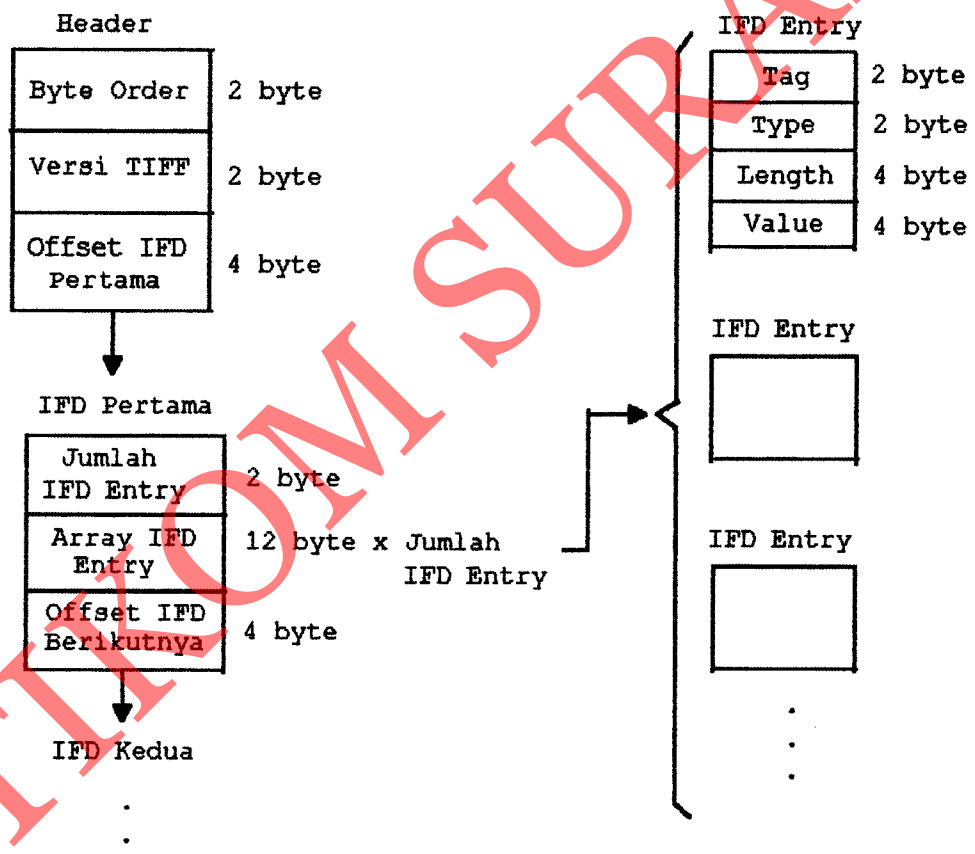
BAB IV

ANALISIS DAN PEMECAHAN MASALAH

4.1. Membaca File TIFF

Dalam file TIFF terdapat tiga bagian, yaitu Header, Image File Directory (IFD), dan Image File Directory Entry (IFD Entry). Hubungan antara ketiga bagian tersebut dapat dilihat pada gambar 4.1.



Gambar 4.1. Hubungan antara Header, IFD, dan IFD Entry

Langkah yang harus ditempuh untuk membaca file TIFF adalah :

- Membaca header.
- Membaca Image File Directory.
- Membaca Image File Directory Entry.
- Membaca data citra dan meletakkan dalam buffer

4.1.1. Membaca header

Langkah pertama yang harus dilakukan untuk membaca file TIFF adalah membaca header. Dua byte pertama menyatakan Byte Order, yaitu urutan byte yang digunakan pada saat membaca file TIFF. Bagian ini bertipe unsigned integer. Jika nilai yang diperoleh 0x4949 atau karakter II (Intel atau *little-endian*) maka urutan pembacaan byte dari Least Significant Byte (LSB) ke Most Significant Byte (MSB). Jika 0x4D4D atau karakter MM (Motorola atau *big-endian*), maka urutan pembacaan byte dari Most Significant Byte ke Least Significant Byte. Sedangkan apabila bagian tersebut tidak bernilai 0x4949 atau 0x4D4D maka file yang dibaca bukan merupakan file TIFF.

Dua byte berikutnya menyatakan versi TIFF. Pada umumnya bernilai 42 (0x002A atau 0x2A00, tergantung nilai yang diperoleh dari Byte Order, II atau MM).

Empat byte terakhir dari header menunjukkan offset ke IFD pertama. Bagian ini bertipe unsigned long. Nilai yang diperoleh merupakan lokasi IFD pertama sehingga pointer file harus diletakkan pada posisi tersebut sebelum membaca IFD.

4.1.2. Membaca IFD

Bagian pertama IFD sebesar dua byte, bertipe unsigned integer, dan menyatakan jumlah IFD Entry yang terdapat dalam IFD tersebut. Hasil yang diperoleh setelah membaca bagian ini dipakai untuk menentukan jumlah looping pada saat membaca IFD Entry.

Bagian kedua adalah array IFD Entry. Di sinilah letak IFD Entry. Tempat yang dibutuhkan sebesar jumlah IFD Entry dikalikan dengan 12 byte, karena satu IFD Entry sebesar 12 byte. Langkah selanjutnya adalah membaca IFD Entry.

4.1.3. Membaca IFD Entry

IFD Entry mempunyai empat bagian, terdiri dari Tag (2 byte), Type (2 byte), Length (4 byte), dan Value (4 byte).

Tahap pertama untuk membaca sebuah IFD Entry adalah membaca tag. Seperti yang sudah dijelaskan pada bab sebelumnya bahwa tag digunakan untuk mengidentifikasi

data citra yang terdapat dalam file TIFF yang bersangkutan. Misalnya nilai Tag sama dengan 256 berarti lebar citra dan nilai Tag sama dengan 257 berarti tinggi citra. Tetapi nilai 256 dan 257 ini bukan berarti bahwa lebar citra 256 piksel dan tinggi citra 257 piksel. Nilai tersebut hanya merupakan kode untuk menyatakan lebar citra dan tinggi citra.

Tahap berikutnya adalah membaca Type. Type menunjukkan tipe data dari masing-masing tag di atas. Contohnya tag 256 mempunyai nilai tipe short/long, tag 257 juga mempunyai tipe short/long. Macam-macam tag serta tipe data masing-masing bisa dilihat pada tabel 2.1.
Dalam tugas akhir ini didefinisikan sebagai berikut :

- Type = 1 artinya tipe data Byte.
- Type = 2 artinya tipe data ASCII.
- Type = 3 artinya tipe data Short.
- Type = 4 artinta tipe data Long.
- Type = 5 artinya tipe data Rasional.

Membaca empat byte berikutnya diperlukan untuk mendapatkan nilai Length, yaitu jumlah tipe data yang dimiliki oleh suatu tag. Misalnya tag suatu citra mempunyai nilai Type sama dengan empat (long) dan nilai Length sama dengan dua, artinya tag tersebut mempunyai data bertipe long sebanyak dua.

Empat byte terakhir dari IFD Entry menyatakan Value. Nilai yang didapatkan setelah membaca Value tersebut menyatakan informasi mengenai citra yang berada dalam file TIFF yang sedang dibaca. Misalnya nilai Tag sama dengan 256 dan nilai Value sama dengan 315, artinya file TIFF yang sedang dibaca mempunyai citra yang lebarnya 315 piksel. Proses membaca IFD Entry ini dilakukan sebanyak jumlah IFD Entry yang terdapat dalam IFD.

Empat byte terakhir dari IFD adalah offset ke IFD berikutnya. Jika bagian ini bernilai nol, berarti tidak ada IFD berikutnya, atau dengan kata lain dalam file TIFF tersebut hanya ada satu IFD. Pada umumnya satu file TIFF mempunyai sebuah IFD.

Proses membaca Header, IFD, maupun IFD Entry memerlukan suatu fungsi untuk membaca dua byte atau empat byte file TIFF .

Berikut ini terdapat dua buah fungsi untuk membaca dua byte atau empat byte file TIFF dengan urutan byte dari LSB ke MSB (II-Intel) maupun dari MSB ke LSB (MM-Motorola), (Rimmer, 1993).

```
Baca2Byte(fp)
   FILE *fp;
{
   if(ByteOrder == 'II')
      return((fgetc(fp)& 0xff)+((fgetc(fp)& 0xff)<<8));
   else return((fgetc(fp)& 0xff)<<8)+(fgetc(fp)& 0xff);
}


unsigned long Baca4Byte(fp)
   FILE *fp;
{
   if(ByteOrder == 'II')
     return((unsigned long)(fgetc(fp) & 0xff) +
            (unsigned long)(fgetc(fp) & 0xff) << 8 ) +
            (unsigned long)(fgetc(fp) & 0xff) << 16) +
            (unsigned long)(fgetc(fp) & 0xff) << 24))
   else
     return((unsigned long)(fgetc(fp) & 0xff) << 24) +
            (unsigned long)(fgetc(fp) & 0xff) << 16) +
            (unsigned long)(fgetc(fp) & 0xff) << 8 ) +
            (unsigned long)(fgetc(fp) & 0xff));
}
```

4.1.4. Membaca data citra

Setelah membaca Header, IFD, maupun IFD Entry, kemudian membaca data citra dari file TIFF. Sebelum ditampilkan ke layar, data citra perlu di masukkan buffer terlebih dahulu. Hal ini dilakukan untuk mempercepat proses menampilkan citra tersebut, karena data citra terdiri dari ribuan piksel sehingga proses untuk membaca piksel lebih cepat apabila data citra sudah berada di buffer dibandingkan dengan data citra yang masih berada dalam file TIFF.

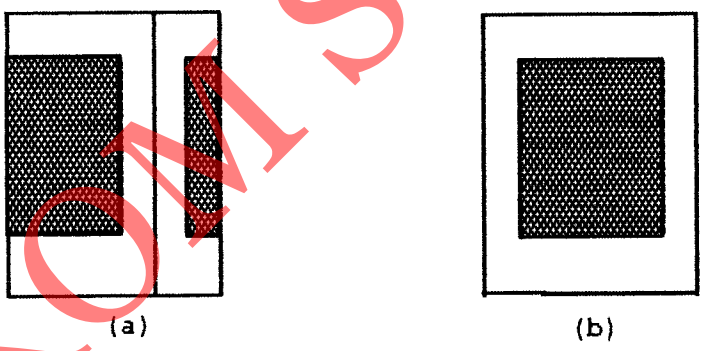
Sebelum meletakkan data citra ke dalam buffer harus mengalokasikan memori buffer sebesar data citra. Besarnya citra dapat diketahui dari nilai Value pada Tag 256 (ImageWidth) dan Tag 257 (ImageLength) yang menyatakan lebar dan tinggi citra dalam satuan piksel yang diperoleh pada saat membaca IFD Entry. Pada citra grayscale 256 graylevel satu piksel memerlukan memori sebesar satu byte, sehingga besarnya citra dalam satuan byte adalah lebar kali tinggi citra. Alokasi memori buffernya sebagai berikut :

```
unsigned char *buffer;
unsigned int LebarCitra,TinggiCitra;
unsigned long CitraSize;
                .
                .
CitraSize = (long)LebarCitra x (long)TinggiCitra;
buffer = (unsigned char *)farmalloc(CitraSize);
```

Pada saat membaca IFD Entry terdapat Tag yang menyatakan StripOffset (Tag=273) yaitu mendefinisikan alamat masing-masing strip terhadap awal file. Nilai Value pada tag inilah letak data citra. Citra TIFF bisa mempunyai strip lebih dari satu. Jumlah strip dapat diketahui dari nilai Length pada Tag sama dengan 273 ketika membaca IFD Entry. Apabila nilai Length sama dengan satu maka citra yang dibaca mempunyai strip sebanyak satu. Begitu juga bila nilai Length sama dengan

dua maka jumlah strip sebanyak dua. Jumlah strip tidak dipengaruhi oleh besar kecilnya citra. Citra yang besar belum tentu mempunyai strip lebih dari satu, begitu juga sebaliknya, citra yang kecil bisa mempunyai strip lebih dari satu.

Ketika proses membaca data citra akan dilakukan, pointer harus menunjuk pada alamat strip. Jika suatu citra mempunyai strip lebih dari satu maka sebelum membaca data citra pointer harus diletakkan pada strip pertama. Kalau hal ini tidak dilakukan citra yang ditampilkan akan tampak terpotong secara vertikal seperti pada gambar 4.2.(a).



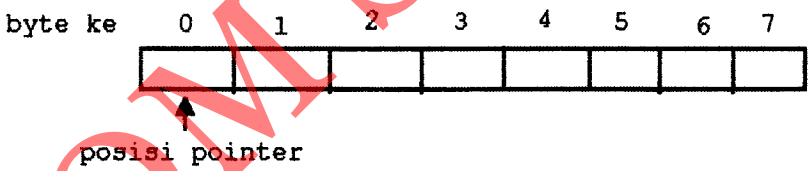
Gambar 4.2. (a) Citra terpotong, (b) Citra asli

Proses membaca data citra memerlukan pointer yang menunjuk ke buffer. Berikut ini terdapat suatu fungsi yang berfungsi sebagai pointer buffer, (Rimmer, 1993) :

```
char *farPtr(p,l)
     unsigned char *p;
     long l;
{
     unsigned int seg,off;

     seg = FP_SEG(p);
     off = FP_OFF(p);
     seg += (off/16);
     off = &= 0x000f;
     off += (unsigned int)(l & 0x000fL);
     seg += (l/16L);
     p = (unsigned char *)MK_FP(seg,off);
     return(p);

}
```

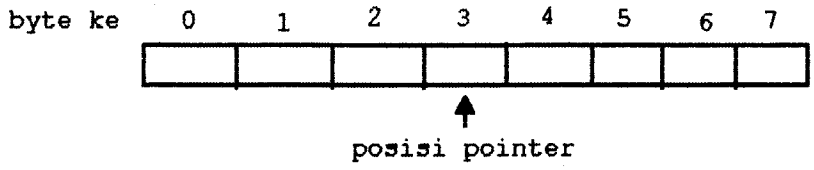
Parameter *p* pada fungsi di atas adalah buffer, sedangkan *l* adalah jumlah langkah (byte) yang harus ditempuh oleh pointer di buffer.

