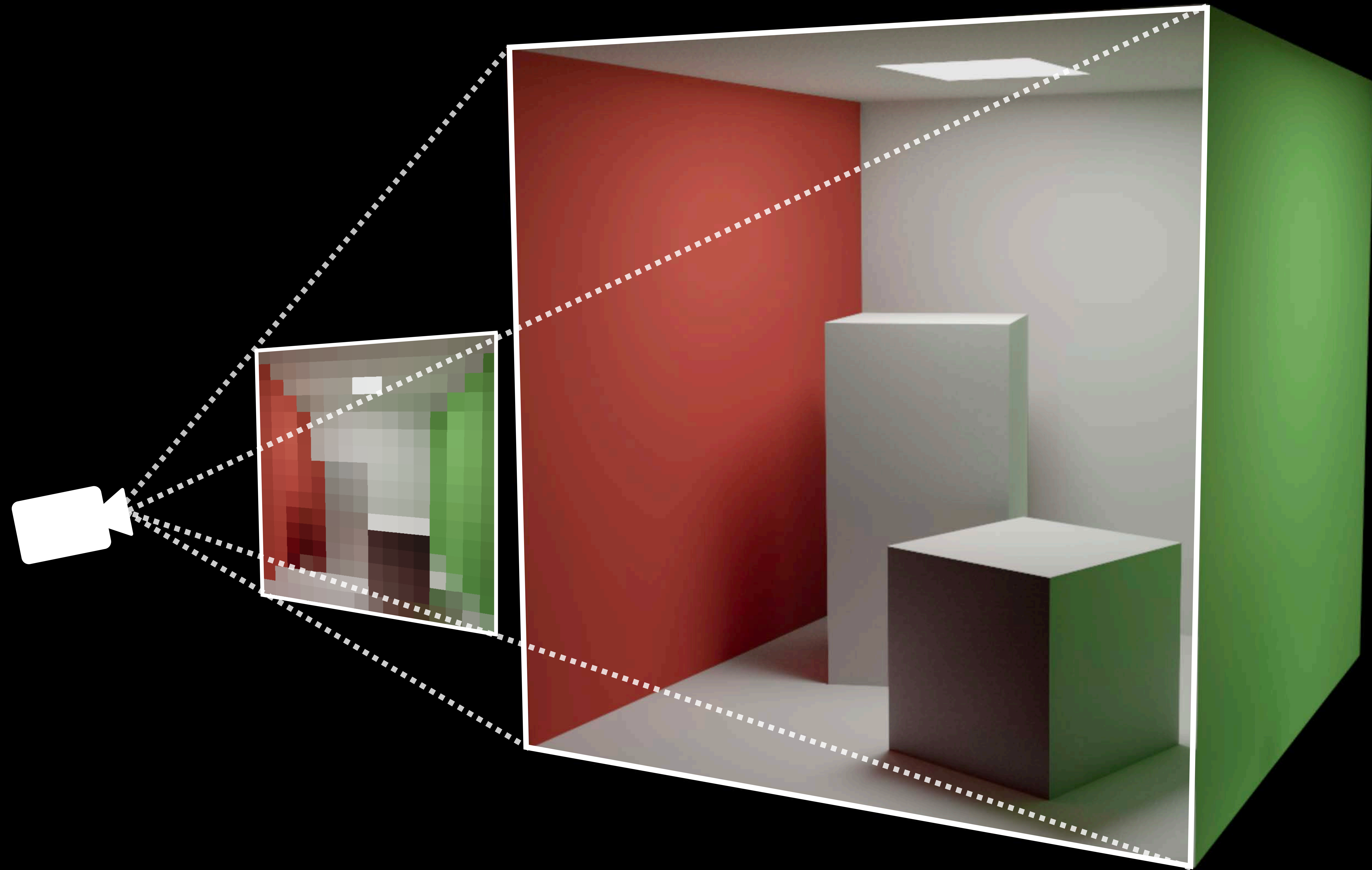
3



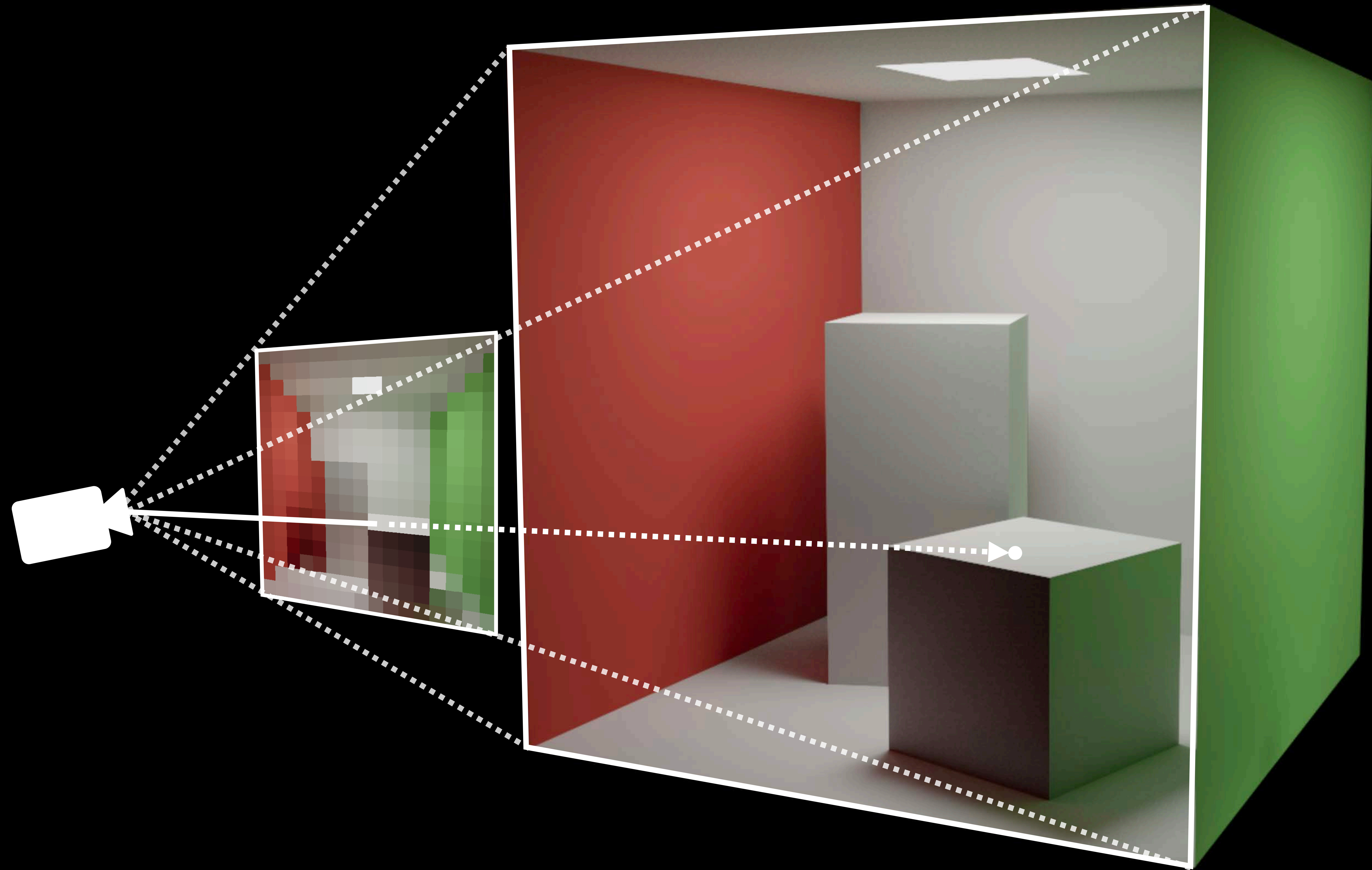
4



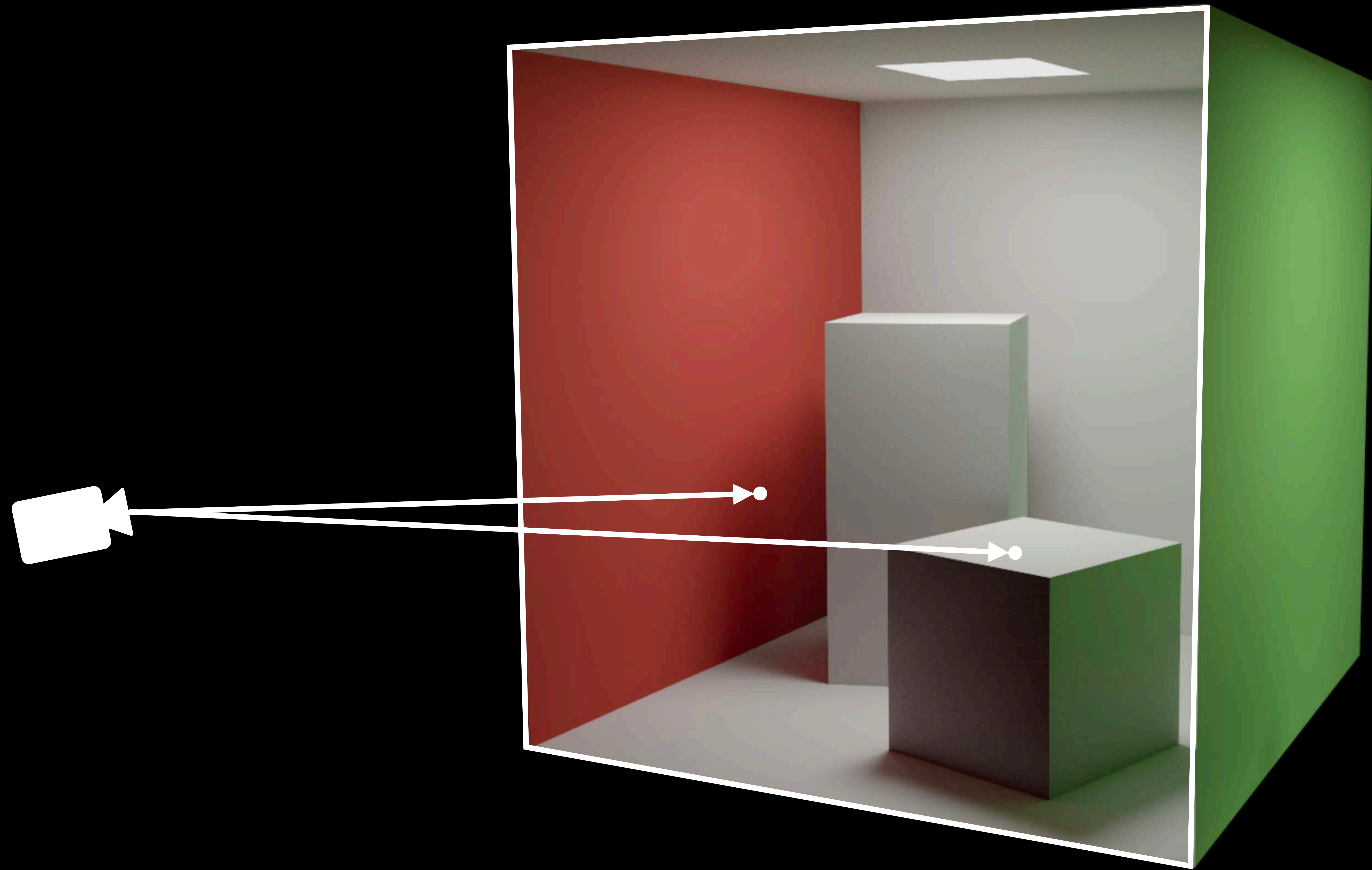
Global Illumination



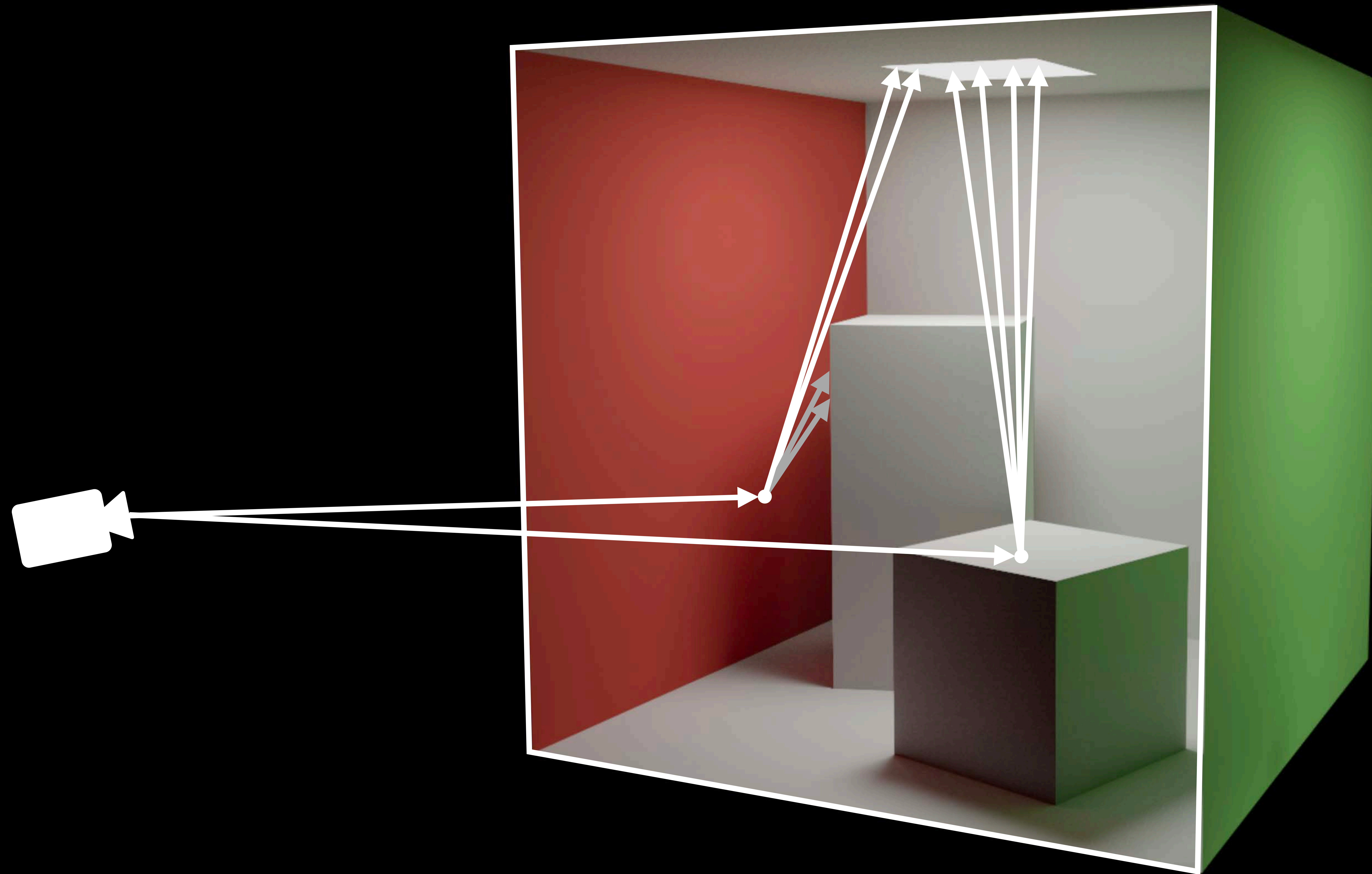
Global Illumination



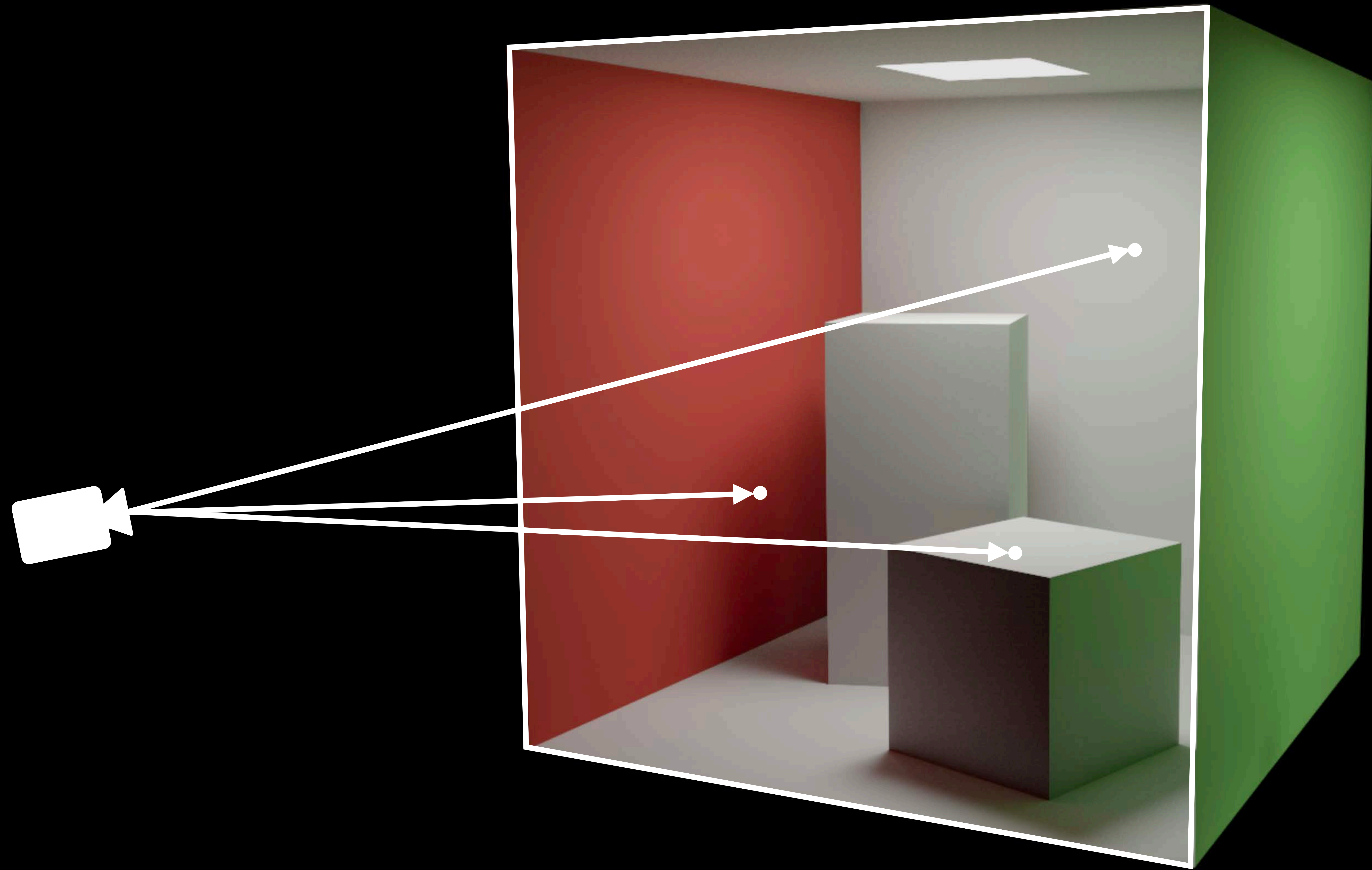
Global Illumination



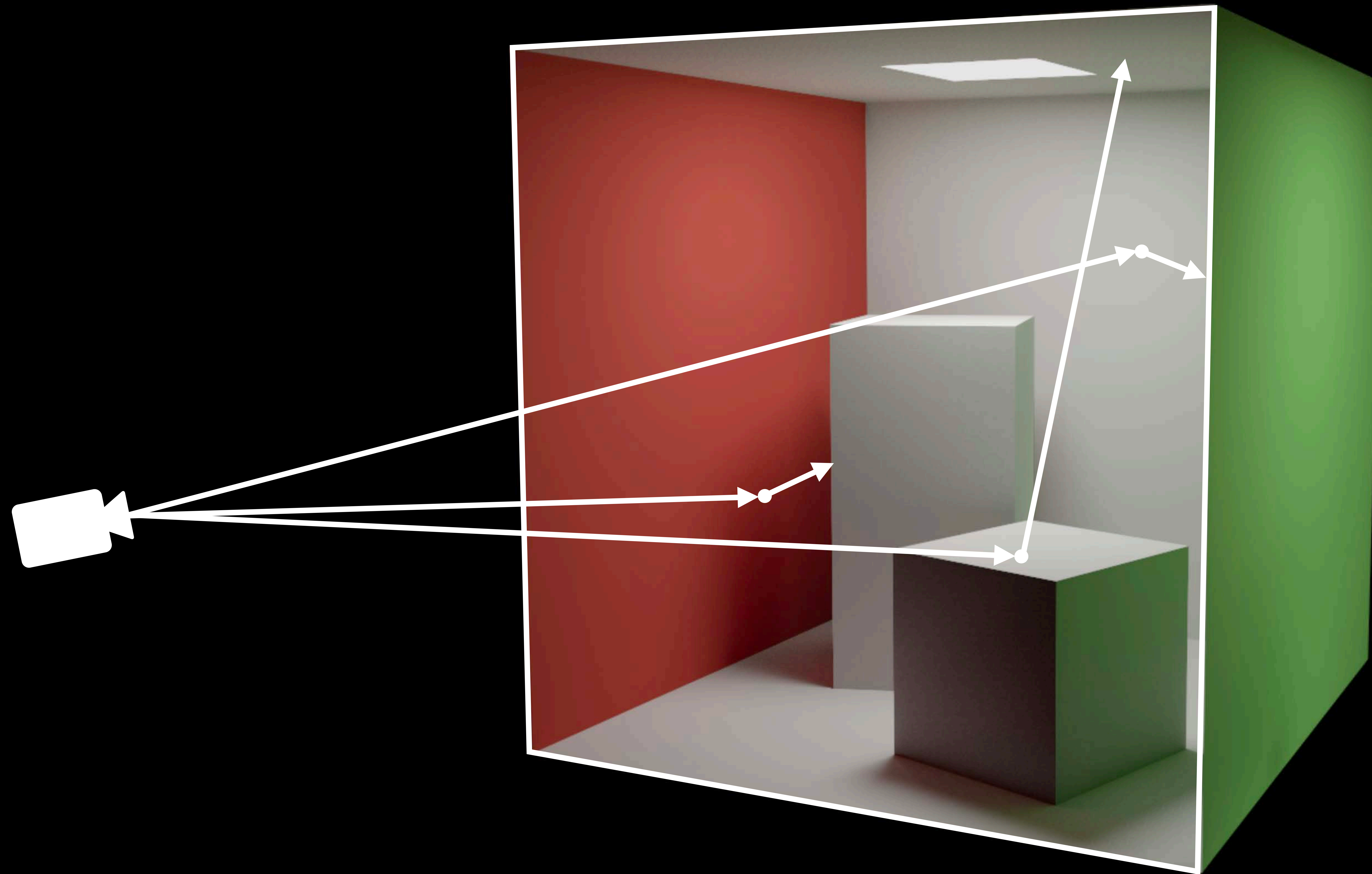
Global Illumination



Global Illumination



Global Illumination

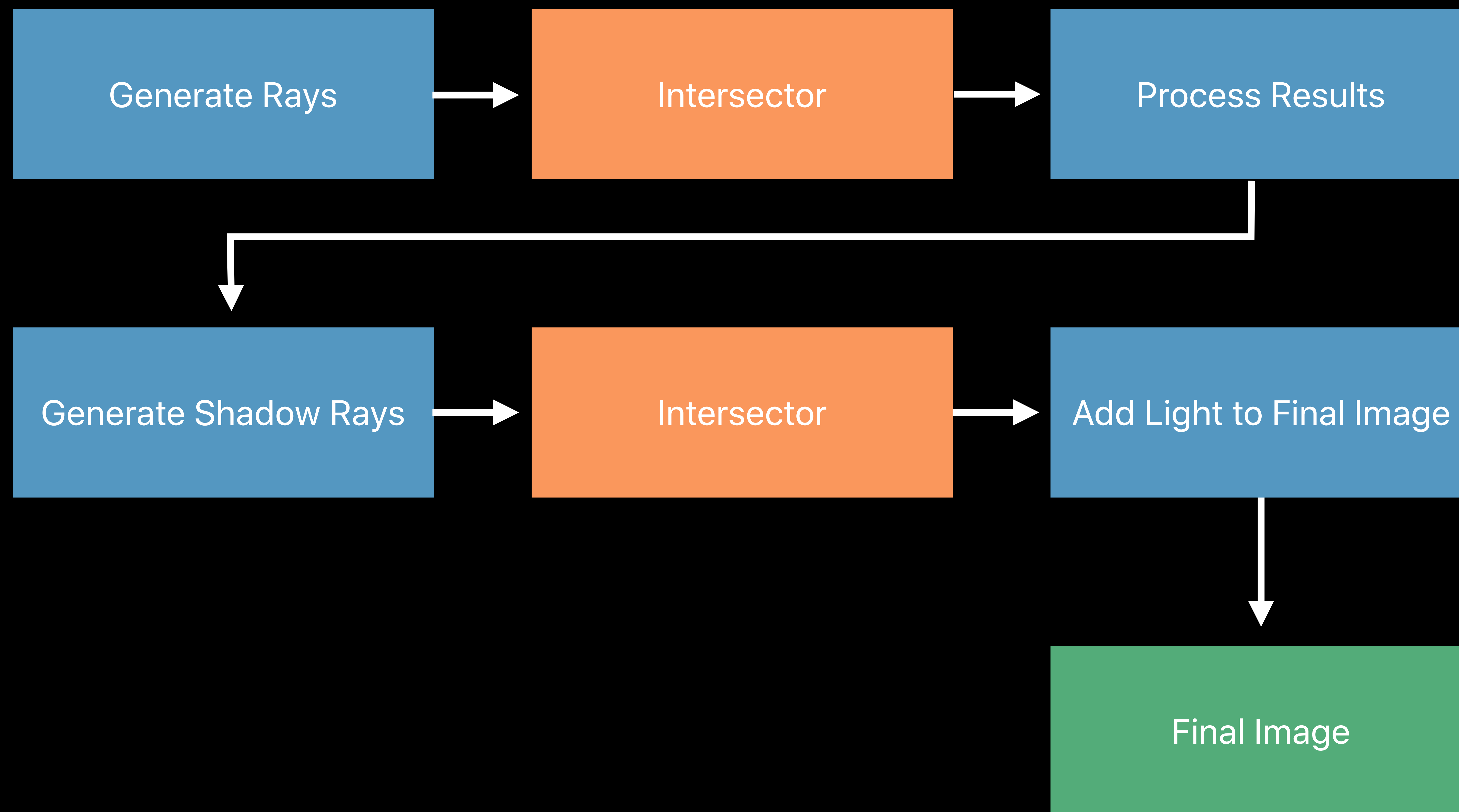


Global Illumination

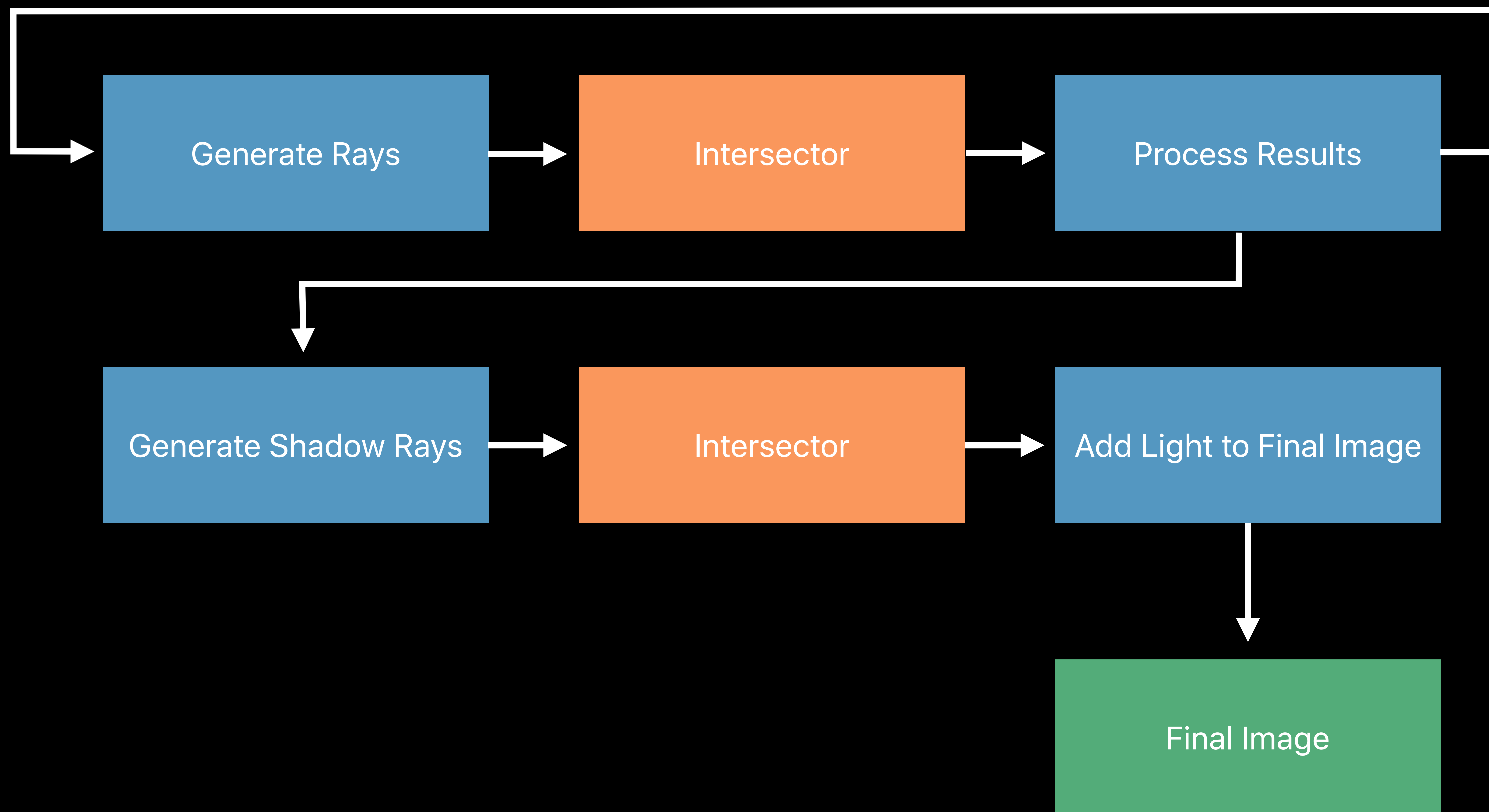
