**实验报告**

专业：计算机科学与技术

姓名：张三

学号：1111111

日期：2021/10/20

课程名称： 图像信息处理 指导老师： 成绩：

实验名称： 切块运用大津算法实现二值化及四种形态学操作

1. **实验目的和要求**

1实验目的

1.1实现彩色bmp图片的二值化操作，二值化后的图片便于进行各种形态学操作。

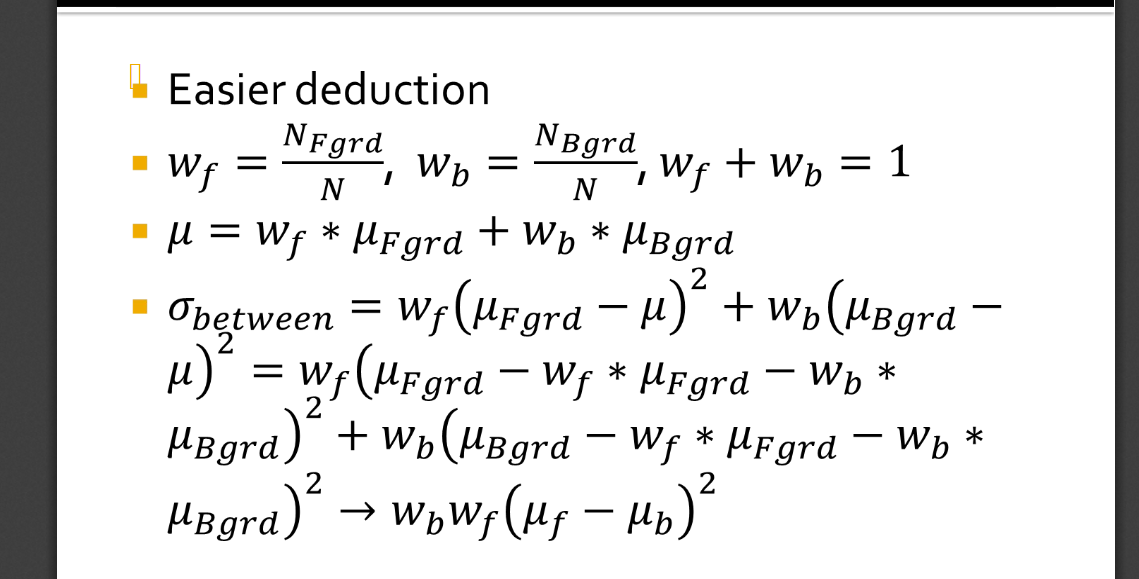
1.2在生成二值化图片上进行四种形态学操作，观察生成的新的图片与与原图片有哪些差别

