第 18 章　影像處理程式

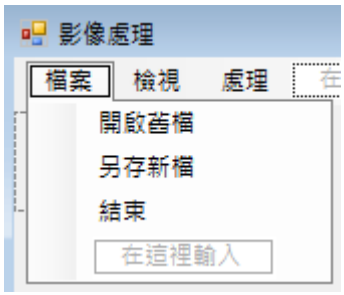
簡介：

之前我們已經在秀圖軟體與小畫家單元中學過不少影像物件的操作，但是真正的影像處理應該是像彩色變黑白、圖片變亮、變暗或改變影像實際大小等等，真正將影像本質作改變的技術。雖然目前影像處理軟體已經非常成熟，但是如果你的科系必須學會影像處理的原理(如電機系)，甚至你未來想進行數位影像領域的研究，這個單元就非常適合你了！從程式操作中我們可以較深入的了解影像結構與原理，為你的進一步的學習或研究打下良好基礎。

18-1 建立開啟影像與存檔的程式介面

[建立主功能表]

請開啓新專案，加入一個 menuStrip1 主功能表，先填入以下項目。



接著自工具箱取出 openFileDialog1 與 saveFileDialog1 對話方塊，設定他們的 Filter 屬性皆為『*.jpg|*.jpg』，就是簡化本單元處理的對象皆為 jpg 檔案。當然我們也需要一個 pictureBox1 來裝載被處理的影像，將其 SizeMode 屬性設為 AutoSize(自動縮放到原圖大小)即可。接著請點擊三個功能表項目寫入程式如下，至此你的程式就可以開啟與儲存檔案了！這些程式與小畫家單元幾乎完全相同，只是多宣告了一個公用的影像物件 B 用來記錄載入影像的初始狀態，稍後就是拿這個物件 B 作為影像處理的目標。

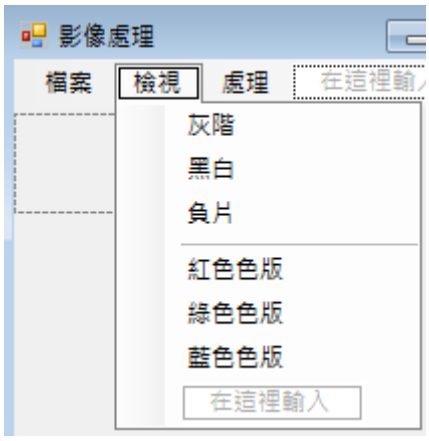
```
Bitmap B;
private void 開啟舊檔ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog()  == System.Windows.Forms.DialogResult.OK)
    {
        B = new Bitmap(openFileDialog1.FileName);
        pictureBox1.Image  = B;
    }
}
```

```
private void 另存新檔ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog()  == System.Windows.Forms.DialogResult.OK)
    {
        Bitmap S = new Bitmap(pictureBox1.Image);
        S.Save(saveFileDialog1.FileName,  System.Drawing.Imaging.ImageFormat.Jpeg);
    }
}

private void 結束ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Application.Exit();
}
```

18-2 檢視 RGB 色版的功能

請在檢視功能表輸入以下項目：



[三原色的解析抽離]

我們先從較簡單的色版處理開始，請記得標準的數位影像全彩模式是以 RGB，就是紅(Red)綠(Green)藍(Blue)三原色組成顏色資訊，每種顏色亮度值範圍是 0~255，也就是一個 byte(位元組)的值域，因為 C#很注重資料型別，所以請先記得每個顏色的亮度資料型別是 byte。請雙擊『紅色色版』項目寫程式如下：

