HW5. Ray Tracing

2015-12986 박창휘

1. 구동 방법

Ray tracer 디렉토리에서 다음 명령어를 차례로 입력하면 코드를 컴파일할 수 있다. 멀티스레딩을 위해 OpenMP가 적용되어 있으므로 컴파일러가 OpenMP를 지원해야 한다.

```
> make
```

다음 명령어를 입력하면 Scene0 혹은 Scene1을 Ray tracing 할 수 있다.

```
> make run_zero
> make run_one
```

Scene0은 HW4에서 만든 씬을 그대로 옮긴 것이며, Scene1은 Specification 가운데 Scene0에서 잘 드러나지 않는 부분을 명확히 보충하여 보여주기 위해 따로 만들어진 씬이다. 위 커맨드를 입력하면 30분 이상의 시간이 소요된 후 해당 디렉토리에 결과가 Result.bmp 파일로 출력된다. HW5-2015-12986 디렉토리에는 Scene0와 Scene1의 Ray tracing 결과, 그리고 Scene0을 OpenGL에서 재현한 Comparison.jpeg가 있다.

2. 구현 사항(전반적인 구현 방법)

크게 Scene, Shape, Light 세 가지 클래스로 나누어 모듈화 하였다. Shape는 Face와 Sphere로 나누어진다. OBJImporter를 통해 Import된 오브젝트들은 여러 개의 Face들로 나누어지지만, Sphere는 하나마다 하나의 객체가 된다. 따라서 Sphere는 렌더링에 있어 Imported object들에 비해 훨씬 부담이 적다. Shape들은 각각 Transform, Diffuse, Specular, Emission 등의 Member들을 갖고 있고, 이러한 Scene들을 Scene에 Insert함으로써 씬을 구성할 수 있다. Light도 마찬가지로 위치, 방향, Diffuse, Attenuation, Infinitely distant 여부 등의 Member들을 갖고 있으며, Scene에 Insert함으로써 씬에 배치할 수 있도록 Abstract하게 모듈들을 구성하였다.

Ray tracing을 수행하는 과정은 다음과 같다.

1) 가상의 COP를 (0, 0, FOCAL_LENGTH)에 배치한다.
2) Image plane이 x-y plane에 존재한다고 상정하고 각 픽셀로 Primary ray를 투사한다.
3) 모든 Shape에 대해 충돌을 검사하여 가장 가까운 Shape를 알아낸다.
4) 충돌 지점에서 Shadow ray에 따라 Phong illumination의 Contribution을 구한다.

5) Specular와 Alpha에 따라 Reflection ray와 Refraction ray를 투사한다.

6) 3, 4, 5번 과정을 Recursive하게 수행하여 픽셀마다 RGB 값을 구한다.

3. 구현 사항(Scene0)

1) Ray tracing spheres

LED, 금, 사파이어, 은으로 이루어진 Sphere들을 씬에 배치하여 Ray tracing 하였다. Sphere는 Polygon으로 이루어져 있지 않고 위치와 반지름, 즉 구의 방정식을 이루는 정보를 갖고 있다. Ray와 Sphere의 교점은 강의 교안에 나와있는 이차방정식의 근을 구해, 그 가운데 가까운 점으로 구한다. 따라서 수많은 Polygon으로 Sphere를 구하는 것보다 훨씬 빠르게 Ray tracing이 이루어진다.

2) Ray tracing polygons

유리 패널, ThinkPad 글자, 트랙포인트, 그리고 바닥과 배경을 씬에 배치하여 Ray tracing하였다. 각 Triangle을 하나의 객체로 취급하여 충돌 검사를 수행하므로 ThinkPad 글자와 같이 Polygon 수가 많은 오브젝트들은 계산적인 부담이 크다.

3) Recursive reflection

금으로 이루어진 Sphere에 바닥의 텍스처가 반사되고 있는 점, 사파이어 Sphere에 금에 비춰진 빛이 다시 한 번 반사되고 있는 점을 통해 Recursive reflection을 확인할 수 있다.

4) Recursive refraction

유리 패널을 통해 비춰진 배경 텍스처에 왜곡이 일어난 점, 사파이어 Sphere를 통해 바닥 텍스처가 왜곡되어 보이는 점을 통해 Recursive refraction을 확인할 수 있다. Refraction ray를 투사할 때에는 Snell의 법칙이 적용되어 각 Shape마다 고유의 Refraction parameter를 지정할 수 있도록 구현하였으며. 또한 전반사가 일어날 경우와 Ray가 오브젝트의 내부에서 외부로 뚫고 나가는 경우의 굴절도 고려하였다.

