Hogeschool van Amsterdam

DirectX Initialization

Advanced Graphics Programming

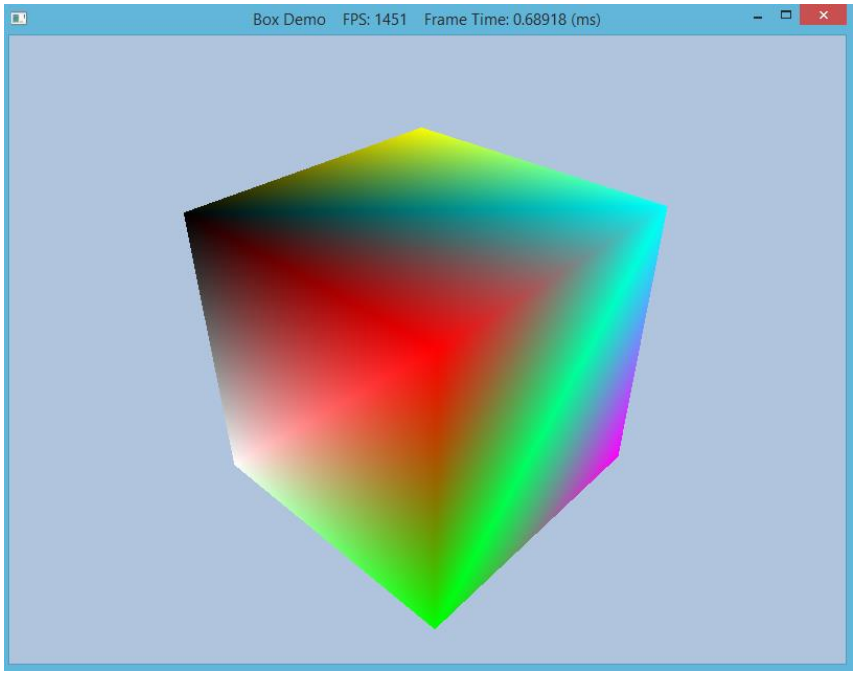
Alexander Bonnee,
Richard de Koning,
Reggie Schildmeijer

CREATING TOMORROW

DirectX Initialization

- End result ☺
- Win32 Game Loop
- Device,
- Device Context,
- Resources & Binding,
- Resource views
- Init
- Create
- Swap chain
- Depth stencil
- Exercise

End result ☺




```
class BoxApp : public D3DApp
```

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE prevInstance,
    PSTR cmdLine, int showCmd)
{
    BoxApp theApp(hInstance);

    if (!theApp.Init())
        return 0;

    return theApp.Run();
}
```

```
bool BoxApp::Init()
{
    if (!D3DApp::Init())
        return false;

    BuildGeometryBuffers();
    BuildFX();
    BuildVertexLayout();

    return true;
}
```

WIN32 GAME LOOP

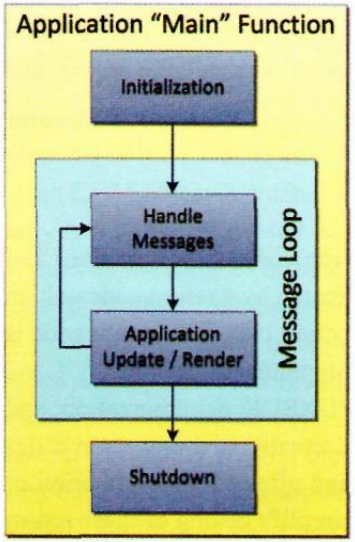
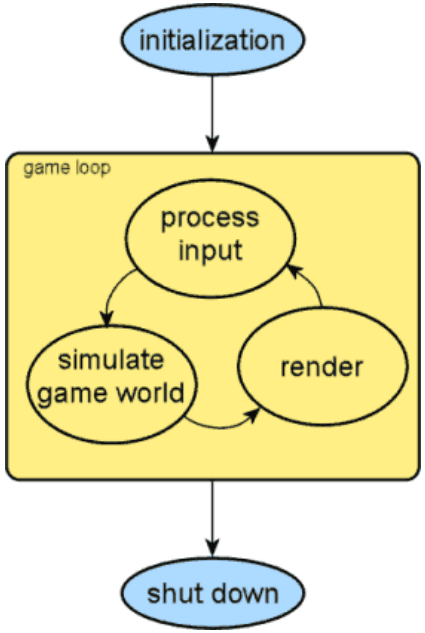


Figure 1.3. The standard operations performed in a Win32 application.



- Win32 app
- Game loop
- Rendering pipeline

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE prevInstance,
    PSTR cmdLine, int showCmd)
{
    InitDirect3DApp theApp(hInstance);

    if (!theApp.Init())
        return 0;

    return theApp.Run();
}
```

