Bindless Resource

목차

개요

Bindless Resource는 셰이더에서 리소스를 사용하는 새로운 방식으로 Direct3D 12, Vulkan, Metal과 같은 최신 그래픽스 API에서 제공되는 기능입니다. 이 글에서는 Bindless Resource를 사용하는 경우 어떤 면이 달라지는지 자체 구현한 Direct3D 12 샘플을 통해 살펴보겠습니다. Direct3D 11, 12의 API를 예시로 설명할 것이기 때문에 해당 API에 대한 지식이 있으면 이해하기 쉬울 것입니다. 다만 모르셔도 Bindless Resource를 이해하실 수 있도록 필요한 부분은 다루도록 하겠습니다.

Resource Binding

우선 Bindless Resource를 사용하지 않았을 때 셰이더에서 리소스를 어떻게 사용했는지 알아보겠습니다. 렌더링을 위한 모델이나 텍스쳐와 같은 리소스를 생성하고 나면 이를 그래픽 파이프라인의 셰이더에서 사용할 수 있도록 연결하는 과정이 필요합니다. 이 과정을 **리소스 바인딩 (Resource Binding)**이라고 부릅니다.

Direct3D 11의 경우

일단 비교적 간단한 그래픽스 API인 Direct3D 11의 API 예시를 통해서 리소스 바인딩의 과정을 알아보겠습니다. 다음과 같은 픽셀 셰이더 코드가 있습니다.

```
Texture2D DiffuseTex : register( t0 );
SamplerState DiffuseTexSampler : register( s0 );

float4 main( PS_INPUT input ) : SV_Target0
{
    return DiffuseTex.Sample( DiffuseTexSampler, input.texcoord );
}
```

DiffuseTex 텍스쳐를 DiffuseTexSampler 샘플러로 샘플링한 결과를 반환하는 간단한 텍스쳐 샘플링 코드입니다.

Direct3D 11에서는 리소스 바인딩을 위한 함수를 셰이더 단계별로 리소스의 종류마다 제공하고 있습니다. 다음이 리소스 바인딩을 위해 제공되는 함수들입니다.

```
/* 상수 버퍼 */
ID3D11DeviceContext::CSSetConstantBuffers
ID3D11DeviceContext::DSSetConstantBuffers
ID3D11DeviceContext::GSSetConstantBuffers
ID3D11DeviceContext::HSSetConstantBuffers
```

```
ID3D11DeviceContext::PSSetConstantBuffers
ID3D11DeviceContext::VSSetConstantBuffers

/* 샘플러 */
ID3D11DeviceContext::CSSetSamplers
ID3D11DeviceContext::DSSetSamplers
ID3D11DeviceContext::GSSetSamplers
ID3D11DeviceContext::HSSetSamplers
ID3D11DeviceContext::PSSetSamplers
ID3D11DeviceContext::VSSetSamplers

/* 셰이더 리소스 뷰 */
ID3D11DeviceContext::CSSetShaderResources
ID3D11DeviceContext::DSSetShaderResources
ID3D11DeviceContext::GSSetShaderResources
ID3D11DeviceContext::HSSetShaderResources
ID3D11DeviceContext::PSSetShaderResources
ID3D11DeviceContext::VSSetShaderResources

/* 순서가 지정되지 않은 액세스 뷰 */
ID3D11DeviceContext::CSSetUnorderedAccessViews
```

픽셀 셰이더에서 사용되는 DiffuseTex와 DiffuseTexSampler에 실제 리소스를 연결해 주기 위해서 CPU 측에서 아래와 같은 예제 코드의 형태로 리소스를 바인딩할 수 있습니다.

```
/*복수의 리소스를 바인딩 가능*/
deviceContext.PSSetShaderResources( 0 /*바인딩 시작 슬롯 */, 1 /* 바인딩할 리소스 갯수 */, <텍스쳐 리소스
deviceContext.PSSetSamplers( 0, 1, <샘플러> );
```

셰이더 코드에서 DiffuseTex와 DiffuseTexSampler를 0번 슬롯(DiffuseTex의 경우에는 t0로 DiffuseTexSampler의 경우에는 s0로)으로 선언했기 때문에 해당 슬롯에 리소스가 바인딩 될 수 있도록 인자를 전달하는 것을 볼 수 있습니다.

Direct3D 12의 경우

Direct3D 12에서도 리소스 바인딩은 사라지지 않았습니다. 하지만 그 과정은 Direct3D 11보다 복잡해졌습니다.

Direct3D 11에서의 리소스 바인딩 과정은 기성품을 사용하는 것으로 생각할 수 있습니다. Direct3D 11은 미리 일정한 규격대로 슬롯의 개수를 정해 놓은 그래픽스 파이프라인을 제공하며 사용자는 API를 사용하여 자신이 원하는 슬롯에 원하는 리소스를 자유롭게 바인딩할 수 있습니다.

