<Fundamentals>

FrameRate

- 게임에서 중요한 것은 60 프레임을 유지하는 것

렌더링

- Painter's Alogorithm
  카메라에서 멀리 있는 것부터 먼저 그림
- Batch Processing
  유사한 일들을 모아서 한꺼번에 처리

기하학적 모델링

- 객체 / 모델 / 메쉬
  모델: 물체의 외관을 표현하기 위한 연결된 다각형(대개 삼각형)들의 집합
- 면(은면/정면): 메쉬의 각 다각형
- x y z 실수 3 개가 점 하나 -> 점들의 집합 -> 실수들의 집합 -> 메쉬의 정보

좌표계

- 왼손 좌표계(D3D 사용): y-up 왼손 좌표계

모델 좌표계

- 모델을 표현하기 위한 좌표계
  모델마다 자체적인 별도의 좌표계를 갖고 있다고 가정

월드 좌표계

- 게임 세계를 하나의 통일된 좌표계로 표현

모델과 객체

- 모델 = 메쉬(기하학적 모델)
- 객체 = 모델의 인스턴스
- 객체: 위치와 방향

다각형 와인딩 순서

- 은면 제거
  카메라가 볼 수 없는 면은 그리지 않음
- 와인딩 순서
  다각형의 정점들을 나열하는 순서
  D3D 는 바깥쪽 면 기준으로 시계방향 와인딩 순서

변환 파이프라인

정점(모델 좌표계)

1. 월드 변환

월드 좌표계

2. 카메라 변환

카메라 좌표계

3. 투영 변환

투영 좌표계

4. 화면 변환

화면 좌표계

변환: 평행이동(Translation)

- 평행이동(Translation): 객체를 월드좌표 위치로 평행 이동
- 인스턴싱: 하나의 메쉬를 여러 객체에서 공유하는 방법

변환: 회전(Rotate)

- 항상 원점을 기준으로 회전한다고 가정함.

- 평행이동과 회전
  공전: 선 평행이동, 후 회전(TR)
  자전: 선 회전, 후 평행이동(RT)

객체

- 객체의 클래스에 위치(좌표)와 방향은 필수

카메라 변환

- 카메라 정보
  월드 좌표계에서 카메라의 위치
  카메라의 방향
  카메라의 화각(FOV)

- 카메라 좌표계
  게임 월드를 카메라 중심의 상대적인 좌표계로 표현

- 카메라 변환

1. 카메라를 월드 좌표계의 원점으로 평행이동. 같은 평행이동을 모든 객체에 적용
2. 카메라 좌표계의 축이 월드 좌표계 축과 일치하도록 카메라 ㅏ 회전. 같은 회전을 모든 객체에 적용
3. 객체는 카메라를 중심으로 한 좌표계에 표현됨

투영 변환

- 원근 투영 변환
  - ■ 원근감

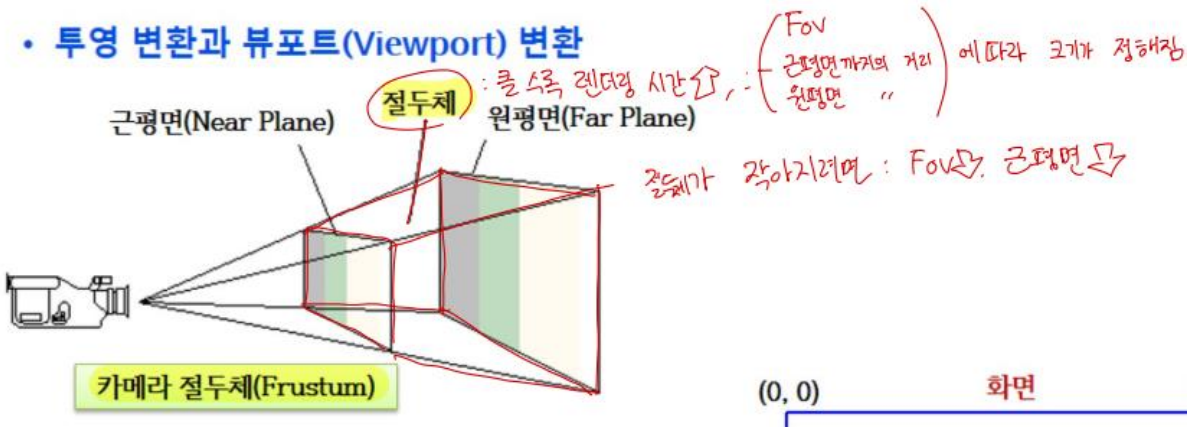
    멀고 가까움의 기준은 카메라.

