FDU 数字图像处理 6. 形态学操作

本文参考以下教材:

- Digital Image Processing (4th Edition, R. Gonzalez, R. Woods) Chapter 9
- 数字图像处理 (第四版, R. Gonzalez, R. Woods) 第 9 章

欢迎批评指正!

6.1 Introduction

形态学运算 (morphological operations) 是用集合来定义的.
在图像处理中，我们使用两类像素集合的形态学: **目标元素** (objects) 和**结构元** (structuring elements, SE)
目标通常被定义为前景像素集合，结构元可以按照前景像素和背景像素来规定. |
(处理实际的二值图像时， 我们说目标是前景像素， 而所有其他像素是背景像素)

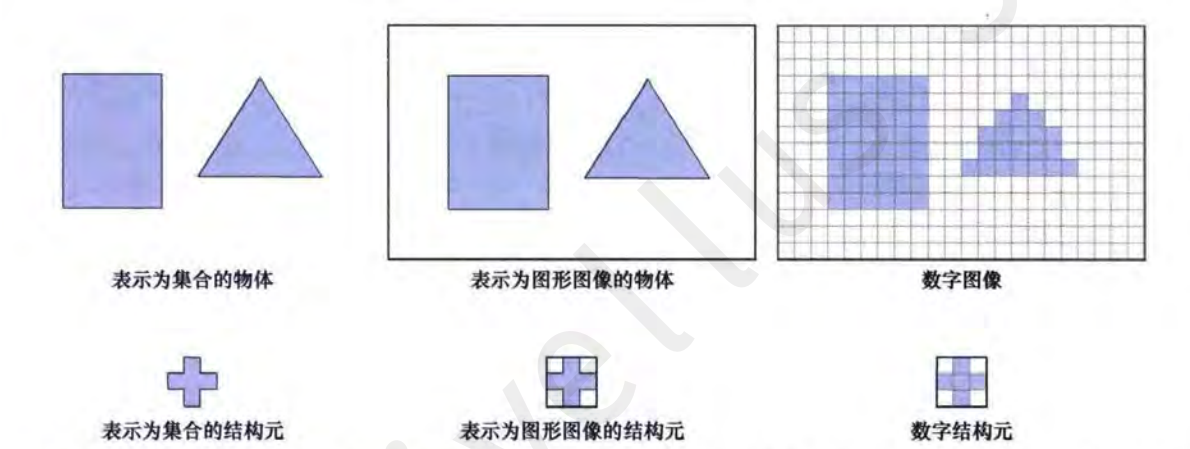


图 9.1 上一行。左侧：表示为图形集合的目标；中间：背景中嵌入目标后形成的图形图像；右侧：目标和背景经数字化后形成的一幅数字图像（注意网格）。第二行：表示为集合、图形图像和数字的结构元示例

数字图像的表示方式与数字结构元的表示方式有一个重要的区别.
目标周围存在一个由背景像素组成的边框，而结构元中则不存在这样一个边框.
结构元的用法类似于空间卷积核，并且上述图像边框类似于填充.
在形态学中，这些运算是不同的，但填充运算和滑动运算与卷积中的相同.

形态学中广泛使用集合的反射和平移概念与结构元有关.
集合 (结构元) B 相对于其原点的**反射** (reflection) 定义如下:

$$B_{\text{reflect}} := \{(-x, -y) | (x, y) \in B\}$$

它将整个结构元相对于其原点旋转 180°，包括背景和不关心元素 (记为 $\times$)
如图所示: (图中的黑点表示结构元的原点)

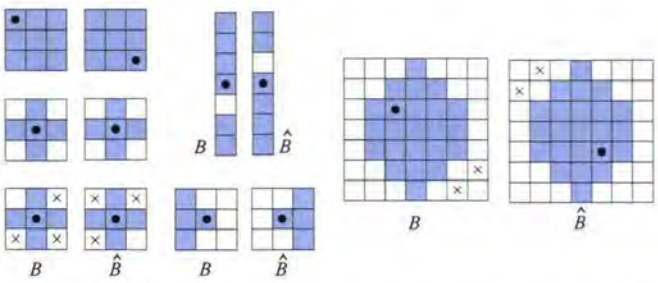


图 9.2　结构元及其相对于原点的反射（×是“不关心”元素，黑点表示原点）；反射是指 SE 相对于原点旋转 180°

集合 (结构元) B 相对于点 $z = (z_1, z_2)$ 的**平移** (translation) 定义如下:

$$B_{\text{translate}(z)} := \{(x + z_1, y + z_2) : (x, y) \in B\}$$

6.2 腐蚀与膨胀

6.2.1 腐蚀

考虑一幅简单的二值图像 I，记其目标集合为 A (显示为阴影)
设结构元的大小为 3×3，元素均为 1 (前景像素) (背景像素的值为 0，表示为白色)
我们进行如下形态学运算:

- ① 生成一幅大小与原二值图像 I 相同的新图像，最初只包含背景值.
- ② 将结构元 B 在原二值图像 I 上滑动.
  在每个位置上，如果结构元 B 完全包含于目标像素集 A，
  则将 B 的原点位置标记为新图像的一个前景像素;
  否则将其标记为一个背景像素.

形态学运算结果如下:

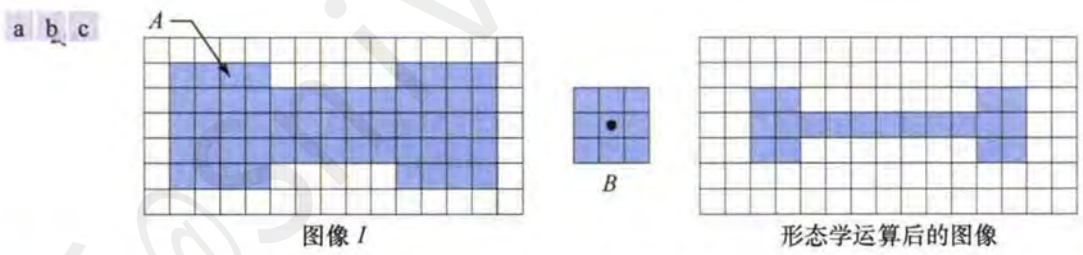


图 9.3　(a)包含一个目标（集合）的二值图像 A；(b)一个结构元 B；(c)形态学运算后的图像

上述形态学运算称为结构元 B 对目标集合 (前景像素集) A 的**腐蚀** (erosion):

$$A \ominus B := \{(x, y) : (x, y) \in A \text{ such that } B_{\text{translate}(x,y)} \subseteq A\}$$

它是一种收缩、细化的运算.

我们用 $A \ominus B$ 来表示只使用前景像素的腐蚀，
用 $I \ominus B$ 来表示使用前景和背景像素的腐蚀.
此时结构元可由前景和背景像素混合而成，甚至可能包含不关心元素，
但 $\ominus$ 的语义可能需要修正，我们不过多讨论.

