Computer Graphics 과제 4 보고서

컴퓨터 공학부

2004-11881 고우종

# 개발 환경

1. Microsoft Windows 7 Professional K (64bit)
2. Microsoft Visual Studio 2010 Professional
3. Mac OS X 10.6.7

# 실행 방법

Hw.sln 파일을 열어서 hw4 프로젝트 선택 후 Release나 Debug 모드에서 Run (F5)을 하시면 컴파일 및 실행이 됩니다. 이번 과제 역시 속도가 조금 요구되므로 되도록이면 release 모드에서 실행하시는 것을 추천 드립니다. 그리고 swept surface 데이터 입력은 “data.txt” 파일로 받고, 파일이 없으면 실행이 되지 않습니다.

혹시 **hw1** 프로젝트가 선택되어 컴파일 된다면, 왼쪽 사이드 바에서 hw4 프로젝트에서 오른쪽 버튼을 클릭하여 “Set as StartUp project”를 클릭 후 다시 컴파일 및 실행해주시기 바랍니다. 설정하여 제출하였는데 이상하게 가끔씩 풀리는 경우가 있는 것 같습니다.

# 3D viewer 사용법

1. 화면 회전 – 마우스 왼쪽 버튼 드래그
2. Dolly in/out – ‘d’, ‘D’
3. Zoom in/out – ‘z’, ‘Z’
4. Seek – 위치에 마우스 오른쪽 버튼 클릭
5. Translation - ‘[‘ (위). ‘/’ (아래). ‘;’ (왼쪽), ‘’’ (오른쪽)
6. Wireframe mode 켜기/끄기 – ‘w’

