Spring 2021 ME/CS/ECE759 Final Project Report

University of Wisconsin-Madison

Straight Line Boundary Detection for High-resolution Images

Jingjun Liu

May 7, 2021

**Abstract**

The boundary of an object is a vital characteristic in image processing. Hough Transformation is a common way of detecting boundaries. In this project, a program was developed with openMP parallel computing to detect lines in images. To reduce the runtime of this program, nested parallel computing was used. The output is compared with Matlab built-in package for correctness check. A clear runtime saving can be observed by comparing the runtime on GPU and CPU. For small and simple image, the boundary detection is almost complete. However, for large scale, high-resolution images, the boundary detection are generally correct but not perfect, as some obvious boundaries are not detected. Possible reasons for this fail and future work on boundary detection for high-resolution images are discussed.

Link to Final Project git repo: https://euler.wacc.wisc.edu/jliu798/me759-jliu798.git

Contents

[1. Problem statement 4](#_Toc531620937)

[2. Solution description 4](#_Toc531620938)

[3. Overview of results. Demonstration of your project 6](#_Toc531620939)

[4. Deliverables: 10](#_Toc531620940)

[5. Conclusions and Future Work 10](#_Toc531620941)

[References 11](#_Toc531620942)

# General information

1. Your home department:
   * Mechanical Engineering.
2. Current status: PhD student
3. Individuals working on the Final Project (include yourself)
   * Jingjun Liu
4. Choose one of the following two statements (there should be only one statement here):
   * I release the ME759 Final Project code as open source and under a BSD3 license for unfettered use of it by any interested party.

# Problem statement

The boundary of an object is a vital characteristic in image processing. There exist many ways to detect whether a pixel of image is an edge pixel or not. For example, Sobel edge detector1 uses a pair of 3 x 3 convolution masks to traverse the whole image and detect edge pixels. However, naïve edge detection cannot tell us the boundaries of an object, especially when part of the object is blocked or there are many noise pixels. Hough transformation2 is an algorithm that can detect boundaries regardless of blocking and is robust towards noises. Nevertheless, it has a high order of complexity will use many resources especially when dealing with large images. An efficient way of finding boundaries of objects in a large image need to be developed with parallel programming on GPU.

# Solution description

1. Algorithm:

Hough transformation for detecting straight lines is the main algorithm used in this project. In general, the straight line y = mx + b can be represented as a point (b, m) in the parameter space. However, vertical lines pose a problem. They would give rise to unbounded values of the slope parameter m. Therefore, a line is represented in Hesse normal form:

where is the distance from the origin to the closest point on the straight line, and is the angle between the axis and the line connecting the origin with that closest point, as shown in figure 1.

Chart, line chart

Description automatically generated

Figure Hesse normal form of a boundary line

An accumulator array is created. One dimension of this array is the quantized angle θ and the other dimension is the quantized distance r. each bin of this matrix will get a vote if there is an edge pixel lies on the line formed with these two parameters. A nested parallel computing is used as shown in Figure 2.

Diagram

Description automatically generated

Figure

The steps of this algorithm is as follow:

a. Use an edge detection algorithm to detect each pixel of an image is an edge pixel and generate the edge map.

b. Initialize the accumulator array with range of θ and possible maximum distance r. In this project θ ranges from - to with step size 0.01.

c. For each pixel of the edge map, run parallelly to see if it is an edge pixel.

d. For an edge pixel, calculate in parallel the distance for each θ.

e. Update the accumulator array atomically.

f. Find the bins with most votes in the accumulator array, annotate in the original image.

2. Code structure:

When I firstly implemented this algorithm, I found that I could not install some packages for image processing on Euler. Also, I cannot see the annotated result on Euler with only terminal user interface. To ensure that this program can be run under any environment and does not need much external packages, the edge maps and size of all images have been processed and stored in files. The program will read the edge map and sizes directly and begin the Hough transformation process. The resulting accumulator array is also saved to a file. A Matlab script is provided to display the annotated result.

3. data structure:

The major output of this program is the accumulator array. The number of columns of this array is 315. The number of rows is two times the diagonal of the image as it is the largest distance a boundary line can reach.

# Overview of results. Demonstration of your project

1. Boundary detection result

Figure 3 and Figure 4 are two boundary detection results of high-resolution pictures. The layout of image is:

|  |  |
| --- | --- |
| Original image | Edge map |
| Boundaries detected by my implementation of Hough Transformation. | Boundaries detected by Matlab built-in Hough Transformation function. |

