Università degli Studi di Genova
Facoltà di Ingegneria - Polo di Savona
via Cadorna 7 - 17100 Savona
Tel. +39 019 264555 - Fax +39 019 264558

*Cogne, 7-9 Giugno 2001*

# *Qualche riflessione sui compressori*

*P. Oliva*

## Introduzione.

L'esigenza di comprimere i file nasce dalla necessità di contenere il maggior numero di informazioni possibile nello spazio che si ha a disposizione (memoria ram, disco fisso, cd), che per quanto grande sia, sembra sempre essere insufficiente; qualche decina di anni fa ci saremmo ritenuti molto fortunati se avessimo potuto avere a disposizione qualche centinaia di KB e oggi non ci bastano i Mega o i GigaBytes (nulla di più vero de "l'appetito vien mangiando").

In altre situazioni ci si trova di fronte alla necessità di comprimere file, ad esempio nella trasmissione di dati, internet, ecc., per accelerare la quantità di informazioni trasmesse e diminuire di conseguenza i tempi.

Vi sono poi casi in cui non vi è l'obbligo di trasmettere un file compresso che sia esattamente ricostruibile all'arrivo: si cerca cioè di comprimere il più possibile anche a scapito di qualche lieve perdita di informazione (si pensi alle trasmissioni della telefonia, telefoni cellulari, ecc., oppure delle immagini televisive, o alla registrazione su nastro o CD di filmati digitali o di brani musicali).

Si consideri che, per quel che riguarda file di testo, ad esempio l'intera Divina Commedia richiede circa 500 KB, i Promessi Sposi circa 1.300 KB; un'immagine 800×600 a 256 colori, 480 KB (sullo schermo ci sono tanti punti quante sono le lettere nella Divina Commedia); un secondo di audio su un CD necessita di 176 KB, un secondo di filmato più di 25000 KB (preciseremo in seguito il motivo di tali valori).

Divideremo pertanto queste brevi considerazioni sui compressori in due parti: nella prima affronteremo quelli che non hanno perdita di informazione, nella seconda quelli con perdita di informazione.

## Compressori senza perdita di informazione.

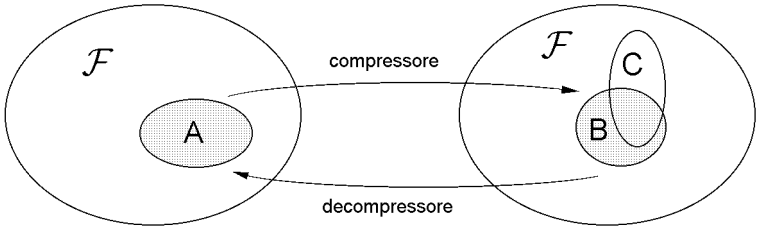
Prima di esaminare singolarmente alcuni tipi di compressori, facciamo una semplice considerazione.

Un compressore è un'applicazione $c$ (bigettiva, se si vuole poi ricostruire esattamente il file originale mediante la sua applicazione inversa $d$ di decompressione) dall'insieme dei file $\mathcal{F}$ nello stesso insieme. Se $f \in \mathcal{F}$ e $\text{len}(f)$ indica la sua lunghezza in byte, il file sarà effettivamente compresso se $\text{len}(c(f)) < \text{len}(f)$.

Per essere un buon compressore, $c$ deve comprimere più file possibile, il più possibile; ma c'è un limite teorico al numero di files comprimibili?

Per rispondere a questa domanda facciamo un esempio: ci chiediamo, dato un compressore $c$, tra tutti i file $f$ di lunghezza $10^6$ byte (che sono tantissimi, ovvero $256^{1000000}$ ) quanti di questi saranno compressi da $c$ per almeno un 1% (cioè $\text{len}(f) \leq 990000$)?

Sia pertanto $A$ l'insieme dei file di lunghezza $10^6$ byte e $B$ l'insieme dei file ottenuti dopo l'utilizzo del compressore.




Essendo $c$ un'applicazione bigettiva, il numero di file contenuti in $A$ è uguale al numero dei file contenuti in $B$.

Ora, se indichiamo con $C$ l'insieme dei file di lunghezza minore di 990000, i file effettivamente compressi per almeno l'1% saranno quelli di $B$ contenuti in $C$; il loro numero sarà pertanto minore o uguale al numero di file che stanno in $C$; per la verità probabilmente molto minore di tale numero.

Comunque, il numero di file che hanno lunghezza minore o uguale a 990000 è dato da

$$256 + 256^2 + 256^3 + \cdots + 256^{990000} = 256\frac{256^{990000} - 1}{255}$$

(circa uguale a $1.004 \cdot 256^{990000}$ ).

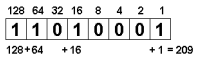
Pertanto, dei $256^{1000000}$ file dati, meno di circa $256^{990000}$ verranno compressi da $c$ per almeno l'1%, cioè meno di uno su $256^{10000}$ (ovvero uno su $2.5 \cdot 10^{24082}$ ).

Questo risultato è tale da scoraggiare qualunque persona (ragionevole) che pensi di affrontare il problema della compressione di un file.

In realtà però, per fortuna, le cose non stanno proprio così. Basta pensare ad un file di testo in cui sono utilizzati non più di un centinaio di simboli (le 26 lettere minuscole e maiuscole, le cifre e un po' di segni di punteggiatura e poco altro) per cui meno di sette bit sono sufficienti per ogni simbolo (essendo $2^7 = 128$), e quindi un qualunque file di testo può facilmente essere ridotto di 1/8.

Un altro caso è ad esempio quello dei disegni, nei quali molti punti adiacenti hanno lo stesso colore, e quindi le informazioni in essi contenute sono in realtà molto poche.

## Codifiche.




L'unità di misura della lunghezza di un file è il *byte* (il protagonista principale nascosto dietro ogni situazione della nostra storia) che è costituito da 8 *bit* (0 o 1, che sono il minimo elemento di informazione).

Vi sono in tal modo $2^8 = 256$ disposizioni differenti di 0 ed 1 in un byte, o più semplicemente ogni byte è identificabile con un numero intero tra 0 e 255, estremi compresi.

Tali valori sono pure identificabili con opportuni codici: ad esempio il codice ASCII (American Standard Code for Information Interchange) attribuisce un carattere ad ogni numero.

I primi 32 valori (da 0 a 31) sono codici di controllo: per esempio 10 è LF (line feed), 12 è FF (form feed), 13 è CR (carriage return), ecc.

Dal 32 al 126 vi sono caratteri alfabetici, cifre, segni di interpunzione vari:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | | 48 | 0 | 64 | @ | 80 | P | 96 | ‘ | 112 | p |
| 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 34 | ” | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 39 | ’ | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 44 | , | 60 | ¡ | 76 | L | 92 | \ | 108 | l | 124 | — |
| 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 46 | . | 62 | ¿ | 78 | N | 94 | ˆ | 110 | n | 126 | ˜ |
| 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | | |

Dal 127 in poi vi sono altri simboli grafici.

In questo senso va interpretata la precedente affermazione che attribuiva alla Divina Commedia una dimensione di circa 500 KB (cioè vi sono in essa 500000 caratteri alfabetici, virgole, punti, ecc.)

Abbiamo già notato però che utilizzando le ventisei lettere dell'alfabeto maiuscolo e minuscolo, le cifre e i vari segni di interpunzione, bastano (e avanzano) 128 simboli differenti, che sono codificabili con soli 7 bit; cioè lo stesso testo potrebbe essere ricodificato con una lunghezza pari a 7/8 di quella precedente.

Tale effetto corrisponde comunque ad una precisa scelta: attribuire ad ogni simbolo usato lo stesso numero di bit.

Questa scelta appare molto semplice, ed in particolare molto facile è la decodifica del file, in quanto basterà dividere la sequenza di bit in gruppi di lunghezza fissata e quindi decodificarli utilizzando la tabella opportuna.

Un esame un po' più attento di un file di testo mostra però alcune caratteristiche di un certo interesse: le varie lettere non si ripetono con uguale frequenza, ad esempio le vocali compaiono più spesso delle consonanti, ed anche tra le consonanti ve ne sono alcune meno frequenti di altre (la *w* è quasi assente in testi italiani e comunque molto più rara della *b*).

Questo fa venire in mente di utilizzare codici a lunghezza variabile, più corti per le lettere più frequenti e più lunghi per quelle meno frequenti.

Un noto esempio di tale tipo di codifica è dato dal codice Morse:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | ·- | G | --· | M | -- | S | ··· | Y | -·-- |
| B | -··· | H | ···· | N | -· | T | - | Z | --·· |
| C | -·-· | I | ·· | O | --- | U | ··- | | |
| D | -·· | J | ·--- | P | ·--· | V | ···- | | |
| E | · | K | -·- | Q | --·- | W | ·-- | | |
| F | ··-· | L | ·-·· | R | ·-· | X | -··- | | |

Per la verità tale codice non utilizza solo due simboli (punto e linea), ma anche un intervallo più lungo per lo spazio tra i caratteri; questo perché il codice Morse non rispetta un semplice criterio che evita ambiguità: ad esempio, se non vi fossero le separazioni tra i caratteri, la sequenza -··· potrebbe essere interpretata sia come la lettera B, sia come la sequenza DE.

Questo è lo stesso problema che sorge quando si devono attribuire agli abbonati i numeri di telefono: o si decide che ogni numero ha una quantità fissa prestabilita di cifre, oppure, se si vogliono lasciare i numeri brevi per le chiamate di emergenza (112, 113, 115 ecc.) non sarà più possibile attribuire a nessuno numeri che iniziano con tali sequenze. È chiaro che se io dovessi chiamare qualcuno che ha il numero 115234, quando ho digitato 115 mi risponderebbero i Vigili del Fuoco.

Ora, Morse dedusse la frequenza dei singoli caratteri dalla quantità di tipi utilizzati dai tipografi per comporre gli articoli di giornale.

Noi possiamo determinare tali frequenze per la lingua italiana, ad esempio nel testo dei Promessi Sposi (circa 1300000 byte); raggruppando le lettere senza distinzione tra maiuscole e minuscole si ottiene la seguente tabella di frequenze:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | 11.5 | G | 1.7 | M | 2.4 | S | 5.4 | Y | 0.0 |
| B | 1.0 | H | 1.3 | N | 7.3 | T | 6.1 | Z | 0.7 |
| C | 4.7 | I | 9.6 | O | 9.6 | U | 3.6 | | |
| D | 3.7 | J | 0.0 | P | 3.0 | V | 2.3 | | |
| E | 12.0 | K | 0.0 | Q | 0.8 | W | 0.0 | | |
| F | 1.1 | L | 5.6 | R | 6.6 | X | 0.0 | | |

È quindi evidente come vi siano delle grosse differenze tra le varie lettere, ed è pertanto chiaro che una oculata codifica possa permettere un certo risparmio.

A tale scopo va però tenuto presente il Teorema di Shannon (1940), che pone un limite teorico alla capacità di compressione di una sequenza di simboli.

*Dato un testo contenente n simboli distinti, ciascuno con frequenza* $f_i$, $i = 1, .., n$, *e avente lunghezza totale* $N$, *una qualunque ritrascrizione che faccia uso di codifiche a lunghezza variabile non sarà comunque mai più breve di un numero di bit pari a*

$$-N \sum_{i=1}^{n} f_i \log_2 f_i$$

## La codifica di Huffman.

Vediamo quindi come sia possibile attribuire i codici ai vari simboli in maniera tale da sfruttare al massimo le differenti frequenze di comparsa nel testo (ed in modo comunque da rendere univoca la decodifica), avvicinandosi al limite teorico visto sopra.

