形态学运算中腐蚀，膨胀，开运算和闭运算 (针对二值图而言)

6.1 腐蚀

腐蚀是一种消除边界点，使边界向内部收缩的过程。可以用来消除小且无意义的物体。

腐蚀的算法：
用 3x3 的结构元素，扫描图像的每一个像素
用结构元素与其覆盖的二值图像做“与”操作
如果都为 1，结果图像的该像素为 1。否则为 0。
结果：使二值图像减小一圈
把结构元素 B 平移 a 后得到 Ba，若 Ba 包含于 X，我们记下这个 a 点，所有满足上述条件的 a 点组成的集合称做 X 被 B 腐蚀(Erosion)的结果。用公式表示为：$E(X)=\{a|\ Ba \subset X\}=X \ominus B$，如图 6.8 所示。

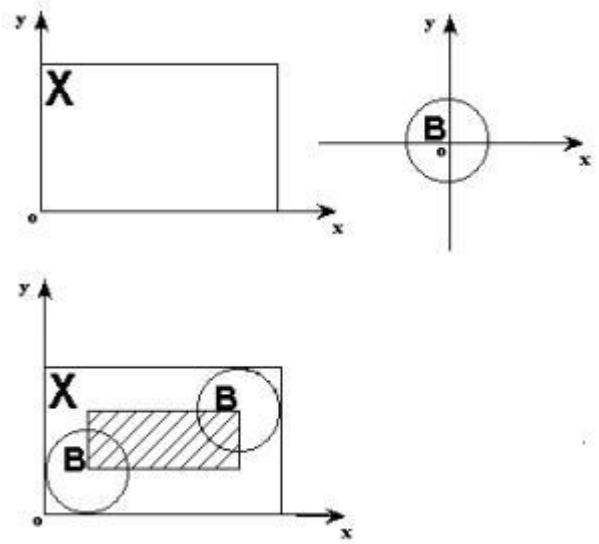

图 6.8　腐蚀的示意图

图 6.8 中 X 是被处理的对象，B 是结构元素。不难知道，对于任意一个在阴影部分的点 a，Ba 包含于 X，所以 X 被 B 腐蚀的结果就是那个阴影部分。阴影部分在 X 的范围之内，且比 X 小，就象 X 被剥掉了一层似的，这就是为什么叫腐蚀的原因。

值得注意的是，上面的 B 是对称的，即 B 的对称集 Bv=B，所以 X 被 B 腐蚀的结果和 X 被 Bv 腐蚀的结果是一样的。如果 B 不是对称的，让我们看看图 6.9，就会发现 X 被 B 腐蚀的结果和 X 被 Bv 腐蚀的结果不同。

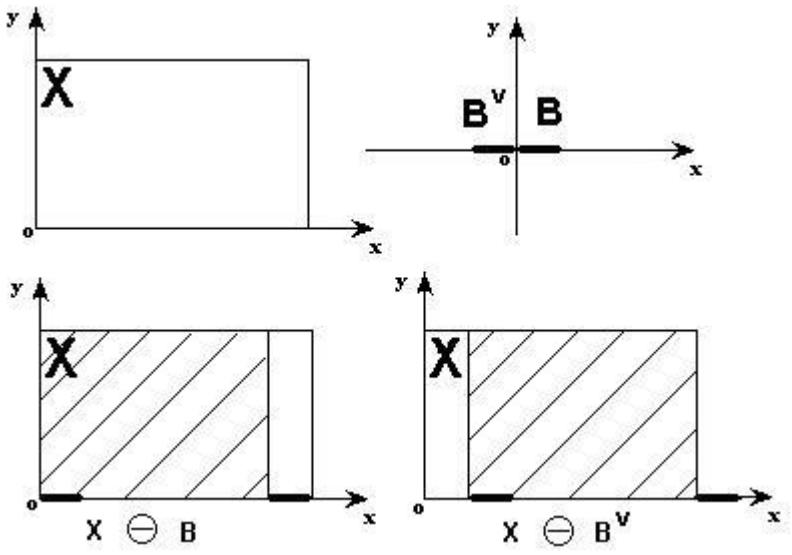

图 6.9　结构元素非对称时，腐蚀的结果不同

图 6.8 和图 6.9 都是示意图，让我们来看看实际上是怎样进行腐蚀运算的。

在图 6.10 中，左边是被处理的图象 X(二值图象，我们针对的是黑点)，中间是结构元素 B，那个标有 origin 的点是中心点，即当前处理元素的位置，我们在介绍模板操作时也有过类似的概念。腐蚀的方法是，拿 B 的中心点和 X 上的点一个一个地对比，如果 B 上的所有点都在 X 的范围内，则该点保留，否则将该点去掉；右边是腐蚀后的结果。可以看出，它仍在原来 X 的范围内，且比 X 包含的点要少，就象 X 被腐蚀掉了一层。

