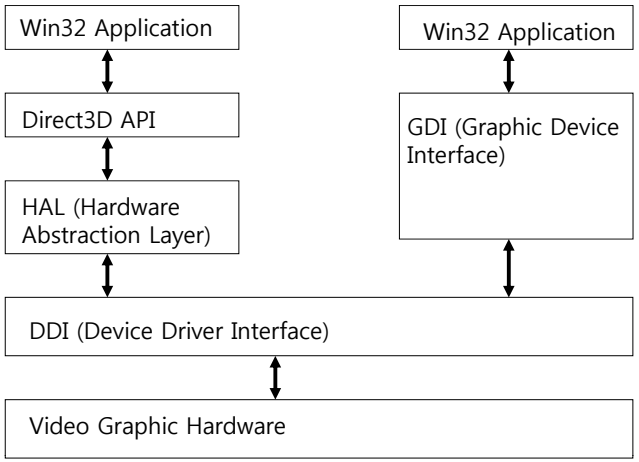
# Direct3D Initialization

305890
Spring 2010
3/26/2010
Kyoung Shin Park

## Initializing Direct3D

- Direct3D overview
- COM
- Direct3D Initialization

## DDI, HAL, GDI




## Direct3D Overview

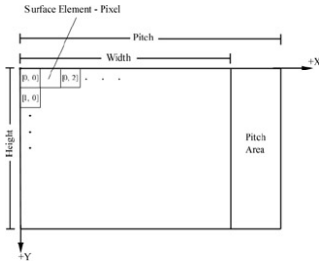
- Direct3D
  - 3D 가속 하드웨어를 이용해 3D 세계를 표현할 수 있도록 하는 저수준 그래픽 API
  - Application->Direct3D->HAL->Graphic device
    - HAL (Hardware Abstraction Layer)는 그래픽 카드마다 다른 처리방식을 해결. HAL에 포함되어 있는 기능들은 장치마다 다름.
    - HAL에서 구현하지 않은 Direct3D함수를 호출하면 오류 발생
- REF (Reference Rasterizer)
  - 그래픽 장치에서는 지원하지 않는 Direct3D 기능을 제공하기 위함.
  - 모든 Direct3D API를 소프트웨어로 emulation하는 REF를 제공함.
  - REF 장치는 개발 목적으로 제공됨. DirectX SDK에만 포함되어 있음. 최종 사용자에게는 배포 불가.
- D3DDEVTYPE
  - HAL장치 – D3DDEVTYPE_HAL
  - REF 장치 – D3DDEVTYPE_REF

# COM

- COM (Component Object Model)
  - COM은 DirectX를 프로그래밍 언어에 독립적으로 만들어주고 하위 호환성을 갖출 수 있게 하는 기술
  - COM object를 interface라고 부르며, C++ class와 비슷하게 이용
  - 모든 COM interface는 IUnknown COM interface에서 기능을 상속받음
    - COM object는 자신의 메모리 관리를 스스로 수행함
    - C++ new 키워드가 아닌 다른 COM interface의 method나 특수한 함수를 통해 COM interface의 pointer를 얻음
    - 마찬가지로, COM interface를 이용하는 작업이 모두 끝나면 C++ delete 키워드가 아니라 interface의 Release를 호출함
  - COM interface는 접두어로 "I"를 붙여서 명명
    - E.g., 표면을 나타내는 COM 인터페이스의 이름 IDirect3DSurface9

# Direct3D Basics - Surface

- Surface
  - A surface is a matrix of pixels that Direct3D uses primarily to store 2D image data.
  - *The width and height of a surface are measured in pixels.*
  - Pitch – the width of a surface in byte
    - **pitch = width * sizeof(pixelFormat)**




