Computer Graphics Project#3

2023-15725 곽민서

1. Introduction

이번 프로젝트는 주어진 obj파일을 통해 정점과 관련된 정보를 읽어들인후, 물체를 렌더링, 그리고 이제 광원을 추가하여 쉐이딩을 구현하고, 텍스쳐를 입히는 것을 목표로 한다. Phong Shading과 Gourang Shading, 두 종류의 Surface Shading의 계산 방식을 이해하고, 또 다양한 텍스쳐파일을 읽어들여서 좌표 변환을 하여 읽어서 기존의 Phong Shading에 더하는 구현을 하였다.

2. Implementation

이번 프로젝트를 진행하면서 스켈레톤 코드에서 고친 파일은 다음과 같다. :

-main.py

-shader.py

-render.py

-primitive.py

Main.py에서는 주어진 obj파일을 읽어서 vertice 정보, indices 정보, vertex normal, vertex texture 등을 파싱하고 render에 넘겨줘서 렌더링이 가능하게 한다. 이때, render의 Window클래스의 add_line함수를 통해 와이어를 추가하고, add_polygon을 통해 obj을 추가한다.

Shader.py에는 phong shading, gourang shading, textured phong shading, textured normal phong shading을 GLSL 언어로 구현한 쉐이딩 프로그램이 vertex source와 fragment source로 구현되어있다. 정점 단위로 연산하는 과정은 vertex soucr에서, 픽셀 단위로 연산하는 과정은 fragment sourc로 구현되어 있다.

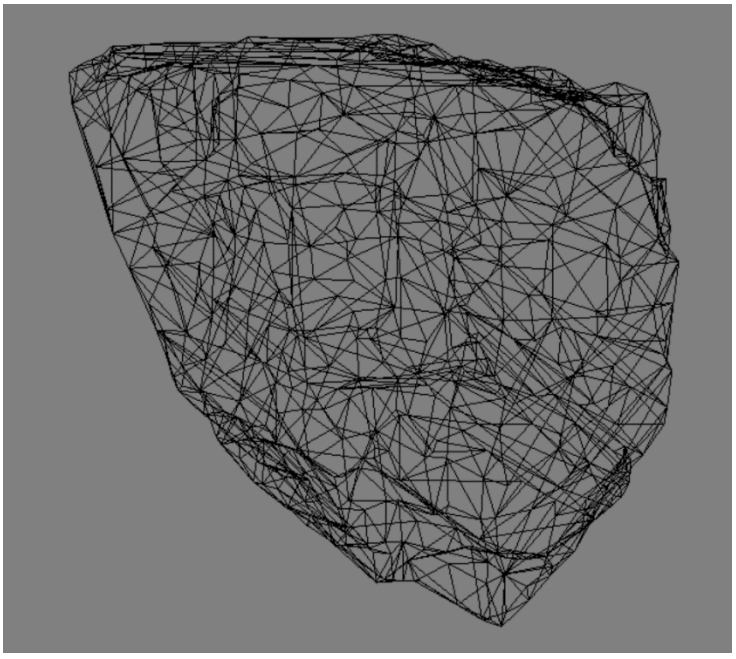
Render.py는 main.py에서 정점과 인덱스 등의 정보를 받아서 이를 렌더링 해주는 Window 클래스의 add_polygon 함수가 있다. Add_polygon 함수는 indexed_vertex_list를 생성할 뿐만 아니라, shading의 종류에 맞게 GLSL에 넘겨줄 uniform 데이터를 연결해주거나. K_a, K_s, K_d, Intensity, Ambient Light, Shineness 등의 각종 상수를 포함하고 있다. 각 Shading 종류마다 해줘야할 동작이 다르기에, 각 shading 마다 1개씩 파일, 총 4개의 파일(renderGouraud.py,

renderNormalTexture.py, render.Phong.py, renderTexture.py)로 나누어 구현하였다.

