ECE 1305, Fall 2015

Project 1

Summary of task: Write a program that can produce a contour map from a digital elevation map.

Background:

We have finally covered enough of C++ to be able to solve some complex problems, by breaking them down into subtasks, writing each subtask as a short function, and writing a main 'driver' routine to make use of the subtasks to get a larger task done. This is what we talked about early in the semester as "top down" program design, sometimes refered to as "functional decomposition."

We'll use this approach to create contour maps from digital elevation data. (For our final project, we'll take the same digital elevation data and "paint" it by adding lighting to create (somewhat) realistic images from the elevation data.)

The task of creating contour maps has been greatly simplified now that most of the world's surface has been mapped using laser altimiters from satellites and planes. (Most of Mars and the moon have also been mapped this way.) This data is available as DEMs – digital elevation maps.

By applying some standard image processing techniques to the DEMs, we can easily determine where the isolines (lines of identical elevation) occur on a DEM. Given a DEM where each pixel in the 2-D array gives you the elevation of the surface at that location, segment the image by assigning a single value to all pixels within each range (100 ft or 1000 feet or meters, the desired resolution of the contour map), then use an edge detector to identify the transition between each segment. The result will give you the isolines marking the boundaries between each elevation range. Details of the steps needed are given below.

There are links below to several different DEMs, including some simulated data, a small volcano in New Zealand, a couple of mountain peaks in the White Mountains of New Hampshire, and Milankovic crater, a large crater in the northern hemisphere of Mars.

Data Description

The DEM data is provided in a text-based format almost identical to the PGM file format; I've given these files the extension of ".egm" for "elevation graymaps". There is an image header giving the size of the data array, the range of the data (min & max), and an optional comment section where additional information such as latitude, longitude, data units (feet, meters, centimeters, etc.). This is followed by an array of values giving the elevation as seen from a satellite looking straight down on the site. We'll ignore the units, lat, long, etc., and just treat it as a rectangular area of elevation values.

Image header:
Line 1:
E2 ncols nrows min_elev max_elev
E2 the string "E2", identifying the file as an EGM file
ncols nrows # of columns & rows in the data array (integers)
min_elev min & max elevation in the file (floats)
Line 2:
ncmnts
# of comment lines following
Lines 3 to (2+nmnts)
comments describing the site, lat, lon, units, etc.
remainder of file
float values, one per pixel, giving elevation at that location

Example:

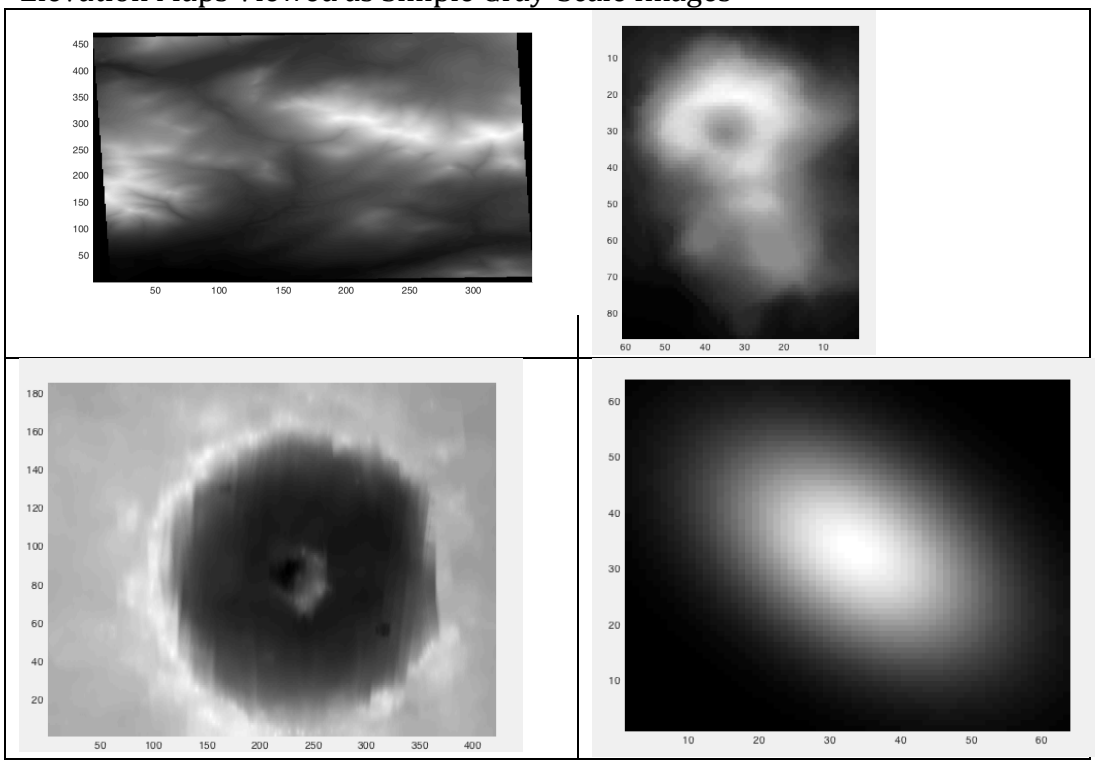
```
E2  5 10 0.1 14.9
2
# comment 1:  simulated data
# units:  arbitrary
   2.4   10.3    8.1   12.3    2.7
  11.9   11.2   14.9   13.0    4.0
   4.7    6.8    1.2    1.3    2.2
   7.9    1.3    6.6    6.0    2.0
   2.5    3.4    1.6    3.9   13.0
   9.0   13.7   14.4   12.0    8.7
   3.9    2.3    0.1    6.5    8.2
   9.8   12.4   11.6   13.7    2.2
```

