영상처리 HW #1

2011253020 이화중

# Generate a 512x512 RAW Image

주어진 그래프를 바탕으로 RAW Image를 생성하고 Mach Bands를 찾는 문제이다. 그래프의 n을 Image의 행으로 가정하고, r(n)을 Pixel 값으로 한다. 각 열은 각각 같은 값이다.

문제에서는 512\*512 크기의 RAW Image를 요구하였다. 그런데 RAW Image는 폭 및 높이의 Pixel 수 및 Pixel당 바이트 등 Image에 대한 별도의 정보가 없다. 따라서, 생성한 RAW Image를 Photoshop 등의 상용 프로그램에서 열기 위해서는, Figure 1과 같이, 사전에 폭 및 높이의 Pixel 수와 Pixel당 바이트 등의 Image에 대한 별도의 정보를 설정하는 작업이 필요하다. Figure 2는, 생성된 RAW Image를 Adobe Photoshop CC에서, Figure 1과 같이 - 폭 512, 높이 512 Pixels 및 Pixel 당 8-bits (1 byte), Header Size 0 byte로 - 설정하여 얻은 것이다.

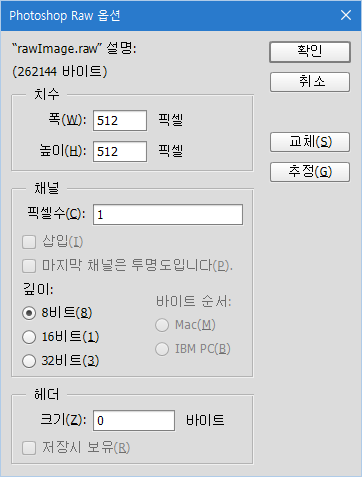


Figure 1 Options for opening RAW image on Adobe Photoshop CC

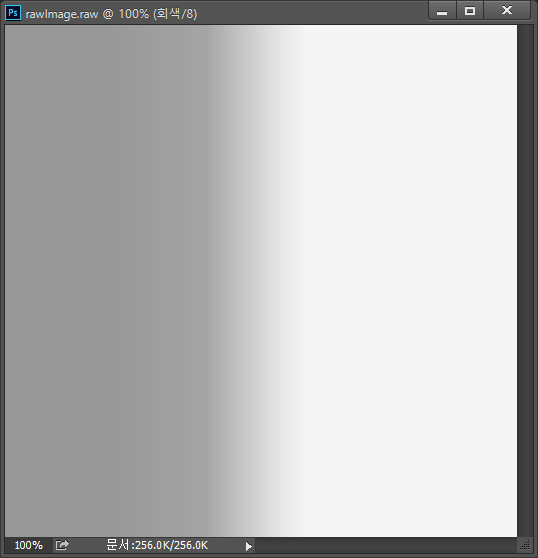


Figure 2 RAW image on Adobe Photoshop CC (rawImage.raw)

문제에서 주어진 그래프에서 기울기가 급격하게 변하는 부분은 n=200과 n=300이다. 따라서 이 언저리에서 Mach Band가 발생한다. 이 효과에 따라 사람의 눈은, n=200의 앞은 실제보다 더 어둡게 인식할 것이고, n=300의 뒤는 실제보다 더 밝게 인식할 것이다. 이 부분을 표시한 그림은 Figure 3와 같다.

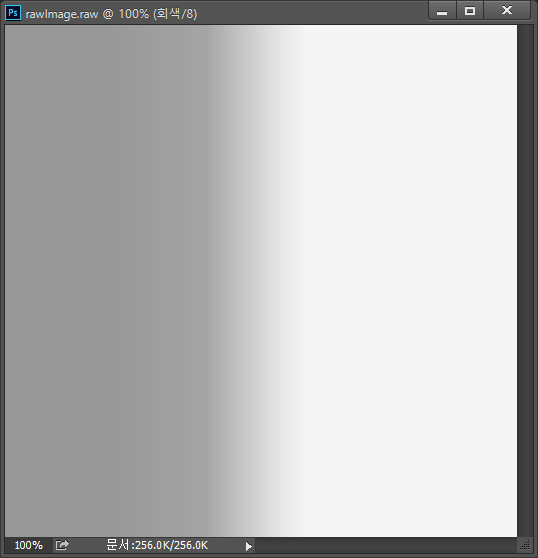


Figure 3 RAW image with mach bands

이 Image에서 [0, 200) 영역은 밝기가 거의 일정해 보인다. 이 영역에서 실제의 밝기 값이 밝아지지만, 사람 눈에는 실제 값보다 더 어둡게 인식되어, 결과적으로 밝기가 일정한 것처럼 인식되기 때문이다. 그리고, (280, 300) 영역은 (200, 280) 영역보다 기울기가 다소 완만해졌지만, 그 그래프 기울기 차이가 상대적으로 크지 않아 n=280에서는 뚜렷한 경계선을 느끼기 어렵다. 그럼에도 불구하고, n=200의 직전과 n=300의 직후는 전후의 그래프 기울기 차이가 크기 때문에, 각각 더 주변보다 어둡고 밝게 보여 경계선이 상대적으로 뚜렷해 보인다. 이를 통하여 Mach Bands를 시각적으로 찾을 수 있다.