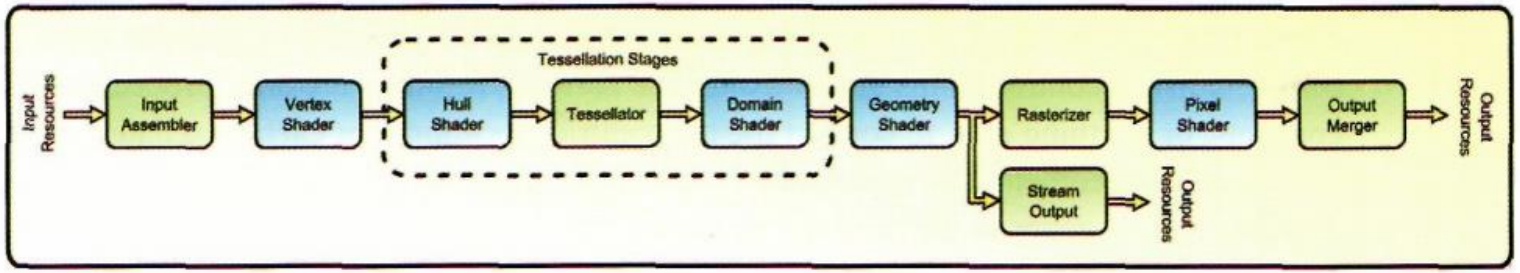


Figure 1.2. The complete Direct3D 11 rendering pipeline.

D3DAPP.H

```
class D3DApp
{
public:
    int Run();

    // Framework methods.  Derived client class overrides these methods to
    // implement specific application requirements.

    virtual bool Init();
    virtual void OnResize();
    virtual void UpdateScene(float dt) = 0;
    virtual void DrawScene() = 0;
    virtual LRESULT MsgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);

    // Convenience overrides for handling mouse input.
    virtual void OnMouseDown(WPARAM btnState, int x, int y){ }
    virtual void OnMouseUp(WPARAM btnState, int x, int y)  { }
    virtual void OnMouseMove(WPARAM btnState, int x, int y){ }

protected:
    bool InitMainWindow();
    bool InitDirect3D();

    void CalculateFrameStats();
};
```

REVIEW D3DAPP.CPP

- Init()
  - InitMainWindow()
  - InitDirect3D()
- Run()

- What is MsgProc(…) ?
- What happens with OnResize()?

Run()

```
int D3DApp::Run()
{
    MSG msg = {0};

    mTimer.Reset();

    while(msg.message != WM_QUIT)
    {
        // If there are Window messages then process them.
        if(PeekMessage( &msg, 0, 0, 0, PM_REMOVE ))
        {
            TranslateMessage( &msg );
            DispatchMessage( &msg );
        }
        // Otherwise, do animation/game stuff.
        else
        {
            mTimer.Tick();

            if( !mAppPaused )
            {
                CalculateFrameStats();
                UpdateScene(mTimer.DeltaTime());
                DrawScene();
            }
            else
            {
                Sleep(100);
            }
        }
    }

    return (int)msg.wParam;
}
```

- Message pump
- Stats/Update/Draw

Graphics Concepts

The Direct3D 11 **object model** separates

- resource creation and rendering functionality into a **device** and
- one or more **device contexts**;
- this separation is designed to facilitate multithreading.

Device

ID3D11Device is the software counterpart of its (physical) graphics display adapter (hardware).

- Interact with hardware
- Check feature support (e.g. pixel, shader models)
- Allocate resources (e.g. buffers, textures)

Most applications create a device for the hardware driver installed on your machine by calling D3D11CreateDevice or 3D11CreateDeviceAndSwapChain and specify the driver type with the D3D_DRIVER_TYPE flag.

