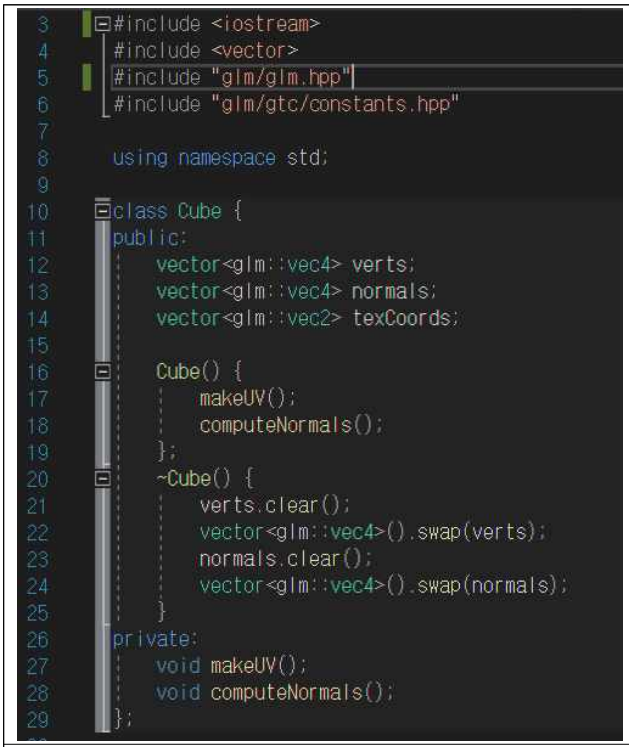
Swimming Man Texture & Light

ID/ Name : 20174089/ 윤석원

1. Cube.cpp



```
#include <iostream>
#include <vector>
#include "glm/glm.hpp"
#include "glm/gtc/constants.hpp"

using namespace std;

class Cube {
public:
    vector<glm::vec4> verts;
    vector<glm::vec4> normals;
    vector<glm::vec2> texCoords;

    Cube() {
        makeUV();
        computeNormals();
    };
    ~Cube() {
        verts.clear();
        vector<glm::vec4>().swap(verts);
        normals.clear();
        vector<glm::vec4>().swap(normals);
    }
private:
    void makeUV();
    void computeNormals();
};
```

- swimming man을 cube를 변형시켜 그릴 것이기 때문에 cube class를 정의한다.
- texture를 입힐 것이기 때문에 texCorrds을 저장할 공간이 필요하다.
- light계산을 위해 각 vertex마다 normal vector를 저장할 공간이 필요하다.
- 생성과 동시에 triangle들을 이루는 36개의 vertex를 생성하고 각 vertex의 texture coordinate와 normal vector를 계산한다.

```
void Cube::makeUV() {
        vector<glm::vec4> vertList;
        // 8개의 vertex정의
        vertList.push_back(glm::vec4(-0.5, -0.5, 0.5, 1.0));
        .....생략.....

        // 반시계 방향으로 vertex를 추가하여 삼각형을 하나씩 만들어준다.
        verts.push_back(vertList[1]);
        verts.push_back(vertList[0]);
        verts.push_back(vertList[3]);
         .....생략.....

        // 추가된 vertex마다 올바른 texCoordinate를 지정해준다.
        // texture이미지의 각 꼭지점 위치를 매핑해준다 (0,0), (0,1), (1,0), (1,1)
        for (int i = 0; i < 6; i++) {
                glm::vec2 texcoord[4];
                const int U = 0, V = 1;

                texcoord[0][U] = 1;
                texcoord[1][U] = 1;
                texcoord[2][U] = 0;
                texcoord[3][U] = 0;

                texcoord[0][V] = 0;
                texcoord[1][V] = 1;
```

```
                texcoord[2][V] = 1;
                texcoord[3][V] = 0;

                texCoords.push_back(texcoord[0]);
                texCoords.push_back(texcoord[1]);
                texCoords.push_back(texcoord[2]);

                texCoords.push_back(texcoord[2]);
                texCoords.push_back(texcoord[3]);
                texCoords.push_back(texcoord[0]);
        }
}

// light 계산에 필요한 normal vector를 vertex마다 계산해서 저장한다.
// 한 면을 이루는 네 vertex의 normal vector는 동일하다.
void Cube::computeNormals() {
        for (int i = 0; i < verts.size(); i++)
        {
                glm::vec4 n;
                for (int k = 0; k < 3; k++)
                {
                        n[k] = verts[i][k];
                }
                n[3] = 0.0;
                glm::normalize(n);
                normals.push_back(n);
        }
}
```

