

图象处理系统

C++课程设计



3076行

162210107蔡蕾

南京航空航天大学

目录

[需求分析 3](#_Toc134792649)

[图像处理基础知识 3](#_Toc134792650)

[数字图像的表示 3](#_Toc134792651)

[数字图像的空间分辨率即灰度级分辨率 4](#_Toc134792652)

[像素间的基本关系 4](#_Toc134792653)

[邻接性与连通性 4](#_Toc134792654)

[数字图像文件格式 5](#_Toc134792655)

[BMP图像文件的注意事项 6](#_Toc134792656)

[系统总框图 7](#_Toc134792657)

[模块的设计分析 7](#_Toc134792658)

[位图结构体的定义sss.h 7](#_Toc134792659)

[虚基类bmp.h 9](#_Toc134792660)

[像素值RGBTriple\* surface 10](#_Toc134792661)

[偏移offset 10](#_Toc134792662)

[success 10](#_Toc134792663)

[基类之单位图bmp4.h/cpp 11](#_Toc134792664)

[基类之八位图bmp2.h/cpp 11](#_Toc134792665)

[基类之十六位图bmp3.h/cpp 12](#_Toc134792666)

[掩饰码 12](#_Toc134792667)

[基类之二十四位图bmp1.h/cpp 13](#_Toc134792668)

[帮助图像像素处理的函数help.h/cpp 13](#_Toc134792669)

[主要功能deal.h/cpp 13](#_Toc134792670)

[图像数组指针和个数 14](#_Toc134792671)

[清屏及关闭 15](#_Toc134792672)

[主函数main.cpp 16](#_Toc134792673)

[登录注册功能 16](#_Toc134792674)

[音乐播放功能 17](#_Toc134792675)

[流运算符重载 17](#_Toc134792676)

[所有定义的函数及说明 18](#_Toc134792677)

[存在问题与不足及对策 24](#_Toc134792678)

[本项目特色 25](#_Toc134792679)

[双线性插值 25](#_Toc134792680)

[多边形扫描线填充算法 26](#_Toc134792681)

[图像旋转算法（数学计算） 27](#_Toc134792682)

[模糊算法（3\*3网格） 29](#_Toc134792683)

[十六位图的处理 31](#_Toc134792684)

[快速排序 31](#_Toc134792685)

[桶排序 33](#_Toc134792686)

[灰度化 33](#_Toc134792687)

[取平均值法 33](#_Toc134792688)

[加权平均法 34](#_Toc134792689)

[二值化处理 34](#_Toc134792690)

[使用说明 35](#_Toc134792691)

[程序 36](#_Toc134792692)

[程序运行平台 36](#_Toc134792693)

[程序主要功能 36](#_Toc134792694)

[程序自己拍的b站视频 36](#_Toc134792695)

# 需求分析

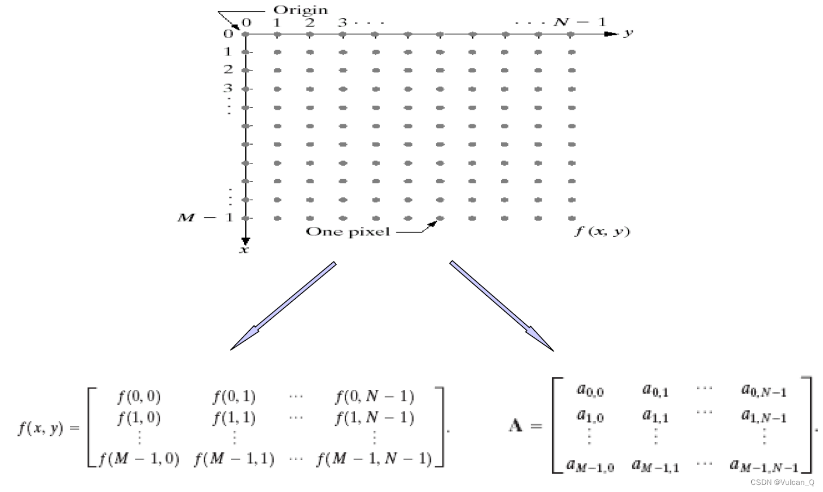
数字图像在外存储器设备中的存储形式是图像文件，图像必须按照某个已知的、公认的数据存储顺序和结构进行存储，才能使不同的程序对图像文件顺利进行打开或存盘操作，实现数据共享。图像数据在文件中的存储顺序和结构称为图像文件格式。

BMP文件是windows操作系统所推荐和支持的图像文件格式，是一种将内存或显示器的图像数据不经过压缩而直接按位存盘的文件格式，所以称为位图（bitmap）文件，因其文件扩展名为BMP，故称为BMP文件格式，简称BMP文件。

图像处理也含有许许多多的算法，比如：双线性插值，三线性插值，扫描线优化的填充算法，中值滤波，灰度化，旋转，二值化等等。研究图像的算法还在继续，相信会有更多的高效率算法的出现。本项目着重对bmp图像进行相关的处理。

## 图像处理基础知识

### 数字图像的表示



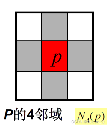
f(x,y)——灰度级值，又称灰度矩阵，取值范围为[0,L-1]其中，L=2k。则相应的，k=1时，就是所谓的二值图像。

### 数字图像的空间分辨率即灰度级分辨率

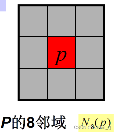
定义：空间分辨率 ：图像中可辨别的最小细节

定义：灰度级分辨率 ：在灰度级别中可辨别的最小变化

### 像素间的基本关系

4邻域——N4( P)

对角邻域——ND( P) : (x+1, y+1)、(x+1, y-1)、(x-1, y+1)、(x-1, y-1)

8邻域——N8( P)

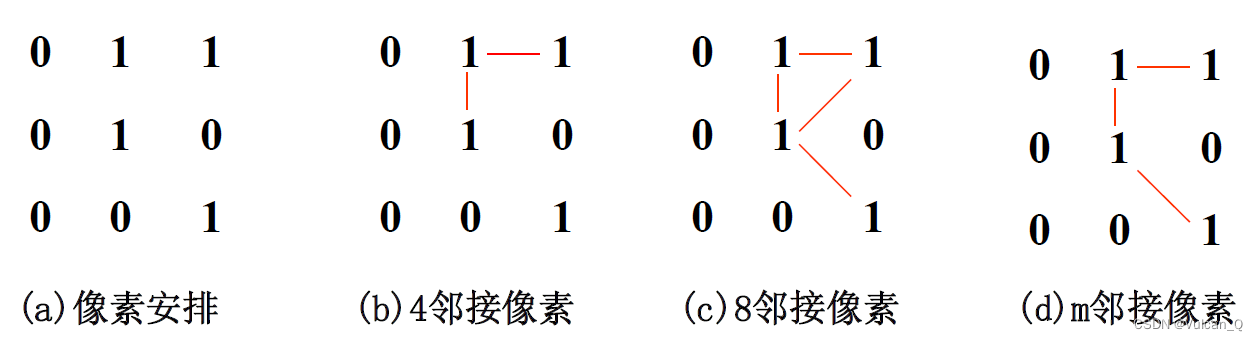
N8( P) = N4( P) + ND( P)

### 邻接性与连通性

#### 邻接性（Adjacency）：

两个像素是否相邻以及它们的灰度值是否满足特定的相似性准则

令V为定义邻接性的灰度值集合，（）例如，在具有5级灰度等级的图像中，当V={0,1,2} 时，意味着灰度值为0或1或2的像素满足灰度值相似准则。）



#### 连通性 ：

通路（Path）：从像素p(x0,y0)到像素q(xn,yn)的通路是指一个像素序列(x0,y0)，(x1,y1)，…，(xn,yn),其中像素(xi,yi)和(xi-1,yi-1)是邻接的。此处，n为通路的长度。

