

Gamefest

MICROSOFT GAME TECHNOLOGY CONFERENCE 2 0 1 0

Microsoft





DirectX 11 Technology Update

Chuck Walbourn
Senior Software Design Engineer
Microsoft

Microsoft®
DirectX[®]11

Gamefest
MICROSOFT GAME TECHNOLOGY CONFERENCE 2010

Gamefest 2008

Graphics Track

Introduction to the Direct3D 11 Graphics Pipeline

Direct3D 11 Tessellation

High Level Shader Language (HLSL) Update—Introducing Version 5.0

Multithreaded Rendering for Games

Direct3D 11 Compute Shader—More Generality for Advanced Techniques

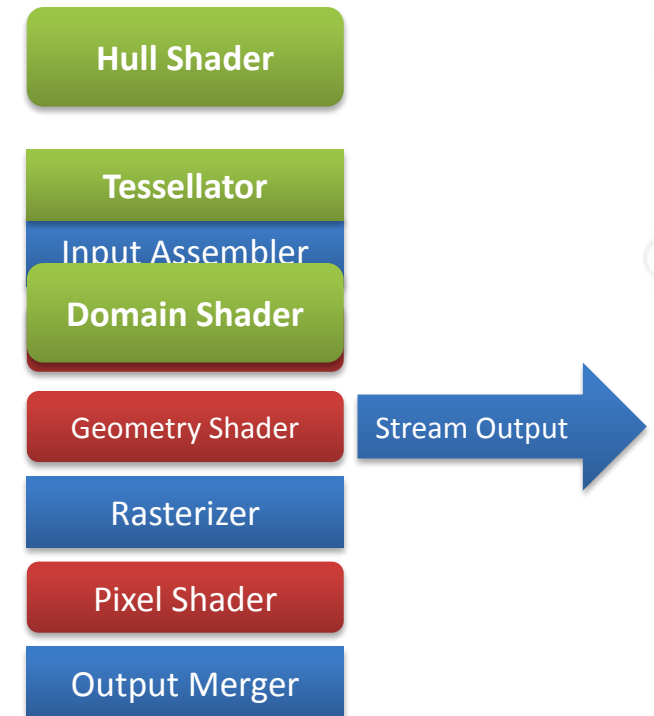
Partners Track

Advanced Topics in GPU Tessellation (AMD)

Water-Tight, Textured, Displaced Subdivision Surface Tessellation Using Direct3D 11 (NVIDIA)

DirectX 11 Overview

- Direct3D 11 is based on Direct3D 10.1
 - Similar API design & rendering pipeline
- Enables new DirectX 11 hardware features
- Supports existing DirectX 10 and 10.1 hardware
 - Enables some new features as well
- Supports a subset of DirectX 9 SM 2.0+ hardware
- Supported on Windows 7, Windows Vista, Windows Server 2008, and Windows Server 2008 R2





Multithreaded Rendering & Resource Creation

- Runtime emulation or Driver optimized



DirectCompute

- CS 5.0 on DirectX 11 HW
- CS 4.x on some DX10.x h/w



HLSL Shader Model 5

- 5.0 on DirectX 11 HW
- Some new constructs emulated on older profiles



Dynamic Shader Linkage

- Class Linkage on DirectX 11 HW
- interface construct works on older profiles



Hardware Tessellation

- DirectX 11 HW only






BC6H/BC7 Texture Formats

- DirectX 11 HW only

Features & Feature Levels

- Feature levels are well-defined sets of functionality
 - Not a sea of “opt-in” caps bits
- Each feature level is a superset of the previous one
- Concept was introduced with Direct3D 10.1
 - `ID3D10Device1::GetFeatureLevel()`
 - `D3D10_FEATURE_LEVEL_10_1`
 - `D3D10_FEATURE_LEVEL_10_0`

