描画パイプライン解説

Directx12編
2019/02/23

アジェンダ

- 座学（~14:00）
  - 聞きながらプログラムをイジってもOK
- 課題（~16:30）
  - 課題は全部で4セット
    - HelloTriangle
    - HelloTexture
    - HelloModel
    - HelloRenderTarget
- 反省会（~16:55）

Directx12

- 基本的思想はDirectx11と変わらない
- 近年の要求スペックに対してGPUの並列処理に対応
  - そのためAPIが増えた

HelloWorld

- ClearRenderTarget
  ↓
- SwapShain回す

HelloWorld

- d3dx12.h
  - d3d12.hのラッパーファイル
  - マイクロソフト優しい
- d3dcompiler.h
  - shaderとか読み込みに使う

HelloWorld

- Device
  - GPUデバイス
  - 各APIを作成するのに必要
- CommandList
  - GPUに実行させる 命令を積む
- CommandQueue
  - CommandListに積んだ命令を実行する

HelloWorld

- RenderTarget
  - 描画対象
  - 中身はTextureと大差ない

- SwapChain
  - windowに紐づけるダブルバッファ
  - 1window : 1swapchain

HelloWorld

- Resource
  - Texture
    - その名の通りテクスチャ
    - RenderTargetも含まれる
  - Buffer
    - CB,IB,VB...
    - バイトデータ

