3D게임프로그래밍2

1차 과제 설명 문서

2018182025 이승준

1. 과제에 대한 목표 및 가정

* 프로젝트의 사용

본 과제는 기본적으로 주어진 Sample Project 여러 개를 서로 결합해야 한다. 나는 한 프로젝트를 선택하여 다른 프로젝트를 결합하기보다, 졸업작품에 사용하기 위해 방학 때부터 제작 중이던 내 클라이언트 코드를 바탕으로 주어진 Sample Project의 로직을 적용하여 과제를 진행해 보고자 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

*(ComPtr 사용) (Vertex Buffer 구조체 선언)*

코드의 구현 로직은 주어진 Lab Project 및 Sample Project 등과 크게 다르지 않지만, 객체의 선언에 ***ComPtr***이나 ***shared\_ptr*** 등 스마트 포인터를 사용하였고, 교수님께서 수업시간에 제안해주신 방법과는 다르게 Mesh의 생성에서 구조체를 사용하여 하나의 ***Vertex Buffer***만을 선언하여 제작하였다는 정도의 차이가 있다. 이렇게 하면 단 하나의 ***Vertex Buffer, Vertex Buffer View*** 변수만을 선언하면 되기 때문에 각각의 Mesh별 서로 다른 변수를 정의하고 초기화해줘야 할 필요가 줄어들어 코드의 간결함이 상승한다고 생각하였다. 그 외에 Microsoft에서 제공하는 *d3dx12.h* 헤더 파일을 사용하여 루트 시그니처나 파이프라인 상태 객체들의 초기화를 좀 더 용이하게 하였다.

* 기하 셰이더와 인스턴싱을 사용한 빌보드 구현

빌보드는 풀과 같이 수없이 많은 오브젝트를 렌더링하여야 한다. 따라서 셰이더로 객체 하나당 world matrix를 하나씩 넘겨가며 렌더링할 경우 프로그램에 걸리는 부하가 심하므로, 수업시간에 다룬 **기하 셰이더**를 활용하여 빌보드 객체의 위치를 나타내는 XMFLOAT3 변수 하나만을 넘겨 빌보드 객체를 렌더링해보고자 하였다. 또한 그렇다고 해도 수많은 빌보드 오브젝트를 반복해서 그리게 되면 비효율적이므로 인스턴싱을 사용하여 한번의 draw 명령으로 오브젝트를 그려 보고자 하였다. 이렇게 되면 기존과 같이 인스턴싱을 위해 input layout으로 world matrix들의 배열을 넘기는 것이 아니라, XMFLOAT3 변수 하나만의 배열을 셰이더로 넘기면 될 것이다.

* 지형 디테일 텍스처 매핑

커다란 base texture를 읽어들여 terrain 전체의 색을 결정하고, 그 사이사이를 detail texture를 상세히 렌더링 해보고자 하였다. 이러한 과정은 셰이더를 통해 단일 패스 렌더링으로 구현된다.

* 알파 블렌딩을 사용한 물 지형 구현

수업시간에 다룬 알파 블렌딩을 통해 물 속이 비쳐 보이도록 구현해보고자 하였다. 이러한 과정은 블렌드 상태 객체를 이용한 다중 패스 렌더링이 사용된다.

* 계층 구조 오브젝트에 대한 텍스처 매핑

1학기때와는 다르게 material 또한 텍스처로 읽어 와 렌더링하여야 한다. Sample Project의 로직을 내 코드로 잘 이식하되, C++ 함수를 바탕으로 계층 구조 오브젝트를 읽어오는 로직을 새로 구현해 보기로 하였다. 읽어 온 diffuse, emissive, specular, ambient map 등의 색은 Sample Project와 같이 어떤 texture를 파일로부터 읽어 왔는지 비트 마스킹을 통해 확인하고, 해당되는 색을 픽셀 셰이더에서 서로 더해주는 식으로 구현하고자 하였다.

* 자연스러운 적 움직임 구현

적은 플레이어를 계속해서 추적하도록 구현하고자 한다. 그런데 적 오브젝트가 단순히 position만을 바꿔가며 플레이어를 추적하면 굉장히 어색할 것이다. 적이 플레이어를 바라보도록 회전하고, 회전한 방향으로 다가오도록 구현하여 자연스러움을 더하고자 하였다.

