Introducing Metal 2

Session 601

Michal Valient, GPU Software Engineer
Richard Schreyer, GPU Software Engineer
Make expensive things happen once
GPU in the driving seat
New experiences
VR with Metal 2
Hall 3
Wednesday 10:00AM
Using Metal 2 for Compute  

Grand Ballroom A  
Thursday 4:10PM
Using Metal 2 for Compute

Grand Ballroom A
Thursday 4:10PM
Introducing Metal 2

Agenda
Introducing Metal 2

Agenda

Argument Buffers
Introducing Metal 2

Agenda

Argument Buffers

Raster Order Groups
Introducing Metal 2

Agenda

Argument Buffers

Raster Order Groups

ProMotion Displays
Introducing Metal 2

Agenda

- Argument Buffers
- Raster Order Groups
- ProMotion Displays
- Direct to Display
Introducing Metal 2

Agenda

Argument Buffers
Raster Order Groups
ProMotion Displays
Direct to Display
Everything Else
Argument Buffers
Material Example

- roughness: 0.6
- intensity: 0.3
- surfaceTexture
- specularTexture
- sampler
Material Example

- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler
Traditional Argument Model

- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler
Traditional Argument Model

MTLBuffer

roughness intensity surfaceTexture specularTexture sampler
Traditional Argument Model

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

setBuffer

API calls

CPU time
Traditional Argument Model

MTLBuffer

- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls

setBuffer

setTexture

CPU time
Traditional Argument Model

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls:
- setBuffer
- setTexture
- setTexture

CPU time
Traditional Argument Model

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls:
- setBuffer
- setTexture
- setTexture
- setSampler

CPU time
Traditional Argument Model

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls:
- setBuffer
- setTexture
- setTexture
- setSampler
- draw

CPU time
Traditional Argument Model

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls

CPU time
Traditional Argument Model

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

Object 1
Object 2
Object 3

API calls

CPU time
Argument Buffers

MTLBuffer

roughness    intensity    surfaceTexture    specularTexture    sampler
Argument Buffers

MTLBuffer

- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler
Argument Buffers

MTLBuffer

- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls

CPU time
Argument Buffers

MTLBuffer

- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls

setBuffer

CPU time
Argument Buffers

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls
- setBuffer
- draw

CPU time
Argument Buffers

MTLBuffer
- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls
CPU time
Argument Buffers

MTLBuffer

- roughness
- intensity
- surfaceTexture
- specularTexture
- sampler

API calls

CPU time

Object 1  Object 2  Object 3  Object 4  Object 5  Object 6  Object 7  ...
Reduced CPU Overhead

The diagram illustrates the time (in microseconds, µs) for different resource levels on an iPhone 7. The Y-axis represents the time, with lower values indicating better performance. The X-axis shows resource levels, with 2, 8, and 16 resources as categories. The graph compares two models: Traditional model and Argument Buffers.

- Traditional model: Use case for 2 Resources, 8 Resources, and 16 Resources.
- Argument Buffers: Use case for 2 Resources, 8 Resources, and 16 Resources.

Lower times indicate better performance, with the Argument Buffers model generally performing better than the Traditional model across all resource levels.
Reduced CPU Overhead

<table>
<thead>
<tr>
<th>Resources</th>
<th>Traditional model</th>
<th>Argument Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.64</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4.53</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>7.97</td>
<td></td>
</tr>
</tbody>
</table>

iPhone 7

Time (µs) Lower is better
Reduced CPU Overhead

Time (µs) Lower is better

<table>
<thead>
<tr>
<th>Resources</th>
<th>Traditional model</th>
<th>Argument Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.64</td>
<td>0.23</td>
</tr>
<tr>
<td>8</td>
<td>4.53</td>
<td>0.24</td>
</tr>
<tr>
<td>16</td>
<td>7.97</td>
<td>0.25</td>
</tr>
</tbody>
</table>

