

What's New in Metal, Part 2

Session 607

Dan Omachi GPU Software Frameworks Engineer

Anna Tikhonova GPU Software Frameworks Engineer

Metal at WWDC

What's New in Metal, Part 1

- Metal in Review
- New Features
- Metal and App Thinning

What's New in Metal, Part 2

- Introducing
- Metal
Shaders

Metal Performance Optimization Techniques

- Metal System Trace T
- Metal Best P

Metal at WWDC

What's New in Metal, Part 1

- Metal in Review
- New Features
- Metal and App T

What's New in Metal, Part 2

- Introducing MetalKit
- Metal Performance Shaders

Metal Performance Optimization Techniques

- Metal System Trace T
- Metal Best P

Metal at WWDC

What's New in Metal, Part 1

- Metal in Review
- New Features
- Metal and App T

What's New in Metal, Part 2

- Introducing
- Metal
Shaders

Metal Performance Optimization Techniques

- Metal System Trace Tool
- Metal Best Practices

MetalKit

Utility functionality for Metal Apps

MetalKit

NEW

MetalKit

NEW

MetalKit provides efficient implementations for commonly used scenarios

- Less effort to get up and rendering
- Increased performance and stability

MetalKit

Overview

MetalKit

Overview

MetalKit View

- Unified view class for rendering Metal scenes

MetalKit

Overview

MetalKit View

- Unified view class for rendering Metal scenes

Texture Loader

- Metal texture object creation from image files

MetalKit

Overview

MetalKit View

- Unified view class for rendering Metal scenes

Texture Loader

- Metal texture object creation from image files

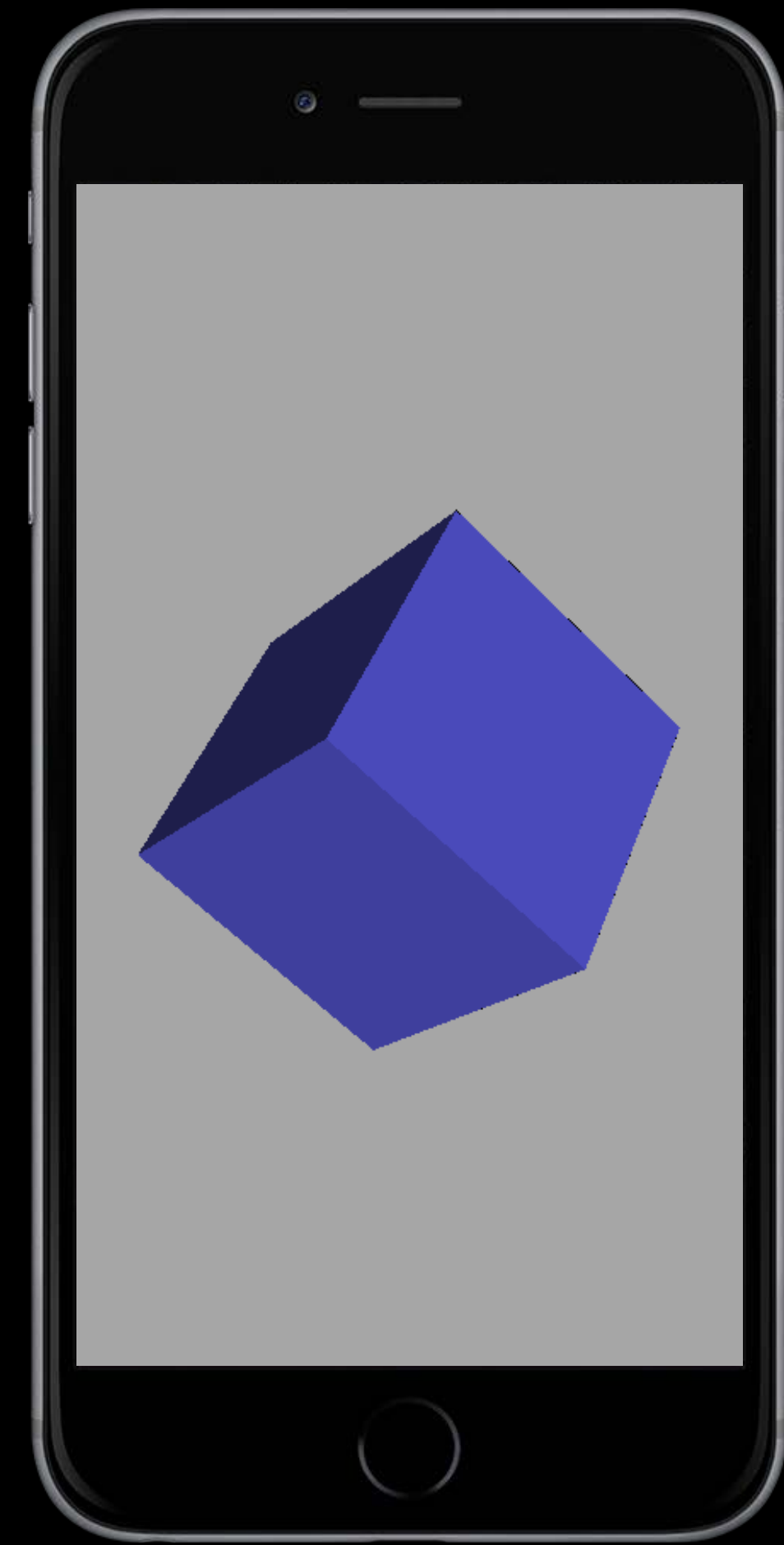
Model I/O Integration

- Load and manage mesh data for Metal rendering

MetalKit View

Overview

Simplest way to get Metal rendering on screen



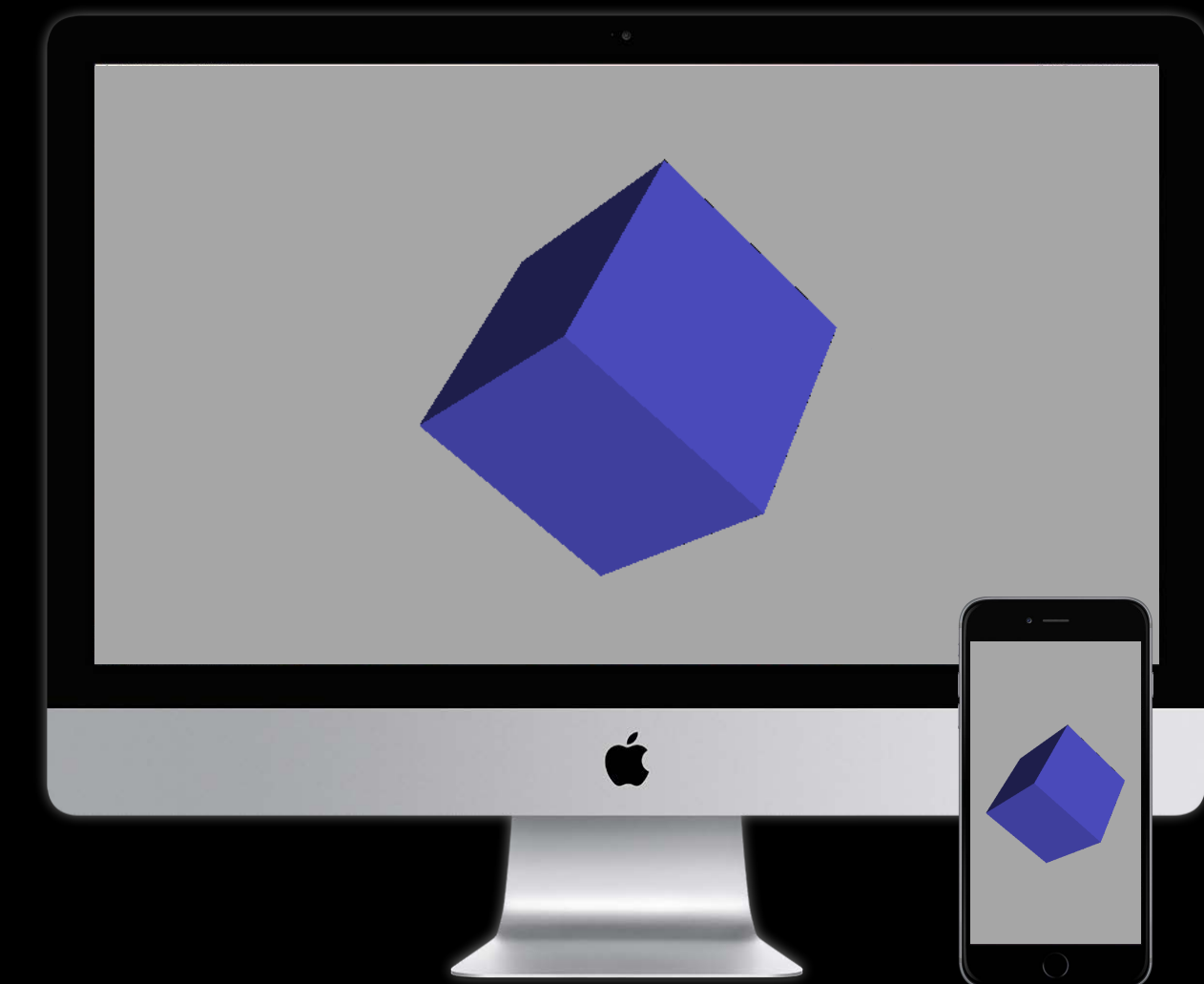
MetalKit View

Overview

Simplest way to get Metal rendering on screen

Unified between iOS and OS X

- Subclass of UIView for iOS
- Subclass of NSView for OS X



MetalKit View

Overview

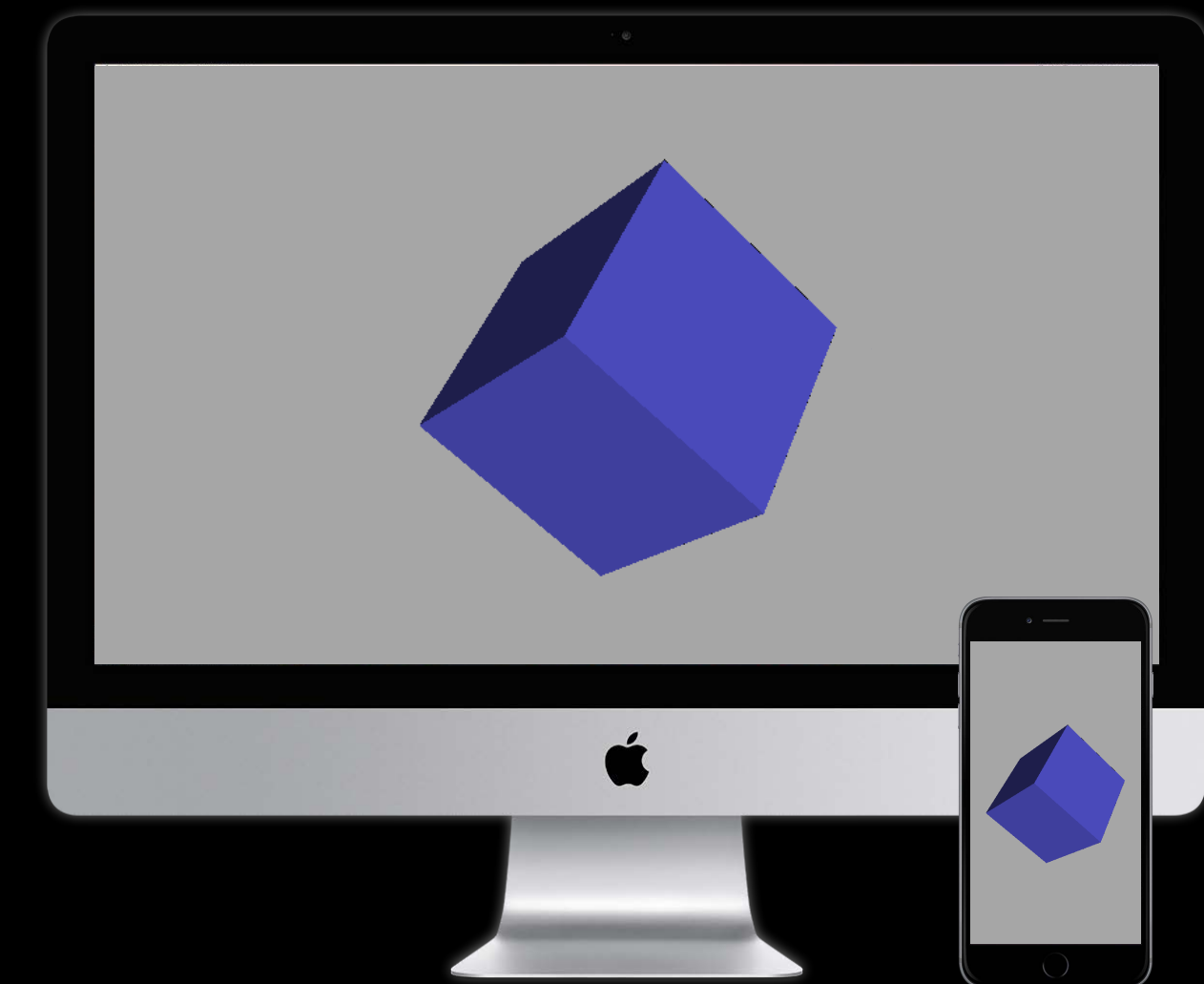
Simplest way to get Metal rendering on screen

Unified between iOS and OS X

- Subclass of UIView for iOS
- Subclass of NSView for OS X

Manages render targets

- Creates render pass descriptors



MetalKit View

Overview

Simplest way to get Metal rendering on screen

Unified between iOS and OS X

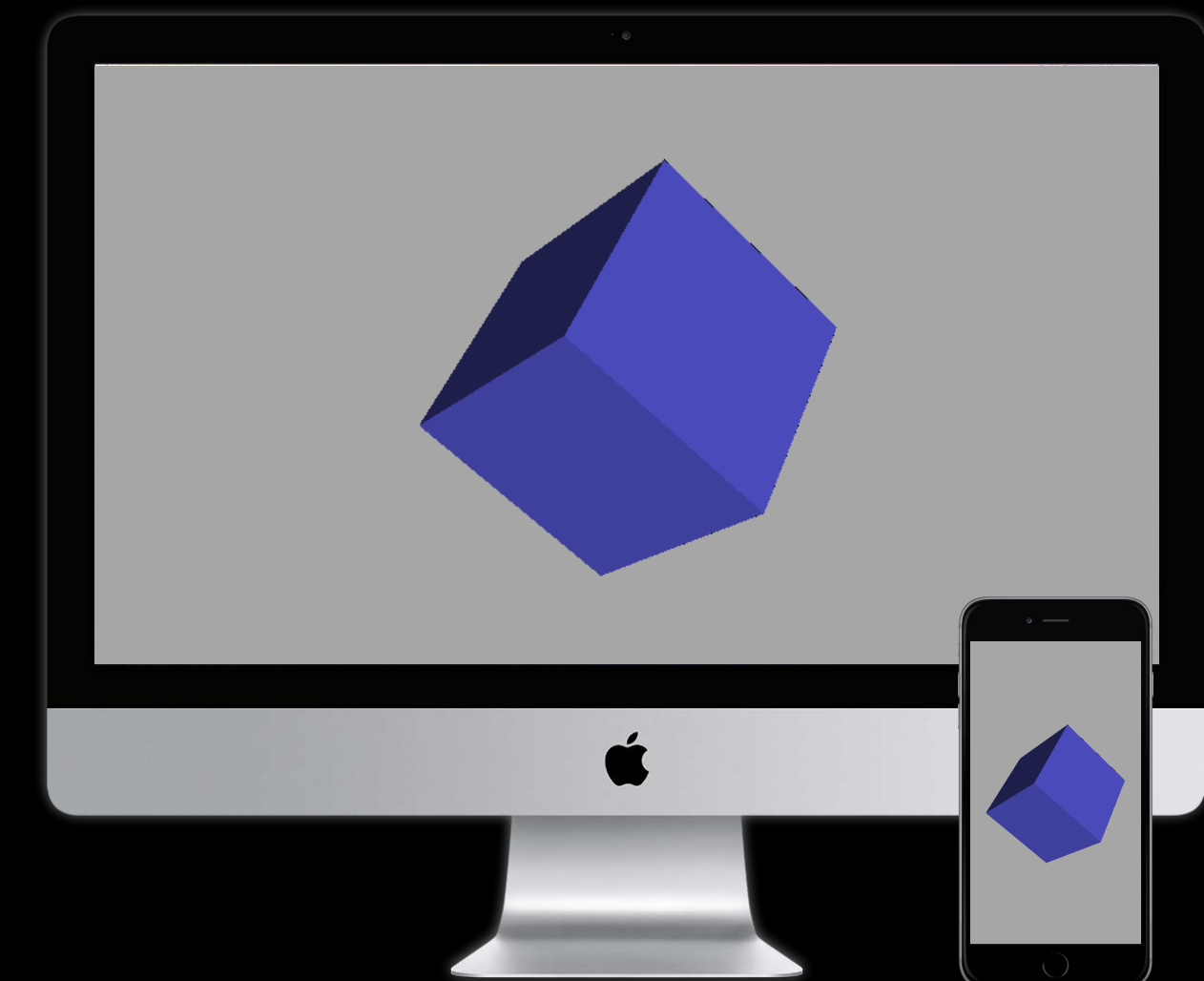
- Subclass of UIView for iOS
- Subclass of NSView for OS X

Manages render targets

- Creates render pass descriptors

Multiple draw loop modes supported

- Timer, event, or explicitly driven draw loop



MetalKit View Setup

Approaches to using MTKView

MetalKit View Setup

Approaches to using MTKView

A. Implement a delegate

- `(void)drawInView:(MTKView *)view`
- `(void)view:(MTKView *)view willLayoutWithSize:(CGSize)size`

MetalKit View Setup

Approaches to using MTKView

A. Implement a delegate

- `(void)drawInView:(MTKView *)view`
- `(void)view:(MTKView *)view willLayoutWithSize:(CGSize)size`

MetalKit View Setup

Approaches to using MTKView

A. Implement a delegate

- `(void)drawInView:(MTKView *)view`
- `(void)view:(MTKView *)view willLayoutWithSize:(CGSize)size`

B. Subclass MTKView

- iOS
 - `(void)drawRect:(CGRect)rect`
 - `(void)layoutSubviews`
- OS X
 - `(void)drawRect:(CGRect)rect`
 - `(void)setFrameSize:(NSSize)newSize`

MetalKit View Setup

Approaches to using MTKView

A. Implement a delegate

- `(void)drawInView:(MTKView *)view`
- `(void)view:(MTKView *)view willLayoutWithSize:(CGSize)size`

B. Subclass MTKView

- iOS

- `(void)drawRect:(CGRect)rect`
- `(void)layoutSubviews`

- OS X

- `(void)drawRect:(CGRect)rect`
- `(void)setFrameSize:(NSSize)newSize`

MetalKit View Setup

Approaches to using MTKView

A. Implement a delegate

- `(void)drawInView:(MTKView *)view`
- `(void)view:(MTKView *)view willLayoutWithSize:(CGSize)size`

B. Subclass MTKView

- iOS

- `(void)drawRect:(CGRect)rect`
- `(void)layoutSubviews`

- OS X

- `(void)drawRect:(CGRect)rect`
- `(void)setFrameSize:(NSSize)newSize`

MetalKit View Setup

Initializing view properties

```
- (void)viewDidLoad
{
    MTKView *view = (MTKView *)self.view;

    view.delegate = self;

    view.device = device;

    view.colorPixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
    view.depthStencilPixelFormat = MTLPixelFormatDepth32Float_Stencil8;
    view.sampleCount = 4;
    view.clearColor = MTLClearColorMake(0.8, 0.8, 0.8, 1.0);
}
```

MetalKit View Setup

Initializing view properties

```
- (void)viewDidLoad
{
    MTKView *view = (MTKView *)self.view;

    view.delegate = self;

    view.device = device;

    view.colorPixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
    view.depthStencilPixelFormat = MTLPixelFormatDepth32Float_Stencil8;
    view.sampleCount = 4;
    view.clearColor = MTLClearColorMake(0.8, 0.8, 0.8, 1.0);
}
```

MetalKit View Setup

Initializing view properties

```
- (void)viewDidLoad
{
    MTKView *view = (MTKView *)self.view;

    view.delegate = self;

    view.device = device;

    view.colorPixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
    view.depthStencilPixelFormat = MTLPixelFormatDepth32Float_Stencil8;
    view.sampleCount = 4;
    view.clearColor = MTLClearColorMake(0.8, 0.8, 0.8, 1.0);
}
```


MetalKit View Setup

Initializing view properties

```
- (void)viewDidLoad
{
    MTKView *view = (MTKView *)self.view;

    view.delegate = self;

    view.device = device;

    view.colorPixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
    view.depthStencilPixelFormat = MTLPixelFormatDepth32Float_Stencil8;
    view.sampleCount = 4;
    view.clearColor = MTLClearColorMake(0.8, 0.8, 0.8, 1.0);
}
```

MetalKit View Setup

Initializing view properties

```
- (void)viewDidLoad
{
    MTKView *view = (MTKView *)self.view;

    view.delegate = self;

    view.device = device;

    view.colorPixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
    view.depthStencilPixelFormat = MTLPixelFormatDepth32Float_Stencil8;
    view.sampleCount = 4;
    view.clearColor = MTLClearColorMake(0.8, 0.8, 0.8, 1.0);
}
```

MetalKit View Setup

Initializing view properties

```
- (void)viewDidLoad
{
    MTKView *view = (MTKView *)self.view;

    view.delegate = self;

    view.device = device;

    view.colorPixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
    view.depthStencilPixelFormat = MTLPixelFormatDepth32Float_Stencil8;
    view.sampleCount = 4;
    view.clearColor = MTLClearColorMake(0.8, 0.8, 0.8, 1.0);
}
```

MetalKit View Setup

Initializing view properties

```
- (void)viewDidLoad
{
    MTKView *view = (MTKView *)self.view;

    view.delegate = self;

    view.device = device;

    view.colorPixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
    view.depthStencilPixelFormat = MTLPixelFormatDepth32Float_Stencil8;
    view.sampleCount = 4;
    view.clearColor = MTLClearColorMake(0.8, 0.8, 0.8, 1.0);
}
```

MetalKit View Drawing

Simple usage

```
- (void)drawInView:(nonnull MTKView *)view {
    id <MTLRenderPassDescriptor> descriptor =
        view.currentRenderPassDescriptor;

    // Create render command encoder and encode final pass
    ...

    [commandBuffer presentDrawable:view.currentDrawable];
    [commandBuffer commit];
}
```

MetalKit View Drawing

Simple usage

```
- (void)drawInView:(nonnull MTKView *)view {  
    id <MTLRenderPassDescriptor> descriptor =  
        view.currentRenderPassDescriptor;  
  
    // Create render command encoder and encode final pass  
    ...  
  
    [commandBuffer presentDrawable:view.currentDrawable];  
    [commandBuffer commit];  
}
```

MetalKit View Drawing

Simple usage

```
- (void)drawInView:(nonnull MTKView *)view {  
    id <MTLRenderPassDescriptor> descriptor =  
        view.currentRenderPassDescriptor;
```

```
    // Create render command encoder and encode final pass  
    ...
```

```
    [commandBuffer presentDrawable:view.currentDrawable];  
    [commandBuffer commit];  
}
```

MetalKit View Drawing

Simple usage

```
- (void)drawInView:(nonnull MTKView *)view {  
    id <MTLRenderPassDescriptor> descriptor =  
        view.currentRenderPassDescriptor;  
  
    // Create render command encoder and encode final pass  
    ...  
  
    [commandBuffer presentDrawable:view.currentDrawable];  
    [commandBuffer commit];  
}
```


Managing Drawables

Managing Drawables

Limited pool of drawables

Managing Drawables

Limited pool of drawables

Drawables concurrently used in many stages of the display pipeline

Managing Drawables

Limited pool of drawables

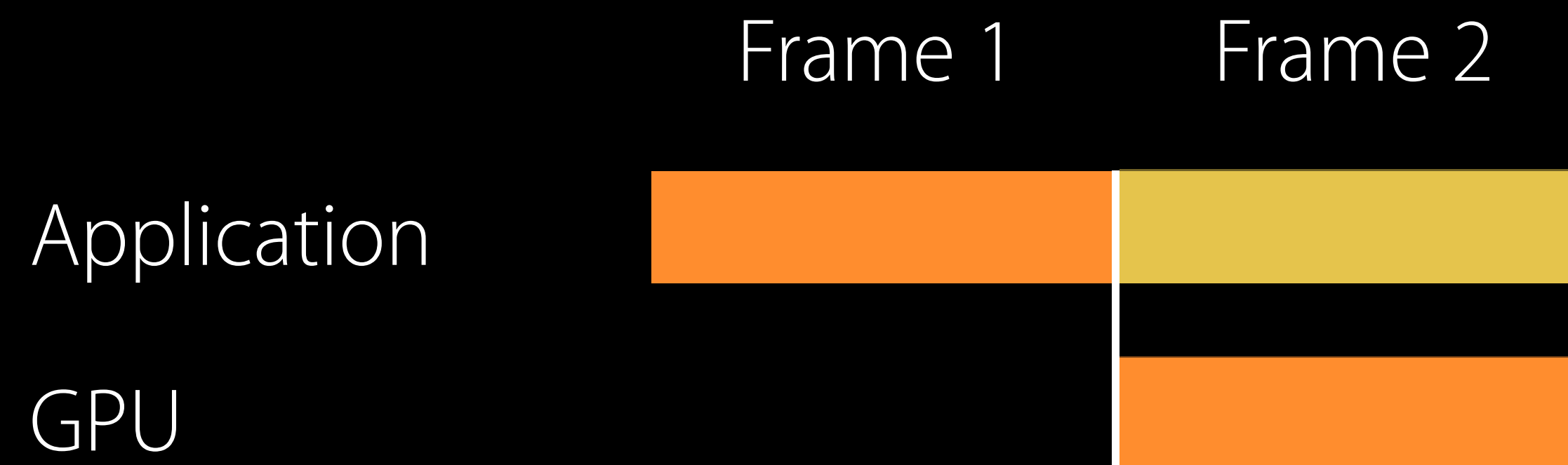
Drawables concurrently used in many stages of the display pipeline



Managing Drawables

Limited pool of drawables

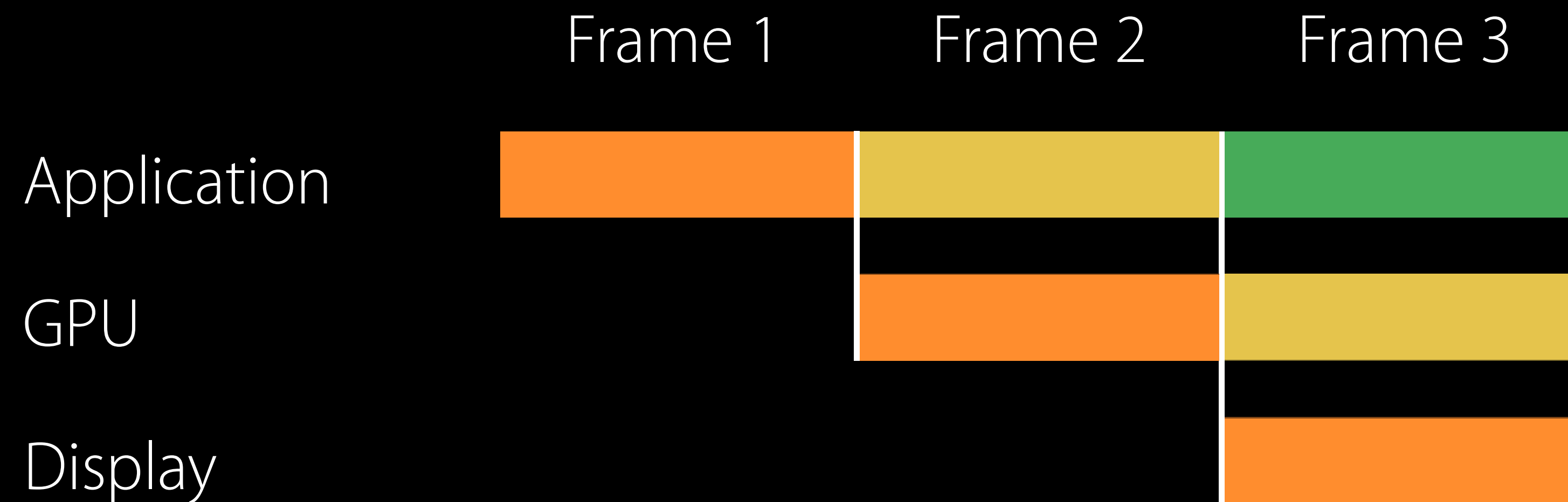
Drawables concurrently used in many stages of the display pipeline



Managing Drawables

Limited pool of drawables

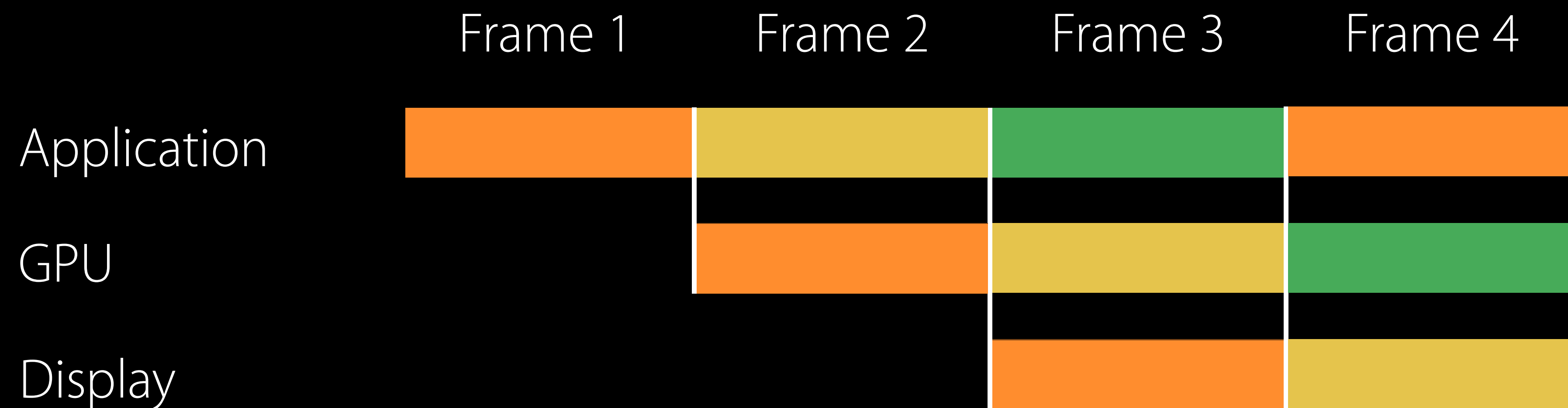
Drawables concurrently used in many stages of the display pipeline



