Working With Metal—Advanced

Session 605

Gokhan Avkarogullari
GPU Software

Aaftab Munshi
GPU Software

Serhat Tekin
GPU Software

Agenda

Introduction to Metal

Fundamentals of Metal

- Building a Metal application
- Metal shading language

Advanced Metal

- Deep dive into creating a graphics application with Metal
- Data-Parallel Computing with Metal
- Developer Tools Review

Creating Multi-Pass Graphics Applications with Metal

Graphics Application with Multiple Passes

Multiple framebuffer configurations

Render to off-screen and on-screen textures

Meshes that are used with multiple shaders

Multiple encoders

Deferred Lighting with Shadow Maps

Shadow Map

- Depth-only render from the perspective of the directional light

Deferred Lighting

- Multiple render targets
- Framebuffer fetch for in-place light accumulation
- Stencil buffer for light culling

Deferred Lighting with Shadow Maps

Shadow Map

- Depth-only render from the perspective of the directional light

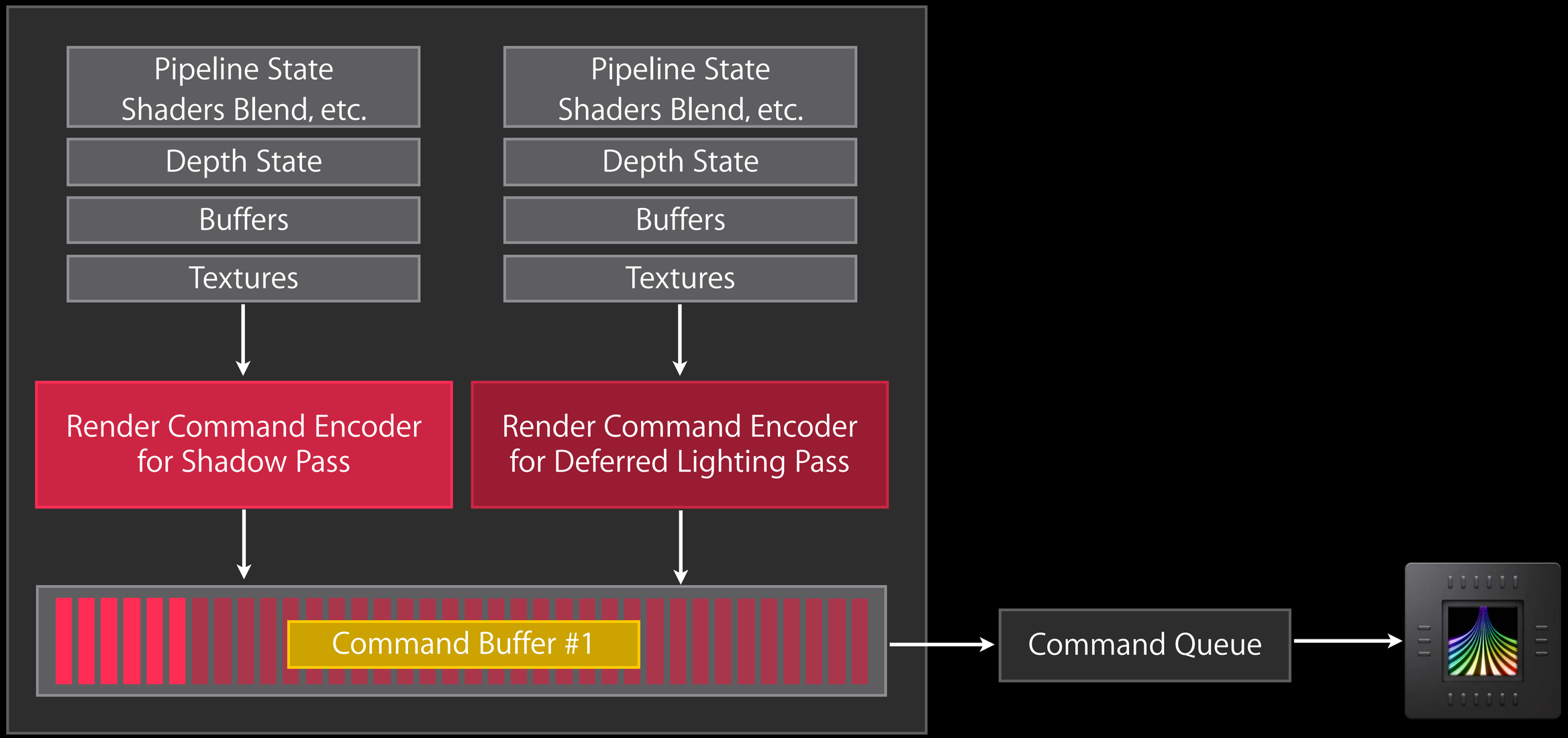
Deferred Lighting

- Multiple render targets
- Framebuffer fetch for in-place light accumulation
- Stencil buffer for light culling

Don’t worry about the algorithm

- Focus on the details of how the API is used

Deferred Lighting with Shadow Maps



Demo

Deferred Lighting

Render Setup

Actions to take once at application start time

Actions that are taken as needed

- Level load time
- Texture streaming
- Mesh streaming

Actions to take every frame

Actions to take every render-to-texture pass

Render Setup

Do once

Create device

Create command queue

Render Setup

Do as needed

Create framebuffer textures

Create render pass descriptors

Create buffers for meshes

Create render pipeline objects

Create textures

Create state objects

Create uniform buffers

Render Setup

Do every frame

Create command buffer

Update frame-based uniform buffers

Submit command buffer

Render Setup

Do every render to texture pass

Create command encoder

Draw many times

- Update uniform buffers
- Set states
- Make draw calls

Render Setup

Do as needed

Create framebuffer textures

Create render pass descriptors

Create buffers for meshes

Create render pipeline objects

Create textures

Create state objects

Create uniform buffers

Render Setup

A word on descriptors

Descriptors are like blueprints

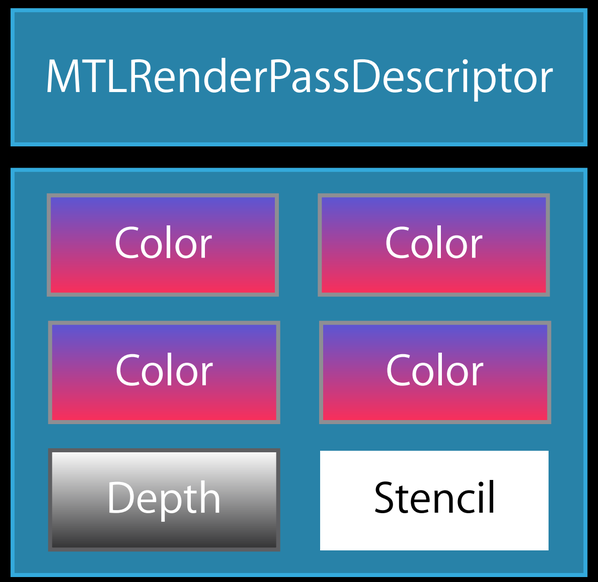
- Once the object is created the connection to the descriptor is gone
- Changing descriptors will not change the object

