Measuring Sperm Cell Length

CSE 559A Final Project Report

Andy Kim(435168, dohoon.kim), Nigel Kim(431659, seunghwan.kim)

Introduction

Our project implemented thinning algorithm on Sperm Cell images of different contrast and noise of background to obtain actual skeleton of sperm cell and its length. To optimize the accuracy of obtained sperm cell length to be close to ground truth length, thinning algorithm with window contrasting and window thresholding and moderate amount of user interaction with GUI was combined.

Initial bounding polygon on the region of interest around sperm cell using GUI helped algorithm to efficiently work on smaller number of pixels and more accurately with blackened background except the object pixels inside the bounding polygon. GUI has also achieved extracting local intensities on the sperm cell and contrasting or adaptive contrasting are made to image more separable in segmentation of sperm cell from background. Convenient feature such as zooming in ROI and changing brightness, and obtaining pixel intensity values helps user to easily detect the sperm cell. Most of the features proposed in project proposal were made done and with binding with the thinning algorithm, the result became more accurate.

All of the Required Features and Wish List Features proposed were successfully implemented except for the 2nd Wish List Feature, which was denoising the image. This was omitted because it did not help our algorithm to obtain a better result, but rather blurred the image, complicating the measurement process. Instead, a lot of new wish list features were implemented in GUI, with it having 18 tools for the user to interact with.

Required Features

1. User can use the GUI to select the region of interest(smallest polygon enclosing the entire sperm cell structure).
2. User can modify the contrast within the selected region to obtain a visually more separable sub-image.
3. User can click one or more points on the sperm cell, and obtain the local intensities consisting the sperm cell(because sperm cell often do not have uniform intensity values for the entire structure)
4. User can input an image to GUI, and obtain the cell length and its computed error compared to the ground truth value is output.

Wish List Features

1. User can zoom into the image for more careful region of interest(ROI) selection, and more accurate local sperm cell pixel intensity value
2. Denoise an input image.

Algorithm:

1. Initialization:

   The process of algorithms is initialized from generating rectangular image bounding the cropped polygon. The initial background of polygon in rectangle is set to be black to hold lowest intensity value and we convert this rectangle image to be in gray image.
2. Window Thresholding:

   Then, the project implements simple region based window thresholding with size k generates the binary map as each pixel is marked 1 or 0 corresponding to comparison of its gray value with neighbors' mean within window size. This thresholding method is devised based on the thought that sperm cells are mostly with higher intensities than background, but with the case of facing noise cluster tightly attached to sperm cell with similar intensity values, GUI can help marking with brush on noise can separate the sperm cell from noise and run algorithm.
3. Window Contrasting:

   The project implements additional feature in algorithm that takes account the local intensity of pixel and average intensities of neighbor pixels in window size and compare them to assign higher weights of intensities added to certain pixels with relatively higher intensities. Purpose of window contrasting is to make foreground sperm cell to be separable from the background pixels since the images are mostly containing sperm cell having relatively higher intensities than those of background's.
4. Number of Connected Components:

   After thresholding is done, the step of labeling components is done on the generated binary map of image. With 8-connectivity checking all the orientations of neighbors, we get number of connected components of object pixels marked from binary map. Checking the eight neighbors for connectivity and mark the label to the pixels belonging to the same component.
5. K Largest Components:

Then, with these labeled components, k largest components can be obtained through comparing number of pixels belonging to the components. The k labels from connected components should be merged in one largest component as sperm cell is continuous.

6. Cell Complex:

   Final Step of thinning to achieve skeleton of sperm cell starts with building cell complex on the extracted largest components that are marked as 1 in binary map. Cell is in 3 dimensions containing points, edges, and face as points, edges and faces are updated with coordinates, point indexes, and edge indexes respectively. Edges in cell complex is updated through whether the current pixel from cell-0 marked object as 1 has right or bottom pixel that is also marked 1. Looping over all the pixels, unique edges are stored in cell-1. Plane as in cell-2 are updated in same logic with 4 connected edges to build one plane.

