연구일지

2015182017 게임공학과 신재욱

2018-06-27

Scene들은 기본적으로 RootSignature를 공통적으로 가져야 한다.

플레이어가 실제로 움직이는 Scene에서는 Object 배열이 필요하다.

- Character

- Collide Object

- None-Collide Object

- Light

- UI

- Level

- Bullet

- SkillProjectile(플레이어가 발사하면 날아가는 투사체)

- SkileObject(SkillProjectile이 충돌하면 경우에 따라 생성되는 판정 범위)

- Billboard

- Particle

- Effect

- 그리고 TestRendering(충돌박스 표시, Gizmo 표시 등)

충돌체크와 렌더링을 위해서 Object 배열을 구분해줘야 한다. 또한, PSO를 가지고 있을 클래스가 필요한데 (기존의 Shader 클래스) PSO에서 Blending 등을 정하기 때문에 용도에 맞춰서 Object 배열별로 따로 Set 해줘야 하기 때문이다.

Character의 머리카락 표현에 있어서 plane을 쓰지 않고 mesh로 표현을 하면 투명한 재질을 표현하지 않아도 원하는 느낌을 줄 수 있을 것 같다. (그림 1-1 참고) Object에서는 철창 등의 표현을 Texture로 할 경우 PSO에서 투명한 것을 그릴 수 있게 설정해줘야 한다. (그림 1-2 참고) Billboard, Particle 그리고 Effect 또한 마찬가지로 Texture로 투명도 표현을 해줘야 하기 때문에 PSO에서 투명한 객체를 그릴 수 있도록 해야 한다. (그림 1-3 참고)



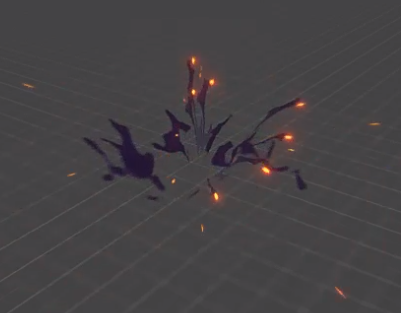
(그림 1-1, 스플래툰 2의 Character Rendering)



머리카락의 광택있는 면과 없는 면, 피부, 상의, 장갑, 하의 등의 재질 묘사가 전부 다르지만 그것은 지금 중요한 것이 아니고 투명한 표현을 하는 곳이 없다.



(그림 1-2, 디비전의 반투명 유리 재질의 정류장)



(그림 1-3, 이펙트에 사용된 Texture)

2018-06-28

Scene에서 Object 리스트를 관리한다면 GPU에서 사용할 변수들을 CPU에서 업로드해주기 위해 사용하는 Constant Buffer를 생성하고 갱신하고 사용이 끝나면 언맵까지 해줄 필요가 있다. 따라서 ID3D12Resource 포인터 변수를 Scene에서 가져야 한다. 또한, Object의 위치나 사용하는 재질 번호 등을 GPU로 전달해주기 위해 구조체를 선언해야 한다. (예를 들어 3D 게임 프로그래밍 과제의 CB\_GAMEOBJECT\_INFO)

Object를 GPU에서 그리기 위해 필요한 정보들은 위치 행렬, 재질 Index 등이 있다. UI 중에서 HP나 로딩 프로그래스바 같은 경우는 UI의 크기가 매 갱신마다 변화하고 따라서 기존 UI 크기에서 각 면으로 얼마만큼 변화하였는지 GPU가 알아야 한다. 따라서 매 Update 마다 GPU로 해당 정보들을 보내줘야 한다.

기존 방식에서는 Constant Buffer를 생성하고 갱신하는 부분들을 Scene 클래스에도 두고 Shader 클래스에도 두고 Object 클래스에도 두는 등 전체적으로 프로젝트가 관리가 안되고 있었기 때문에 이를 보완하여 Scene에서 모든 Object들을 총괄하는 것으로 하였다.

Scene 클래스 자체를 사용하지 않고 이를 상속받은 각 개별 Scene들을 사용할 예정이므로 Scene 클래스는 추상 클래스로 구현하고 인터페이스 클래스처럼 활용하기로 하였다.

2018-06-30

Title Scene과 Matching Scene, Result Scene은 따로 3D Object를 그릴 필요가 없다. 해당 Scene 들이 해야 할 일들은 평면에 이미지를 띄우고 현재 진행 정도 등을 표현할 프로그래스 바 정도를 보여주는 것이다.

