# 初始化和渲染管线

# 初始化

**1. 创建ID3D11Device和ID3D11DeviceContext**

关键API：

**D3D11CreateDevice**

|  |
| --- |
| void InitGame::CreateDevice()  {  // Create device and context  UINT createDeviceFlag = 0;  #ifdef \_DEBUG  // Check for SDK Layer support.  HRESULT hrDebug = D3D11CreateDevice(  nullptr,  D3D\_DRIVER\_TYPE\_NULL, // There is no need to create a real hardware device.  0,  D3D11\_CREATE\_DEVICE\_DEBUG, // Check for the SDK layers.  nullptr, // Any feature level will do.  0,  D3D11\_SDK\_VERSION, // Always set this to D3D11\_SDK\_VERSION for Windows Store apps.  nullptr, // No need to keep the D3D device reference.  nullptr, // No need to know the feature level.  nullptr // No need to keep the D3D device context reference.  );  bool bSdkLayersAvailable = SUCCEEDED(hrDebug);  if (bSdkLayersAvailable)  {  // If the project is in a debug build, enable debugging via SDK Layers with this flag.  createDeviceFlag |= D3D11\_CREATE\_DEVICE\_DEBUG;  }  #endif    HRESULT hr = D3D11CreateDevice(  nullptr,  D3D\_DRIVER\_TYPE\_HARDWARE,  nullptr,  createDeviceFlag,  nullptr,  0,  D3D11\_SDK\_VERSION,  m\_d3dDevice.GetAddressOf(),  &m\_featureLevel,  m\_d3dContext.GetAddressOf()  );  DX::ThrowIfFailed(hr);  } |

**2.创建IDXGISwapChain**

Note:当窗口大小改变之后，需要ResizeBuffers

关键API：

获得IDXGIFactory：m\_d3dDevice->QueryInterface，dxgiDevice->GetParent，dxgiAdapter->GetParent

|  |
| --- |
| ComPtr<IDXGIDevice> dxgiDevice;  m\_d3dDevice->QueryInterface(\_\_uuidof(IDXGIDevice), &dxgiDevice);  ComPtr<IDXGIAdapter> dxgiAdapter;  dxgiDevice->GetParent(\_\_uuidof(IDXGIAdapter), &dxgiAdapter);  ComPtr<IDXGIFactory> dxgiFactory;  dxgiAdapter->GetParent(\_\_uuidof(IDXGIFactory), &dxgiFactory); |

dxgiFactory->**CreateSwapChain**

|  |
| --- |
| DXGI\_SWAP\_CHAIN\_DESC scDesc;  scDesc.BufferDesc.Width = m\_outputWidth;  scDesc.BufferDesc.Height = m\_outputHeight;  scDesc.BufferDesc.RefreshRate.Numerator = 60;  scDesc.BufferDesc.RefreshRate.Denominator = 1;  scDesc.BufferDesc.Format = backBufferFormat;  scDesc.BufferDesc.ScanlineOrdering = DXGI\_MODE\_SCANLINE\_ORDER\_UNSPECIFIED;  scDesc.BufferDesc.Scaling = DXGI\_MODE\_SCALING\_UNSPECIFIED;  if (bEnable4xMsaa)  {  scDesc.SampleDesc.Count = 4;  scDesc.SampleDesc.Quality = m4xMsaaQuality - 1;  }  else  {  scDesc.SampleDesc.Count = 1;  scDesc.SampleDesc.Quality = 0;  }  scDesc.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;  scDesc.BufferCount = backBufferCount;  scDesc.OutputWindow = m\_window;  scDesc.Windowed = true;  scDesc.SwapEffect = DXGI\_SWAP\_EFFECT\_DISCARD;  scDesc.Flags = 0;    hr = dxgiFactory->CreateSwapChain(  m\_d3dDevice.Get(),  &scDesc,  m\_swapChain.GetAddressOf()  );  DX::ThrowIfFailed(hr); |

**3.创建ID3D11RenderTargetView和ID3D11DepthStencilView，为设置RenderTarget做准备**

关键API：

**CreateRenderTargetView**

|  |
| --- |
| // Create render targer view  ComPtr<ID3D11Resource> backBuffer;  m\_swapChain->GetBuffer(0, IID\_PPV\_ARGS(&backBuffer));  hr = m\_d3dDevice->CreateRenderTargetView(backBuffer.Get(), 0, m\_renderTargetView.GetAddressOf());  DX::ThrowIfFailed(hr); |