如果(x0,y0) = (xn,yn), 则通路为闭合通路。

根据邻接关系，通路分为4通路、8通路和m通路

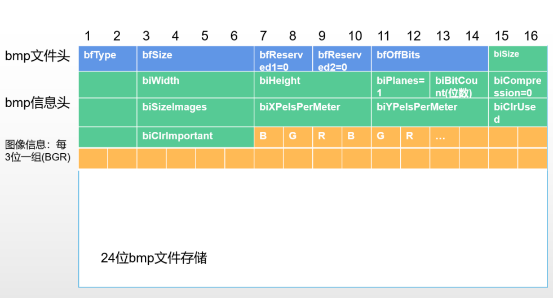
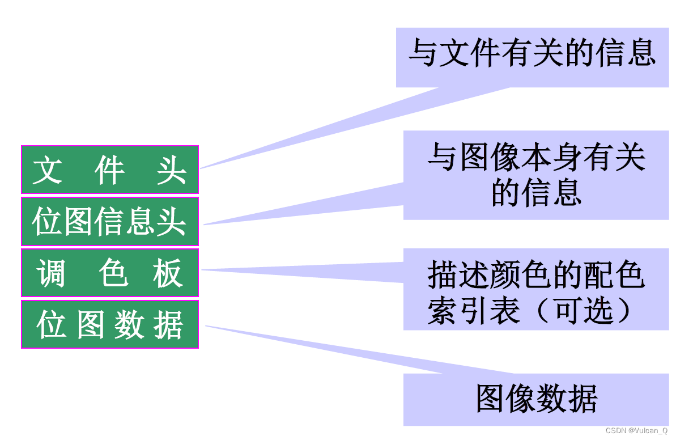
设p和q是一个图像子集S中的两个像素，如果存在一条完全由在S中的像素组成的从p到q的通路，则称p在S中与q相连通。

连通成分 ：一个像素集合，如果集合内的每一个像素与集合内的其他像素连通，则称该集合为一个连通成分。

## 数字图像文件格式

位图：位图又称点阵图像，它以像素为单位，描述图像的实际信息，适合于描述自然界的图像

下图是位图文件在内存中的分布以及所占字节数！



### BMP图像文件的注意事项

在BMP文件格式中规定每行的字节数必须是4的整数倍。若不是，要补0直至4的整数倍。

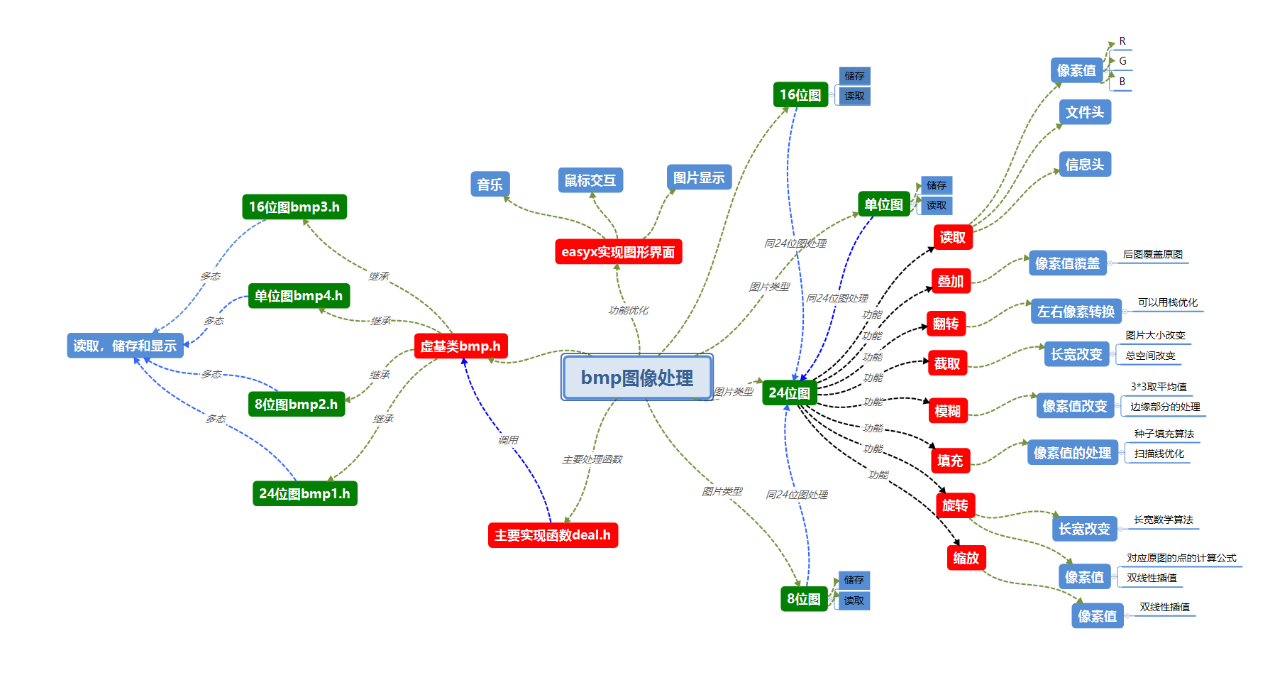
图像字节数biSizeImage的计算方法为：biSizeImage＝(biWidth×biBitCount+31)/32×4×biHeight或＝bfSize－bfOffBits

一般来说，BMP文件的像素数据从下到上，从左到右存放的。也就是说，从文件中最先读到的是图像最下面一行的左边第一个像素，然后是左边第二个像素……接下来是倒数第二行左边第一个像素，左边第二个像素……依次类推，最后得到的是最上面一行的最右一个像素。所以对于像素的储存和读取需要额外注意。

在调色板数据结构和图像数据中，每个像素的颜色排列顺序为：蓝、绿、红 ，不同于传统的RGB顺序！！

每个位图像素的所占位数不同，读取和存储时尤其要注意！

# 系统总框图



画在了一张图中。

# 模块的设计分析

## 位图结构体的定义sss.h

位图文件头（bmp file header）： 提供文件的格式、大小等信息

位图信息头（bitmap information）：提供图像数据的尺寸、位平面数、压缩方式、颜色索引等信息

struct BmpFileHeader

{

unsigned short bfType; // 2 Bytes，文件类型，必须为 "BM"，即 0x424D

unsigned int bfSize; // 4 Bytes，整个 BMP 文件的大小，单位为字节

unsigned short bfReserved1; // 2 Bytes，保留位置为 0

unsigned short bfReserved2; // 2 Bytes，保留位置为 0

unsigned int bfOffBits; // 4 Bytes，文件起始位置到图像位图数据的字节偏移量

} ;

struct BmpFileInFoHeader

{

unsigned int biSize; // 本结构所占用字节数

int biWidth; // 位图的宽度，以像素为单位

int biHeight; // 位图的高度，以像素为单位

unsigned short biPlanes; // 目标设备的平面数不清，必须为1

unsigned short biBitCount// 每个像素所需的位数，必须是1(双色), 4(16色)，8(256色)或24(真彩色)之一

unsigned int biCompression; // 位图压缩类型，必须是 0(不压缩),1(BI\_RLE8压缩类型)或2(BI\_RLE4压缩类型)之一

unsigned int biSizeImages; // 位图的大小，以字节为单位

int biXPelsPerMeter; // 位图水平分辨率，每米像素数

int biYPelsPerMeter; // 位图垂直分辨率，每米像素数

unsigned int biClrUsed;// 位图实际使用的颜色表中的颜色数

unsigned int biClrImportant;// 位图显示过程中重要的颜色数

} ；

还有像素的结构体

struct RGBTriple

{

unsigned char blue;

unsigned char green;

unsigned char red;

};

