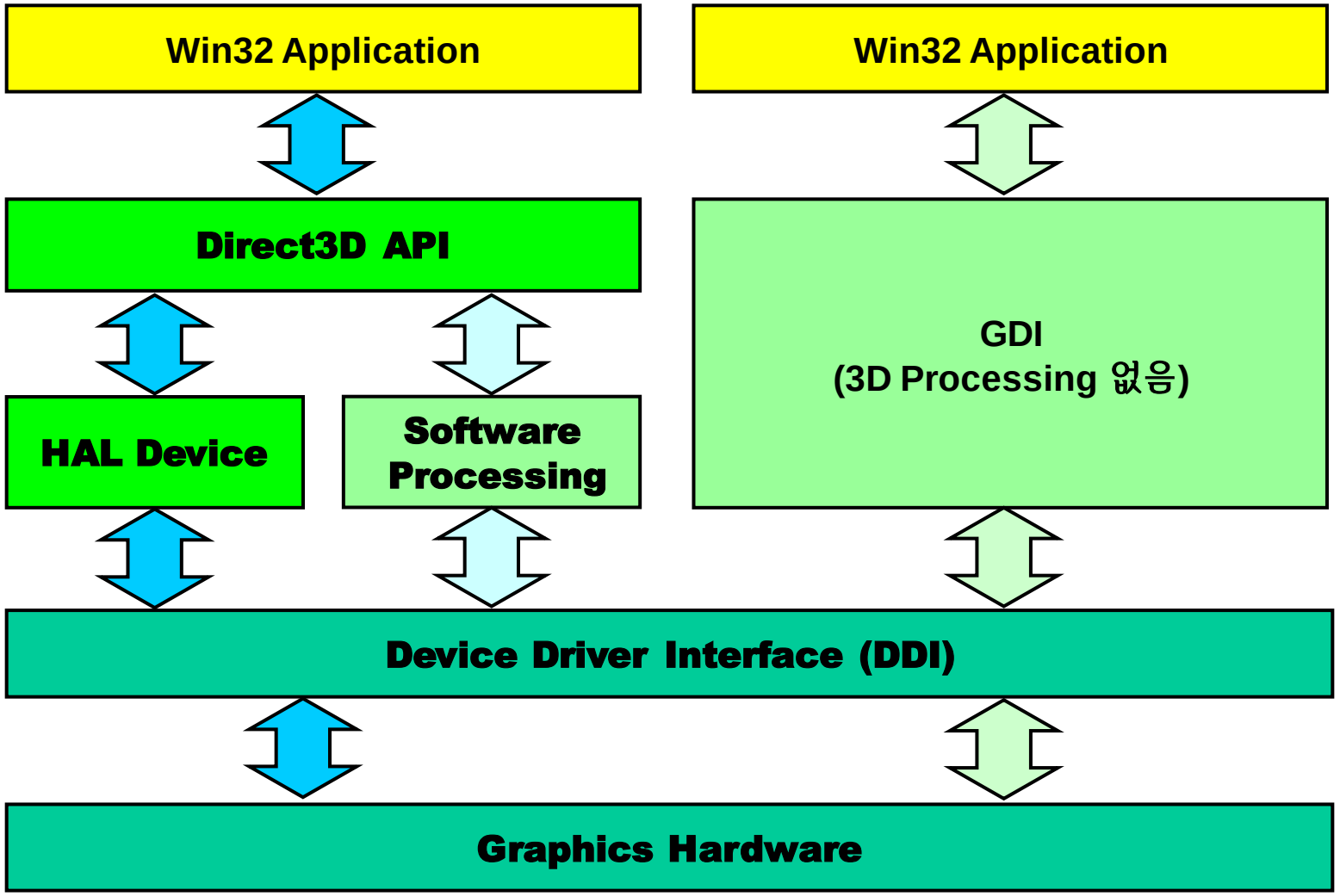
# 3D Game Programming 01

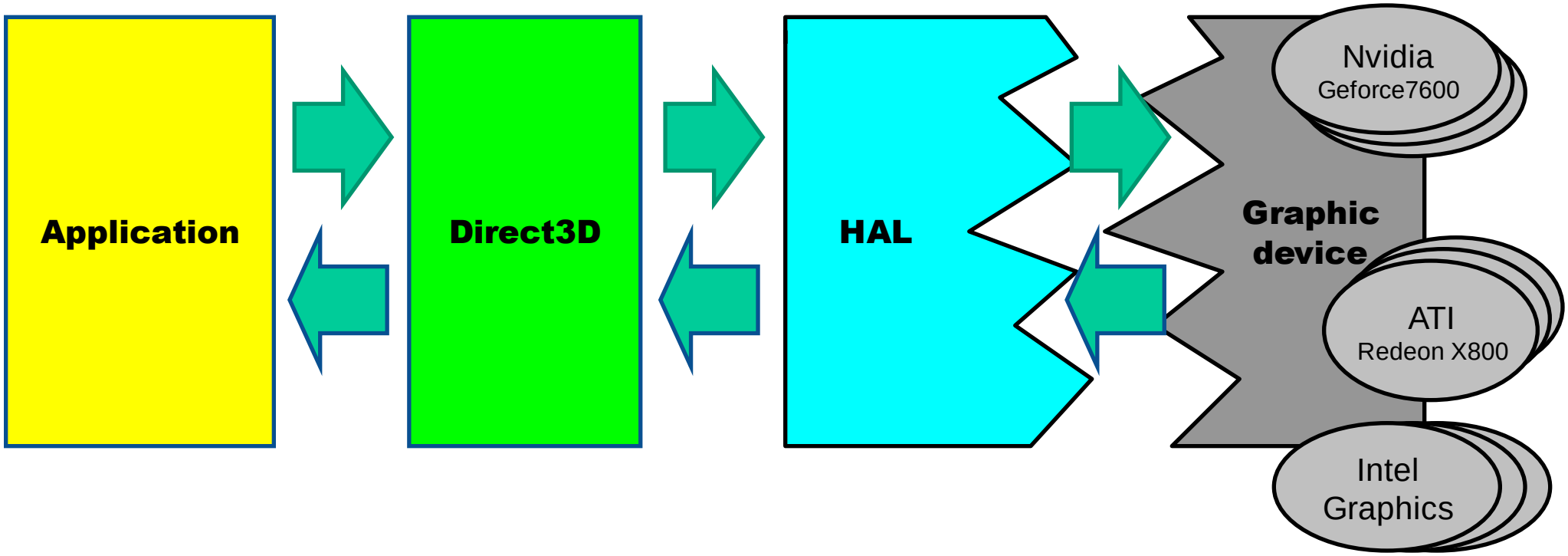
afewhee@gmail.com

- Direct3D: 하드웨어 또는 소프트웨어로 3D를 표현하는 그래픽 API
  - HAL(Hardware Abstraction Layer)
    - 하드웨어 추상 계층: 응용프로그램이 하드웨어를 이용하되 하드웨어에 독립성 유지함
    - 하드웨어에 정점 처리, 픽셀 처리 등을 지시
    - Direct3D가 각 장치의 세부적인 부분을 제어할 필요 없게 함
    - HAL에서 하드웨어 지원이 안 되는 Direct3D함수 호출 시 에러 발생 ➔ 장치 의존
  - REF 장치
    - 3D 장면을 만들기 위한 정점 처리, 픽셀 처리 등을 전부 소프트웨어로 처리
      ➔하드웨어 애뮬레이트하는 레퍼런스 래스터라이저 제공
    - 장치 비 의존적
- D3DDEVTYPE
  - HAL이 가능한 하드웨어: D3DDEVTYPE_HAL
  - REF: D3DDEVTYPE_REF
  - 하드웨어 테스트를 통해서 타입 결정

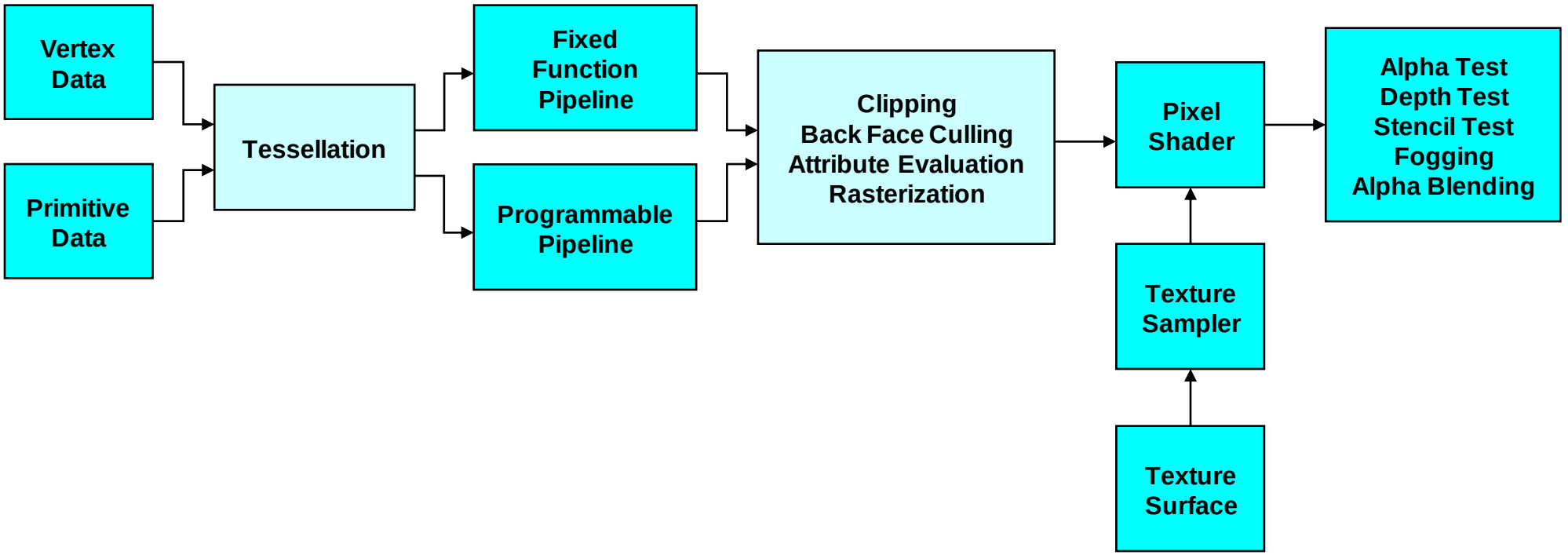

## Direct3D와 GDI 비교



- Application, Direct3D, HAL, 하드웨어 간의 관계