**CreateDepthStencilView**

|  |
| --- |
| // Create depth stencil view  D3D11\_TEXTURE2D\_DESC depthStencilDesc;  depthStencilDesc.Width = m\_outputWidth;  depthStencilDesc.Height = m\_outputHeight;  depthStencilDesc.MipLevels = 1;  depthStencilDesc.ArraySize = 1;  depthStencilDesc.Format = DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT;  if (bEnable4xMsaa)  {  depthStencilDesc.SampleDesc.Count = 4;  depthStencilDesc.SampleDesc.Quality = m4xMsaaQuality - 1;  }  else  {  depthStencilDesc.SampleDesc.Count = 1;  depthStencilDesc.SampleDesc.Quality = 0;  }  depthStencilDesc.Usage = D3D11\_USAGE\_DEFAULT;  depthStencilDesc.BindFlags = D3D11\_BIND\_DEPTH\_STENCIL;  depthStencilDesc.CPUAccessFlags = 0;  depthStencilDesc.MiscFlags = 0;    ComPtr<ID3D11Texture2D> mDepthStencilBuffer;  hr = m\_d3dDevice->CreateTexture2D(&depthStencilDesc, nullptr, mDepthStencilBuffer.GetAddressOf());  DX::ThrowIfFailed(hr);    hr = m\_d3dDevice->CreateDepthStencilView(mDepthStencilBuffer.Get(), nullptr, m\_depthStencilView.GetAddressOf());  DX::ThrowIfFailed(hr); |

**OMSetRenderTargets**

|  |
| --- |
| // Set render target  m\_d3dContext->OMSetRenderTargets(1, m\_renderTargetView.GetAddressOf(), m\_depthStencilView.Get()); |

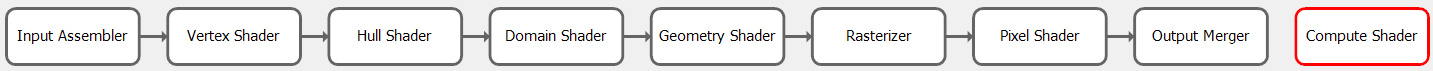
**4.设置Viewport**

关键API：

**RSSetViewports**

|  |
| --- |
| // Set viewport  D3D11\_VIEWPORT viewport;  viewport.TopLeftX = 0;  viewport.TopLeftY = 0;  viewport.Width = static\_cast<float>(m\_outputWidth);  viewport.Height = static\_cast<float>(m\_outputHeight);  viewport.MinDepth = 0.f;  viewport.MaxDepth = 1.f;    m\_d3dContext->RSSetViewports(1, &viewport); |

# 渲染管线



**0.清除之前的状态**

|  |
| --- |
| void InitGame::Clear()  {  // Clear the views.  m\_d3dContext->ClearRenderTargetView(m\_renderTargetView.Get(), Colors::CornflowerBlue);  m\_d3dContext->ClearDepthStencilView(m\_depthStencilView.Get(), D3D11\_CLEAR\_DEPTH | D3D11\_CLEAR\_STENCIL, 1.0f, 0);    m\_d3dContext->OMSetRenderTargets(1, m\_renderTargetView.GetAddressOf(), m\_depthStencilView.Get());    // Set the viewport.  CD3D11\_VIEWPORT viewport(0.0f, 0.0f, static\_cast<float>(m\_outputWidth), static\_cast<float>(m\_outputHeight));  m\_d3dContext->RSSetViewports(1, &viewport);  } |

**1.Input Assembler stage：构建几何体**

关键API：

**D3DCompileFromFile/D3DX11CompileFromFile，**或是借助VS自己的shader编译功能提前编译好shader，然后直接读取cso文件

**CreateVertexShader**

**CreatePixelShader**

|  |
| --- |
| ComPtr<ID3DBlob> mvsByteCode = d3dUtil::CompileShader(L"Shaders\\color.hlsl", nullptr, "VS", "vs\_5\_0");  ComPtr<ID3DBlob> mpsByteCode = d3dUtil::CompileShader(L"Shaders\\color.hlsl", nullptr, "PS", "ps\_5\_0");    m\_d3dDevice->CreateVertexShader(mvsByteCode->GetBufferPointer(), mvsByteCode->GetBufferSize(), nullptr, m\_vertexShader.GetAddressOf());  m\_d3dDevice->CreatePixelShader(mpsByteCode->GetBufferPointer(), mpsByteCode->GetBufferSize(), nullptr, m\_pixelShader.GetAddressOf()); |

