IMAGE Report 6 - Programmation

Carl Robinson **19th Jan 2018**

Studying the Model

1)

The image size is 256 x 256 = 65536 pixels, so you would assume it was 65536 bytes in size.
However the command `ls -l lacornou.pgm` shows the size to be 65551 bytes (15 bytes extra).

The difference is due to the header row in the image file, as revealed by the command `head -1 lacornou.pgm`. This header row is `P5 256 256 256`, which is 15 characters long including the whitespaces. As each ascii character occupies 1 byte of memory, this accounts for the extra 15 bytes discovered earlier.

2)

The pixels should be visited row by row. This can be implemented as a nested for loop, where the outer loop iterates over the rows, and the inner loop iterates over the columns in each row. Reading one row at a time from the RAM allows for the most efficient use of the RAM and the L1/L2 caches with respect to pagination. A sequential block of image data is read from RAM into cache at one time for each pagination. This minimises the number of RAM accesses required, and maximises the speed of read and write operations on the image. The same benefit applies to read and write operations between hard drive and RAM, which are much slower by comparison, and for which such optimisation is even more important.

3)

The skeleton program uses malloc to allocate space for 4 lines of the image, and defines pointers to each of these lines. It then reads in the first two pixel rows of the image into line0 and line1. Then for each row of pixels of the image, it reads the row into line2

Using pointer manipulation allows us to permute the pointers pointing to the 3 memory buffers used for our operations. This helps us avoid performing many read and write operations in RAM, as for each new pixel row read from the image, the two of the previous three rows in memory can be reused. Permuting the pointers is computationally inexpensive in comparison.

We update pointers 1 and 2 by one, so they point to the buffer one place ahead. Pointer 3 is updating to point to the buffer that holds the top pixel row of the three in memory, and becomes the buffer that holds the newly read line.

Sobel Operator

4)

The first row of pixels has been replaced by the second line of pixels, which has been written to the output file twice. Similarly the last row of pixels has been replaced by the second-to-last line of pixels, which has been written to the output file twice. Both input and output files have the same number of rows.

Top of input image:

Top of output image:

This happens because our outer loop starts at l=1, and ends at l < hauteur-1 (the row index starts at 0). The code writes a line twice only when l=1 (the second line) and when l=hauteur-2 (the second-to-last line).

5)

L1-norm, also known as least absolute deviations, minimises the sum of absolute differences between the target value ($||\nabla||$) and the estimated values (∇_x^2, ∇_y^2).

L2-norm is also known as least squares. It is basically minimizing the sum of the square of the differences between the target value ($||\nabla||$) and the estimated values ($|\nabla_x|$, $|\nabla_y|$).

- **Robustness to outliers:** L1 is robust in that it is resistant to outliers in the data, so it is helpful where outliers may be safely and effectively ignored. If it is important to pay attention to any and all outliers, the L2 method is a better choice.
- **Computational efficiency:** L1-norm does not have an analytical solution. L2-norm does, allowing for solutions to be calculated computationally efficiently.
- **Unique solutions:** L2-norm has unique solutions while L1-norm does not.

6)

```
// iterate over cols (all pixels in the line)
for (c=1 ; c<largeur ; c++)
{
  /* A vous de jouer... */
  deltaX = ligne[1+1][c+1] + (2 * ligne[1][c+1]) + ligne[1-1][c+1] - ligne[1+1][c-1] - (2 * ligne[1][c-1]) - ligne[1-1][c-1] ;
  deltaY = ligne[1+1][c+1] + (2 * ligne[1+1][c]) + ligne[1+1][c-1] - ligne[1-1][c+1] - (2 * ligne[1-1][c]) - ligne[1-1][c-1] ;

  normL1 = (abs(deltaX) + abs(deltaY)) / 6 ;
  normL2 = (sqrt(pow(deltaX, 2) + pow(deltaY, 2))) / 4 ;

  ligne_res[c] = normL1 ;
  // ligne_res[c] = normL2 ;
}
```

- Thanks to the permutation code at the end of the main function, the pointers ligne[0], ligne[1], ligne[2] always point to buffers that contain pixel rows that are vertically arranged in the original image. This is to say the row of pixels pointed to by ligne[1] is in the line immediately below the row pointed to by ligne[0], and above the row pointed to by ligne[2]. This allows us to perform simple manipulation of the counters to point to all surrounding pixels of the image.

The resulting images for L1 (left) and L2 (right) norms:

- We see bright white lines marking the contours of the images, corresponding to pixels with high gradients, as calculated using the relative difference in grayscale value of the pixels immediately surrounding it.
- Large areas of black coloured pixels are present, corresponding to pixels in the original image whose gradients are very low or zero.
- The white contours have a soft, slightly blurred appearance, which corresponds to a smooth transition of grayscale value in the original image. This is particularly noticeable on the text ‘Lacornou’ on the boat.

Zoom on images for L1 (left) and L2 (right) norms:


- The L2 norm produces brighter, clearer contours than the L1 norm.