International Journal of Computer Applications (0975 – 8887)
Volume 14– No.5, January 2011

# Implementation of PPM Image Processing and Median Filtering

Sushant Pawar
SITRC
Nashik, India

Prasad S.Halgaonkar
MITCOE
Pune, India

J.W.Bakal
SESYTIET
Mumbai, India

V.M.Wadhai
MAE
Alandi, India

## ABSTRACT

Image consists of a rectangular array of discrete picture elements called pixels. The PPM (portable pixmap format (.ppm)) is a particularly simple way used to represent or encode a rectangular image as uncompressed data file. This .ppm file can be viewed with number of tools, including GIMP, gThumb image viewer etc. Reading and Writing PPM images are a difficult task which has been overcome successfully in our experiment.

In signal processing, it is often desirable to be able to perform some kind of noise reduction on an image or signal. The median filter is a nonlinear digital filtering technique, often used to remove noise. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image). Median filtering is very widely used in digital image processing because under certain conditions, it preserves edges whilst removing noise. The results shows the smoothened image.

**Keywords** – Digital Image Processing, Median Filter.

## I. INTRODUCTION

The PPM format is a lowest common denominator color image file format. It should be noted that this format is egregiously inefficient. It is highly redundant, while containing a lot of information that the human eye can't even discern. Furthermore, the format allows very little information about the image besides basic color, which means you may have to couple a file in this format with other independent information to get any decent use out of it. However, it is very easy to write and analyze programs to process this format, and that is the point.

It should also be noted that files often conform to this format in every respect except the precise semantics of the sample values. These files are useful because of the way PPM is used as an intermediary format. They are informally called PPM files, but to be absolutely precise, you should indicate the variation from true PPM. For example, "PPM using the red, green, and blue colors that the scanner in question uses."

The name "PPM" is an acronym derived from "Portable Pixel Map." Images in this format (or a precursor of it) were once also called "portable pixmaps." File extension for ppm images are .ppm, .pgm, .pbn, .pnm. PPM is for “pixmap” which represents full RGB colors. Each file start with a two-byte file descriptor (in ASCII) that explains its type (pbm, pgm and ppm) and its encoding (ASCII or binary). The descriptor is a capital P followed by a single digit number.

| File Descriptor | Type | Encoding |
|---|---|---|
| *P1* | Portable bitmap | ASCII |
| *P2* | Portable graymap | ASCII |
| *P3* | Portable pixmap | ASCII |
| *P4* | Portable bitmap | Binary |
| *P5* | Portable graymap | Binary |
| *P6* | Portable pixmap | Binary |

The ASCII based formats allow for human readability and easy transport so long as those platforms understand ASCII, while the binary formats are more efficient both at saving space in the file and easy to understand. When using the binary formats, PBM uses 1 bit per pixel, PGM uses 8 bit per pixel, PPM uses 24 bit per pixel, 8 for red, 8 for green, and 8 for blue.

The binary format of the image stores each color component of each pixel with one byte (thus three bytes per pixel) in the order of red, green, and blue. The file will be smaller in size but the color information will not be readable by humans.

The .ppm file is used to represent two types of images, one for grayscale images, corresponding to black/white photographs and the other for color images. For the grayscale image each pixel consists of 1 byte which is represented as unsigned char; a value ‘0’ is solid black and a value of ‘255’ is bright white. Intermediate are “gray” values of increasing brightness. The second one is color images correspond to color photographs, each pixel consists of 3 bytes with each byte representing as unsigned char, and this format is called RGB. Three byte represents the red component, green component and blue component when red = = green = = blue a grayscale “color” is produced.

These colors are additive,

(255, 255, 0) = red + green = bright yellow

(255, 0, 255) = red + blue = magenta (purple)

(0, 255, 255) = green + blue = cyan

(255, 255, 255) = red + green + blue = white

Using filtering it is possible to filter out the noise present in image. A high pass filter passes the frequent changes in the gray level and a low pass filter reduces the frequent changes in the gray level of an image. That is; the low pass filter smoothes and often removes the sharp edges.

A special type of **low pass** filter is the Median filter. The Median filter takes an area of image (3 x 3, 5 x 5, 7 x 7 etc), observes all pixel values in that area and puts it into the array called element array. Then, the element array is sorted and the median value of the element array is found out. We have achieved this by sorting the element array in the ascending order using bubble sort and returning the middle elements of the sorted array as the median value. The output image array is the set of all the median values of the element arrays obtained for all the pixels. Median filter goes into a series of loops which cover the entire image array.

Following are some of the important features of the Median filter [1], [2], [3]:

- It is a **non-linear** digital filtering technique.
- It works on a monochrome color image.
- It reduces 'speckle' and 'salt and paper' noise.
- It is easy to change the size of the Median filter.
- It removes noise in image, but adds small changes in noise-free parts of image.
- It does not require convolution.
- Its edge preserving nature makes it useful in many cases.

The median value selected will be exactly equal to one of the existing brightness value so that no round-off error is involved when we work independently with integer brightness values comparing to the other filters.

