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
Metal in Review
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading
Frame Time with CPU as the Bottleneck

- Application frame $N$
- GPU API frame $N$
- GPU Work frame $N-1$
Frame Time with CPU as the Bottleneck

0 ms

CPU

Application frame N

GPU API frame N

33.3 ms

GPU

GPU Work frame N-1

50 ms

GPU Idle Time
Metal Reduces GPU API Overhead

- CPU: Application (frame N)
- GPU: GPU Work (frame N-1)
Metal Reduces GPU API Overhead

- **Application frame N**
- **CPU Idle Time**
- **GPU Work frame N-1**

0 ms

20 ms

33.3 ms
Improve Your Game

CPU

Application
frame N

More Physics
More AI

CPU Idle Time

GPU

GPU Work
frame N-1
Or Issue More Draw Calls

0 ms

CPU

Application frame N  More Physics  More AI  Up to 10x More Draw Calls

GPU Work frame N-1

33.3 ms
Or Issue More Draw Calls

- Application \textit{frame N}
- More Physics
- More AI
- Up to 10x More Draw Calls

- GPU Work \textit{frame N-1}
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading

<table>
<thead>
<tr>
<th>Build Time</th>
<th>Load Time</th>
<th>Draw Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Never”</td>
<td>Infrequent</td>
<td>1000s per Frame</td>
</tr>
</tbody>
</table>

- Shader Compilation
- State Validation
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading

<table>
<thead>
<tr>
<th>Build Time</th>
<th>Load Time</th>
<th>Draw Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Never”</td>
<td>Infrequent</td>
<td>1000s per Frame</td>
</tr>
</tbody>
</table>

- Shader Compilation
- State Validation
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading

<table>
<thead>
<tr>
<th>Build Time</th>
<th>Load Time</th>
<th>Draw Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Never”</td>
<td>Infrequent</td>
<td>1000s per Frame</td>
</tr>
</tbody>
</table>

- Shader Compilation
- State Validation
Metal

- Dramatically reduced overhead
- Precompiled shaders
- Graphics and compute
- Efficient multithreading
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading
Metal

Dramatically reduced overhead
Precompiled shaders
Graphics and compute
Efficient multithreading
VAIN GLORY
ベイン グローリー
Broad Support for Metal
Tools Support

Frame Debugger
Shader Profiler
Shader Editor
State Inspector
Driver Instruments
API Analysis Tools
Metal OS X
Metal OS X

Minimal code change required for existing iOS applications
Metal OS X

Minimal code change required for existing iOS applications

• Device selection
Metal OS X

Minimal code change required for existing iOS applications

• Device selection
• Support for Discrete Memory
Metal OS X

Minimal code change required for existing iOS applications

- Device selection
- Support for Discrete Memory
- New texture formats for desktop GPUs
Adopting Metal on OS X

The Foundry

Jack Greasley
Courtesy of Adidas
Day One
Day One
Day Five
Day Five
Day Fifteen
Day Fifteen
Day Twenty
Day Twenty
Day Twenty-Five
Day Twenty-Five
What did we learn?
New Features
Device Selection
New Shader Constant Update APIs
New Compressed Texture Formats
Texture Barriers

New Texture Features
MetalKit
Layer Select
Metal Performance Shaders

Counting Occlusion Queries
New Memory Model

Multi-sample Depth resolves
Depth Clipping Support

GPU Family Sets
Draw and Compute Indirect
Metal System Trace Tool

Separate front/back stencil reference values
New Shader Constant Update APIs
Device Selection

New Texture Features

Counting Occlusion Queries

New Compressed Texture Formats

Multi-sample Depth resolves

Layer Select

Texture Barriers

MetalKit

New Memory Model

Draw and Compute Indirect

Depth Clipping Support

Metal Performance Shaders

GPU Family Sets

Metal System Trace

Separate front/back stencil reference values

Metal Performance Shaders

New Shader Constant Update APIs
Metal Feature Set Definitions
Feature sets represent a collection of capabilities by GPU generation.

