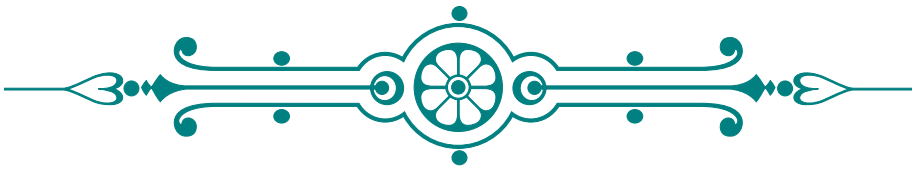


컴퓨터 그래픽스 입문

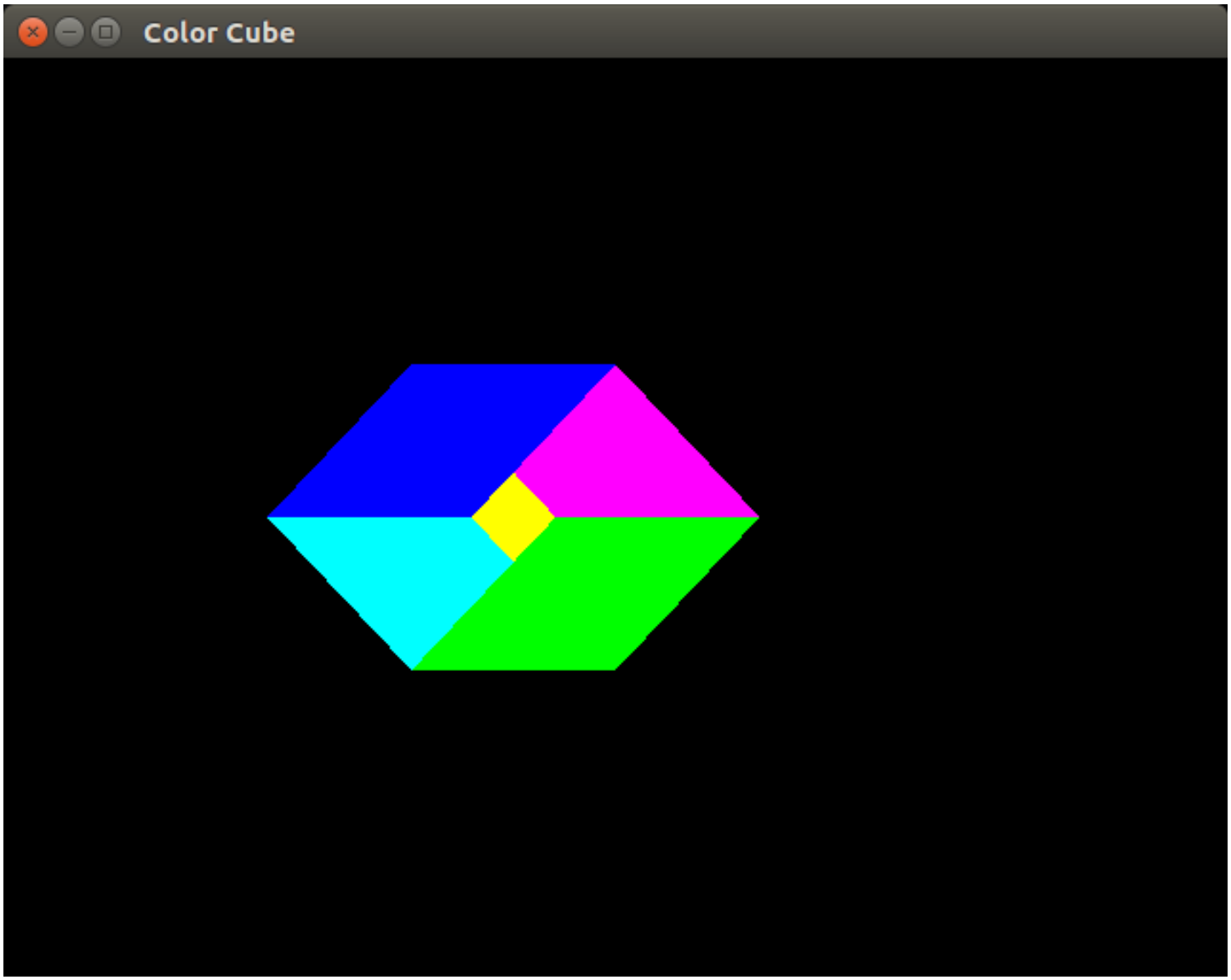
학번 : 2016110056

학과 : 불교학부

이름 : 박승원

날짜 : 2017년 3월 23일

제 1 절 Draw a cube



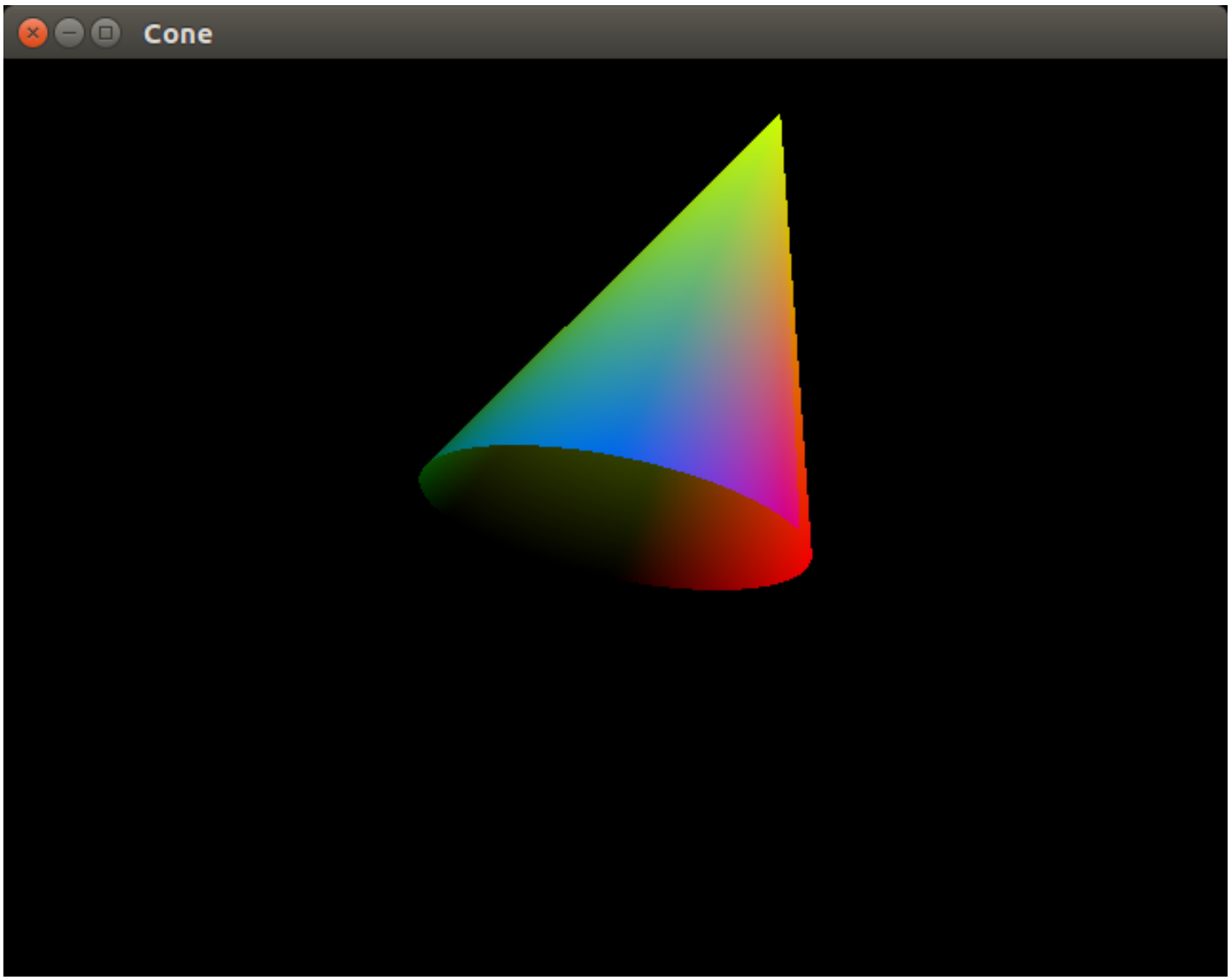

```
#include"glutil.h"
using namespace std;
extern Matrix<float> grotate;
extern Matrix<float> translate;

int main()
{
    grotate.glrotateY(M_PI/4);
    translate.gltranslate(0,0,sqrt(2));
    if(!glfwInit()) return -1;
    GLFWwindow* window = glfwCreateWindow(640, 480, "Color Cube", NULL, NULL);
    if(!glinit(window)) return -1;
    glortho(3);

    auto pl = polygon(4);
    vector<Matrix<float>> pl2;
    pl2.insert(pl2.end(), begin(pl), end(pl));
```

```
    pl = translate * pl; //z+1
    pl2.insert(pl2.end(), begin(pl), end(pl)); //append elevated 4 vertex
    for(auto& a : pl2) a = grotate * a; // rotate a little to have a good view
    valarray<float> color(72); //{}does not make 72 size — initialize_list construct
    color[slice(0,12,3)] = 1;
    color[slice(26,12,3)] = 1;
    color[slice(13,4,3)] = 1;
    color[slice(49,8,3)] = 1;

    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};
    vector<Matrix<float>> v;
    for(auto& a : idx) v.push_back(pl2[a]);

    auto fc = gl_transfer_data(color);
    auto fv = gl_transfer_data(v);

    while (!glfwWindowShouldClose(window)) {
        glClear(GL_COLOR_BUFFER_BIT);
        glBindBuffer(GL_ARRAY_BUFFER, fc);
        glEnableClientState(GL_COLOR_ARRAY);
        glColorPointer(3, GL_FLOAT, 0, nullptr);

        glBindBuffer(GL_ARRAY_BUFFER, fv);
        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, nullptr); //3 float is 1 vertex stride 0,

        glDrawArrays(GL_QUADS, 0, 24);//mode, first , count

        glDisableClientState(GL_COLOR_ARRAY);
        glDisableClientState(GL_VERTEX_ARRAY);

        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwTerminate();
}
```