## II. PPM IMAGE FILE AND MEDIAN FILTER

### a) PPM Image File

#### 1. PPM file structure

The ppm file structure for the ppm header and Packed image data is as follows:
The ppm header

```
P6
320 240
255
```

The P6 indicates this is a color image (P5 means grayscale). The width (number of columns) of the image is 320 pixels and height (number of rows) of the image is 240 pixels. The 255 is the maximum value of pixel. Following the 255 is a \n (0x0a) which is a new line character. The image data, the value of red component of the upper left pixel must immediately follow the new line following the green and then the blue component. There must be a total of 3 x 320 x 240 = 230400 bytes of data.

#### 2. Read/Write PPM Image

The routines for reading and writing an image file are as follow,

1. int rows=0, columns=0;
2. unsigned char *buf1, *buf2;
3. char *in_name = "input_file.ppm";
4. char *out_name = "output_file.ppm";
5. buf1 = (unsigned char *) malloc ((3000000) *size of (unsigned char));
6. creat_image_file (in_name, out_name);
7. get_image_size (in_name, &columns, &rows);
8. the_image = allocate_imageArray (rows, columns);
9. read_image_array (in_name, imageArray);
10. /*call an image processing routines.*/
11. write_image_array (out_name, imageArray);
12. free_image_array (imageArray, rows);\

The first four lines declare the basic variable needed. Line '6'creates the output image to be just like the input image (same type and size). The output image is needed because the routine cannot write to an image file that does not exist. Line '7' reads the size of the input image after opening the input image file header information. The height (rows) and width (columns) are necessary to allocate the image array. These allocations take place in line '8', where it has been shown on buf1 declared as in line '5'. Actually the size (height and width) does not matter to the programmer, because we just read the size and allocate this size to the variable. Line '9' reads the image array from the input file. In this step after first checking the type of file (ppm, bmp, tif), this information is given from the header of the input image file. After getting information from header it extracts all the data of the input image and puts it into the image array. It also includes all the RGB component values present in the file.

When we get all the required information and data of an input image, the appropriate image processing routines are called to process the image data, and subsequently, the processed data is written into the output image file [4]. Finally, the memory array allocated as in line '8' is freed. These routines ride on top of the other routines that work with specific image file formats. They create file, determine if the file exists or not, manipulate headers, pull important information or put it into the file etc.

## 3. Read/Write Video file

The steps for video read/write are as follow,

1. Capture video of any type of format (.avi, .jpg, .mpeg etc.).
2. Strip video into a number of frames in .ppm format.
3. Read frames one by one with the use of **read ppm** routine.
4. Call the appropriate image processing routine.
5. Write frames one by one with the use of **write ppm** routine into the output file.
6. Merge all these output frames into the compressed form to make a movie.
7. Make an output movie/video in the required format to play using Mencoder.

We captured the video file and striped it into a number of frames and stored them into the input folder from where we extracted the frames one by one for read/write purpose. This is done by using **Mplayer** and **Mencoder (video codec)** on **Linux** platform. Mplayer strips the video file into a number of frames in the required format using command line for Mplayer.

- **mplayer -vo pnm input_file**

The above command strips the input video file into the number of frames. To read the frames one by one from the input folder in 'C' language we created a junk file that contains only the names of all the frames (e.g. 000000001.ppm, 00000002.ppm etc.). The junk file is opened in the read mode for reading the names of the frames one by one, and by the use of string concatenation, proper address is given to the input folder containing the input image frames that have been input to the read ppm routine. We address the output folder where the output frames will be stored in the same manner as explained for the case of input folder.

We used **read ppm** routine to read the frames one by one. Once the reading is completed, an appropriate image processing routine is called for processing the image data, which have been extracted while reading of the image, for example, RGB components etc. In the next step, the image file is written into the output folder one by one after processing. This is done by using **write ppm** routine.

Now, we have an output folder that contains all output image frame in a sequential manner. From these we create an output video file [5]. But prior to this we will require to convert the uncompressed type of output frames into the compressed format type (ppm to png). This is done with the help of **shell scripting** using the command as follows:

- **convert input_file.ppm output_file.png**

Now the output folder contains the sequence of frames in compressed png format (e.g. 00000001.png, 00000002.png etc.). From these we make a movie/video by the use of Mencoder command

- **mencoder mf://*.png fps = 25 –o video.avi –ovc lavc**

Using movie player Mplayer, the video file video.avi is played. The above command is only for video codec and for audio we need to specify appropriately. The Mencoder has various applications in the context of video processing. It includes the conversion of one video format into the other, frame rate specification, frame size specification etc. The implemented read/write routine can handle any number of movie frames for reading and writing.

### b) Median Filter

The steps for the implemented median filtering are as follow.

1. Take an input image array.
2. Append two rows and columns of 'zeros' at the end of input image array.
3. Initiate a 3 x 3 matrix, starting from the pixel, whose value is going to change after filtering. Initialize from the first pixel of input image array.
4. Extract all 3 x 3 matrix elements and put into the 1-D element array.
5. Sort the element array in ascending order.
6. Extract the middle value of sorted element array and put into the output image Array.
7. Repeat steps three to six for a complete image.