Same descriptor can be reused to create another object

Descriptor can be modified and then reused to create a new object

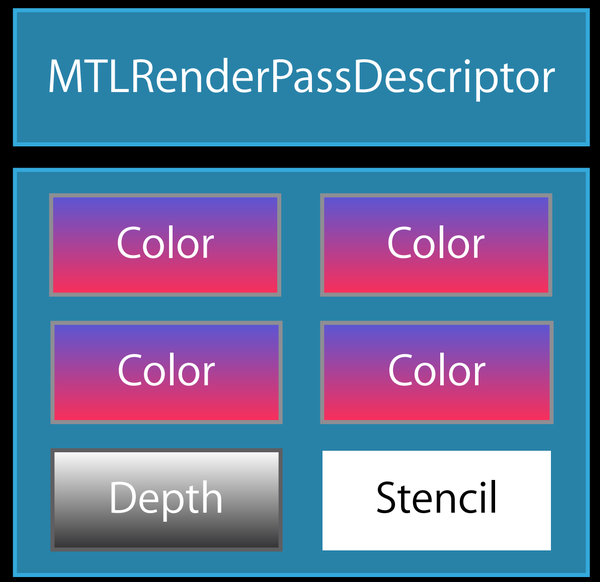
Render Setup

Creating render pass descriptors



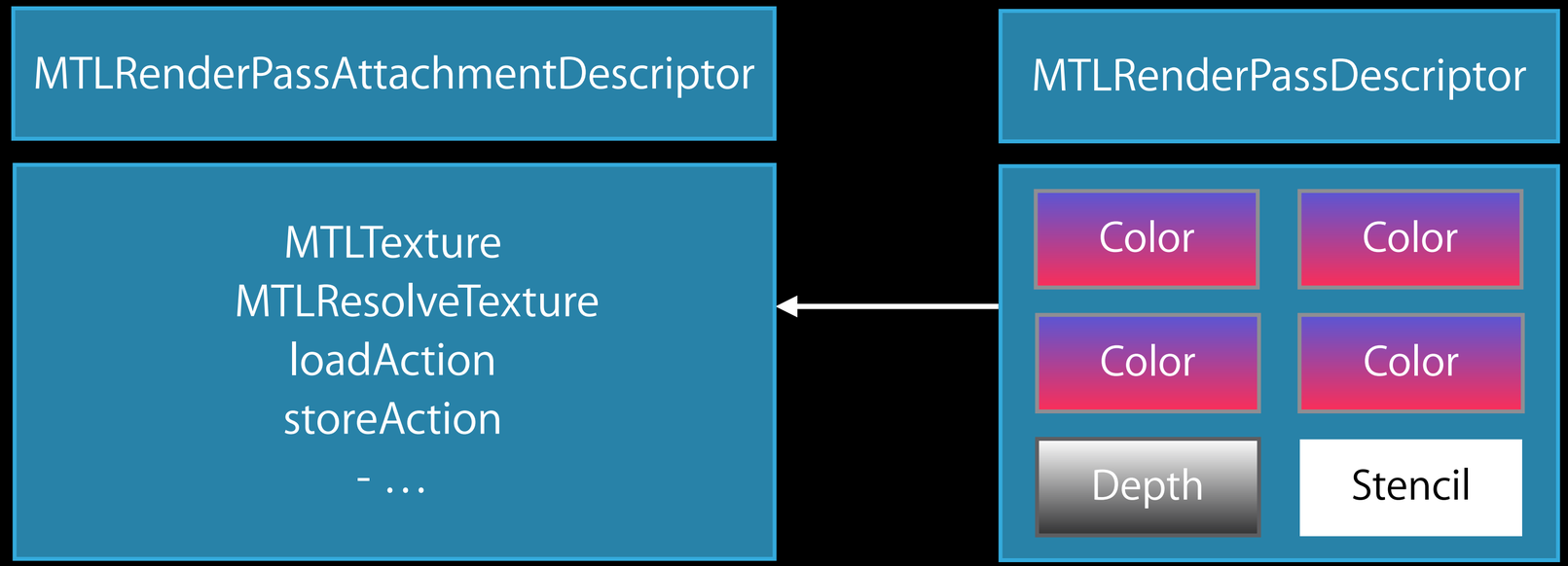
Render Setup

Creating render pass descriptors



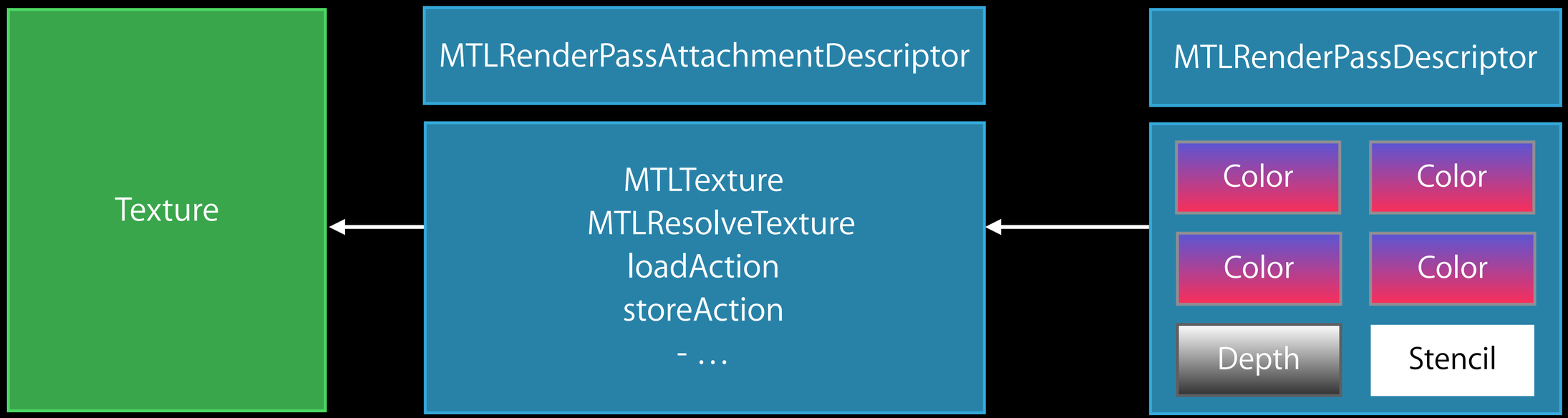
Render Setup

Creating render pass descriptors



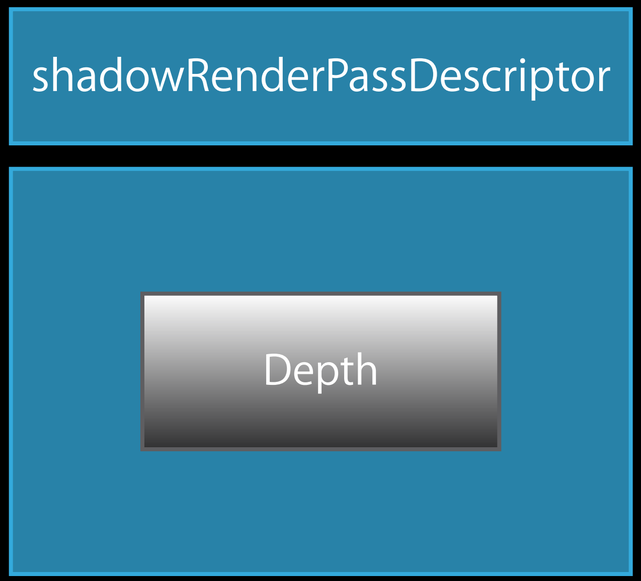
Render Setup

Creating render pass descriptors



Render Setup

Render pass descriptor for shadowMap pass



Render Setup

Render pass descriptor for shadowMap pass

```
// Create texture
id<MTLTexture> shadow_texture;
MTLTextureDescriptor *shadowTextureDesc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatDepth32Float
                                width: 1024
                               height: 1024
                            mipmapped: NO];
shadow_texture = [device newTextureWithDescriptor: shadowTextureDesc];
```

Render Setup

Render pass descriptor for shadowMap pass

```
// Create texture
id<MTLTexture> shadow_texture;
MTLTextureDescriptor *shadowTextureDesc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatDepth32Float
                                width: 1024
                               height: 1024
                            mipmapped: NO];
shadow_texture = [device newTextureWithDescriptor: shadowTextureDesc];
```

Render Setup

Render pass descriptor for shadowMap pass

```
// Create texture
id<MTLTexture> shadow_texture;
MTLTextureDescriptor *shadowTextureDesc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatDepth32Float
                                 width: 1024
                                height: 1024
                             mipmapped: NO];
shadow_texture = [device newTextureWithDescriptor: shadowTextureDesc];
```

Render Setup

Render pass descriptor for shadowMap pass

```
// Create render pass descriptor
MTLRenderPassDescriptor *shadowMapPassDesc =
                    [MTLRenderPassDescriptor renderPassDescriptor];
```

Render Setup

Render pass descriptor for shadowMap pass

```
// Create render pass descriptor
MTLRenderPassDescriptor *shadowMapPassDesc =
                  [MTLRenderPassDescriptor renderPassDescriptor];

// Set the texture on the render pass descriptor
shadowMapPassDesc.depthAttachment.texture = shadow_texture;
```

Render Setup

Render pass descriptor for shadowMap pass

```
// Create render pass descriptor
MTLRenderPassDescriptor *shadowMapPassDesc =
                    [MTLRenderPassDescriptor renderPassDescriptor];
```

```
// Set the texture on the render pass descriptor
shadowMapPassDesc.depthAttachment.texture = shadow_texture;
```

Render Setup

Render pass descriptor for shadowMap pass

```
// Create render pass descriptor
MTLRenderPassDescriptor *shadowMapPassDesc =
                  [MTLRenderPassDescriptor renderPassDescriptor];

// Set the texture on the render pass descriptor
shadowMapPassDesc.depthAttachment.texture = shadow_texture;

// Set other properties on the render pass descriptor
shadowMapPassDesc.depthAttachment.clearValue = MTLClearValueMakeDepth(1.0);
shadowMapPassDesc.depthAttachment.loadAction = MTLLoadActionClear;
shadowMapPassDesc.depthAttachment.setStoreAction = MTLStoreActionStore;
```

Render Setup

Render pass descriptor for shadowMap pass

```
// Create render pass descriptor
MTLRenderPassDescriptor *shadowMapPassDesc =
                  [MTLRenderPassDescriptor renderPassDescriptor];

// Set the texture on the render pass descriptor
shadowMapPassDesc.depthAttachment.texture = shadow_texture;
```

```
// Set other properties on the render pass descriptor
shadowMapPassDesc.depthAttachment.clearValue = MTLClearValueMakeDepth(1.0);
shadowMapPassDesc.depthAttachment.loadAction = MTLLoadActionClear;
shadowMapPassDesc.depthAttachment.setStoreAction = MTLStoreActionStore;
```

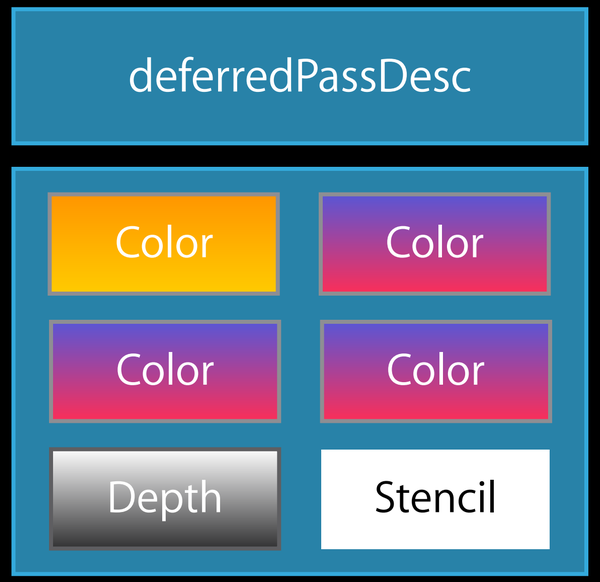
Render Setup

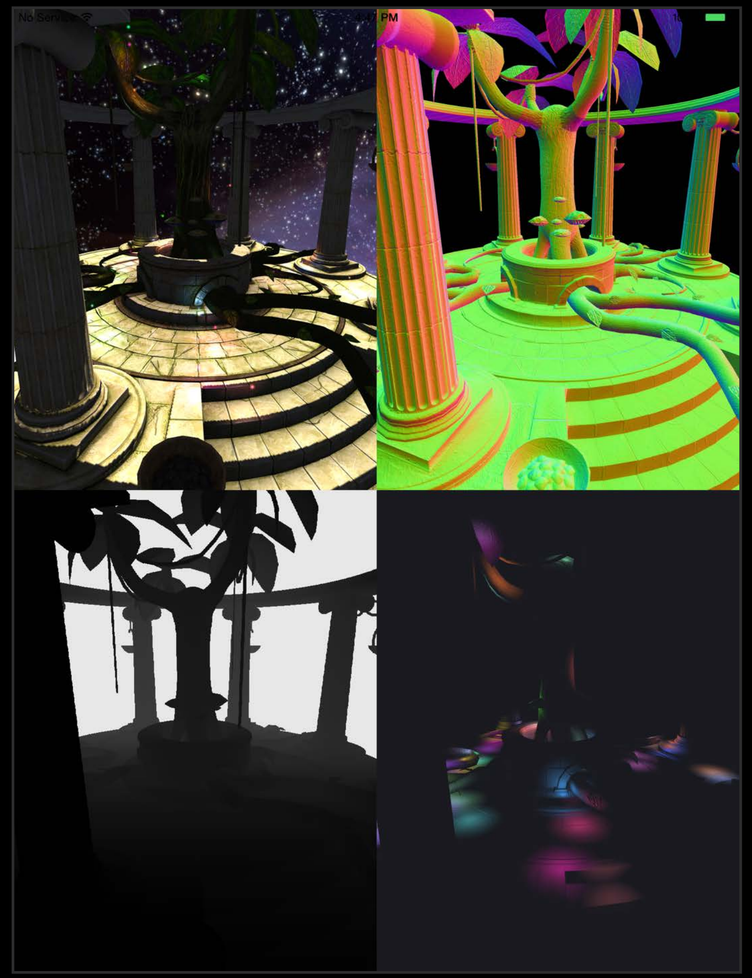
Render pass descriptor for shadowMap pass

```
// Create render pass descriptor
MTLRenderPassDescriptor *shadowMapPassDesc =
                  [MTLRenderPassDescriptor renderPassDescriptor];

// Set the texture on the render pass descriptor
shadowMapPassDesc.depthAttachment.texture = shadow_texture;

// Set other properties on the render pass descriptor
shadowMapPassDesc.depthAttachment.clearValue = MTLClearValueMakeDepth(1.0);
shadowMapPassDesc.depthAttachment.loadAction = MTLLoadActionClear;
shadowMapPassDesc.depthAttachment.setStoreAction = MTLStoreActionStore;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass





Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Create descriptor
MTLTextureDescriptor *attachmentX_texture_desc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatBGRA8Unorm
                                width: desc.width
                               height: desc.height
                            mipmapped: NO];
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Create descriptor
MTLTextureDescriptor *attachmentX_texture_desc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatBGRA8Unorm
                                width: desc.width
                               height: desc.height
                            mipmapped: NO];

// Create textures based on the descriptors
gbuffer_texture1 = [device newTextureWithDescriptor: attachmentX_tex_desc];
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Create descriptor
MTLTextureDescriptor *attachmentX_texture_desc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatBGRA8Unorm
                                width: desc.width
                               height: desc.height
                            mipmapped: NO];
```

```
// Create textures based on the descriptors
gbuffer_texture1 = [device newTextureWithDescriptor: attachmentX_tex_desc];
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Create descriptor
MTLTextureDescriptor *attachmentX_texture_desc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatBGRA8Unorm
                                width: desc.width
                               height: desc.height
                            mipmapped: NO];

// Create textures based on the descriptors
gbuffer_texture1 = [device newTextureWithDescriptor: attachmentX_tex_desc];

// Modify descriptor and create new texture
[attachmentX_texture_desc setPixelFormat: … ];
gbuffer_texture2 = [device newTextureWithDescriptor: attachmentX_tex_desc];
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Create descriptor
MTLTextureDescriptor *attachmentX_texture_desc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatBGRA8Unorm
                                width: desc.width
                               height: desc.height
                            mipmapped: NO];

// Create textures based on the descriptors
gbuffer_texture1 = [device newTextureWithDescriptor: attachmentX_tex_desc];

// Modify descriptor and create new texture
[attachmentX_texture_desc setPixelFormat: … ];
gbuffer_texture2 = [device newTextureWithDescriptor: attachmentX_tex_desc];
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Create descriptor
MTLTextureDescriptor *attachmentX_texture_desc = [MTLTextureDescriptor
    texture2DDescriptorWithPixelFormat: MTLPixelFormatBGRA8Unorm
                                width: desc.width
                               height: desc.height
                            mipmapped: NO];

// Create textures based on the descriptors
gbuffer_texture1 = [device newTextureWithDescriptor: attachmentX_tex_desc];

// Modify descriptor and create new texture
[attachmentX_texture_desc setPixelFormat: … ];
gbuffer_texture2 = [device newTextureWithDescriptor: attachmentX_tex_desc];
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Create render pass descriptor
MTLRenderPassDescriptor *deferredPassDesc =
                    [MTLRenderPassDescriptor renderPassDescriptor];
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil; //will come from drawable
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil; //will come from drawable
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;

// Describe color attachment 1
deferredPassDesc.colorAttachments[1].texture = gbuffer_texture1;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;
```

```
// Describe color attachment 1
deferredPassDesc.colorAttachments[1].texture = gbuffer_texture1;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;

// Describe color attachment 1
deferredPassDesc.colorAttachments[1].texture = gbuffer_texture1;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;

// Describe color attachment 1
deferredPassDesc.colorAttachments[1].texture = gbuffer_texture1;

deferredPassDesc.colorAttachments[1].clearValue = clearColor1;
deferredPassDesc.colorAttachments[1].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[1].storeAction = MTLStoreActionDontCare;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;

// Describe color attachment 1
deferredPassDesc.colorAttachments[1].texture = gbuffer_texture1;

deferredPassDesc.colorAttachments[1].clearValue = clearColor1;
deferredPassDesc.colorAttachments[1].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[1].storeAction = MTLStoreActionDontCare;
```

