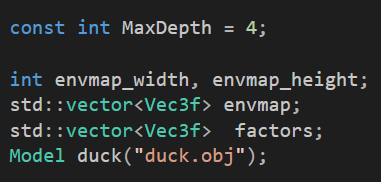
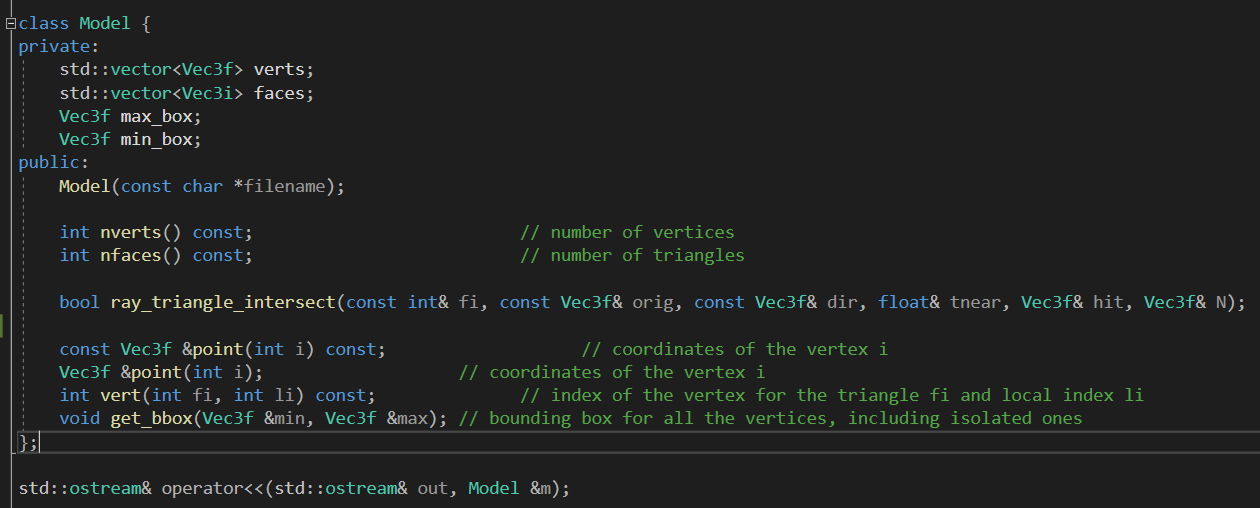
**컴퓨터 그래픽스 프로젝트 3 – Soft Shadow**

