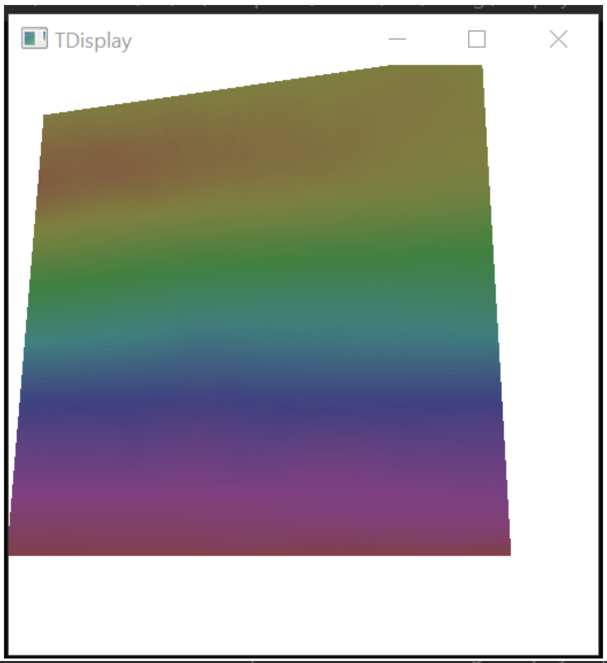
```cpp
/********************************************
 *
 *              TDisplay
 *
 *  Display a 3D texture on a plane that can
 *  move through the texture and rotate.
 *
 *****************************************/
#include <Windows.h>
#include <gl/glew.h>
#define GLFW_DLL
#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "shaders.h"
#include <stdio.h>

GLuint program;         // shader programs
GLuint objVAO;          // the data to be displayed
int triangles;          // number of triangles
int window;
GLuint texName;         // texture name
glm::mat4 projection;   // projection matrix

#define XDIM    500
#define YDIM    500
#define ZDIM    100

unsigned char texture[ZDIM][YDIM][XDIM][3];

float eyex, eyey, eyez; // eye position
double theta, phi;
double r;

float dy = 0.0;
float angle = 0.0;

char* Shader = "";

void read_texture(char *base) {
    char filename[256];
    FILE *infile;

    sprintf(filename,"%s.tex",base);
    infile = fopen(filename, "rb");
    if(infile == NULL)
```

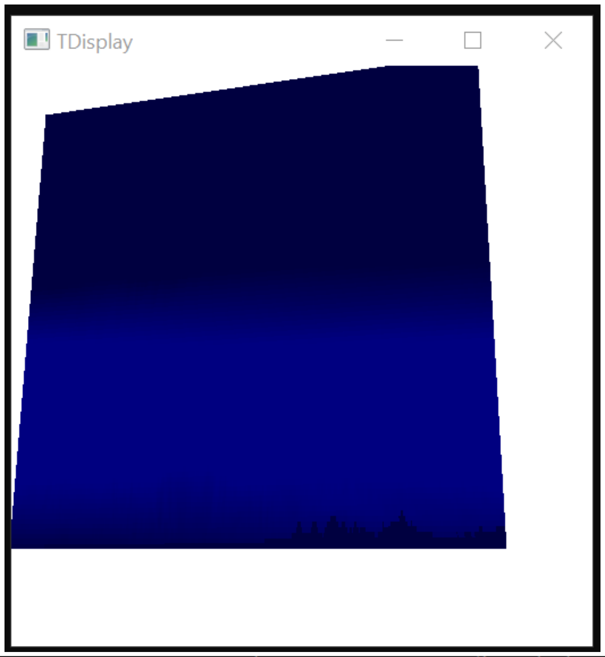
```
        printf("Couldn't read texture file: %s\n",filename);
    fread(texture, XDIM*YDIM*ZDIM*3, sizeof(unsigned char), infile);
    fclose(infile);

}

/*
 *  Initialize the vertex buffer object and
 *  the texture.
 */
void init(void) {
    GLuint vbuffer;
    GLuint ibuffer;
    GLuint vs;
    GLuint fs;
    GLuint vPosition;
    GLuint vTexture;
    char vertex[256];
    char fragment[256];

    glGenVertexArrays(1, &objVAO);
    glBindVertexArray(objVAO);

    /*
     *  Two squares, one straight on, the other
     *  slanted into the screen.
     */
    GLfloat vertices[] = {
         0.0, 0.5, 0.0,
         0.0, 0.5, 1.0,
         1.0, 0.5, 1.0,
         1.0, 0.5, 0.0,
    };

    GLuint indices[] = {0, 1, 2, 2, 3, 0 };

    glGenBuffers(1, &vbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), NULL, GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vertices), vertices);

    glGenBuffers(1, &ibuffer);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibuffer);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
      GL_STATIC_DRAW);

    sprintf(vertex,"volume%s.vs",Shader);
    sprintf(fragment,"volume%s.fs",Shader);
    vs = buildShader(GL_VERTEX_SHADER, vertex);
    fs = buildShader(GL_FRAGMENT_SHADER, fragment);
    program = buildProgram(vs,fs,0);

    glUseProgram(program);
    vPosition = glGetAttribLocation(program,"vPosition");
    glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(vPosition);
    triangles = 2;

    glClearColor (0.3, 0.3, 0.3, 1.0);
    glEnable(GL_DEPTH_TEST);

    /*
     *  Create the texture.
     */

    glGenTextures(1, &texName);
    glBindTexture(GL_TEXTURE_3D, texName);

    glTexImage3D(GL_TEXTURE_3D, 0, GL_RGB, XDIM,
        YDIM, ZDIM, 0, GL_RGB, GL_UNSIGNED_BYTE,
        &texture[0][0][0][0]);

    glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

}

void framebufferSizeCallback(GLFWwindow *window, int w, int h) {

    // Prevent a divide by zero, when window is too short
    // (you cant make a window of zero width).

    if (h == 0)
        h = 1;

    float ratio = 1.0f * w / h;

    glfwMakeContextCurrent(window);

    glViewport(0, 0, w, h);

    projection = glm::perspective(0.7f, ratio, 1.0f, 100.0f);

}

