컴퓨터그래픽스 과제3 텍스쳐 매핑

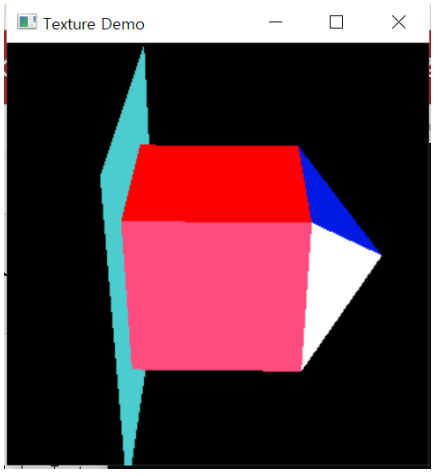
학과, 학번 : 컴퓨터공학과 21011575

이름 : 박준형

과제3에서는 텍스쳐 매핑을 통하여 원래 있떤 opengl 화면을 텍스쳐가 덮어 그래픽을 표시할 수 있도록 하는 과제였습니다.

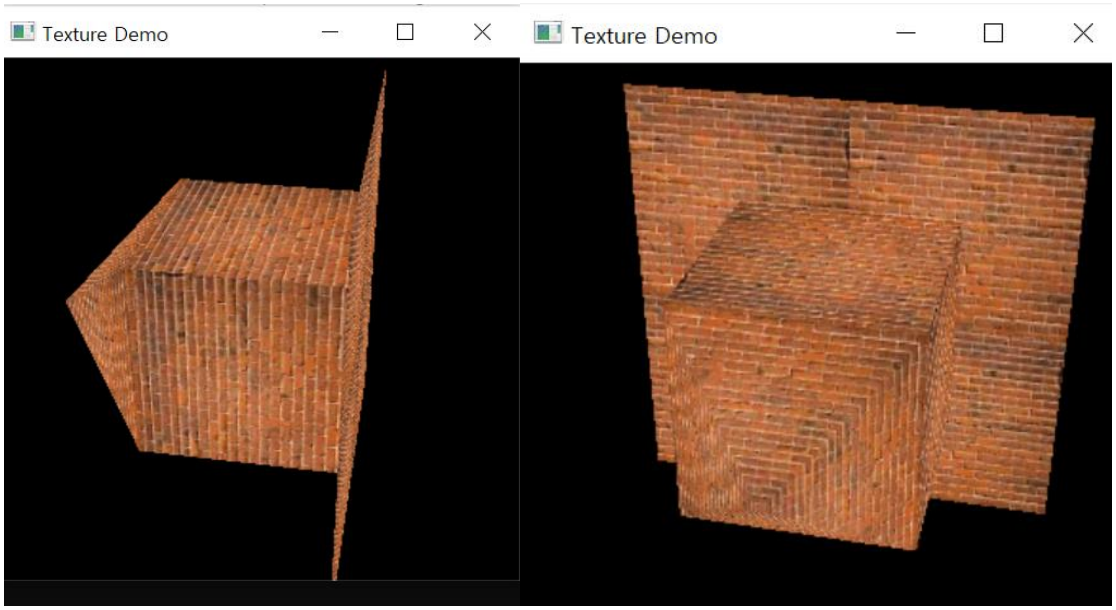
이 보고서에서는 교수님의 수업자료 9주차의 texturingcube.cpp와 dice_texture.ppm을 이용한 예제코드를 토대로 하여 수정해가며 벽돌집 텍스쳐 그래픽을 구현하였습니다.

1) 예제코드를 토대로 하되 예제코드에서는 육면체 모양 그래픽이었기 때문에 좌표를 수정해주어 집의 지붕부분, 벽의 벽면 부분, 벽면 아랫부분과 연결된 땅 부분으로 나누어 좌표를 설정하였고 텍스쳐 매핑을 하기 전 색깔을 각 면마다 넣어주어 좌표설정이 제대로 되었는지 확인 하였습니다.



2) 무료 벽돌 이미지를 인터넷에서 다운받아 ppm파일로 바꾼후 텍스쳐 매핑을 실행시켜 벽돌이 그래픽을 싸도록 설정하였습니다.

