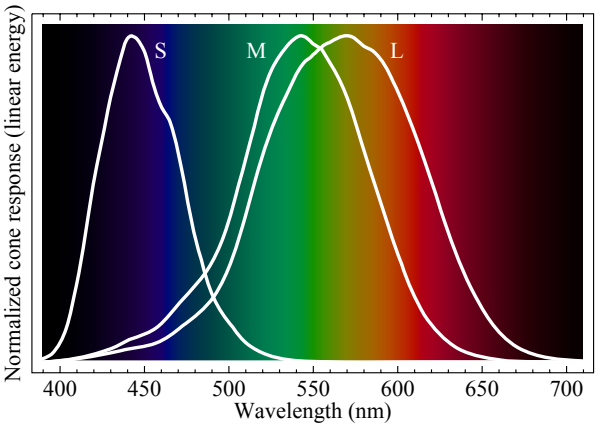
Bitmap 作業 (I)

資訊之芽語法班 2015 suhorng

像素、顏色、圖片

- 顯示器只發出三種顏色的光: 紅, 綠, 藍
- 點陣圖片的原始資料是 R, G, B 二維陣列
- 三種感光細胞 (圖片作者 BenRG, 可見 http://en.wikipedia.org/wiki/Color_vision)



點陣圖 (Bitmap)

- 直接把 R, G, B 的量存下來, 範圍從 0 到 255
- **不保證每種裝置顯示出來的圖片色調完全一樣**
- 我們討論的是特例: Windows 的一種圖片儲存格式 `.bmp`
- 完整檔案格式:

  http://en.wikipedia.org/wiki/BMP_file_format#File_structure

  - `BITMAPFILEHEADER`
  - `BITMAPINFOHEADER`

引入 Bitmap 處理程式

- 輸入檔: `sprout03_in.bmp` (跟 `.exe` 放在一起)
- 輸出檔: `sprout03_out.bmp`
- 程式格式
  - 引入 `bmp_hdlr.h` (跟 `.cpp` 放在一起)
  - 宣告三個**全域**二維 `int` 陣列, 如下

```
#include <iostream>
#include "bmp_hdlr.h"

int canvas_r[bmp_size][bmp_size];
int canvas_g[bmp_size][bmp_size];
int canvas_b[bmp_size][bmp_size];

int main() {
  // 在這裡處理圖片

  // 對 canvas_r, canvas_g, canvas_b 的修改,
  // 在程式結束後會自動存導
  return 0;
}
```

開始畫圖

- 設定圓心: $(80, 80)$; 半徑: 50
- 把圓中的所有點都塗成黃色 `#FFFF00`
  - 對每個點, 判斷它在不在圓中

$$(x-80)^2 + (y-80)^2 \leq 50^2 ?$$

```
for (int i = 0; i < bmp_size; ++i) {
  for (int j = 0; j < bmp_size; ++j) {
    if ((i-80)*(i-80) + (j-80)*(j-80) <= 50*50) {
      canvas_r[i][j] = 255;
      canvas_g[i][j] = 255;
      canvas_b[i][j] = 0;
    }
  }
}
```

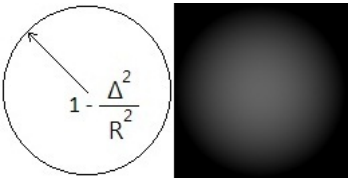
範例 - 黑白圖片

- 圖片轉成黑白
  - 直接平均 (***✗***)
  - 依照人眼對 R, G, B 光的感受程度加權 (✓)
- 名詞: Y-Cb-Cr 色彩空間
  - 依據一般圖片的特色, 可以壓縮!

```
int y = (299*canvas_r[i][j] + 587*canvas_g[i][j] + 114*canvas_b[i][j])/1000;

canvas_r[i][j] = y;
canvas_g[i][j] = y;
canvas_b[i][j] = y;
```

範例 - 氣泡特效

- 顏色可以**疊加**
- 選定圓心與半徑, 疊加的數字從圓心往外逐漸降低




```
int radius2 = radius*radius;

for (int j = -radius; j <= +radius; ++j) {
  for (int k = -radius; k <= +radius; ++k) {
    int ty = center_y + j, tx = center_x + k, dist2 = j*j + k*k;
    if (dist2<radius2 && 0<=ty && ty<bmp_size && 0<=tx && tx<bmp_size) {
      canvas_r[ty][tx] += color_r*(radius2 - dist2)/radius2;
      canvas_g[ty][tx] += color_g*(radius2 - dist2)/radius2;
      canvas_b[ty][tx] += color_b*(radius2 - dist2)/radius2;
    }
  }
}
```