**20182931 신석경**

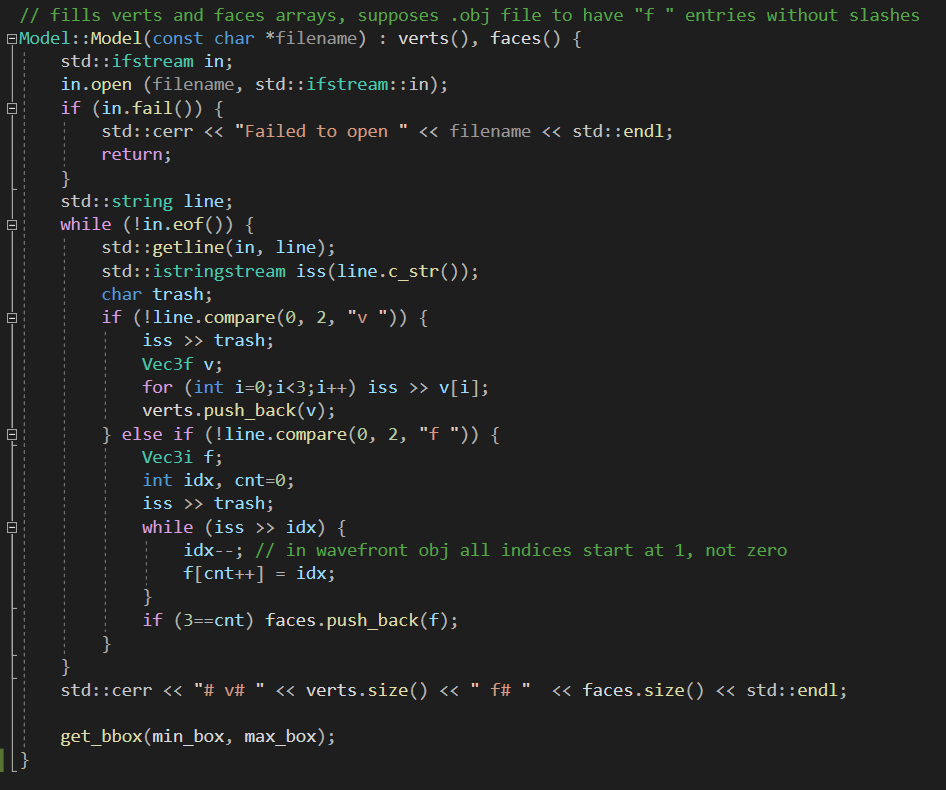
1. **코드 설명**



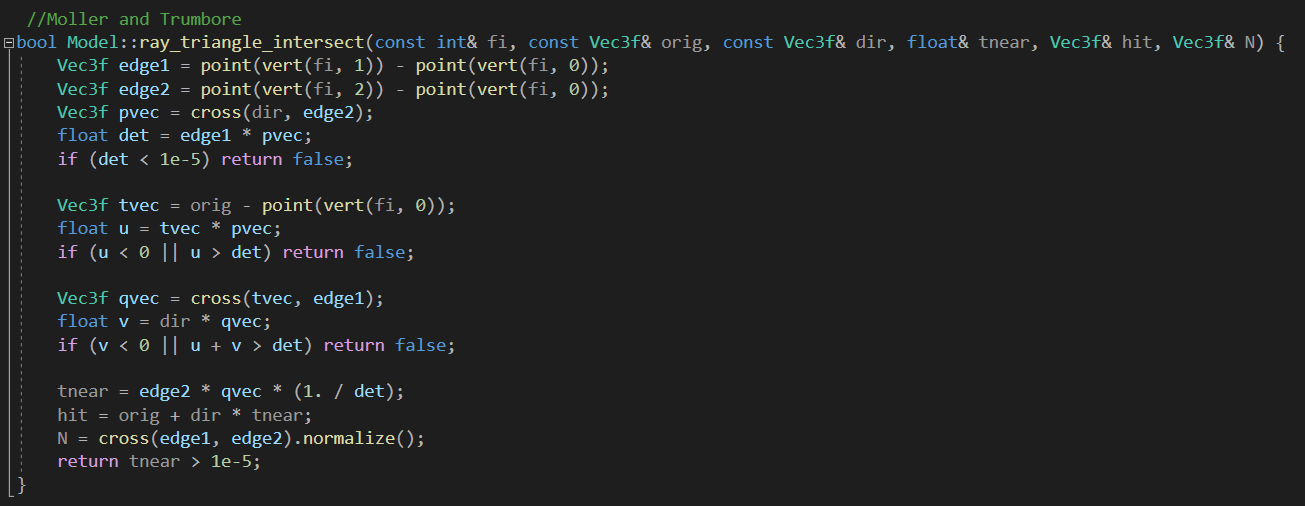
먼저 최대 Depth 크기를 정하고, 필요한 변수를 선언한다. envmap의 경우 환경 맵을 저장하는 변수로 사용하고, factors의 경우 이후에 Light Source를 원으로 설정하는데, 이때, distribute를 나타내기 위한 벡터들의 모음이다. 그리고 duck.obj를 duck이라는 변수로 모델을 불러온다. 이 Model에 관해 자세히 살펴보자.



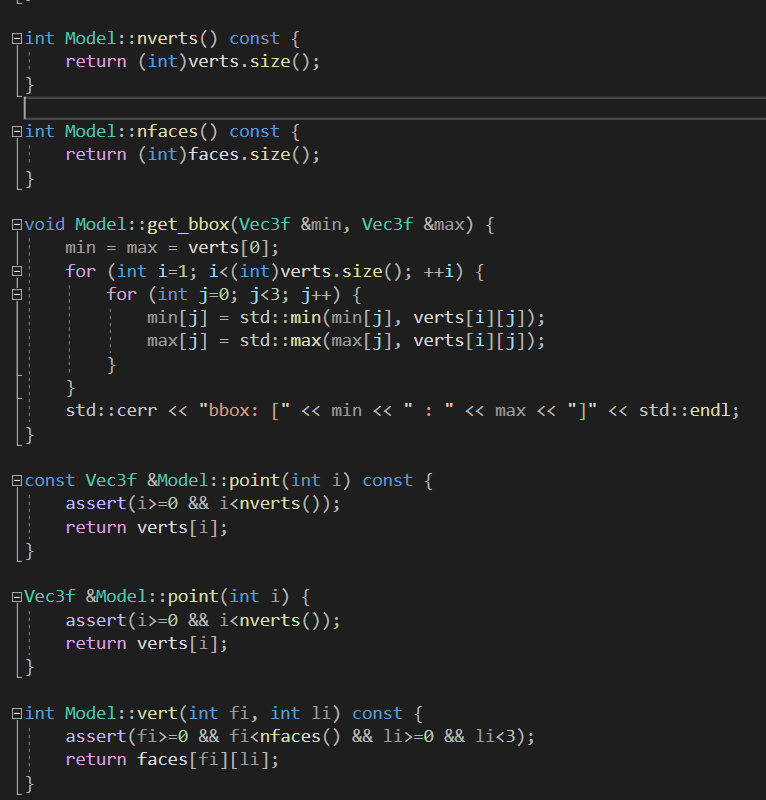
먼저 클래스의 정의이다. 변수로는 vertices를 저장하는 verts 변수와, triangle를 저장하는 faces 변수가 있다. 그리고 모델의 전체 크기의 최대 박스와 최소 박스는 저장하는 max\_box와 min\_box가 존재한다. 메소드를 살펴보면, 먼저 생성자가 있고, vertices의 개수를 반환하는 nverts, trianlges의 개수를 반환하는 nfaces 함수가 있다. 그리고, ray\_triangle\_intersect 함수가 있는데, 원래 함수의 정의에는 hit과 N을 구하는 부분이 없어 추가하였다. 이 함수의 경우 주어진 vertices에서, 원점 orig와 프레임 버퍼까지의 방향 dir에 해당하는 값을 입력 받아, 화면에 보이는 가까운 거리 tnear를 구하고, 해당 면에 들어오는 벡터인 hit과 노멀벡터 N을 구하는 함수이다. point 메소드는 vertex의 좌표를 구하는 메소드이고, vert 메소드는 triangle index와 그것과 관련된 vertex 좌표를 알아오는 메소드이다. get\_bbox는 모델을 감싸는 전체 바운딩 박스를 구하는 메소드이다. 이제 구현부를 살펴보자.



