hw3

171860617 肖维城

实验平台

windows10
matlab R2017b

文件说明

文件名	说明	
mycorr2.m	自己实现的相关操作函数	
myconv2.m	卷积函数	
GLPSpatialfilter.m	产生n*n高斯低通空间滤波器的函数	
gradSobel.m	使用Sobel算子计算图像梯度	
setdirection.m	设置梯度的方向	
non_max_supression.m	非最大抑制	
mycanny.m	Canny算法的实现	
mooreneighbortracing.m	Moore-Neighbor Tracing算法的实现	
applyCanny.m	应用Canny算法	
applyMooretracing.m	应用Moore-Neighbor Tracing算法	

Canny边缘检测算法

主要有五个步骤：

1.高斯函数平滑输入图像f：
$f_s(x,y)=G(x,y)\bigstar f(x,y), \bigstar$表示卷积

2.计算f_s的梯度

3.非最大抑制(将粗边缘变细)

4.滞后阈值(减少伪边缘点)

5.连通性分析(连接边缘)

卷积和相关

使用matlab实现卷积和相关计算函数

卷积：myconv2(A,B,shape)

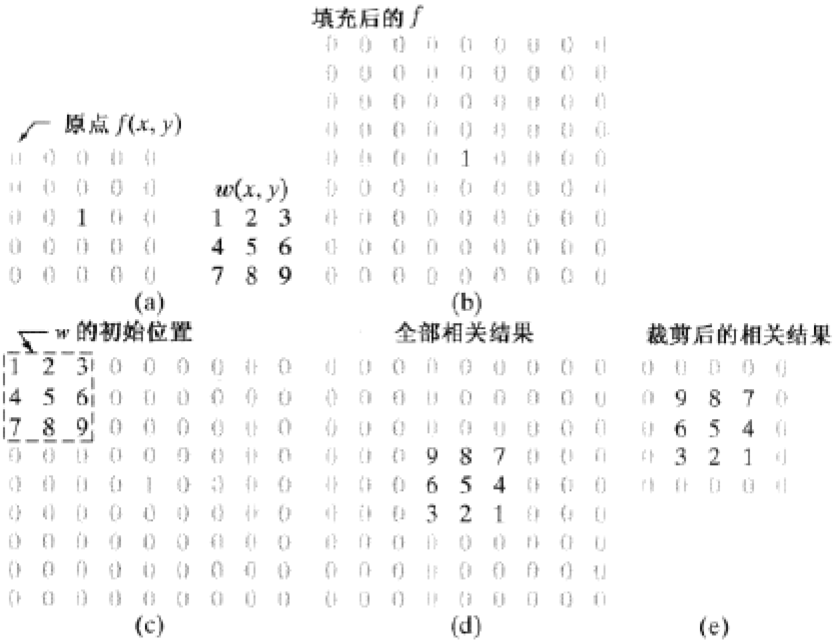
相关：mycorr2(A,B,shape)

实现

输入：A(m*n)(图像), B(p*q)(滤波器：p==q=2k+1, 3*3, 5*5,..., (2k+1)*(2k+1)), shape('full','same','valid')

输出：经过滤波器B滤波的图像C

相关操作过程如图：



先对图像A进行0填充：对顶部和底部，左侧和右侧各填充(p-1)行/列0；

然后使用B对A进行滑动阵列乘积求和；

参考matlab提供的库函数conv2，使用参数shape定义输出矩阵的大小：

shape=='full'：输出矩阵是拓展后的矩阵，大小比原图像矩阵大；

shape=='same'：输出矩阵与原图像矩阵大小一致；

shape=='valid'：不对图像A进行填充，直接进行滑动乘积求和；输出矩阵大小比原图像矩阵小；

例如：

```
A =

     0     2     1     3
     2     4     0     0
     2     3     2     0
     2     1     0     3
```

```
B =

     1     2     3
     4     5     6
     7     8     9
```

```
c_full =

     0     2     5    11     9     9
     2    16    28    41    21    18
    10    47    69    84    39    27
    24    71    93    70    18     9
    22    51    73    61    33    18
    14    23    26    30    24    27
```

```
c_same =

    16    28    41    21
    47    69    84    39
    71    93    70    18
    51    73    61    33
```

```
c_valid =

    69    84
    93    70
```

```
E = ones(3,3)
F = ones(3,3)
myconv2(E,F,'valid') = 9
```