# 과제 4의 기능 사용법

1. Depth ordering 켜기/끄기 – ‘o’
2. 투명도 증가/감소 - ‘t’, ‘T’
3. 곡면 반투명화 켜기/끄기 - ‘f’ (처리시간이 좀 걸립니다.)

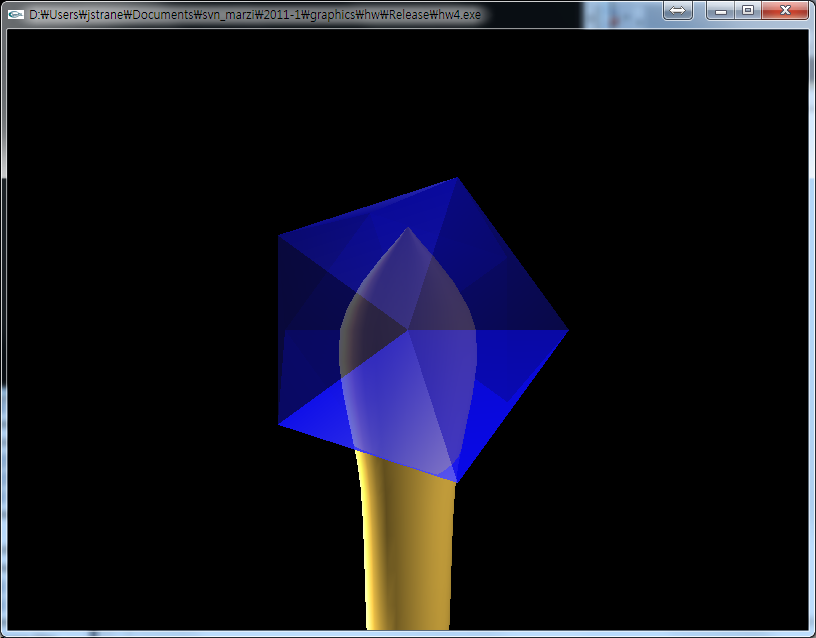


그림 시작 화면

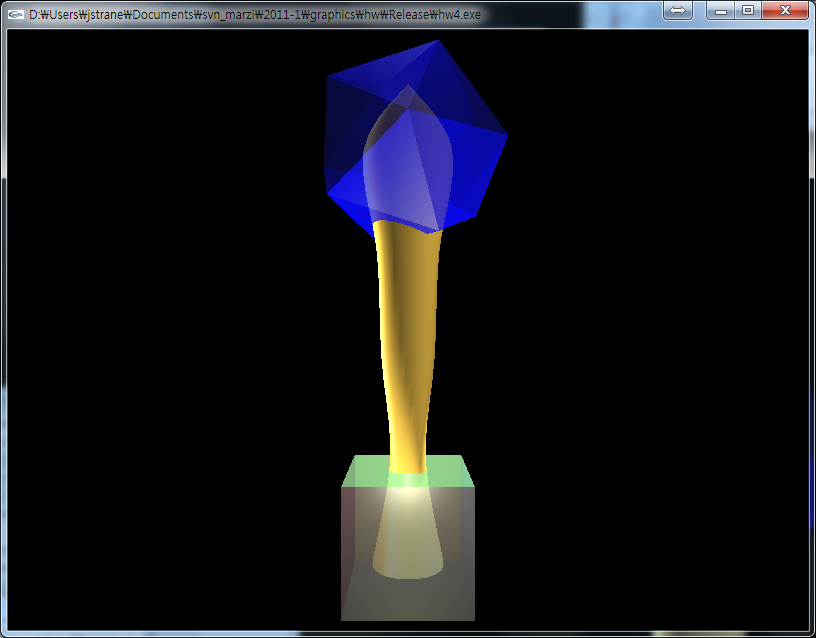


그림 전체 scene의 모습

# 각 표면의 material 속성 설명

이번 과제에서 제가 simulate 해보고 싶었던 material은 금, 은, 옥, 크롬, 플라스틱, 고무입니다.

이 material들을 구현한 방법은 우선 diffuse는 최대한 선택한 material의 색상과 비슷하도록 RGB table과 Photoshop의 color picker 툴을 사용해서 결정했습니다. 그리고 specular의 경우 전반사 될 때 보이는 색깔을 눈과 사진으로 확인하였고 shininess, 즉 n값은 매우 광택이 나는 재질은 highlight가 매우 좁게 반사되므로 값을 높이고 광택이 좀 적게 나는 재질은 값을 낮췄습니다.

Ambient 값은 무광택인 재질들은 0이나 매우 적게 주었고, 광택이 있고 보석류처럼 빛이 투과하는 재질들은 빛이 비추는 부분 외에도 자체 반사나 투과에 의해 밝아지므로 ambient 값을 더 주었습니다. 이와 같이 디지털화된 사진을 color picker 등으로 측정하고 결과를 계속 돌려보면서 눈으로 직접 확인하여 최대한 유사한 coefficient들을 찾아보았습니다. 그리고 추가로 F.S. Hill의 Computer Graphics Using OpenGL, 2nd edition, Prentice Hall (2001)에 나온 값들도 참고했습니다.

아래는 과제에서 cube의 각 면에 설정한 material 6개입니다. OpenGL에서 glMaterial 함수로 material을 설정할 때 사용되는 coefficient값(R, G, B)들로 표기했습니다.

## 크롬

|  |  |  |  |
| --- | --- | --- | --- |
|  | R | G | B |
| Ambient | 0.25 | 0.25 | 0.25 |
| Diffuse | 0.4 | 0.4 | 0.4 |
| Specular | 0.774597 | 0.774597 | 0.774597 |
| Shininess | 76.8 | | |

## 빨간색 플라스틱

|  |  |  |  |
| --- | --- | --- | --- |
|  | R | G | B |
| Ambient | 0.0 | 0.0 | 0.0 |
| Diffuse | 0.5 | 0.0 | 0.0 |
| Specular | 0.7 | 0.6 | 0.6 |
| Shininess | 32 | | |

## 금

|  |  |  |  |
| --- | --- | --- | --- |
|  | R | G | B |
| Ambient | 0.24725 | 0.1995 | 0.0745 |
| Diffuse | 0.75164 | 0.60648 | 0.22648 |
| Specular | 0.628281 | 0.555802 | 0.366065 |
| Shininess | 51.2 | | |

## 광택 내지 않은 은(silver)

|  |  |  |  |
| --- | --- | --- | --- |
|  | R | G | B |
| Ambient | 0.19225 | 0.19225 | 0.19225 |
| Diffuse | 0.50754 | 0.50754 | 0.50754 |
| Specular | 0.508273 | 0.508273 | 0.508273 |
| Shininess | 51.2 | | |

