专业：计算机科学与技术
姓名：
学号：
日期：2021/10/29

浙江大学实验报告

课程名称：图像信息处理 指导老师：宋明黎 成绩：

实验名称：图片二值化以及腐蚀、膨胀、开和闭操作

一、实验目的和要求

1. 图片二值化
2. 腐蚀
3. 膨胀
4. 开
5. 闭

二、实验内容和原理

1. 图片二值化和大津算法

二值化图片每个像素值只有 0 和 255，建立二值化图片的方法是将大于阈值的像素值置为 255，小于阈值的值置为 0。而如何找到这个阈值使得前景和背景的对比明显是关键，这就是大津算法解决的问题。这个算法的思路是遍历所有的阈值可能值，然后最大化前景和背景间的方差，具体的公式如上课时所提到的：

- $\sigma^2_{within}(T) = \frac{N_{Fgrd}(T)}{N}\sigma^2_{Fgrd}(T) + \frac{N_{Bgrd}(T)}{N}\sigma^2_{Bgrd}(T)$
- $\sigma^2_{between}(T) = \sigma^2 - \sigma^2_{within}(T)$
- $= \left(\frac{1}{N}\sum_{x,y}(f^2[x,y] - \mu^2)\right) - \frac{N_{Fgrd}}{N}\left(\frac{1}{N_{Fgrd}}\sum_{x,y\in Fgrd}(f^2[x,y] - \mu^2_{Fgrd})\right) - \frac{N_{Bgrd}}{N}\left(\frac{1}{N_{Bgrd}}\sum_{x,y\in Bgrd}(f^2[x,y] - \mu^2_{Bgrd})\right)$
- $= -\mu^2 + \frac{N_{Fgrd}}{N}\mu^2_{Fgrd} + \frac{N_{Bgrd}}{N}\mu^2_{Bgrd}$
- $= \frac{N_{Fgrd}}{N}(\mu_{Fgrd} - \mu)^2 + \frac{N_{Bgrd}}{N}(\mu_{Bgrd} - \mu)^2$
- $\rightarrow \frac{N_{Fgrd}(T)\cdot N_{Bgrd}(T)}{N^2}\left(\mu_{Fgrd}(T) - \mu_{Bgrd}(T)\right)^2$

上面公式的中σ^2_{within}是加权后的背景方差和前景方差和，$\sigma^2_{between}$是总方差减去σ^2_{within}，为了使得$\sigma^2_{between}$更大，而N^2又是常数，因此目标就是使得

$$N_{Fgrd}(T)\cdot N_{Bgrd}(T)\cdot\left(\mu_{Fgrd}(T) - \mu_{Bgrd}(T)\right)^2$$

这个式子最大。

上课之后给出了简化后的公式，笔者使用的也是简化后的公式：

Easier deduction

- $w_f = \frac{N_{Fgrd}}{N}, w_b = \frac{N_{Bgrd}}{N}, w_f + w_b = 1$
- $\mu = w_f * \mu_{Fgrd} + w_b * \mu_{Bgrd}$
- $\sigma_{between} = w_f(\mu_{Fgrd} - \mu)^2 + w_b(\mu_{Bgrd} - \mu)^2 = w_f(\mu_{Fgrd} - w_f * \mu_{Fgrd} - w_b * \mu_{Bgrd})^2 + w_b(\mu_{Bgrd} - w_f * \mu_{Fgrd} - w_b * \mu_{Bgrd})^2 \rightarrow w_b w_f(\mu_f - \mu_b)^2$

可以看到上述方法，是先求出前背景占的比例，再进行计算的。

2. 腐蚀、膨胀、开和闭

这几个图片处理方法是针对二值化图像的，都是使用一个 Structure Element 来对图像进行处理。腐蚀的原理是：用 Structure Element 的中心点来依次与图片的所有点重合，只有 Structure Element 能完整覆盖整个二值化图区域（即 1 对应 1，0 对应 0），才保留原图中心点位置的 255 值；膨胀的原理是：用 Structure Element 的中心点来依次与图片的所有点重合，只要 Structure Element 能覆盖整个二值化图区域的一个值，原图中心点位置就设为 255 值；开操作是先腐蚀再膨胀，闭操作是先膨胀再腐蚀。