3) 이외에 gluLookat을 이용하여 첫 시점을 고정시켰고, 추가로 timer callback을 넣어 시간에 따라서 벽의 이곳저곳을 다 볼 수 있도록 회전시킬 수 있게 설정하도록 구현하였습니다.



4. 아래는 코드와 해당 벽돌 이미지 ppm파일을 따로 보내지면 혹시나 코드파일이 열리지 않을 경우를 대비해 코드파일을 따로 보내드립니다. (만약 코드 파일이 열리면 아래는 보실 필요 없습니다.)

```
#include <GL/freeglut.h>

#define _USE_MATH_DEFINES

#include <math.h>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <vector>

using namespace std;

class Renderer {

public:
    float t;
private:
```

```
    GLuint texID;

public:
    // constructor
    Renderer() : t(0.0), texID(0) {}

    // destructor
    ~Renderer() {
        if (texID != 0) glDeleteTextures(1, &texID);
    }

public:
    void init() {
        glEnable(GL_DEPTH_TEST);
        //벽돌로 텍스쳐 매핑
        std::string fileName("block.ppm");
        texID = loadTexture(fileName);
    }

    void resize(int w, int h) {
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(30.0, (float)w / (float)h, 0.1, 50.0);
    }

    void display() {
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        // set camera
        gluLookAt(8.0, -2.0, 5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);
        // draw scene
        glRotatef(t, 0.0f, 0.0f, 1.0f);
        drawTexturedCube();
    }

private:
    // 성공 시 적절한 텍스쳐 아이디 반환하기
    GLuint loadTexture(std::string& filename) {

        unsigned width;
        unsigned height;
        int level = 0;
        int border = 0;
        std::vector<unsigned char> imgData;

        // 이미지 데이터 로드하기
        if (!loadPPMImageFlipped(filename, width, height, imgData)) return 0;

        // data is aligned in byte order
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
```

```
        //텍스쳐 아이디 요청
        GLuint textureID;
        glGenTextures(1, &textureID);

        // bind texture
        glBindTexture(GL_TEXTURE_2D, textureID);

        //텍스쳐 필터링 정의
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

        //텍스쳐 매핑을 통해 텍스쳐 컬러가 원래 색을 덮게 된다.
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE); //GL_MODULATE

        // specify the 2D texture map
        glTexImage2D(GL_TEXTURE_2D, level, GL_RGB, width, height, border, GL_RGB,
GL_UNSIGNED_BYTE, &imgData[0]);

        // 텍스쳐 아이디 반환
        return textureID;
    }

    //집의 지붕, 벽면, 벽면과 맞닿아있는 땅의 좌표를 찍고 텍스쳐 매핑한다.
    void drawTexturedCube() {
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texID);
        //벽면1
        glColor3f(1.0f, 1.0f, 0.0f);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 1.0f);
        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, -1.0f);
        glTexCoord2f(1.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glTexCoord2f(0.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glEnd();
        //벽면2
        glColor3f(1.0f, 0.0f, 0.0f);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 1.0f);
        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 1.0f);
        glTexCoord2f(1.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glTexCoord2f(0.0f, 1.0f);
        glVertex3f(1.0f, 1.0f, 1.0f);
        glEnd();
        //벽면3
        glColor3f(0.0f, 1.0f, 1.0f);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, -1.0f);
```

```
glTexCoord2f(1.0f, 0.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glEnd();
//벽면4
glColor3f(1.0f, 0.3f, 0.5f);
glBegin(GL_QUADS);
glTexCoord2f(0.0f, 0.0f);
glVertex3f(1.0f, -1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f);
glVertex3f(1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f);
glVertex3f(1.0f, 1.0f, 1.0f);
glEnd();

//지붕1
glColor3f(0.0f, 0.1f, 0.9f);
glBegin(GL_TRIANGLES);
glTexCoord2f(0.5f, 1.0f);
glVertex3f(0.0f, 2.0f, 0.0f);
glTexCoord2f(0.0f, 0.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glTexCoord2f(1.0f, 0.0f);
glVertex3f(1.0f, 1.0f, 1.0f);
glEnd();
//지붕2
glColor3f(1.0f, 1.0f, 1.0f);
glBegin(GL_TRIANGLES);
glTexCoord2f(0.5f, 1.0f);
glVertex3f(0.0f, 2.0f, 0.0f);
