撰寫影像處理程式 難不倒我 !!

- 簡單的數位影像處理 (C# 篇)

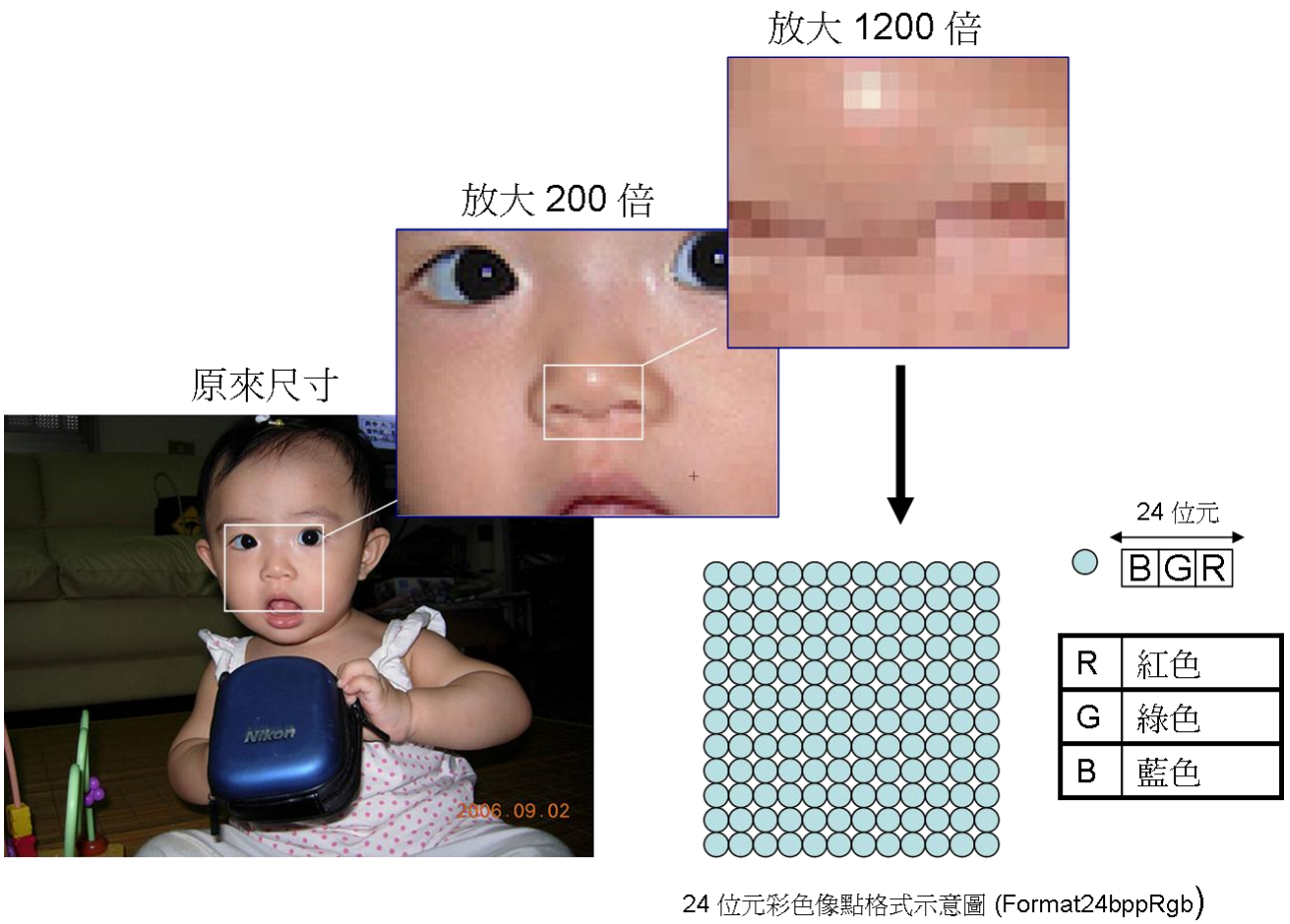
文：井民全
個人程式設計心得
http://mqjing.twbbs.org/~ching/Course/course.htm

之前向大家介紹如何取得無名小站的照片，不知道有沒有人去試試看 ？如今我們已經會把線上的照片下載回來，大家會不會想知道 [如何自己寫一個秀圖的軟體] 甚至是一個 [影像處理軟體] ？也許有人會問：已經有 ACDSee 這麼優秀的秀圖工具，爲什麼還需要自己寫一個相同功能的軟體呢？在這裡我們並不是要你寫一個程式去跟 ACDSee 打對台，這篇文章主要的目的是希望透過這樣的教學讓你瞭解，如何把秀圖與簡單的影像處理功能加入既有的軟體中，進而擴充原來軟體的功能．你也可以藉由這篇文章，知道顯示一張圖片是如此簡單的事情．其實如果你有將 9 月號電腦王的下載無名小站照片的程式拿來實作，很快的你就會發現你需要一個能夠顯示圖片的功能．本文正好可以當作續集來閱讀．讓我們一起來學習數位影像處理吧!

我們希望用電腦來處理照片，那麼起步走的第一件事就是要瞭解電腦如何處理圖片．我們希望知道:

照片如何被表示，照片如何被儲存，而我們又該怎麼去操作它呢?

顯示在電腦螢幕上的圖片或者是照片，基本上可以用點陣圖來表示．點陣圖將圖片視爲一堆點的方形集合．我們可以想像電腦裡面的照片就是由一個一個像點所組成的一個方形點陣列，這些點我們稱之爲像素(pixel)．在一張全彩(full-color)或者是真實顏色(true-color)點陣圖中，每個像素可以由 24 個位元(bit)來表示．也就是紅色 8 個位元，綠色 8 個位元，藍色 8 個位元，總共 24 個位元可用來表達 2^{24}=16777216 種顏色．所以我們說一個點陣圖中的每個像點都以 24 位元單位儲存，稱爲一個 24bit BMP 檔．在.Net Framework 上，我們使用 PixelFormat 類別的 Format24bppRgb 來表示這種格式．當然也有些點陣列圖使用 32 bit 來存放一個像素 (同理可用 Format32bppRgb 表示)，而多出來的那 8 個 bit 通常用來表示透明度．下面是一個簡單的 24bit BMP 檔像點示意圖．



圖說：點陣列影像像點示意圖
圖檔：點陣列影像示意圖.png

使用 BMP 檔案格式表示圖形資料，所需的容量可能非常龐大．以一張寬高 500 x 375 照片爲例，就需要 562500 個位元組的容量儲存該張照片．所以大多數的圖片都使用壓縮的方式，將圖形儲存在檔案中以節省所需要的磁碟空間．當需要的時候才解壓縮回適當的格式進行處理．常見的圖形檔案格式有 JPEG，BMP，GIF，PNG．每一種圖檔格式都有自己的一套複雜方法用來表示每一個像點的資訊．你可以在網路或坊間書籍得到相關圖檔格式的詳細說明．我們的重點將放在如何寫程式操作影像．我們目前只要知道怎麼從各種影像檔格式轉換成我們能夠處理的 R(紅色)，G(綠色)，B(藍色) 模式．

