26 - Pemampatan Citra

(Bagian 2)

IF4073 Pemrosesan Citra Digital

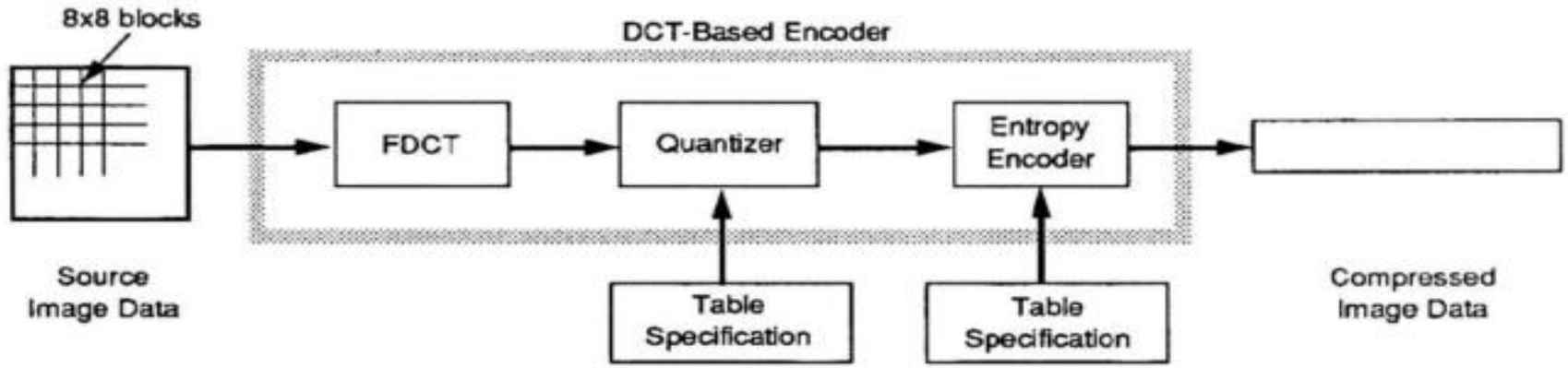
Oleh: Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

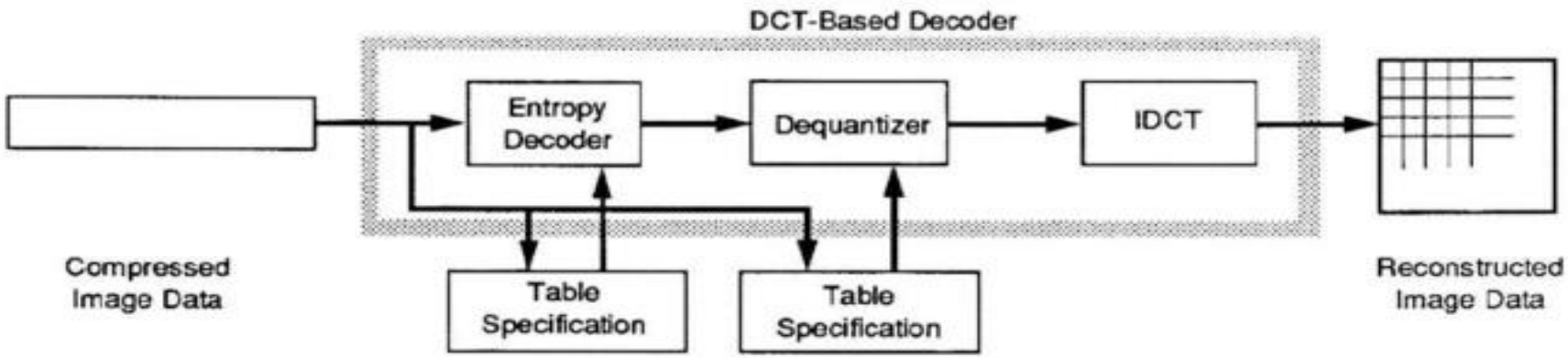
Metode Pemampatan JPEG

- JPEG = *Joint Photographic Experts Groups*.
- Merupakan standard kompresi citra sejak tahun 1992.
- Termasuk *lossy compression*, yang berarti beberapa kualitas visual ada yang hilang selama proses kompresi. Hasil dekompresi tidak kembali sama dengan citra semula.
- Metode JPEG dapat diterapkan pada citra *grayscale* dan citra berwarna RGB.
- Jika citra berwarna, maka RGB dikonversi terlebih dahulu ke ruang warna *YCbCr*, yang dalam hal ini *Y* adalah komponen *luminance*, dan *Cb* dan *Cr* adalah komponen *chrominance* dari citra.
- Terhadap komponen *chrominance* dilakukan *subsampling* untuk mengurangi ukuran file. *Subsampling* dikerjakan dengan mengambil 4 *pixel* bertetangga dan menghitung rata-ratanya menjadi satu nilai.

Diagram umum JPEG Compression



Encoder:



Decoder:

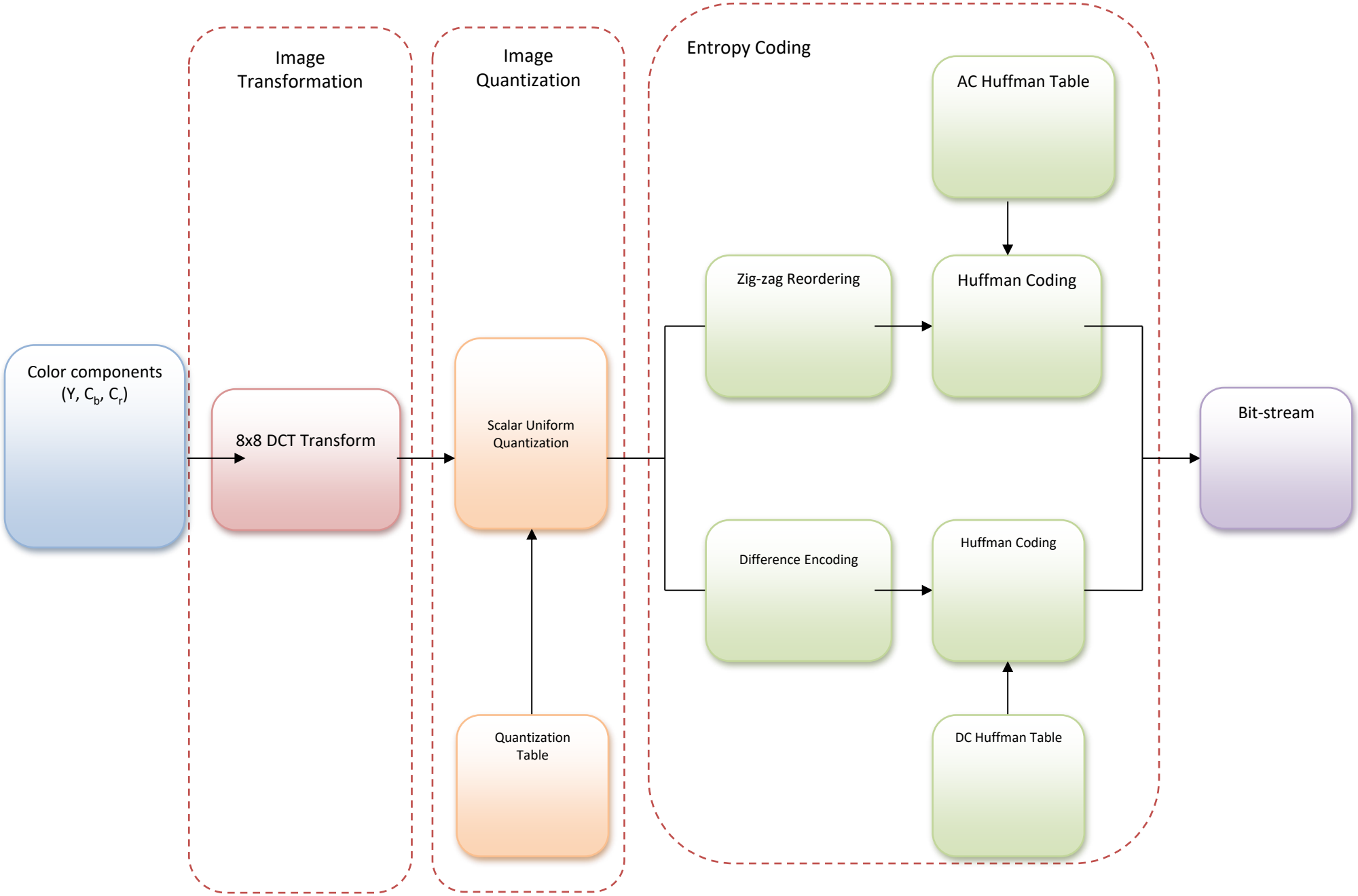
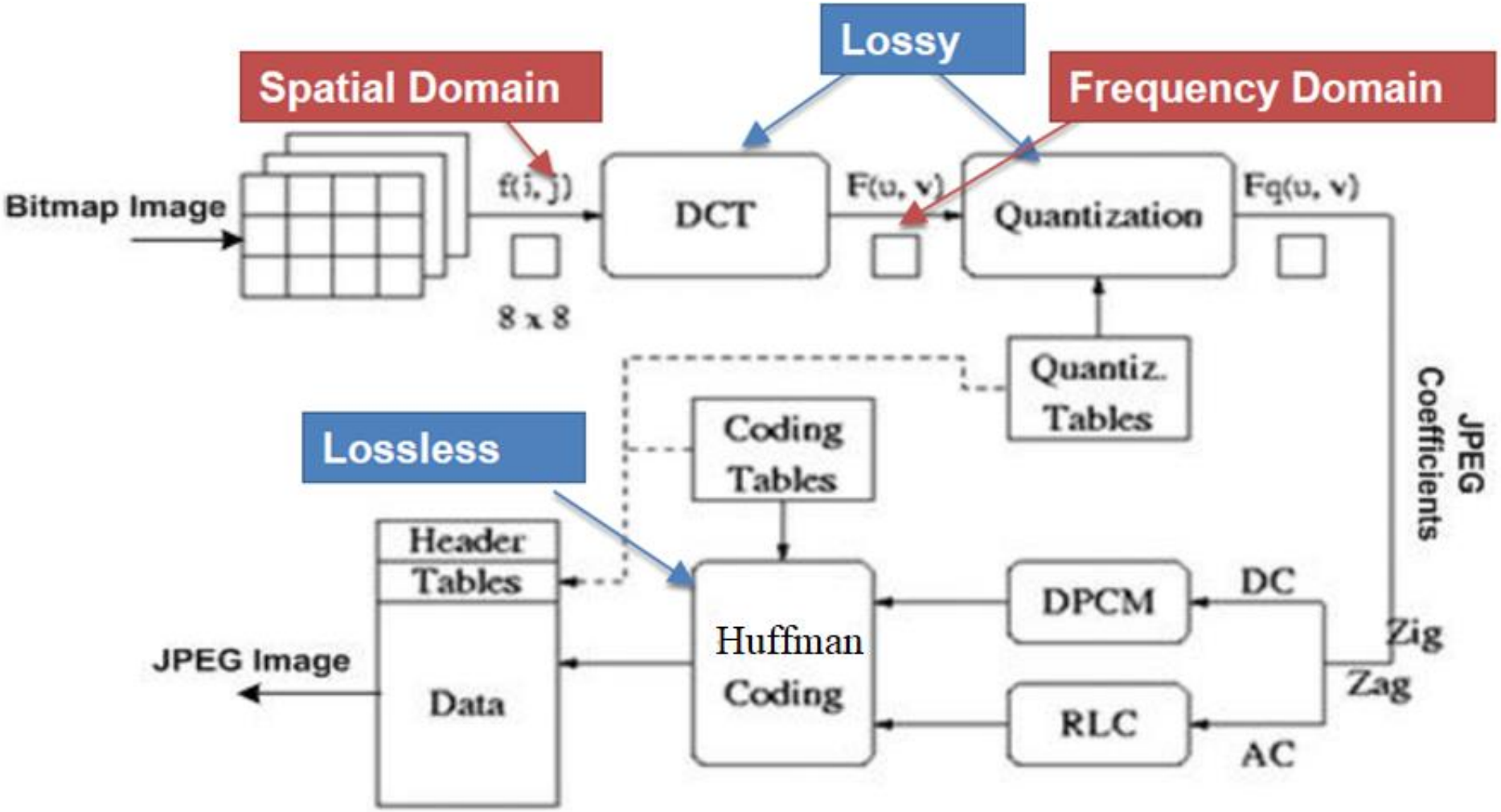