```

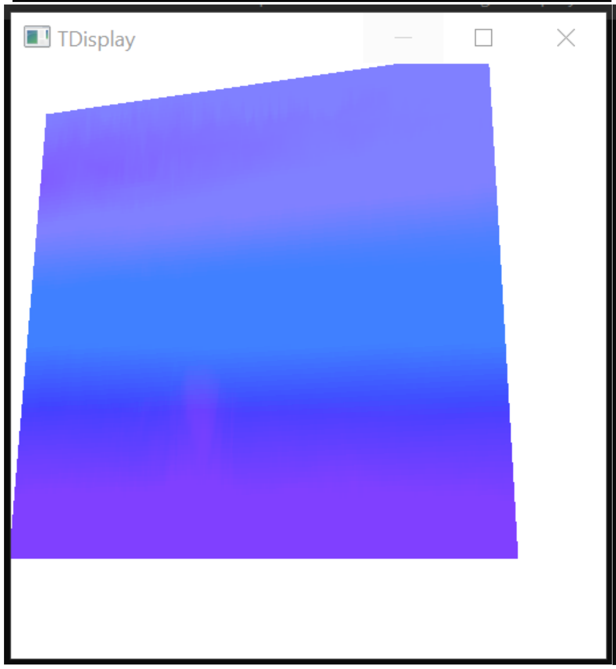
```

void display(void) {
    glm::mat4 view;
    glm::mat4 model;
    int modelLoc;
    int viewLoc;
    int projectionLoc;

    model = glm::translate(glm::mat4(1.0), glm::vec3(0.0, -0.5, 0.0));
    model = glm::rotate(model, angle, glm::vec3(0.0, 0.0, 1.0));
    model = glm::translate(model, glm::vec3(0.0, 0.5,0.0));
    model = glm::translate(model, glm::vec3(0.0, dy, 0.0));

    view = glm::lookAt(glm::vec3(eyex, eyey, eyez),
                    glm::vec3(0.5, 0.0, 0.5),
                    glm::vec3(0.0f, 0.0f, 1.0f));

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glUseProgram(program);
    modelLoc = glGetUniformLocation(program,"model");
    glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
    viewLoc = glGetUniformLocation(program,"view");
    glUniformMatrix4fv(viewLoc, 1, 0, glm::value_ptr(view));
    projectionLoc = glGetUniformLocation(program,"projection");
    glUniformMatrix4fv(projectionLoc, 1, 0, glm::value_ptr(projection));

    glBindTexture(GL_TEXTURE_3D, texName);
    glBindVertexArray(objVAO);
    glDrawElements(GL_TRIANGLES, 3*triangles, GL_UNSIGNED_INT, NULL);

}

/*
 *  Called each time a key is pressed on
 *  the keyboard.
 */

static void key_callback(GLFWwindow* window, int key, int scancode, int
  action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, GLFW_TRUE);

    if (key == GLFW_KEY_A && action == GLFW_PRESS)
        phi -= 0.1;
    if (key == GLFW_KEY_D && action == GLFW_PRESS)
        phi += 0.1;
    if (key == GLFW_KEY_W && action == GLFW_PRESS)
        theta += 0.1;
    if (key == GLFW_KEY_S && action == GLFW_PRESS)
        theta -= 0.1;
    if (key == GLFW_KEY_T && action == GLFW_PRESS) {
        dy -= 0.05;
        if (dy < -0.5)
            dy = -0.5;
    }
    if (key == GLFW_KEY_Y && action == GLFW_PRESS) {
        dy += 0.05;
        if (dy > 0.5)
            dy = 0.5;
    }
    if (key == GLFW_KEY_R && action == GLFW_PRESS)
        angle += 0.02;
    if (key == GLFW_KEY_F && action == GLFW_PRESS)
        angle -= 0.02;

    eyex = (float)(r*sin(theta)*cos(phi));
    eyey = (float)(r*sin(theta)*sin(phi));
    eyez = (float)(r*cos(theta));

}

void error_callback(int error, const char* description)
{
    fprintf(stderr, "Error: %s\n", description);
}


int main(int argc, char** argv)
{
    int type = 1;
    GLFWwindow *window;

    glfwSetErrorCallback(error_callback);

    if (!glfwInit()) {
        fprintf(stderr, "can't initialize GLFW\n");
    }

    window = glfwCreateWindow(512, 512, "TDisplay", NULL, NULL);

    if (!window)
    {
        fprintf(stderr, "can't create window\n");
        glfwTerminate();
        exit(EXIT_FAILURE);
    }

    glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);

    glfwSetKeyCallback(window, key_callback);

    if(argc > 1)
        type = atoi(argv[1]);

    /*
     *  initialize glew
     */
    glfwMakeContextCurrent(window);
    GLenum error = glewInit();
    if(error != GLEW_OK) {
        printf("Error starting GLEW: %s\n",glewGetErrorString(error));
        exit(0);
    }


    eyex = 0.0;
    eyey = 2.0;
    eyez = 0.0;

    theta = 1.6;
    phi = 1.5;
    r = 2.0;

    switch(type) {
    case 1:
        read_texture("TCf20");
        break;
    case 2:
        read_texture("Pf20");
        break;
    case 3:
        read_texture("PRECIPf20");
        break;
    case 4:
        read_texture("QCLOUD20");
        break;
    default:
        printf("Unrecognized type: %d\n", type);
        exit(0);
    }
    init();

    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glViewport(0, 0, 512, 512);

    projection = glm::perspective(0.7f, 1.0f, 1.0f, 200.0f);

    glfwSwapInterval(1);

    while (!glfwWindowShouldClose(window)) {
        display();
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    glfwTerminate();


}