Render Setup

Render pass descriptor for the Deferred Lighting pass

```
// Describe color attachment 0
deferredPassDesc.colorAttachments[0].texture = nil;

deferredPassDesc.colorAttachments[0].clearValue = clearColor1;
deferredPassDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[0].storeAction = MTLStoreActionStore;

// Describe color attachment 1
deferredPassDesc.colorAttachments[1].texture = gbuffer_texture1;

deferredPassDesc.colorAttachments[1].clearValue = clearColor1;
deferredPassDesc.colorAttachments[1].loadAction = MTLLoadActionClear;
deferredPassDesc.colorAttachments[1].storeAction = MTLStoreActionDontCare;
```

Render Setup

Do as needed

Create framebuffer textures

Create render pass descriptors

Create textures

Create buffers for meshes

Create state objects

Create pipeline objects

Create uniform buffers

Creating Textures

```
// Copy texture data to bitmapData
unsigned Npixels = tex_info.width * tex_info.height;

id<MTLTexture> texture = [device newTextureWithDescriptor: …];
```

Creating Textures

```
// Copy texture data to bitmapData
unsigned Npixels = tex_info.width * tex_info.height;

id<MTLTexture> texture = [device newTextureWithDescriptor: …];

[texture replaceRegion: bitmapData …
```

Creating Textures

```
// Copy texture data to bitmapData
unsigned Npixels = tex_info.width * tex_info.height;

id<MTLTexture> texture = [device newTextureWithDescriptor: …];

[texture replaceRegion: bitmapData …
```

Creating Buffers for Meshes

```
float4 temple[100];
…
spriteBuffer = [device newBufferWithBytes: &temple
                                   length: sizeof(temple)
                                  options: 0];
```

Creating Buffers for Meshes

```
float4 temple[100];
…
spriteBuffer = [device newBufferWithBytes: &temple
                                   length: sizeof(temple)
                                  options: 0];
```

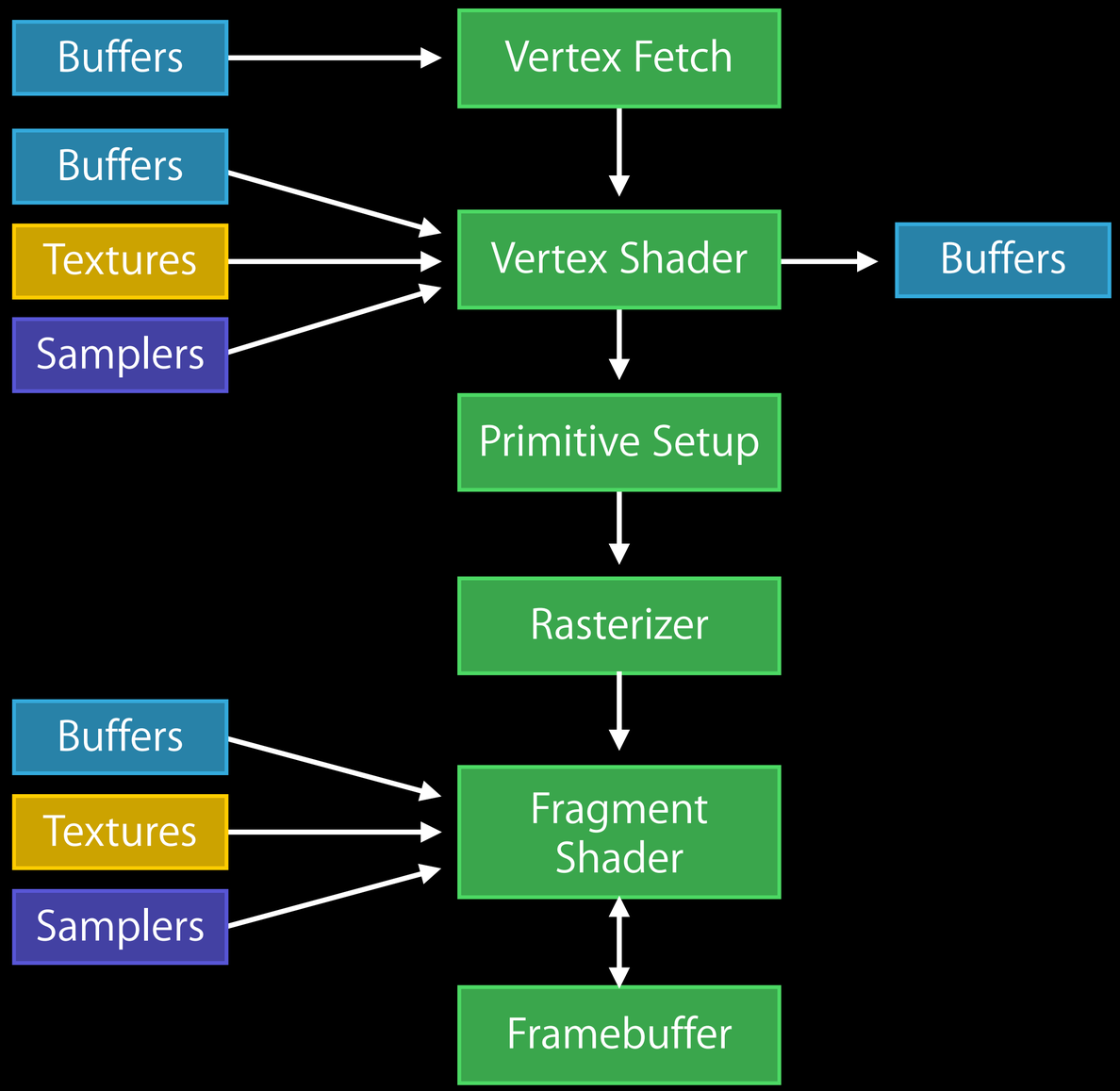
Creating State Objects

```
MTLDepthStencilDescriptor *desc = [[MTLDepthStencilDescriptor alloc] init];
desc.depthCompareFunction = MTLCompareFunctionLessEqual;
desc.depthWriteEnabled = YES;

MTLStencilDescriptor *stencilStateDesc = [[MTLStencilDescriptor alloc] init];
stencilState.stencilCompareFunction = MTLCompareFunctionAlways;
stencilState.stencilFailureOperation = MTLStencilOperationKeep;
…
desc.frontFaceStencilDescriptor = stencilStateDesc;
desc.backFaceStencilDescriptor = stencilStateDesc;

id <MTLDepthStencilState> shadowDepthStencilState = [device
newDepthStencilStateWithDescriptor: desc];
```

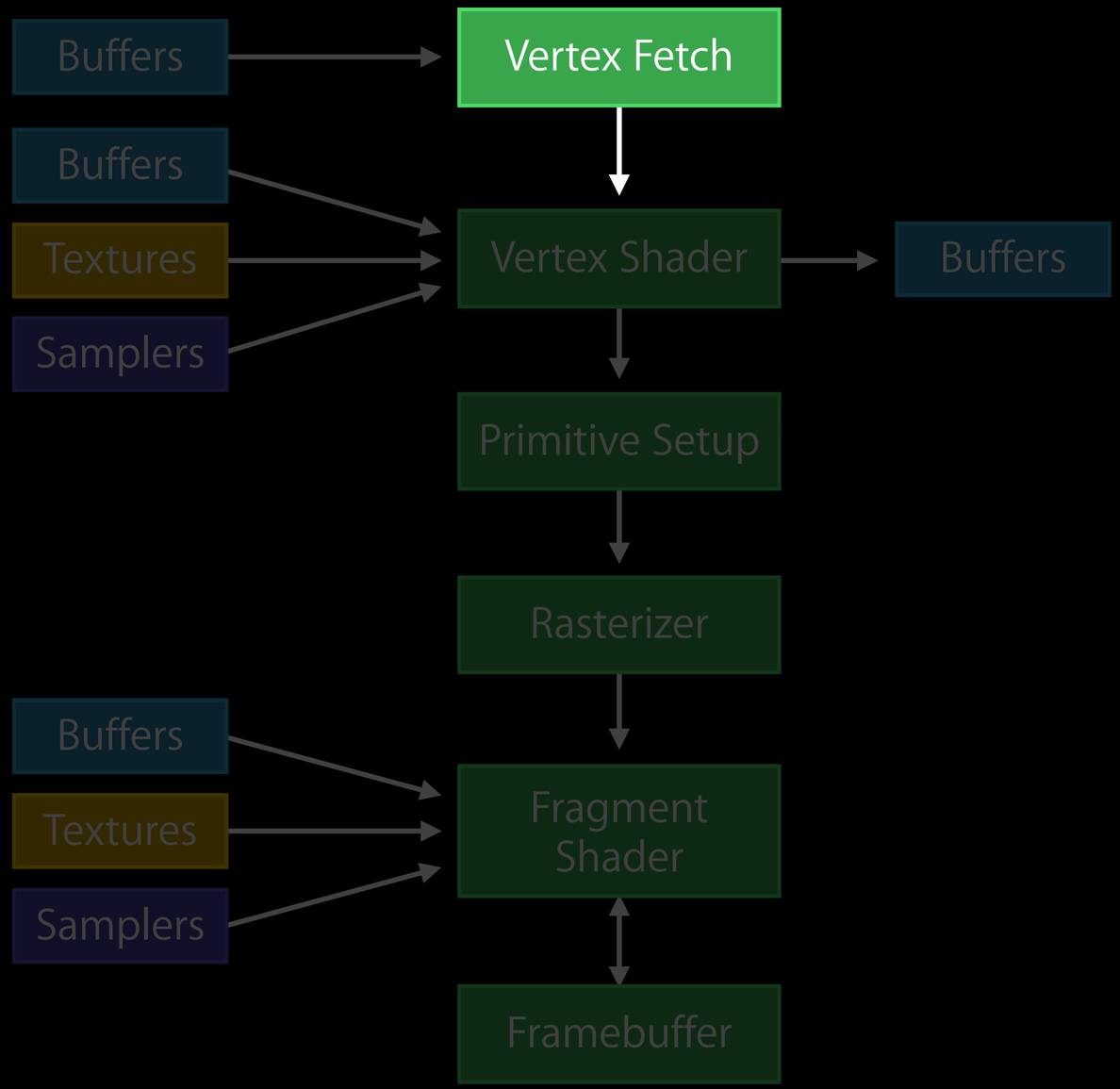
Creating State Objects

```
MTLDepthStencilDescriptor *desc = [[MTLDepthStencilDescriptor alloc] init];
desc.depthCompareFunction = MTLCompareFunctionLessEqual;
desc.depthWriteEnabled = YES;

MTLStencilDescriptor *stencilStateDesc = [[MTLStencilDescriptor alloc] init];
stencilState.stencilCompareFunction = MTLCompareFunctionAlways;
stencilState.stencilFailureOperation = MTLStencilOperationKeep;
…
desc.frontFaceStencilDescriptor = stencilStateDesc;
desc.backFaceStencilDescriptor = stencilStateDesc;

id <MTLDepthStencilState> shadowDepthStencilState = [device
newDepthStencilStateWithDescriptor: desc];
```

