

## Image Processing

### Program Description :

In this MP, you need to write a C program to process a given image and blur the image with a given specific metric. The specification of the image file is given. For the first checkpoint MP4.1, you have to read the given color image and store it in a 2d array. Since it is a color image, you need 3 arrays, one for each color (Red, Green and Blue). The resultant smoothened (blurred) image should be stored in a new image file.

For the second checkpoint MP4.2, you will be given a image which contains many rectangular shapes and there will be exactly one square in the image. You have to identify the square and return the value of its area and the center of the square. The shapes will be in solid color.

### Image format and Specification :

The image format to be used is the Portable Pixmap format (PPM). Each file starts with a two-byte file descriptor (in ASCII) that explains the type of file and its encoding. The descriptor is a capital P followed by a single digit number. We will always be using the P3 format and hence the first line will be P3.

An example of a PPM file is as follows :

```
P3
3 2
255
255 0 0
0 255 0
0 0 255
255 255 0
255 255 255
0 0 0
```

- The first line P3 denotes that it is a color image of PPM format
- The second line gives the size of the image ( 3 pixels by 2 pixels)
- The third line specifies the maximum pixel value
- The next line is the first pixel's RGB value (R = 255; G=0; B=0)
- Then comes the second pixel's RGB value and so on ..

The above file corresponds to the following image



### Checkpoint 1

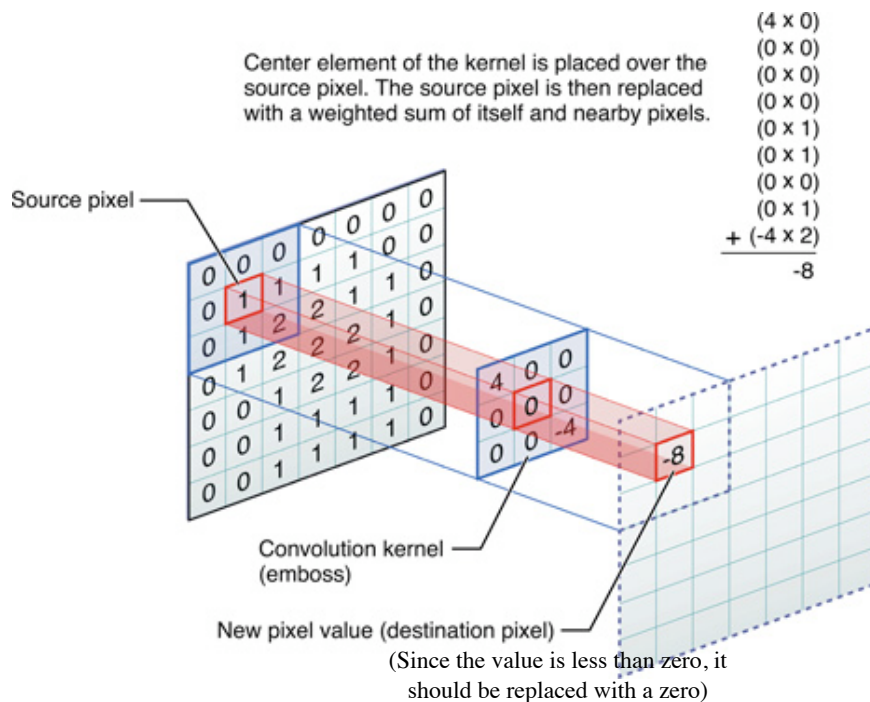
For checkpoint 1, you need to read a file *image.ppm* and store the values of the file in arrays. The file will be in ASCII encoding. File I/O has to be used to read in the values and stored in a 2d array. There should be three arrays named Red, Green, Blue and the corresponding values should be stored in the arrays. The maximum pixel value is specified in the image file and the pixel values of the output image file should not exceed this maximum pixel value. The size of the given image will be more than 100 x 100 pixels and will not exceed 200 x 200 pixels and it will always be of aspect ratio 1 (height will be equal to width). Each of the color channels R,G and B has to be blurred by the process described below and then the blurred R,G and B arrays should be written into a single color image file.

## Image Blurring

Images can be modified by a process called filtering. Various image processing techniques such as blurring, edge detection, noise cancellation etc can be achieved by the filtering process. The filter is defined by the filter array. The filter array will be given to you as *filter.txt*. The file format is as follows

```
3
4 0 0      The first line is the size of the filter ( 3 in the example). The rest of the file is the filter
0 0 0      array of the given size. This file should be read into an array named 'Filter'.
0 0 -4
```

The filter array is called the **convolution kernel**. This kernel is run through the entire image by the process defined in the figure. Based on what the filter array (kernel) values the output image will vary. This process is called as filtering.



