그래픽스프로그래밍 HW5

화학부 2016-14043 이준영

1. Intersection function

   우선 카메라를 pinhole camera로 가정하여 카메라에서부터 TexCoord 방향으로 ray를 만들었다. 이것을 view space에서 world space로 옮겼고 단일 ray에 대하여 ray tracing을 시작하였다. 각각의 object (box, triangle, plane, sphere)에 대해 ray가 object에 부딪혔는지를 계산하는 intersection함수를 작성하였다. object의 종류마다 ray가 object을 hit 했는지 계산하는 방식이 거의 완전히 달랐다. 자세한 부분은 코드에서 확인할 수 있다. 만약 ray가 object에 부딪혔고 그 부딪힌 point가 이전 hit record의 point보다 카메라에서 가깝다면 hit record가 갱신된다. 이것을 모든 object에 대해 수행했다.

2. Phong illumination

   각각의 material마다 phong illumination의 식을 이용해서 col를 부여했다. phong illumination은 이전의 HW4과 거의 동일한 방식으로 진행했고 각 sample마다 네 개의 광원에서 비롯된 색을 모두 더했다. I_diffuse = I_specular로 가정했다.

3. Shadow

   Shadow를 계산하기 위해 각 hit point에서 광원(lights[0]만 shadow cast로 가정했음)으로 shadow ray를 만들었고, 만약 shadow ray가 object에 부딪힌다면 diffuse와 specular에 상수를 곱하여 diffuse와 specular가 color에 더해지지 않게 했다. shadow ray의 ray tracing은 object intersection을 구하는 함수와 거의 비슷하게 구현하였다.

4. Recursive reflection

   castRay함수 안에서, hit point에서부터 reflect vector 방향으로의 ray를 만들어서 다시 ray tracing을 반복하였다. 한 번 ray tracing을 할 때마다 material의 R0 값을 곱해서 color에 더했다. 이것을 MAX_DEPTH(= 4) 만큼 반복했다. hit point에서 reflect 방향으로 가는 ray는 다음과 같이 만들었다.

   ```
   Ray newRay;
   newRay.origin = hit.p + 0.00001 * normalize(hit.normal);
   newRay.direction = normalize(reflect(ray.direction, hit.normal));
   ray = newRay;
   ```

5. Environment map

   ray가 어떠한 object를 hit하지 못한 경우, skybox의 TexCoord에서의 color를 mapping하고 iteration을 종료했다.

6. Fresnel effect

Fresnel effect를 통해 ray direction과 hit point에서의 normal 사이의 각을 구해 smooth reflection을 구현했다. 다음은 shlick 함수와 R0를 구하는 코드이다.

```
float schlick(float cosine, float r0) {
    return r0 + (1 - r0) * pow((1 - cosine), 5);
}

vec3 schlick(float cosine, vec3 r0) {
    return vec3(schlick(cosine, r0.x), schlick(cosine, r0.y), schlick(cosine, r0.z));
}
```

```
float cosine = dot(hit.normal, ray.direction)/(length(hit.normal)* length(ray.direction));
prevR0 = prevR0 * schlick(cosine, hit.mat.R0);
```

7. 기타 사항

   이외 optional은 구현하지 않았다. 스크린샷은 screenshots 폴더에 있다. a와 b는 freeimage lib을 이용해 찍었지만 이후, MacOS에서 화면 크기가 이상하게 캡처되는 현상이 있어서 c~f는 따로 스크린샷을 찍었다. a와 b에서는 box의 normal을 잘못 구현한 부분이 있었는데 c부터는 수정해서 box의 normal을 올바르게 구현했다.

8. Environment

   a. Language : C++17

   b. Compiler : clang++

   c. IDE : Visual Studio Code

   d. OS : MacOS Monterey 12.3

   e. tasks.json (for build)

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: clang++ build active file",
      "command": "/usr/bin/clang++",
      "args": [
        "-std=c++17",
        "-fdiagnostics-color=always",
        "-Wall",
        "-g",
        "-I${workspaceFolder}/includes",
        "-L${workspaceFolder}/lib",
        "${workspaceFolder}/lib/libglfw.3.3.dylib",
        "${workspaceFolder}/lib/libfreetype.6.dylib",
        "${workspaceFolder}/lib/libassimp.5.2.0.dylib",
        "${workspaceFolder}/lib/libfreeimage.3.18.0.dylib",
        "${workspaceFolder}/assignments/hw5/src/*.cpp",
```

```
        "${workspaceFolder}/assignments/glad.c",
        "-o",
        "${workspaceFolder}/assignments/hw5/bin/main",
        "-framework",
        "OpenGL",
        "-framework",
        "Cocoa",
        "-framework",
        "IOKit",
        "-framework",
        "CoreVideo",
        "-framework",
        "CoreFoundation",
        "-Wno-deprecated"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "compiler: /usr/bin/clang++"
    }
  ]
}
```