iPhone 7
Argument Buffers

Benefits

Improve performance

Enable new use cases

Easy to use
Argument Buffers
Shader example

```cpp
struct Material {
    float roughness;
    float intensity;
    texture2d<float> surfaceTexture;
    texture2d<float> specularTexture;
    sampler textureSampler;
};

kernel void my_kernel(constant Material &material [[buffer(0)]]) {
    ...
}
```
struct Material
{
    float roughness;
    float intensity;
    texture2d<float> surfaceTexture;
    texture2d<float> specularTexture;
    sampler textureSampler;
};

kernel void my_kernel(constant Material &material [[buffer(0)]])
{
    ...
}

Dynamic Indexing
Crowd rendering
Dynamic Indexing
Crowd rendering

| MTLBuffer | position | height | ... | specularTexture | pantsTexture | ... |
Dynamic Indexing

Crowd rendering

MTLBuffer

character[0]

position  height  ...  specularTexture  pantsTexture  ...

CPU

GPU
### Dynamic Indexing

Crowd rendering

<table>
<thead>
<tr>
<th>MTLBuffer</th>
<th>character[0]</th>
<th>position</th>
<th>height</th>
<th>...</th>
<th>specularTexture</th>
<th>pantsTexture</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>character[1]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td></td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td>character[2]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td></td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CPU**

**GPU**
Dynamic Indexing
Crowd rendering

<table>
<thead>
<tr>
<th>MTLBuffer</th>
<th>character[0]</th>
<th>position</th>
<th>height</th>
<th>...</th>
<th>specularTexture</th>
<th>pantsTexture</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>character[1]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>character[2]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Dynamic Indexing
Crowd rendering

<table>
<thead>
<tr>
<th>MTLBuffer</th>
<th>character[0]</th>
<th>position</th>
<th>height</th>
<th>...</th>
<th>specularTexture</th>
<th>pantsTexture</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>character[1]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>character[2]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
</tbody>
</table>

...
Dynamic Indexing
Crowd rendering

<table>
<thead>
<tr>
<th>MTLBuffer</th>
<th>character[0]</th>
<th>position</th>
<th>height</th>
<th>...</th>
<th>specularTexture</th>
<th>pantsTexture</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>character[1]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>character[2]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
</tbody>
</table>

...
## Dynamic Indexing

**Crowd rendering**

<table>
<thead>
<tr>
<th>MTLBuffer</th>
<th>character[0]</th>
<th>position</th>
<th>height</th>
<th>...</th>
<th>specularTexture</th>
<th>pantsTexture</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>character[1]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>character[2]</td>
<td>position</td>
<td>height</td>
<td>...</td>
<td>specularTexture</td>
<td>pantsTexture</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- GPU
  - setBuffer
  - drawIndexedPrimitives: instanceCount: 1000

- CPU
  - Fragment Shader
    - use per-character textures
  - Vertex Shader
    - use per-character position
Dynamic Indexing
Vertex shader

```cpp
vertex VertexOutput instancedShader(uint id [[instance_id]],
constant Character *crowd [[buffer(0)]])
{
    // Dynamically pick the character for given instance index
    constant Character &obj = crowd[id];

    return transformCharacter(obj);
}
```
Dynamic Indexing
Vertex shader

```cpp
vertex VertexOutput instancedShader(uint id [[instance_id]],
                                     constant Character *crowd [[buffer(0)]])
{
    // Dynamically pick the character for given instance index
    constant Character &obj = crowd[id];

    return transformCharacter(obj);
}
```
vertex VertexOutput instancedShader(uint id [[instance_id]],
                                        constant Character *crowd [[buffer(0)]])
{
    // Dynamically pick the character for given instance index
    constant Character &obj = crowd[id];

    return transformCharacter(obj);
}
Set Resources on GPU
Particle simulation
Set Resources on GPU
Particle simulation

<table>
<thead>
<tr>
<th>MTLBuffer</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>particle[0]</td>
<td>orientation</td>
<td>material</td>
</tr>
<tr>
<td>particle[1]</td>
<td>orientation</td>
<td>material</td>
</tr>
<tr>
<td>particle[2]</td>
<td>orientation</td>
<td>material</td>
</tr>
</tbody>
</table>

