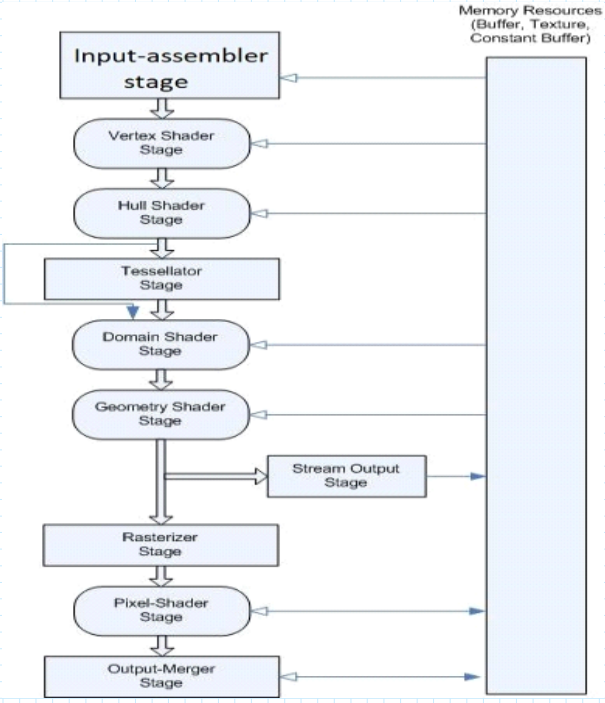
2023년 5월 17일 수요일 오후 8:27

렌더링 파이프라인

렌더링 파이프라인 또는 그래픽스 렌더링 파이프라인이라 부르는 과정은 3차원으로 만들어진 3D 모델 데이트들을 2차원(모니터)에 투영(바꾸다)하는 과정의 프로세를 자세하게 표현한 것이다.

크게 4가지 단계로 나뉜다.
1. 정점 데이터 처리 단계
2. 레스터라이저 단계
3. 픽셀(pixel) 단계 또는 프래그먼트(fragment) 단계
4. 출력 또는 병합 단계

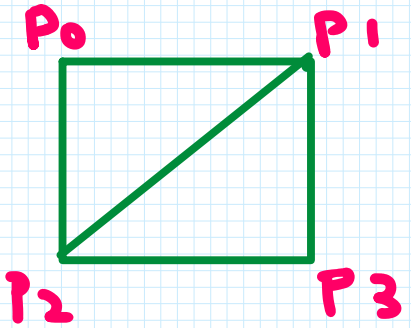


[Input Assembler]

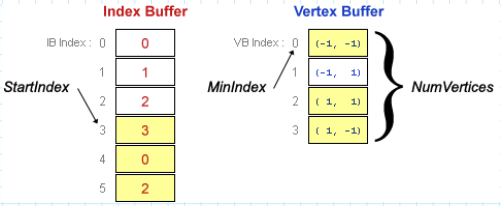
3차원 모델 하나를 3차원 세상에 나타내기 위해서는 가장 먼저 해줘야 할것이 무엇일까?
우선 모델정보를 GPU로 전달해야한다.
여기서 모델은 점의 집합 (폴리곤) 이다. 주로 삼각형을 가지고 3D 물체를 정의한다. 이때 정점들을 운반하는 자료구조를 vertex buffer 라고 합니다.

정점버퍼와 같이 등장하는 용어로 Index buffer라는것이 있다.

인덱스 버퍼는 쉽게 생각하면 정점들의 인덱스를 저장하고 있는 버퍼라고 할 수 있는데 사각형을 예를들어서 생각해보면 사각형을 그리기 위해서는 2*3 = 6 개의 정점이 필요한데 실제로 사각형을 구성하는데 4개의 정점만 있으면 된다.