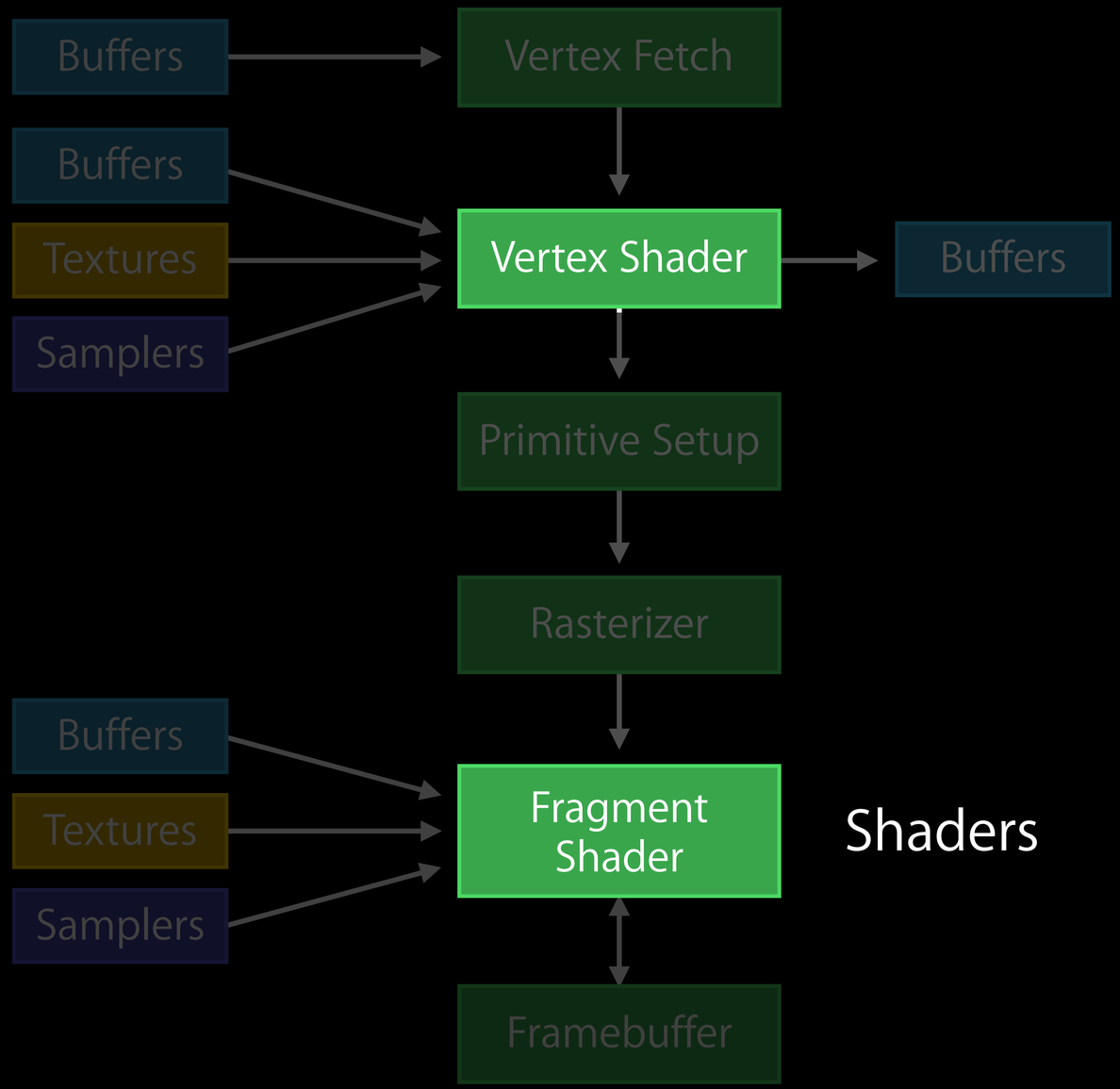
GPU Render Pipeline



Render Pipeline State Object

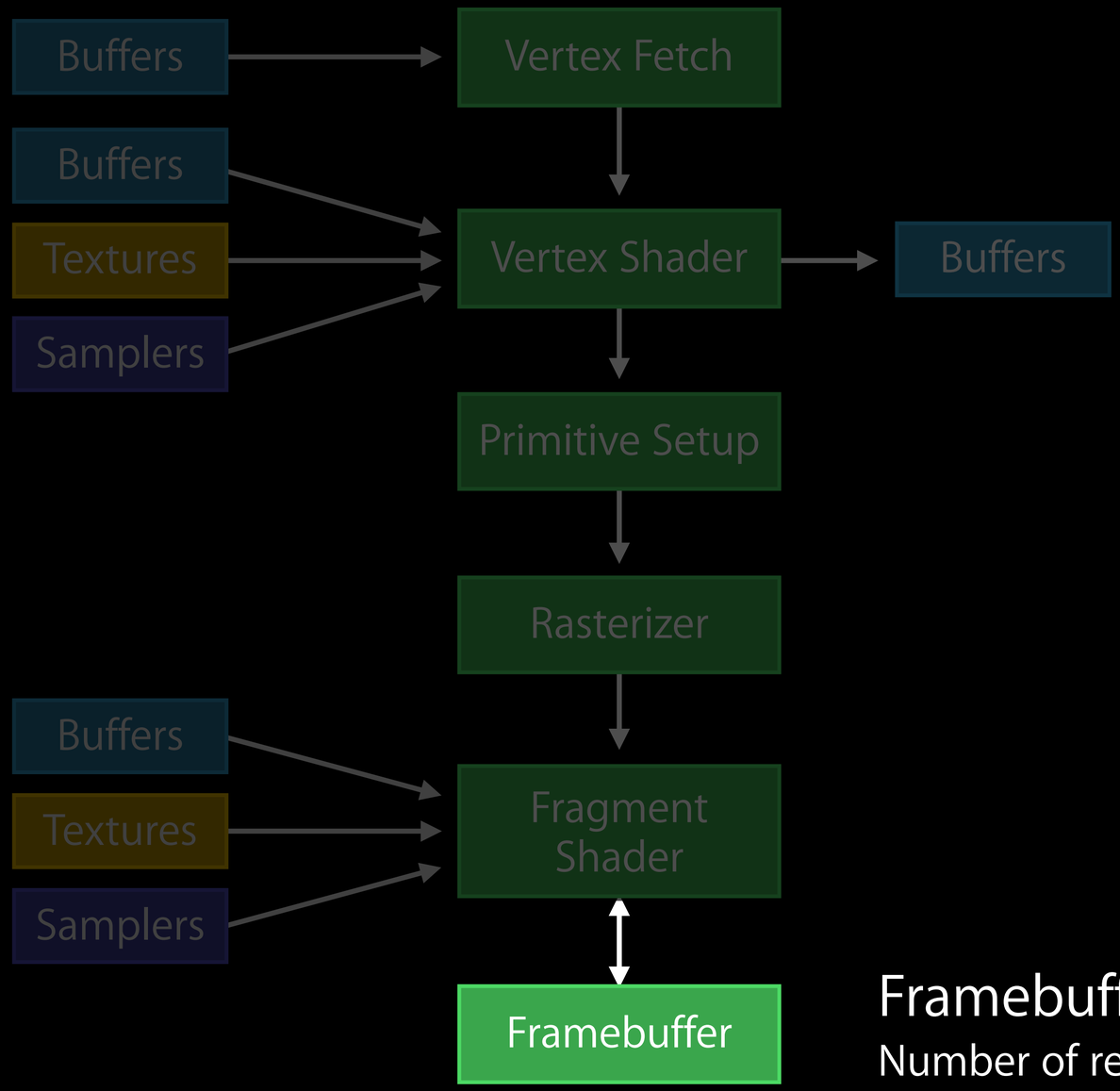


Render Pipeline State Object



Shaders

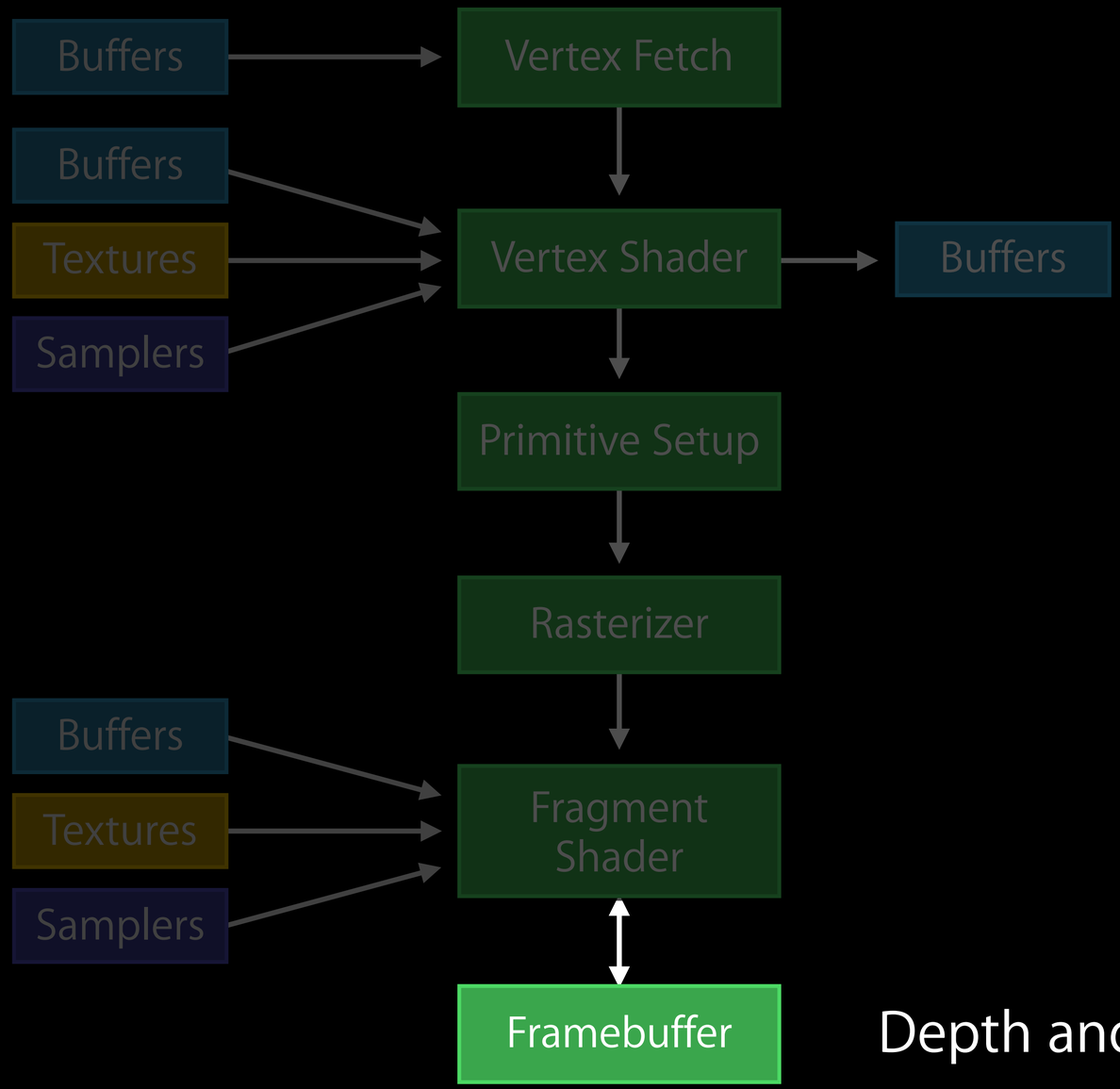
Render Pipeline State Object



Framebuffer configuration

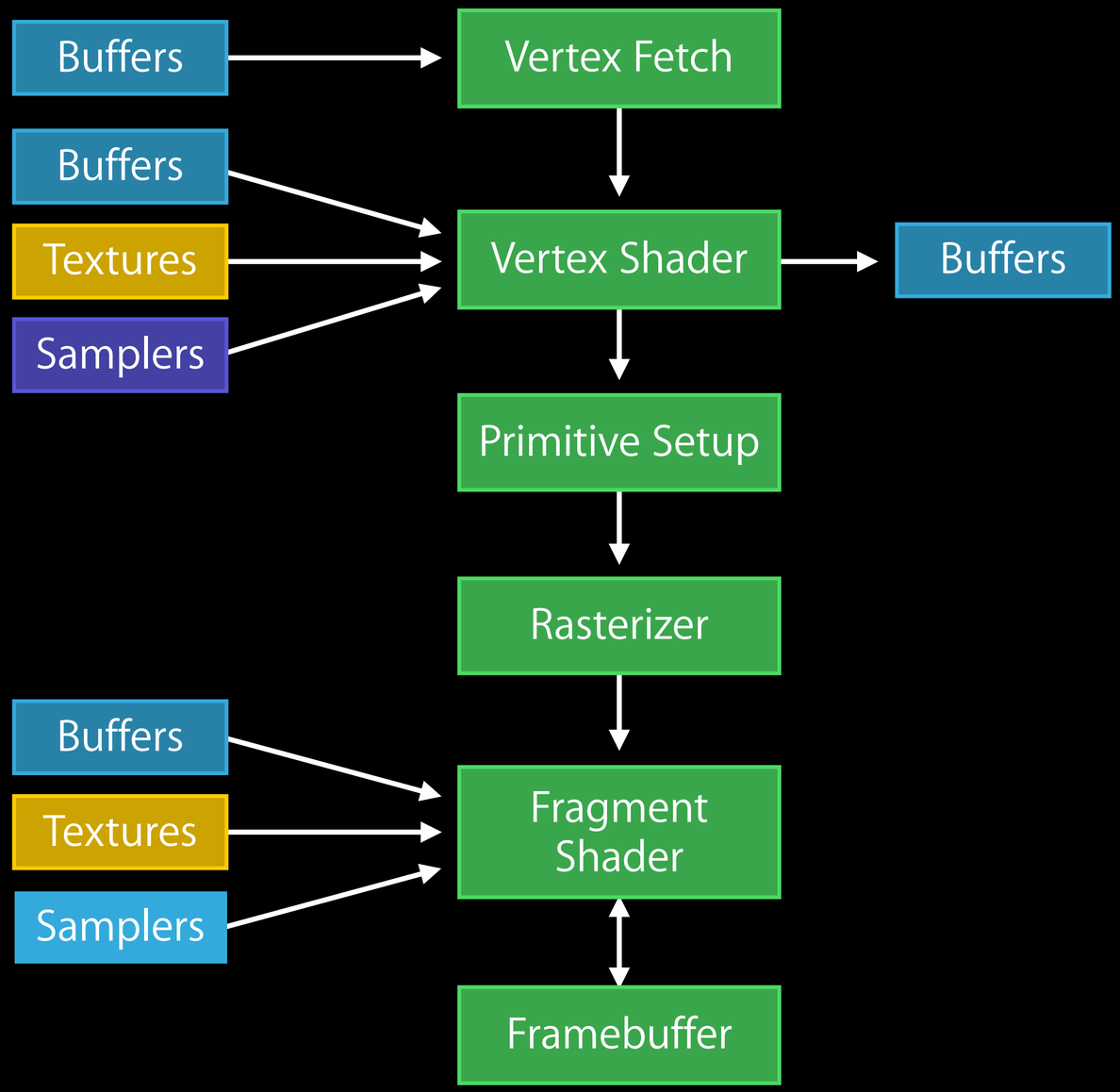
Number of render targets, pixel format, sample count

