Université Libre de Bruxelles

Image processing

INFO-F404 - Parallel computing project

Baudru Julien

December 3, 2020

Contents

1 Introduction

In the scope of the new "global security" law voted in France, we were asked to create the most efficient software possible to blur the faces of police officers.

2 User manual

To compile this program simply move it to the directory *BAUDRU_Julien_000460130* and run the following command:

```
make
```

Then the program can be started by entering the following command:

```
./main <source_image>.raw <mask> <output_file_name>.raw <blur_level>
```

The source image must be the an image of type *raw*, same for the output image. The mask is a simple file containing at each line the positions of the upper left and lower right points of the rectangle defining where the image should be blurred. The last argument, the blur level, defines the number of neighboring pixels to be taken into account to create the blur effect, the higher this number will be, the more blurred the effect will be. The operation of this algorithm is explained in point 5.1 of this report.

In addition to the main program, you will also find a program to generate masks in a pseudo-random way. This program must be compiled with the following command in the directory *BAUDRU_Julien_000460130* :

```
gcc maskgenerator.c -o maskgenerator
```

And executed with the options shown below :

```
./maskgenerator <output_file_name> <number_of_blurred_areas>
```

For example, if we start this program with the command *./maskgenerator data/mask/mask_test 4* we can obtain the result below:

```
1103 339 1735 380
563 536 1459 607
520 198 1046 212
695 485 1667 680
```

The only purpose of this second program is to facilitate the testing of the main blurring program.

3 Structure

3.1 Overall structure

The structure of the project presented in these pages is the following :

```
BAUDRU_Julien_000460130/
|
|---- data/
|     |---- mask/
|     |     |---- mask1
|     |     |---- mask2
|     |     ...
|     |---- image/
|           |---- police1.raw
|           |---- police2.raw
|           ...
|
|---- openmpi/
|
|---- test/
|     |---- test.raw
|     |---- test2.raw
|     ...
|
|---- image.c
|---- image.h
|---- main.c
|---- makefile
|---- maskgenerator.c
|---- parse.c
|---- parse.h
```

The program created for this project uses the MPI library for process parallelization, this library is located in the folder called *openmpi*.

ENCORE

3.2 Overview of functions

3.2.1 In the parse.c file

The 'class' *image* contains only the function *parse*. This function is used to transform the data contained in the mask given as an argument into a usable data structure, in this case this data structure is a matrix of size ***Number of line in file x 4***.

3.2.2 In the image.c file

The 'class' *image* contains the following functions : *loadimage*, *createimage* and *blur*.
The purpose of the *loadimage* function is to transform the *raw* image given as an argument into a usable data structure, in this case into a character list of size ***1280 x 720***. On the other hand, function *createimage* performs the opposite role to the function described above, this function is called at the end of the program execution to create the new image resulting from the transformations.
Finally, as its name indicates, the role of the *blur* function is to blur a given rectangle of the image. This function takes as parameter the coordinates of a single rectangle and not all the lines of the mask file because, during the parallelization, each process will take care of blurring a particular rectangle.

4 Additional work

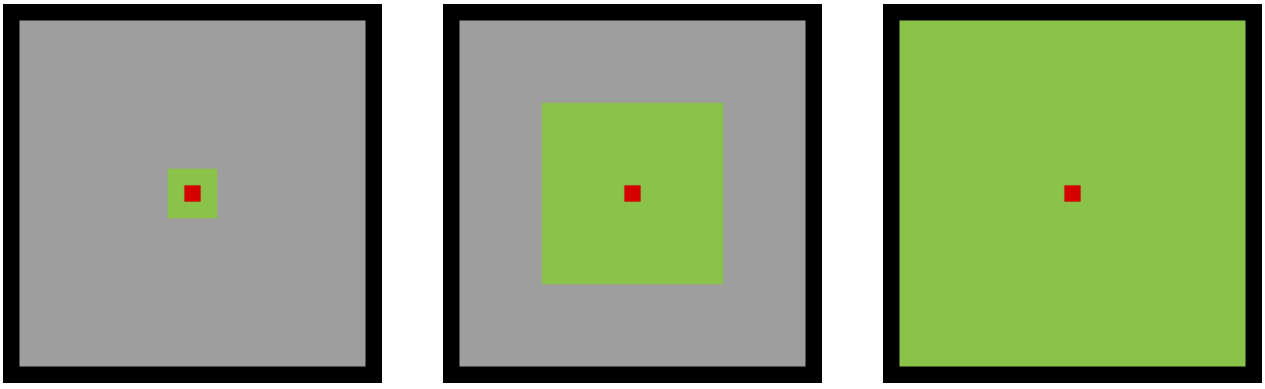
5 Implementation choices

To code this program the C language has been chosen rather than Rust to avoid the loss of time caused by learning a new language.

EXPLIQUER CE QUI SUIT

5.1 Blurring algorithm

The operation of this algorithm is as follows: For each pixel included in the rectangle to be blurred, we take the mean of the values of the neighboring pixels of this pixel and we attribute this value to the pixel we are processing. Neighboring pixels are the set of pixels that form a square of $N\ x\ N$ size whose center is the pixel being processed. The size of the neighborhood is defined by the parameter N, more concretely, for the examples below, the value of the red pixel is determined by the average value of the green pixels :



(a) N = 1 (b) N = 5 (c) N = 10

Figure 1: Visualization of the average calculation

Here below are the effects of the variation of the parameter N on the same image.

(a) Source image (b) N = 5 (c) N = 10

(d) N = 25 (e) N = 50

Figure 2: Variation of the parameter N

Note : To improve the layout of this report, in the examples above, the blur mask has the same size as the image and the images have been cropped.

5.2 Slave/Master work dispatch

SCHEMA MAITRE ESCLAVE

6 Difficulties