심화전공실습 (CGL)

HW11_Ray Tracing

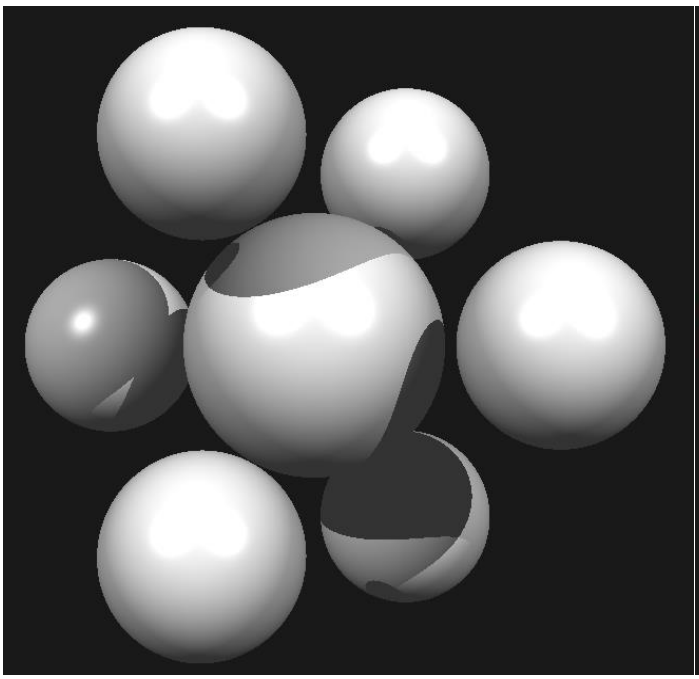
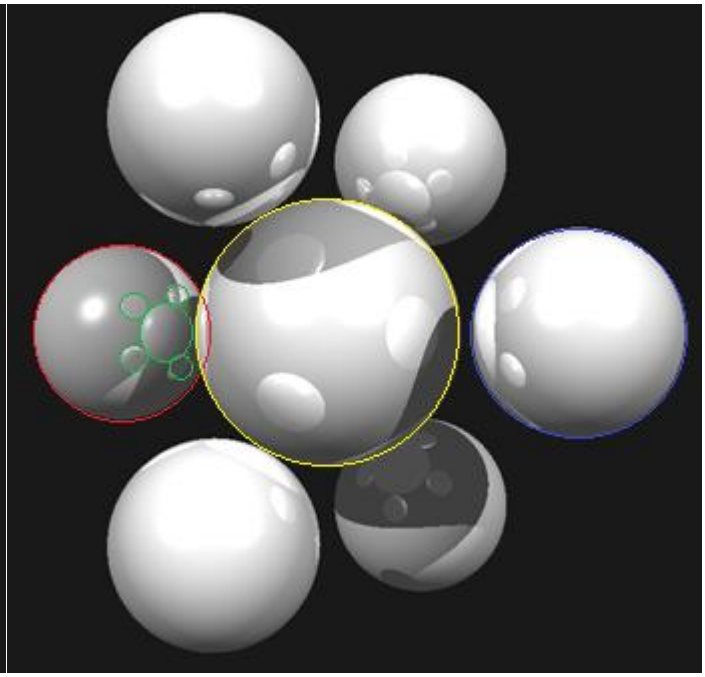
Self-scoring table