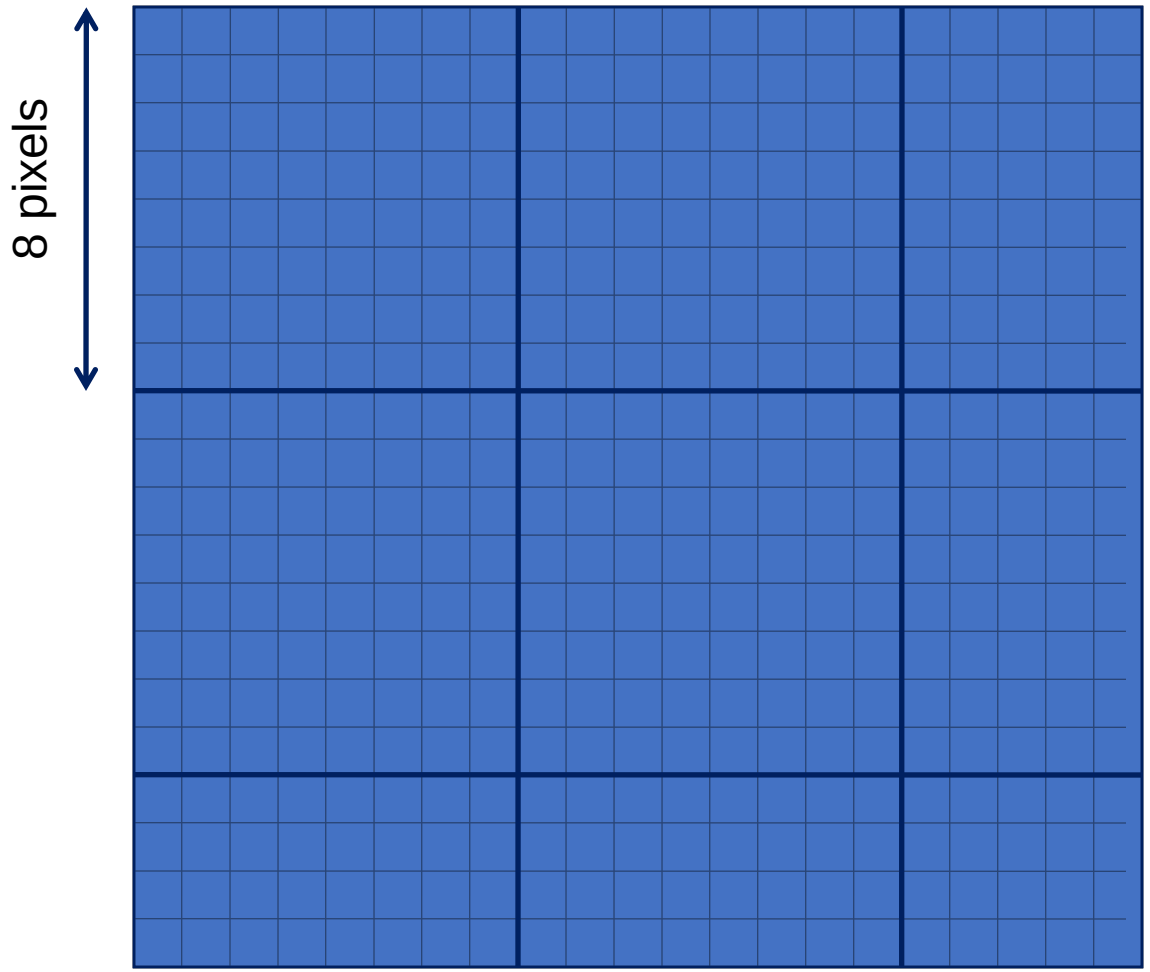
Image Transformation
Image Quantization
Entropy Coding
AC Huffman Table
Color components (Y, C_b, C_r)
8x8 DCT Transform
Scalar Uniform Quantization
Zig-zag Reordering
Huffman Coding
Bit-stream
Difference Encoding
Huffman Coding
Quantization Table
DC Huffman Table

- Diagram rinci JPEG encoder):



Sumber: Mahendra Kumar, *Steganography and Steganalysis of JPEG Images*.

Citra dibagi menjadi blok-blok berukuran 8 x 8 *pixel*:



- Contoh blok berukuran 8 x 8:

200 202 189 188 189 175 175 175
200 203 198 188 189 182 178 175
203 200 200 195 200 187 185 175
200 200 200 200 197 187 187 187
200 205 200 200 195 188 187 175
200 200 200 200 200 190 187 175
205 200 199 200 191 187 187 175
210 200 200 200 188 185 187 186

$f(i, j)$

Setiap blok spasial (baik komponen Y, Cb maupun Cr) yang berukuran 8 x 8 ditranformasi ke dalam ranah frekuensi dengan *Discrete Cosine Transform* (DCT):

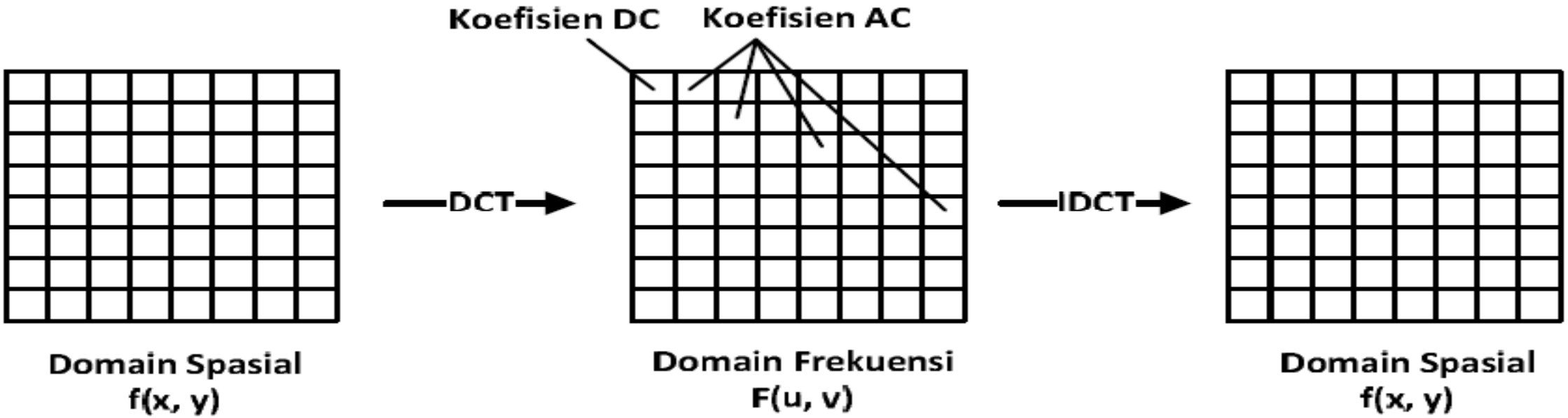
$$F(u,v) = \frac{1}{4} C(u)C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{\pi(2x+1)u}{16} \cos \frac{\pi(2y+1)v}{16}$$

$$C(u), C(v) = \frac{1}{\sqrt{2}}, \text{untuk } u, v = 0$$

$$C(u), C(v) = 1, \text{untuk } u, v \text{ lainnya}$$

Sedangkan transformasi balikannya adalah Inverse *Discrete Cosine Transform* (IDCT):

$$f(x,y) = C(u)C(v) \sum_{u=0}^{7} \sum_{v=0}^{7} F(u,v) \cos \frac{\pi(2x+1)u}{16} \cos \frac{\pi(2y+1)v}{16}$$



Elemen pojok kiri atas (1,1) disebut koefisien DC, sedangkan 63 elemen lainnya disebut koefisien AC.

DC coefficient

AC coefficients

200	202	189	188	189	175	175	175
200	203	198	188	189	182	178	175
203	200	200	195	200	187	185	175
200	200	200	200	197	187	187	187
200	205	200	200	195	188	187	175
200	200	200	200	200	190	187	175
205	200	199	200	191	187	187	175
210	200	200	200	188	185	187	186

$f(i, j)$

Original image (8 x 8)

DCT

515	**65**	**-12**	**4**	**1**	**2**	**-8**	**5**
-16	**3**	**2**	**0**	**0**	**-11**	**-2**	**3**
-12	**6**	**11**	**-1**	**3**	**0**	**1**	**-2**
-8	**3**	**-4**	**2**	**-2**	**-3**	**-5**	**-2**
0	**-2**	**7**	**-5**	**4**	**0**	**-1**	**-4**
0	**-3**	**-1**	**0**	**4**	**1**	**-1**	**0**
-3	**-2**	**-3**	**3**	**3**	**-1**	**-1**	**3**
-2	**5**	**-2**	**4**	**-2**	**2**	**-3**	**0**

$F(u, v)$

Transform domain (8 x 8)

- Namun, sebelum melakukan transformasi DCT pada blok 8×8, elemen-elemen di dalam matriks 8 x 8 digeser nilainya dari kisaran positif ke nilai yang berpusat pada nol.

- Untuk citra 8-bit, setiap entri dalam blok berada dalam rentang [0 , 255]. Titik tengah rentang (dalam hal ini nilai 128) dikurangi dari setiap entri untuk menghasilkan rentang data yang berpusat pada nol, sehingga rentang yang dimodifikasi adalah [− 128 , 127]

- Langkah ini mengurangi persyaratan rentang dinamis pada tahap pemrosesan DCT berikutnya.

