**创建设备和设备上下文**

ID3DDevice 用于检查支持的特性, 和分配资源

ID3DDeviceContext 用于设置渲染状态, 将资源绑定到图形管线, 和发布渲染命令

HRESULT D3D11CreateDevice(

IDXGIAdapter \*pAdapter,

D3D\_DRIVER\_TYPE DriverType,

HMODULE Software,

UINT Flags,

CONST D3D\_FEATURE\_LEVEL \*pFeatureLevels,

UINT FeatureLevels,

UINT SDKVersion,

ID3D11Device \*\*ppDevice,

D3D\_FEATURE\_LEVEL \*pFeatureLevel,

ID3D11DeviceContext \*\*ppImmediateContext

);

pAdapter

指定使用的渲染适配器. 传入 NULL 表示使用主适配器

DriverType

驱动类型. 一般都用 D3D\_DRIVER\_TYPE\_HARDWARE

其他可用的类型有

D3D\_DRIVER\_TYPE\_REFERENCE

D3D\_DRIVER\_TYPE\_SOFTWARE

Software

用于指定驱动软件, 如果使用硬件渲染的话则传入 NULL

Flags

可选的设备创建标识, 常用的有

D3D11\_CREATE\_DEVICE\_DEBUG 调试模式, 这时 D3D 会输出发送信息到 VS 的输出窗口

D3D11\_CREATE\_DEVICE\_SINGLETHREADED 只用单线程, 如果不使用多线程的话用这个可以提高性能, 这时调用 ID3D11Device::CreateDeferredContext 方法会失败

pFeatureLevels

一个 D3D\_FEATURE\_LEVEL 枚举元素的数组, 指定用于特性检查时的顺序.

如果使用默认的 ( 从高到低 ), 传入 NULL

FeatureLevels

上面那个数组的元素个数, 如果是 NULL, 传入 0

SDKVersion

永远用 D3D11\_SDK\_VERSION

ppDevice

返回创建的设备

pFeatureLevel

返回支持的特性等级

ppImmediateContext

返回创建的设备上下文

UINT createDeviceFlags = 0;

#if defined(DEBUG) || defined(\_DEBUG)

createDeviceFlags |= D3D11\_CREATE\_DEVICE\_DEBUG;

#endif

D3D\_FEATURE\_LEVEL featureLevel;

ID3D11Device\* d3dDevice;

ID3D11DeviceContext\* d3dImmediateContext;

HRESULT hr = D3D11CreateDevice(

NULL,

D3D\_DRIVER\_TYPE\_HARDWARE,

NULL,

createDeviceFlags,

0,

0,

D3D11\_SDK\_VERSION,

&d3dDevice,

&featureLevel,

&d3dImmediateContext

);

if( FAILED(hr) )

{

MessageBox(0, "D3D11CreateDevice Failed.", 0, 0);

return false;

}

if( featureLevel != D3D\_FEATURE\_LEVEL\_11\_0 )

{

MessageBox(0, "Direct3D Feature Level 11 unsupported.", 0, 0);

return false;

}

**检查 4X MSAA 质量等级支持**

所有的 DirectX 11 设备都对任何渲染对象支持 4X MSAA, 但支持的质量等级可能不同

UINT 4xMsaaQuality;

HR ( d3dDevice->CheckMultisampleQualityLevels (

DXGI\_FORMAT\_R8G8B8A8\_UNORM,

4,

&4xMsaaQuality ));

assert ( 4xMsaaQuality > 0 );

可用的质量等级为 0 到 4xMsaaQuality - 1

**描述交换链**

通过填充一个 DXGI\_SWAP\_CHAIN 结构

typedef struct DXGI\_SWAP\_CHAIN\_DESC

{

DXGI\_MODE\_DESC BufferDesc;

DXGI\_SAMPLE\_DESC SampleDesc;

DXGI\_USAGE BufferUsage;

UINT BufferCount;

HWND OutputWindow;

BOOL Windowed;

DXGI\_SWAP\_EFFECT SwapEffect;

UINT Flags;

}

DXGI\_SWAP\_CHAIN\_DESC;

BufferDesc

描述后缓存区的结构

SampleDesc

描述多重缓存和质量等级的结构

BufferUsage

后缓存的用处. 指定为 DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT, 因为我们将渲染到后缓存区

