高等電腦視覺

作業#(1)

姓名:闕楷宸
學號:114318047
指導老師:黃正民

- 作業說明
  - 作業架構
    - bmp.hpp
    - bmp.cpp
    - HW1.cpp
    - HW1_opencv.cpp
- 建置執行**(window,linux)**
  - 如何在linux,window建置、執行
- Q1
- Q2
- Q3
- Qbonus_1
- Qbonus_b

作業說明:

作業架構:

```
├── bmp.cpp
├── bmp.hpp
├── CMakeLists.txt
├── HW1.cpp
├── HW1_opencv.cpp
├── output_image
│   ├── task1.bmp
│   ├── task1_bonus_0.5x.bmp
│   ├── task1_bonus_2x.bmp
│   ├── task1_bonus_2x_copy.bmp
│   ├── task1_bonus_2x_rotated.bmp
│   ├── task1_bonus_2x_rotated_channel_interchanged.bmp
│   ├── task1_opencv.bmp
│   ├── task2.bmp
│   ├── task2_bonus.bmp
│   ├── task2_bonus_copy.bmp
│   ├── task2_bonus_rotated.bmp
│   ├── task2_bonus_rotated_channel_interchanged.bmp
│   ├── task2_opencv.bmp
│   ├── task3.bmp
│   └── task3_opencv.bmp
├── test_image.bmp
└── window_user
    ├── HW1.exe
    ├── HW1_opencv.exe
    ├── opencv_world4110.dll
    └── test_image.bmp
```

程式介紹:

bmp.hpp:

定義bmp檔structure與讀寫function(writeBMP,ReadBMP)

BMPHeader

bfType	2 bytes
bfSize	4 bytes
bfReserved1	2 bytes
bfReserved2	2 bytes
bfOffBits	4 bytes

```
struct BMPHeader {
    uint16_t bfType;
    uint32_t bfSize;
    uint16_t bfReserved1;
    uint16_t bfReserved2;
    uint32_t bfOffBits;
};
```

BMPInfoHeader

biSize	4 bytes
biWidth	4 bytes
biHeight	4 bytes
biPlanes	2 bytes
biBitCount	2 bytes
biCompression	4 bytes
biSizeImage	4 bytes
biXPelsPerMeter	4 bytes
biYPelsPerMeter	4 bytes
biClrUsed	4 bytes
biClrImportant	4 bytes

```
struct BMPInfoHeader {
    uint32_t biSize;
    int32_t  biWidth;   // defi
    int32_t  biHeight;  // same
    uint16_t biPlanes;
    uint16_t biBitCount;
    uint32_t biCompression;
    uint32_t biSizeImage;
    int32_t  biXPelsPerMeter;
    int32_t  biYPelsPerMeter;
    uint32_t biClrUsed;
    uint32_t biClrImportant;
};
```

readBMP,writeBMP

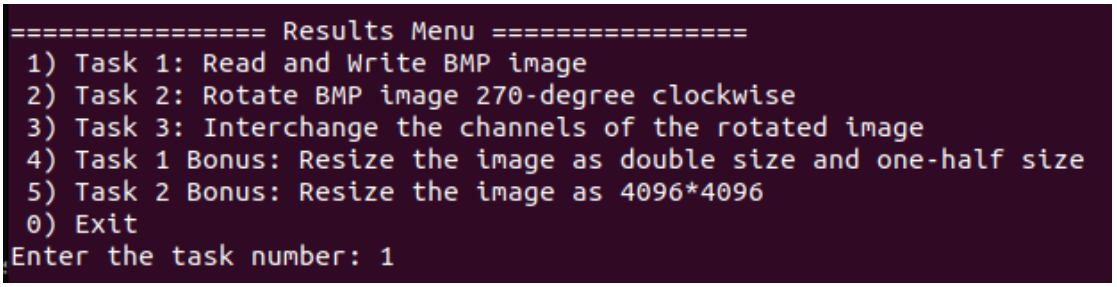
```
BMPImage readBMP(const char* filename);

// writeBMP will be defined in bmp.cpp
void writeBMP(const char* filename, const BMPImage& img);
```

bmp.cpp:

根據bmp.hpp所定義撰寫讀寫程式

HW1.cpp主程式:



輸入題號輸出答案，圖像也會輸出至同路徑

HW1_opencv.cpp:

使用opencv進行撰寫，檔名會附註xxx_opencv.bmp

建置執行(window,linux)

Linux:

1.使用opencv 4.5.4(sudo apt install libopencv-dev)

2.建置與執行:

```
cd HW1_114318047/
cmake -S . -B build
cmake --build build -j
./build/HW1
./build/HW1_opencv
```

Window:

```
cd window_user
./HW1.exe
```

Window 建置:

1:Cmake:

winget install Kitware.CMake

2:OpenCV for window:

C:\opencv\build\x64\vc16\bin(make sure DLLs are in this dir)

3:Confirm CMake config file exists:

C:\opencv\build\x64\vc16\lib\OpenCVConfig.cmake

4:Build and run

cd "C:\HW1_114318047"

5:Configure:

cmake -S . -B build -G "Visual Studio 17 2022" -A x64 -DOpenCV_DIR="C:\opencv\build\x64\vc16\lib"

6:Produce exe:

cmake --build .\build --config Release

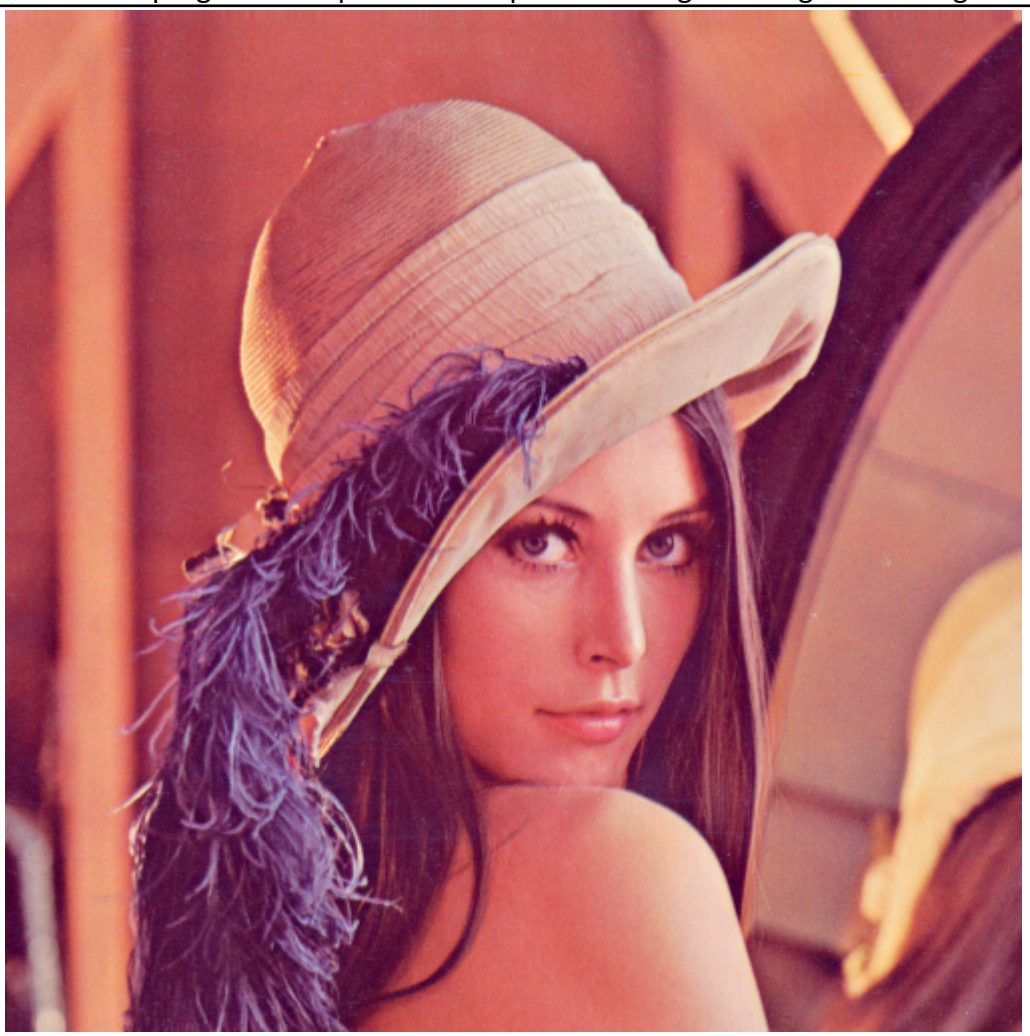
7:Run exe

.\build\Release\HW1.exe

.\build\Release\HW1_opencv.exe

1. Image Read/Write

Write a program to implement “bmp format image reading and writing.



Discussion(1)

```
static void task1(const char* input,const char* output)
{
    // Read readBMP(const char* filename)
    bmp::BMPImage img = bmp::readBMP(input);

    // write
    bmp::writeBMP(output, img);
}
```

```
: task1("test image.bmp","task1.bmp");
```

w/o OpenCV(**HW1**):
使用先前在bmp.cpp、bmp.hpp定義的讀寫功能進行，並且使用test_image.bmp(512x512)的參考圖像，並輸出圖像(**task1.bmp**)

w OpenCV(**HW1_opencv**):
先使用OpenCV的imread讀取圖像,再使用imwrite寫出圖像(**task1_opencv.bmp**)

2.Image Rotation

Do a 270-degree clockwise rotation over the input image
to generate the output image.

Discussion (2)

```cpp
for (int r = 0; r < N; ++r) {
    const uint8_t* srcRow = &img.data[r * rowSize];