Contoh :



Gambar 4.3. Posisi awal pointer buffer

Setelah instruksi : p = farPtr(p,3L); maka posisi pointer seperti pada gambar 4.4.



Gambar 4.4. Posisi pointer setelah instruksi
p = farPtr(p,1L);

Pada saat pointer file sudah berada di posisi di mana data citra diletakkan, maka pembacaan data citra untuk dimasukkan ke buffer siap dilakukan.

Proses membaca data citra dapat dilakukan tiap line sehingga diperlukan proses looping sebanyak tinggi citra. Itu berarti pointer buffer harus naik sebanyak lebar citra.

```
FILE *fp;              /* file TIFF */
unsigned char *p;      /* pointer buffer */
        .
        .

for(i=0;i<TinggiCitra;++i)    {
   fread(p,1,LebarCitra,fp);
   p = farPtr(p,(long)LebarCitra);
}
```

Data citra yang sudah berada di buffer dapat digunakan untuk proses selanjutnya, misalnya ditampilkan ke layar, proses filtering, atau proses deteksi batas tepi.

4.2. Menampilkan Citra ke Layar

Proses pengolahan citra tidak lepas dari proses untuk menampilkan citra ke layar. Tujuannya untuk mengetahui hasil pemrosesan tadi. Sebelum ditampilkan, data citra sudah berada dalam buffer.

Tahapan untuk menampilkan gambar di layar :

- Setting mode video.
- Mengubah default warna menjadi grayscale.
- Membaca dan menampilkan data citra.

4.2.1. Setting mode video

Sebelum menampilkan citra ke layar, mode video harus disesuaikan terlebih dahulu, yaitu diubah ke dalam mode grafik. Pada mode grafik juga terdiri dari bermacam-macam mode. Hal ini dapat dilihat pada tabel mode video di bagian lampiran.

Untuk mengubah mode video digunakan interrupt 10H ROM BIOS Video Service dengan memberikan nilai 00H pada register AH dan nilai mode video yang diinginkan pada register AL (lihat tabel 2.2).

Dalam hal ini mode video yang digunakan adalah mode 13H karena citra yang akan ditampilkan adalah citra grayscale 256 graylevel. Sedangkan mode 13H adalah mode video yang mampu menampilkan warna sebanyak 256 warna.

Fungsi untuk mengubah mode video menjadi mode 13H dapat dituliskan sebagai berikut :

```
ModeVideo()
{
    union REGS reg;

    reg.h.ah = 0x00;
    reg.h.al = 0x13;
    int86(0x10,&r,&r);
}
```

Instruksi reg.h.ah = 0x00; dan reg.h.al = 0x13; dapat dijadikan satu menjadi reg.x.ax = 0x0013;.

4.2.2. Mengubah default warna menjadi grayscale

Mode video 13H mampu menampilkan warna sebanyak 256 warna, mulai warna nomor 0 (nol) sampai dengan nomor 255. Apabila warna nomor nol sampai dengan 255 tersebut ditampilkan ke layar, akan tampak warna merah, hijau, kuning, biru, dan sebagainya, termasuk juga 16 warna graylevel.

Warna-warna tadi merupakan perpaduan dari tiga warna primer, yaitu merah, hijau, dan biru (Red, Green, Blue) yang masing-masing mempunyai nilai intensitas sebesar enam bit, sehingga setiap warna primer mempunyai nilai intensitas antara 0 (nol) sampai dengan 63.

Untuk mengetahui nilai intensitas masing-masing warna primer dari suatu warna dapat digunakan INT 10H ROM BIOS Video Service dengan nilai register AH sama dengan 10H, register AL sama dengan 15H, dan register BX diisi dengan nomor warna yang ingin diketahui nilai intensitas warna primernya (0 sampai dengan 255). Nilai yang dihasilkan oleh register CH, CL, dan DH berturut-turut menunjukkan nilai intensitas Green, Blue, dan Red (lihat tabel 2.6).

Pada 256 warna di atas terdapat 16 warna grayscale, yaitu warna nomor 16 sampai dengan nomor 31. Apabila nilai intensitas warna primer dari 256 warna tadi ditampilkan, akan tampak bahwa warna grayscale mempunyai nilai intensitas warna primer yang unik dibandingkan dengan warna lain. Warna grayscale mempunyai nilai intensitas yang besarnya sama pada ketiga warna primer tadi. Sedangkan pada warna lain nilai intensitas ketiga warna primernya berbeda atau hanya ada dua warna primer yang nilai intensitasnya sama.

Citra yang akan ditampilkan adalah citra grayscale 256 graylevel, sehingga 256 warna tadi harus diubah menjadi warna grayscale 256 graylevel.

Mengubah 256 warna menjadi 256 graylevel berarti harus mengubah nilai intensitas warna primer pada tiap

warna. Untuk mengatasi hal itu digunakan INT 10H ROM BIOS Video Service dengan nilai register AH sama dengan 10H, register AL sama dengan 10H, dan register BX berisi nomor warna (0 - 255). Nilai intensitas warna primer Red, Green, dan Blue berturut-turut diletakkan pada register DH, register CH, dan register CL (lihat tabel 2.5.).

Ketiga warna primer tadi harus mempunyai nilai intensitas yang sama. Misalnya warna nomor nol mempunyai nilai intensitas Red sama dengan nol, Green sama dengan nol, dan Blue sama dengan nol. Warna nomor empat dengan nilai intensitas Red sama dengan satu, Green sama dengan satu, dan Blue sama dengan satu, dan seterusnya. Seperti yang sudah disebutkan di atas bahwa besarnya intensitas tiap warna primer adalah enam bit, yaitu nilai intensitas nol sampai dengan 63. Kalau satu warna grayscale mempunyai nilai intensitas yang sama pada ketiga warna primernya, berarti hanya ada 64 graylevel. Padahal yang dibutuhkan adalah 256 graylevel. Untuk mengatasi hal itu dibuat suatu skala, caranya 256 dibagi 64 sama dengan eempat. Berarti setiap empat warna mempunyai nilai intensitas warna primer yang besarnya sama, yaitu :

warna nomor 0 - 3 --> Red=0, Green=0, Blue=0

warna nomor 4 - 7 --> Red=1, Green=1, Blue=1

warna nomor 8 - 11 --> Red=2, Green=2, Blue=2

dan seterusnya sampai dengan :

warna nomor 252 - 255 --> Red=63, Green=63, Blue=63.

Berikut ini terdapat fungsi untuk mengubah 256 warna menjadi 256 graylevel :

```
Setgrey()
{
     union REGS reg;
     int i,warna,n=0;

     for(i=0;i<256;++i)
     {
          reg.x.ax = 0x1010;
          reg.x.bx = i;
          warna = n/4;
          reg.h.ch = warna;
          reg.h.cl = warna;
          reg.h.dh = warna;
          int86(0x10,&reg,&reg);

          n += 1;
     }
}
```

4.2.3. Membaca dan menampilkan data citra

Sebelum ditampilkan ke layar, data citra sudah berada dalam buffer. Sehingga pada saat menampilkan citra, yang dibaca adalah data citra dalam buffer.

Perlu diketahui bahwa pada mode video 13H, satu layar dapat memuat piksel sebanyak 320 kolom dan 200 baris. Piksel di pojok kiri atas berada pada kolom nol

dan baris nol, sedangkan piksel di pojok kanan bawah berada pada kolom 319 baris 199.

Untuk menuliskan piksel pada layar digunakan INT 10H ROM BIOS Video Service dengan nilai register AH sama dengan 0CH. Warna piksel yang akan dituliskan diletakkan pada register AL. Register BH menyatakan halaman tampilan (display page). Posisi baris dan kolom pada layar berturut-turut diletakkan pada register DX dan CX (lihat tabel 2.3.).

Berikut ini fungsi untuk menuliskan piksel pada layar :

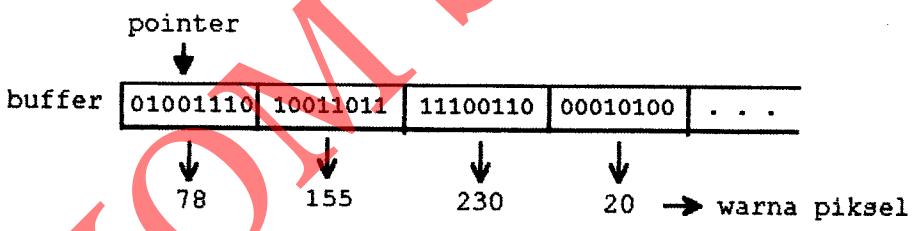
```
TulisPiksel(kolom,baris,warna)
     int kolom,baris;
     unsigned char warna;
{
     union REGS reg;

     reg.h.ah = 0x0C;
     reg.h.al = warna;
     reg.h.bh = 0;
     reg.x.dx = baris;
     reg.x.cx = kolom;
     int86(0x10,&reg,&reg);
}
```

Di bawah ini adalah potongan program untuk menampilkan citra ke layar :

```
unsigned char *buffer,warna;
int baris,kolom;
          .
          .

for(baris=0;baris<TinggiCitra;++baris)
     for(kolom=0;kolom<LebarCitra;++kolom)
```

```
        {
            warna = *buffer;
            TulisPiksel(kolom,baris,warna);
            buffer = farPtr(buffer,1L);
        }
    }
```

Sebelum menampilkan citra ke layar, pointer diletakkan pada posisi awal buffer. Satu byte data citra yang ditunjuk oleh pointer tersebut adalah data citra yang ditampilkan ke layar yang sebetulnya menyatakan warna piksel. Setelah satu piksel ditampilkan ke layar pada baris dan kolom yang sudah ditentukan, pointer buffer dinaikkan untuk membaca dan menampilkan data citra berikutnya. Hal ini diulangi sampai seluruh data citra ditampilkan.



Gambar 4.5. Data citra dalam buffer

4.3. Noise

Menciptakan noise pada tugas akhir ini diperlukan karena untuk mendapatkan citra yang mempunyai noise akibat gangguan pada saat perekaman data sulit didapatkan.

Noise tersebut dibangkitkan dengan distribusi tertentu dari distribusi Uniform(0,1) dengan menggunakan metode Transformasi Invers, (M.Law, 1991).
Macam distribusi yang digunakan di sini adalah distribusi Uniform, distribusi Eksponensial, dan distribusi Gaussian.

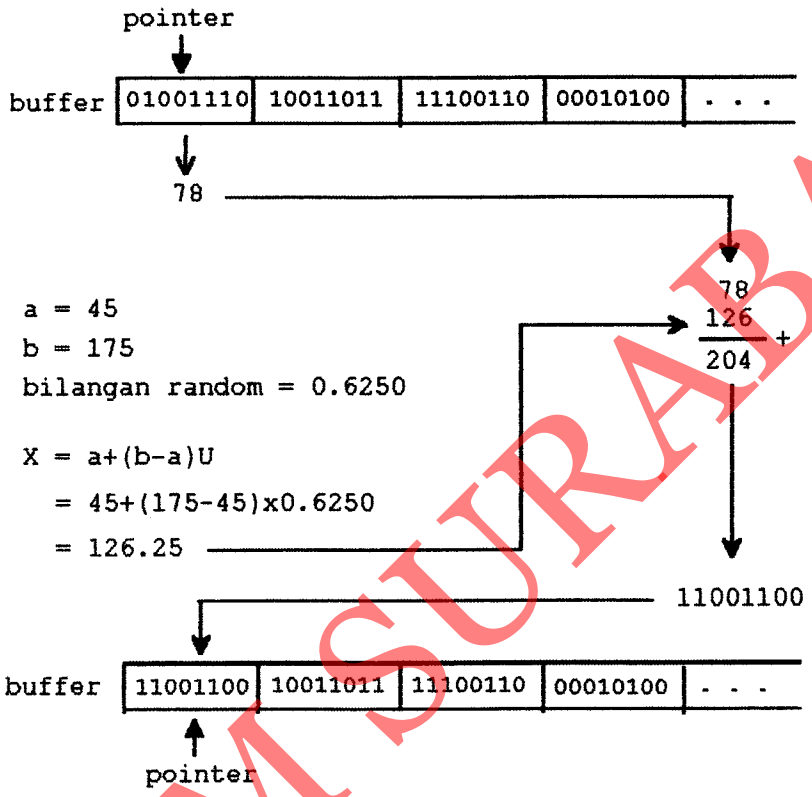
Ada juga noise yang disebut dengan salt and pepper noise. noise ini berbentuk bercak putih (salt) dan hitam (pepper) yang tersebar secara acak pada citra.

4.3.1. Noise berdistribusi Uniform

Langkah untuk membangkitkan noise dengan distribusi Uniform adalah :

- Letakkan pointer di posisi awal buffer.
- Bangkitkan bilangan acak antara 0 dan 1.
  (Generate U ~ U(0,1)).
- Hitung X = a + (b - a)U, dimana a adalah batas atas dan b adalah batas bawah (lihat gambar 2.7.) sedangkan U adalah bilangan acak antara 0 dan 1.
- Tambahkan nilai X pada isi buffer yang ditunjuk oleh pointer dan letakkan hasilnya pada posisi buffer tersebut.
- Jika hasil penjumlahan itu lebih besar dari 255, maka nilai yang dimasukkan buffer adalah 255.

- Naikkan pointer buffer pada posisi berikutnya.
- Ulangi semua langkah di atas sebanyak jumlah piksel dalam citra.