Vi sono differenti modi per fare ciò; i più noti sono quelli dovuti a Shannon-Fano e ad Huffman (1952), simili ed entrambi molto semplici.

Illustriamo qui il secondo metodo e per fare ciò facciamo un esempio con un testo a caso:

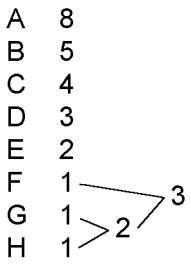
BCACABDHEAACBACEFADAGBBDA

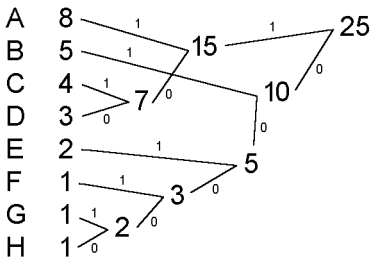
In tale testo sono presenti 8 caratteri (identificati con A,B,C,D,E,F,G,H) che compaiono in numero differente di volte, secondo la seguente tabella

A:8 B:5 C:4 D:3 E:2 F:1 G:1 H:1

(il fatto che le frequenze decrescano in ordine alfabetico, cioè che la lettera A sia la più frequente e così via, è del tutto ininfluente).

Poste ora in una tabella i vari simboli in ordine decrescente, rispetto al numero delle loro apparizioni (e quindi per puro caso qui in ordine alfabetico) uniamo con due linee i simboli che compaiono meno volte (G ed H) scrivendo al punto di incontro il valore 2 (somma di 1 ed 1).


Trascurando i valori già utilizzati, uniamo i due nuovi valori più bassi (1 di F, e 2 del gruppo

GH), con due linee, scrivendo al punto di incontro 3 (somma di 1 e 2), e così via fino ad arrivare ad un solo ultimo valore, somma di tutti i valori di ogni singola lettera (25, ovvero i totale dei caratteri nel testo considerato).

Partendo ora dal vertice in alto a destra (contrassegnato dal numero 25) è possibile raggiungere ogni simbolo seguendo i vari rami dell'albero costruito; il codice da attribuire ad ogni simbolo è determinato da tale percorso, avendo l'accortezza di attribuire ad ogni diramazione il valore 1 al ramo superiore ed il valore 0 a quello inferiore (o viceversa).

Si ottiene in tal modo la seguente tabella:

A : 11 B : 01 C : 101 D : 100 E : 001 F : 0001 G : 00001 H : 00000

e la precedente sequenza BCACABDHEAACBACEFADAGBBDA verrebbe quindi codificata da

01 101 11 101 11 01 100 00000 001 ..... ovvero 01101111011110110000000001.....

Si noti che se avessimo utilizzato codici a lunghezza fissa, avremmo dovuto utilizzare 3 bit per ogni lettera (essendo le lettere $8 = 2^3$) ed il messaggio avrebbe avuto una lunghezza di 75 bit ($3 \times 25$).

Con la codifica a lunghezza variabile il testo viene codificato con 67 bit ($2 \times 8$ per le A, più $2 \times 5$ per le B, e così via più $3 \times 4 + 3 \times 3 + 3 \times 2 + 4 \times 1 + 5 \times 1 + 5 \times 1$) con un risparmio di 8 bit su 75, ovvero quasi l'11%.

È notevole osservare come il teorema di Shannon fissi il limite minimo per quel testo a 65.7 bit, molto vicino a quello qui ottenuto.

Per decodificare il messaggio sarà sufficiente avere la tabella di codifica e quindi esaminare i bit ad uno ad uno:

il primo bit è 0 e non corrisponde a nessun carattere; i primi due sono 01 e questo codice corrisponde a B; e così di seguito fino a decodificare tutto il messaggio.

Si noti tra l'altro che qualora per errore si perdesse qualche bit, ad esempio il primo, il messaggio verrebbe interpretato come

11 01 11 101 11 01 100 ... ovvero ABACABD ....

cioè dopo i primi caratteri (in questo caso due) interpretati in maniera erronea, il testo viene codificato in maniera corretta.

Ciò non sarebbe accaduto se avessimo usato codici a lunghezza fissa, dove iniziare dal secondo bit avrebbe portato ad una divisione dei bit a tre a tre sfasati di uno, e quindi ad un testo totalmente errato.

Un inconveniente del metodo di Huffman, comunque tipico dei codici a lunghezza variabile, è che bisogna preliminarmente esaminare tutto il file per calcolare le frequenze dei singoli byte, e ciò può essere dispendioso in termini di tempo.

Per concludere osserviamo che, se utilizzassimo tale metodo per codificare i Promessi Sposi, (questa volta tenendo conto di tutti i caratteri minuscoli e maiuscoli e della punteggiatura) potremmo ridurre i circa 1308000 byte a soli 714000, con un risparmio del 45%, compressione 1.8/1.

## I formati RunLengthEncode e PCX.

Un altra situazione interessante si presenta nel caso in cui molti byte consecutivi sono uguali (ad esempio come già osservato nel caso di file di tipo grafico: in un disegno vi sono molti punti adiacenti dello stesso colore).

Una semplice codifica in tal caso è rappresentata dal cosiddetto metodo RunLengthEncode (RLE) di cui diamo qui una delle varie versioni.

In tale metodo il file compresso è costituito da vari blocchi di byte, ciascuno dei quali è formato da un byte *lunghezza* seguito da 1 a 128 byte di dati. Se *lunghezza* è nell'intervallo da 0 a 127, i

seguenti *lunghezza+1* byte (da 1 a 128) sono copiati così come sono durante la decompressione. Se *lunghezza* è nell'intervallo da 129 a 255, il successivo byte è ricopiato per 257-*lunghezza* volte (da 2 a 128) durante la decompressione. Il valore 128 è utilizzato come file del file.

Ad esempio la sequenza di byte (scritti in decimale)

6 5 12 203 2 2 2 2 2 2 43 221 3 5 76 .....

viene compressa con

**3** 6 5 12 203 **251** 2 **4** 43 221 3 5 76 ....

dove si sono indicati in grassetto i byte relativi alla *lunghezza*.

Questo metodo, nel caso migliore, porta ad una compressione di 64/1 con una sequenza di tutti byte uguali (ad esempio una sequenza di 128 zeri, viene codificata con **129** 0).

Nel caso peggiore, byte senza alcuna ripetizione, si ha una espansione di 128/129 (128 byte vengono codificati con **127** ed i 128 byte di dati).

Un altro formato simile, utilizzato espressamente per i file grafici, è il PCX.

In tale formato, nel file compresso, codici compresi tra 0 e 191 vengono ricopiati così come sono dal decompressore, mentre un *codice* tra 193 e 255 indica che il byte successivo va ripetuto *codice-192* volte.

La solita sequenza

6 5 12 203 2 2 2 2 2 2 43 221 3 5 76 .....

viene compressa con

6 5 12 **193** 203 **198** 2 43 **193** 221 3 5 76 .....

ove si sono indicati in grassetto i codici superiori a 192.

Si notino in particolare le sequenze **193** 203 e **193** 221, necessarie per codificare i byte superiori a 192.

Pertanto, in tale formato, nel caso migliore di byte tutti uguali, si ha una compressione di 63/2 (ad esempio 63 byte uguali a zero sono compressi con **255** 0); mentre nel caso peggiore, con tutti i byte senza ripetizioni e maggiori di 192, si ha una espansione di 1/2, poiché sono sempre necessari 2 byte per ogni dato.

## Compressori con dizionario.

Tutti i compressori visti prima lavorano sul singolo byte (o su gruppi di byte uguali), ma l'esame di vari file rivela la ripetitività di più gruppi di byte (ad esempio le parole in un file di testo).

Per comprimere al meglio questi file si sono introdotti i compressori cosiddetti con dizionario.

Tali compressori si possono dividere in due grandi gruppi:

-) al primo appartengono quelli che determinano se una certa sequenza di caratteri è già comparsa precedentemente, ed in tal caso codificano solo un puntatore alla precedente apparizione (LZ77 : Lempel e Ziv 1977, e suoi raffinamenti, LZSS : Store e Szymanski 1982);
-) al secondo gruppo appartengono quelli che creano un dizionario per ogni nuova sequenza che compare e quindi alla prima occorrenza successiva generano un puntatore al dizionario (LZ78 : Lempel e Ziv 1978, e suoi raffinamenti, LZW : Welch 1984).

Per capire come funzionano, si consideri il seguente esempio: sia data la stringa

AABBCBBAABC

Il compressore considera la prima sequenza di due lettere AA e controlla se questa è già comparsa precedentemente (per una certa precedente lunghezza prestabilita); poiché questi sono i primi caratteri è evidente che non vi è stata ancora alcuna ripetizione, e quindi viene emesso il primo carattere A.

Si passa poi al secondo gruppo AB e poiché neppure questo è ancora apparso, viene emesso il secondo carattere A.

Si procede in tal modo emettendo i singoli caratteri B, B, C, finchè non si arriva alla sequenza BB, che è già comparsa, per cui viene emesso un codice relativo alla coppia (*distanza, lunghezza*), in questo caso (3,2).

Al successivo passaggio si incontra la sequenza AAB che si è già ripetuta, e quindi viene emesso un codice relativo ai valori (7,3).

Si procede così fino alla fine della stringa e tutti i codici in tal modo ottenuti vengono poi compressi ulteriormente utilizzando ad esempio una codifica di tipo Huffman.

Tale metodo risulta notevolmente vantaggioso: necessita di un certo tempo per la ricerca delle eventuali ripetizioni, ma è poco dispendioso in termini di memoria e la decompressione risulta velocissima: ogni volta che si incontra un puntatore all'indietro è sufficiente ricopiare la parte indicata di stringa, già decodificata.

È utilizzato ad esempio dal Pkzip.

I Promessi Sposi, compressi con tale metodo, si riducono da 1303 KB a 498 KB, con una compressione 2.6/1.

Il secondo metodo usa il seguente algoritmo:

1) All'inizio si ha un dizionario contenente tutti i differenti caratteri che appaiono nella stringa e P è una stringa nulla;
2) Sia C il successivo carattere nella stringa da codificare;
3) La stringa P+C è presente nel dizionario?
   a) se sì, si definisce P:=P+C ;
   b) se no:
      i) si genera il codice corrispondente a P,
      ii) si aggiunge la stringa P+C al dizionario,
      iii) si definisce P:=C,
4) si ripete dal punto 2) fino alla fine della stringa.

In pratica, data ad esempio la stringa

ABBABABAC

si parte da un dizionario costituito da tre caratteri

(1) : A , (2) : B , (3) : C

e si ottiene:

| Passo | Output | Dizionario | Stringa rimanente |
|---|---|---|---|
| 1 | (1) | (4) : AB | BBABABAC |
| 2 | (2) | (5) : BB | BABABAC |
| 3 | (2) | (6) : BA | ABABAC |
| 4 | (4) | (7) : ABA | ABAC |
| 5 | (7) | (8) : ABAC | C |
| .. | .. | ..... | ..... |

Il decompressore ripete all'inverso quanto fatto dal compressore (ricostruendo anche il dizionario, con qualche accortezza che qui non stiamo a precisare).

