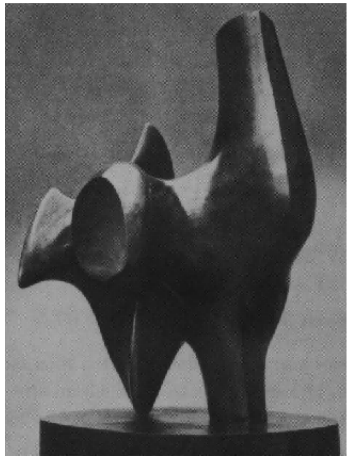
第9章
边缘检测

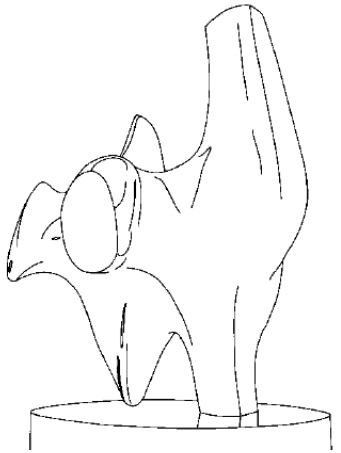
目录

背景知识

- **图像灰度或颜色的显著变化，对感知和理解图像非常重要。通常把那些灰度或色彩显著变化的点称为边缘点，邻接连通的边缘点构成的线段，称为边缘（edge）；位于不同物体区域之间的边缘称为边界（boundary），由围绕一个物体区域的边界所形成的闭合通路称为轮廓（contour）。**

  - 边缘对于人类视觉感知非常重要，如漫画或素描，寥寥数笔线条就可以清楚地描绘一个物体或者场景。
  - 图像锐化的实质就是检测并增强边缘，以提高图像中物体的可识别度。





在各边缘线的两边，图像的灰度值有明显的不同。

背景知识

- 物体检测与识别（object detection and recognition）是图像分析和计算机视觉的研究重点。
- 由于边缘是物体与背景之间、不同物体之间的边界，这意味着，如果能够准确地识别图像中的边缘，就可以定位并测量物体区域的面积、轮廓周长和形状等基本属性，进而对图像中的物体进行识别和分类。
- 因此，边缘检测（edge detection）是图像分析不可或缺的工具。

回顾 - 图像处理的目的

- **Image improvement – low level *Image Processing***
  - Improvement of pictorial information for human interpretation (Improving the visual appearance of images to a human viewer)
- **Image analysis – high level *Image Processing***
  - Processing of scene data for autonomous machine perception *(Preparing images for measurement of the features and structures present)*
  - Extracting information from an image
    - Step 1 : segment the image → objects or regions (Region of interest)
    - Step 2 : describe and represent the segmented regions in a form

      suitable for computer processing
    - Step 3 : image recognition and interpretation

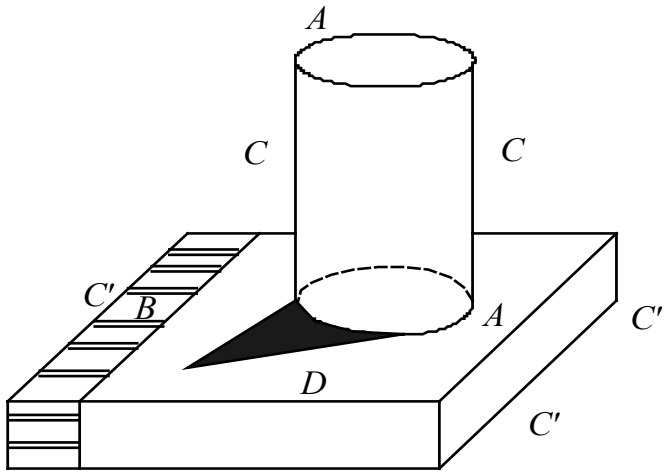
什么是图像分割(segmentation)?

- **图像分割的定义Definition**
  - Subdivides an image into its constituent regions or objects
  - 从图像中提取出所需的语义对象(semantic object)
  - 将图像划分成若干有一定含义的区域
- **Heavily rely on one of two properties of intensity values:**
  - Discontinuity（不连续性）---- Partition based on *abrupt changes* in intensity, e.g. edges in an image
    - point / line / edge / corner detection
  - Similarity（相似性） ---- Partition based on intensity similarity.
    - thresholding
    - region growing / splitting / merging

边缘的形成与分类

- **物体边缘以图像的局部特征不连续的形式出现，即图像局部亮度、颜色变化最显著的部分，例如灰度值的突变、颜色的突变、纹理结构的突变等，同时物体的边缘也是不同区域的分界处。**
  - *A*类--空间曲面上的不连续点
  - *B*类--由不同材料或相同材料不同颜色产生的
  - *C*类--物体与背景的分界线，一般称为轮廓线
  - *D*类--阴影引起的边缘

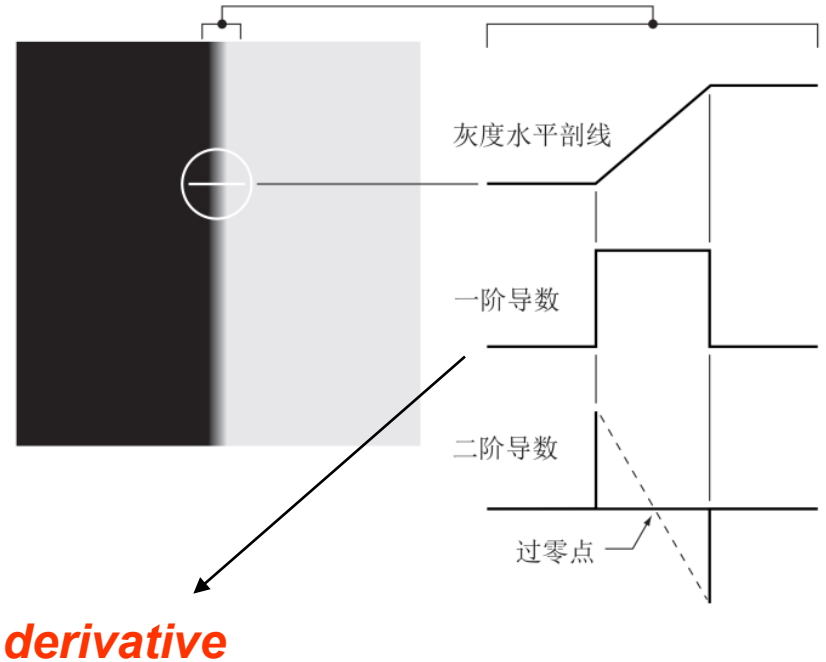
右图画出了一幅图像中的边缘点，仅仅根据这些边缘点，就能识别出三维物体， 可见边缘点确实包含了图像中的大量信息。



图像中的边缘

9.1 基于梯度的边缘检测

图像的一阶、二阶导数



1st and 2nd Derivative of gray level

- **The magnitude of the *first derivative* can be used to detect the presence of an edge at a point in an image (i.e., to determine if a point is on a ramp).**
- **The sign of the *second derivative* can be used to determine whether an edge pixel lies on the dark or light side of an edge.**
  - Producing 2 values for every edge in an image (an undesirable feature).
  - Center of a thick edge is located at the zero crossing.

边缘检测的基本步骤

- 对图像进行平滑处理-降噪
- 边缘点的检测，以提取图像中所有的边缘候选点
- 边缘定位，从候选边缘点中确认真正的边缘点

图像梯度

- **图像$f(x,y)$为二元函数，其沿x、y坐标轴的导数称为偏导数，由$f(x,y)$沿x轴和y轴的一阶偏导数所构成的二维向量，称为像素(x,y)处的梯度向量，简称梯度（gradient），定义为：**

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

  - 梯度的幅值：为$f(x,y)$沿梯度向量方向的变化率，用$M(x,y)$表示，在含义明确时把梯度的幅值简称为梯度，即：

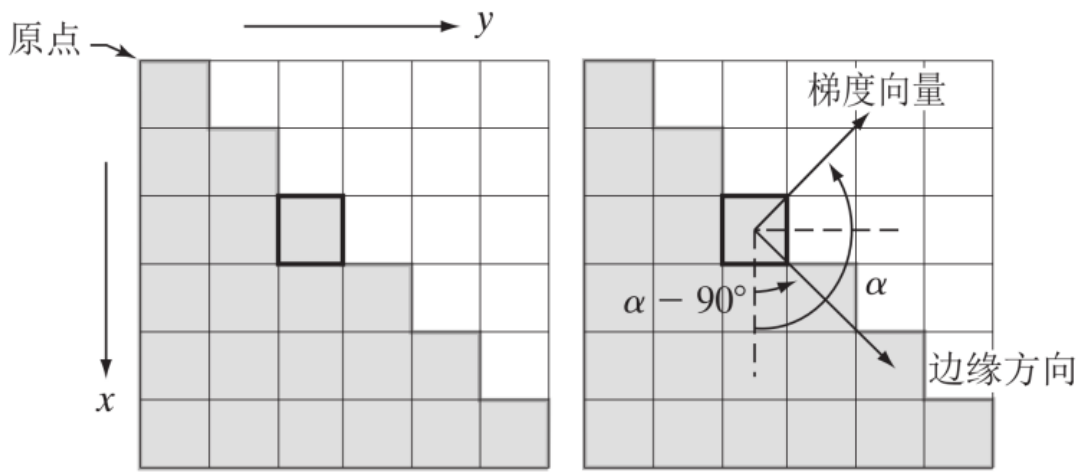
$$M(x,y) = \|\nabla f(x,y)\| = \sqrt{g_x^2 + g_y^2}$$

  - 梯度的方向：用相对于x轴正向的角度(x,y)给出：

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

梯度的性质

- 沿点(x,y)的梯度方向，函数$f(x,y)$增加最快。换句话说，点(x,y)的梯度方向是函数在这点的方向导数取得最大值的方向，梯度幅值就是方向导数的最大值。所谓方向导数，就是函数$f(x,y)$在点(x,y)处沿某一方向的函数变化率。
- 函数$f(x,y)$沿梯度的反方向减小最快，函数在此方向的导数达到最小值，为梯度幅值的负值。
- 沿梯度方向的正交方向，函数$f(x,y)$的变化率为零。



性质(1)表明，像素(x,y)梯度幅值的大小反映了该像素的边缘强度，可据此判断该像素是否为边缘点。

性质(3)表明，像素(x,y)处的边缘方向与该点处的梯度方向垂直，即梯度方向就是该点处边缘的法线方向，这一性质常被用于精确的边缘定位与连接。

梯度的计算

基本梯度算子：

$$\begin{cases} g_x = f(x+1,y) - f(x,y) \\ g_y = f(x,y+1) - f(x,y) \end{cases}$$

Roberts交叉梯度算子：

$$\begin{cases} g_x = f(x+1,y+1) - f(x,y) \\ g_y = f(x+1,y) - f(x,y+1) \end{cases}$$