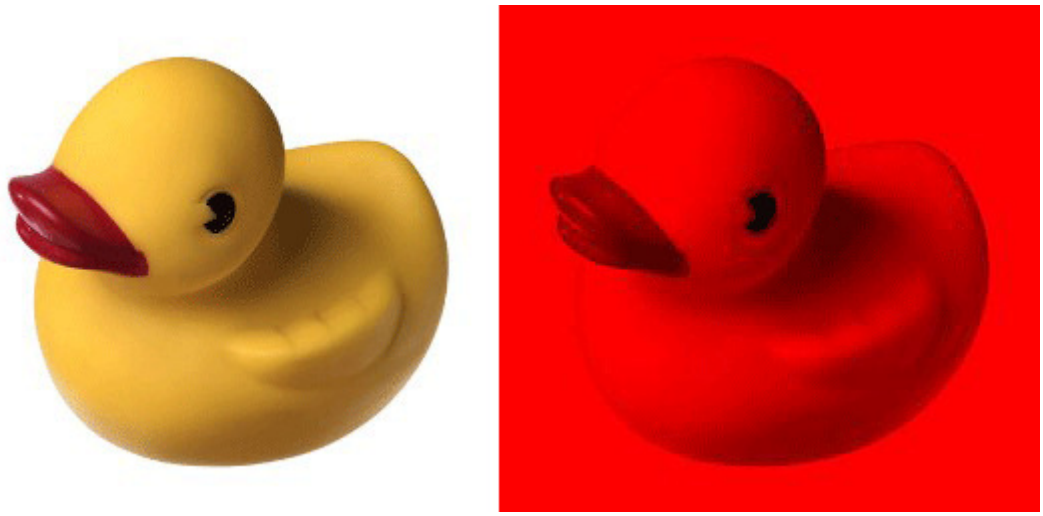
```
private void 紅色色版ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            C = Color.FromArgb(C.R, 0, 0);
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

上面程式首先宣告一個準備承接處理結果的影像物件 P，它的大小(Width & Height)

應該與載入的 B 影像物件相同(B.Width, B.Height)。須注意：程式這樣寫，如果你在沒有載入影像之前就按紅色色版是會當掉的！因為 B 還不存在嘛！接著是一個雙重迴圈，就是將影像上面的各個像素點(Color C)逐一取出(GetPixel)運算的意思。我們在打磚塊單元也用過類似的雙層迴圈，當時是用來建立 10×10 的磚牆物件，二維資料的處理多半是這樣作的。

[只抽取紅色回存]

取出 C 點後，我們要的是該點顏色的紅色成分，可使用 C.R 屬性取出，然後用 Color.FromArgb 方法重組顏色點 C 的成分，使得藍與綠色值(第二與第三個參數)都為 0，就是只剩紅色成分了！再將此顏色點放到 P 影像物件對應的位置(i, j)上面。跑完迴圈後 P 物件就是處理完成的影像，將它貼到 pictureBox1 的 Image 屬性就可以看到結果了！如下圖左是橡皮鴨影原始影像，右邊是像抽取的紅色色版。



[綠與藍色的色板抽取]

同理可證，綠色與藍色的色版程式如下：

```
private void 綠色色版ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            C = Color.FromArgb(0, C.G, 0);
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

```
private void 藍色色版ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            C = Color.FromArgb(0, 0, C.B);
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

18-3 灰階與黑白影像

[RGB 三色亮度相等稱為灰階]

所謂灰階就是 RGB 三色的值都變成一樣，全部為 0 時是黑色，全部為 255 時就是白色了！如何合理的讓彩色變成灰階？最直接的想法是將三原色平均，以平均值取代原來的三色成分值就是灰階了！

[小心！顏色值為 Byte 型別]

這種數學很簡單，比較麻煩的地方是 C#對於資料型別的嚴苛，計算過程中必須隨時小心型別的使用。譬如 C.R=100 與 C.G=200 加起來就是 300，但是兩個數值(C.R & C.G)原本型別都是 byte(範圍 0~255)相加後的答案也會預設是 byte 整數，可是將一個 300 的數字放到 byte 型別的整數時程式就先當掉了！

要避免這個問題，計算之前就必須將資料型別(C.R，C.G 與 C.B)擴大為可以容納超過 255 數值的 int 整數型別！譬如指定將 C.R 變成整數型別的寫法就是**(int)**C.R，所以變成灰階的程式應該如下：

```
private void 灰階ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            Byte K = (byte)(((int)C.R + (int)C.G + (int)C.B) / 3);
            C = Color.FromArgb(K, K, K);
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

上面程式中的 K 就是三原色的平均值，它還是必須為 byte 型別，但是計算三色值相加時有可能會超過 255，所以都先轉成 int，相加除以 3 之後確定不會超過 255 了，再將型

別轉回為 byte。挺麻煩的！不是嗎？不過使用型別特別嚴謹的 C#目前還是必須如此。轉換成灰階的橡皮鴨影像如下：

[黑白影像]

