

# 华中科技大学

Huazhong University of Science and Technology



## 生物医学图像处理课程

# 编程练习题

学院： 计算机学院      专业： 计算机体系结构

姓名： 余 乔      班级： 博 1901 班

学号： D201980953      时间： 2019 年 10 月

## 1. 图像旋转

### 题目描述

设一幅大小为  $M \times N$  的灰度图像  $I$  中，现要将其逆时针旋转  $A$  度，得到图像  $J$ ，请写出  $J$  的生成算法（要求使用近邻插值）。

### 算法详解

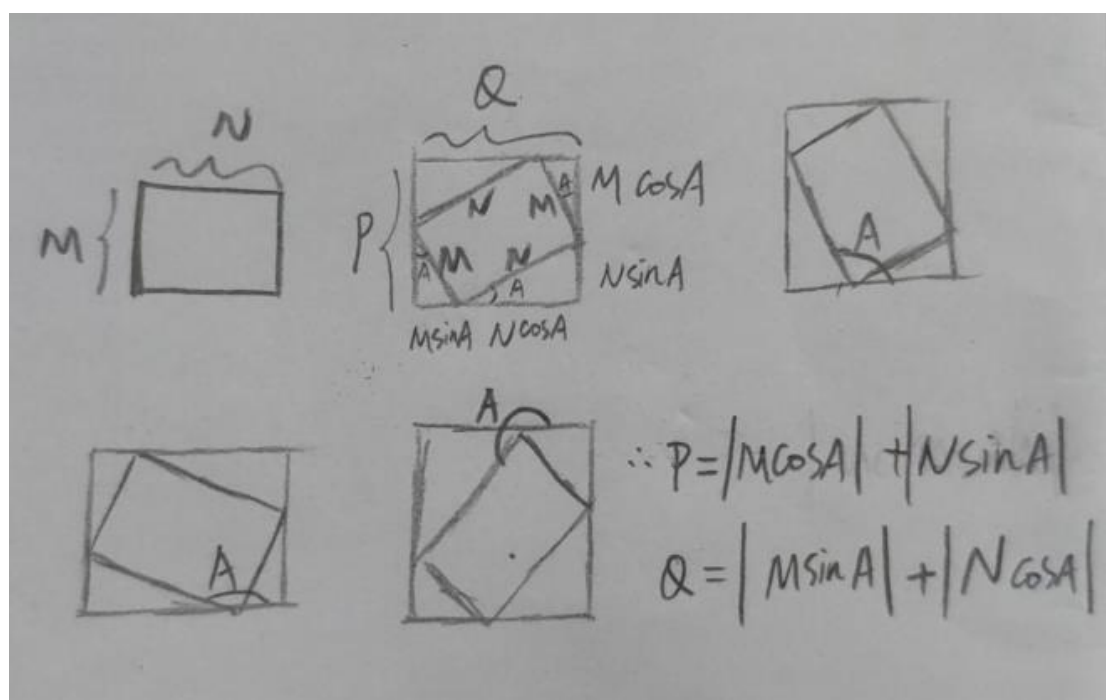
#### 基本思路

1. 首先求出旋转后图像  $J$  的行数  $P$  和列数  $Q$ 。
2. 对  $J$  中的每个像素的坐标  $(x_1, y_1)$ ，求出与其对应的原图  $I$  中像素的坐标  $(x, y)$
3. 用最邻近插值，求出最接近于坐标  $(x, y)$  的整数坐标  $(x_0, y_0)$
4. 将  $J(x_1, y_1)$  的灰度值赋为  $I(x_0, y_0)$  的灰度值。

#### 细节与推导

求  $J$  的行列数  $P, Q$

其中，行列数  $PQ$  的求法如下：



求  $J(x_1, y_1)$  对应点  $I(x, y)$  的坐标

$J(x_1, y_1)$  对应点  $I(x, y)$  的坐标求法如下：

1. 将  $J$  的中心点  $J(P/2, Q/2)$  移动到  $(0, 0)$  处
2. 将整个图像顺时针旋转  $A$  度
3. 最后将图像的  $(0, 0)$  平移到  $(M/2, N/2)$ 。

此时，原来在  $(x_1, y_1)$  的像素，就被变换到了  $(x, y)$  处。

下面画图进行推导：

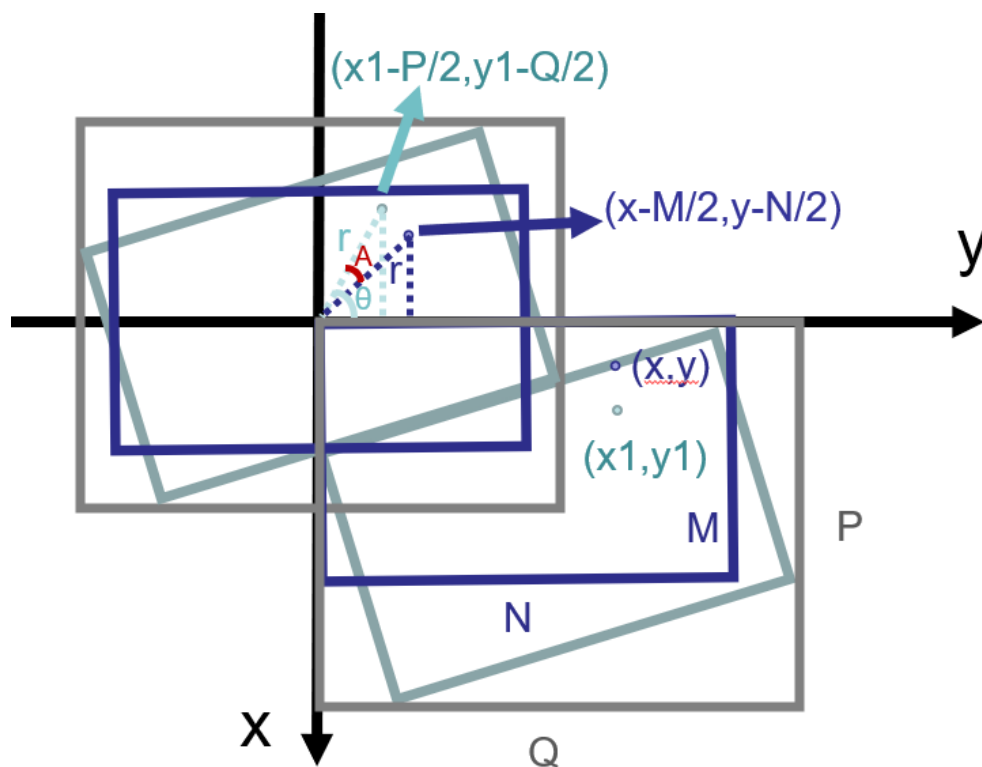
$$x - \frac{M}{2} = r \cos(\theta - A) = r(\cos\theta \cos A + \sin\theta \sin A) = \left(x_1 - \frac{P}{2}\right) \cos A + \left(y_1 - \frac{Q}{2}\right) \sin A$$