iOS_GPUFamily2_v1
Metal Feature Set Definitions

Feature sets represent a collection of capabilities by GPU generation

- Prefix defines the platform

iOS_GPUFamily2_v1
Metal Feature Set Definitions

Feature sets represent a collection of capabilities by GPU generation

- Prefix defines the platform
- Family Name is unique to a hardware generation

iOS_GPUFamily2_v1
Metal Feature Set Definitions

Feature sets represent a collection of capabilities by GPU generation

- Prefix defines the platform
- Family Name is unique to a hardware generation
- Revision number can change as features are added over time

iOS_GPUFamily2_v1
Metal Feature Set Definitions

Simple query to see if device supports a given feature set
Metal Feature Set Definitions

Simple query to see if device supports a given feature set

[myMetalDevice supportsFeatureSet: iOS_GPUFamily2_v1]
## iOS Metal Feature Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>Introduced</th>
<th>Feature Additions</th>
<th>Supported Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS_GPUFamily1_v1</td>
<td>iOS 8</td>
<td>Core Metal features for A7 devices</td>
<td>iPhone 5s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>iPad Air</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>iPhone 6 and 6 Plus</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>iPad Air 2</td>
</tr>
<tr>
<td>iOS_GPUFamily1_v2</td>
<td>iOS 9</td>
<td>New texture features, Multi-sample depth resolves, Depth clipping support, Separate stencil reference values</td>
<td>iPhone 6 and 6 Plus</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>iPad Air 2</td>
</tr>
<tr>
<td>iOS_GPUFamily2_v1</td>
<td>iOS 8</td>
<td>ASTC texture support</td>
<td>iPad Air 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>iPhone 6 and 6 Plus</td>
</tr>
<tr>
<td>iOS_GPUFamily2_v2</td>
<td>iOS 9</td>
<td>New texture features, Multi-sample depth resolves, Depth clipping support, Separate stencil reference values</td>
<td>iPad Air 2</td>
</tr>
</tbody>
</table>
## OS X Metal Feature Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>Introduced</th>
<th>Feature Additions</th>
<th>Supported Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSX_GPUFamily1_v1</td>
<td>OS X 10.11</td>
<td>Same core feature as iOS, plus BCn texture compression formats, combined depth stencil formats, managed resource model, multi-GPU device selection, draw and compute indirect counting occlusion queries, layer select, texture barriers, new texture features, multi-sample depth resolves, depth clipping support, separate stencil reference values</td>
<td>All Macs since 2012</td>
</tr>
</tbody>
</table>
Shader Constant Updates
Shader Constant Updates

Command Buffer

Draw 1

Draw 2

Draw 3

Draw n
Shader Constant Updates

- Shader Constants 1
- Constant Data for Draw 2
- Constant Data for Draw 3
- Constant Data for Draw n

Draw 1 → Draw 2 → Draw 3 → Draw n → Command Buffer
Shader Constant Updates

Constant Buffer

- Shader Constants 1
- Constant Data for Draw 2
- Constant Data for Draw 3
- Constant Data for Draw n

Command Buffer

- Draw 1
- Draw 2
- Draw 3
- Draw n
Shader Constant Updates

Constant Buffer
- Shader Constants 1
- Constant Data for Draw 2
- Constant Data for Draw 3
- Constant Data for Draw n

Command Buffer
- Draw 1
- Draw 2
- Draw 3
- Draw n
Shader Constant Updates

Bind per draw

id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer

    [render_pass setVertexBuffer:constant_buffer offset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
Shader Constant Updates
Bind per draw

```swift
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer

    [render_pass setVertexBuffer:constant_buffer offset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
```
Shader Constant Updates
Bind per draw

id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer
    [render_pass setVertexBuffer:constant_buffer offset:i*sizeof(MyConstants) atIndex:0];

    // draw
}

Shader Constant Updates

Bind per draw

```c
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer

    [render_pass setVertexBuffer:constant_buffer offset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
```
Shader Constant Updates
Bind per draw

```swift
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer

    [render_pass setVertexBuffer:constant_buffer offset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
```
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer

    [render_pass setVertexBuffer:constant_buffer offset:i*sizeof(MyConstants) atIndex:0];

    // draw
}

Shader Constant Updates
Bind per draw
Shader Constant Updates

Bind per draw

```c
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
```
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
Shader Constant Updates

Bind per draw

```c
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
```
Shader Constant Updates

Bind per draw

```cpp
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex: 0];

    // draw
}
```
Shader Constant Updates

Bind per draw

```c
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

[renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer

    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
```
for (i=0; i<draw_count; i++)
{
    MyConstants constants = // generate constants onto the stack
    [renderpass setVertexBytes:&constants length:sizeof(MyConstants) atIndex:0];

    // draw
}
Shader Constant Updates

Internal constant buffer managed by Metal

for (i=0; i<draw_count; i++)
{
    MyConstants constants = // generate constants onto the stack
    [renderpass setVertexBytes:&constants length:sizeof(MyConstants) atIndex:0];

    // draw
}
New Memory Model
New Memory Model

Support both unified and Discrete Memory model with minimal code change
New Memory Model

Support both unified and Discrete Memory model with minimal code change

New storage modes to specify where the resource should reside

- Shared storage mode
- Private storage mode
- Managed storage mode
Shared Storage Mode

Introduced with iOS 8

Full coherency between CPU and GPU at command buffer boundaries
Private Storage Mode

New with iOS 9 and OS X

Resources are only accessible to GPU—blit, render, compute

Metal can store data more optimally for the GPU
Private Storage Mode with Discrete Memory
Managed Storage Mode

New with OS X

Metal manages where the resource resides

Performance of private memory with convenience of shared
Managed Storage Mode

No extra copy for unified memory systems
Managed Storage Mode

CPU modified data

App must notify Metal when modifying a resource with CPU

```swift
[myBuffer didModifyRange:...];
[myTexture replaceRegion:...];
```
Managed Storage Mode

CPU read back

App must synchronize resource before CPU read

```objective-c
[blitCmdEncoder synchronizeResource:myBuffer];
[cmdBuffer waitUntilCompleted]; // Or use completion handler
contents = [myBuffer contents];
```

```objective-c
[blitCmdEncoder synchronizeResource:myTexture];
[cmdBuffer waitUntilCompleted]; // Or use completion handler
[myTexture getBytes:...];
```
id <MTLBuffer> constant_buffer = ...;
MyConstants* constant_ptr = constant_buffer.contents;

[renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];

for (i=0; i<draw_count; i++)
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
Constant Updates on Discrete Systems
Using Managed Buffers

Fast and easy shader constant uploads

```swift
id <MTLBuffer> constant_buffer = [device newBufferWithOptions:MTLResourceStorageModeManaged
length:kMyConstantBufferSize];
MyConstants* constant_ptr = constant_buffer.contents;

[renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];

foreach i in draw_count
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
[constant_buffer didModifyRange:NSRangeMake(0, i*sizeof(MyConstants))];
```
Fast and easy shader constant uploads

```c
id <MTLBuffer> constant_buffer = [device newBufferWithOptions:MTLResourceStorageModeManaged
  length:kMyConstantBufferSize];

MyConstants* constant_ptr = constant_buffer.contents;

[renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];

foreach i in draw_count
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}

[constant_buffer didModifyRange:NSRangeMake(0, i*sizeof(MyConstants))];
```
Fast and easy shader constant uploads

```swift
id <MTLBuffer> constant_buffer = [device newBufferWithOptions:MTLResourceStorageModeManaged
length:kMyConstantBufferSize];

MyConstants* constant_ptr = constant_buffer.contents;

[renderpass setVertexBuffer: constant_buffer offset: 0 atIndex: 0];

foreach i in draw_count
{
    constant_ptr[i] = // write constants directly into the buffer
    [renderpass setVertexBufferOffset:i*sizeof(MyConstants) atIndex:0];

    // draw
}
[constant_buffer didModifyRange:NSRangeMake(0, i*sizeof(MyConstants))];
```
Metal Managed Memory Model

