Introduction to DirectX Programming

Basic Direct3D Programming

Contents

- Direct3D concepts
  - Device
  - Resource
- Direct3D helper library
- Direct3D programming
  - Setting up each pipeline stage
  - Copying and accessing resource data
  - Programming with HLSL

Device

- Gateway to Direct3D functionalities
- Abstraction of one or more physical devices
  - Allocates and destroys objects
  - Renders primitives
  - Communicates with graphics driver and hardware
- Associated with one of more device contexts
- Device context
  - Sets pipeline state
  - Generates rendering commands

D3D11CreateDevice Function

Syntax

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

Example

```
ID3D11Device* pDev = NULL;
ID3D11DeviceContext* pDevImmCtx = NULL;

hr = D3D11CreateDevice(
    …
    &pDev,
    …
    &pDevImmCtx
);

pDevImmCtx->Release();
pDev->Release();
```

D3D11CreateDeviceAndSwapChain Function

Syntax

```
HRESULT D3D11CreateDeviceAndSwapChain(
    __in   IDXGIAdapter *pAdapter,
    __in   D3D_DRIVER_TYPE DriverType,
    __in   HMODULE Software,
    __in   UINT Flags,
    __in   const D3D_FEATURE_LEVEL *pFeatureLevels,
    __in   UINT FeatureLevels,
    __in   UINT SDKVersion,
    __in   const DXGI_SWAP_CHAIN_DESC *pSwapChainDesc,
    __out  IDXGISwapChain **ppSwapChain,
    __out  ID3D11Device **ppDevice,
    __out  D3D_FEATURE_LEVEL *pFeatureLevel,
    __out  ID3D11DeviceContext **ppImmediateContext
);
```

Example

```
ID3D11Device* pDev = NULL;
ID3D11DeviceContext* pDevImmCtx = NULL;
IDXGISwapChain* pSwapChain = NULL;

hr = D3D11CreateDevice(
    …
    &pSwapChain, &pDev, …, &pDevImmCtx
);

pSwapChain->Release();
pDevImmCtx->Release();
pDev->Release();
```

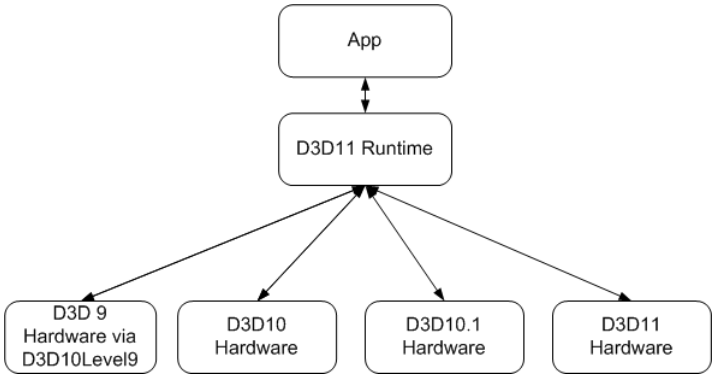
Creating Swap Chain

1) Fill DXGI_SWAP_CHAIN_DESC structure
   - Display mode (width, height, refresh rate, format, …)
   - Other properties of swap chain
2) Obtain IDXGISwapChain interface
   - D3D11CreateDeviceAndSwapChain
     - Initialize swap chain at the same time as device creation
     - Usually the easiest option
   - IDXGIFactory::CreateSwapChain
     - After creating swap chain, call D3D11CreateDevice to create device

Feature Level

- Direct3D 11 applications can run on a downlevel hardware
- D3D_FEATURE_LEVEL enumeration

```
enum D3D_FEATURE_LEVEL {
    D3D_FEATURE_LEVEL_9_1    = 0x9100,
    D3D_FEATURE_LEVEL_9_2    = 0x9200,
    D3D_FEATURE_LEVEL_9_3    = 0x9300,
    D3D_FEATURE_LEVEL_10_0   = 0xa000,
    D3D_FEATURE_LEVEL_10_1   = 0xa100,
    D3D_FEATURE_LEVEL_11_0   = 0xb000
};
```