代码

myconv2.m：

```
function C = myconv2(A,B,shape)
%myconv2 convolution of image-A and kernel-B
% 卷积需要将B旋转180度，之后的步骤与相关操作一样;

%rotate 180
Br = rot90(B,2);
%correlation
C = mycorr2(A,Br,shape);

end
```

mycorr2.m:

```
function Out = mycorr2(A,B,shape)

[m,n] = size(A);
[p,q] = size(B);

%paddedsize:
PM = m+2*(p-1);
PN = n+2*(q-1);

%zero padding
Cpadded = zeros(PM,PN);
C = zeros(PM,PN);
Cpadded(p:p+m-1,q:q+n-1) = A;

%calc loop variables
a = (p-1)/2;
b = (q-1)/2;
rstart = (p+1)/2;
rend = PM-a;
cstart = (q+1)/2;
cend = PN-b;

%calc
for i=rstart:1:rend
    for j=cstart:1:cend
        tmp = Cpadded(i-a:i+a,j-a:j+a).*B;
        C(i,j) = sum(sum(tmp));
    end
end

%cut
if strcmp(shape,'full')
    Out = C(rstart:rend,cstart:cend);
elseif strcmp(shape,'same')
    Out = C(p:PM-p+1,q:PN-q+1);
```

```
elseif strcmp(shape,'valid')
    rs = rstart + p-1;
    re = rend - (p-1);
    cs = cstart + q-1;
    ce = cend - (q-1);
    Out = C(rs:re,cs:ce);
end

end
```

高斯模糊

2D-Gaussian function

$G(x,y)=\frac{1}{2\pi\delta^2}e^{-\frac{x^2+y^2}{2\delta^2}}$

对于一个n*n的高斯滤波器，设其中心位置坐标为(0,0)，则其它位置的坐标也可以随之确定；由于高斯函数的对称性，只要中心位置坐标相同，即使坐标轴方向不同，计算出来的滤波器也相同；

例如，对于3*3的滤波器：

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

只需要根据坐标带入高斯函数计算对应值即可；

高斯滤波器生成

对于一个n*n滤波器(n=2k+1)：中心坐标为(k+1,k+1)，其x和y坐标的范围都是$[-k,k]$；

代码实现如下：

```matlab
function G = GLPSpatialfilter(n,delta)
%GLPSpatialfilter 高斯低通空间滤波器
%    此处显示详细说明

k = (n-1)/2;
[x,y]=meshgrid(-k:k,-k:k);

G = exp(-(x.*x+y.*y)./(2*delta*delta));
G = G/sum(sum(G));

end
```

高斯模糊:

```matlab
%高斯模糊
G = GLPSpatialfilter(n,delta);
bluredgray = myconv2(gray,G,'same');
figure(2),imshow(bluredgray,[]);
```

