Metal for VR

Session 611

Karol Gasiński, GPU Software
What’s new in VR?
What’s new in VR?

New Metal features
What’s new in VR?

New Metal features

VR advanced techniques
VR on macOS
VR on macOS
VR on macOS
VR on macOS
VR on macOS
VR on macOS

Support for HTC Vive Pro
VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
VR on macOS

Support for HTC Vive Pro
• Two 1440x1600 AMOLED displays, at 615 ppi
• 78% resolution increase
VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
- 78% resolution increase
- 37% ppi increase
VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
- 78% resolution increase
- 37% ppi increase
- Stereo cameras for MR
VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
- 78% resolution increase
- 37% ppi increase
- Stereo cameras for MR

SteamVR Tracking System 2.0
VR on macOS
VR on macOS

Valve SteamVR runtime
VR on macOS

Valve SteamVR runtime
Valve OpenVR framework
Metal 2 Features for VR
VR Application

Rendering overview

App
- 2D MS Textures
- IOSurfaces

VR Compositor
- VR Compositor
- L
- R

App
- L
- R

2D MS Textures
- L
- R

IOSurfaces
- L
- R

VR Compositor
- L
- R

2D MS Textures
- L
- R

IOSurfaces
- L
- R

VR Compositor
- L
- R

VR Compositor
- L
- R
VR Application

Rendering overview

App

2D MS Textures

IOSurfaces

App

VR Compositor

VR Compositor

IOSurfaces

2D MS Textures

VR

Compositor
VR Application

Rendering overview

App

2D MS Textures

IOSurfaces

App

L

R

L

R

Resolve

Render

VR Compositor

VR Compositor

L

R

L

R

2D MS Textures

IOSurfaces

L

R

Resolve

Render

VR Compositor

L

R

L

R
VR Application

Rendering overview

![Diagram of VR Application with 2D MS Textures, IOSurfaces, and VR Compositor with L and R labels.]
VR Application
Rendering overview

Diagram showing the flow of data from the app to the VR compositor, including stages such as rendering, resolving, submitting, and warping.
VR Application

Rendering overview

App

- Render L R
- Resolve L R
- 2D MS Textures
- IOSurfaces

VR Compositor

- Submit
- Warp
- VR Compositor
- Present
Rendering Layouts
Different rendering patterns depending on MSAA
Rendering Layouts
Different rendering patterns depending on MSAA

Dedicated

```
L
R
```

Drawbacks
2x Draw calls
2x Render passes
2x Resolves
Rendering Layouts

Different rendering patterns depending on MSAA

Dedicated

Shared

Drawbacks
- 2x Draw calls
- 2x Render passes
- 2x Resolves
Rendering Layouts
Different rendering patterns depending on MSAA

Dedicated

L
R

Drawbacks
2x Draw calls
2x Render passes
2x Resolves

Shared

L
R

Drawbacks
Issues with Multi-resolution shading
Post-processes
Rendering Layouts
Different rendering patterns depending on MSAA

Dedicated

Shared

Layered

Drawbacks
2x Draw calls
2x Render passes
2x Resolves

Drawbacks
Issues with Multi-resolution shading Post-processes
Rendering Layouts
Different rendering patterns depending on MSAA

Dedicated
- Drawbacks
  - 2x Draw calls
  - 2x Render passes
  - 2x Resolves

Shared
- Drawbacks
  - Issues with Multi-resolution shading
  - Post-processes

Layered
- Drawbacks
  - Cannot be used today for MSAA
Rendering Layouts
Different rendering patterns unified
Rendering Layouts
Different rendering patterns unified

New Metal texture type
Rendering Layouts
Different rendering patterns unified

New Metal texture type
MTLTextureType2DMultisampleArray

2DMultisampleArray
Rendering Layouts
Different rendering patterns unified

New Metal texture type

```
MTLTextureType2DMultisampleArray
```

Separate control of

```
2DMultisampleArray
```
Rendering Layouts
Different rendering patterns unified

New Metal texture type

MTLTextureType2DMultisampleArray

Separate control of
• Rendering space
Rendering Layouts
Different rendering patterns unified

New Metal texture type
MTLTextureType2DMultisampleArray

Separate control of
• Rendering space
• Views count
Rendering Layouts
Different rendering patterns unified

