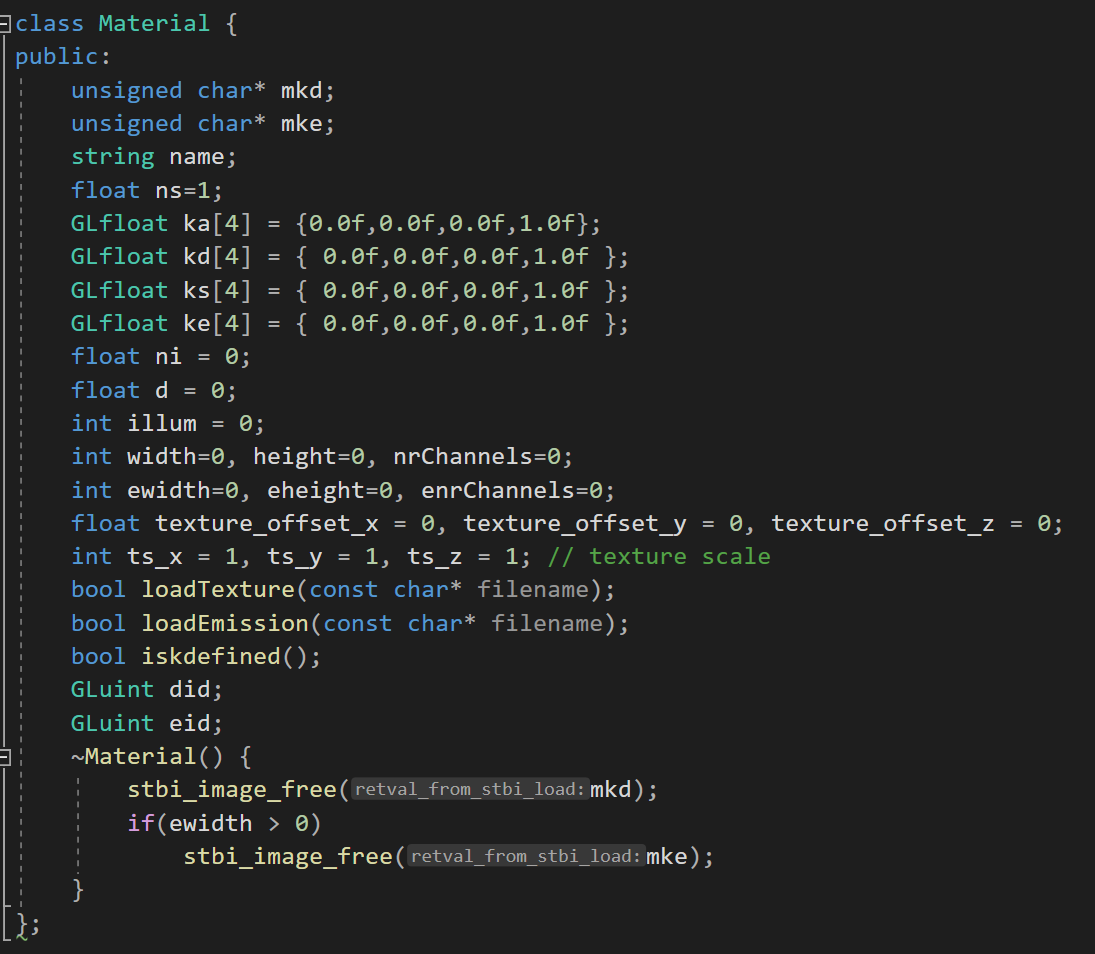
1. 모델을 담을 클래스 저장

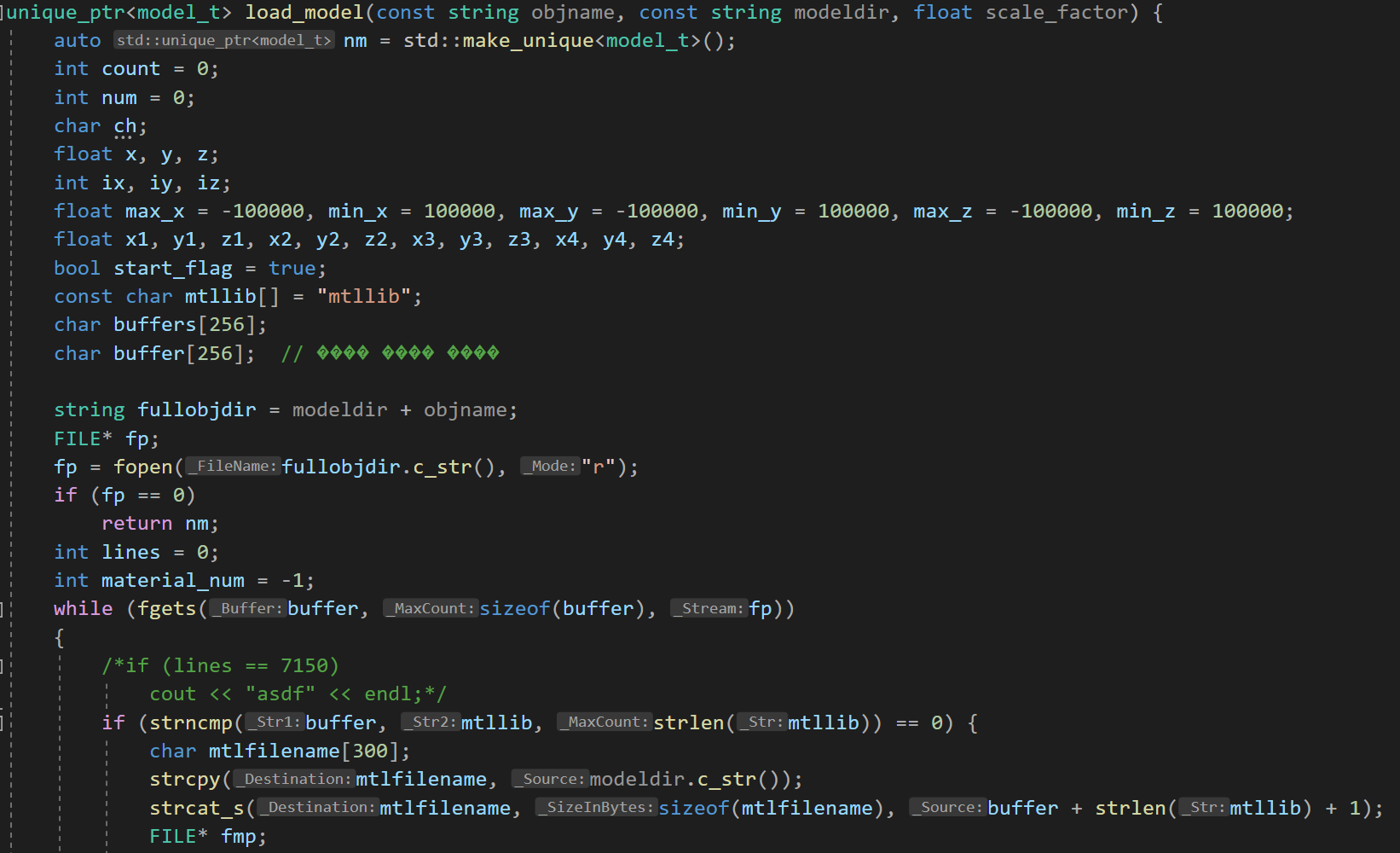


모델에 포함되는 vertex, 텍스쳐 매핑, 폴리곤 정보, 머테리얼 정보를 저장하고 회전과 이동, 복사 생성을 위한 클래스를 정의했다.