RAW Image File을 생성하는 데 구현한 함수는 genrateRAWofProblem1(), getBrightnessOfProblem1(), writeRAW()이다. genrateRAWofProblem1() 함수는 문제 1번에 종속적인 RAW Image를 만들어내는 전반적인 동작을 관장한다. getBrightnessOfProblem1() 함수를 호출하여 주어진 그래프의 값을 n이 증가하는 순으로 구하고, 이 값들을 토대로 이차원 배열에 값을 할당하며, writeRAW() 함수를 호출하여 이 배열에 해당하는 RAW Image를 생성한다. getBrightnessOfProblem1() 함수는 그래프의 r(n) 값들을 구하여 n이 증가하는 순서에 따라 순차적으로 배열에 저장하여 Out Parameter로 데이터를 반환한다. writeRAW() 함수는 배열의 값을 File에 쓴다. RAW Image는 폭 및 높이 Pixel에 대한 정보가 없으므로, 일차원 배열을 받는다. 다만, 사용자가 원하는 Image를 얻기 위해서는, 생각한 Image의 Pixel의 행 단위로 연속하여, Parameter로 전달할 배열의 데이터를 구성해야할 것이다. fputc()를 배열의 크기만큼 반복 호출하여 File에 byte 단위로 쓴다. 이 루틴은 main()에서 genrateRAWofProblem1()를 호출하여 시작하며, 출력하는 File은 “rawImage.raw”(Figure 2)이다.

# Generate a BMP (bitmap) Image File

1번 문제에서 생성한 RAW Image를 Windows Bitmap Image(이하. BMP)로 변환하는 문제이다. RAW Image와 달리 BMP Image에는 Header가 붙어 있기 때문에, Header를 추가적으로 만들어야 한다. 이 정보는, 문제의 요구에 따라, “lena\_512x512.bmp”에서 가져온다.

BMP Header는 File Header, Image Info Header, Palette가 순서대로 붙어있다. File Header는 BMP File 자체에 대한 정보를 담는다. Image Info Header는 BMP 영상에 대한 정보를 담는다. Palette는 BMP의 영상이 256색 이하(8-bits 이하)인 경우에 존재하며, Image의 색상 정보를 담는다. BMP File임을 나타내는 Magic Number는, File Header에 포함되어 있으며, 그 값은 0x4D42이다. 따라서 BMP Image File을 생성할 때는 반드시 이 값으로 설정해야 한다.

Windows 상에서 BMP를 다루는 경우, BMP Image의 각 Header에 대한 정보(구조체)는 wingdi.h에 정의되어 있으므로, Windows.h Header File를 include하고 BMP Header 구조체 변수를 선언하여 Header를 생성할 수 있다. File Header는 BITMAPFILEHEADER 구조체, Image Info Header는 BITMAPINFOHEADER 구조체로 구현되어 있으며, Palette는 RGBQUAD 구조체의 배열로 구성된다.

BMP Header의 뒤에는 실제 영상에 대한 Pixel 데이터가 저장된다. 이 Pixel 데이터와 RAW Image의 그것과 가장 큰 차이점은, 이 데이터의 아랫부분이 위로 가도록 출력된다는 것이다. 다시 말해, Pixel 데이터를 그대로 저장하면 상하 대칭된 영상을 얻는다는 것이다. 따라서, RAW Image를 BMP Image로 변환할 때는 Pixel 데이터를 상하대칭시켜 저장하여야 한다.

BMP의 Header를 가져오는 데는 readBMPHeader()와 getBMPHeader()를, RAW Image와 합쳐 BMP Image를 생성하는 데는 convertRAWtoBMP()와 writeBMP(), putBMPHeader() 함수를 구현하였다. readBMPHeader()는 File 이름 문자열을 받아 그 File을 열고 getBMPHeader()를 호출하여 BMP Header를 가져온다. 이때 BMP File의 Magic Number 검사를 함께 한다. getBMPHeader()는 실제로 File에서 BITMAPFILEHEADER 구조체, BITMAPINFOHEADER 구조체, RGBQUAD 구조체 배열을 순서대로 읽어오며, 읽어온 Header를 Out Parameter로 반환한다. 이때 Pixel 크기가 8-bits를 초과하는 경우는 Palette는 읽지 않는다.