## 虚基类bmp.h

位图文件类的共同之处（父亲），利用位图文件格式的共同部分如：名称（filename）文件头（fileHeader），信息头（fileInFoHeader），像素值，偏移（后面会详细介绍）。还明确了相关虚函数(读图像，写图像，显示图像)，用来方便基类对虚函数表的继承和调用：

//虚基类

class bmp

{

public:

virtual void readPic(char fileName[]) = 0;//读文件

virtual void writePic(void (\*myMethod)(int, int, RGBTriple\*), char\* outFileName)=0;

//基于本类对象进行处理后存储到outfilename这个图像中

virtual void ShowImage(int i) = 0;

//指定窗口中显示图像

friend ostream& operator<<(ostream& out, bmp& obj);

//输出流重载

int success=0;

//记录是否成功读取和存储

int offset=0;

//内存偏移量

RGBTriple\* surface;

//指向储存像素的空间

BmpFileHeader fileHeader;

//文件头

BmpFileInFoHeader fileInFoHeader;

//信息头

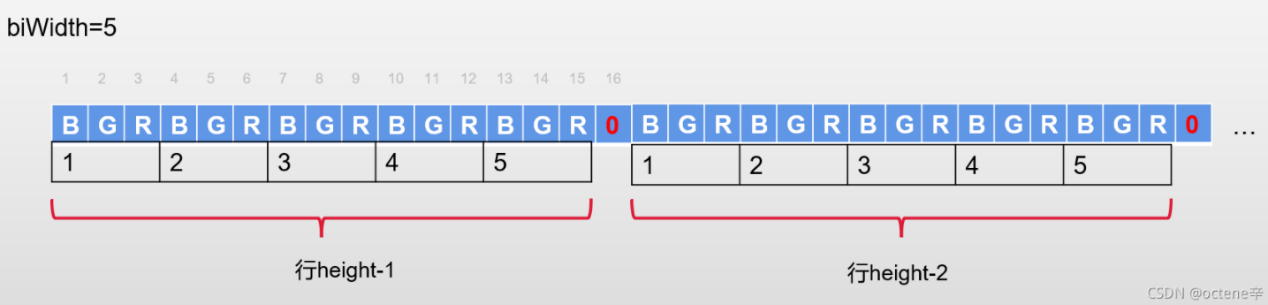
char\* mfilename;

//图像名称

};

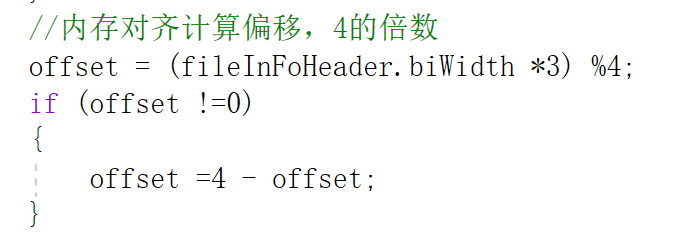
### 像素值RGBTriple\* surface

由于像素的存储顺序是从下到上，从左到右，在文件中以类似一维数组的方法线性存储。每个像素的颜色信息每3个字节一组，按BGR的顺序存放，所以我取名为RGBTriple\* surface;//指向储存像素的空间。其中每个字节只存一个颜色值。颜色值范围是0~255，用无符号char型存储。 对于宽度不是4的倍数的图片(如：474px)，每一行的像素数据存储完后，会自动空出几个字节，直到这一行的字节数为4的倍数为止。



### 偏移offset

所以，在读取宽度不是4的倍数的图片时，一行的数据读完后，要跳过几个字节才能读到下一行的数据。跳过字节的个数，我取名为offset。



### success

程序的健壮性考虑：读或写取失败的情况，报给程序本身可以及时停止。在类中新增success变量用来记录读取和写入每种操作是否成功（0为失败，1为成功）。

## 基类之单位图bmp4.h/cpp

单色图，还在开发中

## 基类之八位图bmp2.h/cpp

继承虚基类的部分以及自己会多出来一块数据区需要新增。

//基类八位图

class bmp2 :public bmp

{

public:

RGBQUAD\* rgbQuad;

bmp2();

bmp2(const bmp2& obj);//拷贝构造

virtual void readPic(char fileName[]);

virtual void writePic(void (\*myMethod)(int, int, RGBTriple\*), char\* outFileName);

virtual void ShowImage(int i);

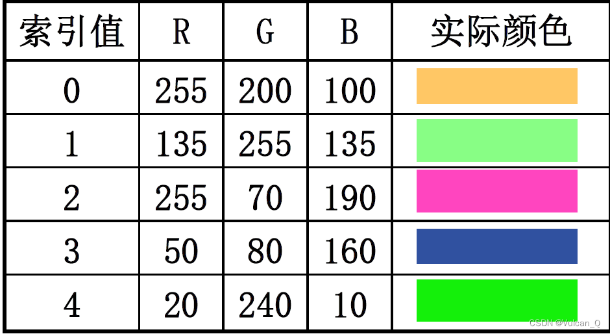
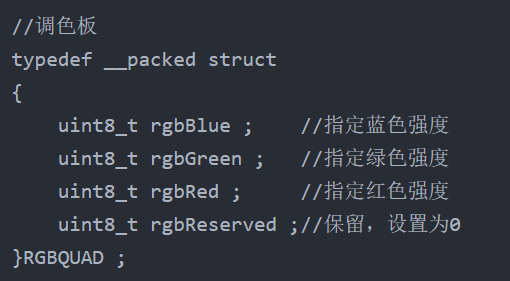
friend ostream& operator<<(ostream& out, bmp& obj);

~bmp2();

};

#### 调色板RGBQUAD\* rgbQuad;

小于等于8的位图在图片存储中会多出调色板（rgbquad\* rgbquad）：



它相当于一个像素索引，每个像素值对应调色板第几个然后根据调色板中rgb数值进行像素赋值，将每个值读出来再存进像素数组中。对应于信息头中的biClrUsed,注意在信息头和文件头后面紧跟着的是调色板数据而不是像素数据。在出来读取存储是要将调色板读取和存进去才能保证位图格式保存准确。

## 基类之十六位图bmp3.h/cpp

继承虚基类的部分，同时自己会新增掩码部分（本项目只讨论565格式掩码）

class bmp3 :public bmp

{

public:

char\* rgb\_mask;

//掩饰码

char\* imagebuf;//实际像素值

char\* bmpbuf;//掩饰码值

bmp3();

bmp3(const bmp3& obj);//拷贝构造

virtual void readPic(char fileName[]);

virtual void writePic(void (\*myMethod)(int, int, RGBTriple\*), char\* outFileName);

virtual void ShowImage(int i);

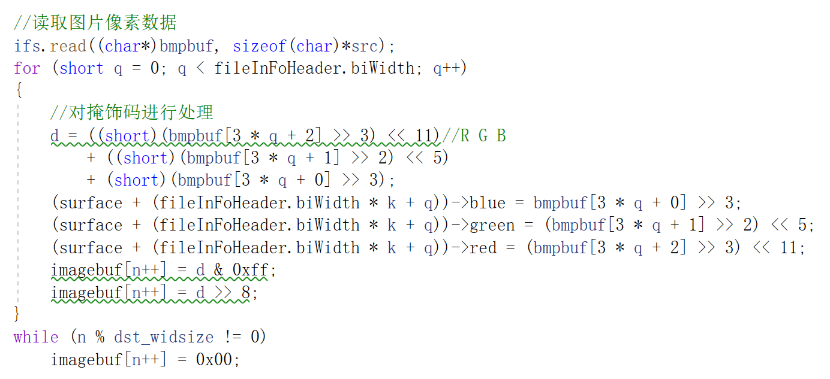
friend ostream& operator<<(ostream& out, bmp& obj);

~bmp3();

};