Default storage modes

Buffers are shared

Default texture storage mode depends on platform

- iOS default is shared
- OS X default is managed
Customizing Storage Modes

Private textures

Use private for GPU-only resources

```objective-c
fbTextureDesc = [MTLTextureDescriptor texture2DDescriptorWithPixelFormat:myColorFormat
  width:myWidth
  height:myHeight
  mipmapped:NO];

fbTextureDesc.storageMode = MTLStorageModePrivate;

fbTexture = [device newTextureWithDescriptor:fbTextureDesc];
```
Customizing Storage Modes

Private textures

Use private for GPU-only resources

```objective-c
fbTextureDesc = [MTLTextureDescriptor texture2DDescriptorWithPixelFormat:myColorFormat
    width:myWidth
    height:myHeight
   .mipmap:NO];

fbTextureDesc.storageMode = MTLStorageModePrivate;

fbTexture = [device newTextureWithDescriptor:fbTextureDesc];
```
Device Selection
On multi-GPU systems

Use MTLCreateSystemDefaultDevice

- Picks the device connected to the main display
- Activates the discrete GPU on systems with automatic graphics switching
Device Selection

Selecting the Auxiliary GPU on a Mac Pro

New `MTLCopyAllDevices` API to enumerate all Metal capable devices

- 'headless' property identifies auxiliary GPU

```swift
id <MTLDevice> aux_gpu = nil;
for (id <MTLDevice> device in MTLCopyAllDevices()) {
    if ([device isHeadless]) {
        aux_gpu = device;
        break;
    }
}
```
Device Selection

Selecting the Auxiliary GPU on a Mac Pro

New MTLCopyAllDevices API to enumerate all Metal capable devices

- ‘headless’ property identifies auxiliary GPU

```swift
id <MTLDevice> aux_gpu = nil;
for (id <MTLDevice> device in MTLCopyAllDevices())
{
    if ([device isHeadless]) {
        aux_gpu = device;
        break;
    }
}
```
Device Selection

Selecting the Auxiliary GPU on a Mac Pro

New MTLCopyAllDevices API to enumerate all Metal capable devices

- ‘headless’ property identifies auxiliary GPU

```objective-c
id <MTLDevice> aux_gpu = nil;
for (id <MTLDevice> device in MTLCopyAllDevices()) {
    if ([device isHeadless]) {
        aux_gpu = device;
        break;
    }
}
```
Device Selection

Selecting the ‘best’ device in a dual-GPU MacBook Pro

Ideal for applications that are not full-screen games and want to optimize for power
Device Selection

Selecting the ‘best’ device in a dual-GPU MacBook Pro

Ideal for applications that are not full-screen games and want to optimize for power

Register for a GPU ‘switch’ notification

• NSViewGlobalFrameDidChangeNotification
Device Selection
Selecting the ‘best’ device in a dual-GPU MacBook Pro

Ideal for applications that are not full-screen games and want to optimize for power

Register for a GPU ‘switch’ notification

• NSViewGlobalFrameDidChangeNotification

Use the CoreGraphics convenience API to query the current active device

• Pass in the current display your view is on

```
#include <CoreGraphics/CGDirectDisplayMetal.h>

id <MTLDevice> device = CGDirectDisplayCopyCurrentMetalDevice(display);
```
Layered Rendering

Rasterize to multiple layers of a texture

- Slices of a 2D array texture
- Plane of a 3D texture
- Face of a cube texture
Layered Rendering

Rasterize to multiple layers of a texture
- Slices of a 2D array texture
- Plane of a 3D texture
- Face of a cube texture

Output the target layer from a vertex shader

```c
struct VSOut {
    float4 position [[position]];
    ushort layer [[render_target_array_index]];
}
```
Texture Barriers

GPUs overlap execution of many draw calls
Output of one draw cannot be safely read by a later draw
New API to insert barriers between draw calls
Texture Barriers

GPUs overlap execution of many draw calls
Output of one draw cannot be safely read by a later draw
New API to insert barriers between draw calls
Texture Barriers