New Metal texture type

**MTLTextureType2DMultisampleArray**

Separate control of
- Rendering space
- Views count
- Anti-aliasing mode
Rendering Layouts
Different rendering patterns unified

New Metal texture type

MTLTextureType2DMultisampleArray

Separate control of
• Rendering space
• Views count
• Anti-aliasing mode
Rendering Layouts
Different rendering patterns unified

New Metal texture type

MTLTextureType2DMultisampleArray

Separate control of
• Rendering space
• Views count
• Anti-aliasing mode

Single draw, render, and resolve pass

STEREO RENDERING

2DMultisampleArray

MSAA On

L
R

2DArray

MSAA Off

L
R

2DMultisample Array

NEW

MONO RENDERING

2DMultisample

Mono

2D
Application creates 2D MS Array texture for rendering VR content

MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width = width;
descriptor.height = height;
descriptor.sampleCount = 4u;
descriptor.arrayLength = 2u;
descriptor.storageMode = MTLStorageModePrivate;
descriptor.usage = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newTextureWithDescriptor:descriptor];
Application creates 2D MS Array texture for rendering VR content

```swift
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];

descriptor.textureType   = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 4u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newTextureWithDescriptor:descriptor];
```
Application creates 2D MS Array texture for rendering VR content

```swift
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 4u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newTextureWithDescriptor:descriptor];
```
Application creates 2D MS Array texture for rendering VR content

```objective-c
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 4u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newTextureWithDescriptor:descriptor];
```
VR Application
Rendering to 2D multisample array textures
VR Application
Rendering to 2D multisample array textures
Cross-Process Texture Sharing
IOSurfaces versus shareable Metal textures
Cross-Process Texture Sharing

IOSurfaces versus shareable Metal textures

Ability to share Metal textures between processes
Cross-Process Texture Sharing
IOSurfaces versus shareable Metal textures

Ability to share Metal textures between processes
Shareable Metal textures can have complex structure
Cross-Process Texture Sharing

IOSurfaces versus shareable Metal textures

Ability to share Metal textures between processes

Shareable Metal textures can have complex structure

Shareable texture can be used only in scope of single GPU
Cross-Process Texture Sharing
IOSurfaces versus shareable Metal textures

Ability to share Metal textures between processes

Shareable Metal textures can have complex structure

Shareable texture can be used only in scope of single GPU

Enable advanced VR use cases, but are not limited to them
Cross-Process Texture Sharing
Standard versus shareable texture creation

Application creates 2D Array texture that can be shared with Compositor

```objective-c
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType       = MTLTextureType2DArray;
descriptor.pixelFormat       = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width             = width;
descriptor.height            = height;
descriptor.sampleCount       = 1u;
descriptor.arrayLength       = 2u;
descriptor.storageMode       = MTLStorageModePrivate;
descriptor.usage             = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
```
Application creates 2D Array texture that can be shared with Compositor

```swift
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType = MTLTextureType2DArray;
descriptor.pixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width = width;
descriptor.height = height;
descriptor.sampleCount = 1u;
descriptor.arrayLength = 2u;
descriptor.storageMode = MTLStorageModePrivate;
descriptor.usage = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
```
Cross-Process Texture Sharing
Standard versus shareable texture creation

