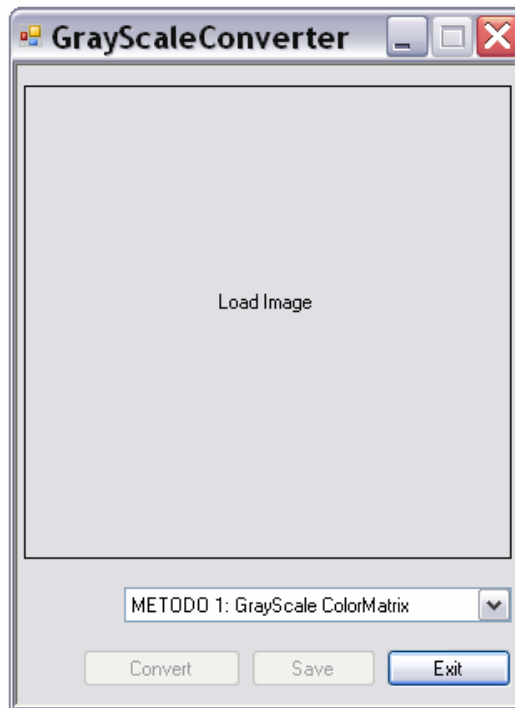


GRAYSCALE CONVERTER

Una semplice applicazione per la conversione di immagini in scala di grigi.



Massimo Rabbi
massimo.rabbi(AT)techtown.it
<http://www.techtown.it>
<http://www.secornetwork.net>
<http://www.brainweb.it>



INTRODUZIONE

In questo articolo vedremo come sia possibile realizzare con poco sforzo un'applicazione che partendo da una classica immagine a colori effettui una conversione in scala di grigi della stessa.

Il programma verrà realizzato in C#, sfruttando quindi questo linguaggio considerabile non a torto il fulcro del Framework .NET.

UN PO' DI TEORIA

La prima cosa da tenere a mente quando si lavora con la grafica nella piattaforma .NET è l'importanza della libreria **GDI+**.

Si tratta essenzialmente di una interfaccia che consente al programmatore di realizzare applicazioni Web e Windows in grado di dialogare con dispositivi/devices grafici come stampanti, scanner e monitors.

GDI+ è il "terzo" componente che si interpone tra programma e device grafico, fungendo da filtro e elaborando i dati provenienti dall'uno, inviandoli all'altro.



Esemplificando: pensiamo al semplice compito di dover disegnare una linea sullo schermo.

Questa è composta da una sequenza di pixels che vanno da un punto A iniziale ad uno B finale. Il monitor deve sapere dove disegnare questi pixels, per questo invece che farlo "manualmente", viene invocato il metodo di GDI+ *DrawLine* che consente di disegnare la linea specificando unicamente i punti A e B. Sarà poi compito di GDI+ partendo da punto iniziale e finale impartire al monitor le istruzioni corrette affinché la sequenza completa di pixels venga disegnata a schermo.

Tecnicamente parlando GDI+ è una libreria composta da classi C++ collocate nel file *Gdiplus.dll*. Questo file è parte integrante dei sistemi Windows XP e Windows 2003 Server, tuttavia può essere reso disponibile sugli altri sistemi installando GDI+ o anche installando una qualche forma del framework .NET.

Dopo questa breve premessa possiamo spendere due parole circa lo spazio dei colori RGBA, visto che una delle classi che utilizzeremo nel progetto, ovvero *ColorMatrix* prevede che le operazioni effettuate, coinvolgano proprio questo spazio.

RGBA è l'acronimo per **R**ed **G**reen **B**lue **A**lpha, ovvero i quattro canali:

- rosso
- verde
- blu
- alpha

dove l'alpha channel è indicatore di opacità.

I valori di questi quattro canali possono assumere diverse rappresentazioni:

- percentuale 0% a 100%
- valori interi tra 0 e 255
- valori decimali variabili tra 0 e 1.0