Gambar 4.6. Pembangkit noise berdistribusi Uniform

Pada gambar 4.6 tampak suatu distribusi Uniform mempunyai nilai batas atas *a* sama dengan 45 dan batas bawah *b* sama dengan 175. Bilangan random U yang diperoleh dari membangkitkan bilangan acak antara nol sampai dengan satu adalah 0.6250. Setelah dihitung dengan menggunakan rumus untuk membangkitkan noise yang berdistribusi

Uniform (a+(b-a)U) diperoleh nilai sebesar 126.25. Bilangan 126.25 tersebut didefinisikan pada suatu variabel yang bernilai integer sehingga diperoleh nilai sebesar 126. Pada saat itu pointer buffer menunjuk pada suatu alamat berisi data piksel yang mempunyai tingkat keabuan 78 dan bertipe unsigned character. Tambahkan 78 dengan 126, maka akan diperoleh nilai 204. 204 merupakan nilai piksel yang baru, yaitu piksel yang mempunyai noise berdistribusi uniform.

4.3.2. Noise berdistribusi Eksponensial

Langkah untuk membangkitkan noise dengan distribusi Eksponensial adalah :

- Letakkan pointer di posisi awal buffer.
- Bangkitkan bilangan acak antara 0 dan 1.
  (Generate U ~ U(0,1)).
- Jika U > 0, X = $-\beta$ ln U, sedangkan jika U < 1, X = 0;
- Tambahkan nilai X pada isi buffer yang ditunjuk oleh pointer dan letakkan hasilnya pada posisi buffer tersebut.
- Jika hasil penjumlahan itu lebih besar dari 255, maka nilai yang dimasukkan buffer adalah 255.

- Naikkan pointer buffer pada posisi berikutnya.
- Ulangi semua langkah di atas sebanyak jumlah piksel dalam citra.

4.3.3. Noise berdistribusi Gaussian

Langkah untuk membangkitkan noise dengan distribusi Gaussian adalah :

a. Letakkan pointer di posisi awal buffer.

b. Bangkitkan bilangan acak antara 0 dan 1.
(Generate $U_1 \sim U(0,1)$ dan $U_2 \sim U(0,1)$).

c. Hitung $V_1 = 2U_1-1$, $V_2 = 2U_2-1$ dan $W = V_1^2 + V_2^2$.

d. Jika W > 1 kembali ke langkah (b).
Jika W <= 1 ke langkah (e).

e. hitung $Y = ((-2 \ln W)/W)^{1/2}$

$$X_1 = V_1 Y(\sigma^2)^{1/2} + \mu$$

$$X_2 = V_2 Y(\sigma^2)^{1/2} + \mu$$

dimana σ^2 adalah varian dan μ adalah mean.

f. Tambahkan nilai X_1 atau X_2 pada isi buffer yang ditunjuk oleh pointer dan letakkan hasilnya pada posisi buffer tersebut.

g. Jika hasil penjumlahan itu lebih besar dari 255, maka nilai yang dimasukkan buffer adalah 255, sedangkan

jika lebih kecil dari nol nilai yang dimasukkan buffer adalah nol.

h. Naikkan pointer buffer pada posisi berikutnya.

i. Ulangi semua langkah di atas sebanyak jumlah piksel dalam citra.

4.3.4. Salt and pepper noise

Salt and pepper noise adalah gangguan yang seringkali ditemui pada suatu citra. Noise ini tampak sebagai bercak hitam dan/atau putih yang menyebar secara acak pada seluruh bagian citra.

Salt noise berupa bercak putih dengan nilai intensitas 255, sedangkan pepper noise berupa bercak hitam dengan nilai intensitas nol.

Berikut ini adalah fungsi untuk membangkitkan salt and pepper noise, (R. Weeks, 1993) :

```
SaltPepper(buffer,probabilitas)
    unsigned char *buffer;
    float probabilitas;
{
    int i,data,data1,data2;

    data  = (int)(probabilitas * 32768 / 2);
    data1 = data + 16384;
    data2 = 16384 - data;
```

```
        for(i=0;i<CitraSize;++i)
        {
            data = rand();
            if(data >= 16384 && data < data1)
                *buffer = 0;
            if(data >= data2 && data < 16384)
                *buffer = 255;
            buffer = farPtr(buffer,1L);
        }
    }
```

4.4. Filtering

Filtering adalah salah satu proses yang terdapat pada pengolahan citra untuk memperbaiki citra atau meningkatkan mutu citra. Proses filtering digunakan untuk menghilangkan gangguan (noise) yang terdapat pada suatu citra. Gangguan tersebut dapat berupa garis-garis, titik-titik hitam atau putih yang tersebar secara acak pada citra.

Metode filtering sendiri terdiri dari beberapa macam, antara lain Adaptive filter, Morphological filter, Nonlinear filter, dan Spatial filter. Masing-masing juga mempunyai beberapa metode seperti di bawah ini.

Adaptive filter :

- Double Window Modified Trimmed Mean (DW-MTM) filter
- Minimum Mean-Square Error (MMSE) filter

Morphological filter :

- Erosion filter
- Dilation filter

- Opening filter
- Closing filter

Nonlinear filter :

- Alpha-Trimmed Mean filter
- Contra-Harmonic filter
- Geometric Mean filter
- Harmonic Mean filter
- Maximum filter
- Median filter
- Midpoint filter
- Minimum filter
- Weighted Median filter
- Yp Mean filter

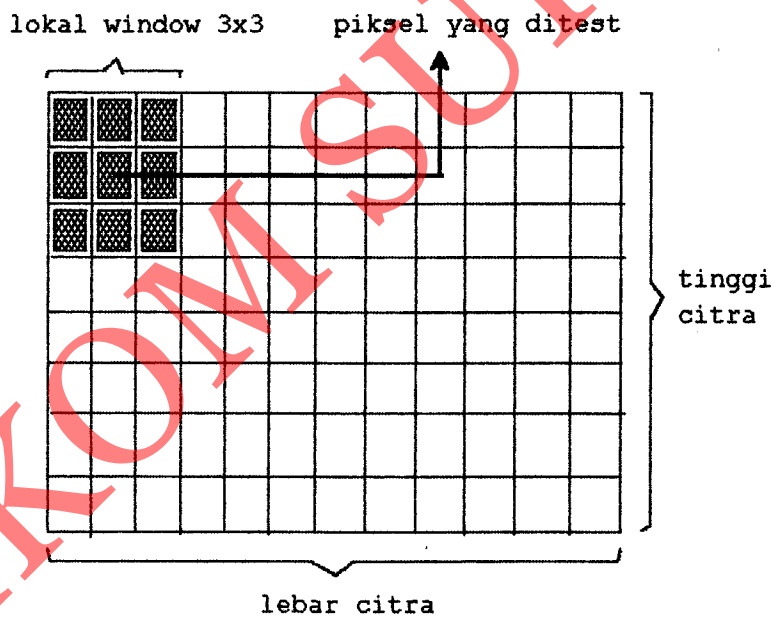
Spatial filter :

- Arithmetic Mean filter
- Gaussian filter
- Haigh Pass filter
- Low Pass filter
- Weighted Mean filter

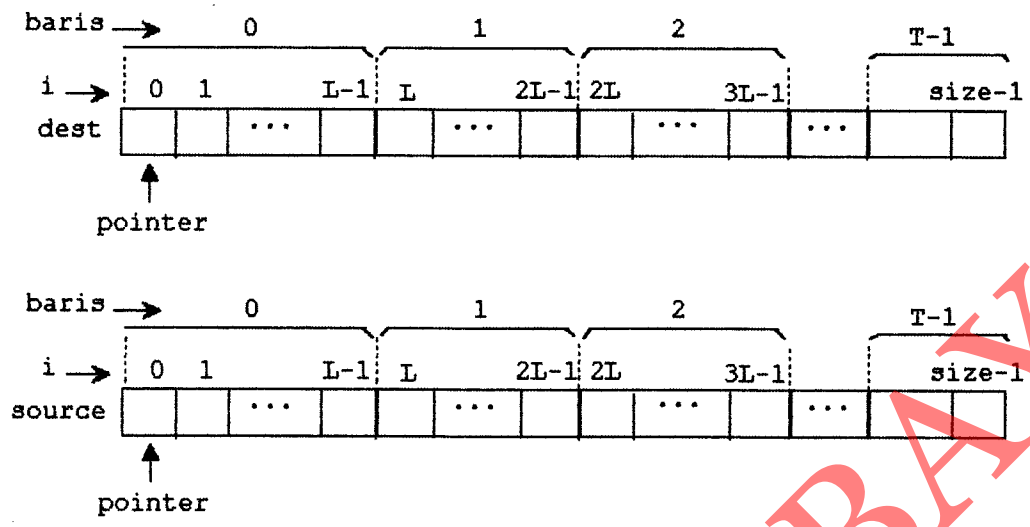
Proses filtering memerlukan dua buah buffer. Buffer pertama (buffer asal) berisi data citra asli, sedangkan buffer kedua (buffer tujuan) berisi data citra hasil proses filtering. Hal ini dilakukan karena operasi

filtering terhadap masing-masing piksel harus menggunakan nilai piksel asli, bukan nilai piksel yang sudah berubah hasil operasi filtering.

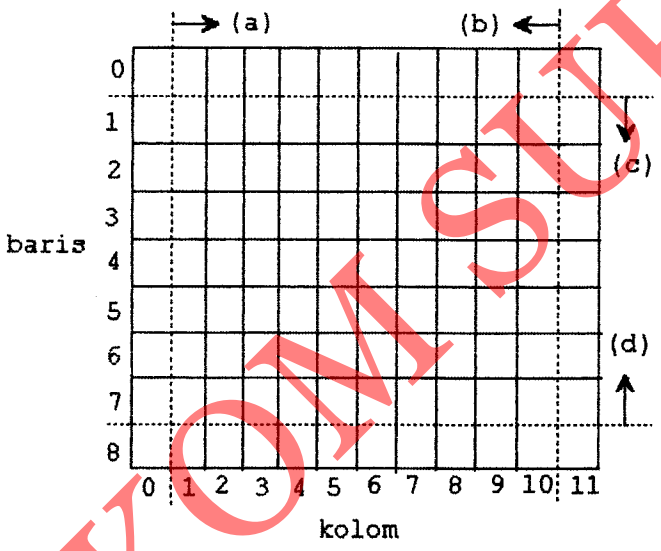
Pada proses filtering digunakan istilah lokal window. Ukuran lokal window pada umumnya berupa bilangan ganjil, misalnya 3x3, 5x5, 9x9, atau 11x11. Piksel yang berada di tengah lokal window adalah piksel yang ditest, yaitu suatu piksel yang nilainya akan diganti dengan nilai hasil operasi terhadap lokal window. Untuk lebih jelasnya dapat dilihat pada gambar 4.7.



Gambar 4.7. Lokal window dalam citra dijital



misal : wind = 3



wind : ukuran lokal window
L : lebar citra
T : tinggi citra
Size : ukuran citra (L x T)
i : nomor byte (i=0,1,2,...,size-1)

kondisi (a). i >= baris*lebar+(wind/2)
kondisi (b). i <= (baris+1)*lebar-((wind/2)+1)
kondisi (c). i > (wind/2)*lebar
kondisi (d). i < size-(wind/2)*lebar

Gambar 4.8. Kondisi untuk menentukan daerah operasi filtering

Pada proses filtering maupun deteksi batas tepi tidak semua piksel dalam citra nilainya diganti dengan nilai hasil operasi (ditest). Bila ukuran lokal window 3x3 maka piksel yang berada pada kolom pertama, kolom terakhir, baris pertama, dan baris terakhir dalam citra tidak mengalami perubahan. Sedangkan jika ukuran lokal window 5x5 maka piksel di posisi dua kolom pertama, dua kolom terakhir, dua baris pertama, dan dua baris terakhir dalam citra tidak mengalami perubahan, sehingga apabila ukuran lokal window NxN maka piksel di posisi N/2 - 1/2 kolom pertama, N/2 - 1/2 kolom terakhir, N/2 - 1/2 baris pertama, dan N/2 - 1/2 baris terakhir tidak mengalami perubahan.

Jika pointer buffer asal menunjuk pada piksel di daerah yang tidak memenuhi kondisi (a), (b), (c), dan (d) maka nilai piksel tersebut langsung dimasukkan pada buffer tujuan. Jika pointer buffer asal menunjuk pada piksel di daerah yang memenuhi keempat kondisi di atas maka dilakukan operasi filtering sesuai dengan metode yang digunakan kemudian nilai yang dihasilkan dimasukkan dalam buffer tujuan. Lihat gambar 4.8.

