作业完整基础知识部分

1、 什么是数字图像，请举例说明，并说明其中 f(x , y)表示的含义

答：数字图像是由像素组成的二维排列，可以用矩阵表示，比如数码相机拍的照片，其本质是像素阵列，每一个像素包含具体色彩等信息。f(x , y)是在空间坐标下点(x , y)的幅值，拿上一个例子来说明就是照片上(x , y) 的点的色彩信息。对于灰度图像是一个 0-255 的亮度值；对于彩色图像就是 RGB 三个色彩分量组成的值。

2、 数字图像处理包括哪几个层次？说明并举例

i. 低级处理：得到图像输出图像，例如对图像去噪，增强等。

ii. 中级处理：得到图像输出对应属性，例如目标识别，场景分割。

iii. 高级处理：得到图像输出对应理解，例如场景理解，导航。

3、 数字图像处理应用在哪些领域，请举例说明？

图像增强/恢复、艺术效果、医学可视化、工业检测、人机界面

医学、空间应用、地理学、生物学、军事、航空航天等

数码相机(DC)、数码摄像机(DV)

指纹识别、人脸识别

互联网、视频、多媒体等

基于内容的图像检索、视频检索、多媒体检索

游戏、电影特技、虚拟现实、电子商务等。

举例：比如说指纹识别，是对图像中指纹信息进行识别后与数据进行对比核对来实现验证身份的功能。再比如说医学可视化中有器官边缘识别、细胞识别等图像增强技术，使医学图像更清晰。再例如 GIS 是地理学中图像识别的应用，识别不同地形并进行分类标识，处理星云图使其更清晰可见。

4、 数字图像处理包括哪几个关键阶段？

图像采集->图像增强->图像复原->形态学处理->图像分割->目标识别->表达及描述

最后还有彩色图像处理和图像压缩两个选择。

5、 在灰度图像中，灰度值为 0 表示是什么颜色？255 灰度值呢？在彩色图像中，颜色（255,0,0）表示什么颜色？（0,255,0）颜色呢？（0, 0,255）颜色呢？

0 表示黑色，255 表示白色。

（255,0,0）表示红色；（0,255,0）表示绿色；（0, 0,255）表示蓝色

6、 数字图像处理的两种主要方法是什么？

空域法和变换域法

空域法是把图像看作是平面中各个像素组成的集合，然后直接对这个二维函数空间的像素进行相应的处理，其包括领域处理法和点处理法。

变换域法首先对图像进行正交变换，得到变换域系数阵列，然后再施行各种处理；处理后再反变换到空间域，得到处理结果。其中包括：滤波、数据压缩和特征提取等处理。

W02

1、什么采样?什么是量化?

采样是将连续的图像空间的电压转换成离散采样集合的操作。

量化就是对图像函数值（灰度值或颜色值）的数字化处理。

2、简单叙述 256 个灰度等级图像的量化过程。

1. 得到连续的模拟信号。
2. 根据采样的结果在模拟信号波形上取值。
3. 对从黑到白分为 256 个灰度值，然后将波形上的取值对应到 256 个等级上。
4. 量化完成。

3、对于分辨率为 640×480 的图像来说，共有多少行像素？有多少列像素？

答：480 行，640 列。

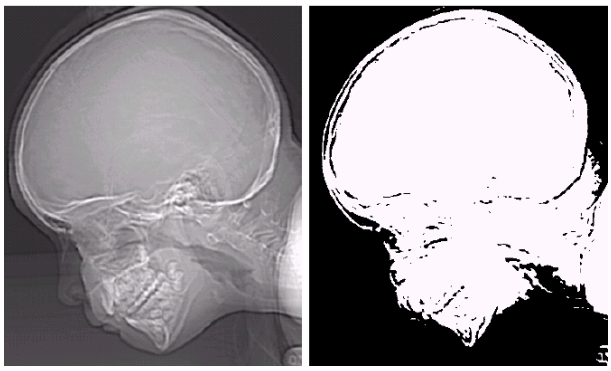
4、下面两幅图像中，哪个质量较高？为什么？

(a) 512X512

(b) 128X128

答：A 好，因为 A 的分辨率高，清晰。

5、下面两幅图中，第一幅是 256 个灰度级，第二个是 2 个灰度级，哪一个质量较高？为什么？



a) (b)

答：A 好，因为 A 的灰度级高，图像描述的细节越精细。

6、像素 p(3,10)和 q(6,12)之间的棋盘距离 D8 为多少？ 请写出计算过程。

答；D=3

7、影响数字图像质量的因素有哪几种？

答：空间分辨率、强度等级分辨率（灰度等级数目）、对比度、清晰度

W03

1、 常用的图像代数运算有哪几种？举一个例子，简单说明加法运算的用途。

Ans：两幅图像对应像素的加、减、乘、除运算。

加法运算用途主要是合成图像（将几幅图像叠加在一起再做处理） 和消除图像的随机噪声（同一场景的图像相加后取平均）

2、 常用的图像逻辑运算有哪几种？举一个例子，简单说明与运算的用途。

Ans：AND, OR 和补运算。

在灰度化和二值化处理之后，几幅图与运算之后可以得到各副图的公共突出部分。

3、 常用的图像几何运算有哪几种？举一个例子，简单说明旋转运算的用途。

Ans：平移变换、旋转变换、缩放、镜像变换

旋转运算顾名思义就是按照某一个点旋转某个角度。例如：


->


4、 灰度图像进行二值化处理，怎样做？简单阐述。二值化处理的结果与阈值有关吗？为什么？

Ans：

$$s = T(r) = \begin{cases} 255 & \text{if } A \le r \le B \\ 0 & \text{otherwise} \end{cases}$$

选定二值化区间[A,B]，然后将图像像素点的灰度值 r 带入公式得到对应的二值化的灰度值。其中一般 B 设定为 255，A 设定为 threshold。

二值化处理的结果与上述提及的 threshold 有关。在上述的模式中，thres 越高，灰度图像保留的较白区域越多。反之保留的越少。

5、 彩色图像进行二值化处理，怎样做？简单阐述。

Ans：Gray = R*0.299 + G*0.587 + B*0.114

首先将彩色图像各个像素点的彩色分量带入上述公式中的（R，G，B）三个参数中，然后得到对应的灰度图像，再应用 4 中的方法得到二值化处理的图像

6、在图像平移变换中，如果一幅图像中的像素坐标为（20,30），原来图像的原点平移到（5,12），该像素在新的坐标系统中的坐标怎样计算？是多少？

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\because \begin{cases} \Delta \mathrm{X} = 5 \\ \Delta \mathrm{Y} = 12 \\ X = 20 \\ y = 30 \end{cases}$$

$$\therefore \begin{cases} X' = 25 \\ y' = 42 \end{cases}$$

Ans:结果坐标为（25，42）

7、如果原来图像的分辨率是 50×120，其上有一个像素的坐标为（20,60），该图像旋转 30 度，计算该像素在旋转后图像中的坐标。写出计算过程。

Ans:假设顺时针旋转

$$\begin{cases} \mathrm{x}_0 = \mathrm{x} - {}^{\mathrm{w\_old}}/_2 = 20 - \dfrac{50}{2} = -5 \\ \mathrm{y}_0 = \mathrm{y} - {}^{\mathrm{h\_old}}/_2 = 60 - \dfrac{120}{2} = 0 \end{cases}$$

$$\begin{bmatrix} \chi_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \chi_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 30° & -\sin 30° & 0 \\ \sin 30° & \cos 30° & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -4.33 \\ -2.5 \\ 1 \end{bmatrix}$$

$$\begin{cases} \mathrm{x}_2 = \chi_1 + {}^{\mathrm{w\_new}}/_2 = -4.33 + 50.1 = 45.77 \\ \mathrm{y}_2 = \mathrm{y}_1 + {}^{\mathrm{h\_new}}/_2 = -2.5 + 64.3 = 61.8 \end{cases}$$

∴得到最后的坐标为（46，62）

8、如果对图像进行水平镜像，原来图像的分辨率是 50×120，其上有一个像素的坐标为（30,70），镜像后该像素的坐标为多少，怎样计算？按照这种方法，怎样将一幅图像进行水平镜像？