	P1	P2	E1	E2	Total
Score	**1**	**1**	**1**	**1**	**4**

2018707068 김경환

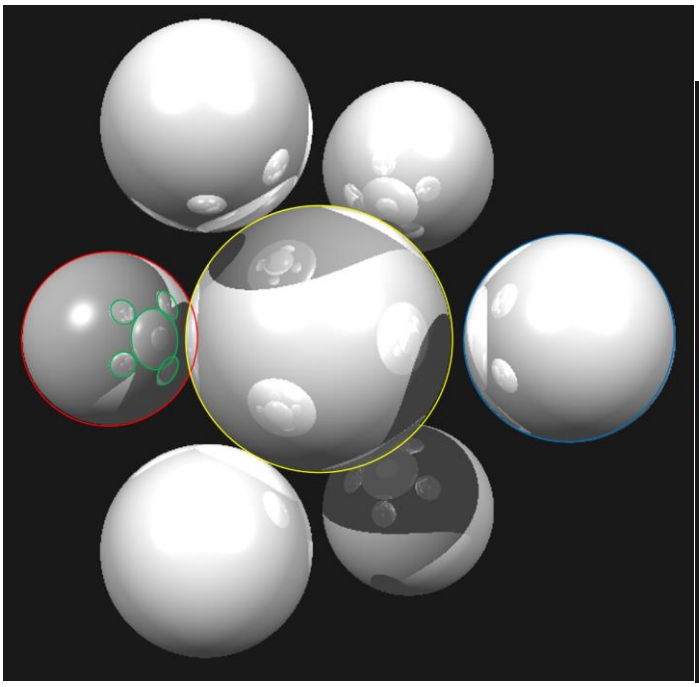
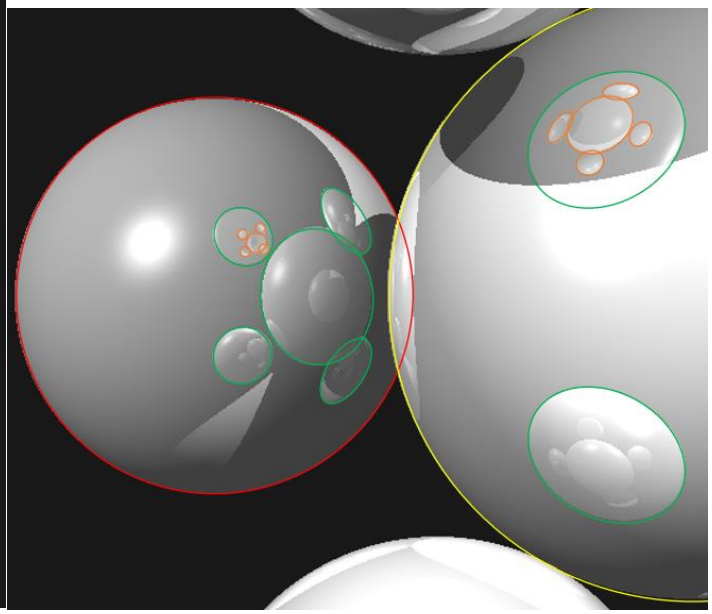
KwangWoon University

Practice01 Snapshot, Explanation:

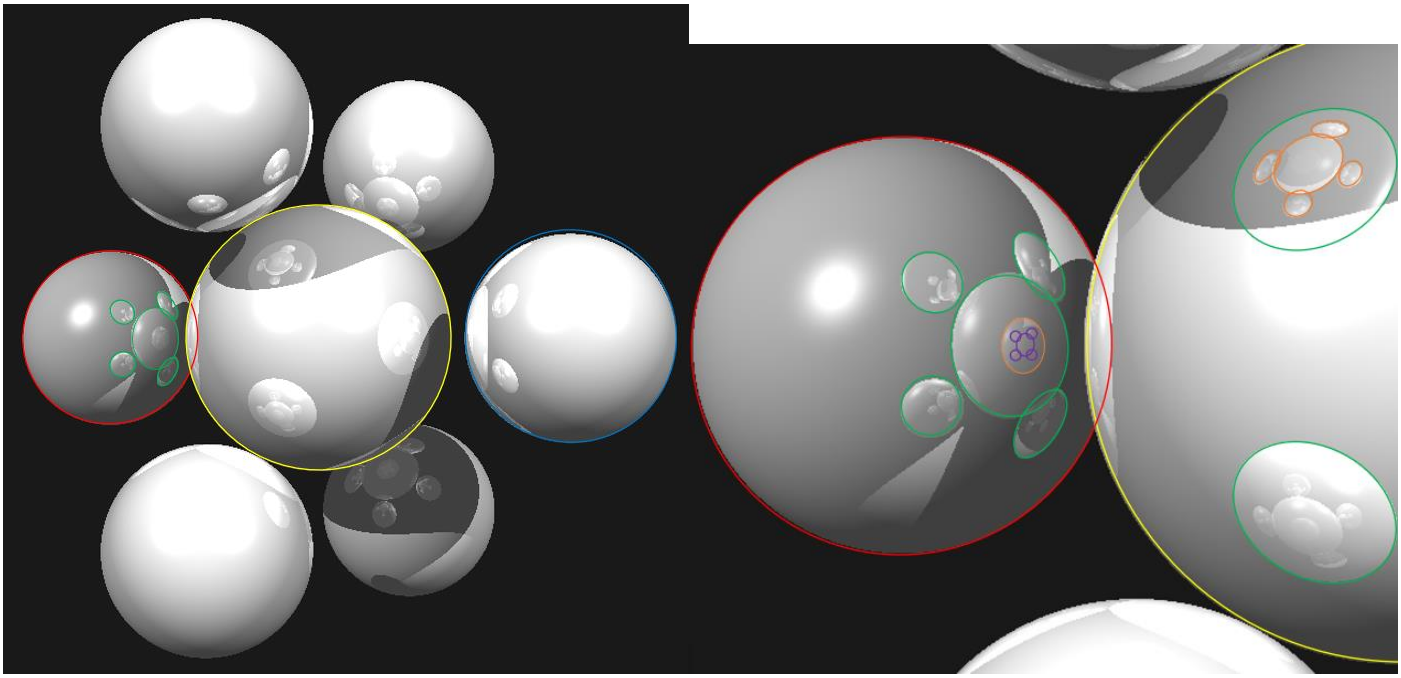
우선 왼쪽 사진은 depth가 1일 때의 모습이다. 이때는 1인 depth에 의해 recursive하지 않고 한 번의 ray casting만을 하기 때문에 구들끼리의 반사에 의한 상은 구에 보이지 않는다. 하지만 shadow ray를 통한 shadowed를 판단하기 때문에 구들끼리의 그림자는 구현되는 것을 볼 수 있다.

