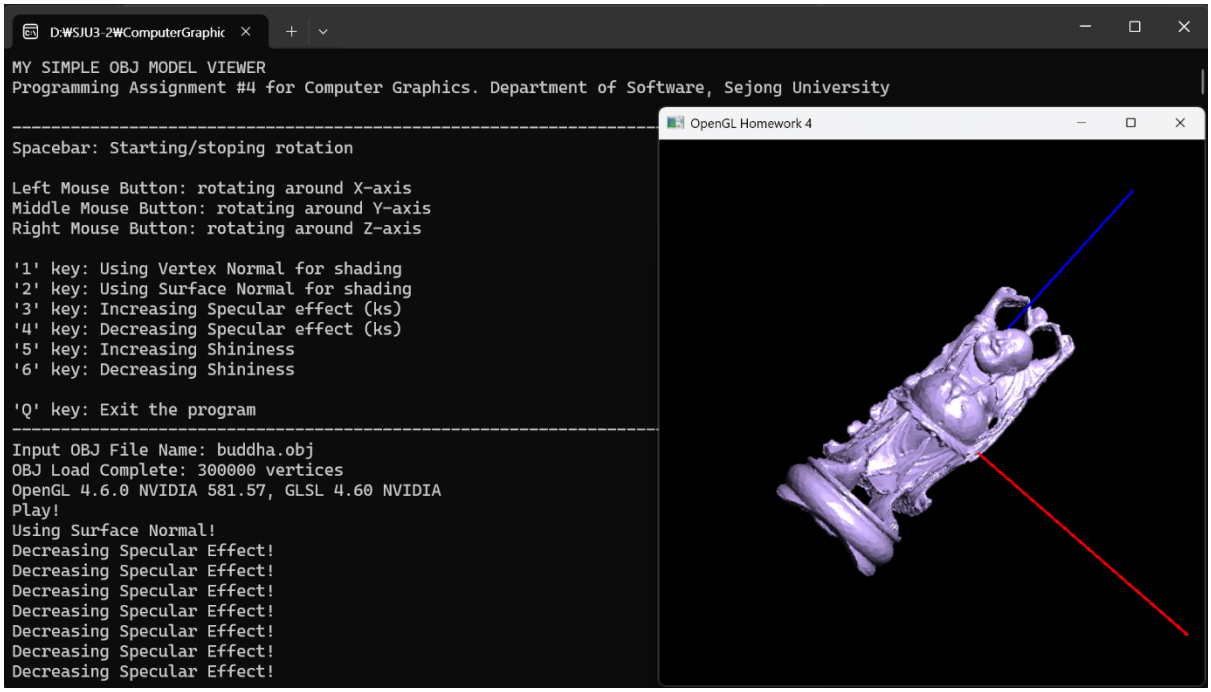
Homework#4: Obj Model Viewer만들기에 관한 보고

21011794 황재동(2025.11.29)

1. 개요:

본 과제는 3차원 컴퓨터 그래픽스에서 가장 범용적으로 사용되는 파일 포맷잇 OBJ 파일을 Parsing하여 읽어들이고 이를 OpenGL 환경에서 시각화하는 뷰어 프로그램 구현하는 것을 목표로 한다. 단순히 모델을 띄우는 것을 넘어 모델의 크기와 위치를 자동으로 정규화하여 화면 중앙에 배치하고 Phong Illumination 모델을 적용하여 사실적인 조명 효과를 구현한다. 또한 키보드와 마우스 입력을 통해 쉐이딩 모드(Phong Shading과 Flat Shading)변경, 조명 파라미터(Specular, Shininess) 조절, 그리고 축 기준 회전 제어 기능을 구현하여 상호작용 가능한 뷰어를 완성한다.