三、实验步骤与分析

0. 读取 24 位 BMP 图片并转成 8 位灰度图

在实验一中已经读取了图片，但使用的方法较笨拙，笔者在此优化了实验一中读取 BMP 的方法，主要是使用结构体来进行读入。笔者一开始直接使用<Windows.h>中已经定义好的 BITMAPFILEHEADER、BITMAPINFOHEADER 和 RGBQUAD，之后直接将其添加到 gray.h 中。另外需要注意使用预编译指令#pragma pack(1)来取消结构体的对齐，确保依次读入的是正确的数据。

```
#pragma pack(1)
typedef unsigned char       BYTE;
typedef unsigned short      WORD;
typedef unsigned long       DWORD;
typedef unsigned long       LONG;

typedef struct tagBITMAPFILEHEADER {
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
} BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER{
        DWORD      biSize;
        LONG       biWidth;
        LONG       biHeight;
        WORD       biPlanes;
        WORD       biBitCount;
        DWORD      biCompression;
        DWORD      biSizeImage;
        LONG       biXPelsPerMeter;
        LONG       biYPelsPerMeter;
        DWORD      biClrUsed;
        DWORD      biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagRGBQUAD {
        BYTE    rgbBlue;
        BYTE    rgbGreen;
        BYTE    rgbRed;
        BYTE    rgbReserved;
} RGBQUAD;
```

然后笔者主要是在 gray.c 中进行读取 BMP 文件的主要属性，在定义好结构体并实例化之后，读取 BMP 文件就变得比较简单。

```
fseek(finput, 0, SEEK_CUR);
fread(&bmpFileHeader, sizeof(BITMAPFILEHEADER), 1, finput);
fread(&bmpInfo, sizeof(BITMAPINFOHEADER), 1, finput);

// Verify BMP
if (bmpFileHeader.bfType != 0x4d42)
{
    printf("This is not BMP file!\n");
    exit(1);
}
```

接下来笔者又通过 WriteGrayBMP 函数来进行 8 位灰度图的写入。读入的时候主要需要修改一些 BMP 文件的属性（位数、图片大小、偏移位置、文件大小、paddiing 大小），另外需要加上一段 256 色的调色板。注意 8 位灰度图也需要满足一行像素占的空间是 4 的倍数。

```
void WriteGrayBMP(BYTE *Gray, FILE *foutput)
{
    int i, j;
    // Write Gray BMP
    bmpInfo.biBitCount = 8;
    bmpInfo.biSizeImage = (bmpInfo.biWidth + 3) / 4 * 4 * bmpInfo.biHeight;
    bmpFileHeader.bfOffBits = sizeof(BITMAPINFOHEADER) + sizeof(BITMAPFILEHEADER) + 256 * sizeof(RGBQUAD);
    bmpFileHeader.bfSize = bmpFileHeader.bfOffBits + bmpInfo.biSizeImage;

    // Create RGBQUAD
    RGBQUAD *rgbquad = (RGBQUAD*)malloc(256 * sizeof(RGBQUAD));
    for (i = 0; i < 256; i++)
        rgbquad[i].rgbBlue = rgbquad[i].rgbGreen = rgbquad[i].rgbRed = i;

    fseek(foutput, 0, SEEK_SET);
    fwrite(&bmpFileHeader, sizeof(BITMAPFILEHEADER), 1, foutput);
    fwrite(&bmpInfo, sizeof(BITMAPINFOHEADER), 1, foutput);
    fwrite(rgbquad, sizeof(RGBQUAD), 256, foutput);

    int grayPad = (bmpInfo.biWidth + 3) / 4 * 4 - bmpInfo.biWidth;
    BYTE *padArray = (BYTE *)malloc(sizeof(BYTE) * grayPad);
    memset(padArray, 0, sizeof(BYTE) * grayPad);
    for (i = 0; i < bmpInfo.biHeight; i++)
    {
        for (j = 0; j < bmpInfo.biWidth; j++)
            fwrite(&Gray[i * bmpInfo.biWidth + j], sizeof(BYTE), 1, foutput);
        fwrite(padArray, sizeof(BYTE), grayPad, foutput);
    }

    free(rgbquad);
    free(padArray);
}
```