    for (int c = 0; c < N; ++c) {
        const uint8_t* srcPx = &srcRow[c * 3];

        uint8_t* dstRow = &out[c * rowSize];
        uint8_t* dstPx  = &dstRow[(N - 1 - r) * 3];

        dstPx[0] = srcPx[0]; // B
        dstPx[1] = srcPx[1]; // G
        dstPx[2] = srcPx[2]; // R
    }
}
img.data.swap(out);
bmp::writeBMP(output, img);
```

讀取完圖像後存入img，並使用列指標、行指標指向要處理的pixel

並且使用(N-1-r)*3可實現逆時針90度的旋轉

srcRow會指向參考圖像的列
srcPx會指向該列(參考圖像)的pixel(指向blue channel)
比如說圖像為

```
[B20][G20][R20] [B21][G21][R21] [B22][G22][R22] [padding]
[B10][G10][R10] [B11][G11][R11] [B12][G12][R12] [padding]
[B00][G00][R00] [B01][G01][R01] [B02][G02][R02] [padding]
```

當r=c=0時，srcRow指向img.data[0](原始圖像的第一位置)
srcPx指向srcRow[0]，相當於也指向img.data[0]

而dstRow指向out[0](輸出圖像的第一個位置)
dstPx指向dstRow[6]，相當於指向out[6]，也就是[B02]的位置
再透過srcPx[0]~srcPx[2]分別將B、G、R通道assign到dstPx所指向的[B02]分次存放

我們可發現原先在[B00]的值經過處理後會到[B02]的位置(完成一次旋轉)
最後輸出圖像會被寫入至**task2.bmp**

3.Image Channel Interchange

Interchange the channels of the rotated image,R=>G G=>B B=>R
R’ = G(old) G’ = B(old) B’ = R(old)

Discussion (3)

```cpp
for (int r = 0; r < outH; ++r) { //height
    // point to first byte of row r
    uint8_t* RowPtr = &img_copy.data[r * outRow];
    for (int c = 0; c < outW; ++c) { //width
        // point at pixel (r,c)
        uint8_t* Px = &RowPtr[c * 3]; // B,G,R

        // Original channels
        uint8_t B = Px[0];
        uint8_t G = Px[1];
        uint8_t R = Px[2];

        // Interchange channels: R=>G, G=>B, B=>R
        Px[0] = R; // B' = R(old)
        Px[1] = B; // G' = B(old)
        Px[2] = G; // R' = G(old)
    }
}
```