BufferCount

后缓存的个数. 一般一个就够了

OutputWindow

指向输出窗口的句柄

Windowed

是否以窗口模式运行

SwapEffect

指定 DXGI\_SWAP\_EFFECT\_DISCARD

Flags

可选的标识

DXGI\_SWAP\_CHAIN\_DESC swapChainDesc;

swapChainDesc.BufferDesc.Width = clientWidth;

swapChainDesc.BufferDesc.Height = clientHeight;

swapChainDesc.BufferDesc.RefreshRate.Numerator = 60;

swapChainDesc.BufferDesc.RefreshRate.Denominator = 1;

swapChainDesc.BufferDesc.Format = DXGI\_FORMAT\_R8G8B8A8\_UNORM;

swapChainDesc.BufferDesc.ScanlineOrdering = DXGI\_MODE\_SCANLINE\_ORDER\_UNSPECIFIED;

swapChainDesc.BufferDesc.Scaling = DXGI\_MODE\_SCALING\_UNSPECIFIED;

if( enable4xMsaa == true )

{

swapChainDesc.SampleDesc.Count = 4;

swapChainDesc.SampleDesc.Quality = m4xMsaaQuality-1;

}

else

{

sd.SampleDesc.Count = 1;

sd.SampleDesc.Quality = 0;

}

swapChainDesc.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;

swapChainDesc.BufferCount = 1;

swapChainDesc.OutputWindow = mhMainWnd;

swapChainDesc.Windowed = true;

swapChainDesc.SwapEffect = DXGI\_SWAP\_EFFECT\_DISCARD;

swapChainDesc.Flags = 0;

**创建交换链**

通过 IDXGIFactory 实例的 CreateSwapChain 接口创建交换链实例 IDXGISwapChain

HRESULT IDXGIFactory::CreateSwapChain(

IUnknown \*pDevice,

DXGI\_SWAP\_CHAIN\_DESC \*pDesc

IDXGISwapChain \*\*ppSwapChain);

pDevice

一个 ID3D11Device 的指针

pDesc

一个交换链描述结构 DXGI\_SWAP\_CHAIN\_DESC 的指针

ppSwapChain

返回生成的交换链实例

IDXGIFactory 实例需要通过一系列的 COM 查询获得

IDXGIDevice\* dxgiDevice = 0;

HR(md3dDevice->QueryInterface(\_\_uuidof(IDXGIDevice),(void\*\*)&dxgiDevice));

IDXGIAdapter\* dxgiAdapter = 0;

HR(dxgiDevice->GetParent(\_\_uuidof(IDXGIAdapter),(void\*\*))&dxgiAdapter));

IDXGIFactory\* dxgiFactory = 0;

HR(dxgiAdapter->GetParent(\_\_uuidof(IDXGIFactory),(void\*\*))&dxgiFactory));

IDXGISwapChain\* mSwapChain;

HR(dxgiFactory->CreateSwapChain(md3dDevice, )&sd, )&mSwapChain));

ReleaseCOM(dxgiDevice);

ReleaseCOM(dxgiAdapter);

ReleaseCOM(dxgiFactory);

**创建渲染对象视图**

ID3D11RenderTargetView\* renderTargetView;

ID3D11Texture2D\* backBuffer;

swapChain->GetBuffer ( 0, \_\_uuidof ( ID3D11Texture2D ), reinterpret\_cast<void\*\*>(&backBuffer) );

d3dDevice->CreateRenderTargetView ( backBuffer,0 , &renderTargetView );

ReleaseCOM ( backBuffer );

使用 IDXGISwapChain::GetBuffer 方法获取交换链的后缓存区.

参数分别为后缓存区的索引, 缓存的接口类型 ( 一般都是 ID3D11Texture2D ), 返回的后缓存区指针

使用 ID3D11Device::CreateRenderTargetView 方法创建 render target view

第一个参数是将要作为渲染对象的资源, 即交换链的后缓存区

第二个参数是一个 D3D11\_RENDER\_TARGET\_VIEW\_DESC 结构, 用于描述资源的格式, 如果资源不是以未知格式方式创建的, 则这个参数可以为 NULL

调用 IDXGISwapChain::GetBuffer 会增加后缓存区的 COM 引用计数, 所以要在使用完后用 ReleaseCOM 释放

**创建深度/模板缓存**

深度/模板缓存实质是一个 2D 纹理

创建纹理首先需填充一个 D3D11\_TEXTURE2D\_DESC 结构, 然后传递给 ID3DDevice::CreateTexture2D 方法

typedef struct D3D11\_TEXTURE2D\_DESC

{

UINT Width;

UINT Height;

UINT MipLevels;

UINT ArraySize;

DXGI\_FORMAT Format;

DXGI\_SAMPLE\_DESC SampleDesc;

D3D11\_USAGE Usage;

UINT BindFlags;

UINT CPUAccessFlags;

UINT MiscFlags;

} D3D11\_TEXTURE2D\_DESC;