### 掩饰码

常见的16位BMP有两种格式-----XRGB555和RGB565。XRGB555跟RGB565这两种格式的BMP最主要的区别是，XRGB555不带调色板（14+40个字节后就是像素点数据了），其存储的数据就是BMP点的像素数据值，也就是XRGB55的数据，而RGB565有调色板（其实也不是什么调色板，而只是RGB三色的掩码（char\* rgb\_mask;//掩饰码）），BMP文件里面实际存储的数据也是跟掩码进行了运算后才存储的。将掩码跟像素值进行“与”运算再进行移位操作就可以得到各色分量值。详见代码：



## 基类之二十四位图bmp1.h/cpp

二十四位图在原虚基类的基础上没啥要增加的，所以继承和重写一下虚函数就可以了。不做过多展示！

## 帮助图像像素处理的函数help.h/cpp

主要用于对于一些只需对图片像素值进行修改的操作：

//水平翻转

void rowReverse(int height, int width, RGBTriple\* image);

//模糊化

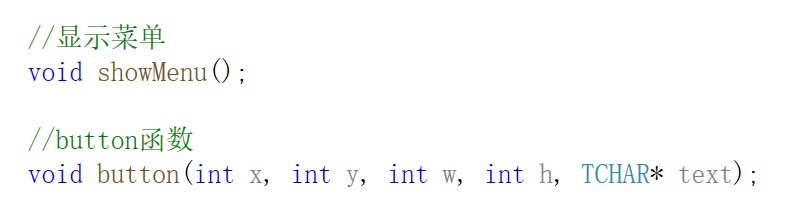
void makeBlur(int height, int width, RGBTriple\* image);

//灰度化

void makeGary(int height, int width, RGBTriple\* image);

## 主要功能deal.h/cpp

主要的图像操作功能我放在了deal.h/cpp中。图形界面(eaxyx)的实现我放在了deal中用来展示图形处理的功能选择界面。（如图）



通过鼠标交互点击不同的按钮（通过调用按钮生成函数来实现，大大减少了代码量）和输入改变的图像数据以及命名的新图片名称去实现不同的功能。处理功能在deal中有：读取，翻转，模糊，旋转，缩放，截取，叠加，灰度，二值，已读，清空，关闭。

### 图像数组指针和个数

deal中对已经读取的bmp名称进行保存，方便每次新的操作对已经处理图像进行展示并选择，创造了图像数组指针（参与虚基类的赋值兼容规则来访问基类对象）用来储存已读取的bmp以及用m\_EmpNum来记录已经处理的个数。对于数组它是指针的指针，它指向存储虚基类指针的地址，也方便了新加入成员时对地址进行更新。每次读取或完成一个功能便需要对m\_EmpNum和指针进行实时更新，由于指针内存的连续性每次都需要重新开辟新的指针数组把之前的重新储存一遍并把新的存入，在把这个新空间地址赋给指针m\_emarray。

int m\_EmpNum;//个数

bmp\*\* m\_EmArray;//图像数组指针

bmp\*\* newspace = new bmp \* [m\_EmpNum + 1];

if (this->m\_EmArray != NULL)

{

for (int i = 0; i < this->m\_EmpNum; i++)

{

newspace[i] = this->m\_EmArray[i];

}

}

newspace[this->m\_EmpNum] = a1;

m\_EmpNum++;

//释放原有空间

delete[]this->m\_EmArray;

//更改新空间的指向

this->m\_EmArray = newspace;

### 清屏及关闭

界面图像数达到上限2时还想继续进行图形处理操作时，增加了清屏并显示上一步处理好的图像的功能。为了更像一个软件，加入了关闭的按钮用来关闭窗口。同时防止意外点击造成的数据缺失和操作连续性遭受破坏我增加弹窗功能来再次确认是否关闭或清屏。

//清屏

void deal::closeImage()

{

//获取句柄

HWND hnd = GetHWnd();

//弹出窗口

int isok = MessageBox(hnd, "清屏并显示最后一步操作图片", "提示", MB\_OKCANCEL);

if (isok == IDOK&&m\_EmpNum!=0)

{

clearrectangle(80, 70, 750, 850);

TCHAR s7[10] = "清屏";

button(700, 820, 40, 20, s7);

bmp\* a = this->m\_EmArray[m\_EmpNum - 1];

a->ShowImage(0);

}

else

{

cout << "go on!";

}

}

//关闭窗口

void deal::close()

{

HWND a = GetHWnd();

int i= MessageBox(a, "是否关闭", "消息框", MB\_OKCANCEL);

if (i == IDOK)

{

closegraph();

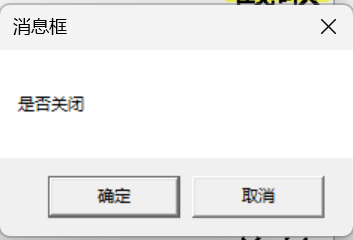
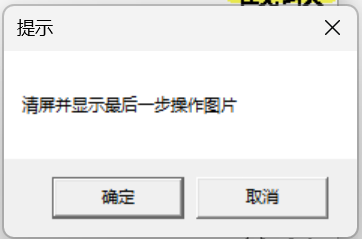
cout << "欢迎使用，下次再见！";

exit(0);

}

}

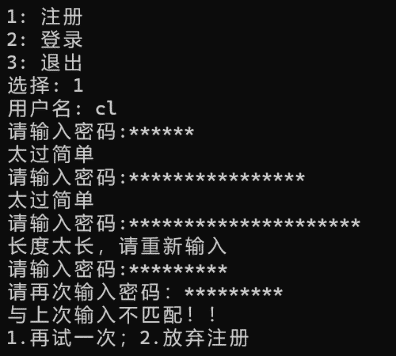
效果图：



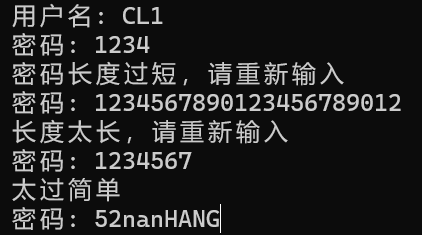
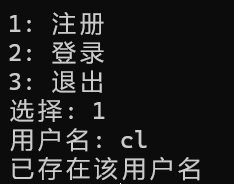
## 主函数main.cpp

### 登录注册功能

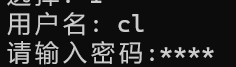
一个软件都有用户登录的功能，我也将此功能纳入我的项目！用data.txt文件用来记录已经注册的用户信息，注册时会扫描已经注册好的用户名中是否已经存在，同时注册时添加的密码隐写功能。登录时，密码只能输三次，这个缺少密码隐写功能。防止误打开的情况加入关闭软件的功能以结束程序运行。



其中有用户名若重复以及密码太过简单太短太长的检查操作。



密码隐写功能！！！



### 音乐播放功能

为了使使用者在使用时心情更加愉悦，特意在主函数中增加背景音乐功能（音乐是我自己挑的）。

#pragma comment(lib,"winmm.lib")//处理接口

void BGM()

{

mciSendString(\_T("open ./music.mp3"), 0, 0, 0);//0，0,0为默认输出控制台

mciSendString(\_T("play ./music.mp3"), 0, 0, 0);

}

### 流运算符重载

在虚基类bmp.h/cpp中加入了流运算符重载的友元，将它的具体实现放在了main.cpp从而和deal中的已读功能相联系。

用来显示所有已经处理过图像的详细信息更加方便操作者使用！

ostream& operator<<(ostream& out, bmp& obj)

{

cout << "名称:"<<obj.mfilename << endl;

cout << "位图格式:" << obj.fileInFoHeader.biBitCount << endl;

cout << "偏移:" << obj.offset << endl;

cout << "高:" << obj.fileInFoHeader.biHeight<<" 宽:"<<obj.fileInFoHeader.biWidth<<endl;

