※ 2D게임과 3D게임의 차이
2차원과 3차원의 차이
리소스 차이(2D는 스프라이트 이미지, 3D는 모델링/리깅/애니메이션 필요)
그림자의 유무

※ 모델 좌표계와 월드 좌표계
모델 좌표계: 3D 모델이 가지고 있는 X/Y/Z 좌표계
월드 좌표계: 월드 공간이라고 함, 3D 장면 내의 모든 객체가 공유하는 공통 참조 프레임

※ 변환 파이프라인
모델 좌표 -> 월드 변환 -> 카메라 변환 -> 투영 변환 -> 화면 변환 -> 화면 좌표

정점 처리: 모델 좌표계에서 월드 좌표계로 변환을 통해 정점이 모델 공간에서 정점 공간으로 변환
기하학적 처리: 테셀레이션, 정점셰이딩, 기본 어셈블리
클리핑: 뷰 절두체 외부에 있는 형상 부분 제거
투영: 정점을 뷰 공간에서 클립 공간으로 변환, 3D 장면을 2D 평면에 매핑
뷰포트 변환: 뷰포트의 크기와 위치를 고려해 클립 공간 좌표를 화면 공간 좌표로 매핑
래스터화: 매쉬를 픽셀 셰이더 단계에서 처리되는 조각이나 픽셀로 변환
픽셀 처리: 매쉬를 음영처리하여 최종 색상과 모양을 처리

※ 정점 와인딩
GL_CW(시계방향): DirectX
GL_CCW(반시계방향): OpenGL

※ 인스턴싱
모델 정보를 가지고 있는 하나의 정점 버퍼와 인스턴스 버퍼라는 각 복사본의 변경점들만 담은 두 번째 버퍼를 사용하여 위의 문제를 해결
인스턴스란?: 클래스의 정의를 통해 만들어진 객체

※ 스칼라 삼중곱
두 벡터의 외적을 나머지 벡터와 내적한 것
기하학적 의미: 스칼라 삼중곱의 절댓값은 세 벡터로 정의되는 평행 육면체의 부피로 정의
스칼라 삼중곱이 0이면 세 벡터는 모두 동일한 평면 위에 있음

※ 월드 변환 행렬(Fundamental pdf 86p 참고)
월드 좌표계에서 객체의 위치와 방향(Right, Up, Look) 포함
단위행렬인지 아닌지에 상관없이 월드 변환 행렬의 3개 행은 로컬 좌표계의 x, y, z축을 나타내는 단위 벡터
월드 좌표 정점 = 모델 좌표 정점 * 월드 변환 행렬(m_WorldMatrix)
Translate, Scale, Rotate

※ 카메라 변환 행렬(Fundamental pdf 88p 참고)
카메라의 위치와 방향을 나타내기 위하여 사용되는 행렬
정점을 카메라 좌표계로 변환하는 것, 카메라를 월드 좌표계의 원점으로 옮기고 카메라의 로컬 좌표축을 월드 좌표축과 일치시키는 것은 카메라를 위한 월드 변환을 반대(역행렬)로 적용하는 것
D3DXMatricInverse(), D3DXMatricLookAtLH()

※ 투영 변환 행렬 사용 이유
현실감
깊이 인식
깊이의 환상
공간 관계
시각적 미학

※ DirectX Math 함수
32비트 실수 또는 정수 요소들의 벡터를 표현하기 위한 자료형
벡터와 행렬을 저장하기 위한 자료형
클래스/구조체 멤버로 직접 사용하는 것은 피할것(캡슐화 및 추상화, 유연성 및 확장성)
벡터와 햄렬 로드/저장 함수
벡터 초기화 함수
색상 함수
스칼라 함수
벡터 산술 함수
벡터 비트 연산 함수
벡터 비교 연산 함수
3D 벡터 비교 함수
벡터 컴포넌트 연산 함수
벡터 기하 함수
초월 함수
3D 백터 기하 함수
3D 벡터 변환 함수
벡터 템플릿 함수
유틸리티 함수
전환 및 기타 함수
행렬 함수
평면 함수
사원수 함수

