BAB 5

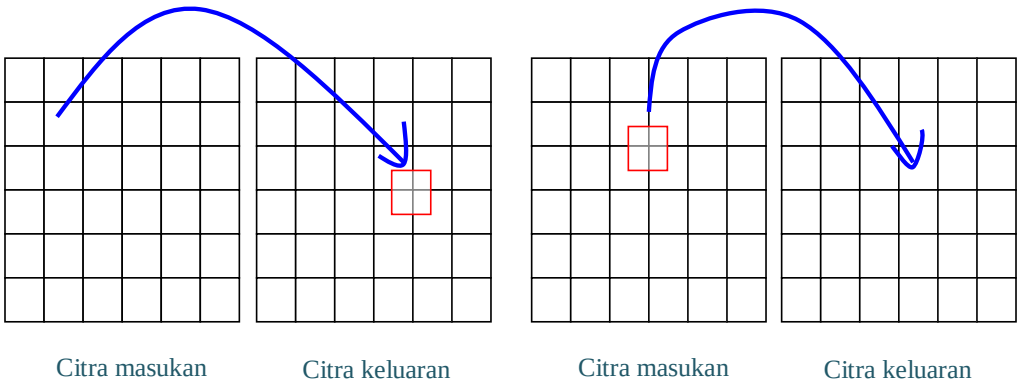
Operasi Geometrik

Setelah bab ini berakhir, diharapkan pembaca mendapatkan pengetahuan mengenai hal-hal berikut dan mampu mempraktikkannya.

- ✓ Pengantar operasi geometrik
- ✓ Penggeseran citra
- ✓ Pemutaran citra
- ✓ Interpolasi piksel
- ✓ Pemutaran citra berdasarkan sebarang koordi
- ✓ Pemutaran citra secara utuh
- ✓ Pembesaran citra
- ✓ Pengecilan citra
- ✓ Pembesaran citra dengan skala vertikal dan horizontal
- ✓ Pencerminan citra
- ✓ Transformasi *affine*
- ✓ Efek *ripple*
- ✓ Efek *twirl*
- ✓ Transformasi *spherical*
- ✓ Transformasi *bilinear*

5.1 Pengantar Operasi Geometrik

Operasi geometrik adalah operasi pada citra yang dilakukan secara geometris seperti translasi, rotasi, dan penyekalaan. Pada operasi seperti ini terdapat pemetaan geometrik, yang menyatakan hubungan pemetaan antara piksel pada citra masukan dan piksel pada citra keluaran. Secara prinsip, terdapat dua cara yang dapat dipakai. Pertama yaitu pemetaan ke depan dan kedua berupa pemetaan ke belakang. Perbedaan secara visual kedua cara tersebut diperlihatkan pada Gambar 5.1.



(b) Pemetaan ke depan

(a) Pemetaan ke belakang (mundur)

Gambar 5.1 Pemetaan geometrik

Gambar di atas menjelaskan bahwa pada cara pemetaan ke depan, posisi pada citra keluaran ditentukan dengan acuan pemrosesan pada citra masukan. Pada gambar tersebut terlihat bahwa kalau piksel keluaran berada pada posisi yang tidak tepat (tidak berupa bilangan bulat), penempatannya dapat berada pada salah satu dari empat kemungkinan. Dengan cara seperti ini, ada kemungkinan sebuah piksel pada citra keluaran tidak pernah diberi nilai atau malah diberi nilai lebih dari satu kali. Hal ini berbeda dengan pada pemetaan ke belakang. Pada pemetaan ke belakang, mengingat pemrosesan dimulai dari citra keluaran maka dipastikan bahwa semua piksel pada citra keluaran akan diberi nilai sekali saja berdasarkan piksel masukan.

Catatan

Lubang yang ditimbulkan karena piksel tidak diberi nilai pada pemetaan ke depan dapat dilihat pada Gambar 5.5.

Pada Gambar 5.1(a), piksel yang digunakan untuk menentukan piksel keluaran dapat ditentukan oleh salah satu piksel yang tercakup dalam kotak yang menggantung pada keempat piksel. Hal itu merupakan cara tersederhana yang dapat dilakukan dan biasa dinamakan sebagai pemilihan berdasarkan tetangga terdekat. Cara lain yang dapat dilakukan adalah dengan memperhitungkan empat piksel yang dapat mewakilinya. Cara ini dikenal dengan sebutan interpolasi bilinear, yaitu linear di arah vertikal dan mendatar. Kedua cara ini akan dibahas saat membicarakan pemutaran citra (Subbab 5.3).

5.2 Menggeser Citra

Penggeseran citra ke arah mendatar atau vertikal dapat dilaksanakan dengan mudah. Rumus yang digunakan sebagai berikut:

$$x_{baru} = x_{lama} + s_x \quad (5.1)$$

$$y_{baru} = y_{lama} + s_y \quad (5.2)$$

Untuk penyederhanaan pembahasan, *sx* dan *sy* dianggap bertipe bilangan bulat.

Contoh berikut menunjukkan program yang digunakan untuk melakukan penggeseran citra.

Program : geser.m

```
% GESER Melakukan operasi penggeseran citra.

F = imread('c:\Image\gedung.png');
[tinggi, lebar] = size(F);
```

```
sx = 45; % Penggesaran arah horisontal
sy = -35; % Penggesaran arah vertikal

F2 = double(F);
G = zeros(size(F2));
for y=1 : tinggi
    for x=1 : lebar
        xlama = x - sx;
        ylama = y - sy;

        if (xlama>=1) && (xlama<=lebar) && ...
           (ylama>=1) && (ylama<=tinggi)
           G(y, x) = F2(ylama, xlama);
        else
           G(y, x) = 0;
        end
    end
end

G = uint8(G);
figure(1); imshow(G);

clear all;
```

Akhir Program

Pada contoh di atas, citra digeser ke kanan sebesar 45 piksel (ditentukan melalui `sx`) dan ke atas sebesar 35 piksel (diatur melalui `sy`). Apabila `xlama` hasil perhitungan di luar jangkauan [1, lebar] atau `ylama` hasil perhitungan di luar jangkauan [1, tinggi], intensitas piksel pada posisi (y, x) diisi dengan nol (warna hitam). Posisi yang tidak berada pada posisi koordinat yang valid dalam citra lama akan diisi dengan nilai nol melalui

G(y, x) = 0;

Hasilnya diperlihatkan pada Gambar 5.2.

(a) Citra gedung asli (b) Hasil penggeseran

Gambar 5.2 Contoh penggeseran citra

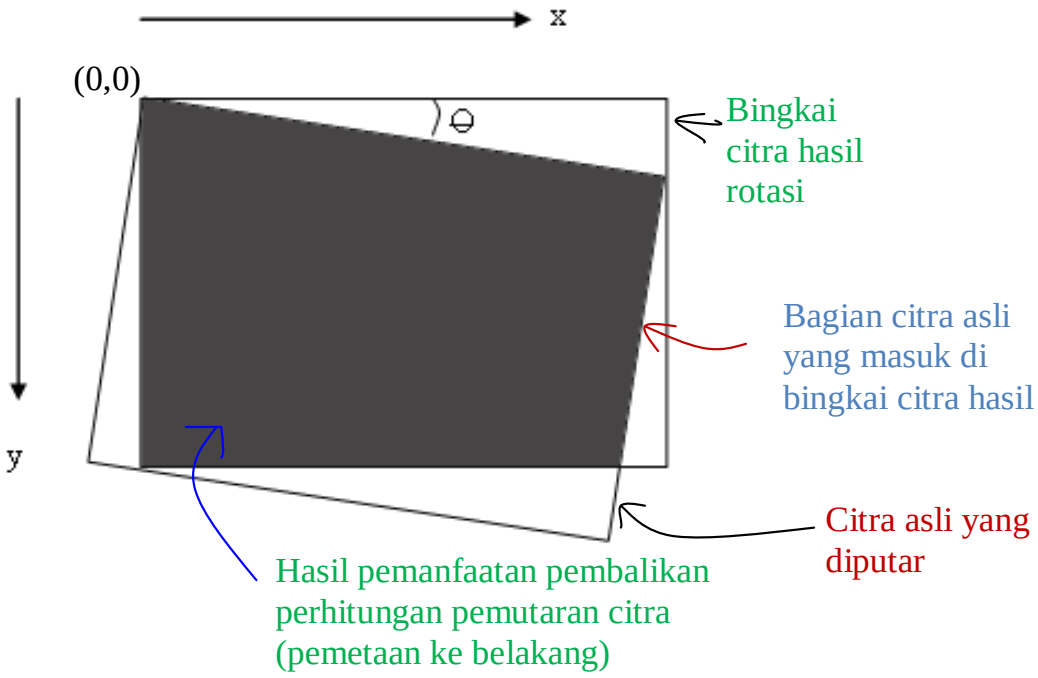
Gambar hitam di bagian kiri dan bagian atas adalah efek dari

