Mountain Paths (Part 1)

Objectives

- 2d vectors
  - Store
  - Use
- Nested Loops
- Parallel data structures (i.e. parallel vectors or called multiple vectors in the zyBook)
- Transform data
- Read from files
- Write to files

Submission

1. **Source Code**: Submit the source code to Mimir as a file named "**main.cpp**"

Background

There are many contexts in which you may want to know the most efficient way to travel over land. When traveling through mountains (let's say you're walking), perhaps you want to take the route that requires the least total change in elevation with each step you take — call it the path of least resistance. Given some topographic data it should be possible to calculate a "greedy lowest-elevation-change walk" from one side of a map to the other.

In part I of this homework, you will implement code to read the topographical data and manipulate it so that it can be displayed. In part II (next homework), you will implement the code to identify and displays certain paths.

Specifications

Design (optional)

As you become experienced you may not always choose to do flowcharts. Regardless, you should sketch out algorithms with pseudocode or a flowchart.

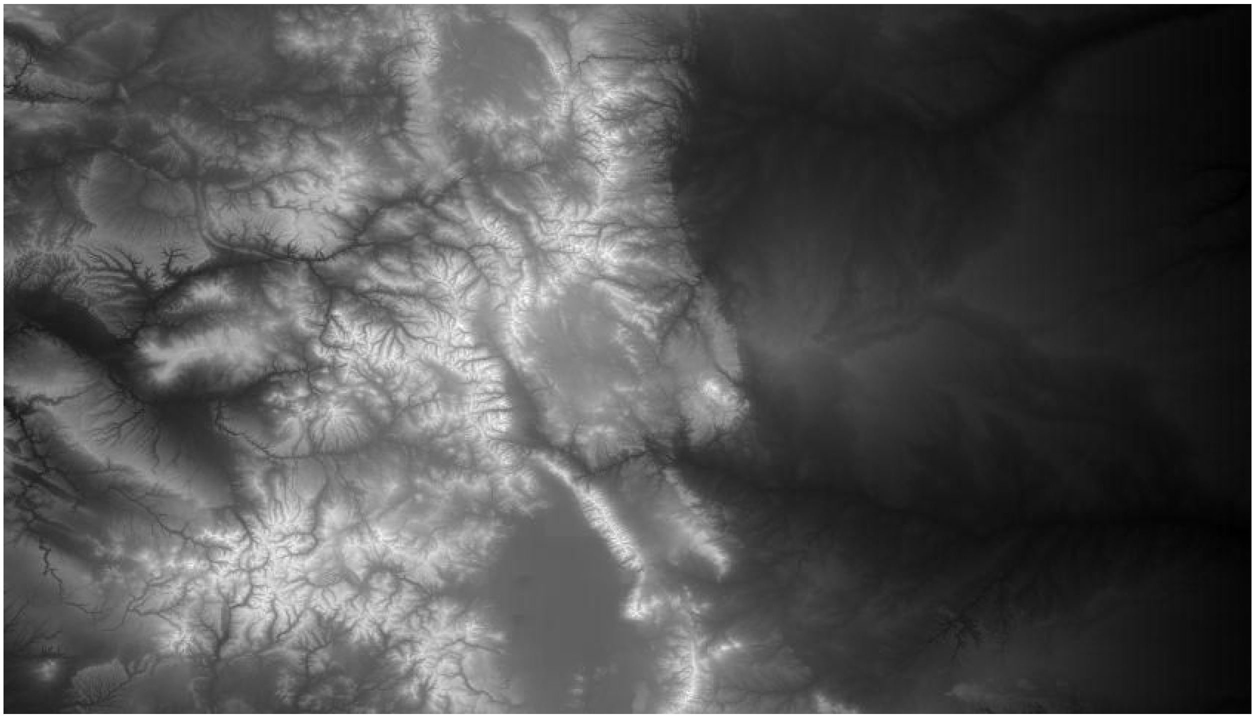
In this homework, you will read a set of topographic (land elevation) data into a 2D vector and manipulate the data so that it can be visualized. In the next homework, you will extend your code so that it computes some paths through the topographic terrain (mountains) as well as visualize those paths.

Create a PDF document containing your design and additional components.

- Design. The following might help you as you create your design:
    - How do you get the elevations and store in a 2d vector?
    - How do you know if the file had too much or not enough data?
    - Elevations can be below sea level. How will you deal with negative elevations?
    - Do you algorithms work with asymmetric images, i.e. width and height are not the same?
- Additional Components
    - The smallest example input file has 100X100 numbers in it. This is a bit too large for initial testing. Create a test input file with sample elevations that is small enough so you could hand calculate the results of any of your algorithms with the file. A 4X4 file might work.