2. main.cpp : init()

```
void init()
{
        .....생략.....
        // set up vertex arrays
        // shader에서 texture와 light를 계산하려면 vertex의 위치, normal vector,
        // texture coordinate가 필요하기 때문에 버퍼에 저장하고 위치를 알려준다.
        GLuint vPosition = glGetAttribLocation(program, "vPosition");
        glEnableVertexAttribArray(vPosition);
        glVertexAttribPointer(vPosition, 4, GL_FLOAT, GL_FALSE, 0,
                BUFFER_OFFSET(0));

        GLuint vNormal = glGetAttribLocation(program, "vNormal");
        glEnableVertexAttribArray(vNormal);
        glVertexAttribPointer(vNormal, 4, GL_FLOAT, GL_FALSE, 0,
                BUFFER_OFFSET(vertSize));

        GLuint vTexCoord = glGetAttribLocation(program, "vTexCoord");
        glEnableVertexAttribArray(vTexCoord);
        glVertexAttribPointer(vTexCoord, 2, GL_FLOAT, GL_FALSE, 0,
                BUFFER_OFFSET(vertSize+normalSize));
        // 밝기, 위치를 vertex shader에서 계산 해야하기 때문에
        // 필요한 project, view, model matrix를 shader에서 접근할 수 있도록 해준다.
        projectMatrixID = glGetUniformLocation(program, "mProject");
        projectMat = glm::perspective(glm::radians(65.0f), 1.0f, 0.1f, 100.0f);
        glUniformMatrix4fv(projectMatrixID, 1, GL_FALSE, &projectMat[0][0]);

        viewMatrixID = glGetUniformLocation(program, "mView");
        viewMat = glm::lookAt(glm::vec3(0, 4, 0), glm::vec3(0, 0, 0), glm::vec3(0, 0, 1)); //right
        glUniformMatrix4fv(viewMatrixID, 1, GL_FALSE, &viewMat[0][0]);

        modelMatrixID = glGetUniformLocation(program, "mModel");
        modelMat = glm::mat4(1.0f);
        glUniformMatrix4fv(modelMatrixID, 1, GL_FALSE, &modelMat[0][0]);

        .....생략.....
        // 사용할 texture 이미지를 업로드한다.
        // ID를 부여하고 shader에서 접근할 수 있도록 해준다.
        // 끝으로 바인딩까지 해준다.
        GLuint Texture = loadBMP_custom("flag.bmp");
        GLuint TextureID = glGetUniformLocation(program, "cubeTexture");
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, Texture);
        glUniform1i(TextureID, 0);
        .....생략.....
}
```

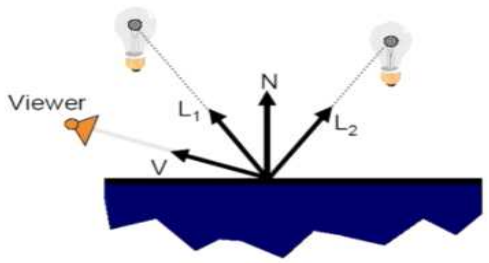
3. Vertex shader

```
3   const int NO_LIGHT = 0, GOURAUD = 1, PHONG = 2;
4
5   in  vec4 vPosition;
6   in  vec4 vNormal;
7   in  vec2 vTexCoord;
8
9   out vec4 fragPos;
10  out vec4 color;
11  out vec4 normal;
12  out vec2 texCoord;
13
14  uniform mat4 mProject;
15  uniform mat4 mView;
16  uniform mat4 mModel;
17  uniform int shadeMode;
18  uniform int isTexture;
19
20  uniform mat4 mPVM;
```