    멀리있는 객체일수록 카메라 중심선 근처에 위치

- 투영 좌표계

  카메라의 중심선을 기준으로 상하좌우를 하나의 좌표계로 표현

- 카메라 절두체



  - ■ 절두체 클수록 렌더링 시간 증가.
  - ■ 절두체가 작아지려면 FOV(화각)와 근평면까지의 거리 감소
  - ■ FOV 가 크면

망원경과 유사.

물체가 아주 작게 그려짐.

물체가 화면에서 사라지는 데 한참 걸림

■ FOV 가 작으면

현미경과 유사.

물체가 크게 그려짐.

물체가 화면에서 금방 사라짐(ex. 레이싱 게임).

프레임 레이트가 잘 나옴.

화면에 나오는 obj 수가 줄어들기 때문에.

화면 좌표 변환

- 투영 좌표 공간을 화면으로 매핑
- 뷰포트
  투영 좌표 공간이 실제로 매핑될 화면

벡터

- 점 또는 방향과 크기를 나타냄
- 점은 항상 벡터지만, 벡터는 아님.

벡터 정규화

- 단위 벡터 = 크기가 1 인 벡터
- 크기가 1 이 아닌 벡터를 단위 벡터로 만드는 것 = 벡터 정규화
- 방향 벡터는 항상 단위 벡터임

<DirectX Math>

DirectX Math

- SIMD(Single Instruction Multiple Data) 컴퓨터
  SIMD 명령어는 동시에 4 개의 32 비트 실수(정수)를 연산할 수 있음
  (CPU 가 내부적으로 SIMD 로 연산함. Ex. +, -, *, %)
  두 개의 4-요소 벡터를 더할 경우 한 번의 덧셈 명령으로 처리할 수 있음. (SIMD 아니면 4 번 더해야 함)
- XMVECTOR/XMMATRIX
  XMVECTOR: SIMD 하드웨어 레지스터에 대응, 16 바이트 정렬 필요
  XMMATRIX: 4 개의 SIMD 하드웨어 레지스터에 대응, 16-바이트 정렬이 필요함.
- 클래스/구조체의 멤버로는 XMFLOAT3, XMFLOAT4, XMFLOAT4X4(행우선 행렬) 등을 사용할 것.

- 정확성 대신 성능을 위해 근사 함수 버전을 사용
- 정렬된 데이터 형식과 연산을 사용
  Ex. XMFLOAT4X4A, XMStoreFLOAT4X4A

<DirectX Math>

DirectX 충돌 검사

- 충돌 구조체
  - BoundingBox (구조체: AABB)
  - BoundingOrientedBox (구조체: OBB)
  - BoundingSphere
  - BoundgingFrustum (frustrum: 절두체)
  -

<4. Device>

프레임 버퍼의 공유

- 1/60 초마다 프레임 버퍼의 내용을 바꾼다.
- 프레임 버퍼는 공유된다.

이중 버퍼링

- 2 개의 프레임 버퍼: 전면버퍼, 후면버퍼
  - ■ 전면버퍼: 항상 1 개. 현재 화면에 출력되는 이미지
  - ■ 후면버퍼: 여러 개 있을 수 있음. 다음에 출력할 이미지.
- 스왑 체인
  순차적으로 연결된 프레임 버퍼들의 집합
- 프리젠테이션
  후면버퍼의 내용을 전면버퍼로 옮기는 것
- 플리핑
  하드웨어적인 방법으로 전면버퍼와 후면버퍼를 바꾸는 방법

스왑 체인

- 전면버퍼는 항상 하나만 존재. 후면버퍼는 video memory 가 허락하는 한 여러 개 있을 수 있음. 여러 개 있으면 서로 분업을 함.