5.3 Memutar Citra

Suatu citra dapat diputar dengan sudut θ seiring arah jarum jam atau berlawanan arah jarum jam dengan pusat putaran pada koordinat (0,0). Gambar 5.3 menjelaskan bentuk pemutaran citra. Adapun rumus yang digunakan untuk memutar citra dengan sudut θ berlawanan arah jam berupa:

$$x_{baru} = x * Cos(\theta) + y * Sin(\theta) \quad (5.3)$$

$$y_{baru} = y * Cos(\theta) - x * Sin(\theta) \quad (5.4)$$



Gambar 5.3 Pemutaran citra dengan pusat (0, 0)

Berdasarkan Persamaan 5.3 dan 5.4, pemutaran citra dengan sudut θ searah jarum jam dapat dilakukan. Caranya, dengan menggunakan x dan y sebagai posisi baru dan x_{baru} justru sebagai posisi lama. Pada saat menghitung dengan rumus di atas, apabila posisi koordinat (y_{baru} ,x_{baru}) berada di luar area [1, lebar] dan [1, tinggi], intensitas yang digunakan berupa nol. Cara inilah yang merupakan contoh pemetaan ke belakang. Implementasinya dapat dilihat berikut ini.

Program : rotasi.m

```
% ROTASI Melakukan Operasi pemutaran citra.
%     Versi 1
%     Menggunakan pendekatan pemetaan ke belakang

F = imread('c:\Image\sungai.png');
[tinggi, lebar] = size(F);

sudut = 10; % Sudut pemutaran
rad = pi * sudut/180;
cosa = cos(rad);
```

```
sina = sin(rad);
F2 = double(F);
for y=1 : tinggi
    for x=1 : lebar
        x2 = round(x * cosa + y * sina);
        y2 = round(y * cosa - x * sina);

        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)
           G(y, x) = F2(y2, x2);
        else
           G(y, x) = 0;
        end
    end
end

G = uint8(G);
figure(1); imshow(G);

clear all;
```

Akhir Program

Contoh hasil pemutaran dapat dilihat pada Gambar 5.4.

(a) Citra sungai asli (b) Hasil pemutaran

Gambar 5.4 Contoh pemutaran citra

Apa yang terjadi kalau dilaksanakan pemetaan ke depan dengan menggunakan rumus pada Persamaan 5.3 dan 5.4? Sebagaimana telah dijelaskan di depan (Subbab 5.1), cara seperti itu dapat menimbulkan lubang pada citra hasil. Artinya, akan ada piksel yang tidak terisi dengan piksel dari citra masukan. Untuk melihat efek ini, cobalah jalankan program berikut.

Program : rotasi2.m

```
% ROTASI2 Melakukan operasi pemutaran citra.
%     Versi 2
%     Menggunakan pemetaan ke depan

F = imread('c:\Image\gedung.png');
[tinggi, lebar] = size(F);

sudut = 5; % Sudut pemutaran
rad = pi * sudut/180;
cosa = cos(rad);
sina = sin(rad);
F2 = double(F);
G=zeros(tinggi, lebar);
for y=1 : tinggi
    for x=1 : lebar
        x2 = round(x * cosa - y * sina);
        y2 = round(y * cosa + x * sina);

        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)
           G(y2, x2) = F2(y, x);
        end
    end
end

G = uint8(G);
figure(1); imshow(G);

clear all;
```

Akhir Program

Hasilnya bisa dilihat pada gambar berikut.

(a) Citra gedung asli

(b) Hasil pemutaran yang menimbulkan lubang-lubang (bintik-bintik gelap) pada citra

Gambar 5.5 *Efek pemetaan ke depan*

Perhatikan pada Gambar 5.5(b). Titik-titik hitam pada citra adalah efek lubang yang memerlukan penanganan lebih lanjut untuk menghilangkannya.

5.4 Interpolasi Piksel

Hasil pemutaran citra menggunakan rotasi.m menimbulkan efek bergerigi pada objek citra. Hal itu diakibatkan oleh penggunaan nilai intensitas didasarkan pada piksel tetangga terdekat, yang dilakukan melalui:

```
x2 = round(x * cosa + y * sina);
y2 = round(y * cosa - x * sina);
```

Penggunaan fungsi **round** (pembulatan ke atas) merupakan upaya untuk menggunakan intensitas piksel terdekat. Alternatif lain dilakukan dengan menggunakan **floor** (pembulatan ke bawah). Gambar berikut menunjukkan keadaan tersebut ketika hasil pada Gambar 5.4 (b) diperbesar.

Gambar 5.6 Efek bergerigi pada citra hasil pemutaran memberikan citra terlihat tidak mulus

Keadaan seperti itu dapat diperhalus melalui interpolasi piksel.

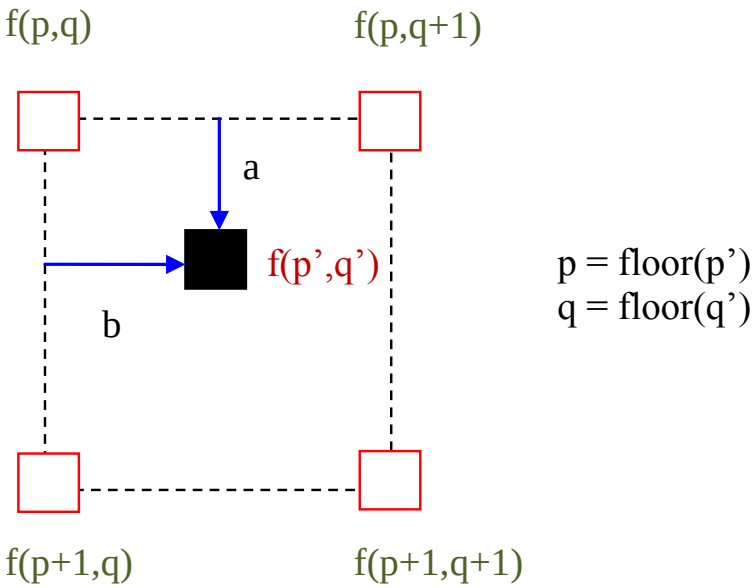
Idenya seperti berikut. Misalnya, hasil perhitungan menghasilkan

xlama = 47,09
ylama = 59,85

Pada contoh di depan, piksel yang digunakan berposisi (60, 47) dengan melakukan pembulatan ke atas. Namun, sesungguhnya bisa saja piksel yang digunakan adalah yang berada pada posisi (59, 47) jika dilakukan pembulatan ke bawah. Hal yang perlu diketahui, kemungkinan yang terjadi dapat melibatkan empat buah piksel. Gambar 5.7 menunjukkan keadaan ini. Oleh karena itu, nilai intensitas yang digunakan dapat melibatkan keempat piksel tersebut.

Catatan

Jika ukuran piksel, yaitu di bawah ukuran kepekaan mata pemandang, spek zig-zag tidak akan terlihat. Namun, bila pemutran citra terjadi berulang secara serial, cacat gerigi akan membesar.



Gambar 5.7 Model pendekatan bilinear interpolation

Perhatikan bahwa f(p'. q') mempunyai empat piksel terdekat berupa f(p,q), f(p,q+1), f(p+1,q), dan f(p+1,q+1).

Pratt (2001) menunjukkan cara menghitung nilai intensitas yang digunakan untuk suatu piksel berdasarkan empat piksel. Rumusnya sebagai berikut:

$$f(p',q') = (1-a)[(1-b)f(p,q) + b\,f(p,q+1)] + a[(1-b)f(p+1,q) + b\,f(p+1,q+1)] \quad (5.5)$$

Dalam hal ini, a dan b dihitung melalui:

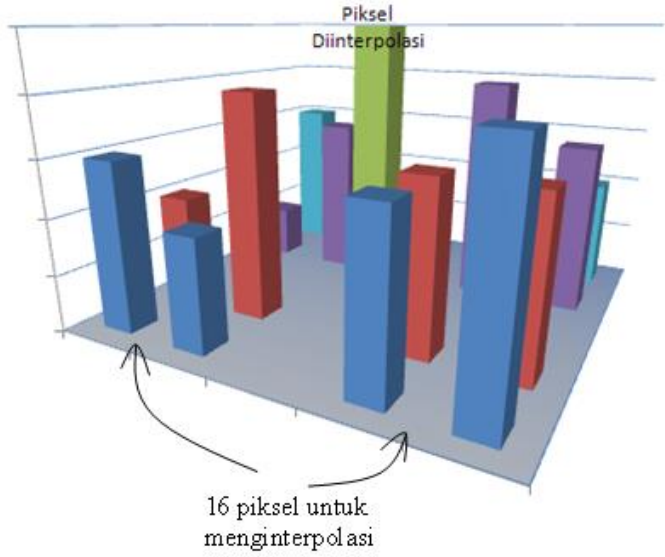
$$a = p' - p \quad (5.6)$$

$$b = q' - q \qquad (5.7)$$

Rumus dalam Persamaan 5.5 itulah yang disebut sebagai *bilinear interpolation*.

