Rank filters : median filter

AUTHORS : *Adrien Rohan

Link : https://github.com/fsoubes/FilterRank

1.Introduction

Since image have been digitized on a computer's memory, it has been possible to interact in new ways with those images that were otherwise impossible. This is called image proccessing, and it consist of methods used to perform operations on a digital image. Those methods are described with various algorithms that can be used for various purposes in multiple fields. Those applications are used for noise filtering and other image enhancement as well as extracting information from images. In this project, will be examined the algorithms used for three types of 2D rank filters : median, min&max and variance.

By definition rank filters are non-linear filters using the local gray-level ordering to compute the filtered value[ˆSoi2002] . The output of the filter is the pixel value selected from a specified position in this ranked list. The ranked list is represented by all the grey values that lies within the window which are sorted, from the smallest to the highest value. For an identical window the pixel value will differ in function of the filters used (median, min, max and variance). Moreover the size of the window is also influencing the output pixel. The median filter is so called because it's an operation which selects the median value.The median filter has been suggested by Tukey[ˆTuk1974]. This filter is widely used for reducing certain type of noise and periodic interference patterns in signal and images without severly degrading the signal[ˆHua1981].The naive algorithm was then improved based on the moving histogram technique[ˆHua1979]. The median filter is used for removing noise, but he main issues of that filter algorithm is his slowness. To overcome these problems the use of small windows and/or low resolution images is required[ˆWei2006].

In this report an implementation of the median filter using WebGL[ˆPar12] and the TIMES module of J.-C. Taveau (https://github.com/crazybiocomputing/times) will be discussed. This report begin by a description of the algorithm that we have implemented in WebGL. Then benchmark analysis will be performed on our implementation and compared with results from the ImageJ and the CPU implementation by J.-C. Taveau of the median filter to compare their performance.

2.Material & Methods

Implementation of the median filter

A median filter is a filter that will compute for each pixel of the starting image the median value of all pixels in the neighborhood of this pixel. This implementation is based on the median filter function of the TIMES module of J.-C. Taveau (https://github.com/crazybiocomputing/times).

A kernel is used to access all the pixel in a neighborhood. A kernel is a small matrix that specify which are in the neighborhood or not of the central pixel in the kernel. They can be of varying size and shape, such

as square or circular. This implementation of the median filter take as argument a kernel as defined by the TIMES module of J.-C. Taveau. It consist of an object that contain for each pixel in a kernel the offset in x-coordinate and the offset in y-coordinate of that pixel compared to the central pixel of the kernel. Thus from the coordinate of a single pixel it is possible to find the coordinate of all the pixel in his neighborhood.

From these coordinate the pixels value can then be accessed, and are then stored in an array. The next step in the median filter is then to sort this array of value. This implementation of the median filter use the OpenGL Shading Language (GLSL) for executing code on the GPU. GLSL doesn't allow the use of recursivity, which mean a sorting algorithm that doesn't use it had to be implemented. The sorting algorithm that has been implemented is the bubble sort algorithm. Here is a pseudo-code :

```
a: an array of sortable element of length N

  swapped = true
  while swapped
    swapped = false
    for j from 0 to N - 1
       if a[j] > a[j + 1]
          swap( a[j], a[j + 1] )
          swapped = true
```

From the sorted array obtained after running the bubble sort, the median is then defined as N/2 element of the array (assuming the kernel contain always contain an odd number of pixels). This whole process is repeated on all the pixels of the starting image to obtain the filtered image.

Benchmarking analysis

Benchmarking analysis is widely used to assess the relative performance of a function[ˆFle1896] by comparing the performance of various algorithms and their implementation. In this rapport, only the relative speed of different analysis will be taken into consideration. The ImageJ and the TIMES modules CPU implementation of the median filter will be compared to our WebGL implementation. The speed of the implementations will be computed for five different sizes of image (180x144, 360x288, 540x432, 720x576 and 900x720) and two different size of kernel (3x3 and 7x7 circular kernel) and three types of images (8-bit, 16-bit and float-32 or (or 32-bit for ImageJ)). For each combination of parameters the function will be run 1000 times (except for the CPU implementation for the 3 larger size of image which are run 100 times).

3.Results

Image results

The Fig. 1 show the results obtained using our median filter implementation compared to the ImageJ implementation for a 360x288 8-bit image and a 5x5 circular kernel. It can be noticed that the ImageJ result are slightly more blurred, and the slight difference between the ImageJ result and our implementation is shown on the 1. D image. The differences are located on the internal border of the image.

