Computer Graphics hw4

2014-18992 DongJin Shin

2017-05-22

1 Recommended Environment

- Linux
- Graphic card supports GLSL $\geq$ 3.3
- OpenGL $\geq$ 3.0
- C++ $\geq$ 6.3.1
- CMake $\geq$ 3.7.2

2 Execution

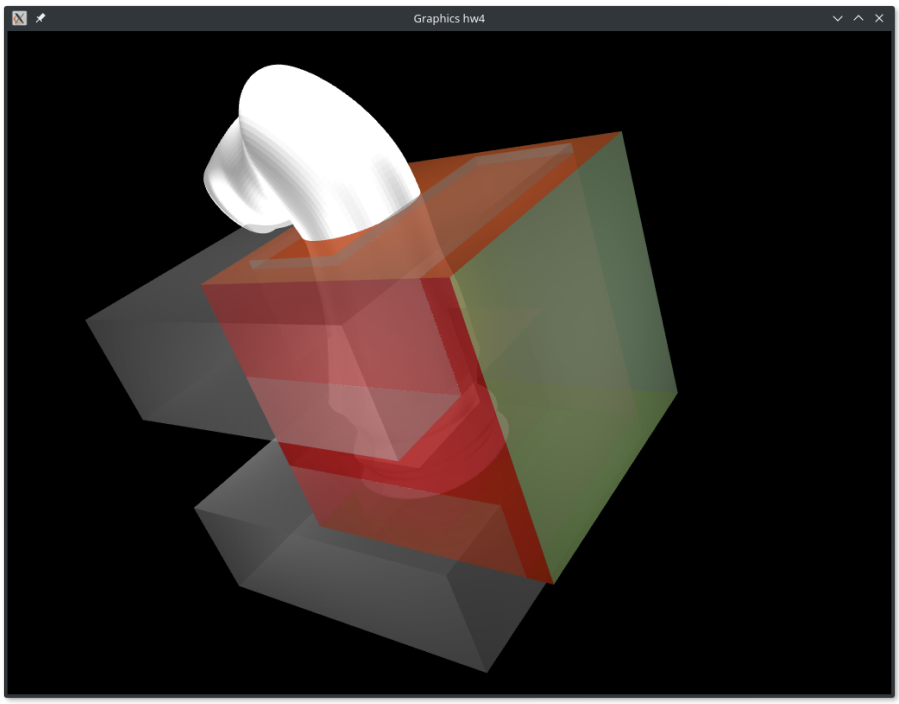
- `mkdir build` (At root directory, where `CMakeLists.txt` is contained)
- `cd build`
- `cmake ..`
- `make`
- `cd ../hw4/` (Directory should be correct, since it loads `obj` and shader files in relative path)
- `./hw4`

