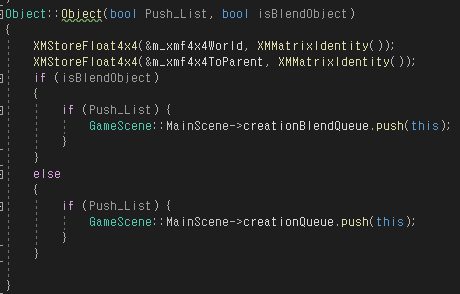
**2018180020 박재우(클라이언트) 14주차 기록**

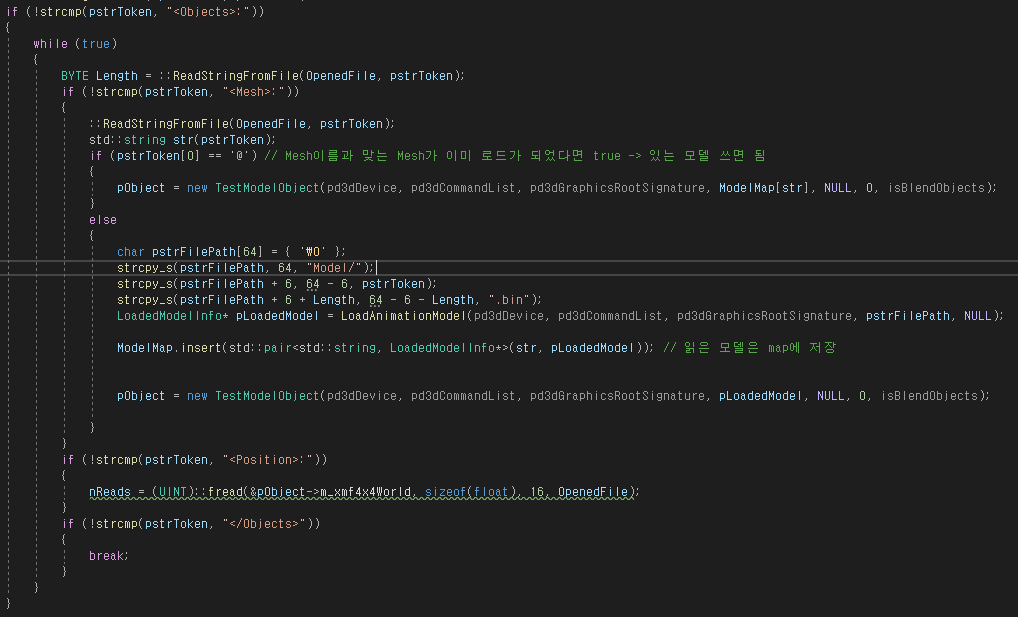
**맵 로드하기**

지난주에 CS파일로 유니티에서 파일을 추출하는 부분을 완성했다. 버그의 유무는 맵을 로드하는 부분을 만들고 생각한다. 파일을 뽑았으니 뽑은 파일을 우리의 게임에서 읽어 맵을 띄우기 위한 함수를 만든다.

맵 파일의 구조는 Mesh의 이름과 위치의 데이터만 포함 되어있다. 실제 모델의 데이터는 Mesh의 이름을 이용해 따로 읽고 읽은 모델을 이용해 객체를 생성한다.



오브젝트 생성자를 위와 같이 수정했다. Push\_List와 isBlendObject로 오브젝트 리스트에 추가를 할지, 추가를 한다면 어느 리스트에 추가를 할지 정해서 리스트로 오브젝트를 관리할 수 있다. 이제 오브젝트를 생성만 하면 알아서 리스트에 추가되어 관리할 수 있다.



맵을 불러올 때 사용할 LoadMapData함수의 일부분이다.

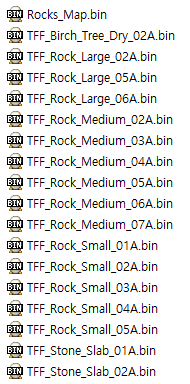
<Objects>: 문구를 만나면 루프를 시작해 <Mesh>: 와 <Position>: 의 데이터를 읽는다.

<Mesh>: 이후 Mesh의 이름이 입력되고 앞에 @가 붙으면 중복된 Mesh라는 뜻 이므로 pstrToken의 0번째 인덱스의 문자가 @이라면 이미 저장된 모델을 이용해 새로운 오브젝트를 생성한다.

