Garrett Tarczuk
10 April 2020
CSC499
Professor Hervé

Shadow Based Locator

For my research project I created a program that would calculate the location of a series of images based on; time, date, height of an object in the images and the length of the shadow in the images. Those last two measurements were calculated in pixels as the ratio was needed not the actual measurements. I created this software in java for the backend with java being used as a frontend as well. The user is able to open the software by running an executable jar and select images using a directory manager. From there it will load images in one at a time and the user will have to click the top of the object then the bottom on the object, and finally the end of the shadow. The program will run imagemagick to try and scrape the metadata of the image to retrieve the time and date otherwise the user will have to manually enter the data themselves. The user will need to adjust the time for UTC or Greenwich time in england (0 longitude).

How the program works. At any given spot on earth with a given date and time the sun will be at a specific declination angle, the angle the sun is above the horizon, and solar elevation, the angle the sun is above the equator. Given those two measurements and the time and date, one can construct a trace of all the positions on earth that the sun will be at those positions. Normally this will be some elliptical shaped trace. The image would have been taken at one of those spots on the trace. Using the same logic that is used to find the epicenter of an earthquake I take three images and find where they all overlap and that will be the most likely position on earth that the image was taken at. The program will output to the terminal the longitude and latitude of the image's location and a google maps URL of the position. If you are running this program on linux and have GNUplot installed it will also open up a GNUplot of the traces along it the location found. For some reason this was not working on Mac. I was unable to identify why other than that I believe it has something to to with running commands in the terminal from java and that mac has some issues with the exec process command in java.

Key parts of the program:

The trace of all the positions on earth which the sun's declination and angle and solar elevation are true to what was measured is multithreaded into three different threads one for each image. The trace goes through every point latitude and longitude on earth incrementing by a small DELTA which optimizes speed and performance. I found the best delta to be 0.05. This would cause 6,480,000 different possible latitudes and longitudes to be checked for each of the three images. Thus why I found the need to multithread and find and ideal delta.

The location approximation of the traces to find the overlap was a challenging this to optimize. I originally looped through all the possible compilations of three points one from each trace and calculated the area of the three points and the one with the smallest was the location. This failed because if the points were collinear the area would be 0 and my program would assume this was the best approximation. I changed this to be perimeter and that fixed the issues with collinearity. However, I still had the issue that the lists of points that made up the traces was normally a list of 2000 points and to get all the possible different pairs of 3 points would be 2000^3. This is equivalent to over 3 minutes of computation as it computes 8 billion different perimeters of triangles. In hopes to speed this up I multithreaded this. This also posed some challenges in regards to not losing and points. My solution was to split one trace up into X different lists of points where X was the number of cores the computer had. Then for each thread it was given one of the smaller lists and the full list of the two other traces. This eliminated any loss of calculations. However this did not have the improvement that I was seeking so I then changed the code to compare the first too traces in each thread. And remove points that have a cartesian distance of greater than 5. This reduced two of the three lists to around 100 to 200 points which made the total calculations around 200*200*2000 which is 100 times smaller and very doable in a short amount of time. This reduced the computation time from 3 minutes to around 3 seconds.

Result:

Unfortunately as my program sits right now, it is unable to accurately approximate the location of an image. The program keeps thinking that images taken in southern Rhode Island are in Antarctica. This sounds really bad but I do believe this is just a minor bug as the traces appear to almost cross in southern Rhode Island. Currently the problem I believe stems from the measuring of the shadow length in 2D of a 3D shadow. If the shadow is not perfectly perpendicular to the direction the camera is facing the measurements will be off and thus skew the results. We tried to get the math for translating a point to correct the shadow however I was unable to run and use the notebook and thus unable to test if the issue was able to be corrected from this math. I do believe that this correction would fix the issue because the results from manually measuring the shadow and object height were better than those results I got from using the images and measuring these in pixels. When manually measured the error was about 2 longitude and 0.5 latitude.