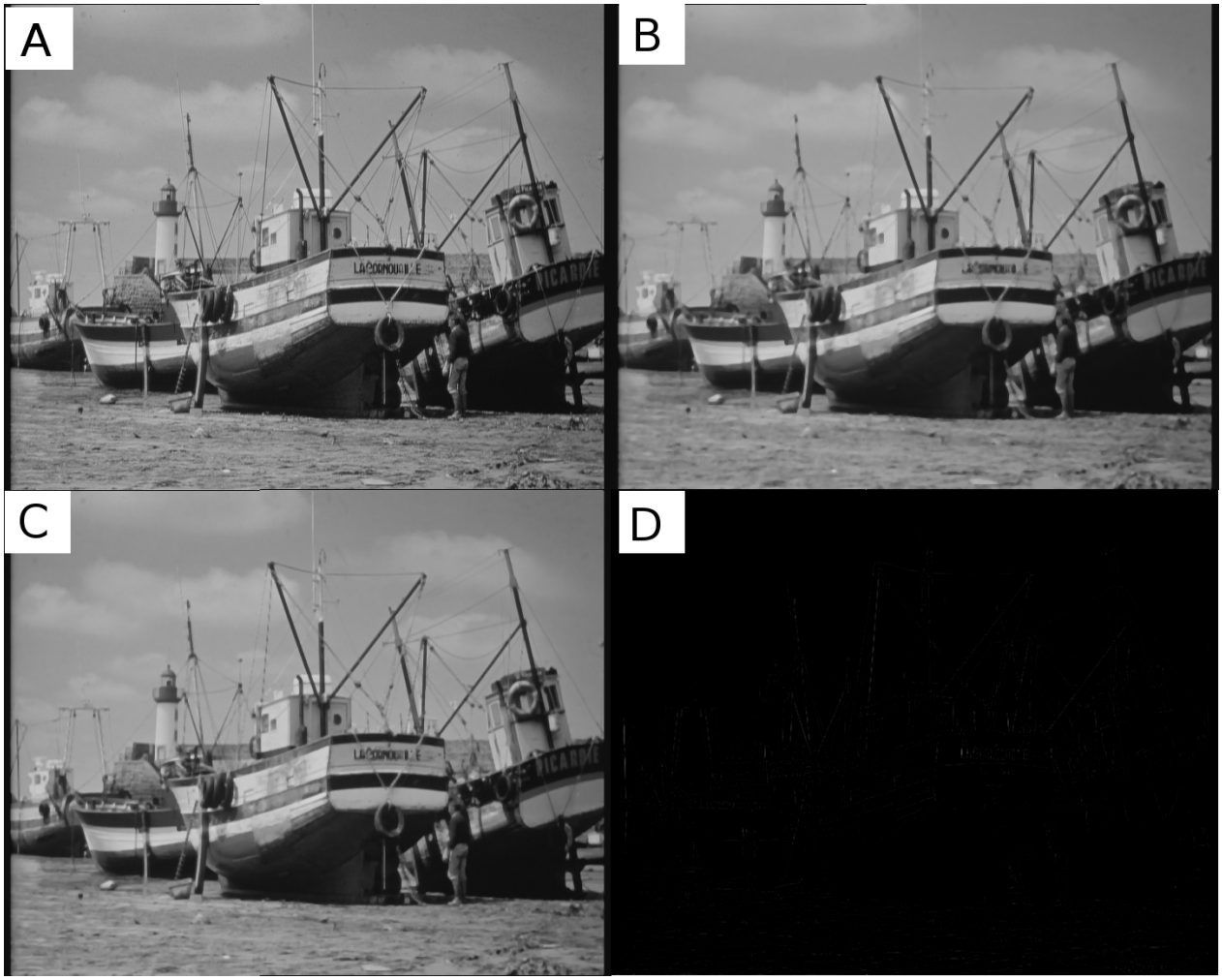

Fig 1. (A) Original image, (B) image obtained using ImageJ median filter (kernel radius equal 2), (C) image obtained using our implementation (5x5 kernel) and (D) difference between B and C.