Catatan

Selain *bilinear interpolation*, sebenarnya terdapat beberapa cara untuk melakukan interpolasi. Dua cara lain yang populer yaitu *bicubic interpolation*, yang menggunakan 16 piksel tetangga untuk memperoleh interpolasi intensitas piksel dan bikuadratik yang melibatkan 9 piksel terdekat.



Contoh program yang menggunakan interpolasi bilinear untuk mendapatkan intensitas piksel dapat dilihat di bawah ini.

Program : rotasi3.m

```
% ROTASI3 Melakukan operasi pemutaran citra.
%     Versi 3 - menggunakan bilinear interpolation

F = imread('c:\Image\gedung.png');
[tinggi, lebar] = size(F);
```

```
sudut = 15; % Sudut pemutaran
rad = pi * sudut/180;
cosa = cos(rad);
sina = sin(rad);
F2 = double(F);
for y=1 : tinggi
    for x=1 : lebar
        x2 = x * cosa + y * sina;
        y2 = y * cosa - x * sina;

        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)
           % Lakukan interpolasi bilinear
           p = floor(y2);
           q = floor(x2);
           a = y2-p;
           b = x2-q;

           if (x2 == lebar) || (y2 == tinggi)
              G(y, x) = F(floor(y2), floor(x2));
           else
              intensitas = (1-a)*((1-b)*F(p,q) +  ...
              b * F(p, q+1)) +      ...
              a *((1-b)* F(p+1, q) + ...
              b * F(p+1, q+1));

              G(y, x) = intensitas;
           end
        else
           G(y, x) = 0;
        end
    end
end

G = uint8(G);
figure(1); imshow(G);

clear all;
```

Akhir Program

Gambar 5.8 memperlihatkan perbedaan hasil antara pemutaran citra yang menggunakan pendekatan interpolasi bilinear dan yang tidak.

(a) Tanpa interpolasi

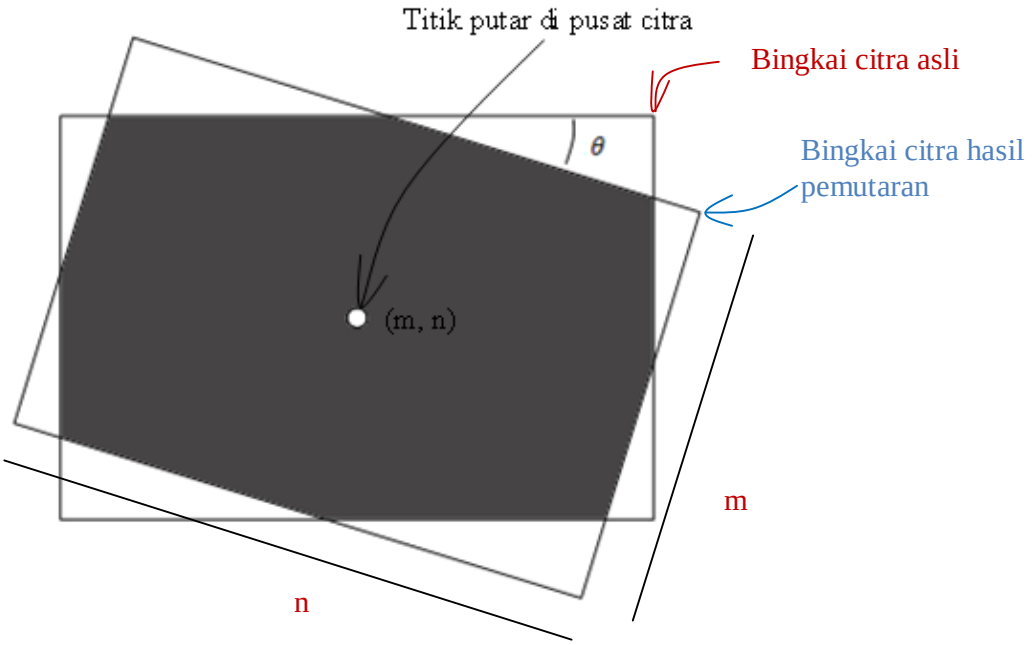
(b) Dengan interpolasi

Gambar 5.8 Perbandingan efek penggunaan interpolasi bilinear

Terlihat bahwa hasil yang menggunakan interpolasi bilinear lebih halus. Namun, tentu saja, kehalusan tersebut harus dibayar dengan waktu komputasi yang lebih lama.

5.5 Memutar Berdasarkan Sebarang Koordinat

Operasi pemutaran citra dapat dilakukan dengan pusat di mana saja; tidak harus dari (0, 0). Gambar 5.9 memperlihatkan keadaan ini.



Gambar 5.9 Pemutaran citra melalui titik pusat citra

Rumus untuk melakukan pemutaran berlawanan arah jarum jam sebesar θ yang diperlihatkan pada Gambar 5.9 diperoleh melalui pemodifikasian Persamaan 5.3 dan 5.4:

$$x_{baru} = (x - n) * Cos(\theta) + (y - m) * Sin(\theta) + n \quad (5.8)$$

$$y_{baru} = (y - m) * Cos(\theta) - (x - n) * Sin(\theta) + m \quad (5.9)$$

Untuk kepentingan pemutaran citra sejauh θ searah jarum jam, intensitas piksel (y, x) dapat diperoleh melalui intensitas pada piksel (y_{baru}, x_{baru}) yang tertera pada Persamaan 5.8 dan 5.9. Implementasi program dapat dilihat pada contoh berikut.

Program : rotasi4.m

```
% ROTASI4 Melakukan operasi pemutaran citra.
%     Versi 4 - pusat putaran pada pusat citra
```

```
F = imread('c:\Image\gedung.png');
[tinggi, lebar] = size(F);

sudut = 5; % Sudut pemutaran
rad = pi * sudut/180;
cosa = cos(rad);
sina = sin(rad);
F2 = double(F);

m = floor(tinggi / 2);
n = floor(lebar / 2);

for y=1 : tinggi
    for x=1 : lebar
        x2 = (x-n) * cosa + (y-m) * sina + n;
        y2 = (y-m) * cosa - (x-n) * sina + m;
        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)
           % Lakukan interpolasi bilinear
           p = floor(y2);
           q = floor(x2);
           a = y2-p;
           b = x2-q;

           if (x2==lebar) || (y2 == tinggi)
              G(y, x) = F(y2, x2);
           else
              intensitas = (1-a)*((1-bF(p,q) +  ...
              b * F(p, q+1)) +      ...
              a *((1-b)* F(p+1, q) + ...
              b * F(p+1, q+1));

              G(y, x) = intensitas;
           end
        else
           G(y, x) = 0;
        end
    end
end

G = uint8(G);
figure(1); imshow(G);

clear all;
```

Akhir Program

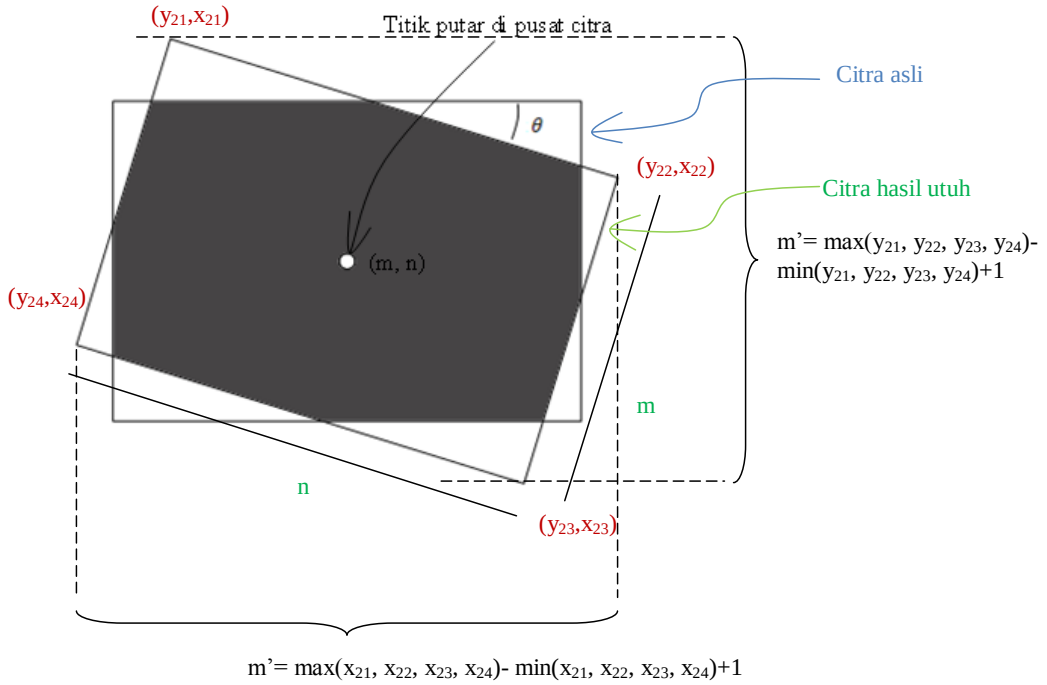
Contoh di atas menggunakan interpolasi bilinear. Hasilnya dapat dilihat pada Gambar 5.10.

