RayTracing

Ray Tracing 을 직접 구현.

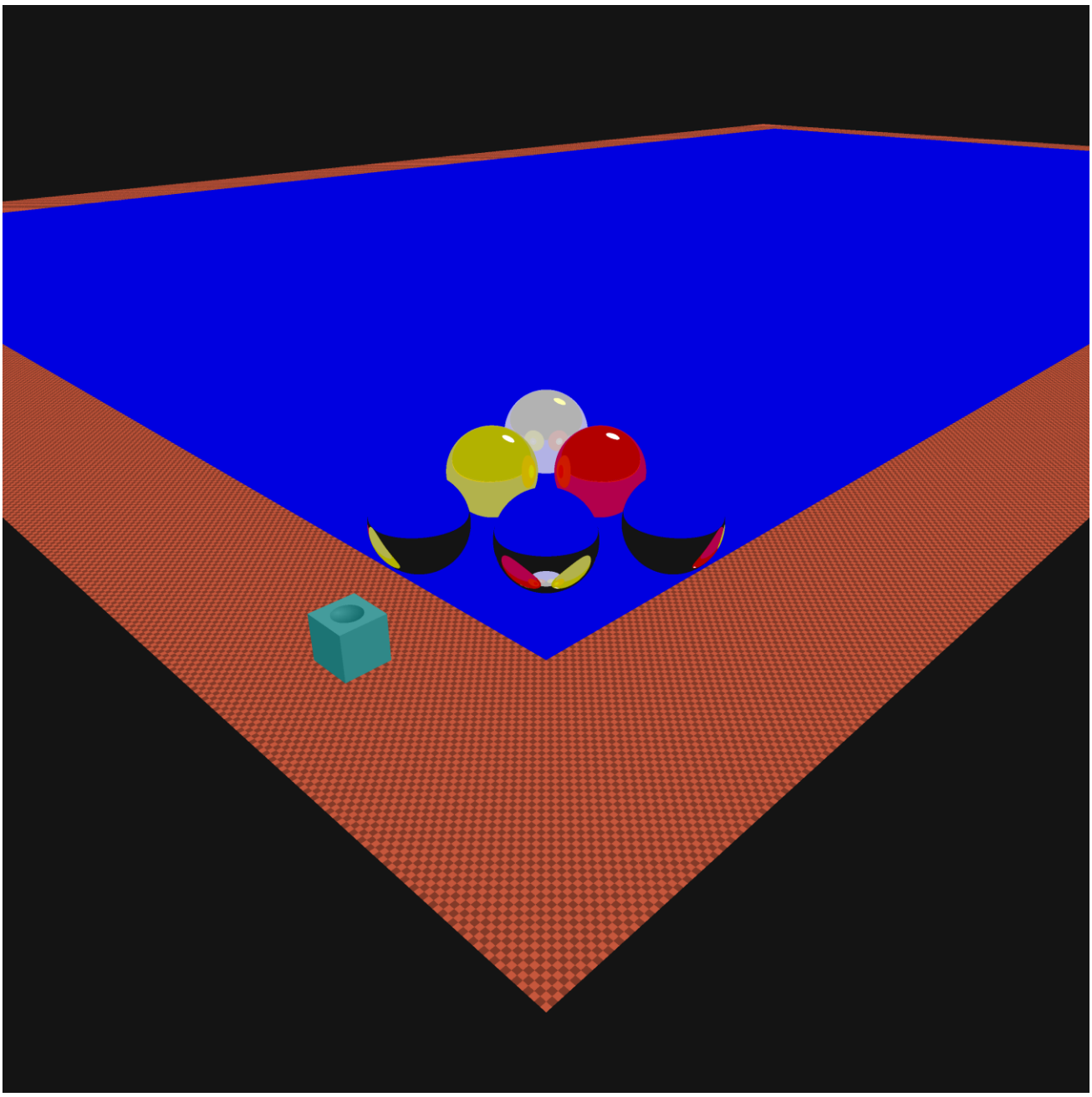
[실행 환경]
Ubuntu 18.04
OpenGL version string: 2.1 Mesa 20.0.8

sudo apt-get install g++
sudo apt-get install freeglut3-dev
sudo apt-get install mesa-utils
sudo apt install libglm-dev
sudo apt-get install libeigen3-dev

[실행 방법]
g++ -o ./RayTracing ./RayTracing.cpp
./RayTracing

실행 시 폴더 내에 Image.ppm 파일 생성.

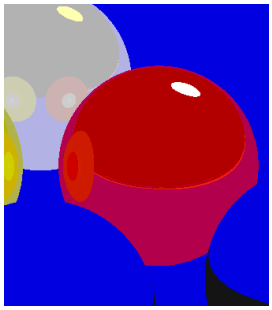
0. Representative Picture



Hw4에서의 Build My own scene의 당구장 테마를 이어나가 당구대 위의 당구공들을 표현하였다.

