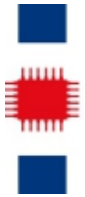
TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN



MICROELECTRONIC
SYSTEMS DESIGN
RESEARCH GROUP

MEDIAN FILTERING

ALGORITHM AND IMPLEMENTATION

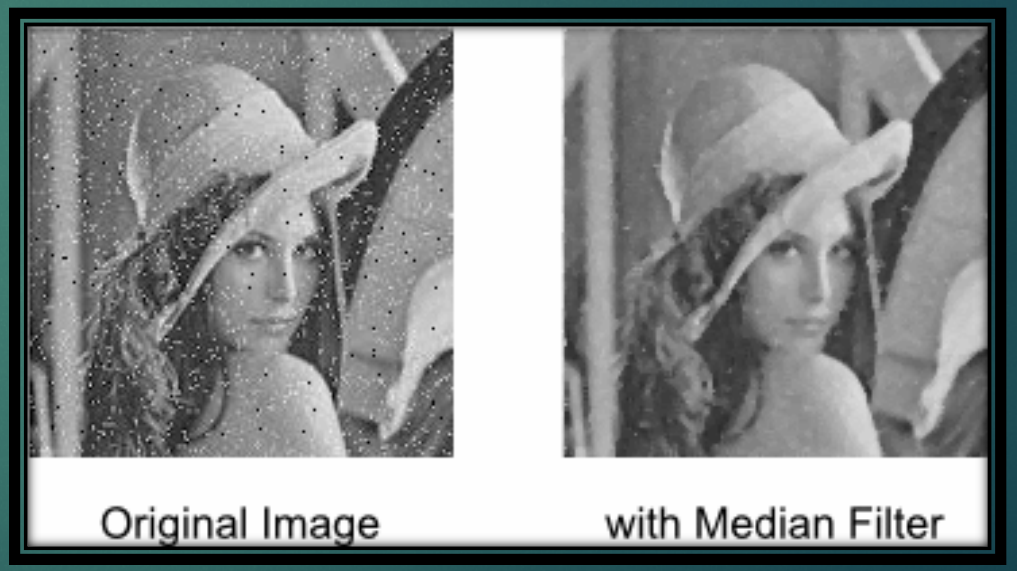
Hymalai Bello
2015-12-16

INTRODUCTION

- In the process of manipulation of images several types of filters are used, and have an important role in improving the characteristics of the image.

The median Filter

- Reduces the noise present in the image.



Median Filter Characteristics

- It is a nonlinear filtering technique.
- It can cope with the nonlinearities of the image and take into account the nonlinear nature of the human visual system.
- Good image detail preservation properties.
- Preserves edges while removing noise.
- It is often use to remove salt and pepper noise.
- It is easy to change the size of the filter.

Median Filter Algorithm

The algorithm basically consist of sorting the values in the image to find the median value. Filtering each pixel and its neighbors to replace the pixel value with the median of those values.

So, The basics step are:

- Selects a filter window (this means the neighborhood of the pixel) for the procedure. (3*3, 5*5, 7*7…etc)
- The Values from the surrounding neighborhood are first sorted into numerical order.
- The value of the pixel in question is replaced with the middle (median) pixel value. This is most likely to be a value of one of the pixels in the neighborhood within the window.

Median Filter Notes

- Zero values are placed to handle the missing window entries at the boundaries of the image.
- Important: The output is always constrained to be the median value in the window, so if the number of pixels is very large, then the median computed will be an impulse and the noise will not be removed.
- The center value replaced is not tested to find out if it is an impulse or not. Hence if it is not an impulse but a fine pixel of the image then it is removed unnecessarily.
- The median filter performs poorly when the intensity of the noise is high.

Median Filter Implementation

- A basic approach to get the median value is as follow:

```
uint16_t middle_of_3(uint16_t a,
uint16_t b, uint16_t c) {
        uint16_t middle;
        if ((a <= b) && (a <= c)) {
          middle = (b <= c) ? b : c;
        } else if ((b <= a) && (b <= c)) {
                middle = (a <= c) ? a : c;
         } else {
                 middle = (a <= b) ? a : b;
         }
        return middle;
 }
```

- The filter size of three is of course the smallest possible filter. It's possible to find the middle value simply via a few if statements. Clearly this is small and fast code.
- The main idea is that if we know the smallest number, then one comparison between the remaining two would give the second smallest and the largest number.

Median Filter Implementation qsort

```c
/* median through qsort example */
#include <stdio.h>
#include <stdlib.h>
 #define ELEMENTS 6
int values[] = { 40, 10, 100, 90, 20, 25 };
int compare (const void * a, const void * b) {
    return ( *(int*)a - *(int*)b );
}
int main () {
    int n;
    qsort (values, ELEMENTS, sizeof(int), compare);
    for (n=0; n<ELEMENTS; n++) {
      printf ("%d ",values[n]);
    }
    printf ("median=%d ",values[ELEMENTS/2]);
    return 0;
}
```

- Every C library comes with an implementation of a **quicksort** routine.
- Implementing the different size is a simple matter of "for" loops in the code.
- This standard routine is however usually very slow compared to the most recent methods, and overkill in the case of median search.