Global Illumination



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

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

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

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

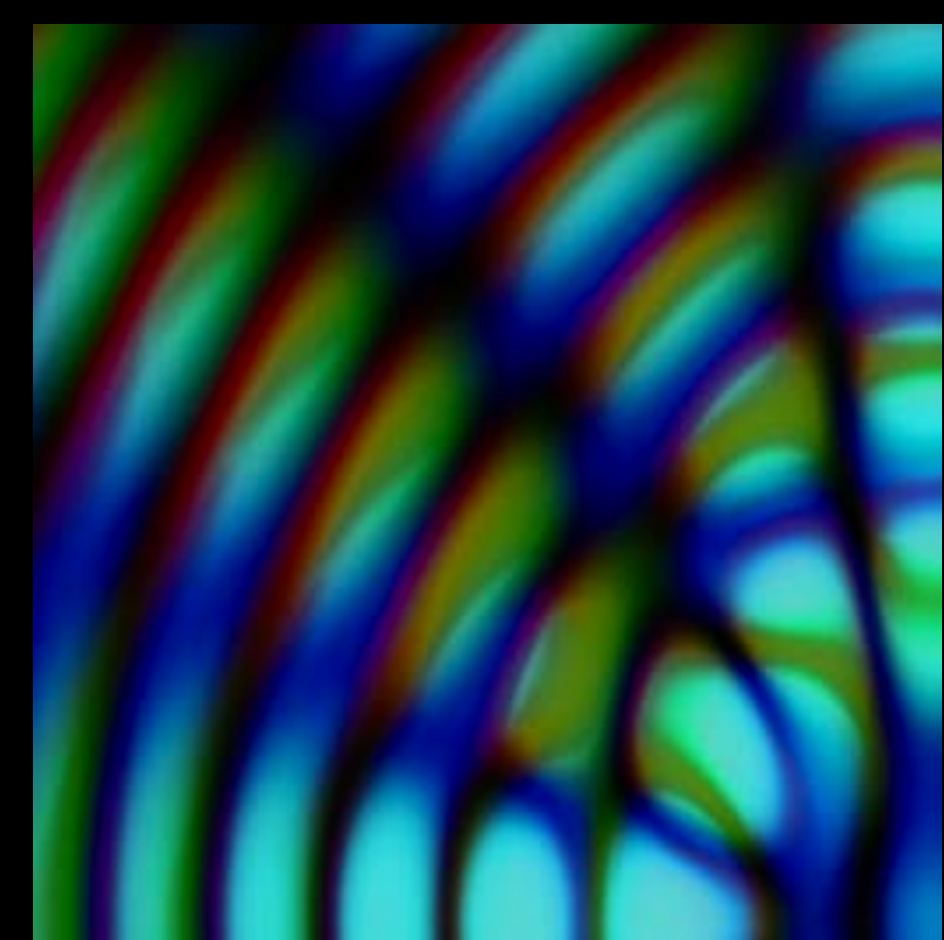
- Track simultaneously live variables
- Don't hold onto structs
- Be careful with loop counters, function calls

Textures

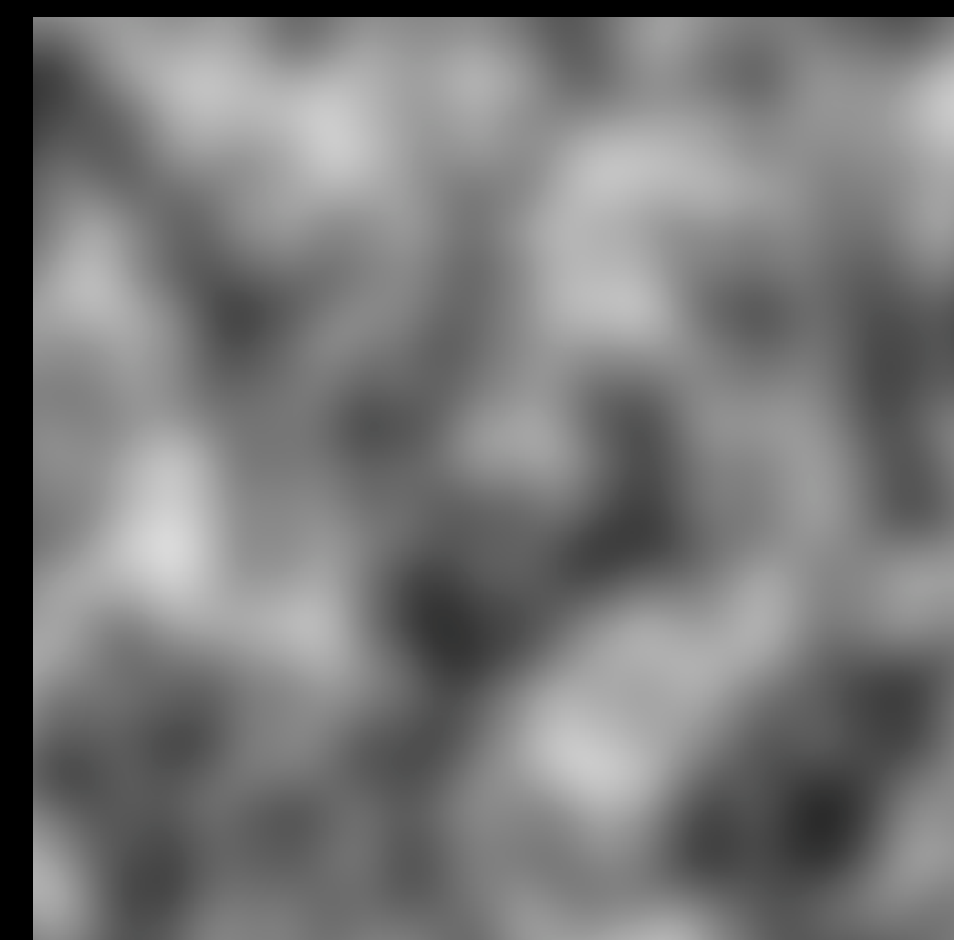
Can't predict what surface a ray will intersect

