专业：计算机科学与技术
姓名：林海滨
学号：3220103645
日期：2023/10/31

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：bmp 图像的二值化以及膨胀、侵蚀、张开、合拢基本操作

一、实验目的和要求

通过项目实践，了解 bmp 图像的二值化，并理解如何使用大津算法来寻找前景和背景的阈值。另外，通过使用小样本来进行 Dilation、Erosion、Opening、Closing 四个基本操作，理解图像处理基本操作的本质以及作用。

本实验要求编写一个程序，读入一个 bmp 图像，通过大津算法来对这个图像进行二值化，然后进行输出。同时，对图像分别进行四大基本操作并分别输出。

二、实验内容和原理

1. 二值化

二值化就是将图片转换成仅黑和白两个单颜色，因此通常通过亮度来判断这个像素到底是黑的还是白的：越亮，则越偏向白色；越暗，则越偏向黑色。

2. 大津算法

我们需要定义什么叫亮、什么叫暗，所以我们需要一个清晰的阈值，看某个像素超过或低于这个阈值来判断它为黑还是白，这对计算机是非常简单的。问题是如何去寻找这个阈值。大津算法使用了一种量化的方式来判断，也就是通过取阈值，使得前景和背景的方差达到最大，那么这个阈值是最优的。证明附后。

但是，没有经过优化的大津算法对细节的处理比较差的。设想在一个明暗差距较大的图中使用大津算法，那么明部大量的细节以及暗部的大量细节会因为一刀切的阈值被全部忽视。解决这个问题需要使用分块的思想，将图片分解为一个个小块，那么每个块就可以有不一样的阈值，就可以体现更多的细节了。

然而，这样的分块可能导致明显的断层错误，要解决这个问题可以采用两个方式，一种就是将每个分块进行一定的重叠，重叠像素的阈值总体做一个平均，这就相当于为阈值做了

一个平滑操作。还有一种方法就是将局部的阈值和总体的阈值做一定比例线性的合成，得到一个最终的阈值，这可以在保留部分细节的情况下，平衡整个图片的整体性。

3. 膨胀、侵蚀、张开、合拢基本操作

这四个操作本质上是用一个非常小的像素矩阵对原来的图像位图矩阵进行一些操作。在本次实验中选择了 3x3 的小十字架范围进行操作。膨胀也就是如果范围内存在前景，那么它将作为新图的前景；同理，侵蚀就是范围内存在背景，那么它将作为新图的背景。张开就是一次侵蚀和一次膨胀的复合，而合拢就是一次膨胀和一次侵蚀的复合。

大津算法推导：

让背景方差趋于小，让前景方差趋于小，让两者方差趋于大，

$\sigma^2_{整间} = 背景比例 \times \sigma^2_{背景} + 前景比例 \times \sigma^2_{前景}$ （类似匀胶）

$\sigma^2_{前后} + \sigma^2_{类间} = \sigma^2$

代入方差公式，这里我们使用 μ_0, σ_0 代表总的，μ_1, σ_1 代表前景，μ_2, σ_2 代表背景

N_0 N_1 N_2

$\sigma^2_{前后} = \sigma^2 - \sigma^2_{类间}$

$= \sigma^2 - \frac{N_1}{N_0}\sigma_1^2 - \frac{N_2}{N_0}\sigma_2^2$

$= \left(\frac{1}{N_0}\sum_{x,y}\left(f^2(x,y) - \mu^2\right)\right) - \frac{N_1}{N_0}\left(\frac{1}{N_1}\sum_{x,y}^{前景}\left(f^2(x,y) - \mu_1^2\right)\right) - \frac{N_2}{N_0}\left(\frac{1}{N_2}\sum_{x,y}^{背景}\left(f^2(x,y) - \mu_2^2\right)\right)$

$= \left(\frac{1}{N_0}\sum_{x,y} f^2(x,y) - \frac{1}{N_0}\sum_{x,y}^{前景} f^2(x,y) - \frac{1}{N_0}\sum_{x,y}^{背景} f^2(x,y)\right) - \mu^2 + \frac{N_1}{N_0}\mu_1^2 + \frac{N_2}{N_0}\mu_2^2$

$= 0$

$= -\mu^2 + \frac{N_1}{N_0}\mu_1^2 + \frac{N_2}{N_0}\mu_2^2$ 。

注意，由方差公式及平均值公式

$\sum_{x,y}[f(x,y) - \mu]^2 = \sum_{x,y} f^2(x,y) - 2\mu\sum_{x,y} f(x,y) + N_0\cdot\mu^2$