Width

纹理的宽, 单位为像素

Height

纹理的高, 单位为像素

MipLevels

纹理的 mipmap 等级, 对于深度/模板缓存只需要 1 个

ArraySize

纹理序列中的纹理个数, 对于深度/模板缓存只需要 1 个

Format

纹理的像素格式, DXGI\_FORMAT 枚举的成员

SampleDesc

多重采样的数量和质量等级. 深度/模板缓存的多重采样设置需和渲染目标的一样

Usage

纹理的用处. D3D11\_USAGE 枚举的成员, 可用的值有

D3D11\_USAGE\_DEFAULT

GPU 可以对该资源读取和写入, CPU 不行

深度/模板缓存用这个

D3D11\_USAGE\_IMMUTABLE

当资源创建后就不再改变的. 只有 GPU 能读取

D3D11\_USAGE\_DYNAMIC

当应用程序需要经常更新资源的内容时指定这个类型, 但有一定的性能损失

CPU 可以向资源写入, GPU 可以读取资源

D3D11\_USAGE\_STAGING

当应用程序需要读取资源的副本时指定这个类型

BindFlags

指定资源应绑定到管线的哪一部分的标识.

对于深度/模板缓存, 指定 D3D11\_BIND\_DEPTH\_STENCIL

CPUAccessFlags

CPU 的访问标识, 可用值有

D3D11\_CPU\_ACCESS\_WRITE

CPU 可以向资源写入, 需要 Usage 为 D3D11\_USAGE\_DYNAMIC

D3D11\_CPU\_ACCESS\_READ

CPU 可以读取资源, 需要 Usage 为 D3D11\_USAGE\_STAGING

对于深度/模板缓存, CPU 不需要读取和写入, 所以指定 0

MiscFlags

对于深度/模板缓存不需要, 设为 0

D3D11\_TEXTURE2D\_DESC depthStencilDesc;

depthStencilDesc.Width = clientWidth;

depthStencilDesc.Height = clientHeight;

depthStencilDesc.MipLevels = 1;

depthStencilDesc.ArraySize = 1;

depthStencilDesc.Format = DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT;

if( enable4xMsaa )

{

depthStencilDesc.SampleDesc.Count = 4;

depthStencilDesc.SampleDesc.Quality = 4xMsaaQuality-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;

ID3D11Texture2D\* depthStencilBuffer;

ID3D11DepthStencilView\* depthStencilView;

HR( md3dDevice->CreateTexture2D( &depthStencilDesc, 0, &depthStencilBuffer ) );

HR( md3dDevice->CreateDepthStencilView( depthStencilBuffer, 0, &depthStencilView));

**绑定视图到输出合成阶段**

d3dImmediateContext->OMSetRenderTargets ( 1, &renderTargetView , depthStencilView );

第一个参数为要绑定的渲染对象的个数

第二个参数为渲染对象数组的第一个元素的指针

第三个参数为要绑定深度/模板缓存

**设置视口**

后缓存区中实际进行绘制的矩形区域称为视口. 用 D3D11\_VIEWPORT 结构描述

typedef struct D3D11\_VIEWPORT

{

FLOAT TopLeftX;

FLOAT TopLeftY;

FLOAT Width;

FLOAT Height;

FLOAT MinDepth;

FLOAT MaxDepth;

} D3D11\_VIEWPORT;

结构的前 4 个成员定义视口相对窗口客户区的位置

MinDepth 和 MaxDepth 定义最小和最大的深度缓存值. 除非用于特殊效果, 否则应与 Direct3D 一致设为 0 和 1

填充完 D3D11\_VIEWPORT 结构后, 使用 ID3D11DeviceContext::RSSetViewports 方法设置视口

D3D11\_VIEWPORT vp;

vp.TopLeftX = 0.0f;

vp.TopLeftY = 0.0f;

vp.Width = static\_cast<float>(mClientWidth);

vp.Height = static\_cast<float>(mClientHeight);

vp.MinDepth = 0.0f;

vp.MaxDepth = 1.0f;

d3dImmediateContext->RSSetViewports(1, &vp);

RSSetViewports 的第一个参数为要设置的视口的个数, 第二个参数是一个 D3D11\_VIEWPORT 结构的数组

**计时器**

\_\_int64 currentCount;

\_\_int64 countsPerSecond;

QueryPerformanceCounter((LARGE\_INTEGER\*)&currentCount );

QueryPerformanceFrequency((LARGE\_INTEGER\*)&countsPerSecond );

QueryPerformanceCounter 获取当前的计时器计数次数

QueryPerformanceFrequency 获取每秒计时器计数的次数