//cout << "com" << obj.fileInFoHeader.biCompression;

return out;

}



# 所有定义的函数及说明

1．virtual void readPic(char filename[])

虚基类，各位图读取bmp图像的信息头，文件头，像素。每个位图会不同所以在每个基类中进行了重写。

2.virtual void writePic (void (\*myMethod)(int, int, RGBTriple\*), char\*outFileName)

存储图片，前面为函数指针根据不同的处理传来不同的数据，来存储进新的图片主要用于对help.cpp中函数的指针传入时的处理。同样每个位图会不同所以在每个基类中进行了重写。

3.virtual void ShowImage(int i)

在窗口不同的位置（根据不同的功能决定在何地）中显示图像,主要利用easyx的图形界面，但由于只有展示的24位图是正确的，所以用于对24位图的处理。

4.friend ostream& operator<<(ostream& out,bmp& obj)

流运算符重载，重载<<方便显示已经读取的图片信息（前文已经有所解释）

5.bmp1(const bmp1& obj)

每个基类的拷贝构造函数

6.bmp1()

每个基类的默认构造函数

7.~bmp1()

每个基类的析构函数

8.void rowReverse(int height,int width,RGBTriple\* image)

水平翻转函数，单独处理类中像素后作为函数指针传入每个类的writePic进行图像的储存。



9.void makeBlur(int height,int width,RGBTriple\* image)

模糊化函数，单独处理类中像素后作为函数指针传入每个类writePic进行图像的储存。（此算法将在特色中具体讲解）



10.void BGM()

音乐播放函数

11.void showMenu()

显示图形界面的菜单

12.void button(int x,int y,int w,int h,TCHAR\*text)

生成按钮

13.void readimagebox()

读取图像功能

14.void ImageInvert()

deal中的翻转功能，和前面help中的翻转合起来用



15.void Imagemuhu()

deal中的模糊功能，和前面help中的模糊合起来用



16.void Imageintercept()

图像截取功能：选取特定的起点和终点，主要针对矩形区域进行截取，然后储存生成新的图像，注意bmp图像相关数据大小的改变。



17.void ImageOver()

图像叠加功能，将一个图像叠加到另一个图像上，只需对其像素进行改变，考虑超出边界的情况。



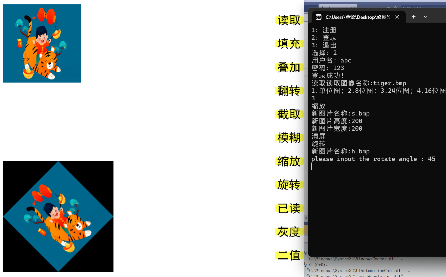
18.void ImageBig()

图像缩放功能



19.void ImageTurn()

图像旋转功能



20.void closeImage()

清屏功能并有弹窗再次确认是否清屏

21.void ImageAdd()

填充图像功能



22.void function\_line(double x1,double y1,double x2,double y2,double& k,double& b)

填充图像功能中所需的算法的求解斜率和截距的函数

23.double solve\_y(double k,double b,double x)

填充图像功能中所需的算法的返回交点的y坐标值

24.double solve\_x(double k,double b,double y)

填充图像功能中所需的算法的返回交点的x坐标值

25.void sort\_min\_to\_max(double suitable\_x[],int n)

填充图像功能中根据每个扫描线交点的x的坐标值大小进行排序，然后两两配对。

26.void Suitable\_x(double x[],double y[],int N,double now\_y,double suitable\_x[],int &n)

填充图像功能中找出扫描线与边界相交的点，并储存在suitable\_x[]中，把相交的点数储存在n中

27.double find\_max(double elem[],int N)

填充图像功能中输出最大的元素值

28.double find\_min(double elem[],int N)

填充图像功能中输出最大的元素值

29.void close()

关闭窗口函数

30.void showall()

展示所有已经处理过的bmp

31.~deal()

析构函数

32. void quick\_sort(double s[], int l, int r)

快速排序

33. bool isloggedin()

判断用户输入账号密码是否与已注册的相同

34. void imagegray()

deal中的灰度化，和前面help中的灰度化合起来用



35. void makegary(int height, int width, rgbtriple\* image)

灰度具体处理函数

36. void binaryimage()

二值化处理函数

因为无法显示8位图，只能自己看同一目录下有没有喽！

37. bool IsLoggedIn()

登录检测用户名和密码是否存在

38. bool IsComp(string password)

判断密码是否长度和复杂度符合要求

39. bool IsExist(string username)

判断当前新注册的用户名是否已存在

# 存在问题与不足及对策

1.主要实现了对24位图的相关操作，对于其他存储形式还存在缺陷。

对策：学习相关存储，类比24位图进行优化。

2.鼠标交互不够智能，需要输入相关的值而不是鼠标移动来实现。就比如截取，没能做出QQ截图的效果。

对策：还需更深入学习相关的软件，如QT等专用于c++开发的软件平台。

3.执行效率低

对策：学习更高效算法进行优化。

4.叠加操作的功能完善性欠佳，对于起始点不在图像区域的处理还没有市面上软件的功能

对策：还需要对代码进行改变，比如在用户输入不在第一个图像大小中的点时也能正确处理！有时间一定要改正！

5.图像的展示功能欠佳，只能在一定区域内展示并且只能展示两个这是一个漏洞

对策：改变函数参数让用户可以自由选择展示位置

6.登录界面不能密码隐写

对策：学习如何在char数组和string当中转化以实现登录功能下的密码隐写或者改变文件读取使读取文件内容整行到char数组。

# 本项目特色

## 双线性插值

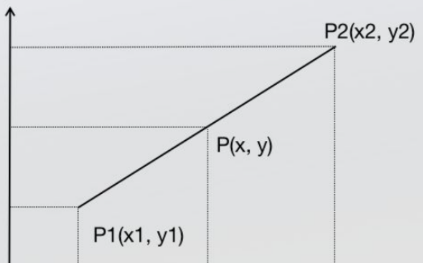
图片放大是图像处理中的一个特别基础的操作。图像领域插值常用在修改图像尺寸的过程，由旧的图像矩阵中的点计算新图像矩阵中的点并插入，不同的计算过程就是不同的插值算法。

