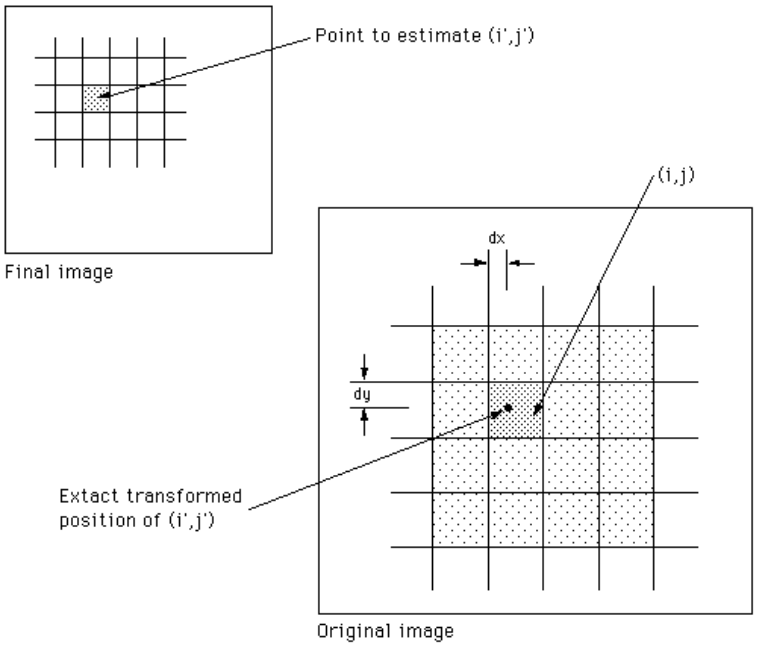
2. TEORI PENUNJANG

2.1. Algoritma *Bicubic Interpolation*

Pendekatan yang digunakan dalam algoritma *bicubic interpolation* untuk *image scaling* adalah memperkirakan warna dari setiap *pixel* pada *final image* dengan mengambil nilai rata-rata dari 16 *pixel* terdekat di sekitar *pixel* yang bersesuaian pada *original image*.



Gambar 2.1. Ilustrasi *scaling*[1]

Contoh gambar di atas menjelaskan prinsip dari *scaling* yang digunakan. Kita ingin menentukan warna dari setiap titik (i',j') pada *final image*.

[1]Sumber: *http://astronomy.swin.edu.au/~pbourke/colour/bicubic/*

Terdapat hubungan linear *scaling* di antara dua *image*, yaitu titik (i',j') menunjuk pada posisi *non integer* pada *original image*. Posisi ini didapat dengan:

$$x = i' w / w' \quad (2.1.)^{[1]}$$

$$y = j' h / h' \quad (2.2.)^{[1]}$$

Keterangan :
w = *width* dari *original image*
h = *height* dari *original image*
w' = *width* dari *final image*
h' = *height* dari *final image*

Koordinat *pixel* terdekat (i,j) adalah nilai *integer* dari x dan y. Nilai dx dan dy dalam gambar diperoleh melalui:

$$dx = x - i \quad (2.3.)^{[1]}$$

$$dy = y - j \quad (2.4.)^{[1]}$$

Persamaan berikut memberikan nilai interpolasi yang diaplikasikan pada setiap komponen *red*, *green* dan *blue* dari suatu *pixel*.

$$F(i',j') = \sum_{m=-1}^{2} \sum_{n=-1}^{2} F(i+m, j+n)\, R(m - dx)\, R(dy - n) \quad (2.5.)^{[1]}$$

Keterangan:
F(i',j') = koordinat *pixel* pada *final image*
F(i ,j) = koordinat *pixel* pada *original image*

dengan fungsi R(x)

$$R(x) = \frac{1}{6}\left[P(x+2)^3 - 4\,P(x+1)^3 + 6\,P(x)^3 - 4\,P(x-1)^3 \right] \quad (2.6.)^{[1]}$$

$$P(x) = \begin{cases} x & x > 0 \\ 0 & x \le 0 \end{cases} \quad (2.7.)^{[1]}$$

[1]Sumber: *http://astronomy.swin.edu.au/~pbourke/colour/bicubic/*

Kita coba mengambil contoh pada sebuah *image* berukuran 200x100 *pixel*. Pada *image* ini kemudian akan dilakukan *scaling bicubic interpolation* menjadi *image* berukuran 300x150 *pixel*.

w = 200

h = 100

w'= 300

h'= 150

Kemudian untuk setiap *pixel* pada *final image*, akan dilakukan fungsi interpolasi sesuai persamaan di atas. Marilah kita mengambil contoh pada suatu *pixel*. Misalnya *pixel* (151,83), berarti i' = 151 dan j' = 83.

Selanjutnya, dengan persamaan (2.1.) dan (2.2.), dapat diperoleh:

x = 100,67

y = 55,33

Kemudian dengan persamaan (2.3.) dan (2.4.) diperoleh nilai dx dan dy.

dx = x – i

= 100,67 – 100

= 0,67

dy = y – j

= 55,33 – 55

= 0,33

Dengan menggunakan persamaan yang ditunjukkan pada persamaan (2.5.), maka akan diperoleh nilai interpolasi dari *pixel* (151,83) pada *final image* dengan melibatkan 16 *pixel* di sekitar *pixel* (100,55) pada *original image*. Hal ini dilakukan untuk masing-masing nilai *red*, *green*, dan *blue* dari *pixel* (151,83).

2.2. Format *File* BMP

Format *file* BMP merupakan format standar sistem operasi *Microsoft Windows* dan *IBM OS/2*. Format ini mendukung resolusi warna dari monokrom hingga *true color* (16,7 juta warna). BMP mudah dibuka dan disimpan, tetapi kurang efisien karena terlalu besar. Meskipun sebenarnya BMP mendukung kompresi *run-length encoding* (RLE), tetapi kebanyakan *file* BMP tidak dikompresi.