Sponza scene has 76 textures

Quickly run out of binding slots



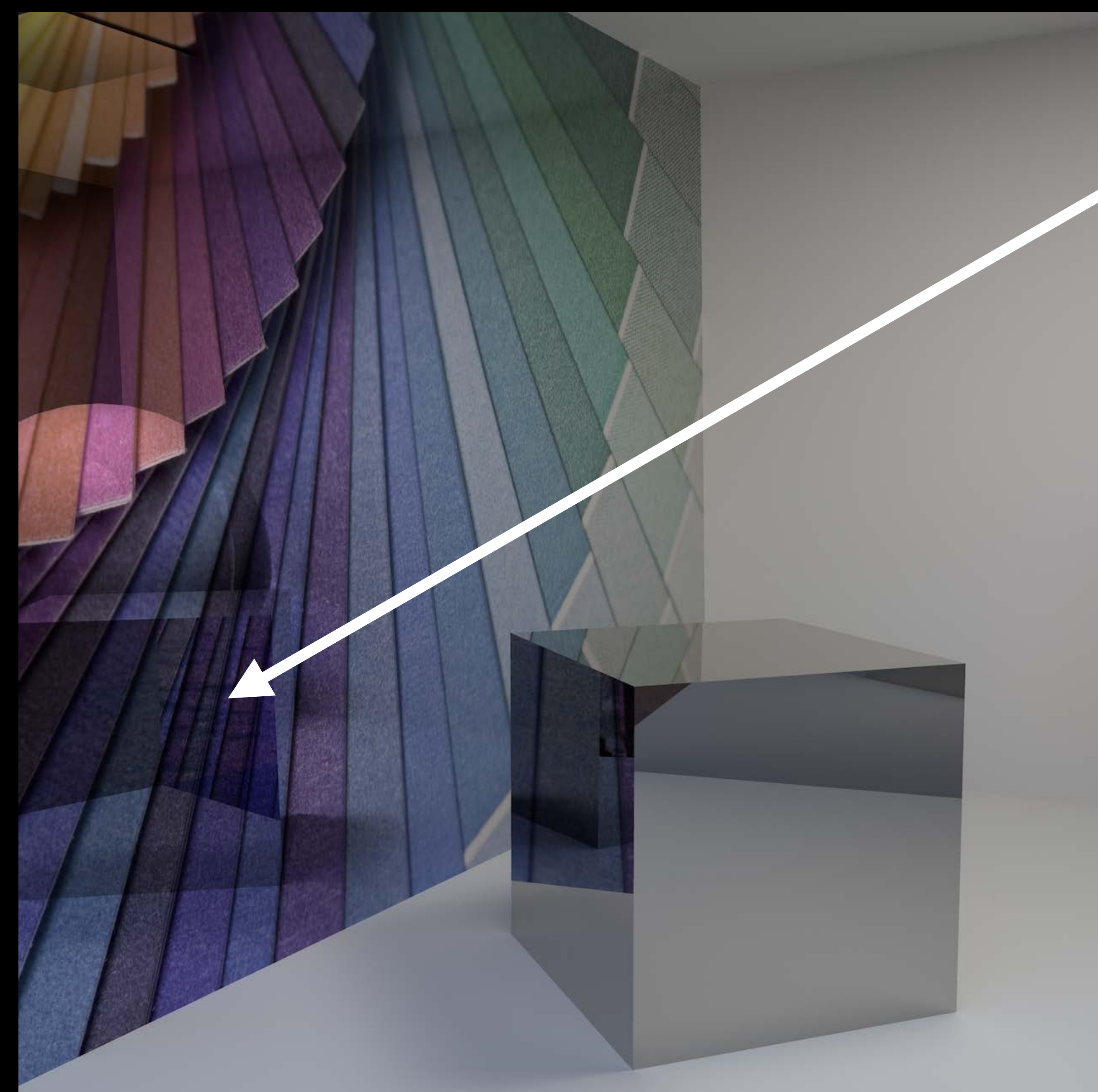
...



...



...



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

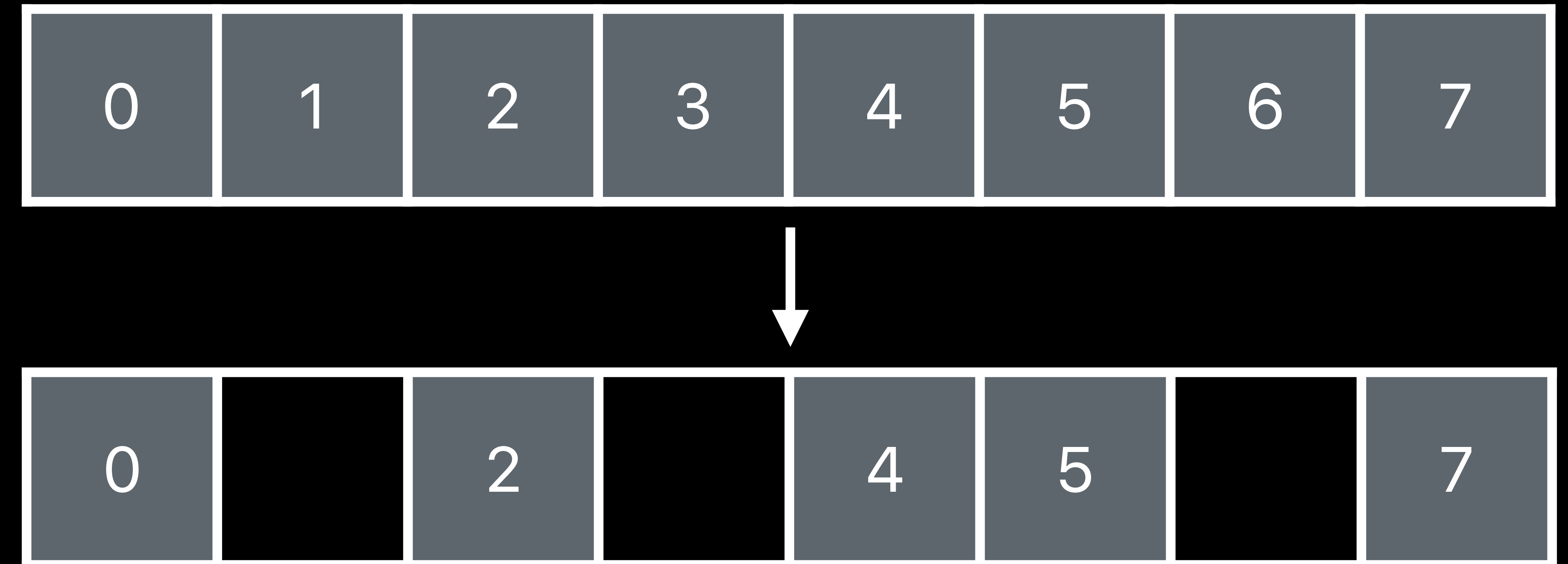
0	1	2	3	4	6	6	7
---	---	---	---	---	---	---	---

