Bab 4

Operasi-operasi Dasar Pengolahan Citra Dijital

Citra dijital direpresentasikan dengan matriks. Operasi pada citra dijital pada dasarnya adalah memanipulasi elemen-elemen matriks. Elemen matriks yang dimanipulasi dapat berupa elemen tunggal (sebuah *pixel*), sekumpulan elemen yang berdekatan, atau keseluruhan elemen matriks. Di dalam bab ini akan diuraikan operasi-operasi dasar pada pengolahan citra dijital.

4.1 Aras Komputasi

Operasi-operasi yang dilakukan pada pengolahan citra dapat dikelompokkan ke dalam empat aras (*level*) komputasi, yaitu aras titik, aras lokal, aras global, dan aras objek [JAI95]. Kita mulai pembahasan komputasi pada aras titik.

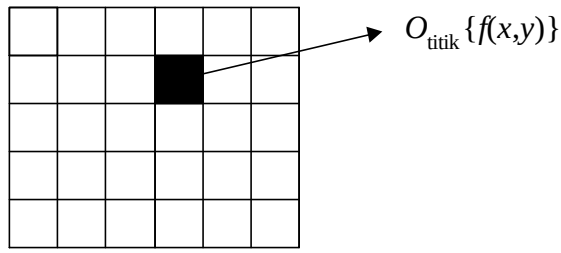
1. Aras Titik

Operasi pada aras titik hanya dilakukan pada *pixel* tunggal di dalam citra. Operasi titik dikenal juga dengan nama operasi *pointwise*. Operasi ini terdiri dari pengaksesan *pixel* pada lokasi yang diberikan, memodifikasinya dengan operasi operasi lanjar (*linear*) atau nirlanjar (*nonlinear*), dan menempatkan nilai *pixel* baru pada lokasi yang bersesuaian di dalam citra yang baru. Operasi ini diulangi untuk keseluruhan *pixel* di dalam citra.

Secara matematis, operasi pada aras titik dinyatakan sebagai (Gambar 4.1):

$$f_B(x, y) = O_{\text{titik}}\{f_A(x, y)\} \qquad (4.1)$$

yang dalam hal ini f_A dan f_B masing-masing adalah citra masukan dan citra keluaran, O_{titik} dapat berupa operasi lanjar (*linear*) atau nirlanjar (*nonlinear*). Yang dimaksud dengan operasi lanjar adalah operasi yang dapat dinyatakan secara matematis sebagai persamaan lanjar, kebalikannya adalah persamaan nirlanjar.



Gambar 4.1 *Operasi aras titik pada citra dijital.*

Operasi pada aras titik dapat dibagi menjadi tiga macam: berdasarkan intensitas, berdasarkan geometri, atau gabungan keduanya.

a. ***Berdasarkan intensitas.***
Nilai intensitas *u* suatu *pixel* diubah dengan transformasi *h* menjadi nilai intensitas baru *v*:

$$v = h(u), \qquad u, v \in [0, L] \tag{4.2}$$

Contoh operasi titik berdasarkan intensitas adalah operasi pengambangan (*thresholding*). Pada operasi pengambangan, nilai intensitas *pixel* dipetakan ke salah satu dari dua nilai, a_1 atau a_2, berdasarkan nilai ambang (*threshold*) *T*:

$$f(x, y)' = \begin{cases} a_1, & f(x,y) < T \\ a_2, & f(x,y) \geq T \end{cases} \tag{4.3}$$

Jika $a_1 = 0$ dan $a_2 = 1$, maka operasi pengambangan mentransformasikan citra hitam-putih ke **citra biner**. Dengan kata lain, nilai intensitas *pixel* semula dipetakan ke dua nilai saja: hitam dan putih. Nilai ambang yang dipakai dapat berlaku untuk keseluruhan *pixel* atau untuk wilayah tertentu saja (berdasarkan penyebaran nilai intensitas pada wilayah tersebut).

Operasi pengambangan pada citra Lena dengan fungsi transformasi:

$$f(x, y)' = \begin{cases} 0, & f(x,y) < 128 \\ 1, & f(x,y) \geq 128 \end{cases} \tag{4.4}$$

menghasilkan citra biner seperti yang diperlihatkan pada Gambar 4.2(a). Persamaan 4.4 menyatakan bahwa *pixel-pixel* yang nilai intensitasnya di bawah 128 diubah menjadi hitam (nilai intensitas = 0), sedangkan *pixel-pixel* yang nilai intensitasnya di atas 128 diubah menjadi putih (nilai intensitas = 1) . Algoritma transformasi citra hitam-putih menjadi citra biner ditunjukkan oleh Algoritma 4.1.