2. 구현 방법 구상

2-1. OBJ 파일 파싱 및 데이터 구조 설계

OBJ 파일은 정점 정보와 면 정보로 구성된다. 이를 효율적으로 처리하기 위해 MyOBJModel 구조체를 정의한다. Phong Shading과 Flat Shading을 시간 지체 없이 실시간으로 전환하기 위해 하나의 정점 데이터에 대해 Face Normal과 Vertex Normal 두가지 데이터를 미리 계산하여 각각 저장을 해주고 실행 중에 데이터를 교체하는 방식을 사용해야 한다.

2-2. 자동 정규화(모델의 Pivot과 Scale 정하기)

불러오는 OBJ 모델마다 좌표계와 크기가 제각각이기 때문에 이를 그대로 그리면 화면에 너무 크거나 작게 보일 수 있다. 따라서 모델을 불러올때 모든 정점을 반복문으로 돌아 최솟값과 최댓값을 구해 경계를 계산한다. 이들의 중점을 원점으로 이동시키고 화면 크기에 맞게 스케일을 조정하는 전처리 과정을 거친다.

2-3. Phong Illumination 및 모델 회전

Specular와 Shininess값을 조정하는 것은 main.cpp에서 uniform 변수를 선언하여 쉐이더와 연결하여 키보드 입력이 들어올 때 그에 맞게 반영해주면 된다. 값이 과하게 설정되는 것을 방지하기 위해 clamp를 해주면 더 자연스럽게 보일 수 있을 것이다. Ks값은 반사광의 강도를 조절하게 해주는데 낮으면 matte한 표면이 되어 하이라이트가 거의 안보이고 거친 질감을 줄것이다. 반대로 높을수록 광택이 나는 표면효과를 주고 밝고 선명한 하이라이트를 주는 금속 느낌의 표면을 줄 수 있다. Shininess값은 하이라이트의 크기와 집중도를 조절하는 변수로 값이 낮으면 넓게 펴진 하이라이트와 부드러운 반사를 줄 수 있고 반대로 값이 높을 수록 작고 날카로운 하이라이트와 집중되고 선명한 반사를 준다.

마우스 입력을 통해 회전축을 선택하고 매 프레임마다 해당 축에 대한 회전 행렬을 생성한다. 회전이 뚝뚝 끊기지 않고 자연스럽게 이어지도록 하기 위해 현재의 회전상태를 저장하는 전역행렬에 새로운 회전을 누적 곱하는 방식을 적용한다. 또한 뷰어에 나와있는 회전축도 회전을 해야하니 동일한 회전 행렬을 적용하여 시각적으로 동기화한다.

3. 구현 단계

3-1. OBJ 파일 파싱 및 데이터 로드

가장 먼저 프로그램이 실행되면 콘솔 창을 통해 사용자로부터 OBJ 파일의 이름을 입력받는다. MyOBJModel 클래스의 ReadOBJ 함수 내부에서는 텍스트 기반의 OBJ 파일을 줄 단위로 읽어 들인다. 파일이 존재하지 않을 경우 에러 메시지를 출력하고 무한 반복문을 통해 올바른 파일명이 입력될 때까지 대기하도록 예외 처리를 구현했다. 파일 파싱 과정에서는 문자열의 첫 글자를 판별하여 v로 시작하는 경우 실수형 데이터 3개를 읽어 임시 정점 배열인 tmpVertex에 저장하고, f로 시작하는 경우 정수형 인덱스 3개를 읽어 tmpFrag 배열에 순차적으로 저장한다. 이때 OBJ 파일 포맷은 인덱스가 1부터 시작하므로 이를 0부터 시작하는 배열 인덱스에 맞추기 위해 1을 빼서 저장해준다.