...
Set Resources on GPU
Particle simulation

<table>
<thead>
<tr>
<th>MTLBuffer</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>particle[0]</td>
<td>orientation</td>
<td>material</td>
</tr>
<tr>
<td>particle[1]</td>
<td>orientation</td>
<td>material</td>
</tr>
<tr>
<td>particle[2]</td>
<td>orientation</td>
<td>material</td>
</tr>
<tr>
<td>. . .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Simulation kernel

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>thread[0]</td>
</tr>
<tr>
<td>thread[1]</td>
</tr>
<tr>
<td>thread[2]</td>
</tr>
<tr>
<td>. . .</td>
</tr>
</tbody>
</table>
Set Resources on GPU

Particle simulation

MTLBuffer

<table>
<thead>
<tr>
<th>particle[0]</th>
<th>orientation</th>
<th>material</th>
</tr>
</thead>
<tbody>
<tr>
<td>particle[1]</td>
<td>orientation</td>
<td>material</td>
</tr>
<tr>
<td>particle[2]</td>
<td>orientation</td>
<td>material</td>
</tr>
</tbody>
</table>

...
Set Resources on GPU
Particle simulation

MTLBuffer
- particle[0] orientation → material
- particle[1] orientation → material
- particle[2] orientation → material

Simulation kernel
- thread[0]
- thread[1]
- thread[2]

MTLBuffer
- material[0] rocks
- material[1] moss
- material[2] grass
Set Resources on GPU
Particle simulation

MTLBuffer
particle[0] orientation ... moss
particle[1] orientation ... material
particle[2] orientation ... material
...

Simulation kernel
thread[0]
thread[1]
thread[2]
...

MTLBuffer
material[0] rocks
material[1] moss
material[2] grass
...

Set Resources on GPU
Particle simulation

MTLBuffer

<table>
<thead>
<tr>
<th>particle[0]</th>
<th>orientation</th>
<th>...</th>
<th>moss</th>
</tr>
</thead>
<tbody>
<tr>
<td>particle[1]</td>
<td>orientation</td>
<td>...</td>
<td>rocks</td>
</tr>
<tr>
<td>particle[2]</td>
<td>orientation</td>
<td>...</td>
<td>material</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Simulation kernel

<table>
<thead>
<tr>
<th>thread[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread[1]</td>
</tr>
<tr>
<td>thread[2]</td>
</tr>
</tbody>
</table>

MTLBuffer

<table>
<thead>
<tr>
<th>material[0]</th>
<th>rocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>material[1]</td>
<td>moss</td>
</tr>
<tr>
<td>material[2]</td>
<td>grass</td>
</tr>
</tbody>
</table>

...
Set Resources on GPU
Particle simulation

MTLBuffer

<table>
<thead>
<tr>
<th>particle[0]</th>
<th>orientation</th>
<th>...</th>
<th>moss</th>
</tr>
</thead>
<tbody>
<tr>
<td>particle[1]</td>
<td>orientation</td>
<td>...</td>
<td>rocks</td>
</tr>
<tr>
<td>particle[2]</td>
<td>orientation</td>
<td>...</td>
<td>grass</td>
</tr>
</tbody>
</table>
...
Set Resources on GPU