Managing Drawables

Limited pool of drawables

Drawables concurrently used in many stages of the display pipeline



Application Frame

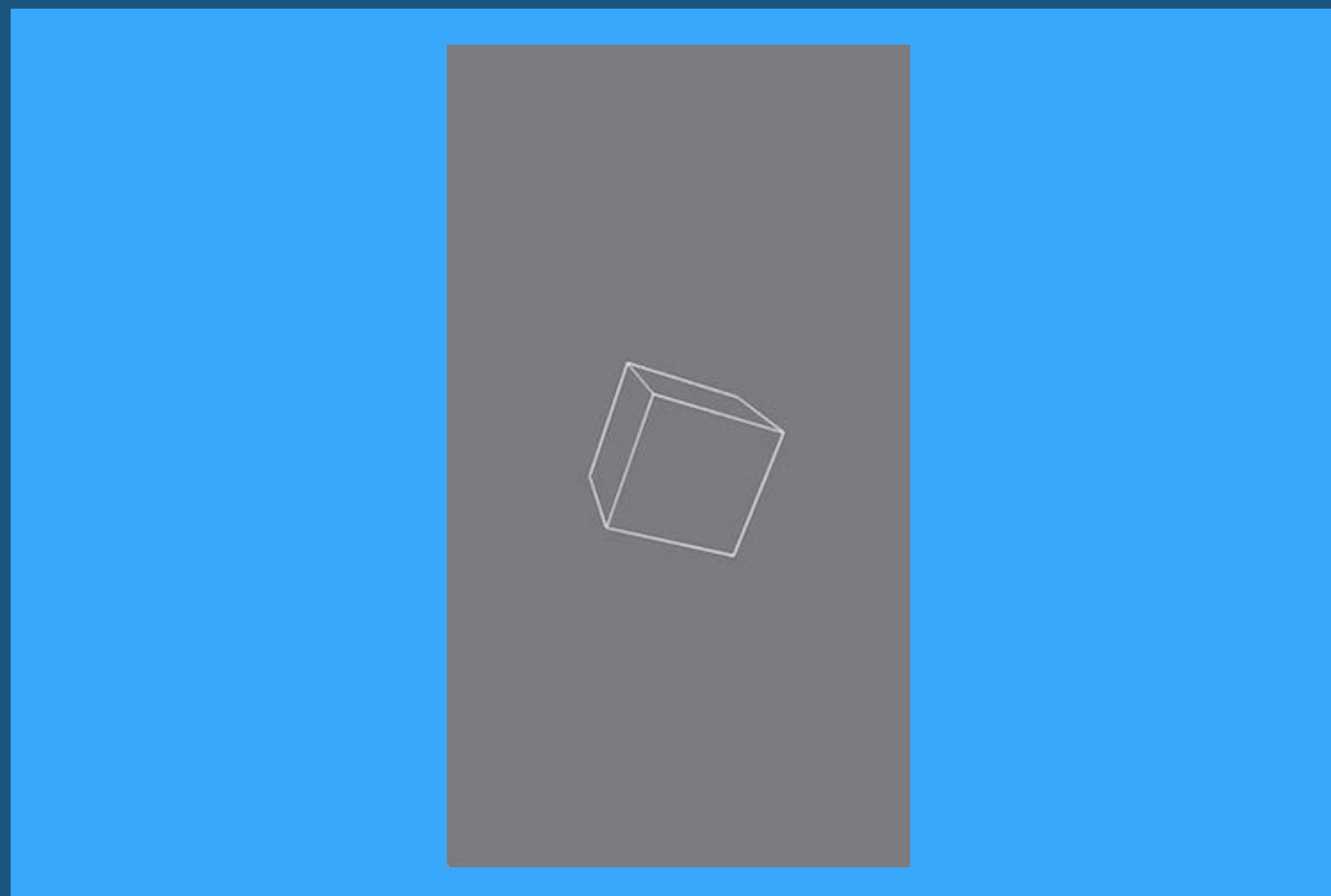
Application Frame

[MTKView currentRenderPassDescriptor]

Reserves a drawable

Application Frame

[MTKView currentRenderPassDescriptor]



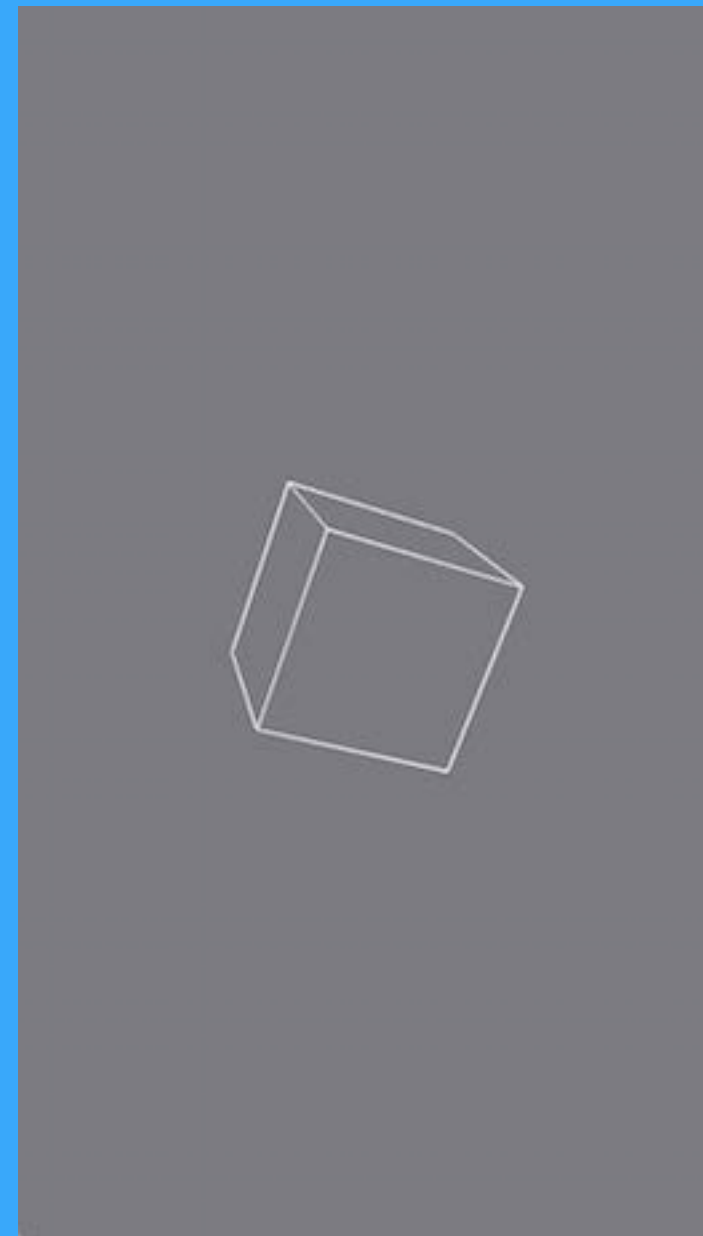
Reserves a drawable

Encodes to the drawable

Application Frame

```
[MTKView currentRenderPassDescriptor]
```

Reserves a drawable



Encodes to the drawable

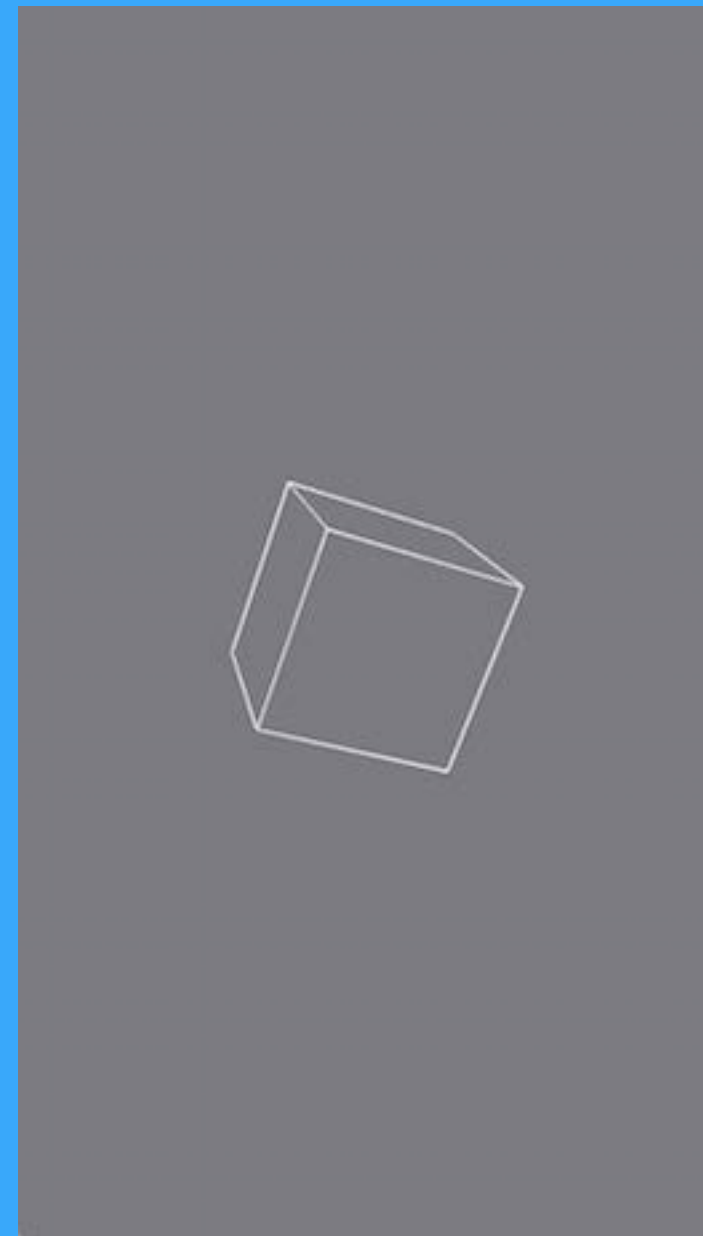
```
[MTLCommandBuffer presentDrawable:]  
[MTLCommandBuffer commit]
```

Releases the drawable

Application Frame

```
[MTKView currentRenderPassDescriptor]
```

Reserves a drawable



Encodes to the drawable

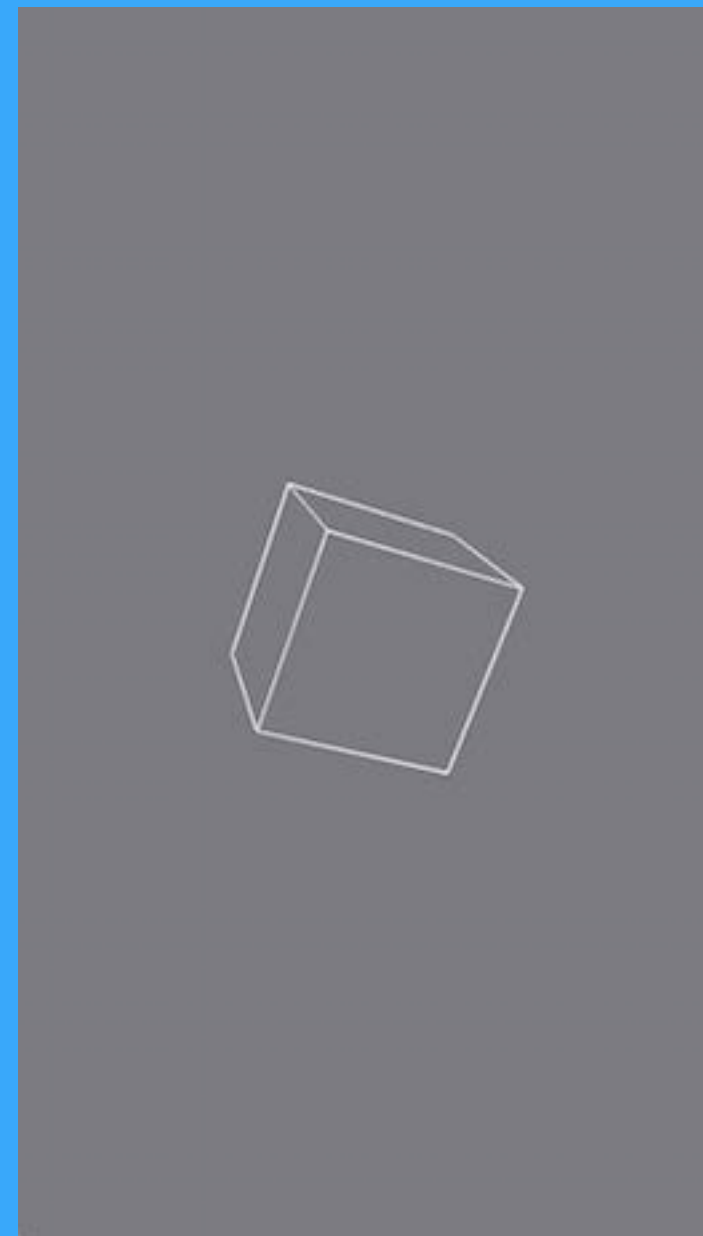
```
[MTLCommandBuffer presentDrawable:]  
[MTLCommandBuffer commit]
```

Releases the drawable

Application Frame

```
[MTKView currentRenderPassDescriptor]
```

Reserves a drawable



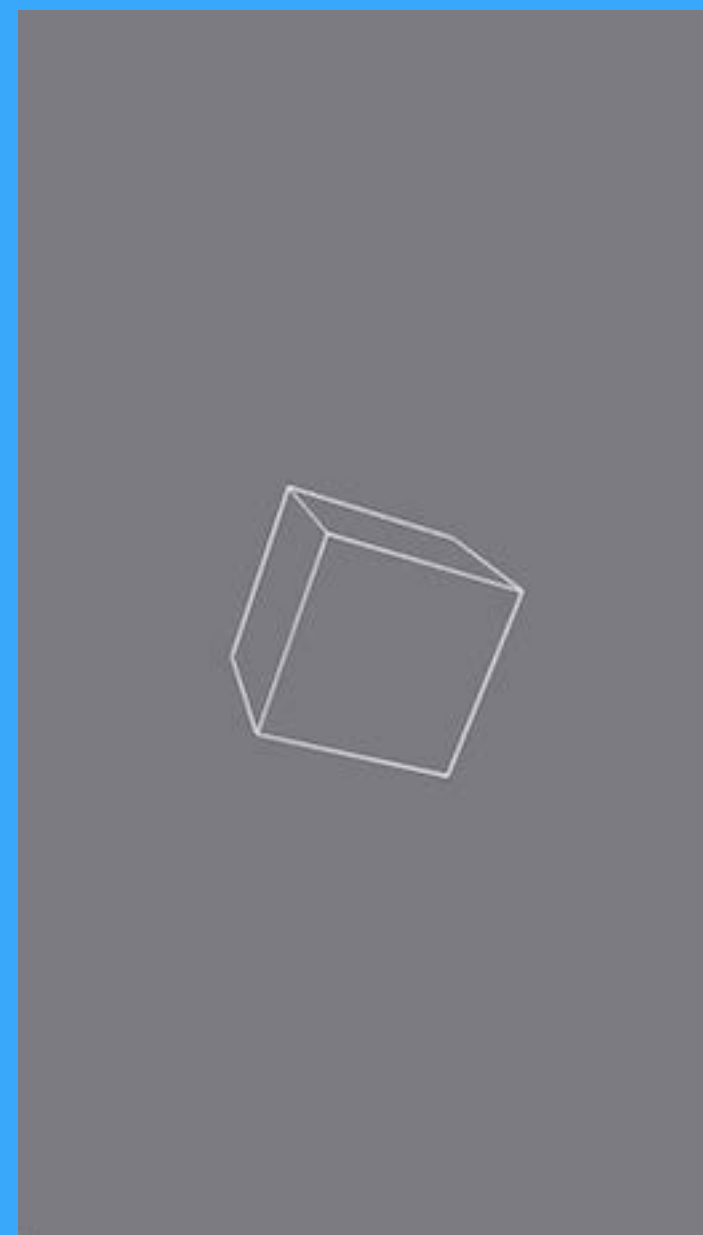
Encodes to the drawable

```
[MTLCommandBuffer presentDrawable:]  
[MTLCommandBuffer commit]
```

Releases the drawable

Application Frame

[MTKView currentRenderPassDescriptor]



[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

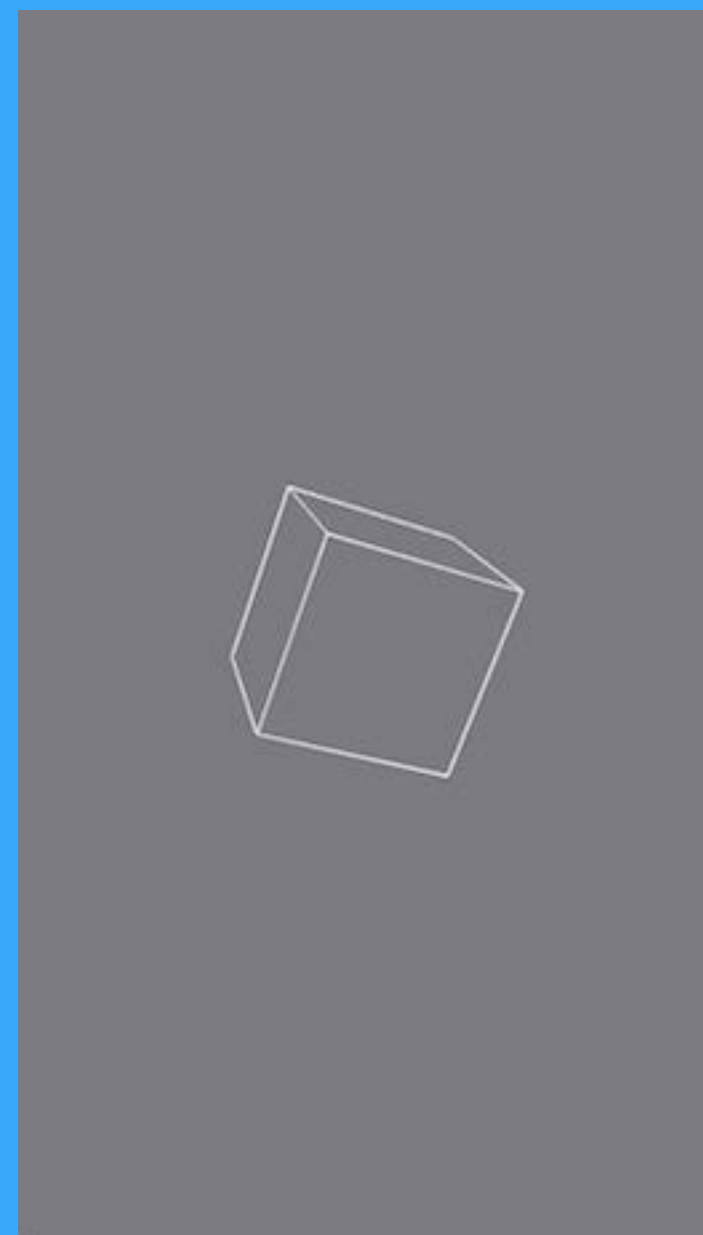
Reserves a drawable

Encodes to the drawable

Releases the drawable

Application Frame

[MTKView currentRenderPassDescriptor]



[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

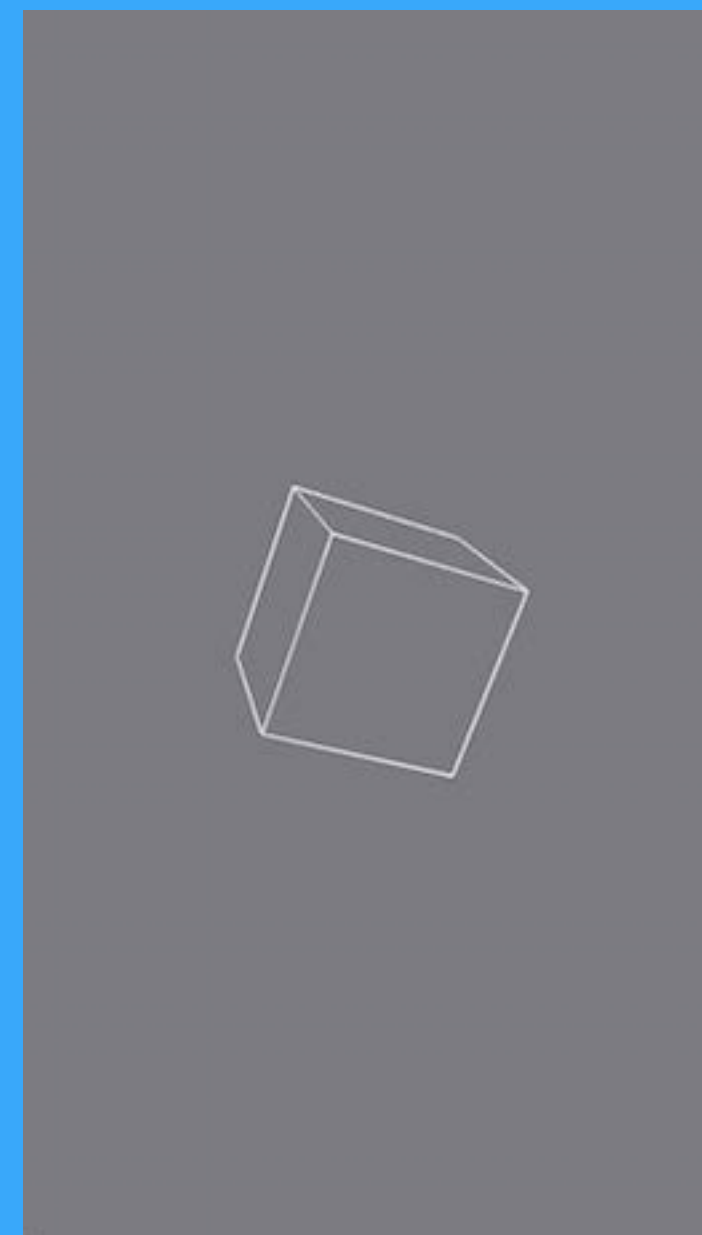
Reserves a drawable

Encodes to the drawable

Releases the drawable

Application Frame

[MTKView currentRenderPassDescriptor]



[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

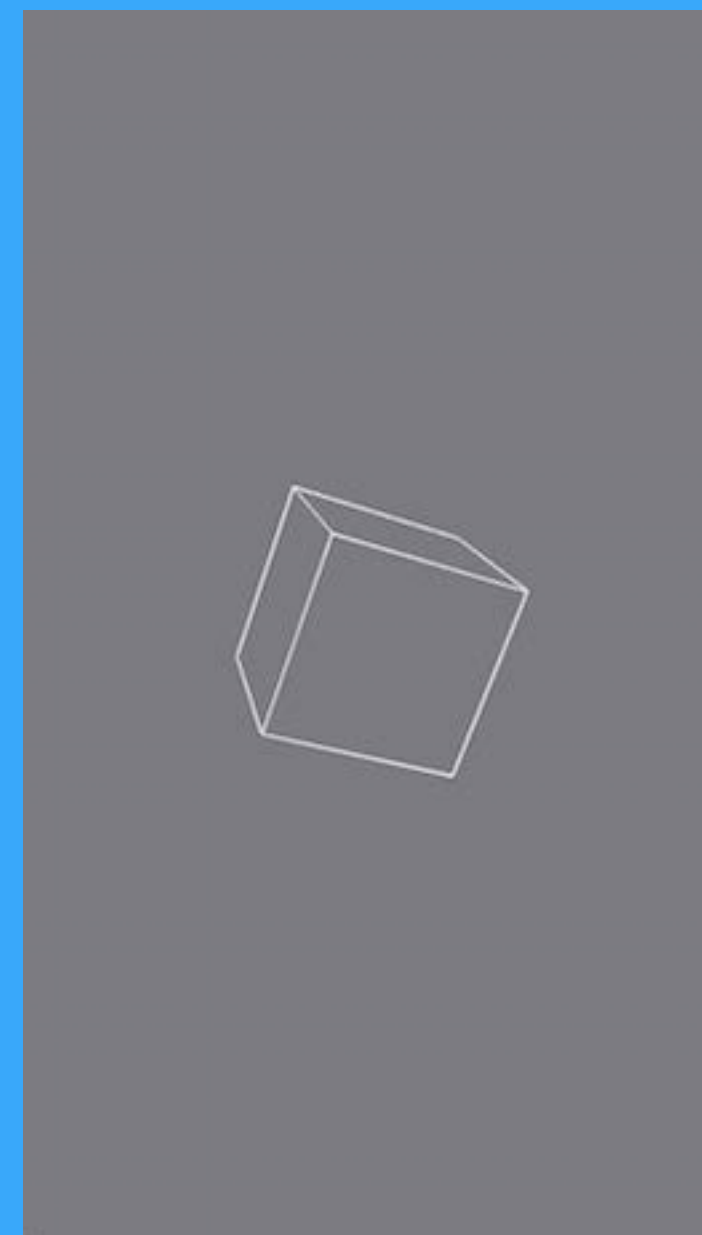
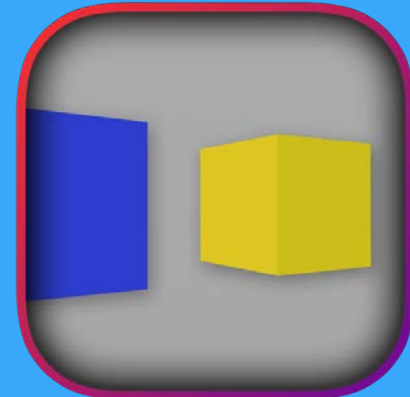
Reserves a drawable

Encodes to the drawable

Releases the drawable

Application Frame

[MTKView currentRenderPassDescriptor]



[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

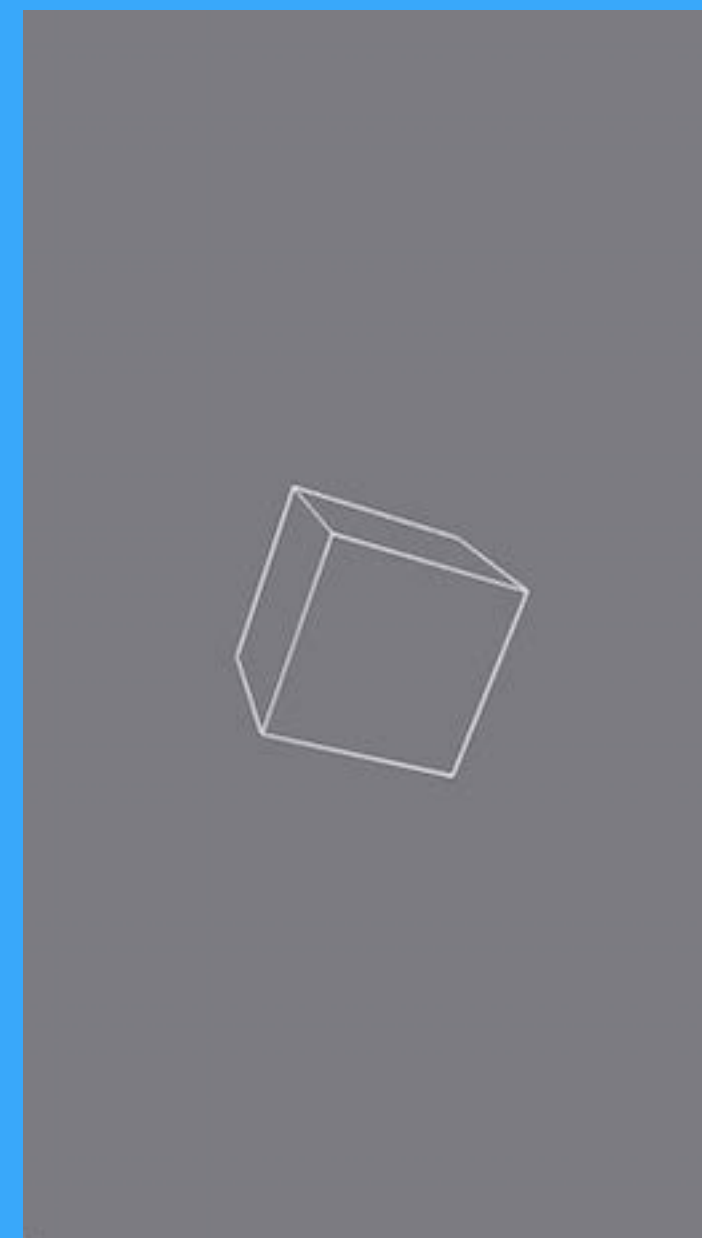
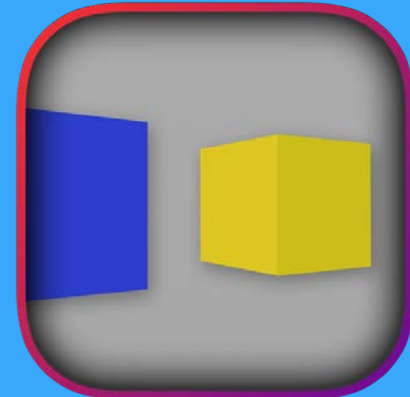
Reserves a drawable

Encodes to the drawable

Releases the drawable

Application Frame

[MTKView currentRenderPassDescriptor]



[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

Reserves a drawable

Encodes to the drawable

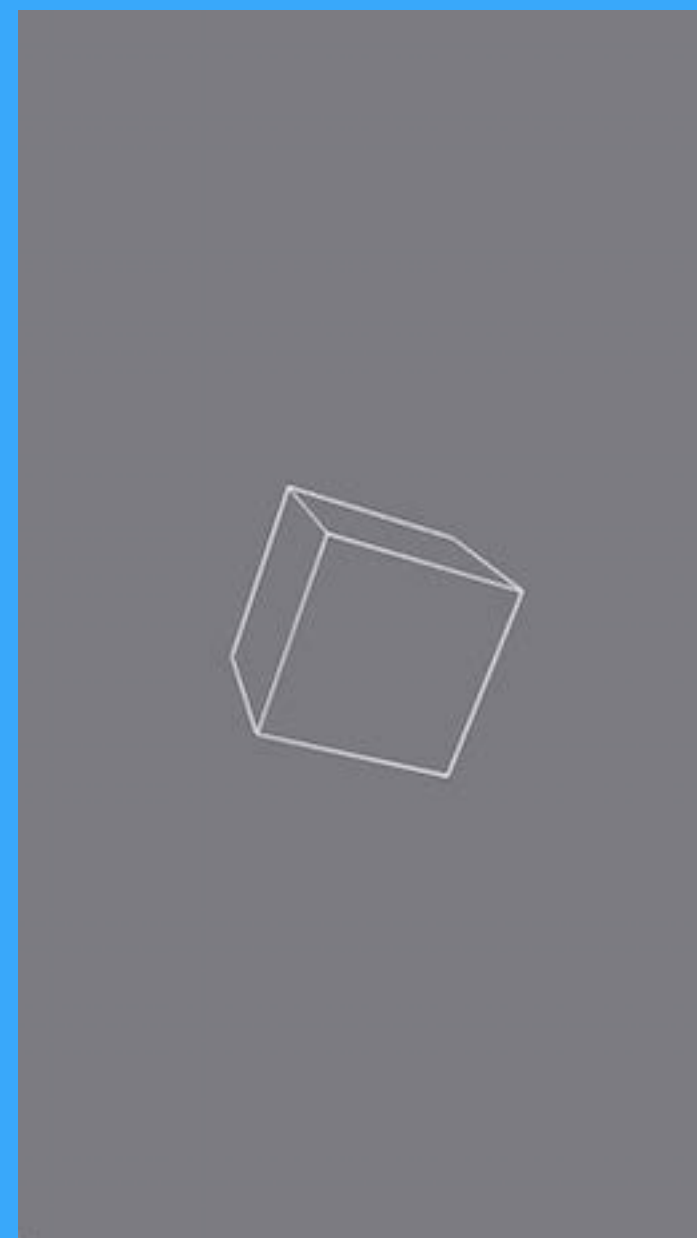
Releases the drawable

Application Frame



[MTKView currentRenderPassDescriptor]

