Kuwahara Filter GPU Optimized

Antoine Maffeis
USTH - Hanoi, Vietnam

January 2023

Abstract

In this paper, we present a Kuwahara filter implementation using the Numba CUDA GPU library for python. The Kuwahara filter is a popular image processing technique used to smooth out noise and preserve edges in images. It is particularly useful for low-light or noisy images.

1 Introduction

The Kuwahara filter is a non-linear image processing technique that was developed by Nobuyuki Kuwahara in 1978. It is used to smooth out noise and preserve edges in images. The Kuwahara filter is particularly useful for low-light or noisy images, as it helps to improve the overall quality and clarity of the image. However, the filter can be computationally expensive, especially for large images. In this report, we present a Kuwahara filter implementation and the Numba CUDA GPU library. The use of a GPU allows for significantly faster execution times compared to a CPU implementation.

2 Methodology

We implemented the Kuwahara filter using the Numba CUDA GPU library. The Numba library allows us to write code in Python and then compile it to run on a GPU.

Our implementation follows one of the several approach of the Kuwahara filter. It works by first converting the RGB image to HSV. Then dividing the image HSV into a grid of smaller windows and calculating the minimum standard deviation of each window for the "V" dimension only. The grid and the windows are represented in the figure 1. The "V" in "HSV" stands for "value". In the HSV color model, the value channel represents the brightness or intensity of a color. The value channel ranges from 0 (darkest) to 100 (brightest). The standard deviation of all the windows of a grid are calculated and the indexes of the elements of the window having the minimum standard deviation are used for the next step of the filter, to define the new R,G, and B values of all the grid. Indeed, the mean of the sum of the R,G and B pixel values that this window

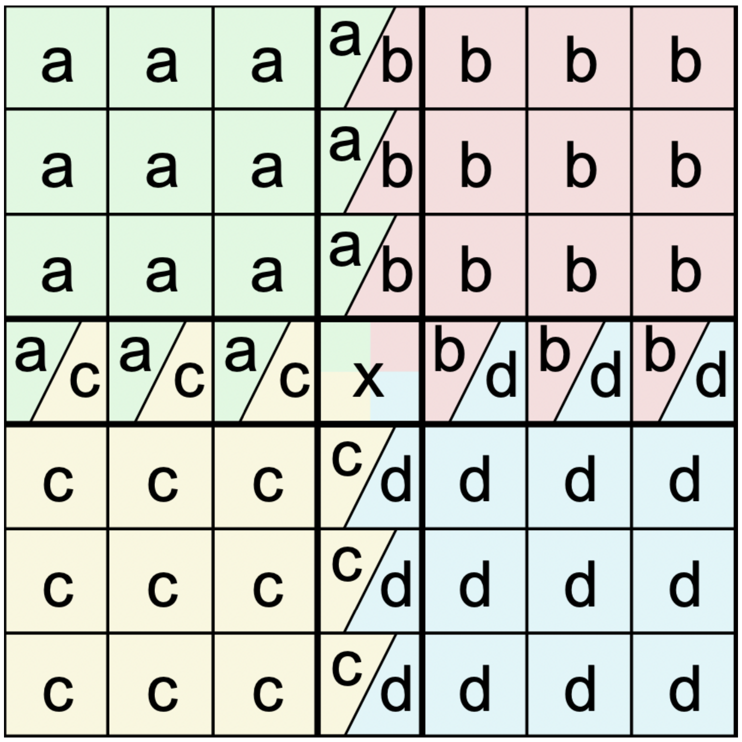
Figure 1: kuwahara filter grid (windows size of 3)

contains is calculated and used as the R,G and B pixel values for the whole grid, for the output image.

3 Results and optimizations

We successfully implemented the Kuwahara filter as figure 2 and 3 display.

We tested our implementation on a variety of images of different sizes. We compared the execution times of our GPU implementation to a CPU implementation.

Our results showed that the use of GPU significantly improved the execution times of the Kuwahara filter. Especially for large images, the GPU implementation was up to 10 times faster than the CPU one. For smaller images, GPU was also several times faster than CPU.

