**实验报告**

专业：计算机科学与技术

日期：2022/10/20

课程名称： 图像信息处理 指导老师： 宋明黎 成绩：

实验名称： 图像二值化与形态学操作

**一、实验目的和要求**

完成下述实验任务：

1.以二进制形式读取 .bmp文件头信息及数据信息

2.运用唯一阈值将 .bmp文件进行二值化

3.使用大津算法（整体与局部）将.bmp文件进行二值化

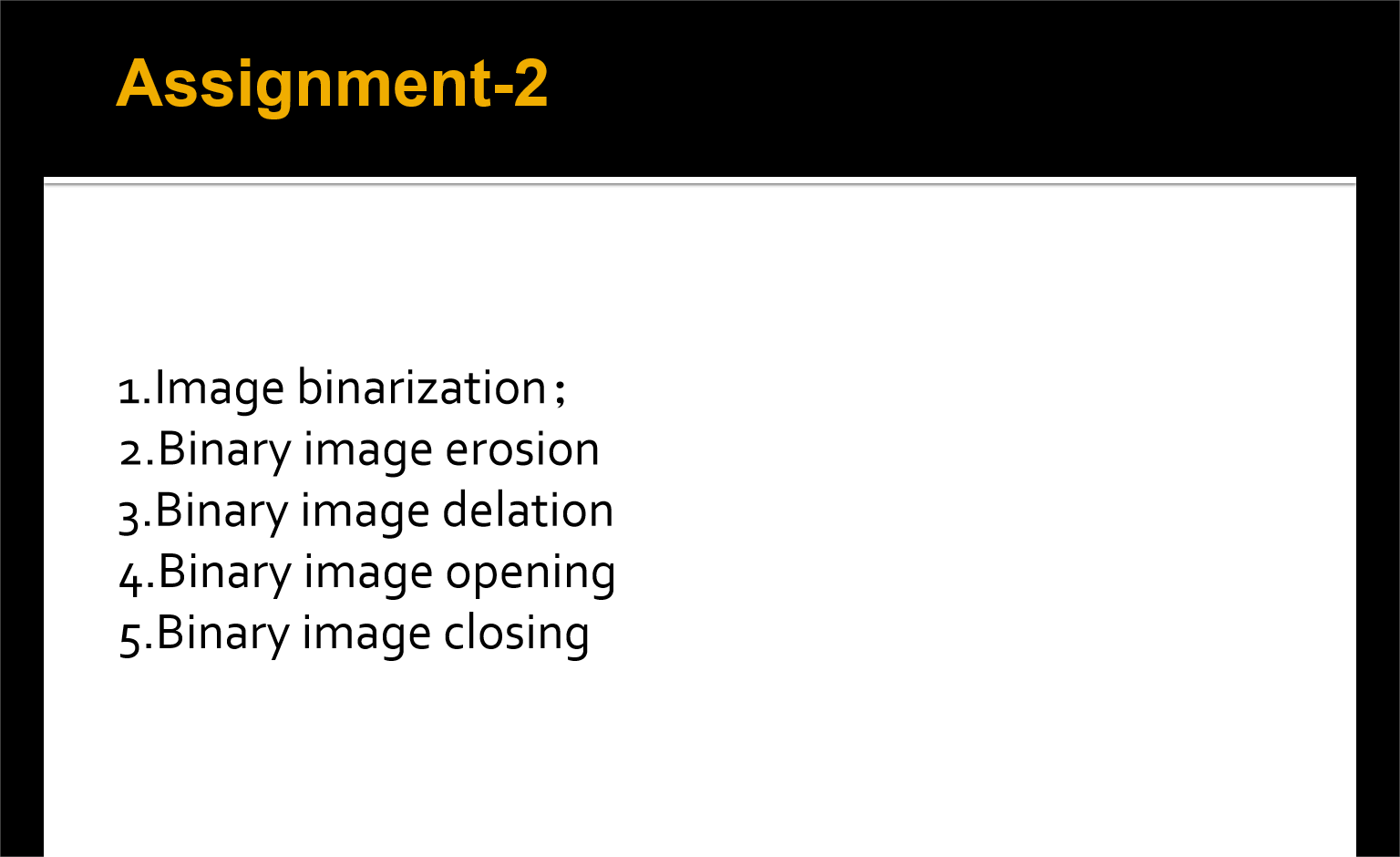
4. 进行形态学四项操作

4.1 膨胀

4.2 腐蚀

4.3 开操作

4.4 闭操作



**二、实验内容和原理**

.bmp图像信息的贮存，往往有固定的格式，用二进制的方式打开bmp文件，可以设置相对映的数据结构接受文件头部信息、数据头部信息以及palette信息与bitmap data，从而进行相应的数据操作；本实验选用256色（512\*512像素）灰度图片进行二值化与形态学操作，因而在bitmap data部分，每一字节只保存相应的palette索引值，若对索引值进行操作，可以等价为直接对“色”进行相应操作。

