前言

OpenCV 是開發電腦視覺（Computer Vision）應用程式時最受歡迎的函式庫，能夠即時執行許多不同的電腦影像演算法，問世多年之後成為電腦視覺領域的標準函式庫，其最大的優點在於高度最佳化，幾乎能夠支援所有的平台。

本書先簡單介紹電腦視覺的各個領域，透過 C++ 語言示範相關的 OpenCV 功能，每章提供真實世界的範例以及相關使用案例的程式碼，讓讀者能夠快速進入相關主題，了解在真實世界中的使用方式。總的來說，這是本針對在 C++ 使用 OpenCV 建立各種應用的實務手冊。

目標讀者

本書針對剛接觸 OpenCV，想要在 C++ 使用 OpenCV 開發電腦視覺（Computer Vision）應用程式的開發人員，基本的 C++ 知識有助於理解本書內容；本書也有助於想要進入電腦視覺領域了解其中概念的讀者。為了從本書獲得最大的收穫，讀者應該要知道基本的數學概念，包含向量、矩陣以及矩陣乘法，讀完本書，讀者就能夠使用 OpenCV 從無到有建構出各種電腦視覺應用程式。

本書主題

第 1 章，*OpenCV 入門*，涵蓋在各種作業系統上安裝 OpenCV 的步驟，介紹人類的視覺系統以及電腦視覺領域的各個主題。

第 2 章，*OpenCV 基礎介紹*，討論如何使用 OpenCV 讀寫圖像與影片，同時說明用 CMake 建置專案的方式。

第 3 章，圖形使用者介面與基本濾鏡，內容包含利用圖形使用者介面與偵測滑鼠事件，建立互動式應用程式的方法。

第 4 章，深入色階直方圖與濾鏡，探討色階直方圖（histograms）與濾鏡，同時示範卡通化圖片的方法。

第 5 章，自動光學檢查、物體分割與偵測，介紹多種影像處理的前置技巧，包括雜訊去除（noise removal）、定限（thresholding）以及輪廓分析（contour analysis）。

第 6 章，學習物體分類，處理物體辨認與機器學習，以及使用支援向量機（Support Vector Machines）建立物體分類系統的方法。

第 7 章，偵測人臉部份與覆蓋遮罩，討論人臉偵測與 Haar 串接（Haar Cascades），接著說明利用這些方法偵測人臉各個部份的作法。

第 8 章，影像監控、背景塑模與形態學運算，探討影像去背（也稱為背景相減法，background subtraction）、影像監控以及形態學影像處理，並說明這些技術之間的關係。

第 9 章，學習物體追蹤，內容涵蓋各種在即時影像追蹤物體的技術，包含以色彩為基礎（color-based）以及以特徵為基礎（feature-based）的追蹤技巧。

第 10 章，為文字辨識開發分割演算法，內容包含光學字元識別（optical character recognition，OCR）、正文切割（text segmentation）以及 Tesseract OCR 引擎的簡介。

第 11 章，用 *Tesseract* 辨識文字，深入 Tesseract OCR 引擎，解釋使用 Tesseract OCR 引擎用於正文偵測、萃取與辨識的方法。

第 12 章，深度學習與 *OpenCV*，探討如何在 OpenCV 使用最常見的兩種深度學習架構：使用 YOLO v3 做物體偵測，以及用 Single Shot Detector 作人臉偵測。

閱讀指南

基本的 C++ 知識有助於理解本書，書中範例使用了下列技術：OpenCV 4.0；CMake 3.3.x 以上的版本；Tesseract；Leptonica（Tesseract 需要使用）；Qt（選用）；以及 OpenGL（選用）。

詳細安裝說明請參考相關章節。

2
OpenCV 基礎介紹

第 1 章「*OpenCV* 入門」介紹了在主流作業系統安裝 OpenCV 的方式，接著要介紹的是 OpenCV 開發的基礎。首先會說明如何透過 CMake 建立專案，本章會介紹影像的基本資料結構、矩陣以及專案需要的其他結構；另外還會介紹透過 OpenCV 存續函式，將變數與資料儲存為 XML ／ YAML 檔案的方法。

本章內容包括：

- 使用 CMake 設定專案
- 從磁碟寫入／讀取影像
- 讀取影片與存取相機設備
- 主要影像結構（如矩陣）
- 其他重要的結構（向量、常量等等）
- 矩陣運算基本介紹
- 使用 OpenCV 的 XML ／ YAML 存取 API 儲存檔案