Reserves a drawable



Encodes to the drawable

[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

Releases the drawable

MetalKit View Drawing

Efficient usage

```
- (void)drawInView:(nonnull MTKView *)view {  
    // Update app's render state and encode offscreen passes  
    ...  
  
    id <MTLRenderPassDescriptor> descriptor =  
        view.currentRenderPassDescriptor;  
  
    // Create render command encoder and encode final pass  
    ...  
  
    [commandBuffer presentDrawable:view.currentDrawable];  
    [commandBuffer commit];  
}
```

MetalKit View Drawing

Efficient usage

```
- (void)drawInView:(nonnull MTKView *)view {  
    // Update app's render state and encode offscreen passes  
    ...  
  
    id <MTLRenderPassDescriptor> descriptor =  
        view.currentRenderPassDescriptor;  
  
    // Create render command encoder and encode final pass  
    ...  
  
    [commandBuffer presentDrawable:view.currentDrawable];  
    [commandBuffer commit];  
}
```

MetalKit View Drawing

Efficient usage

```
- (void)drawInView:(nonnull MTKView *)view {  
    // Update app's render state and encode offscreen passes  
    ...
```

```
    id <MTLRenderPassDescriptor> descriptor =  
        view.currentRenderPassDescriptor;  
  
    // Create render command encoder and encode final pass  
    ...  
  
    [commandBuffer presentDrawable:view.currentDrawable];  
    [commandBuffer commit];  
}
```

MetalKit View Drawing

Efficient usage

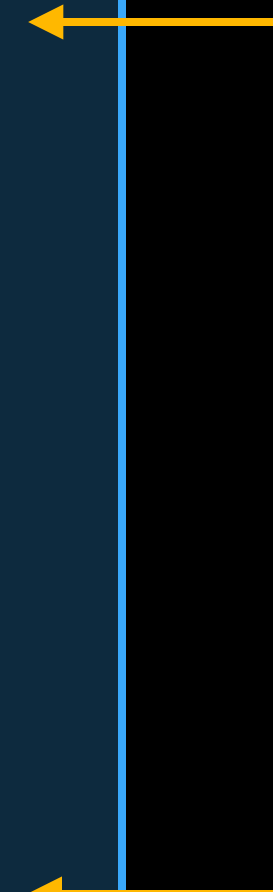
```
- (void)drawInView:(nonnull MTKView *)view {  
    // Update app's render state and encode offscreen passes  
    ...
```

```
    id <MTLRenderPassDescriptor> descriptor =  
        view.currentRenderPassDescriptor;
```

```
    // Create render command encoder and encode final pass  
    ...
```

```
    [commandBuffer presentDrawable:view.currentDrawable];  
    [commandBuffer commit];
```

```
}
```



MetalKit Texture Loader

MetalKit Texture Loader

Texture loading made simple

- Give a reference, get a Metal texture

MetalKit Texture Loader

Texture loading made simple

- Give a reference, get a Metal texture

Fast and fully featured

- Asynchronously decodes files and creates textures
- Support for common image file formats including JPG, TIFF, and PNG
- Also support for PVR and KTX texture file formats

Texture Loader

Basic usage

1. Initialize with Metal device

```
MTKTextureLoader *textureLoader =  
    [[MTKTextureLoader alloc] initWithDevice:device];
```

2. Load texture

```
[textureLoader textureWithContentsOfURL:fileURL  
                                options:nil  
                                completionHandler:myCompletionHandler];
```

Texture Loader

Basic usage

1. Initialize with Metal device

```
MTKTextureLoader *textureLoader =  
    [[MTKTextureLoader alloc] initWithDevice:device];
```

2. Load texture

```
[textureLoader textureWithContentsOfURL:fileURL  
                        options:nil  
                        completionHandler:myCompletionHandler];
```

Texture Loader

Basic usage

1. Initialize with Metal device

```
MTKTextureLoader *textureLoader =  
    [[MTKTextureLoader alloc] initWithDevice:device];
```

2. Load texture

```
[textureLoader textureWithContentsOfURL:fileURL  
                                options:nil  
                                completionHandler:myCompletionHandler];
```

Texture Loader

Basic usage

1. Initialize with Metal device

```
MTKTextureLoader *textureLoader =  
    [[MTKTextureLoader alloc] initWithDevice:device];
```

2. Load texture

```
[textureLoader textureWithContentsOfURL:fileURL  
                                options:nil  
                                completionHandler:myCompletionHandler];
```

Texture Loader

Basic usage

1. Initialize with Metal device

```
MTKTextureLoader *textureLoader =  
    [[MTKTextureLoader alloc] initWithDevice:device];
```

2. Load texture

```
[textureLoader textureWithContentsOfURL:fileURL  
                        options:nil  
                        completionHandler:myCompletionHandler];
```

Texture Loader

Basic usage

1. Initialize with Metal device

```
MTKTextureLoader *textureLoader =  
    [[MTKTextureLoader alloc] initWithDevice:device];
```

2. Load texture

```
[textureLoader textureWithContentsOfURL:fileURL  
                        options:nil  
                        completionHandler:myCompletionHandler];
```


Model I/O Review

Model I/O introduced in iOS 9 and OS X El Capitan

Model I/O Review

Model I/O introduced in iOS 9 and OS X El Capitan

3D Asset loading from various file formats

- Importers and exporters for proprietary formats possible

Model I/O Review

Model I/O introduced in iOS 9 and OS X El Capitan

3D Asset loading from various file formats

- Importers and exporters for proprietary formats possible

Offline baking operations

- Static ambient occlusion
- Light map generation
- Voxelization of meshes

Model I/O Review

Model I/O introduced in iOS 9 and OS X El Capitan

3D Asset loading from various file formats

- Importers and exporters for proprietary formats possible

Offline baking operations

- Static ambient occlusion
- Light map generation
- Voxelization of meshes

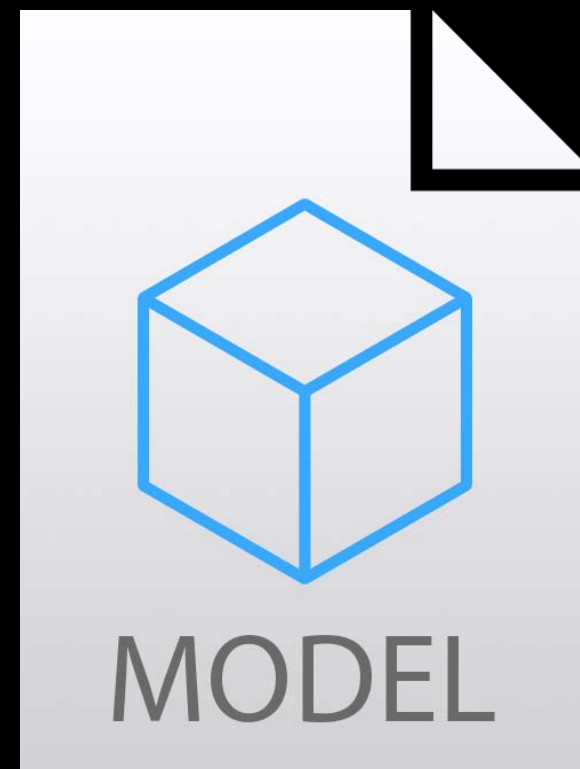
Allows you to focus on your rendering code

MetalKit and Model I/O

Utilities to efficiently use Model I/O with Metal

- Optimized loading of Model I/O meshes into Metal buffers
- Encapsulation of mesh data for Metal
- Functions to prepare mesh data for Metal pipelines

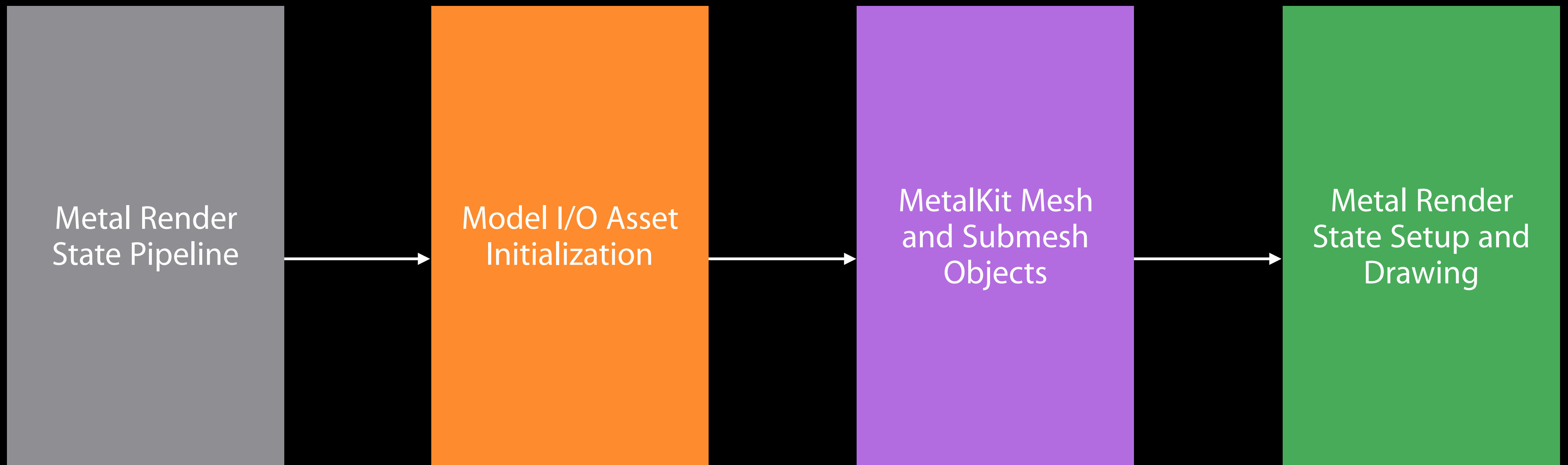
Rendering a Model I/O Asset



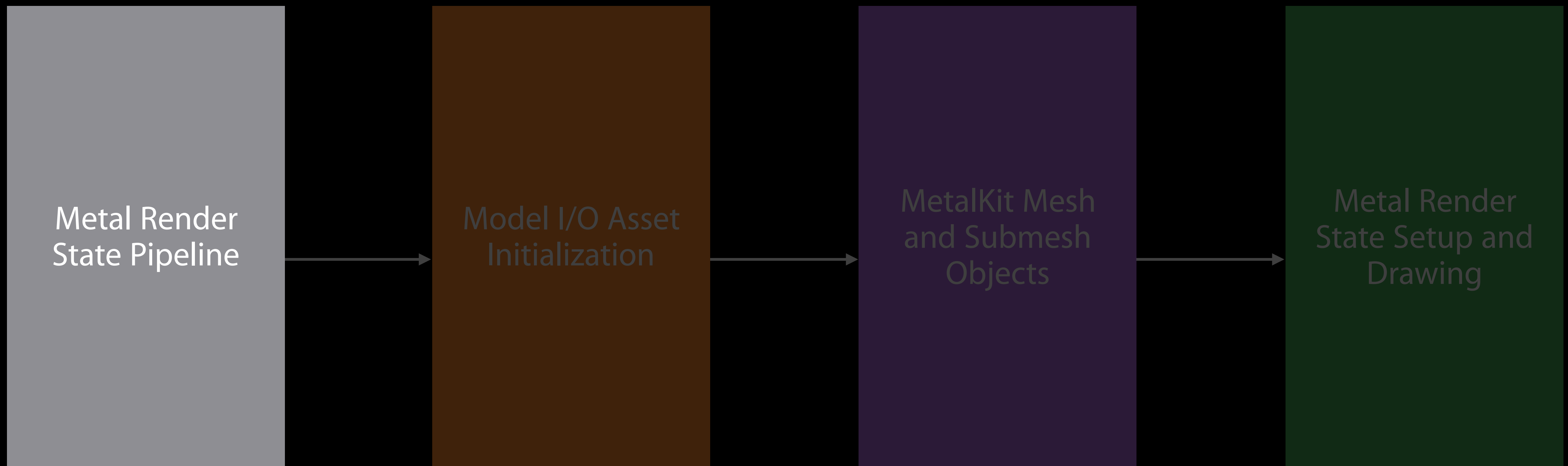
Rendering a Model I/O Asset



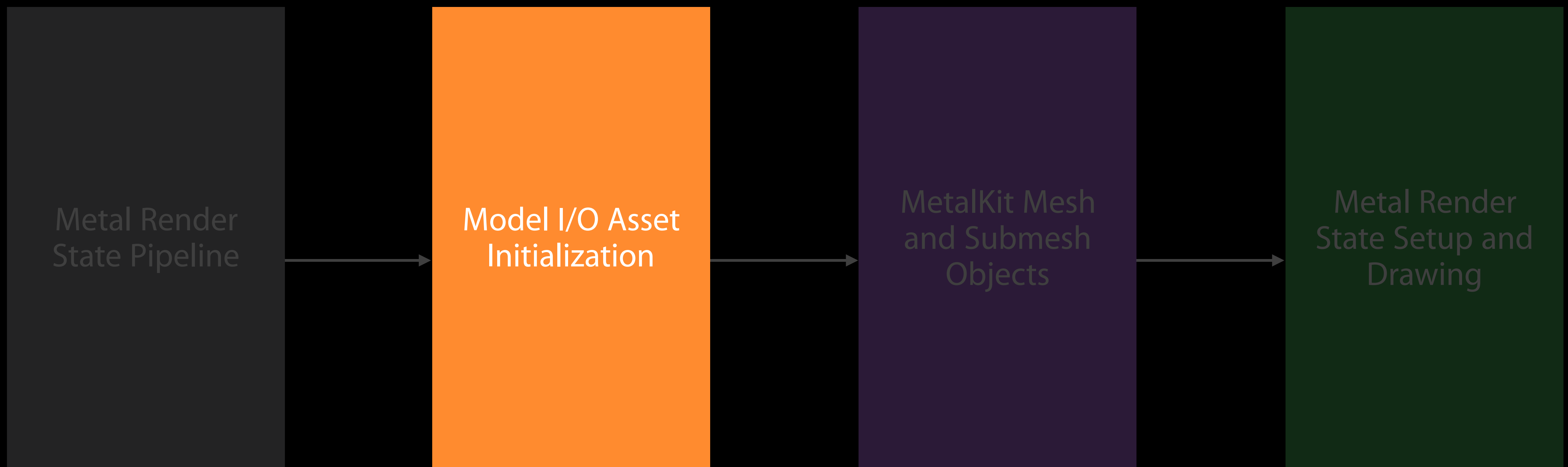
Rendering a Model I/O Asset



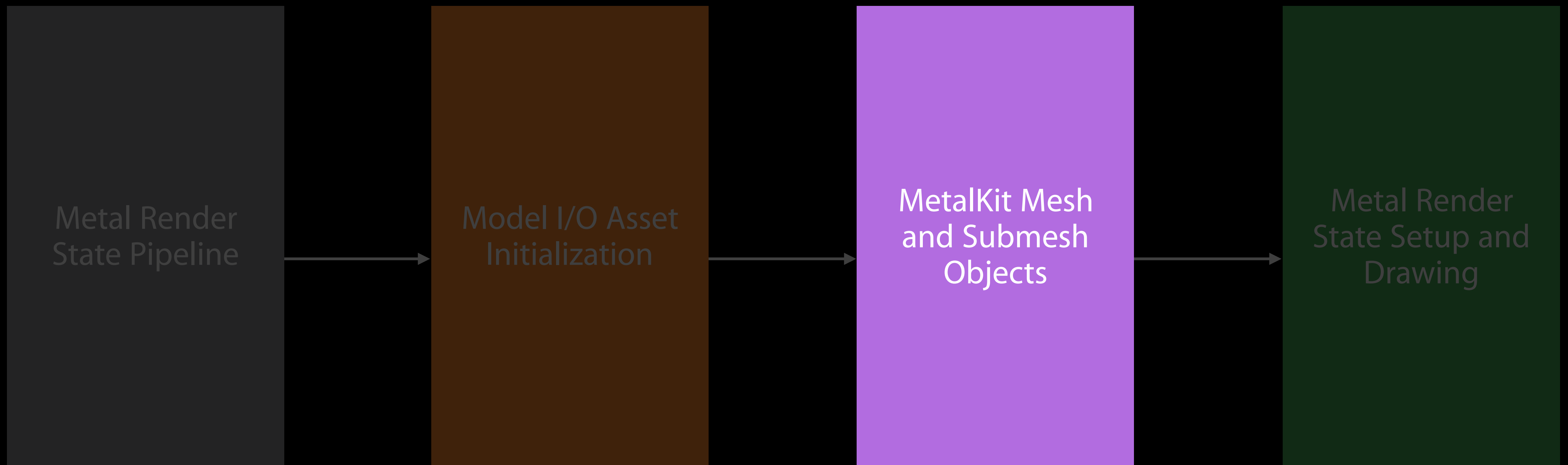
Rendering a Model I/O Asset



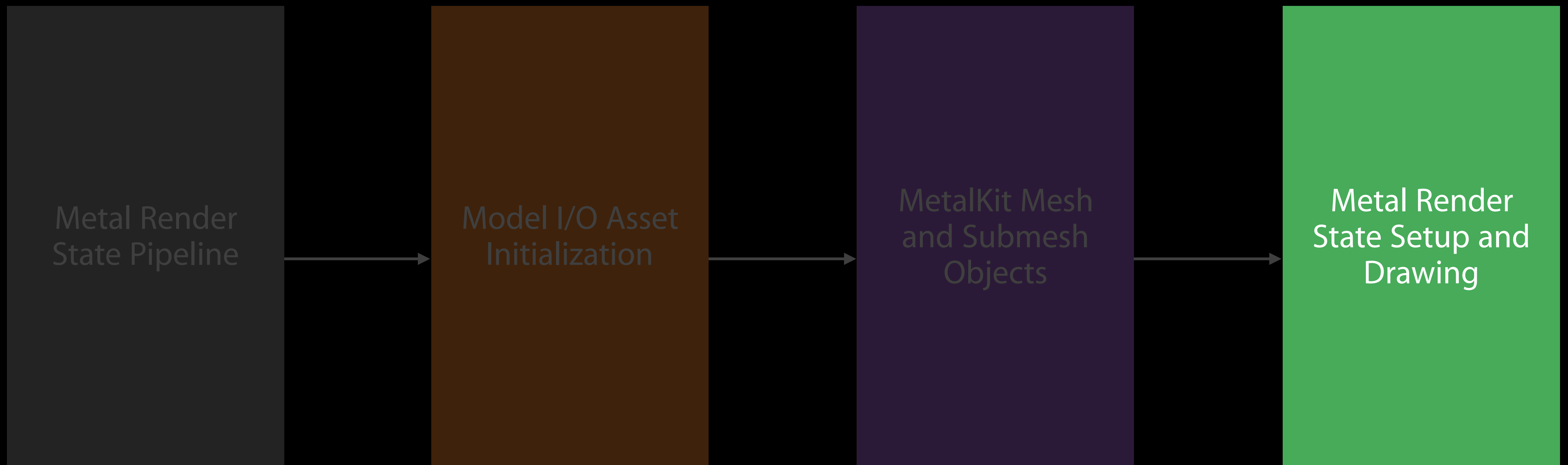
Rendering a Model I/O Asset



Rendering a Model I/O Asset

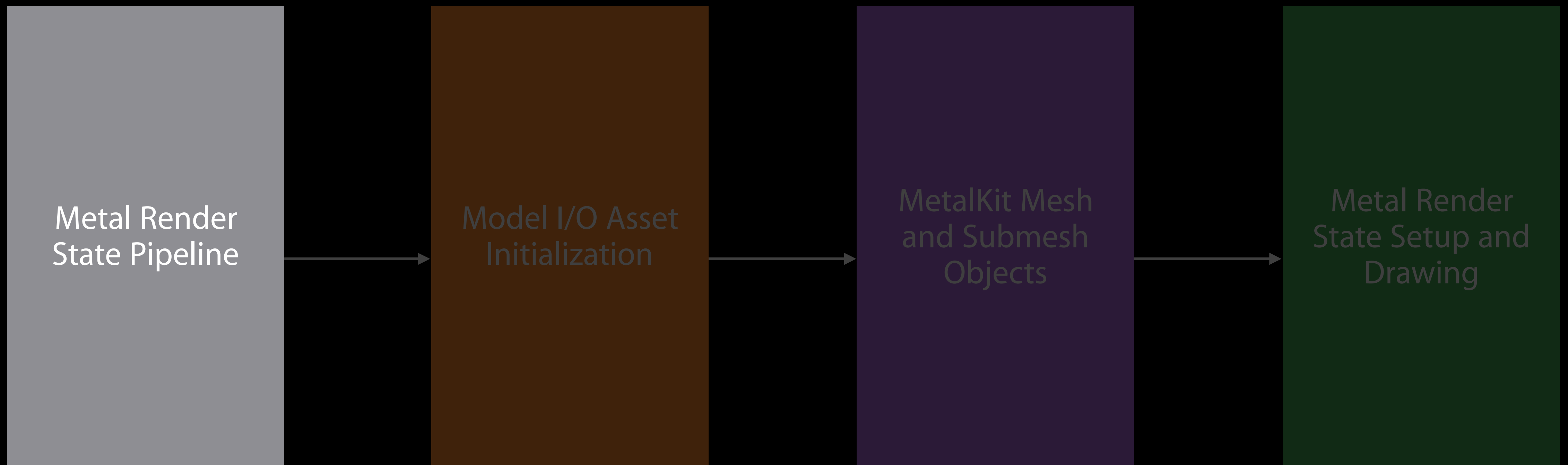


Rendering a Model I/O Asset



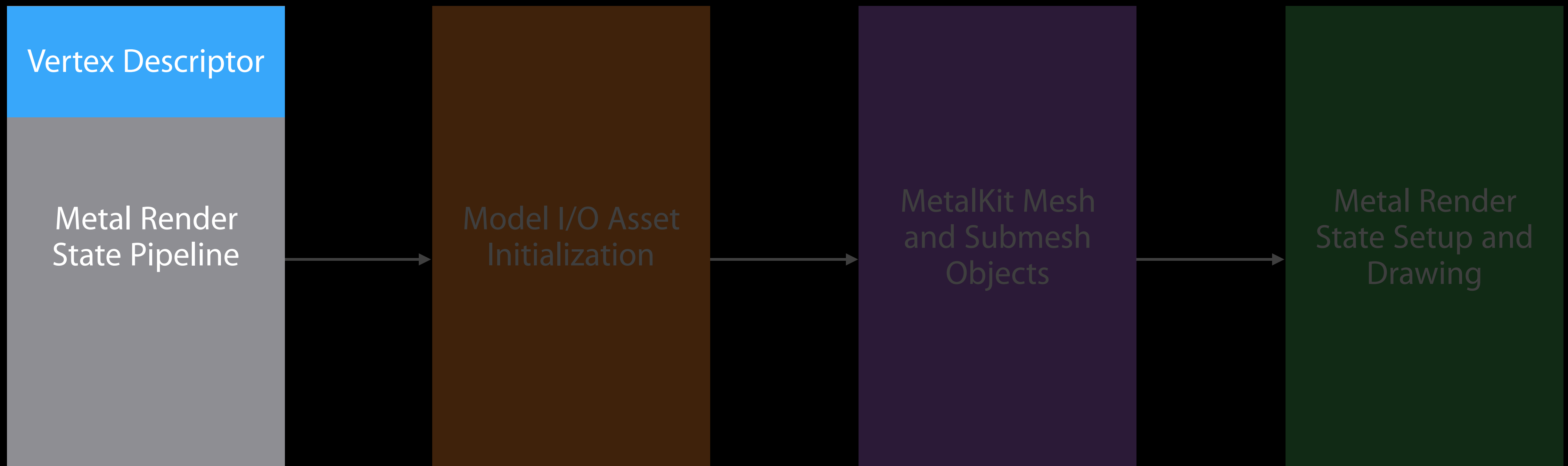
Setup for Model Rendering

Pipeline setup



Setup for Model Rendering

Pipeline setup



Vertex Shader

Setting up Per-Vertex Inputs

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV      [[ attribute(2) ]];  
};  
  
vertex VertexOutput  
vertexFunction(VertexInput current [[ stage_in ]])  
{  
    ...  
}
```

Vertex Shader

Setting up Per-Vertex Inputs

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color    [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};  
  
vertex VertexOutput  
vertexFunction(VertexInput current [[ stage_in ]])  
{  
    ...  
}
```


Vertex Shader

Setting up Per-Vertex Inputs

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};  
  
vertex VertexOutput  
vertexFunction(VertexInput current [[ stage_in ]])  
{  
    ...  
}
```

Vertex Shader

Setting up Per-Vertex Inputs

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color    [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};  
  
vertex VertexOutput  
vertexFunction(VertexInput current [[ stage_in ]])  
{  
    ...  
}
```

Vertex Shader

Setting up Per-Vertex Inputs

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};  
  
vertex VertexOutput  
vertexFunction(VertexInput current [[ stage_in ]])  
{  
    ...  
}
```

```
struct VertexInput {  
    float3    position [[ attribute(0) ]];  
    float4    color     [[ attribute(1) ]];  
    float2    texUV     [[ attribute(2) ]];  
};
```

```
struct VertexInput {  
    float3    position [[ attribute(0) ]];  
    float4    color     [[ attribute(1) ]];  
    float2    texUV     [[ attribute(2) ]];  
};
```

```
struct VertexInput {  
    float3    position [[ attribute(0) ]];  
    float4    color     [[ attribute(1) ]];  
    float2    texUV     [[ attribute(2) ]];  
};
```

```
struct VertexInput {  
    float3    position [[ attribute(0) ]];  
    float4    color     [[ attribute(1) ]];  
    float2    texUV     [[ attribute(2) ]];  
};
```

Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
MTLVertexDescriptor *metalVertexDesc = [MTLVertexDescriptor new];
```


Vertex Descriptor

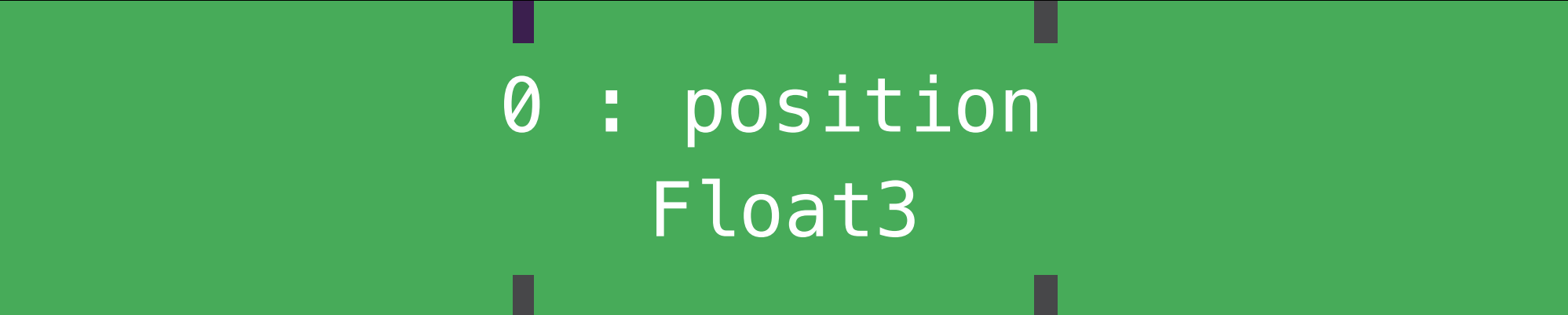
```
struct VertexInput {  
    float3    position [[ attribute(0) ]];  
    float4    color     [[ attribute(1) ]];  
    float2    texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[0].format = MTLVertexFormatFloat3;  
metalVertexDesc.attributes[0].offset = 0;
```

Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV      [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[0].format = MTLVertexFormatFloat3;  
metalVertexDesc.attributes[0].offset = 0;
```

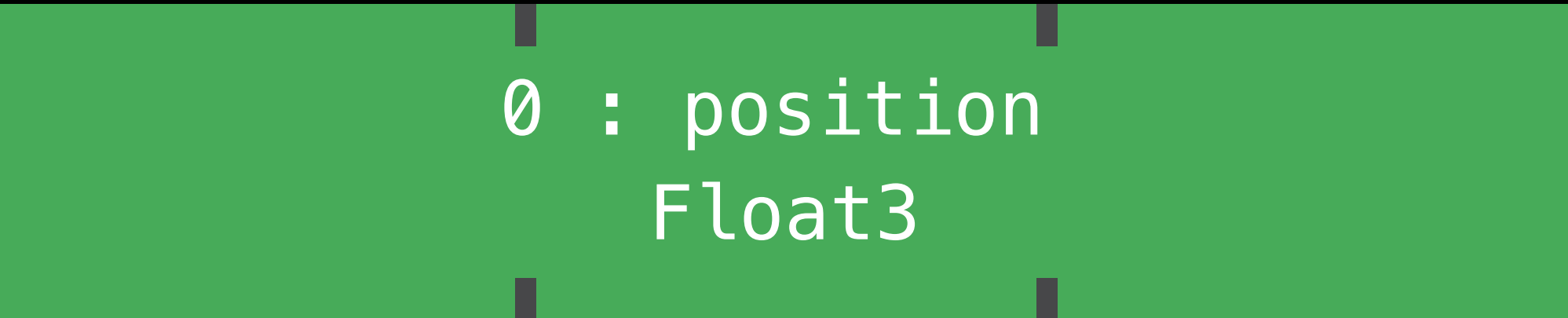


0 : position
Float3

Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV      [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[1].format = MTLVertexFormatUChar4Normalized;  
metalVertexDesc.attributes[1].offset = 12;
```



0 : position
Float3

Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[1].format = MTLVertexFormatUChar4Normalized;  
metalVertexDesc.attributes[1].offset = 12;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[1].format = MTLVertexFormatUChar4Normalized;  
metalVertexDesc.attributes[1].offset = 12;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[1].format = MTLVertexFormatUChar4Normalized;  
metalVertexDesc.attributes[1].offset = 12;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[2].format = MTLVertexFormatHalfFloat2;  
metalVertexDesc.attributes[2].offset = 16;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[2].format = MTLVertexFormatHalfFloat2;  
metalVertexDesc.attributes[2].offset = 16;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color    [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