반면 **Direct3D 12에서의 리소스 바인딩 과정은 주문 제작**입니다. 사용자가 자신이 몇 개의 슬롯을 사용할 것 인지를 적어서 제출하면 해당하는 슬롯만 사용할 수 있습니다. 이를 Root Signature라고 하며 Root Signature를 통해 커스텀된 그래픽스 파이프라인을 생성하게 됩니다. 앞에서 예로 든 셰이더 코드를 위한 그래픽스 파이프라인을 생성하는 코드 샘플을 살펴보겠습니다.

```
D3D12_ROOT_PARAMETER param = {
    .ParameterType = D3D12_ROOT_PARAMETER_TYPE_SRV, /* SRV 리소스에 대한 슬롯 정보 */
    .Descriptor = {
        .ShaderRegister = 0, /* 레지스터(=슬롯) 번호 */
        .RegisterSpace = 0 /* 스페이스 번호 */
    },
    .ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL /* 해당 슬롯이 노출될 셰이더 스테이지 */
};

D3D12_STATIC_SAMPLER_DESC sampler = {
    .Filter = D3D12_FILTER_MIN_MAG_MIP_LINEAR,
    .AddressU = D3D12_TEXTURE_ADDRESS_MODE_CLAMP,
    .AddressV = D3D12_TEXTURE_ADDRESS_MODE_CLAMP,
    .AddressW = D3D12_TEXTURE_ADDRESS_MODE_CLAMP,
    .MipLODBias = 0,
    .MaxAnisotropy = 0,
```

```
    .ComparisonFunc = D3D12_COMPARISON_FUNC_NEVER,
    .BorderColor = D3D12_STATIC_BORDER_COLOR_OPAQUE_BLACK,
    .MinLOD = 0.f,
    .MaxLOD = D3D12_FLOAT32_MAX, /* 여기까지는 샘플러 속성 */
    .ShaderRegister = 0, /* 레지스터(=슬롯) 번호 */
    .RegisterSpace = 0, /* 스페이스 번호 */
    .ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL /* 해당 슬롯이 노출될 셰이더 스테이지 */
};

/* Root Signature 생성을 위한 구조체 초기화 */
D3D12_ROOT_SIGNATURE_DESC desc = {
    .NumParameters = 1, // 리소스를 위한 구조체 갯수
    .pParameters = &param, // 리소스를 위한 구조체 전달
    .NumStaticSamplers = 1, // 샘플러 갯수
    .pStaticSamplers = &sampler // 샘플러를 위한 구조체 전달
};

ComPtr<ID3DBlob> signature;
ComPtr<ID3DBlob> error;

D3D12SerializeRootSignature( &desc, D3D_ROOT_SIGNATURE_VERSION_1, signature.GetAddressOf(), erro

ID3D12RootSignature* rootSignature = nullptr;

/* Root Signature 생성 */
D3D12Device().CreateRootSignature( 0, signature->GetBufferPointer(), signature->GetBufferSize(),

/* Pipeline State 생성을 위한 구조체 초기화 */
D3D12_GRAPHICS_PIPELINE_STATE_DESC pipelineDesc = {
    .pRootSignature = rootSignature,
    /* 나머지 변수 초기화는 생략 */
};

ID3D12PipelineState* pipelineState = nullptr;
D3D12Device().CreateGraphicsPipelineState( &pipelineDesc, IID_PPV_ARGS( &pipelineState ) );
```

이렇게 파이프라인을 생성하고 다음과 같이 리소스를 바인딩합니다.

```
ID3D12DescriptorHeap* heaps[] = {
    <DescriptorHeap에 대한 포인터>,
};

commandList.SetDescriptorHeaps( std::extent_v<decltype( heaps )>, heaps );
commandList.SetGraphicsRootShaderResourceView( 0/* Root Signature 생성시 해당 리소스의 슬롯이 위치한 인
```

먼저 SetDescriptorHeaps이라는 함수를 통해서 Descriptor Heap을 설정해 줍니다. Descriptor Heap은 셰이더 리소스 뷰, 순서가 지정되지 않은 액세스 뷰, 상수 버퍼 뷰, 샘플러와 같은 리소스를 담은 메모리 공간입니다. **SetDescriptorHeaps 함수 사용에는 중요한 제약점이 있는데 같은 종류의 Descriptor Heap을 하나 이상 세팅할 수 없다**는 점입니다. Descriptor Heap는 다음과 같이 4가지 종류가 존재합니다.

```
typedef
enum D3D12_DESCRIPTOR_HEAP_TYPE
{
    D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV  = 0,
    D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER  = ( D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV + 1 ) ,
    D3D12_DESCRIPTOR_HEAP_TYPE_RTV  = ( D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER + 1 ) ,
    D3D12_DESCRIPTOR_HEAP_TYPE_DSV  = ( D3D12_DESCRIPTOR_HEAP_TYPE_RTV + 1 ) ,
```

```
    D3D12_DESCRIPTOR_HEAP_TYPE_NUM_TYPES    = ( D3D12_DESCRIPTOR_HEAP_TYPE_DSV + 1 )
} D3D12_DESCRIPTOR_HEAP_TYPE;
```

여기서 RTV와 DSV는 셰이더의 입력으로 바인딩하지 않으므로 제외하면 CBV_SRV_UAV와 SAMPLER 2종류만 SetDescriptorHeaps 함수로 동시에 설정할 수 있습니다. 예시 코드에서 샘플러는 상수로 Root Signature에 전달되었기 때문에 텍스쳐를 위한 Descriptor Heap만 설정하고 있습니다.

Descriptor Heap 설정 후에는 리소스 종류마다 준비된 함수를 통해 리소스를 바인딩합니다. 다음과 같은 함수가 제공됩니다.

```
ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstant /* 4byte 상수 변수 */
ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstants /* 4byte 상수 변수 */
ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView /* 상수 버퍼 */
ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable /* Descriptor Table */
ID3D12GraphicsCommandList::SetGraphicsRootShaderResourceView /* 셰이더 리소스 뷰 */
```

예시 코드에서는 SetGraphicsRootShaderResourceView을 사용해서 셰이더 리소스 뷰를 바인딩하고 있습니다. 첫 번째 인자로 0을 전달하고 있는데 Root Signature를 생성할 때 해당하는 셰이더 리소스 뷰에 대한 슬롯 정보가 Root Signature 생성 구조체의 첫번째 인덱스에 자리 잡고 있기 때문입니다.