Ray Buffer

Inactive Rays



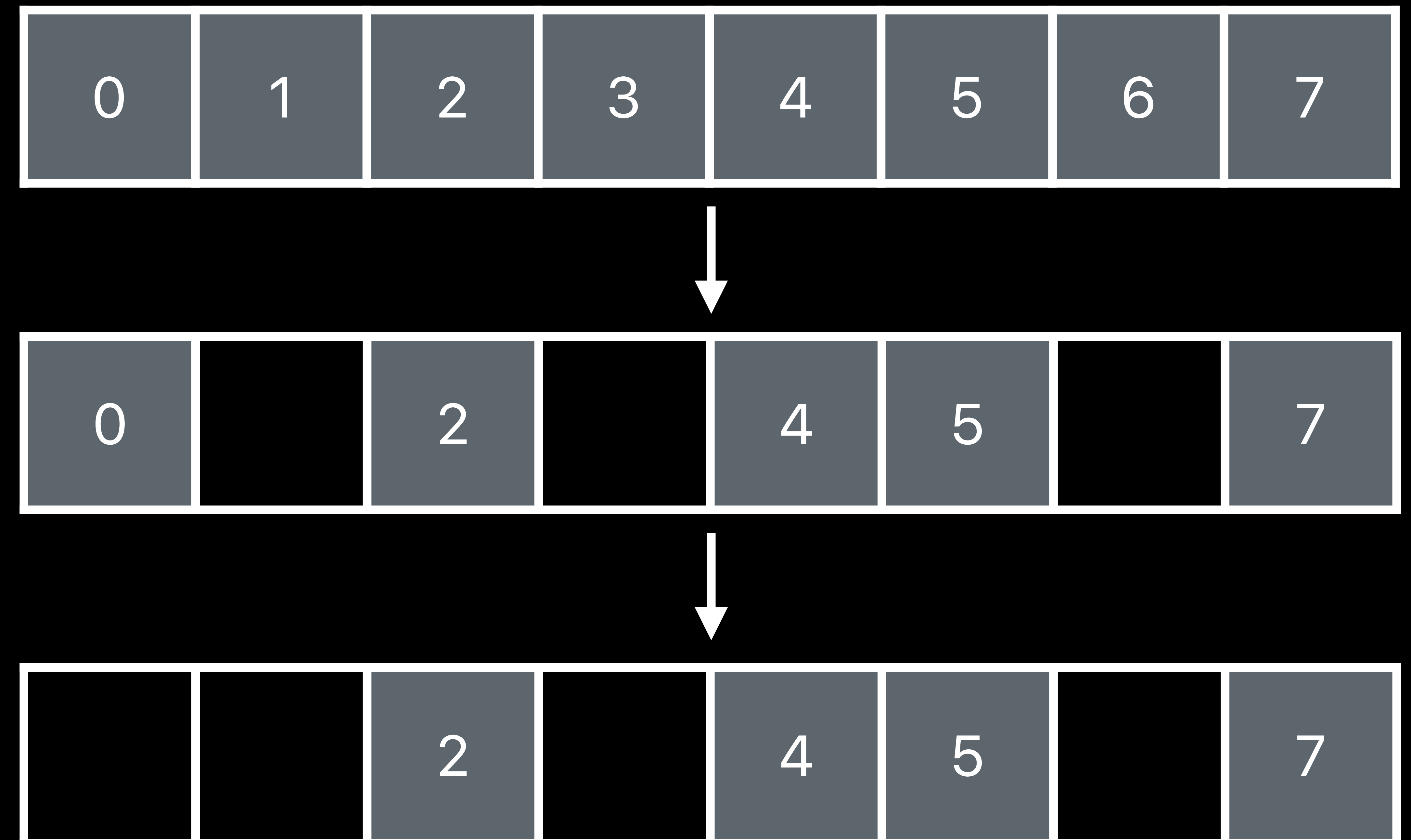
Second iteration: 57% of rays active



Inactive Rays



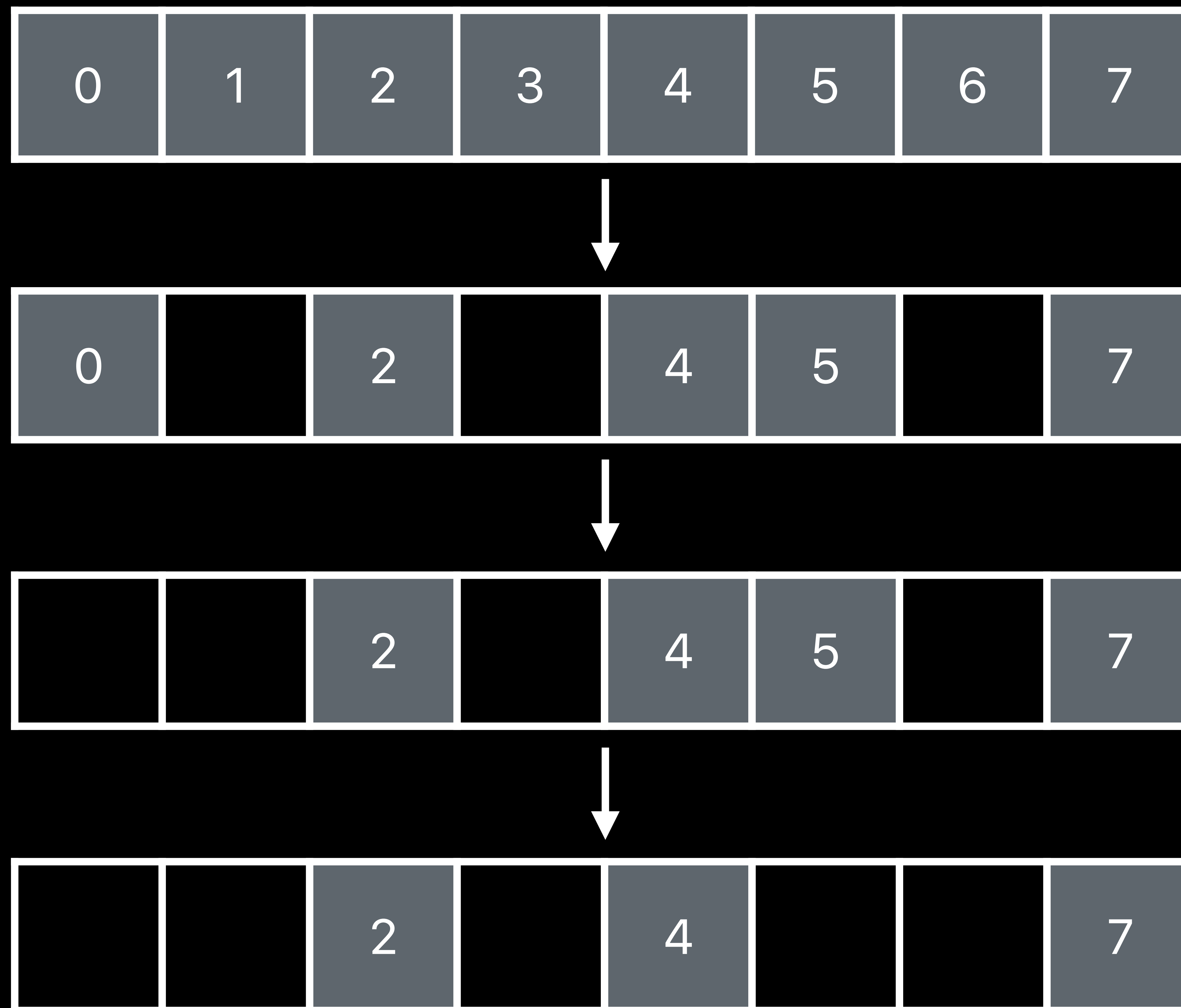
Third iteration: 43% of rays active



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

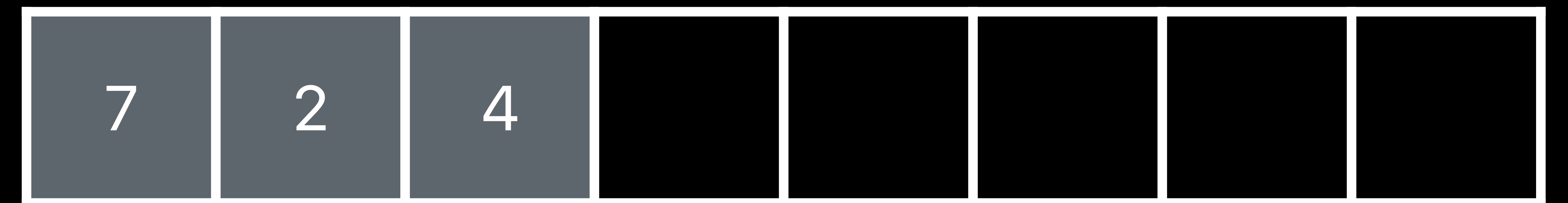


Ray Compaction

Only add active rays to the next ray buffer

Threadgroups are fully utilized

Also works for shadow rays

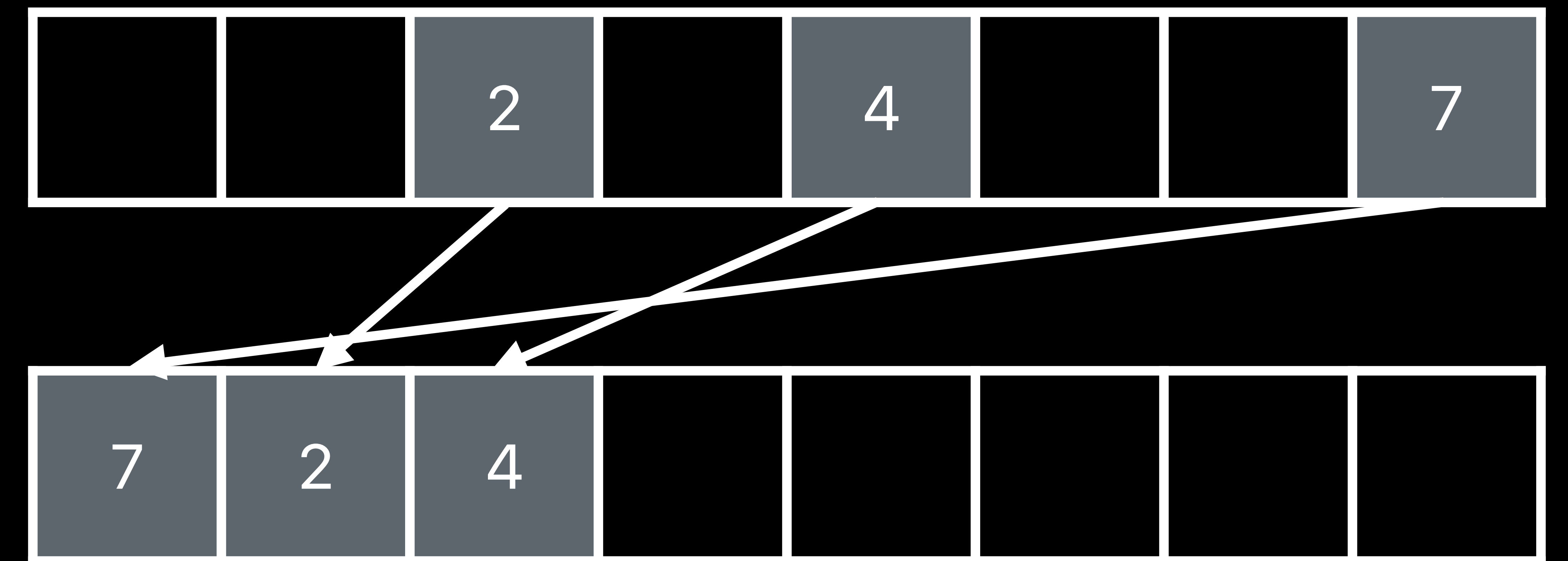


Ray Compaction

Only add active rays to the next ray buffer

Threadgroups are fully utilized

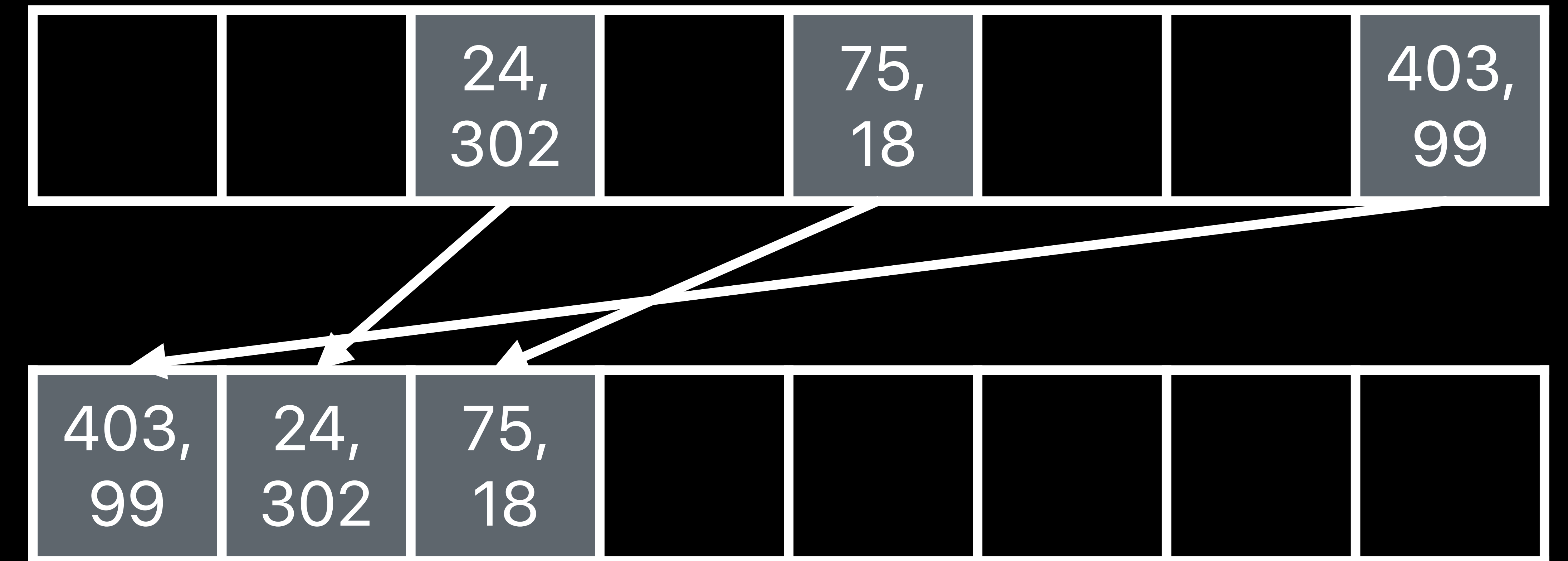
Also works for shadow rays



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

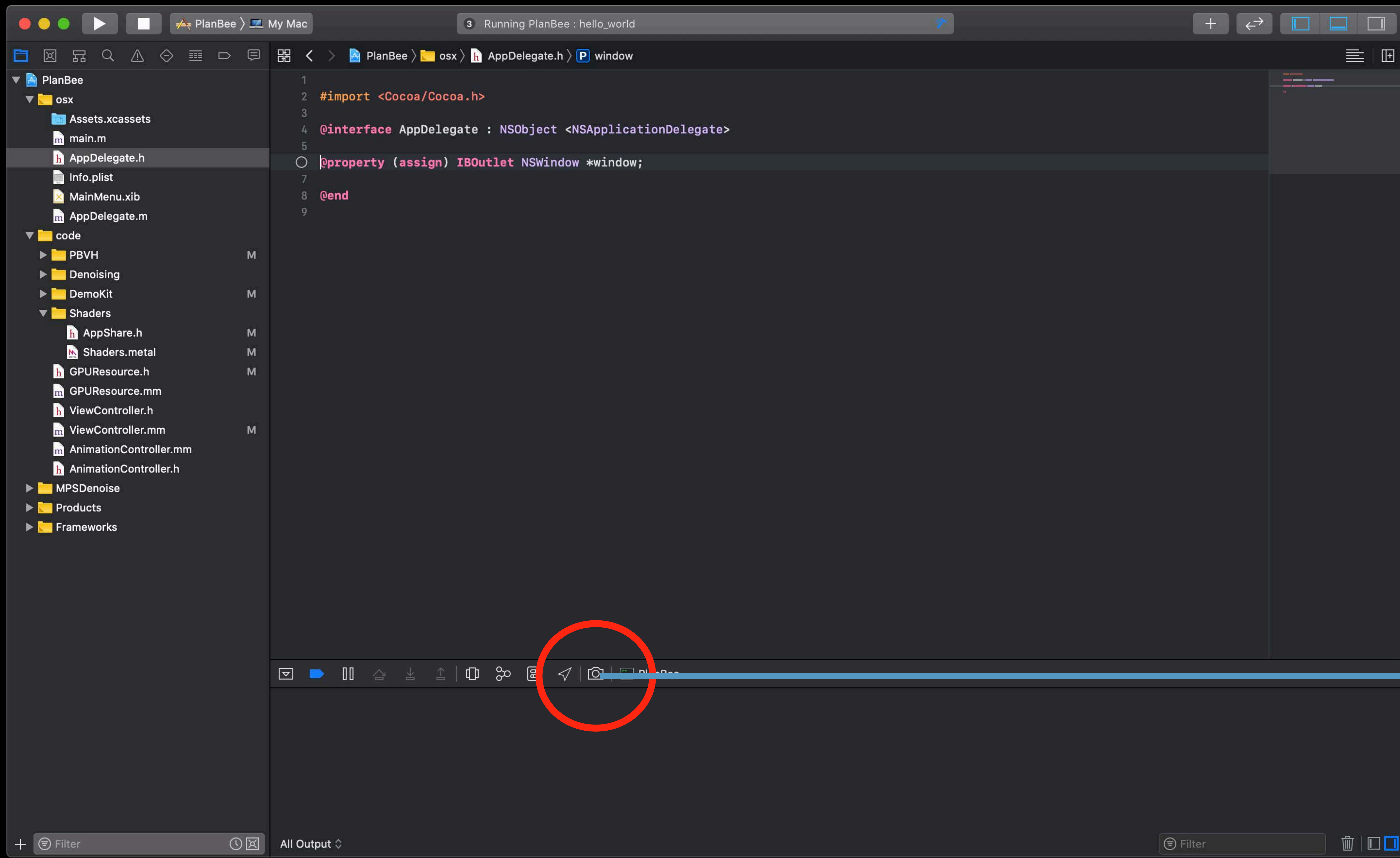
Debugging Image Corruption

Xcode makes this a breeze to debug

Frame capture



Debugging with Xcode



Debugging with Xcode

The screenshot shows the Xcode GPU Frame Debugger interface. The top toolbar includes a play button, a stop button, and a refresh button. The main window is titled "PlanBee - Debugging GPU Frame" and shows a list of draw calls on the left. The selected draw call is "2373 [drawPrimitives:... 317.72 μs]". The right pane displays the details of this draw call, including a table of resources and a 3D render view.

Label	Type	Size	Details
▼ Vertex			
Geometry	Post Vertex Trans...		
VS_FullscreenTriangle	Vertex Function		Library 0x60000175d380...
▼ Fragment			
Texture 0x100d44070	Texture 0	2048 × 1064	RGBA16Float
FS_UpdateDrawable	Fragment Function		Library 0x60000175d380...
▼ Attachments			
CAMetalLayer Drawable	Color 0	1024 × 532	BGRA8Unorm_sRGB

The 3D render view shows a scene with a white building and a path. A blue arrow points from the selected draw call in the left pane to the render view. The bottom pane shows the "RenderPipeline Performance" and "Update Drawable" details.

The screenshot shows the Xcode GPU Frame Debugger interface with a list of draw calls. The top toolbar includes a play button, a stop button, and a refresh button. The main window is titled "PlanBee" and shows a list of draw calls on the left. The selected draw call is "2351 [drawPrimitives:... 318.81 μs]".

- ▶ 2342 0x100d4a380 <- [0x100...
- ▼ Update Drawable 318.81 μs !
 - ▶ 2343 Update Drawable <- [r...
 - ▶ 2344 [setLabel:"Update Dra...
 - ▶ 2345 [setRenderPipelineStat...
 - ▶ 2346 [setDepthStencilState:...
 - ▶ 2347 [setCullMode:None]
 - ▶ 2348 [setFrontFacingWindin...
 - ▶ 2349 [setTriangleFillMode:Fill]
 - ▶ 2350 [setFragmentTexture:0...
 - ▶ 2351 [drawPrimitives:... 318.81 μs
 - ▶ 2352 [endEncoding] !
- ▶ 2353 [presentDrawable:0x600...
- ▼ Update Drawable 317.81 μs !
 - ▶ 2354 Update Drawable <- [r...
 - ▶ 2355 [setLabel:"Update Dra...
 - ▶ 2356 [setRenderPipelineStat...
 - ▶ 2357 [setDepthStencilState:...

Debugging with Xcode

The screenshot displays the Xcode GPU Frame Debugger interface. On the left, a list of rendering operations is shown with their execution times. The middle section contains a table of resources used in the frame:

Label	Type	Size	Details
▼ Vertex			
Geometry	Post Vertex Trans...		
VS_FullscreenTriangle	Vertex Function		Library 0x60000175d380...
▼ Fragment			
Texture 0x100d44070	Texture 0	2048 x 1064	RGBA16Float
FS_UpdateDrawable	Fragment Function		Library 0x60000175d380...
▼ Attachments			
CAMetalLayer Drawable	Color 0	1024 x 532	BGRA8Unorm_sRGB

On the right, a 3D scene view shows a courtyard with chess pieces. Blue arrows indicate the flow of information: one arrow points from the 'VS_FullscreenTriangle' resource to the scene view, and another points from the 'FS_UpdateDrawable' resource to the scene view. A third arrow points from the 'dispatchThrea...' operation in the list to the scene view.



Debugging with Xcode

The screenshot displays the Xcode GPU Frame Debugger interface. The top toolbar shows the application is running. The main window is titled "PlanBee - Debugging GPU Frame" and shows a compute pipeline state for a dispatch thread.

Compute Pipeline State:

Label	Type	Size	Details
▼ Compute			
Texture 0x100d44070	Texture 0	2048 x 1064	RGBA16Float
Buffer 0x100b05a80	Buffer 0	1 MB	Offset: 0x7600
Buffer 0x100d459f0	Buffer 1	24.9 MB	Offset: 0x0
Buffer 0x100d45b50	Buffer 2	8.3 MB	Offset: 0x0
Buffer 0x100d45cb0	Buffer 3	4.2 MB	Offset: 0x0
CS_ResolveRaysToTexture	Kernel Function		Library 0x60000175d380...

Pixel Accumulation Table:

Row	Offset	float3 pixelAccumulation
0	0x0	2.180 1.580 0.000
1	0xC	1.967 1.494 0.000
2	0x18	2.072 1.537 0.000
3	0x24	2.320 1.635 0.000
4	0x30	2.321 1.642 0.000
5	0x3C	2.293 1.668 0.105
6	0x48	2.215 1.716 0.329
7	0x54	2.156 1.725 0.352
8	0x60	1.944 1.486 0.000
9	0x6C	1.920 1.475 0.000
10	0x78	1.928 1.479 0.000
11	0x84	2.006 1.510 0.000
12	0x90	1.845 1.490 0.000
13	0x9C	1.824 1.514 0.066
14	0xA8	2.201 1.670 0.330
15	0xB4	2.433 1.794 0.483
16	0xC0	2.154 1.573 0.001
17	0xCC	2.300 1.588 0.000
18	0xD8	2.312 1.585 0.000
19	0xE4	2.130 1.550 0.000
20	0xF0	1.771 1.468 0.000
21	0xFC	1.411 1.390 0.015
22	0x108	1.918 1.541 0.249
23	0x114	2.359 1.735 0.453
24	0x120	2.452 1.682 0.009
25	0x12C	2.648 1.692 0.000
26	0x138	2.799 1.717 0.000
27	0x144	2.483 1.665 0.000
28	0x150	2.046 1.560 0.000
29	0x15C	1.475 1.403 0.000
30	0x168	1.659 1.380 0.140
31	0x174	1.986 1.450 0.294
32	0x180	2.776 1.789 0.062
33	0x18C	2.835 1.723 0.000
34	0x198	2.846 1.725 0.000
35	0x1A4	2.542 1.674 0.000

The bottom of the interface shows a list of pipeline states, including "ComputePipelineState 0x10390cb70 (MTLComputePipelineState) CS_ResolveRaysToTexture", "ComputePipeline Performance 953.33 μs (0.9%)", and "Texture 0 (MTLTexture) 0x100d44070 - 2048 x 1064, RGBA16Float".

Debugging with Xcode

The screenshot displays the Xcode GPU Frame Debugger interface. The top toolbar shows the application is running. The left sidebar contains a hierarchical tree of GPU operations, with the 'Resolve Rays To Texture' operation selected. The main area is divided into several panels:

- Compute Pipeline State:** Shows the active pipeline state for the selected operation: `ComputePipelineState 0x10390cb70 (MTLComputePipelineState) CS_ResolveRaysToTexture`.
- Resources:** A table listing the resources used by the pipeline state. The selected resource is Buffer 3.
- Attachments:** A table showing the attachment details for the selected resource, including row, offset, and pixel sample count.
- CommandBuffer:** Shows the command buffer state for the selected operation, including the texture and buffers used.

Label	Type	Size	Details
▼ Compute			
Texture 0x100d44070	Texture 0	2048 x 1064	RGBA16Float
Buffer 0x100b05a80	Buffer 0	1 MB	Offset: 0x7600
Buffer 0x100d459f0	Buffer 1	24.9 MB	Offset: 0x0
Buffer 0x100d45b50	Buffer 2	8.3 MB	Offset: 0x0
Buffer 0x100d45cb0	Buffer 3	4.2 MB	Offset: 0x0
CS_ResolveRaysToTexture	Kernel Function		Library 0x60000175d380...

Row	Offset	ushort pixelSampleCount
0	0x0	22
1	0x2	22
2	0x4	22
3	0x6	22
4	0x8	22
5	0xA	22
6	0xC	22
7	0xE	22
8	0x10	22
9	0x12	22
10	0x14	22
11	0x16	22
12	0x18	22
13	0x1A	22
14	0x1C	22
15	0x1E	22
16	0x20	22
17	0x22	22
18	0x24	22
19	0x26	22
20	0x28	22
21	0x2A	22
22	0x2C	22
23	0x2E	22
24	0x30	22
25	0x32	22
26	0x34	22
27	0x36	22
28	0x38	22
29	0x3A	22
30	0x3C	22
31	0x3E	22
32	0x40	22
33	0x42	22
34	0x44	22
35	0x46	22

Debugging with Xcode

The screenshot displays the Xcode IDE interface for debugging a GPU frame. The top toolbar includes standard window and navigation icons. The main workspace is divided into several panels:

- Left Panel (Hierarchy):** A tree view of GPU command buffers and pipeline states. The 'Resolve Rays To Texture' command buffer is expanded, showing a list of sub-commands such as 'Set Dispatch Dimensions', 'Sample Surface', 'Clear Buffer', and 'Generate Shadow Rays'. The 'Resolve Rays To Texture' command is selected, showing its execution time as 953.33 μs.
- Center Panel (Code):** A code editor showing the implementation of the 'CS_ResolveRaysToTexture' kernel. The code is written in Metal and includes a kernel function signature, parameter lists, and a loop that iterates over threads to resolve rays to texture coordinates. The current line of execution is highlighted in grey.
- Right Panel (Texture View):** A preview window showing the rendered scene, which is a ray-traced image of a classical building with arches and columns. The texture being used is identified as 'Texture 0x100d44070'.
- Bottom Panel (Command Buffers):** A list of GPU command buffers and pipeline states. The selected command buffer is 'ComputePipelineState 0x10390cb70 (MTLComputePipelineState) CS_ResolveRaysToTexture'. Other items include 'ComputePipeline Performance', 'Texture 0 (MTLTexture) 0x100d44070 - 2048 x 1064, RGBA16Float', and several buffers.