Primitive.py에선 render.py의 Window 클래스가 Batch에 attatch할 때 이용하는 CustomGroup의 Mesh관련 클래스를 이용하는데, 이 CustomGroup은 디폴트 쉐이딩 프로그램을 사용하기에, add_polygon에서 요구하는 쉐이딩 종류에 따라 각 쉐이딩 프로그램을 만들어준다. 또한, 텍스쳐에 관련해선, 사용하기 위해서 사용할때마다 바인딩이 필요한데, 이를 위해 CustomGroup을 상속하는 CustomTextureGroup을 만들었고, 이 클래스는 add_texture을 통해 여러 텍스쳐를 클래스에 추가시키고, set_state로써 쉐이딩이 사용될 때 같이 모든 텍스쳐들을 바인딩시킨다.

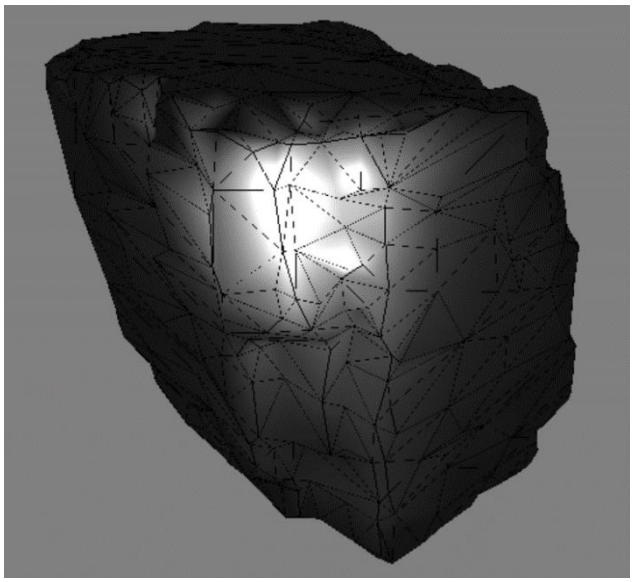
1) Step 0 – Basic

Basic은 wire frame을 구현하는 것이다. Wire frame은 GL_LINE들로 이루어져있는데, 이때 GL_LINE에 정점과 인덱스를 넘겨주기 위해 Window 클래스의 add_line 메소드를 이용한다. 넘겨주는 인덱스는 obj파일을 읽었을 때 받은 인덱스와 다르다. 이 인덱스는 삼각형 폴리곤의 인덱스이기 때문이다. 삼각형의 인덱스를 세 선분을 이루는 인덱스로 변환해서 넘겨준다. Add_line함수는 color이나 shading 없이 GL_LINE으로만 추가한다.