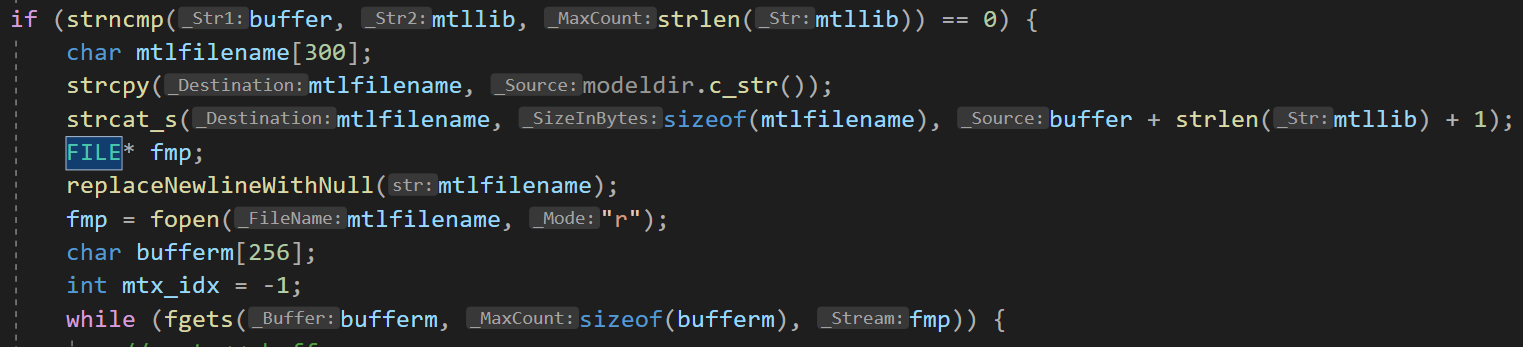


머테리얼, 즉 텍스쳐 정보는 메모리를 많이 사용하기 때문에 shared pointer를 사용하여 여러 오브젝트가 하나의 머테리얼 정보를 공유할 수 있게 하였다.

