DirectX Programming #1

Kang, Seongtae

Computer Graphics, 2008 Spring

Contents

Installation and Settings

Computer Graphics, 2008 Spring

Installation

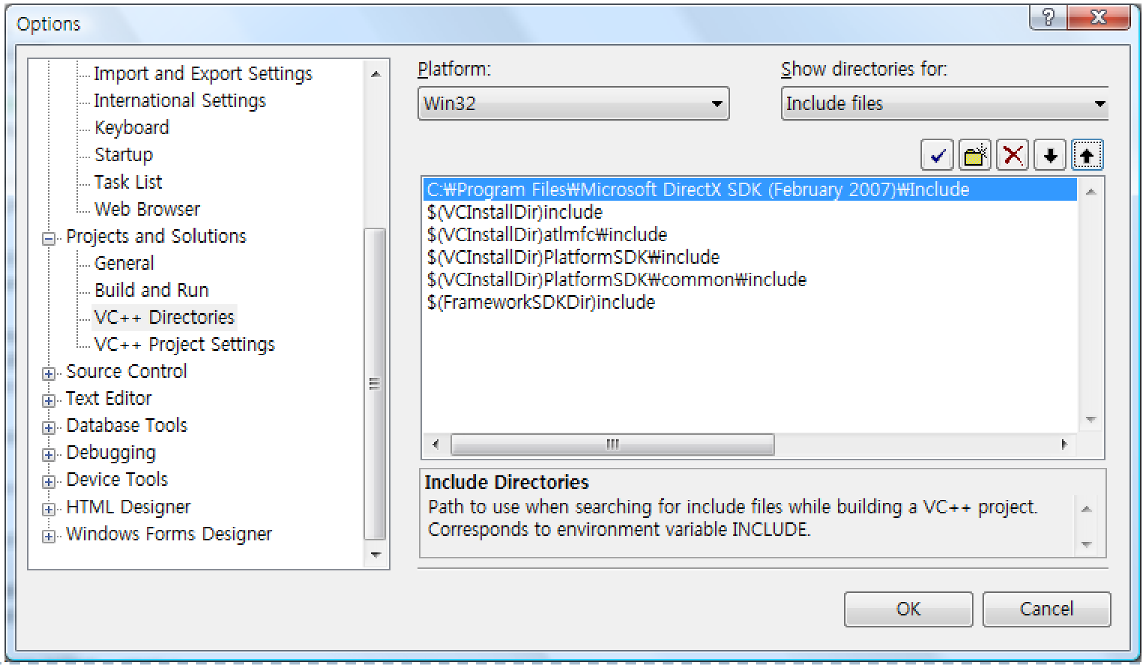
- Download SDK
  - Latest version : March 2008 (442MB)
    - http://www.microsoft.com/downloads/details.aspx?FamilyID=572be8a6-263a-4424-a7fe-69cff1a5b180&DisplayLang=en
  - Recommended version : April 2007 (441MB)
    - http://www.microsoft.com/downloads/details.aspx?FamilyID=86cf7fa2-e953-475c-abde-f016e4f7b61a&DisplayLang=en
    - Contains samples and documentations for MDX and the sample browser

DirectX Layers

- Native DirectX
  - COM-based API
  - Supports C++ and Visual Basic
- Managed DirectX
  - Microsoft .Net wrapper for DirectX API
  - Supports C#, VB.Net, C++.Net, …
  - Obsoleted
    - Managed DirectX 9 for .Net framework 1.1 is the last version
- XNA
  - DirectX-based API for managed platform
  - Highly abstracted layer
  - Game and multimedia oriented API
    - Common API for PC and XBox 360

Settings

- Visual Studio settings
  - Register DirectX include and library directories
    - <Menu> Tools → Options
      → Projects and Solutions → VC++ Directories
    - Move the DirectX directories to the top
      - Visual Studio contains old-version DX include and libraries