프레젠테이션

- 1. 렌더링 -> 2. 프리젠트 -> 3. 렌더링

COM 객체

- DLL 형태로 제공됨.
- 객체의 내부는 노출하지 않고, 호출할 수 있는 메소드 함수들만을 노출함.
- C++ 객체를 사용하는 방법과 유사하게 사용할 수 있음
- COM 객체를 사용하는 유일한 방법 = 노출된 메소드 호출
- 모든 D3D 객체는 COM 객체
- 프로그램에서 COM 객체는 인터페이스를 통해 참조
  ⇨ 실제로는 '인터페이스 포인터'로 참조함. 인터페이스 포인터는 C++의 포인터처럼 사용.

IUnknown 인터페이스

- 모든 COM 인터페이스는 IUnknown 인터페이스에서 파생됨.
- IUnknown::AddRef
  인터페이스의 참조 카운터를 1 증가시킴
  누군가가 이 객체를 사용하고 있음을 의미
- IUnknown::Release
  인터페이스의 참조 카운터를 1 감소시킴
  이 메소드를 호출한 객체가 더 이상 이 객체를 사용하지 않음을 의미
  (참조 카운터가 0 이면 아무도 이 객체를 사용하지 않는다는 것)

COM 객체의 생성과 소멸

- 객체의 참조 카운터가 0 이 되면 객체는 자동으로 소멸
  0 이 되었다는 것은 다른 객체들이 더 이상 이 객체를 사용 또는 참조하지 않는다는 의미
- COM 객체에 대한 인터페이스 포인터가 더 이상 필요 없거나 소멸시키고 싶으면 Release() 함수를 호출한다.

COM 객체의 생성

- 1. 포인터 변수 선언
- 2. 포인터 변수의 주소를 넘김

GUID

- 인터페이스 클래스 식별자(ID)를 나타내는 128 비트 정수 문자열
- __uuidof 연산자
  인터페이스 자료형, 클래스 이름, 인터페이스 포인터에 대한 GUID 를 반환

DXGI

- DirectX 그래픽 런타임에 독립적인 저수준(low-level)의 작업을 관리
- DirectX 그래픽을 위한 기본적이고 공통적인 프레임워크를 제공
- 새로운 그래픽 라이브러리(DirectX)가 나오더라도 변하지 않을 수 있도록 구성

- DXGI 어댑터, 출력장치, 디바이스

Direct3D 12

- 하드웨어에 더 밀접하여 더 빠르고 효율적.
  저수준 프로그래밍
  CPU 와 GPU 모두가 대기 시간을 최소화할 수 있도록 지원
- Direct3D 11 과의 차이점
  - 명시적 동기화
    CPU-GPU 동기화를 명시적으로 책임져야 함.
    가장 이상적인 CPU 와 GPU 의 상태: 둘 다 놀지 않는 상태

Direct3D 디바이스(== 그래픽 카드)

- Direct3D 디바이스는 상태 기계이다.
- D3D12 -> Set&Draw

IDXGIFactory::CreateSwapChain() 함수

- 렌더링을 하려면 스왑 체인(프레임 버퍼)이 필요
- Present 로 후면버퍼의 메모리 내용을 전면버퍼로 옮김
- ResizeBuffers, ResizeTarget 은 바탕화면의 해상도(전면 버퍼 크기) 조정

다중 샘플링

- 계단 현상
  모니터의 픽셀 크기가 무한히 작지 않기 때문에 나타남
  모니터 해상도를 늘리면(== 픽셀 크기를 작게 하면) 계단 현상이 줄어듦
- 계단 현상 제거 기법
  - 슈퍼 샘플링
  - 다중 샘플링
    MSAA
- Direct3D 12 디바이스는 4x 다중 샘플링 지원
  다중 샘플링은 시간이 많이 걸리는 연산이므로 하드웨어적으로 처리해야 함.