We tried to increase the speed of our GPU implementation. We first only compute the V from the RGB-HSV conversion since only V is necessary. Also, we disposed of some unnecessary computing calculations such as the standard deviation described as follows :

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}} \tag{1}$$

where σ is the standard deviation, μ is the mean of the data, x_i is the ith data point, and n is the total number of data points. We removed the division by the number of elements of the window since it is a constant number linked to the window size (window size square). The previous replacement lead to the following formula :

$$\sigma = \sqrt{\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2}$$

where σ is the transformed standard deviation used, μ is the mean of the data, x_i is the ith data point, and n is the total number of data points. We observed that the disposing of the previous division made the square root also unnecessary. Therefore, it lead to the final formula :

$$\sigma = \sum_{i=1}^{n}(x_i - \mu)^2 \tag{3}$$

where σ is the final transformed standard deviation used, μ is the mean of the data, x_i is the ith data point, and n is the total number of data points. Using our optimizations (better loop placement, formulas modifications), we reduced the execution time by around 20% in average. This percentage tends to increase for larger images.

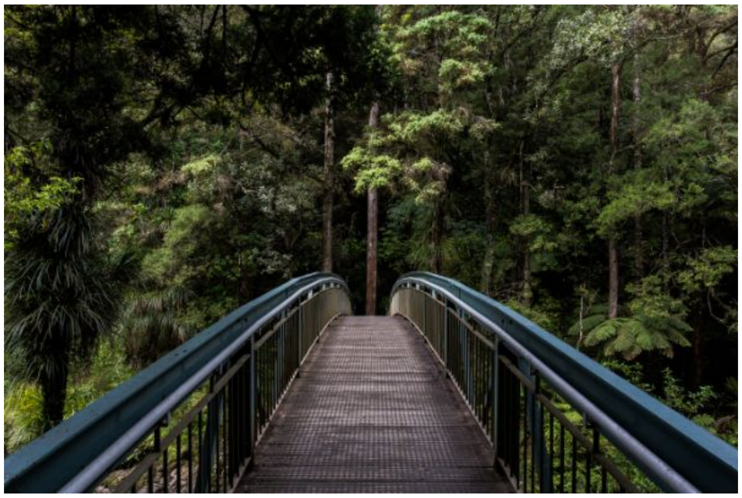

Figure 2: Image of a bridge (no filter)

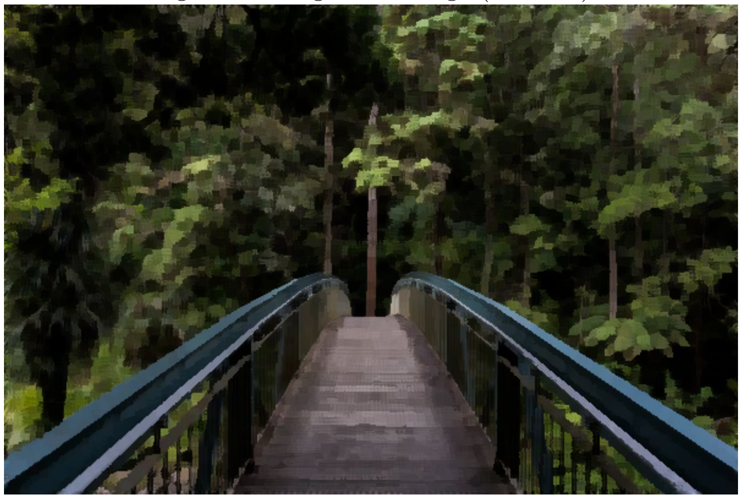

Figure 3: Image of a bridge (our Kuwahara filter applied)

4 Conclusion

In this paper, we presented a Kuwahara filter implementation using the Numba CUDA GPU library. Our implementation significantly improved the execution times of the Kuwahara filter, particularly for large images. The use of Numba CUDA GPU allowed us to take advantage of the parallel processing capabilities of a GPU, resulting in faster execution times, especially for image processing, compared to CPU.