$= \sum_{x,y} f^2(x,y) - 2N_0\mu^2 + N_0\mu^2 = \sum_{x,y}(f^2(x,y) - \mu^2)$

由上，已知 $-\mu^2 + \frac{N_1}{N_0}\mu_1^2 + \frac{N_2}{N_0}\mu_2^2 = \sigma^2_{前后}$

$(\mu = \frac{N_1}{N_0}\mu_1 + \frac{N_2}{N_0}\mu_2)$

那么原式 $= \frac{N_1}{N_0}(\mu_1^2 + \mu^2) + \frac{N_2}{N_0}(\mu_1^2 + \mu^2) - 2\mu^2$

$= \frac{N_1}{N_0}(\mu_1^2 + \mu^2 - 2\mu\mu_1) + \frac{N_2}{N_0}(\mu_2^2 + \mu^2 - 2\mu\mu_2)$

$= \frac{N_1}{N_0}(\mu - \mu_1)^2 + \frac{N_2}{N_0}(\mu - \mu_2)^2$

$= \frac{N_1}{N_0}\left[\frac{N_2}{N_0}(\mu_2 - \mu_1)\right]^2 + \frac{N_2}{N_0}\left[\frac{N_1}{N_0}(\mu_1 - \mu_2)\right]^2$

$= \frac{N_1}{N_0}\frac{\mu_2}{N_0}(\mu_2 - \mu_1)^2\left(\frac{N_2}{N_0} + \frac{N_1}{N_0}\right)$

$= \frac{N_1 N_2}{N_0^2}(\mu_2 - \mu_1)^2$ 。

三、实验步骤与分析

我将这个程序分成以下步骤：

1. 二值化

(1) 读入文件头

```
// step1. read headers
ReadHeaders(BMP);
```

(2) 输出文件头

```
// step2. output headers
WriteHeaders(Output0);
```

(3) 读入像素并记录部分数据（亮度最值）

```
// step3. read pixels and record the minimum Y and the maximum Y
int MINY = 255, MAXY = 0;
long long N0 = BMPInfoHeader.biHeight * BMPInfoHeader.biWidth;
for (int i = 0; i < BMPInfoHeader.biHeight; i++)
    for (int j = 0; j < BMPInfoHeader.biWidth; j++)
    {
        Pixel pixel;
        fread(pixel.Bytes, sizeof(char), 3, BMP);
        swap(&pixel.Bytes[0], &pixel.Bytes[2]);
        pixel.ConvertYUV();
        if (floor(pixel.YUV[0]) < MINY)
            MINY = floor(pixel.YUV[0]);
        if (ceil(pixel.YUV[0]) > MAXY)
            MAXY = ceil(pixel.YUV[0]);
    }
    for (int j = BMPInfoHeader.biWidth * 3; j % 4 != 0; j++)
    {
        unsigned char buf;
        fread(&buf, sizeof(char), 1, BMP);
    }
```

(4) 再次读像素，遍历阈值，记录样本数和平均值

```
        // step 4. read the pixels again and determine N1,N2,AVG1,AVG2
        memset(N1, 0, sizeof(N1));
        memset(N2, 0, sizeof(N2));
        memset(AVG1, 0, sizeof(AVG1));
        memset(AVG2, 0, sizeof(AVG2));
        rewind(BMP);
        for (int i = 1; i <= BMPHeader.bfOffBits; i++)
        {
            unsigned char buf;
            fread(&buf, sizeof(char), 1, BMP);
        }
        for (int i = 0; i < BMPInfoHeader.biHeight; i++)
        {
            for (int j = 0; j < BMPInfoHeader.biWidth; j++)
            {
                Pixel pixel;
                fread(pixel.Bytes, sizeof(char), 3, BMP);
                swap(&pixel.Bytes[0], &pixel.Bytes[2]);
                pixel.ConvertYUV();
                for (int TH = MINY; TH <= MAXY; TH++)
                {
                    // brighter N1, darker N2.
                    if (pixel.YUV[0] > TH)
                    {
                        AVG1[TH] = (double)N1[TH] / (N1[TH] + 1) * AVG1[TH] + pixel.YUV[0]
/ (N1[TH] + 1);
                        ++N1[TH];
                    }
                    else
                    {
                        AVG2[TH] = (double)N2[TH] / (N2[TH] + 1) * AVG2[TH] + pixel.YUV[0]
/ (N2[TH] + 1);
                        ++N2[TH];
```

```
                }
            }
        }
        for (int j = BMPInfoHeader.biWidth * 3; j % 4 != 0; j++)
        {
            unsigned char buf;
            fread(&buf, sizeof(char), 1, BMP);
        }
    }
```