Render Pipeline State Object

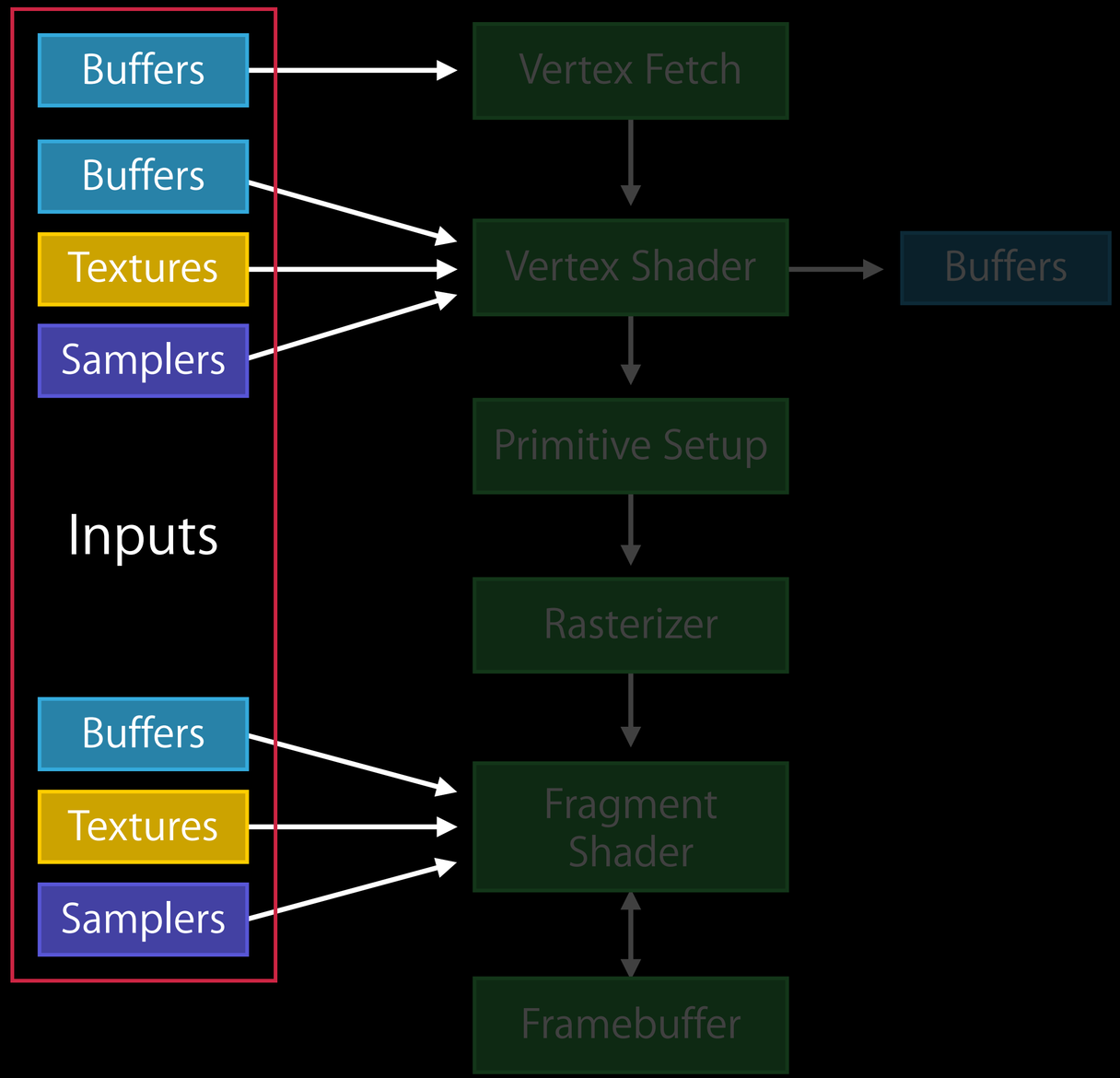


Depth and stencil state

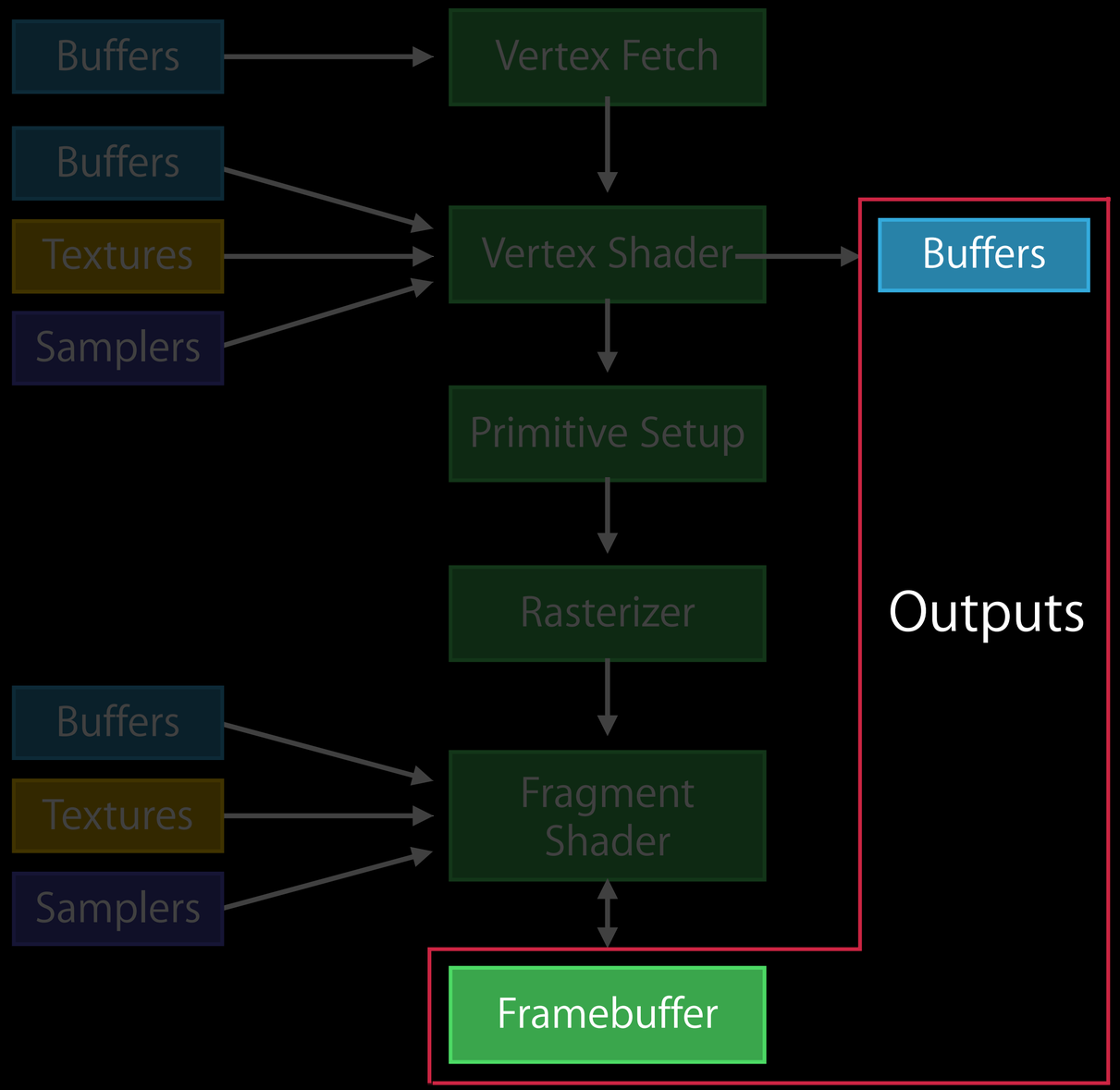
Render Pipeline State Object



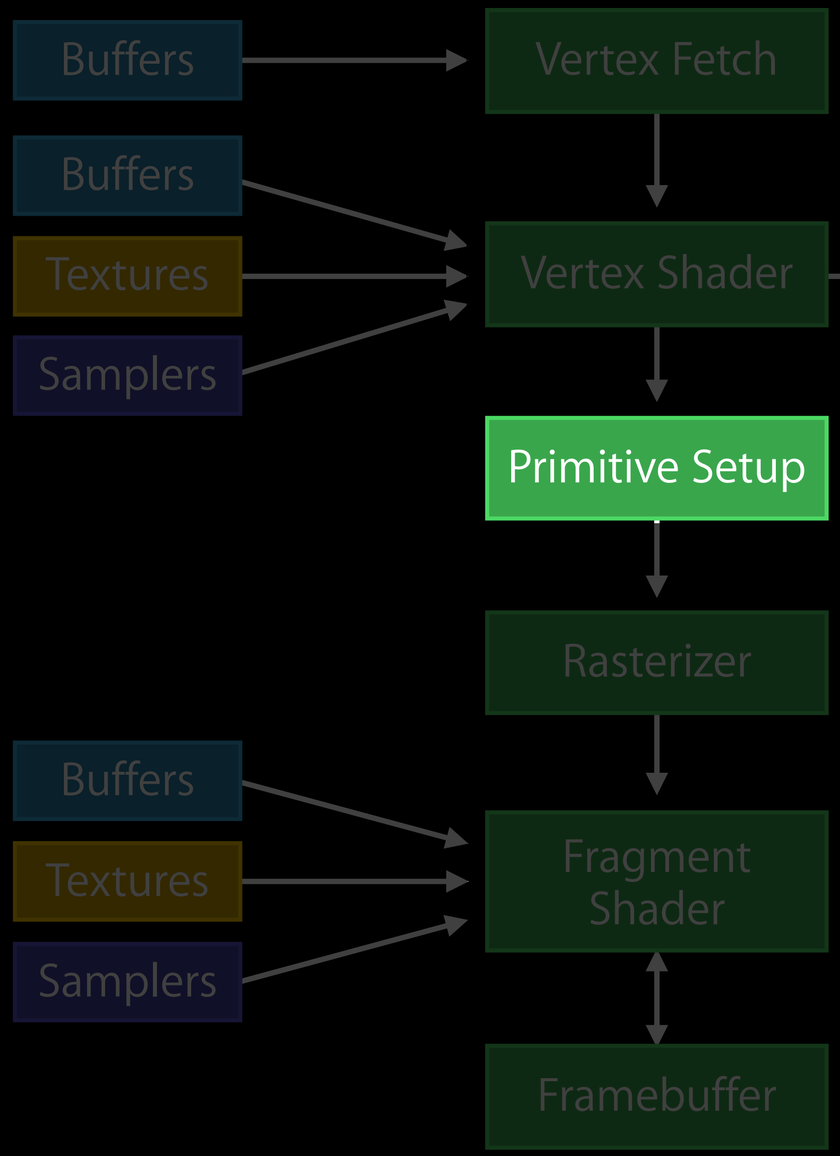
Render Pipeline State Object—Not Included



Render Pipeline State Object—Not Included



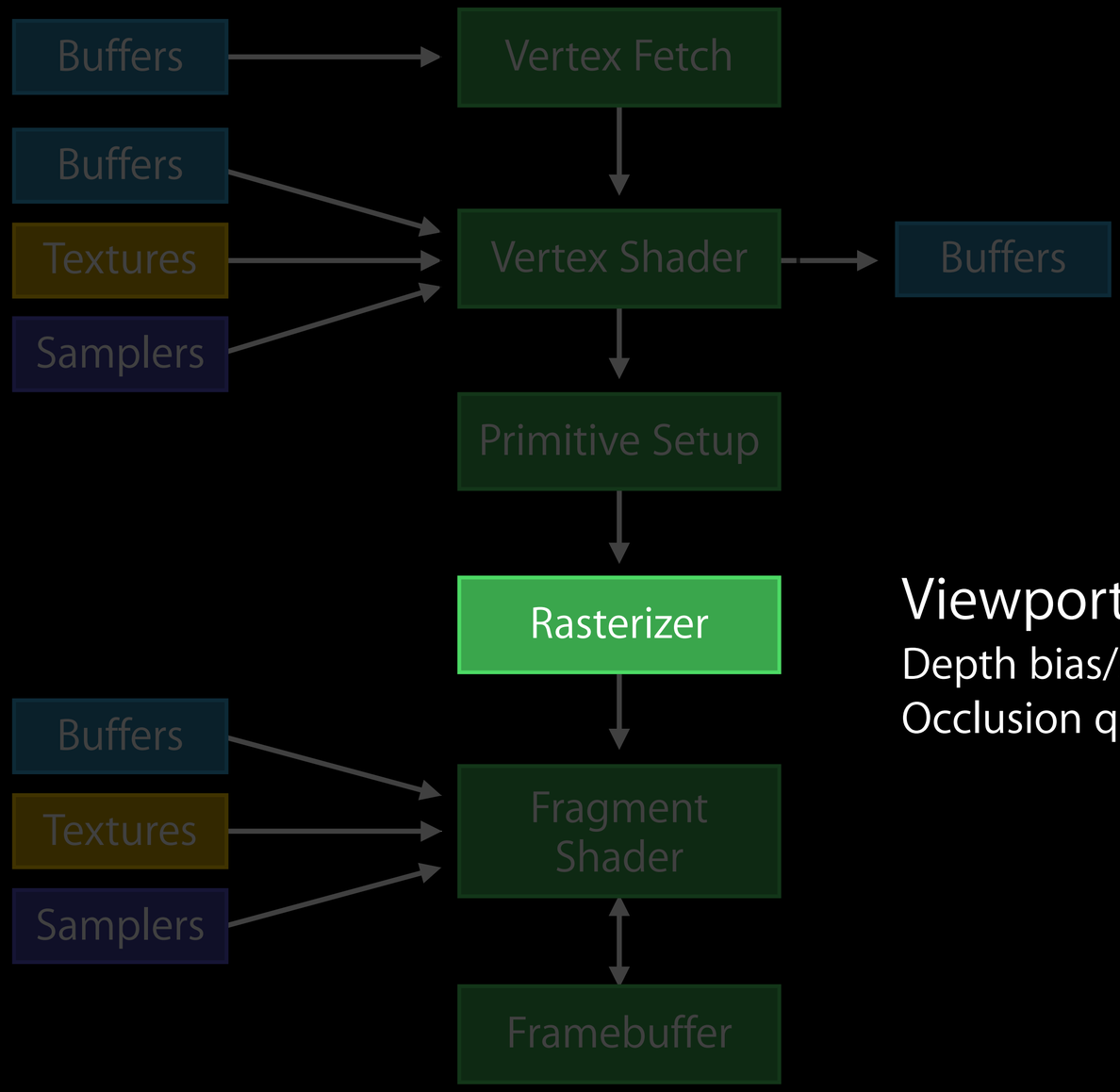
Render Pipeline State Object—Not Included



Primitive setup

Cull mode, facing orientation
depth clipping, polygon mode

Render Pipeline State Object—Not Included



Viewport and scissor

Depth bias/slope/clamp

Occlusion queries

Creating Render Pipeline State Object

Every draw call requires a render pipeline state object to be set

Same mesh usually has multiple pipeline state objects

- The temple object in our demo is rendered twice
  - The shadow pass—Depth only render with simple vertex shader
  - The deferred pass—Rendered to generate g-buffer attributes
  - Each one requires a different render pipeline state object to be created

Creating Render Pipeline State Object

shadowMap pass

```
// Create the descriptor
MTLRenderPipelineDescriptor *desc = [MTLRenderPipelineDescriptor new];
```

Creating Render Pipeline State Object

shadowMap pass

```
// Create the descriptor
MTLRenderPipelineDescriptor *desc = [MTLRenderPipelineDescriptor new];

// Get the shaders from the library
id <MTLFunction> zOnlyVert = [zOnlyLibrary newFunctionWithName:@"ZOnly"];
```

Creating Render Pipeline State Object

shadowMap pass

```
// Create the descriptor
MTLRenderPipelineDescriptor *desc = [MTLRenderPipelineDescriptor new];
```

```
// Get the shaders from the library
id <MTLFunction> zOnlyVert = [zOnlyLibrary newFunctionWithName:@"ZOnly"];
```

Creating Render Pipeline State Object

shadowMap pass

```
// Create the descriptor
MTLRenderPipelineDescriptor *desc = [MTLRenderPipelineDescriptor new];

// Get the shaders from the library
id <MTLFunction> zOnlyVert = [zOnlyLibrary newFunctionWithName:@"ZOnly"];
```

```
// Set the states
desc.label = @"Shadow Render";
desc.vertexFunction = zOnlyVert;
desc.stencilWriteEnabled = false;
desc.depthWriteEnabled = true;
desc.fragmentFunction = nil;  //depth write only