※ 충돌 구조체
BoundingBox(축에 평행하지 않은 직육면체)
멤버: Center, Extents, CORNER_COUNT, CORNERCOUNT = 8
BoundingOrientedBox(축에 평행한 직육면체)
멤버: Center, Extents, Orientation(쿼터니언), CORNERCOUNT = 8
BoundingSphere(구)
멤버: Center, Radius
BoundingFrustum(사각뿔)
멤버: Origin, Orientation, Near, Far, LeftSlope, RightSlope, TopSlope, BottomSlope, CORNERCOUNT = 8
클래스를 만들 때 멤버 변수로 바운드 박스 생성
객체의 애니메이션 함수를 호출할 때, 바운드 박스의 상태를 업데이트
Scene에서 충돌 처리 함수를 호출하면서 충돌을 확인

※ 절두체 컬링
카메라의 시야범위에 포함되는 것들만 렌더링하고, 나머지 것들은 렌더링 하지 않는 기법

※ 프레임 버퍼
모니터로 출력되어야 하는 비디오 메모리의 영역
프레임: 화면에 출력되는 하나의 장면
프레임 레이트: 1초 동안 출력되는 평균 프레임의 수

※ 이중 버퍼
렌더링된 영상을 저장하기 위한하나 이상의 버퍼를 제공
스왑 체인: 순차적으로 연결된 프레임 버퍼들의 집합
프레젠테이션: 후면버퍼의 내용을 전면버퍼로 옮기는 것
플리핑: 하드웨어적 방법으로 전면버퍼와 후면버퍼를 바꾸는 방법
블리트: 버퍼의 내용 복사
HRESULT IDGISwapChain::GetBuffer(~); <- 스왑 체인의 후면 버퍼를 반환

※ COM 객체와 인터페이스
DLL의 형테로 제공
객체의 내부는 노출하지 않고 호출할 수 있는 메소드 함수들만을 노출
일반적인 C++ 객체 사용 방법과 유사, 생성과 소멸이 다름
모든 다렉 객체는 COM 객체
프로그램에서 COM은 인터페이스를 통해 참조, 실제로는 인터페이스 포인터를 참조, 인터페이스 포인터는 C++ 포인터처럼 사용 가능
객체의 참조 카운터가 0이 되면 그 객체는 자동 소멸
다렉 객체는 생성을 위한 API 함수가 별도 존재, new 연산자 사용X
다렉은 delete가 아닌 Release()함수를 호출해 소멸

주의사항
① COM 객체를 생성하기 위한 별도 함수 존재
② COM 객체에 대한 인터페이스 포인터를 다른 변수로 복사할 때 반드시 AddRef() 호출
③ COM 객체에 대한 인터페이스 포인터를 소멸시키고 싶으면 Release() 호출

COM 객체의 생성
① 포인터 변수 선언
② 포인터 변수의 주소 넘김

COM 객체가 아닌 경우
① 변수 선언 또는 메모리 할당
② 주소를 넘김

※ Comptr
인터페이스 포인터의 소멸자에서 Release()를 호출하는 스마트 포인터

※ DXGI
DirectX 그래픽 런타임에 독립적인 저수준의 작업 관리(하드웨어 디바이스 열거, 모니터 열거, 전체화면 등)
DirectX 그래픽을 위한 기본적이고 공통적인 프레임워크 제공
새로운 그래픽 라이브러리가 나오더라도 변하지 않을 수 있도록 구성