Platform: Win32

Show directories for: Include files

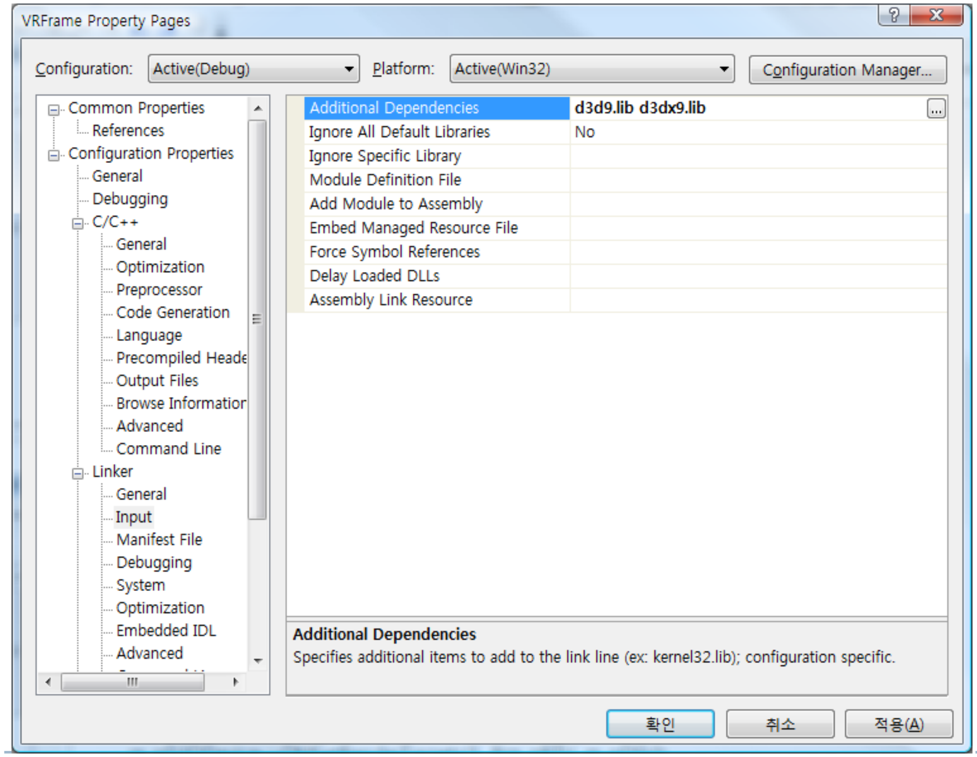
```
C:₩Program Files₩Microsoft DirectX SDK (February 2007)₩Include
$(VCInstallDir)include
$(VCInstallDir)atlmfc₩include
$(VCInstallDir)PlatformSDK₩include
$(VCInstallDir)PlatformSDK₩common₩include
$(FrameworkSDKDir)include
```

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

- DirectX SDK includes both x86 and x64 libraries. Set the proper directory for your environment.

Settings

- Project settings
  - Add library references
    - <Menu> Project → [Project] properties → Linker → Input → Additional Dependencies


VRFrame Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

Additional Dependencies	d3d9.lib d3dx9.lib
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