1. Obj 파일 로드



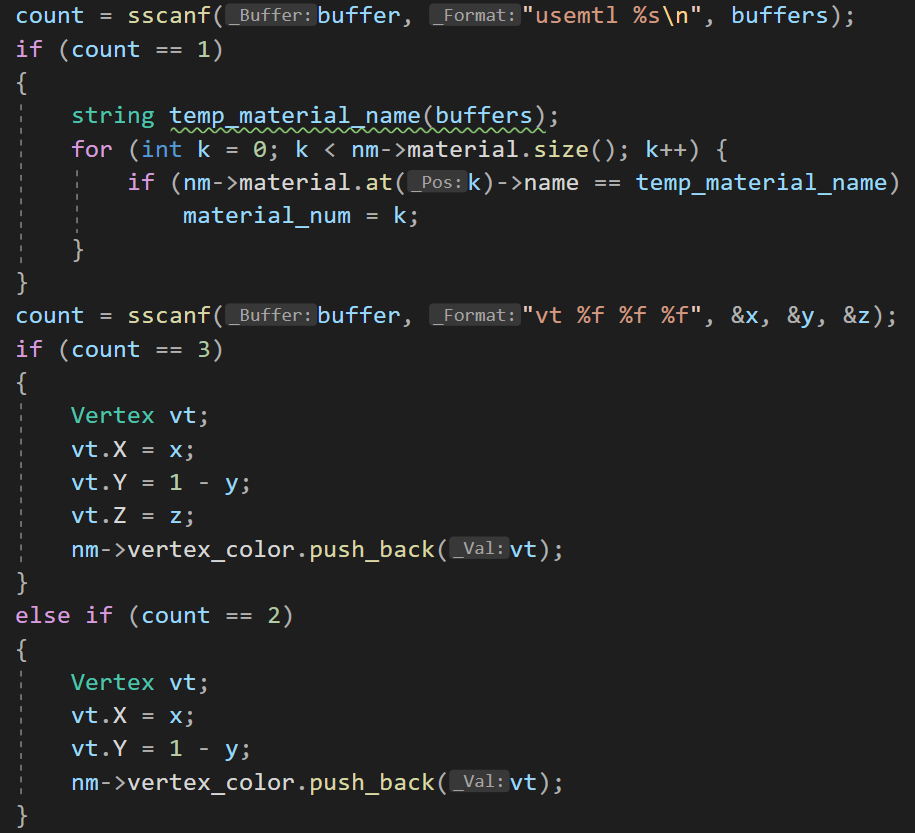
Obj 파일을 로드하여 객체에 담고 해당 객체의 포인터를 반환한다.



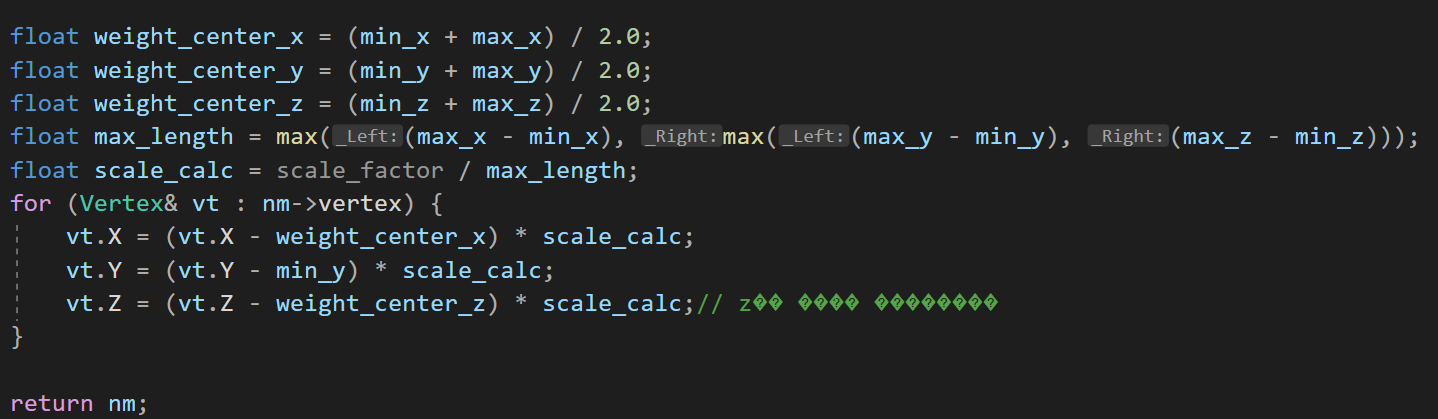
Mtllib 구문이 인식되면 해당 mtl 파일을 불러와 머테리얼 정보를 저장하고 텍스쳐 파일을 불러온다.



-o나 -s와 같은 다양한 옵션들의 경우 각각 분리하여 처리하였다.