Tabel 2.1. *File Header* BMP

Offset	**Ukuran (*byte*)**	**Nama**	**Keterangan**
0	2	*BmpType*	Tipe *file* BMP (BM)
2	4	*BmpSize*	Ukuran *file* dalam *bytes*
6	2	*Reserved1*	Tidak digunakan
8	2	*Reserved2*	Tidak digunakan
10	4	*OffBits*	*Offset byte* awal data citra

Sumber: Tabel 11-2 (Kay & Levine, 1995, 164)

Tabel 2.2. *Image Header* BMP

Offset	**Ukuran (*byte*)**	**Nama**	**Keterangan**
14	4	*HdrSize*	Ukuran *header* dalam *byte* (40)
18	4	*Width*	Lebar *image* dalam *pixel*
22	4	*Height*	Panjang *image* dalam *pixel*
26	2	*Planes*	Jumlah *plane* (hampir selalu 1)
28	2	*BitCount*	Jumlah *bit* per *pixel* (1, 4, 8, atau 24)
30	4	*Compression*	Jenis kompresi (0 = tak terkompresi)
34	4	*ImgSize*	Ukuran citra dalam *byte*
38	4	*HorzRes*	Resolusi horisontal
42	4	*VertRes*	Resolusi vertikal
46	4	*ClrUsed*	Jumlah warna yang digunakan (0 = 256 warna)
50	4	*ClrImportant*	Jumlah warna yang penting
54	4*N	*Colors*	Isi dari *color map*

Sumber: Tabel 11-4 (Kay & Levine, 1995, 167)

Tabel 2.3. *Color Map Entry* BMP

Offset	**Nama**	**Keterangan**
0	*RgbBlue*	Nilai warna biru
1	*RgbGreen*	Nilai warna hijau
2	*RgbRed*	Nilai warna merah
3	*rgbReserved*	Tidak digunakan

Sumber: Tabel 11-3 (Kay & Levine, 1995, 165)

Data citra langsung mengikuti *color map*. Data citra dapat dalam bentuk tidak terkompresi, atau terkompresi dengan metode RLE (*run-length encoding*).

2.3. Format *File* PCX

Format *file* PCX merupakan format asli program *paint* produksi *Zsoft* yaitu *PC Paintbrush*. Format ini paling banyak digunakan untuk citra monokrom

dan 16 warna, meskipun sebenarnya PCX telah dikembangkan untuk menampung 256 warna dan *true color*. Algoritma penyimpanannya sangat sederhana yaitu dengan RLE.

Tabel 2.4. *File Header* PCX

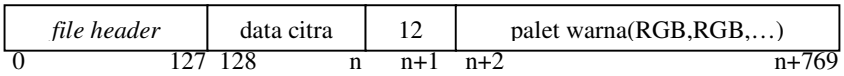
Start Byte	**Ukuran (*byte*)**	**Nama**	**Keterangan**
0	1	*Manufacturer*	Tanda *file* PCX (selalu 10)
1	1	*Version*	Versi *file* PCX: 0: *PC Paintbrush* 2.5 2: *PC Paintbrush* 2.8; dengan palet 3: *PC Paintbrush* 2.8; tanpa palet (gunakan palet standar) 4: *Microsoft Windows* lama; tanpa palet 5: *PC Paintbrush* 3.0; dengan palet
2	1	*Encoding*	Metode pengkodean (selalu 1) – RLE
3	1	*BitsPerPixel*	Jumlah *bit* per *pixel* dalam tiap *plane*
4	2	*XMin*	Lokasi sisi kiri gambar
6	2	*YMin*	Lokasi sisi atas gambar
8	2	*XMax*	Lokasi sisi kanan gambar
10	2	*YMax*	Lokasi sisi bawah gambar
12	2	*HRes*	Resolusi horisontal dalam DPI (*dots*/*inch*)
14	2	*VRes*	Resolusi vertikal dalam DPI (*dots*/*inch*)
16	48	*ColorMap*	Palet 16 warna untuk EGA atau VGA
64	1	*Reserved*	Tidak digunakan
65	1	*Planes*	Jumlah *plane*
66	2	*BytesPerLine*	Jumlah *byte* per baris
68	2	*PaletteType*	Jenis palet: 1: warna atau hitam putih 2: *grayscale* (intensitas kelabu)
70	2	*hScreenSize*	Resolusi horisontal layar
72	2	*vScreenSize*	Resolusi vertikal layar
74	53	*Filler*	Pemborosan 53 *byte*. Abaikan.

Sumber: Tabel 4-3 (Kay & Levine, 1995, 47)

Data citra langsung mengikuti ke 128 *byte header* tadi. PCX menggunakan algoritma kompresi *run-length encoding* (RLE).

Pada awalnya format PCX hanya ditujukan untuk menampung 16 warna. Hal ini dapat dilihat dengan dipasangnya palet pada *header*. 256 warna, *high color* dan *true color* baru dikenal jika *byte version* bernilai 5 dan paletnya tidak disimpan di *header*, melainkan di akhir *file* (setelah data citranya). Informasi palet tersebut didahului oleh sebuah indikator *byte* bernilai 12. Penyimpanan dilakukan secara normal, yaitu merah, hijau dan biru.

Ilustrasi sederhana dari struktur *file* PCX versi 5 :



2.3.1. Algoritma *Run-Length Encoding* (RLE)

Dalam algoritma RLE, setiap paket perulangan data terdiri dari dua bagian yaitu *byte* kunci (yang menentukan adanya perulangan atau tidak beserta jumlahnya) dan *byte* data (yang akan diulang). *Byte* kunci pada citra PCX menentukan apakah ada perulangan atau tidak. Jika dua *bit* tingginya diset (bernilai 1), berarti *byte* berikutnya mempunyai perulangan. Banyaknya perulangan diperoleh dari *byte* kunci tersebut dengan dua *bit* tingginya dihilangkan. Berarti jumlah perulangan yang mungkin ditampung dalam 6 *bit* yaitu 63 kali. Perulangan yang lebih dari 63 kali disimpan sebagai dua paket. Jika kedua *bit* tinggi tidak diset, berarti *byte* yang bersangkutan ditulis apa adanya. Jika *byte* data asli mempunyai dua *bit* tinggi diset, maka terpaksa disimpan dalam bentuk paket perulangan (dengan nilai perulangan=1), berarti yang seharusnya satu *byte* disimpan sebagai dua *byte*. Akan tetapi, hal ini jarang terjadi.

2.4. Format *File* GIF (*Graphics Interchange Format*)

Format *file* GIF merupakan hasil rancangan *Compuserve Information Service*, layanan informasi *online* terbesar di dunia. Format ini dirancang untuk memudahkan pertukaran citra *bitmap* antarkomputer. Karena menggunakan algoritma kompresi LZW (Lempel-Ziv & Welch), *file* GIF terkompresi dengan baik sehingga pertukaran citra dapat dilakukan dengan efektif dan efisien. Tetapi di samping itu algoritma LZW cukup rumit, sehingga penyimpanan dan penampilan *file* GIF relatif lambat.