**CreateInputLayout**

|  |
| --- |
| D3D11\_INPUT\_ELEMENT\_DESC vertexDesc[] =  {  {"POSITION",0,DXGI\_FORMAT\_R32G32B32\_FLOAT,0,0,D3D11\_INPUT\_PER\_VERTEX\_DATA,0},  {"COLOR",0,DXGI\_FORMAT\_R32G32B32A32\_FLOAT,0,12,D3D11\_INPUT\_PER\_VERTEX\_DATA,0}  };    HRESULT hr = m\_d3dDevice->CreateInputLayout(  vertexDesc,  ARRAYSIZE(vertexDesc),  mvsByteCode->GetBufferPointer(),  mvsByteCode->GetBufferSize(),  m\_inputLayout.GetAddressOf()  );  DX::ThrowIfFailed(hr); |

**IASetVertexBuffers**

|  |
| --- |
| // Set vertex buffer  Vertex vertices[] =  {  { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT4(Colors::White) },  { XMFLOAT3(-1.0f, +1.0f, -1.0f), XMFLOAT4(Colors::Black) },  { XMFLOAT3(+1.0f, +1.0f, -1.0f), XMFLOAT4(Colors::Red) },  { XMFLOAT3(+1.0f, -1.0f, -1.0f), XMFLOAT4(Colors::Green) },  { XMFLOAT3(-1.0f, -1.0f, +1.0f), XMFLOAT4(Colors::Blue) },  { XMFLOAT3(-1.0f, +1.0f, +1.0f), XMFLOAT4(Colors::Yellow) },  { XMFLOAT3(+1.0f, +1.0f, +1.0f), XMFLOAT4(Colors::Cyan) },  { XMFLOAT3(+1.0f, -1.0f, +1.0f), XMFLOAT4(Colors::Magenta) }  };    D3D11\_BUFFER\_DESC vbDesc;  vbDesc.ByteWidth = sizeof(vertices);  vbDesc.Usage = D3D11\_USAGE\_IMMUTABLE;  vbDesc.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER;  vbDesc.CPUAccessFlags = 0;  vbDesc.MiscFlags = 0;  vbDesc.StructureByteStride = 0;    D3D11\_SUBRESOURCE\_DATA vbInitData;  vbInitData.pSysMem = vertices;  vbInitData.SysMemPitch = 0;  vbInitData.SysMemSlicePitch = 0;    HRESULT hr = m\_d3dDevice->CreateBuffer(&vbDesc, &vbInitData, m\_vertexBuffer.GetAddressOf());  DX::ThrowIfFailed(hr);    UINT stride = sizeof(Vertex);  UINT offset = 0;  m\_d3dContext->IASetVertexBuffers(0, 1, m\_vertexBuffer.GetAddressOf(), &stride, &offset); |

**IASetIndexBuffer**

|  |
| --- |
| // Set index buffer  UINT indices[] =  {  // front face  0, 1, 2,  0, 2, 3,    // back face  4, 6, 5,  4, 7, 6,    // left face  4, 5, 1,  4, 1, 0,    // right face  3, 2, 6,  3, 6, 7,    // top face  1, 5, 6,  1, 6, 2,    // bottom face  4, 0, 3,  4, 3, 7  };    m\_indexCount = ARRAYSIZE(indices);    D3D11\_BUFFER\_DESC ibDesc;  ibDesc.ByteWidth = sizeof(indices);  ibDesc.Usage = D3D11\_USAGE\_IMMUTABLE;  ibDesc.BindFlags = D3D11\_BIND\_INDEX\_BUFFER;  ibDesc.CPUAccessFlags = 0;  ibDesc.MiscFlags = 0;  ibDesc.StructureByteStride = 0;    D3D11\_SUBRESOURCE\_DATA ibInitData;  ibInitData.pSysMem = indices;  ibInitData.SysMemPitch = 0;  ibInitData.SysMemSlicePitch = 0;    hr = m\_d3dDevice->CreateBuffer(&ibDesc, &ibInitData, m\_indexBuffer.GetAddressOf());  DX::ThrowIfFailed(hr);    m\_d3dContext->IASetIndexBuffer(m\_indexBuffer.Get(), DXGI\_FORMAT\_R32\_UINT, 0); |

**IASetPrimitiveTopology**

|  |
| --- |
| m\_d3dContext->IASetPrimitiveTopology(D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST); |

**2.Vertex Shader stage：处理顶点数据，变换顶点坐标到齐次裁剪空间(homogeneous clip space)**

关键API：

**VSSetConstantBuffers**

|  |
| --- |
| // Set constant buffer  D3D11\_BUFFER\_DESC cbDesc;  cbDesc.ByteWidth = sizeof(XMFLOAT4X4);  cbDesc.Usage = D3D11\_USAGE\_DEFAULT;  cbDesc.BindFlags = D3D11\_BIND\_CONSTANT\_BUFFER;  cbDesc.CPUAccessFlags = 0;  cbDesc.MiscFlags = 0;  cbDesc.StructureByteStride = 0;    hr = m\_d3dDevice->CreateBuffer(&cbDesc, nullptr, m\_constantBuffer.GetAddressOf());    m\_d3dContext->VSSetConstantBuffers(0, 1, m\_constantBuffer.GetAddressOf()); |