Working of Median filter is also shown in the following figure 1.

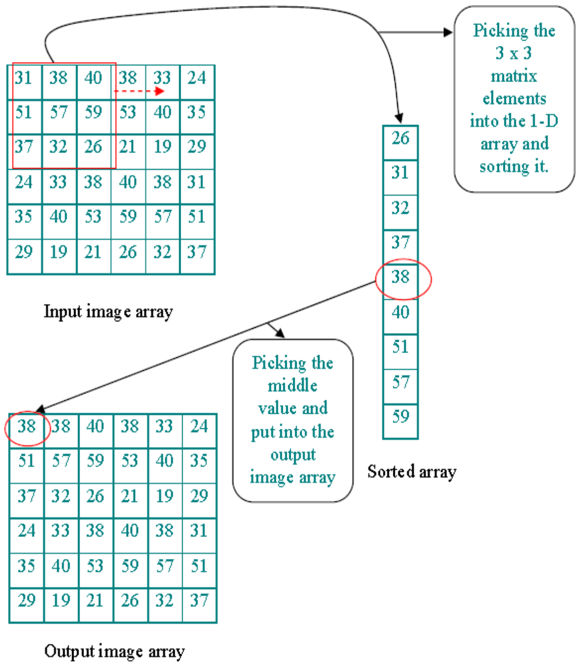

Figure 1: Low-Pass filtering with 3 x 3 Median filter

The speed of computation of an 3 x 3 Median filter and the 3 x 3 convolution filter is almost equal. However, when we moved to use 5 x 5 and higher orders, the Median filter slows down because of the continuous sorting and picking of the median value throughout image.

## III. EXPERIMENTAL RESULTS

The Read/ Write PPM Image routines were implemented in C language in Linux OS with the help of GCC compiler. Whereas median filter was also implemented in C language[6], [7] on Linux platform [8]. Following figure 2 and figure 3 shows the input to the Median filter and resulting output. It is evident that the filtered output image gives the smoothness of the image.

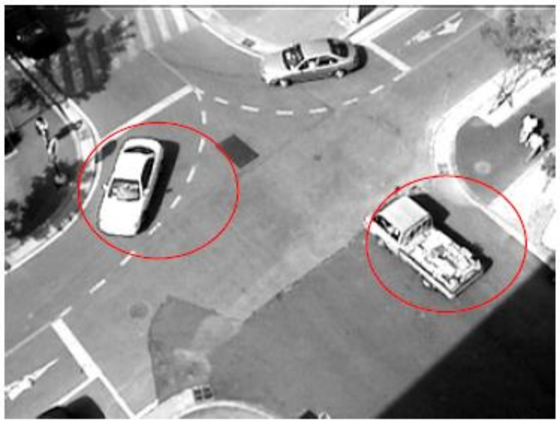

Figure 2: Input image to the Median Filter

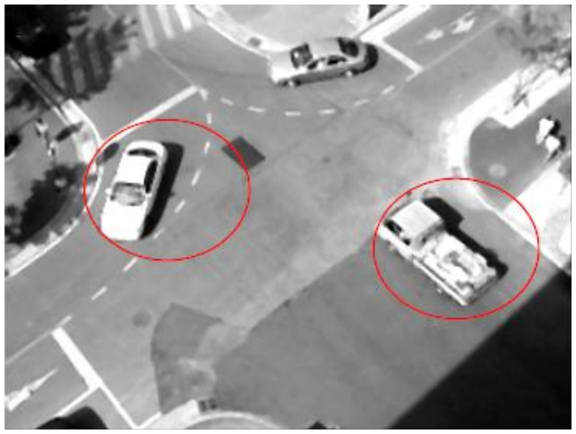

Figure 3: Result of 3 x 3 Median Filter

## IV. CONCLUSION

Read/ Write of PPM Image file and Read/ Write of Video file have successfully been carried out along with median filter. These are the preliminary but necessary steps to satisfy our goal to provide a better and efficient estimation of the object motion to be used in the smart camera for various applications, which captures and processes the image to extract application-specific information. Our future work focuses on developing the algorithm for object tracking.

## V. REFERENCES

[1] Sebastian Montabone, "Beginning Digital Image Processing: Using Free Tools for Photographers", Apress 2010.

[2] Wilhelm Burger, "Principles of Digital Image Processing: Core Algorithms", Springer 2009.

[3] R. Gonzalez, R. Woods, "Digital Image Processing", Pearson Prentice Hall Ltd., 2007.

[4] Fredric Patin, "An Introduction of Digital Image Processing," June 2003.

[5] Allen Bovik, "The Essential Guide to Video Processing," Academic Press- 2nd Edition 2009.

[6] Byron Gottfrried, "Programming with C," Schaum's Outline, 2nd Edition 1999.

[7] Yeshwant Kanetkar, "Let Us C," Allied Publisher, 3rd Edition 1998.

[8] J. Purcell, "Inc Linux Complete Command Reference," Red Hat Software, 1997.