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\because \begin{cases} w = 50 \\ X = 30 \\ y = 70 \end{cases}$$

$$\therefore \begin{cases} X' = 20 \\ y' = 70 \end{cases}$$

Ans:结果坐标为（20，70）;编程中，方法是将计算出像素对应的镜像位置，然后它和镜像位置的像素交换，此方法只需要循环半边像素，也就是分辨率/2 次。

W04

1、 简单说明利用前向映射法对图像进行放大处理的主要思想。

Ans:

a) 通过输入图像像素位置, 计算输出图像对应像素位置;
b) 将该位置像素的灰度值按某种方式分配到输出图像相邻四个像素.
c) 向后送颜色
d) 在前向映射中，目标图像中经常会出现空白点，使用行插值或列插值方法进行处理
   - 行插值是指任何一个空白像素的灰度值或颜色分量取它的左右相邻的非空白像素的灰度或颜色分量的平均值；
   - 列插值是指任何一个空白像素的灰度值或颜色分量取它的上下相邻的非空白像素的灰度或颜色分量的平均值。
   - 必要时要用几次行插值或列插值进行处理。

2、 简单说明利用后向映射法对图像进行放大处理的主要思想。

Ans:

a) 通过输出图像像素位置, 计算输入图像对应像素位置;
b) 根据输入图像相邻四个像素的灰度值计算该位置像素的灰度值.
c) 向前取颜色

3、 我们对图像进行放大处理时，经常采用前向映射法，还是后向映射法，为什么？

Ans：采用向后映射法，因为向后映射法不会出现空白像素，经常被采用。而且输出图像的灰度只要经过一次运算。

4、 在利用后向映射对图像进行放大处理时，采用双线性插值的方法，如果放大倍数为4倍，目标图像上的一个像素坐标为（23,45），对应源图像的四个相邻像素的颜色值分别为：

A：C(5,11)=(80,100,10)
B: C(6,11)=(30,10 ,20)
C: C(6,12)=(10,200 ,40)
D : C(5,12)=(10,30 ,20)

请计算目标图像上该像素的颜色应该是多少？

Ans：

$$f(i+u,j+v) = (1-u)(1-v)f(i,j) + (1-u)vf(i,j+1) + u(1-v)f(i+1,j) + uvf(i+1,j+1)$$

$$u = 0.75, v = 0.25$$

$$i = 5, j = 11$$

$$\therefore \begin{cases} R = 34.375 = 34 \\ G = 63.75 = 64 \\ B = 21.875 = 22 \end{cases}$$

根据这一思想, 请简单说明利用后向映射对图像进行放大处理的主要步骤。

Ans：

a) 在上面的实例中，从目标图像的每一个出发，计算其在源图像中的坐标；
b) 利用上述双线性插值方法，取得其颜色值；
c) 重复步骤b，图像放大处理就实现了。

5、 简单说明一下三角形区域变换处理的主要步骤和做法。

a) 假设有两个 2D 三角形: ABC 及 A’B’C’。

b) 需要求得变换矩阵:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

c) 将 ABC 三对对应点的坐标代入上面公式。

d) 可以求得变换的参数 a、b、c、d、e、f。

e) 对于三角形内和边界的任意一个像素点，再计算其新的坐标，然后进行颜色映像。

W05

无

W06

1、 什么是图像增强? 常用的图像增强方法有哪些?

图像增强是指对图像的某些特征，如边缘、轮廓、对比度等进行强调或尖锐化。可以改善图像的视觉效果，或将图像转换成一种更适合于人或机器进行分析处理的形式。常用的图像增强方法：对比度增强、图像平滑、图像锐化 、同态滤波等

2、 图像增强的手段有哪两大类?简单说明.

a) 空域处理：是在图像平面上处理，这类方法是以对图像的像素直接处理为基础的。
    1. 全局运算：在整个图像空间域进行
    2. 局部运算：在与象素有关的空间域进行
    3. 点运算：对图像作逐点运算

b) 频域处理：以修改图像的傅氏变换为基础的。
    1. 在图像的变换域中进行处理

3、 空域增强处理有哪两大类方法?

a) 空域增强处理的方法主要分为
    i. 点处理:作用于单个像素的空间域处理方法,包括图像灰度变换、直方图处理、伪彩色处理等技术;
    ii. 模板处理两大类：作用于像素邻域的处理方法，包括空域平滑、空域锐化等技术。

4、 什么是点处理？有哪几种点处理方法？

a) 通过图像中每个像素点的灰度值进行计算，改善图像显示效果。
b) 点运算将输入图象映射为输出图象,输出图象每个象素点的灰度值仅由对应的输入象素点的值决定。
c) 点运算因其作用性质有时也被称为对比度增强、对比度拉伸或灰度变换
d) 点处理可以分为：
    i. 线性点运算
    ii. 非线性点运算

5、 直方图有哪两种表示方法？

方法 1：直方图表示:

$$\mathrm{h(k)} = num_k$$

k 表示图像的灰度值，num_k表示灰度值为k的像素的个数。

方法 2：归一化表示：

概率

$$p_r(k) = num_k/num$$

$p_r(k)$表示灰度级为k的像素的统计概率,num表示图像中像素的总数。

6、 在图像伪彩色处理时，常用强度分层法和灰度级变换法，请分别简述这两种伪彩色处理的主要思想。

强度分层技术是将图像的灰度值，用 L 个不同的高度进行截取，从而可以使灰度分 布到 L+1 个间隔中；

灰度级到彩色变换方法是对任何输入像素的灰度级执行三个独立的变换。三个变换结果分别送入彩色电视监视器的红、绿、蓝通道。这种方法产生一幅合成图像，其彩色内容受变换函数特性调制。

W07

1、 请简述均值滤波的主要思想。

取邻域内的所有像素的平均值作为当前像素的值，可以分为领域均值滤波器和加权均值滤波器，常用 4 邻域或者 8 邻域滤波器。

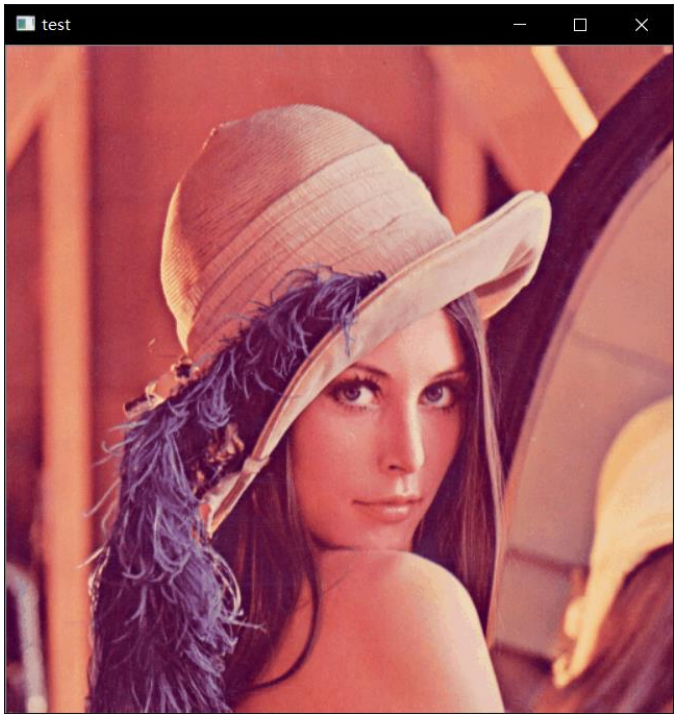
2、 请简述加权平滑滤波的主要思想。

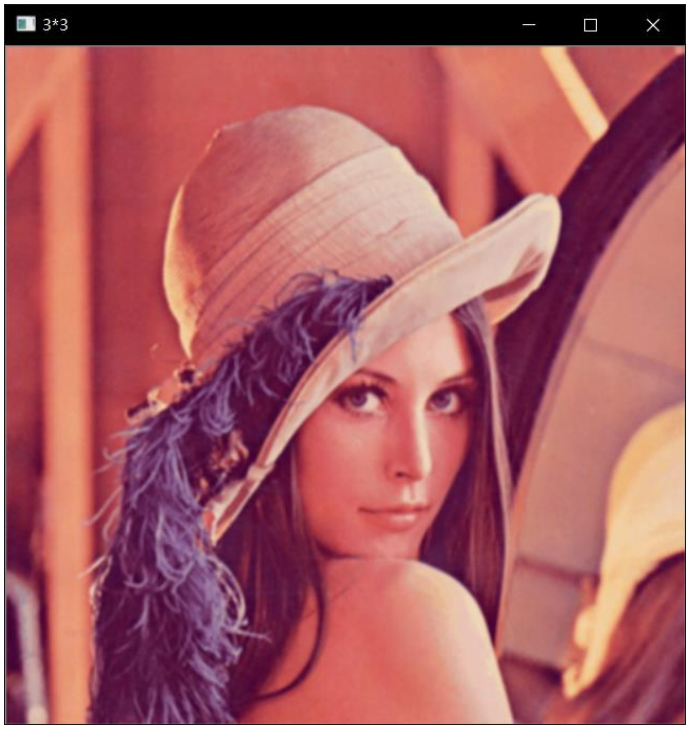
为了使均值滤波器更有效，可以对领域内不同的像素采用不同的权值，体现出离中心像素较近的像素的重要性。

