

Comunicazioni Multimediali

Compressore di immagini

Marco Tieghi

Corso di laurea in Informatica

Obiettivo

- Obiettivo del progetto è stata la creazione di un compressore di immagini che potesse simulare il comportamento di un compressore GIF.
- Il formato dell'immagine compressa è stato reso simile a quello di un'immagine GIF, mentre l'algoritmo di creazione della paletta implementato è stato personale

Alcuni concetti

- Un'immagine è un'insieme di elementi giustapposti in uno spazio bidimensionale. Ognuno di questi elementi si chiama **pixel** (**p**icture **e**lement), e contiene le informazioni relative al colore di tale punto dell'immagine.
- Il modo di rappresentare i colori di un'immagine si chiama **piano colore**. Esistono diversi piani colore, a seconda dello scopo per cui si usa l'immagine, e uno dei più usati è il piano **RGB** (**R**ed **G**reen **B**lue), dove ogni colore è rappresentato da una terna, un valore per ogni componente.

Il progetto

- Il formato immagine usato è stato il **PPM** (**P**ortable **P**ix**M**ap file format), dove ogni colore è rappresentato da un numero a 24 bit, e ogni componente ha 8 bit a disposizione (valore massimo = 255). PPM dispone questi dati in una matrice $3X*Y$
- Analizzando l'immagine e creando una lista dei colori usati, è possibile selezionarne alcuni (fino ad un massimo di, ad esempio, 256) secondo uno specifico criterio. Si crea così una nuova palette
- Questa nuova palette può essere poi usata per costruire l'immagine compressa partendo dall'immagine di partenza, indicando al posto dell'intero valore numerico del colore solo la posizione del colore nella palette (che può essere un numero a 8 bit)
- In questo modo si riducono le informazioni superflue, ottenendo un'immagine di dimensione $X*Y$. Usando numeri a 8 bit (la posizione del colore nella palette) invece che 24 bit (il valore del colore) per rappresentare un colore, il fattore di ridimensionamento è di circa 1/3

- Il progetto è costituito da diversi file:
 - * compressor.c: il compressore vero e proprio
 - * decompressor.c: il decompressore
 - * ImageHandler.h: libreria di supporto per la gestione di immagini in formato PPM
 - * list.h: libreria di supporto per la creazione e gestione della lista colori
 - * psnr.c: programma per determinare il PSNR di due immagini (nel nostro caso, immagine non compressa e immagine compressa)
- I passi eseguiti sono i seguenti:
 - * lettura di immagine in formato PPM
 - * analisi dei colori presenti e creazione di una paletta ridotta
 - * creazione del file compresso analizzando l'immagine originale e usando la nuova paletta creata
 - * lettura del file compresso e ricostruzione di un'immagine PPM usando la paletta ridotta

Il compressore - Passi e algoritmo

Il primo passo eseguito è la lettura di un'immagine in formato PPM a colori binaria (codice P6), e i dati letti vengono salvati in una struttura apposita che memorizza le dimensioni dell'immagine e una lista di tutti i pixel che la compongono. Per memorizzare le informazioni dei colori, è usata un'ulteriore struttura che per ogni pixel ne memorizza i valori di rosso, verde e blu. Questo è eseguito dalla funzione readPPM()

```
typedef struct {
    unsigned char red,green,blue;
} PPMPixel;
```

Dopo l'apertura dell'immagine, la struttura che la rappresenta è passata alla funzione colorList(): questa funzione scorre ogni pixel dell'immagine e crea la lista dei colori unici che la compongono. In questa fase, ogni elemento della lista è rappresentato come una struttura che memorizza i valori RGB del pixel PPM e la sua frequenza nell'immagine (tale frequenza è calcolata nella fase di scansione). È presente un puntatore al prossimo elemento della lista, ma questo riguarda l'aspetto implementativo nel C di una lista linkata. Il ritorno di questa lista è un puntatore alla testa della lista.

```
struct color {  
    int frequency;  
    unsigned char red,green,blue;  
    struct color *next;  
};
```

Dopo l'analisi dei colori, si comincia con il processo di compressione vero e proprio, eseguito dalla funzione `compressImage()`, la quale necessita della struttura dei dati dell'immagine originale, la lista completa dei colori e un dato di tipo `myImage`. `myImage` è il formato personale usato per rappresentare un'immagine PPM compressa.