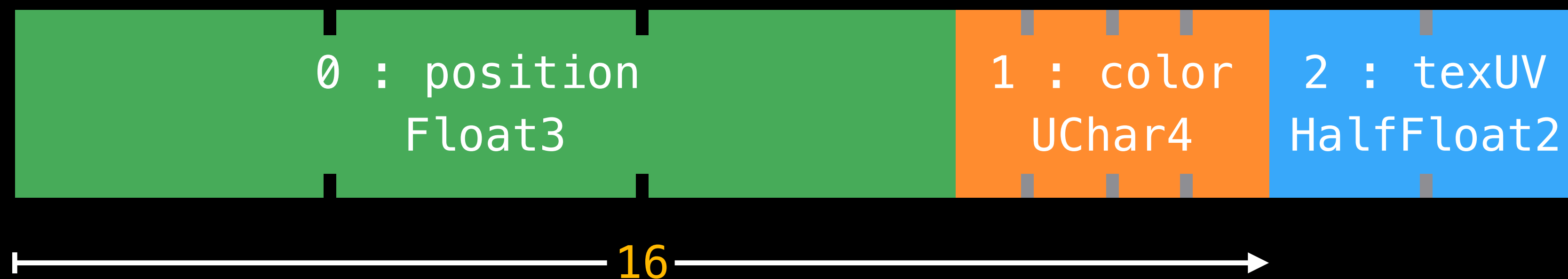
```
metalVertexDesc.attributes[2].format = MTLVertexFormatHalfFloat2;  
metalVertexDesc.attributes[2].offset = 16;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.attributes[2].format = MTLVertexFormatHalfFloat2;  
metalVertexDesc.attributes[2].offset = 16;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

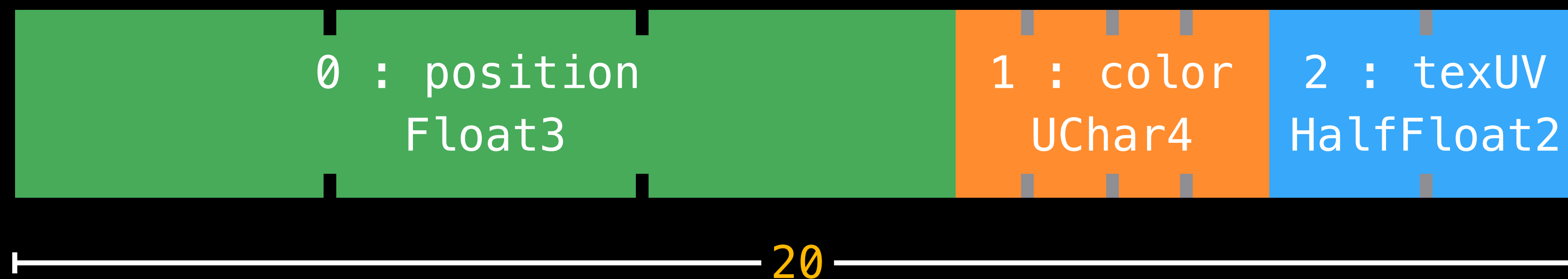
```
metalVertexDesc.layouts[0].stride = 20;
```



Vertex Descriptor

```
struct VertexInput {  
    float3  position [[ attribute(0) ]];  
    float4  color     [[ attribute(1) ]];  
    float2  texUV     [[ attribute(2) ]];  
};
```

```
metalVertexDesc.layouts[0].stride = 20;
```



Vertex Array Layout



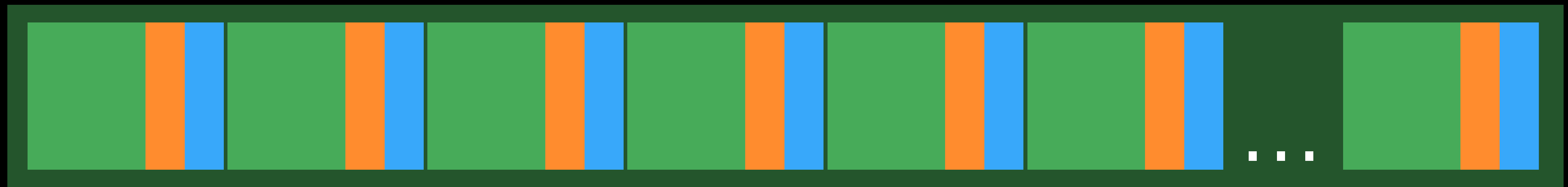
Vertex Array Layout



Vertex Array Layout



Vertex Array Layout



Building the Pipeline

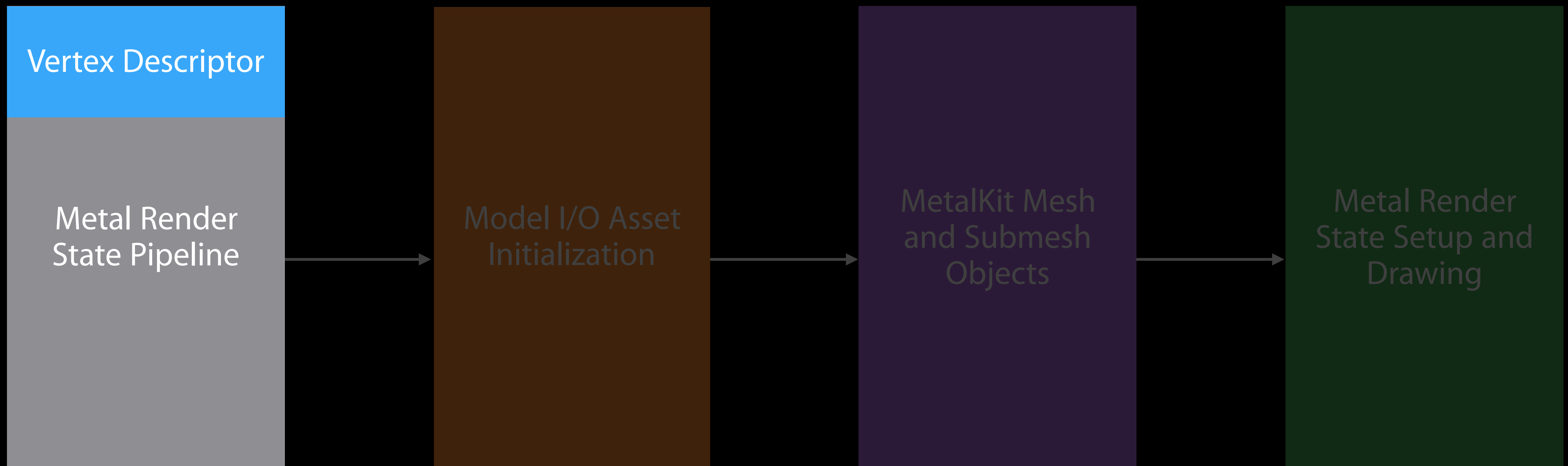
```
renderPipelineDescriptor.vertexDescriptor = metalVertexDesc;
MTLRenderStatePipeline *pipeline =
    [device.newRenderPipelineStateWithDescriptor:renderPipelineDescriptor
                                     error:error];
```

Building the Pipeline

```
renderPipelineDescriptor.vertexDescriptor = metalVertexDesc;
MTLRenderStatePipeline *pipeline =
    [device.newRenderPipelineStateWithDescriptor:renderPipelineDescriptor
                                     error:error];
```

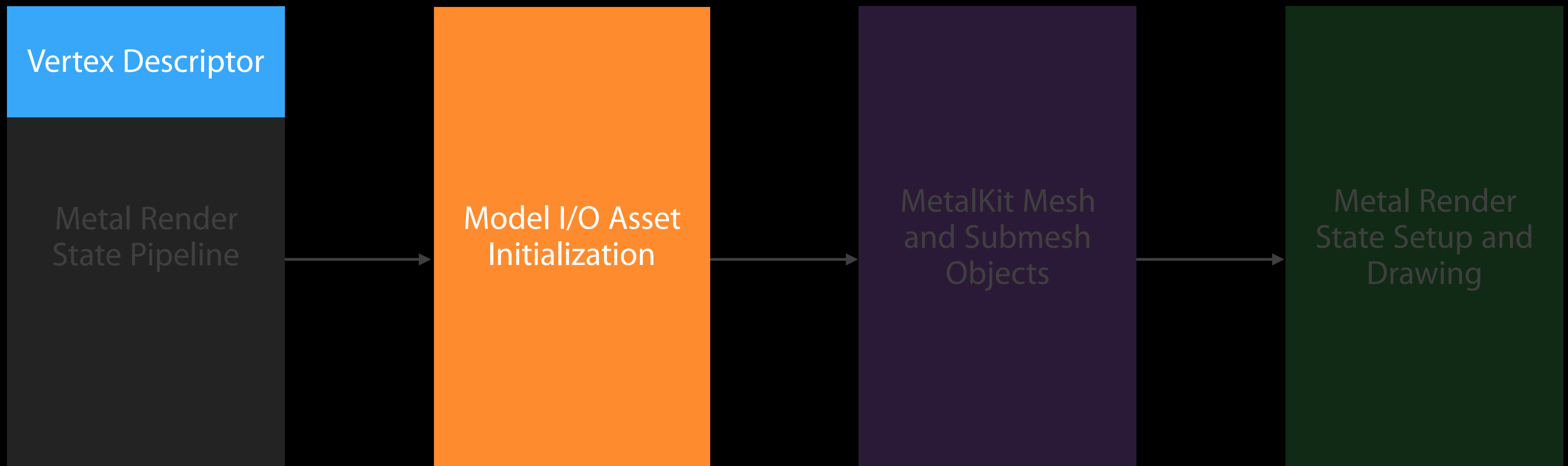

Setup for Model Rendering

Asset initialization



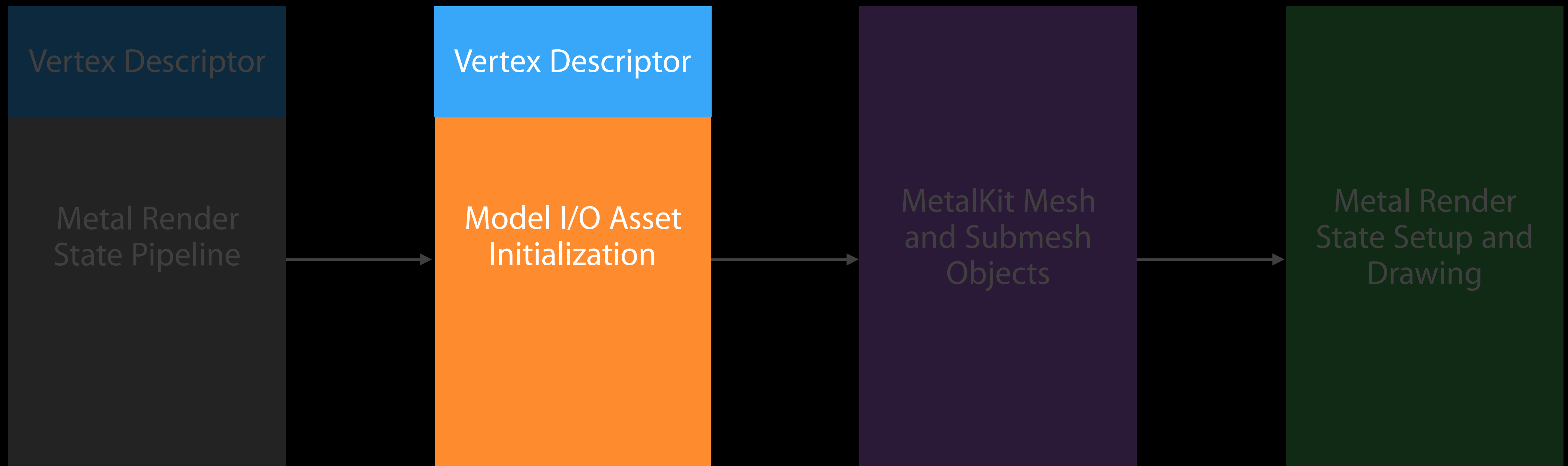
Setup for Model Rendering

Asset initialization



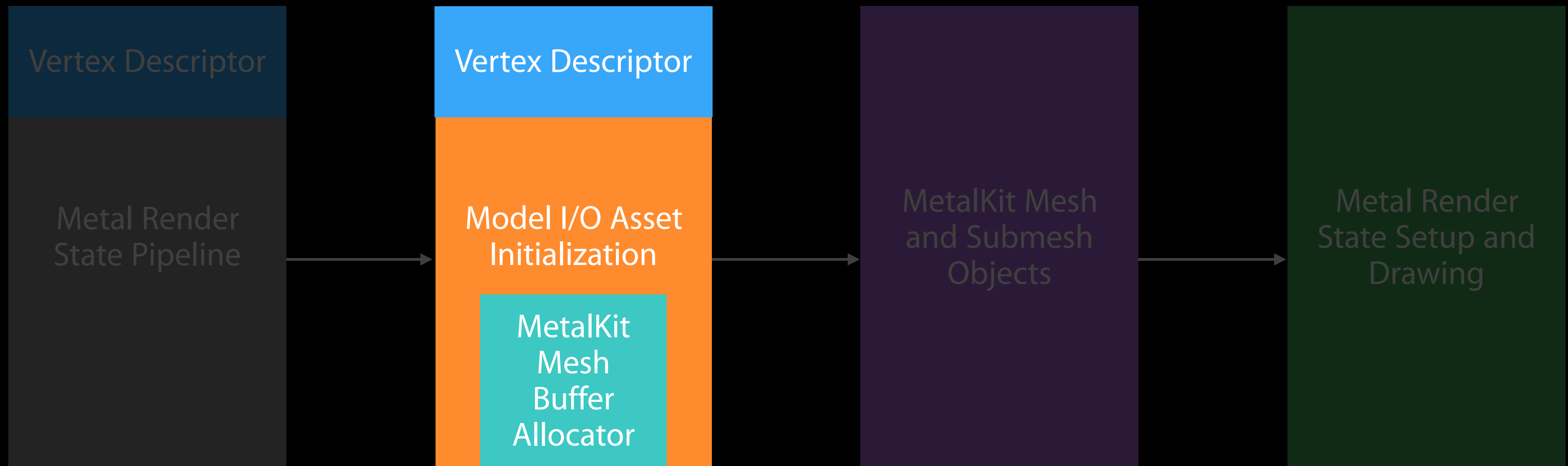
Setup for Model Rendering

Asset initialization



Setup for Model Rendering

Asset initialization



Asset Initialization

Model I/O Vertex Descriptor vs. Metal Vertex Descriptor

Asset Initialization

Model I/O Vertex Descriptor vs. Metal Vertex Descriptor

Model I/O Vertex Descriptor

- Describes the layout of vertex attributes in a mesh

Asset Initialization

Model I/O Vertex Descriptor vs. Metal Vertex Descriptor

Model I/O Vertex Descriptor

- Describes the layout of vertex attributes in a mesh

Metal Vertex Descriptor

- Describes the layout of vertex attribute inputs to a render state pipeline

Asset Initialization

Model I/O Vertex Descriptor vs. Metal Vertex Descriptor

Model I/O Vertex Descriptor

- Describes the layout of vertex attributes in a mesh

Metal Vertex Descriptor

- Describes the layout of vertex attribute inputs to a render state pipeline

Intentionally designed to look similar

- Both contain an array of attribute and buffer layout objects
- Simplifies translation from one type to other

Asset Initialization

Defaults and layout efficiency

Each attribute in a Model I/O Vertex Descriptor has an identifying string-based name

- Model I/O assigns a default name if one does not exist in the model file
 - Names include **@“position”**, **@“normal”**, **@“textureCoordinate”**, and **@“color”**
 - Model I/O defines these with the string-based **MDLVertexAttribute** name constants

Asset Initialization

Defaults and layout efficiency

Each attribute in a Model I/O Vertex Descriptor has an identifying string-based name

- Model I/O assigns a default name if one does not exist in the model file
 - Names include **@“position”**, **@“normal”**, **@“textureCoordinate”**, and **@“color”**
 - Model I/O defines these with the string-based **MDLVertexAttribute** name constants

Custom **MDLVertexDescriptor** recommended

- By default, Model I/O loads vertices as high-precision floating point types
- Feed pipelines with the smallest type that meets your precision requirements
- Improves vertex bandwidth efficiency

Asset Initialization



Asset Initialization



```
MDLVertexDescriptor *modelIOVertexDesc =  
    MTKModelIOVertexFormatFromMetal(metalVertexDesc);  
  
modelIOVertexDesc.attributes[0].name = MDLVertexAttributePosition;  
modelIOVertexDesc.attributes[1].name = MDLVertexAttributeColor;  
modelIOVertexDesc.attributes[2].name = MDLVertexAttributeTextureCoordinate;
```

Asset Initialization



```
MDLVertexDescriptor *modelIOVertexDesc =  
    MTKModelIOVertexFormatFromMetal(metalVertexDesc);
```

```
modelIOVertexDesc.attributes[0].name = MDLVertexAttributePosition;
```

```
modelIOVertexDesc.attributes[1].name = MDLVertexAttributeColor;
```

```
modelIOVertexDesc.attributes[2].name = MDLVertexAttributeTextureCoordinate;
```


Asset Initialization



```
MDLVertexDescriptor *modelIOVertexDesc =  
    MTKModelIOVertexFormatFromMetal(metalVertexDesc);
```

```
modelIOVertexDesc.attributes[0].name = MDLVertexAttributePosition;
```

```
modelIOVertexDesc.attributes[1].name = MDLVertexAttributeColor;
```

```
modelIOVertexDesc.attributes[2].name = MDLVertexAttributeTextureCoordinate;
```

Asset Initialization



```
MDLVertexDescriptor *modelIOVertexDesc =  
    MTKModelIOVertexFormatFromMetal(metalVertexDesc);
```

```
modelIOVertexDesc.attributes[0].name = MDLVertexAttributePosition;  
modelIOVertexDesc.attributes[1].name = MDLVertexAttributeColor;  
modelIOVertexDesc.attributes[2].name = MDLVertexAttributeTextureCoordinate;
```

Asset Initialization



```
MDLVertexDescriptor *modelIOVertexDesc =  
    MTKModelIOVertexFormatFromMetal(metalVertexDesc);
```

```
modelIOVertexDesc.attributes[0].name = MDLVertexAttributePosition;  
modelIOVertexDesc.attributes[1].name = MDLVertexAttributeColor;  
modelIOVertexDesc.attributes[2].name = MDLVertexAttributeTextureCoordinate;
```

Asset Initialization



```
MDLVertexDescriptor *modelIOVertexDesc =  
    MTKModelIOVertexFormatFromMetal(metalVertexDesc);
```

```
modelIOVertexDesc.attributes[0].name = MDLVertexAttributePosition;  
modelIOVertexDesc.attributes[1].name = MDLVertexAttributeColor;  
modelIOVertexDesc.attributes[2].name = MDLVertexAttributeTextureCoordinate;
```

Asset Initialization

MetalKit Mesh Buffer Allocator

```
MTKMeshBufferAllocator *mtkBufferAllocator =  
    [[MTKMeshBufferAllocator alloc] initWithDevice:metalDevice];
```

Asset Initialization

Creating the Asset Object

```
MDLAsset *asset = [[MDLAsset alloc] initWithURL:fileURL  
                vertexDescriptor:modelioVertexDesc  
                bufferAllocator:mtkBufferAllocator];
```

Asset Initialization

Creating the Asset Object

[illegible]

Asset Initialization

Creating the Asset Object

```
MDLAsset *asset = [[MDLAsset alloc] initWithURL:fileURL  
                vertexDescriptor:modelioVertexDesc  
                bufferAllocator:mtkBufferAllocator];
```

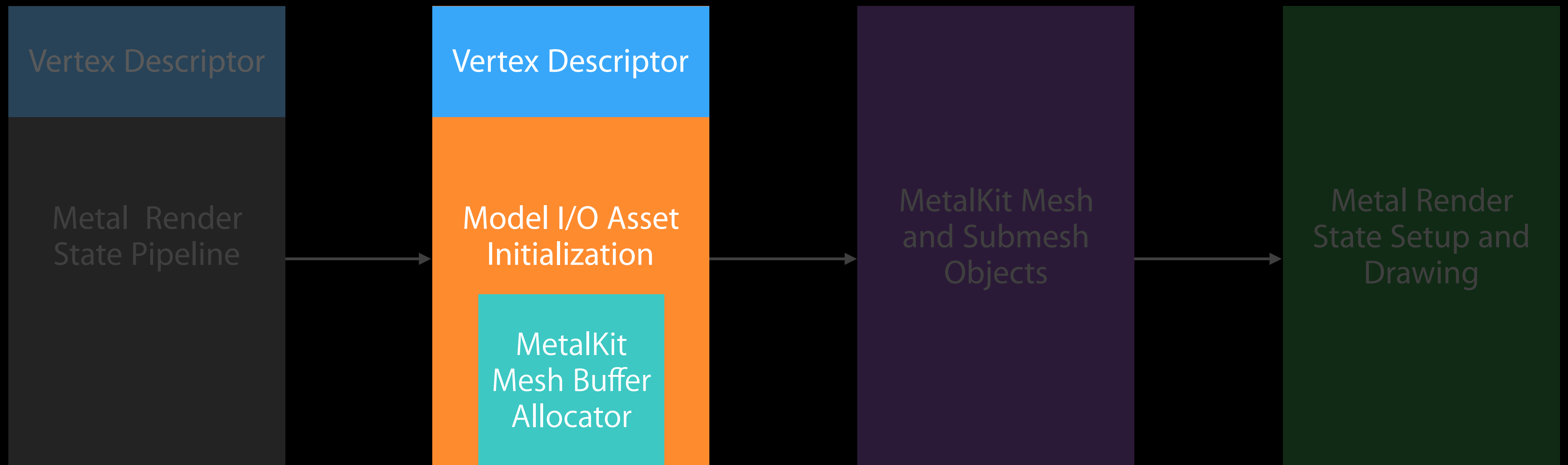

Asset Initialization

Creating the Asset Object