다중 샘플링

- 슈퍼 샘플링: 4X
  - 후면 버퍼와 깊이 버퍼 해상도를 화면 크기보다 4 배(2*2) 크게 만들어 렌더링
    렌더링 할 후면버퍼 픽셀 개수가 4 배가 됨(=> 렌더링 시간도 4 배)
  - Present 할 때 후면버퍼를 샘플링(예: 4 개 픽셀을 평균)하여 픽셀 색상을 구함

- 화면의 한 픽셀마다 후면버퍼에는 4 개의 서브 픽셀이 존재.
  각 서브 픽셀을 렌더링하고 필터링을 통하여 화면 픽셀을 계산
  각 서브 픽셀의 색상은 다를 수 있음
  ⇨ 많은 계산 필요!!

- MSAA: 4X
  - ■ 슈퍼 샘플링처럼 후면버퍼와 깊이 버퍼를 화면보다 4 배 더 크게 생성
    하지만 슈퍼 샘플링보다 각 픽셀 색상을 계산하는 시간이 적게 걸림
  - ■ 화면의 픽셀마다 화면 픽셀의 중심에서 한 번만 색상을 계산
    서브 픽셀의 색상은 서브 픽셀의 가시성과 포함 여부에 따라 결정됨
    - ◆ 가시성: 각 서브 픽셀이 보이는가
      가시성 판단을 위해 깊이/스텐실 검사가 각 서브 픽셀마다 수행됨.
    - ◆ 포함 여부: 각 서브 픽셀의 중심이 다각형 내부에 존재하는가
      포함 여부 판단은 각 서브 픽셀마다 수행됨