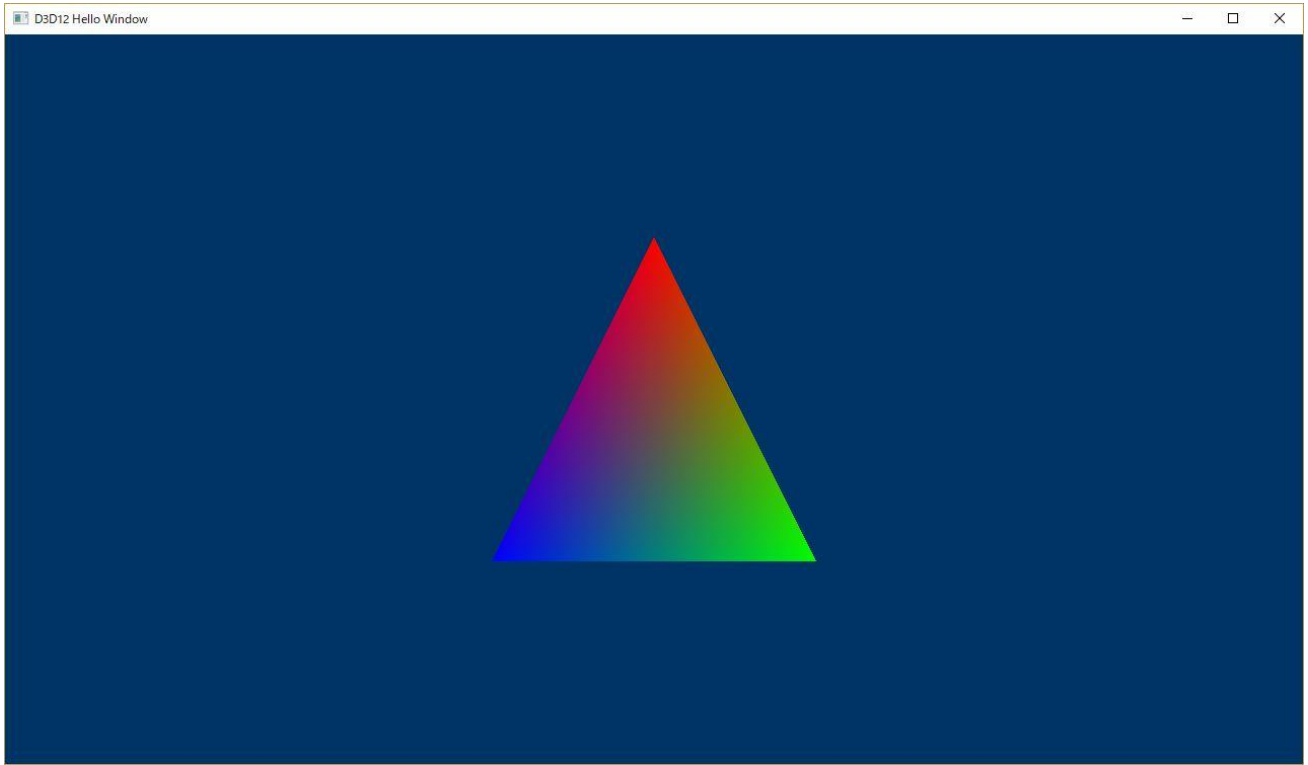
HelloWorld

- ResourceView
  - 各ResourceをGPUに送るときに使用するデータ
  - GPU内でのResourceの種類を判断する必要もある

- 例
  - VertexBuffer...VertexBufferView
  - RenderTarget...RenderTargetView
  - Texture...ShaderResourceView

HelloTriangle

- 課題1



HelloTriangle

- ポリゴン描画の為の要素追加
  - 頂点情報(shaderに渡す)
  - VertexBuffer
  - VertexBufferView
  - RootSignature
  - PipelineState
  - Viewport,Rect
    - 画角
    - コンストラクタで初期化

```
//D3D12HelloWorld.h

struct Vertex
{
        DirectX::XMFLOAT3 position;
        DirectX::XMFLOAT4 color;
};

static constexpr UINT FrameCount = 2;

CD3DX12_VIEWPORT   m_Viewport;
CD3DX12_RECT       m_ScissorRect;

ComPtr<ID3D12RootSignature>     m_RootSignature;
ComPtr<ID3D12PipelineState>     m_PipelineState;

//Resources
ComPtr<ID3D12Resource>          m_VertexBuffer;
D3D12_VERTEX_BUFFER_VIEW        m_VertexBufferView;
```

HelloTriangle

- RootSignature
  - shaderに送るbufferのサイズ・種類を管理
  - 今回は頂点のLayoutだけなので少ない

```
// Create the root signature.

CD3DX12_ROOT_SIGNATURE_DESC rootSignatureDesc = {};
rootSignatureDesc.Init(0, nullptr, 0, nullptr, D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT);

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

D3D12SerializeRootSignature(&rootSignatureDesc, D3D_ROOT_SIGNATURE_VERSION_1, &signature, &error);
m_Device->CreateRootSignature(0, signature->GetBufferPointer(), signature->GetBufferSize(),IID_PPV_ARGS(&m_RootSignature));
```

HelloTriangle

- Shader
  - sampleと同じ
  - .hlslファイルを作る
  - 種類…カスタムビルドツール

```
//shader

struct PSInput
{
	float4 position : SV_POSITION;
	float4 color : COLOR;
};

struct Vertex
{
	float3 position : POSITION;
	float4 color : COLOR;
};

PSInput VSMain(Vertex vertex)
{
	PSInput result;
	result.position = float4(vertex.position, 1.0);
	result.color = vertex.color;

	return result;
}

float4 PSMain(PSInput input) : SV_TARGET
{
	return input.color;
}
```

HelloTriangle

- Shader読み込み
  - 前述したd3dcompiler.hの関数を使う
  - ここでshaderのコンパイルバージョンやエントリポイント確認

```
//Create the shader.

ComPtr<ID3DBlob> vertexShader;
ComPtr<ID3DBlob> pixelShader;
#ifdef _DEBUG
		UINT compileFlags = D3DCOMPILE_DEBUG | D3DCOMPILE_SKIP_OPTIMIZATION;
#else
		UINT compileFlags = 0;
#endif
		ThrowIfFailed(D3DCompileFromFile(L"shaders.hlsl", nullptr, nullptr, "VSMain", "vs_5_0", compileFlags, 0, &vertexShader, nullptr));
		ThrowIfFailed(D3DCompileFromFile(L"shaders.hlsl", nullptr, nullptr, "PSMain", "ps_5_0", compileFlags, 0, &pixelShader, nullptr));
```