在二值化过程中，若采用唯一阈值，可能会随机放大部分噪声或忽略重要细节信息；所以采用大津算法计算.bmp文件全局或局部的最优阈值，再进行相应的二值化，从而可以得到细节更多的二值图片。

对于形态学操作而言，本实验选择了十字型膨胀与十字型腐蚀，可以得到较好的显示效果；最后进行开操作与闭操作的使用，在开操作过程中，因先腐蚀后膨胀，可以在尽可能保留.bmp前景面积的基础上，排除小噪声的干扰；而在闭操作过程中，因先膨胀后腐蚀，可以对一些背景小洞进行填补，增加.bmp文件前景的完整性。

**三、实验步骤与分析**

1.建立存储图像信息的数据结构（头文件）与bmp附带信息基本一一对应。

typedef unsigned char bit;

typedef unsigned short word;

typedef unsigned long dword;

struct tagBITMAPFILEHEADER

{

word bftype; //文件类型

dword bfsize; //文件大小（字节为单位）

word bfreserved1; //保留，必须设为0

word bfreserved2; //保留，必须设为0

dword bfoffbits; //到实际图像实际开始的偏移量

};

struct tagBITMAPINFOHEADER

{

dword bisize; //结构所需字节数

long biwidth; //图像宽度（像素为单位）

long biheight; //图像长度（像素为单位）--正值对映图像倒立

word biplanes; //位面数，常为1

word bibitcounts; //比特/像素比

dword bicompression; //压缩类型

dword bisizeimage; //图像大小

long bixpelspermeter; //水平分辨率

long biypelspermeter; //垂直分辨率

dword biclrused; //颜色索引数

dword biclrimportant; //重要影响颜色索引数

};

struct PALETTE

{

bit R; //存贮调色板中分别索引下的RGB值

bit G;

bit B;

bit reserved ; //保留值为必须为0

};

1. 声明本实验所需使用函数（具体解析见下）

//函数声明

//写入文件函数

void write(tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information, PALETTE\* rgblib,

bit\* change, long int number, const char name[]);

//二值化转化函数

void binary(bit\* firstchange, long int number, int threshold);

//大津算法求得最佳阈值

int otsu(bit\* secondchange, long int number);

//局部大津算法求分别的最佳阈值

void partotsu(bit\* thirdchange, int height, int width, int n);

//二值化后进行膨胀操作

void dilation(bit\* forthchange, int height, int width, int value);

//二值化后进行腐蚀操作

void erosion(bit\* fifthchange, int height, int width, int value);

//二值化后进行开操作

void open(bit\* sixthchange, int height, int width, int value);

//二值化后进行闭操作

void close(bit\* seventhchange, int height, int width, int value);

1. 二进制打开.bmp文件，读取文件内数据信息，构建256色颜色索引表，记录bitmap data区域对映像素索引矩阵。

//如果图片被其他程序占用，打开异常，直接返回-1

if (fp == NULL)

return -1;

fread(&(file.bftype), 1, sizeof(word), fp);

fread(&(file.bfsize), 1, sizeof(dword), fp);

fread(&(file.bfreserved1), 1, sizeof(word), fp);

fread(&(file.bfreserved2), 1, sizeof(word), fp);

fread(&(file.bfoffbits), 1, sizeof(dword), fp);

fread((&information), sizeof(tagBITMAPINFOHEADER), 1, fp);

//获取bmp大小像素信息

picwidth = information.biwidth;

picheight = information.biheight;

//获取调色板索引表---256色位图

PALETTE\* rgblib = (PALETTE\*)malloc(sizeof(PALETTE) \* 256);

fread(rgblib, 256 \* sizeof(PALETTE), 1, fp);

//读入bmp中bitmap data的索引值信息

bit\* rgbgray = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

fread(rgbgray, picwidth \* picheight \* sizeof(bit), 1, fp);

4.1选用唯一阈值进行bmp图像二值化

//唯一阈值二值化

bit\* firstchange = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

for (i = 0;i < picheight \* picwidth;i++)

firstchange[i] = rgbgray[i];

//以0-255均值127作为唯一阈值

//threshold = 127;

//因为全局大津算法最终结果为124，与0-255均值127相差较近；为凸显不同方法的运算效果不同，设定唯一阈值为100

threshold = 100;