使用程式庫幫我們解碼

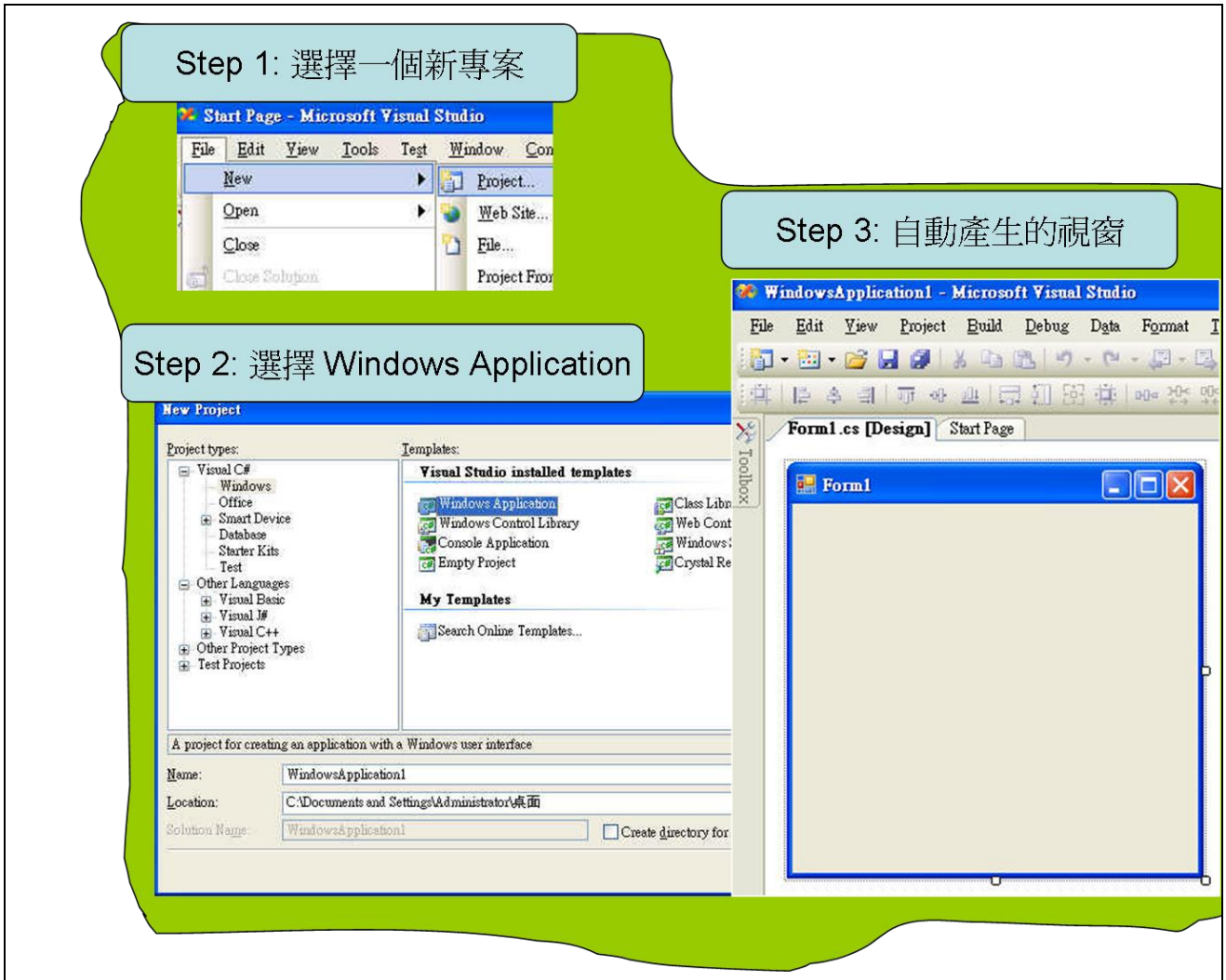
圖形檔案的格式通常是非常的複雜，主要的原因是其採用了不同的視覺模型來解釋顏色與資訊，同時爲了不同的顯示器，有些影像格式還跟顯示裝置相關．例如:最常見的 BMP 就有分爲裝置相關與裝置無關兩種模式，JPEG 則使用一連串複雜的演算法,包含離散餘弦轉換(Discrete Cosine Transform)等，將影像以頻率的方式表示，基於人類視覺對高頻不敏感的性質進行對影像的壓縮．除非你是資訊系的學生，否則通常我們都是使用影像相關的程式庫，幫我們進行解碼的動作，將壓縮過的資料重新轉譯成一般人比較容易接受的 RGB 色彩模型，

以便進行影像處理．如果你不用程式庫，那麼你可能要自己撰寫轉譯的程式．對於這種壓縮/解壓縮程式，我們稱之爲 Codec．事實上這類影像處理的商用程式庫非常多，例如: LEADTOOLS 公司就提供了一組影像處理工具 Raster Imaging SDKs．這個程式庫就支援超過 150 種不同的影像檔案格式(包含了最新的 JPEG 檔案格式 JPEG2000 與較少見到的 LZW 格式)，其中還提供了有關掃描器，色彩轉換，與一些有用的高速影像處理功能．Intel 與 鼎鼎有名的 MathWorks 公司也都有提供功能強大的影像處理程式工具．其實只要祭出 Google 大神，就會列出一堆前人已經寫好的免費工具，你可以輸入 Image Library 搜尋看看相關的資訊．

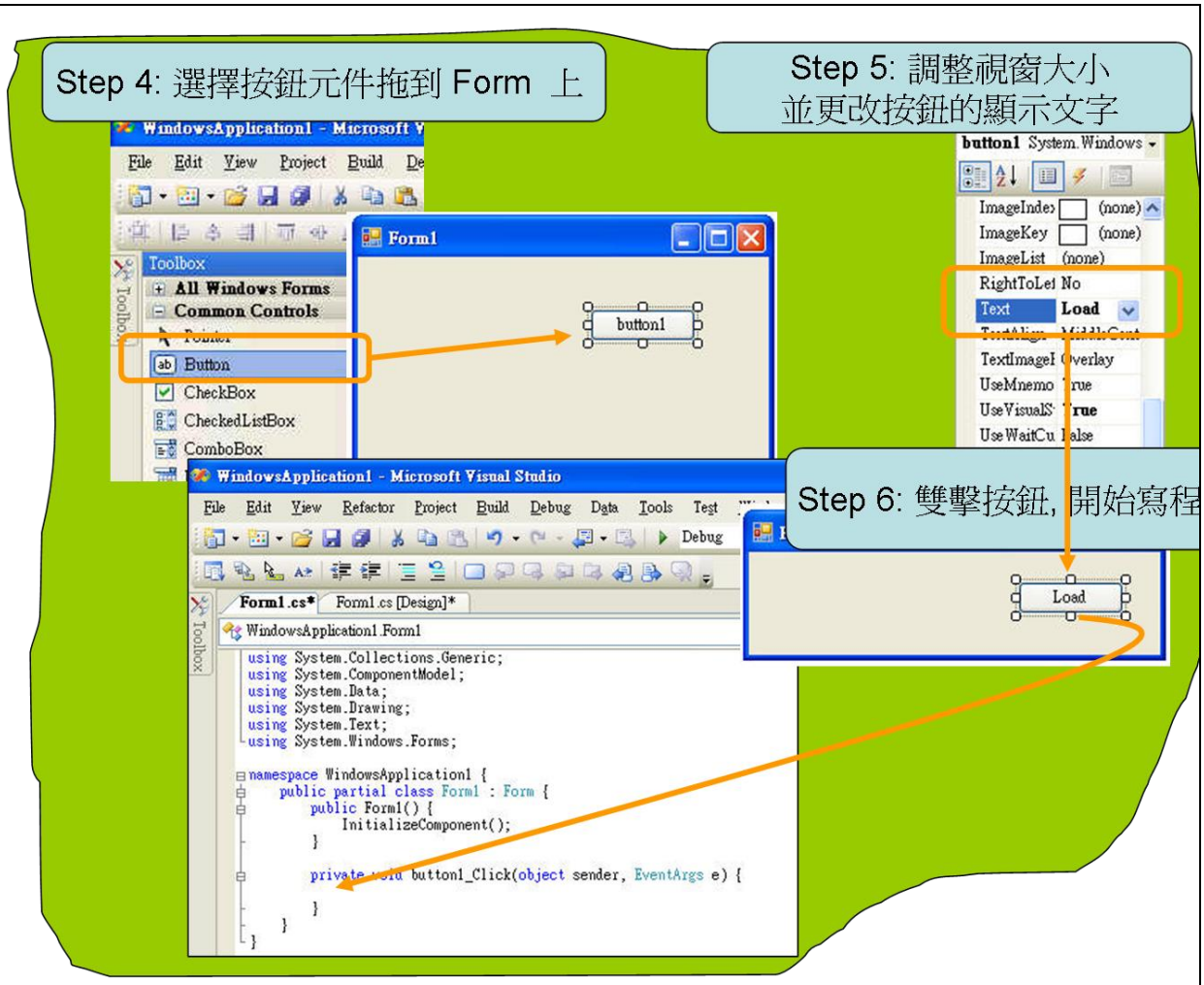
如果你使用 .Net Framework 搭配 C# 程式語言，而且呢 ... 只想要能夠讀取常見影像檔．如: JPEG, PNG, GIF 或 BMP，那根本不用找相關的程式庫，因爲微軟已經幫你寫好解碼程式了．就位於 .Net Framework 中的 Image 類別，這個類別搭配 Graphics 類別一併提供了基本的影像處理工具，例如: 在影像上畫圓，在影像上畫出指定的字串等．當然了 Image 類別也提供了存檔的功能，也就是你可以把 Image 物件的影像存成指定的格式，如 jpg, png, gif 或者是 bmp 等常用的格式．接下來，我們將從一步一步的介紹如何使用 Image 類別，來進行簡單的影像處理，並針對緩慢的 SetPixel/ GetPixel 提出高效率解決方案，讓你的影像處理程式更專業．首先，我們要秀出一張影像．

如何秀出一張影像 ?