(a) (b)

Gambar 4.2. *(a) Citra biner Lena, (b) citra negatif Lena*

```
void biner(citra A, citra_biner B, int T, int N, int M)
/* Membuat citra biner dari citra A berdasarkan nilai ambang (threshold)
T yang dispesifikasikan. Ukuran citra adalah N ´ M. citra_biner adalah
tipe data untuk citra biner).
*/
{ int i, j;
  citra_biner B;

  for (i=0; i<=N-1; i++)
    for (j=0; j<=M-1; j++)
    {
      if (A[i][j] < T)
         B[i][j] = 0;
      else
         B[i][j] = 1;     /* atau diisi dengan 255 pada citra 8-bit */
    }
}
```

Algoritma 4.1. *Mengubah citra A menjadi citra biner.*

Contoh operasi titik yang lain adalah:

(i) Operasi negatif, yaitu mendapatkan citra negatif (*negative image*) meniru film negatif pada fotografi dengan cara mengurangi nilai intensitas *pixel* dari nilai keabuan maksimum. Misalnya pada citra dengan 256 derajat keabuan (8 bit), citra negatif diperoleh dengan persamaan:

$$f(x, y)' = 255 - f(x, y) \tag{4.5}$$

Sedangkan pada citra dengan 128 derajat keabuan,

$$f(x, y)' = 127 - f(x, y) \tag{4.6}$$

Hasil operasi negatif pada citra Lena diperlihatkan pada Gambar 4.2(b). Algoritma pembentukan citra negatif untuk citra hitam-putih dengan 256 derajat keabuan ditunjukkan oleh Algoritma 4.2. Untuk citra berwarna, citra negatifnya diperoleh dengan melakukan hal yang sama untuk setiap komponen *RGB.*

```
void negatif(citra A, citra B, int N, int M)
/* Membuat citra negatif dari citra A. Hasilnya disimpan di dalam citra
B.
   Ukuran citra adalah N ´ M.
*/
{ int i, j;

  for (i=0; i<=N-1; i++)
    for (j=0; j<=M-1; j++)
    {
      B[i][j] = 255 - A[i][j];
    }
}
```

Algoritma 4.2. *Membuat citra negatif dari sebuah citra dengan 256 derajat keabuan*

(ii) Pemotongan (*clipping*)
Operasi ini dilakukan jika nilai intensitas *pixel* hasil suatu operasi pengolahan citra terletak di bawah nilai intensitas minimum atau di atas nilai intensitas maksimum:

$$f(x, y)' = \begin{cases} 255, & f(x,y) > 255 \\ f(x,y), & 0 \le f(x,y) \le 255 \\ 0, & f(x,y) < 0 \end{cases} \tag{4.7}$$

Pemotongan (*clipping*) termasuk ke dalam operasi pengambangan juga.

(iii) Pencerahan citra (*image brightening*)

Kecerahan citra dapat diperbaiki dengan menambahkan (atau mengurangkan) sebuah konstanta kepada (atau dari) setiap *pixel* di dalam citra.

Secara matematis operasi ini ditulis sebagai

$$f(x, y)' = f(x, y) + b \qquad (4.8)$$

Jika *b* positif, kecerahan citra bertambah, sebaliknya jika *b* negatif kecerahan citra berkurang. Lihat contoh pencerahan citra pada Gambar 4.3 yang diterapkan pada citra Zelda. Semula citra Zelda tampak gelap, tetapi dengan menambahkan setiap nilai *pixel* dengan *b* = 10, citra Zelda menjadi lebih terang.

Persamaan 4.8 mengisyaratkan bahwa operasi pencerahan citra dapat menghasilkan nilai di bawah nilai intensitas minimum atau d atas nilai intensitas maksimum. Oleh karena itu, operasi *clipping* perlu diterapkan. Algoritma pencerahan citra ntuk citra dengan 256 derajat keabuan ditunjukkan oleh Algoritma 4.3.

```
void brightening(citra A, int b, citra B, int N, int M)
/* Pencerahan citra dengan cara menjumlahkan setiap pixel di dalam citra
A dengan sebuah skalar b. Hasil disimpan di dalam citra B. Citra A
berukuran N ´ M.
*/
{ int i, j, temp;

  for (i=0; i<=N-1; i++)
    for (j=0; j<=M-1; j++)
    {
       temp = A[i][j] + b;

       /* clipping */
       if (temp < 0)
         B[i][j] = 0;
       else
         if (temp > 255)
            B[i][j]=255;
         else
            B[i][j]=temp;
    }
  }
```

Algoritma 4.3. *Pencerahan citra*

Gambar 4.3. *Kiri: citra Zelda (agak gelap); kanan: citra Zelda setelah operasi pencerahan*

b. ***Berdasarkan geometri.***
Posisi *pixel* diubah ke posisi yang baru, sedangkan intensitasnya tidak berubah. Contoh operasi titik berdasarkan geometri misalnya pemutaran (rotasi), pergeseran (translasi), penskalaan (dilatasi), pembetulan erotan (distorsi) geometri (akan dijelaskan kemudian).