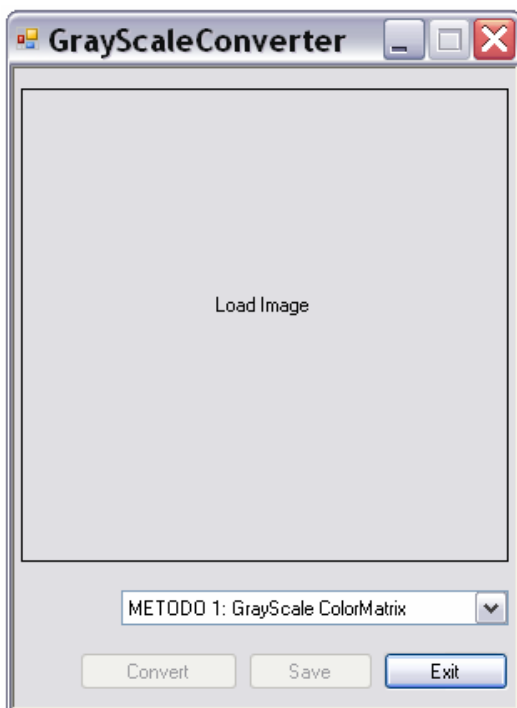
Esemplificando relativamente al canale alfa, avremo che un pixel con valore 0% è completamente trasparente ovvero invisibile. Al contrario un valore 100% è indicatore di un pixel completamente opaco.

Valori percentuali compresi tra 0 e 100 rendono possibile ottenere "effetti trasparenza" tipo vetro.

Tra i formati grafici più diffusi che attualmente utilizzano lo spazio di colori RGBA vi è *PNG* (Portable Network Graphics).

A volte lo spazio di colori RGBA viene indicato con aRGB, ovvero la componente Alpha compare al primo posto: questo succede ad esempio nella linea di software Macromedia.

LA STRUTTURA DEL PROGRAMMA



Il programma è strutturato in maniera elementare:

- da una *PictureBox* centrale necessaria a mostrare dapprima l'immagine caricata e successivamente l'immagine convertita
- da una *ComboBox* che consente di selezionare quale metodo utilizzare per il grayscaleing
- tre button per le operazioni base
- una *ProgressBar* che viene visualizzata durante il processo di conversione usando il metodo 2

E' chiaro che la parte su cui è interessante focalizzarsi è quella relativa ai due metodi di grayscale.

Analizziamo un po' più in dettaglio dando anche qualche snapshot del codice usato.

METODO 1: Grayscale ColorMatrix

Questo è sicuramente il metodo più veloce per eseguire la conversione in scala di grigi, visto che sfrutta direttamente le funzioni della libreria GDI+.

Nella fattispecie per l'appunto si fa uso della classe *ColorMatrix*.

Semplificando, *ColorMatrix* è una matrice quadrata di dimensione 5x5 in cui i valori degli elementi matriciali spaziano da 0 a 1 e le cui operazioni, lo ricordiamo, vengono effettuate nello spazio di colore RGBA.

Sembrerebbe normale quindi aspettarsi che la matrice sia una 4x4, visto che abbiamo a che fare con 4 canali/componenti [R,G,B,A].

Tuttavia una matrice 4x4 è sufficiente unicamente per svolgere operazioni/trasformazioni cosiddette lineari (scaling, rotation, reflection etc.), mentre per tecniche di manipolazione del colore quali l'aggiustamento, è necessario aggiungere una quinta componente "dummy", che consenta di realizzare operazioni/trasformazioni affini.

Le trasformazioni affini sono in linea di massima il risultato dell'unione di un'operazione lineare ed una non-lineare (esempio una traslazione).

E' chiaro che una qualsiasi trasformazione lineare come una rotazione può essere "adattata" in trasformazione affine modificando in maniera semplice la matrice di moltiplicazione.

Esemplifichiamo mostrando il caso di una rotazione nel sistema di coordinate xy:

- operazione lineare di rotazione del punto (x,y) di un angolo θ rispetto all'origine

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Moltiplicazione matriciale per ottenere il nuovo punto

- operazione affine di rotazione del punto (x,y) di un angolo θ rispetto all'origine

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrice di trasformazione modificata

Riportandoci quindi nello spazio di colori RGBA avremmo che il nuovo vettore sarà $[R,G,B,A,w]$, dove per l'appunto w sarà una componente dummy posta a 1.

Questo verrà poi moltiplicato per la matrice di trasformazione ColorMatrix 5x5, permettendoci di ottenere l'effetto desiderato.

Quello che interessa a noi è effettuare un'operazione di Grayscale, vediamo quindi come questo sia possibile e come dobbiamo "comporre" la matrice di trasformazione 5x5.

Tra le varie definizioni, della luminanza sappiamo che è "il segnale che trasporta l'informazione relativa alle immagini in bianco e nero".