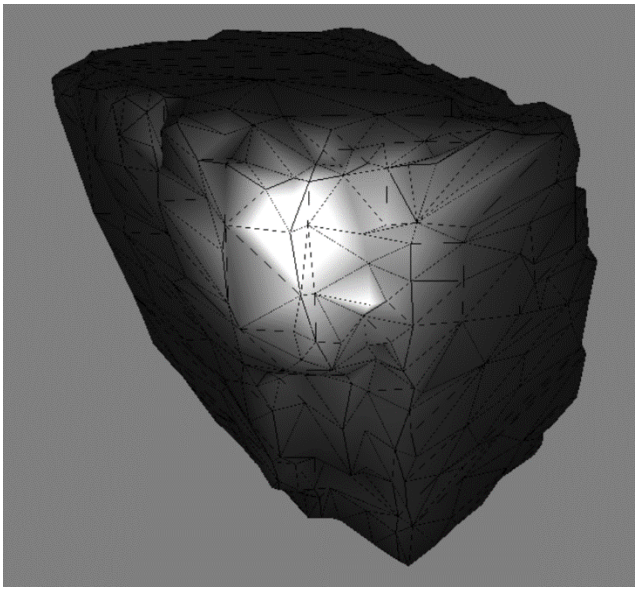
2) Step 1- Phong Shading

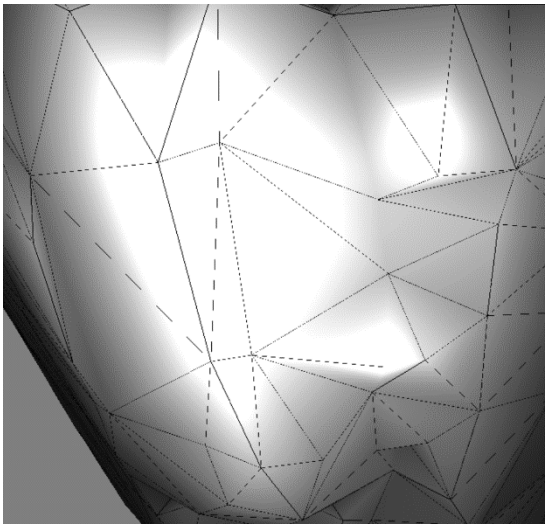
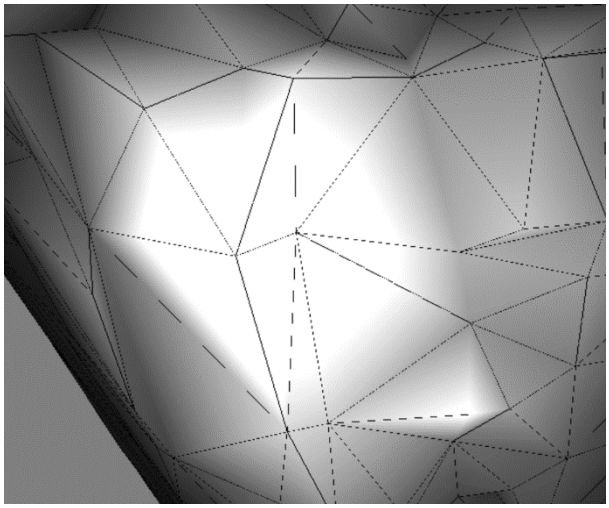
Phong Shading은 폴리곤의 각 픽셀에 대해 광원, 시점, 법선, 그리고 몇몇 상수값의 연산을 통해 픽셀에 할당된 색을 연산한다. 각 픽셀에 대한 연산을 수행하는 것은 fragment_source_phong에 구현되어 있다.



2-1) Gouraud Shading

Gouraud Shading은 모든 픽셀에 대한 연산이 아닌, 각 정점에 대한 연산만으로 쉐이딩 하는 방식이다. 그래서 연산 과정은 vertex_source_gouraud에 구현되어 있다. 연산이 적은 만큼 효과가 좋으나, 폴리곤의 경계에서 불연속적으로 변하는 색깔로 인해 Mach Band Effect가 발생하는 것을 확인할 수 있었다.

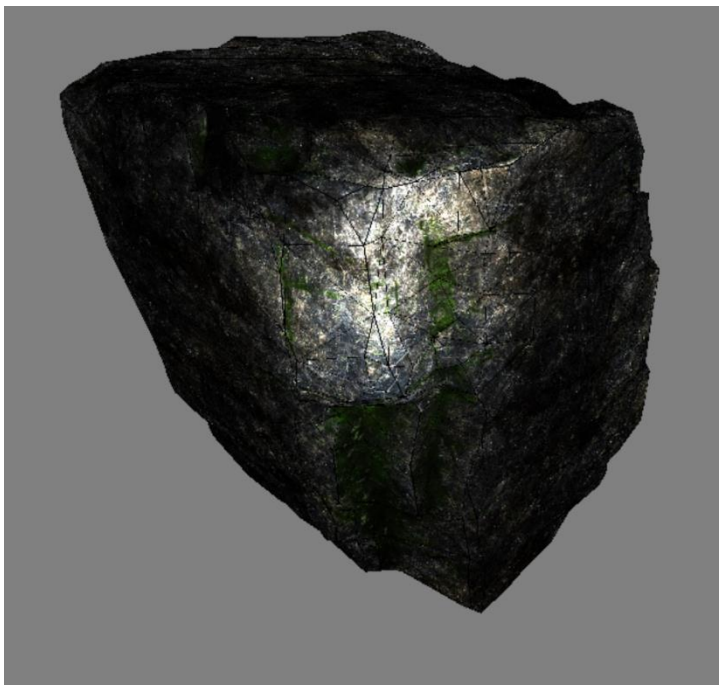


- Comparison : 좌측은 Phong Shading, 우측은 Goraud Shading이다. 좌측은 빛이 비춰지는 경계면이 부드럽지만, 우측은 매끄럽지 못한 것을 볼 수 있다.