$$y - \frac{N}{2} = r \sin(\theta - A) = r(\sin\theta \cos A - \cos\theta \sin A) = \left(y_1 - \frac{Q}{2}\right) \cos A - \left(x_1 - \frac{P}{2}\right) \sin A$$

因此：

$$x = \left(x_1 - \frac{P}{2}\right) \cos A + \left(y_1 - \frac{Q}{2}\right) \sin A + \frac{M}{2}$$

$$y = \left(y_1 - \frac{Q}{2}\right) \cos A - \left(x_1 - \frac{P}{2}\right) \sin A + \frac{N}{2}$$



## 最邻近插值&双线性插值

最邻近插值中，直接  $x_0, y_0$  分别直接取  $x, y$  四舍五入得到的整数。

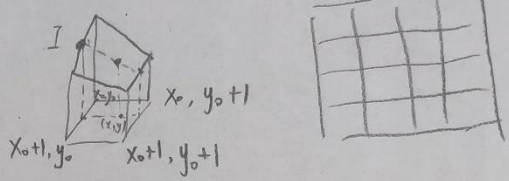
如果使用双线性插值，则公式如下：

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

$$\begin{aligned} \text{对于这道题，即为 } J(x_1, y_1) = & \text{uint8} ( I(x_0, y_0) * (1-x+x_0) * (1-y+y_0) \dots \\ & + I(x_0+1, y_0) * (x-x_0) * (1-y+y_0) \dots \\ & + I(x_0, y_0+1) * (1-x+x_0) * (y-y_0) \dots \\ & + I(x_0+1, y_0+1) * (x-x_0) * (y-y_0)); \end{aligned}$$

推导如下：

双线性公式推导 =



$$\begin{aligned} I(x, y_0) &= \left( \frac{I(x_0+1, y_0) - I(x_0, y_0)}{x_0+1 - x_0} \right) \cdot (x - x_0) + I(x_0, y_0) = (x - x_0) I(x_0+1, y_0) + (1 - x + x_0) I(x_0, y_0) \\ I(x, y_0+1) &= \left( \frac{I(x_0+1, y_0+1) - I(x_0, y_0+1)}{x_0+1 - x_0} \right) \cdot (x - x_0) + I(x_0, y_0+1) = (x - x_0) I(x_0+1, y_0+1) + (1 - x + x_0) I(x_0, y_0+1) \\ \therefore I(x, y) &= I(x, y_0) + \left( \frac{I(x, y_0+1) - I(x, y_0)}{y_0+1 - y_0} \right) \cdot (y - y_0) \\ &= I(x, y_0) (1 - y + y_0) + I(x, y_0+1) (y - y_0) \\ &= \left[ (x - x_0) I(x_0+1, y_0) + (1 - x + x_0) I(x_0, y_0) \right] (1 - y + y_0) \\ &\quad + \left[ (x - x_0) I(x_0+1, y_0+1) + (1 - x + x_0) I(x_0, y_0+1) \right] (y - y_0) \\ &= I(x_0, y_0) (1 - x + x_0) (1 - y + y_0) + I(x_0+1, y_0) (x - x_0) (1 - y + y_0) \\ &\quad + I(x_0, y_0+1) (1 - x + x_0) (y - y_0) + I(x_0+1, y_0+1) (x - x_0) (y - y_0) \end{aligned}$$

## 完整算法描述

```
P = |M cosA| + |N sinA|
Q = |M sinA| + |N cosA|
for x1 = 1:P
    for y1 = 1:Q
```

```

x = (x1-P*0.5) * cos(A) + (y1-Q*0.5)* sin(A) + M*0.5;
y = -(x1-P*0.5) * sin(A) + (y1-Q*0.5)* cos(A) + N*0.5;
x0 = 最接近x的整数
y0 = 最接近y的整数
如果I(x0,y0)的坐标x0不在0~M范围内或y0不在0~N范围内, 则对应的
J(x1,y1) = 0
否则J(x1,y1) = I(x0,y0)

```

## 代码实现

```

function [J] = rotateYQ(I,A)

%rotate 设一幅大小为 M×N 的灰度图像 I 中, 现要将其逆时针旋转 A 度 (角度制),
得到图像 J, 请写出 J 的生成算法 (要求使用近邻插值)。

% 设 I(x,y)经旋转后对应的点为 J(x1,y1).(x0,y0)是最接近(x,y)的整数对

% 求 J 的行列数 P/Q

[M, N] = size(I);

P = round( abs(M * cosd(A)) + abs(N * sind(A)));
Q = round( abs(M * sind(A)) +abs( N * cosd(A)));
J = zeros(P,Q);

for x1 = 1:P
    for y1 = 1:Q
        % 求 J(x1,y1)在原图 I 中对应的像素坐标(x,y)

        x = (x1-P*0.5) * cosd(A) + (y1-Q*0.5)* sind(A) + M*0.5;
        y = -(x1-P*0.5) * sind(A) + (y1-Q*0.5)* cosd(A) + N*0.5;

        % 最邻近插值

        x0 = round(x);
        y0 = round(y);

        % 给 J 赋值

        if x0 < 1 || y0 < 1 || x0 > M || y0 > N

            J(x1,y1) = 0;