오른쪽 사진은 depth가 2일 때의 모습이다. 이제는 2인 depth에 의해 reflection ray에 대해서도 ray casting을 하기 때문에 자기 자신(빨간색)과 가운데 구(노란색)에 가려진 반대편의 구(파란색)를 제외한 나머지 5개의 구(초록색)가 reflect되어 구에 상이 맺히는 것을 볼 수 있다. 또한, depth가 1일 때의 과정도 똑같이 거치기 때문에 depth가 1일 때의 모습도 동일하게 그려져 있다.

위 왼쪽 사진은 depth가 3일 때의 모습이다. 위에서 설명한 것과 같이 recursive하게 ray casting을 하기 때문에 depth가 2일 때 맺힌 5개의 구(초록) 안에 다시 5개의 구(주황)가 맺히는 것을 볼 수 있다.

오른쪽 사진은 왼쪽 사진을 확대한 모습이다. (왼쪽 사진에는 주황색 구가 너무 작아 표시하지 않았다.)



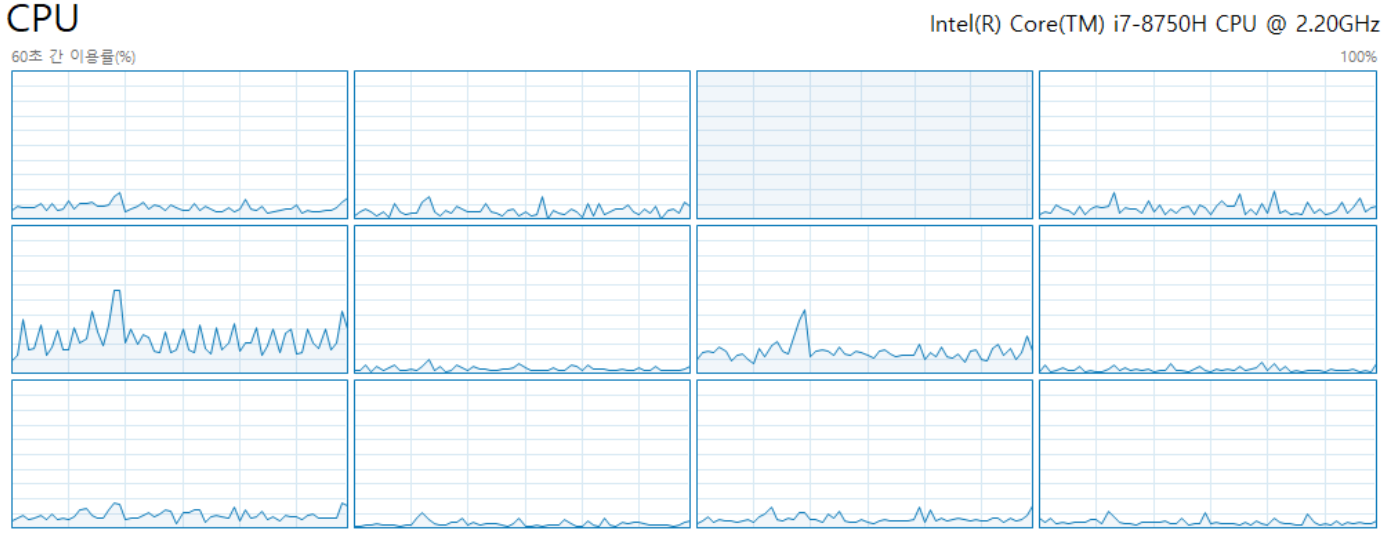
위 왼쪽 사진은 depth가 4일 때의 모습이다. 위에서 설명한 것과 같이 recursive하게 ray casting을 하기 때문에 depth가 3일 때 맺힌 5개의 구(주황) 안에 다시 5개의 구(보라)가 맺히는 것을 볼 수 있다. 오른쪽 사진은 왼쪽 사진을 확대한 모습이다. (왼쪽 사진에는 보라색 구가 너무 작아 표시하지 않았다.)

Practice02 Snapshot, Explanation:

```
Status: Monitor 597mm x 336mm
Status: Screen 1280 x 720
Status: Framebuffer 1280 x 720
Status: Renderer NVIDIA GeForce RTX 2070 with Max-Q Design/PCIe/SSE2
Status: Ventor NVIDIA Corporation
Status: OpenGL 4.6.0 NVIDIA 512.89
reshape(1280, 720) with screen 1280 x 720
# threads = 12

Keyboard input: space for play/pause
Keyboard input: up for increasing specular
Keyboard input: down for decreasing specular
Keyboard input: t for texture mapping/direct drawing
Keyboard input: p for parallel computing on/off
Keyboard input: [1:9] for ray tracing path
Image size: 1280 x 720
Non-parallel computing
Direct Drawing
elapsed = 0.124258
elapsed = 0.124378
elapsed = 0.128642
elapsed = 0.129429
elapsed = 0.136309
```

실행환경은 12개의 thread를 가진 cpu이다.

OpenGL를 사용하지 않고 Non-parallel Computing을 하였다.
수행시간은 대략 0.12초로 측정된다.
아래는 각 thread의 CPU 이용률이다.