File GIF dapat berisi beberapa blok data yang berbeda. Ada tiga blok data yang umum dikenal pada *file* GIF yaitu blok *control* untuk informasi identifikasi mengenai konfigurasi *hardware* dan bagaimana mengolah blok-blok lainnya, blok *graphic-rendering* yang berisi data citra sebenarnya, serta blok *special sequence* untuk informasi dan komentar yang mungkin digunakan oleh aplikasi tertentu. Rakitan blok-blok ini disebut data *stream*.

2.4.1. Struktur *File* GIF

Berikut adalah struktur *file* GIF:

<GIF *Data Stream*> ::= *Header* <*Logical Screen*> <*Data*>* *Trailer*
<*Logical Screen*> ::= *Logical Screen Descriptor* [*Global Color Table*]
<*Data*> ::= <*Graphic Block*> | <*Special-Purpose Block*>
<*Graphic Block*> ::= [*Graphic Control Extension*] <*Graphic-Rendering Block*>
<*Graphic-Rendering Block*> ::= <*Table-Based Image*> | *Plain Text Extension*
<*Table-Based Image*> ::= *Image Descriptor* [*Local Color Table*] *Image Data*
<*Special-Purpose Block*> ::= *Application Extension* | *Comment Extension*

```
Keterangan:
<>  entity
::= terdiri dari
*   pemunculan 0 atau lebih
|   elemen alternatif
[ ] elemen optional
```

Contoh penjelasan:

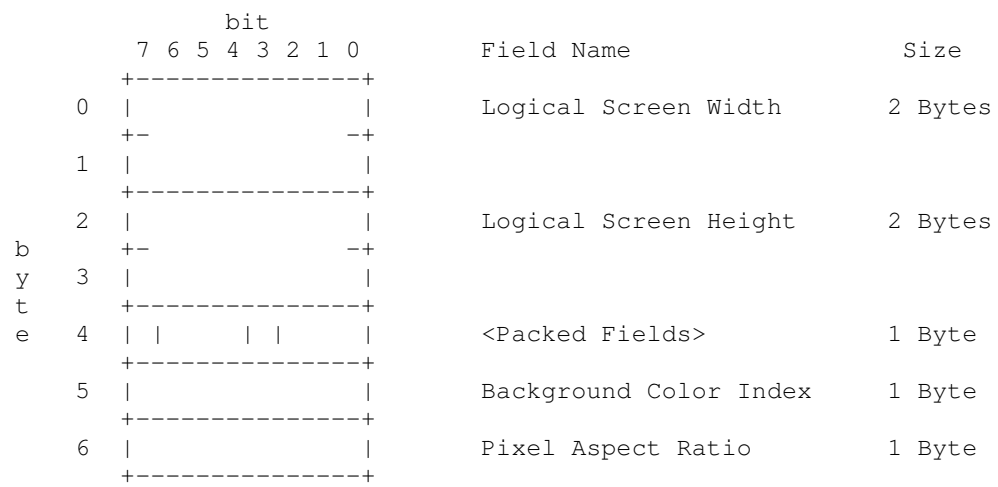
<GIF *Data Stream*> ::= *Header* <*Logical Screen*> <*Data*>* *Trailer*

Rule ini menjelaskan tentang *entity* <GIF *Data Stream*>. Isi dari *entity* <GIF *Data Stream*> dimulai dengan *Header*. Lalu diikuti dengan *entity* berikutnya yaitu *Logical Screen* yang diatur oleh *rule* lainnya. <*Logical Screen*> diikuti oleh *entity* <*Data*> yang juga diatur oleh rule lainnya. <*Data*> memiliki simbol (*) mengikutinya berarti <*Data*> dapat diulangi (*repeat*) beberapa kali, termasuk nol kali. Dan terakhir, *entity* <*Data*> diikuti oleh *Trailer*.

```
              bit
          7 6 5 4 3 2 1 0        Field Name                Size
        +---------------+
    0   |               |        Signature               3 Bytes
        +-             -+
    1   |               |
        +-             -+
 b  2   |               |
 y      +---------------+
 t  3   |               |        Version                 3 Bytes
 e      +-             -+
    4   |               |
        +-             -+
    5   |               |
        +---------------+
Keterangan:
Signature = kode dari citra GIF ('GIF')
Version   = versi dari citra GIF ('87a' atau '89a')
```

Gambar 2.2. Blok *Header* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*



```
                bit
          7 6 5 4 3 2 1 0        Field Name                   Size
        +---------------+
    0   |               |        Logical Screen Width       2 Bytes
        +-             -+
    1   |               |
        +---------------+
    2   |               |        Logical Screen Height      2 Bytes
b       +-             -+
y   3   |               |
t       +---------------+
e   4   | |     | |     |        <Packed Fields>            1 Byte
        +---------------+
    5   |               |        Background Color Index     1 Byte
        +---------------+
    6   |               |        Pixel Aspect Ratio         1 Byte
        +---------------+

      <Packed Fields>  = Global Color Table Flag       1 Bit
                         Color Resolution              3 Bits
                         Sort Flag                     1 Bit
                         Size of Global Color Table    3 Bits
```

Keterangan:

Logical Screen Width = ukuran lebar layar

Logical Screen Height = ukuran tinggi layar

Global Color Table Flag = *flag* keberadaan dari *Global Color Table* (0=tidak ada, 1=ada)

Color Resolution = resolusi warna-1 (jumlah *bit* dari warna yang digunakan)

Sort Flag = *flag* pengurutan *Global Color Table* (0=tidak diurutkan, 1=diurutkan menurut warna yang paling sering digunakan)

Size of Global Color Table = ukuran dari *Global Color Table* Jumlah warna = 2^(*Size of Global Color Table*+1)

Background Color Index = indeks dari warna latar belakang

Pixel Aspect Ratio = aspek rasio *pixel*

Gambar 2.3. Blok *Logical Screen Descriptor* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*

byte	bit 7 6 5 4 3 2 1 0	Field Name	Size
0		Image Separator	1 Byte
1		Image Left Position	2 Bytes
2			
3		Image Top Position	2 Bytes
4			
5		Image Width	2 Bytes
6			
7		Image Height	2 Bytes
8			
9	| | | | | | | |	<Packed Fields>	1 Byte