Prewitt 梯度算子：

$$\begin{cases} g_x = [f(x+1,y-1) + f(x+1,y) + f(x+1,y+1)] \\ \quad\quad -[f(x-1,y-1) + f(x-1,y) + f(x-1,y+1)] \\ g_y = [f(x-1,y+1) + f(x,y+1) + f(x+1,y+1)] \\ \quad\quad -[f(x-1,y-1) + f(x,y-1) + f(x+1,y-1)] \end{cases}$$

$f(x\text{-}1, y\text{-}1)$	$f(x\text{-}1, y)$	$f(x\text{-}1, y\text{+}1)$
$f(x, y\text{-}1)$	$\boldsymbol{f(x, y)}$	$f(x, y\text{+}1)$
$f(x\text{+}1, y\text{-}1)$	$f(x\text{+}1, y)$	$f(x\text{+}1,y\text{+}1)$

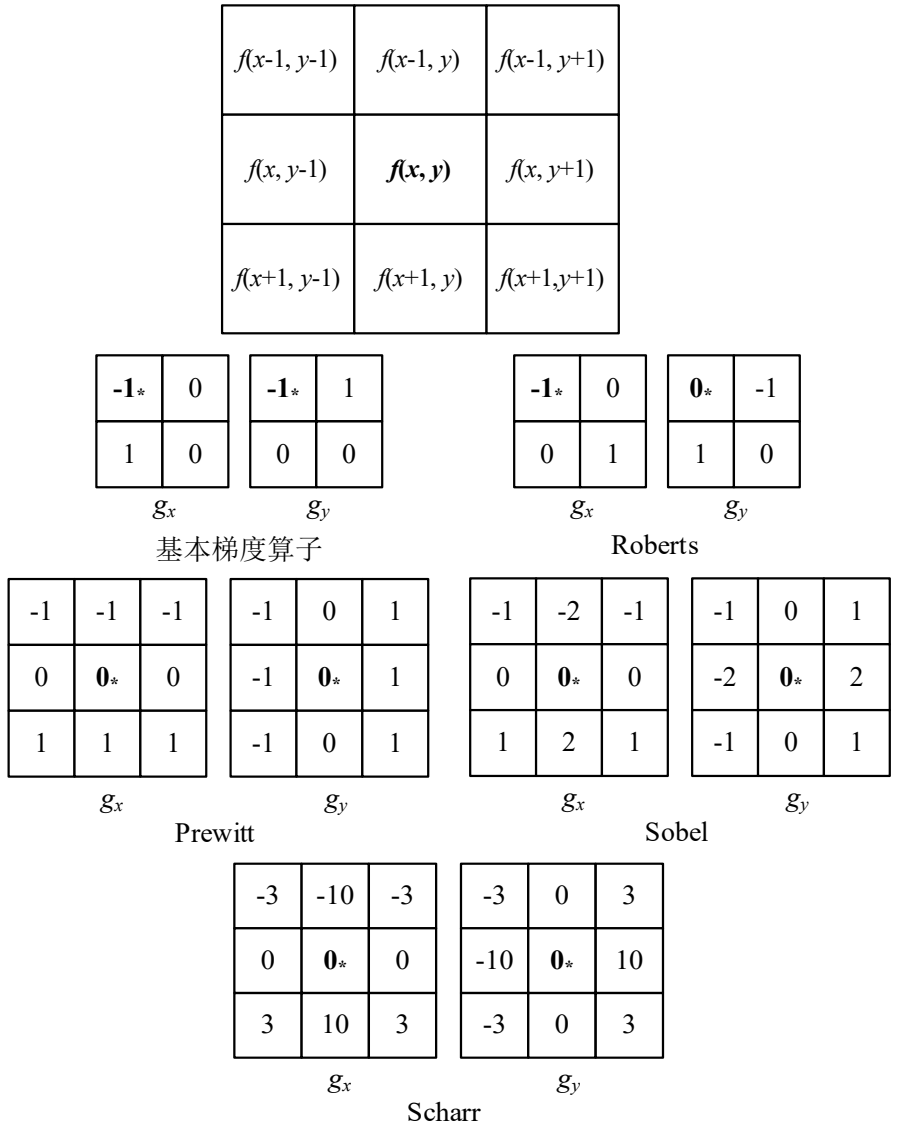
Sobel 梯度算子：

$$\begin{cases} g_x = [f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1)] \\ \quad\quad -[f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1)] \\ g_y = [f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)] \\ \quad\quad -[f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)] \end{cases}$$

Scharr 梯度算子：

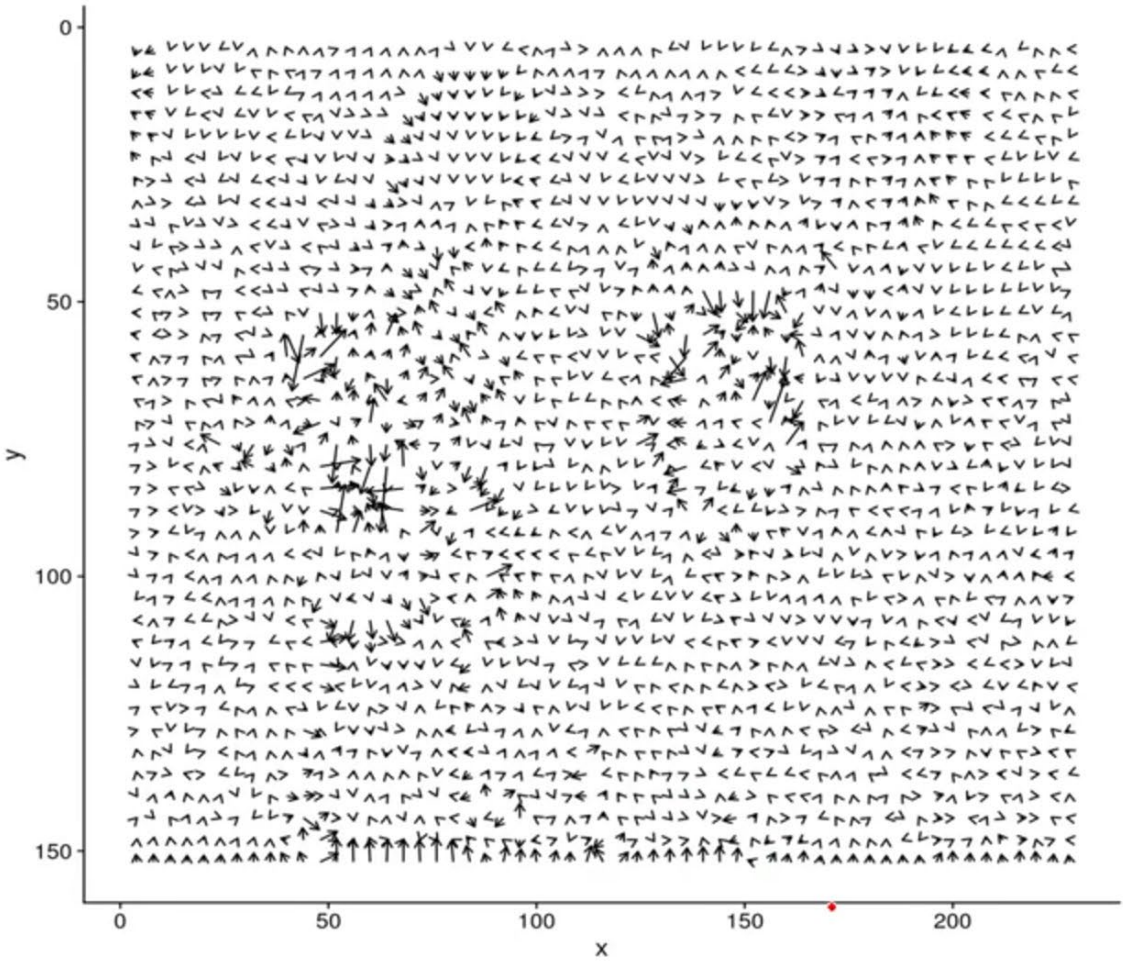
$$\begin{cases} g_x = [3f(x+1,y-1) + 10f(x+1,y) + 3f(x+1,y+1)] \\ \quad\quad -[3f(x-1,y-1) + 10f(x-1,y) + 3f(x-1,y+1)] \\ g_y = [3f(x-1,y+1) + 10f(x,y+1) + 3f(x+1,y+1)] \\ \quad\quad -[3f(x-1,y-1) + 10f(x,y-1) + 3f(x+1,y-1)] \end{cases}$$