7. Thinning:

   Then, at the last step, thinning is applied on this cell complex to remove simple cells from the cell complex to preserve only medial cells to remain for skeleton. Parent tables for points and edges are built to check whether the cell is a simple cell and iso-values are updated and used with the relative and absolute threshold to decide to remove cells.

8. Length Measurement:

   After thinning is done, we have remaining cells in cell-0 and we count the total number of cell left. And, dividing the total length by 3.06 generates actual length of sperm cell in micrometers.

GUI Development:

Our Graphical User Interface(GUI) was built mainly using the Tk interface (tkinter package) in Python. Tkinter is the most commonly used GUI programming toolkit from Python. Tkinter package can be easily downloaded in both Unix and Windows systems, with a simple command "python –m tkinter" from the command line. Tkinter is useful in that it is quite fast and it allows us to build an interactive GUI. Moreover, we utilized the Python Imaging Library(PIL, or Pillow) to read and write out images, manipulate them, and even draw onto the image.

Our GUI was built from the scratch, without using any help other than the documentation. We have implemented 18 tools(or buttons) in our GUI, with 7 buttons consisting of the main algorithm pipeline to measure the sperm cell length and 11 buttons consisting of basic image modification tools.

The 7 buttons of main algorithm pipeline are:

1. Threshold Image
2. Label Components
3. Get Largest Component
4. Build Cell Complex
5. Thin Image
6. Compare Output
7. Measure Cell Length

Buttons 1 through 5 was taken straight from our algorithm design, with slight modifications into objected-oriented style, to integrate with our Tk interface and the PILLOW libarary. Button 6 simply shows the original image by the side of the thinned output image, allowing the user to visually verify the accuracy and precision of the thinned output image before measuring the cell length. Button 7 counts the number of vertices from the thinned cell complex output, and since each vertex corresponds to a pixel, it then converts the number of pixels into units of micrometers using the scale 3.06 pixels per 1 micrometer. Since our GUI resizes the input image to fit the user's laptop screen in the beginning, button 7 also takes the resized scale into account to convert it back to the original scale and computes the cell length.

The 11 buttons of basic image modification tools are:

- Start Over
- Image Dimensions
- Resize Image
- Zoom
- View Pixel Value
- Adjust Brightness
- Adjust Contrast
- Adaptive Contrast
- Crop
- Eraser
- Draw

The "Start Over" button will call the stopclick() function that resets all values to default, and asks the user to start over by selecting an input image.

The "Image Dimensions" button will call the calc_dim() function that simply reads the original input image dimension, resized input image dimension and calls display_dimensions() function to display both dimensions at the top of the GUI. As mentioned previously, our GUI automatically resizes the original input image in its very first step to fit the user's laptop screen. Resized scale will be displayed alongside with the resized image dimensions.

The "Resize Image" button will call the resize() function to prompt the user to input the new image height and width, then display and save the new resized image.

The "Zoom" button does a similar job, but instead it calls the zoom_image() function that similarly prompts the user to input the new height and width, and displays the resized image without saving. It also creates a temporary button named "Original Image" at the top of the GUI, so that when clicked, it calls the display_original() function to allow the user to go back to the original image anytime.

The "View Pixel Value" button will call the select_cell() function, that instructs the user to click points on the screen. Whenever a point is clicked, it calls the getIntensity() function that displays the pixel intensity value of the clicked point at the top of the GUI.

The "Adjust Brightness" button will call the adj_brightness() function that prompts the user to input a new brightness value between 0 and 100. This button modifies the absolute brightness of the entire image. Default brightness will correspond to the brightness value 50. The adj_brightness() function also creates a temporary button named "Original Brightness" that when clicked, calls the original_brightness() function that allows the user to display the image in default brightness. This button modifies the absolute brightness of the entire image.

The "Adjust Contrast" button will call the adj_contrast() function that prompts the user to input a new contrast threshold value between 0 and 255.. This button modifies the absolute contrast of the entire image. The adj_contrast() function also creates a temporary button named "Original Contrast" that when clicked, calls the original_contrast() function that allows the user to display the image in default contrast.