技術要求

讀者需要熟悉基本的 C++ 程式語言，本章的所有程式碼都可以從以下的 GitHub 網址 取 得：`https://github.com/PacktPublishing/Learn-OpenCV-4-By-Building-Projects-Second-Edition/tree/master/Chapter_02`。程式碼應該可以在所有的作業系統執行，但筆者只有在 Ubuntu 上測試。

讀者可以在以下網址的影片中看到程式實際執行的效果：
`http://bit.ly/2QxhNBa`

基本 CMake 設定檔

本書使用 CMake 設定與檢查專案需要的所有相依性，不是非用 CMake 不可，也可以用 **Makefiles** 或 **Visual Studio** 等工具與 IDE 設定專案，但 CMake 是設定跨平台 **C++** 專案可攜性較高的作法。

CMake 的設定檔是 CMakeLists.txt，其中定義了編譯與相依的程序，對於最基本從單一個原始檔產生執行檔的專案而言，CMakeLists.txt 只需要以下的三行內容：

```
cmake_minimum_required (VERSION 3.0)
project (CMakeTest)
add_executable(${PROJECT_NAME} main.cpp)
```

第一行定義了 CMake 的最低版本要求，CMakeLists.txt 一定要有這行資訊，才能夠在後續的內容裡使用特定版本提供的 cmake 功能；上述範例中要求最少要是 CMake 3.0，第二行定義專案名稱，專案名稱會同時儲存為 PROJECT_NAME 變數。

最後一行用 main.cpp 檔案建立一個可執行的命令（add_executable()），將執行檔名稱設定為專案名稱（${PROJECT_NAME}），也就是依據設定的專案名稱，將原始碼編譯為檔名 **CMakeTest** 的可執行檔。${} 表示式能夠存取環境中定義的所有變數，上面的例子使用 ${PROJECT_NAME} 變數做為可執行檔的輸出名稱。

建立函式庫

CMake 能讓開發人員建立供 OpenCV 建置系統使用的函式庫，將應用程式共同的部份抽取為函式庫是在軟體開發上很常見的作法，特別適用於大型應用程式或多個應用程式有共同部份的情況。建立函式庫的時候不需要建立可執行檔，而是建立包含所有函式、類別等編譯後成品的檔案。接著就可以與其他應用程式分享函式庫檔案，而不是分享原始程式碼。

CMake 包含了為此使用的 add_library 函式：

```
# 建立第一個函式庫
    add_library(Hello hello.cpp hello.h

# 建立使用新函式庫的應用程式
```

```
    add_executable(executable main.cpp)

# 連結函式庫建立可執行檔
    target_link_libraries(executable Hello)
```

CMake 會忽略以 # 開頭的該行內容，一般用於註解。add_library(Hello hello.cpp hello.h) 命令定義了函式庫的名稱與原始檔檔名，Hello 是函式庫名稱，hello.cpp 與 hello.h 則是原始檔，加入標頭檔是為了讓 Visual Studio 等 IDE 能夠正確連結到標頭檔。add_library 命令會產生共享函式庫（對 OS X 與 Unix 系統而言是 .so 檔案，在 Windows 系統則是 .dll 檔案）或靜態函式庫（OS X 與 Unix 系統上的 .a 以及 Windows 系統上的 .lib），取決於函式庫名稱與原始檔間是否加上 SHARED 或 STATIC 關鍵字而定。target_link_libraries(executable Hello) 命令會為執行檔連結所需使用的函式庫，以這個範例而言就是 Hello 函式庫。

維護相依性

CMake 能夠搜尋相依性與外部函式庫，能夠在專案中加入對外部元件的相依性與設定額外的需求，建立使用外部元件的複雜專案。

本書中最重要的相依性當然是 OpenCV，所有的專案都會加入這個相依性：

```
    cmake_minimum_required (VERSION 3.0)
    PROJECT(Chapter2)
# 需要 OpenCV
    FIND_PACKAGE( OpenCV 4.0.0 REQUIRED )
# 顯示偵測到的 opencv 版本
    MESSAGE("OpenCV version : ${OpenCV_VERSION}")
# 加入 include 目錄／標頭檔路徑
    include_directories(${OpenCV_INCLUDE_DIRS})
# 加入編譯好的函式庫／物件路徑
    link_directories(${OpenCV_LIB_DIR})
#建立 SRC 變數
    SET(SRC main.cpp)
# 建立可以執行檔
    ADD_EXECUTABLE(${PROJECT_NAME}) ${SRC})
# 連結函式庫
    TARGET_LINK_LIBRARIES(${PROJECT_NAME} ${OpenCV_LIBS})
```