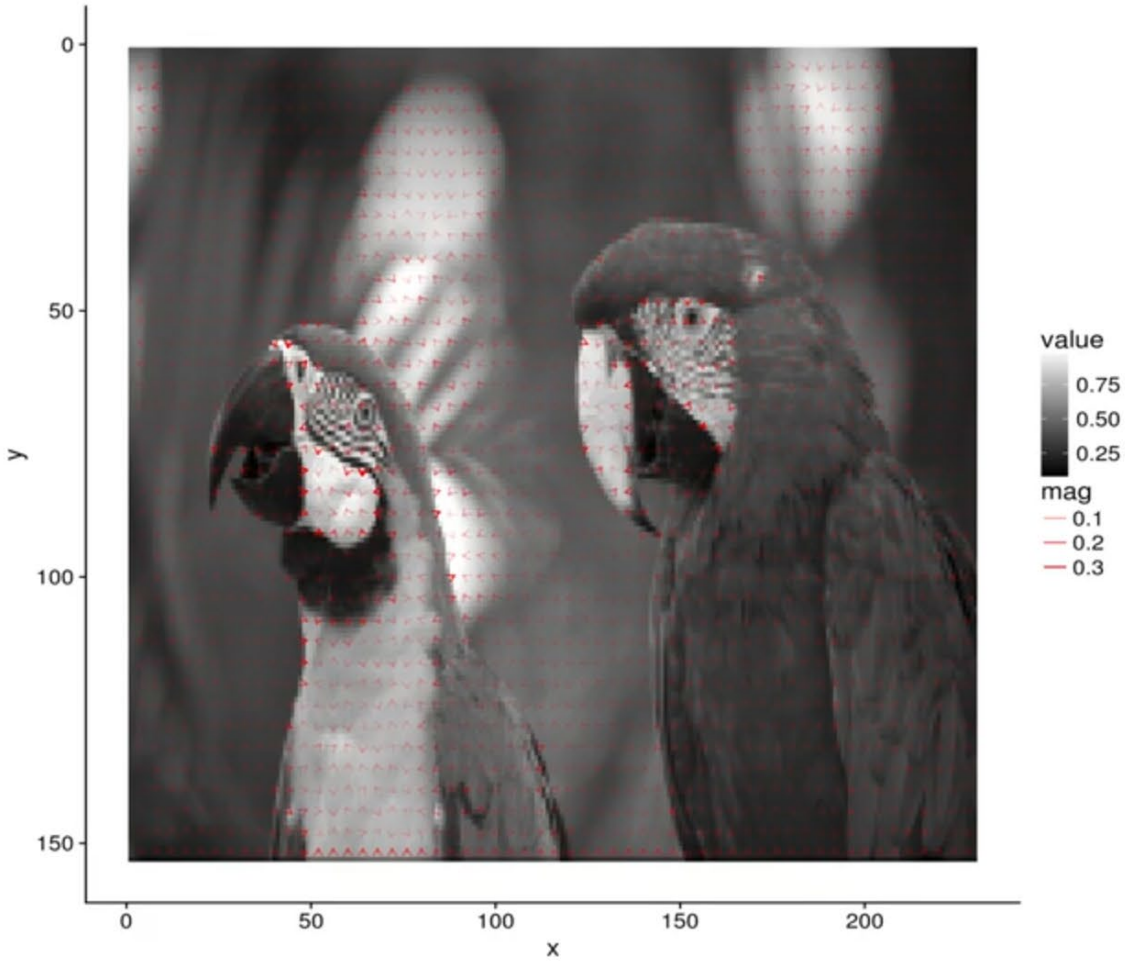
梯度算子的滤波模板



- 图像是二维离散函数，需用该点邻域内的像素值进行差分近似计算，称为梯度算子，或边缘检测算子。
- 梯度算子定义了一阶偏导数的近似计算方法，并用滤波器系数数组的方式指定参与计算的邻域像素及其权重，常用的梯度算子有基本梯度、Roberts、Prewitt、Sobel、Scharr梯度算子等。
- 所有梯度算子的滤波器系数之和为零，这样就可以保证在恒定灰度区域的响应为零。

梯度算子的滤波模板


0
50
100
150
y
0
50
100
150
200
x


0
50
100
150
y
0
50
100
150
200
x
value
0.75
0.50
0.25
mag
0.1
0.2
0.3

示例：图像梯度的计算

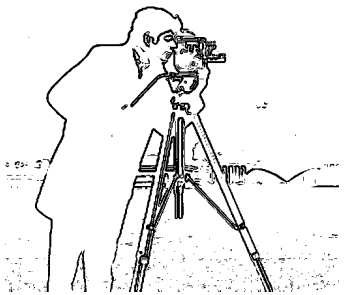
(1)cameraman图像

(2)梯度分量dx绝对值

(3)梯度分量dy绝对值

(4)梯度的方向角

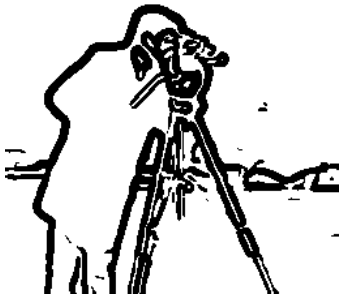
(5)梯度幅值



(6)梯度幅值的阈值分割

(7)高斯平滑滤波图像

(8)Scharr梯度幅值



(9)Scharr梯度幅值的阈值分割

- OpenCV、Scikit-image中图像坐标系的x轴水平向右、y轴垂直向下。
- 使用Sobel、Scharr梯度算子来计算该图像的梯度。

示例：图像梯度的计算

```
#导入本章示例用到的包
import numpy as np
import cv2 as cv
from skimage import io, util, filters, feature, transform, draw, color, morphology
from scipy import ndimage
import matplotlib.pyplot as plt
%matplotlib inline
#OpenCV: Sobel, Scharr算子图像梯度的计算示例
img = cv.imread('./imagedata/cameraman.tif', 0)  #读入一幅灰度图像
#计算图像水平方向x, 垂直方向y的梯度分量
dx, dy= cv.spatialGradient(img)
#获取梯度向量的幅值和方向角
magnitude, angle = cv.cartToPolar(np.float32(dx), np.float32(dy))
#用梯度幅度图像中最大值的0.2倍为阈值进行阈值分割
sobel_edgebw = magnitude > 0.20 * np.max(magnitude)
```

示例：图像梯度的计算

```
#采用标准差为2的高斯滤波器平滑原图像
img_smooth = cv.GaussianBlur(img,ksize=(0,0),sigmaX=2,sigmaY=2)
#采用Scharr算子计算梯度向量
scharr_x = cv.Scharr(img_smooth,ddepth=cv.CV_64F,dx=1,dy=0)
scharr_y = cv.Scharr(img_smooth,ddepth=cv.CV_64F,dx=0,dy=1)
#Scharr梯度向量幅值
scharr_edge = cv.sqrt(scharr_x**2 + scharr_y**2)
#用梯度幅度图像中最大值的0.2倍为阈值进行阈值分割
scharr_edgebw = scharr_edge > 0.20 * np.max(scharr_edge)
#显示结果（略，详见本章Jupyter Notebook可执行笔记本文件）
```

示例：Roberts交叉、Prewitt、Sobel、Scharr梯度算子边缘检测对比

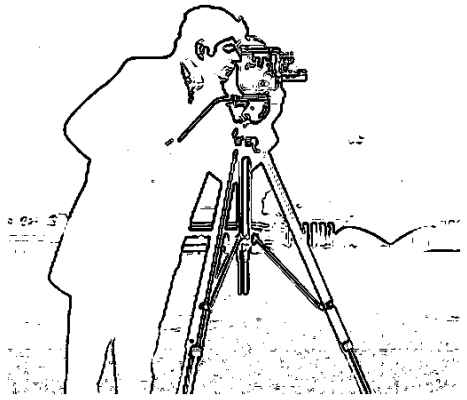
(1)Cameraman图像

(2)Roberts

(3)Prewitt

(4)Sobel

(5)Scharr



(6)Scharr阈值分割

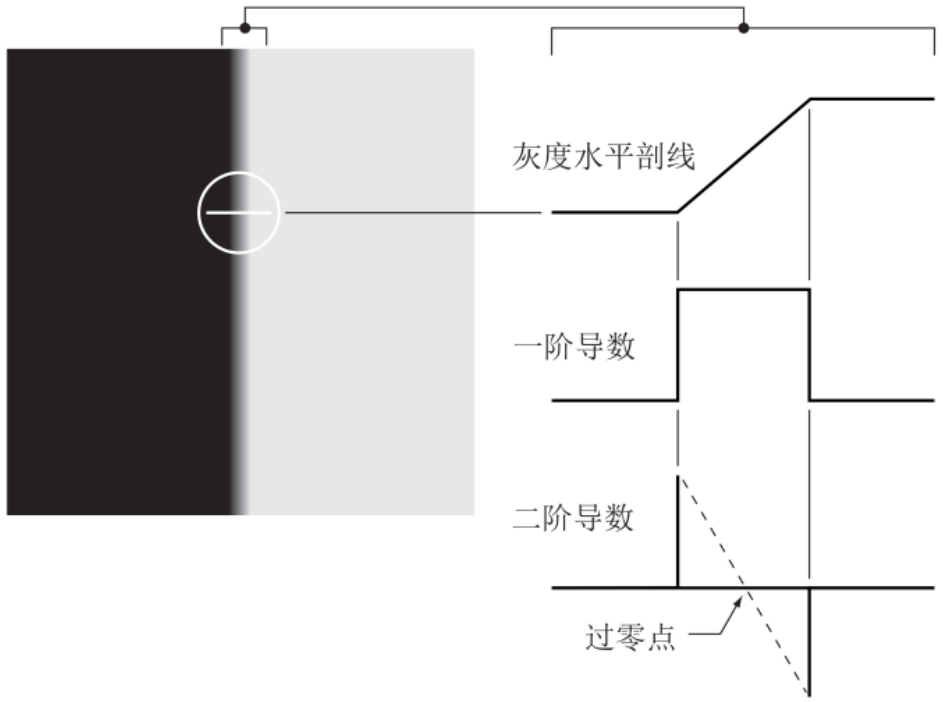
示例：Roberts交叉、Prewitt、Sobel、Scharr梯度算子边缘检测对比

```
#几种常用梯度算子图像边缘强度比较
img = io.imread('./imagedata/cameraman.tif')  #读入一幅灰度图像
edge_roberts = filters.roberts(img)   #Roberts交叉梯度算子
edge_prewitt = filters.prewitt(img)  #Prewitt梯度算子
edge_sobel = filters.sobel(img)        # Sobel梯度算子
edge_scharr = filters.scharr(img)     # Scharr梯度算子
edge_scharr_bw = edge_scharr > 0.20 * np.max(edge_scharr)
```

9.2 基于二阶导数的边缘检测

图像灰度变化及其导数

- 边缘是由于图像灰度显著变化形成的，因此一阶导数的大小可作为判断该像素是否为边缘点的依据。同时，将灰度一阶导数的局部极值点，即二阶导数为零的像素作为边缘点定位的依据。
- 二阶导数在过零点的邻域内发生符号改变，根据函数的凹凸性与二阶导数的关系，若某点的二阶导数大于零，那么该点邻域的函数图形是凹的，则该点位于图像局部较暗区域；若某点的二阶导数小于零，那么其邻域上的函数图形是凸的，则该点位于图像局部较亮区域。



拉普拉斯算子

- 拉普拉斯算子（Laplacian operator）由二元函数$f(x,y)$的两个非混合二阶偏导数构成，具有各向同性，定义为：

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$
$$= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4^* & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

拉普拉斯算子模板

高斯-拉普拉斯算子LoG

- 拉普拉斯算子对噪声非常敏感，一般在使用拉普拉斯算子之前，先使用高斯低通滤波器对图像进行平滑降噪。
- 由于拉普拉斯算子是线性运算，先用高斯低通滤波平滑图像、然后再施加拉普拉斯算子，等同于先计算高斯滤波函数的拉普拉斯二阶微分，再用该结果对图像做卷积。

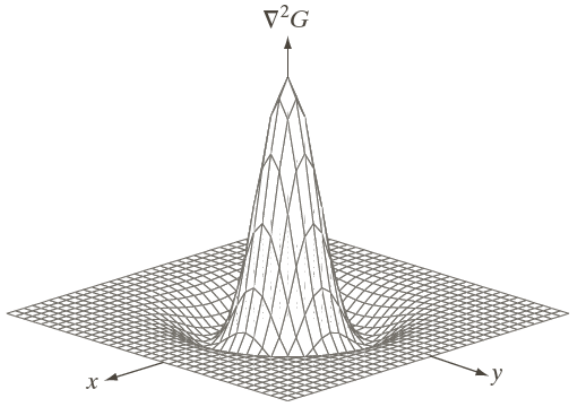
$$\nabla^2\left[G(x,y)*f(x,y)\right]=\left[\nabla^2 G(x,y)\right]*f(x,y)$$