2实验要求

2.1用C语言或者C++实现

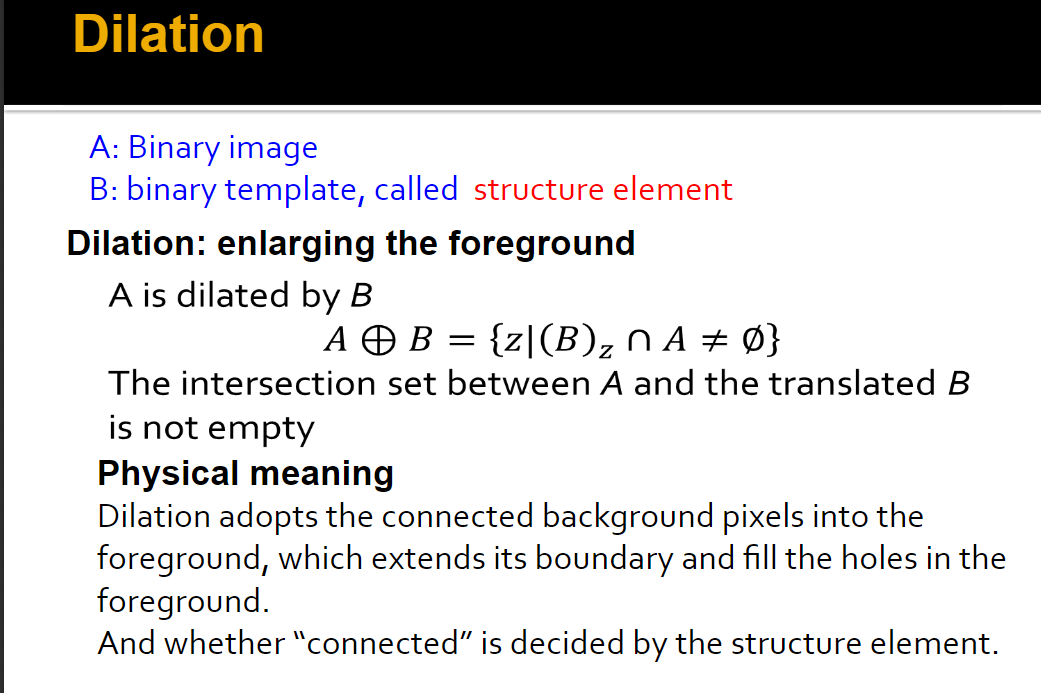
**二、实验内容和原理**

1大津算法设置二值化阈值

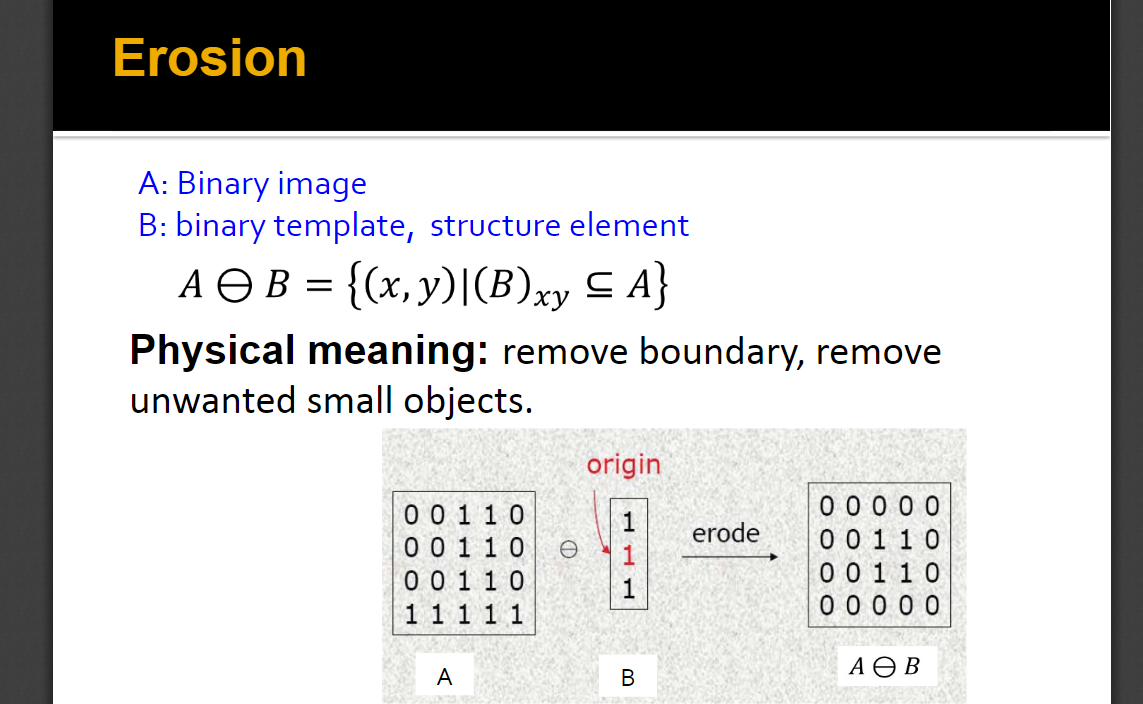


如图，首先找到要进行二值化的范围内图片的灰度范围，从最低的灰度值向最高遍历，对于每一个阈值，计算被视作前景的点的数量和灰度平均值、被视作背景的点的数量和灰度的平均值，并代入计算，得到一个衡量该阈值适用程度的值，以该值最大的那个点作为最二值化的阈值。