接下來詳細的說明命令稿的內容：

```
cmake_minimum_required (VERSION 3.0)
cmake_policy(SET CMP0012 NEW)
PROJECT(Chapter2)
```

第一行定義了 CMake 版本的最低要求，第二行則告訴 CMake 啟用新版 CMake 行為，讓 CMake 能夠正確的辨別數字與布林常數，不需要額外對變數名稱解參考（dereference）；這是 CMake 2.8.0 引進的新原則（policy），從 3.0.2 版後，CMake 會在未設定原則時發出警告訊息。最後一行定義了專案的名稱，定義了專案名稱之後，接著需要定義需求、函式庫與相依性：

```
# 需要 OpenCV
    FIND_PACKAGE( OpenCV 4.0.0 REQUIRED )
# 顯示偵測到的 opencv 版本
    MESSAGE("OpenCV version : ${OpenCV_VERSION}")
    include_directories(${OpenCV_INCLUDE_DIRS})
    link_directories(${OpenCV_LIB_DIR})
```

這段命令稿是為了搜尋 OpenCV 的相依性，FIND_PACKAGE 能夠尋找相依性，並指定相依性的最低版本以及是否為必要相依性。範例中尋找 OpenCV 4.0.0 以上的版本，並設定為必要的套件。

這個 FIND_PACKAGE 命令包含了 OpenCV 所有的模組，讀者可以只指定想要引進的模組，縮小專案的體積並提高執行速度。例如，要是只會用到 OpenCV 的基本型別與核心函式，就可以使用以下命令：FIND_PACKAGE(OpenCV 4.0.0 REQUIRED core)。

要是 CMake 找不到必要的函式庫，會傳回錯誤並停止編譯應用程式。MESSAGE 函式會在終端機（命令列視窗）或 CMake GUI 顯示訊息，範例中顯示了 OpenCV 的版本：

```
OpenCV version : 4.0.0
```

${OpenCV_VERSION} 是 CMake 儲存 OpenCV 套件版號所使用的變數名稱。include_directoreis() 與 link_directories() 會將特定函式庫的標頭檔與路徑加入專案編譯環境，CMake 模組會將 OpenCV 的相關資料儲存在 ${OpenCV_INCLUDE_DIRS} 與 ${OpenCV_LIB_DIR} 變數。並不是所有的平台都需要這兩行設定，例如在 Linux 之類的平台，這些路徑通常會儲存在環境變數，但對於系統中安裝了多個 OpenCV 版本的情況，建議加上這兩行設定，選擇正確的連結與引用的路徑，接著就可以引入專案的原始檔了：

```
#建立 SRC 變數
    SET(SRC main.cpp)
# 建立可以執行檔
    ADD_EXECUTABLE(${PROJECT_NAME}) ${SRC})
# 連結函式庫
    TARGET_LINK_LIBRARIES(${PROJECT_NAME} ${OpenCV_LIBS})
```

最後幾行是建立執行檔與連結 OpenCV 函式庫，這部份與前一節「**建立函式庫**」的作法相同。這段程式碼中使用了一個新的函式 SET，這個函式會建立新變數並設定為指定的數值，範例中建立了 SRC 變數，將數值設為 main.cpp；也可以對相同變數加入多個數值，例如：

```
SET(SRC main.cpp
        util.cpp
        color.cpp
)
```

更複雜的命令稿

本節要介紹更複雜的命令稿，包含子目錄、函式庫與執行檔，從以下的命令稿可以看到，這一切只需要兩個檔案與幾行指令就能夠完成。並沒有硬性規定一定要建立多個 CMakeLists.txt 檔案，所有的設定可以都放在同一個 CMakeLists.txt 裡。但比較常見的做法是在每個專案子目錄下都使用各自的 CMakeLists.txt，這種作法比較有彈性，可攜性也較高。