The bottom status bar shows the current selection as 'No Selection' and provides navigation controls for the command buffers.

Debugging with Xcode

The screenshot displays the Xcode IDE interface for debugging a GPU frame. The top toolbar shows the play button and other debugging controls. The main window is divided into several panels:

- Timeline (Left):** A vertical list of GPU events with their durations and addresses. The event '2337 [dispatchThrea... 953.33 us' is highlighted in red.
- Code Editor (Center):** Shows the source code for the compute shader 'CS_ResolveRaysToTexture'. The line `float3 resolved = pixelAccumulation[linearAddress];` is highlighted with a red box.
- 3D Viewer (Right):** Displays a rendered scene of a building with arches and trees. A texture 'Texture 0x100d44070' is visible in the bottom right corner of the viewer.
- Command Buffer (Bottom Left):** Shows a list of GPU commands, including 'ComputePipelineState 0x10390cb70' and 'Texture 0 (MTLTexture) 0x100d44070'.

Debugging with Xcode

The screenshot displays the Xcode GPU Frame Debugger interface. On the left, a hierarchical tree view shows the execution flow of a GPU frame, with the 'Resolve Rays To Texture' step selected. The central pane shows the source code for the `CS_ResolveRaysToTexture` kernel. A red box highlights a specific line of code:

```
float3 resolved = pixelAccumulation[linearAddress] / pixelSampleCount[linearAddress];
```

On the right, a 3D view shows a ray-traced scene of a building with arches. Below the 3D view, a texture inspector shows details for 'Texture 0x100d44070' at 28% zoom. At the bottom, a console pane displays the state of the compute pipeline, including the pipeline state object, performance metrics, and buffer information.

Debugging with Xcode

The screenshot displays the Xcode IDE interface for debugging a GPU frame. The top toolbar includes standard window and navigation icons. The main workspace is divided into several panels:

- Left Panel:** A hierarchical tree view of GPU command buffers and drawables. The 'Resolve Rays To Texture' command is selected, showing its sub-commands and timing information.
- Center Panel:** A code editor showing the implementation of the `CS_ResolveRaysToTexture` kernel. The code is as follows:

```
kernel void CS_ResolveRaysToTexture(  
    uint tid [[thread_position_in_grid]],  
    constant CS_ResolveRaysToTextureConstants& constants [[buffer(0)]],  
    const device packed_float3* pixelAccumulation [[buffer(1)]],  
    const device ushort2* pixelCoordinate [[buffer(2)]],  
    const device ushort* pixelSampleCount [[buffer(3)]],  
    texture2d<float, access::write> framebuffer [[texture(0)]]  
)  
{  
    if (tid >= constants.pixelCount) {  
        return;  
    }  
  
    ushort2 coordinate = pixelCoordinate[constants.firstPixel + tid];  
    uint linearAddress = BlockLinearAddress((uint2)coordinate,  
        constants.resolution);  
  
    float3 resolved = pixelAccumulation[linearAddress];  
  
    framebuffer.write(float4(resolved * constants.blendColor, 1.0f),  
        coordinate);  
}
```
- Right Panel:** A 3D viewport showing a rendered scene of a classical building with arches and a courtyard. A texture is selected in the bottom right corner, showing a zoomed-in view of a portion of the scene.
- Bottom Panel:** A console and command buffer viewer. It shows the active compute pipeline state, performance metrics (953.33 μs, 0.9%), and a list of buffers and textures used by the kernel.

Debugging with Xcode

The screenshot displays the Xcode IDE interface for debugging a GPU frame. The main window is titled "PlanBee - Debugging GPU Frame" and shows a ray-traced scene of a chessboard with pieces. The interface is divided into several panels:

- Left Panel (Hierarchy):** A tree view of the GPU pipeline. The "Resolve Rays To Texture" node is selected, showing its sub-nodes and execution time (953.33 μs).
- Top Center (Code Editor):** Displays the source code for the compute shader `CS_ResolveRaysToTexture`. The code is in Objective-C++ and defines a kernel function that reads from a texture and writes to a framebuffer. The current line of execution is highlighted in blue.
- Right Panel (Texture Inspector):** Shows the texture being accessed, identified as "Texture 0x100d44070". It includes a zoom control set to 28%.
- Bottom Panel (Debug Console):** Shows the execution log for the selected compute pipeline state, including performance metrics and resource usage.

```
kernel void CS_ResolveRaysToTexture(  
    uint tid [[thread_position_in_grid]],  
    constant CS_ResolveRaysToTextureConstants& constants [[buffer(0)]],  
    const device packed_float3* pixelAccumulation [[buffer(1)]],  
    const device ushort2* pixelCoordinate [[buffer(2)]],  
    const device ushort* pixelSampleCount [[buffer(3)]],  
    texture2d<float, access::write> framebuffer [[texture(0)]]  
)  
{  
    if (tid >= constants.pixelCount) {  
        return;  
    }  
  
    ushort2 coordinate = pixelCoordinate[constants.firstPixel + tid];  
    uint linearAddress = BlockLinearAddress((uint2)coordinate,  
        constants.resolution);  
  
    float3 resolved = pixelAccumulation[linearAddress]  
        / pixelSampleCount[linearAddress];  
  
    framebuffer.write(float4(resolved * constants.blendColor, 1.0f),  
        coordinate);  
}
```