HelloTriangle

- PipelineState
  - オブジェクトの描画パラメータ
  - shaderやRenderTargetのFormatもここで設定

- InputLayout
  - 頂点情報

```
// Define the vertex input layout.
D3D12_INPUT_ELEMENT_DESC inputLayout[] =
{
        {"POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0},
        {"COLOR", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 12, D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0}
};

// Create the graphics pipeline state objects (PSO).
D3D12_GRAPHICS_PIPELINE_STATE_DESC psoDesc = {};
psoDesc.InputLayout = {inputLayout, _countof(inputLayout)};
psoDesc.pRootSignature = m_RootSignature.Get();
psoDesc.VS = CD3DX12_SHADER_BYTECODE(vertexShader.Get());
psoDesc.PS = CD3DX12_SHADER_BYTECODE(pixelShader.Get());
psoDesc.RasterizerState = CD3DX12_RASTERIZER_DESC(D3D12_DEFAULT);
psoDesc.BlendState = CD3DX12_BLEND_DESC(D3D12_DEFAULT);
psoDesc.DepthStencilState.DepthEnable = FALSE;
psoDesc.DepthStencilState.StencilEnable = FALSE;
psoDesc.SampleMask = UINT_MAX;
psoDesc.PrimitiveTopologyType = D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE;
psoDesc.NumRenderTargets = 1;
psoDesc.RTVFormats[0] = DXGI_FORMAT_R8G8B8A8_UNORM;
psoDesc.SampleDesc.Count = 1;
m_Device->CreateGraphicsPipelineState(&psoDesc, IID_PPV_ARGS(&m_PipelineState));
```

HelloTriangle

- 描画
  - 今まで作成してきたものをCommandListに積んでいく

CommandListのReset
↓
RootSignatureセット
↓
Viewport,Rectセット
↓
RenderTarget設定
↓
ClearColor
↓
Draw

```
// Set necessary state.
m_CommandList->SetGraphicsRootSignature(m_RootSignature.Get());
m_CommandList->RSSetViewports(1, &m_Viewport);
m_CommandList->RSSetScissorRects(1, &m_ScissorRect);

// Draw call.
m_CommandList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
m_CommandList->IASetVertexBuffers(0, 1, &m_VertexBufferView);
m_CommandList->DrawInstanced(3, 1, 0, 0);
```

HelloTriangle

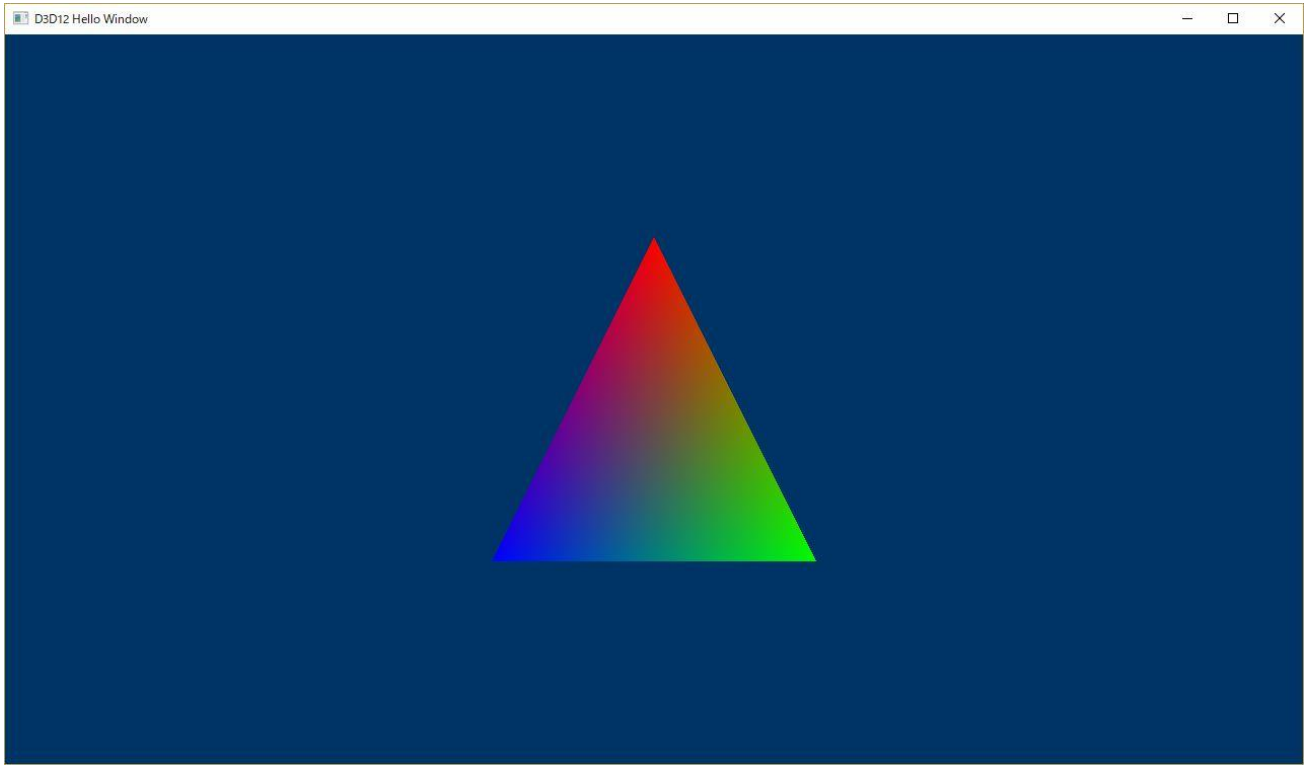
- VertexBuffer
  - 頂点座標,Color,UV等
  - 今回は三角形なので三点
  - ColorはRGBAの4色