Data files

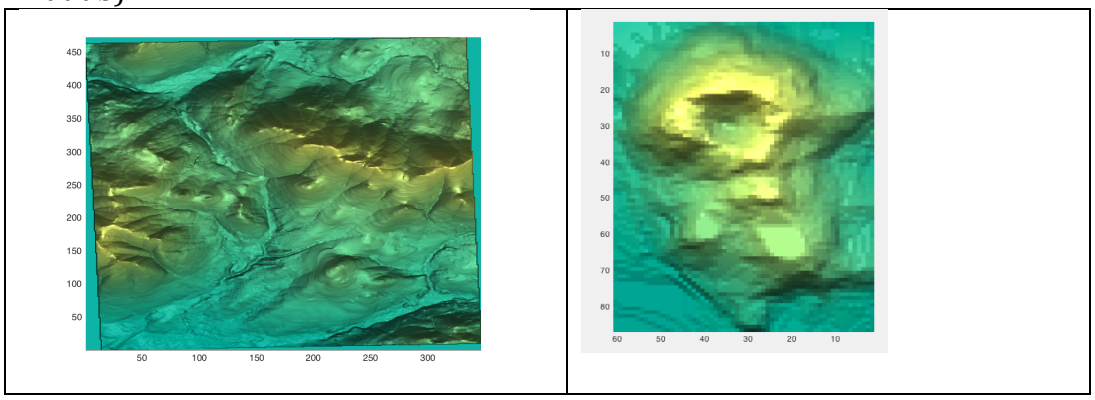
Data files are available here as a zip file. Included is a matlab program (elev2surf.m) which can be used to visualize the data. It draws the site as a surface, colors and shades it, and rotates it, spiralling up to a top view. Here

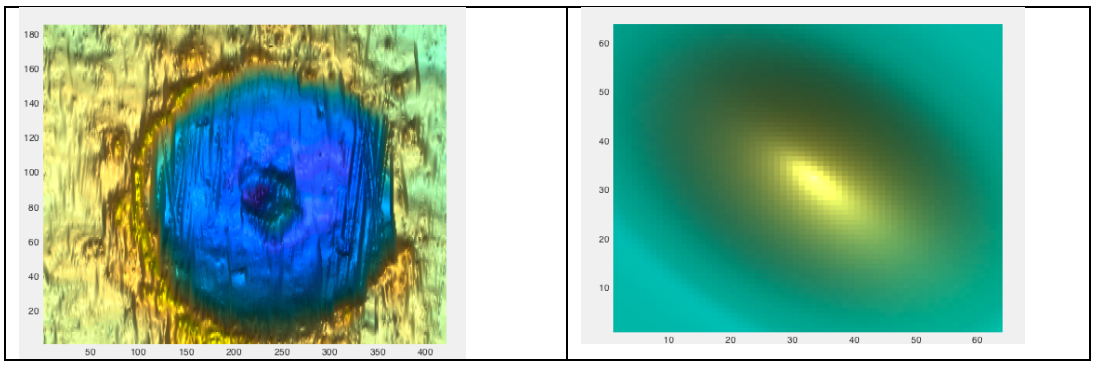
are several different views of several of the DEMs. The first set is the DEMs rendered as gray-scale – dark pixels are lower elevations, bright are higher higher. The second set were rendered as 3-D surfaces, shaded and colored, using matlab, and viewed from above (ending view of elev2surf.m). The third set is a different 3-D view.

