KHENE SORAYA

202031075992

CUDA PROJECT REPORT

Sobel Filter

Scope :

A **filter** in image processing is an operation applied to an image to extract specific features. One common use of filters is edge detection, which highlights the boundaries of objects within an image by identifying regions with a sudden change in intensity (brightness).

Among the many filters used for edge detection, the **Sobel filter** is widely recognized

How does it work :

Inputs:

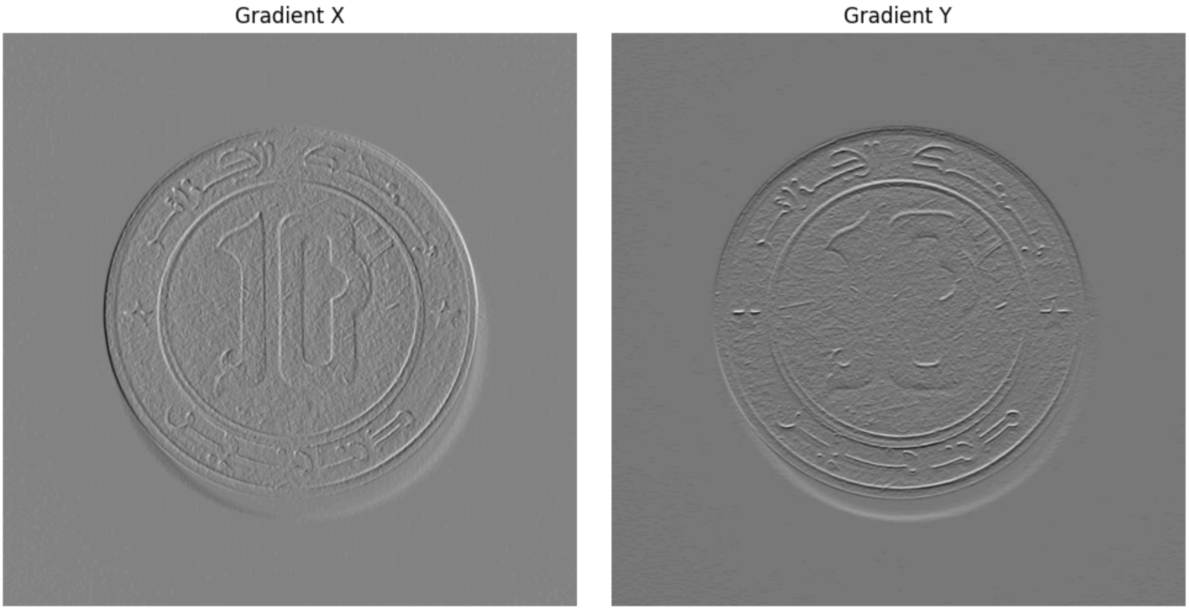
1. **Grayscale Image**: An input image of dimensions x×y, represented in grayscale.
2. **Kernels**: Both are usually 3×3 matrices.
   - **Sobel_x_kernel**: Detects vertical edges.
   - **Sobel_y_kernel**: Detects horizontal edges.

Convolution:

Perform a **convolution operation** between the image and each kernel:

1. Convolution of the image with sobel_x_kernel : this is performed to detect edges in horizontal direction resulting Horizontal gradient image G_x

2. Convolution of the image with sobel_y_kernel : this is performed to detect edges in vertical direction resulting Vertical gradient image Gy



Gradient X

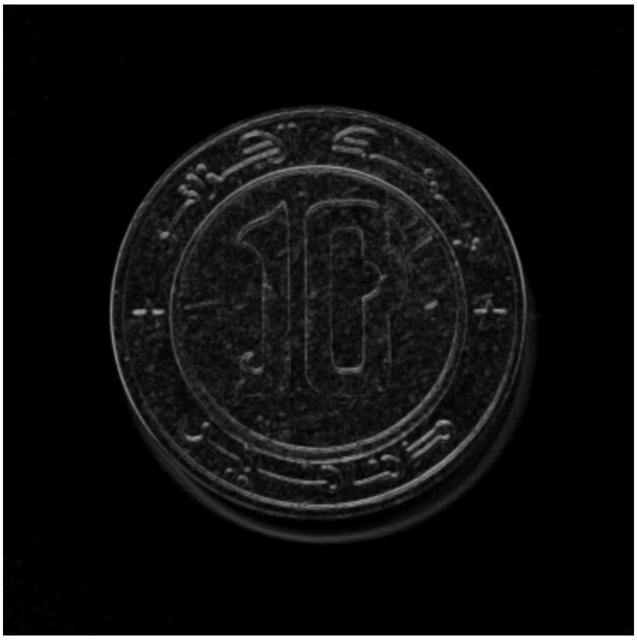
Gradient Y

Gradient Magnitude Calculation: For each pixel, calculate the gradient magnitude G using the formula: $\sqrt{Gx^2 + Gy^2}$

Thresholding (optional): Compute the **threshold** (e.g., mean or median of G) to decide the edge strength. For each pixel in G:

- If pixel value > threshold, set pixel value to **255** (white, edge).
- Otherwise, set the pixel value to **0** (black, no edge).

