Fruit Fly Sperm Project Write Up

Authors: Matt Robinson and Jing Dao Chen

The Problem:

The Fruit Fly, *Drosphila Bifurca*, has some of the longest sperm ever recorded. However, the current practice is to measure the sperm by hand, which can be a time consuming process. The desire is geometrically compute the length of Fruit Fly sperms from .jpg images. The issue with this is that while the images do a good job of showing the sperm, it is difficult to get an accurate reading of the length due to the curvy and overlapping nature of the sperm. Using graphical algorithms learned in class, we hope to be able to not only acquire the length of the sperm, but also do so in a uniform manner between the easy cases where the image is lighter and some of the harder cases where the images are darker.

Method:

Algorithm:

The algorithm for extracting the length of a fruit fly sperm cell consists of 3 steps. Firstly, an adaptive threshold method was used to separate out object pixels (bright pixels) from background pixels (dark pixels). For each pixel, the threshold that is used to determine whether it is an object pixel is calculated from the mean of a square neighborhood around it. In the second step, a flooding method was used to find connected components of object pixels within the image. Each connected component was labeled with an integer ID and the longest connected component, which is assumed to be the sperm cell, is preserved and the other pixels are filtered out. In the third step, a thinning algorithm was applied to a cell complex of the remaining pixels to extract a skeleton of the sperm cell. The final length of the sperm cell was calculated from this skeleton.

GUI:

For the GUI, we wanted to make it as user friendly as possible, The GUI has a basic interface where users can open up a file on their computer, and display the .jpg/.png file. From there, the user has a multitude of options. The first option is that they can either find the sperm length via our algorithm as described above. The sperm should then be highlighted red showing our flooded parts of the algorithm, and the calculated length of the sperm should be printed below the image. The alternative option is that the user can draw on the image in the color/ stroke of their choosing to manually calculate the length. If the user makes a mistake, all they have to do is click the eraser radio button and the draw tool becomes an eraser. Similarly to the first option, once you are done drawing, the user can click the find draw length button, and the length of the sperm should appear below the image. To help with these implementations, there are tools to help the user. The user can increase or decrease the contrast of the image to increase the efficiency of our algorithm or to help clarify where the user should draw. The user can also zoom in or zoom out to see where exactly our algorithm flooded the sperm, or to also help you optimize your drawing.

Data:

Images	Ground Truth (Micrometers)	Our Length (Micrometers)	Percent Difference
24708.1_1 at 20X	1951	2432 (our algorithm)	21.9484%
24708.1_2 at 20X	1787	2392 (our algorithm)	28.9543%
24708.1_3 at 20X	1786	2892 (our algorithm)	47.2852%
24708.1_4 at 20X	1681	1916.993(drawn)	13.118%
24708.1_5 at 20X	1952	1503.267(drawn)	25.9739%
24708.1_6 at 20X	1991	1953(drawn)	1.9%
WT.C.1	1090	911.11 (drawn)	17.879%
WT.C.2	1847	1260 (drawn)	32%

Analysis:

For the automated algorithm, the result is independent of user input because the sperm cell length is calculated directly from the original image. However, the algorithm is not as accurate because the final skeleton contains extra pixels in loops that have to be eliminated. The algorithm can also be improved by calculating the final length using a polyline representation of the skeleton instead of a pixel approximation.

For the drawing algorithm, the result is directly dependent on user input. This can lead to instability of the final result as it depends on how accurately the user draws the sperm cell. However, the way we implemented the draw algorithm was that for every pixel that is drawn, it adds one pixel to the length. This is not inherently accurate as sometimes the length transverses the vertical and not just the horizontal or vertical. Our next steps to improve this would be to take into account the difference between the x and y between the previous point and the current point. If both x and y change then we need to add sqrt (2) to our length counter.