黑白影像只有全黑與全白兩色，最簡單的處理方式是設定一個介於 0~255 之間的門檻值(在此就用 127 了)，大於門檻就變成白色(255)，小於門檻就設為黑色(0)，程式碼如下，還是以灰階為基礎，再根據門檻分成兩色：

```
private void 黑白ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            Byte K = (byte)(((int)C.R + (int)C.G + (int)C.B) / 3);
            if (K > 127)
            { C = Color.White; }//白色
            else
            { C = Color.Black; }//黑色
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

執行黑白影像結果如下：

當然因為只有黑白兩色而且門檻值也未針對特徵調整，結果影像看起來並不清楚是甚麼東西？真正黑白影像的重要作用其實是在影像辨識過程中，在我們已經強化了顏色區塊的對比之後，用適當的門檻值取黑白影像就會達到將目標區塊更清楚辨識的目的。譬如一架飛機在天空飛行的照片，經過強化之後再作黑白分割應該就可以確定飛機的位置了！

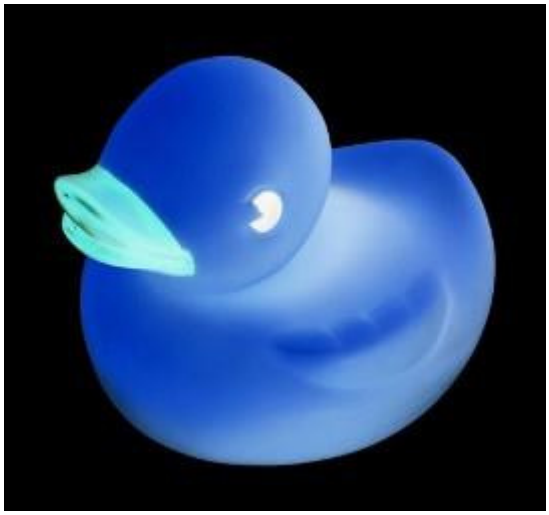
18-4 負片影像

[亮度值相反]

負片就是指各種顏色的亮度值相反，亮的變暗，暗的變亮。因為各個顏色值介於 0~255 之間，所以所謂的『相反』就是以 255 減去各顏色值，再作成新的顏色點畫回去就行了！程式碼如下：

```
private void 負片ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            C = Color.FromArgb(255 - C.R, 255 - C.G, 255 - C.B);
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

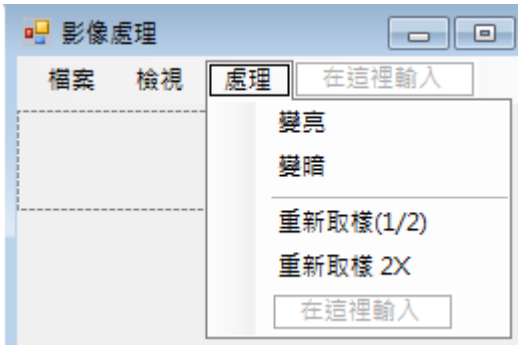
執行結果如下，原來偏紅黃的影像變成偏藍綠的互補色了！



18-5 亮度處理

[建立功能表]

接下來請建立『處理』項目如下：



[變亮時的漲停板機制]