Bindless Resource

여기까지 리소스 바인딩에 대해서 살펴보았습니다. 그럼, Bindless Resource는 무엇일까요? 바인딩이 없다는 건 무슨 뜻일까요? Bindless Resource를 사용하는 예시를 통해 알아보겠습니다. Bindless Resource를 사용할 때 셰이더 코드는 다음과 같이 변하게 됩니다.

```
Texture2D Tex2Ds[] : register( t0 );
Texture2DArray Tex2DArrays[] : register(t0, space1);
TextureCube TexCubes[] : register(t0, space2);
Texture3D Tex3Ds[] : register(t0, space3);

SamplerState Samplers[] : register( s0 );

cbuffer BindlessResourceIndex : register( b0 )
{
    int DiffuseTex;
    int DiffuseTexSampler;
};

float4 main( PS_INPUT input ) : SV_Target0
{
    return Tex2Ds[DiffuseTex].Sample( Samplers[DiffuseTexSampler], input.texcoord );
}
```

리소스의 선언이 **범위가 지정되지 않은 배열**로 변한 것을 볼 수 있습니다. 그리고 실제 리소스는 상수 버퍼를 통해서 전달된 인덱스를 통해 참조하게 됩니다. Root Signature를 생성하는 과정도 다음과 같이 변경됩니다.

```
/* 4개의 SRV(Tex2Ds, Tex2DArrays, TexCubes, Tex3Ds) 를 위한 구조체 초기화 */
D3D12_DESCRIPTOR_RANGE srvRange[4] = {};
for ( unsigned int i = 0; i < 4; ++i )
{
    srvRange[i] = {
        .RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV,
        .NumDescriptors = (unsigned int)-1, /* 리소스 갯수 범위가 지정되지 않았으므로 최대값으로 설정 */
        .BaseShaderRegister = 0, /* 레지스터 번호 */
        .RegisterSpace = i, /* 스페이스 번호 */
        .OffsetInDescriptorsFromTableStart = 0 /* Descriptor Heap에서의 오프셋 */
    };
}
```

```
/* 샘플러를 위한 구조체 초기화 */
D3D12_DESCRIPTOR_RANGE samplerRange = {
    .RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER,
    .NumDescriptors = (unsigned int)-1,
    .BaseShaderRegister = 0,
    .RegisterSpace = 0,
    .OffsetInDescriptorsFromTableStart = 0
};

D3D12_ROOT_PARAMETER param[3] = {};
/* SRV를 위한 구조체 초기화 */
param[0] = {
    .ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE,
    .DescriptorTable = {
        .NumDescriptorRanges = 4,
        .pDescriptorRanges = srvRange
    },
    .ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL /* 모든 셰이더 단계에서 노출 */
};

/* 샘플러를 위한 구조체 초기화 */
param[1] = {
    .ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE,
    .DescriptorTable = {
        .NumDescriptorRanges = 1,
        .pDescriptorRanges = &samplerRange
    },
    .ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL
};

/* 인덱스 전달용 상수 버퍼를 위한 구조체 초기화 */
param[2] = {
    .ParameterType = D3D12_ROOT_PARAMETER_TYPE_CBV,
    .Constants = {
        .ShaderRegister = 0,
        .RegisterSpace = 0,
        .Num32BitValues = 2, /* 4Byte변수를 2개 사용하므로 2로 설정 */
    }
};

D3D12_ROOT_SIGNATURE_DESC desc = {
    .NumParameters = 3,
    .pParameters = param,
    .NumStaticSamplers = 0, /* 상수 샘플러 사용하지 않음 */
    .pStaticSamplers = nullptr
};

ComPtr<ID3DBlob> signature;
ComPtr<ID3DBlob> error;

D3D12SerializeRootSignature( &desc, D3D_ROOT_SIGNATURE_VERSION_1, signature.GetAddressOf(), erro

/* 이하 생략 */
```

여기까지 보시면 “Bindless Resource를 사용하지 않을 때 보다 더 복잡해지지 않았나요?” 의문을 가지실 수 있을 거라 생각됩니다.

지금까지 살펴본 리소스 바인딩의 샘플은 하나의 텍스쳐, 하나의 샘플러라는 아주 간단한 예시였습니다. 하지만 **실제로 우리가 프로그램을 작성할 때 셰이더 파이프라인에 바인딩해야 하는 리소스는 상황에 따라 달라 다양한 조합이 필요**하게 됩니다. 그리고 그러한 조합마다 **Root**

Signature를 새로 만들어야 합니다. 그리고 Root Signature가 새로 만들어지면 파이프라인도 당연히 새로 생성해야 합니다.

반면 위와 같은 구성으로 Bindless Resource를 사용하는 경우에 **Root Signature를 새로 생성하지 않아도 다양한 셰이더 리소스 뷰, 샘플러의 조합에 대응**할 수 있습니다. 셰이더 리소스의 조합마다 Root Signature를 새로 생성하기 위한 코드를 작성하고 파이프라인 상태 생성 비용을 치러야 하는 상황에서 자유로워질 수 있습니다.