腐蚀运算还有以下等价定义:

$$A \ominus B := \{(x,y) \in \mathbb{Z}^2 : B_{\text{translate}(x,y)} \cap A^c = \emptyset\}$$

$$A \ominus B := \{(x,y) \in \mathbb{Z}^2 : (x + x_0, y + y_0) \in A \text{ for all } (x_0, y_0) \in B\}$$

$$A \ominus B := \bigcap_{(x,y)\in B} A_{\text{translate}(-x,-y)}$$

图 9.4 显示了腐蚀的一个例子。集合 A（显示为阴影）的元素是图像 I 的前景像素，背景照例显示为白色。图 9.4(c)中虚线边界内的实线边界是 B 的原点的位移界限，超过这一界限移动时会使得结构元的某些元素不再完全包含于 A。于是，这个边界上和这个边界内的点的轨迹（B 的原点位置）就构成了 B 对 A 的腐蚀的前景元素。图 9.4(c)中以阴影方式显示了这一腐蚀的结果，背景显示为白色。腐蚀是满足式(9.3)或式(9.5)的 z 值集合。在图 9.4(c)和(e)中，集合 A 的边界显示为虚线，它仅供参考，而不是腐蚀的一部分。图 9.4(d)显示了一个加长的结构元，图 9.4(e)显示了这个结构元对集合 A 的腐蚀。注意，原目标已被腐蚀为一条直线。如看到的那样，腐蚀结果受结构元形状的控制。在两种情况下，假设图像均被填充，以容纳 B 的所有位移，并且结果被裁剪到与原图像同样大小，就像第 3 章中对图像进行空间卷积运算时所做的那样。

式(9.3)和式(9.5)并不是腐蚀的唯一定义（另两个等效的定义见习题 9.12 和习题 9.13），但在把结构元 B 视为在集合上滑动的空间核时，上述公式的优点是更加直观。

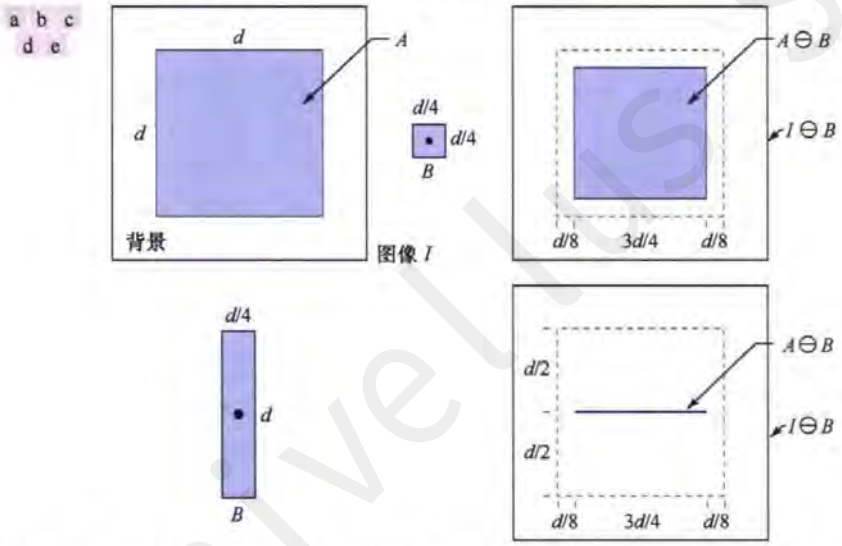


图 9.4 (a)由集合 A（目标）和背景组成的图像 I；(b)方形 SE 和 B（黑点是原点）；(c)B 对 A 的腐蚀（在结果图像中显示为阴影）；(d)加长的 SE；(e)B 对 A 的腐蚀（腐蚀结果是一条直线）。(c)和(e)中的虚线边框是集合 A 的边界，仅供显示参考

例 9.1　使用腐蚀删除图像中的某些部分。

图 9.5(a)是一幅二值图像，它描述的是一个简单的焊线模板。如前所述，我们将二值图像的前景像素显示为白色，将背景像素显示为黑色。假设我们希望删除图 9.5(a)中的中心区域到边界衬垫的连线。使用大小为 11×11、元素都为 1 的方形结构元腐蚀图像（即腐蚀图像中的前景像素）后，结果如图 9.5(b)所示，我们看到结果图像中大多数为 1 的连线都删除了。中间两条垂线被细化而未被完全删除的原因是，它们的宽度大于 11 像素。将 SE 的大小改为 15×15 后，再次腐蚀原图像，得到的结果如图 9.5(c)所示，我们发现所有连线都已被删除。另一种可行的方法是，使用大小为 11×11 或更小的 SE 再次对图 9.5(b)进行腐蚀。增大结构元的尺寸甚至会删除更大的元件。例如，使用大小为 45×45 的结构元腐蚀原图像后，删除了连线和边界衬垫，如图 9.5(d)所示。

由这个例子可以看出，腐蚀缩小或细化了二值图像中的目标。事实上，我们可以将腐蚀视为形态学滤波运算，这一运算将滤除图像中小于结构元的图像细节。在图 9.5 中，腐蚀执行“线滤波”的功能。我们将在 9.4 节和 9.8 节中回到形态学滤波的概念。

a b
c d

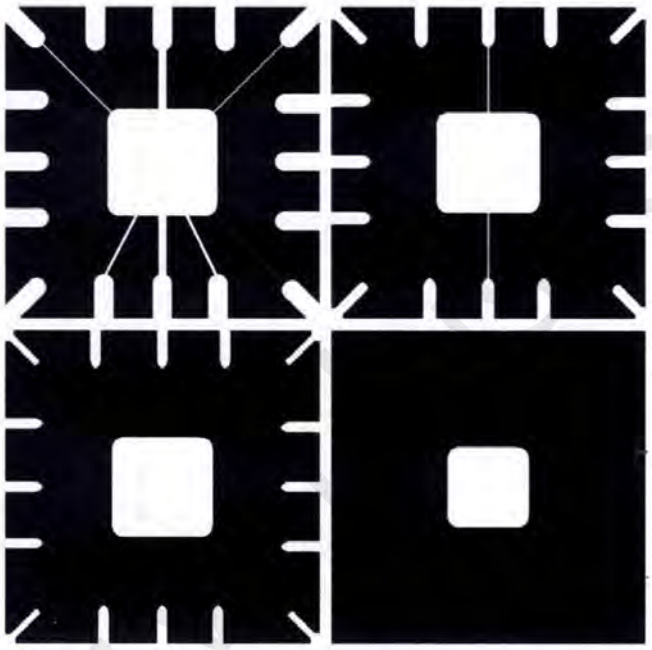


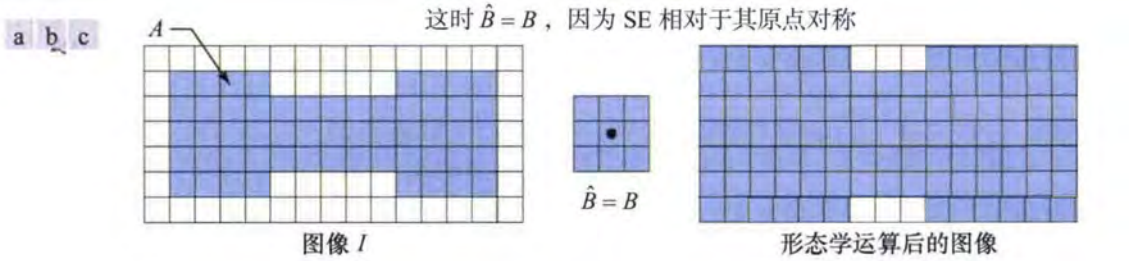
图 9.5　使用腐蚀删除图像中的某些部分：(a)大小为 486×486 的焊线模板的二值图像，前景像素显示为白色；(b) ~ (d)使用所有值为 1、大小分别为 11×11、15×15 和 45×45 的结构元腐蚀图像后的结果 ■

