# 深圳大学考试答题纸

(以论文、报告等形式考核专用)

二〇二一 ～二〇二二 学年度第 2 学期

课程编号 1500760001-01 课程名称 《数字图像处理》 主讲教师 吴惠思 评分

学 号 2020151091 姓名 张欣杰 专业年级 2020 级软件工程 2 班

| 教师评语： |
| --- |
| |

项目名称：

**评分标准：**

| 得分点 | 评分标准 | 分值 | 得分 |
| --- | --- | --- | --- |
| 基本功能模块 | A.颜色（灰度）变换（例如：直方图均衡化、点操作、伽马校正、分段函数、像素级代数运算、像素级逻辑运算、位平面分割等）(15 分)<br>B.图像滤镜增强（例如：邻域操作、线性与非线性滤波、图像平滑、图像锐化等）(20 分)<br>C.空间几何变换（例如：图像平移、图像缩放、图像旋转自由变换组合等）(15 分)<br>注：以上 ABC 每组模块至少选择完成 1 项基本功能，与小实验中的功能应有不同。 | 50 | |
| 特色功能模块 | 结合课程内容及实验的特色扩展模块（自行设计）。 | 25 | |
| 系统界面 | 系统界面简洁美观，功能交互友好。 | 5 | |
| 报告书写 | 系统各模块功能、算法和结果描述清晰，格式规范，内容完整。 | 10 | |
| PPT 答辩（含程序演示） | PPT 编写美观、有逻辑性，程序演示、口头表达清晰准确。 | 10 | |
| | | 总分 | |

# 1 “图变”小型图像处理系统项目简介

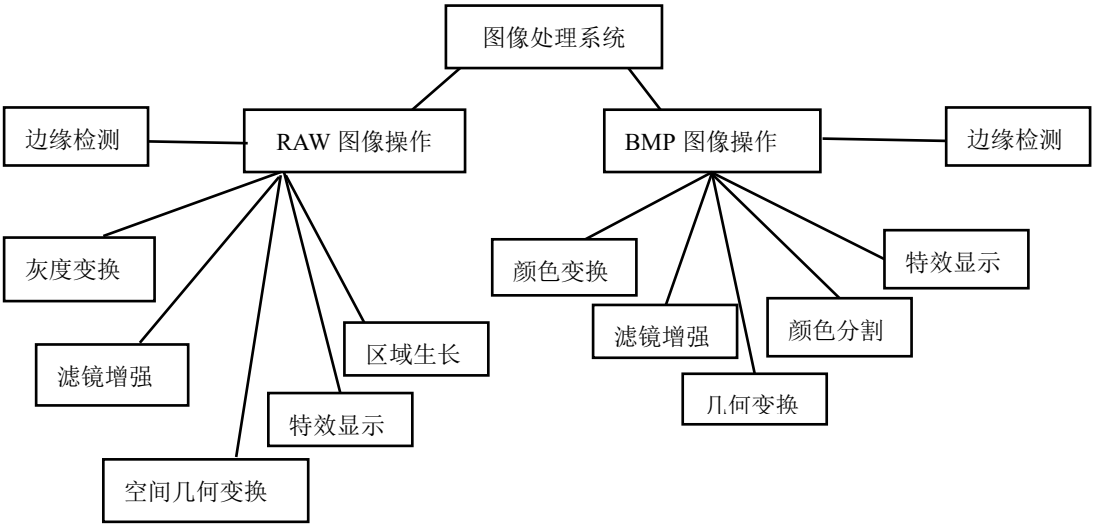
## 1.1 意义及目标

意义：使用 C++编程，通过所学的知识对图像进行处理，实现图像的颜色变换、滤镜增强、空间几何变换等功能模块，加深对图像处理的理解，掌握图像处理的方法。通过图像处理，可以提升图像的视感质量，能够提取图像同所包含的某些特征或者特殊信息，为计算机分析图像提供便利。