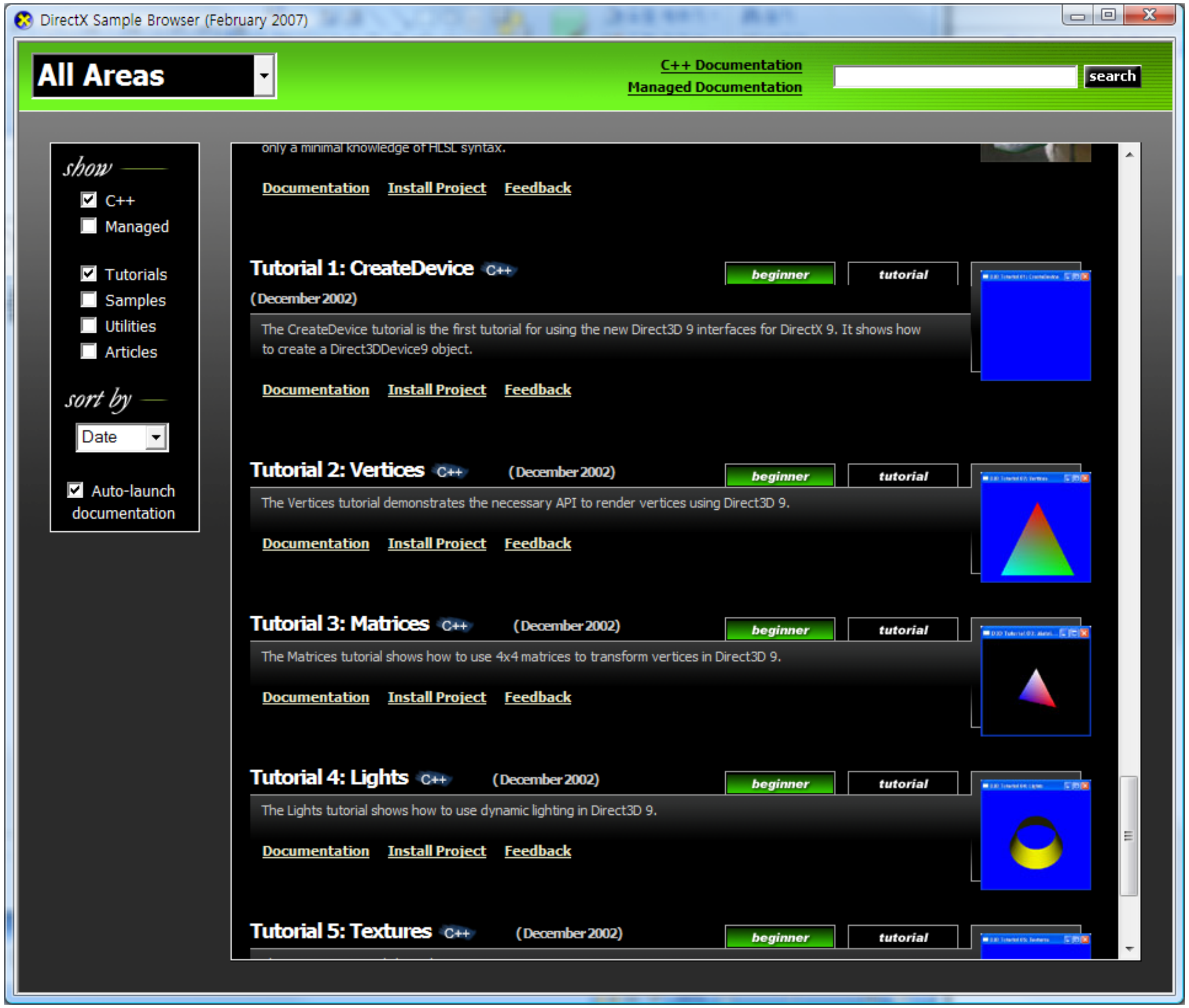
Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

- D3DX is an optional D3D library, but it's almost mandatory. (supports vectors, matrices, transforms, …)

Sample and tutorial codes

- Sample codes
  - [DX directory]/Samples/C++/Direct3D
- Tutorial codes
  - [DX directory]/Samples/C++/Direct3D/Tutorials

Using Sample Browser



DirectX Sample Browser (February 2007)

All Areas

C++ Documentation
Managed Documentation

search

show

- [✓] C++
- [ ] Managed
- [✓] Tutorials
- [ ] Samples
- [ ] Utilities
- [ ] Articles

sort by

Date

[✓] Auto-launch documentation

only a minimal knowledge of HLSL syntax.

Documentation Install Project Feedback

Tutorial 1: CreateDevice C++ (December 2002) beginner tutorial

The CreateDevice tutorial is the first tutorial for using the new Direct3D 9 interfaces for DirectX 9. It shows how to create a Direct3DDevice9 object.

Documentation Install Project Feedback

Tutorial 2: Vertices C++ (December 2002) beginner tutorial

The Vertices tutorial demonstrates the necessary API to render vertices using Direct3D 9.

Documentation Install Project Feedback

Tutorial 3: Matrices C++ (December 2002) beginner tutorial

The Matrices tutorial shows how to use 4x4 matrices to transform vertices in Direct3D 9.