梯度

定义

$$\nabla f \equiv grad(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

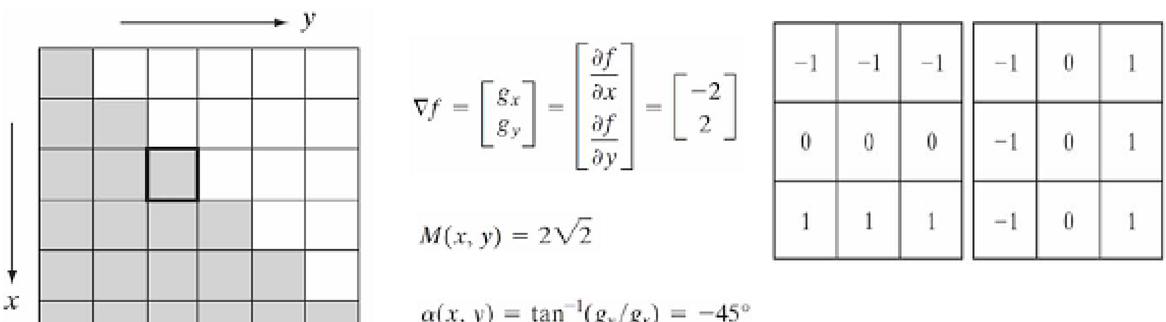
大小和方向

$M(x,y) = mag(\nabla f) = \sqrt{g_x^2 + g_y^2}$

$\alpha(x,y) = arctan\left(\frac{g_y}{g_x}\right)$

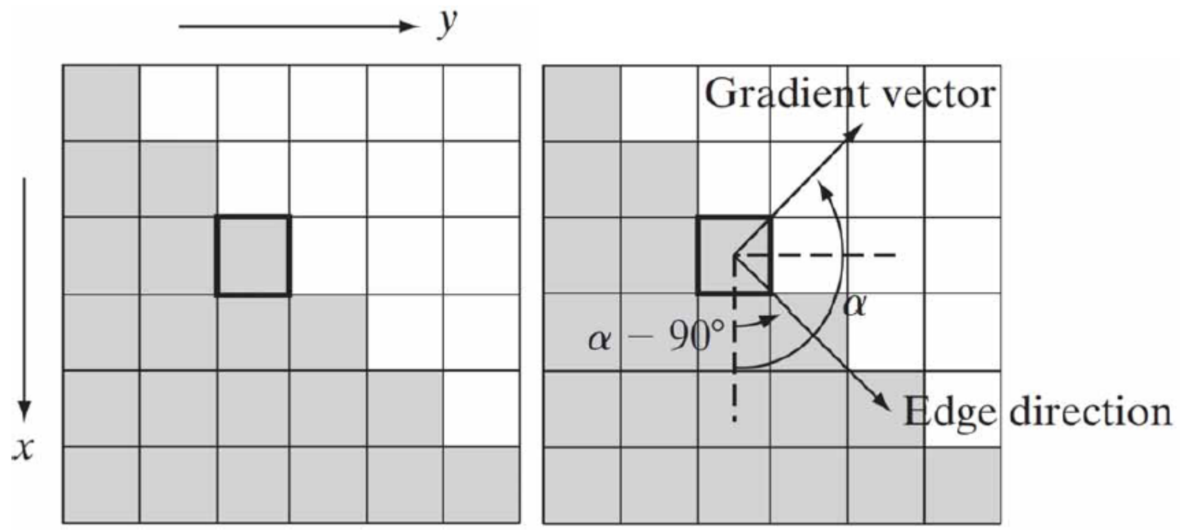
几何意义

以黑色方框为中心的3*3领域，灰色设为0，白色设为1，梯度计算如图：



梯度计算模板

梯度向量与边缘方向垂直

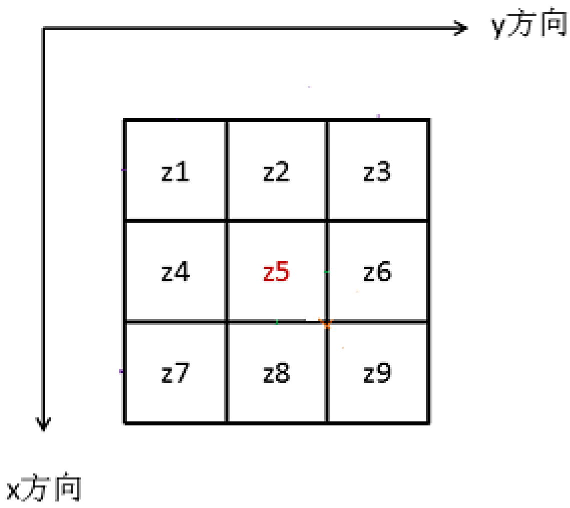


计算

计算图像上每个像素的偏导数的数字近似：

$$
\begin{aligned}
g_x &= \frac{\partial f(x,y)}{\partial x} = f(x+1,y) - f(x,y) \\
g_y &= \frac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y)
\end{aligned}
$$

Sobel算子



$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

使用阵列相乘求和计算gx：

$$
\begin{matrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{matrix}
$$

使用阵列相乘求和计算gy：

$$
\begin{matrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{matrix}
$$

代码实现

```
function [gx, gy, M, a] = gradSobel(img)
%gradSobel(img) 使用Sobel算子计算img的梯度
%    此处显示详细说明

sobelx = [-1 -2 -1; 0 0 0; 1 2 1]
sobely = [-1 0 1; -2 0 2; -1 0 1]

gx = mycorr2(img,sobelx,'same');
gy = mycorr2(img,sobely,'same');

M = sqrt(gx.*gx+gy.*gy);
a = atan2d(gy,gx);

end
```

上述代码中使用atan2d直接计算出每个像素位置的梯度方向对应角度a;

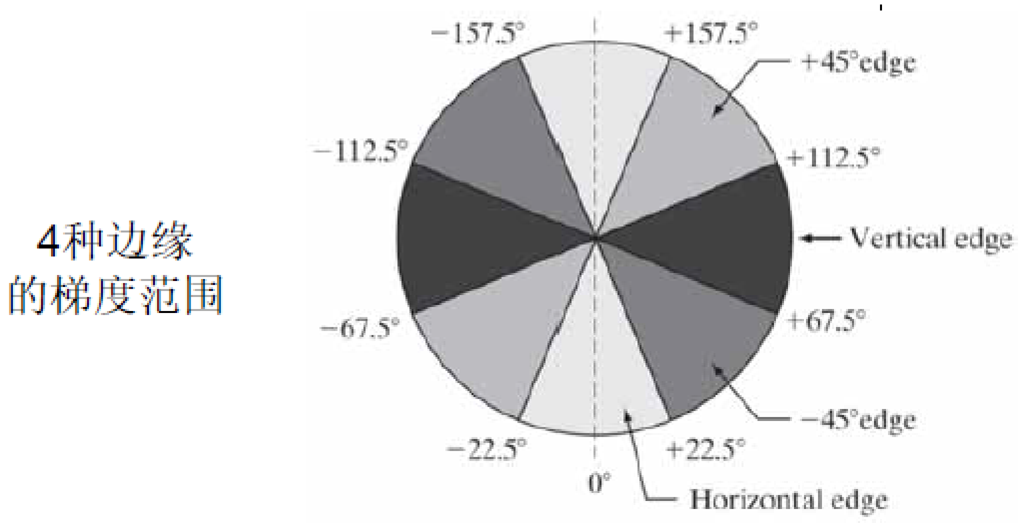
调用上述函数计算高斯模糊后的图像的梯度:

```
%计算梯度
[gx, gy, M, a]=gradSobel(bluredgray);
```