	11_0	10_1	10_0	9_3	9_2	9_1
Shader Model	5.0	4.x	4.0	2.0 (4_0_level_9_3) [vs_2_a/ps_2_b]	2.0 (4_0_level_9_1)	2.0 (4_0_level_9_1)
Geometry Shader	Yes	Yes	Yes	No	No	No
Stream Out	Yes	Yes	Yes	No	No	No
DirectCompute / Compute Shader	Yes	Optional	Optional	N/A	N/A	N/A
Hull and Domain Shaders	Yes	No	No	No	No	No
Texture Resource Arrays	Yes	Yes	Yes	No	No	No
Cubemap Resource Arrays	Yes	Yes	No	No	No	No
BC4/BC5 Compression	Yes	Yes	Yes	No	No	No
BC6H/BC7 Compression	Yes	No	No	No	No	No
Alpha-to-coverage	Yes	Yes	Yes	No	No	No
Extended Formats*(BGRA, etc.)	Yes	Optional	Optional	Yes	Yes	Yes
10-bit XR High Color Format	Yes	Optional	Optional	N/A	N/A	N/A
Max Texture Dimension	16384	8192	8192	4096	2048	2048
Max Cubemap Dimension	16384	8192	8192	4096	512	512
Max Volume Extent	2048	2048	2048	256	256	256
Max Texture Repeat	16384	8192	8192	8192	2048	128
Max Anisotropy	16	16	16	16	16	2
Max Primitive Count	2^32	2^32	2^32	1048575	1048575	65535
Max Input Slots	32	32	16	16	16	16
Simultaneous Render Targets	8	8	8	4	1	1
Occlusion Queries	Yes	Yes	Yes	Yes	Yes	No
Separate Alpha Blend	Yes	Yes	Yes	Yes	Yes	No
Mirror Once	Yes	Yes	Yes	Yes	Yes	No
Overlapping Vertex Elements	Yes	Yes	Yes	Yes	Yes	No
Independent Write Masks	Yes	Yes	Yes	Yes	No	No
Instancing	Yes	Yes	Yes	Yes	No	No
Nonpowers-of-2 conditionally*	No	No	No	Yes	Yes	Yes
Nonpowers-of-2 unconditionally**	Yes	Yes	Yes	No	No	No

Getting Device Feature Level

1) Call either D3D11CreateDevice or D3D11CreateDeviceAndSwapChain with *ppDevice* parameter set to NULL

   — or —

   Call ID3D11Device::GetFeatureLevel after device creation

2) Examine value of returned D3D_FEATURE_LEVEL enumeration

```
{
    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,
}
```

```
HRESULT hr = E_FAIL;
D3D_FEATURE_LEVEL FeatureLevel;

hr = D3D11CreateDevice(
    NULL, D3D_DRIVER_TYPE_HARDWARE, NULL, 0, NULL, 0,
    D3D11_SDK_VERSION, NULL, &FeatureLevel, NULL );

if (FAILED(hr))
    return hr;
```

ID3D11Device Interface

Classification	Methods
Creation of resources	CreateBuffer CreateTexture1D / CreateTexture2D / CreateTexture3D
Creation of resource views	CreateShaderResourceView CreateRenderTargetView CreateDepthStencilView CreateUnorderedAccessView
Creation of pipeline state objects	CreateInputLayout CreateRasterizerState CreateDepthStencilState CreateBlendState CreateSamplerState
Creation of shader objects (not exhaustive)	CreateVertexShader CreateGeometryShader CreatePixelShader
Creation and retrieval of device contexts	GetImmediateContext CreateDeferredContext

Device Context

- Contains circumstances or setting in which device is used for rendering
  - Gets and sets pipeline state
  - Supplies resources owned by device to pipeline
  - Generates rendering commands (Draw functions)
- Two types
  - Immediate context and deferred context
  - Both are represented by ID3D11DeviceContext interface

Immediate Context

- Rendering commands directly issued to driver
  - As soon as rendering methods are called
- One and only one immediate context for each device
- The only device context for single-threaded applications
- Obtained along with device creation or from ID3D11Device::GetImmediateContext function

ID3D11DeviceContext

Classification	Methods
Restoring all default settings	ClearState
Clearing resources	ClearRenderTargetView ClearDepthStencilView
Copying and accessing resource data	CopyResource / CopySubresourceRegion UpdateSubresource Map / Unmap
Setting up IA stage	IASetVertexBuffers IASetIndexBuffers IASetInputLayout IASetPrimitiveTopology
Setting shader to device	[VS]SetShader
Setting up shader stages	[VS]SetConstantBuffers [VS]SetShaderResources [VS]SetSamplers CSSetUnorderedAccessViews
Setting up RS stage	RSSetViewports RSSetScissorRects RSSetState
Setting up OM stage	OMSetDepthStencilState OMSetBlendState OMSetRenderTargets
Rendering	Draw / DrawInstanced DrawIndexed / DrawIndexedInstanced