```
MDLAsset *asset = [[MDLAsset alloc] initWithURL:fileURL  
                vertexDescriptor:modelioVertexDesc  
                bufferAllocator:mtkBufferAllocator];
```

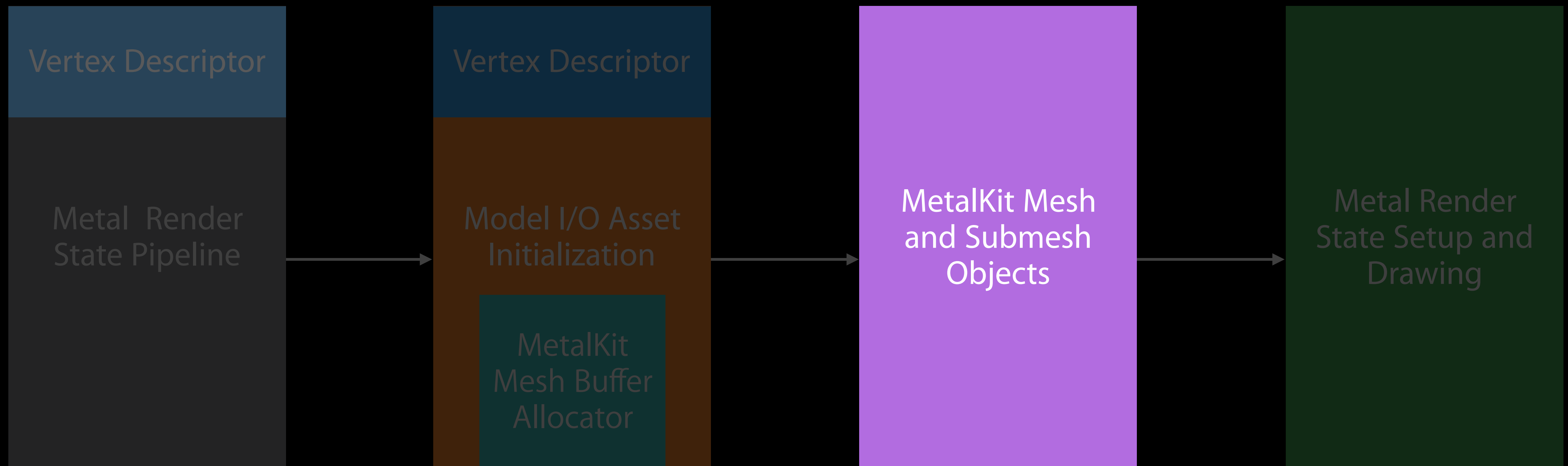
MetalKit Meshes Initialization

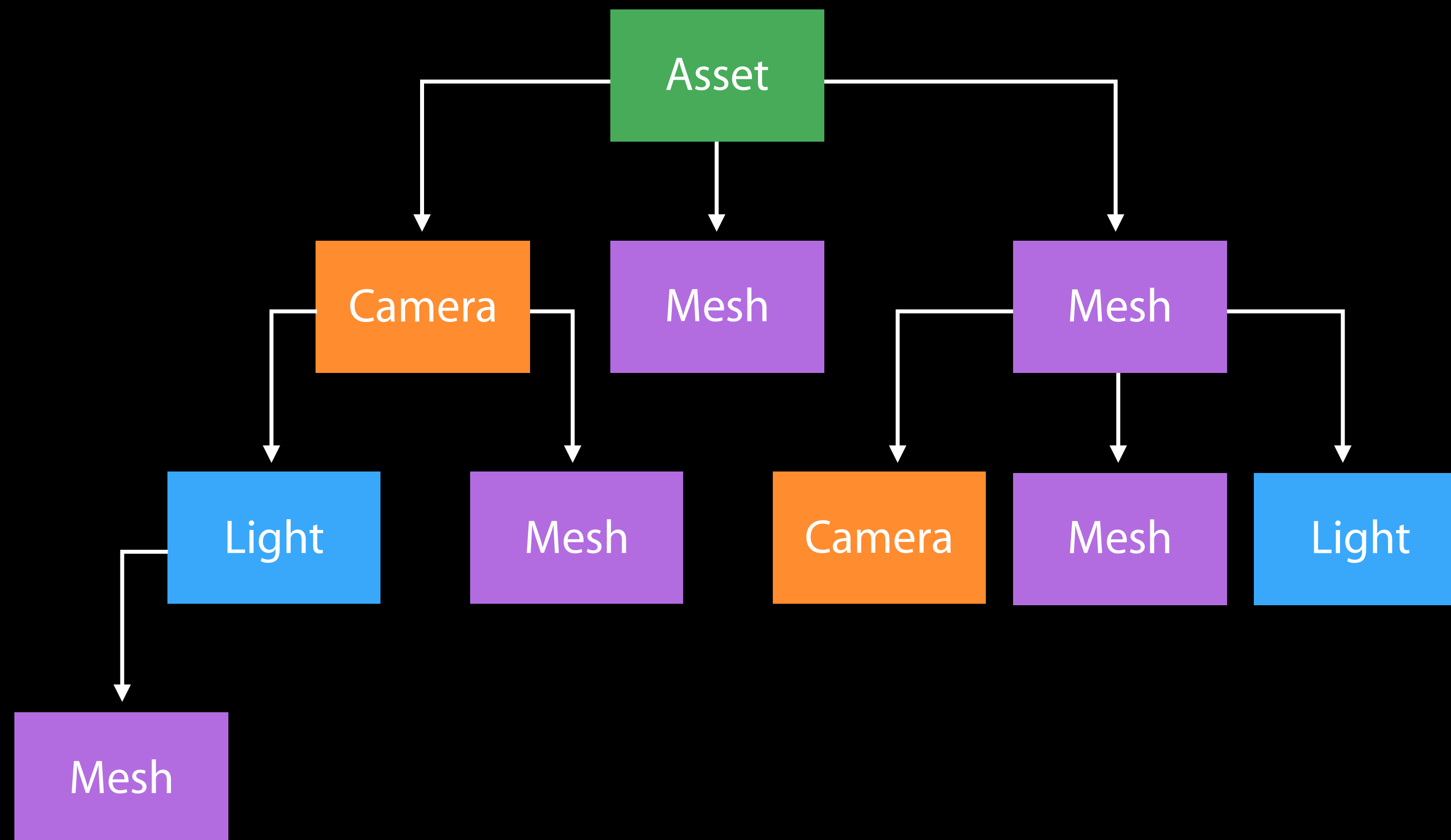
Setup for model rendering

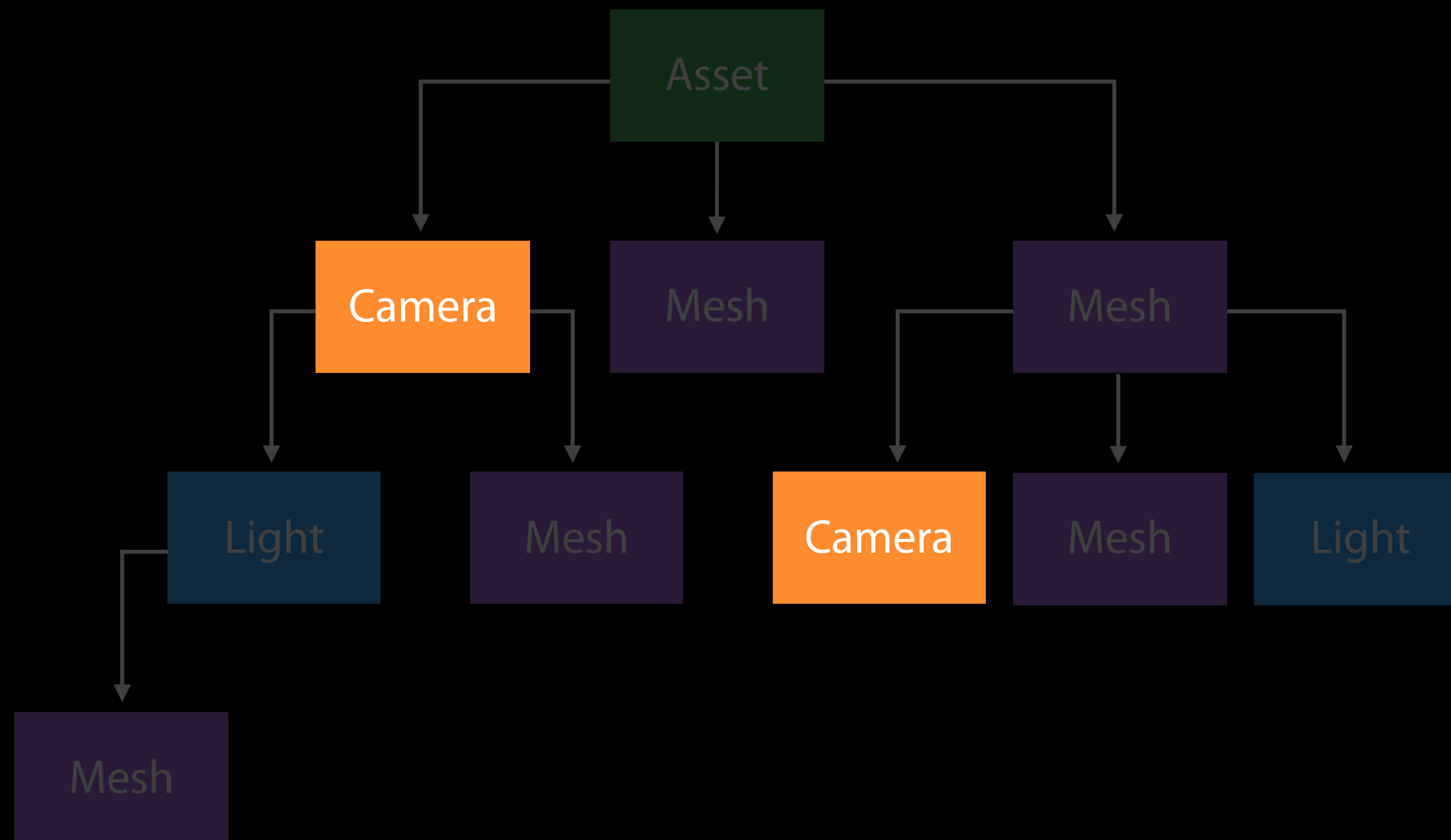


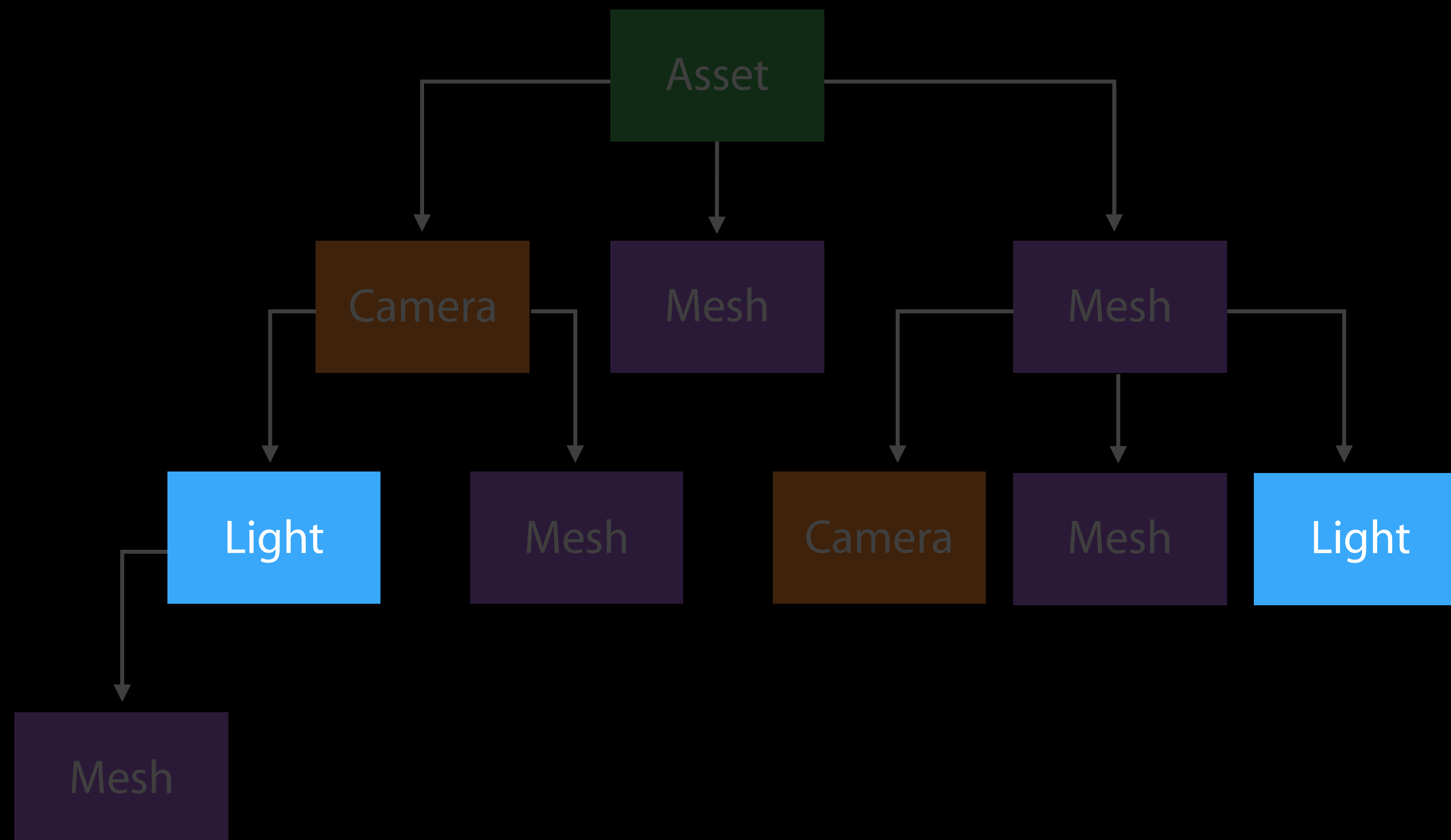
MetalKit Meshes Initialization

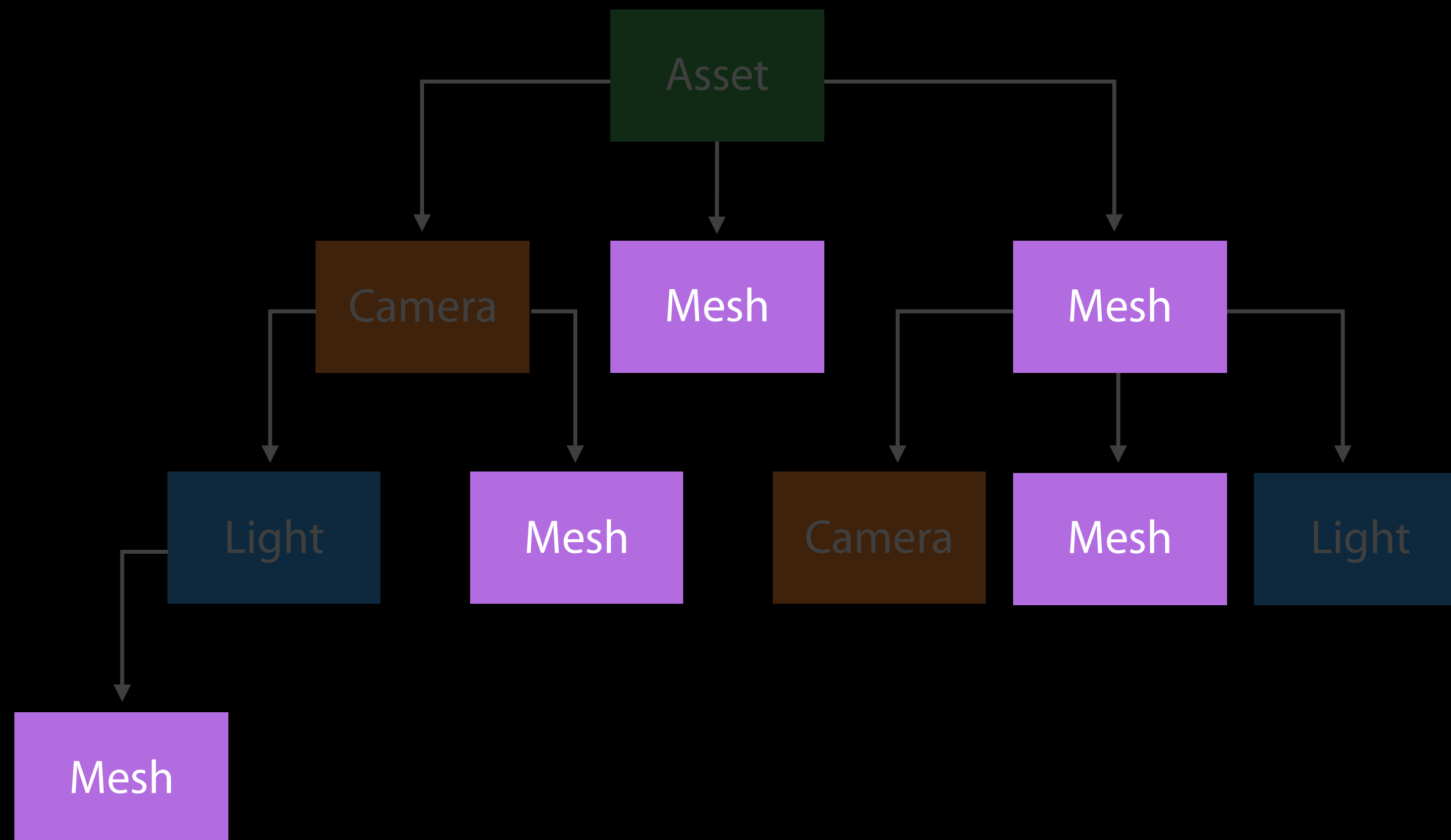
Setup for model rendering



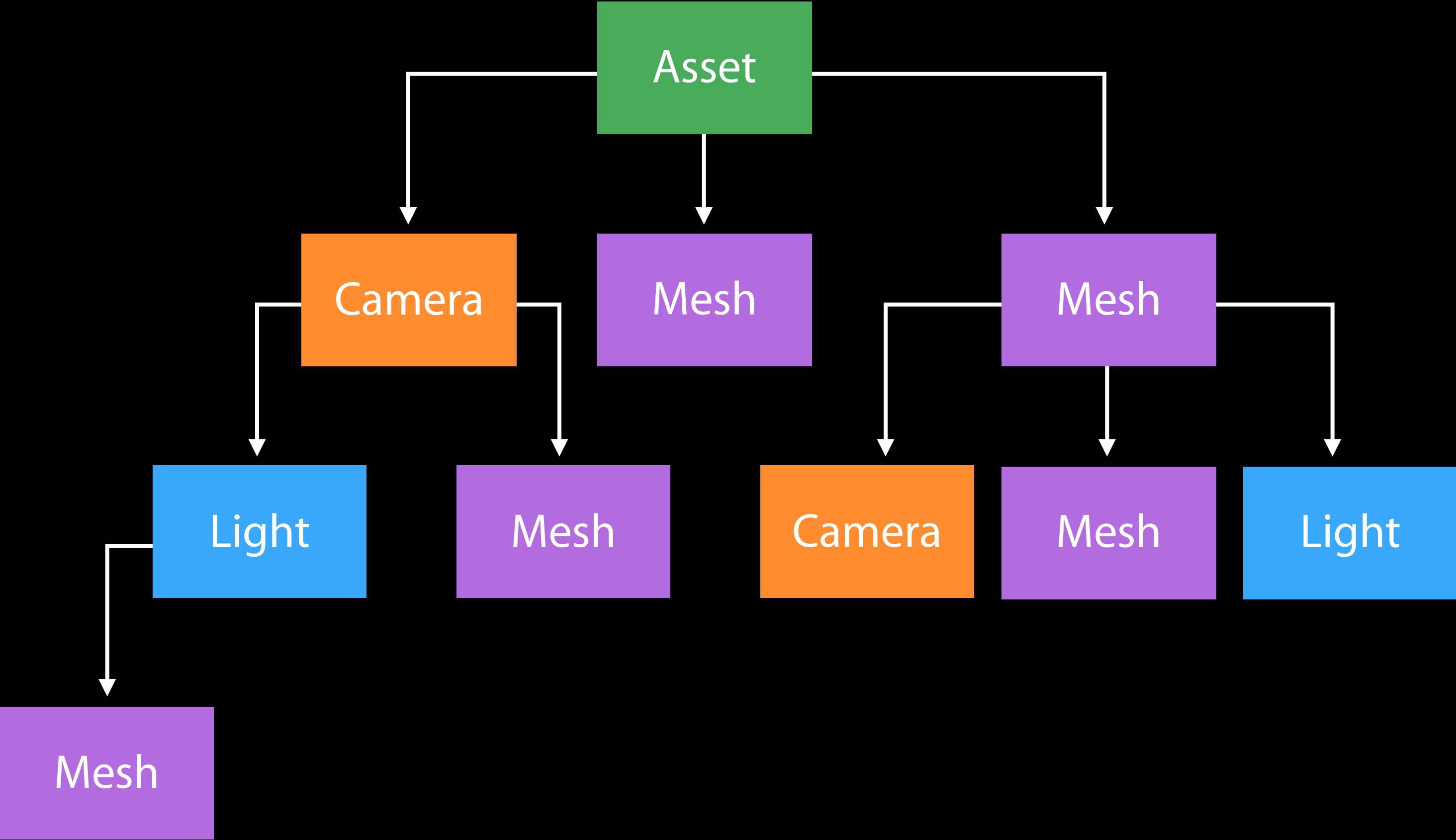




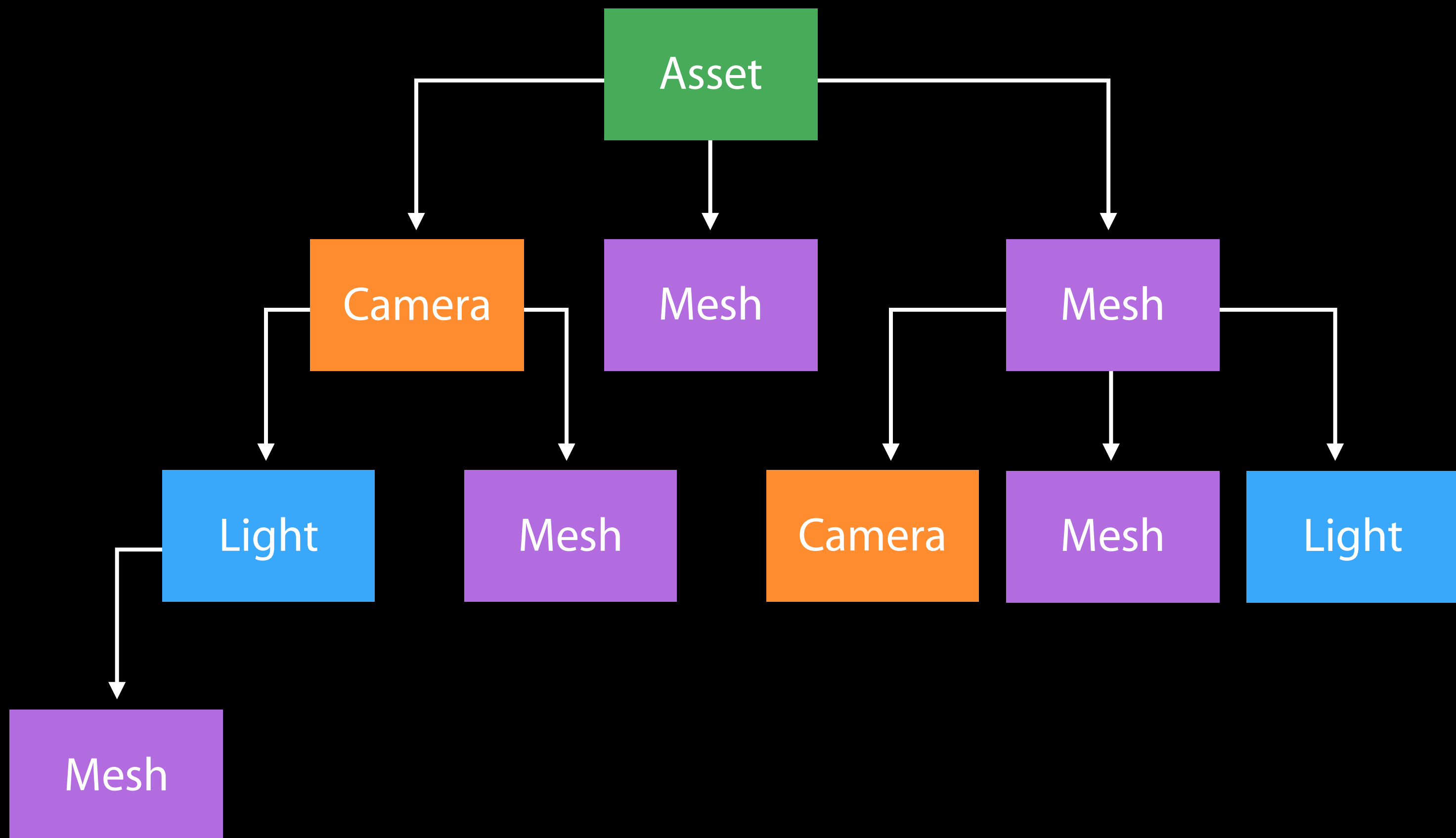




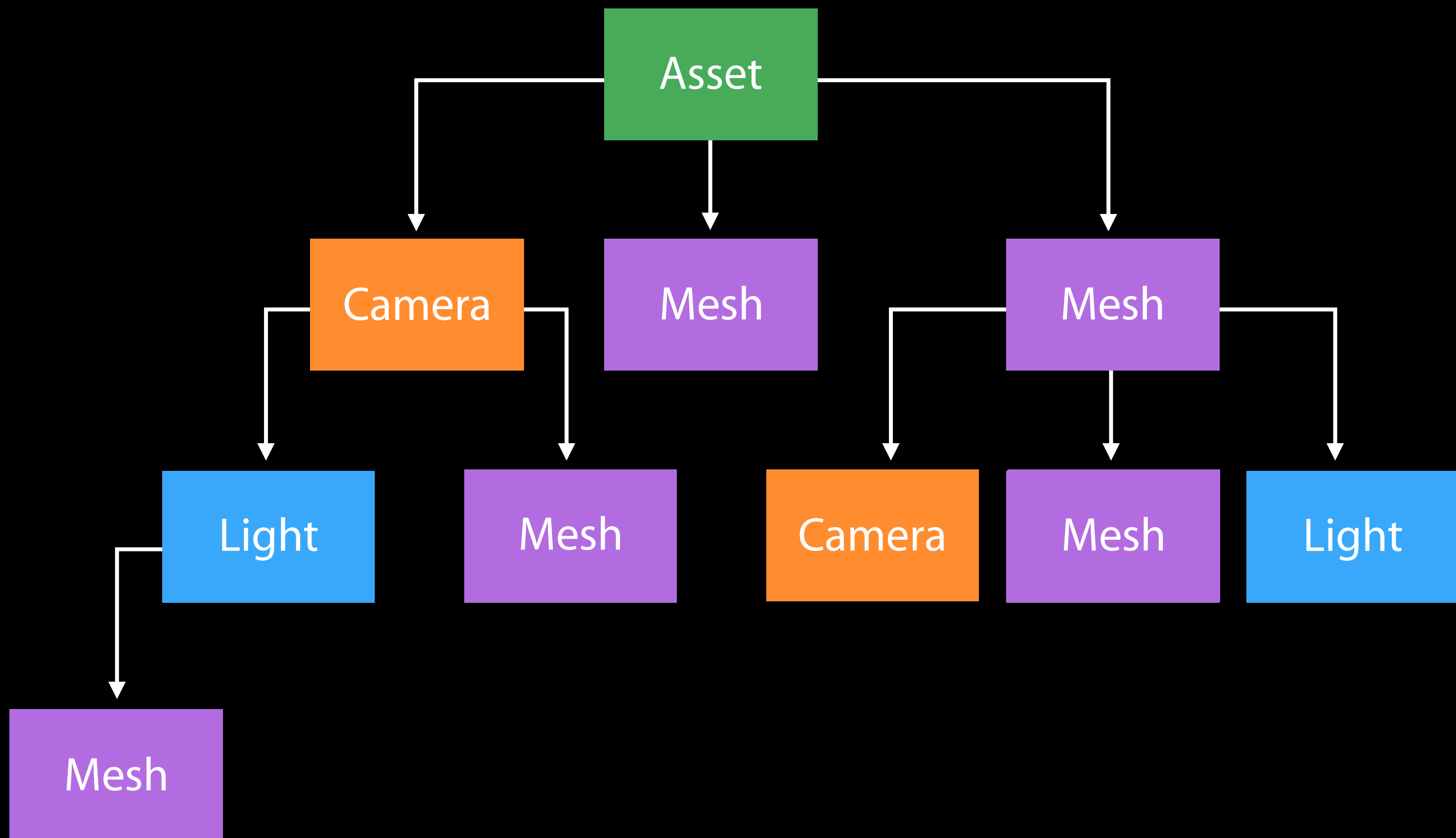
```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```



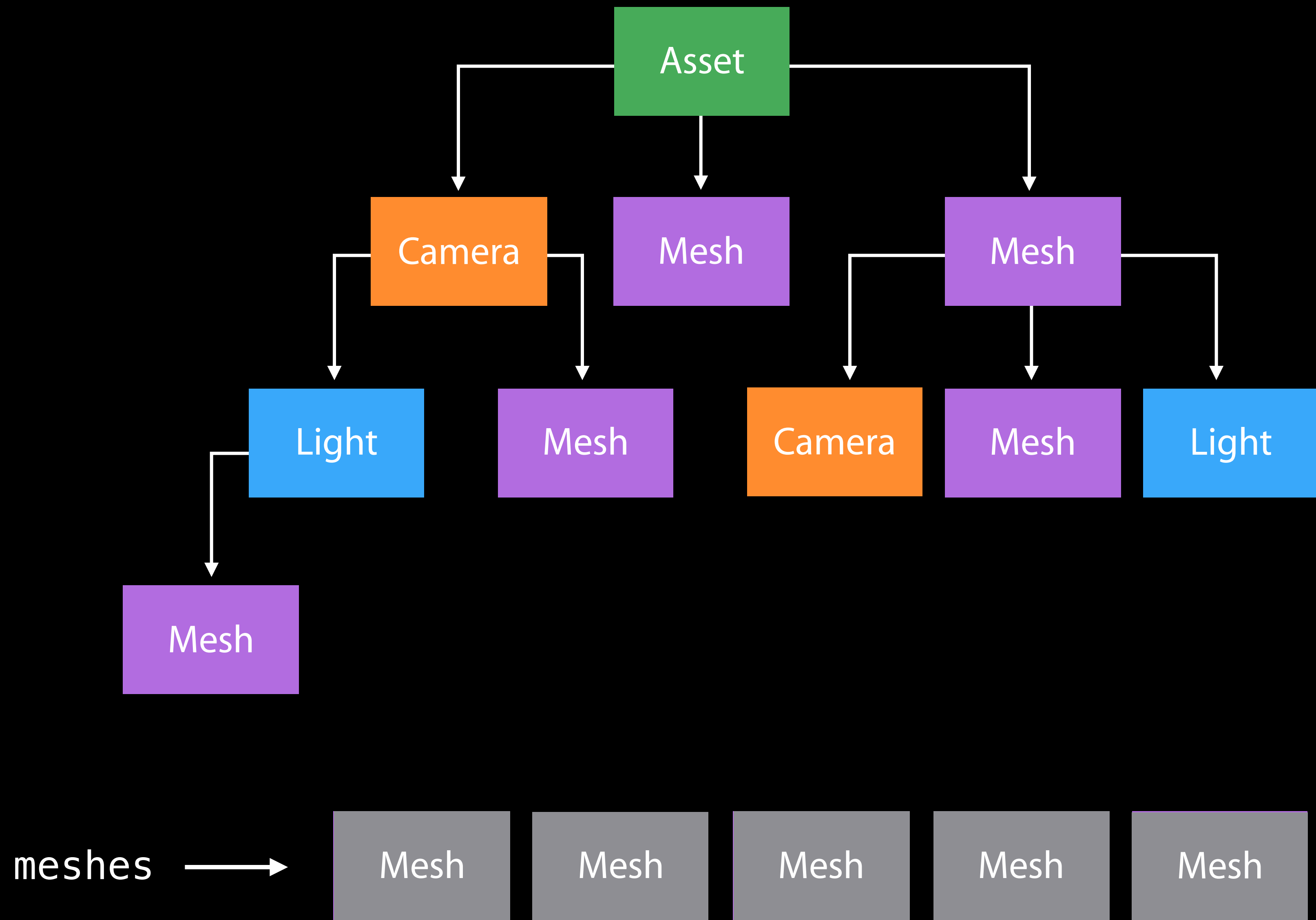

```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```



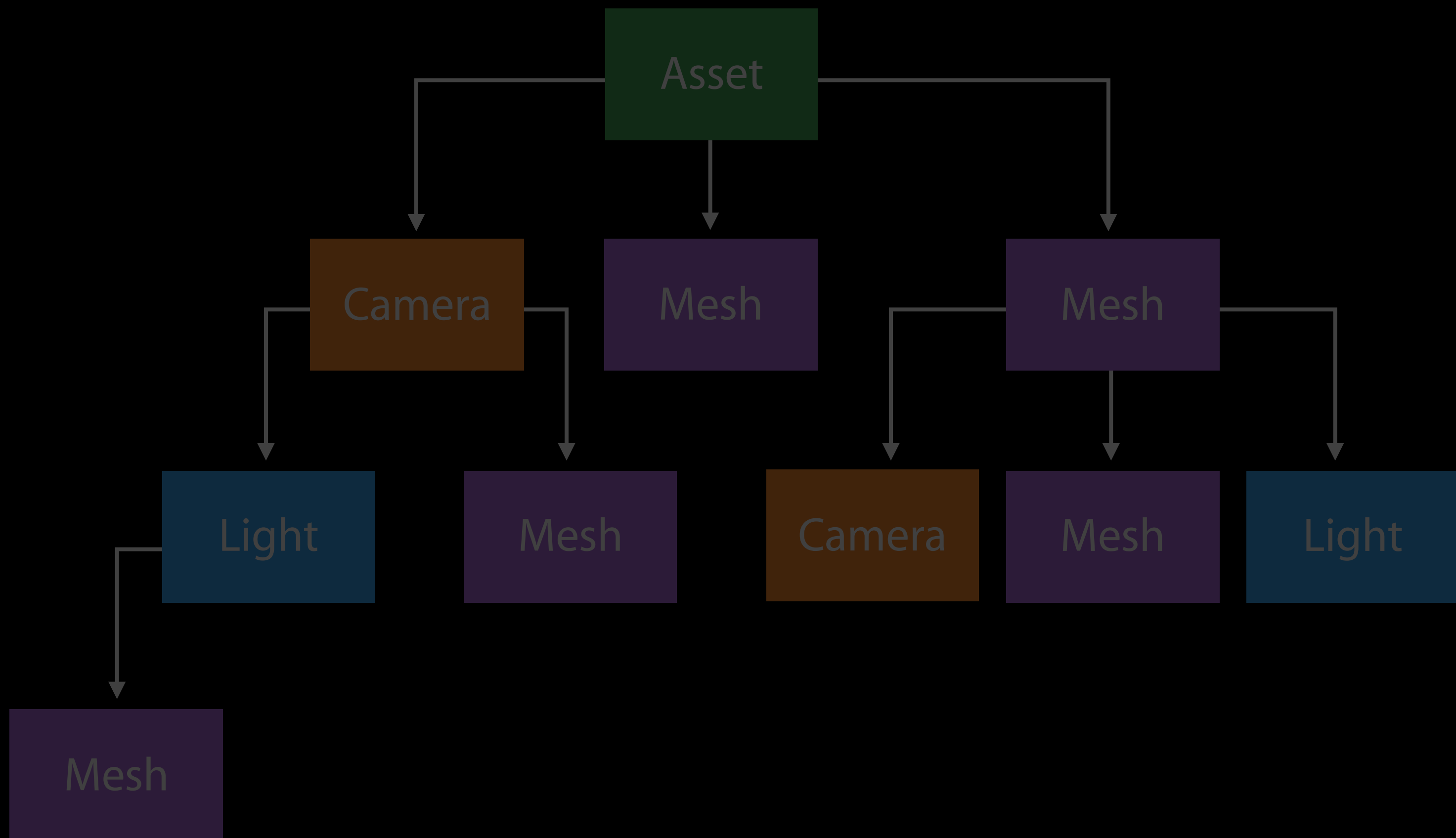
```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```



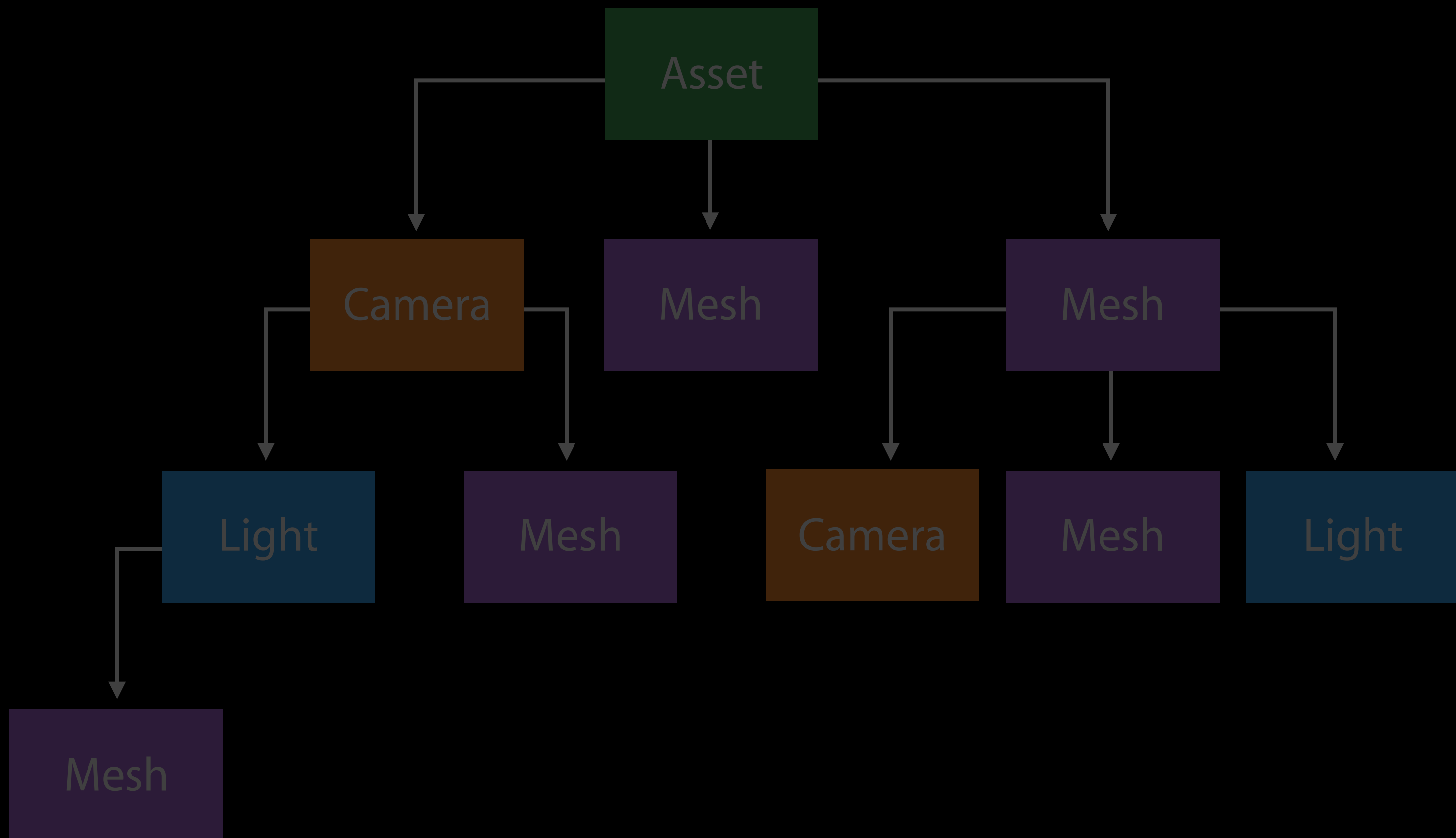
```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```



