Corso di laurea: *Informatica*
Anno accademico: 2013/2014

Insegnamento: Sistemi Multimediali
Prof. Giovanni Dimauro

Caso di studio:

“PNG: il formato definitivo?”

Autore: Gianvito Taneburgo
Matricola: 587645

Indice

Introduzione

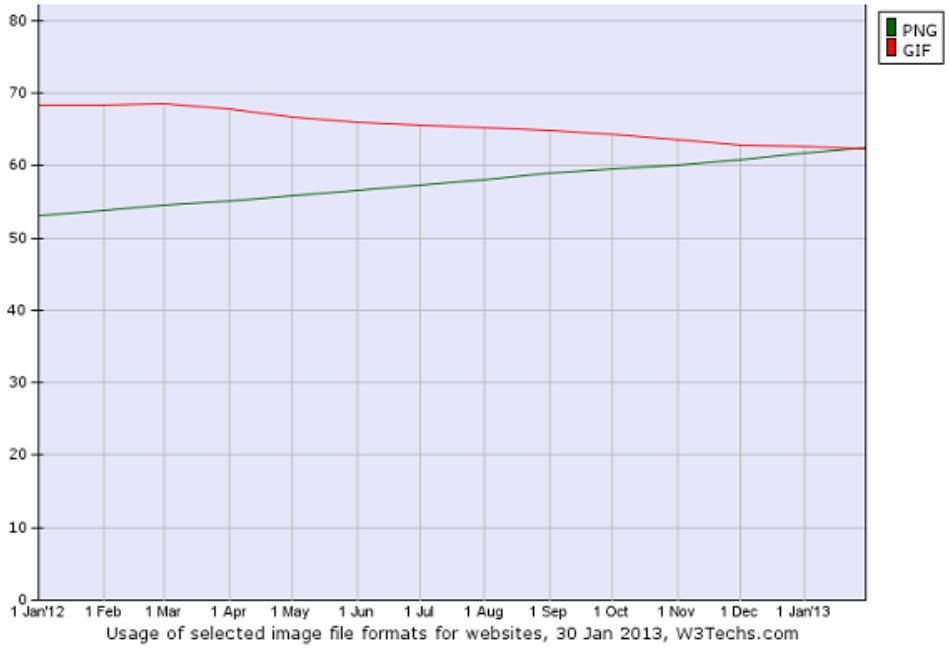
PNG (pronuncia originale: “ping”[1]) è un formato di file utilizzato per la memorizzazione di immagini. È difficile incontrare oggigiorno utilizzatori di prodotti digitali che non lo conoscano ed è ancora più complesso trovare computer o dispositivi che non lo supportino nativamente.
PNG è diventato *de facto* uno degli standard che più si è consolidato nelle ultime due decadi (il paragrafo “*Statistiche e dati*” proverà a quantificarne la popolarità).
Da questo innegabile successo nasce spontanea, o almeno così è stato per chi Vi scrive, la domanda: “PNG è il formato definitivo?”.
La seguente relazione si pone come obiettivo quello di illustrare le caratteristiche tecniche del formato e, contestualmente, le motivazioni che ne hanno decretato il successo. Si cercherà di mantenere un punto di vista il più possibile imparziale, ove possibile scientifico, per confrontare PNG con gli altri più celebri formati utilizzati ai giorni nostri. Alla fine delle relazione si proverà a dare una risposta al quesito.
La trattazione seguirà inizialmente la regola giornalistica anglosassone delle “*5 W*”, per provare a riassumere in modo efficace le informazioni principali riguardanti il *background* del formato e per permettere al Lettore di “familiarizzare” con l’argomento. Le caratteristiche peculiari del formato saranno successivamente affrontate singolarmente.
È presente, sul finire della relazione, il commento al codice del programma “*PNG Manager*”. Si tratta di un semplice *decoder* di immagini PNG, arricchito con qualche funzione non presente nei classici visualizzatori di immagini. Per ulteriori dettagli sul codice, comunque, si rimanda alla *Javadoc* allegata.

Five Ws

La storia del PNG affonda le sue radici nelle travagliate vicende di un altro diffusissimo formato per la memorizzazione delle immagini, il GIF[2]. Questo fu proposto nel 1987 da *CompuServe* come formato capace di trattare immagini a colori e con animazioni, in contrapposizione al formato RLE che permetteva di memorizzare immagini esclusivamente in bianco e nero. Con la diffusione del *World Wide Web* il formato GIF conobbe il successo, affermandosi presto insieme al JPEG[3]. Nel gennaio 1993 *Unisys*, proprietaria (sin dal 1983) del brevetto sull'algoritmo di compressione *LZW*, si rese conto che *CompuServe*, ormai già da diversi anni, stava utilizzando quell'algoritmo per comprimere i dati nelle immagini GIF, senza sapere dell'esistenza del brevetto (o almeno così fu dichiarato dagli accusati). Le due società iniziarono a negoziare per cercare un accordo che fu raggiunto alla fine del 1994: *Unisys* dichiarò che tutte le grandi compagnie che offrivano servizi online facenti uso dell'algoritmo *LZW* avrebbero dovuto pagarla a causa del brevetto, esonerando dalla tassa solo le piccole aziende *non-profit*. Le conseguenze non furono difficili da prevedere: *CompuServe* ed *Unisys* furono disprezzate[4] e condannate in tutto il mondo e le aziende minacciarono di abbandonare GIF. Proprio qui ha inizio la storia del PNG. Nel 1995 il formato venne progettato con due espliciti obiettivi: versatilità e gratuità, per fornire un'alternativa robusta a GIF. Quest'ultimo, inoltre, manifestava alcuni altri problemi che PNG si propose di risolvere; uno di primaria importanza era il suo ragguardevole limite di 256 colori memorizzabili, in un momento storico in cui ormai iniziavano a comparire supporti fisici capaci di mostrarne molti di più.
Una discussione sul *newsgroup Usenet* del gennaio 1995 dal titolo "*Thoughts on a GIF-replacement file format*"[5] ebbe un enorme seguito e permise di raccogliere molte utili idee che sarebbero state implementate successivamente in PNG. Le specifiche furono formalizzate e pubblicate da un gruppo di grafici, matematici, informatici e fan di tutto il mondo; tra questi ricordiamo i principali: *Thomas Boutell* (scrittore della *GD Graphics Library*), *Mark Adler* (co-autore delle librerie *gzip* e *zlib*[6]), *Tom Lane* (sviluppatore di *PostgreSQL* e scrittore di librerie per JPEG), *Lee Daniel Crocker* (programmatore del *core* di *MediaWiki*, su cui tuttora si basa *Wikipedia*). *Oliver Fromme* propose per il formato il nome "*PING is not GIF*", da cui l'acronimo ricorsivo PNG (l'acronimo ufficiale è *Portable Network Graphics*). Fu appoggiata la scelta progettuale di non supportare, a differenza del GIF, brevi animazioni, al fine di non sovraccaricare il formato[7]. Formati gemelli se ne assunsero l'incarico: nacquero così il *Multiple-image Network Graphics*[8] (MNG) e l'*Animated Portable Network Graphics*[9] (APNG). Non sarà, tuttavia, oggetto della relazione la trattazione di questi formati.
Il 1° ottobre 1996 la versione 1.0 delle specifiche fu approvata dal *W3C*[10]; le successive modifiche del 1998, del 1999 e del 2003 aggiunsero solo alcuni dettagli a quello che sembrava, già dalla prima versione, un progetto saldamente fondato e con profonda anticipazione al cambiamento.
Nel 2004 sono scaduti i brevetti sull'*LZW*, rendendo GIF libero[11] (il giorno viene ricordato come "*GIF Liberation Day*"), ma ormai PNG fa da padrone nel mondo dell'*IT.*

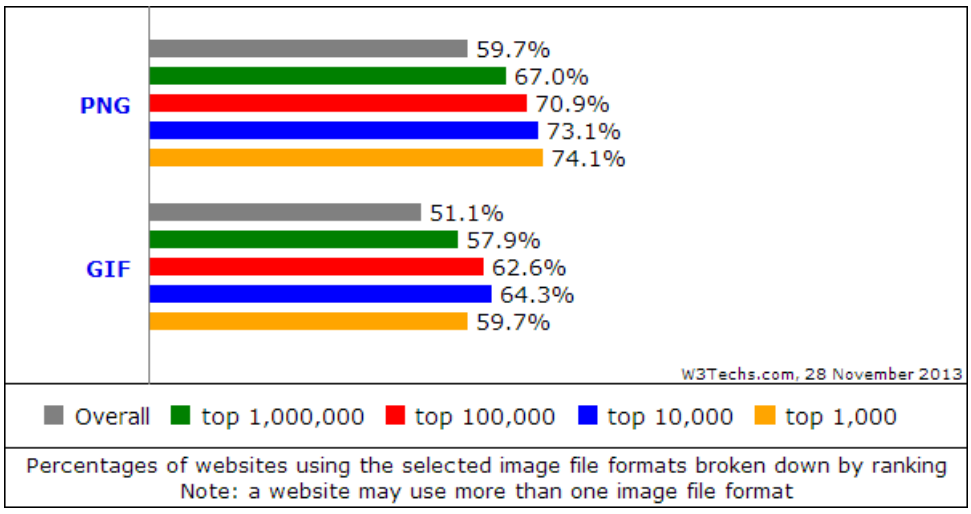
Statistiche e dati

A favorire la diffusione del formato PNG nei primi anni furono certamente il *tool* "*gif2png*" (la cui funzione viene lasciata supporre al Lettore) e la popolare campagna "*Burn All GIFs*"[12] nata per contrastare le *royalties* verso *Unisys*.
Linux supporta nativamente il PNG dal 1999; *Microsoft* iniziò a farlo con *Windows Vista*.
Le immagini PNG sono visualizzabili anche su *Mac OS, iOS*, *Android* e tutti gli altri principali sistemi operativi.
I primi *browser* a supportare il PNG furono *Internet Explorer* 4.0b1 e *Netscape* 4.04. Nonostante il PNG fosse stato approvato dal *W3C*, la diffusione del formato fu lenta a causa di alcuni *bug* in *Internet Explorer,* allora *browser* dominante, nella visualizzazione delle immagini trasparenti[13] (i *bug* sono presenti, invero, fino alla versione 8). Col passare degli anni la situazione è migliorata gradualmente: dalla versione 6 è stato introdotto il supporto all'*alpha channel*, responsabile della trasparenza, e dalla versione 8 viene correttamente gestita la *gamma correction*.
Attualmente PNG è supportato da tutti i *browser*.
W3Techs ha annunciato[14] che dal 31 gennaio 2013 PNG ha superato GIF per utilizzo nei siti *web*, con il 62.4% d'impiego del primo formato contro il 62.3% del secondo. Come si evince dal grafico, all'inizio del 2012 GIF aveva un margine di vantaggio del 15%. L'unico formato più utilizzato è attualmente il JPEG. I problemi legati ai brevetti sul GIF sono oggigiorno stati risolti, ma il PNG viene preferito per la sua superiorità tecnica che ha convinto i *webmaster* ad impiegarlo[15] (spesso le immagini GIF hanno dimensioni superiori rispetto a quelle PNG, che supporta inoltre una più ampia gamma di profondità di colore).