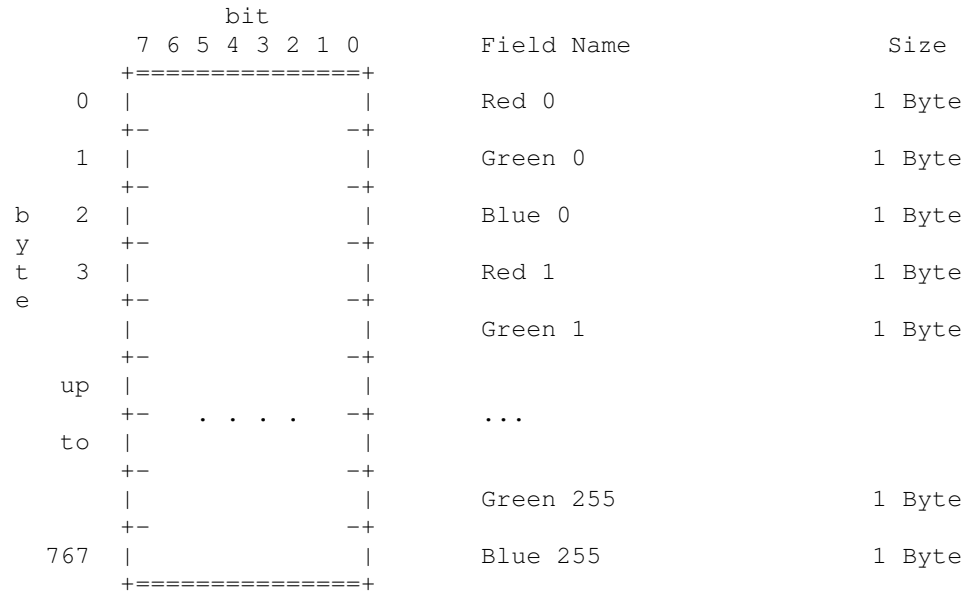
<Packed Fields> =		
Local Color Table Flag	1 Bit	
Interlace Flag	1 Bit	
Sort Flag	1 Bit	
Reserved	2 Bits	
Size of Local Color Table	3 Bits	

Keterangan:

Image Separator	= tanda dimulainya *Image Descriptor* (desimal 44)
Image Left Position	= posisi ujung kiri citra (terhadap *Logical Screen Width*)
Image Top Position	= posisi ujung atas citra (terhadap *Logical Screen Height*)
Image Width	= lebar citra
Image Height	= tinggi citra
Local Color Table Flag	= *flag* keberadaan dari *Global Color Table* (0=tidak ada, 1=ada)
Interlace Flag	= *flag interlace* (0=*noninterlaced*, 1=*interlaced*)
Sort Flag	= *flag* pengurutan *Local Color Table* (0=tidak diurutkan, 1=diurutkan menurut warna yang paling sering digunakan)
Reserved	= tidak digunakan
Size of Local Color Table	= ukuran dari *Local Color Table* Jumlah warna = 2^(*Size of Local Color Table*+1)

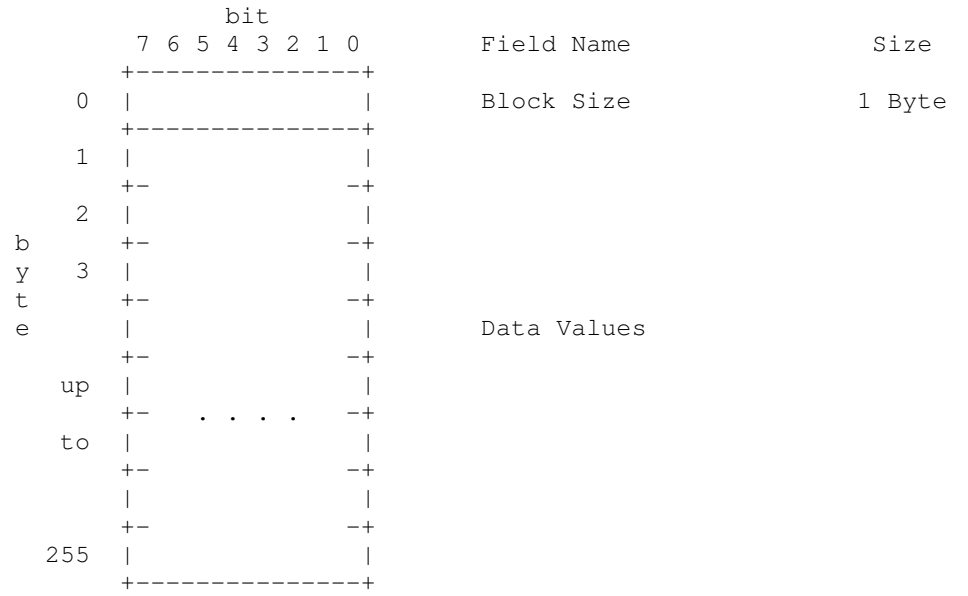
Gambar 2.4. Blok *Image Descriptor* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*



Gambar 2.5. *Global-Local Color Table* GIF

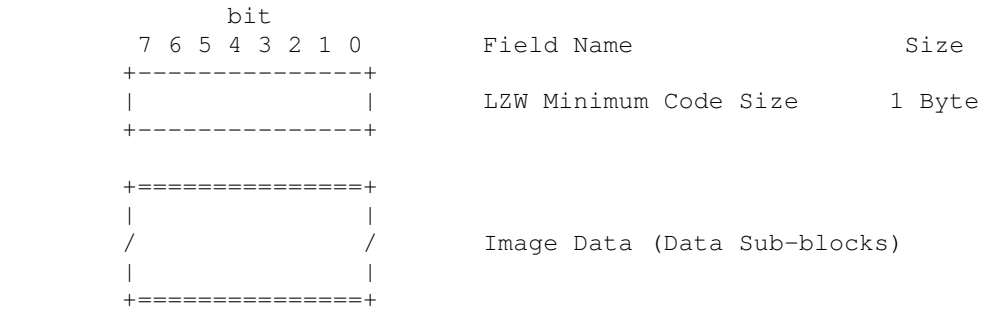
Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*



Keterangan:
Block Size = jumlah *byte* dalam *Data Sub-block* (desimal 0-255)
Data Values = data yang dikompresi secara LZW sebanyak *blocksize*

Gambar 2.6. *Data Sub-block* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*

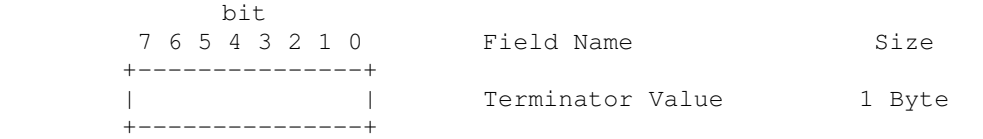


bit 7 6 5 4 3 2 1 0	Field Name	Size
	LZW Minimum Code Size	1 Byte
	Image Data (Data Sub-blocks)	