glTexCoord2f(1.0f, 0.0f);
glVertex3f(1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glEnd();
//지붕3
glColor3f(0.7f, 0.5f, 0.3f);
glBegin(GL_TRIANGLES);
glTexCoord2f(0.5f, 1.0f);
glVertex3f(0.0f, 2.0f, 0.0f);
glTexCoord2f(0.0f, 0.0f);
glVertex3f(1.0f, 1.0f, -1.0f);
glTexCoord2f(1.0f, 0.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glEnd();
//지붕4
glColor3f(0.0f, 0.0f, 0.0f);
glBegin(GL_TRIANGLES);
glTexCoord2f(0.5f, 1.0f);
glVertex3f(0.0f, 2.0f, 0.0f);
```

```
        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glEnd();

        //바닥(땅)
        glColor3f(0.3f, 0.8f, 0.8f);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-2.0f, -1.0f, -2.0f);
        glTexCoord2f(0.0f, 2.0f);
        glVertex3f(-2.0f, -1.0f, 2.0f);
        glTexCoord2f(2.0f, 2.0f);
        glVertex3f(2.0f, -1.0f, 2.0f);
        glTexCoord2f(2.0f, 0.0f);
        glVertex3f(2.0f, -1.0f, -2.0f);
        glEnd();


        glDisable(GL_TEXTURE_2D);
    }

    bool loadPPMImageFlipped(std::string& filename, unsigned& width, unsigned& height,
std::vector<unsigned char>& imgData) {

        ifstream input(filename.c_str(), ifstream::in | ifstream::binary);
        if (!input) { // cast istream to bool to see if something went wrong
            cerr << "Can not find texture data file " << filename.c_str() << endl;
            return false;
        }
        input.unsetf(std::ios_base::skipws);

        string line;
        input >> line >> std::ws;
        if (line != "P6") {
            cerr << "File is not PPM P6 raw format" << endl;
            return false;
        }

        width = 0;
        height = 0;
        unsigned depth = 0;
        unsigned readItems = 0;
        unsigned char lastCharBeforeBinary;

        while (readItems < 3) {
            input >> std::ws;
            if (input.peek() != '#') {
                if (readItems == 0) input >> width;
                if (readItems == 1) input >> height;
                if (readItems == 2) input >> depth >> lastCharBeforeBinary;
                readItems++;
            }
            else { // skip comments
```

```
                std::getline(input, line);
            }
        }

        if (depth >= 256) {
            cerr << "Only 8-bit PPM format is supported" << endl;
            return false;
        }

        unsigned byteCount = width * height * 3;
        imgData.resize(byteCount);
        input.read((char*)&imgData[0], byteCount * sizeof(unsigned char));

        // vertically flip the image because the image origin
        // in OpenGL is the lower-left corner
        unsigned char tmpData;
        for (unsigned y = 0; y < height / 2; y++) {
            int sourceIndex = y * width * 3;
            int targetIndex = (height - 1 - y) * width * 3;
            for (unsigned x = 0; x < width * 3; x++) {
                tmpData = imgData[targetIndex];
                imgData[targetIndex] = imgData[sourceIndex];
                imgData[sourceIndex] = tmpData;
                sourceIndex++;
                targetIndex++;
            }
        }

        return true;
    }
};

//this is a static pointer to a Renderer used in the glut callback functions
static Renderer* renderer;

//glut static callbacks start
static void glutResize(int w, int h)
{
    renderer->resize(w, h);
}

static void glutDisplay()
{
    renderer->display();
    glutSwapBuffers();
    glutReportErrors();
}

static void timer(int v)
{
    float offset = 1.0f;
    renderer->t += offset;
    glutDisplay();
    glutTimerFunc(unsigned(20), timer, ++v);
}
```

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(320, 320);

    glutCreateWindow("Texture Demo");

    glutDisplayFunc(glutDisplay);
    //glutIdleFunc(glutDisplay);
    glutReshapeFunc(glutResize);

    renderer = new Renderer;
    renderer->init();

    glutTimerFunc(unsigned(20), timer, 0);

    glutMainLoop();
}
```