6.2.2 膨胀

考虑一幅简单的二值图像 I，记其目标集合为 A (显示为阴影)
设结构元的大小为 3×3，元素均为 1 (前景像素) (背景像素的值为 0，表示为白色)
我们进行如下形态学运算:

- ① 生成一幅大小与原二值图像 I 相同的新图像，最初只包含背景值.
- ② 首先将结构元 B 关于其原点翻转 (得到 B_{reflect})，然后在原二值图像 I 上滑动.
  在每个位置上，如果结构元 B 的反射 B_{reflect} 至少有一个前景像素与目标像素集 A 重叠，
  则将 B 的原点位置标记为新图像的一个前景像素;
  否则将其标记为一个背景像素.

形态学运算结果如下:



上述形态学运算称为结构元 B 对目标集合 (前景像素集) A 的**膨胀** (dilation):

$$A \oplus B := \{(x,y) \in \mathbb{Z}^2 : [(B_{\text{reflect}})_{\text{translate}(x,y)} \cap A] \subseteq A\}$$

它是一种增长、粗化的运算.
与腐蚀一样，将结构元 B 视为卷积核时，上述定义将更加直观.
但要记住，膨胀是以集合运算为基础的，因此是非线性运算; 而卷积是乘积求和，是线性运算.

其等效定义为:

$$A \oplus B := \{(x,y) \in \mathbb{Z}^2 : (x,y) = (a_1,a_2) + (b_1,b_2) \text{ for some } (a_1,a_2) \in A \text{ and } (b_1,b_2) \in B\}$$

$$A \oplus B := \bigcup_{(x,y)\in B} A_{\text{translate}(x,y)}$$

腐蚀是一种收缩或细化运算，而膨胀会“增长”或“粗化”二值图像中的目标。粗化的方式和宽度受所用结构元的大小和形状控制。图 9.6(a)中的目标与图 9.4 中的相同（为容纳膨胀结果，背景区域更大），图 9.6(b)是一个结构元（这时 $\hat{B}=B$，因为 SE 相对于其原点对称）。图 9.6(c)中的虚线是原目标的边界，实线是一个界限，$\hat{B}$ 的原点的位移 z 超出这一界限时，会使得 $\hat{B}$ 和 A 的交集为空。因此，位于这个边界上或这个边界内的所有点，就构成了 B 对 A 的膨胀。图 9.6(d)是一个设计的结构元，这个结构元对垂直方向的膨胀多于对水平方向的膨胀；图 9.6(e)是使用这个结构元膨胀后的结果。

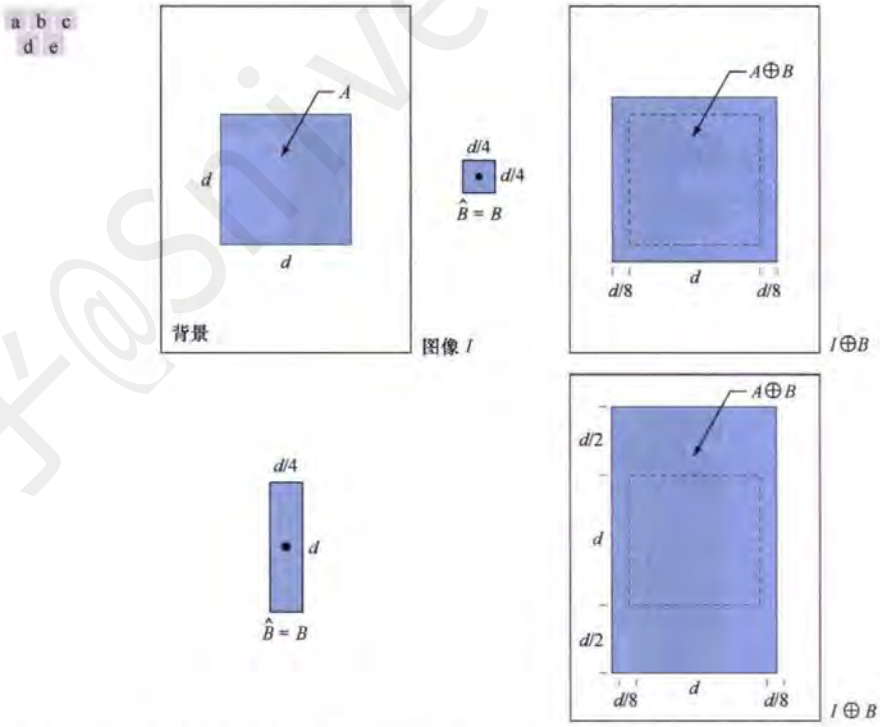


图 9.6 (a)由集合 A（目标）和背景组成的图像 I；(b)方形 SE（黑点表示原点）；(c)B 对 A 的膨胀（显示为阴影）；(d)拉长的 SE；(e)这个结构元对 A 的膨胀。(c)和(e)中的虚线是 A 的边界，仅作为显示参考

例 9.2　使用膨胀修复图像中的断裂字符。

一种最简单的膨胀应用是连接裂缝。图 9.7(a)是与图 4.48 相同的带有断裂字符的图像，为了连接断裂的字符，当时对这幅图像进行了低通滤波。已知最大长度的断裂是 2 个像素。图 9.7(b)是能够修复这些裂缝的结构元。如前所述，在处理图像时，我们用白色（1）表示前景，而用黑色（0）表示背景。图 9.7(c)是使用这个结构元对原图像膨胀后的结果，结果中裂缝已被连上。与图 4.48 中用来连接断裂的低通滤波法相比，形态学方法的一个重要优点是可以直接得到一幅二值图像；而从二值图像开始生成灰度图像的低通滤波法，需要对灰度图像进行阈值处理以便转换为二值形式（第 10 章中将讨论阈值处理）。显然，在这个应用中，集合 A 由许多不连贯的前景像素目标组成。