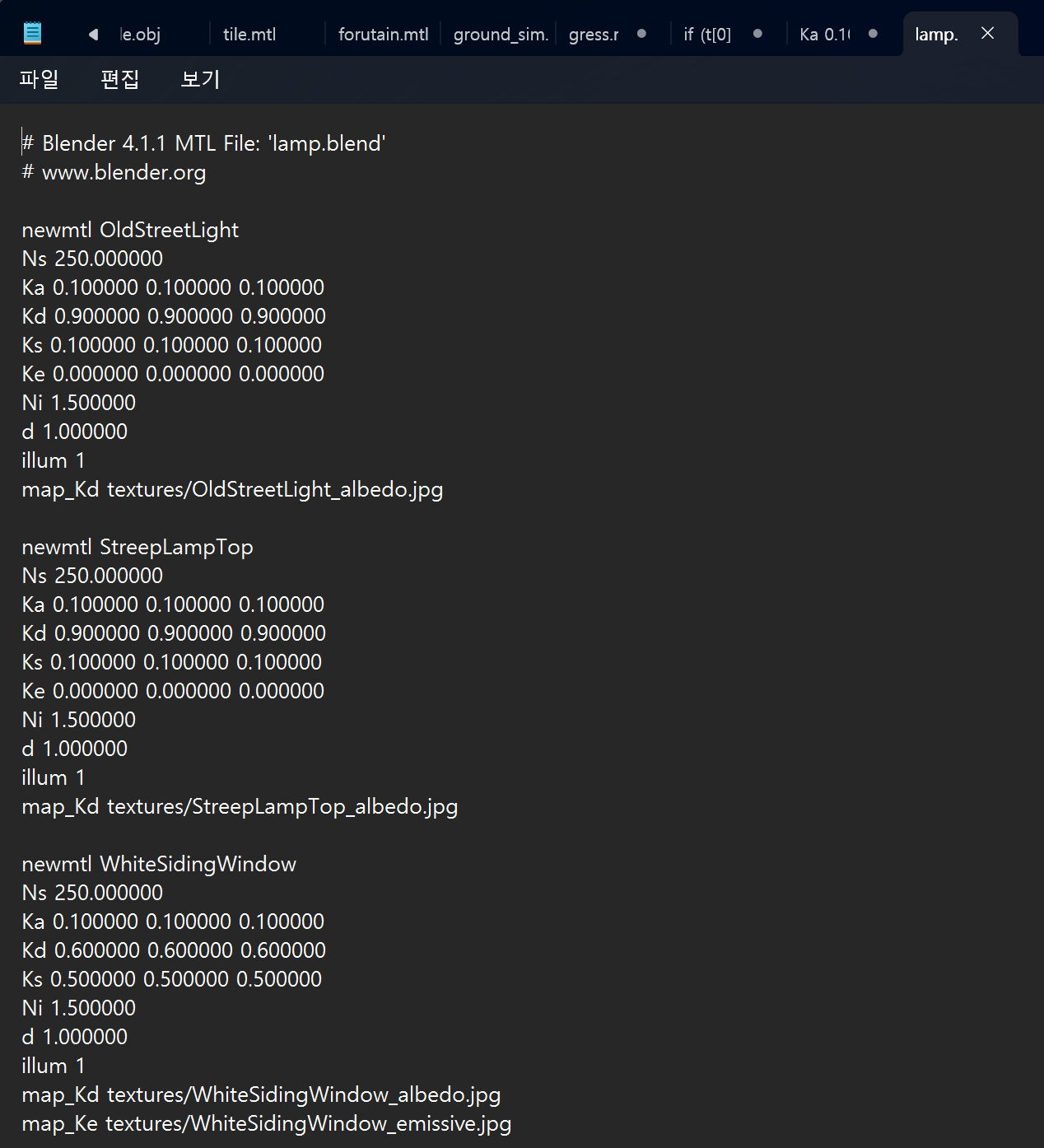


Mtl 파일 인식 후, v, vt, f등을 처리하고 usemtl 구문이 나왔을 경우 이후에 나오는 폴리곤에 대한 머테리얼 설정을 변경하여 저장하도록 하였다.



Obj 파일을 모두 읽은 후, 모델의 최대 및 최소 좌표를 계산하여 최대 길이가 1\*scale\_factor가 되도록 만들어주었고 모델의 중심을 0, mean\_y, 0에 위치하도록 조정하였다.

