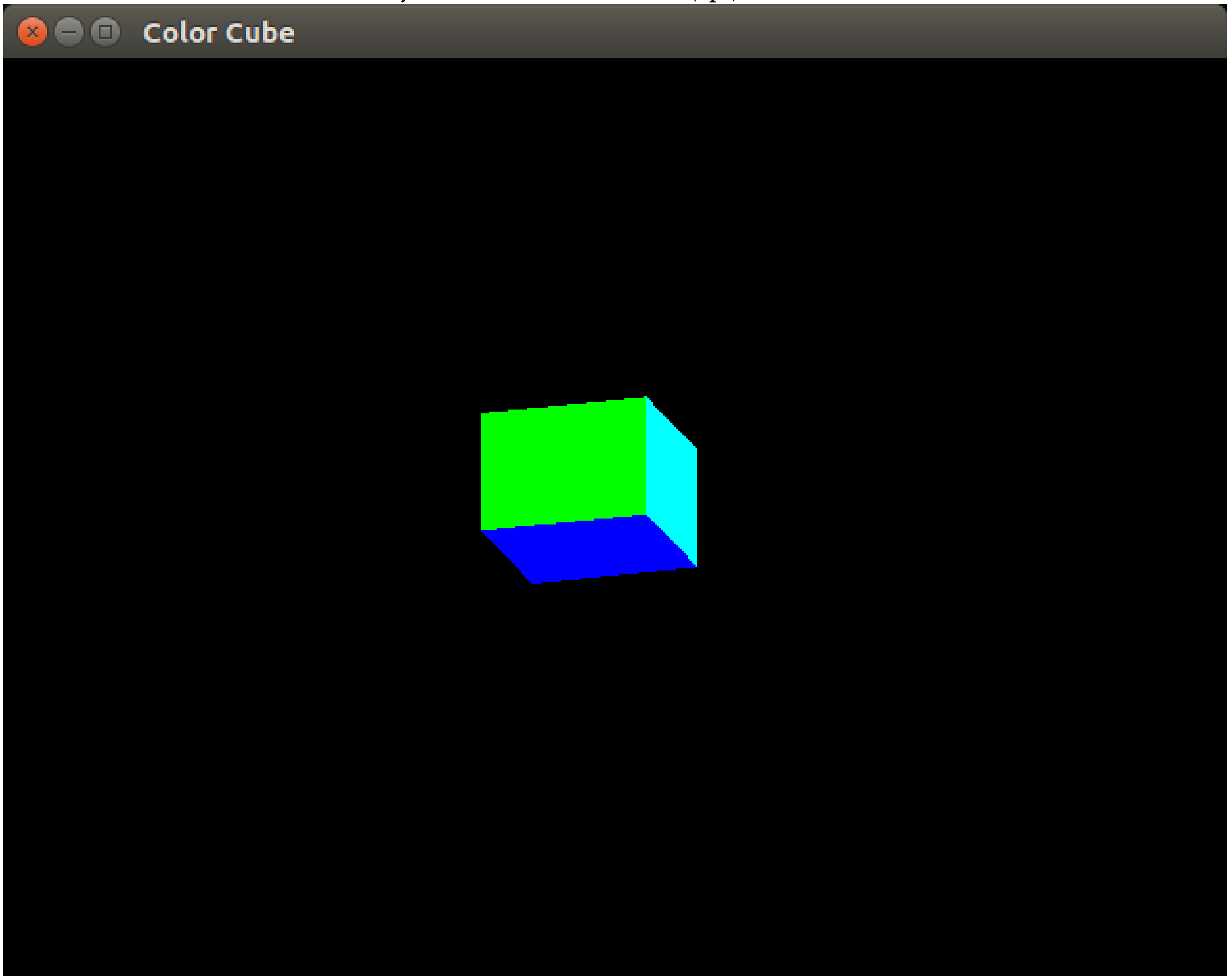
컴퓨터 그래픽스 입문

학번 :	2016110056
학과 :	불교학부
이름 :	박승원
날짜 :	2017년 4월 6일

dongguk UNIVERSITY

제 1 절 Shader Programming

1. Send color array to your vertex shader to draw a color box as you did last week. (5pt)
2. Use uniform matrix 4x4 to rotate your box in vertex shader.(5pt)