2膨胀

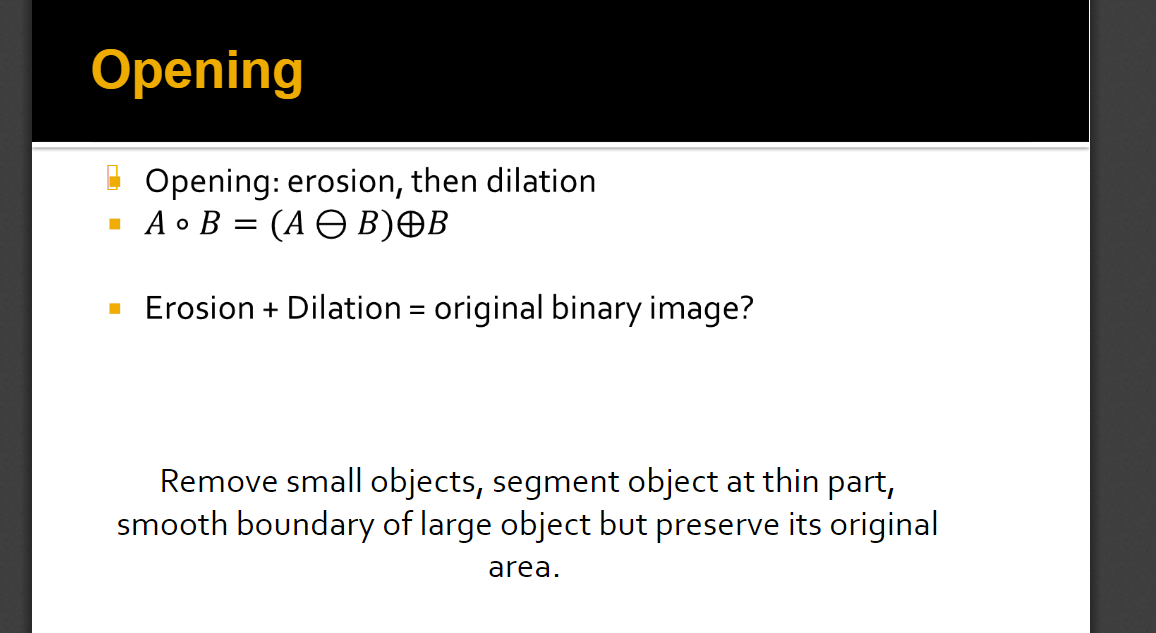
用一个设定了原点的结构体扫描图片，结构体一旦与前景相交，就把原点设置为前景，否则设置为背景，原点遍历不到的地方保持原值。

3腐蚀



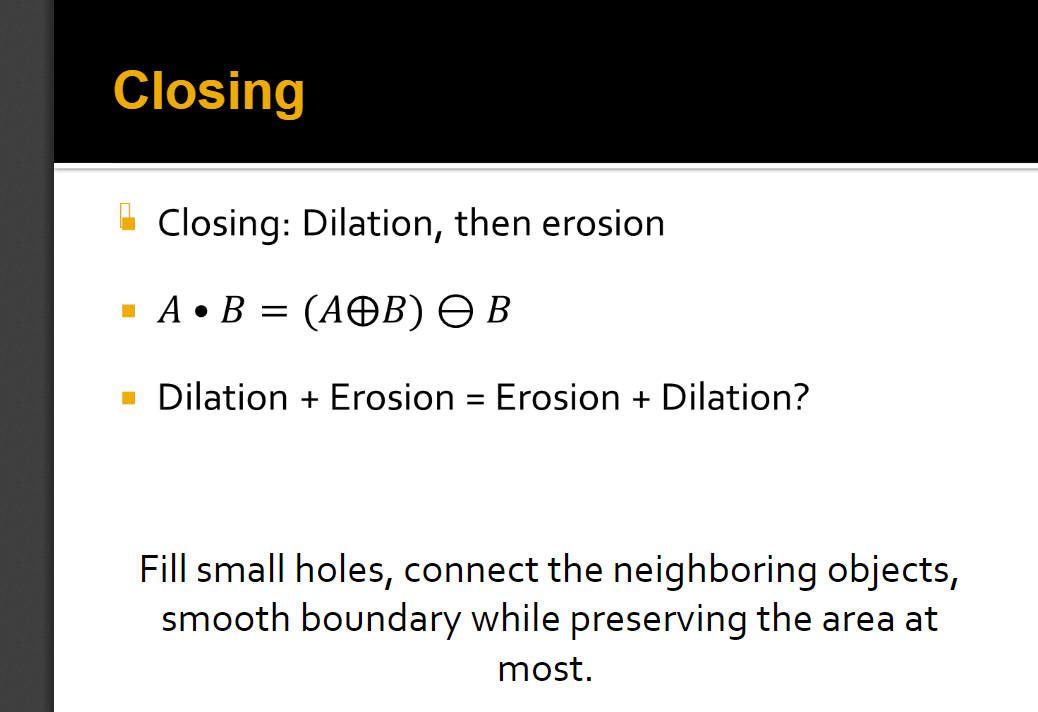
利用设置好中心点的结构体扫面图片，一旦结构体与背景相交，就把原点设置为0，否则设置为0 原点扫描不到的地方保持原值

