

# BMP位图缩放

BMP文件是逐像素存储位图的一种图片格式. 它的数据主要分为三部分: 文件头, 信息头和图片数据部分.

文件头位于文件最开始, 组成部分如下:

名称	大小 (字节)	介绍
bfType	2	Windows 格式的 BMP 文件为「BM」。
bfSize	4	BMP 文件的大小。
bfReserved1	2	保留值。
bfReserved2	2	保留值。
bfOffBits	4	位图像素内容的偏移量 (起始地址)。

信息头位于文件头之后, 组成部分如下:

名称	大小 (字节)	介绍
biSize	4	此文件头 (BITMAPINFOHEADER) 的大小。
biWidth	4	图像的宽度, 以像素为单位。
biHeight	4	图像的高度, 以像素为单位。
biPlanes	2	图像的色彩平面, 正常情况下均为 1。
biBitCount	2	每个像素的位数。
biCompression	4	BMP 文件的压缩种类, 0 为无压缩。
biSizeImage	4	位图像素内容的大小, 无压缩时可以填 0。
biXPelsPerMeter	4	图像横向每米存放的像素。
biYPelsPerMeter	4	图像纵向每米存放的像素。
biClrUsed	4	图像使用的色彩数, 可以为 0。

名称	大小（字节）	介绍
biClrImportant	4	图像使用的「重要」的色彩数，0时均重要。

bmp文件几乎所有信息均为小端序存储，这对后面代码编写有利。

目前应用最多的是24位位图，每个像素用24位来存储RGB值，所以以下仅考虑24位位图。

图片数据部分把从左到右，从下到上地把每个像素的RGB值按顺序存储。为了加速计算机读取位图，每一行存储完后，会在最后用冗余字节补齐，使每一行的大小为4字节的整数倍。所以对于宽度为Width的位图来说，每一行的占用字节数Rowsize为：

$$Rowsize = \lceil \frac{Width \times 24}{32} \rceil \times 4$$

向上取整在计算机中不易实现，考虑：

$$\lceil \frac{a}{b} \rceil = \lfloor \frac{a-1}{b} \rfloor + 1$$

得到：

$$Rowsize = (\lceil \frac{Width \times 24 - 1}{32} \rceil + 1) \times 4 = \lfloor \frac{Width \times 24 + 31}{32} \rfloor \times 4$$

对于一张29\*26的24位位图来说， $Rowsize = \lfloor \frac{29 \times 24 + 31}{32} \rfloor \times 4 = 88$ 。图片数据部分大小  $88 \times 26 = 2288$ 。文件头与信息头一共54字节，文件大小应该为2342字节。经过验证，实际情况与计算结果相同。

属性	图像
分辨率	29 x 26
宽度	29 像素
高度	26 像素
位深度	24

大小： 2.28 KB (2,342 字节)

为了获取bmp文件信息，需要先定义文件头与信息头的结构体。在wingdi.h中已经有这两个头的定义，这里直接使用。

```
BITMAPFILEHEADER FileHeader; //位图文件头
```

```
BITMAPINFOHEADER InfoHeader; //位图信息头
```

定义像素结构体, 因为位图文件中像素是按照BGR的顺序存储, 所以这里也按照这个顺序定义结构体.

```
struct Pixel
{
    unsigned char b;
    unsigned char g;
    unsigned char r;
}; //像素结构体, 存储rgb值
```

为了方便人阅读相关信息, 写一个函数将小端法存储的整数转为十进制, 这里直接将要转换的内存拷贝到一个long int中, 然后输出这个long int即可.

```
//将小端法十六进制内存转为十进制
int LD2DEC(char* c, int len = 4)
{
    long l = 0;
    memcpy(&l, c, len);
    return (signed)l;
}
```

用vector<Pixel> 数组来表示一个位图, 每一个vector记录一行的像素.

