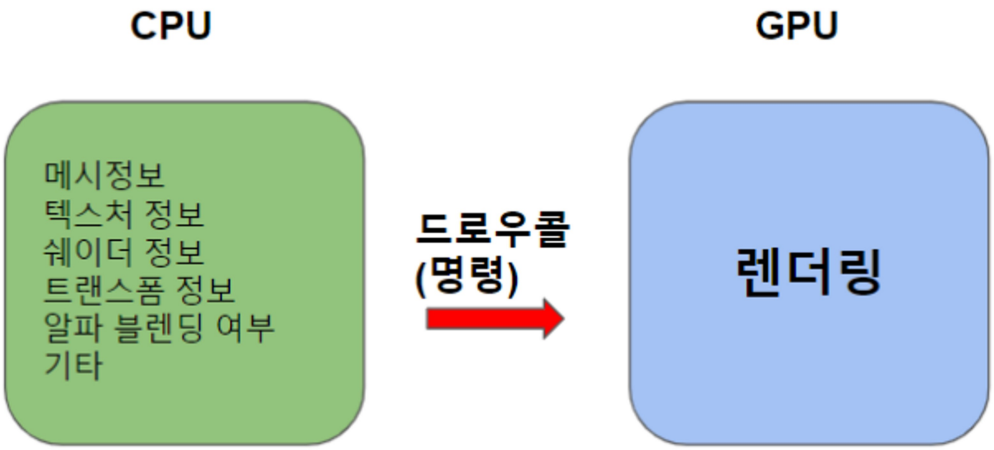
드로우 콜

2022년 5월 16일 월요일 오후 3:49

드로우 콜

CPU가 GPU에게 렌더링 작업을 수행하도록 명령함



드로우 콜은 렌더의 상태를 알려주는 명령과 CPU가 GPU에게 그리라는 명령(DP Call)까지이며,
이런 과정은 한번에 보내는게 아니라 순차적으로 보내게 된다.

데이터의 흐름

Stroage(HDD, SDD, SD) -> CPU메모리(RAM) -> GPU메모리(VRAM)
메시,텍스쳐등이 Storage에 있다. 그리고 CPU는 데이터들을 CPU메모리(RAM)으로 이동하고, GPU는 CPU 메모리의 데이터들을 GPU메모리에 복사한다.

커맨드 버퍼

CPU에서 GPU로 명령을 내릴 때 GPU가 다른 명령을 수행하고 있을 수 있기 때문에 커맨드 버퍼가 임시로 명령을 쌓아놓고 있는다.

CPU 성능에 의존되는 드로우 콜

GPU가 그리기 위한 신호들을 모두 CPU가 GPU에 맞게 변형하면 보내기 때문에, 텍스쳐의 크기를 줄이거나 메시를 줄인다고 부하가 없는게 아니다 (드로우 콜의 횟수를 줄여야함)

배치, SetPass

배치는 각각의 드로우 콜 사이에서 그래픽스 드라이버 유효성 체크를 진행 할 때 GPU에 의해 접근되어지는 리소스들을 변경하는 일련의 작업들을 총칭함
SetPass는 쉐이더로 인한 렌더링 패스 횟수를 의미하고 쉐이더의 변경시 SetPass 카운트는 증가한다.

드로우 콜의 발생조건

오브젝트 하나를 그릴 때 메시 1개, 머테리얼 1개로 이루어져 있으면 배치는 1이다. 하지만 하나의 오브젝트가 여러 파츠로 나누어져 있고 메시가 여러개이면 하나의 머테리얼을 공유했다고 해도 파츠 수 만큼 드로우 콜이 발생한다.

파츠 수 X 오브젝트 수 = 드로우 콜 수

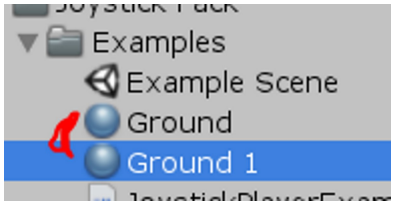
머테리얼도 마찬가지.

드로우 콜 줄이기 방법

드로우 콜에 대해서 알았다면, 이제 이것을 줄이는 방법을 알아야 한다. 가장 효율적인 방법으로 배칭이 있다. 효율적인 방법이라고 한 것은 여러의 배치를 하나로 묶어서 하나의 배치로 만드는 것이 바로 배칭인 것이다. 한 가지 예를 들어보면 오브젝트가 3개 있다면 원래는 3개의 드로우 콜 즉, 3개의 배치가 필요하지만 조건에 따라서 1개까지 줄이는 것이다. 메시가 서로 다른 오브젝트이지만 같은 머테리얼을 사용하면 하나의 배치로 만들 수 있다.