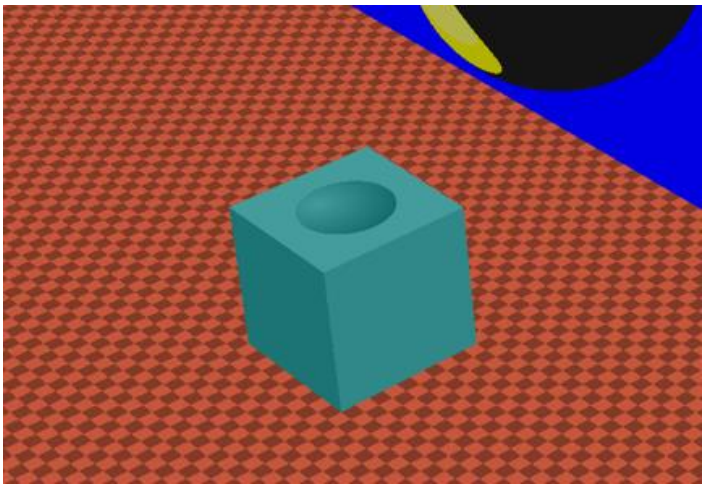
기존 camera의 위치에서 바라보는 원점과 viewing angle, clipping plane을 고려하여 관측 가능한 Ray 의 범위 내에서 width * height 개수만큼의 Ray를 보내 Tracing 하였다. 사용한 라이브러리로는 ROS에서 주로 쓰이는 Eigen 라이브러리를 사용하였다.

1. Ray Tracing Spheres



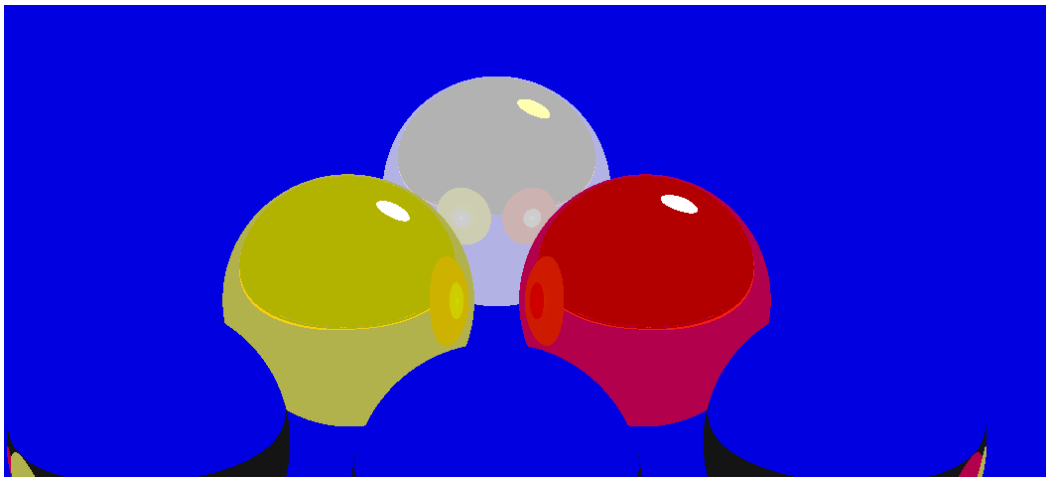
먼저 Sphere에 대한 Ray Tracing이다. Ray 라는 구조체를 생성하여 Ray와 구의 중심 사이 거리를 구하는 distanceFromPoint 함수, radius 를 주었을 때, 입사하는 지점을 구하는 contactPoint 함수, 최종 반사된 광선을 리턴하는 reflectedRay 함수를 이용하여 당구공의 ray tracing을 구현하였다.

2. Ray Tracing Polygons



당구대 파란색 다이와 당구대 테두리 나무 부분, 그리고 초크가 quad로 이루어진 Ray Tracing Polygon들이다. Diffuse를 적용하여 면 별로 밝기가 서로 다른 것을 알 수 있다.

