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 MetalKit
• Metal Performance Shaders

Metal Performance Optimization Techniques
• Metal System Trace Tool
• Metal Best Practices
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 MetalKit
- Metal Performance Shaders

Metal Performance Optimization Techniques
- Metal System Trace Tool
- Metal Best Practices
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 MetalKit
- Metal Performance Shaders

Metal Performance Optimization Techniques
- Metal System Trace Tool
- Metal Best Practices
MetalKit
Utility functionality for Metal Apps
MetalKit
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

Frame 1 | Frame 2 | Frame 3 | Frame 4
---|---|---|---
Application | | |
GPU | | |
Display | | |
Application Frame
Application Frame

[MTKView currentRenderPassDescriptor] Reserves a drawable
[MTKView `currentRenderPassDescriptor`]

Reserves a drawable

Encodes to the drawable
Application Frame

[MTKView currentRenderPassDescriptor]

- Reserves a drawable

[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

- Encodes to the drawable
- Releases the drawable
Application Frame

[MTKView currentRenderPassDescriptor]

Reserves a drawable

[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

Encodes to the drawable

Releases the drawable
Application Frame

- [MTKView currentRenderPassDescriptor]
  - Reserves a drawable

- [MTLCommandBuffer presentDrawable:]
  - Encodes to the drawable

- [MTLCommandBuffer commit]
  - Releases the drawable
Application Frame

[MTKView currentRenderPassDescriptor]

Reserves a drawable

[MTLCommandBuffer presentDrawable:]

Encodes to the drawable

[MTLCommandBuffer commit]

 Releases the drawable
Application Frame

[MTKView currentRenderPassDescriptor]

Reserves a drawable

[MTLCommandBuffer presentDrawable:] [MTLCommandBuffer commit]

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
Application Frame

- **[MTKView currentRenderPassDescriptor]**
  - Reserves a drawable

- **[MTLCommandBuffer presentDrawable:]**
  - Encodes to the drawable

- **[MTLCommandBuffer commit]**
  - Releases the drawable
Application Frame

[MTKView currentRenderPassDescriptor]

Reserves a drawable

[MTLCommandBuffer presentDrawable:]
[MTLCommandBuffer commit]

Encodes to the drawable

[MTLCommandBuffer commit]

Releases the drawable
Application Frame

MTKView currentRenderPassDescriptor

Reserves a drawable

MTLCommandBuffer presentDrawable:

Encodes to the drawable

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];
}

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
MetalKit View Drawing

Efficient usage

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

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

2. Load texture

```objectivec
[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

1. Metal Render State Pipeline
2. Model I/O Asset Initialization
3. MetalKit Mesh and Submesh Objects
4. Metal Render State Setup and Drawing
Rendering a Model I/O Asset

- Metal Render State Pipeline
- Model I/O Asset Initialization
- MetalKit Mesh and Submesh Objects
- Metal Render State Setup and Drawing
Rendering a Model I/O Asset

- Metal Render State Pipeline
- Model I/O Asset Initialization
- MetalKit Mesh and Submesh Objects
- Metal Render State Setup and Drawing
Rendering a Model I/O Asset

1. Metal Render State Pipeline
2. Model I/O Asset Initialization
3. MetalKit Mesh and Submesh Objects
4. Metal Render State Setup and Drawing
Rendering a Model I/O Asset

Metal Render State Pipeline → Model I/O Asset Initialization → MetalKit Mesh and Submesh Objects → Metal Render State Setup and Drawing
Setup for Model Rendering
Pipeline setup

Metal Render State Pipeline → Model I/O Asset Initialization → MetalKit Mesh and Submesh Objects → Metal Render State Setup and Drawing
Setup for Model Rendering
Pipeline setup

Vertex Descriptor
Metal Render State Pipeline
Model I/O Asset Initialization
MetalKit Mesh and Submesh Objects
Metal Render State Setup and Drawing
Vertex Shader

Setting up Per-Vertex Inputs

```c
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

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

text vertex VertexOutput

text vertexFunction(VertexInput current [[ stage_in ]])
{
    ...
}
```
Vertex Shader

Setting up Per-Vertex Inputs

```cpp
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) ]];
};

vertex VertexOutput
vertexFunction(VertexInput current [[ stage_in ]])
{
    ...
}
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

```c
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;
```
Vertex Descriptor

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

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

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

```c
metalVertexDesc.attributes[1].format = MTLVertexFormatUChar4Normalized;
m 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

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

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

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