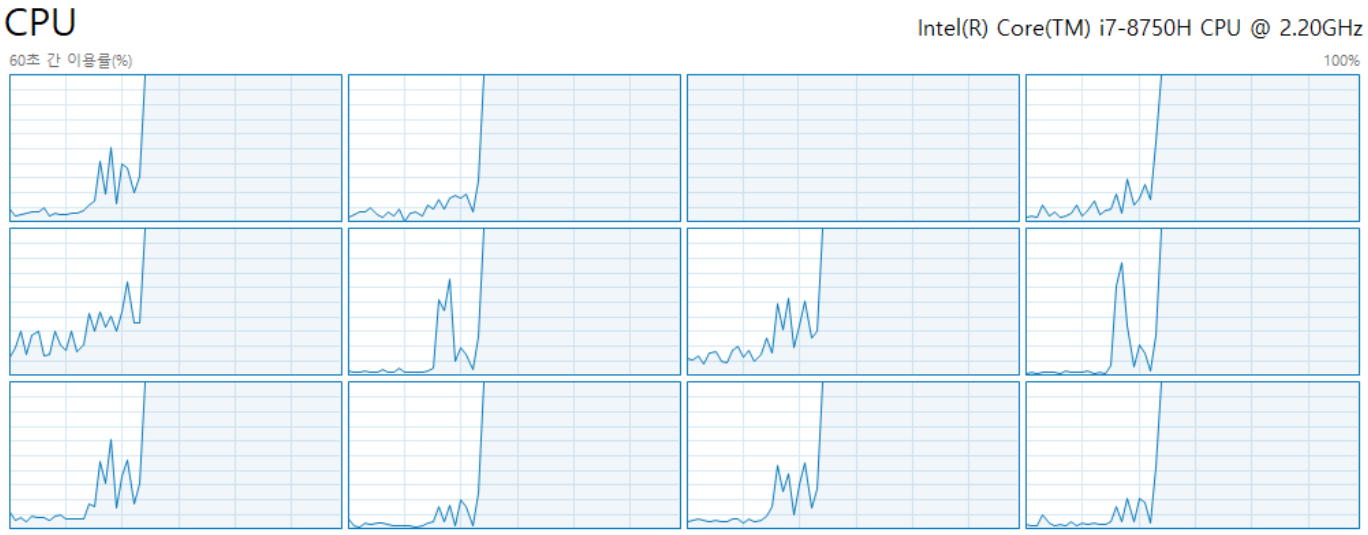
```
Status: Monitor 597mm x 336mm
Status: Screen 1280 x 720
Status: Framebuffer 1280 x 720
Status: Renderer NVIDIA GeForce RTX 2070 with Max-Q Design/PCIe/SSE2
Status: Ventor NVIDIA Corporation
Status: OpenGL 4.6.0 NVIDIA 512.89
reshape(1280, 720) with screen 1280 x 720
# threads = 12

Keyboard input: space for play/pause
Keyboard input: up for increasing specular
Keyboard input: down for decreasing specular
Keyboard input: t for texture mapping/direct drawing
Keyboard input: p for parallel computing on/off
Keyboard input: [1:9] for ray tracing path
Image size: 1280 x 720
Non-parallel computing
Parallel computing
Direct Drawing
elapsed = 0.0247869
elapsed = 0.0282469
elapsed = 0.0271864
elapsed = 0.0264468
elapsed = 0.0297289
```

실행환경은 12개의 thread를 가진 cpu이다.

OpenGL를 사용하여 parallel Computing을 하였다.

수행시간은 대략 0.026초로 측정된다.

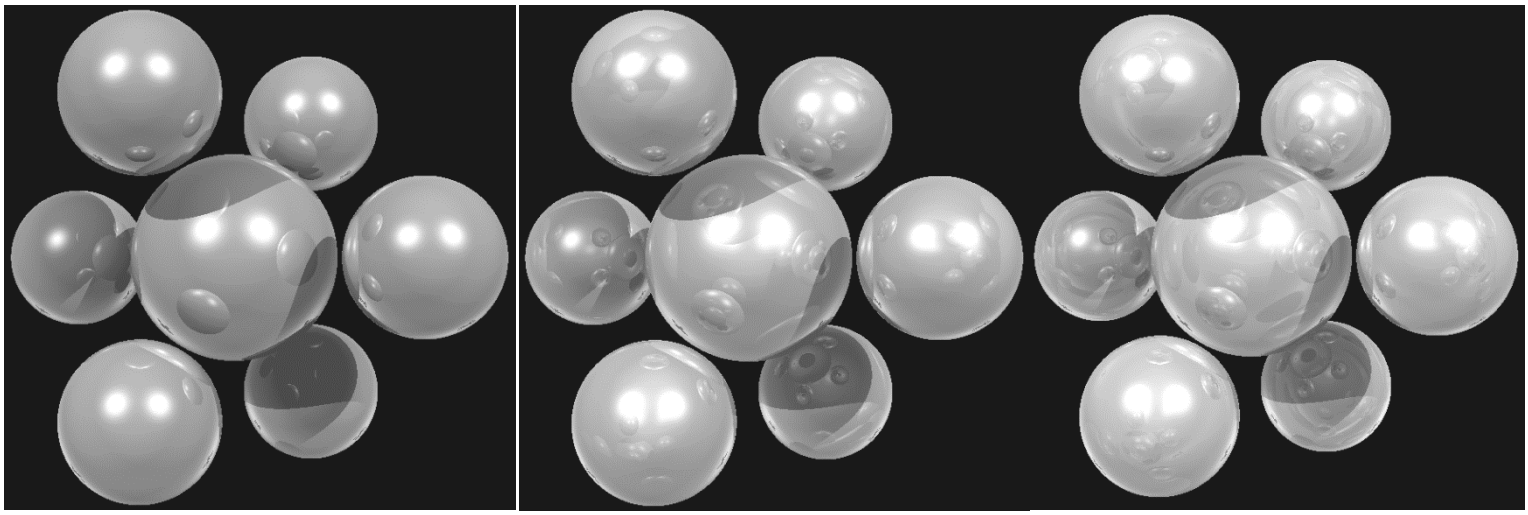
아래는 각 thread의 CPU 이용률이다.



