CUDA GPU Parallel Programming

A. Overview:

In the modern world, edge detection has been playing an increasingly important role in people's day to day life in fields such as machine learning and computer vision. Among a variety of techniques and methods for edge detection, although the Sobel Edge Detection algorithm is one of the most commonly used, the Canny Edge Detection algorithm has often been considered to have better results. Specifically, the Canny algorithm is often implemented using the output of the Sobel Algorithm as input and performs further processing to it and therefore takes longer to perform.

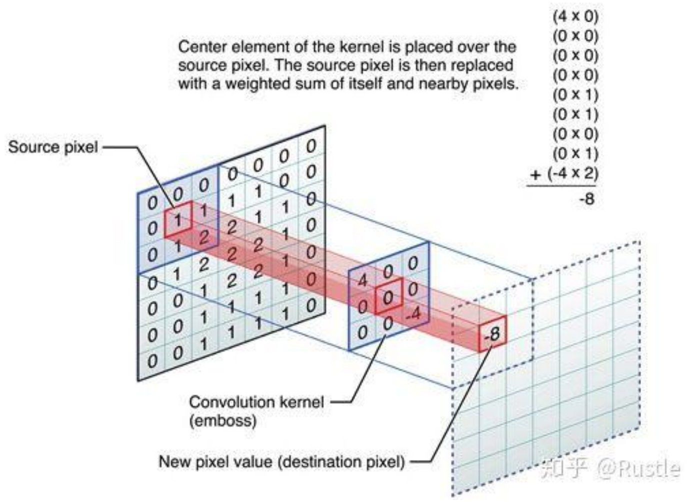
In this project, both methods and algorithms are implemented and benchmark with 12 image with various size, resolution, complexity and style (art or image) to compare the visual result of these edge detection, the execution time of each methods over each image, the speedup sacrifice of using the Canny method over the Sobel method, and how each method perform on different image types.

B. Technical Description/Methods:

a. The Sobel Edge Detection Algorithm:

First we import the concept of filter, which could be used to simplify the information in the image for subsequent image processing by extracting image features and Eliminate the noise mixed in when the image is digitized.

Filtering can be understood as the filter (usually 3*3, 5*5 matrix) traverses the image from top to bottom, from left to right, calculates the value of the filter and the corresponding pixel, and performs numerical calculation according to the filtering purpose. Return Value to current pixel.


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)
知乎 @Rustle

Here we have sobel filter which respectively represent the edge detection operator for X axis and Y axis:

$$s_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, s_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Two filtering operations are performed along the X axis and Y axis, and the result obtained is squared and added with the root sign to obtain the image gradient of the current pixel.

Gaussian filtering is currently the most popular denoising filtering algorithm, the principle is to perform a weighted average according to the gray value of the pixel to be filtered and its neighboring points according to the parameter rule generated by the Gaussian formula, which can effectively filter the high-frequency noise superimposed in the ideal image.

Common Gaussian filters are:

$$\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \frac{1}{273} \times \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

The Gaussian filter is very similar to a pyramid structure. The value of the filter can be understood as a weight. The larger the value, the greater the weight of the pixel, and the larger the component. Therefore, we can see from the Gaussian filter Corresponding to the current pixel, the further the distance, the smaller the weight, and the smaller the contribution to the gray value.

b. The Canny Edge Detection Algorithm:

Upon receiving the result of the Sobel Edge Algorithm, the Canny Edge Detection applies a few more steps to optimize the result produced by the Sobel method. Specifically, these methods include:

1. Non Maximum Suppression
2. Double Threshold

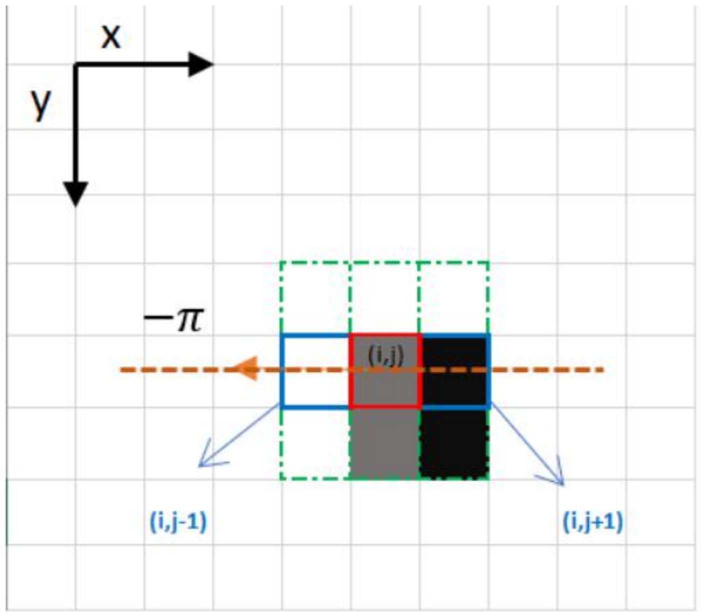
For simplicity, these methods are implemented in the same CUDA kernel of the sobel method.

i. Non Maximum Suppression:

The idea of Non Maximum Suppression is to reduce the thickness of the result of the Sobel output, and remove redundant edge lines to minimize the gradient translation of the pixel values and get sharper lines.

To do so, first the angle/direction of each pixel/edge from the output of the Sobel method was calculated. To do so, the formula *angle = atan2((Gx/Gy)*3.1415926)*180 was utilized*

Then, for each pixel, the Non Max Suppression looks at the two adjacent pixels in the same angle of the current processing pixel. If only one of the two adjacent pixels has a higher pixel value, then the pixel would be set to 0 and not seen as an edge. On the other hand, if both adjacent pixels have higher values, then this pixel is considered as an edge and the value is kept. [see example image below]



As a result, all transnational pixels would be removed.

ii. Double Threshold

After the process of Non Maximum Suppression, we would still see some noise in the image where some lines are created that are lower in pixel value. Therefore, the Double Threshold method aims to not only remove these noise and lower valued pixels, as well as amplify dominant pixels that are edges.

To do so, both a high threshold and a low threshold is set based on the maximum value of the strongest pixel. For example, if the strongest is of value 240 (all pixel value range from 0-255), we can set the high and low threshold to 240 * weight (i.e 0.2 for high, and 0.05 for low). Then, we set all pixel with value higher than the high threshold to 255, and lower than the low threshold to 0. Lastly, we set all pixel in between the low and high threshold to a selected midtone value. As a result, all pixels are distributed to three values : 0, selected mid value, and 255. Then most of the noise should be removed and dominant edges should be amplified.

To Implement this method, the maximum pixel value was first computed on the CPU using OpenCV library and passed to the kernels. Then, in the same kernel with the above methods, each thread computes the current pixel value to the higher and lower threshold and makes adjustments mentioned above.

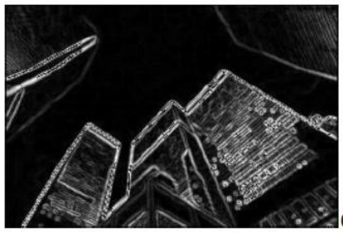
C. Library Used

The **OpenCV Library** was used to translate these jpg images to pixels data as well as displaying the result in the end.

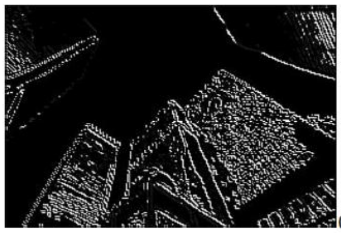
D. Status

The project worked successfully on all various image sizes that we tested it on. As a limitation of this project, we noticed that any of the image results of the Canny Edge Detection Algorithm is sometimes restricted by the performance of the output Sobel Edge Detection. For example, in the image below, the second image uses the Sobel Edge Detection Algorithm while the third method is the result of the Canny Edge Detection Algorithm using the output of the Sobel method as input. Due to the smaller resolution of the image, it is clear that the Sobel method had problems identifying the small/complex edges on the top section of the building and producing blur as a result. This issue was therefore also carried over to the Canny method as a result. Therefore, we wish to have also used a few other edge calculation methods as the input of the Canny method as comparison.