# Architectural Overview for Direct3D




Tessellation

미술: 크고 작은 같은 모양의 도형들을 이용하여 어떤 면이나 공간을 빈틈 없이 메우는 미술 장르.
Ex) 조각보, 타일, 에셔 그림

3D: 이미지의 처리 또는 화면 출력을 위해서 이미지를 작은 사각형 조각으로 나누는 작업

- COM(Component Object Model)
  - 소프트웨어 컴포넌트(구성요소)를 개발하기 위한 마이크로소프트 플랫폼
  - COM은 이 기술을 지원하는 모든 언어에서 프로세스간 통신과 동적 오브젝트 생성을 위해 사용

- DirectX 객체
  - DirectX의 모든 객체들은 COM을 상속
  - COM 객체 생성
    - 함수 또는 다른 COM객체의 메서드를 통해 인스턴스 생성
  - COM 객체 해제: Release() 메서드 호출

# 2.1 Software Rendering vs. Hardware



- 3D 렌더링 순서
  - 정점 변환: 월드 변환, 뷰 변환, 정규 변환, 장치 의존 변환(뷰포트 변환)
  - Rasterizing: 기하 정보를 이용해서 픽셀을 생성
  - 픽셀 처리: 텍스처 샘플링, Addressing, Filtering, Alpha, Depth, Stencil 비교

- Software Rendering ➔ 장치에 독립
  - 위의 3D 렌더링 전 과정을 CPU에서 처리
  - Rasterizing, 픽셀 처리부분에서 상당히 과부하