위 사진들을 보면 Non-parallel computing을 할 때는 12개의 thread 중에 하나의 thread만을 사용하고 parallel computing을 할 때는 12개의 thread를 모두 사용하는 것을 볼 수 있다.

직관적으로 1개의 thread에서 12개의 thread로 parallel computing을 하게 되면 수행시간이 12배 빨라질 것 같지만 그렇지는 않고 대략 4.6배 정도 빨라지는 것을 확인할 수 있다.

Exercise01 SnapShot, Explanation:



Pracitce1에서 수행한 7개를 구와 camera를 전부 감싸는 큰 구를 만들어 ray tracing을 수행한 모습이다.

왼쪽부터 depth가 2, 가운데는 3, 오른쪽은 4이다.

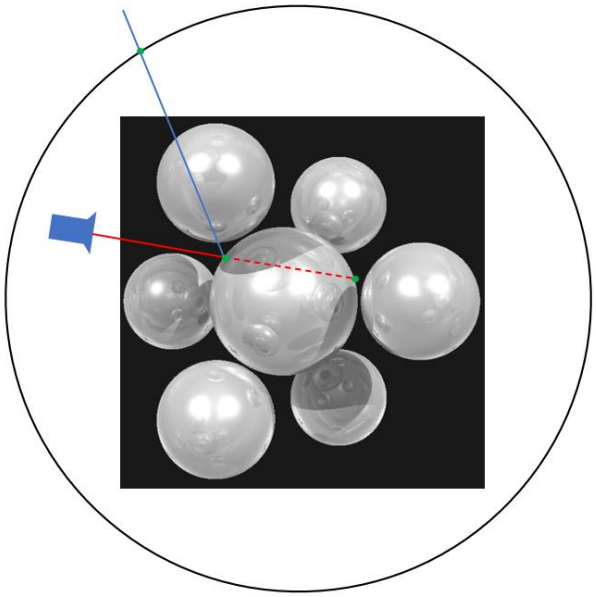
기존 Practice1과 다른 점으로는 감싸고 있는 구에 의해 더 많은 반사가 이루어지므로 7개의 구에 맺히는 반사에 의한 상이 더욱 입체적으로 보인다는 것이다. 또한, 앞에서 말한 더 많은 반사로 인해 더 밝고 더 매끄러운 것처럼 보이는 효과가 있다.

이를 구현한 방법으로는 원의 중심이 원점이고 camera와 7개의 구를 전부 감쌀 수 있을만큼 큰 반지름으로 20을 갖는 원을 하나 scene에 추가하였다. (center = (0, 0, 0), radius = 20.0f)

그리고 intensity를 계산하는 부분에서 shadow ray가 가장 바깥쪽의 감싸는 구에 빛이 가려지는 것으로 판단하지 않고 빛이 전달되는 것으로 판단되어 phong reflection이 계산될 수 있도록 한다.

(Not shadowed를 판단하는 조건에 가장 큰 구와의 intersection을 추가)

위에서는 가장 큰 구의 밖에서 들어오는 빛을 가리지 않기 위한 작업이었다. 이제는 가장 큰 구 안에서 반사되는 빛을 계산하기 위해서 ray 시작과 끝 사이에서 구와 몇 개의 intersection이 일어나는가를 확인한다. 이는 구의 바깥쪽과 안쪽의 반사를 구분짓는 것이 intersetion의 개수이기 때문이다.