3-2. 모델 정보 생성(적절한 법선 벡터 계산)

파일 불러오기가 완료되면 MakeModelData 함수를 호출하여 렌더링에 필요한 최종 모델 정보를 생성한다. Phong Shading을 구현하기 위해서는 각 정점에서 뻗어 나오는 법선 벡터를 저장해야하므로 먼저 모든 삼각형을 순회하며 두 변의 벡터 외적을 통해 면의 법선 벡터를 계산한다. 이렇게 계산된 면의 법선 벡터를 해당 면을 구성하는 세 개의 정점 인덱스에 해당하는 accumulatedNormals 배열에 누적하여 합산한다. 이렇게 하나의 정점에 이것이 구성하고 있는 면의 모든 법선 벡터을 누적해주면 복잡한 과정없이 빠르게 정점의 법선 벡터를 계산할 수 있다. 모든 면에 대한 순회가 끝나면, 누적된 법선 벡터들을 normalize 함수를 통해 정규화함으로써 각 정점의 법선 벡터를 완성한다. 동시에 각 면의 고유 법선 벡터도 별도로 저장하여, 1번 키와 2번 키 입력에 따라 부드러운 음영과 각진 음영을 실시간으로 교체할 수 있도록 데이터를 구성했다. 최종적으로 위치, 법선, 색상 정보를 담은 MyOBJVertex 구조체 배열을 생성하여 쉐이더로 보낸다.

3-3. 모델 자동 스케일링 및 피벗 계산

랜덤한 크기를 가진 모델을 화면 중앙에 적절한 크기로 출력하기 위해 FindPivot 함수에서 피벗을 찾는 과정을 수행한다. 불러온 모델의 모든 정점을 순회하며 X, Y, Z축 각각에 대한 최솟값(minPos)과 최댓값(maxPos)을 찾아낸다. 이 두 값의 평균을 계산하여 모델의 중심인 피벗을 결정하고 가장 긴 축의 길

이를 기준으로 스케일 팩터를 역으로 계산하여 모델이 뷰어 내에 정규화 돼서 들어오도록 조정한다. 이 계산된 값들은 렌더링 시 모델 행렬을 구성할 때 모델을 원점으로 이동시키고 크기를 정규화하는 데 사용된다.

3-4. OpenGL 버퍼 생성 및 쉐이더 변수 연결

데이터 구축이 완료되면 InitModel 함수에서 glGenVertexArrays와 glGenBuffers를 호출하여 VAO와 VBO를 생성한다. 구축된 정점 데이터를 glBufferData를 통해 GPU 메모리로 전송한 뒤, SetPositionAndColorAttribute 함수를 통해 쉐이더와의 연결을 설정한다. 이때 glVertexAttribPointer를 호출하여 정점 배열의 메모리 레이아웃을 정확하게 연결함으로써, 데이터가 쉐이더로 올바르게 전달되도록 구현했다.

3-5. 카메라 및 조명 제어

display 함수에서는 먼저 myLookAt 함수를 통해 카메라를 (0, 0, 3.5) 위치에 고정하고 원점을 바라보게 설정하여 뷰 행렬을 생성한다. 모델 변환 행렬은 회전 행렬과 기본 모델 행렬(Scale * Translate)을 곱하여 구성하는데, 이를 통해 모델이 원점에서 제자리 회전하는 효과를 구현했다. 조명 효과를 위해 프래그먼트 쉐이더에 정의된 uShininess와 uSpecularEffect 유니폼 변수에 glUniform 함수를 사용하여 값을 전송한다. 이를 통해 키보드 입력에 따라 물체의 shininess와 specular 강도가 즉각적으로 변화하는 것을 시각적으로 확인할 수 있게 했다.

3-6. 사용자 상호작용 및 누적 회전 시스템

사용자의 마우스 및 키보드 입력을 처리하기 위해 콜백 함수를 활용했다. MyMouse 함수에서는 마우스 클릭에 따라 전역 변수 g_AxisMode를 변경하여 회전축을 X, Y, Z 중 하나로 선택한다. 회전의 연속성을 보장하기 위해 idle 함수에서는 현재 선택된 축에 대한 델타 회전 행렬을 생성하고, 이를 기존의 회전 행렬 g_RotationMat에 누적 곱하는 방식을 적용했다. 이 방식을 통해 회전축을 변경하더라도 물체가 튀지 않고 부드럽게 이어서 회전하며, 로컬 좌표계가 아닌 월드 좌표계를 기준으로 회전하는 직관적인 조작감을 구현했다. 또한 좌표축 그리기 함수에도 모델과 동일한 회전 행렬을 인자로 전달하여, 물

체와 좌표축이 동기화되어 회전하도록 처리했다.