200	202	189	188	189	175	175	175
200	203	198	188	189	182	178	175
203	200	200	195	200	187	185	175
200	200	200	200	197	187	187	187
200	205	200	200	195	188	187	175
200	200	200	200	200	190	187	175
205	200	199	200	191	187	187	175
210	200	200	200	188	185	187	186

$- 128 =$

72	74	61	60	61	47	47	47
72	75	70	60	61	54	50	47
75	72	72	67	72	59	57	47
72	72	72	72	69	59	59	59
72	77	72	72	67	60	59	47
72	72	72	72	72	62	59	47
77	72	71	72	63	59	59	47
82	72	72	72	60	57	59	58

- Kode program Matlab

```
f = [200 202 189 188 189 175 175 175;
200 203 198 188 189 182 178 175;
203 200 200 195 200 187 185 175;
200 200 200 200 197 187 187 187;
200 205 200 200 195 188 187 175;
200 200 200 200 200 190 187 175;
205 200 199 200 191 187 187 175;
210 200 200 200 188 185 187 186];
f2 = f - 128;
F = dct2(f2)
```

Hasil run program:

F =

514.8750	65.0169	-11.8199	3.5716	1.3750	2.2531	-7.9574	4.8890
-15.8910	3.4146	1.9509	0.3708	-0.2179	-11.2135	-2.0505	3.3167
-12.3419	6.3602	11.4738	-0.7718	3.0799	0.4245	1.2777	-1.8869
-7.7629	2.6106	-4.4276	2.3042	-2.3899	-2.5523	-4.6449	-1.5003
0.3750	-2.0596	6.5512	-5.3506	4.3750	-0.4741	-1.4959	-3.5484
0.1219	-3.0050	-0.7559	0.3190	4.1586	1.1958	-1.1196	0.2623
2.5415	-1.5842	-3.2223	3.0396	3.1891	-0.6597	-0.7238	2.9414
-1.5441	4.6880	-1.6743	3.5618	-2.3737	1.8760	-2.5608	0.0854

- Hasil transformasi DCT adalah blok-blok berukuran 8 x 8 dengan nilai *floating point*.

$$F(u,v) = \begin{bmatrix} 515 & 65 & -12 & 4 & 1 & 2 & -8 & 5 \\ -16 & 3 & 2 & 0 & 0 & -11 & -2 & 3 \\ -12 & 6 & 11 & -1 & 3 & 0 & 1 & -2 \\ -8 & 3 & -4 & 2 & -2 & -3 & -5 & -2 \\ 0 & -2 & 7 & -5 & 4 & 0 & -1 & -4 \\ 0 & -3 & -1 & 0 & 4 & 1 & -1 & 0 \\ 3 & -2 & -3 & 3 & 3 & -1 & -1 & 3 \\ -2 & 5 & -2 & 4 & -2 & 2 & -3 & 0 \end{bmatrix}$$

- Selanjutnya, setiap nilai di dalam blok dikuantisasi menjadi *integer* dengan cara membaginya dengan elemen matriks kuantisasi Q berukuran 8 x 8 dan membulatkannya ke integer terdekat.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

$$\hat{F}(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right)$$

- Contoh hasil proses kuantisasi: $\hat{F}(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right)$

Contoh:
round(515/16) = round(32.1875) = 32

515	65	-12	4	1	2	-8	5
-16	3	2	0	0	-11	-2	3
-12	6	11	-1	3	0	1	-2
-8	3	-4	2	-2	-3	-5	-2
0	-2	7	-5	4	0	-1	-4
0	-3	-1	0	4	1	-1	0
3	-2	-3	3	3	-1	-1	3
-2	5	-2	4	-2	2	-3	0

$F(u,v)$

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

$$\begin{bmatrix} 32 & 6 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\hat{F}(u, v)$

- Kuantisasi ke *integer* membuat komponen frekuensi tinggi dibulatkan menjadi nol, sedangkan komponen frekuensi sisanya menjadi bilangan positif kecil dan bilangan negatif kecil.

- Proses kuantisasi inilah yang dinamakan *lossy*, karena ada informasi yang hilang akibat pembulatan.

- Pada proses penirmampatan (di dalam *decoder*), hampiran nilai F(u,v) diperoleh kembali dengan mengalikan $\hat{F}(u,v)$ dengan Q(u,v).

$$\tilde{F}(u,v) = \hat{F}(u,v) \times Q(u,v)$$

- Hasil decoding tidak tepat sama dengan nilai pixel semula, ada selisih yang disebut *error*, inilah yang disebut *lossy compression*.

Contoh:

- An 8 × 8 block from the Y image of ‘Lena’

200	202	189	188	189	175	175	175
200	203	198	188	189	182	178	175
203	200	200	195	200	187	185	175
200	200	200	200	197	187	187	187
200	205	200	200	195	188	187	175
200	200	200	200	200	190	187	175
205	200	199	200	191	187	187	175
210	200	200	200	188	185	187	186

Original Image Block $f(i, j)$

515	65	-12	4	1	2	-8	5
-16	3	2	0	0	-11	-2	3
-12	6	11	-1	3	0	1	-2
-8	3	-4	2	-2	-3	-5	-2
0	-2	7	-5	4	0	-1	-4
0	-3	-1	0	4	1	-1	0
3	-2	-3	3	3	-1	-1	3
-2	5	-2	4	-2	2	-3	0

$F(u, v)$ DCT Coefficients

- Fig. 9.2: JPEG compression for a smooth image block.

Quantized DCT Coefficients

$$\begin{bmatrix} 32 & 6 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\hat{F}(u, v)$

De-Quantized DCT Coefficients

$$\begin{bmatrix} 512 & 66 & -10 & 0 & 0 & 0 & 0 & 0 \\ -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & 0 & 16 & 0 & 0 & 0 & 0 & 0 \\ -14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\tilde{F}(u, v)$

Reconstructed Image Block

$$\begin{bmatrix} 199 & 196 & 191 & 186 & 182 & 178 & 177 & 176 \\ 201 & 199 & 196 & 192 & 188 & 183 & 180 & 178 \\ 203 & 203 & 202 & 200 & 195 & 189 & 183 & 180 \\ 202 & 203 & 204 & 203 & 198 & 191 & 183 & 179 \\ 200 & 201 & 202 & 201 & 196 & 189 & 182 & 177 \\ 200 & 200 & 199 & 197 & 192 & 186 & 181 & 177 \\ 204 & 202 & 199 & 195 & 190 & 186 & 183 & 181 \\ 207 & 204 & 200 & 194 & 190 & 187 & 185 & 184 \end{bmatrix}$$

$\tilde{f}(i, j)$

Error

$$\begin{bmatrix} 1 & 6 & -2 & 2 & 7 & -3 & -2 & -1 \\ -1 & 4 & 2 & -4 & 1 & -1 & -2 & -3 \\ 0 & -3 & -2 & -5 & 5 & -2 & 2 & -5 \\ -2 & -3 & -4 & -3 & -1 & -4 & 4 & 8 \\ 0 & 4 & -2 & -1 & -1 & -1 & 5 & -2 \\ 0 & 0 & 1 & 3 & 8 & 4 & 6 & -2 \\ 1 & -2 & 0 & 5 & 1 & 1 & 4 & -6 \\ 3 & -4 & 0 & 6 & -2 & -2 & 2 & 2 \end{bmatrix}$$

$(i, j) = f(i, j) - \tilde{f}(i, j)$

Fig. 9.2 (cont'd): JPEG compression for a smooth image block.

- Another 8 × 8 block from the Y image of ‘Lena’

70	70	100	70	87	87	150	187
85	100	96	79	87	154	87	113
100	85	116	79	70	87	86	196
136	69	87	200	79	71	117	96
161	70	87	200	103	71	96	113
161	123	147	133	113	113	85	161
146	147	175	100	103	103	163	187
156	146	189	70	113	161	163	197

f(i, j)

-80	-40	89	-73	44	32	53	-3
-135	-59	-26	6	14	-3	-13	-28
47	-76	66	-3	-108	-78	33	59
-2	10	-18	0	33	11	-21	1
-1	-9	-22	8	32	65	-36	-1
5	-20	28	-46	3	24	-30	24
6	-20	37	-28	12	-35	33	17
-5	-23	33	-30	17	-5	-4	20

F(u, v)

Fig. 9.3: JPEG compression for a textured image block.

$$\begin{bmatrix} -5 & -4 & 9 & -5 & 2 & 1 & 1 & 0 \\ -11 & -5 & -2 & 0 & 1 & 0 & 0 & -1 \\ 3 & -6 & 4 & 0 & -3 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\hat{F}(u, v)$

$$\begin{bmatrix} -80 & -44 & 90 & -80 & 48 & 40 & 51 & 0 \\ -132 & -60 & -28 & 0 & 26 & 0 & 0 & -55 \\ 42 & -78 & 64 & 0 & -120 & -57 & 0 & 56 \\ 0 & 17 & -22 & 0 & 51 & 0 & 0 & 0 \\ 0 & 0 & -37 & 0 & 0 & 109 & 0 & 0 \\ 0 & -35 & 55 & -64 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\tilde{F}(u, v)$

$$\begin{bmatrix} 70 & 60 & 106 & 94 & 62 & 103 & 146 & 176 \\ 85 & 101 & 85 & 75 & 102 & 127 & 93 & 144 \\ 98 & 99 & 92 & 102 & 74 & 98 & 89 & 167 \\ 132 & 53 & 111 & 180 & 55 & 70 & 106 & 145 \\ 173 & 57 & 114 & 207 & 111 & 89 & 84 & 90 \\ 164 & 123 & 131 & 135 & 133 & 92 & 85 & 162 \\ 141 & 159 & 169 & 73 & 106 & 101 & 149 & 224 \\ 150 & 141 & 195 & 79 & 107 & 147 & 210 & 153 \end{bmatrix}$$

$\tilde{f}(i, j)$

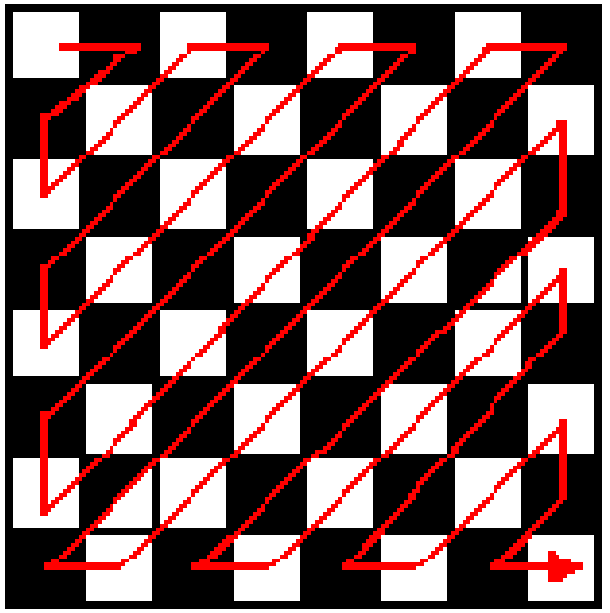
$$\begin{bmatrix} 0 & 10 & -6 & -24 & 25 & -16 & 4 & 11 \\ 0 & -1 & 11 & 4 & -15 & 27 & -6 & -31 \\ 2 & -14 & 24 & -23 & -4 & -11 & -3 & 29 \\ 4 & 16 & -24 & 20 & 24 & 1 & 11 & -49 \\ -12 & 13 & -27 & -7 & -8 & -18 & 12 & 23 \\ -3 & 0 & 16 & -2 & -20 & 21 & 0 & -1 \\ 5 & -12 & 6 & 27 & -3 & -2 & 14 & -37 \\ 6 & 5 & -6 & -9 & 6 & 14 & -47 & 44 \end{bmatrix}$$

$(i, j) = f(i, j) - \tilde{f}(i, j)$

Fig. 9.3 (cont'd): JPEG compression for a textured image block.