//二值化后写入文件并输出

binary(firstchange, picheight \* picwidth, threshold);

write(file, information, rgblib, firstchange, picheight \* picwidth, "only threshold.bmp");

free(firstchange);

4.2运用大津算法进行全局最优阈值寻找，再进行bmp文件二值化

//全局二值化

bit\* secondchange = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

for (i = 0;i < picheight \* picwidth;i++)

secondchange[i] = rgbgray[i];

//通过大津算法获得最优阈值

threshold = otsu(secondchange, picheight \* picwidth);

//二值化后写入文件并输出

binary(secondchange, picheight \* picwidth, threshold);

write(file, information, rgblib, secondchange, picheight \* picwidth, "total otsu.bmp");

free(secondchange);

4.2.1 大津算法

输入值：1.需寻找最优阈值的字典索引矩阵

2.矩阵总大小

返回值：该矩阵分布中最优阈值字典索引值

//大津算法

int otsu(bit\* secondchange, long int number)

{

int i,j,index;

int max=0, min=255;

int numdown = 0, numup = 0;

double sumdown=0, sumup=0;

double result,remain = 0;

double avgup,avgdown,avg;

//遍历bitmap data，寻得bmp文件最大最小字典值

for (i = 0;i < number;i++)

{

if (secondchange[i] < min)

min = secondchange[i];

if (secondchange[i] > max)

max = secondchange[i];

}

index = min;

//输出最小字典值，用于检验

//printf("%d", index);

//遍历最小至最大字典值，寻得 k 使得大津算法结果最大化

for (i = min;i <= max;i++)

{

numdown = 0;

sumdown = 0;

numup = 0;

sumup = 0;

for (j = 0;j < number;j++)

{

if (secondchange[j] > i)

{

//大于指定字典值，增加up比列

numup++;

//累加大于字典总值

sumup = sumup + secondchange[j];

}

else

{

//小于指定字典值，增加down比列

numdown++;

//累加小于字典总值

sumdown = sumdown + secondchange[j];

}

}

//计算指定字典值下大津算法结果

avgup = numup \* 1.0 / number;

avgdown = numdown \* 1.0 / number;

avg = sumup \* 1.0 / numup - sumdown \* 1.0 / numdown;

result = avgup \* avgdown \* pow(avg, 2);

//更新最优解与最优字典值

if (result >= remain)

{

index = i;

remain = result;

}

}

//用于检验程序正确性

//printf("%d", index);

return index;

}

4.2.2二值化过程

输入值：1.需进行二值化的字典索引值记录矩阵

2.上述矩阵总大小

3.二值化分界阈值

//二值化函数

void binary(bit\* firstchange, long int number, int threshold)

{

long int i;

for (i = 0;i < number;i++)

{

if (firstchange[i] > threshold)

//大于阈值，保留为黑色

firstchange[i] = 255;

else

//小于阈值，保留为白色

firstchange[i] = 0;

}

}

4.3运用大津算法进行局部最优阈值寻找，再局部进行bmp文件二值化

//局部二值化

bit\* thirdchange = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

for (i = 0;i < picheight \* picwidth;i++)

thirdchange[i] = rgbgray[i];

//对图像进行分块，使用大津算法寻得每块最优阈值并进行相应的二极化

partotsu(thirdchange, picheight, picwidth,128 );

write(file, information, rgblib, thirdchange, picheight \* picwidth, "part otsu.bmp");

free(thirdchange);

4.3.1局部大津算法

输入值：1.需寻找最优阈值的字典索引矩阵

2.bmp文件总行像素

3.bmp文件总列像素

4.所划分块行（列）像素

//局部大津算法，以n\*n为最小单元

void partotsu(bit\* thirdchange, int height, int width, int n)

{

bit\* data = (bit\*)malloc(sizeof(bit) \* n \* n);

int i,j,x,y;

int start;

int threshold;

//将bmp文件总体进项分块

for (i = 0;i < height / n;i++)

{

for (j = 0;j < width / n;j++)

{

//重置当前区块初始位置

start = n \* width \* i + j \* n;

for (x = 0;x < n;x++)

{

//区块赋值

for (y = 0;y < n ;y++)

data[x \* n + y] = thirdchange[start + width \* x + y];

}

//寻得局部最优阈值，并进行相应二极化

threshold = otsu(data, n \* n);

binary(data, n \* n, threshold);

for (x = 0;x < n;x++)

{

for (y = 0;y < n;y++)

//返回重置后结果

thirdchange[start + width \* x + y] = data[n \* x + y];

