Labwork 10: Gather and Scatter
Fine-art Transformation

Pham Gia Phuc

November 2024

1 Subject

This lab aims to implement the Kuwahara filter in CUDA, both with and without shared memory.

2 Implementation

2.1 Preparation

This report uses a CUDA kernel provided by Google Colaboratory.

Attribute	**Value**
Number of CUDA Devices Found	1
Device ID	0
Name	Tesla T4
Compute Capability	7.5
PCI Device ID	4
PCI Bus ID	0
UUID	GPU-af936f72-170a-716a-326e-6053e93d8f54
Watchdog	Disabled
FP32/FP64 Performance Ratio	32
Multiprocessor Count	40
Approximate Core Count	2560
Total Memory Size	14.75 GB
Environment	Google Colab

Table 1: CUDA Device Information

2.2 Kuwahara Kernel

The Kuwahara filter is a non-linear smoothing filter used in image processing for adaptive noise reduction. Unlike typical linear low-pass filters, which reduce

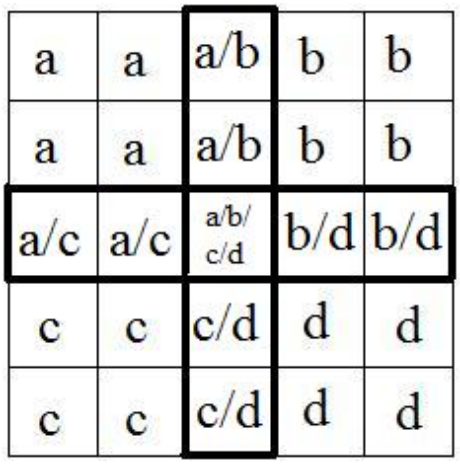

Figure 1: Window used by a Kuwahara filter. It is divided into 4 square regions: a, b, c, and d. Pixels in the central row and column belong to multiple regions.

noise but blur edges, the Kuwahara filter smooths an image while preserving edges. It divides a window into four square regions (a, b, c, and d), as illustrated in Figure 1, with overlapping central rows and columns.
The Kuwahara filter serves the following purposes:

- Noise reduction
- Edge preservation
- Creation of an "oil painting" effect

3 Results

3.1 Sample Image and Filter Application

A sample image, as shown in Figure 2, was processed using the Kuwahara filter kernel to achieve an oil painting effect.

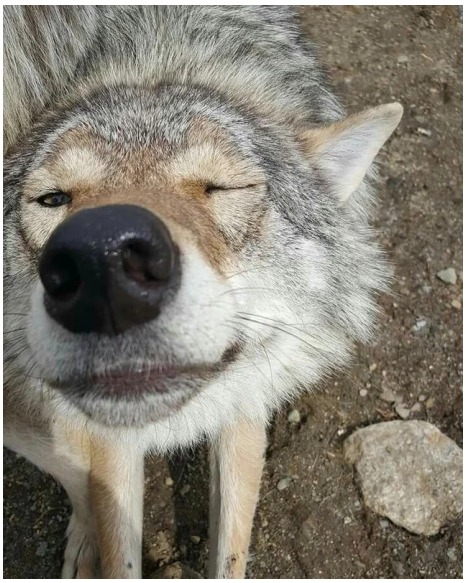

Figure 2: A sample image

The filter was applied to various image resolutions, yielding the results in Figures 3 and 4. These images illustrate the filter's effectiveness in producing an oil effect while preserving edges.

3.2 Performance Analysis

The performance of the Kuwahara kernel was analyzed on images of different resolutions to understand the effect of using shared memory in a CUDA environment.
For a lower-resolution image (640x800), using shared memory resulted in a marginally higher runtime compared to not using it, as shown in Table 2.

Using Shared Memory	**Resolution**	**Time (seconds)**
Yes	640x800	0.05
No	640x800	0.04

Table 2: Runtime comparison on a 640x800 image

For a higher-resolution image (9000x5625), shared memory usage reduced runtime slightly, as shown in Table 3.

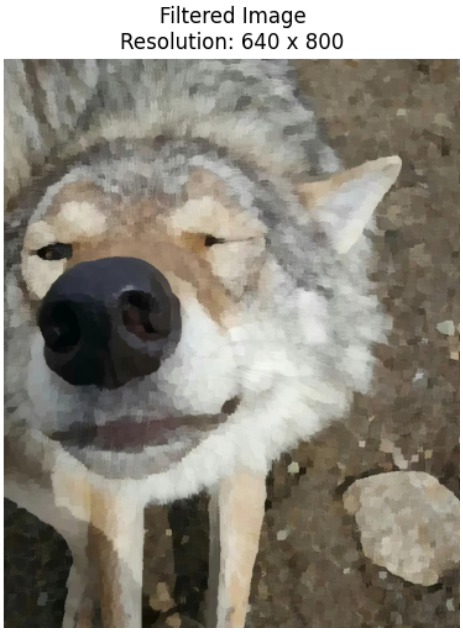

Figure 3: Filtered result with a 640x800 pixel image

Figure 4: Filtered result with a 9000x5625 pixel image

Using Shared Memory	Resolution	Time (seconds)
Yes	9000x5625	2.46
No	9000x5625	2.48

Table 3: Runtime comparison on a 9000x5625 image

4 Conclusion

Using shared memory in CUDA can boost program performance, particularly with large images that require significant computation. However, for smaller images, the overhead from shared memory management can outweigh its benefits. Consequently, it is most effective to bypass shared memory for small images while leveraging it for larger images to enhance performance in Kuwahara filter applications.