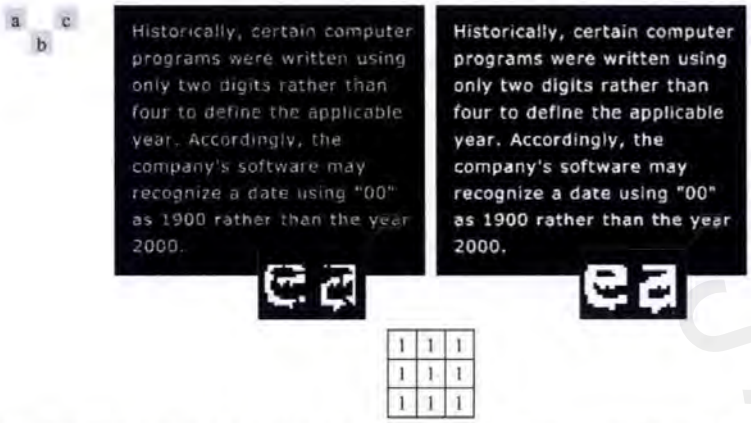


图 9.7　(a)带有断裂字符的低分辨率文本（见放大后的视图）；(b)结构元；(c)(b)对(a)的膨胀。断裂已连上 ■

6.2.3 对偶关系

腐蚀和膨胀相对于补集和反射彼此对偶:

$$(A \ominus B)^c := A^c \oplus B_{\text{reflect}}$$
$$(A \oplus B)^c := A^c \ominus B_{\text{reflect}}$$

换言之，B 对 A 的腐蚀是 B_{reflect} 对 A^c 的膨胀的补集，
而 B 对 A 的膨胀是 B_{reflect} 对 A^c 的腐蚀的补集.

(证明待补充)

当结构元相对于其原点对称时 (通常如此)，我们有 $B_{\text{reflect}} = B$，因此对偶性特别有用.
此时使用一个相同的结构元膨胀 A 的背景 A^c，然后求结果的补集，即可得到这个结构元对 A 的腐蚀.
而使用一个相同的结构元腐蚀 A 的背景 A^c，然后求结果的补集，即可得到这个结构元对 A 的膨胀.

6.2.4 实现

简单起见，我们可以使用距离来实现腐蚀和膨胀操作.
膨胀就是将前景边界像素的邻域内的像素都置为前景像素，
而腐蚀就是将背景边界像素的邻域内的像素都置为背景像素.

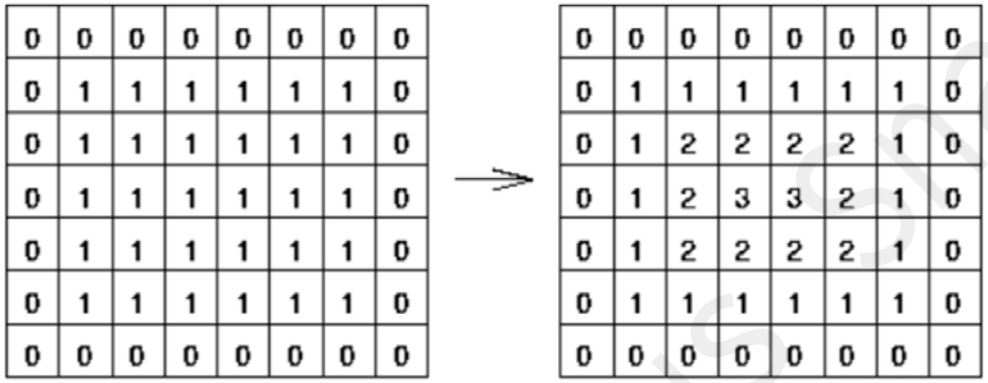
- ① **Manhattan 距离** (又称 city-block 距离)
  $(\text{2D image } \mathrm{D}_4 \text{ distance}/\text{3D image } \mathrm{D}_6 \text{ distance})$

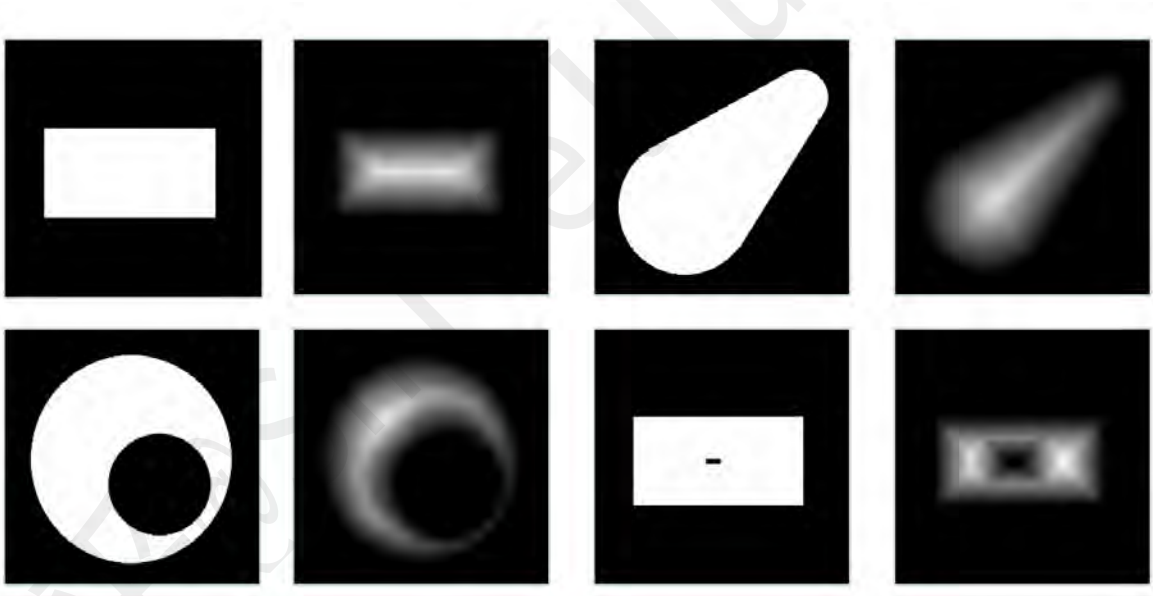
$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array}$$

- **② Chebyshev 距离**
  $(\text{2D image } \mathrm{D}_8 \text{ distance}/\text{3D image } \mathrm{D}_{26} \text{ distance})$

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

距离变换 (Distance Transform) 算法:
距离变换是一种图像处理技术，用于计算图像中每个前景像素到边缘的距离.





实现距离变换的方法有很多，常见的方式包括基于**动态规划**的算法.
它通过逐步更新每个像素的距离信息来避免直接计算每个像素到所有前景像素的距离，从而提高计算效率.

- **前向和后向扫描**: 通过前向扫描计算每个像素点到最近前景的距离，再通过后向扫描进一步优化.
  通常来说，前向扫描在一个方向上进行 (如从左上到右下)，后向扫描在反方向上进行 (如从右下到左上)
- **水平和垂直扫描**: 将图像分解成行和列的扫描，逐行逐列进行距离计算和更新，
  确保每个像素的距离基于其周围邻域的已有计算结果.

其时间复杂度通常为 $O(\text{Width} \cdot \text{Height})$
这是因为每个像素都要参与计算，且每次计算依赖于前一个像素的信息，因此处理复杂度是线性的.