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;  
};
```

Potential Hotspots/Bottlenecks

- ⚠ Texture unit is heavily stalled - 70%

Counters

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

Before Change

Potential Hotspots/Bottlenecks

- No potential bottlenecks detected

Counters

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

After Change

CommandBuffer 0 @ 0 0x100c29670

- Set Single Buffer Value 6.73 μ s
- Generate Primary Rays 3.45 ms
- MPSRayIntersector 11.61 ms
- Set Dispatch Dimensions 7.97 μ s
- Sample Surface 5.50 ms
- 78 [dispatchThreadgrou... 5.50 ms**
- Clear Buffer 6.67 μ s

Max	Total
20.01 ms	122.92 ms
2,521,472	28,261,680
80.43%	N/A
99%	N/A
89.03%	N/A
99%	N/A
88%	N/A
100%	N/A
91%	N/A

Memory

L2 Throughput	L2 Bandwidth	Color Block Bandwidth
2.92	33.39	0
2.93	33.48	0
0.02	0.31	0
3.32	57.65	357.43
N/A	N/A	N/A

ComputePipelineState 0x100c13f40 (MTLComputePipelineSta...)

ComputePipeline Performance 21.14 ms (17.2%)

Texture 0 (MTLTexture) 0x100c20700 - 8192 x 4096, BC1_RG...

Buffer 0 0x100b1c2e0, Offset=0x00000300

Buffer 1 0x100b376a0

CommandBuffer 0 @ 0 0x100c496e0

- Set Single Buffer Value 6.85 μ s
- Generate Primary Rays 3.43 ms
- MPSRayIntersector 11.59 ms
- Set Dispatch Dimensions 8.09 μ s
- Sample Surface 4.00 ms
- 84 [dispatchThreadgrou... 4.00 ms**
- Clear Buffer 6.52 μ s

Max	Total
19.72 ms	117.03 ms
2,522,560	28,266,800
80.2%	N/A
99%	N/A
90.48%	N/A
99%	N/A
88%	N/A
100%	N/A
91%	N/A

Memory

L2 Throughput	L2 Bandwidth	Color Block Bandwidth
1.17	49.51	0
1.18	49.61	0
0.02	0.31	0
2	57.87	354.18
N/A	N/A	N/A

ComputePipelineState 0x100d0c870 (MTLComputePipelineSt...)

ComputePipeline Performance 19.02 ms (16.3%)

Texture 0 (MTLTexture) 0x100b155a0 - 8192 x 4096, BC1_RG...

Buffer 0 0x100c38d20, Offset=0x00000300

Buffer 1 0x100b1fa60

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

Value	Encoder	Median	Max	Total
4.01 ms	4.01 ms	7.62 μs	19.72 ms	117.03 ms
Surface				
2,073,600	2,073,600	160	2,522,560	28,266,800
13.59%	13.64%	0.11%	80.2%	N/A
99%	99%	56%	99%	N/A
80.62%	80.57%	52.05%	90.48%	N/A
93%	94%	2%	99%	N/A
54.01%	55%	0%	88%	N/A
74.59%	74.55%	81.82%	100%	N/A
39%	40%	0%	91%	N/A

▶ Set Dispatch Dimensions	7.56 μs	L2 Throughput	2.92	2.93	0.02	3.32	N/A