本節範例使用的檔案結構如下，有一個 utils 函式庫的目錄，包含主執行檔在內的其他程式碼則放在 root 目錄：

```
CMakeLists.txt
main.cpp
utils/
   CMakeLists.txt
   computeTime.cpp
```

```
    computeTime.h
    logger.cpp
    logger.h
    plotting.cpp
    plotting.h
```

接著需要定義這兩個 CMakeLists.txt 檔案的內容：一個位於 root 目錄，另一個則位於 utils 目錄，根目錄的 CMakeLists.txt 檔案的內容如下：

```
    cmake_minimum_required (VERSION 3.0)
    project (Chapter2)

# 需要 OpenCV 套件
    FIND_PACKAGE( OpenCV 4.0.0 REQUIRED )

# 加入 opencv 標頭檔到專案
    include_directories(${OpenCV_INCLUDE_DIR})
    link_directories(${OpenCV_LIB_DIR})

# 將子目錄加入建置
    add_subdirectory(utils)

# 加入額外的 log 訊息以及編譯前的定義
    option(WITH_LOG "Build with output logs and images in tmp" OFF)
    if(WITH_LOG)
      add_definitions(-DLOG)
    endif(WITH_LOG)

# 產生新的執行檔
    add_executable(${PROJECT_NAME} main.cpp)
# 連結專案與相依性
    target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS} Utils)
```

幾乎所有的命令都已經在前一節介紹過了，接下來介紹新出現的函式，add_subdirectory() 讓 CMake 知道要分析指定子目錄下的 CMakeLists.txt。在繼續說明主要 CMakeLists.txt 內容之前，先看看 utils 目錄下的 CMakeLists.txt。

在 utils 目錄下的 CMakeLists.txt 檔會產生新的函式庫，並將產生的函式庫放到主專案目錄：

```
# 加入 utils 函式庫原始檔的變數
    SET(UTILS_LIB_SRC
       computeTime.cpp
       logger.cpp
       plotting.cpp
)
```

```
# 建立新的 utils 函式庫
    add_library(Utils ${UTILS_LIB_SRC})
# 確保編譯器到正確的位置尋找函式庫的引入檔
    target_include_directories(Utils PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})
```

這個 CMake 命令稿定義了 UTILS_LIB_SRC 變數，將變數指定為函式庫中所有的檔案名稱，接著透過 add_library 產生函式庫，並使用 target_include_directories 函式讓主專案能夠偵測得到所有的標頭檔。離開 utils 子目錄再次回到根目錄的 CMake 命令稿，option 函式建立一個新的變數並提供一小段說明訊息，也就是範例中的 WITH_LOG。這個變數可以透過 ccmake 命令列或 CMake GUI 介面改變數值，在圖形介面上，會出現描述訊息與勾選框供使用者啟用／停用這個設定。這個函式非常有用，能讓使用者決定編譯時期使用的功能，例如是否啟用 log 功能，或是像 OpenCV 一樣決定是否要編譯 Java 或 Python 的支援等等。

本節的範例使用了這個選項啟用應用程式的 logger，在程式碼透過前置處理器定義啟用 logger：

```
#ifdef LOG
    logi("Number of iteration %d", i);
#endif
```

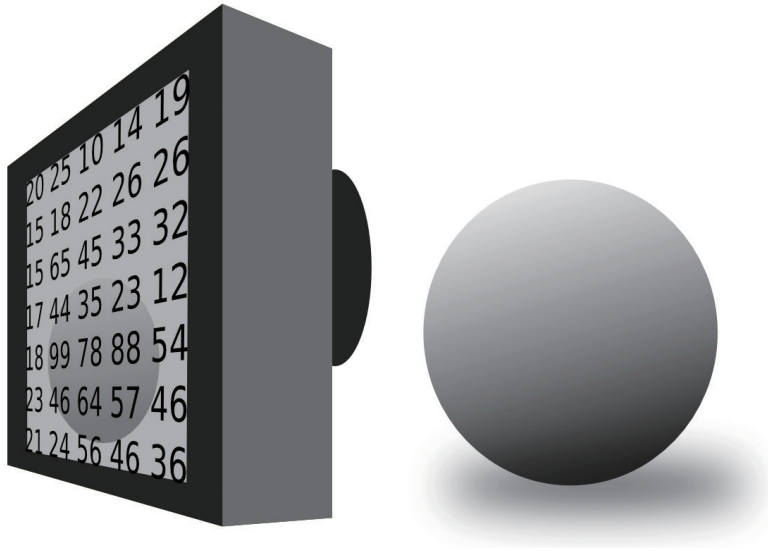
LOG 巨集是在 CMakeLists.txt 中呼叫 add_definitions 定義（-DLOG），而這個函式則又透過 WITH_LOG 變數決定是否呼叫：

```
if (WITH_LOG)
    add_definitions(-DLOG)
endif(WITH_LOG)
```