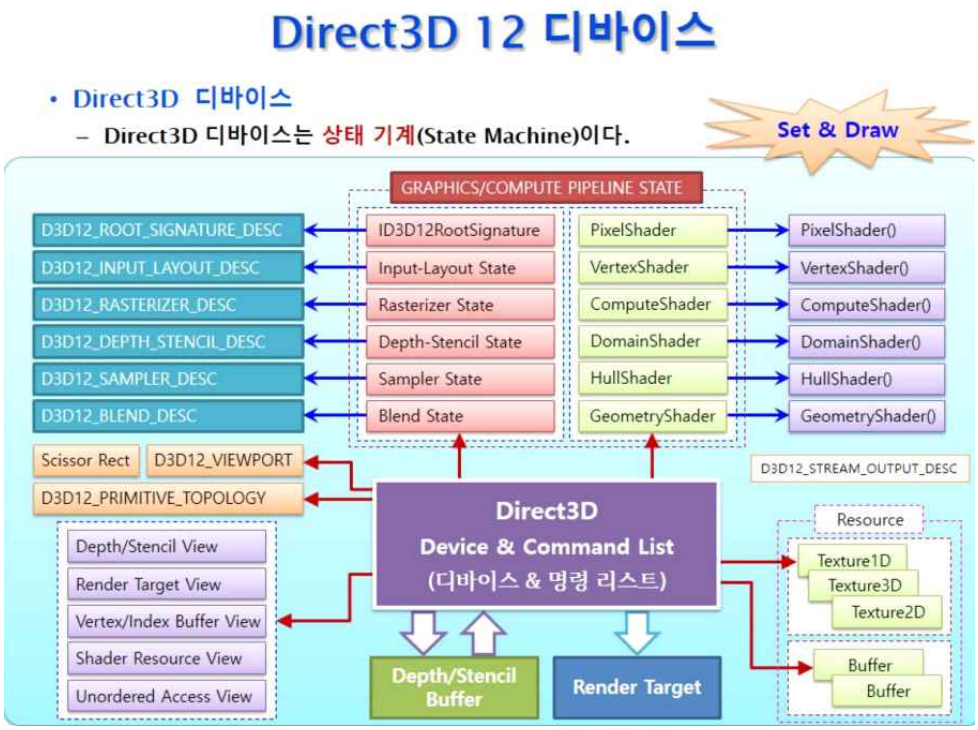
응용 프로그램 -> D3D10/11/12 -> 사용자-모드 드라이버 -> DXGI -> 커널-모드 드라이버 -> 하드웨어

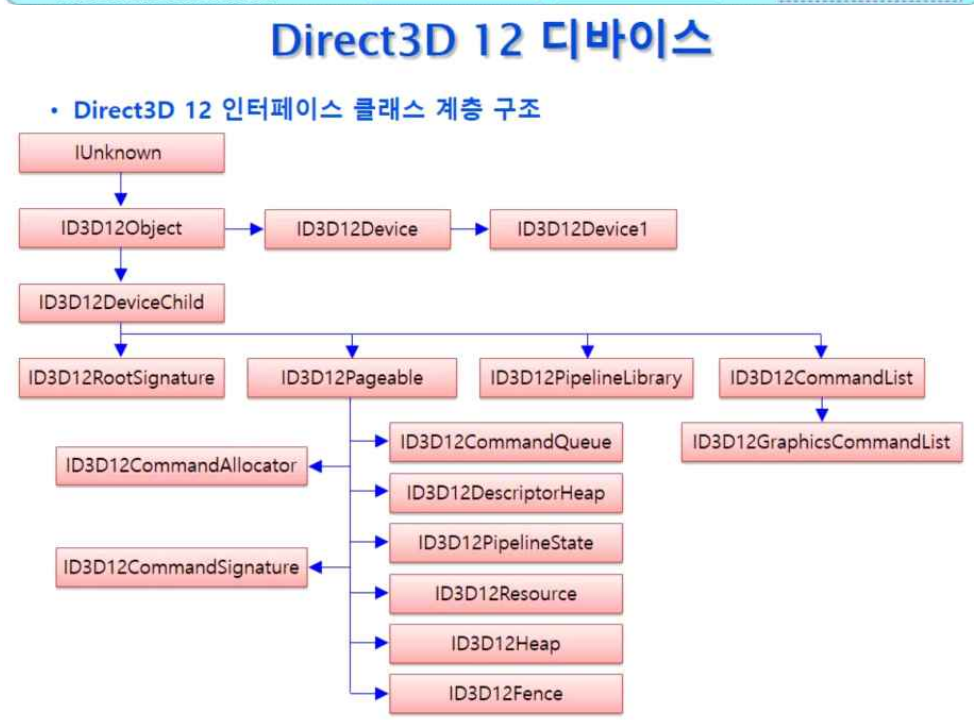
※ Direct3D Device 세팅
다렉 장치 초기화
리소스 생성
뷰 생성
루트 서명 생성
CommandList 생성
명령 실행

CreateGraphicsRootSignature(): 렌더링 중 GPU 쉐이더가 예상하는 루트 매개변수의 레이아웃을 정의하는 그래픽 루트 서명을 만드는 데 사용
CreateInputLayout(): 렌더링 중 GPU에 전달되는 정점 데이터 구조를 설명하는 입력 레이아웃 생성
CreateSwapChain(): 렌더링된 이미지를 화면에 효율적으로 표시하기 위한 백버퍼인 스왑 체인을 생성
CreateDirect3DDevice(): GPU와 상호작용하기 위한 기본 인터페이스인 기본 Direct3D 장치를 생성