(a) Citra gedung asli (b) Hasil pemutaran 5^o

Gambar 5.10 Pemutaran melalui titik pusat citra

5.6 Memutar Citra Secara Utuh

Pada seluruh contoh yang telah diberikan, ada bagian gambar yang hilang ketika pemutaran dilaksanakan. Namun, adakalanya dihendaki agar pemutaran citra tidak membuat ada bagian citra asli hilang. Untuk keperluan ini, ukuran citra hasil pemutaran harus diubah sesuai dengan sudut putaran. Dalam hal ini, Persamaan 5.8 dan 5.9 digunakan untuk menentukan keempat pojok gambar semula. Adapun lebar dan tinggi gambar hasil pemutaran dengan menghitung nilai terkecil dan terbesar dari koordinat keempat pojok hasil pemutaran. Untuk lebih jelasnya, lihat Gambar 5.11.



Gambar 5.11 Penentuan lebar dan tinggi citra hasil pemutaran

Implementasi pemutaran citra secara utuh diperlihatkan pada program rotasi5.m.

Program : rotasi5.m

```
% ROTASI5 Melakukan operasi pemutaran citra.
%     Versi 5
%     Memutar dengan hasil utuh

F = imread('c:\Image\gedung.png');
[tinggi, lebar] = size(F);

sudut = 45; % Sudut pemutaran
rad = pi * sudut/180;
cosa = cos(rad);
sina = sin(rad);

x11 = 1;      y11 = 1;
x12 = lebar;  y12 = 1;
x13 = lebar;  y13 = tinggi;
x14 = 1;      y14 = tinggi;
```

```
m = floor(tinggi/2);
n = floor(lebar/2);

% Menentukan pojok
x21 = ((x11-n) * cosa + (y11-m) * sina + n);
y21 = ((y11-m) * cosa - (x11-n) * sina + m);

x22 = ((x12-n) * cosa + (y12-m) * sina + n);
y22 = ((y12-m) * cosa - (x12-n) * sina + m);

x23 = ((x13-n) * cosa + (y13-m) * sina + n);
y23 = ((y13-m) * cosa - (x13-n) * sina + m);

x24 = ((x14-n) * cosa + (y14-m) * sina + n);
y24 = ((y14-m) * cosa - (x14-n) * sina + m);

ymin = min([y21 y22 y23 y24]);
xmin = min([x21 x22 x23 x24]);

ymak = max([y21 y22 y23 y24]);
xmak = max([x21 x22 x23 x24]);

lebar_baru = xmak - xmin + 1;
tinggi_baru = ymak - ymin + 1;
tambahan_y = floor((tinggi_baru-tinggi)/2);
tambahan_x = floor((lebar_baru-lebar)/2);
F2=zeros(tinggi_baru, lebar_baru);
for y=1 : tinggi
   for x=1 : lebar
       F2(y+tambahan_y, x+tambahan_x) = F(y, x);
   end
end

figure(1);
imshow( uint8(F2));

% Putar citra
m = floor(tinggi_baru/2);
n = floor(lebar_baru/2);

for y=1 : tinggi_baru
    for x=1 : lebar_baru
        x2 = round((x-n) * cosa + (y-m) * sina + n);
        y2 = round((y-m) * cosa - (x-n) * sina + m);

        if (x2>=1) && (x2<=lebar_baru) && ...
           (y2>=1) && (y2<=tinggi_baru)
           G(y, x) = F2(y2,x2);
        else
           G(y,x) = 0;
        end
    end
end

figure(2);
G = uint8(G);

imshow(G);
```

```
clear all;
```

Akhir Program

Hasil pemutaran gambar dengan menggunakan rotasi5.m ditunjukkan pada Gambar 5.12.

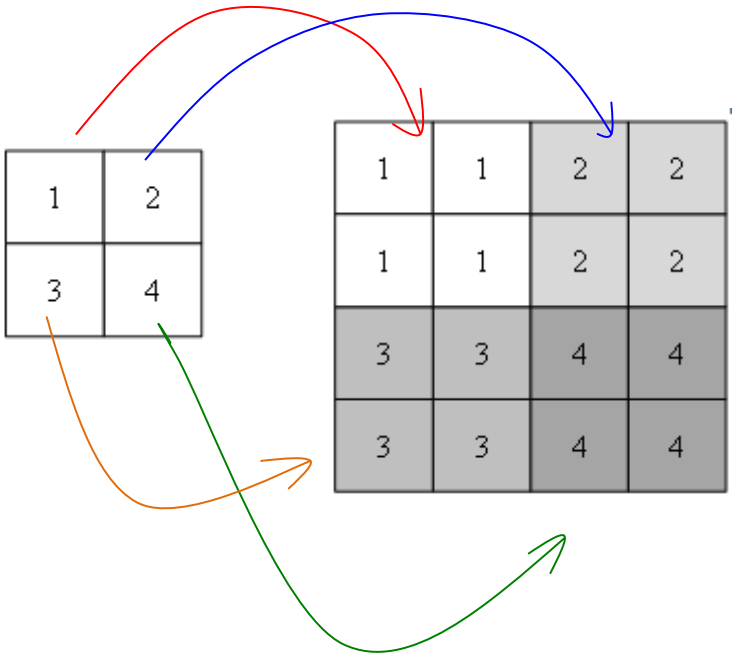
(a) Pengubahan ukuran citra Supaya kalau diputar tidak ada yang hilang

(b) Hasil pemutaran citra dengan sudut 45°

Gambar 5.12 Pemutaran citra secara utuh

5.7 Memperbesar Citra

Suatu citra dapat diperbesar dengan membuat setiap piksel menjadi beberapa piksel. Gambar 5.13 memberikan contoh cara memperbesar citra.



Gambar 5.13 Cara memperbesar citra

Pada contoh di atas pembesaran pada arah vertikal dan horizontal sebesar 2 kali. Berikut adalah fungsi yang memperlihatkan cara perbesaran tersebut.

Program : perbesar.m

```
function G = perbesar(berkas, sy, sx)
% PERBESAR Melakukan operasi pembesaran citra.
%     Masukan: berkas = nama berkas image
%              sy : skala pembesaran pada sumbu Y
%              sx : skala pembesaran pada sumbu X
%
%     Versi 1

F = imread(berkas);
[tinggi, lebar] = size(F);

tinggi_baru = tinggi * sy;
lebar_baru = lebar * sx;

F2 = double(F);
for y=1 : tinggi_baru
    y2 = ((y-1) / sy) + 1;
    for x=1 : lebar_baru
        x2 = ((x-1) / sx) + 1;
        G(y, x) = F(floor(y2), floor(x2));
```

```
    end
end

G = uint8(G);
```

Akhir Program

Perlu diketahui, tinggi dan lebar citra keluaran dihitung berdasarkan

```
tinggi_baru = tinggi * sy;
lebar_baru = lebar * sx;
```

Kemudian,

```
y2 = ((y-1) / sy) + 1;
```

digunakan untuk memperoleh nilai `y2` yang berkisar antara 1 sampai dengan lebar citra asli. Hal yang serupa dilakukan untuk `x2` yang dilaksanakan melalui

```
x2 = ((x-1) / sx) + 1;
```

Berdasar fungsi perbesar di atas, dapat diberikan perintah seperti berikut:

```
>> Img = perbesar('C:\Image\lena128.png', 3, 3);
```

Pada perintah di atas, citra lena12.png diperbesar tiga kali baik pada arah vertikal maupun horizontal.

Selanjutnya, hasil perbesaran ditampilkan melalui

```
>> imshow(Img);
```

Hasilnya dapat dilihat pada Gambar 5.14.