Application creates 2D Array texture that can be shared with Compositor

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType = MTLTextureType2DArray;
descriptor.pixelFormat = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width = width;
descriptor.height = height;
descriptor.sampleCount = 1u;
descriptor.arrayLength = 2u;
descriptor.storageMode = MTLStorageModePrivate;
descriptor.usage = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
```
Cross-Process Texture Sharing
Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```c
IOSurfaceRef   backingIOSurface[2];
id<MTLTexture> resolvedTexture[2]; // Two 2D textures (backed by IOSurfaces)
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
    textureDesc[i].eType       = vr::ETextureType::TextureType_IOSurface;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
    textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Passing frame for presentation, using shareable Metal texture

ID<MTLTexture> resolvedTexture[2]; // Two 2D textures (backed by IOSurfaces)

Cross-Process Texture Sharing

Passing texture to compositor

iosurfaceRef   backingIOSurface[2];

for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
    textureDesc[i].eType       = vr::ETextureType::TextureType_IOSurface;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_LINEAR;
    textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
Cross-Process Texture Sharing
Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture[2]; // Two 2D textures (backed by IOSurfaces)
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
    textureDesc[i].eType = vr::ETextureType::TextureType_IOSurface;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
    textureBounds[i] = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing

Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture[2];  // Two 2D textures (backed by IOSurfaces)

vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
    textureDesc[i].eType = vr::ETextureType::TextureType_IOSurface;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
    textureBounds[i] = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr::VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vvr::VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing

Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture;  // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
    textureDesc[i].eType = vr::ETextureType::TextureType_IOSurface;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
    textureBounds[i] = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}
vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vrl.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing
Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture;  // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
    textureDesc[i].eType = vr::ETextureType::TextureType_IOSurface;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
    textureBounds[i] = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}
vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing
Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture;       // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vk::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle = reinterpret_cast<void*>(intptr_t(resolvedTexture));
    textureDesc[i].eType = vk::ETextureType::TextureType_IOSurface;
    textureDesc[i].eColorSpace = vk::EColorSpace::ColorSpace_Linear;
    textureBounds[i] = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr::VRCompositor()->Submit(vk::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr::VRCompositor()->Submit(vk::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing
Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture;    // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(resolvedTexture));
    textureDesc[i].eType       = vr::ETextureType::TextureType_Metal;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
    textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}
vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing
Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture; // Shareable 2D Array texture
textureDesc[2];
textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle = reinterpret_cast<void*>(intptr_t(resolvedTexture));
textureDesc[i].eType = vr::ETextureType::TextureType_Metal;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i] = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };}

vr::VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr::VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing
Passing texture to compositor

Passing frame for presentation, using shareable Metal texture

```cpp
id<MTLTexture> resolvedTexture;  // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
    textureDesc[i].handle = reinterpret_cast<void*>(intptr_t(resolvedTexture));
    textureDesc[i].eType       = vr::ETextureType::TextureType_Metal;
    textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
    textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f }; 
}
vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```
Cross-Process Texture Sharing
Passing across process boundary

Process A

```objective-c
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

```objective-c
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```
Cross-Process Texture Sharing
Passing across process boundary

Process A

```objective-c
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

```objective-c
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```
Cross-Process Texture Sharing
Passing across process boundary

Process A

```objective-c
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

```objective-c
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```
Cross-Process Texture Sharing
Passing across process boundary

**Process A**

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor);
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

**Process B**

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```
Cross-Process Texture Sharing
Passing across process boundary

Process A

```objective-c
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

// Make sure you recreate your texture handle on the same GPU
```objective-c
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```
Cross-Process Texture Sharing
Passing across process boundary

Process A

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```
VR Application
Using shareable Metal textures

App
- App
- Render
- L
- Resolve
- IOSurfaces
- R
- 2D MS Array Texture
- VR Compositor
- Submit
- VR Compositor
- Warp
- L
- R
- Present
- VR Application
- Using shareable Metal textures
VR Application
Using shareable Metal textures

App

2D MS Array Texture
2D Array Textures

Resolve
Submit

VR Compositor

L
R

Warp

Present

2D MS Array Texture
2D Array Textures

NEW

VR Application
Using shareable Metal textures

App

2D MS Array Texture
2D Array Textures

Resolve
Submit

VR Compositor

L
R

Warp

Present

2D MS Array Texture
2D Array Textures

NEW

VR Application
Using shareable Metal textures

App

2D MS Array Texture
2D Array Textures

Resolve
Submit

VR Compositor

L
R

Warp

Present

2D MS Array Texture
2D Array Textures

NEW

VR Application
Using shareable Metal textures

App

2D MS Array Texture
2D Array Textures

Resolve
Submit

VR Compositor

L
R

Warp

Present

2D MS Array Texture
2D Array Textures

NEW

VR Application
Using shareable Metal textures

App

2D MS Array Texture
2D Array Textures

Resolve
Submit

VR Compositor

L
R

Warp

Present

2D MS Array Texture
2D Array Textures

NEW

VR Application
Using shareable Metal textures

App

2D MS Array Texture
2D Array Textures

Resolve
Submit

VR Compositor

L
R

Warp

Present

2D MS Array Texture
2D Array Textures

NEW
VR Application
Compositor optimizations

App

2D MS Array Texture
2D Array Textures

Render
Resolve

Submit

VR Compositor

Warp

2D Array Textures
Present

NEW
VR Application

Compositor optimizations

App

- 2D MS Array Texture
- 2D Array Textures

VR Compositor

- Submit
- VR Compositor
- Warp
- Present

NEW
Metal 2 Features for VR

Recap

Shareable Metal textures

2D multisample array textures

OpenVR SDK support
VR Advanced Techniques
Advanced Frame Pacing
Reducing Fill Rate
Frame Pacing