目标：通过 C++实现图像的颜色变换、滤镜增强、空间几何变换等多个模块，、理解和掌握图像变换、图像增强、边缘检测、几何变换等多种图像处理的方法。

## 1.2 主要功能模块架构

在本项目中，主体框架为 MFC 图形界面，在框架中分为 RAW 图像操作和 BMP 图像操作模块，在 RAW 模块中，一共包含边缘检测、灰度变换、滤镜增强、特效显示和空间几何变换 5 个基本模块，区域生长作为本模块的特色功能。在 BMP 图像操作模块中，包含颜色变换、滤镜增强、边缘检测、特效显示、几何变换 5 个基本模块，颜色分割模块作为本项目的特色模块。




# 2 项目基本功能模块

## 2.1 RAW 图像操作模块

### 2.1.1 Gamma 校正

Gamma 校正本质上使幂次变换，它的原理如下：首先对图像灰度数组进行遍历，对每一个灰度值进行归一化处理；归一化处理是将灰度值转换为 0-1 之间的实数；归一化处理之后，对每个像素点进行预补偿，即对像素点进行幂次运算，当幂次运算的幂值大于 1 的时候，Gamma 校正之后的图像比原图像偏白，而幂值小于 1 的时候，校正后的图像会偏暗的；经过预补偿之后，再对每个点的

灰度值进型反归一化处理，即将归一化后的实数转换为 0-255 之间的实数。核心代码如下：

```
void Gamma(char* oImage, char* nImage, int wImage, int hImage) {
    for (int i = 0; i < wImage * hImage; i++) {
        double Nor = (BYTE)oImage[i] / 255.0;//归一化
        double Pre = pow(Nor, 1 / 0.7);     //预补偿
        nImage[i] = (BYTE)(Pre * 255);      //反归一化
    }
}
```

图 1 灰度图 Gamma 校正

### 2.1.2 位图分割

位图切割实际上是对为平面进行二值化处理，将一半区间的灰度值置为 0，将另外一半区间的灰度值置为 255。位从高到低依次降低的时候，位图切割出来的大小是逐渐减小的，而区间数是呈指数增长的。因此只需要将位平面从高到低依次变换灰度值并将图像打印出来即可完成位图的切割。核心代码如下：

```
void BitCut(char* oImage, int wImage, int hImage) {
    for (int i = 1; i <= 8; i++) {
        int Bit = 256 / (pow(2, i));
        char* Image = new char[1024 * 1024];
        strcpy(Image, oImage);
        for (int j = 0; j < pow(2, i - 1); j++) {
            for (int m = 0; m < wImage * hImage; m++) {
                if ((BYTE)Image[m] < (BYTE)(((2 * j) + 1) * Bit)
                    && (BYTE)Image[m] >= (BYTE)((2 * j) * Bit)
                    && (BYTE)Image[m] != (BYTE)0
                    && (BYTE)Image[m] != (BYTE)255) {
                    Image[m] = (BYTE)0;
                }
                else if ((BYTE)Image[m] >= (BYTE)(((2 * j) + 1) * Bit)
                    && (BYTE)Image[m] < (BYTE)(((2 * j) + 2) * Bit
                        && (BYTE)Image[m] != (BYTE)0
                        && (BYTE)Image[m] != (BYTE)255)) {
                    Image[m] = (BYTE)255;
                }
            }
        }
        ShowImage(Image, wImage, hImage, XPOS + 300 * (i / 3), YPOS + 300 * (i % 3));
    }
}
```

图 2 位图切割

### 2.1.3 动态模糊

动态模糊也称运动模糊，可以通过只在一个方向模糊达到，在这里我使用的是 9*9 的运动模糊滤波器，值得注意的是，在灰度进行滤波器预算过后需要除以 9 才能得到想要的效果，这个效果就像是摄像机从一个方向移动到另一个方向缩排出来的效果。滤波模板如下：

$$K_n = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

### 2.1.4 图像动态显示

通过控制图像的输出顺序即可控制图像的动态显示，例如在开门特效中，只需要将图像同中间分别向两边同时输出即可得到开门特效显示，以此类推，通过控制图像的输出可以实现图像的开关门特效显示、百叶窗显示、淡入淡出效果显示等特效显示的效果。