1. 二值化图像和大津算法

二值化图片的算法是很简单的，就如上述原理中所述，实现如下：

```
void Binarization(BYTE *Gray)
{
    BYTE threshold = FindThreshold(Gray);

    int i;
    for (i = 0; i < bmpInfo.biWidth * bmpInfo.biHeight; i++)
        if (Gray[i] < threshold)
            Gray[i] = 0;
        else
            Gray[i] = 255;
}
```

主要需要注意的是上面的 FindThreshold 函数，也就是大津算法。由于上面的原理中也产生的比较清楚，笔者的命名也可以见名知义，具体的实现如下：

```
BYTE FindThreshold(BYTE *Gray)
{
    // Get min and max pixel value
    int i, j;
    BYTE min = Gray[0];
    BYTE max = Gray[0];
    for (i = 0; i < bmpInfo.biHeight; i++)
        for (j = 0; j < bmpInfo.biWidth; j++)
        {
            if (Gray[i * bmpInfo.biWidth + j] < min)
                min = Gray[i * bmpInfo.biWidth + j];
            if (Gray[i * bmpInfo.biWidth + j] > max)
                max = Gray[i * bmpInfo.biWidth + j];
        }

    int threshold, optimalThreshold;
    double nF, nB;
    double wF, wB;
    double sumF, sumB;
    double averageF, averageB;
    double betweenVariance, maxBetweenVariance;
    maxBetweenVariance = 0;
```

上图中主要是一些初始化和函数的定义，其中 B 代表的是 Background 也就是背景，而 F 代表的是 Frontground 也就是前景。

```
for (threshold = min + 1; threshold < max; threshold++)
{
    nF = nB = sumF = sumB = 0;
    for (i = 0; i < bmpInfo.biHeight * bmpInfo.biWidth; i++)
        if (Gray[i] < threshold) // Background
        {
            nB++;
            sumB += Gray[i];
        }
        else // Foreground
        {
            nF++;
            sumF += Gray[i];
        }

    wF = nF / (bmpInfo.biHeight * bmpInfo.biWidth);
    wB = nB / (bmpInfo.biHeight * bmpInfo.biWidth);
    averageB = sumB / nB;
    averageF = sumF / nF;

    betweenVariance = wF * wB * (averageF - averageB) * (averageF - averageB);
    if (betweenVariance > maxBetweenVariance)
    {
        maxBetweenVariance = betweenVariance;
        optimalThreshold = threshold;
    }
}

return (BYTE)optimalThreshold;
```

这个部分是大津算法的主要实现，可以看到笔者遍历所有的 threshold 可能值，然后分别计算前背景方差值（如原理部分提到的简化公式），找到最大方差，得到最优 threshold。

3. 腐蚀、膨胀、开和闭

腐蚀的算法传入一个 Structure Element，然后移动这个 Structure Element 遍历整个图像。对图像上每一个中心点，比较它周围的点和 Structure Element 是否匹配，除非全部匹配，否则就将值为 255 的中心点置为 0。主要是注意一下 Structure Element 的 index 和原始图像的 index 的关系，因为笔者是从 Structure Element 的中心定义为(0,0)，且规定 Structure Element 是奇数边长的正方形，实际上对于任意形状的 Structure Element 都可以用边长为奇数的正方形来代替，只要将其中需要的部分置为 1 即可。

```
BYTE *Erosion(BYTE *Gray, BYTE *structureElement, int sHeight, int sWidth)
{
    int i, j, m, n;
    BYTE *ErosionGray = (BYTE*)malloc(sizeof(BYTE) * bmpInfo.biWidth * bmpInfo.biHeight);
    memcpy(ErosionGray, Gray, sizeof(BYTE) * bmpInfo.biWidth * bmpInfo.biHeight);
    for (i = 0; i < bmpInfo.biHeight; i++)
    {
        for (j = 0; j < bmpInfo.biWidth; j++)
        {
            if (ErosionGray[i * bmpInfo.biWidth + j] == 0)
                continue;
            for (m = - sHeight / 2; m <= sHeight / 2; m++)
            {
                for (n = -sWidth / 2; n <= sWidth / 2; n++)
                {
                    if (i + m < 0 || i + m >= bmpInfo.biHeight || j + n < 0 || j + n >= bmpInfo.biWidth)
                        continue;
                    if (Gray[(i + m) * bmpInfo.biWidth + j + n] == 0 && structureElement[(sHeight / 2 + m) * sWidth + (sWidth / 2 + n)])
                        ErosionGray[i * bmpInfo.biWidth + j] = 0;
                }
            }
        }
    }
    return ErosionGray;
}
```