Elevation Maps Viewed as Simple Gray-Scale Images

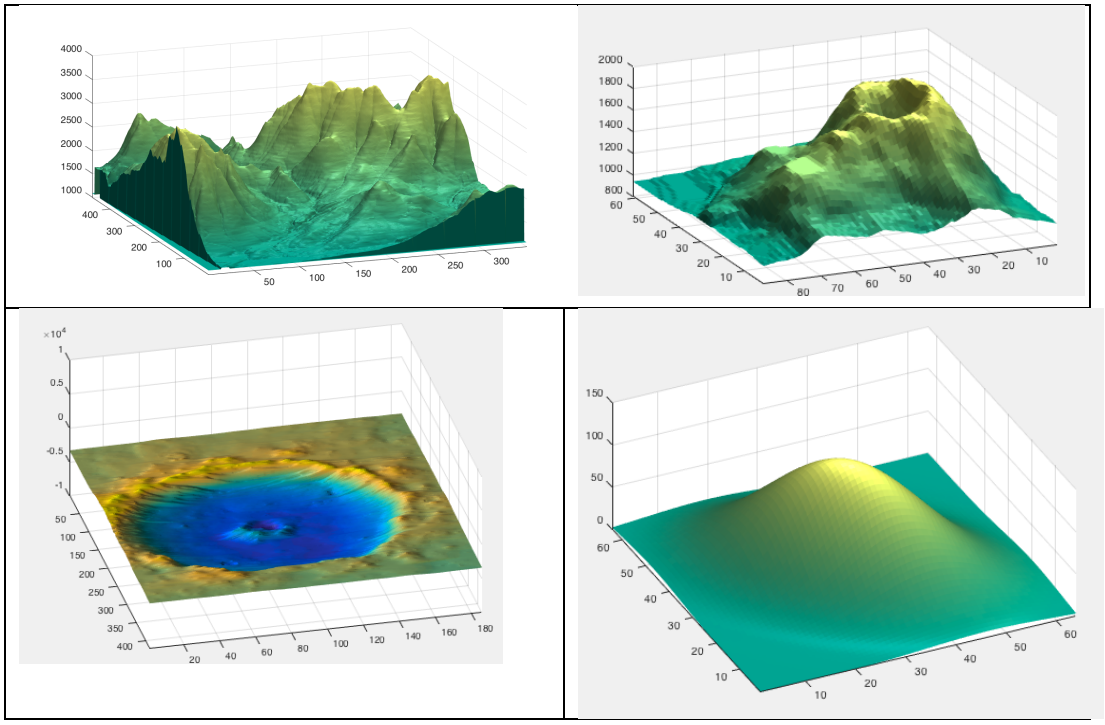


Elevation Map as 3D Surfaces, Colored & shaded, Top View (produced with Matlab)



