Jonathan Song

Haoran Fang

CSCI 4830-012, Concurrent Programming

Project Checkpoint

At this point in the project, we have completed the sequential implementation of the digital image processing. The goal of this project was to be able to enlarge an image and perform smoothing operations on a specified region of the digital image and to compare the performance of a sequential implementation against a concurrent implementation of the algorithm. The image is enlarged by iterating through the entire image and increasing the image ratios so that we can return a copy of the same image that it slightly larger. There are many methods for image smoothing, and we chose to implement the Gaussian blur method, which performs smoothing on images using the Gaussian function.

The program is run through a central User Interface, which is currently implemented through JFrame. The user interface allows the user to select the image that he or she wishes to run the program on. There is an option for the Processing Method, which is automatically set to "Smoothing," as this is clearly the action that the user will want to perform on the picture. There is also an option that allows the user to select the Enlarge Ratio for the image and sets the bounds for the JFrame that will hold the image. At this point, the UI allows the user to choose the file that he or she wishes to use to test the program. Only one file can be selected at this point, but we may allow for the user to be able to select multiple files later on if we believe that doing so would be useful in testing performance. The user will then click on the JButton for "Enlarge-Smoothing" that will open another window with a resized version of the chosen file, which is determined by the bounds of the JFrame. This is done so that the image will be of higher resolution and the user is better able to select regions of the image. At this point, hovering over the image will show the approximate coordinates on the pointer over the image. The user can then

draw a rectangle over the image, which specifies the particular part of the resized image that he or she would like to perform spotting on. The user may draw a rectangle of whatever size and location that he or she may want as long as it is within the bounds of the enlarged image. Once this is done, the user may select the JButton for "Process," and the program will return the smoothed and enlarged image.

The majority of the work in this program is done in the files smoothing.java and Enlarge.java. When called, the program will first call the smoothing.java file, and this file specifies the actions that are performed on the image. In our implementation, the program performs smoothing via the Gaussian Blur method and then enlarges the smoothed image in a sequential manner. The Gaussian Blur is used to reduce image noise and detail by utilizing the Gaussian function to apply the smoothing transformation to each of the pixels in the specified regions of the image. The implementation for this method is in GaussianBlur.java, and is called by the smoothing file prior to the enlarging method calls. In order to perform the Gaussian Blur, the program first imports data of the image and obtains the dimensions of the selected parts of the image, which we have named "captureRect." We have specified an input value for sigma of 1.4 that will be used in the Gaussian algorithm, which is the standard deviation of the function that determines how much the pixel values of the picture will be reduced. Once this is done, the Gaussian algorithm loops through the entire selected region of the image and performs the Gaussian function on each of the pixels. The Gaussian algorithm takes in a file named "smoothingPartSource" as input, which we have specified to be a copy of the "final_img" that was chosen by the user. The result of this algorithm is stored in a new image variable called "smoothingPartTarget," which will be later used for the enlargement algorithm.

Once the Gaussian blur algorithm is finished, the smoothing algorithm continues by taking the dimensions of the smoothed portions of the image and sets it to the appropriate sections of another variable called the "smoothingTarget". The image variable of "smoothingTarget" was initially set to be the same as "final_img", or in other words, it is the same as the original image that was chosen by the

user at the start of the program. Doing this is the same as taking the specified sections of the program that had undergone smoothing operations and putting it back into the original image. At this point, the smoothing operation is complete and the program proceeds to enlarge the smoothed image. Currently, the program runs sequentially, as it performs smoothing on the image and then enlarges it in succession rather than having both operations being performed at the same time. The Enlarge.java file begins by initializing the inputImage as the input file and the outputImage, which will return the enlarged image. Within the UI, the user specify the enlargement ratio, which we have set to the value of 2. At this point, the enlargement algorithm loops through the entire image and re-creates the image by increasing the ratio of each pixel and coloring them appropriately so that the final product will look the same as the original input image. At this point, the final image is returned by the program and the program has successfully performed smoothing and enlarged the image sequentially.

The final product admittedly appears somewhat strange and impractical, but this proves that the program corresponds to the nature of the Gaussian blur. In actual practice, the Gaussian blur is typically used as a pre-requisite for further enhancement of images. The main function of such an algorithm is to serve as a low pass filter for the input image. Now that the sequential part of the program is complete, we are now prepared to implement another version of the program that runs concurrently. The best way to test and compare the performance between these two implementations is to create a timer variable that will begin after the user specifies a region on the image and clicks the "Process" button to begin the process till the program completion. We may revisit the program to allow the user to select multiple files at once for the program to run to provide more comparison data if this proves to be the best way to test between the implementations. Otherwise, we will use larger image files while maintaining consistency in the tests for the two implementations. Thus far, we have been able to keep the schedule that we had set up for ourselves at the beginning by having the sequential run

completed so that we can have the concurrent run and test results prepared for the presentation at the end.