

CS 365 Lab 1 Drawing Triangles

You will write a program that allows you to draw 2D colored triangles. This program is written in OpenGL/GLUT, but you will rasterize the lines and color the triangles yourself instead of using OpenGL's facilities for that purpose. You will be using GLUT mouse and keyboard functions for triangle drawing, erasing, and moving, however.

Attached to this writeup is the skeleton of an OpenGL program that you are to use for your lab. There are sections marked ??? where most likely your code will be inserted.

What it does

When the program starts, it displays a light blue window for drawing.

1. You draw a triangle by clicking the mouse (use the major, or "left" button) at three different spots in the window. After the first two clicks, a line appears in black. After the third click, the remaining two sides of the triangle appear, and the triangle is filled in with a gradient of color.
2. Repeatedly clicking draws more triangles. The program keeps a stack of triangles you have drawn, and places them on the screen with the most recent triangles on top.
3. Pressing the E or e key erases the most-recently-drawn triangle. Since this triangle may lie on top of others, you will need to redraw the whole stack of triangles again.
4. Pressing the minor (usually called "right") mouse button, with the cursor over a triangle, to drag it around. If you are clicking on a point where several triangles cross, you move the topmost one. Again, you will redraw the whole stack after the mouse moves.
5. Pressing Q or q quits.

How it does it

The attached OpenGL/GLUT skeleton program (also available from the class web page) contains an array of pixels called `canvas`. Each pixel in the array contains three bytes of type `rgb_pixel` for the three colors. The program sets up OpenGL so that it uses OpenGL pixel mode to draw the entire canvas from this pixel array.

The parts marked with question marks allow you to insert your own code. The screen is initialized, the viewing camera is set for 2D orthogonal, and callbacks are installed for you. If you update the pixel array, it will automatically appear on the screen.

The array is one dimensional, but you can index the pixel array using the `LOC` macro. The x and y coordinates range from 0 to `screenWidth` and `screenHeight` respectively. The following illustrates setting a single pixel element at x_0 and y_0 to pure white:

```
canvas[LOC(x0,y0)].r = 255;
canvas[LOC(x0,y0)].g = 255;
canvas[LOC(x0,y0)].b = 255;
```

Alternatively you could have a single variable that contains pure white:

```
rgb_pixel white = {255, 255, 255};
...
canvas[LOC(x0,y0)] = white;
```

At the beginning of the program set the entire canvas to light blue.

In the mouse-click left-button handler, you keep track of triangles as they are being drawn. First click sets the first point for the first triangle. Next sets the second point and draws a line. Third click set the third point (finishing the triangle), and draws two more lines, then colors the triangle. The triangle is saved away in a data structure, and subsequent clicks start the next triangle.

In keyboard `E` key handler, you remove the last-added triangle and call a routine to re-draw the entire stack of triangles. The triangles are re-drawn in the order they were entered.

In the mouse-click right-button-down handler, you set a flag saying that a triangle is being dragged (assuming the coordinates of the click are within a triangle—you need to scan the stack from most recent to oldest triangle to find out). You also

save these coordinates. In the button-up handler you clear the flag, so the triangle will not be dragged any more.

In the mouse-move handler, you see whether a triangle is being dragged. If so, you compute the difference between the current mouse coordinates and the previous coordinates (originally saved saved on mouse-down, updated as you move) and use the delta-x and delta-y differences to move the whole triangle. Since triangles can be on top of each other, this means redrawing the whole shebang.

A warning about y coordinates: it is usual for drawing programs to have y increase from 0 at the bottom of the screen to its maximum value at the top. It is also usual for mouse locations to be counted from 0 at the *top* of the screen, and increasing going down. This means that the mouse y_m coordinate and the screen-drawing y coordinates are related by $y = \text{screenHeight} - y_m$.

Colors

At each triangle vertex, you assign a pseudo-random color based on its location. The color is then smoothly interpolated throughout the area of the triangle. Do not overwrite the black-line triangle edges.

The formula for assigning a color at vertex x_i, y_i is

$$\begin{aligned}r_i &= (3 \times x) \bmod 255 \\g_i &= (3 \times y) \bmod 255 \\b_i &= (3 \times (x + y)) \bmod 255\end{aligned}$$

When the triangle moves, it *retains its original colors*. That means you need to save these colors in the triangle data structure.