**Class Project 1**

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

1. 3x3 average filter **(lena, noise):**

**Algorithm:**

|  |  |  |
| --- | --- | --- |
| **(i-1,j-1)** | **(i-1,j)** | **(i-1,j+1)** |
| **(i,j-1)** | **(i,j)** | **(i,j+1)** |
| **(i+1,j-1)** | **(i+1,j)** | **(i+1,j+1)** |

**Calculate the average pixel in 3×3 window like the above from (1,1) and ignore the four edges.** **Replace (i,j) by the average pixel.**

(1.1)

(1.2)

(1.3)

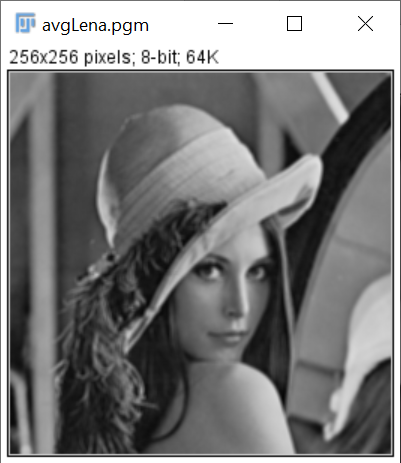
**Results (including pictures):**

Result of processing “Lena.pgm”:

Source Image:

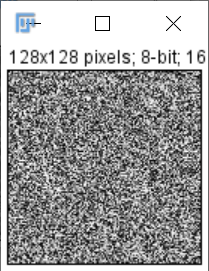


Result after average filter:

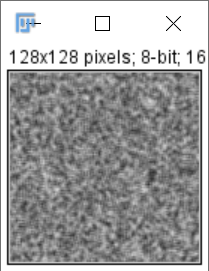


Result of processing “Noise.pgm”:

Source Image:



Result after average filter:



**Discussion:**

From the equation, we can conclude that differences between neighbor pixels become smaller by averaging, which leads to the smoother of image.

The result images do turn out to be smoother. For example, the edges between objects are less obvious.

**Codes:**

Image \*AverageFilt(Image \*image){

unsigned char \*tempin, \*tempout;

int size, i, j, k, t;

Image \*outimage;

outimage=CreateNewImage(image,"#testing Swap");

tempin=image->data;

tempout=outimage->data;

if(image->Type==GRAY) size = image->Width \* image->Height;

else if(image->Type==COLOR) size = image->Width \* image->Height \* 3;

for(i = 1; i < image->Height - 1; ++i){

for(j = 1; j < image->Width - 1; ++j){

int sum = 0;

for(k = -1; k < 2; ++k)

for(t = -1; t < 2; ++t)

sum += tempin[image->Width \* (i + k) + (j + t)];

tempout[image->Width \* i + j] = sum/9;

}

}

return(outimage);

}

1. 3x3 median filter **(lena, noise):**

**Algorithm:**

The principle is just same as the average filter. The only change is that replace (i,j) by the median pixel in the window. Use bubble sort to get the median.

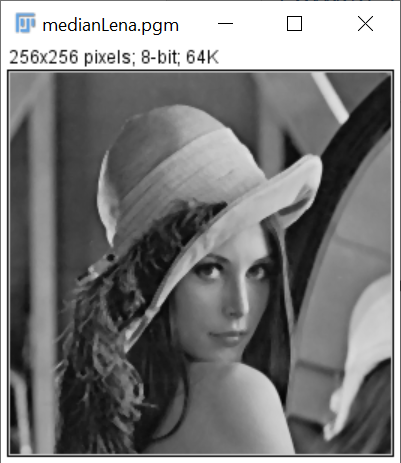
**Results (including pictures):**

Result of processing “Lena.pgm”:

Source Image:

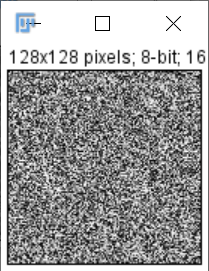


Result after median filter:

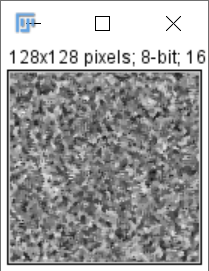


Result of processing “Noise.pgm”:

Source Image:



Result after median filter:



**Discussion:**

The median filter also smooths the input image. But the result of median filter is sharper than that of average filter. Median filter simply substitutes the target pixel with the median of its 3 x 3 neighbor, that is, the operation does not operate all pixels in the neighbor. So the differences between pixels in the output image by median filter is larger than that by average filter.

**Codes:**

Image \*MedianFilt(Image \*image){

unsigned char \*tempin, \*tempout, local[9];

int size, i, j, k, t;

Image \*outimage;

outimage=CreateNewImage(image,"#testing Swap");

tempin=image->data;

tempout=outimage->data;

if(image->Type==GRAY) size = image->Width \* image->Height;

else if(image->Type==COLOR) size = image->Width \* image->Height \* 3;

for(i = 1; i < image->Height - 1; ++i){

for(j = 1; j < image->Width - 1; ++j){

int pos = 0;

for(k = -1; k < 2; ++k)

for(t = -1; t < 2; ++t)

local[pos++] = tempin[image->Width \* (i + k) + (j + t)];

tempout[image->Width \* i + j] = findMedian(local, 9);

}

}

return(outimage);

}

unsigned char findMedian(unsigned char\* arr, int length){

unsigned char temp;

int i, j;

for(i = 0; i < length/2+1; ++i)

for(j = i+1; j < length; ++j)

if(arr[j] < arr[i]){

temp = arr[j];

arr[j] = arr[i];

arr[i] = temp;

}

return arr[length/2];

}