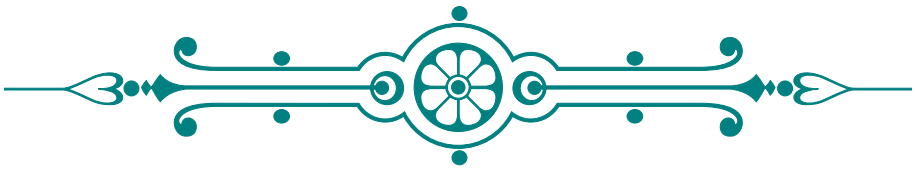


컴퓨터 그래픽스 입문

학번 : 2016110056

학과 : 불교학부

이름 : 박승원

날짜 : 2017년 5월 25일

Lab 11. Texturing

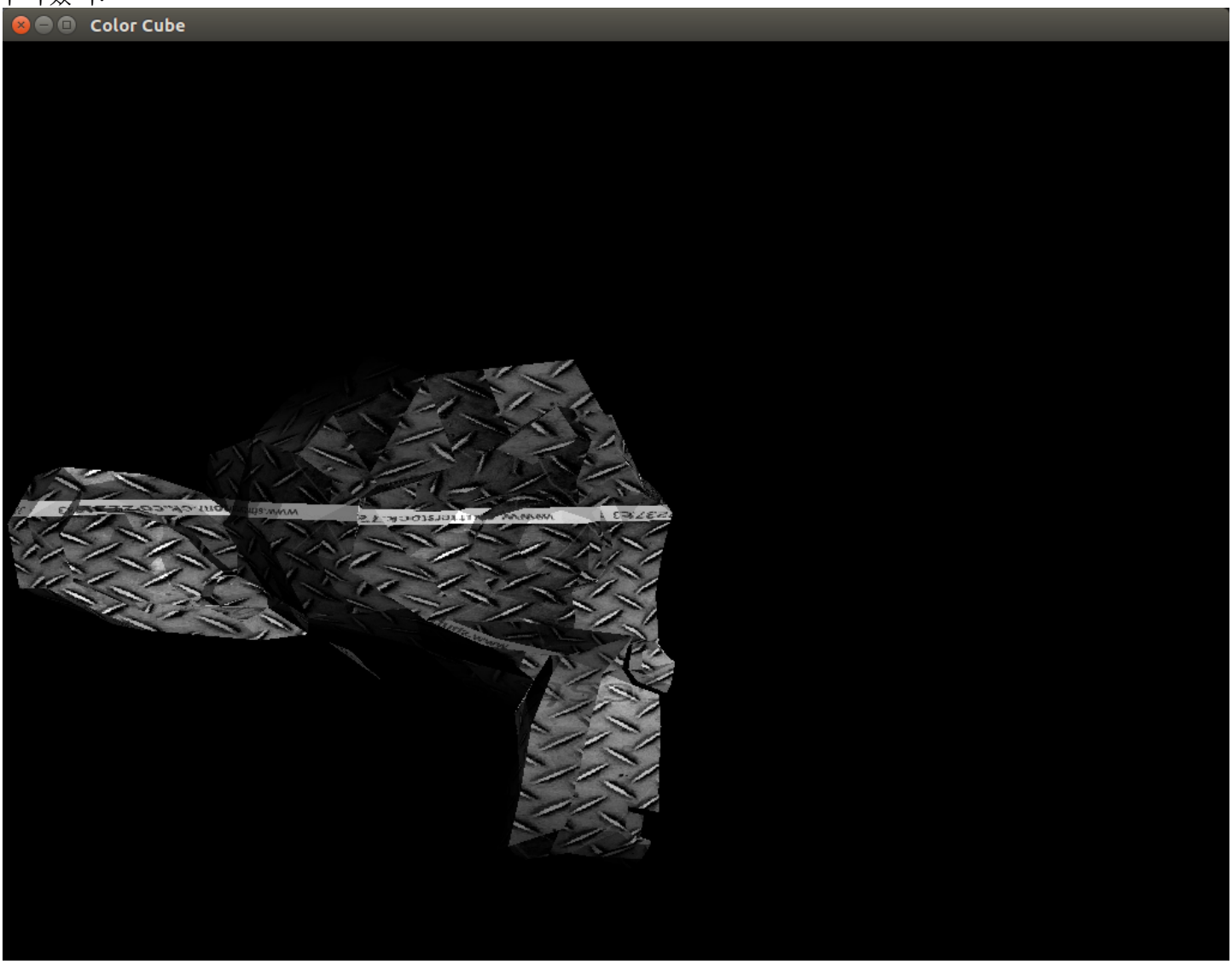
`Pick a relatively low resolution 3D model from the thingiverse.`