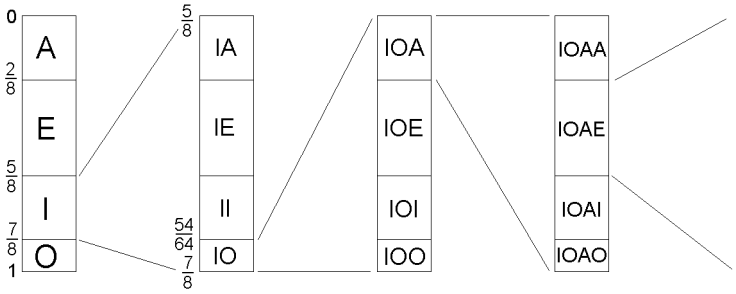
Tale metodo richiede un po' di memoria temporanea per far posto al dizionario; in compenso risulta più veloce in quanto diminuiscono i confronti da effettuare.

È usato ad esempio nei formati grafici di tipo GIF, ed è vincolato da licenza, per cui non è utilizzabile senza autorizzazione, se non per scopi personali.

## La codifica aritmetica.

Vediamo ora un metodo che permette di associare ad ogni sequenza di caratteri un numero razionale in [0,1) che identifica univocamente la sequenza data.

Il seguente esempio illustra come ottenere tale numero: sia data la sequenza IOAEEIAE ; calcolate le frequenze $f_i$ con cui compaiono i quattro simboli utilizzati, ovvero 2/8, 3/8, 2/8 e 1/8, rispettivamente per A, E, I, O, il segmento [0,1) viene suddiviso in parti proporzionali alle frequenze dei singoli caratteri. Ciascuna parte viene risuddivisa ancora con le stesse proporzioni e così via.




Nell'esempio considerato la prima barra a sinistra mostra come al primo carattere 'I' della sequenza venga attribuito l'intervallo $\left[\frac{5}{8}, \frac{7}{8}\right)$; tale segmento è poi ingrandito nella figura per mostrare come alla sequenza dei primi due caratteri 'IO' viene assegnato l'intervallo $\left[\frac{54}{64}, \frac{56}{64}\right)$, e tale procedimento viene iterato fino alla fine della sequenza.

In tal modo alla sequenza completa viene assegnato l'intervallo $B = \left[\frac{888290}{2^{20}}, \frac{888317}{2^{20}}\right)$ ; è facile provare che l'ampiezza di tale intervallo è data da $\prod_{i=1}^{4} f_i^{(8 f_i)}$ dove 4 sono i simboli e 8 è la lunghezza totale della sequenza; pertanto è maggiore di $\frac{1}{2^m}$, con $m > -8 \sum_{i=1}^{4} f_i \log_2 f_i \approx 15.245$ (limite teorico dal teorema di Shannon), e quindi esiste almeno un intero $h$ tale che $\frac{h}{2^m} \in B$, (in questo caso con $h = 55519$ e $m = 16$ risulta $\frac{55519}{2^{16}} = \frac{888304}{2^{20}} \in B$).

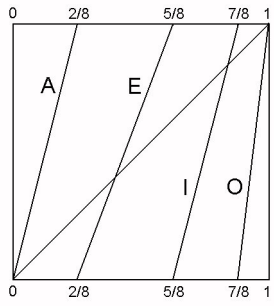
Ne segue che $h$ è codificabile con 16 bit, ovvero si è raggiunto il limite teorico minimo.

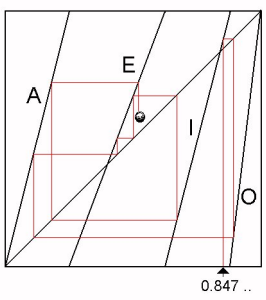
Invertendo il procedimento, note le frequenze ed il numero razionale, si può ricostruire la sequenza originale.

Vi è un curioso modo per riottenere la sequenza, che utilizza il cosiddetto *biliardo frattale* (Barnsley 1993): dato il quadrato di lato 1 si costruiscono le quattro linee (che identificano i quattro caratteri utilizzati) che congiungono i punti sui lati inferiore e superiore di ascissa 0, 2/8, 5/8, 7/8 e 1 (relativi alle frequenze cumulate) e quindi si traccia la diagonale del quadrato. La pallina si muove sul biliardo in orizzontale ed in verticale (rimbalzando se incontra il bordo) secondo le seguenti semplici regole:

-) quando incontra una linea relativa ad un carattere rimbalza in orizzontale

-) quando incontra la diagonale rimbalza in verticale




Partendo dal punto di ascissa uguale a $\frac{55519}{2^{16}} \approx 0.847$ la pallina segue un percorso che la porta a rimbalzare (come per magia) sulle linee relative ai caratteri nell'ordine IOAEEIAE.... !!

## La trasformazione di Burrows-Wheeler.

Concludiamo illustrando un'ultimo metodo che indica un modo per dare una risposta al seguente quesito:

È possibile, data una sequenza, costruirne in modo univoco un'altra, dalla quale si possa poi risalire all'originale, che sia più facilmente comprimibile utilizzando i metodi già visti?

La risposta è positiva ed un modo per costruirla (Burrows-Wheeler 1994) è semplice (misteriosa appare a prima vista l'idea che ha portato alla sua costruzione).

Consideriamo come esempio la solita sequenza IOAEEIAE e generiamo le otto sequenze ottenute ruotando ogni volta la sequenza di un carattere a destra, e riportando l'ultimo all'inizio (figura a sinistra) .

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| I | O | A | E | E | I | A | E |
| E | I | O | A | E | E | I | A |
| A | E | I | O | A | E | E | I |
| I | A | E | I | O | A | E | E |
| E | I | A | E | I | O | A | E |
| E | E | I | A | E | I | O | A |
| A | E | E | I | A | E | I | O |
| O | A | E | E | I | A | E | I |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | A | E | E | I | A | E | I | O |
| 2 | | A | E | I | O | A | E | E | I |
| 3 | | E | E | I | A | E | I | O | A |
| 4 | | E | I | A | E | I | O | A | E |
| 5 | | E | I | O | A | E | E | I | A |
| 6 | | I | A | E | I | O | A | E | E |
| 7 | → | I | O | A | E | E | I | A | E |
| 8 | | O | A | E | E | I | A | E | I |

Ordiniamo poi secondo un certo criterio fissato (ad esempio quello alfabetico) le sequenze ottenute (figura a destra); in tal modo otteniamo sull'ultima colonna a destra una nuova sequenza OIAEAEEI a partire dalla quale (e dal ricordare che la sequenza originale si trova nella settima riga dell'ordine alfabetico) è possibile ricostruire la sequenza originale.

Vediamo infatti come sia possibile, nota la sola sequenza finale, ricostruire la matrice di destra della precedente figura.

Intanto noi conosciamo l'ultima colonna, e quindi anche la prima, che è fatta degli stessi simboli in ordine alfabetico.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | · | · | · | · | · | · | O |
| A | · | · | · | · | · | · | I |
| E | · | · | · | · | · | · | A |
| E | · | · | · | · | · | · | E |
| E | · | · | · | · | · | · | A |
| I | · | · | · | · | · | · | E |
| I | · | · | · | · | · | · | E |
| O | · | · | · | · | · | · | I |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | E | · | · | · | · | · | O |
| A | E | · | · | · | · | · | I |
| E | · | · | · | · | · | · | A |
| E | · | · | · | · | · | · | E |
| E | · | · | · | · | · | · | A |
| I | · | · | · | · | · | · | E |
| I | · | · | · | · | · | · | E |
| O | · | · | · | · | · | · | I |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | E | · | · | · | · | · | O |
| A | E | · | · | · | · | · | I |
| E | E | · | · | · | · | · | A |
| E | I | · | · | · | · | · | E |
| E | I | · | · | · | · | · | A |
| I | · | · | · | · | · | · | E |
| I | · | · | · | · | · | · | E |
| O | · | · | · | · | · | · | I |

Dall'esame di tali colonne si deduce che, essendo ogni riga una rotazione della stessa sequenza, dopo una A può esistere solo una E (entrambe le A dell'ultima colonna sono 'seguite' da una E nella prima colonna della stessa riga); pertanto possiamo aggiungere due E nella seconda colonna. Con lo stesso ragionamento deduciamo che dopo una E (ultima colonna) vi sono E,I,I e quindi possiamo aggiungere nella seconda colonna E,I,I (in ordine alfabetico).

Procedendo in tale modo si finisce la seconda colonna e poi si costruisce la terza, e così via fino alla fine. Ricordando che la sequenza corretta era la settima, si conclude.

La nuova sequenza non appare molto diversa dalla prima, ma questo è dovuto al fatto che è troppo corta e non si notano alcune caratteristiche. Se noi proviamo a trasformare una sequenza più lunga, ad esempio

*Nel-mezzo-del-cammin-di-nostra-vita,-mi-ritrovai-per-una-selva-oscura,-che-la-diritta-via-era -smarrita.-Ahi-quanto-a-dir-qual-era-e'-cosa-dura,-esta-selva-selvaggia-e-aspra-e-forte,*

essa diventa

*eaaae.oel,'onaaaaaala,eel,iairiiaaaaraaa-,rrt-lsririvntvrtrtvvucum----s------t--hdNsssp---m-agc Adahmgvmddvrrreae-eees--maiu-atzfc-nr-s--ieuuepetr-itaooo----aeoistirnsiiqq-cdlllo--ze*

e si possono notare bene come vi siano in essa molte ripetizioni di lettere uguali consecutive, e ciò, come ben sappiamo, rende la sequenza più facilmente comprimibile.

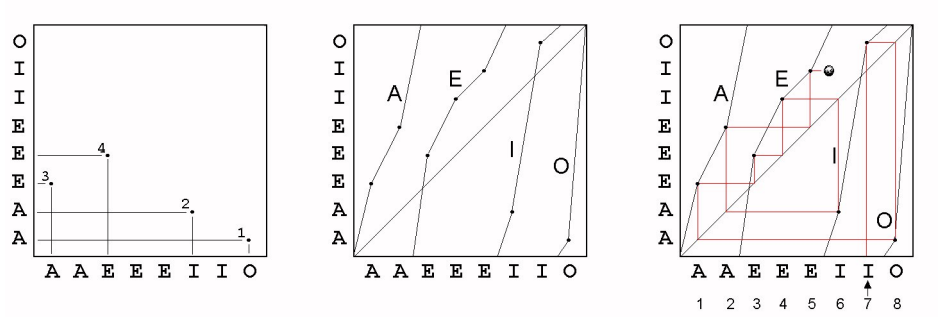
Per finire notiamo come ancora una volta sia possibile utilizzare un biliardo frattale per ricostruire la sequenza di partenza: consideriamo la sequenza finale OIAEAEEI e la stessa in ordine alfabetico AAEEEIIO; da queste generiamo le coppie

(O,A) , (I,A) , (A,E) , (E,E) ....

ottenute prendendo il primo elemento dalla prima ed il secondo dalla seconda.

In un quadrato fissiamo otto punti sui due lati identificati con le lettere AAEEEIIO e consideriamo in successione i punti dati dalle coppie precedenti nel primo posto disponibile (riga e colonna) verso sinistra e verso l'alto.

Congiungiamo poi i punti relativi alle singole lettere con linee crescenti e tracciamo la solita diagonale.




La sequenza è facilmente generata a partire dal 'settimo' punto in ascissa !