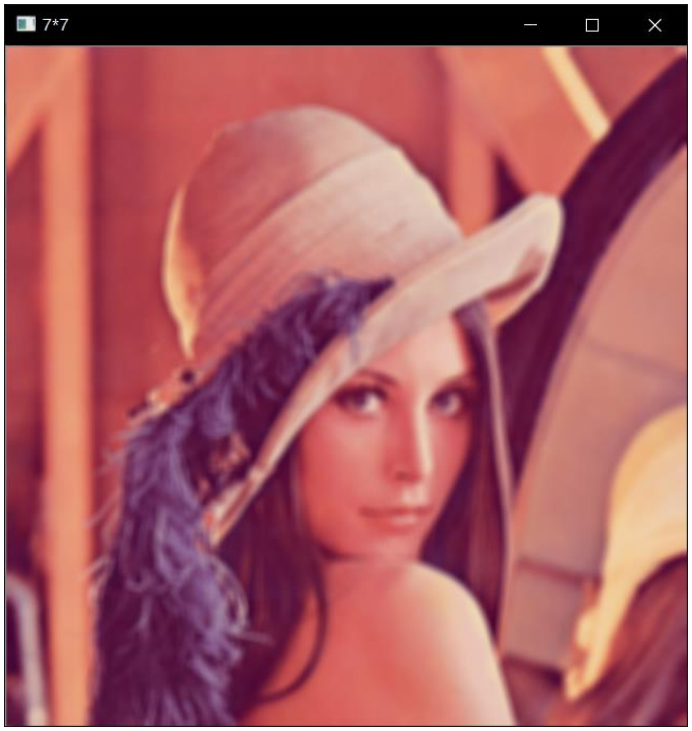
3、 请简述中值滤波的主要思想。

中值滤波法是将当前像素点的灰度值取为邻域窗口内的所有像素点灰度值的中值，属于非线性平滑方法。

4、在空域中对图像的平滑处理时，邻域的尺度选取太大，结果会怎么样？你试一试，如果邻域 7X7，结果会怎么样？







可以看到在用 7*7 的模板对图像平滑过后产生了强烈的模糊效果

W08

1. 请分别用 Sobel 算子、Prewitt 算子、 Robert 算子对下列图像进行运算。

2.	3	1	2	8	2	3
3.	1	3	16	8	3	2
4.	7	2	17	18	9	2
5.	0	1	7	19	2	1
6.	1	2	4	20	1	3

并简单对结果分析一下。

分别利用 Sobel、Prewitt、Robert 的水平边检测和垂直边检测算子做检测，保留得到的每个点两个值中较大的值。其中边缘的处理方法选择不保留的解决方案。

Sobel 结果：

00, 00, 00, 00, 00, 00,
00, 39, 41, 42, 33, 00,
00, 42, 55, 34, 56, 00,
00, 27, 70, 21, 69, 00,
00, 13, 54, 47, 52, 00,
00, 00, 00, 00, 00, 00,

Prewitt 结果：

00, 00, 00, 00, 00, 00,
00, 24, 28, 32, 27, 00,
00, 32, 39, 26, 40, 00,
00, 20, 52, 19, 51, 00,
00, 10, 36, 28, 35, 00,
00, 00, 00, 00, 00, 00,

Robert 结果：

00, 15, 06, 05, 00, 00,
01, 14, 02, 01, 01, 00,
06, 05, 02, 16, 08, 00,
02, 03, 13, 18, 01, 00,
01, 02, 04, 20, 01, 00,
00, 00, 00, 00, 00, 00,

2、空域中对图像进行锐化处理一共几种方法，简单总结怎样用它们进行锐化处理。

a) 空间差分：用于衡量函数的变化率
b) 微分滤波器：用一阶微分滤波器或二阶，设定阈值检测梯度大小来判断是否为边缘点
c) 边缘增强算子：包括用 Robert 算子、Prewitt 算子、Sobel 算子、拉普拉斯算子、高斯拉普拉斯算子等对边缘点判断。
d)

3、在进行锐化处理时，什么时候需要选择阈值？利用阈值处理后有什么好处？

- 是在微分或者边缘增强算子得到结果后对是背景还是边缘需要用阈值进行判断。
- 一个好的合适的阈值可以很好的分辨出边缘点，又保留原图像和背景不被破坏。

4、用 Prewitt 算子或 Sobel 算子进行边缘检测时，算子具有方向性，有的水平方向的，有的是垂直方向的，怎样结合它们得到完整的一个图像边缘？请编程举例说明，并用结果证明你的方法的正确性。

- 分别利用 Sobel 或 Prewitt 的水平边检测和垂直边检测算子做检测，保留得到的每个点两个值中较大的值做为最后的结果。采用阈值得到边缘。
- 对应代码：

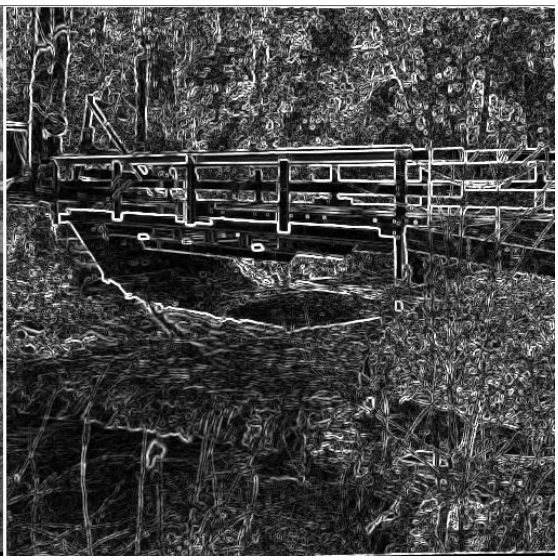
```
float aTemplate[4];
//template01
aTemplate[0] = 1.0;
aTemplate[1] = 0;
aTemplate[2] = 0;
aTemplate[3] = -1.0;
int _tab1[36] = { 0 };
 Template2X2(tab1, aTemplate,_tab1);
// template02
 aTemplate[0] = 1.0;
 aTemplate[1] = 0;
 aTemplate[2] = 0;
 aTemplate[3] = -1.0;
int _tab2[36] = { 0 };
Template2X2(tab2, aTemplate, _tab2);
//choose the larger one
for (int j = 0; j < 6; j++)
{
    for (int i = 0; i < 6; i++)
    {
        int color1, color2, color;

        color1 = _tab1[j * 6 + i];


        color2 = _tab2[j * 6 + i];

        if (color2 > color1)
        {
            color = color2;
        }
        else
            color = color1;
        cout.fill('0');
        cout.width(2);
        cout << color << ", ";

    }
    cout << endl;
}
```