DirectX 11 Feature Levels

Feature Level	Capabilities
D3D_FEATURE_LEVEL_10_0 (e.g. NVIDIA GeForce 8000/9000 Series; GTX 260/280)	Direct3D 10 hardware: Shader Model 4.0, geometry shader, stream out, alpha-to-coverage, MSAA textures, 2-sided stencil, general render target views, texture arrays, BC4/BC5, optional DirectCompute (CS 4.0), all 9_3 features. 
D3D_FEATURE_LEVEL_10_1 (e.g. ATI Radeon HD 2000/3000/4000 Series; NVIDIA GeForce G210/GT220)	Direct3D 10.1 hardware: Shader Model 4.1, cubemap arrays, extended MSAA, optional DirectCompute (CS 4.1), all 10_0 features. 
D3D_FEATURE_LEVEL_11_0 (e.g. ATI Radeon HD 5000 Series)	Direct3D 11 hardware: Shader Model 5.0, hull & domain shaders, mandatory DirectCompute (CS 5.0), BC6H/BC7, all 10_1 features. 

10level9 Feature Levels

Feature Level	Capabilities
D3D_FEATURE_LEVEL_9_1 (e.g. Intel G965 Express Chipset, NVidia GeForce FX 5200)	Direct3D 9 hardware: must support Shader Model 2.0 (vs_2_0/ps_2_0), 2K textures, volume textures, event queries, BC1-3 (aka DXTn), and a few other specific capabilities.
D3D_FEATURE_LEVEL_9_2 (e.g. ATI Radeon 9500)	Direct3D 9 hardware: must support Shader Model 2.0 (vs_2_0/ps_2_0), occlusion queries, float formats (no blending), extended caps, all 9_1 features.
D3D_FEATURE_LEVEL_9_3 (e.g. NVidia GeForce 6600, ATI Radeon X1300)	Direct3D 9 hardware: must support Shader Model 2.0 (vs_2_0/ps_2_b) with instancing, 4K textures, multiple render targets (4 MRTs), floating-point blending, all 9_2 features.

Driver Optional Features

- Devices can expose some new **DXGI_FORMATs**
 - `ID3D11Device::CheckFormatSupport`
- **BGRA (B8G8R8*8) formats**
 - required for 9_1, 9_2, 9_3 and 11_0
 - optional for 10_0 / 10_1
- **10:10:10:2 X2 biased High Color mode (R10G10B10_XR_BIAS_X2_A2_UNORM)**
 - required for 11_0
 - optional for 10_0 / 10_1
 - not available for 9_1, 9_2, or 9_3
- **Majority of format support is defined by feature level**

Detailed in the DXGI Programmer's Guide in the Windows Graphics documentation

Driver Optional Features

- DirectX 11 drivers can support four optional features
 - `ID3D11Device::CheckFeatureSupport`
- DirectCompute
 - `D3D_FEATURE_LEVEL_10_0` / `10_1` support for CS 4.x is optional
CS 5.0 is required for `D3D_FEATURE_LEVEL_11_0`
- Double-precision shader support is optional
- Multithreading Driver support is optional
 - Concurrent object creation
 - Command lists

Related Technologies



DirectX Graphics Infrastructure (DXGI) 1.1

- Version 1.0 introduced in Windows Vista
- Enumerates adapters, display modes, and outputs (e.g. monitors)
- New `DXGI_FORMATs` and improved support for remote desktops



Windows Display Driver Model (WDDM) 1.1

- WDDM 1.0 introduced in Windows Vista
- Unified driver model for all Direct3D APIs



Windows Advanced Rasterization Platform (WARP) 10

- Software rasterization device supporting 10.1 device functionality
- No 'driver optional feature' support
- Much faster than the DirectX SDK's Reference device

Demo

- Updated DxCapsViewer Utility

- DXGI 1.1 Devices
 - NVIDIA GeForce 8600M GT
 - Outputs
 - Direct3D 10
 - Features
 - Shader sample (any filter)
 - Mipmap Auto-Generation
 - Render Target
 - Blendable Render Target
 - 2x MSAA
 - 4x MSAA
 - 8x MSAA
 - Other MSAA
 - MSAA Load
 - Direct3D 10.1
 - Direct3D 11
 - D3D_FEATURE_LEVEL_10_0**
 - Additional Feature Levels
 - D3D_FEATURE_LEVEL_9_3
 - D3D_FEATURE_LEVEL_9_2
 - D3D_FEATURE_LEVEL_9_1
 - Windows Advanced Rasterization Platform
 - Direct3D 10.1
 - Direct3D 11
 - Reference
 - Direct3D 10
 - Direct3D 10.1
 - Direct3D 11
 - D3D_FEATURE_LEVEL_11_0
 - Additional Feature Levels
 - 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
 - Shader sample (any filter)
 - Shader gather4
 - Mipmap Auto-Generation
 - Render Target
 - 2x MSAA
 - 4x MSAA (all required)
 - 8x MSAA (most required)