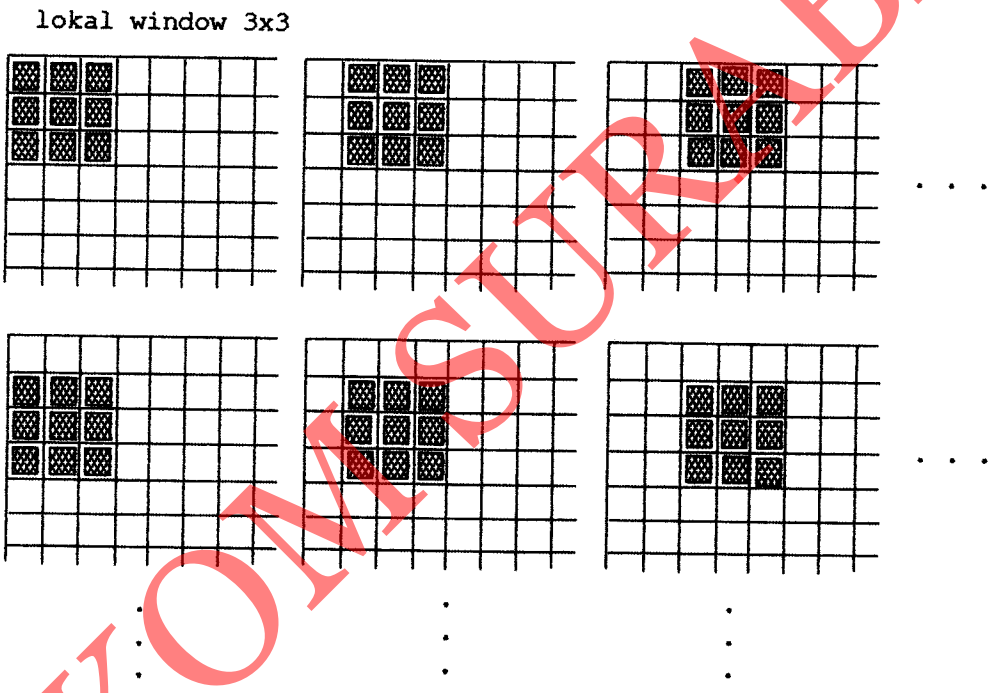
Berikut ini terdapat potongan program untuk kondisi seperti di atas :

```
                .
                .
unsigned char *source,*dest;
int i,baris,kolom,wind,tinggi,lebar;
                .
                .
i = 0;

for(baris=0;baris<tinggi;++baris)
  for(kolom=0;kolom<lebar;++kolom)
  {
    if( (i>(wind/2)*lebar)&&
        (i>=baris*lebar+(wind/2))&&
        (i<=(baris+1)*lebar-((wind/2)+1))&&
        (i<size-(wind/2)*lebar) )
    {
                .
                .
                .
        operasi filtering dengan metode tertentu
                .
                .
    }
    else *dest = *source;

    source = farPtr(source,1L);
    dest = farPtr(dest,1L);
    i++;
}
```

Keterangan :

source	=	buffer asal
dest	=	buffer tujuan
wind	=	ukuran lokal window
tinggi	=	tinggi citra
lebar	=	lebar citra
size	=	ukuran citra (lebar x tinggi)
i	=	nomor byte dalam citra (byte ke 0,1,2,3,...,size-1)

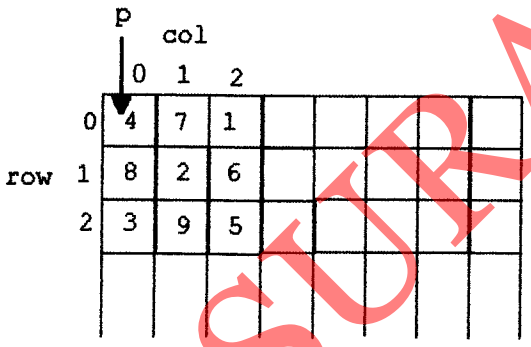
Lokal window diperlukan dalam proses filtering, oleh sebab itu harus ditentukan bahwa piksel mana saja yang harus berada dalam lokal window untuk dilakukan operasi filterig. Setelah operasi terhadap suatu lokal window dilakukan, maka dilanjutkan dengan lokal window berikutnya sampai dengan seluruh lokal window dalam citra. Lihat gambar 4.9.



Gambar 4.9. Perubahan posisi lokal window dalam operasi filtering

Potongan program di bawah ini bertujuan untuk menentukan piksel yang harus dimasukkan dalam lokal window atau dengan kata lain menentukan isi lokal window.

```
unsigned char *p;
int seg,off,n;

seg = FP_SEG(buffer);
off = FP_OFF(buffer);
p = MK_FP(seg,off);

n = 0;

for(row=0;row<wind;++row)
 for(col=0;col<wind;++col)
 {
   lok_wind[n] = *(farPtr(p,(long)(row*LebarCitra+col)));
   n++;
 }
```



Gambar 4.10. Lokal window 3x3

Pada gambr 4.10. terdapat suatu lokal window 3x3. Masing-masing elemen lokal window di atas dimasukkan dalam array satu dimensi. Misalkan lebar citra adalah delapan piksel maka :

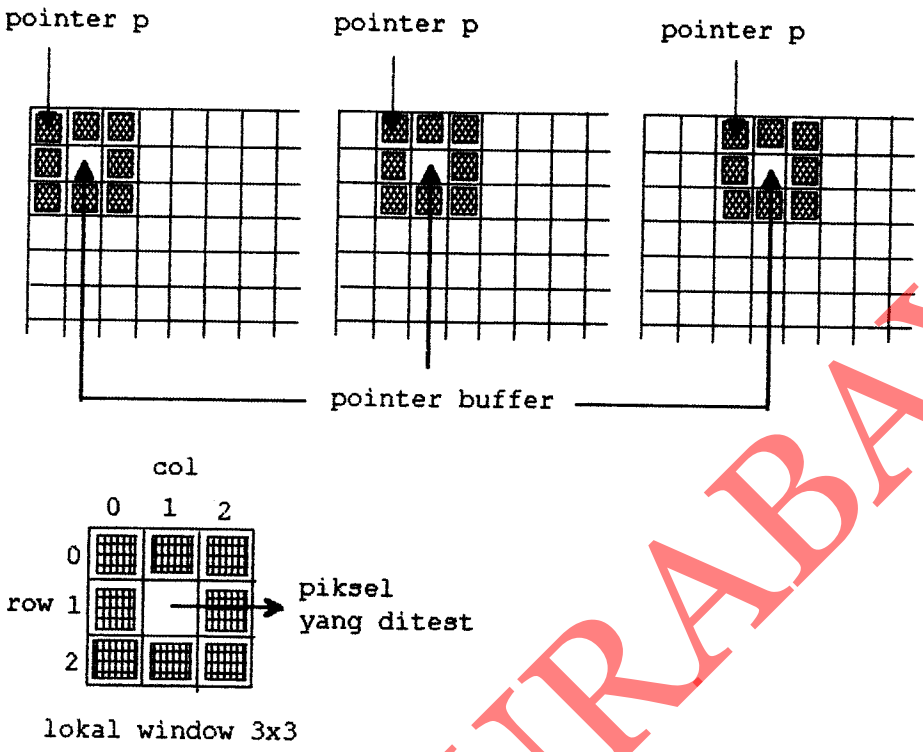
```
lok_wind[0] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(0*8+0)))
            = *(farPtr(p,0L))
            = 4
```

```
lok_wind[1] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(0*8+1)))
            = *(farPtr(p,1L))
            = 7

lok_wind[2] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(0*8+2)))
            = *(farPtr(p,2L))
            = 1

lok_wind[3] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(1*8+0)))
            = *(farPtr(p,8L))
            = 8

lok_wind[4] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(1*8+1)))
            = *(farPtr(p,9L))
            = 2

lok_wind[5] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(1*8+2)))
            = *(farPtr(p,10L))
            = 6
```

```
lok_wind[6] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(2*8+0)))
            = *(farPtr(p,16L))
            = 3

lok_wind[7] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(2*8+1)))
            = *(farPtr(p,17L))
            = 9

lok_wind[8] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(2*8+2)))
            = *(farPtr(p,18L))
            = 5
```

p adalah pointer yang menunjuk pada buffer yang berisi citra asli dengan posisi awal berada pada awal buffer. Pointer p akan menunjuk pada byte berikutnya bila i>=(wind/2)*LebarCitra+(wind/2). *Wind* adalah ukuran lokal window dan *i* adalah nomor byte yang mempunyai nilai nol sampai dengan CitraSize-1. Lihat gambar 4.11.

Masing-masing piksel dalam lokal window dimasukkan dalam suatu array satu dimensi untuk selanjutnya dilakukan operasi filtering sesuai dengan metode yang digunakan.



Gambar 4.11. Menentukan lokal window

4.4.1. Adaptive filter

Adaptive filter terdiri dari dua metode, yaitu :

- Double Window Modified Trimmed Mean (DW-MTM) filter.
- Minimum Mean-Square Error (MMSE) filter.

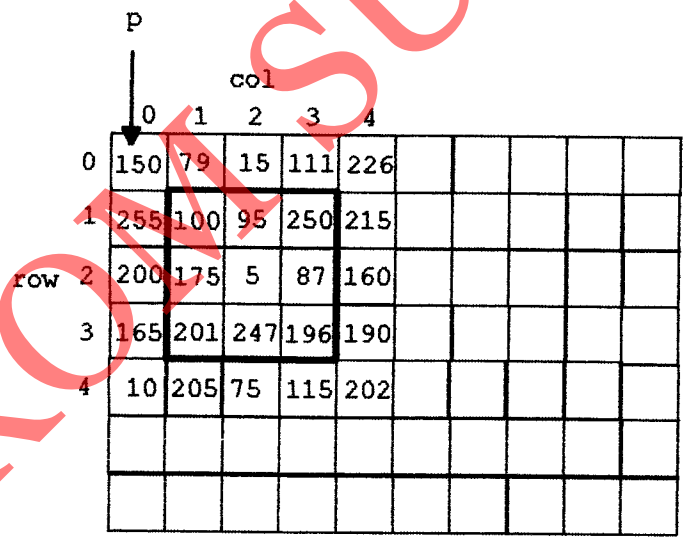
A. DW-MTM Filter

Operasi DW-MTM filter terhadap suatu lokal window meliputi :

- Menentukan nilai konstanta K.
- Menentukan nilai standard deviasi noise (SDN).

- Menentukan isi lokal window 3x3 dari lokal window 5x5.
- Mengurutkan nilai piksel dalam lokal window 3x3 kemudian menentukan nilai mediannya.
- Menentukan nilai mean lokal window 5x5 untuk piksel>=(median-K*SDN) dan piksel<=(median+K*SDN).

Konstanta K dan standard deviasi noise (SDN) bertipe float. K mempunyai nilai 1.5 sampai dengan 2.5. Jika standard deviasi yang diinputkan semakin mendekati nilai standard deviasi suatu noise yang terdapat pada citra, maka citra hasil operasi filter DW-MTM semakin baik.



Gambar 4.12. Lokal window 5x5

Untuk menentukan isi lokal window 3x3 dari lokal window 5x5 dapat digunakan potongan program berikut ini :

```
     .
     .
wind = 5;
     .
     .
n = 0;
for(row=1;row<wind-1;++row)
  for(col=1;col<wind-1;++col)
  {
    lok_wind[n] = *farPtr(p,(long)(row*LebarCitra+col)));
    n++;
  }
   .
   .
```

Pada gambar 4.12 terdapat lokal window 5x5. Dari lokal window 5x5 tersebut ditentukan lokal window 3x3. Piksel yang berada di lokal window 3x3 dimasukkan dalam array dua dimensi. Misalkan lebar citra adalah sepuluh piksel maka :

```
lok_wind[0] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(1*10+1)))
            = *(farPtr(p,11L))
            = 100

lok_wind[1] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(1*10+2)))
            = *(farPtr(p,12L))
            = 95
```

```
lok_wind[2] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(1*10+3)))
            = *(farPtr(p,13L))
            = 250

lok_wind[3] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(2*10+1)))
            = *(farPtr(p,21L))
            = 175

lok_wind[4] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(2*10+2)))
            = *(farPtr(p,22L))
            = 5

lok_wind[5] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(2*10+3)))
            = *(farPtr(p,23L))
            = 87

lok_wind[6] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(3*10+1)))
            = *(farPtr(p,31L))
            = 201
```

```
lok_wind[7] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(3*10+2)))
            = *(farPtr(p,32L))
            = 247

lok_wind[8] = *(farPtr(p,(long)(row*LebarCitra+col)))
            = *(farPtr(p,(long)(3*10+3)))
            = *(farPtr(p,33L))
            = 196
```

Piksel yang berada dalam lokal window 3x3 tersebut diurutkan untuk menentukan nilai median.

Berikut ini program untuk mengurutkan piksel (ascending) dari lokal window 3x3 yang sudah dimasukkan dalam array satu dimensi (lok_wind[]) :

```
for(j=1;j<9;++j)
{
   temp = lok_wind[j];
   k = j - 1;
   while( k>=0 && lok_wind[k]>temp)
   {
     lok_wind[k+1] = lok_wind[k];
     k--;
   }
   lok_wind[k+1] = temp;
}
```

Setelah diurutkan dengan menggunakan program di atas maka :

lok_wind[0] = 5
lok_wind[1] = 87
lok_wind[2] = 95
lok_wind[3] = 100
lok_wind[4] = 175
lok_wind[0] = 196
lok_wind[0] = 201
lok_wind[0] = 247
lok_wind[0] = 250

Nilai mediannya adalah isi lok_wind[4] yaitu 175.

Nilai median yang diperoleh digunakan untuk menentukan mean lokal window 5x5. Piksel yang ikut dalam perhitungan mean adalah piksel yang memenuhi syarat :

piksel >= (median-K*SDN) dan **piksel <= (median+K*SDN)**

Di bawah ini adalah program untuk menentukan mean lokal window 5x5 :

```
total_piksel = 0; juml_piksel = 0;
for(row=0;row<wind;++row)
 for(col=0;col<wind;++col)
 {
   piksel = *(farPtr(p,(long)(row*LebarCitra+col)));
   if((piksel>=(median-K*SDN))&&(piksel<=(median+K*SDN)))
   {
     total_piksel = total_piksel + piksel;
     juml_piksel = juml_piksel + 1;
   }
 }
if(juml_piksel>0)
   *dest = (unsigned char)(total_piksel/juml_piksel);
else  *dest = *source;
```

Keterangan :

piksel : nilai masing-masing piksel dalam lokal window 5x5.

total_piksel : hasil penjumlahan piksel yang memenuhi syarat perhitungan mean.

juml_piksel : jumlah piksel yang ikut dalam perhitungan mean.

dest : buffer yang berisi citra hasil operasi DW-MTM filter.

source : buffer yang berisi data citra asli.

Lihat lokal window pada gambar 4.12.