首先我們使用 Visual Studio 建立一個新的專案並且放入一個按鈕．整個流程如下:



圖檔: Visual Studio 2005 建構專案步驟_1



圖說: Visual Studio .Net 2005 建構專案的步驟

圖檔: Visual Studio 2005 建構專案步驟_2

經過一連串的操作後, Visual Studio 會幫你把整個程式的骨架建立好, 其中 button1_Click 這個方法, 就是按鈕的事件處理函式.

接下來, 我們要寫一個新的類別專門用來秀圖, 給他一個名字叫做 ImageForm. 假設我們希望讓載入的圖片能夠放在一個獨立的視窗中, 以便針對獨立視窗中的影像作處理. 所以呢 ... 讓這個秀圖類別 ImageForm 繼承 Form 應該沒錯 !

這樣的設計會使得 ImageForm 擁有視窗的性質.也就是說, 當我們建立 ImageForm 物件並且顯示它時, 會以視窗的方式來呈現 ImageForm 的外觀與行為.

使用 C#程式語言的寫法如下,

```
// 建立一個專門秀圖的 Form 類別
class ImageForm : Form {
}
```

另外，當秀圖視窗 ImageForm 顯示在桌面上時，希望能立即秀出我們載入的照片．所以呢 ... 我們應該把載入圖檔的程式碼放在秀圖類別的建構子中．這樣的安排會使得每次新建立一個 ImageForm 物件，就會載入圖形檔，另外把秀圖程式放到 OnPaint 方法中.會使得每次視窗重繪時，都會重新把照片畫上去．使得 ImageForm 物件的視窗外表，就是我們的圖片．

所以程式碼的架構，應該要長的像下面這樣．

```
// 建立一個專門秀圖的 Form 類別
class ImageForm : Form {
        Image image;
        // 建構子
        public ImageForm() {
            // 載入影像的程式碼放在這裡 ...
        }
        protected override void OnPaint(PaintEventArgs e) {
            // 顯示出影像的程式碼放在這裡 ...
        }
    }
```

由於前面建立專案時,拉了一個按鈕進來．我們作這個動作的目的是希望當使用者按下 Load 鈕時，直接開啓顯示照片的視窗．所以這時候就應該把建構秀圖物件的程式碼，放到 Load 按鈕的事件處理函式中，作法如下．

```
        // Load 按鈕事件處理函式
        private void button1_Click(object sender, EventArgs e) {
            ImageForm MyImage= new ImageForm(); // 建立秀圖物件
            MyImage.Show();// 顯示秀圖照片
        }
```

完整的 C# 秀圖程式碼，列表如下

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Imaging; // for ImageFormat

namespace WindowsApplication1 {
```

```
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        // Load 按鈕事件處理函式
        private void button1_Click(object sender, EventArgs e) {
            ImageForm MyImage= new ImageForm(); // 建立秀圖物件
            MyImage.Show();// 顯示秀圖照片
        }
    }

    // 建立一個專門秀圖的 Form 類別
    class ImageForm : Form {
        Image image;
        // 建構子
        public ImageForm() {
            // Step 1: 載入影像
            image = Image.FromFile(@"c:\淡蘭古道_山貂嶺段.bmp");
            this.Text = @"c:\淡蘭古道_山貂嶺段.bmp";
        }

        protected override void OnPaint(PaintEventArgs e) {
            // Step 2: 調整視窗的大小
            this.Height=image.Height;
            this.Width=image.Width;

            // Step 3: 顯示出影像
            e.Graphics.DrawImage(image, 0, 0,Width,Height);
        }
    }
}
```

你可以試試看，把 `image = Image.FromFile(@"c:\淡蘭古道_山貂嶺段.bmp");`
改成下面幾種不同格式的照片檔，看看是否能夠正確秀出相片.

```
image = Image.FromFile(@"c:\淡蘭古道_山貂嶺段.gif"); // 載入 gif
image = Image.FromFile(@"c:\淡蘭古道_山貂嶺段.png"); // 載入 png
image = Image.FromFile(@"c:\淡蘭古道_山貂嶺段.jpg"); // 載入 jpg
```

圖說: ImageForm 秀圖結果
圖檔: ImageForm 秀圖結果.png

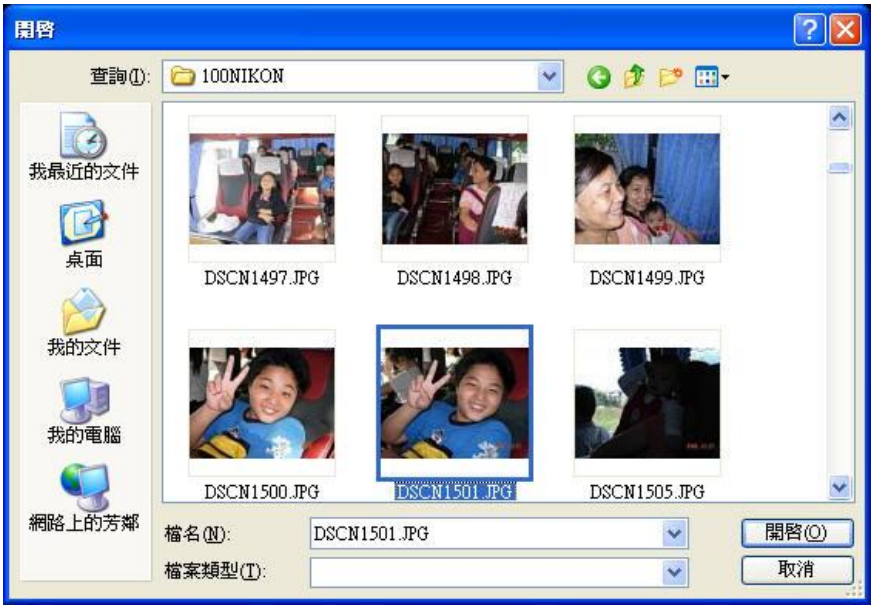
另外，如果你希望顯示出一個對話框，讓使用者選擇想要處理的影像檔．你可以使用 OpenFileDialog 類別．詳細的作法如下:

```
// 使用 OpenFileDialog 開檔的範例
public partial class Form1 : Form {
//  <略>
        private void button1_Click(object sender, EventArgs e) {
            // Step 1: 建立  OpenFiledialog 元件
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            openFileDialog1.InitialDirectory = "c:\\";

            // Step 2: 顯示出對話框
            if (openFileDialog1.ShowDialog() == DialogResult.OK) {
                // Step 3: 建立秀圖物件
```

```
                ImageForm MyImage = new ImageForm(openFileDialog1.FileName);
                MyImage.Show();// 顯示秀圖照片
                CurrentImage = MyImage;
            }
        }
}
class ImageForm : Form {
       //  <略>
        // 建構子
        public ImageForm(String Filename) {
            LoadImage(Filename);
        }
        // 載入影像公用程式
        public void LoadImage(String Filename) {
            // Step 1: 載入影像
            image = Image.FromFile(Filename);
            this.Text = Filename; // 設定標題

            // Step 2: 調整視窗的大小
            this.Height = image.Height;
            this.Width = image.Width;
        }
}
```



圖說: 使用 OpenFileDialog 選擇檔案

圖檔: 使用 OpenFileDialog 選擇檔案.png

放大 縮小 形變旋轉

還記得前面提到每次重繪視窗的時候，都會呼叫 OnPaint 方法．如果想要進行影像的幾何轉換，如放大，縮小或旋轉等功能，你都可以使用功能強大的 Graphics 類別方法完成．其實大部分的畫圖函式，你在 Graphics 的方法中都可以找到．例如：畫線或畫曲線你可以使用 DrawLine 方法，畫圓你可以使用 DrawArc，畫框你可以使用 DrawRectangle 方法．在這篇文章中，我們將使用 Graphics 所提供的畫影像方法 DrawImage，來進行放大縮小旋轉等扭轉影像的功能展示．下面是 DrawImage 方法的原型．

