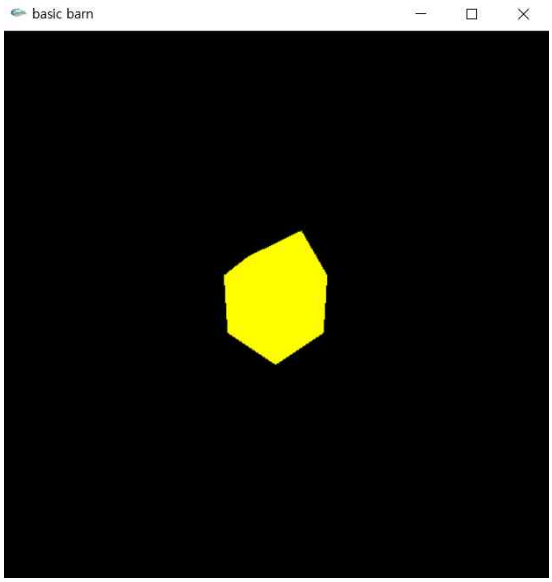
컴퓨터 그래픽스 HW4

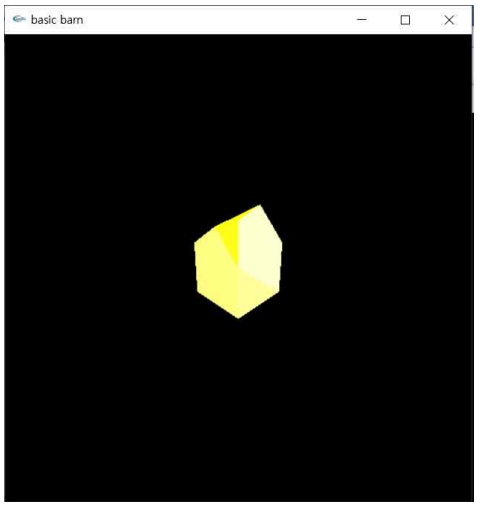
201501489 최영진

1.