다각형을 만드는 polygon이란 함수를 이용했다.
이 함수는 다각형의 정점들을 반환한다.
육면체를 만들기 위해 4각형의 정점을 생성하고 이 점들을 Z축으로 이동시킨 4점을 추가하여 8개의 정점을 만든다.
8개의 정점들을 인덱스를 이용하여 4각형을 만들기 위한 순서로 다시 벡터에 입력한다.

색을 동일하게 하기 위해 valarray를 조작한다.
gltranferdata라는 함수는 벡터나 valarray를 이용하여 데이터를 보내고 리턴값으로 vbo를 받아온다.

제 2 절 Draw a cone




```
#include" glutil .h"
using namespace std;
extern Matrix<float> grotate;
extern Matrix<float> translate ;

int main()
{
    if (! glfwInit () ) return -1;
    GLFWwindow* window = glfwCreateWindow(640, 480, "Cone", NULL, NULL);
    if (! glinit (window)) return -1;
    glortho (3) ;

    auto pl = polygon(100); // return 100 circle vertexes
```

```
    vector<Matrix<float>> pl2;
    pl2.push_back({0,0,3}); // top point of cone
    pl2.insert(pl2.end(), begin(pl), end(pl));
    pl2.push_back(pl[0]);
    grotate.glrotateX(5*M_PI/4 + 0.1);
    for(auto& a : pl2) a = grotate * a; // rotate a little to have a good view
    grotate.E();
    auto fv = gl_transfer_data(pl2);
    while (!glfwWindowShouldClose(window)) {
        glClear(GL_COLOR_BUFFER_BIT);

        glBindBuffer(GL_ARRAY_BUFFER, fv);
        glEnableClientState(GL_COLOR_ARRAY);
        glColorPointer(3, GL_FLOAT, 0, nullptr);

        glBindBuffer(GL_ARRAY_BUFFER, fv);
        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, nullptr); //3 float is 1 vertex stride 0,

        glDrawArrays(GL_TRIANGLE_FAN, 0, 102);//mode, first , count

        glDisableClientState(GL_COLOR_ARRAY);
        glDisableClientState(GL_VERTEX_ARRAY);

        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwTerminate();
}
```

polygon함수로 100각형을 생성한다. 인간의 눈에는 원으로 보인다.
앞 뒤에 원뿔의 꼭지점과 원의 시작점을 추가한다.
색의 데이터를 점의 데이터와 같게 하여 변화를 준다.