接下來就可以為本書的電腦視覺專案，建立能夠在任何作業系統上編譯的 CMake 命令稿了。在著手建立範例專案之前，先讓我們再次回到 OpenCV 的基礎。

影像與矩陣

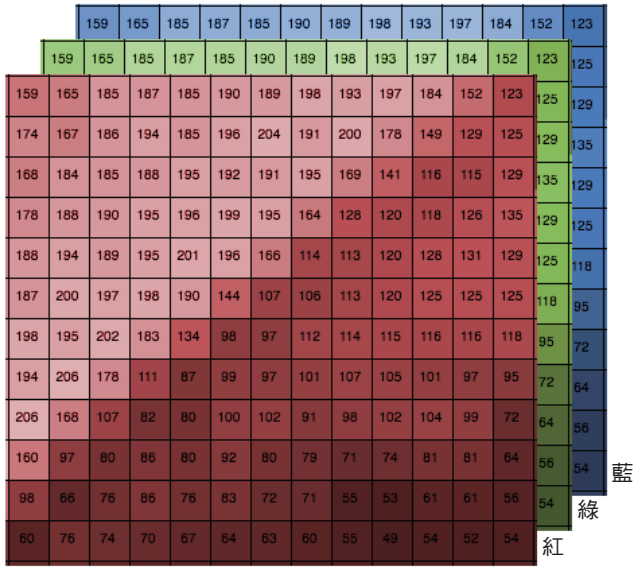
電腦視覺中最重要的結構除了影像之外不作第二人想，電腦視覺領域的影像是指透過數位設備從真實世界捕捉到的資訊。如下圖所示，照片是以矩陣格式儲存的一長串數字序列，每個數字代表了特定波長（如彩色圖片中的紅、綠、黃）的光線強度，圖片中的每個點稱為「**像素**」（**pixel**，影像元素），每個像素依據照片的不同儲存了一個以上的數值，對於只有灰、黑或白的圖片（稱為二值化影像）只需要儲存一個數值（如 0 或 1），灰階圖片也只需要一個數值，彩色圖片則需要儲存三個數值。通常數值是在 0 到 255 之間的整數，但也可以使用其他的範圍，例如，**HDRI（High Dynamic Range Imaging）**或熱成像圖會使用 0 到 1 之間的浮點數。



影像是以矩陣的格式儲存，每個像素都有其位置，可以透過行、列的數值定位。OpenCV 定義了 Mat 類別作為這個用途，對於灰階影像只會使用一個矩陣，如下圖：

159	165	185	187	185	190	189	198	193	197	184	152	123
174	167	186	194	185	196	204	191	200	178	149	129	125
168	184	185	188	195	192	191	195	169	141	116	115	129
178	188	190	195	196	199	195	164	128	120	118	126	135
188	194	189	195	201	196	166	114	113	120	128	131	129
187	200	197	198	190	144	107	106	113	120	125	125	125
198	195	202	183	134	98	97	112	114	115	116	116	118
194	206	178	111	87	99	97	101	107	105	101	97	95
206	168	107	82	80	100	102	91	98	102	104	99	72
160	97	80	86	80	92	80	79	71	74	81	81	64
98	66	76	86	76	83	72	71	55	53	61	61	56
60	76	74	70	67	64	63	60	55	49	54	52	54

如果是彩色圖片，如下圖所示，會使用寬 × 長 × 顏色通道數的矩陣：



Mat 類別不只可以儲存影像，也可以儲存其他不同類型、任意大小的矩陣，執行矩陣代數等運算，下一節會介紹最重要的矩陣運算，包含矩陣加法、乘法以及對角化矩陣等等。在這之前，讀者必須要先知道矩陣在電腦記憶體中的儲存方式，因為存取記憶體位置永遠比透過 OpenCV 函式存取個別像素還要更有效率。