(a) Citra lena 128x128

(b) Pembesaran 3x tanpa interpolasi

Gambar 5.14 Contoh pembesaran citra

Untuk memperhalus hasil perbesaran citra, interpolasi piksel perlu dilakukan. Contoh dapat dilihat pada kode berikut.

Program : perbesar2.m

```
function G = perbesar2(berkas, sy, sx)
% PERBESAR2 Melakukan operasi pembesaran citra
%     dengan interpolasi.
%     Masukan: berkas = nama berkas image
%              sy : skala pembesaran pada sumbu Y
%              sx : skala pembesaran pada sumbu X
%
% Versi 2

F = imread(berkas);
[tinggi, lebar] = size(F);

tinggi_baru = round(tinggi * sy);
lebar_baru = round(lebar * sx);

F2 = double(F);
for y=1 : tinggi_baru
    y2 = (y-1) / sy + 1;
    for x=1 : lebar_baru
```

```
        x2 = (x-1) / sx + 1;
        % Lakukan interpolasi bilinear
        p = floor(y2);
        q = floor(x2);
        a = y2-p;
        b = x2-q;

        if (floor(x2)==lebar) || …
            (floor(y2) == tinggi)
            G(y, x) = F(floor(y2), floor(x2));
        else
            intensitas = (1-a)*((1-b)*F(p,q) +  ...
            b * F(p, q+1)) +      ...
            a *((1-b)* F(p+1, q) + ...
            b * F(p+1, q+1));

            G(y, x) = intensitas;
        end
    end
end

G = uint8(G);
```

Akhir Program

Penghalusan citra keluaran dilakukan melalui interpolasi bilinear, seperti yang telah dibahas di Subbab 5.4.

Untuk melihat hasil interpolasi pada pembesaran citra, dapat diberikan perintah seperti berikut:

```
>> Img = Perbesar2('C:\Image\lena128.png', 4, 4); ↵
>> imshow(Img); ↵
```

Hasilnya dapat dilihat pada Gambar 5.15.

(a) Citra lena 128x128

(b) Pembesaran 3x

Gambar 5.15 Contoh perbesaran citra dengan interpolasi

Cobalah untuk membandingkan hasil di atas dengan hasil pada Gambar 5.14.

5.8 Memperkecil Citra

Bagaimana kalau ingin memperkecil citra? Secara prinsip, pengecilan citra berarti mengurangi jumlah piksel. Algoritma yang digunakan untuk mewujudkan perbesar.m maupun perbesar2.m dapat digunakan untuk keperluan ini dengan m berupa bilangan pecahan seperti 1/2, ¼, 1/8, dan seterusnya. Contoh:

```
>> Img = perbesar2('C:\Image\lena256.png', 0.5, 0.5); ↵
>> imshow(Img); ↵
```

Hasilnya dapat dilihat pada Gambar 5.16.

(a) Citra lena 256x256

(b) Hasil pengecilan 0,5 x pada arah vertikal dan horisontal

Gambar 5.16 Contoh pengecilan dengan interpolasi

5.9 Perbesaran dengan Skala Vertikal dan Horizontal Berbeda

Fungsi `perbesar` dan `perbesar2` dapat digunakan untuk melakukan perbesaran/pengecilan dengan skala horizontal dan vertikal yang berbeda. Sebagai contoh, dapat diberikan perintah seperti berikut:

```
>> Img = perbesar2('C:\Image\gedung.png', 0.5, 2.5); ↵
>> imshow(Img); ↵
```

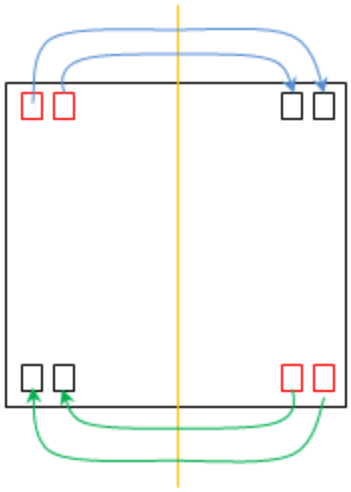
Hasilnya ditunjukkan pada Gambar 5.17.

Gambar 5.17 Gedung diperbesar 1/2 kali pada arah vertikal dan 2,5 kali pada arah horizontal

5.10 Pencerminan Citra

Pencerminan yang umum dilakukan berupa pencerminan secara vertikal dan pencerminan secara horizontal. Pencerminan secara horizontal dilakukan dengan menukarkan dua piksel yang berseberangan kir-kanan, sebagaimana diperlihatkan

pada Gambar 5.18. Algoritma untuk menangani pencerminan secara horizontal diperlihatkan Algoritma 5.1.



Gambar 5.18 *Pencerminan secara horizontal*

ALGORITMA 5.1 – Mencerminkan gambar secara horizontal

Masukan:
- f (M,N): Citra masukan berukuran M baris dan N kolom

Keluaran:
- g (M, N): Hasil citra yang telah dicerminkan secara horizontal

1. FOR baris ← 1 TO M
2. FOR kolom ← 1 TO N
3. g(baris, kolom) ← f(N – baris + 1, kolom)
4. END-FOR
5. END-FOR

Implementasinya ditunjukkan pada program berikut.

Program : cerminh.m

```
function G = cerminh(F)
% CERMINH Berfungsi untuk mencerminkan citra
%      secara horizontal
```

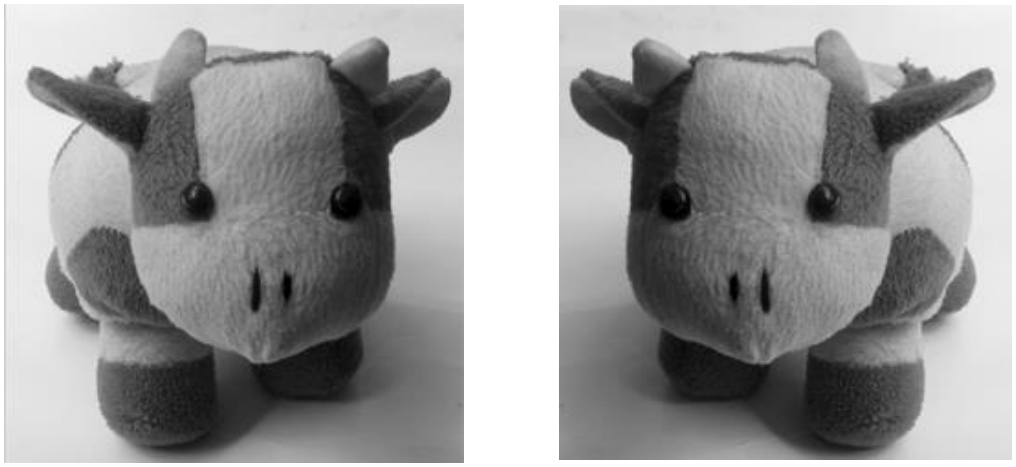
```
%     Masukan: F = Citra berskala keabuan

[tinggi, lebar] = size(F);

for y=1 : tinggi
    for x=1 : lebar
        x2 = lebar - x + 1;
        y2 = y;

        G(y, x) = F(y2, x2);
    end
end

G = uint8(G);
```

Akhir Program

Contoh pemakaian fungsi `cerminh`:

```
>> F = imread('C:\Image\boneka.png');
>> G = cerminh(F); imshow(G)
```

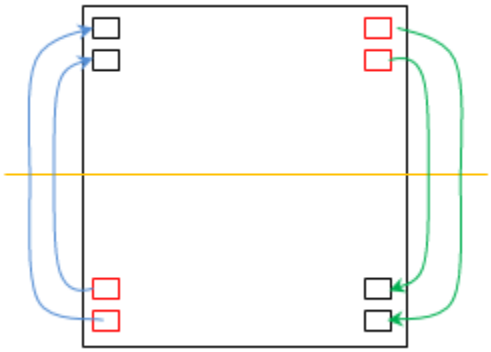
Contoh pencerminan gambar secara horizontal ditunjukkan pada Gambar 5.19.



(a) Citra boneka.tif (b) Pencerminan secara horizontal

Gambar 5.19 Pencerminan secara horizontal

Pencerminan secara vertikal dilakukan dengan menukarkan dua piksel yang berseberangan atas-bawah, sebagaimana diperlihatkan pada Gambar 5.20. Algoritma untuk menangani pencerminan secara horizontal diperlihatkan Algoritma 5.2.