* 적과의 전투에서 미사일 발사 시 헬기의 Child 오브젝트 활용

헬기 오브젝트에는 자식 오브젝트로 미사일이 이미 구현되어 있다. 적을 격추하기 위한 미사일을 발사할 때, 새로운 mesh를 구현하기보다 이러한 자식 오브젝트를 활용하여 미사일이 발사되는 모습을 나타내면 더 사실감이 있을 것 같아 이렇게 구현해보고자 하였다.

1. 조작법

* **W, A, S, D**
  + 플레이어가 전, 후, 좌, 우로 이동한다.
* **SPACE**
  + 플레이어가 위로 이동한다.
* **SHIFT**
  + 플레이어가 아래로 이동한다.
* **마우스 회전**
  + 플레이어의 시야가 회전한다.
* **마우스 좌 클릭**
  + 플레이어가 바라보는 방향으로 미사일을 발사한다.

1. 실행 화면
2. 구현 사항

* **기하 셰이더와 인스턴싱을 이용한 빌보드 구현**

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 빌보드를 렌더링하기 위한 Billboard Shader를 구현하였다. Billboard Shader에는 기하 셰이더가 포함되기 때문에 Pipeline State Desc에 기하 셰이더에 관한 내용을 포함해 주었다. 빌보드 셰이더는 깊이 검사를 하지 않고, 맨 마지막에 렌더링한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 후 input layout에 인스턴싱을 위한 WPOSITION을 선언해 주었다. 인스턴싱을 위해 사용되므로 D3D12\_INPUT\_CLASSIFICATION\_PER\_INSTANCE\_DATA로 선언해 주었다.

SIZE는 기하 셰이더로 넘어가 새로 생성하는 사각형의 크기를 결정해 준다.

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이렇게 넘어간 WPOSITION은 빌보드 셰이더의 정점 셰이더에서 새로운 정점 좌표 생성을 위한 output.position으로 변환된다.

Size는 그대로 다음 파이프라인으로 넘겨주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

기하 셰이더에서는 position과 size를 바탕으로 4개의 새로운 정점을 생성한다. 이러한 정점과 4개의 정점에 각각 새롭게 선언해준 uv 좌표를 바탕으로 우리가 원하는 텍스처를 렌더링 할 수 있게 되었다.

텍스트이(가) 표시된 사진

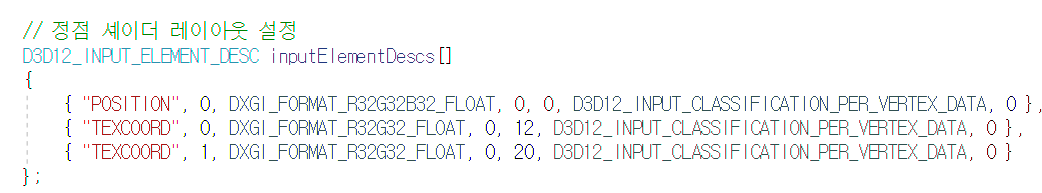
자동 생성된 설명

먼저 기하 셰이더와 인스턴싱 없이 빌보드를 구현한 결과이다. 4종류의 풀 빌보드를 일정 간격으로 랜덤하게 64x64개 배치하였는데 대략 70프레임 정도의 성능을 보이는 것을 알 수 있었다. 128x128개를 배치하게 되면 10프레임 후반대 정도로 떨어져 플레이가 상당히 불쾌하였고, 256x256개를 배치하게 되면 3프레임 미만으로 내려가 정상적인 플레이가 불가능하였다.

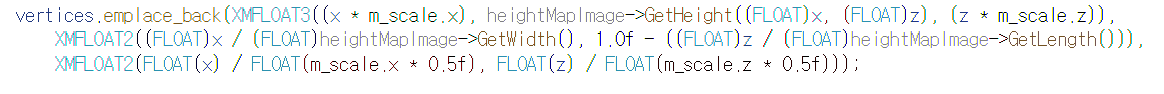


반면 기하 셰이더와 인스턴싱을 사용하여 빌보드를 배치한 결과 256x256개의 빌보드를 배치하였음에도 초당 70프레임 이상의 성능을 뽑아내는 것을 볼 수 있었다.