OpenCV 库提供了距离变换算法的一个实现:

```
cv2.distanceTransform(src, distanceType, maskSize, dst=None)
```

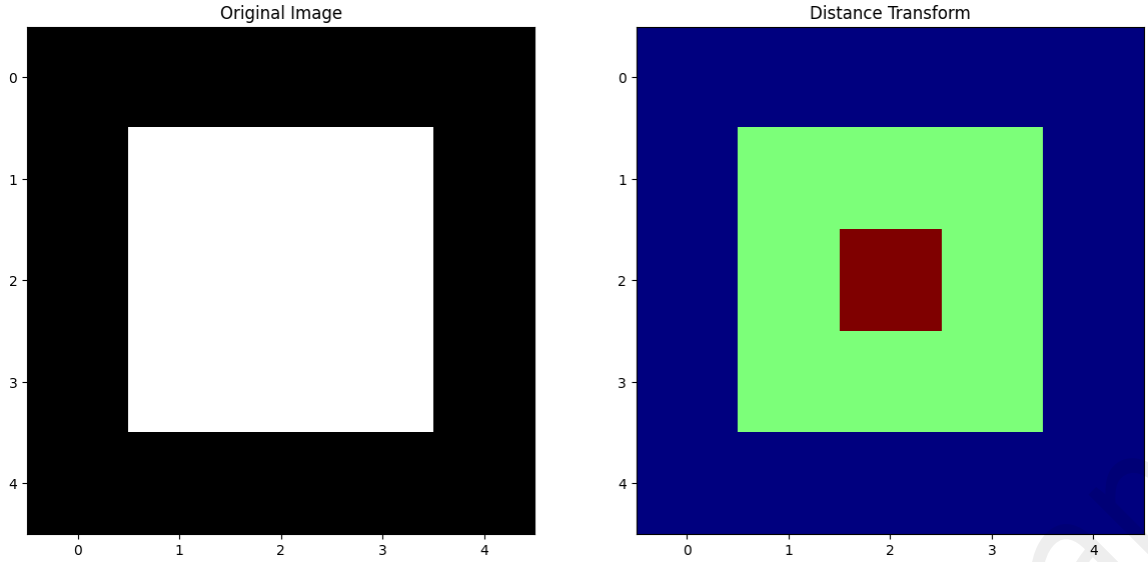
- `src`: 二值输入图像
- `distanceType`: 距离度量类型，常见的有:
  - `cv2.DIST_L2`: Euclid 距离
  - `cv2.DIST_L1`: Manhattan 距离
  - `cv2.DIST_C`: Chebyshev 距离
- `maskSize`: 用于计算距离的掩膜大小，决定计算时邻域的大小，常用值为 `3`、`5`、`7`
- `dst`: 输出图像，它将保存计算得到的距离图，每个像素值表示该像素到最近前景像素的距离.

函数调用:

```
# 创建一个简单的0-1矩阵 (5x5的二值矩阵)
image = np.array([
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
], dtype=np.uint8)

# 计算距离变换 (使用 Chebyshev 距离)
distance = cv2.distanceTransform(image, cv2.DIST_C, 5).astype(np.uint8)

# 显示距离变换结果
plt.subplot(121), plt.imshow(image, cmap='gray'), plt.title('Original Image')
plt.subplot(122), plt.imshow(distance, cmap='jet'), plt.title('Distance
Transform')
plt.show()

# 打印距离变换矩阵
print("Original Image:")
print(image)
print("\nDistance Transform Result:")
print(distance)
```

运行结果:

```
Original Image:
[[0 0 0 0 0]
 [0 1 1 1 0]
 [0 1 1 1 0]
 [0 1 1 1 0]
 [0 0 0 0 0]]

Distance Transform Result:
[[0 0 0 0 0]
 [0 1 1 1 0]
 [0 1 2 1 0]
 [0 1 1 1 0]
 [0 0 0 0 0]]
```



6.3 开运算 & 闭运算

膨胀扩展集合的组成部分，而腐蚀缩小集合的组成部分.
本节讨论另外两个重要的形态学运算: **开运算** (opening) 和**闭运算** (closing)

- 开运算通常平滑物体的轮廓，断开狭窄的狭颈，消除细长的突出物.
  结构元 B 对目标集合 A 的开运算定义为:

$$A \circ B := (A \ominus B) \oplus B$$

  即结构元 B 首先对 A 腐蚀，再对 A 膨胀.

- 闭运算同样平滑轮廓，但与开运算相反，它通常弥合狭窄的断裂和细长的沟壑，消除小孔，并填补轮廓中的缝隙.
  结构元 B 对目标集合 A 的闭运算定义为:

$$A \bullet B := (A \oplus B) \ominus B$$

  即结构元 B 首先对 A 膨胀，再对 A 腐蚀.

开运算的另一种解释是:
结构元 B 对目标集合 A 的开运算是 B 的所有包含在 A 中的平移的并集:

$$A \circ B := \bigcup \{B_{\text{translate}(x,y)} : (x,y) \in \mathbb{Z}^2 \text{ such that } B_{\text{translate}(x,y)} \subseteq A\}$$

删除比结构元更窄区域的能力是形态学开运算的关键特征之一.

式(9.10)有一个简单的几何解释：B 对 A 的开运算是 B 的所有平移的并集，以便 B 完全拟合 A。图 9.8(a)是包含一个集合（目标）A 的图像，图 9.8(b)是一个实心的圆形结构元 B。图 9.8(c)是 B 的一些平移，这些平移包含于 A，且在图 9.8(d)中显示为阴影的集合是所有这些平移的并集。这时，我们发现开运算是由两个不相交的子集组成的集合，因为 B 不能拟合 A 的中心位置的狭形段。后面很快讲到，删除比结构元更窄区域的能力是形态学开运算的关键特征之一。

B 对 A 的开运算是 B 的所有平移的并集，条件是 B 完全拟合于 A，这种解释可用公式写为

$$A \circ B = \bigcup \{(B)_z \mid (B)_z \subseteq A\} \tag{9.12}$$

式中，$\cup$ 表示大括号内所有集合的并集。

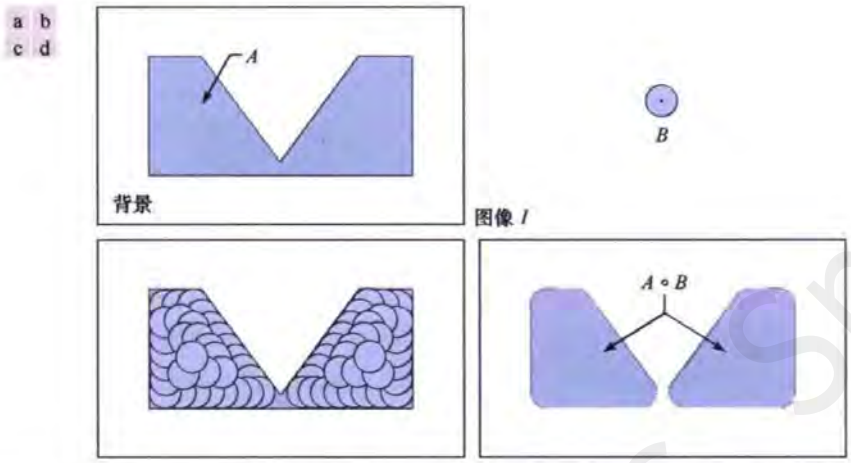


图 9.8 (a)由集合（目标）A 和背景组成的图像 I；(b)结构元 B；(c) B 在 A 内的平移（为清楚起见，A 显示为暗色）；(d)B 对 A 的开运算