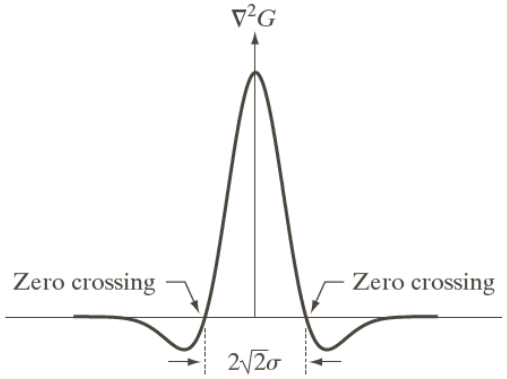
$$G(x,y)=\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\nabla^2 G(x,y)=\left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right]\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

高斯-拉普拉斯算子LoG

- 上式定义了一个新的边缘检测算子，称为高斯-拉普拉斯算子（LoG, Laplacian of Gaussian），由Marr和Hildreth提出，故又称Marr-Hildreth（马尔-海尔德斯）边缘检测算子。
- σ是LOG算子的空间尺度，决定了算子中高斯滤波器对图像平滑的模糊程度，在小尺度的情况下，边缘将包含大量细节信息。如果增大尺度，细节就被抑制，小区域的边缘就会被丢弃。





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

- Because of its shape, the LoG operator is commonly called a ***Mexican hat***.

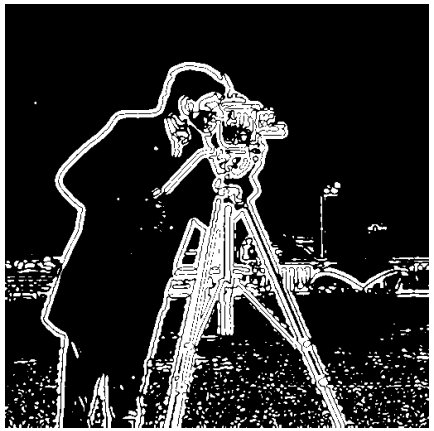
示例：利用高斯-拉普拉斯(LoG)算子检测图像边缘

```
#采用Laplace算子和LoG算子检测图像边缘
img = cv.imread('./imagedata/cameraman.tif',0)  #读入一幅灰度图像
edge_laplace1 = cv.Laplacian(img, ddepth=cv.CV_64F)  #OpenCV:Laplace算子
edge_laplace2 = filters.laplace(img)  #Scikit-image:Laplace算子
edge_log = ndimage.gaussian_laplace(np.float32(img), sigma=2)  #SciPy:LoG算子
#对edge_log的绝对值进行阈值分割
edge_log_bw = np.abs(edge_log)>0.1*np.max(np.abs(edge_log))
```

(1)cameraman原图

(2)Laplace边缘

(3)LoG边缘



(4)对LoG边缘绝对值二值化

9.3 Canny边缘检测算子

Canny 边缘检测算子

- **John F. Canny于1986年提出的一个多级边缘检测算法，它使用一系列不同尺寸的高斯滤波器对图像进行平滑滤波，然后从这些平滑后的图像中检测边缘，并将不同尺度的边缘融合起来形成最终的边缘图。Canny定义了三个主要准则：**
  - 低错误率。非边缘点被错标为边缘点的数量最少。
  - 定位精确。标为边缘的点应尽可能靠近真实边缘的中心位置。
  - 单像素边缘宽度。在每个边缘点位置仅给出单个像素。
- **通常只使用Canny算法中平滑滤波尺度因子可调的单一尺度版本，即便如此，Canny边缘检测算子仍然优于大多数的简单边缘检测算子。**

Canny, J. A Computational Approach to Edge Detection. IEEE Trans. on Pattern Analysis and Machine Intelligence, 1986,8(6):679-698.

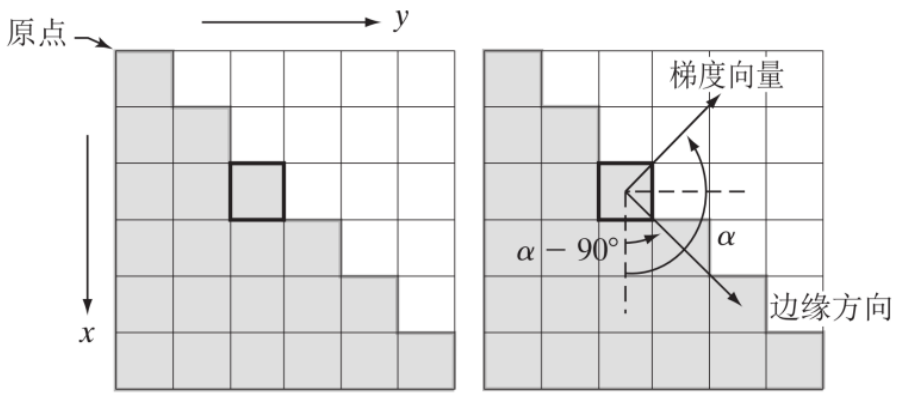
Canny 边缘检测算子计算步骤

❶首先用2D高斯滤波模板进行卷积以平滑图像。

❷利用微分算子，计算梯度的幅值和方向。

❸对梯度幅值进行非极大值抑制。即遍历梯度幅值图像，若某个像素的梯度幅值与其梯度方向上前后两个像素的梯度幅值相比不是最大，那么这个像素值置为0，即不是边缘。

❹使用双阈值算法检测和连接边缘。使用直方图计算两个阈值，凡是梯度的幅值大于高阈值的一定是边缘；凡是梯度的幅值小于低阈值的一定不是边缘。如果检测结果大于低阈值但又小于高阈值，那就要看这个像素的邻接像素中有没有超过高阈值的边缘像素，如果有，则该像素就是边缘，否则就不是边缘。



示例：采用Canny算子检测图像边缘

```
#采用Canny算子检测图像边缘
img = cv.imread('./imagedata/cameraman.tif', 0)   #读入一幅灰度图像
edge_canny1 = cv.Canny(img,threshold1=50,threshold2=200)  #OpenCV: Canny算子
edge_canny2 = feature.canny(img)   #Scikit-image: Canny算子，采用缺省参数
#标准差sigma=1，指定高低阈值(边缘幅值的百分位数)
edge_canny3 = feature.canny(img,sigma=1, low_threshold=0.05, high_threshold=0.95, use_quantiles=True)
#增大高斯平滑滤波器的标准差sigma=2
edge_canny4 = feature.canny(img,sigma=2)
```

(1)cameraman原图　(2)OpenCV Canny算子　(3)Scikit-image指定高/低阈值　(4)Scikit-image sigma=2

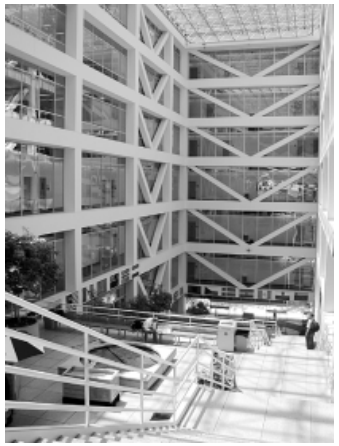
Canny边缘检测算子及其性能对比，为便于观察边缘反色显示

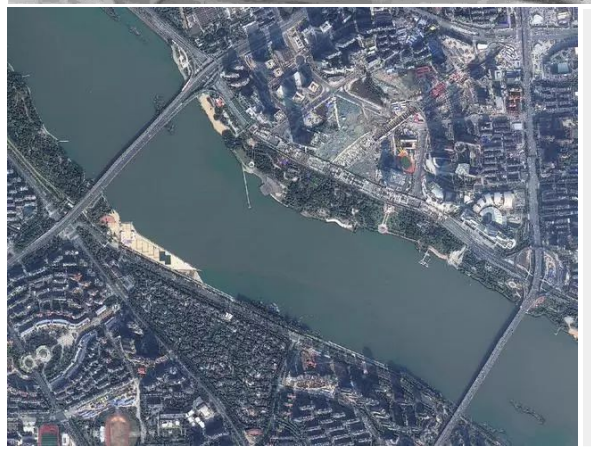
9.4 Hough变换

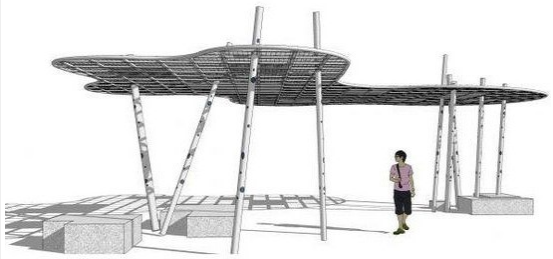
几何形状的检测

- 前几节讨论的边缘检测方法都是基于像素灰度值的梯度、二阶导数等图像局部性质，边缘图中通常很少存在完美的轮廓线，在边缘强度弱的地方出现间断，多数情况包含很多细小的、不连续的轮廓片段。同时，边缘图中存在许多无关结构，也可能丢掉我们感兴趣的重要结构。
- 图像中经常含有大量的人造物体，这些人造物体的轮廓或区域边界常以简单的几何形状出现，如直线、圆和椭圆等。
- 因此，我们常常对边缘图中的边缘点是否构成了直线、圆和椭圆等特定形状的几何曲线感兴趣。