▶ Shade Shadow Rays	6.87 μs	L2 Bandwidth	33.39	33.48	0.31	57.65	N/A
▶ Clear Buffer	6.55 μs	Color Block Bandwidth	0	0	0	357.43	N/A
▶ 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					
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

ComputePipelineState 0x100c13f40 (MTLComputePipelineSta...)

ComputePipeline Performance 21.14 ms (17.2%)

Texture 0 (MTLTexture) 0x100c20700 - 8192 x 4096, BC1_RG...

Buffer 0 0x100b1c2e0, Offset=0x00000300

Buffer 1 0x100b376a0

Potential Hotspots/Bottlenecks

▶ **No potential bottlenecks detected**

FPS: 9 FPS

Counters

Memory: 842.6 MB

CommandBuffer 0 @ 0 0x100c496a0

- 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

ComputePipelineState 0x100d0c870 (MTLComputePipelineSt...)

ComputePipeline Performance 19.02 ms (16.3%)

Texture 0 (MTLTexture) 0x100b155a0 - 8192 x 4096, BC1_RG...

Buffer 0 0x100c38d20, Offset=0x00000300

Buffer 1 0x100b1fa60

PlanBee > My Mac | 4 PlanBee - Debugging GPU Frame | 20

2337 [dispat...up:{256, 1, 1}] | Bound Resources | Automatic | Attachments

Label	Type	Size	Details
Texture 0x100b1fcd0	Texture 0	2048 x 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		Library 0x6000017

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 | Auto | Filter

Texture 0 | 0% | Filter

All Output | 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 x 1152, RGBA16Float

Buffer 0 0x100b27090, Offset=0x00007600

Auto | Filter

No Selection

ComputePipelineState 0x100e02bf0 (MTLComputePipelineState) CS_ResolveRaysToTexture

StageInputDescriptor = (MTLStageInputOutputDescriptor) nil

IsMultipleOfExecutionWidth = (BOOL) NO

ThreadExecutionWidth = (NSUInteger) 64

MaxTotalThreadsPerThreadgroup = (NSUInteger) 1024

Function = (MTLFunction) "CS_ResolveRaysToTexture"

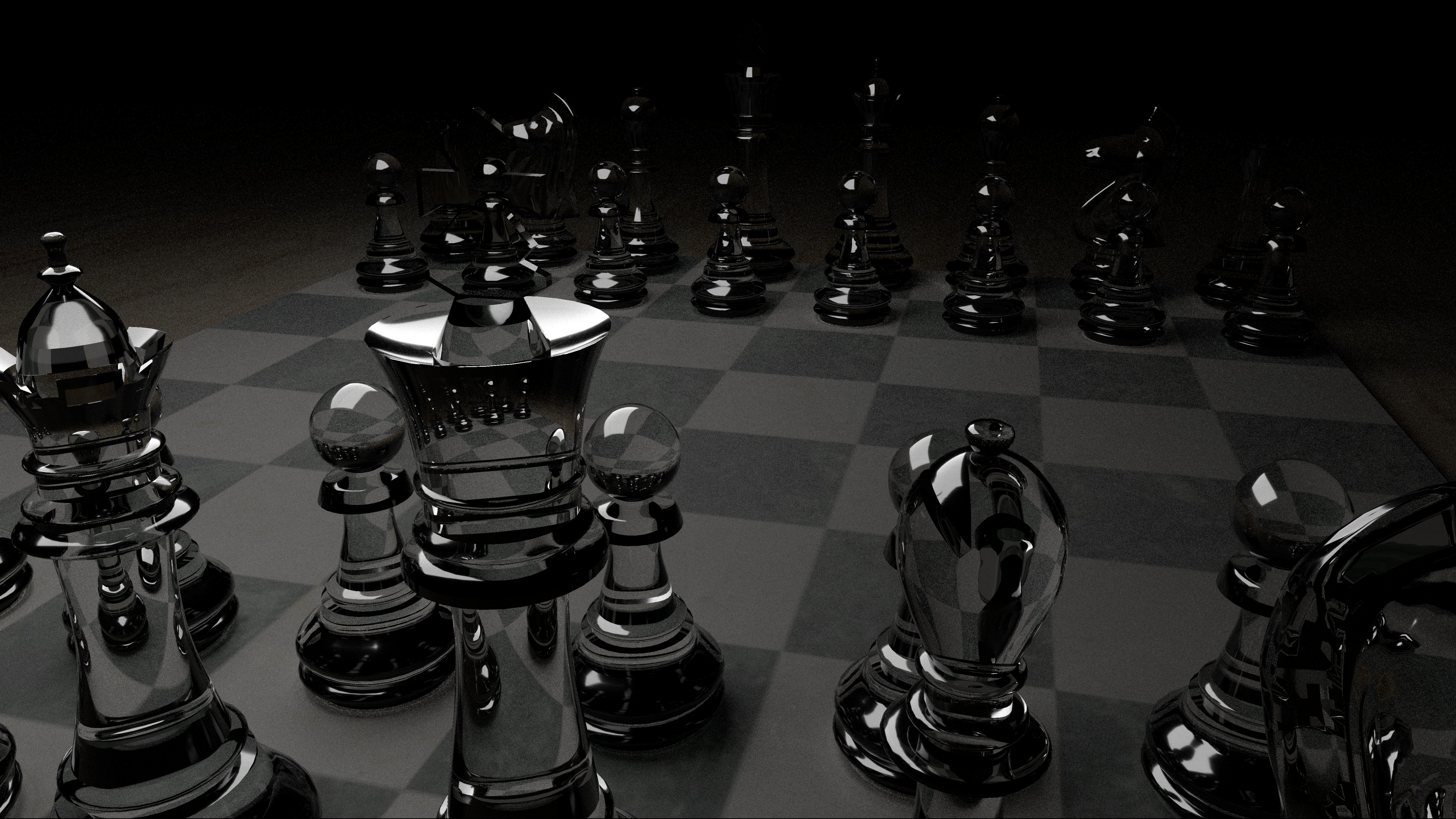
Buffers (MTLPipelineBufferDescriptorArray) 31

ComputePipeline Performance 1.08 ms (1.0%)

Texture 0 (MTLTexture) 0x100b1fcd0 - 2048 x 1152, RGBA16Float

Buffer 0 0x100b27090, Offset=0x00007600

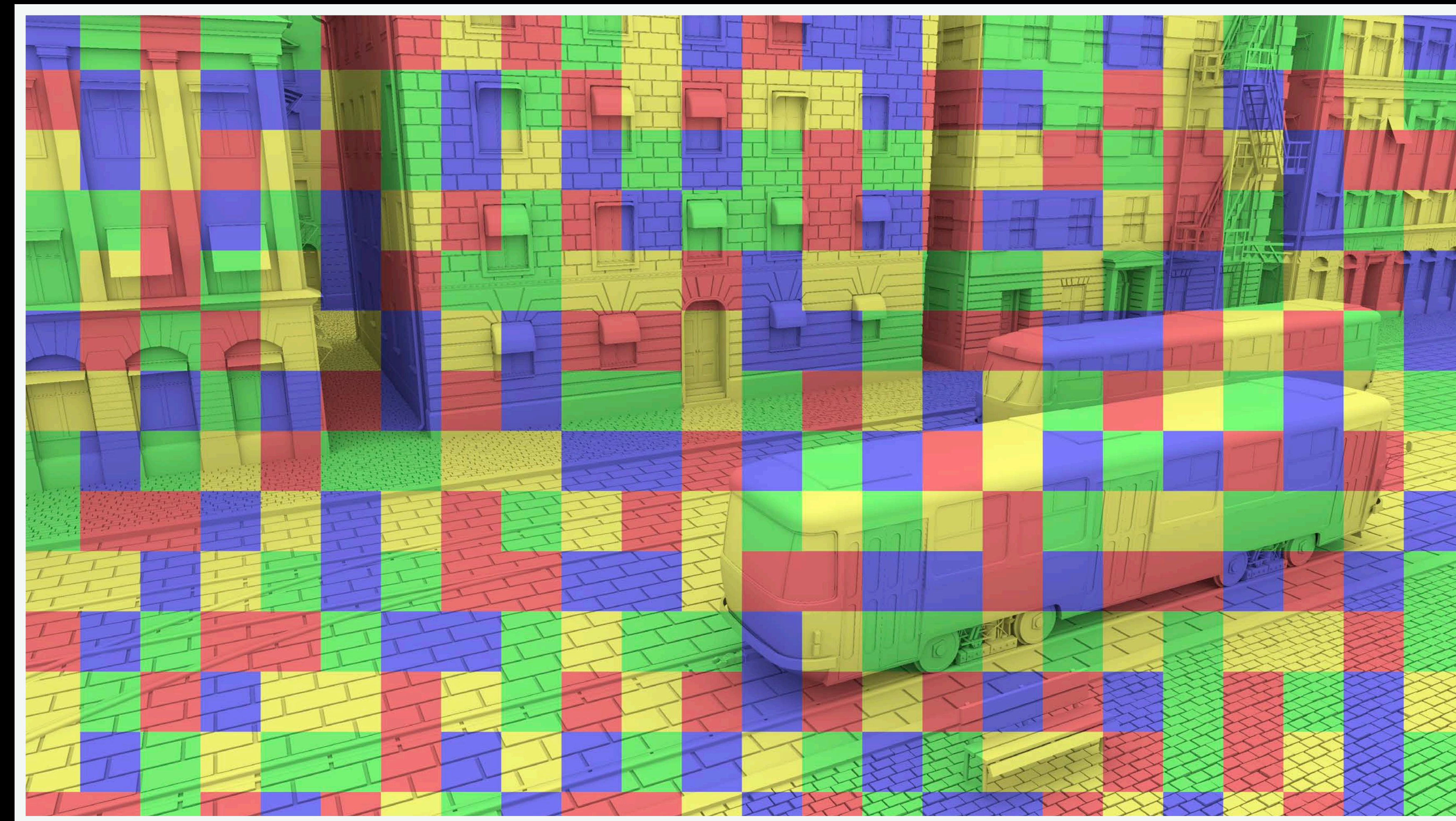
Auto | Filter



Demo

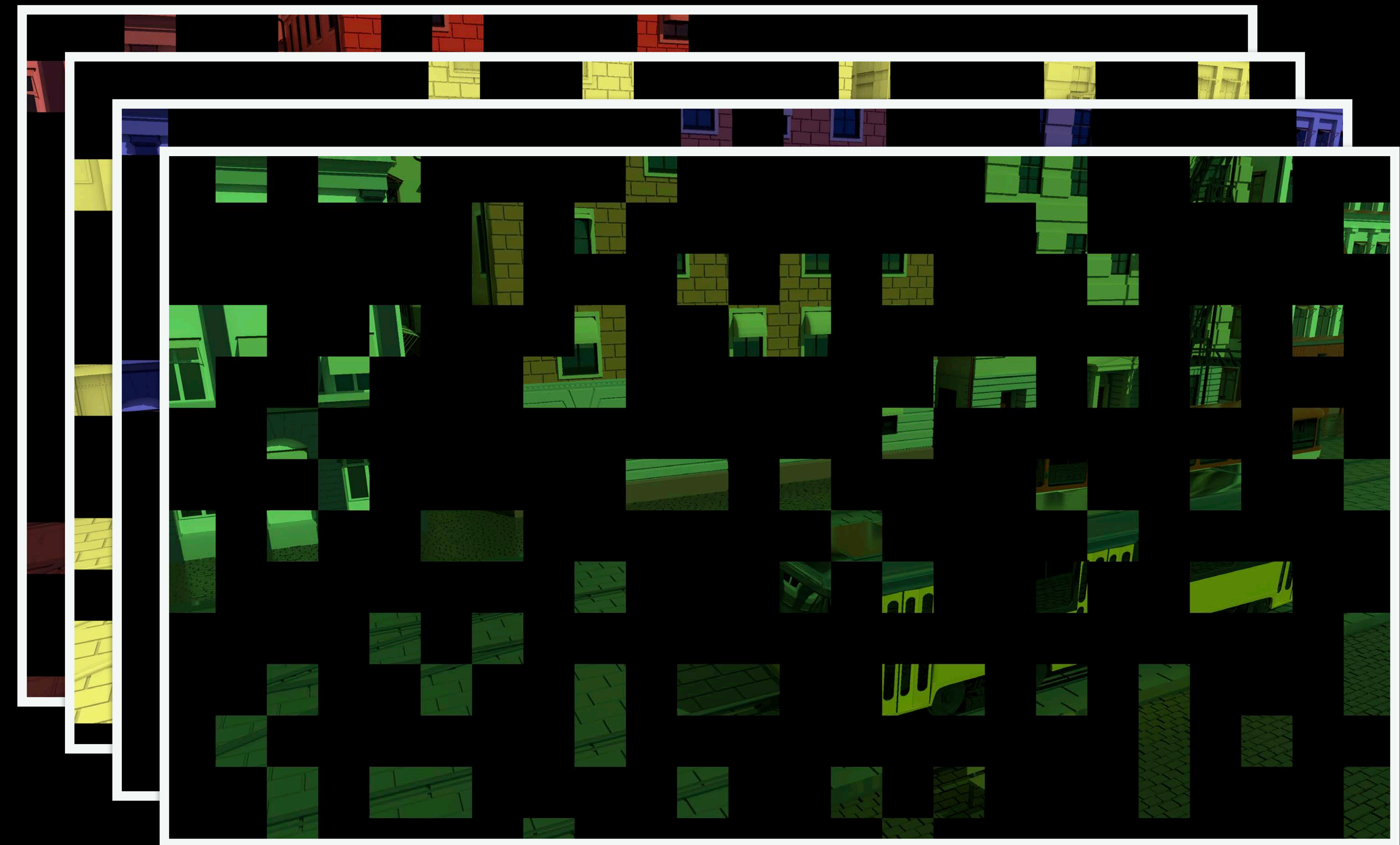
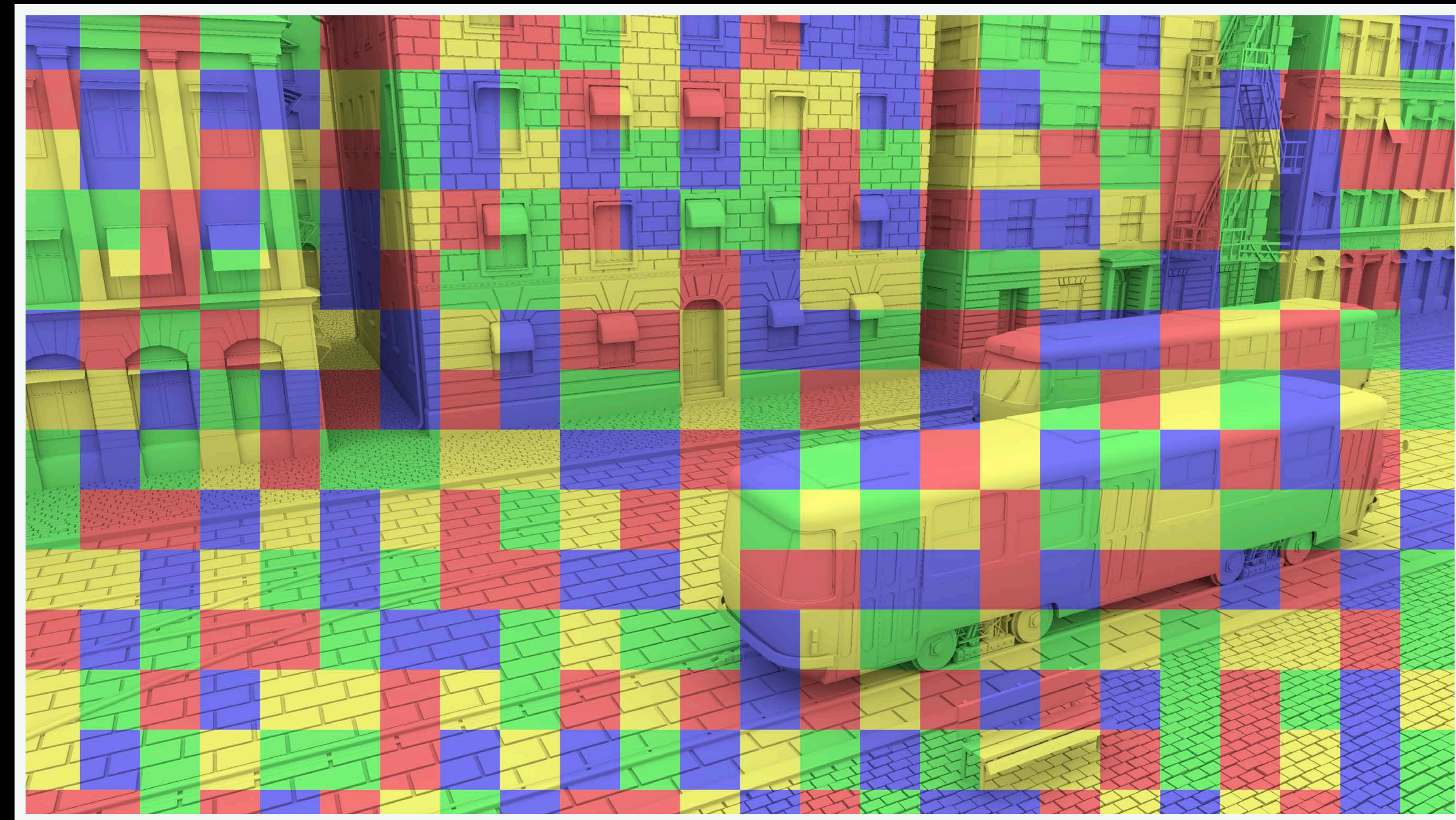
Interleaved Tiling

Divide screen into tiles, render on different GPUs



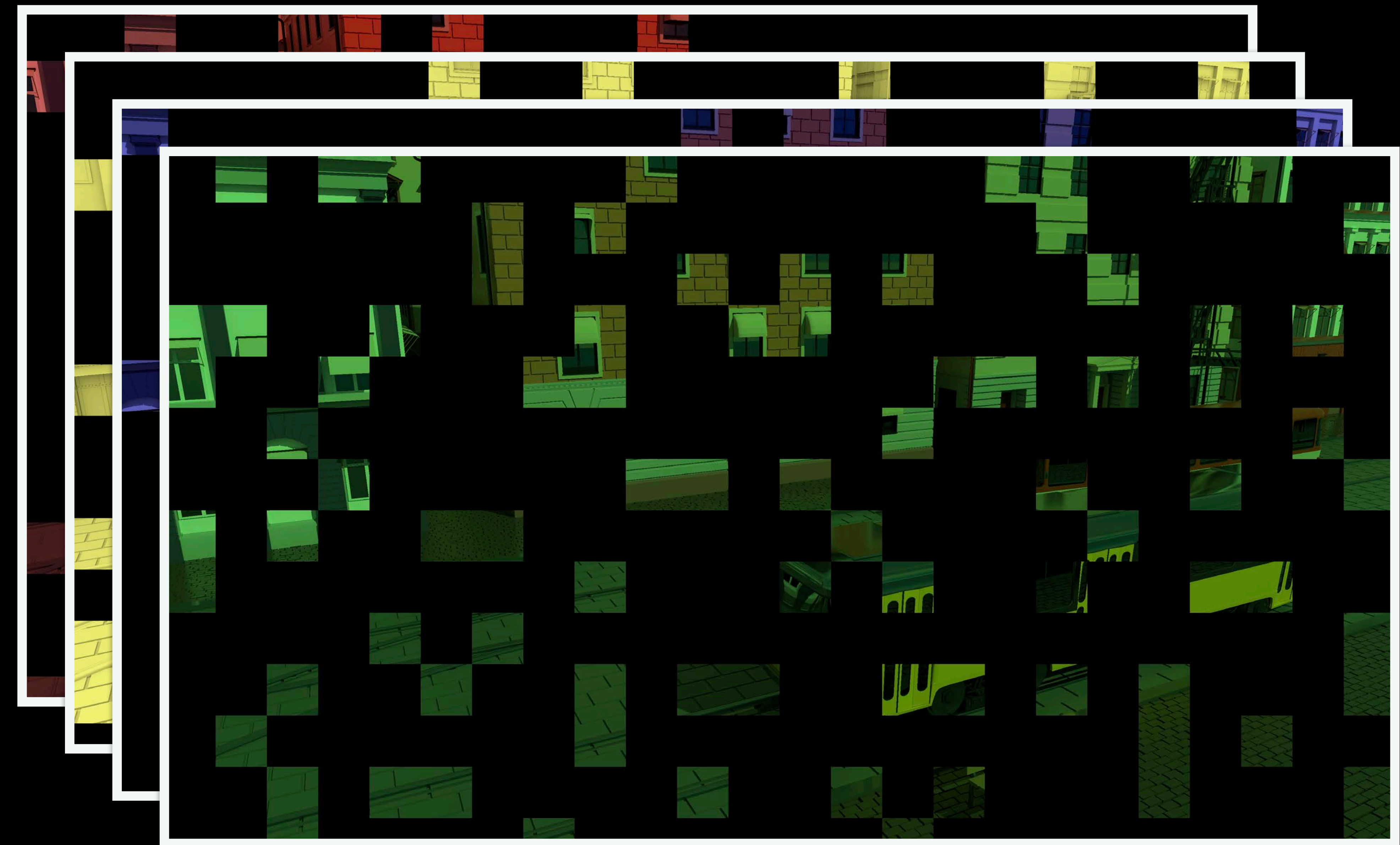
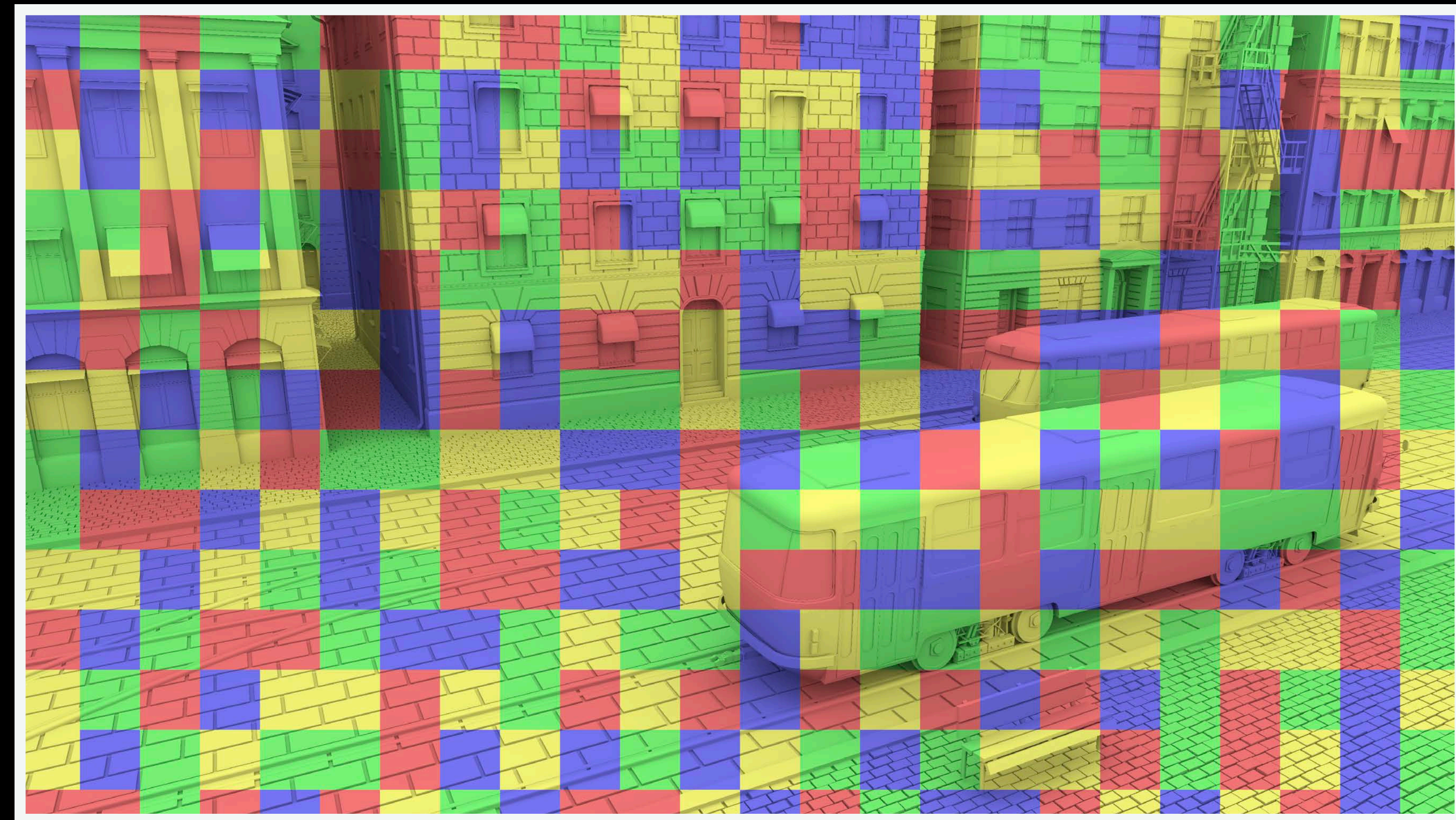