- 或者可以利用两个值的平方和来阈值作比较得到最后的图像

5、在利用各种算子检测边缘时，得到的边缘结果有什么特点？请编程举例说明？
梯度方法:因为梯度得到的对边缘和噪声的区别不敏感,所以保留了很多嘈杂的信息

```
void CwO7View::OnLadderDoit()
{
    // TODO: 在此添加命令处理程序代码

    int thresh =70;

    mybmp_output.CreateCDib(CSize(sizeDibDisplay_1.cx, sizeDibDisplay_1.cy), 24);
    sizeDibDisplay_output = mybmp_output.GetDimensions();
    for (int i = 0; i < sizeDibDisplay_1.cy - 1; i++)
    {
        for (int j = 0; j < sizeDibDisplay_1.cx - 1; j++)
        {
            RGBQUAD color1, color2, color3, color;
            // 指向DIB第i行，第j个象素的指针
            color1 = mybmp_1.GetPixel(j, i);

            // 指向DIB第i+1行，第j个象素的指针
            color2 = mybmp_1.GetPixel(j, i + 1);

            // 指向DIB第i行，第j+1个象素的指针
            color3 = mybmp_1.GetPixel(j + 1, i);

            int bTemp = abs(color1.rgbBlue - color2.rgbBlue) + abs(color1.rgbBlue -

            // 判断是否小于阈值
            // 判断是否大于阈值，对于小于情况，灰度值不变。
            if (bTemp >= thresh)
            {
                if (bTemp < 255)
                {
                    color.rgbGreen = bTemp;
                    color.rgbRed = bTemp;
                    color.rgbBlue = bTemp;
                    mybmp_output.WritePixel(j, i, color);
                }
                else {
                    color.rgbGreen = 255;
                    color.rgbRed = 255;
                    color.rgbBlue = 255;
                    mybmp_output.WritePixel(j, i, color);
                }
            }
            else {

                //mybmp_output.WritePixel(j, i, mybmp_1.GetPixel(j,i));
            }
        }
    }
    Invalidate(true);
}
```

Prewitt 算子:Prewitt 算子具有方向性,在采用两个方向取大者的方法之后取消了方向性,效果比 Robert 算子好,但是卷积模板是没有权重的.



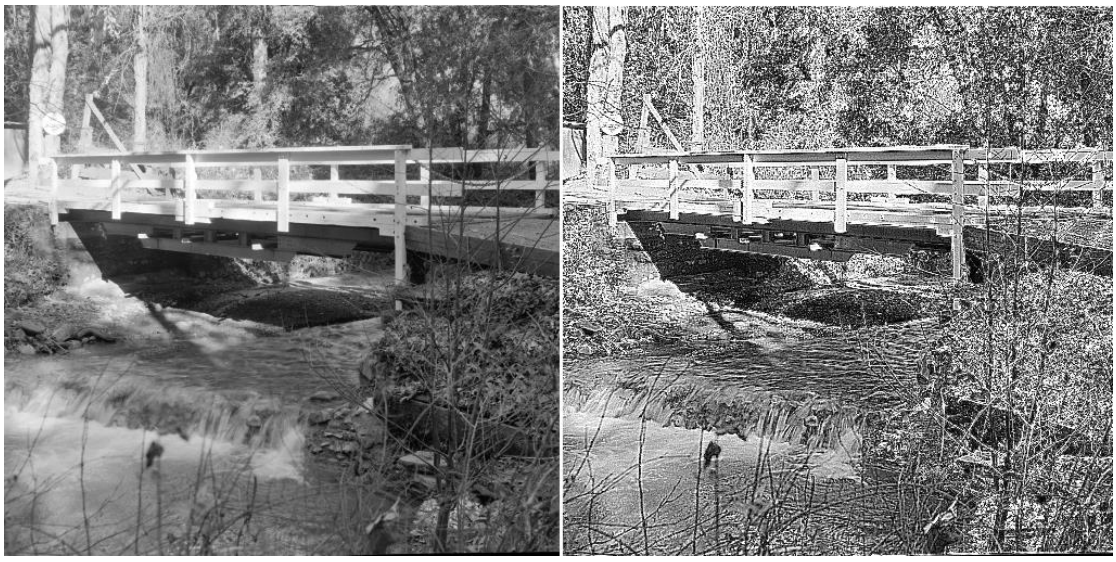
```
mybmp_output.CreateCDib(CSize(sizeDibDisplay_1.cx, sizeDibDisplay_1.cy), 24);
sizeDibDisplay_output = mybmp_output.GetDimensions();

CSize sizeimage = mybmp_1.GetDimensions();
CDib tmpbmp1;
tmpbmp1.CreateCDib(sizeimage, mybmp_1.m_lpBMIH->biBitCount);
tmpbmp1.CopyDib(&mybmp_1);

CDib tmpbmp2;
tmpbmp2.CreateCDib(sizeimage, mybmp_1.m_lpBMIH->biBitCount);
tmpbmp2.CopyDib(&mybmp_1);

float aTemplate[9];         // 模板数组

                            // 设置Prewitt模板参数
aTemplate[0] = -1.0;
aTemplate[1] = -1.0;
aTemplate[2] = -1.0;
aTemplate[3] = 0.0;
aTemplate[4] = 0.0;
aTemplate[5] = 0.0;
aTemplate[6] = 1.0;
aTemplate[7] = 1.0;
aTemplate[8] = 1.0;

Template3X3(tmpbmp1, aTemplate);

// 设置Prewitt模板参数
aTemplate[0] = 1.0;
aTemplate[1] = 0.0;
aTemplate[2] = -1.0;
aTemplate[3] = 1.0;
aTemplate[4] = 0.0;
aTemplate[5] = -1.0;
aTemplate[6] = 1.0;
aTemplate[7] = 0.0;
aTemplate[8] = -1.0;

Template3X3(tmpbmp2, aTemplate);
```

```
//求两幅缓存图像的最大值
for (int j = 0; j < sizeimage.cy; j++)
{
    for (int i = 0; i < sizeimage.cx; i++)
    {
        RGBQUAD color, color1, color2;

        // 指向缓存图像1第j行，第i个象素的指针
        color1 = tmpbmp1.GetPixel(i, j);

        // 指向缓存图像2第j行，第i个象素的指针
        color2 = tmpbmp2.GetPixel(i, j);

        if (color2.rgbRed > color1.rgbRed)
        {
            color = color2;
        }
        else
            color = color1;

        mybmp_output.WritePixel(i, j, color);
    }
}
Invalidate(true);
```

Sobel 算子:与 Prewitt 算子类似,但是 Sobel 提取的边缘更明确

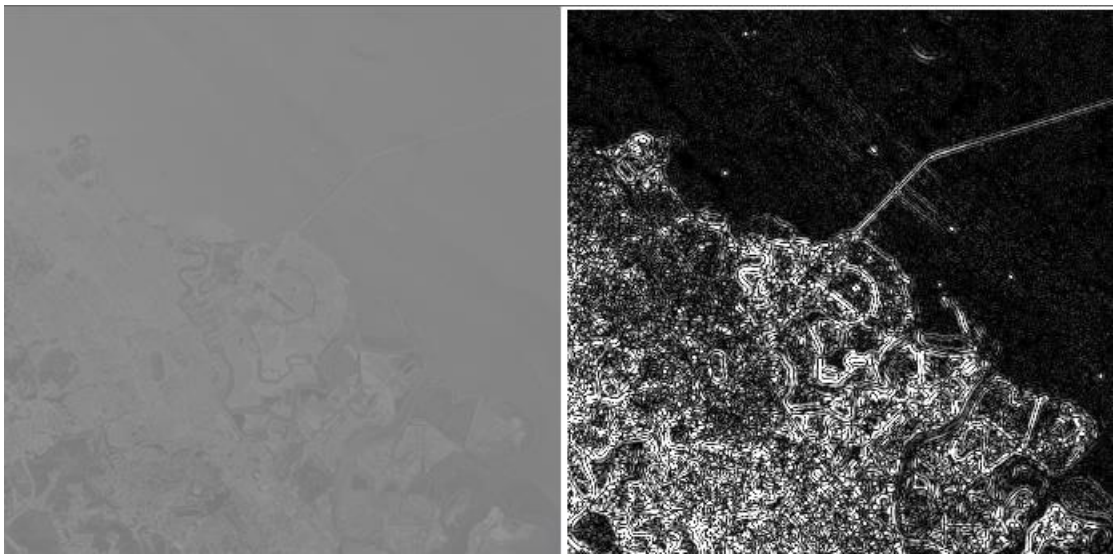
```
aTemplate[0] = -1.0;
aTemplate[1] = -2.0;
aTemplate[2] = -1.0;
aTemplate[3] = 0.0;
aTemplate[4] = 0.0;
aTemplate[5] = 0.0;
aTemplate[6] = 1.0;
aTemplate[7] = 2.0;
aTemplate[8] = 1.0;


Template3X3(tmpbmp1, aTemplate);

// 设置Prewitt模板参数
aTemplate[0] = -1.0;
aTemplate[1] = 0.0;
aTemplate[2] = 1.0;
aTemplate[3] = -2.0;
aTemplate[4] = 0.0;
aTemplate[5] = 2.0;
aTemplate[6] = -1.0;
aTemplate[7] = 0.0;
aTemplate[8] = 1.0;
```