Questo tipo di dato contiene due valori che si riferiscono alle dimensioni dell'immagine, un intero che memorizza la dimensione della palette colori (fino al massimo di 256), la palette dei colori usati (lista di tipo `PPMPixel`) e una lista di `char` a 8 bit che rappresentano, per ogni pixel, la posizione nella palette del colore associato.

```
typedef struct {
    int x, y;
    int pSize;
    PPMPixel *colors;
    unsigned char *data;
} myImage;
```

La parte principale della compressione è costituita dalla creazione della palette. Infatti, il processo di selezione dei colori più adatti è il nucleo dell'elaborazione.

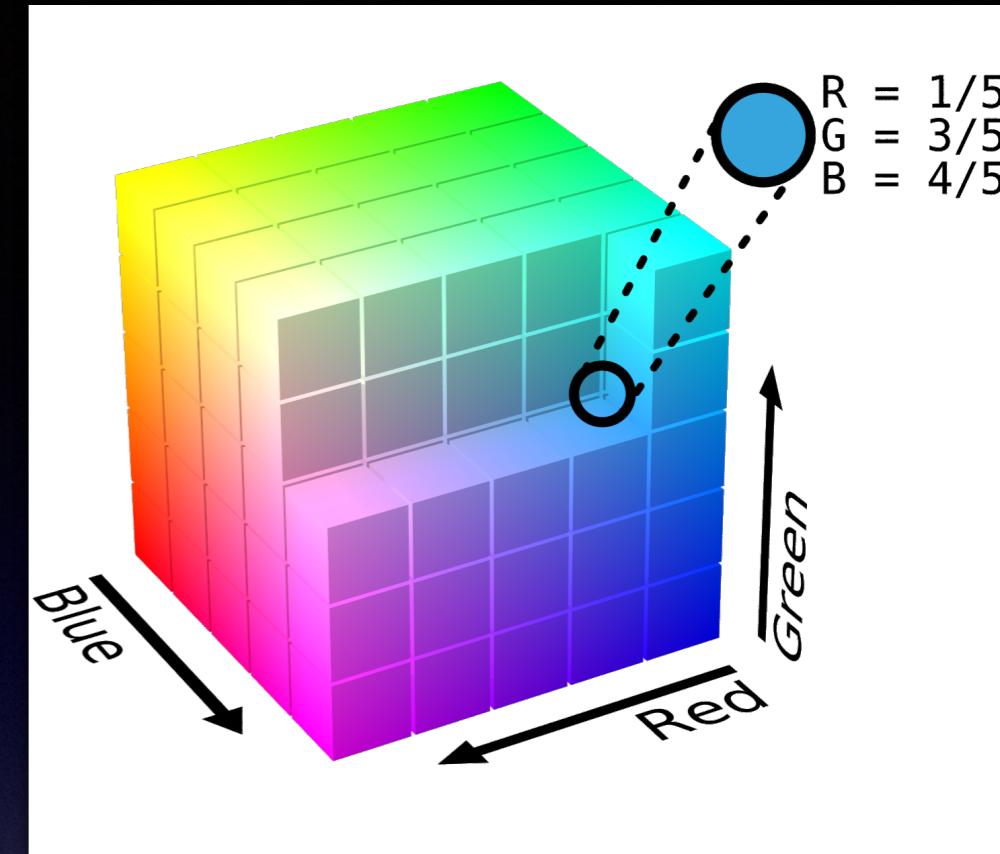
Si è infatti scelto di operare sui colori più frequenti presenti nell'immagine, attuando però una selezione per assicurare varietà cromatica e non selezionare una particolare tendenza cromatica.