Single threaded application

![Diagram showing frame pacing with VBLs and CPU and Panel levels.]
Frame Pacing
Single threaded application

Running
Start

WaitGetPoses() returns
Frame Pacing
Single threaded application

90Hz—11.1ms

WaitGetPoses() returns
Running Start
Frame Pacing

Single threaded application

90Hz—11.1ms

WaitGetPoses() returns

Running

Start

Simulation

CPU

CPU

Panel
Frame Pacing

Single threaded application

90Hz—11.1ms

WaitGetPoses() returns

CPU

Simulation

Encode

Panel

GPU
Frame Pacing
Single threaded application

90Hz—11.1ms

WaitGetPoses() returns

Simulation
Encode

Render
Frame Pacing
Single threaded application

Running
Start

WaitGetPoses() returns

Simulation
Encode

90Hz—11.1ms

Render

vrcompositor

C

CPU
Panel

GPU

Encode

VBL VBL

Panel

CPU
Frame Pacing
Single threaded application

90Hz—11.1ms

CPU

Simulation

Compile

Encode

Render

Panel

Scanout

WaitGetPoses() returns
Frame Pacing
Single threaded application

90Hz—11.1ms

WaitGetPoses() returns → Running Start
Simulation → Encode
Render

vrcompositor → C

Scanout photons
Frame Pacing
Single threaded application

90Hz—11.1ms

WaitGetPoses() returns
Running Start
Simulation
Encode
vrcompositor
Render
C
Scanout
Prediction to Photons ~25ms
Returned by WaitGetPoses()
Frame Pacing

Single threaded application

Encode VBL VBL Prediction to Photons ~25ms Returned by WaitGetPoses()

Panel

Prediction to Photons ~25ms Returned by WaitGetPoses()
Frame Pacing
Single threaded application

CPU
- Simulation
- Encode
- Simulation
- Encode
- Simulation
- Encode
- Simulation
- Encode
- Simulation
- Encode

VR Compositor
- Render
- Render
- Render
- Render

Panel
- photons
- photons
- photons
- photons

Prediction to Photons ~25ms
Returned by WaitGetPoses()
Frame Pacing
Multi-threaded application

Simulation
Simulation
Simulation
Encode
Encode
Encode

Running Start

90Hz—11.1ms

CPU
Encode
Encode
Encode

Render
Render
Render

Panel
photons
photons
photons

Rendering Prediction to Photons ~25ms
Returned by WaitGetPoses()
Frame Pacing
Multi-threaded application

Simulation Prediction to Photons ~36 ms
Returned by GetDeviceToAbsoluteTrackingPose()

Rendering Prediction to Photons ~25 ms
Returned by WaitGetPoses()

90Hz—11.1 ms
Frame Pacing
Splitting command buffer encoding

- Running Start
- Simulation
- Encode
- Render
- C
- photons

CPU

Panel
Frame Pacing

Splitting command buffer encoding
Frame Pacing
Splitting command buffer encoding
Frame Pacing
Splitting command buffer encoding
Advanced Frame Pacing

Multi-threaded application

90Hz—11.1ms

Simulation Prediction to Photons ~36 ms
Returned by GetDeviceToAbsoluteTrackingPose()

Rendering Prediction to Photons ~25ms
Returned by WaitGetPoses()

Simulation

CPU

90Hz—11.1ms

Encode

Encoder

Render

Panel

photon
**Advanced Frame Pacing**

Multi-threaded application

90Hz—11.1ms

Simulation Prediction to Photons ~36 ms
Returned by GetDeviceToAbsoluteTrackingPose()

Rendering Prediction to Photons ~25ms
Returned by WaitGetPoses()
Advanced Frame Pacing
GPU workload examples

