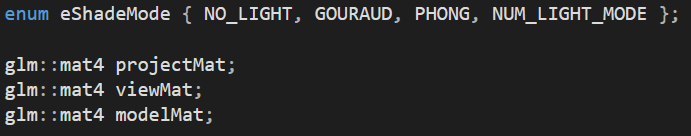
**컴퓨터 그래픽스 프로젝트 2 – Lighting and Texture**

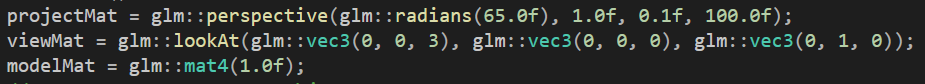
**20182931 신석경**

**기본적인 코드를 설명하기 전에, 아래에서 설명한 부분 이외의 것은 앞선 “컴퓨터 그래픽스 프로젝트 1”의 코드와 동일함을 언급한다.**

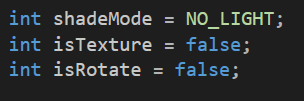
1. **기본 코드 구성**



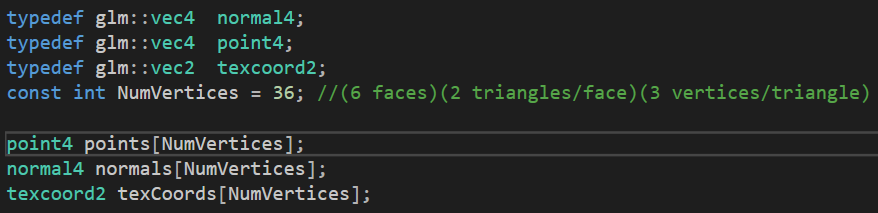
먼저 ShadeMode를 결정하는 열거형을 선언하였다. 이는 Shader에 uniform으로 들어가 이 입력에 따라 모델의 illumination을 다르게 계산한다. 그리고 3가지 mat4 변수를 선언했는데, projection matrix를 나타내는 projectMat, view matrix를 나타내는 viewMat, model transform matrix인 modelMat이다. 2차원 framebuffer에서 위치를 결정할 때, projection matrix \* view matrix \* model matrix \* vertex position으로 결정하기 때문에 따로 따로 변수로 선언해주었다.



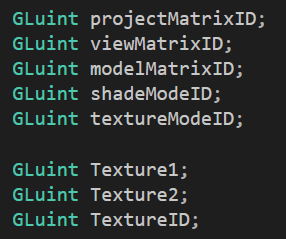
설정한 matrix들이다. 먼저 projection은 fov 값을 65도, 가로 세로 비율을 1:1, near 0.1, far 100.0 으로 설정해주었다. 그리고 view는 (0, 0, 3)에서 (0, 0, 0)을 바라보고, up vector는 y축 방향이다. model은 기본적으로 mat4(1.0f)를 설정해주었다.



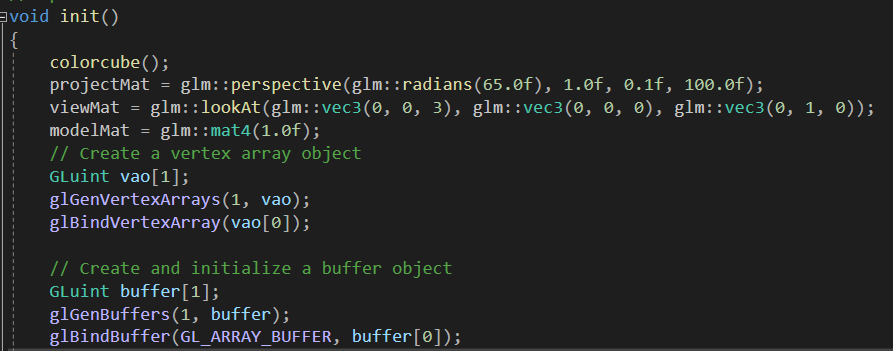
일단 기본적인 세팅이다. shadeMode는 lighting이 없는 것으로, texture도 없는 것으로, Rotate도 하지 않는 것으로 설정하였다.



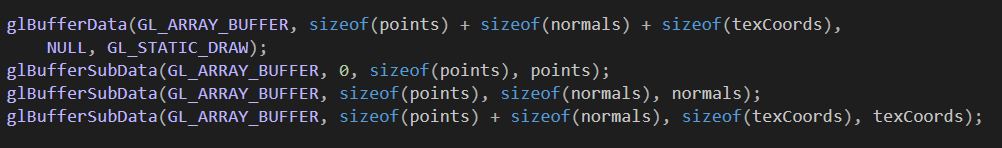
이제 normal vector, points, texcoord에 대한 정의이다. 각 points가 그려지는 순서에 따라 normal vector와 texcoord를 저장할 것이다. 그래서 동일하게 36개의 배열을 선언하였다.



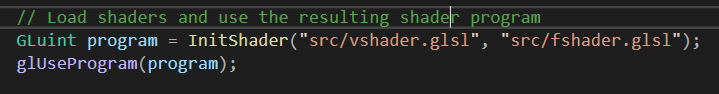
이제 각각의 shader의 uniform location을 저장하는 변수들을 선언했다. 그리고 Texture에 관한 변수를 선언하였다. 이 모델에서는 Texture를 2개 사용하기 때문에 Texture를 저장하는 변수 2개와 uniform location을 저장하는 변수 1개를 선언하였다.