输出一个位图时, 需要先构造相应的文件头与信息头. 然后逐像素输出文件. 当vector对应位置上没有记录时, 用空相素补齐. 注意当每一行输出完后, 还需要输出空相素来让每一行大小为4字节的整数倍.

```
//把bmp信息写到文件
void writebmp(vector<Pixel> *p, int height, int width, int rowsize,
ofstream& os)
{
    //构造文件头与信息头
    FileHeader.bfSize = sizeof(BITMAPFILEHEADER) + rowsize * height;
    InfoHeader.biWidth = width;
    InfoHeader.biHeight = height;

    cout << "New Size: " << FileHeader.bfSize << endl;
    cout << "New Rowsize: " << rowsize << endl;
    //输出文件头与信息头
```

```

os.write(reinterpret_cast<char*>(&FileHeader),
sizeof(BITMAPFILEHEADER));
os.write(reinterpret_cast<char*>(&InfoHeader),
sizeof(BITMAPINFOHEADER));

for(int i = 0;i < abs(height); i++)
{
    for(int j = 0;j < width; j++)
    {
        //cerr << i << " " << j << endl;
        if(j < p[i].size())
            os.write(reinterpret_cast<char*>(&p[i][j]), sizeof(Pixel));
        else
        {
            //输出空像素
            os.write(reinterpret_cast<char*>(&p0), sizeof(Pixel));
        }
    }
    //补齐四字节
    for(int j = 0;j < rowsize-3*width; j++)
    {
        os.write(reinterpret_cast<char*>(&p0.b), 1);
    }
}
}

```

放大图片时,先计算出新图片大小,然后将新图片上的像素按比例映射到原图片中,使用映射后的位置的像素即可.如果追求放大的高画质,还可以使用双线性插值在这里进行计算新像素的坐标.

```

//放大bmp
vector<Pixel>* zoom(vector<Pixel>* bmp, int height, int width, double
bs, int& rowsize, int&nheight, int& nwidth)
{
    nheight = int(bs * height);
    nwidth = int(bs * width);
    vector<Pixel> * bmp2 = new vector<Pixel> [nheight];

    for(int i = 0;i < abs(nheight); i++)
    {
        for(int j = 0;j < nwidth; j++)
        {
            bmp2[i].push_back(bmp[clamp(int(double(i)/bs), 0, height-1)]
[clamp(int(double(j)/bs), 0, width-1)]);
        }
    }
    //新的rowsize
    rowsize = ((InfoHeader.biBitCount * nwidth + 31) / 32) * 4;
    return bmp2;
}

```

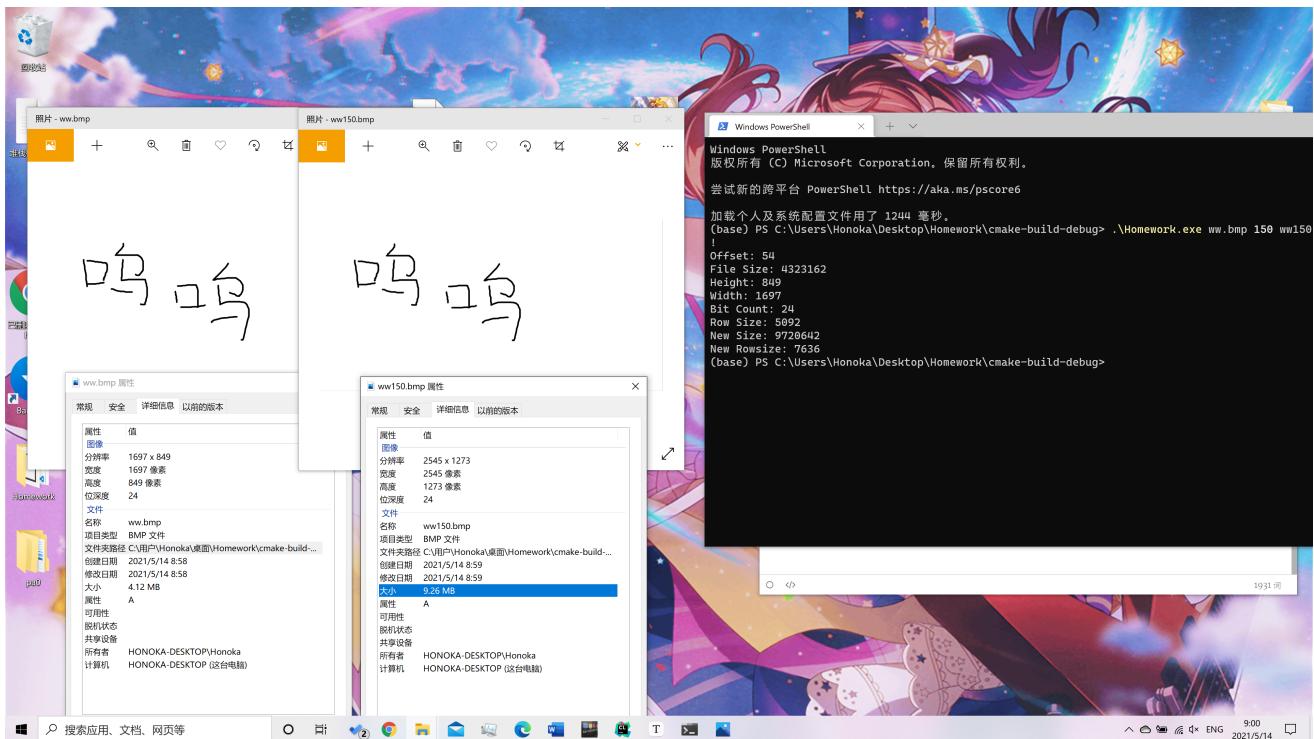
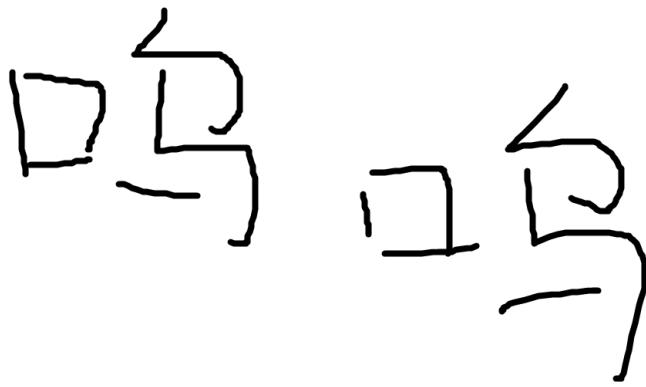
```
}
```