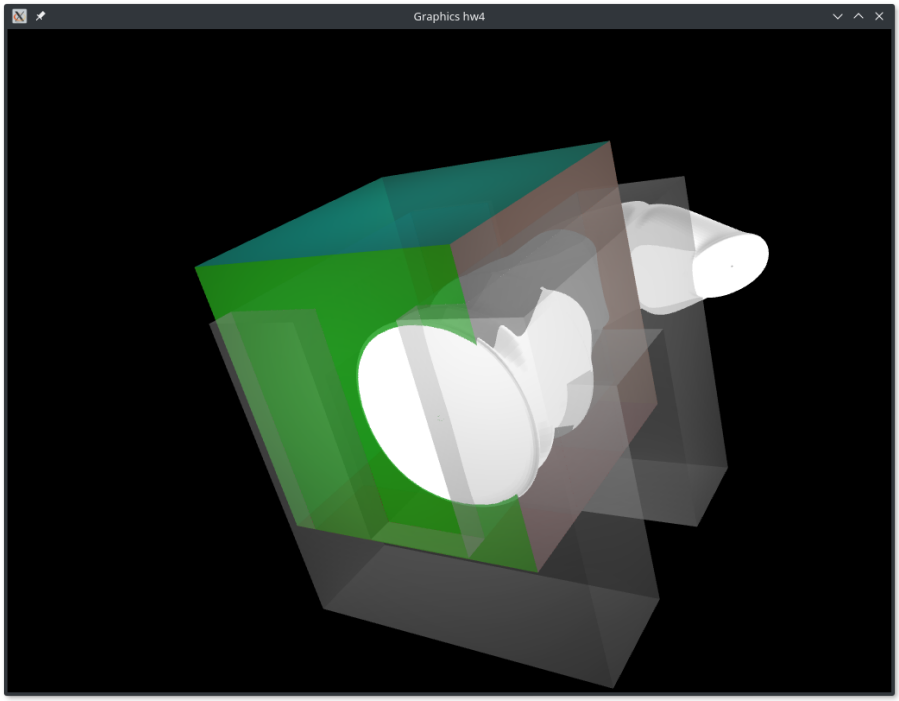
3 Controls

- Arrow keys to move camera position
- Page Up / Down to dolly in / out
- Home / End to zoom in / out
- Mouse drag to rotate
- Right click to seek
- ESC to exit

4 Description

기본적으로 과제에서 주어진 모든 기본 및 추가사항을 구현하였다.





- 위 이미지로 볼 수 있듯이, 불투명한 체스말, 반투명한 정육면체 그리고 복잡한 반투명 물체로 반투명한 ㄷ자 입체 구조물 총 3개를 겹쳐 shading을 하였다.
- 정육면체의 각 면은 서로 다른 ambient, diffuse, specular 계수를 가지게 하여 각각 다른 material을 표현할 수 있도록 구현하였다.

  material에 대한 정보는 `http://www.it.hiof.no/~borres/j3d/explain/light/p-materials.html`를 참조하였으며, `main.cpp` 156-185 line에 구현되어 있다.

  각 면의 material은 다음과 같다.

  - Yellow Rubber : $K_a = (0.05, 0.05, 0.0)$, $K_d = (0.5, 0.5, 0.4)$, $K_s = (0.7, 0.7, 0.04)$
  - Chrome : $K_a = (0.25, 0.25, 0.25)$, $K_d = (0.4, 0.4, 0.4)$, $K_s = (0.774597, 0.774597, 0.774597)$
  - Copper : $K_a = (0.19125, 0.0735, 0.0225)$, $K_d = (0.7038, 0.27048, 0.0828)$, $K_s = (0.256777, 0.137622, 0.086014)$
  - Emerald : $K_a = (0.0215, 0.1745, 0.0215)$, $K_d = (0.07568, 0.61424, 0.07568)$, $K_s = (0.633, 0.727811, 0.633)$
  - Ruby : $K_a = (0.1745, 0.01175, 0.01175)$, $K_d = (0.61424, 0.04136, 0.04136)$, $K_s = (0.727811, 0.626959, 0.626959)$
  - Cyan Plastic : $K_a = (0.0, 0.1, 0.06)$, $K_d = (0.0, 0.50980392, 0.50980392)$, $K_s = (0.50196078, 0.50196078, 0.50196078)$

- Depth ordering을 위해 BSP tree를 구현하였다. 자세한 구현은 `bsp.h`에서 확인할 수 있다.

  Opaque한 체스말을 가장 먼저 렌더링한 후, 반투명한 표면을 가진 정육면체와 ㄷ자 구조물은 하나의 BSP트리로 구성하여 렌더링 순서를 설정하였다.

- Viewing은 hw2에서부터 구현한 기능을 그대로 사용하였다. Lighting은 총 6개의 백색광을 두어, 정육면체의 각 면 방향에 위치하도록 설정하였다. 각 광원의 위치는 `main.cpp` 248-253 line에 구현되어 있다.

  모든 광원은 directional light source로 작동하도록 구현하였다.

  효율적인 계산을 위해 셰이더에 6개를 각각 입력하여 연산하였다. 자세한 구현은 `shader/LightShading.frag`에서 확인할 수 있다.