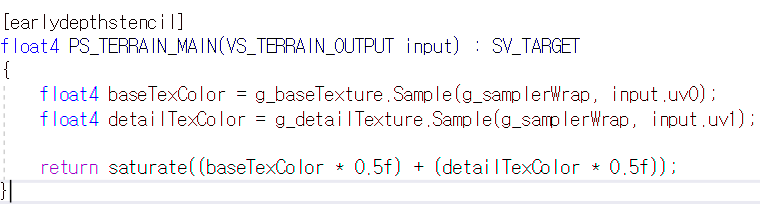
* **지형 디테일 텍스처 매핑**



디테일 텍스처를 구현하기 위한 Terrain Shader를 선언해 주었다. Terrain Shader에서는 두 개의 uv 좌표를 사용하기 때문에 input layout에 이를 반영해 주었다.



Base texture가 렌더링 될 첫 번째 uv는 전체 height map image의 길이를 나누어 반영해 주었고, detail texture가 될 두 번째 uv는 일정 간격으로 분할해 주며 반영해 주었다.



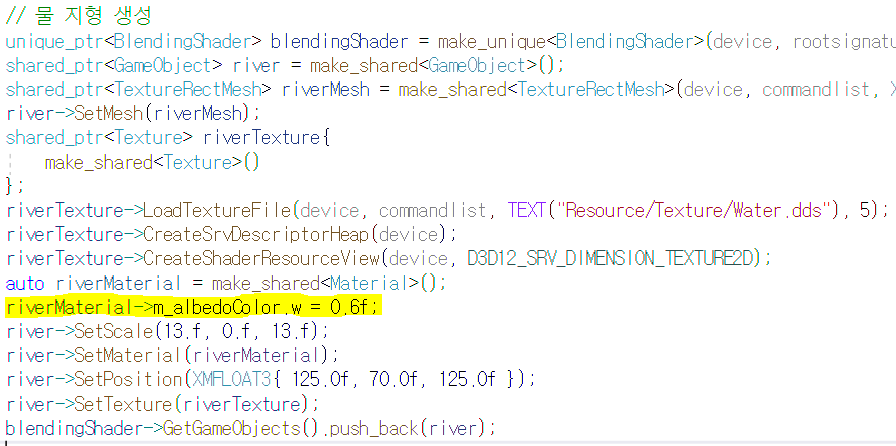
Terrain Shader의 Pixel Shader에서는 이 두 개의 uv 좌표를 사용해 base texture와 detail texture의 색을 구해준 후, 이를 절반씩 섞어준다.

* **알파 블렌딩을 사용한 물 지형 구현**

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 물 지형 렌더링을 위한 Blending Shader를 선언해 주었다. Blending Shader의 BLEND\_DESC를 적절하게 선언하여 준 후 깊이 검사를 끄고 맨 마지막에 렌더링해준다.



물 지형의 경우 Plane에 물 texture를 렌더링한 후 적절한 위치에 그려주는 식으로 구현하고자 했는데, 물 지형에 알파 값이 저장되어 있지 않아 material의 albedoColor의 알파값을 직접 선언해 주고, Blending Shader의 Pixel Shader에서 albedoColor의 알파값을 통해 투명 정도를 지정해 주는 방식으로 구현하였다.