## Compressione con perdita di informazioni.

Occupiamoci ora di quei compressori che non conservano esattamente la sequenza dei byte del file, ma la alterano in modo da comprimere meglio il file stesso senza allo stesso tempo modificarne troppo alcune caratteristiche.

Tali compressori sono utilizzati per i file di suono o di video, dove piccole modifiche possono essere trascurabili ai fini degli scopi desiderati.

Per capire meglio cosa sia possibile fare per perdere informazioni senza che la cosa sia troppo evidente, bisogna fare i conti con i limiti dei nostri sensi, limiti per la verità fondamentali, senza i quali sarebbe impossibile fare quello che stiamo per descrivere.

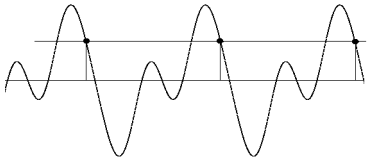
Iniziamo con i file di tipo audio; per quanto riguarda questi, si consideri un'onda sonora come quella rappresentata in figura:

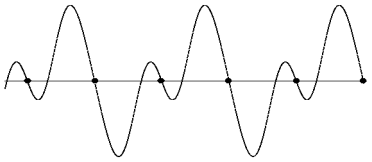
L'ampiezza di tale onda varia con continuità nel tempo, ma ciò che viene fatta è una campionatura

ad intervalli regolari di tempo: vengono cioè rilevati soltanto i valori contrassegnati (figura a destra) con un circoletto; (PCM: Pulse Code Modulation).

È intanto evidente che più valori vengono rilevati, più precisa sarà poi la ricostruzione dell'onda originaria, (e maggiore sarà però lo spazio necessario per memorizzare tali dati).

È inoltre evidente che se si aumenta troppo l'intervallo tra una rilevazione e l'altra, non sarà poi più possibile ricostruire il segnale originario. In altre parole, se il segnale come quello in figura è periodico con una data frequenza (si ripete cioè ogni dato intervallo di tempo), una campionatura fatta con la stessa frequenza (figura seguente a sinistra) non ci fornisce più alcuna informazione sul segnale originario (tutti i valori campionati risultano uguali e pertanto il segnale ricostruito sarà quello costante; si immagini di vedere solo i circoletti).





Per la verità, anche campionando con una frequenza doppia, può verificarsi il caso illustrato nella figura di destra, che ancora non fornisce alcuna indicazione utile per ricostruire il segnale originario, (non è un caso: $\sin(\omega t)$ è periodico di periodo $2\pi/\omega$, ma si annulla con periodo $\pi/\omega$).

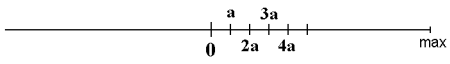
Sarà pertanto necessaria una campionatura a frequenza superiore al doppio di quella originaria, per avere informazioni sufficienti utili.

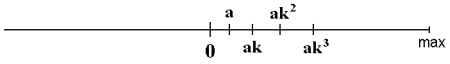
Poiché la massima frequenza udibile dal nostro orecchio è inferiore ai 20 KHz, dovremo quindi campionare con frequenza superiore ai 40 KHz, e per lo standard della musica da registrare sui CD, si è stabilita una frequenza di 44100 campionature al secondo.

I valori così ottenuti formano una successione di numeri, che è poi necessario quantizzare, prima di poterli registrare, sfruttando il fatto che i nostri sensi non rilevano piccole variazioni; in altre parole, se si decide di memorizzarli ciascuno in un byte è necessario riportarli nell'intervallo da 0 a 255, troncando l'eventuale parte decimale.

Poiché 256 valori sono pochi, per ingannare l'orecchio umano, si è deciso di utilizzare 2 byte per ogni valore, ottenendo in tal modo $2^{16} = 65536$ valori differenti. Tali decisione è stata imposta dai limiti della tecnologia al momento della standardizzazione (più di dieci anni fa). Oggi gli audiofili più esigenti sostengono di sentire le imprecisioni dei CD dovute a tali quantizzazioni, e si ritiene che siano necessari almeno 22 bit (quasi 3 byte), più di 4 milioni di valori, per ingannare completamente l'orecchio.

Osserviamo per inciso che la quantizzazione dei valori dovrebbe non essere lineare, ma esponenziale (figura a destra), in quanto la percezione delle differenze di valori non è tanto assoluta quanto relativa: piccole variazioni sono chiaramente percepite se il valore è basso, mentre diventano trascurabili su valori elevati.




In altre parole il cinguettio di un usignolo lontano si sente bene nel silenzio di un bosco, ma diventa difficilmente percettibile se si è vicino ad un martello pneumatico in funzione.

Diventa comunque chiaro che per ogni secondo di audio in stereo sono necessari

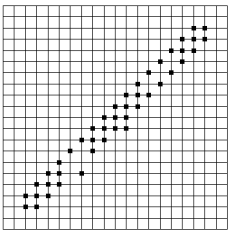
$$44100 \times 2 \times 2 \ = 176400 \text{ byte}$$

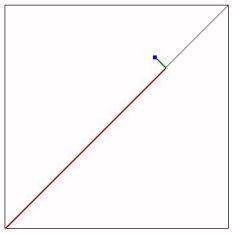
ovvero 44100 campionature per 2 byte per 2 canali.

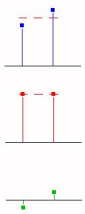
Un'ora di musica richiede perciò 635 MB. Attualmente, con il forte uso di Internet, vanno diffondendosi metodi di compressione di tipo Mpeg Layer-3 (MP3), che riducono lo spazio necessario fino

a 12 volte, con qualità paragonabile a quella del CD, basati su metodi come quelli che illustreremo in seguito per le immagini.

Brevemente osserviamo che, se noi raggruppiamo i dati del file audio ad esempio a due a due, e li riportiamo in un piano cartesiano, i punti (che sono contenuti in un quadrato con le ascisse e le ordinate comprese tra i valori minimi e massimi della quantizzazione) non sono uniformemente distribuiti su tutto il quadrato, ma si concentrano prevalentemente vicino alla diagonale.





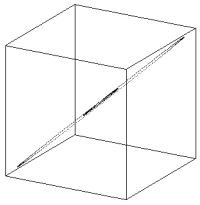


Tutto ciò è molto naturale in quanto ogni dato non è molto diverso dal precedente, cioè in ogni coppia $(x_i, y_i)$ si ha $x_i \approx y_i$.

Questo ci permette di utilizzare meno dati di quelli che ci saremmo potuti aspettare per memorizzare i byte del file; in altre parole basterà identificare ogni punto (figura a destra) anziché con le sue due coordinate $(x, y)$ con quelle ottenute in un sistema di riferimento ruotato di 45 gradi: la prima coordinata rappresenterà il valore medio dei due valori consecutivi, la seconda sarà legata alle differenze dal valore medio (differenze piccole e perciò codificabili con meno bit, oppure addirittura trascurabili).

Unendo le due informazioni sarà possibile riottenere il valore originale, od uno ad esso molto vicino.

Naturalmente maggiore vantaggio si potrà trarre dall'osservare che se noi raggruppiamo i dati a tre a tre e li rappresentiamo nello spazio, in un cubo, tali dati si verranno a concentrare lungo la diagonale del cubo, e pertanto saranno codificabili (dopo un opportuno cambio di coordinate) con tre valori, di cui due decisamente piccoli.



Prima di proseguire osserviamo quanto sia importante avere a che fare con un segnale quantizzato, ovvero con numeri. L'elaborazione del segnale diventa estremamente semplice e flessibile: se indichiamo con $a_n$ il valore dell'ennesimo campionamento, si possono ottenere svariati effetti definendo un nuovo file di dati $b_n$ ad esempio come

| | |
|---|---|
| $b_n = 2a_n$ | volume raddoppiato |
| $b_n = a_{2n}$ | frequenza raddoppiata |
| $b_n = a_n + \frac{1}{2}a_{n-4410}$ | effetto eco: ritardo di 1/10 di sec., ampiezza 1/2 |
| $b_n = \frac{a_{n-1}+a_n+a_{n+1}}{3}$ | riduzione di fruscio, ma suono più cupo. |

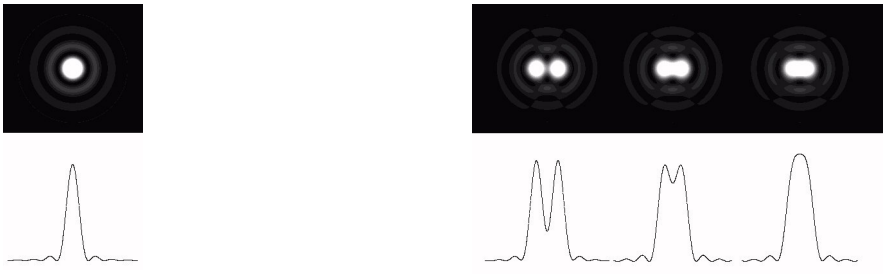
Per ciò che riguarda il video, ricordiamo che i filmati sono registrati come una sequenza di immagini fisse (25 al secondo nello standard televisivo europeo, 30 in quello americano) grazie al fatto che la persistenza dell'immagine nella retina dell'occhio ci permette di far sembrare continuo ciò che continuo non è.

L'immagine fissa (il singolo fotogramma) è quindi scomposto in molti punti, e anche qui è necessario cercare di capire che cosa si può fare, per arrivare ad ingannare l'occhio.

Si consideri intanto che l'occhio contiene il cristallino, che è una lente, e come tutte le lenti ha un potere risolutivo che non è infinito.

Bisogna ricordare che un punto viene messo a fuoco da una lente non in un punto, ma (come ci insegna la teoria ondulatoria della luce) in una figura di diffrazione formata da un disco centrale, di

raggio piccolo, ma non nullo, e da tanti anelli concentrici alternativamente chiari e scuri, di luminosità decrescente e trascurabile. Pertanto due punti molto vicini saranno distinguibili soltanto se i due corrispondenti circoletti centrali nell'immagine virtuale saranno separati, altrimenti si noterà soltanto un'unica macchia.



Dalla teoria si ottiene che il potere risolutivo di una lente, ovvero il minimo angolo visivo al di sotto del quale non si distinguono più separati due oggetti è dato (in radianti) da

$$\theta = 1.22\ \frac{\lambda}{D}$$

ove $\lambda$ è la lunghezza d'onda della luce e $D$ è il diametro della lente.

Tale valore corrisponde alla situazione di destra della figura precedente, quando la massima luminosità di un punto cade nel primo minimo dell'altro.

In particolare, per $\lambda = 5.6 \cdot 10^{-7}$ m (luce visibile da $4 \cdot 10^{-7}$ m per il violetto a $7 \cdot 10^{-7}$ m per il rosso) e per $D = 2.5$ mm (diametro medio della pupilla, da 1.5 a 5 mm, a seconda delle condizioni di illuminazione) si ottiene

$$\theta \approx \frac{1}{3660}\text{rad} \ \approx\ 1 \text{ primo di grado.}$$

Questo, ad una distanza di 15 mm (la distanza focale dell'occhio) corrisponde a 4 $\mu$m, che è circa la dimensione delle cellule sensibili sulla retina; cioè la qualità delle lenti e l'accuratezza della distribuzione delle cellule nervose sono proprio quelle necessarie per utilizzare il sistema fino ai limiti posti dalla diffrazione.