非最大抑制

利用该方法将边缘变细;

梯度的方向可分为水平,垂直,+45度,-45度四个方向,划分为如图范围:



根据gradSobel.m中求出的角度矩阵a,即可得到梯度在上图中所属范围;

对任一位置(x,y),若该位置梯度的模值M(x,y)比梯度方向上任一邻居的数值小,则对其抑制:即将其值置为0;否则保留原值。

首先确定梯度方向:输入梯度计算产生的角度矩阵a,按照角度将四个方向分别设置不同数值代表;产生梯度方向矩阵dir

```
function dir = setdirection(a)
%setdirection(a): 根据角度矩阵a设置各个确定梯度方向
%1: 垂直梯度, 水平边
%2: +45度梯度
%3: 水平梯度
%4: -45度梯度

dir1 = ((-22.5<=a&a<22.5)|(abs(a)>=157.5));
dir1 = dir1*1;

......

end
```

然后对每一个位置(i,j)，读取方向矩阵dir相应值判断其方向，取该方向的两个邻居的mag值与其比较；

```
function out = non_max_supression(dir,mag)
%non_max_supression(dir,mag) 非最大抑制
%   mag: 梯度的模; dir: 梯度方向
[m,n] = size(mag);

out = mag;

i:1->m,j:1->n:
    switch dir(i,j)
        case 1
            %垂直梯度
            if(i-1>=1)
                out(i,j)=(mag(i,j)>=mag(i-1,j))*out(i,j);
            end
            if(i+1<=m)
                out(i,j)=(mag(i,j)>=mag(i+1,j))*out(i,j);
            end
        case 2
            %+45度梯度方向(西北-东南方向)
            ......
        case 3
            %水平梯度方向
            ......
        case 4
            %-45度梯度方向(东北-西南方向)
            ......
    end
end
```

根据之前梯度计算中(gradSobel.m)产生的角度矩阵a，计算出梯度方向矩阵dir，然后根据方向dir和模M进行非最大抑制过程：

```
%非最大抑制
%根据梯度方向角度设置方向
dir = setdirection(a);
Mag = non_max_supression(dir,M);
figure(5),imshow(Mag);
```

滞后阈值

使用该方式减少伪边缘点;

设定两个阈值：T_L和T_H，比值2 : 1或者3 : 1，$T_H > T_L$;

利用阈值T_H得到强边缘点:
$g_{NH}(x,y) = g_N(x,y) \geq T_H$

利用阈值T_L得到弱边缘点:
$g_{NL}(x,y) = g_N(x,y) \geq T_L$
$g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)$

```
%滞后阈值
%强边缘
g_NH=(Mag>=high);
g_NL=(Mag>=low);
%弱边缘
g_NL = g_NL-g_NH;
figure(6),imshow(g_NH);
figure(7),imshow(g_NL);
```

连通性分析

遍历g_{NH}中的每一个点p，保留g_{NL}中与p连通的点，去掉g_{NL}中剩余的点，合并g_{NH}和g_{NL}

```
%连通性分析
%八个邻居位置偏移量
neigh=[-1 -1;-1 0;-1 1;0 -1;0 1;1 -1;1 0;1 1];
%padding
gNHpad = padarray(g_NH,[1,1],'replicate');
gNLpad = padarray(g_NL,[1,1],'replicate');
res = zeros(size(gNLpad));
%保留gNL中八连通的点，并将gNH和gNL合并到res中
for i=2:1:size(gNLpad,1)-1
    for j=2:1:size(gNLpad,2)-1
        if gNHpad(i,j)>0
            %将gNH中的点加入res
            res(i,j)=gNHpad(i,j);
            %遍历当前点的八个邻居，将gNL中的非零值加入res
            for k=1:8
                if gNLpad(i+neigh(k,1),j+neigh(k,2))>0
 res(i+neigh(k,1),j+neigh(k,2))=gNLpad(i+neigh(k,1),j+neigh(k,2));
                end
            end
        end
    end
end
```

完整代码见mycanny.m文件

实现效果

用法见applyCanny.m;

lenna转为灰度图：

使用高斯模糊(n=13,delta=2.0):