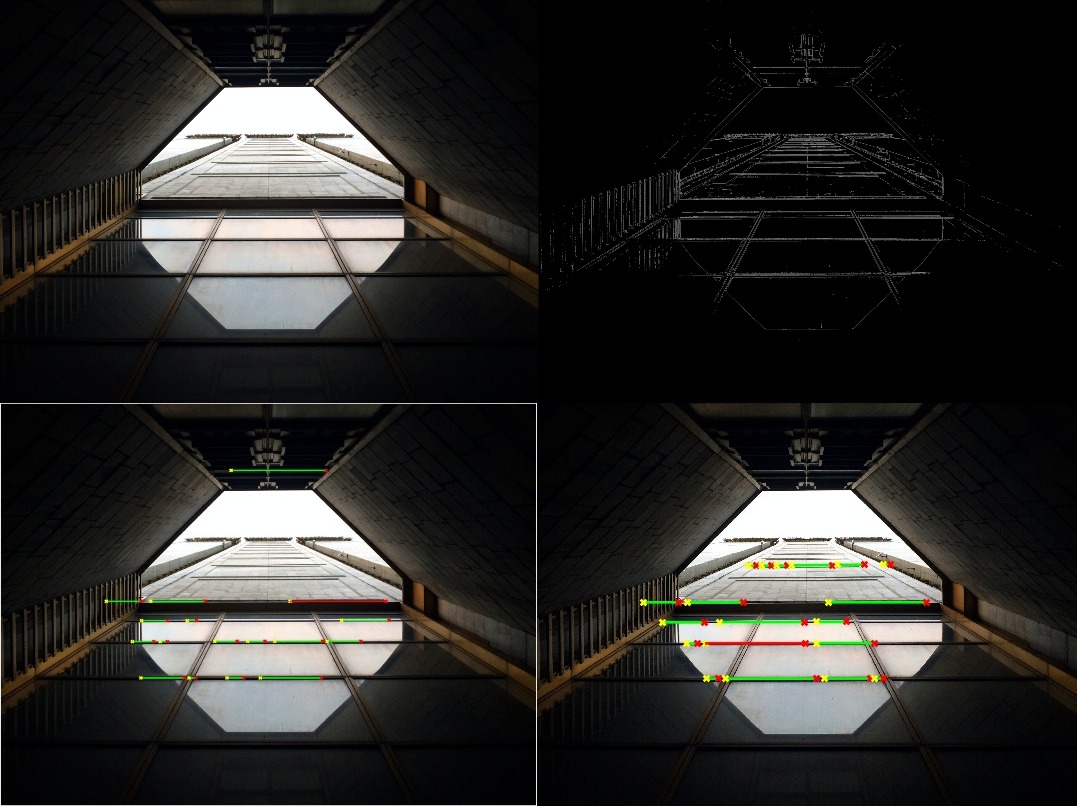


Figure 3

A picture containing text, outdoor, transport

Description automatically generated

Figure 4

In general, my implementation demonstrates good detection of boundaries, and has only small discrepancies with Matlab’s built-in function. The boundaries detected fits with our general conception of the picture.

However, neither of them annotates some other obvious boundaries. For instance, the tree in the second picture. There may be two reasons for this. The first reason is the resolution of picture. In a high-resolution picture, a boundary of image is wider than that in a low-resolution one. The number of pixels in a boundary line is not only determined by its length, but also wideness. Therefore, a short but bold boundary can get more votes than a long, thin boundary.

The second reason is the noise pixels, especially in some pictures with colored background. Many pixels in the background are classified as edge pixel. These misclassified pixels can pollute the accumulator because they can give vote to lines of all distances and angles. I have tried to filter the noise pixels but have not come up with a good solution so far.

Considering these possible reasons, another trial is done with a low-resolution, simple image. The result is shown in Figure 5.

A picture containing shoji, green

Description automatically generated

Figure 5

In this trial, all boundaries are detected and well-annotated. So, the program works well with small and simple image.

2. Running Time Analysis

Chart, scatter chart

Description automatically generatedThis program was run on both GPU and CPU, Figure 6 demonstrates the difference in runtime.

Figure 6

Generally, runtime on GPU is about 70% of runtime on CPU. The main speed up barrier is that a bin in the accumulator array can only be updated atomically to avoid race condition. Therefore, the major speedup occurs in parallel processing of each pixel and parallel computing of distance.

# Deliverables:

1. Main file:

hough.cpp

2. Input files:

‘input’ folder: six original high-resolution images I found on Pexels.

‘output’ folder: edge map for each image.

‘size’ folder: files that has information of image sizes.

3. Output files:

‘vote’ folder: accumulator matrix for each image.

‘img\_withline’ folder: annotated images generated by displayimg.m

4. To run the code:

g++ hough.cpp -Wall -O3 -std=c++17 -o hough -fopenmp

./hough n

Where n is the image number.

OR on Euler

sbatch hough.sh

5. To view the annotated image: run displayimg(n) in Matlab, where n is the image number.

# Conclusions and Future Work

To detect object boundaries in a high-resolution image, I used Hough Transformation algorithm and parallel computing. The program was able to detect major boundaries and worked well for small and simple pictures. However, due to the nature of high-resolution pictures, Hough Transformation may not be the best way for boundary detection. The runtime of this program was shortened by 30% with parallel computing. The barrier for getting a shorter runtime is the unavoidable atomic update in accumulator array.

There are still many problems in this program that need future work. The first problem is whether Hough Transformation is suitable for large images. As we can see, this algorithm worked well with small image. However, from the result, neither my implementation nor the built-in packages gave a perfect detection of boundaries for high-resolution images. The second problem is how to optimize Hough Transform algorithm for high-resolution images if we are still going to use this method for boundary detection.

# References

[1] Gao, Wenshuo, Xiaoguang Zhang, Lei Yang, and Huizhong Liu. "An improved Sobel edge detection." In 2010 3rd International conference on computer science and information technology, vol. 5, pp. 67-71. IEEE, 2010.

[2] Illingworth, John, and Josef Kittler. "A survey of the Hough transform." Computer vision, graphics, and image processing 44, no. 1 (1988): 87-116.