### Note :

**During the filtering process, you have to divide this value (-8) by the sum of the values in the filter. If the sum of the values is 0, then you do not perform the division.**

**In this example, the division is not performed because the sum of the filter values is  $4 - 4 = 0$ .**

**If the sum of the filter values was not zero, (say the sum was 's'), then you have to divide -8 by 's'.**

Source : Internet

The filter array for blurring a image will be,

```
3
1 1 1
1 5 1
1 1 1
```

This process is iteratively done for every pixel in the image and for each array R,G and B. If the blurring filter is used, the image is blurred. The new pixel value **CANNOT** be negative. If the value goes below zero, it should be replaced with a zero. The pixel value cannot exceed the maximum pixel value which is mentioned in the input file. If the value exceeds it, then it should be replaced with the maximum value.

As you can see, if the filter is placed over a pixel at the edge of the image, it does not have enough information to process. The edge pixels are calculated with the filter values which are on the image and the other filter values are ignored.

## Checkpoint 2

For checkpoint 2, you will use the filtering technique used for the previous checkpoint. Recalling the fact that based on the numerical values of the kernel, different modifications can be done on the image, Checkpoint 2 requires you to run a given filter over the image which can detect edges in the image. An image with many rectangular shapes will be given to you. Out of those shapes, exactly one will be a square (the others will be rectangles). You have to identify the square and return the area and the center of the square.

### Edge Detection

The edge detection filter which when run over an image array will return an image in which the edges are of a different value and the rest of the image will be zeros.

```

3
-1 2 -1
-1 2 -1
-1 2 -1

```

This is the filter to detect vertical edges.

```

3
-1 -1 -1
2 2 2
-1 -1 -1

```

This is the filter to detect horizontal edges.

For example, consider the vertical edge shown below. After applying the vertical edge detection filter, the values will be as shown.

30 30 30 30 100 100 100 100	→	0 0 0 -210 210 0 0 0
30 30 30 30 100 100 100 100		0 0 0 -210 210 0 0 0
30 30 30 30 100 100 100 100		0 0 0 -210 210 0 0 0
30 30 30 30 100 100 100 100		0 0 0 -210 210 0 0 0
30 30 30 30 100 100 100 100		0 0 0 -210 210 0 0 0
30 30 30 30 100 100 100 100		0 0 0 -210 210 0 0 0
		↓
0 0 0 0 1 0 0 0		0 0 0 0 210 0 0 0
0 0 0 0 1 0 0 0		0 0 0 0 210 0 0 0
0 0 0 0 1 0 0 0	←	0 0 0 0 210 0 0 0
0 0 0 0 1 0 0 0		0 0 0 0 210 0 0 0
0 0 0 0 1 0 0 0		0 0 0 0 210 0 0 0
0 0 0 0 1 0 0 0		0 0 0 0 210 0 0 0

In the first array, the edge is there because the pixel values jump from 30 to 100. After filtering the array with the vertical edge detection filter, the second image is obtained which detects the edges but with negative values. The negative values are replaced by zeros (since the pixel values cannot be negative). Then the non-zero values are replaced by 1's. In the resulting array, 1 is the presence of an edge and 0 is the absence of an edge.

The vertical and horizontal edges are detected independently and stored in different arrays. Then these two arrays with horizontal and vertical edges are analyzed with co-ordinate geometry to detect the square and find the area and the center of the square.

### Handling Color Images and Edges

You can use any of the Red, Green or Blue image to detect the square. The image will be of sufficient contrast so that the edge can be detected using **ANY ONE** of the three color arrays.

## Grading

### Functionality (70%)

- Checkpoint 1 (30%)
  - The program should read in an color ppm image file and store it in three 2d arrays. Each channel should be blurred and then the three 2d arrays should be written into a single output color ppm image file.
  - The program should be able to read in images of any given size (as per the constraints previously stated)
  - The blurring filter which is specified as *filter.txt* should be used. You cannot use your own blurring filters.
- Checkpoint 2 (40%)
  - The program should use any one of the color channels and use the vertical and the horizontal edge detection filters.
  - The edge detection filters are as given in this handout. You **NEED NOT** read a *filter.txt*.
  - The area of the square can have an error of  $\pm 5\%$
  - The center of the pixel can have an error of  $\pm 2$  pixels

### Efficiency (10%)

- The program should run in the minimum time possible (The code should be optimized for speed. Unwanted loops can increase running time) (5%)
- The program should use minimum memory space (Declaring unwanted variables increases memory space) (5%)

### Style and Documentation (20%)

- Good and consistent indentations (5%)
- Introductory paragraph (5%)
- Commenting the loops and functions properly (5%)
- Using unambiguous variable names (5%)

---

## James Scholar Component

### Checkpoint JS1

The image should be magnified by 2x when it is being blurred. Magnification is done by duplicating pixel values. If the image given is 100 x 100 pixels, the output image will be 200 x 200 pixels. Each pixel in the input image will be duplicated 4 times in the output image. The image is first magnified 2x times and then the blurring filter is applied. This removes the jagged edges created due to magnification

### Checkpoint JS2

An image with many rectangular shapes will be given. Edge detection filters should be used to identify the shapes. There will be more than one square in the image. You are to find out the square with the largest area and return the area and the center of the square. (The rectangles should be ignored).