Misalkan : K = 1.5

SDN = 40.963

Isi lokal window 3x3 :

100 95 250 175 5 87 201 247 196

Setelah diurutkan menjadi :

5 87 95 100 175 196 201 247 250

Nilai median : 175

median - (K x SDN) = 175-(1.5x40.963) = 113.5555 ≈ 113

median + (K x SDN) = 175+(1.5x40.963) = 236.4445 ≈ 236

Piksel di lokal window 5x5 yang bernilai 113 sampai dengan 236 adalah piksel yang ikut dalam perhitungan mean.

total_piksel = 150 + 226 + 215 + 200 + 175 + 160 + 165 +
201 + 196 + 190 + 205 + 115 + 202
= 2400

juml_piksel = 13

mean = total_piksel/juml_piksel
= 2400/13
= 184.615 ≈ 184

Jika nilai mean yang dihasilkan lebih besar dari 255 maka nilai mean sama dengan 255 karena piksel grayscale 256 graylevel tidak ada yang bernilai lebih besar dari 255.

Nilai 184 akan menggantikan nilai piksel di lokal window 5x5, sehingga isi lokal window 5x5 yang baru adalah :

150	79	15	111	226
255	100	95	250	215
200	175	184	87	160
165	201	247	196	190
10	205	75	115	202

B. MMSE filter

Langkah untuk melakukan operasi MMSE filter terhadap suatu lokal window adalah :

- Menentukan ukuran lokal window.
- Menentukan varian noise yang bertipe float.
- Menentukan isi lokal window.
- Memasukkan isi lokal window dalam array satu dimensi (lok_wind[]).
- Menghitung nilai varian lokal window.
- Menghitung nilai output MMSE filter.

Varian noise adalah nilai varian suatu noise yang terdapat pada citra. Ketepatan menentukan nilai varian ini mempengaruhi hasil operasi MMSE filter. Apabila nilai varian noise yang diinputkan semakin mendekati nilai varian noise dalam citra maka citra hasil operasi filter ini semakin baik.

Pada operasi MMSE filter diperlukan nilai varian lokal window. Program di bawah ini bertujuan untuk menentukan nilai varian lokal window.

```
float MeanPikselKw,LokalMean,LokalVar;
   .
   .
total = 0;
total_kuadrat = 0;

for(n=0;n<wind*wind;++n)
{
```

```
 total = total + lok_wind[n];
 total_kuadrat = total_kuadrat + lok_wind[n]*lok_wind[n];
}

MeanPikselKw = (float)total_kuadrat/(float)(wind*wind);
LokalMean = (float)total/(float)(wind*wind);
LokalVar = MeanPikselKw - LokalMean * LokalMean;
    .
    .
```

Variabel *total* adalah hasil penjumlahan seluruh piksel dalam lokal window. Nilai variabel *total_kuadrat* diperoleh dengan cara masing-masing piksel dalam lokal window dikuadratkan kemudian dijumlahkan seluruhnya.

MeanPikselKw adalah nilai variabel *total_kuadrat* dibagi dengan jumlah piksel dalam lokal window, sedangkan *LokalMean* merupakan nilai rata-rata piksel dalam lokal window, yaitu hasil penjumlahan seluruh piksel dalam lokal window dibagi dengan jumlah piksel dalam lokal window.

Untuk memperoleh nilai varian lokal window (*LokalVar*) maka isi variabel *MeanPikselKw* dikurangi dengan kuadrat dari isi variabel *LokalMean*.

Apabila nilai *LokalVar* sama dengan nol, maka output MMSE filter sama dengan nilai *LokalMean*, sedangkan jika lebih besar dari nol, maka output MMSE filter adalah :

$$\left(1 - \frac{\mathbf{NoiseVar}}{\mathbf{LokalVar}}\right) \mathbf{X} * \mathbf{source} + \frac{\mathbf{NoiseVar}}{\mathbf{LokalVar}} \mathbf{X}\ \mathbf{LokalMean}$$

*source adalah nilai piksel dalam buffer asal yang berada di tengah-tengah lokal window.

Jika output MMSE filter lebih besar dari 255 maka nilai output tersebut sama dengan 255, begitu pula jika lebih kecil dari nol maka nilai output sama dengan nol karena citra yang diperbaiki adalah citra grayscale yang mempunyai nilai graylevel nol sampai dengan 255.

Contoh :

5	12	6	7	0
1	25	70	9	11
13	8	99	3	4
21	20	17	10	2
6	16	14	22	23

Gambar 4.13. Lokal window 5x5

total = 5 + 12 + 6 + 7 + 0 + 1 + 25 + 70 + 9 + 11 + 13 +
8 + 99 + 3 + 4 + 21 + 20 + 17 + 10 + 2 + 6 + 16 +
14 + 22 + 23
= 424

$$total\_kuadrat = 5^2 + 12^2 + 6^2 + 7^2 + 0^2 + 1^2 + 25^2 + 70^2 + 9^2 + 11^2 + 13^2 + 8^2 + 99^2 + 3^2 + 4^2 + 21^2 + 20^2 + 17^2 + 10^2 + 2^2 + 6^2 + 16^2 + 14^2 + 22^2 + 23^2$$

$$= 18776$$

Karena ukuran lokal window pada gambar 4.13 adalah 5 (lokal window 5x5) maka jumlah piksel dalam lokal window adalah 25, sehingga :

$$MeanPikselKw = \frac{18776}{25}$$

$$= 751.04$$

$$LokalMean = \frac{424}{25}$$

$$= 16.96$$

$$LokalVar = 751.04 - (16.96)^2$$

$$= 751.04 - 287.6416$$

$$= 463.3984$$

Misalkan : NoiseVar = 200.0

$$MMSE = \left(1 - \frac{200.0}{463.3984}\right) \times 99 + \frac{200.0}{463.3984} \times 16.96$$

$$= 56.27 + 7.32$$

$$= 63.59 \approx 63$$

Nilai 63 akan menggantikan nilai piksel di tengah lokal window, sehingga lokal window pada gambar 4.13 berubah seperti tampak pada gambar 4.14.

5	12	6	7	0
1	25	70	9	11
13	8	63	3	4
21	20	17	10	2
6	16	14	22	23

Gambar 4.14. Lokal window hasil operasi MMSE filter

Hal itu bukan berarti nilai piksel 63 tersebut dimasukkan dalam buffer asal (buffer yang berisi data citra asli), tetapi harus dimasukkan dalam buffer tujuan (buffer yang berisi data citra hasil proses MMSE filter) karena operasi filtering untuk lokal window berikutnya harus menggunakan data citra asli.

4.4.2. Morphological filter

Morphological filter terdiri dari metode Erosion filter, Dilation filter, Closing filter, dan Opening filter. Operasi Closing filter dan Opening filter merupakan gabungan dari operasi Erosion filter dan Dilation filter.

A. Erosion filter

Erosion filter adalah metode filtering yang beroperasi dengan cara mencari nilai minimum perbedaan antara nilai piksel dalam lokal window dengan suatu mask. Ukuran lokal window sama dengan ukuran mask. Bila ukuran lokal window adalah 3x3 maka ukuran mask juga 3x3.

Program untuk melakukan operasi erosion filter terhadap suatu lokal window adalah :

```
        .
        .
minimum = 255;

for(j=0;j<wind*wind;++j)
{
   hasil = lok_wind[j] - mask[j];
   if(hasil<minimum)  minimum = hasil;
}
if(hasil<0)  *dest = 0;
else  *dest = (unsigned char)hasil;
        .
        .
```

Variabel *minimum* diberi harga awal 255, yaitu nilai tertinggi suatu piksel. Nilai piksel dalam lokal window dikurangi dengan elemen mask. Jika hasil pengurangan tersebut lebih kecil dari nilai variabel *minimum* maka variabel *minimum* akan berisi hasil pengurangan tadi. Proses tersebut dilakukan sampai dengan seluruh piksel dalam lokal window.

Kalau nilai akhir variabel *minimum* lebih kecil dari nol maka hasil operasi erosion filter sama dengan nol karena tidak ada nilai piksel yang lebih kecil dari nol. Sedangkan apabila nilai akhir variabel *minimum* lebih besar atau sama dengan nol maka nilai itulah hasil operasi erosion filter.

Operasi Erosion filter bertujuan untk memperhalus garis bentuk suatu obyek dengan cara pisel yang berada di daerah batas tepi menjadi lebih gelap. Kalau obyek gambar berwarna lebih gelap dan backgroundnya berwarna lebih terang maka obyek tersebut tampak lebih besar karena piksel berwarna terang (background) yang berada di daerah batas tepi menjadi lebih gelap. Begitu pula sebaliknya.

Pada gambar 4.15.(a). tampak citra dengan lebar enam piksel dan tinggi lima piksel. Citra tersebut mempunyai obyek gambar berwarna gelap dan background berwarna terang. Berikut ini adalah operasi erosion filter untuk citra pada gambar 4.15.(a) menggunakan lokal window 3x3 dengan mask pada gambar 4.15.(c) :

125	100	112	198	225	223
185	250	10	195	215	197
200	202	18	2	170	222
255	5	20	7	211	168
223	12	8	15	198	159

(a)

2	1	10	7	8	0
12	26	200	27	4	9
20	15	150	250	13	11
36	197	212	118	21	14
29	255	190	203	22	27

(b)

0	1	0
1	1	1
0	1	0

(c)

Gambar 4.15. (a),(b) Citra 6x5, (c) Mask 3x3

		125	100	112		0	1	0
lokal window	:	185	250	10	mask :	1	1	1
		200	202	18		0	1	0

Erosion = min(125-0,100-1,112-0,185-1,250-1,10-1,200-0,
202-1,18-0)
= min(125,99,112,184,249,9,200,201,18)
= 9

Lokal window di atas berubah menjadi :

125	100	112
185	9	10
200	202	18

Setelah operasi erosion filter terhadap seluruh lokal window dalam citra itu dilakukan, maka berubah seperti pada gambar 4.16.

125	100	112	198	225	223
185	9	2	1	2	197
200	4	1	1	1	222
255	4	2	1	2	168
223	12	8	15	198	159

Gambar 4.16. Citra hasil operasi erosion filter

Gambar 4.15.(b) adalah citra yang mempunyai obyek gambar berwarna terang dan background berwarna gelap. Setelah dilakukan operasi Erosion filter dengan mask pada gambar 4.15.(c) maka obyek gambar menjadi lebih kecil seperti tampak pada gambar 4.17.

2	1	10	7	8	0
12	0	1	6	0	9
20	12	14	4	3	11
36	14	15	13	11	14
29	255	190	203	22	27

Gambar 4.17. Citra hasil operasi erosion filter

B. Dilation filter

Dilation filter adalah salah satu metode filtering yang terdapat dalam Morphological filter. Proses dilation filter adalah mencari nilai maksimum hasil penjumlahan antara nilai piksel dalam lokal window dengan suatu mask. Ukuran mask sama dengan ukuran lokal window.

Di bawah ini adalah program untuk melakukan proses dilation filter :

```
      .
      .
maksimum = 0;

for(j=0;j<wind*wind;++j)
{
   hasil = lok_wind[j] + mask[j];
   if(hasil>maksimum)  maksimum = hasil;
}
if(maksimum>255)  *dest = 255;
else  *dest = (unsigned char)maksimum;
      .
      .
```

Variabel *maksimum* diberi harga awal nol yaitu nilai piksel paling rendah. Piksel dalam lokal window dijumlah dengan elemen mask. Jika hasil penjumlahan itu lebih besar dari nilai variabel *maksimum* maka hasil penjumlahan tersebut dimasukkan dalam variabel *maksimum*. Proses tersebut dilakukan sampai dengan seluruh piksel dalam lokal window. Nilai akhir pada variabel *maksimum* merupakan output dari operasi dilation filter.

Jika nilai akhir variabel *maksimum* lebih besar dari 255 maka output operasi dilation filter menjadi 255, tetapi bila kurang dari atau sama dengan 255 maka nilai akhir variabel *maksimum* adalah output operasi dilation filter. Output tersebut diletakkan pada buffer yang berisi data citra hasil proses filtering (**dest*).

Operasi dilation filter bertujuan untuk memperhalus garis bentuk obyek gambar dengan cara piksel di daerah batas tepi menjadi lebih terang.

Jika obyek gambar yang berwarna gelap dan background berwarna terang maka obyek gambar menjadi lebih kecil karena piksel obyek gambar yang berada di daerah batas tepi menjadi lebih terang. Begitu juga sebaliknya pada obyek gambar yang berwarna terang dan background yang berwarna gelap.

Berikut ini adalah contoh operasi dilation filter antara citra pada gambar 4.15.(a) dengan mask pada gambar 4.15.(c).

	125	100	112		0	1	0
lokal window :	185	250	10	mask :	1	1	1
	200	202	18		0	1	0

Dilation = max(125+0,100+1,112+0,185+1,250+1,10+1,200+0,
202+1,18+0)

= max(125,101,112,186,251,11,200,203,18)

= 251

Lokal window di atas berubah menjadi :

125	100	112
185	251	10
200	202	18

Setelah operasi dilation filter pada semua lokal window, maka citra pada gambar 4.15.(a) tampak seperti gambar 4.18.

Jika operasi dilation filter dilakukan pada citra yang mempunyai obyek berwarna terang dan background berwarna gelap seperti gambar 4.15.(b) maka obyek gambar menjadi lebih besar seperti pada gambar 4.19.

125	100	112	198	225	223
185	251	251	225	226	197
200	255	250	215	223	222
255	255	202	212	222	168
223	12	8	15	198	159

Gambar 4.18. Citra hasil operasi dilation filter

2	1	10	7	8	0
12	201	250	251	250	9
20	212	251	251	251	11
36	255	255	251	250	14
29	255	190	203	22	27

Gambar 4.19. Citra hasil operasi dilation filter

C. Closing filter

Operasi closing filter adalah gabungan dari operasi dilation filter dengan erosion filter. Operasi dilation filter dilakukan terlebih dahulu, kemudian citra hasil operasi dilation filter tersebut dikenai operasi erosion

filter. Proses closing filter dapat menggabungkan obyek yang terpisah atau menutup lubang dalam obyek gambar. Gambar 4.20 adalah citra dengan obyek gambar yang terpisah.

