Computer Graphics Assignment 4 Report

2023-15725 곽민서

1. Abstraction

Local Illumination을 이용한 Shading에는 정교한 음영, 굴절과 반사와 같은 정교한 빛의 특성을 고려하지 못한다는 한계가 존재한다. 이를 위해 대안으로 제시된 것이 Global Illumination을 이용한 Ray Tracing 기법이다. Ray Tracing은 픽셀이 받는 광선을 역으로 추적해나가며 빛의 정보를 받아가는 기법이다. 이때 빛의 광선은 직진과 반사, 굴절을 하기에 빛의 물리적 특성을 담아낼 수 있는 것이다. 이번 과제는 현대 그래픽의 핵심이라고 할 수 있으며 고성능 그래픽카드의 본격적인 상용화의 신호탄이었던 Ray Tracing의 구현을 하는 것이다.

2. Implementation

이번 과제는 스켈레톤 없이 진행하였기에, 자유롭게 파일을 구상하였다. 사용된 파일은 다음과 같다.

- Func.py
  - 여러 파일에서 공통적으로 사용된 함수 몇가지를 모아놓은 파일이다.
- Image.py
  - 픽셀당 색생값을 이미지 파일로 바꿔주는 클래스가 정의된 파일이다. 이미지 제작에는 MatLab의 PyPlot이 이용되었다.
- Object.py
  - Cornell Box에 나타난 다양한 물체에 관한 클래스가 정의된 파일이다. 가상 클래스 Shape을 상속한 폴리곤 Polygon 클래스와 Sphere 클래스가 정의되어있다.
- Ray.py
  - 레이트레이싱을 위한 광선에 관한 클래스가 정의된 파일이다. 광선 추적이 핵심 메소드이며, 일반 광선 Rays와 그림자광선 ShadowRays가 정의되어있다.
- Render.py
  - 카메라와 렌더링에 관한 클래스이다. Camera 클래스의 대부분의 상수나 계산 과정은 OpenGL에서 가져왔으며, render 메소드는 여러 광선을 병렬 연산하여 조사한다.
- World.py
  - Cornell Box 그 자체에 관한 클래스이다. Cornell Box의 모든 물체의 정보를 저장하고, Rays가 Intersection을 확인할 때 사용할 hit 메소드를 제공한다.
- Test.py

- 물체의 obj파일을 만드는데 사용하는 파일이다. 레이트레이싱 프로그램 그 자체에는 영향을 주지 않으며, 우선적으로 실행하여 obj파일을 만든후, 이를 본 프로그램에서 읽어서 반영하게 된다. 다양한 물체를 만들 수 있는데, 현재는 3축 회전을 하는 정육면체를 만들어낸다

- Material.py
  - 물질의 특성에 관한 클래스이다. 물질의 색상이나 반사율 등을 정의한다..

- Main.py
  - 본함수이다. World와 Camera를 정의하고 다양한 Object를 만들어서 World에 추가시키고 이미를 만들어낸다.

[implementation Detail]

1. Ray.py

Class Rays가 핵심이 되어 작동한다. Rays는 시작점, 방향 벡터, 색상, 빛이 속해있는 굴절률의 클래스변수를 가진다. 본 프로그램에서 Rays를 비롯한 모든 프로그램은 GPU 병렬연산을 수행하는 PyTorch의 Tensor를 기반으로 연산을 수행한다. 그리하여 모든 클래스변수 또한 텐서로 저장이 되어있다. Rays에는 연산을 편하게 하기 위하여 여러 메소드가 정의되어있다. Rays를 직선의 방정식이라고 할 때, t값에 따른 좌표를 반환하는 t2p, 방향벡터의 길이와 그 제곱은 length, squared_length, 단위 방향벡터인 unit_direction이 있다.

이 클래스의 핵심은 trace메소드로, 이 광선과 World 변수, depth를 받아 픽셀의 색상을 반환한다. 우선 이 광선과 World의 Intersection 정보를 읽고, 가장 가까운 교점의 t는 near로 저장한다. Color은 우선 0부터 시작한다. Hit은 intersection정보와 점에 관한 정보만 주므로, world.object와의 Join을 통해 Object의 정보를 얻는다. 만약 hit이 되었다면, 즉 무한의 범위 내에서 교점이 있다면 색을 계산한다.