Usage of selected image file formats for websites, 30 Jan 2013, W3Techs.com

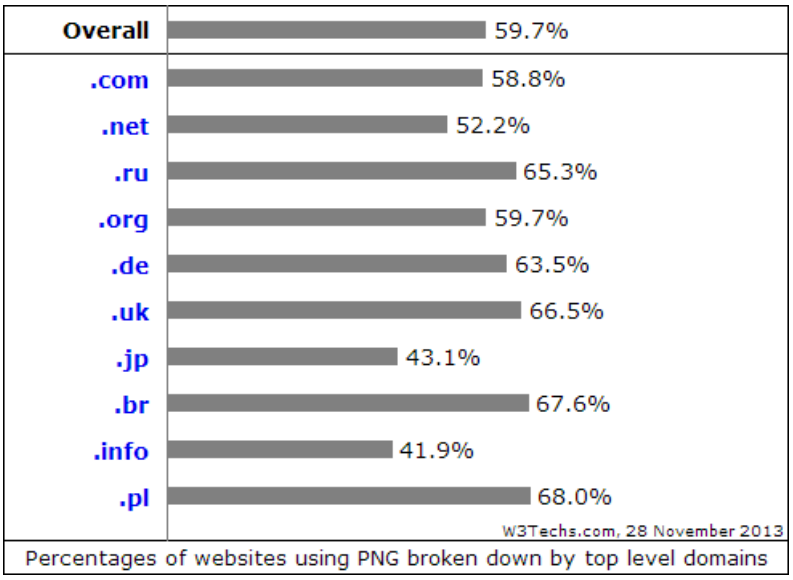
E' interessante notare come GIF fosse più popolare di PNG nei primi 1000 siti web all'inizio del 2013, ma, come avevano predetto gli esperti, il dato ora è cambiato. Il trend lo lasciava immaginare: per ogni sito che passava da PNG a GIF ce n'erano altri 3 che effettuavano la scelta in direzione contraria.



Percentages of websites using the selected image file formats broken down by ranking
Note: a website may use more than one image file format

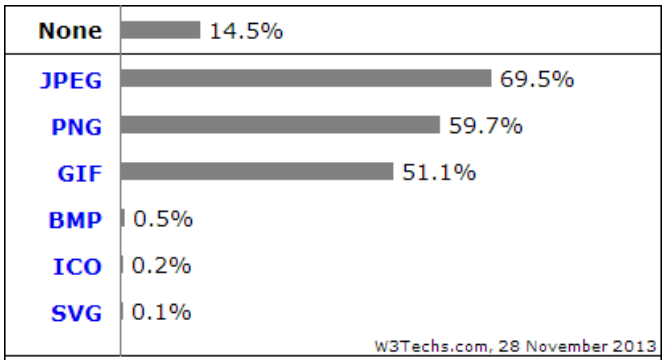
Vi sono inoltre significative differenze geografiche[16] nel tasso di utilizzo di PNG: è molto popolare in Europa, con oltre il 70% in Francia, Italia, Spagna, Olanda, ma molto meno in Asia, con il 43.1% in Giappone, il 34.6% in Korea e solo il 30.6% in Cina. L'utilizzo è anche maggiore sui sistemi *Unix* rispetto a quelli *Windows*.

Un ultimo dato, forse più significativo di tutti gli altri, potrà fornirVi una percezione più veritiera della attuale situazione: *Google* sembra essere un forte sostenitore del PNG, poiché ne fa uso sul 95.1% dei suoi *server*[17].

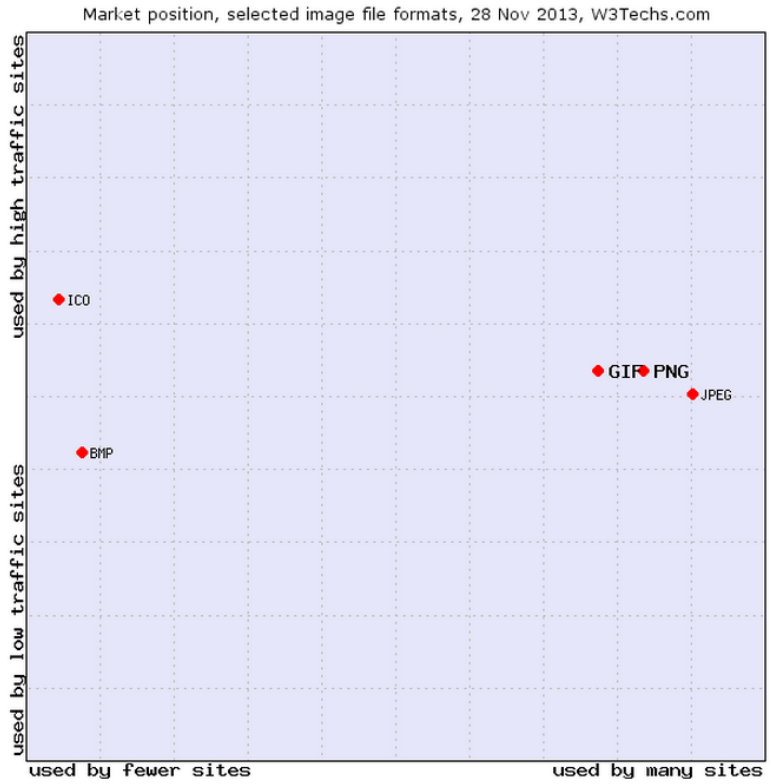


Percentages of websites using PNG broken down by top level domains

GIF è stato leader negli ultimi 25 anni, ma la realtà è che oggi è stato "spodestato".



Percentages of websites using various image file formats
Note: a website may use more than one image file format



Accenni sulle caratteristiche tecniche

Come è possibile leggere sin dalla prima versione delle specifiche *W3C* del PNG[18], "il formato PNG fornisce uno standard portabile, libero, con un buon livello di compressione e ben strutturato per file immagini bitmap". Nato per sostituire GIF, il PNG, per merito della sua attenta progettazione, include molte caratteristiche utili non disponibili nel precedente formato.

Tra le caratteristiche che PNG eredita da GIF, possiamo citare le seguenti:

- Indipendenza completa dall'*hardware* e dalla piattaforma correntemente in uso;
- Capacità di inserire metadati (commenti, annotazioni, ecc.) direttamente nell'immagine;
- Compressione effettiva senza assoluta perdita;
- Visualizzazione progressiva (interlacciamento): immagini in formato PNG, adeguatamente preparate, possono essere visualizzate progressivamente, mostrandosi prima in bassa risoluzione e, gradualmente, aumentando il livello di dettaglio;
- Supporto per immagini con indice di colori (fino a 256);
- *Streamability*: le immagini sono assimilabili a dei flussi, dai quali è possibile leggere contenuto o scriverlo. In questo modo il formato può essere usato come un protocollo di comunicazione per la generazione e visualizzazione *on-the-fly* delle immagini.

Come detto, tuttavia, ve ne sono di nuove. Queste sono le principali:

- Supporto per immagini *truecolor* (fino a 48 bit per pixel);
- Supporto per immagini in scala di grigio (fino a 16 bit per pixel);
- Trasparenza (*alpha channel*): *pixel* dell'immagine possono essere trasparenti. Se ad essere trasparenti sono le parti più esterne l'utente vedrà immagini di forma non rettangolare;
- Informazioni sulla gamma, che rende possibile, in modo automatico, la visualizzazione delle immagini con luminosità e contrasto esatti, prescindendo dalla macchina sulla quale sta venendo mostrata l'immagine;
- Introduzione di un modo affidabile e diretto per rilevare file corrotti che garantisce al formato robustezza;
- Visualizzazione progressiva più veloce del GIF;
- Compressione efficace di immagini (in molti casi migliore di tanti altri formati – vedere il paragrafo "*Confronto con altri formati*" per informazioni in merito).

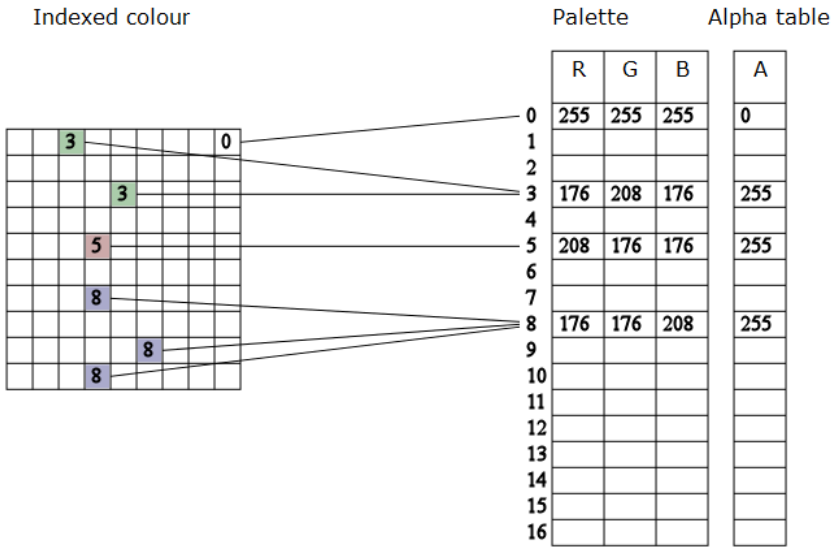
Numerosi sforzi sono stati fatti per verificare che tutti gli algoritmi usati nel formato non infrangessero alcun brevetto: il risultato è che PNG è *free* al 100%.
Il formato è stato concepito per essere flessibile, poiché permette di integrare future estensioni in modo retro-compatibile e senza grossi sforzi (tuttora alcuni bit sono riservati e vengono mantenuti tali per eventuali cambiamenti). Infine, il formato è interscambiabile: ogni *decoder* conforme agli standard è in grado di visualizzare qualunque tipo di file PNG.