1. Mtl 파일 설정

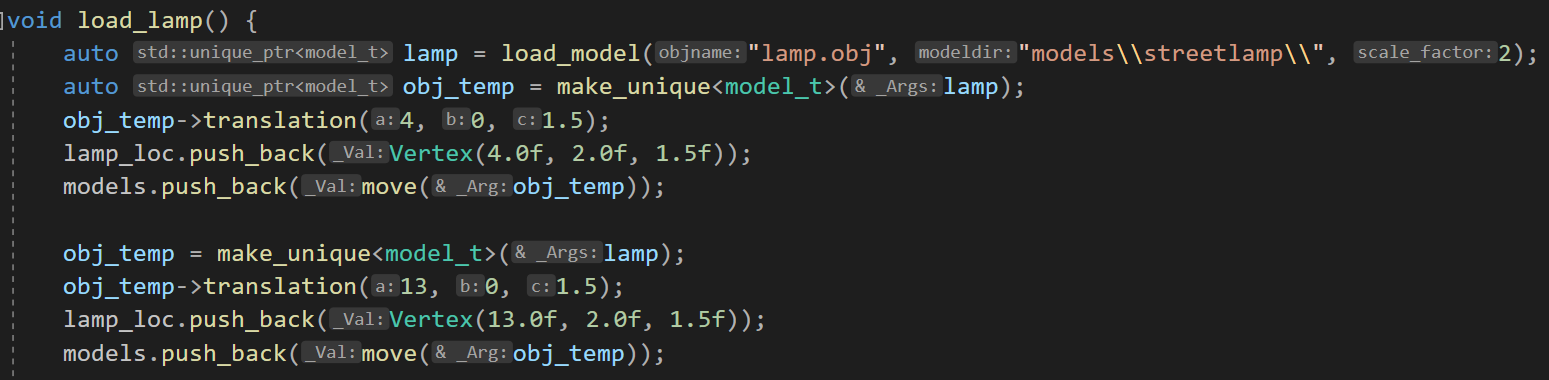


각 모델의 mtl 파일을 수정하여 각 부분마다의 머테리얼에 맞게 ambient, diffuse, spacular 값을 조정해 주었고 추가로 가로등과 같이 빛을 내는 물체들은 emission을 설정해 광원이 없는 곳에서도 스스로 빛나게 설정해주었다.

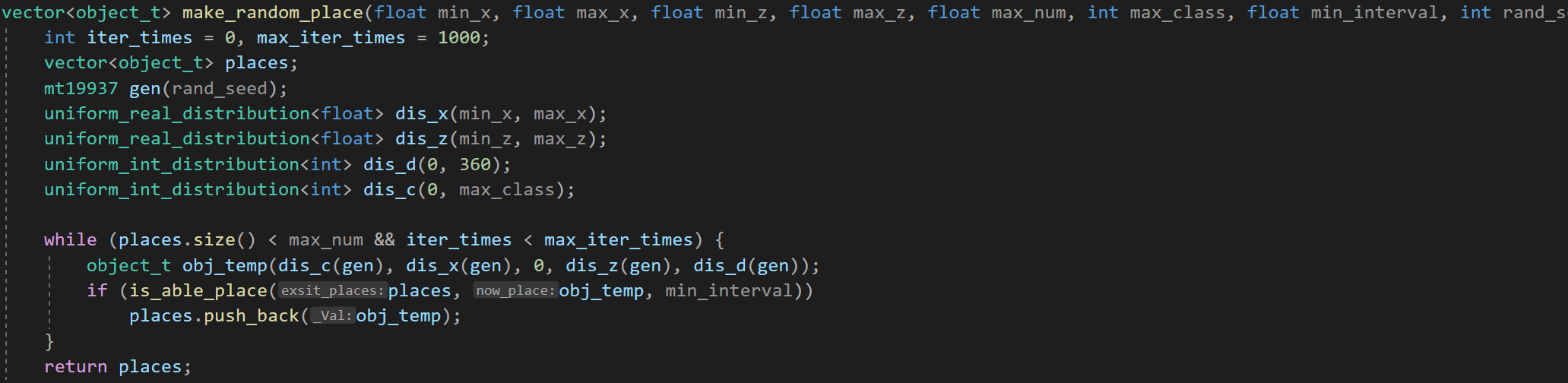
1. 오브젝트 로드 및 배치



위에서 생성한 함수를 이용하여 오브젝트를 로드하고 회전 및 이동을 적용하여 배치시켰다.