1: display 모드에 따라 shader에서 vertex에 대해 계산을 다르게 적용해야 하므로 상수를 정의
5~8: 각 vertex에 대해 새로운 위치, texture coordinate, light, color를 계산할 것이기 때문에 input으로 받는다.
14~18 : 사용하고 있는 view, project, model matrix에 접근하기 위한 uniform 선언

```
void main()
{
        // vertex의 위치 이동
        gl_Position = mProject * mView * mModel * vPosition;
        // texture를 입힐 것이기 때문에 중요하지는 않지만 vertex의 color 지정
        vec4 vColor = vec4(0, 0, 1, 1);
        if (isTexture == 1) {
                vColor = vec4(1, 1, 1, 1);
        }
        // 광원의 방향 및 ambient, diffuse, specular, shininess 등의 값을 정해준다.
        vec4 L = normalize(vec4(3, 3, 5, 0));
        float kd = 0.8, ks = 1.0, ka = 0.2, shininess = 60;
        vec4 Id = vColor;
        vec4 Is = vec4(1, 1, 1, 1);
        vec4 Ia = vColor;

        .....생략.....
        else if (shadeMode == GOURAUD)
        {       // GOURAUD는 vertex shader에서 vertex의 색을 normal vector를 가지고 계산한다.
                // 이 최종 color를 fragment shader에 보내주면
                // interpolation으로 다른 pixel의 색을 결정한다.
                // ambient
                float ambient = ka;

                // diffuse
                normal = transpose(inverse(mModel)) * vNormal;
                vec4 N = normalize(normal);
                float diff = kd * clamp(dot(N, L), 0, 1);

                // specular
                vec4 viewPos = inverse(mView) * vec4(0, 0, 0, 1);
                vec4 worldPos = mModel * vPosition;
```

```
        vec4 V =  normalize(viewPos - worldPos);
        vec4 R = reflect(-L, N);
        float spec = ks * pow(clamp(dot(V, R), 0, 1), shininess);

        color = ambient * Ia + diff * Id + spec * Is;
    }
```



$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$

수식에 따라 최종 vertex의 밝기를 구한다

```
    else // if (shadeMode == PHONG)
    {       // 반면 PHONG shading은 각 pixel의 normal vector들을 interpolation으로 구하고
            // fragment shader에서 각 pixel의 최종 color를 계산해서 구한다.
            // 따라서 fragPos, normal vector를 계산한다.
            fragPos = mModel * vPosition;
            normal = transpose(inverse(mModel)) * vNormal;
            color = vColor;
    }
    // 우리가 사전에 mapping했던 texture coordinate 좌표를 매칭해준다.
    // texture coordinate
    texCoord = vTexCoord;
}
```

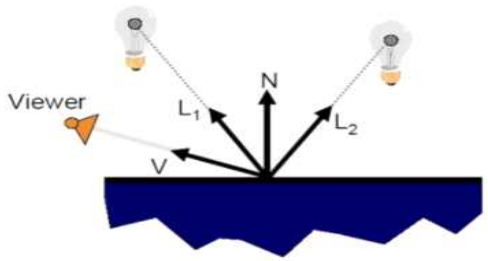
4. Fragment shader

```
3   const int NO_LIGHT = 0, GOURAUD = 1, PHONG = 2;
4
5   in vec4 fragPos;
6   in vec4 color;
7   in vec4 normal;
8   in vec2 texCoord;
9
10  out vec4  fColor;
11
12  uniform int shadeMode;
13  uniform mat4 mView;
14  uniform int isTexture;
15  uniform sampler2D cubeTexture;
```

1: display 모드에 따라 shader에서 vertex에 대해 계산을 다르게 적용해야 하므로 상수를 정의
5~8: 최종 pixel color를 계산하기 위해 필요한 내용들을 input으로 받는다.
12~15: 사용하고 있는 view matrix와 texture map에 접근하기 위한 uniform 선언

```
void main()
{       // phong shading에 사용할 광원의 방향 및 ambient, diffuse, specular, shininess를 정의
        vec4 L = normalize(vec4(3, 3, 5, 0));
        float kd = 0.8, ks = 1.0, ka = 0.2, shininess = 60;
        vec4 Id = color;
        vec4 Is = vec4(1, 1, 1, 1);
        vec4 Ia = color;
```

```
.....생략.....

else if (shadeMode == GOURAUD)
{       // gourad shading의 경우 이미 lighting을 계산했기 때문에
        // texture map과 texture coordinate에 맞춰서 최종 픽셀의 색을 정한다.
        if (isTexture == 1) {
                fColor = color * texture( cubeTexture, texCoord ).rgba;
        }
        else {
                fColor = color;
        }
}
else // if (shadeMode == PHONG)
{       // phong shading은 vertex shader에서 구한 픽셀의 normal vector를 가지고
        // 픽셀의 색을 계산한다.
        // ambient
        float ambient = ka;

        // diffuse
        vec4 N = normalize(normal);
        float diff = kd * clamp(dot(N, L), 0, 1);

        // specular
        vec4 viewPos = inverse(mView) * vec4(0, 0, 0, 1);
        vec4 V =  normalize(viewPos - fragPos);
        vec4 R = reflect(-L, N);
        float spec = ks * pow(clamp(dot(V, R), 0, 1), shininess);

        fColor = ambient * Ia + diff * Id + spec * Is;
```



$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$

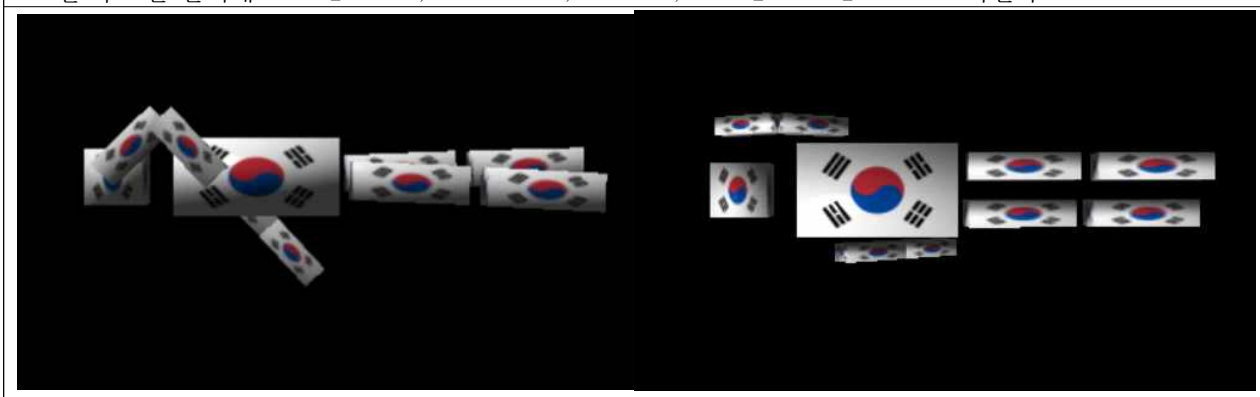
수식에 따라 최종 vertex의 밝기를 구한다

```
        // texture map과 texture coordinate에 맞춰서 최종 픽셀의 색을 정한다.
        if (isTexture == 1) {
                fColor = fColor * texture( cubeTexture, texCoord ).rgba;
        }
}
```

8. 결과 확인

```
316  void keyboard(unsigned char key, int x, int y)
317  {
318      switch (key) {
319      case 'l': case 'L':
320          shadeMode = (++shadeMode % NUM_LIGHT_MODE);
321          glUniform1i(shadeModeID, shadeMode);
322          glutPostRedisplay();
323          break;
324      case 'c': case 'C':
325          viewMat = glm::lookAt(glm::vec3(0, 0, 4), glm::vec3(0, 0, 0), glm::vec3(0, -1, 0));
326          glUniformMatrix4fv(viewMatrixID, 1, GL_FALSE, &viewMat[0][0]);
327          break;
328      case 'v': case 'V':
329          viewMat = glm::lookAt(glm::vec3(0, 4, 0), glm::vec3(0, 0, 0), glm::vec3(0, 0, 1));
330          glUniformMatrix4fv(viewMatrixID, 1, GL_FALSE, &viewMat[0][0]);
331          break;
332      case 033:  // Escape key
333      case 'q': case 'Q':
334          exit(EXIT_SUCCESS);
335          break;
336      }
337  }
```

- keyboard input으로 C가 들어오면 swimming man의 등을 볼 수 있고 V가 들어오면 측면을 관찰한다.
- 카메라 눈의 각도를 바꿔가며 올바르게 modeling이 되었는지 확인한다.
- L을 누르면 순서대로 NO_LIGHT, GOURAUD, PHONG, NUM_LIGHT_MODE로 바뀐다.



- 올바르게 texture mapping이 되었다.
- swimming man의 아래 부분이 어둡도록 light 계산이 잘 구현되어있다.
- 위에서 모델을 관찰하면 보이는 모든 면이 밝게 빛을 받고 있다.
- 움직이는 팔, 다리에 맞춰서 vertex shader에서 밝기를 잘 계산하고 있다.