```
public void DrawImage(Image image, Point[] p);
```

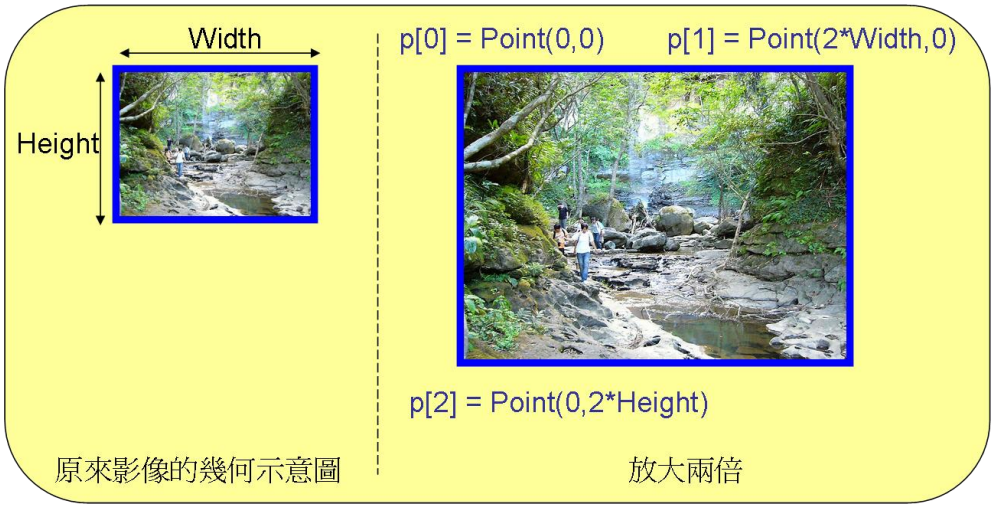
其中第一個參數 image 爲要畫上的影像，而 Point 陣列 p 則標示了要繪製影像的目標三個幾何座標點：

p[0] 表示目標的影像左上角位置

p[1] 表示目標的影像右上角位置

p[2] 表示目標的影像左下角位置

如果 Point[] p 表示的是正方形，那畫出來的影像就是依照比例的正方形．



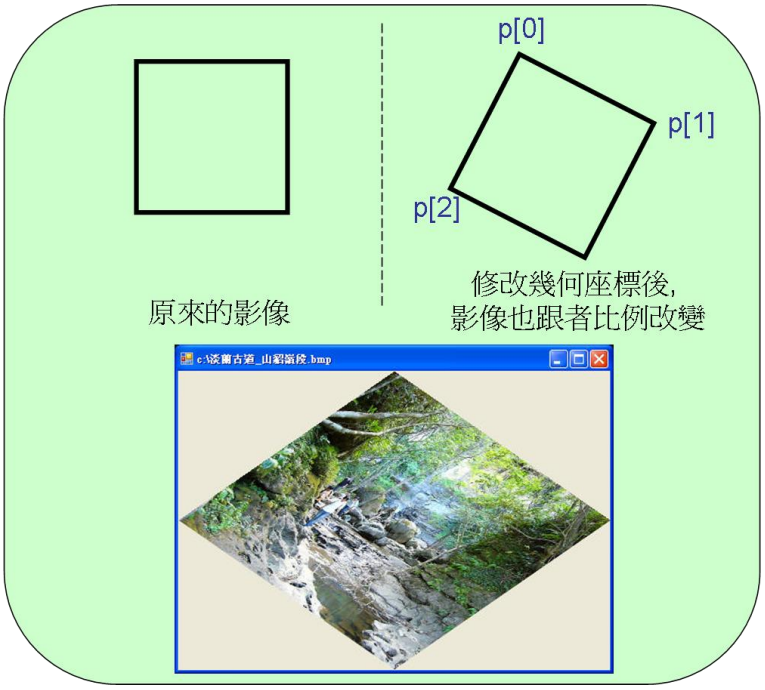
圖說：利用 DrawImage 將原始影像放大兩倍的設定範例．其中 Height 與 Width 是原始影像的高與寬．

圖檔：放大兩倍.png

```
// 放大 2 倍程式碼
            this.Height = image.Height * 2;
            this.Width = image.Width * 2;
            e.Graphics.DrawImage(image, new Point[] { new Point(0, 0),
                                        new Point(2*image.Width, 0),
                                        new Point(0, 2*image.Height) });
```

```
// 縮小 1/2 倍程式碼
            e.Graphics.DrawImage(image, new Point[] { new Point(0, 0),
                                                      new Point(image.Width/2, 0),
                                                      new Point(0, image.Height/2) });
```

如果指定的目標座標爲平行四邊形，則繪製出來的影像則是等比例的平行四邊形．下面是利用平行四邊形形變模擬旋轉的範例．



圖說：形變模擬旋轉範例
圖檔：形變的示意圖.png

```
// 形變旋轉程式碼 (cx 爲視窗的寬度, cy 爲視窗的高度)
e.Graphics.DrawImage(image, new Point[] { new Point(cx / 2, 0),
                            new Point(cx, cy / 2),
                            new Point(0, cy / 2) });
```

讀取影像的 RGB 值

在此之前，我們使用 Image 類別來儲存影像以及顯示影像．其實 .Net Framework 還提供了另一個類別 Bitmap．Bitmap 類別提供了像點(pixel) 操作的服務，我們可以利用內附 GetPixel 方法得到指定位置的 R,G,B 顏色資訊．進行點對點的影像處理．例如我們可以利用這個方法進行彩色影像轉成黑白影像的處理．我們也可以把整張影像的顏色資訊儲存成一個 3 維的整數陣列，以方便將來的影像處理工作．下面是實作一個函式 getRGBData 將影像資料轉成整數陣列的範例．呼叫 getRGBData 將傳回一個三維陣列．[0] 代表 Red，[1] 代表 Green，[2] 代表 Blue．

詳細程式碼，可以像這樣寫:

```
// 傳回 RGB 陣列資訊
        public int[,,] getRGBData() {
            // Step 1: 利用 Bitmap 將 image 包起來
            Bitmap bimage = new Bitmap(image);
            int Height = bimage.Height;
            int Width = bimage.Width;
            int[,,]rgbData=new int[Width,Height,3];

            // Step 2: 取得像點顏色資訊
            for(int y=0;y<Height;y++){
                for(int x=0;x<Width;x++){
                    Color color=bimage.GetPixel(x,y);
                    rgbData[x, y, 0] = color.R;
                    rgbData[x, y, 1] = color.G;
                    rgbData[x, y, 2] = color.B;
                }
            }
            return rgbData;
        }
```

將彩色照片變成黑白照片

我們舉一個簡單的數位影像處理的例子．假設我們想要把一張彩色照片變成一張黑白灰階顏色的照片，基本上只要把整個影像上，所有像點(pixel)的 R,G,B 值設成一個固定的灰階值即可．一般來說，這個灰階值可以爲 $\frac{R+G+B}{3}$．利用 GetPixel 取得目前影像的顏色資料，接著使用 SetPixel 方法把灰階值寫進去．這樣就可以完成簡單的黑白影像的工作．

詳細的作法如下．

```
public void doGray(int[,,]rgbData) {
            // Step 1: 建立 Bitmap 元件
            Bitmap bimage = new Bitmap(image);
            int Height = bimage.Height;
            int Width = bimage.Width;

            // Step 2: 設定像點資料
            for (int y = 0; y < Height; y++) {
                for (int x = 0; x < Width; x++) {
```