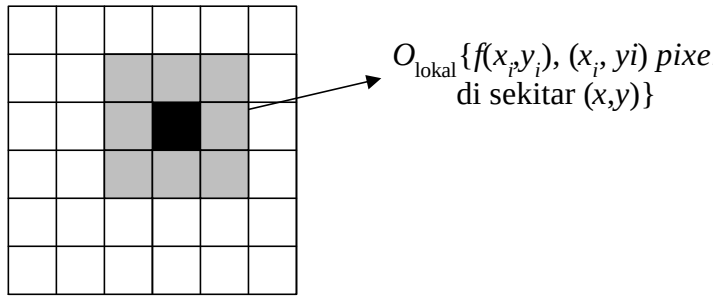
c. ***Gabungan intensitas dan geometri.***
Operasi ini tidak hanya mengubah nilai intensitas *pixel*, tapi juga mengubah posisinya. Misalnya *image morphing*, yaitu perubahan bentuk objek beserta nilai intensitasnya.

2. Aras Lokal

Operasi pada aras lokal menghasilkan citra keluaran yang intensitas suatu *pixel* bergantung pada intensitas *pixel-pixel* tetangganya (Gambar 4.4).

$$f_B(x, y)' = O_{\text{lokal}}\{f_A(x_i, y_j);\quad (x_i, y_j) \in N(x, y)\ \}\qquad (4.9)$$

(keterangan: N = *neighborhood*, yaitu *pixel-pixel* yang berada di sekitar (x, y))



Gambar 4.4. *Operasi aras lokal*

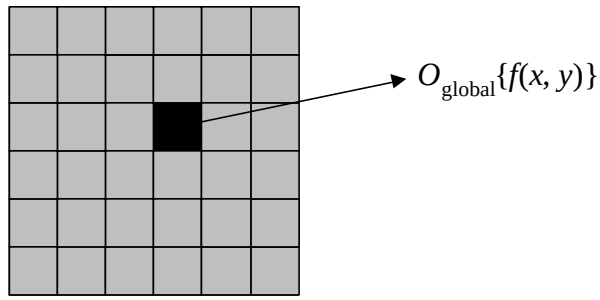
Contoh operasi beraras lokal adalah operasi konvolusi untuk mendeteksi tepi (*edge detection*) dan pelembutan citra (*image smoothing*). Gambar 4.5 adalah citra Lena hasil pendeteksian tepi. Konsep pendeteksian tepi dan penghalusan citra masing-masing akan dibahas di dalam Bab 8 dan Bab 7.

Gambar 4.5. *Hasil pendeteksian semua tepi dari citra Lena*

3. Aras Global

Operasi pada aras global menghasilkan citra keluaran yang intensitas suatu *pixel* bergantung pada intensitas keseluruhan *pixel* (Gambar 4.6).

$$f_B(x, y)' = O_{\text{global}}\{f_A(x, y)\} \qquad (4.10)$$



Gambar 4.6. Operasi aras global

Contoh operasi beraras global adalah operasi penyetaraan histogram untuk meningkatkan kualitas citra (akan dibahas pada kuliah selanjutnya).

4. Aras Objek

Operasi jenis ini hanya dilakukan pada objek tertentu di dalam citra. Tujuan dari operasi pada aras objek adalah untuk mengenali objek tersebut, misalnya dengan menghitung rata-rata intensitas, ukuran, bentuk, dan karakteristik lain dari objek. Operasi aras objek adalah operasi yang sangat sulit, karena sebelumnya kita harus dapat menjawab: apakah objek itu, bagaimana menemukannya?

4.2 Operasi Aritmetika

Karena citra dijital adalah matriks, maka operasi-operasi aritmetika matriks juga berlaku pada citra. Operasi matriks yang dapat dilakukan adalah:

1. Penjumlahan atau pengurangan antara dua buah citra A dan B:

   $$C(x, y) = A(x, y) \pm B(x, y),$$

2. Perkalian dua buah citra:

   $$C(x, y) = A(x, y)\, B(x, y),$$

3. Penjumlahan/pengurangan citra A dengan skalar c:

   $$B(x, y) = A(x, y) \pm c,$$

4. Perkalian/pembagian citra A dengan sebuah skalar c:

   $$B(x, y) = c \cdot A(x, y)$$

Ditinjau dari aras komputasi, operasi aritmetika termasuk ke dalam operasi aras titik. Penjelasan masing-masing operasi aritmetika matriks adalah sebagai berikut.