(not exhaustive)

Resource

- Building blocks of scene
- Area in memory accessed by Direct3D pipeline
  - Input geometry, shader resources, textures, …
  - Two categories: buffer and texture
- Several options to create with
  - Fully-typed or typeless
  - Read/write access
  - Accessible to only CPU, GPU, or both
  - Expected usage

Resource Types

Interfaces	Description
ID3D11Buffer	Accesses buffer data
ID3D11Texture1D	Accesses data in 1D texture and 1D texture array
ID3D11Texture2D	Accesses data in 2D texture and 2D texture array
ID3D11Texture3D	Accesses data in 3D texture and 3D texture array

Enumerations	Description
DXGI_FORMAT	Data format
D3D11_USAGE	Expected usage
D3D11_BIND_FLAG	Binding to pipeline stages
D3D11_CPU_ACCESS_FLAG	CPU accessibility
D3D11_RESOURCE_MISC_FLAG	Less common options

Resource Data Format

- DXGI_FORMAT enumeration
  - Includes fully-typed and typeless formats
  - DXGI_FORMAT_R32G32B32A32_FLOAT, DXGI_FORMAT_R8G8B8A8_UINT, DXGI_FORMAT_R32_FLOAT, DXGI_FORMAT_R8_UNORM, DXGI_FORMAT_R32G32B32A32_TYPELESS, …

Format modifiers	Description
_FLOAT	32-bit floating-point format following IEEE 754
_UINT	Unsigned integer (3-bit: 0, 1, 2, 3, 4, 5, 6, 7)
_SINT	Two's-complement signed integer (3-bit: -4, -3, -2, -1, 0, 1, 2, 3)
_UNORM	Unsigned normalized integer (2-bit: 0.0f, 1/3, 2/3, 1.0f)
_SNORM	Signed normalized integer (3-bit: -1.0f, -1.0f, -2/3, -1/3, 0, 1/3, 2/3, 1.0f)
_TYPELESS	Format type is to be resolved when bound to shader

Resource Usage

- D3D11_USAGE enumeration
  - Identifies the way resource is intended to be used
  - Reflects whether resource is accessible by GPU and/or CPU

Resource usage	Description
Default	Most common choice
Immutable	Cannot be changed after initialized
Dynamic	Updates frequently (at least once per frame)
Staging	Supports data transfer (copy) from GPU to CPU

Usage	Default	Dynamic	Immutable	Staging
GPU-Read	✓	✓	✓	✓
GPU-Write	✓			✓
CPU-Read				✓
CPU-Write		✓		✓

Can be bound as	Default	Dynamic	Immutable	Staging
Input to a stage	✓	✓	✓	
Output from a stage	✓			

Creating Buffer

1) Fill D3D11_BUFFER_DESC structure

```
typedef struct D3D11_BUFFER_DESC {
    UINT        ByteWidth;
    D3D11_USAGE Usage;
    UINT        BindFlags;
    UINT        CPUAccessFlags;
    UINT        MiscFlags;
    UINT        StructureByteStride;
} D3D11_BUFFER_DESC;
```

2) Prepare initialization data (D3D11_SUBRESOURCE_DATA) if necessary

3) Call ID3D11Device::CreateBuffer method

```
HRESULT CreateBuffer(
    [in]   const D3D11_BUFFER_DESC *pDesc,
    [in]   const D3D11_SUBRESOURCE_DATA *pInitialData,
    [out]  ID3D11Buffer **ppBuffer
);
```

Resource View

- Pipeline stage interprets resource data using resource view
  - Dictates the role resource is bound to pipeline with
  - Conceptually similar to casting data
  - Specifies data format of typeless resource

Resource interface	Description
ID3D11ShaderResourceView	Access shader resource such as constant buffer, texture, or sampler
ID3D11RenderTargetView	Access texture resource that is used as render-target
ID3D11DepthStencilView	Access texture during depth-stencil testing
ID3D11UnorderedAccessView	Access unordered resource using pixel/compute shader

