# Short Programming Project: Implementing the algorithm from Vaz et al.

Nick Borchers (s2865947)
Supervised by dr. M.H.F. Wilkinson

February 2019

## 1 Problem statement

Micheal S. Vaz et al. proposed a new algorithm for the decomposition of 2D and 3D Euclidean spheres [1]. This decomposition is very useful for speeding up the elementary operations of mathematical morphology: dilation and erosion. Mathematical morphology is used to analyze image shape features with many applications in for example object recognition and many specific applications in medical imaging. Dilation and erosion (the elementary operations of mathematical morphology) can be a computational obstacle when handled incorrectly; the computation time will increases polynomially with respect to the diameter of the SE when handled naively.

For this project we will implement this algorithm in C as well as make use of its parallel capabilities with OpenMP. We will implement variation **2b** of the algorithm listed in the paper[1]. With this variation the structuring element will be decomposed in a union of partitions $P$. A partition consists of a cubic factor and a sparse factor. The cubic factor is big compared to the sparse factor, but we know it is cubic. This cubic property is very useful when using for example the HGW algorithm for computing the dilation/erosion. Conveniently, HGW algorithm also lends itself to a parallel implementation which we will also do.

In this report I will show the design steps in designing a parallel implementation of the SE decomposition

# 2 Structure Chart

Several features of this algorithm lend themselves to a very useful parallel implementation on various levels of abstraction: on a very low level we can parallelize the HGW algorithm to run over every row simultaneously. On a high level we can have two types of threads: one that decomposes the structuring element step by step and enqueues every partition p. The other type of thread takes partitions from the queue and applies the dilation/erosion operations on the image with the cubic and sparse factor.

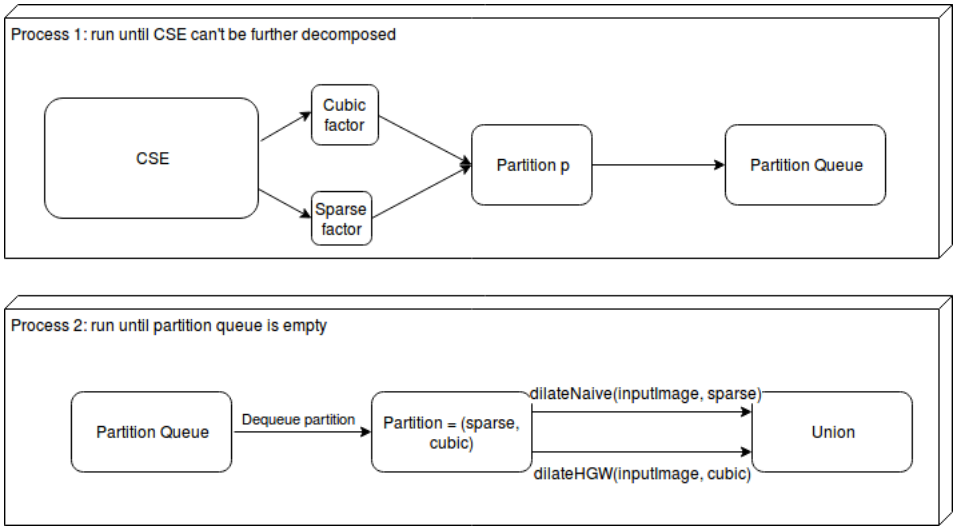
## 2.1 The partitions

The sparse factor consists of a maximum of 4 'true' pixels, this factor is so small that when computed naively it will be faster than more complicated algorithms due to overhead of allocating buffers, starting threads, etc. A handy feature of this factor is that it can be represented as 4 numbers, namely 4 offsets from the origin in all directions.

The cubic factor is where the computational bottleneck resides. However, due to its cubic shape we are going to use a parallel HGW implementation to decrease its computational complexity drastically.

We compute these dilations/erosions one after another and wait for the next partition to be dequeued from the partition queue.

**A visual representation of the high-level workings of the implementation**




# 3 Implementation

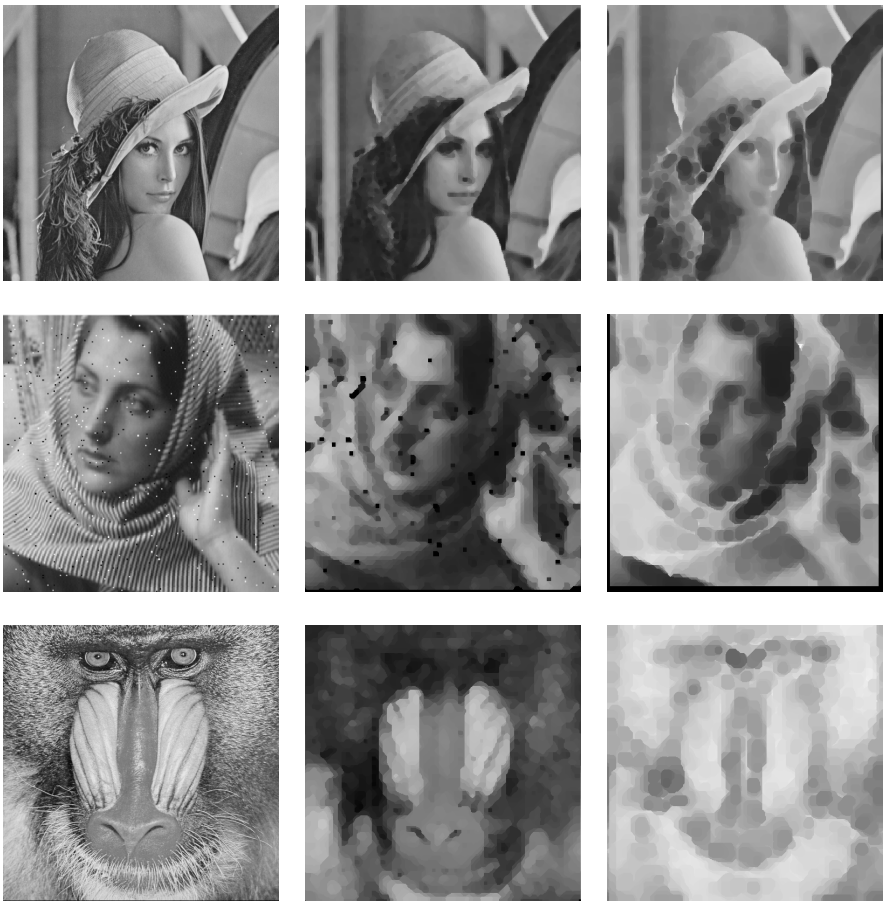
First I created a framework in C with parallel morphological operations like dilation and erosion. I also use an open-source image I/O library called *stbi_image*. I implemented the HGW algorithm for dilation and erosion to be ran in parallel across blocks of the image. I also implemented naive dilation and erosion for the cubic factors due to efficiency reasons and the erosion and dilation functions for the unfortunate case when the cubic factor is of size 3.