5) Phong illumination

Sphere들과 트랙포인트, ThinkPad 글자에 음영이 표현된 것을 통해 Phong illumination을 확인할 수 있다. Phong illumination은 각 Triangle의 Vertex normal들을 Interpolation한 후 Light vector, Normal vector, View vector와 물체 고유의 파라미터들을 조합하여 모든 Light의 Contribution을 계산함으로써 해당 표면의 밝기 값을 계산하도록 구현되었다.

6) Export image files

가장 간단한 포맷인 BMP 파일을 Export하기 위해 [1]의 답변들 가운데 ProjectPhysX의 코드를 참조하여 이미지 입출력을 구성하였다. Ray tracing 결과가 Results.bmp 파일로 출력되는 점을 통해 확인할 수 있다.

7) Texture mapped spheres and polygons

바닥과 배경 평면에 탄소섬유 무늬 텍스처가 적용되었다. 각 Face마다 텍스처와 UV coordinate를 지정하면 Barycentric coordinate를 사용해 텍스처로부터 픽셀 값을 읽어오고, 이 값이 Phong illumination을 계산할 때 Diffuse term으로 반영된다. 금 Sphere에 나타나는 무늬 역시 확인할 수 있다. Sphere의 경우 따로 UV coordinate를 지정할 필요 없이 Procedural하게 UV Coordinate를 구하도록 구현하였다.

8) Distributed ray tracing

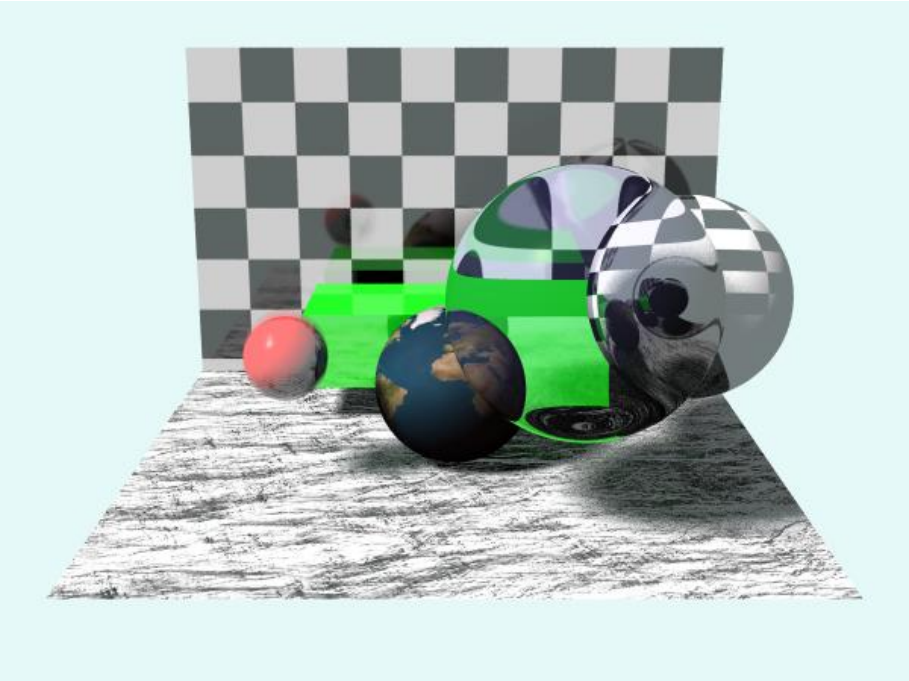
Area light와 Distributed shadow ray를 구현함으로써 부드러운 그림자를 표현하였다. 또 Distributed specular reflection을 구현하여 금 Sphere에 바닥의 텍스처가 약간 흐릿하게 반사되도록 하여 중간 정도의 Glossiness를 갖는 열처리가 가해진 금 재질을 표현하였다. Distributed

