컴퓨터 그래픽스 입문

학번 : 2016110056

학과 : 불교학부

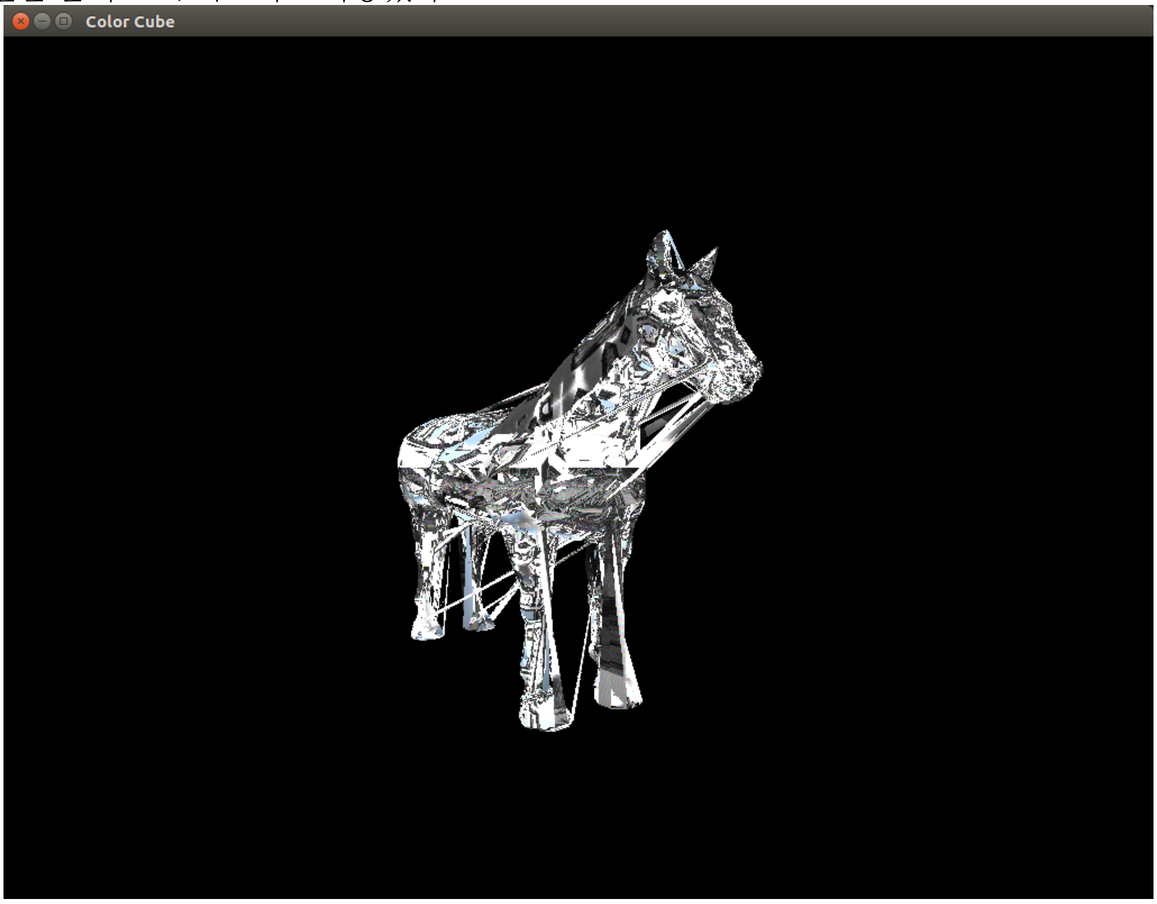
이름 : 박승원

날짜 : 2017년 5월 18일

Lab 11. Texturing

```
Lab 11. Texturing

Programming Practice
1. Download a 3D model from thingiverse.

2. Convert it to an OBJ file.

3. Render it with phong shading. (3pts)

4. Map a texture on it. You may use any image you like. (4pts)

5. Animate texture. (3pts)

* Note: you are going to study bump mapping next week.
```

horse.obj 파일을 다운 받아서, 텍스쳐를 입혔다. cubeMap 으로 하려고 했으나, 실패하고, 2D 맵으로 했다. 첨부한 소스 파일의 중간 쯤에 텍스쳐를 불러오는 부분이 있다. opencv 라이브러리의 루틴을 호출하여, 이미지 파일을 불러오고, 텍스쳐로 사용했다.