Keterangan:
LZW Minimum Code Size = jumlah *bit* yang digunakan pertama kali pada kompresi LZW
Misalkan untuk 256 warna, *LZW Code Size*-nya adalah 8.
Image Data = (lihat gambar 2.6)

Gambar 2.7. Blok *Table Based Image Data* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*

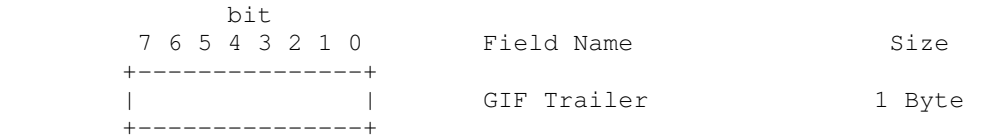


bit 7 6 5 4 3 2 1 0	Field Name	Size
	Terminator Value	1 Byte

Keterangan:
Terminator value = nilai yang menunjukkan berakhirnya suatu *data sub-block* (desimal 0)

Gambar 2.8. *Terminator Sub-block* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*

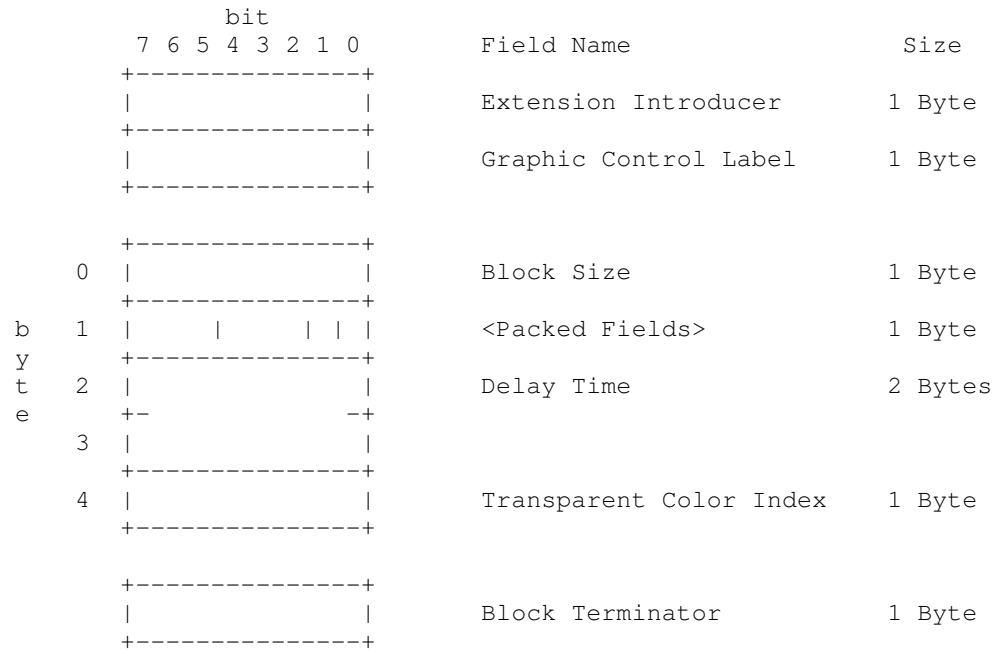


bit 7 6 5 4 3 2 1 0	Field Name	Size
	GIF Trailer	1 Byte

Keterangan:
GIF *Trailer* = nilai yang menunjukkan berakhirnya suatu data *stream* GIF (desimal 59)

Gambar 2.9. *Trailer* GIF

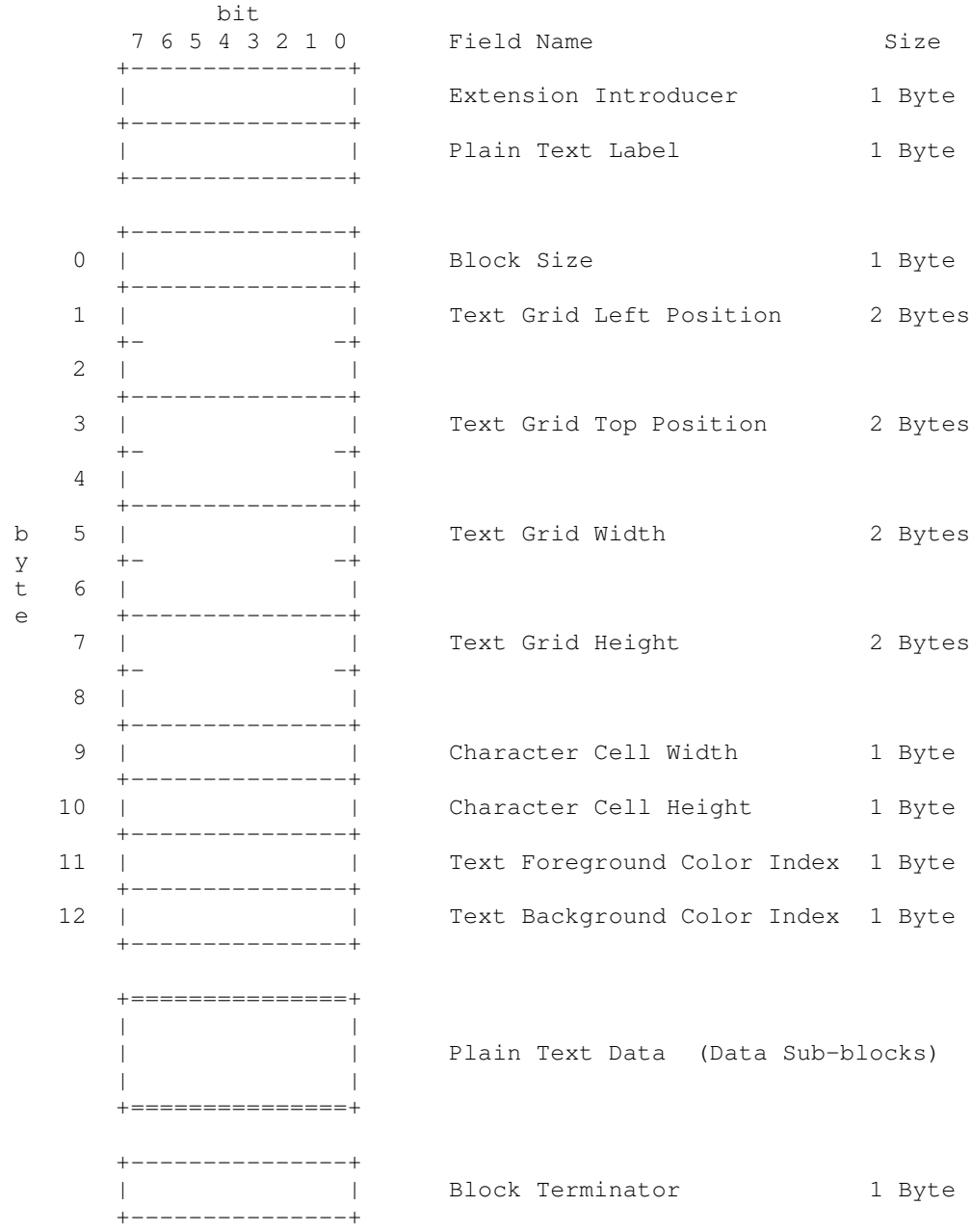
Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*



Keterangan:
Extension Introducer = tanda dimulainya Blok *Extension* (desimal 33)
Graphic Control Label = label dari *Graphic Control Extension* (desimal 249)
Block Size = bernilai 4

Gambar 2.10. *Graphic Control Extension* GIF

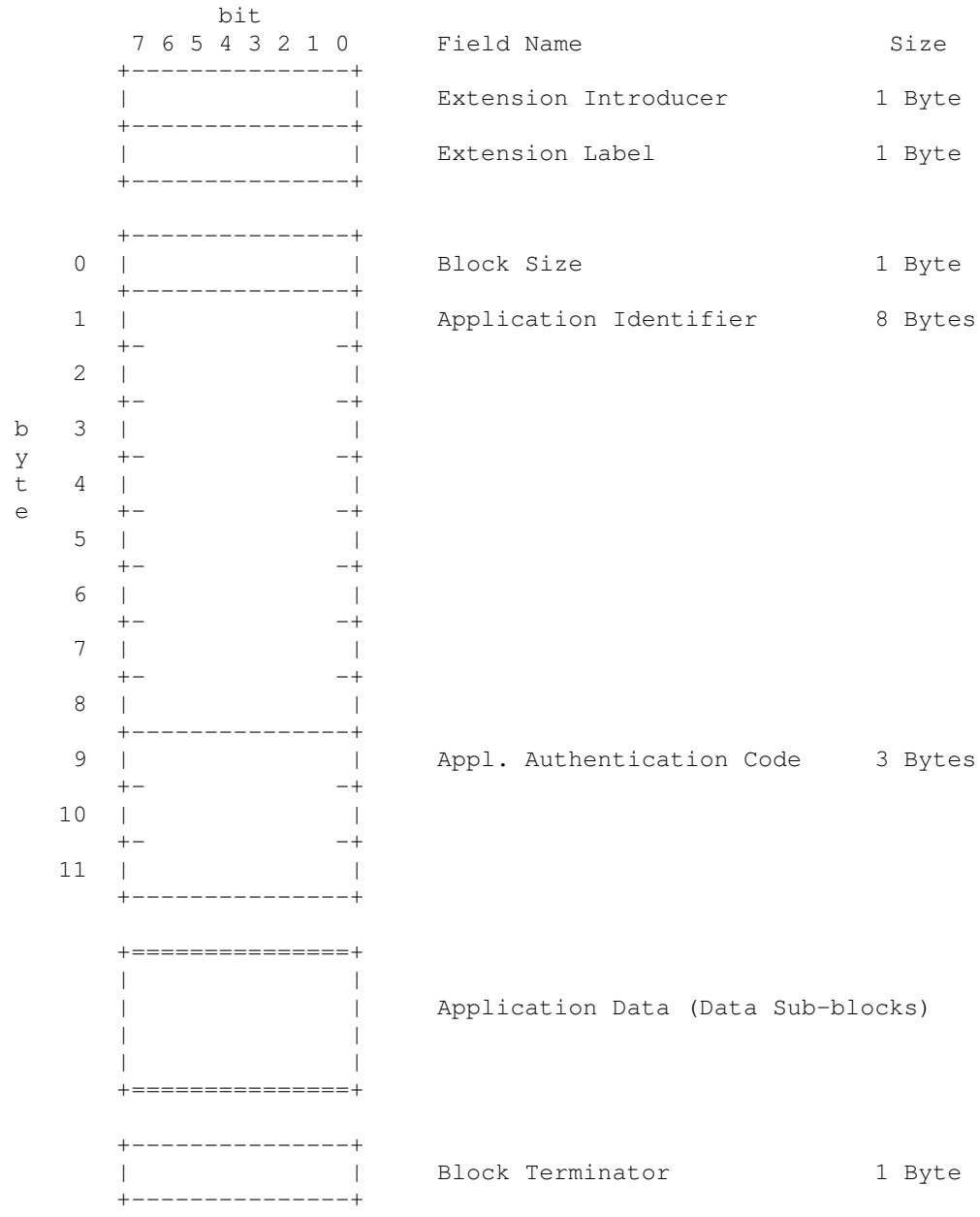
Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*



Keterangan:
Extension Introducer = tanda dimulainya Blok *Extension* (desimal 33)
Plain Text Label = label dari *Plain Text Extension* (desimal 1)
Block Size = bernilai 12

Gambar 2.11. *Plain Text Extension* GIF

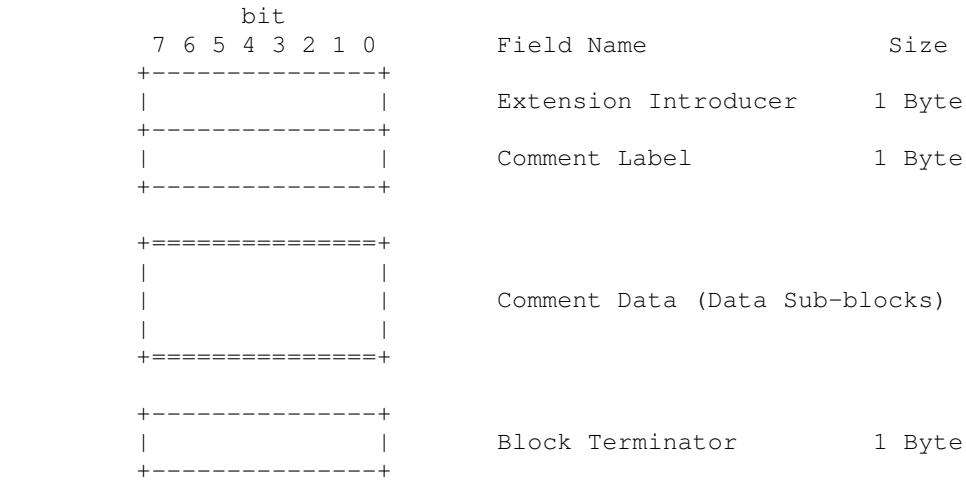
Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*