Companion / mirroring window
Physics, cloth, water simulation
Reflection maps
Frustum culling
Shadow maps
Occlusion queries
Rendering geometry
Final post-processing
Advanced Frame Pacing
GPU workload examples

WaitGetPoses()

Companion / mirroring window
Physics, cloth, water simulation
Reflection maps
Frustum culling
Shadow maps
Occlusion queries
Rendering geometry
Final post-processing
Advanced Frame Pacing

GPU workload examples

Companion / mirroring window
Physics, cloth, water simulation
Reflection maps
Frustum culling
Shadow maps
Occlusion queries
Rendering geometry
Final post-processing

Pose-independent workloads
WaitGetPoses()

Pose-dependent workloads
Advanced Frame Pacing

Pose independent work

CPU

Simulation
Encode
Render
Encode
Render
Encode
Render
Encode

Panel

photon

photon

photon
Advanced Frame Pacing
Pose independent work
Advanced Frame Pacing
Multi-GPU workload distribution
Advanced Frame Pacing
Multi-GPU workload distribution

CPU
- Simulation
- Encode
- Encode
- Encode

GPU
- Render
- Render
- Render
- Render

GPU 2
- Render
- Render
- Render

Panel
- photons
- photons
- photons
- photons
Advanced Frame Pacing

Multi-GPU workload distribution
Multi-GPU Workload Distribution

Synchronization primitives

MTLEvent
• Mechanism of synchronizing GPU workloads
• Synchronizes across Command Queues

MTLSharedEvent
• Extends Event
• Synchronizes across GPU’s
• Can be shared between processes
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // . . . Encode Pose-Independent work
[commandA ...]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA ...]; // ... Encode Pose-Independent work
[commandA ...]; // ... Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB ...]; // ... Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB ...]; // ... Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
id<MTEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB ...]; // ... Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB ...]; // . . . Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB ...]; // . . . Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()–>WaitGetPoses();
Advanced Frame Pacing
Multi-GPU workload distribution
Advanced Frame Pacing
Multi-GPU workload distribution
Advanced Frame Pacing

Summary
Advanced Frame Pacing

Summary

Multi-thread application
Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers
Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Separate pose-independent from pose-dependent workloads
Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Separate pose-independent from pose-dependent workloads

Separate workloads by frequency of update
Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Separate pose-independent from pose-dependent workloads

Separate workloads by frequency of update

Benefit from multi-GPU configurations
Advanced Frame Pacing

Summary

Multi-thread application
Split command buffers
Separate pose-independent from pose-dependent workloads
Separate workloads by frequency of update
Benefit from multi-GPU configurations
Rendering thread per GPU
Advanced Frame Pacing

Reducing Fill Rate
Increasing Resolutions
Optimizing for Vive Pro

Rendering at higher resolution
Increasing Resolutions

Optimizing for Vive Pro

Rendering at higher resolution

Vive @ 90Hz
436MP/s
Increasing Resolutions
Optimizing for Vive Pro

Rendering at higher resolution

Vive @ 90Hz
436MP/s

4K

UHD TV @ 60Hz
475MP/s
Increasing Resolutions
Optimizing for Vive Pro

Rendering at higher resolution

Vive @ 90Hz
436MP/s

UHD TV @ 60Hz
475MP/s

Vive Pro @ 90Hz
775MP/s
Reducing Fill Rate

Clipping invisible pixels

Multi-resolution shading

Monoscopic far-field rendering

Stereo shading reprojection
Reducing Fill Rate

Clipping invisible pixels

Multi-resolution shading

Monoscopic far-field rendering

Stereo shading reprojection
Reducing Fill Rate
Clipping invisible pixels

Loss of information in periphery due to lens distortion correction
Reducing Fill Rate
Cliping invisible pixels

Loss of information in periphery due to lens distortion correction
Reducing Fill Rate
Cliping invisible pixels

Use SteamVR stencil mask to clip invisible pixels

VivePro

775MP/s
Reducing Fill Rate
Clipping invisible pixels

Use SteamVR stencil mask to clip invisible pixels

VivePro
775MP/s
620MP/s
Reducing Fill Rate
Multi-resolution shading

Contribution of edge and corner regions
Contribution of edge and corner regions

80° x 80°
Contribution of edge and corner regions

80° x 80°
Reducing Fill Rate
Multi-resolution shading

Contribution of edge and corner regions

80° x 80°
Reducing Fill Rate
Multi-resolution shading

Rendering edge and corner regions at reduced resolution
Reducing Fill Rate
Multi-resolution shading