1. Penjumlahan Dua Buah citra

Persamaannya:

$$C(x, y) = A(x, y) + B(x, y) \qquad (4.11)$$

C adalah citra baru yang intensitas setiap *pixel*-nya adalah jumlah dari intensitas tiap *pixel* pada *A* dan *B*. Jika hasil penjumlahan intensitas lebih besar dari 255, maka intensitasnya dibulatkan ke 255. Algoritma penjumlahan dua buah citra ditunjukkan pada Algoritma 4.4.

```
void addition(citra A, citra B, citra C, int N, int M)
/* Menjumlahkan dua buah citra A dan B menjadi citra baru, C.
   Citra A, B, dan C masing-masing berukuran N ´ M.
*/
{ int i, j, temp;

  for (i=0; i<=N-1; i++)
    for (j=0; j<=M-1; j++)
    {
       temp=A[i][j] + B[i][j];
       if (temp > 255) C[i][j]=255; else C[i][j]=temp;
    }
}
```

Algoritma 4.4*.Penjumlahan dua buah citra*

Operasi penjumlahan citra dapat digunakan untuk mengurangi pengaruh derau (*noise*) di dalam data, dengan cara merata-ratakan derajat keabuan setiap *pixel* dari citra yang sama yang diambil berkali-kali. Misalnya untuk citra yang sama direkam dua kali, f_1 dan f_2, lalu dihitung intensitas rata-rata untuk setiap *pixel*:

$$f\,'(x,y) = \frac{1}{2}\{\, f_1(x, y) + f_2(x, y) \,\}$$

Hasil operasi mungkin bernilai riil, karena itu semua nilai riil tersebut perlu dibulatkan ke nilai bulat terdekat, nilai maksimum adalah 255.

2. Pengurangan Dua Buah Citra

Persamaannya:

$$C(x, y) = A(x, y) - B(x, y) \qquad (4.12)$$

C adalah citra baru yang intensitas setiap *pixel*-nya adalah selisih antara intensitas *pixel* pada *A* dan *B*.

Ada kemungkinan hasil operasi ini menghasilkan nilai negatif, oleh karena itu, operasi pengurangan citra perlu melibatkan operasi *clipping*.

Contoh aplikasi operasi pengurangan citra adalah untuk memperoleh suatu objek dari dua buah citra [HEN95]. Citra pertama misalnya foto sebuah ruangan yang kosong, citra kedua adalah foto ruangan yang sama tetapi ada orang di dalamnya. Hasil pengurangan citra kedua dengan gambar pertama menghasilkan citra yang latar belakangnya hitam, sedangkan latar depannya (objek orang) berwarna putih. Algoritmanya ditunjukkan pada Algoritma 4.5.

```
void substraction (citra A, citra B, citra C, int N, int M)
/* Mengurangkan dua buah citra A dan B menajdi citra baru, C.
   Citra A, B, dan C berukuran N ´ M.
*/
{ int i, j;

  for (i=0; i<=N-1; i++)
    for (j=0; j<=M-1; j++)
    {
      C[i][j]=A[i][j] - B[i][j];
      if (C[i][j] != 0) C[i][j]=255;  /* nyatakan objek berwarna putih */
    }
}
```

Algoritma 4.5. *Pengurangan dua buah citra untuk mendapatkan objek di dalamnya.*

Pengurangan citra juga dapat digunakan untuk mendeteksi perubahan yang terjadi selama selang waktu tertentu bila dua buah citra yang diambil adalah citra dari adegan yang sama. Teknik semacam ini dipakai pada *moving images*.

3. Perkalian Citra

Persamaannya:

$$C(x, y) = A(x, y)\, B(x, y) \tag{4.13}$$

Perkalian citra sering digunakan untuk mengoreksi kenirlanjaran sensor dengan cara mengalikan matriks citra dengan matrik koreksi. Jadi, dalam hal ini *A* adalah citra sedangkan *B* adalah matriks koreksi. Hasil operasi mungkin bernilai riil, karena itu semua nilai dibulatkan ke nilai bulat terdekat, nilai maksimum adalah 255. Algoritma perkalian citra dengan matriks koreksi ditunjukkan pada Algoritma 4.3. Kita mengasumsikan di sini ukuran citra dan matriks koreksi adalah $N \times N$.