```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```

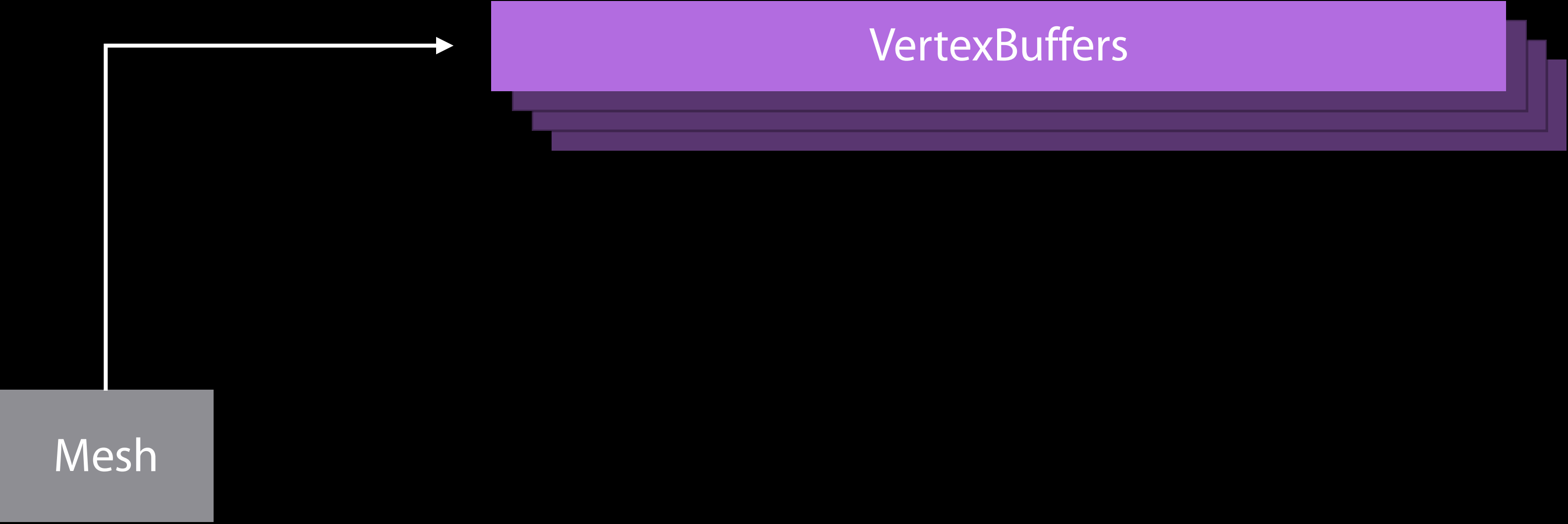


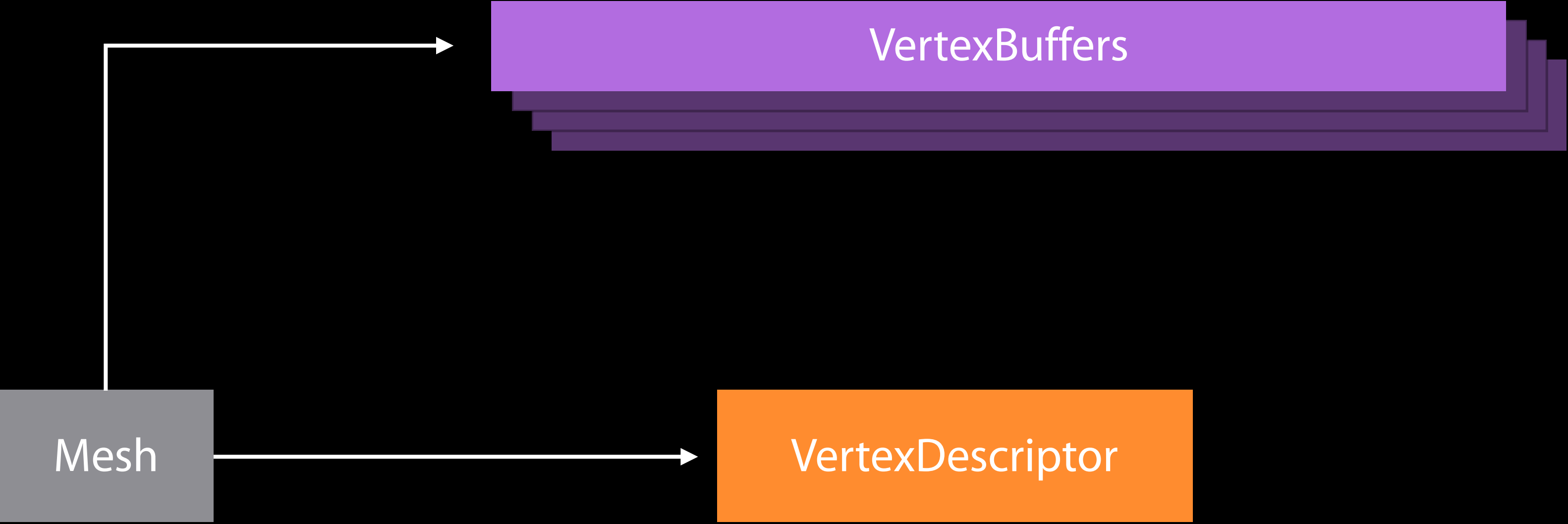
```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```

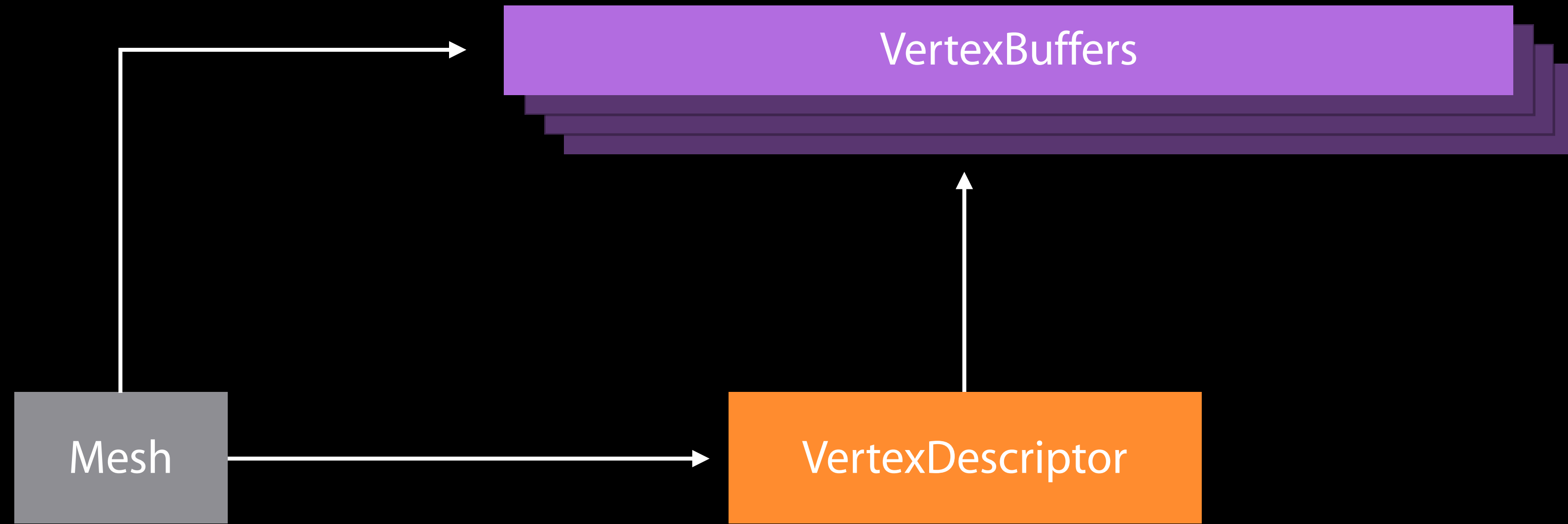


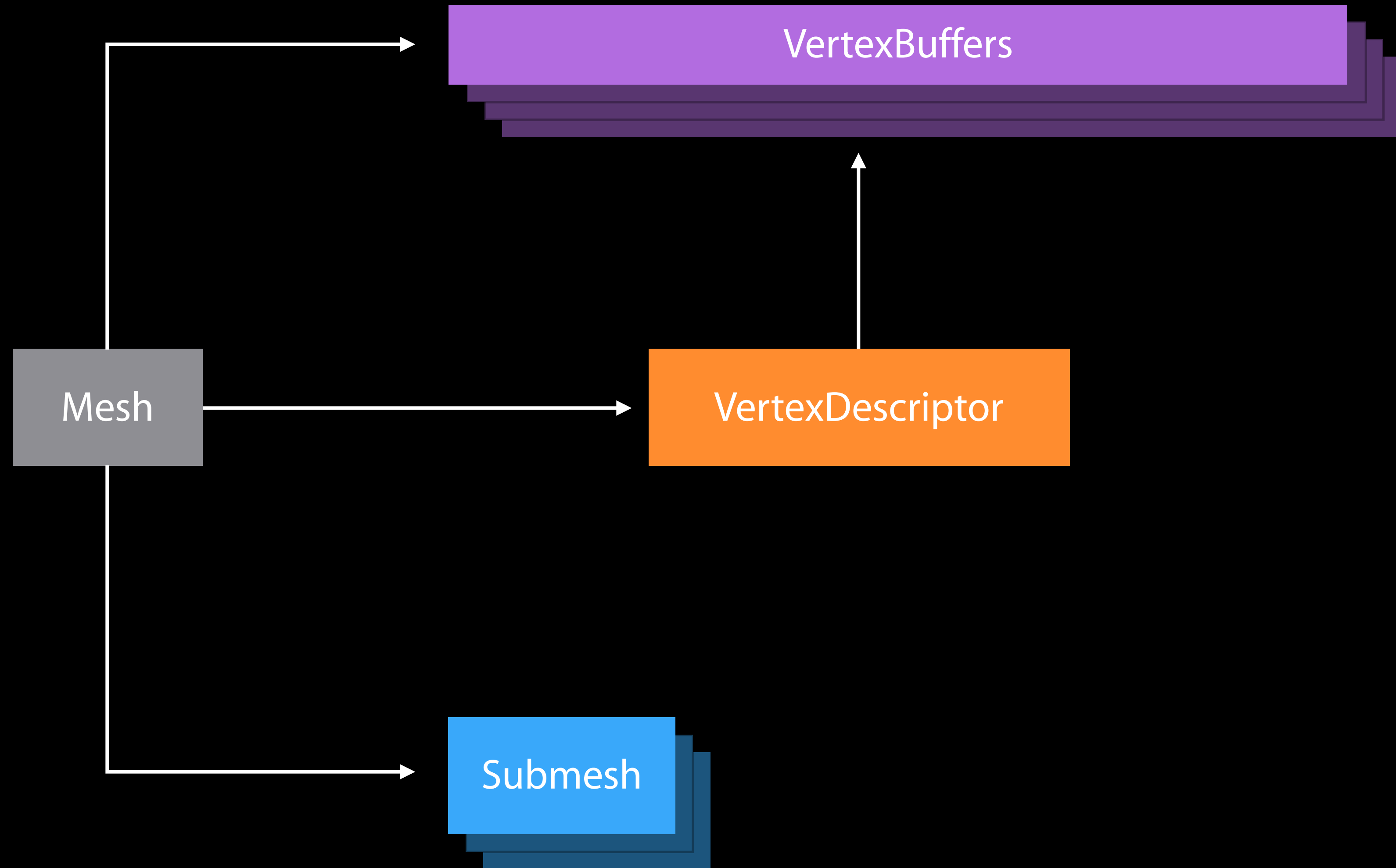
Mesh

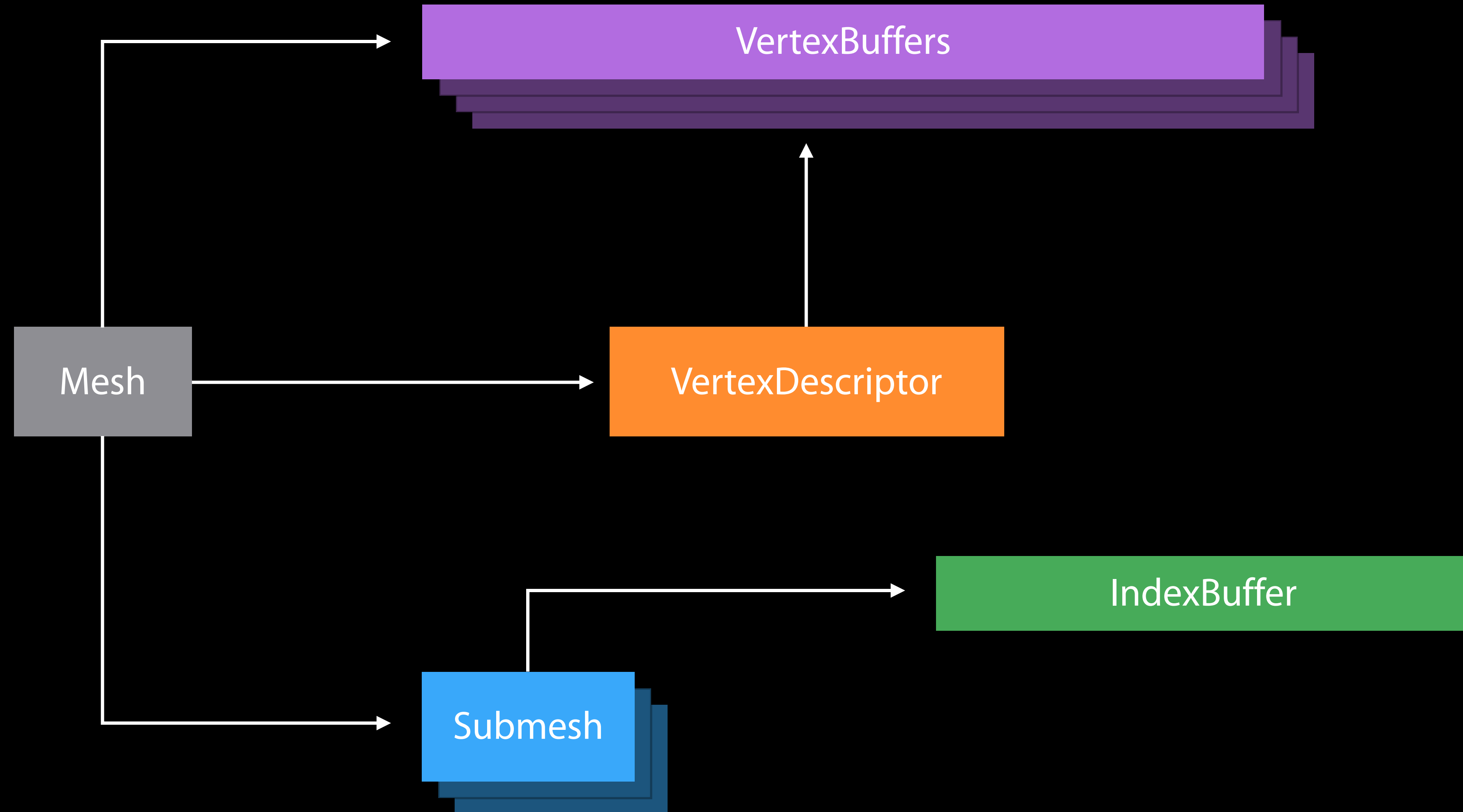
Mesh

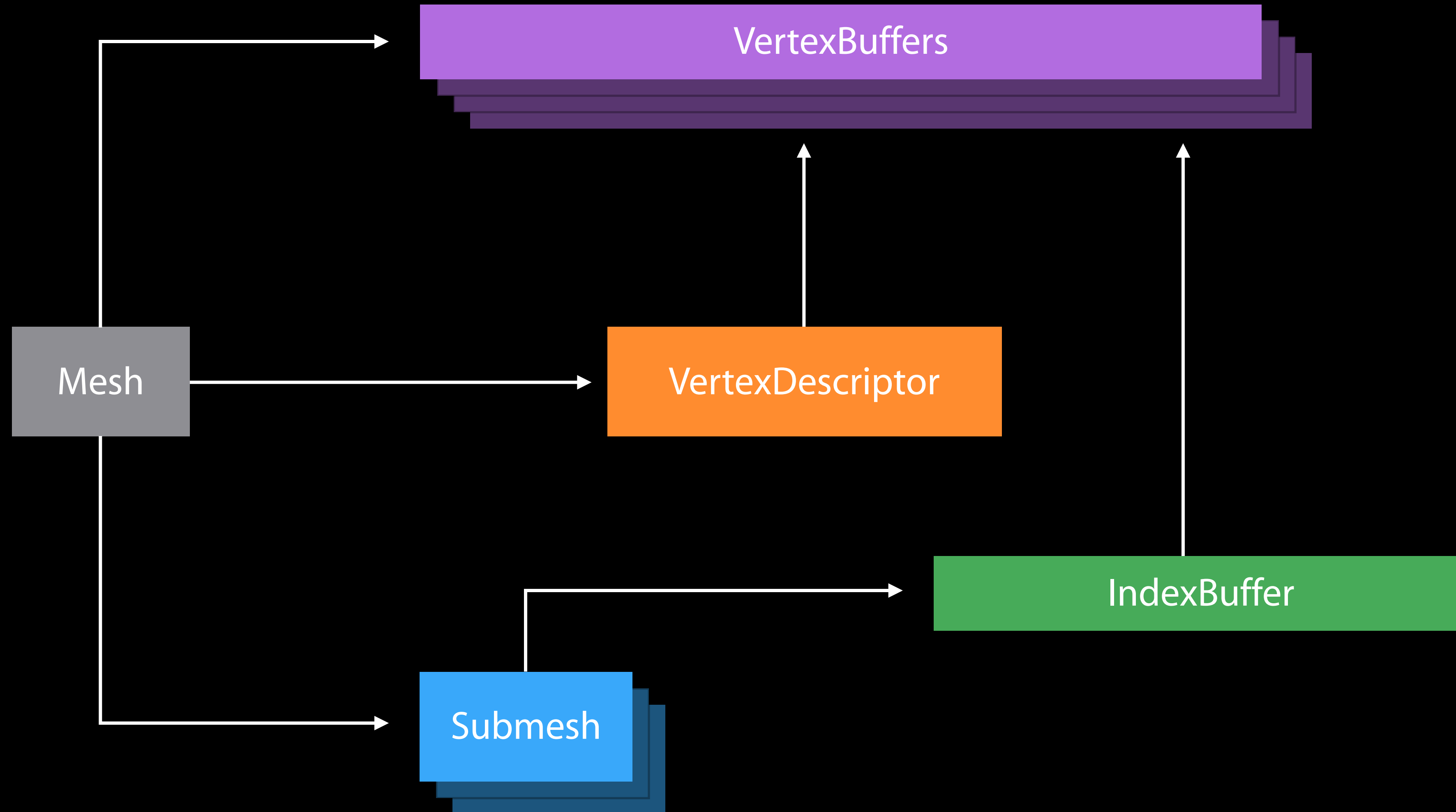


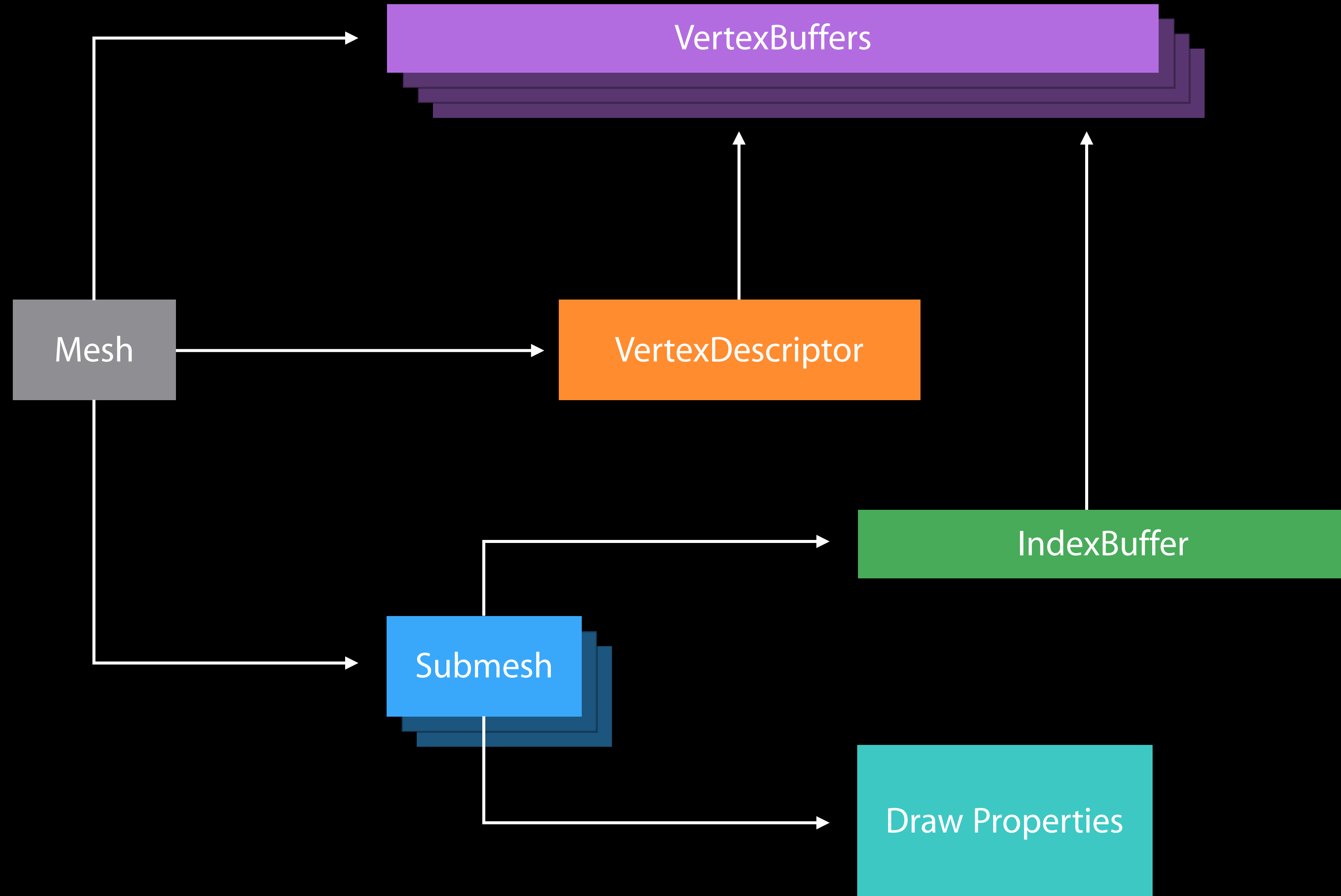


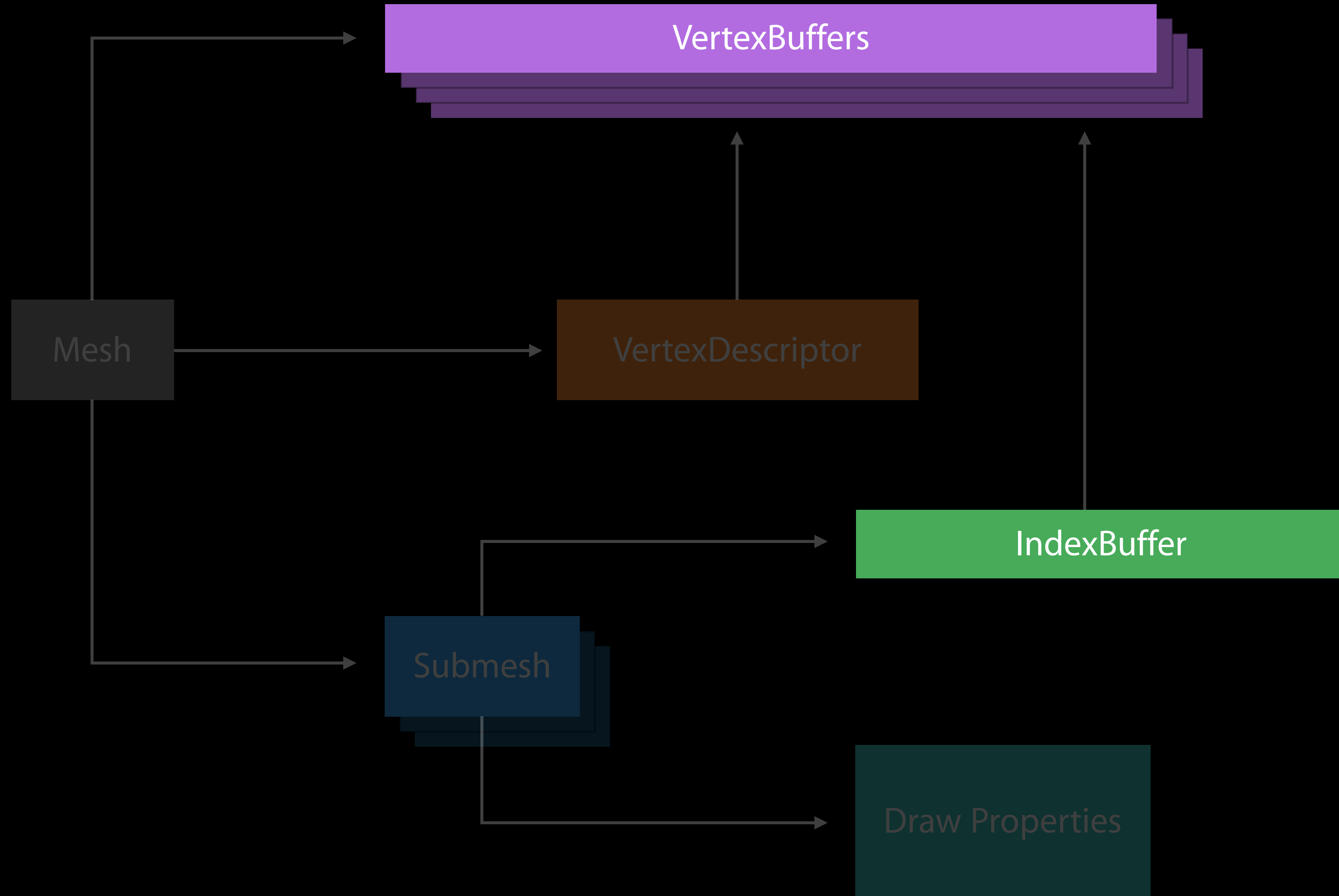






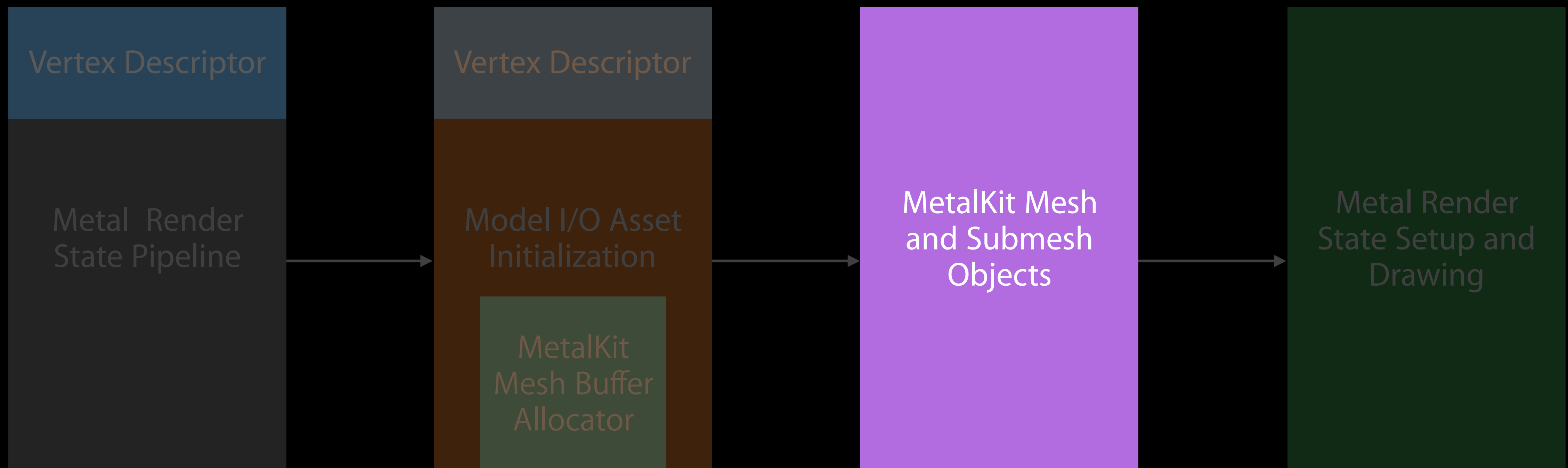






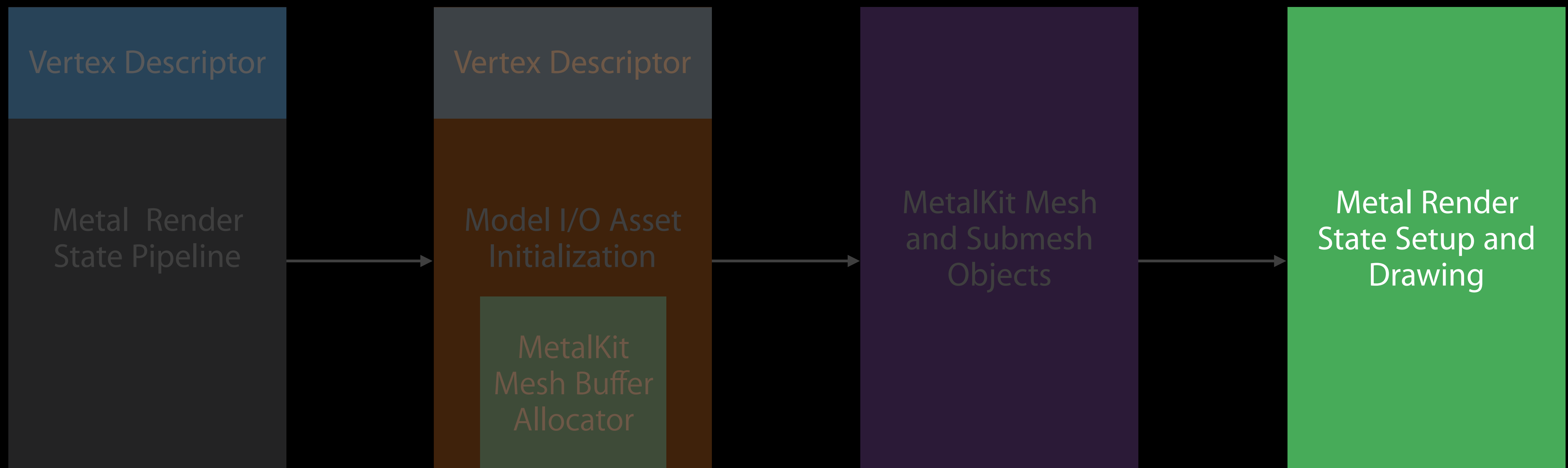
Mesh Rendering with Metal

Setup for model rendering



Mesh Rendering with Metal

Setup for model rendering



Setting Mesh State

```
NSUInteger bufferIndex = 0;
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {
    if(vertexBuffer.buffer != nil) {
        [renderEncoder setVertexBuffer:vertexBuffer.buffer
                           offset:vertexBuffer.offset
                           atIndex:bufferIndex];
    }
    bufferIndex++;
}
```

Setting Mesh State

```
NSUInteger bufferIndex = 0;  
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {  
    if(vertexBuffer.buffer != nil) {  
        [renderEncoder setVertexBuffer:vertexBuffer.buffer  
                        offset:vertexBuffer.offset  
                        atIndex:bufferIndex];  
    }  
    bufferIndex++;  
}
```

Setting Mesh State