Il primo passo consiste quindi nella creazione di un vettore contenente, in ordine decrescente, tutte le frequenze di apparizione dei colori. La funzione `frequencyVector()` esegue questa operazione, leggendo il campo `frequency` di ogni colore presente nella lista di colori unici, e ordinando in un vettore tali valori, dalla maggior frequenza alla minor frequenza. La dimensione massima di tale vettore è stata limitata a 256 valori. In quanto, anche assumendo che per ogni valore di frequenza ci sia un solo colore, il numero massimo di colori che posso considerare è 256.

Una volta ottenuta la matrice delle frequenze, comincio a costruire la palette.

Scorro il vettore delle frequenze, usando un valore alla volta. Scorro la lista dei colori totali ricavando la struttura del colore che ha il valore della frequenza che soddisfa la ricerca. Bisogna ricordare che ad una stessa frequenza è associato almeno un colore, quindi posso ottenere anche più colori. Tuttavia, non aggiungo immediatamente il colore alla palette perché se usassi solo i colori più frequenti, otterrei un'immagine compressa con una dominanza cromatica invece che varietà andando a perdere sicuramente alcune informazioni. Volendo ridurre al minimo la perdita di porzioni di immagine, devo attuare un'ulteriore scelta sui colori ricavati

Una terna RGB può essere rappresentata come punto in un piano XYZ, all'interno di un cubo di lato 255. Suddividendo ricorsivamente tale cubo, posso ottenere sottocubi che identificano diverse zone del piano colore. Maggiore è la suddivisione, più dettagliata sarà la scelta cromatica all'interno di ogni sottocubo.



Prima quindi di inserirli, valuto l'appartenenza del colore al sottocubo del mio piano colore. Se non ne ho inseriti ancora un numero elevato per l'area specifica, lo aggiungo alla palette, altrimenti lo salto e considero il colore successivo. Questo mi permette di selezionare i colori più frequenti attuando anche una scelta cromatica più ampia possibile. Ho scelto di suddividere lo spazio dei colori in 8 sottocubi.

Al termine di questo procedimento, ho costruito la palette con una buona varietà di colori. Per costruire l'immagine compressa, scorro i dati dell'immagine originale, e ricavo dalla palette ridotta quale colore ha distanza minima dal colore che sto selezionando, usando la seguente formula:

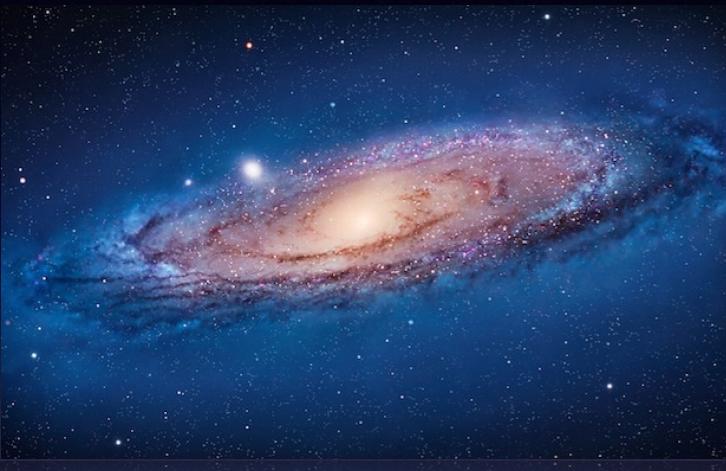
$$\text{colorDistance} = \sqrt{(\text{color1.red} - \text{color2.red})^2 + (\text{color1.green} - \text{color2.green})^2 + (\text{color1.blue} - \text{color2.blue})^2}$$

Nella matrice di dati dell'immagine compressa vado a porre semplicemente il valore dell'indice che nella palette ridotta riferisce a tale colore di distanza minima. Una volta terminato il processo, salvo il flusso in un file (di estensione customizzata PPMC, PPM Compressed) che rappresenta la mia immagine compressa

Il decompressore - Passi e algoritmo

Il programma di decompressione è molto più semplice del compressore, in quanto vado a leggere il mio file compresso PPMC, inserendo tutti i valori salvati nella struttura di tipo myImage vista precedentemente (dimensioni immagine, dimensione paletta, paletta e dati). Poi, scorro la lista dati, e leggendo il valore dell'indice ricavo dalla paletta i valori RGB del colore in tale posizione. Questo valore lo memorizzo nella posizione del rispettivo pixel in una struttura che rappresenta l'immagine PPM decompressa. Terminato questo processo, salvo il mio flusso dati in un file secondo la struttura del PPM, così che possa essere correttamente letto e visualizzato.