(5) 计算各个阈值的方差，得到最优阈值

```
    // step 5. calculate sigma of each threshold, choose the largest one as the opitimal threshold.
    int OptTH = MINY;
    double MaxSig = 0;
    for (int i = MINY; i <= MAXY; i++)
    {
        double Sigma = (double)N1[i] / N0 * N2[i] / N0 * (AVG1[i] - AVG2[i]) * (AVG1[i] - AVG2[i]);
        if (Sigma > MaxSig)
        {
            MaxSig = Sigma;
            OptTH = i;
        }
    }
```

(6) 再次读像素，通过和阈值比较进行二值化

(7) 输出二值化图片

```
// step 6. read the pixels again, turn it to YUV and set UV to both 0.

// step 7. Output the bitmap
rewind(BMP);
for (int i = 1; i <= BMPHeader.bfOffBits; i++)
{
    unsigned char buf;
    fread(&buf, sizeof(buf), 1, BMP);
}
for (int i = 0; i < BMPInfoHeader.biHeight; i++)
{
    for (int j = 0; j < BMPInfoHeader.biWidth; j++)
    {
        Pixel pixel;
        fread(pixel.Bytes, sizeof(char), 3, BMP);
        swap(&pixel.Bytes[0], &pixel.Bytes[2]);
        pixel.ConvertYUV();
        pixel.YUV[0] = (pixel.YUV[0] > OptTH) ? 255 : 0;
        pixel.YUV[1] = pixel.YUV[2] = 0;
        pixel.ConvertRGB();
        swap(&pixel.Bytes[0], &pixel.Bytes[2]);
        fwrite(pixel.Bytes, sizeof(char), 3, Output0);
    }
    for (int j = BMPInfoHeader.biWidth * 3; j % 4 != 0; j++)
    {
        unsigned char buf;
        fread(&buf, sizeof(char), 1, BMP);
        fwrite(&buf, sizeof(char), 1, Output0);
    }
}
```

2. 第一步 Dilation、Erosion

(1) 读入二值化图像备用

```
// Step1. read the basic info of binarized image
FILE *BInput = fopen(".\\Output0.bmp", "rb");
FILE *Output1 = fopen(".\\Output1.bmp", "wb+");
FILE *Output2 = fopen(".\\Output2.bmp", "wb+");
ReadHeaders(BInput);
WriteHeaders(Output1);
WriteHeaders(Output2);
for (int i = 55; i <= BMPHeader.bfOffBits; i++)
{
    unsigned char buf;
    fread(&buf, sizeof(char), 1, BInput);
    fwrite(&buf, sizeof(char), 1, Output1);
    fwrite(&buf, sizeof(char), 1, Output2);
}
```

(2) 对首行像素进行处理

```
// Step4. Reserve data on the first line.
for (int i = 0; i < BMPInfoHeader.biWidth; i++)
{
    fwrite((*VecMap[0])[i].Bytes, sizeof(char), 3, Output1);
    fwrite((*VecMap[0])[i].Bytes, sizeof(char), 3, Output2);
}
for (int i = BMPInfoHeader.biWidth * 3; i % 4 != 0; i++)
{
    unsigned char buf = 0;
    fwrite(&buf, sizeof(char), 1, Output1);
    fwrite(&buf, sizeof(char), 1, Output2);
}
```