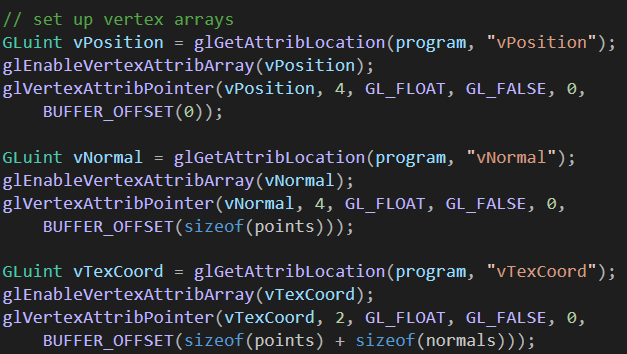
Init 함수이다. 먼저 colorcube()를 호출해 각각의 vertex를 정의해준다. 이 과정을 뒤에 자세하게 설명하도록 하겠다. 그리고 앞에서 설명한 projection matrix와 view matrix를 설정해주고, model matrix를 기본으로 설정한다. 그리고 vertex array object를 설정하고 buffer object를 설정해준다.



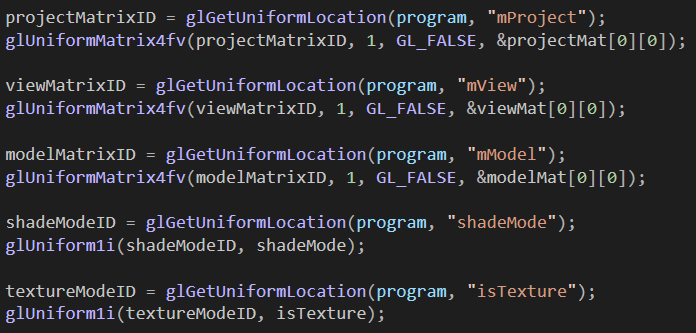
bufferData 생성이다. 먼저 points, normals, texCoords 만큼의 size를 할당한다. 그리고 각각의 위치에 points와 normals와 texCoords를 할당해준다.



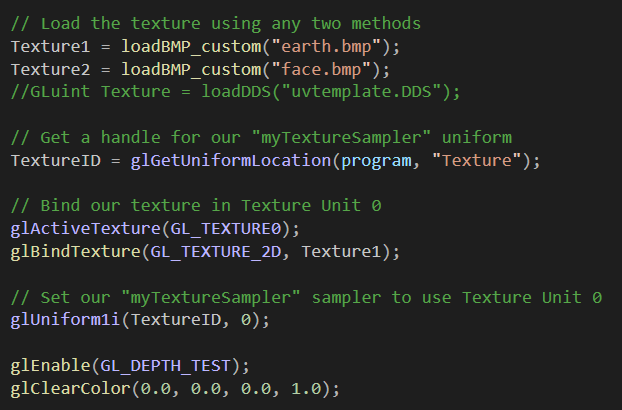
그리고 shader를 실행한다.



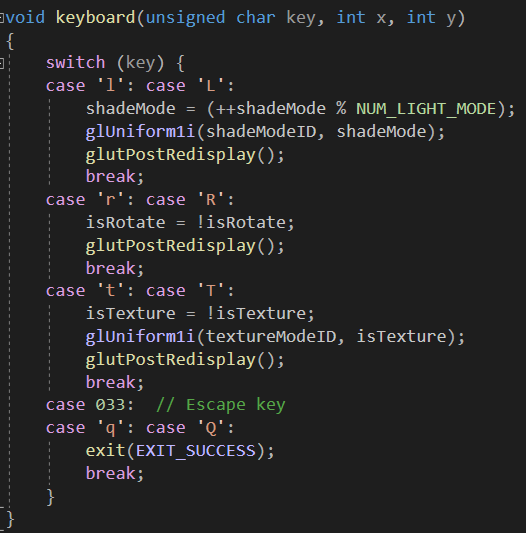
Shader에 설정을 해준다. 먼저 vertex shader의 vPosition에 해당하는 ID를 저장하고, 그 위치에 vertex attribute pointer를 설정해준다. 비슷한 방법으로 vNormal에 해당하는 ID를 저장하고, sizeof(points) 만큼의 offset 뒤에 normal vectors를 설정해준다. 비슷한 방법으로 vTexCoord에 해당하는 ID를 저장하고 points + normals의 size 만큼의 offset을 가지고 할당해준다.



Uniform value의 값 설정이다. 먼저 projection matrix를 mProject value에 할당하고, 나머지도 비슷하게 설정해준다. 이때, projection matrix, view matrix, model matrix는 mat4 value이기 때문에 uniform matrix 4fv로 설정하고, shadeMode, isTexture 같은 것은 int 형이기 때문에 Uniform 1i로 설정해준다.

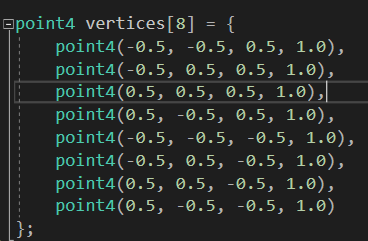


그 다음 Texture을 가져온다. 이 과제에서는 2가지의 Texture를 사용하기 때문에 각각 Texture1, Texture2로 가져온다. 그리고 shader에서 Texture에 해당하는 location을 가져오고, GL\_TEXTURE0로 Texture1을 할당해주고, GL\_TEXTURE1으로 Texture2를 할당해준다. 그리고 현재 Texture를 0번째로 설정해준다. 그리고 opengl을 시작한다.