Output: The final output is a binary image highlighting the edges detected by the Sobel filter.



Dataset :

I’ve created a dataset containing 12 images of different objects, including glass, flowers, apples, coins…… The images have varying sizes, with the largest dimension being 1280x1280 pixels and smallest 612x612

Sobel Filter on CPU :

I’ve implemented the classical steps using python. In this approach, the Sobel filter is applied sequentially with a for-loop. It processes one pixel at a time.

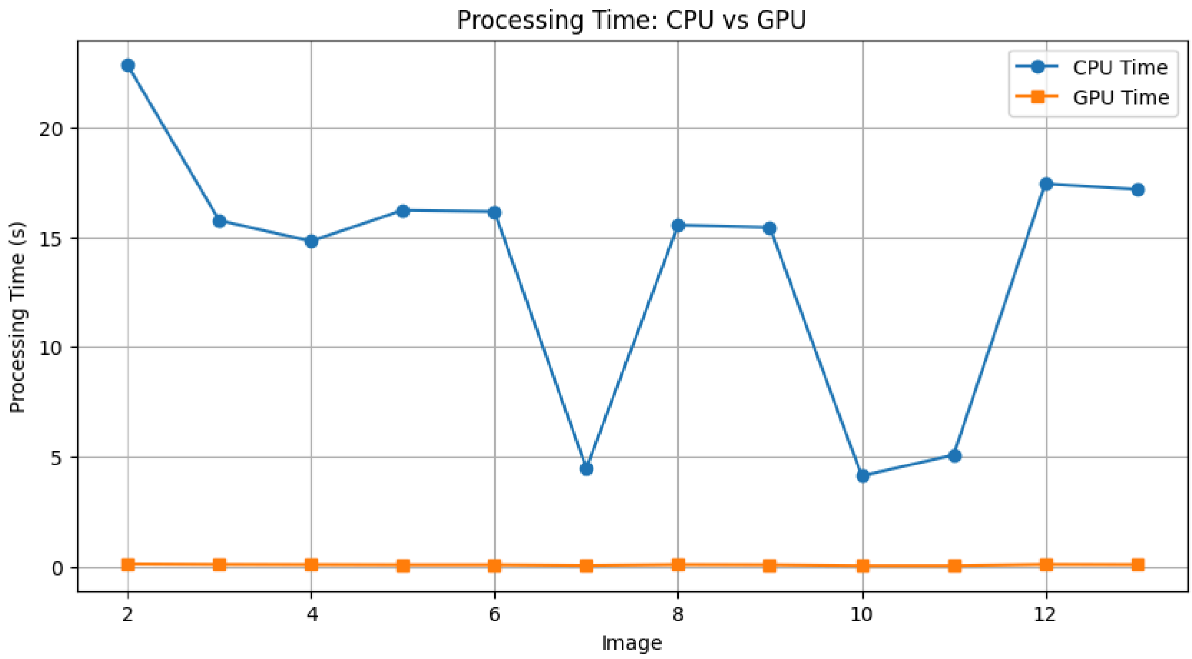
Sobel Filter on GPU :

To achieve parallelism, the image is divided into **blocks of threads**, and each thread is responsible for processing a single pixel, ensuring simultaneous computation of edge detection across the entire image.

Workflow

1. **Initialization**: The host function loads the input image and allocates memory on both the CPU and GPU.
2. **Data Transfer**: The input image data is copied from the CPU memory to the GPU memory.
3. **Kernel Launch**: The Sobel filter kernel is launched, with the grid and block dimensions specified to cover all pixels in the image.
4. **Parallel Processing**: Each thread on the GPU executes the kernel function, calculating the edge magnitude for its assigned pixel.
5. **Synchronization**: The CPU waits for the GPU to finish processing before continuing.
6. **Data Retrieval**: The processed image data, containing the edge magnitudes, is copied from the GPU memory back to the CPU memory.
7. **Output**: The host function saves the processed image to a file.