- Selanjutnya, koefisien DCT hasil kuantisasi dibaca secara zig-zag untuk membantu tahap *entropy coding*.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63



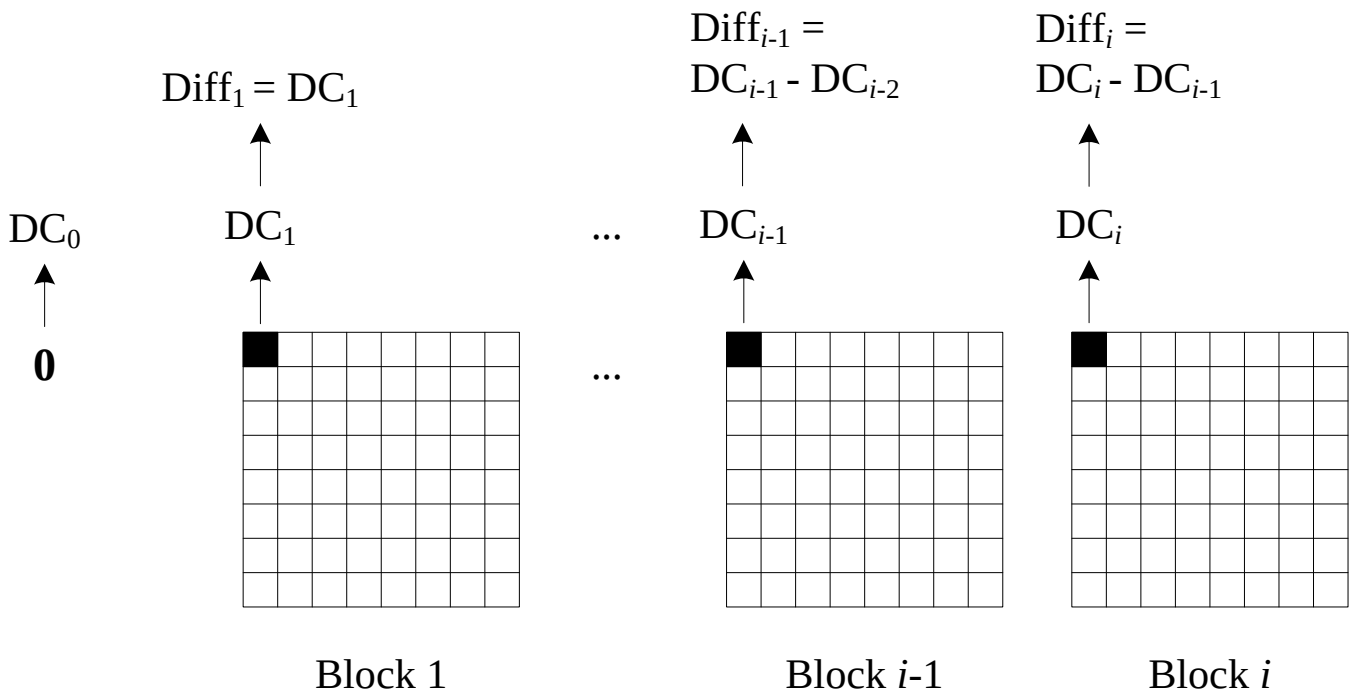
- Tahap *entropy coding* adalah *arithmetic coding* (RLE dan DPCM) dan *Huffman coding*, keduanya adalah metode *lossless compression*. RLE dan Huffman coding sudah dijelaskan pada materi PPT Bagian 1

- *Huffman coding* mengkompresi hasil *arithmetic coding*. Pohon Huffman kemudian disimpan sebagai di dalam file JPEG.

DPCM (*Differential Pulse Code Modulation*) pada Koefisien DC

- Koefisien DC pada setiap blok dikodekan dengan DPCM sebagai berikut:

$$Diff_i = DC_{i+1} - DC_i \qquad \text{dan} \quad Diff_0 = DC_0.$$

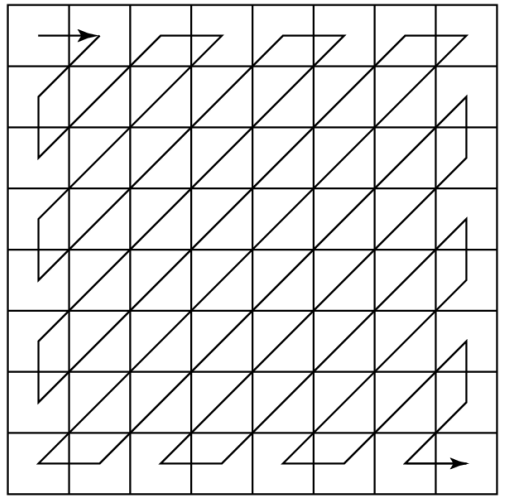


- Jika koefisien DC untuk 5 blok pertama adalah 150, 155, 149, 152, 144, maka DPCM akan menghasilkan 150, 5, -6, 3, -8.
- Selanjutnya, semua *Diff* tersebut dikodekan dengan Huffman Coding bersama-sama dengan koefisien AC

Run-length Encoding (RLE) pada Koefisien AC

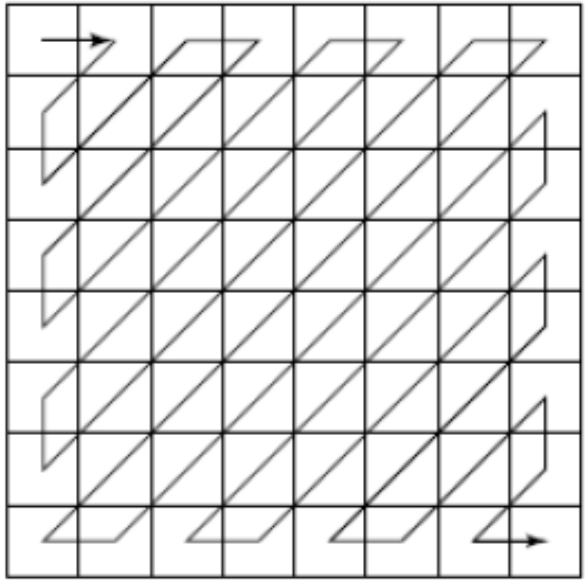
- RLE bertujuan untuk menghasilkan himpunan *{(koefisien AC, runlength)*}. Metode RLE (atau RLC) sudah dijelaskan pada PPT Bagian 1

- Untuk membuatnya paling mungkin mencapai *run* nol yang panjang, maka dlakukan pemindaian zig-zag untuk mengubah matriks 8 × 8 menjadi vektor dengan 64 elemen.



$$
\begin{matrix}
32 & 6 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}
$$

$\hat{F}(u, v)$



$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

F(u,v) =

Hasil pembacaan secara zig-zag:

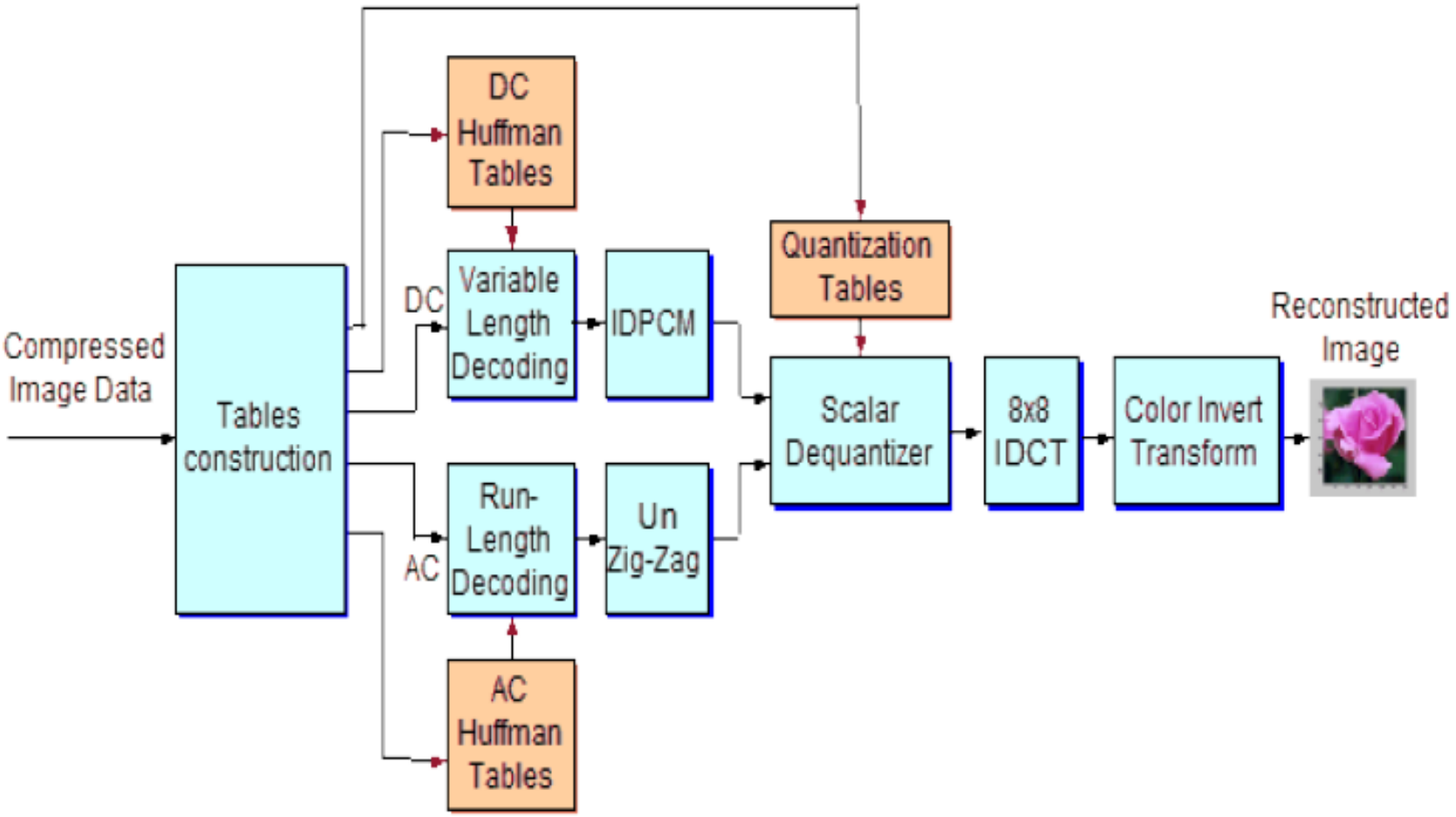
−26,
−3, 0,
−3 , −2, −6,
2, −4 , 1, −3,
1, 1, 5, 1, 2,
−1 , 1 , −1 , 2 , 0 , 0,
0 , 0 , 0 , −1 , −1 , 0 , 0,
0 , 0 , 0 , 0 , 0 , 0 , 0, 0,
0 , 0 , 0 , 0 , 0 , 0 , 0,
0 , 0 , 0 , 0 , 0 , 0,
0, 0 , 0 , 0 , 0,
0 , 0 , 0 , 0,
0 , 0 , 0,
0 , 0,
0

- Elemen pertama vektor = DC, 63 elemen lainnya adalah elemen AC. Elemen DC diproses dengan DCPM, sedangkan 63 elemen AC dikodekan dengan RLE.