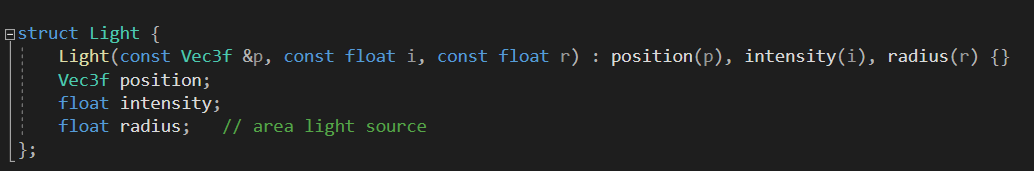
먼저, 생성자이다. obj 파일을 읽어와, vertex 정보이면 verts에 저장하고, triangle 정보이면 faces에 저장한다. 그리고 get\_bbox를 호출하여 전체 bounding box를 구한다.



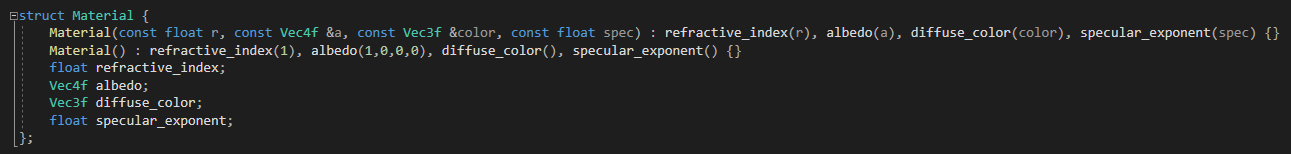
ray\_triangle\_intersect 함수이다. triangle index fi를 입력 받아 해당 triangle과 원점인 orig, 원점부터 해당 triangle 까지의 방향인 dir를 비교하여 해당 dir 방향이 triangle 내부에 해당하는지 확인한다. 이 과정은 Moller and Trumbore 알고리즘을 사용하여 계산한다. 이후 orig부터 triangle까지의 거리인 tnear를 구하고, 그에 해당하여 해당 triangle로 들어가는 빛의 벡터인 hit과 노멀벡터를 구한다. 이후 tnear가 양수 인 경우에만 true를 반환한다.



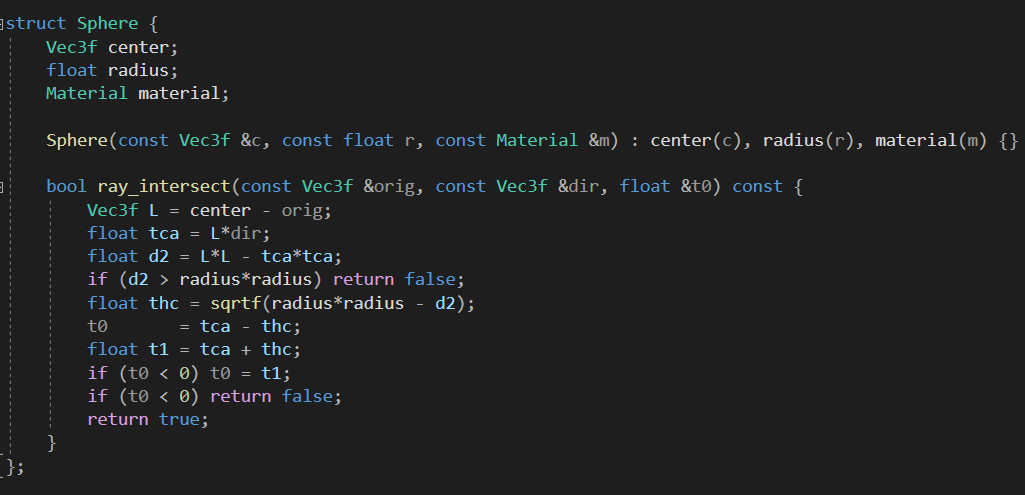
나머지 함수에 대해서는 구현이 간단하여 생략하도록 하겠다.



다시 main 코드로 돌아와 light를 정의한 Light 구조체에 대해 알아보겠다. position과 intensity와 radius를 입력 받아 각각의 값을 할당한다. 이때, 우리는 area Ligth source를 사용할 것이기 때문에 radius를 추가해주었다.