The "Apply Adaptive Contrast" button will call the adaptivecontrast() function that was previously designed alongside with the adaptive thresholding algorithm. The algorithm is very similar to the adaptive thresholding algorithm in that it visits each pixel and applies a window centered on the pixel, and compares the mean of the window pixels and the center pixel to determine whether to increase or decrease the brightness of the center pixel.
The example of result from window thresholding using GUI is shown in Figure 1:

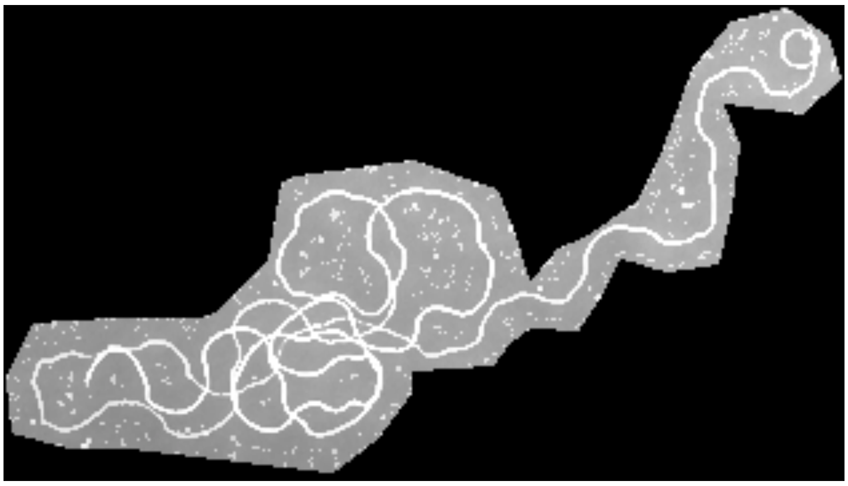


Figure 1. Example of adaptive thresholding that considers local pixel intensity relationships. Image: 24708.1_1 at 20X

The “Crop” button will call the select_points() function that instructs the user to click points on the image to construct a cropping polygon. On each click, the function will then call the getcoordinates() function to obtain the spatial coordinate information of the clicked point, and connect the clicked points in a yellow line using the ImageDraw object’s Draw() function provided from the PILLOW library. The getcoordinates() function will also create a temporary button named “Crop Image”. Once the user is done selecting polygon region, the user can click the “Crop Image” button, which will call the crop_poly() function to apply a mask onto the selected polygon and its background, then crop the smallest bounding rectangle that encloses the polygon. The background pixel values will be masked to 0, making it to be black. Then, it will display the cropped image onto the screen and save the cropped image output.

The “Eraser” button will call the select_erasePts() function, that prompts the user to input the desired eraser size. To make user interactivity more accurate, we have only allowed the user to put in odd numbers between 0 and 100, making the eraser symmetric along the clicked center pixel. Then, on mouse-click or left-click-drag, it calls the erasePts() function, that will update the eraser region around the clicked point to 0, making it to black.

The “Draw” button is very similar to the “Eraser” button. It will call the select_drawPts() function that also prompts the user to input the desired brush size. We have allowed the user to put in odd numbers between 0 and 20, also making the brush symmetric along the clicked enter pixel for better interactive accuracy. Then, on mouse-click or left-click-drag, it calls the drawPts() function, that will update the brush region around the clicked point to 255, making it to white.

Experimental Results:

ImageID	Length.Manual.mm	Measured Length	Error(%)