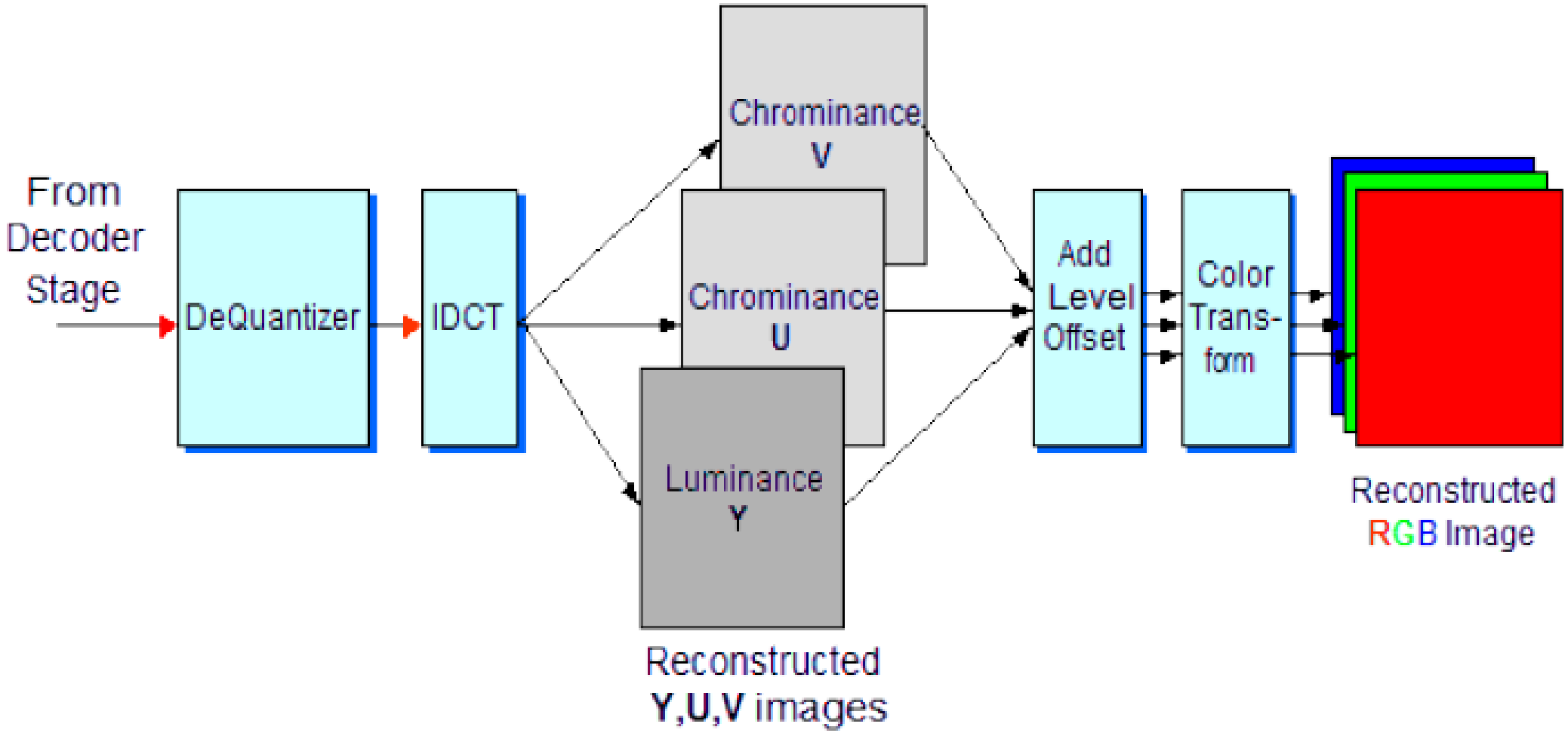
- Selanjutnya, hasil pengkodean RLE dan DCPM dimampatkan dengan Huffman coding.

- Tabel Huffman dan string bit hasil pengkodean Huffman disimpan di dalam file JPEG.

Penirmampatan (Dekompresi) JPEG



Sumber: Multimedia - Image Compression, Prepared by Dr. Omar A. Dawood



Sumber: Multimedia - Image Compression, Prepared by Dr. Omar A. Dawood

Original Image

Compressed Image

- JPEG image compression menghasilkan artefak-artefak di dalam citra hasil pemampatan, namun masih dapat ditolerir secara visual

- *Less computational complexity*

Sumber*: Image Processing*, *Image Compression*, DR. FERDA ERNAWAN
Faculty of Computer Systems & Software Engineering, Pahang University

Metrik Pengukuran Pemampatan Citra

1. Nisbah (*ratio*) pemampatan

   Bermacam-macam rumus menghitung nisbah pemampatan

$$Nisbah1 = \frac{ukuran\ citra\ sesudah\ dimampatkan}{ukuran\ citra\ sebelum\ dimampatkan}\ \text{x}\ 100\%$$

artinya ukuran citra sekarang menjadi Nisbah1 (dalam persen) kali ukuran citra semula.

$$Nisbah2 = 100\% - \frac{ukuran\ citra\ sesudah\ dimampatkan}{ukuran\ citra\ sebelum\ dimampatkan}\ \text{x}\ 100\%$$

artinya citra sebanyak Nisbah2 (dalam persen) telah dimampatkan.

- Contoh: Citra semula berukuran 256×256 pixels, 8-bit per pixel, grayscale.

  Ukuran citra adalah 65536 byte (64 kb).

  Setelah dimampatkan, ukuran citra menjadi 40280 byte

  Nisbah = (40280/65536) x 100% = 61,5 %

  (artinya ukuran citra menjadi 61,5% dari ukuran semula)

  Nisbah = 100% - 61,5% = 38,5%

  (artinya 38,5% citra sudah dimampatkan)

2. Ukuran *fidelity*

- Kualitas sebuah citra bersifat subyektif dan relatif, bergantung pada pengamatan orang yang menilainya
- Kualitas hasil pemampatan dapat diukur secara kuantitatif dengan menggunakan besaran *PSNR* (*peak signal-to-noise ratio*).
- *PSNR* dihitung untuk mengukur perbedaan antara citra semula dengan citra hasil pemampatan

$$PSNR = 20 \times \log_{10}\left(\frac{b}{rms}\right)$$

b adalah nilai sinyal terbesar (pada citra dengan 256 derajat keabuan, $b = 255$)
rms (*root mean square*) adalah akar pangkat dua dari selisih antara citra semula dengan citra hasil pemampatan

$$rms = \sqrt{\frac{1}{\text{Lebar} \times \text{Tinggi}} \sum_{i=1}^{N}\sum_{j=1}^{M}(f_{ij} - f'_{ij})^2}$$

f dan f’ masing-masing menyatakan nilai pixel citra semula dan nilai pixel citra hasil pemampatan

- *PSNR* memiliki satuan *decibel* (dB).

- *PSNR* berbanding terbalik dengan *rms*. Nilai *rms* yang rendah yang menyiratkan bahwa citra hasil pemampatan tidak jauh berbeda dengan citra semula, sehingga menghasilkan *PSNR* yang tinggi, yang berarti kualitas pemampatannya bagus.

- Nilai rms yang tinggi menyatakan galat yang besar akibat pemampatan, sehinga menghasilkan PSNR yang rendah.

- Semakin besar nilai *PSNR*, semakin bagus kualitas pemampatannya.

- Dalam prakteknya, nilai PSNR $\geq$ 30 menyatakan kualitas citra yang sudah dianggap baik. Di bawah 30 itu dikatakan citra mengalami degradasi yang semakin besar dengan menurunnya PSNR.

peppers.bmp, 256 x 256
(193 KB)

peppers.jpg, 256 x 256
(52.2 KB), JPEG Quality = 5

```
>> ref = imread('peppers512.bmp');
>> A = imread('peppers512-med.jpg');
>> psnr = psnr(A, ref)

psnr =

   35.08
```

peppers2.jpg, 256 x 256
(24 KB), JPEG Quality = 1

```
>> ref = imread('peppers512.bmp');
>> A = imread('peppers512-low.jpg');
>> psnr = psnr(A, ref)

psnr =

   30.5051
```

Fractal Image Compression

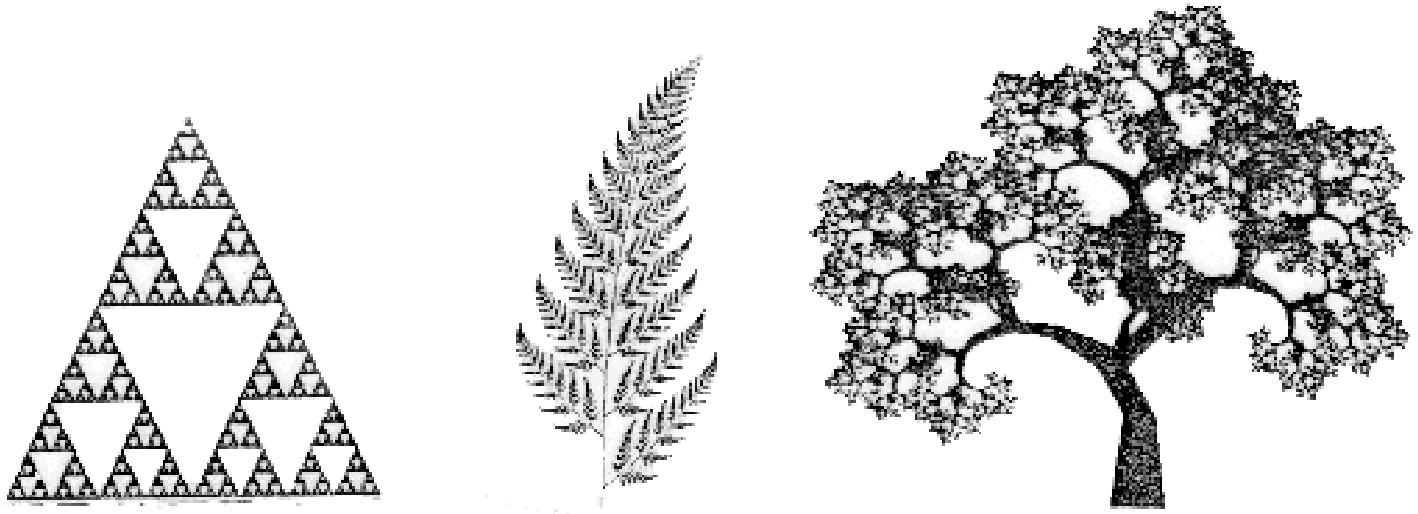
- Algoritma pemampatan citra dengan cara kerja yang unik.
- Prinsip: mencari bagian di dalam citra yang memiliki kemiripan dengan bagian lainya namun ukurannya lebih besar (*self similarity*).
- Cari matriks yang mentransformasikan bagian yang lebih besar tersebut dengan bagian yang lebih kecil.
- Simpan hanya elemen-elemen dari sekumpulan matriks transformasi tersebut (yang disebut matriks transformasi *affine*).
- Pada proses penirmampatan, matriks ransformasi *affine* di-iterasi sejumlah kali terhadap sembarang citra awal.