範例 - 失焦

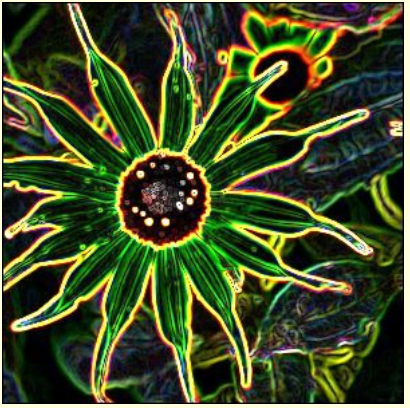
- 新的 pixel 值是周圍 pixel 值的平均
    - 需要新開陣列, 儲存新的 pixel 值

```
int cnt = 0; // 計算有幾個 pixel

for (int dx = -diff; dx <= diff; ++dx) {
  for (int dy = -diff; dy <= diff; ++dy) {
    int y_src = y + dx, x_src = x + dy;

    if (0 <= y_src && y_src < bmp_size && 0 <= x_src && x_src < bmp_size) {
      buf_r[y][x] += canvas_r[y_src][x_src];
      buf_g[y][x] += canvas_g[y_src][x_src];
      buf_b[y][x] += canvas_b[y_src][x_src];
      cnt = cnt + 1;
    }

  }
}
buf_r[y][x] /= cnt;
buf_g[y][x] /= cnt;
buf_b[y][x] /= cnt;
```

- 失焦程度也可以變化, 例如距離圓心 (x, y) 越遠越模糊

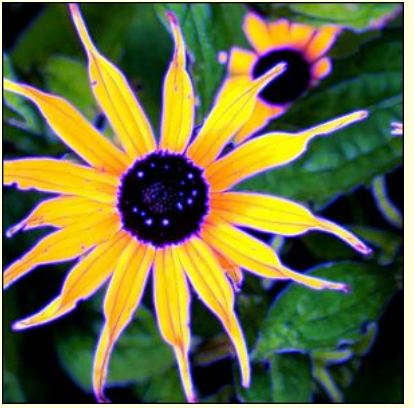
作業 - 邊緣偵測 (Sobel Filter)

- 直接把 Sobel filter 套用到圖片上

	before	after
範例圖片		

作業 - 邊緣偵測 (Sobel Filter)

- 直接把 Sobel filter 套用到圖片上

	before	a variation
範例圖片		

作業 - 邊緣偵測 (Sobel Filter)

- 直接把 Sobel filter 套用到圖片上
- 對於位於 (x, y) 的像素 $p_{x,y}$, 其
  - x 方向的 Sobel filter 運算結果, G_x, 為

    $$(p_{x+1,y+1} - p_{x-1,y+1}) + 2 \times (p_{x+1,y} - p_{x-1,y}) + (p_{x+1,y-1} - p_{x-1,y-1})$$

  - y 方向的 Sobel filter 運算結果, G_y, 為

    $$(p_{x+1,y+1} - p_{x+1,y-1}) + 2 \times (p_{x,y+1} - p_{x,y-1}) + (p_{x-1,y+1} - p_{x-1,y-1})$$

  - 總 Sobel filter 的結果為 $\sqrt{{G_x}^2 + {G_y}^2}$
- 為了簡單, (1) 對 R,G,B 分別取 Sobel filter (2) 忽略圖片邊緣的一圈 pixels

作業 - 邊緣偵測 (Sobel Filter)

- x 方向的 Sobel filter

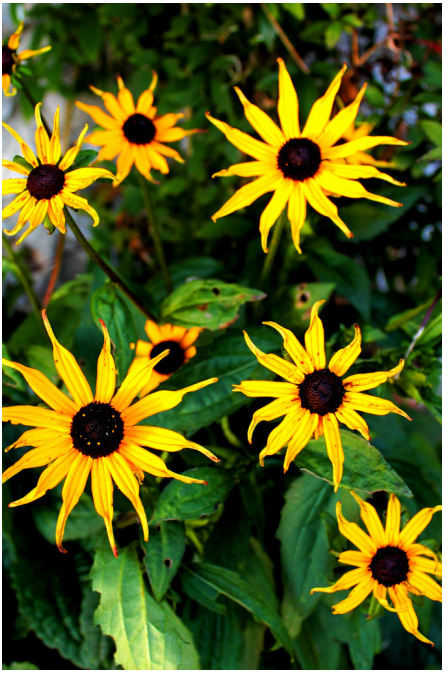
-1		$+1$
-2	**P**	$+2$
-1		$+1$

- y 方向的 Sobel filter

-1	-2	-1
	P	
$+1$	$+2$	$+1$

- 相鄰像素相減 $\Longrightarrow$ 顏色變化
- 上下相鄰(G_x)或左右相鄰(G_y)的點也考慮, 但加權較低

圖片來源



- By **Les Haines**,

  14 Sep 2013.

- Released under CC license 2.0

https://www.flickr.com/photos/leshaines123/9817029973/