convertRAWtoBMP() 함수는 RAW Image File에 BMP Header를 합쳐서 새로운 BMP Image를 생성한다. RAW Pixel Data의 상하반전으로 BMP Image에 저장해야하므로, RAW Image File을 읽어서 Pixel Data 배열에 저장할 때 상하반전하여 저장하였다. 그리고 Header와 함께 이 배열을 writeBMP()를 호출하여 BMP File을 생성하였다. writeBMP() 함수는 Header들과 Pixel Data를 BMP Image File에 쓴다. Header를 쓸 때는 putBMPHeader()를 호출한다.

이 루틴은 각각 readBMPHeader() 함수와 convertRAWtoBMP() 함수를 호출하여 시작한다. readBMPHeader() 함수 호출에 사용하는 BMP Image File은 “lena\_512x512.bmp” 이다. convertRAWtoBMP()에 입력되는 File은 “rawImage.raw”(Figure 2)이고, 생성하는 File은 “bmpImage.bmp”이다. 생성된 Image인 “bmpImage.bmp”는 Figure 4와 같다.

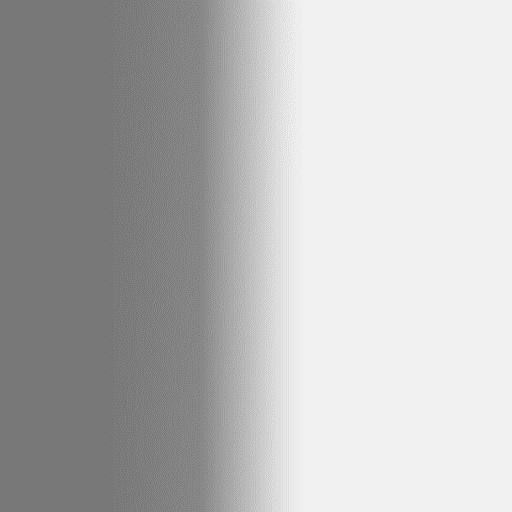


Figure 4 BMP image converted from RAW image (bmpImage.bmp)

# Rotate the Image Obtained to the Clockwise

2번 문제에서 생성한 BMP Image를 시계방향으로 90도 회전하여 새로운 BMP Image File로 저장한다. 이 문제에서 중요한 것은 회전 알고리즘이다. 시계 방향으로 90도 회전하라고 하였, BMP Image File에 저장된 Pixel Data 배열을 시계방향으로 회전시키면 안된다. BMP 영상은 저장된 Pixel Data를 상하반전하여 출력되기 때문에, 이 배열을 시계방향으로 회전시키면 반시계방향으로 90도 회전시킨 영상이 출력된다. 따라서, Pixel Data 배열을 반시계방향으로 90도 회전시켜야 시계방향으로 회전된 영상을 얻을 수 있다.

시계방향으로 90도 회전한 BMP 영상을 얻는 데 구현한 함수는 rotateBMP(), getBMPHeader(), writeBMP(), putBMPHeader()이다. rotateBMP() 함수는 우선, getBMPHeader()를 호출하여 원본 BMP Image의 Header를 가져온다. 그리고 시계방향으로 90도 회전하므로, Header의 폭 Pixel 수와 높이 Pixel 수를 서로 맞바꾼다. 다음으로 BMP Image의 영상 크기로 동적 할당한 배열에 원본 Image에서 가져온 Pixel Data를 저장한다. 이때 배열에 좌측 하단에서부터 상단 방향으로, 다시 우측 방향의 순서로 저장한다. 이를 통해 Pixel Data는 반시계방향으로 회전되어 저장되며, 결과적으로 시계방향으로 회전된 영상을 얻게 된다. 마지막으로 변환된 데이터를 writeBMP() 함수를 통하여 파일에 쓴다. 나머지 함수는 문제 2번에서 설명한 것과 동일하다. 이 루틴은 rotateBMP()를 호출하여 시작하며, 입력 파일은 “bmpImage.bmp”(Figure 4)이고 출력 파일은 “rotated\_bmpImage.bmp”(Figure 5)이다.