}

}

}

}

4.4进行形态学操作

//膨胀操作

bit\* forthchange = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

for (i = 0;i < picheight \* picwidth;i++)

forthchange[i] = rgbgray[i];

//先对bmp文件进行全局大津算法二极化

threshold = otsu(forthchange, picheight \* picwidth);

binary(forthchange, picheight \* picwidth, threshold);

dilation(forthchange, picheight, picwidth, 0);

write(file, information, rgblib, forthchange, picheight \* picwidth, "dilation.bmp");

free(forthchange);

//腐蚀操作

bit\* fifthchange = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

for (i = 0;i < picheight \* picwidth;i++)

fifthchange[i] = rgbgray[i];

//先对bmp文件进行全局大津算法二极化

threshold = otsu(fifthchange, picheight \* picwidth);

binary(fifthchange, picheight \* picwidth, threshold);

erosion(fifthchange, picheight, picwidth, 0);

write(file, information, rgblib, fifthchange, picheight \* picwidth, "erosion.bmp");

free(fifthchange);

//开操作

bit\* sixthchange = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

for (i = 0;i < picheight \* picwidth;i++)

sixthchange[i] = rgbgray[i];

// 先对bmp文件进行全局大津算法二极化

threshold = otsu(sixthchange, picheight \* picwidth);

binary(sixthchange, picheight \* picwidth, threshold);

open(sixthchange, picheight, picwidth, 0);

write(file, information, rgblib, sixthchange, picheight \* picwidth, "open.bmp");

free(sixthchange);

//闭操作

bit\* seventhchange = (bit\*)malloc(sizeof(bit) \* picwidth \* picheight);

for (i = 0;i < picheight \* picwidth;i++)

seventhchange[i] = rgbgray[i];

// 先对bmp文件进行全局大津算法二极化

threshold = otsu(seventhchange, picheight \* picwidth);

binary(seventhchange, picheight \* picwidth, threshold);

close(seventhchange, picheight, picwidth, 0);

write(file, information, rgblib, seventhchange, picheight \* picwidth, "close.bmp");

free(seventhchange);

4.4.1十字型膨胀操作

输入值：1.所需膨胀区域字典索引值矩阵

2.膨胀图像的像素总行数

3.膨胀图像的像素总列数

4.十字型膨胀标准值（前景/背景）

//十字形膨胀操作

void dilation(bit\* forthchange, int height, int width, int value)

{

int i, j;

int start;

bit\* data = (bit\*)malloc(sizeof(bit) \* height \* width);

for (i = 0;i < height \* width;i++)

data[i] = forthchange[i];

for (i = 1;i < height - 1;i++)

{

for (j = 1;j < width - 1;j++)

{

start = i \* width + j;

//若像素块本身为黑，保留前景

if (forthchange[start] == value)

continue;

//若像素块周围存在前景像素，将此像素置为前景

else if (data[start - 1] == value || data[start + 1] == value || data[start + wid th] == value || data[start - width] == value)

forthchange[start] = value;

}

}

//第一行前景重置

for (i = 1;i < width - 1;i++)

{

if (forthchange[i] == value)

continue;

else if (data[i - 1] == value || data[i + 1] == value || data[i + width] == value)

forthchange[i] = value;

}

//最后一行前景重置

for (i = 1;i < width - 1;i++)

{

j = (height - 1) \* width + i;

if (forthchange[j] == value)

continue;

else if (data[j - 1] == value || data[j + 1] == value || data[j - width] == value)

forthchange[j] = value;

}

//第一列前景重复

for (i = 1;i < height - 1;i++)

{

j = i \* width;

if (forthchange[j] == value)

continue;

else if (data[j + 1] == value || data[j - width] ==value || data[j + width]==value)

forthchange[j] = value;

}

//最后一列前景重置

for (i = 1;i < height - 1;i++)

{

j = (i + 1) \* width - 1;

if (forthchange[j] == value)

continue;

else if (data[j - 1]==value || data[j - width] ==value || data[j + width] == value)

forthchange[j] = value;

}

//四角前景重置

//左上

if (forthchange[0] == value);

else if (data[width] == value || data[1] == value)

forthchange[0] == value;

//右上

if (forthchange[width - 1] == value);

else if (data[width - 2] == value || data[width \* 2 - 1] == value)

forthchange[width - 1] == value;

//左下

j = (height - 1) \* width;

if (forthchange[j] == value);

else if (data[j + 1] == value || data[j - width] == value)

forthchange[0] == value;

//右下

j = height \* width - 1;

if (forthchange[j] == value);

else if (data[j-1] == value || data[j-width] == value)

forthchange[j] == value;

free(data);

