**변환 파이프라인**

3D로 표현된 모델 좌표계의 정점을 입력하면 컴퓨터 화면에 그릴 수 있도록 2D 좌표로 변환하는 함수

월드변환 : 모델좌표계의 정점과 게임객체를 입력으로 받아 정점좌표를 월드 좌표계로 변환

카메라 변환 : 월드 좌표계의 점을 카메라 좌표계로 변환

투영 변환 : 카메라 좌표계의 점을 투영 좌표계로 변환

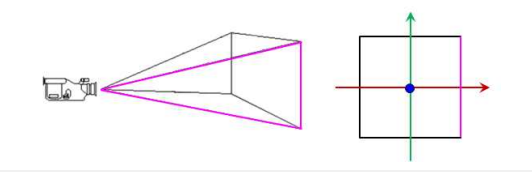
화면 변환 : 투영 좌표계의 점을 화면 좌표계로 변환

1. 인스턴싱이란? : 여러 게임 객체들이 같은 외관을 가지면 각 게임 객체가 모델을 따로 가질 필요 없이 하나의 모델을 서로 공유하는 것.
2. 평행이동 변환과 회전 변환의 적용 순서 회전 변환은 기본적으로 좌표계의 원점 또는 원점을 지나는 좌표축을 기준으로 하는 것으로 가정한다.

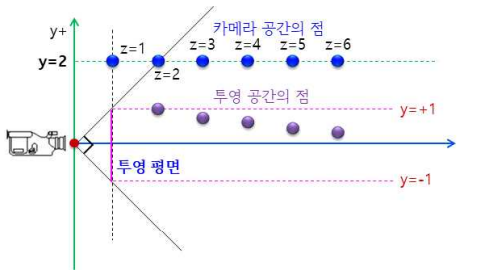
평행이동 변환과 회전 변환이 같이 적용될 때 변환을 하는 순서가 중요하 다. 평행이동 변환을 먼저 하고 회전 변환을 나중에 하는 결과와 회전 변환을 먼저 하고 평행이동 변환을 나중에 하는 결과는 다르다.

1. 카메라 변환(Camera Transformation) 카메라를 이동하고 회전할 때 주의할 점은 카메라의 움직임과 화면상의 게임 객체의 움 직임은 방향이 서로 반대라는 것이다. 예를 들어, 카메라를 앞으로 움직이면, 화면상에서 게임 세계가 전체적으로 카메라의 이동 방향과 반대로(카메라에 가까워지는 방향으로) 이동 하는 것처럼 보이게 된다. 이것은 카메라를 움직이지 않고 게임 세계의 게임 객체들을 카 메라 이동 방향과 반대 방향으로 이동하는 것과 화면상의 결과와 같다. 또한, 카메라를 왼 쪽으로 이동하면 화면상에서 게임 세계는 오른쪽으로 이동하게 된다. 카메라를 회전하는 경우도 화면상에서 게임 세계는 반대 방향으로 회전을 한다
2. 원근 투영 변환 : 가장 쉬운 예는 그림자이다. 게 3차원 점(좌표)을 2차원 점(좌표)으로 변환하는 과정을 투영(Projection)이라고 한다. 즉, 3차원 게임 객체들을 2차원 화면에 렌더링하였을 때, 화면에 그려진 그림(사진)을 보고 입체감 또는 거리감을 느낄 수 있도록 그려야 한다. 원근 투영 나누기를 하는 것은 3차원 좌표계의 점을 2차 원 좌표계의 점으로 변환하는 것이다.
3. 투영 사각형

카메라에 보이는 게임 세계의 게임 객체들은 카메라의 2차원 사각형 영역으로 원근 투영 될 것이다(실세계의 사진기 또는 디지털 카메라를 생각해보라). 이 사각형의 중심은 카메라 의 중심(카메라 좌표계의 원점)이 되고, 사각형의 오른쪽은 카메라 좌표계의 +(x-축)이 되 고, 사각형의 위쪽은 카메라 좌표계의 +(y-축)이 된다. 이 사각형은 투영 사각형 (Projection rectangle)이라고 한다.