Hard	472.1A.1_1	1421.724	1613	13.45380
	472.1A.1_2	1721.22	1951	13.34983
	28369.2.6_2	1798.116	2160	20.12573
	28369.2.6_3	1820.409	2223	22.11541
	472.1A.1_5	1827.506	1941	6.21032
	472.1A.1_4	1836.393	1598	12.98158
	42568.b4.7	1840.172	3372	83.24374
	LHM.1B.3_2&3	1847	1789	3.14022
	LHM.1B.3_7	1849.383	1839	0.56143
	472.1A.1_3	1849.996	1933	4.486712
	472.1B.1_5&6	1870.82	2183	16.68680
	53387.1B.2_7&8	1873.806	2210	17.94177
Medium	24708.1_4 at 20X	1681	1915	13.92028
	24708.1_5 at 20X	1952	2322	18.95491
	24708.1_6 at 20X	1991	2675	34.35459
	WT.C.1	1090	1266	16.14678
	WT.C.2	1847	2329	26.09637
Easy	24708.1_1 at 20X	1951	2122	8.76473
	24708.1_2 at 20X	1787	1979	10.74426
	24708.1_3 at 20X	1786	1986	11.19820

Table 1. Measured cell lengths in units of micrometers, and the percent difference between the measured length and the ground truth length values of each sperm cell.

For Easy examples, we have obtained the percent difference in range of 8.7%-11.2%. For Medium examples, we have obtained the percent difference in range of 13.9%-34.4%. For Hard examples, we have obtained the percent difference in range of 0.5%-83%. Generally, we have achieved the percent difference within 20% for most of the cells, but two examples from Medium and two examples from Hard gave us over 20% percent difference.

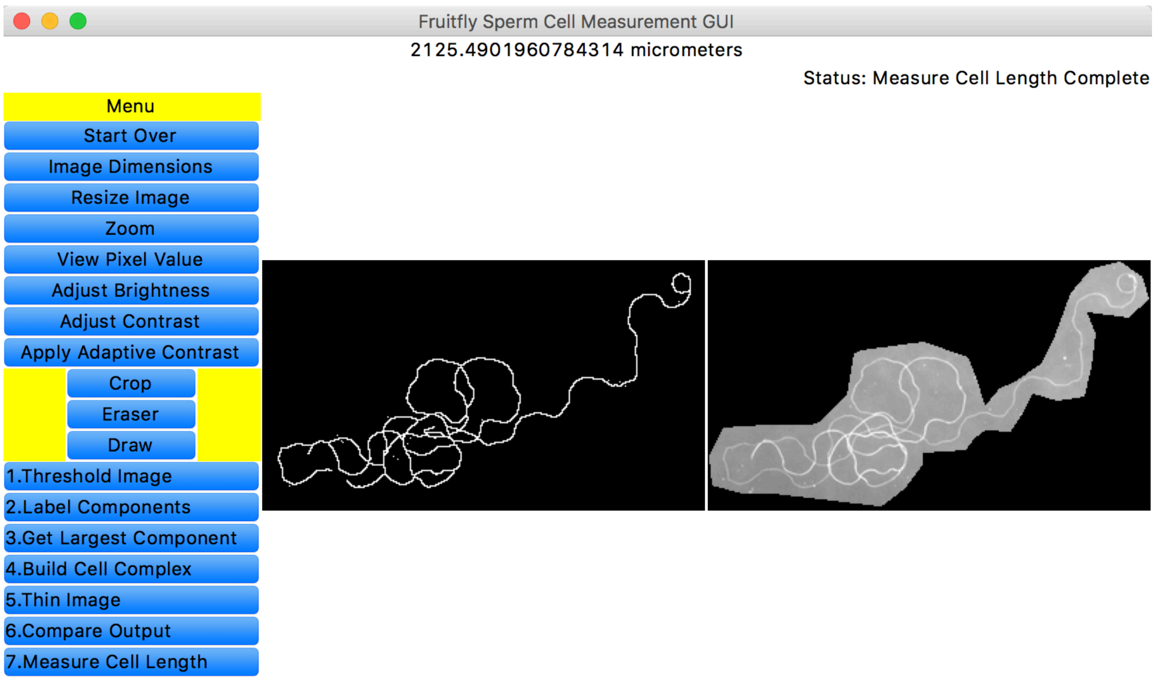
In most cases, we hypothesize that the percent difference was within 20% but the measured length was consistently larger than the ground truth value possibly because of two reasons:

1. We auto-resize our input image in the beginning to fit the user's laptop screen, and scale the measured length back to the original scale. In this process, the number of pixels covered by the skeleton may not exactly correspond with the scale. By scaling the measurement by 4 times(which is the original input resize scale) each time, we may be producing more pixels than we would have if we used the original scaled image in the first place.
2. During the thinning step, cells are being removed from the cell complex to build the skeleton and we assume there remain unnecessary cells left in cell-0 since the edges

connecting those that should not be included in skeleton were not removed as simple cells.