4打开



先腐蚀再膨胀，可以去除小噪点

5关闭



先膨胀再腐蚀，可以连接分离的物体

**三、实验步骤与分析**

//include

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

//declare

#define DEVIDE 8 //this defines how many pieces the picture will be cutted into ,which is followed by binarilazing

#define SCALE 9 //the area of the structuring element matrix

typedef struct BMP{

    FILE\* file;

    BITMAPFILEHEADER bf;

    BITMAPINFOHEADER bi;

} BMP;

typedef struct MIN\_MAX{

    int min;

    int max;

}MIN\_MAX; // to store the max and min num of gray number

typedef struct THR{

    int threshold;

    double sigma;

}THR; // to store each threshold together with sigma

int write\_data(BMP bmp,unsigned char\* black\_white);//write the prepared bmp data into file

int test\_overlap\_0(unsigned char \*m1,unsigned char \*m2,int row, int column,int off1, int off2);// to check if the 0s of picture has overlapped the structure element

int test\_overlap\_1(unsigned char\* m1,unsigned char\*m2,int row,int column,int off1,int off2);//to check if the 1s of the picture has overlapped the structure element

int initial\_bmp(BMP);//write the file header into the file

int erosion(unsigned char\* eromatrix,unsigned char \* black\_white,unsigned char \* structure\_element,BMP erobmp);

int delasion(unsigned char\* delmatrix,unsigned char \* black\_white,unsigned char \* structure\_element,BMP delbmp);

BMP create\_bmp();//read the first bmp in D://test.bmp

MIN\_MAX min\_max(unsigned char \*,int);//find the max and min bound of the array

int fill\_gray\_array(BMP,unsigned char\*);//read bmp data from the 24color file and transfer it into gray number array

int write\_gray\_palette(BMP);//write gray palette into the file

unsigned char\* copy\_matrix(unsigned char \*,int ,int,int);// copy part of a big matrix and store it into a small matrix

int transfer(unsigned char \*,int,int,int,int,unsigned char \*,BMP );//transfer the gray number array into binary number array

int main(){

    BMP rawbmp = create\_bmp();

    unsigned char gray[rawbmp.bi.biWidth \* rawbmp.bi.biHeight];// to store the gray number for gray bmp

    fill\_gray\_array(rawbmp,gray);//fill in the array

    FILE \* bifile = fopen("D:\\binary.bmp","wb");

    BMP bibmp={bifile,rawbmp.bf,rawbmp.bi};

    bibmp.bf.bfOffBits = 14 + 40 + 256\*sizeof(RGBQUAD);//modify the file header for new file

    int new\_width = (bibmp.bi.biWidth +3)/4\*4;

    bibmp.bf.bfSize = bibmp.bf.bfOffBits + new\_width\*bibmp.bi.biHeight;

    bibmp.bi.biBitCount = 8;

    bibmp.bi.biSizeImage = new\_width\*bibmp.bi.biHeight;

    initial\_bmp(bibmp);

    write\_gray\_palette(bibmp);

    unsigned char black\_white[bibmp.bi.biWidth\*bibmp.bi.biHeight];//to store the binary number array

    int a=bibmp.bi.biWidth/DEVIDE, b=bibmp.bi.biHeight/DEVIDE; //cut the picture

    int x=-a;

    for(int i=0; i<DEVIDE;i++){

        x+=a;

        int y=-b;