- VertexBufferView
  - 前述の通りGPUへ送るための型

```
// Create the vertex buffer.
Vertex triangleVertices[]=
        {
                { { 0.0f, 0.25f * m_aspectRatio, 0.f }, { 1.f, 0.f, 0.f, 1.f } },
                { { 0.25f, -0.25f * m_aspectRatio, 0.f }, { 0.f, 1.f, 0.f, 1.f } },
                { { -0.25f, -0.25f * m_aspectRatio, 0.f }, { 0.f, 0.f, 1.f, 1.f } }
        };
const UINT vertexBufferSize = sizeof(triangleVertices);

m_Device->CreateCommittedResource(
        &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_UPLOAD),
        D3D12_HEAP_FLAG_NONE,
        &CD3DX12_RESOURCE_DESC::Buffer(vertexBufferSize),
        D3D12_RESOURCE_STATE_GENERIC_READ,
        nullptr,
        IID_PPV_ARGS(&m_VertexBuffer));

UINT8* vertexDateBegin;
CD3DX12_RANGE readRange(0, 0);
m_VertexBuffer->Map(0, &readRange, reinterpret_cast<void**>(&vertexDateBegin));
memcpy(vertexDateBegin, triangleVertices, sizeof(triangleVertices));
m_VertexBuffer->Unmap(0, nullptr);

// Initialize the vertex buffer view;
m_VertexBufferView.BufferLocation = m_VertexBuffer->GetGPUVirtualAddress();
m_VertexBufferView.StrideInBytes = sizeof(Vertex);
m_VertexBufferView.SizeInBytes = vertexBufferSize;
```

