**Lab: High Performance Computing with Graphics Cards**

**Project Report: Canny Edge Detection**

Muhammad Zamik Shahid (3507157)

Usman Sikander Mirza (3507018)

Mohammed Asjadulla (3578966)

**Introduction**

The Canny Edge Detector is a popular edge detection operator that uses a sequence of several algorithms to detect different types of edges in images. The process of Canny edge detection consists of the following five steps and we discuss each of them in detail on the next page:

1. Removal of noise by application of Gaussian filter to smoothen the image
2. Finding the intensity gradients of the image
3. Applying non-maximum suppression for removing spurious responses to edge detection
4. Applying double threshold to determine potential edges
5. Tracking the edges using hysteresis

As part of the project, we ran the edge detection algorithm on CPUs as well as GPUs and noted the execution times for each of them respectively. We then calculated the speedup, which is the principal parameter here, to determine the performance improvements of using a GPU over a CPU.

**GPU Settings**

| GPUs tested on | NVIDIA GeForce GTX 1060 and 1070 |
| --- | --- |
| Number of elements in each group | 16\*16 = 256 |
| Total number of threads | (16\*40) \* (16\*30) = 307,200 |
| Total memory (in bytes) | Total number of threads \* 4 = 1,228,800 |

**Steps in the Canny Edge Detection algorithm:**

1. **Gaussian Filter**

* The first step is to remove noise from the images as edge detection results are easily altered by the noise in the image.
* We apply a Gaussian filter which blurs the image and suppresses the noise by smoothening it.
* The key variables in the equation of a Gaussian filter are sigma and the kernel size.
* The kernel size is of the form (2k+1)x(2k+1) where *k = 1,2,3,......* and can be modified according to the amount of blurring required. For our project we use a *3x3* Gaussian filter.

1. **Calculation of gradient**

* After the noise in the image has been reduced, we determine the intensity and direction of an edge by calculating the gradient of the image using an edge detection operator.
* We use a Sobel operator for our project, which is an edge detection operator that works in both horizontal and vertical directions.
* Sobel operator has two filters ***Kx***and ***Ky***which are convolved with the image ***A*** to obtain ***Ax*** and ***Ay***  and produces an intermediate image with varying edge thickness.

1. **Edge Thinning**

* In this technique all the points in the gradient intensity matrix are processed and the intensity of the current pixel is compared with the intensity of the neighboring pixels having either positive or negative gradient.
* If the intensity of the current pixel is the highest (255 in our case) amongst all its neighbors, the intensity value will be retained, otherwise it will be *suppressed* or in other words set to 0.
* If no neighbor is found to have a greater intensity value than the current pixel then the intensity value of the current pixel remains unchanged.
* Non-maximum suppression produces an image with thinner edges as compared to the edges found in the image obtained after applying a Sobel filter.

1. **Double threshold**

* The thin edges in the image obtained after step 3 have varying brightness.

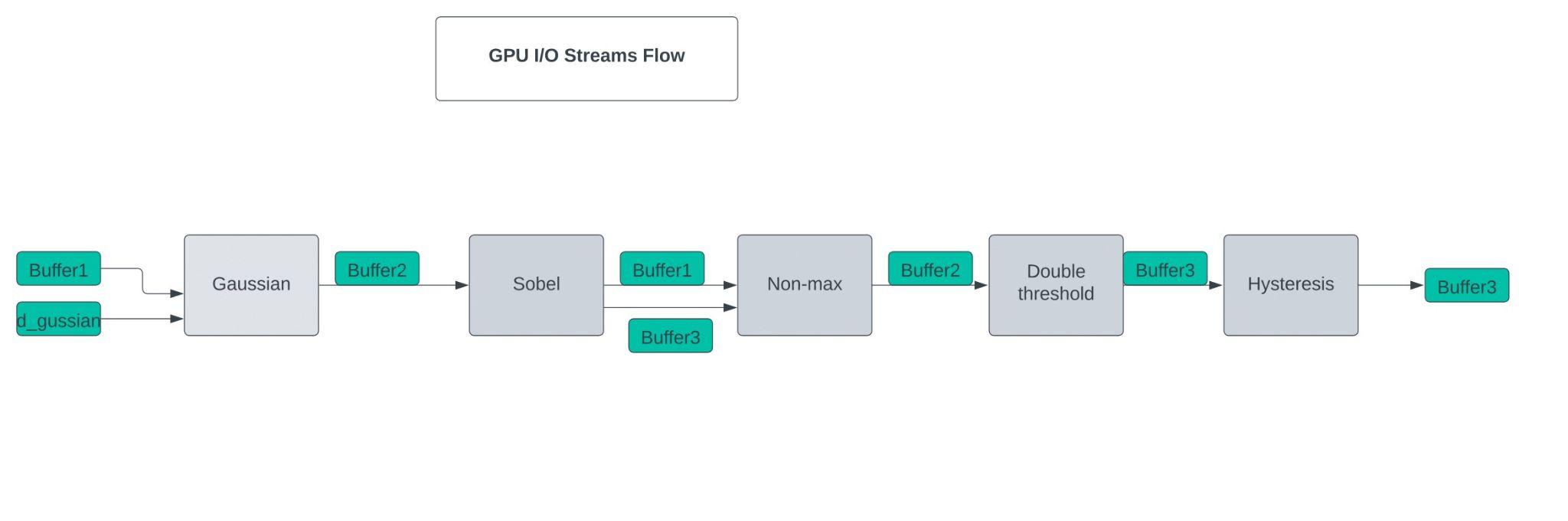
We apply the double threshold technique to obtain uniformity in their brightness. This technique uses a high and low threshold to identify strong, weak, and non-relevant pixels.