그에 비해 EnterRoom Scene과 Play Scene은 3D Object를 그려야 한다. 공통적으로 위에서 말했던 Character, Collide Object, None-Collide Object, Default UI, Floating UI, Level, Bullet, SkillBullet, SkillObject, Effect, Particle 들을 가져야 한다. EnterRoom Scene과 Play Scene이 다른 것은 EnterRoom Scene에서 NPC와 상호작용하여 Skill 구성이나 기본무기 선택을 바꾸고 매칭하기 위해 ‘F’ 키를 상호작용 키로 처리한다는 점과, EnterRoom Scene에서 Skill이나 기본무기를 사용할 수 없도록 해당 키들을 처리하지 않게 한다는 점, Play Scene과 달리 Character를 8명까지 생성하지 않고 1명만 생성하여 플레이어만 보이게 한다는 점 등이 있다. 따라서 EnterRoom Scene과 Play Scene을 Ground Scene을 상속받아 만들고 위의 내용들만 다르게 작성하도록 하였다.

2018-07-01

RootSignature에서 인덱스를 주고 등록하는 것이 GPU에서 ‘상수 버퍼 몇 번째에 들어갈 것이다! 잘 알아둬라!’ 하는 건데 그럼 Object나 Texture들은 자기가 어디에 들어가는지 어떻게 알고 있을까? pd3dCommandList->SetGraphicsRootDescriptorTable()를 사용해서? 서술자테이블을 쓰지 않는 Vertex나 Light 등은 저 함수를 쓰지 않는데 어떻게 몇 번째에 들어갈 것이라고 알고 있는 것일까?

Mesh 클래스의 CreateVertexBuffer()와 CreateIndexBuffer() 쪽을 봐도 Vertex 정보를 어디에 올리는지 알 수가 없다. 애초에 RootParameter에서 Vertex 관련 부분을 정의하는 것 외에는 Vertex 인덱스를 참조하는 곳을 찾아볼 수가 없었다.

Light와 Material을 찾던 도중, GpuVirtualAddress에 저장한 내용을 Light 인덱스로 RootConstantBufferView로 Set하는 내용을 찾았고 이를 통해 ID3D12Resource의 GPUVirtualAddress를 받아와서 해당 주소를 연결하고, 내용 업로드는 해당 가상주소에 접근하여 진행한다는 것을 알 수 있었다.

DescriptorHeap을 쓰는 곳도 먼저 내부에서 DescriptorHeapDESC를 만들고 내용을 채워넣은 뒤, CreateDescriptorHeap()을 통해서 DescriptorHeap을 만들고 그 주소에 접근할 수 있게 m\_pd3dCbvSrvDescriptorHeap에 그 주소를 넣어주는 것 같다. 이후에 그 주소를 사용하여 해당 주소에 값들을 갱신해서 넣으면 이를 GPU의 Shader.hlsl에서 사용할 수 있는 것 같다.

기존의 Shader 클래스에서 해주던 일들을 이제 Scene에서 해야 하기 때문에 Shader.hlsl 코드를 읽어오는 것을 Scene 클래스로 옮겨서 작성하기로 하였다. 기존의 Shader 클래스에서는 파생된 Shader 클래스마다 하나의 PSO를 가지기 때문에 가상함수로 PSO마다 다른 Shader.hlsl 코드를 연결해줄 수 있었는데 이제는 Scene에서 모두 모아서 관리하기 때문에 이에 대해서 따로 대책을 세워야 한다.

따라서 PSO를 따로 클래스로 나누고 이를 사용하기로 하였다. 필요한 PSO는 일단 가장 기본적인 Textured, IlluminatedTextured 만 구현하도록 하였다.

2018-07-02

RootSignature 만들고 PipelineState 만들고 Object 메모리 할당하고 Texture 불러오고 DescritorHeap 만들고 CB 만들고 Map해주고 CBV 만들고 SRV 만들고 Material 만들고 Texture Set해주고 Mesh 만들고 Object 만들고 PipelineState Set해주고 DescriptorHeap Set해주고 Object 정보들 GPU에서 쓸 수 있게 갱신해주고 Texture를 RootSignature에 미리 정한 Texture 인덱스에 넣어주고… 🡪 Render()에서 SetGraphicsRootSignature()를 해줘서 해결했다…

2018-07-03

Map을 해줄 때 사용하는 변수(예를 들어 Camera 클래스의 경우 VS\_CB\_CAMERA\_INFO \*m\_pcbMappedCamera)는 각 객체마다 하나씩 있어야 한다. 그 주소에 내용을 적고 GPU에서 그 주소의 데이터를 미리 정해 둔 방식으로 읽어 HLSL에서 사용하는 것 같다. 아니면 Scene에서 Object INFO를 임시로 만들어서 거기에 Object들이 가지고 있는 주소를 넣어준 뒤 갱신하는 식으로 할 수도 있다.

렌더링을 하기 위해서 해줘야 하는 것들이 많다.

1. Object 단계: 우선 ConstantBuffer를 만들고 접근할 수 있게 Map을 하여 연결하여야 한다. 또한, 렌더링하기 전 계속해서 갱신을 해주어야 한다. CreateShaderVariables(), UpdateShaderVariables()에서 상수버퍼를 만들고 이를 접근할 수 있게 Map하여 관리한다. 이 때, Object 개수만큼 수를 만들어서 관리하여야 모든 Object를 갱신할 수 있다.