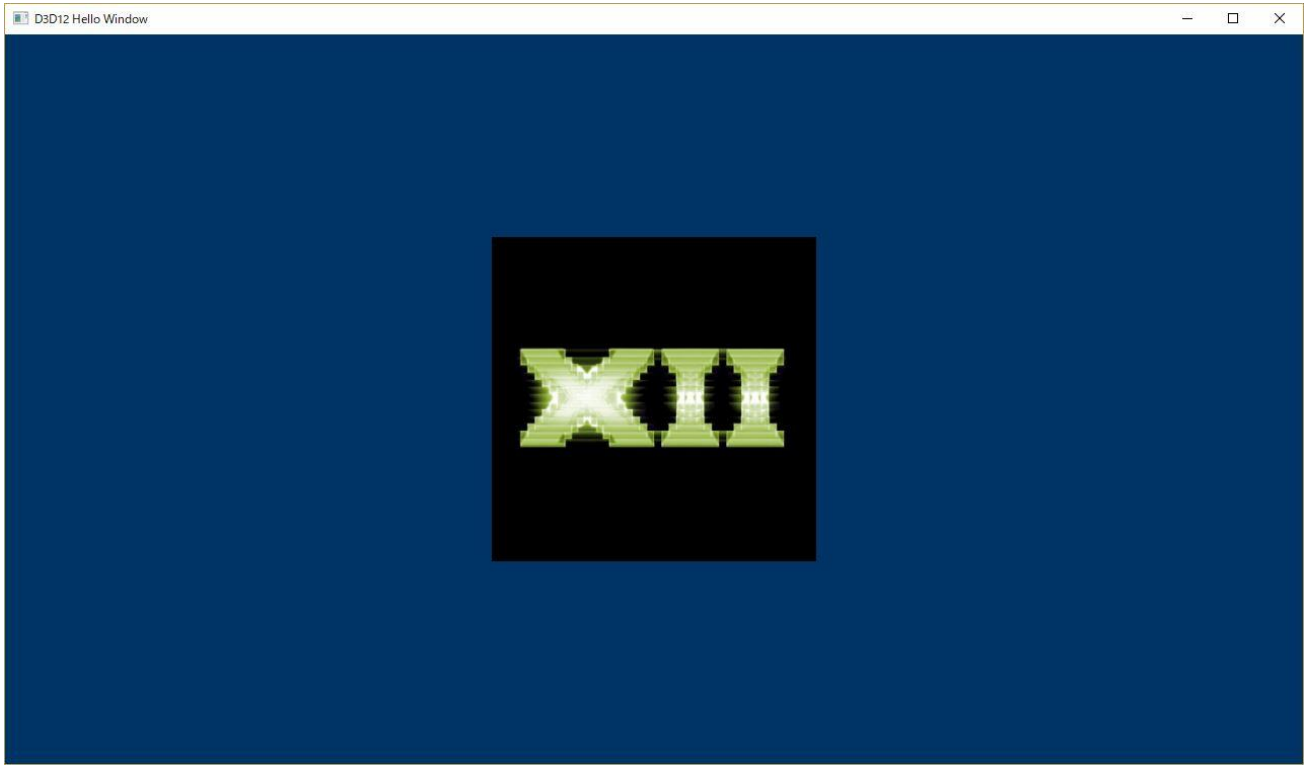
HelloTriangle

- 実行結果



HelloTexture

- 課題2



HelloTexture

- テクスチャファイルのための要素
  - 頂点情報(uvになった)
  - Resource
  - DescriptorHeap
- 4角形のための要素
  - IndexBuffer
  - IndexBufferVIew

```
//D3D12HelloWorld.h

struct Vertex
{
        DirectX::XMFLOAT3 position;
        DirectX::XMFLOAT2 uv;
};

static constexpr UINT TextureWidth = 256;
static constexpr UINT TextureHeight = 256;

ComPtr<ID3D12Resource>          m_IndexBuffer;
D3D12_INDEX_BUFFER_VIEW         m_IndexBufferView;

ComPtr<ID3D12Resource>                  m_Texture;
ComPtr<ID3D12DescriptorHeap>            m_SRVDescriptorHeap;
```

HelloTexture

- テクスチャファイル
  - Directx12には標準のファイル読み込みが無い
  - マイクロソフト標準のDDSLoader使用
  - DDSTextureLoader12.h
  - 画像ファイルは.ddsを使用

HelloTexture

- DescriptorHeap
  - テクスチャ用
  - RenderTargetとは違い1つで良い

D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV

D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE

- IndexBuffer/IndexBufferVIew
  - 基本的にはVertexBufferと作り方は一緒
  - InputLayoutの中身を変更するのを忘れない