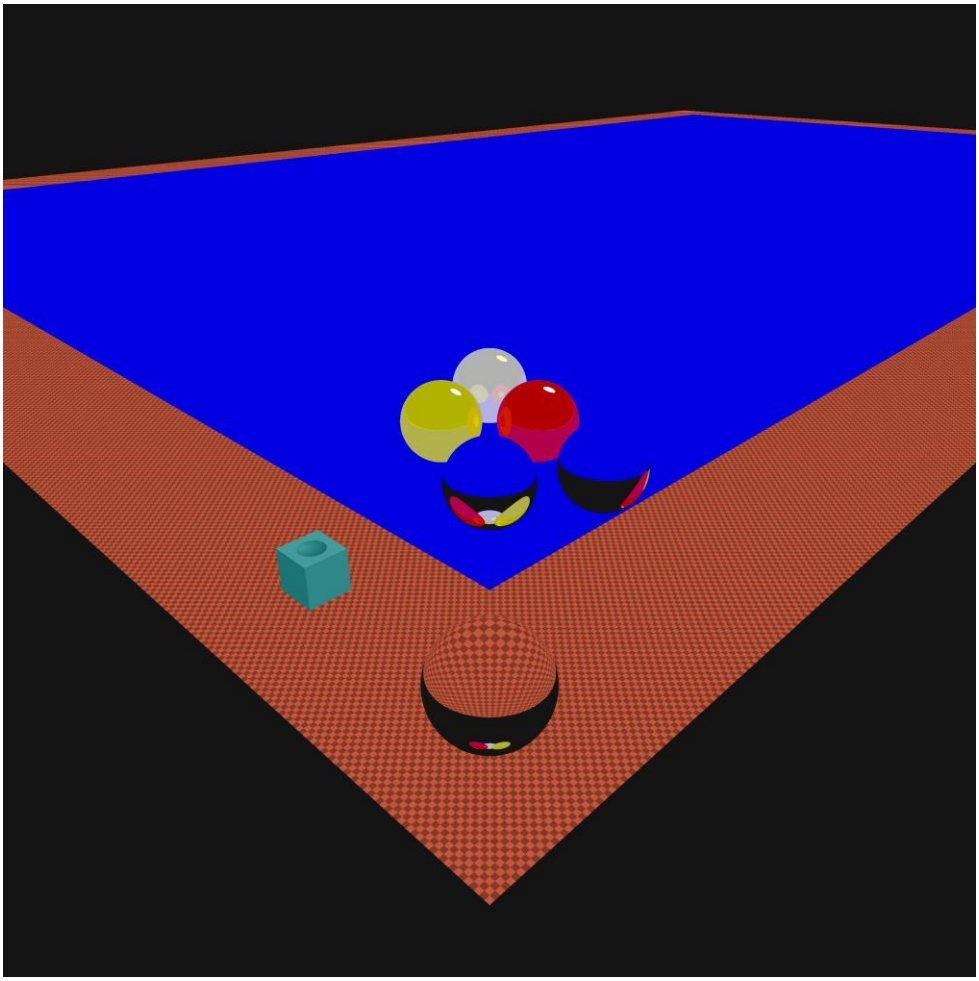
3. Recursive Reflection

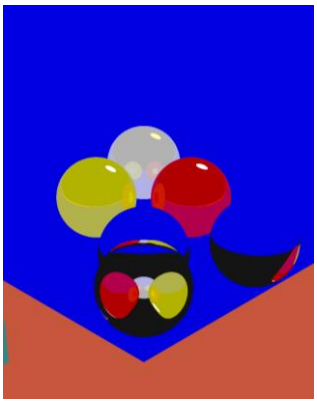


흰 당구공을 예시로 들면, camera -> white -> light / camera -> white -> yellow -> light / camera -> white -> yellow -> white -> light 모두 구현되어 있다.

4. Recursive Refraction

굴절률을 가진 구에 광선이 들어올 때, 두 번 굴절되어 나가는 광선을 리턴하는 sphereRefraction 함수를 구현하였다. 좌측 그림에서 당구대의 texture가 두 번 연속으로 굴절되어 무늬가 나타나고 있는 것을 볼 수 있다. 우측 그림은 굴절률을 가진 구 하나만으로도 두 번 굴절되지만, Recursive refraction을 더욱 잘 드러내기 위해 굴절률을 가진 투명 구를 빛의 경로가 겹치도록 수정 후 배치하여 실행한 결과이다.





```
291     Vector3f Ref1;
292     Ref1[0]=60.;
293     Ref1[1]=18.;
294     Ref1[2]=0.;
295     Vector3f Ref2;
296     Ref2[0]=35.;
297     Ref2[1]=35.;
298     Ref2[2]=0.;
299     Vector3f Ref3;
300     Ref3[0]=25.;
301     Ref3[1]=25.;
302     Ref3[2]=0.;
```

5. Phong illumination

모든 ray는 최종적으로 Ambient, Diffuse, Specular 각각 RGB 성분 하나씩 총 9개의 성분을 가진다. 빛이 도달하는 마지막 표면의 성질에 따라 갖는 값이 달라진다. Ambient의 경우 처음 도달한 표면에 대해서만 결정되며, Diffuse의 경우 광원이 현재 위쪽에 위치하며, 반짝거리는 당구공 표면의 경우 아랫쪽 부분은 diffuse가 어둡게, 또 반사되어 직접적으로 광원으로 들어가는 부분은 specular가 훨씬 밝게 처리하였다. 전부 계산이 끝난 후 RGB 각각 세 성분을 모두 더하여 해당 pixel의 RGB 값을 결정하였다.

