**컴퓨터 그래픽스**

**- 설계과제 -**

가상 박물관의 모델링 및 네비게이션

컴퓨터공학부 20143026 강지형

컴퓨터공학부 20143390 김선영

**1.** **개요**

이 프로젝트는 쉐이더 기반의 모던 OpenGL (OpenGL 2.x 이상)을 이용하여 3차원 가상 박물관을 모델링하고 실시간 렌더링을 통해 박물관을 네비게이션 하는 시스템을 설계하고 구현하는 것을 목표로 한다.

3차원 물체의 형상 데이터를 모델 파일로부터 읽어 들인 후, 3차원 변환을 통해 각각의 물체를 가상 세계에 위치시키고, 가상 카메라를 통해 가상 세계를 네비게이션 한다. 그리고 각 물체에 대해 텍스쳐 색상을 입히고 Phong reflection model을 적용하는 것을 기본적인 목표로 하고, 그 외의 추가적인 기능을 구현하도록 한다.

**2.** **설계**

이번 포트폴리오 과제는 강지형(20143026)과 김선영(20143390)이 2인 1조가 되어 과제를 진행하였다. 장면파일 작성 및 물체를 변환하여 가상 세계에 위치시키는 부분, 키보드를 통한 조작, 초기 보고서 작성 및 검토 등은 주로 조원 강지형(20143026)이 맡고, 키보드를 통한 조작, 쉐이더, 텍스쳐 매핑과 렌더링, 보고서 검토 부분은 주로 조원 김선영(20143390)이 맡아서 진행하였다.

**2-1. 가상 박물관의 장면(scene) 데이터의 표현**

가상 박물관에는 여러 가지 물체들로 이루어져 있는데 여기서 존재하는 물체들의 장면 데이터는 ‘scene file’이라는 텍스트 파일로 저장되어 있다. 파일의 형식은 다음과 같이 표현 되어 있다.

**(obj 파일명) (3차원 이동벡터) (y축 회전각) (확대/축소값) (텍스쳐 파일명)**

• • •

< 표1. 장면 파일의 형식 >

각각의 물체는 파일에서 받아 온 장면 데이터를 기반으로 프로그램 내에서 모델링 된다.

**2-2. 가상 박물관 네비게이션**

1. **키보드 조작**

키보드 조작을 통해 가상 박물관을 네비게이션 할 수 있다.

|  |  |
| --- | --- |
| 키보드 | 기능 |
| ← | 카메라 왼쪽으로 이동 |
| → | 카메라 오른쪽으로 이동 |
| ↑ | 카메라 앞으로 전진 |
| ↓ | 카메라 뒤로 후진 |
| A or a | 카메라 y축을 기준으로 왼쪽으로 회전 |
| D or d | 카메라 y축을 기준으로 오른쪽으로 회전 |

< 표2. 키보드 조작 기능 >

1. **충돌 감지**

가상 박물관 네비게이션 시, 장면에 놓인 물체를 뚫고 지나가지 않도록 충돌 감지 기능을 구현한다.

구현하고자 했던 방식은 구의 반지름을 이용하여 충돌을 감지하는 방법이다. 물체가 구로 둘러 쌓여있다고 가정하고, 현재 카메라의 위치와 물체와의 거리를 계산하여 그 거리가 특정 거리 이하면 더 이상 접근하지 못하도록 구현하려 하였다.

|  |
| --- |
| •카메라의 위치 벡터 : (x, y, z)  •물체의 위치 벡터 : (a, b, c) |

< 표3. 구의 반지름 구하는 공식 >

위의 공식을 이용하여 충돌 감지 기능을 구현하려 하였으나, 거리 측정에 대해 정확성이 떨어지고 생각했던 결과가 나오지 않았다.