Creating Render-Target View

```
HRESULT CreateRenderTargetView(
    [in]   ID3D11Resource *pResource,
    [in]   const D3D11_RENDER_TARGET_VIEW_DESC *pDesc,
    [out]  ID3D11RenderTargetView **ppRTView
);
```

```
typedef struct D3D11_RENDER_TARGET_VIEW_DESC {
    DXGI_FORMAT         Format;
    D3D11_RTV_DIMENSION ViewDimension;
    union {
        D3D11_BUFFER_RTV        Buffer;
        D3D11_TEX1D_RTV         Texture1D;
        D3D11_TEX1D_ARRAY_RTV   Texture1DArray;
        D3D11_TEX2D_RTV         Texture2D;
        D3D11_TEX2D_ARRAY_RTV   Texture2DArray;
        D3D11_TEX2DMS_RTV       Texture2DMS;
        D3D11_TEX2DMS_ARRAY_RTV Texture2DMSArray;
        D3D11_TEX3D_RTV         Texture3D;
    };
} D3D11_RENDER_TARGET_VIEW_DESC;
```

Direct3D Helper Library

- D3DX utility library
  - Helper functions for shader, texture, image and mesh
  - Other helper interfaces
  - Mathematical functions
- XNA Math Library
  - Supersedes Xbox Math Library, D3DX 9 math library and D3DX 10 math library

D3DX Math Library

- Declared in D3DX10Math.h
- Structures
  - D3DXCOLOR, D3DXVECTOR3, D3DXMATRIX, D3DXQUATERNION, …
- Functions
  - D3DXVec3Normalize, D3DXVec3Cross, D3DXMatrixTranspose, D3DXMatrixLookAtLH, D3DXMatrixRotationQuaternion, …

```
D3DXVECTOR3 a(1,0,0), b(0,1,0);
D3DXVECTOR3 x, y;
x = a + b;
D3DXVec3Normalize(&y, D3DXVec3Cross(&x, &a, &b));
```

XNA Math Library

- XMVECTOR
  - Proxy for SIMD hardware register
- XMMATRIX
  - Logical grouping of four SIMD hardware registers
- Memory alignment (16-bit alignment)
  - Automatic on stack or data segment
  - Extra care needed for allocation from heap
- Load to and store from XMVECTOR using XMFLOAT3, XMFLOAT4, etc.

XMVECTOR and XMMATRIX

```
XMFLOAT4 f4a(1,0,0,0), f4b(0,1,0,0);
XMVECTOR a = XMLoadFloat4(&f4a);
XMVECTOR b = XMLoadFloat4(&f4b);

XMVECTOR x, y;
x = a + b;
y = XMVector4Normalize(x);

XMFLOAT4 result;
XMStoreFloat4(&result, y);
```

```
XMVECTOR eye, focus, up;
XMMATRIX view = XMMatrixLookAtLH(eye, focus, up);
XMMATRIX tview = XMMatrixTranspose(view);

XMMATRIX world, proj;
XMMATRIX wvp = world * view * proj;
```

Direct3D Programming

- Setting up each pipeline stage
  - Setting up IA stage
  - Setting up shader stages
  - Setting up RS stage
  - Configuring OM stage
- HLSL programming

ID3D11Device Interface

Classification	Methods
Creation of resources	CreateBuffer CreateTexture1D / CreateTexture2D / CreateTexture3D
Creation of resource views	CreateShaderResourceView CreateRenderTargetView CreateDepthStencilView CreateUnorderedAccessView
Creation of pipeline state objects	CreateInputLayout CreateRasterizerState CreateDepthStencilState CreateBlendState CreateSamplerState
Creation of shader objects (not exhaustive)	CreateVertexShader CreateGeometryShader CreatePixelShader
Creation and retrieval of device contexts	GetImmediateContext CreateDeferredContext

ID3D11DeviceContext

Classification	Methods
Restoring all default settings	ClearState
Clearing resources	ClearRenderTargetView ClearDepthStencilView
Copying and accessing resource data	CopyResource / CopySubresourceRegion UpdateSubresource Map / Unmap
Setting up IA stage	IASetVertexBuffers IASetIndexBuffers IASetInputLayout IASetPrimitiveTopology
Setting shader to device	[VS]SetShader
Setting up shader stages	[VS]SetConstantBuffers [VS]SetShaderResources [VS]SetSamplers CSSetUnorderedAccessViews
Setting up RS stage	RSSetViewports RSSetScissorRects RSSetState
Setting up OM stage	OMSetDepthStencilState OMSetBlendState OMSetRenderTargets
Rendering	Draw / DrawInstanced DrawIndexed / DrawIndexedInstanced