拉普拉斯算子:是不依赖边缘方向的二阶微分算子,是一个标量.由于一阶导数的局部最大值对应着二阶导数的零交叉点,通过求图像的二阶导数的零交叉点可以找到精确的边缘点.



```
// TODO: 在此添加命令处理程序代码
mybmp_output.CreateCDib(CSize(sizeDibDisplay_1.cx, sizeDibDisplay_1.cy), 24);
sizeDibDisplay_output = mybmp_output.GetDimensions();
mybmp_output.CopyDib(&mybmp_1);


float aTemplate[9];          // 模板数组

                             // 设置Prewitt模板参数
aTemplate[0] = -1.0;
aTemplate[1] = -1.0;
aTemplate[2] = -1.0;
aTemplate[3] = -1.0;
aTemplate[4] = 9.0;
aTemplate[5] = -1.0;
aTemplate[6] = -1.0;
aTemplate[7] = -1.0;
aTemplate[8] = -1.0;


Template3X3(mybmp_output, aTemplate);
Invalidate(true);
```

LOG 算子:首先对图像进行高斯滤波去除噪声,然后求二阶导数,二阶导数为 0 的像素就是图像的边缘.LOG 算子是对拉普拉斯算子的优化,去除了拉普拉斯算子对噪声的敏感性.



```
float aTemplate[25];              // 模板数组
mybmp_output.CreateCDib(CSize(sizeDibDisplay_1.cx, sizeDibDisplay_1.cy), 24);
sizeDibDisplay_output = mybmp_output.GetDimensions();
                                  // 设置模板参数
aTemplate[0] = -2.0;
aTemplate[1] = -4.0;
aTemplate[2] = -4.0;
aTemplate[3] = -4.0;
aTemplate[4] = -2.0;
aTemplate[5] = -4.0;
aTemplate[6] = 0.0;
aTemplate[7] = 8.0;
aTemplate[8] = 0.0;
aTemplate[9] = -4.0;
aTemplate[10] = -4.0;
aTemplate[11] = 8.0;
aTemplate[12] = 24.0;
aTemplate[13] = 8.0;
aTemplate[14] = -4.0;
aTemplate[15] = -4.0;
aTemplate[16] = 0.0;
aTemplate[17] = 8.0;
aTemplate[18] = 0.0;
aTemplate[19] = -4.0;
aTemplate[20] = -2.0;
aTemplate[21] = -4.0;
aTemplate[22] = -4.0;
aTemplate[23] = -4.0;
aTemplate[24] = -2.0;
Template5X5(aTemplate);
Invalidate(true);
```

```
CDib newbmp;
CSize sizeimage = sizeDibDisplay_1;
newbmp.CreateCDib(sizeimage, mybmp_1.m_lpBMIH->biBitCount);

for (int y = 2; y < sizeimage.cy - 2; y++)  // 行(除去边缘几行)
{
    for (int x = 2; x < sizeimage.cx - 2; x++)// 列(除去边缘几列)
    {
        RGBQUAD color;

        double fResult = 0;

        for (int j = -2; j <= 2; j++)
        {
            for (int i = -2; i <= 2; i++)
            {
                color = mybmp_1.GetPixel(x + i, y + j);
                fResult += color.rgbRed * fpArray[(i + 2) * 5 + (j + 2)];
            }
        }

        fResult = fabs(fResult);    // 取绝对值
        if (fResult > 255)
        {
            color.rgbGreen = 255;
            color.rgbRed = 255;
            color.rgbBlue = 255;
            newbmp.WritePixel(x, y, color);

        }
        else {
            color.rgbGreen = (unsigned char)(fResult + 0.5);
            color.rgbRed = (unsigned char)(fResult + 0.5);
            color.rgbBlue = (unsigned char)(fResult + 0.5);
            newbmp.WritePixel(x, y, color);
        }
    }
}
mybmp_output.CopyDib(&newbmp);
```

W09

1、 图像的空域处理和频域处理的区别是什么?

空域处理：在图像的平面空间 f(x,y)上进行的处理。这样的处理是完全基于图像像素的位置和像素色彩强度的。

频域处理：通过图像信号的成分分析、并改变图像信号的成分，从而达到图像处理的目的。首先需要将图像转换到频域空间，然后分析高频成分和低频成分，接着使用滤波器等过滤、增强、削弱不同的成分，达到频域处理的效果。

2、 常用的频域平滑滤波器有哪几种?

低通滤波器:容许低频信号通过、但减弱（或减少）频率高于截止频率信号通过

- 理想低通滤波器-对一定范围内的低通信号完全保留，其他信号完全剔除。
- 巴特沃思低通滤波器-通频带内的频率响应曲线最大限度平坦，没有起伏，而在阻频带内则逐渐下降为 0，从某一边界角频率开始，振幅随着角频率的增加而逐渐减少，趋向无穷大。
- 指数形低通滤波器-可以根据阶数来调整阻频带内的下降频率。
- 高斯低通滤波器-基础为高斯函数，能构成一个在频域具有平滑性能的低通滤波器。
- 梯形低通滤波器-梯形状的滤波器，可以完全保留 R_0 范围内的低频成分。

3、 数字图像灰度分布与频谱特征之间的关系是什么?

前提：频谱中心化完成

（1）图像中灰度变化剧烈的特征在频谱中的高频区的特征加以反映。

（2）图像中的平滑区域的特性是通过频谱空间中，靠中心处的低频成分加以反映。

4、 一幅图像在频域内进行滤波的主要步骤是什么?

输入图图像->预处理->傅里叶变换->频域内进行低通滤波处理->傅里叶反变换->得到增强图像

5、 在利用各种低通平滑滤波器进行对图像进行平滑时,截止半径对平滑结果有怎样的影响?截止半径的选择应该注意哪些事项?怎样根据当前的平滑结果判断截止半径选择的好不好?

如果 d 取得太大，低通性能不好，几乎所有频率都能通过，平滑效果欠佳。如果 d 取得太小，图像则会过度模糊，关键信息丢失。

注意集中低频区域集中的半径是多大，如果太模糊则适当放宽截止半径，否则限制截止半径。可以使用二分的形式来逐渐缩小得到适合的截止半径。

W10

1、 在利用各种高通锐化滤波器进行对图像进行锐化处理时,截止半径对锐化结果有怎样的影响? 截止半径的选择应该注意哪些事项?怎样根据当前的锐化结果判断截止半径选择的好不好?