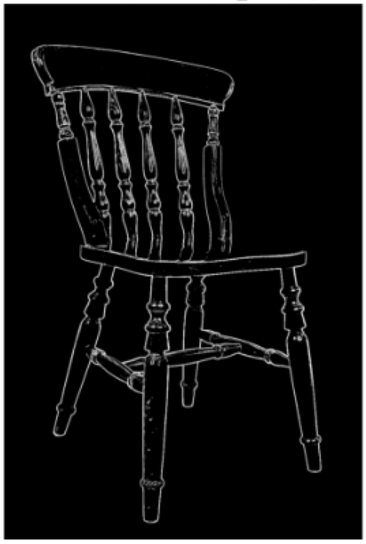
CPU vs GPU :



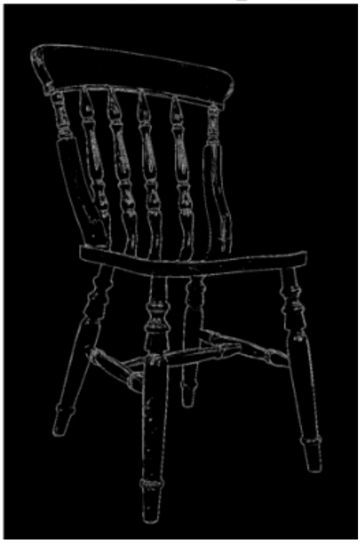
- The **CPU** processing times are consistently **higher** compared to the GPU
- The **GPU** processing time remains **nearly constant** across all images, demonstrating its ability to handle tasks efficiently regardless of image complexity.
- The **GPU's** performance indicates **better scalability** for large datasets compared to the CPU.
- If the **images differ in size**, this could account for the **variation in CPU processing times**.
- We notice that images generated by GPU are **sharper** than those generated by CPU

Examples :

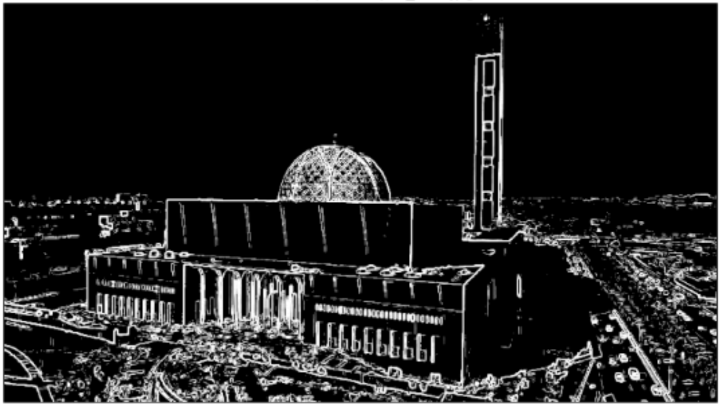
GPU Output: output_im9.jpg



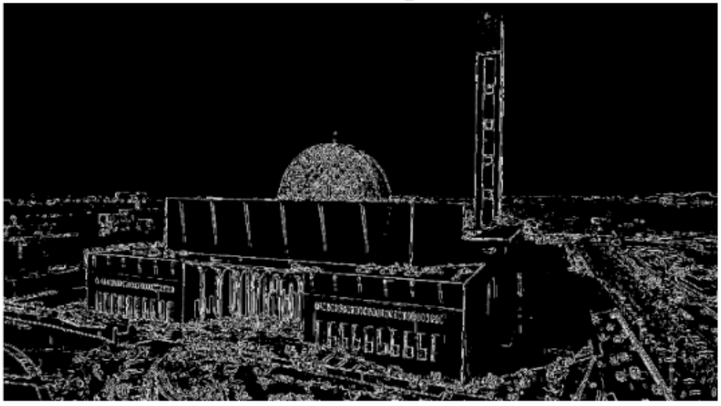
CPU Output: output_im9.jpg



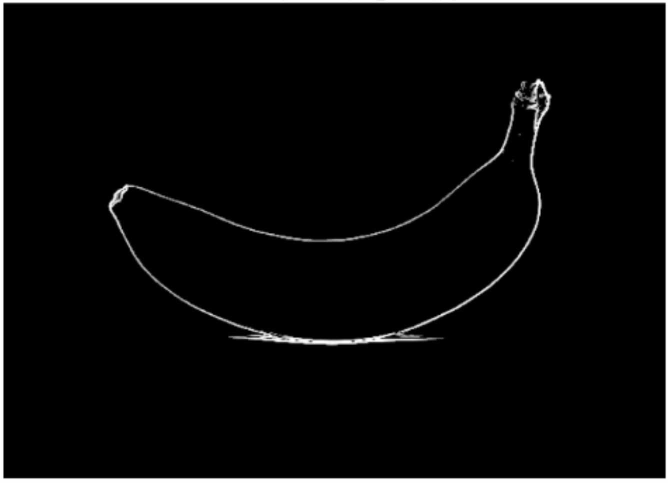
GPU Output: output_im4.jpg



CPU Output: output_im4.jpg



GPU Output: output_im10.jpg



CPU Output: output_im10.jpg