Rappresentazione dell'immagine e dei dati

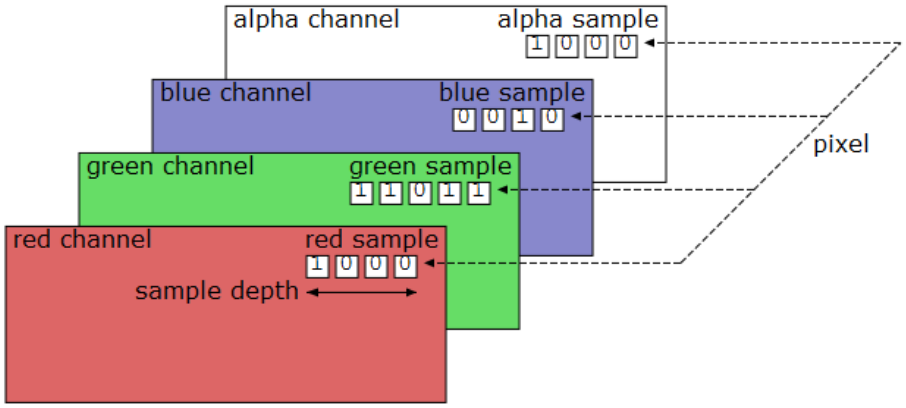
Una immagine PNG è un *array* di *pixel* rettangolari, che compaiono da sinistra a destra per ogni *scanline* (riga), dall'alto verso il basso[19]. La dimensione di ogni *pixel* è determinata dalla *bit depth* (profondità), che rappresenta il numero di bit per ogni campione.

Tre tipologie di pixel sono supportati:

- *Indexed-colour* (indice ad un colore): è rappresentato da un unico campione che fa riferimento ad una *palette* (tavolozza). La *bit depth* determina il numero massimo di *entry* (voci) presenti nella *palette*;
- *Grayscale* (scala di grigio): è rappresentato da un unico campione che è un livello della scala, ove 0 rappresenta il nero mentre il valore pari alla *bit depth* rappresenta il bianco;
- *Truecolor* (milioni di colori): è rappresentato da tre campioni (rosso, verde, blu). Ciascuno varia tra 0 (nero) e un valore massimo. La *bit depth* specifica la dimensione di ogni campione e non la dimensione totale del *pixel*.



Struttura di un'immagine con indici a colori di una tavolozza.



Rappresentazione di un pixel con i canali RGBA.

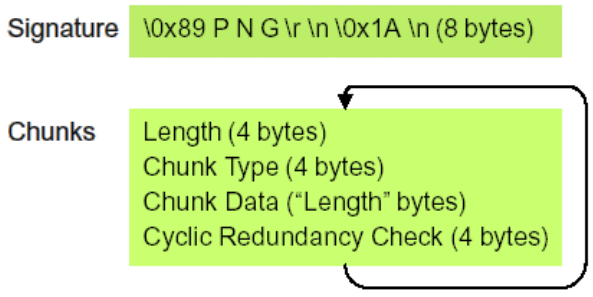
I *pixel* vengono impacchettati in *scanline*, senza lasciare spazio tra uno e l'altro.
Quando i *pixel* hanno meno di 8 bit e l'ampiezza della *scanline* non è divisibile per il numero di *pixel* per byte allora gli ultimi bit dell'ultimo byte di ogni *scanline* vengono scartati: il loro contenuto non è predeterminato.
Un tipo di filtro è aggiunto all'inizio di ogni *scanline*. Tale filtro non fa parte dei dati dell'immagine ma viene incluso nel flusso di dati inviati al compressore. Tale filtro permette di massimizzare la compressione dell'immagine (per ulteriori dettagli si rimanda al paragrafo "*Filtraggio*").
Grayscale e *truecolor* possono include un campione *alpha* per la trasparenza. Un valore di *alpha* uguale a 0 rappresenta un *pixel* completamente trasparente, un valore pari a $2^{bit\ depth}$-1 un *pixel* totalmente opaco, valori intermedi *pixel* parzialmente trasparenti, che lasceranno intravedere lo sfondo. L'*alpha channel*, di conseguenza, è portatore di informazioni sulla opacità del *pixel*, e non della trasparenza come si è solito credere (una sottile ma determinante differenza se doveste trovarVi ad impostare manualmente il valore dell'*alpha channel* per un'immagine). L'*alpha channel* può essere incluso in immagini che hanno 8 o 16 bit per campione. I campioni *alpha* sono rappresentati con la stessa *bit depth* e sono memorizzati immediatamente dopo il *grayscale* o il campione RGB del *pixel*. I valori dei campioni dei colori di un *pixel* sono memorizzati senza tener conto del grado di trasparenza dello stesso; questa regola viene detta *non-premultiplied alpha* (in PNG, dunque, i colori non sono premoltiplicati per la frazione espressa dal canale *alpha*).
E' possibile specificare una trasparenza senza memorizzare un ulteriore canale: in una immagine con indici a colori, un valore di *alpha* può essere ricavato da un'*entry* della *palette*; in una immagine in scala di grigio o *truecolor*, un singolo *pixel* può essere identificato come "trasparente". Queste tecniche sono gestite tramite l'utilizzo del *chunk tRNS* (la spiegazione viene rimandata al paragrafo che affronta la costituzione dei *chunk*).

Struttura del file

Un file in formato PNG è costituito da una *signature* e da una lista di *chunk*. La *signature* è comune a tutte le immagini, è lunga 8 byte ed i suoi valori decimali sono: *137 80 78 71 13 10 26 10*.
Di seguito viene riportata una tabella per spiegare il significato di tali byte:

Byte	Finalità
137	*Setta il più alto bit ad 1 per rilevare sistemi di trasmissione che non supportano dati ad 8 bit. Riduce, inoltre, la possibilità che un file di testo venga interpretato come un'immagine PNG e viceversa.*
80 78 71	*Rappresentano, in ASCII, le lettere 'P', 'N', 'G', per permettere ad una persona di riconoscere il formato qualora il file venisse aperto in un editor di testi.*
13 10	*Ritorno a capo in stile DOS (CRLF), per rilevare conversioni nei dati dei ritorni a capo da DOS a Unix.*
26	*Interrompe la visualizzazione del file nelle finestre DOS se il comando "type" è stato utilizzato (carattere EOF).*
10	*Ritorno a capo in stile Unix (LF), per rilevare conversioni nei dati dei ritorni a capo da Unix a DOS.*

I *chunk* sono assimilabili a frammenti; ogni *chunk* è portatore di alcune informazioni sull'immagine. Un file PNG contiene, subito dopo la *signature*, una lista, arbitrariamente lunga, di *chunk*.



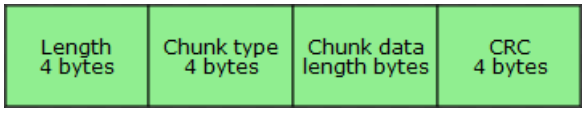
Questi sono scritti nel file consecutivamente (alla stregua dei *pixel* nella *scanline*), senza lasciare spazi tra essi, seguendo alcune convenzioni. Il successivo paragrafo analizzerà la struttura di questi componenti ed alcune scelte progettuali adottate dal formato PNG.

Chunk: composizione, finalità, caratteristiche

I *chunk* rappresentano il "*core*" del PNG[20]. Una parte del successo del formato è riconducibile innegabilmente ad essi. L'idea alla base dei *chunk* è in perfetta armonia con molti dei principi cardine di progettazione, in modo particolare con quello dell'anticipazione al cambiamento. Per comprendere meglio quanto detto è necessario, innanzitutto, illustrare la composizione dei *chunk*.

Ogni *chunk* è costituito da 4 campi:

- Lunghezza: intero senza segno di 4 byte che esplicita la lunghezza del solo campo dati del *chunk* (esclude dal conteggio, perciò, sé stesso, il campo tipo ed il campo CRC). Zero è un valore possibile. Il suo valore non può superare, in ogni caso, 2^{31}-1.
- Tipo: codice di 4 byte che descrive la tipologia del *chunk*. Spesso tali byte sono interpretabili come una stringa di 4 lettere, e perciò i byte assumono i valori *ASCII* delle corrispondenti lettere (A-Z ed a-z, ovvero 65-90 e 97-122 in decimale).
- Dati: byte associati al *chunk* in esame. Tale parte può anche essere di lunghezza nulla.
- CRC: 4 byte avvalorati col *Cyclic Redundancy Check* del *chunk*[21]. Il CRC viene calcolato su tutti i byte dei campi del tipo e dei dati. Il CRC è sempre presente, anche nei *chunk* privi di dati. Il CRC permette di controllare l'integrità del *chunk* e, di riflesso, anche quella dell'immagine (come era stato precedentemente detto nel paragrafo "*Accenni sulle caratteristiche tecniche*"). Brevi chiarimenti sul calcolo del CRC sono esplicitati nel paragrafo "*Approfondimenti sul formato*". Un *decoder* che calcola un CRC differente da quello presente in questo campo, può, ad esempio, decidere di notificare all'utente un errore di integrità dell'immagine ed interrompere l'analisi del flusso di byte.



In una immagine è consentita la presenza di un solo *chunk* per ogni tipo, eccezion fatta per alcuni tipi esplicitamente permessi dalle specifiche del formato.
Interpretando i byte del tipo come una stringa, allora è possibile affermare che i tipi sono *case-sensitive*.
I nomi dei tipi di *chunk* sono assegnati in modo da permettere ad un *decoder* di determinare aprioristicamente alcune proprietà del *chunk* stesso, anche quando il tipo non è tra quelli a lui conosciuti. Queste regole permettono l'esistenza di estensioni sicure e flessibili del formato PNG. Un *decoder*, infatti, analizzando tali proprietà potrà, in presenza di tipi sconosciuti, assumere un comportamento precedentemente determinato (ad esempio, ignorare il *chunk* o notificare un errore all'utente).
Il quinto bit di ciascuno dei 4 byte costituenti il tipo del *chunk* è utilizzato per trasmettere una proprietà. La scelta ricade proprio sul quinto bit poiché è quello che determina se la lettera, ottenuta dall'interpretazione del byte corrente in notazione ASCII, sarà maiuscola o minuscola. In tal modo, un essere umano può immediatamente riconoscere se la 1ª, 2ª, 3ª o 4ª proprietà è vera per il *chunk* in esame semplicemente guardando se è maiuscola rispettivamente la 1ª, 2ª, 3ª o 4ª lettera del tipo. Un esempio chiarirà al lettore questo semplice ma astuto meccanismo. Si supponga di avere un *chunk* il cui tipo, interpretato come stringa, sia *exAM*. Un *decoder*, imbattendosi in tale

chunk, saprà che la 3ª e la 4ª proprietà sono vere, poiché la 3ª e la 4ª lettera del tipo ('A' ed 'M') sono maiuscole (1ª e 2ª proprietà sono false). Ma quali sono queste proprietà? La tabella seguente si propone di spiegarle al lettore.