如果锐化半径太小，高通性能不好，几乎所有频率的成分都能通过，则锐化效果欠佳。但是如果锐化半径取得太大，几乎所有频率的成分都不能通过，则信号都被阻止。图像在 ifft 之后就会几乎没有内容，这样的效果也是不能接受的。

注意集中低频区域集中的半径是多大，如果太模糊则适当扩大截止半径，否则限制截止半径。可以使用二分的形式来逐渐缩小得到适合的截止半径。

2、 对图像在频域中进行处理时,程序实现的主要步骤是什么，请简单说明。

输入图图像->预处理->傅里叶变换->频域高通锐化滤波处理->傅里叶反变换->得到增强图像

W11

1、频域中常用的图像复原方法有哪两种？请说一下它们的处理方法和步骤。

一、 逆滤波复原

用退化函数H去除退化图像的傅立叶变换（G(u,v)）来计算原始图像的傅立叶变换估计F’(u,v)

$$\hat{F}(u,v)=\frac{G(u,v)}{H(u,v)}=F(u,v)+\frac{N(u,v)}{H(u,v)}$$

对上述式子求傅立叶反变换就得到复原后的图像,主语 N(u,v)是随机函数，H(u,v)要限制避免为零值，将滤波频率限制在接近原点的地方，否则这一项会变得很大。

具体步骤：

（1）对于退化图像 g(x,y)求二维傅里叶变换 G(u,v);

（2）计算系统冲激响应 h(x,y)的二维傅里叶变换 H(u,v);

（3）用 n(u,v)产生随机噪声;

（4）用下式子计算 F’(u,v);

$$\hat{F}(u,v)=\frac{G(u,v)}{H(u,v)}=F(u,v)+\frac{n(u,v)}{H(u,v)}$$

（5）计算 F’(u,v)反变换即得复原后的图像结果。

二、 维纳滤波器

目标是寻找一个滤波器，使得复原后图像 f’(x,y)与原始图像 f(x,y)的均方误差最小，即:

$$E\{[\hat{f}(x,y)-f(x,y)]^2\}=\min$$

维纳滤波方法综合考虑了退化函数和噪声统计特征两个方面因素来达到这个目的。

在频域中用下式表示复原图像的频率：

$$\hat{F}(u,v)=\left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2+S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$

G(u,v)是退化图像的傅立叶变换，H(u,v)是退化函数

$|H(u,v)|^2=H(u,v)*H(u,v)$，其中 H*(u,v)是 H(u,v)的复共轭

$S_n(u,v)=|N(u,v)|^2$ 为噪声的功率谱，即信号在每个频率分量上的功率

$S_f(u,v)=|F(u,v)|^2$ 为为退化图像的频率谱

具体步骤：

（1)对退化图像 g(x,y)求二维傅里叶变换 G(u,v)；

(2)计算系统冲激响应 h(x,y)的二维傅里叶变换 H(u,v)；

(3)计算噪声图像 $S_n(u,v)$的功率谱和输入图像的功率谱 $S_f(u,v)$；

(4) 利用下式计算 F’(u,v)；

$$\hat{F}(u,v)=\left[\frac{1}{H(u,v)}\times\frac{|H(u,v)|^2}{|H(u,v)|^2+S_n(u,v)/S_f(u,v)}\right]G(u,v)$$

（5）计算 F’(u,v)反变换即得复原后的图像结果。

但是实际计算中 $S_n(u,v)/S_f(u,v)$很难计算，可以用一个比值 k 来代替两者之比，从而简化公式：

$$\hat{F}(u,v)=\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2+k}G(u,v)$$

W12

1、 什么是图像分割?

把图像空间按照一定的要求分成一些“有意义”的区域的技术叫图像分割。图像分割可以使人们将视线的焦点集中于感兴趣对象上。图像分割的本质是将图像中像素按照特性的不同进行分类。

2、 图像的分割的原则是什么?

a) 从简到难，逐级分割:从图像的整体出发，分离主要目标，然后再考虑细节；

b) 控制背景环境，降低场景分割的复杂程度；

c) 把焦点放在感兴趣的对象上，这样可以减少不相干图像成分的干扰

3 图像分割的出发点就是检测图像中非连续的图元,图像中图元包括哪些?

点、线、面

4、 简述图像中的点图元的检测的主要方法.

a) 利用拉普拉斯变换的各向同性的性质，检测大于一定阈值的像素点

具体掩码为: -1 -1 -1

-1 8 -1

-1 -1 -1

b) 设定阈值 T，计算结果 R 足够大时，即|R|>T，设该点为孤立点

5、 常用的边缘检测算子有哪些?

◆ 梯度边缘检测方法

用前向差分或者后向差分等计算梯度大小和方向

◆ Canny 边缘检测器

◆ Robert 算子边缘检测器

1	0
0	−1

0	1
−1	0

Roberts 算子

◆ Prewitt 算子边缘检测器

−1	−1	−1
0	0	0
1	1	1

−1	0	1
−1	0	1
−1	0	1

Prewitt 算子

◆ Sobel 算子边缘检测器

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Sobel 算子

◆ 拉普拉斯算子

0	1	0	0	-1	0
1	-4	1	-1	4	-1
0	1	0	0	-1	0
1	1	1	-1	-1	-1
1	-8	1	-1	8	1
1	1	1	-1	-1	-1

拉普拉斯算子

- LOG 算子边缘检测器

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

LOG 算子

6、阈值分割的主要思路是什么？

- 利用图像中要提取的目标区域与其背景在灰度特性上的差异，把图像看作具有不同灰度级的两类区域(目标区域和背景区域)的组合。
- 选取一个比较合理的阈值，把每个像素点的灰度值（强度）和阈值相比较，根据比较的结果把像素划分为两类：
  - 前景
  - 背景

7、全局阈值常见的确定方法有哪几种？

- 固定阈值方法（根据直方图直观给出）
- 全局阈值迭代方法
- OTSU 方法（最大类间方差法）

8、简述利用迭代方法确定全局阈值的主要步骤。

1. 选择一个初始化的阈值 TP (通常取所有像素灰度值的平均值)
2. 使用阈值 T 将图像所有像素分为两集合 G1 和 G2 : G1 包含灰度满足大于 TP 的像素，G2 包含灰度满足小于 TP 的像素。
3. 计算 G1 中所有像素灰度的均值 μ1，计算 G2 中所有像素的均值 μ2
4. 进一步计算当前阈值：$T_N = \frac{\mu_1 + \mu_2}{2}$

5. 重新步骤 2-4，直到在前后两次计算阈值的差值|TP-TN|小于一个预先指定的阈值误差 ET 为止。

9、简述利用 OTSU 方法（最大类间方差法）确定全局阈值的主要步骤。

1.对已知图像计算归一化的直方图，将其直方图成分记为 p_i，i=0,1,2…,L-1

2.利用 p_i 计算累计直方图 $P_i(k)$，i=1，2…，k=0,1,2,…,L-1;

3.计算累计的均值 m_i,i=1,2

4.计算整体的均值为:$m_G=P_1m_1+P_2m_2$

5.计算类间方差 σ_B^2

6.求取 $k^*=\text{argmax}_k\sigma_B^2$

7.利用阈值 k*对图像进行分割

W13

1、 在以下实例中,从灰度为 60 的种子像素出发,灰度差阈值设置为 20,分别写出 4 邻域和 8 邻域的生长结果.

10	10	10	10	10	10	10
10	10	10	69	70	10	10
59	10	60	64	59	56	60
10	59	10	60	70	10	62
10	60	59	65	67	10	65
10	10	10	10	10	10	10
10	10	10	10	10	10	10

4 领域生长

10	10	10	10	10	10	10
10	10	10	69	70	10	10
59	10	60	64	59	56	60
10	59	10	60	70	10	62
10	60	59	65	67	10	65
10	10	10	10	10	10	10
10	10	10	10	10	10	10

8 领域生长

10	10	10	10	10	10	10
10	10	10	69	70	10	10
59	10	60	64	59	56	60
10	59	10	60	70	10	62
10	60	59	65	67	10	65
10	10	10	10	10	10	10
10	10	10	10	10	10	10

2、 在第一题的例子中,还是从灰度为 60 的种子像素出发,如果将灰度差阈值设置为 5,再写出 4 邻域和 8 邻域的生长结果.