(3) 分行读像素，为节省空间，使用三行循环读入，对首尾像素进行处理

(4) Dilation 和 Erosion 操作同时进行，样本为 3x3 小十字

```
// Step5. Begin dialation and erosion process while reading
for (int i = 0; i < BMPInfoHeader.biHeight - 2; i++)
{
    // Read a line.
    ReadLine(VecMap[2], BInput);
    // Reserve the first element.
    Pixel p;
    p = (*VecMap[1])[0];
    fwrite(p.Bytes, sizeof(char), 3, Output1);
    fwrite(p.Bytes, sizeof(char), 3, Output2);
    // Process in the mid.
    for (int j = 1; j <= BMPInfoHeader.biWidth - 2; j++)
    {
        int flag1 = 0, flag2 = 0;
        if ((*VecMap[0])[j].YUV[0] >= 128)
            flag1 = 1;
        else
            flag2 = 1;
        if ((*VecMap[2])[j].YUV[0] >= 128)
            flag1 = 1;
        else
            flag2 = 1;
        if ((*VecMap[1])[j - 1].YUV[0] >= 128)
            flag1 = 1;
        else
            flag2 = 1;
        if ((*VecMap[1])[j + 1].YUV[0] >= 128)
            flag1 = 1;
        else
            flag2 = 1;
        if ((*VecMap[1])[j].YUV[0] >= 128)
            flag1 = 1;
```

```
            else
                flag2 = 1;
            Pixel pixel;
            pixel.YUV[0] = flag1 ? 255 : 0;
            pixel.YUV[1] = pixel.YUV[2] = 0;
            pixel.ConvertRGB();
            swap(&pixel.Bytes[0], &pixel.Bytes[2]);
            fwrite(pixel.Bytes, sizeof(char), 3, Output1);
            pixel.YUV[0] = flag2 ? 0 : 255;
            pixel.YUV[1] = pixel.YUV[2] = 0;
            pixel.ConvertRGB();
            swap(&pixel.Bytes[0], &pixel.Bytes[2]);
            fwrite(pixel.Bytes, sizeof(char), 3, Output2);
        }
        // Reserve the last element.
        p = (*VecMap[1]).back();
        fwrite(p.Bytes, sizeof(char), 3, Output1);
        fwrite(p.Bytes, sizeof(char), 3, Output2);
        // zeros in the end.
        for (int j = BMPInfoHeader.biWidth * 3; j % 4 != 0; j++)
        {
            unsigned char buf = 0;
            fwrite(&buf, sizeof(char), 1, Output1);
            fwrite(&buf, sizeof(char), 1, Output2);
        }
        // clear the first Line.
        VecMap[0]->clear();
        // Reset the VecMap
        swapVec();
    }
    // Reserve the last line.
    for (int i = 0; i < BMPInfoHeader.biWidth; i++)
    {
        fwrite((*VecMap[2])[i].Bytes, sizeof(char), 3, Output1);
        fwrite((*VecMap[2])[i].Bytes, sizeof(char), 3, Output2);
    }
```

(5) 重置第一行并进行循环进程，改变对应下标

```
void swapVec()
{
    vector<Pixel> *buf;
    buf = VecMap[0];
    VecMap[0] = VecMap[1];
    VecMap[1] = buf;
    buf = VecMap[1];
    VecMap[1] = VecMap[2];
    VecMap[2] = buf;
    buf = VecMap2[0];
    VecMap2[0] = VecMap2[1];
    VecMap2[1] = buf;
    buf = VecMap2[1];
    VecMap2[1] = VecMap2[2];
    VecMap2[2] = buf;
}
```

(6) 对末行进行处理，并且凑整 4 的倍数

```
    for (int i = BMPInfoHeader.biWidth * 3; i % 4 != 0; i++)
    {
        unsigned char buf = 0;
        fwrite(&buf, sizeof(char), 1, Output1);
        fwrite(&buf, sizeof(char), 1, Output2);
    }
```

(7) 输出图像

3. 第二步 Dilation、Erosion（Opening 和 Closing 操作）

操作同上，不再赘述。

另外，我将分块的大津算法也进行了撰写，整体思路是使用滑动窗口进行处理。

1.读入一个 BMP

2.记录最小和最大的 Y（整体）

3.遍历像素，遍历 TH，记录 N1，N2，μ1，μ2.

4.计算 TH 对应的σ，比较得到最终整体阈值 TH0.

5.遍历左上角位置(0 ~ height - WindowSize, 0 ~ width - WindowSize).

(1)遍历像素，找出窗口内最大 Y 和最小 Y

(2)遍历像素，遍历 TH，记录 N1，N2，μ1，μ2.