Benchmark comparison between ImageJ and our implementation

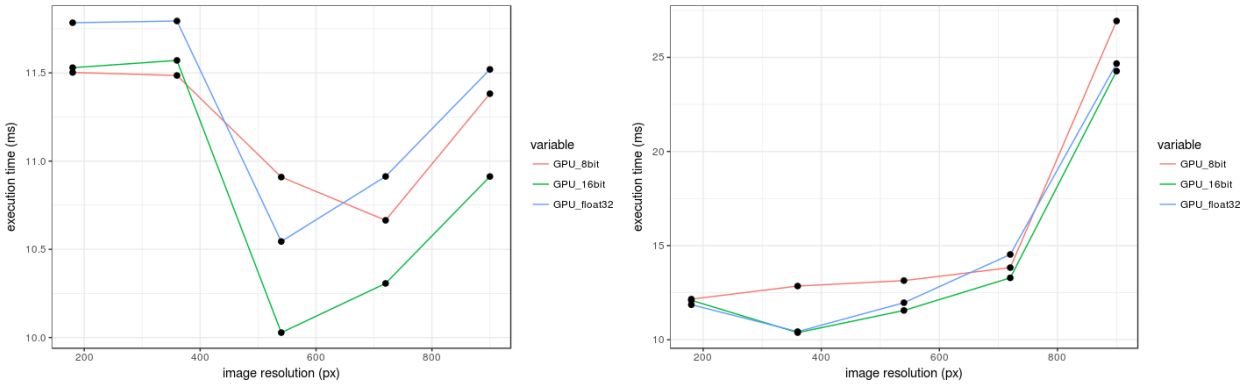

Fig 2. Execution time of our WebGL GPU implementation of the median filter depending on the size of the image, for 8-bit, 16bit and float32 images, with a 3x3 kernel (left) and a 7x7 kernel (right).

As shown on the Fig. 2, for the GPU implementation with the 3x3 kernel, the execution time for the two smallest image is longer than the 3 bigger images, but the times are very close for all the image resolution. Times for 8-bit, 16-bit and float-32 appear similar, as the fastest execution between these 3 image type vary between the different resolution. For the 7x7 kernel, only a slight increase in time is visible from the smallest image to the fourth biggest one, but there is a sharp rise in execution time for the 900x720 image. There is no noticeable difference between the different image types. The times are overall slightly longer than for the 3x3 kernel.

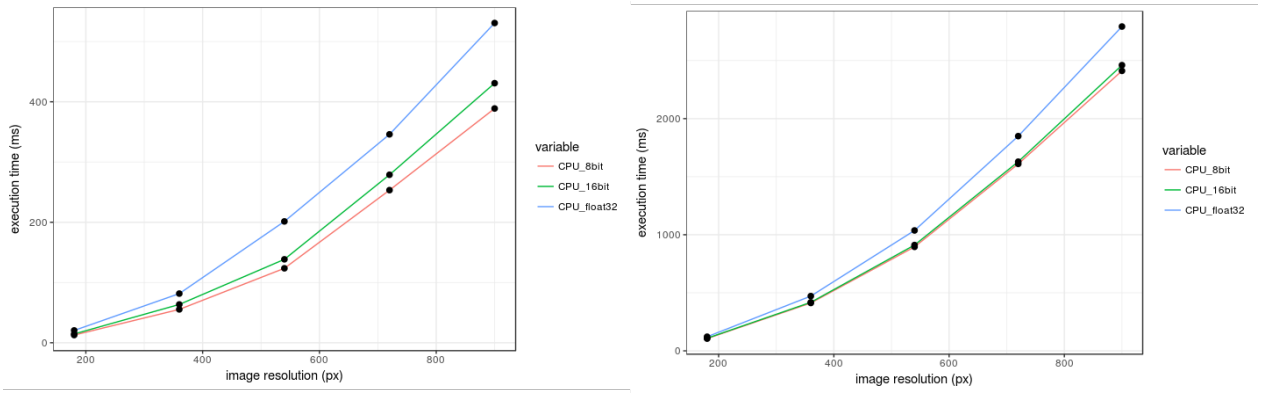

Fig 3. Execution time of the TIMES CPU implementation of the median filter depending on the size of the image, for 8-bit, 16bit and float32 images, with a 3x3 kernel (left) and a 7x7 kernel (right).

For the TIMES module implementation the graph has an identical profile for the 3x3 and 7x7 kernel (Fig. 3), only with an overall much worse time for the 7x7 kernel for all image resolution. The execution time

increase exponentially with the size of the image, and float-32 is slightly slower than the other image types. The execution times are also much longer than for the GPU implementation.