Fraktal

Definisi

Fraktal: objek yang memiliki kemiripan dirinya-sendiri (*self-similarity*) namun dalam skala yang berbeda.

Fraktal: objek yang memiliki matra berupa pecahan (*fractional*). Kata terakhir inilah yang menurunkan kata **fraktal**

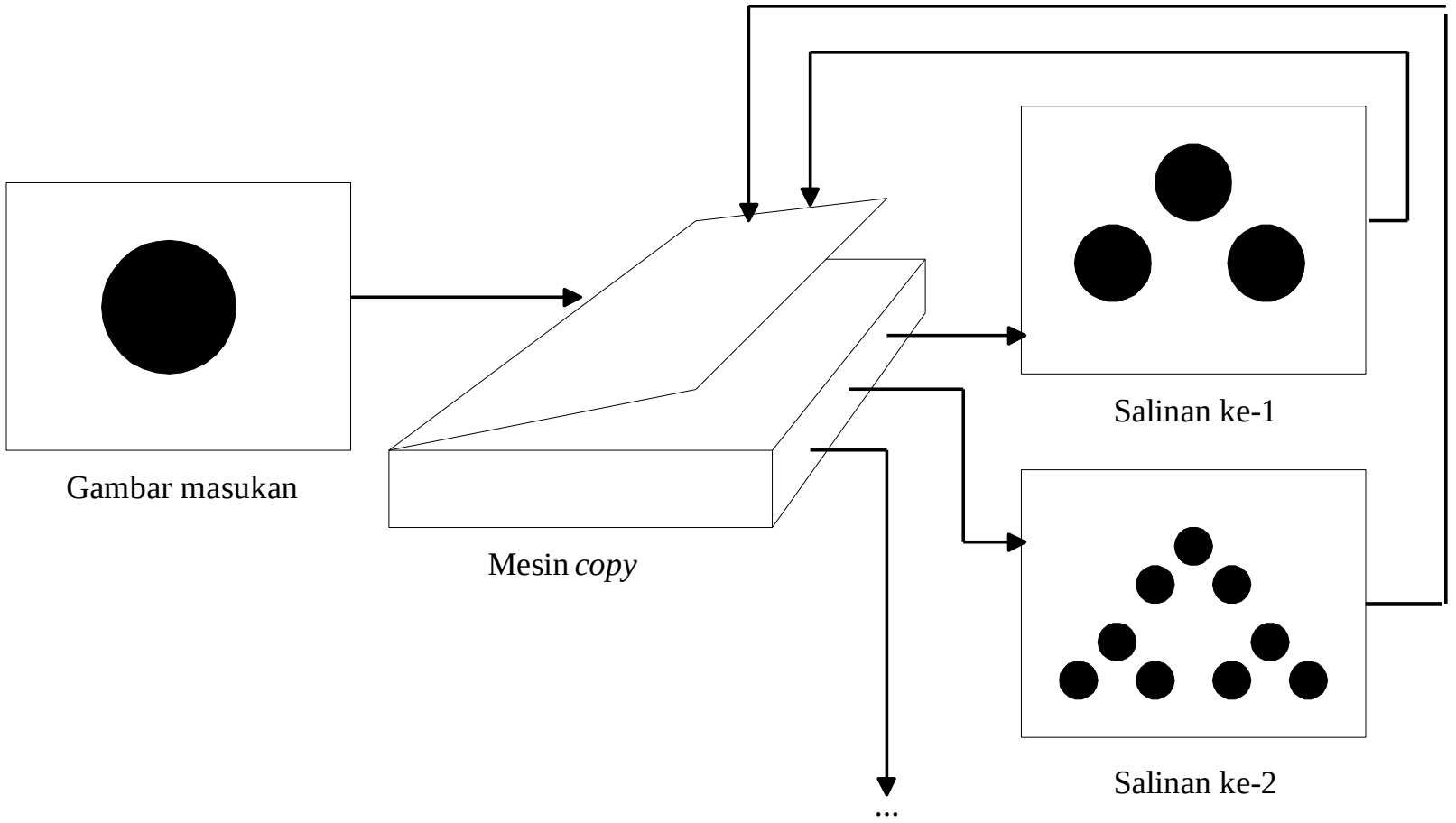


Segitiga Sierpinski, daun pakis Barsnsley, dan pohon fractal

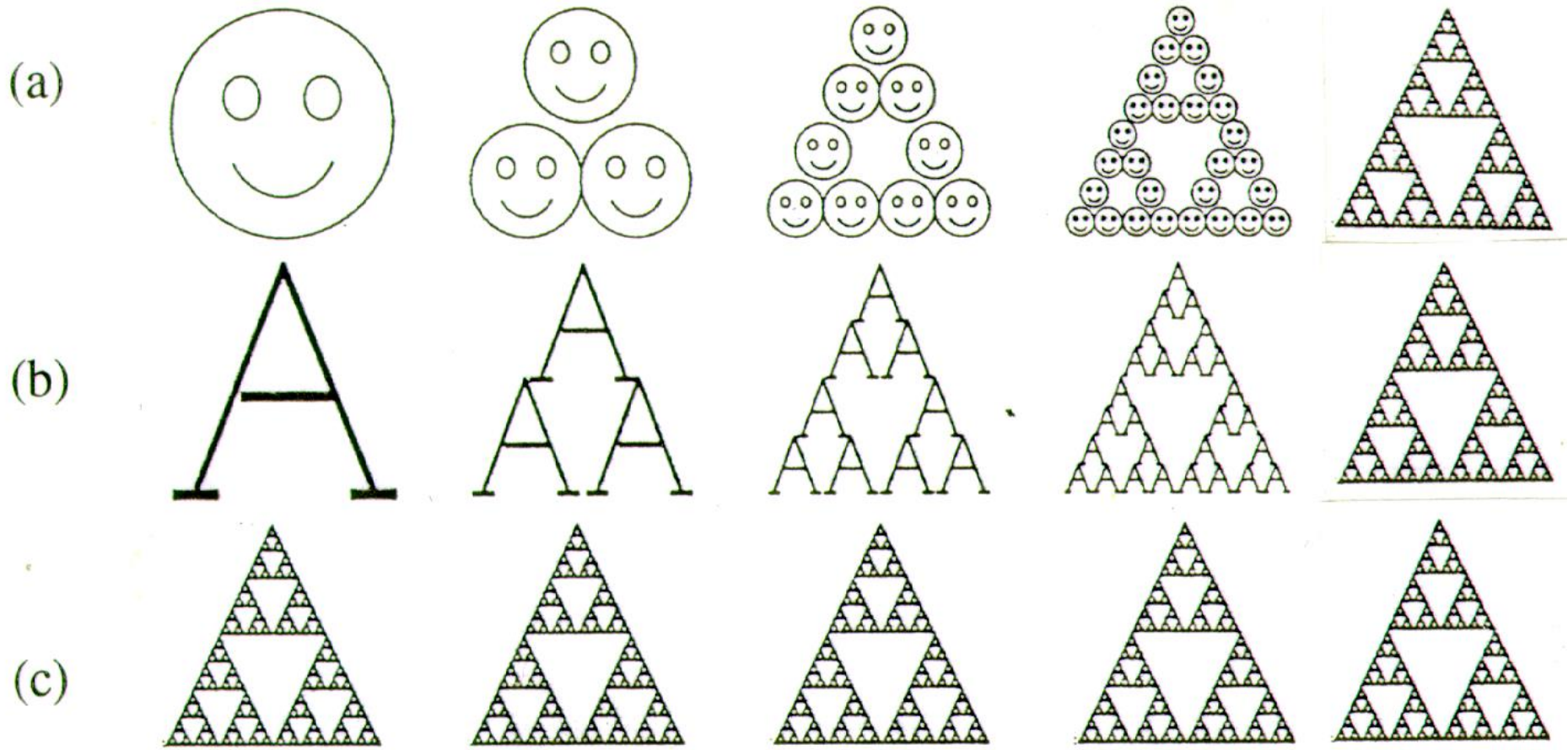
Fraktal di alam

Iterated Function System (IFS)

Penemu: Michael Barnsley (1988)



Multiple Reduction Copy Machine (MRCM)



Apapun gambar awalnya, MRCM selalu menghasilkan segitiga Sierpienski .