Program (Required)

The minimum requirements for this homework are that you write code to read elevation data and produce a file representing an image that, when visualized, displays each position on the map with a color that represents the position's elevation. The visualization will produce a picture like the one below:



Program Flow

1. Read the data into a 2D vector
2. Find min and max elevation to correspond to darkest and brightest color, respectively
3. Compute the shade of gray for each cell in the map
4. Produce the output file in the specified format (PPM)
5. View your image result

Step 1 - Read the data into a 2D vector

Your first goal is to read the values into a 2D vector (i.e., a matrix) of *ints* from a data file. *Note: You should use `vector_name.at(index)` to access the elements of the vector to benefit from the additional runtime checks.*

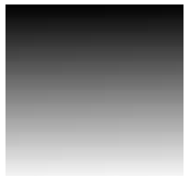
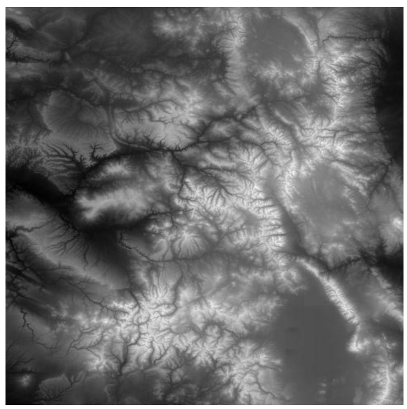
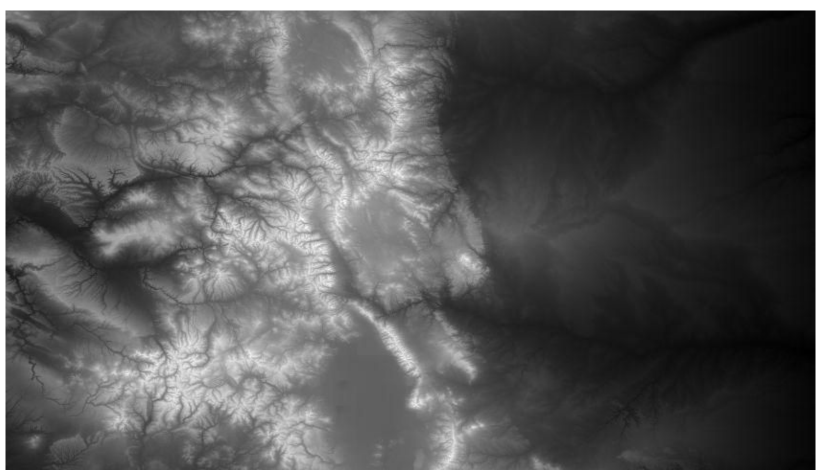
This data file contains, in plain text, data representing the average elevations of patches of land (roughly 700x700 meters) in the US. The user of your program will provide as input three values (your program will read these values from standard input, i.e., using cin, and do so in the following order):

1. The number of rows in the map; *(The height of the image to be produced.)*
2. The number of columns in the map; *(The width of the image to be produced.)*
3. The name of the file containing the map data.

After reading these data items from the user, you should read the elevation data from the specified file. We provide several sample data files containing elevation data for most of the state of Colorado (mountains!). Other input files can be obtained from the National Oceanic and Atmospheric Administration (http://maps.ngdc.noaa.gov/viewers/wcs-client/).
A data file comes as one large, space-separated list of integers. A file representing a 480-row by 844-col grid contains 405,120 integers (480*844). Each integer is the average elevation in meters of each cell in the grid. The data is listed in row-major order, i.e., first the 844 numbers for row 0, then the 844 numbers for row 1, etc.

Hints/Warnings:

- The data is given as a continuous stream of numbers - there may be no line breaks in the file.
- **You should validate the input from the user (number of rows and columns), and you should also validate that the input file exists and that it contains all the data for the expected numbers of rows and columns.**
    - It is ok to exit the program execution with an error message if there is an input error, i.e., you do not need to give the user a chance to enter corrected data.
    - **Start all error messages with "Error:" (note the colon). Then exit the program with a non-zero exit code (just an integer) to indicate you are exiting the program with an error. You can call exit(n) where n is the exit code. Returning a value from main will do the same thing.**
- You may want to make sure that you are reading the data correctly before you move to the next step of this project. Implement code to print your map data and try out your code with a file containing a small map first (you can create an input file corresponding to 4 rows and 4 columns, for example.) Then check what happens when you read the sample map data files: (100 by 100, 480 by 480, 844 by 480).
- You can download the example map data files (shown below) packaged together (in zip format) via https://drive.google.com/open?id=0B_ouNNuWgNZCRWwyVFVVTGpNb3M (note: will need to be logged into TAMU Google account for access).

