Windows + D3D 게임-> macOS/iOS + metal로 포팅하기

유영천

https://megayuchi.com

https://youtube.com/megayuchi

사전 공지

- 아직 모르는 것이 더 많습니다.
- 진행중인 프로젝트에서 기본적인 포팅은 거의 완료된 상태입니다. 이 과정에서 학습한 내용으로 발표합니다.

그래픽 API소개

- DirectX
  - Direct3D 3/5/6/7 - 다분히 OpenGL스러운 고전적인 디자인
  - Direct3D 8 - 최초의 Shader지원. XBOX Original의 기본 API.
  - Direct3D 9 - shader 지원 본격화. 이전까지 D3D의 완성형
  - Direct3D 10 - 다소HW친화적인 디자인. 바뀐 GPU드라이버 모델에 따라 device lost처리의 난해함/어드레스 공간 제약이 사라짐. Windows Vista 망하면서 같이 망함.
  - Direct3D 11 - Tessellation/Compute Shader지원. 아직까지 쌩쌩한 현역.
  - Direct3D 12 - Windows 진영의 차세대 API. 망하는듯 했으나 Raytracing지원으로 화려하게 부활.

그래픽 API소개

- OpenGL
  - 1992년 실리콘 그래픽스에서 만든 그래픽스 표준 API 규격. 한 때는 그래픽 어플리케이션과 게임에서 진정한 표준이었다. 이전에 많은 게임에서 사용했으나 이제 슬슬 물러나는 추세.
- OpenGL ES
  - 임베디드를 위한 Open GL. 실질적으로 널리 쓰이는 버전은 2.0 이상, shader사용을 전제로 한다.Windows/iOS/Linux/Android 모두 사용 가능. 모바일 업계에서 표준에 가까웠다. 안드로이드에서는 여전히 강세.
- Vulkan
  - AMD Mantle에서 파생된 크로스플랫폼 그래픽 API. Linux/Android 계열에서 사용할 수 있는 차세대 API. Windows에서도 사용 가능.

Metal

- 소위 말하는 차세대 API(경량/하드웨어 친화적)중 Apple 제품 전용.
- D3D에 익숙한 프로그래머 입장에서 보자면 D3D9/D3D11/D3D12의 특징을 모두 가지고 있음.
- OpenGL ES보다 빠르다고 함.
- D3D의 모든 버전과 OpenGL ES와 Metal을 경험해본 입장에서 보면 Metal의 학습 난이도는 D3D11과 D3D12의 중간 정도.
- 하지만 개발 환경이 영 좋지 않아서 시간 들어가는건 D3D12보다 적지 않음.
- 애플 제품 타겟으로 그래픽 코드를 집어넣을 경우 강제였…었는데 지금은 아닌 듯.

Windows+D3D -> macOS/iOS + metal 포팅

Windows+D3D -> macOS/iOS + Metal 포팅 목적

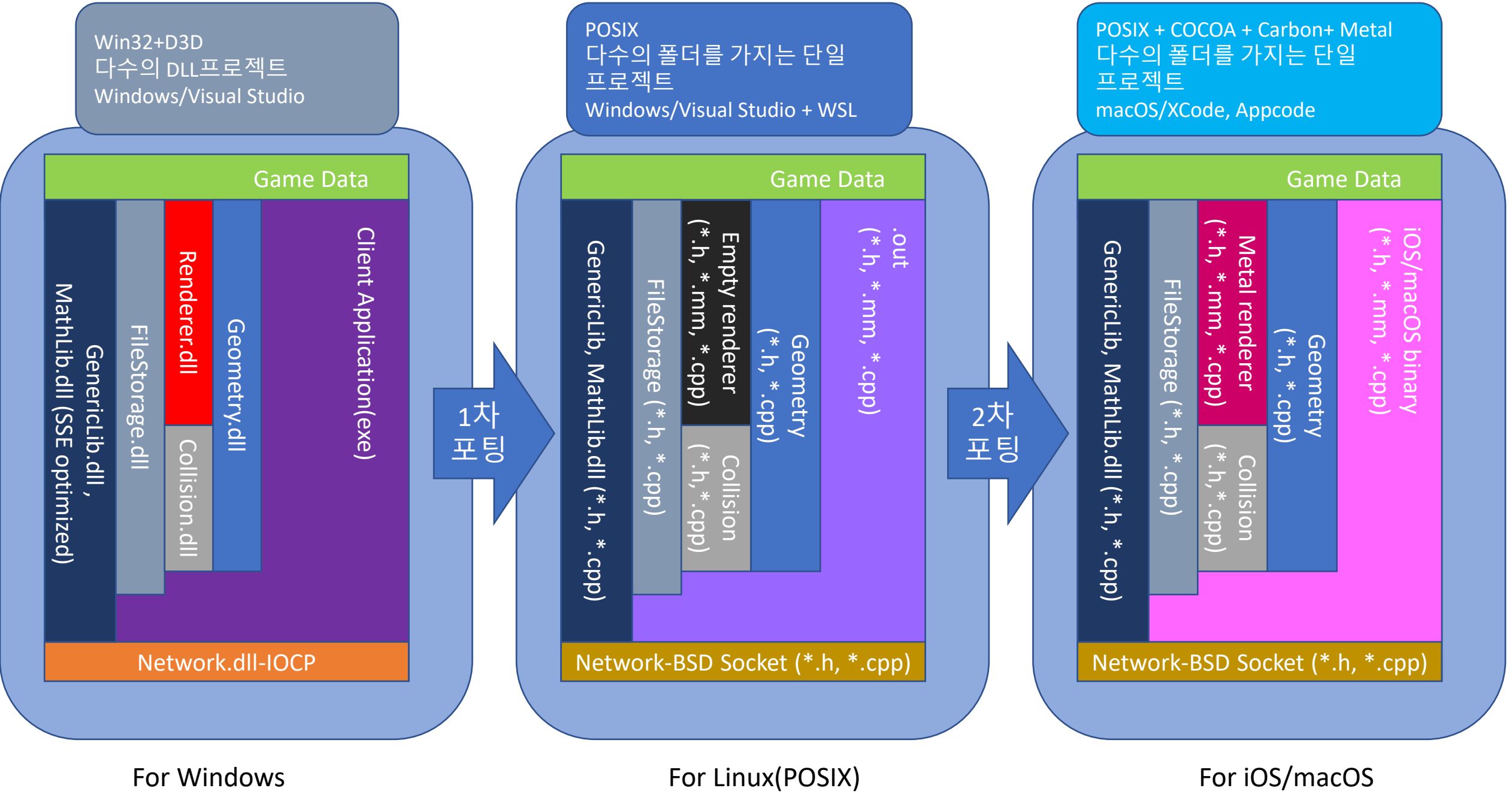
- 프로젝트 종료하기 전에 모바일 포팅 한번 해보자.
  - 혼자 만드는 게임이라 더는 컨텐츠 추가 불가.
  - 재미없는 작업 하기에도 지쳤다.
- iOS? 안드로이드? 고민 끝에 iOS로 결정.
- Metal로 개발하지 않으면 리젝. 기존 게임들도 Metal로 다시 작성하지 않으면 퇴출이라던데?
  - 나중에 알게 된 사실로는 철회했다고 함.
  - 하지만 XCode의 프로젝트 템플릿에서 OpenGL ES프로젝트는 사라지고 없다. 사실상 Metal을 강제하고 있음.

Windows 버전 엔진 구조

- MathLib.dll – 수학함수
- GenericLib.dll – 자료구조,검색,정렬,Memory Pool, Heap등.
- FileStorage.dll – pakcing된 파일과 일반파일을 동시 억세스 하기 위한 파일시스템.
- Renderer.dll – DirectX,OpenGL등 그래픽 API를 직접 호출하고 상위 레이어에 대해 독립적인 렌더링 기능을 제공한다.
- Geoemtry.dll – 오브젝트 관리,맵 관리, 트리거, 충돌처리 등등 직접 화면에 렌더링만 안하는 게임 엔진 본체
- Collision.dll – 충돌처리 및 오브젝트와 속도벡터를 넣었을때 최종 위치와 속도벡터를 산출해주는 이동처리 엔진. GPGPU지원을 위해 따로 분리했다.
- Network.dll – IOCP기반, 서버와 클라이언트 공용 네트워크 엔진.