(not exhaustive)

More Interfaces

Classification	Interfaces
Resource	ID3D11Buffer ID3D11Texture1D / ID3D11Texture2D / ID3D11Texture3D
Resource view	ID3D11ShaderResourceView ID3D11UnorderedAccessView ID3D11RenderTargetView ID3D11DepthStencilView
Pipeline state	ID3D11InputLayout ID3D11SamplerState ID3D11RasterizerState ID3D11BlendState ID3D11DepthStencilState

Setting Up IA Stage

1) Create input buffers
   - One or more vertex buffers
   - An index buffer (optional)
2) Create input-layout object
3) Bind objects to IA stage
4) Specify primitive topology

Setting Up Shader Stages

1) Create shader resources
2) Create shader resource view
3) Bind the view

1) Create constant buffer
2) Bind the buffer

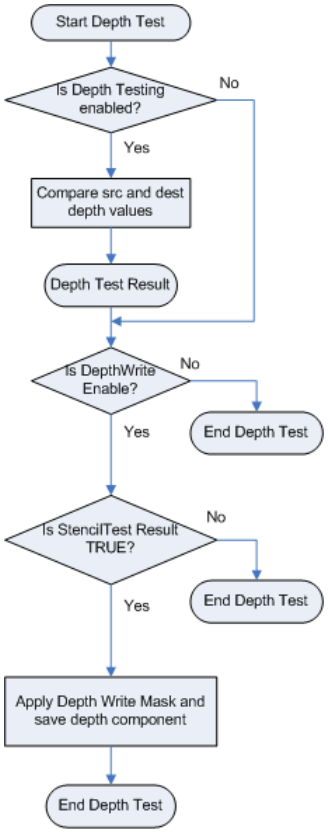
1) Create sampler state
2) Bind the state

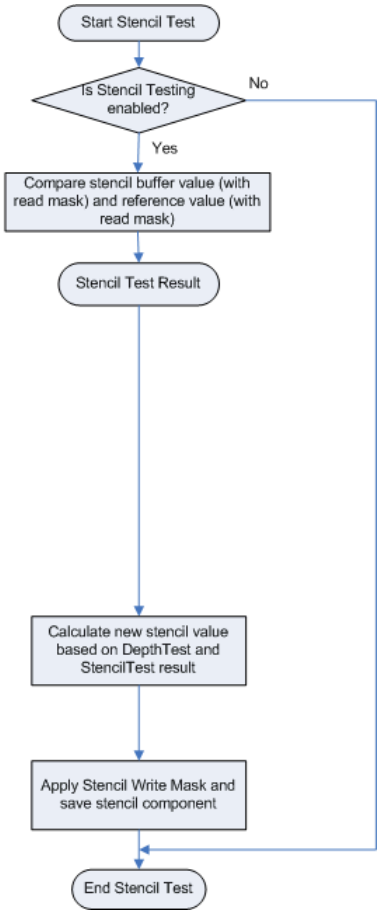
Setting Up RS Stage

1) Set viewport
   - Viewport maps vertex positions (in clip space) into render-target positions
2) Set scissor rectangle
   - Pixels outside the rectangle are discarded
3) Create rasterizer state
   - Fill mode, cull mode, convention of determining front-facing triangle, multisampling, anti-aliasing, …
4) Set rasterizer state to RS stage