Contoh 4.1. [GAL90] Mengalikan citra *A* dengan matriks koreksi:

$$\begin{bmatrix} 0 & 12 & 142 & 255 \\ 1 & 6 & 40 & 254 \\ 24 & 0 & 20 & 255 \\ 30 & 2 & 10 & 240 \end{bmatrix} \begin{bmatrix} 0.3 & 0.4 & 0.1 & 0.1 \\ 0.3 & 0.0 & 0.0 & 0.1 \\ 0.3 & 0.0 & 0.0 & 0.0 \\ 0.4 & 0.1 & 0.0 & 0.1 \end{bmatrix} = \begin{bmatrix} 0 & 17 & 157 & 255 \\ 2 & 6 & 40 & 255 \\ 32 & 0 & 20 & 255 \\ 42 & 3 & 10 & 255 \end{bmatrix}$$

Matriks citra A Matriks koreksi B Matriks keluaran C ■

```
void multiplication(citra A, matriks_riil B, citra C, int N)
/* Mengalikan buah citra A dengan matriks koreksi B menjadi citra C.
   Citra A, matriks B, dan hasil perkalian C berukuran N ´ N.
*/
{ int i, j, temp;

  for (i=0; i<=N-1; i++)
    for (j=0; j<=N-1; j++)
     {
       temp=0;
       for (k=0; k<=N-1; k++)
       {
         temp = temp + A[i][k]*B[k][j];

         /* clipping */
         if (temp < 0)
           C[i][j] = 0;
         else
           if (temp > 255)
              C[i][j]=255;
           else
              C[i][j]=temp;
       }
     }
}
```

Algoritma 4.6. *Perkalian citra A dengan matriks koreksi B.*

4. Penjumlahan/pengurangan citra dengan skalar

Persamaannya:

$$B(x, y) = A(x, y) \pm c \qquad (4.14)$$

Penjumlahan citra *A* dengan skalar *c* adalah menambah setiap *pixel* di dalam citra dengah sebuah skalar *c*, dan menghasilkan citra baru *B* yang intensitasnya lebih terang daripada *A*. Kenaikan intensitas sama untuk seluruh *pixel*, yaitu *c*.

Pengurangan citra *A* dengan skalar *c* adalah mengurangkan setiap *pixel* di dalam citra dengah sebuah skalar *c*, dan menghasilkan citra baru *B* yang intensitasnya lebih gelap daripada *A*. Penurunan intensitas sama untuk seluruh *pixel*, yaitu *c*. Contoh operasi penjumlahan/pengurangan citra dengan sebuah skalar adalah operasi pencerahan citra (lihat pembahasan operasi aras titik).

Baik operasi penjumlahan maupun pengurangan citra dengan sebuah skalar melibatkan operasi *clipping*. Algoritma penjumlahan/pengurangan citra dengan sebuah skalar sama seperti Algoritma 4.3.

5. Perkalian/pembagian Citra dengan Skalar
Persamaannya:

$$B(x, y) = c \cdot A(x, y), \text{ dan } B(x, y) = A(x, y) / c \qquad (4.15)$$

Perkalian citra *A* dengan skalar *c* menghasilkan citra baru *B* yang intensitasnya lebih terang daripada *A*. Kenaikan intensitas setiap *pixel* sebanding dengan *c*. Operasi perkalian citra dengan skalar dipakai untuk kalibrasi kecerahan (*callibration of brightness*).

Pembagian citra *A* dengan skalar *c* menghasilkan citra baru *B* yang intensitasnya lebih gelap daripada *A*. Penurunan intensitas setiap *pixel* berbanding terbalik dengan *c*. Operasi pembagian citra dengan skalar dipakai untuk normalisasi kecerahan (*normalization of brightness*).

Algoritma perkalian/pembagian citra dengan sebuah skalar serupa dengan Algoritma 4.3, hanya saja operasi + atau – diganti dengan * atau /.

4.3 Operasi Boolean pada Citra

Selain operasi aritmetika, pemrosesan citra dijital juga melibatkan operasi Boolean (**and**, **or**, dan **not**):

$$C(x, y) = A(x, y) \textbf{ and } B(x, y),$$

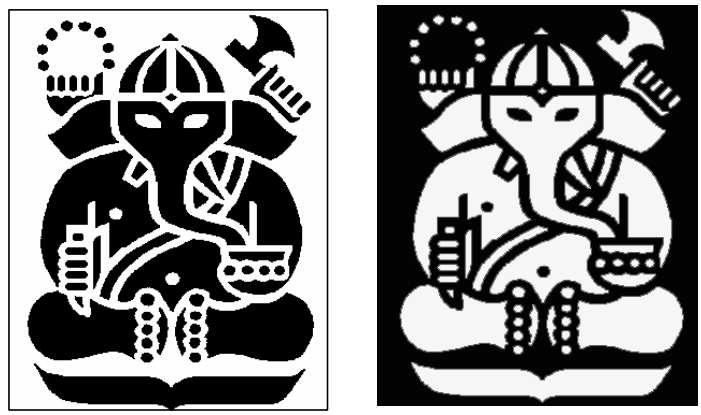
$$C(x, y) = A(x, y) \textbf{ or } B(x, y),$$

$$C(x, y) = \textbf{not } A(x, y). \qquad (4.16)$$

(dalam notasi Bahasa C, ketiga operasi di atas ditulis sebagai:

```
C[x][y]=A[x][y]&B[x][y]
C[x][y]=A[x][y]|B[x][y]
C[x][y]=!A[x][y]
```