포팅 계획 및 수행

1. DLL-기반의 다수의 프로젝트를 다수의 폴더를 가지는 1개 프로젝트로 새로 만듦.
2. VC++ 의 Conformance mode 켬
3. 정상 빌드/실행 될 때까지 수정
4. Renderer/Network/sound 제외한 나머지 코드들을 WSL을 이용해서 리눅스 타겟으로 빌드
5. Renderer/Network/Sound는 기능 구현되지 않은 빈 클래스/함수의 코드로 작성.
6. 5)의 코드들을 XCode에서 만든 iOS/macOS게임 프로젝트에 포함
7. 비어있는 Renderer기능을 Metal로 구현


Win32+D3D
다수의 DLL프로젝트
Windows/Visual Studio
Game Data
Client Application(exe)
Geometry.dll
Renderer.dll
Collision.dll
FileStorage.dll
GenericLib.dll ,
MathLib.dll (SSE optimized)
Network.dll-IOCP
1차 포팅
POSIX
다수의 폴더를 가지는 단일 프로젝트
Windows/Visual Studio + WSL
Game Data
.out
(*.h, *.mm, *.cpp)
Geometry
(*.h, *.cpp)
Collision
(*.h,*.cpp)
Empty renderer
(*.h, *.mm, *.cpp)
FileStorage (*.h, *.cpp)
GenericLib, MathLib.dll (*.h, *.cpp)
Network-BSD Socket (*.h, *.cpp)
2차 포팅
POSIX + COCOA + Carbon+ Metal
다수의 폴더를 가지는 단일 프로젝트
macOS/XCode, Appcode
Game Data
iOS/macOS binary
(*.h, *.mm, *.cpp)
Geometry
(*.h, *.cpp)
Collision
(*.h,*.cpp)
Metal renderer
(*.h, *.mm, *.cpp)
FileStorage (*.h, *.cpp)
GenericLib, MathLib.dll (*.h, *.cpp)
Network-BSD Socket (*.h, *.cpp)
For Windows
For Linux(POSIX)
For iOS/macOS

D3D9/D3D11과 Metal의 유사점

- Texture, Buffer 각각 객체가 따로 있음(D3D9)
- CPU/GPU메모리간 전송이 추상화 되어있음.(D3D9/11)
- Texture, Buffer등 리소스들을 각각의 객체가 아닌 배열 형태로 전달(D3D11)

D3D12와 Metal의 유사점

- 렌더링 관련 기능 호출시 즉시(immediate) 렌더링 하지 않고 렌더링 커맨드를 recording 후 submit.
- 수동 리소스 상태 추적(D3D12는 완전히 수동. Metal은 필요에 따라)
- 리소스 관리 완전 수동. 프로그래머가 렌더링이 완료될 때까지 리소스의 유효성을 보장해야 함.
- Resource renaming 없음. 렌더링이 완료되기 전에 리소스에 덮어쓰기 하면 안됨.
- 각종 렌더링 상태 설정과 shader설정을 Pipeline State 객체로 묶어서 처리. -> Pipeline State폭발-_-

D3D와 Metal의 차이점

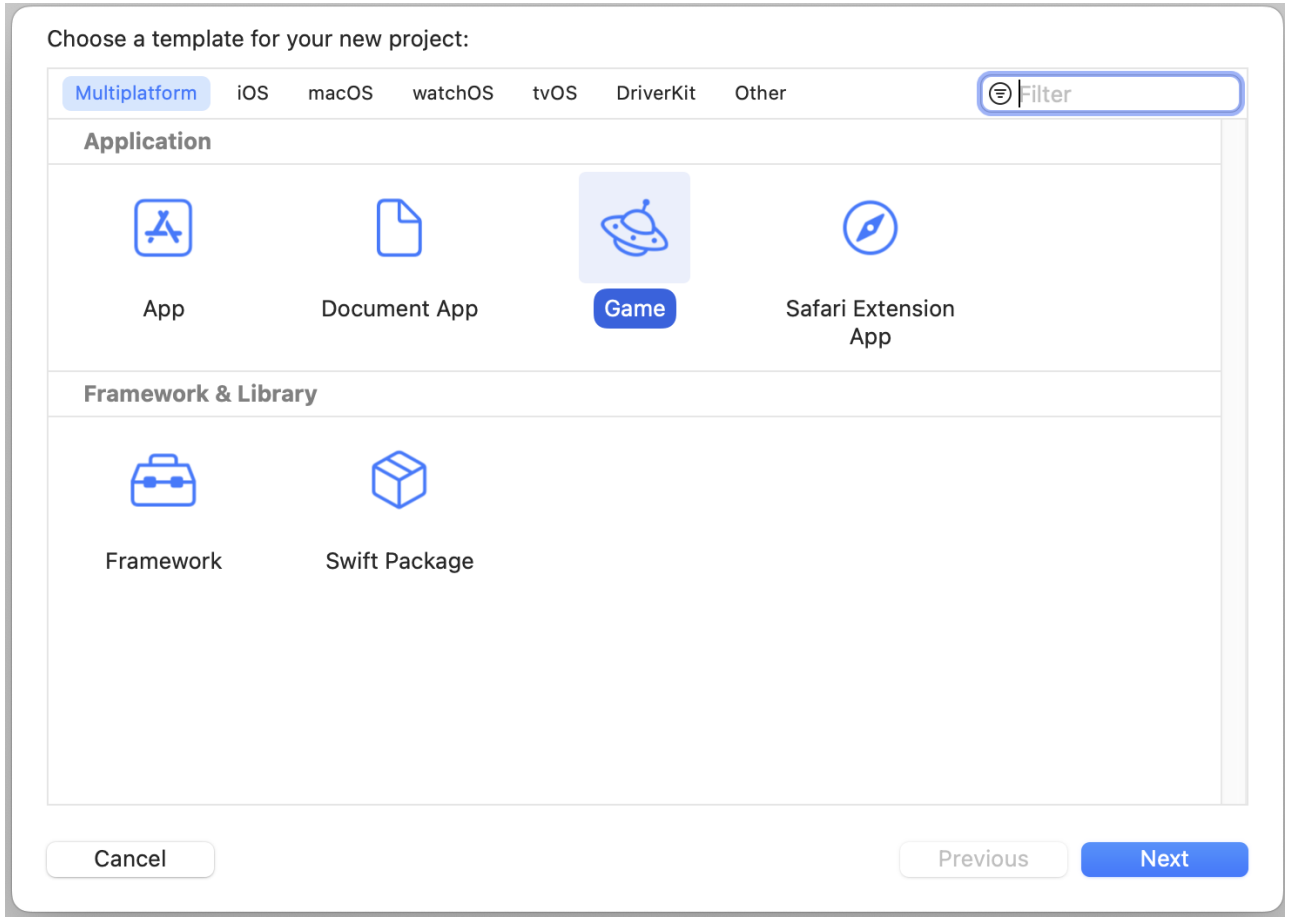
- Render Target과 Command Encoder
  - D3D: RenderTaraget과 Command를 기록하는 객체는 완전히 별개.
  - Metal: Render Target과 Command를 기록하는 객체는 강하게 묶여있음. RenderCommandEncoder를 얻을때 인자로 Render Target(RenderPassDescriptor)를 전달해야함.이 때문에 클래스 설계할 때 상당히 차이가 발생함.
- CommandEncoder/CommandList의 범용성
  - D3D: CommandList하나로 computing/redering 모두 처리.
  - Metal: RenderComputeEncoder와 RenderCommandEncoder가 분리되어 있음.