Listing 1: texture loading

```
#include<fstream>
#include<GL/glew.h>
```

```
#include<highgui.h>
#include"globj.h"
using namespace std;

GLObjs::GLObjs(unsigned prog)
{
    shader_program_ = prog;
    glPolygonMode(GL_FRONT, GL_FILL);
}

GLObjs& GLObjs::operator+=(GLObject& r)
{
    r.normals();
    r.colors(); // if texture
    if(r.indices_.empty()) for(int i=0; i<r.vertexes_.size(); i++) // if no indices
        r.indices_.push_back(i);
//  for(int i=0; i<10; i++) cout << r.vertexes_[i] << r.colors_[i] << r.normals_[i];
    auto sz = vertexes_.size();
    vertexes_.insert(vertexes_.end(), r.vertexes_.begin(), r.vertexes_.end());
    colors_.insert(colors_.end(), r.colors_.begin(), r.colors_.end());
    normals_.insert(normals_.end(), r.normals_.begin(), r.normals_.end());
    auto idx = r.indices_;
    for(auto& a : idx) a += sz;
    indices_.insert(indices_.end(), idx.begin(), idx.end());

    index_chunks_.push_back(r.indices_.size());
    modes_.push_back(r.mode_);
    matrixes_.push_back(r.matrix_);
    texture_files_.push_back(r.texture_file_);
}

void GLObjs::transfer_all()
{
    read_texture();
    vbo[0] = transfer_data(vertexes_, "vertexes_");
    vbo[1] = transfer_data(colors_, "colors_");
    vbo[2] = transfer_data(normals_, "normals_");
    vbo[3] = indices(indices_);
    cout << indices_.size() << endl;
}

unsigned GLObjs::read_texture()
```

```
{///cube map texture
    using namespace cv;
    int cubewall[6] = {
        GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
        GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
        GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z
    };
    unsigned vbo[2];
    glGenTextures(2, vbo);
    glActiveTexture (GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_CUBE_MAP, vbo[0]);
    for(int i=0; i<6; i++) {
        Mat image = imread("b.jpg");
//    int sq = min(image.cols/4, image.rows/3);
//    Rect                      r1(sq, 0, sq, sq),
//          r2(0, sq, sq, sq), r3(sq, sq, sq, sq), r4(2*sq,sq,sq,sq), r5(3*sq,sq,sq,sq),
//                              r6(sq, 2*sq, sq, sq);
    // glActiveTexture (GL_TEXTURE0);
        glTexImage2D(cubewall[i], 0, GL_RGB, image.cols, image.rows, 0,
                GL_BGR, GL_UNSIGNED_BYTE, image.data);
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glUniform1i(glGetUniformLocation(shader_program_, "TEXTURE"), 0);

    glActiveTexture (GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, vbo[1]);
    Mat im = imread("b.jpg");
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, im.cols, im.rows, 0, GL_BGR, GL_UNSIGNED_BYTE, im.
        data);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glUniform1i(glGetUniformLocation(shader_program_, "TEXTURE2D"), 1);

    return vbo[0];
}

unsigned GLObjs::indices(const vector<unsigned>& v, unsigned vbo)
{
    if (!vbo) glGenBuffers(1, &vbo);
    glBindBuffer (GL_ELEMENT_ARRAY_BUFFER, vbo);
    glBufferData (GL_ELEMENT_ARRAY_BUFFER, sizeof(unsigned) * v.size(),
                v.data() , GL_STATIC_DRAW);
    return vbo;
}

Matrix<float> GLObjs::operator[](int n)
{
    return matrixes_[n];
}

void GLObjs::operator()(int n)
{
    unsigned offset = 0;
    for(int i=0; i<n; i++) offset += index_chunks_[i];
    glDrawElements(modes_[n], index_chunks_[n], GL_UNSIGNED_INT,
            (void*)( offset * sizeof(unsigned)));
}

unsigned GLObjs::transfer_data (const vector<Matrix<float>>& v, const char* var,
        unsigned vbo)
{
    int sz = v.size();
    if(!vbo) glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);

    float ar[sz * 3];
    for(int i=0; i<sz; i++) memcpy(ar + 3*i, v[i].data(), 3 * sizeof(float));
    glBufferData(GL_ARRAY_BUFFER, sizeof(ar), ar, GL_STATIC_DRAW);

    unsigned loc = glGetAttribLocation(shader_program_, var);
    glEnableVertexAttribArray(loc);
    cout << var << " : " << loc << ", " << v.size() <<endl;
    glVertexAttribPointer(loc, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
    return vbo;
}
```

Listing 2: main함수

```
#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);

    auto vv = polygon(3, 1) ;
    GLObject obj3d;
//  obj3d. vertexes (vv);
    auto sz = obj3d. read_obj_file (av [1]) ;
//  obj3d. colors ({ sz , {.5,1,.5}}) ;
    obj3d. texture_file ("b.jpg") ;
    GLObjs stage{shader_program};
    stage += obj3d;
    stage . transfer_all () ;
    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");

    while (! glfwWindowShouldClose(window)) {
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        transfer_matrix (shader_program, light . glprojection (−1,1,−1,1,−1,1) *
                KeyBindMatrix, "KeyBindMatrix");

        stage (0) ;

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

Listing 3: fragment shader

```
#version 130
uniform mat4 LIGHT;
uniform mat4 KeyBindMatrix;
uniform samplerCube TEXTURE;
uniform sampler2D TEXTURE2D;
in vec3 color;
in vec4 NN;
in vec3 pos;
out vec4 f_color;

vec3 ambient = LIGHT[0].xyz;
vec3 diffuse = LIGHT[1].xyz;
vec3 specular = LIGHT[2].xyz;
vec3 light_pos = LIGHT[3].xyz;
vec3 view = vec3 (0,0,2) ;

void main() {
    vec4 lp = vec4( light_pos , 1);
    lp = KeyBindMatrix * lp;
    vec3 N = normalize(NN.xyz);
    vec3 L = normalize(lp.xyz - pos);
    vec3 V = normalize(view);
    vec3 R = -V + 2 * dot(N, V) * N;

//  vec3 color2 = vec3(0.0f,0.0f,1.0f);
//  vec4 texture = textureCube(TEXTURE, color2);
    vec4 texture = texture(TEXTURE2D, color.xy + vec2(0.1f,0.1f));
    f_color = vec4( texture.xyz * (0.1 * ambient + 0.8 * diffuse * dot(L, N) + specular * pow(dot(V,
        R), 25)), 1.0);
}
```