        for(int j=0;j<DEVIDE;j++){

            y+=b;

            if(i==DEVIDE-1)//adjust the width and height if edge piece is met

                a+=bibmp.bi.biWidth%DEVIDE;

            if(j==DEVIDE-1)

                b+=bibmp.bi.biHeight%DEVIDE;

            unsigned char \* copymatrix=copy\_matrix(gray+y\*bibmp.bi.biWidth+x,a,b,bibmp.bi.biWidth);//copy one piece of the big matrix

            transfer(copymatrix,a,b,x,y,black\_white,bibmp);//transfer it into binary number and save in black\_white array

            a=bibmp.bi.biWidth/DEVIDE;//cover the adjustment

            b=bibmp.bi.biHeight/DEVIDE;

            free(copymatrix);

        }

    }

    write\_data(bibmp,black\_white);//the binarilized bmp is completed

    unsigned char structure\_element[SCALE] = {0,1,0,   1,1,1,   0,1,0};// the structure element

    FILE \* erofile = fopen("D:\\erosion.bmp","wb");

    BMP erobmp={erofile,bibmp.bf,bibmp.bi};

    initial\_bmp(erobmp);

    write\_gray\_palette(erobmp);

    unsigned char \*eromatrix = malloc(erobmp.bi.biWidth\*erobmp.bi.biHeight);// to store the binary number array of erosed bmp

    erosion(eromatrix,black\_white,structure\_element,erobmp);

    write\_data(erobmp,eromatrix);

    free(eromatrix);

    FILE \* delfile = fopen("D:\\delasion.bmp","wb");

    BMP delbmp={delfile,bibmp.bf,bibmp.bi};

    initial\_bmp(delbmp);

    write\_gray\_palette(delbmp);

    unsigned char \*delmatrix= malloc (delbmp.bi.biWidth\*delbmp.bi.biHeight);// to store the binary number array of delated file

    delasion(delmatrix,black\_white,structure\_element,delbmp);

    write\_data(delbmp,delmatrix);

    free(delmatrix);

    FILE \* openfile = fopen("D:\\open.bmp","wb");

    BMP openbmp={openfile,bibmp.bf,bibmp.bi};

    initial\_bmp(openbmp);

    write\_gray\_palette(openbmp);

    unsigned char \*openmatrix= malloc (openbmp.bi.biWidth\*openbmp.bi.biHeight);

    unsigned char \* tempmatrix = malloc(openbmp.bi.biWidth\*openbmp.bi.biHeight);

    erosion(tempmatrix,black\_white,structure\_element,openbmp);  //the erosed data is stored in tempmatrix

    delasion(openmatrix,tempmatrix,structure\_element,openbmp); // delate the erosed data

    write\_data(openbmp,openmatrix);

    free(openmatrix);

    FILE \* closefile = fopen("D:\\close.bmp","wb");

    BMP closebmp={closefile,bibmp.bf,bibmp.bi};

    initial\_bmp(closebmp);

    write\_gray\_palette(closebmp);

    unsigned char \*closematrix= malloc (closebmp.bi.biWidth\*closebmp.bi.biHeight);

    delasion(tempmatrix,black\_white,structure\_element,closebmp); // the delated data is stored in tempmatrix

    erosion(closematrix,tempmatrix,structure\_element,closebmp); // erose the delated data

    write\_data(closebmp,closematrix);

    free(closematrix);

    free(tempmatrix);

}

//function

unsigned char\* copy\_matrix(unsigned char \* gray, int a,int b,int width){

    unsigned char \* new =malloc(a\*b);

    for(int i=0;i<a;i++){

        for(int j=0; j<b;j++){

            \*(new+j\*a+i) = \*(gray+i+j\*width);

        }

    }

    return new;

}

int transfer(unsigned char \*copymatrix,int a,int b,int x,int y,unsigned char\*black\_white,BMP bibmp){

    MIN\_MAX range = min\_max(copymatrix,a\*b); //the range of thresholds

    THR best ={0,0};

    for(int i=range.min;i<=range.max;i++){

        THR temp;

        temp.threshold = i;

        double f\_num=0,b\_num=0,f\_sum=0,b\_sum=0;//count the data of the threshold

        for(int j=0;j<a\*b;j++){

            if (copymatrix[j]>i){

                b\_num ++;

                b\_sum += copymatrix[j];