Ad una distanza di 3 m, corrisponde a circa .82 mm, che è circa 1/520 dell'altezza dello schermo di un televisore da 28 pollici (la diagonale è $28 \times 2.54 = 71.1$ cm, e l'altezza ($\frac{3}{5}$ della diagonale) è circa 42.7 cm).

Questo spiega perché la risoluzione televisiva dello standard PAL (Phase Alternating Line) è stata fissata a 576 righe attive (visibili).

Tenuto conto delle proporzioni dello schermo (4/3) ne conseguono 768 punti in orizzontale, e questo è stato il primo standard Kodak per le foto digitali; le attuali telecamere digitali di tipo DV utilizzano il formato 720×480.

Tale risoluzione, pur non essendo molto elevata, basta per la digitalizzazione dei fotogrammi di un video (abbiamo sempre visto la televisione con tale risoluzione e, se la trasmissione è esente da disturbi di altro tipo, la qualità è più che soddisfacente).

Diverso è il caso di immagini fisse (fotografie). Se noi vogliamo fare un ingrandimento di una foto, allora la risoluzione richiesta deve essere decisamente maggiore.

Un semplice calcolo mostra che, alla minima distanza di messa a fuoco, che per un individuo adulto è in media di 25 cm, l'occhio separa due punti alla distanza di 0.07 mm ( 250/3660 , ovvero circa 1/370 di pollice).

Questo spiega il perché le stampanti lavorino intorno ad una risoluzione di 300 dpi (dot per inch: punti a pollice) almeno sul colore.

A questa risoluzione una stampa su di un formato A4, più o meno un ingrandimento $20 \times 30$, richiede circa $3500 \times 2300$ punti.

Va però tenuto presente che le attuali stampanti a getto di inchiostro sono in grado di stampare, per ogni punto (1/300 di pollice), un punto di colore ciano, magenta, giallo, o nero, di intensità fissa

(generando i colori con la tecnica sottrattiva, e quindi utilizzando i colori complementari a quelli del rosso, verde e blu).

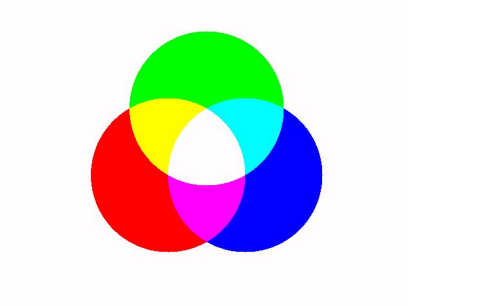
In parole povere, il punto è o non è colorato con l'inchiostro, ma non sono in grado di dosare l'intensità del colore del punto. Le varie sfumature sono quindi generate utilizzando matrici di punti colorati o no: ogni punto dell'immagine è quindi trasportato sul foglio come una matrice di $2 \times 2$, $3 \times 3$ o più punti, ottenendo in tal modo, per ogni colore, 4, 9 o più gradazioni, a seconda del numero di punti colorati.

È pertanto inutile avere tutta quella risoluzione (metà o un terzo è già più che sufficiente), perché oltre a sprecare grande quantità di memoria, si corre il rischio di veder ridotta d'ufficio dal software di stampa la quantità di dati utilizzata (senza che ci se ne accorga).

Fatte queste precisazioni, osserviamo ancora che, per quel che riguarda la memorizzazione e la visione dei filmati, ciò che viene richiesto al compressore ed al decompressore è, oltre al fatto di essere il più efficiente possibile (comprimere più che si può), anche il fatto di essere di veloce esecuzione, ovvero semplice (ogni immagine deve essere codificata e decodificata in 1/25 di secondo).

Notiamo infine che se l'utilizzo di pochi colori, come in un disegno, ci permette di utilizzare codifiche semplici, come la RLE o il PCX, in un'immagine con molti dettagli, dove il colore di un punto è quasi sicuramente differente da quello vicino, anche se per qualche piccola sfumatura, tali decodifiche corrono il rischio di generare file più lunghi di quelli originali, costituiti dalla semplice trascrizione sequenziale byte dopo byte, (formato BMP: BitMaP).

Ma torniamo alla codifica di un'immagine.



Intanto, ad ogni punto è possibile associare un colore. Per capire poi come siano codificabili i colori si ricordi che, come mostra l'immagine a fianco, è possibile utilizzare solo tre fasci di luce rossa, verde e blu, (red, green, blue o RGB) di intensità variabile per ottenere ogni colore.

Pertanto, ad ogni punto di un immagine è possibile associare tre numeri, corrispondenti all'intensità di ognuno dei colori fondamentali. Il range di tali numeri è di solito scelto nell'intervallo da 0 a 255, in modo da attribuire un byte per ogni valore ( (0,0,0) sarà il nero, (255,255,255) il bianco); in tal modo si ottengono 256 gradazioni di sfumature per ogni colore fondamentale (quanto basta perché l'occhio non le distingua), che generano $256^3 = 16777216$ colori differenti, più che sufficienti per immagini di qualità fotografica (in molti casi 64 gradazioni danno già ottimi risultati, come era nello standard VGA).

Notiamo per inciso anche qui che, come nel caso dell'audio, l'avere a disposizione dei valori numerici permette di modificare facilmente l'immagine; se $a_{i,j}$ rappresenta la luminosità del punto di coordinate $(i, j)$, definendo una nuova immagine $b_{i,j}$ si ha ad esempio:

| | |
|---|---|
| $b_{i,j} = ca_{i,j} + l$ | modifica di contrasto $c$ e luminosità $l$ |
| $b_{i,j} = \frac{a_{i,j} + a_{i-1,j} + a_{i+1,j} + a_{i,j-1} + a_{i,j+1}}{5}$ | riduce disturbi, sfuoca |
| $b_{i,j} = 5a_{i,j} - a_{i-1,j} - a_{i+1,j} - a_{i,j-1} - a_{i,j+1}$ | evidenzia dettagli |
| $b_{i,j} = .5(a_{i+1,j} - a_{i,j}) + k$ | traccia contorni, effetto bassorilievo |

Da quando prima detto si deduce quindi che, per un secondo di solo video, saranno necessari

$$720 \times 480 \text{ punti } \times 3 \text{ byte } \times 25 \text{ fotogrammi } = 25920000 \text{ byte}$$

ovvero circa 25 MB (una discreta quantità).

Ci occuperemo pertanto della compressione di immagini video, e non di file audio, in quanto tra i due il primo è certamente il problema più pressante, data l'enorme quantità di dati utilizzati.

Prima di proseguire però facciamo un momento il punto della situazione attuale:

vi sono due aspetti da tenere ben presente in queste situazioni: la quantità di dati memorizzabili sui supporti di massa che vogliamo utilizzare (nastri, CD, ecc.) e la velocità di trasmissione dei dati da tali supporti al video e viceversa.

Per quanto riguarda la capacità dei supporti si tenga presente che allo stato attuale un CD-ROM (per utilizzo audio o memoria dati) può contenere circa 650 MB; un DVD può variare tra i 4.7 GB ed i 17 GB (nei più recenti e costosi, doppio strato, doppia faccia); un nastro mini-DV da un'ora contiene 13 GB.

Ora gli standard televisivi utilizzati nel mondo sono di fatto due: il PAL (europeo, 720×576 punti, per 25 fotogrammi al secondo) ed il NTSC (americano, 640×480 punti, per 29.97 fotogrammi al secondo).

Dalle considerazioni prima fatte appare chiaro che se non si utilizza nessuna forma di compressione, su di un CD da 640 MB possiamo registrare circa 26 secondi di filmato (un po' troppo poco).

Prima di arrendersi bisogna però tenere presente alcune altre cose: un normale registratore VHS (che tutto sommato, quando la registrazione è ben fatta, fornisce immagini di ottima qualità) utilizza circa 240 linee.

Questo è dovuto anche al fatto che le trasmissioni televisive inviano i singoli fotogrammi in modo interallacciato; cosa significa ciò: bisogna considerare che su di uno schermo televisivo l'immagine è resa dal fatto che i fosfori eccitati dal fascio di elettroni diventano luminosi; tale luminosità però non rimane molto a lungo e nei primi televisori, quando il raggio di elettroni arrivava vicino al fondo dello schermo (dopo circa 1/25 di sec.) i fosfori in alto erano già diventati scuri, rendendo le immagini sfarfallanti.

Per evitare ciò la soluzione trovata è stata quella di dividere ogni fotogramma in due, l'uno fatto con le righe dispari (trasmesso perciò in 1/50 di sec.) e l'altro con le righe pari; in tal modo l'intero schermo viene percorso due volte nel tempo di un fotogramma, annullando il problema.

Tale procedimento non è più necessario negli attuali monitor per computer.

Per quel che riguarda la velocità di trasmissione dei dati, altro grosso problema, si consideri che un modem riceve a 7 KB/sec (57.6 Kb/sec), un lettore di CD-ROM di qualche anno fa (a singola velocità) legge i dati a circa 176 KB/sec (quanto abbiamo visto essere necessario per i file audio non compressi); un lettore di DVD si situa intorno ad 1 MB/sec, un registratore o telecamera digitale mini-DV trasmette a circa 3.5 MB/sec.

Sempre comunque troppo poco rispetto ai circa 25 MB/sec necessari.

Si sono pertanto sviluppati molti tipi di compressioni, alcuni dei quali sono poi diventati gli attuali standard; tra questi citiamo

| standard | risoluzione | velocità | utilizzo | compressione |
|---|---|---|---|---|
| MPEG-1 | 352×240 punti | 0.01-0.06 MB/sec | CD-ROM, Web | $\approx 150$ |
| MPEG-2 | 720×480 punti | 0.01-2 MB/sec | DVD - TV Satellitare | $\approx 25$ |
| DV | 720×480 punti | 3.5 MB/sec | telecamere digitali | $\approx 8$ |

Osservando le specifiche dell'MPEG-1 (Mpeg = Moving Picture Experts Group) si nota come una prima soluzione possa essere quella di ridurre il numero di punti nel fotogramma: con 320×240 si hanno un quarto dei punti presenti in 640×480, e se si riduce il numero di fotogrammi al secondo (da 30 a 15) si raggiungono facilmente compressioni pari ad 8 volte.

Ma questa non è la soluzione migliore, perché vedremo poi che è difficile ricostruire l'immagine originale senza difetti troppo visibili, mentre se si usano meno fotogrammi al secondo il filmato appare a scatti, almeno nelle sequenze veloci.

Appare chiaro che è necessario utilizzare una qualche forma di compressione dei singoli fotogrammi, che per il DVD è ancora più spinta, naturalmente a scapito della definizione dell'immagine. Infatti non vengono registrati interamente tutti i singoli fotogrammi, ma per alcuni viene registrata solo la differenza tra di essi, contando sul fatto che molto spesso un fotogramma è molto simile a quello precedente.

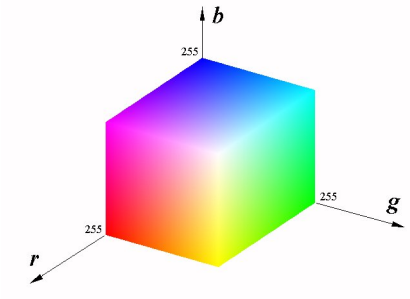
In pratica vi sono tre tipi di frame, quelli di tipo I (intraframe) che sono registrati per intero, quelli di tipo P (predicted) che sono ricavati dalle differenze con i precedenti, e quelli di tipo B (bi-

directional) che sono ottenuti sia dai precedenti che dai successivi; un esempio di sequenza standard può essere IBBPBBPBBPBBI...

