3D Visualization of petroleum data

Exercise 3: Color mapping

Einar Baumann

October 2, 2013

1 Setup

As I am using Linux and not Windows, the recommended tutorial was not of much help for getting started. Instead I used a tutorial I found on the OpenGL Wiki [**?**]. This tutorial recommend that I used *GLX* and *Xlib* to enable displaying OpenGL using the X Window system, which is the default window system for most Linux distributions.

I used the framework laid out in the tutorial (with some tweaking) to build my application, substituting my own OpenGL commands.

I used plain C for the implementation, as it's what I'll have to do for a project in a very similar course I'm taking parallel to this course.

2 Drawing the squares

For drawing the squares, I used a *Vertex Array Object*, which is now the standard way of doing such things, after the `glBegin` / `glEnd` structure was deprecated. A vertex array object encapsulates all of the state needed to specify vertex data, including the format of the vertex data (e.g. quads, triangles, polygons) and the sources for the vertex arrays.

Vertex Buffer Objects were not used, as the program only contains one instance of one object.

3 Color mapping

I used a *Color Array* to map colors onto the squares. The array defining the colors were generated using the scalar algorithm outlined in the course to map the temperatures given for the vertices onto the given colors.

The algorithm relies on a specified set of *n* colors in a zero-indexed list, as well as knowing the *min* and *max* values in our range. We input this, along with the value s_i that we wish to map, into equation (**??**) to calculate an index *i*. The index points us to one of the *n* defined colors.

$$i = n \left(\frac{s_i - min}{max - min} \right) \tag{1}$$

The result of the color mapping is shown in Figures **??** and **??**.

4 Symmetry

First off, GL_QUADS has been deprecated since OpenGL v3.1 -- this is due to the fact that GPU's are optimized for rendering triangles. Even when using GL_QUADS, the squares are decomposed into two triangles. As far as I can see, what is done is roughly equivalent to placing an extra edge between the first and the third vertex that is drawn according to the index list.

If two vertices have the same color and an edge connecting them, the entire edge will have the same constant color as the vertices; i.e. there is no real interpolation along the center of it. This is seen in Figure **??**. This results in an unnatural-looking temperature distribution on our case.

In the symmetry seen in Figure **??** we can still see the edges created when the two squares are decomposed (and of course the center edge), but the resulting interpolation looks -- to me at least -- more natural.

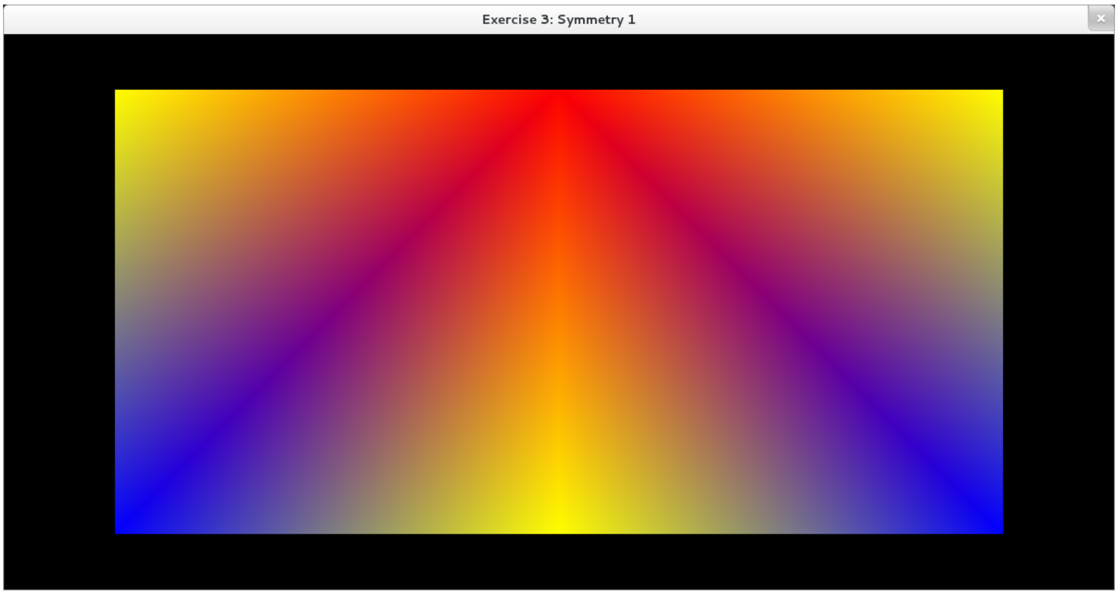

Figure 1: A screen-cap of the first of two possible configurations with symmetry along the vertical center line.