膨胀算法是类似的，只是笔者判断只要满足有一个点匹配上（1 对应 1），就将中心点置为 255。

```
BYTE *Dilation(BYTE *Gray, BYTE *structureElement, int sHeight, int sWidth)
{
    int i, j, m, n;
    BYTE *DilationGray = (BYTE*)malloc(sizeof(BYTE) * bmpInfo.biWidth * bmpInfo.biHeight);
    memcpy(DilationGray, Gray, sizeof(BYTE) * bmpInfo.biWidth * bmpInfo.biHeight);
    for (i = 0; i < bmpInfo.biHeight; i++)
    {
        for (j = 0; j < bmpInfo.biWidth; j++)
        {
            if (DilationGray[i * bmpInfo.biWidth + j] == 255)
                continue;
            for (m = - sHeight / 2; m <= sHeight / 2; m++)
            {
                for (n = -sWidth / 2; n <= sWidth / 2; n++)
                {
                    if (i + m < 0 || i + m >= bmpInfo.biHeight || j + n < 0 || j + n >= bmpInfo.biWidth)
                        continue;
                    if (Gray[(i + m) * bmpInfo.biWidth + j + n] == 255 && structureElement[(sHeight / 2 + m) * sWidth + (sWidth / 2 + n)])
                        DilationGray[i * bmpInfo.biWidth + j] = 255;
                }
            }
        }
    }
    return DilationGray;
}
```

开只需要先调用腐蚀再调用膨胀，闭只需要先调用膨胀再调用腐蚀即可。

```c
BYTE *Opening(BYTE *Gray, BYTE *structureElement, int sHeight, int sWidth)
{
    int i, j, m, n;
    BYTE *ErosionGray = Erosion(Gray, structureElement, sHeight, sWidth);
    BYTE *OpeningGray = Dilation(ErosionGray, structureElement, sHeight, sWidth);
    free(ErosionGray);
    return OpeningGray;
}

BYTE *Closing(BYTE *Gray, BYTE *structureElement, int sHeight, int sWidth)
{
    int i, j, m, n;
    BYTE *DilationGray = Dilation(Gray, structureElement, sHeight, sWidth);
    BYTE *ClosingGray = Erosion(DilationGray, structureElement, sHeight, sWidth);
    free(DilationGray);
    return ClosingGray;
}
```

四、实验环境及运行方法

编译环境：

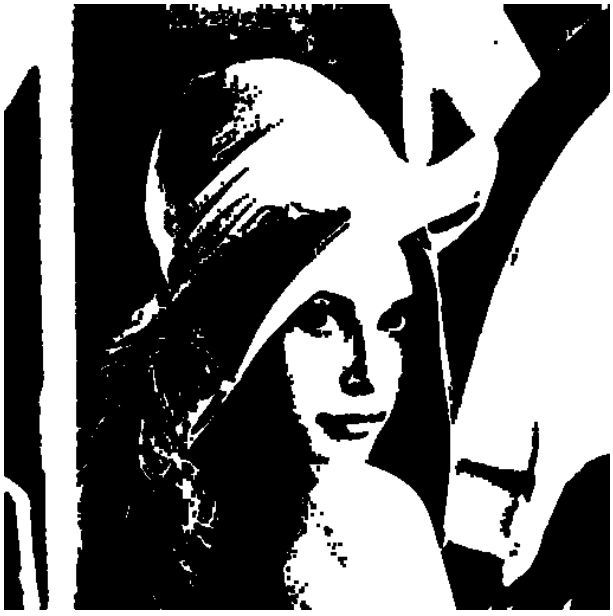
gcc 6.3.0、Windows 11 Insider Preview 22483.1011