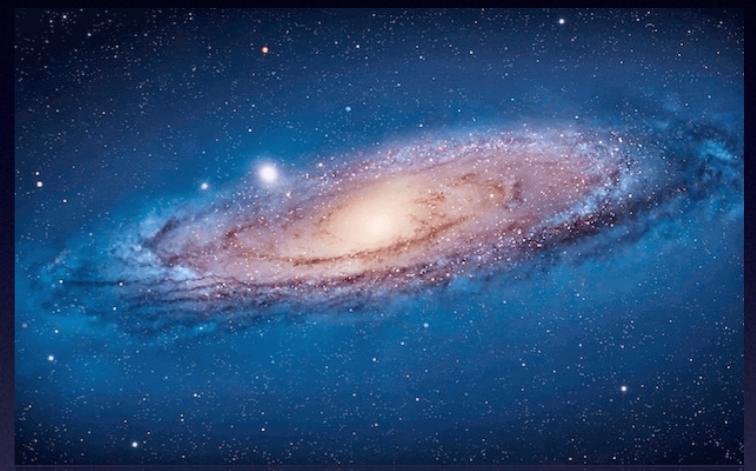
Esempi - Andromeda



Andromeda.ppm
705KB



Andromeda.ppc
241KB



Andromeda.gif
137KB

Risoluzione
620 x 388

PSNR
28.55

Esempi - Sunflowers



Sunflowers.ppm
2MB



Sunflowers.ppmc
654KB



Sunflowers.gif
536KB

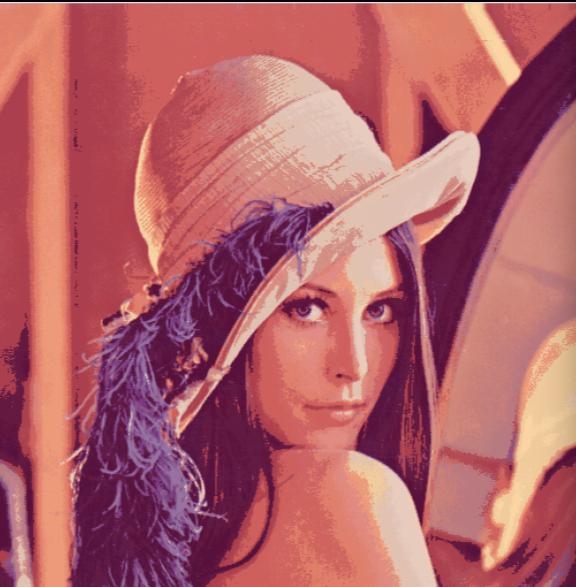
Risoluzione
720 x 908

PSNR
26.293

Esempi - Lena



Lena.ppm
768KB



Lena.ppmc
263KB



Lena.gif
213KB

Risoluzione
512 x 512

PSNR
29.062

Esempi - The Starry Night



StarryNight.ppm
1.2MB



StarryNight.ppmc
401KB



StarryNight.gif
406KB

Risoluzione
800 x 500

PSNR
23.669

Conclusioni e osservazioni

L'algoritmo adottato non è certamente l'unico usabile per la selezione di una palette. Ha i suoi vantaggi e i suoi svantaggi: al pari di permettere una buona variazione dei colori, porta lo svantaggio di non funzionare correttamente con immagini con forti dominanze cromate, riducendo la selezione della paletta per tali colori. Un miglioramento possibile può essere: valutazione della tendenza cromatica e maggior selezione dei colori su quella tendenza.

Si è implementato comunque questo algoritmo consci delle possibili problematiche, per una riflessione critica sui risultati e possibili miglioramenti.

Nonostante ciò, lo scopo del progetto si è ritenuto soddisfacente.