색은 여러 종류의 색이 있다. Hit만 되었다면 상황에 상관없이 존재하는 Ambient Light, 그림자가 없다면 추가되는 Diffuse Color, 그리고 반사광과 굴절광이다. Ambient Light는 Object의 getambient를 통해 우선적으로 얻게되고, 이제 그림자의 여부를 파악해야한다. 부딪친 점에서부터 광원까지의 ShadowRays에 대해 이 ShadowRays가 World와의 Intersection을 구해서 가장 가까운 점까지의 거리와 광원까지의 거리를 비교하여 hit여부를 판정한다. Hit일 경우, 다른 빛들도 더해주는데, 이 빛의 방향, 노멀벡터, 빛의 색상 등 여러 정보를 바탕으로 Object의 diffuse color을 구하는 getColor메소드로 확산광의 색을 구한다.

다음은 레이트레이싱의 핵심이라 할 수 있는 굴절과 반사이다. Hit된 점으로부터, 반사광과 굴절광에 관한 Rays를 만들고, 그 Ray각각의 Trace를 수행한다. 이렇게 해서 얻은 빛을 적절히 조합하여 한가지 색으로 만들어줘하는데, 이는 Material의 cols와 kersnel에 의존한다. 기본 색상과 굴절 및 반사광의 비율을 결정하는 cols, 반사광과 굴절광의 비율을 나타내는 kersnel에 따라 색을 분배한다. 이때, cols가 0일 때, 색상 100프로이고, 1이면 반사굴절 100프로이며, kesnel이 1일 때 완전 반사광, 0일 때 완전굴절광이다. 그림자와 굴절, 반사는 다른 광선을 파생시킬 때, 0.1씩 방향으로 더한 값을 origin으로 삼는데, 이는 엡실론 기법(?)을 이용하여 중복 그림자를 막는다.

마지막으로 고려해줄 것은 depth인데, 한번의 굴절 및 반사시마다 depth가 1씩 감소한다. Depth가 1일때엔 반사광과 굴절광은 계산하지 않는다.

2. Object.py

클래스 계층이 구현되어있다. 최상단엔 가상 클래스 Shape이 있으며, 이는 ABC 라이브러리를 통해 구현된다. 이는 폴리곤에 관한 Polygon 클래스와 구에 관한 Sphere 클래스에 의해 상속된다. 모두 각각 Object와 광선의 교차 여부를 확인하는 intersect, 확산색상을 얻는 getColor, 주변광에 관한 getambient, 해당 점에서 노말벡터를 얻는 getnormal을 가진다.

Polygon은 정점 3개로 이루어진 폴리곤에 관한 클래스이다. 세 점을 3x3 텐서로 가지고 있으며, 사용자가 직접 입력했던 노말값이나 혹은 3개의 정점을 Cross하여 얻은 노말값을 가진다. 또한 무슨 소재로 이루어져있는지에 관한 material도 가진다.

Intersect여부는 BaryCentric Coordinate를 이용해 결정한다. 평면의 방정식과 직선의 방정식의 교점을 구하고 각 점으로 부터의 무게의 비율을 이용하여 Intersect되었는지 판단한다. 그리고 P + t*d에서 t를 반환한다. T_max는 사실상 무한대의 교점으로, 이는 교점이 없었음을 의미한다.

getColor은 Lambert의 법칙을 따른다. 거리에 제곱하여 작아지고, cosine값에 의존하는 특징을 반영하여 물질, material에 diffuse한 결과를 반환한다. Getambient는 단순히 material의 ambient값에 material의 반사율에 곱해서 반환한다.

Getnormal은 저장되어있는 노말벡터를 단위벡터로 반환한다.

Polygon외에도 Sphere가 저장되어있는데, 굳이 폴리곤으로 표현할 수 있긴 하나, 구는 수식으로 표현한다면 어느 물체보다도 간단히 표현할 수 있는 object기에, 우선적으로 구현해보았다. 중심의 좌표와 반지름, 그리고 물질 정보를 저장한다.