- 슈퍼 샘플링: 후면 버퍼의 각 서브 픽셀 색상을 계산
- 다중 샘플링: 화면의 픽셀 색상을 계산

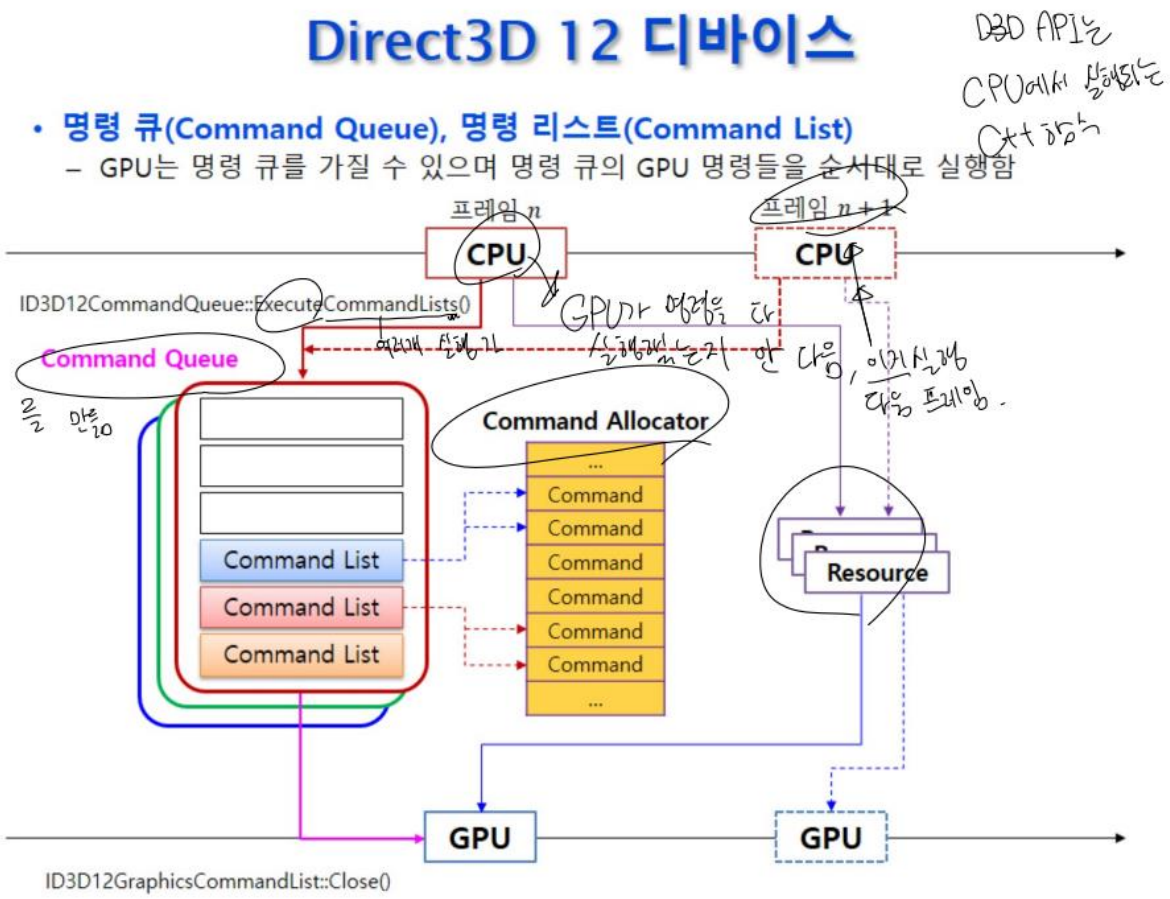
IDXGISwapChain

- 스왑체인을 만드는 목적
  우리의 그림이 전면버퍼를 통해 디스플레이에 나올 수 있도록 하기 위함

명령 큐, 명령 리스트

- Queue: FIFO
- GPU 는 명령 큐를 가지고 있으며, 큐의 GPU 명령들을 순서대로 실행함.

- Command Queue(명령 큐) 중 Copy Queue, Rendering Queue, Compute Queue: Rendering Queue 만 각각의 세 엔진을 다 사용할 수 있음



명령 리스트

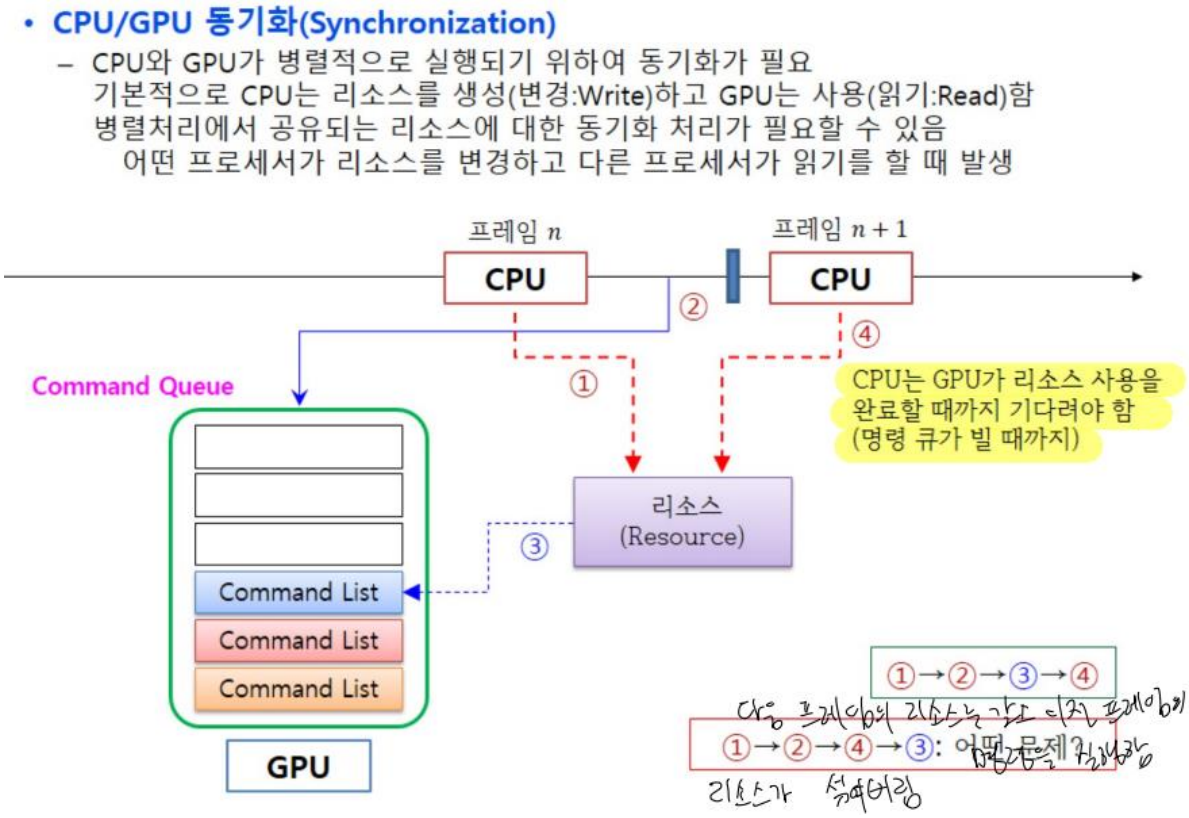
- 모든 명령의 실행은 GPU 가 함.
  Ex. 그림이 후면버퍼에 그려짐 -> 프리젠트 -> 모니터에 그려짐