除了是在边界的外侧平移 B，闭运算的几何说明类似.
换言之，闭运算是结构元 B 的所有不与 A 相交的平移的并集的补集:

$$A \bullet B := \left[\bigcup \{B_{\text{translate}(x,y)} : (x,y) \in \mathbb{Z}^2 \text{ such that } B_{\text{translate}(x,y)} \cap A = \emptyset\}\right]$$

除了是在边界的外侧平移 B，闭运算的几何说明类似。于是，闭运算是 B 的所有不与 A 重叠的平移的并集的补集。图 9.9 说明了这一概念。注意，闭运算的边界是在 B 不进入 A 的任何部分的前提下，由 B 可以到达的最远的那些点决定的。根据这一解释，我们可把 B 对 A 的闭运算写为

$$A \bullet B = \left[\bigcup \{(B)_z \mid (B)_z \cap A = \varnothing\}\right]^c \tag{9.13}$$

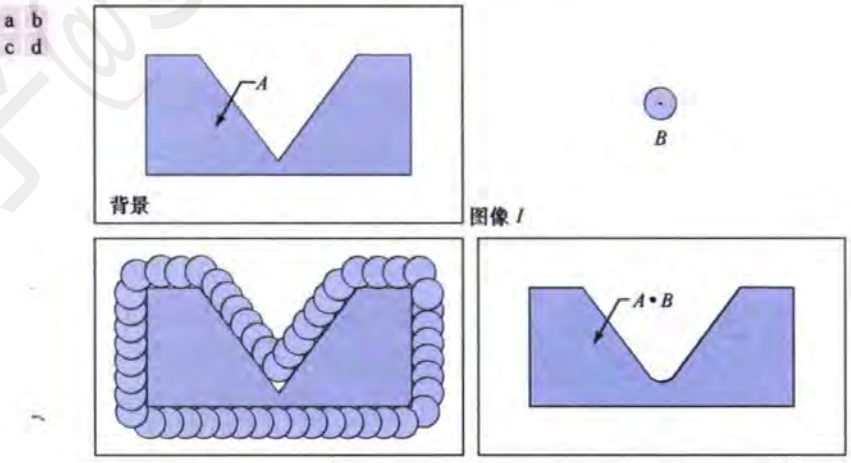


图 9.9 (a)由集合（目标）A 和背景组成的图像 I；(b)结构元 B；(c) B 不与 A 的任何部分重叠时，B 的平移（为清楚起见，A 显示为暗色）；(d)B 对 A 的闭运算

同膨胀与腐蚀一样，开运算和闭运算相对于补集和反射彼此对偶:

$$(A \circ B)^{\mathrm{c}} = (A^{\mathrm{c}} \bullet B_{\mathrm{reflect}})$$
$$(A \bullet B)^{\mathrm{c}} = (A^{\mathrm{c}} \circ B_{\mathrm{reflect}})$$

(证明待补充)

形态学开运算具有以下性质:

- 开操作的结果 $A \circ B$ 是 A 的一个子集
- 若 $A_1 \subseteq A_2$，则开操作的结果 $(A_1 \circ B) \subseteq (A_2 \circ B)$
- $(A \circ B) \circ B = A \circ B$ (这表明进行一次开运算后，继续做开运算是没有意义的)

对应地，形态学闭运算具有以下性质:

- 闭操作的结果 $A \bullet B$ 是 A 的一个包含集
- 若 $A_1 \subseteq A_2$，则闭操作的结果 $(A_1 \bullet B) \subseteq (A_2 \bullet B)$
- $(A \bullet B) \bullet B = A \bullet B$ (这表明进行一次闭运算后，继续做闭运算是没有意义的)

例 9.3　形态学开运算和闭运算。

图 9.10 详细地显示了开运算和闭运算的过程与性质。图 9.8 和图 9.9 的主要目标是总体几何解释，而图 9.10 则显示了各个过程，并重点说明最终结果的尺度与结构元的大小之间的关系。

图 9.10(a)是包含一个目标（集合）A 的一幅图像和一个圆盘形结构元，图 9.10(b)是腐蚀期间圆形结构元的各个位置。这一处理的结果是图 9.10(c)中不相交的集合。注意两个主要部分之间的连线是如何消除的。与结构元的直径相比，连线的宽度很窄，不能完全包含在集合的这一部分中，因此违反了腐蚀的定义。物体最右侧的两个部分同样如此。圆形无法拟合的突出部分已被删除。图 9.10(d)显示了对腐蚀后的集合进行膨胀的过程，图 9.10(e)显示了开运算的最终结果。形态学开运算删除了不能包含结构元的区域，平滑了目标轮廓，断开了细连线，删除了细凸起部分。

图 9.10(f)到(i)是用同一结构元对 A 进行闭运算的结果。与开运算一样，闭运算也平滑了目标的轮廓，但与开运算不同的是，闭运算连接了狭窄的断点，填充了细长的沟槽和小于结构元的目标。在本例中，闭运算的主要结果是，它填充了集合 A 左侧的小沟槽。