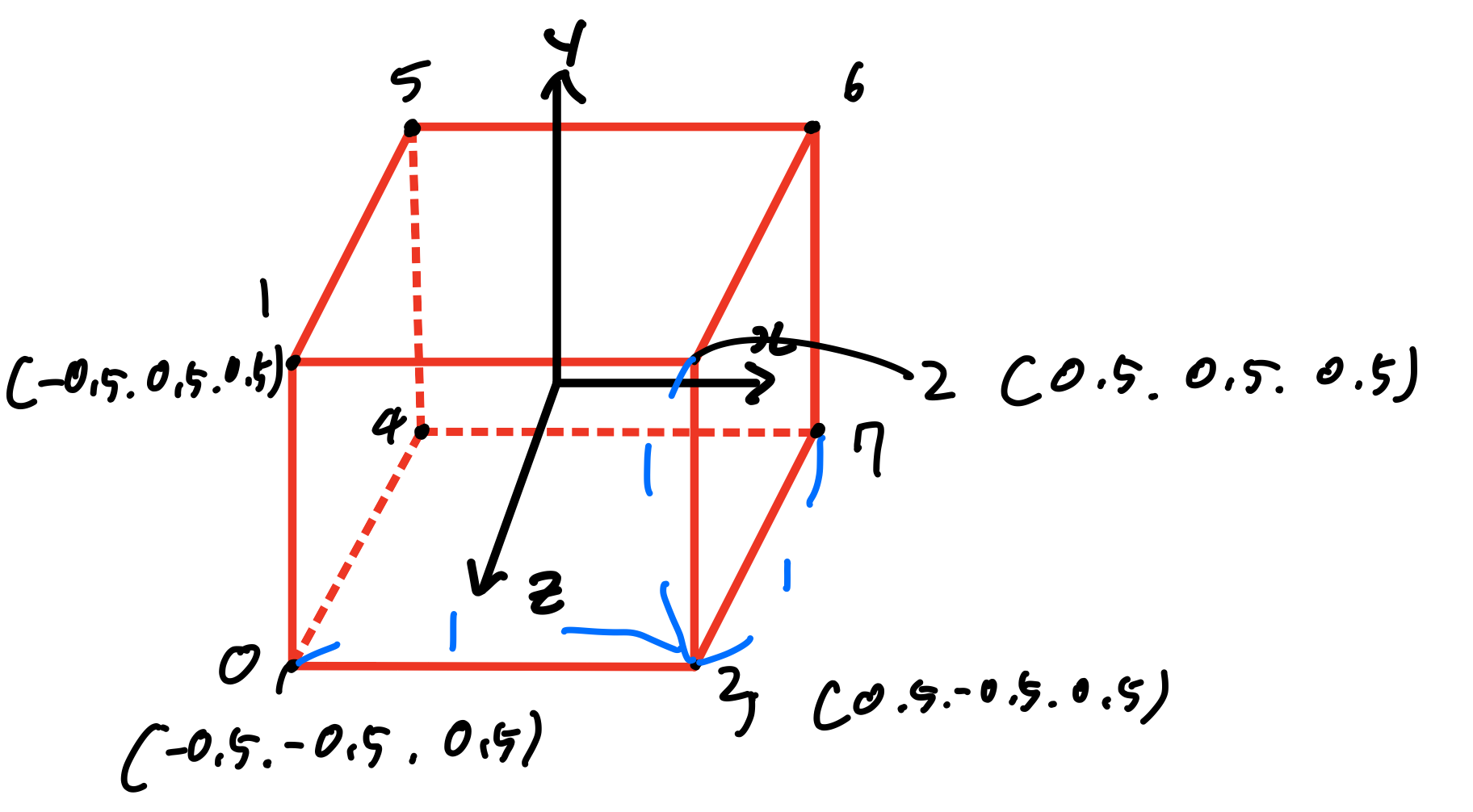


키보드 입력에 대한 처리이다. 먼저 L을 입력하면, shadeMode를 바꾸고 uniform value에 할당해준다. R을 누르면, 회전을 바꾼다. T를 누르면 Texture를 키고 끈다.

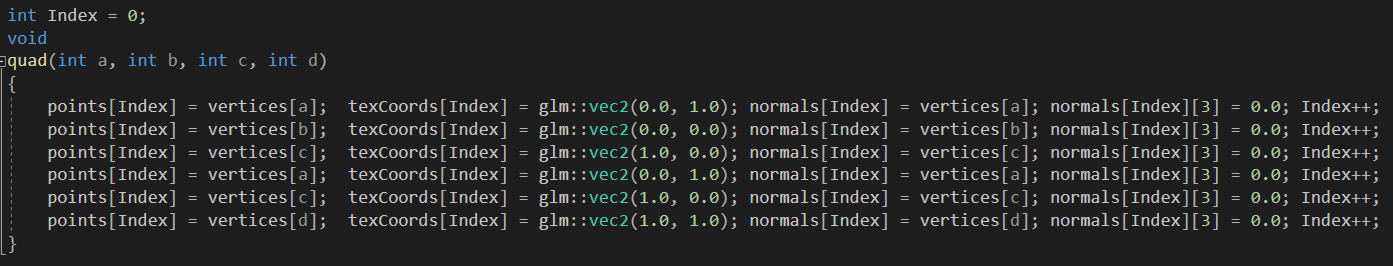
1. **Points, Normals, Texture Coordinate vector**



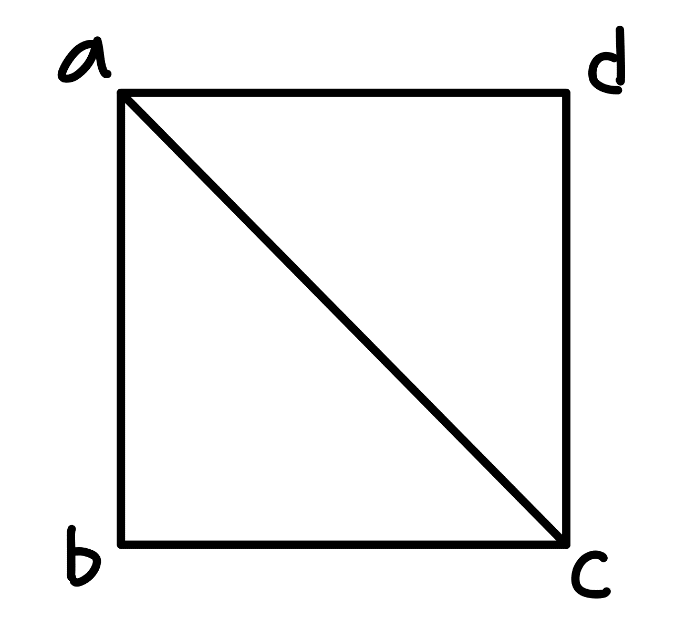
먼저 각 vertex의 위치를 지정해주었다. 위의 vertex를 그림으로 그려보면



위와 같은 vertex 위치와 좌표가 나온다. 이제 이것을 바탕으로 normal vector와 texcoord를 설정하겠다.