```
                int gray = (rgbData[x, y, 0] + rgbData[x, y, 1] + rgbData[x, y, 2]) / 3;
                bimage.SetPixel(x, y, Color.FromArgb(gray, gray, gray));
            }
        }
        // Step 3: 更新顯示影像
        image = bimage;
        this.Refresh();
    }
```

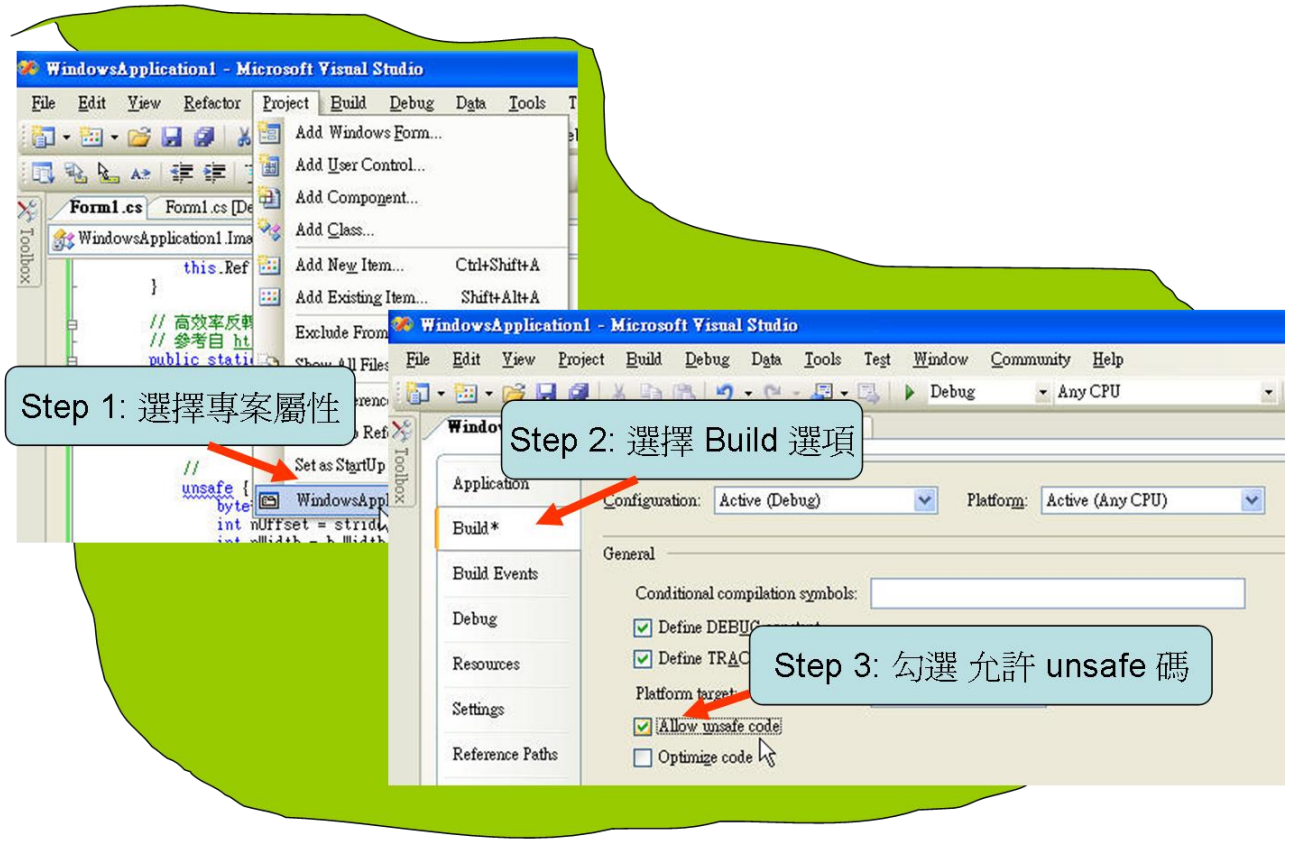
效率問題的討論

如果你去實作上面的程式，你會發現使用 Bitmap 類別的 SetPixel 與 GetPixel 方法存取影像資訊非常的緩慢．爲了解決這個問題，讓我們先來分析爲什麼會慢 ？

我們已經知道 SetPixel 是用來設定顏色資料用的，GetPixel 是用來讀取顏色資訊的方法．所以若要針對影像中每一個像點作影像處理，那麼每取一個像點的彩色資訊，就要呼叫一次 GetPixel，計算後得到新的顏色值，也要呼叫 SetPixel 設定．若影像中每個座標的彩色資料都要存取，舉一個例子：一個 300*400 大小的影像，使用 SetPixel 與 GetPixel 進行顏色的設定，將需要 120000 *2 =240000 次的函式呼叫．這還只是影像存取的動作而已，不包含中間的影像處理函式所需要的時間，這在效率上是一個驚人的浪費．這也就是爲什麼很少人直接使用這兩個函式來存取數位影像的顏色資訊．在這裡有必要提出一個有效率的方法，來解決這個問題．

如果有學過 C/C++ 的人，應該對這種情況非常熟悉．老朋友又來了 --- 指標．如果能使用指標直接存取存放在 Bitmap 物件中的影像資料，那就可以免去數以十萬計的函式呼叫．然而如果你想使用指標，那你必須使用 unsafe 這個關鍵字，來宣告使用指標的那些程式段落可能會危害系統．這在受管理的程式如 C# 等語言必須特別加以註明．另外，如果你想在程式中使用 unsafe 程式片段，那麼整合開發環境(IDE) 也要作相關的設定，使得該專案允許使用 unsafe 程式碼．一般預設的情況下，這個選項通常都是 disable．

致能 unsafe 選項，在 Visual Studio .Net 2005 的詳細設定如下



圖說: unsafe 關鍵字與 Visual Studio 2005 的相關設定
圖檔: unsafe 關鍵字.png

當我們在 IDE 上勾選了允許 unsafe 程式片段的選項後，接下來就是講解如何使用指標存取 Bitmap 物件中存放的影像資料.

如果你使用以前的 C++，沒問題！那就直接將指標指向目的地就可以直接存取了．但是在受管理的環境下，我們要先將指標的目的記憶體區塊鎖在系統記憶體中，以免自動記憶體管理程式把這整段區塊移到其他位址去了.

下面是 Bitmap 類別提供的鎖住記憶體內容方法.

```
[C#] Bitmap:: LockBits 方法
// 回傳該 Bmp 影像的屬性
public BitmapData LockBits(
    Rectangle rect,        // 指定要鎖住的影像範圍
    ImageLockMode flags,   // 指定記憶體的拴鎖模式
    PixelFormat format     // 指定這個 Bitmap 的資料格式
);
```

使用 LockBits 方法,會傳回一個 BitmapData 元件表示 Bitmap 的內容，我們來看看 BitmapData 類別的成員,有哪些資料可以幫助我們作指標定位.

Height	Bitmap 元件的高度 (以 pixel 爲單位)
PixelFormat	Bitmap 元件內的 pixel 資訊格式
Resereved	保留
Scan0	Bitmap 影像資料的起始位址
Stride	Bitmap 元件的 scan 寬度(單位 byte)
Width	Bitmap 元件的寬度(以 pixel 爲單位)

其中，我們將使用到 Height, Scan0, Stride, Width 等重要資訊. 而 Scan0 就是影像資料的起始位址. 指標就是要指向這個位址. 綜合上面的說明，若想要直接存取 Bitmap 元件的內容，可以分爲下面三個簡單的步驟.

Step 1: 鎖住 Bitmap 整個影像內容

因爲自動記憶體管理員的關係，所以首先我們要先鎖住整個影像的內容. 而在記憶體拴鎖模式方面，我們設定爲 ImageLockMode.ReadWrite，這是因爲我們希望等一會兒直接讀取 Bitmap 上的影像資料，經過計算後，接著直接將結果寫回 Bitmap 影像物件中. 最後我們指定 pixel 目前資料的操作格式爲 Format24bppRgb.

程式碼範例如下:

```
BitmapData bmData = bimage.LockBits(new Rectangle(0, 0, bimage.Width, bimage.Height),
                                    ImageLockMode.ReadWrite,
                                    PixelFormat.Format24bppRgb);
```

Step 2: 取得影像資料的起始位址

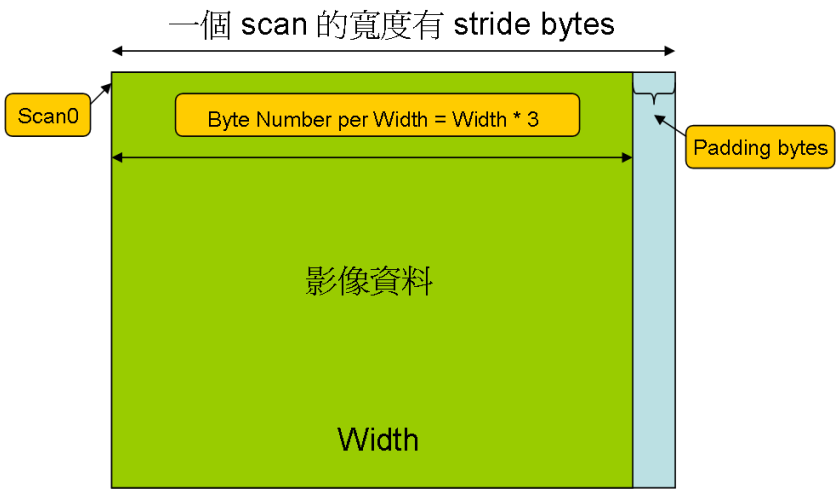
```
System.IntPtr Scan0 = bmData.Scan0;
```

Step 3: 直接利用指標進行影像處理