명령 리스트의 실행

- 실행할 명령 리스트를 명령 큐에 추가하면 GPU 가 순서대로 실행
  명령 리스트는 추가된 순서대로 실행됨(FIFO)
  명령 리스트는 명령 큐에 추가되기 전에 반드시 닫힌(Clsoe) 상태가 되어야 함.
- Close(): 명령 리스트를 닫힌 상태로 만듦, Reset() 호출 전에 실행
- Reset(): 명령 리스트를 초기화 상태(Open)로 만듦

CPU/GPU 동기화

- CPU 와 GPU 가 병렬적으로 실행되기 위한 동기화
  CPU 는 리소스를 생성(변경: write)하고, GPU 는 사용(읽기: Read)함



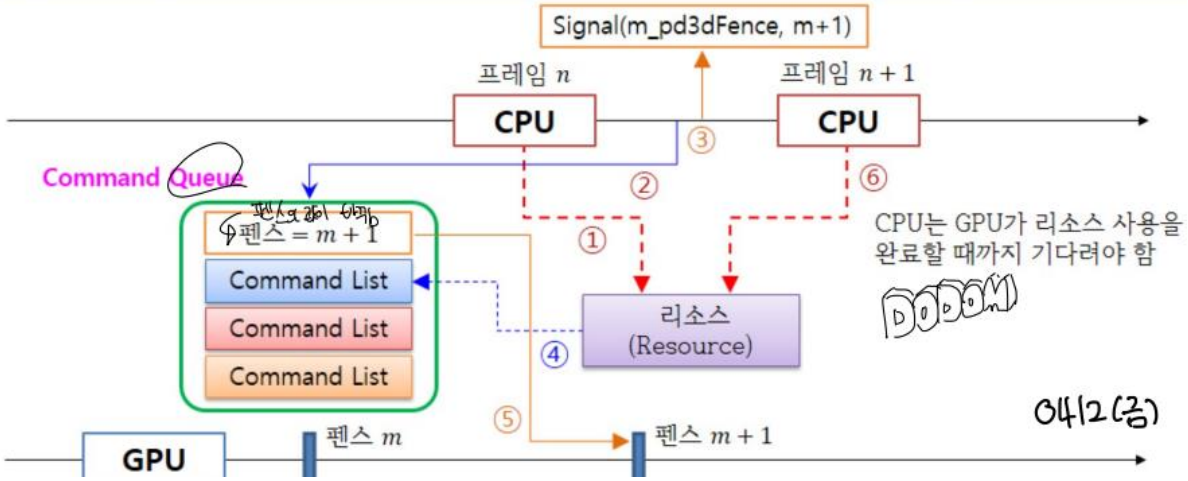
펜스 객체(ID3D12Fence 인터페이스)

- CPU 와 GPU 의 동기화를 위해 사용함.