※ view(SRV, CBV, UAV)
SRV(Shader Resource View): 읽기 전용 액세스를 위해 텍스처, 버퍼 및 구조화된 버퍼와 같은 리소스를 셰이더 단계에 바인딩하는 데 사용
CBV(Constant Buffer View): 셰이더 상수 및 매개변수가 포함된 상수 버퍼를 셰이더 단계에 바인딩하는 데 사용
UAV(Unordered Access View): 읽기-쓰기 액세스를 위해 리소스를 바인딩하는 데 사용되며 셰이더가 버퍼와 텍스처를 읽고 쓰기 가능

※ Command
명령 개체는 GPU에 렌더링 명령을 보내는 데 사용
여기에는 기본 요소 그리기, 렌더링 상태 설정 및 컴퓨팅 셰이더 실행을 위한 명령이 포함
명령 개체는 'ID3D12GraphicsCommandList' 인터페이스를 사용하여 생성되며 명령 목록 내에 기록





※ 다중 샘플링
계단 현상을 제거 위한 기법
모든 다렉 디바이스는 모든 DXGI 형식에 대한 4X 다중 샘플링 지원
시간이 많이 걸리는 연산이므로 하드웨어적 처리해야 함
구조체: DXGI_SAMPLE_DESC
멤버: Count - 픽셀마다의 샘플 개수(1: 다중 샘플링 안함), Quality - 품질 레벨(0: 다중 샘플링 안함)

※ 슈퍼 샘플링: 후면 버퍼와 깊이 버퍼 해상도를 화면 크기보다 4배 크게 만들어 렌더링
MSAA: 슈퍼 샘플링처럼 후면 버퍼와 깊이 버퍼를 화면보다 4배 더 크게 생성, 시간이 더 적게 걸림

※ CPU/GPU 동기화
CPU와 GPU가 병렬적으로 실행되기 위하여 동기화 필요
기본적으로 CPU는 리소스를 생성하고 GPU는 사용함
병렬처리에서 공유되는 리소스에 대한 동기화 처리가 필요할 수 있음
펜스 객체를 사용

※ 쉐이더 리소스
커밋 리소스: 가상 메모리 공간과 물릿거 메모리 공간을 모두 생성
위치 리소스: 힙의 어떤 영역(연속적인)에 대한 포인터
예약 리소스: 자체적인 GPU 가상 주소 공간을 가짐
다렉에서 힙을 할당할 때 물리적 메모리 공간을 생성함
UMA 어댑터는 시스템 메모리 세그먼트만 가짐

※ 리소스 -> 개념/리소스 베리어/리소스 상태
리소스: 가상 메모리에 생성, GPU가 리소스를 사용하려면 물리적 메모리로 매핑되어야 함
리소스 베리어: 리소스에 대한 상태 관리를 위한 객체

리소스 상태

COMMON:	0x00, CPU가 텍스쳐에 접근, COPY 큐의 상태 전이 전
VERTEX_AND_CONSTANT_BUFFER:	정점 버퍼 또는 상수 버퍼로 사용될 때
INDEX_BUFFER:	인덱스 버퍼로 사용될 때
RENDER_TARGET:	렌더 타겟으로 사용될 때, ClearRenderTargetView()
UNORDERED_ACCESS:	무순서 접근으로 사용될 때
DEPTH_WRITE:	깊이 버퍼에 쓸 때, ClearDepthStencilView()
DEPTE_READ:	깊이 버퍼를 읽을 때
NON_PIXEL_SHADER_RESOURCE:	픽셀 쉐이더 이외의 리소스로 사용될 때
PIXEL_SHADER_RESOURCE:	픽셀 쉐이더 리소스로 사용될 때
STREAM_OUT:	스트림 출력으로 사용될 때
INDIRECT_ARGUMENT:	ExecuteIndirect()
COPY_DEST:	복사 연산의 목표로 사용될 때
COPY_SOURCE:	복사 연산의 소스로 사용될 때
RESOLVE_DEST:	리졸브 연산의 목표로 사용될 때
RESOLCE_SOURCE:	리졸브 연산의 소스로 사용될 때
GENERIC_READ:	업로드 힙의 초기 상태
PRESENT:	0x00, D3D12_RESOURCE_STATE_COMMON
PREDICATION:	리소스가 예측을 위하여 사용될 때