```
// 這是一個顏色反轉的範例
unsafe{
        byte* p = (byte*)(void*)Scan0;
        p[0] = (byte)(255 - p[0]); // 彩色資料反轉
}
// 最後別忘了解開記憶體鎖
bimage.UnlockBits(bmData);
```

在這裡有些細節要提醒一下. 由於 Bitmap 內部存放彩色像點的方式是以 WORD 爲單位，也就是說，如果一行上面的像點總和長度不是 WORD 長度的倍數，那麼就會用一些無用的 byte 空間補上來，以便湊齊 WORD 的倍數. 這些 bytes 不表示任何東西，他們只是單純地

用來湊數用的．我們稱這個動作爲 Padding．這些 Padding bytes 對我們有甚麼關係呢? 有 ！我們要注意的是，在每次移動指標的時候，每次到一行的最後像點時，不要忘記跳過那些 Padding bytes，才能繼續往下存取像點資訊．如果不跳過 Padding bytes 將會造成錯誤．用圖來說明，可能會讓你更明白．



圖說: Padding bytes 與影像資料的關係.
圖檔: BitmapData 的資料說明.png

計算每行有幾個 Padding bytes 長度的計算方式如下:

```
int ByteNumber_Width = bimage.Width * 3;
// 計算每一行後面幾個 Padding bytes
 int ByteOfSkip = stride - ByteNumber_Width;
```

高效率影像處理實作 -- 顏色反轉

下面是一個高效率直接使用指標存取影像的程式碼範例，以後我們將一直套用這個高效率程式骨架，進行一系列的影像處理．

```
        // 高效率反轉圖片
        // 修改與參考 Christian Graus 先生的文章
        // 詳見 http://www.codeproject.com/cs/media/csharpgraphicfilters11.asp
        public static bool Invert(Bitmap bimage) {
            // Step 1: 先鎖住存放圖片的記憶體
            BitmapData bmData = bimage.LockBits(new Rectangle(0, 0, bimage.Width, bimage.Height),
                                                ImageLockMode.ReadWrite,
                                                PixelFormat.Format24bppRgb);
            int stride = bmData.Stride;

            // Step 2: 取得像點資料的起始位址
```

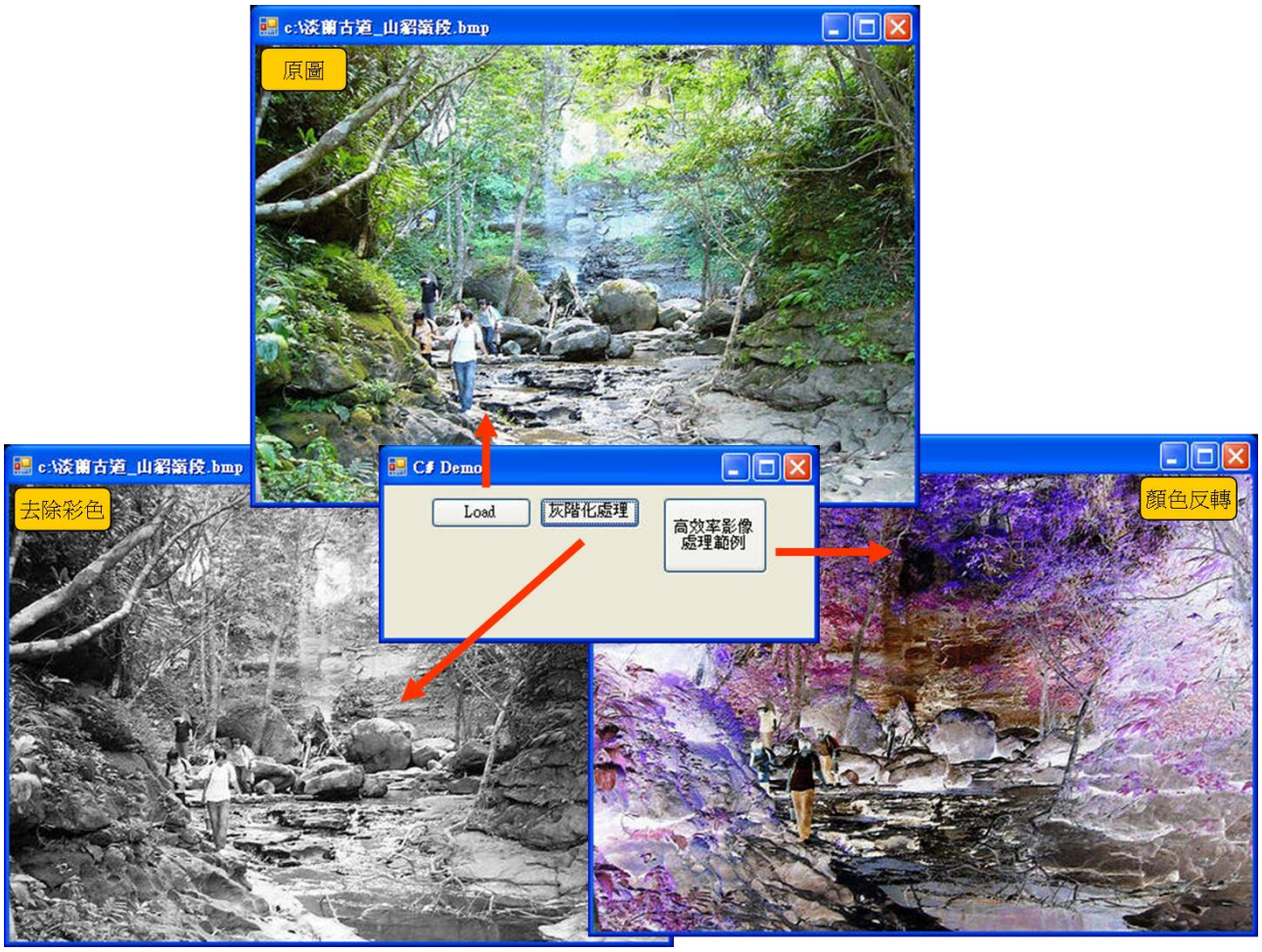
```
        System.IntPtr Scan0 = bmData.Scan0;

        // 計算每行的像點所佔據的byte 總數
        int ByteNumber_Width = bimage.Width * 3;

        // 計算每一行後面幾個 Padding bytes
        int ByteOfSkip = stride - ByteNumber_Width;

        // Step 3: 直接利用指標，更改圖檔的內容
        int Height = bimage.Height;
        unsafe {
            byte* p = (byte*)(void*)Scan0;
            for (int y = 0; y < Height; y++) {
                for (int x = 0; x < ByteNumber_Width; x++) {
                    p[0] = (byte)(255 - p[0]); // 彩色資料反轉
                    ++p;
                }
                p += ByteOfSkip; // 跳過剩下的 Padding bytes
            }
        }
        bimage.UnlockBits(bmData);
        return true;
    }
```

現在請你把以前的程式改寫，你會發現使用指標這個方法相當有效率，原本數十秒的影像處理動作，現在可能不到一秒就可以完成．由此可知，數十萬次的 SetPixel/GetPixel 函式呼叫是多麼的浪費時間．當然，我們付出的代價是如果沒有好好思考程式的行爲，可能會讓程式不安全地執行．另外，程式的可讀性也變差了．下面是顏色反轉與灰階化影像執行的結果．



圖說: 去除彩色以及顏色反轉的結果
圖檔: 去除彩色以及顏色反轉的結果.png

寫過 C++ 程式語言的人都知道指標不易掌控. 爲了讓程式的可讀性與安全性問題, 鎖在一個區域不讓它擴散在程式碼的每一個角落, 一個有效的方法就是建造高效率的影像處理轉換函式集, 把所有關於指標部分不安全的程式碼放在這些固定的函式中, 而所有的影像處理程式則完全建立在這些工具上. 這樣做的目的是當發生問題時, 我們可以很容易的把目光焦聚在這些函式, 並進行檢視.

高效率影像處理工具

由前面的解說, 我們知道利用指標的直接存取影像資料威力, 現在我們再把剛剛使用 SetPixel/GetPixel 完成的讀取影像 RGB 值的函式, 改寫成高效率版本的 setRGBData 與 getRGBData. 我們希望把所有不安全的程式都封裝在 getRGBData_unsafe() 以及 setRGBData_unsafe()上面. 由於篇幅的關係, 所以我們只列出高效率版本讀取影像資料程式, 如果你想要知道如何 setRGBData_unsafe() 如何實作, 請至電腦王下載完整程式範例.

(高效率版本) 讀取影像資料程式.