與前提類似，一樣使用列與行指標
當r=c=0
RowPtr指向img_copy.data[0](參考圖像的第一位)
Px指向RowPtr[0]，也是指向img_copy.data[0]
(當c增加時，Px會分別指向同一列的不同pixel)
(當r增加時，換成不同列，繼續掃描)

使用B、G、R存各個pixel的BGR，再使用Px[0]到Px[2]分別存R、B、G

因為R' = G(old),G'=B(old),B'=R(old)
所以如Px[0] = R(將R(old)存入B')

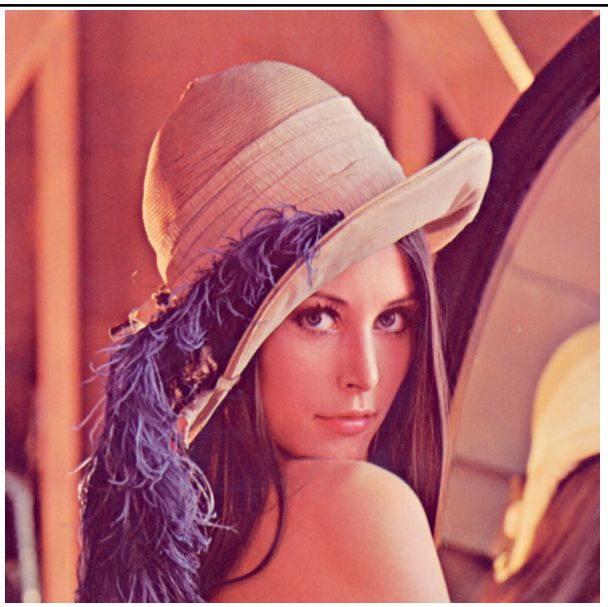
最後會將輸出圖像寫入task3.bmp

bonus1

Resize the image as double size and one-half size.
Repeat 1~3 for the resized image

原圖(512x512)(test_image.bmp)

縮小一半(256x256)(task1_bonus_0.5x.bmp)



放大1倍(1024x1024)(task1_bonus_2x.bmp)

Image read/write(task1_bonus_2x_copy.bmp)

順時針270旋轉(task1_bonus_2x_rotated.bmp)

Interchange the channels of the rotated image(task1_bonus_2x_rotated_channel_interchanged.bmp)

Discussion (1.2b)

```
const int srcW = img.width;
const int srcH = img.height;
const int srcRow = bmp::rowSizeBytes(srcW);    // inclu

const int dstW_2x = srcW * 2;
const int dstH_2x = srcH * 2;
const int dstRow_2x = bmp::rowSizeBytes(dstW_2x);

const int dstW_05x = srcW / 2;
const int dstH_05x = srcH / 2;
const int dstRow_05x = bmp::rowSizeBytes(dstW_05x);

// Allocate destination pixel buffer with correct padd
std::vector<uint8_t> out_2x(dstRow_2x * dstH_2x);
std::vector<uint8_t> out_05x(dstRow_05x * dstH_05x);
```

由於我們需要將輸出圖像resize成原本的0.5跟2倍，因此先宣告輸出圖像的長寬資訊
rowSizeByte是為了確保每一列的大小都可以被4整除(BMP alignment)
參考圖像(test_image.bmp)為512x512的圖像

