Image Processing Homework 1 Report

311511022 邱政岡

Image input/output

Bmp files include:

1. File header (14 bytes)
   - bfType (2 byte): Type of file, two char ‘B’ and ’M’ (or 0x424d) in bmp file.
   - bfSize (4 byte): Size of the file
   - bgReserved1 (2 byte)
   - bgReserved1 (2 byte)
   - bgOffBits (4 byte): Image offset. 14 + 40 + 0(palette) = 56 in this work
2. Info header (40 bytes)
   - biSize (4 byte): Size of the info header
   - biWidth (4 byte): Width of the image
   - biHeight (4 byte): Height of the image
   - biPlanes (2 byte):
   - biBitCount (2 byte): Bits per pixel
   - biCompression (4 byte)
   - biSizeImage (4 byte): Width * Height * channel
   - biXPelsPerMeter (4 byte)
   - biYPelsPerMeter (4 byte)
   - biClrUsed (4 byte)
   - biClrImportant (4 byte)
3. Palette (1024 bytes or 0 byte)
   No palette in this work.
4. Raw Image
   Store ABGR(32 bits) or BGR(24 bits) left to right and down to up. Must align to 4n bytes.

Read / Write:

In order, read/Write the file header, info header, and raw image. Before reading/writing the raw image, whether the width is aligned should be checked first. Don’t read the padding 0 / Write padding 0, if it is not aligned. Otherwise, the reading order is A G B R.

Image Flip

I use an array to store the raw image, so just outputting the 2nd index in reverse order can flip the image.

```
1.  BmpImage BmpImage::flip() const {
2.      BmpImage flipBmp;
```

```
    flipBmp.fileHeader = fileHeader;
    flipBmp.infoHeader = infoHeader;
    flipBmp.rawImage = new Pixel* [infoHeader.biHeight];
    for (int i=0; i<infoHeader.biHeight; i++) {
        flipBmp.rawImage[i] = new Pixel[infoHeader.biWidth];
        for (int j=0; j<infoHeader.biWidth; j++) {
            flipBmp.rawImage[i][j] = rawImage[i][infoHeader.biWidth - j];
        }
    }
    return flipBmp;
}
```

Resolution

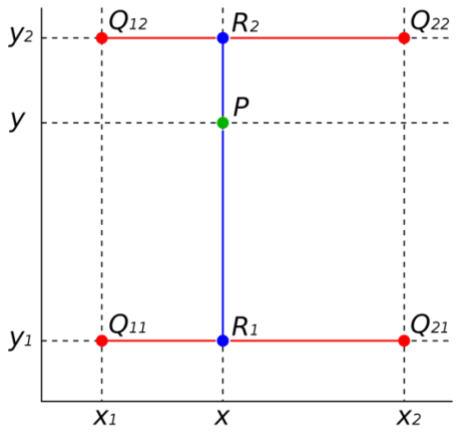
To quantify the signal from 8 bits to 6/4/2 bits. Rounding and truncation are typical examples of quantization processes. I choose truncation because it is simpler.
A bitwise and operation can do this. For example, bitwise and 111100 can truncate a 8 bits signal to 6 bits by setting the last bits to 0.

```
BmpImage BmpImage::resolution(int quanBit=6) const {
    uint8_t mask = 0xFF;
    for (int i=0; i<(8-quanBit); i++) {
        mask = mask << 1;
    }

    BmpImage resolBmp;
    resolBmp.fileHeader = fileHeader;
    resolBmp.infoHeader = infoHeader;
    resolBmp.rawImage = new Pixel* [infoHeader.biHeight];

    for (int i=0; i<infoHeader.biHeight; i++) {
        resolBmp.rawImage[i] = new Pixel[infoHeader.biWidth];
        for (int j=0; j<infoHeader.biWidth; j++) {
            if (infoHeader.biBitCount == 32) {
                resolBmp.rawImage[i][j].A = rawImage[i][j].A & mask;
            }
            resolBmp.rawImage[i][j].B = rawImage[i][j].B & mask;
            resolBmp.rawImage[i][j].G = rawImage[i][j].G & mask;
            resolBmp.rawImage[i][j].R = rawImage[i][j].R & mask;
```

```
        }
    }
    return resolBmp;
}
```

Scaling

Use inverse warping and bilinear interpolation. According to the new scaled image, calculate every pixel's corresponding location in the original image. Then bilinear interpolation algorithm helps to get the value between the discrete index.



To find the point P value f(x, y), we know the four points surrounding it. Q11 = (x1, y1), Q12 = (x1, y2), Q21 = (x2, y1), and Q22 = (x2, y2).

Do the linear interpolation at x-direction:

$$f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

And same at y-direction, calculate P by R_1 and R_2. Simplify the equation, we get:

$$f(x, y) = a_{00} + a_{10}x + a_{01}y + a_{11}xy$$

Where,

$$a_{00} = f(0\ ,0)$$
$$a_{10} = f(1,0) - f(0\ ,0)$$
$$a_{01} = f(0,1) - f(0\ ,0)$$
$$a_{11} = f(1,1) - f(0\ ,0) - f(0,1) + (0\ ,0)$$