Documentation Install Project Feedback

Tutorial 4: Lights C++ (December 2002) beginner tutorial

The Lights tutorial shows how to use dynamic lighting in Direct3D 9.

Documentation Install Project Feedback

Tutorial 5: Textures C++ (December 2002) beginner tutorial

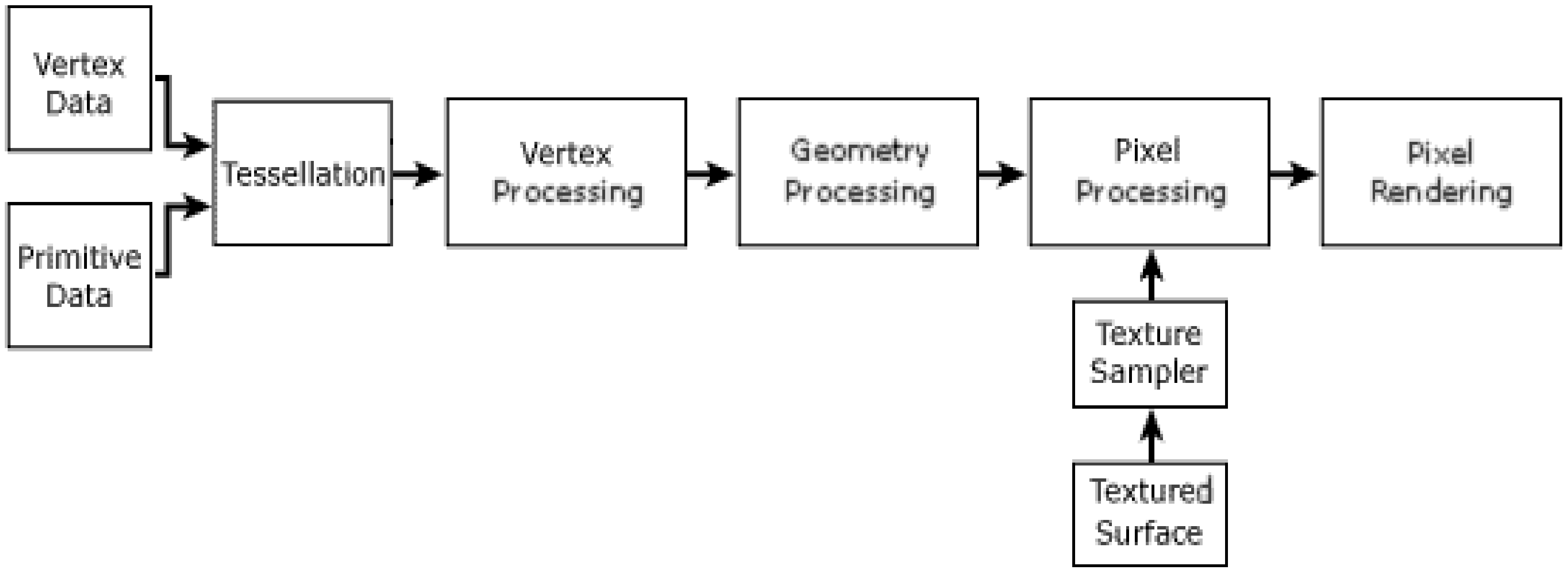
- Some old versions of DirectX SDK do not contain the sample browser.

Introduction to Direct3D 9

Computer Graphics, 2008 Spring

Direct3D Graphics Pipeline

- Overview



Direct3D Graphics Pipeline

- Vertex processing
  - Vertex transformation : World, View, Projection
  - Fully programmable
- Geometry processing
  - Clipping, back face culling
  - Rasterization
- Pixel processing
  - Shading and texturing for each rasterized pixel
  - Fully programmable
- Pixel rendering
  - Alpha/depth/stencil testing
  - Alpha blending