Keterangan:
Extension Introducer = tanda dimulainya Blok *Extension* (desimal 33)
Application Label = label dari *Application Extension* (desimal 255)
Block Size = bernilai 11

Gambar 2.12. *Application Extension* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*



Keterangan:
Extension Introducer = tanda dimulainya Blok *Extension* (desimal 33)
Comment Label = label *Comment Extension* (desimal 254)

Gambar 2.13. *Comment Extension* GIF

Sumber: *http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/GIF89a.txt*

2.4.2. Kompresi Citra GIF

GIF menggunakan variasi dari algoritma kompresi LZW (Lempel-Ziv dan Welch) yaitu dengan adanya penambahan *clear code* dan *end of image code* (*end code*) pada *hash table* setelah inisialisasi. Kompresi citra dimulai dengan penciptaan sebuah tabel *string* atau disebut juga *hash table*. Dilakukan inisialisasi terhadap *hash table* dengan nilai dari setiap warna yang digunakan, ditambah dengan *clear code* dan *end code*.

Proses kompresi dilakukan dengan melakukan pencocokan terhadap pola dari *byte-byte* yang ditemui pada citra *bitmap* dengan *key* pada *hash table*. *Code* dari *key* yang cocok lalu dituliskan pada citra GIF. Kemudian tambahkan ke *hash table* dengan nilai sebagai berikut:

key = *key* dari *code* yang cocok tadi + *byte* selanjutnya
code = *increament* dari *code* sebelumnya

Sebuah citra GIF dikodekan sampai 12 *bit* kompresi. Jadi *hash table* GIF dapat berisi maksimum $2^{12} = 4096$ elemen. *Hash table* tidak perlu disimpan dalam

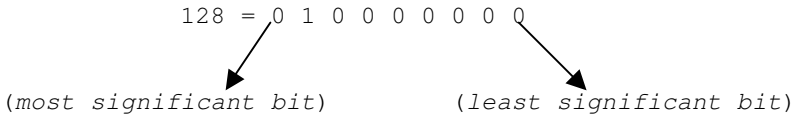
file, karena setiap *key* disimpan dalam *file* sebagaimana dikenali, sehingga dapat merekonstruksi *hash table* tersebut secara keseluruhan pada saat mendekompresi.

Citra GIF menggunakan pengembangan lain dari algoritma LZW, yaitu dengan menggunakan kode pembersihan (*clear code*). Kode ini memberitahu program dekompresor untuk menghapus dan melakukan inisialisasi ulang *hash table*-nya agar program tidak kehabisan memori karena proses penyimpanan semua pola *string* pada *hash table* tersebut memakan memori yang cukup besar pada saat program dijalankan.

Selain itu, *file* GIF diakhir dengan kode akhir citra (*end of image code* atau cukup disebut *end code*). Jika kode ini ditemui, maka program dekompresor akan berhenti melakukan dekompresi karena seluruh citra telah selesai didekompresi. Kode tambahan ini akan ditempatkan pada satu baris setelah baris terakhir yang dimiliki citra, dan tidak mengandung data citra sama sekali.

Dengan adanya kedua kode tambahan ini, penambahan kode baru ke tabel selalu bergeser dua dari kode yang diinisialisasi terakhir.

Karena data citra GIF bisa membengkak sampai 12 *bit*, maka sebelum dijadikan dalam bentuk *file*, terlebih dahulu data hasil kompresi tersebut di-*pack* menjadi per 8 *bit* dengan urutan penggabungan mulai dari *least significant bit* sampai *most significant bit*.



2.4.3. Dekompresi Citra GIF

Proses dekompresi citra GIF merupakan kebalikan dari proses kompresinya. Sebagaimana yang telah dijelaskan pada bagian kompresi, *hash table* tidak perlu disimpan dalam *file* karena proses dekompresi nantinya akan membentuk *hash table* yang sama.

Setiap *byte* dari data citra GIF dibaca dan di-*unpack*. Proses *unpack* tergantung pada nilai *LZW Code* yang terdapat pada blok *Table Based Image Data*. Jika *LZW Code* bernilai 8, maka jumlah *bit* awal dari data citra GIF sebelum di-*pack* adalah 9 *bit*.

Setelah di-*unpack*, dilakukan pencocokan dengan setiap *code* pada *hash table*.

Ada tiga kondisi yang mungkin muncul dalam proses pencocokan tersebut, yaitu:

Kondisi 1: Jika data citra GIF cocok dengan salah satu *code* dari *hash table*, maka gantikan data citra tersebut dengan *key* dari *code* yang cocok tadi. Kemudian tambahkan ke *hash table* dengan nilai sebagai berikut:

```
key    = prefix + suffix
code   = increament dari code sebelumnya
prefix = hasil dekompresi data citra sebelumnya
suffix = potongan byte pertama dari key (dari code
         yang cocok tadi)
```

Kondisi 2: Jika data citra GIF tidak ditemukan kecocokannya dengan salah satu *code* dari *hash table*, maka tambahkan ke *hash table* dengan nilai sebagai berikut:

```
key    = prefix + suffix
code   = increament dari code sebelumnya
prefix = hasil dekompresi data citra sebelumnya
suffix = potongan byte pertama dari prefix
```

Lalu gantikan data citra dengan *key* dari *code* yang baru dibentuk.

Kondisi 3: Jika data citra GIF merupakan *clear code*, maka *hash table* akan diinisialisasi dengan nilai dari setiap warna yang digunakan + *clear code* + *end code*. Berarti *hash table* akan berisi sebanyak

```
2^(color resolution+1) + clear code +end code
```

Karena adanya penambahan secara terus-menerus pada *hash table*, maka jumlah *bit* dari data citra GIF secara otomatis akan bertambah juga. Dari contoh di atas, jika jumlah kode dari *hash table* sudah mencapai 512 (kode selanjutnya adalah 512), maka proses *unpack* akan berubah dari 9 *bit* ke 8 *bit* menjadi 10 *bit* ke 8 *bit*.

2.5. Komponen *TJpegImage* Pada *Borland Delphi 6.0*

Komponen *TJpegImage* digunakan untuk memanipulasi citra yang menggunakan kompresi JPEG.

Beberapa *method* yang dimiliki oleh komponen *TJpegImage* adalah:

- *Create*

  Gunakan *method Create* untuk menciptakan sebuah objek citra JPEG pada saat *runtime*. *Create* mengalokasikan memori untuk sebuah objek citra JPEG.