* Strong pixels are those pixels having a gradient value higher than the high threshold value. Weak pixel’s gradient value is lower than the high threshold but higher than the low threshold value. Non-relevant pixels are those pixels having a gradient value lower than the low threshold value.
* The values of the thresholds are determined heuristically and depend on the image that is fed to the algorithm.

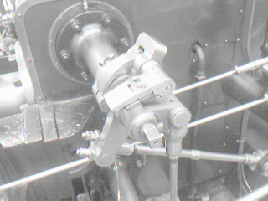
1. **Edge Tracking by Hysteresis**

* The previous step produces an image with edges having two types of pixels: weak and strong.
* Strong and weak pixels in an image may include object edges which are irrelevant. So, it is necessary to convert the weak pixels into strong pixels wherever possible.
* Conversion of weak into strong pixels takes place if the pixel being processed currently has a neighbor (in both horizontal and vertical axis) which has an intensity value greater than the current pixel’s intensity value.
* If a higher intensity is found in any of the neighboring pixels, the current pixel’s intensity value is changed to that neighboring pixel’s intensity value.
* This converts the weak pixel having intensity value less than 255 to a strong pixel having intensity value of 255.
* This final step produces an image having distinct bright edges of the same intensity.

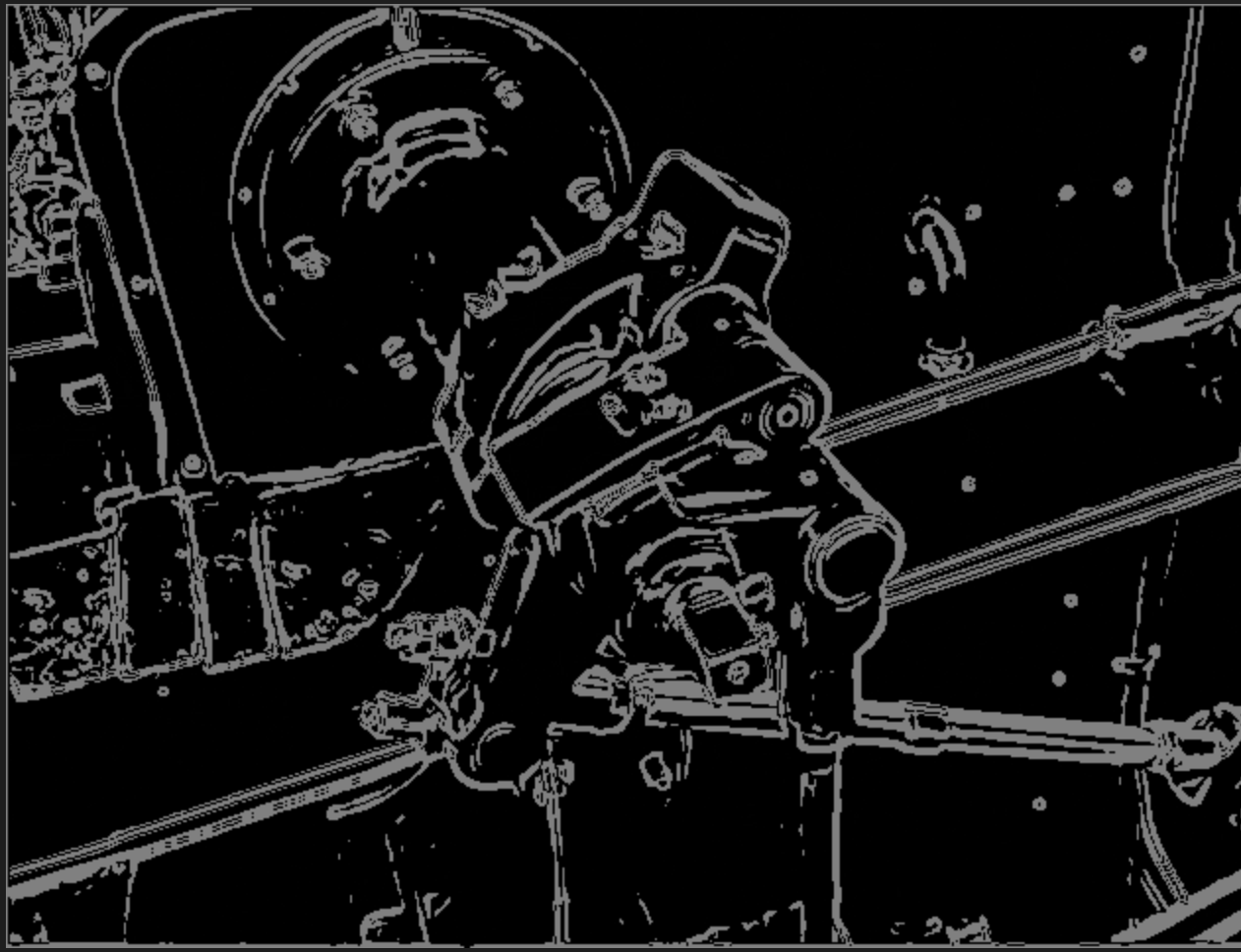
**GPU I/O Flow Diagram:**

****

**Input Image**

****

**Output Image (Step5)**



**Work Distribution**

| **Task** | **Asjad** | **Zamik** | **Usman** |
| --- | --- | --- | --- |
| Canny CPU Code conversion from python to c++ |  |  |  |
| Gaussian Filter |  |  |  |
| Calculation of gradient |  |  |  |
| Non-maximum suppression |  |  |  |
| Double Threshold |  |  |  |
| Edge Tracking by hysteresis |  |  |  |
| Report |  |  |  |

**References**

1. <https://en.wikipedia.org/wiki/Canny_edge_detector>
2. <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>
3. <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>