만약 처음 불리는 Mesh라면 파일의 경로를 생성해서 LoadAnimationModel 함수를 이용해 모델을 읽는다. 읽은 모델을 다음에 중복되는 오브젝트에 사용해야 하기 때문에 따로 저장해 둔다. LoadedModelInfo 클래스에는 Mesh의 이름을 따로 저장하지 않는다. 저장하도록 만들려면 기존의 함수를 고쳐야 하기 때문에 이름과 모델의 포인터 주소를 같이 저장할 수 있는 map 컨테이너로 저장하고 Mesh의 이름을 string으로 변환해 Key로 사용하고 모델의 포인터 주소를 데이터로 map에 저장했다. 그래서 @가 붙었을 때의 경우를 보면 오브젝트를 만들 때 모델을 넣어주는 인자부분에 ModelMap[str]을 넣은 모습을 볼 수 있다.

<Position>: 부분은 행렬이 입력되므로 월드 행렬에 직접 넣어준다. </Objects>가 나오면 맵의 모든 데이터를 읽었다는 뜻이기에 break로 루프를 빠져나가 함수를 종료한다.

이제 유니티에서 맵을 추출해 실제로 띄워본다.



우선 바위 오브젝트들만 뽑아봤다.

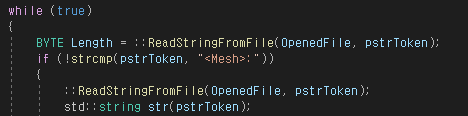
Rocks\_Map.bin가 바위 오브젝트들의 Mesh와 위치 데이터 파일이고 밑의 TFF\_파일들이 바위 모델 데이터 이다.



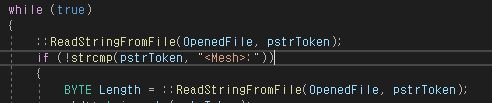
Scene의 BuildObject 부분에 LoadMapData를 불러준다. 오브젝트 생성 과정에서 알아서 리스트에 추가되므로 반환을 따로 저장할 필요가 없다. 애초에 void 형이다.



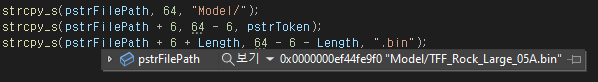
실행을 해보니 파일이름이 제대로 들어가지 않아 문제가 되고 있다.



Length는 모델의 이름의 길이를 나타내야하는데 <Mesh>:를 읽는 부분에서 길이를 재고 있었다.

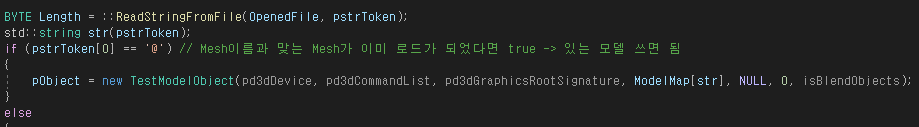


위치를 <Mesh>: 이후로 옮겨줬다.



정상적인 파일 경로가 들어가는 모습을 확인할 수 있었다.

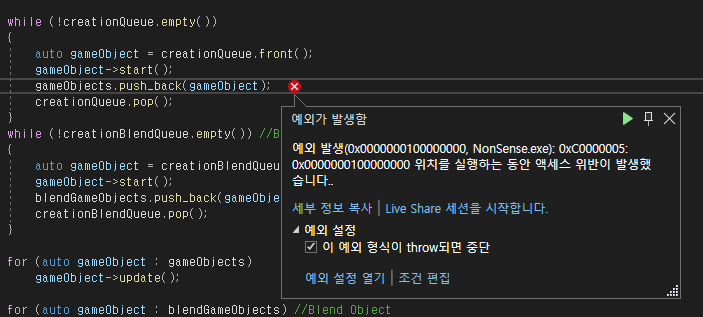
디버그 과정 중 오류를 하나 더 찾았다.



이 부분인데, @가 포함된 문자열을 str로 만들고 이 str로 모델을 찾으려 하니 제대로 찾을 수 없다.



@가 붙어있을 때는 포인터에 +1을 해줘서 @를 뺀 나머지 문자열을 저장한다. 정상적으로 들어가는 모습을 볼 수 있다. @가 없을 때는 처음과 같이 그대로 string으로 변환해준다.