双线性插值是用原图像中4(22)个点计算新图像中1个点，效果略逊于双三次插值，速度比双三次插值快，属于一种平衡美，在很多框架中属于默认算法。

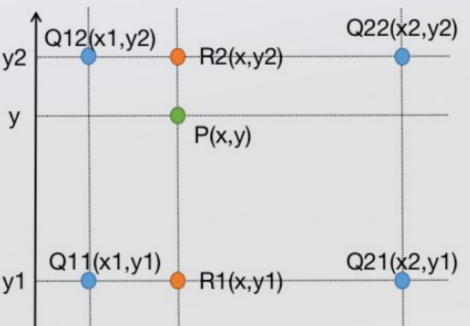


双线性插值通过待求像素点在源图像中4个最近邻像素值的加权和计算得到。（如图）

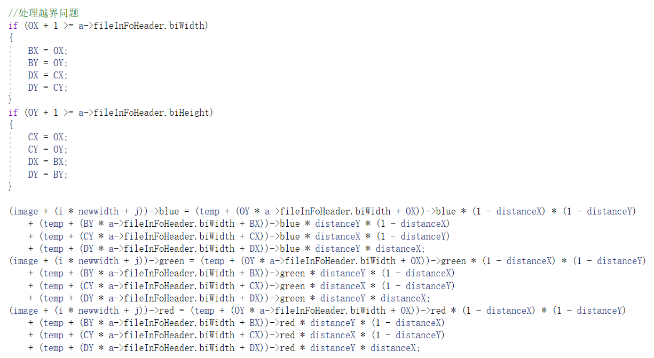
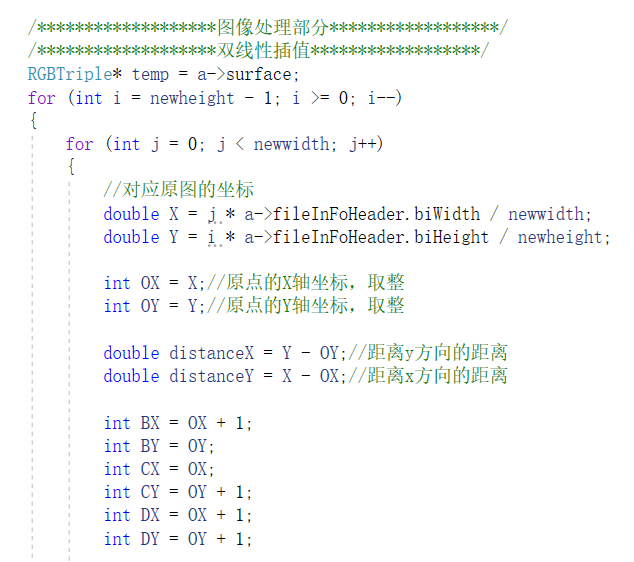
首先介绍单线性插值（如图），不难理解。



数学的一次函数关系，由两个临近点取得。



双线性插值则是进行两次插值，先计算橙色，再由两个橙色得到绿色即可。具体实现代码如下：（用到了旋转，缩放中）



## 多边形扫描线填充算法

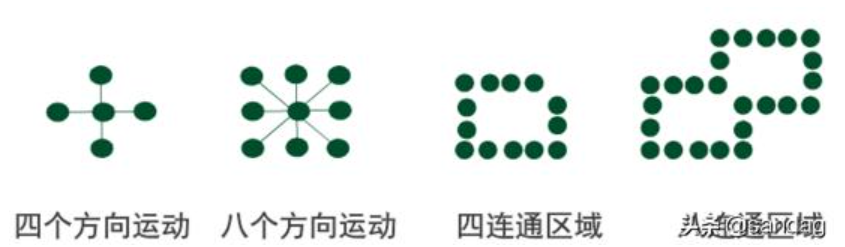
首先介绍一下基本概念！

区域：指已经表示成点阵形式的填充图形，是象素的集合。

区域填充：将区域内的一点(常称 【种子点】)赋予给定颜色，然后将这种颜色 扩展到整个区域内的过程。

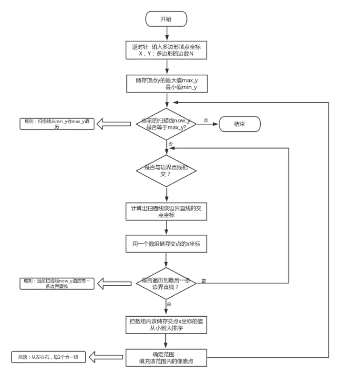
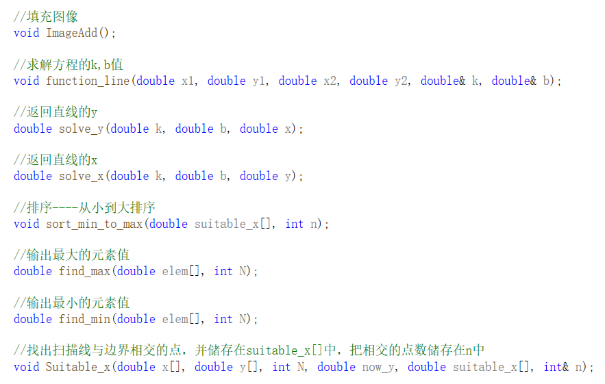
区域填充算法要求区域是连通的，因为只有在连通区域中，才可能将种子点的颜色扩展到区域内的其它点。



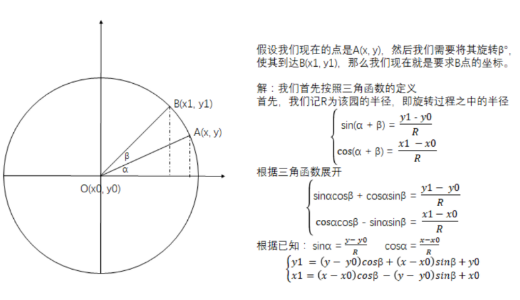
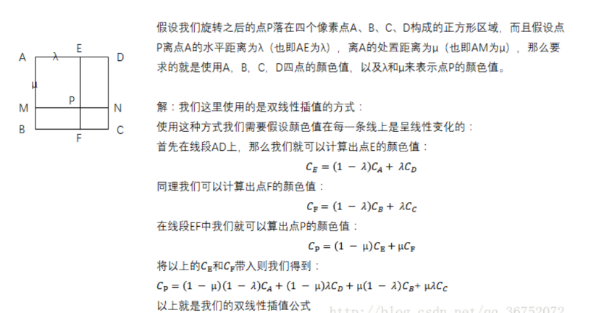
1. 内点表示：枚举出区域内部的所有象素，内部所有象素着同一个颜色，边界像素着与内部象素不同的颜色。2)边界表示：枚举出区域外部的所有象素，边界上的所有象素着同一个颜色，内部像素着与边界像素不同的颜色。1)四向连通区域：从区域上一点出发可通过【上、下、左、右】四个方向移动的组合，在不越出区域的前提下，到达区域内的任意像素。

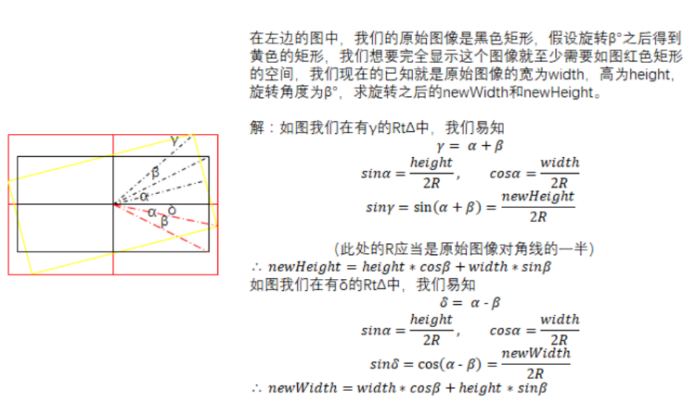
2)八向连通区域：从区域上一点出发可通过【上、下、左、右、左上、右上、左下、右下】八个方向移动的组合，在不越出区域的前提下，到达区域内的任意像素。

基本思想： 按行扫描，与扫描线的交点按从小到大储存在数组中每次取两个对其中区域进行填充，再去两个不填充以此类推！直到出了扫描边界！



## 图像旋转算法（数学计算）





简单来说就是找到新图中点对应的原图的点（这个就是上面图片展示的），由于这个点不一定是整数于是取这个浮点数点周围的四个点进行双线性插值！

//计算新图的宽度和高度

//原图的四个角坐标

auto SrcX1 = (float)(-0.5 \* width);

auto SrcY1 = (float)(0.5 \* height);

auto SrcX2 = (float)(0.5 \* width);

auto SrcY2 = (float)(0.5 \* height);

auto SrcX3 = (float)(-0.5 \* width);

auto SrcY3 = (float)(-0.5 \* height);

auto SrcX4 = (float)(0.5 \* width);

auto SrcY4 = (float)(-0.5 \* height);

//新图的四个角坐标

float DstX1 = cosa \* SrcX1 + sina \* SrcY1;

float DstY1 = -sina \* SrcX1 + cosa \* SrcY1;

float DstX2 = cosa \* SrcX2 + sina \* SrcY2;

float DstY2 = -sina \* SrcX2 + cosa \* SrcY2;

float DstX3 = cosa \* SrcX3 + sina \* SrcY3;

float DstY3 = -sina \* SrcX3 + cosa \* SrcY3;

float DstX4 = cosa \* SrcX4 + sina \* SrcY4;

float DstY4 = -sina \* SrcX4 + cosa \* SrcY4;