Posizione nel tipo del *chunk*	Valore del bit (0 = maiuscolo, 1 = minuscolo) e semantica
5° bit del 1° byte	*"Ancillary" (0 = critico, 1 = ausiliario).* *I chunk non strettamente necessari al fine di mostrare significativamente il contenuto del file sono impostati ad ausiliari. Un decoder che incontra un chunk ausiliario (a lui conosciuto oppure no) può ignorarlo e riuscire comunque a visualizzare l'immagine correttamente.* *I chunk critici sono quelli strettamente necessari per mostrare l'immagine. Un decoder che incontra un chunk critico a lui sconosciuto deve notificare un errore all'utente (se è conosciuto saprà come gestirlo).*
5° bit del 2° byte	*"Private" (0 = pubblico, 1 = privato).* *Un chunk pubblico è uno che appartiene alle specifiche del PNG oppure è registrato tra i chunk pubblici special-purpose; viceversa è privato.* *Un decoder non è interessato a tale proprietà: è solo una comodità burocratica per assicurarsi che tipi pubblici e privati non collidano.*
5° bit del 3° byte	*"Reserved" (obbligatoriamente 0).* *Il significato di tale proprietà è riservato per possibili usi futuri. Viene impostato a 0 in modo da garantire, a causa di una eventuale estensione, che i decoder implementino il supporto alla nuova feature: un domani, comunque, potrebbe essere avvalorato 1 e venire ignorato dai decoder.*
5° bit del 4° byte	*"Safe-to-copy" (0 = insicuro da copiare, 1 = sicuro da copiare).* *Questa proprietà non è di interesse dei decoder, ma degli editor e descrive come gestire i chunk ausiliari in un file che ha subito modifiche.* *Un chunk sicuro da copiare può essere ricopiato nel nuovo file PNG in output anche se l'encoder non ne ha riconosciuto il tipo, prescindendo dalle modifiche che l'immagine ha subito.* *Un chunk è insicuro da copiare se il suo contenuto dipende dai dati dell'immagine. Se un programma effettua cambiamenti su chunk critici (aggiunte, modifiche, rimozioni o riordinamenti) allora tutti i chunk insicuri da copiare che non sono riconosciuti dall'editor non devono essere copiati nel file di output (se invece i chunk insicuri sono riconosciuti allora il programma può procedere ad aggiornarli in modo appropriato e quindi a copiarli nel file di output). I chunk critici appartengono sempre a questa categoria.* *N.B.: Chunk ausiliari non possono dipendere da chunk ausiliari. Per questo motivo un editor può sempre ricopiare chunk non riconosciuti se le modifiche hanno riguardato chunk ausiliari.*

Riconsiderando per un momento l'esempio precedente, il *chunk* con tipo *exAM* è ausiliario, privato ed insicuro da copiare. Mentre *bLOb* è un *chunk* ausiliario, pubblico e sicuro da copiare.

E' interessante notare come le proprietà dei bit siano parte del nome, perciò sono fisse per qualunque tipo di *chunk.* Di conseguenza, *TEXT* e *Text* sarebbero due tipi di *chunk* indipendenti, e non lo stesso *chunk* con differenti proprietà.
In virtù alle caratteristiche dei *chunk*, è possibile abbinare ad una immagine metadati di qualunque natura (testo, audio, ecc.) senza ostacolare un decoder, che potrà ignorare i *chunk* poiché saranno ausiliari. Non a caso, diversi celebri *software* per *l'editing* di immagini sono soliti inserire diversi metadati (potreste trovare, ad esempio, *chunk* di tipo *tEXt* contenenti, nel campo dati, la stringa "*Created with Photoshop*").

Di seguito verranno presentati alcuni dei principali *chunk*, critici ed ausiliari, per i quali PNG ha specificato la costituzione. Per ulteriori approfondimenti si rimanda alla specifiche del PNG, ove è possibile trovare maggiori dettagli.

Una immagine PNG corretta dovrebbe contenere, dopo la *signature*, un *chunk IHDR*, uno o più *chunk IDAT* e concludersi con un *chunk IEND.* Cominciamo la rassegna proprio da questi *chunk*.

a. Chunk IHDR

Il *chunk* critico *IHDR* rappresenta le caratteristiche dell'immagine e contiene, consecutivamente, degli interi senza segno rappresentanti la larghezza dell'immagine (4 byte), l'altezza dell'immagine (4 byte), la *bit depth* (1 byte), il tipo di colore (1 byte), il metodo di compressione (1 byte), il metodo di filtraggio (1 byte) ed il metodo di interlacciamento (1 byte).
Larghezza e altezza non posso essere nulle. La *bit depth* esplicita il numero di bit per campione o per un indice della *palette* (non per *pixel*); valori validi sono 1, 2, 4, 8 e 16, sebbene solo alcune combinazioni con il tipo di colore siano consentite. Il tipo di colore definisce la natura dell'immagine PNG; valori validi sono 0, 2, 3, 4 e 6.
Come detto, sono poste alcune restrizioni sulle combinazioni tra tipi di colore e *bit depth*[22]. La tabella seguente riporta le combinazioni valide e le loro interpretazioni.

Tipo di immagine PNG	Tipo di colore	*Bit depth* consentite	Interpretazione
Scala di grigio	*0*	*1, 2, 4, 8, 16*	*Ogni pixel è un campione di una scala di grigio.*
Truecolor	*2*	*8, 16*	*Ogni pixel è una tripletta R, G, B.*
Indice a colore	*3*	*1, 2, 4, 8*	*Ogni pixel è l'indice di una tavolozza. Un chunk PLTE deve essere presente.*
Scala di grigio con alpha	*4*	*8, 16*	*Ogni pixel è un campione di una scala di grigio seguito da un campione alpha.*
Truecolor con alpha	*6*	*8, 16*	*Ogni pixel è una tripletta R, G, B seguita da un campione alpha.*

Il metodo di compressione indica come sono stati compressi i dati dell'immagine. Attualmente è definito solo un metodo, contrassegnato col numero 0, che corrisponde all'algoritmo *deflate/inflate*[23] con finestra scorrevole (al più 32768 byte). Similmente, il metodo di filtraggio indica come sono stati preprocessati i dati dell'immagine, prima di essere passati come input al compressore. Al momento è presente solo un tipo di filtro, contrassegnato anch'esso col numero 0, ovvero il filtro adattivo basata su cinque filtri elementari. Infine, il metodo di interlacciamento designa l'ordine di trasmissione dei dati dell'immagine: valore 0 significa che l'immagine non ha interlacciamento, valore 1 che è stata interlacciata con l'algoritmo Adam7.

b. Chunk PLTE

Il *chunk* critico *PLTE* contiene tutti i colori della *palette* necessari al *decoder* per effettuare la corrispondenza con gli indici. Contiene da 1 a 256 voci, ciascuna costituita da una serie di 3 byte (un byte per il rosso, uno per il verde, uno per il blu). Il numero di voci nella tavolozza è determinato dalla lunghezza del *chunk*, che perciò deve essere necessariamente un multiplo di 3. Questo *chunk* è presente obbligatoriamente nelle immagini con tipo di colore pari a 3; può comparire in quelle con tipo 2 e 6; non deve assolutamente comparire per il tipo 0 e 4. Possono esserci più *chunk PLTE*. Alle voci della *palette* vengono assegnati valori crescenti, a partire da 0 (ogni voce ha grandezza costante uguale ad un byte, anche nelle immagini *truecolor*). Non è richiesto che tutti i colori della tavolozza siano effettivamente utilizzati dall'immagine, né che siano tutti diversi tra loro. Nel caso di immagini con tipo di colore pari a 2 o 6, la *palette* viene fornita per suggerire un insieme di colori con i quali l'immagine *truecolor* può essere quantizzata se il *decoder* non riesce a mostrarla correttamente (si consiglia, tuttavia, l'utilizzo del *chunk sPLT* in queste situazioni).

c. Chunk IDAT

Il *chunk* critico *IDAT* contiene i dati dell'immagine, ottenuti in seguito all'applicazione dell'algoritmo di compressione su di essa. E' possibile che vi siano più *chunk IDAT*; in questo caso essi devono apparire consecutivamente, senza che nessun altro tipo di *chunk* si frapponga. Concatenando il contenuto di tutti i *chunk IDAT* si ottiene un *array* di byte da fornire come input al decompressore, al fine di ottenere i dati originali dell'immagine.

d. Chunk IEND

Il *chunk* critico *IEND* segnala la fine del file PNG. E' necessariamente l'ultimo *chunk* della sequenza ed il suo campo dati è vuoto.

e. Chunk tRNS

Il *chunk* ausiliario *tRNS* gestisce la trasparenza e, perciò, non si trova nelle immagini con tipo di colore 4 e 6, in quanto esse possiedono già il canale *alpha* che specifica la trasparenza dei *pixel*. Per le immagini con tipo di colore 3, il *chunk* contiene una serie di valori *alpha* di un byte,

corrispondenti alle voci della tavolozza, presenti nel *chunk PLTE*, precedentemente illustrato. I valori sono trattati come un canale *alpha* ad 8 bit: 0 significa completa trasparenza, 255 completa opacità, prescindendo dalla *bit depth* dell'immagine. Se il *chunk* contiene un numero di trasparenze inferiore al numero di *entry* nella *palette* allora tutte le mancanti si assumono completamente opache. Per le immagini con tipo di colore 0 o 2, vengono utilizzati 2 byte: i *pixel* con valore del campione di grigio o RGB pari a quello specificato sono trattati come trasparenti, gli altri opachi.

f. Chunk gAMA

Il *chunk* ausiliario *gAMA* fornisce informazioni addizionali sullo spazio di colori; in particolare, specifica la relazione tra i campioni dell'immagine e l'intensità di visualizzazione desiderata in output. La gamma è una trasformazione non lineare usata per codificare e decodificare la luminanza (grandezza fotometrica che esprime la proiezione dell'intensità luminosa della sorgente luminosa su una superficie)[24]. La codifica gamma aiuta a mantenere i dati digitali in un dominio quanto più possibile uniforme dal punto di vista percettivo. Il *chunk* contiene un intero senza segno di 4 byte, che, moltiplicato per 10000, indica il valore della gamma (ad esempio, per 1/2.2 il *chunk* conterrà il valore 45455). Bisogna ammettere, tuttavia, che anche ignorando il valore di gamma (ignorando perciò la relazione $campione\_immagine = output\_visualizzato^{gamma}$) l'immagine risulta sufficientemente nitida (la differenza risulta percepibile ad un occhio attento).

g. Chunk sRGB

Il *chunk* ausiliario *sRGB*, come *gAMA*, è portatore di informazioni sullo spazio di colori. Se questo *chunk* è presente allora i campioni dell'immagine sono conformi allo spazio di colori sRGB: il *decoder* dovrà utilizzare il *renderer* specificato dall'*International Color Consortium*[25]. Il *chunk* contiene un solo byte che esprime l'intento di rappresentazione. I valori che tale byte può assumere sono: 0 (per immagini in cui si preferisce un buon adattamento alla gamma d'output del dispositivo, a costo dell'accuratezza colorimetrica – es: foto), 1 (per immagini che richiedono comparsa di abbinamento di colori rispetto al bianco del dispositivo – es: loghi), 2 (per immagini che mirano a preservare la saturazione a costo della tinta e della luminosità – es: grafici e diagrammi) e 3 (per immagini che richiedono la conservazione della colorimetria assoluta – es: anteprime di immagini destinate ad altri dispositivi di output).

