Digital Image Processing HW3

鄧奕辰 311605014

- Chromatic Adaptation

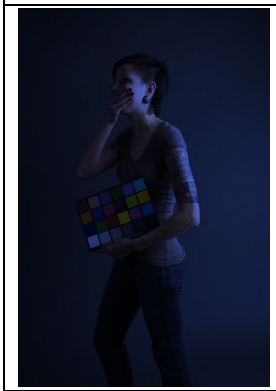
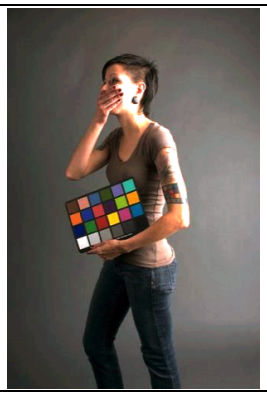
By specifying specific white areas in the image as a reference for white balance, the red, green, and blue values within the designated region are analyzed. The red, green, and blue values in that region are divided by the values of standard white, establishing a white balance ratio for the image. In the example below, I selected a 10x10 pixel area to mitigate the impact of individual pixels. After multiplying the pixel values of the entire image by the white balance ratio and constraining them to be within 255, the desired white balance effect can be achieved.

For input4.bmp, the presence of saturated colors and the brightest areas being pure white make it impractical to use the previously mentioned method of white balance through a pure white region. Therefore, the Gray World method must be employed. This involves individually determining the ratio of each color relative to the entire image.

```
void applyChromaticAdaptation(const RGBA** input, RGBA** dest, int width, int height, int patchSize, int wX, int wY) {
    int sumR = 0, sumG = 0, sumB = 0;
    const long double perfectWhite[3] = {255.0, 255.0, 255.0}; // Perfect white reference

    // input1 Wx = 410, Wy = 220
    // input2 Wx = 103, Wy = 467
    // input3 Wx = 450, Wy = 280
    for (int i = wX; i < wX+patchSize; ++i) {
        for (int j = wY; j < wY+patchSize; ++j) {
            sumR += input[i][j].r;
            sumG += input[i][j].g;
            sumB += input[i][j].b;
        }
    }
    int numPixels = patchSize * patchSize;
    // Calculate scaling factors
    long double redFactor = perfectWhite[0] / (sumR / numPixels+0);
    long double greenFactor = perfectWhite[1] / (sumG / numPixels+30);
    long double blueFactor = perfectWhite[2] / (sumB / numPixels+130);

    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; ++j) {
            dest[i][j].r = min(255, static_cast<int>(input[i][j].r * redFactor));
            dest[i][j].g = min(255, static_cast<int>(input[i][j].g * greenFactor));
            dest[i][j].b = min(255, static_cast<int>(input[i][j].b * blueFactor));
            dest[i][j].a = 0;
        }
    }
}
```

```cpp
void grayworld(const RGBA** input, RGBA** dest, int width, int height) {
    long double sumR = 0 ,sumG = 0, sumB = 0;
    long double avgR = 0 ,avgG = 0, avgB = 0;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            sumR += input[i][j].r;
            sumG += input[i][j].g;
            sumB += input[i][j].b;
        }
    }

    avgR = sumR / (width * height);
    avgG = sumG / (width * height);
    avgB = sumB / (width * height);

    // Calculate scaling factors
    long double redFactor = (avgR + avgG + avgB) / (3 * avgR);
    long double greenFactor = (avgR + avgG + avgB) / (3 * avgG);
    long double blueFactor = (avgR + avgG + avgB) / (3 * avgB);

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
        // Scale each channel based on the factors
            dest[i][j].r = min(255, static_cast<int>(input[i][j].r * redFactor));
            dest[i][j].g = min(255, static_cast<int>(input[i][j].g * greenFactor));
            dest[i][j].b = min(255, static_cast<int>(input[i][j].b * blueFactor));
        }
    }
}
```

原圖 1	白平衡 1	原圖 2	白平衡 2
			

原圖 3	白平衡 3	原圖 4	白平衡 4
			

- Image Enhancement

In this section, I applied the brightness enhancement from the previous assignment.This task involves transforming a low-brightness image into a high-brightness image, with the aim of achieving a higher enhancement ratio for the low-brightness regions compared to the high-brightness areas. The following operations were performed:

1. **Normalization** : Each pixel's red, green, and blue channel values were divided by 255 to normalize the channel values within the range of 0 to 1.
2. **Gamma Correction** : The normalized values for each channel were raised to the power of the gamma value, as illustrated in the diagram below.
3. **Reverse Normalization** : The calculated values were then multiplied by 255 to reverse the normalization and obtain new red, green, and blue channel values.
4. **Clipping** : To ensure that the color values remain within the range of 0 to 255, the min function was applied, and the resulting values were converted to integers before being stored in the target image.

The process is designed to disproportionately enhance low-brightness areas while applying a milder enhancement to high-brightness regions.

原圖 1	亮度調整 1	原圖 2	亮度調整 2
			

原圖 3	亮度調整 3	原圖 4	亮度調整 4