人造物体中常见的简单几何形状







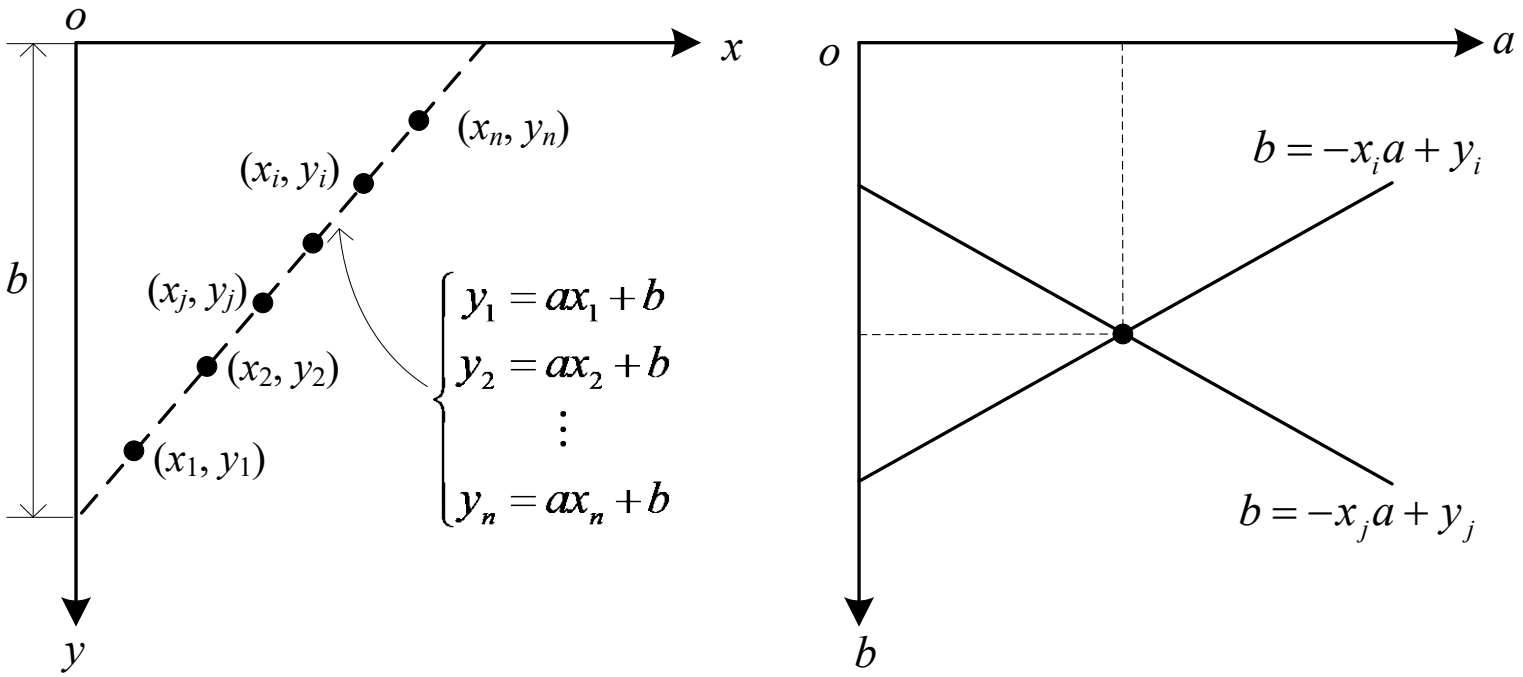
直线、圆、椭圆

Hough变换

- Hough变换（Hough Transform，霍夫变换）由Paul Hough于1959年提出，1962年被授予美国专利，经Richard Duda、Peter Hart和Ballard改进推广后得到广泛应用。
- 起初Hough变换主要用来检测图像中的直线，后来逐渐扩展到识别圆、椭圆等几何曲线。
- Hough变换是一种基于“投票表决”的几何曲线形状识别技术。Hough变换根据局部度量来计算全局描述参数，因而对于区域边界被噪声干扰或被其他目标遮盖而引起的边界间断情况，具有很好的容错性和鲁棒性。

Hough变换的直线检测

- Hough变换的直线检测原理，是利用图像空间和参数空间的“点-线”对偶性，把图像空间中检测“共线点”问题，转换为在参数空间中检测“共点线”问题。



(1)图像空间中“共线点”所在直线的斜截式方程表示　(2)对应参数空间中的“共点线”的形成

Hough Transform

- **Consider a point (x_i, y_i) and the general equation of a straight line in slope-intercept form:**

$$y_i = a\, x_i + b$$

  - Q: How many lines may pass through (x_i, y_i) ?

- **Re-writing the equation in "*ab*-plane"—parameter space:**

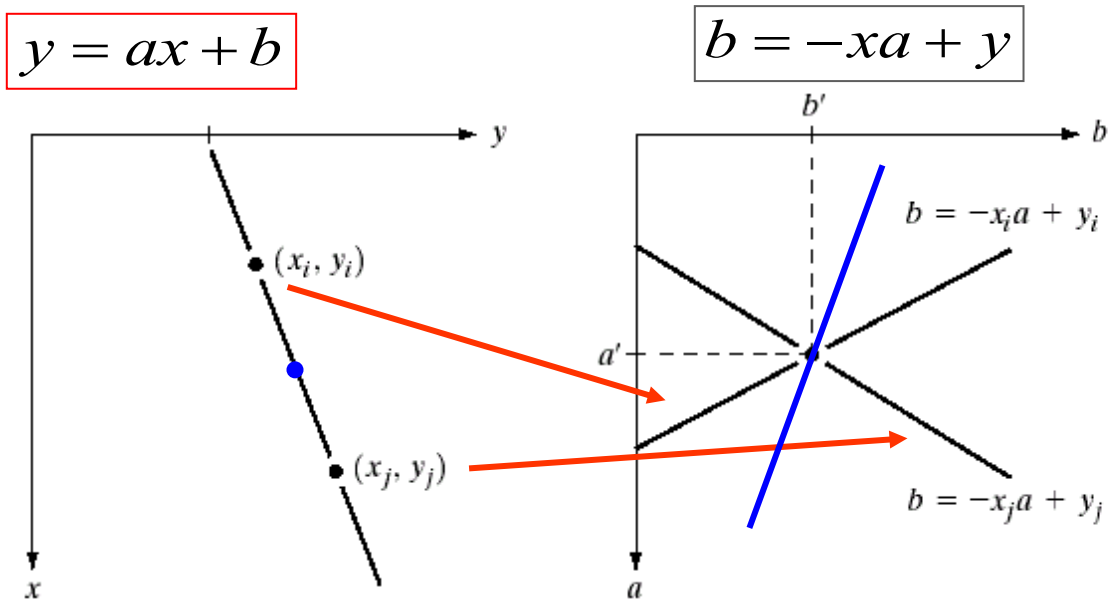
$$b = -\, x_i\, a + y_i$$

  - Q: How many lines do we get for a fixed (x_i, y_i) in "*ab*-plane"?

- **Now given another point (x_j, y_j),**
  - How to find the parameter (a', b') which defines the line that contains both points?

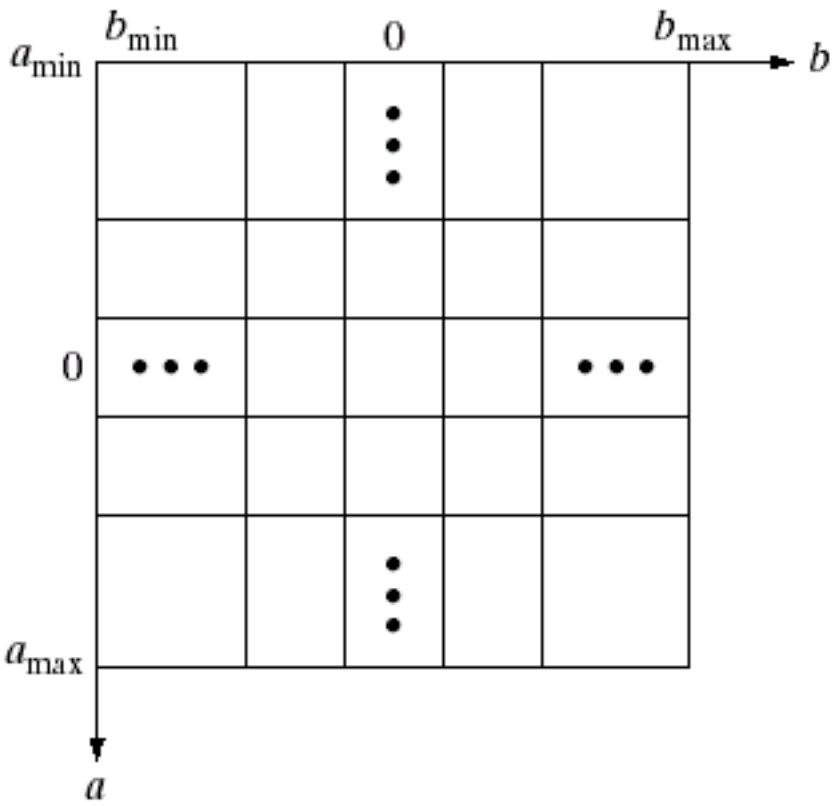
Hough Transform (cont’ d)

- All points on the line passing through (x_i, y_i) and (x_j, y_j), have lines in parameter space that intersect at (a', b').



Hough Transform (con’t)

- **Polling all edge points for lines**



◆为了统计参数空间中相交于某一点的直线（或曲线）的数量，Hough变换把参数空间离散化为网格，如图所示。

◆设想把每个网格看成一个“票箱”，称为累加器单元，所有网格对应的累加器单元便形成了一个累加器数组。对于图像中的每一个边缘点，都可以在参数空间中画出一条直线（或曲线），当该直线（或曲线）经过某一网格时，对应的累加器单元计数值加1，相当于向该网格“票箱”中投1票。

◆“点-线”变换结束后，每个累加器单元中的计数值，等于经过与该累加器单元相对应的参数空间网格的直线（或曲线）数量，也就是图像中位于同一直线上的边缘点的个数，而网格位置坐标便是这条直线的参数值。

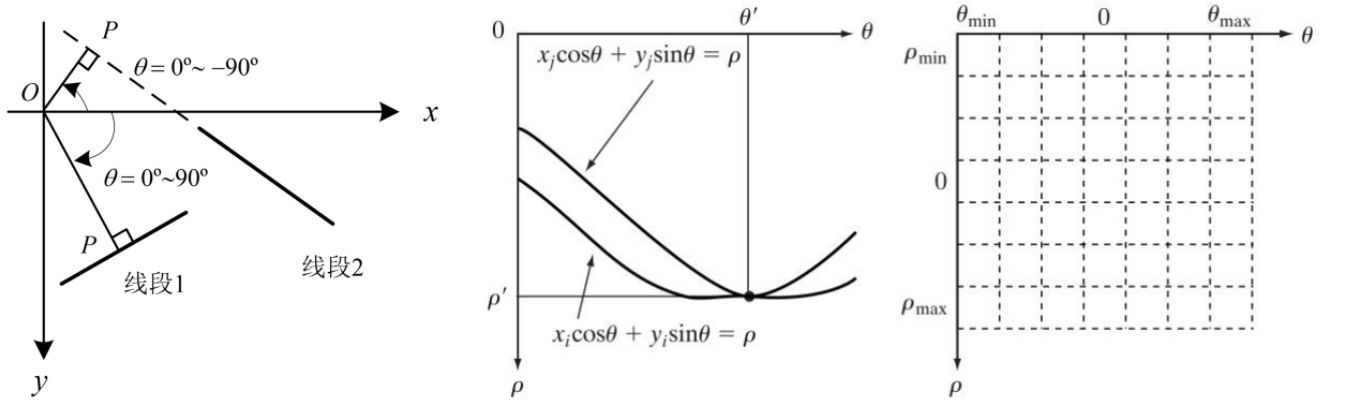
Hough Transform (cont’ d)