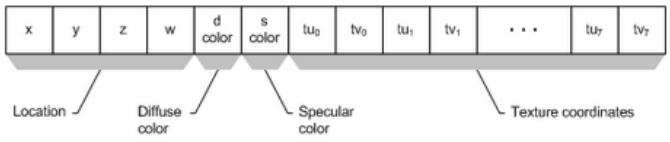
6개의 정점으로 표현한다면 2개의 정점이 중복되어 메모리 낭비가 발생할수 있다. 만약에 사각형 하나가 아니라 사각형 10,000개로 구성된 모델이라고 가정하면 20000개의 정점이 낭비된다. 이것은 메모리적으로 엄청난 손해이다. 또한 같은 정점작업도 추가적으로 20000번의 작업이 더 이루어져야 한다. 그럼으로 계산량도 늘어난다.

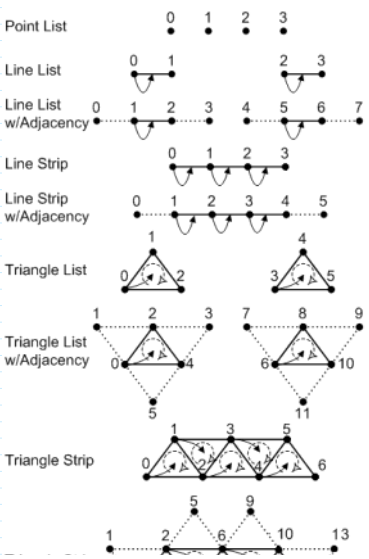


이러한 중복되는 정저믕ㄹ 해결하기 위한 방법이 Index Buffer이다.
정점은 vertex buffer[] = {p0, p1, p2, p3}; 4개만 보내주고
Indxe buffer[] = { 0, 1, 2, 1, 3, 2};
요런식으로 정점을 어떤 순서로 그려야 하는지 알려주는 목록을 제공하면 위에서 설명한 두가지 문제 (메모리낭비, 연산양 증가) 를 해결할수 있다.

그럼 추가적으로 인덱스버퍼를 사용하는데 그것도 결국 메모리를 사용하니까 똑같은게 아닌가 생각할수 있지만 정점은 위치데이터 말고도 색깔, 법선(Normal), UV 등등 여러가지 프로그래머 원하는 데이터를 추가하여 사용이 가능하기 때문에 단순하게 정수만 저장하는 인덱스버퍼가 훨씬 메모리적으로 효율적이다.

정점데이터가 위치데이터뿐만 아니라 다양한 데이터를 전달하는 예시

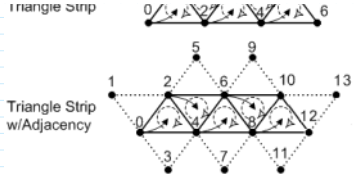




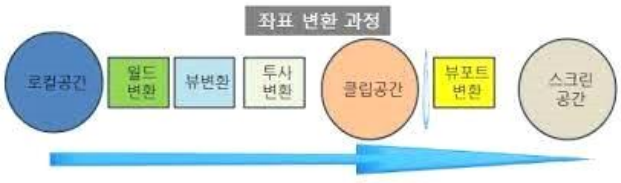
https://learn.microsoft.com/ko-kr/windows/win32/direct3d9/rendering-from-vertex-and-index-buffers

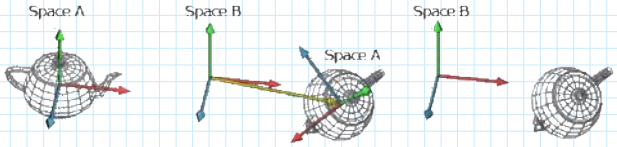
[정점 버퍼]
다시 정점버퍼로 돌아와서
정점버퍼는 그냥 정점들의 연속적인 메모리에 저장하는자료구조에 불과하기 때문에 실제로 gpu에서는 이러한 정점들을 이용하여 어떤 도형을 만들어야 하는지 정보가 필요하다. 해당 도형 정보를 Directx3D에서는 Primitive Topology라고 한다.
대표적으로 point list, line list, triangle strip, trinangle list등이 있습니다.
Input Assembler(입력 조립기)에서는 이러한 정점들 읽어서 삼각형과

대표적으로 point list, line list, triangle strip, trinangle list등이 있습니다.
Input Assembler(입력 조립기)에서는 이러한 정점들 읽어서 삼각형과 같은 도형으로 조립하는 단계의 일을 합니다. 그래서 Input Assembler(입력 조립기)라고 합니다.