矩陣在記憶體裡依據行（column）或列（row）的順序儲存為陣列或連續的數值，下表是 **BGR** 影像格式一連串像素的數值：

Row 0									Row 1									Row 2								
Col 0			Col 1			Col 2			Col 0			Col 1			Col 2			Col 0			Col 1			Col 2		
Pixel 1			Pixel 2			Pixel 3			Pixel 4			Pixel 5			Pixel 6			Pixel 7			Pixel 8			Pixel 9		
B	G	R	B	G	R	B	G	R	B	G	R	B	G	R	B	G	R	B	G	R	B	G	R	B	G	R

依照這個順序，就可以透過以下的公式存取任何的像素：

```
數值 = Row_i*num_cols*num_channels + Col_i + channel_i
```

雖然 OpenCV 函式已經對隨機存取作了高度最佳化，但有時（透過指標運算）直接存取記憶體會更有效率，例如在迴圈中存取每個像素之類的情況。

讀取／寫入影像

緊接著矩陣的介紹之後，接著要寫個基本的 OpenCV 程式，首先要學的就是影像的讀取與寫入：

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

// 引入 OpenCV
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;

int main( int argc, const char** argv )
{
    // 讀取影像
    Mat color= imread("../lena.jpg");
    Mat gray= imread("../lena.jpg", IMREAD_GRAYSCALE);

    if(! color.data ) //檢查輸入是否可用
    {
    cout << "Could not open or find the image" << std::endl ;
    return -1;
    }

    // 寫出影像
    imwrite("lenaGray.jpg", gray);
    // 用 opencv 函式取得同一像素
    int myRow=color.cols-1;
    int myCol=color.rows-1;
    auto pixel= color.at<Vec3b>(myRow, myCol);
    cout << "Pixel value (B,G,R): (" << (int)pixel[0] << "," <<
    (int)pixel[1] << "," << (int)pixel[2] << ")" << endl;
    // 顯示影像
    imshow("Lena BGR", color);
    imshow("Lena Gray", gray);
    // 等待按下任何按鍵
    waitKey(0);
    return 0;
}
```

接下來試著了解程式碼：

```
// 引入 OpenCV
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;
```

首先必須引進範例程式使用的函式宣告，這些函式都包含在 core 模組（基本影像資料處理）以及 highgui（OpenCV 提供的跨平台 I/O 函式是 core 與 highgui 模組，前者包含了矩陣等基本類別，後者則包含讀取、寫入與透過圖形介面顯示影像等函式）。接著是讀取影像：

```
// 讀取影像
Mat color= imread("../lena.jpg");
Mat gray= imread("../lena.jpg", IMREAD_GRAYSCALE);
```

imread 是讀取影像的重要函式，會讀取影像檔儲存為矩陣格式。imread 函式有兩個參數，第一個參數是包含影像路徑的字串，第二個是選用參數，預設是讀取彩色影像。第二個參數接受以下數值：

- cv::IMREAD_UNCHANGED：使用這個常數時會傳回輸入檔色彩濃度（depth）的 16 位元／ 32 位元影像，否則 imread 函式會將影像轉換為 8 位元影像。
- cv::IMREAD_COLOR：設為這個常數時會強制將影像轉換為彩色（BGR，無號八位元格式）。
- cv::IMREAD_GRAYSCALE：設為這個常數則會轉換為灰階影像（無號八位元格式）。

儲存影像則是使用 imwrite 函式，會將記憶體的矩陣影像儲存為檔案：

```
// 寫出影像
imwrite("lenaGray.jpg", gray);
```

第一個參數是要儲存影像的檔案路徑以及影像格式的副檔名，第二個參數是要儲存的影像；範例程式建立並儲存影像的灰階版本，接著將載入並儲存在 gray 變數的灰階影像寫入到 .jpg 檔案：