Name	Value
Shader Model	4.0
Geometry Shader	Yes
Stream Out	Yes
Compute Shader	Optional (Yes)
Hull & Domain Shaders	No
Texture Resource Arrays	Yes
Cubemap Resource Arrays	No
BC4/BC5 Compression	Yes
BC6H/BC7 Compression	No
Alpha-to-coverage	Yes
Extended Formats (BGRA, etc.)	Optional (Yes)
10-bit XR High Color Format	Optional (Yes)
Max Texture Dimension	D3D10_REQ_TEXTURE2D_U_OR_V_DIMENSION(8192)
Max Cubemap Dimension	D3D10_REQ_TEXTURECUBE_DIMENSION(8192)
Max Texture Repeat	D3D10_REQ_FILTERING_HW_ADDRESSABLE_RESOURCE_DIMENSION...
Note	This feature summary is derived from hardware feature level

Direct3D 10.0 API

ID3D10Device

ID3D10DeviceChild

ID3D10VertexShader

ID3D10GeometryShader

ID3D10PixelShader

ID3D10InputLayout

ID3D10DepthStencilState

ID3D10BlendState

ID3D10RasterizerState

ID3D10SamplerState

ID3D10Asynchronous

ID3D10Query

ID3D10Predicate

ID3D10Counter

ID3D10Resource

ID3D10Buffer

ID3D10Texture1D

ID3D10Texture2D

ID3D10Texture3D

ID3D10View

ID3D10ShaderResourceView

ID3D10RenderTargetView

ID3D10DepthStencilView

Direct3D 10.1 API

ID3D10Device1

ID3D10DeviceChild

ID3D10VertexShader

ID3D10GeometryShader

ID3D10PixelShader

ID3D10InputLayout

ID3D10DepthStencilState

ID3D10BlendState1

ID3D10RasterizerState

ID3D10SamplerState

ID3D10Asynchronous

ID3D10Query

ID3D10Predicate

ID3D10Counter

ID3D10Resource

ID3D10Buffer

ID3D10Texture1D

ID3D10Texture2D

ID3D10Texture3D

ID3D10View

ID3D10ShaderResourceView1

ID3D10RenderTargetView

ID3D10DepthStencilView

Direct3D 11 API (partial)

ID3D11Device

ID3D11DeviceChild

ID3D11Asynchronous

ID3D11Resource

ID3D11View

ID3D11VertexShader

ID3D11GeometryShader

ID3D11PixelShader

ID3D11InputLayout

ID3D11DepthStencilState

ID3D11BlendState

ID3D11RasterizerState

ID3D11SamplerState

ID3D11Query

ID3D11Predicate

ID3D11Counter

ID3D11Buffer

ID3D11Texture1D

ID3D11Texture2D

ID3D11Texture3D

ID3D11ShaderResourceView

ID3D11RenderTargetView

ID3D11DepthStencilView

ID3D11Device

ID3D11DeviceContext

ID3D11DeviceChild

- ID3D11VertexShader
- ID3D11GeometryShader
- ID3D11PixelShader
- ID3D11ComputeShader
- ID3D11DomainShader
- ID3D11HullShader
- ID3D11InputLayout
- ID3D11DepthStencilState
- ID3D11BlendState
- ID3D11RasterizerState
- ID3D11SamplerState
- ID3D11ClassInstance
- ID3D11ClassLinkage
- ID3D11CommandList

ID3D11Asynchronous

- ID3D11Query
- ID3D11Predicate
- ID3D11Counter

ID3D11Resource

- ID3D11Buffer
- ID3D11Texture1D
- ID3D11Texture2D
- ID3D11Texture3D

ID3D11View

- ID3D11ShaderResourceView
- ID3D11RenderTargetView
- ID3D11DepthStencilView
- ID3D11UnorderedAccessView