Gambar 5.20 Pencerminan secara vertikal

ALGORITMA 5.2 – Mencerminkan gambar secara vertikal

Masukan:

- f (M,N): Citra masukan berukuran M baris dan N kolom

Keluaran:

- g (M, N): Hasil citra yang telah dicerminkan secara horizontal

1. FOR baris ← 1 TO M
2. FOR kolom ← 1 TO N
3. g(baris, kolom) ← f(baris, N – kolom + 1)
4. END-FOR
5. END-FOR

Implementasinya ditunjukkan pada program berikut.

Program : cerminv.m

```
function G = cerminv(F)
% CERMINV Berfungsi untuk mencerminkan citra
%      secara vertikal
%      Masukan: F = Citra berskala keabuan
```

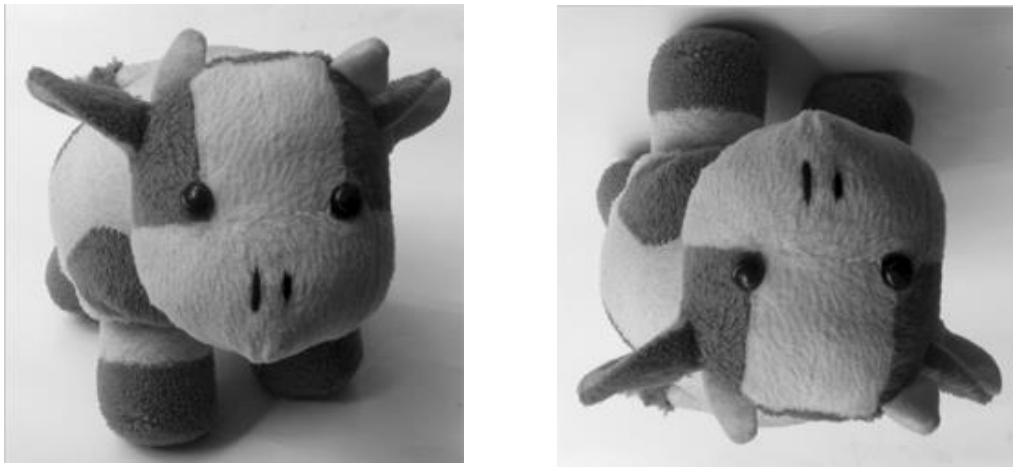
```
[tinggi, lebar] = size(F);

for y=1 : tinggi
    for x=1 : lebar
        x2 = x;
        y2 = tinggi - y + 1;

        G(y, x) = F(y2, x2);
    end
end

G = uint8(G);
```

Akhir Program

Contoh pemakaian fungsi `cerminv`:

```
>> F = imread('C:\Image\boneka.png'); ↵
>> G = cerminv(F); imshow(G) ↵
```

Contoh pencerminan gambar secara vertikal ditunjukkan pada Gambar 5.21.



(a) Citra boneka.tif (b) Pencerminan secara vertikal

Gambar 5.21 Pencerminan secara vertikal

Catatan

Di beberapa *software*, pencerminan secara horizontal justru dinamakan **vertical flip**.

5.11 Transformasi *Affine*

Transformasi *affine* adalah transformasi linear yang menyertakan penskalaan, pemutaran, penggeseran, dan *shearing* (pembengkokan). Transformasi *affine* dapat dinyatakan dengan persamaan seperti berikut:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.10}$$

Persamaan di atas dapat ditulis pula menjadi seperti berikut:

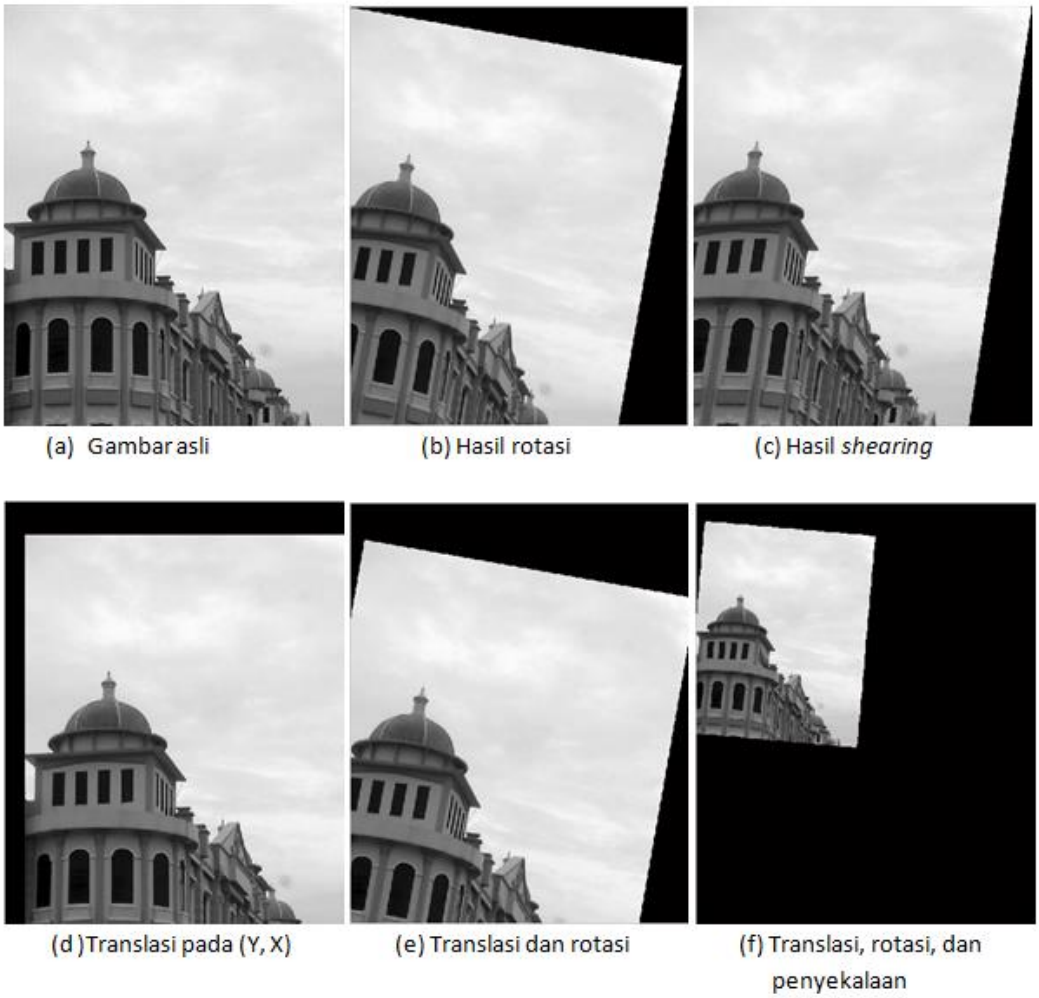
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.11}$$

Berdasarkan persamaan di atas, terlihat bahwa transformasi *affine* memiliki enam derajat kebebasan: dua untuk translasi (t_x dan t_y) dan empat buah untuk rotasi, penskalaan, *stretching*, dan *shearing* (a_{11}, a_{12}, a_{21}, dan a_{22}).

Tabel 5.1 menunjukkan koefisien yang digunakan dalam matriks di depan untuk menyatakan operasi dasar penskalaan, rotasi, translasi, dan pembengkokan. Tentu saja, keenam koefisien tersebut dapat diatur secara bebas untuk mendapatkan transformasi *affine*. Untuk melakukan penggabungan dua operasi dasar, koefisien yang sama dari dua jenis transformasi dapat dikalikan. Contoh dapat dilihat pada Gambar 5.22.

Tabel 5.1 Koefisien untuk menentukan efek penskalaan, rotasi. translasi, dan pembengkokan

Transformasi	a_{11}	a_{12}	a_{21}	a_{22}	t_x	t_y
Translasi sebesar (y, x)	1	0	0	1	x	Y
Rotasi sebesar θ	cos θ	sin θ	-sin θ	cos θ	0	0
Penyekalaan sebesar s	s	0	0	S	0	0
Pembengkokan secara vertikal sebesar s	1	S	0	1	0	0
Pembengkokan secara horizontal sebesar s	1	0	s	1	0	0



Gambar 5.22 Contoh transformasi linear yang mencakup rotasi, penyekalaan, dan *affine*

Fungsi berikut berguna untuk mewujudkan transformasi *affine*.