Device Context

ID3D11DeviceContext is used to set pipeline state and execute rendering commands using the resources owned by a device.

- Immediate or deferred rendering (multithreaded)
- Binds resources (shaders/buffers) to the pipeline
- Renders stuff
- Very very rich interface (100+ methods)

Checkout ID3D11DeviceContext on MSDN! Study the responsibility.

Resource Binding

- Buffers and textures
- CPU/GPU Access
- Semantics (bind flags)

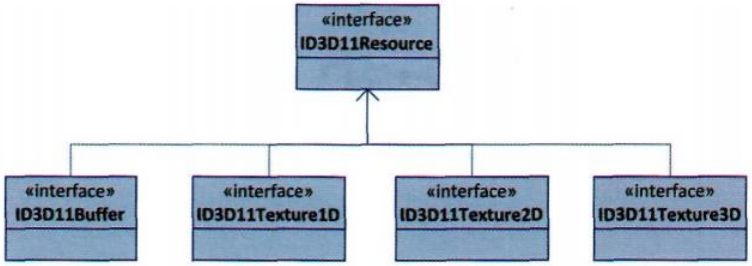
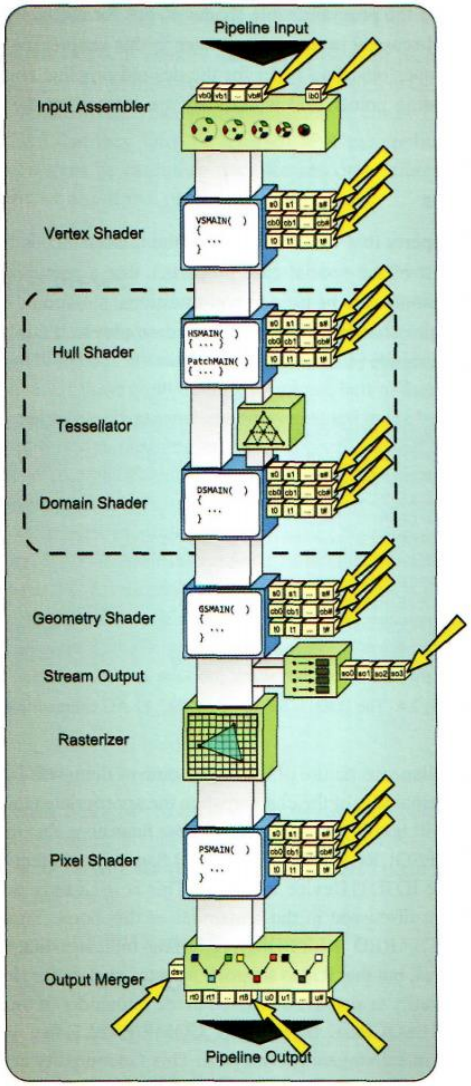


Figure 2.1. The Direct3D 11 Resource Interface Hierarchy.

Resource Usage	Default	Dynamic	Immutable	Staging
GPU-Read	yes	yes	yes	yes
GPU-Write	yes			yes
CPU-Read				yes
CPU-Write		yes		yes

Table 2.1. The accessibility defined for each usage flag.




```
enum D3D11_BIND_FLAG {
    D3D11_BIND_VERTEX_BUFFER,
    D3D11_BIND_INDEX_BUFFER,
    D3D11_BIND_CONSTANT_BUFFER,
    D3D11_BIND_SHADER_RESOURCE,
    D3D11_BIND_STREAM_OUTPUT,
    D3D11_BIND_RENDER_TARGET,
    D3D11_BIND_DEPTH_STENCIL,
    D3D11_BIND_UNORDERED_ACCESS
}
```

Figure 2.2. The resource binding locations of the rendering pipeline.

Resource Views

- Render target view (ID3D11RenderTargetView)
  - Receive output of the rendering pipeline
- Depth stencil view (ID3D11DepthStencilView)
  - Used in depth and stencil tests
- Shader resource view (ID3D11ShaderResourceView)
  - Input for shader (for example a texture)
- Unordered acces view (ID3D11UnorderedAccessView)
  - Output for shader (for example a texture)

Check OnResize() !