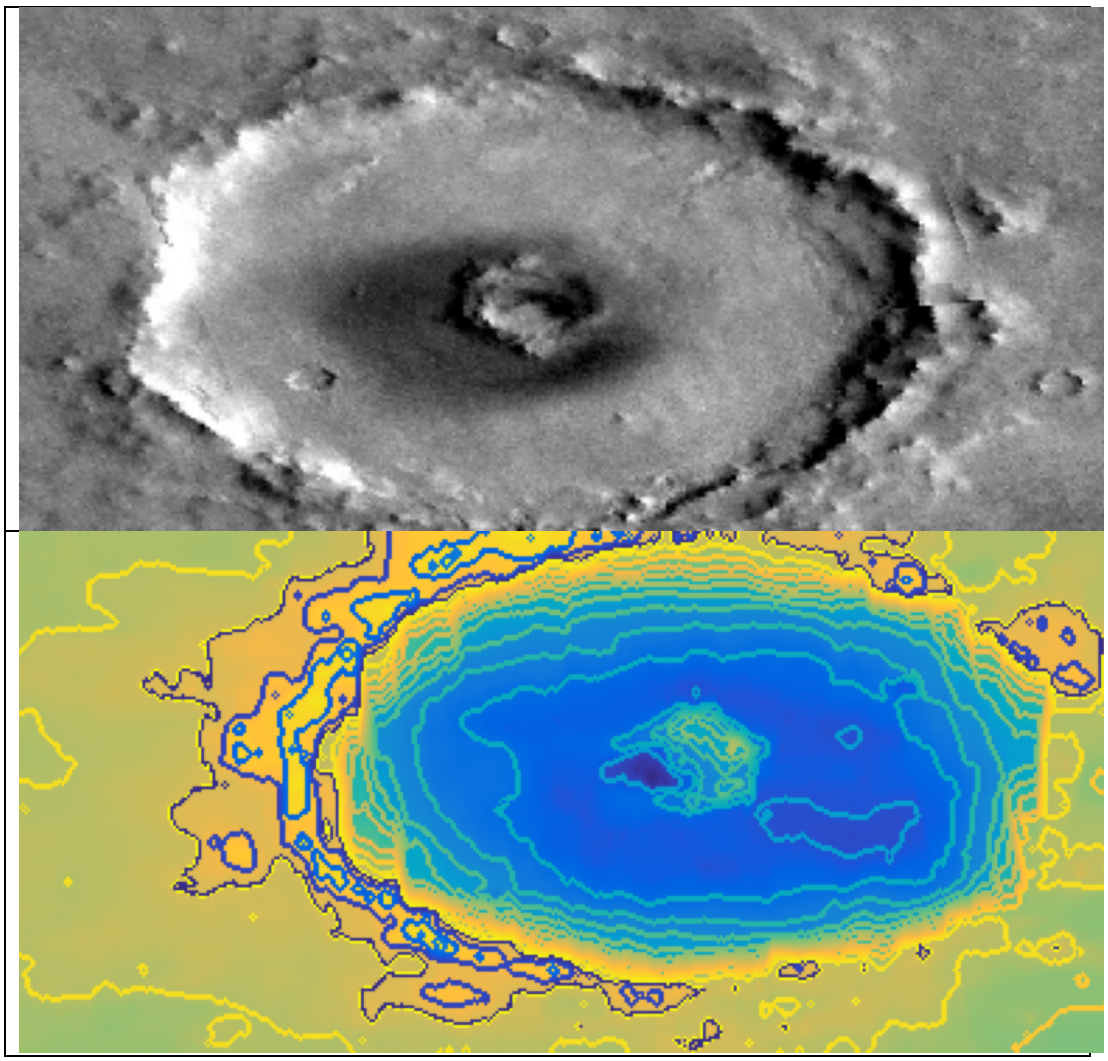


Elevation Maps as 3-D surfaces (produced with Matlab)



Here's what the Milankovic crater actually looks like...taken from a satellite orbiting Mars, and a contour map made from the DEM superimposed on the DEM (note: the satellite image is rotated 180 degrees from the DEM).

Milankovic Crater is a rather large crater in the northern hemisphere of Mars, approximately 75 miles in diameter – approximately 100 times the size of Meteor Crater in Arizona. (See https://en.wikipedia.org/wiki/Diacria_quadrangle for map showing it's approximate location).



Steps needed to create a contour map

There are various ways to create contour maps. One way is to segment the DEM by elevation range and use a simple edge detector to identify the transitions between elevation ranges. Here's a broad-brush-stroke breakdown of the process.

1. Read the DEM
2. threshhold the image into elevation ranges, assigning the same pixel value to all pixels in each range
3. Do a horizontal derivative
4. Do a vertical derivative
5. Combine the 2 derivatives to find the locations of the transitions between ranges

6. Save the resulting array as a gray-scale PGM

Image Derivatives

A horizontal derivative is done by simply subtracting every pixel from the one next to it. (Technically this introduces a ½ pixel phase shift, but we'll ignore that. Doing it this way works better on small images such as the 64x64 gaussian and the volcano.) A vertical derivative does the same thing, but in the vertical direction. These derivatives represent the horizontal and vertical edge-strength at each pixel in the image, and by taking the 2-D magnitude (square-root of the sum of the squares of the derivatives) you get a measure of the overall edge-strength. If you do this on a segmented image, you end up with identifying the boundaries between each segment. And if you segment the images into 20-ft or 100-ft or 250-ft elevation ranges, you end up with a contour map.

Segmenting an image

Segmenting the image is easy to do if you make use of integer arithmetic. Suppose you want your contours 100 ft apart (and the elevation map is in feet). Simply divide each pixel by 100 using integer division, and you'll have it. All pixels between 400 & 499 will have a value of 4, between 500 & 599 a value of 5, etc.. Then when you do your derivatives, the result will be zero except when you move from the 400's to the 500's (or vice versa).