Intersect는 기하학적 특징을 이용해 구한다. 점과 직선사이의 거리 공식을 이용해서 만약 거리가 반지름보다 작다면 만나고, 크면 안만나는 것을 이차방정식의 판별식을 이용하여 판단한다. 이때, 더 작은 해를 반환한다. getColor계산시 노말벡터가 요구되는데, 이는 해당 점과 중심까지의 벡터를 이용하면 더 쉽게 얻어진다.

3. World.py

세상 그자체를 정의한 World에 관한 파일이다. World는 World에 있는 모든 Object를 담고 있으며, 광원의 위치 정보를 가지고 있다. World는 initialize시 object를 추가해주는 add와 광선의 모든 intersection을 수행하는 hit 메소드를 가진다. Hit이 핵심인데, hit은 모든 object와 입력한 광선의 intersect를 구하고, 이 중 가장 작은 t를 반환한다. 이에 torch의 stack과 min이 사용된다.

4. Material.py

물질의 정보에 관한 파일이다. Material.py에는 ambient값, kersnel값, cols값, 그리고 Kd값, 굴절률을 가지고 있다. 확산광을 구하는 diffuse가 여기에 정의되어있는데, diffuse는 Lambert Law에 따라 구현하였다. 빛의 세기는 10으로 가정하였다.

5. Render.py

카메라 클래스. 카메라를 정의하는데 필요한 lookfrom, lootat, vup, fov과 이미지 생성을 위한 width, height, render depth를 기록하여 갖가지 값을 저장한다. 자세한 구현은 OpenGL을 참고하였고, 설명이 필요없다고 생각한다.

핵심은 render메소드로, Camera와 World의 상호작용을 통해 이미지를 만들어낸다. 원래대로라면 한픽셀씩 계속 순회하며 값을 얻어야하나, PyTorch를 사용하여 병렬 연산을 지원하므로 모든 픽셀의 광선을 한 개의 텐서에 담아서, 그러니 300x200이라면 60000개의 광선을 하나의 텐서로써 Rays로 만들어 trace를 한번에 진행하여 속도를 매우 향상시켰다.(안타깝게도 추가점수는 아니다...) 또한 레이트레이싱의 한계를 극복하기 위해 안티얼레이싱을 진행하였다. 광선을 1번만 쏘는 것이 아닌, 여러 번 반복한다. 픽셀을 중심으로 랜덤하게 플러스마이너스하여 얻은 모든 색상값을 평균을 내어 더 정교한 색을 얻는다. 안티얼레이싱 방법과 병렬연산 값은 인터넷의 도움을 받았다.

6. main.py

메인 함수이다. Camera와 World를 생성하고, 파일을 읽어와서 Object를 만든뒤, Render하여 이미지 파일을 얻어서 저장까지 한다.

7. image.py

이미지를 다루는 파일이다. 텐서로써 저장이 되어있는 이미지 파일을 재정렬하여 RGB 3채널의 png 파일을 만들어주도록 한다. 즉, 넘겨받는 이미지 텐서데이터를 재정렬하여 PLT의 이미지 지원 기능을 이용한다. 재정렬은 einops를 사용하였다.

8. func.py

모든 파일에서 두루두루 사용하는 여러 함수가 정의되어있다. 주어진 벡터를 정규화시키는 unit, 입력된 벡터의 크기를 단순 반환을 하는 것이 아닌, 복잡한 텐서에 대해서도 브로드캐스팅해서 반환하는 dirunit, 색을 0~1의 값으로 최저와 최대를 한정시키는 rcolor, dirunit처럼 작동하는 dot.

9. param.py

별거 없다. 사용하는 몇가지 상수에 대해 정의되어있다. 최대 t인 t_max(1000), 그리고 텐서에 사용될 디바이스 정보이다. 이때, 텐서는 cuda로 정의되어있으므로, 반드시 cuda 환경에서 구동해야한다.

10. test.py

구현에서 사용할 Object 파일을 만드는데 사용한 파일이다. 다양한 구현을 시도하였으나, 현재는 3축 회전한 정육면체를 만드는 파일이다. 주어진 정점을 중심으로 회전하는 것으로, 4x4 아핀 변환을 통해 8개의 변환 정점, 이에 관한 인덱스, 그리고 노말 벡터와 색상 정보를 기록한다. 원하면 다른 파일로 바꿀 수 있다. 이미 object파일은 생성된 상태이므로, 이 파일은 없어도 상관없다. 같은 디렉토리 내 testsphere.py(실패작)도 같은 역할을 한다.