Direct3D 11 API Change

- Main difference is **ID3D10Device** was split in two
 - Object creation in **ID3D11Device** interface
 - Other methods split off into **ID3D11DeviceContext** interface
 - Rendering and state configuration
 - **Map()** / **Unmap()** from resource objects
 - **Begin()** , **End()** , and **GetData()** from query objects
 - One immediate context for rendering directly to the device
 - Zero or more deferred contexts for creating command lists
- Provides basis of multithreading improvements

Multi-threading Rules

- **ID3D11Device** is thread-safe
 - Without driver support for Concurrent Creates, runtime will enforce thread-safety with a coarse lock
 - Without driver support for Concurrent Creates, creating objects and rendering with the immediate context will not be concurrent (using the same coarse lock)
 - Methods on most other objects (**ID3D11DeviceChild**-derived) are also thread-safe
 - Can opt-out by using `D3D11_CREATE_DEVICE_SINGLETHREADED`

Multi-threading Rules

- **ID3D11DeviceContext** is not thread-safe
 - Typical usage is one device context per thread, one of them using immediate and the rest using deferred contexts
 - Note that DXGI methods should not be used concurrently while rendering with the immediate device context
 - For example, **Present()** uses the immediate device context
- It is thread-safe to use the methods inherited from **ID3D11DeviceChild**
 - **AddRef()**, **Release()**, **QueryInterface()**
 - **GetDevice()**, **GetPrivateData()**, **SetPrivateData()**, **SetPrivateDataInterface()**

Multi-threading Rules

- `ID3D11DeviceContext` deferred mode limitations
 - `Map ()` must be used with `D3D11_MAP_WRITE_DISCARD` and/or `D3D11_MAP_WRITE_NO_OVERWRITE`
 - `GetData ()` for queries is not allowed
 - Queries can be used in conjunction with predication
 - If executing a deferred command list with a query active and the command list itself uses the same query, then the command list submission is ignored as invalid

Multi-threading Recommendations

- Concurrent creation is a no brainer
 - Many engines already have resource loading threads
 - Runtime emulation is “good enough” for a win
 - Less overhead than the default Direct3D 10 M/T behavior
 - ConcurrentCreates driver support makes it better
 - When creating objects with M/T driver support, providing initial data for static objects should be more efficient
 - i.e Use the `pInitialData` parameter on the `Create` rather than staging resources, `UpdateSubResource()`, or `Map()` when possible

Multi-threading Recommendations

- Concurrent submission depends on the scenario
 - Useful for Triple-core, Quad-core, or more
 - For Dual-core, it is less likely to be worthwhile
 - Best use scenario is one rendering thread per core
 - Ideally use the Windows Vista/Windows 7 thread pool API
 - If you roll your own solution, see the DirectX SDK **CoreDetection** sample for the robust way to determine number of cores
 - Similar but not identical to Xbox 360 Command Buffers
 - Driver CommandLists support is currently rare

Porting to Direct3D 11 from 10.x

- Start with a simple text translation
 - `ID3D10*` -> `ID3D11*`
 - `D3D10_*` -> `D3D11_*`
- If starting with Direct3D 10.0, will need to fix up a few minor structure differences
 - (11 matches the 10.1 version)
 - `D3D10_BLEND_DESC1`
 - `D3D10_SHADER_RESOURCE_VIEW_DESC1`

Porting to Direct3D 11 from 10.x

- Change rendering & state calls from device to immediate context
 - After getting port done, will want to revisit this
- Change resource **Map ()** and query **Begin ()**, **End ()**, & **GetData ()** to use immediate context
- **Create*Shader** takes an additional class linkage parameter (can use **NULL**)
- ***SetShader** and ***GetShader** take an additional class instance parameter (can use **NULL**)

Porting to Direct3D 11 from 10.x

- Some defines changed, so be sure you aren't using magic numbers (`D3D10_RESOURCE_MISC_FLAG`)
- A minor feature was completely dropped
 - `ID3D10Device::GetTextFilterSize`
 - `ID3D10Device::SetTextFilterSize`
 - `D3D10_FILTER_TEXT_1BIT`
- Vendor-neutral performance counters removed
 - Were rarely implemented or consistent
 - i.e. `D3D11_COUNTER_DEVICE_DEPENDENT_0` is the only counter enumeration