Initializing Direct3D

Computer Graphics, 2008 Spring

Creating Devices

- Initializing Direct3D
  - After the windows is created
  - Creating Direct3D
  - Creating device
    - Setting presentation parameters

```
LPDIRECT3D9      g_pD3D = NULL;
LPDIRECT3DDEVICE9  g_g_pd3dDevice;

if( NULL == ( g_pD3D = Direct3DCreate9( D3D_SDK_VERSION ) ) )
        return E_FAIL;

D3DPRESENT_PARAMETERS d3dpp;
ZeroMemory( &d3dpp, sizeof(d3dpp) );
d3dpp.Windowed = TRUE;
d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;

if( FAILED( pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
                        D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                        &d3dpp, &g_g_pd3dDevice ) ) )
                                                :
```

Basic Rendering routine

- Clear
  - Clear back buffer, depth buffer, and stencil buffer
- BeginScene / EndScene pair
  - Compose a rendering block
- Present
  - Presents the back buffer

```
    g_g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,255), 1.0f,
0 );

    if( SUCCEEDED( g_g_pd3dDevice->BeginScene() ) )
    {
        // Render here
        g_g_pd3dDevice->EndScene();
    }

    g_g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
```

Locating the Rendering Routine

- WM_PAINT handler
  - When the rendered scene is static
  - Minimum load for rendering
  - Refresh by WM_PAINT message
    - InvalidateRect

```
LRESULT WINAPI MsgProc( HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
    switch( msg )
    {
                                        :
                                        :
     case WM_PAINT:
          Render();
          ValidateRect( hWnd, NULL );
          return 0;
                                        :
                                        :
```

Locating the Rendering Routine

- Message Loop
  - When the rendered scene changes continuously
    - e.g. time-series animation
  - Infinite loop
    - If the rendering routine is heavy, the application drains system resource

```
MSG msg = {0};
do        // message loop
{
        if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))        // if there's a delivered message
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
        else                                          // render when there's no delivered message
        {
                Render();
        }
} while(WM_QUIT != msg.message);      // until 'quit the application' message is delivered
```

Shutting Down

- When the D3D application ends
  - Usually located in WM_DESTROY message hander
- Clear objects
  - Release all created objects

```
case WM_DESTROY:
  if( g_g_pd3dDevice != NULL)
     g_g_pd3dDevice->Release();
  if( g_pD3D != NULL)
     g_pD3D->Release();
```

Result of Tutorial 1

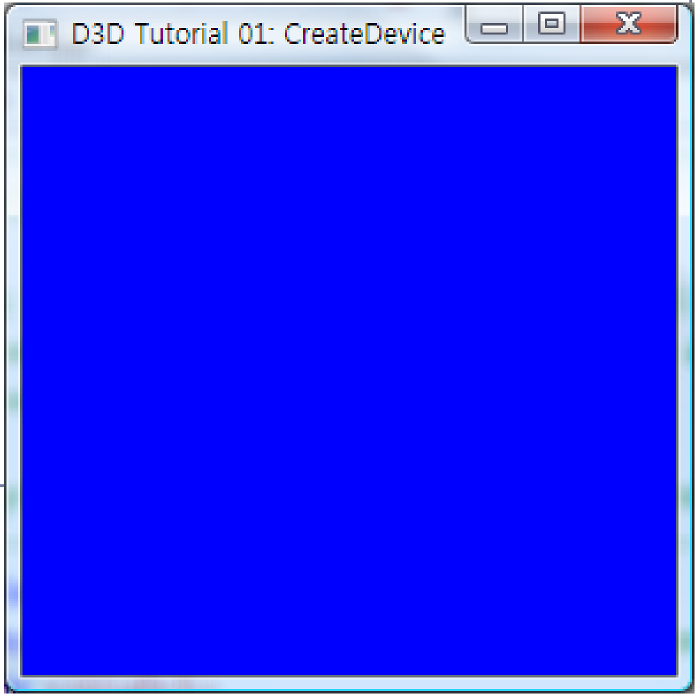
```
VOID Render()
{
    if( NULL == g_pd3dDevice )
        return;

    // Clear the backbuffer to a blue color
    g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,255), 1.0f, 0 );

    // Begin the scene
    if( SUCCEEDED( g_pd3dDevice->BeginScene() ) )
    {
        // Rendering of scene objects can happen here

        // End the scene
        g_pd3dDevice->EndScene();
    }

    // Present the backbuffer contents to the display
    g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
}
```