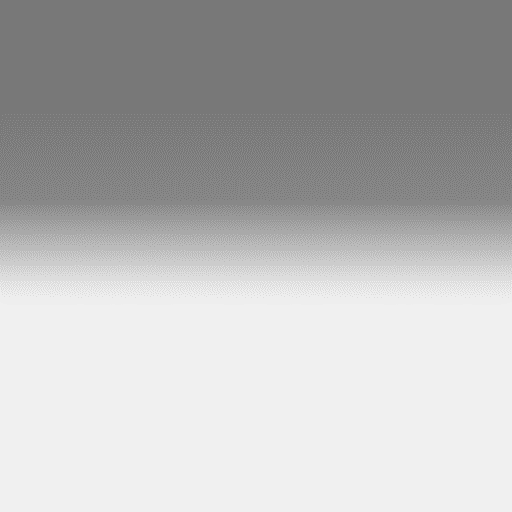


Figure 5 Rotated BMP image of Problem 2 (rotated\_bmpImage.bmp)

# Discussion

이번 과제는 Pixel Data, Mach Band, BMP Image File Structure에 대한 이해를 필요로 한다.

각 함수는 최대한 일반적으로 작성하고자 하였다. 따라서 과제와 독립적으로 사용할 수 있는 함수들과 과제에 종속적인 함수를 별도로 분리하여 구현하였다. 독립적으로 동작할 수 있는 함수는 ImageProcessor.h에 선언하였고 ImageProcessor.c에서 정의하였다. 문제 1번에서 요구한 특정 동작을 취하는 함수는 Problem1.h에서 선언하였고, Problem1.c에서 정의하였다. 그리고 각 함수에서는 8-bits Image가 아닌 경우에도 동작하도록 대응하였고, BMP File의 Magic Number도 검사하였으며, BMP Image의 폭과 높이를 별도로 관리하였다.

문제 1번은 512\*512 크기의 Image를 가정하였으나, 코드를 작성할 때는 다른 크기의 정방형 Image뿐만 아니라 폭과 높이의 크기가 다른 Image까지 대응하였다. 따라서 512\*512 크기가 아니더라도 RAW Image File을 쓰는 데 아무 문제가 없다. 다만, 문제에서 주어진 그래프가 [0, 512) 범위를 나타내므로, Problem1.h에서 선언된 함수를 사용하기 위해서는, 배열의 폭(WIDTH)가 512 이상이어야 한다. 또한 문제에서는 RAW Image가 이차원이라고 암시적으로 가정하므로 genrateRAWofProblem1() 함수는 이차원 배열을 받도록 하였지만, 실제의 RAW Image는 폭과 높이에 대한 정보를 갖고 있지 않으므로 writeRAW() 함수는 일차원 배열을 받도록 하였다.

문제 2번에서, RAW Image의 Pixel Data를 상하로 대칭하여, BMP의 Pixel Data로 옮겨야 한다. 그런데 문제 1번에서 주어진 Image는 자체가 상하가 이미 대칭되어 있다. 그래서 상하로 대칭하여 저장하지 않아도, 제대로 BMP Image를 생성한 것과 결과가 같았다. 아무리 프로그램에 상하로 대칭하는 코드를 추가하지 않았다 해도, Image만 놓고 본다면 충분히 정상적인 출력으로 착각할 만하다. 시각적으로 확실하게 파악하기 위해, 문제 1번의 RAW Image의 폭과 높이가 서로 뒤바뀐 Image를 테스트에 이용하였다. 이를 통하여 상하가 대칭되어 출력된다는 것을 시각적으로 확인할 수 있었고, Pixel Data의 상하를 대칭하는 코드를 적소에 추가할 수 있었다.

문제 3번에서, 영상을 시계 방향으로 90도 회전시켜야 한다. 그런데 BMP Image는 상하가 대칭되어 영상으로 출력되므로, 원하는 영상을 시계방향으로 회전시키고자 한다면, BMP의 Pixel Data는 반시계방향으로 회전시켜야 했다. 이를 위하여 파일을 BYTE 단위로 순차적으로 읽을 때, 읽은 순서대로 배열의 좌측 하단에서부터 상단으로 우측으로 저장하게 하였다.

그런데 문제의 Image는 상하가 대칭이므로, 시계 방향 90도 회전한 결과의 영상과 좌하-우상을 대칭한 결과의 영상이 같다. 따라서 이 Image를 한 번만 회전해서는 코드를 정확하게 작성하였는지 알 수 없었다. 따라서 두 가지 방법으로 테스트하였는데, 첫 번째는 회전 함수로 얻은 Image를 계속해서 회전시켜보는 것이었고, 두 번째는 영상 자체가 대칭이 아닌 BMP Image(lena\_512x512.bmp)를 입력하는 것이었다. 이 같은 방법으로 처음 작성한 회전 함수에 좌하-우상이 대칭되는 논리적 오류가 있다는 것을 발견하였고, 올바른 알고리즘으로 수정할 수 있었다. 그리고 이와 별개로 폭과 높이가 서로 다른 경우도 대응하였다.

# Environment

Language: C

Target: Win32 Application (console)

Tool: Microsoft Visual Studio 2015 (v140)