Shader example

```cpp
struct Data {
    texture2d<float> tex;
    float value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device   Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;     // Assign constants
    dst.tex   = src.tex;  // Copy just a texture
    dst       = src;      // Copy entire structure
}
```
Set Resources on GPU
Shader example

```cpp
struct Data {
    texture2d<float> tex;
    float value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device Data &dst [[buffer(1)]])
{
    dst.value = 1.0f; // Assign constants
    dst.tex = src.tex; // Copy just a texture
    dst = src;        // Copy entire structure
}
```
struct Data {
    texture2d<float> tex;
    float value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device   Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;       // Assign constants
    dst.tex   = src.tex;    // Copy just a texture
    dst       = src;       // Copy entire structure
}
Set Resources on GPU
Shader example

```cpp
struct Data {
    texture2d<float> tex;
    float value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;  // Assign constants
dst.tex   = src.tex;  // Copy just a texture
dst      = src;      // Copy entire structure
}
```
Set Resources on GPU
Shader example

```cpp
struct Data {
    texture2d<float> tex;
    float value;
};

kernel void copy(constant Data &src [[buffer(0)]],
    device Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;  // Assign constants
    dst.tex = src.tex;  // Copy just a texture
    dst = src;  // Copy entire structure
}
```
Multiple Indirections

Argument Buffers can reference other Argument Buffers

Create and reuse complex object hierarchies

```c
struct Object {
    float4 position;
    device Material *material;  /// Many objects can point to the same material
};

struct Tree {
    device Tree *children[2];
    device Object *objects;  /// Array of objects in the node
};
```
**Multiple Indirections**

Argument Buffers can reference other Argument Buffers

Create and reuse complex object hierarchies

```c
struct Object {
    float4 position;
    device Material *material;     ///< Many objects can point to the same material
};

struct Tree {
    device Tree *children[2];
    device Object *objects;        ///< Array of objects in the node
};
```
Argument Buffers can reference other Argument Buffers

Create and reuse complex object hierarchies

```c
struct Object {
    float4 position;
    device Material *material;  ///< Many objects can point to the same material
};

struct Tree {
    device Tree *children[2];
    device Object *objects;  ///< Array of objects in the node
};
```
<table>
<thead>
<tr>
<th>Support Tiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU overhead reduction</td>
</tr>
<tr>
<td>Set resources on GPU</td>
</tr>
<tr>
<td>Multiple indirections</td>
</tr>
<tr>
<td>Arrays of Argument Buffers</td>
</tr>
<tr>
<td>Resources per draw call</td>
</tr>
</tbody>
</table>
## Support Tiers

<table>
<thead>
<tr>
<th></th>
<th>Tier 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU overhead reduction</td>
<td>Yes</td>
</tr>
<tr>
<td>Set resources on GPU</td>
<td>No</td>
</tr>
<tr>
<td>Multiple indirections</td>
<td>No</td>
</tr>
<tr>
<td>Arrays of Argument Buffers</td>
<td>No</td>
</tr>
<tr>
<td>Resources per draw call</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
## Support Tiers

<table>
<thead>
<tr>
<th>Feature</th>
<th>Tier 1</th>
<th>Tier 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU overhead reduction</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Set resources on GPU</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple indirections</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Arrays of Argument Buffers</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Resources per draw call</td>
<td>Unchanged</td>
<td>500,000 textures and buffers</td>
</tr>
</tbody>
</table>
Argument Buffers

Terrain rendering

Material changes with terrain

Trees placed by GPU

Particle materials generated by GPU

Available as sample code
API Highlights

Argument Buffers are stored in plain `MTLBuffer`

Use `MTLArgumentEncoder` to fill Indirect Argument Buffers

Abstracts platform differences behind simple interface

Up to eight Argument Buffers per stage
// Shader syntax

struct Particle {
    texture2d<float> surface;
    float4 position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) {
    ... }

// Shader syntax

struct Particle {
    texture2d<float> surface;
    float4 position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]) { ... }
// Shader syntax

```cpp
struct Particle {
    texture2d<float> surface;
    float4 position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }
```

// Create encoder for first indirect argument buffer in Metal function 'simulate'
```cpp
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
```
// Shader syntax

struct Particle {
    texture2d<float> surface;
    float4 position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }

// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
// Shader syntax
struct Particle {
  texture2d<float> surface;
  float4 position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }

// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
// Shader syntax

struct Particle {
    texture2d<float> surface;
    float4 position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]) { ... }

// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)

// API calls to fill the indirect argument buffer
particleEncoder.setTexture(mySurfaceTexture, at: 0)
particleEncoder.constantData(at: 1).storeBytes(of: myPosition, as: float4.self)
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4 position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }

// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)

// API calls to fill the indirect argument buffer
particleEncoder.setTexture(mySurfaceTexture, at: 0)
particleEncoder.constantData(at: 1).storeBytes(of: myPosition, as: float4.self)
Shader syntax

```cpp
struct Particle {
    texture2d<float> surface;
    float4 position;
};
```

```cpp
kernel void simulate(constant Particle &particle [[buffer(0)]]) {
    ...
}
```

Create encoder for first indirect argument buffer in Metal function 'simulate'

```cpp
let simulateFunction = library.makeFunction(name: "simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
```

API calls to fill the indirect argument buffer

```cpp
particleEncoder.setTexture(mySurfaceTexture, at: 0)
particleEncoder.constantData(at: 1).storeBytes(of: myPosition, as: float4.self)
```
Managing Resource Usage

Tell Metal what resources you plan to use

Use Metal Heaps for best performance

```swift
// Use for textures with sample access or buffers
commandEncoder.\texttt{use}(myTextureHeap)

// Used for all render targets, views or read/write access to texture
commandEncoder.\texttt{use}(myRenderTarget, usage: \texttt{.write})
```
Managing Resource Usage

Tell Metal what resources you plan to use

Use Metal Heaps for best performance

// Use for textures with sample access or buffers
commandEncoder.use(myTextureHeap)

// Used for all render targets, views or read/write access to texture
commandEncoder.use(myRenderTarget, usage: .write)
Managing Resource Usage

Tell Metal what resources you plan to use

Use Metal Heaps for best performance

```objective-c
// Use for textures with sample access or buffers
commandEncoder.use(myTextureHeap)

// Used for all render targets, views or read/write access to texture
commandEncoder.use(myRenderTarget, usage: .write)
```
Best Practices

Organize based on usage pattern
• Per-view vs. per-object vs. per-material
• Dynamically changing vs. Static

Favor data locality

Use traditional model where appropriate
Raster Order Groups

Richard Schreyer
Raster Order Groups

Ordered memory access from fragment shaders
Enables new rendering techniques
  • Order-independent transparency
  • Dual-layer GBuffers
  • Voxelization
  • Custom blending
Fragment Shaders with Blending

Time

Fragment Shader Thread 1  Blend
Fragment Shaders with Blending

Time

Fragment Shader Thread 1  Blend  Fragment Shader Thread 2  Blend
Fragment Shaders with Blending

- Fragment Shader Thread 1
- Blend

- Fragment Shader Thread 2
- Wait
- Blend
Mid-Shader Memory Access

Fragment Shader Thread 1
Write Memory

Fragment Shader Thread 2
Read Memory
With Raster Order Groups

Fragment Shader Thread 1
[Blue Triangle]
Write Memory

Fragment Shader Thread 2
[Red Triangle]
Wait
Read Memory

Time →
With Raster Order Groups

Fragment Shader Thread 1
- Write Memory

Fragment Shader Thread 2
- Wait
- Read Memory

Time
// Blending manually to a pointer in memory

fragment void BlendSomething(
    texture2d<float, access::read_write> framebuffer [[texture(0)]]
) {

    float4 newColor = ...

    // Non-atomic access to memory without synchronization
    float4 priorColor = framebuffer.read(framebufferPosition);
    float4 blended = custom_blend(newColor, priorColor);
    framebuffer.write(blended, framebufferPosition);
}
// Blending manually to a pointer in memory

fragment void BlendSomething(
  texture2d<float, access::read_write> framebuffer [[texture(0)]]
) {

  float4 newColor = ...  

  // Non-atomic access to memory without synchronization
  float4 priorColor = framebuffer.read(framebufferPosition);
  float4 blended = custom_blend(newColor, priorColor);
  framebuffer.write(blended, framebufferPosition);
}
// Blending manually to a pointer in memory

fragment void BlendSomething(
    texture2d<float, access::read_write> framebuffer [[texture(0)]]) {

    float4 newColor = ...

