컴퓨터 그래픽스 HW4

2017-11522 컴퓨터공학부 박종석

과제 구현사항 및 구현방법

Build your own scene
obj 파일을 읽어서 각 정점과 법선 벡터들의 정보를 읽고 그것들을 polygon의 vector 형태로 저장했습니다. polygon은 float의 vector로 저장했습니다. 그리고 rendering할 때, 이 polygon들을 glDrawArrays를 통해서 그리게 됩니다. 저번 과제로 만든 swept surface인 coffee cup도 scene에 포함시켰고 implicit surface로는 구를 몇 개 scene에 포함시켰습니다.

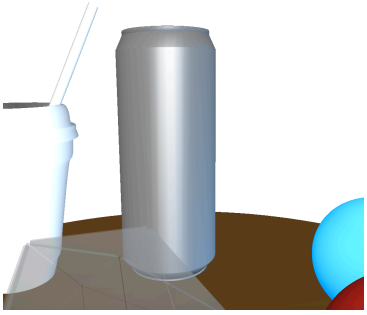
Material
각 물체 별로 최대한 다른 재질을 표현하려고 노력했습니다. 구현한 재질은 다음과 같습니다.

테이블 - 코팅된 나무

흔히 볼 수 있는 테이블에 사용되는 나무 재질을 표현하고자 했습니다. ambient와 diffuse는 갈색 빛(111/255, 79/255, 40/255, 1.0)을 반사하도록 했고 나무가 정반사되는 정도가 적다는 점을 반영해서 shininess를 8 정도로 주고 specular로 (0.5, 0.5, 0.5, 1.0)으로 채택했습니다.

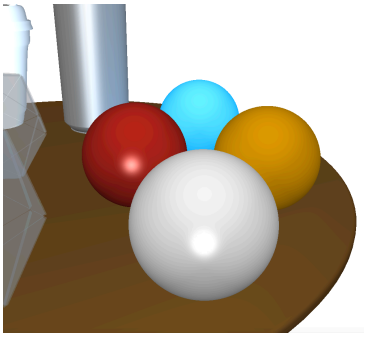
머그컵 - 세라믹

컵 재질로 가장 많이 사용되는 세라믹을 표현하고자 했습니다. 흰색이므로 ambient와 diffuse 모두 (1.0, 1.0, 1.0, 1.0)을 택했고 정반사가 많이 되는 세라믹 재질의 특성 상 specular는 (1.0, 1.0, 1.0, 1.0)에 shininess를 128로 주었습니다.



캔 - 알루미늄

흔히 볼 수 있는 캔의 재질을 표현하고자 했습니다. 기본적으로 회색을 난반사하도록 ambient와 diffuse로 (0.7, 0.7, 0.7, 1.0)을 주고, 정반사가 있으면서도 매트한 느낌을 주기 위해서 specular (1.0, 1.0, 1.0, 1.0)에 shininess를 30으로 주었습니다.



구 - 발광 재질, 광택 플라스틱, 고무

고무 재질의 구는 정반사가 없도록 ambient와 diffuse만 해당 색깔을 반사하도록 (232/255, 169/255, 53/255, 1.0)을 주었고, 맨 뒤의 발광 재질의 구는 emission으로 (12/255, 116/255, 232/255, 1.0)을 주었습니다. 왼쪽과 맨 앞의 구는 광택이 있는 플라스틱의 느낌을 내고자 왼쪽은 diffuse로 (232/255, 70/255, 53/255, 1.0)을 주고 ambient는 diffuse에 0.53씩 곱한 값을 주었습니다. 둘이 같은 값으로 하기에는 너무 specular light이 밝고 경계가 많이 져서 ambient를 약하게 했습니다. 흰색 구는 ambient와 diffuse를 모두 (1.0, 1.0, 1.0, 1.0)으로 주었고, specular는 (0.7, 0.7, 0.7, 1.0)으로 주었습니다.

테이크 아웃 컵 - 종이

테이크 아웃 컵은 종이 재질을 주고자 했습니다. 흰색인 (1.0, 1.0, 1.0, 1.0)으로 ambient와 diffuse를 주었고, 종이가 정반사가 거의 없다는 점에서 specular는 (0.5, 0.5, 0.5, 1.0)을 주고 shininess로 1을 주었습니다.

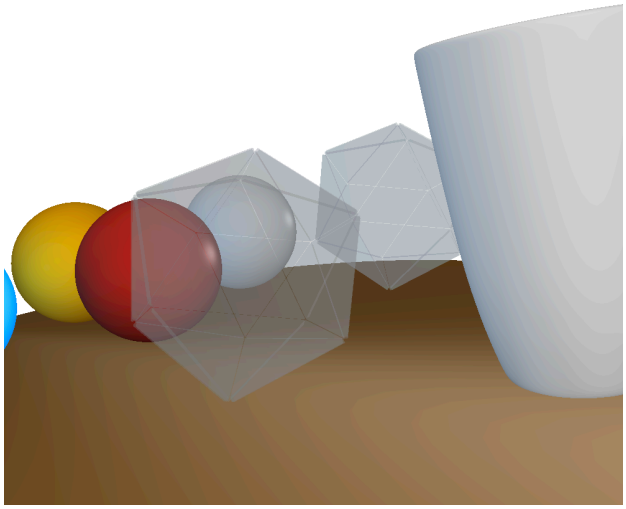
나머지 투명한 icosphere는 재질을 따로 주진 않고 투명한 색을 입혔습니다. material는 drawModel() 함수에 구현해놓았습니다.