```

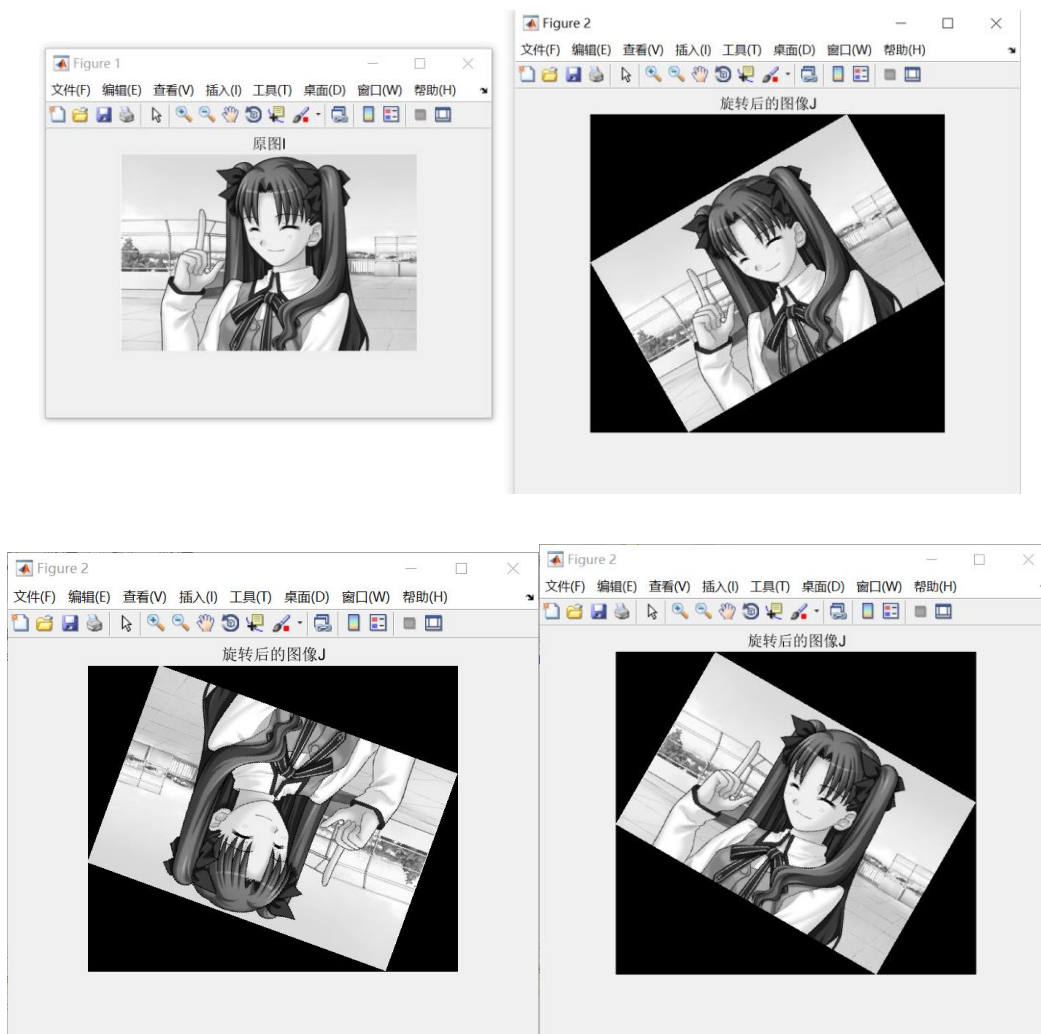
```

else
    J(x1,y1) = I(x0,y0);
end
end
end
end
% 显示结果
imshow(uint8(I));title('原图 I');
figure(2),imshow(uint8(J));title('旋转后的图像 J');
end

```

## 结果展示

下面四幅图从左到右从上到下，分别展示了原图，旋转 30°、160°和-30°的结果。



## 2. 直方图规定化

### 题目描述

设一幅大小为  $M \times N$  的灰度图像  $I$  中，灰度为  $g$  的像素数为  $h(g)$ 。另给定一个直方图  $t(g)$ 。请写出对图像  $I$  进行变换的方法，使得变换后的新图像的直方图与  $t$  相同（近似相等）。

### 算法详解

#### 基本思路

将原图中灰度  $g$  映射到  $g'$ ，使得原图  $I$  中，灰度小于等于  $g$  的像素占  $I$  中总像素数之比尽可能接近模板图中，灰度小于等于  $g'$  的像素个数占模板图总像素数之比。

#### 完整算法描述

$h$  是含一个 256 个元素的数组(假设下标从 0 开始)，用于表示图像  $I$  的直方图。

将  $h$  中所有元素置为零。

遍历每一个点，如果其灰度为  $i$ ，则  $h(i)++$ 。

For( $k$  从 0 到 255)

求累积分布  $rh(k) = \sum_{i=0}^k h(i)$  ,

求累积分布  $rt(k) = \sum_{i=0}^k t(i)$  ,

For( $k$  从 0 到 255)

求累计频率  $frequencySrc(k) = rh(k) / rh(255)$  ;

求累计频率  $frequencyDes(k) = rt(k) / rt(255)$ ;

For( $i$  从 0 到 255)

Map( $i$ ) = 与  $frequencySrc(i)$  最接近的  $frequencyDes(k)$  的  $k$  值

遍历每个点，若其灰度值为  $i$ ，则将其值置为 Map( $i$ )。

## 代码实现

```
function [ J ] = HistogramSpecificate( I,t )

%HistogramSpecificate 设一幅大小为 M×N 的灰度图像 I 中，灰度为 g 的像素数为 h(g)，另给定一个直方图 t(g)。

%请写出对图像 I 进行变换的方法，使得变换后的新图像的直方图与 t 相同（近似相等）。

[M, N ] = size(I);
J = zeros(M, N);
h = zeros(1,256);

%求直方图 h
for x = 1:M
    for y = 1:N
        h(I(x,y)+ 1) = h(I(x,y)+ 1)+1;
    end
end

%求累积分布 rh 和 rt
rh = zeros(1,256);
rt = zeros(1,256);
for i =1:256
    for j = 1:i
        rh(i) = rh(i) + h(j);
        rt(i) = rt(i) + t(j);
    end
end

%求各灰度级的像素出现的频率 frequencySrc 和 frequencyDes
frequencySrc = zeros(1,256);
frequencyDes = zeros(1,256);
for i =1:256
```



```

    frequencySrc(i) = rh(i) / rh(256);
    frequencyDes(i) = rt(i) / rt(256);
end
%找出与 frequencySrc(i)最相近的 frequencyDes(index)的 index 值
%distance 是 frequencySrc(i) 与 frequencyDes(index) 的差
%map 用于记录所有 i 对应的 index 值
map = zeros(1,256);
for i = 1:256
    [distance,index] = min(abs(frequencySrc(i)-frequencyDes));
    map(i) = index;
end

%按照映射 map 进行直方图规定化
for x = 1:M
    for y = 1:N
        J(x,y) = map(I(x,y)+1) ;
    end
end

% 显示结果
subplot(211)
imshow(uint8(I));title('原图 I');
subplot(212)
imshow(uint8(J));title('规定化化后的图像 J');
end