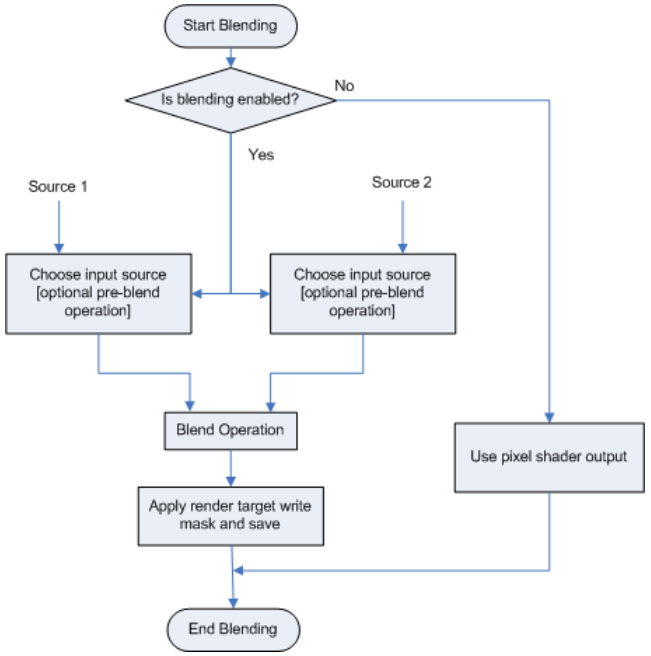
How OM Stage Works

- Depth-stencil testing





- Blending



Depth-Stencil Functionality

1) Create depth-stencil resource (2D texture)
   - DXGI_FORMAT_D32_FLOAT, DXGI_FORMAT_D24_UNORM_S8_UINT, …
2) Create depth-stencil resource view
3) Create depth-stencil state
4) Bind depth-stencil data to OM stage
   - ID3D11DeviceContext::OMSetRenderTargets function

```
void OMSetRenderTargets(
    [in]  UINT NumViews,
    [in]  ID3D11RenderTargetView *const **ppRenderTargetViews,
    [in]  ID3D11DepthStencilView *pDepthStencilView
);
```

Blending Functionality

1) Create blend state
2) Bind blend state

Dealing with Resource Data

- Resource usage affects accessibility
  - GPU-accessible: default or immutable (non-mappable)
  - CPU-accessible: dynamic or staging (mappable)
  - Copying from one to another may be necessary
- Mappable resources
  - Process of giving CPU access to underlying memory
  - Copying among non-mappable resources is very fast
  - Mapping and copying among mappable resources should be done with care

Copying Resource Data

Method	Description
CopyResource CopySubresourceRegion	Copying data from one resource to another
UpdateSubresource	Copying data from memory to non-mappable resource

```
// Create constant buffer
D3D11_BUFFER_DESC bd;
ZeroMemory( &bd, sizeof(bd) );
bd.Usage = D3D11_USAGE_DEFAULT;
bd.ByteWidth = sizeof( XMMATRIX );
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
bd.CPUAccessFlags = 0;
hr = g_pd3dDevice->CreateBuffer( &bd, NULL, &pCBMatView );
if( FAILED( hr ) )
    return hr;

// Update constant buffer
XMMATRIX matView;
XMVECTOR vecEye = XMVectorSet( 0.0f, 1.0f, -5.0f, 0.0f );
XMVECTOR vecAt = XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );
XMVECTOR vecUp = XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );
matView = XMMatrixLookAtLH( Eye, At, Up );
matView = XMMatrixTranspose( matView );
pImmediateContext->UpdateSubresource( pCBMatView, 0, NULL, &matView, 0, 0 );
```

Accessing Resource Data

- Mapping
  - Obtains CPU access to underlying resource data
  - Denies GPU access to the resource

```
// Create constant buffer
D3D11_BUFFER_DESC bd;
ZeroMemory( &bd, sizeof(bd) );
bd.Usage = D3D11_USAGE_DYNAMIC;
bd.ByteWidth = sizeof( XMMATRIX );
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
bd.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
hr = g_pd3dDevice->CreateBuffer( &bd, NULL, &pCBMatView );
if( FAILED( hr ) )
    return hr;

// Update constant buffer
D3D11_MAPPED_SUBRESOURCE MappedResource;
hr = pd3dImmediateContext->Map( pCBMatView , 0, D3D11_MAP_WRITE_DISCARD, 0, &MappedResource );
XMMATRIX* pMatView = ( XMMATRIX* )MappedResource.pData;
*pMatView = XMMatrixTranspose( matView );
pd3dImmediateContext->Unmap( g_pcbPSPerFrame, 0 );
```

Programming with HLSL

- Compiling shader functions
  - Creating blob object
  - Creating shader object
- Intrinsic functions
- Semantics
- Programming Direct3D with HLSL
  - Input layout
  - Constant buffer
  - Texture and sampler