## 옥(jade)

|  |  |  |  |
| --- | --- | --- | --- |
|  | R | G | B |
| Ambient | 0.135 | 0.2225 | 0.1575 |
| Diffuse | 0.54 | 0.89 | 0.63 |
| Specular | 0.316228 | 0.316228 | 0.316228 |
| Shininess | 76.8 | | |

## 노란색 고무

|  |  |  |  |
| --- | --- | --- | --- |
|  | R | G | B |
| Ambient | 0.05 | 0.05 | 0.0 |
| Diffuse | 0.5 | 0.5 | 0.4 |
| Specular | 0.7 | 0.7 | 0.04 |
| Shininess | 10.0 | | |

화면에서 기둥 아래에 자리한 정6면체의 각 면에 위 material들을 지정하였는데, 간단히 위치 설명을 하면 맨 윗면이 옥(jade)이고 바닥면이 노란색 고무입니다. 그리고 옆면들은 정면부터 오른쪽 방향으로 순서대로 크롬, 은, 금, 빨간색 플라스틱입니다. 크롬과 은은 둘 다 색깔이 흰색에 가까운 회색이라 헷갈릴 수 있는데, specular를 봤을 때 크롬이 shininess가 높아서 상이 좀 더 좁고 밝게 맺히는 것으로 구분할 수 있습니다. 참고로 맨 위의 옥색의 경우 옆면들을 비추는 4개의 spotlight와 맨 위의 정20면체를 비추는 2개의 spotlight 때문에 빛을 많이 받아 원래 색보다 조금 더 밝게 표시되고 있습니다.

각 cube의 면에 비치는 lighting의 gradation이 좀 더 자연스럽고 부드럽게 하고 specular의 모양을 좀 더 선명하게 하기 위해서 tessellation, 즉 triangle 쪼개기를 여러 번 하였습니다. ‘w’키를 눌러 wireframe mode로 들어가보면 cube의 각 면들이 꽤 많은 수의 polygon들로 이루어져 있음을 확인할 수 있습니다. Tessellation을 좀 더 많이 해서 더 부드러운 이미지를 얻기 위해서는 “void polygon(int i\_, int j, int k, int l, const string& name)” 함수에서 변수 N의 값을 6 정도로 올려주면 됩니다.

|  |
| --- |
| // Triangle tessellation  const int N = 5 |

그리고 참고로 곡면 반투명화 기능 때문에 곡면의 resolution을 과제 3에 비하여 조금 낮춰두었습니다. 왜냐하면 곡면을 반투명화하면 곡면의 각 polygon들이 모두 평면을 나누는 기준 polygon들이 될 수 있어서 결과적으로 BSP tree의 높이가 꽤 커질 수 있습니다. 따라서 속도를 위해 곡면을 이루는 polygon 수들을 줄이려고 resolution을 조금 낮춰둔 상황입니다. 그래서 빛을 많이 받는 기둥 아랫부분은 specular의 경계에서 polygon 모양들이 이전 과제에 비하여 조금 도드라지는 경향이 있습니다. 이 역시 cross\_resolution, sweep\_resolution 두 변수의 값을 0.03에서 0.01 사이의 값 정도로 낮추어 주면 과제 3에서 본 정도의 부드러운 swept surface를 볼 수 있습니다. 반대로 반투명화 시 렌더링 속도가 느릴 때는 이 값을 0.04 정도로 높여주면 polygon 수가 줄어들어 돌려보기가 보다 수월합니다.

|  |
| --- |
| const float cross\_resolution = 0.03;  const float sweep\_resolution = 0.03; |

위 6가지 중 고무를 제외한 5가지 정도가 Phong shading model로 잘 구현된 material의 예라고 생각합니다.

## Phong shading model로 구현하기 힘든 material의 예 세가지

1. 대리석
2. 사람 피부
3. 카페트 (또는 보풀이 있는 천이나 털가죽)
4. 알루미늄

알루미늄이나 카페트처럼 결이 있는 표면의 경우 같은 지점을 바라보아도 보는 각도에 따라 서로 다른 빛이 반사됩니다. 하지만 Phong shading model은 바라보는 지점이 같고 normal vector와 view vector가 이루는 각이 같으면 normal vector를 기준으로 360도 어느 방향에서 바라보아도 같은 빛을 반사합니다. 따라서 이처럼 결이 있는 모델은 표현이 어렵습니다.