In alcuni casi però, rapidi spostamenti di un oggetto, zoomate o anche soltanto una scena con pioggia o un primo piano di acqua in movimento, rende ogni fotogramma molto diverso da quelli vicini, e questo produce un filmato non gradevole in quanto dovendo salvare più dati per il singolo fotogramma, ma restare nella velocità di trasmissione stabilita, si deve poi saltare qualche fotogramma successivo.

Un altro problema relativo al DVD risiede nel fatto che per avere un singolo fotogramma bisogna conoscere anche quelli immediatamente precedenti (o seguenti) e ciò rende gravoso l'utilizzo di memoria e particolarmente difficile l'editing del video sui singoli quadri; va comunque ricordato che tale standard è per ora offerto a persone che utilizzano il video digitale solo come utenti finali, per la visione di film, Tv da satellite, ecc. e non come produttori.

Ma torniamo al nostro problema originale:




L'immagine a fianco riproduce il cubo dei colori, ove sui tre assi sono posti i tre colori base. Quelle che si vedono sono le tre facce corrispondenti alle massime intensità; man mano che si scende all'interno del cubo verso l'origine delle coordinate, nel vertice opposto al bianco, si trovano gli stessi colori della superficie gradatamente più scuri fino al nero (nell'origine degli assi).

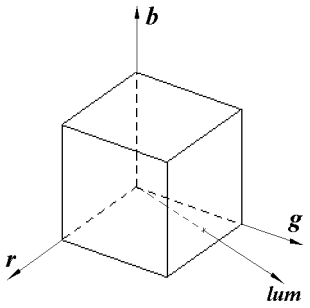
A questo punto va considerato che l'occhio non è ugualmente sensibile ai vari colori, o meglio i coni (più concentrati al centro della retina) ed i bastoncelli (più diffusi sul bordo) che sono rispettivamente predisposti alla percezione dei colori e della luminanza (l'intensità del bianco e nero), rendono l'occhio più sensibile alle variazioni di luminanza che a quelle del colore.

Essendo la luminanza definita come

$$l = .299r + .587g + .114b$$

ove si sono indicate con $r, g, b$ rispettivamente le intensità di rosso, verde e blu, si deduce che più importanti sono le variazioni del verde, meno quelle del rosso e meno ancora sono percepite quelle del blu. In altre parole, sebbene si siano suddivisi in egual modo (256 valori) gli intervalli dei campi dei tre colori, sarebbero sufficienti meno valori per il blu, rispetto a quelli per il rosso e per il verde.

Questo è il motivo per cui, ad esempio, nelle vecchie schede grafiche a soli 256 colori (dove cioè ogni colore era rappresentato da un solo byte, ovvero 8 bit), 3 bit (8 sfumature) venivano riservati al verde ed al rosso, e solo 2 bit (4 sfumature) al blu .




È quindi opportuno utilizzare un sistema di coordinate che abbia come primo asse quello lungo la direzione $(.299, .587, .114)$ cioè l'asse della luminanza. Per quel che riguarda gli altri due assi si potrebbero scegliere come logico due direzioni ortogonali, ma può essere più utile scegliere ad esempio le direzioni $(.5, -.5, 0)$ e $(0, -.5, .5)$, (corrispondenti a $.5(r - g)$ e $.5(b - g)$ ) in modo da avere componenti nulle nel caso di figure in bianco e nero, dove $r = g = b$.

Ad ogni punto pertanto verrano quindi attribuiti tre nuovi valori, ottenuti dalla trasformazione

$$\begin{cases} l = .299r + .587g + .114b \\ c_r = .5r - .5g \\ c_b = -.5g + .5b \end{cases}$$

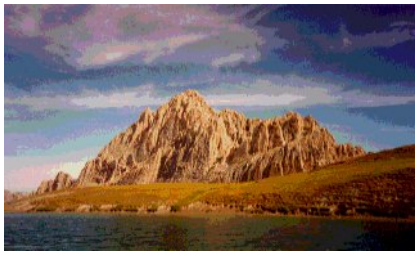
e quindi si potrà pensare ogni immagine come la composizione di tre immagini: quella in bianco e nero, relativa alla prima coordinata $l$ (luminanza), e le altre due relative alle due crominanze $c_r$ e $c_b$.

Vediamo ora come si può studiare la singola immagine, ricordando che per la prima componente è necessaria una buona precisione, mentre per le altre ne basta di meno: in altre parole il range di valori interi per la luminanza può andare da 0 a 255 mentre, senza perdere molto in qualità, per le altre due componenti è possibile scendere a valori più bassi, ad esempio da 0 a 128 o oltre; alternativamente si può ridurre per la crominanza la definizione dell'immagine ad esempio dimezzando le dimensioni orizzontali e verticali. Ciò è fatto praticamenti da tutti gli standard; ad esempio nel DV è fissato a priori un rapporto di compressione di 5 a 1, e poi luminosità e colore vengono memorizzati secondo lo schema 4:1:1, che significa: ogni quattro valori della luminanza viene trasmesso un solo valore per le due crominanze, portando così la compressione vicino ad 8.

Le seguenti figure mostrano l'immagine originale qui trattata e le scomposizioni nei tre colori rosso, verde e blu.




Sono sotto riprodotte le immagini ottenute riducendo i valori di rosso, verde e blu, di un fattore 8 e 32 rispettivamente.

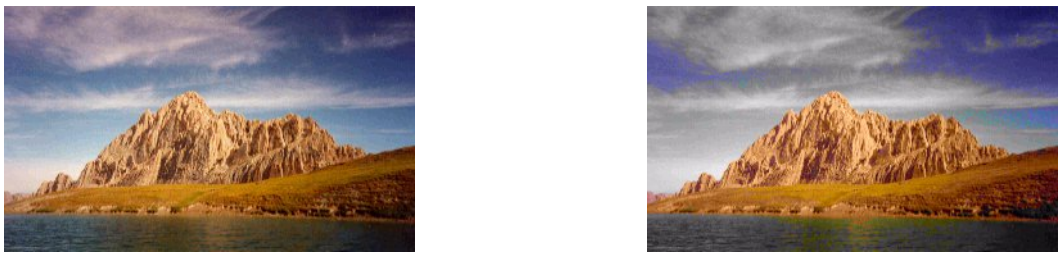



Si nota nella figura di destra (dove sono usati solo 8 valori per i vari colori, invece dei soliti 256) un effetto tipo 'dipinto' specialmente in quelle aree dove vi sono piccole variazioni di colori simili (ad esempio nel cielo).

Le seguenti figure mostrano la figura originale e le scomposizioni (a gradazioni di grigio) in luminosità e le due crominanze (rosso e blu).

Si noti come le immagini relative alle crominanze contengano meno informazioni visibili.

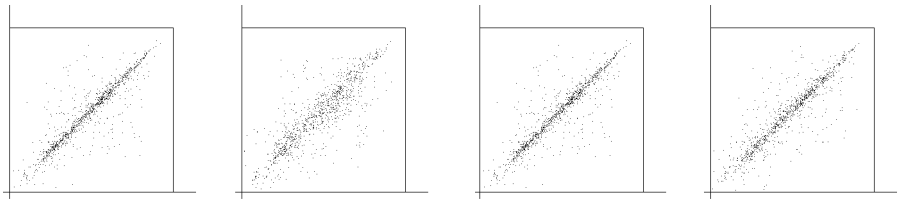
Le prossime due figure mostrano invece la stessa immagine ove si sono ridotti i valori della luminosità e del colore di un fattore rispettivamente 8 e 8, 8 e 32.



Si noti come nella figura di destra, nonostante si usino solo 8 valori per le crominanze, l'immagine non risulti sgradita alla vista, seppure con qualche errore nella saturazione del colore, in più o in meno, in alcuni punti.

Suddividiamo ora l'immagine in tanti quadrati (ad esempio di 8 punti di lato per semplicità) e consideriamo questa parte di immagine come una matrice $8 \times 8$, ovvero come un elemento di uno spazio vettoriale di dimensione pari a 64.

Una base canonica per tale spazio è quella costituita dalle matrici aventi 1 al posto $(i, j)$ e 0 in tutti gli altri elementi; ma anche qui facili considerazioni mostrano che i punti corrispondenti alle immagini sono per lo più concentrati lungo la direzione avente come coordinate tutti uno (o 1/64, la direzione del valor medio, ovvero la diagonale principale del cubo, in quanto punti vicini hanno spesso luminosità simile), come mostra il grafico seguente che riporta alcune viste bidimensionali della distribuzione dei punti luminosità nel cubo a 64 dimensioni, per la figura in seguito esaminata; e sono invece raramente posti intorno a quelle direzioni che alternano in punti vicini 1 e -1 (bianchi e neri disposti a scacchiera).



La direzione del valor medio può quindi essere assunta come uno degli assi (il più importante) di un nuovo sistema di coordinate.

Si veda in appendice come sia possibile definire facilmente una base ortogonale nello spazio delle matrici $n \times n$ che rispecchi quanto sopra detto (e quali sono pure le basi utilizzate ad esempio dal JPEG : Joint Photographic Experts Group).

Si osservi ad esempio come un quadrato in tinta unita, cioè una matrice fatta di 64 valori tutti uguali ad $a$ si trasformi con la suddetta rotazione, in un punto di prima componente uguale ad $a$ (il valore medio) e tutte le altre nulle (che quindi possono non essere memorizzate in quanto forniscono informazione nulla).

Come semplice esempio si consideri il caso di un quadrato 2×2 i cui punti assumano i valori $a, b, c, d$ ; naturalmente si potrebbe scomporre

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} a & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & b \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ c & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & d \end{pmatrix}$$

ma utilizzando quanto sopra osservato la decomposizione si ottiene determinando quattro valori $w, x, y, z$ in modo che il quadrato si possa ottenere come somma di quattro quadrati

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} w & w \\ w & w \end{pmatrix} + \begin{pmatrix} x & -x \\ x & -x \end{pmatrix} + \begin{pmatrix} y & y \\ -y & -y \end{pmatrix} + \begin{pmatrix} z & -z \\ -z & z \end{pmatrix}$$

cioè risolvendo il sistema

$$\begin{cases} w + x + y + z = a \\ w - x + y - z = b \\ w + x - y - z = c \\ w - x - y + z = d \end{cases}$$

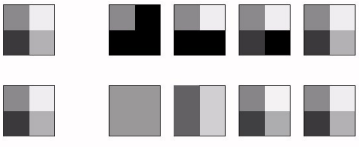
Ad esempio

$$\begin{pmatrix} 140 & 240 \\ 60 & 180 \end{pmatrix} = 155 \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} - 55 \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} + 35 \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} + 5 \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

mentre se avessimo utilizzato la base canonica:

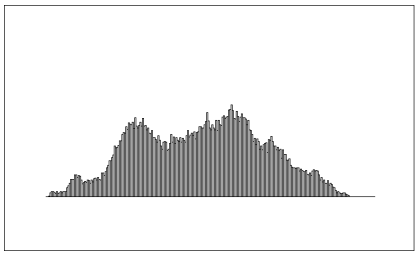
$$\begin{pmatrix} 140 & 240 \\ 60 & 180 \end{pmatrix} = 140 \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + 240 \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + 60 \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + 180 \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

e la figura seguente mostra a sinistra il quadrato originale, in scala di grigi, e quindi la sequenza di approssimazioni, dal primo quadrato, alla somma dei primi due, fino alla somma di tutti e quattro, nei due casi (scelta canonica o con assi ruotati).