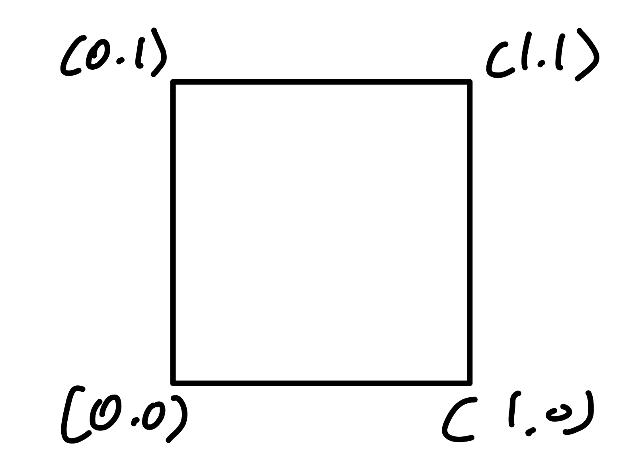


다시 언급하지만 points, normals, texCoords는 36길이를 가진 vector 배열이다. points, normals는 vec4 배열이고, texCoords는 vec2 배열이다. quad(a, b, c, d) 순서로 오면, (a, b, c), (a, c, d) 순서로 triangle이 할당된다. 이것을 그림으로 그려보면,

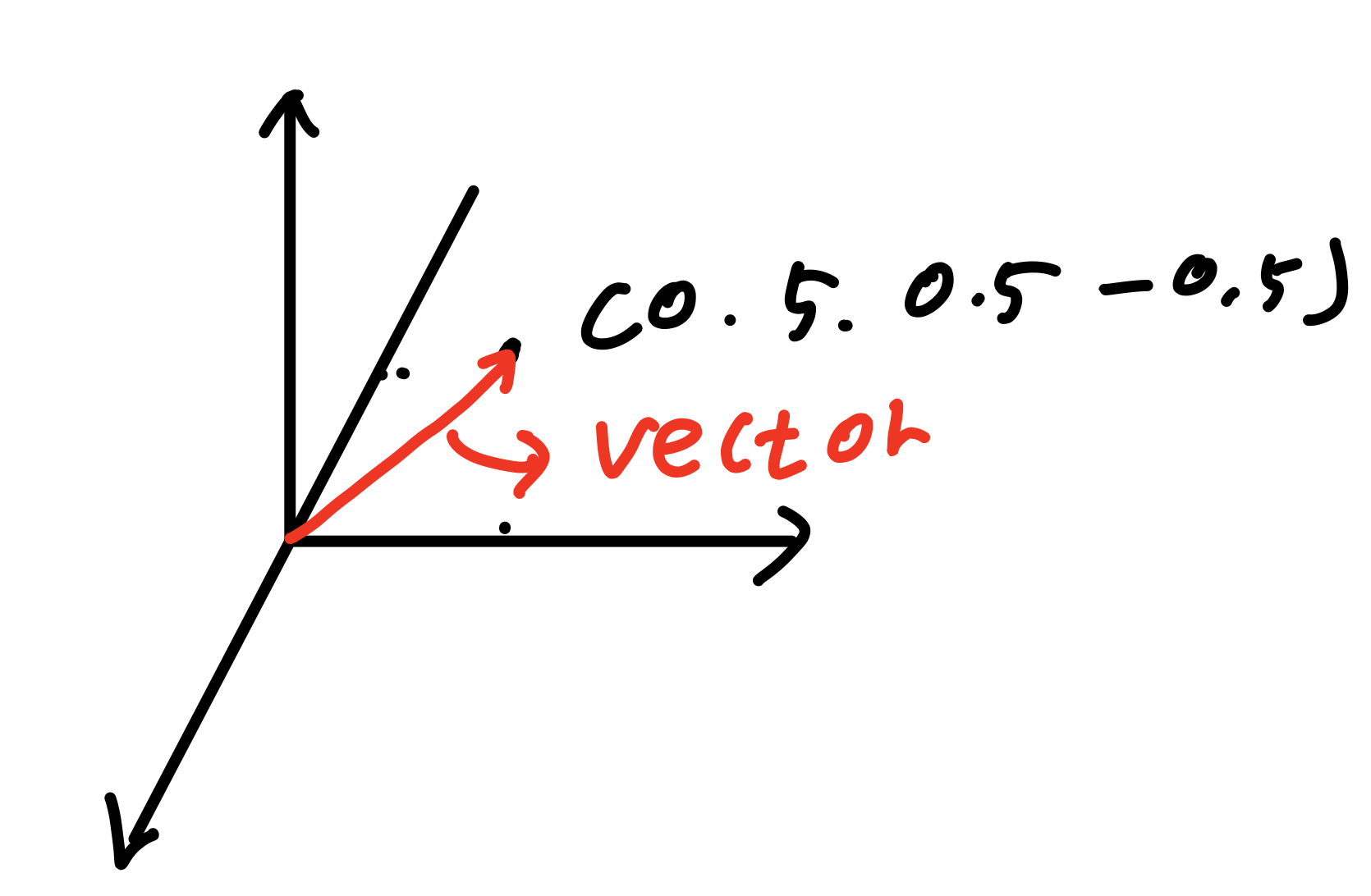


위의 그림과 같이 그려진다고 가정할 수 있다. 이것이 vertex array에 (a, b, c), (a, c, d) 순서로 저장된다.

Texture Coordinate는 U, V 값으로 결정되는데 texture map이

옆의 그림과 같이 할당되어 있다. 따라서 이 U, V 값을 texCoords로 저장하여야 한다. 우리는 이 값을 1대1 대응으로 위의 a, b, c, d에mapping 할 것이다. 따라서 코드를 살펴보면 (a, b, c), (a, c, d) 순서대로 ([0,1], [0,0], [1,0]), ([0,1], [1,0], [1,1]) 순서로 할당해준다.