# Direct3D Basics - Surface

- IDirect3DSurface9
  - We describe surfaces with IDirect3DSurface9 interface, methods for reading and writing data directly to a surface as well as a surface.
  - IDirect3DSurface9 Method
    - LockRect – to obtain a pointer to the surface memory
    - UnlockRect – after you have called LockRect and are done accessing unlock the surface by calling this method
    - GetDesc – to obtain a description of the surface (in structure D3DSURFACE_DESC structure)
    - SetDesc – to set a description of the surface

```
typedef struct _D3DLOCKED_RECT {
        INT Pitch; // surface pitch
        void *pBits; // pointer to the start of the surface memory
} D3DLOCKED_RECT;
```

# Direct3D Basics - Surface

- E.g. Locking a surface and writing to each pixel to red

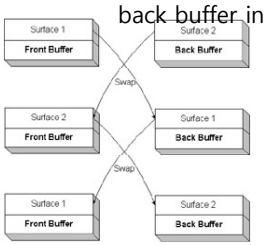
```
// _surface가 IDirect3DSurface9 interface로의 pointer라고 가정
// 32-bit pixel format을 이용한다고 가정

// surface 정보를 얻음
D3DSURFACE_DESC surfaceDesc;
_surface->GetDesc(&surfaceDesc);
// surface pixel data로의 pointer를 얻음
D3DLOCKED_RECT lockedRect;
_surface->LockRect(&lockedRect, 0, 0);
// surface의 각 pixel을 대상으로 반복하여 pixel을 빨간색으로 지정
DWORD* imageData = (DWORD*) lockedRect.pBits;
for (int i=0; i<surfaceDesc.Height; i++) {
    for (int j=0; j<surfaceDesc.Width; j++) {
      // pitch는 byte 단위이며 DWORD당 4bytes이므로 4로 나눔
      int index = i * lockedRect.Pitch / 4 + j;
      imageData[index] = 0xffff0000;          // red
    }
}
_surface->UnlockRect();
```

## Direct3D Basics – Pixel format

- Pixel format
  - We need to specify the pixel format when we create surfaces in Direct3D.
- D3DFORMAT
  - D3DFMT_R8G8B8
    - 24-bit pixel format. 8-bit red, 8-bit green, 8-bit blue
  - D3DFMT_X8R8G8B8
    - 32-bit pixel format. The leftmost 8-bits are not used.
  - D3DFMT_A8R8G8B8
    - 32-bit pixel format. The leftmost 8-bits are for alpha.
  - D3DFMT_A16B16G16R16F
    - 64-bit floating point pixel format
  - D3DFMT_A32B32G32R32F
    - 128-bit floating point pixel format
  - In general, 32-bit pixel formats are used for the display and back buffer surfaces.

## Direct3D Basics – Double buffering

- Double buffering
  - *Swap chains*, and more specifically *page flipping* (or *double buffering*) are used to provide smooth, flicker-free animation between frames.
    - Front Buffer – The contents of this buffer are currently being displayed by the monitor.
    - Back Buffer – The frame currently being processed is rendered to this buffer.
    - When the monitor is done displaying the surface in the front buffer, we move it to the end of the swap chain and the next back buffer in the swap chain is promoted to the front buffer

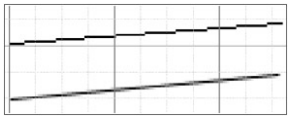



## Direct3D Basics – Depth buffer

- Depth buffer
  - A surface that does not contain image data but rather depth information about a particular pixel.
  - The possible *depth values range from 0.0(closest) to 1.0(farthest)*.
  - It uses a technique called *depth buffering (or z-buffering)*.
  - If the back buffer had a resolution of 1280x1024, there would be 1280x1024 depth buffer.
  - The format of depth buffer determines the accuracy of the depth test. Most applications work fine with a 24-bit depth buffer.
- Depth buffer format
  - D3DFMT_D32 - 32-bit depth buffer
  - D3DFMT_D24S8 - 24-bit depth buffer, 8-bit stencil buffer
  - D3DFMT_D24X8 - 24-bit depth buffer
  - D3DFMT_D24X4S4 - 24-bit depth buffer, 4-bit stencil buffer
  - D3DFMT_D16 - 16-bit depth buffer