① 실행 화면 캡처

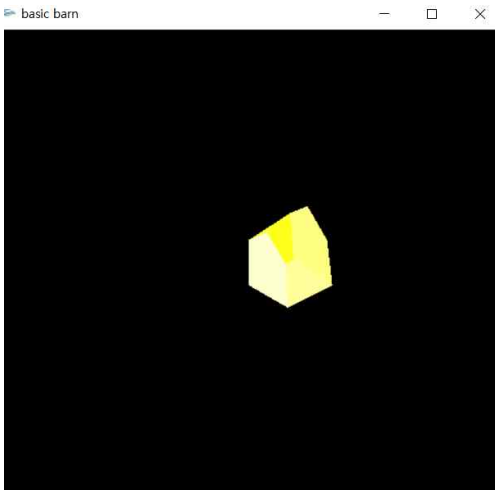


〈일괄 노란색〉



〈face별 노란색상에 차이 둔 경우〉

eye〈0.2, 0.5, 5.0〉



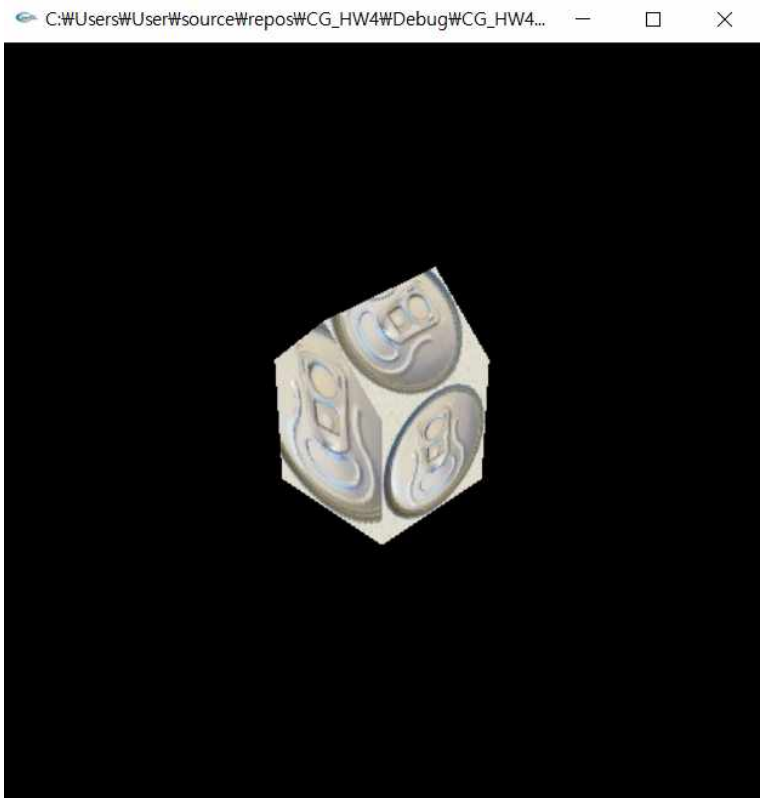
eye〈−5, −5, 5〉

② 설명 및 토의 사항

glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 15.0)로 설정, 각 face별로 GL_POLYGON을 사용하여 7개의 polygon을 그렸다. 제대로 그려졌는지 확인하기 위해 색을 다르게 해보고 gluLookat함수를 조작해서 여러방향에서 확인했다.

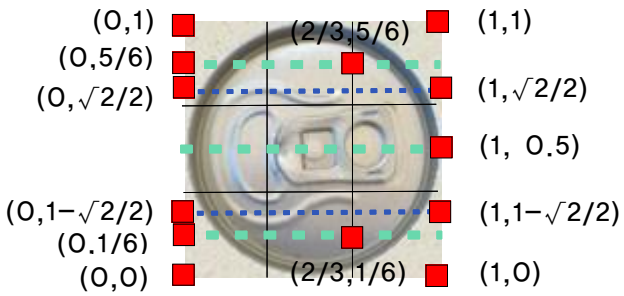
2.

① 실행 화면 캡처



② 설명 및 토의 사항

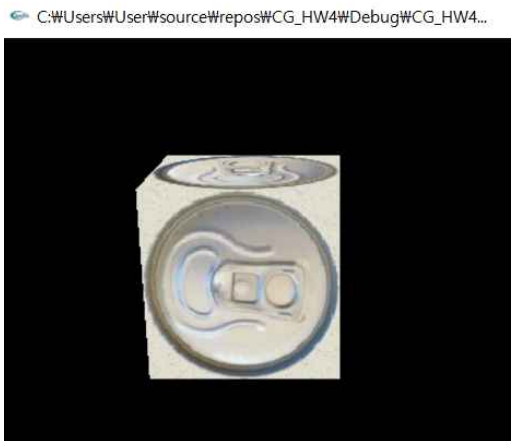
glFrustum(-1.0, 1.0, -1.0, 1.0, 3.0, 15.0)로 설정, front face의 양 옆쪽에 해당하는 정사각형 face(face 0, face 3)는 canTop의 그림이 반듯하게 서 있는 모양으로 mapping하기 위해 (0,1)를(bitmap 이미지의 왼쪽 위, 아래 그림 참고) 왼쪽 아래 vertex와, (0,0)를 face의 오른쪽 아래 vertex, 이미지의 오른쪽 아래(1,0)를 face의 오른쪽 위 vertex, 이미지의 오른쪽 위(1,1)를 face의 왼쪽 위 vertex와 대응시켰다. 밑 face도 마찬가지의 방식으로 지정했다. 지붕에 해당하는 부분은 직사각형이므로 길이를 계산하여 그림의 일부로 mapping했으며, front와 back에 해당하는 face는 이미지의 각 꼭짓점을 그대로 사용했다가는 비율이 망가지므로 마찬가지로 일부 잘라서 사용했다. 다만 mapping하는 과정에서 '/'연산이 되지 않았기에 2/3와 1/6에 해당하는 값을 X, Y로 지정해서 진행했다.