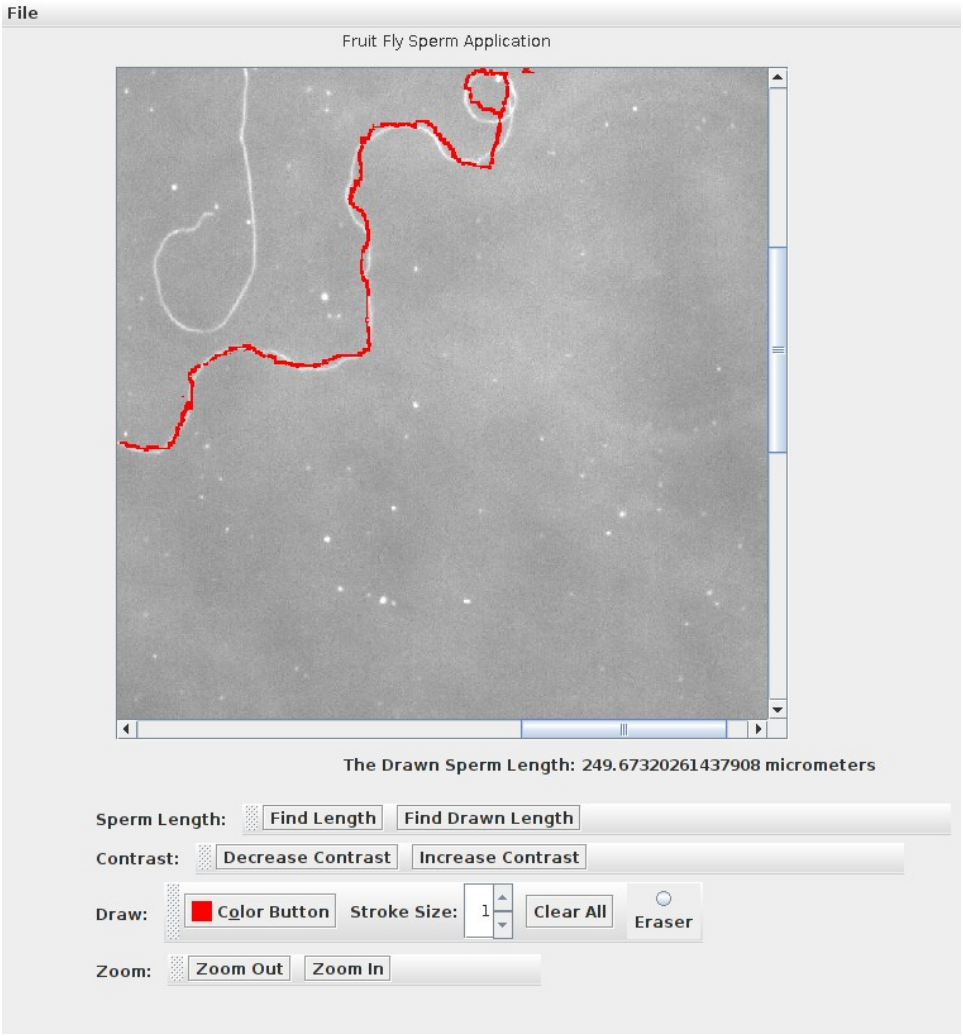
Another step for the GUI would be to add a cropping tool, where the user can crop out a section of the image they would like to run the algorithm on. This would increase the efficiency of our algorithm as we could crop out irrelevant noise that reduces the accuracy of the algorithm output.