Come detto precedentemente, si possono abbinare molti metadati ad una immagine PNG. Il formato offre almeno 3 *chunk* per memorizzare stringhe, quali descrizione dell'immagine, avvisi di *copyright*, ecc.[26] All'interno di tali *chunk* è sempre presente una *keyword* che, per l'appunto, ne indica la natura. Questi *chunk* possono essere presenti in qualsiasi quantità ed è anche consentita la presenza di più *chunk* caratterizzati con la medesima *keyword*.
PNG definisce alcune *keyword* predefinite, che dovrebbero essere usate quando appropriato. Altre *keyword* possono essere ideate per nuovi fini e, se di interesse generale, essere registrate dalla *PNG Registration Authority*. E' permesso utilizzare *keyword* non registrate (si auspica abbiano un titolo

auto-esplicativo), purché utilizzino esclusivamente caratteri latini (con valori decimali nei seguenti range: 32-126 e 161-255).
Di seguito viene riportata una tabella riassuntiva delle *keyword* registrate:

Keyword	Significato
Author	*Nome del creatore dell'immagine*
Description	*Descrizione dell'immagine, possibilmente dettagliata*
Copyright	*Avvisi di copyright*
Creation Time	*Ora di creazione dell'immagine originale*
Software	*Software utilizzato per la creazione dell'immagine*
Disclaimer	*Dichiarazione legale di limitazione di responsabilità*
Warning	*Avvisi sulla natura del contenuto*
Source	*Dispositivo utilizzato per la creazione dell'immagine*
Comment	*Commenti di varia natura*

h. Chunk tEXt

Il *chunk* ausiliario *tEXt* è il primo di quelli che offrono la possibilità di allegare metadati testuali all'immagine. E' costituito dalla *keyword* (da 1 a 79 byte), il carattere *null* (1 byte) che funge da separatore e la stringa. Né la keyword, né la stringa possono contenere il carattere *null*. La stringa può avere lunghezza compresa tra 0 e quella specificata dal *chunk*.

i. Chunk zTXt

Il *chunk* ausiliario z*TXt* è simile al precedente, ma è consigliato per memorizzare testi più lunghi. E' costituito dalla keyword (da 1 a 79 byte), il carattere *null* (1 byte) che funge da separatore, il metodo di compressione (1 byte) e il flusso di dati della stringa compresso. La *keyword* non è compressa. L'unico metodo di compressione attualmente definito è 0 (*deflate/inflate*). La lettera *z* del nome del *chunk* è un riferimento a *zlib*.

j. Chunk iTXt

Il *chunk* ausiliario *iTXt* è ideato per testi internazionali. La struttura dati del *chunk* è la seguente: *keyword* (da 1 a 79 byte), carattere separatore *null* (1 byte), *flag* di compressione (1 byte), metodo di compressione (1 byte), tag della lingua (1 byte), carattere separatore *null* (1 byte), traduzione della *keyword* (0 o più byte), carattere separatore *null* (1 byte), testo (0 o più byte, fino a quanto consentito dal *chunk*). Il *flag* di compressione vale 0 se il testo non è compresso, 1 diversamente. Solo il campo testo può essere compresso. Come per *zTXt*, l'unico metodo di compressione definito

è 0 (flusso di dati *zlib* con compressione *deflate*). *Keyword* e testo non possono contenere il carattere *null*. Il testo può contenere ritorni a capo (decimale 10). La traduzione della *keyword* dovrebbe, come si intuisce facilmente, contenere il significato della parola chiave nella lingua indicata dal *tag*.

k. Chunk bKGD

Il *chunk* ausiliario *bKGD* specifica il colore di default da usare per lo sfondo dell'immagine (qualora l'immagine non faccia parte di un contesto più ampio, ad esempio in un *browser*). Per immagini con tipi di colore 0 e 4, il *chunk* contiene una tonalità di grigio (2 byte); per i tipi di colore 2 e 6 una tripletta RGB (2x3 byte); per il tipo di colore 3 un indice ad un colore della tavolozza (1 byte).

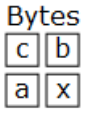
l. Chunk tIME

Il *chunk* ausiliario *tIME* conserva la data di quando è avvenuta l'ultima modifica all'immagine (e non la data inziale di creazione). Il *chunk* contiene 2 byte per l'anno completo (es: 1995, non 95); 1 byte per il mese (1-12); 1 byte per il giorno (1-31); 1 byte per l'ora (0-23); 1 byte per il minuto (0-59); 1 byte per il secondo (0-60). La data va specificata in accordo all'UTC (*Coordinated Universal Time*[27]*)* e deve essere aggiornata ad ogni modifica.

Filtraggio

Il filtraggio, come anticipato sommariamente in precedenza, altera l'immagine al fine di migliorare la compressione che riceverà subito dopo. Il filtraggio in sé, perciò, non riduce la dimensione dell'immagine. Scambiando alcuni valori dell'immagine con altri l'algoritmo di compressione può risultare più efficace; va da sé che il *decoder* dovrà essere in grado di ricostruire l'immagine originale. PNG supporta diversi metodi di filtraggio, ma solo uno alla volta può essere utilizzato per la stessa immagine.
Attualmente è definito solo il metodo 0, che racchiude cinque tipi di filtro: ogni *scanline* può utilizzare il tipo di filtro più opportuno. Non tutti i tipi di filtri sono egualmente efficaci sui dati. Quando un tipo di filtro viene scelto per una *scanline*, il suo identificativo viene anteposto ai byte della riga. I filtri sono applicati sui byte, non sui *pixel*, e perciò prescindono dal tipo di colore, dalla *bit depth* e dalla presenza di un canale *alpha* (che sarà anch'esso filtrato).
L'algoritmo di *filtering* scambia il valore di un byte con un altro. Di qui in poi si assume di star trattando con il metodo di filtraggio con identificativo 0. Al fine di comprendere il meccanismo di scambio è necessario spiegare la semplice notazione alla base del processo.

x	*Il byte che sta per subire il processo di filtering*
a	*Il byte corrispondente ad x nel pixel che precede immediatamente il pixel contenente x*
b	*Il byte corrispondente ad x nella precedente scanline*
c	*Il byte corrispondente a b nel pixel che precede immediatamente il pixel contenente b*
Orig(x)	*Il valore originale (non filtrato) del byte x*
Filt(x)	*Il valore del byte x dopo che è stato applicato il filtro*
Recon(x)	*Il valore ottenuto dopo aver applicato la corrispondente funzione di ricostruzione*



La figura mostra le posizione relative dei byte x, a, b e c.

Non è possibile aggiungere altri tipi di filtro ai cinque del metodo 0: questo assicura che i *decoder* abbiano sempre i mezzi per defiltrare le immagini. Riportiamo la tabella coi tipi di filtro:

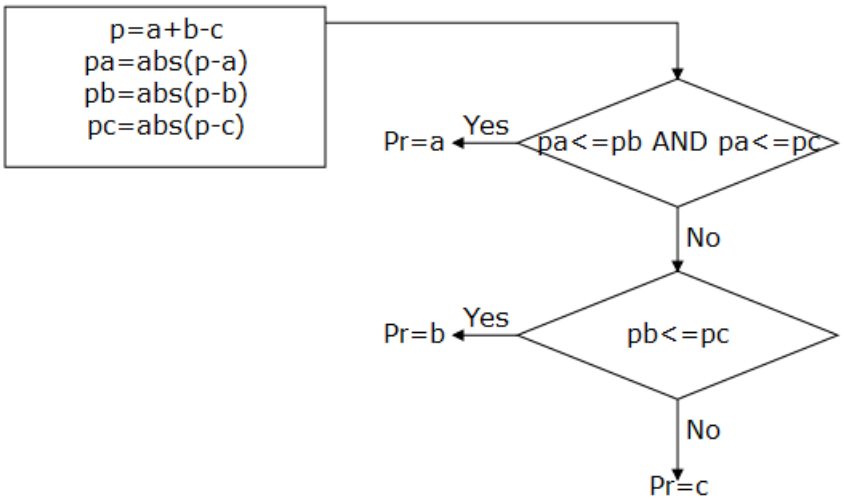
Tipo	Nome	Funzione di filtraggio	Funzione di ricostruzione
0	*None*	*Filt(x) = Orig(x)*	*Recon(x) = Filt(x)*
1	*Sub*	*Filt(x) = Orig(x) - Orig(a)*	*Recon(x) = Filt(x) + Recon(a)*
2	*Up*	*Filt(x) = Orig(x) - Orig(b)*	*Recon(x) = Filt(x) + Recon(b)*
3	*Average*	*Filt(x) = Orig(x) – floor((Orig(a) + Orig(b)) / 2)*	*Recon(x) = Filt(x) + floor((Recon(a) + Recon(b)) / 2)*
4	*Paeth*	*Filt(x) = Orig(x) - PaethPredictor(Orig(a), Orig(b), Orig(c))*	*Recon(x) = Filt(x) + PaethPredictor(Recon(a), Recon(b), Recon(c))*

Eventuali "byte a sinistra" del primo byte della scanline si assumono valere 0. Stessa assunzione vale per i "byte della scanline antecedente la prima". Ogni operazione viene eseguito modulo 256, al fine di permettere all'input e all'output di essere contenuti in un byte.

Nella filtro *Average*, la somma *Orig(a) + Orig(b)* deve essere eseguita senza *overflow*; *floor()* indica che i risultati della divisione, se frazionari, sono arrotondati all'intero più piccolo.
La funzione di filtraggio di *Paeth*[28] calcola una semplice funzione lineare dei tre *pixel* vicini a quello da elaborare (quello a sinistra, in alto ed in alto a sinistra), poi sceglie tra questi quello più simile al valore calcolato. L'algoritmo è un adattamento della tecnica di *Alan W Paeth*.
La funzione *PaethPredictor*, della tabella precedente, è la seguente (*Pr* è la predizione per x):

```
p = a + b - c
pa = abs(p - a)
pb = abs(p - b)
pc = abs(p - c)
if pa <= pb and pa <= pc then Pr = a
else if pb <= pc then Pr = b
else Pr = c
return Pr
```

La logica del funzionamento è rappresentabile con questo diagramma:



N.B.: I calcoli all'interno della funzione *PaethPredictor* devono essere eseguiti in modo preciso, senza *overflow*.
Poiché nella funzione di *filtering* i byte vengono elaborati in funzione dei precedenti, è necessario che l'ordine con cui vengono effettuati i confronti non sia modificato e sia comune sia in fase di filtraggio che in fase di defiltraggio (dall'alto verso il basso, da sinistra versa destra).