Suggested Approach

By this time you should have all the tools you need to write each of the above steps as a function, and how to write a main program to put it all together.

To get this working in C++, I recommend you write functions to accomplish each step, and test each function individually, outputting the results of each step to a PGM and looking at the results both as a text file and viewed as an image. Once you have all the pieces working, you simply write a main routine to walk through all the steps sequentially. A useful approach to use while developing and testing the individual steps is to use a "command loop" where you prompt the user for the step they want to run, and call the appropriate function, then loop back for a new command. It might look something like this:

```
string cmd, fname;
float pixels[...];
...
```

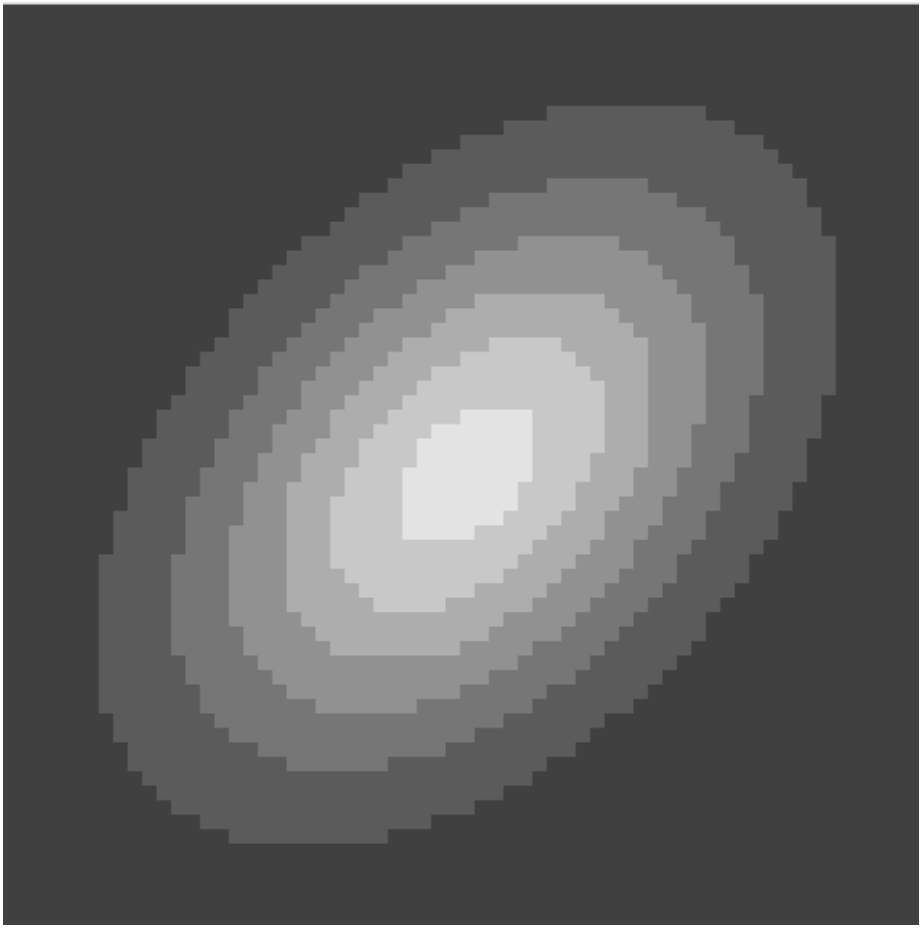
```
while (cmd != "quit")
{
     cout << "command? ";
     cin >> cmd;
     if (cmd == "readPGM")          // if your program
can read PGMs, you can save your intermediate output
          cout << "filename? ";     // and read it back
in later...
          cin >> fname;
          readPGM(fname,...);
     else if (cmd == "readEGM")     // read a DEM map
          ...
          readEGM(fname);
     else (if cmd == "save")        // save results to
a PGM file
          cout << "output filename?";
          cin >> outname;
          writePGM(outname,...);
     else (if cmd == "horiz")       // do a horizontal
derivative
          horiz_deriv(nr, nc, pixels,...)
     else...
}
```

Example

As an example, here are the results of each step for the gaussian, where I've segmented every 20 (range is 0 to 128, so I should have 6 contour lines):

