IMAGE Report 7 - Programmation pt.2

Carl Robinson & Majd Abazid **2nd Feb 2018**

Sobel Operator

5)

The maximum value of the L1 norm is ~1,448, calculated as √2 multiplied by the maximum value of the L2 norm (1024). This value is outside of the range of possible colour values expressed with one byte (256 colours), so we must scale the value down. As 1448/6 = ~241, a number less than 255, we choose to divide by 6.

For the square mesh, the L1 norms make it possible to explicitly characterize the neighborhood by adopting a discrete ray, which is why L1 norm is sometimes preferred. The norm L1 gives a maximum value equal to the maximum value of L2 multiplied by √2, that is to say it amplifies any noise (which is high frequency).

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.

7)

Adding some code allows us to correctly handle the special cases of the top/bottom/left/right edges, and the four corners:

```
//iterate over rows
for (l=0; l<hauteur; l++)
{
  //read second line (the new row of pixels from the image)
  READ_LINE (ligne[2]) ;

  // iterate over cols (all pixels in the line)
  for (c=0 ; c<largeur ; c++)
  {
    //if top row
    if (l==0) {
      //if top left corner
      if (c==0) {
        deltaX = (2 * ligne[1+1][c+1])  + (2 * ligne[l][c+1]) - (2 * ligne[1+1][c]) -(2 * ligne[1][c]) ;
        deltaY = (2 * ligne[1+1][c])  + (2 * ligne[l+1][c+1]) - (2 * ligne[1][c]) -(2 * ligne[1][c+1]) ;
      }
      //if top right corner
      else if (c==largeur-1) {
        deltaX = (2 * ligne[1+1][c+1])  + (2 * ligne[l][c+1]) - (2 * ligne[1+1][c]) -(2 * ligne[1][c]) ;
        deltaY = (2 * ligne[1+1][c])  + (2 * ligne[l+1][c+1]) - (2 * ligne[1][c]) - (2 * ligne[1][c+1]) ;
      }
      //else (other top row)
      else {
        deltaX = (2 * ligne[1][c+1]) + (2 * ligne[1-1][c+1]) - (2 * ligne[1][c-1]) - (2 * ligne[1-1][c-1]) ;
        deltaY = ligne[1-1][c-1] + (2 * ligne[1-1][c]) + ligne[1-1][c+1] - ligne[1][c-1] - (2 * ligne[1][c]) - ligne[1+1][c] ;
      }
    }
    //if bottom row
    else if (l==hauteur-1) {
      //if bottom left corner
      if (c==0) {
        deltaX = (2 * ligne[1-1][c+1])  + (2 * ligne[l][c+1]) - (2 * ligne[1-1][c]) - (2 * ligne[1][c]) ;
        deltaY = (2 * ligne[1-1][c])  + (2 * ligne[l-1][c+1]) - (2 * ligne[1][c]) - (2 * ligne[1][c+1]) ;
      }
      //if c=largeur (bottom right corner)
      else if (c==largeur-1) {
        deltaX = (2 * ligne[1][c])  + (2 * ligne[l+1][c]) - (2 * ligne[1][c-1]) - (2 * ligne[1+1][c-1]) ;
        deltaY = (2 * ligne[1][c-1])  + (2 * ligne[l][c]) - (2 * ligne[1+1][c-1]) - (2 * ligne[1+1][c]) ;
      }
      //else (other bottom row)
      else {
        deltaX = (2 * ligne[1][c+1]) + (2 * ligne[1+1][c+1]) - (2 * ligne[1][c-1]) - (2 * ligne[1+1][c-1]) ;
        deltaY = ligne[1-1][c-1] + (2 * ligne[1-1][c]) + ligne[1-1][c+1] - ligne[1][c-1] - (2 * ligne[1][c]) - ligne[1+1][c] ;
      }
    }
    //if other left edge
    else if (c==0) {
      deltaX = (2 * ligne[1][c+1]) + (1 * ligne[1+1][c+1]) + (1 * ligne[1-1][c+1]) - (2 * ligne[1][c]) - (1 * ligne[1+1][c]) - (1 * ligne[1-1][c]) ;
      deltaY = (2 * ligne[1-1][c+1]) + (2 * ligne[1-1][c]) + ligne[1-1][c+1] - (2 * ligne[1+1][c+1]) - (2 * ligne[1+1][c]) ;
    }
    //if other right edge
    else if (c==largeur-1) {
      deltaX = (2 * ligne[1][c-1]) + (1 * ligne[1+1][c-1]) + (1 * ligne[1-1][c-1]) - (2 * ligne[1][c]) - (1 * ligne[1+1][c]) - (1 * ligne[1-1][c]) ;
      deltaY = (2 * ligne[1-1][c-1]) + (2 * ligne[1-1][c]) + ligne[1-1][c-1] - (2 * ligne[1+1][c-1]) - (2 * ligne[1+1][c]) ;
    }
    //else (not on any edge/corner) - lines from previous class
    else {
      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] ;
    }

    // calc the L1 and L2 norms
    normL1 = (abs(deltaX) + abs(deltaY)) / 6 ;
    normL2 = (sqrt(pow(deltaX, 2) + pow(deltaY, 2))) / 4 ;

    // add norm to list
    // ligne_res[c] = normL1 ;
    ligne_res[c] = normL2 ;
  }

  //write the res line to the output file
  WRITE_LINE (ligne_res) ;
```

Local Averaging

8)

The algorithm that saves partial sums over columns along an image line allows computing the local sums with only 4 operations per pixel. This technique is most beneficial for large values of N, as the number of operations remains constant whatever the value of N.

9)

```
//iterate over rows (don't do first and last)
for (l=1; l<hauteur-1; l++)
{
  //read second line (the new row of pixels from the image)
  READ_LINE (ligne[2]) ;

  //sum the 3 lines column-wise and put result in ligne_colsum
  //2 additions per pixel
  for (c=0 ; c<largeur ; c++)
  {
    ligne_colsum[c] = ligne[0][c] + ligne[1][c] + ligne[2][c] ;
  }

  //for each element of vector, sum it with the two following elements (don't do edges)
  //2 additions per pixel
  for (c=1 ; c<largeur-1 ; c++)
  {
    ligne_res[c] = (ligne_colsum[c-1] + ligne_colsum[c] + ligne_colsum[c+1]) / 9 ;
  }

  //write the res line to the output file
  WRITE_LINE (ligne_res) ;

  //update all pointer by one, so they shunt forwards one place before treating the next row
  //this permutes the pointers in a cyclical fashion
  //save line0 pointer
  tmp =ligne[0] ;
  //shift pointer up
  ligne[0]=ligne[1] ;
  //shift pointer up
  ligne[1]=ligne[2] ;
  //copy line0 pointer into line2
  ligne[2]=tmp ;
}
```

10)

Original Image (left)

Local Averaged Image (right)

- We see that applying the local average algorithm to the image has resulted in it becoming blurred.
- This is equivalent to applying a low pass filter, which removes some of the high frequencies.
- The larger the value of N, the more blurred we can expect the resulting image to be.