Rendering Vertices

Computer Graphics, 2008 Spring

Defining a Vertex Type

- Two methods
  - Vertex declaration
    - Flexible declaration
    - Complicated code
  - FVF
    - Combination of pre-defined features
    - Fixed order
    - Simple

Defining a Vertex Type

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZRHW|D3DFVF_DIFFUSE)
```

- FVF
  - Description of structure for a vertex
  - OR Combination of predefined FVF values
  - Element order
    - Position – Normal – Diffuse and specular term – Texture coordinates
  - e.g.
    - D3DFVF_XYZRHW | D3DFVF_DIFFUSE
      - transformed position with diffuse
    - D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_DIFFUSE
      - 3D position, normal, and diffuse
    - D3DFVF_XYZ | D3DFVF_TEX2
      - 3D position and two 2D texture coordinates
    - D3DFVF_XYZ | D3DFVF_TEX1 | D3DFVF_TEXCOORDSIZE3(0)
      - 3D position with one 3D texture coordinate

Defining a Vertex Type

- Vertex structure
  - Just for comfortable coding
  - You can even use just byte array
- Size of elements
  - Position and texture coordinate : (# of dim.) x FLOAT
  - Diffuse/specular color : DWORD (D3DCOLOR_ARGB)
- Element ordering
  - Defining order of elements in the structure must be matched to FVF element ordering

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZRHW|D3DFVF_DIFFUSE)

struct CUSTOMVERTEX
{
    FLOAT x, y, z, rhw; // The transformed position for the vertex.
    DWORD color;        // The vertex color.
};
```

Preparing the Vertex Buffer

- Vertex Buffer
  - Holds vertices to render
  - Reside in the video memory

- Creating a vertex buffer

```
HRESULT CreateVertexBuffer(UINT Length, DWORD Usage, DWORD FVF, D3DPOOL Pool,
                           IDirect3DVertexBuffer9** ppVertexBuffer,  HANDLE* pSharedHandle);
```

- Length : Size of the vertex buffer, in bytes
- Usage : Usage of the resource; Usually 0 for vertex buffer
- FVF : FVF to use for this vertex buffer
- Pool : description of the memory class that holds the buffer. See D3DPOOL
  Usually D3DPOOL_DEFAULT or D3DPOOL_MANAGED for vertex buffer
- ppVertexBuffer : pointer of the vertex buffer object
- pSharedHandle : Not used

- Objects created with D3DPOOL_MANAGED need not be released explicitly after use

Preparing the Vertex Buffer

- Locking and Unlocking
  - Lock
    - Lock the buffer and obtains a pointer to the memory
    - CPU can access the locked resource buffer
    - GPU memory → CPU memory (download)
    - When locking for writing, setting D3DLOCK_DISCARD flag helps performance (no downloading)
  - Unlock
    - CPU memory → GPU memory (upload)
    - For reading only, locking with D3DLOCK_READONLY flag helps unlocking performance (no uploading)

```
HRESULT Lock(UINT OffsetToLock, UINT SizeToLock, VOID ** ppbData, DWORD Flags);
```

- OffsetToLock : Offset into the vertex data to lock, in bytes; 0 for locking the entire buffer
- SizeToLock : Size of the vertex data to lock, in bytes; 0 for locking the entire buffer
- ppbData : VOID* pointer to a memory buffer
- Flags : Locking flags; See D3DLOCK on the SDK document

Preparing the Vertex Buffer

```
LPDIRECT3DVERTEXBUFFER9 g_pVB;

CUSTOMVERTEX vertices[] =
{
    { 150.0f,  50.0f, 0.5f, 1.0f, 0xffff0000, }, // x, y, z, rhw, color
    { 250.0f, 250.0f, 0.5f, 1.0f, 0xff00ff00, },
    {  50.0f, 250.0f, 0.5f, 1.0f, 0xff00ffff, },
};

if( FAILED( g_pd3dDevice->CreateVertexBuffer( 3*sizeof(CUSTOMVERTEX),
         0 /*Usage*/, D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, &g_pVB, NULL ) ) )
    return E_FAIL;

VOID* pVertices;
if( FAILED( g_pVB->Lock( 0, sizeof(vertices), (void**)&pVertices, 0 ) ) )
    return E_FAIL;

memcpy( pVertices, vertices, sizeof(vertices) );

g_pVB->Unlock();
```