4 领域生长，$\Delta f<5$(严格小于 5)

10	10	10	10	10	10	10
10	10	10	69	70	10	10
59	10	60	64	59	56	60
10	59	10	60	70	10	62
10	60	59	65	67	10	65
10	10	10	10	10	10	10
10	10	10	10	10	10	10

8 领域生长

10	10	10	10	10	10	10
10	10	10	69	70	10	10
59	10	60	64	59	56	60
10	59	10	60	70	10	62
10	60	59	65	67	10	65
10	10	10	10	10	10	10
10	10	10	10	10	10	10

3、从第 1 题和第 2 题中，你能得出什么结论，灰度差阈值设置太大，结果会怎样？如果设置太小，生长结果会如何？

从第一题中发现，如果阈值设置太大，在区域生长过程中会不断地将区域中的灰度值上界和下届扩张，从而导致区域生长过大。

如果阈值设置得太小，在区域生长过程中将排除很多本该涉及的相似像素点，例如 5 为阈值时 4 领域生长的效果就非常不好，仅仅生长出了三个点。

同时值得注意的是，八邻域生长的相似像素包含的更准确，所以它可以生长得到比较精确的结果。

4、利用蛇模型，实现区域边界提取

- 根据能量方程，计算出表示曲线受力的欧拉方程

  图像的强度变化（与灰度值有关）—梯度—外能量

  曲线的形状（与控制点有关）----曲线的平滑性和连续性—内能量

  平滑性：曲线端点处一阶导数——弹性势能

  连续性：曲线端点处二阶导数——弯曲势能

  曲线上能量函数 E_{snake} 定义为：

$$E_{snake} = E_{external} + E_{\text{internal}}$$

外部能量可以从图像的强度(灰度)及边缘特征获得,它是用来吸引曲线到目标边缘的动力

$$E_{external} = -\left|\nabla I(x,y)\right|^2$$

其中∆I(x,y)表示图像在该控制点处的梯度

内部能量是建立在曲线的本身属性上的，其值为曲线的弹性势能和弯曲势能之和

$$E_{elastic} = \int_0^1 \frac{1}{2}\alpha(s)\left|v'(s)\right|^2 ds$$

$$E_{blending} = \int_0^1 \frac{1}{2}\beta(s)\left|v''(s)\right|^2 ds$$

- 按照曲线各点的受力来对曲线进行变形，直至曲线各点的受力为 0。
- 能量方程达到最小值时，曲线收敛到目标物体边缘。

W14

1.常见的数学形态学基本运算有哪些?用集合论的方法简单解释说明膨胀、腐蚀、开运算和闭运算的含义。

- 膨胀

  膨胀（Dilation）是将与某物体接触的所有背景点合并到该物体中的过程。其结果是使物体的面积增大了。如果物体是圆的，它的直径在每次膨胀后增大两个像素。膨胀主要用于填补分割后物体中的空洞。

$$A \oplus B = \{a + b \mid a \in A，\ b \in B\}$$

- 腐蚀

  图像的腐蚀（Erosion ）是消除物体的所有边界点的过程，其结果使剩下的物体沿其周边比原物体小一个像素的面积。

  集合 A（输入图像）被集合 B（结构元素）腐蚀，可以定义为：

$$A \Theta B = \{a \mid (a + b) \in A，\ a \in A, b \in B\}$$

- 开操作

  开运算就是先腐蚀后膨胀；定义为

$$A \circ B = (A \Theta B) \oplus B$$

- 闭操作

  闭运算是先膨胀后腐蚀的过程。定义为：

$$A \bullet B = (A \oplus B) \Theta B$$

2.利用数学形态学方法对图像处理时，能够实现哪些功能?请简述并给出在实际应用中的实例。

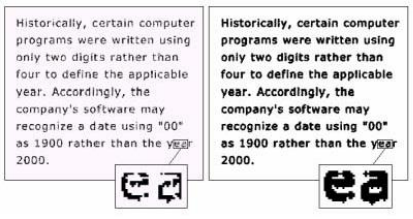
- 膨胀

  膨胀的主要应用于:

  （1） 可以修复断裂的边缘

  （2） 可以修复边缘；

  （3） 桥接裂缝文字。

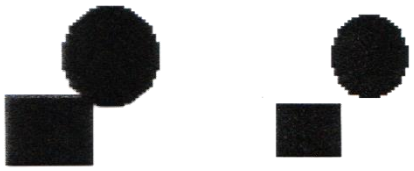


膨胀中的结构元

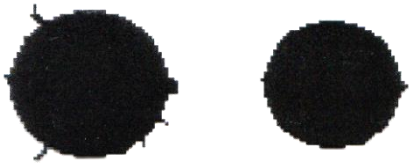
- **腐蚀**

  腐蚀操作的用途是：

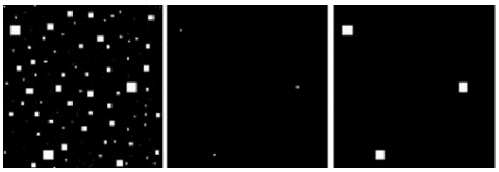
（1） 能将连接的物体分离开；



（2） 能够去掉物体周围的毛刺,使得边缘能够平滑；



（3） 消除图像细节，起到滤波的作用。



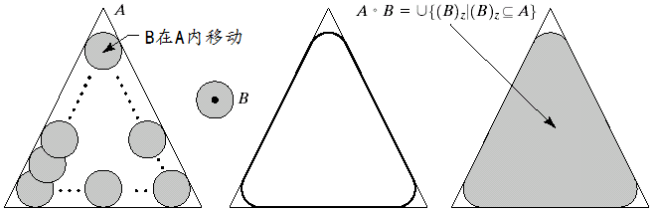
- 开操作

开操作的功能是：

（1）用来消除小物体，在纤细点处分离物体；

（2）平滑较大物体，的边界的同时并不明显改变其面积

实例：



- 闭操作

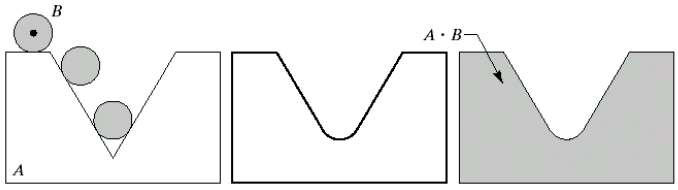
闭操作同样能使图像的轮廓变得光滑，但与开操作相反。

（1）能消除狭窄的间断和长细的鸿沟；

（2) 消除小的空洞，并填补轮廓线中的裂痕;

（3）平滑其边界的同时并不明显改变其面积

实例



- 边缘提取

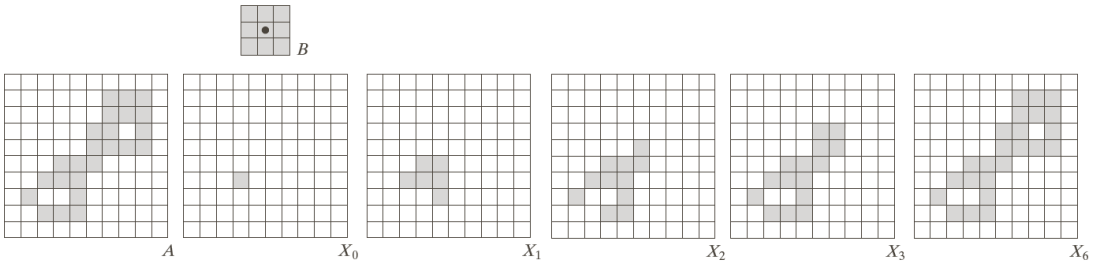
用形态学方法可以对图像边缘进行提取，实例为：

- 孔的填充

  利用数学形态学方法可以填充孔，实例为:

- 提取连通成分

  利用数学形态学方法可以提取连通成分，实例为:

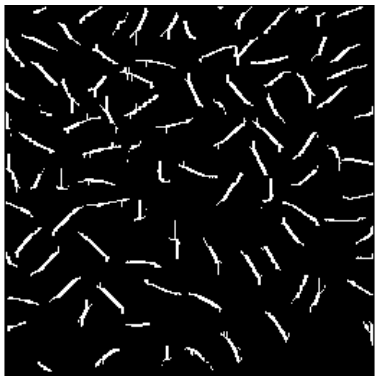


- 骨架提取

细化是一种图像处理运算，可以把二值图像区域缩成线条，以逼近区域的中心线，也称之为骨架或核线。

细化的目的是减少图像成份，直到只留下区域的最基本信息，以便进一步分析和识别.