Image 10



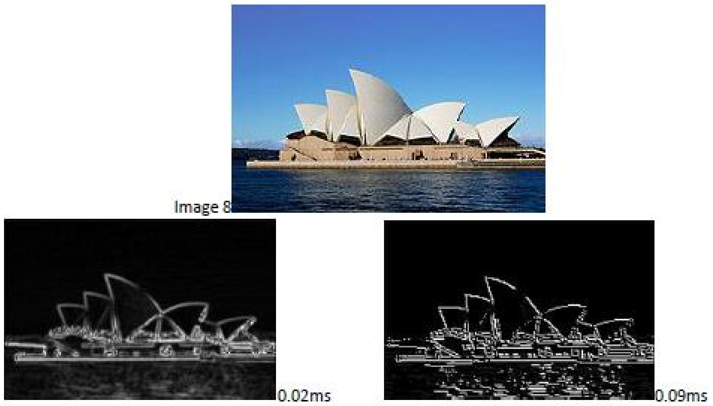
0.02ms



0.12ms

E. Results/Evaluations

For all the image output and timing of each input, please refer to the "Result.pdf" file in our submission. [example image below]



In the result report, the first edge detection image is the Sobel Method, while the second image is the Canny Method result. Through analysing the results, there were a few observations that stood out as listed below.

a. The Timing of Sobel Method and Canny Method

Image #	1	2	3	4	5	6	7	8	9	10	11	12
Sobel	0.08ms	0.15ms	0.04ms	0.03ms	0.09ms	0.11ms	0.12ms	0.02ms	0.04ms	0.02ms	0.61ms	0.68ms
Canny	0.39ms	2.08ms	0.18ms	0.09ms	0.42ms	1.15ms	0.69ms	0.09ms	0.13ms	0.12ms	3.56ms	5.49ms
Speedup (Canny/Sobel)	4.87	13.86	4.5	3	4.67	10.5	5.75	4.5	3.25	6	5.84	8.07

[Note: Speedup here shows how many times is the sobel method is faster than the canny method]

From a timing standpoint, the above results show that the Canny Edge Detection method takes on average 6.23 times more execution time than the Sobel Method. With a larger number of inputs, this difference can be significant. In addition, as edge detection is often used in real time computer vision and machine learning, which makes lower execution time even more important.

b. Performance Comparison of Sobel vs. Canny with various Input

In the images which have simple lines and colors and don't have many details, Sobel Method has similar effect with Canny method or even more clear, at the same time Sobel Method needs less time:

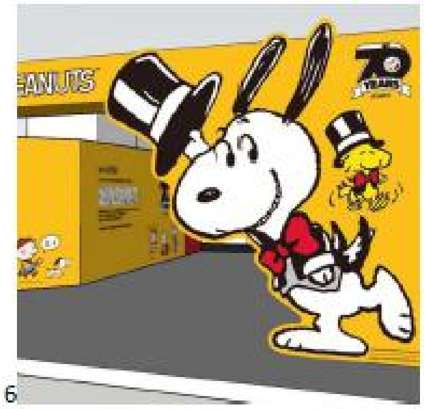
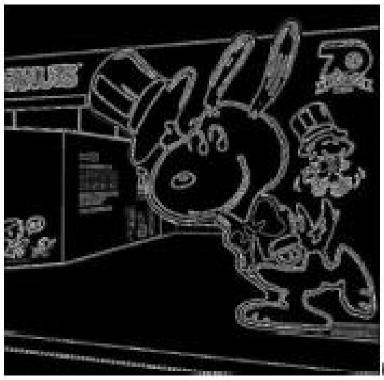
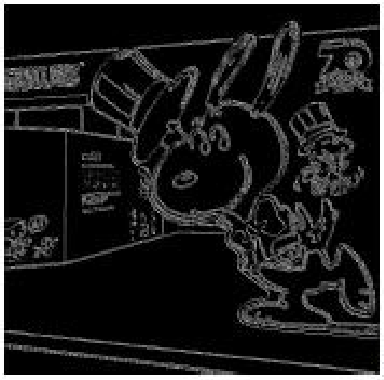


Image 6



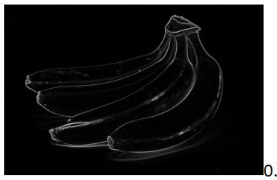
0.11ms



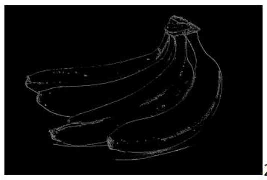
1.