Normal vector의 경우 vertices의 값을 사용하였다. 그 이유는 vertices의 값이 애초에 원점에서의 거리이기 때문에 그 점에 해당하는 normal vector라고 생각할 수 있다.



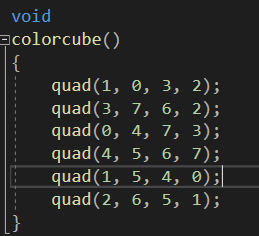
즉 (0.5, 0.5, -0.5) vertex의 경우 (0.5, 0.5, -0.5)를 vector로 가진다. 그래서 normal vector들을 그려보면 원점에서 해당 vertex로 이은 직선의 방향이 normal vector의 방향이 된다. Normalize는 shader에서 진행하기 때문에 고려하지 않아도 된다.

Opengl은 homogeneous coordinate을 사용하기 때문에 점은 나타낼 때는 마지막 원소에 1을 넣었다. (ex. (0.5, 0.5, -0.5, 1))과 같이 말이다. 그러나 normal vector는 vector이기 때문에 마지막 원소를 0으로 바꿔준다. (ex. (0.5, 0.5, -0.5, 0.0))과 같이 말이다. 그래서 코드를 살펴보면,

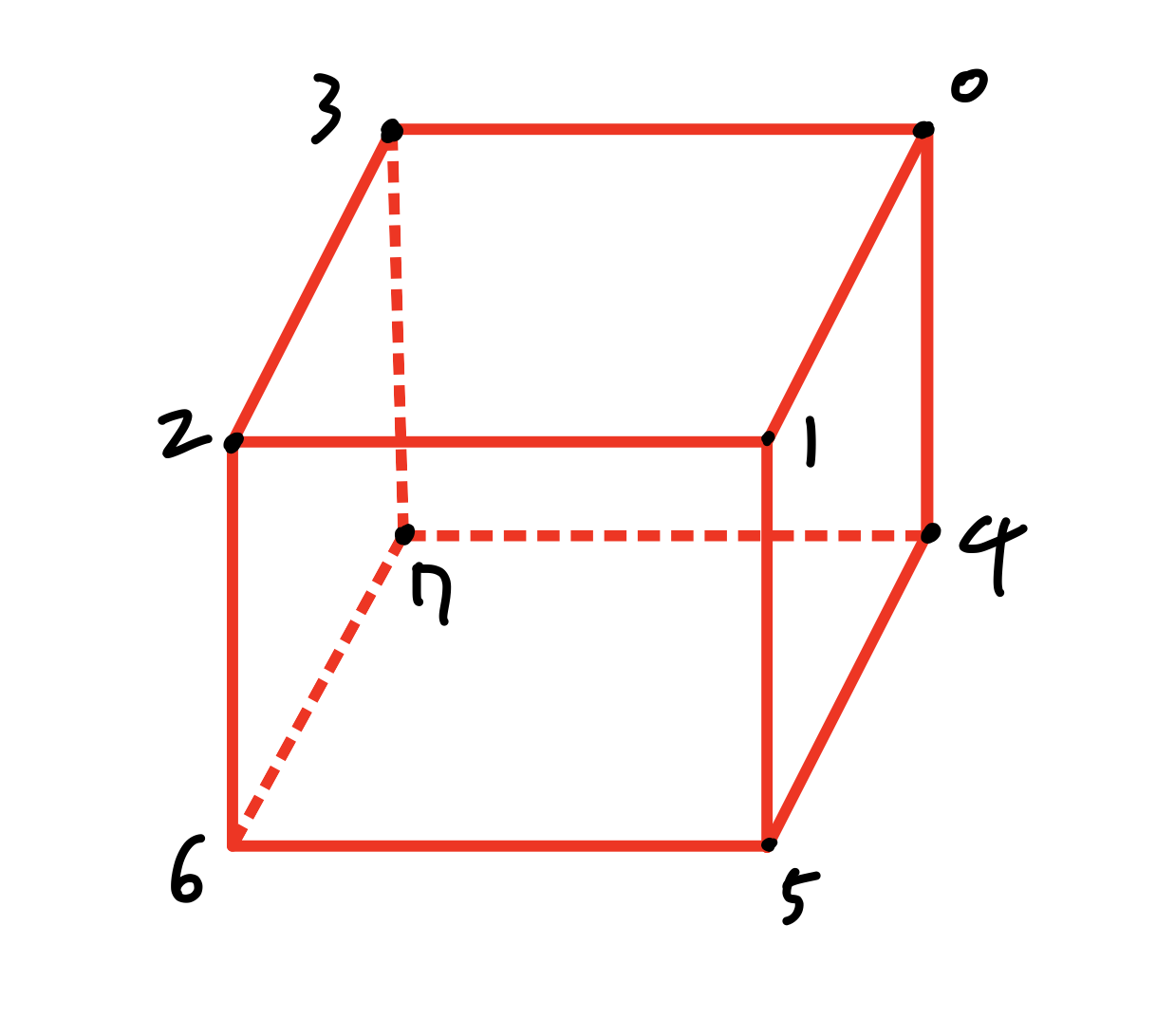


과 같이 마지막 원소를 0으로 바꿔주는 것을 알 수 있다.

앞선 과정을 통하여서 points, normal vector, texture coordinate를 모두 결정하였다.