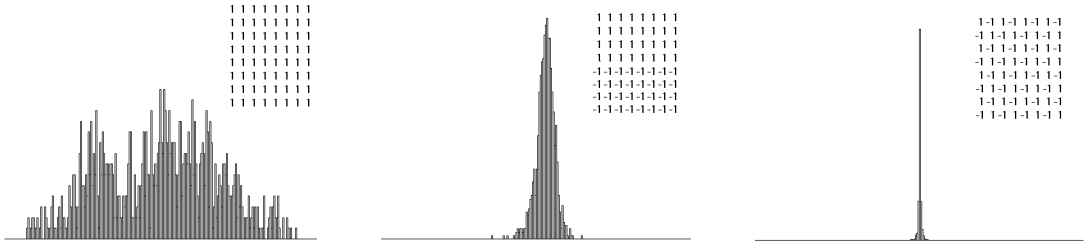


Si noti come, nel secondo caso, già alla terza approssimazione si abbia un quadrato quasi indistinguibile dall'originale.

Il grafico seguente a destra (non è il profilo di una montagna) mostra la distribuzione dei coefficienti per la luminosità della figura a sinistra, nel caso della scelta della base canonica (relativa ad un qualunque elemento della base).

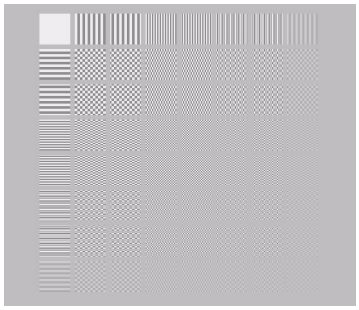


I tre grafici seguenti mostrano invece la distribuzione (non normalizzata) dei coefficienti nel caso della scelta di un'altra base (è indicata su ogni grafico la matrice corrispondente all'elemento della base).



Si noti come la distribuzione sia abbastanza ampia solo per quel che riguarda il valore medio (prima componente), mentre diventi fortemente concentrata su piccoli valori per le altre direzioni (già nel secondo caso, che rappresenta la seconda base in ordine di *importanza*).

Si noti pure che, per di più, l'occhio è sempre meno sensibile a quelle direzioni che alternano in posti vicini valori di -1 ed 1 (ovvero una sequenza di punti neri e bianchi messi a scacchiera verrà interpretata comunque dall'occhio come un grigio).



La figura a fianco illustra questo fatto: sono riportati 64 quadrati (ciascuno $8 \times 8$) che si differenziano dal grigio uniforme in egual misura, lungo le direzioni del nuovo sistema di coordinate.

Si può chiaramente vedere come sia evidente la differenza sul valor medio (primo a sinistra in alto) ove tutti i punti differiscono di un valore prefissato; mentre tale differenza diminuisce verso destra e verso il basso fino a diventare quasi impercettibile nell'ultimo in basso a destra (scacchiera di -1 e 1).

Si tenga tra l'altro presente che tutti i 64 quadrati differiscono dal grigio uniforme in maniera uguale secondo la norma quadratica, ovvero (indicato con $d_i$ la differenza nel punto $i-$esimo)

$$\sum_{i=1}^{64} d_i^2 = 64$$

essendo, per ogni $i$, $|d_i| = 1$, e ciò mostra come tale valutazione dell'errore, che è quella usata normalmente per valutare la differenza tra due immagini, non rispecchi quella che è la nostra percezione visiva di errore.

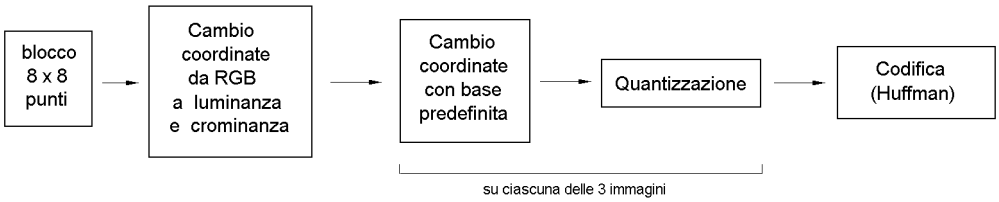
Sarà pertanto possibile quantizzare in maniera più pesante i coefficienti relativi alle ultime coordinate senza perdere in qualità.

Ciò è fatto moltiplicando tali valori per un coefficiente (minore di uno) prima di memorizzarne il valore dell'intero più vicino. La matrice corrispondente a tali valori è detta matrice di quantizzazione, e quella relativa alle crominanze, se non si è già ridotta la definizione dell'immagine, potrà contenere valori più piccoli di quella della luminanza.

Osserviamo che è soltanto in questo passaggio che si ha la perdita di informazione che permette il maggiore compattamento dei dati.

Va infine notato che dopo tale quantizzazione di solito, se si ordinano i vettori della base da quelli più *importanti* a quelli meno, molti coefficienti relativi agli ultimi vettori diventano nulli e pertanto non vengono poi memorizzati.

Lo schema seguente riassume tutto il processo utilizzato per codificare un'immagine.




Si rimanda come già detto all'appendice per quel che riguarda possibili scelte di basi predefinite.

Si possono in tal modo ottenere compressioni anche molto elevate: le due immagini sottostanti mostrano il risultato di una compressione di 7/1 e di quasi 25/1; si evidenzia nella figura di destra una certa quadrettatura dell'immagine. (Si noti che tali compressioni sono state ottenute su immagini in bianco e nero, e pertanto solo sulla luminanza; sarebbero molto superiori su immagini a colori dove i valori della crominanza sono maggiormente compressi)

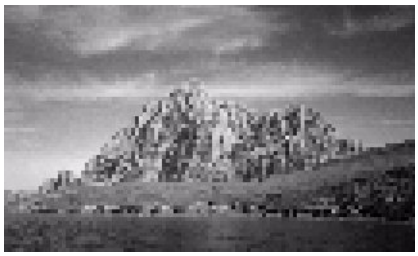
Per concludere osserviamo che spingendo al massimo la compattazione dei dati, cioè considerando soltanto il coefficiente relativo al primo elemento della base (vettore valor medio) ed annullando tutti gli altri, si arriverebbe ad avere una compressione di 64/1, con un'immagine a quadretti $8 \times 8$ di intensità costante.

Questo è quello che ugualmente si otterrebbe riducendo la definizione sia orizzontale che verticale dell'immagine di 8 volte e quindi dilatando l'immagine così ottenuta per riportarla alle dimensioni originali.

Quest'ultimo metodo è certamente il modo più semplice per ridurre i dati necessari alla memorizzazione di un'immagine; senza arrivare a valori così alti, si può ridurre la definizione di 2 o 3 volte (sia in orizzontale che in verticale) ottenendo compressioni di 4/1 o 9/1.

L'immagine può poi essere ricostruita banalmente facendo diventare ogni punto un quadrato di dimensioni $2 \times 2$ o $3 \times 3$, ma tale immagine risulta però visibilmente quadrettata e fastidiosa a vedersi.

Perfezionando il metodo di espansione dell'immagine si possono approssimare i valori intermedi con funzioni lineari (o bilineari), ma anche qui la figura risultante appare decisamente sfuocata.



Le due figure precedenti riportano le immagini ottenute nei due modi sopra citati (espansione costante o interpolazione bilineare).

Naturalmente vi sono infinite altre possibilità per ridurre ulteriormente le dimensioni di un file immagine.

Si può per esempio ridurre come già osservato la risoluzione orizzontale e verticale di un fattore 2 o 4 o più e quindi, dopo aver riprodotto l'immagine originale con uno dei metodi sopra visti, memorizzare solo le differenze tra quella ottenuta e l'immagine di partenza. Essendo tali differenze di solito minime, si possono ottenere alti rapporti di compressione.

Un'altra possibilità può essere offerta dalle seguenti considerazioni: se si osservano le immagini relative alla crominanza si nota che esse non sono poi molto dissimili da quella della luminanza (almeno in immagini fotografiche); un esame della distribuzione dei valori in vari quadretti 8×8 dell'immagine rivela una certa dipendenza della crominanza dalla luminanza e questo può facilmente essere sfruttato per comprimere ancora il file.

Ma queste come già detto sono solo alcune delle possibilità.

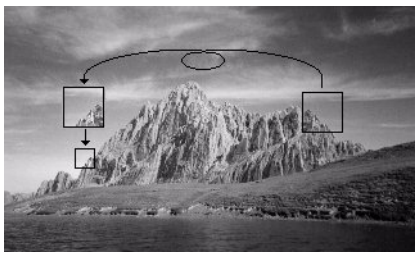
## La compressione frattale

Per ultimo ci occupiamo di una tecnica decisamente interessante, per la sua diversità dai metodi fin qui esposti: la tecnica frattale.

Tale metodo si basa sull'osservazione che parti delle immagini analizzate sono di solito simili ad altre parti dell'immagine stessa.

Anche tale metodo è operativamente molto semplice e può essere espresso nel seguente modo:

-) l'immagine viene divisa in quadretti di dimensione fissata (ad esempio $4 \times 4$ punti) e per ognuno di questi si cerca, nell'intera figura (o in parte di essa) quel quadrato di lato doppio, che ridotto del 50%, eventualmente ruotato o riflesso in uno degli 8 modi possibili, meglio approssima (in norma quadratica) attraverso una modifica lineare, quello dato (la luminosità $x$ di un punto viene trasformata in $cx + l$, ottimizzando sulla variabile $l$ o su entrambe $c$ ed $l$).



Per ognuno di tali quadretti viene poi memorizzata la posizione del quadrato origine ed i dati della trasformazione lineare.

Il decompressore applicherà poi tali trasformazioni iterativamente e la figura ottenuta dopo alcune iterazioni, se $|c| < 1$, è molto simile a quella originaria.

Si rimanda all'appendice per una giustificazione teorica del metodo.

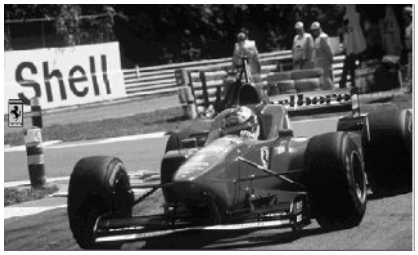
Si ottengono in tal modo compressioni abbastanza buone: la seguente immagine è stata riprodotta dopo una compressione 7/1, (utilizzando per $c$ il valore fisso 0.8)

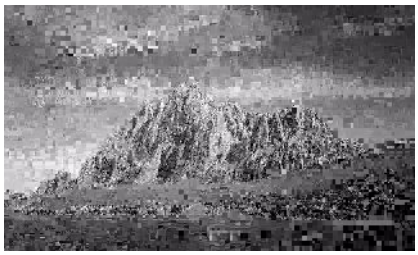
Ad una grande semplicità di programmazione, fa purtroppo riscontro una grande lentezza del processo di compressione, dovuta alla grande quantità di confronti necessari per scegliere il quadrato migliore, per ogni quadratino in cui viene suddivisa l'immagine.

In compenso risulta molto veloce la decompressione; per questo tale metodo non può essere usato per filmati, laddove la compressione deve essere eseguita in tempo reale, ma è utilizzato per memorizzare figure la cui compressione viene fatta una volta per tutte su sistemi potenti e veloci, come ad esempio per raccolte di immagini su CD, enciclopedie (Encarta), ecc.