Dynamic Resource

Shader Model 6.6에서는 Bindless Resource에 더해 Dynamic Resource라는 기능이 새롭게 추가 되었습니다. 이 기능을 사용하면 셰이더 코드는 다음과 같이 변경됩니다.

```
cbuffer BindlessResourceIndex : register( b0 )
{
    int DiffuseTex;
    int DiffuseTexSampler;
};

float4 main( PS_INPUT input ) : SV_Target0
{
    Texture2D tex = ResourceDescriptorHeap[NonUniformResourceIndex(DiffuseTex)];
    SamplerState sam = SamplerDescriptorHeap[NonUniformResourceIndex(DiffuseTexSampler)];

    return tex.Sample( sam, input.texcoord );
}
```

ResourceDescriptorHeap과 SamplerDescriptorHeap 이라는 새로운 HLSL 내장 오브젝트를 통해서 SetDescriptorHeaps로 설정한 리소스에 인덱스를 통해 접근할 수 있습니다.

따라서 **범위가 지정되지 않은 리소스 배열을 바인딩할 필요가 사라지기 때문에 리소스 인덱스가 담긴 상수 버퍼만 바인딩하면 되어 Root Signature 작성 과정이 더욱더 간략화** 됩니다.

참고 사이트에 따르면 Dynamic Resource 사용 시에는 Root Signature 생성 시 다음과 같은 설정이 필요하다고 합니다.

1. Root Signature 플래그로 `D3D12_ROOT_SIGNATURE_FLAG_CBV_SRV_UAV_HEAP_DIRECTLY_INDEXE` 와 `D3D12_ROOT_SIGNATURE_FLAG_SAMPLER_HEAP_DIRECTLY_INDEXED` 를 설정합니다.
2. SetDescriptorHeaps 함수를 SetGraphicsRootSignature 혹은 SetComputeRootSignature 함수 호출 전에 먼저 호출해서 Descriptor Heap을 설정합니다.
3. Descriptor Range의 플래그로 `DESCRIPTORS_VOLATILE` 혹은 `DATA_VOLATILE` 을 설정합니다.

구현 사례

구현한 Direct3D 12 샘플을 통해 Bindless Resource를 살펴보겠습니다. Dynamic Resource의 경우 Shader Model 6.6이 필요한데 테스트 PC가 Shader Model 6.6을 미지원하여 여기서는 Dynamic Resource는 사용하지 않습니다.

먼저 D3D12BindlessManager 클래스를 살펴보도록 하겠습니다. 앞에서 SetDescriptorHeaps 함수가 동일한 종류의 Descriptor Heap을 하나만 동시에 세팅할 수 있다고 했었는데요. 이 말인즉슨 **렌더링에 필요한 모든 리소스를 담고 있는 Descriptor Heap이 필요**합니다. D3D12BindlessManager 클래스는 Descriptor Heap을 std::vector와 같이 만든 클래스로 파이프라인에 필요한 모든 리소스를 담은 Descriptor Heap입니다.

```
class D3D12BindlessManager
{
public:
    /* SRV, UAV, CBV 디스크립터를 추가하고 해당하는 핸들을 반환 */
    [[nodiscard]] int32 AddDescriptor( D3D12CpuDescriptorHandle handle );

    /* 핸들을 넘겨 해당 디스크립터를 제거 */
    void RemoveDescriptor( int32 bindlessHandle );

    /* 샘플러의 디스크립터를 추가하고 해당하는 핸들을 반환 */
    [[nodiscard]] int32 AddSamplerDescriptor( D3D12CpuDescriptorHandle handle );
```

```
    /* 핸들을 넘겨 해당 디스크립터를 제거 */
    void RemoveSamplerDescriptor( int32 bindlessHandle );

    /* SRV, UAV, CBV의 Descriptor Heap을 반환 */
    D3D12DescriptorHeap& GetHeap();
    const D3D12DescriptorHeap& GetHeap() const;

    /* 샘플러의 Descriptor Heap을 반환 */
    D3D12DescriptorHeap& GetSamplerHeap();
    const D3D12DescriptorHeap& GetSamplerHeap() const;

    D3D12BindlessManager();

private:
    D3D12BindlessDescriptorHeap m_descriptorHeap;
    D3D12BindlessDescriptorHeap m_samplerDescriptorHeap;
};
```

D3D12BindlessDescriptorHeap 클래스가 실제 std::vector와 같은 동적 배열의 로직을 구현하고 있습니다.

```
class D3D12BindlessDescriptorHeap
{
public:
    [[nodiscard]] int32 Add( D3D12CpuDescriptorHandle handle );
    void Remove( int32 bindlessHandle );

    D3D12DescriptorHeap& GetHeap();
    const D3D12DescriptorHeap& GetHeap() const;

    explicit D3D12BindlessDescriptorHeap( D3D12_DESCRIPTOR_HEAP_TYPE type );

private:
    void Grow();

    D3D12_DESCRIPTOR_HEAP_TYPE m_type;

    D3D12DescriptorHeap m_cpuHeap;
    D3D12DescriptorHeap m_gpuHeap;

    uint32 m_size = 0;
    uint32 m_capacity = 0;

    BitArray m_freeFlag;
};
```