```
// 高效率圖形轉換工具 -- 讀取影像資料
```

```
public int[, ,] getRGBData_unsafe() {
    Bitmap bimage = new Bitmap(image);
    return getRGBData(bimage);
}
public static int[, ,] getRGBData(Bitmap bimage) {
    // Step 1: 先鎖住存放圖片的記憶體
    BitmapData bmData = bimage.LockBits(new Rectangle(0, 0, bimage.Width, bimage.Height),
                                        ImageLockMode.ReadOnly,
                                        PixelFormat.Format24bppRgb);
    int stride = bmData.Stride;
    // Step 2: 取得像點資料的起始位址
    System.IntPtr Scan0 = bmData.Scan0;
    // 計算每行的像點所佔據的byte 總數
    int ByteNumber_Width = bimage.Width * 3;
    // 計算每一行後面幾個 Padding bytes
    int ByteOfSkip = stride - ByteNumber_Width;
    int Height = bimage.Height;
    int Width = bimage.Width;
    int[, ,] rgbData = new int[Width, Height, 3];

    // Step 3: 直接利用指標, 把影像資料取出來
    //         請注意先取 B 然後是 G, 最後才是  R
    unsafe {
        byte* p = (byte*)(void*)Scan0;
        for (int y = 0; y < Height; y++) {
            for (int x = 0; x < Width; x++) {
                rgbData[x, y, 2] = p[0];    // B
                ++p;
                rgbData[x,y,1]=p[0];        // G
                ++p;
                rgbData[x,y,0]=p[0];        // R
                ++p;
            }
            p += ByteOfSkip; // 跳過剩下的 Padding bytes
        }
    }
    bimage.UnlockBits(bmData);
    return rgbData;
```

```
        }
```

在這裡要特別說明一下，因爲 Windows 24bit Bitmap 圖檔格式在 R,G,B 的顏色排列方式，是先放 B，中間放 G 最後才是放 R，也就是說從低位元到高位元的排列方式爲 B,G,R．所以我們使用讀取顏色的順序也要依照此順序一筆一筆的讀回來．現在有了 getRGBData_unsafe() / setRGBData_unsafe() 這兩個函式工具，我們就可以將心力集中在數位影像的彩色資訊上，而非影像的特殊檔案格式上打轉．其實東西只要一複雜，那就容易出錯．這是一個不變的道理．現在就馬上寫一個顏色濾鏡的程式出來，給大家看看．

彩色濾鏡實作

基本上,一個紅色濾鏡程式,就是對每一個像點只留下紅色資訊，其他顏色的資訊全部設定爲 0．很簡單吧．詳細的作法如下:

```
// 高效率影像處理 --  紅色濾鏡範例
        private void button4_Click(object sender, EventArgs e) {
                // Step 1: 取出顏色資料
                int[, ,] rgbData = CurrentImage.getRGBData_unsafe();

                // Step 2: 數位影像處理
                //  將所有顏色 Channel 資料改成 0, 只留下紅色資訊
                int Width = rgbData.GetLength(0);
                int Height = rgbData.GetLength(1);
                // int r;
                for (int y = 0; y < Height; y++) {
                    for (int x = 0; x < Width; x++) {
                        // r=rgbData[x, y, 0]; // Red Channel 保留不動
                        rgbData[x, y, 1] = 0; // Green Channel 改成 0
                        rgbData[x, y, 2] = 0; // Blue Channel 改成 0
                    }
                }
                // Step 3: 將處理後的資料寫回 CurrentImage
                CurrentImage.setRGBData_unsafe(rgbData);
}
```

因爲篇幅的關係，綠色與藍色濾鏡的程式實作，就留給各位當作業．我們先來看看彩色濾鏡的執行結果:

圖說：高效能影像處理彩色濾鏡
圖檔：高效能影像處理彩色濾鏡.png

增加亮度實作

對於一張很暗的數位照片，如果想要增加亮度最簡單的方法就是對 RGB 加入一個固定的亮度值．例如我們希望增加的亮度值爲 30，那麼作法就是 R+30，G+30，B+30．這麼簡單．然而在這個想法中，有個關鍵那就是你必須檢查[增加亮度的結果是否超過可儲存的容量範圍]．爲什麼要作檢查呢? 原因是在 24bit BMP影像格式的情況下，那麼每一個顏色只使用 8個位元存放,也就是說一像點的顏色分量只能表示 $2^8 = 256$種顏色變化，可存放的顏色範圍爲 [0 – 255]，所以當你存放的值超過 255 時，就會產生錯誤．這裡要特別的注意．檢查數值大小的程式碼，列表如下:

```
// 檢查運算結果 g 的大小
if (g > 255)
```

```
    g = 255;
if(g<0)
    g=0;
```

完整的增加亮度程式作法如下:

```
// 高效率影像處理範例 --  增加亮度 30
        private void button7_Click(object sender, EventArgs e) {
            // Step 1: 取出顏色資料
            int[, ,] rgbData = CurrentImage.getRGBData_unsafe();
            int Width = rgbData.GetLength(0);
            int Height = rgbData.GetLength(1);

            // Step 2: 增加亮度 30
            int g;
            for (int y = 0; y < Height; y++) {
                for (int x = 0; x < Width; x++) {
                    for (int c = 0; c < 3; c++) {
                        g = rgbData[x, y, c];
                        g += 30;
                        if (g > 255)
                            g = 255;
                        rgbData[x, y, c] = g;
                    }
                }
            }

            // Step 3: 將處理後的資料寫回 CurrentImage
            CurrentImage.setRGBData_unsafe(rgbData);
        }
```

下面是執行的結果.

圖說：左邊是比較暗的原圖，右邊是將原圖每個點加上 30 的結果
圖檔：增加亮度.png

影像存檔

接下來，要如何將影像處理的結果存起來呢？使用 .Net Framework 的好處是微軟都幫你寫好了存檔的相關程式碼，只要你呼叫 Image 類別的 Save 方法即可完成．部分存檔格式使用範例如下．

```
image.Save(@"c:\Result.bmp", ImageFormat.Bmp);
image.Save(@"c:\Result.gif", ImageFormat.Gif);
image.Save(@"c:\Result.png", ImageFormat.Png);
image.Save(@"c:\Result.jpg", ImageFormat.Jpeg);
image.Save(@"c:\Result.exif", ImageFormat.Exif);
```

圖說：影像處理存檔的結果
圖檔：影像處理存檔的結果.png

結論

這次我們介紹了如何使用 .Net Framework 搭配 C# 程式語言，進行簡單的影像處理，我們也提到了如何使用指標直接存取影像內容，提高影像處理的效率．科學家始終企圖將人類的智能轉移到電腦上，利用機械自動化取代人類的大腦．這樣的努力過去稱之爲人工智能或人工智慧．近年來,人們轉而將心力專注在特定的領域上．例如: 自動車導航，自動文字辨識，自動人臉偵測與辨識，自動尋找電路版上的瑕疵等專門領域的應用．在這些領域上的應用很多都已經有良好的商業用途．影像處理可說是許多應用的第一步．今天教大家的內容是影像處理的墊腳石，希望大家有所收穫．下一期我們將會介紹如何使用 Java 進行簡單的影像處理．

指標的概念是什麼，

在電腦程式語言中，我們可以用變數來存放不同型態的資料．如果我們要用一個變數來存放整數，那麼我們就要宣告一個整數型態的變數．例如: int 這個資料型態是用來存放整數資料．C# 程式的寫法就像 int x=1234; 這樣．把 1234 這個整數值存放到 x 這個整數型態的變數中．除了整數之外，還有一些用來存放其他的資料型態變數，如 浮點數 float，位元組 byte 等．然而，指標是一種資料型態，與其他型態不同的是指標變數型態，指明了該變數是專門用來存放記憶體的位址．

我們都知道在電腦中，任何資料全部都是存在記憶體中．而電腦中的每個記憶體空間都有一個唯一的序號對應．這就好像每個房屋都有一個地址一樣，可供我們經由地址找到朋友．這意味著我們可以有兩種方式來讀取或寫入資料到記憶體中．第一: 直接使用變數名稱明確的讀取存在記憶體中的資料進行運算，例如: int y=x+1; 就是把 x 變數中的資料讀取出來 加上 1 然後存放到 y 變數中．而變數各自代表者不同的記憶體空間．另外，我們也可以利用指定記憶體位址的方式來讀取寫入資料．例如;