- MRCM dapat dinyatakan dalam bentuk transformasi *affine*:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = w\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = Ax + t$$

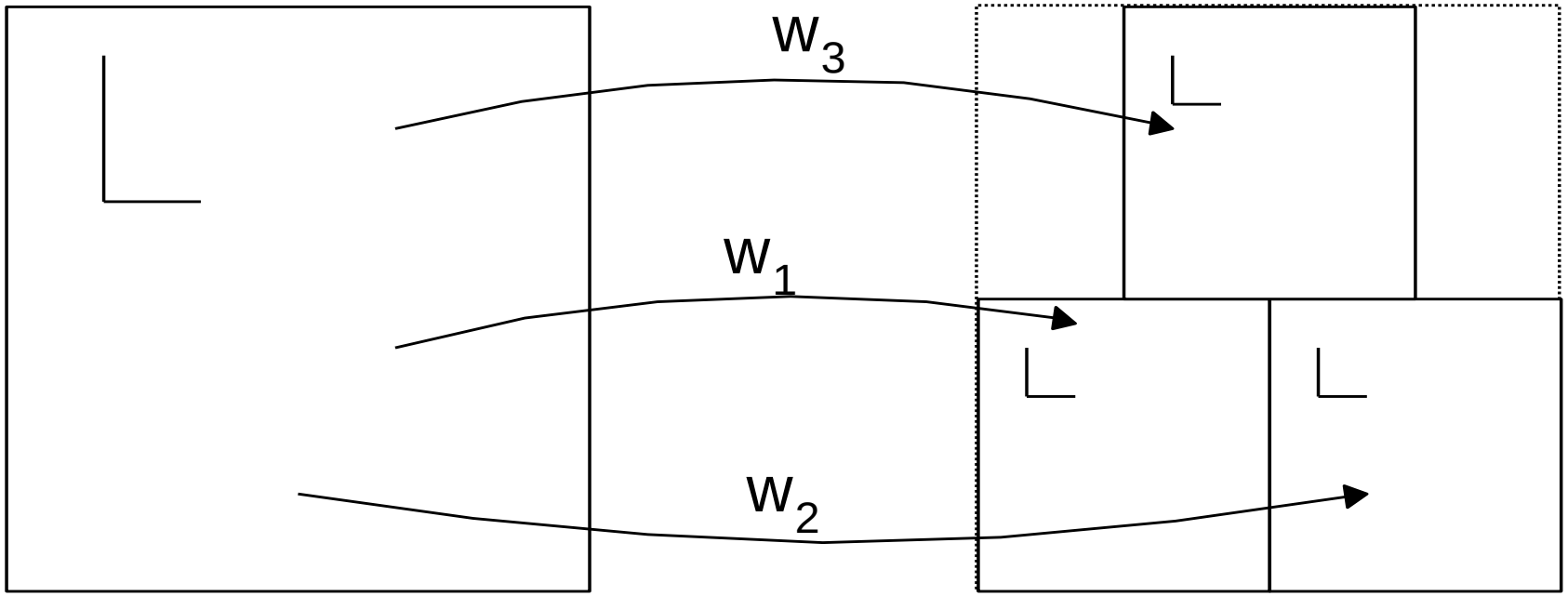
- Untuk sembarang citra awal A, dihasilkan salinan *affine*, $w_1(A)$, $w_2(A)$, …, $w_n(A)$.

- Gabungan dari seluruh salinan tersebut adalah $W(A)$, yang merupakan keluaran dari mesin,

$$W(A) = w_1(A) + w_2(A) + \ldots + w_n(A)$$

- Tansformasi *affine* yang menghasilkan citra segitiga Sierpinski:

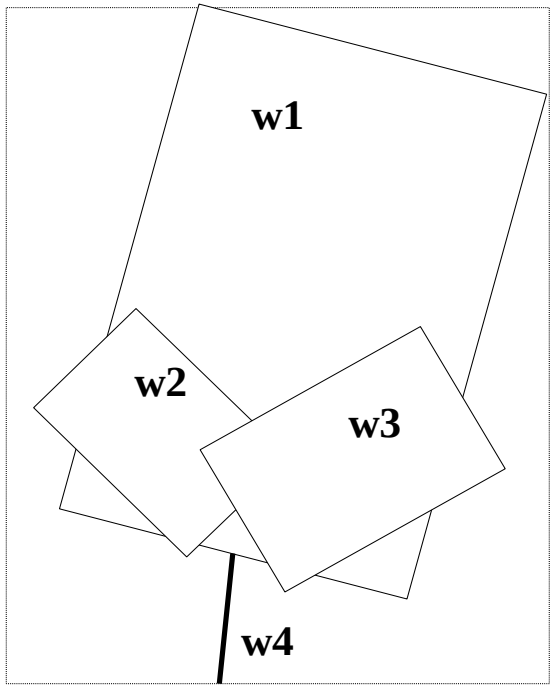
$$w_1 = \begin{bmatrix} 0.5 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} \quad w_2 = \begin{bmatrix} 0.5 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} \quad w_3 = \begin{bmatrix} 0.5 & 0.0 & 0.25 \\ 0.0 & 0.5 & 0.5 \end{bmatrix}$$



- Tansformasi *affine* yang menghasilkan citra daun pakis:

$$w_1 = \begin{bmatrix} 0.85 & 0.04 & 0.0 \\ -0.04 & 0.85 & 1.6 \end{bmatrix} \qquad w_2 = \begin{bmatrix} 0.20 & -0.26 & 0.0 \\ 0.23 & 0.22 & 1.6 \end{bmatrix}$$

$$w_3 = \begin{bmatrix} -0.15 & 0.28 & 0.0 \\ 0.26 & 0.52 & 0.44 \end{bmatrix} \qquad w_4 = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.16 & 0.0 \end{bmatrix}$$



- Menyimpan citra sebagai kumpulan pixel membutuhkan memori yang besar, namun bila yang disimpan adalah transformasi *affine*-nya, maka memori yang dibutuhkan jauh lebih sedikit.

- Cara ini melahirkan gagasan pengkodean citra dengan nisbah pemampatan yang tinggi.

- Pakis Barnsley misalnya, dibangkitkan dengan empat buah transformasi affine, masing-masingnya terdiri atas enam buah bilangan riil (4 byte), sehingga dibutuhkan 4 x 6 x 4 byte = 96 byte untuk menyimpan keempat transformasi itu.

- Bandingkan bila citra pakis Barnsley disimpanxdengan representasi pixel hitam putih (1 pixel = 1 byte) berukuran 550 x 480 membutuhkan memori sebesar 264.000 byte. Maka, nisbah pemampatan citra pakis adalah 264.000 : 96 = 2750 : 1, suatu nisbah yang sangat tinggi.

Partitioned Iterated Function System (PIFS)

Penemu: Arnaud D. Jacquin (1992), mahasiswa bimbingan Michael Barnsley

Dasar pemikiran:

- Citra alami (*natural image*) umumnya hampir tidak pernah *self-similar* secara keseluruhan.

- Karena itu, citra alami pada umumnya tidak mempunyai transformasi *affine* terhadap dirinya sendiri.

- Tetapi, untunglah citra alami seringkali memiliki *self-similarity* lokal, yaitu memiliki bagian citra yang mirip dengan bagian lainnya.

- Setiap transformasi itu dari bagian citra ke bagian citra lain yang mirip dapat direpresentasikan dengan transformasi *IFS* lokal atau *Partitioned Iterated Function System* (PIFS)

Kemiripan lokal pada citra Lena

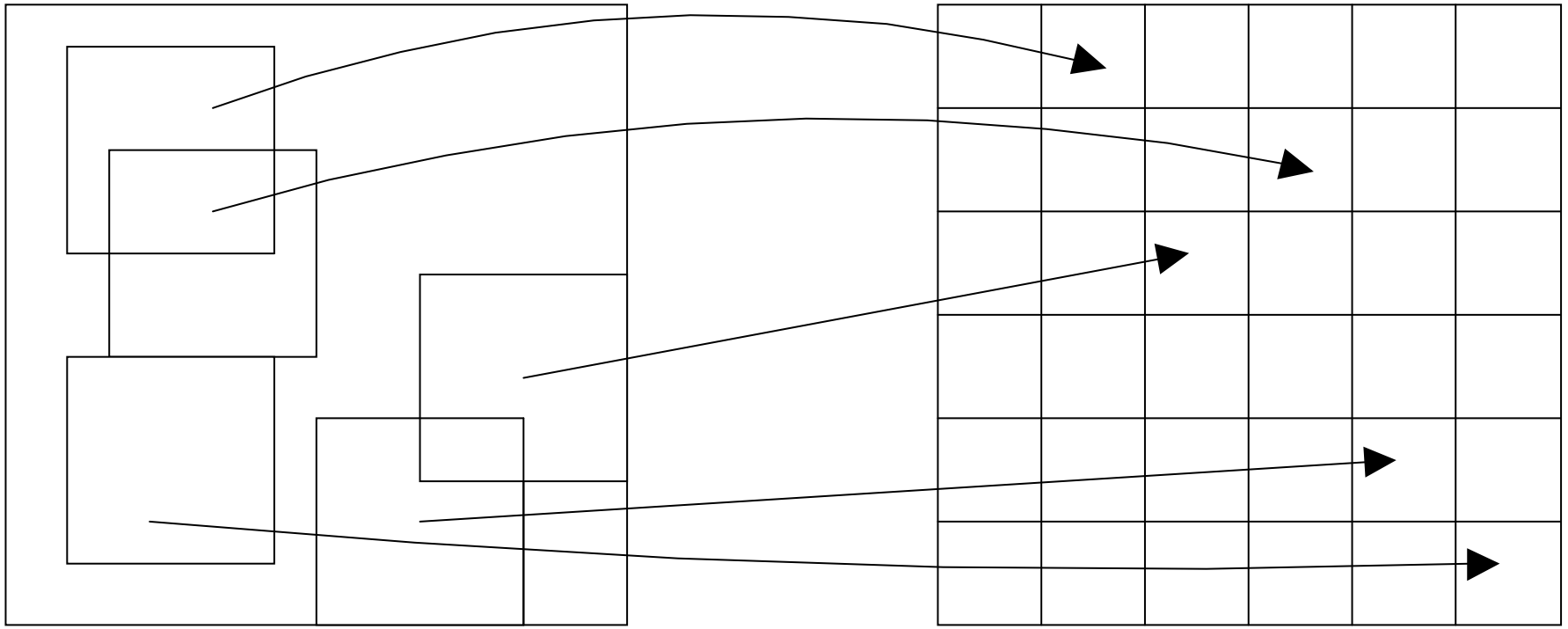
Algoritma:

1. Bagi citra atas sejumlah blok yang berukuran sama dan tidak saling beririsan, yang disebut blok jelajah (*range*).

2. Untuk setiap blok jelajah, cari bagian citra yang berukuran lebih besar dari blok jelajah –yang disebut blok ranah (*domain*)- dan paling mirip (cocok) dengan blok jelajah tersebut.

3. Turunkan transformasi *affine* (*IFS* lokal) w_i yang memetakan blok ranah ke blok jelajah.

4. Hasil dari semua pemasangan ini adalah *Partitioned Iterated Function System* (*PIFS*).



Blok ranah

Blok jelajah

Pemetaan dari blok ranah ke blok jelajah

- Kemiripan antara dua buah (blok) citra diukur dengan metrik jarak. Metrik jarak yang digunakan misalnya *rms (root mean square*):

$$d_{rms} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(z'_{ij} - z_{ij})^2}$$

z dan *z*' adalah nilai *pixel* dari dua buah blok, dan *n* = jumlah *pixel* di dalam citra

- Ukuran blok ranah diambil dua kali blok jelajah.
- Contoh: Untuk blok jelajah 8×8 *pixel* dan blok ranah berukuran 16×16 *pixel*, citra 256×256 dibagi menjadi 1024 buah blok jelajah yang tidak saling beririsan dan $(256 - 16 + 1)^2 = 58.081$ buah blok ranah berbeda (yang beririsan).
- Himpunan blok ranah yang digunakan dalam proses pencarian kemiripan dimasukkan ke dalam *pul ranah* (*domain pool*).
- Pul ranah yang besar menghasilkan kualitas pemampatan yang lebih baik, tetapi membutuhkan waktu pencocokan yang lebih lama.