先從變亮開始，就是要將原影像各點的 RGB 值一起變大！此時可能會有些值超過 255 的範圍，因此和之前的灰階計算相似，增大亮度值時必須先轉為 int 資料型別，如果計算結果超過 255 應該將它設定為漲停板的 255，再轉回 byte 資料型別。個別的 RGB 都如此處理之後再將 RGB 三色重組為新的較亮的顏色，程式碼如下：

```
private void 變亮ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            int Rc = (int)C.R + 50;
            if (Rc > 255) { Rc = 255; }
            int Gc = (int)C.G + 50;
            if (Gc > 255) { Gc = 255; }
            int Bc = (int)C.B + 50;
            if (Bc > 255) { Bc = 255; }
            C = Color.FromArgb((byte)Rc, (byte)Gc, (byte)Bc);
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

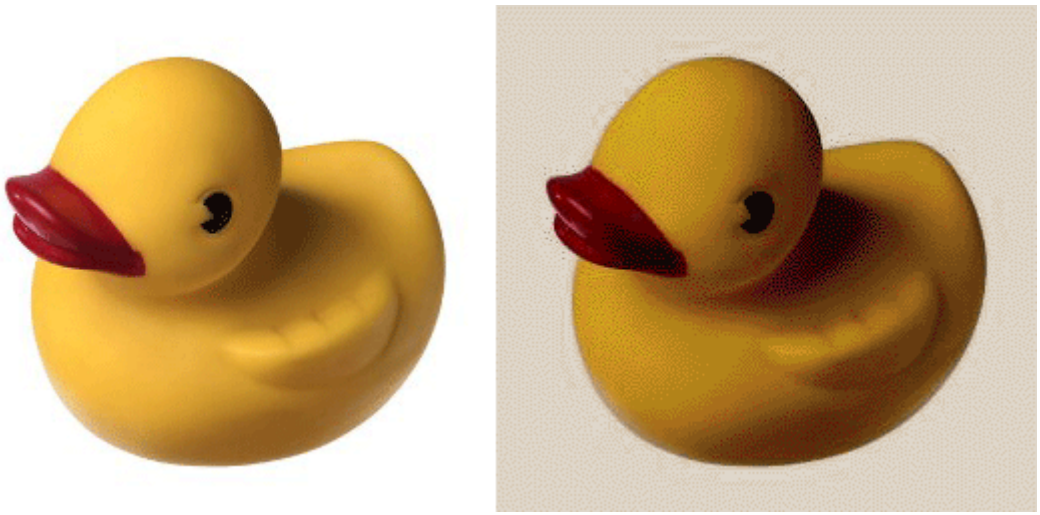
執行結果與原圖比較如下：

[變暗時的跌停板機制]

變亮要加 50，變暗的程式碼當然就是 RGB 都一起減少 50 了！此時可能發生的出軌情況是計算結果會小於 0，因此要設定亮度小於 0 時要將跌停板設定為 0，負的亮度值會讓程式當掉的。程式碼如下：

```
private void 變暗ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width, B.Height);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            int Rc = (int)C.R - 50;
            if (Rc < 0) { Rc = 0; }
            int Gc = (int)C.G - 50;
            if (Gc < 0) { Gc = 0; }
            int Bc = (int)C.B - 50;
            if (Bc < 0) { Bc = 0; }
            C = Color.FromArgb((byte)Rc, (byte)Gc, (byte)Bc);
            P.SetPixel(i, j, C);
        }
    }
    pictureBox1.Image = P;
}
```

執行結果與原圖比較如下：



18-6 重新取樣調整影像大小

重新取樣就是改變原圖的實際大小，如果是變小，通常就是跳過一些點，變大就是重複一些點，當然有些運算法可以比較精緻的用內插法計算橡素值，讓影像縮放之後顯得平滑一些，那是進階的學習，此處就只用作簡單的方式處理，以下是縮為 1/2 的程式碼：

```
private void 重新取樣12ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width / 2, B.Height / 2);
    for (int i = 0; i < B.Width; i += 2)
    {
        for (int j = 0; j < B.Height; j += 2)
        {
            Color C = B.GetPixel(i, j);
            int i2 = i / 2;
            int j2 = j / 2;
            if (i2 < P.Width && j2 < P.Height)
            {
                P.SetPixel(i / 2, j / 2, C);
            }               }
    }
    pictureBox1.Image = P;
}
```

[填充像素值到較小的影像物件]

上述程式首先宣告一個只有原圖 1/2 寬高的影像物件 P，接著在迴圈中跳點取得 P 影像(原圖)的資訊，參數 i+=2 與 j+=2 的意義就是每次前進 2 點(跳過一點)，取得的點置放到 P 時座標除以 2，因此原來索引(0, 2, 4…)的資料點會填充到新圖的索引(0, 1, 2…)的位置。這種運算必須小心的是不能將點填出(SetPixel)影像物件的範圍之外(當然取點 GetPixel 時也不能)，如果超出範圍程式會當掉的！所以前面的程式才需要先計算落點(i2, j2)，確定不大於輸出圖的範圍才作 SetPixel 的動作。蠻有趣的是：如果你不是用 SetPixel 而是使用繪圖物件 Graphics 繪圖時超出範圍則是不會當掉的，只是畫不出結果而已。下面是原圖與執行結果的對照。

[填充像素值到較大的影像物件]

下面是影像放大兩倍的程式：