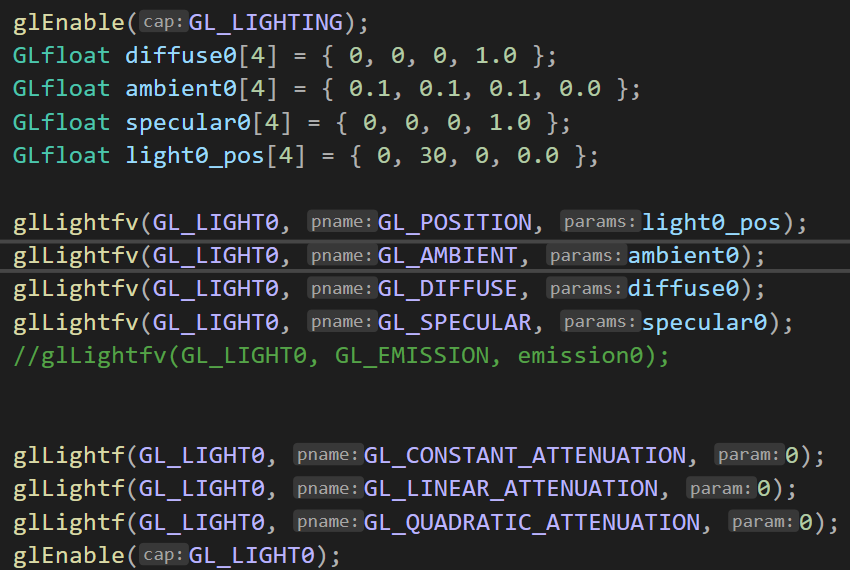


가로등 같은 경우 추후 광원설정을 위해 가로등이 설치된 위치까지 같이 저장해주었고,

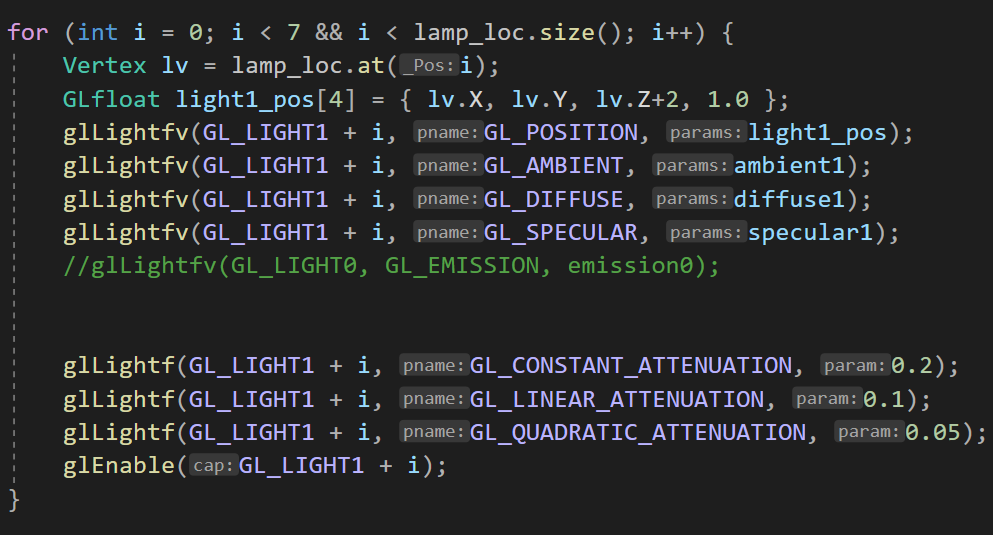


나무와 풀 등 랜덤성이 필요한 오브젝트들은 랜덤 배치함수를 제작하여 배치하였다.

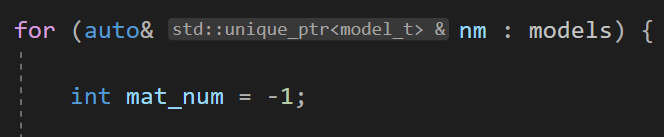
1. 디스플레이



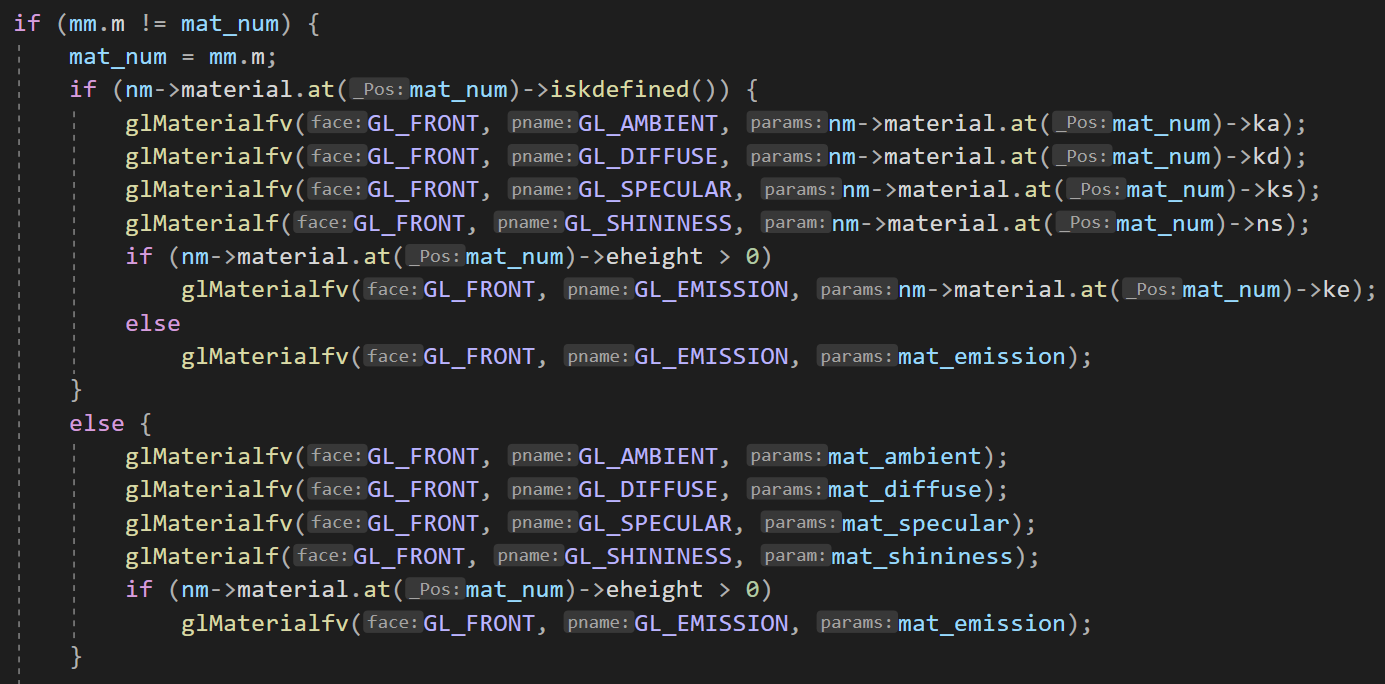
전체적인 ambient를 위해 감쇄없이 직진하는 direct light를 설정해 주고