Rendering

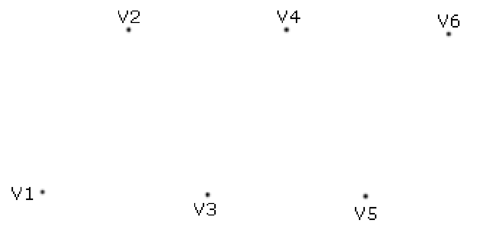
- Attaching vertex buffers to the rendering pipeline

```
HRESULT SetStreamSource(UINT StreamNumber, IDirect3DVertexBuffer9 * pStreamData,
                                                    UINT OffsetInBytes, UINT Stride);
```

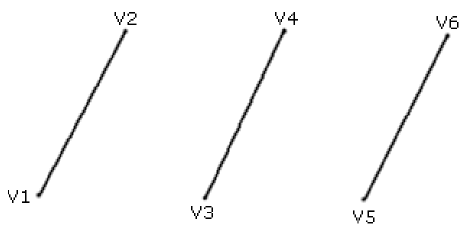
- StreamNumber : Specifies the data stream
- pStreamData : Pointer to a vertex buffer
- OffsetInBytes : Offset from the beginning of the stream to the beginning of the vertex data, in bytes
  Usually 0; (Actually, many hardwares don't support VB offset features)
- Stride : Stride, i.e. size of a component, in bytes