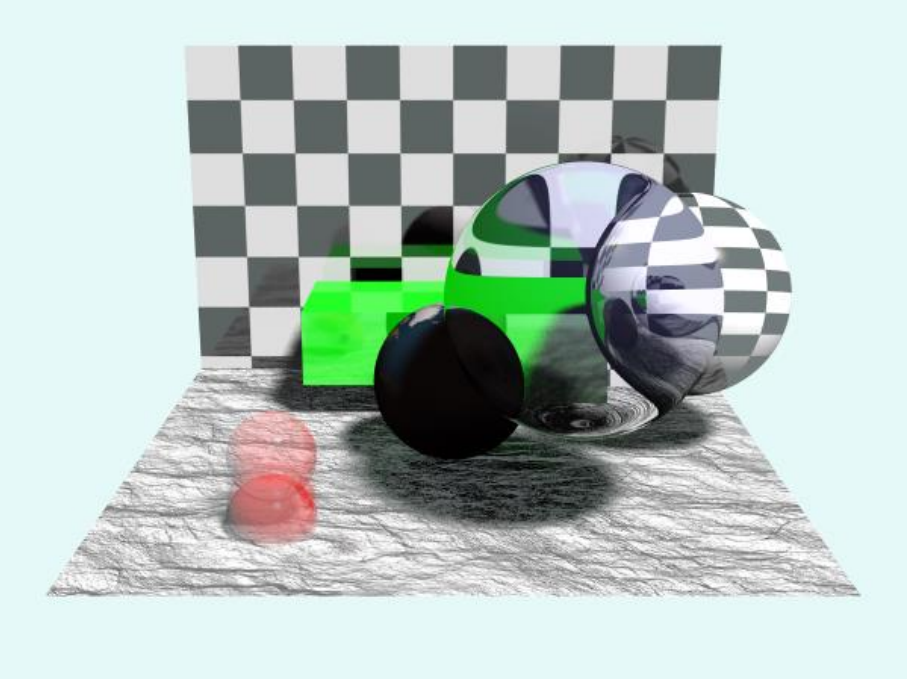
ray tracing을 위해 각 픽셀마다 20개의 Primary ray를 투사하는 Supersampling을 수행한 후, 얻어진 이미지들의 평균을 취했다. 또 Shadow ray와 reflection에서 무작위의 Ray 방향을 설정하기 위해, Light나 Surface의 파라미터에 따라 벡터를 무작위로 Perturbation하도록 구현하였다.

9) Bump mapping

Bump mapping의 일종인 Normal mapping을 구현하였다. Normal mapping은 이미지의 R, G, B 값에 Normal의 X, Y, Z를 저장하는 방법이다. 텍스처의 UV space에 저장된 R, G, B를 Surface space의 Normal로 변환하여, Phong illumination 계산과 Reflection 및 Refraction 계산에 사용할 Normal을 Normal map으로부터 얻는 코드를 구현하였다. 바닥과 배경에 CarbonNormal.bmp 텍스처가 적용되어 울퉁불퉁한 표면을 확인할 수 있다.

4. 구현 사항(Scene1)





1) Ray tracing spheres

다양한 Diffuse, Specular, Alpha로 이루어진 Sphere들을 씬에 배치하여 Ray tracing 하였다.

2) Ray tracing polygons

초록색 상자, 씬에 배치하여 Ray tracing하였다. 각 Triangle을 하나의 객체로 취급하여 충돌 검사를 수행하므로 ThinkPad 글자와 같이 Polygon 수가 많은 오브젝트들은 계산적인 부담이 크다.

3) Recursive reflection

빨간색 Sphere에 지구본과 바닥이 반사되고 있는 점, 유리 구슬 너머의 은색 Sphere에 지구본을 비롯해 여러가지 오브젝트들이 반사되고 있는 점을 통해 Recursive reflection을 파악할 수 있다.

4) Recursive refraction

유리 구슬을 통해 보이는 씬을 통해 왜곡을 확인할 수 있다.

5) Phong illumination

Scene0와 동일하게 오브젝트들의 음영을 통해 확인할 수 있다.

6) Export image files

Scene0와 동일하게 Result.bmp로 결과가 출력된다.

7) Texture mapped spheres and polygons

지도 텍스처를 적용한 지구본과 체크무늬 Sphere를 배치하였다.

8) Distributed ray tracing

Scene0와 동일하게 부드러운 그림자, 중간 Glossiness가 구현되었다. 다만 여기서는 Temporal하게 Ray를 Distribute함으로써 빨간 공이 떨어지면서 보여주는 Motion blur를 구현하였다. 미리 움직임을 저장해 놓고, 각 샘플마다 빨간 공을 움직인 후 Ray tracing을 수행함으로써 해당 효과를 구현하였다. Motion blur가 뚜렷하게 보이도록 160개의 샘플을 사용하였다.

9) Bump mapping

Bump mapping의 일종인 Normal mapping을 구현하였다. RockNormal.bmp 텍스처가 적용되어 바위처럼 울퉁불퉁한 표면을 연출하고 있는 점을 통해 확인할 수 있다. 빛의 위치를 달리하여 얻은 두 장면에서 바닥의 음영이 달라졌음을 알 수 있고, 이로부터 일반적인 텍스처와의 차이를 확인할 수 있다.

5. 구현 사항 정리

- Ray tracing spheres
- Ray tracing polygons

- Recursive reflection
- Recursive refraction
- Phong illumination
- Export image files
- Texture mapped spheres and polygons
- Report
- Representative pictures
- Bump mapping(extra)
- Distributed ray tracing(extra): soft shadows, motion blur, glossiness

5. References

[1] https://stackoverflow.com/questions/36288421/c-create-png-bitmap-from-array-of-numbers/58395323
[2] Carbon texture: https://www.pinterest.cl/pin/363876844871773928/
[4] Checkerboard: https://eleif.net/checker.html
[5] Earth: https://www.peerservice.org/
[6] Rock normal map: https://www.pinterest.co.uk/pin/435934438924595238/