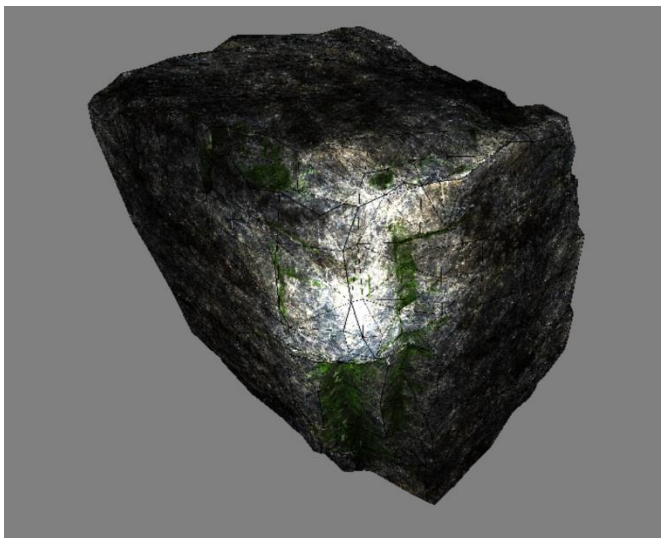
3) Step 2 – Texture Shading

Texture Shading은 Phong Shading에선 상수로 썼던 값을 이젠 픽셀 단위로 변화하는 값으로 차용한다. K_a는 Mixed_Ao.jpg와 Base_Color.jpg를 곱한 값으로, K_d는 Base_Color.jpg으로, K_s는 Specular.jpg의 값이 필요하다. 그러나, 색상은 Vec3이지만, 이미지의 위치는 Vec2의 x,y로 저장되어 있다. 다행히, 이 x, y를 색상의 vec3으로 바꾸어주는 것은 GLSL의 내장함수 texture가 texcoords, 텍스쳐의 x, y값을 토대로 값을 얻을 수 있게 해준다. Shineness는 한 색 정보만 저장되어있는 roughness의 데이터를 토대로 연산하여 적용하였다.



4) Step 3 – Texture Normal Mapping

마지막으로, Texture shading에서 각 vertex단위로 연산을 하였던 normal을 이젠, 그 정보가 기록되어있는 Normal.jpg 데이터를 이용하여 normal 연산시 참고한다. 마찬가지로 texture함수를 이용하여 vec3을 읽어서 normal에 사용한다. 단순한 Texture Shading 보다도 정교한 Normal 정보를 사용하기에, 표면의 울퉁불퉁함이 잘 구현된 것을 확인 할 수 있다.



3. How to Use

각 Shading 모드를 바꾸는 법은 main.py에서 Window를 Import해오는 파일을 바꿔줌으로써 가능하다. Main.py 최상단엔 주석이 달린 from import 절이 있는데, 원하는 쉐이딩을 적용하기 위해 다른 모든 것은 주석을 지워주고, 해당 파일만 import해오면 된다. 그뒤, main.py를 실행하면 원하는 3d 구조물을 볼 수 있다.

작동의 편의성을 위하여 키보드 움직임을 구현해놓았다. 다른 건 고정되고, 카메라만 움직인다. 이때, 카메라의 움직임이 phong계산시 view vector에 영향을 주진 않는다. WASD로 전후좌우로 움직이며, QE로 좌회전, 우회전, RF으로 상회전, 하회전, TG로 확대,축소가 가능하다.

4. Appendix – 계산시 상수값

-광원의 위치 : (7, 7, 7)

-시점의 위치 : (10, 10, 10)

-I_intensity : 100 -> 점광원의 세기

-I_ambient ; 2 -> 앰비언트 빛의 세기

-Shineness : 4 (Phong Shading, Goraud Shading)

-K_a : (0.1, 0.1, 0.1)

-K_d : (0.5, 0.5, 0.5)

-K_s : (0.8, 0.8, 0.8)

-Shineness(Texture) : 1 / (a*roughness + b)

-> a = 0.1, b = 0.1

5. Additional Reference

-https://learnopengl.com/Getting-started/Shaders : OpenGL / GLSL Information