Download Link for Map Data Files	Resulting Image
100X100	
480X480	
844X480	

Step 2 - Find the min and max values

In order to draw the map with a grayscale that indicates the elevation, you will need to determine which scale to use, i.e., which will be shown as a dark area (with low elevation) and which ones will be shown as bright areas (high elevation). This means you need to find the min and max values in the map first, so that you know which

values to associate with the brightest and darkest colors. You can compute the min and max elevation values by writing code that scans the entire map data and keeps track of the smallest and largest values found so far.
Test this functionality to make sure you're getting the correct values for min and max, before you proceed to implement the rest of the program. You may want to check with a friend or colleagues in the Piazza forum to see if other students are getting the same min and max values for the provided input files.

Step 3 - Compute the color for each part of the map and store

For each value in the vector of elevation data, you will calculate the shade of grey that corresponds to that elevation point using the following equation:

$$shade\ of\ grey = \frac{(elevationPoint - minimumElevation)}{maximumElevation - minimumElevation} * 255$$

Use the min and max elevations calculated from Step 2. Also **round** the value, you get to the nearest whole number since RGB values are integers.

Note: you can use the `round()` from `#include<cmath>`

As you compute the shade of grey, store that value in three parallel vectors for R, G and B. Putting the same value for R, G and B will result in grey. The structure of the vector should mirror the vector with the elevation data. This structure is required in the next part of the homework. If you use an alternative structure, then you will have to redo that part in the next homework. Search the zyBook for more information about parallel vectors searching for "multiple vectors". Unfortunately, the zyBook uses this more ambiguous term.

Step 4 - Produce the output file in the PPM format

PPM (portable pixel map) format is a specification for representing images using the RGB color model. PPM is not used widely because it is very inefficient (for example, it does not apply any data compression to reduce the space required to represent an image.) But PPM is very simple, and there are programs available for Windows, Mac, and Linux that can be used to view ppm images. Even more conveniently, you can use an online tool with your browser to convert a PPM file into a widely used format such as JPG. We will be using Plain PPM, which has the information in text format (readable as decimal numbers instead of binary.)
In Step 3, you computed the RGB values for the shades of gray. All you will need to do to get a PPM image for these RGB values is to write a preamble before writing the RGB numbers into a file. We will follow the convention that if the input file is named *input.dat*, then your program will generate a file named *input.dat.ppm*

A PPM file has the following format:

- First line: string “P3”
- Second line: width (number of columns) and height (number of rows)
- Third line: max color value (for us, 255)
- Rest of the file: list of RGB values for the image, expressed as a raster of rows, from top to bottom. Each row contains the RGB values (i.e., three values) for each pixel in the column.

Example:

Note: We have added colors showing that each three numbers represents a single pixel.

```
P3
4 4
255
0 0 0  255 0 0  0 0 0  0 255 0
255 255 255  255 0 255  0 0 0  0 255 0
255 255 0  0 0 255  125 0 255  255 0 125
0 0 255  255 255 0  125 125 125  239 239 239
```

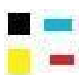
This same file could also look like (all pixels on a single line):

```
P3
4 4
255
0 0 0 255 0 0 0 0 0 0 255 0 255 255 255 255 0 255 0 0 0 0 255 0 255
255 0 0 0 255 125 0 255 255 0 125 0 0 255 255 255 0 125 125 125 239
239 239
```

Or even this: (But this one makes no sense to a human reading it.)

```
P3
4 4
255
0 0 0 255 0 0
0 0 0 0 255
0 255 255 255 255
0 255 0 0 0
0 255 0 255 255
0 0 0 255 125
0 255 255 0 125
0 0 255 255 255
0 125 125 125 239
239 239
```

Sample PPM File:

blocks.ppm 

Step 5 - View Your Image

If you are using the VM, right click on MountainPathsOne and choose refresh to see the file created. Then double-click the .ppm file. It will automatically open in an image viewing program.

Otherwise, use another way to view the file.

- You can find and install a program capable of showing this format (e.g. GIMP).
- You can use an online tool that will convert the .ppm file into a .jpg file.
  - The tool is available at: http://paulcuth.me.uk/netpbm-viewer/
  - Another Tool: https://www.coolutils.com/online/PPM-to-PNG#

The usage is very intuitive: