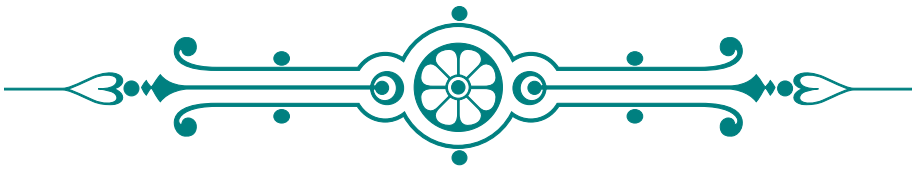

컴퓨터 그래픽스 입문

학번 : 2016110056
학과 : 불교학부
이름 : 박승원
날짜 : 2017년 5월 4일

Lab 9. Viewing

Use 'keyboard' to change your view like mouse controls do in the example code.

- 'Q / E' : rotation (imagine that you are a game character of a FPS game. Q turns your head left, E turns your head right) (2pt)
- 'W' : move forward to the object (2pt)
- 'S' : move backward from the object (2pt)
- 'A' : side step left (imagine one more time that you are a FPS game character) (2pt)
- 'D' : side step right (imagine one more time that you are a FPS game character) (2pt)

머리를 돌리는 것은 Y축을 기준으로 회전하는 것과 같고, 가까이 다가가는 것과 멀어지는 것은 scale과 같고, 좌우로 움직이는 것은 translation과 같다.
기존에 구현했던 callback을 조금 고치는 것으로 가능했다.
KeyBindMatrix는 메인화일에서 물체들을 변환한다.
또한 프로젝션 매트릭스를 이용해야 접근과 멀어짐, 좌우 회전이 더욱 실감난다.

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{// && action == GLFW_PRESS)
    switch(key) {
        case GLFW_KEY_A:
        KeyBindMatrix = m.gltranslate(-STEP, 0, 0) * KeyBindMatrix; break;
        case GLFW_KEY_DOWN:
        KeyBindMatrix = m.gltranslate(0, -STEP, 0) * KeyBindMatrix; break;
        case GLFW_KEY_D:
        KeyBindMatrix = m.gltranslate(STEP, 0, 0) * KeyBindMatrix; break;
        case GLFW_KEY_UP:
        KeyBindMatrix = m.gltranslate(0, STEP, 0) * KeyBindMatrix; break;
        case GLFW_KEY_W:
        KeyBindMatrix = m.gltranslate(0, 0, STEP) * KeyBindMatrix; break;
        case GLFW_KEY_S:
        KeyBindMatrix = m.gltranslate(0, 0, -STEP) * KeyBindMatrix; break;

        // case GLFW_KEY_W: KeyBindMatrix = m.glrotateX(STEP) * KeyBindMatrix; break;
        // case GLFW_KEY_A: KeyBindMatrix = m.glrotateY(-STEP) * KeyBindMatrix; break;
        // case GLFW_KEY_S: KeyBindMatrix = m.glrotateX(-STEP) * KeyBindMatrix; break;
        // case GLFW_KEY_D: KeyBindMatrix = m.glrotateY(STEP) * KeyBindMatrix; break;
        case GLFW_KEY_Q: KeyBindMatrix = m.glrotateY(-STEP) * KeyBindMatrix; break;
        case GLFW_KEY_E: KeyBindMatrix = m.glrotateY(STEP) * KeyBindMatrix; break;

        case GLFW_KEY_SPACE: KeyBindMatrix.E(); break;
```

```
            case GLFW_KEY_J: camera_x -= STEP; break;
            case GLFW_KEY_K: camera_y -= STEP; break;
            case GLFW_KEY_L: camera_x += STEP; break;
            case GLFW_KEY_I: camera_y += STEP; break;
        }
    }
```

Listing 1: main

```
#include<chrono>
#include<thread>
#include<iostream>
#include"glutil.h"
#include"globj.h"
using namespace std;
extern Matrix<float> KeyBindMatrix;

int main(int ac, char** av)
{
    if (!glfwInit()) return -1;
    GLFWwindow* window = glfwCreateWindow(1024, 768, "Color Cube", NULL, NULL);
    if (!glinit(window)) return -1;

    ///compile shaders
    unsigned shader_program =
        make_shader_program("src/vertex_shader.glsl", "src/fragment_shader.glsl");
    if (!shader_program) return 0;
    glUseProgram(shader_program);

    GLObject obj3d;
    unsigned sz = obj3d.read_obj_file("BuddhaSculpture.obj");
    vector<Matrix<float>> color{sz, {1,0,0}};
    obj3d.colors(color);
    Matrix<float> m{4,4};
    obj3d.matrix(m.glrotateX(M_PI/2) * m.glrotateX(M_PI) * m.glscale(0.01,0.01,0.01));

    GLObject ironman;
    sz = ironman.read_obj_file("ironman.obj");
    vector<Matrix<float>> col{sz, {1,1,0}};
    ironman.colors(col);
    ironman.matrix(m.glrotateX(-M_PI/2) * m.gltranslate(0.3,-0.2,0) * m.glscale(0.01,0.01,0.01));
```

```
GLObject cube;
Matrix<float> ve[8] = {{0,0,0}, {1,0,0}, {1,1,0}, {0,1,0},
    {0,0,1}, {1,0,1}, {1,1,1}, {0,1,1}};
vector<Matrix<float>> v, c;
int idx[24] = {0,1,2,3, 4,5,6,7, 0,4,5,1, 1,5,6,2, 2,6,7,3, 0,4,7,3};
for(auto a : idx) v.push_back(ve[a]);
for(int i=0; i<8; i++) for(int j=0; j<4; j++) {
    if(i==0 || i==6) continue;
    c.push_back(ve[i]);
}
vector<unsigned> id;
for(int i=0; i<24; i++) id.push_back(i);
cube.vertexes(v);
cube.colors(c);
cube.indices(id);
cube.matrix(m.glscale (0.1,0.1,0.1) * m.glortho (0,1,0,1,0,1) );
cube.mode(GL_QUADS);

GLObjs objs(shader_program);
objs += ironman;
objs += obj3d;
objs += cube;
objs.transfer_all ("a_pos", "a_color", "norm");



Matrix<float> light = {
    {0.2, 0.2, 0.2, 1}, //ambient
    {1, 1, 1, 1}, // diffuse
    {1, 1, 1, 1}, // specular
    {0, 0, 2, 1} // position 1 means a point 0 means a vector light
};
transfer_matrix (shader_program, light.transpose(), "LIGHT");
Matrix<float> proj{4,4};
proj.glprojection (-1,1,-1,1,-1,1);

float k = 0;
while (!glfwWindowShouldClose(window)) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    transfer_matrix (shader_program, proj * KeyBindMatrix * objs[0], "KeyBindMatrix");
    objs(0);
    transfer_matrix (shader_program, proj * KeyBindMatrix * objs[1], "KeyBindMatrix");
```

```
        objs (1) ;
        transfer_matrix (shader_program, proj * KeyBindMatrix * m.glrotateY (k) * m. gltranslate
            (0,0.5,0.4) * m.glrotateX(k) * objs [2], "KeyBindMatrix");
        objs (2) ;

        k+= 0.1;
        glfwSwapBuffers(window);
        glfwPollEvents () ;
        this_thread :: sleep_for (50ms);
    }
    glfwTerminate() ;
}
```