並且宣告out_2x 跟 out_05x 分別存入兩倍與0.5倍的輸出圖像，之所以不使用malloc或new是因為使用std::vector無須free 或delete memory，也不會造成memory leak，std::vector的特點是當離開function後即會自動free memory。

```cpp
// 2x (Nearest Neighbor)
for (int r = 0; r < srcH; ++r) {
    //each row has srcRow bytes
    const uint8_t* srcRowPtr = &img.data[r * srcRow];
    for (int c = 0; c < srcW; ++c) {
        const uint8_t* srcPx = &srcRowPtr[c * 3]; // B,G,R
        const int dst_r = r * 2;
        const int dst_c = c * 2;

        uint8_t* dstRowPtr = &out_2x[dst_r * dstRow_2x];
        uint8_t* dstPx     = &dstRowPtr[dst_c * 3];

        for (int dr = 0; dr < 2; ++dr) {
            for (int dc = 0; dc < 2; ++dc) {
                uint8_t* p = &out_2x[(dst_r + dr) * dstRow_2x + (dst_c + dc) * 3];
                // srcPx[0] srcPx[1] srcPx[2] are B,G,R in one pixel
                /*
                    dr=dc=0=> p[0]=>&out_2x[0]=[B00]
                    dr=0,dc=1=> p[0]=>&out_2x[3]
                */
                p[0] = srcPx[0]; // B
                p[1] = srcPx[1]; // G
                p[2] = srcPx[2]; // R
            }
        }
    }
}
```

此code block是使用nearest neighbor的方法進行 resize
次function目標是resize成原先的2倍

一樣是使用列與行指標，分別指向輸入及輸出的行、列
並且使用兩層for loop將輸入圖像的pixel指向輸出2x2區塊，以實現放大兩倍功能

當r=c=0時，srcRowPtr會指向img.data[0](參考圖像的第一個byte)
srcPx會指向srcRowPtr[0]，相當於指向img.data[0]

並且使用dstRowPtr指向out_2x[0](輸出圖像)，dstPx會指向dstRowPtr[0]，也就是out_2x[0]

```cpp
// 0.5x dstH_05x = srcH / 2
for (int r = 0; r < dstH_05x; ++r) {
    uint8_t* dstRowPtr = &out_05x[r * dstRow_05x];
    for (int c = 0; c < dstW_05x; ++c) {
        const int src_r = r * 2;
        const int src_c = c * 2;
        const uint8_t* srcRowPtr = &img.data[src_r * srcRow];
        const uint8_t* srcPx = &srcRowPtr[src_c * 3]; // B,G,R

        uint8_t* dstPx = &dstRowPtr[c * 3];
        // Copy B, G, R
        dstPx[0] = srcPx[0]; //img.data[0] = B
        dstPx[1] = srcPx[1];
        dstPx[2] = srcPx[2];
    }
}
```

此code block是resize成½的方法與先前類似

使用列、行指標進行處理
需要將迴圈範圍改成原先的½，並且可將參考圖像的pixel每隔兩個存入目標pixel位置
相似4x4 resize 成2x2為例

```
[B12][G12][R12] [B13][G13][R13] [B14][G14][R14] [B15][G15][R15] [padding]
[B08][G08][R08] [B09][G09][R09] [B10][G10][R10] [B11][G11][R11] [padding]
[B04][G04][R04] [B05][G05][R05] [B06][G06][R06] [B07][G07][R07] [padding]
[B00][G00][R00] [B01][G01][R01] [B02][G02][R02] [B03][G03][R03] [padding]
```

```
[B03][G03][R03] [B04][G04][R04] [padding]
[B00][G00][R00] [B01][G01][R01] [padding]
```

當r=c=0時，dstRowPtr會指向輸出圖像的第一個byte[B00]，並且srcRowPtr會指向參考圖像的第一個byte[B00]

srcPx會指向srcRowPtr[0]也就是img.data[0][B00]
dstPx會指向dstRowPtr[0]，因此會指向輸出圖像的第一個Byte[B00]

當r=0,c=1時，srcPx會指向srcRowPtr[6]，也就是[B02]，並且dstPx會指向out_05x[3]，也就是輸出圖像的第二個pixel[B01]

由此依序執行此可將參考圖像resize成原先的一半。
最後輸出圖像為**task1_bonus_0.5x.bmp** 、**task1_bonus_2x.bmp**以及其他檔名帶有task1_bonus的圖像(Repeat 1~3 for the resized image)(使用放大兩倍的圖像進行處理)

bonus2
Resize the image as 4096x4096 Repeat 1~3 for the resized image.