Listing 1: vertex shader

```
#version 130
uniform mat4 KeyBindMatrix;
in vec3 a_pos;
in vec3 a_color;
out vec3 color;

void main() {
    gl_Position = KeyBindMatrix * vec4(a_pos, 1.0);
    color = a_color;
}
```

Listing 2: fragment shader

```
#version 130
in vec3 color;
```

```
out vec4 f_color ;
void main() {
      f_color = vec4(color , 1.0) ;
}
```

Listing 3: 메인함수

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

int main()
{
      if (! glfwInit ()) return −1;

      GLFWwindow* window = glfwCreateWindow(640, 480, "Color Cube", NULL, NULL);
      if (! glinit (window)) return −1;
//    glortho (10);

      vector<Matrix<float>> colors, vertexes, vtx;
      Matrix<float> mm{4,4};

      auto pl = polygon(4);
      vtx. insert (vtx.end(), begin(pl), end(pl));
      Matrix<float> m{4,4};
      pl = m. gltranslate (0,0, sqrt (2)) * pl; //z+1
      vtx. insert (vtx.end(), begin(pl), end(pl)); //append elevated 4 vertex
      int idx[24] = {0,1,2,3,   4,5,6,7,   0,1,5,4,   1,2,6,5,   2,3,7,6,   0,3,7,4};
      for(int a : idx) vertexes.push_back(vtx[a]);
      for(auto& a : vertexes) a = m.glrotateZ(M_PI/4) * a;

      Matrix<float> clr[] = {{1,0,0}, {0,1,0}, {0,0,1}, {1,1,0}, {1,0,1}, {0,1,1}};
      for(int i=0; i<6; i++) for(int j=0; j<4; j++) colors.push_back(clr[i]);

      unsigned element[24];
      for(int i=0; i<24; i++) element[i] = i;

      unsigned vbo[3];
      vbo[1] = gl_transfer_data (colors);
      vbo[0] = gl_transfer_data (vertexes);
```

```
    vbo[2] =  gl_transfer_data (element, element + 24, GL_ELEMENT_ARRAY_BUFFER);

    const valarray<Matrix<float>> cube{vertexes.data(), vertexes . size () };
    valarray<Matrix<float>> v;

    ///compile  shaders
    unsigned shader_program =
        make_shader_program("src/vertex_shader . glsl ",  "src/fragment_shader. glsl ",
                "a_pos",  "a_color") ;
    if (! shader_program) return 0;
    float  k = 0;
    while (! glfwWindowShouldClose(window)) {
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glUseProgram(shader_program);
        auto tm = m. glscale   (0.2,0.2,0.2)    * KeyBindMatrix * m.glrotateX(k) ;
        transfer_matrix (shader_program, tm, "KeyBindMatrix");
        k += 0.1;

        glBindBuffer (GL_ARRAY_BUFFER, vbo[1]);
        glEnableVertexAttribArray (1) ;
        glVertexAttribPointer  (1,  3,  GL_FLOAT, GL_FALSE, 0, (void*)0) ;
        glBindBuffer (GL_ARRAY_BUFFER, vbo[0]);
        glEnableVertexAttribArray (0) ;
        glVertexAttribPointer  (0,  3,  GL_FLOAT, GL_FALSE, 0, (void*)0) ;
        // attribute  0,  xyz3,  float ,  normalized ?,  stride ,  offset

        glBindBuffer (GL_ELEMENT_ARRAY_BUFFER, vbo[2]);
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glDrawElements(GL_QUADS, 24, GL_UNSIGNED_INT, 0);