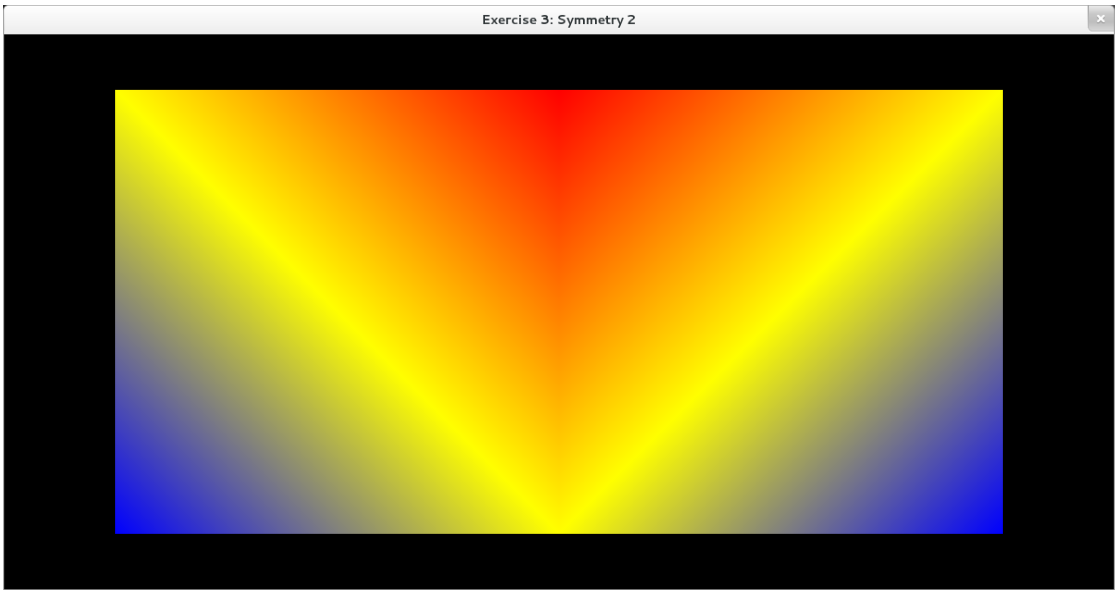

Figure 2: A screen-cap of the second of two possible configurations with symmetry along the vertical center line.

A Source code

A.1 Main program

ex3-colorquads.c

```c
/*
* 3D Visualization of petroleum data
*   Exercise 3
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <GL/gl.h>      // Header file for OpenGL32 Library
#include <GL/glu.h>     // Header File For The GLu32 Library



float * getRgbColorArray(int temps[], int length) {
    /*
    * 1       0-25  Blue    rgb(0.0f, 0.0f, 1.0f)
    * 2      26-50  Green   rgb(0.0f, 1.0f, 0.0f)
    * 3      51-75  Yellow  rgb(1.0f, 1.0f, 0.0f)
    * 4     76-100  Red     rgb(1.0f, 0.0f, 0.0f)
    */
    float red[3]    = {1.0f, 0.0f, 0.0f};
    float yellow[3] = {1.0f, 1.0f, 0.0f};
    float green[3]  = {0.0f, 1.0f, 0.0f};
    float blue[3]   = {0.0f, 0.0f, 1.0f};

    // Setting values to be used for this case of the color mapping algorithm
    int min = 0;
    int max = 100;
    int n = 4;

    // Declaring and allocating the array to be returned
    float *rgbArray = malloc(3*length*sizeof(float));

    // Appending appropriate rgb values to the rgbArray using the color mapping
    algorithm
    for (int j = 0; j < length; j++)
    {
        int i = n * (temps[j]-min)/(max-min);
        if (i == 0) {
            memcpy(rgbArray + j*3, blue, 3*sizeof(float));
        }
        else if (i == 1){
            memcpy(rgbArray + j*3, green, 3*sizeof(float));
        }
        else if (i == 2){
            memcpy(rgbArray + j*3, yellow, 3*sizeof(float));
        }
        else if (i >= 3){
            memcpy(rgbArray + j*3, red, 3*sizeof(float));
        }
    }

    return rgbArray;
}



void DrawTwoQuadsUsingVertexArrays() {
    // Defining coordinate data
    const float coords[] = {
        -0.8f, -0.8f, 0.0f,     // Lower left   [#1]
         0.0f, -0.8f, 0.0f,     // Lower middle [#2]
         0.0f,  0.8f, 0.0f,     // Upper middle [#3]
        -0.8f,  0.8f, 0.0f,     // Upper left   [#4]
         0.8f, -0.8f, 0.0f,     // Lower right  [#5]
         0.8f,  0.8f, 0.0f,     // Upper right  [#6]
    };
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, coords);

    // Defining temperature data
    int temperatures[] = {10, 50, 100, 50, 10, 50};
    float *rgbArray;
    rgbArray = getRgbColorArray(temperatures, 6);
    glEnableClientState(GL_COLOR_ARRAY);
    glColorPointer(3, GL_FLOAT, 0, rgbArray);

    // Clearing color buffer
    glClear(GL_COLOR_BUFFER_BIT);

    // Defining indeces
    // Note: Triangles are generated for each of the two quads, with
    //  the hypothenuse connecting the first and the third vertices
    const unsigned char indeces[] = { 0, 1, 2, 3, 4, 1, 2, 5 }; // Symmetry 1
    //const unsigned char indeces[] = { 1, 0, 3, 2, 1, 4, 5, 2 }; // Symmetry 2

    glColor3f(1.0f, 1.0f, 1.0f);    // set red, green, blue
    glDrawElements(GL_QUADS, 8, GL_UNSIGNED_BYTE, indeces);
}
```

```
void DrawGLfigs() {
    DrawTwoQuadsUsingVertexArrays();
}


// Including OpenGL methods
#include"ex3-glfuns.c"

int main(int argc, char *argv[]) {
    initializeWindow();
    displayWindow();
    createContext_enableDepthTest();
    startProgram();
}
```