In the image 6 example above, due to a lack of transitioning colors and values, the sobel method seems to have a much better performance and more clear edges which canny method produces a more choppy line. In addition, the Canny method took more than 10 times more execution time to complete. Same can be seen in image 2 in the report.pdf document[as well as screenshot below].

Image 2

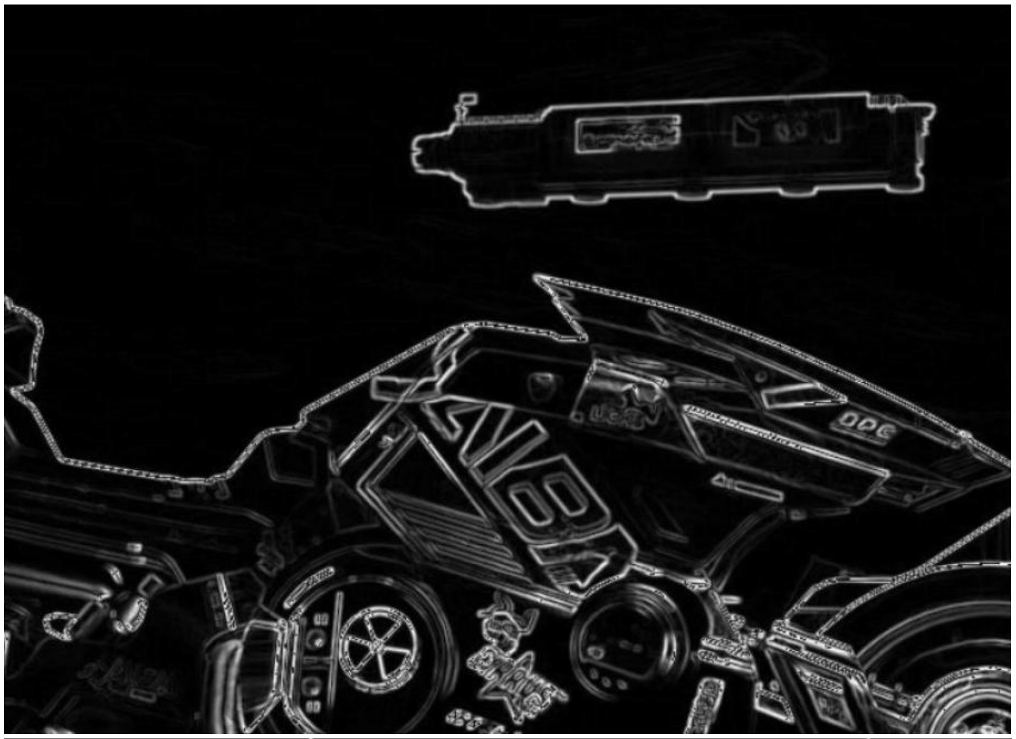


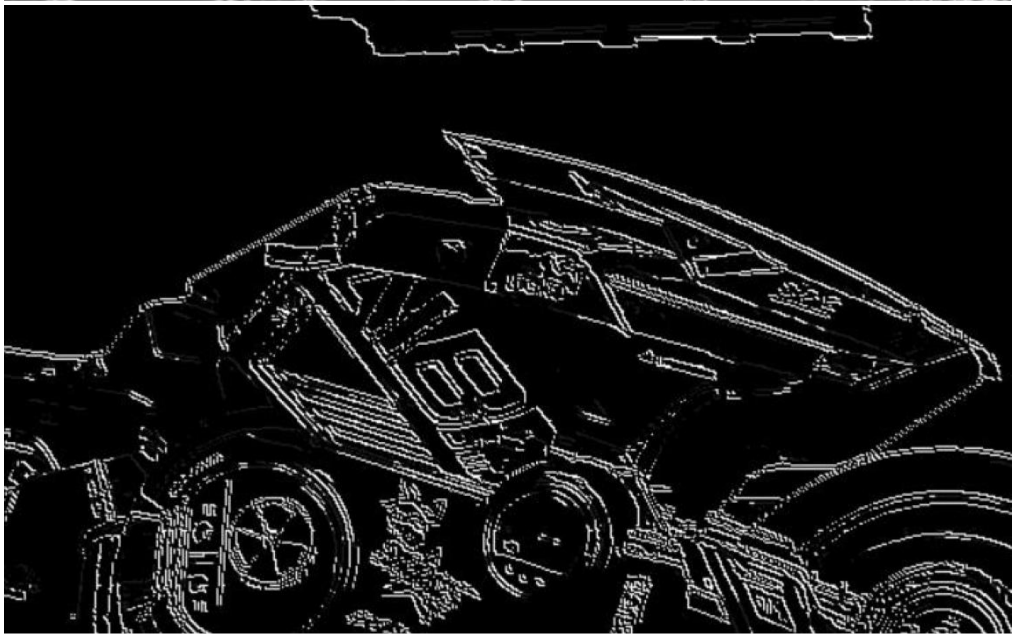
0.15ms



2.08ms

On the other hand, In the images which have many value transition of lines and colors, Canny Method has a better performance removing the none edge pixels and separating the lines as shown below