```
private void 重新取樣2XToolStripMenuItem_Click(object  sender, EventArgs e)
{
    Bitmap P = new Bitmap(B.Width * 2, B.Height * 2);
    for (int i = 0; i < B.Width; i++)
    {
        for (int j = 0; j < B.Height; j++)
        {
            Color C = B.GetPixel(i, j);
            for(int ii = 0; ii < 2; ii++)
            {
                for (int jj = 0; jj < 2; jj++)
                {
                    P.SetPixel(i * 2 + ii, j * 2 + jj, C);
                }
            }
        }
    }
    pictureBox1.Image = P;
}
```

有何不同呢？先是宣告大兩倍的圖，然後每取得一點(i, j)就要用迴圈在垂直與水平方向都重複畫兩次(ii = 0~1，jj = 0~1)。這樣圖就大兩倍了！

18-7 進階挑戰

一、如何製作可以任意調整大小的重新取樣？

提示：以放大及縮小取樣的程式為範本，將倍數設為變數。

二、如何製作可交換色版的功能？

提示：交換 C.R、C.G 與 C.B 的數值。

三、如何製作模糊化效果？

提示：將相鄰像素點的亮度值平均。

課後閱讀

為什麼要學影像處理？

影像處理一向被認為是高度仰賴電腦處理能力的工作，影像資料點越密集，每個點的資訊越豐富，影像品質就越好。但是同時間也就需要越多的電腦記憶空間，要處理它們時也需要花費更多的電腦運算時間。拜電腦科技進步之賜，數位影像的畫素越來越多，計算處理的速度也越來越快，網路傳輸也快了，所以現在誰都可以輕鬆的使用不昂貴的軟體處理以及使用影像，豐富你的工作與休閒內容。

[影像處理已經是全民運動]

總之，影像處理不再是只存在實驗室或電影公司裡面的昂貴稀有資源。了解影像是電腦使用者必須學習的基本知識，處理影像也是所有電腦工作者都能執行的工作，不學影像處理就落伍了！

數位影像的基本結構

[像素點構成影像]

數位影像物件的原文是 Bitmap，中文翻譯是『點陣圖』，就是由點構成的矩陣，影像長寬各數百到幾千點，多數人都知道。一般人比較不清楚的是『像素點』的內容！最常見的如本單元所介紹的全彩模式是以RGB三原色各用一個byte儲存，一個點就是3個byte(位元組)總共 24 個 bit(位元)，所以又稱 24 位元全彩影像，有時候你會看到 32 位元的全彩模式(24+8)，那是指加上了一個 byte 紀錄顏色的透明度，就是我們在本單元使用 Color.Fromm**A**rgb 函數時的"A"了！透明度可以讓畫在此點後面的影像顏色可以部份『透』出來，其實就是與前景顏色相混合而已，多數的假靈異照片(阿飄？)就是這樣做出來的。

[像素點有很多種]