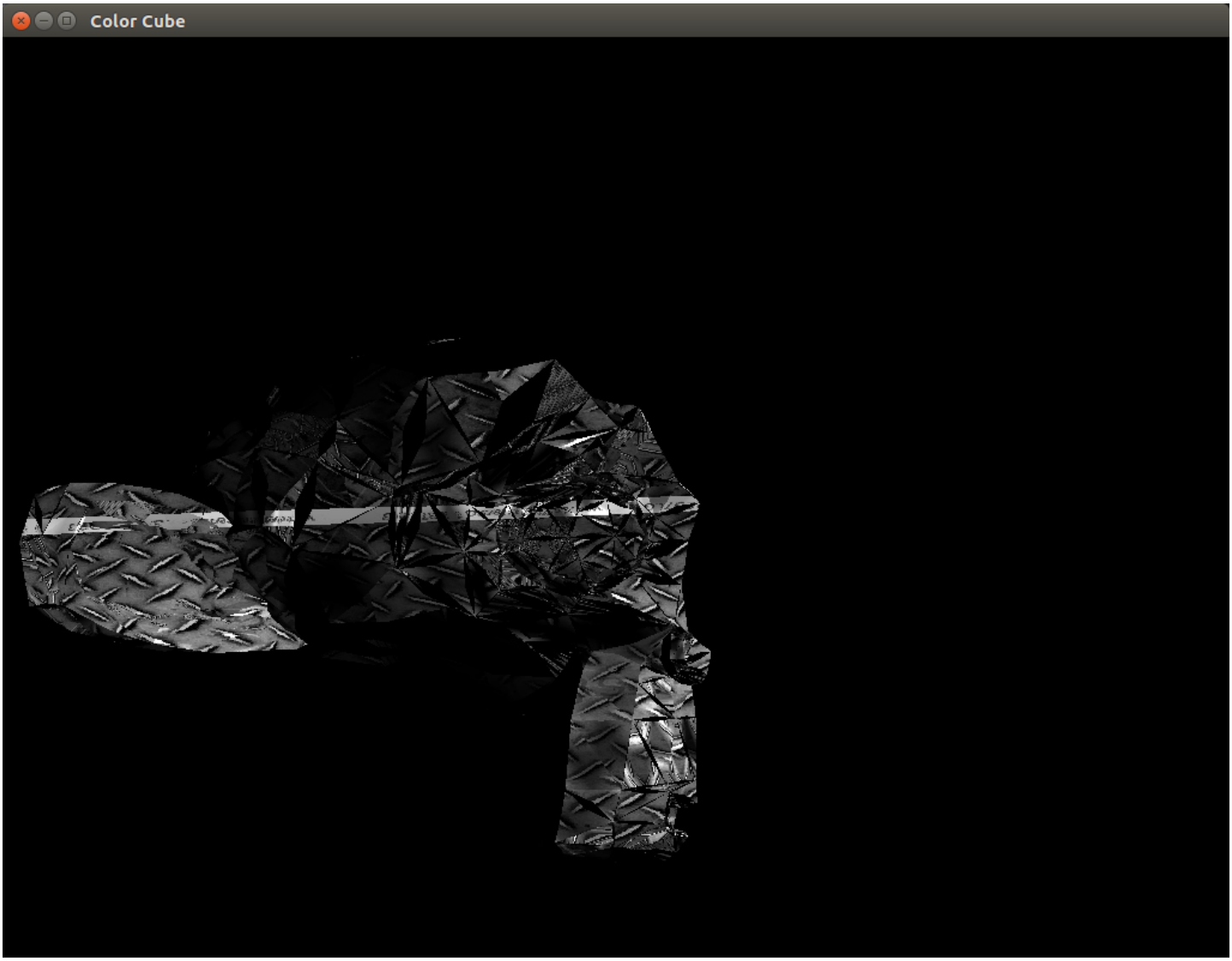
`Step 1. Apply the butterfly subdivision. Show the difference before and after`