desc.depthAttachmentPixelFormat = pixelFormat;
```

Creating Render Pipeline State Object

shadowMap pass

```
// Create the descriptor
MTLRenderPipelineDescriptor *desc = [MTLRenderPipelineDescriptor new];

// Get the shaders from the library
id <MTLFunction> zOnlyVert = [zOnlyLibrary newFunctionWithName:@"ZOnly"];
```

```
// Set the states
desc.label = @"Shadow Render";
desc.vertexFunction = zOnlyVert;
desc.stencilWriteEnabled = false;
desc.depthWriteEnabled = true;
desc.fragmentFunction = nil;  //depth write only
desc.depthAttachmentPixelFormat = pixelFormat;
```

Creating Render Pipeline State Object

shadowMap pass

```
MTLRenderPipelineDescriptor *desc = [MTLRenderPipelineDescriptor new];

id <MTLFunction> zOnlyVert = [zOnlyLibrary newFunctionWithName:@"ZOnly"];

desc.label = @"Shadow Render";
desc.vertexFunction = zOnlyVert;
desc.stencilWriteEnabled = false;
desc.depthWriteEnabled = true;
desc.fragmentFunction = nil;  //depth write only
desc.depthAttachmentPixelFormat = pixelFormat;

// Create the render pipeline state object
id<MTLRenderPipelineState> pipeline = [device
    newRenderPipelineStateWithDescriptor: desc error: &err];
```

Creating Render Pipeline State Object

shadowMap pass

```
MTLRenderPipelineDescriptor *desc = [MTLRenderPipelineDescriptor new];

id <MTLFunction> zOnlyVert = [zOnlyLibrary newFunctionWithName:@"ZOnly"];

desc.label = @"Shadow Render";
desc.vertexFunction = zOnlyVert;
desc.stencilWriteEnabled = false;
desc.depthWriteEnabled = true;
desc.fragmentFunction = nil;  //depth write only
desc.depthAttachmentPixelFormat = pixelFormat;

// Create the render pipeline state object
id<MTLRenderPipelineState> pipeline = [device
    newRenderPipelineStateWithDescriptor: desc error: &err];
```

Creating Render Pipeline State Object

Deferred Lighting pass

```
desc.vertexFunction   = gBufferVert;
desc.fragmentFunction = gBufferFrag;

desc.colorAttachments[0].pixelFormat = gbuffer_texture0.pixelFormat;
desc.colorAttachments[1].pixelFormat = gbuffer_texture1.pixelFormat;
```

```
…

desc.depthAttachmentPixelFormat = depth_texture.pixelFormat;
desc.stencilAttachmentPixelFormat = stencil_texture.pixelFormat;
```