```
metalVertexDesc.attributes[2].format = MTLVertexFormatHalfFloat2;
metalVertexDesc.attributes[2].offset = 16;
```
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) ]];
};
Vertex Descriptor

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

descriptor.attributes[2].format = MTLVertexFormatHalfFloat2;
descriptor.attributes[2].offset = 16;
Vertex Descriptor

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

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

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

```
metalVertexDesc.layouts[0].stride = 20;
```
Vertex Array Layout

0: position
  Float3

1: color
  UChar4

2: texUV
  HalfFloat2
Vertex Array Layout

0: position
  Float3

1: color
  UCHAR4

2: texUV
  HALF_FLOAT2
Vertex Array Layout
Vertex Array Layout
renderPipelineDescriptor.vertexDescriptor = metalVertexDesc;
MTLRenderStatePipeline *pipeline =
    [device.newRenderPipelineStateWithDescriptor:renderPipelineDescriptor
    error:error];
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

- **Vertex Descriptor**
- **Metal Render State Pipeline**
- **Model I/O Asset Initialization**
- **MetalKit Mesh and Submesh Objects**
- **Metal Render State Setup and Drawing**
Setup for Model Rendering
Asset initialization

1. Vertex Descriptor
2. Metal Render State Pipeline
3. Model I/O Asset Initialization
4. MetalKit Mesh and Submesh Objects
5. Metal Render State Setup and Drawing
Setup for Model Rendering
Asset initialization

- Vertex Descriptor
- Model I/O Asset Initialization
- MetalKit Mesh and Submesh Objects
- Metal Render State Setup and Drawing
Setup for Model Rendering
Asset initialization

1. Vertex Descriptor
2. Model I/O Asset Initialization
   - MetalKit Mesh Buffer Allocator
3. MetalKit Mesh and Submesh Objects
4. Metal Render State Setup and Drawing
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
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

0 : position
   Float3

1 : color
   UChar4

2 : texUV
   HalfFloat2
Asset Initialization

MDLVertexDescriptor *modelIOVertexDesc =
    MTKModelIOVertexFormatFromMetal(metalVertexDesc);

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

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

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

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

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

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

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

modelIOVertexDesc.attributes[0].name = MDLVertexAttributePosition;
modelIOVertexDesc.attributes[1].name = MDLVertexAttributeColor;
modelIOVertexDesc.attributes[2].name = MDLVertexAttributeTextureCoordinate;
MTKMeshBufferAllocator *mtkBufferAllocator =
[[MTKMeshBufferAllocator alloc] initWithDevice:metalDevice];
Asset Initialization

Creating the Asset Object

```swift
MDLAsset *asset = [[MDLAsset alloc] initWithURL:fileURL
vertexDescriptor:modelioVertexDesc
bufferAllocator:mtkBufferAllocator];
```
MDLAsset *asset = [[MDLAsset alloc] initWithURL:\texttt{fileURL} \\
vertexDescriptor:modelioVertexDesc \\
bufferAllocator:mtkBufferAllocator];
Asset Initialization

Creating the Asset Object

```swift
MDLAsset *asset = [[MDLAsset alloc] initWithURL:fileURL
    vertexDescriptor:modelioVertexDesc
    bufferAllocator:mtkBufferAllocator];
```
MDLAsset *asset = [[MDLAsset alloc] initWithURL:fileURL
vertexDescriptor:modelioVertexDesc
bufferAllocator:mtkBufferAllocator];
MetalKit Meshes Initialization
Setup for model rendering

- Vertex Descriptor
- Model I/O Asset Initialization
- MetalKit Mesh Buffer Allocator
- MetalKit Mesh and Submesh Objects
- Metal Render State Setup and Drawing
MetalKit Meshes Initialization
Setup for model rendering

- Vertex Descriptor
- Model I/O Asset Initialization
- MetalKit Mesh Buffer Allocator
- MetalKit Mesh and Submesh Objects
- Metal Render State Setup and Drawing
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```objective-c
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```objc
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
```
NSArray<MTKMesh *> *meshes = [MTKMesh meshesFromAsset:asset device:device];
Mesh Rendering with Metal
Setup for model rendering

- Vertex Descriptor
- Metal Render State Pipeline
- Model I/O Asset Initialization
- MetalKit Mesh Buffer Allocator
- MetalKit Mesh and Submesh Objects
- Metal Render State Setup and Drawing
Mesh Rendering with Metal
Setup for model rendering