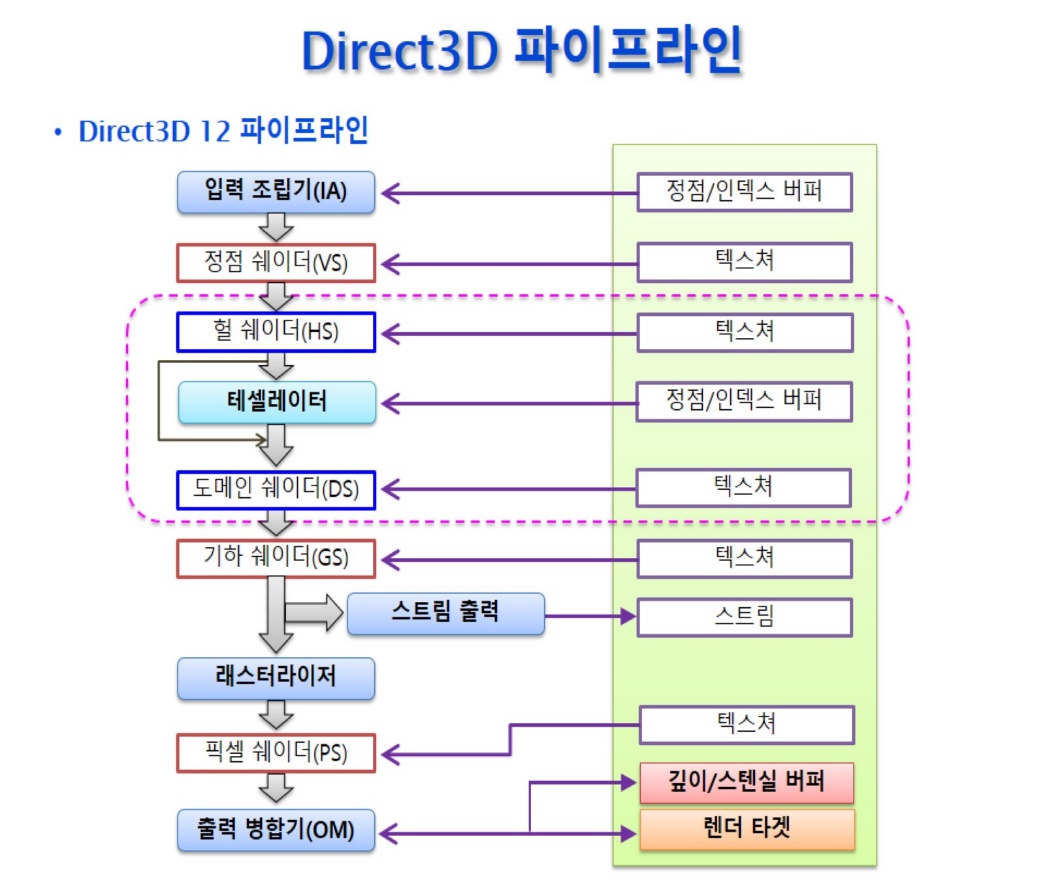


1. 투영평면 (원근감 표현은 어떻게?)



Z 좌표가 클수록 원근 투영의 결과는 z축에 점점 접근하여 소멸된다.

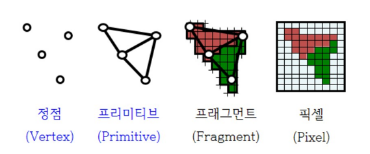
**Direct3D 파이프라인 – ‼(자주 물어보는 유형)**

1. Direct3D 12 파이프라인 순서 외우기
2. 파이프라인이란 ? GPU를 사용해서 정점/텍스쳐등 리소스를 2D 이미지로 렌더링(그리는)과정

프로그램 가능단계와 고정 프로그램 단게로 구분한다.

* 고정프로그램단계 -> IA (입력조립기) / TS (테셀레이터) / SO(스트림출력) / RS(레스터라이저) / OM(출력병합)
* 프로그램 가능 단계 -> VS/ HS/ DS/ GS/ PS [S가 붙은 Shader들만 가능!]

1. 각 단계들마다 뭘하는지?