11. world.txt / rotated_cube.txt

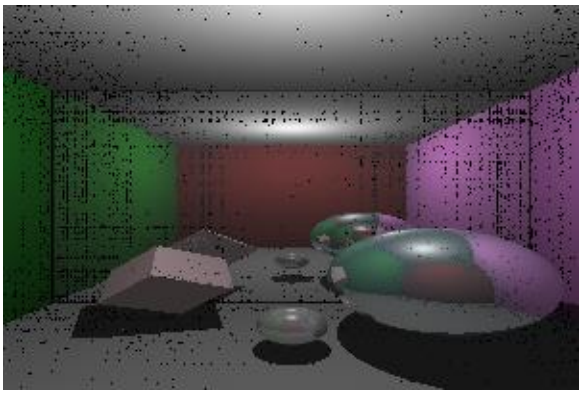
오브젝트 파일이다.(txt로 저장된) world.txt는 Cornell box의 6개의 면과 색상 정보, rotated_cube.txt는 test.py에 의해 생성된다.

3. Result

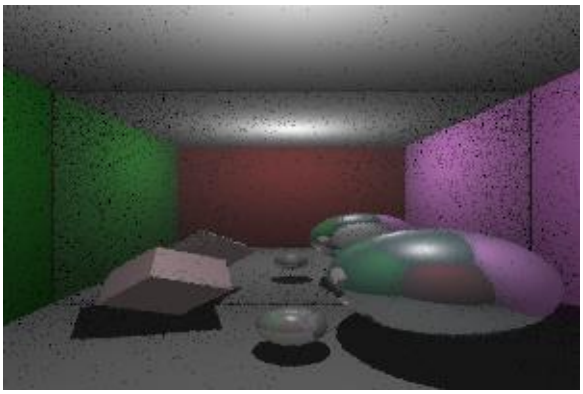
모든 경우에 따른 스크린샷을 보여줄 순 없겠지만, 핵심 구현을 중심으로 보여주도록 하겠다.

몇 개는 구에 대한 굴절을 계산하기 어렵기에, 잠시 반사로 대체하였다.

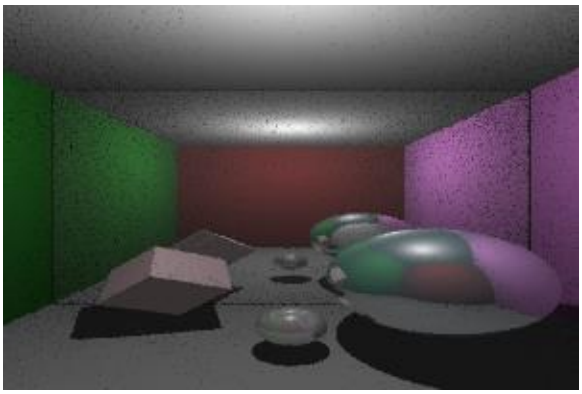
1) 안티엘리어싱



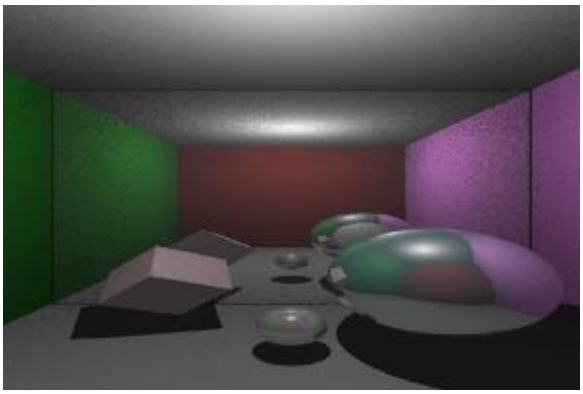
Antializing = 1 (No Antializing)