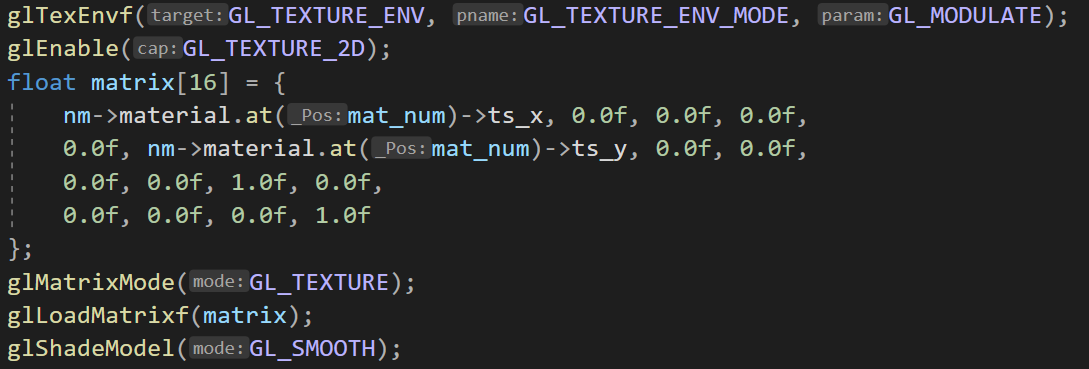
가로등 위치에 맞게 point light 형태의 추가 광원을 설정해 주었다.



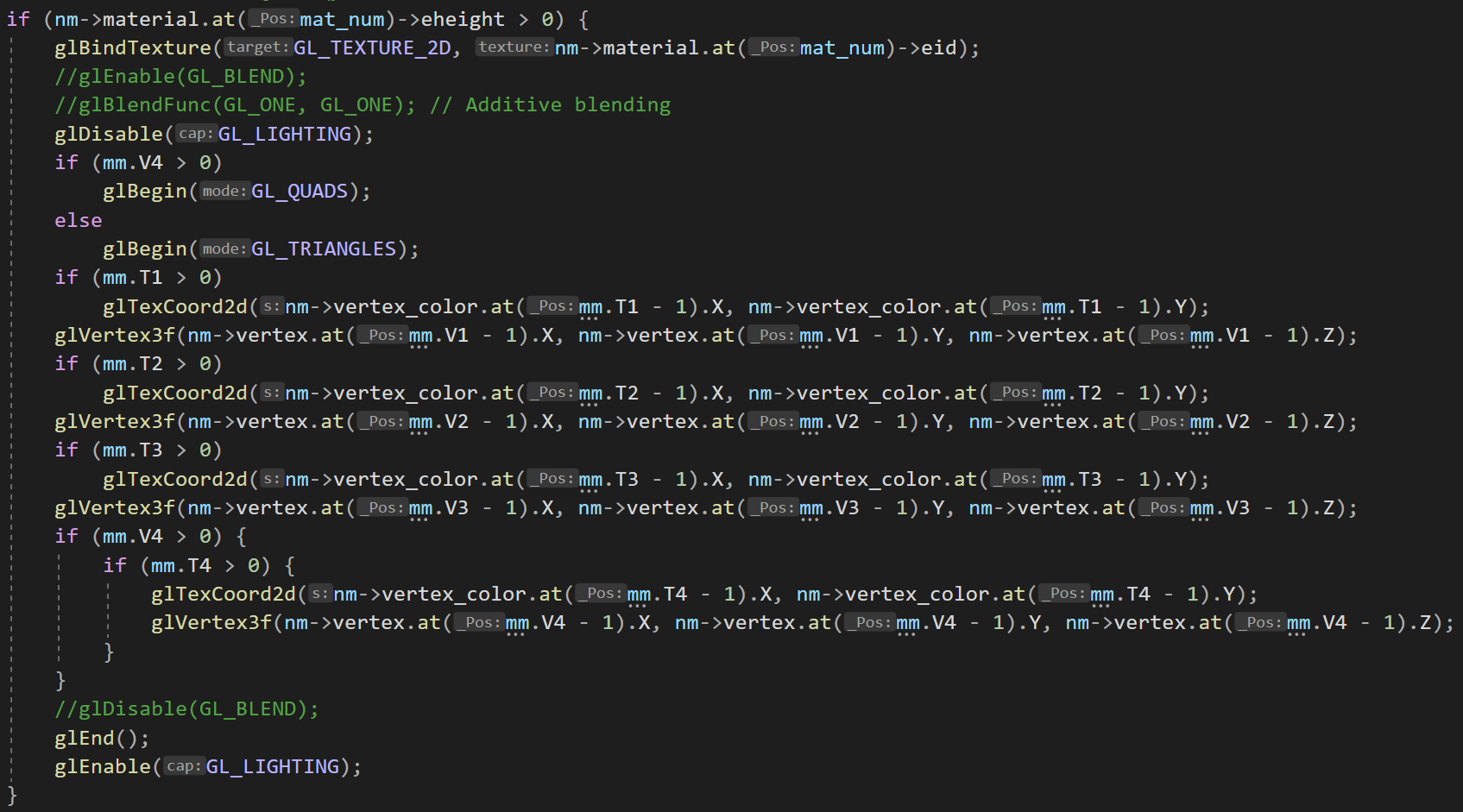
이후 각 오브젝트마다 아래의 과정을 반복한다.