    // Non-atomic access to memory without synchronization
    float4 priorColor = framebuffer.read(framebufferPosition);
    float4 blended = custom_blend(newColor, priorColor);
    framebuffer.write(blended, framebufferPosition);
}
// Blending manually to a pointer in memory

fragment void BlendSomething(
    texture2d<float, access::read_write> framebuffer [[texture(0), raster_order_group(0)]]
) {

    float4 newColor = ...

    // Hardware waits on first access to raster ordered memory
    float4 priorColor = framebuffer.read(framebufferPosition);
    float4 blended = custom_blend(newColor, priorColor);
    framebuffer.write(blended, framebufferPosition);
}
With Raster Order Groups
With Raster Order Groups

Fragment Shader Thread 1
Write Memory

Fragment Shader Thread 2
Wait
Read Memory
Raster Order Groups

Summary

Synchronization between overlapping fragment shader threads

Check for support with `MTLDevice.rasterOrderGroupsSupported`
ProMotion Displays
Without ProMotion

60 FPS

16.6 ms
With ProMotion
120 FPS

GPU
1

Display
1

8.3ms
Without ProMotion

48 FPS
With ProMotion
48 FPS
Without ProMotion

Dropped frame

16.6ms  21ms  16.6ms

GPU

1  2  3

Display

1  2  3
With ProMotion
Dropped frame

16.6ms 21ms 16.6ms

GPU Display

1 2 3
1 2 3
With ProMotion

Dropped frame

16.6ms  21ms  16.6ms

GPU

1        2        3

1        2        3

16.6ms  21ms  16.6ms

Display
Opting in to ProMotion

UIKit animations use ProMotion automatically.

Metal views require opt-in with Info.plist key:

```xml
    <key>CADisableMinimumFrameDuration</key>
    <true/>
```
Metal Presentation APIs
Present immediately

present(drawable)
present(drawable, afterMinimumDuration: 1000.0 / 30.0)
let time: CFTimeInterval = projectNextDisplayTime();
present(drawable, atTime: time)
let targetTime = // project when intend to display this drawable
// render your scene into a command buffer for 'targetTime'
let drawable = metalLayer.nextDrawable()
commandBuffer.present(drawable, atTime: targetTime)

// after a frame or two...

let presentationDelay = drawable.presentedTime - targetTime
// Examine presentationDelay and adjust future frame timing
ProMotion Displays

Summary

120 FPS

Reduced latency

Improved framerate consistency

Reduced stuttering from missed display deadlines
Direct to Display
Displaying Metal Content

Two paths

GPU Composition

Direct to Display
GPU Composition

- Metal View
- Other UIKit/AppKit Views
- Other Application's Windows

System Compositor
- Scale
- Colorspace Conversion
- Composition

Display
Direct to Display Requirements
Direct to Display Requirements

☑️ Opaque Layer
Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on
Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on
- Full screen (or with “black bars” via an opaque black background color)
Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on
- Full screen (or with "black bars" via an opaque black background color)
- Dimensions matching the display or smaller
Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on
- Full screen (or with “black bars” via an opaque black background color)
- Dimensions matching the display or smaller
- Color Space and Pixel Format compatible with the display
# Colorspace Requirements

<table>
<thead>
<tr>
<th>Color Space</th>
<th>Metal Pixel Format</th>
<th>P3 Display</th>
<th>sRGB Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>sRGB</td>
<td>bgra8Unorm_srgb</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
# Colorspace Requirements

<table>
<thead>
<tr>
<th>Color Space</th>
<th>Metal Pixel Format</th>
<th>P3 Display</th>
<th>sRGB Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>sRGB</td>
<td>bgra8Unorm_srgb</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Display P3 (macOS)</td>
<td>bgr10a2Unorm</td>
<td>Direct</td>
<td>GPU Composited</td>
</tr>
</tbody>
</table>
## Colorspace Requirements