Rendering edge and corner regions at reduced resolution
Reducing Fill Rate
Multi-resolution shading

Rendering edge and corner regions at reduced resolution
Reducing Fill Rate
Multi-resolution shading

Rendering edge and corner regions at reduced resolution
Reducing Fill Rate
Multi-resolution shading

Rendering edge and corner regions at reduced resolution
Reducing Fill Rate
Multi-resolution shading

Rendering edge and corner regions at reduced resolution
Reducing Fill Rate
Multi-resolution shading

Rendering edge and corner regions at reduced resolution
Reducing Fill Rate

Multi-resolution shading

Upscaling rendered regions to final resolution for submission
Reducing Fill Rate
Multi-resolution shading

Fill rate reduction

VivePro
775MP/s
### Reducing Fill Rate

Multi-resolution shading

| Fill rate reduction | VivePro | 775MP/s | 80° x 80° | 491MP/s |
Reducing Fill Rate
Multi-resolution shading

Fill rate reduction

<table>
<thead>
<tr>
<th></th>
<th>VivePro</th>
<th>(Clipped)</th>
</tr>
</thead>
<tbody>
<tr>
<td>775MP/s</td>
<td>620MP/s</td>
<td></td>
</tr>
<tr>
<td>491MP/s</td>
<td>436MP/s</td>
<td></td>
</tr>
<tr>
<td>80° x 80°</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reducing Fill Rate
Multi-resolution shading

## Fill rate reduction

<table>
<thead>
<tr>
<th></th>
<th>VivePro</th>
<th>(Clipped)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>80° x 80°</strong></td>
<td>775MP/s</td>
<td>620MP/s</td>
</tr>
<tr>
<td><strong>491MP/s</strong></td>
<td>436MP/s</td>
<td></td>
</tr>
<tr>
<td><strong>70° x 70°</strong></td>
<td>433MP/s</td>
<td></td>
</tr>
</tbody>
</table>
Reducing Fill Rate
Multi-resolution shading

Fill rate reduction

<table>
<thead>
<tr>
<th></th>
<th>VivePro</th>
<th>(Clipped)</th>
</tr>
</thead>
<tbody>
<tr>
<td>775MP/s</td>
<td>620MP/s</td>
<td></td>
</tr>
<tr>
<td>80° x 80°</td>
<td>491MP/s</td>
<td>436MP/s</td>
</tr>
<tr>
<td>70° x 70°</td>
<td>433MP/s</td>
<td>382MP/s</td>
</tr>
</tbody>
</table>
## Reducing Fill Rate

### Multi-resolution shading

### Fill rate reduction

<table>
<thead>
<tr>
<th>Description</th>
<th>VivePro</th>
<th>(Clipped)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70° x 70°</td>
<td>433MP/s</td>
<td>382MP/s</td>
</tr>
<tr>
<td>55° x 55°</td>
<td>362MP/s</td>
<td></td>
</tr>
<tr>
<td>80° x 80°</td>
<td>491MP/s</td>
<td>436MP/s</td>
</tr>
<tr>
<td></td>
<td>775MP/s</td>
<td>620MP/s</td>
</tr>
</tbody>
</table>
Reducing Fill Rate
Multi-resolution shading

Fill rate reduction

<table>
<thead>
<tr>
<th></th>
<th>VivePro</th>
<th>(Clipped)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>775MP/s</td>
<td>620MP/s</td>
</tr>
<tr>
<td>80° x 80°</td>
<td>491MP/s</td>
<td>436MP/s</td>
</tr>
<tr>
<td>70° x 70°</td>
<td>433MP/s</td>
<td>382MP/s</td>
</tr>
<tr>
<td>55° x 55°</td>
<td>362MP/s</td>
<td>316MP/s</td>
</tr>
</tbody>
</table>
Summary

Support for HTC VivePro

Advanced VR development enabled with Metal 2

Take advantage of multi-GPU
More Information

https://developer.apple.com/wwdc18/611