Accelerating Image Convolution for Edge Detection

Fernando Porres
Undergrad Student, Dept.
Computer Science
University of Central Florida
Orlando, FL, USA
fe973753@ucf.edu

Noah Vaden
Undergrad Student, Dept.
Computer Science
University of Central Florida
Orlando, FL, USA
no572033@ucf.edu

Cameron Dudeck
Undergrad Student, Dept.
Computer Science
University of Central Florida
Orlando, FL, USA
ca535398@ucf.edu

Euan Selkirk
Undergrad Student, Dept.
Computer Science
University of Central Florida
Orlando, FL, USA
eu128372@ucf.edu

Jason Laveus
Undergrad Student, Dept.
Computer Science
University of Central Florida
Orlando, FL, USA
ja005353@ucf.edu

Abstract__This paper will describe the process in which we (students listed above) programmed a Canny Edge Detection Algorithm to detect and display prominent and necessary edges that create an outline of the major objects in an image. We then have then improved the runtime of the algorithm by implementing parallelization concepts that will allow simultaneous calculations.

I. Introduction

The Edge detection algorithm that we have chosen to enhance with parallelization is the Canny Edge Detection algorithm. This algorithm can be broken into five major sections that process the image sequentially. In order, these sections are the loading of a black and white image, applying a gaussian filter, calculating the intensity gradients of each pixel, applying a magnitude threshold, and finally processing the remaining edges through hysteresis. We specifically chose the Canny algorithm due to its more complex and time consuming yet more accurate running. This will allow the magnification of the improvements we implement using parallelization concepts. In order to distribute the work among members effectively we have each taken a separate major portion of the functionality of the algorithm and as such, each function will be described in the first couple sections of this paper with both how the function works normally and how parallelization techniques were applied through multithreading.

II. Edge Detection Functions

II.I Get Image This is the first step in the Canny Edge Detection process as this is where we grab our input image from a file and convert to a format which is easy to use and modify during the rest of the algorithm. For the initial processing of the image from a file we are using stb_image library which processes the image file into an unsigned char array that stores the color data for each pixel in the image. The next step in this function converts the image to grayscale by averaging the rgb color values. This single calculated value also represents the intensity of a pixel. Instead of storing the pixel values back into an unsigned char array, we have made a Vector2 struct that will store the intensity in the x and the gradient (which will be set to 0 at the moment) in the y. We then store this array of Vector2s along with the height and width of the image into our own custom IntensityGradientImage struct to ensure that we have a universal, easy to use image format that will allow ease of manipulation throughout the algorithm.

II.II Gaussian Filter A Gaussian filter is a smoothing filter used in image processing and computer vision to reduce noise and blur images. It works by convolving an image with a Gaussian function, which is a bell-shaped curve defined by:

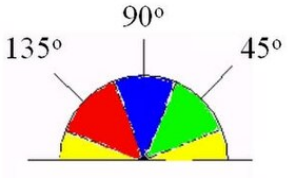
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where **x** and **y** are the spatial coordinates, and **σ** (standard deviation) controls the spread of the filter. The function first creates a kernel based on the Gaussian function. Larger **σ** results in stronger blurring. Then we Apply Convolution operation. For each pixel (x,y), the algorithm multiplies the surrounding pixels with corresponding kernel values and sums the results. This operation is performed only on valid pixels, (ignoring edges where a fill 5x5 region is not available).

Gaussian Filtering is Useful because it performs Noise Reduction which removes random variations in intensity. Smooth the image and help create a gradual transition between intensity variations. Finally It Preserves the Structure of the image. Unlike box filters, Gaussian filters give more importance to the center pixel, preserving edges better.

In summary, this function smooths an image while maintaining important structures, making it ideal for preprocessing before further analysis.

II.III Intensity Gradient The intensity gradient function is the next big stepping stone in setting up the image for having the strongest edges chosen. This function goes through every pixel and calculates the gradient for each one. The gradient of the pixel describes the direction that an edge is likely going in when it hits this pixel. However, the calculated gradient is not an exact angle but instead a rounded degree that will be to the nearest approximate line of horizontal, vertical, and the two diagonals (0°, 45°, 90°, 135°).



This will allow the future functions to choose and fit pixels together to figure out where the strongest lines are located and display those.

II.IV Magnitude Threshold The magnitude threshold function applies gradient magnitude thresholding to filter weak edges in an image, based on a given threshold value. It uses the IntensityGradientImage structure, which stores gradient vectors for each pixel. The function retrieves image dimensions and the pointer to the gradient array. Each pixel's gradient magnitude, stored in the x-component of the gradient vector, is compared to the threshold. If the magnitude is below the weakThresh value, the gradient vector is suppressed by setting both its x and y components to zero, removing weak edges from the image. This approach allows for only strong edges to contribute to the final edge map, increasing the accuracy of the edge detection. To improve the efficiency of the function, it utilizes parallelization through OpenMP, by parallelizing the nested loops in order to distribute computations over multiple threads. This allows the function to act quicker on larger images.

II.V Hysteresis The hysteresis function builds upon the process of removing weak edges started in the magnitude threshold function by going through the remaining edges with a more "fine toothed comb" so to speak. It starts with having a classification for strong edges. These are edges that have been determined to be strong enough that we want to keep them no matter what. This function then proceeds to navigate along the edges formed through other pixels that follow this starting strong edge and if they haven't already been removed the will be kept as edges for the final product. This process continues until all strong edges have been looped through to find their connected important edges and all other edges are deemed noise and thus removed.

II.VI Save Image The final function we have included is made for the sole purpose of outputting the image after its been modified. This means that this function is not officially part of the Canny Edge Detection algorithm, but is instead something we have added to formulate an output from our custom image struct. The first step is to break down our custom image struct back into an unsigned char array representing the grayscale pixel values of the image. This is necessary because of the stb_image library that we use for image input and output functions on these unsigned char arrays. We then simply call the write function from the library to write the image to an output file where we can see the progress of the image. This function will be very useful for seeing the output and for inspecting the image at the end of each function to ensure that everything is running correctly and just to have the steps of the process in a visual format.

III. Parallelizing The Functions

III.I Get Image We have not implemented the parallelization of this function yet, this is our next objective after the milestone.

III.II Gaussian Filter We have not implemented the parallelization of this function yet, this is our next objective after the milestone.

III.III Intensity Gradient We have not implemented the parallelization of this function yet, this is our next objective after the milestone.

III.IV Magnitude Threshold We have not implemented the parallelization of this function yet, this is our next objective after the milestone.

III.V Hysteresis We have not implemented the parallelization of this function yet, this is our next objective after the milestone.

III.VI Save Image We have not implemented the parallelization of this function yet, this is our next objective after the milestone.

IV. Conclusion

We have not reached a conclusion yet.

References

[1] “Canny Edge Detector”, https://en.m.wikipedia.org/wiki/Canny_edge_detector