```

## 结果展示

以下图所示的老虎图片作为模板图，将动漫图片进行规定化。可以看到规定化后图片明显变暗了。



### 3. 频域滤波

#### 题目描述

对图像进行频域低、高通滤波。使用理想、高斯、巴特沃斯滤波器。

#### 算法详解

##### 基本思路

用傅里叶变换将图片从空域变换到频域。

将低频部分移动到图像中间。

对每一频率的分量，将其乘以滤波器函数。（低通滤波则滤波器函数单调递减，高通滤波则滤波器函数单调递增）

将之前的平移操作还原回去。

将变换后的频域图像通过傅里叶反变换变回到空域。

## 离散二维傅里叶变换/反变换公式

其二维离散傅里叶变换 (DFT) 为

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

离散傅里叶反变换 (IDFT) 为

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

## 为什么要平移

在 matlab 中，经过 fft 变换后，数据的频率范围是从[0,fs]排列的。

而一般，我们在画图或者讨论的时候，是从[-fs/2,fs/2]的范围进行分析。

因此，需要将经过 fft 变换后的图像的[fs/2,fs]部分移动到[-fs/2,0]这个范围内。

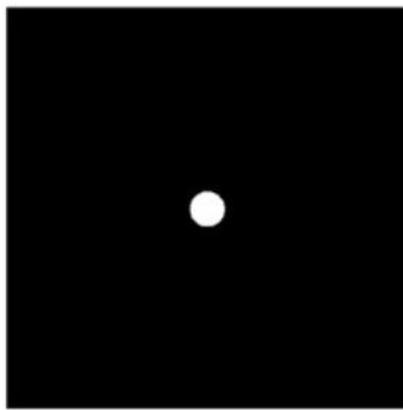
而 fftshift 就是完成这个功能。

通常，如果想得到所见的中间是 0 频的图像，经过 fft 变换后，都要再经过 fftshift。

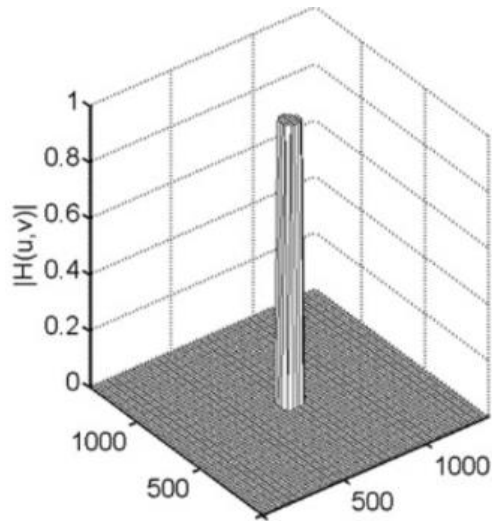
## 理想、高斯、巴特沃斯滤波器

$$H(u, v) = \begin{cases} 1, D(u, v) \leq D_0 \\ 0, D(u, v) > D_0 \end{cases}$$

理想低通滤波器如下：



c). Ideal Lowpass filter(D=60)



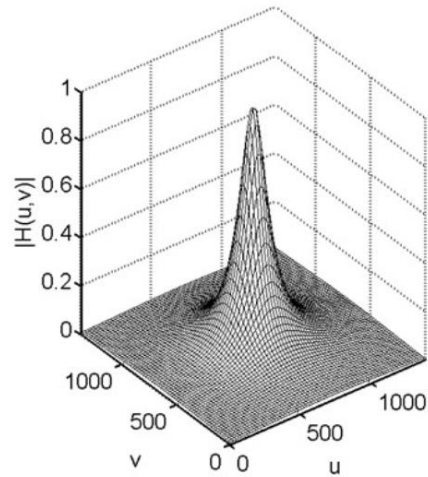
巴特沃斯低通滤波器如下：

$$H(u,v) = \frac{1}{1 + (D(u,v)/D_0)^{2n}}$$

其中 n 是巴特沃斯低通滤波器的次数。

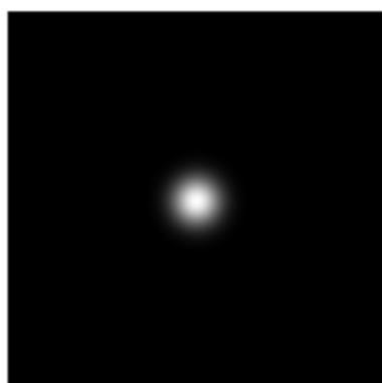


c) Butterworth Lowpass ( $D_0=100, n=1$ )

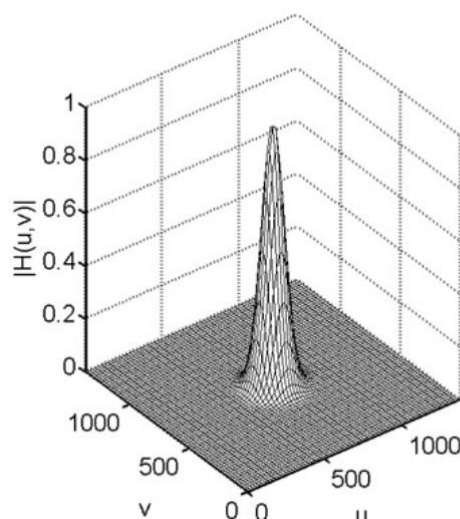


高斯低通滤波器如下：

$$H(u,v) = e^{\frac{-D^2(u,v)}{2D_0^2}}$$



c)Gaussian Lowpass ( $D_0=60$ )