}

4.4.2十字型腐蚀

输入值：1.所需膨胀区域字典索引值矩阵

2.膨胀图像的像素总行数

3.膨胀图像的像素总列数

4.十字型膨胀标准值（前景/背景）

//十字型腐蚀操作

void erosion(bit\* fifthchange, int height, int width, int value)

{

if (value == 0)

value = 255;

else

value = 0;

//等价于背景膨胀

dilation(fifthchange, height, width, value);

}

4.4.3开操作

输入值：1.所需膨胀区域字典索引值矩阵

2.膨胀图像的像素总行数

3.膨胀图像的像素总列数

4.十字型膨胀标准值（前景/背景）

//开操作，先腐蚀后膨胀

void open(bit\* sixthchange, int height, int width, int value)

{

erosion(sixthchange, height, width, value);

dilation(sixthchange, height, width, value);

}

4.4.4闭操作

输入值：1.所需膨胀区域字典索引值矩阵

2.膨胀图像的像素总行数

3.膨胀图像的像素总列数

4.十字型膨胀标准值（前景/背景）

//闭操作,先膨胀后腐蚀

void close(bit\* seventhchange, int height, int width, int value)

{

dilation(seventhchange, height, width, value);

erosion(seventhchange, height, width, value);

}

4.5写入函数

输入值：1.bmp文件头文件

2.bmp文件数据头文件

3.bmp文件字典索引表

4.bmp文件字典值记录表

5.bmp文件总像素大小（行数乘以列数）

6.bmp文件名称

//写入文件函数

void write(tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information,PALETTE\* rgblib, bit\* change, long int number, const char name[])

{

FILE\* fp;

//打开/新建文件

fp = fopen(name, "wb");

//写入bmp图像各类数据

fwrite(&(file.bftype), 1, sizeof(file.bftype), fp);

fwrite(&(file.bfsize), 1, sizeof(file.bfsize), fp);

fwrite(&(file.bfreserved1), 1, sizeof(file.bfreserved1), fp);

fwrite(&(file.bfreserved2), 1, sizeof(file.bfreserved2), fp);

fwrite(&(file.bfoffbits), 1, sizeof(file.bfoffbits), fp);

fwrite(&information, 1, sizeof(information), fp);

fwrite(rgblib, 1, sizeof(PALETTE) \* 256 , fp);

fwrite(change, 1, sizeof(bit) \* number, fp);

//关闭bmp图像文件

fclose(fp);

}

**四、实验环境及运行方法**

用C语言编写，vs正常编译环境运行，其中需要以下语句忽略vs对fopen函数的安全性中断。

#pragma warning(disable:4996) //忽略vs对fopen的安全性错误

对于实验结果，点击运行即可。在调试过程（先以进行注释）中，会通过printf输出函数检测使用大津算法时寻得的全局（局部）最优阈值，若程序运行正确，程序返回0；若bmp文件已被其他程序调用或打开失败，程序返回-1。

最后输出七张图片，分别为制定阈值二值化bmp文件、全局大津算法二值化bmp文件、局部大津算法二值化bmp文件、十字型膨胀二值化bmp文件，十字型腐蚀二值化bmp文件、开操作二值化bmp文件和闭操作bmp文件。

**五、实验结果展示**

****

原始图片 指定阈值输出图片

****

全局大津算法输出图片 局部大津算法输出图片

****

膨胀操作输出结果 腐蚀操作输出结果

****

开操作输出结果 闭操作输出结果

1. **心得体会**

在学会二进制读取bmp文件的各项数据后，对数据进行各类操作终于变得直观，但不得不感叹课堂与真正的实践实现还是相去甚远，像二值化bmp文件以及四个形态学操作，颇具挑战性，但实现之后确为收获满满。

这次的实验目标，我选用了256色的（512\*512）灰度图像文件，相较于使用16位色图的project 1，这次的实验过程中需要考虑到palette的读取，虽然没有应用；但bitmap data的索引值贮存方式，在大津算法求取最优阈值时，也提供了很多便利，无需考虑RGB对映的实际值。

可能0-255的均数与大津算法所求得的最优阈值相差较小，使得输出图片的前景间没有较大的差别，但对于开操作和闭操作而言，膨胀或腐蚀的先后使用，确实对bmp文件造成了不同的影响，或是排除了噪声，或是填补了空洞。

比较遗憾的是，在使用局部大津算法时，如果只是简单对全局进行划分，最后的part 都otsu输出图片还是有一些小bug，因为相邻划分不同的最优阈值，可能会导致连续部分图片的灰度值产生255->0或0->255的跃变，从而产生一些奇怪的前景形状布局。