Metal API프로그래밍

Metal C++

- 기본적으로 Metal은 Objective-C API를 노출함.
- C++에 익숙한 대부분의 게임 프로그래머들에게 말도 안되는 환경.
- Apple에서 공식적으로 C++ wrapper클래스를 제공.
- 소개 및 설치지침 공식 문서
  - https://developer.apple.com/kr/metal/cpp/
- 프로그래밍 가이드(반드시 시청해야함)
  - https://developer.apple.com/videos/play/wwdc2022/10160/

프로젝트 생성


Choose a template for your new project:
Multiplatform
iOS
macOS
watchOS
tvOS
DriverKit
Other
Filter
Application
App
Document App
Game
Safari Extension App
Framework & Library
Framework
Swift Package
Cancel
Previous
Next

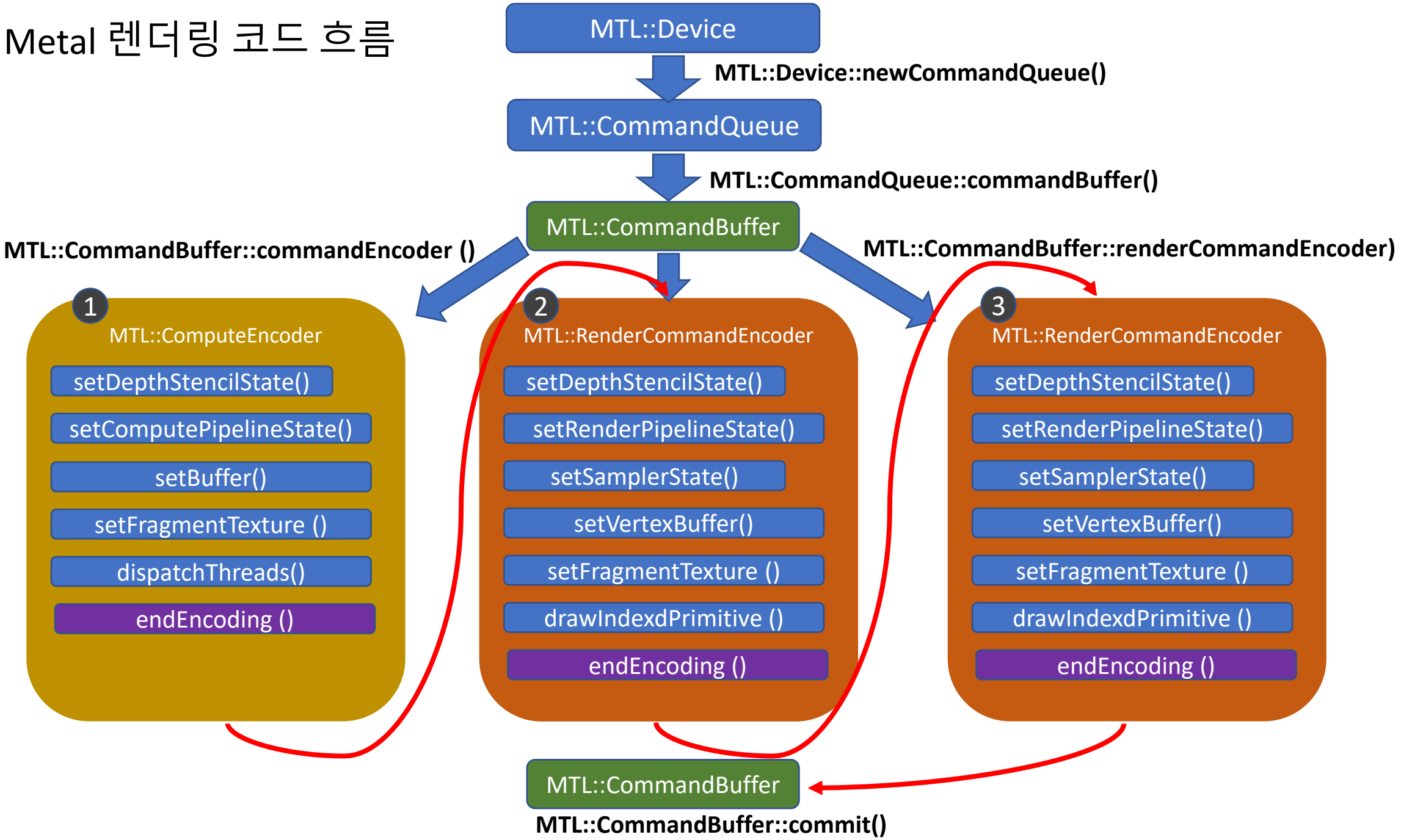
코딩 시작 위치 - 초기화

```
34
35  -(nonnull instancetype)initWithMetalKitView:(nonnull MTKView *)view;
36  {
37      self = [super init];
38      if(self)
39      {
40          CGSize size = view.drawableSize;
41          float width = size.width;
42          float height = size.height;
43          _device = view.device;
44
45          [self _loadMetalWithView:view];
46
47          NSURL* url = [[NSBundle mainBundle] executableURL];
48          NSString* dummyDataPath = [[NSBundle mainBundle] pathForResource:@"Dummy" ofType:@"txt"];
49          const char* szDummyDataPath = [dummyDataPath cStringUsingEncoding:1];
50          printf("%s\n",szDummyDataPath);
51
52          NS::AutoreleasePool* pPool = NS::AutoreleasePool::alloc()->init();
53
54          MTL::Device* pDevice = (__bridge MTL::Device*)_device;
55
56          MTL::RenderPassDescriptor* pRenderPassDescriptor = (__bridge MTL::RenderPassDescriptor*)view.currentRenderPassDescriptor;
57
58          m_pMtlWrapper = new CMtlWrapper;
59          m_pMtlWrapper->Init(pDevice, pRenderPassDescriptor, width, height, fDPI, szDummyDataPath);
60
61
62          pPool->release();
63      }
64
65      return self;
66  }
67
```

코딩 시작 위치 - per Frame

```
340
341 - (void)drawInMTKView:(nonnull MTKView *)view
342 {
343     /// Per frame updates here
344     view.clearColor = MTLClearColorMake(0.0,0.0,1.0,1.0);
345     view.clearDepth = 1.0;
346
347     MTL::RenderPassDescriptor* pRenderPassDescriptor = (__bridge_retained MTL::RenderPassDescriptor*)view.currentRenderPassDescriptor;
348
349     MTL::Drawable* pDrawable = (__bridge MTL::Drawable*)view.currentDrawable;
350     m_pMtlWrapper->OnDraw(pRenderPassDescriptor, pDrawable);
351
352     pRenderPassDescriptor->release();
353 }
```