아래 그림은 gluLookAt 함수를 조작하여 확인할 수 있는 결과이다.

gluLookAt(5.0,5.0,−5.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0)

〈뒷면 확인용〉



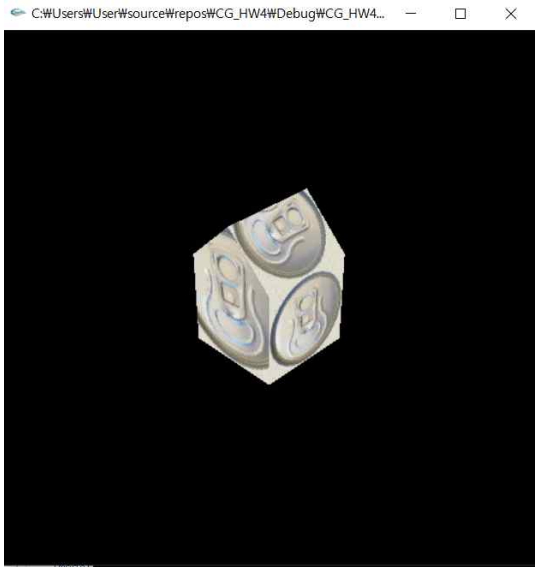
gluLookAt(2.0,−5.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0)

〈아래 확인용〉

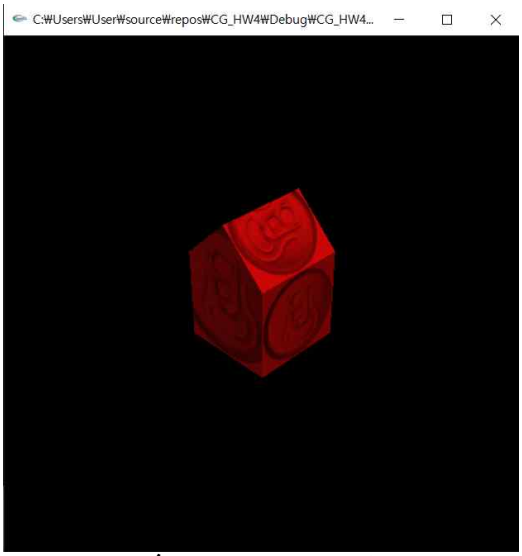
모두 의도한 대로 텍스쳐가 제대로 입혀졌다.

3.

① 실행 화면 캡처



〈GL_REPLACE〉



〈GL_MODULATE〉

② 설명 및 토의 사항

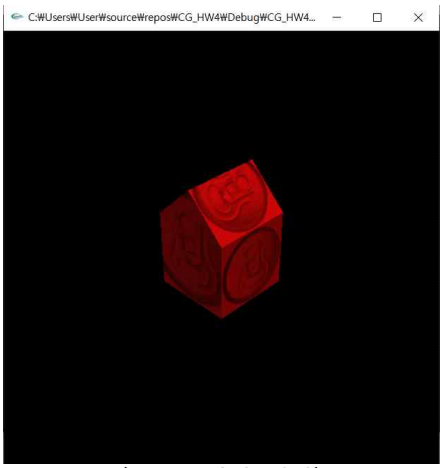
glFrustum(-1.0, 1.0, -1.0, 1.0, 3.0, 15.0)로 설정하여 위와 같이 보이도록 하였다.

텍스터 환경에 무엇이 들어가는지에 따라 조명과 텍스처가 동시에 적용되는지 그 여부가 달라진다.

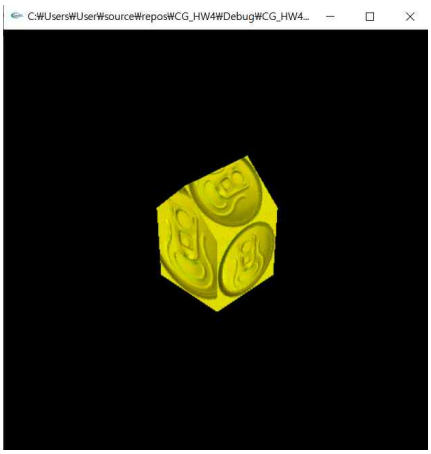
GL_REPLACE : 기존 물체면의 색을 완전히 텍스쳐 색으로 대체

GL_MODULATE : 기존 물체면의 색과 TEXTURE의 색을 곱함

각각의 환경에 대한 정의에 따라 GL_REPLACE는 완전히 텍스쳐만으로 대체되며, 조명과 텍스처를 동시에 사용하려면 GL_MODULATE 환경을 사용해야한다.