虽然细化可以用在包含任何区域形状的二值图像，但它主要对细长形(而不是凸圆形或水滴状)区域有效．细化一般用于文本分析预处理阶段，骨架提取技术可以应用在很多领域。实例为:

3.分别说明利用数学形态学方法对图像进行边缘提取、孔洞填充、提取连通成分和骨架提取的主要思路和步骤。

- 边缘提取

  用形态学方法可以对图像边缘进行提取

  具体的提取步骤：

  （1）源图像为 A，结构元为 B；

  （2）A 在 B 的结构元作用下进行腐蚀，得到 C 的结果;

  （3）用 A 源图像减去 C，便得到边缘 D 的结果。

  上面的这个过程可以表示为：

$$\beta(A) = A - (A \Theta B)$$

- 孔的填充

  利用数学形态学方法可以填充孔，具体做法是：

  （1）求带孔图像 A 的补集记为 A^c；

  （2）确定结构元 B；

  （3）在带孔边缘内部选择一个点，并将该点作为初始化的种子 X

  （4）利用下面的形态学运算得到 X_k（k=1,2…）；

$$X_k = (X_{k-1} \oplus B) \bigcap A^c$$

  （5）判断 $X_k=X_{k-1}$ 是否成立，如果成立，转下一步；否则，转步骤（4）

  （6）利用(5)中得到的 X_k 和 A 求并集，得到最后的目标。

- 提取连通成分

  利用数学形态学方法可以提取连通成分，具体做法是：

  （a) 确定结构元 B；

  (b) 在源图像 A 内部选择一个未处理的点，并将该点作为初始化的 X

  (c) 利用下面的形态学运算得到 Xk（k=1,2…）；

$$X_k = (X_{k-1} \oplus B) \bigcap A$$

  (d) 判断 $X_k=X_{k-1}$ 是否成立，如果成立，转下一步；否则，转步骤（c）

  (e) 在源图像 A 中是否存在未处理的点，如果有，转步骤（b）；否则，下一步。

  (f) 算法结束。

- 骨架提取

  细化是一种图像处理运算，可以把二值图像区域缩成线条，以逼近区域的中心线，也称之为骨架或核线。

细化的目的是减少图像成份，直到只留下区域的最基本信息，以便进一步分析和识别.

虽然细化可以用在包含任何区域形状的二值图像，但它主要对细长形(而不是凸圆形或水滴状)区域有效．细化一般用于文本分析预处理阶段，骨架提取技术可以应用在很多领域。

为了描述骨架提取算法，我们定义 B 对 A 相继腐蚀 k 次的形式为：

$$A\Theta kB = (\cdots(A\Theta B)\Theta B)\ldots)\Theta B)$$

某个集合 A 骨架记为 S(A)，其可用腐蚀和开运算来表达

$$S(A) = \bigcup_{k=1}^{K} S_k(A)$$

其中：

$$S_k(A) = (A\Theta kB) - (A\Theta kB) \circ B$$

$$K = \max\{k \,|\, (A\Theta kB) \neq \phi\}$$

即 K 是集合 A 被腐蚀为空集前的最大迭代次数

总结为:

$$S(A) = \bigcup_{k=1}^{K} (A\Theta kB) - (A\Theta kB) \circ B$$

W15

1、 编程实现彩色图像的处理功能(取反、灰度化、浮雕效果、马赛克效果)

取反 C++实现:

```
void CW15View::OnStartReverse()
{
    // TODO: 在此添加命令处理程序代码
    newbmp.CreateCDib(sizeimage, 24);

    for (int x = 0; x < sizeimage.cx; x++)
    {
        for (int y = 0; y < sizeimage.cy; y++)
        {
            RGBQUAD color;
            color = mybmp.GetPixel(x, y);
            color.rgbBlue = 255 - color.rgbBlue;
            color.rgbGreen = 255 - color.rgbGreen;
            color.rgbRed = 255 - color.rgbRed;
            newbmp.WritePixel(x, y, color);
        }
    }
    Invalidate();
}
```

灰度化 C++实现

```
void CW15View::OnStartGray()
{
    // TODO: 在此添加命令处理程序代码
    newbmp.CreateCDib(sizeimage, 24);
    // 对图象的象素值进行变换
    // 每列
    for (int x = 0; x < sizeimage.cx; x++)
    {
        // 每行
        for (int y = 0; y < sizeimage.cy; y++)
        {
            RGBQUAD color;
            color = mybmp.GetPixel(x, y);
            //RGB 图像转灰度图像 Gray = R*0.299 + G*0.587 + B*0.114
            double gray = color.rgbRed*0.299 + color.rgbGreen*0.587 + color.rgbBlue*0.114;
            color.rgbBlue = (unsigned char)gray;
            color.rgbGreen = (unsigned char)gray;
            color.rgbRed = (unsigned char)gray;
            newbmp.WritePixel(x, y, color);
        }
    }
    Invalidate();

}
```

浮雕效果 C++实现

```
void CW15View::OnStartEmbossment()
{
    // TODO: 在此添加命令处理程序代码
    newbmp.CreateCDib(sizeimage, 24);
    for (int x = 1; x < sizeimage.cx; x++)
    {
        for (int y = 1; y < sizeimage.cy; y++)
        {
            RGBQUAD color;
            color = mybmp.GetPixel(x, y);

            RGBQUAD color1;
            color1 = mybmp.GetPixel(x - 1, y);

            //G(i,j)= f(i,j)- f(i-1,j)+常量
            color.rgbBlue = color.rgbBlue - color1.rgbBlue + 128;
            color.rgbGreen = color.rgbGreen - color1.rgbGreen + 128;
            color.rgbRed = color.rgbRed - color1.rgbRed + 128;

            if (color.rgbBlue > 255) color.rgbBlue = 255;
            if (color.rgbBlue < 0) color.rgbBlue = 0;
            if (color.rgbGreen > 255) color.rgbGreen = 255;
            if (color.rgbGreen < 0) color.rgbGreen = 0;
            if (color.rgbRed > 255) color.rgbRed = 255;
            if (color.rgbRed < 0) color.rgbRed = 0;
            newbmp.WritePixel(x, y, color);
        }
    }
    Invalidate();
}
```

马赛克 C++实现：

```
void CW15View::OnStartMosaic()
{
    // TODO: 在此添加命令处理程序代码
    newbmp.CreateCDib(sizeimage, 24);
    for (int x = 1; x < sizeimage.cx - 1; x += 3)
    {
        for (int y = 1; y < sizeimage.cy - 1; y += 3)
        {
            RGBQUAD color;
            color = mybmp.GetPixel(x, y);

            int r = 0, g = 0, b = 0, num = 0;
            for (int m1 = -1; m1 <= 1; m1++)
                for (int m2 = -1; m2 <= 1; m2++)
                {
                    if (x + m1 >= sizeimage.cx || x + m1 < 0 || y + m2 >= sizeimage.cy || y + m2< 0) continue;
                    num++;
                    RGBQUAD color1;
                    color1 = mybmp.GetPixel(x + m1, y + m2); r += color1.rgbRed;
                    g += color1.rgbGreen; b += color1.rgbBlue;
                }

            color.rgbRed = (unsigned char)(r*1.0 / num); color.rgbGreen = (unsigned char)(g*1.0 / num); color.rgbBlue = (unsigned char)(b*1.0 / num);

            for (int m1 = -1; m1 <= 1; m1++)
                for (int m2 = -1; m2 <= 1; m2++)
                {
                    newbmp.WritePixel(x + m1, y + m2, color);
                }
        }
    }
    Invalidate();
}
```

2、 简述马赛克功能的具体实现方法。

将图像从形式上划分为很多小块，在每块内的各个像素都取到相同的红、绿、蓝颜色值，从而对某些细节进行模糊化处理，使图像粗糙化。

例如，如果对于 3×3 的矩阵区域进行处理时，并且原来图像表示为 f(x,y)，处理后的图像用 g(x,y)表示，那么处理方法为：

$$g(x,y)=\frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1}f(x+i,y+j)$$

$$g(x+m,y+n)=g(x,y)\ \text{-}1\le m\le 1,\text{-}1\le n\le 1$$

3、 简述浮雕功能的具体实现方法。

将图像的变化部分突出出来，相同颜色部分则被淡化，使图像出现纵深感.

方法:

$$g(x,y)=f(i,j)-f(i-1,j)+常量$$

其中

$$R(x, y) = R(i, j) - R(i-1, j) + 128$$

$$G(x, y) = G(i, j) - G(i-1, j) + 128$$

$$B(x, y) = B(i, j) - B(i-1, j) + 128$$