//计算新图的宽度和高度

auto newWidth = (int)(max(fabs(DstX4 - DstX1), fabs(DstX3 - DstX2)) + 0.5);

auto newHeight = (int)(max(fabs(DstY4 - DstY1), fabs(DstY3 - DstY2)) + 0.5);

//原来的旋转中心

double rotateX = width / 2;

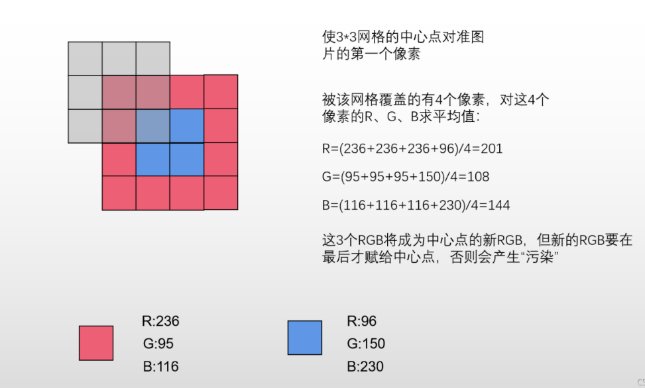
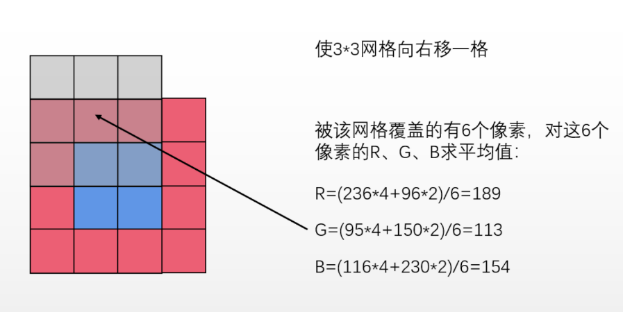
double rotateY = height / 2;

//新图的中心

double write\_rotateX = newWidth / 2;

double write\_rotateY = newHeight / 2;

## 模糊算法（3\*3网格）



模糊算法又叫去噪，或者中值滤波

//是3x3网格的中心对准图片每一个像素

//求取其覆盖所有像素点的rgb的平均值

//新的rgb最后复制，否则会造成污染

void makeBlur(int height, int width, RGBTriple\* image)

{

RGBTriple\* temp = new RGBTriple[height \* width];

int ured = 0;

int ugreen = 0;

int ublue = 0;

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

//计算实际覆盖的像素数量

int total = 0;

for (int y = i - 1, myCount = 0; myCount < 3; y++, myCount++)//每个像素点前一个

//mycout用于记录3\*3网格长度，最多取三个

{

if (y >= 0 && y < height)//保证3\*3网格所覆盖的像素点在图像中

{

for (int x = j - 1, myCount1 = 0; myCount1 < 3; x++, myCount1++)

{

if (x >= 0 && x < width)

{

ured += (image + (y \* width + x))->red;

ugreen += (image + (y \* width + x))->green;

ublue += (image + (y \* width + x))->blue;

total++;

}

}

}

}

(temp + (i \* width + j))->red = (ured / total);

(temp + (i \* width + j))->green = (ugreen / total);

(temp + (i \* width + j))->blue = (ublue / total);

ured = ugreen = ublue = 0;//开始下一轮

}

}

//传递值

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

(image + (i \* width + j))->blue = (temp + (i \* width + j))->blue;

(image + (i \* width + j))->green = (temp + (i \* width + j))->green;

(image + (i \* width + j))->red = (temp + (i \* width + j))->red;

}

}

delete[] temp;//释放临时数组

}

## 十六位图的处理

主要针对565格式的存储，对图片的像素进行处理得到真正的像素值！

//存储像素数据的数组

surface = new RGBTriple[fileInFoHeader.biHeight \* fileInFoHeader.biWidth];

short d;

for (short k = fileInFoHeader.biHeight-1, n = 0; k >=0; k--)

{

//读取图片像素数据

ifs.read((char\*)bmpbuf, sizeof(char)\*src);

for (short q = 0; q < fileInFoHeader.biWidth; q++)

{

//对掩饰码进行处理

d = ((short)(bmpbuf[3 \* q + 2] >> 3) << 11)//R G B

+ ((short)(bmpbuf[3 \* q + 1] >> 2) << 5)

+ (short)(bmpbuf[3 \* q + 0] >> 3);

(surface + (fileInFoHeader.biWidth \* k + q))->blue = bmpbuf[3 \* q + 0] >> 3;

(surface + (fileInFoHeader.biWidth \* k + q))->green = (bmpbuf[3 \* q + 1] >> 2) << 5;

(surface + (fileInFoHeader.biWidth \* k + q))->red = (bmpbuf[3 \* q + 2] >> 3) << 11;

imagebuf[n++] = d & 0xff;

imagebuf[n++] = d >> 8;

}

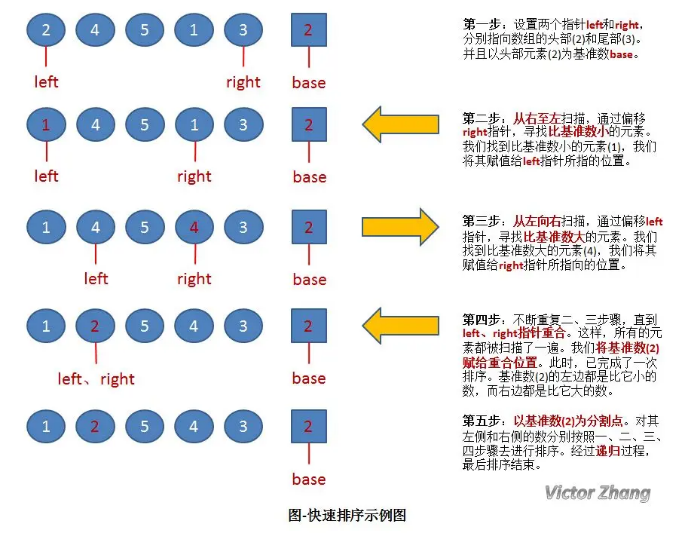
while (n % dst\_widsize != 0)

imagebuf[n++] = 0x00;

}

## 快速排序

快速排序与归并排序一样，也是基于分治的递归算法，体现在：在每一趟快速排序中，需要选出枢轴元素，然后将比枢轴元素大的数组元素放在枢轴元素的右边，比枢轴元素小的数组元素都放在枢轴元素的左边。然后，再对分别对 枢轴元素左边 和 枢轴元素右边的元素进行快速排序。



void deal::quick\_sort(double s[], int l, int r)

{

if (l < r)

{

//Swap(s[l], s[(l + r) / 2]); //将中间的这个数和第一个数交换 参见注1

int i = l, j = r, x = s[l];

while (i < j)

{

while (i < j && s[j] >= x) // 从右向左找第一个小于x的数

j--;

if (i < j)

s[i++] = s[j];

while (i < j && s[i] < x) // 从左向右找第一个大于等于x的数

i++;

if (i < j)

s[j--] = s[i];