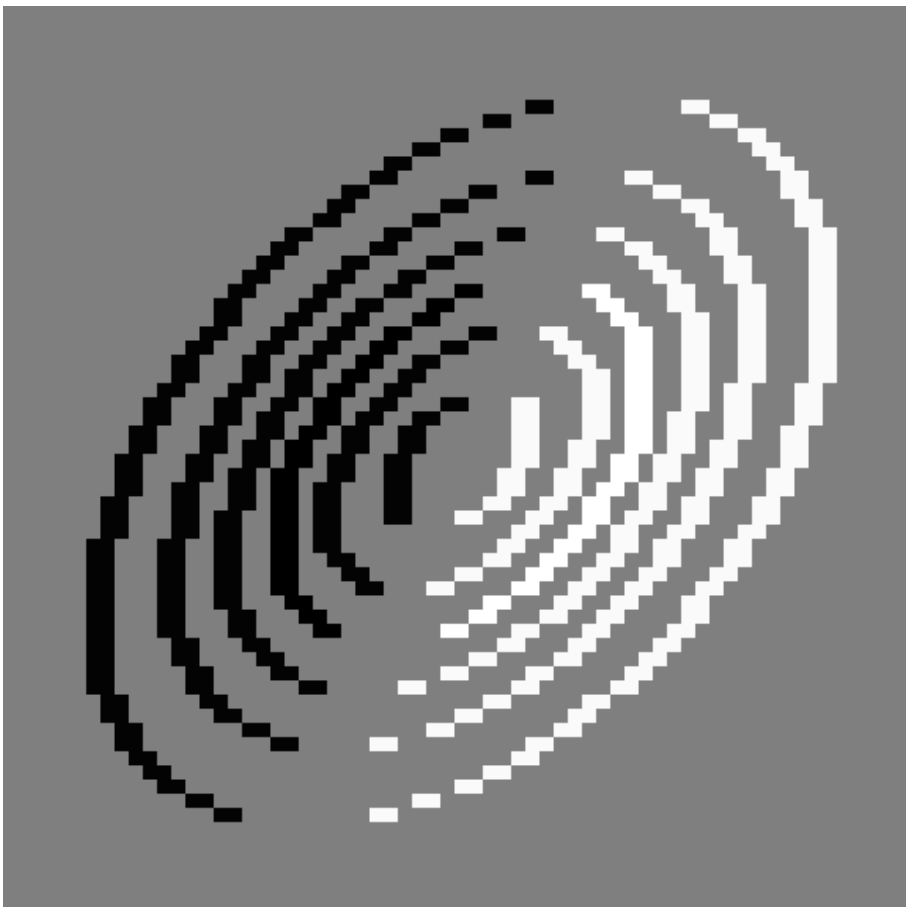
Step 1: read the EGM file into a pixel array
Step 2: segment the image at levels 0-19, 20-39, etc.
and do a contrast stretch so I can see them clearly
and save the result as a PGM so I can view it as an image and examine it as a text file

work1.pgm (outer level is too dark to see here...)



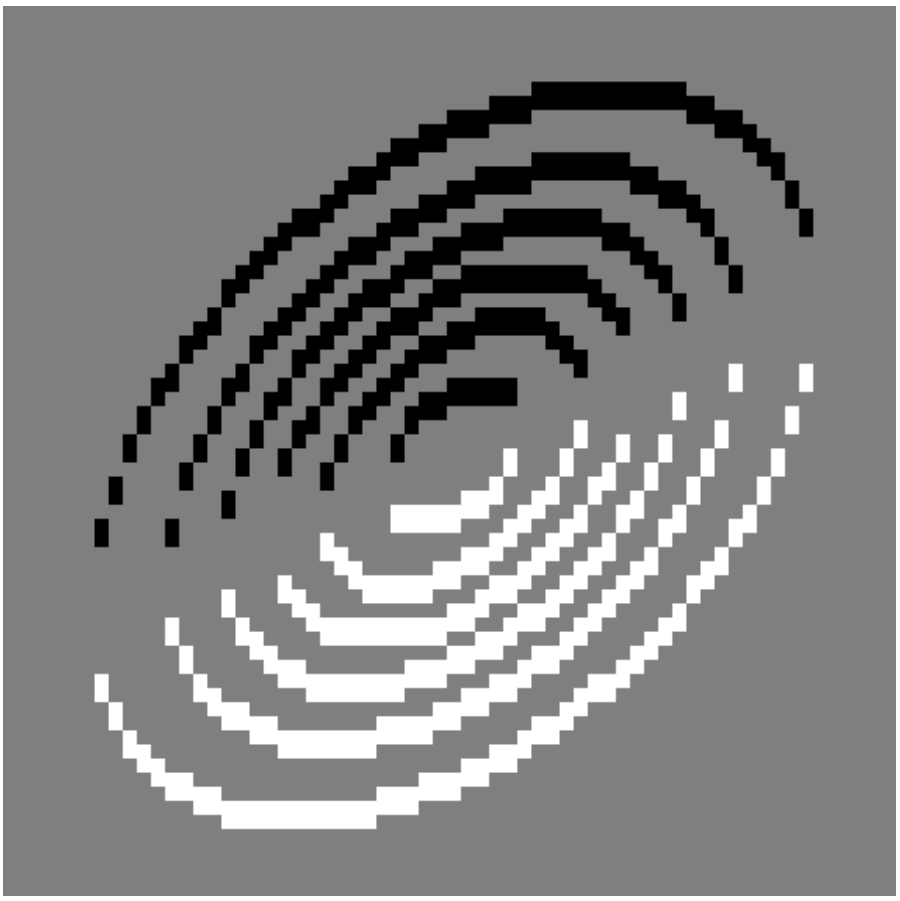
Step 3: do horizontal derivative on result of step 2
Note how the horizontal derivative doesn't see any edge on the horizontal lines.