Now that I had a basic framework for my implementation, I started writing the actual decomposition. Due to complexity/time limitations, I decided on limiting this program to only use symmetrical structuring elements (this means that structuring elements that are convex but not symmetrical do not work properly).

The implementation and details on how to use it can be found at the following Github repository:
https://github.com/Nickborchers/seDecomposition

# 4 Results

Here you can see a couple of images with their morphological opening and closing, respectively. From the top down: SE decomposition using a spherical SE with a radius of 5, 6 and 8.



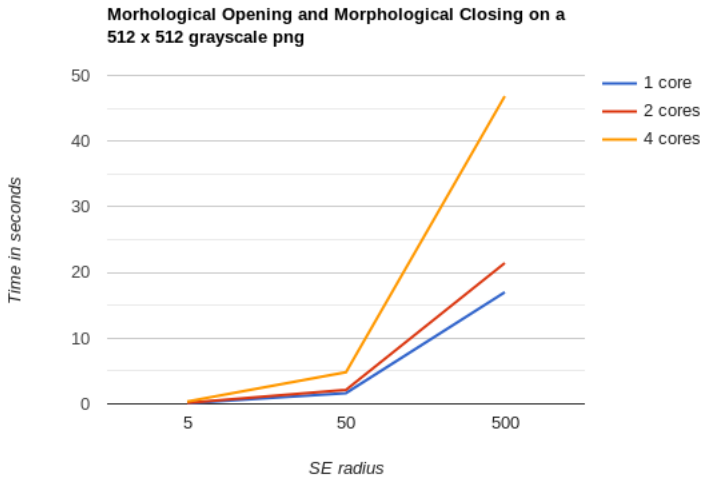
It would also be really interesting to compare the performance of this implementation of other ones but unfortunately I did not have access to a big library of suitable test images (8 bit gray-scale *png* images) or a brute-force dilation/erosion implementation to compare this implementation with.

**Decomposition of a sphere with radius 5, 50, 500 and 1000:**

| | 5 | 50 | 500 | 1000 |
|---|---|---|---|---|
| 1 | 0.000 | 0.002 | 2.269 | 19.842165 |
| 2 | 0.000 | 0.003 | 2.316 | 18.630728 |
| 4 | 0.000 | 0.003 | 2.264 | 18.313162 |

**Morphological opening followed by morphological closing using the decomposed SE on a 512 x 512 grayscale png**

| | 5 | 50 | 500 |
|---|---|---|---|
| 1 | 0.091 | 1.645 | 17.008 |
| 2 | 0.132 | 2.131 | 21.462 |
| 4 | 0.336 | 4.812 | 46.827 |




We can see that the computing time it takes for the morphological operations to complete increases when we increase the number of cores. This is because of the way memory is handled with this implementation. I will discuss what can be improved in the next section.

# 5 Improving the program

For every write/read from/to the original image, we need a critical section in OpenMP. To circumvent this we use a buffer to temporarily store the part of the image the thread needs and write the result back to the image after the necessary computation. This is very expensive because we need two critical sections in every thread to write/read from/to the buffer. This is a nice place to start if one would want to improve the program; find a good way around this. Perhaps by using only one named critical section for the writing back to the original image and not restricting the reading of the image initially. Another big im-

provement would be implementing a good solution to the producer-consumer problem for the queue which is supposed to be handled concurrently by using a mutex semaphore on the queue and two more semaphores to count the number of empty and available spots in the queue. After this, it would be interesting to look at a 3D implementation.

# 6 Epilogue

I knew from the start that this was not going to be the simplest of projects; I was unfamiliar with image morphology and even image processing in general. But I was also interested in taking up a challenge. One problem I encountered early on was that I did not know where to start. I did not understand some parts of the algorithm initially. Therefore I just decided on writing a framework first, while asking questions to my supervisor about the algorithm.

Also I first spent quite some time on trying to port an open-source HGW implementation. In hindsight this was not really a good idea because it did not help me understand the actual algorithm and in the end I still wrote it myself.

Another unfortunate side-effect I discovered later in the process was that the I/O library I'm using has memory leaks. I decided on still sticking to it because picking a different one could potentially take more time and this problem could also be considered out of scope.

# References

[1] Micheal S. Vaz, Atilla P. Kiraly, and Russell M. Mersereau. *Multi-level decomposition of Euclidean spheres.* Rio de Janeiro, Brazil, Oct. 10 –13, 2007, MCT/INPE, v. 1, p. 461–472