Lab 1 - Line Drawing

Last updated 2017/02/11 21:39:29

Update history:

```
2017/02/11: initial version
```

---

1. Introduction

In this course, you will implement some of the 2D drawing routines we have been discussing in class. This will help improve your understanding of these algorithms.

This assignment involves line drawing. You will implement the Midpoint Line Drawing algorithm discussed in class. You have the option of doing this assignment in C or C++.

---

2. Programming Environment

The programming environment that you will use for this assignment (along with the the remainder of the 2D assignments) is a set of simple modules with implementations in C and C++ These modules are designed as object-oriented classes in the C++ version, and as equivalent non-OO versions in C. You are free to use any of the implementations. The classes include:

- `Buffers` - a support module providing OpenGL vertex and element buffer support.
- `Canvas` - a simple 2D image module that allows the ability to set a pixel.
- `Rasterizer` - a rasterization module which includes a Canvas for drawing.
- `ShaderSetup` - a support module that handles shader program compilation and linking.
- `lineMain` - the main program for the application.
- `shader.vert`, `shader.frag` - simple GLSL 1.50 shaders.
- `alt.vert`, `alt.frag` - simple GLSL 1.20 shaders.

The C version includes a module named `FloatVector` which provides an extensible vector holding floating-point values. Both the C and C++ versions include a file named `header.mak` for use with the `gmakemake` program on our systems to create a `Makefile` to simplify compilation and linking of the program.

See the *Supplied Files* section (below) for details on how to download the framework.

---

3. Routines to Implement

You will need to modify the `Rasterizer` module. For this assignment, you will need to complete the method `drawLine()` using the Midpoint Line Drawing algorithm, and the method `myInitials()` to draw your initials (see below).

In your `drawLine()` implementation, you need only make use of the method `setPixel()` on the `Canvas` object (which is part of the `Rasterizer`) holding the pixels being drawn.

The prototypes for the `drawLine()` and `myInitials` methods vary a bit between languages:

C:
```
void drawLine( int x0, int y0, int x1, int y1, Rasterizer *R );
void myInitials( Rasterizer *R );
```

C++:
```
void drawLine( int x0, int y0, int x1, int y1 );
```

```
void myInitials( void );
```

In all cases, the line is drawn from vertex `(x0,y0)` to vertex `(x1,y1)`. The additional parameter in the C version is due to the fact that C is not an object-oriented language. In C++, the `drawLine()` method is part of the `Rasterizer` class, and is invoked in the standard way:

```
Rasterizer R;

R.drawLine( ... );
```

In C, because `drawLine()` is a standalone function, the `Rasterizer` it is being applied to must be supplied as a parameter:

```
Rasterizer *R;

drawLine( ..., R );
```

You are free to add additional members and methods to `Rasterizer` as you see fit; however, you cannot modify any other module.

In addition to implementing the `drawLine()` method, you must modify the `myInitials()` method to add your own first and last initials to the image, drawing these letters in blue. You are free to use any style you wish for these letters; the only constraint is that they should be drawn as a series of line segments (as the existing letters are drawn).

Our in-class discussion of the midpoint line algorithm covered the case where the line segment had a small positive slope ($0 < m < 1$); your implementation must also handle the other three cases (large positive slope, small negative slope, and large negative slope). Also, remember that our discussion assumed that the line was being drawn from left to right (i.e., that `x0` was less than `x1`); you are not guaranteed that this will be the case, so you'll need to remember to check the incoming coordinates and swap the vertices if necessary.

Figure 1 below shows the solution for the letters 'G' and 'O' (these coordinates are provided for you). Figure 2 shows the complete solution for a student with the initials 'RB', drawn using the same style as the original letters.