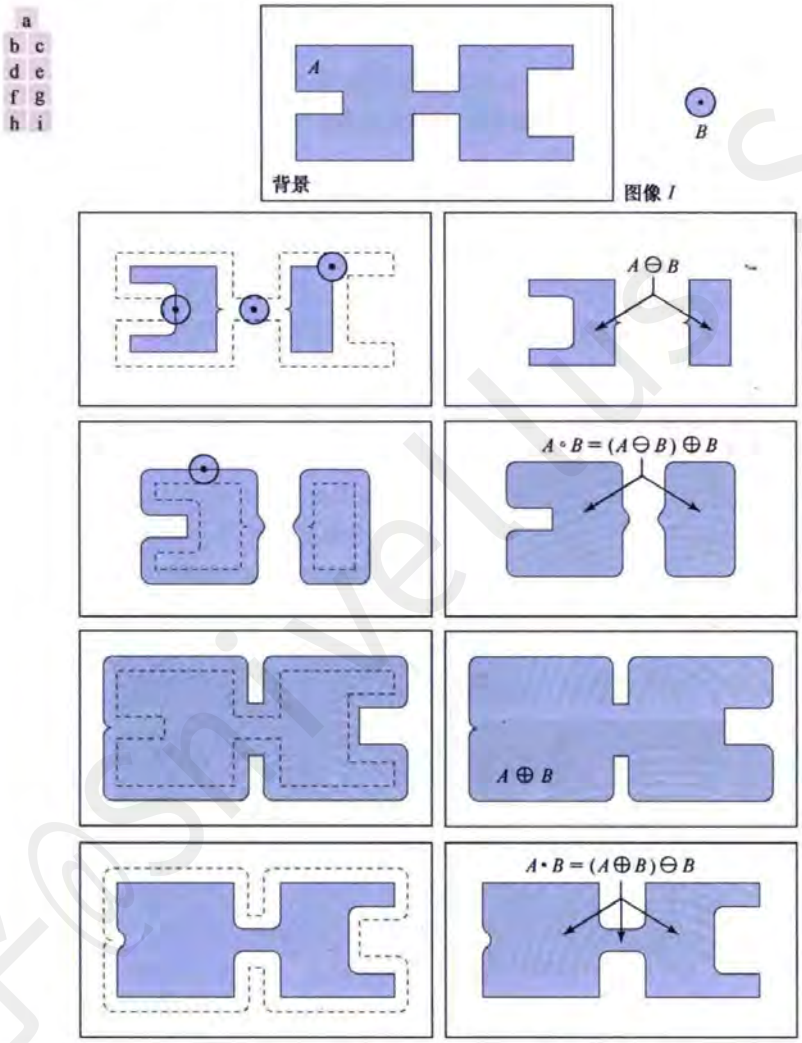


图 9.10　形态学开运算和闭运算：(a)由集合（目标）A 和背景组成的图像 I，以及实心的圆形结构元（黑点是原点）；(b)位于不同位置的结构元；(c)～(i)用于得到开运算和闭运算结果的形态学运算 ■

例 9.4　使用开运算和闭运算进行形态学滤波。

形态学运算可用于构建滤波器，这种滤波器概念上类似于第 3 章中讨论的空间滤波器。图 9.11(a)中的二值图像是被噪声污染指纹的一部分。根据前面的表示方式，A 是所有前景像素（白色）的集合，它包括感兴趣目标（指纹脊线）和白色的随机噪声斑点。背景照例显示为黑色。噪声是暗色背景上的白色斑点和白色指纹成分中的暗斑。目的是消除噪声及噪声对打印的影响，同时使图像的失真尽可能小。我们可以使用由开运算和闭运算组成的形态学滤波来达到这一目的。

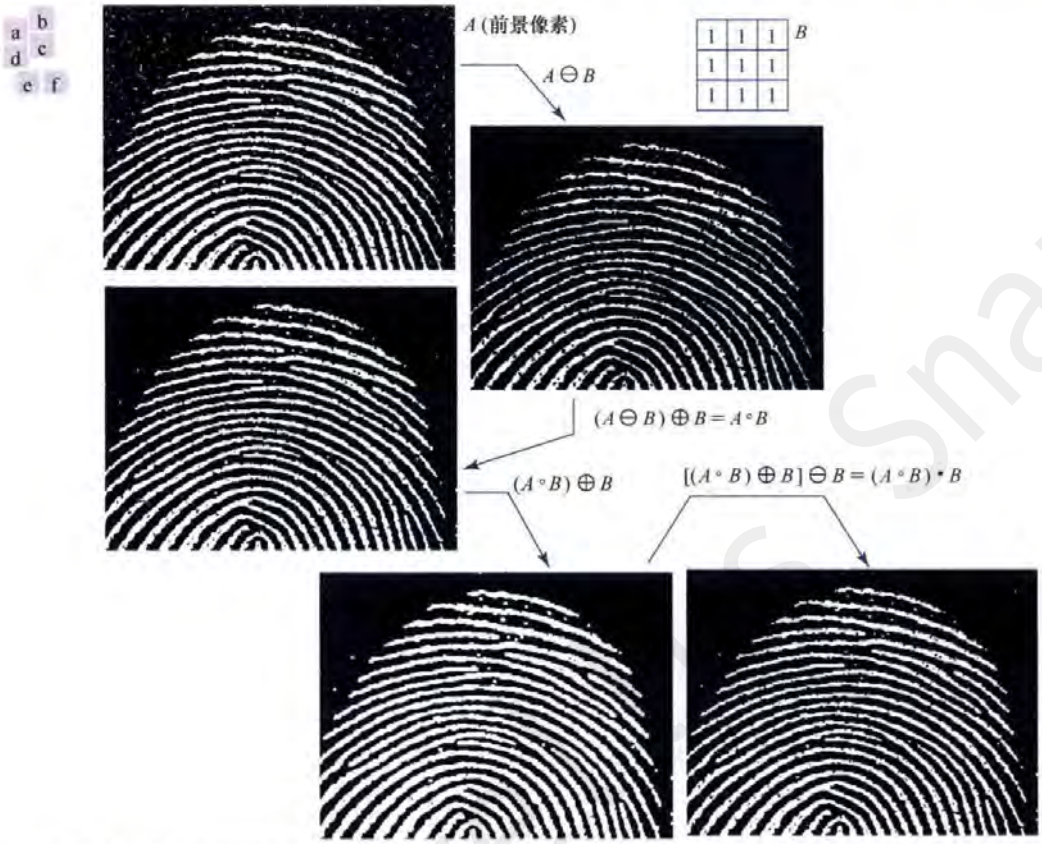


图 9.11　(a)噪声图像；(b)结构元；(c)腐蚀后的图像；(d)腐蚀的膨胀（A 的开运算）；(e)开运算的膨胀；(f)开运算的闭运算（原图由美国国家标准和技术研究所提供）

图 9.11(b)是所用的结构元，图 9.11 的余下部分是滤波运算的顺序，图 9.11(c)是 B 对 A 的腐蚀结果。背景中的白斑噪声在开运算的腐蚀阶段几乎被完全消除，因为这种情况下的大多数噪声成分都小于结构元。指纹内噪声元素（黑点）的尺寸实际上增大了。原因是随着目标被腐蚀，这些元素的内部边界增大。这种增大可通过对图 9.11(c)进行膨胀来抵消。图 9.11(d)显示了抵消后的结果。

刚才描述的两个运算构成了 B 对 A 的开运算。在图 9.11(d)中，我们注意到，开运算的实际效果是降低了背景和指纹本身中的噪声成分，但在指纹脊线之间却形成了新的断裂。为了消除这种不良的影响，如图 9.11(e)所示，我们对开运算的结果进行膨胀。膨胀后的结果中，大部分断裂得以修复，但脊线却变粗了，这一状况可通过腐蚀运算来弥补。图 9.11(f)是对图 9.11(d)中的开运算进行闭运算后的结果。在最终结果中，尽管消除了大部分噪声斑点，但仍然有一些类似于单个像素的噪声斑点。这些单像素噪声斑点可用本章后面讨论的方法消除。 ■

6.4 基本形态学算法

6.4.1 骨架

如图 9.25 所示，集合 A 的骨架 $S(A)$概念上很简单。由该图我们可以得出如下结论：

(a) 若 z 是 $S(A)$的一点，$(D)_z$ 是 A 内以 z 为圆心的最大圆盘，则不存在包含 $(D)_z$ 且位于 A 内的更大圆盘（不必以 z 为中心）。满足这些条件的圆盘 $(D)_z$ 称为最大圆盘。