- 使用斜截式$y=ax+b$直线方程带来的问题是，当直线与x轴接近垂直时，直线的斜率a接近无穷大，参数空间a-b平面无界，无法离散为有限多个网格。
- 为获取有界的参数空间，Duda采用了称为Hessian标准形(Hessian normal form）的法线式直线方程。这样，图像平面上的边缘点(x_i,y_i)，被映射为ρ-θ平面上的一条正弦曲线：

$$x\cos\theta + y\sin\theta = \rho$$

$$-90° \le \theta < 90°, \quad -D \le \rho \le D$$

ρ是图像平面坐标原点O到该直线的垂直线的代数距离OP，θ为OP与x轴正方向所成的夹角，顺时针方向取正值，范围为-90°$\le\theta<$90°。



(1)x-y平面法线式直线方程　(2) ρ-θ平面中对应的正弦曲线　(3)将ρ-θ平面离散化为有限多个网格

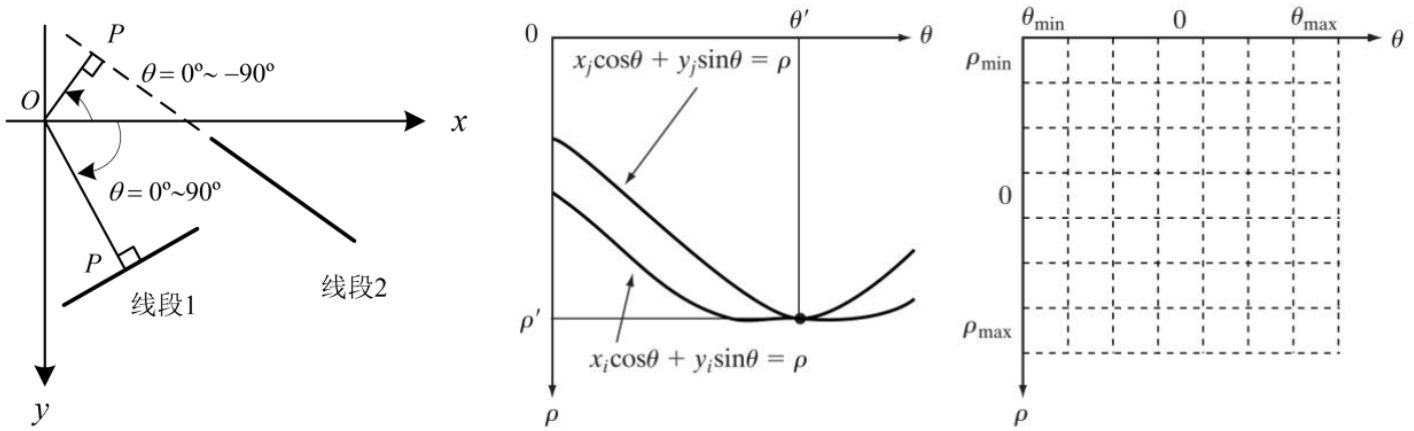
图9.10 ρ-θ法线式直线方程的Hough变换原理示意

Hough Transform (cont’ d)

- 对于一幅图像，尺寸已知，图像平面中直线的参数ρ小于图像的对角线长度D。因此，这意味着我们感兴趣的那些直线的(ρ,θ)值形成了ρ-θ平面的一个有界子集，即：

$$-90° \le \theta < 90°, \quad -D \le \rho \le D$$

- 这样，就可以把ρ-θ平面离散化为有限个网格。设想把每个网格看成一个“票箱”，称为累加器单元，所有网格对应的累加器单元形成累加器数组。



(1)x-y平面法线式直线方程　(2) ρ-θ平面中对应的正弦曲线　(3)将ρ-θ平面离散化为有限多个网格

图9.10 ρ-θ法线式直线方程的Hough变换原理示意

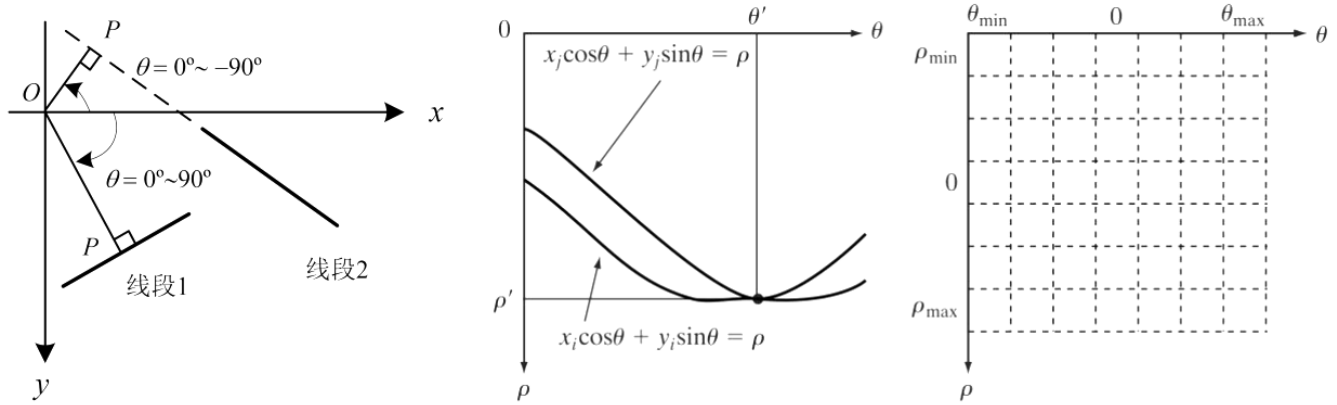
Hough Transform (cont’ d)

- **对于图像中的每一个边缘点(x_i,y_i)，有：**

$$\rho = x_i \cos\theta + y_i \sin\theta$$

- **令在-90≤ ρ <90范围内变化，按上式计算出相应的值，就可以在参数空间ρ-θ平面上画出一条正弦曲线，该正弦曲线经过的所有网格都增加一票，交点(ρ', θ')对应于x-y平面上过点(x_i,y_i)和(x_j,y_j)的直线参数。**

  - 如果有 N 个边缘点共线，那么就会有 N 张票“投到”该直线对应的网格累加器单元(ρ', θ')中，其累计值为N。



(1)x-y平面法线式直线方程　(2) ρ-θ平面中对应的正弦曲线　(3)将ρ-θ平面离散化为有限多个网格

图9.10 ρ-θ法线式直线方程的Hough变换原理示意

Hough变换直线检测的基本步骤

- 创建累加器数组
- 点线映射
- 确定累加器数组的极大值
- 确定直线的端点及间断连接

创建累加器数组

- 为了计算Hough变换，必须先为ρ和θ选择合适的步长，把连续的参数空间离散为有限个网格（类似图像数字化过程的采样），并为每个网格沿ρ坐标轴和θ坐标轴的位置顺序分配两个自然整数序号，譬如(s, t)。
- 数组H用来记录参数空间中每个网格位置与正弦曲线相交的次数，因此H被称作累加器数组，又称Hough变换矩阵。
- 注意：ρ和θ离散步长大小影响直线的定位精度，合适的网格尺寸很难选择。太粗糙的网格导致某个“票箱”的投票值太大而无效，因为许多不同的直线对应于同一个“票箱”。太精细的网格导致直线可能找不到，因为边缘点并不是准确地共线，因此所产生的投票会被记录到不同的“票箱”里，而没有一个“票箱”能累计到大的投票数。

点线映射

- 将累加器数组H各元素初始化为0。对于边缘图像 f 中的每一个边缘点(x_i,y_i)，令θ等于其每一个允许的离散值θ_t，按下式计算对应的每一个ρ值：

$$\rho = x_i \cos\theta_t + y_i \sin\theta_t$$

- 查找与其最接近的离散值ρ_s，得到相应的索引下标s，然后令H(s,t) = H(s,t) +1。
- 上述“点-线”映射过程完成后，累加器数组的每个元素值H(s,t)就给出了边缘图像$f(x,y)$中位于直线$\rho_s = x\cos\theta_t + y\sin\theta_t$上的边缘点数量。

确定累加器数组的极大值（1）

- 从累加器数组H中找出一组有意义的极大值，是Hough变换的关键。
- 我们知道，即使图像中的线段在几何上是直的，但是映射到离散参数空间中的相关曲线的交点并不是精确地向累加器数组中同一个单元“投票”，而是在一个小范围内分布，这主要是因累加器数组中的离散坐标引起的误差。
- 因此，简单地遍历数组并返回前K个极大值是不充分的。确定累加器数组中极大值的方法很多，下面介绍一种采用阈值并结合**邻域最大值抑制**的方法。

确定累加器数组的极大值（2）

- **假设要从累加器数组H中确定K个极大值。首先选择一个阈值T，其大小由期望在图像中找到的线段含有的最少边缘点数来定。再选择进行最大值抑制的邻域尺寸，一般采用$m \times n$矩形邻域，m、n为奇数，大小依据在图像中期望找到的线段之间的间距而定，线段间距小，相应m、n也应小。然后按下述步骤确定所有K个极大值：**

(1)找出累加器数组H中的最大值H_{max}，如果H_{max}大于或等于阈值T，记录H_{max}所在元素的位置(p,q)。若同时找到多个相等的最大值，仅记录其中一个。