```
NSUInteger bufferIndex = 0;
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {
    if(vertexBuffer.buffer != nil) {
        [renderEncoder setVertexBuffer:vertexBuffer.buffer
                           offset:vertexBuffer.offset
                           atIndex:bufferIndex];
    }
    bufferIndex++;
}
```

Setting Mesh State

```
NSUInteger bufferIndex = 0;
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {
    if(vertexBuffer.buffer != nil) {
        [renderEncoder setVertexBuffer:vertexBuffer.buffer
                           offset:vertexBuffer.offset
                           atIndex:bufferIndex];
    }
    bufferIndex++;
}
```

Setting Mesh State

```
NSUInteger bufferIndex = 0;
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {
    if(vertexBuffer.buffer != nil) {
        [renderEncoder setVertexBuffer:vertexBuffer.buffer
                                offset:vertexBuffer.offset
                                atIndex:bufferIndex];
    }
    bufferIndex++;
}
```

Setting Mesh State

```
NSUInteger bufferIndex = 0;
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {
    if(vertexBuffer.buffer != nil) {
        [renderEncoder setVertexBuffer:vertexBuffer.buffer
                           offset:vertexBuffer.offset
                           atIndex:bufferIndex];
    }
    bufferIndex++;
}
```

Rendering the Mesh

```
for(MTKSubmesh *submesh in mesh.submeshes)
{
    [renderEncoder drawIndexedPrimitives:submesh.primitiveType
                    indexCount:submesh.indexCount
                    indexType:submesh.indexType
                    indexBuffer:submesh.indexBuffer.buffer
                    indexBufferOffset:submesh.indexBuffer.offset];
}
```


Rendering the Mesh

```
for(MTKSubmesh *submesh in mesh.submeshes)
{
    [renderEncoder drawIndexedPrimitives:submesh.primitiveType
                    indexCount:submesh.indexCount
                    indexType:submesh.indexType
                    indexBuffer:submesh.indexBuffer.buffer
                    indexBufferOffset:submesh.indexBuffer.offset];
}
```

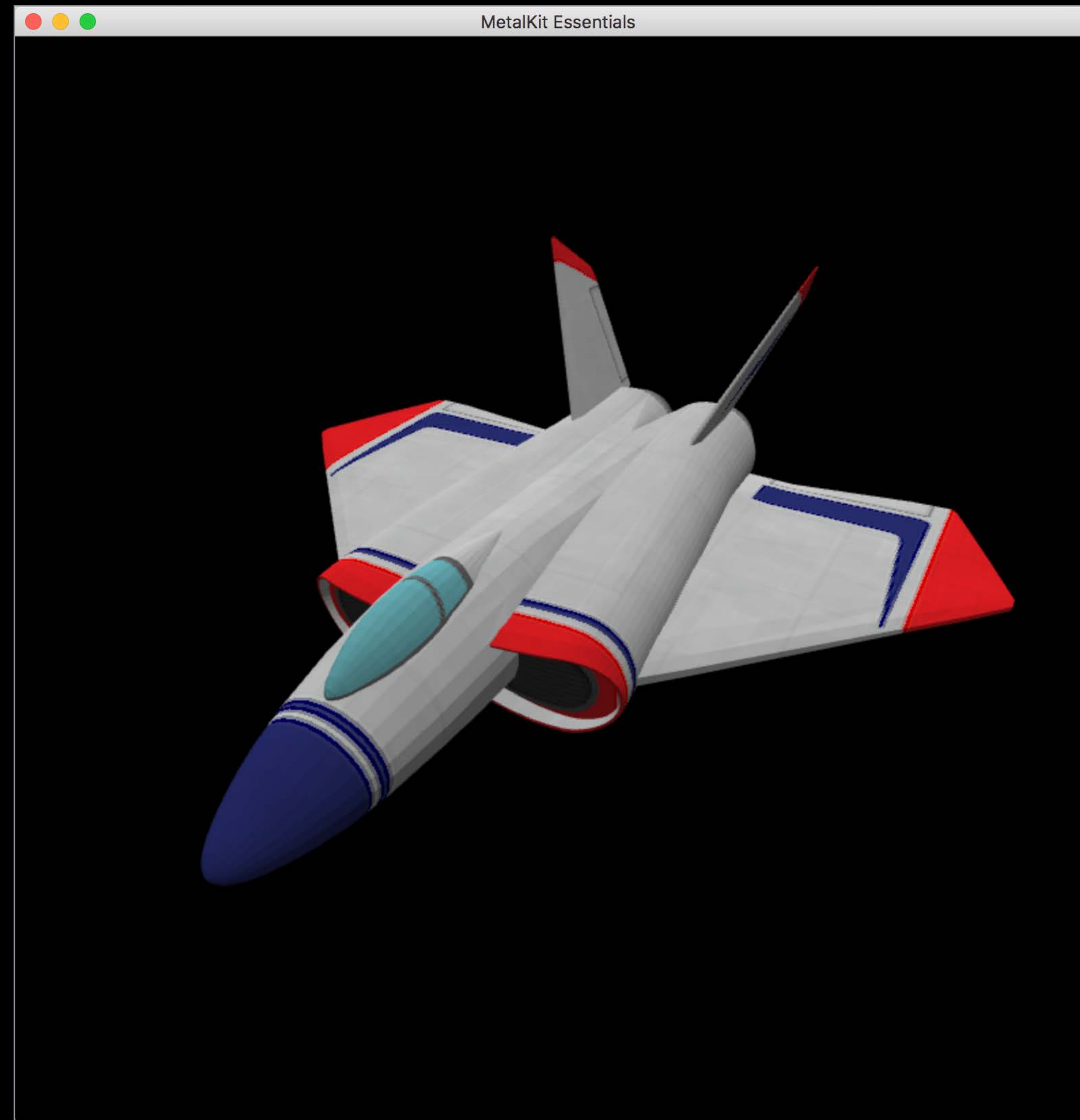
Rendering the Mesh

```
for(MTKSubmesh *submesh in mesh.submeshes)
{
    [renderEncoder drawIndexedPrimitives:submesh.primitiveType
                    indexCount:submesh.indexCount
                    indexType:submesh.indexType
                    indexBuffer:submesh.indexBuffer.buffer
                    indexBufferOffset:submesh.indexBuffer.offset];
}
```

Rendering the Mesh

```
for(MTKSubmesh *submesh in mesh.submeshes)
{
    [renderEncoder drawIndexedPrimitives:submesh.primitiveType
                                indexCount:submesh.indexCount
                                indexType:submesh.indexType
                                indexBuffer:submesh.indexBuffer.buffer
                                indexBufferOffset:submesh.indexBuffer.offset];
}
```

MetalKit Essentials



Metal Performance Shaders

Anna Tikhonova GPU Software Frameworks Engineer

Metal Performance Shaders

NEW

Introduction

A framework of data-parallel algorithms for the GPU

CPU-style library for the GPU

Metal Performance Shaders

NEW

Introduction

Optimized for iOS

Available in iOS 9 for the A8 processor



iPad Air 2

iPhone 6 Plus

iPhone 6

Metal Performance Shaders

NEW

Introduction

Designed to integrate easily into your Metal applications

As simple as calling a library function



Metal Performance Shaders

NEW

Supported image operators

Histogram, Equalization, and Specification

Morphology—Min, Max, Dilate, and Erode

Lanczos Resampling

Median

Thresholding

Integral

Convolution—General, Gaussian Blur, Box, Tent, and Sobel

Metal Performance Shaders

NEW

Supported image operators

Histogram, Equalization, and Specification

Morphology—Min, Max, Dilate, and Erode

Lanczos Resampling

Median

Thresholding

Integral

Convolution—General, Gaussian Blur, Box, Tent, and Sobel

Equalization



Equalization



Metal Performance Shaders

Supported image operators

Histogram, Equalization and Specification

Morphology — Min, Max, Dilate, and Erode

Lanczos Resampling

Median

Thresholding

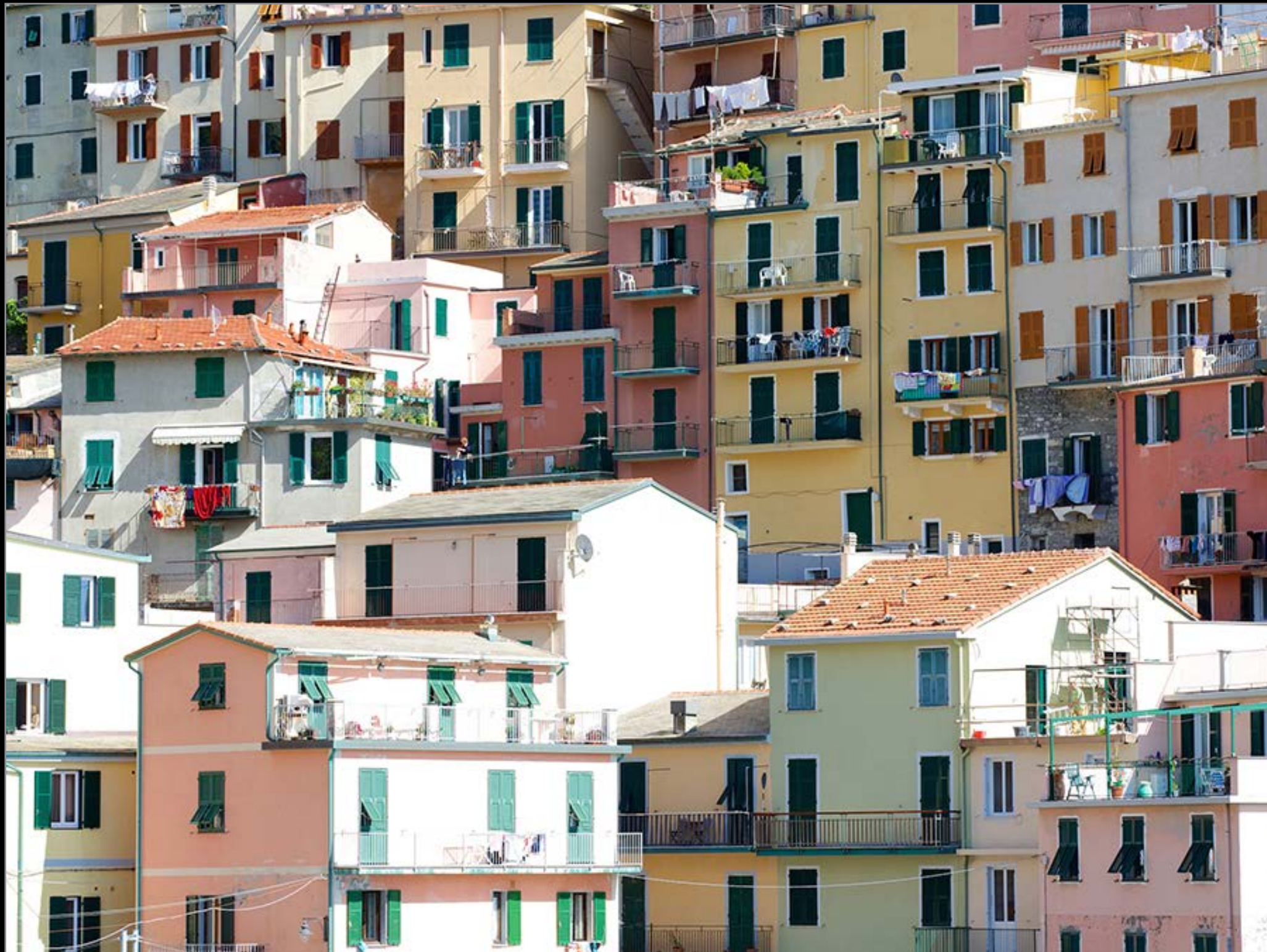
Integral

Convolution — General, Gaussian Blur, Box, Tent, and Sobel

Lanczos Resampling



Lanczos Resampling



Metal Performance Shaders

Supported image operators

Histogram, Equalization and Specification

Morphology — Min, Max, Dilate, and Erode

Lanczos Resampling

Median

Thresholding

Integral

Convolution — General, Gaussian Blur, Box, Tent, and Sobel

Thresholding



Thresholding



Thresholding and Sobel



Thresholding and Sobel



Metal Performance Shaders

Supported image operators

Histogram, Equalization and Specification

Morphology — Min, Max, Dilate, and Erode

Lanczos Resampling

Median

Thresholding

Integral

Convolution — General, Gaussian Blur, Box, Tent, and Sobel

Gaussian Blur



Gaussian Blur



Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
```

```
MPSImageGaussianBlur *blurFilter =  
    [[MPSImageGaussianBlur alloc]  
     initWithDevice: device sigma: 3];
```

```
// Encode filter to the command buffer
```

```
[blurFilter encodeToCommandBuffer: commandBuffer  
    source: sourceTexture  
    destination: destinationTexture];
```


Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
MPSImageGaussianBlur *blurFilter =
    [[MPSImageGaussianBlur alloc]
     initWithDevice: device sigma: 3];
```

```
// Encode filter to the command buffer
[blurFilter encodeToCommandBuffer: commandBuffer
 source: sourceTexture
 destination: destinationTexture];
```

Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
```

```
MPSImageGaussianBlur *blurFilter =  
    [[MPSImageGaussianBlur alloc]  
     initWithDevice: device sigma: 3];
```

```
// Encode filter to the command buffer
```

```
[blurFilter encodeToCommandBuffer: commandBuffer  
    source: sourceTexture  
    destination: destinationTexture];
```

Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
```

```
MPSImageGaussianBlur *blurFilter =  
    [[MPSImageGaussianBlur alloc]  
     initWithDevice: device sigma: 3];
```

```
// Encode filter to the command buffer
```

```
[blurFilter encodeToCommandBuffer: commandBuffer  
    source: sourceTexture  
    destination: destinationTexture];
```

Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
MPSImageGaussianBlur *blurFilter =
    [[MPSImageGaussianBlur alloc]
     initWithDevice: device sigma: 3];

// Encode filter to the command buffer
[blurFilter encodeToCommandBuffer: commandBuffer
    source: sourceTexture
    destination: destinationTexture];
```

Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
MPSImageGaussianBlur *blurFilter =
    [[MPSImageGaussianBlur alloc]
     initWithDevice: device sigma: 3];

// Encode filter to the command buffer
[blurFilter encodeToCommandBuffer: commandBuffer
 source: sourceTexture
 destination: destinationTexture];
```

Command Buffer

Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
MPSImageGaussianBlur *blurFilter =
    [[MPSImageGaussianBlur alloc]
     initWithDevice: device sigma: 3];

// Encode filter to the command buffer
[blurFilter encodeToCommandBuffer: commandBuffer
 source: sourceTexture
 destination: destinationTexture];
```

Command Buffer

Draw

Draw

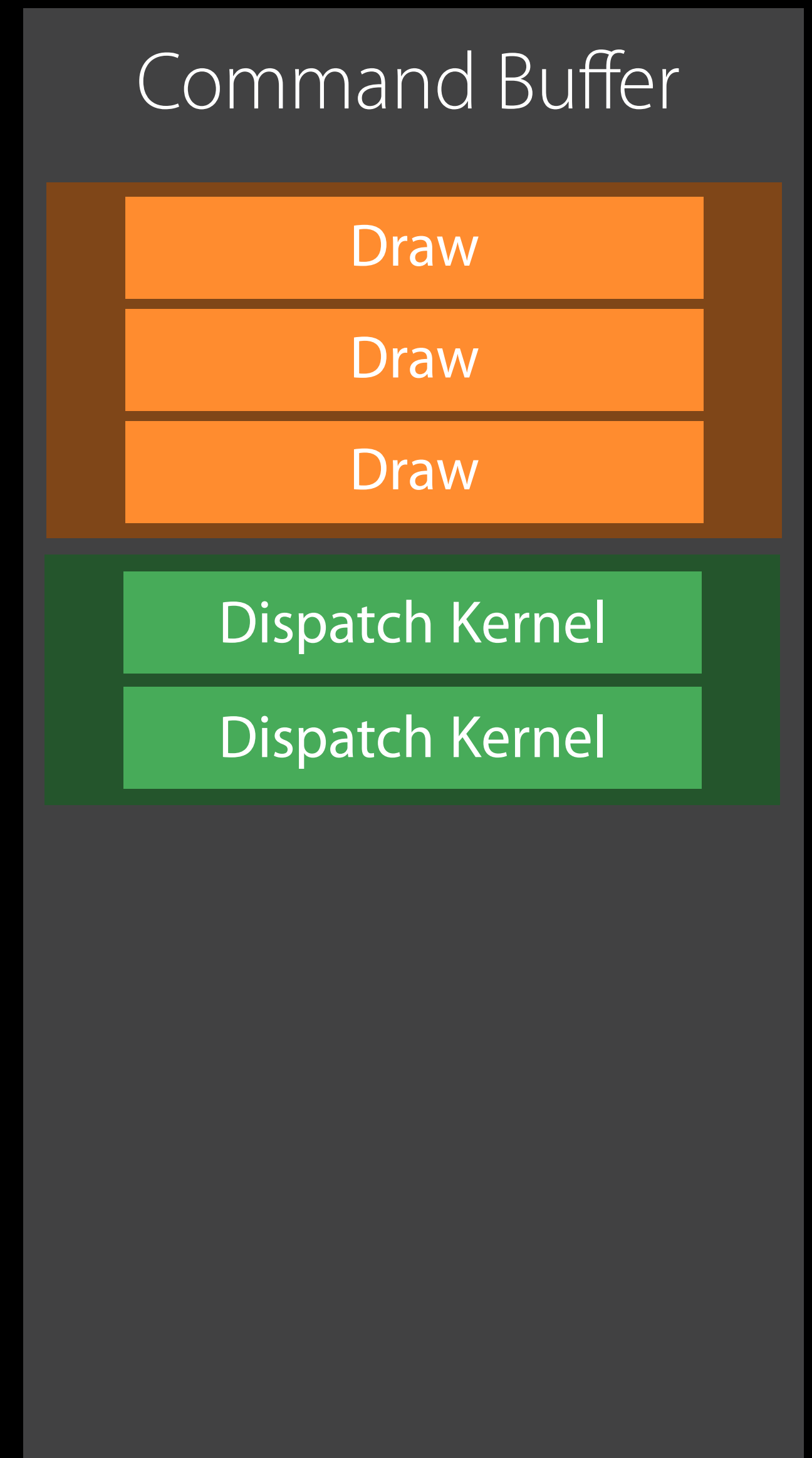
Draw

Using Metal Performance Shaders

Use a 'blur' filter

```
// Create a filter object
MPSImageGaussianBlur *blurFilter =
    [[MPSImageGaussianBlur alloc]
     initWithDevice: device sigma: 3];

// Encode filter to the command buffer
[blurFilter encodeToCommandBuffer: commandBuffer
 source: sourceTexture
 destination: destinationTexture];
```



Using Metal Performance Shaders

Use a 'blur' filter

// Create a filter object

```
MPSImageGaussianBlur *blurFilter =  
    [[MPSImageGaussianBlur alloc]  
     initWithDevice: device sigma: 3];
```

// Encode filter to the command buffer

```
[blurFilter encodeToCommandBuffer: commandBuffer  
    source: sourceTexture  
    destination: destinationTexture];
```

Command Buffer

Draw

Draw

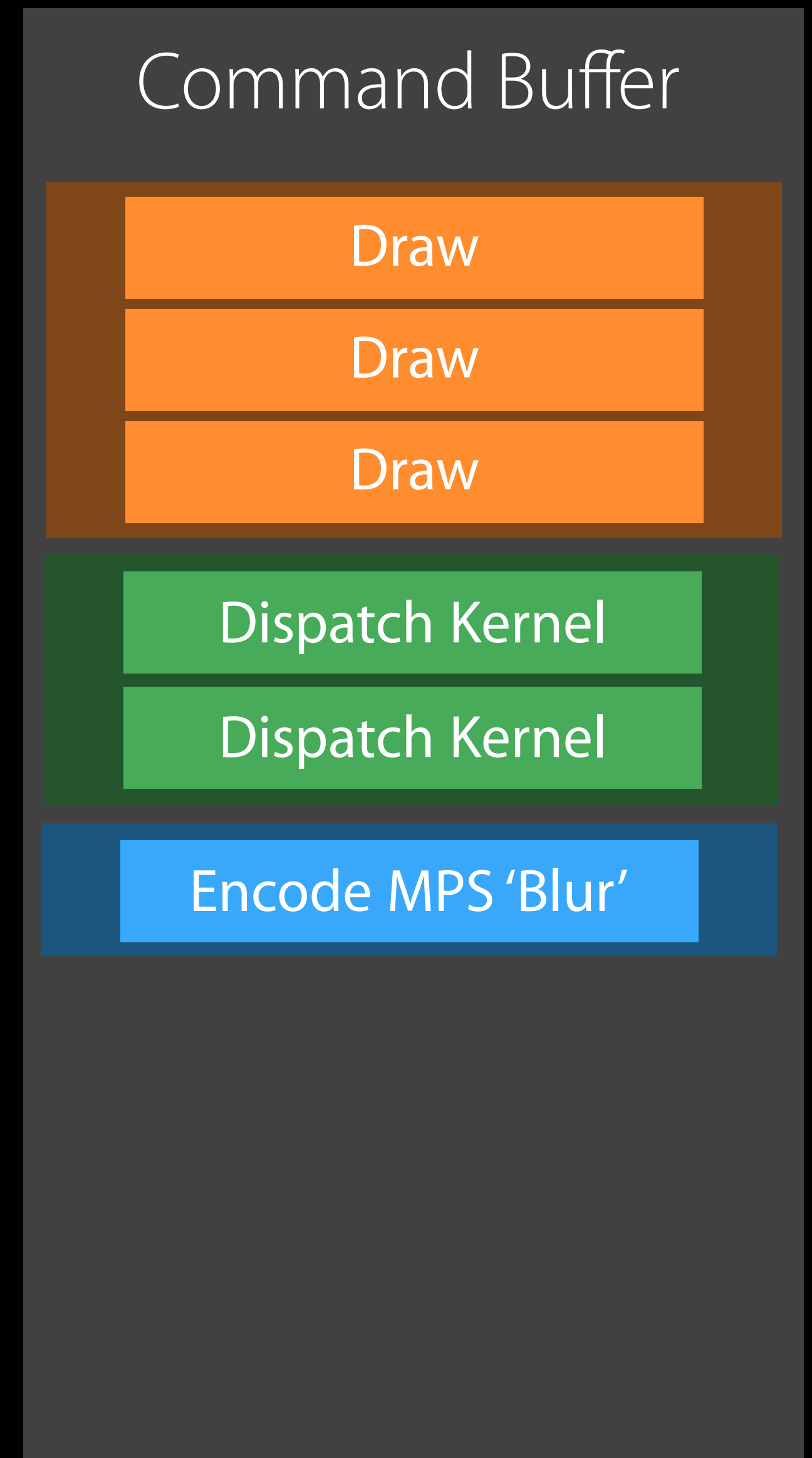
Draw

Dispatch Kernel

Dispatch Kernel

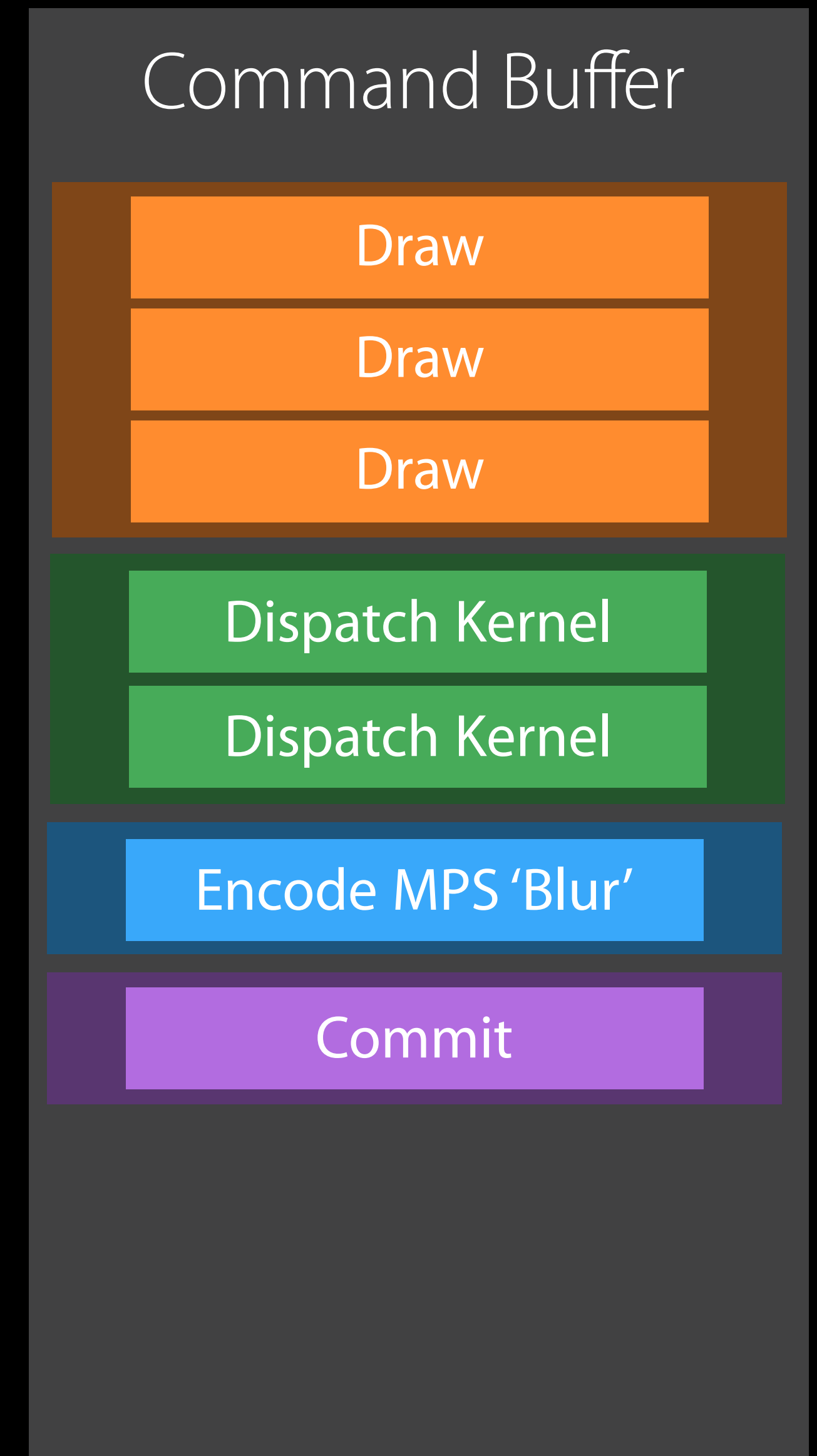
Using Metal Performance Shaders

Use a 'blur' filter



Using Metal Performance Shaders

Use a 'blur' filter



Download Sample Code

[https://developer.apple.com/library/prerelease/ios/samplecode/
MetalPerformanceShadersHelloWorld](https://developer.apple.com/library/prerelease/ios/samplecode/MetalPerformanceShadersHelloWorld)

Performance

Behind the scenes

Choose the right algorithm

Tune for

- Kernel radius
- Pixel format
- Memory hierarchy
- Number of pixels per thread
- Threadgroup dimensions

CPU optimizations

Performance

Behind the scenes

Choose the right algorithm

Tune for

- Kernel radius
- Pixel format
- Memory hierarchy
- Number of pixels per thread
- Threadgroup dimensions

CPU optimizations

Performance

Behind the scenes

Choose the right algorithm

Tune for

- Kernel radius
- Pixel format
- Memory hierarchy
- Number of pixels per thread
- Threadgroup dimensions

CPU optimizations

Performance

Behind the scenes

Choose the right algorithm

Tune for

- Kernel radius
- Pixel format
- Memory hierarchy
- Number of pixels per thread
- Threadgroup dimensions

CPU optimizations

Performance

Behind the scenes

Choose the right algorithm

Tune for

- Kernel radius
- Pixel format
- Memory hierarchy
- Number of pixels per thread
- Threadgroup dimensions

CPU optimizations



Optimized Gaussian Blur

Statistics

Optimized Gaussian Blur

Statistics

49

Metal kernels

2000

Lines of kernel code

821

Different Gaussian
filter implementations

Demo

Filter Performance

Not capped by screen refresh rate

Smaller is better

16.6 ms = 60 FPS



A Few More Details

Source 'offset' and Destination 'clipRect'

Filter properties



Source



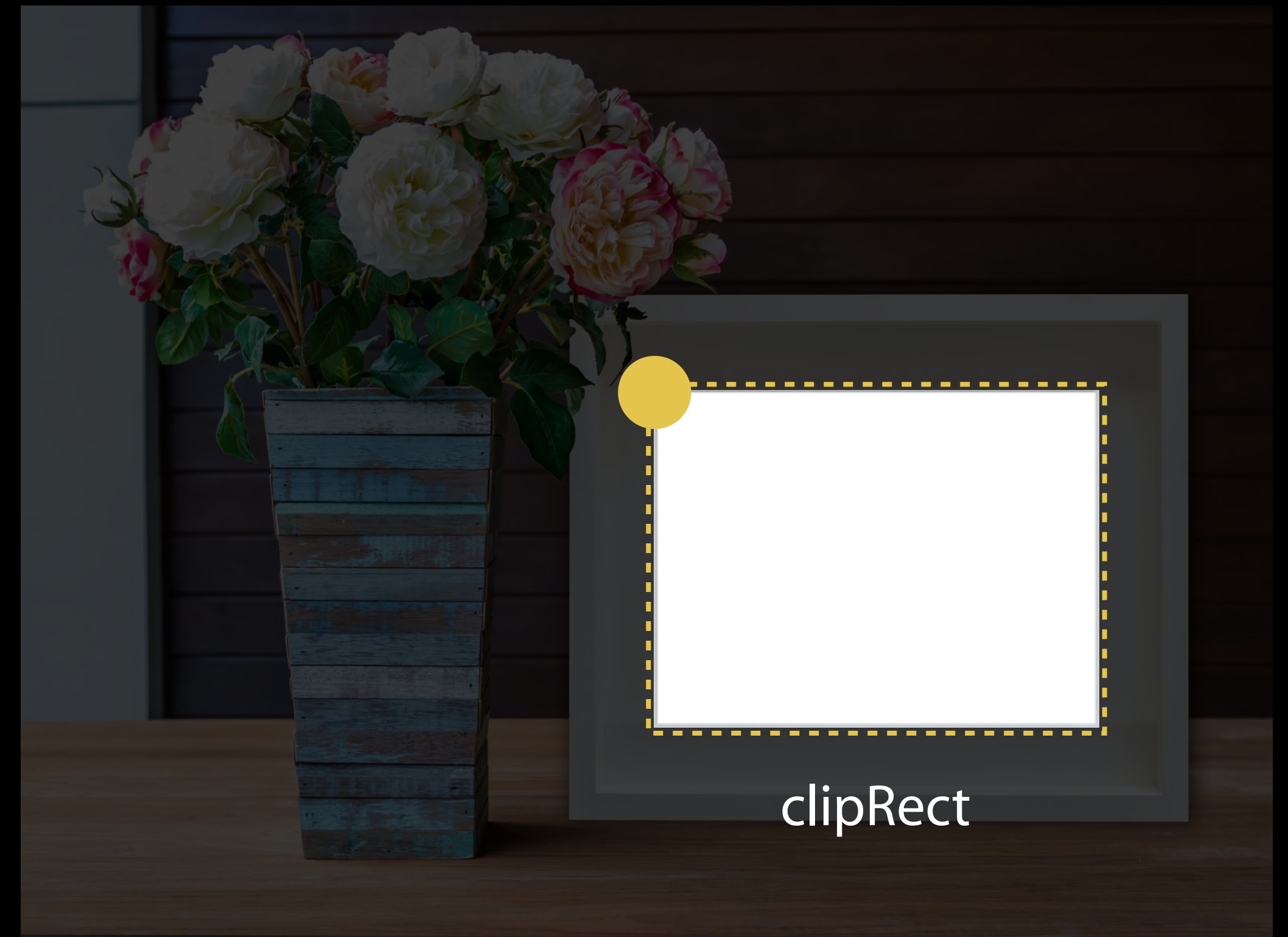
Destination

Source 'offset' and Destination 'clipRect'