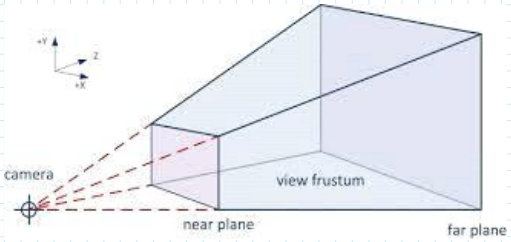
[Vertex Shader]
Input Assembler 에서 받은 정점 정보들의 정보로 도형은 생성이 되었지만 로컬좌표계에 있기때문에 해당 데이터를 화면에 그대로 출력 해버리면 여러가지의 도형 전부 같은 위치에 생성이 된다. 공간좌표계(World)로 변환할 필요가 있다. Local Space에서 World Space로 변환하고, 실제 플레이어가 바라보는 카메라 중심이되는 공간 View Space변환해준다. 그리고 마지막으로 Projection 을 거쳐서 최종적으로 정의된 Clip Space 공간으로 변환해준다.





[월드 변환]
Local space라고도 불리는 오브젝트 공간은 3차원 세상에서 표현될 각각의 개인공간에 정의된 영역이다. 그런데 이러한 오브젝트들을 하나가 아닌 여러 개를 한 화면에 모아놓은 공간을 World Space 라고 한다.

[카메라 변환]
월드 변환이 완료되여 모든 물체가 한공간(world sapce) 모아지면 이제 우리가 원하는 시점에서 물체를 관찰할수있어야 한다. 이때 관찰자로써 가상의 카메라가 필요하고, 이 카메라가 볼수 있는 영역의 공간 혹은 뷰 공간이라고 한다. 월드 공간의 모든 물체를 카메라 공간으로 변환하게되면 효율적으로 렌더링 작업을 수행할수 있다.
여기서 잠깐 카메라가 바라보는 세상에 대해 생각해보자. 가상의 카메라는 컴퓨터 성능의 한계 때문에 실제 세상의 카메라와는 다르게 시야가 제한될수 있다. Fov(시야각), aspect(종횡비)에 의해 결정되는데 이러한 시야의 가시영역을 뷰 볼륨(view volume)이라고 부르고, 이렇게 생성된 뷰 볼륨은 n(near), f(far)에의해 전달되어 View Frustrum(절두체)의 영역으로 다시 정의된다.



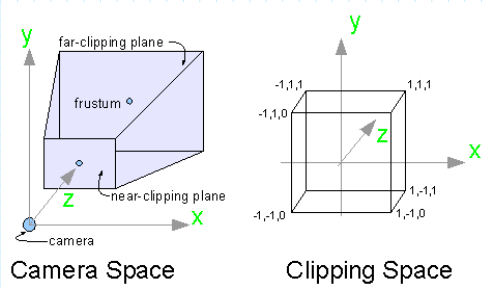
[view frustrum]
절두체 공간 밖에 있는 물체는 그리지 않는데, 우리가 살고있는 3차원 세상은 모든걸 보여준다. 이렇게 밖에 없는 이유는 계산상 효율성을 위해 어쩔수 없이 도입된 개념입니다. 만약 물체가 절두체의 경계에 걸치게되면 바깥족 부분은 잘려져 버리게 된다. 이를 클리핑이라고 한다. 이 클리핑은 카메라 변환에서 일어나지 않고 나중에 클립공간에서 수행되는데 이에 대해서는 뒤에서 설명하겠습니다.

[Projection(투영) 변환]
카메라 변환에서 월드의 모든 물체를 카메라 공간으로 이동시켰고, 이제 카메라 시점에서 세상을 바라볼수 있게 됐죠? 우리가 카메라를 통해 바라보는 가상의 공간은 현실세계 처럼 3차원 이지만 우리가 최종적으로 바라봐야 할 공간은 2차원이 되어야 합니다. 3차원 공간을 어떻게 2차원 공간에 표현할수 있을까요?