이제 colorcube를 통해 vertex들을 설정해준다. 모델을 x축으로 회전하여 보여주기 때문에 그에 해당하는 texture mapping을 하기 위해 알맞은 순서대로 quad 속 parameter를 결정해주었다. 특히 마지막 quad은 얼굴 면인데 model matrix가 x축으로 -90도, y축으로 180도 돌아서 보면,

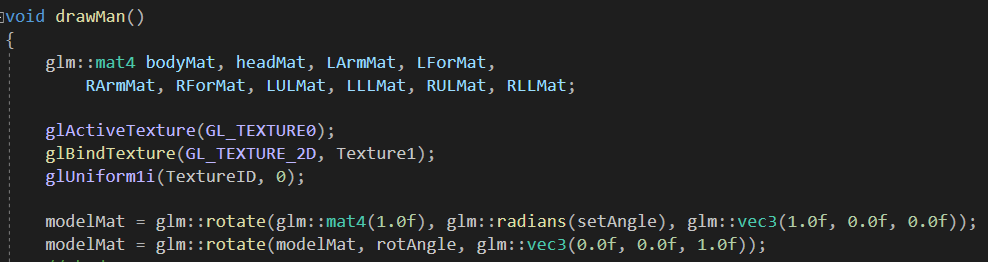


와 같이 생겼기 때문에 2, 6, 5, 1 면을 위의 a, b, c, d에 1대1 mapping하기 위하여 2, 6, 5, 1 순서로 vertex를 설정해주었다.

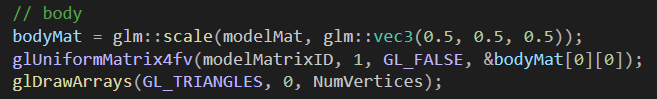
앞선 과정을 거치면 이제 36개의 순서로 vertex가 할당되고 그 순서에 맞춰서 normal vector, texture coordinate가 할당된다.

1. **Model draw**

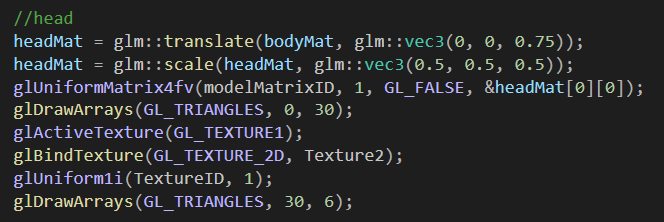
여기서 언급하지 않은 내용은 “컴퓨터 그래픽스 프로젝트 1”의 코드와 동일하다.



모델을 그리는 것은 drawMan() 함수에서 수행된다. 먼저 기본 Texture를 GL\_TEXTURE0에 할당해 놓은 Texture1을 Texture로 설정한다. 그리고 기본적인 modelMat을 설정한다. 이것은 “컴퓨터 그래픽스 프로젝트 1”에서 설명한 worldMat과 동일한 것이다.

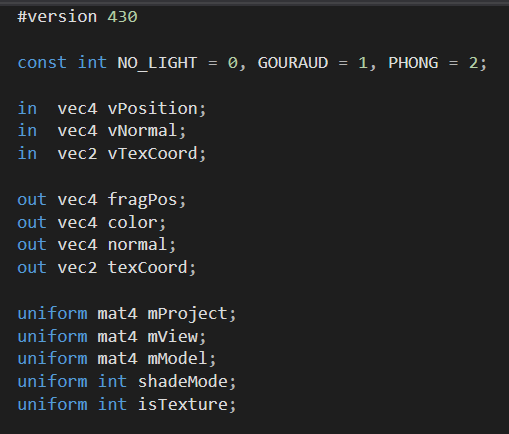


그리고 modelMatrixID에 정의한 modelMatrix를 할당한다. 원래 예전 코드에서는 projection \* view \* model mat을 하여 통째로 shader에 보냈지만, 이번 코드에서는 그 계산을 shader에게 맡기고 model mat만 정의하였다. 나머지 부분에 대해서도 model mat만 할당하는 식으로 코드가 구현되어 있다.



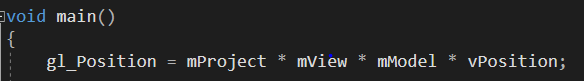
Head를 그리는 부분은 기존 코드와 다르다. 그 이유는 head 정육면체 중 한 면에 대해서만 얼굴에 해당하는 Texture를 할당해야 하기 때문에 그것에 대한 처리를 해주어야 한다. 먼저 할당된 삼각형 중 5면에 해당하는 30개의 vertex만 그린다. 그 이후 Texture2로 바꾸어 얼굴에 해당하는 Texture를 shader에 할당한 이후에 나머지 1면에 해당하는 vertex 6개를 그린다. 이렇게 하여 정육면체 중 한 면에만 다른 texture를 할당할 수 있다.

1. **Shader**
2. vshader.glsl

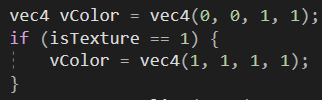


먼저, input, output, uniform이다. Input으로는 vPosition(points와 연결), vNormal(normals와 연결), vTexCoord(texCoords와 연결)이 있다. 그리고 output으로는 fragment position을 나타내는 fragPos와, color가 있고, phong shading에서 사용할 normal vector의 결과를 넘기는 normal, vTexCoord로부터 texCoord를 계산하여 넘기는 texCoord가 있다

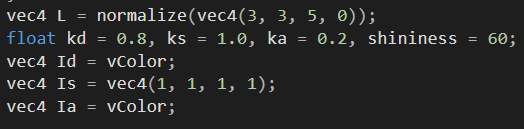
uniform으로는 mProjection(Projection matrix), mView(View matrix), mModel(model matrix)과 shadeMode, isTexture변수가 들어온다.



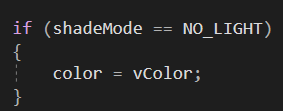
먼저 vPosition을 가지고 mProjection \* mView \* mModel \* vPosition을 계산하여 gl\_Position을 계산한다.