            }else {

                f\_num ++;

                f\_sum += copymatrix[j];

            }

        }

    temp.sigma = b\_num\*f\_num/(b\_num+f\_num)/(b\_num+f\_num)\*(f\_sum/f\_num-b\_sum/b\_num)\*(f\_sum/f\_num-b\_sum/b\_num);// the current sigma

    if(temp.sigma > best.sigma)

        best = temp;

    }// best will store the max sigma

    black\_white += x + y\*bibmp.bi.biWidth;//change the matrix into binary number using the best threshold

    for(int i=0;i<a;i++){

        for(int j=0;j<b;j++){

            if(\*(copymatrix+i+j\*a)>best.threshold)

                \*(black\_white + i +j\*bibmp.bi.biWidth) = 255;

            else

                \*(black\_white + i + j \*bibmp.bi.biWidth) = 0;

        }

    }

}

int fill\_gray\_array(BMP rawbmp,unsigned char\* gray){

    unsigned char  red,green,blue;

    int width = rawbmp.bi.biWidth\*3;

    int adjusted\_width = (width+3)/4\*4;

    int count=0;

    for(int i=0; i< rawbmp.bi.biWidth \* rawbmp.bi.biHeight; i++){

        fread(&blue,1,1,rawbmp.file);// read the  rgb number

        fread(&green,1,1,rawbmp.file);

        fread(&red,1,1,rawbmp.file);

        gray[i] = 0.299\*red + 0.587\*green + 0.114\*blue ; //calculate the gray number

        count+=3;

        if (count == width){// read the 0s that just take up places

            while(count++ != adjusted\_width){

                fread(&red,1,1,rawbmp.file);

            }

            count = 0;

        }

    }

}

int test\_overlap\_1(unsigned char \*m1,unsigned char \*m2,int row, int column,int off1, int off2){// check if there is a 1 overlap in structure element and the bmp

    for(int i=0;i<row\*column;i++){

        if(\*(m1+i%column+i/column\*off1) \* \*(m2+i%column+i/column\*off2))

            return 1;

    }

    return 0;

}

int test\_overlap\_0(unsigned char \*m1,unsigned char \*m2,int row, int column,int off1, int off2){//check if there is a 0 overlap in structure element and the bmp

    for(int i=0;i<row\*column;i++){

        if(!(\*(m1+i%column+i/column\*off1) \* \*(m2+i%column+i/column\*off2))&&\*(m2+i%column+i/column\*off2))

            return 1;

    }

    return 0;

}

int write\_gray\_palette(BMP bibmp){

    RGBQUAD rgbquad[256];

    for(int i=0; i<256; i++){

        rgbquad[i].rgbBlue = i;

        rgbquad[i].rgbGreen = i;

        rgbquad[i].rgbRed = i;

        rgbquad[i].rgbReserved = 0 ;

    }

    fwrite(rgbquad,sizeof(rgbquad),1,bibmp.file);

}

MIN\_MAX min\_max(unsigned char \*arr, int n){

    int bottom = 255,up = 0;

    for(int i=0;i<n;i++){

        if(\*(arr+i)< bottom)

            bottom = \*(arr+i);

        if(\*(arr+i) > up)

            up = \*(arr+i);

    }

    MIN\_MAX result ={bottom,up};

    return result;

}

BMP create\_bmp(){

    BMP newbmp;

    FILE \* bmp;

    BITMAPFILEHEADER bf;

    BITMAPINFOHEADER bi;

    if( !(bmp = fopen("D:\\test.bmp","rb"))){

        printf("Not find file in D:\\test.bmp");

        return newbmp;

    }

    fread(&bf,sizeof(BITMAPFILEHEADER),1,bmp);

    fread(&bi,sizeof(BITMAPINFOHEADER),1,bmp);

    newbmp.bf = bf;

    newbmp.bi = bi;

    newbmp.file = bmp;

    return newbmp;

}

int initial\_bmp(BMP bmp){

    fwrite(&bmp.bf,sizeof(bmp.bf),1,bmp.file);

    fwrite(&bmp.bi,sizeof(bmp.bi),1,bmp.file);