(3)计算σ，得到子块的阈值 THx

(4)遍历像素，将 THx 和自有的 TH 做平均。

6.遍历像素，TH = k TH0 + (1-k) THx.判断黑白，进行二值化。

7.输出图像

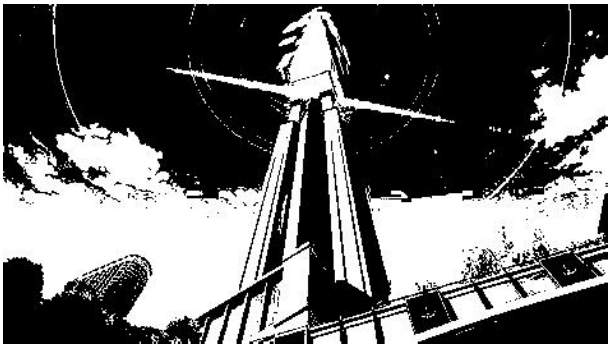
代码不再赘述。

四、实验环境及运行方法

该程序使用 vscode 软件编写完成，可以使用 MingW 进行编译后，在执行程序的文件夹中放入 input.bmp 文件，输出 output0.bmp，output2.bmp，output3.bmp，output4.bmp 五个文件，分别对应二值化、膨胀、侵蚀、合拢和张开四个基本操作。目前经测试可以在 windows 系统上稳定运行。

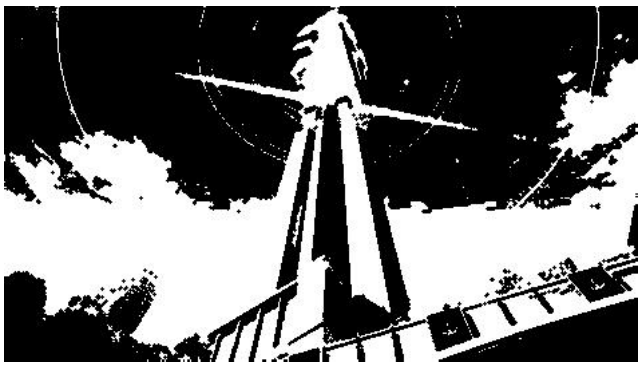
五、实验结果展示

原图片(带调色盘，但是没有使用)



二值化图片

After Dilation



After Closing

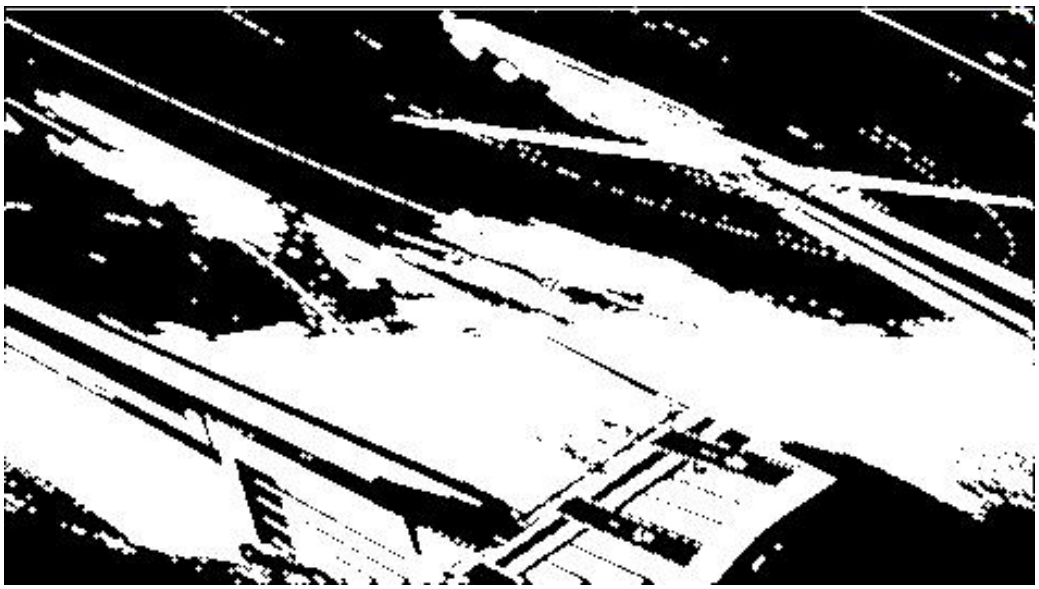
After Erosion

After Opening

六、心得体会

本次实验进行得非常顺利，这得益于非常清晰的分析规划，把大任务分解成小人物，因此难以出错。然而在实验当中仍然出现了两个 BUG。

第一个 BUG 导致了严重的图像错误。因为重复地使用了 ReadLine（读入一行像素的函数）和 fread（读入一个像素），导致像素读入发生了错误，于是图像被整体拉伸并产生了一些缺失。



还挺好看

另外，第二个 BUG 在于 swapVec 函数中，普通的交换流程有三步，少写了一步，导致循环数组中基本上全部变成了 Output1 中的像素，对图像的生成影响非常大，成像基本像对二值化图像做了两次 Dilation 操作。