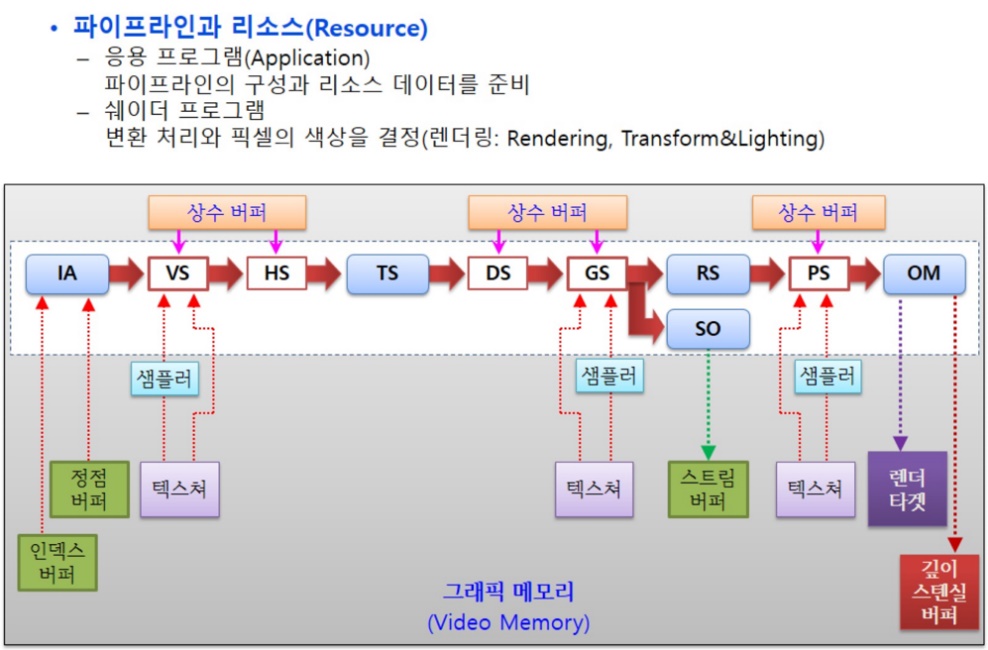
* IA 단계 : 정점 데이터를 프리미티브(벡터정보)로 조립하고 시스템 생성값을 추가해서 VS로 출력
* RS 단계: 프리미티브를 구성하는 각 픽셀에 대한 연산을 수행하고 깊이 값을 출력. 픽셀의 속성은 선형보간을 통해 계산
* SO 단계 : 각 픽셀에 대한 깊이-스텐실 검사와 블렌딩 연산을 수행하여 최종 출력 색상을 결정
* \* 블렌딩 : 픽셀 값 들을 결합(블렌딩)하여 하나의 최종 픽셀 색상을 생성하는 과정
* 쉐이더 단계

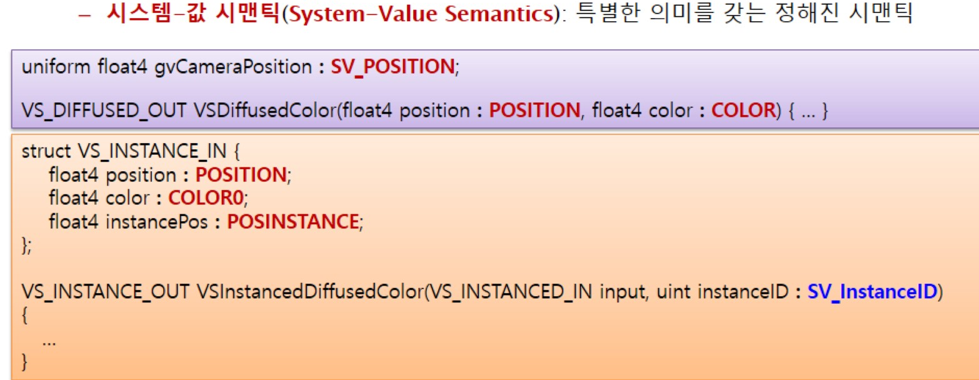
VS 단계 : 각 정점에 대한 연산을 수행, 최소 하나의 입출력을 가짐.

렌더링 파이프라인 단계에서 항상 수행되어야 함

PS 단계 : 프리미티브를 구성하는 각 픽셀에 대한 연산을 수행하고 깊이 값을 출력, 기하 쉐이더(GS)단계가 없으면 PS입력값은 16개 32비트 벡터, 있다면 32개 32비트 벡터

정리하자면..