주요 함수인 Add와 Grow의 구현을 보면 다음과 같습니다.

```
int32 D3D12BindlessDescriptorHeap::Add( D3D12CpuDescriptorHandle handle )
{
    /* 크기가 한도보다 크면 Grow */
    if ( m_size >= m_capacity )
    {
        Grow();
    }

    /* 비어 있는 인덱스를 검색 */
    auto freeIdx = static_cast<int32>( m_freeFlag.FindFirstSetBit() );
```

```
    assert( freeIdx != m_freeFlag.Size() );

    /* 해당 인덱스에 할당 */
    D3D12Device().CopyDescriptorsSimple( 1, m_cpuHeap.GetCpuHandle().At( freeIdx ), handle.At(),
    D3D12Device().CopyDescriptorsSimple( 1, m_gpuHeap.GetCpuHandle().At( freeIdx ), handle.At(),

    m_freeFlag[freeIdx] = false;
    ++m_size;

    return freeIdx;
}

void D3D12BindlessDescriptorHeap::Grow()
{
    /* 1.5배씩 증가 */
    auto newCapacity = static_cast<uint32>( m_capacity * 1.5f + 1 );
    auto newCpuHeap = D3D12DescriptorHeapAllocator::GetInstance().AllocCpuDescriptorHeap( m_type
    auto newGpuHeap = D3D12DescriptorHeapAllocator::GetInstance().AllocGpuDescriptorHeap( m_type

    /* 기존 내용 복사 */
    if ( m_capacity > 0 )
    {
        D3D12Device().CopyDescriptorsSimple( m_capacity, newCpuHeap.GetCpuHandle().At(), m_cpuHe
        D3D12Device().CopyDescriptorsSimple( m_capacity, newGpuHeap.GetCpuHandle().At(), m_cpuHe
    }

        /* 증가된 값으로 멤버 변수 초기화 */
    m_capacity = newCapacity;
    m_cpuHeap = newCpuHeap;
    m_gpuHeap = newGpuHeap;
    m_freeFlag.Resize( m_capacity, true );
}
```

셰이더 리소스 뷰, 순서가 지정되지 않은 액세스 뷰, 상수 버퍼 뷰, 샘플러와 같은 리소스 생성 시에 다음과 같이 등록합니다.

```
/* SRV 초기화 */
void D3D12ShaderResourceView::InitResource()
{
    m_descriptorHeap = D3D12DescriptorHeapAllocator::GetInstance().AllocCpuDescriptorHeap( D3D12
    D3D12Device().CreateShaderResourceView( m_d3d12Resource, &m_desc, m_descriptorHeap.GetCpuHan

    m_bindlessHandle = D3D12BindlessMgr().AddDescriptor( m_descriptorHeap.GetCpuHandle() );
}

/* SRV 해제 */
void D3D12ShaderResourceView::FreeResource()
{
    BaseClass::FreeResource();

    D3D12BindlessMgr().RemoveDescriptor( m_bindlessHandle );
}
```

Root Descriptor 구성은 이전에 보았던 예시와 크게 다르지 않습니다.

```
/* 셰이더의 리소스 선언
#if SupportsBindless == 1
Texture2D Tex2D[] : register(t0, space100);
Texture2DArray Tex2DArray[] : register(t0, space101);
```

```
TextureCube TexCube[] : register(t0, space102);
Texture3D Tex3D[] : register(t0, space103);

SamplerState Samplers[] : register(s0, space100);

#define DefineBindlessIndices cbuffer BindlessIndices : register( b0 )
#endif
*/

void D3D12RootSignature::InitializeForBindless( InlineShaderArray& shaders )
{
    /* SRV를 위한 구조체 초기화 */
    {
        constexpr int32 MaxStandardSrvCount = 4;
        for ( int32 i = 0; i < MaxStandardSrvCount; ++i )
        {
            D3D12_DESCRIPTOR_RANGE& range = m_descritorRange.emplace_back();

            range.RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
            range.NumDescriptors = (uint32)-1;
            range.BaseShaderRegister = 0;
            range.RegisterSpace = 100 + i;
            range.OffsetInDescriptorsFromTableStart = 0;
        }

        D3D12_ROOT_PARAMETER& param = m_parameters.emplace_back();

        param.ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
        param.ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;
        param.DescriptorTable.NumDescriptorRanges = MaxStandardSrvCount;
        param.DescriptorTable.pDescriptorRanges = &m_descritorRange[0];
    }

    /* Sampler를 위한 구조체 초기화 */
    {
        D3D12_DESCRIPTOR_RANGE& range = m_descritorRange.emplace_back();

        range.RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER;
        range.NumDescriptors = (uint32)-1;
        range.BaseShaderRegister = 0;
        range.RegisterSpace = 100;
        range.OffsetInDescriptorsFromTableStart = 0;

        D3D12_ROOT_PARAMETER& param = m_parameters.emplace_back();

        param.ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
        param.ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;
        param.DescriptorTable.NumDescriptorRanges = 1;
        param.DescriptorTable.pDescriptorRanges = &range;
    }

    /* 바인딩할 리소스를 위한 구조체 초기화 부분은 생략 */
}
```

그리고 렌더링 함수 호출 전 다음과 같이 Descriptor Heap을 파이프라인에 설정합니다.

```
void D3D12PipelineCache::BindBindlessResources( ID3D12GraphicsCommandList6& commandList, GlobalC
{
```

