四資工二甲　B11132021　蔣博元

電腦圖學 Project1

目次

TOP

API 文件

- ## Palette.h

Palette.h 內的 class 和函數都宣告在 namespace Color 中。

`struct RGB_t`

用來儲存 RGB 的物件，在建構時可以分別指定 RGB 的值，或直接從 uint8_t 的陣列取出 RGB。

`class Palette_t`

繼承自 std::vector<Color::RGB_t>，用來儲存可用的顏色。

method:

`find_closest_to`	在 Palette 中搜尋歐氏距離最近的顏色。

`int euclidean_distance_square (RGB_t c1, RGB_t c2)`

計算兩顏色的歐氏距離的平方。

`void memset (uint8_t* arr, RGB_t color)`

將 color 的 RGB 依序寫入 arr[0]、arr[1]、arr[2]。

`constexpr RGB_t White(255, 255, 255)`

代表白色的 RGB

`constexpr RGB_t Black`

代表黑色的 RGB

TOP

- ## Filter.h

Filter.h 內的 class 和函數都宣告在 namespace Filter 內。

`struct ImageInfo_t`

儲存圖片的資訊，這是給 Filter 計算時看的。

`class Filter_t`

用來表示濾波器的物件。

method:

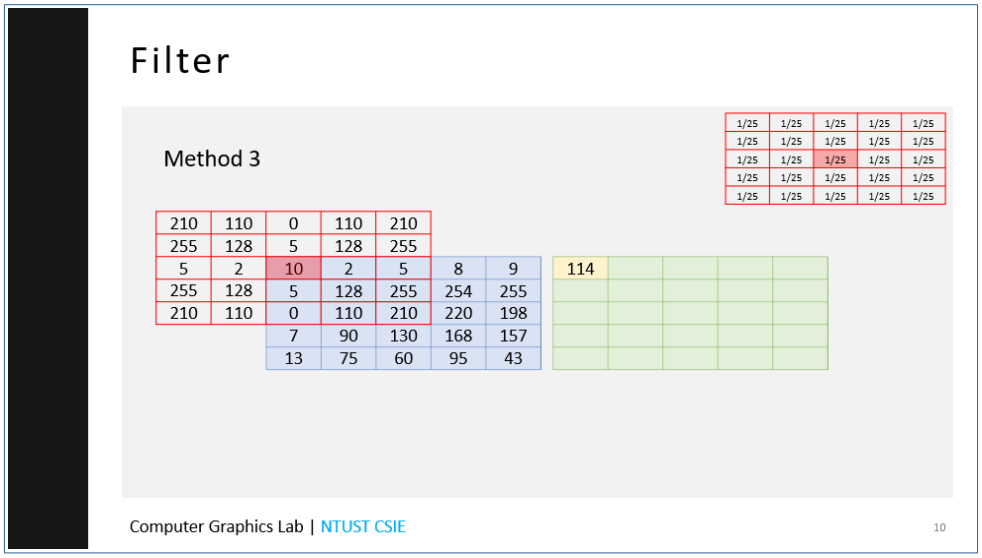
`calculate`	將濾波器套用在圖上的(r1, c1)和(r2, c2)所圍的區域，其中(r1, c1)和(r2, c2)分別代表該區域的左上角和右下角
`at`	存取濾波器內的值，若物件為 const 則回傳 float，否則回傳 float&。
`getRow`	濾波器有幾列。
`getCol`	濾波器有幾欄。

TOP

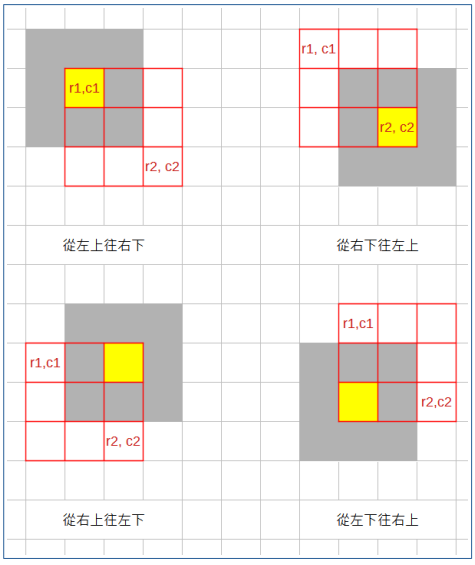
演算法

- ## Filter—像素的鏡像

在使用 Filter 時我採用鏡像的方式來補足超出圖像邊界的像素，正如實習課的投影片所示。



我的作法是先從 Filter 作用的範圍選擇一個角作為「初始位置」——那個角必須在圖內，並從初始位置開始將像素和 Filter 內的值一一相乘並相加。當碰到在圖外的像素時，再去算它是從圖內的哪個像素鏡像而來的。



（右圖）呈現了四種情況下初始位置的選法。灰色部分為圖片的一角，紅色邊框為 Filter 作用的範圍，黃色為初始位置，並且假設 Filter 作用在（r1 列, c1 欄）~（r2 列, c2 欄）。