<table>
<thead>
<tr>
<th>Color Space</th>
<th>Metal Pixel Format</th>
<th>P3 Display</th>
<th>sRGB Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>sRGB</td>
<td>bgra8Unorm_srgb</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Display P3 (macOS)</td>
<td>bgr10a2Unorm</td>
<td>Direct</td>
<td>GPU Composited</td>
</tr>
<tr>
<td>Extended sRGB (iOS)</td>
<td>bgr10_xr_srgb</td>
<td>Direct</td>
<td>GPU Composited</td>
</tr>
</tbody>
</table>
## Colorspace Requirements

<table>
<thead>
<tr>
<th>Color Space</th>
<th>Metal Pixel Format</th>
<th>P3 Display</th>
<th>sRGB Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>sRGB</td>
<td>bgra8Unorm_srgb</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Display P3 (macOS)</td>
<td>bgr10a2Unorm</td>
<td>Direct</td>
<td>GPU Composited</td>
</tr>
<tr>
<td>Extended sRGB (iOS)</td>
<td>bgr10_xr_srgb</td>
<td>Direct</td>
<td>GPU Composited</td>
</tr>
<tr>
<td>Linear Extended sRGB, or Display P3</td>
<td>rgba16Float</td>
<td>GPU Composited</td>
<td>GPU Composited</td>
</tr>
</tbody>
</table>
Detecting P3 Display Gamut

UIKit

UITraitCollection.displayGamut == .P3

AppKit

NSScreen.canRepresent(.p3)
Direct to Display
Summary

Eliminate compositor usage of GPU
Useful for full-screen scenes
Supported on iOS, tvOS, and macOS
Use Metal System Trace to verify
Everything Else
Memory Usage Queries

New APIs to query memory usage per allocation
- MTLResource.allocatedSize
- MTLHeap.currentAllocatedSize

Query total GPU memory allocated by the device
- MTLDevice.currentAllocatedSize
SIMDGroup-scoped Data Sharing

Share data across a SIMDGroup

```
simd_broadcast(data, simd_lane_id)
simd_shuffle(data, simd_lane_id)
simd_shuffle_up(data, simd_lane_id)
simd_shuffle_down(data, simd_lane_id)
simd_shuffle_xor(data, simd_lane_id)
```
SIMDGroup-scoped Data Sharing

Share data across a SIMDGroup

```c
simd_broadcast(data, simd_lane_id)
simd_shuffle(data, simd_lane_id)
simd_shuffle_up(data, simd_lane_id)
simd_shuffle_down(data, simd_lane_id)
simd_shuffle_xor(data, simd_lane_id)
```
Non-Uniform Threadgroup Sizes
Non-Uniform Threadgroup Sizes

```
0  1  2  3  4  5  6  7  8  9 10 11
0
1
2
3
4
5
6
7

4x4 Threadgroup
4x4 Threadgroup
4x4 Threadgroup
4x4 Threadgroup
4x4 Threadgroup
4x4 Threadgroup
4x4 Threadgroup
4x4 Threadgroup
```
Non-Uniform Threadgroup Sizes

- 4x4 Threadgroup
- 4x4 Threadgroup
- 3x4 Threadgroup
- 4x3 Threadgroup
- 4x3 Threadgroup
- 3x3 Threadgroup
Viewport Arrays