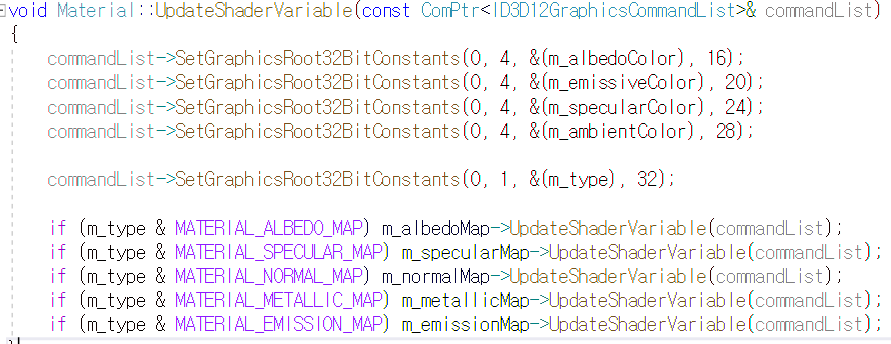
텍스트이(가) 표시된 사진

자동 생성된 설명

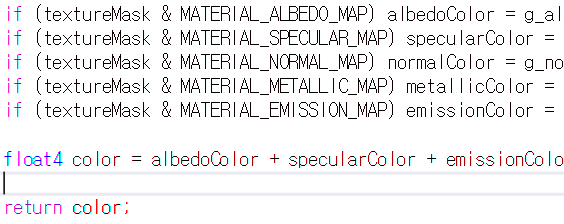
* **계층 구조 오브젝트에 대한 텍스처 매핑**



파일로부터 읽어들인 AlbedoMap, SpecularMap, NormalMap등의 정보를 material 클래스에 저장하였다. 또한 비트 마스킹을 통해 어떤 타입의 재질들이 셰이더로 넘어가는지를 지정해 주었다.

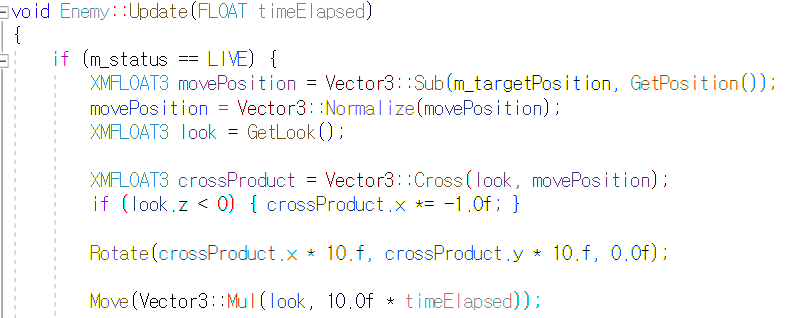


이렇게 저장된 재질 정보는 루트 상수 및 서술자 테이블을 통해 셰이더로 전달된다.



이렇게 셰이더로 받아온 색에 관한 정보들을 픽셀 셰이더에서 서로 더해주는 식으로 계층 구조 오브젝트의 색을 표현하였다.

* **자연스러운 적 움직임 구현**



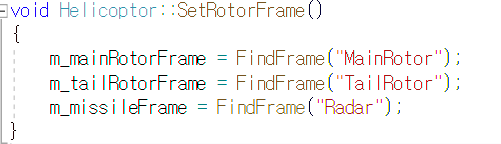
적이 플레이어에게 자연스럽게 다가오도록 하기 위해 먼저 플레이어의 위치와 적 오브젝트 객체의 위치를 빼고 이를 정규화하여 방향벡터를 구하였다. 이는 내가 향하고자 하는 새로운 *Look벡터*일 것이다. 이를 원래 *Look벡터*와 외적하면 새로운 *Up벡터*를 구할 수 있고, 이는 음수일 수도 있으므로(비행기가 뒤집어져 보인다.) 이를 보정해준다. 이는 객체의 새로운 *Pitch*값이 되고, 새로운 *Look벡터*를 객체의 새로운 *Yaw*로 지정해 주면 객체가 향하고자 하는 방향으로 알맞게 회전된다. 이제 앞으로 이동하게 되면 플레이어를 향하는 방향으로 객체가 이동한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 외에 헬리콥터 객체가 Terrain 안으로 파고들지 않도록 보정하는 부분을 추가해 주었다.

* **적과의 전투에서 미사일 발사 시 헬기의 Child 오브젝트 활용**



발사하기 위한 객체를 찾아 missile Frame으로 지정해 주었다. 다만 처음 계획과는 다르게 생각보다 발사하기 위해 적합한 객체가 존재하지 않아 Rader 프레임을 사용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

미사일은 발사 시에 미사일 고유의 world matrix를 가지고 이를 미사일이 파괴될 때까지 유지하여야 한다. 왜냐하면 미사일은 기본적으로 부모 world matrix를 가지는 계층 구조 오브젝트이기 때문에 발사된 이후로도 플레이어의 이동에 따라 상하좌우로 이동될 수 있기 때문이다. 따라서 플레이어의 이동에 영향을 받지 않게 하기 위해 world matrix를 미사일 발사 시 저장해 주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

미사일은 발사 시 플레이어가 바라보는 방향으로 쭉 날아가게 되며, 일정 시간이 지나거나 지형에 충돌할 시 사라진다. 물론 적과 충돌하게 되어도 사라진다.