〈face 컬러 제거〉

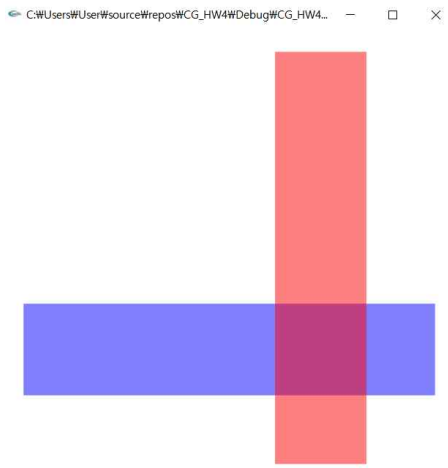


〈조명 제거〉

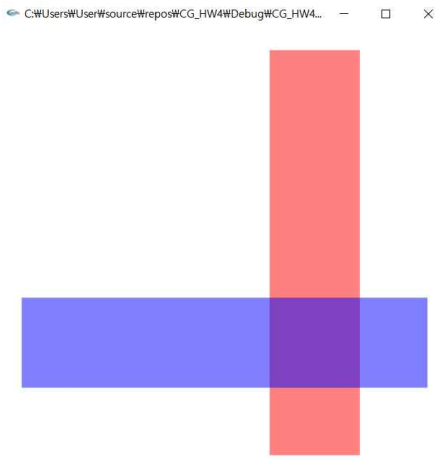
GL_MODULATE를 썼을 때 face의 색과 texture의 색이 혼합되었는지 비교하기 위해 각각 조건을 한번씩 지워보았다. face 컬러가 제거된 코드(좌측)는 기존에 face 컬러 코드가 있을 때와 차이가 없었다. 두 번째로, 조명 코드만을 주석 처리했을 때(우측)는 실행시켜보았더니 face컬러와 texture컬러가 혼합되어 나옴을 그림과 같이 볼 수 있었다.

4.

① 실행 화면 캡처 (blending)



〈source : R, destination : B〉



〈source : B, destination : R〉

② 설명 및 토의 사항

가시부피는 (0, 20, 0, 20, -1, 1)로 설정, alpha값은 모두 0.5로 주었다.

그려지는 순서에 따라 destination(원래 색)과 source(새로운 물체의 색)가 결정된다. 좌측 그림은 파란색 직사각형을 먼저 그렸기 때문에 destination이 blue의 색, source가 red의 색이다. 우측 그림은 빨간색 직사각형을 먼저 그렸기 때문에 destination이 red의 색, source가 blue의 색이다.

공식상으로는 둘이 같은 색상을 가져야 할 것 같지만, 사실 과제에서 사용한 직사각형의 색상들 역시 배경과 블렌딩 된 값을 가지고 있으므로 실제로는 색이 위처럼 다르다.

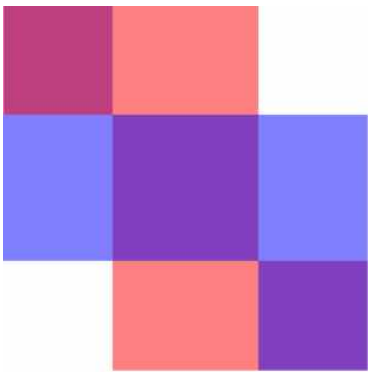
즉 파란색 직사각형이 먼저 그려질 때의 destination은 사실상 (0.5, 0.5, 1)이고 빨간색 직사각형이 먼저 그려질 때의 destination은 사실상 (1, 0.5, 0.5)임을 알 수 있다. (source R : (1,0,0), source B : (0,0,1))

이를 확인하기 위해 추가 코드를 작성하여 색 비교를 했다.