Proprio quello che serve a noi per ottenere l'effetto scala di grigi.

La luminanza indicata anche come componente Y può essere ottenuta dalla seguente formula basata sullo standard della tv americana NTSC:

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Stabilito questo possiamo ora ricavare la matrice ColorMatrix che a noi serve:

$$\begin{bmatrix} R & G & B & A & w \end{bmatrix} \times \begin{bmatrix} 0.299 & 0.299 & 0.299 & 0 & 0 \\ 0.587 & 0.587 & 0.587 & 0 & 0 \\ 0.114 & 0.114 & 0.114 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R' & G' & B' & A' & w' \end{bmatrix}$$

In questo modo le componenti le nuove componenti RGB avranno il valore Y , mentre A' manterrà il suo.

Diamo uno sguardo a come si traduce tutto questo in codice:

```
private void grayscaleMethod1()
{
    Image img = pictureBox1.Image;
    Bitmap bm = new Bitmap(img.Width, img.Height);
    Graphics g = Graphics.FromImage(bm);

    ColorMatrix cm = new ColorMatrix(new float[][]{
        new float[] {0.299f, 0.299f, 0.299f, 0, 0},
        new float[] {0.587f, 0.587f, 0.587f, 0, 0},
        new float[] {0.114f, 0.114f, 0.114f, 0, 0},
        new float[] {0, 0, 0, 1, 0},
        new float[] {0, 0, 0, 0, 1}});

    ImageAttributes ia = new ImageAttributes();
    ia.SetColorMatrix(cm);
    g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height), 0, 0,
img.Width, img.Height, GraphicsUnit.Pixel, ia);
    g.Dispose();
}
```

```

        pictureBox1.Image = bm;
    }

```

Dal codice si nota anche come sia relativamente semplice applicare una *ColorMatrix* ad un'immagine:

1. si associa un oggetto *ColorMatrix* ad un oggetto *ImageAttributes*
2. si passa l'oggetto *ImageAttributes* come parametro al metodo *Graphics.DrawImage*

METODO 2: Grayscale Pixel per Pixel

Questo metodo molto più spartano prevede la "sostituzione manuale" di ogni singolo pixel dell'immagine in maniera da creare la nuova immagine convertita in scala di grigi.

Per la modifica delle componenti RGB utilizziamo sempre la formula vista precedentemente per il calcolo della luminanza, ovvero:

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Ecco il codice:

```

private void grayscaleMethod2()
{
    progressBar1.Visible = true;

    Bitmap grayscaleImg = (Bitmap)pictureBox1.Image;
    int height = grayscaleImg.Size.Height;
    int width = grayscaleImg.Size.Width;

    progressBar1.Minimum = 0;
    progressBar1.Maximum = height;
    progressBar1.Step = 1;

    for (int j = 0; j < height; j++)
    {
        for (int i = 0; i < width; i++)
        {
            Color col;
            col = grayscaleImg.GetPixel(i, j);
            int grayValue = (int)(0.299 * col.R + 0.587 * col.G + 0.114
* col.B);
            grayscaleImg.SetPixel(i, j, Color.FromArgb(grayValue,
grayValue, grayValue));
        }

        progressBar1.PerformStep();
    }

    pictureBox1.Image = grayscaleImg;

    progressBar1.Visible = false;
}

```

Qui vediamo come vengano sfruttati i metodi GetPixel e SetPixel per recuperare e impostare rispettivamente le informazioni sul colore di ogni singolo pixel dell'immagine.

CONCLUSIONI

Questo articolo si conclude qui, sperando di aver mostrato alcune semplici funzionalità della classe ColorMatrix e qualche principio teorico sempre utile per chi come me si è trovato per la prima volta a lavorare/modificare immagini programmando.

Per ulteriori approfondimenti vi rimando alla sitografia in calce all'articolo.

RIFERIMENTI E SITOGRAFIA

[1]. GDI+ Faq main index:

<http://www.bobpowell.net/faqmain.htm>

[2]. Matrix Operations for Image Processing

<http://www.graficaobscura.com/matrix/index.html>

[3]. Color FAQ

<http://www.poynton.com/ColorFAQ.html>

[4]. ColorMatrix Basics – Simple Color Adjustment

<http://www.codeproject.com/vb/net/colormatrix.asp>

[5]. Addison Wesley - GDI+ Programming with C#

[6]. Transformation Matrix

http://en.wikipedia.org/wiki/Transformation_matrix