}

int erosion(unsigned char \*eromatrix,unsigned char \* black\_white,unsigned char\* structure\_element,BMP erobmp){

    for(int i=0;i<erobmp.bi.biWidth;i++){// copy the places that wont be erosed

        \*(eromatrix + i) = \*(black\_white +i);

        \*(eromatrix + i + (erobmp.bi.biHeight-1)\*erobmp.bi.biWidth) = \*(black\_white + i +(erobmp.bi.biHeight-1)\*erobmp.bi.biWidth);

    }

    for(int i=0;i<erobmp.bi.biHeight;i++){//copy the left places that wont be erosed

        \*(eromatrix + i\*erobmp.bi.biWidth) = \*(black\_white + i\*erobmp.bi.biWidth);

        \*(eromatrix + i\*erobmp.bi.biWidth+erobmp.bi.biWidth - 1) = \*(black\_white + i\*erobmp.bi.biWidth + erobmp.bi.biWidth - 1);

    }

    for(int i=0;i<erobmp.bi.biWidth-3;i++){//erosion

        for(int j=0;j<erobmp.bi.biHeight-3;j++){

            if(test\_overlap\_1(black\_white+i+j\*erobmp.bi.biWidth,structure\_element,3,3,erobmp.bi.biWidth,3))

                \*(eromatrix + i+1 + (j+1)\*erobmp.bi.biWidth) = 255;

            else

                \*(eromatrix + i+1 + (j+1)\*erobmp.bi.biWidth) = 0;

        }

    }

}

int delasion(unsigned char\* delmatrix,unsigned char \* black\_white,unsigned char \* structure\_element,BMP delbmp){

    for(int i=0;i<delbmp.bi.biWidth;i++){//copy the places that wont be delated

        \*(delmatrix + i) = \*(black\_white +i);

        \*(delmatrix + i + (delbmp.bi.biHeight-1)\*delbmp.bi.biWidth) = \*(black\_white + i +(delbmp.bi.biHeight-1)\*delbmp.bi.biWidth);

    }

    for(int i=0;i<delbmp.bi.biHeight;i++){//copy the left places that wont be delated

        \*(delmatrix + i\*delbmp.bi.biWidth) = \*(black\_white + i\*delbmp.bi.biWidth);

        \*(delmatrix + i\*delbmp.bi.biWidth+delbmp.bi.biWidth - 1) = \*(black\_white + i\*delbmp.bi.biWidth + delbmp.bi.biWidth - 1);

    }

    for(int i=0;i<delbmp.bi.biWidth-3;i++){//delation

        for(int j=0;j<delbmp.bi.biHeight-3;j++){

            if(test\_overlap\_0(black\_white+i+j\*delbmp.bi.biWidth,structure\_element,3,3,delbmp.bi.biWidth,3))

                \*(delmatrix + i+1 + (j+1)\*delbmp.bi.biWidth) = 0;

            else

                \*(delmatrix + i+1 + (j+1)\*delbmp.bi.biWidth) = 255;

        }

    }

}

int write\_data(BMP bmp,unsigned char\* black\_white){

    int new\_width = (bmp.bi.biWidth+3)/4\*4;

    int count=0;

    unsigned char zero = 0;

    for(int i=0; i<bmp.bi.biSizeImage; i++){

        fwrite(&black\_white[i],1,1,bmp.file);

        count++;

        if(count == bmp.bi.biWidth){// put 0s that take up places

            while(count++ != new\_width){

                fwrite(&zero,1,1,bmp.file);

            }

            count=0;

        }

    }

}

**四、实验环境及运行方法**

把需要测试的test.bmp放在D://test.bmp 把源代码用编译器（gcc）编译运行即可

**五、实验结果展示**

test binary

erosion dilation

open close

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**六、心得体会**

本次作业并无太多难点，先将图片切块，并将切好的块拿去用大津算法计算阈值，完成二值化，之后的腐蚀膨胀等操作也较为简单。

不过只是单纯的用切片大津算法有所缺陷 从图中可以看出，某块本是连续的，亮度相近的区域，因为被切进不同的块中，获得了不同的阈值，可能会有明显的黑白边界，显得很突兀。

希望未来会有更好的算法来解决这个问题。