" 멀리 있는 물체일수록 작게", "멀리 있는 물체일수록 소실점에 가깝게"

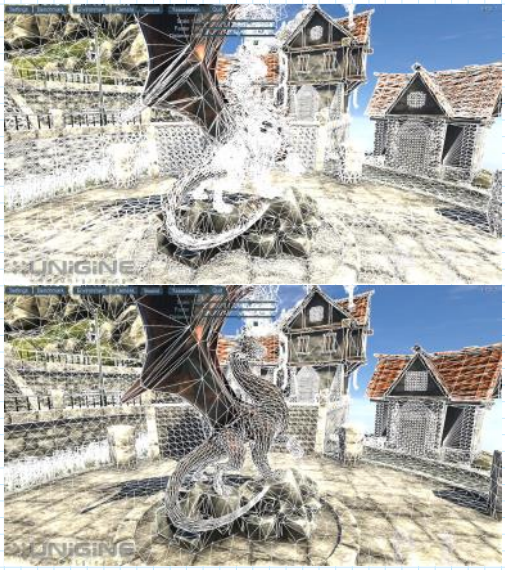
3차원 세상은 2차원 평면에 표현하는 방법은 투행이 이미 화가들에 의해서 개발되었기 때문에 우리도

이러한 방법을 따르면 된다.



투영변환은 이러한 원근 법을 구현하기 위해 카메라 공간에서 정의된 절두체를 축에 나란한 직육면체 볼륨으로 변경하여 카메라공간의 모든물체를 3차원 클립공간으로 변환하는것 을 의미한다.
여기서 투영변환이라는 이름과는 다르게 3차원 카메라 공간의 물체를 2차원 평면으로 투영하는 것이 아니라 또 다른 3차원 물체로 '변형' 됨에 주목 해야한다. 이러한 투영 변환을 거친 물체들을 관찰해보면 절두체 뒤쪽에 있던 영역의 폴리곤은 상대적으로 작아지는것을 볼수 있는데 우리가 원했던 원근법을 적용된 것이라고 볼수 있다. 다시 정리하자면 3차원 공간 내에서 원근법을 실행한것이다.

추가적으로 한개더 생각해 보면 원근법을 3차원 공간에서 실현하기 위해 직육면체 볼륨으로 물체들을 변환 시켰는데 여기서 얻는 이점이 있다. 좀더 간단한 공식으로 쉽게 클리핑 작업을 쉽게 할수 있다.

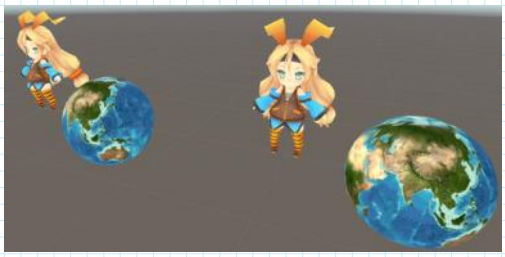


[Tesselation] 생략가능

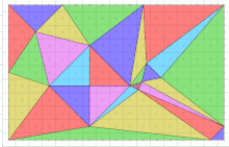
주어진 모델의 정점을 더 잘게 쪼개서 디테일하게 표현할수 있고,
또는 더 듬성듬성 정점을 합쳐서 저해상도 표현해서 최적화 성능을 올릴 수 있다. LOD(level of detail) 이라고 한다.
Tesselation을 사용하면 하나의 모델에 여러 해상도의 모델데이터를 가지고 있을 필요가 없다.

[Geometry]
기본 도형에서 정점을 추가하거나 삭제하여 모델을 변경할수 있는 셰이더 이다. Geometry Shader로 정점 정보를 조금 추가하여 표현할수 있는 모델이라면 그만큼의 정점정보를 빼고 저장할수 있으니 디스크 용량과 그래픽 메모리에 도움이 될 수 있으며, 테셀레이션등으로 추가된 정점들을 표현할 때도 사용된다.

게임 오브젝트 2개를 만든게 아니라 하나의 정점정보를 GPU 안에서 복사해서 만든것이다.