어떤 버퍼 서술자는 그 버퍼 주소에 있는 데이터를 어떤 식으로 해석할지를 적어 둔 것이라 알고 있다. CreateCbvAndSrvDescriptorHeaps()에서 서술자들 개수만큼 공간을 만들어야 하기 때문에 DescriptorHeapDesc에 적어 두고 CreateDescriptorHeap()로 서술자 힙을 생성한다. 여기서 서술자는 Object 개수만큼 만들어야 하나? 아니 근데 서술자면 ‘그 서술자가 가리키는 주소에 있는 데이터를 이렇게 해석해라’의 역할을 할텐데 굳이 서술자를 많이 만들어야 하는 이유가? 그 주소때문에? 이 부분은 나중에 따로 더 알아보자.

2. Mesh 단계: PSO에서 CCW로 후면 컬링을 하고 있을 때 보일 수 있도록 Mesh를 만들어야 한다. 이 후에 Picking 등을 할 때 고려해야 하는 사항들을 따로 작성해두자.

3. Camera 단계: 카메라 위치, 카메라가 보는 방향, View 행렬, Projection 행렬의 값을 정하고 HLSL에서 사용할 수 있게 갱신해줘야 한다. 저 모든 것들이 의도한 값으로 HLSL에서 읽히는지 검사해야 한다.

4. HLSL 단계: Texture는 잘 들어오고 있는지 다른 함수들은 잘 들어가는지 확인할 것.

2018-07-07

렌더링 체크리스트를 만들어 두자.

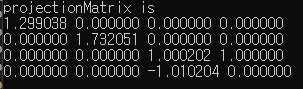
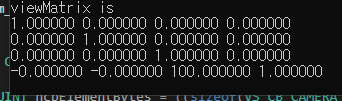
- View 행렬, Projection 행렬 잘 들어가는지 확인. (카메라 위치, 카메라 방향)

- Object 위치

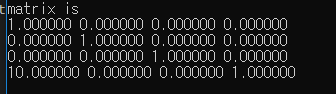
- Mesh CCW에서 Backface Culling 안 걸리는지 체크

- View 행렬, Projection 행렬, Object 위치, Mesh 점 위치, 전부 확인할 것.

애초에 HLSL에 버텍스를 안 넘겨주면 그려지는게 없으므로 테스트가 안 될 수 있다. 버텍스를 넘겨주는지 확인 해야 한다.

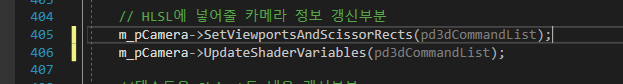


View 행렬과 Projection 행렬은 잘 들어가고 있다.

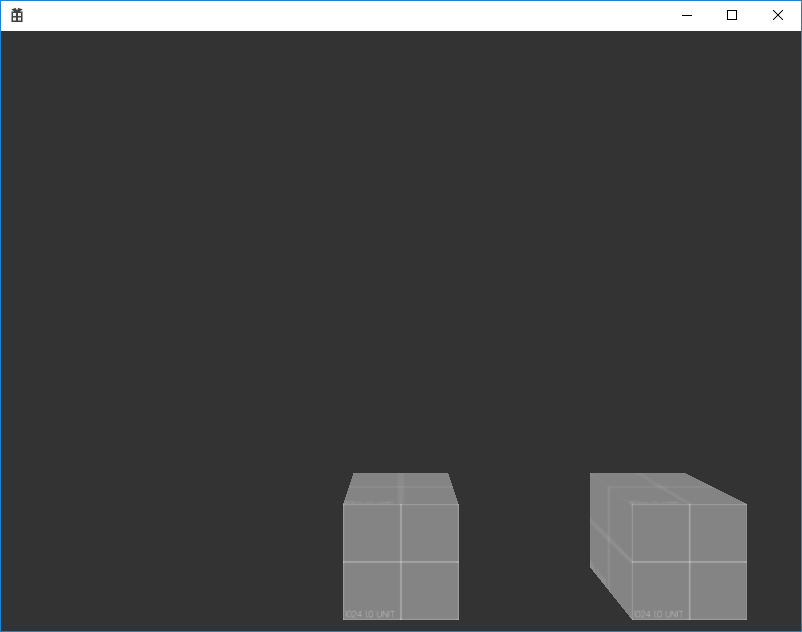


Object 위치 또한 문제없이 갱신되고 있다.

Rasterizer 생성부에서 Backface Culling 설정 값을 D3D12\_CULL\_MODE\_NONE으로 했음에도 불구하고 그려지지 않는 것을 보면 HLSL에 문제가 있는 것으로 보인다. 가장 기초적인 테스트용 삼각형을 그려본 뒤 위의 내용들이 HLSL에 올바르게 갱신되고 있는지 확인하도록 하자.



확인해본 결과 SetViewportsAndScissorRects()를 호출하지 않고 있었음이 드러났다.



Model Rendering 과 Texture Rendering이 해결됐다.