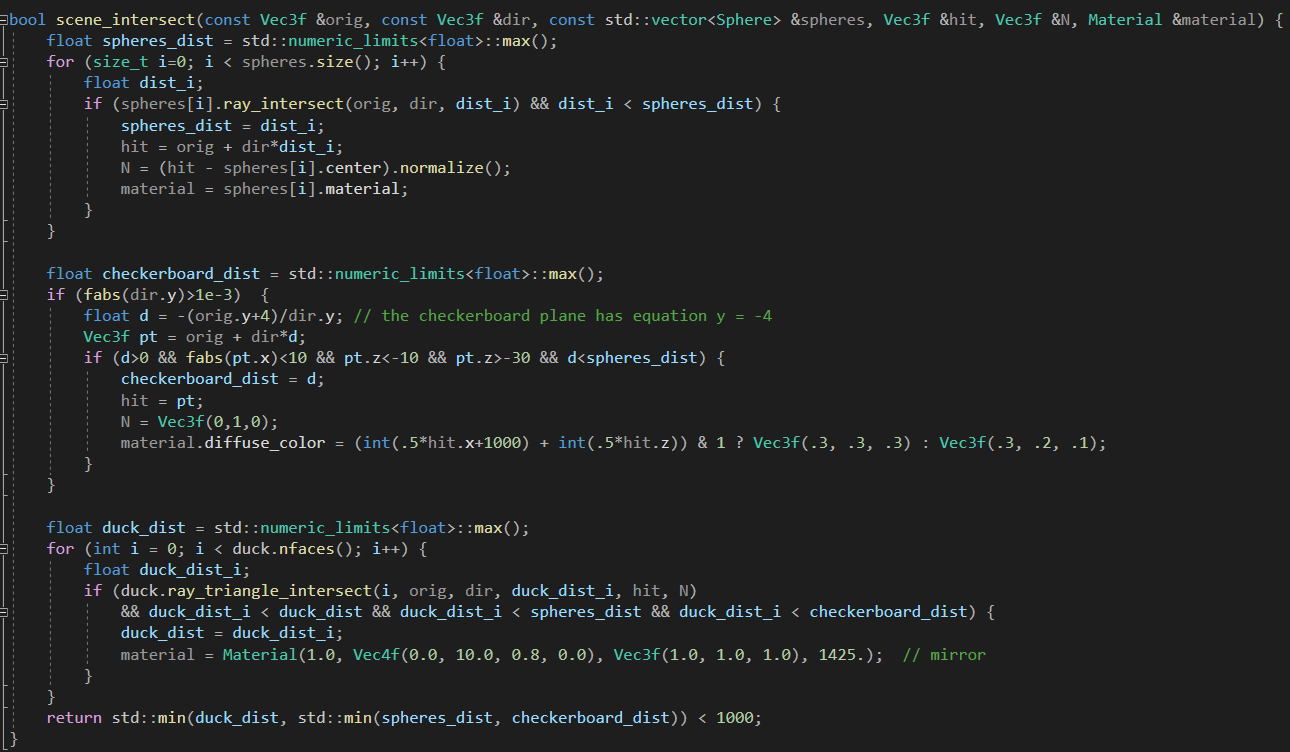


물체의 질감을 나타내는 Material 구조체이다. refractive\_index와 albedo, diffuse\_color, specular\_exponent를 정의하여 준다.



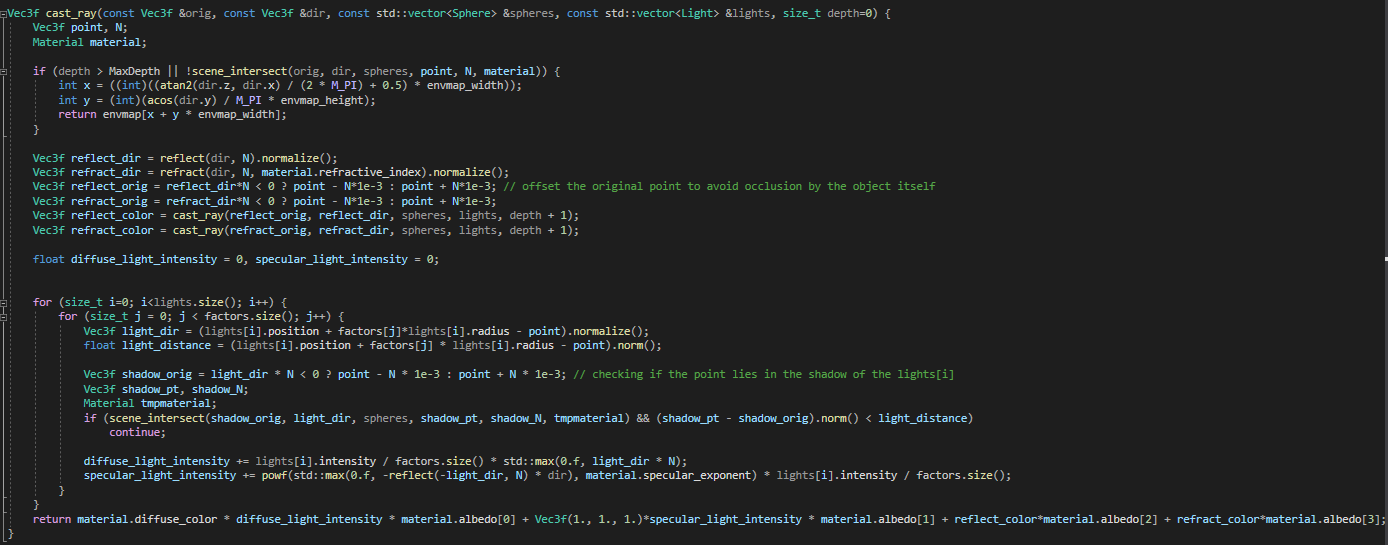
구를 나타내는 Sphere 구조체이다. 먼저 구의 중심인 center, 구의 반지름인 radius, 구의 질감을 나타내는 material 변수를 선언한다. Model과 마찬가지로 이 또한 ray\_intersect가 필요하여 선언되어 있었다. ray가 원과 만나는지 검사하고, 그 만나는 점 중 가까운 점을 t0, 멀리 있는 점을 t1이라 하여 둘 중 한 점이라도 양의 정수인지 확인하여 true를 반환한다.