HelloTexture

- Sampler
  - テクスチャをGPUで使用する際にコンバータのように使用する
  - RootSignature経由でDescriptorに登録

```
CD3DX12_DESCRIPTOR_RANGE1 ranges[1];
ranges[0].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 1, 0, 0, D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC);

CD3DX12_ROOT_PARAMETER1 rootParameters[1];
rootParameters[0].InitAsDescriptorTable(1, &ranges[0], D3D12_SHADER_VISIBILITY_PIXEL);

D3D12_STATIC_SAMPLER_DESC sampler = {};
sampler.Filter = D3D12_FILTER_MIN_MAG_MIP_POINT;
sampler.AddressU = D3D12_TEXTURE_ADDRESS_MODE_BORDER;
sampler.AddressV = D3D12_TEXTURE_ADDRESS_MODE_BORDER;
sampler.AddressW = D3D12_TEXTURE_ADDRESS_MODE_BORDER;
sampler.MipLODBias = 0;
sampler.MaxAnisotropy = 0;
sampler.ComparisonFunc = D3D12_COMPARISON_FUNC_NEVER;
sampler.BorderColor = D3D12_STATIC_BORDER_COLOR_TRANSPARENT_BLACK;
sampler.MinLOD = 0.f;
sampler.MaxLOD = D3D12_FLOAT32_MAX;
sampler.ShaderRegister = 0;
sampler.RegisterSpace = 0;
sampler.ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;

CD3DX12_VERSIONED_ROOT_SIGNATURE_DESC rootSignatureDesc;
rootSignatureDesc.Init_1_1(_countof(rootParameters), rootParameters, 1, &sampler,
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT);
```

HelloTexture

- Texture
  - Resource作成
    ↓
  - UploadHeapに登録
    ↓
  - CommandListで作成

- ShaderResourceView
  - SRV作成
    ↓
  - CommandListで作成

- 最後にCommandQueueで実行

HelloTexture

```
ComPtr<ID3D12Resource> textureUploadHeap;
// Create the texture.
{
        std::unique_ptr<uint8_t[]> ddsData;
        std::vector<D3D12_SUBRESOURCE_DATA> subresouceData;
        ThrowIfFailed(DirectX::LoadDDSTextureFromFile(m_Device.Get(),
                        L"directx12.DDS",
                        &m_Texture,
                        ddsData,
                        subresouceData));

        D3D12_RESOURCE_DESC textureDesc = m_Texture->GetDesc();
        const UINT subresoucesize = static_cast<UINT>(subresouceData.size());
        const UINT64 uploadBufferSize = GetRequiredIntermediateSize(m_Texture.Get(), 0, subresoucesize);
```

Resource作成

```
        // Create the GPU upload buffer.
        ThrowIfFailed(m_Device->CreateCommittedResource(
                        &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_UPLOAD),
                        D3D12_HEAP_FLAG_NONE,
                        &CD3DX12_RESOURCE_DESC::Buffer(uploadBufferSize),
                        D3D12_RESOURCE_STATE_GENERIC_READ,
                        nullptr,
                        IID_PPV_ARGS(&textureUploadHeap)));
        UpdateSubresources(m_CommandList.Get(), m_Texture.Get(), textureUploadHeap.Get(), 0, 0, subresoucesize, &subresouceData[0]);
        m_CommandList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::Transition(m_Texture.Get(), D3D12_RESOURCE_STATE_COPY_DEST, D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE));
```

UploadHeap登録

```
        // Describe and create a SRV for the texture.
        D3D12_SHADER_RESOURCE_VIEW_DESC srvDesc = {};
        srvDesc.Shader4ComponentMapping = D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING;
        srvDesc.Format = textureDesc.Format;
        srvDesc.ViewDimension = D3D12_SRV_DIMENSION_TEXTURE2D;
        srvDesc.Texture2D.MipLevels = subresoucesize;
        m_Device->CreateShaderResourceView(m_Texture.Get(), &srvDesc, m_SRVDescriptorHeap->GetCPUDescriptorHandleForHeapStart());
        m_CommandList->DiscardResource(textureUploadHeap.Get(), nullptr);
}
```

SRV作成

HelloTexture

- Shader
  - 頂点情報がUVに変わった
  - Sampler経由でTextureの情報を取り出す

```
struct PSInput
{
	float4 position : SV_POSITION;
	float2 uv : TEXCOORD;
};

struct Vertex
{
	float3 position : POSITION;
	float2 uv : TEXCOORD;
};

Texture2D g_texture : register(t0);
SamplerState g_sampler : register(s0);

PSInput VSMain(Vertex vertex)
{
	PSInput result;
	result.position = float4(vertex.position, 1.0);
	result.uv = vertex.uv;

	return result;
}

float4 PSMain(PSInput input) : SV_TARGET
{
	return g_texture.Sample(g_sampler, input.uv);
}
```