梯度的模：

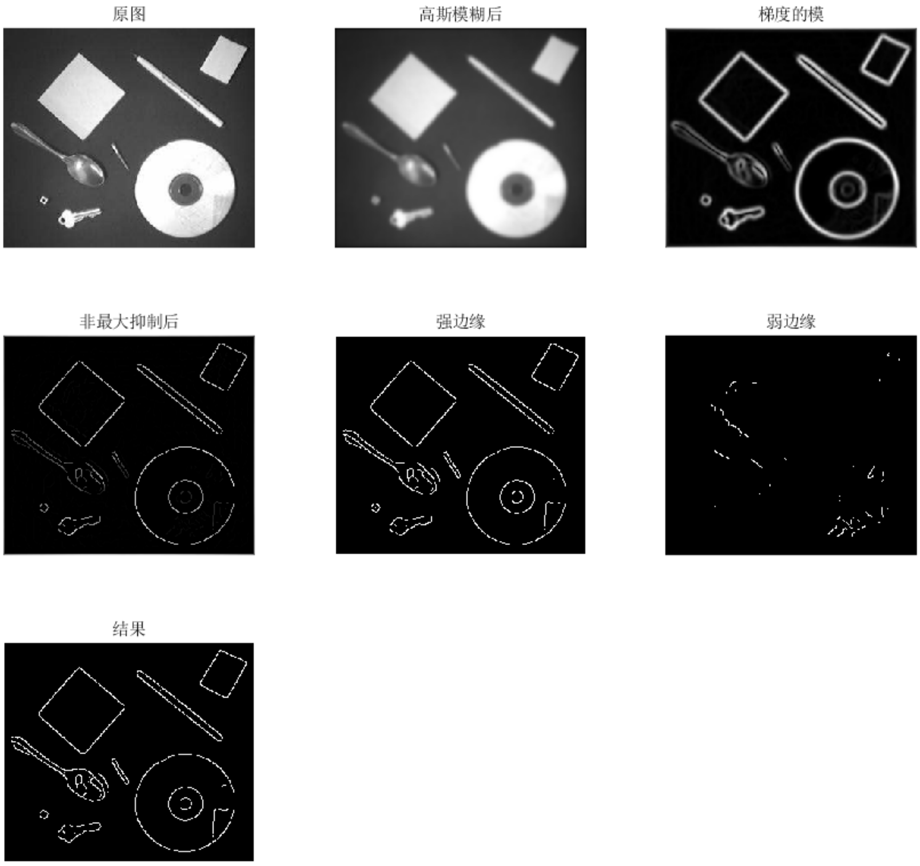
非最大抑制后：

强边缘(阈值0.1)：

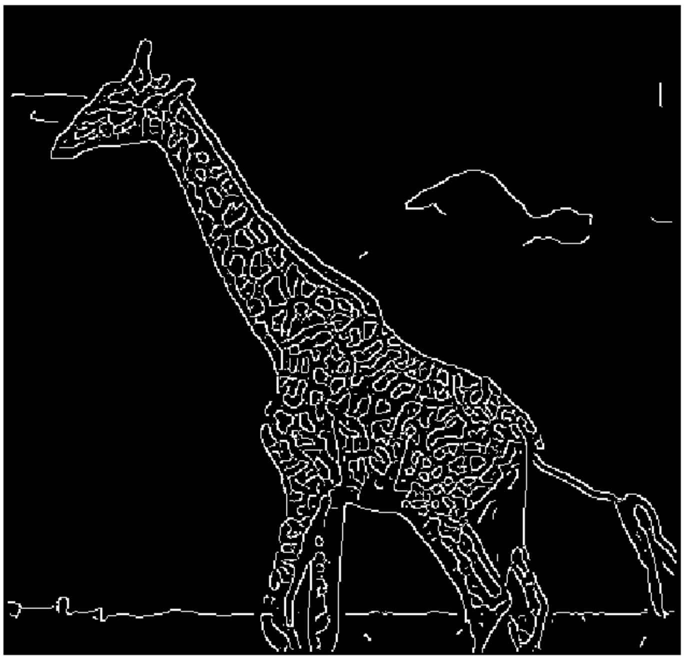
弱边缘(阈值0.04)：

最后的结果：

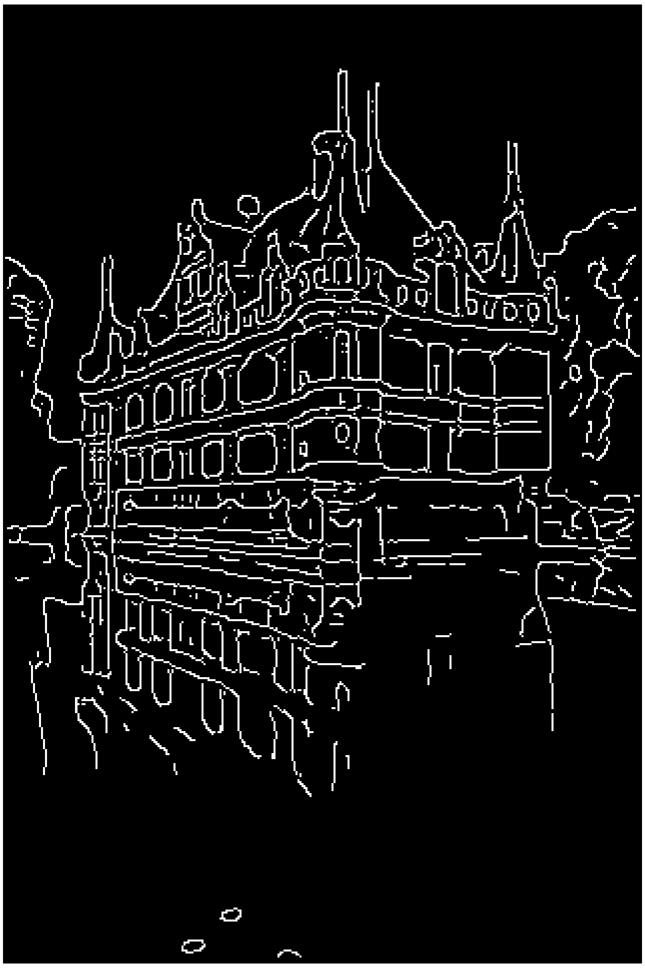
disk.jpg边缘(n=13,delta=2.0,TL=0.04,TH=0.1)：