注意，如果Usage设置为了D3D11\_USAGE\_DYNAMIC，则需要用**ID3D11DeviceContext::Map**和**ID3D11DeviceContext::Unmap**来更新Buffer，而不是UpdateSubresource 。

<https://docs.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-id3d11devicecontext-updatesubresource>

**UpdateSubresource**

|  |
| --- |
| void BoxGame::Update(DX::StepTimer const& timer)  {  // Rotate the model  XMMATRIX world = XMLoadFloat4x4(&m\_world);  XMMATRIX view = XMLoadFloat4x4(&m\_view);  XMMATRIX proj = XMLoadFloat4x4(&m\_proj);    float radiansPerSecond = XMConvertToRadians(45);  double totalRotation = timer.GetTotalSeconds() \* radiansPerSecond;  float radians = static\_cast<float>(fmod(totalRotation, XM\_2PI));  world = XMMatrixRotationY(radians);    XMMATRIX mWorldViewProj = XMMatrixMultiply(XMMatrixMultiply(world, view), proj);  XMFLOAT4X4 cbWorldViewProj;  XMStoreFloat4x4(&cbWorldViewProj, XMMatrixTranspose(mWorldViewProj)); // Use XMMatrixTranspose before send to GPU    m\_d3dContext->UpdateSubresource(m\_constantBuffer.Get(), 0, nullptr, &cbWorldViewProj, 0, 0);  } |

注意，C++与HLSL处理矩阵的格式不同，C++为row-major，而HLSL为column-major。即对于x[0][0]之后的元素，C++是x[0][1]，HLSL是x[1][0]。  
所以在将矩阵从CPU传给GPU时需要转置(调用XMMatrixTranspose)。

<https://docs.microsoft.com/zh-cn/windows/win32/direct3dhlsl/dx-graphics-hlsl-per-component-math?redirectedfrom=MSDN#matrix-ordering>

**VSSetShader**

shader示例：

|  |
| --- |
| VertexOut VS(VertexIn vin)  {  VertexOut vout;    // Transform to homogeneous clip space.  vout.PosH = mul(float4(vin.PosL, 1.0f), gWorldViewProj);    // Just pass vertex color into the pixel shader.  vout.Color = vin.Color;    return vout;  } |

**3.Rasterization stage：由硬件执行光栅化，做背面剔除(Backface Culling)，对顶点属性进行插值**

关键API：

**RSSetState**

|  |
| --- |
| bool bShowWireframe = false;  if (bShowWireframe)  {  CD3D11\_RASTERIZER\_DESC rsDesc(D3D11\_DEFAULT);  rsDesc.FillMode = D3D11\_FILL\_WIREFRAME;  rsDesc.CullMode = D3D11\_CULL\_NONE;  ComPtr<ID3D11RasterizerState> mRSState;  m\_d3dDevice->CreateRasterizerState(&rsDesc, mRSState.GetAddressOf());  m\_d3dContext->RSSetState(mRSState.Get());  } |

**4.Pixel Shader stage：逐像素进行颜色处理，例如计算光照后的颜色**

关键API：

**PSSetShader**

|  |
| --- |
| m\_d3dContext->VSSetShader(m\_vertexShader.Get(), nullptr, 0);  m\_d3dContext->PSSetShader(m\_pixelShader.Get(), nullptr, 0); |

shader示例：

|  |
| --- |
| float4 PS(VertexOut pin) : SV\_Target  {  return pin.Color;  } |

**5.Output Merger stage：通过混合像素着色器输出的颜色和后台缓冲区中的颜色，实现透明等效果**

关键API：

**OMSetRenderTargets**

**OMSetBlendState**

**OMSetDepthStencilState**

**6.Drawcall**

关键API：

**DrawIndexed**

|  |
| --- |
| m\_d3dContext->DrawIndexed(m\_indexCount, 0, 0); |

**7.Present**

|  |
| --- |
| void InitGame::Present()  {  // The first argument instructs DXGI to block until VSync, putting the application  // to sleep until the next VSync. This ensures we don't waste any cycles rendering  // frames that will never be displayed to the screen.  HRESULT hr = m\_swapChain->Present(1, 0);    // If the device was reset we must completely reinitialize the renderer.  if (hr == DXGI\_ERROR\_DEVICE\_REMOVED || hr == DXGI\_ERROR\_DEVICE\_RESET)  {  OnDeviceLost();  }  else  {  DX::ThrowIfFailed(hr);  }  } |