Lab 1

Use two images for each operation to do the following operations and write down their advantages and disadvantages and explain your results:

2. Test the given pgmreader program use 3x3 average filter and median filter to process images and "lena.pgm" and "noise.pgm" and output the processed image (using pure c++).

Algorithm:

Use pseudo code or figure to display the algorithm.

1) Average Filter

```
AverageFilter(Image *inimage)
 begin
    tempData= the copy of original 2D matrix of the picture
    outImage= the copy of inimage
    for i=1 to Height-1 do    //Except the margin
        for j=1 to Width-1 do
            tempArr= values of 8 surrounding pixel in 3×3 matrix of tempData[i][j]
            average= the average of numbers in the tempArr
            tempData[i][j]=average
    OneDimData=change the tempData to 1 dimensional array by adding the data one by one
    outImage->data= OneDimData
    return outImage
```

2) Median Filter

```
MedianFilter(Image *inimage)
 begin
    tempData= the copy of original 2D matrix of the picture
    outImage= the copy of inimage
    for i=1 to Height-1 do    //Except the margin
        for j=1 to Width-1 do
            tempArr= values of 8 surrounding pixel in 3×3 matrix of tempData[i][j]
            median= the median of numbers in the tempArr
            tempData[i][j]=average
    OneDimData=change the tempData to 1 dimensional array by adding the data one by one
    outImage->data= OneDimData
    return outImage
```

Results (compare the results with the original image):

Paste the result images and the original ones.

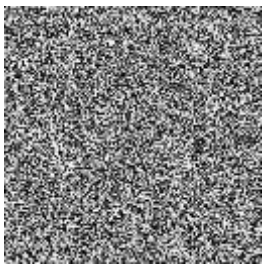
1) Lena.png

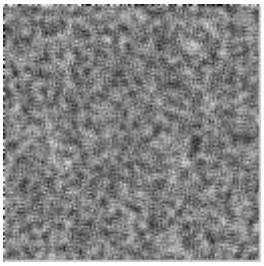

Before

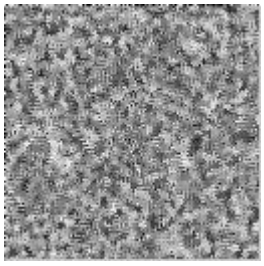

After Average Filter


After Median Filter

2) noise.pgm


Before


After Average Filter


After Median Filter

Discussion:

Write down your discovery about the test.

1) After calling ReadPNMImage() function, the data of the image is stored in a 1D array, we then change it to a 2D array is easier to conduct the following algorithm of 2 filters.
2) If we strictly follow the 3×3 form, the data in the margin cannot have 8 neighbors.
3) The result of the average filter seems dimmer than the result of the median filter.

Codes:

You don't need to paste all the codes. Just show the pieces of code that present the algorithm displayed above.

1) Average Filter

```
Image* AverageFilter(Image* image) {
    Image* outImage;
    int i, j, k, average;
    int count = 0;
    outImage = SwapImage(image);
    int tempArr[9];
    int tempData[300][300];

    GetArray(image);
    for (i = 1; i < image->Height - 1; i++) {
        for (j = 1; j < image->Width - 1; j++) {
            tempArr[0] = imaArr[i - 1][j - 1];
            tempArr[1] = imaArr[i - 1][j];
            tempArr[2] = imaArr[i - 1][j + 1];
            tempArr[3] = imaArr[i][j - 1];
            tempArr[4] = imaArr[i][j + 1];
            tempArr[5] = imaArr[i + 1][j - 1];
            tempArr[6] = imaArr[i + 1][j];
            tempArr[7] = imaArr[i + 1][j + 1];
            average = Average(tempArr, 8);
            tempData[i][j] = average;
        }
    }
    for (i = 0; i < image->Height; i++) {
        tempData[i][0] = imaArr[i][0];
        tempData[image->Width][0] = imaArr[image->Width][0];
    }
    for (i = 0; i < image->Width; i++) {
        tempData[0][i] = imaArr[0][i];
        tempData[0][image->Height] = imaArr[0][image->Height];
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            outImage->data[count] = tempData[i][j];
            count++;
        }
    }
    return outImage;
}
```

```
int Average(int num[], int m) {
    int sum = 0;
    int i ;
    for (i = 0; i < m; i++) {
        sum = sum + num[i];
    }
    return sum / m;
}
```

2) Median Filter

```
Image* MedianFilter(Image* image) {
    Image* outImage;
    int i, j, k, median;
    int count = 0;
    outImage = SwapImage(image);
    int tempArr[9];
    int tempData[300][300];
    GetArray(image);
    for (i = 1; i < image->Height-1; i++) {
        for (j = 1; j < image->Width - 1; j++) {
            tempArr[0] = imaArr[i - 1][j - 1];
            tempArr[1] = imaArr[i - 1][j];
            tempArr[2] = imaArr[i - 1][j + 1];
            tempArr[3] = imaArr[i][j - 1];
            tempArr[4] = imaArr[i ][j + 1];
            tempArr[5] = imaArr[i + 1][j - 1];
            tempArr[6] = imaArr[i + 1][j];
            tempArr[7] = imaArr[i + 1][j + 1];
            median = Median(tempArr, 8);
            tempData[i][j] = median;
        }
    }
    for (i = 0; i < image->Height; i++) {
        tempData[i][0] = imaArr[i][0];
        tempData[image->Width][0] = imaArr[image->Width][0];
    }
    for (i = 0; i < image->Width; i++) {
        tempData[0][i] = imaArr[0][i];
        tempData[0][image->Height] = imaArr[0][image->Height];
    }

    for (i = 0; i < image->Height; i++) {
        for (j = 0; j < image->Width; j++) {
            outImage->data[count] = tempData[i][j];
            count++;
        }
    }
    return outImage;
}
```

```
int Median(int num[], int m) {
    int i, j;
    int temp;
    for (i = 0; i < m; i++) {
        for (j = i + 1; j < m; j++) {
            if (num[j] < num[i]) {
                temp = num[i];
                num[i] = num[j];
                num[j] = temp;
            }
        }
    }
    return num[m / 2];
}
```