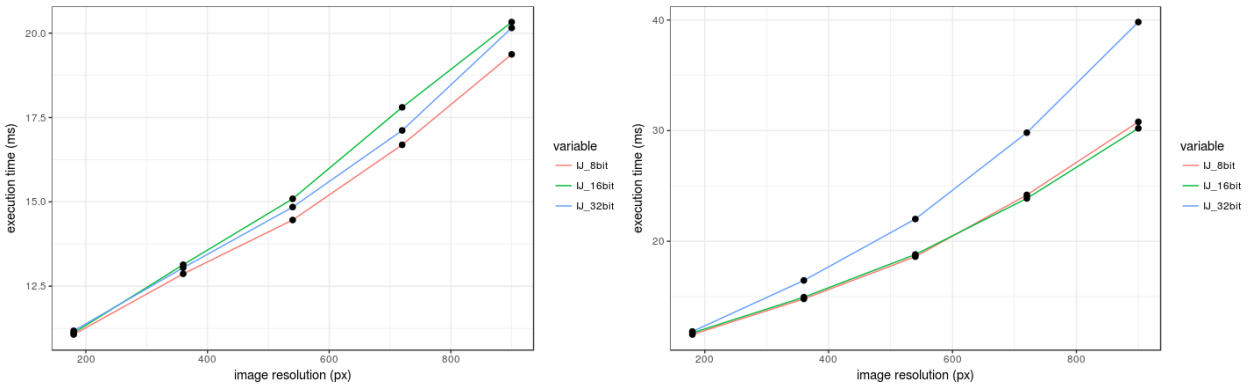

Fig 4. Execution time of the ImageJ implementation of the median filter depending on the size of the image, for 8-bit, 16bit and float32 images, with a 3x3 kernel (left) and a 7x7 kernel (right).

For the ImageJ implmentation, the execution time increase linearly with the image size for both the 3x3 and 7x7 kernels (Fig. 4). The execution time is noticeably slower for the 32-bit with a 7x7 kernel compared to the other image types, but not with a 3x3 kernel.

4.Discussion

By comparing the results obtained between our implementation of the median filter and the ImageJ filter, slight differences between all internal border are visible. The results obtained with ImageJ are similar to what could be obtained with a larger kernel size with our implementation. This differences might be explained by a difference in the definition of circular kernel between ImageJ and the TIMES modules (how much of a pixel must be in the circle to be considered part of the kernel ?).

In terms of performance, the GPU implementation, CPU implementation and ImageJ implementation all behaves significantly differently. The CPU implementation execution time increase exponentially, which means it increase faster than the other two implementation. Even for the 360x288 image it is more than one order of magnitude slower than the other implementation, and it exceeds the two order of magnitude of difference with ImageJ and the GPU filter. It is also the filter with the greatest difference in execution time between the two kernel size. The imageJ filter increase linearly in execution time. With the image tested, it never went beyond double the execution times of the GPU implementation. The times for the 7x7 kernel are less than the double than for the 3x3 kernel, so it is far less sensible than the CPU implementation to an increase of the kernel size. Among our results, the GPU implementation is the fastest. The results for the 3x3 kernel are surprisingly slower for the smallest image, which is surprising. In this case the benchmark might not have been executed properly. The increase in time between image size is minimal except for the 900x720 image. The GPU filter is much faster than the CPU implementation, which is to be expected, as the usage of WebGL allowed for the parallelization of the treatment of each pixel. The difference in time

between GPU and ImageJ could be explained by the facts that the ImageJ filter seems to blur image more, which could indicate that more pixels are included in the kernel, which would mean more pixel value to sort, and sorting is a time consuming process.

5.Conclusion

Results obtained with our filter are similar, but not quite equal, as the ImageJ result are slightly more blurry. The execution time of our GPU implementation is much faster than the CPU implementation because of the parallelization of the treatment of each pixel, which mean our implementation has been successful. Further improvement to add to our filter may be to chang ethe implementation of the sort algorithm, as the the bubble sort used is a fairly slow sorting algorithm, and sorting is the most time-consuming part of median filters.

References

[ˆHua1979] Huang T, Yang G, Tang G. A fast two-dimensional median filtering algorithm. IEEE Transactions on Acoustics, Speech, and Signal Processing. 1979;27(1):13–18.

[ˆHua1981] Huang TS. Two-dimensional digital signal processing II: transforms and median filters. Springer-Verlag New York, Inc.; 1981.

[ˆWei2006] Weiss B. Fast median and bilateral filtering. 2006;25(3):519–526. Acm Transactions on Graphics (TOG).

[ˆSoi2002] Soille P. On morphological operators based on rank filters. 2002;35(2):527–535. Pattern recognition.

[ˆTuk1974] Tukey J. Nonlinear (nonsuperposable) methods for smoothing data. Congr Rec 1974 EASCON. 1974;673.

[ˆFle1896] Philip J. Fleming and John J. Wallace. 1986. How not to lie with statistics: the correct way to summarize benchmark results. Commun. ACM 29, 3 (March 1986), 218-221.

[ˆPar12] Tony Parisi. WebGL: up and running. ” O’Reilly Media, Inc.“, 2012.