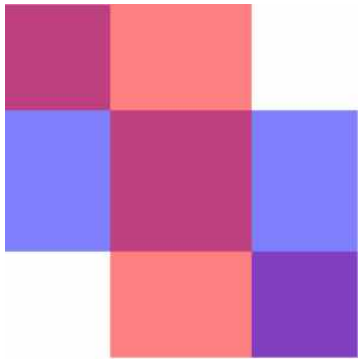
추가코드

```
//파란색(0.5,0.5,1)이 destination일 때 색상 비교
//빨간색이 source이므로 source의 알파값 sa = 0.5
// sa * (0.5, 0.5, 1) + (1-sa) * (1, 0, 0)  = (0.25, 0.25, 0.5) + (0.5, 0, 0)
glColor3f(0.75, 0.25, 0.5);
glRecti(9, 11, 12, 8);
//빨간색(1, 0.5, 0.5)이 destination일 때 색상 비교
//파란색이 source이므로 source의 알파값 sa = 0.5
// sa * (1, 0.5, 0.5) + (1-sa) * (0, 0, 1)  = (0.5, 0.25, 0.25) + (0, 0, 0.5)
glColor3f(0.5, 0.25, 0.75);
glRecti(16, 4, 19, 1);
```

코드에 따르면 교차해서 색이 바뀌는 부분의 왼쪽 위에 사각형은 blue가 destination, red가 source일 때의 색과 같아야 한다. 또한 오른쪽 아래 사각형은 red가 destination, blue가 source일 때와 색이 같아야 한다. 다음 그림은 각각 실행 시킨 후 해당 부분만 색상비교를 위해 잘라 가져온 그림들이다.



〈source : B, destination : R〉

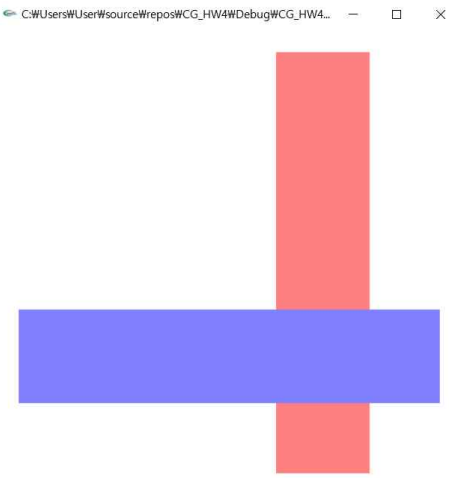


〈source : R, destination : B〉

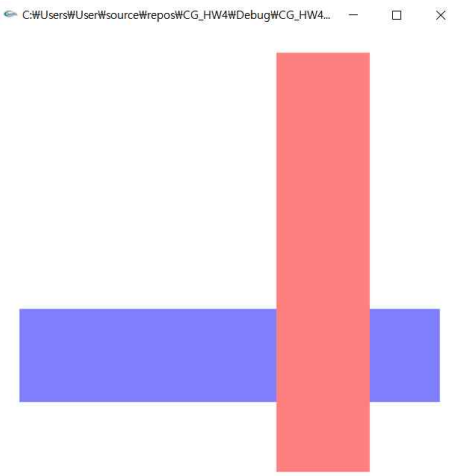
각각의 색상을 가까이 두고 직접 비교를 하니, 주어진 공식을 통해 계산한 색상과 blending 효과를 입힌 결과 색상이 서로 동일함을 확인할 수 있다.

---

① 실행 화면 캡처 (depth_buffer)