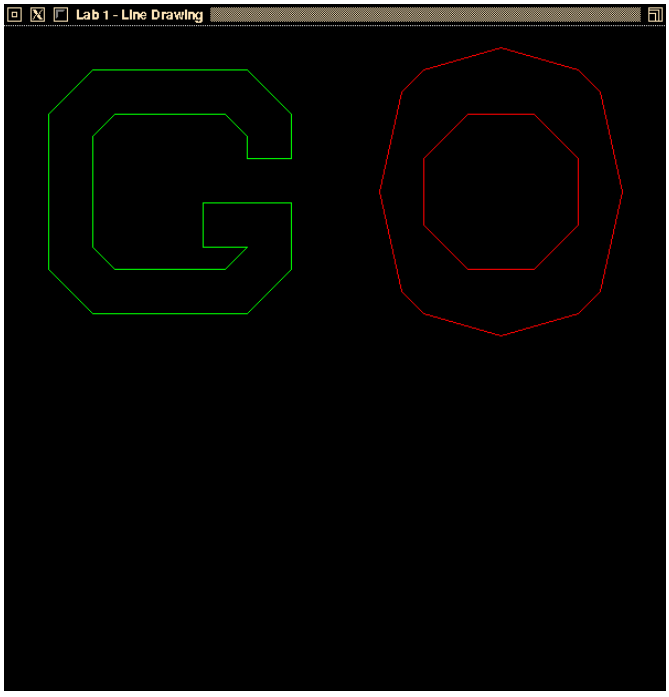

Figure 1

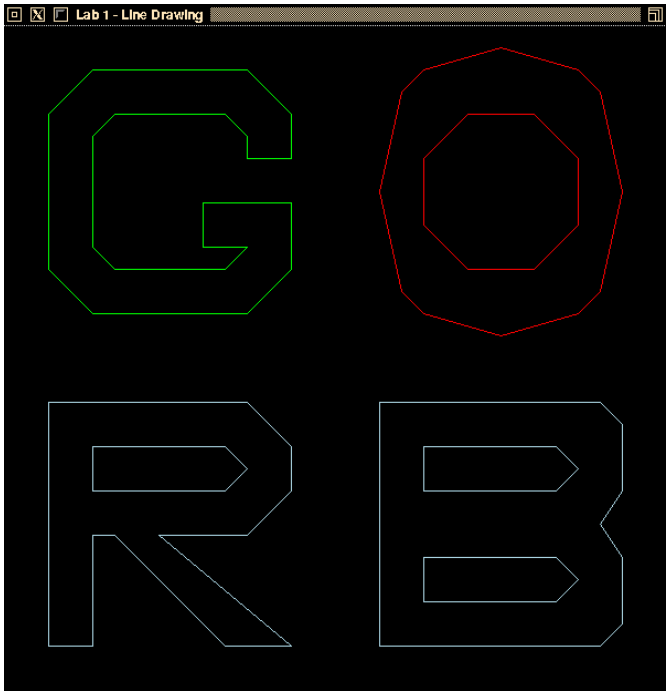

Figure 2

---

4. Supplied Files

The programming framework for this assignment is available as a ZIP archive. You can either download the `lab1.zip` archive directly, or retrieve it by executing the command

```
get  cscix10  lab1
```

on any CS Ubuntu® system.

The ZIP archive contains a folder named `lab1`; under that are subfolders `c` and `c++`, which contain the obvious things. In the C and C++ folders you'll find a file named `header.mak`, for use on the CS systems to help you generate a `Makefile` that will

compile and link your program with the libraries used by the framework. See the contents of `header.mak` for details on how to do this. There is also a subfolder named `misc` which contains a shellscript named `compmac` for use on Mac systems.

---

5. What to Submit

Your implementations will be tested using a set of driver programs; some of them may be different from the driver found in the framework archive. Submit ***only*** your modified `Rasterizer` module - do not submit any other source files from the framework.

If you are working in C++, your implementation must be in a file named `Rasterizer.cpp`. If your implementation requires making changes to the `Rasterizer.h` file (e.g., you have added data members or member functions to the class declaration), you should submit that file along with `Rasterizer.cpp`. If you make changes to the `header.mak` file, you may submit your modified file as well.

Similarly, if you are working in C, your implementation must be in a file named `Rasterizer.c`. Again, if your implementation requires making changes to the `Rasterizer.h` file, you should submit that file along with `Rasterizer.c`. If you make changes to the `header.mak` file, you may submit your modified file as well.

If you have not submitted anything to the grader account yet this semester, you will need to register your account. Do this with the command

```
try  grd-x10  register  /dev/null
```

and answer the question about your section number based on the schedule that will be displayed.

Turn in only your C or C++ implementation file(s) described above and an optional `README` file using this command:

```
try  grd-x10  lab1  Rasterizer.X  optional_files
```

where *X* is the correct suffix for your code.

The 'try' scripts will reject submissions that attempt to turn in more than the required and optional files listed above.

If you are working in C or C++, you may optionally submit a modified `header.mak` file and/or a modified `Rasterizer.h` file. The modified `header.mak` should be based on the version found in the ZIP archive. If you do not submit one or both of these files, the 'try' scripts will use the version provided with the framework.

The minimum acceptance test is that your code must be complete - that is, it must compile and link cleanly when submitted. Submissions *will not be accepted* if they fail to compile and link cleanly. (Warning messages from the compiler are acceptable, but not fatal errors.)

Finally, you can verify that your submission was archived with the command

```
try  -q  grd-x10  lab1
```

This command will tell you whether or not an archive exists, and if so, what files submitted by you are in it.

---

6. Grading

Your grade will be based on your implementation of the required routine and its usability with the supplied test programs.

The lists of situations to be checked in your submission (see below) is not exhaustive; the tests run during grading may include other combinations. You may want to modify the test program you are given to cover a wide range of input situations.

`drawLine` Implementation	35 points
gentle positive slope	
gentle negative slope	
steep positive slope	
steep negative slope	
horizontal	
vertical	
diagonal	
polygon drawn with multiple calls to `drawLine`	

`myInitials` Implementation	5 points
initials are clearly drawn	

Other Considerations documentation	10 points
Grade	_____ / 50

---

7. Notes

Java applets are available online to help you visualize line drawing.

You are guaranteed that the dimensions of the drawing window will be 600x600 pixels.

The vertices given to `drawLine()` may occur in any order - that is, there is no guarantee that `(x0,y0)` is the leftmost vertex of the line segment.

You are to implement the integer Midpoint algorithm discussed in class. Implementations of other approaches (Bresenham's, slope/intercept, floating point, etc.) will receive *no credit*.

It is common knowledge that the code for the Midpoint algorithm is freely available on the Internet and in textbooks. You are free to use these references as a guide, but please do not simply cut and paste code from any of these sources.

Refer back to the "Hello, OpenGL!" programming assignment for information about obtaining and installing the GLFW and/or GLEW libraries.

Don't wait until the last minute to submit things! You may, in fact, want to submit even a partially-working solution as you work on it - there is no penalty for making multiple submissions, and this will help ensure that you get *something* submitted for this assignment.

Do not make any changes to the function prototypes. This means that your implementations must match the prototypes exactly in terms of number, types, and order of parameters. The reason for this is that the test programs assume that your implementations match those prototypes; if you make changes, there will be compilation errors, and even if the test programs link, they almost certainly won't execute correctly (which means you'll lose substantial amounts of credit for incorrect program performance).