Metal 렌더링 코드 흐름

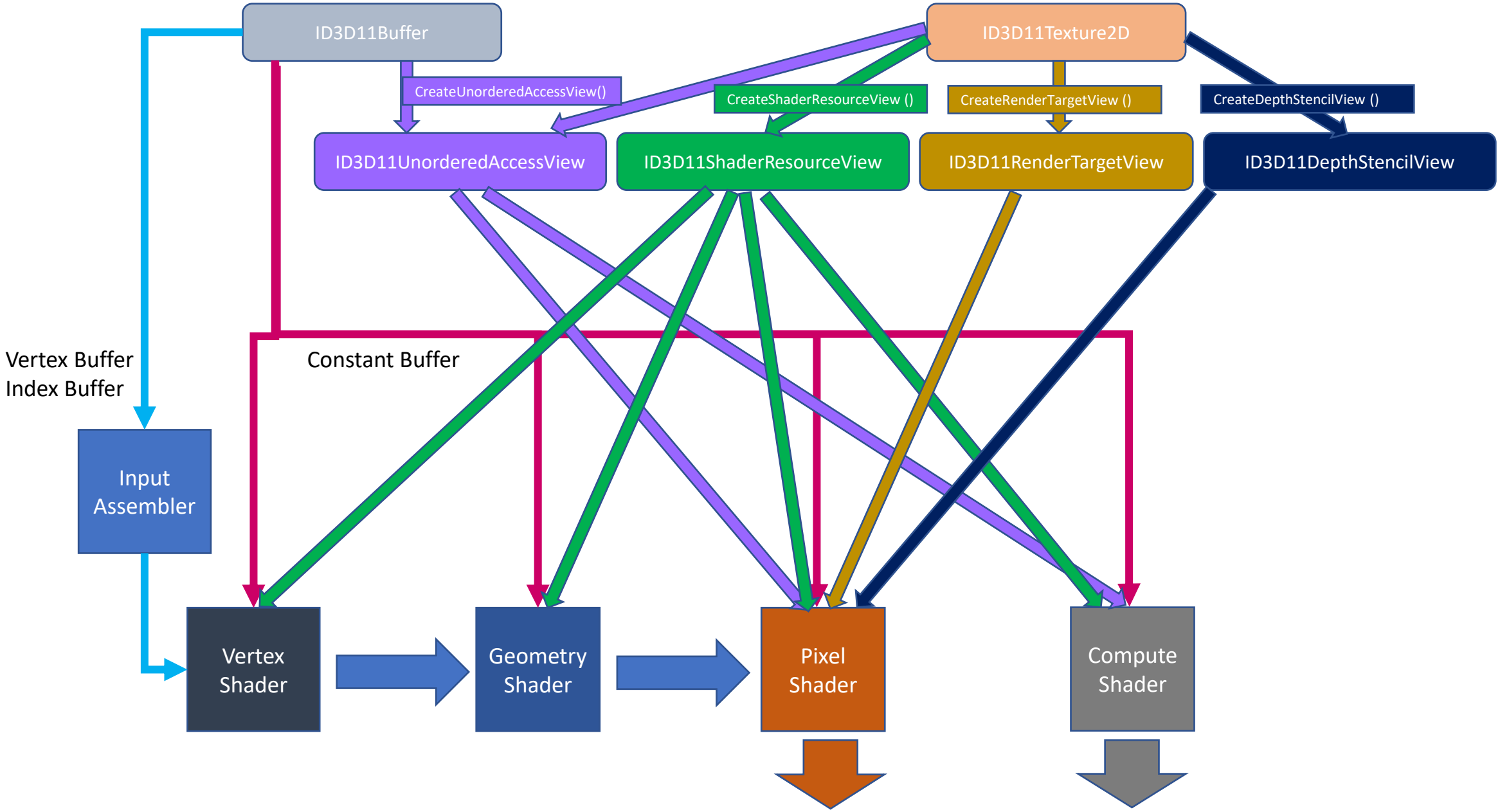


주요객체 대응 관계

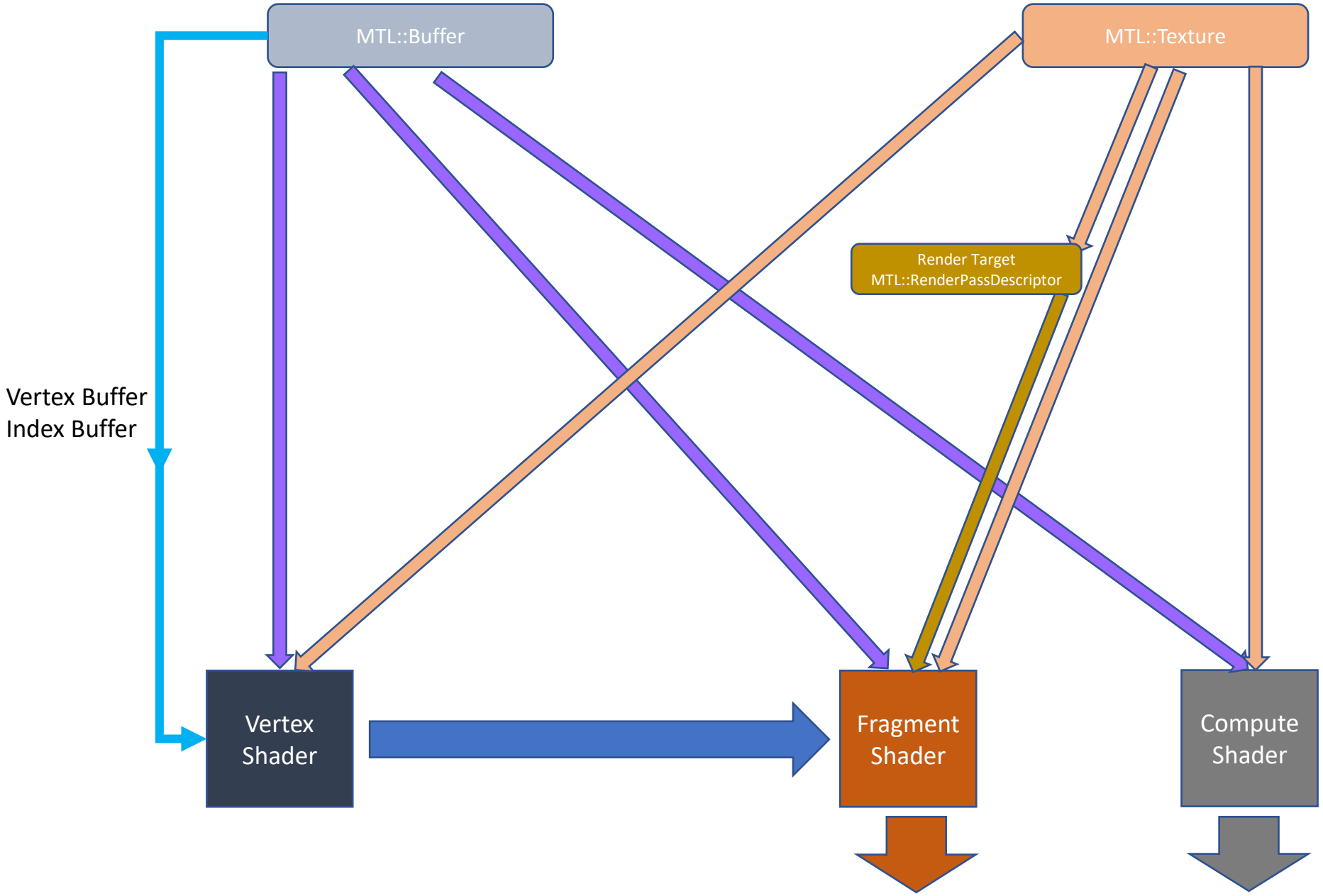
Metal	D3D11	D3D12
MTL::Device	ID3D11Device	ID3D12Device
MTL::RenderCommandEncoder	ID3D11DeviceContext	ID3D12GraphicsCommndList
MTL::ComputeCommandEncoder	ID3D11DeviceContext	ID3D12GraphicsCommndList
MTL::CommandBuffer	ID3D11DeviceContext	ID3D12GraphicsCommndList
MTL::CommandQueue	ID3D11DeviceContext	ID3D12CommandQueue
MTL::Texture	ID3D11Texture2D	D3D12_CPU_DESCRIPTOR_HANDLE
MTL::Buffer	ID3D11Buffer	D3D12_VERTEX_BUFFER_VIEW
MTL::RenderPipelineState	ID3D11VertexlShader/ID3D11PixelShader	ID3D12PipelineState
MTL::ComputePipelineState	ID3D11ComputeShader	ID3D12PipelineState
MTL::SamplerState	ID3D11SamplerState	D3D12_STATIC_SAMPLER_DESC + ID3D12RootSignature
MTL::DepthStencilState	ID3D11DepthStencilState	ID3D12PipelineState

Resource Binding

D3D11 Resource흐름



Metal Resource흐름



D3D9

- IDirect3DDevice9::CreateTexture() ->IDirect3DTexture
  - As Texture
    - IDirect3DDevice9::SetTexture();
  - As Render Target
    - IDirect3DTexture -> GetSurfaceLevel() -> IDirect3DSurface9
    - IDirect3DDevice9 ::SetRenderTarget(IDirect3DSurface9)
- IDirect3DDevice9::CreateVertexBuffer() -> IDirect3DVertexBuffer9
  - IDirect3DDevice9 ::SetStreamSource (IDirect3DVertexBuffer9)

D3D11

- ID3D11Device::CreateTexture() -> ID3D11Texture3D
  - As texture
    - ID3D11Texture3D -> ID3D11Device ::CreateShaderResourceView() -> ID3D11ShaderResourceView
    - ID3D11DeviceContext::PSSetShaderResource(ID3D11ShaderResourceView)
  - As Render Target
    - ID3D11Texture3D -> ID3D11Device::CreateRenderTargetView() -> ID3D11RenderTargetView
    - ID3D11DeivceContext::OMSetRenderTarget(ID3D11RenderTargetView)
- ID3D11Device::CreateBuffer() -> ID3D11Buffer
  - As Vertex Buffer
    - ID3D11DeivceContext::IASetVertexBuffers(ID3D11Buffer)
  - As UAV
    - ID3D11Deivce::CreateUnorderedAccessView() -> ID3D11UnorderedAccessView
    - ID3D11DeivceContext::CSSetUnorderedAccessViews(ID3D11UnorderedAccessView)

D3D12

- ID3D12Device:: CreateCommittedResource() -> ID3D12Resource
  - As Texture
    - ID3D12Device ::CreateShaderResourceView() -> D3D12_CPU_DESCRIPTOR_HANDLE
    - ID3D12GraphcisCommandList::SetGraphicsRootDescriptorTable(D3D12_CPU_DESCRIPTOR_HANDLE)
  - As Vertex Buffer
    - ID3D12Resource::GetGPUVirtualAddress()
    - ID3D12GraphcisCommandList::IASetVertexBuffer()
  - As Unordered Access Memory
    - ID3D12Device ::CreateUnorderedAccessView() -> D3D12_CPU_DESCRIPTOR_HANDLE
    - ID3D12GraphcisCommandList::SetComputeRootDescriptorTable(D3D12_CPU_DESCRIPTOR_HANDLE)

Metal

- MTL::Device::newTexture() -> MTL::Texture
  - As texture
    - MTL::RenderCommandEncoder::setFragmentTexture(MTL::Texture)
  - As Render Target
    - MTL::RenderPassDescriptor::colorAttachments()->object()->setTexture(MTL::Texture);
    - MTL::CommandBuffer::renderCommandEncoder()
- MTL::Device::newBuffer () -> MTL::Buffer
  - As Vertex Buffer
    - MTL::RenderCommandEncoder::setVertexBuffer(MTL::Buffer)
  - As Unordered Access Memory
    - MTL::ComputeCommandEncoder::setBuffer(MTL::Buffer)

Metal Shader

Metal Shader 특징

- 포인터 사용 가능.
- D3D와는 달리 중간에 어셈블러를 거치지 않고 날(raw)포인터 그대로 전달됨.
- 따라서 Align 규칙에 신경 써야 함.
- 내장함수는 HLSL/GLSL과 거의 1:1에 가깝게 대응됨.
- 런타임 컴파일시 다른 .metal파일의 include문 처리에 문제가 있는 것으로 보임.
- Geometry Shader는 없음.