[Rasterizer]
정점 처리 단계를 지난 정점은 이제 렌더링 파이프라인의 다음 단계인 레스터라이저 단계로 들어갑니다. 우선 정점들은 삼각형으로 묶어지는데 이 시점부터는 하나의 도형이 독자적으로 처리가된다.
우선 화면에 그려질 2차원 삼각형의 세 정점이 결정되면 다음과 같은 일이 일어난다.



1. 이 삼각형이 포함하는 포함하는 모든 픽셀마다 픽셀셰이더가 실행된다.
2. 삼각형의 세 정점에 할당되었던 여러데이터 (pos, uv, normal)등의 데이터는 보간(interpolation)이 되어 삼각형 내부의 각각 픽셀셰이더로 넘어온다.

Direct3D에서는 이러한 과정들을 통틀어서 레스터라이제이션이라고 부른다. 이 레스터라이저는 고정 파이프라인 단계로 프로그래밍이 불가능하여 하드웨어 자체 알고리즘 통해 동작하며

- 클리핑(Cliping)
- 원근 나눗셈(perpective division)
- 뒷면제거 (backface culling)
- 뷰보트 변환(viewport transform)
- 스캔변환 (ndc scan transform)

[클리핑]
클리핑은 투영변환 이후의 클립공간 볼륨 바깥에 놓인 폴리곤들을 잘라내는 작업을 말합니다. 이전 부터 언급되었던 이 작업이 바로 이 레스터라이저 단계에서 일어납니다.

[원근 나눗셈]
현재 단계에서 투영변환을 통해 원근법이 적용된 3차원 물체들을 직육면체 클리핑 공간에서 정의되어 있습니다. 우리가 최종적으로 필요한건 2차원 공간인데 그렇다면 어떻게 3차원공간을 2차원 공간으로 변환시킬 수 있을까요?(수학적)
단순히 생각하면 3차원에서 2차원으로 차원을 줄이면 됩니다. 바로 Z좌표로 모든 성분을 나눠버리는거죠. 투영변환을 마친 정점데이터는 (x, y, z, w)에서 w성분에 z값이 저장된다. 원근 나눗셈이 적용 된 이후에는 x,y,z,w -> x,y,z의 좌표계로 변환되는데 이를 NDC(normailize device coordinate) 공간이라고 부릅니다. 여기서 정규화라는 이름이 붙는 이 좌표의 xy 범위는 [-1 ~ 1] z의 범위는 [0~1]이기 때문입니다.

[뒷면제거]
다음으로 레스터라이저에서 하는 기능으로 뒷면 제거가 있습니다. 카메라가 바라보고 있는 방향에 물체에 가려진 면적은 굳이 연산을 할 필요가 없다. 외적(Cross product) 삼각형의 바라보고있는 면의 방향을 구하여 뒷면일 경우에 연산에서 제외 시킨다.

[뷰포트 변환]
컴퓨터 화면상의 윈도우 스크린 공간을 갖는데 이 스크린 공간 내에 2차원 이미지가 그려질 뷰포트가 정의되는데 NDC공간의 물체들을 스크린 공간으로 이전시키는변환을 뷰포트 변환이라고 합니다.

[스캔 변환]
이전의 변환들은 자세한 사항을 몰라도 프로그래밍하는데 문제가 없었지만 이 스캔 변환은 렌더링 프로그램에서 직접적인 영향을 미치기 때문에 꽤 중요하다. 삼각형 하나가 내부에 차지하는 모든픽셀(fragment)들을 생성하는 작업이다. 이때 정점데이터에 들어온 데이터들은 보간(선형보간)되어서 픽셀셰이더로 넘어간다.

[Pixel shader]
레스터화된 도형에 텍스쳐 매핑, 법선 매핑, 노말 매핑, 기법으로 텍스쳐를 입혀서 색을 표현한다. 원한다면 특정색깔로 표현이 가능하다.
조명처리나 이미지 처리를 할때 유용하게 사용된다.

[Output merger]
깊이 -스텐실 테스트와 블렌딩이 일어나서 최종적인 렌더타겟(frame buffer)물체를 그려줍니다.