Example

// start render pass, drawing to Texture A
[renderPass draw...];

[renderPass setFragmentTexture:textureA atIndex:0];
[renderPass draw...];
Texture Barriers

Example

// start render pass, drawing to Texture A
[renderPass draw...];

[renderPass textureBarrier];

[renderPass setFragmentTexture:textureA atIndex:0];
[renderPass draw...];
## New Texture Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>iOS GPUFamily1</th>
<th>iOS GPUFamily2</th>
<th>OS X GPUFamily1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Number of Textures per Shader Stage</td>
<td>31</td>
<td>31</td>
<td>128</td>
</tr>
<tr>
<td>Max Texture Size</td>
<td>8k</td>
<td>8k</td>
<td>16k</td>
</tr>
<tr>
<td>Max Render Target Count</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>MSAA</td>
<td>2x, 4x</td>
<td>2x, 4x</td>
<td>4x, 8x</td>
</tr>
<tr>
<td>Cube Array Support</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Compute Pixel Writes</td>
<td>Int32, Float32</td>
<td>Int32, Float32</td>
<td>Int32, Float32, packed</td>
</tr>
</tbody>
</table>
New Texture Features

Texture usage

New property in the texture descriptor to declare how a texture will be used.
Allows the Metal implementation to optimize for that usage.

- MTLTextureUsageUnknown
- MTLTextureUsageShaderRead
- MTLTextureUsageShaderWrite
- MTLTextureUsageRenderTarget
- MTLTextureUsageBlit
New Texture Features

Texture usage

New property in the texture descriptor to declare how a texture will be used
Allows the Metal implementation to optimize for that usage

MTLTextureUsageUnknown
MTLTextureUsageShaderRead
MTLTextureUsageShaderWrite
MTLTextureUsageRenderTarget
MTLTextureUsageBlit
New Texture Features

Texture usage

New property in the texture descriptor to declare how a texture will be used
Allows the Metal implementation to optimize for that usage

MTLTextureUsageUnknown
MTLTextureUsageShaderRead
MTLTextureUsageShaderWrite
MTLTextureUsageRenderTarget
MTLTextureUsageBlit
New Texture Features

Depth/stencil textures

Mac GPUs only support combined depth and stencil formats

- Depth32Float_stencil8
  - Supported on all Metal Devices
- Depth24Unorm_stencil8
  - If available and meets your precision requirements
# New Texture Features

iOS texture compression formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Bits Per Pixel</th>
<th>Support</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVRTC</td>
<td>2, 4</td>
<td>All iOS devices</td>
<td>RGB content, Widest support</td>
</tr>
<tr>
<td>ETC2</td>
<td>4 - RGB, 8 - RGBA</td>
<td>All Metal devices</td>
<td>RGB content, Good alpha support</td>
</tr>
<tr>
<td>EAC</td>
<td>4 - One channel, 8 - Two channels</td>
<td>All Metal devices</td>
<td>Height/Bump Maps, Normal Maps, Alpha Masks</td>
</tr>
<tr>
<td>ASTC</td>
<td>0.9 - 8</td>
<td>iOS GPUFamily2</td>
<td>Highest quality at all sizes, Many size vs. quality options, Slowest encoding</td>
</tr>
</tbody>
</table>
# New Texture Features

## iOS texture compression formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Bits Per Pixel</th>
<th>Support</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVRTC</td>
<td>2, 4</td>
<td>All iOS devices</td>
<td>RGB content, Widest support</td>
</tr>
<tr>
<td>ETC2</td>
<td>4 - RGB, 8 - RGBA</td>
<td>All Metal devices</td>
<td>RGB content, Good alpha support</td>
</tr>
<tr>
<td>EAC</td>
<td>4 - One channel, 8 - Two channels</td>
<td>All Metal devices</td>
<td>Height/Bump Maps, Normal Maps, Alpha Masks</td>
</tr>
<tr>
<td>ASTC</td>
<td>0.9 - 8</td>
<td>iOS GPUFamily2</td>
<td>Highest quality at all sizes, Many size vs. quality options, Slowest encoding</td>
</tr>
</tbody>
</table>
New Texture Features