Va notato che tale metodo risulta paragonabile a quelli precedentemente esposti (tipo JPEG) e quasi migliore per compressioni molto elevate.

La seguente sequenza mostra le prime 6 iterazioni della decompressione frattale, che può essere eseguita, come già osservato, a partire da una qualunque immagine; nel caso mostrato dall'immagine di una Ferrari:



Un altro aspetto molto interessante di tale metodo, ultimo, ma non per importanza, risiede nel fatto che, utilizzando solo similitudini, la decompressione è indipendente dalla scala, per cui se noi ricostruiamo la figura con dimensioni doppie, utilizzando le stesse similitudini, otteniamo un'immagine ancora gradevole, non quadrettata, né tanto sfuocata, con un rapporto di compressione 4 volte superiore, e quindi per il caso precedente di 28/1.

Le immagini seguenti mostrano a sinistra una parte dell'immagine (ingrandita) ottenuta raddoppiando le dimensioni con la tecnica frattale ed a destra l'immagine corrispondente JPEG con rapporto di compressione simile.

Ancora un esempio, con un'altra immagine; a sinistra quadruplicando le dimensioni con tecnica frattale, a destra immagine JPEG con analoga compressione (data la poca complessità dell'immagine si raggiunge in tal modo una compressione di 236/1 !).

Si osservi per concludere, che la trattazione qui fatta, essendo i problemi affrontati discreti, utilizza solo semplici nozioni sugli spazi vettoriali di dimensione finita, e non fa volutamente uso di strumenti più sofisticati quali le trasformate di Fourier (discrete o meno), che comunque si riducono poi al caso qui trattato.

Notiamo pure che con la sempre più grande diffusione di Internet, si vanno diffondendo altri ottimi tipi di compressori, basati su codifiche di tipo MPEG-4 che permettono di raggiungere ottime compressioni conservando un buon livello di qualità: a risoluzione 320×240 punti, con 25 fotogrammi al secondo e audio compresso con MP3, si possono registrare su di un normale CD-ROM, da 30 ad 80 minuti di filmato.

Contemporaneamente si evolvono sempre di più gli stessi compressori: l'MPEG-7, attualmente in corso di definizione, usa fra l'altro tecniche frattali.

## Appendice.

Per quel che riguarda la possibilità di definire delle basi nello spazio delle matrici $n \times n$ si noti che

• *Se* $x_i,\ i = 1..n$ *sono* $n$ *vettori ortogonali in* $\mathbb{R}^n$*, allora i* $2n$ *vettori di* $\mathbb{R}^{2n}$ *definiti da* $(x_i, \pm x_i),\ i = 1..n$ *sono ortogonali.*

Dim. Si ha $(x_i, x_i) \cdot (x_i, -x_i) = 0$ per ogni $i$, mentre $(x_i, \pm x_i) \cdot (x_j, \pm x_j) = 0$ per ogni $i \neq j$ (essendo $x_i \cdot x_j = 0$)

Esempio di $n$ vettori ortogonali in $\mathbb{R}^n$:
per $n = 1$

$$1$$

per $n = 2$

$$(1,1) \quad , \quad (1,-1)$$

per $n = 4$

$$(1,1,1,1) \quad , \quad (1,1,-1,-1) \quad , \quad (1,-1,1,-1) \quad , \quad (1,-1,-1,1)$$

per $n = 8$

$$\begin{array}{rl} (1,1,1,1,1,1,1,1) & , \quad (1,1,1,1,-1,-1,-1,-1) \\ (1,1,-1,-1,1,1,-1,-1) & , \quad (1,1,-1,-1,-1,-1,1,1) \\ (1,-1,1,-1,1,-1,1,-1) & , \quad (1,-1,1,-1,-1,1,-1,1) \\ (1,-1,-1,1,1,-1,-1,1) & , \quad (1,-1,-1,1,-1,1,1,-1) \end{array}$$

**Osservazione.** Si noti che i vettori sopra descritti possono essere ordinati in modo che abbiano esattamente $i$ transizioni da -1 a 1 o viceversa, con $i = 0, .., n-1$.

• *Siano* $x_i,\ i = 1..n$*,* $n$ *vettori ortogonali in* $\mathbb{R}^n$*, allora le* $n^2$ *matrici definite da*

$$(A_{p,q})_{i,j} = (x_p)_i\ (x_q)_j \qquad , \qquad p, q = 1..n\ ,\ i, j = 1..n$$

*sono ortogonali (nello spazio* $\mathbb{R}^{n \times n}$*)*

Dim. Si ha

$$A_{h,k} \cdot A_{p,q} = \sum_{i,j} (x_h)_i (x_k)_j (x_p)_i (x_q)_j = \sum_j (x_k)_j (x_q)_j \ \sum_i (x_h)_i (x_p)_i = (x_k \cdot x_q)(x_h \cdot x_p)$$

e, ricordando che i vettori erano ortogonali, tale prodotto è zero se $(h, k) \neq (p, q)$.

Esempio, nel caso $n = 2$, di quattro matrici ortogonali

$$A_{1,1} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad , \quad A_{1,2} = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} \quad , \quad A_{2,1} = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} \quad , \quad A_{2,2} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

• Il metodo JPEG utilizza la *Discrete Cosine Transform*, in parole povere utilizza come base per $\mathbb{R}^n$ i vettori $x_p$ di componenti

$$(x_p)_i = \cos\left(\frac{\pi p}{2n}(2i+1)\right)$$

(che sono quasi perpendicolari), generando poi la base per lo spazio delle matrici nel modo visto sopra.

• Per quel che riguarda la compressione frattale, si consideri l'immagine come una funzione $f$ definita su di un rettangolo (matrice a punti) i cui valori sono rappresentati dalla luminosità (ad esempio nel caso del bianco e nero) del singolo punto.

Sullo spazio di tali funzioni si consideri la norma $L^\infty$, cioè la norma di un'immagine è il massimo valore assunto in valore assoluto.

Dato un quadrato $Q$ di dimensione prefissata $k \times k$ viene applicata ad ogni quadrato $R$ della figura di dimensione $2k \times 2k$ (ad ogni punto del grafico di $f$) una trasformazione lineare del tipo

$$\begin{pmatrix} \alpha & \beta & 0 \\ \gamma & \delta & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} u \\ v \\ l \end{pmatrix} \tag{1}$$

dove $(x, y)$ sono le coordinate del punto e $z = f(x, y)$ è il valore della funzione.

$\alpha, \beta, \gamma, \delta, u, v$ sono fissate in modo da generare un quadrato dimezzato di proporzioni, e con esso le otto possibili simmetrie e rotazioni (il nuovo asse $x$ può essere orientato in una delle 4 possibili direzioni, alto, basso, destra e sinistra, e di conseguenza il nuovo asse $y$ ha, per ogni scelta di $x$ due possibilità di orientamento perpendicolare).

I valori di $c$ ed $l$ rappresentano il contrasto e la variazione di luminosità.

$c$ ed $l$ sono scelti in modo da minimizzare lo scarto quadratico tra i valori di $f$ sul quadrato originale $Q$ e la trasformazione di $R$ effettuata; se indichiamo con $q_i$ ed $r_i$ i $k^2$ valori sul quadrato $Q$ e sul trasformato di $R$, si ottiene

$$c = \frac{k^2 \left(\sum_{i=1}^{k^2} q_i r_i\right) - \left(\sum_{i=1}^{k^2} q_i\right)\left(\sum_{i=1}^{k^2} r_i\right)}{k^2 \left(\sum_{i=1}^{k^2} r_i^2\right) - \left(\sum_{i=1}^{k^2} r_i\right)^2}$$

e

$$l = \frac{\left(\sum_{i=1}^{k^2} q_i\right) - c\left(\sum_{i=1}^{k^2} r_i\right)}{k^2}$$

Con tali valori viene calcolato lo scarto quadratico e viene scelto quel quadrato $R$ che minimizza l'errore (il valore assoluto di $c$ dovrà risultare minore di uno, affinché sia possibile ricostruire l'immagine, come vedremo in seguito; se ciò non si verifica $c$ verrà posto d'ufficio uguale ad un numero in valore assoluto minore di 1, prima di calcolare $l$).

Vengono quindi memorizzate: il tipo di trasformazione effettuata (delle 8 possibili, utilizzando quindi 3 bit), le coordinate dell'origine del quadrato $R$ prescelto (relative alla posizione di $Q$; esaminando ad esempio $64^2$ quadrati in un intorno di $Q$, 64 per ogni coordinata, saranno necessari 12 bit, essendo $64^2 = 2^{12}$), ed infine i valori di $c$ ed $l$ (per la verità le prove qui viste sono state effettuate con $c$ fissato e quindi si è memorizzato solo il valore di $l$, che essendo nel range da -255 a 255 richiede 9 bit).

In totale, per ogni quadrato $Q$ di dimensioni $k \times k$ sono stati necessari 24 bit = 3 byte, contro i $k^2$ necessari per memorizzare tutti i punti; la compattazione è perciò di 16/3 per $k = 4$ e può essere aumentata a piacere (ad esempio 64/3 per $k = 8$), perdendo ovviamente in qualità.

Va osservato che un miglioramento si può ottenere variando le dimensioni del quadrato $k$ a seconda delle zone della figura: nell'immagine vista precedentemente, si possono utilizzare senza problemi quadrati $8 \times 8$ o più nel cielo o sul lago, mentre è necessaria una maggiore definizione sulla parte rocciosa.

Tale procedimento è poi ripetuto per ogni quadrato $Q$.

Per quel che riguarda la ricostruzione dell'immagine, basterà partire da una qualunque immagine (ad esempio una figura uniformemente grigia) ed applicare sui singoli quadrati $R$, le trasformazioni memorizzate, ottenendo in tal modo una nuova figura.

Poiché, come vedremo tra poco, se $|c| < 1$ la trasformazione ottenuta è una contrazione dallo spazio delle immagini in se stesso, questo procedimento ripetuto più volte converge verso il punto fisso, immagine finale cercata.

Per provare che tale trasformazione $T$ è una contrazione, siano $f$ e $g$ due immagini qualunque (definite su un dominio $D$).

La norma di $Tf - Tg = T(f - g)$ sarà assunta in un certo punto $(x_0, y_0)$ ed il valore di $T(f - g)$ in tale punto sarà il risultato di una trasformazione lineare del tipo (1), avente un certo $\bar{c}$ ed un certo $\bar{l}$, applicata ad un punto particolare $(x_1, y_1)$ dell'immagine stessa (residente in un certo quadrato).

Pertanto

$$\|T(f-g)\| \;=\; |Tf(x_0,y_0) - Tg(x_0,y_0)| \;=\; |\bar{c}\,(f(x_1,y_1) - g(x_1,y_1))| \;\leq\; |\bar{c}|\,\|f-g\|$$

e quindi si avrà una contrazione se $|\bar{c}| < 1$.

## Indice.

Introduzione ..... pag. 1
Compressori senza perdita di informazione ..... pag. 1
Codifiche ..... pag. 2
La codifica di Huffman ..... pag. 4
I formati RunLengthEncode e PCX ..... pag. 5
Compressori con dizionario ..... pag. 6
La codifica aritmetica ..... pag. 8
La trasformazione di Burrows-Wheeler ..... pag. 9
Compressione con perdita di informazioni ..... pag. 10
La compressione frattale ..... pag. 22
Appendice ..... pag. 25