〈파란색 직사각형 먼저 그렸을 때〉



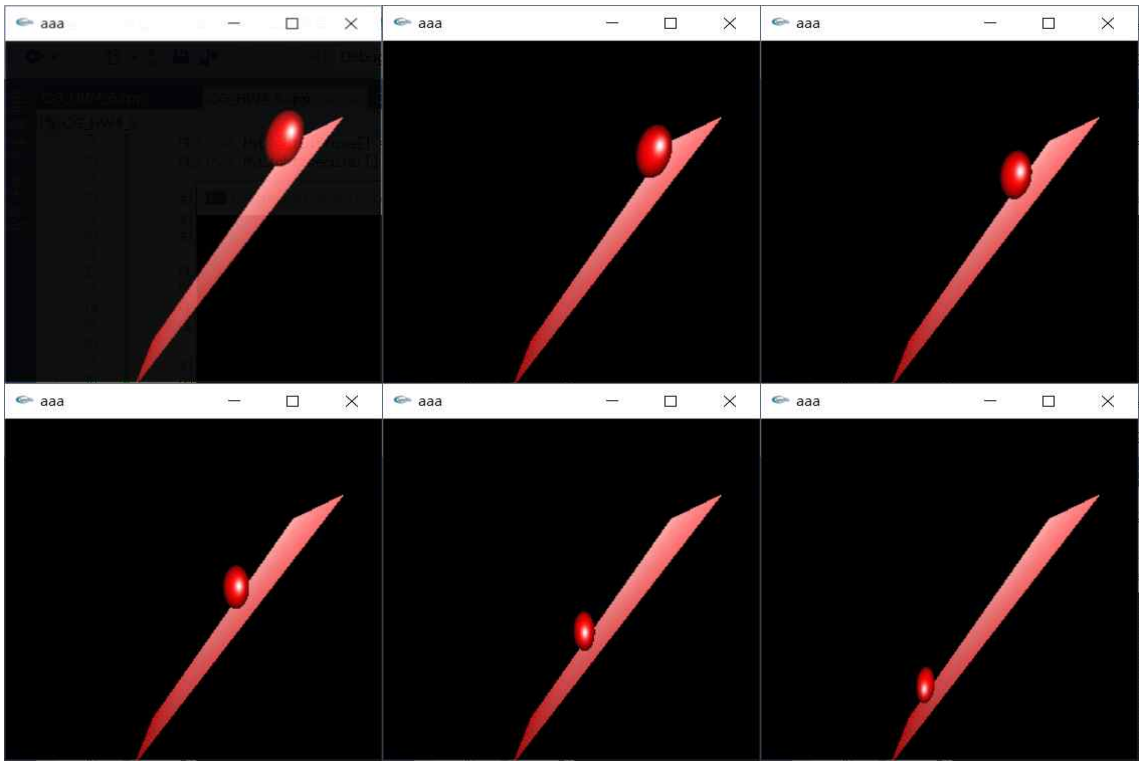
〈빨간색 직사각형 먼저 그렸을 때〉

② 설명 및 토의 사항

depth_buffer 사용은 Lecture9의 38page를 참고했다. 깊이 버퍼 테스트란, 시점에 가까운 fragment가 멀리있는 fragment를 가려 가까운 fragment의 깊이가 버퍼에 기록되는 것을 말하지만, 위의 경우에는 z좌표가 동일하기에 먼저 기록된 색의 값만이 남아서 보이게 된다.

5.

① 실행 화면 캡처



② 설명 및 토의 사항

glFrustum(−10.0, 10.0, −5.0, 5.0, 5.0, 100.0), gluLookAt(−1.0, 10, −50.0, 5, 5, −25.0, 0.0, 1.0, 0.0)로 설정했다. center의 z값이 eye의 z값보다 더 크므로 대략 +y방향으로 바라보게 된다.

구의 중심점 계산을 쉽게 하기 위해 판자의 기울기(y 변화량/x변화량)을 -1로 잡았고, 구의 중심점의 이동 경로 역시 z=−25인 평면 위 y=−x으로 설정했다. 밑으로 떨어질수록 속도가 빨라지는 것은 마찰력과 공기저항이 없는 공간에서 중력가속도 g의 영향을 받는 것으로 설정(중력가속도 g는 0.2)하여 x값에 t를 이용한 식(등가속도 운동 위치 변화량 $(a/2)*t^2$)을 넣었다. y값은 x값에 -를 곱하여 넣었다. z값은 -25로 변하지 않기 때문에 glTranslatef식에 다른 값을 넣을 필요가 없어, 0을 적었다.

6. 보너스 문제는 하지 않았습니다.