## Direct3D Basics - Multisampling

- Multisampling
  - Direct3D supports an *anti-aliasing* technique called *multisampling*. It uses multiple pixel samples to compute the final color of a pixel.
  - D3DMULTISMPLE_TYPE enumerated type
    - consists of values that allow use to specify the number of samples to use in multisampling.
    - D3DMULTISAMPLE_NONE – no multisampling
    - D3DMULTISAMPLE_[2~16]_SAMPLE – to specify to use 2,…, 16 samples.
  - Need to check whether HW supports multisampling by using
    - IDirect3D9::CheckDeviceMultiSampleType( UINT Adapter,



D3DDEVTYPE DeviceType,
D3DFORMAT SurfaceFormat,
BOOL Windowed,
D3DMULTISAMPLE_TYPE MultiSampleType,
DWORD * pQualityLevels);

## Direct3D Basics – Memory pool

- Memory pool
  - Direct3D resources (e.g. surface) can exist in system memory (RAM), video memory (VRAM), or AGP memory.
  - Direct3D memory pool determines the memory type in which a resource is placed.
- D3DPOOL enumerated type
  - D3DPOOL_DEFAULT
    - instructs Direct3D to place the resource in the memory that is best suited for the resource type and usage – usually VRAM or AGP
    - Must be destroyed (released) prior to an IDirect3DDevice9::Reset
  - D3DPOOL_MANAGED
    - Resources placed in the managed pool are backed up in system memory, and are automatically moved between system memory and device accessible memory as needed.
    - When resources in the managed pool are accessed and changed by the application, they work with the system memory backup color and then automatically updates them to device accessible memory as needed.
  - D3DPOOL_SYSTEMMEM
    - Specifies that the resource be placed in system memory.

## Direct3D Basics – Device Capabilities

- D3DCAPS9
  - Initialize the members of a D3DCAPS9 object based on the capabilities of a particular hardware device.

```
// D3DCAPS9 instance가 초기화되었다고 가정
bool supportsHardwareVertexProcessing = false;

// 장치가 변환과 조명계산을 하드웨어로 처리할 수 있는지 여부
If (caps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT) {
   // bit가 켜져 있으므로 기능이 지원됨
   supportsHardwareVertexProcessing = true;
} else {
   supportsHardwareVertexProcessing = false;
}
```

## Direct3D Initialization

- Initializing Direct3D
  1. Acquire a pointer to an *IDirect3D9* interface
  2. Verify *hardware support* for using the current display mode format
  3. Check *the device capabilities (D3DCAPS9)* to see if the primary display adapter (i.e., primary graphics card) supports hardware vertex processing
  4. Initialize an instance of the *D3DPRESENT_PARAMETERS* structure, consisting of a number of data members that allow us to specify the characteristics of the IDirect3DDevice9 interface
  5. Create the *IDirect3DDevice9* object based on an initialized D3DPRESENT_PARAMETERS structure

## Initialization – Acquiring IDirect3D9

- IDirect3D9
  - 3D graphics를 화면에 표시하는데 이용될 물리적 하드웨어 장치의 C++객체임
  - 그래픽스 카드가 제공하는 기능, display mode, format등의 특성에 대한 정보를 얻는 과정

```
IDirect3D9* d3d9;
d3d9 = Direct3DCreate9(D3D_SDK_VERSION);
```

## Initialization – Verifying HAL Support

- After we have created IDirect3DDevice9 object, we should verify the pixel format combination are supported by the hardware.
- Check device type

```
HRESULT IDirect3D9::CheckDeivceType (
    UINT Adapter,        // 특성을 얻고자 하는 physical display adapter
    D3DDEVTYPE DeviceType, // 이용할 장치 type을 지정
    D3DFORMAT DisplayFormat, // 이용할 장치의 디스플레이 포멧
    D3DFORMAT BackBufferFormat, // 이용할 장치의 후면버퍼 포멧
    BOOL Windowed
);

D3DDISPLAYMODE mode;
d3d9->GetAdapterDisplayMode(D3DADAPTER_DEFAULT, &mode);
d3d9->CheckDeviceType(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
    mode.Format, mode.Format, true);
```