Program : taffine.m

```
function G = taffine(F, a11, a12, a21, a22, tx, ty)
% TAFFINE Digunakan untuk melakukan transformasi affine.
%     Masukan: F = Citra berskala keabuan
%              a11, a12, a21, a22, tx, ty = mengatur
%              transformasi affine

[tinggi, lebar] = size(F);

for y=1 : tinggi
    for x=1 : lebar
        x2 = a11 * x + a12 * y + tx;
        y2 = a21 * x + a22 * y + ty;
        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)

           % Lakukan interpolasi bilinear
           p = floor(y2);
           q = floor(x2);
           a = y2-p;
           b = x2-q;

           if (floor(x2)==lebar) || ...
              (floor(y2) == tinggi)
              G(y, x) = F(floor(y2), floor(x2));
           else
              intensitas = (1-a)*((1-b)*F(p,q) +  ...
              b * F(p, q+1)) +      ...
              a *((1-b)* F(p+1, q) + ...
              b * F(p+1, q+1));

              G(y, x) = intensitas;
           end
        else
           G(y, x) = 0;
        end
    end
end

G = uint8(G);
```

Akhir Program

Contoh penggunaan fungsi `taffine` untuk melakukan pembengkokan:

```
>> F = imread('C:\Image\gedung.png'); ↵
>> G = taffine(F,1,0.15,0,1,0,0); ↵
>> imshow(G) ↵
```

Contoh berikut digunakan untuk memutar gambar:

```
>> rad = 10 * pi / 180;
>> G = taffine(F,cos(rad),sin(rad), …
                -sin(rad),cos(rad),0,0);
>> imshow(G) ↵
```

Contoh penggabungan rotasi dan translasi:

```
>> G = taffine(F,cos(rad),sin(rad),-sin(rad), …↵
               cos(rad),-30,-50); ↵
```

Contoh penggabungan rotasi, penskalaan, dan translasi:

```
>> G = taffine(F,2 * cos(rad),sin(rad),-sin(rad), …↵
         2 * cos(rad),-30,-50); ↵
```

Perlu diketahui, angka seperti 2 di depan cos(rad) menyatakan bahwa hasilnya adalah kebalikannya, yaitu ½ kalinya.

5.12 Efek *Ripple*

Efek *ripple* (riak) adalah aplikasi transformasi citra yang membuat gambar terlihat bergelombang. Efek riak dapaat dibuat baik pada arah x maupun y. Transformasinya seperti berikut:

$$x = x' + a_x \, sin\frac{2\pi y'}{T_x} \tag{5.12}$$

$$y = y' + a_y \, sin\frac{2\pi x'}{T_y} \tag{5.13}$$

Dalam hal ini, a_x dan a_y menyatakan amplitudo riak gelombang sinus, sedangkan T_x dan T_y menyatakan periode gelombang sinus.

Implementasi efek gelombang dapat dilihat di bawah ini.

Program : ripple.m

```
function G = ripple(F, ax, ay, tx, ty)
% RIPPLE Berfungsi untuk melakukan transformasi 'ripple'.

dimensi = size(F);
tinggi = dimensi(1);
lebar = dimensi(2);
for y=1 : tinggi
    for x=1 : lebar
        x2 = x + ax * sin(2 * pi * y / tx);
        y2 = y + ay * sin(2 * pi * x / ty);
        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)

           % Lakukan interpolasi bilinear
           p = floor(y2);
           q = floor(x2);
           a = y2-p;
           b = x2-q;

           if (floor(x2)==lebar) || ...
              (floor(y2) == tinggi)
              G(y, x) = F(floor(y2), floor(x2));
           else
              intensitas = (1-a)*((1-b)*F(p,q) +  ...
              b * F(p, q+1)) +      ...
              a *((1-b)* F(p+1, q) + ...
              b * F(p+1, q+1));

              G(y, x) = intensitas;
           end
        else
           G(y, x) = 0;
        end
    end
end

G = uint8(G);
```

Akhir Program

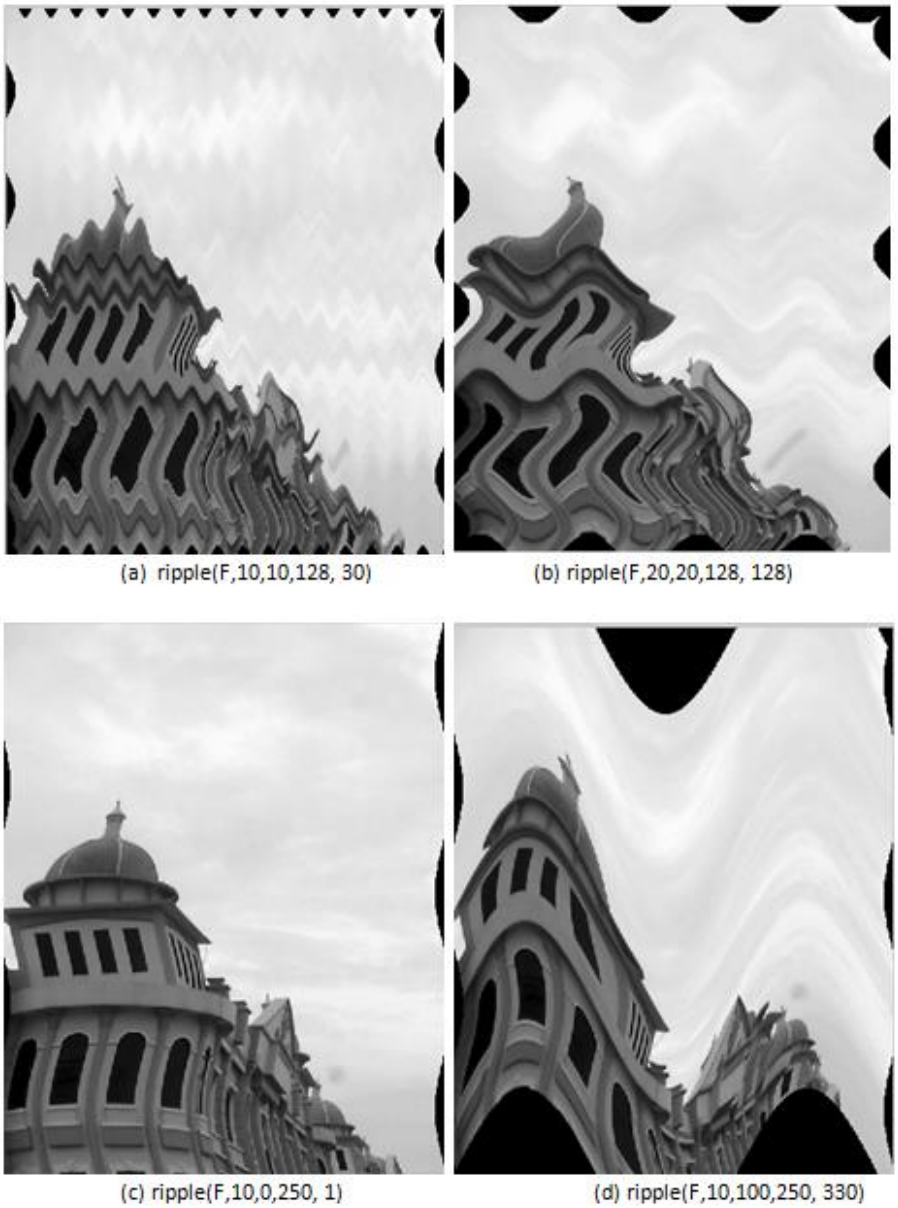
Contoh penggunaan fungsi `ripple`:

```
>> F = imread('C:\image\gedung.png'); ↵
>> G = ripple(F,10,15,120, 250); ↵
>> imshow(G) ↵
```

Pada contoh di atas, amplitude gelombang sinus yang digunakan berupa 10 dan 15, sedangkan periode yang digunakan 120 dan 250. Contoh hasil perintah di atas ditunjukkan pada Gambar 5.23.

Gambar 5.23 Contoh hasil efek ripple

Beberapa contoh yang lain dapat dilihat pada Gambar 5.24.



Gambar 5.24 Berbagai hasil efek ripple

5.13 Efek *Twirl*

Transformasi *twirl* (olak atau puntiran) dilakukan dengan memutar citra berdasarkan titik pusat citra, tetapi tidak bersifat linear. Salah satu varian bentuk transformasinya, yang diadaptasi dari Burger & Burge (2008), sebagai berikut:

$$x' = x_c + r\cos(\beta) \quad (5.14)$$

$$y' = y_c + r\sin(\beta) \quad (5.15)$$

dengan

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2} \quad (5.16)$$

$$\beta = ArcTan(d_y, d_x) + \alpha(r_{maks} - r)/r_{maks} \quad (5.17)$$

Contoh berikut menggunakan r_{maks} sebesar ½ diagonal citra dan α sebesar 43°.