Antializing = 2



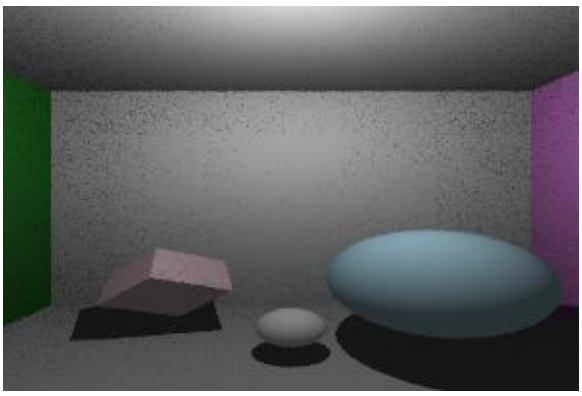
Antializing = 4



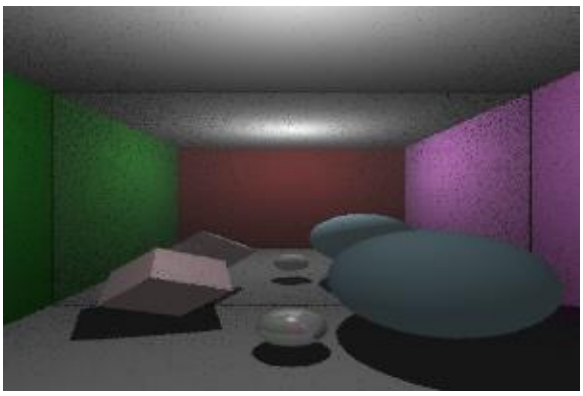
Antializing = 10

안티얼라이징을 할수록 검은 점이 사라지는 것을 볼 수 있다.