※ 서술자
리소스는 쉐이더 파이프라인에 연결(바인딩: 리소스 객체를 쉐이더에 연결)되어야 사용 가능
리소스는 파이프라인에 직접 연결되지 않고 서술자를 통해 연결
서술자는 하나의 리소스에 대한 정보를 포함하고 있는 객체(자료 구조)
GPU에게 리소스를 완전하게 설명하는 데이터
파이프라인 리소스에 대한 접근을 하기 위하여 서술자를 참조
서술자는 서술자 테이블을 구성하도록 그룹화됨
각 서술자 테이블은 하나의 유형의 리소스에 대한 정보를 저장
서술자 테이블은 서술자 힙에 저장
서술자 크기는 GPU 하드웨어에 따라 다름

※ 루트 시그너쳐
쉐이더가 사용하는 리소스 바인딩에 대한 규약
쉐이더가 요구하는 리소스와 명령 리스트를 연결하기 위한 정보
쉐이더 프로그램이 요구하는 각 바인딩에 대하여 하나이 루트 시그니쳐 필요
(그래픽스 루트 시그니쳐, 계산 루트 시그니쳐)

※ ID3D12DescriptorHeap
리소스를 서술하는 서술자들을 저장하는 연속적인 메모리 영역
서술자 힙은 유형별로 구분되는 연속적인 메모리 영역으로 구성됨
GetCPUDescriptorHandleForHeapStart(): 힙의 시작을 나타내는 CPU 서술자 핸들을 반환

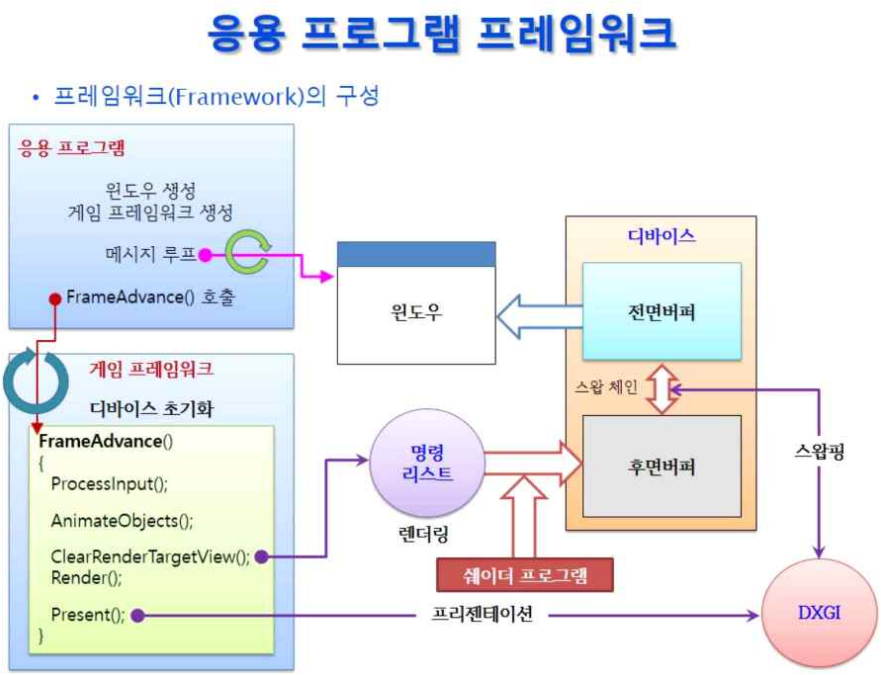
ID3D12Device::CreateRenderTargetView
ID3D12Device::CreateDepthStencilView
ID3D12Device::CreateCommittedResource -> 디폴트힙, 업로드힙, 리드백힙