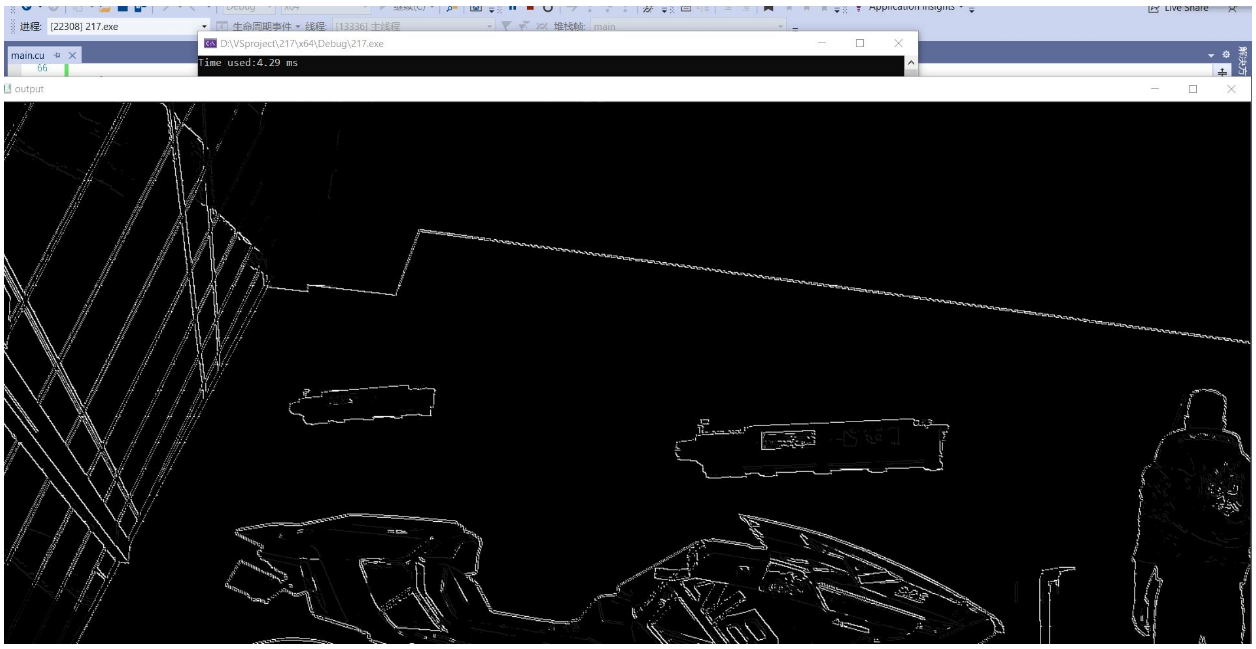




F. Compile and Run

Required Library/ Technology: OpenCV, CUDA

For convenience of finding testing images, no parameter input is needed when testing the application. When the program is compiled and executed, it runs a sample image named 12.jpg that is located in the same folder and displays the final result of the application on the screen. Then, the result should look like the screenshot shown below where both the final image and the timing would appear.



```
D:\VSproject\217\x64\Debug\217.exe
Time used:4.29 ms
```