La *ratio* dell'algoritmo è rilevare in quale delle tre direzioni (verticale, orizzontale e diagonale) il gradiente dell'immagine è minimo.

Compressione

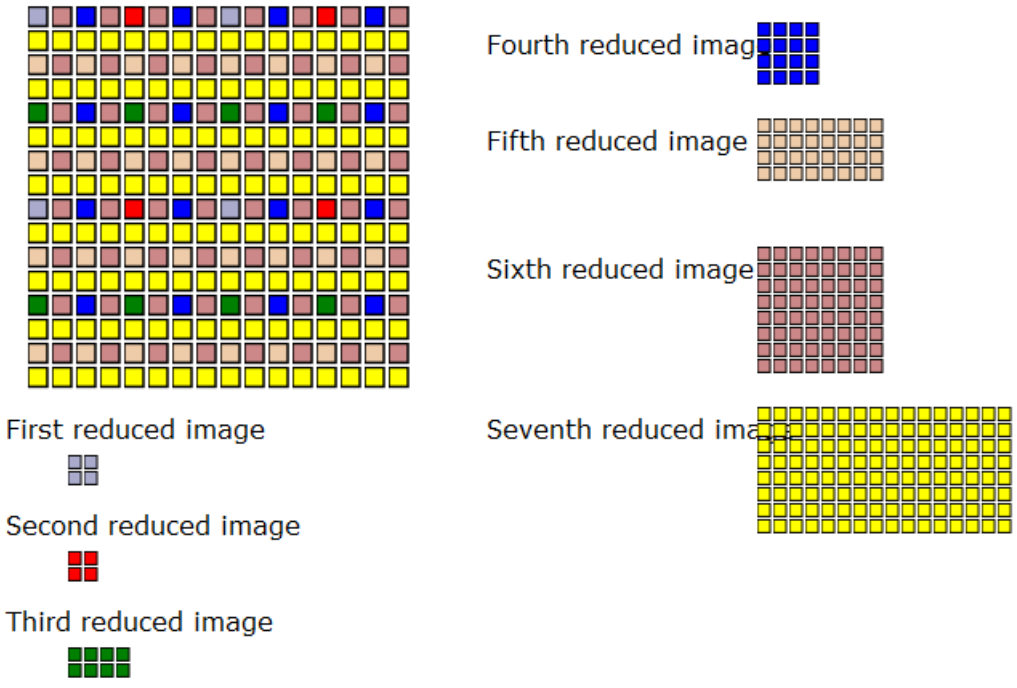
La compressione ha, come è facilmente intuibile, l'obiettivo di ridurre quanto più possibile la dimensione del file (confronti con altri formati sul rapporto dimensione/qualità sono affrontati più avanti). L'unico metodo definito attualmente per la compressione è quello con identificativo 0. Il metodo 0 del PNG è una compressione *deflate/inflate* (sgonfiamento/gonfiamento) con una finestra scorrevole (ovvero un limite superiore sulla dimensione del flusso di *deflate*) di al più 32768 byte. La compressione *deflate* è una derivata dell'*LZ77*.
I flussi di dati compressi sono memorizzati nel formato *zlib*, che è così composto: metodo di compressione/*flag* (1 byte) – che non è lo stesso del *chunk IHDR*; *flag* addizionali/bit di controllo (1 byte); blocchi dei dati compressi (n byte); valore di controllo (4 byte).
I dati compressi nel *datastream zlib* sono memorizzati come una serie di blocchi, ciascuno dei quali può rappresentare dati non compressi, dati compressi con *LZ77* codificati con codici di *Huffman* fissi o dati compressi con *LZ77* codificati con codici di *Huffman* personalizzati[29]. Un bit di demarcazione viene posto nel blocco finale per permettere al *decoder* di riconoscere la fine dei dati compressi. L'algoritmo usato per verificare se c'è corrispondenza con il valore di controllo non è il CRC (usato nei *chunk* del PNG): il valore di controllo, riferito ai dati non compressi, è un semplice controllo che permette di stabilire se gli algoritmi di compressione e decompressione sono stati implementati correttamente e non, come nel CRC, se i dati sono stati trasmessi integri.
Il meccanismo alla base della compressione è il seguente: ogni *scanline*, dopo essere stata filtrata, viene concatenata alla precedente. Il risultato di tale operazione viene compresso e il flusso di dati di output viene suddiviso in uno o più *chunk IDAT*. Di conseguenza, la concatenazione del contenuto di tali *chunk* costituisce il *datastream zlib*. E' importante sottolineare come non vi sia alcuna corrispondenza tra i blocchi dei dati compressi dall'algoritmo ed i dati contenuti nei *chunk IDAT*: capita soventemente, infatti, che si passi da un *chunk IDAT* al successivo frammentando un blocco del *deflate*. Questo non costituisce un problema, poiché, in fase decompressione, i dati dei *chunk IDAT* vengono concatenati tra loro, comportando la "ricostruzione" del blocco che era stato frammentato. Parimenti non vi è alcuna relazione tra i *chunk IDAT* e le *scanline* dell'immagine: un *chunk* potrebbe contenere più righe o anche parti di essa (purché consecutive).
Si ricorda, infine, che il metodo di compressione 0 viene adoperato anche nei *chunk iTXt, iCCP, zTXt,* ecc. In tal caso il *chunk* contiene tutto il *datastream zlib*, che non viene perciò frazionato.
Il Lettore che desidera approfondire l'argomento può leggere la documentazione su *zlib* e sull'algoritmo deflate.

Interlacciamento

L'interlacciamento consente di visualizzare una immagine PNG in modo progressivo, partendo da una vista grezza e migliorandola per successivi raffinamenti. Questo processo comporta mediamente un lieve aumento delle dimensioni del file, ma permette all'utente di avere un'anteprima dell'immagine rapidamente. Un *decoder* che esegue una visualizzazione "on-the-fly" dell'immagine interlacciata, produce l'effetto dissolvenza.
Due valori sono attualmente possibili per il metodo di interlacciamento: 0 e 1. Il metodo 0 significa che l'algoritmo di interlacciamento non è stata eseguito e che le *scanline* rappresentano l'immagine immutata. Il metodo 1, invece, noto come *Adam7* (dal nome del creatore *Adam Costello*), definisce 7 passi da applicare sull'immagine. Ogni passo seleziona e trasmette un sottoinsieme dei *pixel* dell'immagine, seguendo un *pattern* di dimensione 8x8, che si ripete per tutta l'immagine, a partire dall'angolo in alto a sinistra. Il *pattern* è il seguente:

```
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
```

Immaginando che la precedente sia una immagine di 64 *pixel*, marcati ciascuno con un numero, il *decoder* mostrerà progressivamente i pixel segnati col numero 1, poi quelli col numero 2, ecc. fino al settimo passo, quando l'intera immagine sarà stata visualizzata.
Considerando il sottoinsieme di *pixel* coinvolti in ciascun passo, possiamo idealmente ritenere che l'immagine venga divisa in 7 sotto-immagini. Per chiarire quanto detto, si consideri la seguente illustrazione:



Ad ogni sotto-immagine è stato assegnato un colore diverso. Dall'esempio si nota anche come il pattern 8x8 venga ripetuto sia orizzontalmente che verticalmente.
Ad ogni passo, scorrendo l'immagine dall'alto verso il basso, i *pixel* marcati sono presi ed inseriti in una nuova *scanline*, uno di fianco all'altro, da sinistra verso destra. Ad esempio, considerando ancora l'immagine precedente, col passo 2 viene creata una nuova *scanline* contenente i *pixel* 4 e 12 delle *scanline* 0 e 8 dell'immagine originale (si è assunto che il *pixel* nell'angolo in alto a sinistra sia il numero 0 della *scanline* 0); al passo 7 corrisponde una *scanline* contenente i *pixel* presenti nelle *scanline* originali numero 1, 3, 5, ecc.
Per poter essere applicato l'algoritmo di *filtering*, ogni scanline deve avere lo stesso numero di *pixel*, sicché questi vengono ripartiti in modo equo ad ogni passo. Per esempio, una immagine PNG di 16x16 *pixel* sarà ridotta al terzo passo in 2 *scanline*, ciascuna contenente 4 *pixel*.
Scanline che non riescono a raggiungere un numero intero di byte sono "imbottite" a sufficienza.

N.B.: Se l'immagine contiene meno di 5 colonne o righe alcuni passi saranno vuoti. *Encoder* e *decoder* dovranno gestire questo caso correttamente: in particolare i tipi di filtro vanno applicati solo a scanline non vuote (nessun filtro va applicato in un passo vuoto).

Approfondimenti sul formato

Di seguito, per completezza, vengono riportate alcune informazioni più dettagliate su aspetti più tecnici del formato PNG.

- ### *CRC*

I CRC dei *chunk* sono calcolati utilizzando il metodo standard definito nelle specifiche ISO 3309[30] e ITU-T V-42[31]. Il polinomio adoperato è: $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$.
I 32 bit del codice CRC vengono inizializzati tutti ad 1. Successivamente i dati di ogni byte vengono processati a partire dal bit meno significativo fino a quello più significativo. Viene poi eseguito il complemento a 1 del risultato. Il valore ottenuto viene trasmesso e memorizzato. Il bit meno significativo dei 32 costituenti il CRC è per definizione il coefficiente di x^{31}.
All'atto del calcolo del CRC viene spesso adoperata una tabella con valori pre-calcolati per accelerare la computazione.

- ### *Decompressione*

La tecnica di decompressione adoperata nella attuale versione non richiede che il flusso di dati compressi sia interamente disponibile. La decompressione e la visualizzazione possono cominciare anche solo con i dati dei primi *chunk IDAT*. E' estremamente improbabile che tale peculiarità venga meno con eventuali futuri metodi di compressione.