- Transformasi affine di dalam PIFS memiliki komponen z selain komponen koordinat (x,y):

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

  - s_i menyatakan faktor kontras *pixel* (nilanya antara 0 dan 1)
  - o_i menyatakan ofset kecerahan (*brightness*) *pixel*
  - $z' = s_i z + o_i$

- Dengan asumsi ukuran blok ranah = dua kali ukuran blok jelaja (2:1), maka transformasi *affine* menjadi lebih sederhana, yaitu:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

- Parameter e_i dan f_i mudah dihitung karena keduanya menyatakan pergeseran sudut kiri blok ranah ke sudut kiri blok jelajah yang bersesuaian.

- Sedangkan s_i dan o_i dihitung dengan menggunakan rumus regresi berikut:

$$s = \frac{\left[n\sum_{i=1}^{n} d_i r_i - \sum_{i=1}^{n} d_i \sum_{i=1}^{n} r_i \right]}{\left[n\sum_{i=1}^{n} d_i^2 - (\sum_{i=1}^{n} d_i)^2 \right]} \qquad o = \frac{1}{n}\left[\sum_{i=1}^{n} r_i - s\sum_{i=1}^{n} d_i \right]$$

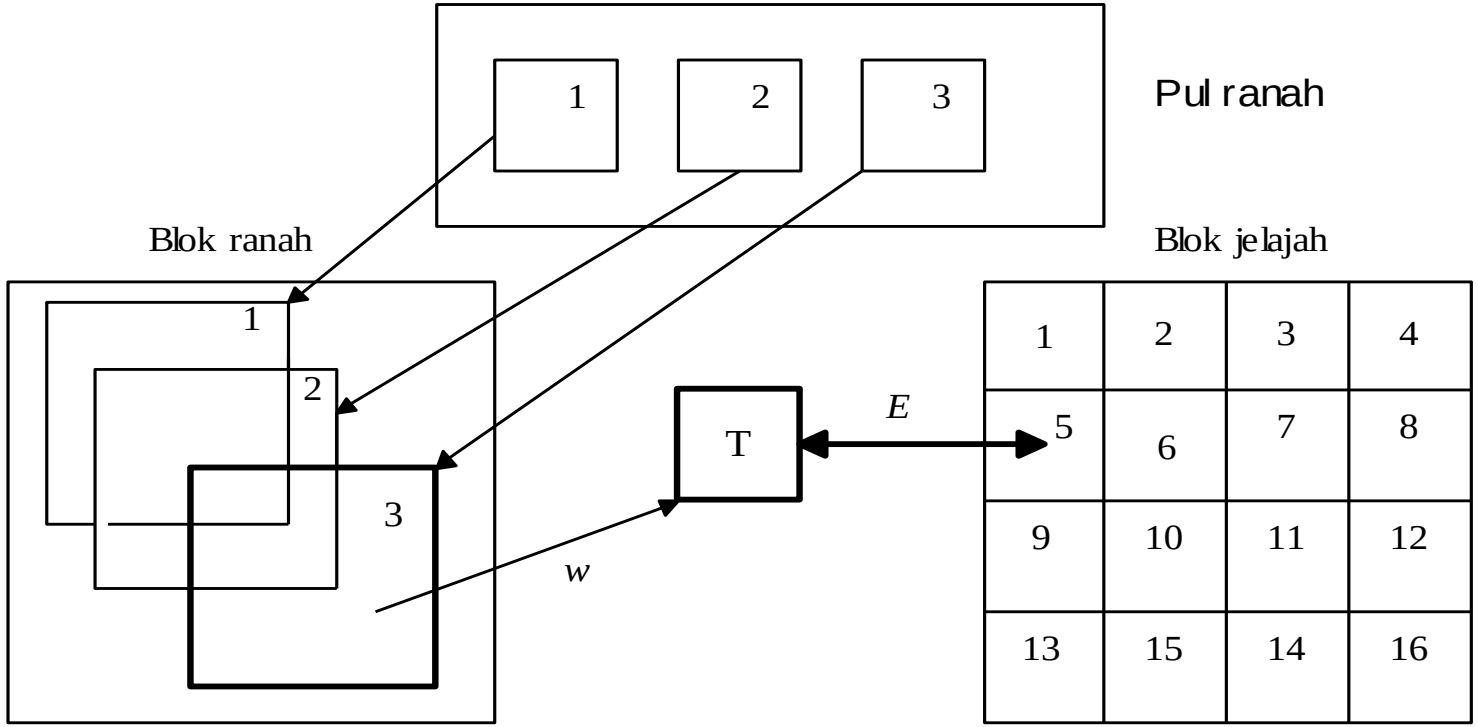
- Dengan nilai *s* dan *o* yang telah diperoleh, maka kuadrat galat antara blok jelajah dan blok ranah adalah

$$E = \frac{1}{n}\left[\sum_{i=1}^{n} r_i^2 + s\left(s\sum_{i=1}^{n} d_i^2 - 2\sum_{i=1}^{n} d_i r_i + 2o\sum_{i=1}^{n} d_i \right) + o\left(no - 2\sum_{i=1}^{n} r_i \right) \right]$$

- Lalu hitung *rms* sebagai berikut:

$$d_{rms} = \sqrt{E}\ /n$$

- Transformasi *affine* w_i diuji terhadap blok ranah D_i menghasilkan blok uji $T_i = w_i(D_i)$.
- Jarak antara T dan R_i dihitung dengan persamaan d_{rms}.
- Transformasi *affine* yang terbaik ialah w yang meminimumkan d_{rms}.



Blok jelajah 5 dibandingkan dengan blok ranah 3 di dalam pul ranah. Transformasi w ditentukan, lalu blok ranah 3 ditransformasikan dengan w menghasilkan T. Jarak antara T dengan blok jelajah 5 diukur.

- Runtunan pencarian dilanjutkan untuk blok jelajah berikutnya sampai seluruh blok jelajah sudah dipasangkan dengan blok ranah.

- Hasil dari proses pemampatan adalah sejumlah *IFS* lokal yang disebut *PIFS*.

- Seluruh parameter *PIFS* di-pak dan disimpan di dalam berkas eksternal. Parameter *PIFS* yang perlu disimpan hanya e_i, f_i, s_i, o_i.

- Algoritma pencocokan blok yang dijelaskan di atas adalah algoritma *brute force*, karena untuk setiap blok jelajah pencocokan dilakukan dengan seluruh blok ranah di dalam pul untuk memperoleh pencocokan terbaik.

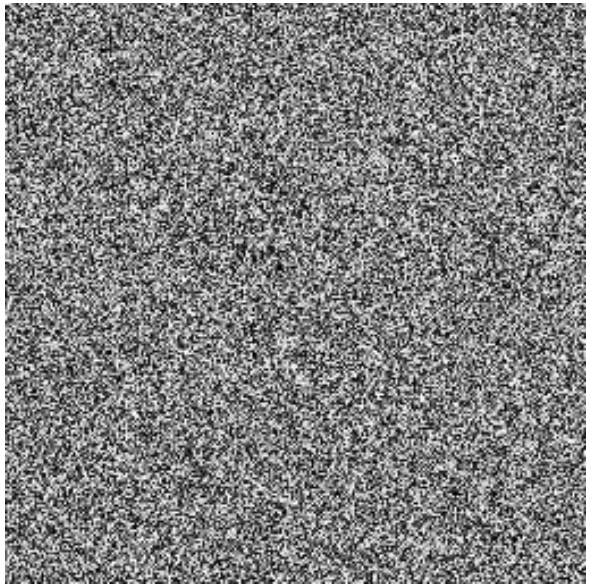
Rekonstruksi Citra (penirmpatan)

- Rekonstruksi (dekompresi) citra dilakukan dengan melelarkan *PIFS* dari citra awal sembarang.

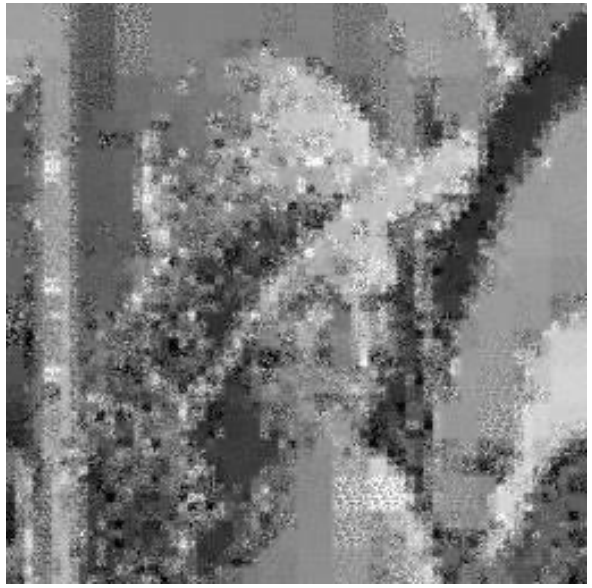
- Karena setiap *PIFS* lokal kontraktif, baik kontraktif dalam matra intensitas maupun kontraktif dalam matra spasial maka lelarannya akan konvergen ke citra titik-tetap *PIFS*.

- Kontraktif intensitas penting untuk menjamin konvergensi ke citra semula, sedangkan kontaktif spasial berguna untuk membuat rincian pada citra untuk setiap skala.

- Konvergensi ke citra titik-tetap berlangsung cepat. Konvergensi umumnya dapat diperoleh dalam 8 sampai 10 kali lelaran



Citra awal



Citra lelaran ke-1

Citra lelaran ke-2

Citra lelaran ke-6

- Beberapa hasil pemampatan fraktal:

Citra asli (256 × 256 *pixel*)
Lena.bmp (66 KB)

Citra hasil pemampatan fraktal
Lena.fra (7 KB)

Citra asli (256 × 256 *pixel*)
Collie.bmp (66 KB)

Citra hasil pemampatan fraktal
Collie.fra (9 KB)

Citra asli (512 × 512 *pixel*)
Kapal.bmp (258 KB)

Citra hasil pemampatan fraktal
Kapal.fra (9 KB)

Citra asli (316 × 404 *pixel*)
Potret.bmp (126 KB)

Citra hasil pemampatan fraktal
Potret.fra (17 KB)

Tabel 1. Perbandingan ukuran berkas citra sebelum dan sesudah dimampatkan

No.	Citra BMP (*byte*)	Ukuran (*byte*)	Citra FRA (*byte*)	Ukuran	Nisbah (%)
1	Kapal.bmp	263.222	KAPAL512.FRA	8.956	96,6
2	Lena.bmp	66.614	LENA256.FRA	8.137	87,6
3	Collie.bmp	66.614	COLLI256.FRA	9.150	86,3
4	Potret.bmp	128.782	POTRET.FRA	17.437	86,5

Tabel 2. Perbandingan ukuran citra berformat BMP, JPG, GIF, dan fraktal(FRA)

Nama Citra	Format BMP (*byte*)	Format JPG (*byte*)	Format GIF (*byte*)	Format FRA (*byte*)
Kapal.bmp	263.222	24.367	242.452	8.956
Lena.bmp	66.614	7.126	70.292	8.137
Collie.bmp	66.614	7.021	69.965	9.150
Potret.bmp	128.782	16.377	136.377	17.437