giraffe.jpg边缘(n=13,delta=2.0,TL=0.04,TH=0.1)：

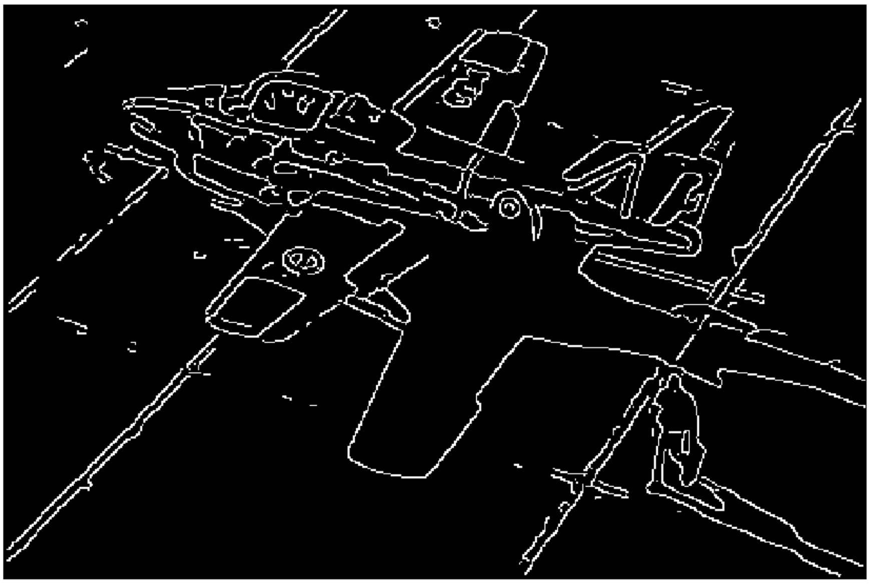


castle.jpg的边缘：

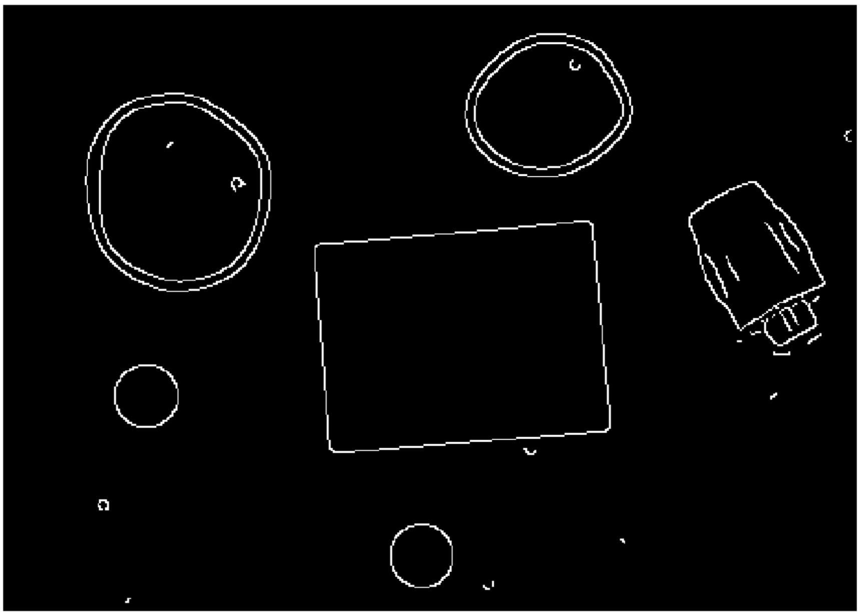


leaf.jpg的边缘：

plane.jpg的边缘：



rubberband_cap.png的边缘：



边缘追踪(Moore-Neighbor Tracing)

使用Moore-Neighbor算法进行边缘追踪;