125	100	112	198	225	223
70	81	40	225	57	91
55	79	48	215	84	50
63	59	51	212	66	61
73	60	80	175	198	159

Gambar 4.20. Citra dengan obyek terpisah

Setelah dilakukan operasi dilation filter dengan mask pada gambar 4.15.(c) tampak seperti gambar 4.21.

125	100	112	198	225	223
70	125	226	226	226	91
55	82	225	226	225	50
63	80	215	216	215	61
73	60	80	175	198	159

Gambar 4.21. Citra hasil operasi dilation filter

Operasi erosion filter antara gambar 4.21 dengan mask pada gambar 4.15.(c) akan menghasilkan citra seperti pada gambar 4.22.

125	100	112	198	225	223
70	55	82	112	50	91
55	54	80	215	49	50
63	55	60	80	50	61
73	60	80	175	198	159

Gambar 4.22. Citra hasil operasi dilation filter

Gambar 4.23 adalah obyek citra yang mempunyai lubang di tengahnya. Apabila dilakukan operasi dilation filter dengan mask pada gambar 4.15.(c) menghasilkan gambar 4.24.

73	49	50	57	63	223
70	55	170	145	50	91
55	54	82	85	49	50
63	55	60	80	50	61
125	60	80	75	98	159

Gambar 4.23. Obyek citra dengan lubang di tengah

Gambar 4.24 setelah dikenai operasi erosion filter menghasilkan citra seperti gambar 4.25 yang merupakan output proses closing filter.

73	49	50	57	63	223
70	171	171	171	223	91
55	170	171	170	145	50
63	125	85	98	159	61
125	60	80	75	98	159

Gambar 4.24. Citra hasil dilation filter

73	49	50	57	63	223
70	48	49	50	50	91
55	54	84	85	61	50
63	55	60	74	50	61
125	60	80	75	98	159

Gambar 4.25. Citra hasil operasi closing filter

D. Opening filter

Opening filter adalah kebalikan dari operasi closing filter. Jika operasi closing filter adalah

dilation filter diikuti erosion filter, maka opening filter adalah erosion filter diikuti dilation filter. Opening filter dapat memisahkan obyek yang saling berdekatan atau memperbesar lubang pada obyek gambar. Operasi erosion filter terhadap citra pada gambar 4.26 dengan mask pada gambar 4.15.(c) menghasilkan citra gambar 4.27.

125	100	112	198	225	223
70	55	82	79	50	91
55	54	111	215	110	50
63	55	199	200	217	61
180	255	206	175	198	159

Gambar 4.26. Citra dengan obyek berwarna gelap

125	100	112	198	225	223
70	53	54	49	49	91
55	53	53	50	49	50
63	53	54	110	50	61
180	255	206	175	198	159

Gambar 4.27. Citra hasil operasi erosion filter

Output opening filter adalah hasil operasi dilation filter antara citra gambar 4.27 dengan mask gambar 4.15.(c) seperti pada gambar 4.28.

125	100	112	198	225	223
70	125	198	225	225	91
55	70	110	111	110	50
63	255	255	206	199	61
180	255	206	175	198	159

Gambar 4.28. Citra hasil operasi opening filter

Pada gambar 4.29 terdapat obyek citra berwarna gelap dengan lubang di tengahnya. Setelah proses erosion filter dengan mask gambar 4.15.(c) maka citra tersebut tampak seperti gambar 4.30.

100	49	50	57	63	69
70	55	170	145	200	91
55	54	221	199	189	50
63	55	208	188	213	61
125	60	80	75	98	77

Gambar 4.29. Obyek citra dengan lubang di tengah

100	49	50	57	63	69
70	48	49	50	50	91
55	53	53	144	49	50
63	53	54	74	50	61
125	60	80	75	98	77

Gambar 4.30. Citra hasil operasi erosion filter

Hasil operasi opening filter tampak seperti gambar 4.31, yaitu hasil proses dilation filter antara gambar 4.30 dengan mask gambar 4.15.(c).

100	49	50	57	63	69
70	100	144	145	144	91
55	70	145	145	145	50
63	125	144	145	144	61
125	60	80	75	98	77

Gambar 4.31. Citra hasil operasi opening filter

4.4.3. Nonlinear filter

Nonlinear filter adalah proses filtering pada citra dengan cara menghitung fungsi nonlinear tiap lokal window dalam citra kemudian mengganti nilai piksel di tengah lokal window dengan hasil fungsi nonlinear tersebut.

Nonlinear filter terdiri dari metode Alpha-Trimmed Mean, Contra-Harmonic, Geometric Mean, Maximum, Median, Midpoint, Minimum, Weighted Median, dan Yp Mean filter.

A. Alpha-Trimmed Mean filter

Sebelum dilakukan proses alpha-trimmed mean filter harus ditentukan ukuran lokal window dan nilai *endpoint*. Nilai endpoint menunjukkan jumlah piksel dalam lokal window yang tidak terlibat dalam proses filtering. Misalkan nilai endpoint sama dengan dua maka dua piksel pertama dan dua piksel terakhir lokal window tidak terlibat dalam proses filtering, dengan syarat nilai piksel dalam lokal window sudah diurutkan terlebih dahulu.

Program untuk mengurutkan piksel dalam lokal window dapat dilihat pada halaman 98. Nilai piksel lokal window yang sudah diurutkan berada dalam array satu dimensi (*lok_wind[]*).

Berikut ini program untuk melakukan proses filtering dengan metode alpha-trimmed mean filter :

```
      .
      .
  TotPiksel = 0;

for(n=P;N<wind*wind-P;++n)
   TotPiksel = TotPiksel + lok_wind[n];

JumlPiksel = wind*wind - 2*P;

*dest = (unsigned char)(TotPiksel/JumlPiksel);
      .
      .
```

Variabel *P* adalah nilai endpoint. *wind* sama dengan ukuran lokal window. *TotPiksel* merupakan hasil penjumlahan piksel dalam lokal window yang sudah diurutkan (tidak termasuk piksel yang dihilangkan oleh nilai endpoint). *JumlPiksel* adalah jumlah piksel dalam lokal window yang terlibat dalam proses filtering, yaitu jumlah piksel dalam lokal window dikurangi dengan nilai *P* dikalikan dua. Output filtering ini adalah *TotPiksel* dibagi dengan *JumlPiksel*. Nilai output tersebut menggantikan nilai piksel dalam lokal window dan diletakkan dalam buffer yang berisi citra hasil proses filtering (**dest*).

Contoh :

35	27	20	41	15
11	39	16	29	12
36	72	37	77	52
24	78	86	42	95
68	22	40	83	89

Gambar 4.32. Lokal window 5x5

Piksel dalam lokal window gambar 4.32 diurutkan seperti di bawah ini :

11 12 15 16 20 22 24 27 29 35 36 37 39 40
41 42 52 68 72 77 78 83 86 89 95

Misalkan nilai endpoint (P) = 3, maka piksel yang terlibat proses filtering adalah :

16 20 22 24 27 29 35 36 37 39 40 41 42 52
68 72 77 78 83

Jadi nilai piksel 11, 12, 15, 86, 89, dan 95 tidak masuk dalam proses filtering.

TotPiksel = 16 + 20 + 22 + 24 + 27 + 29 + 35 + 36 +
37 + 39 + 40 + 41 + 42 + 52 + 68 + 72 +
77 + 78 + 83
= 838

JumlPiksel = (wind x wind) - (2 x P)
= (5 x 5) - (2 x 3)
= 25 - 6
= 19

$$\text{Output} = \frac{\text{TotPiksel}}{\text{JumlPiksel}} = \frac{838}{19} = 40.1 \approx 40$$

Setelah proses filtering maka nilai output sebesar 40 tersebut menggantikan nilai piksel di tengah lokal window, sehingga bentuk lokal window gambar 4.32 tampak seperti gambar 4.33.

35	27	20	41	15
11	39	16	29	12
36	72	40	77	52
24	78	86	42	95
68	22	40	83	89

Gambar 4.33. Lokal window 5x5

B. Contra-Harmonic filter

Contra-Harmonic filter digunakan untuk menghilangkan positif outlier (white spot) atau negatif outlier (black spot). Ukuran lokal window (*wind*) harus ditentukan dulu sebelum proses filtering pada lokal window dilakukan. Selain itu harus ditentukan suatu nilai yang menjadi indikator apakah filtering tersebut digunakan untuk menghilangkan positif outlier atau negatif outlier.

Apabila indikator bernilai positif maka proses filtering digunakan untuk menghilangkan negatif outlier, tetapi jika indikator bernilai negatif maka digunakan untuk menghilangkan positif outlier.

Proses contra-harmonic filter dapat dilihat pada program berikut ini :

```
    .
    .

n = 0;
TotPiksel = 0.0;
TotPiksel_1 = 0.0;

for(j=0;j<wind*wind;++j)
 if((lok_wind[j]==0) && (P<0))  n = 1;
 else
 {
   TotPiksel = TotPiksel + pow((double)lok_wind[j],
                              (double)(P+1));
   TotPiksel_1 = TotPiksel_1 + pow((double)lok_wind[j],
                                  (double)P);
 }
```

```
if(n==1)   *dest = 0;
else
  if(TotPiksel_1 == 0.0)   *dest = 0;
  else
    if(TotPiksel/TotPiksel_1 > 255)   *dest = 255;
    else  *dest = (unsigned char)(TotPiksel/TotPiksel_1);

    .
    .
```

Variabel *P* berfungsi sebagai indikator. jika *P* bernilai negatif maka tujuan filtering untuk menghilangkan positif outlier, sedangkan jika *P* bernilai positif berarti untuk menghilangkan negatif outlier.

TotPiksel menyatakn hasil penjumlahan masing-masing piksel dalam lokal window yang sudah dipangkatkan dengan *P+1* dan variabel *TotPiksel_1* adalah hasil penjumlahan masing-masing piksel dalam lokal window yang sudah dipangkatkan dengan *P*.

Output contra-harmonic filter adalah nilai variabel *TotPiksel* dibagi dengan nilai variabel *TotPiksel_1*.

Contoh :

97	100	102
105	5	99
92	89	93

(a)

10	21	17
23	225	9
13	19	29

(b)

Gambar 4.34. (a) Black spot dalam lokal window
(b) White spot dalam lokal window

Gambar 4.34.(a) adalah black spot yang terdapat di tengah lokal windo 3x3. Untuk menghilangkan black spot itu harus menggunakan nilai *P* positif.

Misalkan P = 2, maka :

$$\text{TotPiksel} = (97)^{2+1} + (100)^{2+1} + (102)^{2+1} + (105)^{2+1} + (5)^{2+1} + (99)^{2+1} + (92)^{2+1} + (89)^{2+1} + (93)^{2+1}$$

$$= 7389944$$

$$\text{TotPiksel\_1} = (97)^{2} + (100)^{2} + (102)^{2} + (105)^{2} + (5)^{2} + (99)^{2} + (92)^{2} + (89)^{2} + (93)^{2}$$

$$= 75698$$

$$\text{Output contra-harmonic} = \frac{\text{TotPiksel}}{\text{TotPiksel\_1}}$$

$$= \frac{7389944}{75698}$$

$$= 97,6 \approx 97$$

Setelah dilakukan operasi contra-harmonic pada gambar 4.34.(a) maka nilai piksel 5 (black spot) yang berada di tengah lokal window akan diganti dengan nilai 97 hasil proses filtering yang berarti black spot di tengah lokal window menjadi hilang.

Bentuk lokal window gambar 4.34.(a) berubah seperti gambar 4.35.(a).

97	100	102
105	97	99
92	89	93

(a)

10	21	17
23	13	9
13	19	29

(b)

Gambar 4.35. Hasil contra-harmonic filter
(a) P positif
(b) P negatif

Gambar 4.34.(b) adalah white spot yang berada di tengah lokal window 3x3. Hal ini berarti nilai *P* harus bernilai negatif untuk menghilangkan white spot tersebut. Misalkan P = -2, maka :

$$\text{TotPiksel} = (10)^{-2+1} + (21)^{-2+1} + (17)^{-2+1} + (23)^{-2+1} + (225)^{-2+1} + (9)^{-2+1} + (13)^{-2+1} + (19)^{-2+1} + (29)^{-2+1}$$

$$= 0.53$$

$$\text{TotPiksel\_1} = (10)^{-2} + (21)^{-2} + (17)^{-2} + (23)^{-2} + (225)^{-2} + (9)^{-2} + (13)^{-2} + (19)^{-2} + (29)^{-2}$$

$$= 0.04$$

$$\text{Output contra-harmonic} = \frac{\text{TotPiksel}}{\text{TotPiksel\_1}}$$

$$= \frac{0.53}{0.04}$$

$$= 13.25 \approx 13$$

White spot di tengah lokal window 4.34.(b) akan hilang karena diganti dengan nilai 13 hasil proses contra-harmonic. Lihat gambar 4.35.(b).

C. Geometic Mean filter

Geometric mean filter adalah proses filtering dengan cara mengalikan tiap piksel dalam lokal window yang sudah dipangkatkan dengan 1/N, dimana N adalah jumlah piksel dalam lokal window.

Program untuk melakukan proses geometric mean filter adalah :

```
    .
    .

   hasil = 1.0;
for(n=0;n<wind*wind;++n)
   hasil = hasil * pow((double)lok_wind[n],
                       (double)(1/(wind*wind)));

if(hasil>255)   *dest = 255;
else  *dest = (unsigned char)hasil;
   .
   .
```

Contoh : lokal window 3x3

205	12	15
179	20	18
182	17	25

Ukuran lokal window (wind) = 3

hasil = $(205)^{1/9}$ x $(12)^{1/9}$ x $(15)^{1/9}$ x $(179)^{1/9}$ x $(20)^{1/9}$ x
$(18)^{1/9}$ x $(182)^{1/9}$ x $(17)^{1/9}$ x $(25)^{1/9}$

= 38.45 ≈ 38

Bentuk lokal window menjadi :

205	12	15
179	38	18
182	17	25

D. Harmonic Mean filter

Harmonic mean filter digunakan untuk menghilangkan positif outlier (white spot).