Conclusion:

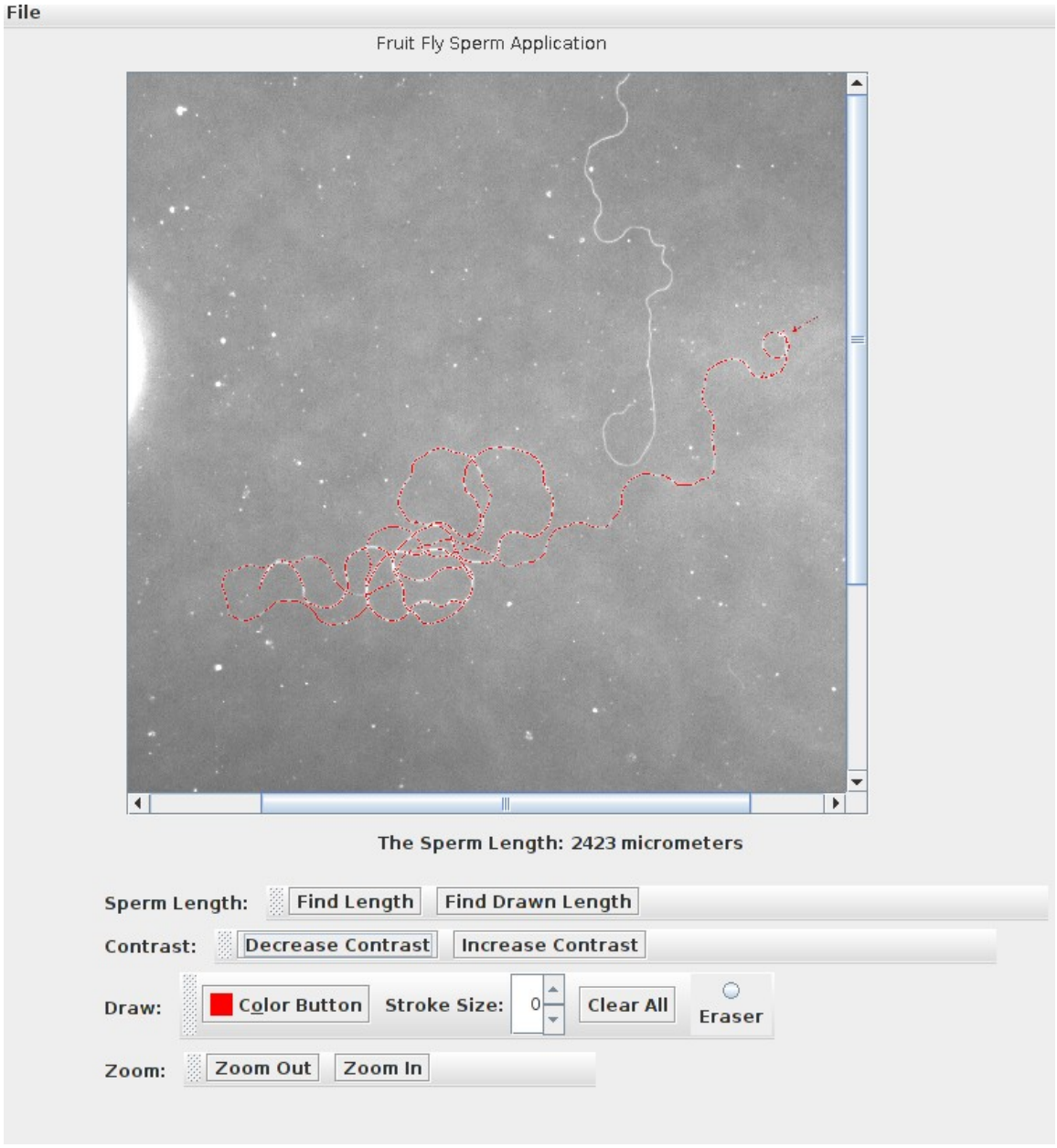
Overall, while our application is not as accurate as we would like it to be there are many benefits due to our implementation. Our project is lightweight, and does not require many dependencies. The program is easy to download, and everything is

contained in a single .jar file. The application requires very little work on the users part, as most of the image manipulation is done behind the scenes in our code. Were we to continue this project our hope is one day increase the efficiency and add even more tools for the user within the GUI to better their results.

Screenshots:



(example of what a manual drawing portion looks like in our GUI)



(example of what our automated portion looks like in our GUI)