- ### *Chunk*

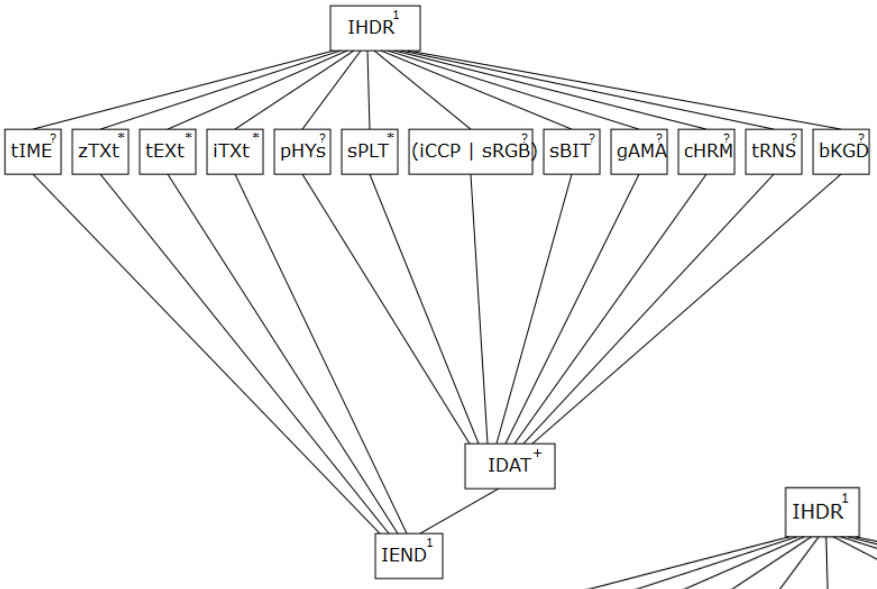
Come già riportato sommariamente in precedenza, i chunk possono apparire nel file una sola volta, più volte o non comparire affatto (come, ad esempio, quelli ausiliari). Inoltre devono susseguirsi in un certo ordine all'interno del formato. Tali regole possono essere riassunte dalle seguenti immagini, tratte direttamente dalle specifiche del formato[z], senza perdersi in troppe parole:

Critical chunks (shall appear in this order, except PLTE is optional)		
Chunk name	**Multiple allowed**	**Ordering constraints**
IHDR	No	Shall be first
PLTE	No	Before first IDAT
IDAT	Yes	Multiple IDAT chunks shall be consecutive
IEND	No	Shall be last

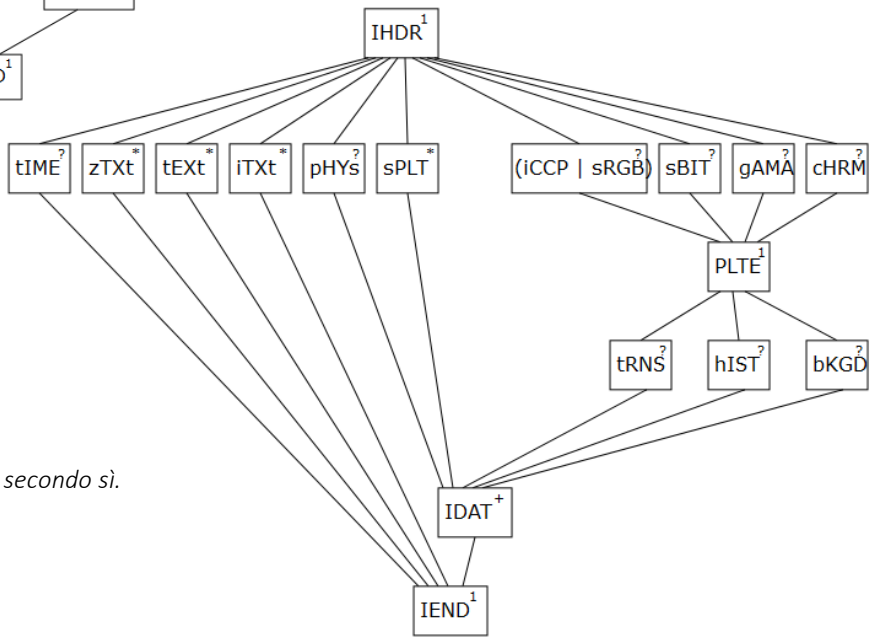
Regole per i chunk critici.

Ancillary chunks (need not appear in this order)		
Chunk name	**Multiple allowed**	**Ordering constraints**
cHRM	No	Before PLTE and IDAT
gAMA	No	Before PLTE and IDAT
iCCP	No	Before PLTE and IDAT. If the iCCP chunk is present, the sRGB chunk should not be present.
sBIT	No	Before PLTE and IDAT
sRGB	No	Before PLTE and IDAT. If the sRGB chunk is present, the iCCP chunk should not be present.
bKGD	No	After PLTE; before IDAT
hIST	No	After PLTE; before IDAT
tRNS	No	After PLTE; before IDAT
pHYs	No	Before IDAT
sPLT	Yes	Before IDAT
tIME	No	None
iTXt	Yes	None
tEXt	Yes	None
zTXt	Yes	None

Regole per i chunk ausiliari.

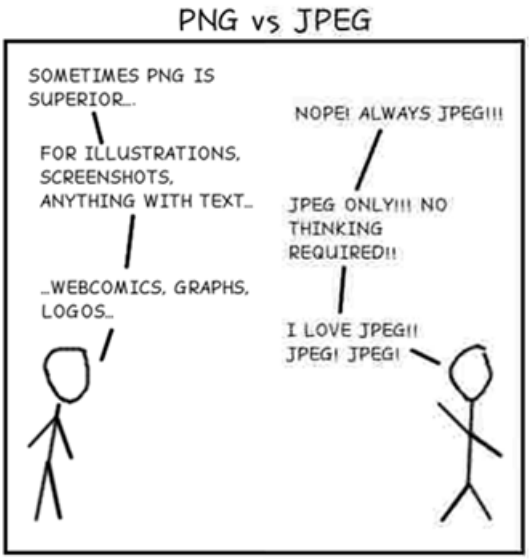


Symbol	Meaning
+	One or more
1	Only one
?	Zero or one
*	Zero or more
|	Alternative



Diagrammi reticolari di riepilogo.
Il primo non contiene il chunk PLTE, il secondo sì.

Confronto con altri formati



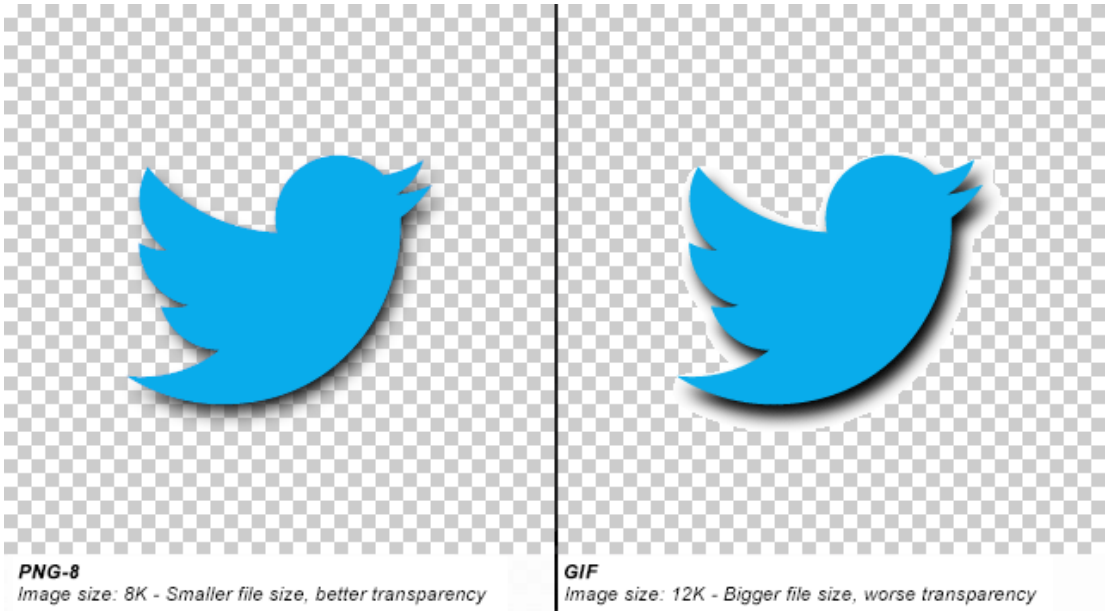
Web comic di lbrandy.com

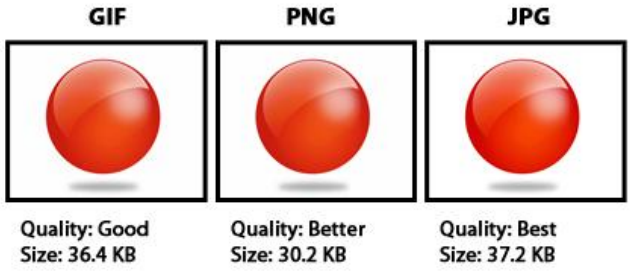
Stabilire quale formato d'immagine sia il migliore in assoluto non è cosa facile, forse non è proprio possibile. La verità è che ogni formato è adatto in alcune situazioni e la cosa più ragionevole da fare, quando bisogna sceglierne uno, è provarli tutti e vedere quale si adatta meglio al dominio applicativo. Di seguito considereremo solo tre formati: PNG, JPEG e GIF. La scelta ricade su questi poiché sono quelli che hanno avuto un maggiore impatto sulla comunità informatica dallo sviluppo dei computer (BMP è un formato obsoleto: non presenta alcuna caratteristica abbastanza valida da giustificare le enormi dimensioni delle sue immagini).

Il formato GIF supporta le animazioni a differenza degli altri due: questo lo rende la scelta obbligata se si presenta la necessità di mostrare immagini animate. Il suo inconveniente è il limitato *range* di colori che può memorizzare; tale caratteristica lo rende inappropriato per immagini in alta risoluzione, quali le fotografie, e più adatto ad immagini con blocchi "piatti" di colore, senza gradienti. Trova spesso impiego nei loghi. Sul versante dimensioni, inoltre, GIF risulta spesso peggiore di PNG, sebbene quest'ultimo sia molto dipendente da numerosi parametri (tipo di colore, filtraggio, interlacciamento, presenza di *chunk* ausiliari). File PNG possono essere più grandi di quelli GIF nelle situazioni in cui le due immagini vengono create a partire da una stessa sorgente in alta qualità, perché PNG è capace di memorizzare molte più informazioni di GIF, ad esempio quelle sulla trasparenza. Se prendiamo come base di confronto una immagine in formato PNG a 256 colori e senza metadati, allora la sua dimensione sarà minore della corrispondente immagine in formato GIF. Seppur dipendendo dal file e dal compressore, PNG ha dimensioni che oscillano tra il 10% e il 50% in meno di GIF; solo in rari casi si registra un incremento del 5% sul suo *competitor*[32]. La ragione è da ricercare nelle migliori *performance* dell'algoritmo *deflate* del PNG rispetto all'*LZW* di GIF. La maggiore efficienza deriva dalla fase di precompressione del PNG in cui viene applicato il filtro predittivo, capace di massimizzare i risultati del compressore.