ID3D12CommandQueue 인터페이스

• ID3D12CommandQueue 인터페이스

```
HRESULT ID3D12CommandQueue::Signal(
    ID3D12Fence *pFence,
    UINT64 Value //GPU가 펜스를 이 값으로 설정하는 명령을 명령 큐에 추가
);
HRESULT ID3D12CommandQueue::Wait(
    ID3D12Fence *pFence,
    UINT64 Value //명령 큐가 기다리는 펜스 값. 펜스의 현재 값이 이 값보다 크거나 같으면 기다림이 끝남
);
```



쉐이더 리소스

- 리소소는 렌더링 과정 동안 GPU 가 사용하는 비디오 메모리에 저장된 데이터임
- 텍스쳐와 버퍼
- 파이프라인
  - ■ 그래픽스 파이프라인
  - ■ 계산 파이프라인
- 리소스 뷰
  - ■ 렌더 타겟 뷰(RTV)
  - ■ 깊이 스텐실 뷰(DSV)

깊이 버퍼

- Direct3D 에서의 깊이 버퍼(depth buffer)는 3D 그래픽에서 깊이(거리)를 저장하고 관리하는 데 사용되는 버퍼입니다. 깊이 버퍼는 일반적으로

프레임의 각 픽셀에 대한 실제 3D 공간에서의 깊이 값을 저장합니다. 이는 렌더링되는 개체의 깊이를 추적하고, 깊이 테스트 및 깊이 기반의 픽셀 블렌딩과 같은 기술을 가능하게 합니다.

- 깊이 버퍼는 보통 렌더 타겟(render target)과 함께 사용됩니다. 렌더 타겟은 렌더링 작업의 결과를 저장하는 버퍼이고, 깊이 버퍼는 각 픽셀에 대한 깊이 정보를 저장합니다. 일반적으로 렌더링 작업이 수행될 때, 프레임의 각 픽셀에 대한 깊이 값이 깊이 버퍼에 기록됩니다. 이후에는 렌더링되는 개체의 깊이 값과 깊이 버퍼에 저장된 값 사이의 비교를 통해 깊이 테스트를 수행합니다.

<3-1. Graphics Pipeline1>

-Direct3D 12 파이프라인 <= 커다란 함수

Draw 함수로 동작한다.

고정 프로그램 단계(고정 파이프라인): Direct3D에서 모든 처리가 진행되며 응용 프로그램에서 변경할 수 없는 단계

프로그램 가능 단계(쉐이더 단계): 응용 프로그램에서 쉐이더 프로그램을 통하여 제공해야 하는 단계

입력 조립기(IA) ->

정점 쉐이더(VS) ->

헐 쉐이더(HS) ->

테셀레이터

도메인 쉐이더(DS) ->

기하 쉐이더(GS) ->

스트림 출력 ->

래스터라이저 ->

픽셀 쉐이더(PS) ->

출력 병합기(OM)

Direct3D 디바이스는 상태 기계이다.

-입력-조립 단계(IA: Input-Assembler Stage)

- 정점 버퍼의 정점 데이터를 다른 파이프라인 단계에서 사용할 프리미티브(선 리스트, 삼각형 리슽, 삼각형 스트립, 인접성을 가지는 프리미티브 등)로 조립
- 시스템 생성값(System-Generated Values)을 추가.
  - 시스템 생성 값은 시맨틱(Semantic)이라고 하는 문자열의 값. 다음 단계의 쉐이더에서 사용할 수 있음
- 정점 데이터를 프리미티브로 조립하고 시스템 생성 값을 추가하여 정점-쉐이더(VS)로 출력

-래스터라이저 단계

- 벡터 정보(프리미티브)를 래스터 이미지(픽셀)로 변환
- 래스터라이저 단계
  - 원근 투영 나누기(z 나누기)
  - 카메라 절두체를 벗어나는 점(픽셀)들을 클리핑
  - 프리미티브(벡터 정보)를 2차원 뷰포트로 매핑
  - 프리미티브의 모든 픽셀들에 대하여 픽셀-쉐이더(PS) 호출
- 래스터라이저 단계에서 픽셀의 속성 계산
  - 보간: 선형 보간, 색상 보간, 다른 데이터도 보간함
- 프리미티브를 구성하는 각 픽셀에 대한 연산을 수행하고 깊이값을 출력