Output harmonic mean filter diperoleh dengan cara jumlah piksel dalam lokal window dibagi dengan hasil penjumlahan satu per nilai piksel dalam lokal window.

Programnya adalah :

```
      .
      .
   n = 0;
   TotPiksel = 0.0;
for(j=0;j<wind*wind;++j)
   if(lok_wind[j] == 0)    n = 0;
   else
       TotPiksel = TotPiksel + 1.0/(float)lok_wind[j];
if(n==1)  *dest = 0;
else
  if((wind*wind)/TotPiksel > 255)  *dest = 255;
  else
      *dest = (unsigned char)((wind*wind)/TotPiksel);
    .
    .
```

Misalkan suatu white spot dalam lokal window 3x3 seperti pada gambar 4.34.(b).

Ukuran lokal window adalah 3 (lokal window 3x3), sehingga jumlah piksel dalam lokal window adalah 9.

$$\text{TotPiksel} = 1/10 + 1/21 + 1/17 + 1/23 + 1/225 + 1/9 + 1/13 + 1/19 + 1/29$$

$$= 0.53$$

$$\text{Output} = \frac{\text{jumlah piksel dalam lokal window}}{\text{TotPiksel}}$$

$$= \frac{9}{0.53}$$

$$= 16.9 \approx 16$$

White spot dalam lokal window gambar 4.34.(b) menjadi hilang karena nilai piksel 225 di tengan lokal window diganti dengan nilai 16 sebagai hasil proses harmonic mean filter.

Setelah proses harmonic mean filter dilakukan, maka bentuk lokal window gambar 4.34.(b) tampak seperti gambar 4.36.

10	21	17
23	16	9
13	19	29

Gambar 4.36. Hasil proses harmonic mean filter

E. Maximum filter

Maximum filter digunakan untuk menghilangkan negatif outlier noise (black spot) dalam suatu citra dengan cara mencari nilai maksimum piksel dalam lokal window.

Berikut ini adalah potongan program untuk proses maximum filter :

```
    .
    .
  maksimum = 0;
for(n=0;n<wind*wind;++n)
   if(lok_wind[n] > maksimum)   maksimum = lok_wind[n];
*dest = (unsigned char)maksimum;
    .
    .
```

Gambar 4.34.(a) adalah black spot dalam lokal window 3x3. Output maximum filter adalah :

maksimum = max(97, 100, 102, 105, 5, 99, 92, 89, 93)

= 105

Nilai piksel 5 di tengah lokal window gambar 4.34.(a) akan diganti dengan nilai 105, sehingga black spot dalam lokal window tersebut hilang. Lihat gambar 4.37.

97	100	102
105	105	99
92	89	93

Gambar 4.37. Hasil proses maximum filter

F. Median filter

Proses median filter adalah mencari nilai median dalam lokal window, sehingga piksel dalam lokal window harus diurutkan terlebih dahulu. Median filter dapat mempertahankan edge dalam citra.

Berikut ini program untuk melakukan proses median filter dalam suatu lokal window :

```
      .
      .
for(j=1;j<wind*wind-1;++j)
{
   temp = lok_wind[j];
   k = j - 1;
   while(k >= 0 && lok_wind[k] > temp)
   {
     lok_wind[k+1] = lok_wind[k];
     k = k - 1;
   }
   lok_wind[k+1] = temp;
}

*dest = (unsigned char)lok_wind[(wind*wind)/2];
      .
      .
```

Contoh : lokal window 3x3

27	9	12
15	45	3
18	20	7

Isi lokal window tersebut diurutkan :

3 7 9 12 15 18 20 27 45

Nilai 15 merupakan nilai mediannya, maka nilai 15 menggantikan nilai piksel di tengah lokal window sehingga bentuk lokal window menjadi :

27	9	12
15	15	3
18	20	7

G. Minimum filter

Minimum filter dapat digunakan untuk menghilangkan positif outlier noise (white spot) dalam lokal window. Output filter ini diperoleh dari nilai minimum piksel yang berada dalam lokal window.

Program berikut ini bertujuan untuk melakukan proses minimum filter :

```
      .
      .
   minimum = 255;
for(n=0;n<wind*wind;++n)
   if(lok_wind[n] < minimum)    minimum = lok_wind[n];

*dest = (unsigned char)minimum;
      .
      .
```

Contoh pada gambar 4.34.(b). Gambar tersebut mempunyai white spot di tengah lokal window 3x3. Proses minimum filter lokal window 4.34.(b) :

minimum = min(10, 21, 17, 23, 225, 9, 13, 19,29)

= 9

Nilai 9 akan menggantikan nilai piksel di tengah lokal window, sehingga gambar 4.34.(b) tampak sebagai berikut :

10	21	17
23	9	9
13	19	29

Dari gambar itu tampak bahwa white spot dalam lokal window telah hilang.

H. Midpoint filter

Output midpoint filter diperoleh dari perhitungan rata-rata antara nilai maksimum dan nilai minimum piksel yang berada dalam lokal window.

Potongan program berikut ini untuk melakukan proses midpoint filter :

```
      .
      .
     minimum = 255;
for(n=0;n<wind*wind;++n)
   if(lok_wind[n] < minimum)    minimum = lok_wind[n];

     maksimum = 0;
for(n=0;n<wind*wind;++n)
   if(lok_wind[n] > maksimum)    maksimum = lok_+wind[n];

*dest = (unsigned char)((minimum+maksimum)/2);
      .
      .
```

Variabel *minimum* berisi nilai minimum piksel yang berada dalam lokal window, sedangkan variabel *maksimum* berisi nilai maksimum piksel yang berada dalam lokal window yang sama.

Untuk mendapatkan output midpoint filter maka nilai variabel *minimum* ditambah dengan nilai variabel *maksimum* kemudian dibagi dua. Output tersebut diletakkan dalam suatu buffer yang berisi citra hasil proses filtering (**dest*) untuk menggantikan nilai piksel di tengah lokal window.

Contoh : lokal window 3x3

54	27	77
19	9	84
39	81	67

minimum = min(54, 27, 77, 19, 9, 84, 39, 81, 67)

= 9

maksimum = max(54, 27, 77, 19, 9, 84, 39, 81, 67)

= 84

output midpoint = (9 + 84)/ 2

= 46.5 ≈ 46

Bentuk lokal window yang baru :

54 27 77

19 46 84

39 81 67

I. Weighted Median filter

Weighted median filter tidak sama dengan median filter. Metode filtering ini memerlukan suatu mask. Ukuran mask harus sama dengan ukuran lokal window.

Masing-masing piksel dalam lokal window diulangi sebanyak elemen mask, yaitu piksel di kolom pertama baris pertama lokal window diulangi sebanyak elemen mask di kolom pertama baris pertama, piksel di kolom kedua baris pertama lokal window diulangi sebanyak elemen mask di kolom kedua baris pertama, begitu seterusnya untuk seluruh piksel dalam lokal window.

Program untuk melakukan proses pengulangan piksel dalam lokal window adalah :

```
   n = 0;
for(j=0;j<wind*wind;++j)
   for(m=0;m<mask[j];++m)
   {
      ulang[n] = lok_wind[j];
      n++;
   }
```

Mask yang digunakan sudah dimasukkan dalam array satu dimensi (*mask[]*). Hasil pengulangan diletakkan dalam array *ulang[]* kemudian diurutkan dan dicari nilai mediannya. Program untuk mengurutkan dan mencari nilai median seperti di bawash ini :

```
for(j=1;j<n-1;++j)
{
    temp = ulang[j];
    k = j - 1;
    while(k>=0 && ulang[k]>temp)
    {
       ulang[k+1] = ulang[k];
       k = k - 1;
    }
    ulang[k+1] = temp;
}
*dest = (unsigned char)ulang[n/2];
```

Contoh : lokal window 3x3 mask 3x3

45	71	65	1	1	1
52	170	25	3	3	3
30	112	37	1	1	1

Pengulangan :

45 71 65 52 52 52 170 170 170 25 25 25
30 112 37

ulang[0] = 45
ulang[1] = 71
ulang[2] = 65
ulang[3] = 52
ulang[4] = 52
ulang[5] = 52
ulang[6] = 170
ulang[7] = 170
ulang[8] = 170
ulang[9] = 25
ulang[10] = 25
ulang[11] = 25
ulang[12] = 30
ulang[13] = 112
ulang[14] = 37

Data tersebut diurutkan :

25 25 25 30 37 45 52 52 52 65 71
112 170 170 170

ulang[0] = 25
ulang[1] = 25
ulang[2] = 25
ulang[3] = 30
ulang[4] = 37
ulang[5] = 45
ulang[6] = 52
ulang[7] = 52
ulang[8] = 52
ulang[9] = 65
ulang[10] = 71
ulang[11] = 112
ulang[12] = 170
ulang[13] = 170
ulang[14] = 170

Output weighted median filter pada lokal window itu adalah nilai pada variabel *ulang[7]* yaitu 52, sehingga bentuk lokal window yang baru menjadi :

45	71	65
52	170	25
30	112	37

J. Yp Mean filter

Yp mean filter digunakan untuk menghilangkan white spot atau black spot. Pada Yp mean filter terdapat suatu variabel untuk menentukan apakah proses filtering tersebut menghilangkan white spot atau black spot. Jika variabel itu bernilai negatif maka Yp mean filter digunakan untuk menghilangkan white spot, sedangkan jika variabel bernilai positif maka digunakan untuk menghilangkan black spot.

Program Yp mean filter adalah :

```
    .
    .
  j = 0;
  TotPiksel = 0.0;
for(n=0;n<wind*wind;++n)
  if(lok_wind[n]==0 && P<0)   j = 1;
  else
      TotPiksel = TotPiksel + pow((double)lok_wind[n],
                                  (double)P);
if(j==1)   *dest = 0;
else
 if(TotPiksel==0.0)   *dest = 0;
 else
```

```
hasil = (int)pow((double)TotPiksel/(double)(wind*wind),
                 (double)(1.0/P));

if(hasil > 255)   *dest = 255;
else  *dest = (unsigned char)hasil;
    .
    .
```

Apabila variabel *P* bernilai positif maka Yp mean filter untuk menghilangkan black spot, tetapi jika bernilai negatif berarti untuk menghilangkan white spot. *TotPiksel* adalah hasil penjumlahan piksel dalam lokal window yang sudah dipangkatkan dengan nilai variabel *P*.

Output Yp mean filter berada dalam variabel *hasil* yang diperoleh dari nilai *TotPiksel* dibagi dengan jumlah piksel dalam lokal window kemudian dipangkatkan dengan *1/P*.

Pada gambar 4.34.(a) terdapat black spot dalam lokal window 3x3. Untuk menghilangkan black spot itu nilai *P* harus positif.

Misalkan nilai P = 2 dan ukuran lokal window (wind) = 3.

$$\text{TotPiksel} = (97)^2 + (100)^2 + (102)^2 + (105)^2 + (5)^2 + (99)^2 + (92)^2 + (89)^2 + (93)^2$$

$$= 75698$$

$$hasil = \left(\frac{TotPiksel}{N} \right)^{1/P} \quad ; N = wind * wind = 3x3 = 9$$

$$= \left(\frac{75698}{9} \right)^{1/2}$$

$$= 91.7 \approx 91$$

Lokal window gambar 4.34.(a) berubah menjadi :

97	100	102
105	91	99
92	89	93

Contoh white spot dalam lokal window 3x3 dapat dilihat pada gambar 4.34.(b), sehingga diperlukan *P* bernilai negatif untuk menghilangkannya.

Misalkan P = -2 dan wind = 3.

$$TotPiksel = (10)^{-2} + (21)^{-2} + (17)^{-2} + (23)^{-2} + (225)^{-2} + (9)^{-2} + (13)^{-2} + (19)^{-2} + (29)^{-2}$$

$$= 0.04$$

$$hasil = \left(\frac{0.04}{9} \right)^{1/-2}$$

$$= 15$$

Maka bentuk lokal window gambar 4.34.(b) menjadi :

10	21	17
23	15	9
13	19	29

4.4.4. Spatial filter

Metode spatial filter terdiri dari beberapa metode, antara lain Arithmetic Mean filter, Gaussian filter, High Pass filter, Low Pass filter, dan Weighted mean filter.

A. Arithmetic Mean filter

Arithmetic mean filter didefinisikan sebagai nilai rata-rata dari semua piksel yang berada dalam lokal window. Metode filtering ini menyebabkan batas tepi dalam citra tampak kabur. Semakin besar ukuran lokal window menyebabkan citra semakin tampak kabur.

Program untuk melakukan proses arithmetic mean filter pada lokal window :

```
  TotPiksel = 0;
for(n=0;n<wind*wind;++n)
   TotPiksel = TotPiksel + lok_wind[n];
*dest = (unsigned char)(TotPiksel/(wind*wind))
```

Variabel *TotPiksel* berisi hasil penjumlahan seluruh piksel dalam lokal window. Nilai output diperoleh dari operasi pembagian antara nilai variabel *TotPiksel* dengan

wind * *wind*, yaitu jumlah piksel yang berada dalam lokal window, dimana *wind* adalah ukuran lokal window.

Contoh : lokal window 5x5

52	12	46	77	42
25	43	7	26	63
8	21	25	84	71
10	86	37	27	98
74	0	42	9	11

TotPiksel = 52 + 12 + 46 + 77 + 42 + 25 + 43 + 7 + 26 + 63 + 8 + 21 + 25 + 84 + 71 + 10 + 86 + 37 + 27 + 98 + 74 + 0 + 42 + 9 + 11

= 996

$$\text{Output} = \frac{\text{TotPiksel}}{\text{wind x wind}} = \frac{996}{5 \text{ x } 5} = \frac{996}{} = 39.84 \approx 39$$

Nilai output 39 akan menggantikan nilai piksel di pusat lokal window di atas.