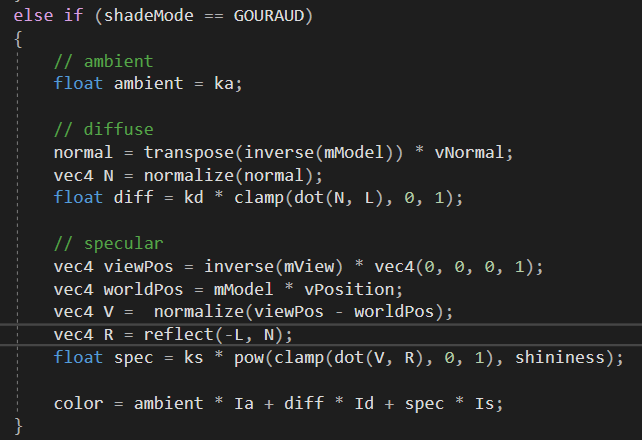
먼저 기본 Color는 blue로 설정해준다. 만약 texture가 있으면 1, 1, 1 이므로 하얀색으로 설정해준다.



먼저, L 벡터를 정한다. L 벡터는 3, 3, 5 방향이다. 그리고 unit vector로 만들기 위하여 normalize를 사용하였다. 그리고 이 물체의 표면을 정의하였다. kd = 0.8, ks = 1.0, ka = 0.2, shininess = 60으로 하였다. 그리고 각각 ambient에 해당하는 광 Ia = vColor, diffuse에 해당하는 광 I­d = vColor, specular 광인 Is = (1, 1, 1, 1) 을 설정해준다.



이 경우는 쉐이더가 NO\_LIGHT인 경우이다. 이때의 color는 vColor 그대로 사용한다.



이 경우는 쉐이드가 GOURAUD일 때의 경우이다. 먼저 ambient를 ka로 설정한다. 그리고 shader에 넘어온 vNormal을 가지고 mModel transform을 이용하여 transform된 normal vector를 계산한다. 그리고 이것을 normalize하여 최종 unit normal vector를 구한다. 그리고 diffuse 관련 계산을 진행하는데,



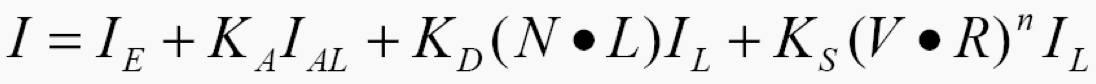
이 식을 계산하기 위함이다. Kd에 N와 L의 내적을 곱한다. 그것은 dot(N, L)을 통해 내적을 계산한다. 그리고 clamp을 이용하여 0보다 작은 값은 0으로, 1보다 큰 값은 1로 계산한다.

그리고 specular를 계산하기 위하여 먼저 transform된 viewPos와 vertex의 worldPos를 구하고, 그 두 vector를 빼고 normalize하여 V vector를 구한다. 그리고 -L vector를 N에 대해 반사시켜 R vector를 구한다. 이 vector들을 가지고

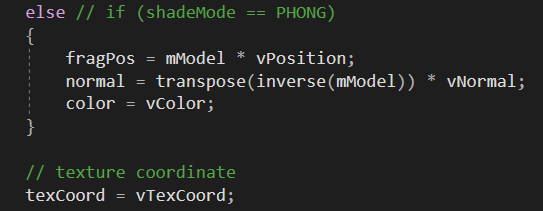


이 식을 계산한다. Clamp()을 이용하여 V와 R의 내적 계산한 값을 처리하고 이후 pow를 통해 V R 내적 값에 shininess를 제곱한다. 그리고 ks와 곱해 specular 상수를 만든다.

이후 앞서 계산한 상수를 가지고 color를 계산한다.

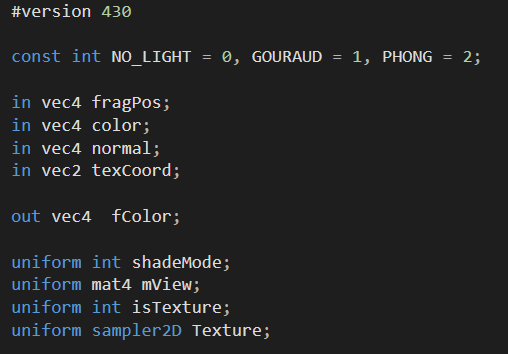


이 식을 계산한다. IE = 0 이기 때문에 계산하지 않고, 나머지의 경우 일반 상수는 다 계산했기 때문에 상수 \* I 들을 계산해서 총 I를 구한다.

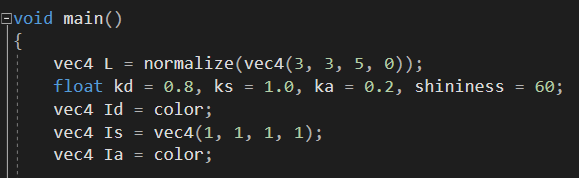


PHONG shading일 때의 계산 과정이다. 먼저 Phong shading에서는 vertex의 normal vector를 가지고 fragment shader에서 lighting을 계산하기 때문에 vertex shader에서는 변환된 fragment Position과 normal vector와 color만 계산한다. 따라서 fragment position은 vertex position을 가지고 mModel로 transform 시켜 구하고, normal vector를 계산한다. 그리고 texture coord는 그대로 들어가도 되기 때문에 그대로 넘겨준다.

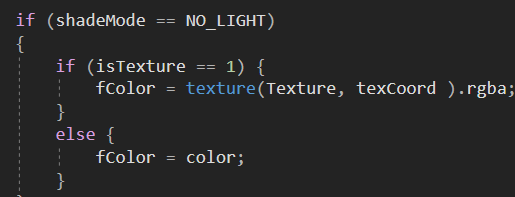
1. fshader.glsl



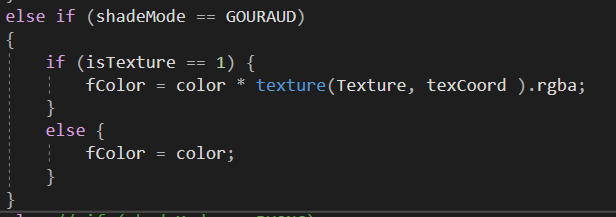
Fragment shader이다. 여기서는 4개의 input이 있다. Fragment Position을 나타내는 fragPos, Color를 나타내는 color, normal vector를 나타내는 normal, 그리고 texture coord를 나타내는 texCoord를 받아온다. 그리고 output으로는 fragment color인 fColor를 output한다. Fragment shader에서 사용하는 uniform은 shadeMode를 나타내는 shadeMode와 view matrix를 나타내는 mView, texture를 사용할지 저장하는 isTexture, 마지막으로 Texture 이미지인 Texture를 가져온다.