(2)将记录的最大值所在元素$H(p,q)$及其$m \times n$邻域元素都设为零。

(3)重复步骤(1)和(2)，直到找到K个极大值，或数组H中的最大值小于指定的阈值T时为止。

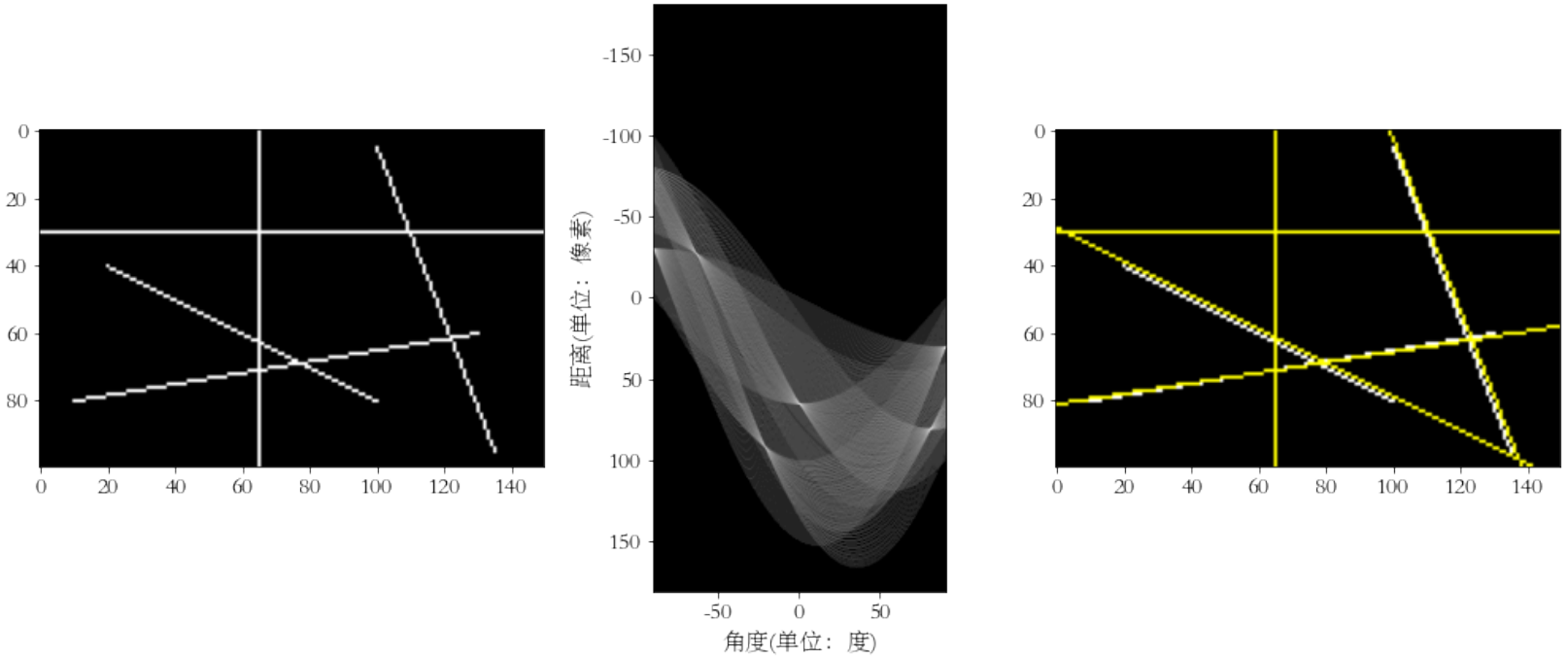
确定直线的端点及间断连接

- **一旦找到累加器数组H中K个极大值的位置，就获得了图像中对应K个直线的参数ρ和θ，但诸如直线是否存在间断、线段的端点等信息还有待确定。下面给出一种获取这些信息的方法：**
  - 利用点线映射方法，对于每个极大值点(ρ_k,θ_k)，寻找满足该直线方程$\rho_k = x\cos\theta_k + y\sin\theta_k$的所有边缘点$(x,y)$，并对这些边缘点排序。
  - 计算排序后的相邻边缘点之间的距离。如果所有邻点距离小于或等于给定阈值，则该直线可视为无间断，边缘点序列的头、尾点就是该直线的两个端点；如果存在邻点距离大于给定阈值，那么，该直线存在间断，找到间断位置，就可以确定构成该直线的每一片段的两个端点。

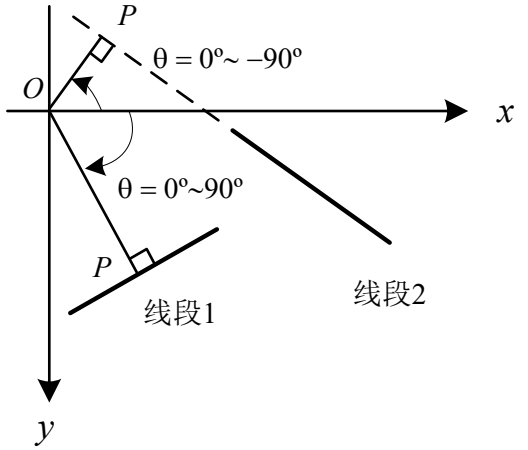
Hough变换的编程实现

- Scikit-image提供了实现标准Hough变换的两个函数：transform.hough_line()完成参数空间的离散化、创建累加器数组（Hough空间）和点线映射计算；transform.hough_line_peaks()从累加器数组中找出一组有意义的极大值（峰值），返回对应的直线参数ρ、θ和累加值。
- 另外Scikit-image还提供了概率Hough变换函数transform.probabilistic_hough_line()，能够快速检测边缘图像中的直线段。
- OpenCV提供了标准Hough变换函数cv.HoughLines()和概率Hough变换函数cv.HoughLinesP()。

示例：用Hough变换检测直线



边缘点数：[150 100 88 85 83]
角度theta：[90. 0. 81. -21.5 -63.5]
距离 rho：[31. 66. 81. 92. -27.]



示例：用Hough变换检测直线

```
# Scikit-image: 标准Hough变换直线检测示例
#构建一幅二值图像用于测试
img = np.zeros((100, 150), dtype=np.uint8)
img[30, :] = 255; img[:, 65] = 255
rr, cc = draw.line(60, 130, 80, 10); img[rr, cc] = 255
rr, cc = draw.line(r0=40, c0=20, r1=80, c1=100); img[rr, cc] = 255
rr, cc = draw.line(r0=5, c0=100, r1=95, c1=135 ); img[rr, cc] = 255
#角度间隔为0.5度Set a precision of 0.5 degree.
angles_axis = np.deg2rad(np.arange(-90, 91, 0.5))
#标准Hough变换
hspace, theta, rho_dist = transform.hough_line(img,theta=angles_axis)
#获取Hough空间hspace中的峰值及对应直线参数
accum,angles,dists= transform.hough_line_peaks(hspace, theta,rho_dist)
#显示检测到的直线参数
print('边缘点数:',accum);  print('角度theta:',np.rad2deg(angles));
print('距离  rho:',np.round(dists))
```

示例：用Hough变换检测直线

```
#将Hough变换检测得到的直线以黄色叠加到图像中
img_result = color.gray2rgb(img)   #将二值测试图像转换为RGB颜色通道图像
xp = np.arange(0,img.shape[1])   #设定检测到的直线x轴坐标范围(数组列下标)
#根据参数rho,theta计算每条直线对应的y轴坐标值(数组行下标)
for i in range(angles.shape[0]):
    if angles[i] == 0: #垂直线
        #确定在图像范围内的直线端点坐标
        x1 = np.int32(dists[i]);  y1 = 0
        x2 =  np.int32(dists[i]); y2 = img.shape[0]-1
    else:
        yp =np.int32((dists[i] - xp * np.cos(angles[i])) / np.sin(angles[i]))
        #确定在图像范围内的直线端点坐标
        yidx = np.logical_and(yp>=0, yp<img.shape[0])
        x1 = xp[yidx][0];  y1 = yp[yidx][0]
        x2 = xp[yidx][-1]; y2 = yp[yidx][-1]
    img_result = cv.line(img_result,(x1,y1),(x2,y2),(255,255,0),1)  #画线
```

Hough变换的圆检测

- 图像平面上的圆，需要三个参数来表示，即：

$$(x-a)^2+(y-b)^2=r^2$$

  - 式中，(a,b)为圆心位置的坐标，而r为圆的半径，这些参数构成了一个a-b-r三维参数空间。

- 图像中的一个边缘点(x_i,y_i)，通过该点的圆应满足方程：

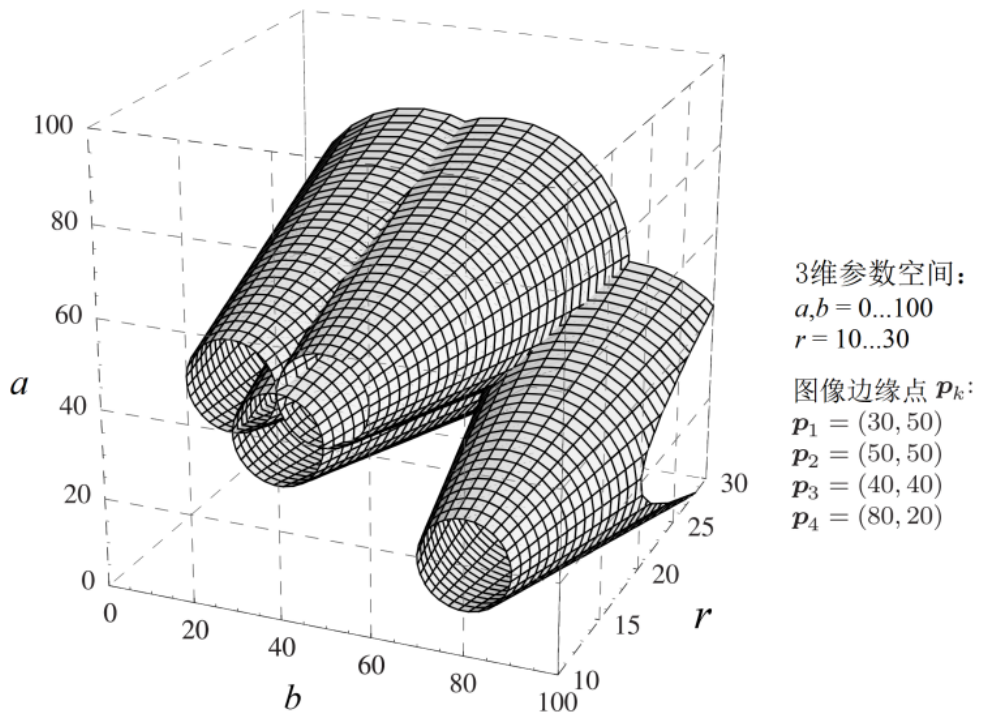
$$(x_i-a)^2+(y_i-b)^2=r^2$$

将其改写为：

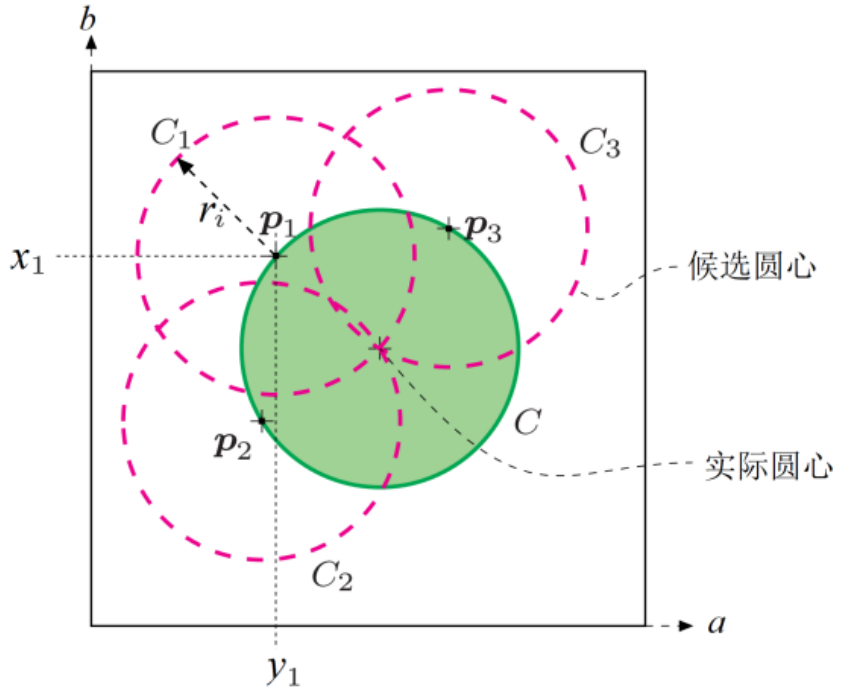
$$(a-x_i)^2+(b-y_i)^2=r^2$$

- 上式表示了在三维参数空间中以(x_i,y_i)圆心、半径为r的一系列圆。因此，圆的Hough变换，图像中的一个边缘点，按上式被映射为a-b-r三维参数空间中一个圆锥面。