Porting to Direct3D 11 from 9

- Essentially the same as porting from Direct3D 9 to Direct3D 10.x
 - Remove all fixed-function pipeline usage
 - Use state management based on immutable state objects
 - Obey strict shader linkage and input layout rules
 - Use shader resource views associated with texture resources
 - Map data to `DXGI_FORMATs` (no 16-bit formats, no 24-bit color format, strict RGB color order)
 - Rework global constant data into several constant buffers for efficient update

Porting to Direct3D 11 from 9

- Start with the existing guidance for moving from Direct3D 9 to Direct3D 10

SIGGRAPH 2007 Course #5

Introduction to Direct3D 10

<http://msdn.microsoft.com/directx/presentations>

Gamefest 2007 talk

“Windows to Reality: Getting the Most out of Direct3D 10 Graphics in Your Games”

<http://www.microsoftgamefest.com/presentations/2007.htm>

HLSL Compiler

- DirectX 11 requires 4.0 or later profile shaders
- D3DCompile DLL contains latest HLSL compiler
 - Used by D3DX9, D3DX10, D3DX11, and **FXC.EXE**
 - Can use directly (i.e. without using D3DX)
 - Note it is in its own DirectSetup CAB file in the REDIST folder
 - Supports all shader models except Pixel Shader 1.x profiles

Vertex	Pixel	Geometry	Compute	Hull	Domain
vs_4_0	ps_4_0	gs_4_0	cs_4_0		
vs_4_1	ps_4_1	gs_4_1	cs_4_1		
vs_5_0	ps_5_0	gs_5_0	cs_5_0	hs_5_0	ds_5_0
vs_4_0_level_9_1	ps_4_0_level_9_1				
vs_4_0_level_9_3	ps_4_0_level_9_3				

Shader Profiles and Feature Levels

- Shader profiles in DirectX 11 can be applied to higher feature levels, but not lower
- 10level9 shader profiles are compiled twice internally
 - `vs_4_0_level_9_*` => `vs_2_0` + `vs_4_0`
 - `ps_4_0_level_9_1` => `ps_2_0` + `ps_4_0`
 - `ps_4_0_level_9_3` => `ps_2_0` + `ps_4_0`

Shader Profile vs. Device Feature Level	11_0	10_1	10_0	9_3	9_2	9_1
5_0	Yes	-	-	-	-	-
4_1	Yes	Yes	-	-	-	-
4_0	Yes	Yes	Yes	-	-	-
4_0_level_9_3	Yes	Yes	Yes	Yes	-	-
4_0_level_9_1	Yes	Yes	Yes	Yes	Yes	Yes

HLSL Recommendations

- Use the latest compiler
 - Esp. avoid the ‘in box’ D3D10Compile APIs
- Generally use the lowest profile possible for VS/PS when supporting 10level9
- For DirectCompute
 - prefer CS 5.0 over CS 4.x
 - Prefer CS 4.1 over CS 4.0
- *Compile your shaders offline for your retail game*

Effects (FX) Library

- Effects for Direct3D 11 (FX11) is shared source in DX SDK
 - FX9 was in D3DX9
 - FX10 was in box with the OS
- Porting from FX10 -> FX11 is fairly trivial
 - Essentially the same API without effects pools
- Porting from FX9 -> FX11 requires significant code change

	HLSL Profile
Effects 9	fx_2_0
Effects 10	fx_4_0 and fx_4_1
Effects 11	fx_5_0

D3DX11

- Includes texture loaders (BMP, JPG, PNG, DDS, TIFF, GIF)
 - and asynchronous loaders introduced with D3DX10
- Does not include D3DX Math, Mesh, Sprite, or Font
 - See XNAMath as alternative for D3DX Math
 - and DXUT11 for alternative to font, etc.

D3DX11 uses a CPU codec for BC6H/BC7 texture compression, which can be time-consuming.

For a fast DirectCompute 4.x solution, see the `BC6HBC7EncoderDecoder11` sample.