Compiling Shader Functions

- Example: creating vertex shader object from file

```
HRESULT hr;

DWORD dwShaderFlags = D3DCOMPILE_ENABLE_STRICTNESS;
#if defined( DEBUG ) || defined( _DEBUG )
dwShaderFlags |= D3DCOMPILE_DEBUG;
#endif

ID3DBlob* pVSBlob;
hr = D3DX11CompileFromFile(
    _T("test.hlsl"), NULL, NULL, "VS", "vs_4_0", dwShaderFlags, 0, NULL,
    ppBlobOut, NULL, NULL );
if( FAILED(hr) )
    return hr;

ID3D11VertexShader* pVertexShader;
hr = pd3dDevice->CreateVertexShader(
    pVSBlob->GetBufferPointer(), pVSBlob->GetBufferSize(), NULL,
    &pVertexShader );
if( FAILED(hr) )
{
    pVSBlob->Release();
    return hr;
}
```

Intrinsic Functions

- Provides commonly used operations
- Efficient and tested — use them if available

- abs, acos, atan, atan2, ceil, clamp, cos, cosh, cross, degrees, determinant, distance, dot, exp, exp2, floor, fmod, frac, frexp, isfinite, isinf, isnan, length, lerp, lit, log, log10, log2, mad, max, min, modf, mul, noise, normalize, pow, radians, rcp, reversebits, round, rsqrt, saturate, sign, sin, sincos, sinh, sqrt, step, tan, tanh, transpose, trunc, …

```
matrix World;
matrix View;
matrix Proj;

float4 VS( float4 Pos )
{
    float4 Out;
    Out = mul( Pos, World );
    Out = mul( Out, View );
    Out = mul( Out, Proj );
    return Out;

    // Out = Pos * World * View * Proj
}
```

Semantics

- String attached to shader input/output
  - Conveys information about intended use of parameter
  - Required on all variables passed between shader stages

```
matrix World, View, Proj;

float4 VS( float4 Pos : POSITION ) : SV_Position
{
    float4 Out;
    Out = mul( Pos, World );
    Out = mul( Out, View );
    Out = mul( Out, Proj );
    return Out;

    // Out = Pos * World * View * Proj
}
```

```
matrix World, View, Proj;

struct VS_INPUT
{
    float4 Pos : POSITION;
};
struct VS_OUTPUT
{
    float4 Pos : SV_Position;
};

VS_OUTPUT VS( VS_INPUT In )
{
    VS_OUTPUT Out = (VS_OUTPUT)0;
    Out.Pos = mul( In.Pos, World );
    Out.Pos = mul( Out.Pos, View );
    Out.Pos = mul( Out.Pos, Proj );
    return Out;
}
```

Vertex Shader Semantics

Input	Description	Type
POSITION	Vertex position in object space	float4
NORMAL	Normal vector of vertex	float4
COLOR	Vertex color	float4
TEXCOORD	Texture coordinates associated with vertex	float4

Output	Description	Type
POSITION	Vertex position in homogeneous space	float4
NORMAL	Normal vector of vertex	float4
COLOR	Vertex color	float4
TEXCOORD	Texture coordinates associated with vertex	float4

(not exhaustive)

Pixel Shader Semantics

Input	Description	Type
VPOS	Pixel location in screen space	float2
COLOR	Interpolated color	float4
TEXCOORD	Interpolated texture coordinates	float4

Output	Description	Type
COLOR	Output color	float4
DEPTH	Output depth	float

(not exhaustive)

System-Value Semantics

- Available in Direct3D 10 and above
- Begins with ‘SV_’ prefix
  - SV_ClipDistance, SV_CullDistance, SV_Coverage, SV_Depth, SV_IsFrontFace, SV_Position, SV_RenderTargetArrayIndex, SV_SampleIndex, SV_Target, SV_ViewportArrayIndex, SV_InstanceID, SV_PrimitiveID, SV_VertexID, …
- Some map directly to Direct3D semantics

Direct3D 10 Semantic	**Direct3D 9 Equivalent Semantic**
SV_Position	POSITION
SV_Target	COLOR
SV_Depth	DEPTH