반사와 투과 계산 방법이다. 반사의 경우, 기존의 반사 식이였던 R = 2\*N\*dot(N,L) - L에서 -를 붙여 들어오는 방향으로 L-2\*N\*dot(N,L)을 계산한 것이다. 투과의 경우 Snell’s Row를 계산한 것이다.



ray가 scene의 어디에 casting 되는지 결정하는 scene\_intersect이다. 원점 orig과 들어오는 ray 방향 dir, 원들을 입력 받아, 들어오는 벡터인 hit, 노멀벡터 N, 질감 material을 반환하는 함수이다.

먼저 모든 sphere에 대하여 ray\_intersect를 계산하여 해당 ray 방향에 있고, 가장 가까운 거리에 있는 원의 거리와 hit 벡터, 노멀벡터, material을 계산한다. 이후, Scene 하단에 checkboard가 있다고 가정하고, 해당 checkboard와 비교하여 거리 d가 spheres\_dist보다 가까운 경우 그 ray에 대해 hit, N, material를 설정해준다. 이후, 오리 모델에 대해서도 비슷한 과정을 진행한다. 오리의 모든 triangle에 대하여, ray\_triangle\_intersect를 계산해 그 때의 거리가 기존의 원, checkboard, 이전에 계산된 duck triangle과의 최소값 보다 작은 경우, 그 triangle에 대한 hit, N, material를 설정해준다. 이때, 이미 hit과 N은 ray\_triangle\_intersect에서 처리해주었기 때문에 material만 조작해주면 된다. 이때, mirror에 해당하는 Material로 설정해 주었다. 이후 모든 intersect를 검사하고 그 값이 1000보다 작은 경우에만 true를 반환해준다.

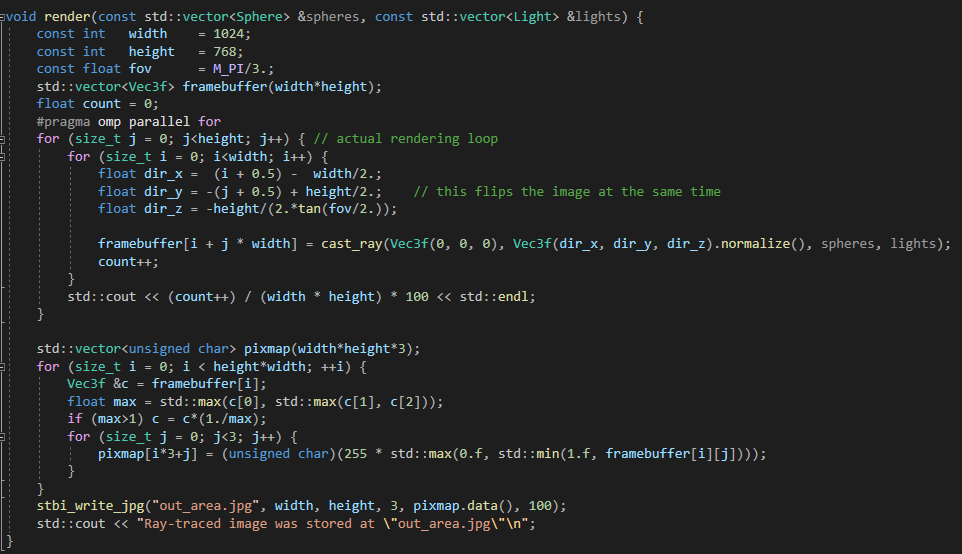


ray casting의 실제 동작이다. 먼저 scene\_intersect를 검사하여 아무 것도 ray에 걸리지 않는다면, 즉 아무것도 ray와 만나지 않는다면 배경색으로 설정해주어야 한다. 이때, load된 envmap에 해당하는 u, v 좌표를 계산한다. 현 프로젝트에서 사용하는 envmap의 경우 spherical environment map이기 때문에 해당 좌표로 변환해주어 texture mapping을 해주었다.