work2.pgm



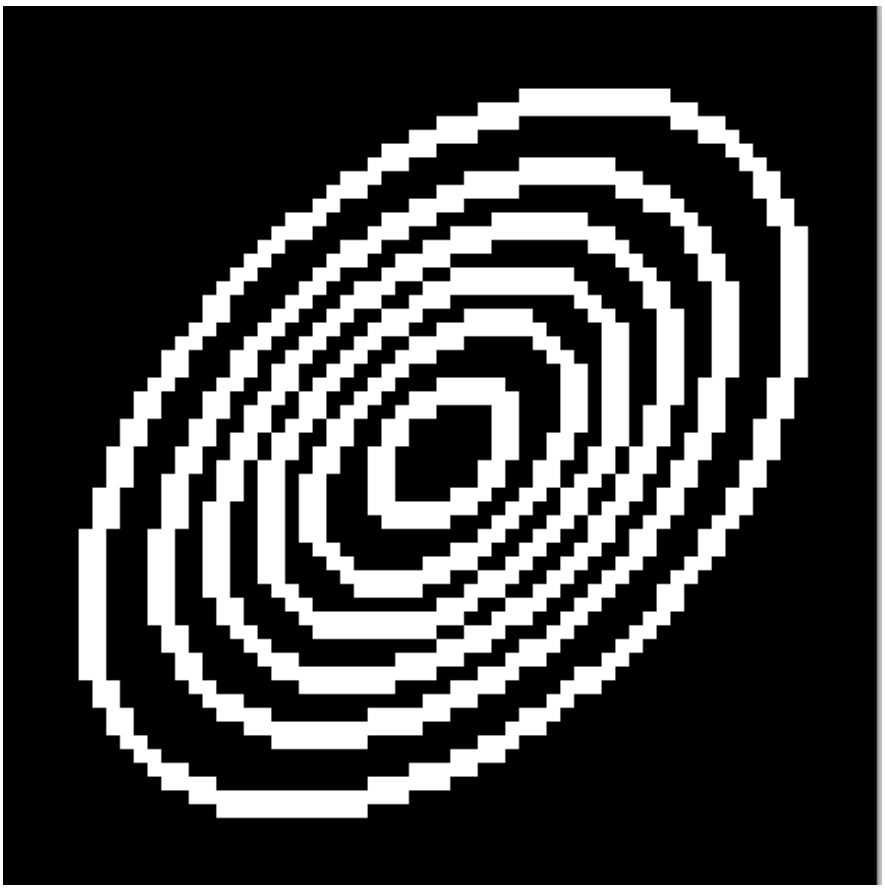
Step 4: do a vertical derivative (also on results of step 2 (segmented image)!, not of step 3)
Note how the vertical derivative doesn't see the vertical lines.

work3.pgm



Step 5: combine the horizontal and vertical derivatives and threshold the image:

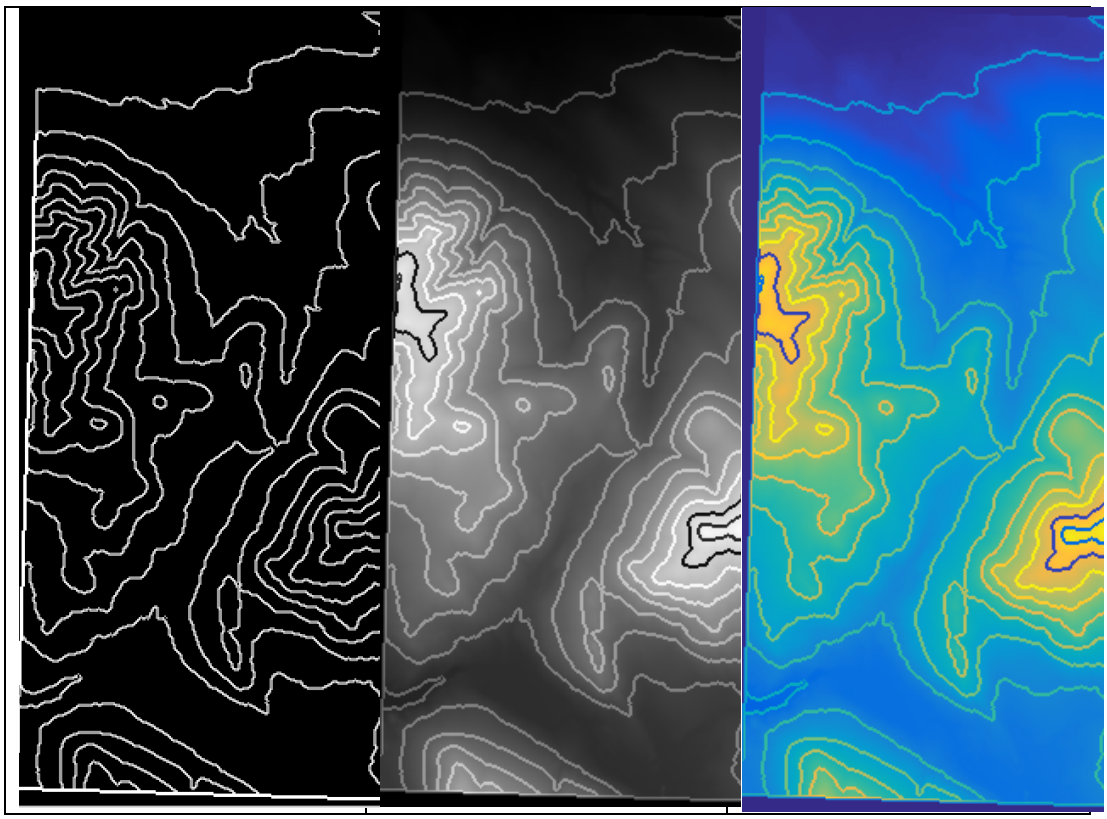
work4.pgm



Another Example

Here's a contour map of Mt. Dartmouth, with contour lines every 250 feet. In the right 2 images, I've combined the contourmap with the original map and colorized it with the coloring scheme in the "parula" colormap found in the zip file (note: you do NOT need to be able to produce the combined or colored PGMs):

dartmouth_contour..pgm



Note: my contours are drawn 2 pixels wide because I calculate my gradients slightly differently than described above. Also, the exact location of the contours will depend on where your starting contour is. Mine are located approximately at the 250, 500, 750 ft, etc. elevations. If you use the minimum pixel value as your 'zero' point, your contours will be different than mine.

Assignment:

Write a program which does the following:

1. prompts for the name of a DEM file (one of the .EGM files in the zip above),
2. Reads in the file, and writes it back out again as a grayscale PGM, rescaled to range 0-255.
   (or you could write it out as a color PPM, using one of the colormaps in the zip file)
3. reports the range of elevations in the file
4. asks the user for the desired contour step size
5. Outputs the contour map to a PGM

Optionally, you could also have your program output the results of the intermediate steps as separate PGM images. This would certainly help during the debugging process.

If you get it all working, you could overlay the contour map on the original grayscale (or color) DEM by summing together the contour pixel array with the original DEM array, rescaling and writing out the combined result (as shown above).

Grading:

A well written program deserving of an A will use functions to accomplish the individual steps, and be able produced PGM images from the EGM files (DEM maps). It will be nicely formatted and well commented. A program that can produce horizontal and vertical gradients would earn about a B, provided it is using functions and is reasonably well written. One that can read the EGMs and create PGMs from them would earn a about a C.

Keep in mind that this is an intermediate step towards the final project, so don't panic if you don't get everything working. If you don't, then write a brief explanation of what you did get working and put it in a text file to include in your submission. If you struggle on this part but manage to get everything working on the final project, I'll weight your final project a little more heavily.