배칭을 위해서는 다른메시를 이용하더라도 머티리얼을 공유해서 사용해야 한다. 파츠가 여러 개인 경우 메시는 각각 다르지만 하나의 머티리얼을 통해 색상을 입히는 것이다. 이렇게 하기 위해서는 여러 장의 텍스처를 하나를 묶어 텍스처 아틀라스 기법으로 하나의 텍스처로 여러 메시들이 사용할 수 있게 하는 것이다.

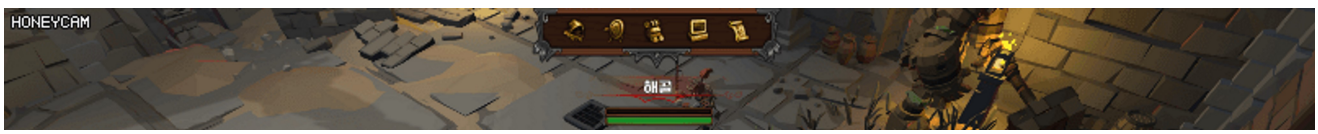
같은 머테리얼이라는 의미는 동일한 인스턴스를 말하며, 같은 텍스처를 사용하는 머테리얼이지만 그것이 2개라면 같은 인스턴스가 아닌 것이다.



다른 머테리얼

이러한 이유로 코드상에서 머티리얼을 접근할때 유의할 점이있다. 머테리얼 색상을 바꾸는 코드는 GetComponent<Renderer>().material.color = Color.red;을 사용하면 색상을 바꿀 수 있다. 하지만 이 코드를 사용할 때마다 계속해서 새로운 머테리얼을 복사 생성하게 된다. 그러면 다른 머테리얼의 인스턴스를 생성하는 것이다. 하지만 같은 인스턴스를 사용하는 방법도 있다. 머테리얼의 속성을 바꾸는 Renderer.sharedMaterial을 사용하면 된다. 물론 상황에 따라 사용하겠지만 한 가지 예시로,

아래 그림처럼 스켈레톤 몬스터는 모두 같은 머티리얼을 사용하고 있다. 그러면 현재는 한 개의 인스턴스가 있는 것이다. 그런데 플레이어가 스켈레톤을 공격하면 빨간색으로 피격 효과를 받게 되는데 같은 인스턴스를 공유하고 있기 때문에 피격을 당하지 않은 몬스터까지도 피격 효과(빨갛게) 변할 것이다.



이것은 분명 개발 기획의도는 아닐 것이다. 그렇다면 어쩔 수 없이 몬스터마다 각각 다른 인스턴스를 생성해서 적용해야 하는 걸까?

2019/10/03 - [유니티/레퍼런스] - 유니티)MaterialPropertyBlock

유니티)MaterialPropertyBlock

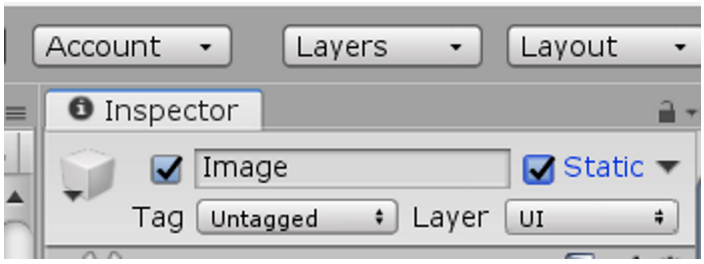
sharedMaterial 머테리얼의 속성을 변경할때 SharedMaterial으로 색상을 바꾼다던지 쉐이더코드의 프로퍼티값들을 변경한다. 그런데 이 런타임중에 변경된 머테리얼의 값들은 종료해도 바뀐 값 그대로 유지된다...

funfunhanblog.tistory.com

MaterialPropertyBlock를 이용하면 객체의 변화를 주는 동안만 배칭이 되지 않고 작업이 끝나면 다시 배칭을 하는 방법이다.

8. 스태틱배칭

스태틱 배칭은 말 그대로 정적인 오브젝트 즉 움직이지 않는 오브젝트를 위한 기법이다. 주로 배경 오브젝트들이 해당되는데 이 기법을 이용하려면 Static 플래그를 켜야 한다. 이 플래그를 켜야 유니티가 이 오브젝트는 움직이지 않는 스태틱 오브젝트라는 것을 알 수 있기 때문이다.