        glfwSwapBuffers(window);
        glfwPollEvents () ;
        this_thread :: sleep_for (chrono :: milliseconds (50) ) ;
    }
    glfwTerminate() ;
}
```

shader 프로그램을 만들고 변수를 보내는 make shader program 이란 함수를 만들었다.
또, 매트릭스 데이터를 보내고 바인드하는 transfer matrix 함수도 만들었다.
내가 만든 매트릭스는 자료구조가 약간 달라 transpose 한 후에 데이터를 보내야 되었다.
밑의 callbacks.cc의 하단에 있으니 참조바랍니다.

프로그램은 이전에 만들어두었던 큐브를 그대로 사용하여, 데이터를 세이더에 보내고, 바인드하는 작업을 한다.
세이더에서 입력받을 변수를 선언하고, vbo를 배열로 일렬로 만든 후, 바인드하면 되었다.

Listing 4: callbacks.cc

```
#include<GL/glew.h>
#include<GLFW/glfw3.h>
#include<iostream>
#include<vector>
#include<valarray>
#include<fstream>
#include"matrix.h"
#define STEP 0.05
using namespace std;
Matrix<float> KeyBindMatrix{4,4};

 static  Matrix<float> m{4,4};
bool record  =  false;
 float  camera_x=1, camera_y=1;

void key_callback (GLFWwindow* window, int key, int scancode, int  action ,  int  mods)
{//  && action  == GLFW_PRESS)
    switch(key) {
    case GLFW_KEY_LEFT:
        KeyBindMatrix = m. gltranslate (-STEP, 0,  0)  *  KeyBindMatrix; break;
    case GLFW_KEY_DOWN:
        KeyBindMatrix = m. gltranslate  (0,  -STEP, 0) *  KeyBindMatrix; break;
    case GLFW_KEY_RIGHT:
        KeyBindMatrix = m. gltranslate (STEP,  0,  0)  *  KeyBindMatrix; break;
    case GLFW_KEY_UP:
        KeyBindMatrix = m. gltranslate  (0,  STEP,  0)  *  KeyBindMatrix; break;
    case GLFW_KEY_Z:
        KeyBindMatrix = m. gltranslate  (0,   0,  STEP) *  KeyBindMatrix; break;
    case GLFW_KEY_X:
        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.glrotateZ(-STEP) * KeyBindMatrix; break;
    case GLFW_KEY_E: KeyBindMatrix = m.glrotateZ(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;
    }
}
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    double x, y;
    if(button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {//GLFW_RELEASE
        glfwGetCursorPos(window, &x, &y);
        cout << '(' << x / 4 << ',' << y / 4 << ')' << flush;
    }
}
void cursor_pos_callback (GLFWwindow* window, double xpos, double ypos) {
    if(record) cout << xpos << ' ' << ypos << ' ' << flush;
}

void glortho(float r) {
    glOrtho(-r,r,-r,r,-r,r);
}
void glcolor (unsigned char r, unsigned char g, unsigned char b, unsigned char a) {
    glColor4f(float(r)/256, float(g)/256, float(b)/256, float(a)/256);
}

bool glinit (GLFWwindow* window)
{
    if (!window) {
        glfwTerminate();
        return false;
    }
    // callbacks here
    glfwSetKeyCallback(window, key_callback);
    glfwSetMouseButtonCallback(window, mouse_button_callback);
    glfwSetCursorPosCallback(window, cursor_pos_callback);
    /* Make the window's context current */
    glfwMakeContextCurrent(window);
    glClearColor(0, 0, 0, 0); // black background
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);

    glewExperimental = true; // Needed for core profile
    if (glewInit() != GLEW_OK) {
        cerr << "Failed to initialize GLEW" << endl;
        glfwTerminate();
        return false;
    }
    KeyBindMatrix.E();
    cout << "shading language version : " <<
            glGetString(GL_SHADING_LANGUAGE_VERSION) << endl;
    cout << glGetString( GL_VENDOR ) << endl;
    cout << glGetString( GL_RENDERER ) << endl;
    cout << glGetString( GL_VERSION ) << endl;
    return true;
}