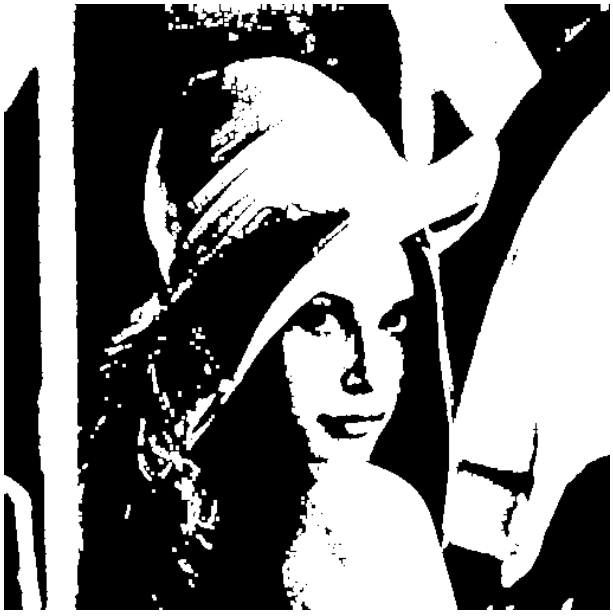
测试方法：

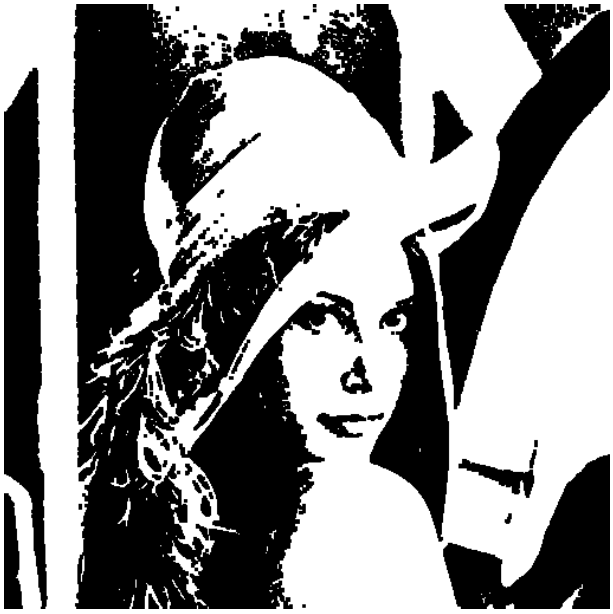
在命令行中输入 gcc gray.c main.c，然后运行.\a.exe。本输入图片为 24 位 BMP 文件，文件名为 input.bmp（放在源代码同一级目录中），输出的图片有 8 位灰度图 outputG.bmp、二值化图片 outputB.bmp、腐蚀图片 outputE.bmp、膨胀图片 outputD.bmp、开图片 outputO.bmp 和闭图片 outputC.bmp。要修改 Structure Element，可以修改 mian.c 文件中相应数组的 01 值以及数组的大小，并同时修改 4 个二值化操作函数后面的两个参数，分别是 Structure Element 的行列数，注意 Structure Element 在本程序中设定为奇数边长。

五、实验结果展示

24 位原图	
8 位灰度图	

二值化	
腐蚀	

膨胀	
开	

闭	

六、心得体会

在本次实验中我对于 BMP 文件的读取有了更深刻的认识，对二值化和四种操作也有了了解和实践经验。在本次实验中也遇到了诸多问题，比如结构体的内存对齐问题、实现大津算法时超出 int 和 double 存储范围等，可以说经过本次实验对 C 语言也有了更多的认识。