### 2.1.5 其他

其他的图像操作在前面的小实验中均已实现过，因此在这里不做过多赘述。

## 2.2 BMP 图像操作模块

### 2.2.1 灰度化

由于 BMP 图像为 24 位真彩色图像，与 RAW 格式的图像不同，它每一个像素点含有三个通道，因此进行灰度化之前，我们需要对图像中每个像素点的 RGB 三个通道分别保存到指定的数组中，然后通过灰度转换公式，依次将每一个像素点转换为灰度值。公式如下：

$$G = 0.3r + 0.59g + 0.11b$$

### 2.2.2 图像反转

BMP 图像的图像颜色反转与 RAW 图像类似，RAW 只需要将灰度值进行反转即可，而 BMP 图像则需要分别对 R、G、B 三个通道进行反转处理，反转处理的方法也很简单，只需要使用 255 减去每个像素点对应的值，将反转后的新的三通道数组进行输出处理。

$$New = 255 - Old$$

### 2.2.3 锐化

图像锐化是补偿图像的轮廓，增强图像的边缘及灰度跳变的部分，使图像变得清晰。图像锐化是为了突出图像上地物的边缘、轮廓，或某些线性目标要素的特征。这种滤波方法提高了地物边缘与周围像元之间的反差，因此也被称为边缘增强。BMP 图像锐化可以使用锐化卷积核模板分别处理 RGB 三个通道，然后将处理过后的图像显示即可。锐化 3*3 卷积模板：

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

### 2.2.4 边缘检测

首先对图像使用公式 G=0.3r+0.59g+0.11b 进行灰度化处理，灰度处理完成之后，使用算子对图像进行卷积运算，即可得到图像的边缘效果。同时在边缘检测需要注意的是，如果需要同时进行 XY 双方向的边缘检测，需要在卷积运算完成之后得到的值进行几何平均处理。以下是边缘检测算子的模板。

$$Lapcian = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$SobelX = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} SobelY = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$PrewittX = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} PrewittY = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$RobertsX = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} RobertsY = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

### 2.2.5 其他

其他的图像操作的实现方法均与 RAW 图像类似，只需要将 RAW 处理时的灰度值换成 RBG 三通道进行处理即可，因此在这里不做过多赘述。

# 3 特色功能模块

## 3.1 区域生长

区域生长是把一幅图像分成许多小区域，在本次大作业中，我将图像分割成单个像素。然后将某一像素作为种子点，对其所有邻域进行运算，若邻域中有像素与种子相似，则该像素数与该区域，依次重复上述过程，知道图像中所有区域的点考查完毕为止。实现方法：首先对定义一个种子点，定义队列数据结构，将种子点进行入队操作，定义一个标识数组来对像素点的遍历与否进行标记，在非队空条件下进行循环，依次遍历种子点的邻域，若邻域位相似点，则对该点进行入队操作，使用 NewImage 数组将该点对应的坐标的灰度值进行存储，然后对该点进行标记，在邻域遍历完成之后将队头元素弹出，然后再对下一个对头元素的坐标点进行遍历。核心代码：

```
void Growth(char* oImage, char* nImage, int wImage, int hImage) {
    for (int i = 0; i < wImage * hImage; i++)
        nImage[i] = (BYTE)0;
    int point = wImage * hImage / 2;
    nImage[point] = (BYTE)oImage[point];
    queue<int>points;
    points.push(point);
    int k = 1;
    while (!points.empty()) {
        int p = points.front();
        int m, n;
        if (p / wImage - 1 >= 0 && p / wImage + 1 <= hImage) {
            for (m = -k; m <= k; m++) {
                for (n = -k; n <= k; n++) {
                    if (abs((BYTE)oImage[p] - (BYTE)oImage[p + (m * wImage) + n]) < 8 && (BYTE)nImage[p + (m * wImage) + n] == 0) {
                        nImage[p + (m * wImage) + n] = (BYTE)oImage[p + (m * wImage) + n];
                        points.push(p + (m * wImage) + n);
                    }
                }
            }
        }
        points.pop();
    }
}
```