D3DCSX

- Optional extended D3DX DLL for Compute Shader
 - Resides in its own DirectSetup / REDIST CAB
- DirectCompute (CS 5.0) utility functions
 - **ID3DX11Scan**
 - Unsegmented Scan or Multiscan
 - Segmented Scan
 - **ID3DX11FFT**
 - 1D, 2D, 3D support
 - Real or Complex
 - Forward or Inverse Transform with optional scale

XNAMath

- aka xboxmath 2.0
- Inline C++ SSE/SSE2 optimized math library
 - VMX128 optimized on Xbox
- ~350 functions
 - Focused on single-precision floating-point operations
 - Limited integer operations
 - Conversion to/from packed graphics formats
 - Implemented using Visual Studio intrinsics
 - Supports x86 and x64 native
- Common 3D primitives
 - Vectors, matrices, planes, quaternions, etc.

DirectX 11 Deployment

- DirectX 11 Runtime is included with Windows 7 and Windows Server 2008 R2
- DirectX 11 Runtime can be deployed down-level to Windows Vista / Server 2008
- D3DX11, D3DCSX, D3DCompile, etc. installed by DirectSetup / DX SDK REDIST
 - Just like D3DX9, D3DX10, XAUDIO2, etc.
- **The DirectX SDK does *not* install the DirectX 11 Runtime**
 - The DX SDK does install the debug layers and reference device

DirectX 11 Runtime

Direct3D 11

- New API supporting 10, 10.1, 11, 10level9, and WARP10

DXGI 1.1

- D3D glue library updated for new formats and WDDM 1.1 driver features

WARP10

- 10.1 level software renderer

10level9

- Direct3D 9 Shader Model 2.0 h/w support (9_1, 9_2, 9_3 feature levels)

Direct3D 10.1

- Updated existing API to support WARP10, 10level9

Direct2D

- GDI-like 2D drawing API for working on Direct3D surfaces

DirectWrite

- High-quality, feature-rich font rendering API (works with Direct2D)

KB 971644

- *Platform Update for Windows Vista*
<http://go.microsoft.com/fwlink/?LinkId=160189>
- Deployed through Windows Update
- Requires Windows Vista / Server 2008 SP2 to be installed

See the `D3D11InstallHelper` sample in the DirectX SDK for detection, applying the KB, and messaging for RTM / SP1

Detailed in *Direct3D 11 Deployment for Game Developers* technical article

KB 971512

- For corporate network environments using Windows Server Update Servers (WSUS), KB 971644 is not available
- Use this update instead
Windows Graphics, Imaging, and XPS Library
<http://support.microsoft.com/kb/971512/>
- Local IT admin will need to approve the update through the managed WSUS servers
- Requires Windows Vista / Server 2008 SP2 to be installed

Recommendations

- Update your existing Direct3D 10.x code path to use Direct3D 11
 - This requires some installer/deployment work
 - Your DX11 code path will require Windows Vista SP2+ or Windows 7
- For Windows Vista / Windows 7 titles
10level9 feature levels can provide more hardware support, so you don't need a Direct3D9 code path
- For titles that need Windows XP support,
you will need a legacy Direct3D9 code path

Recommendations

- If you still only have a legacy Direct3D 9 code path
 - Now's the time to invest in DirectX 11
 - Take advantage of the existing resources
 - Lessons learned moving from D3D9 -> D3D10 all apply to moving from D3D9 -> D3D11
- Direct3D 11 provides
 - the latest hardware features
 - new features for existing 10.x hardware
 - and supports the majority of video cards with WDDM drivers

Resources

- Latest DirectX SDK
<http://msdn.microsoft.com/directx>
- Gamefest 2008 Graphics and Partners Tracks
<http://www.microsoftgamefest.com/presentations/2008.htm>
- Gamefest 2010 Graphics Track
“Think DirectX11 Tessellation! - what are your options?”
“DirectX 11 DirectCompute - A Teraflop for Everyone”
“Block Compression Smorgasbord”
and additional talks from AMD & NVIDIA



APPENDIX

```
#include "d3d10.h"
IDXGISwapChain *g_pSwapChain = NULL;
ID3D10Device *g_pDevice = NULL;
...
DXGI_SWAP_CHAIN_DESC sd;
// Set to desired values
...
HRESULT res = D3D10CreateDeviceAndSwapChain( NULL, D3D10_DRIVER_TYPE_HARDWARE,
        NULL, 0, D3D10_SDK_VERSION, &sd,
        &g_pSwapChain, &g_pDevice );

if ( FAILED(res) ) // Error Handling
// Bind render target from swap chain
// Set up viewport
...