}

s[i] = x;

quick\_sort(s, l, i - 1); // 递归调用

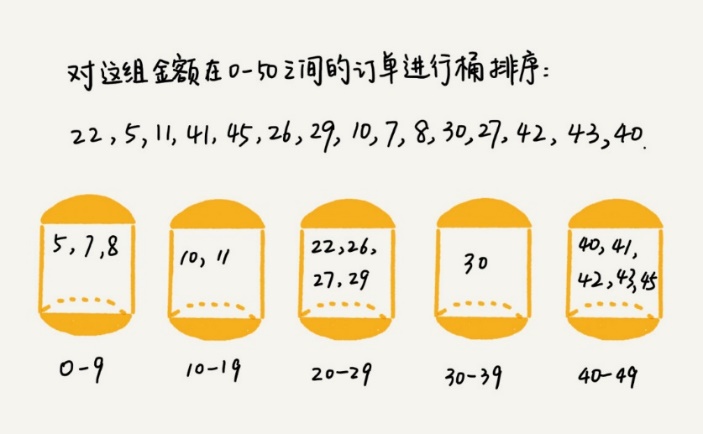
quick\_sort(s, i + 1, r);

}

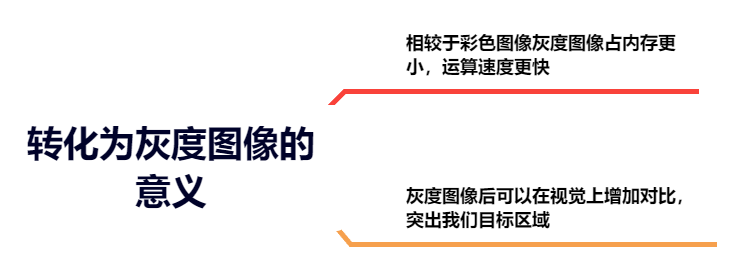
}

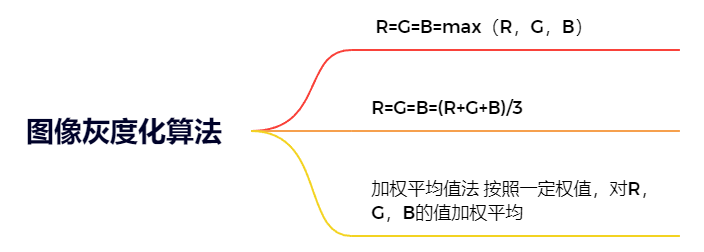
## 桶排序

桶排序的核心思想就是将要排序的数据分到几个有序的桶里，每个桶里的数据再单独进行排序。桶排序完之后，再把每个桶里的数据按照顺序依次取出，组成的序列就是有序的了。



## 灰度化





本项目用到了两种

### 取平均值法

void makeGary(int height, int width, RGBTriple\* image)

{

for (int i =0; i < height; i++)

{

for (int j =0; j < width; j++)

{

int aver = ((image + (i \* width + j))->blue + (image + (i \* width + j))->green + (image + (i \* width + j))->red) / 3;

(image + (i \* width + j))->blue = (image + (i \* width + j))->green = (image + (i \* width + j))->red = aver;

}

}

}

这个就用到了灰度化当中

### 加权平均法

这个方法用在了二值化处理中

b = (int)(0.114 \* (float)temp->blue + 0.587 \* (float)temp->green + 0.299 \* (float)temp->red);

## 二值化处理

图像的二值化处理就是将图像上的点的灰度置为0或255，也就是讲整个图像呈现出明显的黑白效果。即将256个亮度等级的灰度图像通过适当的阀值选取而获得仍然可以反映图像整体和局部特征的二值化图像。所有灰度大于或等于阀值的像素被判定为属于特定物体，其灰度值为255表示，否则这些像素点被排除在物体区域以外，灰度值为0，表示背景或者例外的物体区域。

灰度二值化处理是模式识别、机器学习等，在人脸识别等方面对图片处理的常见操作，可以减少图片的不稳定性，便于之后计算机对图片的处理，目前灰度二值化处理几乎是每种图片处理都必不可少的操作。

它包括两步，先灰度化后二值化（二值是8位图这就涉及其他位图转24位图）

主要还是8位图的保存，本软件不支持展示8位图，只能看同一路径下的图片来看效果啦！

void deal::binaryimage()

{

cout << endl;

bmp\* a = this->m\_EmArray[m\_EmpNum - 1];

cout << "新图片名称:";

char name[100] = { 0 };//bmp名称

cin>>name;

RGBQUAD rg[2];

BmpFileHeader newfile= a->fileHeader;

BmpFileInFoHeader newfilein = a->fileInFoHeader;

newfilein.biBitCount = 8;

newfilein.biClrUsed = 2;//颜色索引表数量，二值图为2

newfilein.biClrImportant = 0;//重要颜色索引为0，表示都重要

newfile.bfOffBits= sizeof(BmpFileHeader) + sizeof(BmpFileInFoHeader) + 2 \* sizeof(RGBQUAD);

//数据区偏移量，等于文件头，信息头，索引表的大小之和

newfile.bfSize = newfile.bfOffBits + newfilein.biSizeImages;//文件大小，等于偏移量加上数据区大小

rg[0].rgbBlue = rg[0].rgbGreen = rg[0].rgbRed = rg[0].rgbReserved = 0;//调色板颜色为黑色对应的索引为0

rg[1].rgbBlue = rg[1].rgbGreen = rg[1].rgbRed = 255;//白色对应的索引为1

rg[1].rgbReserved = 0;

FILE\* wfile;//输出文件

wfile = fopen(name, "wb");//打开文件为存储修改后图像做准备

if (wfile == NULL)

{

cout << "打开文件错误";

return;

}

if (fwrite(&newfile, 1, sizeof(BITMAPFILEHEADER), wfile) != 14)

{

cout << "错误文件头" << endl;

return;

}//写回位图文件头信息到输出文件

if (fwrite(&newfilein, 1, sizeof(BITMAPINFOHEADER), wfile) != 40)

{

cout << "错误信息头" << endl;

return;

}//写回位图信息头信息到输出文件

if (fwrite(&rg, 1, 2\*sizeof(RGBQUAD), wfile) != 8)

{

cout << "错误调色板" << endl;

return;

}

RGBTriple\* temp = a->surface;

unsigned char b;//存储每个像素的灰度值或二值

unsigned char\* c;//存储每行像素的二值

/\*将彩色图转为二值图\*/

c = (unsigned char\*)malloc((newfilein.biWidth + 3) / 4 \* 4);

//给变量c申请目标图每行像素所占大小的空间,同样四字节对齐

for (int i = 0; i < newfilein.biHeight; i++)

{//遍历图像每行的循环

for (int j = 0; j < newfilein.biWidth; j++)

{//循环像素宽度次,就不会计算读入四字节填充位

b = (int)(0.114 \* (float)temp->blue + 0.587 \* (float)temp->green + 0.299 \* (float)temp->red);

//a中每三个字节分别代表BGR分量，乘上不同权值转化为灰度值

if (190 <= (int)b) b = 1;//将灰度值转化为二值，这里选取的阈值为190

else b = 0;

c[j] = b;//存储每行的二值

}

fwrite(c, (newfilein.biWidth + 3) / 4 \* 4, 1, wfile);

//将二值像素四字节填充写入文件,填充位没有初始化,为随机值

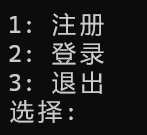
}

fclose(wfile);

}

# 使用说明

1. 点击运行。
2. 输入用户名和密码或者注册后再登录，只有输入正确密码才能进入



1. 刚开始先点读取，输入图像名称，然后选择位图（只有几个操作可以用于单位图，8位图和16位图掩码模式）



1. 选择不同的功能根据不同的调控板指令进行操作（会有自动保存功能，程序同一路径下查找）举个例子！



1. 点击清屏可以处理其他需要处理的图片
2. 点击关闭退出程序

# 程序

## 程序运行平台

VS2022

## 程序主要功能

对bmp24,8,16位图进行处理

## 程序自己拍的b站视频

<https://t.bilibili.com/787959433383641094?share_source=pc_native>

这是自己录的实现！

主要是登录板块的改动！！！！

点开链接就可以查看！！！