Median Filter Implementation Wirth

- It browses through the input array just enough to determine what is the Kth smallest element in the input list.
- It is not recursive and does not need to allocate any memory, nor that it use any external function.
- As a result, it gains a factor 25 in speed compared to the qsort() based method.
- It is needed an initial copy of the input array, because it is modified during the process.
- The median search is defined as a macro which finds the kth smallest element.
- It defines the median for an odd number of points as the one in the middle, and for even number the one below the middle.

```
typedef float elem_type ;
#define ELEM_SWAP(a,b) { register elem_type t=(a);(a)=(b);(b)=t;
}

elem_type kth_smallest(elem_type a[], int n, int k)
{
register i,j,l,m ;
register elem_type x ;
l=0 ; m=n-1 ;
while (l<m) {
x=a[k] ;
i=l ;
j=m ;
do {
while (a[i]<x) i++ ;
while (x<a[j]) j-- ;
if (i<=j) {
ELEM_SWAP(a[i],a[j]) ;
i++ ; j-- ;
}
} while (i<=j) ;
if (j<k) l=i ;
if (k<i) m=j ;
}
return a[k] ;
}
#define median(a,n) kth_smallest(a,n,(((n)&1)?((n)/2):(((n)/2)-1)))
```

Algorithm from N. Wirth's book, implementation by N. Devillard.
This code in public domain.

Median Filter Implementation Quick select

- It is a close tie with the Wirth. On the average this one is faster.
- It works in situ and modifies the input array. So the input date set must be copied prior to applying the medians search.

```
#define ELEM_SWAP(a,b) { register elem_type t=(a);(a)=(b);(b)=t; }
elem_type quick_select(elem_type arr[], int n)
{
int low, high ;
int median;
int middle, ll, hh;
low = 0 ; high = n-1 ; median = (low + high) / 2;
for (;;) {
if (high <= low) /* One element only */
return arr[median] ;
if (high == low + 1) { /* Two elements only */
if (arr[low] > arr[high])
ELEM_SWAP(arr[low], arr[high]) ;
return arr[median] ;
}
/* Find median of low, middle and high items; swap into position low */
middle = (low + high) / 2;
if (arr[middle] > arr[high]) ELEM_SWAP(arr[middle], arr[high]) ;
if (arr[low] > arr[high]) ELEM_SWAP(arr[low], arr[high]) ;
if (arr[middle] > arr[low]) ELEM_SWAP(arr[middle], arr[low]) ;
/* Swap low item (now in position middle) into position (low+1) */
ELEM_SWAP(arr[middle], arr[low+1]) ;
/* Nibble from each end towards middle, swapping items when stuck */
ll = low + 1;
hh = high;
for (;;) {
do ll++; while (arr[low] > arr[ll]) ;
do hh--; while (arr[hh] > arr[low]) ;
if (hh < ll)
break;
ELEM_SWAP(arr[ll], arr[hh]) ;
}
/* Swap middle item (in position low) back into correct position */
ELEM_SWAP(arr[low], arr[hh]) ;
/* Re-set active partition */
if (hh <= median)
low = 11;
if (hh >= median)
high = hh - 1;
}
}
#undef ELEM_SWAP
```

This Quickselect routine is based on the algorithm described in
"Numerical recipes in C", Second Edition,
Cambridge University Press, 1992, Section 8.5, ISBN 0-521-43108-5
This code by Nicolas Devillard - 1998. Public domain.

Median Filter Implementation Torben

- It is not the fastest way to find the median.
- It does not modify the input array when looking for the median.
- It is extremely powerful when the elements to consider starts to be large.

```
typedef float elem_type ;
elem_type torben(elem_type m[], int n)
{
int i, less, greater, equal;
elem_type min, max, guess, maxltguess, mingtguess;
min = max = m[0] ;
for (i=1 ; i<n ; i++) {
if (m[i]<min) min=m[i];
if (m[i]>max) max=m[i];
}
while (1) {
guess = (min+max)/2;
less = 0; greater = 0; equal = 0;
maxltguess = min ;
mingtguess = max ;
for (i=0; i<n; i++) {
if (m[i]<guess) {
less++;
if (m[i]>maxltguess) maxltguess = m[i] ;
} else if (m[i]>guess) {
greater++;
if (m[i]<mingtguess) mingtguess = m[i] ;
} else equal++;
}
if (less <= (n+1)/2 && greater <= (n+1)/2) break ;
else if (less>greater) max = maxltguess ;
else min = mingtguess;
}
if (less >= (n+1)/2) return maxltguess;
else if (less+equal >= (n+1)/2) return guess;
else return mingtguess;
}
```

Algorithm by Torben Mogensen, implementation by N. Devillard.
This code in public domain.

Bibliography

for further information please refer to:

- **Median Filter Performance Based on Different Window Sizes for Salt and Pepper Noise Removal in Gray and RGB Images,** Elmustafa S.Ali Ahmed1, Rasha E. A.Elatif2 and Zahra T.Alser3, 2005. http://www.sersc.org/journals/IJSIP/vol8_no10/34.pdf
- **Image Processing in C Second Edition,** Dwayne Phillips, April 2000. http://homepages.inf.ed.ac.uk/rbf/BOOKS/PHILLIPS/cips2ed.pdf
- https://www.youtube.com/watch?v=xFaddafLbcg_Basic and fast description of the median filter.
- http://embeddedgurus.com/stack-overflow/2010/10/median-filtering/ Simple Algorithm of a window 3*3
- http://ndevilla.free.fr/median/median.pdf Implementation of the median filter in ANSI C.