相应的高通滤波器，只需用 1 减去低通滤波器即可。

## 完整算法描述

F=快速傅里叶变换(I)

将 F 进行居中变换（低频部分移动到图像中心）

对于 F 中每一点

$d$  = 该点到图像中心的距离， $d_0$  是滤波的阈值

如果是理想低通滤波，则 if  $d > d_0$ ,  $h = 0$ ; 否则  $h = 1$ ;

如果是高斯低通滤波器，则  $h = 1 * \exp(-d^2 / (2 * d_0^2))$ ;

如果是巴特沃斯低通滤波器，则  $h = 1 / (1 + (d/d_0)^{2*n})$ ;

如果是相应的高通滤波器，则  $h = 1 -$  相应的低通滤波器对应的  $h$

该点对应的 F 值 = 该点对应的 F 值 \*  $h$

将 F 进行居中变换的反变换（还原刚才的平移操作）

对 F 进行快速傅里叶反变换，即得处理完成的图片。

## 代码实现

```
function [J] = frequencyDomainFiltering( I, d0, way, n)
```

```
%frequencyDomainFiltering 频域滤波
```

```
% I 为原图，d 为低通滤波或高通滤波的阈值
```

```
% way 1 理想低通滤波，高斯低通滤波，巴特沃斯低通频域滤波，
```

```
% n: 如果使用 butterworth 滤波，取的参数 n（如果不采用 butterworth 滤波，则 n 不被使用

% 变换到频域

I=double(I);

F=fftshift(fft2(I)); %fft2 傅里叶变换 % fftshift Shift zero-frequency component to center of spectrum.

[M,N]=size(F);

centerM = fix(M/2);

centerN = fix(N/2);

% 进行低通滤波操作

for x=1:M

    for y=1:N

        d=sqrt((x-centerM)^2+(y-centerN)^2);

        switch way

            case 1 % 理想低通滤波 (ILPF)

                if d > d0

                    h = 0;

                else

                    h = 1;

                end

            case 2 %高斯低通滤波 (GLPF)

                h=1*exp(-d^2/(2*d0^2));

            case 3 %Butterworth 低通滤波器(BLPF)

                h = 1/(1+(d/d0)^(2*n));

            case 4 % 理想高通滤波 (IHPF)

                if d > d0
```

```

        h = 1;
    else
        h = 0;
    end

    case 5 %高斯高通滤波 (GHPF)
        h= 1 - 1*exp(-d^2/(2*d0^2));
    case 6 %Butterworth 高通滤波器(BLPF)
        h = 1 - 1/(1+(d/d0)^(2*n)) ;
    end

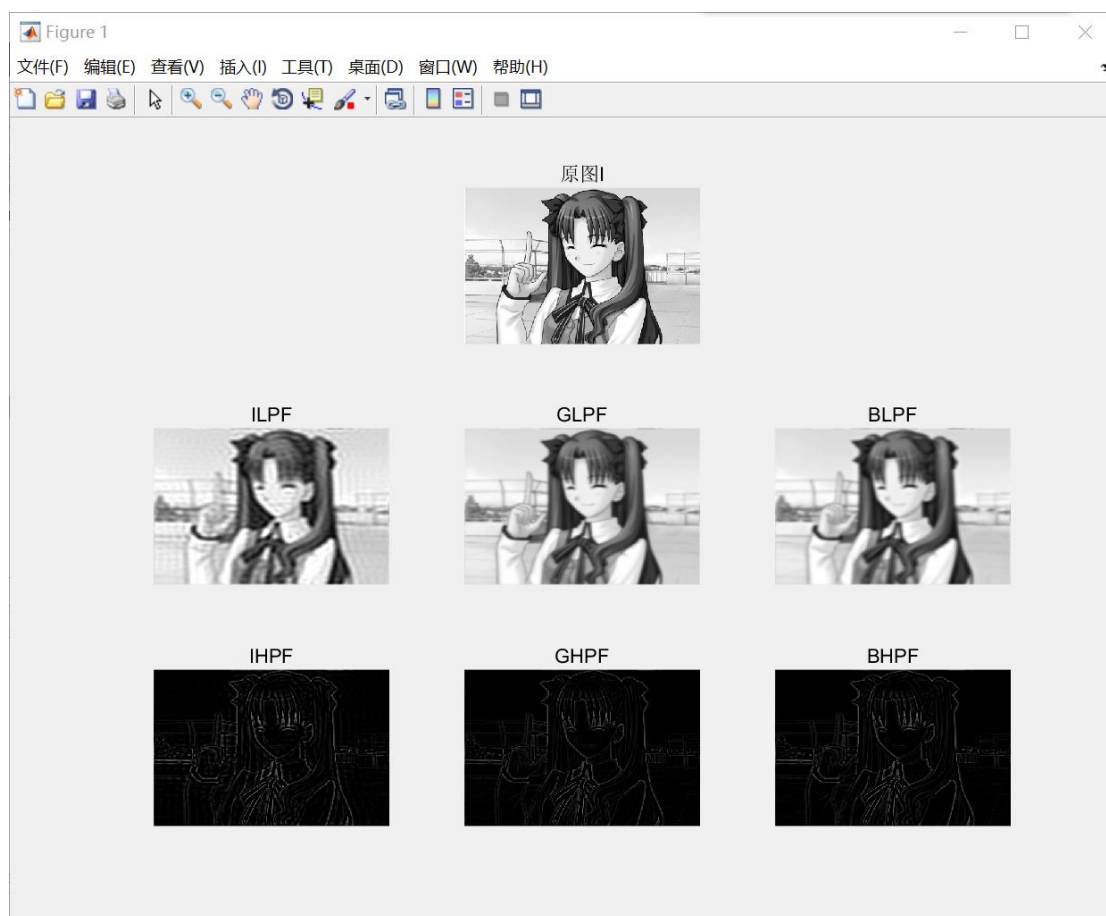
    F(x,y)=h*F(x,y);
end

end

% 变回空域
F=ifftshift(F); % 平移回去
J=uint8(real(ifft2(F))); %傅里叶反变换
% 显示结果
end

```

## 结果展示



## 4. 连通区域标记

### 题目描述

设有一幅二值图像（元素取值为 0 或 1），请生成该图像的标记图像。（即第一个连通区域中的每一个白色像素的值都置为 1，第二个连通区域中的每一个白色像素的值都置为 2，依此类推。区域编号可不考虑顺序）