`Step 2. Apply the Loop subdivision. Show the difference before and after apply`

`Step 3. Show the difference between the butterfly and the Loop subdivision. (4`

블렌더의 대표 마스코트인 몽키를 triangle subdivision했다. 위가 loop이고, 아래가 butterfly이다. 그런데, normal에서의 버그가 있어서인지, 트라이앵글을 나누어서 노말을 자체적으로 계산했을때, 원숭이의 얼굴 반만이 나왔다.






```
#include<fstream>
#include<GL/glew.h>
#include<highgui.h>
#include"globj.h"
using namespace std;

GLObject::GLObject() : matrix_{4,4} {
    matrix_.E();
}
void GLObject::matrix(const Matrix<float>& m) { matrix_ = m; }
void GLObject::mode(GLenum md) { mode_ = md; }
void GLObject::vertexes(const vector<Matrix<float>>& v) { vertexes_ = v; }
void GLObject::vertexes(vector<Matrix<float>>&& v) { vertexes_ = move(v); }
void GLObject::colors(const vector<Matrix<float>>& v) { colors_ = v; }
void GLObject::colors(vector<Matrix<float>>&& v) { colors_ = move(v); }
void GLObject::indices(const vector<unsigned>& v) { indices_ = v; }
void GLObject::indices(vector<unsigned>&& v) { indices_ = move(v); }
void GLObject::texture_file(string f) { texture_file_ = f; }
void GLObject::subdiv_triangle()
```

```
{
    vector<Matrix<float>> v;
    vector<unsigned> ix;
    try {
        for(int i=0; i<indices_.size(); i+=3) {
            unsigned a = indices_[i];
            unsigned b = indices_[i+1];
            unsigned c = indices_[i+2];
            v.push_back(vertexes_[a]);
            v.push_back(vertexes_[b]);
            v.push_back(vertexes_[c]);
            v.push_back((vertexes_[a] + vertexes_[b]) * 0.5f);
            v.push_back((vertexes_[b] + vertexes_[c]) * 0.5f);
            v.push_back((vertexes_[c] + vertexes_[a]) * 0.5f);
            int sz = ix.size();
            unsigned rel_pos[] = {0, 3, 5, 3, 1, 4, 3, 4, 5, 5, 4, 2};
            for(unsigned i : rel_pos) ix.push_back(sz + i);
        }
    } catch(const char* e) { cerr << e << endl; }
    vertexes_ = v;
    indices_ = ix;
    normals_.clear();
}

void GLObject::butterfly()
{
    vector<Matrix<float>> v;
    vector<unsigned> ix;
    try {
        for(int i=0; i<indices_.size(); i+=3) {
            unsigned a = indices_[i];
            unsigned b = indices_[i+1];
            unsigned c = indices_[i+2];
            v.push_back(vertexes_[a]);
            v.push_back(vertexes_[b]);
            v.push_back(vertexes_[c]);
            auto d1 = vertexes_[a] - vertexes_[b];
            auto d2 = vertexes_[c] - vertexes_[b];
            auto d3 = vertexes_[a] - vertexes_[c];
            auto c1 = cross(vertexes_[a], vertexes_[b]);
            auto c2 = cross(vertexes_[b], vertexes_[c]);
            auto c3 = cross(vertexes_[c], vertexes_[a]);
```

```
            v.push_back((vertexes_[a] + vertexes_[b]) * 0.5f + c1 * 0.1 * d1.distance());
            v.push_back((vertexes_[b] + vertexes_[c]) * 0.5f + c2 * 0.1 * d2.distance());
            v.push_back((vertexes_[c] + vertexes_[a]) * 0.5f + c3 * 0.1 * d3.distance());
            int sz = ix.size();
            unsigned rel_pos[] = {0, 3, 5, 3, 1, 4, 3, 4, 5, 5, 4, 2};
            for(unsigned i : rel_pos) ix.push_back(sz + i);
        }
    } catch(const char* e) { cerr << e << endl; }
    vertexes_ = v;
    indices_ = ix;
    normals_.clear();
}

void GLObject::normals()
{///should come after setting mode
    if(normals_.size() == vertexes_.size()) return;
    normals_.resize(vertexes_.size());
    int face;
    switch(mode_) {
        case GL_TRIANGLES: face = 3; break;
        case GL_QUADS: face = 4; break;
        default: face = 3;
    }
    try{
        for(int i=0; i<vertexes_.size(); i+=face) {
            auto v1 = vertexes_[i+1] - vertexes_[i];
            auto v2 = vertexes_[i+2] - vertexes_[i];
            auto n = cross(v1, v2);
            for(int j=0; j<face; j++) normals_[i+j] = normals_[i+j] + n;
        }
    } catch(const char* e) { cerr << e << endl; }
    for(auto& a : normals_) {
        a = a * (1.0f / sqrt(a[1][1]*a[1][1] + a[1][2]*a[1][2] + a[1][3]*a[1][3]));
        a[1][4] = 1;
    }
}

Matrix<float> GLObject::cross(const Matrix<float>& v1, const Matrix<float>& v2)
{
    Matrix<float> m{v1[1][2]*v2[1][3] - v1[1][3]*v2[1][2],