要放大一个图片，首先要进行读入。读入文件头与信息头之后，再逐字节读入像素即可。注意跳过冗余补齐字节。

```
int rowsize, width, height, bitcount;
cout << "!" << endl;
bmp.read(reinterpret_cast<char*>(&FileHeader),
sizeof(BITMAPFILEHEADER));
cout << "Offset: " << LD2DEC(reinterpret_cast<char*>
(&FileHeader.bfOffBits)) << endl;
cout << "File Size: " << LD2DEC(reinterpret_cast<char*>
(&FileHeader.bfSize)) << endl;
bmp.read(reinterpret_cast<char*>(&InfoHeader),
sizeof(BITMAPINFOHEADER));
cout << "Height: " << (height = LD2DEC(reinterpret_cast<char*>
(&InfoHeader.biHeight))) << endl;
cout << "Width: " << (width = LD2DEC(reinterpret_cast<char*>
(&InfoHeader.biWidth))) << endl;
cout << "Bit Count: " << (bitcount =
LD2DEC(reinterpret_cast<char*>(&InfoHeader.biBitCount))) << endl;
cout << "Row Size: " << (rowsize = ((InfoHeader.biBitCount *
InfoHeader.biWidth + 31) / 32) * 4) << endl;

auto *p = new vector<Pixel>[abs(height)];
//读入bmp
for (int i = 0; i < abs(height); i++)
{
    for (int j = 0; j < width; j++)
    {
        Pixel t;
        bmp.read(reinterpret_cast<char*>(&t), sizeof(Pixel));
        p[i].push_back(t);
    }
    bmp.ignore(rowsize - 3*width); //跳过补齐字节
}
int orowsize, nheight, nwidth;
auto outb = zoom(p, height, width, bs, orowsize, nheight, nwidth);
writebmp(outb, nheight, nwidth, orowsize, bmpout);
```

对这样一张图片进行放大：



可以看到，图片从1697\*849 放大到了2545\*1273，放大为150%.

## 小结

BMP文件格式是存储位图的最简单的格式之一，但是还是有很多诸如内存对齐的问题。在这个程序中放大仅仅使用了最简单的像素复制的方法，如果想得到更高质量的图片，可以使用双线性插值算法对图片进行推测性的补全，但是这就不是图片放大的范畴了。

在这个程序中，我对文件的二进制表示，内存相关的问题有了更深刻的了解，复杂的二进制文件表示也是按照一定的格式进行组织的。只需要按照对应的格式一层一层解开，文件的内容就清晰可见了。