### 算法详解

#### 基本思路

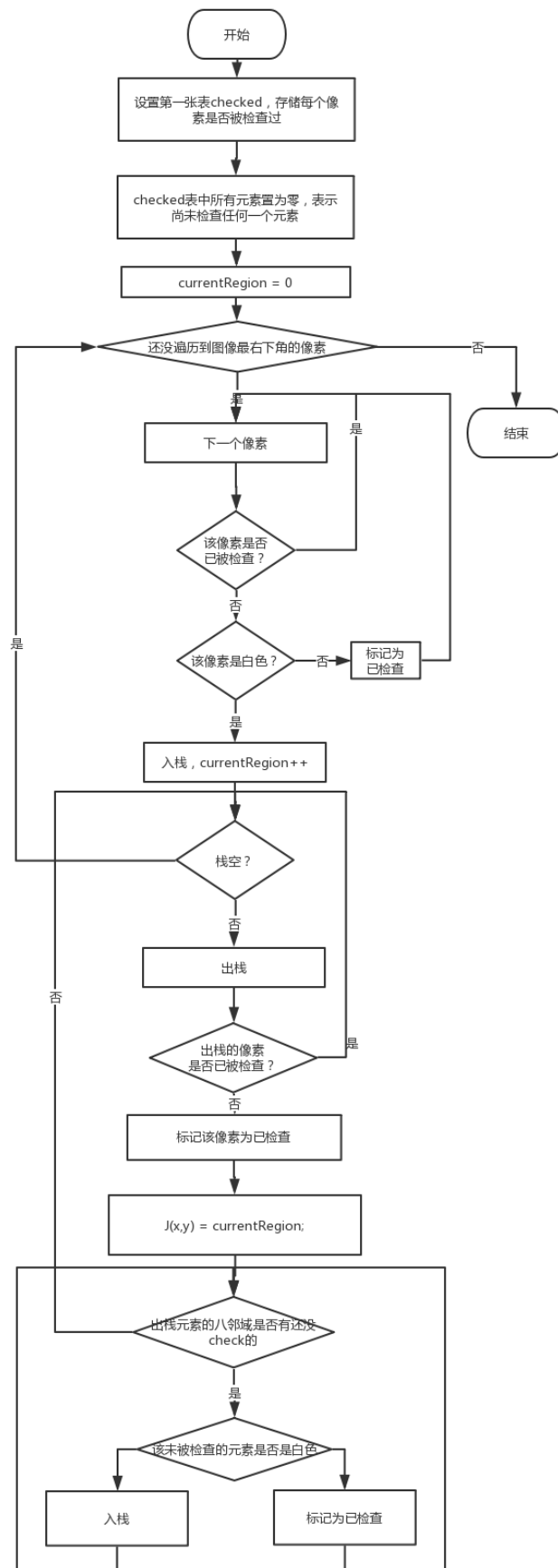
设置一个栈，来进行深度优先搜索。维护一张表，来统计每个像素是否已被检查过。

从第一个点开始，对于图像中的每个像素，判断其是否是白色，如果是，就继续考察其八邻域。如此进行深度优先的搜索，直到八邻域没有白色像素（栈空）。

栈空时说明，该区域的所有像素都已完成标记。栈空后，就考察下一个像素，直到所有像素都被考察完。



## 完整算法描述



## 代码实现

```
function [J] = regionMark(I)

%UNTITLED 设有一幅二值图像（元素取值为 0 或 1），请生成该图像的标记图像。

% （即第一个连通区域中的每一个白色像素的值都置为某种颜色，第二个连通区域中的
% 的每一个白色像素的值都置为另一种颜色，依此类推。区域编号可不考虑顺序）

%

[M, N] = size(I);

J = zeros(M, N, 3);

checked = zeros(M,N);

currentRegion = 0;

%填色

colors = [254 67 101;252 157 154;249 205 173;200 200 169;131 175
155;138,151,123;224,208,0;229,131,8;220,87,18;182,194,154;];

% 对每一个点，如果是白色 1，则考察其四邻域（上下左右的四个像素）中尚未被考察的点

% 用矩阵 stack 表示一个栈，每行表示一个像素，第一列是像素的 x 坐标，第二列是像素的 y 坐标

%初始化栈

stack = zeros(M*N,2);

stackSize = 0;

for x = 1:M

    for y = 1:N

        if checked(x,y) == 0 && I(x,y)==1 % 如果还没检查过这个点且是白色，就入栈

            %fprintf('顺序入栈: %d %d\n',x,y)

            stackSize = stackSize + 1;

            stack(stackSize,1) = x;

            stack(stackSize,2) = y;
```

```

        currentRegion = currentRegion + 1

        elseif checked(x,y) == 0 && I(x,y)==0 % 如果还没检查过这个点且是黑色，就
标记为已检查

            checked(x,y) = 1;

            continue;

        else %已经检查过了，就跳过

            continue;

        end

        while(stackSize ~= 0) % 若栈非空

            % 出栈

            currentX = stack(stackSize,1);

            currentY = stack(stackSize,2);

            fprintf('当前检查的像素: %d %d\n',currentX,currentY)

            stackSize = stackSize - 1;

            fprintf('因检查而出栈: %d %d\n',currentX,currentY)

            if checked(currentX,currentY) == 1 % 如果已经检查过这个点，就跳过

                continue;

            end

            checked(currentX,currentY) = 1;%将该点标记为已检查

            J(currentX,currentY,:) = colors(mod(currentRegion,10),:); %只设置了十种颜色，如果区域太多，就再从第一种颜色开始填色

            % 判断八邻域像素是否已经检查过

            for neighborX = currentX-1:1:currentX+1

                for neighborY = currentY-1:1:currentY+1

                    tempX = neighborX;

                    tempY = neighborY;

                    %边缘处理

                    if tempX < 1