- Hardware Rendering ➔ 장치에 의존
  - 그래픽 하드웨어에서 지원이 되면 하드웨어에게 처리를 맡김
  - Fixed Pipe Line: 프로세스의 흐름이 고정되어 있음
  - Shader: 고정파이프라인 일부가 프로그램 가능 ➔ Vertex Shader, Pixel Shader

  - 가상 머신(Virtual Machine) 개념을 두어 하드웨어 처리에서 필요한 값을 소프트 웨어에서 설정

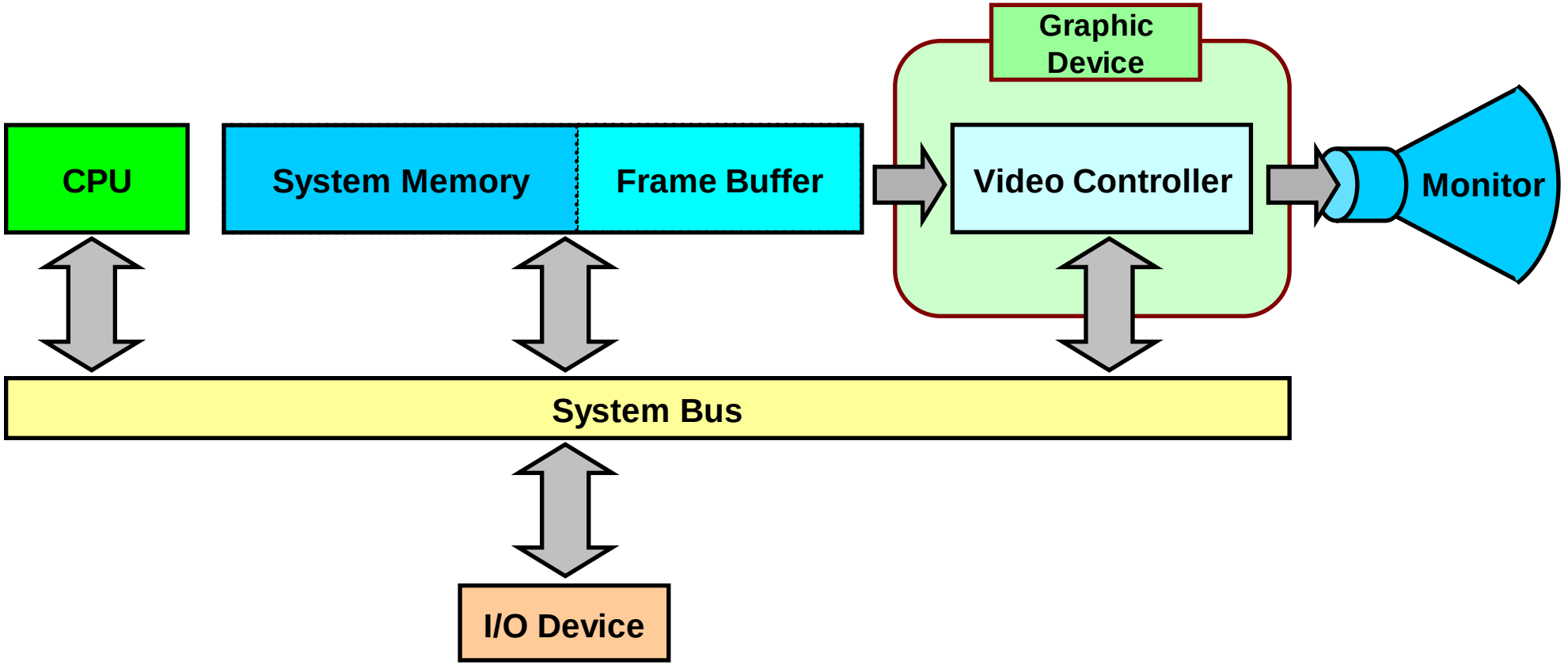
# 2.2 가상 머신

- Direct3DDevice 가상 머신
  - 프로그램 독립성을 위해 Direct3D는 내부에서 하드웨어, 혹은 소프트웨어 처리의 방식에 상관 없이 사용자에게 일정한 인터페이스를 제공 ➔ 프로세스 가상머신

- 인터페이스의 종류
  - 객체 생성 함수: Create로 시작
    Ex) CreateVertexBuffer, CreateIndexBuffer, CreateTexture...
  - 설정 함수: Set으로 시작
    - 렌더링 설정: SetRenderState(), GetRenderState();
    - 샘플러 설정: SetSamplerState()
    - 멀티 텍스처링 설정: SetTextureStageStage()
    - 정점 변환 설정: SetTransform
    - 기타 값 설정: Ex) SetTexture, SetMaterial, SetFVF, SetVertexShader...
  - 후면버퍼에 렌더링 함수: Draw로 시작
    Ex) DrawPrimitive, DrawIndexedPrimitive, DrawPrimitiveUP,...