```
// 用 opencv 函式取得同一像素
int myRow=color.rows-1;
int myCol=color.cols-1;
```

利用矩陣的 .cols 與 .rows 屬性能夠取得影像的行數與列數，也就是影像的寬與高：

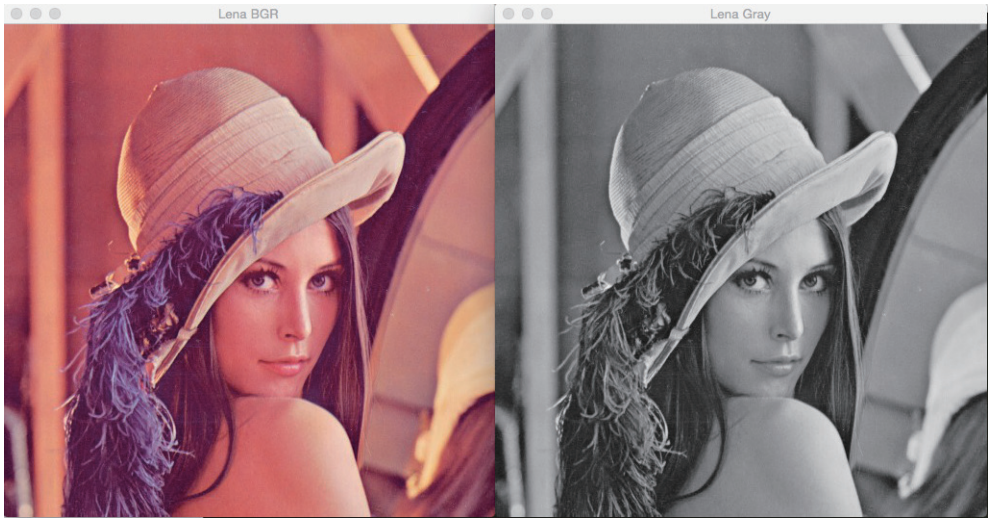
```
Vec3b pixel= color.at<Vec3b>(myRow, myCol);
cout << "Pixel value (B,G,R): (" << (int)pixel[0] << "," << (int)pixel[1]
<< "," << (int)pixel[2] << ")" << endl;
```

要存取影像特定位置的像素，可以使用 OpenCV Mat 類別的 cv::Mat::at<typename t>(row,col) 樣板函式。樣板的參數是傳回的型別，8-bit 影像使用的 typename 是 Vec3b 類別，其中包含了三個 unsigned char 資料（Vec=vector，3= 成員個數，b= 1 byte）；如果是灰階影像，可以直接使用 unsgined char 或任何影像使用的數字格式，例如 uchar pixel= color.at<uchar>(myRow, myCol)。最後，透過 imshow 函式建立顯示影像的視窗，第一個參數是視窗標題，第二個參數則是影像矩陣：

```
// 顯示影像
imshow("Lena BGR", color);
imshow("Lena Gray", gray);
// 等待按下任何按鍵
waitKey(0);
```

如果想要暫停程式等待使用者按下按鍵，可以透過 OpenCV 提供的 waitKey 函式。函式的參數是想要等待的毫秒（milliseconds）數，設定為 0 就會一直等待下去，直到使用者按下按鍵。

程式碼執行後會顯示如下的圖片，左側是彩色影像，右側是灰階：



最後，作為後續範例的範例程式，還需要一個 CMakeLists.txt，讓讀者能夠自行編譯範例程式。

以下是 CMakeLists.txt 的內容：

```
cmake_minimum_required (VERSION 3.0)
cmake_policy(SET CMP0012 NEW)
PROJECT(project)


set (CMAKE_CXX_STANDARD 11)[譯註6]

# 需要 OpenCV
FIND_PACKAGE( OpenCV 4.0.0 REQUIRED )
MESSAGE("OpenCV version : ${OpenCV_VERSION}")

include_directories(${OpenCV_INCLUDE_DIRS})
link_directories(${OpenCV_LIB_DIR})

ADD_EXECUTABLE(sample main.cpp)
TARGET_LINK_LIBRARIES(sample ${OpenCV_LIBS})
```

譯註 6　在 macOS 上需要加上 set(CMAKE_CXX_STANDARD 11) 啓用 C++ 11 標準，否則編譯範例時會發生錯誤。

利用 CMakeLists.txt 編譯程式碼的步驟如下：

1. 建立 build 目錄。
2. 在 build 目錄下，執行 cmake 或在 Windows 開啟 CMake gui 應用程式，選擇原始碼目錄（source）與建置目錄（build），按下 **Configure** 與 **Generate** 鍵。
3. 要是使用 Linux 或 macOS，如先前一般產生 Makefile，就可以使用 make 命令編譯專案；要是使用的是 Windows，就必須用編譯器開啟前一個步驟產生的專案進行編譯。

最後，在編譯完應用程式之後，會在 build 目錄下產生名為 sample 的可執行檔。

讀取影像與相機

本節透過以下簡單的例子，介紹讀取影片檔與相機輸入。在說明讀取影片檔或相機輸入的程式之前，要先介紹一個協助處理命令列參數十分方便的類別，這個類別是在 OpenCV 3.0 版加入，名稱是 CommandLineParser：