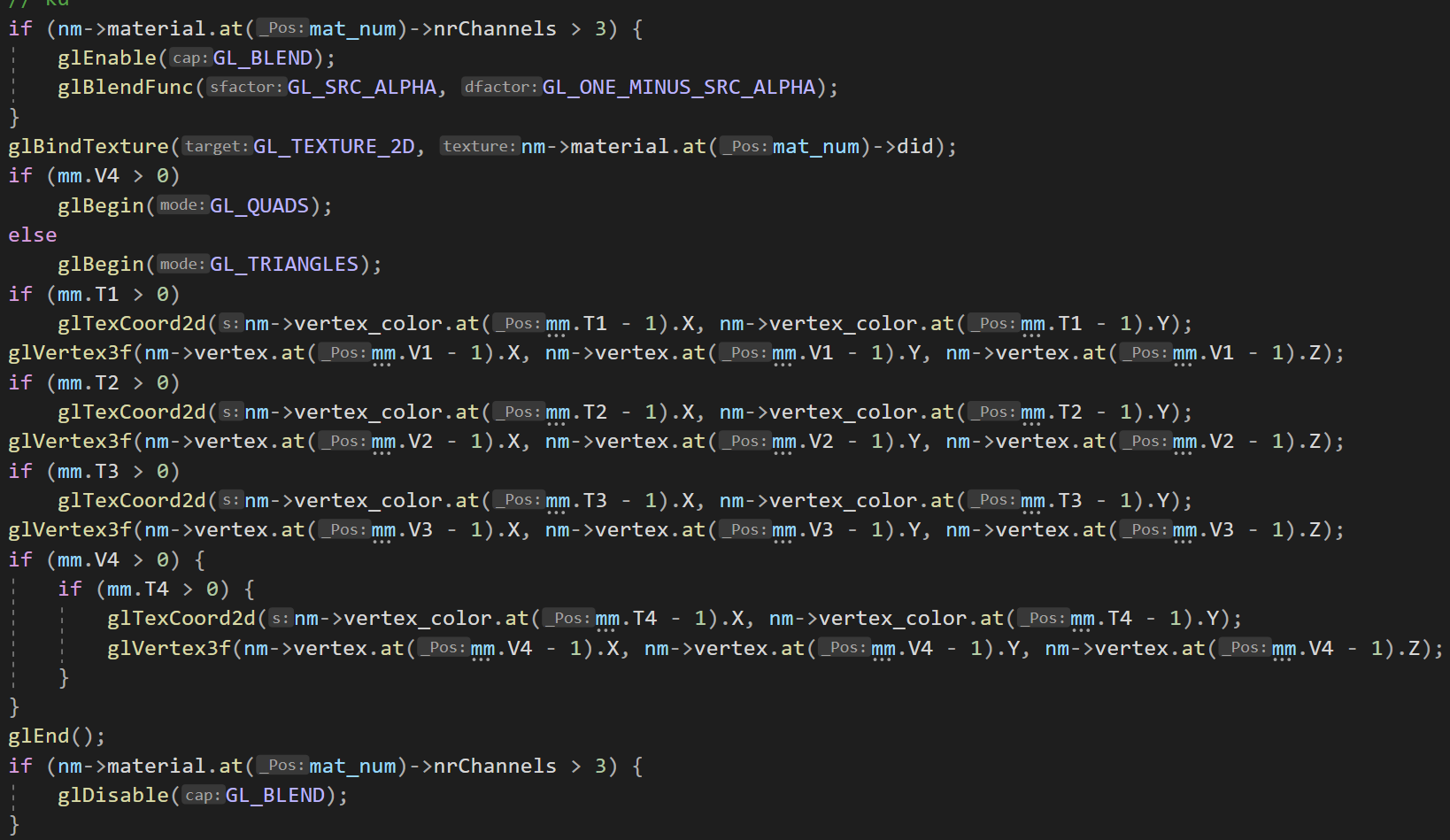
머테리얼이 바뀔 경우 머테리얼을 재설정 해준다.



텍스쳐 크기조절과 Gouraud 쉐이딩을 적용해 주었다.



Emission 텍스쳐가 있을 경우 고정 파이프라인에서는 area light를 구현할 방법이 없어서 단순하게 해당 텍스쳐가 조명의 영향을 받지 않게 만들어 조명의 영향을 받지 않고 모든 상황에서 보이게 만들어주었다.



만약 텍스쳐의 채널이 4일경우 4번째 채널은 알파값을 적용할 수 있게 블랜드 옵션을 활성화 시켜주었고 triangles와 quads를 구분하여 랜더링하게 만들었다.