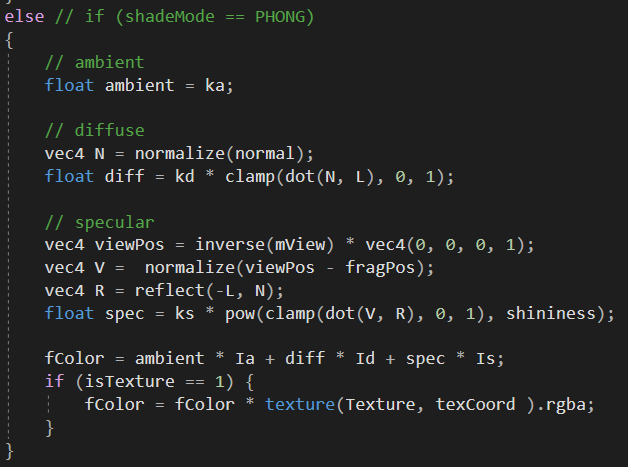
Fragment shader에서도 vertex shader와 마찬가지로 위와 같이 상수들을 정의해준다.



shadeMode가 NO\_LIGHT인 경우, 먼저 texture를 사용한다면, fColor에 Texture의 texCoord에 해당하는 rgba 값을 할당한다. 만약 texture를 사용하지 않는다면 vertex shader에서 넘어온 color를 fColor에 할당한다.

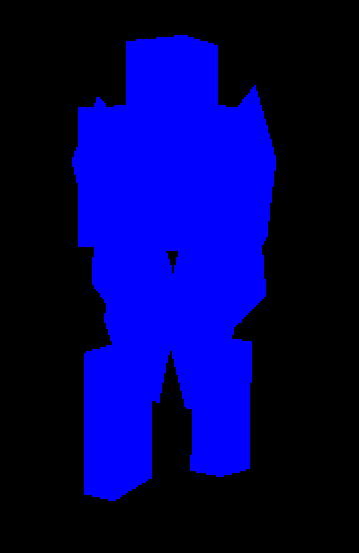


shadeMode가 GOURAD인 경우, 먼저 texture를 사용한다면, vertex shader에서 넘어온 color에 Texture의 texCoord에 해당하는 rgba 값을 곱해서 fColor를 결정한다. 만약 texture를 사용하지 않으면, vertex shader에서 계산된 illumination 값을 fColor에 할당한다.



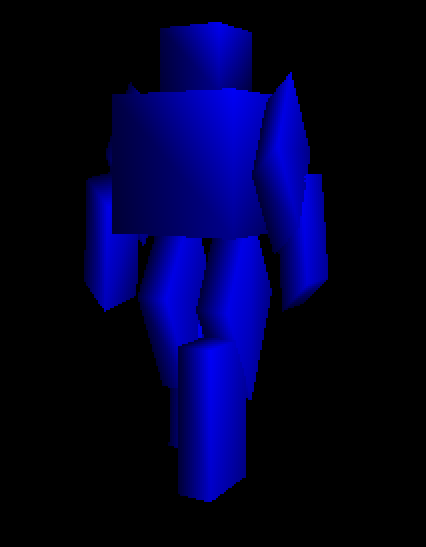
만약 shadeMode가 PHONG인 경우, 먼저 lighting부터 계산한다. 이 과정은 앞에 vertex shader에서 GOURAUD에서 lighting을 계산한 과정과 같다. 단지 다른 것은 PHONG shading에서는 이 과정을 fragment shader에서 진행하기 때문에 각각의 fragment 마다 계산한다는 것이다.

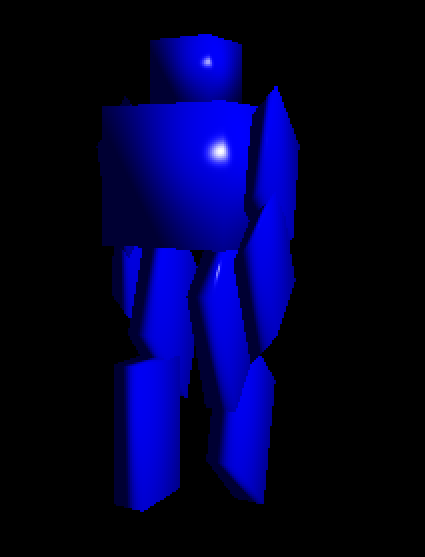
계산한 값을 fColor에 할당하고, 만약 Texture를 사용한다면 그 fColor에 texture를 할당해준다.

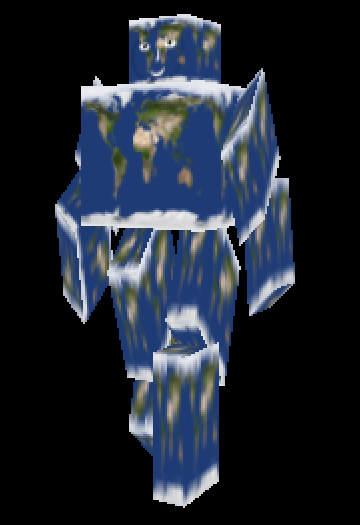
1. **결과**

기본

NO\_LIGHT, NO\_TEXTURE

GOURAUD, NO\_TEXURE

PHONG, NO\_TEXTURE

NO\_LIGHT, TEXTURE

GOURAUD, TEXTURE

PHONG, TEXTURE



다른 각도의 PHONG, TEXTURE

**소스코드 위치: /src/main.cpp**

**작업 OS: Windows 10**

**작업 컴파일러: Visual Studio 2019**