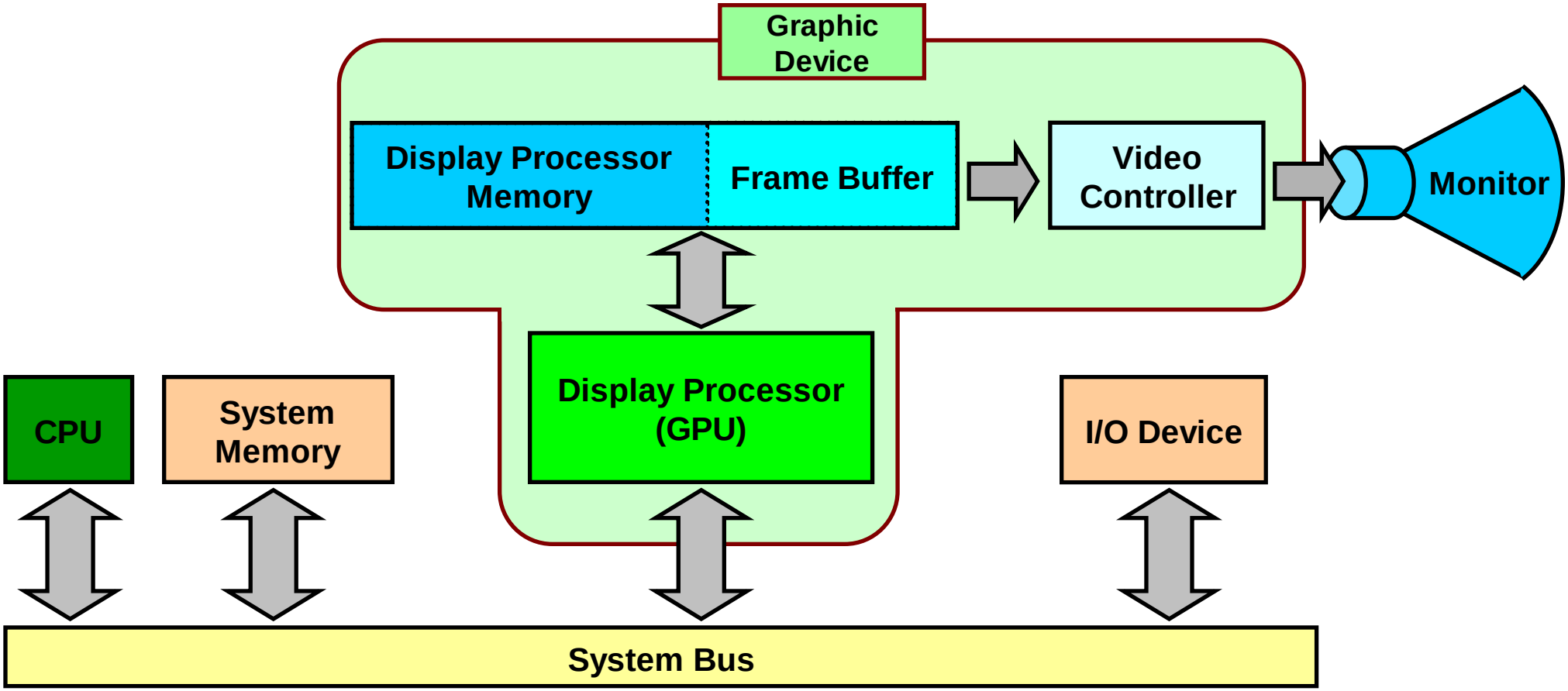
# 2.3 래스터 주사 시스템(Raster Scan System)

- 시스템 메모리의 일부를 프레임 버퍼로 사용하는 구조
  - CPU가 Graphic 작업을 수행
  - System 메모리 일부를 Frame Buffer로 사용



# 2.3 래스터 주사 시스템(Raster Scan System)

- Display Processor == Graphic Processor
  - 응용 프로그램의 픽셀 처리에 대한 Graphic 작업을 CPU 대신 수행



- 기하 정보(Geometry), 이미지 처리에 의해 생성된 픽셀들의 정보를 보관하고 있는 장소

- Direct3D는 1D, 2D, 3D 픽셀들을 다룰 수 있으며 이 때 서피스에 보관

- 픽셀 뿐만 아니라 깊이, 스텐실 정보도 서피스에 저장 ➔ Depth & Stencil Buffer

- 서피스의 너비, 높이, 깊이는 픽셀 단위로 계산

# 3.2 서피스(surface)

- IDirect3DSurface9 인터페이스
  - GetDesc – 서피스에 대한 정보 ➔ 서피스 정보 구조체: D3DSURFACE_DESC
  - LockRect – 픽셀이 저장되어 있는 메모리 포인터 가져옴.
  - UnlockRect – Lock에 의한 서피스 사용에 대한 독점 모드를 해제

- 서피스를 가져올 수 있는 객체: 디바이스의 Backbuffer, 텍스처

- Ex)

```
D3DSURFACE_DESC surfaceDesc;
pSurface->GetDesc(&surfaceDesc);

D3DLOCKED_RECT lockedRect;
pSurface->lockRect(&lockedRect, 0, 0);
...
DWORD* pImageData = (DWORD*)lockedRect.pBits; // 이미지 포멧에 맞게 casting 필요
...
pSurface->UnlockRect();
```

# 3.3 멀티 샘플링(Multisampling)

- Aliasing:
  - 저주파 표본화(Low-Level Frequency Sampling) ➔ Under Sampling
  - 표본화(Sampling) 과정 중 고주파 성분이 저주파 성분으로 변환되는 현상
  - 영상에서 계단 모양의 외형을 갖는 왜곡 현상
- Antialiasing
  - Under Sampling 처리를 보상하는 방법
  - Nyquist Sampling Frequency (나이키스트 표본화 율)
    - $f_s = 2\ f_{max}$ ($f_{max}$ :Sampling 대상에서 나타나는 최대 주파수)
    - $\Delta x_s = \Delta x_{cycle}/2$
    - $\Delta x_{cycle} = 1/f_{max}$
  - Raster Scan Display: 전자 광선이 한번에 한 행씩 위에서 아래로 스크린을 가로질러 가면서 광선의 세기를 조절해서 디스플레이하는 방식

    ➔ 미세한 농담과 색 패턴을 담고 있는 장면 표현에 적합

    필연적으로 Aliasing문제가 발생
- Super Sampling (Post Filtering)
  - Aliasing문제를 해결하기 위한 방법