4. 시행착오 및 해결과정

4-1. Vertex Normal 계산 알고리즘 최적화

자연스러운 음영을 갖는 Phong Shading을 구현하기 위해서는 각 정점에서의 법선 벡터들이 필수적이다. 초기 구현 단계에서는 특정 정점의 법선 벡터을 구하기 위해 전체 면 데이터를 이중 루프로 순회하며 해당 정점을 포함하는 모든 면을 찾아 법선을 합산하는 방식을 사용했다. 이 방식을 사용함과 동시에 이것을 '1', '2'키가 입력될때마다 다시 계산하는 방식도 사용하니 너무 느려서 사용이 불가능한 프로그램이 되어 버렸다. 삼각형이 별로 없는 큐브 같은 모델은 문제가 없었지만 삼각형이 많은 모델일수록 연산량이 급격히 늘기 때문이다. 일단 이 문제를 해결하기 위해 법선 벡터들을 모델이 갖고 있게끔 저장하는 방식으로 변경했다. 법선 벡터들을 구워놓으면 상황에 맞춰 쉐이더에 보내지는 법선 벡터만 바꿔주면 된다. 그 다음 정점의 법선 벡터을 계산하는 방식도 수정했다. 리스트를 한 번만 순회하여 정점에 벡터를 누적하는 방식을 사용했다. 각 면을 순회할 때 계산된 면의 법선 벡터을 해당 면을 구성하는 세 정점의 accumulatedNormals에 즉시 더해준다. 모든 면에 대한 순회가 끝나면 각 정점에 누적된 벡터를 정규화함으로써 시간 복잡도를 낮추면서 자연스러운 결과를 갖고 오는 정점 법선 벡터을 구할 수 있었다.

4-2. 모델 크기 불일치 및 피벗 보정

OBJ파일을 테스트하던 중 다른 모델들은 화면의 정중앙에 잘 표시가 됐으나 Cube 모델은 화면에 중앙에 오지않고 우측으로 치우친 상태로 표시가 되는 문제가 있었다. 아마 OBJ 파일 내부 정점 좌표들이 원점을 기준으로 정의되어 있지 않고 특정 좌표를 기준으로 생성되어서 그럴것이라고 생각했다. 이를 해결하기 위해 모델을 불러오는 단계에서 FindPivot 함수를 구현하여 모든 정점의 X,Y,Z 좌표의 최댓값과 최솟값을 찾았다. 이를 통해 모델의 중점을 계산했고 최초 출력시 해당 값만큼 Translate을 적용하여 모델을 강제로 중앙에 오게끔 했다.

4-3. Shader 데이터 연동 오류 및 뷰어 시점 보정(사소한 문제들)

쉐이더 프로그래밍 과정에서 물체의 색상이 의도한 흰색으로 나오지 않고 무지개색으로 출력되거나 회전하면 색이 이상해지는 현상이 발생했다. 디버깅 결과 쉐이더로 데이터를 넘기는 과정에서 변수명이 달라 vColor에 속성이

제대로 연결되지 않아서 발생한 문제였다. 그리고 분명 제대로 된 회전 행렬값을 모델과 좌표축에 줬음에도 불구하고 local 좌표에서 회전하는 것처럼 보였는데 LookAt함수에서 카메라의 위치가 (2, 2, 2)로 대각선에 놓였기 때문이었다. 카메라가 너무 중점으로 가도 모델이 클리핑돼서 안보이기 때문에 적절한 위치를 찾아 배치했다.