HelloTriangle

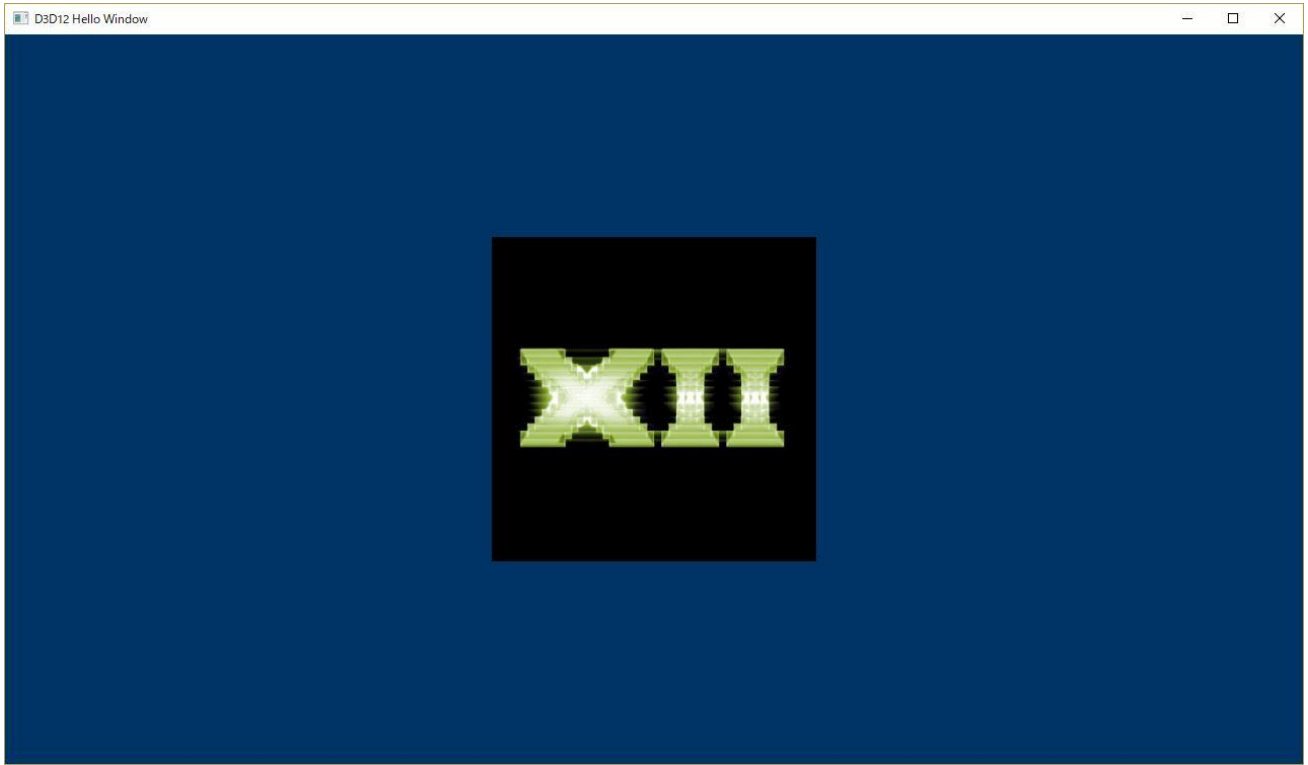
- 描画
  - 今まで作成してきたものをCommandListに積んでいく
  - 今回追加したものを忘れずに

CommandListのReset
↓
RootSignatureセット
↓
SRVDescriptorHeapセット
↓
Viewport,Rectセット
↓
RenderTarget設定
↓
ClearColor
↓
Draw

```
ID3D12DescriptorHeap* heaps[] = {m_SRVDescriptorHeap.Get()};
m_CommandList->SetDescriptorHeaps(_countof(heaps), heaps);
```