## Initialization – Checking Hardware Vertex Processing

- Must specify the type of vertex processing:
  - Hardware Vertex Processing vs. Software Vertex Processing
- Initialize a D3DCAPS9 instance
  - Based on the capacilities of default display adapter

```
HRESULT IDirect3D9::GetDeivceCaps (
    UINT Adapter,        // 특성을 얻고자 하는 physical display adapter
    D3DDEVTYPE DeviceType, // 이용할 장치 type을 지정
    D3DCAPS9 *pCaps
);
```

보통 D3DDEVTYPE_HAL로 지정

```
D3DCAPS9 caps;
d3d9->GetDeviceCaps(D3DADAPTER_DEFAULT, deviceType, &caps);
int vp =0;
if (caps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT) {
    vp = D3DCREATE_HARDWARE_VERTEXPROCESSING;
} else {
    vp = D3DCREATE_SOFTWARE_VERTEXPROCESSING;
}
```

## Initialization – D3DPRESENT_PARAM

- Fill the instance of D3DPRESENT_PARAMETERES structure
  - Determines the characteristics of IDirect3DDevice9 object

```
typedef struct _D3DPRESENT_PARAMETERS {
    UINT BackBufferWidth;          // 후면버퍼 너비
    UNIT BackBufferHeight;         // 후면버퍼 높이
    D3DFORMAT BackBufferFormat; // 후면버퍼 픽셀 포멧
    UINT BackBufferCount;          // 이용할 후면버퍼 수 – 보통 "1" 지정
    D3DMULTISAMPLE_TYPE MultiSampleType; // 후면버퍼에 쓸 멀티샘플링타입
    DWORD MultiSampleQuality;  // 멀티샘플링 레벨
    D3DSWAPEFFECT SwapEffect; // 스와핑 방법 – 보통 D3DSWAPEFFECT_DISCARD
    HWND hDeviceWindow;        // 서비스와 윈도우 핸들
    BOOL Windowed;             // 윈도우 모드로 실행할 때는 true
    BOOL EnableAutoDepthStencil; // 자동으로 깊이/스텐실 버퍼 관리하려면 true
    D3DFORMAT AutoDepthStencilFormat; // 깊이/스텐실 버퍼 포멧
    DWORD Flags;               // 0 (플래그 없음)
    UINT FullScreen_RefreshRateInHz; // 재생율 지정
    UNIT PresentationInterval;     // 시연간격 D3DPRESENT_INTERVAL_DEFAULT
} D3DPRESENT_PARAMETERS;
```

## Initialization – Creating IDirect3DDevice9

- Create the IDirect3DDevice9 interface

```
IDirect3DDevice9::CreateDevice(
    // 객체와 대응될 physical display adapter를 지정
    UINT Adapter,
    // HW 장치 (D3DDEVTYPE_HAL)
    // 또는 래퍼런스 래스터기 장치 (D3DDEVTYPE_REF)
    D3DDEVTYPE DeviceType,
    // 장치와 연결될 window handle
    HWND hFocusWindow,
    // 정점처리를 어디서 할 지 결정하는 값
    // 하드웨어 처리 (D3DCREATE_HARDWARE_VERTEXPROCESSING)
    // 또는 소프트웨어 처리 (D3DCREATE_SOFTWARE_VERTEXPROCESSING)
    DWORD BehaviorFlags,
    // 초기화된 D3DPRESENT_PARAMETERS instance 지정
    D3DPRESENT_PARAMETERS *pPresentationParameters,
    // 생성된 장치 return
    IDirect3DDevice9 **ppReturnedDeviceInterface
)
```

# Hello Direct3D