原圖(512x512)(test_image.bmp)

Resize to 4096x4096(task2_bonus.bmp)

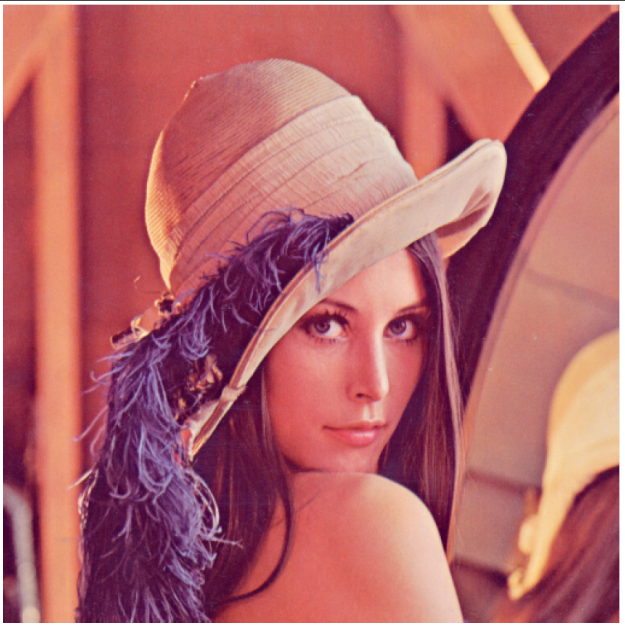
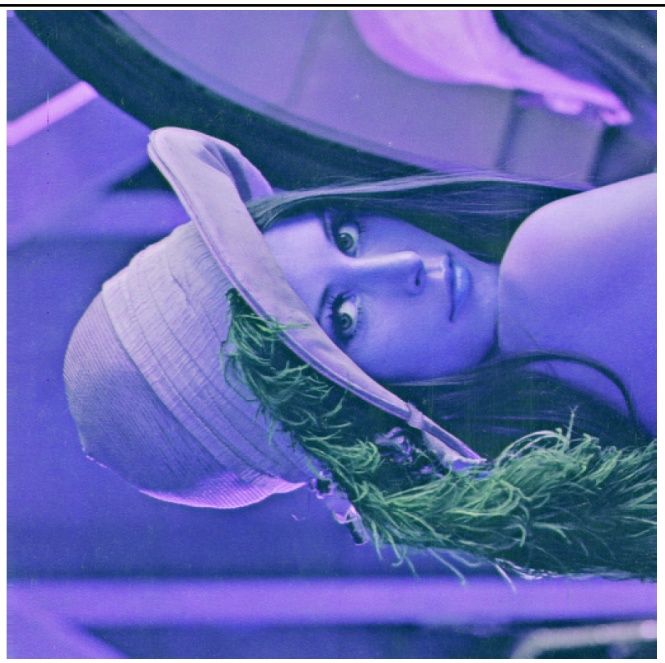


Image read/write(task2_bonus_copy.bmp)

順時針轉270(task2_bonus_rotated.bmp)



Interchange the channel of image
(task2_bonus_rotated_channel_interchanged.bmp)

Discussion(bonus2)

```
for (int r = 0; r < dstH; ++r) {
    uint8_t* dstRowPtr = &out[r * dstRowByte];
    const int src_r = r / 8;
    for (int c = 0; c < dstW; ++c) {
        const int src_c = c / 8;
        const uint8_t* srcRowPtr = &img.data[src_r * srcRowByte];
        const uint8_t* srcPx = &srcRowPtr[src_c * 3]; // B,G,R

        uint8_t* dstPx = &dstRowPtr[c * 3];
        // Copy B, G, R
        dstPx[0] = srcPx[0]; //img.data[0] = B
        dstPx[1] = srcPx[1];
        dstPx[2] = srcPx[2];
    }
}
```

由於參考圖像(test_image.bmp)為512x512，因此需要放大8倍才能變成4096x4096的輸出圖像，所以使用此程式，從以上程式可以發現當r跟c能被8整除時，才會將將參考圖片的pixel複製至輸出圖像，以實現放大8倍的功能

最後輸出圖像為**task2_bonus.bmp**以及其他檔名帶有task2_bonus的圖像
(Repeat 1~3 for the resized image)