명령 목록은 ID3D12CommandQueue 인터페이스를 사용하여 실행

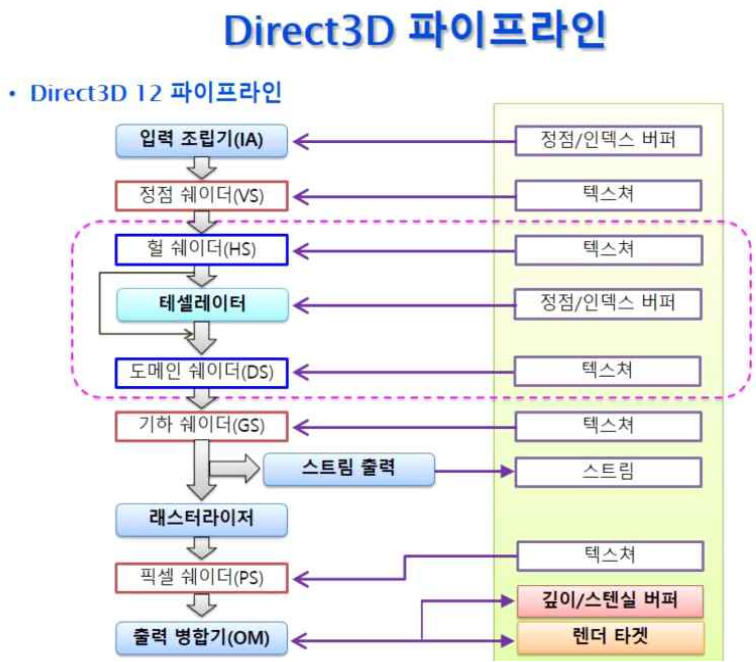
※ 리소스 뷰
리소스는 쉐이더 파이프라인에 연결(바인딩: 리소스 객체를 쉐이더에 연결)되어야 사용 가능
리소스는 파이프라인에 직접 연결되지 않고 서술자를 통해 연결
▶ 종류
상수 버퍼 뷰: 상수 버퍼이므로, GPU에서 값 못바꿈
무순서화 접근 뷰: GPU에서 이 버퍼의 내용을 바꿀 수 있다는 의미 내포
쉐이더 리소스 뷰: 쉐이더가 텍스처, 버퍼, 기타 데이터 같은 구조화된 리소스에 액세스 하는 방법
샘플러: 쉐이더 내에서 텍스쳐가 샘플링 되는 방식 정의
렌더 타겟 뷰: 렌더링 작업의 대상 역할을 하는 버퍼나 텍스처를 지정하는데 사용
깊이 스텐실 뷰: 렌더링 중 깊이 스텐실 정보에 사용되는 버퍼 또는 텍스처 지정
정점 버퍼 뷰: 형상 렌더링에 사용되는 꼭지점 데이터가 포함된 꼭지점 버퍼의 보기를 지정
인덱스 버퍼 뷰: 정점 버퍼의 정점을 참조하는 인덱스를 포함하는 인덱스 버퍼의 뷰 지정
스트림 출력 뷰: GPU가 나중에 사용하기 위해 처리된 정점 데이터를 버퍼에 쓸 수 있게 하는 기능

※ 상수 버퍼를 연결하는 방법
-> 상수버퍼는 최대 16개(packoffset(c16);) 까지 사용 가능 -> 상수 버퍼 인덱스는 CreateRootSignature()에서 세팅된 루프 파라미터 인덱스 값을 기반으로, Update() 함수에서 초기화 후 사용 -> 세팅된 인덱스 값은 b0, b1 선언

※ 응용 프로그램 프레임워크



※ 렌더링 파이프라인



고정 프로그램 단계
다렉에서 모든 처리가 진행되며 응용 프로그램에서 변경할 수 없는 단계
입력 조립, 테셀레이터, 스트림 출력, 래스터라이저, 출력 병합

프로그램 가능 단계
응용 프로그램에서 쉐이더 프로그램을 통하여 제공해야 하는 단계
정점 쉐이더, 헐 쉐이더, 도메인 쉐이더, 기하 쉐이더, 픽셀 쉐이더

※ 입력-조립 단계
고정 함수 단계
응용 프로그램에서 제공한 정점 버퍼의 정점 데이터를 다른 파이프라인 단계에서 사용할 프리미티브로 조립
시스템 생성 값을 추가
시스템 생성 값은 시맨틱이라고 하는 문자열의 값

※ 래스터라이저 단계
벡터 정보를 래스터 이미지로 변환
① 원근 투영 나누기
② 카메라 절두체를 벗어나는 점들을 클리핑
③ 프리미티브를 2차원 뷰포트로 매핑
④ 프리미티브의 모든 픽셀들에 대해 픽셀-쉐이더 호출