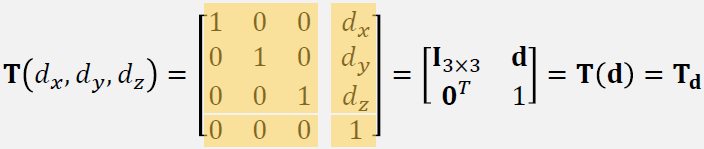
**3.** **설계 주안점**

**3-1. 물체의 변환**

각각의 물체는 OBJ 파일로 구성되어 있으며, 각 물체는 파일로부터 받아온 데이터를 이용하여 모델링 한다.

물체의 변환은 크게 이동(Translation), 회전(Rotation), 확대/축소(Scaling) 이 세가지로 볼 수 있다.

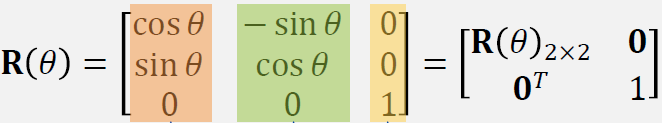
• 물체를 ()의 위치로 이동할 때에는 4x4행렬을 이용한다. 행렬은 다음과 같이 구성되어 있다.

****

< 그림1. 이동(Translation) 행렬의 구성 >

이 수식이 OpenGL상의 코드로는 glTranslate() 함수를 사용한다.

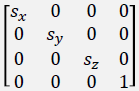
• 물체를 Ɵ 각도만큼 회전할 때에는 다음과 같이 행렬이 구성되어 있다.

****

< 그림2. 회전(Rotation) 행렬의 구성 >

이 수식이 OpenGL상의 코드로는 glRotate() 함수를 사용한다.

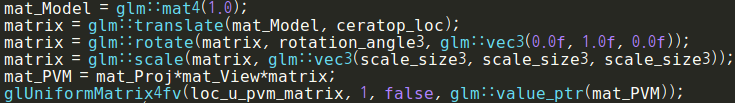
• 물체를 S만큼 크기를 확대, 축소할 때에는 다음과 같이 행렬이 구성되어 있다.

****

< 그림3. 확대/축소(Scale) 행렬의 구성 >

이 수식이 OpenGL상의 코드로는 glScale() 함수를 사용한다.

물체를 합성 변환할 때, 우리가 일반적으로 계산할 때에는 원점에서 scale을 먼저 해 준 다음, rotate와 translate을 하게 되지만, 프로그램 상에서는 반대로 적용해 주게 된다. 프로그램 상에서는 모델을 먼저 변환을 해 준 뒤, 모델의 위치를 카메라 위치 중심으로 계산하여 쉐이더로 보내준다.

****

< 코드1. 물체의 변환에 관한 부분 >

**3-2. 물체의 표현**

각각의 물체는 빛(light)과 재질(material)의 상호 작용을 통한 색상과 텍스쳐 이미지 정보를 통해 렌더링 된다. 이 프로젝트에서는 텍스쳐 파일이 명시 되어 있는 물체를 사용하여, 텍스쳐 색상을 diffuse reflection coeffcient로 활용하여 Phong reflection model을 구현한다.

1. **빛과 재질**

Phong reflection model에서 빛은 주변광(ambient), 난반사광(diffuse), 정반사광(specular)으로 이루어져 있고 각각의 빛이 따로따로 주어진다. 먼저, 주변광(ambient)은 주위에서 들어오는 작은 빛들로, 다음과 같이 계산한다.

**** 여기서 L은 ambient상의 빛이고, 카파값은 이 빛을 얼마나 적용할지에 대한 값이 들어간다.

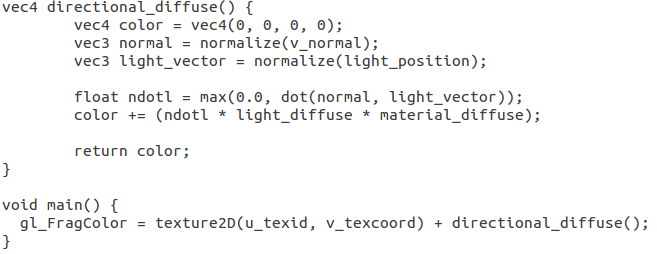
난반사광(diffuse)은 빛이 부드러우면서도 거친 면에 닿아 반사되는 빛들로, 전체적으로 밝은 느낌이 들도록 한다. 난반사광의 계산은 다음과 같다.