REVIEW INITDIRECT3D()

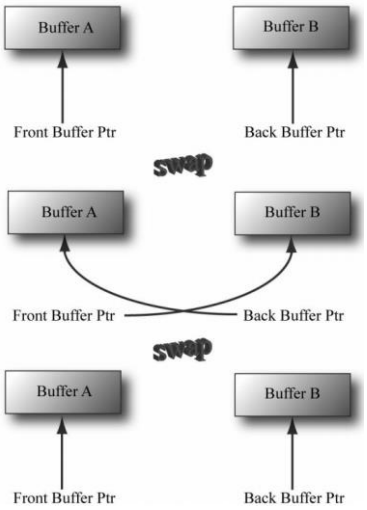
```
bool D3DApp::InitDirect3D()
{
    // Create the device and device context.

    UINT createDeviceFlags = 0;
#if defined(DEBUG) || defined(_DEBUG)
    createDeviceFlags |= D3D11_CREATE_DEVICE_DEBUG;
#endif

    D3D_FEATURE_LEVEL featureLevel;
    HRESULT hr = D3D11CreateDevice(
            0,                  // default adapter
            md3dDriverType,
            0,                  // no software device
            createDeviceFlags,
            0, 0,               // default feature level array
            D3D11_SDK_VERSION,
            &md3dDevice,
            &featureLevel,
            &md3dImmediateContext);

    if( FAILED(hr) )
    {
        MessageBox(0, L"D3D11CreateDevice Failed.", 0, 0);
        return false;
    }

    if( featureLevel != D3D_FEATURE_LEVEL_11_0 )
    {
        MessageBox(0, L"Direct3D Feature Level 11 unsupported.", 0, 0);
        return false;
    }

    // Check 4X MSAA quality support for our back buffer format.
    // All Direct3D 11 capable devices support 4X MSAA for all render
    // target formats, so we only need to check quality support.

    HR(md3dDevice->CheckMultisampleQualityLevels(
        DXGI_FORMAT_R8G8B8A8_UNORM, 4, &m4xMsaaQuality));
    assert( m4xMsaaQuality > 0 );

    // Fill out a DXGI_SWAP_CHAIN_DESC to describe our swap chain.

    DXGI_SWAP_CHAIN_DESC sd;
    sd.BufferDesc.Width  = mClientWidth;
    sd.BufferDesc.Height = mClientHeight;
    sd.BufferDesc.RefreshRate.Numerator = 60;
    sd.BufferDesc.RefreshRate.Denominator = 1;
    sd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
    sd.BufferDesc.ScanlineOrdering = DXGI_MODE_SCANLINE_ORDER_UNSPECIFIED;
    sd.BufferDesc.Scaling = DXGI_MODE_SCALING_UNSPECIFIED;

    // Use 4X MSAA?
    if( mEnable4xMsaa )
    {
        sd.SampleDesc.Count   = 4;
        sd.SampleDesc.Quality = m4xMsaaQuality-1;
    }
    // No MSAA
    else
    {
        sd.SampleDesc.Count   = 1;
        sd.SampleDesc.Quality = 0;
    }

    sd.BufferUsage  = DXGI_USAGE_RENDER_TARGET_OUTPUT;
    sd.BufferCount  = 1;
    sd.OutputWindow = mhMainWnd;
    sd.Windowed     = true;
    sd.SwapEffect   = DXGI_SWAP_EFFECT_DISCARD;
    sd.Flags        = 0;

    // To correctly create the swap chain, we must use the IDXGIFactory that was
    // used to create the device.  If we tried to use a different IDXGIFactory instance
    // (by calling CreateDXGIFactory), we get an error: "IDXGIFactory::CreateSwapChain:
```

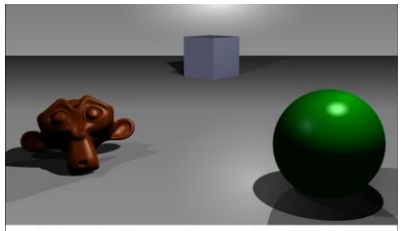
- Create Device
- Swap chain
- Render target
- Depth/stencil view
- Viewport
- Multi sampling

InitDirect3D()

- D3D11CreateDevice(…)
- CreateSwapChain(..)
- OnResize()

  binds resources and sets rendering states
  - Swap Chain
  - Render Target View
  - Depth/Stencil View
  - Viewport
  - Multisampling