※ 출력-병합 단계
최종적으로 픽셀의 색상을 생성하여 렌더 타겟으로 출력
▶ 깊이-스텐실 검사
픽셀이 그려져야 하는지를 결정하기 위해 깊이 값과 스텐실 값 데이터를 사용
깊이 검사
하나의 깊이/스텐실 버퍼만 활성화됨
▶ 블렌딩
픽셀 값들을 결합하여 하나의 최종 픽셀 색상을 생성하는 과정
펜더 타겟의 픽셀 색상과 출력 색상을 결합

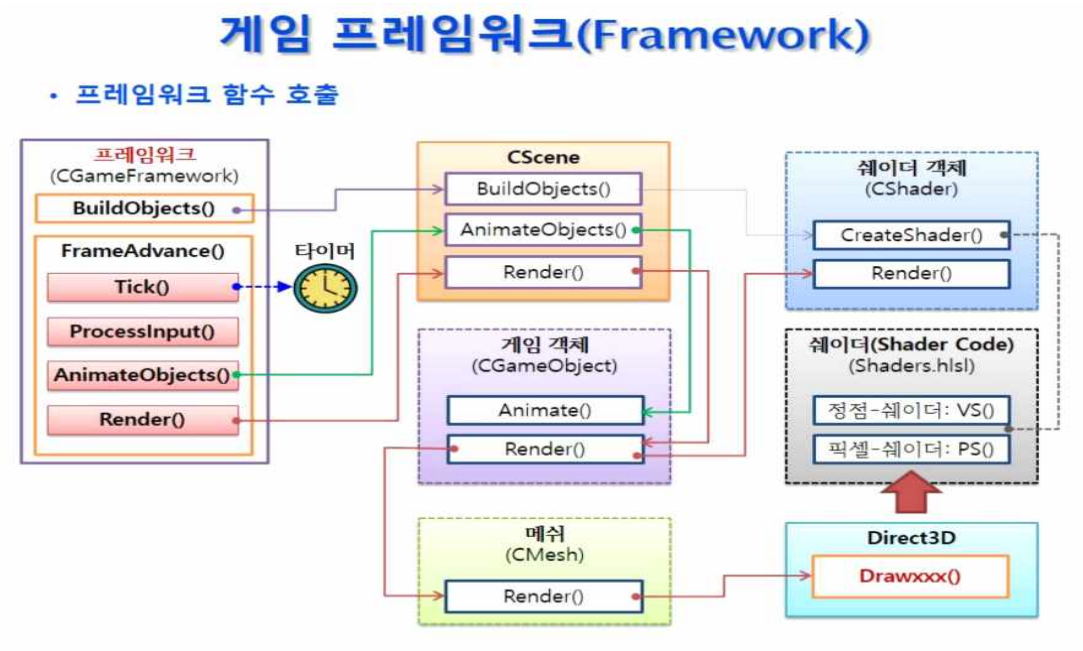
※ 쉐이더 단계
▶ 정점-쉐이더 단계
각 정점에 대한 연산 수행(변환, 스키닝, 모핑, 조명)
하나의 정점에 대하여 한 번 호출되며 하나의 출력 정점을 생성
파이프라인 단계에서 항상 수행되어야 함
최소 하나의 입력과 하나의 출력을 가져야 함
▶ 픽셀-쉐이더 단계
출력 색상을 결정
각 픽셀에 대하여 픽셀-쉐이더를 한 번 씩 호출
픽셀-쉐이더 입력은 시스템-값 시맨틱으로 선언
스텐실 값 출력 불가능

VS_OUTPUT: 버텍스 쉐이더의 출력에 대한 구조체 정의, 개별 정점을 처리하고 변환된 정점 데이터 출력
VS_INPUT: 버텍스 쉐이더 입력에 대한 구조체 정의
SV_TARGET: 픽셀 쉐이더에서 쉐이더 출력 색상을 나타내는데 사용
cbuffer: 상수 데이터를 CPU에서 GPU로 전달하는 데 사용되는 상수버퍼 정의
register(b0): 상수 버퍼에 할당된 레지스터 공간과 인덱스를 나타냄, 상수 버퍼 레지스터 세트를 나타내는 레지스터 공간 b0에 할당되도록 지정
packoffset(c0): 상수 버퍼 내에서 패킹 오프셋을 지정하는 것