Filter properties



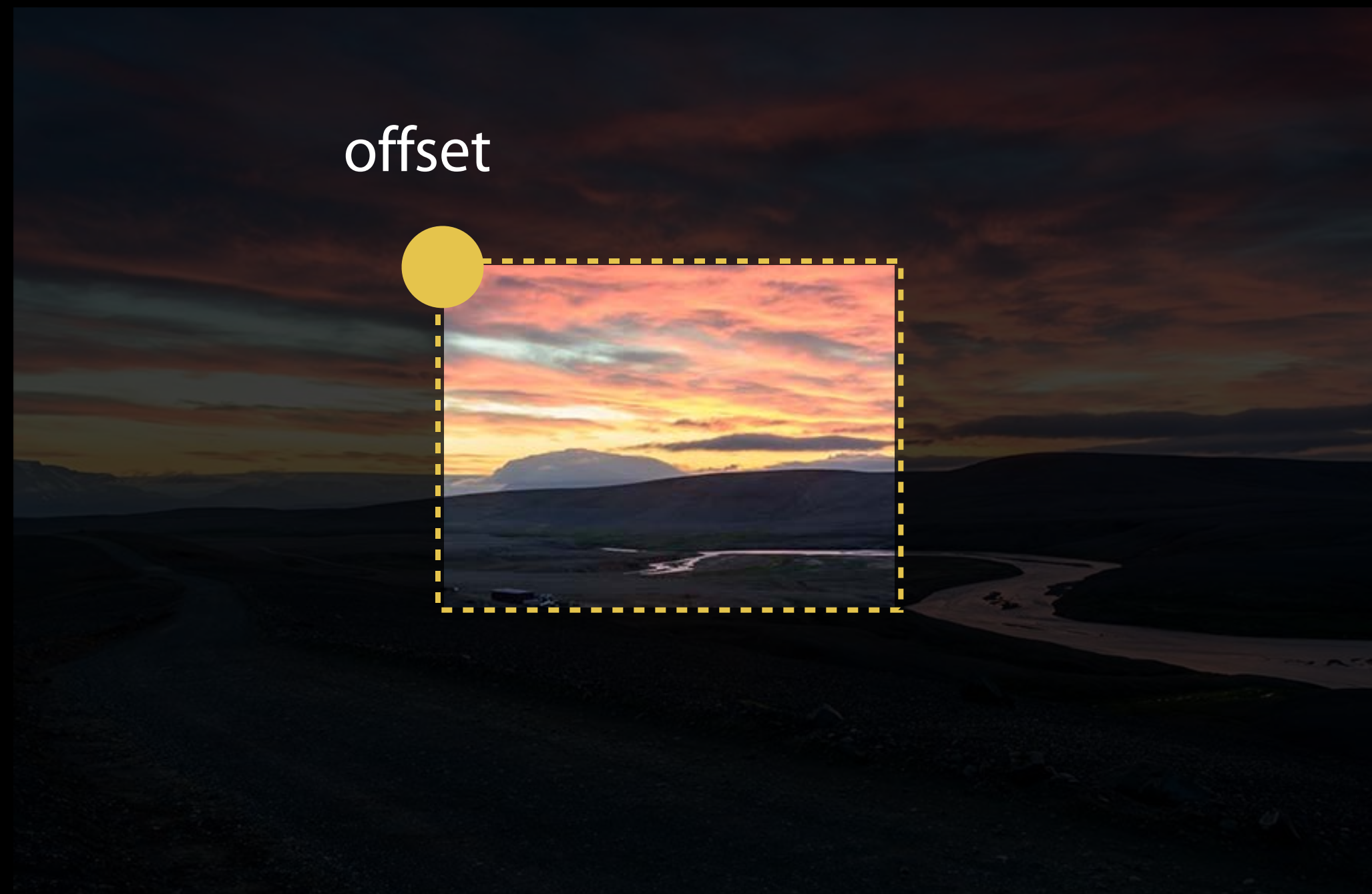
Source



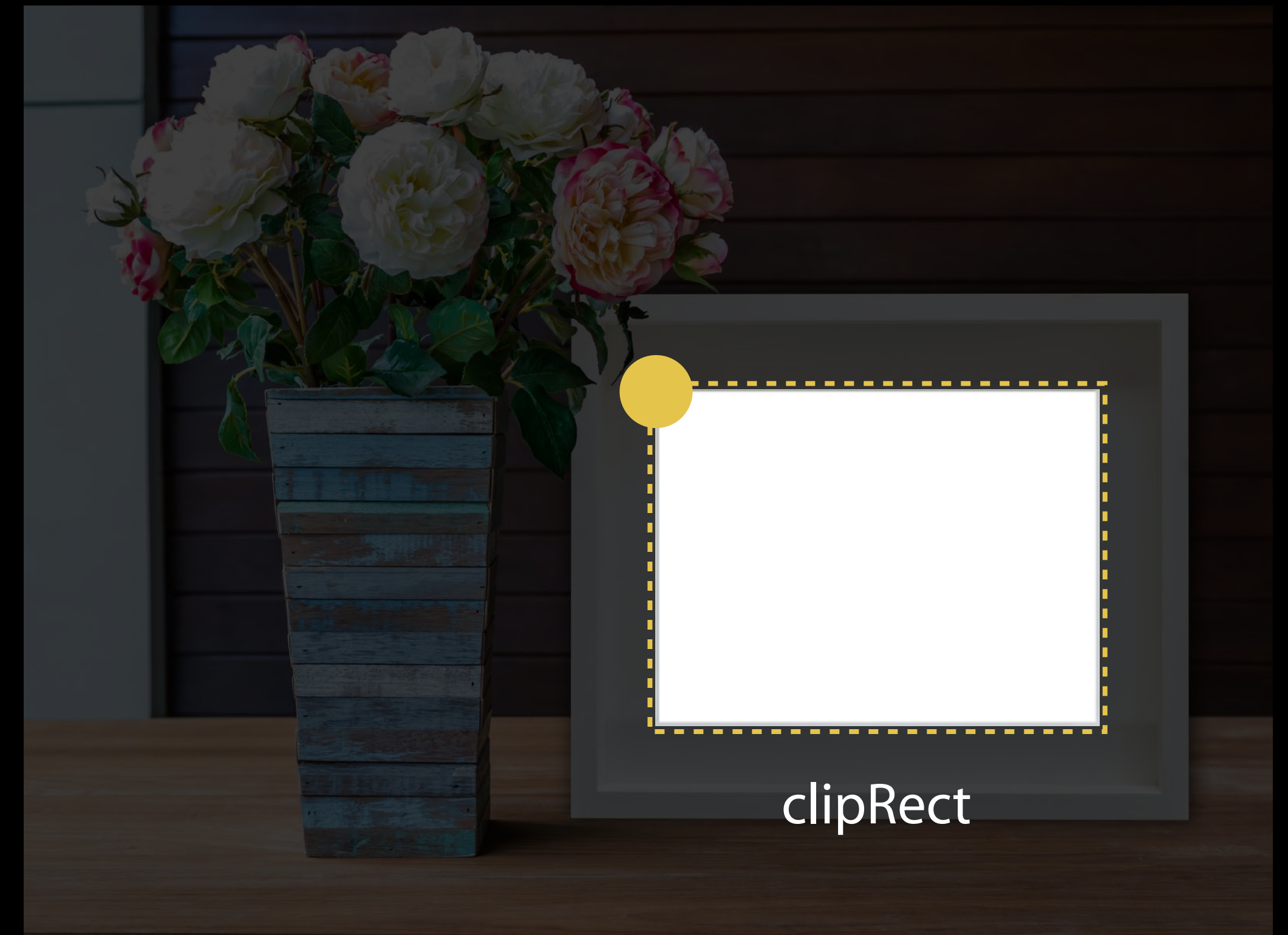
Destination

Source 'offset' and Destination 'clipRect'

Filter properties



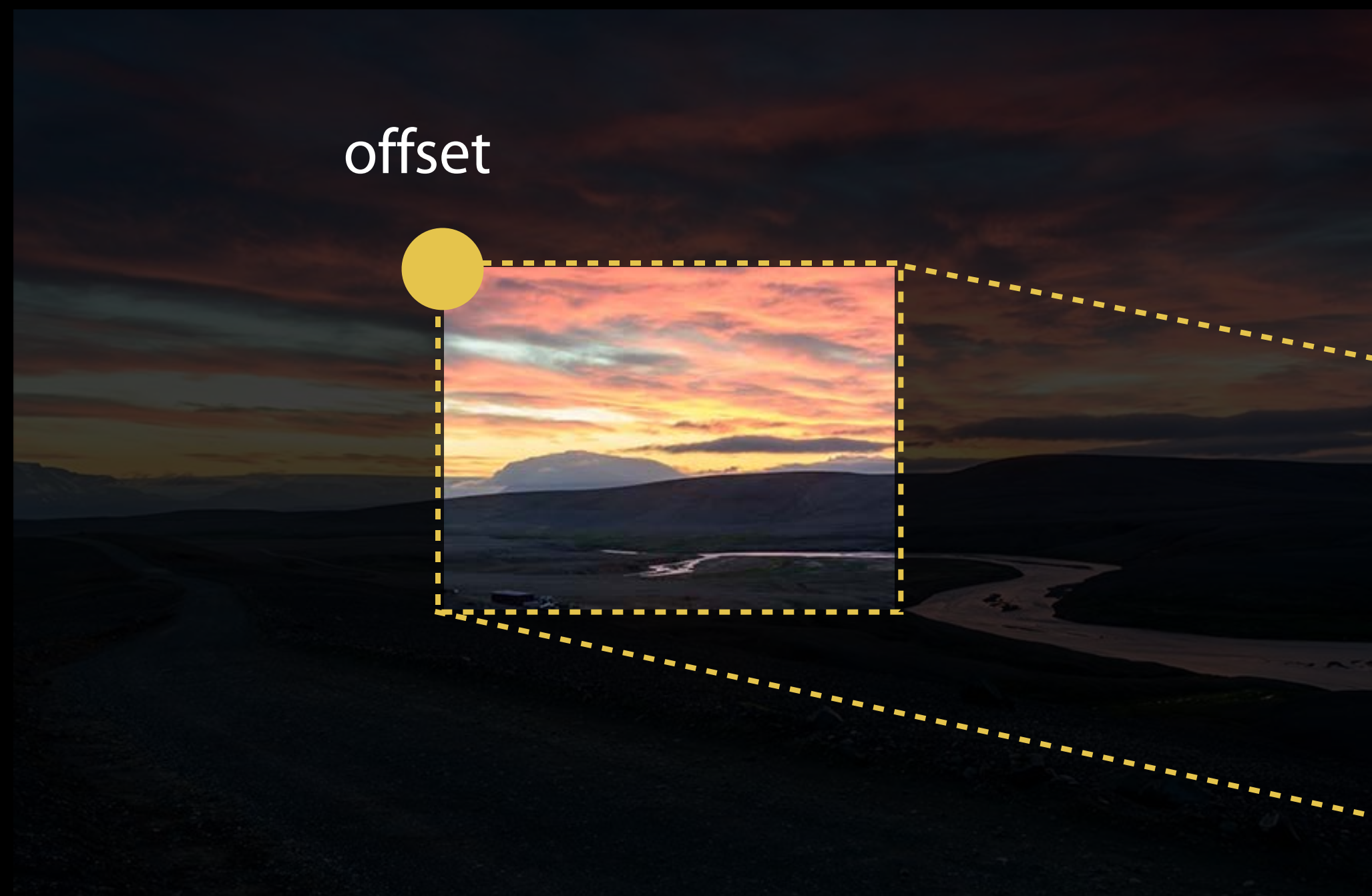
Source



Destination

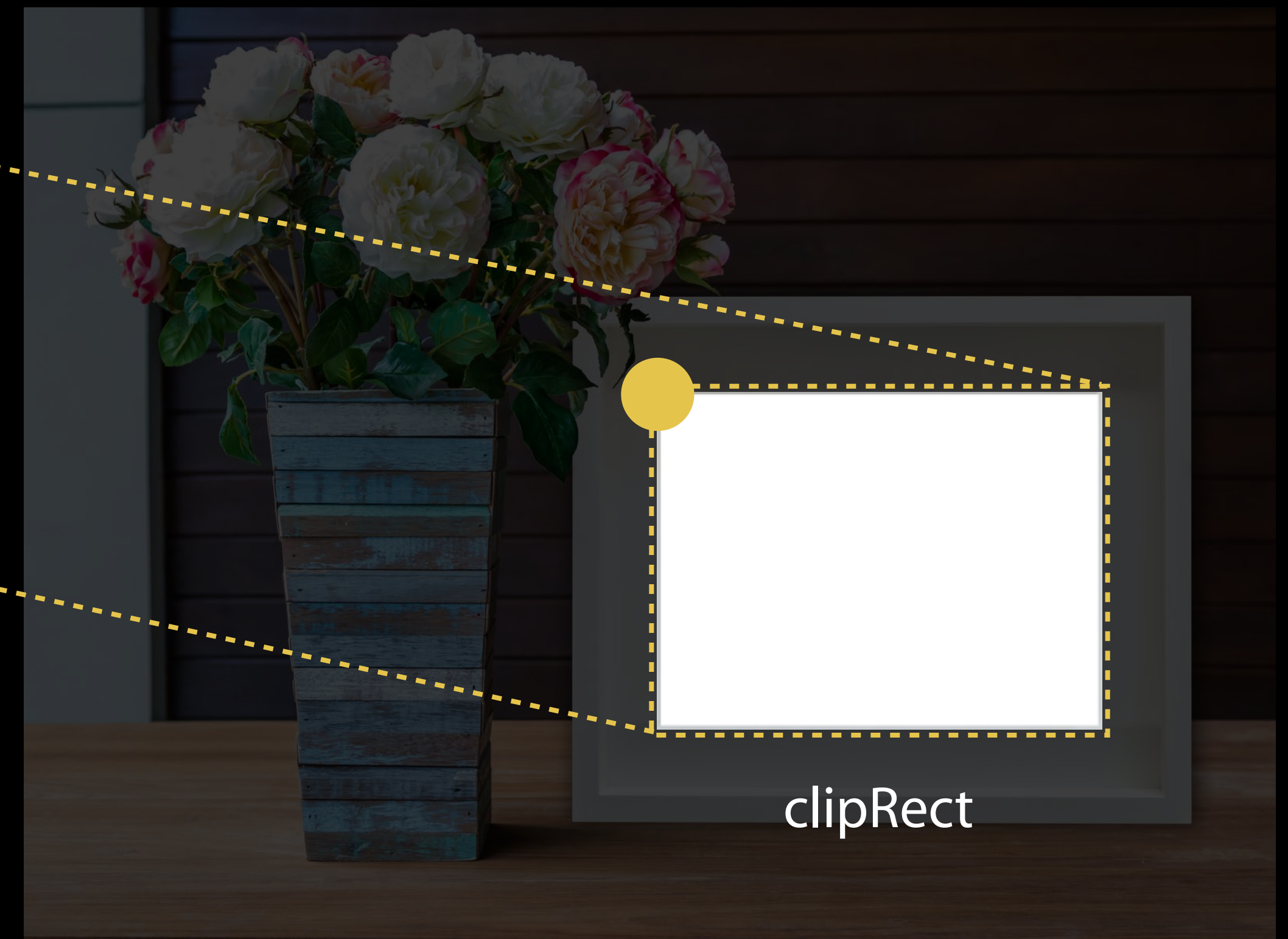
Source 'offset' and Destination 'clipRect'

Filter properties



offset

Source



clipRect

Destination

Source 'offset' and Destination 'clipRect'

Filter properties



Source



Destination

In-place Operation

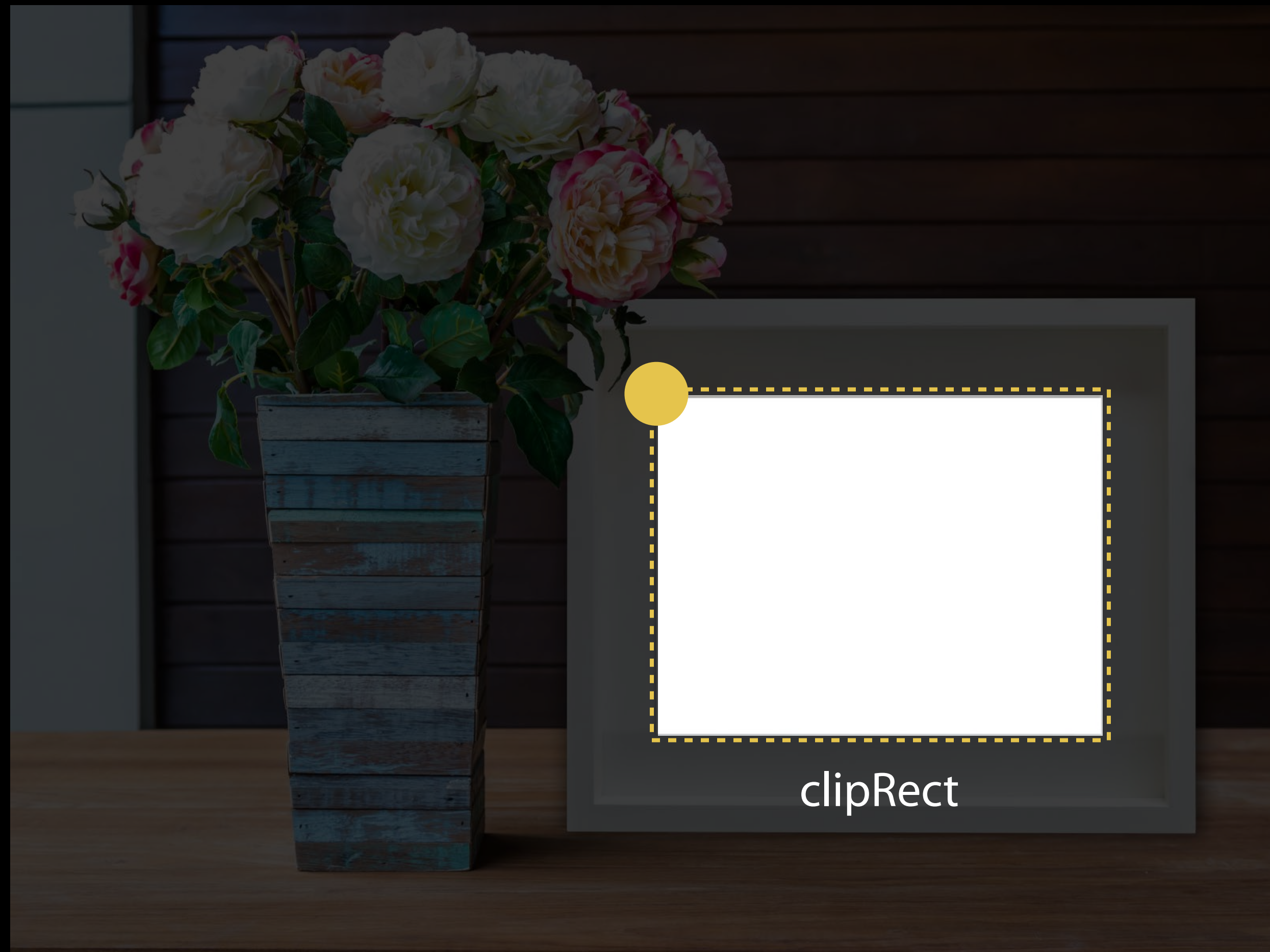
Save memory



Source == Destination

In-place Operation

Save memory

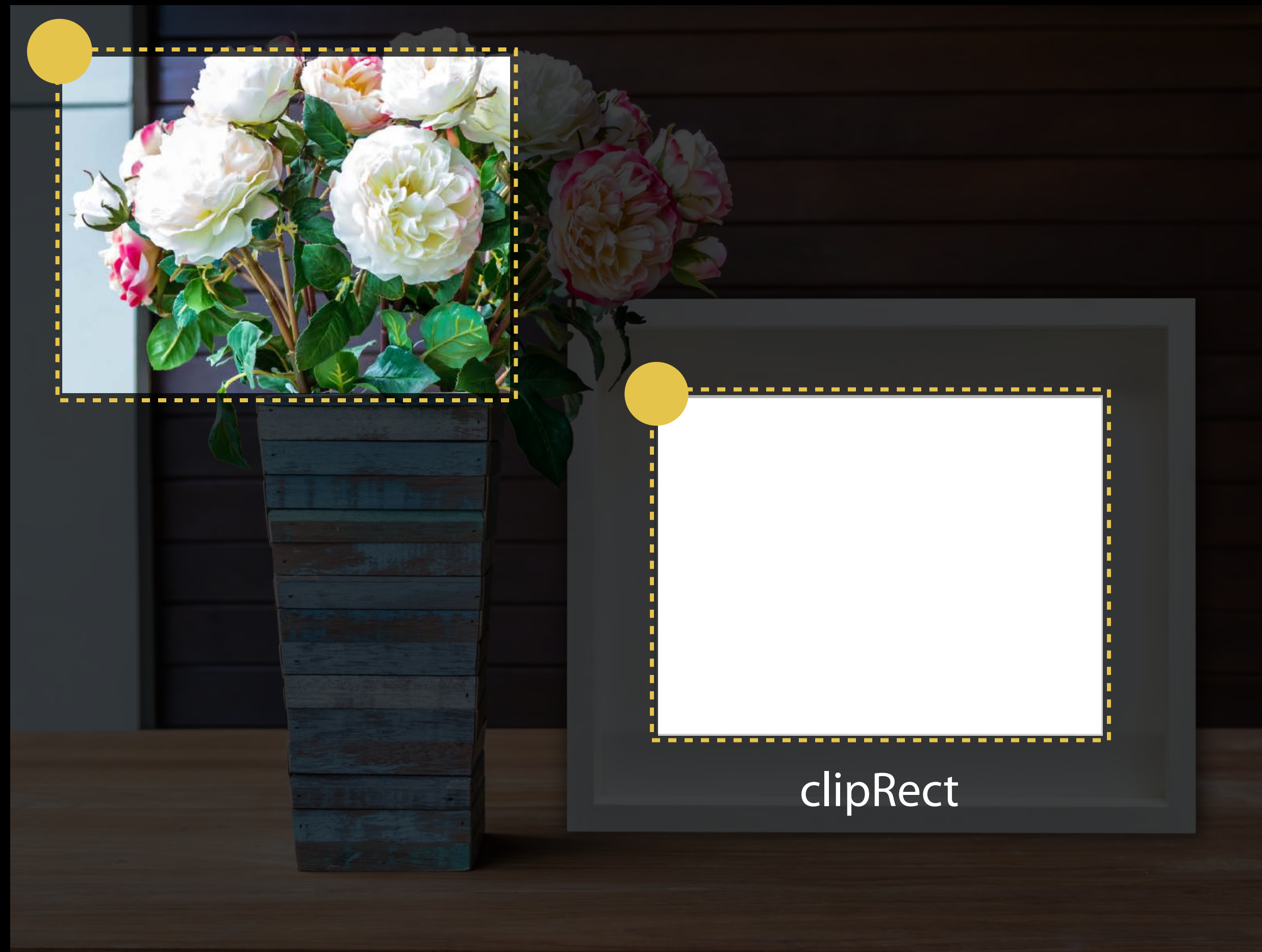


Source == Destination

In-place Operation

Save memory

offset

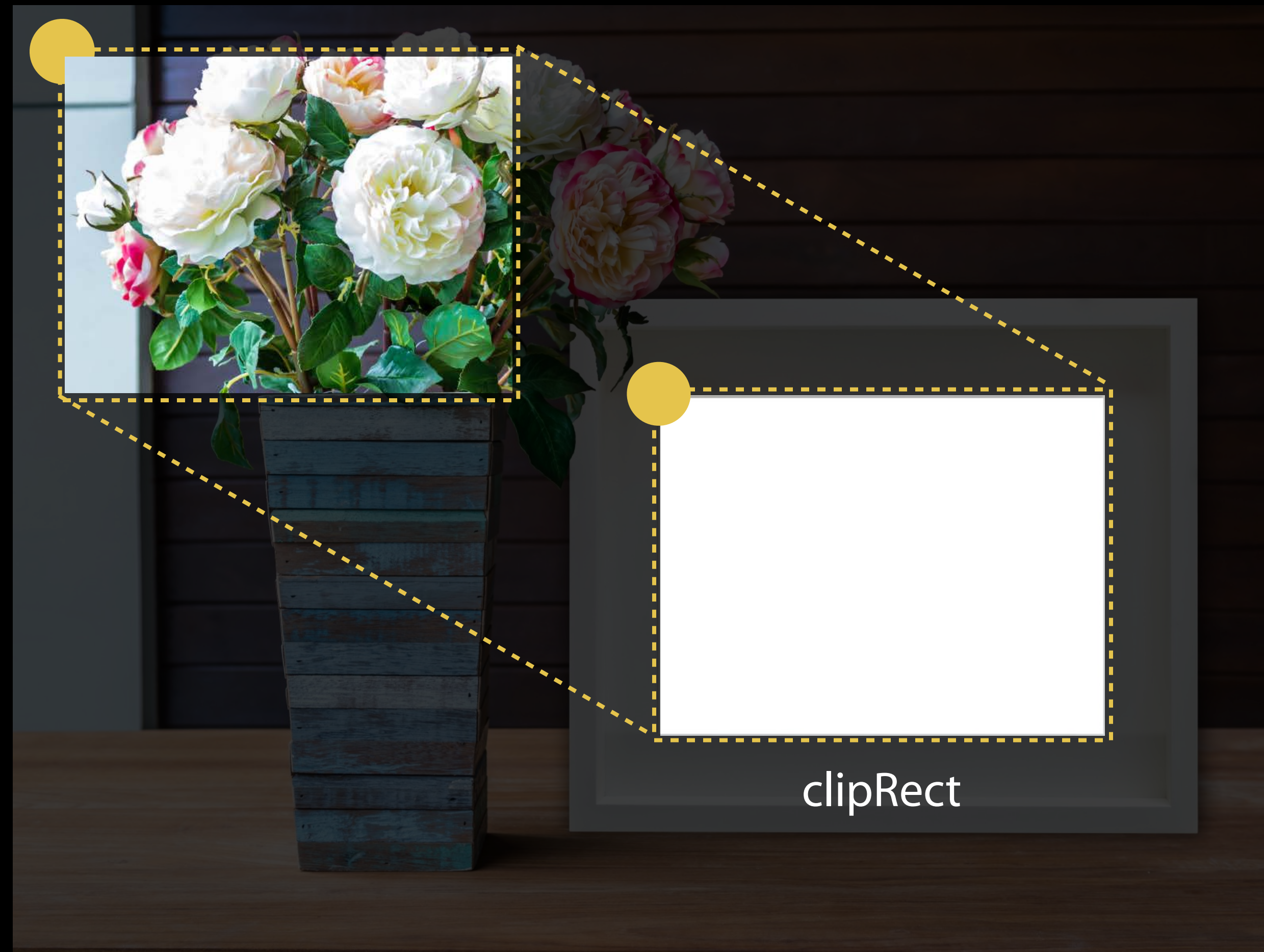


Source == Destination

In-place Operation

Save memory

offset



Source == Destination

In-place Operation

Save memory



Source == Destination

In-place Operation

How?

```
// Encode filter in-place in a Metal command buffer
[blurFilter encodeToCommandBuffer: commandBuffer
    inplaceTexture: sourceTexture
    fallbackCopyAllocator: myAllocator];
```


In-place Operation

How?

```
// Encode filter in-place in a Metal command buffer
[blurFilter encodeToCommandBuffer: commandBuffer
    inplaceTexture: sourceTexture
    fallbackCopyAllocator: myAllocator];
```

It's not always possible for Metal Performance Shaders filters to run in-place

Depends on filter, filter parameters and properties

Copy allocator will create a destination texture, so operation can proceed out-of-place

Fallback Copy Allocator

Example

```
MPSCopyAllocator myAllocator = ^id <MTLTexture>(
    MPSKernel * filter,
    id <MTLCommandBuffer> commandBuffer,
    id <MTLTexture> sourceTexture)
{
    MTLTextureDescriptor * desc = descriptorFromTexture(sourceTexture);
    id <MTLTexture> destinationTexture = [commandBuffer.device
        newTextureWithDescriptor: desc];

    return destinationTexture;
};
```

Fallback Copy Allocator

Example

```
MPSCopyAllocator myAllocator = ^id <MTLTexture>(
    MPSKernel * filter,
    id <MTLCommandBuffer> commandBuffer,
    id <MTLTexture> sourceTexture)
{
    MTLTextureDescriptor * desc = descriptorFromTexture(sourceTexture);
    id <MTLTexture> destinationTexture = [commandBuffer.device
        newTextureWithDescriptor: desc];

    // Use Metal encoder to initialize the destinationTexture from
    // sourceTexture, if necessary

    return destinationTexture;
};
```

Fallback Copy Allocator

Example

```
MPSCopyAllocator myAllocator = ^id <MTLTexture>(
    MPSKernel * filter,
    id <MTLCommandBuffer> commandBuffer,
    id <MTLTexture> sourceTexture)
{
    MTLTextureDescriptor * desc = descriptorFromTexture(sourceTexture);
    id <MTLTexture> destinationTexture = [commandBuffer.device
        newTextureWithDescriptor: desc];

    // Use Metal encoder to initialize the destinationTexture from
    // sourceTexture, if necessary

    return destinationTexture;
};
```

Summary

Make use of the new Metal support frameworks

- Robust, optimized, easy to integrate
- Faster bring-up of your application
- Less code to write and maintain

Give us your feedback!

More Information

Metal Documentation and Videos

<http://developer.apple.com/metal>

Apple Developer Forums

<http://developer.apple.com/forums>

Developer Technical Support

<http://developer.apple.com/support/technical>

General Inquiries

Allan Schaffer, Game Technologies Evangelist

aschaffer@apple.com

Related Sessions

Managing 3D Assets with Model I/O	Mission	Tuesday 2:30PM
What's New in Metal, Part 1	Presidio	Tuesday 3:30PM
Metal Performance Optimization Techniques	Pacific Heights	Friday 11:00AM

Related Labs

Metal Lab	Graphics D	Friday 12:00PM
-----------	------------	----------------