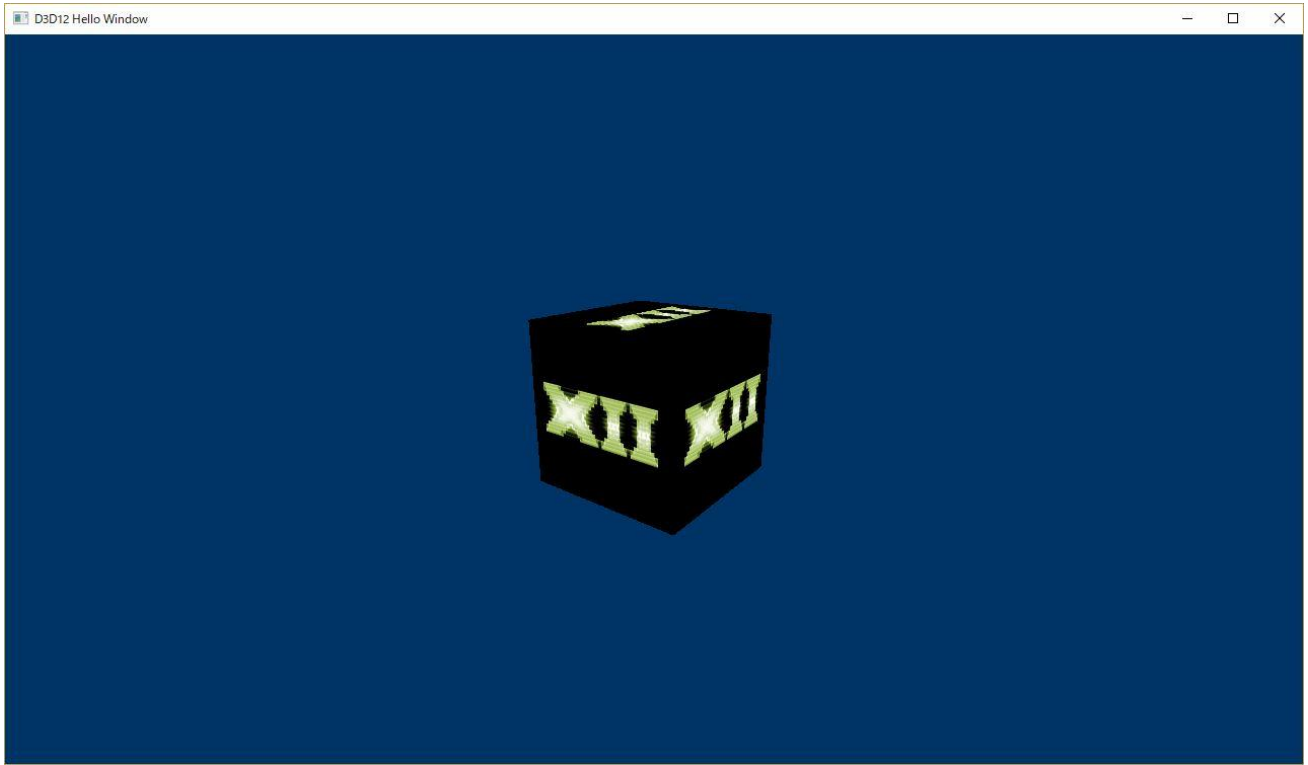
HelloTexture

- 実行結果



HelloCube

- 課題3



HelloCube

- 3Dオブジェクトの為の要素
  - 深度バッファ適用
  - 行列

- Cubeの為の要素
  - 頂点情報追加
  - 定数バッファ追加

```
//
struct Matrix
{
        DirectX::XMFLOAT4X4 Model;
        DirectX::XMFLOAT4X4 View;
        DirectX::XMFLOAT4X4 Projection;
};

ComPtr<ID3D12Resource>          m_ConstantBuffer;
UINT        m_ConstantBufferSize;
UINT8*      m_ConstantBufferBegin;
Matrix      m_CBMatrix;

ComPtr<ID3D12DescriptorHeap>    m_SRVCBVDescriptorHeap;
UINT                            m_SRVCBVDescriptorSize;

float                           m_Angle;
```

HelloCube

- DepthStencilState
  - 深度バッファ
  - 3d空間での奥行きとかで使う
  - モデル等を使用する際に

```
CD3DX12_DEPTH_STENCIL_DESC(D3D12_DEFAULT)
```

HelloCube

- 実行結果