A.2 OpenGL and X commands

ex3-glfuns.c

```
#include <X11/X.h>      // X window system
#include <X11/Xlib.h>   // Libraries for X window system
#include <GL/glx.h>     // OpenGL Extension for X window system

Display                 *dpy;
Window                  root;
GLint                   att[] = { GLX_RGBA, GLX_DEPTH_SIZE, 24, GLX_DOUBLEBUFFER, None
    }; // Defining the needed OpenGL capabilities for our program. More possible
    options can be found in GL/glx.h
XVisualInfo             *vi;
Colormap                cmap;
XSetWindowAttributes    swa;
Window                  win;
GLXContext              glc;
XWindowAttributes       gwa;
XEvent                  xev;




void initializeWindow() {
    dpy = XOpenDisplay(NULL);           // Opening graphical output on local computer
    root = DefaultRootWindow(dpy);      // Creating handle to root window (the desktop
    background window)
    vi = glXChooseVisual(dpy, 0, att);  // Assigning an available visual
    cmap = XCreateColormap(dpy, root, vi->visual, AllocNone);   // Creating colormap
    for the window

    /* Initializing XSetWindowAttributes structure
    *   The The colormap we created is used
    *   The window shall respond to Exposure and KeyPress events
    */
    swa.colormap = cmap;
    swa.event_mask = ExposureMask | KeyPressMask;
}




void displayWindow() {
    /* Creating window
    *  The arguments are:
    *   Displaypointer, determining which display to create window on: dpy
    *   The handle of the root window passed as the parent window: root
    *   Initial x,y positions of the window; usually ignored: 0, 0
    *   Window width and height in pixels: 600, 600
    *   Border width: 0
    *   Depth, defined in XVisualInfo structure *vi: vi->depth
    *   Window type: InputOutput
    *   bitwise-OR of the values CWColormap and CWEventMask. This tells the X server which
    *       fields of the XSetWindowAttributes structure swa were filled by the program and
    *       should be taken into account when creating the window: CWColormap | CWEventMask
    *   Pointer to the structure itself: &swa
    *  Returns a window id (int)
    */
    win = XCreateWindow(dpy, root, 0, 0, 1200, 600, 0, vi->depth, InputOutput, vi->visual, CWColormap | CWEventMask, &swa);


    XMapWindow(dpy, win);       //Makes the window appear
    XStoreName(dpy, win, "Exercise 3: Symmetry 2"); // Change the string in the title bar
}




void createContext_enableDepthTest() {
    // Create GL context to enable displaying 3D things and bind it to the window
    glc = glXCreateContext(dpy, vi, NULL, GL_TRUE);
    glXMakeCurrent(dpy, win, glc);

    // Enable depth test bebause we want to use depth buffering (GLX_DEPTH_SIZE)
    glEnable(GL_DEPTH_TEST);
}
```

```
void startProgram() {
    while(1) { // Start infinite loop
        XNextEvent(dpy, &xev);      // Blocks program execution untill an ExposureMask or KeyPressMask event occurs

        /* If event is an Exposure Event (an event generated when the system thinks a window should be updated):
        *   Get information about current window size
        *   Resize viewport
        *   Draw the thing
        *   Swap the buffer (we're drawing a double buffered visual)
        */
        if(xev.type == Expose) {
                XGetWindowAttributes(dpy, win, &gwa);
                glViewport(0, 0, gwa.width, gwa.height);
                DrawGLfigs();
                glXSwapBuffers(dpy, win);
        }

        /* If the event is a keypress, the program is terminated:
        *   GL context binding to the window is released
        *   GL context is destroyed
        *   Kill the window
        *   Close the display
        *   Exit the program
        */
        else if(xev.type == KeyPress) {
                glXMakeCurrent(dpy, None, NULL);
                glXDestroyContext(dpy, glc);
                XDestroyWindow(dpy, win);
                XCloseDisplay(dpy);
                exit(0);
        }
    }
}
```