std::valarray<Matrix<float>> polygon(int points_count, float r)
{
    Matrix<float> p{r, 0, 0}; //when arg is 3 or 4, it makes 4x1 matrix r ,0,0,1
    Matrix<float> rz{4, 4}; // this makes 4x4 matrix
    std::valarray<Matrix<float>> pts{Matrix<float>{0,0,0}, points_count};
    rz.glrotateZ(2 * M_PI / points_count);
    for(int i=0; i<points_count; i++) {
        pts[i] = p;
        p = rz * p;
    }
    return pts;
}

void bindNdraw(unsigned color, unsigned vertex, GLenum mode, int first, int count,
        unsigned indices)
{
    glBindBuffer(GL_ARRAY_BUFFER, color);
    glEnableClientState(GL_COLOR_ARRAY);
    glColorPointer(3, GL_FLOAT, 0, nullptr);

    glBindBuffer(GL_ARRAY_BUFFER, vertex);
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, nullptr); //3 float is 1 vertex stride 0,
    if ( indices ) {
        glBindBuffer (GL_ELEMENT_ARRAY_BUFFER, indices);
        glPolygonMode(GL_FRONT, GL_FILL);
        glDrawElements(mode, count, GL_UNSIGNED_INT, 0);
    } else glDrawArrays(mode, first , count) ; // mode, first , count

    glDisableClientState (GL_COLOR_ARRAY);
    glDisableClientState (GL_VERTEX_ARRAY);
}

string read_file ( string file )
{
    string r;
    char c;
    ifstream f( file );
    while(f >> noskipws >> c) r += c;
    return r;
}

unsigned make_shader_program(const char* vsh, const char* fsh, const char* a_pos, const char*
    a_color )
{
    string v_shader = read_file (vsh);
    string f_shader = read_file (fsh);
    auto* vp = v_shader . data () ;
    auto* fp = f_shader . data () ;
    const char** vertex_shader = &vp;
    const char** fragment_shader = &fp;

    unsigned vs = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vs, 1, vertex_shader , NULL);
    glCompileShader(vs);
    int status ;
    glGetShaderiv(vs, GL_COMPILE_STATUS, &status);
    if ( status == GL_FALSE) cerr << "compile error in vertex shader" << endl;

    unsigned fs = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource( fs , 1, fragment_shader, NULL);
    glCompileShader(fs );
    glGetShaderiv( fs , GL_COMPILE_STATUS, &status);
    if ( status == GL_FALSE) cerr << "compile error in fragment shader" << endl;
    unsigned shader_program = glCreateProgram();
    glAttachShader(shader_program, vs);
    glAttachShader(shader_program, fs);
    glBindAttribLocation (shader_program, 0, a_pos);
    glBindAttribLocation (shader_program, 1, a_color);
    glLinkProgram(shader_program);

    // linking error message
    int linked = 0;
    glGetProgramiv(shader_program, GL_LINK_STATUS, &linked);
    if (! linked) {
        int infolen = 0;
        glGetProgramiv(shader_program, GL_INFO_LOG_LENGTH, &infolen);
        if (infolen > 1) {
            char* infoLog = (char*)malloc(sizeof(char) * infolen);
            glGetProgramInfoLog(shader_program, infolen, NULL, infoLog);
            cout << "error linking program\n" << infoLog << endl;
            free (infoLog);
        }
        glDeleteProgram(shader_program);
        return 0;
    }
    return shader_program;
}

void replace (char* str, string anchor, const Matrix<float>& mat)
{
    auto m = mat.transpose ();
    float* p = m.data ();
    stringstream ss;
    for(int i=0; i < m.get_width() * m.get_height () -1; i++) ss << *p++ << ',';
    ss << *p;
    string s{str};
    s.replace (s.find (anchor), anchor.size (), ss.str ());
    strcpy (str, s.data ());
}

void transfer_matrix (unsigned shader_program, const Matrix<float>& m,
        const char* var_name)
{
    int fd = glGetUniformLocation(shader_program, var_name);
    if(fd != -1) {
        auto a = m.transpose();
        glUniformMatrix4fv(fd, 1, GL_FALSE, a.data());
    }
}
```