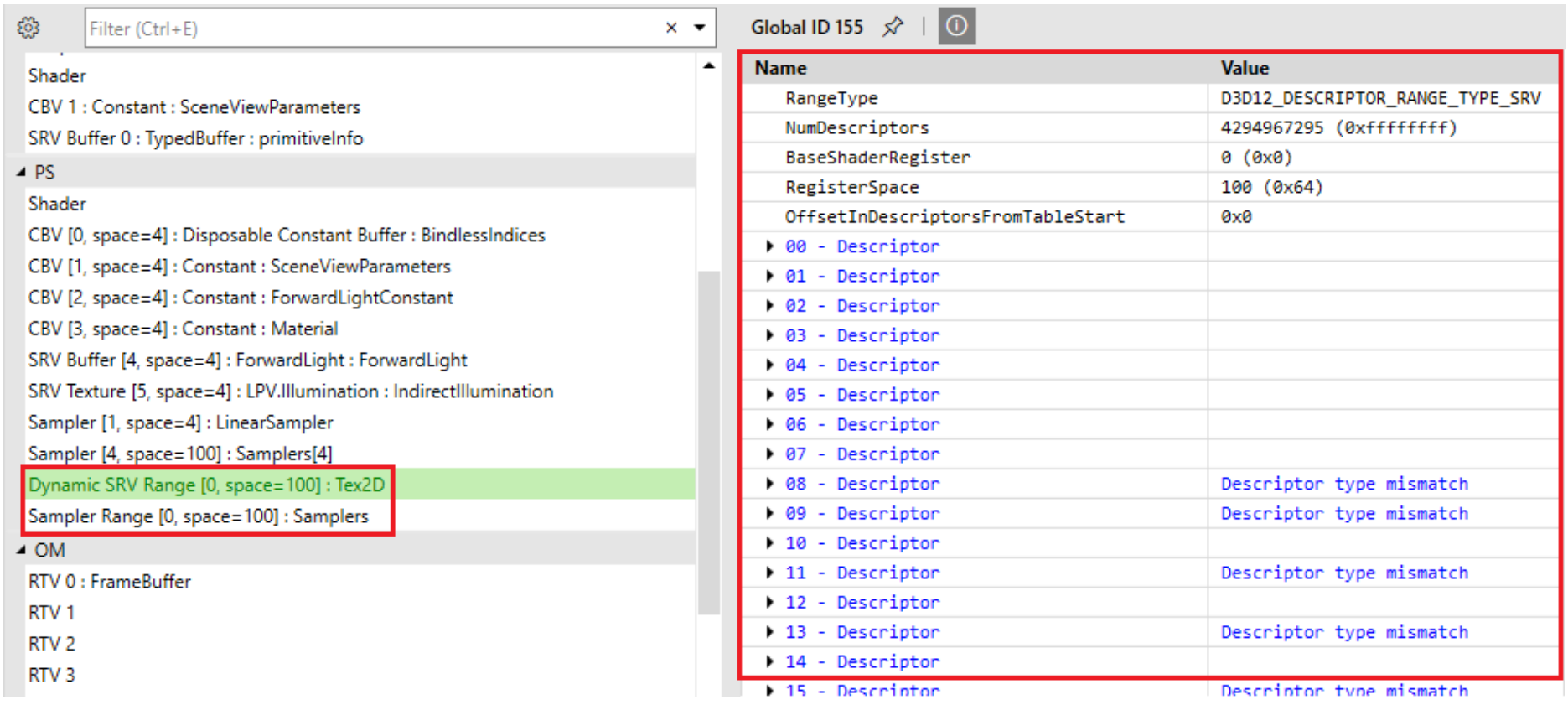
```
    /* 생략 */
    ID3D12DescriptorHeap* heaps[] = {
        D3D12BindlessMgr().GetHeap().Resource(),
        D3D12BindlessMgr().GetSamplerHeap().Resource()
    };

    RegisterRenderResource( heaps[0] );
    RegisterRenderResource( heaps[1] );

    commandList.SetDescriptorHeaps( std::extent_v<decltype( heaps )>, heaps );

    /* 생략 */
    else
    {
        commandList.SetGraphicsRootDescriptorTable( 0, D3D12BindlessMgr().GetHeap().GetGpuHandle
        commandList.SetGraphicsRootDescriptorTable( 1, D3D12BindlessMgr().GetSamplerHeap().GetGp

        /* 생략 */
    }

    /* 생략 */
}
```

실제 셰이더에서는 다음과 같이 텍스쳐를 샘플링하게 됩니다.

```
#include "Common/BindlessResources.fxh"
#include "Common/LightCommon.fxh"

#if SupportsBindless == 1
DefineBindlessIndices
{
    int DiffuseTex;
    int DiffuseTexSampler;
};
#else
Texture2D DiffuseTex : register( t2 );
SamplerState DiffuseTexSampler : register( s2 );
#endif

struct PS_INPUT
{
    float4 position : SV_POSITION;
    float3 worldPos : POSITION0;
    float3 viewPos : POSITION1;
    float4 projectionPos : POSITION2;
    float3 normal : NORMAL;
    float2 texcoord : TEXCOORD;
};

float4 main( PS_INPUT input ) : SV_Target0
{
    GeometryProperty geometry = (GeometryProperty)0;
    geometry.worldPos = input.worldPos;
    geometry.viewPos = input.viewPos;
    geometry.normal = input.normal;
    geometry.screenUV = ( input.projectionPos.xy / input.projectionPos.w ) * float2( 0.5f, -0.5f

    LIGHTCOLOR cColor = CalcLight( geometry );
```

```
#if SupportsBindless == 1
    float4 lightColor = (float4)0.f;
    if ( DiffuseTex > -1 && DiffuseTexSampler > -1 )
    {
        lightColor = cColor.m_diffuse * MoveLinearSpace( Tex2D[DiffuseTex].Sample( Samplers[Dif
    }
#else
    float4 lightColor = cColor.m_diffuse * MoveLinearSpace( DiffuseTex.Sample( DiffuseTexSample
#endif
    lightColor += cColor.m_specular * MoveLinearSpace( Specular );

    return float4( lightColor.rgb, 1.f );
}
```

PIX를 통해서 샘플의 장면을 캡처해서 보면 아래와 같이 Bindless Resource가 사용된 것을 확인할 수 있습니다.