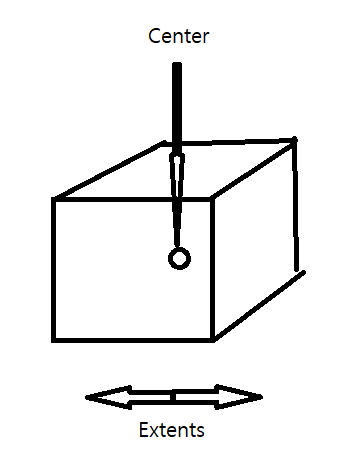
1. 시맨틱

쉐이더의 입력 또는 출력 매개변수에 부착되는 문자열 매개변수의 사용 의도를 나타내는 정보를 컴파일러에게 전달하기 위하여 사용

1. 각 단계 준비하기



**충돌검사**

Bounding Box 구조체는 바운딩 박스의 중심이 Center이고 대각선이 Extents인 AABB 바운딩 박스이며, 다음과 같은 멤버 함수를 제공한다. Bounding Box는 구조체이므로 구조 체 객체를 통하여 멤버 함수를 호출함에 주의하라

꼭지점 반환함수, 포함 함수, 교차 함수등을 통해 처리한다.

**-리소스**

**Descriptor Heap**

**모든 리소스들을 서술자로 표현하기 위해 필요한 메모리 블록**

**서술자의 주소를 표현하기위해 cpu gpu 서술자 핸들이라는 구조체를 사용**

**d3d는 d3d가 제공하는 데이터 타입에 대해 c++형태로 좀더 편하게 사용할 수 있도록**

**CD3D12 -> 를 통해 struct를 익숙한 class형식으로 데이터 타입을 정의해 제공한다.**

**\*\*OFFSET -> 서술자를 저장하는 메모리 블록의 시작주소에 offset을 더해 각각의 핸들들의 위치를**

**내부에서 처리해준다.**

**우리가 View를 그리는 상황에 대해서 다음 순서를 따른다.**

**리소스 들이 필요**

**각각의 리소스들의 타입에 맞는 서술자를 위한 Heap을 확보해야함**

**여기서 Heap은 각각의 리소스들의 서술자를 담을 수 있는 배열임**

**Creat View 를 수행할 때 리소스에 해당하는 서술자가 인터페이스를 제공해주고**

**그를 바탕으로 View를 Heap에 생성**

**Set View를 통해 Heap에 있는 뷰를 가져가 설정할 수 있다.**

**그 다음 원하는 파이프라인을 셋한다.**

**Resoucres -> heaps -> CreateView -> SetView -> PipeLine**

**DEVICE / command List**

**하나의 리소스를 통해 여러가지 뷰를 셋팅이 가능한데 가령**

**텍스처 리소스를 쉐이더 렌더 뷰와 랜더 타겟 뷰로 나눈다고 했을때,**

**이 두 뷰는 텍스처 리소스의 서술자를 해석하는 방식이 다르다.**

**그래서 각각의 뷰 구조체에 DXGI FORMAT이라는 원소가 해석하는 방식에 대한 정보를 담는다.**

**DIMENSION -> 이 리소스를 버퍼로 이해할 것인지 텍스처로 이해할 것인지에 대한 설정**

**이 리소스에 접근하기 위한 추가적인 옵션들을 나타냄.**

**디멘션 타입에 따라 union멤버로 추가적인 구조체 내용을 리소스마다 적어놓는다.**

**VIew는 종류에 따라 서술자 Heap을 따로 만들어야 한다.**

**Render Desc**

**Depth Desc**

**둘의 CreateView는 거의 유사하나 Flag와 Format의 형식에서 약간의 차이를 볼 수 있다.**

**Depth버퍼가 도대체 왜 필요한가..?**

**우리는 원근 투영 변화를 통해 depth 값을 0 과 1사이로 노말라이즈 한 후**

**이 범위에 해당하는 객체만 출력하게 한다.**

**여기서 만약에 3개의 물체가 일직선 상에 존재한다고 했을때**

**점을 어떻게 출력해야 하는가..?**

**depth는 같은 점으로 투영되는 절두체 내의 모든 점들의 카메라로 부터의 실측 거리가 아니라 상대적인**

**거리를 나타낸다고 볼 수 있다.. 그렇다면 가장 가까운 위치에 해당하는 점이 찍히게 될 것이다.**

**--> Painters Algorithm 이것을 수행하려면 멀리있고 가까움에 대한 값이 필요하고**

**이 값들에 대한 정렬이 필요하다.. 근데 이러한 작업은 각각의 점들에 대해 모두 정렬을 수행해야 하는데**

**이는 너무 비효율적이다...**

**3개의 오브젝트를 그리는 경우의 수는 3! 만큼 존재할 것이다. 6가지 경우가 있다.