Setting Input Layout

```
// Define the input layout
D3D11_INPUT_ELEMENT_DESC layout[] =
{
    { "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 },
};
UINT numElements = ARRAYSIZE( layout );

// Create the input layout
hr = pd3dDevice->CreateInputLayout( layout, numElements, pVSBlob->GetBufferPointer(),
    pVSBlob->GetBufferSize(), &pVertexLayout );
```

```
struct VS_OUTPUT
{
    float4 Pos : SV_POSITION;
    float4 Color : COLOR0;
};

VS_OUTPUT VS( float4 Pos : POSITION, float4 Color : COLOR )
{
    VS_OUTPUT output = (VS_OUTPUT)0;
    output.Pos = mul( Pos, World );
    output.Pos = mul( output.Pos, View );
    output.Pos = mul( output.Pos, Proj );
    output.Color = Color;
    return output;
}
```

Using Constant Buffer

```
struct ConstantBuffer
{
    XMMATRIX mWorld;
    XMMATRIX mView;
    XMMATRIX mProj;
};

// Create
D3D11_BUFFER_DESC bd;
ZeroMemory( &bd, sizeof(bd) );
bd.Usage = D3D11_USAGE_DEFAULT;
bd.ByteWidth = sizeof(ConstantBuffer);
bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
hr = pd3dDevice->CreateBuffer(
    &bd, NULL, &pConstantBuffer );
if( FAILED( hr ) )
    return hr;

// Update
ConstantBuffer cb;
cb.mWorld = XMMatrixTranspose( g_World );
cb.mView = XMMatrixTranspose( g_View );
cb.mProj = XMMatrixTranspose( g_Proj );
pImmediateContext->UpdateSubresource(
    pConstantBuffer, 0, NULL, &cb, 0, 0 );

// Bind
pImmediateContext->VSSetConstantBuffers(
    0, 1, &pConstantBuffer );
```

Note: packing in 128-bit registers

```
cbuffer ConstantBuffer : register( b0 )
{
    matrix World;
    matrix View;
    matrix Proj;
}

struct VS_OUTPUT
{
    float4 Pos : SV_POSITION;
    float4 Color : COLOR0;
};

VS_OUTPUT VS( float4 Pos : POSITION, float4 Color : COLOR )
{
    VS_OUTPUT output = (VS_OUTPUT)0;
    output.Pos = mul( Pos, World );
    output.Pos = mul( output.Pos, View );
    output.Pos = mul( output.Pos, Proj );
    output.Color = Color;
    return output;
}
```

Using Texture and Sampler

```
ID3D11ShaderResourceView* pTextureSRV = NULL;
ID3D11SamplerState* pSamplerLinear = NULL;

// Create shader resource view
hr = D3DX11CreateShaderResourceViewFromFile(
    pd3dDevice, _T("seafloor.dds"), NULL, NULL,
    &pTextureSRV, NULL );
if( FAILED( hr ) )
    return hr;

// Create sampler
D3D11_SAMPLER_DESC sampDesc;
ZeroMemory( &sampDesc, sizeof(sampDesc) );
sampDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
sampDesc.AddressU = D3D11_TEXTURE_ADDRESS_BORDER;
sampDesc.AddressV = D3D11_TEXTURE_ADDRESS_BORDER;
sampDesc.AddressW = D3D11_TEXTURE_ADDRESS_BORDER;
sampDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;
sampDesc.MinLOD = 0;
sampDesc.MaxLOD = D3D11_FLOAT32_MAX;
hr = pd3dDevice->
    CreateSamplerState( &sampDesc, &pSamplerLinear );
if( FAILED( hr ) )
    return hr;

// Bind
pImmediateContext->
    PSSetShaderResources( 0, 1, &pTextureSRV );
pImmediateContext->
    PSSetSamplers( 0, 1, &pSamplerLinear );
```

```
Texture2D txDiffuse : register( t0 );
SamplerState samLinear : register( s0 );

struct VS_INPUT
{
    float4 Pos : POSITION;
    float2 Tex : TEXCOORD0;
};

struct PS_INPUT
{
    float4 Pos : SV_Position;
    float2 Tex : TEXCOORD0;
};

PS_INPUT VS( VS_INPUT input )
{
    PS_INPUT output = (PS_INPUT)0;
    output.Pos = mul( input.Pos, World );
    output.Pos = mul( output.Pos, View );
    output.Pos = mul( output.Pos, Projection );
    output.Tex = input.Tex;

    return output;
}

float4 PS( PS_INPUT input ) : SV_Target
{
    return txDiffuse.Sample( samLinear, input.Tex );
}
```