- *LoadFromFile*

  Method LoadFromFile merupakan *method* turunan dari *TGraphic*. Digunakan untuk menge-*load* sebuah citra grafik yang tersimpan dalam *file*.

- *Free*

  Method *Free* merupakan *method* turunan dari *Tobject*. Digunakan untuk men-*destroy* sebuah objek dan me-*release* memori yang diasosiasikan untuk objek tersebut.

Salah satu *property* yang dimiliki oleh komponen *TJpegImage* adalah:

Compression Quality

type TJPEGQualityRange = 1..100;

property CompressionQuality: *TJPEGQualityRange*;

Property Compression Quality mengatur kualitas dari kompresi citra JPEG pada saat menulis citra JPEG. Semakin tinggi nilai *TJPEGQualityRange*, maka kualitas citra akan semakin baik, tetapi semakin besar pula ukuran *file*. Semakin rendah nilai *TJPEGQualityRange*, menghasilkan ukuran file yang lebih kecil, tetapi kualitas citra akan berkurang.

2.6. Komponen *TBitmap* Pada *Borland Delphi 6.0*

Komponen *TBitmap* digunakan untuk memanipulasi citra *bitmap*.

Beberapa *method* yang dimiliki oleh komponen *TBitmap* adalah:

- *Assign*

  Method Assign digunakan untuk meng-*copy* citra *bitmap* yang terdapat dalam *Source* menjadi sebuah objek *bitmap*.

- *SaveToFile*

  Method SaveToFile merupakan *method* turunan dari *TGraphic*. Digunakan untuk meng-*save* sebuah citra grafik ke dalam *file*.

- *Free*

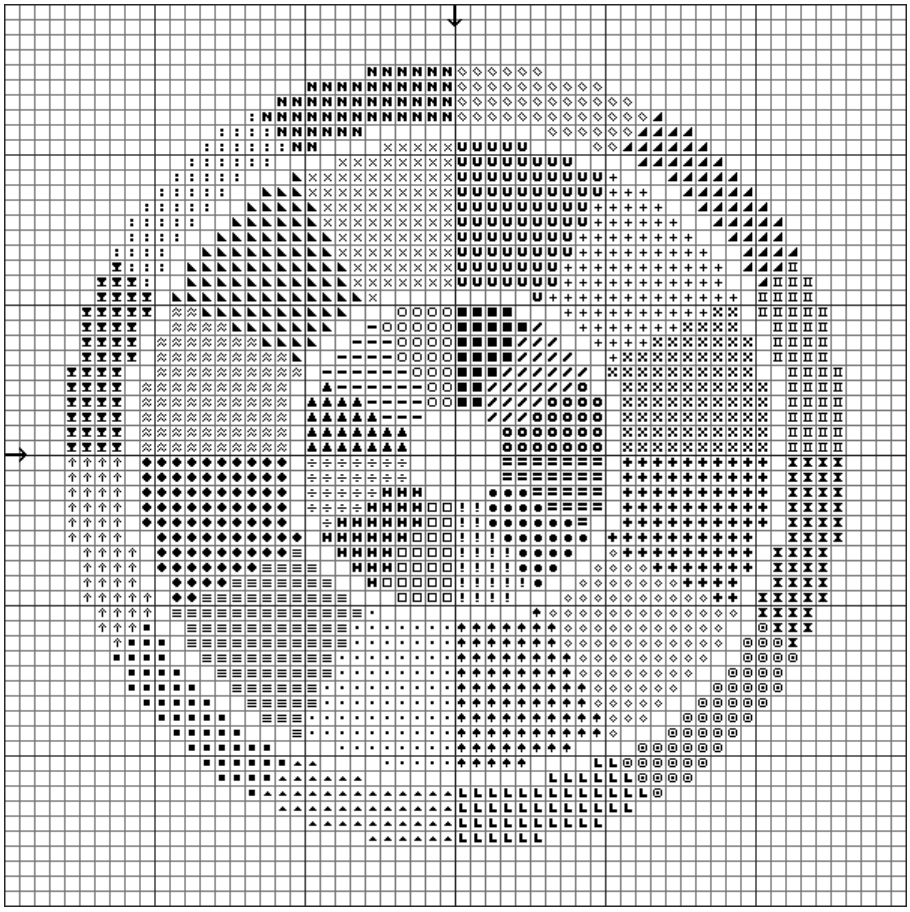
  Method *Free* merupakan method turunan dari *Tobject*. Digunakan untuk men-*destroy* sebuah objek dan me-*release* memori yang diasosiasikan untuk objek tersebut.

2.7. *Color Wheel* Benang DMC

36 warna utama dari benang DMC ditampilkan dalam gambar berikut ini:

Gambar 2.14. *Color Wheel* Benang DMC

Sumber: *http://www.jccniowa.org/~pgega/workshops/*



Sym	DMC	ANC	JPC	Description
	301	1049	----	Mahogany - med
	307	289	2288	Lemon
	319	218	6246	Pistachio Green - vy dk
	333	119	----	Blue Violet - vy dk
	340	118	7110	Blue Violet - med
	550	102	4107	Violet - vy dk
	552	99	4092	Violet - med
	554	96	4104	Violet - lt
	606	334	2334	Bright Orange-Red
	666	46	3046	Christmas Red
	718	88	4089	Plum
	731	924	6844	Olive Green - dk
	741	304	2314	Tangerine - med
	743	302	2294	Yellow - med
	747	158	7053	Sky Blue - vy lt
	772	259	6250	Yellow Green -vy lt
	776	24	3281	Pink - med
	791	178	7024	Cornflower Blue - vy dk

Sym	DMC	ANC	JPC	Description
	797	132	7143	Royal Blue
	800	144	7020	Delft - pale
	806	169	7169	Peacock Blue - dk
	815	43	3000	Garnet - med
	820	134	7024	Royal Blue - vy dk
	907	255	6001	Parrot Green - lt
	911	205	6205	Emerald Green - med
	915	1029	----	Plum - med
	920	1004	3337	Copper - med
	936	269	6269	Avocado Green - dk
	947	330	2330	Burnt Orange
	955	206	6020	Nile Green - lt
	3078	292	2292	Golden Yellow - vy lt
	3341	328	----	Apricot
	3609	85	4085	Plum - ul lt
	3808	----	----	Turquoise - ul vy dk
	3825	1047	----	Pale Pumpkin
	3829	901	2876	Old Gold - vy dk

Gambar 2.15. Keterangan *Color Wheel* Benang DMC

Sumber: http://www.jccniowa.org/~pgega/workshops/