Metal Shader 특징

- HLSL의 mul()함수는 없고 *연산자만 지원.
- 기본적으로 행렬 x 벡터로 HLSL과 순서가 반대이다.
- matrx4x3, matrx3x3등 캐스팅이 안된다. 포인터로 캐스팅할 경우 행렬도 Raw포인터로 전달되는 값이기 때문에 다른 결과가 나온다.
- Constant Buffer, Vertex Buffer, Texture, Sampler 모두 파라미터로 전달받는다.
- Buffer종류는 모두 raw 포인터로 전달받는다.
- Address space라는 것이 존재한다. thread, constant, device구분 필요.
- 포인터와 참조 모두 사용 가능.
- 전달된 텍스처와 버퍼에 바로 read/write할 수 있다.

HLSL -> Metal Shader 포팅 tip

- HLSL의 코드를 거의 그대로 사용하고 싶다면 mul()함수를 만들어 둔다. (바깥에서 전치된 행렬로 전달)
- HLSL의 out 으로 지정된 파라미터는 C++의 참조로 넘기면 포팅이 더 수월하다.
- 런타임에 컴파일하면 너무 과정이 길기 때문에 CUI컴파일러를 사용해서 한 줄씩 포팅하면서 컴파일이 잘 되는지 확인한다.

Metal Shader에서 HLSL의 mul()함수의 구현

```
float4 mul(float4 v, constant float4x4* m)
{
    // multiply vector4 * traonsposed 4x4 matrix
    float4 r;
    constant float4* p_m = (constant float4*)m;

    r.x = dot(v, *(p_m + 0));
    r.y = dot(v, *(p_m + 1));
    r.z = dot(v, *(p_m + 2));
    r.w = dot(v, *(p_m + 3));
    return r;
}
float3 mul(float3 v, constant float4x4* m)
{
    // multiply vector3 * traonsposed 4x4 matrix as 3x3 matrix
    float3 r;
    constant float4* p_v4 = (constant float4*)m;

    r.x = dot(v, *(constant float3*)(p_v4 + 0));
    r.y = dot(v, *(constant float3*)(p_v4 + 1));
    r.z = dot(v, *(constant float3*)(p_v4 + 2));

    return r;
}
float4 mul(float4 v, constant float4x3* m)
{
    // multiply vector4 * traonsposed 4x3 matrix
    float4 r;
    constant float4* p_v4 = (constant float4*)m;

    r.x = dot(v, *(p_v4 + 0));
    r.y = dot(v, *(p_v4 + 1));
    r.z = dot(v, *(p_v4 + 2));
    r.w = 1.0;

    return r;
}
```

```metal
#include <metal_stdlib>
#include <simd/simd.h>


using namespace metal;

#include "sh_dynamic_common.metal"
#include "sh_util.metal"


struct PS_INPUT_IMM
{
    float4  Pos [[position]];   // : SV_POSITION;
    float2  TexCoord;
    float4  PosWorld;
    float   Dist;
    float4  Diffuse;
    float4  NormalColor;
};

struct PS_INPUT_DEPTH
{
    float4 Pos [[position]];    // : SV_POSITION;
    float Depth [[depth(any)]];
};

vertex PS_INPUT_IMM vsDefault(device const VS_INPUT_VL* pVertex [[buffer(0)]], device const float2* pTexCoord [[buffer(1)]],
                              constant CONSTANT_BUFFER_DEFAULT* pConstBufferDefault [[buffer(2)]], constant CONSTANT_BUFFER_MATERIAL* pConstBufferMaterial [[buffer(4)]],
                              uint vertexId [[vertex_id]], uint instanceId [[instance_id]])
{
    PS_INPUT_IMM    output = {};

    output.Pos = mul(float4(pVertex[vertexId].Pos, 1.0), &pConstBufferDefault->matWorldViewProjArray[0]);
    float3  NormalWorld = mul(pVertex[vertexId].Normal, &pConstBufferDefault->matWorld);    // 월드공간에서 노말

    NormalWorld = normalize(NormalWorld);                           // 다시 노멀라이즈(스케일이 들어있을 경우를 대비해서)
    output.NormalColor.rgb = (NormalWorld * 0.5f) + 0.5f;
    output.NormalColor.a = 1;
    output.TexCoord = float2(pTexCoord[vertexId].x, pTexCoord[vertexId].y);
    output.Diffuse = pConstBufferMaterial->MtlDiffuse;

    return output;
}

fragment PS_TARGET psDefault(PS_INPUT_IMM inputFrag [[stage_in]],
                             constant CONSTANT_BUFFER_DEFAULT* pConstBufferDefault [[buffer(2)]], constant CONSTANT_BUFFER_MATERIAL* pConstBufferMaterial [[buffer(4)]],
                             texture2d< float, access::sample > texDiffuse [[texture(0)]],
                             sampler samplerWrap [[sampler(0)]], sampler samplerClamp [[sampler(1)]], sampler samplerBorder [[sampler(2)]])
{
    PS_TARGET outColor = {};

    float4 texColor = texDiffuse.sample(samplerWrap, inputFrag.TexCoord);

    float4  NormalColor = float4(inputFrag.NormalColor.xyz, pConstBufferDefault->Property / 255.0f);
    outColor.Color0 = half4(texColor * inputFrag.Diffuse);
    outColor.Color1 = half4(NormalColor);
    outColor.Color2 = half4(0, 0, 0, 0);

    return outColor;
}
```

```metal
struct VS_INPUT_VL
{
    float3 Pos;         // 0    : POSITION;
    float3 Normal;      // 16   : NORMAL;
    float4 Tangent;     // 32   : TANGENT(x,y,z) + UINT Property;
};  // 48 bytes
```

Shader본체가 아닌 유틸리티 함수인 경우는 거의 1:1로 포팅할 수 있다.

Select Files or Folders | sh_att_light.hlsl - sh_att_light.metal

Location

C:\DEV\DAIKON_ROOT\VOXEL_HORIZON\MegayuchiRenderer11\Shaders\sh_att_light.hlsl

```
float3 CalcAttLightColor(float3 Pos, int AttLightNum)
{
    float3      LightColorSum = 0;

    for (int i = 0; i < AttLightNum; i++)
    {
        float3      LightVec = (AttLight[i].Pos.xyz - Pos.xyz);
        float       LightVecDot = dot(LightVec, LightVec);
        float       Dist = sqrt(LightVecDot);
        float       RsSubDist = AttLight[i].Rs - Dist;

        if (RsSubDist < 0.0f)
        {
            continue;
        }
        float4  ColorCenter = ConvertDWORDTofloat(AttLight[i].ColorCenter);
        float4  ColorSide = ConvertDWORDTofloat(AttLight[i].ColorSide);

        float   NrmDist = (Dist / AttLight[i].Rs);
        float   FallOff = (RsSubDist*RsSubDist) * AttLight[i].RcpRsRs;
        float3  LightColor = lerp(ColorCenter.rgb, ColorSide.rgb, NrmDist) * FallOff;

        LightColorSum += LightColor;
    }
    return LightColorSum;
}
float3 CalcAttLightColorWithNdotL(float3 Pos, float NdotL[8], int AttLightNum)
{

    float3 LightColorSum = 0;

    for (int i = 0; i < AttLightNum; i++)
    {
        if (NdotL[i] <= 0.0f)
        {
            continue;
        }

        float3 LightVec = (AttLight[i].Pos.xyz - Pos.xyz);
        float LightVecDot = dot(LightVec, LightVec);
        float Dist = sqrt(LightVecDot);
        float RsSubDist = AttLight[i].Rs - Dist;

        if (RsSubDist < 0.0f)
        {
            continue;
        }
```