A simple three-dimensional scene

Z-buffer representation

D3D11CreateDevice(…)

```
HRESULT D3D11CreateDevice(
  _In_opt_        IDXGIAdapter        *pAdapter,
                  D3D_DRIVER_TYPE     DriverType,
                  HMODULE             Software,
                  UINT                Flags,
  _In_opt_  const D3D_FEATURE_LEVEL   *pFeatureLevels,
                  UINT                FeatureLevels,
                  UINT                SDKVersion,
  _Out_opt_       ID3D11Device        **ppDevice,
  _Out_opt_       D3D_FEATURE_LEVEL   *pFeatureLevel,
  _Out_opt_       ID3D11DeviceContext **ppImmediateContext
);
```

```
typedef enum D3D_DRIVER_TYPE {
  D3D_DRIVER_TYPE_UNKNOWN     = 0,
  D3D_DRIVER_TYPE_HARDWARE    = ( D3D_DRIVER_TYPE_UNKNOWN + 1 ),
  D3D_DRIVER_TYPE_REFERENCE   = ( D3D_DRIVER_TYPE_HARDWARE + 1 ),
  D3D_DRIVER_TYPE_NULL        = ( D3D_DRIVER_TYPE_REFERENCE + 1 ),
  D3D_DRIVER_TYPE_SOFTWARE    = ( D3D_DRIVER_TYPE_NULL + 1 ),
  D3D_DRIVER_TYPE_WARP        = ( D3D_DRIVER_TYPE_SOFTWARE + 1 )
} D3D_DRIVER_TYPE;
```

```
typedef enum D3D11_CREATE_DEVICE_FLAG {
  D3D11_CREATE_DEVICE_SINGLETHREADED                                = 0x1,
  D3D11_CREATE_DEVICE_DEBUG                                         = 0x2,
  D3D11_CREATE_DEVICE_SWITCH_TO_REF                                 = 0x4,
  D3D11_CREATE_DEVICE_PREVENT_INTERNAL_THREADING_OPTIMIZATIONS      = 0x8,
  D3D11_CREATE_DEVICE_BGRA_SUPPORT                                  = 0x20,
  D3D11_CREATE_DEVICE_DEBUGGABLE                                    = 0x40,
  D3D11_CREATE_DEVICE_PREVENT_ALTERING_LAYER_SETTINGS_FROM_REGISTRY = 0x80,
  D3D11_CREATE_DEVICE_DISABLE_GPU_TIMEOUT                           = 0x100,
  D3D11_CREATE_DEVICE_VIDEO_SUPPORT                                 = 0x800
} D3D11_CREATE_DEVICE_FLAG;
```

```
{
    D3D_FEATURE_LEVEL_11_0,
    D3D_FEATURE_LEVEL_10_1,
    D3D_FEATURE_LEVEL_10_0,
    D3D_FEATURE_LEVEL_9_3,
    D3D_FEATURE_LEVEL_9_2,
    D3D_FEATURE_LEVEL_9_1,
};
```