Creating Render Pipeline State Object

Deferred Lighting pass

```
desc.vertexFunction   = gBufferVert;
desc.fragmentFunction = gBufferFrag;

desc.colorAttachments[0].pixelFormat = gbuffer_texture0.pixelFormat;
desc.colorAttachments[1].pixelFormat = gbuffer_texture1.pixelFormat;
```

```
…

desc.depthAttachmentPixelFormat = depth_texture.pixelFormat;
desc.stencilAttachmentPixelFormat = stencil_texture.pixelFormat;
```

Creating Render Pipeline State Object

Deferred Lighting pass

```
desc.vertexFunction   = gBufferVert;
desc.fragmentFunction = gBufferFrag;

desc.colorAttachments[0].pixelFormat = gbuffer_texture0.pixelFormat;
desc.colorAttachments[1].pixelFormat = gbuffer_texture1.pixelFormat;
```

```
…

desc.depthAttachmentPixelFormat = depth_texture.pixelFormat;
desc.stencilAttachmentPixelFormat = stencil_texture.pixelFormat;
```

Render Setup

Do every frame

Create command buffer

Update frame-based uniform buffers

Submit command buffer

Render Setup

Create and submit command buffer

```
// BeginFrame
commandBuffer = [commandQueue commandBuffer];


// EndFrame
[commandBuffer addPresent: drawable];
[commandBuffer commit];
commandBuffer = nil;
```

Render Setup

Create and submit command buffer

```
// BeginFrame
commandBuffer = [commandQueue commandBuffer];


// EndFrame
[commandBuffer addPresent: drawable];
[commandBuffer commit];
commandBuffer = nil;
```

Render Setup

Create and submit command buffer

```
// BeginFrame
commandBuffer = [commandQueue commandBuffer];


// EndFrame
[commandBuffer addPresent: drawable];
[commandBuffer commit];
commandBuffer = nil;
```

Render Setup

Do every render to texture pass

Create command encoder

Draw many times

- Update uniform buffers
- Set states
- Make draw calls

Render Setup

shadowMap pass render encoding

```
// Create encoder
id<MTLRenderCommandEncoder> encoder = [commandBuffer
    renderCommandEncoderWithDescriptor: shadowMapPassDesc];
```

Render Setup

shadowMap pass render encoding

```
// Create encoder
id<MTLRenderCommandEncoder> encoder = [commandBuffer
    renderCommandEncoderWithDescriptor: shadowMapPassDesc];

// Set states and draw
[encoder setRenderPipelineState: shadow_render_pipeline];
[encoder setDepthStencilState: shadowDepthStencilState];
…
[encoder setVertexBuffer: structureVertexBuffer offset:0 atIndex: 0];
[encoder drawIndexedPrimitives: …
```

Render Setup

shadowMap pass render encoding

```
// Create encoder
id<MTLRenderCommandEncoder> encoder = [commandBuffer
    renderCommandEncoderWithDescriptor: shadowMapPassDesc];

// Set states and draw
[encoder setRenderPipelineState: shadow_render_pipeline];
[encoder setDepthStencilState: shadowDepthStencilState];
…
[encoder setVertexBuffer: structureVertexBuffer offset:0 atIndex: 0];
[encoder drawIndexedPrimitives: …
```

Render Setup

shadowMap pass render encoding

```
// Create encoder
id<MTLRenderCommandEncoder> encoder = [commandBuffer
    renderCommandEncoderWithDescriptor: shadowMapPassDesc];

// Set states and draw
[encoder setRenderPipelineState: shadow_render_pipeline];
[encoder setDepthStencilState: shadowDepthStencilState];
…
[encoder setVertexBuffer: structureVertexBuffer offset:0 atIndex: 0];
[encoder drawIndexedPrimitives: …

// end encoding
[encoder endEncoding];
```

Render Setup

shadowMap pass render encoding

```
// Create encoder
id<MTLRenderCommandEncoder> encoder = [commandBuffer
    renderCommandEncoderWithDescriptor: shadowMapPassDesc];

// Set states and draw
[encoder setRenderPipelineState: shadow_render_pipeline];
[encoder setDepthStencilState: shadowDepthStencilState];
…
[encoder setVertexBuffer: structureVertexBuffer offset:0 atIndex: 0];
[encoder drawIndexedPrimitives: …

// end encoding
[encoder endEncoding];
```

Render Setup

Deferred Lighting pass render encoding

```
deferredPassDesc.colorAttachments[0].texture = texture_from_drawable;
```

Render Setup

Deferred Lighting pass render encoding

```
deferredPassDesc.colorAttachments[0].texture = texture_from_drawable;

// Create encoder
id<MTLRenderCommandEncoder> encoder = [commandBuffer
    renderCommandEncoderWithDescriptor: deferredPassDesc];
```

Render Setup

Deferred Lighting pass render encoding

```
deferredPassDesc.colorAttachments[0].texture = texture_from_drawable;

// Create encoder
id<MTLRenderCommandEncoder> encoder = [commandBuffer
    renderCommandEncoderWithDescriptor: deferredPassDesc];

…

// End encoding
[encoder endEncoding];
```

Agenda

Introduction to Metal

Fundamentals of Metal

- Building a Metal application
- Metal shading language

Advanced Metal

- Deep dive into creating a graphics application with Metal
- Data-Parallel Computing with Metal
- Developer Tools Review

Data-Parallel Computing with Metal

Aaftab Munshi
GPU Software

Data-Parallel Computing with Metal

What you'll learn

Data-Parallel Computing with Metal

What you'll learn

What is data-parallel computing?

Data-Parallel Computing with Metal

What you'll learn

What is data-parallel computing?

Data-parallel computing in Metal

Data-Parallel Computing with Metal

What you'll learn

What is data-parallel computing?

Data-parallel computing in Metal

Writing data-parallel kernels in Metal

Data-Parallel Computing with Metal

What you'll learn

What is data-parallel computing?

Data-parallel computing in Metal

Writing data-parallel kernels in Metal

Executing kernels in Metal

Data-Parallel Computing

A brief introduction

Data-Parallel Computing

A brief introduction

Similar and independent computations on multiple data elements

Data-Parallel Computing

A brief introduction

Similar and independent computations on multiple data elements

Example—Blurring an image

- Same computation for each input
- All results are independent

Data-Parallelism in Metal

Data-Parallelism in Metal

Code that describes computation is called a kernel

Data-Parallelism in Metal

Code that describes computation is called a kernel

Independent computation instance

- Work-item

Data-Parallelism in Metal

Code that describes computation is called a kernel

Independent computation instance

- Work-item

Work-items that execute together

- Work-group
- Cooperate by sharing data
- Can synchronize execution

Data-Parallelism in Metal

Computation domain

Data-Parallelism in Metal

Computation domain

Number of dimensions

- 1D, 2D, or 3D

Data-Parallelism in Metal

Computation domain

Number of dimensions

- 1D, 2D, or 3D

For each dimension specify

- Number of work-items in work-group also known as work-group size
- Number of work-groups

Data-Parallelism in Metal

Computation domain

Number of dimensions

- 1D, 2D, or 3D

For each dimension specify

- Number of work-items in work-group also known as work-group size
- Number of work-groups

Choose the dimensions that are best for your algorithm

Pseudo Code for a Data-Parallel Kernel

```
void
square(const float* input,
       float* output,
       uint id
{
    output[id] = input[id] * input[id];
}
```

Pseudo Code for a Data-Parallel Kernel

```
#include <metal_stdlib>
using namespace metal;
void
square(const float* input,
       float* output,
       uint id
{
    output[id] = input[id] * input[id];
}
```

Pseudo Code for a Data-Parallel Kernel

```
#include <metal_stdlib>
using namespace metal;
kernel void
square(const float* input,
       float* output,
       uint id
{
    output[id] = input[id] * input[id];
}
```

Pseudo Code for a Data-Parallel Kernel

```
#include <metal_stdlib>
using namespace metal;
kernel void
square(const global float* input [[ buffer(0) ]],
       global float* output [[ buffer(1) ]],
       uint id
{
    output[id] = input[id] * input[id];
}
```

Metal Kernel

```
#include <metal_stdlib>
using namespace metal;
kernel void
square(const global float* input [[ buffer(0) ]],
       global float* output [[ buffer(1) ]],
       uint id [[ global_id ]])
{
    output[id] = input[id] * input[id];
}
```

Another Kernel Example in Metal

Using images in a kernel

Another Kernel Example in Metal

Using images in a kernel

```
kernel void
horizontal_reflect(texture2d<float> src [[ texture(0) ]],
                   texture2d<float, access::write> dst [[ texture(1) ]],
                   uint2 id [[ global_id ]])
{
    float4 c = src.read(uint2(src.get_width()-1-id.x, id.y));
    dst.write(c, id);
}
```

Another Kernel Example in Metal

Using images in a kernel

```
kernel void
horizontal_reflect(texture2d<float> src [[ texture(0) ]],
                   texture2d<float, access::write> dst [[ texture(1) ]],
                   uint2 id [[ global_id ]])
{
    float4 c = src.read(uint2(src.get_width()-1-id.x, id.y));
    dst.write(c, id);
}
```

Another Kernel Example in Metal

Using images in a kernel

```
kernel void
horizontal_reflect(texture2d<float> src [[ texture(0) ]],
                   texture2d<float, access::write> dst [[ texture(1) ]],
                   uint2 id [[ global_id ]])
{
    float4 c = src.read(uint2(src.get_width()-1-id.x, id.y));
    dst.write(c, id);
}
```

Another Kernel Example in Metal

Using images in a kernel

```
kernel void
horizontal_reflect(texture2d<float> src [[ texture(0) ]],
                   texture2d<float, access::write> dst [[ texture(1) ]],
                   uint2 id [[ global_id ]])
{
    float4 c = src.read(uint2(src.get_width()-1-id.x, id.y));
    dst.write(c, id);
}
```

Another Kernel Example in Metal

Using images in a kernel

```
kernel void
horizontal_reflect(texture2d<float> src [[ texture(0) ]],
                   texture2d<float, access::write> dst [[ texture(1) ]],
                   uint2 id [[ global_id ]])
{
    float4 c = src.read(uint2(src.get_width()-1-id.x, id.y));
    dst.write(c, id);
}
```

Another Kernel Example in Metal

Using images in a kernel

```
kernel void
horizontal_reflect(texture2d<float> src [[ texture(0) ]],
                   texture2d<float, access::write> dst [[ texture(1) ]],
                   uint2 id [[ global_id ]])
{
    float4 c = src.read(uint2(src.get_width()-1-id.x, id.y));
    dst.write(c, id);
}
```

Built-in Kernel Variables

Attributes for kernel arguments

```
kernel void
my_kernel(texture2d<float> img [[ texture(0) ]],
          ushort2 gid [[ global_id ]],
          uint glinear [[ global_linear_id ]],
          ushort2 lid [[ local_id ]],
          ushort linear [[ local_linear_id ]],
          uint wgid [[ work_group_id ]],
          …)
{
    …
}
```

Built-in Kernel Variables

Attributes for kernel arguments

```
kernel void
my_kernel(texture2d<float> img [[ texture(0) ]],
          ushort2 gid [[ global_id ]],
          uint glinear [[ global_linear_id ]],
          ushort2 lid [[ local_id ]],
          ushort linear [[ local_linear_id ]],
          uint wgid [[ work_group_id ]],
          …)
{
    …
}
```

Built-in Kernel Variables

Attributes for kernel arguments

```
kernel void
my_kernel(texture2d<float> img [[ texture(0) ]],
          ushort2 gid [[ global_id ]],
          uint glinear [[ global_linear_id ]],
          ushort2 lid [[ local_id ]],
          ushort linear [[ local_linear_id ]],
          uint wgid [[ work_group_id ]],
          …)
{
    …
}
```