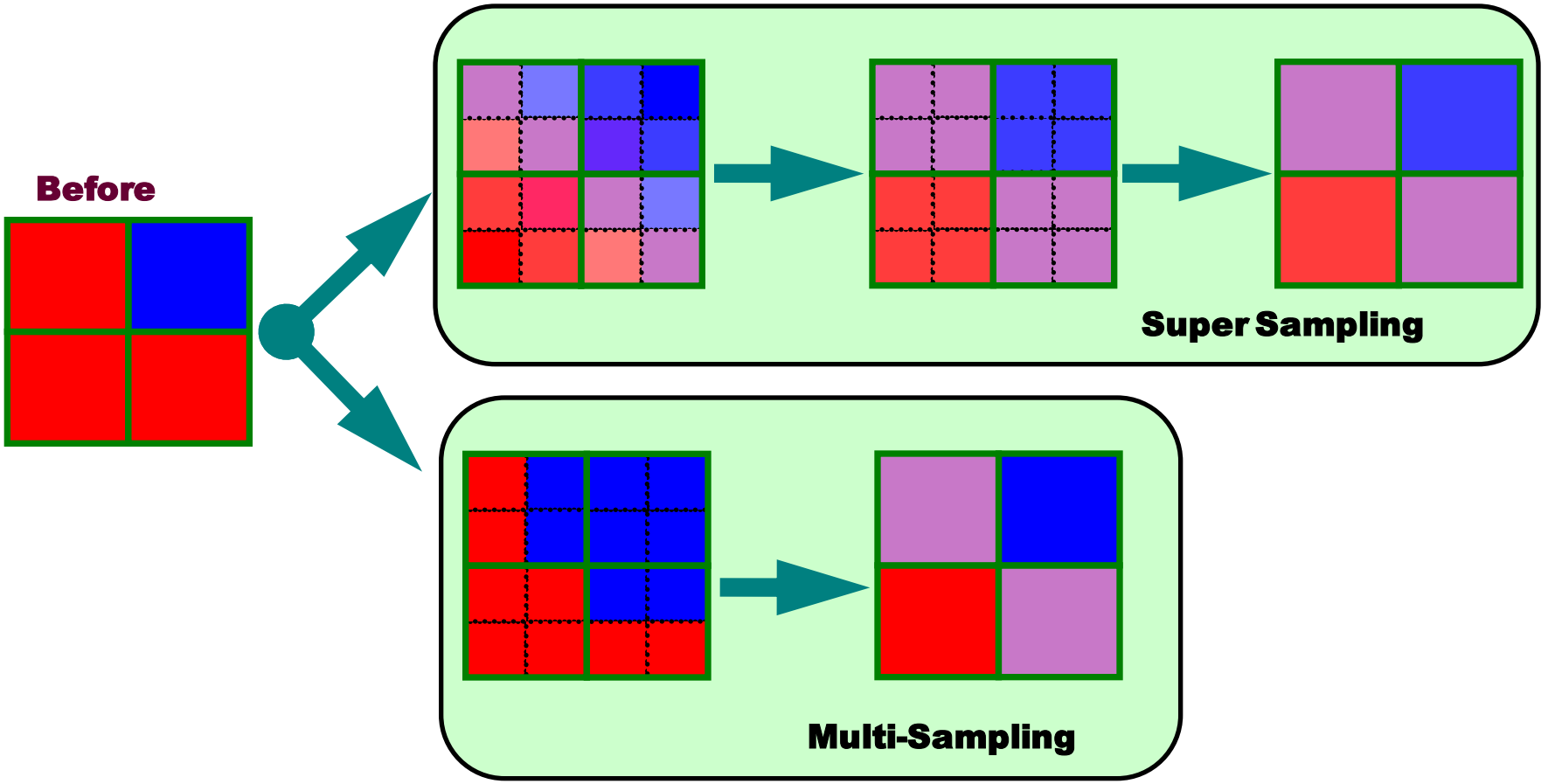
# 3.3 멀티 샘플링(Multisampling)

- Super Sampling (Post Filtering)
  - 화면에 표시할 픽셀의 적절한 값을 설정하기 위해 고 해상도에서 렌더링 물체를 표본화(Sampling)한 후 최종 픽셀 값을 결정
  - 하위 픽셀들은 Sampling 이론을 근사한 값들을 이용해 픽셀을 결정
  - 하위 픽셀들을 모아서 최종 픽셀 계산

- Multi-Sampling
  - Super Sampling의 한 종류
  - 단순한 Sampling을 방식으로 하위 픽셀 계산 ➔ 하위 픽셀을 더해서 최종 픽셀 결정
  - 처리 속도와 Anti-aliasing 효과 우수

- D3DMULTISAMPLE_TYPE: 멀티 샘플링 레벨
  - D3DMULTISAMPLE_NONE:
    - 멀티 샘플링 지정 없음
  - D3DMULTISAMPLE_{1..16}_SAMPLE … D3DMULTISAMPLE_16_SAMPLE:
    - 1에서 16까지의 멀티샘플링을 지정
  - IDirect3D9::CheckDeviceMultiSampleType() 멀티 샘플링 타입 확인

- Super Sampling vs Multi-Sampling(2x2)



- Direct3D의 서피스에 대한 픽셀 포맷

- D3DFORMAT
  - D3DFMT_R8G8B8 – 24bit. R: 8bit, G: 8Bit, B: 8Bit
  - D3DFMT_X8R8G8B8 – 32bit. X는 사용하지 않음
  - D3DFMT_A8R8G8B8 – 32bit. A: alpha 8Bit
  - D3DFMT_X1R5G5B5 – 16bit.
  - D3DFMT_R5G6B5 – 16bit.
  - D3DFMT_A1R5G5B5 – 16bit. Alpha 1Bit
  - D3DFMT_A16B16G16R16F – 64bit 부동소수점 포맷.
  - D3DFMT_A32B32G32R32F – 128bit 부동소수점 포맷
  - ...
- 해석 방법
  - A8R8G8B8이면 전체 32비트 중 가장 왼쪽부터 8bit씩 할당