Ln: 22 Col: 22/22 Ch: 13/13 iso-ir-149 Win

C:\DEV\git\Linux\Shaders_Metal\sh_att_light.metal

```
using namespace metal;

float3 CalcAttLightColor(float3 Pos, constant ATT_LIGHT* AttLight, int AttLightNum)
{
    float3      LightColorSum = 0;

    for (int i = 0; i < AttLightNum; i++)
    {
        float3      LightVec = (AttLight[i].Pos.xyz - Pos.xyz);
        float       LightVecDot = dot(LightVec, LightVec);
        float       Dist = sqrt(LightVecDot);
        float       RsSubDist = AttLight[i].Rs - Dist;

        if (RsSubDist < 0.0f)
        {
            continue;
        }
        float4  ColorCenter = ConvertDWORDTofloat(AttLight[i].ColorCenter);
        float4  ColorSide = ConvertDWORDTofloat(AttLight[i].ColorSide);

        float   NrmDist = (Dist / AttLight[i].Rs);
        float   FallOff = (RsSubDist*RsSubDist) * AttLight[i].RcpRsRs;
        float3  LightColor = mix(ColorCenter.rgb, ColorSide.rgb, NrmDist) * FallOff;

        LightColorSum += LightColor;
    }
    return LightColorSum;
}
float3 CalcAttLightColorWithNdotL(float3 Pos, float NdotL[8], constant ATT_LIGHT *AttLight, int
{

    float3 LightColorSum = 0;

    for (int i = 0; i < AttLightNum; i++)
    {
        if (NdotL[i] <= 0.0f)
        {
            continue;
        }

        float3 LightVec = (AttLight[i].Pos.xyz - Pos.xyz);
        float LightVecDot = dot(LightVec, LightVec);
        float Dist = sqrt(LightVecDot);
        float RsSubDist = AttLight[i].Rs - Dist;

        if (RsSubDist < 0.0f)
        {
            continue;
        }
```

Ln: 70 Col: 38/71 Ch: 32/65 iso-ir-149 Win

HLSL

```hlsl
RWByteAddressBuffer StartOffsetBuffer : register(u3);   // W
RWStructuredBuffer <FRAGMENT_LINK> FLBuffer : register(u4); // RW
RWByteAddressBuffer PropertyBuffer : register(u5);  // RW
RWByteAddressBuffer FailCountBuffer : register(u6); // RW

[numthreads(1024, 1, 1)]
void csClearABuffer(uint3 groupID : SV_GroupID, uint3 dispatchThreadId : SV_DispatchThreadID)
{
    uint2   CurPixel = uint2(dispatchThreadId.x % ScreenWidth, dispatchThreadId.x / ScreenWid
    uint    ArrayIndex = groupID.y;

    // Skip out of bound pixels
    if (CurPixel.y < ScreenHeight)
    {
        uint address = ((CurPixel.x + CurPixel.y*ScreenWidth) + (ScreenWidth*ScreenHeight*Arr
        StartOffsetBuffer.Store(address, ClearValue);

        PropertyBuffer.Store(address, 0);
    }
}
void SortFragList(inout FRAGMENT_EX FragList[MAX_SORT_NUM], uint FragCount)
{
    for (uint i = 0; i < FragCount; i++)
    {
        for (uint j = 0; j < FragCount; j++)
        {
            if (FragList[i].fDepth > FragList[j].fDepth)
            {
                FRAGMENT_EX t = FragList[i];
                FragList[i] = FragList[j];
                FragList[j] = t;
            }
        }
    }
}
/*
float4 ABufferResolvePS(FullScreenTriangleVSOut input) : SV_Target
{
    float3 blendColor = 0;
    float  totalTransmittance = 1;
    int2   screenAddress = int2(input.positionViewport.xy);
    uint   firstNodeOffset = FL_GetFirstNodeOffset(screenAddress);

    // Get offset to the first node
    uint outerNodeOffset = firstNodeOffset;

    // Fetch and sort nodes
```

MSL

```msl
    {
        float value = (float)((uNext & 0x80000000) != 0);
        return value;
    }
};

kernel void csClearABuffer(uint2 dispatchThreadId [[thread_position_in_grid]], uint2 gridSize
                           constant CONSTANT_BUFFER_A_BUFFER* pConstABuffer [[buffer(6)]],
                           device uint* pFLCounter [[buffer(7)]],
                           device uint* pStartOffetBuffer [[buffer(8)]],
                           device uint* pPropertyBuffer [[buffer(10)]],
                           device uint* pFailCountBuffer [[buffer(11)]])
{
    uint index = dispatchThreadId.x + (dispatchThreadId.y * pConstABuffer->ScreenWidth);

    uint BufferSize = pConstABuffer->ScreenWidth * pConstABuffer->ScreenHeight;

    if (index < BufferSize)
    {
        pStartOffetBuffer[index] = INVALID_A_BUFFRE_OFFSET_VALUE;
        pPropertyBuffer[index] = 0;
    }
    if (index < 16)
    {
        pFailCountBuffer[index] = 0;
    }
    if (index == 0)
    {
        *pFLCounter = 0;
    }
}
void SortFragList(thread FRAGMENT_EX* FragList, uint FragCount)
{
    // pFragList 사이즈는 MAX_SORT_NUM
    for (uint i = 0; i < FragCount; i++)
    {
        for (uint j = 0; j < FragCount; j++)
        {
            if (FragList[i].fDepth > FragList[j].fDepth)
            {
                FRAGMENT_EX t = FragList[i];
                FragList[i] = FragList[j];
                FragList[j] = t;
            }
        }
    }
}
```

A-Buffer를 clear하는 Compute shader

HLSL 의 Semantics -> MSL의 Attribute

- SV_Position -> [[position]]
- SV_Target0 -> [[color(0)]], SV_Target1 -> [[color(1)]], SV_Target2 -> [[color(2)]]
- SV_Depth -> [[depth(any)]]
- SV_ClipDistance -> [[clip_distance]]
- SV_RenderTargetArrayIndex -> [[render_target_array_index]]

Metal’s Command-Line Tools

- https://developer.apple.com/documentation/metal/shader_libraries/compiling_shader_code_into_a_library_with_metal_s_command-line_tools
- 커맨드 라인에서 미리 shader코드를 빌드. 게임을 실행하지 않고 shader코드가 정상적으로 빌드 되는지 확인 가능.
- 게임 로딩시간을 단축할 수 있음.
- 다른 .metal파일을 #include했을 때 정상적으로 작동함.
- #define 매크로 전달 가능.
- Windows 환경에서도 사용 가능
  - https://developer.apple.com/download/all/?q=metal

CUI(Windows)환경에서의 metal shader 컴파일

ex) sh_bboard.metal을 컴파일 후 metallib로 변환 할 경우
metal.exe -c -std=ios-metal2.1 -mmacosx-version-min=12.0 -o sh_bboard-macOS.air sh_bboard.metal
metallib.exe sh_bboard-macOS.air -o sh_bboard-macOS.air.metallib