-출력-병합 단계(OM: Outpu-Merger Stage)

- 최종적으로 픽셀의 색상을 생성하여 렌더 타겟으로 출력하는 단계.
  파이프라인 상태(State) 정보, PS가 생성한 픽셀 색상, 렌더 타겟의 내용, 깊이/스텐실 버퍼의 내용들을 조합하여 출력할 색상을 결정.
- 깊이-스텐실 검사

- 픽셀이 그려져야 하는 지를 결정하기 위해 깊이 값과 스텐실 값 데이터를 사용
- 깊이 검사
  출력 픽셀의 깊이 값을 깊이 버퍼의 같은 위치의 깊이 값과 비교.
  비교 결과에 따라 출력 픽셀을 렌더 타겟에 출력하거나 하지 않음.
- 블렌딩
  픽셀 값(색)들을 결합하여 하나의 최종 픽셀 색상을 생성하는 과정.
  렌더 타겟의 픽셀 색상과 출력(픽셀 쉐이더) 색상을 결합함.
  깊이 검사를 통해 불투명한 걸 먼저 그리고 투명한 걸 그려서 Blending함.

- OM: 각 픽셀에 대한 깊이/스텐실 검사와 블렌딩 연상을 수행하여 최종 출력 색상을 결정함.

-쉐이더 단계

- 공통-쉐이더 코어를 가짐
  메모리 리소스에는 상수버퍼/텍스쳐/버퍼가 있음
  - 상수 버퍼
    - 이 파이프 라인이 끝날 때까지 상수의 의미를 가짐
    - 버퍼의 내용이 자주 바뀌지 않는다는 의미
  - 텍스쳐
    - 배열 또는 이미지의 집합 등
- 샘플러라는 게 필요함
- 쉐이더 모델4
  하나의 쉐이더에 16개의 샘플러를 연결 가능
  하나의 쉐이더에 128개의 텍스쳐와 버퍼 연결 가능
  하나의 쉐이더에 16개의 상수 버퍼를 연결 가능

-정점 쉐이더 단계(VS)

- IA 단계에서 출력되는 프리미티브의 각 정점에 대한 연산 수행
  (변환, 스키닝, 모핑, 조명 등)
  - VS에서 조명 계산하는 게 좋다. 삼각형 하나에 대해서 조명 계산을 3번만 하면 됨. PS에서 하면 조명 계산 횟수가 가변적이라 별로임.
- 하나의 정점에 대하여 한 번 호출되며 하나의 출력 정점을 생성.
- 파이프라인 단계에서 항상 수행되어야 함.
- VS의 입력 정점은 16개의 32-비트 벡터(4-요소 벡터)까지 구성 가능
- VS의 출력 정점은 16개의 32-비트 벡터(4-요소 벡터)까지 구성 가능
- 각 정점에 대한 연산을 수행. 최소 하나의 입력과 출력을 가짐

-픽셀-쉐이더(PS)

- 각 픽셀의 데이터(기본적으로 색상)를 생성
- 하나의 프리미티브를 구성하는 각 픽셀에 대하여 PS를 한 번씩 호출함.
- 다중-샘플링을 사용해도 PS는 하나의 픽셀에 대해 한 번만! 호출됨.
- 8개의 32-비트 4-요소 색상을 출력할 수 있음
  픽셀을 버리면(discard) 색상과 깊이 값이 출력되지 않음.
- OM에서의 깊이 값 검사를 위해 깊이 값 시맨틱을 출력할 수 있음
- 프리미티브를 구성하는 각 픽셀에 대한 연산을 수행하고 깊이 값을 출력

시맨틱(Semantics)

- 쉐이더의 입력 또는 출력 매개변수에 부착되는 문자열
  매개변수의 사용의도를 나타내는 정보를 컴파일러에게 전달하기 위함
- 임의의 시맨틱이 허용됨.
- 시스템-값 시맨틱(SV Semantics): 특별한 의미를 갖는 정해진 시맨틱