```
// OpenCv 命令列剖析函式
// 命令列剖析器接受的命令
const char* keys =
{
  "{help h usage ? | | print this message}"
  "{@video | | Video file, if not defined try to use webcamera}"
};
```

使用命令列剖析器（CommandLineParser）的第一步是利用常數 char 向量定義需要或允許使用的參數，每個參數都必須符合以下的格式：

```
"{參數名稱 | 預設值 | 說明}"
```

參數名稱能夠以 @ 作為字首，表示參數是預設參數。一個應用程式可以包含多個參數名稱：

```
CommandLineParser parser(argc, argv, keys);
```

建構子需要 main 函式輸入的參數以及先前定義的常數：

```
// 如果要求顯示 help
if (parser.has("help"))
{
       parser.printMessage();
       return 0;
}
```

.has 類別方法會檢查參數是否存在，範例中檢查了使用者是否使用了 -help 或 ? 參數，如果使用了這個參數，就透過 printMessage 顯示所有參數的說明：

```
String videoFile= parser.get<String>(0);
```

程式可以透過 .get<typename>(parameterName) 函式取得所有輸入的參數：

```
// 檢查參數值是否能正確剖析為變數
if (!parser.check())
{
    parser.printErrors();
    return 0;
}
```

取得所有的必要參數之後，可以檢查是否能夠正確的剖析參數，如果有任何參數無法正確剖析，就顯示錯誤訊息，例如加入了字串而不是數字：

```
VideoCapture cap; // 開啟預設相機
if(videoFile != "")
   cap.open(videoFile);
else
   cap.open(0);
if(!cap.isOpened()) // 檢查是否成功
   return -1;
```

讀取影片與相機使用相同類別，VideoCapture 類別屬於 videoio 模組，而不是如以往的 OpenCV 版本屬於 highgui 模組。建立物件之後，先檢查命令列參數的 videoFile 參數是否包含完整的檔案名稱，如果沒有指定檔案名稱，就試著開啟網路相機（webcam），有指定檔名就開啟影片檔。開啟的方式是使用 open 函式，指定影片檔案名稱或想要開啟的相機索引作為參數，如果系統只有一部相機，可以使用 0 作為參數。

要判斷是否能夠讀取影片檔案或相機，可以使用 isOpened 函式：

```
namedWindow("Video",1);
for(;;)
{
    Mat frame;
    cap >> frame;  // 從相機取得新畫格
    if(frame)
       imshow("Video", frame);
    if(waitKey(30) >= 0) break;
}
// 釋放相機或影像擷取
cap.release();
```

最後，透過 namedWindow 函式建立視窗，並在無窮迴圈內使用 >> 運算子抓取個別畫格內容，要是能夠正確的取得畫格內容，就使用 imshow 函式顯示影像。以這個範例而言，並不希望就此停止應用程式，而是用 waitKey(30) 等待 30 毫秒，確認使用者是否按下任何按鍵，以示意停止應用程式。

使用相機時，必須依據相機的速度決定取得下個畫格前的等待時間，例如，假設相機的運作速度是 20 fps，而演算法得花上 10 毫秒才能執行完畢，則適當的等待時間就是 *30* = (*1000*/*20*) - *10* 毫秒。這個計算公式的邏輯考慮到等待時間必須夠長，才能確保下個畫格的內容已完全讀入緩衝區；如果使用的相機需要 40 毫秒才能處理完一個影像，而演算法同樣得花 10 毫秒，那麼 waitKey 函式的參數值就只需要使用 30 毫秒，因為 30 毫秒的等待時間再加上 10 毫秒的演算法運算時間，就相當於能夠從相機取得下個畫格所需要的時間。

使用者想要停止應用程式時，只需要按下任何按鍵，程式就會使用 release 函式釋放所有的影片資源。

對電腦影像應用程式而言，釋放執行過程中取得的所有資源十分重要，要是沒有釋放資源，就會耗盡所有的隨機記憶體，程式可以透過 release 函式釋放矩陣。

程式的執行結果會開啟新的視窗，顯示影片內容，或以 BGR 格式顯示 webcam 抓取到的畫面。