그런 후 다른 작업 없이 게임을 시작하면 자동으로 유니티는 스태틱 플래그가 켜진 오브젝트를 배칭 처리 작업을 진행한다. 오브젝트 버텍스가 많을수록 렌더링 연산 비용이 크다. 당연히 버텍스 수가 많으면 런타임 과정 중에 부담이 크다. 하지만 스태틱 배칭 된

오브젝트는 런타임에 연산이 이루어지지 않는 게 장점이 되는 것이다.
드로우콜수를 줄이기 위해 오브젝트들을 합쳐 내부적으로 하나의 메시로 만들어준다.
만일 3개의 오브젝트가 1개의 메시만 사용하더라도 3개의 메시를 합친 만큼 추가 메모리가 필요하게 되는 것이다. (내 생각에는 SpriteAtlas처럼 생각하면 될듯하다)
물론 이 작업은 유니티가 이러한 정보들을 저장해 야하기 때문에 메모리가 조금 더 필요하게 된다. 렌더링(드로우 콜)이냐 메모리냐 이 논제는 최적화에서 자주 결정해야 할 사항이다. 스태틱 배칭을 간단하게 정의하자면
대상 :움직이지 않는 오브젝트
장점 : 드로우콜 감소
단점 : 메모리 증가

9. 다이내믹 배칭

동적으로 움직이는 오브젝트들끼리 배칭 처리를 하는 기능이다. 동일한 머티리얼을 사용하며 자동으로 배칭이 이루어진다. 배칭 처리는 런타임상에 이루어지며 Static플래그가 체크되어 있지 않는 오브젝트들의 버텍스들을 모아서 합쳐주는 과정을 거친다. 이러한 버텍스들을 모아서 다이내믹 배칭에 쓰이는 버텍스 버퍼와 인덱스 버퍼에 담는다. GPU는 이를 가져가서 렌더링 하는 것이다. 매프 레임 데이터 구축과 갱신이 발생하기 때문에 오버헤드가 발생하게 된다. 다이내믹은 자동으로 이루어지지만 여러 가지 조건들이 있다.
1. 버텍스 900개 이하로 된 메시로만 적용된다.
2. 트랜스폼을 미러링 한 오브젝트들은 배칭 되지 않는다.(예를 들어 Scale값이 1인 오브젝트를 -1로 적용했을 때)
3. 다른 머티리얼 인스턴스를 사용하면 같이 배칭 되지 않는다.
4. 라이트맵이 있는 오브젝트는 추가 렌더러 파라미터를 가지는데, 일반적으로 동적으로 라이트매핑된 오브젝트가 완전히 동일한 라이트맵의 지점에 있어야 배칭 된다.
5. 멀티 패스 셰이더를 쓰면 배칭이 되지 않는다.
스태틱 배칭은 버텍스 쉐이더에서 월드 스페이스로의 변환이 이루어진다. 즉 GPU에서 연산이 된다. 하지만 다이내믹 배칭은 실시간으로 오브젝트의 위치가 변환되기 때문에 CPU에서 연산이 이루어진다.

10. 2D 스프라이트 배칭

2D도 배칭 작업이 가능하다. 텍스처 아틀라스 기법처럼 여러 이미지들을 한 이미지에 모아서 사용하는 방법과 SpriteAtlas를 사용하여 배칭 처리한다.
2019/01/09 - [유니티/최적화] - 유니티) 아틀라스 Sprite Atlas이용해 드로우콜을 줄여보자

유니티) 아틀라스 Sprite Atlas이용해 드로우콜을 줄여보자

스프라이트 아틀라스 유니티에서 아틀라스는 텍스처를 한곳에 모은 한장의 큰 텍스처라고 할 수있다. 텍스처 하나씩 따로따로 사용하고 관리하는것은 효율적이지 못하다. 아틀라스를 사용하면 드로우콜을 줄일 수..
funfunhanblog.tistory.com

11. GPU 인스턴 싱

적은 수의 드로우 콜을 사용하여 동일한 메시의 여러 복제본을 한 번에 그리거나 렌더링 한다. 특히 씬에서 반복적으로 나타나는 건물, 나무, 풀 등의 오브젝트를 보여줄 때 좋다. 또 같은 메시를 사용하더라도 간단한 변화 컬러, 스케일 등의 변화를 줄 수 있다.

별도의 메시를 메시를 생성하지 않기에(트랜스폼 정보를 별도의 버퍼에 저장) 다이내믹 배칭과 스태틱 배칭보다 오버헤드가 적다.

출처: <https://funfunhanblog.tistory.com/302?category=818491>