# 3.5 메모리 풀 (Memory Pool)

- Direct3D 인터페이스에 의해 생성된 자원은 Direct3D에 의해 관리. ➔ 자원 관리를 위한 메모리 풀 이용

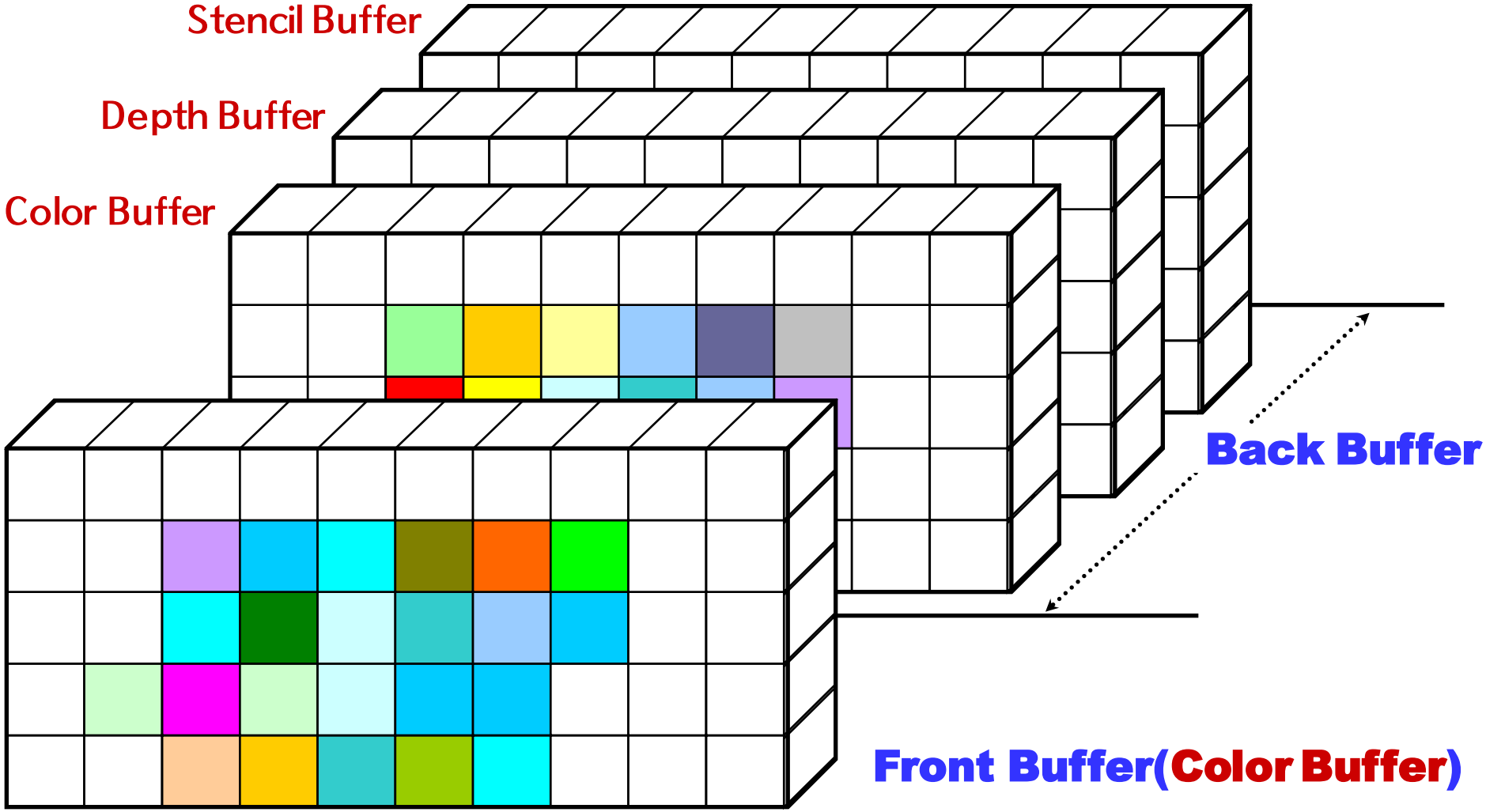
- D3DPOOL_"메모리 풀"
  - D3DPOOL_DEFAULT: 자원의 타입과 이용에 가장 적합한 메모리에 보관하도록 Direct3D에 요청. 대부분 비디오 메모리나 AGP를 이용
    ➔자원은 Device9::Reset 호출 이전에 반드시 해제, Reset 호출 이후에 다시 생성
  - D3DPOOL_MANAGED: 디바이스가 필요하다면 자원을 복사해서 백업을 만듦 ➔디바이스가 Lost 상태에서도 자원을 다시 재 생성 필요 없음
  - D3DPOOL_SYSTEMMEM: 일반적으로 3D 장치가 접근할 수 없는 메모리 영역. ➔시스템 메모리에 자원을 생성
  - D3DPOOL_SCRATCH: 자원을 시스템 메모리에 생성. 장치의 제한을 따르지 않음. 장치를 통해 자원 접근이 어려우나 생성, Lock, 복사는 가능

- 가장 일반적인 메모리 풀: D3DPOOL_MANAGED

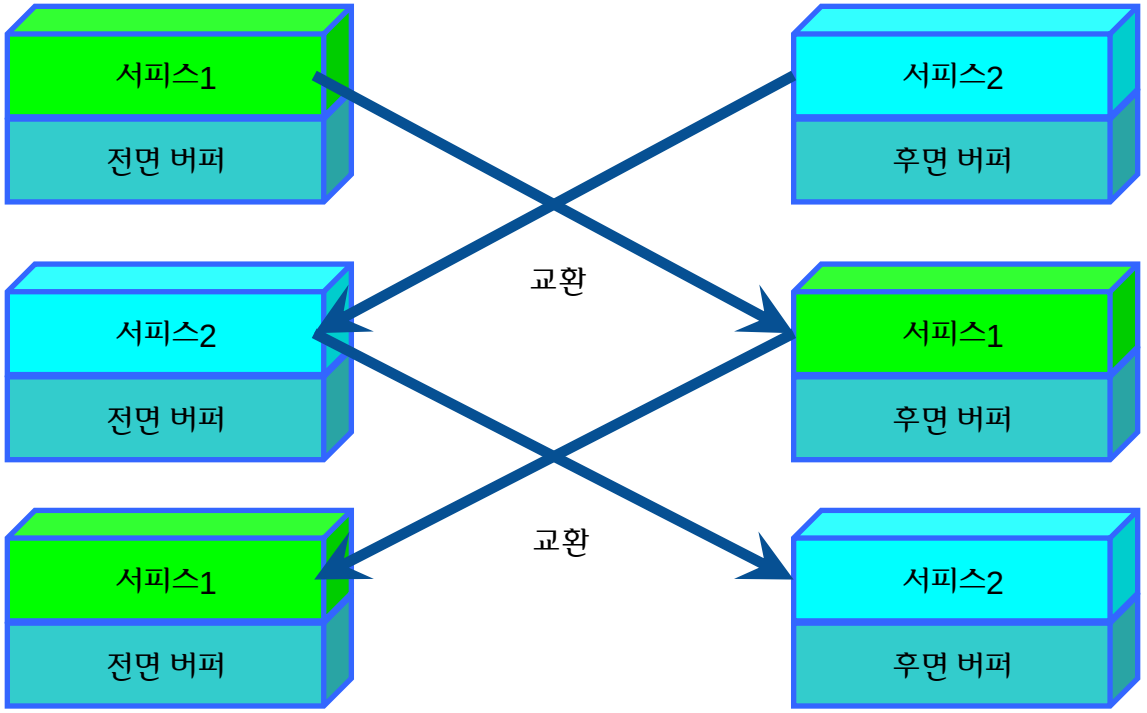
- 모니터와 같은 Video 장치 화면에 색상을 출력하기 위한 여러 버퍼들의 집합 == 재생 버퍼