Built-in Kernel Variables

Attributes for kernel arguments

```
kernel void
my_kernel(texture2d<float> img [[ texture(0) ]],
          ushort2 gid [[ global_id ]],
          uint glinear [[ global_linear_id ]],
          ushort2 lid [[ local_id ]],
          ushort linear [[ local_linear_id ]],
          uint wgid [[ work_group_id ]],
          …)
{
    …
}
```

Built-in Kernel Variables

Attributes for kernel arguments

```
kernel void
my_kernel(texture2d<float> img [[ texture(0) ]],
          ushort2 gid [[ global_id ]],
          uint glinear [[ global_linear_id ]],
          ushort2 lid [[ local_id ]],
          ushort linear [[ local_linear_id ]],
          uint wgid [[ work_group_id ]],
          …)
{
    …
}
```

Built-in Kernel Variables

Attributes for kernel arguments

```
kernel void
my_kernel(texture2d<float> img [[ texture(0) ]],
          ushort2 gid [[ global_id ]],
          uint glinear [[ global_linear_id ]],
          ushort2 lid [[ local_id ]],
          ushort linear [[ local_linear_id ]],
          uint wgid [[ work_group_id ]],
          …)
{
    …
}
```

Built-in Kernel Variables

Attributes for kernel arguments

```
kernel void
my_kernel(texture2d<float> img [[ texture(0) ]],
          ushort2 gid [[ global_id ]],
          uint glinear [[ global_linear_id ]],
          ushort2 lid [[ local_id ]],
          ushort linear [[ local_linear_id ]],
          uint wgid [[ work_group_id ]],
          …)
{
    …
}
```

Executing Kernels in Metal

Post-processing example

Post-Processing Kernel

Kernel source

Post-Processing Kernel

Kernel source

```
kernel void
postprocess_filter(texture2d<float> inImage [[ texture(0) ]],
                   texture2d<float, access::write> outImage [[ texture(1) ]],
                   texture2d<float> curveImage [[ texture(2) ]],
                   constant Parameters& param [[ buffer(0) ]],
                   uint2 gid [[ global_id ]])
{
    // Transform global ID using param.transformMatrix

    float4 color = inImage.sample(s, transformedCoord);

    // Apply post-processing effect

    outImage.write(color, gid);
}
```

Post-Processing Kernel

Kernel source

```
kernel void
postprocess_filter(texture2d<float> inImage [[ texture(0) ]],
                   texture2d<float, access::write> outImage [[ texture(1) ]],
                   texture2d<float> curveImage [[ texture(2) ]],
                   constant Parameters& param [[ buffer(0) ]],
                   uint2 gid [[ global_id ]])
{
    // Transform global ID using param.transformMatrix

    float4 color = inImage.sample(s, transformedCoord);

    // Apply post-processing effect

    outImage.write(color, gid);
}
```

Post-Processing Kernel

Kernel source

```
kernel void
postprocess_filter(texture2d<float> inImage [[ texture(0) ]],
                   texture2d<float, access::write> outImage [[ texture(1) ]],
                   texture2d<float> curveImage [[ texture(2) ]],
                   constant Parameters& param [[ buffer(0) ]],
                   uint2 gid [[ global_id ]])
{
    // Transform global ID using param.transformMatrix

    float4 color = inImage.sample(s, transformedCoord);

    // Apply post-processing effect

    outImage.write(color, gid);
}
```

Post-Processing Kernel

Kernel source

```
kernel void
postprocess_filter(texture2d<float> inImage [[ texture(0) ]],
                   texture2d<float, access::write> outImage [[ texture(1) ]],
                   texture2d<float> curveImage [[ texture(2) ]],
                   constant Parameters& param [[ buffer(0) ]],
                   uint2 gid [[ global_id ]])
{
    // Transform global ID using param.transformMatrix

    float4 color = inImage.sample(s, transformedCoord);

    // Apply post-processing effect

    outImage.write(color, gid);
}
```

Post-Processing Kernel

Processing multiple pixels/work-item

```
constexpr constant int num_pixels_work_item = 4;

kernel void
postprocess_filter(…, uint2 gid [[global_id]],
                      uint2 lsize [[local_size]])
{
    for (int i=0; i<num_pixels_work_item; i++)
    {
        uint2 gid_new = uint2(gid.x+i*lsize.x, gid.y);

        // Transform gid_new using param.transformMatrix

        // Read from input image
        float4 color = inImage.sample(s, transformedCoord);

        // apply post-processing effect

        // Write to output image
        outImage.write(color, gid_new);
    }
}
```

Post-Processing Kernel

Processing multiple pixels/work-item

```
constexpr constant int num_pixels_work_item = 4;

kernel void
postprocess_filter(…, uint2 gid [[global_id]],
                      uint2 lsize [[local_size]])
{
    for (int i=0; i<num_pixels_work_item; i++)
    {
        uint2 gid_new = uint2(gid.x+i*lsize.x, gid.y);

        // Transform gid_new using param.transformMatrix

        // Read from input image
        float4 color = inImage.sample(s, transformedCoord);

        // apply post-processing effect

        // Write to output image
        outImage.write(color, gid_new);
    }
}
```

Post-Processing Kernel

Processing multiple pixels/work-item

```
constexpr constant int num_pixels_work_item = 4;

kernel void
postprocess_filter(…, uint2 gid [[global_id]],
                      uint2 lsize [[local_size]])
{
    for (int i=0; i<num_pixels_work_item; i++)
    {
        uint2 gid_new = uint2(gid.x+i*lsize.x, gid.y);

        // Transform gid_new using param.transformMatrix

        // Read from input image
        float4 color = inImage.sample(s, transformedCoord);

        // apply post-processing effect

        // Write to output image
        outImage.write(color, gid_new);
    }
}
```

Executing a Kernel

Compute command encoder

Executing a Kernel

Compute command encoder

```
// Load library and kernel function
id <MTLLibrary> library = [device newLibraryWithFile:libname error:&err];
id <MTLFunction> filterFunc = [library
                               newFunctionWithName:@“postprocess_filter”];
```

Executing a Kernel

Compute command encoder

```
// Load library and kernel function
id <MTLLibrary> library = [device newLibraryWithFile:libname error:&err];
id <MTLFunction> filterFunc = [library
                               newFunctionWithName:@“postprocess_filter”];

// Create compute state
id <MTLComputePipelineState> filterKernel =
    [device newComputePipelineStateWithFunction:filterFunc error:&err];
```

Executing a Kernel

Compute command encoder

```
// Load library and kernel function
id <MTLLibrary> library = [device newLibraryWithFile:libname error:&err];
id <MTLFunction> filterFunc = [library
                               newFunctionWithName:@“postprocess_filter”];

// Create compute state
id <MTLComputePipelineState> filterKernel =
    [device newComputePipelineStateWithFunction:filterFunc error:&err];

// Create compute command encoder
```

Executing a Kernel

Compute command encoder

```
// Load library and kernel function
id <MTLLibrary> library = [device newLibraryWithFile:libname error:&err];
id <MTLFunction> filterFunc = [library
                                newFunctionWithName:@“postprocess_filter”];

// Create compute state
id <MTLComputePipelineState> filterKernel =
    [device newComputePipelineStateWithFunction:filterFunc error:&err];

// Create compute command encoder
id <MTLComputeCommandEncoder> computeEncoder =
                                    [commandBuffer computeCommandEncoder];
```

Executing a Kernel

Encode compute commands

Executing a Kernel

Encode compute commands

```
// Set compute state
        [computeEncoder setComputePipelineState:filterKernel];
```

Executing a Kernel

Encode compute commands

```
// Set compute state
        [computeEncoder setComputePipelineState:filterKernel];

// Set Resources used by kernel
[computeEncoder setTexture:inputImage atIndex:0];
[computeEncoder setTexture:outputImage atIndex:1];
[computeEncoder setTexture:curveImage atIndex:2];
[computeEncoder setBuffer:params offset:0 atIndex:0];
```

Executing a Kernel

Encode compute commands

Executing a Kernel

Encode compute commands

```
// Calculate the work-group size and number of work-groups
        MTLSize wgSize = { 16, 16, 1 };
        MTLSize numWorkGroups = {
                                    (outputImage.width + wgSize - 1)/wgSize.x,
                                    (outputImage.height + wgSize - 1)/wgSize.y,
                                    1
                                };
```

Executing a Kernel

Encode compute commands

```
// Calculate the work-group size and number of work-groups
        MTLSize wgSize = { 16, 16, 1 };
        MTLSize numWorkGroups = {
                                  (outputImage.width + wgSize - 1)/wgSize.x,
                                  (outputImage.height + wgSize - 1)/wgSize.y,
                                  1
                                };
```

Executing a Kernel

Encode compute commands

```
// Calculate the work-group size and number of work-groups
        MTLSize wgSize = { 16, 16, 1 };
        MTLSize numWorkGroups = {
                                    (outputImage.width + wgSize - 1)/wgSize.x,
                                    (outputImage.height + wgSize - 1)/wgSize.y,
                                    1
                                  };

// Execute Kernel
        [computeEncoder executeKernelWithWorkGroupSize:wgSize
                                    workGroupCount:numWorkGroups];
```

Executing a Kernel

Encode compute commands

```
// Calculate the work-group size and number of work-groups
        MTLSize wgSize = { 16, 16, 1 };
        MTLSize numWorkGroups = {
                                    (outputImage.width + wgSize - 1)/wgSize.x,
                                    (outputImage.height + wgSize - 1)/wgSize.y,
                                    1
                                  };

    // Execute Kernel
        [computeEncoder executeKernelWithWorkGroupSize:wgSize
                                     workGroupCount:numWorkGroups];
```

Executing a Kernel

Encode compute commands

```
// Calculate the work-group size and number of work-groups
        MTLSize wgSize = { 16, 16, 1 };
        MTLSize numWorkGroups = {
                                    (outputImage.width + wgSize - 1)/wgSize.x,
                                    (outputImage.height + wgSize - 1)/wgSize.y,
                                    1
                                };

    // Execute Kernel
        [computeEncoder executeKernelWithWorkGroupSize:wgSize
                                     workGroupCount:numWorkGroups];

    // Finish encoding
        [computeEncoder endEncoding];
```

Executing a Kernel

Submit commands to the GPU

```
// Commit the command buffer
        [commandBuffer commit];
```

Demo

Post-processing kernels

Agenda

Introduction to Metal

Fundamentals of Metal

- Building a Metal application
- Metal shading language

Advanced Metal

- Deep dive into creating a graphics application with Metal
- Data-Parallel Computing with Metal
- Developer Tools Review

Tools

Debugging and profiling Metal applications in Xcode

Summary

Summary

A deeper dive into Metal

- Structuring your application for Metal
- Using descriptors and state objects for rendering
- Multi-pass encoding in Metal

Summary

A deeper dive into Metal

- Structuring your application for Metal
- Using descriptors and state objects for rendering
- Multi-pass encoding in Metal

Data-parallel computing in Metal

- How data-parallelism works in Metal
- Write and execute kernels in Metal

Summary

A deeper dive into Metal

- Structuring your application for Metal
- Using descriptors and state objects for rendering
- Multi-pass encoding in Metal

Data-parallel computing in Metal

- How data-parallelism works in Metal
- Write and execute kernels in Metal

Tools

- How to create and compile Metal Shaders in Xcode
- Debug and profile a Metal application

More Information

Filip Iliescu
Graphics and Games Technologies Evangelist
filiescu@apple.com

Allan Schaffer
Graphics and Games Technologies Evangelist
aschaffer@apple.com

Documentation
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

• Working with Metal—Overview	Pacific Heights	Wednesday 9:00AM
• Working with Metal—Fundamentals	Pacific Heights	Wednesday 10:15AM

Labs

• Metal Lab	Graphics and Games Lab A	Wednesday 2:00PM
• Metal Lab	Graphics and Games Lab B	Thursday 10:15AM

WWDC14