(b) 若 $(D)_z$ 是一个最大圆盘，则它在两个或多个不同的位置与 A 的边界接触。

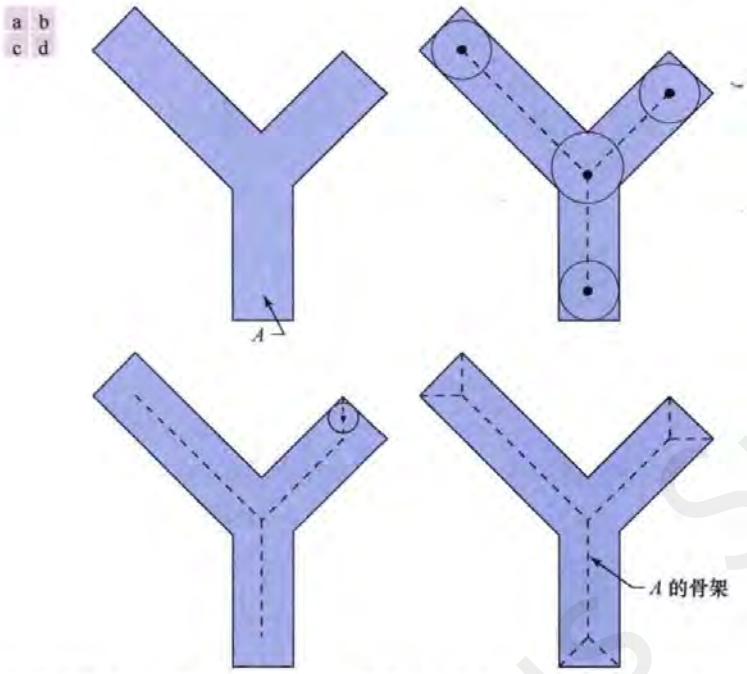


图 9.25 (a)集合 A；(b)最大圆盘的不同位置，这些最大圆盘的中心部分地定义了 A 的骨架；(c)另一个最大圆盘，其中心定义了 A 的骨架的不同线段；(d)完整的骨架（虚线）

前景集合 A 的**骨架** (skeleton) $\mathrm{Skeleton}(A)$ 可用腐蚀和开运算来表示:

$$S_k(A) := (A \ominus kB) - (A \ominus kB) \circ B$$
$$K := \max\{k : (A \ominus kB) \neq \emptyset\}$$
$$\mathrm{Skeleton}(A) := \bigcup_{k=0}^{K} S_k(A)$$

其中 B 是一个结构元，$(A \ominus kB)$ 表示 B 对 A 的连续 k 次腐蚀:

$$(A \ominus kB) := (\cdots((A \ominus B) \ominus B) \cdots) \ominus B$$

而 $K := \max\{k : (A \ominus kB) \neq \emptyset\}$ 代表 A 被腐蚀成空集所需的最小次数.
可以证明前景集合 A 可由这些子集来重建:

$$A := \bigcup_{k=0}^{K} (S_k(A) \oplus kB)$$

其中 $(A \ominus kB)$ 表示 B 对 A 的连续 k 次膨胀:

$$(A \oplus kB) := (\cdots((A \oplus B) \oplus B) \cdots) \oplus B$$

例 9.8　计算一个简单集合的骨架。

图 9.26 说明了刚才讨论的概念。第一列是原集合（顶部）及使用图中的结构元 B 两次腐蚀后的结果。注意，对 A 再进行一次腐蚀将产生空集，因此这时 $K=2$。第二列是用 B 对第一列中的集合进行开运算后的结果。根据开运算的拟合特性并结合图 9.8，我们很容易解释这些结果。第三列是第一列和第二列的差集。因此，第三列中的三项分别是 $S_0(A)$ 、$S_1(A)$ 和 $S_2(A)$ 。

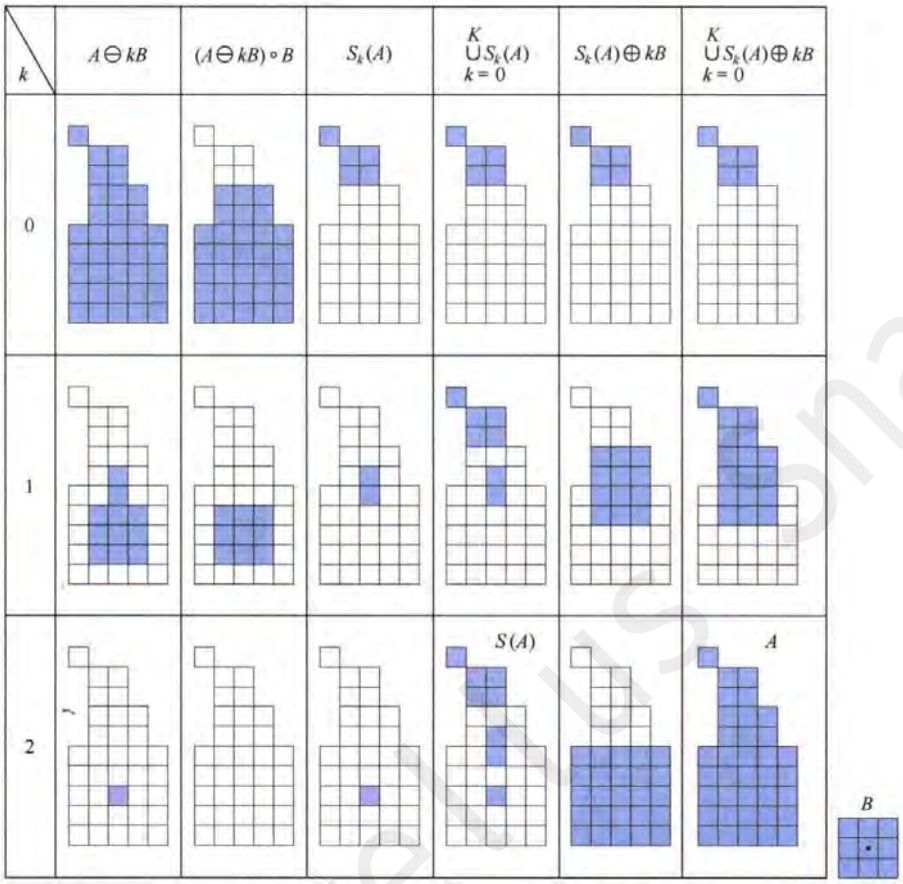


图 9.26　式(9.28)到式(9.33)的实现。原集合位于左上角，其形态学骨架位于第四列的底部。重建的集合位于第六列底部

第四列包含两个部分骨架，该列的底部是最终结果。最后的骨架不仅比需要的粗，而且未连上。这一结果并不令人意外，因为前面的形态学骨架公式中没有任何东西可以保证连接性。形态学根据已知集合的腐蚀和开运算给出了一个精致的公式，但在要求最大限度的细化、连通和最小的腐蚀时，则需要启发式公式（见 11.2 节）。

第五列和第六列中的各项处理的是由骨架子集重建原集合。第五列是对 $S_k(A)$ 的膨胀，即 $S_0(A)$ 、$S_1(A) \oplus B$ 和 $S_2(A) \oplus 2B = (S_2(A) \oplus B) \oplus B$ 。最后一列是根据式(9.32)重建的集合 A，它是第五列中膨胀后的骨架子集的并集。 ■

The End