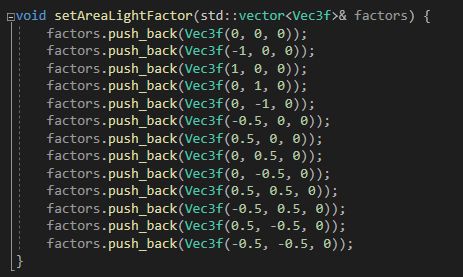
만약 ray에 걸리는 것이 있다면, hit, N, material에는 해당 물체에 대한 정보가 들어있을 것이다. 이것을 바탕으로 reflect, refract에 해당하는 방향, 원점, 색깔을 구한다. 이때, 색깔의 경우, reflect, refract를 하는 해당 좌표를 기준으로 다시 cast\_ray를 호출하여 재귀적으로 모든 반사와 투과를 계산할 수 있다.

이후 Light Source에 대하여 shadow를 계산한다. 이때, light source를 area로 결정했기 때문에 기존의 light source 위치에서 factor[j]에 해당하는 벡터를 반지름과 곱해 여러 개의 distributed raytracting을 계산할 수 있다. light source의 방향과 거리를 계산하고, shadow의 원점을 계산한다. 이후, shadow의 원점과 light의 방향을 기반으로 다시 scene\_intersect를 계산한다. 만약 계산이 올바르게 되고, shadow가 만나는 점 shadow\_pt와 shadow 원점 shadow\_orig 사이의 거리가 light source와의 거리보다 작으면 다른 light source에 대해서 계산한다. 만약 거리가 더 크면 해당 점에 대하여 diffuse\_light\_intensity와 specular\_light\_intensity에 대해서 계산한다. 이때, factors.size()로 각각의 light[i].intensity를 나누어 줬는데, 이것은 area light source로 변환하면서 빛의 밝기가 분산되었다고 가정했기 때문이다. 실제로 나누지 않고 계산할 경우 너무 밝아 saturation이 발생하는 경우가 있었다.

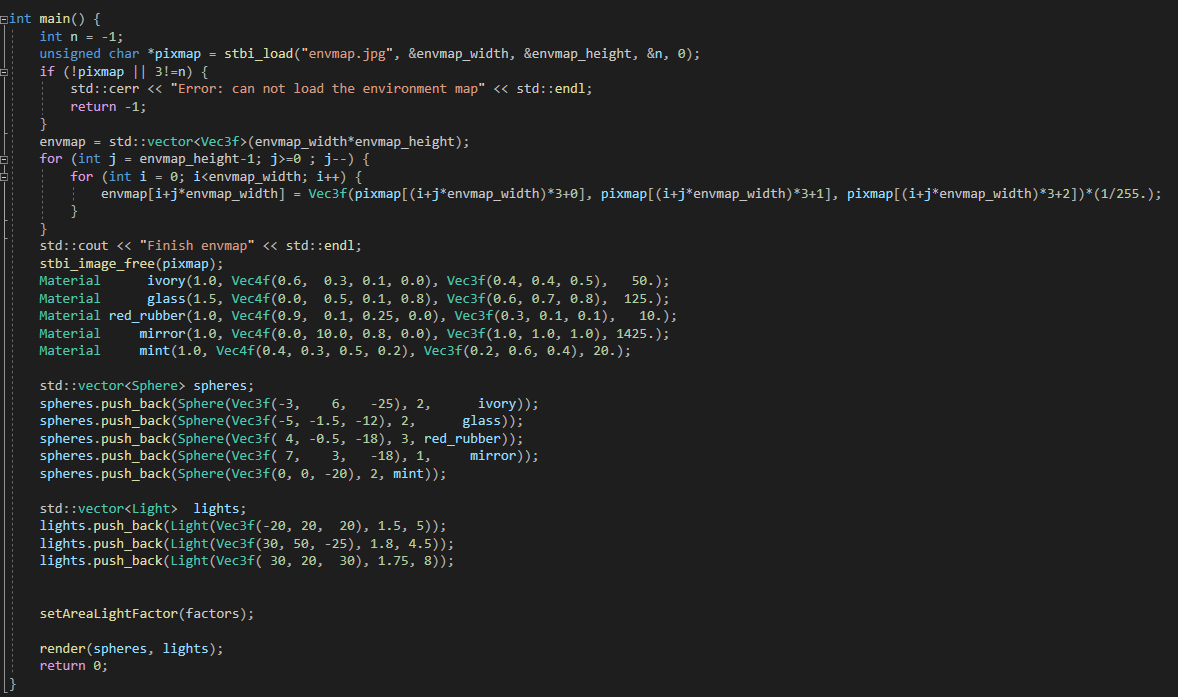
이제 해당 점에 대해 모든 조건을 계산하였으므로 material의 각각의 값에 light source의 값을 곱해 색을 반환해준다.



실제 rendering을 진행하는 render 함수이다. 원과 빛을 입력 받는다. 이제 하나의 frame을 1024 X 768 크기로 결정하고, 한 픽셀 마다 접근하면서 (0,0,0)부터 해당 픽셀 까지의 dir를 구하여 cast\_ray를 진행한다. cast\_ray의 최종값이 해당 픽셀에서의 색깔이기 때문에 그것을 framebuffer에 할당해준다. count의 경우는 사용자가 얼만큼 render되었는지 확인하기 위해 추가해 주었다. 이제, pixmap을 할당해준다. 기존에는 Vec3f로 저장했지만 pixmap에는 unsigned char로 저장하기 때문에 이미지의 크기에 RGB에 해당하는 3을 곱해 크기를 키워준다. 그리고 framebuffer 한 픽셀에 해당하는 값을 3개의 값으로 pixmap에 저장해준다. 그리고 stbi\_write\_jpg를 사용하여 해당 pixmap의 값을 out\_area.jpg로 생성해준다.

