DirectX 12

Why DirectX?
- Limited platforms
  - Windows 10
  - Xbox One
- Large, connected API
  - Direct3D
  - DirectXMath
  - XInput
  - XAudio2
- Good for games on Microsoft platforms

Setting Up for DX12 Development
- Be on Windows 10
  - Enable developer mode in Update and Security Settings
- Download Visual Studio 2017
  - Install selecting the Universal Windows Platform and Game Development with C++ workloads
- Start Visual Studio
- Create new project
  - Under C++ select Windows Universal
  - Select DirectX 12 App
Initialization

Pipeline Overview

Debug Layer

```cpp
#ifdef _DEBUG
// Enable the D3D12 debug layer.
{
    ComPtr<ID3D12Debug> debugController;
    if (SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(&debugController))))
    {
        debugController->EnableDebugLayer();
    }
} #endif
```
Create Device

ComPtr<IDXGIFactory4> factory;
ThrowIfFailed(CreateDXGIFactory1(IID_PPV_ARGS(&factory)));  
ComPtr<IDXGIAdapter1> hardwareAdapter;
GetHardwareAdapter(factory.Get(), &hardwareAdapter);
ThrowIfFailed(D3D12CreateDevice(
 hardwareAdapter.Get(),
 D3D_FEATURE_LEVEL_11_0,
 IID_PPV_ARGS(&m_device)
));

Command Queue

D3D12_COMMAND_QUEUE_DESC queueDesc = {};  
queueDesc.Flags = D3D12_COMMAND_QUEUE_FLAG_NONE;
queueDesc.Type = D3D12_COMMAND_LIST_TYPE_DIRECT;
ThrowIfFailed(m_device->CreateCommandQueue(&queueDesc,
IID_PPV_ARGS(&m_commandQueue)));

Swap Chain Description

DXGI_SWAP_CHAIN_DESC swapChainDesc = {
 swapChainDesc.BufferCount = FrameCount;
 swapChainDesc.BufferDesc.Width = m_width;
 swapChainDesc.BufferDesc.Height = m_height;
 swapChainDesc.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
 swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
 swapChainDesc.SwapEffect = DXGI_SWAP_EFFECT_FLIP_DISCARD;
 swapChainDesc.OutputWindow = Win32Application::GetHwnd();
 swapChainDesc.SampleDesc.Count = 1;
 swapChainDesc.Windowed = TRUE;
}

Swap Chain Creation

ComPtr<IDXGISwapChain> swapChain;
ThrowIfFailed(factory->CreateSwapChain(
 m_commandQueue.Get(),
 &swapChainDesc,
 &swapChain
));
ThrowIfFailed(swapChain.As(&m_swapChain));
Create Descriptor Heap (RTVs)

```c
D3D12_DESCRIPTOR_HEAP_DESC rtvHeapDesc = {};
rtvHeapDesc.NumDescriptors = FrameCount;
rtvHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_RTV;
rtvHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_NONE;
ThrowIfFailed(m_device->CreateDescriptorHeap(&rtvHeapDesc, IID_PPV_ARGS(&m_rtvHeap)));

m_rtvDescriptorSize = m_device->GetDescriptorHandleIncrementSize(D3D12_DESCRIPTOR_HEAP_TYPE_RTV);
```

Create RTVs for Each Frame

```c
CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap->GetCPUDescHandleForHeapStart());
for (UINT n = 0; n < FrameCount; n++)
{
    ThrowIfFailed(m_swapChain->GetBuffer(n, IID_PPV_ARGS(&m_renderTargets[n])));
    m_device->CreateRenderTargetView(m_renderTargets[n].Get(), nullptr, rtvHandle);
    rtvHandle.Offset(1, m_rtvDescriptorSize);
}
```

Setup for Pipeline

- Create Root Signature Description
  - Bind Root Signature to device
    - m_device -> CreateRootSignature
  - Compile and Load Shaders
    - D3DCompileFromFile function
  - Define Input Element Description array
    - Tells the pipeline how vertices will be defined

Pipeline Creation

```c
D3D12_GRAPHICS_PIPELINE_STATE_DESC psoDesc = {};
psoDesc.InputLayout = { inputElementDescs, _countof(inputElementDescs) };
psoDesc.pRootSignature = m_rootSignature.Get();
psoDesc.VS = { reinterpret_cast<UINT8*>(vertexShader->GetBufferPointer()), vertexShader->GetBufferSize() };
psoDesc.PS = { reinterpret_cast<UINT8*>(pixelShader->GetBufferPointer()), pixelShader->GetBufferSize() };
psoDesc.RasterizerState = CD3DX12_RASTERIZER_DESC(D3D12_DEFAULT);
psoDesc.BlendState = CD3DX12_BLEND_DESC(D3D12_DEFAULT);
psoDesc.DepthStencilState.DepthEnable = FALSE;
psoDesc.DepthStencilState.StencilEnable = FALSE;
psoDesc.DepthStencilState.StencilWriteMask = 0;
psoDesc.PrimitiveTopologyType = D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE;
psoDesc.NumRenderTargets = 1;
psoDesc.RTVFormats[0] = DXGI_FORMAT_R8G8B8A8_UNORM;
psoDesc.SampleDesc.Count = 1;
ThrowIfFailed(m_device->CreateGraphicsPipelineState(&psoDesc, IID_PPV_ARGS(&m_pipelineState)));
```
Command List

● Create command list for use when drawing

ThrowIfFailed(m_device->CreateCommandList(0, D3D12_COMMAND_LIST_TYPE_DIRECT, m_commandAllocator.Get(), m_pipelineState.Get(), IID_PPV_ARGS(&m_commandList)));
ThrowIfFailed(m_commandList->Close());

● Command List must be closed before continuing initialization

Create Vertex Buffer

● Create array of structs for geometry

Create Resource on Device

ThrowIfFailed(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)));

● Do not use UPLOAD for static data

Vertex Buffer (cont.)

● Copy vertices to the buffer

UINT8* pVertexDataBegin;
CD3DX12_RANGE readRange(0, 0); // We do not intend to read from this resource on the CPU.
ThrowIfFailed(m_vertexBuffer->Map(0, &readRange, reinterpret_cast<void**>(&pVertexDataBegin)));
memcpy(pVertexDataBegin, objectVertices, sizeof(objectVertices));
ThrowIfFailed(m_vertexBuffer->Unmap(0, nullptr));

● Initialize Vertex Buffer View

m_vertexBufferView.BufferLocation = m_vertexBuffer->GetGPUVirtualAddress();
m_vertexBufferView.StrideInBytes = sizeof(Vertex);
m_vertexBufferView.SizeInBytes = vertexBufferSize;

Fence

● Create a fence

Wait for fence before ending initialization

○ Ensures all setup is complete
Rendering

Populate Command List
- Enter commands into the command list
  - m_commandList -> SomeCommand
- Some commands include:
  - SetGraphicsRootSignature
  - RSSetViewport
  - RSSetScissorRects
  - IASetPrimitiveTopologies
  - IASetVertexBuffer
  - DrawInstanced
- Close command list after completion

Execute Command List

Present Swap Chain
- Wait for fence
Closing the App

- Wait for fence
  - Make sure the gpu is done with all commands
- Destroy the fence
  - CloseHandle(m_fenceEvent);