Moore Neighborhood

Moore领域：像素点P的八个邻居，如图所示：

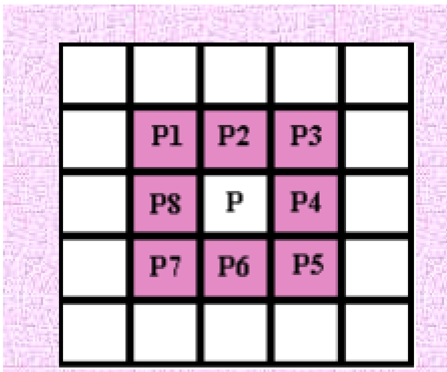


Figure 1

算法过程

输入二值图，指定需要追踪的边缘上的一个点P作为起始点，按照顺时针或者逆时针顺序遍历P的Moore邻域(整个算法过程中遍历顺序需要保持一致)，每次遇到值为1的点，同样对该点的Moore邻域进行遍历，直到遇到一个值为1的点，重复上述过程。当再次回到起始点后算法终止。

伪代码如图：

Input: A square tessellation, **T,** containing a connected component **P** of black cells.

Output: A sequence **B** (b_1, b_2 ,..., b_k) of boundary pixels i.e. the contour.

Define **M(a)** to be the Moore neighborhood of pixel **a**.
Let **p** denote the current boundary pixel.
Let **c** denote the current pixel under consideration i.e. **c** is in **M(p)**.

Begin

- Set **B** to be empty.
- From bottom to top and left to right scan the cells of **T** until a black pixel, **s,** of **P** is found.
- Insert s in **B**.
- Set the current boundary point **p** to s i.e. **p**=s
- Backtrack i.e. move to the pixel from which s was entered.
- Set **c** to be the next clockwise pixel in **M(p)**.
- While **c** not equal to s do

    If **c** is black
    - insert **c** in **B**
    - set **p**=**c**
    - backtrack (move the current pixel **c** to the pixel from which **p** was entered)

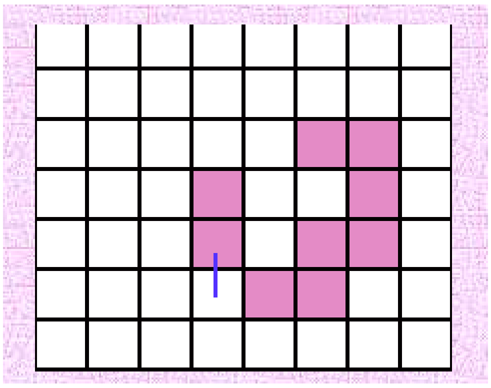
    else
    - advance the current pixel **c** to the next clockwise pixel in **M(p)**

  end While

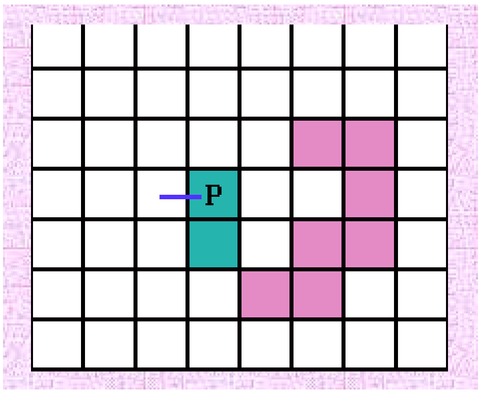
End

Jacob停止条件

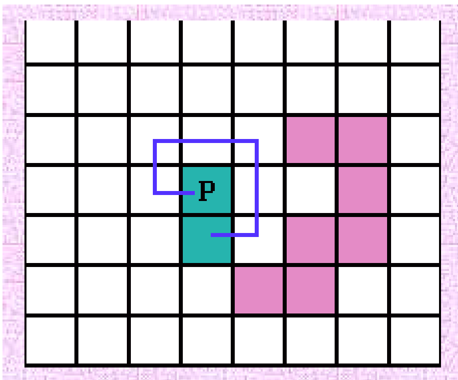
上述过程存在问题，如图：



从图中蓝色线所在的点开始，顺时针遍历，遇到邻域中的点P，再从P点开始遍历：



顺时针遍历P点的邻域，最后又回到起始点，根据终止条件，算法终止。



显然最后得到的边缘并不完整，算法在这种条件下不能正确运行，于是更改终止条件，使用Jacob终止条件；

Jacob's stopping criterion: the algorithm terminates when it visits the **start** pixel for a second time in the same direction it did the first time around.

当第二次从同一方向进入起始点才停止算法；

定义：

[2][3][4]
[1][X][5]
[8][7][6]

设上表表示点X的Moore邻域，按照数字顺序进行遍历：1->2->3->...->8

遍历时，若从同一个点遍历到起始点则视为从同一方向进入起始点；即将某一点X的方向定义为到达该点X前的点的位置；

例如按照顺时针顺序遍历，则有：8->1, 1->2, 2->3, 3->4, 4->5, 5->6, 6->7, 7->8, 8->1;

于是对应方向：8在1的邻域中位置7上，1在2的邻域中位置7上，2在3的邻域中位置1上......

