Direct3D 12 파이프라인 (part1)

래스터라이저

안티 에일리어싱, Depth Cull등 여러가지 설정을 할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

또한 Viewport를 설정 할 수 있다.

Viewport란 Render Target의 영역을 나타내며, 명령 리스트가 Reset될 때마다 다시 설정해야 한다.

Scissor Rect 또한 설정이 가능한데, 이는 렌더링에서 제거하지 않을 영역을 나타낸다.

RSSetViewports와 RSSetScissorRects함수를 이용하여 Set해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

정점 버퍼를 넘겨주지 않고 렌더링하는 코드이다.

삼각형 하나를 그릴 때 입력 조립기에서 정점마다 정점 Shader를 한 번씩, 총 3번 부른다.

하지만 정점 데이터가 없으므로 int값 Vertex ID를 하나 설정한 후 SV\_VertexID라는 시멘틱을 달았다.

그래서 정점 데이터는 없지만 첫번째 정점에 대하여 0번을 nVertexID에 담아서 정점 Shader를 호출한다.

그 후 SV\_POSITION은 래스터라이저가 기대하는 투영 좌표계로 변환이 된 정점을 넘기겠단 뜻이다.

Shader 컴파일

컴파일 : 어떤 언어의 코드 전체를 다른 언어의 코드로 바꾸는 것

속성 – HLSL 컴파일러 – 일반

진입점 이름이라는 Shader메인 함수의 이름을 적어주어야 한다.

또한 최적화 사용과 디버깅 정보 사용을 설정 할 수 있으며 Shader형식과 Shader모델을 설정해준다.

프로젝트를 빌드할 때 Shader 이름으로 cso파일을 만들어준다.

정점 Shader 와 픽셀 Shader를 같은 파일로 작성하면 같이 컴파일 되므로 다른 파일로 컴파일해야 함

출력파일에 가면 개체 파일 이름을 바꾸어 주어서 하나의 소스파일로도 여러 개의 cso생성 가능하다.

컴파일 된 Shader코드 읽기

D3DCompileFromFile 함수를 사용해서 읽을 수도 있고, 비주얼 스튜디오가 컴파일을 하고 응용 프로그램에서 생성된 바이너리 파일을 읽을 수도 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 비주얼 스튜디오에서 컴파일한 obj코드를 파일을 open해서 읽고 Shader에 연결하는 코드이다.

미리 컴파일된 파일을 파일로부터 읽으면 응용 프로그램에서 불러와서 읽는 것 보다 시간이 절약된다.

가급적이면 비주얼 스튜디오에서 컴파일한 쉐이더 코드를 바이너리 파일을 읽어서 설정하는 것이 좋다.

(따라하기 코드는 안 좋은 방법임)

Direct3D 리소스

리소스들은 비디오 메모리에 저장되어 있다.

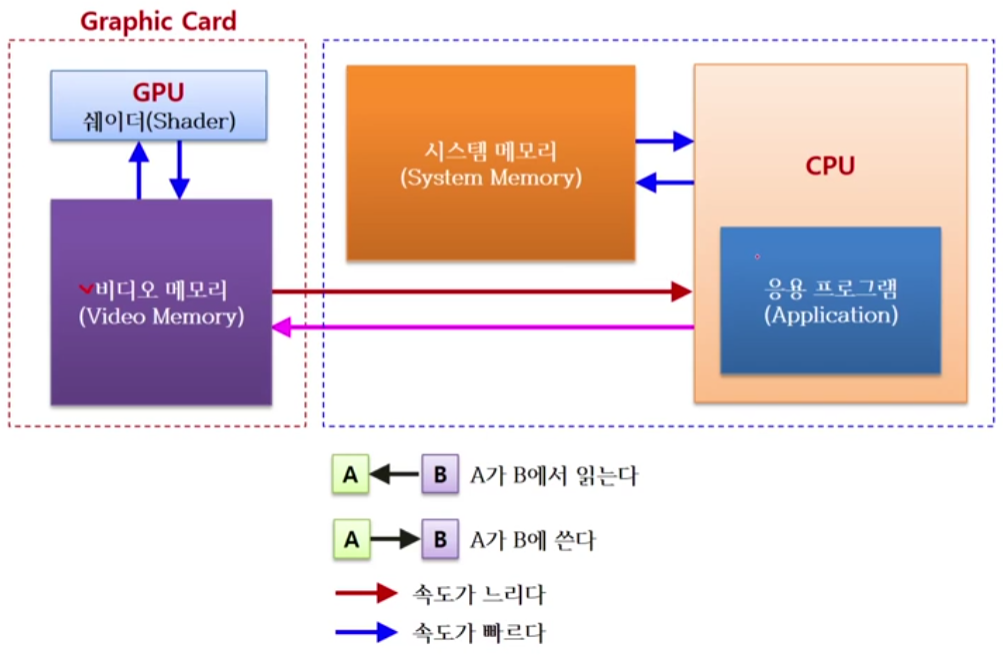
보통의 경우엔 파일형태의 리소스를 CPU에서 읽어서 생성하고 GPU렌더링 과정에서 그것을 사용한다.

리소스는 버퍼 또는 Texture의 형식을 가지는데 Type, 읽기 속성, 쓰기 속성을 지정할 수 있다.

또한 리소스는 CPU와 GPU가 리소스에 접근을 할 수 있는 여부를 지정할 수 있다.

비디오 메모리란 GPU에서 사용할 수 있는 메모리다.