圆的Hough变换



圆的a-b-r三维参数空间。每一个图像边缘点p_k对应的圆锥面所经过的三维累加数组单元值加1

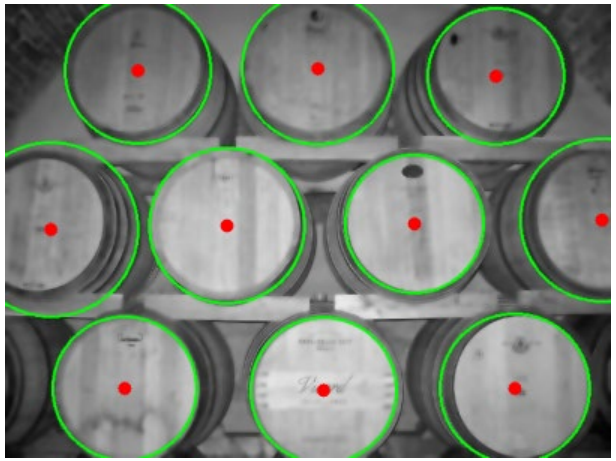


图中描述了在给定半径$r = r_i$时三维累加数组的一个剖面。经过给定边缘点p_1=(x_1,y_1)的所有可能的圆的圆心位置，形成了以p_1为圆心、半径为r_i的圆C_1；同样，经过边缘点p_2、p_3的所有可能的圆的圆心位置，形成了圆C_2和C_3。

示例：检测图像中的圆

- 图(1)是一幅含有10个橡木桶的灰度图像，它们的半径约在50~200像素之间。
- 图(2)为采用OpenCV函数cv.HoughCircles()的圆检测结果，将检测到的圆及圆心叠加到原图像上显示。

(1)橡木桶图像



(2)将检测的圆及圆心叠加到原图像

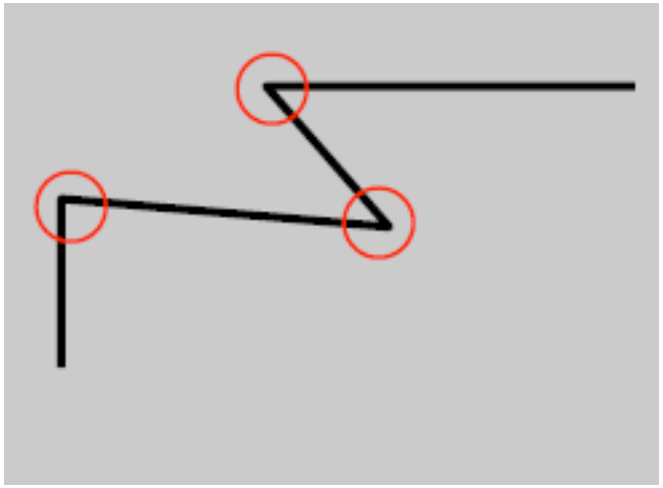
示例：检测图像中的圆

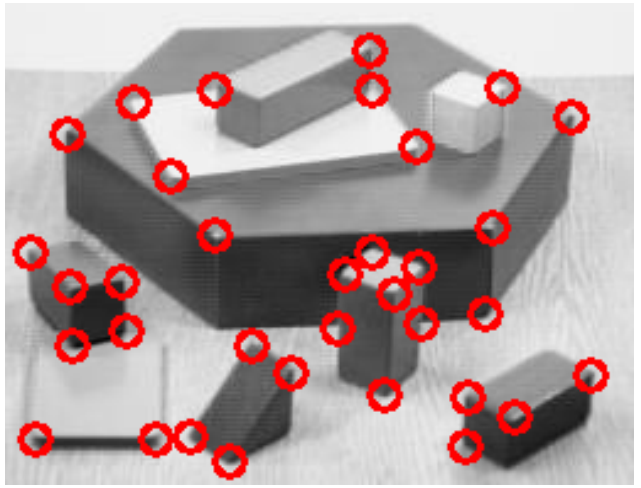
```
# OpenCV: Hough变换圆检测示例
img = cv.imread('./imagedata/oak_barrels.png',0)  #读入一幅灰度图像
img2 = cv.bilateralFilter(img,9,75,75)  #对图像进行双边滤波
#Hough圆变换，如果maxRadius=0, 则使用图像的最大尺寸
circles = cv.HoughCircles(img2,cv.HOUGH_GRADIENT,1,30, \
                          param1=100,param2=90,minRadius=20,maxRadius=0)
#将灰度图像转换为RGB颜色通道图像
imgrgb = cv.cvtColor(img2,cv.COLOR_GRAY2RGB)
#用绿色绘出检测到的每个圆环,红色画圆心
circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    imgrgb = cv.circle(imgrgb,(i[0],i[1]),i[2],(0,255,0),2)  #画圆环
    imgrgb = cv.circle(imgrgb,(i[0],i[1]),7,(255,0,0),-1)     #画圆心
#显示结果（略）
```

9.5 角点检测

角点检测

- 角点检测（Corner detection）是图像处理和计算机视觉中常用的一种算子。现实世界中，角点对应于物体的拐角、道路的十字路口、丁字路口等，是物体的显著结构要素。
- 角点在人类视觉与机器视觉中都有着重要的作用，它不但可为人类视觉提示边缘信息，而且是机器视觉中的少量“鲁棒”特征之一。
- 因此，角点检测广泛应用于目标跟踪、目标识别、图像配准与匹配、三维重建、摄像机标定等计算机视觉领域。





角点检测

- **所谓“鲁棒”特征，主要是指那些在三维场景中非偶然出现的，且在大范围视角和光照条件下相对稳定并能准确定位的特征。**
- **角点在保留图像图形重要特征的同时，可有效地减少信息的数据量，使其信息的含量很高，有效地提高了计算速度，有利于图像的可靠匹配，使得实时处理成为可能。对于同一场景，即使视角发生变化，通常具备稳定性质的特征。**
- **Harris角点检测器的基本思想为：**
  - 角点存在于图像梯度在多个方向上同时取得较大值的位置；
  - 只有一个方向梯度较大的边缘位置不是角点，并且因为角点在任意方向上都存在，因此检测器应该是各向同性的。

Harris角点检测器计算流程

- **对每个像素，计算其梯度分量g_x、g_y。**
- **对每个像素，计算三个值A、B和C。**

$$\begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

$$A = g_x^2, \quad B = g_y^2, \quad C = g_x \cdot g_y$$

- **然后对上述计算结果A、B和C进行高斯平滑滤波。**

$$\overline{A} = A * h_{G,\sigma}, \quad \overline{B} = B * h_{G,\sigma}, \quad \overline{C} = C * h_{G,\sigma}$$

- **构造局部结构矩阵M，计算每个像素的角点响应函数$R(M)$作为“角点强度”的度量。**

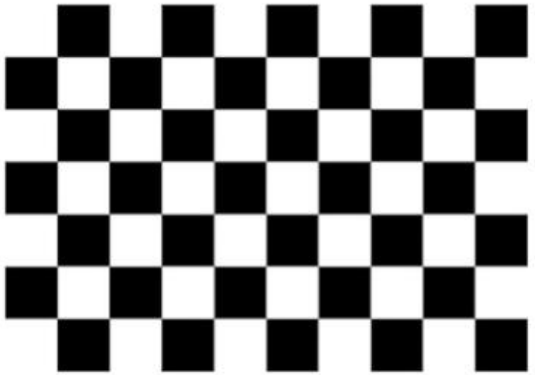
$$M = \begin{bmatrix} \overline{A} & \overline{C} \\ \overline{C} & \overline{B} \end{bmatrix}$$

$$\begin{aligned} R(M) &= \det(M) - \alpha \cdot (\operatorname{trace}(M))^2 \\ &= \left(\overline{A}\,\overline{B} - \overline{C}^2\right) - \alpha \cdot \left(\overline{A} + \overline{B}\right)^2 \end{aligned}$$

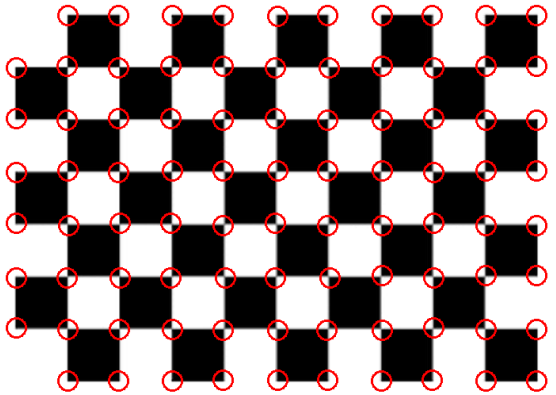
  - 式中，参数α决定了角点检测器的灵敏度，通常在0.04~0.06之间取值，α值越大，角点检测器越不敏感，检测到的角点数就越小。
- **对角点响应函数$R(M)$取阈值，并进行非最大值抑制，得到检测到的角点坐标。**

示例：角点检测

```
#角点检测示例Corner detection
img = io.imread('./imagedata/chessboard.png')  #读入一幅灰度图像
harris_res = feature.corner_harris(img)  #计算Harris角点响应图像
#从Harris角点响应图像获取峰值点及其行列下标，即为角点
coords = feature.corner_peaks(harris_res, min_distance=5, threshold_rel=0.01)
#以检测到角点为圆心，用红色圆环标出
img_corners = util.img_as_ubyte(color.gray2rgb(img))  #将灰度图像转换为RGB图像
for corner in coords:
    #画红色圆环
    img_corners = cv.circle(img_corners,(corner[1],corner[0]),12,(255,0,0),thickness=2)
```



⑴棋盘格图像



⑵采用Harris方法得到的角点（圆心位置）

Q&A