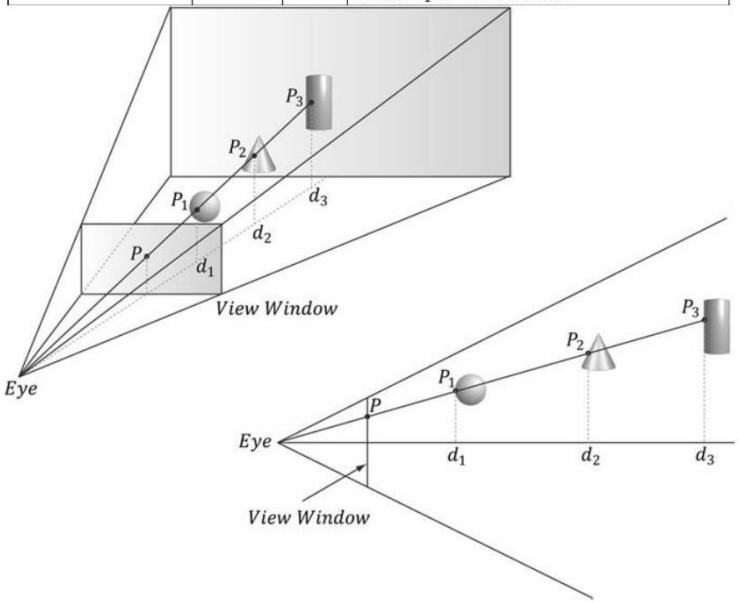
YES

CreateSwapChain(…)

```
struct DXGI_MODE_DESC {
  UINT                     Width;
  UINT                     Height;
  DXGI_RATIONAL            RefreshRate;
  DXGI_FORMAT              Format;
  DXGI_MODE_SCANLINE_ORDER ScanlineOrdering;
  DXGI_MODE_SCALING        Scaling;
}

struct DXGI_SAMPLE_DESC {
  UINT Count;
  UINT Quality;
}

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_USAGE_BACK_BUFFER
DXGI_USAGE_DISCARD_ON_PRESENT
DXGI_USAGE_READ_ONLY
DXGI_USAGE_RENDER_TARGET_OUTPUT
DXGI_USAGE_SHADER_INPUT
DXGI_USAGE_SHARED
DXGI_USAGE_UNORDERED_ACCESS

enum DXGI_SWAP_EFFECT {
  DXGI_SWAP_EFFECT_DISCARD,
  DXGI_SWAP_EFFECT_SEQUENTIAL
}

enum DXGI_SWAP_CHAIN_FLAG {
  DXGI_SWAP_CHAIN_FLAG_NONPREROTATED      = 1,
  DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH  = 2,
  DXGI_SWAP_CHAIN_FLAG_GDI_COMPATIBLE     = 4
}
```

Depth Stencil

Operation	P	d	Description
Clear Operation	Black	1.0	Pixel and corresponding depth entry initialized.
Draw Cylinder	P_3	d_3	Since $d_3 \leq d = 1.0$ the depth test passes and we update the buffers by setting $P = P_3$ and $d = d_3$.
Draw Sphere	P_1	d_1	Since $d_1 \leq d = d_3$ the depth test passes and we update the buffers by setting $P = P_1$ and $d = d_1$.
Draw Cone	P_1	d_1	Since $d_2 > d = d_1$ the depth test fails and we do not update the buffers.



Depth Stencil

Depth or Z-Buffer

- For each pixel, we store COLOR (color buffer) and DEPTH (depth buffer).

- Algorithm:

  Initialize all elements of buffer COLOR(row, col) to background color, and DEPTH(row, col) to maximum-depth;

  FOR EACH polygon:

    Rasterize polygon to frame;

    FOR EACH pixel center (x, y) that is covered:

      IF polygon depth at (x, y) < DEPTH(x, y)

        THEN COLOR(x, y) = polygon color at (x, y)

         AND DEPTH(x, y) = polygon depth at (x, y)

```
// Create the depth/stencil buffer and view.
HR(md3dDevice->CreateTexture2D(&depthStencilDesc, 0, &mDepthStencilBuffer));
HR(md3dDevice->CreateDepthStencilView(mDepthStencilBuffer, 0, &mDepthStencilView));

// Bind the render target view and depth/stencil view to the pipeline.
md3dImmediateContext->OMSetRenderTargets(1, &mRenderTargetView, mDepthStencilView);
```

```
HRESULT CreateDepthStencilView(
  [in]                ID3D11Resource                *pResource,
  [in, optional]  const D3D11_DEPTH_STENCIL_VIEW_DESC *pDesc,
  [out, optional]     ID3D11DepthStencilView        **ppDepthStencilView
);
```

CreateDepthStencilView(...)

```
// Create the depth/stencil description
D3D11_TEXTURE2D_DESC depthStencilDesc;

depthStencilDesc.Width = mClientWidth;
depthStencilDesc.Height = mClientHeight;
depthStencilDesc.MipLevels = 1;
depthStencilDesc.ArraySize = 1;
depthStencilDesc.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;

// Use 4X MSAA? --must match swap chain MSAA values.
if (mEnable4xMsaa)
{
    depthStencilDesc.SampleDesc.Count = 4;
    depthStencilDesc.SampleDesc.Quality = m4xMsaaQuality - 1;
}
// No MSAA
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;
```

- Pixel based
- Same dimensions
- Depends on sampling