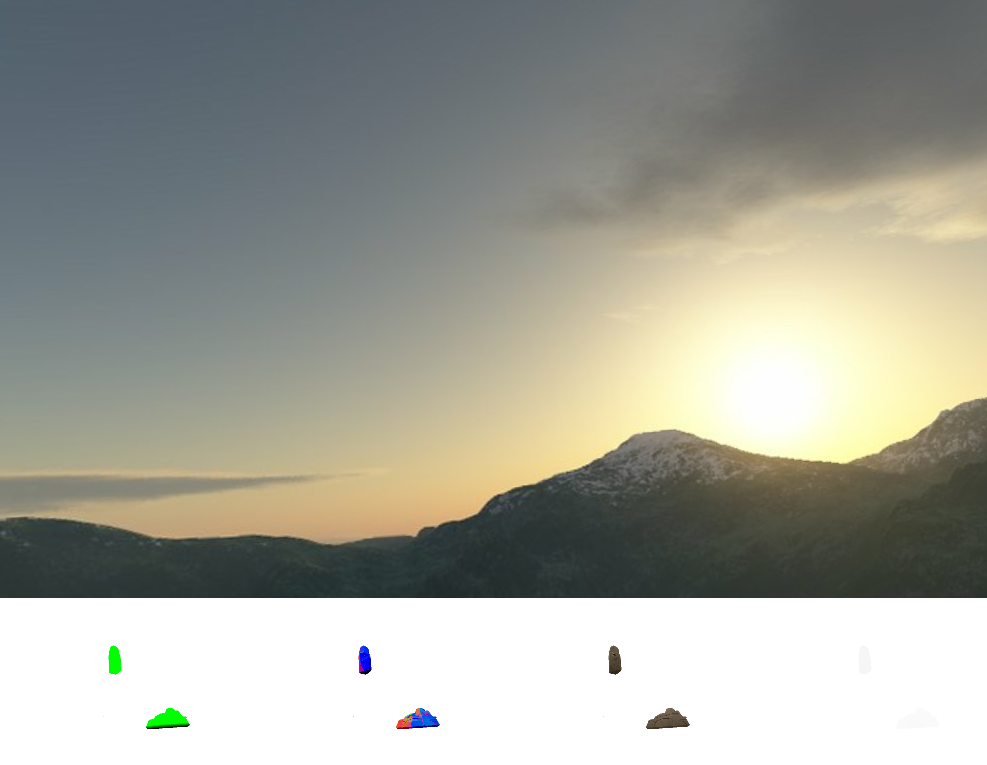


오브젝트 리스트에 생성한 오브젝트를 추가하는 부분에서 문제가 생겼다.

블랜딩을 해줄 오브젝트와 블랜딩 없이 디퍼드로 랜더링 할 오브젝트를 나눠야 할 필요가 있었기 때문에 생성자를 새로 만들어 주었었다. 이에 맞게 TestModelObject의 생성자도 알맞게 변경을 했어야 했다.



isblendObject라고 하는 지금 읽는 모델이 블랜딩 오브젝트인가를 판단하는 bool변수를 받고 그 변수를 이용해 새로 만든 생성자를 불러주도록 만들었다.

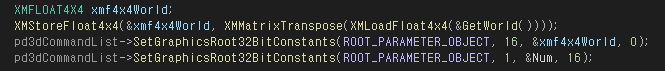


다른 렌더타겟에서는 보이지만 메인 화면에서는 보이지 않는다.