)
Operasi Boolean mempunyai terapan yang penting pada pemrosesan morfologi pada citra biner. Pada citra biner, operasi **not** dapat digunakan untuk menentukan komplemen dari citra (Gambar 4.7). Algoritma membentuk komplemen dari citra biner ditunjukan oleh Algoritma 4.7.



(a) Ganesha (b) **not** Ganesha

Gambar 4.7. *Hasil operasi* ***not*** *pada citra biner Ganesha*

```
void not(citra_biner A, citra_biner B, int N, int M)
/* Membuat citra komplemen dari citra biner A. Komplemennya disimpan di
dalam B. Ukuran citra A adalah N ´ M.
*/
{ int i, j;

  for (i=0; i<=N-1; i++)
    for (j=0; j<=M-1; j++)
    {
      B[i][j] = !A[i][j];
    }
}
```

Algoritma 4.7. *Membuat citra komplemen dari citra biner*

4.5 Operasi Geometri pada Citra

Pada operasi geometrik, koordinat *pixel* berubah akibat transformasi, sedangkan intensitasnya tetap. Ini berbeda dengan dengan operasi aritmetika yang mana koordinat *pixel* tetap sedangkan intensitasnya berubah.

Operasi geometri yang dilakukan misalnya translasi, rotasi, penskalaan citra, dan pencerminan citra (*flipping*). Pengubahan geometri dari citra $f(x, y)$ menjadi citra baru $f'(x, y)$ dapat ditulis sebagai:

$$f'(x', y') = f(g_1(x, y), g_2(x, y)) \quad (4.17)$$

yang dalam hal ini, $g_1(x)$ dan $g_2(y)$ adalah fungsi transformasi geometrik. Dengan kata lain,

$$x' = g_1(x, y);$$
$$y' = g_2(x, y) \quad (4.18)$$

a. Translasi

Rumus translasi citra:

$$x' = x + m$$
$$y' = y + n \quad (4.19)$$

yang dalam hal ini, m adalah besar pergeseran dalam arah x, sedangkan n adalah besar pergeseran dalam arah y. Jika citra semula adalah A dan citra hasil translasi adalah B, maka translasi dapat diimplementasikan dengan menyalin citra dari A ke B:

$$B[x][y] = A[x + m][y + n] \quad (4.20)$$

Algoritma translasi citra ditunjukkan oleh Algoritma 4.8, sedangkan contoh translasi pada citra *camera* diperagakan pada Gambar 4.8.

```
void translation(citra A, citra B, int N, int M, int m, int n)
/* Mentranslasi citra A sejauh m, n. Hasil translasi disimpan di dala B.
   Ukuran citra adalah N ´ M.
*/
{ int i, j;

  for (i=0; i<=N-1; i++)`
    for (j=0; j<=M-1; j++)
    {
       B[i][j]=A[i+m][j+n];
    }
}
```

Algoritma 4.8. *Operasi translasi citra*

(a) (b)

Gambar 4.8. *Translasi pada citra camera: (a) citra semula, (b) citra hasil translasi dengan m = 30 dan n = 25. (Terima kasih kepada Nanda Firdausi M atas izin menggunakan output programnya).*

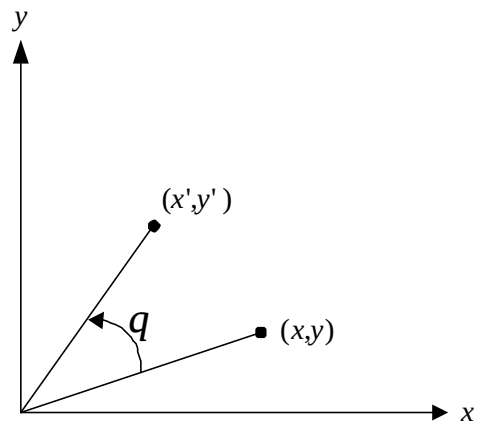
b. Rotasi

Rumus rotasi citra:

$$x' = x\cos(q) - y\sin(q)$$
$$y' = x\sin(q) + y\cos(q) \qquad (4.21)$$

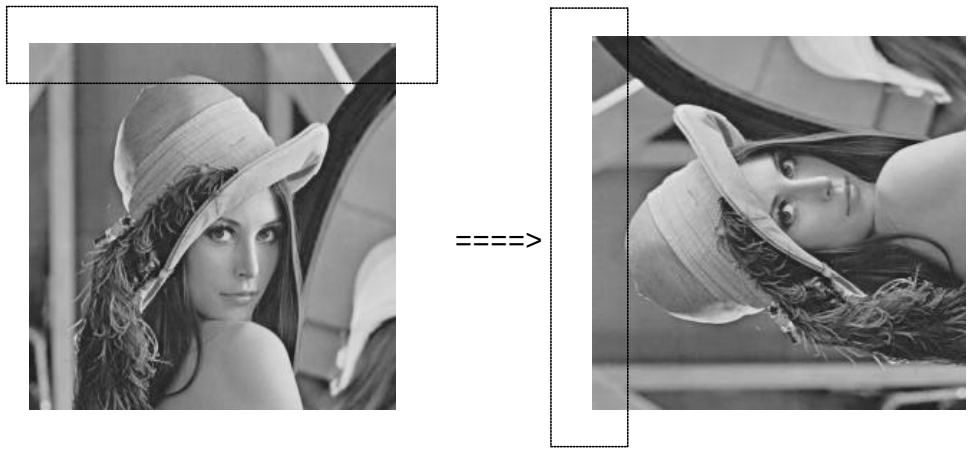
yang dalam hal ini, q = sudut rotasi berlawanan arah jarum jam (lihat Gambar 4.9). Jika citra semula adalah *A* dan citra hasil rotasi adalah *B*, maka rotasi citra dari *A* ke *B*:

$$B[x'][y'] = B[x\cos(q) - y\sin(\theta)][x\cos(q) + y\cos(q)] = A[x][y] \qquad (4.22)$$



Gambar 4.9. *Model rotasi citra*

Jika sudut rotasinya 90°, maka implementasinya lebih mudah dilakukan dengan cara menyalin *pixel-pixel* baris ke *pixel-pixel* kolom pada arah rotasi (Gambar 4.10). Rotasi 180° diimplementasikan dengan melakukan rotasi 90° dua kali. Algoritma rotasi citra sejauh 90 derajat berlawanan arah jarum jam ditunjukkan pada Algoritma 4.9, sedangkan rotasi citra sejauh 90 derajat searah jarum jam ditunjukkan pada Algoritma 4.10 [HEN95].



Gambar 4.10. *Rotasi citra Lena sejauh 90° berlawanan arah jarum jam*

```
void rotation90CCW(citra A, citra B, int N, int M)
/* Rotasi citra A sejauh 90° berlawanan arah jarum jam (CCW = Clock
Counter-wise).
   Ukuran citra adalah N ´ M. Hasil rotasi disimpan di dalam citra B.
*/
{ int i, j, k;

  for (i=0; i<=N-1; i++)
  {
    k=M-1;
    for (j=0; j<=M-1; j++)
    {
      B[k][i]=A[i][j];
      k--;
    }
  }
}
```

Algoritma 4.8. *Rotasi citra sejauh 90° berlawanan arah jarum jam.*

```
void rotation90CW(citra A, citra B, int N, int M)
/* Rotasi citra A sejauh 90° searah jarum jam (CW = Clock-wise).
   Ukuran citra adalah N ´ M. Hasil rotasi disimpan di dalam cira B.
*/
{ int i, j, k;

  k=M-1;
  for (i=0; i<=N-1; i++)
  {
    for (j=0; j<=M-1; j++)
    {
      B[j][k]=A[i][j];
    }
    k--;
  }
}
```

Algoritma 4.9. *Rotasi citra sejauh 90° searah jarum jam.*

c. Penskalaan Citra

Penskalaan citra, disebut juga *image zooming*, yaitu pengubahan ukuran citra (membesar/*zoom out* atau mengecil/*zoom in*).

Rumus penskalaan citra:

$$x' = s_x \cdot x$$
$$y' = s_y \cdot y \qquad (4.23)$$

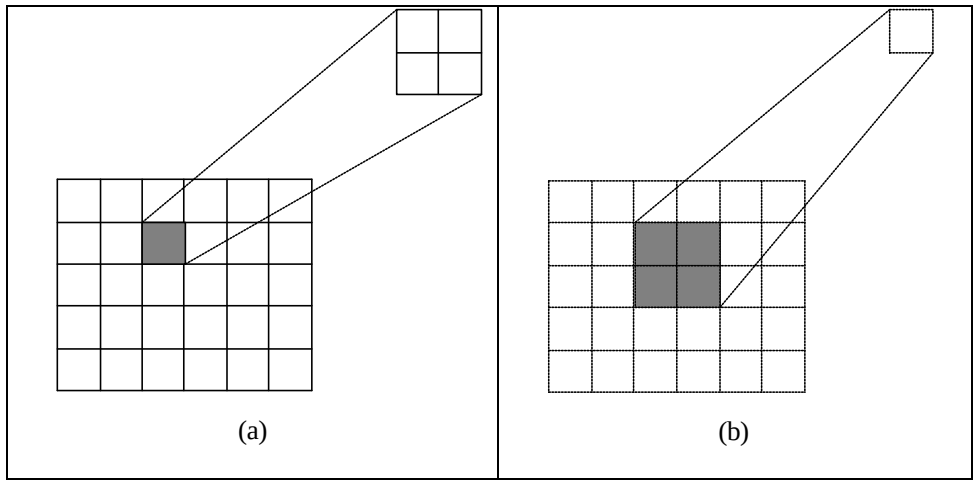
yang dalam hal ini, s_x dan s_y adalah faktor skala masing-masing dalam arah x dan arah y.

Jika citra semula adalah A dan citra hasil penskalaan adalah B, maka penskalaan citra dinyatakan sebagai:

$$B[x'][y'] = B[s_x \cdot x][\, s_y \cdot y] = A[x][y] \qquad (4.24)$$

Operasi *zoom out* dengan faktor 2 (yaitu, $s_x = s_y = 2$) diimplementasikan dengan menyalin setiap *pixel* sebanyak 4 kali (Gambar 4.11a). Jadi, citra 2×2 *pixel* akan menjadi 4×4 *pixel*. Algoritma *zoom out* dengan faktor skala = 2 ditunjukkan oleh Algoritma 4.10. Contoh citra yang diperbesar dua kali diperlihatkan pada Gambar 4.12 (citra kota San Fransisco).

Operasi *zoom in* (pengecilan) dengan faktor skala = ½ dilakukan dengan mengambil rata-rata dari 4 *pixel* yang bertetangga menjadi 1 *pixel* (Gambar 4.11b).



Gambar 4.11. *(a) Zoom out dengan faktor skala = 2, (b) Zoom in dengan faktor skala = 1/2*

```
void zoom_out(citra A, citra B, int N, int M)
/* perbesaran citra A dengan faktor skala 2
   Ukuran citra adalah N ´ M. Hasil perbesaran disimpa d dalam citra B.
*/
{ int i, j, k, m, n;

  m=0; n=0;
  for (i=0; i<=N-1; i++)
   {
     for (j=0; j<=M-1; j++)
     {
       B[m][n]= A[i][j];
       B[m][n+1]= A[i][j];
       B[m+1][n]= A[i][j];
       B[m+1][n+1]= A[i][j];
       n=n+2;
      }
      m=m+2;
      n=0;
   }

}
```

Algoritma 4.10. *Zoom out dengan faktor skala = 2*

(a) (b)

Gambar 4.12. *(a) Citra kota San Fransisco (ukuran normal), (b) citra kota San Fransisco setelah diperbesar 2 kali ($s_x = s_y = 2$):*

d. Flipping

Flipping adalah operasi geometri yang sama dengan pencerminan (*image reflection*). Ada dua macam *flipping*: horizontal dan vertikal (Gambar 4.13).

(a) citra (b) *flip* horizontal (c) *flip* vertikal

Gambar 4.13. *Flipping*

Flipping horizontal adalah pencerminan pada sumbu-*Y* (*cartesian*) dari citra *A* menjadi citra *B*, yang diberikan oleh:

$$B[x][y] = A[N - x][y] \quad (4.25)$$

Flipping vertikal adalah pencerminan pada sumbu-*X* (*cartesian*) dari citra *A* menjadi citra *B*, yang diberikan oleh:

$$B[x][y] = A[x][M - y] \quad (4.26)$$

Algoritma *flipping* vertikal ditunjukkan oleh Algoritma 4.11 [HEN95].

Pencerminan pada titik asal (*cartesian*) dari citra *A* menjadi citra *B* diberikan oleh:

$$B[x][y] = A[N - x][M - y] \quad (4.27)$$

Pencerminan pada garis $x = y$ dari citra *A* menjadi citra *B* diberikan oleh:

$$B[x][y] = A[y][x]$$

```
void vertical_flip(citra A, citra B, int N, int M)
/* Flipping vertikal (pencerminan terhadap sumbu-X) terhadap citar A. */
   Ukuran citra adalah N ´ M. Hasil flipping disimpan di dalam citra B.
*/
{ int i, j, k;

  k=M-1;
  for (i=0; i<=N-1; i++)
  {
   for (j=0; j<=M-1; j++)
   {
     B[k][j]=A[i][j];
   }
   k--;
   }

}
```

Algoritma 4.11. *Flipping vertikal.*