ASTC format
New Texture Features

ASTC format

Very high-quality compression
New Texture Features

ASTC format

Very high-quality compression
Great for a broad range of usages

• Photographic content
• Height maps
• Normal maps
• Sprites
New Texture Features

ASTC format

Very high-quality compression
Great for a broad range of usages

- Photographic content
- Height maps
- Normal maps
- Sprites

Finer grained control of size vs. quality
- 1-8 bits per pixel
New Texture Features

ASTC format

Very high-quality compression

Great for a broad range of usages

- Photographic content
- Height maps
- Normal maps
- Sprites

Finer grained control of size vs. quality

- 1-8 bits per pixel

New in GPUFamily2
# New Texture Features

### OS X texture compression formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Bits Per Pixel</th>
<th>Also Known As</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC1</td>
<td>4</td>
<td>S3TC, DXT1</td>
<td>RGB content</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Very fast encoding</td>
</tr>
<tr>
<td>BC2, BC3</td>
<td>8</td>
<td>S3TC, DXT3, DXT5</td>
<td>RGBA content</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Very fast encoding</td>
</tr>
<tr>
<td>BC4, BC5</td>
<td>4 - One channel</td>
<td>RGTC</td>
<td>Height/Bump Maps</td>
</tr>
<tr>
<td></td>
<td>8 - Two channels</td>
<td></td>
<td>Normal Maps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Alpha Masks</td>
</tr>
<tr>
<td>BC6, BC7</td>
<td>8</td>
<td>BPTC</td>
<td>RGBA content</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slowest encoding</td>
</tr>
</tbody>
</table>
Metal and App Thinning
Art Pipeline

Game Assets

Game Binary

Game Binary

Game Assets

App Store

Game Assets

Legacy Assets
Art Pipeline

Game Binary

App Store

Legacy Assets

Legacy Assets

Game Binary

Legacy Assets

Game Binary

Legacy Assets

Legacy

Metal Capable
With App Thinning, only the assets applicable to the device are downloaded on install.
# Capability Matrix

<table>
<thead>
<tr>
<th></th>
<th>512MB</th>
<th>1GB</th>
<th>2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal GPUFamily2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal GPUFamily1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenGL ES Legacy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Capability Matrix

Typical normal map example

<table>
<thead>
<tr>
<th>Metal GPUMFamily1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL ES Legacy</td>
</tr>
</tbody>
</table>
## Capability Matrix

### Typical normal map example

<table>
<thead>
<tr>
<th>Metal GPUFamily1</th>
<th>OpenGL ES Legacy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>512x512 EAC</td>
</tr>
<tr>
<td>Capability Matrix</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>Typical normal map example</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metal GPUFamily1</th>
<th>512x512 EAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL ES Legacy</td>
<td>512x512 RG8</td>
</tr>
</tbody>
</table>
## Capability Matrix

**Typical normal map example**

<table>
<thead>
<tr>
<th>Metal GPUPFamily1</th>
<th>512x512 EAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL ES Legacy</td>
<td>512x512 RGB</td>
</tr>
</tbody>
</table>
## Capability Matrix

Extended normal map example

<table>
<thead>
<tr>
<th>Format</th>
<th>512MB</th>
<th>1GB</th>
<th>2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUFamily2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUFamily1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenGL ES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legacy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>256x256</td>
<td>RGB</td>
</tr>
<tr>
<td>512x512</td>
<td>RG8</td>
</tr>
<tr>
<td>512x512</td>
<td>EAC</td>
</tr>
<tr>
<td>512x512</td>
<td>ASTC</td>
</tr>
<tr>
<td>1024x1024</td>
<td>ASTC</td>
</tr>
</tbody>
</table>
## Capability Matrix

### Extended normal map example

<table>
<thead>
<tr>
<th>Format</th>
<th>512MB</th>
<th>1GB</th>
<th>2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal GPUFamily2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal GPUFamily1</td>
<td></td>
<td></td>
<td>1024x1024 ASTC</td>
</tr>
<tr>
<td>OpenGL ES Legacy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 256x256 RGB
- 512x512 RG8
- 512x512 EAC
- 512x512 ASTC
## Capability Matrix