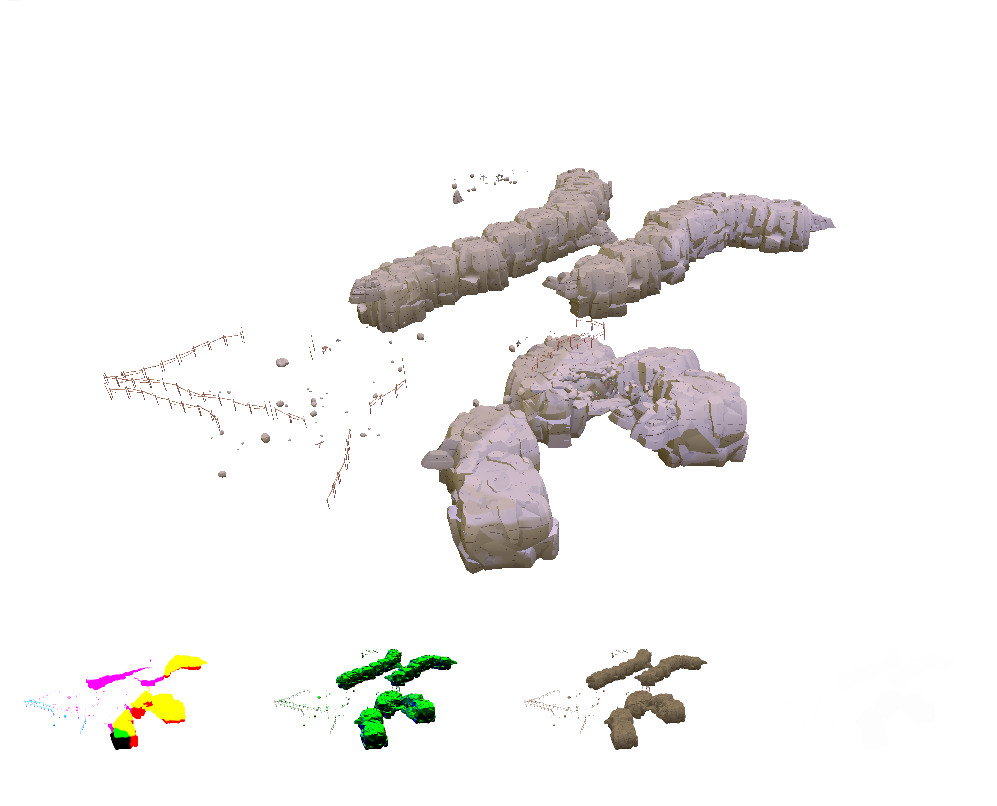


가까이 접근하니까 보인다. 스카이 박스에 가려졌다는 사실을 알아냈다.

근데 실제 추출한 맵보다 바위가 적게 그려졌다. 모델을 불러오긴 했지만 불러온 모델을 재사용해서 공유할 수는 없게 되어있다.

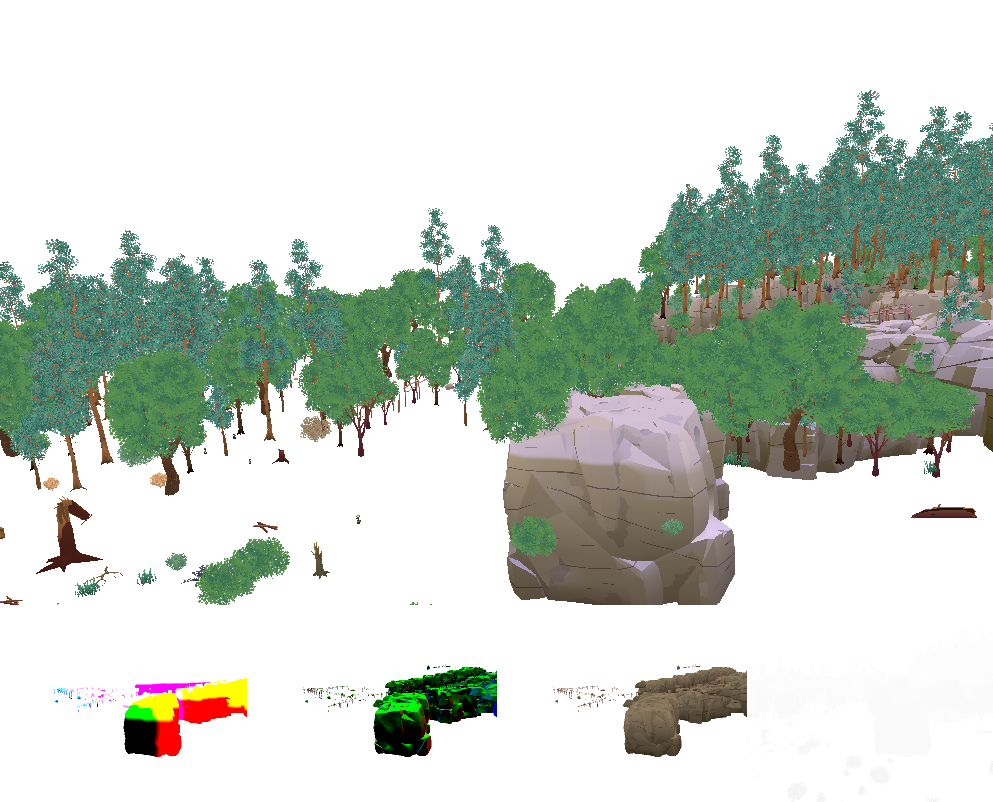


오브젝트의 월드 행렬을 쉐이더에 넘겨주기 위해 CBV를 사용하던 방법에서 32BitConstants를 이용해 직접 넣어주는 방법을 사용했다.



모든 오브젝트가 잘 그려지게 되었다. 같은 오브젝트를 그리려고 CBV에 값을 넣어주는데 포인터의 위치는 변하지 않은 상태로 값만 바꿔서 결국 마지막 오브젝트의 값만 들어가서 한 개 씩 그려지는 것 아닌가 하고 생각은 하고 있지만 정확한 이유는 모르겠다. 좀더 찾아봐야겠다.

테스트맵의 모든 오브젝트를 추출해서 띄워봤다.



13주차에서 구조물들을 띄웠을 때 보다 비교도 안될 정도로 로딩 속도가 빨라졌다. 또 왜인지는 모르겠지만 CBV로 데이터를 넘겨줄 때와 다르게 렉이 없이 부드럽고 나무와 같은 오브젝트의 색도 문제없이 잘 나온다. CBV와 32BitConstants 사이에 많은 차이가 있는 것 같다.