事實上，常見的像素點內容還有灰階以及索引色，一般是以 256 色來代表，如本單元的灰階其實是使用全彩模式來顯示灰階，因為它實際只有 0~255，就是 256 種組合顏色，我們可以只用一個 byte 來儲存。如果希望 256 色不是全部灰色，而是彩色，可以製作一個調色盤，自己定義第幾號顏色的 RGB 值各為多少，這樣就可以很省空間(全彩的 1/3)的儲存彩色影像了！目前的網路上常見的 GIF 檔案就是這樣做的，事實上經過較精緻的影像處理軟體，你還可以選擇索引色的數目，最多是 256 色，但是 128，64，32，16…色也都可以。因為顏色數減少，紀錄每個點的 bit 數目也可以減少，圖檔就變小了！

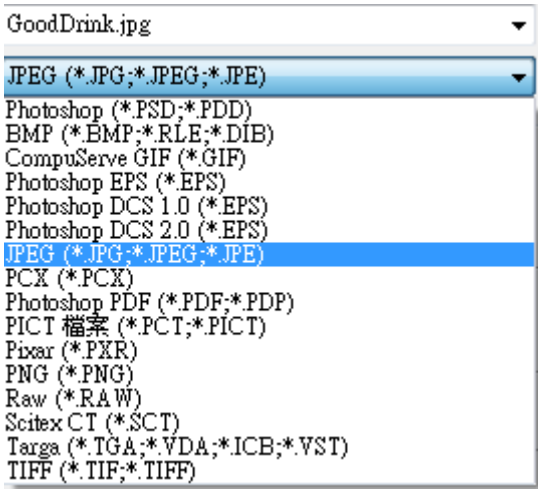
BMP、JPG 與 GIF

[格式多只因為想壓縮]

數位影像檔案格式繁多，雖然現在的軟體都很厲害，已經很少有人會因為格式問題傷腦筋，但是看得懂的人還是用得比較好，會有優勢的！譬如我常常收到一些研究報告或論

文原稿，因為裡面有很多圖，所以動輒上百 M！不但無法用 Email 寄送，開啟閱讀時也慢得不得了！但是相對的，我自己寫的研究計畫報告，甚至像這本書的整本稿件檔案都可以控制在 10M 以內！而且裡面的近兩百張影像品質依舊很好，沒有一張會模糊或有花邊！

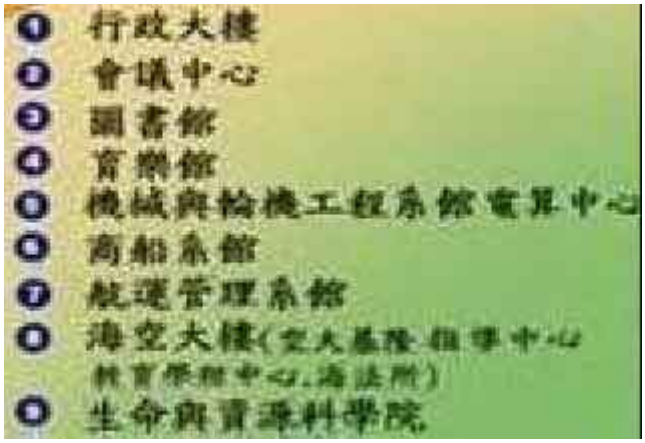
影像檔格式多的原因只有一個，就是資料壓縮！如果直接將點陣圖資料一點一點完整地存到電腦，就是 BMP 格式！玩過小畫家的人都知道，直接存預設的 BMP 會比其他檔案格式大很多，原因就是它沒有壓縮。在以往各影像產品公司都會自行開發不同的壓縮格式，因此一度有多達數十種格式在市面上流通，到現在你進入多數的影像軟體還是可以參觀到這些壯觀的古蹟，下圖是目前算老舊的 PhotoShop 7.0 版另存新檔的選項：



這種情況真是災難，當時有太多人常常拿著無法讀取的影像檔到處求援。現在好多了，實際流通的格式主要就是 JPG 與 GIF 而已，更新的 PNG 格式原本是希望兼具前兩者的優勢而設計，目前好像並不是很流通，或許是使用習慣尚未改變吧？

[正確選擇 JPG 與 GIF 使用時機]

其實 JPG 與 GIF 格式都會使得影像失真(與原圖不一樣，而且無法復原)，但是因為壓縮運算法不同，失真的狀況很不一樣！JPG 的壓縮是保留最豐富的色彩，但是將逐點的顏色變化用 SIN 函數取代(圓滑化)，所以原本清晰的變界線轉為 JPG 之後會變模糊，網頁上常見的文字或邊界線花掉的圖都是不當使用 JPG 的後果，如下圖。



相對的，GIF 會完全忠實於各點的位置，但是會將顏色縮減到最多 256 色，碰到有很多區塊同色時它可以壓縮得很小，顏色漸層或很複雜時就壓不下去了！總之，如果你的影像是真實照片，用 JPG 是對的，顏色漂亮『看起來』不失真，壓縮得也很小！反之，如果像本書中很多自螢幕擷取的電腦操作畫面用 JPG 就是最笨的選擇了！試試看，這些圖用 GIF 存會非常清晰也非常小！256 色太少嗎？對於人工設計的圖案或以文字為主的圖片，絕對夠用了！當然最恐怖的錯誤是用 GIF 存照片，你會發現顏色變得比較難看(不平滑)，檔案還大得不得了！

有關影像的話題談不完，只希望本單元不僅教你學會如何用程式操作數位影像的基本動作，也因此對於數位影像的結構有進一步的認識，可以將影像用得更好更有效率。