```



```
% 显示结果

subplot(121)

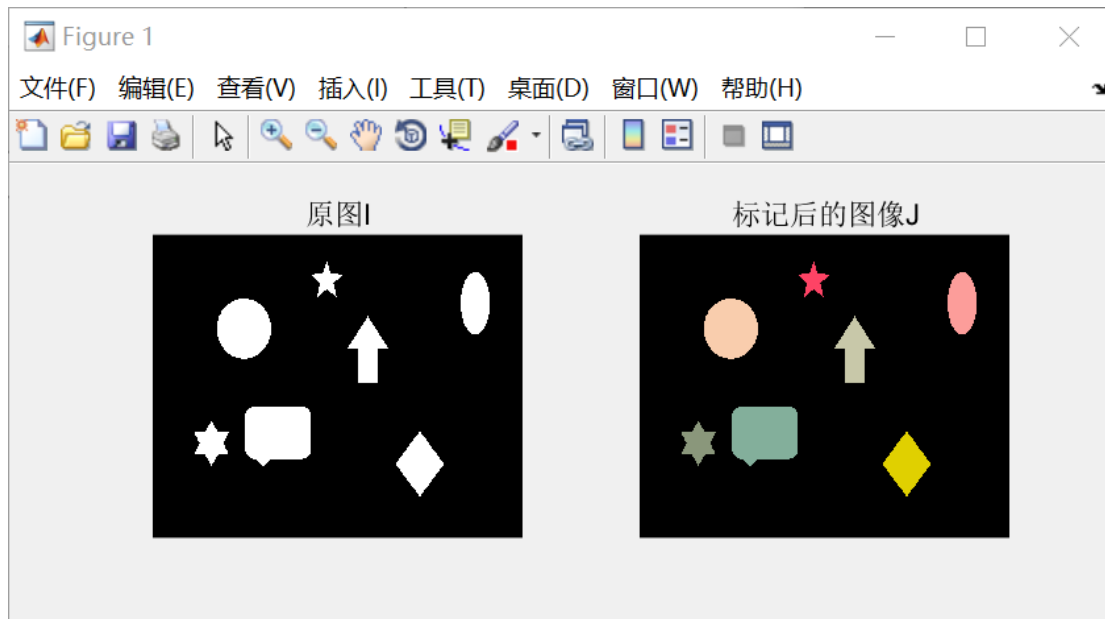
imshow(logical(I));title('原图 I');

subplot(122)

imshow(uint8(J));title('标记后的图像 J');

end
```

## 结果展示



## 5. 边界追踪

### 题目描述

设一幅二值图像中，只有一个白色区域，试给出求该区域外围轮廓线的方法（要求按顺时针的顺序给出各点的坐标，即行/列号）。

### 算法详解

#### 基本思路

1. 遍历每个像素，如果该像素本身是黑色，且其八邻域中有白色像素，就将该像素存入边界像素表。
2. 统计白色像素横纵坐标的平均值，找到区域的重心。
3. 遍历边界像素表中的每个像素，计算其到重心的角度 $\theta$ ( $0\sim 360^\circ$ )
4. 根据 $\theta$ 对边界像素进行排序
5. 排序结果即为按顺或逆时针方向排序的边界点坐标。

### 完整算法描述

得到图像的行数  $M$  和列数  $N$

```
for x = 1:M
```

```
    for y = 1:N
```

```
        如果(x,y)是白色点,
```

```
            xSum = xSum + x;
```

```
            ySum = ySum + y;
```

```
            whiteSum = whiteSum+1;
```

```
        如果(x,y)是黑色点
```

```
            考察其八邻域是否有白色像素
```

```
            如果有，就将(x,y)的坐标存入边界像素表 boundaryPixels
```

求重心像素坐标：  $xCenter = xSum / whiteSum$ ;  $yCenter = ySum / whiteSum$ ;

对 boundaryPixels 中的每个像素，求其到角度重心的角度 $\theta$ ( $0\sim 360^\circ$ ):

```
     $\theta = \text{atand}((y-yCenter)/(x-xCenter));$ 
```

```
    如果(x-xCenter) < 0 则  $\theta = \theta + 180$ ;
```

如果  $(x-xCenter) > 0$  &&  $((y-yCenter) < 0)$  则  $\theta = \theta + 360$ ;

将 boundaryPixels 中的所有边界像素按其对应的 $\theta$ 降序排列

## 代码实现

```
function [ J ] = outerBoundaryTracking( I )
%UNTITLED2 此处显示有关此函数的摘要
% 遍历，如果有邻域有黑的白色像素就是边界，把边界坐标存起来
% 用极坐标表示这些坐标，并排序，按角度从小到大输出成动画
[M, N ] = size(I);
J = zeros(M,N);

boundaryPixels = zeros(M*N,3); % 每行前两个变量是边界像素坐标，第三个像素是其
斜率 x/y，用于顺时针排序

numOfBoundaryPixels = 0;

xSum = 0;

ySum = 0;

whiteSum = 0;

% 找到所有边界点的坐标
for x = 1:M
    for y = 1:N
        if I(x,y) == 1 %是白色点
            %求中心点
            xSum = xSum + x;
            ySum = ySum + y;
            whiteSum = whiteSum+1;
        else
            hasWhiteNeighbor = 0;
            %考察其八领域是否有白色点
            for neighborX = x-1:1:x+1
                if hasWhiteNeighbor == 1
```

```

        break;
    end
    for neighborY = y-1:1:y+1
        tempX = neighborX;
        tempY = neighborY;
        %边缘处理
        if tempX < 1
            tempX = 1;
        end
        if tempY <1
            tempY =1;
        end
        if tempX >M
            tempX = M;
        end
        if tempY > N
            tempY = N;
        end
        % 判断是否有白色邻居
        if l(tempX,tempY) == 1
            hasWhiteNeighbor = 1;
            break;
        end
    end
end

%如果是黑点且有白色邻居，就认为是外边界
if hasWhiteNeighbor == 1
    numOfBoundaryPixels = numOfBoundaryPixels+1;

```



```

        boundaryPixels(numOfBoundaryPixels,1) = x;
        boundaryPixels(numOfBoundaryPixels,2) = y;
    end
end
end
end

%求中心点
xCenter = xSum / whiteSum
yCenter = ySum / whiteSum

%求以中心点为原点建立极坐标系后，每个点到中心点的角度 theta,存入
boundaryPixels(i,3)
for i =1:numOfBoundaryPixels
    x = boundaryPixels(i,1);
    y = boundaryPixels(i,2);
    boundaryPixels(i,3) = atand((y-yCenter)/(x-xCenter));
    if (x-xCenter) <0
        boundaryPixels(i,3) = boundaryPixels(i,3) + 180;
    elseif (x-xCenter) >0 && ((y-yCenter)<0)
        boundaryPixels(i,3) = boundaryPixels(i,3) + 360;
    end
end

end

% 按顺时针方向排序
boundaryPixels(numOfBoundaryPixels+1:end,:) = []; % 删除多余行
boundaryPixels = sortrows(boundaryPixels, 3,'descend');

% 显示结果