Vertex Function selects which viewport

Useful for VR when combined with instancing
Multisample Pattern Control

Select where within a pixel the MSAA Sample Patterns are located

Useful for custom anti-aliasing
Multisample Pattern Control

Select where within a pixel the MSAA Sample Patterns are located

Useful for custom anti-aliasing

Toggle sample locations

1 Pixel
Resource Heaps

Now available on macOS

Control time of memory allocation

Fast reallocation and aliasing of resources

Group related resources for faster binding
Resource Heaps

Memory Allocation for A
  Texture A

Memory Allocation for B
  Texture B

Memory Allocation for C
  Texture C
Resource Heaps

- Memory Allocation for A
- Texture A

- Memory Allocation for B
- Texture B

- Memory Allocation for C
- Texture C

Heap

- Texture A
- Texture B
- Texture C
Resource Heaps

Memory Allocation for A

Texture A

Memory Allocation for B

Texture B

Heap

Texture A
Texture B
Resource Heaps

Memory Allocation for A
- Texture A

Memory Allocation for B
- Texture B

Memory Allocation for D
- Texture D

Heap
- Texture A
- Texture B
- Texture D
Resource Heaps

Memory Allocation for A
  Texture A

Memory Allocation for B
  Texture B

Memory Allocation for D
  Texture D

Heap
  Texture A
  Texture B
  Texture D
<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
</tr>
</thead>
</table>

Other Features
## Other Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Textures</td>
<td>Create textures from a MTLBuffer without copying</td>
</tr>
</tbody>
</table>
## Other Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Textures</td>
<td>Create textures from a MTLBuffer without copying</td>
</tr>
<tr>
<td>Function Constant for Argument Indexes</td>
<td>Specialize bytecodes to change the binding index for shader arguments</td>
</tr>
</tbody>
</table>
## Other Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Textures</td>
<td>Create textures from a MTLBuffer without copying</td>
</tr>
<tr>
<td>Function Constant for Argument Indexes</td>
<td>Specialize bytecodes to change the binding index for shader arguments</td>
</tr>
<tr>
<td>Additional Vertex Array Formats</td>
<td>Add some additional 1 component and 2 component vertex formats, and a BGRA8 vertex format</td>
</tr>
<tr>
<td>Feature</td>
<td>Summary</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>Linear Textures</td>
<td>Create textures from a MTLBuffer without copying</td>
</tr>
<tr>
<td>Function Constant for Argument Indexes</td>
<td>Specialize bytecodes to change the binding index for shader arguments</td>
</tr>
<tr>
<td>Additional Vertex Array Formats</td>
<td>Add some additional 1 component and 2 component vertex formats, and a BGRA8 vertex format</td>
</tr>
<tr>
<td>IOSurface Textures</td>
<td>Create MTLTextures from IOSurfaces on iOS</td>
</tr>
<tr>
<td>Feature</td>
<td>Summary</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>Linear Textures</td>
<td>Create textures from a MTLBuffer without copying</td>
</tr>
<tr>
<td>Function Constant for Argument Indexes</td>
<td>Specialize bytecodes to change the binding index for shader arguments</td>
</tr>
<tr>
<td>Additional Vertex Array Formats</td>
<td>Add some additional 1 component and 2 component vertex formats, and a BGRA8 vertex format</td>
</tr>
<tr>
<td>IOSurface Textures</td>
<td>Create MTLTextures from IOSurfaces on iOS</td>
</tr>
<tr>
<td>Dual Source Blending</td>
<td>Additional blending modes with two source parameters</td>
</tr>
</tbody>
</table>
Summary

Argument Buffers
Raster Order Groups
ProMotion Displays
Direct to Display
Everything Else
More Information

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR with Metal 2</td>
<td>Hall 3</td>
<td>Wednesday 10:00AM</td>
</tr>
<tr>
<td>Metal 2 Optimization and Debugging</td>
<td>Executive Ballroom</td>
<td>Thursday 3:10PM</td>
</tr>
<tr>
<td>Using Metal 2 for Compute</td>
<td>Grand Ballroom A</td>
<td>Thursday 4:10PM</td>
</tr>
</tbody>
</table>
## Previous Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>WWDC 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>What's New in Metal, Part 1</td>
<td>WWDC 2016</td>
</tr>
<tr>
<td>Working with Wide Color</td>
<td>WWDC 2016</td>
</tr>
<tr>
<td>Labs</td>
<td>Location</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Metal 2 Lab</td>
<td>Technology Lab A</td>
</tr>
<tr>
<td>VR with Metal 2 Lab</td>
<td>Technology Lab A</td>
</tr>
<tr>
<td>Metal 2 Lab</td>
<td>Technology Lab F</td>
</tr>
</tbody>
</table>