图 3 区域生长算法

## 3.2 颜色分割

在本次大作业中，进行颜色分割的对象是 bmp 图像，其色彩空间为 RGB 三通道，因此可以分别对三个通道进行颜色分割。以 R 通道分割为例，首先将 R 通道的值加起来，然后对 R 通道取平均值，以该值作为颜色分割的阈值，以该阈值为标准，低于该阈值的三个通道均置为 0，高于该阈值的点保留三个通道的值。核心代码：

```
void BmpR(int wImage, int hImage) {
    double Rvector = 0;
    for (int i = 0; i < wImage * hImage; i++)
        Rvector += (BYTE)R[i];
    Rvector = Rvector / (wImage * hImage);
    for (int i = 0; i < wImage * hImage; i++) {
        if ((BYTE)R[i] > Rvector) {
            RNew[i] = (BYTE)R[i];
            GNew[i] = (BYTE)G[i];
            BNew[i] = (BYTE)B[i];
        }
        else{
            RNew[i] = (BYTE)0;
            GNew[i] = (BYTE)0;
            BNew[i] = (BYTE)0;
        }
    }
}
```

图 4 R 通道颜色分割

# 4 项目效果图

## 4.1 RAW 功能模块

图 5 参数为 0.7 的伽马校正

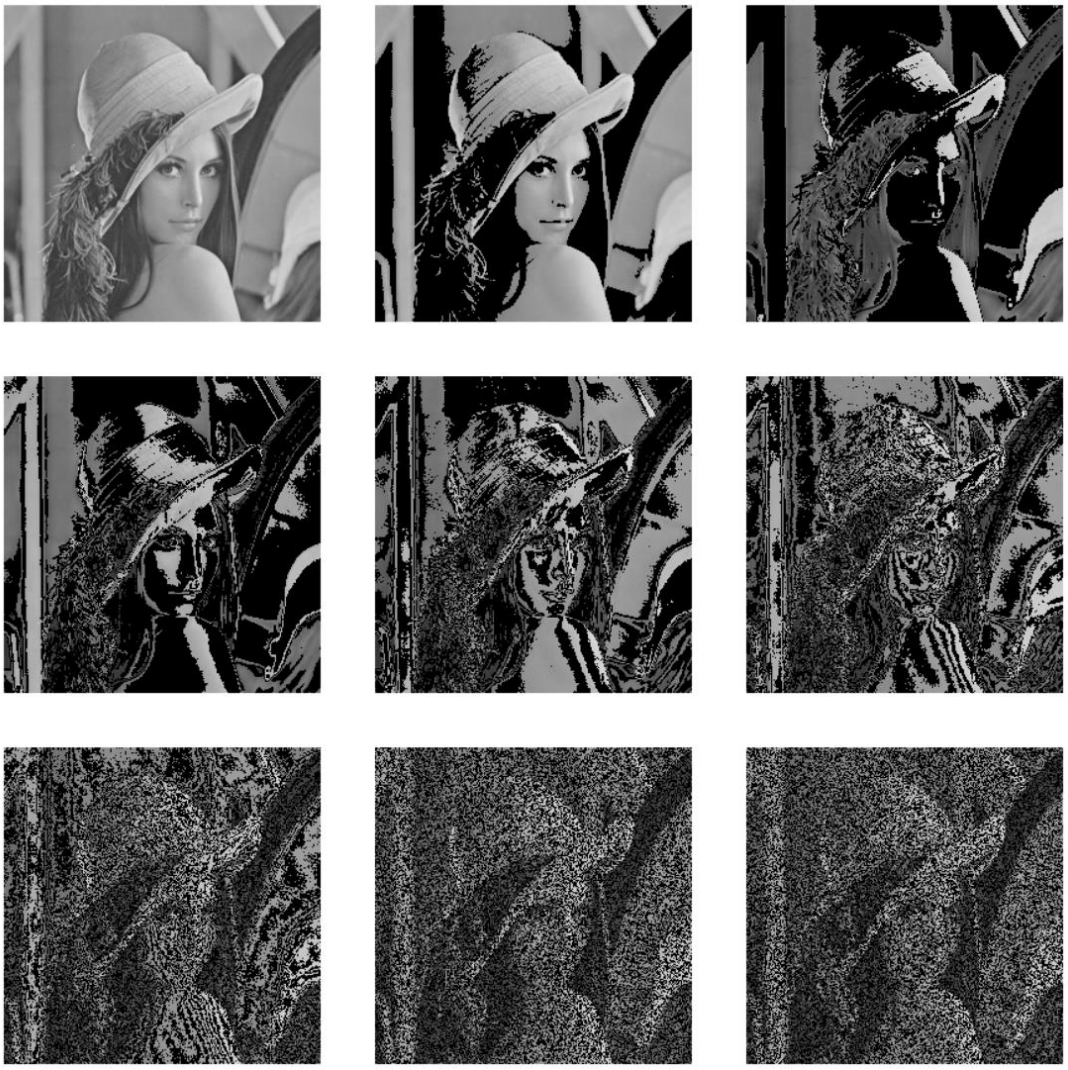


图 6 位图切割

## 4.2 BMP 功能模块

图 7 BMP 图像灰度化

图 8 动态模糊

图 9 颜色反转

## 4.3 特色模块

图 10 阈值为 8 的区域生长


图 11 对 R 通道进行颜色分割