그리고 대리석이나 사람 피부의 경우 표면을 투과한 빛이 안에서 여러 번 굴절되다가 나중에 밖으로 튀어나오게 됩니다. 이처럼 반사가 표면에서만 이루어지는 것이 아닌 재질의 경우 Phong shading model은 내부에서 반사되어 나오는 빛에 대한 고려가 없으므로 역시 표현이 어렵습니다.

# Depth ordering algorithm과 BSP 구현 설명 (extra)

이번 과제에서 반투명 polygon의 처리를 위하여 BSP(Binary Space Partitioning)을 이용한 depth ordering algorithm을 구현했습니다. 알고리즘을 설명하면, 일단 모든 polygon을 triangle로 만들고 나서 polygon을 하나 선택한 다음 그 polygon을 포함하는 평면을 기준으로 앞에 있는 polygon들과 뒤에 있는 polygon들을 다른 두 개의 그룹으로 partition하였습니다. 그리고 앞, 뒤 그룹 각각에서 또 polygon을 선택하여 partition 하는 작업을 recursive하게 반복해나갔습니다.

그 결과로 앞(front)과 뒤(back) 그룹을 child node로 가지는 binary tree를 만들 수 있게 되었는데 이 BSP tree를 traverse하면서 현재 camera의 위치에서 어느 polygon이 멀리 있고 가까이 있는지 ordering 할 수 있었습니다.

그 과정은 tree를 traverse 하는데 현재 node의 pivot polygon의 평면, 그러니까 현재 node의 child node들을 나눈 기준이 되는 평면과 camera의 위치를 비교하여 camera가 이 평면 앞에 있는지 뒤에 있는지를 판단합니다. 만약 camera가 앞에 있다면 child node 중 back을 먼저, front를 나중에 가는 순서로 traverse하면 됩니다. 반대로 뒤에 있다면 child node 중 front를 먼저, back을 나중에 가는 순서로 traverse하면 됩니다. 물론 첫 번째 child node를 방문한 후에는 현재 node와 같은 평면에 있는 polygon들을 표시해주고 두 번째 child node를 방문해야 합니다.

이렇게 camera에서 제일 먼 곳에 있는 polygon들부터 판별할 수 있으므로 traverse 순서 그대로polygon들을 화면에 표시해주면 먼 곳부터 그려지므로 불투명한 polygon과 반투명한 polygon들이 intersect하고 overlap된다 하더라도 비정상적으로 가려지지 않고 정상적인 모습으로 그려지게 됩니다.

이 과정에서 속도를 높이기 위해 몇가지 tweak을 사용하였는데, 우선 공간을 분할하는 기준이 되는 polygon들은 다음 polygon 들에 의하여 분할되지 않으므로, **가장 크기가 가장 큰 polygon 순서로 선택**하여 polygon이 분할되는 것을 최소화 하려고 노력했습니다. 이를 위해 triangle들이 담기 vector를 크기 순으로 sort하고 나서 순서대로 BSP tree를 구성했습니다.

그 다음 불투명 polygon들이 있을 경우 불투명 polygon들끼리는 서로 순서가 크게 상관이 없는 점을 이용하여 최적화를 했습니다. 순서가 상관이 없는 이유는, 불투명 polygon끼리는 순서 상관없이 그려도 depth buffer에 의하여 가장 가까운 polygon들만 그려지고 또한 반투명과 달리 그 뒤의 polygon들은 처리하지 않아도 되기 때문입니다. 따라서 불투명 polygon들이 섞여있는 경우 우선 반투명한 polygon들을 공간을 분할하는 기준 polygon들로 사용했고 불투명한 polygon들끼리는 마치 서로 같은 평면에 있는 것처럼 처리하였습니다. 그 결과 불투명 polygon들은 기준 polygon이 되지 않고 leaf에 한 덩어리로, 마치 같은 평면인 것처럼 같은 level에 있게 되어서 tree의 높이를 획기적으로 줄여 속도를 매우 많이 높일 수 있었습니다.