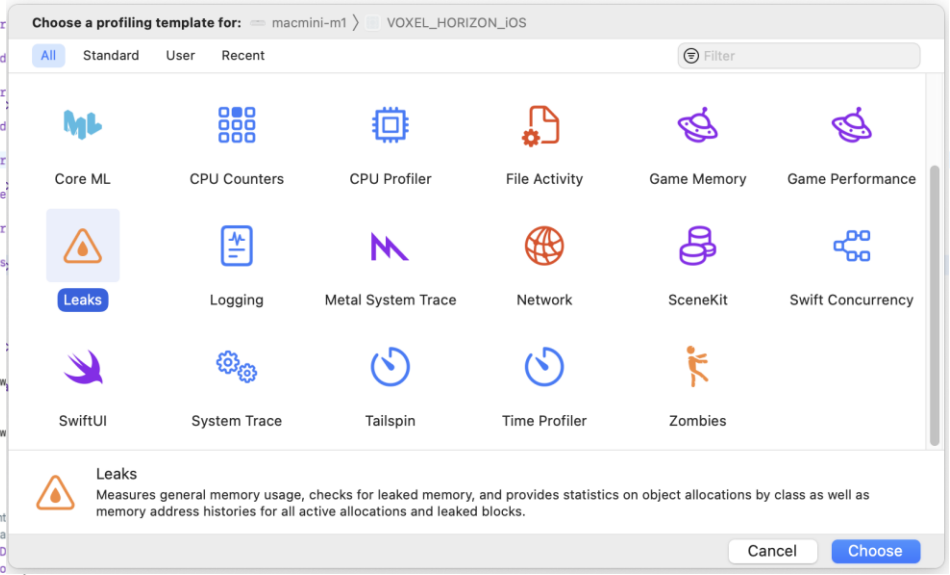
```
C:¥DEV¥git¥Linux¥Shaders_Metal>build_sh_bboard

C:¥DEV¥git¥Linux¥Shaders_Metal>call compile_macos sh_bboard sh_bboard_oit0 "SHADER_PARAMETER_USE_OIT=0"
In file included from sh_bboard.metal:25:
./sh_a_buffer.metal:165:12: warning: unused variable 'index' [-Wunused-variable]
    int    index;
           ^

sh_bboard.metal:104:24: error: use of undeclared identifier 'NormalColor'; did you mean 'NormalColor1'?
        output.Color1 = half4(NormalColor);
                              ^~~~~~~~~~~
                              NormalColor1
sh_bboard.metal:101:9: note: 'NormalColor1' declared here
        float4 NormalColor1 = float4(inputFrag.NormalColor.xyz, (float)pConstBufferDefault->Property / 255.0f);
               ^

1 warning and 1 error generated.
In file included from sh_bboard.metal:25:
./sh_a_buffer.metal:165:12: warning: unused variable 'index' [-Wunused-variable]
    int    index;
           ^

sh_bboard.metal:104:24: error: use of undeclared identifier 'NormalColor'; did you mean 'NormalColor1'?
        output.Color1 = half4(NormalColor);
                              ^~~~~~~~~~~
                              NormalColor1
sh_bboard.metal:101:9: note: 'NormalColor1' declared here
        float4 NormalColor1 = float4(inputFrag.NormalColor.xyz, (float)pConstBufferDefault->Property / 255.0f);
               ^

sh_bboard.metal:141:84: error: use of undeclared identifier 'NormalColor'; did you mean 'NormalColor1'?
        pFLBuffer[uPixelCount].uNormal_ElementID = Pack_Normal_Property_ElementID_To_UINT(NormalColor.rgb, pCons...
                                                                                           ^~~~~~~~~~~
                                                                                           NormalColor1
```