### Extended normal map example

<table>
<thead>
<tr>
<th>Format</th>
<th>512MB</th>
<th>1GB</th>
<th>2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal GPUFamily2</td>
<td>512x512 ASTC</td>
<td>1024x1024 ASTC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal GPUFamily1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenGL ES Legacy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legacy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Capability Matrix

**Extended normal map example**

<table>
<thead>
<tr>
<th>Format</th>
<th>512MB</th>
<th>1GB</th>
<th>2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUFamily2</td>
<td>512x512 ASTC</td>
<td>1024x1024 ASTC</td>
<td></td>
</tr>
<tr>
<td>Metal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUFamily1</td>
<td>512x512 EAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenGL ES Legacy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **512x256 RGB**: 2GB
- **512x512 RG8**: 1GB
## Capability Matrix

### Extended normal map example

<table>
<thead>
<tr>
<th>Format</th>
<th>512MB</th>
<th>1GB</th>
<th>2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal GPUFamily2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>512x512 ASTC</td>
<td>1024x1024 ASTC</td>
<td></td>
</tr>
<tr>
<td>Metal GPUFamily1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>512x512 EAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenGL ES Legacy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>512x512 RG8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Capability Matrix

### Extended normal map example

<table>
<thead>
<tr>
<th>Format</th>
<th>512MB</th>
<th>1GB</th>
<th>2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Metal</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUFamily2</td>
<td>512x512</td>
<td>1024x1024</td>
<td>ASTC</td>
</tr>
<tr>
<td>GPUFamily1</td>
<td>512x512</td>
<td>EAC</td>
<td>ASTC</td>
</tr>
<tr>
<td><strong>OpenGL ES</strong></td>
<td>256x256</td>
<td>RG8</td>
<td>RG8</td>
</tr>
<tr>
<td><strong>Legacy</strong></td>
<td>256x256</td>
<td>RG8</td>
<td>RG8</td>
</tr>
</tbody>
</table>
Custom Tools Pipelines
Custom Tools Pipelines

Publicly documented JSON file format
Easily integrated into custom toolchain
Retrieving Named Data
Retrieving Named Data

NSDataAsset class provides correctly matched data resource from Asset Catalog
Retrieving Named Data

NSDataAsset class provides correctly matched data resource from Asset Catalog

#import <UIKit/NSDataAsset.h>
Retrieving Named Data

NSDataAsset class provides correctly matched data resource from Asset Catalog

```cpp
#import <UIKit/NSDataAsset.h>

NSDataAsset *asset = [[NSDataAsset alloc] initWithName:@"NormalMaps"];
NSData *data = asset.data;
```
Art Pipeline

Game Binary

App Store

512x512 RG8

512x512 ASTC

512x512 EAC

512x512 RG8

512x512 ASTC

512x512 EAC

512x512 RG8

512x512 ASTC

512x512 EAC
Developers are using Metal to create next-generation games and professional apps.

Metal now available for OS X

New Xcode Metal tools

New API features in iOS 9 and OS X

Support Metal-specific assets with App Thinning
More Information

Metal Documentation and Videos
http://developer.apple.com/metal

Apple Developer Forums
http://developer.apple.com/forums

Developer Technical Support
http://developer.apple.com/support/technical

General Inquiries
Allan Schaffer, Game Technologies Evangelist
aschaffer@apple.com
### Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Venue</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in Metal, Part 2</td>
<td>Mission</td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>Metal Performance Optimization Techniques</td>
<td>Pacific Heights</td>
<td>Friday 11:00AM</td>
</tr>
<tr>
<td>Labs</td>
<td>Graphics, Games, and Media Lab C</td>
<td>Wednesday 11:00AM</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Metal Lab</td>
<td>Graphics, Games, and Media Lab D</td>
<td>Friday 12:00PM</td>
</tr>
</tbody>
</table>
apple WWDC15