Listing 1: glutil.h

```
#pragma once
#include<GL/glew.h>
#include<GLFW/glfw3.h>
#include<vector>
#include<valarray>
#include<type_traits>
#include"matrix.h"
```

```
void glortho(float r);
void glcolor (unsigned char r, unsigned char g, unsigned char b, unsigned char a);
bool glinit (GLFWwindow* window);
std :: valarray<Matrix<float>> polygon(int points_count=100, float r=1);

template <typename T>
unsigned int gl_transfer_data (T* begin, T* end, GLenum mode = GL_ARRAY_BUFFER)
{
    int sz = end - begin;
    unsigned int vbo;
    glGenBuffers (1, &vbo);
    glBindBuffer(mode, vbo);

    T ar[sz];
    memcpy(ar, begin, sizeof(ar));
    glBufferData(mode, sizeof(ar), ar, GL_STATIC_DRAW);
    return vbo;
}

static void mcopy(float* p, const Matrix<float>& m) {
    memcpy(p, m.data(), 3 * sizeof(float));
}
static void mcopy(float* p, float m) {
    *p = m;
}
template <typename T>
unsigned int gl_transfer_data (const T& v, GLenum mode = GL_ARRAY_BUFFER)
{// v should offer operator []
    int dim = 1;
    int sz = v.size();
    unsigned int vbo;
    glGenBuffers (1, &vbo);
    glBindBuffer(mode, vbo);

    if(std :: is_class <typename T::value_type>::value) dim = 3; // if Matrix
    float ar[sz * dim];
    for(int i=0; i<sz; i++) mcopy(ar+i*dim, v[i]);
    glBufferData(mode, sizeof(ar), ar, GL_STATIC_DRAW);
    return vbo;
}
```

Listing 2: callbacks.cc : glutil implementation

```
#include<GL/glew.h>
#include<GLFW/glfw3.h>
#include<iostream>
#include<vector>
#include<valarray>
#include"matrix.h"
using namespace std;
Matrix<float> translate {4,4};
Matrix<float> grotate{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) {
    if (key == GLFW_KEY_LEFT && action == GLFW_PRESS) translate[4][1]-=0.01;
    if (key == GLFW_KEY_DOWN && action == GLFW_PRESS) translate[4][2]-=0.01;
    if (key == GLFW_KEY_RIGHT && action == GLFW_PRESS) translate[4][1]+=0.01;
    if (key == GLFW_KEY_UP && action == GLFW_PRESS) translate[4][2]+=0.01;

    if (key == GLFW_KEY_W && action == GLFW_PRESS) grotate *= m.glrotateX(0.01);
    if (key == GLFW_KEY_A && action == GLFW_PRESS) grotate *= m.glrotateY(-0.01);
    if (key == GLFW_KEY_S && action == GLFW_PRESS) grotate *= m.glrotateX(-0.01);
    if (key == GLFW_KEY_D && action == GLFW_PRESS) grotate *= m.glrotateY(0.01);
    if (key == GLFW_KEY_SPACE && action == GLFW_PRESS) grotate.E();
    if (key == GLFW_KEY_J && action == GLFW_PRESS) camera_x-=0.1;
    if (key == GLFW_KEY_K && action == GLFW_PRESS) camera_y-=0.1;
    if (key == GLFW_KEY_L && action == GLFW_PRESS) camera_x+=0.1;
    if (key == GLFW_KEY_I && action == GLFW_PRESS) camera_y+=0.1;
}

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); // white background

    glewExperimental = true; // Needed for core profile
    if (glewInit() != GLEW_OK) {
        fprintf(stderr, "Failed to initialize GLEW\n");
        glfwTerminate();
        return false;
    }
    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;
}
```