- 구성:
  - Front Buffer: 현재 비디오 장치에 출력되고 있는 메모리 영역. 색상버퍼의 일종
  - Back Buffer: 속도를 향상 시키기 위해 Front Buffer와 동일한 색상 정보를 정보를 저장하고 있는 메모리 영역. 픽셀 연산을 위한 여러 하위 버퍼가 존재
    - Color Buffer: 픽셀를 저장하는 메모리
    - Depth Buffer: 새로운 색상 값(픽셀)을 색상 버퍼에 덮어 쓸 것인가에 대한 판별 값. = Z buffer라 부르기도 함
    - Stencil Buffer: Depth Buffer와 비슷하나 판별 값이 Making 값

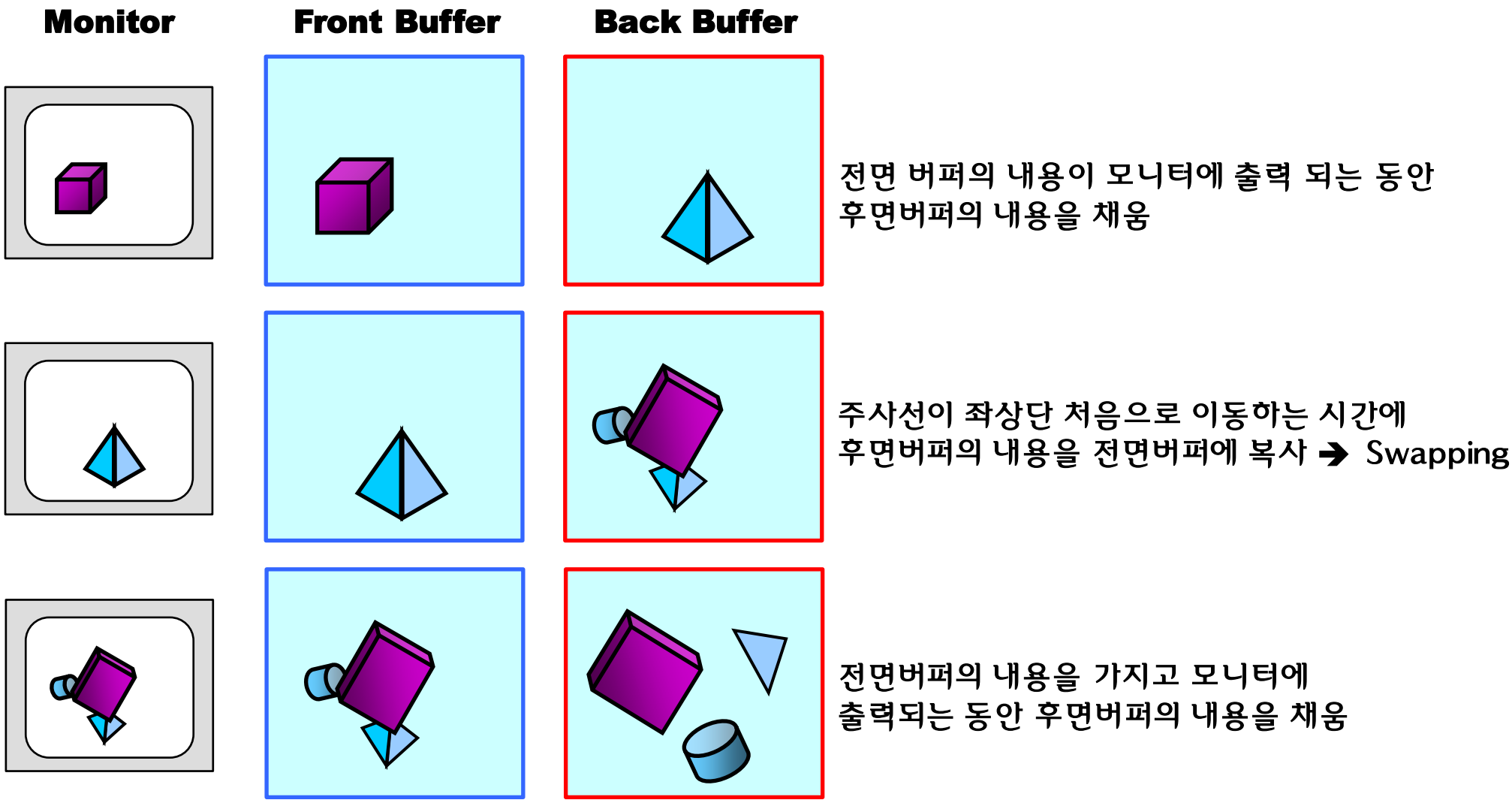

- OpenGL에서 지원하는 Frame Buffer: Auxiliary, color indices, overlay planes



- Swap Chain: Direct3D는 보통 두 개나 세 개의 서피스를 하나의 컬렉션으로 관리. ➔ Chain처럼 연결해서 관리
  - IDirect3DSwapChain9
  - 대부분 Direct3D가 직접 관리
  - Swap Change: Swap Chain을 교환