B. Gaussian filter

Proses gaussian filter dilakukan dengan cara konvolusi antara lokal window dengan mask yang elemennya berdistribusi gaussian.

Contoh : Gaussian mask 7x7

0	0	-1	-1	-1	0	0
0	-1	-3	-3	-3	-2	0
-1	-3	5	5	5	-3	-1
-1	-3	5	16	5	-3	-1
-1	-3	5	5	5	-3	-1
0	-2	-3	-3	-3	-2	0
0	0	-1	-1	-1	0	0

Dalam hal ini yang dimaksud dengan konvolusi adalah proses perkalian dan penjumlahan antara antara piksel dalam lokal window dengan elemen mask. Ukuran lokal window harus sama dengan ukuran mask. Semakin besar ukuran mask atau lokal window menyebabkan gambar tampak semakin kabur. Contoh proses konvolusi dapat dilihat di halaman 46.

Berikut ini adalah potongan program untuk melakukan proses konvolusi :

```
    .
    .
   jumlah = 0;
for(n=0;n<wind*wind;++n)
   jumlah = jumlah + lok_wind[n] * mask[n];

if(jumlah<0)  *dest = 0;
else if(jumlah>255)  *dest = 255;
else
     *dest = (unsigned char)jumlah;
    .
    .
```

C. High Pass filter

High pass filter menggunakan proses konvolusi untuk melakukan proses filtering. Filter ini mengakibatkan batas tepi suatu citra tampak lebih tajam karena piksel gelap di daerah batas tepi nilainya menjadi lebih kecil sedangkan piksel terang di daerah batas tepi nilainya semakin besar.

Contoh proses filtering pada gambar 4.38 dengan menggunakan mask :

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{matrix}$$

225	243	249	0	4	0
244	225	255	2	4	0
229	230	238	5	1	1
237	229	254	1	3	3
211	216	255	2	0	1
241	244	249	7	6	2

Gambar 4.38. Citra 6x6

Bentuk citra gambar 4.38 berubah seperti gambar 4.39.

225	243	249	0	4	0
244	148	255	0	4	0
229	159	255	0	1	1
237	191	255	0	3	3
211	24	255	0	0	1
241	244	249	7	6	2

Gambar 4.39. Hasil proses High pass filter

D. Low Pass filter

Low pass filter digunakan untuk mengurangi ketajaman gangguan berbentuk garis. Metode ini melakukan proses dengan cara konvolusi. Mask yang digunakan dapat menimbulkan efek pemerataan tingkat keabuan.

Contohnya mask di bawah ini :

1/9 1/9 1/9

1/9 1/9 1/9

1/9 1/9 1/9

Mask tersebut menyebabkan terjadinya proses perhitungan rata-rata pada lokal window, sehingga piksel atau garis yang tajam akan mnyesuaikan diri dengan piksel di sekitarnya, akibatnya gambar tampak agak kabur.

E. Weighted Mean filter

Weighted mean filter adalah rata-rata pembobotan semua piksel dalam lokal window. Metode ini memerlukan suatu mask karena bobot masing-masing piksel dalam lokal window ditentukan oleh tiap-tiap elemen mask. Ukuran mask sama dengan ukuran lokal window dan elemen mask tersebut dimasukkan dalam array satu dimensi (*mask[]*).

Program weighted mean filter :

```
      .
      .

   TotMask = 0;
   jumlah = 0;
for(n=0;n<wind*wind;++n)
{
   jumlah = jumlah + lok_wind[n] * mask[n];
   TotMask = TotMask + mask[n];
}

*dest = (unsigned char)(jumlah/TotMask);
     .
     .
```

jumlah adalah variabel yang menyatakan hasil penjumlahan seluruh piksel dalam lokal window yang sudah dikalikan dengan elemen mask, sedangkan *TotMask* adalah hasil penjumlahan seluruh elemen mask.

Nilai output metode filtering ini diperoleh dari nilai variabel *jumlah* dibagi dengan nilai variabel *TotPiksel*.

Contoh : lokal window 3x3 mask 3x3

12	20	21
9	8	17
27	19	30

1	2	1
2	3	2
1	2	1

jumlah = (12x1)+(20x2)+(21x1)+(9x2)+(8x3)+(17x2)+
(27x1)+(19x2)+(30x1)
= 236

TotPiksel = 1 + 2 + 1 + 2 + 3 + 2 + 1 + 2 + 1
= 15

$$\text{Output} = \frac{\text{jumlah}}{\text{TotPiksel}} = \frac{236}{15} = 15.73 \approx 15$$

Lokal window 3x3 di atas berubah menjadi :

12	20	21
9	15	17
27	19	30

4.5. Deteksi Batas Tepi

Proses deteksi batas tepi bertujuan untuk meningkatkan penampakan garis bata tepi (edge) suatu citra. Batas tepi yang dideteksi berwarna lebih terang sedangkan daerah lainnya berwarna gelap. Metode deteksi

batas tepi bermacam-macam. Berikut ini akan dijelaskan beberapa metode untuk melakukan deteksi batas tepi.

A. Kirsch, Prewitt, dan Sobel Edge Detector

Mask Kirsch, Prewitt, dan Sobel adalah mask yang dapat digunakan untuk melakukan proses deteksi batas tepi dengan metode konvolusi. Mask ini dapat melakukan deteksi batas tepi pada citra sebanyak delapan macam arah mata angin. Tiap mask hanya dapat mendeteksi satu arah.

Contoh mask Sobel yang digunakan untuk deteksi batas tepi arah utara :

$$\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

0	0	1	3	1	2
4	4	1	3	0	6
0	2	5	1	2	7
249	255	238	254	255	249
243	255	230	229	216	244
225	244	229	237	211	241

Gambar 4.40. Citra 6x6

Batas tepi adalah derah dalam citra yang mempunyai perubahan nilai intensitas yang tinggi. Gambar 4.40 dapat dideteksi dengan mask di atas karena pada daerah batas tepi, piksel yang berada di atas piksel yang dideteksi lebih kecil dari piksel yang berada di bawah piksel yang dideteksi, sehingga proses deteksi tersebut menghasilkan suatu nilai intensitas yang tinggi.

Sedangkan apabila gambar 4.40 tersebut diputar 180 derajat maka batas tepinya tidak akan terdeteksi dengan menggunakan mask di atas karena piksel yang berada di atas piksel yang dideteksi jauh lebih bedar dari piksel yang berada di bawah, sehingga menghasilkan nilai nol.

B. Quick Mask Edge Detector

Bentuk quick mask :

$$\begin{matrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{matrix}$$

Quick mask edge detector dapat digunakan untuk mendeteksi batas tepi delapan arah sekaligus karena mask tersebut dapat menghasilkan nilai intensitas yang tinggi pada daerah batas tepi di delapan posisi arah .

C. Faler Edge Detector

Wesley Faler menggunakan beberapa jenis mask untuk melakukan deteksi batas tepi.

Contohnya mask :

-1	0	1
-1	0	1
-1	0	1

Mask tersebut digunakan untuk melakukan deteksi batas tepi arah vertikal.

Daerah batas tepi pada gambar 4.38 dapat dideteksi oleh mask di atas karena terjadi perubahan nilai intensitas yang tinggi antara piksel di sebelah kiri dan kanan piksel yang dideteksi.

D. Range filter

Range filter adalah bagian dari nonlinear filter yang berfungsi untuk melakukan deteksi batas tepi. Prosesnya yaitu mencari perbedaan antara nilai minimum dan maksimum dari piksel yang berada dalam lokal window.

Program range filter :

```
.
.
   minimum = 255;
for(n=0;n<wind*wind;++n)
   if(lok_wind[n] < minimum)  minimum = lok_wind[n];
   maksimum = 0;
```

```
for(n=0;n<wind*wind;++n)
   if(lok_wind[n] > maksimum)  maksimum = lok_wind[n];

*dest = (unsigned char)(maksimum-minimum);
      .
      .
```

minimum adalah variabel yang berisi nilai minimum piksel yang berada dalam lokal window dan *maksimum* berisi nilai maksimum piksel dalam lokal window.

Jika isi variabel *maksimum* dikurangi dengan isi variabel *minimum* akan diperoleh suatu nilai yang merupakan output range filter.

Gambar 4.38 adalah citra dengan lebar dan tinggi enam piksel. Pada gambar itu terdapat daerah berwarna terang (putih) di sebelah kiri dan daerah berwarna gelap (hitam di sebelah kanan. Berikut ini akan dilakukan proses deteksi batas tepi yang menggunakan lokal window 3x3 untuk mengetahui garis outline antara daerah terang dan gelap.

Contoh lokal window pertama gambar 4.38 :

225	243	249
244	225	255
229	230	238

minimum = min(225, 243, 249, 244, 225, 255, 229,
230, 238)
= 225

maksimum = max(225, 243, 249, 244, 225, 255, 229,
230, 238)
= 255

Output = maksimum - minimum
= 255 - 225
= 30

Nilai piksel di pusat lokal window tersebut diganti dengan nilai 30, sehingga :

225 243 249
244 225 255
229 230 238

Dengan cara yang sama proses itu dilakukan pada semua lokal window dalam gambar 4.38, maka akan diperoleh citra yang baru seperti pada gambar 4.41 yang mempunyai garis outline di kolom ketiga dan keempat.

225	243	249	0	4	0
244	30	255	255	5	0
229	30	254	254	5	1
237	44	254	255	5	3
211	44	254	255	7	1
241	244	249	7	6	2

Gambar 4.41. Citra hasil range filter

E. Homogeneity Operator edge detector

Homogeneity operator edge detector adalah proses deteksi batas tepi dengan menggunakan operasi pengurangan, yaitu piksel di tengah lokal window dikurangi dengan masing-masing delapan buah piksel di sekelilingnya. Outputnya adalah nilai maksimum dari nilai absolut tiap-tiap hasil pengurangan itu.

Potongan program homogeneity operator edge detector :

```
    maksimum = 0;
for(n=0;n<wind*wind;++n)
   if(n != 4)
   {
      kurang = abs(lok_wind[4] - lok_wind[n])
      if(kurang > maksimum)  maksimum = kurang;
   }

*dest = (unsigned char)maksimum;
```

Contoh lokal window pertama gambar 4.38 :

225 243 249

244 225 255

229 230 238

Output = max(|225-225|,|225-243|,|225-249|,|225-244|,
|225-225|,|225-255|,|225-229|,|225-230|,
|225-238|)

= 30

Apabila proses semacam itu dilakukan pada seluruh lokal window pada gambar 4.38 maka diperoleh gambar 4.42.

225	243	249	0	4	0
244	30	255	253	4	0
229	25	237	250	4	1
237	26	253	254	3	3
211	39	254	253	7	1
241	244	249	7	6	2

Gambar 4.42. Hasil homogeneity operator edge detector

F. Difference Operator edge detector

Metode difference operator edge detector juga menggunakan operasi pengurangan dalam lokal window 3x3, yaitu piksel di kiri atas dikurangi piksel di kanan bawah, piksel di kanan atas dikurangi piksel di kiri bawah, piksel di tengah atas dikurangi piksel di tengah bawah, dan piksel di kiri tengah dikurangi dengan piksel di kanan tengah. Nilai maksimum dari nilai absolut hasil pengurangan itu merupakan outputnya.

Program difference operator edge detector :

```
    .
    .

    maksimum = 0;
    m = 8;
for(n=0;n<4;++n)
{
   kurang = abs(lok_wind[n] - lok_wind[m]);
   if(kurang > maksimum)   maksimum = kurang;
   m--;
}

*dest = (unsigned char)maksimum;
    .
    .
```

Contoh lokal window pertama gambar 4.38 :

225	243	249
244	225	255
229	230	238

Output = max(|225-238|,|249-229|,|243-230|,|244-255|)

= 20

Dengan cara yang sama proses tersebut dilakukan pada seluruh lokal window dalam gambar 4.38, sehingga menghasilkan gambar 4.43.

225	243	249	0	4	0
244	20	238	251	5	0
229	18	227	252	4	1
237	27	228	254	4	3
211	44	243	255	4	1
241	244	249	7	6	2

Gambar 4.43. Hasil difference operator edge detector

G. Contrast-based Edge Detector

Contrast-based edge detector di gunakan untuk melakukan deteksi batas tepi pada daerah yang pencahayaanya baik maupun yang kurang baik.

Proses yang dilakukan Contrast-based Edge Detector adalah melakukan proses deteksi batas tepi suatu lokal window dengan menggunakan salah satu metode deteksi batas tepi. Output yang dihasilkan dibagi dengan nilai rata-rata area. Yang dimaksud dengan nilai rata-rata area adalah hasil konvolusi antara lokal window dengan mask

yang semua elemennya berupa angka satu dan dibagi dengan jumlah piksel dalam lokal window.

Contoh :

Quick mask				Mask		
-1	0	-1		1	1	1
0	4	0	1/9 *	1	1	1
-1	0	-1		1	1	1

Batas tepi dengan pencahayaan baik :

50	100	100
50	100	100
50	100	100

Konvolusi dengan Quick mask:

400-50-50-100-100 = 100

Konvolusi dengan Mask :

50+50+50+100+100+100+100+100+100 / 9 = 750/9 = 83

Output = 100/83 = 1

Batas tepi dengan pencahayaan kurang baik :

5	10	10
5	10	10
5	10	10

Konvolusi dengan Quick mask:

40-5-5-10-10 = 10

Konvolusi dengan Mask :

5+5+5+10+10+10+10+10+10 / 9 = 75/9 = 8

Output = 10/8 = 1

Dari kedua output tersebut tampak bahwa dengan contrast-based edge detector maka daerah batas tepi dapat dideteksi di daerah yang pencahayaannya baik maupun yang pencahayaannya kurang baik.