Interleaved Tiling

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

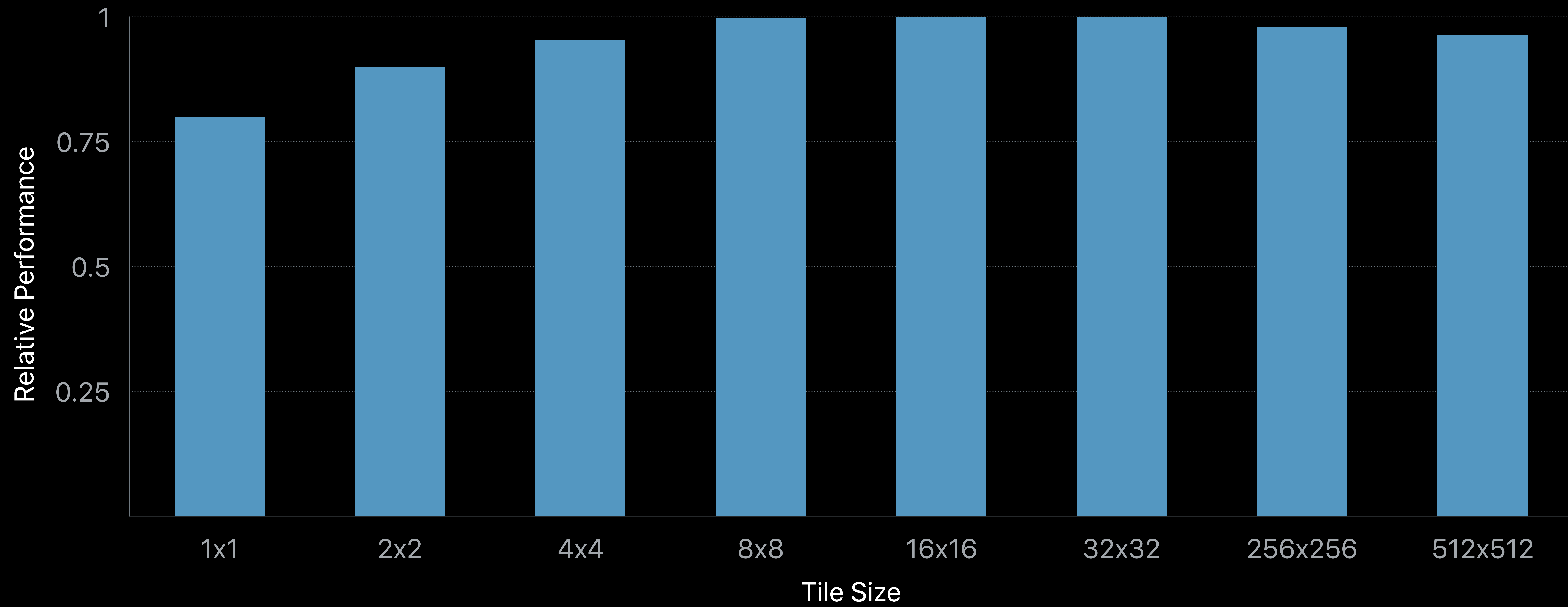


Choosing a Tile Size

How small should I make my tiles?

Experiment: vary tile size, measure performance

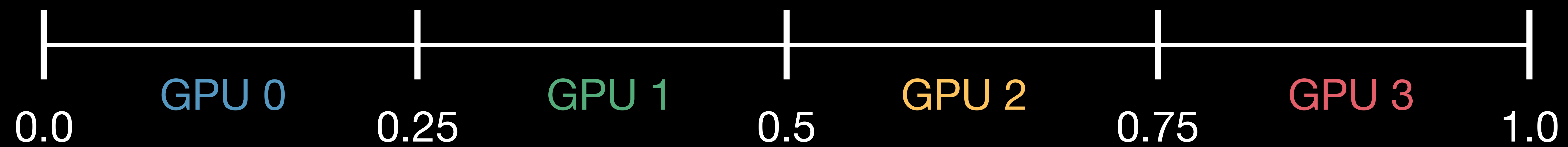
Choosing a Tile Size



Tile Assignment

Assign each tile a random number

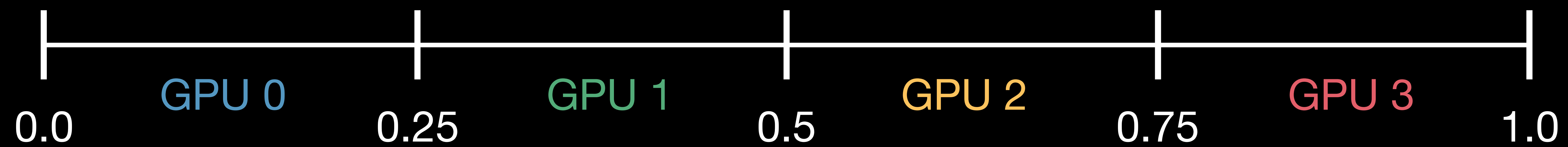
Compare against thresholds to pick GPU



Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

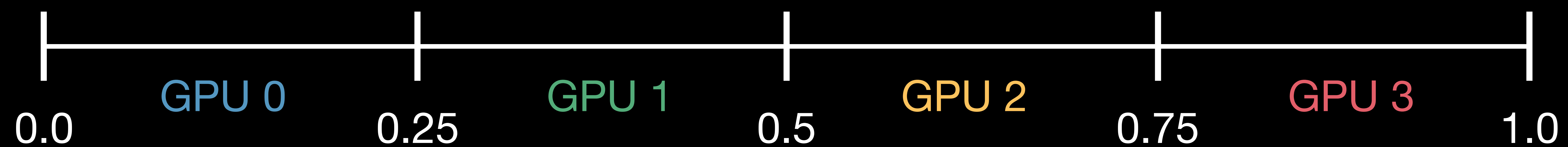


Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

0.4	0.55	0.22	0.78
0.64	0.12	0.35	0.89
0.1	0.39	0.72	0.61
0.79	0.2	0.9	0.42

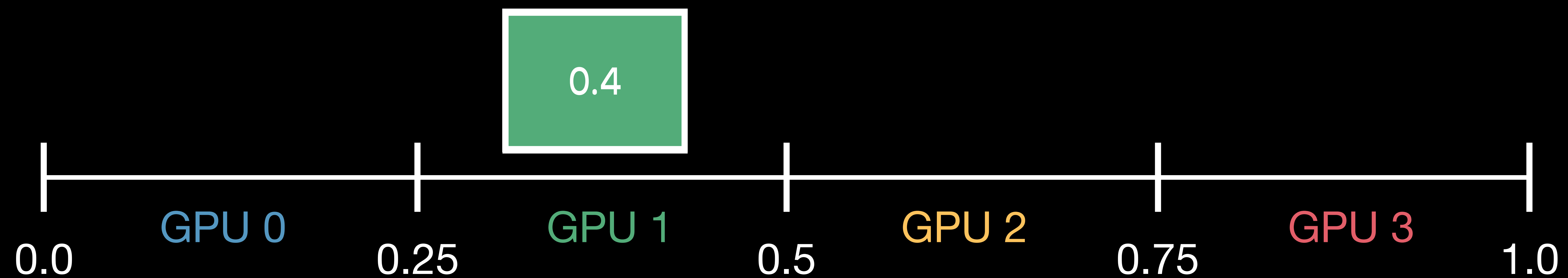


Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

0.4	0.55	0.22	0.78
0.64	0.12	0.35	0.89
0.1	0.39	0.72	0.61
0.79	0.2	0.9	0.42

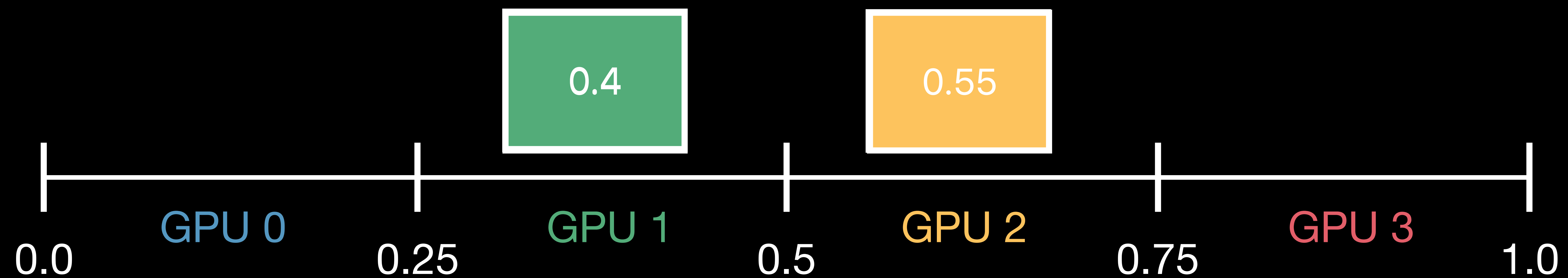


Tile Assignment

Assign each tile a random number

Compare against thresholds to pick GPU

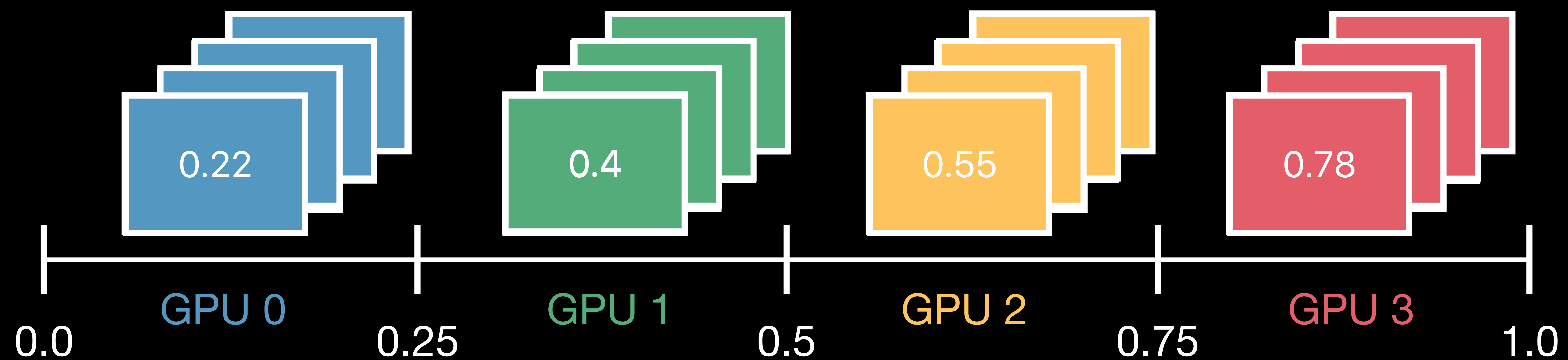
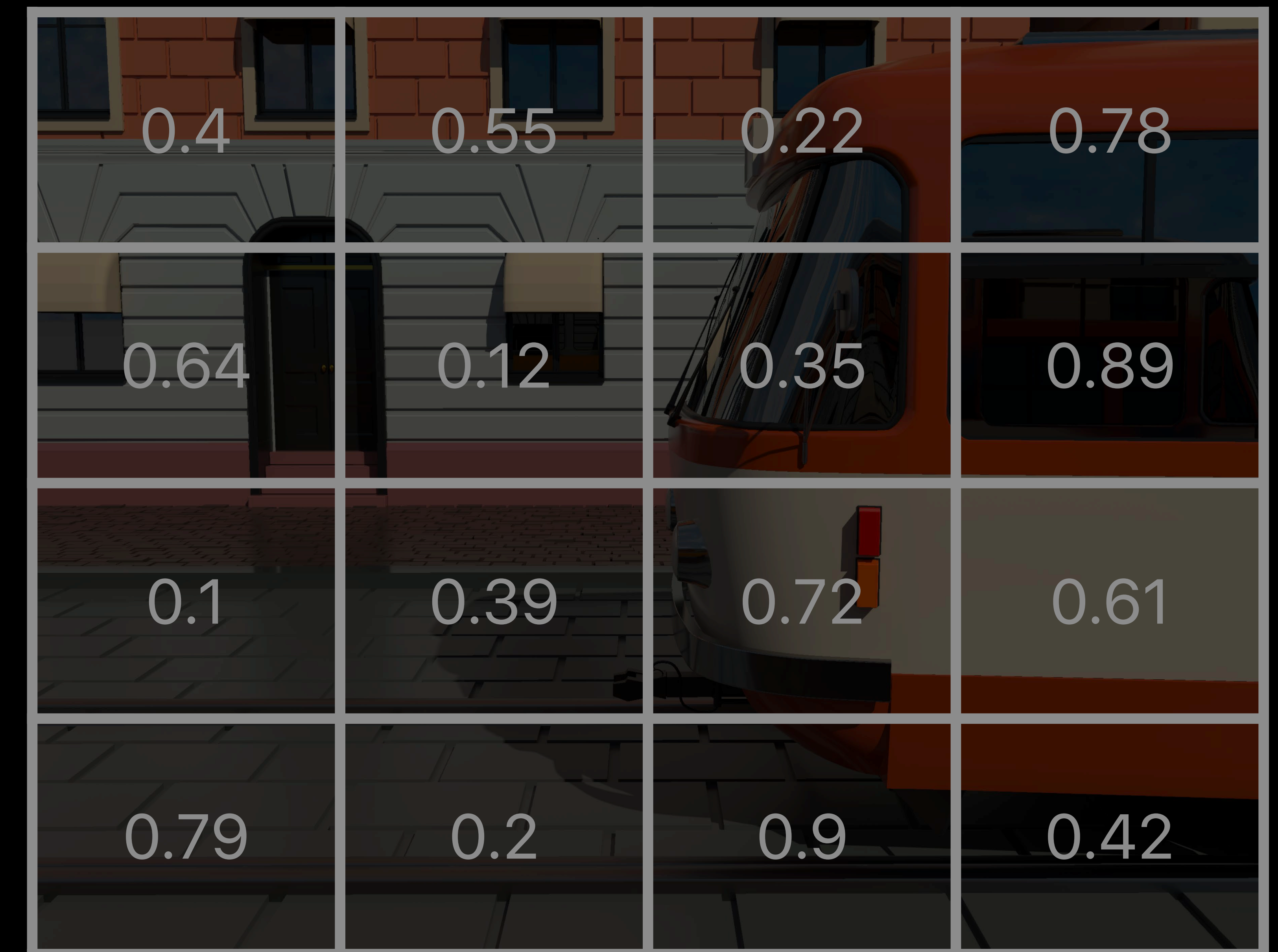
0.4	0.55	0.22	0.78
0.64	0.12	0.35	0.89
0.1	0.39	0.72	0.61
0.79	0.2	0.9	0.42



Tile Assignment

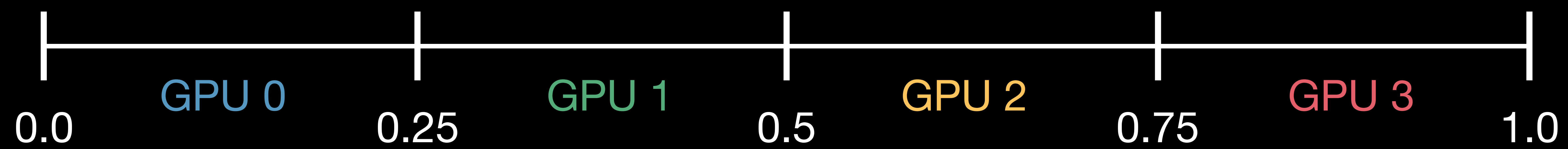
Assign each tile a random number

Compare against thresholds to pick GPU



Tile Assignment

Adjust ranges to control distribution



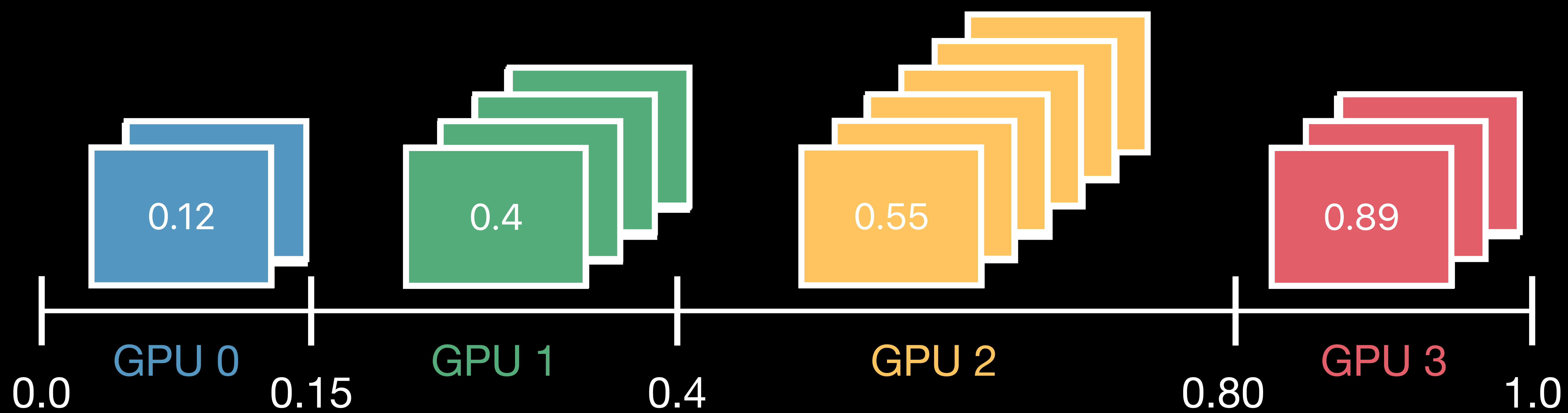
Tile Assignment

Adjust ranges to control distribution

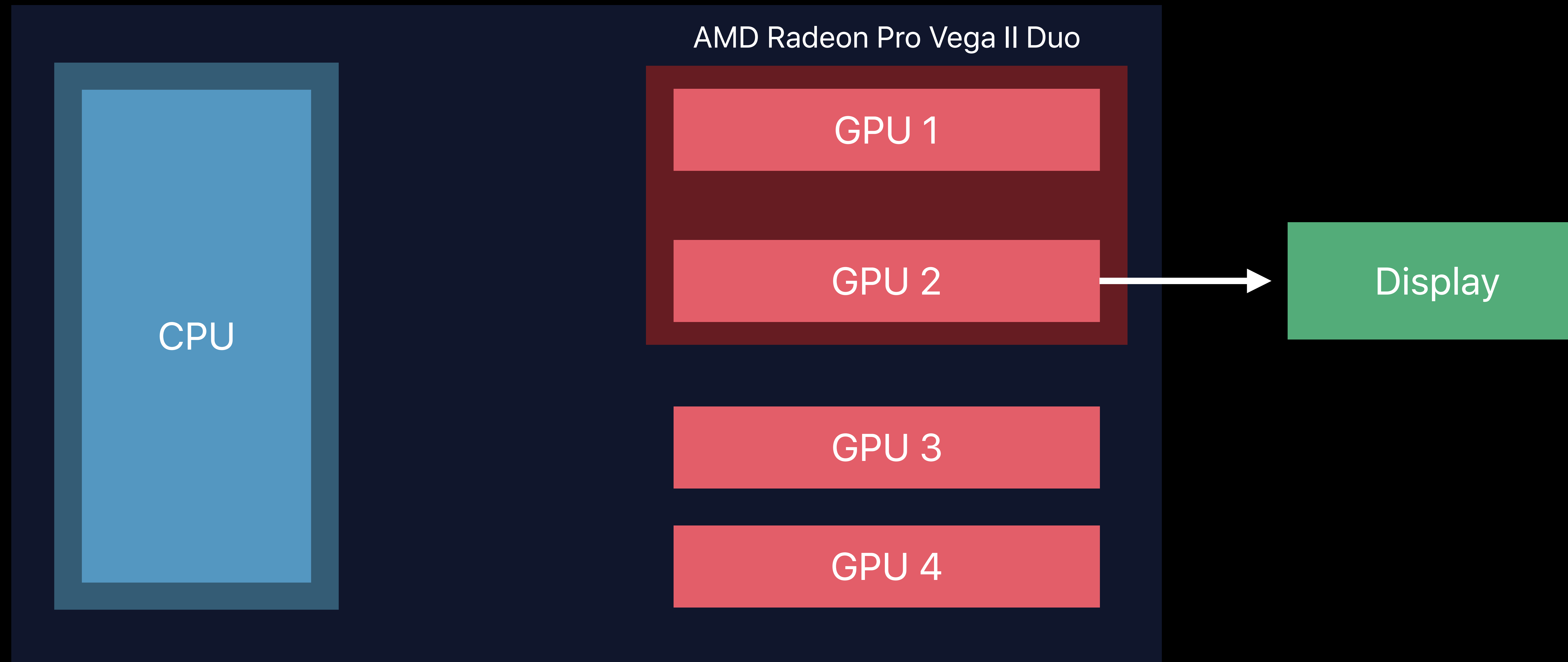


Tile Assignment

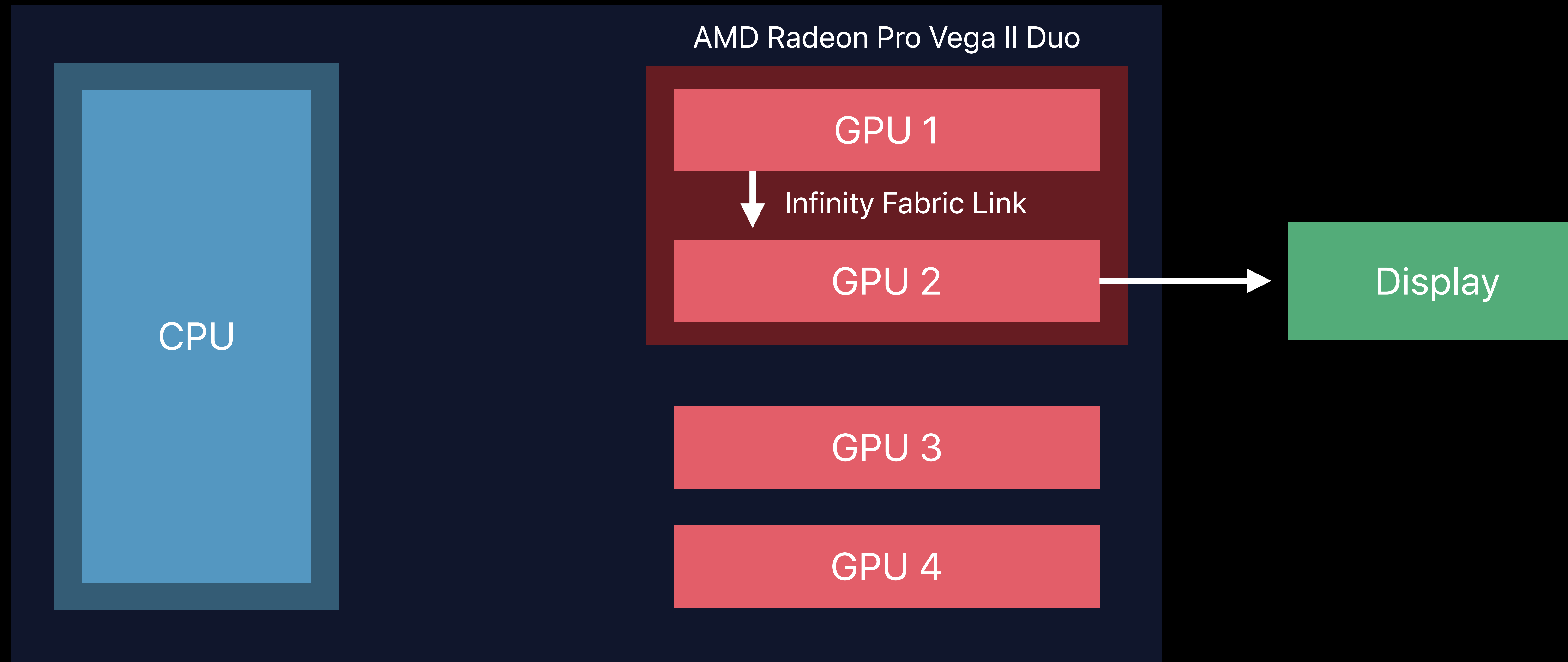
Adjust ranges to control distribution



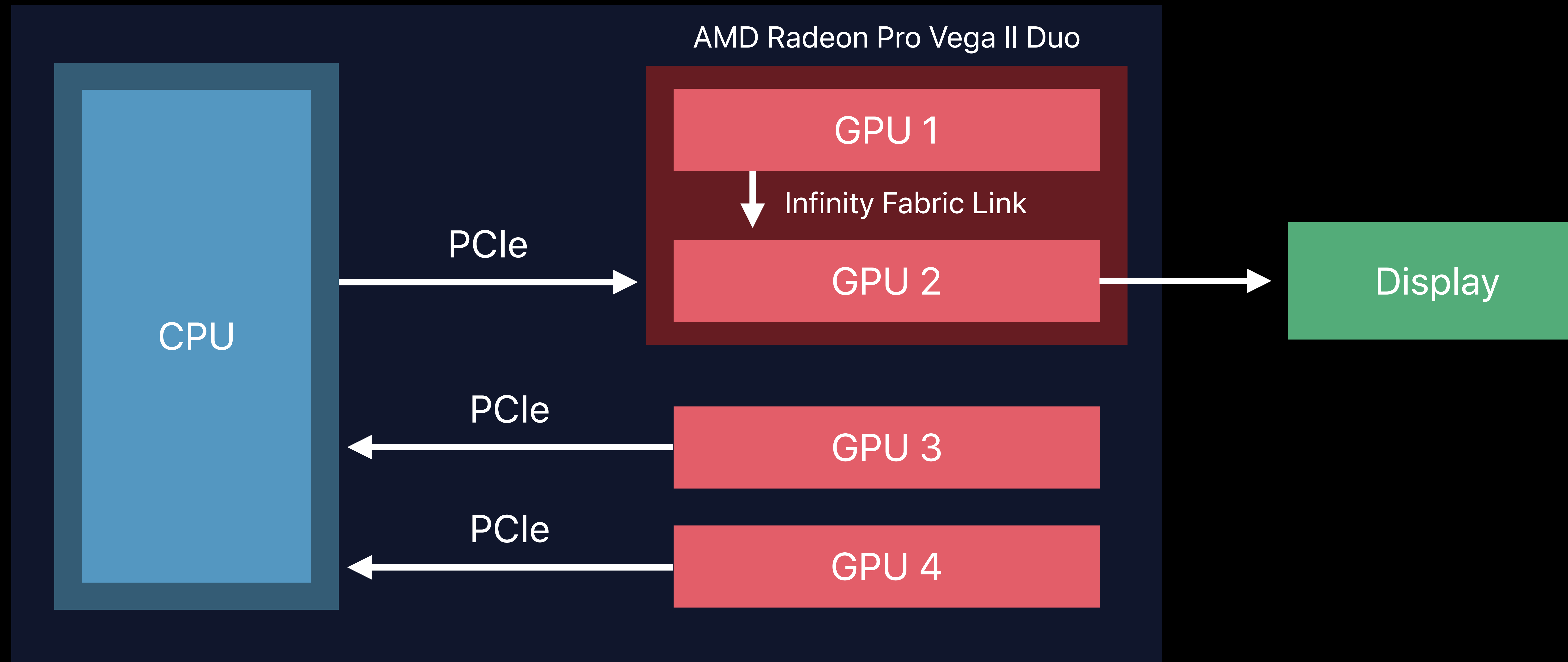
Data Transfers



Data Transfers



Data Transfers



Data Transfers

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 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



Data Transfers

GPU 0

Render tiles for frame 0

Render tiles for frame 1

Render tiles for frame 2

Copy tiles
for frame 0

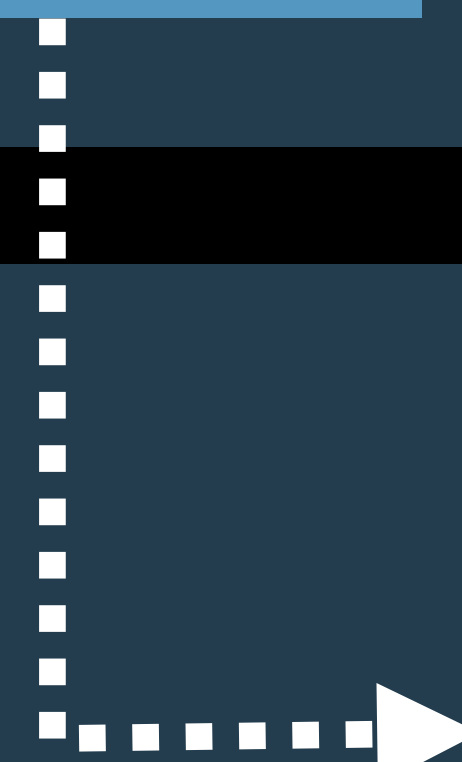
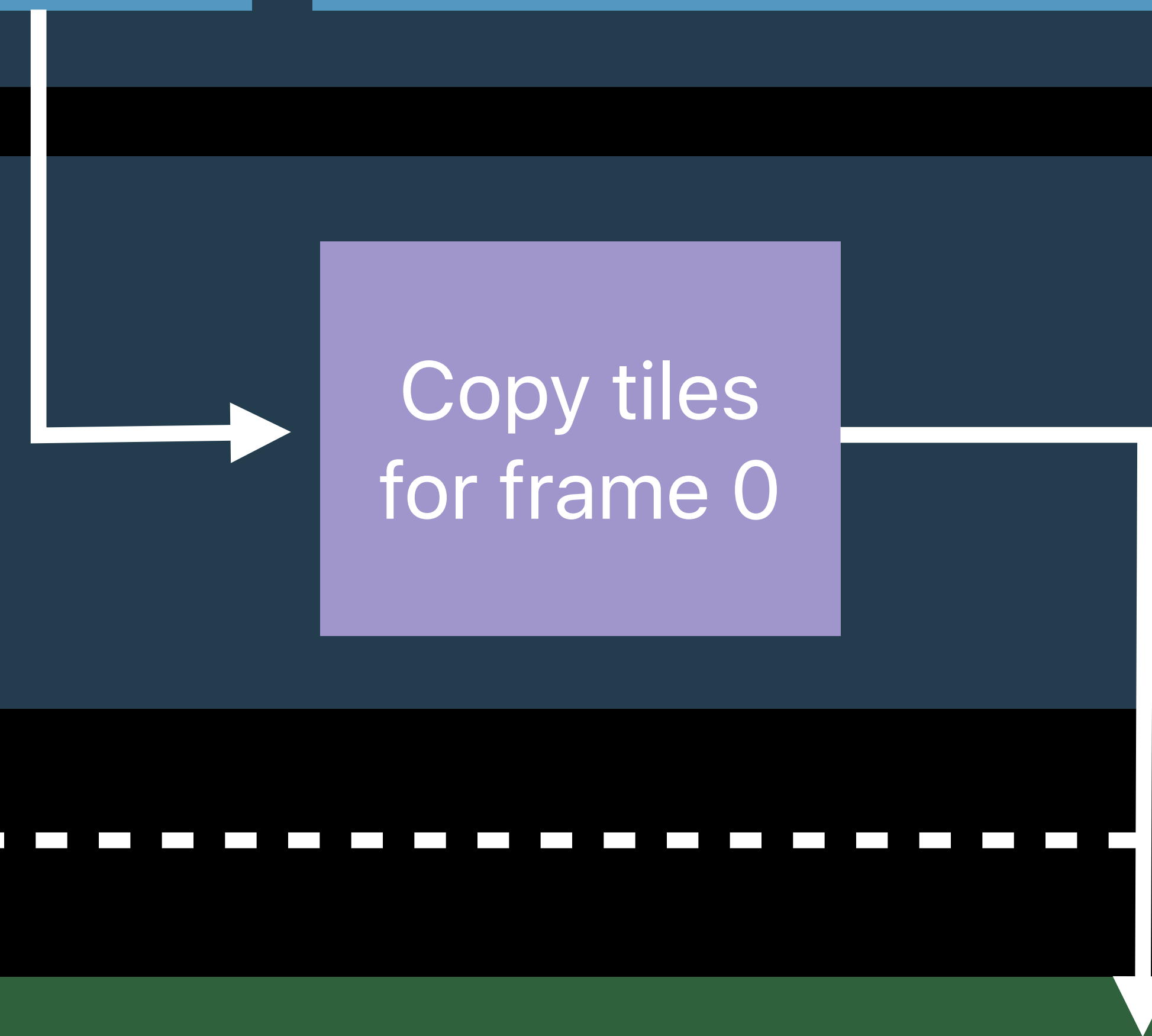
Copy tiles
for frame 1

GPU 1

Render tiles for frame 0

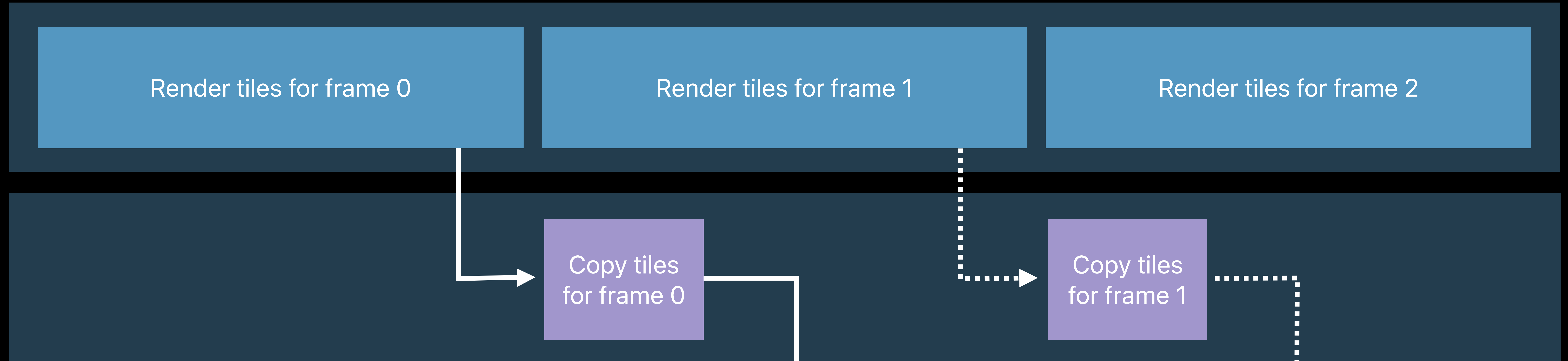
Render tiles for frame 1

Composite
frame 0

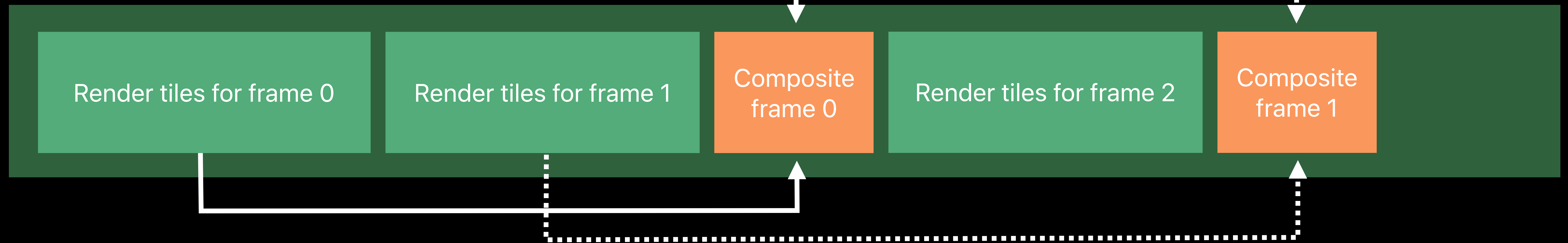


Data Transfers

GPU 0



GPU 1



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