 여기서 L은 diffuse상의 빛이고, 카파값은 이 빛을 얼마나 적용할 지에 대한 값이, l은 빛이 들어오는 방향, n은 노말값이다. 여기서 max를 해주는 이유는 빛이 반대 방향에서 들어오는 경우를 방지하기 위해서이다.

정반사광(specular)은 빛이 노말값을 중심으로 들어오는 각도와 같은 각도로 반사되는 빛으로, 다음과 같이 계산한다.

**** 여기서 L은 specular상의 빛이고, 카파값 역시 이 빛을 얼마나 적용할 지에 대한 값이다. 여기서, r과 v의 거리가 가까울수록 빛을 더 반짝거리게 할 수 있다.

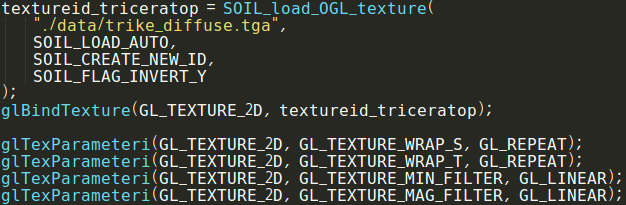
Phong reflection model에서는 이 세가지 빛을 실시간으로 더해서 계산한다. 그러나, 우리 프로그램에서는 diffuse에 대해서만 계산한다.

****

< 코드2. 쉐이더에서 구현한 Phong reflection model >

1. **텍스쳐 매핑**

각각의 모델은 obj 파일로 이루어져 있고 텍스쳐 파일을 가지고 있는데 이와 함께 mtl 파일도 가지고 있다. 각 모델에 픽셀 단위로 텍스쳐 좌표값을 매핑 시키게 되면, 모델에 텍스쳐가 쓰여지게 된다. 프로그램에서 구현한 텍스쳐 매핑 코드를 보며 자세히 설명하도록 한다.



< 코드3. main에서 구현한 텍스쳐 매핑 부분 >

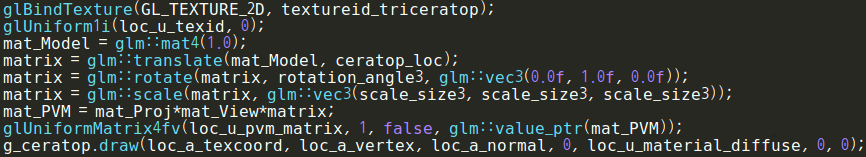
위와 같이, 각 물체마다 텍스쳐를 연결 시켜준다.

텍스쳐를 연결할 때에는, 텍스쳐를 연속적으로 연결시키는 GL\_REPEAT과 텍스쳐의 끝부분을 확장시키는 GL\_CLAMP\_TO\_EDGE가 있다.

텍스쳐를 적용했을 때 생기는 문제들(magnification, minification 등)은 샘플링을 몇 번 하느냐에 따라 결과가 조금씩 다르게 나타나는데 여기에는 한 번 샘플링하는 Nearest sampling과 여러 번 샘플링하는 Linear sampling이 있다.

여기서 우리는 텍스쳐를 연결하여 GL\_REPAT과 GL\_LINEAR를 사용하여 나타냈다.

각각의 물체마다 텍스쳐를 연결 시킨 다음, 물체를 변환시킬 때 함께 텍스쳐를 연결하여 쉐이더로 보내게 된다.

< 코드3-1. main에서 구현한 텍스쳐 매핑 부분 >

그리고 쉐이더에서는 앞서 계산했던 Phong reflection model의 diffuse와 함께 칼라값을 계산하여 최종적으로 화면에 나타내어 준다.



< 코드4. Fragment shader에서 구현한 텍스쳐 매핑 부분 >

**4.** **결과 화면**