                    v1[1][3]*v2[1][1] - v1[1][1]*v2[1][3],
                    v1[1][1]*v2[1][2] - v1[1][2]*v2[1][1]};
```

```
    float r = sqrt(m[1][1] * m[1][1] + m[1][2] * m[1][2] + m[1][3] * m[1][3]);
    m = m * (1.0f/r);
    m[1][4] = 1;
    return m;
}

unsigned GLObject:: read_obj_file ( string file )
{
    int face = 0;
    string s;
    ifstream f( file );
    while(getline(f, s)) {
        stringstream ss{s};
        ss >> s;
        if(s == "v") {
            float x, y, z;
            ss >> x >> y >> z;
            vertexes_.push_back(Matrix<float>{x,y,z});
        } else if(s == "f") {
            while(getline(ss, s, '/')) {
                indices_.push_back(stoi(s)-1);
                getline(ss, s, ' ');
                face++;
            }
            if(face == 3) mode(GL_TRIANGLES);
            else if(face == 4) mode(GL_QUADS);
        } else if(s == "vn") {
            float x, y, z;
            ss >> x >> y >> z;
            normals_.push_back(Matrix<float>{x, y, z});
        }
    }
    cout << file << "\'s indices size : " << indices_.size() << endl;
    return vertexes_.size();
}


void GLObject:: colors ()
{
    if( texture_file_ == "") return;
    colors_.clear();
    normalize_vertex();
```

```
    for(int i=0; i<normals_.size(); i++) {
        float x = normals_[i][1][1];
        float y = normals_[i][1][2];
        float z = normals_[i][1][3]; // find biggest abs->vertex coord
        float vx = vertexes_[i][1][1];
        float vy = vertexes_[i][1][2];
        float vz = vertexes_[i][1][3];

//      if(abs(x) > abs(y) && abs(x) > abs(z)) colors_.push_back({x>0?1:-1, vy, vz});
//      else if(abs(y)>abs(z) && abs(y)>abs(x)) colors_.push_back({vx, y>0?1:-1, vz});
//      else colors_.push_back({vx, vy, z>0?1:-1});

        if(abs(x) > abs(y) && abs(x) > abs(z)) // map to 육면체 전개도
            colors_.push_back({x > 0 ? 0.5 - (vz + 1) / 8 : (vz + 1) / 8,
                    1.0f / 3 + (vy + 1) / 6, 0});
        else if(abs(y)>abs(z) && abs(y)>abs(x))
            colors_.push_back({0.25 + (vx + 1) / 8,
                    y > 0 ? (vz + 1) / 6 : 2.0f / 3 - (vz + 1) / 6, 0});
        else colors_.push_back({z > 0 ? 0.25 + (vx + 1) / 8 : 0.75 - (vx + 1) / 8,
                1.0f / 3 + (vy + 1) / 6, 0});
    }

    cout << "colors_ size : " << colors_.size() << endl;
    for(auto& a : colors_) {
        assert(a[1][1] >= -1 && a[1][1] <= 1);
        assert(a[1][2] >= -1 && a[1][2] <= 1);
        assert(a[1][3] >= -1 && a[1][3] <= 1);
    }
}

void GLObject::normalize_vertex()
{
    float xmin, xmax, ymin, ymax, zmin, zmax;
    xmin = xmax = vertexes_[0][1][1];
    ymin = ymax = vertexes_[0][1][2];
    zmin = zmax = vertexes_[0][1][3];
    for(auto& a : vertexes_) {
        if(xmin > a[1][1]) xmin = a[1][1];
        if(xmax < a[1][1]) xmax = a[1][1];
        if(ymin > a[1][2]) ymin = a[1][2];
        if(ymax < a[1][2]) ymax = a[1][2];
        if(zmin > a[1][3]) zmin = a[1][3];
```

```
        if(zmax < a[1][3]) zmax = a[1][3];
    }
    float x = xmax - xmin;
    float y = ymax - ymin;
    float z = zmax - zmin;
    float rate = max(x, max(y,z));
    for(auto& a : vertexes_) {
        a[1][1] -= xmin;
        a[1][2] -= ymin;
        a[1][3] -= zmin;
        for(int i=1; i<4; i++) {
            a[1][i] /= rate;
            a[1][i] -= 0.5;
            a[1][i] *= 2;
        }
    }
    for(auto& a : vertexes_) {
        assert(a[1][1] >= -1 && a[1][1] <= 1);
        assert(a[1][2] >= -1 && a[1][2] <= 1);
        assert(a[1][3] >= -1 && a[1][3] <= 1);
    }
}
```

Listing 1: 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;

    GLObject obj3d;
    obj3d.read_obj_file("monkey.obj");
//  obj3d.subdiv_triangle();
    obj3d.butterfly();
```

```
    obj3d. texture_file (" steel .png");
    GLObjs stage;
    stage += obj3d;
    stage. transfer_all ();

    Matrix<float> m{4,4};
    while (!glfwWindowShouldClose(window)) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        stage.matrix(KeyBindMatrix * m.glscale (0.8,0.8,0.8) );
        stage (0) ;

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