**

**어떤 순서로 그리든 상관없이 최종적으로 투영되는 점을 정하기 위한 알고리즘을 만들기 위해 이 depth값을 깊이버퍼에다가**

**저장하는 것이다.**

**이를 depth buffer algorithmn이라고 한다. OM이 하는 가장 중요한 일 중하나가 depth Test**

**그렇기에 depth buffer view를 만들어 OM에 설정**

**그래서 Z-Fighting**

**리소스**

**1. place resource**

**2. reserved resource**

**-------- 3번만 사용할 것임.**

**3. commited resource**

**리소스를 제작했다. -> 서술자 힙에다가 뷰로 만들어서 파이프라인으로 Set을 한다.**

**Set -> grahpic command list에 존재**

**OM set을 통해 OM으로 연결.**

**IA -> 리소스를 set 해주고**

**파이프라인 수행후**

**OM ->여기에 뷰가 연결되어 랜더 타겟에는 색깔을 리턴**

**깊이 뷰에는 깊이 버퍼를 리턴**

**리소스 생성과정에 대해 알아보자**

**1. heap properties ->**

**-> Type 할당된 메모리(heap)의 타입 -> cpu와 gpu가 리소스를 사용하는 용도**

**1. default -> gpu가 r/w가 가능하지만 cpu는 접근 불가 대부분의 리소스가 이 특성을 가진다.**

**파이프라인에서 작업하기 때문..**

**2. upload -> cpu가 w할 수 있는 접근 권한을 가지고 있다. 대부분의 게임에 대한 리소스는 응용 프로그램이**

**가지고 있다. 이 데이터들을 비디오 메모리로 옮겨야하는데 gpu가 비디오 메모리에 대한 접근 권한을**

**가지고 있기에, 이 heap 타입을 default로 하면 cpu의 리소스 데이터를 사용할 수 없다. 그렇기에 default**

**힙을 만드려면 upload 힙이 매개로 필요하다. upload 힙에다가 응용 프로그램의 리소스를 write하고**

**upload에 있는 리소스를 default 힙으로 copy해서 gpu가 일하게 한다.**

**동적인 데이터를 관리하기 위해서도 사용이 가능하다.**

**3. readback -> cpu에서 리소스 데이터를 응용 프로그램 쪽으로 다시 읽어오기 위한 타입.**

**정말로 필요하지 않으면 사용하지 않는다. -> 속도가 존나 느림.**

**-> page properties 메모리 페이지의 목적**

**1. unknown -> gpu에서 알아서 설정**

**-> memory pool -> 메모리 풀 타입**

**크게 신경 쓸일이 없다.**

**-> nodemask -> 단일 gpu 인가 아닌가**

**-> visible node mask -> 다중어댑터인 경우**

**2. Flags ->여러 어댑터를 사용하지 않는 이상 별도로 세팅하지 않는다.**

**None을 디폴트로 사용.**

**3. ClearValue 리소스 메모리를 할당하고 효율적으로 초기하기 위한 값을 세팅**

**DXGI format -> 초기화할 포맷**

**union 멤버 -> 초기화할 값들**

**초기화할 값을 정해주지않으면 실행은 되지만 속도가 매우 느려진다.**

**3. ResourceState**

**리소스를 생성시 그 리소스를 사용하는 작업에 의해 상태가 바뀌게 되는데**

**그 상태값 중 리소스 생성시의 초기상태를 열거형 값으로 정의해줄 수 있다.**

**4. 리소스 서술자**

**->dimension -> 리소스가 버퍼(메쉬)인지 텍스처(그림 데이터 1~3차원)인지 구별**

**-> dimension에 따라 가로와 세로 크기를 지정 width height**

**만약 텍스처 2d라고하면 가로픽셀 세로 픽셀의 수**

**3d라고하면 가로 세로 픽셀 + z 축의 크기인 depth or arraysize**

**-> alignment 리소스를 할당받는 메모리의 위치가 64kb 단위 위치에 존재**

**통상적으로 0를 쓰면 디바이스가 알아서 정해줄 것이다.