```

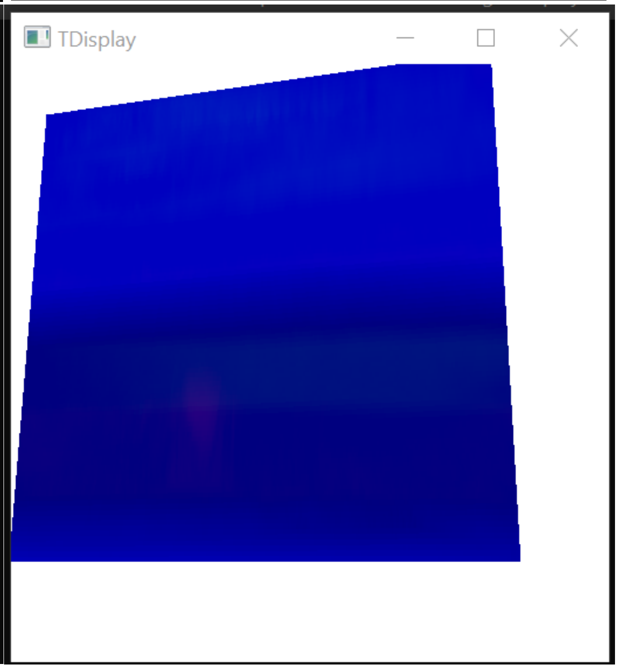
```glsl
#version 330 core

in vec3 texCoord;
in vec3 dir;
uniform sampler3D tex;

void main(void) {
    int slices = 10;
    int i;

    float step = 0.1;
    vec3 coord = texCoord;

    vec4 average = vec4(0.0, 0.0, 0.0, 1.0);
    vec4 minVec = vec4(0.0, 0.0, 1.0, 1.0);
    vec4 maxVec = vec4(0.0, 0.0, 1.0, 1.0);
    vec4 diffVec;

    for(i = 0; i < slices; i++) {
        average += texture(tex, coord);
        minVec = min(minVec, texture(tex, coord));
        maxVec = max(maxVec, texture(tex, coord));
        diffVec = maxVec - texture(tex, coord);

        coord += step*dir;
    }

    average /= slices;
    gl_FragColor = texture(tex, texCoord);

    //gl_FragColor = average;
    //gl_FragColor = minVec;
    //gl_FragColor = maxVec;
    //gl_FragColor = diffVec;
}

```

```
#version 330 core

in vec4 vPosition;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

out vec3 texCoord;
out vec3 dir;

void main(void) {
    vec4 position = vPosition;
    gl_Position = projection * view * model * vPosition;
    position.y -= 0.5;

    texCoord = (model * position).xyz;
    dir = (model * vec4(0.0, 1.0, 0.0, 0.0)).xyz;
}

```


TDisplay


TDisplay


TDisplay


TDisplay