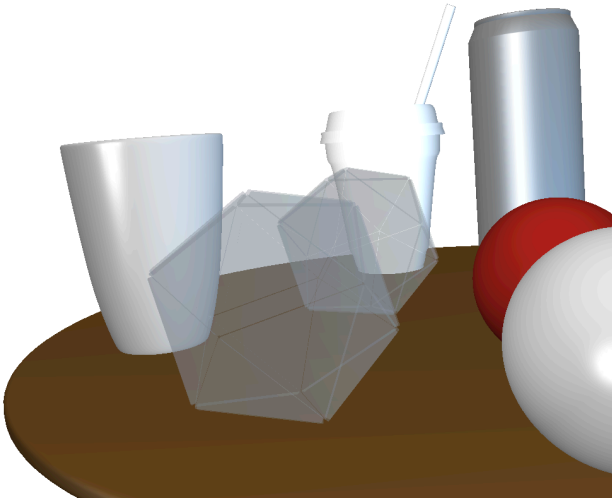
VIewing and Lighting

총 4종류의 light를 scene에 배치했습니다. 첫 번째와 두 번째 light는 diffuse를 조금씩 가지고 specular는 없는 light source를 (-1, 1, 0)과 (1, 1, 0) 방향으로 쐬어주도록 했습니다. 그리고 세 번째 light로는 viewpoint의 위치로부터 쏘는 diffuse를 조금 더 강하게 가지고 specular가 있는 light를 주었습니다. 그에 따라 보는 각도에 따라서 조금씩 다른 반사광이 나타나게 됩니다. 마지막 light는 scene에 있는 발광재질 구가 내뿜는 light를 표현한 light source입니다. scene에 있는만큼 attenuation을 주어서 멀어질수록 빛이 약해지도록 구현했습니다. 해당 코드는 setLight(float* rcam) 함수에 정리했습니다.

Depth Ordering

scene에 배치된 icosphere는 총 두개로 이 투명한 icosphere들이 제대로 rendering될 수 있도록 bsp tree를 이용한 depth ordering algorithm을 구현했습니다. 먼저 icosphere들을 구성한 polygon들을 받은 뒤에 이들을 크기를 기준으로 정렬하고 그 상태에서 bsp tree를 construct합니다. tree를 구성할때는 기본적으로 polygon의 vertex마다 현재 bsp tree node에 있는 polygon의 평면의 방정식을 통해서 그 평면을 기준으로 앞에 있는지 뒤에 있는지 판별식을 세웁니다. 그리고 그 판별식의 값이 모두 앞이거나 뒤일 때는 각각 맞는 children에 넣어주고, 만약 그렇지 않으면 걸친다고 판단해 polygon을 쪼개서 쪼개진 polygon들을 올바른 children에 배치합니다. 이런 식으로 recursive하게 tree를 생성하고 난 뒤에는 viewpoint가 바뀔때마다 viewpoint 위치에 따라서 tree를 traverse하여 viewpoint에서 멀리 있는 것부터 가까운 것 순서로 polygon들을 그리고 있습니다. 이 연산이 모든 polygon들에 대해서 하기에는 느려서 투명한 iconsphere들을 구성하는 polygon들만 depth ordering을 해주고, 나머지는 depth buffer를 통해서 올바르게 rendering하고 있습니다. 해당 코드는 BSPTree.cpp 파일에 구현했습니다. 그 결과 다음 그림과 같이 다양한 viewpoint에서도 문제없이 투명한 icosphere들이 rendering되는 것을 확인할 수 있습니다.





과제 조작방법

과제 3과 같이 viewpoint를 translation과 rotation을 통해서 옮길 수 있습니다. 조작방법은 과제3과 같습니다. 조작 방법은 다음과 같습니다.

Rotation: Drag
Translation: Shift + Drag
Zoom In/Out: Keyboard Up/Down
Dolly In/Out: Shift + Keyboard Up/Down
Show All: Keyboard 'a'
Seek: Keyboard ’s’ + Click
Back to Origin: Keyboard 'o'
Full Screen: Keyboard ’s’
Quit: Keyboard 'q'

과제 컴파일 방법

과제는 c++로 작성하였고 맥에서 작성하고 실행시켰습니다. makefile은 맥과 리눅스 모두에서 돌아가도록 고쳤습니다. makefile을 이용하여 main.cpp, vec3.cpp, BSPTree.cpp를 컴파일합니다. 라이브러리가 설치된 상태에서

1. make all
2. make run

이 두 명령어를 통해 과제를 실행시킬 수 있습니다.