Rendering

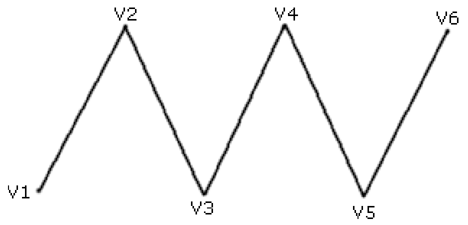
- Primitives



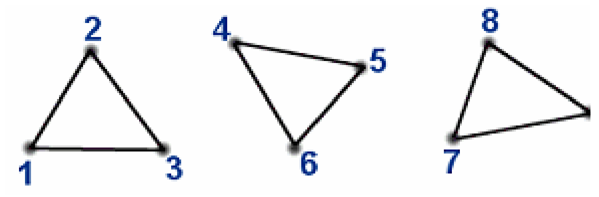
D3DPT_POINTLIST



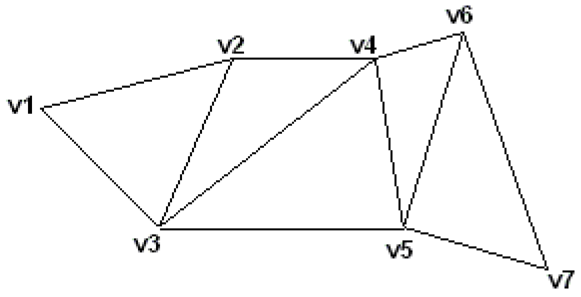
D3DPT_LINELIST



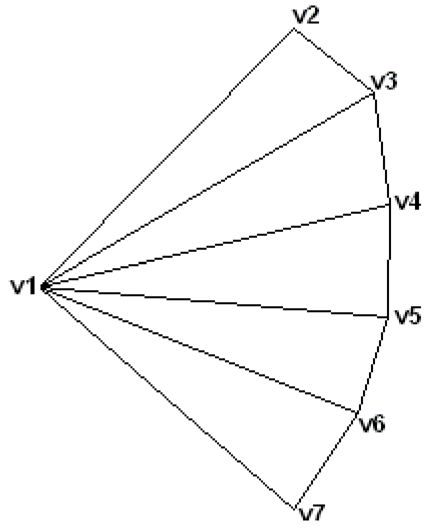
D3DPT_LINESTRIP



D3DPT_TRIANGLELIST



D3DPT_TRIANGLESTRIP



D3DPT_TRIANGLEFAN

Rendering

- Drawing primitives

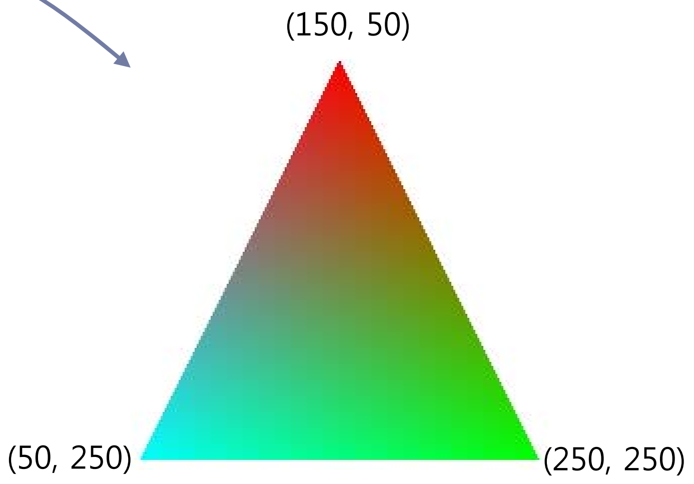
```
HRESULT DrawPrimitive(D3DPRIMITIVETYPE PrimitiveType, UINT StartVertex, UINT PrimitiveCount);
```

- PrimitiveType : primitive type
- StartVertex : Index of the first vertex to load
- PrimitiveCount : Number of primitives to render

```
DrawPrimitive( D3DPT_TRIANGLELIST, 0, 1 );
```

D3DFVF_XYZRHW|D3DFVF_DIFFUSE

X	Y	Z	rhw	Diffuse
150.0	50.0	0.5	1.0	0xFFFF0000
250.0	250.0	0.5	1.0	0xFF00FF00
50.0	250.0	0.5	1.0	0xFF00FFFF



Result of tutorial 02

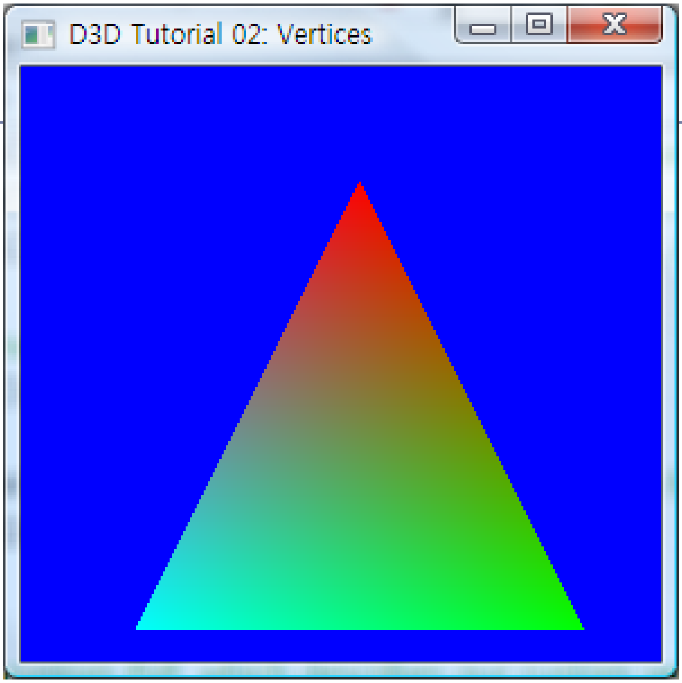
```
g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,255), 1.0f, 0L );
g_pd3dDevice->BeginScene();

g_pd3dDevice->SetStreamSource( 0, g_pVB, 0, sizeof(CUSTOMVERTEX) );

g_pd3dDevice->SetFVF( D3DFVF_CUSTOMVERTEX );

g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLELIST, 0, 1 );

g_pd3dDevice->EndScene();
g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
```


D3D Tutorial 02: Vertices