# Capitolo 4

# Formati grafici

# Indice

## 4.1 XBM

Il formato *XBitMap* è stato creato per rappresentare piccole immagini mediante un vettore di dati in formato `ASCII`. L'immagine può essere contenuta in un file esterno o essere inclusa in un programma scritto in C. Le principali caratteristiche di questo formato sono mostrate nella tabella seguente.

| | |
|---|---|
| Nome | XBitMap |
| Tipo | bitmap |
| Colori | mono |
| Compressione | nessuna |
| Dimensione massima | illimitata |
| Più immagini per file | si |
| Formato numerico | `ASCII` |
| Creatore | Consorzio X |

La descrizione dell'immagine inizia con le definizioni

```
#define <nome>_width <larghezza in pixel>
#define <nome>_height <altezza in pixel>
```

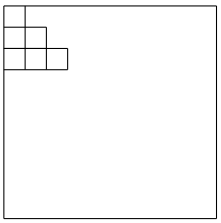
e continua con

```
static char <nome>_bits[] = {...};
```

Ad esempio, la descrizione

```
#define x_width 8
#define x_height 8
static char x_bits[] = {
   0x01, 0x03, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00};
```

corrisponde al disegno seguente.



Ad ogni bit posto a uno corrisponde un *pixel* "nero". I bit sono raggruppati in *byte* rappresentati in base esadecimale.

Questi file si possono editare ad esempio con uno dei programmi seguenti:

- /usr/bin/X11/bitmap (sistemi `UNIX` e `Linux`)
- decw$utils:bitmap (`OpenVMS`)

## 4.2 XPM

Con il formato *XPixMap* si possono rappresentare immagini a colori, in tonalità di grigio e in bianco e nero. Le principali caratteristiche sono mostrate nella tabella che segue.

| | |
|---|---|
| Nome | XPixMap |
| Tipo | bitmap |
| Colori | senza limiti |
| Compressione | nessuna |
| Dimensione massima | |
| Più immagini per file | si |
| Formato numerico | `ASCII` |
| Creatore | Groupe Bull |

Questo formato è descritto nel documento *XMP Manual, The X PixMap Format* pubblicato il primo febbraio 1996 da *Arnaud Le Hors - Groupe BULL.*

Il file è diviso in sezioni. Nella prima sezione sono contenute la larghezza, l'altezza, la quantità di colori, il numero di *byte* per *pixel*. La prossima sezione contiene la definizione dei colori usati. Nella descrizione vengono usati i simboli seguenti:

| simbolo | descrizione |
|---|---|
| c | colore |
| m | colore equivalente nel caso bianco e nero |
| g | colore equivalente nel caso di tonalità di grigio |
| s | simbolo del colore definito |

Nella prossima sezione viene mappata l'immagine facendo uso dei simboli definiti nella sezione precedente come mostrato nell'esempio seguente. Le definizioni dei colori sono quelle di `X-Windows`.

```
/* XPM */
static char * stringa[] = {
/* larghezza, altezza, numero colori, caratteri per pixel */"40 20 4 1",
/* colori */
"   c white     m white  s sfondo bianco ",
"R  c red       m black  s rosso ",
"Y  c yellow    m white  s giallo ",
"N  c black     m black  s nero ",
/* pixel */
"NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN",
"N                                      N",
"N                                      N",
"N           R              R           N",
"N            R            R            N",
"N             R          R             N",
"N              R        R              N",
"N               R      R               N",
"N                R   R                 N",
"N                 R R                  N",
"N                  R                   N",
"N                 R  R                 N",
"N                R    R                N",
"N               R      R               N",
"N              R        R              N",
"N             R          R             N",
"N            R            R            N",
"N           R              R           N",
"N          R                R          N",
"N                                      N",
"NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN",
} ;
```

Questo formato è documentato in:

Arnaud Le Hors
XPM Manual - The XPixMapFormat
BULL

## 4.3 BMP

Le principali caratteristiche sono illustrate nella tabella (4.1). Questo formato è no-

| | |
|---|---|
| Nome | Microsoft Windows Bitmap |
| Tipo | bitmap |
| Colori | mono, 4 bit, 8 bit, 16 bit, 24 bit, 32 bit |
| Compressione | nessuna, RLE |
| Dimensione massima | 64K x 64K pixel |
| Più immagini per file | no |
| Formato numerico | Little endian |
| Creatore | Microsoft Corporation |

Tabella 4.1: BMP

tevolmente evoluto negli anni; a partire dalla versione 1.X, dipendente dall'*hardware* usato, è giunto alla versione 3.X, noto anche sotto il nome di *Device Independent Bitmap - DIB*.

### 4.3.1 Struttura dei file della versione 3.X

I file contengono le sezioni seguenti:

- BitmapFileHeader
- BitmapInfoHeader
- Palette
- BitmapInfo

#### 4.3.1.1 BitmapFileHeader

È definito mediante la struttura

```
typedef struct _Win3xBitmapHeader {
  WORD  ImageFileType;    /* Tipo del file immagine */
                          /* (sempre 0x4D42 ("BM")) */
  DWORD FileSize;         /* Dimensione del file in byte */
  WORD  Reserved1;        /* Sempre 0 */
  WORD  Reserved2;        /* Sempre 0 */
  DWORD ImageDataOffset;  /* Offset dell'immagine in byte dall'inizio*/
} WIN3XHEAD;
```

#### 4.3.1.2 BitmapInfoHeader

È definito mediante la struttura

```
typedef struct _Win3xBitmapInfoHeader {
  DWORD HeaderSize;          /* Dimensione della testata */
  LONG  ImageWidth;          /* Larghezza dell'immagine in pixel */
  LONG  ImageHeight;         /* Altezza dell'immagine in pixel */
  WORD  NumberOfImagePlanes; /* Numero di piani (sempre 1) */
  WORD  BitsPerPixel;        /* (1, 4, 8 , 24 o 32) */
  DWORD CompressionMethod;   /* 0: BI_RGB  non compresso con BitPerPixel=1,4,8,16,24,32
                             /* 1: BI_RLE8 compressione RLE con BitPerPixel=8
                             /* 2: BI_RLE4 compressione RLE con BitPerPixel=4
                             /* 3: BI_BITFIELDS Bitfields  con BitPerPixel=16, 24 o 32*/
                             /* 0x41424752: RGBA   con BitPerPixel=16 o 32 */
                             /* 0x54424752: RGBT   con BitPerPixel=16 o 32 */
  DWORD SizeOfBitmap;        /* in byte, (puo' essere 0 se non compresso) */
  DWORD HorzResolution;      /* Risoluzione orizzontale in pixel per metro */
  DWORD VertResolution;      /* Risoluzione verticale in pixel per metro */
  DWORD NumColorsUsed;       /* Numero colori della paletta usati */
                             /* se =0, calcolare con la formula "1 << BitsPerPixel" */
  DWORD NumSignificantColors;/* Numero colori importanti (0: tutti importanti) */
} WIN3XINFOHEAD;
```

Nel caso in cui `NumColorsUsed` è zero, il numero di colori usati è quello massimo possibile ed è calcolabile con la formula $1 << $ `BitsPerPixel`.

#### 4.3.1.3 Tavola dei colori

La tavola dei colori è presente nei casi seguenti:

- se `BitsPerPixel` è 1, 4 o 8
- se `BitsPerPixel` è 16 o 32 e `CompressionMethod` è `BI_BITFIELDS`. In questo caso la tavola dei colori contiene una terna di `DWORD` che specifica la maschera necessaria per ricavare le componenti `RGB` di ogni *pixel*. Ad esempio, nel caso di `BitPerPixel=16`, se si specifica la maschera `0xF800`, `0x07E0`, `0x001F`, i bit dedicati alla componente `R` sono 5, quelli per la componente `G` 6 e quelli per la componente `B` 5.

La dimensione della tavola dipende dal valore di `NumColorsUsed` e ogni elemento può assumere la struttura seguente:

- se `BitsPerPixel` è 1 4 o 8:

  ```
  typedef struct _Win3xPalette {
    RGBQUAD Palette[]; /* fino a 2, 16 o 256 elementi */
  } WIN3XPALETTE;
  ```

  Il tipo è `RGBQUAD` è definito come segue

```
typedef struct _Win3xRgbQuad {
  BYTE Blue;       /* Componente del blu */
  BYTE Green;      /* Componente del verde */
  BYTE Red;        /* Componente del rosso */
  BYTE Reserved;   /* Valore di riserva */
} RGBQUAD
```

- se `BitsPerPixel` è 16 o 32 e `CompressionMethod` è `BI_BITFIELDS`, la tavola è formata da tre `DWORD` che rappresentano la maschera per ricavare le componenti RGB di ogni *pixel*.

  Esempio:

  BitPerPixel=16
  Maschera: 0xF800,0x07E0, 0x001F
  Si ricava: red: 5 bit, green: 6 bit, blu: 5 bit.

Se i colori bianco e nero sono presenti nella paletta, essi occupano le prime due posizioni.

I colori importanti sono quelli localizzati all'inizio della tabella.

#### 4.3.1.4 BitmapInfo

Nel caso in cui la tavola dei colori è presente, i dati di questa sezione sono da interpretare secondo la tabella (4.2). Nei casi in cui i dati non rappresentano indici alla

| `BitsPerPixel` | |
|---|---|
| 1 | ogni *byte* rappresenta 8 *pixel* , `MSB` è il valore del primo *pixel* |
| 4 | ogni *byte* rappresenta 2 *pixel*<br>il *nibble* più significativo è il primo *pixel* |
| 8 | ogni *byte* corrisponde a un *pixel* |
| 16 | ogni *word* corrisponde a un *pixel* |
| 32 | ogni *dword* corriponde a un *pixel* |

Tabella 4.2: Interpretazione dei dati in `BitmapInfo` con tavola dei colori

paletta del colori, essi sono da interpretare secondo la tabella (4.3). In generale viene codificata e decodificata una linea dopo l'altra. Nel caso di immagini non compresse, ogni linea è sempre formata da un multiplo di 4 *byte* ed è riempita artificialmente con `NULL`, se necessario.
I dati vengono associati di regola ai *pixel* partendo dall'angolo in basso a sinistra e risalendo verso l'alto. Se però `ImageHight` è negativo, l'interpretazione avviene a partire dall'angolo in alto a sinistra scendendo verso il basso.

| BitsPerPixel | Interpretazione |
|---|---|
| 16 | 5 *bit* per ogni componente RGB (*bit* più significativo non usato) |
| 24 | un *byte* per ogni componente RGB, nell'ordine i valori del blu, verde e rosso |
| 32 | un *byte* per ogni componente RGB, (quarto *byte* non usato) |

Tabella 4.3: Interpretazione dei dati in BitmapInfo senza tavola dei colori

Se CompressionMethod è RGBA, per ogni *pixel*, oltre alle componenti RGB del colore, viene memorizzato il valore *alpha* che definisce la trasparenza. I valori ammessi per BitPerPixel sono 16 o 32. L'interpretazione dei dati avviene nello stesso modo come nel caso di CompressionMethod = BI_BITFIELDS.

Se CompressionMethod è RGBT, per ogni *pixel*, oltre alle componenti RGB del colore, nel bit più significativo viene memorizzato un'informazione sulla trasparenza. Se il bit è posto a 1, il *pixel* è visibile, altrimenti è completamente trasparente. Anche in questo caso i valori ammessi per BitPerPixel sono 16 o 32.

### 4.3.2 Esempio di file BMP

Il contenuto del file seguente rappresenta un file BMP non compresso.

```
00 28 00 00 00 3A 00 00 00 00 00 00 04 3A 4D 42 BM:.......:...(. 000000
00 00 00 01 00 01 00 00 00 40 00 00 00 7F 00 00 .....@......... 000010
00 01 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ................ 000020
00 00 00 00 00 01 00 40 40 FF 00 00 00 01 00 00 .......@@....... 000030
 ..............................................  ..............  ......
 ..............................................  ..............  ......
```

Interpretandolo in base a quanto visto in precedenza si ottiene l'informazione seguente

| | | |
|---|---|---|
| 4D 42 | BMP | ImageFileType |
| 00 00 04 3A | 1082 | FileSize (little endian) |
| 00 00 | BMP | Reserved1 |
| 00 00 | BMP | Reserved2 |
| 00 00 00 3A | 58 | offset |
| 00 00 00 28 | 40 | HeaderSize |
| 00 00 00 7F | 127 | Imagewidth |
| 00 00 00 40 | 64 | ImageHeight |
| 00 01 | 1 | NumberOfImagePlanes |
| 00 01 | 1 | BitsPerPixel |
| 00 00 00 00 | 0 (non compresso) | CompressionMethod |
| 00 00 04 00 | 1024 | SizeofBitmap |
| 00 00 00 00 | non specificata | HorzResolution |
| 00 00 00 00 | non specificata | VertResolution |
| 00 00 00 01 | 1 | NumColorsUsed |
| 00 00 00 01 | 1 | NumSignificantColors |
| 00 40 40 FF | blu | Colore 0 |
| ... | | Dati (tutti zero) |

Il file descrive un'immagine monocolore (blu) di $127 \times 64 = 8128$ *pixel*.
Siccome l'immagine è larga 127 *pixel* e ogni linea è sempre formata da un multiplo di 4 byte, si perde un bit per ogni riga. In totale si perdono quindi 64 bit = 8 byte. Per ottenere lo spazio totale occupato dall'immagine occorre aggiungere ai $127 \times 64 = 8128$ bit corrispondenti a 1016 byte, 58 byte di testata e 8 byte persi. Si giustifica così la `FileSize` di 1082 byte contenuta nella tabella.

### 4.3.3 L'algoritmo di compressione RLE

Il formato `BMP` può comprimere i dati usando l'algoritmo di compressione `RLE`, descritto nei suoi principi generali nel capitolo "Compressione dati" del corso "Algoritmi e Matematica numerica". In `BMP` la compressione può essere orientata a *byte* (`BI_RLE8`) o a *nibble* (`BI_RLE4`). In ogni caso i dati sono organizzati a pacchetti.

#### 4.3.3.1 Compressione orientata a *byte* (`BI_RLE8`, 256 colori)

- **encoded mode**

  Se il primo *byte* del pacchetto è diverso da zero, esso è da interpretare come contatore binario e rappresenta il numero di volte che il *byte rappresentante* seguente deve essere ripetuto.
  Se il primo *byte* del pacchetto è uguale a zero, e il secondo *byte* contiene 0,1 o 2 l'intepretazione si deduce dalla tabella seguente.

| valore | interpretazione | |
|---|---|---|
| 0 | fine della linea | |
| 1 | fine dell'immagine | |
| 2 | cambiamento di coordidate | i due *byte* successivi<br>sono gli *offset* ($> 0$) orizzzontale e verticale<br>del prossimo *run* (sequenza delta). |

- **absolute mode**

  Se il primo *byte* è zero e il secondo è compreso fra 3 e 255, gli $n$ *byte* rimanenti del pacchetto sono una stringa letterale non compressa.

#### 4.3.3.2 Compressione orientata a nibble (BI_RLE4, 16 colori)

- **encoded mode**

  Se il primo *byte* del pacchetto è diverso da zero, esso indica il numero di *pixel* da disegnare in base alle informazioni contenute nel secondo *byte*. Quest ultimo *byte* contiene due indici di colori, uno nei 4 bit meno significati e l'altro nei rimanenti. Il primo *pixel* è disegnato usando il primo indice, il prossimo usando il secondo. Per il terzo si usa ancora il primo indice e così di seguito.
  Se il primo *byte* del pacchetto è nullo, il secondo *byte* è da interpretare nello stesso modo come nella compressione orientata a *byte*.

- **absolute mode**

  Se il primo *byte* è nullo e il secondo contiene il numero di indici descritti nel pacchetto corrispondenti ad altrettanti *pixel*.

Le tecniche di compressione descritte non sono adatte per comprimere file che usano 24 bit per *pixel*.
In generale per decodificare un file BMP è suffciente un *buffer* di lavoro corrispondente a una linea. Le sequenze delta però possono richiedere un *buffer* grande quanto l'intera immagine complicando quindi il processo di decodifica.

### 4.3.4 Esempio di file BMP compresso

Consideriamo il file con il contenuto seguente che rappresenta la medesima immagine dell'esempio precedente.

```
00 28 00 00 00 3A 00 00 00 00 00 00 01 3C 4D 42 BM.....:...(. 000000
00 01 00 08 00 01 00 00 00 40 00 00 00 7F 00 00 .....@..... 000010
00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............  000020
00 80 00 00 00 80 00 40 40 FF 00 00 00 01 00 00 .......@@.......  000030
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ..............  000040
```

```
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000050
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000060
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000070
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000080
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000090
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 0000A0
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 0000B0
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 0000C0
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 0000D0
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 0000E0
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 0000F0
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000100
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000110
00 80 00 00 00 80 00 00 00 80 00 00 00 80 00 00 ................ 000120
            01 00 00 00 00 80 00 00 00 80 00 00 ............ 000130
```

Analizzando il suo contenuto si ottiene l'informazione seguente

| | | |
|---|---|---|
| 4D 42 | BMP | ImageFileType |
| 00 00 01 3C | 316 | FileSize (little endian) |
| 00 00 | BMP | Reserved1 |
| 00 00 | BMP | Reserved2 |
| 00 00 00 3A | 58 | offset |
| 00 00 00 28 | 40 | HeaderSize |
| 00 00 00 7F | 127 | Imagewidth |
| 00 00 00 40 | 64 | ImageHeight |
| 00 01 | 1 | NumberOfImagePlanes |
| 00 08 | 8 | BitsPerPixel |
| 00 00 00 01 | 1 (BI_RLE8) | CompressionMethod |
| 00 00 00 00 | 0 | SizeofBitmap |
| 00 00 00 00 | non specificata | HorzResolution |
| 00 00 00 00 | non specificata | VertResolution |
| 00 00 00 01 | 1 | NumColorsUsed |
| 00 00 00 01 | 1 | NumSignificantColors |
| 00 40 40 FF | blu | Colore 0 |
| 00 00 00 80 | | 64 volte |
| ..... | | |
| 00 01 | | EOF |

## 4.4 GIF

Le principali caratteristiche sono illustrate nella tabella seguente:

| | |
|---|---|
| Nome | Graphics Interchange Format |
| Tipo | bitmap |
| Colori | 1 ... 8 bit |
| Compressione | LZW |
| Dimensione massima | 64Kb x 64Kb |
| Più immagini per file | si |
| Formato numerico | Little endian (LSB precede MSB) |
| Creatore | Compuserve Inc. |

### 4.4.1 Struttura di un file GIF87a

La prossima tabella mostra la struttura di un file versione `GIF87a`.

| | |
|---|---|
| Informazioni generali | Header |
| | Logical Screen Descriptor |
| | eventuale Global Color Table |
| prima immagine | Image Descriptor |
| | Local Color Table |
| | Image Data |
| seconda immagine | Image Descriptor |
| | Local Color Table |
| | Image Data |
| ... | |
| | GIF Terminator (0x3B) |

Il formato è descrittao nel documento:
*GIF - Graphics Interchange Format: A standard defining a mechanism for the storage and transmission of raster-based graphics information* del 15 giugno 1987, pubblicato da *CompuServe Incorporated, 5000 Arlington Centre Blvd. Columbus, Ohio 43220.*

Le informazioni generali sono contenute nei blocchi `Header`, `Logical Screen Descriptor` e `Global Color Table`. Ogni immagine è composta da un *Image Descriptor*, una *Local Color Table* e dagli *image data*.

#### 4.4.1.1 Header

La testata ha una lunghezza di 6 *byte*. I primi tre *byte* contengono GIF, gli altri tre la versione (87a).

| | | |
|---|---|---|
| (logical) Screen Width | unsigned word | |
| (logical) Screen Height | unsigned word | |
| Global Color Table Flag [1] | 1 bit | 1→ esiste |
| Color Resolution<br>Sort Flag[2]<br>Size of Global Color Table | 3 bit<br>1 bit<br>3 bit | → color resolution + 1<br><br>→ $2^{size+1}$ |
| Background color index | byte | |
| Pixel Aspect Ratio | byte | se $\neq 0$: altezza/larghezza<br>$\approx \frac{Pixel\ Aspect\ Ratio + 15}{64}$ |

Tabella 4.4: Logical Screen Descriptor

#### 4.4.1.2 Logical Screen Descriptor

La sua struttura è mostrata nella tabella (4.4). La presenza della `Global Color Table` segnalata dal bit più significativo del *byte*) corrispondente. Il bit stabilisce anche come interpretare il `Background Color Index`: se è uno, il valore di quest'ultimo campo deve essere usato. Il campo `Colour Resolution` specifica il numero di bit disponibili per rappresentare i colori. Il `Sort Flag` indica se la `Global Color Table` è ordinata in ordine di importanza decrescente. Questa informazione è utile a un decodificatore che dispone di pochi colori per conoscere la distribuzione della frequenza dei colori presenti.
Se il campo `Global Color Table` è posto a uno, la `Size of Global Color Table` permette di calcolare il numero di *byte* della `Global Color Table` mediante la formula $2^{size+1}$. Il `Background Color Index` è l'indice nella `Global Color Table` per lo sfondo. Il *Pixel Aspect Ratio* è un fattore per calcolare in modo approssimato il quoziente fra la larghezza e l'altezza del *pixel*. Se il valore è diverso da zero, il quoziente si calcola con la formula:

$$\frac{Pixel\ Aspect\ Ratio + 15}{64}$$

#### 4.4.1.3 Global Color Table

È una sequenza di *byte* di dimensione $3 \cdot 2^{size+1}$. Le terne di *byte* rappresentano l'intensità RGB dei colori associati agli indici a partire da 0. È usata dalle immagini quando non esiste una *Local Color Table*.

[1] Usata quando non esiste la tavola locale
[2] 1 → tavola ordinata in ordine di importanza

#### 4.4.1.4 Image Descriptor

La struttura del blocco è descritta nella tabella (4.5). Il blocco contiene l'informa-

| | |
|---|---|
| Image separator | 0x2C (byte) |
| Image Left Position | unsigned word |
| Image Top Position | unsigned word |
| Image Width | unsigned word |
| Image Height | unsigned word |
| Local Color Table Flag | 1 bit |
| Interlace Flag | 1 bit |
| Sort Flag | 1 bit |
| Reserved | 2 bit |
| Size of Local Color Table | 3 bit |

Tabella 4.5: Image Descriptor

zione necessaria per interpretare i dati che descrivono l'immagine. Le coordinate indicate sono riferite al *Logical Screen* e sono espresse in *pixel*. L'*Interlace flag* indica se l'immagine è interlacciata o meno. Le righe di un'immagine interlacciata sono organizzate secondo lo schema mostrato nella tabella (4.6). Nella tabella (4.7) ven-

| | | |
|---|---|---|
| Gruppo 1 | Ogni otto righe, partendo dalla riga 0 | Primo passo |
| Gruppo 2 | Ogni otto righe, partendo dalla riga 4 | Secondo passo |
| Gruppo 3 | Ogni quattro righe, partendo dalla riga 2 | Terzo passo |
| Gruppo 4 | Ogni due righe, partendo dalla riga 1 | Quarto passo |

Tabella 4.6: Gruppi di interlacciamento

gono mostrati i passi con cui viene disegnata un'immagine di 20 righe interlacciata.

| Riga | Passo |
|---|---|
| 0 | 1 |
| 1 | 4 |
| 2 | 3 |
| 3 | 4 |
| 4 | 2 |
| 5 | 4 |
| 6 | 3 |
| 7 | 4 |
| 8 | 1 |
| 9 | 4 |
| 10 | 3 |
| 11 | 4 |
| 12 | 2 |
| 13 | 4 |
| 14 | 3 |
| 15 | 4 |
| 16 | 1 |
| 17 | 4 |
| 18 | 3 |
| 19 | 4 |

Tabella 4.7: Esempio di interlacciamento

#### 4.4.1.5 Local Color Table

La sua presenza è segnalata dal bit *Local Color Table Flag* posto a uno. È usata dall'immagine immediatamente seguente.

#### 4.4.1.6 Image Data

I dati dell'immagine sono strutturati nel modo seguente:

| | | |
|---|---|---|
| Initial code size | byte | |
| Block size | byte | sottoblocco |
| dati | | |
| Block size | byte | sottoblocco |
| dati | | |
| ··· | | sottoblocco |
| Block terminator (0) | byte | |

L'*Initial code size* è la lunghezza iniziale in bit dei codici prodotti dall'algoritmo di compressione descritto in seguito. Ogni sottoblocco inizia con un contatore (*block size*), che specifica il numero di *byte* che seguono, che possono essere al massimo 255. L'ultima *block size* contiene 0. I *byte* di dati sono ottenuti impacchettando i codici prodotti dall'algoritmo di compressione LZW con codifica a lunghezza variabile. L'algoritmo comprime gli indici alla tavola dei colori attiva. Ogni indice corrisponde a un *pixel*; la sequenza è da sinistra verso destra e dall'alto verso il basso.
In generale il valore di *block size* è perciò diverso dal numero di codici trasmessi.

#### 4.4.1.7 Compressione Lempel-Ziv-Welch

Il principio di funzionamento dell'algoritmo viene trattato nel capitolo "Compressione dati" del corso "Algoritmi e Matematica numerica". Per la variante usata nel formato `GIF` valgono le seguenti considerazioni:

- L'algoritmo costruisce una tabella di codici corrispondenti a sequenze di *pixel*. L'algoritmo fissa la $Initial\ CodeSize$, cioè il numero di bit con cui rappresentare il codice. Di regola questo numero coincide con il numero di bit necessari per descrivere i colori. Immagini in bianco e nero usano tuttavia una lunghezza minima di 2 bit. Se il numero di sequenze da codificare supera il valore massimo del codice, il numero di bit è incrementato di uno. La lunghezza massima dei codici è 12 e quindi il suo valore massimo vale `0xFFF (4095)`.

- Un *ClearCode* speciale viene usato per riinizializzare le tavole usate dall'algoritmo. Il suo valore è $2^{CodeSize}$. Il *ClearCode* può apparire in ogni punto della sequenza di dati e deve essere scritto come primo carattere di ogni nuova sequenza di dati.

- Il codice *EndOfInformation* indica la fine di ogni sequenza di dati. Il suo valore è $2^{CodeSize} + 1$. Questo codice deve essere l'ultimo della sequenza.

- Il primo codice disponibile per la compressione di sequenze di indici ha il valore $ClearCode + 2$, cioè ha lunghezza $CodeSize + 1$.

### 4.4.2 Struttura di un file GIF89a

Questa struttura è descritta nel documento *Graphics Interchange Format Version 89a* del 31 luglio 1990, pubblicato da *CompuServe Incorporated, Columbus, Ohio.*

La prossima tabella mostra la struttura tipica di una file `GIF89a`.

| | |
|---|---|
| Informazioni | Header (GIF89a) |
| | Logical Screen Descriptor |
| | eventuale Global Color Table |
| prima immagine | eventuale Comment Extension |
| | eventuale Application Extension |
| | eventuale Graphic Control Extension |
| | Image Descriptor |
| | eventuale Local Color Table |
| | image data |
| seconda immagine | eventuale Comment Extension |
| | Image Descriptor |
| | eventuale Local Color Table |
| | image data |
| ... | |
| | Trailer (0x3B) |

La struttura è definita in termini di blocchi e sottoblocchi. Ad eccezione del *Logical Screen Descriptor* e della *Global Color Table* che concernono tutta la stringa di dati, gli altri blocchi di controllo hanno validità locale. I blocchi sono identificati da un *label* in base alla tabella seguente:

| Nome del blocco | | Label | Estensione | Versione |
|---|---|---|---|---|
| Application Extension | Opzionale | 0xFF | si | 89a |
| Comment Extension | Opzionale | 0xFE | si | 89a |
| Global Color Table | Opzionale (1) | | no | 87a |
| Graphic Control Extension | Opzionale | 0xF9 | si | 89a |
| Header | Obbligatorio | | no | |
| Image Descriptor | Opzionale | 0x2C | no | 87a |
| Local Color Table | Opzionale | | no | 87a |
| Logical Screen Descriptor | Obbligatorio (1) | | no | 87a |
| Plain Text Extension | Opzionale | 0x01 | si | 89a |
| Trailer | Obbligatorio (1) | 0x3B | no | 87a |

(1) Possono essere presenti una volta sola.

I blocchi possono essere classificati in tre gruppi

- `Control Block`
- `Graphic-Rendering Block`
- `Special Purpose Block`

#### 4.4.2.1 Control Block

Sono classificati blocchi di controllo:

- `Header`
- `Logical Screen Descriptor`
- `Global Color Table`
- `Graphic Control Extension`
  Contiene parametri usati durante l'elaborazione di un *Graphic-Rendering Block*. L'estensione ha valore solo per il prossimo blocco grafico, è opzionale e contiene un solo sottoblocco. Il blocco può apparire davanti a qualsiasi immagine, ma una volta sola. La struttura di questa estensione è mostrata nella tabella (4.8).

| | |
|---|---|
| Extension Introducer (0x21) | byte |
| Graphic Control Label (0xF9) | byte |
| Block Size (4) | byte |
| Reserved | 3bit |
| Disposal Method | 3 bit |
| User Input Flag | 1 bit |
| Transparent Color Flag | 1 bit |
| Delay Time | unsigned |
| Transparent Color Index | byte |
| Block Terminator (0) | byte |

Tabella 4.8: Graphic Control Extension

  - *Disposal Method*
    Specifica il modo di trattare l'immagine dopo la visualizzazione. I valori possibili sono commentati nella tabella (4.9).
  - *User Input Flag*
    Se il *bit* è impostato, un'azione di *input* è richiesta prima di continuare. Se si usa il *Delay time* e il *bit* è impostato, l'azione continua all'arrivo di uno dei due eventi.

| Valore | significato |
|---|---|
| 0 | non specificato |
| 1 | lasciare visualizzata l'immagine |
| 2 | ripristinare il colore di *background* |
| 3 | ripristinare il contenuto presente in precedenza sul *display* |

Tabella 4.9: Disposal Method

  - *Trasparent Color Flag* (*bit* meno significativo)
    Se il *bit* è impostato, il campo *Transparent Color Index* è definito.
  - *Delay time*
    Se il suo valore è diverso da zero, questo campo specifica (in centesimi di secondo) quanto tempo attendere prima di procedere con l'analilsi del flusso di dati. L'orologio scatta immediatamente dopo la conclusione della visualizzazione.
  - *Transparency Index*
    Quando nei dati viene incontrato questo indice, il relativo *pixel* è lasciato invariato.

- `Trailer`
  Questo blocco è lungo un byte, contiene 0x3B (;) e indica la fine di una stringa `GIF`.

#### 4.4.2.2 Graphic-Rendering Block

- `Image Descriptor`
  È definito come nella versione 87a.

- `Plain Text Extension`
  La struttura del blocco è definita nella tabella (4.10). Il blocco contiene testi e i parametri necessari per rappresentarli in modo grafico. I testi sono rappresentati con il codice `ASCII` a 7 bit e sono disegnati all'interno di celle a partire dall'alto a sinistra e procedendo riga dopo riga. Se le dimensioni non permettono di coprire l'area con un numero intero di celle, le frazioni rimanenti sono scartate. Questo blocco richiede la presenza della *Global Color Table* e può apparire dappertutto in concomitanza con una immagine. Le applicazioni che sanno interpretarlo sono rare.

#### 4.4.2.3 Special Purpose Block

- `Comment Extension`
  La struttura del blocco è definita nella tabella (4.11). Contiene informazioni testuali che non appartengono alla parte grafica. Serve per includere commenti, descrizioni, ecc. Il blocco può apparire ovunque.

| | |
|---|---|
| Extension Introducer (0x21) | byte |
| Plain Text Label(0x01) | byte |
| Block Size (12) | byte |
| Text Grid Left Position | Unsigned |
| Text Grid Top Position | Unsigned |
| Image Grid Width | Unsigned |
| Image Grid Height | Unsigned |
| Character Cell Width | byte |
| Character Cell Height | byte |
| Text Foreground Color Index | byte |
| Text Background Color Index | byte |
| Plain Text Data | sottoblocco |
| ... | sottoblocco |
| Block Terminator (0) | byte |

Tabella 4.10: Plain Text Extension

| | |
|---|---|
| Extension Introducer (0x21) | byte |
| Comment Label(0xFE) | byte |
| Comment Data | sottoblocco |
| ... | sottoblocco |
| Block Terminator (0) | byte |

Tabella 4.11: Comment Extension

- `Application Extension`
  Contiene informazioni specifiche per un'applicazione. La sua struttura è mostrata nella tabella (4.13). Questa estensione serve a descrivere con quale

| | |
|---|---|
| Extension Introducer (0x21) | byte |
| Application Extension Label(0xFF) | byte |
| Block Size (11) | byte |
| Application Identifier | 8 byte |
| Application Authentication Code | 3 byte |
| Application Data | Sottoblocco |
| Block Terminator | byte |

Tabella 4.12: Application Extension

applicazione l'immagine può essere visualizzata. Il o i sottoblocchi contengono eventuali informazioni necessarie all'applicazione per trattare l'immagine.

L'esempio più noto è l'estensione con *Application Identifier* NETSCAPE e *Authentication Code* 2.0 per produrre animazioni mediante ripetizione ciclica della sequenza di immagini contenute nel file. I sottoblocchi *Application Data* possono essere di due tipi:

  - `Looping Extension`
    con la struttura:

| | | |
|---|---|---|
| byte | 0x03 | Dimensione del blocco |
| byte | xxxxx001 | Codice del blocco |
| word | | Numero di ripetizioni (0 $\rightarrow$ infinito) |

Tabella 4.13: Looping Extension

  - `Buffering Extension`
    con la struttura

| | | |
|---|---|---|
| byte | 0x05 | Dimensione del blocco |
| byte | xxxxx010 | Codice del del blocco |
| longword | | Numero di *byte* da attendere prima di iniziare la visualizzazione |

Tabella 4.14: Buffering Extension

### 4.4.3 Esempio di file GIF

Il contenuto del file seguente rappresenta la medesima immagine vista negli esempi precedenti nel formato `GIF89a`.

```
FF 40 40 00 01 F0 00 40 00 7F 61 39 38 46 49 47
00 00 00 00 2C 00 00 00 00 00 04 F9 21 00 00 00
B4 9C A3 0F ED CB A9 8F 84 65 02 00 00 40 00 7F
CA EA A6 89 E6 96 48 E2 86 0F FB BC DE B3 8B DA
0C 0F FE F7 CE FA E7 8D F6 D7 4C F2 C7 0B EE B6
A7 4A 8D 09 F3 A6 CC 97 2A 4C 88 F1 A2 C4 87 0A
B2 E4 C7 8B 0E 0B F7 AE DC B7 6A CD 8A F5 AA D4
8E FD BA F4 E7 CB 8F 0D FB B6 EC D7 AB 4E 8C F9
00 00 3B 00 00 58 38 28 18 0F FF BE FC F7 EB CF
```

Interpretandolo in base a quanto visto in precedenza si ottiene l'informazione seguente.

| | | |
|---|---|---|
| 61 39 38 46 49 47 | Testata | GIF89a |
| 00 7F | FileScreen Width | 127 |
| 00 40 | FileHeight | 64 |
| F0 | Size of Global Color Table | 000 (bit)→ $2^{(0+1)} = 2$ |
| | Sort Flag | 0 (tavola non ordinata) |
| | Color resolution (depth) | 111=7+1=8 |
| | Global Color Table | 1 (esiste) |
| 01 | Background color index | 1 (nero) |
| 00 | Pixel Aspect Ratio | 0 (pixel quadrati) |
| FF 40 40 | Global color table | blu |
| 00 00 00 | | nero |
| 21 | Extension introducer | |
| F9 | Graphic Control Label | |
| 04 | Block Size | 4 byte |
| 00 00 00 00 | Disposal Method | |
| | User input flag | |
| | Transparent Color flag | |
| | Delay Time | |
| | Transparent Color Index | |
| 00 | Block Terminator | |
| 2C | Image Separator | |
| 00 00 | Image Left Position | 0 |
| 00 00 | Image Top Position | 0 |
| 00 7F | Image Width | 127 |
| 00 40 | Image Height | 64 |
| 00 | Size Local Color Table | 000 (bit) =0 |
| | Reserved | 00 (bit) |
| | Sort flag | 0 (bit) |
| | Interlace flag | 0 (non interlacciata) |
| | Local Color Table | 0 (non esiste) |
| 02 | Initial Code Size | 2 |
| 65 | Block Size | 101 |
| ... | Dati compressi | 101 byte |
| 00 | Block terminator | |
| 3B | Trailer | |

## 4.5 TIFF

La prima versione di questo formato è stata pubblicata nel 1986 dalla Aldus Corporation per poter rappresentare immagini rasterizzate catturate con gli *scanner*. Si tratta di un linguaggio che può essere usato anche da semplici *scanner* poichè il numero di campi di descrizione è piccolo. Il linguaggio è estendibile [1] e facilmente portabile da un sistema a un altro ed è adatto quando devono essere registrate informazioni colorimetriche. Gli spazi dei colori conosciuti sono:

- Grayscale
- Pseudocolor
- RGB
- YCbCr
- CMYK
- CIELab

Le principali caratteristiche sono mostrate nella tabella (4.15). Una descrizione

| | |
|---|---|
| Nome | Tag Image File Format |
| Tipo | bitmap |
| Colori | 1 ... 24 bit |
| Compressione | `RLE` , `LZW`, `CCITT`, `Group 3,4`, `JPEG`, nessuna |
| Dimensione massima | $2^{32} - 1$ |
| Più immagini per file | si |
| Formato numerico | *big* o *little endian* |
| Creatore | Aldus Corporation |

Tabella 4.15: Pricipali caratteristiche del formato `TIFF`

esaustiva del formato è contenuta nel documento $TIFF$ , *Revision* 6.0 del 3 giugno 1992 pubblicato da *Aldus Developers Desk*.

[1] Anche estensioni private sono possibili. È infatti possibile riservare presso il *TIFF Administrator* nuovi *tag* oppure valori per *tag* esistenti ad uso privato.

### 4.5.1 Struttura di un file `TIFF`

- File Header
  La testata ha la struttura mostrata nella tabella (4.16).

| Byte | | |
|---|---|---|
| 0-1 | ordine dei *byte* | II→ *little endian* |
| | | MM→ *big endian* |
| 2-3 | identificatore del formato `TIFF` | $42_{10}$ |
| 4-7 | *offset* al primo indirizzario | |

Tabella 4.16: Formato della testata di un file `TIFF`

- Image File Directory-IFD
  Il formato dell'$IFD$ è mostrato nella tabella (4.17).

| Indirizzo (in byte) | |
|---|---|
| $i$ | contatore ($n$) di elementi nella $IFD$ |
| $i+2$ | Elemento 0 |
| $i+14$ | Elemento 1 |
| $\cdots$ | |
| $i+2+n*12$ | *offset* al prossimo $IFD$ |

Tabella 4.17: Formato di una $IFD$ di un file `TIFF`

Ogni elemento di una $IFD$ ha la struttura mostrata nella tabella (4.18).

| Byte | |
|---|---|
| 0-1 | etichetta che identifica il campo |
| 2-3 | tipo di dato |
| 4-7 | contatore di valori del tipo indicato |
| 8-11 | valore/$offset$ |

Tabella 4.18: Formato di un elemento di una $IFD$ di un file `TIFF`

I tipi di dati conosciuti sono quelli elencati nella tabella (4.19). Il campo valore/$offset$ può contenere sia il o i valori dei dati per un totale fino a quattro byte oppure l'*offset* ai dati, se i dati sono in numero maggiore a quattro byte.

- *Valori*
- $\cdots$

| Codice | Tipo di dato | |
|---|---|---|
| 1 | `BYTE` | |
| 2 | `ASCII` | |
| 3 | `SHORT` | |
| 4 | `LONG` | |
| 5 | `RATIONAL` | $2 \times$ LONG |
| 6 | `SBYTE` | signed byte |
| 7 | `UNDEFINED` | |
| 8 | `SSHORT` | signed short |
| 9 | `SLONG` | signed long |
| 10 | `SRATIONAL` | signed rational |
| 11 | `FLOAT` | formato IEEE 32 bit |
| 12 | `DOUBLE` | formato IEEE 64 bit |

Tabella 4.19: Tipi di dato noti a `TIFF`

La sequenza "$IFD$ - valori" può essere ripetuta più volte.

Le tabelle (4.20), (4.21) e (4.22) contengono l'elenco delle etichette definite, ordinate per codice.

| Nome etichetta | Valore | Codice | Tipo | Numero di valori |
|---|---|---|---|---|
| NewSubfileType | | FE | LONG | 1 |
| SubfileType | | FF | SHORT | 1 |
| ImageWidth | | 100 | SHORT or LONG | 1 |
| ImageLength | | 101 | SHORT or LONG | 1 |
| BitsPerSample | | 102 | SHORT | SamplesPerPixel |
| Compression | | 103 | SHORT | 1 |
| Uncompressed | 1 | | | |
| CCITT 1D | 2 | | | |
| Group 3 Fax | 3 | | | |
| Group 4 Fax | 4 | | | |
| LZW | 5 | | | |
| JPEG | 6 | | | |
| PackBits | 32773 | | | |
| PhotometricInterpretation | | 106 | SHORT | 1 |
| WhiteIsZero | 0 | | | |
| BlackIsZero | 1 | | | |
| RGB | 2 | | | |
| RGB Palette | 3 | | | |
| Transparency mask | 4 | | | |
| CMYK | 5 | | | |
| YCbCr | 6 | | | |
| CIELab | 8 | | | |
| Threshholding | | 107 | SHORT | 1 |
| CellWidth | | 108 | SHORT | 1 |
| CellLength | | 109 | SHORT | 1 |
| FillOrder | | 10A | SHORT | 1 |
| DocumentName | | 10D | ASCII | |
| ImageDescription | | 10E | ASCII | |
| Make | | 10F | ASCII | |
| Model | | 110 | ASCII | |
| StripOffsets | | 111 | SHORT or LONG | StripsPerImage |
| Orientation | | 112 | SHORT | 1 |
| SamplesPerPixel | | 115 | SHORT | 1 |
| RowsPerStrip | | 116 | SHORT or LONG | 1 |
| StripByteCounts | | 117 | LONG or SHORT | StripsPerImage |
| MinSampleValue | | 118 | SHORT | SamplesPerPixel |
| MaxSampleValue | | 119 | SHORT | SamplesPerPixel |
| XResolution | | 11A | RATIONAL | 1 |
| YResolution | | 11B | RATIONAL | 1 |
| PlanarConfiguration | | 11C | SHORT | 1 |
| PageName | | 11D | ASCII | |

Tabella 4.20: Etichette di TIFF ordinate per codice

| Nome etichetta | Codice | Tipo | Numero di valori |
|---|---|---|---|
| XPosition | 11E | RATIONAL | |
| YPosition | 11F | RATIONAL | |
| FreeOffsets | 120 | LONG | |
| FreeByteCounts | 121 | LONG | |
| GrayResponseUnit | 122 | SHORT | 1 |
| GrayResponseCurve | 123 | SHORT | $2^{\text{BitsPerSample}}$ |
| T4Options | 124 | LONG | 1 |
| T6Options | 125 | LONG | 1 |
| ResolutionUnit | 128 | SHORT | 1 |
| PageNumber | 129 | SHORT | 2 |
| TransferFunction | 12D | SHORT | $1\vert\ _{\text{SamplesPerPixel}*2^{\text{BitsPerSample}}}$ |
| Software | 131 | ASCII | |
| DateTime | 132 | ASCII | 20 |
| Artist | 13B | ASCII | |
| HostComputer | 13C | ASCII | |
| Predictor | 13D | SHORT | 1 |
| WhitePoint | 13E | RATIONAL | 2 |
| PrimaryChromaticities | 13F | RATIONAL | 6 |
| ColorMap | 140 | SHORT | $3*2^{\text{BitsPerSample}}$ |
| HalftoneHints | 141 | SHORT | 2 |
| TileWidth | 142 | SHORT or LONG | 1 |
| TileLength | 143 | SHORT or LONG | 1 |
| TileOffsets | 144 | LONG | TilesPerImage |
| TileByteCounts | 145 | SHORT or LONG | TilesPerImage |
| InkSet | 14C | SHORT | 1 |
| InkNames | 14D | ASCII | numero totale caratteri `ink` |
| NumberOfInks | 14E | SHORT | 1 |
| DotRange | 150 | BYTE or SHORT | 2, oppure 2*NumberOfInks |
| TargetPrinter | 151 | ASCII | |
| ExtraSamples | 152 | BYTE | no. compon. extra per *pixel* |
| SampleFormat | 153 | SHORT | SamplesPerPixel |
| SMinSampleValue | 154 | Any | SamplesPerPixel |
| SMaxSampleValue | 155 | Any | SamplesPerPixel |
| TransferRange | 156 | SHORT | 6 |
| JPEGProc | 200 | SHORT | 1 |
| JPEGInterchangeFormat | 201 | LONG | 1 |
| JPEGInterchangeFormatLngth | 202 | LONG | 1 |
| JPEGRestartInterval | 203 | SHORT | 1 |

Tabella 4.21: Etichette di TIFF ordinate per codice (continuazione)

| Nome etichetta | Codice | Tipo | Numero di valori |
|---|---|---|---|
| JPEGLosslessPredictors | 205 | `SHORT` | SamplesPerPixel |
| JPEGPointTransforms | 206 | `SHORT` | SamplesPerPixel |
| JPEGQTables | 207 | `LONG` | SamplesPerPixel |
| JPEGDCTables | 208 | `LONG` | SamplesPerPixel |
| JPEGACTables | 209 | `LONG` | SamplesPerPixel |
| YCbCrCoefficients | 211 | `RATIONAL` | 3 |
| YCbCrSubSampling | 212 | `SHORT` | 2 |
| YCbCrPositioning | 213 | `SHORT` | 1 |
| ReferenceBlackWhite | 214 | `LONG` | 2*`SamplesPerPixel` |
| Copyright | 8298 | `ASCII` | |

Tabella 4.22: Etichette di `TIFF` ordinate per codice (continuazione)

### 4.5.2 Alcuni esempi di descrizioni

#### 4.5.2.1 Immagini bitonali (bilevel)

Le immagine più semplici da descrivere sono quelle bitonali (bianco e nero). Il formato `TIFF` usa in questo caso le etichette descritte nella tabella (4.23). Alcune etichette ammettono un valore di difetto e perciò possono anche mancare nella descrizione.

#### 4.5.2.2 Immagini a toni di grigio (grayscale)

Le etichette richieste per la rappresentazione sono tutte quelle usate nelle immagini bitonali più l'etichetta `BitPerSample`, che porta il codice 0x102 e rappresenta il numero di bit per ogni componente. I bit possono essere 4 o 8. Il tipo di dato è `SHORT`. I valori del campo `Compression` possono essere solo 1 o 32773. Occorre osservare che nel caso di immagini a tono continuo la compressione sovente non è efficace e può essere omessa.

#### 4.5.2.3 Immagini a colori indicizzati (pseudocolor)

Il formato, nella sua struttura è simile a quello delle immagini a toni di grigio, ma l'etichetta `PhotometricInterpretation` contiene il valore 3. Per ogni *pixel* viene memorizzata una sola componente che rappresenta l'indice alla mappa dei colori RGB. La mappa viene individuata attraverso l'etichetta `ColorMap`, con codice 0x140, i cui dati sono di tipo `SHORT` e il contatore contiene il valore $3*(2^{BitsPerSample})$. Nella mappa tuttti i valori di ognuna delle tre componenti RGB sono raggruppate assieme.

#### 4.5.2.4 Immagini a colori veri RGB

L'etichetta `BitPerSample` contiene 8 8 8 e l'etichetta `PhotometricInterpretation` contiene 2. In questo formato si aggiunge l'etichetta `SamplePerPixel` con codice 0x115 che contiene il numero di componenti per *pixel* di tipo `SHORT`.

Negli esempi di formato precedenti, i *pixel* delle immagine sono memorizzate riga per riga o colonma per colonna cioè in *strip*. La descrizione a *strip* non è conveniente nel caso di immagini ad alta risoluzione, poichè i meccanismi di compressione sono poco efficienti. In questi casi è opprtuno descrivere l'immagine come insieme di rettangolari della stessa dimensione (*tiled images*). Le dimensioni sono fissate dall'utente e quindi alcune non sono completamente occupate da *pixel*, ma contengono caratteri di *padding*. La tabella (4.24) contiene le etichette necessarie per descrivere un'immagine composta di rettangoli.

| Campo | Contenuto | Interpretazione |
|---|---|---|
| Etichetta | 0x106 | PhotometricInterpretation |
| Tipo di dato | `SHORT` | |
| Valori | 0 | bianco |
| | 1 | nero |
| Etichetta | 0x103 | Compressione |
| Tipo di dato | `SHORT` | |
| Valori | 1 | senza compressione |
| | 2 | compressione `CCITT ID` |
| | 32773 | Compresssione RLE dei bit |
| Etichetta | 0x101 | Lunghezza dell'immagine |
| Tipo di dato | `SHORT` o `LONG` | |
| Valori | | Numero di righe (*scanlines*) |
| Etichetta | 0x100 | Larghezza dell'immagine |
| Tipo di dato | `SHORT` o `LONG` | |
| Valori | | Numero di colonne |
| Etichetta | 0x128 | Unità di risoluzione |
| Tipo di dato | `SHORT` | |
| Valori | 1 | Nessuna unità |
| | 2 | inch (default) |
| | 3 | cm |
| Etichetta | 0x11A | Risoluzione in direzione X (larghezza) |
| Tipo di dato | `RATIONAL` | |
| Valori | | *pixel* per unità di risoluzione |
| Etichetta | 0x11B | Risoluzione in direzione Y (lunghezza) |
| Tipo di dato | `RATIONAL` | |
| Valori | | *pixel* per unità di risoluzione |
| Etichetta | 0x116 | Righe per blocco di dati |
| Tipo | `SHORT` o `LONG` | |
| Valore | | Righe per *strip* |
| Etichetta | 0x111 | *Offset* al blocco di dati |
| Tipo | `SHORT` o `LONG` | |
| Valore | | |
| Etichetta | 0x117 | Contatore di byte |
| Tipo | `SHORT` o `LONG` | |
| Valore | | Numero di byte dopo eventuale compressione |

Tabella 4.23: Etichette usate per descrivere immagini bitonali

| Campo | Contenuto | Interpretazione |
|---|---|---|
| Etichetta | 0x142 | Tilewidth |
| Tipo di dato | `SHORT` o `LONG` | |
| Valore | | larghezza in *pixel* |
| Etichetta | 0x143 | TileLenght |
| Tipo di dato | `SHORT` o `LONG` | |
| Valore | | altezza in *pixel* |
| Etichetta | 0x144 | TileOffsets |
| Tipo di dato | `LONG` | |
| Valore | | $Offset$ rispetto all'inizio del file `TIFF` |
| Etichetta | 0x145 | TileByteCounts |
| Tipo di dato | `SHORT` or `LONG` | |
| Valore | | numero di byte compressi per $Tile$ |

Tabella 4.24: Etichette usate per descrivere immagini *tiled*

### 4.5.3 Esempio di file TIFF

Il contenuto del file seguente rappresenta un file `TIFF` con la medesima immagine monocolore vista in precedenza.

```
FF FF FF FF FF FF FF FF 14 00 00 00 2A 00 4D 4D
7F 00 01 00 00 00 03 00 00 01 0F 00 00 80 08 00
02 01 00 00 40 00 01 00 00 00 03 00 01 01 00 00
00 00 03 00 03 01 00 00 01 00 01 00 00 00 03 00
03 00 01 00 00 00 03 00 06 01 00 00 04 00 01 00
0D 01 00 00 02 00 01 00 00 00 03 00 0A 01 00 00
00 00 04 00 11 01 CE 00 00 00 07 00 00 00 02 00
01 00 01 00 00 00 03 00 12 01 08 00 00 00 01 00
16 01 00 00 01 00 01 00 00 00 03 00 15 01 00 00
00 00 04 00 17 01 00 00 40 00 01 00 00 00 03 00
01 00 01 00 00 00 03 00 1C 01 0B 00 00 00 01 00
40 01 D6 00 00 00 3F 00 00 00 02 00 31 01 00 00
2E 78 00 00 00 00 16 01 00 00 06 00 00 00 03 00
4D 65 67 61 6D 49 29 23 28 40 00 00 66 66 69 74
30 2F 34 30 20 30 2E 33 2E 35 20 6B 63 69 67 61
2F 3A 70 74 74 68 20 36 31 3A 51 20 31 30 2F 31
6B 63 69 67 61 6D 65 67 61 6D 69 2E 77 77 77 2F
FF FF 00 00 40 40 00 00 40 40 00 00 67 72 6F 2E
                                        00 00
```

Analizzando il suo contenuto, si ottiene la tabella seguente.

| | | |
|---|---|---|
| 4D 4D | MM | Ordine dei byte:*big endian* |
| 2A 00 | 42 | Identificatore di TIFF |
| 14 00 00 00 | 20 | offset al primo indirizzario |
| 80 08 00 FF FF FF FF FF FF FF FF 00 | | |
| 0F 00 | 15 | Numero elementi nell'indirizzario |
| 00 00\|7F 00\|01 00 00 00\|03 00\|00 01 | 127 | Image Width |
| 00 00\|40 00\|01 00 00 00\|03 00\|01 01 | 64 | ImageLength |
| 00 00\|01 00\|01 00 00 00\|03 00\|02 01 | 1 | BitsPerSample |
| 00 00\|04 00\|01 00 00 00\|03 00\|03 01 | 3 | Compressione: gruppo 4 FAX |
| 00 00\|03 00\|01 00 00 00\|03 00\|06 01 | paletta RGB | PhotometricInterpretation |
| 00 00\|02 00\|01 00 00 00\|03 00\|0A 01 | 2 | FillOrder [a] |
| CE 00 00 00\|07 00 00 00\|02 00\|0D 01 | 206 (offset) | DocumentName |
| 08 00 00 00\|01 00 00 00\|04 00\|11 01 | 8 | StripOffsets [b] |
| 00 00\|01 00\|01 00 00 00\|03 00\|12 01 | 1 | Orientation [c] |
| 00 00\|01 00\|01 00 00 00\|03 00\|15 01 | 1 | SamplesPerPixel [d] |
| 00 00\|40 00\|01 00 00 00\|03 00\|16 01 | 64 | RowsPerStrip |
| 0B 00 00 00\|01 00 00 00\|04 00\|17 01 | 11 | StripByteCounts [e] |
| 00 00\|01 00\|01 00 00 00\|03 00\|1C 01 | 1 | PlanarConfinuration [f] |
| D6 00 00 00\|3F 00 00 00\|02 00\|31 01 | 241 (offset) | Software |
| 16 01 00 00\|06 00 00 00\|03 00\|40 01 | 278 (offset) | Colormap [g] |
| 00 00 00 00 | 0 | offset al prossimo IFD |
| 66 66 69 74 2E 78 00 00 | x.tiff | |
| 67 61 4D 65 67 61 6D 49 29 23 28 40 34 30 20 30 2E 33 2E 35 20 6B 63 69 20 36 31 3A 51 20 31 30 2F 31 30 2F 69 2E 77 77 77 2F 2F 3A 70 74 74 68 6F 2E 6B 63 69 67 61 6D 65 67 61 6D 67 72 00 00 | @(#)ImageMagick 5.3.0 04 0101 Q:16 / http:/www.i/ magemagick.org | |
| 00 00 FF FF 00 00 40 40 00 00 40 40 | blu | paletta |

[a] In ogni *byte* i *pixel* sono descritti in modo che a colonne con numeri bassi corrispondono *bit* meno significativi

[b] Rispetto all'inizio del file

[c] La prima riga descritta è quella in alto dell'immagine, la prima colonna quella a sinistra

[d] Numero di componenti per *pixel*

[e] Numero di *byte* dello strip dopo la compressione

[f] Metodo di memorizzazione delle componenti del *pixel*. 1→ componenti continue

[g] $N = 3 * 2^{BitsPerSsmple}$

## 4.6 Portable Network Graphics (PNG)

PNG (da pronunciare *ping*) viene usato per descrivere un formato estendibile a compressione esatta per immagini. Il formato è stato creato per sostituire il formato GIF e può in certi casi sostituire anche il formato TIFF. Esso permette di memorizzare immagini a toni di grigio, immgini a colori indicizzati, immagini a colori veri e permette di fare uso del canale *alpha* per specificare la trasparenza. Il formato è stato creato in particolare per applicazioni funzionanti su *World Wide Web*. È possibile infatti visualizzare un'immagine progressivamente, prima grossolanamente, e poi raffinandola gradualmente con i dettagli. Il formato permette inoltre di detettare l'integrità dei file e di mescolare informazioni testuali. PNG non usa algoritmi proprietari.
I seguenti tipi di immagini non realizzabili in GIF possono essere creati in PNG:

- Immagini a colori veri fino a 48 bit per *pixel*,
- Immagini a toni di grigio fino a 16 bit per *pixel*,
- Immagini con maschera di trasparenza,
- Immagini con informazione *gamma* che permettono la visualizzazione automatica con il rapporto luminosità/contrasto corretto, indipendente dall'*hardware* che ha generato l'immagine,
- Immagini che permettono di detettare errori nel file che le contengono,
- Immagini a rappresentazione progressiva.

### 4.6.1 Interlacciamento

Per realizzare una visualizzazione progressiva dell'immagine, PNG può usare la tecnica dell'interlacciamento. Il metodo scelto risale a *Adam M. Costello* ed è noto come metodo Adam 7. L'immagine viene ricostruita mediante una sequenza di sette passi. Durante ogni passo si visualizza un sottoinsieme di *pixel* appartenenti a quadrati di $8 \times 8$ *pixel* che hanno origine in alto a sinistra dell'immagine secondo lo schema mostrato nella figura (4.1). Ad ogni passo i *pixel* selezionati sono visualizzati da sinistra a destra lungo una linea di scansione e le linee selezionate sono visualizzate dall'alto verso il basso. Il passo 2 contiene per esempio i *pixel* $4, 12, 20, \cdots$. delle linee di scansione $0, 8, 16, \cdots$. (la numerazione inizia con 0,0 in alto a sinsistra). L'ultimo passo contiene le linee intere di scansione $1, 3, 5, \cdots$.
Se l'immagine contiene meno di cinque linee di scansione, alcuni passi sono completamente vuoti. Codificatori e decodificatori devono saper trattare correttamente anche questo caso particolare.

### 4.6.2 Correzione gamma

PNG è in grado di specificare la caratteristica *gamma* di un'immagine rispetto alla scena originale.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 6 | 4 | 6 | 2 | 6 | 4 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 5 | 6 | 5 | 6 | 5 | 6 | 5 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 3 | 6 | 4 | 6 | 3 | 6 | 4 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 5 | 6 | 5 | 6 | 5 | 6 | 5 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

Figura 4.1: PNG - schema d'interlacciamento

### 4.6.3 Stringhe di testo

Stringhe di testo possono essere associate alle immagini. L'insieme di caratteri consigliati è il *Latin-1 - ISO 8859*. Altri insiemi di caratteri sono sconsigliati essendo dipendenti dalla piattaforma usata.

### 4.6.4 Struttura del file

#### 4.6.4.1 Firma e componenti

I primi 8 *byte* di un file `PNG` contengono la firma (*signature*) decimale seguente:

```
137 80 78 71 13 10 26 10
```

La sequenza indica che il resto del file contiene una singola immagine `PNG` formata da componenti (*chunk*). La prima è denominata `IHDR`, l'ultima `IEND`.

#### 4.6.4.2 Struttura delle componenti

Ogni componente è formata dalle quattro parti seguenti.

- Lunghezza
  È un campo di quattro *byte* che contiene il numero di *byte* della componente. Il contatore indica esclusivamente il numero di *byte* di dati e non comprende il numero di *byte* di se stesso, quello dei *byte* necessari per indicare il tipo e quelli usati dal CRC.

- Tipo
  Questa parte è lunga quattro *byte*. La stringa può contenere i caratteri con codice ASCII degli intervalli 65-90 (A-Z) e 97-122 (a-z). Codificatori e decodificatori devono trattare le stringe come numeri binari a lunghezza fissa.

- Dati
  Questa parte può anche essere vuota.

- CRC
  È un campo di quattro *byte* che contiene il *Cyclic Redundancy Check* dei *byte* precedenti, compresi quelli che specificano il tipo, ma non quelli che contengono la lunghezza. Questo parte è sempre presente.

#### 4.6.4.3 Convenzione sui nomi

I nomi delle componenti sono definiti in modo che i decodificatori possano ricostruire alcune loro caratteristiche anche senza conoscere la definizione esatta del tipo. Il meccanismo usato garantisce un'estensione flessibile e sicura del formato `PNG` e permette ai decodificatori di decidere come comportarsi quando incontrano un nuovo tipo. Esso consiste mell'usare il bit 5 (corrispondente al valore 32) di ognuno dei quattro *byte* che definiscono il tipo. [2] I quattro bit sono da interpretare secondo le regole seguenti:

- bit ausiliario (*ancillary* bit): bit 5 del primo *byte*
  - 1 → componente ausiliaria
    Se il bit è uno, la componente non è strettamente necessaria per poter decodificare in modo sensato il file. Se il decodificatore incontra una componente di tipo sconosciuto con questo bit posto a uno, la può ignorare e procedere ciò nonostante all'interpretazione.
  - 0 → componente critica
    Se il decodificatore incontra un tipo sconosciuto dove questo bit posto a 0, deve segnalare all'utente che il file contiene informazioni che non possono essere interpretate in modo sicuro. Il tipo `IHDR` è un esempio in questo senso.
- bit privato: bit 5 del secondo *byte*
  - 1 → componente privata
    Tipi privati non sono registrati in nessuna lista di `PNG` pubblica.
  - 0 → componente pubblica
    Tipi pubblici sono registrati in liste di `PNG` rese pubbliche.

  Questa suddivisione ha unicamente scopi amministrativi e serve a separare correttamente tipi pubblici da tipi privati.
- bit riservato: bit 5 del terzo *byte*
  - 0: nelle versioni attuali di `PNG`

[1] Se questo bit è zero, il carattere `ASCII` corrispondente al *byte* è di regola una lettera maiuscola; in caso contrario la lettera è minuscola. I decodificatori non devono basarsi però su questo test poichè il codice locale associato al *byte* potrebbe essere diverso da quello ASCII, ma devono testare direttamente il valore numerico del bit.

  - 1: riservato per possibili future estensioni
    Se questo bit è uno, i decodificatori attuali trattano le componenti come sconosciute.

- bit copia sicura (*save-to-copy* bit): bit 5 del quarto *byte*.
  Questo bit non è interessante per i decodificatori, ma per gli editori di immagini.

  - 0 → copia non sicura
    Se il programma ha fatto modifiche a componenti critiche del file, eventuali componenti non riconosciute con questo bit posto a zero non possono essere semplicemente ricopiate sull'*output*.
  - 1 → copia sicura
    La componente può essere ricopiata sia nel caso che il programma riconosce il tipo o meno.

#### 4.6.4.4 Componenti critiche

Tutti i decodificatori devono essere in grado di decodificare e visualizzare le componenti critiche standard elencate in seguito.

**IHDR - Image header**
Deve apparire all'inizio e contiene le informazioni mostrate nella tabella (4.25).

| | numero di byte |
|---|---|
| Larghezza | 4 |
| Altezza | 4 |
| Profondità | 1 |
| Tipo di colore | 1 |
| Metodo di compressione | 1 |
| Metodo di filtraggio | 1 |
| Metodo di interlacciamento | 1 |

Tabella 4.25: IHDR - Image header

La *profondità* indica il numero di bit usati per rappresentare ogni campione di colore oppure l'indice dei colori. I valori validi sono 1,2,4,8 e 16.

Il *tipo di colore* può essere la somma dei valori 1,2 e 4. I numeri ammessi e la loro interpretazione sono elencati nella tabella (4.26).

Attualmente l'unico valore per il *metodo di compressione* ammesso è zero. Esso corrisponde a una compressione con l'algortimo *deflate* (vedi RFC 1951) con *sliding window* di 32k.

| tipo | Profondità | Interpretazione |
|---|---|---|
| 0 | 1,2,4,8,16 | Ogni *pixel* è un tono di grigio |
| 2 | 8,16 | Ogni *pixel* è una terna $R$,$G$,$B$ |
| 3 | 1,2,4,8 | Ogni *pixel* è un indice della paletta dei colori |
| 4 | 8,16 | Ogni *pixel* è un tono di grigio seguito da un valore *alfa* |
| 6 | 8,16 | Ogni *pixel* è una terna $R$,$G$,$B$ seguita da un valore *alfa* |

Tabella 4.26: Tipi di colori ammessi

L'unico valore ammesso per il *metodo di filtraggio* è zero. Esso indica che il filtraggio applicato all'immagine prima della compressione è quello adattivo con cinque tipi di filtri di base.

I valori ammessi per il *metodo d'interlacciamento* sono due: 0 quando non esiste interlacciamento e 1 quando si applica l'interlacciamento Adam7.

**PLTE - Palette entries**
Questa componente contiene elementi formati da tre *byte* ciascuno e che rappresentano terne $R$,$G$,$B$. Il primo elemento corrisponde all'indice 0, il secondo all'indice 1, ecc. Il numero di elementi non deve superare il valore specificato mediante la profondità. Una paletta più corta del valore massimo è però permessa. In questo caso ogni indice a cui non corrisponde nessun elemento nella paletta è considerato un errore. Il numero di elementi è determinato dalla lunghezza della componente. Una lunghezza non divisibile per tre costituisce un errore. Questa componente è obbligatoria per il tipo di colore 3, ma può anche apparire per il tipi 2 e 6. Se è presente può essere usata per suggerire una quantizzazione dei colori veri, quando il sistema di visualizzazione non è in grado di visualizzarli. La componente non deve apparire invece per i tipi 0 e 4.
Quando appare, questa componente deve precedere la prima componente di tipo `IDAT`. La componente `PLTE` deve apparire una sola volta.

**IDAT - Image Data**
Questa componente contiene i dati risultanti dai processi di scansione, filtraggio e compressione. Essa può essere presente più volte nel file. L'immagine viene ottenuta in questo caso concatenando tutte le componenti `IDAT` presenti. Il codificatore può suddividere a piacimento i dati in più componenti `IDAT`. Tipicamente i codificatori suddividono i dati in modo da poter operare con una quantità di memoria fissa corrispondente al loro *buffer*.

**IEND**
Questa componente deve essere l'ultima del file. La relativa parte dati è vuota.

#### 4.6.4.5 Componenti ausiliarie

Sono componenti opzionali che possono essere omesse dai codificatori o ignorate dai decodificatori. Valgono però le seguenti raccomandazioni: i codificatori devono generare quelle standard ogni volta che i dati necessari sono disponibili mentre i decodificatori devono analogamente interpretare queste componenti quando è possibile e ragionevole farlo.

**bKGD - Background color**
Specifica il colore di difetto dello sfondo. Per il colore tipo 3, questa componente contiene un *byte*, per i tipi 0 e 4 due *byte*. Per i tipi 2 e 6 la componente contiene 6 *byte* (due *byte* per ogni componente $R$,$G$,$B$).

Quando è presente la componente deve precedere la prima componente `IDAT`, ma deve seguire la componente `PLTE`.

**cHRM - Primary chromaticities and white point**
Per poter specificare i colori in modo preciso, il modello $RGB$ non è adatto. Si deve perciò far ricorso al modello $XYZ$ da cui è derivato il modello CIE $Yxy$ definito nel 1931. Questo modello fa uso della *luminanza* $Y$ , e della *crominanza* $x$,$y$.
La componente `cHRM` contiene l'informazione necessaria per ricostruire i valori $X$,$Y$ e $Z$ a partire dai valori $R$,$G$ e $B$. La tabella (4.27) contiene l'informazione fornita, formata dalle quattro cromaticità seguenti:

- **cromaticità del punto bianco**
  Il punto bianco di un *monitor* è la cromaticità $x, y$ del colore bianco nominale, cioè quello prodotto con $R = G = B =$ massimo.
- **cromaticità dei fosfori**
  Per ognuno dei tre fosfori viene specificata la cromaticità $x, y$.

| Cromaticità | byte |
|---|---|
| $x$ punto bianco | 4 |
| $y$ punto bianco | 4 |
| $x$ rosso | 4 |
| $y$ rosso | 4 |
| $x$ verde | 4 |
| $y$ verde | 4 |
| $x$ blu | 4 |
| $y$ blu | 4 |

Tabella 4.27: Componente `cHRM`

Ogni valore di $x$ e $y$ è rappresentato con un numero intero senza segno che corrisponde al valore di cromaticità moltiplicato per 10000. Un valore di 31270 corrisponde

ad esempio a 0.3127.

Se il codificatore non conosce i valori da inserire nella tabella (4.27), non deve scrivere la componente `cHRM`. L'assensa della componente indica che i colori primari usati sono dipendenti dal *device*.

Se questa componente è presente, essa deve precedere la prima componente `IDAT` e deve anche precedere la componente `PLTE`, se questa è presente.

**gAMA - Image gamma**
Questa componente specifica il valore di *gamma* moltiplicato per 10000 in un campo di lunghezza di 4 *byte*. Se il valore non è noto, questa componente non deve essere specificata.

Quando la componente è presente, essa deve precedere la prima componente `IDAT` e deve anche precedere la componente `PLTE` se esiste.

**hIST - Image histogram**
Questa componente contiene la frequenza approssimata d'uso di ogni colore della paletta e perciò essa può apparire solo nel caso in cui appare la componente `PLTE`. Se il visualizzatore non è in grado di mettere a disposizione tutti i colori specificati nella paletta, questo istogramma può aiutare a decidere come scegliere un sottoinsieme di colori visualizzabili.

Questa componente è formata da un insieme di numeri interi senza segno di 2 *byte* ciascuno. L'insieme deve essere grande esattamente come il numero di colori contenuti nella paletta. Ogni valore è proporzionale alla frazione di *pixel* contenuti nell'immagine con un dato indice.

Quando la componente è presente, essa deve seguire la componente `PLTE` e precedere la prima componente `IDAT`.

**pHYs - Physical pixel dimensions**
La componente specifica la dimensione dei *pixel* oppure il rapporto fra le dimensioni del *display*. La struttura dei dati contenuti è descritta nella tabella (4.28) Quando

| | | |
|---|---|---|
| *Pixel* per unità, asse $x$ | 4 byte (intero senza segno) | |
| *Pixel* per unità, asse $y$ | 4 byte (intero senza segno) | |
| Specificatore di unità | 1 byte | 0 → unità sconosciuta<br>1 → unità in metri |

Tabella 4.28: Componente `pHYs`

l'unità è zero, la componente definisce il rapporto fra le dimensioni del *display*.

Se la componente non è presente, si assume che i *pixel* sono quadrati e che la loro dimensione è sconosciuta.

Quando la componente è presente, essa deve precedere la prima componente `IDAT`.

**tEXt - Textual data**
Questa componente contiene un'informazione testuale. La struttura è descritta nella tabella (4.29). Parola-chiave e testo non possono contenere il carattere `null`. Questa

| | | |
|---|---|---|
| Parola-chiave | 1-79 byte | Testo (non è necessario che termini con `null`) |
| Separatore | 1 byte | il caratterie `null` |
| Testo | n byte | |

Tabella 4.29: Componente `tEXt`

componente può apparire più volte nel file. Le parole-chiave predefinite sono contenute nella tabella (4.30). Altre parole-chiave d'interesse generale possono essere

| Parola-chiave | Significato |
|---|---|
| Titel | Corto titolo di un'immagine |
| Author | Nome del creatore dell'immagine |
| Description | Descrizione dell'immagine |
| Copyright | Informazione sui diritti di autore |
| Creation Time | Data della creazione dell'immagine originale[1] |
| Software | Software usata per creare l'immagine |
| Disclaimer | Denuncia legale |
| Warning | Avvertimento sulla natura del contenuto |
| Source | Strumento usato per la creazione dell'immagine |
| Comment | |

Tabella 4.30: Parole-chiave predefinite della componente `tEXt`

inventate e registrate presso i curatori delle specifiche di `PNG`. L'uso di parole-chiave private è però anche permesso. Le parole-chiave e i testi possono contenere solo caratteri stampabili e spazi e sono interpretati secondo l'insieme di caratteri `ISO 8859-1 (Latin-1)`. Il separatore di riga deve essere il codice `ASCII` 10. Le parole-chiave non possono nè iniziare nè terminare con spazi e non possono contenere spazi multipli. Il carattere spazio (`ASCII 160`) non è ammesso all'interno delle parole chiave.

**tIME - Image last-modification time**
Questa componente ha la struttura mostrata nella tabella (4.31). Viene raccoman-

[1]Si suggerisce di usare il formato definito in RFC 1123.

| | | |
|---|---|---|
| Anno | 2 byte | a quattro cifre |
| Mese | 1 byte | 1-12 |
| Giorno | byte | 1-31 |
| Ora | byte | 0-23 |
| Minuto | byte | 0-59 |
| Secondo | byte | 0-60 |

Tabella 4.31: Struttura e interpretazione della componente `tIME`

dato di specificare la componente in `UTC` e di non usare il tempo locale. La data deve essere aggiornata solo quando l'immagine viene modificata.

**tRNS - Transparency**
Questa componente descrive la trasparenza di un'immagine sia quando si usa la paletta dei colori, sia nel caso di immagini a toni di grigio.
Per il colore tipo 3 la componente contiene una lista di *byte* con i valori *alfa* corrispondenti alla sequenza degli indici nella paletta. Il valore zero significa piena trasparenza, il valore 255 completa opacità. Se questa lista è più corta di quella degli indici nella paletta, i rimanenti colori sono considerati completamente opachi.

Per il tipo 0 (toni di grigio), la componente contiene un solo valore lungo due *byte*. I *pixel* il cui valore coincide con quello contenuto in questa componente sono da considerare completamente trasparenti, tutti gli altri completamente opachi.

Per il tipo 2 (*truecolor*), la componente contiene un singolo valore `RGB` lungo tre volte due *byte*. I *pixel* il cui valore coincide con quello contenuto in questa componente sono da considerare completamente trasparenti, tutti gli altri completamente opachi.

La compontente `tRSN` non è ammessa per i tipi 4 e 6.

### 4.6.5 Esempio di file PNG

Il contenuto del file seguente rappresenta un file `PNG`.

```
52 44 48 49 0D 00 00 00 0A 1A 0A 0D 47 4E 50 89
3A 5F 09 00 00 00 03 01 40 00 00 00 7F 00 00 00
92 10 9B 0A FF 40 40 45 54 4C 50 03 00 00 00 28
60 A3 05 18 60 63 DA 78 54 41 44 49 12 00 00 00
00 00 0B C3 F7 03 01 00 40 04 00 00 F2 02 8C 14
                  82 60 42 AE 44 4E 45 49 00 00
```

Analizzando il suo contenuto, si ottiene la tabella seguente.

| | | |
|---|---|---|
| 0A 1A 0A 0D 47 4E 50 89 | <HTJ>PNG<CR><LF><SUB><LF> | Firma |
| 0D 00 00 00 | 13 | Lunghezza dati della componente |
| 52 44 48 49 | IHDR | Image header |
| 7F 00 00 00 | 127 | Larghezza |
| 40 00 00 00 | 64 | Altezza |
| 01 | 1 | Profondità |
| 03 | 3 | Tipo di colore: paletta |
| 00 | 0 | Metodo di compressione *deflate* |
| 00 | 0 | Metodo di filtraggio |
| 00 | 0 | Senza interlacciamento |
| 28 3A 5F 09 | | CRC |
| 03 00 00 00 | 3 | Lunghezza dati della componente |
| 45 54 4C 50 | PLTE | Palette entries |
| FF 40 40 | blu | |
| 92 10 9B 0A | | CRC |
| 12 00 00 00 | 18 | Lunghezza dati della componente |
| 54 41 44 49 | IDAT | Image Data |
| 60 A3 05 18 60 63 DA 78 | | Codici compressi |
| 01 00 40 04 00 00 F2 02 8C 14 | | |
| 0B C3 F7 03 | | CRC |
| 00 00 00 00 | 18 | Lunghezza dati della componente |
| 44 4E 45 49 | IEND | |
| 82 60 42 AE | | CRC |

## 4.7 JPEG (Joint Photographic Experts Group)

È il primo standard internazionale `ISO/CCITT` (`ISO/IEC IS 10918`) per immagini a tono continuo ed è stato creato fra il 1985 e il 1992. Esso permette due tipi di compressione: quella esatta (*lossless compression*) e quella approssimata (*lossy compression*). Con le tecniche di compressione esatta si raggiungono tipicamente rapporti di compressione di 2:1; con quelle di compressione approssimata si raggiungono invece rapporti fra 10:1 e 20:1. Il rapporto di compressione può essere variato scegliendo opportunamente i parametri necessari. Aumentando il rapporto di compressione si perde in qualità dell'immagine. Vale la pena di ricordare la regola empirica mostrata nella tabella (4.32). I decodificatori possono a loro volta fare

| bit/pixel colorato | Qualità |
|---|---|
| 0.25-0.50 | da moderata a buona |
| 0.50-0.75 | da buona a molto buona |
| 0.75-1.50 | eccellente |
| $> 1.5$ | non distinguibile dall'originale |

Tabella 4.32: Qualità delle immagini JPEG

compromessi fra velocità di decodifica e qualità. Questo formato è adatto nei casi in cui le immagini prodotte vengono osservate dall'occhio umano, meno nel caso di un'elaborazione automatizzata.
Le principali caratteristiche sono mostrate nella tabella (4.33). Lo standard definisce

| | |
|---|---|
| Nome | JPEG File Interchange Format |
| Tipo | bitmap |
| Colori | 1 ... 24 bit |
| Compressione | JPEG |
| Dimensione massima | 64K x 64K pixel |
| Più immagini per file | no |
| Formato numerico | Big endian |
| Creatore | C-Cube Microsystems |

Tabella 4.33: JPEG

quattro modi con cui codificare un'immagine:

- Modo sequenziale basato su DCT
  Permette di trasmettere sequenzialmente un'immagine compressa in modo approssimato. Da questo modo è derivato il modo *sequential baseline*.
- Modo progressivo basato su DCT
  Permette di trasmettere un'immagine in modo da aumentare la qualità progressivamente su tutta l'immagine.

- Modo sequenziale esatto
  Permette di trasmettere un'immagine compressa in modo esatto.

- Modo gerarchico
  Permette di trasmettere un'immagine in modo progressivo basata su DCT, ma in modo da poter prevedere le fasi successive.

Nelle sottosezioni seguenti ci limitiamo a discutere il modo sequenziale basato su DCT.

### 4.7.1 Compressione approssimata (*lossy compression*)

È quella che ha avuto maggior successo nelle applicazioni. Siccome il metodo di compressione usato in questo caso, viene oggi utilizzato anche in altri formati `TIFF`, `JNG`, ..., è opportuno studiarlo indipendentemente dalla struttura del file in cui i dati vengono memorizzati.
I passi necessari per produrre uno *stream* `JPEG` sono descritti nelle sottosezioni seguenti.

#### 4.7.1.1 Trasformazione dello spazio dei colori

Sebbene il metodo di compressione sia applicabile ad ogni spazio dei colori, è però più conveniente operare negli spazi luminanza/crominanza quali $YCbCr/YUV$ oppure $YIQ$. Siccome l'occhio è più sensibile alla componente di luminanza che alle due componenti di crominanza, mediante il metodo di compressione approssimata, le componenti di crominanza possono essere maggiormente compresse di quella di luminanza.

#### 4.7.1.2 Campionamento (Downsampling)

Questo passaggio è opzionale e si applica solo alle componenti di crominanza, mediando i valori fra *pixel* vicini. Di regola si mediano i valori di coppie di *pixel* sulla stessa riga (campionamento $2h1v$) oppure di quartine disposte su un quadrato (campionamento $2h2v$). Con queste tecniche il volume di dati delle crominanze si riduce nel primo caso a metà, nel secondo a un quarto del volume iniziale.

#### 4.7.1.3 Trasformazione discreta del coseno

L'immagine viene suddivisa in blocchi di $8 \times 8$ *pixel*. Ad ogni blocco viene applicato l'algoritmo di trasfomazione discreta del coseno (*discrete cosine transform*) che produce un insieme di $8 \times 8$ coefficienti $b_{uv}$ per ogni componente. Questa trasformazione permette nel prossimo passo di diminuire l'informazione necessaria per descrivere le frequenze più alte senza significative perdite di qualità, poiché la sensibilità dell'occhio diminuisce con l'aumento della frequenza delle variazioni spaziali.

I coefficienti $b_{uv}$ vengono calcolati mediante la formula

$$b_{uv} = \frac{1}{4}c(u)c(v)\sum_{i=0}^{N_1-1}\sum_{j=0}^{N_2-1} a_{ij}\cos(\frac{\pi u}{2N_1}(2i+1))\cos(\frac{\pi v}{2N_2}(2j+1)) \quad , \tag{4.1}$$

dove $i = 0, 1, \cdots, N_1 - 1, j = 0, 1, \cdots, N_2 - 1$.

I coefficienti $a_{ij}$ rappresentano le intensità dei *pixel* della riga $i$ e colonna $j$. I coefficienti $c_u$ e $C_v$ sono definiti mediante la relazione

$$c(u), c(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{se} \quad u, v = 0 \\ 1 & \text{altrimenti} \end{cases} .$$

Nel nostro caso vale $N_1 = N_2 = 8$. Il coefficiente $b_{00}$ è detto sovente coefficiente $DC$, mentre i rimanenti 63 coefficienti sono i coefficienti $AC$. La trasformata DCT è la parte più costosa dell'intero algoritmo. La relazione (4.7.1.3) può essere scritta anche nel modo seguente

$$b_{uv} = \frac{1}{2}c(u)\sum_{i=0}^{N_1-1}\cos(\frac{\pi u}{2N_1}(2i+1))\frac{1}{2}c(v)\sum_{j=0}^{N_2-1} a_{ij}\cos(\frac{\pi v}{2N_2}(2j+1)) \quad .$$

Introducendo l'abbreviazione

$$b'_{iv} = \frac{1}{2}c(v)\sum_{j=0}^{N_2-1} a_{ij}\cos(\frac{\pi v}{2N_2}(2j+1)) \quad , \tag{4.2}$$

si ottiene

$$b_{uv} = \frac{1}{2}c(u)\sum_{i=0}^{N_1-1}\cos(\frac{\pi u}{2N_1}(2i+1))b'_{iv} \quad . \tag{4.3}$$

La relazione (4.2) può essere interpretata come trasformata a una dimensione (1D) in direzione orizzontale mentre l'ultima relazione come trasformata (1D) in direzione verticale.

#### 4.7.1.4 Quantizzazione

I coefficienti vengono in seguito quantizzati mediante una matrice di normalizzazione, i cui coefficienti sono di regola piccoli in alto a sinistra e più grandi in basso a destra. Ogni coefficiente $b_{ij}$ viene diviso per il corrispondente coefficiente della matrice di quantizzazione e poi arrotondato all'intero più vicino cioè si calcola la nuova matrice $b^q_{uv}$ con la formula

$$b^q_{uv} = \text{Integer\_Round}\left(\frac{b_{uv}}{q_{uv}}\right) \quad ,$$

dove $q_{uv}$ è la matrice di quantizzazione. Di conseguenza molti coefficienti ad alta frequenza vengono ridotti a zero facilitando perciò la successiva codifica.
Le matrici di quantizzazione possono essere arbitrarie, ma in pratica si sono imposte alcune matrici ottimali. Si fa differenza però fra le matrici usate per la luminanza e quelle per la crominanza.

**Esempio di matrici di quantizzazione**
Le tabelle 4.34 e 4.35 mostrano le matrici di quantizzazione nel caso di un campionamento $2h1v$.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

Tabella 4.34: Matrice di quantizzazione della luminanza

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17 | 18 | 24 | 47 | 99 | 99 | 99 | 99 |
| 18 | 21 | 26 | 66 | 99 | 99 | 99 | 99 |
| 24 | 26 | 56 | 99 | 99 | 99 | 99 | 99 |
| 47 | 66 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |

Tabella 4.35: Matrice di quantizzazione della crominanza

#### 4.7.1.5 Codifica

Dopo la quantizzazione i coefficiente $DC$ e $AC$ sono trattati in modo diverso. Siccome i coefficienti $DC$ di blocchi vicini sono di regola molto correlati, per ogni blocco viene codificata solo la differenza con il blocco precedente. Questa differenza e i valore $AC$ sono memorizzati secondo una sequenza zig-zag, come mostrato nella figura (4.2). Il motivo della scelta di questa sequenza risiede nel fatto che i coefficienti non nulli della matrice risiedono prevalentemente in lato a sinistra della matrice. I coefficienti $DC$ e $AC$ vengono poi codificati mediante sequenze di bit a lunghezza

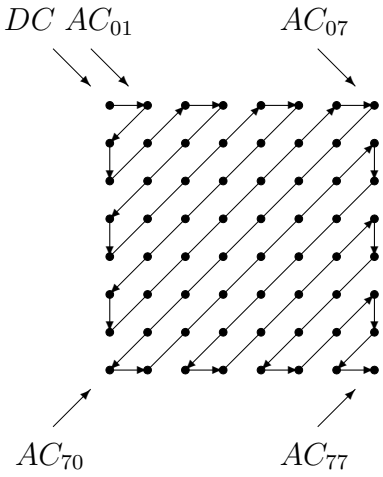



Figura 4.2: Sequenza zig-zag

variabile. Il *baseline JPEG* fa uso della codifica di Huffman. Ulteriori estensioni usano altri tipi di codifica (codifica aritmetica).

### 4.7.2 Decompressione

Il procedimento di decompressione consiste nell'esecuzione di una sequenza di passi che corrispondono nell'ordine inverso ai passi di compressione.

#### 4.7.2.1 Decodifica

La decodifica consiste nella ricostruzione dei coefficienti $b_{ij}^q$ a partire dallo *stream* compresso.

#### 4.7.2.2 Ricostruzione dei coefficienti $b_{uv}$

I coefficienti quantizzati vengono moltiplicati con i coefficienti della matrice di normalizzazione (che non deve essere necessariamente essere quella usata nella fase di codifica). Si effettua dunque il calcolo

$$b_{uv} = b_{uv}^q \cdot q_{uv} \quad .$$

#### 4.7.2.3 Trasformata inversa del coseno

Si ottiene con il calcolo

$$a_{ij} = \frac{1}{4} \sum_{u=0}^{N_1-1} \sum_{v=0}^{N_2-1} c_u c_v b_{uv} \cos(\frac{\pi u}{2N_1}(2i+1)) \cos(\frac{\pi v}{2N_2}(2j+1)) \quad .$$

### 4.7.3 JPEG File Interchange Format - JFIF

È il formato che permette lo scambio di immagini `JPEG` fra sistemi diversi. È suddiviso in segmenti introdotti da *Marker* preceduti sempre dal byte 0xFF. Questo

carattere, può anche precedere il carattere 0x00. La tabella (4.36) contiene l'elenco di alcuni *Marker* tipici. Solo i *Marker* `SOI` e `EOI` devono essere presenti una sola

| Marker | Codice | Descrizione |
|---|---|---|
| `SOI` | 0xd8 | Start of Image |
| `APP0` | 0xe0 | JFIF Application Segment |
| `APPn` | 0xe1$\cdots$ 0xef | Altri Application Segment |
| `DQT` | 0xdb | Quantization Table |
| `SOF0` | 0xc0 | Start of Frame |
| `DHT` | 0xc4 | Huffman Table |
| `SOS` | 0xda | Start of Scan |
| `EOI` | 0xd9 | End of Image |

Tabella 4.36: Elenco dei *Marker* di `JFIF`

volta nel file. Gli altri segmenti hanno la struttura mostrata nella tabella (4.37). Tipicamente un file `JPEG` *baseline* contiene la seguente sequenza di segmenti: `SOI`,

| | |
|---|---|
| 1 byte | 0xFF |
| 1 byte | *Marker* |
| 2 byte | Lunghezza (inclusi questi due byte ma non il *Marker*) |
| | Dati specifici del segmento |

Tabella 4.37: Struttura dei segmenti di `JFIF`

`APP0`, `DQT`, `SOF0`, `DHT`, `SOS`, `EOI`.

#### 4.7.3.1 Il segmento `APP0`

Questo segmento deve seguire immediatamente il *marker* `SOI`. La sua struttura è mostrata nella tabella (4.38). Esso contiene l'identificatore `JFIF`[3]. Altri segmenti `APP0`, detti estensioni, possono essere presenti, ma se presenti, devono seguire immediatamente quello identificato da `JFIF`.

**L'estensione `Thumbnail`**
Questa estensione, introdotta a partire dalla versione 1.02, contiene l'identificatore `JFXX` e permette di descrivere il `Thumbnail` in modo più flessibile che non usando direttamente il segmento `JFIF`. I tipi di descrizione possibili sono tre.

- **Thumbnail codificato JPEG**
  La sintassi dei dati compressi corrisponde a quella definita nell'annesso B della norma `ISO DIS 10918-1`.

[3]In generale, l'identificatore è una stringa terminata con `NULL`.

- **Estensione codificata con un *byte/pixel***
  La struttura dei dati in questo caso è quella mostrata nella tabella (4.40).

- **Estensione codificata con tre *byte/pixel***
  La struttura dei dati in questo caso è quella mostrata nella tabella (4.41).

| | | |
|---|---|---|
| 5 byte | 0x4A 0x46 0x49 0x46 0x00 (JFIF) | Identificatore |
| 2 byte | 0x01 0x02 | Versione |
| 1 byte | 0 | prossimi due campi → *Aspect Ratio* |
| | 1 | prossimi due campi → *dots/inch* |
| | 2 | prossimi due campi → *dots/cm* |
| 2 byte | | Densità dei *pixel* sull'asse $x$ |
| 2 byte | | Densità dei *pixel* sull'asse $y$ |
| 1 byte | | *pixel* orizzontali del *Thumbnail* $X_t$ [a]. |
| 1 byte | | *pixel* verticali del *Thumbnail* $Y_t$ |
| 3n byte | $X_t \times Y_t$ *pixel* `RGB` (24 bit) | *Thumbnail* |

[a] Thumbnail: In questo contesto significa immagine molto piccola che lascia capire come apparirà l'immagine vera. Se non si desidera memorizzare un *Thumbnail*, occorre porre a zero i due campi che definiscono la sua dimensione. La memorizzazione del *Thumbnail* nel segmento `APP0` è ammessa per ragioni di compatibilità con le prime versioni. A partire dalla versione 1.02 questa informazione viene memorizzata nell'estensione `JFXX`

Tabella 4.38: Struttura dei dati del segmento `APP0`

| | | |
|---|---|---|
| 2 byte | | Lunghezza |
| 5 byte | 0x4A 0x46 0x58 0x58 0x00 (JFXX) | Identificatore |
| 1 byte | 0x10 | *Thumbnail* codificato `JPEG` |
| | 0x11 | *Thumbnail* codificato con un *byte/pixel* |
| | 0x13 | *Thumbnail* codificato con tre *byte/pixel* |
| n byte | | dati del *Thumbnail* |

Tabella 4.39: Struttura del segmento `JFXX`

| | |
|---|---|
| 1 byte | *pixel* orizzontali del *Thumbnail* |
| 1 byte | *pixel* verticali del *Thumbnail* |
| 768 byte | paletta ($0 \cdots 255$ colori codificati con 3x8 bit) |
| n | codici dei colori di ogni *pixel* |

Tabella 4.40: Struttura dei dati del segmento `JFXX` nel caso di un *byte/pixel*

| | |
|---|---|
| 1 byte | *pixel* orizzontali del *Thumbnail* |
| 1 byte | *pixel* verticali del *Thumbnail* |
| 3n byte | pacchetti `RGB` di 24 bit per *pixel* |

Tabella 4.41: Struttura dei dati del segmento `JFXX` nel caso di tre *byte/pixel*

#### 4.7.3.2 Il segmento `DQT`

Il segmento *Define Quantisation Table* contiene le matrici di quantizzazione. La sua struttura è contenuta nella tabella (4.42).

| | |
|---|---|
| 1 byte | bit $0 \cdots 3$: numero di matrici ($0 \cdots 3$) |
| | bit $4 \cdots 7$: precisione ($0 \rightarrow 8$ bit, altrimenti 16 bit) |
| n byte | elementi delle matrici (64 * (precisione +1)) |

Tabella 4.42: Struttura dei dati del segmento `DQT`

#### 4.7.3.3 Il segmento `SOF0`

Il segmento *Start Of Frame Baseline* contiene informazioni sulla dimensione dell'immagine, sulla precisione e sul numero delle componenti usate per rappresentare i colori. I dettagli della sua struttura sono contenuti nella tabella (4.43).

| | |
|---|---|
| 1 byte | *bit/sample*, di regola 8 |
| 2 byte | altezza dell'immagine |
| 2 byte | larghezza dell'immagine |
| 1 byte | numero di componenti ($1 \rightarrow$ grigio, $3 \rightarrow$ `YCbCr` o `YIQ` |
| 1 byte[a] | identificatore della componente |
| | $1 \rightarrow$ `Y`, $2 \rightarrow$ `Cb`, $3 \rightarrow$ `Cr`, $4 \rightarrow$ `I`, $5 \rightarrow$ `Q` |
| 1 byte | *sampling factor* (bit $0 \cdots 3$ verticale, bit $4 \cdots 7$ orizzontale) |
| 1 byte | numero della matrice di quantizzazione |

[a] La seguente terna di byte è presente un numero di volte pari al numero di componenti

Tabella 4.43: Struttura dei dati del segmento `SOF0`

#### 4.7.3.4 Il segmento `DHT`

Il segmento *Define Huffman Table* contiene informazioni sulla codifica di *Huffman* usata. I dettagli della sua struttura sono contenuti nella tabella (4.44).

| | |
|---|---|
| 1 byte | *HT information* (bit $0 \cdots 3 \rightarrow$ numero di tabelle |
| | bit $4 \rightarrow 0$: tabella DC, 1: tabella AC |
| | bit $5 \cdots 7$ non usati (devono essere 0) |
| 16 byte | Numero di simboli con codici di lunghezza 1..16 |
| n byte | Tavola dei simboli |

Tabella 4.44: Struttura dei dati del segmento `DHT`

#### 4.7.3.5 Il segmento SOS

I dettagli della sua struttura sono contenuti nella tabella (4.45).

| | |
|---|---|
| 1 byte | Numero di componenti |
| 1 byte [a] | Identificatore della componente |
| 1 byte | Tabella di *Hufmann* da usare<br>(bit $0 \cdot 3 \rightarrow$ AC $(0 \cdots 3)$, bit $4 \cdot 7 \rightarrow$ DC $(0 \cdots 3)$ ) |

[a] La seguente coppia di byte è presente un numero di volte pari al numero di componenti

Tabella 4.45: Struttura dei dati del segmento SOS

### 4.7.4 JPEG Network Graphics (JNG)

È un formato membro della famiglia `xNG` nato per descrivere una singola immagine compressa in modo approssimato. Contiene uno *stream* `JPEG` e un eventuale canale *alpha* più eventuali componenti con informazioni sullo spazio dei colori e eventuali commenti.

#### 4.7.4.1 Componenti critiche

**JHDR - JNG header**
Deve apparire all'inizio e contiene le informazioni mostrate nella tabella (4.46). I

| | byte | | |
|---|---|---|---|
| Larghezza | 4 | | |
| Altezza | 4 | | |
| Tipo di colore | 1 | 1 | grigio |
| | | 10 | colore (YCbCr) |
| | | 12 | grigio-alpha |
| | | 14 | colore-alpha |
| Profondità | 1 | 8 | 8 bit (immagine e matrice di quantizzazione) |
| | | 12 | 12 bit (immagine e matrice di quantizzazione) |
| | | 20 | immagine a 8 bit seguita da una a 12 |
| Compressione | 1 | 8 | JPEG baseline codificato Huffman (`ISO-10918-1` ) |
| Interlacciamento | 1 | 0 | JPEG sequenziale, una sola scansione |
| | | 1 | JPEG progressivo |
| Profondità alpha | 1 | 0 | ,1, 2, 4, 8, 16 se il metodo di compressione alpha è 0 |
| | | 8 | se il metodo di compressione alpha è 8 |
| Compressione alpha | 1 | 0 | formato IDAT PNG a toni di grigio |
| | | 8 | formato JDAT JNG a toni di grigio a 8 bit |
| Filtraggio alpha | 1 | 0 | PNG adattivo oppure non definito (JPEG) |
| Interlacciamento alpha | 1 | | |

Tabella 4.46: JHDR - JNG header

campi `Larghezza` , `Altezza` , `Profondità` , `Compressione` e `Interlacciamento` sono ridondanti, poiché le medesime informazioni sono equivalenti a quelle contenute nella componente `JDAT`. Esse appaiono in questa componente per ragioni di convenienza e devono essere identiche a quelle contenute in `JDAT`. Se il `Tipo di colore` è 8 o 10, gli ultimi 4 byte della component `JDAT` devono essere zero. Il campo `Profondità alfa` deve essre diverso da zero se il canale alfa è presente.

**JDAT - Image data**
La componente è simile a `IDAT`. Un'immagine può essere formata da una o più componenti `JDAT`. Quando sono più di una, non è permesso intercalare fra di loro

altre componenti ad eccezione della componente `IDAT`. La regole di sequenza di altre componenti sussidiarie sono le medesime di quelle definite da `PNG`.

### 4.7.5 JPEG 2000

Questo nuovo standard prevede di sostituire la trasformata discreta del coseno con la tecnica dei *wavelet*. Compressione approssimata e esatta possono essere mescolate. Il nuovo formato è molto resistente a errori di trasmissione, ciò che permette l'uso anche su canali molto disturbati.
Una descrizione esaustiva del formato è contenuta ad esempio nell'articolo pubblicato da *G.K.Wallace, The JPEG Still Picture Compression Standard*.

## 4.8 CGM

| | |
|---|---|
| Nome | Computer Graphics Metafile |
| Tipo | metafile |
| Colori | senza limiti |
| Compressione | RLE , CCITT Group 3 , CCITT Group 4 |
| Dimensione massima | senza limiti |
| Più immagini per file | si |
| Formato numerico | |
| Creatore | ANSI , ISO |

Tabella 4.47: CGM

### 4.8.1 Introduzione

Questo formato è stato pubblicato a partire dal 1987 ed è descritto nella norma $ISO - 8632$. La norma si compone dei quattro volumi seguenti:

- `ISO 8632-1` Part 1: Functional Specification
- `ISO 8632-2` Part 2: Character Encoding
- `ISO 8632-3` Part 3: Binary Encoding
- `ISO 8632-4` Part 4: Clear Text Encoding

È adatto per descrivere oggetti grafici quando la rappresentazione vettoriale risulta vantaggiosa rispetto a quella a *pixel*. Osserviamo che accanto a $CGM$ esistono anche altri formati di tipo vettoriale. Parecchi sono però proprietari quali `PDF` (Adobe), `WMF` (Microsoft), `DXF` (AutoCAD) e `VML` (Microsoft).

La rappresentazione vettoriale offre i vantaggi seguenti:

- Scalabilità;
- Risoluzione indipendente dalla dimensione;
- Può fare uso di un insieme ricco di primitive geometriche;
- Può memorizzare testo in modo nativo e quindi facilmente ritrovabile;
- Permette una descrizione parametrica delle curve.

Con questo formato si possono creare immagini sia mediante l'uso di primitive geometriche, sia mediante sequenze di *pixel*. Per questo motivo si parla di *metafile*. Un metafile è una lista di comandi che permettono di produrre immagini.
Ogni metafile è una collezione di *elementi* con differente funzionalità. Gli elementi sono raggruppabili in classi.

### 4.8.2 Classi di elementi

#### 4.8.2.1 Elementi delimitatori (classe 0)

Ogni metafile inizia con l'elemento `BEGIN METAFILE` e termina con l'elemento `END METAFILE`. Ciò permette di memorizzare e trasferire nello stesso file più metafile.
All'interno di un metafile ogni *immagine* inizia con l'elemento `BEGIN PICTURE` e termina con l'emento `END PICTURE`. La descrizione dell'immagine è separata dal suo corpo dall'elemento `BEGIN PICTURE BODY`.
Gli elementi `BEGIN METAFILE` e `BEGIN PICTURE` sono associati ognuno ad un parametro che permette di identificare il metafile rispettivamente l'immagine.
Nelle versioni 2,3 e 4 le primitive possono essere riunite a formare primitive composite dette *figure*. Le figure sono delimitate dagli elementi `BEGIN FIGURE` e `END FIGURE`.
Nelle versioni 2,3 e 4, gruppi di elementi possono anche formare *segmenti* delimitati dagli elementi `BEGIN SEGMENT` e `END SEGMENT`.
Nelle versioni 3 e 4 è anche possibile definire vettori di elementi includenti fra `BEGIN TILE ARRAY` e `END TILE ARRAY`.

#### 4.8.2.2 Descrittori di metafile (classe 1)

Definiscono le caratteristiche del metafile, le condizioni di difetto, il contenuto funzionale e altre particolarità. Seguono alcuni esempi.

- `METAFILE VERSION` - Versione
- `METAFILE DESCRIPTION` - Specifica fra l'altro l'origine, il proprietario e la data di creazione.
- `METAFILE ELEMENT LIST` - Contiene la lista degli elementi non obbligatori utilizzati. Gli elementi obbligatori devono essere contenuti in ogni metafile.
- `VDC` - *Virtual Device Coordinates*. Può assumere i valori `INTEGER` oppure `REAL`.
- `INTEGER PRECISION` , `REAL PRECISION` - Specifica la precisione degli operandi interi o reali
- `COLOUR PRECISION` - Precisione dei colori definita come ampiezza del campo numerico che specifica ogni colore
- `COLOUR INDEX PRECISION` - Ampiezza del campo numerico che specifica l'indice dei colori
- `MAXIMUM COLOUR INDEX` - Indice dei colori più grande
- `FONT LIST` - Permette di specificare quali *font* sono richiesti dal metafile
- `FONT PROPERTIES` , `GLYPH MAPPING` - Forniscono indicazioni sui *font*, utili nel caso in cui i font richiesti nella `FONT LIST` non sono disponibili

- `CHARACTER SET LIST` - permette di specificare quali insiemi di caratteri (pubblici e privati) sono usati
- `CHARACTER CODING ANNOUNCER` - Permette di definire quale carattere di controllo è usato per invocare e designare un insieme di caratteri.

#### 4.8.2.3 Descrittori di immagini (classe 2)

Definiscono in che modo interpretare le descrizioni delle immagini.

- `SCALING MODE` - Specifica il tipo di spazio in cui è definita l'immagine. Lo spazio può essere di due tipi:
  - *astratto*
    Lo spazio è senza dimensione e l'immagine è visualizzata correttamente in ogni scala.
  - *metrico*
    Lo spazio è metrico. Un' unità `VDC` rappresenta un millimetro moltiplicato per il fattore di scala.
- `COLOUR SELCTION MODE` - Permette di selezionare fra colori *indicizzati* o *diretti*. Nella prima versione questo elemento è valido per l'intera immagine. Nelle versioni successive questo elemento può apparire sia nel corpo dell'immagine che nella descrizione dell'immagine.
- `LINE WIDTH SPECIICATION MODE` - Permette di specificare il modo per definire lo spessore di una riga. Quattro modi sono possibili:
  - in unità `VDC`;
  - mediante un fattore di scala applicato a una larghezza nominale dipendente dal dispositivo grafico;
  - con una frazione della superficie visibile del dispositivo;
  - in millimetri.
- `MARKER SIZE SPECIFICATION MODE` - Permette di specificare il modo per definire le dimensioni dei *marker*.
- `EDGE WIDTH SPECIFICATION MODE` - Permette di specificare il modo per definire i contorni delle figure.
- `VDC EXTENT` - Definisce il senso di orientamento dello spazio cioè la direzione positiva degli assi $x$ e di $y$ e quale regione dell'immagine sarà visualizzata nel *device viewport*.
- `DEVICE VIEWPORT` - Specifica la finestra visualizzata sul dispositivo grafico mediante i due punti estremi della diagonale del rettangolo.

- `VIEWPORT SPECIFICATION MODE` - Permette di specificare il *viewport* in tre modi diversi:
  - in una frazione nell'intervallo $0.0 \cdots 1.0$;
  - in millimetri;
  - in coordinate fisiche.

  Esiste anche la possibilità di mappare il `VDC EXTENT` in una porzione del *viewport* facendo uso dell'elemento `DEVICE VIEWPORT MAPPING`.
- `BACKGROUND COLOR` - Specifica il colore dello sfondo
- `LINE REPRESENTATION` - Permette di far riferimento agli attributi che specificano il tipo, lo spessore e il colore di una linea.
- `MARKER REPRESENTATION` - Permette di far riferimento agli attributi che specificano il tipo, la dimensione e il colore di un *marker*.
- `TEXT REPRESENTATION` - Permette di far riferimento agli attributi che specificano il *font*, la precisione, il fattore di espansione del carattere e il colore del testo.
- `FILL REPRESENTATION` - Permette di far riferimento agli attributi che specificano lo stile, il colore e il tessuto di una superficie.
- `EDGE REPRESENTATION` - Permette di far riferimento agli attributi che specificano il tipo, lo spessore e il colore di un contorno.

#### 4.8.2.4 **Elementi di controllo** (classe 3)

- `CLIP RECTANGLE` - Permette di definire l'area rettangolare da visualizzare quando il `CLIP INDICATOR` è "on".
- `VDC INTEGER` e `VDC REAL` - Permettono di definire l' intervallo all'interno del quale è definita l'immagine.

#### 4.8.2.5 **Primitive grafiche** (classe 4)

Permettono il disegno di oggetti geometrici.

- *Elementi per disegnare linee.*
  - `POLYLINE`
    È una poligonale definita mediante una lista di punti.
  - `DISJOINT POLYLINE`
    È una sequenza di segmenti ottenuti connettendo fra di loro coppie di punti successivi di una lista.

– `CIRCULAR ARC 3 POINT`
È un arco definito mediante 3 punti.

– `CIRCULAR ARC CENTRE`
È un arco definito mediante centro, raggio e due versori.

– `CIRCULAR ARC CENTRE REVERSED`
È un arco definito mediante centro, raggio e due versori. L'arco è disegnato in senso negativo.

– `ELLIPTICAL ARC`
È un arco ellittico definito mediante centro, il punto finale di due diametri coniugati e due versori.

– `CONNECTING EDGE`
È il segmento che congiunge l'ultimo punto dell'elemento precedente con il primo del prossimo elemento o dell'elemento precedente (in questo ulitmo caso per generare linee chiuse).

– `HYPERBOLIC ARC`
È un arco di iperbole definito mediante il centro, due punti finali di raggi e due versori.

– `PARABOLIC ARC`
È un arco di parabola definito mediante tre punti.

– `NON-UNIFORM B-SPLINE`
È una *spline* definita dal suo ordine, dai punti di controllo, (il cui numero deve essere almeno pari all'ordine), dal vettore nodale e dagli estremi dell'intervallo di parametrizzazione. Matematicamente la curva è definita con la relazione

$$\mathbf{G}(t) = \sum_{i=1}^{n} \mathbf{P}_i B_{i,k}(t) \quad , \tag{4.4}$$

dove $n$ è il numero di punti di controllo, $\mathbf{P}_i = (x_i, y_i)$ sono i punti di controllo e $B_{i,k}$ sono le funzioni *B-spline* di base di ordine $k$ associate al vettore nodale $t_1, t_2, \cdots, t_{n+k}$. La curva stessa è definita nell'intervallo $t_k \leq t \leq t_{n+1}$. Il disegno della curva può essere contenuto nell'intervallo $[t_{min} \cdots t_{max}]$, dove

$$t_k \leq t_{min} \leq t_{max} \leq t_{n-1} \quad .$$

Le funzioni di base sono definite dalle relazioni ricorsive

$$B_{i,1}(t) = \begin{cases} 1 & \text{se } t_i \leq t < t_{i+1} \\ 0 & \text{altrimenti} \end{cases}$$

e per $k > 1$ da

$$B_{i,k}(t) = \begin{cases} \dfrac{(t - t_i) B_{i,k-1}(t)}{t_{i+k-1} - t_i} + \dfrac{(t_{i+k} - t) B_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}}, & \text{se } t_i \leq t < t_{i+k} \\ 0, & \text{altrimenti} \end{cases}$$

Nei casi in cui dovesse verificarsi che la formula produce 0/0, i termini corrispondenti sono da porre per definizione a zero.

- `NON-UNIFRM RATIONAL B-SPLINE` Questa curva è definita dalla relazione

$$\mathbf{G}(t) = \frac{\sum_{i=1}^{n} B_{i,k}(t)\mathbf{P}_i w_i}{\sum_{i=1}^{n} B_{i,k}(t) w_i} \quad ,$$

dove $w_i$ sono dei pesi.

- `POLYBEZIER`
  In questo caso si usano curve di *Bezier* cubiche.

- *Elemento per disegnare punti marcanti*:
  `POLYMARKER`

- *Elementi per disegnare testo*

  - `TEXT`
    Genera una stringa di testo allineata a punto dato.
  - `RESCRICTED TEXT`
    Genera una stringa di testo contenuta in un'area data.
  - `APPEND TEXT`
    Aggiunge del testo iniziato con uno dei due elementi precedenti.

- *Elementi per disegnare aree*

  - `POLYGON`
    Genera un'area con contorno definita da una lista di punti. Lo stile può essere:*hollow, solid, pattern, hatch, emtpy, geometric pattern* o *interpolated*
  - `POLYGON SET`
    Genera un insieme di aree con contorno.
  - `RECTANGLE`
    Genera un rettangolo.
  - `CIRCLE`
    Genera un cerchio.
  - `CIRCULAR ARC 3 POINT CLOSE`
    Genera un settore circolare secondo le stesse regole definite per `CIRCULAR ARC 3 POINT`. Il settore può essere tipo "corda" o tipo "torta".
  - `CIRCULAR ARC CENTRE CLOSE`
    Genera un settore circolare secondo le stesse regole definite per `CIRCULAR ARC CENTRE`. Il settore può essere tipo "corda" o tipo "torta".
  - `ELLIPSE`
    Genera un ellisse per un dato centro e in base a due punti finali di due diametri coniugati.

  - `ELLICTICAL ARC CLOSE`
    Genera un settore ellittico secondo le regole definite per `ELLICTICAL ARC`. Il settore può essere tipo "corda" o tipo "torta".

- *Celle vettoriali*

  - `CELL ARRAY` È un vettore bidimensionale i cui valori sono i colori delle celle. I colori possono essere diretti o indicizzati mediante una `COLOUR TABLE` e in accordo con il `COLOUR SELECTION MODE`.
  - `BITONAL TILE/TILE` Questi due elementi sono le celle elementari che permettono di comporre elementi rettangolari complessi. Questi elementi vengono racchiusi fra `BEGIN TILE ARRAY` e `END TILE ARRAY`.

#### 4.8.2.6 Attributi che specificano l'apparenza delle primitive grafiche (classe 5)

- Attributi per le linee
  Sono `LINE TYPE` (*dotted, dashed,...*), `LINE WIDTH` e `LINE COLOUR`.

- Attributi per i marcanti
  Sono `MARKER TYPE`, `MARKER SIZE` e `MARKER COLOUR`.

- Attributi per il riempimento di superfici
  Sono `INTERIOR STYLE`, `FILL COLOUR`, `HATCH INDEX` e `PATTERN INDEX`.

- Attributi per i contorni
  Sono `EDGE TYPE`, `EDGE WIDTH` e `EDGE COLOUR`.

- Attributi per il testo
  Sono `TEXT FONT INDEX`, `TEXT PRECISION`, `CHARACTER EXPANSION FACTOR`, `CHARACTEC SPACING` e `TEXT COLOUR`.

#### 4.8.2.7 Elementi escape (classe 6)

Contengono informazioni dipendenti dal sistema e quindi non standardizzati;

#### 4.8.2.8 Elementi esterni (classe 7)

Servono per la comunicazione non strettamente collegata alla generazione dell'immagine;

#### 4.8.2.9 Segmenti (classe 8)

Permettono di raggruppare oggetti per poter compiere operazioni quali la copia o la ricerca;

#### 4.8.2.10 Descrittori di strutture applicative (classe 9)

### 4.8.3 Codifiche

Un file `CGM` può essere codificato in tre modi diversi.

#### 4.8.3.1 Codifica binaria

L'informazione è strutturata usando parole di 16 bit. La prima parola si ogni elemento contiene informazioni sulla classe, sul tipo di elemento e sul numero di argomenti associati come mostra lo schema che sgue.

| 15 14 13 12 | 11 10 9 8 7 6 5 | 4 3 2 1 0 |
|---|---|---|
| classe | identificatore | lunghezza lista argomenti |
| ... | | |

Gli argomenti sono elencati immediatamente sotto, come mostrato nel prossimo esempio.

| | | | |
|---|---|---|---|
| Polyline | 4 | 1 | 16 |
| punto 0,2 | 0 | | |
| | 2 | | |
| punto 1,3 | 1 | | |
| | 3 | | |
| punto 2,1 | 2 | | |
| | 1 | | |
| punto 0,2 | 0 | | |
| | 2 | | |

In parecchi casi questo tipo di codifica permette di minimizzare lo sforzo necessario per generare e interpretare il metafile.

#### 4.8.3.2 Codifica a caratteri

Ogni elemento è codificato mediante un codice e dei parametri.

**Codici di base**
Ogni codice è composto da uno o due byte. I codici a un byte hanno la struttura $X010bbbb$, dove $X$ è il bit più significativo ed è presente solo nelle codifiche a 8 bit. I codici a due byte hanno la struttura $X011bbbb$ $X01bbbb$. Sovente il primo byte si abbrevia con 2/x, gli ultimi con 3/y 2/z rispettivamente con 3/y 3/z.

**Codici estesi**
Sono introdotti da una sequenza di uno o più byte 3/15 alla quale fanno seguito le sequenze di base precedenti. I valori ammessi $x$, $y$ e $z$ sono i seguenti:

| | |
|---|---|
| x | 0,1,$\cdots$ , 15 |
| y | 0,1,$\cdots$ , 14 |
| z | 0,1,$\cdots$ , 15 |

Questo tipo di codifica risulta essere la più compatta e quindi adatta nei casi in cui il metafile deve essere il più piccolo possibile.

#### 4.8.3.3 Codifica in chiaro

Questo tipo di codifica permette di creare, modificare e leggere comodamente un metafile.

## 4.9 WebCMG

È un profilo particolare di `CGM` definito dal consorzio `W3C` (*www.w3.org*) per il WEB. Usa un sottoinsieme di elementi definiti da `CGM`, ma introduce anche delle novità. L'unica codifica ammessa è quella binaria.

La struttura tipica di un file `WebCGM` è la seguente.

*BEGIN METAFILE*
*Metafile description*

*Picture 1*

*Picture 2*

*...*

*Picture n*

*END METAFILE*

Ogni figura ha la struttura seguente.

*BEGIN PICTURE*
*Picture description*

*Picture body*

*END PICTURE*

Le primitive grafiche ammesse sono un sottoinsieme di quelle definite da `CGM`.

## 4.10 SVG (Scalable Vector Graphics)

Questo formato è stato ideato dal *World Wide Web Consortium* per l'uso in rete. Esso permette la realizzazione di grafica vettoriale a due dimensioni, offre la possibilità di creare animazioni e include funzionalità che permettono l'interazione dell'utente con gli oggetti grafici. Il formato si basa sul linguaggio `XML` e quindi può essere interpretato direttamente dai moderni *browser*. Per poter essere usato con i *browser* delle geneazioni precedenti, occorre disporre di un apposito *Plugin*.

### 4.10.1 Esempio introduttivo

La figura (4.3) mostra un semplice esempio di formato `svg` che rappresenta un insieme di quattro rettangoli, ognuno con attributi diversi. La prima riga permette di

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
   <title>Rettangoli</title>
   <desc> Quattro rettangoli con attributi diversi</desc>

   <!-- Quattro rettangoli -->

   <rect x="50"  y="150" width="100" height="70"
     fill="green"/>
   <rect x="120" y="100" width="100" height="70"
     fill="yellow" stroke="black" />
   <rect x="200" y="150" width="100" height="70" rx="5"
     fill="blue" opacity="0.4" stroke="black" stroke-width="4"/>
   <rect x="120" y="200" width="100" height="70"
     fill="none" stroke="black" stroke-width="2"/>
</svg>
```

Figura 4.3: Quattro rettangoli

riconoscere un file formato `XML`. La seconda specifica che si tratta di un file `svg`, la cui struttura è definita mediante la *Document Type Definition - DTD* contenuta in *svg10.dtd*.
La descrizione grafica inizia con il *tag* `svg`. Al suo interno si specificano innanzitutto attributi che permettono di fissare le dimensioni del disegno (in questo caso in *pixel*) e la locazione del *Namespace*.

Gli elementi `title` e `desc`, usati all'interno di altri, permettono di identificare e

descrivere testualmente l'elemento che li contengono. Il testo non viene di regola visualizzato, ma lo può essere ad esempio nel caso in cui un elemento che reagisce al *mouse*.

L'attributo `fill`, usato all'interno dell'elemento `rect` definisce il colore con cui viene disegnata la superficie di riempimento del rettangolo; gli attributi `stroke` e `stroke-width`, il colore del contorno rispettivamente il suo spessore in *pixel*.

Oggi oggetto disegnato dopo si sovrappone agli oggetti disegnati prima nella stessa posizione.

### 4.10.2 Visualizzazione del file

Un file `SVG` può essere visualizzato direttamente da un *browser* oppure può essere referenziato da un altro file di vario tipo (`svg, html, xml, ...`).

La figura (4.4) mostra come inglobare un file `svg` all'interno di un file `html`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/html4/transitional.dtd">
<html><head><title>SVG in HTML</title></head>

<body style="margin-left:20px; margin-top:10px;">
<p style="font-family:Verdana;">

File SVG inserito in un file HTML mediante il tag object<br>
</p>

<object x="100" data="rettangoli.svg" type="image/svg+xml"
        width="400" height="400">
</object>

</body>
</html>
```

Figura 4.4: File .svg richiamato come oggetto in un file .html

### 4.10.3 Gli elementi *group* e *use*

Permettono, contemporanemante alla visualizzazione, di raggruppare oggetti predefiniti in un unico oggetto identificato da una etichetta e di visualizzarlo una o più volte in seguito. Eventuali attributi validi per tutti gli oggetti del gruppo possono essere specificati come attributi del gruppo, come mostrato nella figura (4.5).

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="800" height="800" xmlns="http://www.w3.org/2000/svg">

<title>Gruppo</title>
<desc>Gruppo di quattro rettangoli con trasformazioni</desc>

  <g id="QuattroRettangoli" stroke="black" stroke-width="2">
    <circle cx="0"  cy="0" r="5"  fill="black"/>
    <rect x="0"  y="0" width="350" height="210"
      fill="none" stroke="black" />
    <circle cx="0"  cy="0" r="5"  fill="black"/>

    <rect x="50"  y="70" width="100" height="70"
      fill="green"/>
    <rect x="120" y="20" width="100" height="70"
      fill="yellow" />
    <rect x="200" y="70" width="100" height="70" rx="5"
      fill="blue" opacity="0.4" />
    <rect x="120" y="120" width="100" height="70"
      fill="none" stroke-width="5"/>
  </g>

  <use x="400" y="0" xlink:href="#QuattroRettangoli" />
  <use  xlink:href="#QuattroRettangoli"
     transform="translate(500,300) scale(0.5)" />

  <use  xlink:href="#QuattroRettangoli"
     transform="translate(100,250) " />
  <use  xlink:href="#QuattroRettangoli"
     transform="translate(100,250) rotate(20) " />
</svg>
```

Figura 4.5: Uso degli elementi *group* e *use*

### 4.10.4 Attributo *transform*

L'ultimo esempio contiene alcune trasformazioni geometriche, quali la traslazione, la rotazione e la dilatazione di un oggetto. Queste possono essere specificate mediante l'attributo `transform`. Le trasformazioni possono essere concatenate fra di loro. Durante ogni trasformazione il sistema di riferimento è solidale con l'oggetto. Le trasformazioni possibili sono riportate nelle prossime sottosezioni.

#### 4.10.4.1 Traslazione

Si ottiene con *trasform*=translate(x,y). Se $y$ non è presente, si assume che il suo valore sia nullo.

#### 4.10.4.2 Rotazione

Si ottiene con *trasform*=rotate($\phi$,$c_x$,$c_y$). Se $c_x$ e $c_y$ non sono presenti, si assume che la rotazione avviene attorno all'origine del sistema d'assi attuale.

#### 4.10.4.3 Dilatazione

Si ottiene con *trasform*=scale($s_x$,$s_y$). Se $s_y$ non è presente, si assume che il suo valore sia uguale a quello di $s_x$.

#### 4.10.4.4 Distorsione

Si ottiene con *trasform*=skewX($\phi$) oppure *trasform*=skewY($\phi$). Le due trasformazioni sono definite mediante le matrici omogenee (non trasposte) seguenti.

$$\begin{pmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad , \quad \begin{pmatrix} 1 & 0 & 0 \\ \tan\phi & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### 4.10.5 Elemento *symbol*

Se si desidera definire un oggetto grafico, ma non visualizzarlo immediatamente durante la definizione, si può usare l'elemento `symbol`, come mostra la figura (4.6).

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="800" height="800" xmlns="http://www.w3.org/2000/svg">

  <title>SimboloQuattroRettangoli</title>
  <desc>Gruppo di quattro rettangoli </desc>

  <symbol id="QuattroRettangoli" stroke="black" stroke-width="2">
     <circle cx="0"  cy="0" r="5"  fill="black"/>
     <rect x="0"  y="0" width="350" height="210"
        fill="none" stroke="black" />
     <circle cx="0"  cy="0" r="5"  fill="black"/>

     <rect x="50"  y="70" width="100" height="70"
        fill="green"/>
     <rect x="120" y="20" width="100" height="70"
        fill="yellow" />
     <rect x="200" y="70" width="100" height="70" rx="5"
        fill="blue" opacity="0.4" />
     <rect x="120" y="120" width="100" height="70"
        fill="none" stroke-width="4"/>
  </symbol>
  <use x="400" y="0" xlink:href="#QuattroRettangoli" />
  <use  xlink:href="#QuattroRettangoli"
       transform="translate(500,300) scale(0.5)" />

  <use  xlink:href="#QuattroRettangoli"
       transform="translate(100,250)" />
  <use  xlink:href="#QuattroRettangoli"
       transform="translate(100,250) rotate(20) " />
</svg>
```

Figura 4.6: Uso dell'elemento*symbol*

### 4.10.6 Elemento *defs*

Ha funzionalità analoga a quella dell'elemento `symbol`, ma è di uso più generale.

### 4.10.7 Elemento *image*

Permette di includere in un file *.svg* un'immagine. I formati ammessi sono *png*, *jpg*, *gif*, oltre al formato *svg* stesso, come mostrato nella figura (4.7).