※ HLSL
planenormalize(): HLSL(고수준 쉐이더 언어)에서 사용되는 내장 함수, 평면 벡터를 정규화

※ 시멘틱
쉐이더 입력 또는 출력 매개변수에 부착되는 문자열
매개변수의 사용 의도를 나타내는 정보를 컴파일러에게 전달하기 위해 사용
쉐이더 컴파일러는 이 문자열을 쉐이더의 입력과 출력을 연결하기 위해 사용
쉐이더 단계들 사이에서 사용되는 모든 매개변수에 시맨틱 사용해야 함
쉐이더 변수 이름 다음에 콜론(:) 과 시맨틱 문자열 추가
쉐이더 프로그램의 전역변수와 매개변수에만 시맨틱 사용해야 함
쉐이더 컴파일러는 다른 변수의 시맨틱은 무시함
일반적으로 파이프라인 단계에서 전달되는 데이터는 고정된 의미를 갖지 않음
임의의 시맨틱이 허용됨
정점/픽셀 쉐이더 시맨틱: 일반적으로 정점/픽셀 쉐이더의 입력 및 출력 매개변수에 사용하는 시맨틱
시스템-값 시맨틱: 특별한 의미를 갖는 정해진 시맨틱, SV_로 시작
CreateInputLayout(): 그래픽 파이프라인의 입력 어셈블러 단계에 대한 입력 레이아웃을 정의하는 데 사용

※ 시스템-값 시멘틱
특별한 의미를 갖는 정해진 시맨틱, SV_로 시작
▶ 정점 쉐이더
SV_Position 시맨틱을 가진 매개변수를 출력해야 함
정점 쉐이더의 입력이 SV_Position 시맨틱을 가질 수 있음
래스터라이저 단계는 SV_Potision 시맨틱 배개변수를 사용
▶ 픽셀 쉐이더
SV_Depth와 SV_Target 시맨틱을 가진 매개변수만을 출력할 수 있음
▶ SV_VertexID, SV_InstanceID, SV_IsFrontFace 시맨틱
파이프라인에서 활성화된 첫번째 쉐이더의 입력으로 전달될 수 있음
다음 쉐이더 단계에 사용하려면 매개변수를 사용하여 직접 전달해야 함
▶ SV_PrimitiveID 시맨틱
정점 쉐이더 사용 불가능
헐 쉐이더, 도메인 쉐이더의 입력으로 사용 가능

※ 게임 프레임워크


※ 리소스의 갱신
▶ DEFAULT 힙
Map()을 호출할 수 없음
UnMap() 해줘야 함
▶ UPLOAD 힙
Write를 하기 위해 만든 힙
Map()을 하면 CPU 주소가 넘어옴
-> 업로드 힙에 복사
-> 업로드 힙 내용을 디폴트 힙으로 복사(CopyResource())
-> GPU가 디폴트 마음대로 접근 가능해짐
-> 즉, 업로드의 필요성은 내용을 디폴트에 옮기기 위함으로 존재
DEFAULT에 비해 빠르고 접근용이
모든 CommandList 함수는 GPU에 의해 실행
즉시 실행되는 것이 아닌 큐에 쌓임, 이전에 명령들이 실행된 후 실행
그러므로 CopyResource() 했다고 바로 릴리즈 하면 안됨
업로드 힙이 만들어진 이유는 '비디오 메모리에 내용은 자주 바뀔거야' 라고 가정하고 만듦
자주 바뀌는 놈들은 대부분 const 버퍼로 만들어짐(상수), 상수는 디폴트 힙이 필요가 없음
비디오 메모리에 CPU, GPU 모두 접근 가능, GPU가 더 개빠름
CPU가 접근하는 것을 줄여야 함, 비디오 메모리를 읽는 행위는 가급적 X
CPU는 없고 GPU는 있는것: 렌더 타겟(스크린 샷)
▶ READBACK 힙
GPU는 저장이 안되기 때문에 CPU가 렌더 타겟을 파일로 저장 가능하게 해줌

※ 입력-레이아웃
입력 조립기(IA) 단계에 연결되는 입력 버퍼의 구조를 표현
정점 데이터의 구조 또는 인스턴스 데이터의 구조를 표현
정점 쉐이더의 입력 시그니쳐와 일치해야 함
입력 버퍼(배열)의 모든 원소들은 같은 자료 구조(자료형)를 가져야 함
입력 데이터는 하나 또는 여러 개의 버퍼로 표현되고 조립될 수 있음