**

**-> mip level 밉맵 레벨**

**-> format**

**-> 다중 샘플링에 위한 샘플 서술자**

**이렇게 리소스를 생성하면 이 리소스의 주소를 이용해**

**인터페이스를 만든다.**

**루트 시그너처**

**루트 파라미터들의 집합이라고 볼 수 있다.**

**64개의 32비트의 value들의 타입으로 루트 파라미터로 표현한 것**

**루트 파라미터 -> 쉐이더에게 전달되어야하는 글로벌 변수들**

**루트 파라미터 종류**

**루트 상수 🡪 루트 매개변수 자체에 직접 값들을 써놓는 것 그 값이 쉐이더 변수로 전달.**

**서술자 힙 필요없이 가장 빠른 방법으로 파라미터 전달 가능.**

**루트 서술자 🡪 32비트 자리 2개리를 사용해서 루트 서술자를 표현 -> 서술자 힙을 만들 필요 없음.**

**버퍼 리소스에 대한 srv/uav 상수 버퍼에 대한 cbv를 표현**

**루트 서술자 테이블🡪 서술자를 만들고 이 서술자를 루트 서술자 테이블에다가 저장을 한다. 그 루트 서술자 테이블의 주소를 루트 시그너처에다가 담아 놓는 것.**

테이블이(가) 표시된 사진

자동 생성된 설명

**0 번 파라미터 : uint 자료형**

**1 번 파라미터 : float(32비트)**

**2 번 파라미터 : 서술자 테이블의 주소가 있고 이 서술자 테이블에 서술자들이 저장되어 있음.**

**4번,5번 파라미터: 루트 서술자. 2자리를 차지하므로 4번 6번 이라고 봐야함 사실**

**옆에 영어 기호**

**상수 버퍼란 4096개의 버퍼를 저장할 수 있는 특별한 메모리 공간. 이 상수 버퍼는 동시에 4096개의 벡터들을 쉐이더 변수들로 맵핑할 수 있는 기능이 있음.**

**상수 버퍼의 사이즈는 항상 256바이트의 배수여야함.**

**cpu에서 이 상수 버퍼의 값들을 자주 바꿀 것이다라는 것을 내포하고 있기에 upload heap에다가 생성한다. 그리고 항사 ㅇgenric read state로 설정함.**

**이 상수버퍼들을 쉐이더 변수로 맵핑하기 위해서 cBuffer라는 키워드로 변수를 선언해야한다.**

**이 상수 버퍼들은 register라는 메모리에 저장된다.**

**B : 상수 버퍼**

**U : 무순서화 접근 뷰**

**S : 샘플러**

**T : 텍스처**

**저 기호 뒤에 나오는 숫자가 해당 뷰의 번호이다.**

**각 쉐이더 단계마다 14개의 상수버퍼 뷰를 연결 가능**

**Begin end 사이에 그 상수 버퍼안에 있는 변수들을 글로별 변수로 선언한다..**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Space X(숫자): 상수버퍼의 논리적인 공간에 대해서 설명**

**루트 시그너처 desc를 통해 루트 파라미터의 배열들을 선언하고 루트 파라미터로 서술자 테이블 , 루트 서술자, 루트 상수의 파라미터를 설정한다**

**Stuct RootParameter {**

**타입 : 루트상수 (레지스터 번호, 공간 32비트 상수 개수)**

**루트 서술자 (레지스터 번호 / 공간)**

**루트 서술자 힙 (레인지 )**

**}**

**예시**

**루트 파라미터들의 배열을 선언** 🡨 **쉐이더에는 6개의 루트 파라 미터들을 응용프로그램을 통해 넘겨줘야함.**

**0번 파라미터는 파라미터 타입이 상수야 이 파라미터는 모든 쉐이더에 대해 볼 수 있어, 레지스터는 2번을 사용해.(b2)**

**아무튼 여차저차 루트 파라미터를 세팅했다고 처**

1. **병렬화한 루트 시그너처의 blob을 가지고 루트 시그너처를 생성**
2. **그래픽 파이프라인 상태를 설정할 때 루트 시그너처에다가 연결.**
3. **커맨드 리스트에서 랜더링 과정 이전에 setgrahicrootsignature 메소드를 호출 해서 등록해줘야 한다.**