이용 사례

이제 Bindless Resource를 사용하여 어떤 일을 할 수 있는지 사례를 살펴보겠습니다. 지금부터 소개할 사례 하나하나가 큰 주제이기 때문에 여기서는 겉핥기로 간략하게 소개하겠습니다.

Ray Tracing

레이 트레이싱에서 Bindless Resource를 통해 광선이 충돌한 표면에 관련된 리소스에 접근할 수 있습니다.
https://github.com/TheRealMJP/DXRPathTracer 의 RayTrace.hlsl 셰이더 코드의 일부분을 살펴보겠습니다.

```
[shader("closesthit")]
void ClosestHitShader(inout PrimaryPayload payload, in HitAttributes attr)
{
    const MeshVertex hitSurface = GetHitSurface(attr, GeometryIndex());
    const Material material = GetGeometryMaterial(GeometryIndex());

    payload.Radiance = PathTrace(hitSurface, material, payload);
}
```

ClosestHitShader는 광선이 어떤 기하 표면에 부딪혔을 때 호출되는 셰이더입니다. GeometryIndex()는 hlsl 내장 함수로 현재 셰이더에 관련하여 광선이 충돌한 기하 표면의 인덱스를 반환합니다. 이 인덱스를 가지고 GetGeometryMaterial() 함수를 호출하여 다음과 같이 기하 표면에 대한 재질 정보를 얻습니다.

```
Material GetGeometryMaterial(in uint geometryIdx)
{
    StructuredBuffer<GeometryInfo> geoInfoBuffer = ResourceDescriptorHeap[RayTraceCB.GeometryInf
    const GeometryInfo geoInfo = geoInfoBuffer[geometryIdx];

    StructuredBuffer<Material> materialBuffer = ResourceDescriptorHeap[RayTraceCB.MaterialBuffer
    return materialBuffer[geoInfo.MaterialIdx];
}
```

재질 정보에는 다음과 같이 텍스쳐 참조를 위한 인덱스들이 담겨 있습니다.

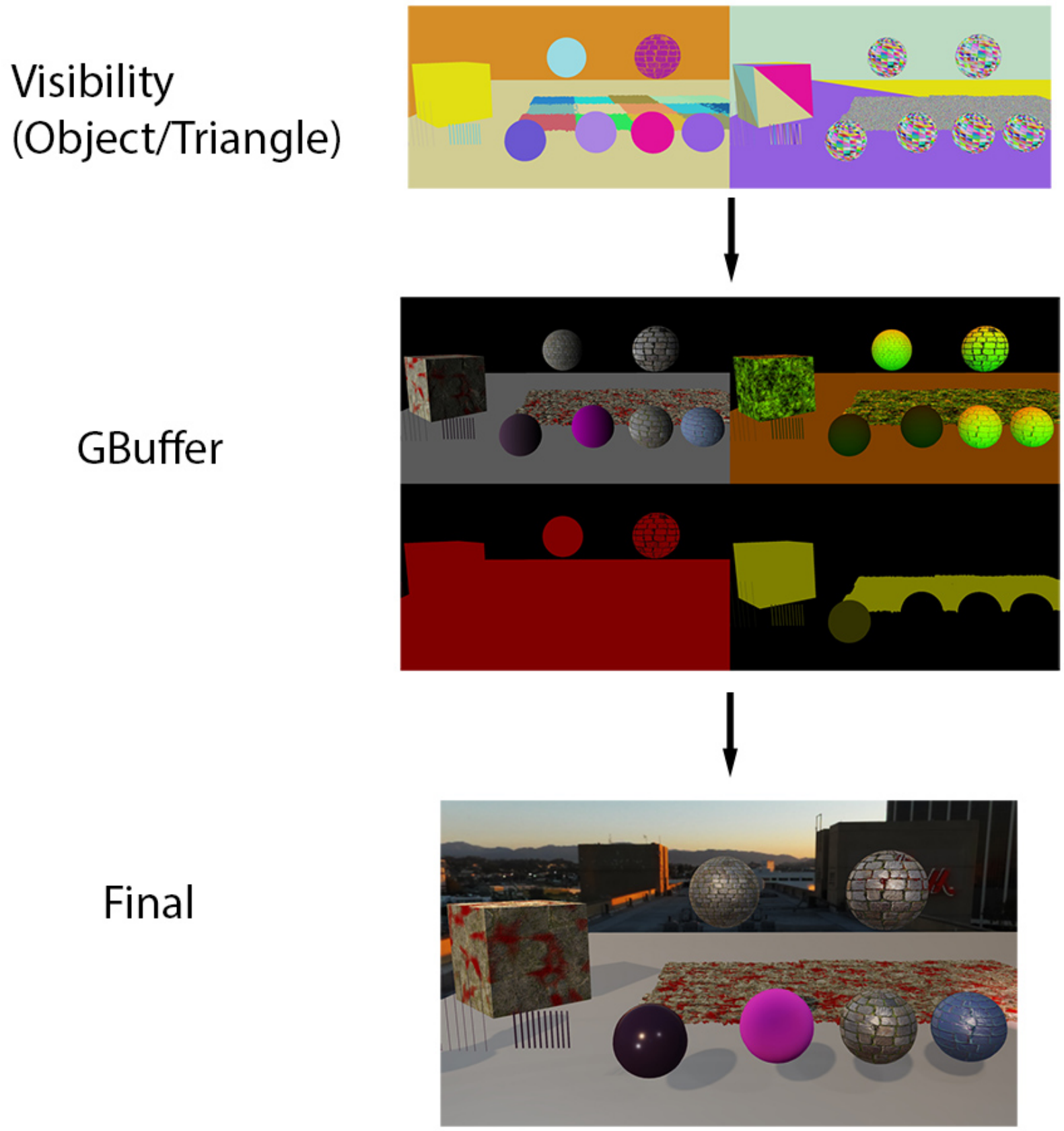
```
struct Material
{
    uint Albedo;
    uint Normal;
    uint Roughness;
    uint Metallic;
    uint Opacity;
    uint Emissive;
};
```