- Vertex Descriptor
- Model I/O Asset Initialization
- MetalKit Mesh and Submesh Objects
- Metal Render State Setup and Drawing
NSUInteger bufferIndex = 0;
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {
    if(vertexBuffer.buffer != nil) {
        [renderEncoder setVertexBuffer:vertexBuffer.buffer
                      offset:vertexBuffer.offset
                   atIndex:bufferIndex];
    }
    bufferIndex++;
}
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++;
}"
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++;
}
NSUInteger bufferIndex = 0;
for(MTKMeshBuffer *vertexBuffer in mesh.vertexBuffers) {
    if(vertexBuffer.buffer != nil) {
        [renderEncoder setVertexBuffer:vertexBuffer.buffer
                    offset:vertexBuffer.offset
               atIndex:bufferIndex];
    }
    bufferIndex++;
}
for(MTKSubmesh *submesh in mesh.submeshes) {
    [renderEncoder drawIndexedPrimitives:submesh.primitiveType
     indexCount:submesh.indexCount
     indexType:submesh.indexType
     indexBuffer:submesh.indexBuffer.buffer
     indexBufferOffset:submesh.indexBuffer.offset];
}
for(MTKSubmesh *submesh in mesh.submeshes)
{
    [renderEncoder drawIndexedPrimitives:submesh.primitiveType
     indexCount:submesh.indexCount
     indexType:submesh.indexType
     indexBuffer:submesh.indexBuffer.buffer
     indexBufferOffset:submesh.indexBuffer.offset];
}
for(MTKSubmesh *submesh in mesh.submeshes) {
  [renderEncoder drawIndexedPrimitives:submesh.primitiveType
   indexCount:submesh.indexCount
   indexType:submesh.indexType
   indexBuffer:submesh.indexBuffer.buffer
   indexBufferOffset:submesh.indexBuffer.offset];
}
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

Introduction

A framework of data-parallel algorithms for the GPU
CPU-style library for the GPU
Metal Performance Shaders

Introduction

Optimized for iOS

Available in iOS 9 for the A8 processor
Metal Performance Shaders

Introduction

Designed to integrate easily into your Metal applications
As simple as calling a library function
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
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
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

```swift
// 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

```swift
// 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

```swift
// Create a filter object
MPSImageGaussianBlur *blurFilter =
    [[MPSImageGaussianBlur alloc]
    initWithFrameWithDevice: 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);

// Encode filter to the command buffer
[blurFilter encodeToCommandBuffer: commandBuffer source: sourceTexture destination: destinationTexture];
Using Metal Performance Shaders

Use a ‘blur’ filter

```objc
// Create a filter object
MPSImageGaussianBlur *blurFilter = [[MPSImageGaussianBlur alloc] initWithDevice: device];
// Encode filter to the command buffer
[blurFilter encodeToCommandBuffer: commandBuffer source: sourceTexture destination: destinationTexture];
```
Download Sample Code

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 ‘offset’ and Destination ‘clipRect’

Filter properties
Source ‘offset’ and Destination ‘clipRect’

Filter properties
Source ‘offset’ and Destination ‘clipRect’
Filter properties
Source ‘offset’ and Destination ‘clipRect’

Filter properties
In-place Operation

Save memory

Source == Destination
In-place Operation

Save memory

Source == Destination
In-place Operation

Save memory

offset

clipRect

Source == Destination
In-place Operation

Save memory

offset

clipRect

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?

```c
// 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
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;
};
FPSCopyAllocator 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;
};
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
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 Schafer, Game Technologies Evangelist
aschafer@apple.com
<table>
<thead>
<tr>
<th>Session</th>
<th>Venue</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing 3D Assets with Model I/O</td>
<td>Mission</td>
<td>Tuesday 2:30PM</td>
</tr>
<tr>
<td>What’s New in Metal, Part 1</td>
<td>Presidio</td>
<td>Tuesday 3:30PM</td>
</tr>
<tr>
<td>Metal Performance Optimization Techniques</td>
<td>Pacific Heights</td>
<td>Friday 11:00AM</td>
</tr>
</tbody>
</table>
## Related Labs

<table>
<thead>
<tr>
<th>Metal Lab</th>
<th>Graphics D</th>
<th>Friday 12:00PM</th>
</tr>
</thead>
</table>