비디오 메모리는 GPU가 읽고 쓸 땐 굉장히 빠르지만 CPU에서 읽고 쓸 땐 오래 걸린다.



CPU를 사용하더라도 GPU로 데이터를 전달하는 횟수를 가급적 줄여야 한다.

그렇기 때문에 한번에 여러 동작을 하도록 코드를 짜는 것이 좋다.

입력 조립 단계 설정하기

1. 정점 버퍼 또는 인덱스 버퍼를 생성한다 (입력 버퍼)

CreateCommittedResource()

1. 정점 버퍼 또는 인덱스 버퍼에 대한 뷰를 생성한다.

VERTEX\_BUFFER\_VIEW 와 INDEX\_BUFFER\_VIEW에 위의 리소스를 연결만 하면 된다.

1. 입력 레이아웃을 생성한다. 입력 레이아웃은 정점 버퍼의 한 원소의 구조를 나타낸다.

INPUT\_ELEMENT\_DESC

1. 입력 버퍼와 입력 레이아웃을 파이프라인 상태에 연결하여 정점의 구조를 파이프라인에 알린다.

InputLayout

1. 프리미티브 유형을 지정하며 IA단계는 입력 데이터를 이 프리미티브 형태로 조립한다.

IASetPrimitiveTopology()

1. 입력 버퍼를 입력 조립 단계에 연결한다.

IASetVertexBuffers(), IASetIndexBuffer()

1. 그리기 함수를 호출한다.

DrawInstanced() – 정점 버퍼, DrawIndexedInstanced() – 인덱스 버퍼

리소스의 갱신

Map()은 리소스의 포인터를 넘겨준다.

리소스는 비디오 메모리인데 CPU에게 비디오 메모리의 주소를 넘겨준다.

어떤 응용프로그램이 리소스에 대하여 Map하여 주소를 넘겨 받으면 또 다른 프로그램이 Map할 수 없다.

Map을 한 뒤 Unmap을 해주어야 사용이 가능해져서 세트로 사용해야 한다.

HEAP\_TYPE\_DEFAULT – Map() 호출 불가능

HEAP\_TYPE\_UPLOAD – 리소스 포인터에서 읽기를 하면 안 된다. 영구적 매핑 가능 Unmap() 불필요

HEAP\_TYPE\_READBACK – CPU와 GPU 접근 사이에 Unmap()을 호출해야 한다.

CopyResource()는 리소스 전체를 복사해준다. 그래서 복사할 리소스, 받을 리소스를 지정해준다.

이때 두 리소스는 같은 크기와 유형을 가져야 한다.

정점 데이터를 Default heap에 초기화 해주기 위해선 Upload heap이 필요하다.

그래서 map할 수 있는 Upload힙을 생성해서 CopyResource로 Upload heap의 내용을 Default heap으로 복사한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이때 Upload heap을 release해준다면 문제가 생기는데 그 이유는 Copy Resource가 그 때 실행되는 것이 아니라 나중에 실행되는데 release해줬다가 Upload heap이 사라지면 문제가 생기기 때문이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

정점 버퍼를 만들어 줬으면 입력 레이아웃(IA)에 정점 버퍼를 View로 연결할 수도 있고 정점 하나의 구조를 입력 조립 단계에 알려줘야 한다.

정점 버퍼의 구조를 설명하는 것이 입력 레이아웃이다.

입력 레이아웃에 따라 정점 버퍼를 읽어 Vertex Shader로 넘기는데 그렇기 때문에 Vertex Shader의 입력 시그니처와 구조가 일치해야 한다.

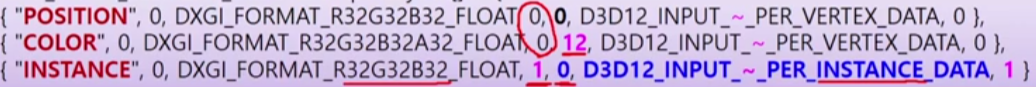
입력 버퍼의 모든 원소들은 같은 자료형을 가져야 한다.

입력 데이터는 하나 또는 여러 개의 버퍼로 표현되고 조립될 수 있다.

정점 버퍼란 정점들의 배열이다.

정점은 여러 개의 값을 가지는 구조체로 이루어져 있다. (ex: position, color, normal, uv … )

하나의 정점의 구조는 input element들의 배열과 같아야 한다.



시멘틱의 이름, 시멘틱 번호, 벡터의 종류, 연결되는 슬롯, 정점의 시작점, 데이터 종류… 순으로 적는다.

이 때 INSTANCE\_DATA란?

텍스트이(가) 표시된 사진

자동 생성된 설명

인스턴스 == 오브젝트 개수이다.

즉, DrawInstanced()의 두번째 인자는 똑같이 그려줄 오브젝트의 개수를 말한다.

Shader 코드에서 SV\_InstanceID라는 것을 받을 수 있는데 Instance의 번호가 이것으로 계속 바뀐다.

즉, 하나의 오브젝트를 모두 그리면 Instance값이 하나 올라서 두번째 INSTANCE 버퍼 값을 가져온다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그렇기에 위와 같은 코딩이 가능해진다.

VS\_INPUT을 보면 4x4행렬을 INSTANCE로 받고, INSTANCE4개를 1번 슬롯에 모두 받을 수 있다.

그러면 4개의 float4값이 만나 4x4행렬이 되고 이를 알아서 행렬로 바꾸어 사용할 수 있다.

위의 코드는 return할 때 Instance값을 그대로 넘겨준다.