```
unsafe{
int x=1234;
int* px=&x; // 取得 x 變數的記憶體位址，存放到 px 指標變數中.
int y=*px+1; // 取得指定記憶體位址中的資料(1234) 加 1 放到 y 變數中.
}
```

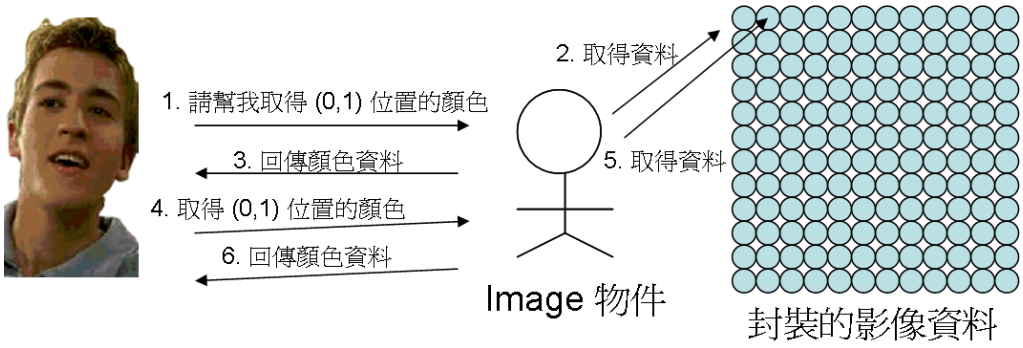
現在我們來討論爲什麼要這麼麻煩，引入指標這樣的觀念．其實一點都不麻煩 !!

爲什麼用了以後會提升存取效率？

電腦記憶體是一塊連續的空間．所謂的連續空間指的是下一筆資料的位址與目前資料的位址是一種遞增的數值．這個觀念很重要．這意味著當我們存放的資料是大筆資料的時候，使用指標就特別的有用．例如: 圖片的資料，就是大筆資料的一個例子它就佔據了大片連續的記憶體． 所以假設我們的指標變數 px 內存放的是第一個 pixel 的記憶體位址，如果想要存取其中下一個 pixel 的內容，很簡單，只要將 px 遞增就可以存取下一筆資料的內容．

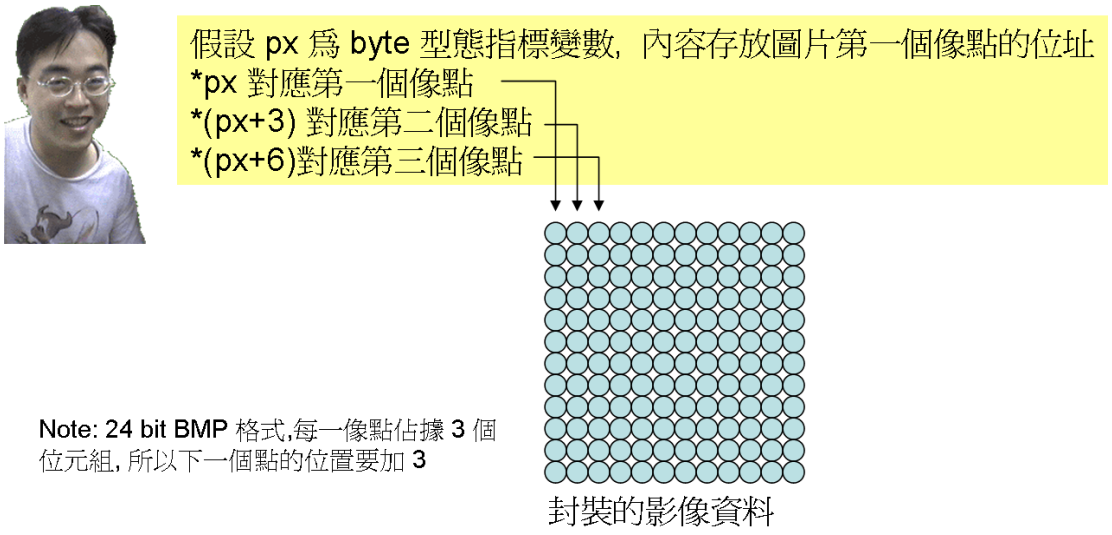
利用指標存取圖形資料比起呼叫 GetPixel/SetPixel 方法，還要直覺與有效率．使用 Image 類別的方法進行影像處理，就好像是每次要取得圖形資料時都要經過一個代理人幫我們處理．如下圖:

@圖 GetPixel.png



@圖說: 使用 GetPixel 的方法，讀取兩筆影像資料的示意圖

然而使用指標呢，則是不需要代理人 Image 物件，我們長驅直入影像資料的儲存地點，把資料讀出來．下面爲指標連續存取三個影像點的示意圖．是不是比較快比較直接?



圖說: 使用指標直接讀取三筆影像資料

@圖: pointer.png

爲什麼指標的程式段落會是危險的呢 ?

指標是個既危險又吸引人的工具，在電腦的世界裡，只要使用指標變數，你就可以對該記憶體位址的內容進行寫入或讀取的行爲．這對系統來說是非常危險的，一個錯誤的存取可能覆寫了底層正在執行的共通語言執行環境 CLR (Common Language Runtime)的狀態，進而導致程式以不預期的方式執行，可能意外當掉或者發生不可預期的問題，這種錯誤無法

預防．所以這也就是爲甚麼使用指標的程式難以除錯的主要原因．新一代的程式語言 C# 強迫使用者加入 unsafe 關鍵字標明他知道這段程式碼是不安全的，並且在整合環境中加入致能使用 unsafe 的選項．

何謂**自動記憶體管理? 在 C# 程式中，爲何要將記憶體鎖住呢?**

一般來說，任何使用 C# 所寫的程式碼，我們稱之爲列管的程式碼(Managed Code)，這些列管的程式受到底層的共通語言執行環境(CLR)所控管．CLR 讓程式設計師不用再去管理記憶體以及配置與刪除記憶體等工作．取而代之的是由一個自動記憶體管理員幫你作這些事情．由程式接手管理記憶體的好處是我們完全可以避開因爲記憶體管理所造成的種種問題．例如: 程式員配置記憶體後,忘記執行釋放記憶體的指令，這種問題稱爲記憶體漏失(Memory Leak)問題．陣列溢出存取(Array access overflows) 問題，也因爲 CLR 的自動記憶體管理機制而獲得完全的預防．這些問題在 C/C++ 所寫的程式中特別常見，只要一出錯，往往不是當機收場，就是程式突然間掛掉．最近的程式語言與軟體系統架構的發展趨勢，已經傾向以自動化的方式管理記憶體，畢竟以手動方式管理記憶體的代價實在太高了．.Net Framework 是一個新的架構，包含了 CLR 與 一個名爲 .Net Framework 的類別程式庫．所有在 CLR 上面接受管理的程式，記憶體全部交給自動管理員幫你管理．這意味著你的資料可能不會永遠固定存在於某個地方．只要在 Managed 堆積中有位址空間可用，CLR 就會繼續爲新物件配置記憶體．但是，記憶體是有限的。最後，CLR 中記憶體回收行程還是必須進行回收以釋放某些記憶體。我們在使用指標的時候，就必須確保這個位址所參考到的資料，不會受制於記憶體回收行程的重新配置或配置。 所以在這篇文章中，我們利用指標直接存取圖檔資料，必須要先用 Bmpimage LockBits 把圖檔資料鎖定在系統記憶體中，等到存取完畢後，才進行解鎖的動作．

井民全

國立交通大學 自動化資訊處理實驗室

專長：數位影像處理

Email：mqJing@msn.com

個人網頁：http://debut.cis.nctu.edu.tw/~ching

附錄一：圖片及圖片圈示

點陣列影像示意圖.png

Visual Studio 2005 建構專案步驟_1.png
Visual Studio 2005 建構專案步驟_2.png
ImageForm 秀圖結果.png
使用 OpenFileDialog 選擇檔案.png
放大兩倍.png
形變的示意圖.png
unsafe 關鍵字.png
BitmapData 的資料說明.png
去除彩色以及顏色反轉的結果.png
高效能影像處理彩色濾鏡.png
增加亮度.png
影像處理存檔的結果.png

參考資料:

[1] JPEG (Joint Photographic Experts Group) 官方網站 http://www.jpeg.org/
[2] 有關 BMP (Windows bitmap) 的詳細資訊, 請參閱
http://en.wikipedia.org/wiki/Windows_bitmap

[3] GIF(Graphics Interchange Format),

可參閱 GIF89a 的規格書 http://www.w3.org/Graphics/GIF/spec-gif89a.txt

[4] PNG (Portable Network Graphics) 的相關資訊,
官方網站: http://www.libpng.org/pub/png/
[5] LEADTOOLS 公司的網址 http://www.leadtools.com/
[6] Christian Graus 先生的文章
http://www.codeproject.com/cs/media/csharpgraphicfilters11.asp