所以1~8各位置的前一个点所在位置：7 7 1 1 3 3 5 5

matlab实现

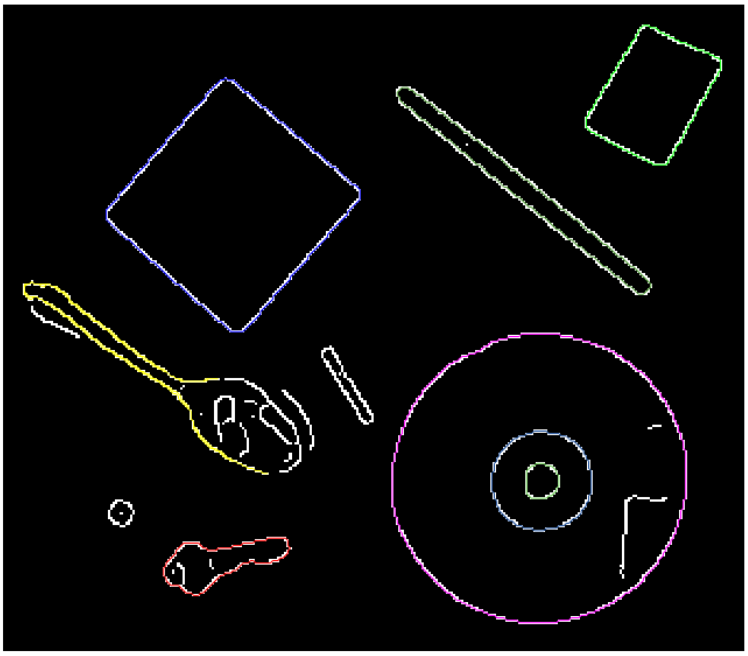
mooreneighbortracing.m：

```
function boundary = mooreneighbortracing(binimg,pos)
%mooreneighbortracing(binimg,pos)
%   binimg: 输入的二值图
%   pos: 给定位置(x,y)作为起始点,寻找该点所在边缘
%   boundary: 输出边缘所有点的坐标集合

initial_entry = pos;

% 点X的Moore邻域，按照数字顺序进行遍历
% [ 2 ][ 3 ][ 4 ]
% [ 1 ][ x ][ 5 ]
% [ 8 ][ 7 ][ 6 ]
% 八个邻居的坐标偏移：
neighborhood = [ 0 -1; -1 -1; -1 0; -1 1; 0 1; 1 1; 1 0; 1 -1 ];
exit_direction = [ 7 7 1 1 3 3 5 5 ];

% 遍历给定点的邻域，找到第一个值为1的点
for n = 1 : 8 % 8-connected neighborhood
    c = initial_entry + neighborhood( n, : );
    if binimg( c( 1 ), c( 2 ) ) == 1
        initial_position = c;
        break;
    end
end

% 基于找到的点的位置设置进入该点的方向
initial_direction = exit_direction( n );

% 将这个点的位置加入边界集合
boundary( 1, : ) = initial_position;

% 初始化循环变量
position = initial_position;
direction = initial_direction;
boundary_size = 1;

% 循环查找
while true

    % 顺时针查找
    for i=1:8
        n = mod(direction + i - 1, 8);
        if n==0
            n = 8;
        end
        c = position + neighborhood(n,:);
        if binimg(c(1),c(2))==1
            position = c;
            break;
        end
    end
```

```matlab

    % 根据找到的点的信息进行更新
    direction = exit_direction( n );
    boundary_size = boundary_size + 1;
    boundary( boundary_size, : ) = position;

    % Jacob's stopping criterion
    if all( position == initial_position ) &&...
    ( direction == initial_direction )
        break;
    end
end

end

```

实现效果

对disk部分图形的边缘进行追踪；



reference

《数字图像处理》第三版 Rafael C. Gonzalez编著

卷积：

```
https://www.cnblogs.com/xiaojianliu/p/9076547.html
https://www.cnblogs.com/yibeimingyue/p/10879506.html
https://blog.csdn.net/mrahut/article/details/81539435
https://www.cnblogs.com/alexanderkun/p/8149088.html
https://www.cnblogs.com/hyb221512/p/9370367.html
```

GLPfilter：

```
https://blog.csdn.net/ckghostwj/article/details/12177273
https://www.cnblogs.com/pacino12134/p/11372555.html
https://www.cnblogs.com/wangguchangqing/p/6407717.html
https://www.cnblogs.com/ImageVision/archive/2012/06/11/2545555.html
```

图像梯度：

```
https://zhuanlan.zhihu.com/p/64350303
https://blog.csdn.net/baidu_38172402/article/details/88991375
https://www.jianshu.com/p/96b77afd323b
```

Moore-Neighbor-Tracing：

```
http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/moore.html
https://www.mathworks.com/matlabcentral/fileexchange/42144-moore-neighbor-boundary-trace
```

Jacob's stopping criterion

```
http://www.imageprocessingplace.net/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/raymain.html
```