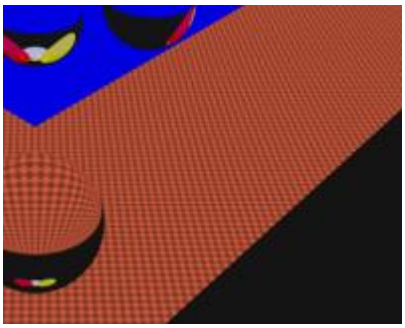
6. Export image files

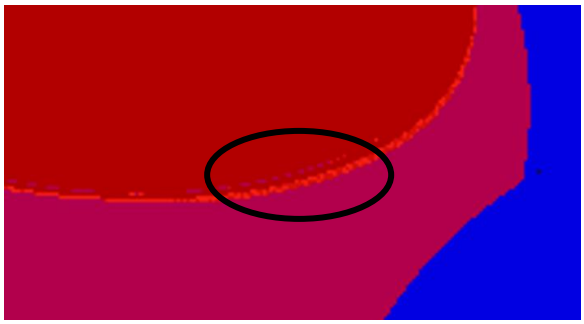
파일을 실행하면 같은 폴더에 Image.ppm 파일이 생성된다. 리눅스(우분투 18.04 기준) 에선 바로 ppm 확장자가 열리지만, 윈도우의 경우 변환을 통하여 열 수 있다.

7. Texture mapped spheres and polygons

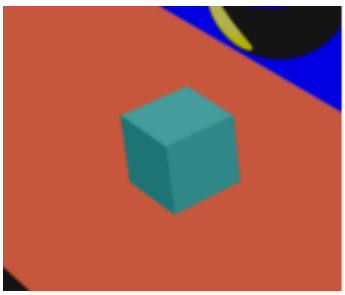
Hw4에서 넣어주었던 재질 별 ambient, diffuse, specular에 대한 texture 값을 최대한 반영하여 넣어주었다. 일례로 chalk와 당구대 파란 다이의 경우 specular은 0이며, 당구공의 경우 specular는 아주 높다.

당구대의 나무 부분은 왼쪽 그림과 같이 체크 무늬의 texture를 넣어주었다. 나무 부분과 ray의 교점 좌표를 구하여 조건을 주어 무늬를 넣어 주었다. 당구공에 반사된 나무 부분에도 무늬를 넣어 보았지만 우측 그림과 같이 해상도의 문제로 잘 보이지는 않는다.

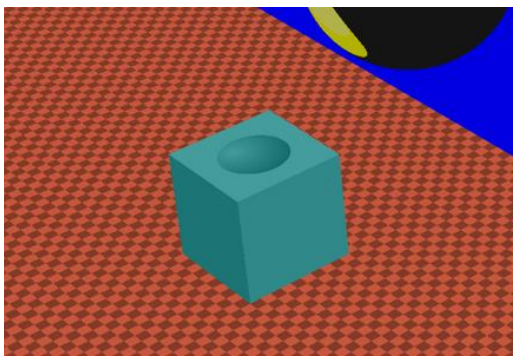




당구대 위에 올려진 초크는 실제로는 그냥 정육면체 모양이지만, 강의시간에 배운 **Bump Mapping** 알고리즘을 넣어 실제 초크처럼 안으로 볼록하게 보일 수 있도록 texture를 입혔다. 그 결과로, 나무 부분과 초크에 아무런 texture를 주지 않은 왼쪽 그림에 비하여 훨씬 실제와 유사한 효과를 얻을 수 있었다.



->



구현 시 벡터 계산은 ROS 등에서 자주 사용되는 Eigen library를 사용하였다.

이 외에도 세 당구공 및 세 굴절되는 공을 움직이려면 아래의 값을 바꿔주면 된다.

```
273     Vector3f yellow;
274     yellow[0]=50.;
275     yellow[1]=70.;
276     yellow[2]=0.;
277     Vector3f red;
278     red[0]=70.;
279     red[1]=50.;
280     red[2]=0.;
281     Vector3f white;
282     white[0]=80.;
283     white[1]=80.;
284     white[2]=0.;
```

```
285     Vector3f Ref1;
286     Ref1[0]=60.;
287     Ref1[1]=18.;
288     Ref1[2]=0.;
289     Vector3f Ref2;
290     Ref2[0]=18.;
291     Ref2[1]=60.;
292     Ref2[2]=0.;
293     Vector3f Ref3;
294     Ref3[0]=33.;
295     Ref3[1]=33.;
296     Ref3[2]=0.;
```