따라서 정리하면, intersecting polygon 문제는 둘 중 하나가 다른 쪽 polygon을 분할하여 서로 intersecting하는 polygon들이 존재하지 않도록 하여 문제를 해결합니다.

# Lighting 설정 설명

총 7개의 light source를 사용했습니다. 2개는 좌표 (0, 2.5, 2.5)과 (0, 2.5, -2.5)에 있는 spotlight로 맨 위에 있는 파란색 정20면체를 비추기 위하여 사용되었습니다. 그리고 나머지 5개는 맨 아래에 있는 정6면체(cube)를 비추기 위하여 사용하였는데, 옆면에 하나씩 면의 중심보다 조금 위에서 비스듬하게 각 면의 중심을 비추는 spotlight를 배치하였고 바닥면은 면의 중심을 수직으로 아래에서 위로 비추도록 배치하였습니다. 면당 하나씩 배치하여 각 면의 재질을 보기 편하도록 하였고, specular는 광원이 위에서 비추므로 각 면을 아래에서 위 방향으로 쳐다보면 specular를 쉽게 확인할 수 있습니다.

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 2.0 | -5.5 + 1 | 0.0 |
| 0.0 | -5.5 + 1 | 2.0 |
| -2.0 | -5.5 + 1 | 0.0 |
| 0.0 | -5.5 + 1 | -2.0 |
| 0.0 | -5.5 - 2.0 | 0.0 |

각 면을 비추는 spotlight의 좌표는 위 표와 같습니다. 보시면 Y 좌표에서 +1씩 한 것은 면의 중심보다 위에서 아래로 비추기 위하여 더한 값입니다. 그리고 예외적으로 마지막 spotlight만 cube의 바닥면을 비추는 spotlight이기 때문에 면 중심 바로 아래에 위치하기 위하여 Y 좌표에 -2만 했습니다.

빛의 색깔은 diffuse와 specular는 모든 흰색, 즉 (1, 1, 1)을 주었고 ambient는 따로 주지 않고 기본값을 그대로 사용했습니다. 빛의 방향성을 방향 벡터로 나타내면 아래의 표와 같습니다.

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 0.0 | -1.0 | -1.0 |
| 0.0 | -1.0 | 1.0 |

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| -1.0 | -1.0 | 0.0 |
| 0.0 | -1.0 | -1.0 |
| 1.0 | -1.0 | 0.0 |
| 0.0 | -1.0 | 1.0 |
| 0.0 | 1.0 | 0.0 |

위의 spotlight 2개는 사선으로 원점을 향해서 비추고 나머지 아래 5개 중 옆면 4개는 비스듬하게 위에서 아래로, 바닥면 1개는 아래에서 위로 비추고 있습니다. Lighting의 목적은 각 object 들을 명확히 볼 수 있도록 하고 material cube의 경우 각 재질들의 특성이 잘 나타나도록 하는 것이었습니다.

# 복잡한 불투명, 반투명 물체들 (extra)

반투명한 정6면체(cube) 외에 추가적으로 반투명한 정20면체(icosahedron)도 그려봤습니다. Depth ordering에 BSP를 사용했기 때문에 정육면체 외에 다른 복잡한 구조의 반투명한 object를 그려도 순서대로 잘 그려집니다. 만약 BSP가 아니였다면 정육면체보다 복잡합 object들이 존재할 경우 polygon 순서를 계산하기가 배우 어려웠을 것입니다.

그리고 swept surface 역시 반투명화해서 그려봤습니다. 이 기능은 기본적으로는 속도를 위해 꺼져있는데 키보드의 ‘f’키를 누르면 시간이 좀 걸린 후 화면의 swept surface를 포함한 모든 면들을 반투명화시킵니다. 이 역시 ‘t’, ‘T’ 키를 이용하여 투명도를 조절할 수 있습니다.

그리고 위 반투명 물체들은 ‘t’ 키를 계속 눌러서 투명도를 0으로 만들면 전부 다 불투명한 물체가 됩니다. 따라서 결과적으로 복잡한 불투명, 반투명 정6면체, 정20면체, swept surface(곡면)을 구현하였습니다.