```

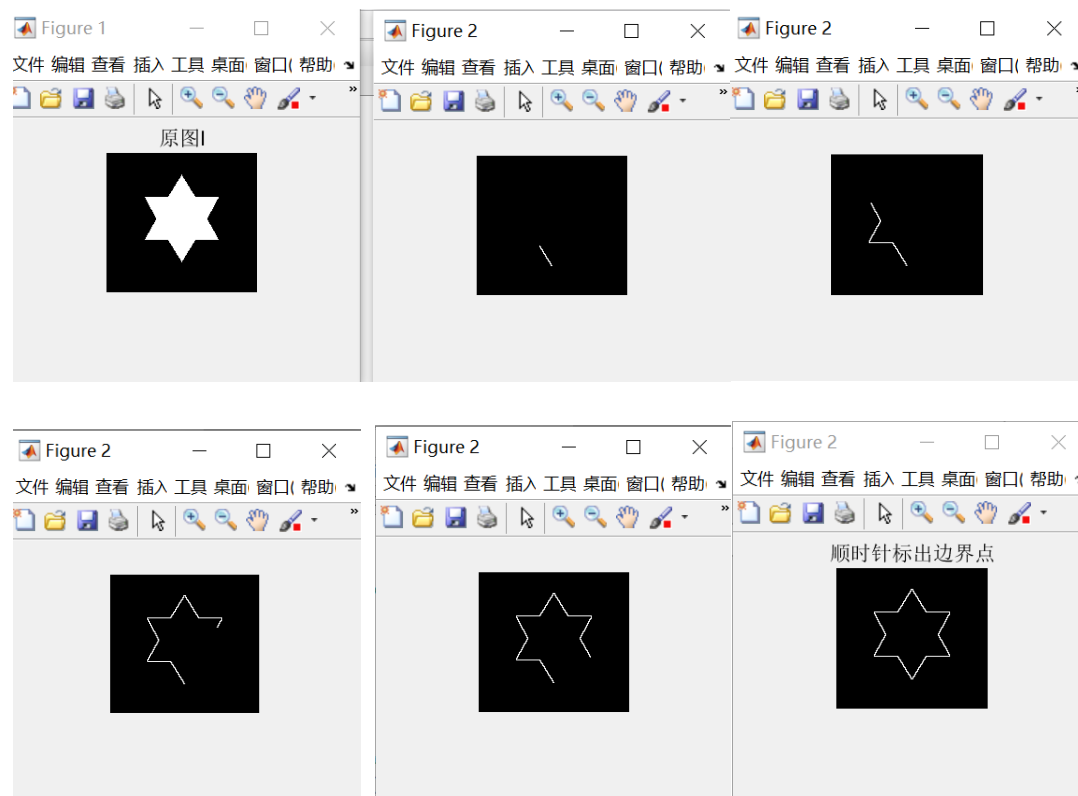
```
imshow(logical(I));title('原图 I');

figure(2),imshow(logical(J));title('顺时针标出边界点');
for i =1:numOfBoundaryPixels
    J(boundaryPixels(i,1),boundaryPixels(i,2)) = 1;
    figure(2),imshow(logical(J));title('顺时针标出边界点');
end

end
```

## 结果展示

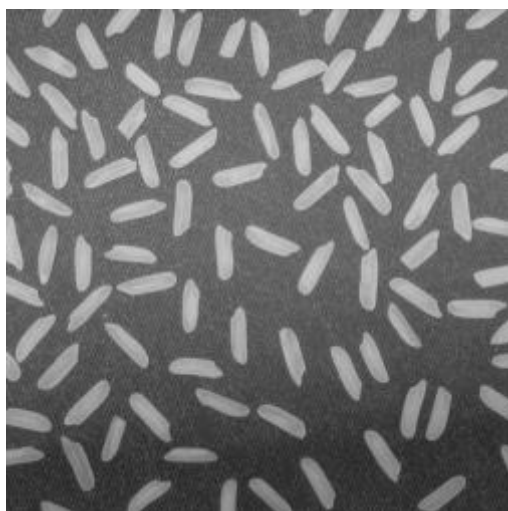
以动画形式顺时针绘出原图 I 的外边界：



## 6. 附加题：图中有多少粒米？

### 题目描述

用 MATLAB 输出下图中米粒个数。



### 算法详解

1. 首先进行中值滤波，去除噪声
2. 先用开运算获得背景
3. 滤波后的图像减去背景得到光照均匀的图像
4. 再用自适应阈值将图像二值化
5. 再次进行开操作，使一些连在一起的米粒分开
6. 对图像进行区域标记
7. 区域个数即为米粒个数。（最后算出答案为 96 个）

### 代码实现

```
function [numOfRiceGrains] = countRice()

%countRice 此处显示有关此函数的摘要

% 此处显示详细说明

numOfRiceGrains = 0;

I = imread('rice.png');

I = rgb2gray(I);
```

```

I = medfilt2(I,[3,3]); % 中值滤波
background =imopen(I,strel('disk',15)); %开操作得到背景
I2=imsubtract(I,background); %减去背景，消除光照不均匀效应
level=graythresh(I2); %获得自适应阈值
IFinal=im2bw(I2,level); %阈值分割得到二值图像
IFinal = imopen(IFinal,strel('disk',3)); % 开操作分开连在一起的米粒

% labeled 是处理后的矩阵， numObjects 是米粒的个数;
[labeled,numOfRiceGrains]=bwlabel(IFinal,8);

% 显示结果
subplot(221)
imshow(uint8( I));title('原图');
subplot(222)
imshow(uint8(background));title('背景');
subplot(223)
imshow(uint8(I2));title('背景均匀后');
subplot(224)
imshow(logical(IFinal));title('最终用于计算的图像');

end

```

结果展示