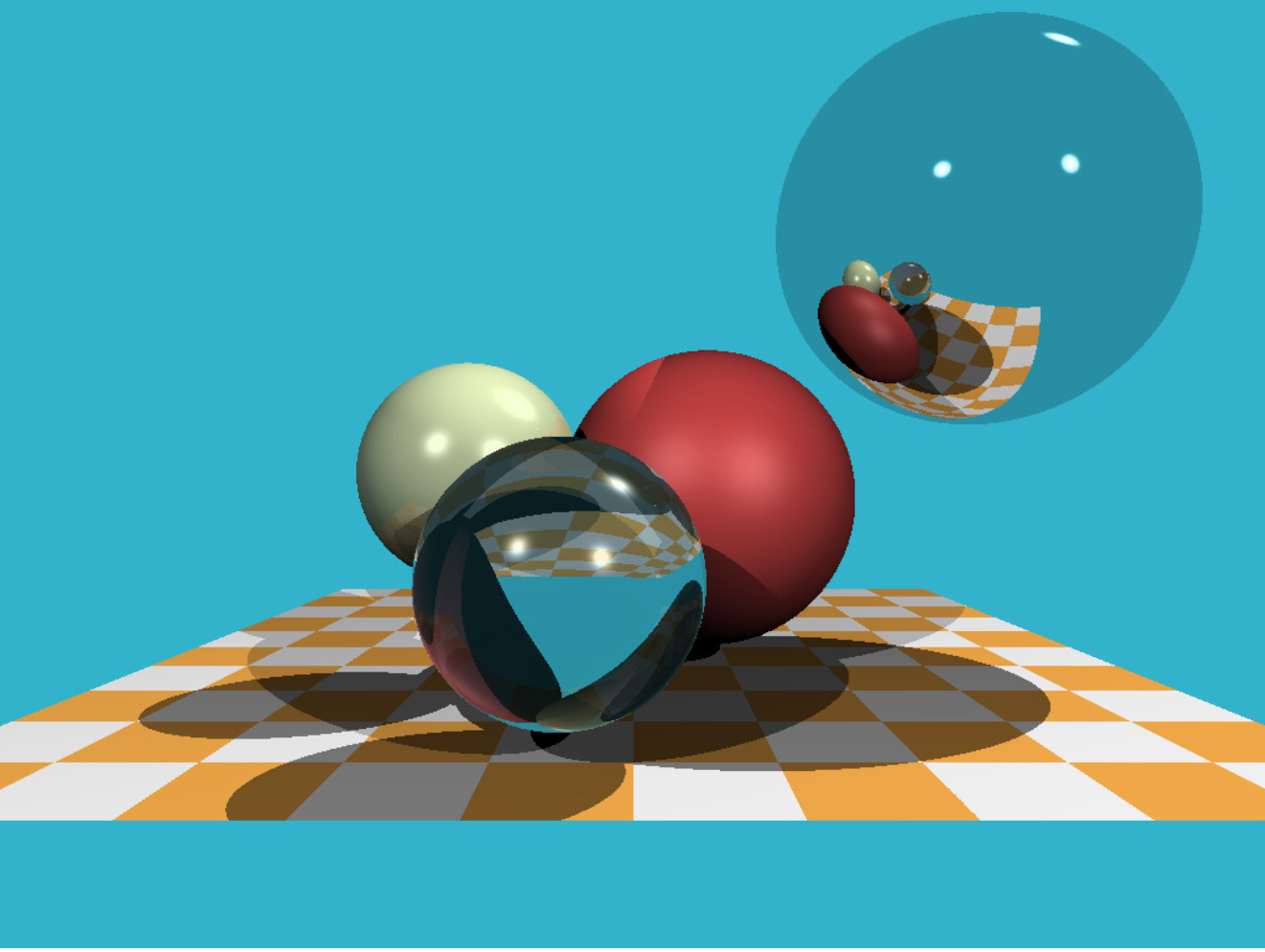


Light source를 distributed raytracing하기 위하여 여러 벡터를 선언한 함수이다. 원이 있다면, 각각의 벡터에 radius를 곱한 값을 더하여 light source를 분할해 주었다.



프로그램 실행 main 함수이다. 먼저, envmap을 load하여 envmap 변수에 저장해준다. 그르고 각각의 Material를 선언해준다. 기존의 코드에서 약간씩 변형을 하였고, 마지막 mint Material를 선언해주었다. 그리고 spheres도 기존의 코드와 조금씩 변형을 하였고, 마지막에 하나의 원을 더 추가해주었다. Light의 경우에는 기존의 코드와 동일한 위치지만, factor를 추가하므로 area light source로 바꾸어 주었다. 그리고 factor를 설정해주고, render을 진행하였다.