當然，可能的情況不只右邊四種，也可能會出現兩個角甚至四個角都在圖內的情況。不過只要選定的初始位置有在圖片內，就沒問題了。

TOP

四資工二甲　B11132021　蔣博元

我是採用「先水平，後垂直」的方式來進行迭代。（下圖 1）

在迭代時，若沒有超出原圖的範圍，則直接拿圖中的相素和 Filter 的值相乘；否則，在遇到原圖中沒有的像素時，要做水平鏡像去取像素。（下圖 2）

當迭代到整列都超出圖外的地方時，鏡像的方向要從水平改垂直。有時做了垂直鏡像後還是沒有對應的像素，這時就要在做一次水平鏡像。（下圖 3）

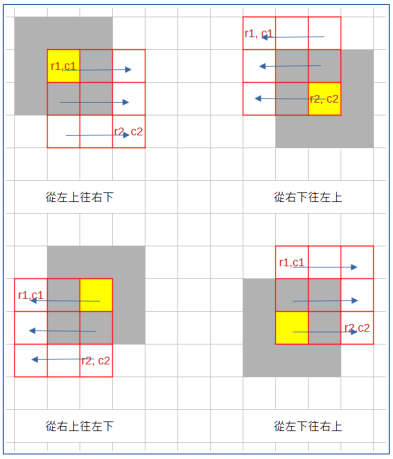


圖 1

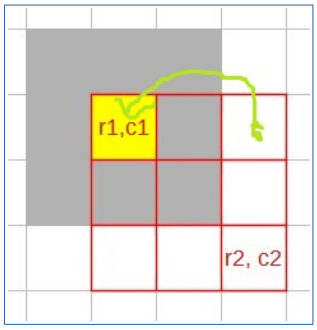


圖 2

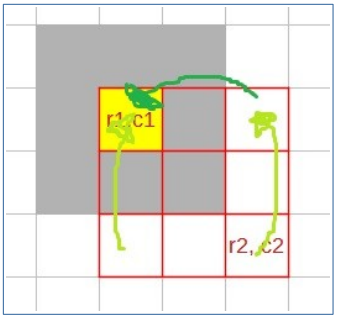


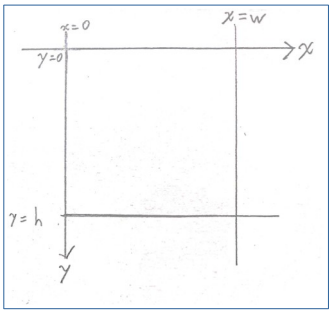
圖 3

上述演算法實作在 Filter_t::do_the_calculate 中，其中（rs, cs）為初始位置，（re, ce）為初始位置的對角，image 為圖片。

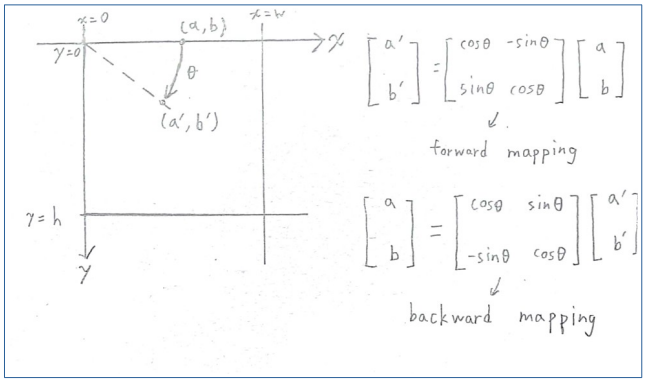
TOP

- **圖片旋轉**

定圖片的左上角為座標（0，0），橫向為 x 軸，縱向為 y 軸，並假設圖片高 h 寬 w。

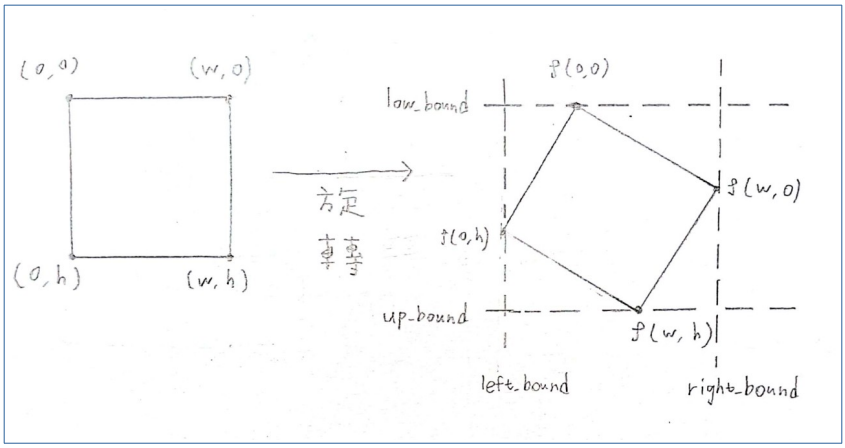


定義 mapping。



確認新圖片的範圍。將原圖片的四個角（0，0）、（w，0）、（w，h）、（0，h）做旋轉，旋轉後的四個點中：x 的最小值為 left_bound、x 的最大值為 right_bound、y 的最小值為 low_bound、y 的最大值為 up_bound，有了這四個邊界就能決定新圖片的寬度、長度。

low_bound 畫在圖中的上方是因為我定義下方為 y 軸正向，所以越上方 y 越小。



對於新圖中第 r 列 c 欄的像素，它左上角的 x y 座標為（left_bound+c, low_bound+r），做 backward mapping 後就能找到它在舊圖中對應的位置。

TOP