# 5 项目总结与问题分析

在本项目中，在原来实验的基础上新增实现了灰度图的区域生长、Gamma 校正等算法，除此之外，还实现了 BMP 图像的多种滤镜和颜色分割算法。在实现 RAW 图像区域生长算法是时候，出现了内存超限，运行时间长和没有显示效果图的现象，观察 ide 的资源管理窗口，发现程序在执行区域生长函数的时候，内存占用直线上升，通过分析可以判断出程序在该函数中陷入了死循环；经过仔细思考，发现没有对遍历过的邻域进行标记，导致相同邻域点多次进入队列，出现了永远达不到队空、陷入死循环的状态，因此需要使用一个状态数组来对遍历过的点进行标记，经过标记后的点，在遍历邻域过程中只入队一次，能够在遍历执行之后跳出循环，最终成功实现。除此之外，在实现 BMP 各种操作的算法的时候，对 BMP 的存储不是很了解，并不知道能够对三个通道分别进行操作，因此在一开始实现 BMP 的高斯平滑的时候，采用了先转换为灰度图在等比例还原 bmp 图像的方式，因此出现了通道值越界的问题，导致整张图片变为白色，在经过查阅资料过后，了解到可以分别对 RGB 三个通道进行运算，然后在将 RGB 数据转化到对应点中，在修改过后，成功实现了 BMP 图像的各种操作。

# 6 项目心得

通过本次项目，在原本实验的基础上，增加实现了多种操作，大大加深了我对图像处理方法的理解，在图像处理技术日益先进的今天，在“数字图像处理”这门课所学的知识也依旧不过时，随着人工智能的发展和神经网络的出现，图像处理变得越来越智能，但是它们对图像的处理的算法依旧是源于数字图像处理这这个学科的。我在这个项目中、从这一门有趣的课程中，学到了很多很实用的算法，在本次项目中通过理论与实验相结合，让我更透彻地理解了数字图像处理，从实验 1 的代码框架补充到现在的接近 2000 行代码，一步一步实现每一个功能，也让我感到了巨大的成就感，总体而言，这门课程对于我来说是十分实用的。

**参考文献**

[1] 飞狼，李春萌，杨涵．数字图像处理．北京：清华大学出版社，2007　　（书）
[2] Microsoft MSDN Library，http://msdn.microsoft.com/library/chs/　　（网页）
[3] 飞狼，李春萌，杨涵．等．粒子对算法在图像矢量量化中的应用．电子学报，2007，38(10):1916-1920．（期刊论文）