Un confronto più complesso è quello tra PNG e JPEG. Quest'ultimo risulta il più in uso sul *web* ed il motivo è sicuramente il suo eccezionale rapporto qualità/dimensione. L'algoritmo alle spalle di JPEG permette di comprimere molto le immagini senza perdere troppo in qualità. Questo rende JPEG adatto ai siti *Internet*, in quanto la velocità di caricamento della pagina è un requisito non funzionale altamente auspicato. Tuttavia l'algoritmo di compressione è di tipo *lossy,* a differenza del compressore del PNG, e questo causa una perdita continua di dettaglio se l'immagine viene salvata ripetutamente (viceversa, nel PNG, le immagini restano immutate nella qualità ad ogni salvataggio). Molti concordano nel ritenere che JPEG sia ideale per memorizzare foto e da evitare quando si tratta con immagini contenenti testo o effetti basati su trasparenza.

Limitandoci a considerare PNG, è doveroso dire bisogna scendere ad un compromesso tra alta profondità di colore, interlacciamento, velocità di compressione e presenza di metadati. In base alle esigenze risulta necessario sacrificare delle funzionalità non indispensabili. L'interlacciamento, ad esempio, velocizza sensibilmente il *rendering* di immagini molto grandi, ma aumenta la dimensione: tale guadagno potrebbe risultare inutile per piccoli file. Una compressione veloce ma poco efficiente potrebbe tornare utile nel momento in cui un'immagine viene ripetutamente modificata e salvata, mentre una lenta ma più efficiente potrebbe essere più appropriata in fase di memorizzazione finale o condivisione.
Per quanto detto, è insensato dichiarare un "vincitore": si preferisce, invece, mostrare al Lettore alcuni esempi di immagini in diversi formati per permetterGli una migliore valutazione[33][34][35].





Logo salvato come JPEG.
Dimensione: 63.05 KB

Logo salvato come PNG.
Dimensione: 9.16 KB

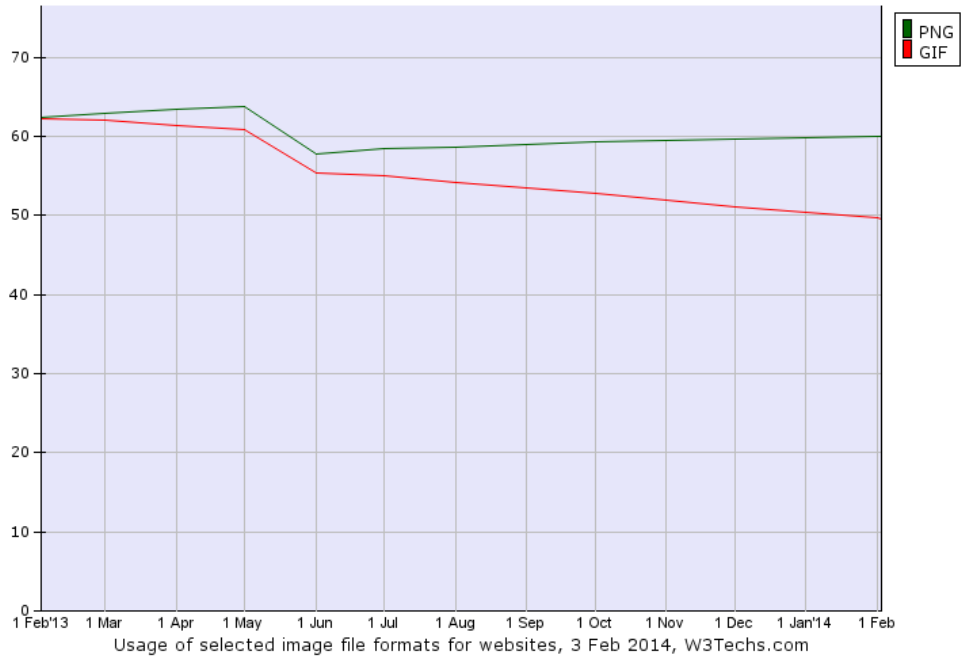
PNG Manager: commento al programma

PNG Manager è un visualizzatore di immagini PNG con alcune caratteristiche che lo differenziano dagli altri programmi perseguenti la stessa finalità. Tali *features* sono strettamente correlate al formato PNG ed alle sue specifiche.
Il programma consente all'utente di selezionare ed aprire immagini, che saranno decodificate e visualizzate nell'interfaccia grafica. In base alla tipologia di immagine PNG selezionata verrà scelto un *decoder* appropriato. I *decoder* a disposizione del *PNG Manager* sono quello nativo di *Java* e quello scritto *ex novo* (che utilizza, a sua volta, un *defilterer* personale). Quest'ultimo supporta i tipi utilizzati più frequentemente (*palette* con qualunque *bit depth*, *grayscale* con *bit depth* pari a 1 o 8, *RGB* con *bit depth* pari a 8), ma non gestisce le informazioni memorizzate nei *chunk* ausiliari o l'interlacciamento. Per questa ragione viene comunque data la possibilità all'utente di invocare il decoder di *Java* che saprà gestire i metadati (la casella che attiva l'opzione non è selezionabile allorquando il *decoder* in uso è già quello di *Java*).
Il programma effettua diversi controlli sulla coerenza e sull'integrità dell'immagine, segnalando all'utente file corrotti o non validi.
Dopo aver caricato un'immagine, nell'interfaccia grafica saranno mostrate tutte le informazioni estraibili dal *chunk IHDR* (dimensione, tipo di colore, profondità di bit, metodi di compressione, filtraggio e interlacciamento), nonché l'elenco dei *chunk* ausiliari con il relativo contenuto.
Una barra degli strumenti offre all'utente la possibilità di esportare tale prospetto in un file di testo, o di attivare/disattivare la modalità in bianco e nero.
Infine, è possibile codificare l'immagine correntemente visualizzata in un nuovo file PNG eliminando i metadati oppure convertirla in una immagine in formato JPEG.

Per maggiori dettagli sul funzionamento del programma si rimanda alla *Javadoc*, che documenta classi e metodi impiegati.

Conclusioni

Dopo aver trattato in modo, si spera, abbastanza esaustivo il formato è giunto il momento di concludere la relazione e provare a rispondere alla domanda posta inizialmente: “PNG è il formato definitivo?”.
Risulta chiaro che il PNG sia un formato solido, flessibile al cambiamento, versatile, dalle grandi performance e libero, proprio come era nelle intenzioni di chi lo progettò. Questo ne fa una importante risorsa per la comunità informatica mondiale, che lo sta impiegando in modo sempre più massiccio.



A testimonianza di questo fatto riportiamo il seguente dato: da ottobre 2013, mese in cui è cominciata la scrittura di questa relazione, a febbraio 2014, mese della sua consegna, GIF ha perso una ulteriore quota del 5% sul PNG.
E’ facile immaginare l’idea che chi Vi scrive ha del formato, ma proviamo ora a compiere un’analisi oggettiva.
PNG sembra avere tutte le caratteristiche progettuali di anticipazione al cambiamento necessarie per sopravvivere in un mondo in continua evoluzione come quello dell’informatica moderna.
Chi, però, potrà dire se PNG sarà destinato a contenere, nelle decadi a venire, le immagini che visualizzeremo sui nostri schermi è solo il tempo. La storia ci insegna che anche i protocolli, i formati e le idee più consolidate a volte divengono improvvisamente obsolete e che piccoli sforzi atti a vincere la forza dell’abitudine possono risultare in miglioramenti significativi dell’esperienza d’uso quotidiana.
Risulta difficile immaginare che, nel futuro, non saranno sviluppati nuovi algoritmi di compressione, metodi di filtraggio o, addirittura, nuovi dispositivi fisici con caratteristiche tecniche tali da rendere inappropriato il formato PNG.
La sfida di PNG è quella di saper resistere al vento del cambiamento (il nuovo formato *WebPi* di *Google* sembra un ottimo *competitor*) incorporando tutte le novità che il progresso ci riserva: andare contro corrente ed arroccarsi nelle proprie consolidate caratteristiche è una scelta che conduce all’inevitabile destino del divenire un formato *legacy*.

Sitografia/Bibliografia

[1] http://www.w3.org/TR/PNG/#1Scope
[2] http://www.w3.org/Graphics/GIF/spec-gif89a.txt
[3] http://w3techs.com/technologies/history_overview/image_format/all/y
[4] http://www.gnu.org/philosophy/gif.html
[5] https://groups.google.com/forum/#!msg/comp.graphics/tylpVt2y9s8/eHWKNVLYMREJ
[6] http://www.zlib.net/
[7] http://www.libpng.org/pub/png/pngfaq.html#animation
[8] http://www.libpng.org/pub/png/book/chapter12.html
[9] https://wiki.mozilla.org/APNG_Specification
[10] http://www.w3.org/TR/REC-png-multi.html
[11] http://www.apogeonline.com/webzine/2004/06/24/01/200406240101
[12] http://burnallgifs.org/
[13] http://www.pcmag.com/article2/0,2817,1645187,00.asp
[14] http://w3techs.com/blog/entry/the_png_image_file_format_is_now_more_popular_than_gif
[15] http://w3techs.com/technologies/comparison/im-gif,im-png
[16] http://w3techs.com/technologies/breakdown/im-png/top_level_domain
[17] http://w3techs.com/technologies/breakdown/im-png/web_server
[18] http://www.w3.org/TR/PNG-Introduction.html
[19] http://www.w3.org/TR/PNG/#4Concepts.Sourceimage
[20] http://www.w3.org/TR/PNG/#11Chunks
[21] http://zlib.net/crc_v3.txt
[22] http://www.w3.org/TR/PNG/#table111
[23] http://www.ietf.org/rfc/rfc1951.txt
[24] http://it.wikipedia.org/wiki/Correzione_di_gamma
[25] http://www.color.org/index.xalter
[26] http://www.w3.org/TR/PNG/#11textinfo
[27] http://it.wikipedia.org/wiki/Tempo_coordinato_universale
[28] Paeth, A.W., "Image File Compression Made Easy", in *Graphics Gems II*, James Arvo, editor. Academic Press, San Diego, 1991. ISBN 0-12-064480-0.
[29] http://en.wikipedia.org/wiki/Huffman_coding
[30] ISO/IEC 3309:1993, Information Technology — Telecommunications and information exchange between systems — High-level data link control (HDLC) procedures — Frame structure.
[31] International Telecommunications Union, Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion, ITU-T Recommendation V.42, 1994, Rev. 1.
[32] http://www.libpng.org/pub/png/book/chapter09.html#png.ch09.div.3
[33] http://stackoverflow.com/questions/2336522/png-vs-gif-vs-jpeg-when-best-to-use
[34] http://www.sitepoint.com/gif-png-jpg-which-one-to-use/
[35] http://www.makeuseof.com/tag/know-when-to-use-which-file-format-png-vs-jpg-doc-vs-pdf-mp3-vs-flac/