```

```
#include "d3d10_1.h"
IDXGISwapChain *g_pSwapChain = NULL;
ID3D10Device1 *g_pDevice = NULL;
...
DXGI_SWAP_CHAIN_DESC sd;
// Set to desired values
...
HRESULT res = D3D10CreateDeviceAndSwapChain1( NULL, D3D10_DRIVER_TYPE_HARDWARE,
    NULL, 0, D3D10_FEATURE_LEVEL_10_1, D3D10_1_SDK_VERSION, &sd,
    &g_pSwapChain, &g_pDevice );

if ( FAILED(res) )
{
    res = D3D10CreateDeviceAndSwapChain1( NULL, D3D10_DRIVER_TYPE_HARDWARE,
        NULL, 0, D3D10_FEATURE_LEVEL_10_0, D3D10_1_SDK_VERSION, &sd,
        &g_pSwapChain, &g_pDevice );
}

if ( FAILED(res) ) // Error Handling
// Bind render target from swap chain
// Set up viewport
...
// use g_pDevice->GetFeatureLevel() to check for 10_1; can otherwise assume 10_0
```

```
#include "d3d11.h"
IDXGISwapChain *g_pSwapChain = NULL;
ID3D11Device *g_pDevice = NULL;
ID3D11DeviceContext* g_pContext = NULL;
...
DXGI_SWAP_CHAIN_DESC sd;
// Set to desired values
...

D3D_FEATURE_LEVEL flv1[] = {
    D3D_FEATURE_LEVEL_11_0, D3D_FEATURE_LEVEL_10_1, D3D_FEATURE_LEVEL_10_0 };

D3D_FEATURE_LEVEL fl;
HRESULT res = D3D11CreateDeviceAndSwapChain( NULL, D3D_DRIVER_TYPE_HARDWARE,
    NULL, 0, flv1, sizeof(flv1)/sizeof(D3D_FEATURE_LEVEL),
    D3D11_SDK_VERSION, &sd,
    &g_pSwapChain, &g_pDevice, &fl, &g_pContext );

if ( FAILED(res) ) // Error Handling
// Bind render target from swap chain
// Set up viewport
...
// use g_pDevice->GetFeatureLevel() (or remember fl above) to check for
// 11_0 or 10_1, assume 10_0 otherwise
```

```
#include "d3d11.h"
IDXGISwapChain *g_pSwapChain = NULL;
ID3D11Device *g_pDevice = NULL;
ID3D11DeviceContext* g_pContext = NULL;
...
DXGI_SWAP_CHAIN_DESC sd;
// Set to desired values
...

D3D_FEATURE_LEVEL flv1[] = {
    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 };

D3D_FEATURE_LEVEL fl;
HRESULT res = D3D11CreateDeviceAndSwapChain( NULL, D3D_DRIVER_TYPE_HARDWARE,
    NULL, 0, flv1, sizeof(flv1)/sizeof(D3D_FEATURE_LEVEL),
    D3D11_SDK_VERSION, &sd,
    &g_pSwapChain, &g_pDevice, &fl, &g_pContext );

if ( FAILED(res) ) // Error Handling
// Bind render target from swap chain
// Set up viewport
...
// use g_pDevice->GetFeatureLevel() (or remember fl above) to check feature level
```