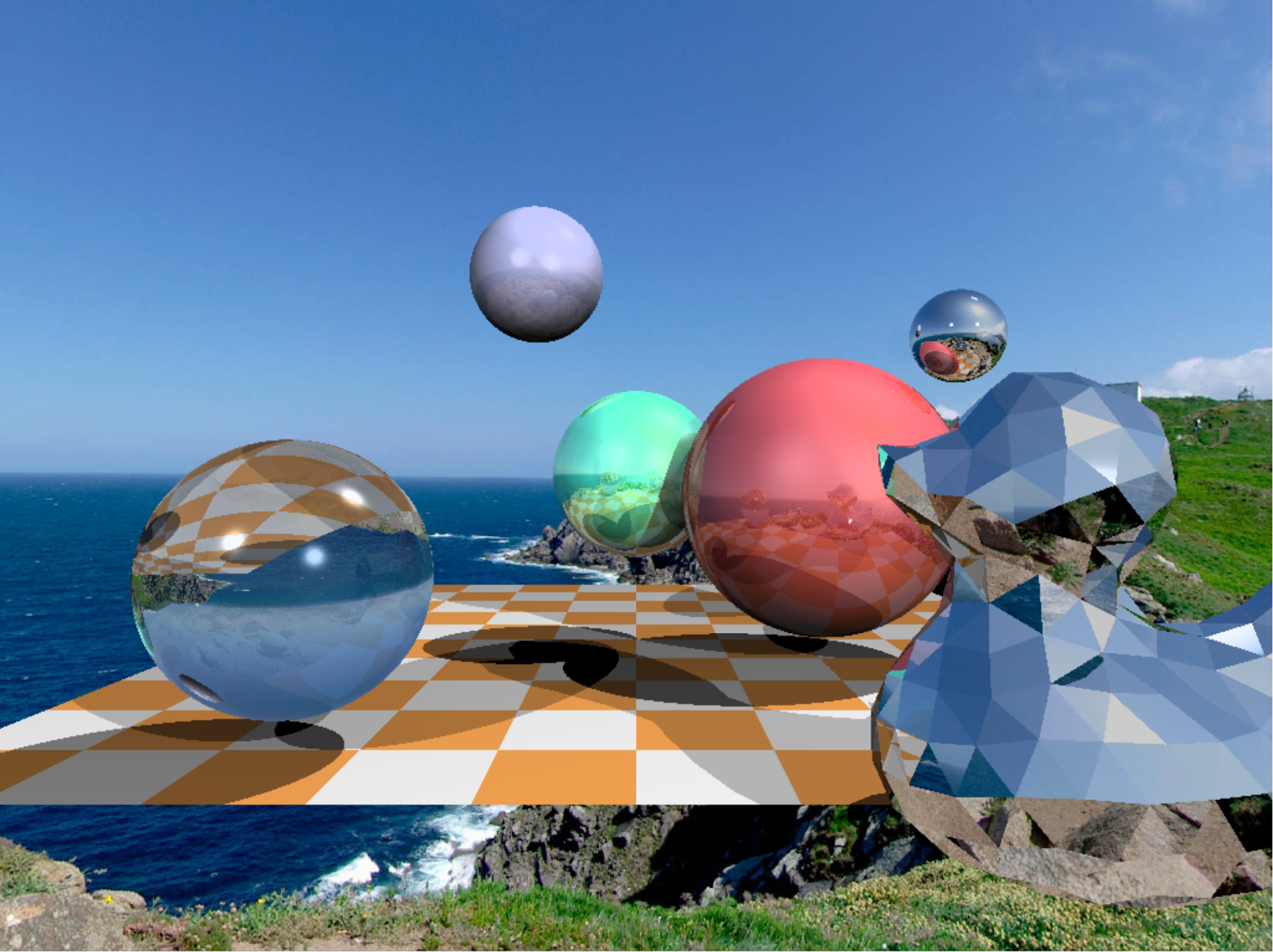
1. **결과**



위의 사진이 가장 기본이 되는 코드 실행 결과이다. Hard Shadow인 것을 확인할 수 있고, 각각의 원의 위치와 크기, 배경색으로 일정한 값으로 활용한 것을 알 수 있다.

1. out\_point.jpg

out\_point.jpg는 Light source를 area로 하지 않고 point로 했을 경우의 결과이다.



Model에서 duck.obj를 load하여 scene에 위치시켜 주었고, 모든 구의 크기와 위치, 재질이 바뀐 것을 확인할 수 있다. 또한 가운데에 민트색의 새로운 구가 형성된 것을 확인할 수 있다. 배경에 대하여 texture를 적용한 것을 확인할 수 있고, 각각의 구와 오리의 reflect에 해당 texture가 잘 적용되고 있는 것을 확인할 수 있다. 이때에는 point light source처럼 작용했기 때문에 shadow의 경계선이 확실한 것을 알 수 있다.

1. out\_area.jpg



이제 light soucre를 area로 바꾸어서 rendering한 결과이다. shadow를 위의 out\_point.jpg와 비교해보면 경계선이 모호해지면서 soft shadow가 된 것을 확인할 수 있다.

PHONG, TEXTURE



다른 각도의 PHONG, TEXTURE

**소스코드 위치: /src/main.cpp**

**작업 OS: Windows 10**

**작업 컴파일러: Visual Studio 2019**