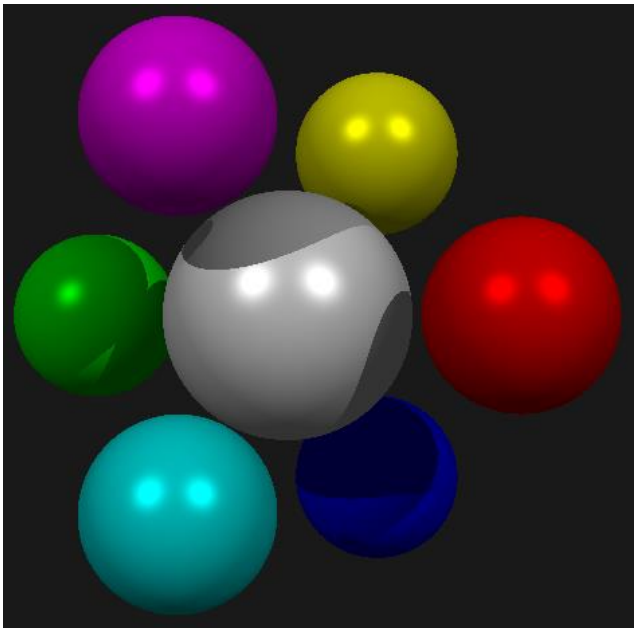
왼쪽 그림에서 빨간색 ray의 경우에는 가운데 구와 두 개의 교점이 생기는 것을 볼 수 있다. (교점 : 초록색 점)

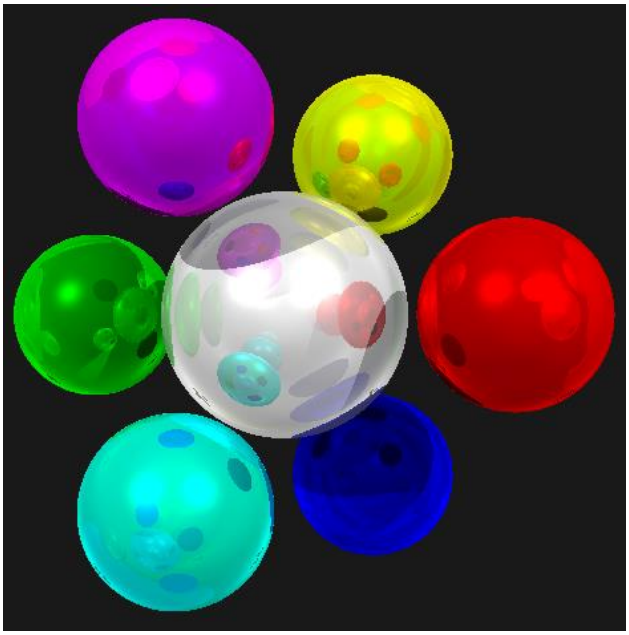
파란색 ray의 경우에는 빨간색 ray의 reflect ray로써 ray casting의 recursive에서 실행된다. 또한, 가장 바깥쪽의 구와 한 개의 교점을 갖는다. (교점 : 초록색 점)

이렇듯이 구의 안쪽에서 시작한 ray의 경우에는 교점이 하나이고, 바깥쪽에서 시작한 ray의 경우에는 교점이 두 개다. 여기서 교점이란 구와 ray의 intersecion에 의해 계산된 2차방정식의 해이다.

그리하여 위와 같은 내용을 구현하기 위해 findintersection 부분에서 2차방정식의 해가 2개일 경우에는 normal vector를 p – center로 지정하고, 1개일 경우에는 center – p로 지정한다.

Exercise02 SnapShot, Explanation:





위 사진은 7개의 구에 각각 다른 색을 적용한 모습이다. (왼쪽 : depth = 1, 오른쪽 : depth = 4)
위와 같은 내용을 구현하기 위해서는 intensity 함수에서 반환되는 빛의 RGB intensity값을 각 구마다 다르게 주면 된다. 다르게 주는 방법으로는 각 구와 ray가 intersection이 일어나는 point에서 intensity가 계산되므로 해당 point가 위치한 구의 index를 통해 switch case문으로 다음과 같은 RGB intensity를 부여해 구현하였다. [(R=0), (G =0), (B=0), (R=0, G=0). (G=0. B=0). (R=0, G=0, B=0) (default)로 총 7가지]