- Flipping



- 픽셀의 깊이 정보를 저장하는 서피스
- 깊이 정보를 이용, 색상 버퍼의 픽셀을 새로운 픽셀로 대체할 것인가를 판별
- 마지막 갱신한 픽셀의 깊이 값만 가짐
- Direct3D는 깊이 버퍼와 스텐실 버퍼를 하나의 버퍼에서 나누어서 사용

- 깊이 버퍼 포맷 == 깊이 테스트 정확도
  - 대부분 24bit 깊이이면 충분
  - D3DFMT_D32 – 32bit 깊이 버퍼
  - D3DFMT_D24S8 – 32bit 중 24bit 깊이, 8bit 스텐실 버퍼
  - D3DFMT_D24X8 – 32bit 중 24bit 깊이 버퍼, 8bit 사용 안함
  - D3DFMT_D24X4S4 – 24bit 깊이, 4bit 스텐실, 4bit 사용 안함
  - D3DFMT_D16 – 16bit 깊이 버퍼

- 특별한 경우가 아니면 D3DFMT_D16는 사용하지 않음


- 버텍스 프로세싱:
  - 소프트웨어 버텍스 프로세싱: 항상 이용 가능
  - 하드웨어 버텍스 프로세싱:
    - 그래픽 카드가 버텍스 프로세싱을 지원하는 경우에만 이용
    - 소프트웨어 방식에 비해 성능이 우수

- 장치 특성
  - D3DCAPS9 구조체 이용

```
// D3D Instance를 이용한 장치 특성 얻기
D3DCAPS9 pCaps={0};
pD3D->GetDeviceCaps(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, &pCaps);

...
// Device Instance를 이용한 장치 특성 얻기
D3DCAPS9 pCaps={0};
pDevice->GetDeviceCaps(&pCaps);
```


- IDirect3D9 객체 생성
- D3DCAPS9 구조체 이용 장치 특성 확인.
- D3DPRESENT_PARAMETERS 구조체 설정
- IDirect3D9 객체 멤버 함수 CreateDevice를 이용 Device 객체를 생성

```
// IDirect3D 객체 생성
IDirect3D9* pD3D;
pD3D  =  Direct3DCreate9(D3D_SDK_VERSION);

// 장치 특성확인
D3DCAPS9        Caps;
memset(&Caps, 0, sizeof(D3DCAPS9));
pD3D->GetDeviceCaps(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, &Caps);

// 하드웨어 버텍스 프로세싱 지원 확인
if ( Caps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT )
{
}
```

# 4.1 Device 객체 - D3DPRESENT_PARAMETERS 구조체 채우기

- D3DPRESENT_PARAMETERS 구조체 내용
  - 전부 후면 버퍼의 내용

```
typedef struct _D3DPRESENT_PARAMETERS_ {
    UINT            BackBufferWidth;            // 후면 버퍼 너비
    UINT            BackBufferHeight;           // 후면 버퍼 높이
    D3DFORMAT       BackBufferFormat;           // 후면 버퍼 픽셀 포맷
    UINT            BackBufferCount;            // 더블 버퍼링을 위한 후면 버퍼 수. 보통=1, 0이면 1
    D3DMULTISAMPLE_TYPE MultiSampleType;// 후면 버퍼에 이용할 멀티 샘플링의 타입
    DWORD           MultiSampleQuality;         // 멀티 샘플링 레벨
    D3DSWAPEFFECT     SwapEffect;               // Swap chain 교환 방법  보통 = D3DSWAPEFFECT_DISCARD
    HWND            hDeviceWindow;              // 서비스의 윈도우 핸들
    BOOL          Windowed;                     // 윈도우 모드: TRUE, Full Window: FALSE
    BOOL          EnableAutoDepthStencil;       // 자동으로 깊이-스텐실 버퍼 생성 보통 = TRUE
    D3DFORMAT AutoDepthStencilFormat;           // 깊이/스텐실 버퍼의 포맷  보통 = TRUE
    DWORD           Flags;                      // Flag 값
    UINT      FullScreen_RefreshRateInHz;       // 재생률. 윈도우 모드 = 0, Full Mode: 지원되는 Hz 지정
    UINT      PresentationInterval;             // Present 간격:
                                    // D3DPRESENT_INTERVAL_IMMEDIATE : 즉시 처리
                                    // D3DPRESENT_INTERVAL_DEFAULT : 주사선이 다 그리고 나서 초기
                                    // 위치로 이동할 때 복사
} D3DPRESENT_PARAMETERS;
```

- 윈도우 핸들, D3DPRESENT_PARAMETERS 구조체 이용

```
HRESULT   IDirect3D9::CreateDevice(
    UINT Adapter,              //디스플레이 어댑터 보통=D3DADAPTER_DEFAULT: 듀얼 모니터의 경우 1번 모니터
    D3DDEVTYPE DeviceType, // 장치 타입(하드웨어:HAL or 소프트웨어:REF)
    HWND  hFocusWindow,        // 장치에 연결할 윈도우 핸들
    DWORD BehaviorFlags,       // Hardware, Software, Mixed Vertex Processing
                               // 보통 Software vertex Processing ➔ 가장 안전
    D3DPRESENT_PARAMETERS *pPPM, // D3DPRESENT_PARAMETERS  변수
    IDirect3DDevice9** ppDevice  // 생성된 IDirect3DDevice9 객체
);

IDirect3DDevice9*   pDevice = NULL;
D3DPRESENT_PARAMETERS d3dpp={0};
...
hr = pD3D->CreateDevice( D3DADAPTER_DEFAULT       // 기본 어댑터
                       , D3DDEVTYPE_HAL           // 하드웨어 가속
                       , m_hWnd                   // 장치와 연결된 윈도우
                       , D3DCREATE_SOFTWARE_VERTEXPROCESSING
                       , &d3dpp
                       , &pDevice);
if( FAILED(hr))
    return E_FAIL;
```

- 2D게임 프로젝트에서 사용했던 코드를 가지고 SDK/Samples/C++/Direct3D/tutorials/에 있는 Tut01... Tut06에 대한 예제를 옮겨와 구현해 보시오.