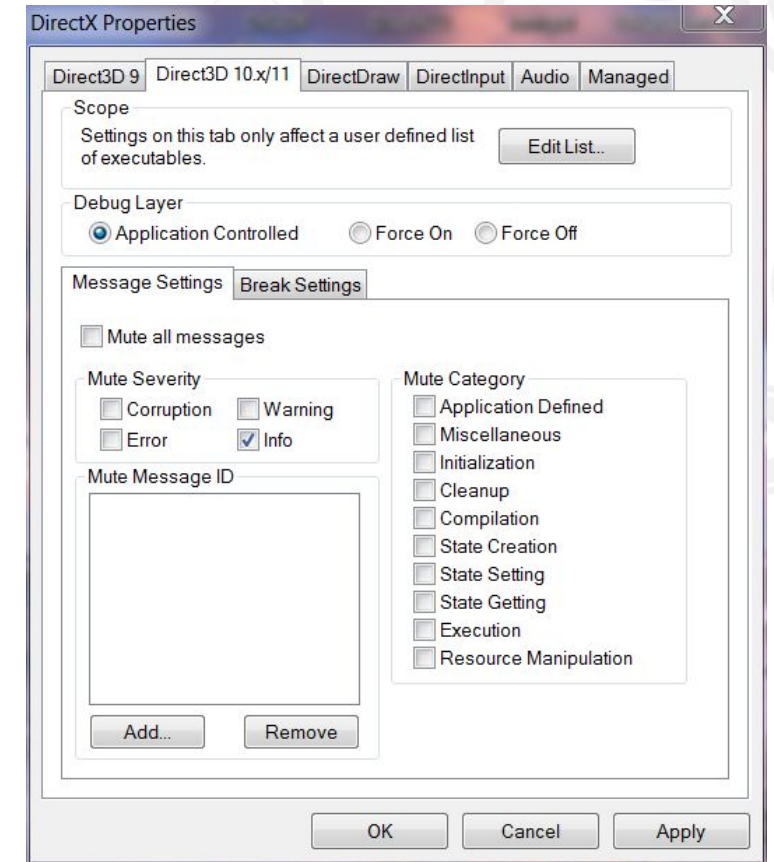
```
#include "d3d11.h"
IDXGISwapChain *g_pSwapChain = NULL;
ID3D11Device *g_pDevice = NULL;
ID3D11DeviceContext* g_pContext = NULL;
...
DXGI_SWAP_CHAIN_DESC sd;
// Set to desired values
...

HRESULT res = D3D11CreateDeviceAndSwapChain( NULL, D3D_DRIVER_TYPE_HARDWARE,
      NULL, 0, NULL, 0,
      D3D11_SDK_VERSION, &sd,
      &g_pSwapChain, &g_pDevice, NULL, &g_pContext );

if ( FAILED(res) ) // Error Handling
// Bind render target from swap chain
// Set up viewport
...
// use g_pDevice->GetFeatureLevel() to check feature level
```

DirectX 11 Debugging

- DirectX SDK provides debugging layer
- Enabled through code (`D3D11_CREATE_DEVICE_DEBUG`) or the DirectX Control Panel utility
 - Control panel controls the 10 and 11 debugging layer through the same settings
 - Unlike Direct3D 9, it is per application not a global setting
- Prints messages to Windows debug output



DirectX 11 Debugging

- Make sure your application runs ‘debug layer’ clean
 - ERROR and CORRUPTION reports are critical to fix
 - Tools like *PIX for Windows* assume this level of correctness
- Can also make use of the `ID3D11Debug` and `D3D11InfoQueue` interfaces
 - Obtain via `QueryInterface` from Direct3D 11 Device
 - Exists only if debug layer is attached
 - `ID3D10Debug::Validate()` split into `ValidateContext()` and `ValidateContextForDispatch()`
 - New method for DX11 Debug Layer
`ID3D11Debug::ReportLiveDeviceObjects()`

Debug Resource Naming

- Debug layer messages in debug window use ‘friendly names’ for resources, defaults to “unnamed”
- Can set the name by using the `SetPrivateData()` API in combination with a ‘well-known’ GUID from `d3dcommon.h`

```
#ifndef NDEBUG
// Only works if device is created with the D3D10 or D3D11 debug layer
const char c_szName[] = "texture.jpg";
pObject->SetPrivateData( WKPDID_D3DDebugObjectName,
    sizeof( c_szName ) - 1, c_szName );
#endif
```

Windows 7 / Server 2008 R2 only

- A few DirectX-branded technology pieces are not available down-level
 - Direct3D9Ex video HD and overlay extensions
 - Direct3D9Ex `D3DSWAPEFFECT_FLIPEX` and improved frame statistics
 - DirectMusic 'core' API for x64 native (time stamped MIDI, software synthesizer)
- Windows Media Foundation improvements are not also included



Gamefest

MICROSOFT GAME TECHNOLOGY CONFERENCE 2 0 1 0

www.microsoftgamefest.com