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="800" height="800" xmlns="http://www.w3.org/2000/svg">
   <title>image_defs</title>
   <desc>Inclusione di immagini e uso di "defs" </desc>
   <defs>
     <g id="escher">
       <rect x="0"  y="0" width="350" height="210"
         fill="none" stroke="black" />
       <image x="100" y="10" width="160px" height="180px"
         xlink:href="escher.jpg">
       </image>
     </g>
     <g id="peperoni">
       <rect x="0"  y="0" width="350" height="210"
         fill="none" stroke="black" />
       <image x="100" y="10" width="180px" height="180px"
         xlink:href="peppers.png">
       </image>
     </g>
   </defs>

   <use x="0" y="0" xlink:href="#escher" />
   <use x="400" y="0" xlink:href="#peperoni" />
   <use xlink:href="#peperoni"
      transform="translate(400,250) rotate(20,175,105)  " />

</svg>
```

Figura 4.7: Uso degli elementi *defs* e *image*

### 4.10.8 Elemento *style*

Permette di definire *classi* di attributi che possono essere più tardi citati nell'istanziazione di oggetti grafici. Nella figura (4.8) vengono definite le classi *VuotoContornoNero*, *GialloContornoRossoSpesso* e *BluTrasparente*.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
 <svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
   <title>Cerchi con diversi style</title>
   <desc> Uso di style </desc>

   <!-- Definizione dello stile  -->
   <defs>
     <style type="text/css">
     <![CDATA[
       .Giallo {fill:yellow;}
       .VuotoContornoNero {fill:none; stroke:black;}
       .GialloContornoRossoSpesso
          {fill:yellow; stroke:red; stroke-width:0.2cm;}
       .BluTrasparente {fill:blue; stroke:black; opacity:.4;}
     ]]>
     </style>
   </defs>

   <!-- Quattro cerchi -->
   <circle cx="100" cy="150" r="70" class="Giallo"/>
   <circle cx="170" cy="100" r="70" class="VuotoContornoNero" />
   <circle cx="250" cy="150" r="70" class="GialloContornoRossoSpesso"/>
   <circle cx="170" cy="200" r="70" class="BluTrasparente"/>
 </svg>
```

Figura 4.8: Uso dell' elemento *style*

Lo stile può essere definito anche all'esterno, in un *Cascading Style Sheet*. In

```
.Giallo {fill:yellow;}
.VuotoContornoNero {fill:none; stroke:black;}
.GialloContornoNeroSpesso
   {fill:yellow; stroke:black; stroke-width:0.2cm;}
.BluTrasparente {fill:blue; stroke:black; opacity:.4;}
```

Figura 4.9: File **miostile.css**

questo caso il file *.svg* fa riferimento al file *.css* , come mostrato nelle figure (4.9) e (4.10).

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>

<?xml-stylesheet type="text/css" href="miostile.css" ?>

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
   <title>Uso di style</title>
   <desc> Uso di style con file .CSS  </desc>

   <!-- Quattro cerchi -->
   <circle cx="100" cy="150" r="70" />
   <circle cx="170" cy="100" r="70" class="VuotoContornoNero" />
   <circle cx="250" cy="150" r="70" class="GialloContornoNeroSpesso"/>
   <circle cx="170" cy="200" r="70" class="BluTrasparente"/>
</svg>
```

Figura 4.10: Uso di un file *.css* esterno

### 4.10.9 Elementi *line. polyline* e *polygon*

La figura (4.11) mostra l'uso degli elementi `line`, `polyline` e `polygon`. L'atttributo `stroke-linecap` permette di definire la forma con cui termina un segmento o una poligonale. Tre valori sono possibili: *butt*, *round* e *square*. La differenza di significato frea il primo e il terzo caso consiste nel fatto che nel primo la linea che demarca l'estremità del segmento passa per il punto estremo del segmento stesso. Nel secondo caso la linea è tracciata oltre la fine del segmento, ad una distanza dipendente dallo spessore dello stesso.

L'attibuto `stroke-linejoin` permette invece di specificare la forma dei giunti nelle poligonali. Tre valori sono possibili: *miter*, *round* e *bevel*.

L'attributo `stroke-dasharray` permette di specificare il tipo di tratteggio come sequenza alternata di unità piene e vuote.

Infine l'attributo `fill-rule` permette di specificare l'algoritmo usato per riempire la superficie di un poligono.

```
  <!-- Definizione dello stile  -->
  <defs>
    <style type="text/css">
     <![CDATA[
       .SquareMiter  {stroke:green; stroke-width:4;}
       .Roundcap {stroke:red; stroke-width:4;stroke-linecap:round;}
       .Roundjoin {stroke:blue; stroke-width:4;stroke-linejoin:round;}
       .DashArray {stroke:black; stroke-width:4;stroke-dasharray:15 3;}
       .DashArrayRJ {stroke:black; stroke-width:4;stroke-dasharray:20 5;
                     stroke-linejoin:round;}
       .Nonzero {fill:silver; stroke:black; stroke-width:4;}
       .Evenodd {fill:silver; stroke:black; stroke-width:4; fill-rule:evenodd;}
     ]]>
    </style>
  </defs>

  <symbol id="zigzag">
    <polyline fill="none"
          points="100,120 120,100 120,100 100,120 100,140 140,100 160,100
                  100,160 100,180 180,100 200,100 100,200 100,220"/>
  </symbol>

  <symbol id="Stella">
    <polygon points="200,100 19,158 130,4.8  130,196 19,41"/> </symbol>

  <symbol id="Segmenti">
    <line x1="50" y1="50" x2="550" y2="450"/>
    <line x1="50" y1="450" x2="500" y2="50"/>
  </symbol>

  <use  xlink:href="#zigzag" class="SquareMiter"/>
  <use  xlink:href="#zigzag" class="Roundjoin" transform="translate(0,120)"/>
  <use  xlink:href="#zigzag" class="Roundcap" transform="translate(0,240)" />
  <use  xlink:href="#zigzag" class="DashArray" transform="translate(160,0)"/>
  <use  xlink:href="#zigzag" class="DashArrayRJ"
       transform="translate(160,120)" />
  <use  xlink:href="#Segmenti" class="Roundcap"/>
  <use  xlink:href="#Stella"  class="Nonzero"
       transform="translate(400,30) rotate(30,100,100)"/>
  <use  xlink:href="#Stella" class="Evenodd"
       transform="translate(400,230) rotate(50,100,100)"/>
</svg>
```

Figura 4.11: Segmenti, poligonali e poligoni

### 4.10.10 Elemento *path*

Permette di realizzare la grafica in modo *plotter incrementale*. Le funzioni disponibili sono mostrate nella tabella (4.48). Le figure (4.12) e (4.13) contengono esempi di

| Comando | Funzione | Arg. | Descrizione |
|---|---|---|---|
| M | moveto | $x\ y$ | Movimento alle coordinate assolute $(x, y)$ |
| m | moveto | $x\ y$ | Movimento alle coordinate relative $(+x, +y)$ |
| L | lineto | $x\ y$ | Segmento da posizione attuale a posizione assoluta $(x, y)$ |
| l | lineto | $x\ y$ | Segmento da posizione attuale a posizione relativa $(+x, +y)$ |
| V | vert. lineto | $y$ | Segmento verticale a posizione assoluta $(y)$ |
| v | vert. lineto | $y$ | Segmento verticale a posizione relativa $(+y)$ |
| H | hor. lineto | $x$ | Segmento orizzontale a posizione assoluta $(x)$ |
| h | hor. lineto | $x$ | Segmento orizzontale a posizione relativa $(+x)$ |
| z | closepath | | Congiunzione con il punto iniziale |
| A | elliptical arc | | Arco da posizione attuale a posizione assoluta |
| | | $r_x\ r_y$ | Raggi |
| | | $o$ | *axis orientation* |
| | | $l$ | *large-arc-flag*: se 1 scelta arco più lungo |
| | | $s$ | *sweep-flag*: se 1, arco disegnato in senso orario |
| | | $x\ y$ | Coordinate finali |
| a | elliptical arc | | Arco da posizione attuale a posizione relativa |
| C | curveto | | Bézier cubica da posizione attuale a posizione assoluta |
| | | $x_1\ y_1$ | Primo punto di controllo |
| | | $x_2\ y_2$ | Secondo punto di controllo |
| | | $x\ y$ | Posizione finale |
| c | curveto | | Bézier cubica da posizione attuale a posizione relativa |
| S/s | smooth | | Bézier cubica: primo punto di controllo riflesso dell'ultimo precedente |
| Q | curveto | | Bézier quadratica da posizione attuale a posizione assoluta |
| | | $x_1\ y_1$ | Punto di controllo |
| | | $x\ y$ | Posizione finale |
| q | curveto | | Bézier quadratica da posizione attuale a posizione relativa |
| T/t | smooth | | Bézier quadratica: primo punto di controllo riflesso dell'ultimo precedente |

Tabella 4.48: Funzioni disponibili per il *plotter incrementale*

file che usano le funzioni descritte.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
 <svg width="800" height="400" xmlns="http://www.w3.org/2000/svg">
   <title>path_lineto_elliptical_arc.svg</title>
   <desc> Moveto, lineto e elliptical arc</desc>

   <path  d="M50 50 l40 60 v -50 h50 m 50 0 h50 v50 l-200 50 z"
          stroke="black" stroke-width="4" fill="none" stroke-linecap="round" />

   <path  d="M120 220 a80 80 0  0 0 100 100 "
          stroke="blue" stroke-width="4" fill="none" stroke-linecap="round" />
   <path  d="M120 220 a80 80 0  0 1 100 100 "
          stroke="red" stroke-width="4" fill="none" stroke-linecap="square" />
   <path  d="M120 220 a80 80 0  1 0 100 100 "
          stroke="yellow" stroke-width="4" fill="none" stroke-linecap="butt" />
   <path  d="M120 220 a80 80 0  1 1 100 100 "
          stroke="green" stroke-width="4" fill="none" stroke-linecap="round" />

   <path  d="M500 150 a80 40 0  0 0 100 100 "
          stroke="blue" stroke-width="4" fill="none" stroke-linecap="round" />
   <path  d="M500 150 a80 40 0  0 1 100 100 "
          stroke="red" stroke-width="4" fill="none" stroke-linecap="square" />
   <path  d="M500 150 a80 40 120  1 0 100 100 "
          stroke="yellow" stroke-width="4" fill="none" stroke-linecap="butt" />
   <path  d="M500 150 a80 40 120  1 1 100 100 "
          stroke="green" stroke-width="4" fill="none" stroke-linecap="round" />
 </svg>
```

Figura 4.12: Moveto, lineto e elliptical arc

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
 <svg width="800" height="400" xmlns="http://www.w3.org/2000/svg">
   <title>path_bezier.svg</title>
   <desc> Bezier cubiche e quadratiche </desc>

   <path  d="M100 100 c75 50 75 50 150 0 c75 -50 75 -50 150 0 "
          stroke="black" stroke-width="4" fill="none" />
   <path  d="M100 200 c75 50 75 50 150 0 s75 -50 150 0 "
          stroke="blue" stroke-width="4" fill="none" />
   <path  d="M100 300 c50 50 100 50 150 0 s100 -50 150 0 z"
          stroke="red" stroke-width="4" fill="none" />

   <path  d="M450 100 q75 50 150 0 q75 -50  150 0 "
          stroke="green" stroke-width="4" fill="none" />
   <path  d="M450 200 q75 50 150 0 t150 0 "
          stroke="yellow" stroke-width="4" fill="none" />
   <path  d="M450 300 q75 50 150 0 t150 0 z"
          stroke="maroon" stroke-width="4" fill="none" />

 </svg>
```

Figura 4.13: Bézier cubiche e quadratiche

### 4.10.11 Elemento *animate*

Con questo elemento è possibile *animare*, cioè variare in modo controllato i valori degli attributi di altri elementi. L'elemento `animate` può essere definito all'interno dell'elemento, un cui suo attributo è da animare, oppure si può fare riferimento dall'esterno. La figura (4.14) mostra come animare alcuni attributi di un cerchio.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
 <svg width="400" height="300" xmlns="http://www.w3.org/2000/svg">
   <title>Cerchi con diversi style</title>
   <desc> Uso di style </desc>

   <!-- Definizione dello stile  -->
   <defs>
     <style type="text/css">
     <![CDATA[
       .Giallo {fill:yellow;}
       .VuotoContornoNero {fill:none; stroke:black;}
       .GialloContornoRossoSpesso
          {fill:yellow; stroke:red; stroke-width:0.2cm;}
       .BluTrasparente {fill:blue; stroke:black; opacity:.0;}
     ]]>
     </style>
   </defs>

   <!-- Quattro cerchi animati-->
   <circle id="c1" cx="100" cy="150" r="0" class="Giallo"/>
   <circle id="c2" cx="170" cy="100" r="70" class="VuotoContornoNero" />
   <circle id="c3" cx="250" cy="150" r="70" class="GialloContornoRossoSpesso"/>
   <circle id="c4" cx="170" cy="200" r="70" class="BluTrasparente"/>
   <animate xlink:href="#c1" attributeName="r"
            begin="2s" dur="15s" from="0" to="70" fill="freeze" />
   <animate xlink:href="#c4" attributeName="opacity"
            begin="2s" dur="15s" from="0" to="1" fill="freeze"/>
   <animate xlink:href="#c3" attributeName="stroke-width"
            begin="2s" dur="15s" from="0.2cm" to="0.0cm" fill="freeze"/>
   <animate xlink:href="#c2" attributeName="fill"
            begin="2s" dur="15s" from="rgb(255,255,255)" to="rgb(0,0,0)"
            fill="freeze"/>
 </svg>
```

Figura 4.14: Animazione di alcuni attributi del cerchio

# Appendice A

# Scalable Vector Graphics

## A.1 Sfera ruotante

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
"http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd" >
<svg>
<defs>
<radialGradient id="myRadGrad" gradientUnits="objectBoundingBox"
cx="50%" cy="50%" r="50%" fx="20%" fy="30%">
<stop offset="10%" stop-color="skyblue"/>
   <stop offset="100%" stop-color="blue"/>
</radialGradient>
</defs>
<g transform='rotate(10)'>
<ellipse fill='url(#myRadGrad)' opacity='.5' stroke='black' stroke-width='.5'
cx='150' cy='150' rx='100' ry='100'/>
<g fill='none' stroke='blue' stroke-width='.1'>
<defs>
<ellipse cx='150' cy='100' rx='85' ry='0'/>
<ellipse cx='150' cy='150' rx='100' ry='0'/>
<ellipse cx='150' cy='200' rx='85' ry='0'/>
</defs>
<path d="M65,100 C85,50 215,50 235,100"/>
<path d="M55,120 C65,100 235,100 245,120"/>
<path d="M50,150 250,150"/>
<path d="M55,180 C65,200 235,200 245,180"/>
<path d="M65,200 C85,250 215,250 235,200"/>
<circle fill='red' stroke='none' cx='0' cy='0' r='3' >
<animateMotion begin='0.5s' dur="2.5s" rotate='auto' fill='freeze' repeatCount="indefinite"
path="M55,120 C65,100 235,100 245,120 " />
<animateColor attributeName='fill' begin='0.5s' dur='5s' attributeType="CSS" repeatCount='indefinite'
values='red;none' calcmode='discrete' />
  </circle>
<path d="M150,50   C17,55 17,245 150,250">
<animate attributeName='d' begin='0s' dur='2.5s' attributeType="XML" fill='freeze' repeatCount='indefinite'
values='M150,50  C17,55 17,245  150,250 ; M150,50  C283,55 283,245  150,250' />
</path>
<path d="M150,50   C17,55 17,245 150,250">
<animate attributeName='d' begin='0.5s' dur='2.5s' attributeType="XML" fill='freeze' repeatCount='indefinite'
values='M150,50  C17,55 17,245  150,250 ; M150,50  C283,55 283,245  150,250' />
</path>
<path d="M150,50   C17,55 17,245 150,250">
<animate attributeName='d' begin='1s' dur='2.5s' attributeType="XML" fill='freeze' repeatCount='indefinite'
values='M150,50  C17,55 17,245  150,250 ; M150,50  C283,55 283,245  150,250' />
</path>
<path d="M150,50   C17,55 17,245 150,250">
<animate attributeName='d' begin='1.5s' dur='2.5s' attributeType="XML" fill='freeze' repeatCount='indefinite'
```

```
values='M150,50  C17,55 17,245  150,250 ; M150,50  C283,55 283,245  150,250' />
</path>
<path d="M150,50   C17,55 17,245 150,250">
<animate attributeName='d' begin='2.0s' dur='2.5s' attributeType="XML" fill='freeze' repeatCount='indefinite'
values='M150,50  C17,55 17,245  150,250 ; M150,50  C283,55 283,245  150,250' />
</path>
</g>
</g>
</svg>
```

## A.2 Orologio

```
<?xml version="1.0" encoding="iso-8859-1"?><!DOCTYPE svg PUBLIC "
-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="300px" height="300px">
  <defs>
    <symbol id="trait" >
      <g style="fill:none; stroke:black; stroke-width:1">
      <line x1="0" y1="85" x2="0" y2="105" />
      </g>
    </symbol>
  </defs>

  <g transform="translate(150,150)">
      <g transform = "rotate(0)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(30)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(60)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(90)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(120)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(150)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(180)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(210)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(240)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(270)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(300)"><use xlink:href="#trait" /></g>
  <g transform = "rotate(330)"><use xlink:href="#trait" /></g>
  </g>

  <g transform="translate(150,150)">
    <g style="fill:red; stroke:black; stroke-width:1">
     <polygon points="-5,-70,0,-85,5,-70,0,20" />
        <animateTransform attributeName="transform" attributeType="XML"
        type="rotate" from="303.308333333" to=" 663.308333333 " begin="0s"
            dur=" 43200s" fill="freeze" />
   </g>
  </g>

  <g transform="translate(150,150)">
    <g style="fill:yellow; stroke:black; stroke-width:1">
     <polygon points="-4,-85,0,-100,4,-85,0,20" />
      <animateTransform attributeName="transform" attributeType="XML"
        type="rotate" from="39.7" to=" 4359.7 " begin="0s"
          dur=" 43200s" fill="freeze" />
   </g>
  </g>


  <g transform="translate(150,150)">
    <g style="fill:green; stroke:black; stroke-width:1">
   <polygon points="-2,-90,0,-105,2,-90,0,20" />
      <animateTransform attributeName="transform" attributeType="XML"
       type="rotate" from="222" to=" 259422 " begin="0s"
          dur=" 43200s" fill="freeze" />
   </g>
    </g>

</svg>
```