Program : twirl.m

```
function G = twirl(F)
% TWIRL Berfungsi untuk melakukan transformasi 'twirl'

dimensi = size(F);
tinggi = dimensi(1);
lebar = dimensi(2);
xc = round(lebar / 2);
yc = round(tinggi / 2);

alpha = 43 * pi / 180;
rmaks = 0.5 * sqrt(xc^2 + yc ^ 2); % 1/2 diagonal citra

for y=1 : tinggi
    for x=1 : lebar
        r = sqrt((x-xc)^2+(y-yc)^2);
        beta = atan2(y-yc, x-xc) + …
               alpha * (rmaks - r) / rmaks;
        x2 = xc + r * cos(beta);
        y2 = yc + r * sin(beta);


        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)

           % Lakukan interpolasi bilinear
           p = floor(y2);
           q = floor(x2);
           a = y2-p;
           b = x2-q;

           if (floor(x2)==lebar) || ...
              (floor(y2) == tinggi)
              G(y, x) = F(floor(y2), floor(x2));
           else
              intensitas = (1-a)*((1-b)*F(p,q) +  ...
              b * F(p, q+1)) +      ...
              a *((1-b)* F(p+1, q) + ...
              b * F(p+1, q+1));

              G(y, x) = intensitas;
```

```
            end
        else
            G(y, x) = 0;
        end
    end
end

G = uint8(G);
```

Akhir Program

Contoh penggunaan fungsi `twirl`:

```
>> F = imread('C:\Image\kotatua.png');
>> G = swirl(F); imshow(G)
```

Hasil ditunjukkan pada Gambar 5.25.

(a) Citra kotatua (b) Hasil transformasi twirl

Gambar 5.25 Efek transformasi *twirl*

5.14 Transformasi Spherical

Transformasi *spherical* memberikan efek bulatan (bola), seperti melihat gambar menggunakan lensa pembesar. Bagian tengah terlihat membesar. Hal seperti itu diperoleh dengan menggunakan transformasi seperti berikut.

$$x' = \begin{cases} x - z * \tan(b_x), & jika\ r \leq r_{maks} \\ x, & jika\ r > r_{maks} \end{cases} \qquad (5.18)$$

$$y' = \begin{cases} y - z * \tan(b_y), & jika\ r \leq r_{maks} \\ y, & jika\ r > r_{maks} \end{cases} \quad (5.19)$$

dengan

$$b_x = \left(1 - \frac{1}{\rho}\right) * arc\ \sin\left(\frac{x - x_c}{\sqrt{(x-x_c)^2 + z^2}}\right) \quad (5.20)$$

$$b_y = \left(1 - \frac{1}{\rho}\right) * arc\ \sin\left(\frac{y - y_c}{\sqrt{(y-y_c)^2 + z^2}}\right) \quad (5.21)$$

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2} \quad (5.22)$$

$$z = \sqrt{r_{maks}{}^2 + r^2} \quad (5.23)$$

$$r_{maks} = x_c \quad (5.24)$$

$$x_c = lebar/2 \quad (5.25)$$

$$y_c = tinggi/2 \quad (5.26)$$

Perlu diketahui, ρ disebut indeks refraksi atau indeks pantulan.

Implementasi transformasi *spherical* dapat dilihat pada program berikut.

Program : spheri.m

```
function G = spheri(F, rho)
% SPHERI Berfungsi untuk melakukan transformasi 'spherical'

dimensi = size(F);
tinggi = dimensi(1);
lebar = dimensi(2);
xc = round(lebar / 2);
yc = round(tinggi / 2);

rmaks = xc;  % 1/2 lebar gambar

for y=1 : tinggi
    for x=1 : lebar
        r = sqrt((x-xc)^2+(y-yc)^2);
        z = sqrt(rmaks^2-r^2);
        bx = (1 - 1/rho) * asin((x-xc)/...
             sqrt((x-xc)^2+z^2));
        by = (1 - 1/rho) * asin((y-yc)/...
```

```
            sqrt((y-yc)^2+z^2));
        if r <= rmaks
            x2 = x - z * tan(bx);
            y2 = y - z * tan(by);
        else
            x2 = x;
            y2 = y;
        end

        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)

           % Lakukan interpolasi bilinear
           p = floor(y2);
           q = floor(x2);
           a = y2-p;
           b = x2-q;

           if (floor(x2)==lebar) || ...
              (floor(y2) == tinggi)
              G(y, x) = F(floor(y2), floor(x2));
           else
              intensitas = (1-a)*((1-b)*F(p,q) +  ...
              b * F(p, q+1)) +      ...
              a *((1-b)* F(p+1, q) + ...
              b * F(p+1, q+1));

              G(y, x) = intensitas;
           end
        else
           G(y, x) = 0;
        end
    end
end

G = uint8(G);
```

Akhir Program

Pemakaian skrip di atas dapat dilihat pada contoh berikut:

```
>> F = imread('C:\Image\kotatua.png'); ↵
>> G = spheri(F, 1.8); imshow(G) ↵
```

Hasil ditunjukkan pada Gambar 5.26.

(a) citra kotatua (b) Hasil transformasi spherical

Gambar 5.26 Transformasi *spherical*

5.15 Transformasi bilinear

Transformasi bilinear mempunyai fungsi pemetaan seperti berikut:

$$x' = a_1 x + a_2 y + a_3 xy + a_4 \quad (5.27)$$

$$y' = b_1 x + b_2 y + b_3 xy + b_4 \quad (5.28)$$

Transformasi ini termasuk dalam transformasi nonlinear mengingat terdapat pencampuran xy. Implementasi dalam bentuk program dapat dilihat berikut ini.

Program : tbilin.m

```
function G = tbilin(F, a1, a2, a3, a4, b1, b2, b3, b4)
% Fungsi untuk melakukan transformasi bilinear

dimensi = size(F);
tinggi = dimensi(1);
lebar = dimensi(2);

for y=1 : tinggi
    for x=1 : lebar
        x2 = a1 * x + a2 * y + a3 * x * y + a4;
        y2 = b1 * x + b2 * y + b3 * x * y + b4;

        if (x2>=1) && (x2<=lebar) && ...
           (y2>=1) && (y2<=tinggi)
```

```
            % Lakukan interpolasi bilinear
            p = floor(y2);
            q = floor(x2);
            a = y2-p;
            b = x2-q;

            if (floor(x2)==lebar) || ...
               (floor(y2) == tinggi)
               G(y, x) = F(floor(y2), floor(x2));
            else
               intensitas = (1-a)*((1-b)*F(p,q) +  ...
               b * F(p, q+1)) +      ...
               a *((1-b)* F(p+1, q) + ...
               b * F(p+1, q+1));

               G(y, x) = intensitas;
            end
         else
            G(y, x) = 0;
         end
      end
end

G = uint8(G);
```

Akhir Program

Contoh pemanggilan fungsi `tbilin` seperti berikut:

```
>> F = imread('C:\Image\kotatua.png'); ↵
>> G = tbilin(F, 1.2,0.1,0.005,-45,0.1,1,0.005,-30); ↵
>> imshow(G) ↵
```

Hasinya dapat dilihat pada Gambar 5.27.

Gambar 5.27 Transformasi *bilinear*

Latihan

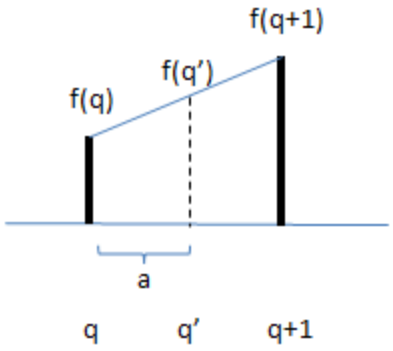
1. Jelaskan pengertian operasi geometrik.
2. Apa yang disebut dengan istilah berikut.
   (a) Pemetaan ke belakang
   (b) Pemetaan ke depan
3. Jelaskan yang dimaksud dengan interpolasi bilinear.
4. Tuliskan persamaan matematika yang menggambarkan operasi pencerminan secara vertikal.
5. Tuliskan sebuah algoritma yang sekaligus digunakan untuk mencerminkan secara horizontal dan vertikal. Implementasikan dalam bentuk program dan ujilah.
6. Cobalah untuk menguji skrip untuk pencerminan dengan menggunakan gambar berwarna. Apa yang terjadi dengan hasilnya? Mengapa begitu?
7. Cobalah untuk memodifikasi cerminh dan cerminv agar bisa digunakan untuk melakukan pencerminan terhadap gambar berwarna.
8. Jelaskan apa yang dimaksud dengan transformasi-transformasi berikut.
   (a) Affine
   (b) Spherical

(c) Twirl

(d) Ripple

(e) Bilinear

9. Gambar berikut memberikan gambaran mengenai proses interpolasi linear (berdimensi satu), yang menjadi dasar dalam penentuan interpolasi bilinear (berdimensi dua).



Berdasarkan gambar di atas, buktikan bahwa

$f(q') = (1-a)\, f(q) + a\, f(q+1)$