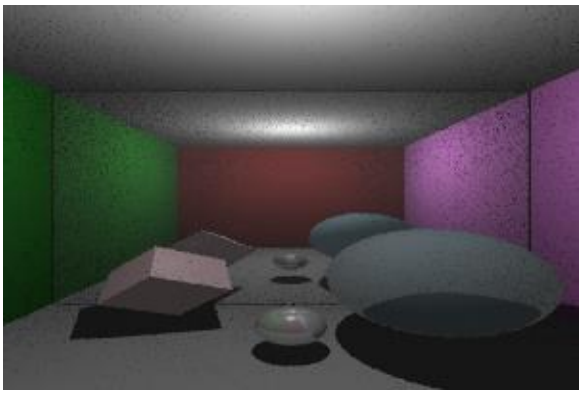
2) depth



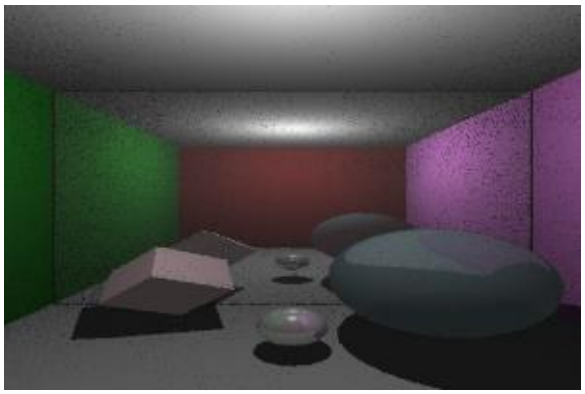
Depth = 1



depth = 2



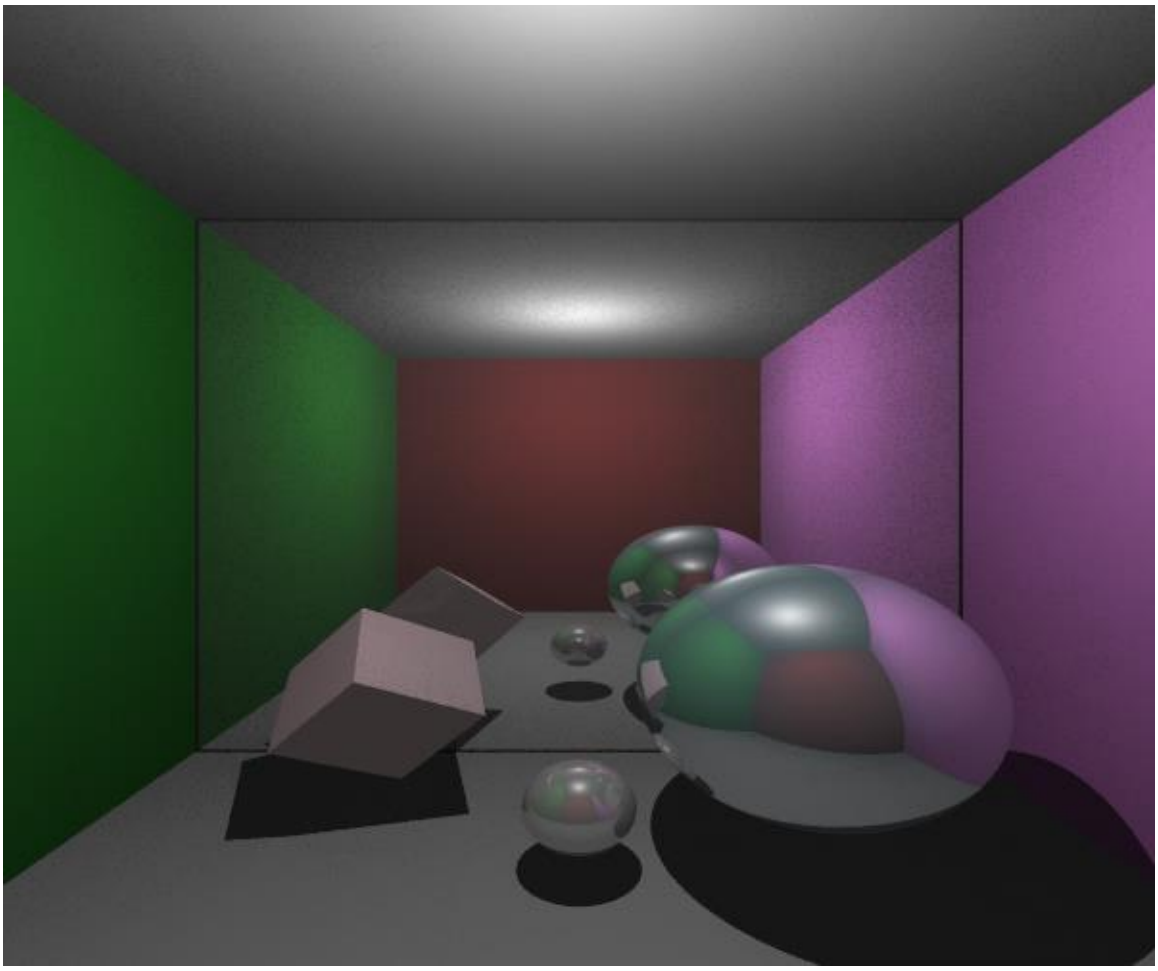
Depth = 3



depth = 4

Depth 에 따라 표현하는 물리적인 효과가 다르게 나타난다. Depth 1 에서는 어떠한 Ray Iteration 도 하지 않기에, 굴절, 반사 없이 그림자만 나타나는 쉐이딩에 불과하다. Depth 2 에서는 반사가 지원되기에, 뒷면의 거울과 앞의 구에서 반사효과가 나타난다. 하지만, 반사도 1 번밖에 안되기에 이중 거울 효과는 나타나지 않는다. 또한, 굴절 빛도 아직 구 내부로 진입한 것이 전부이기에, 구의 내부모습만을 보여주고 있다. Depth 3 에선 이제 거울에 2 번 반사되는 효과를 볼 수 있다. 또한, 굴절도 이제 구의 내부를 빠져나왔기에 굴절되어서 그 너머가 보인다. Depth 4 에선 굴절광선이 굴절을 넘어 거울에 까지 부딪치면서 구 내부에서 반사된 빛이 또 비춰져 더 정교한 묘사가 이루어졌다.