디버깅

- Metal API validation
- Address sanitizer
- Profiling

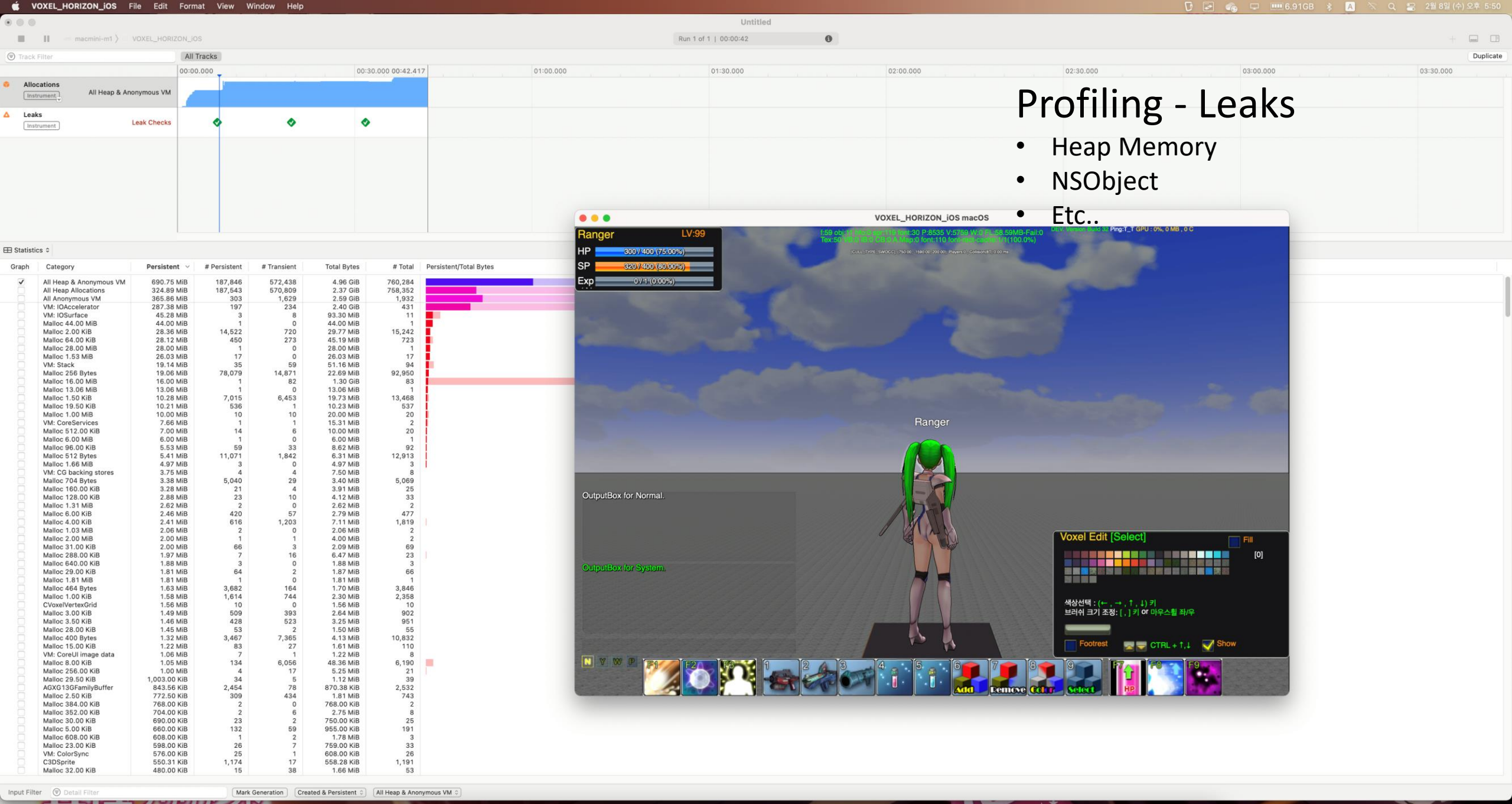
Profiling - Leaks

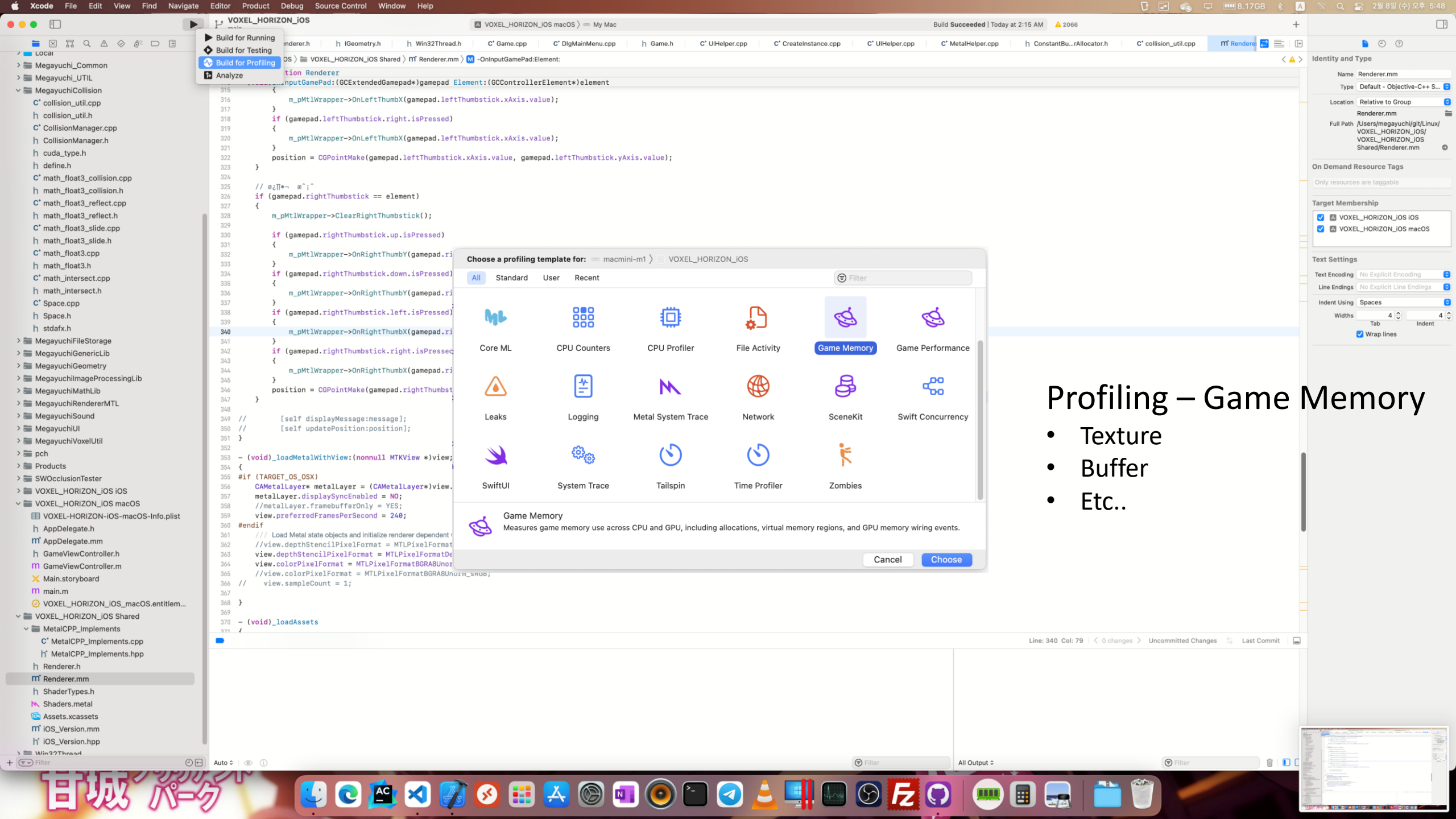
- Heap Memory
- NSObject
- Etc..



Profiling - Leaks

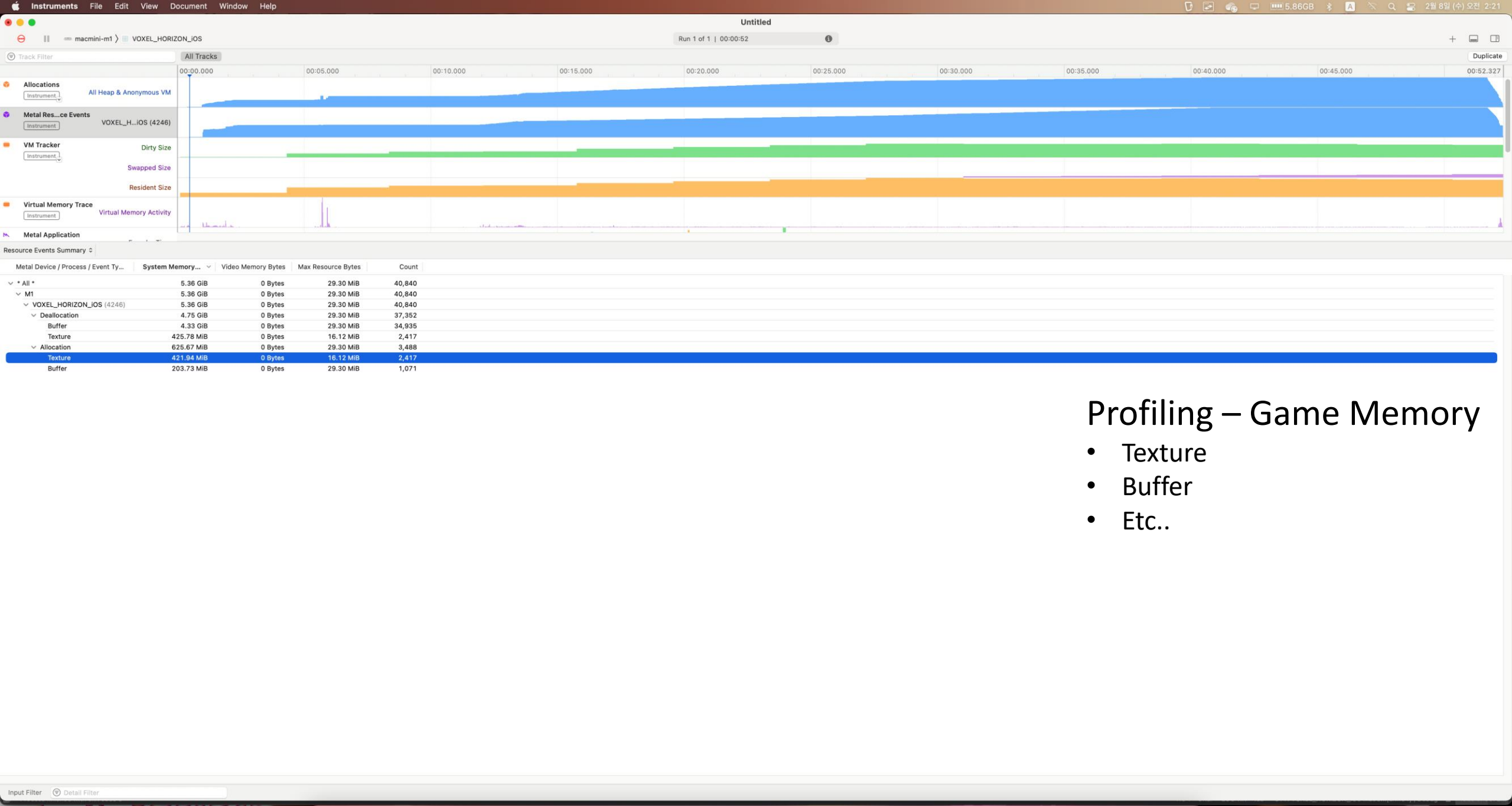
- Heap Memory
- NSObject
- Etc..





Profiling – Game Memory

- Texture
- Buffer
- Etc..



Profiling – Game Memory

- Texture
- Buffer
- Etc..

Windows API -> macOS/POSIX mapping

- Windows버전이 메인이고 Windows빌드의 코드를 최대한 macOS/iOS빌드에서 그대로 사용할 수 있도록 한다.
- Win32 API, VC++ CRT함수들은 지원되지 않으므로 똑같거나 최대한 비슷한 모양의 함수들을 구현한 레이어를 작성.
- Compiler intrinsic에 대한 에뮬레이션 레이어를 작성.
- macOS/iOS는 POSIX환경을 지원하므로 POSIX API로 맵핑.
- Win32의 GDI에 해당하는 기능들은 macOS의 Carbon API, Cocoa Foundation, iOS의 UIView에 있는 기능들로 맵핑(Objective-C코딩 필요).

Tip

- DDS파일을 읽을 수는 없을까?
  - .dds파일을 읽어서 사용할 수 있다면 D3D용 리소스 파일을 그대로 사용 가능.
  - Linux로 빌드 가능한 DirectXTex라이브러를 사용하면 가능.
  - 압축 텍스처 그대로 사용할 수는 없지만 텍스처의 압축을 풀어서 사용할 수는 있음.
  - dds파일을 로드해서 압축을 풀어내기 위한 최소 파일들만 조립해서 static libaray(.a)로 만들어서 사용.
  - https://walbourn.github.io/directxtex-directxmesh-and-uvatlas-now-support-linux/

참고자료

- Metal개발환경 설정
  - https://developer.apple.com/kr/metal/cpp/
- Metal C++ 소개
  - https://developer.apple.com/videos/play/wwdc2022/10160/
- Metal Shader Language
  - https://developer.apple.com/metal/Metal-Shading-Language-Specification.pdf
- Sample code
  - https://developer.apple.com/news/?id=yrb1yuf3

데모

- https://youtu.be/O7ISRjII3Sk
- VOXEL HORIZON Trailers
  - https://youtube.com/playlist?list=PL00yTT-RECdUPitVS0nclg25WkrwzeUMI
- VOXEL HORIZON for iOS/macOS 1차 영상
  - https://youtu.be/wbTBozP9yVk
- VOXEL HORIZON for iOS/macOS mirroring
  - https://youtu.be/O7ISRjII3Sk
- VOXEL HORIZON for iOS/macOS on mac
  - https://youtu.be/K32Xvkk0GKA