PathTrace 함수의 내부를 보면 다음과 같이 인덱스를 이용해서 텍스쳐에 접근하고 있는 것을 확인할 수 있습니다.

```
static float3 PathTrace(in MeshVertex hitSurface, in Material material, in PrimaryPayload inPayl
{
    /* 생략 */
    float3 baseColor = 1.0f;
    if(AppSettings.EnableAlbedoMaps && !AppSettings.EnableWhiteFurnaceMode)
    {
        /* Dynamic Resource를 통한 Albedo Map 샘플링 */
        Texture2D albedoMap = ResourceDescriptorHeap[NonUniformResourceIndex(material.Albedo)];
        baseColor = albedoMap.SampleLevel(MeshSampler, hitSurface.UV, 0.0f).xyz;
    }
    /* 생략 */
}
```

Visibility Buffer

Visibility Buffer는 렌더링시 렌더 오브젝트의 ID와 삼각형의 ID를 렌더 타겟에 기록해 놓은 버퍼로 해당 버퍼를 이용해서 **재질에 대한 텍스쳐 링을 지연해서 처리**합니다. 이는 라이팅을 지연처리하는 디퍼드 셰이딩과 유사한 방식으로 볼 수 있으며 주로 디퍼드 셰이딩을 위한 GBuffer 렌더링을 최적화 하기 위하여 사용됩니다.



출처 : http://filmicworlds.com/blog/visibility-buffer-rendering-with-material-graphs/

Xbox-GDK-Samples의 Visibility Buffer 샘플의 일부를 보면 Visibilty Buffer를 통해 텍스쳐링을 처리하는 SceneReconstructionCS.hlsl 에서 다음과 같이 Dynamic Resource를 사용하여 텍스쳐를 참조하는 것을 볼 수 있습니다.

```
[numthreads(8, 8, 1)]
[RootSignature(MainRS)]
void main(uint3 DTid : SV_DispatchThreadID)
{
    // Get handles to the input and output buffers using dynamic access.
    // ResourceDescriptorHeap[] is a new keyword in HLSL SM 6.6 allowing dynamic access to resou
    Texture2D<uint> Visibility = ResourceDescriptorHeap[Descriptors::VisibilityBuffer];

    /* 생략 */

    /* Visibility Buffer에서 Load */
    uint visibility = Visibility.Load(int3(DTid));

    /* 생략 */
```

```
    // Parse object and primitive IDs out of 32-bit uint. objectID == 0 is reserved to signify n
    /* objectID, primitiveID로 분리 */
    uint objectID = ((visibility >> 20) & 0xfff) - 1;
    uint primitiveID = visibility & 0xfffff;

    // Use dynamic resource access to get a handle to the array of ObjectInfo describing objects
    StructuredBuffer<ObjectInfo> ObjectInfoArray = ResourceDescriptorHeap[Descriptors::ObjectInf

    // Get the object at this pixel out of the info array, must be nonuniform as a wave could co
    /* objectID로 object 정보를 얻어 옴 */
    ObjectInfo object = ObjectInfoArray[objectID];

    /* 생략 */

    // Get handles to the texture and sampler for this object using dynamic access. Must be nonu
    /* object 정보를 통해 Diffuse 텍스쳐를 얻어 옴 */
    Texture2D<float4> txDiffuse = ResourceDescriptorHeap[NonUniformResourceIndex(Descriptors::Di

    // SamplerDescriptorHeap[] is a new keyword in HLSL SM 6.6 allowing dynamic access to sample
    SamplerState sampleState = SamplerDescriptorHeap[NonUniformResourceIndex(Samplers::LinearSam

    // Sample texture using reconstructed UVs and derivatives.
    float4 texColour = txDiffuse.SampleGrad(sampleState, uvs, dUVdx, dUVdx);

    /* 생략 */
}
```

마치며

이번에 정리한 내용은 여기까지입니다. Bindless Resource 샘플 코드는 아래 링크를 참고 부탁드립니다. 감사합니다.

Reference

1. https://github.com/xtozero/SSR/tree/bindless
2. https://github.com/TheRealMJP/DeferredTexturing
3. https://microsoft.github.io/DirectX-Specs/d3d/HLSL_SM_6_6_DynamicResources.html
4. https://github.com/TheRealMJP/DXRPathTracer
5. https://github.com/microsoft/Xbox-GDK-Samples
6. http://filmicworlds.com/blog/visibility-buffer-rendering-with-material-graphs/