5) 최종 결과 – 1 : 반사만



Pixel Width = 600, Pixel Height = 500, Antializing = 20, depth = 4

World

6 Planes, each 10 x 10

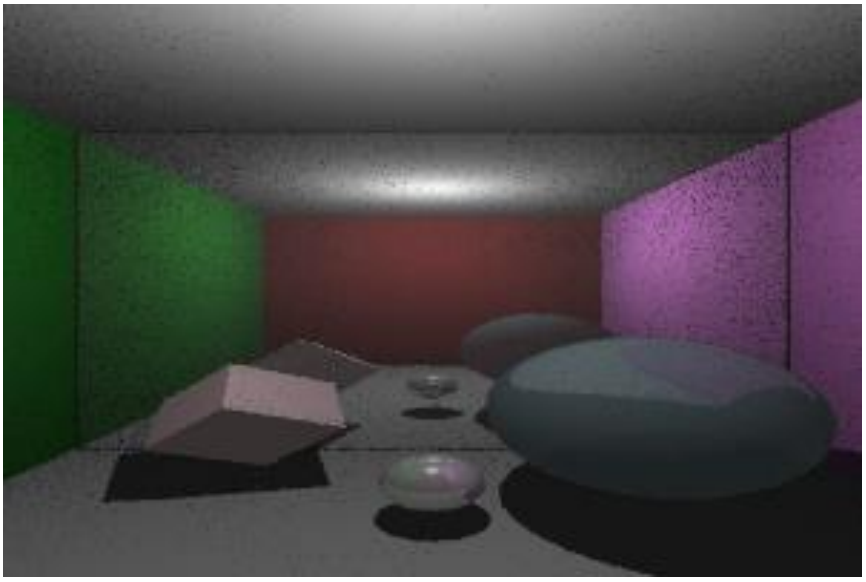
Light 0 5 5

Object 1 : Polygon x 6 (From spinned_cube.txt) / Background Plane is Relecting

Object 2: Sphere (small one) / Reflecting

Object 3 : Sphere (big one) / Reflecting

뒷면의 Plane은 거울로 Reflecting하고 있다. 그리하여 카메라보다 뒤에 있어서 보이지 않은 빨간색 벽이 비춰지고 있다. 약간 삐딱하게 기울어진 정육면체는 object파일을 읽어서 생성한 파일이다. 그림자가 기울어진 육면체에 맞추어 제대로 그려진 것을 확인할 수 있다. 가운데의 작은 구슬은 Reflecting한다. 바닥에서 약간 떠있는데, 그리하여 앞의 그림에선 보이진 않지만, 거울로 봤을땐 명백하게 떠있는게 보인다. 이 또한 레이트레이싱이기에 가능한 그래픽이다.



Pixel Width = 300, Pixel Height = 200, Antializing = 5, depth = 4

위는 오른쪽 커다른 구가 굴절할 수 있도록 바꾼 것이다.

구의 색이 약간 탁한 것을 볼 수 있는데, cols값이 0~1사이이기 때문이다. 그리하여 원래의 diffuse색상인 하늘색이 은은하게 있는 것을 볼 수 있다. 구의 그림자는 단순히 바닥에만 생기는 것이 아닌, 벽면에도 드리울 수 있다. 같은 법칙이 적용되기 때문이다.

이 스크린샷은 레이트레이싱에서만 볼 수 있는 빛의 반사, 굴절, 직진의 성질을 모두 확인 할 수 있는 코넬박스이다.

4. 느낀 점 및 한계

지난번 쉐이딩에 이어 레이트레이싱 과제를 진행하면서 쉐이딩에서 느꼈던 로컬 일루미네이션의 한계를 많이 극복할 수 있었던 것 같다. 퐁 쉐이딩이 자연스럽긴 하지만, 현실을 완전히 반영하진 못하였다. 그리하여 텍스쳐의 도움 없이는 정교한 그래픽은 표현하지 못하였었다. 그때, 이러한 점이 개선되었으면 하는 점들이 모두 레이트레이싱에서 극복되었던 겉 같다.

5. 실행 방법

우선 사용한 라이브러리는 다음과 같다.

- Pytorch : CUDA 11.8 with RTX 2050
- Einops
- Matplotlib -> pyplot

- ABC, math 내장 라이브러리

우선 Pytorch를 설치해야한다. 이를 위해 RTX 2050 GPU 가속을 할 수 있는 CUDA 11.8버전을 설치해준뒤, OS 환경에 맞게 설치를 진행하였다.

Einops는 pip install einops로 설치가 가능하다.

matplotlib또한 마찬가지로 없다면 추가로 설치를 해주면 된다. Pip install matplotlib

그 외는 전부 내장 라이브러리로, 파이썬 3.11이상의 환경에서 내장되있다.

실행방법은 python main.py로 바로 실행가능하다.