In the four cases where our percent difference was bigger that 20%, the input cells have a lot of repetitive overlaps (coil-like structure), and we hypothesize that the overlapping regions can be thresholded falsely due to our adaptive thresholding algorithm that looks at the neighboring window pixels to determine whether to keep the pixel. Also, as a chain reaction, much more cells will be connected as a connected component, ultimately leaving more pixels in the final thinned image.

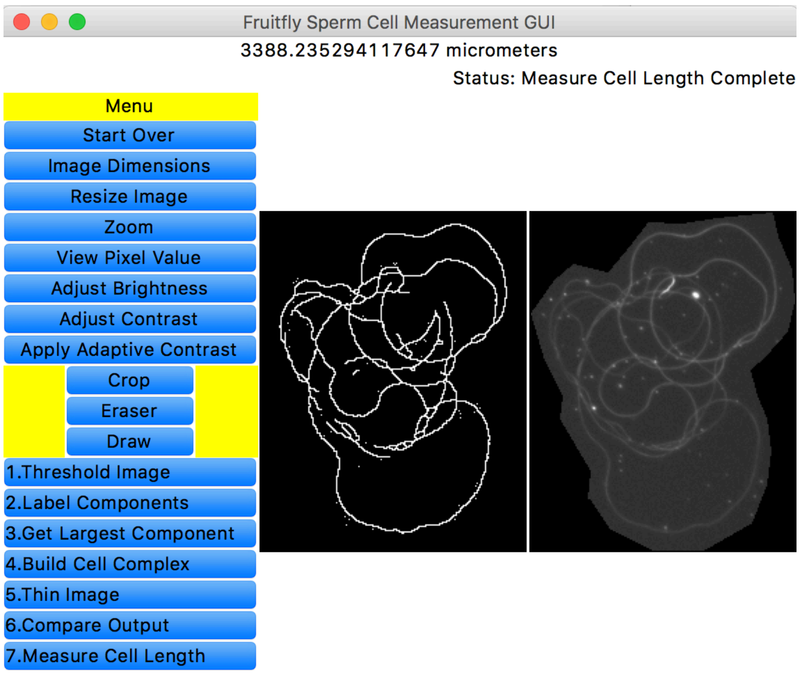
The examples of well performing output and bad performing output from user interaction in GUI and algorithm are shown in Figure 2 and Figure 3, respectively:


Fruitfly Sperm Cell Measurement GUI

2125.4901960784314 micrometers

Status: Measure Cell Length Complete

Figure 2. Example of the final output of a well-performing example. Image: 24708.1_1 at 20X



3388.235294117647 micrometers

Status: Measure Cell Length Complete

Figure 3. Example of the final output of a bad-performing example. Image: 42568.b4.7

Bugs and Future Work:

There was no bug in our program, but rarely, heavy computation from adaptive contrasting and adaptive thresholding could crash our Python kernel.

In the future, we aim to re-design the adaptive contrasting algorithm and the adaptive thresholding algorithm to look at the local intensity relationships more efficiently using matrix operations. Also, we aim to eliminate the initial auto-resizing process increase the overall accuracy, but currently, if we eliminate resizing, then our 2048x2048 input images always exceed the laptop screen size, and it is very hard for the user to perform image manipulation.

For the algorithm, we also aim in future to build adaptive contrast that can manage the image containing separate image zones with different degree of intensities such as top part of the image is conspicuously brighter than bottom part in general sense. Adaptive contrast to deal with this kind of image to be able to convert image in similar degree of intensities and adopting more general threshold that can deal with all the pixels in the image overall are expected to have different accuracies compared to our method.