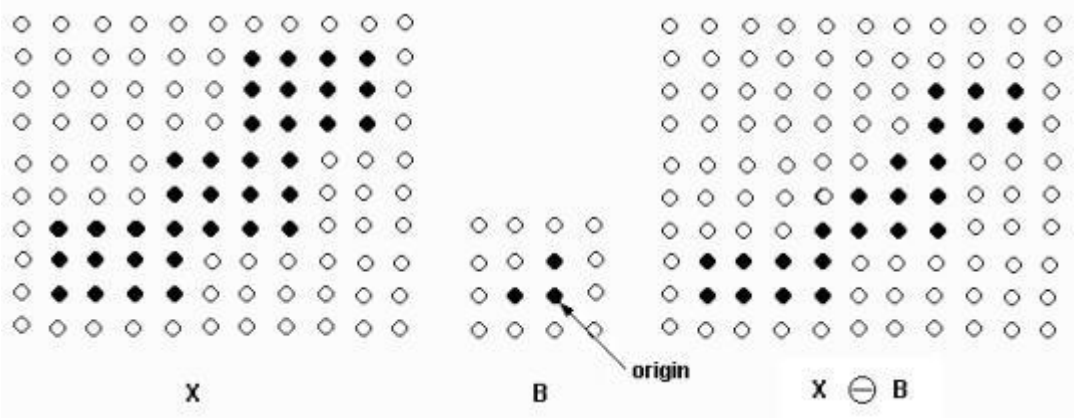

图 6.10　腐蚀运算

图 6.11 为原图，图 6.12 为腐蚀后的结果图，能够很明显地看出腐蚀的效果。

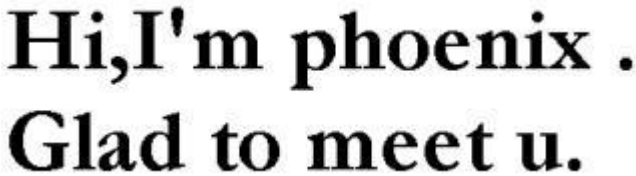

图 6.11　原图

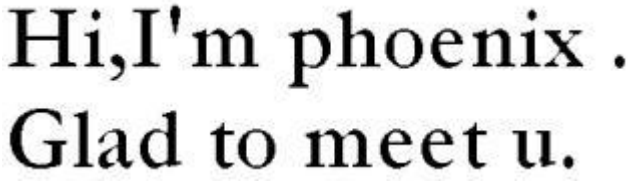

图 6.12　腐蚀后的结果图

下面的这段程序，实现了上述的腐蚀运算，针对的都是黑色点。参数中有一个 BOOL 变量，为真时，表示在水平方向进行腐蚀运算，即结构元素 B 为 $[1\ \ 1.\ \ 1]$；否则在垂直方向上进行腐蚀运算，即结构元素 B 为 $\begin{bmatrix}1\\1.\\1\end{bmatrix}$。

6.2 膨胀

膨胀是将与物体接触的所有背景点合并到该物体中，使边界向外部扩张的过程。
可以用来填补物体中的空洞。

膨胀的算法：

用 3x3 的结构元素，扫描图像的每一个像素
用结构元素与其覆盖的二值图像做“或”操作
如果都为 0，结果图像的该像素为 0。否则为 1
结果：使二值图像扩大一圈

膨胀(dilation)可以看做是腐蚀的对偶运算，其定义是：把结构元素 B 平移 a 后得到 Ba，若 Ba 击中 X，我们记下这个 a 点。所有满足上述条件的 a 点组成的集合称做 X 被 B 膨胀的结果。用公式表示为：$D(X)=\{a \mid Ba \uparrow X\}=X \oplus B$，如图 6.13 所示。图 6.13 中 X 是被处理的对象，B 是结构元素，不难知道，对于任意一个在阴影部分的点 a，Ba 击中 X，所以 X 被 B 膨胀的结果就是那个阴影部分。阴影部分包括 X 的所有范围，就象 X 膨胀了一圈似的，这就是为什么叫膨胀的原因。

同样，如果 B 不是对称的，X 被 B 膨胀的结果和 X 被 Bv 膨胀的结果不同。

让我们来看看实际上是怎样进行膨胀运算的。在图 6.14 中，左边是被处理的图象 X(二值图象，我们针对的是黑点)，中间是结构元素 B。膨胀的方法是，拿 B 的中心点和 X 上的点及 X 周围的点一个一个地对，如果 B 上有一个点落在 X 的范围内，则该点就为黑；右边是膨胀后的结果。可以看出，它包括 X 的所有范围，就象 X 膨胀了一圈似的。

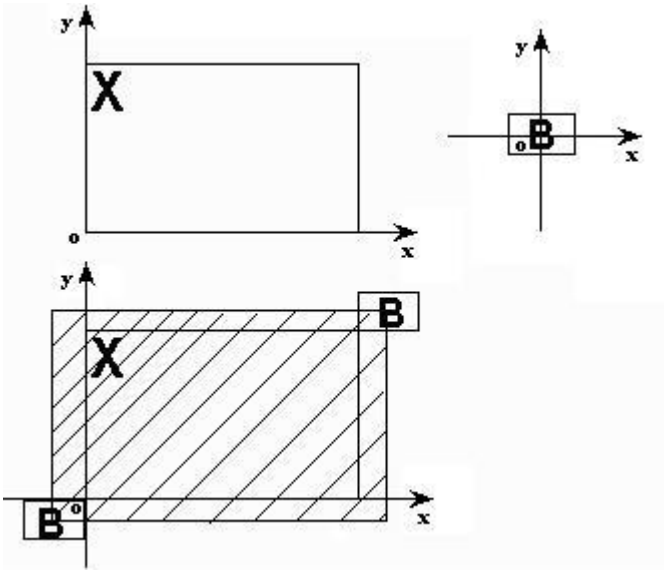

图 6.13　膨胀的示意图

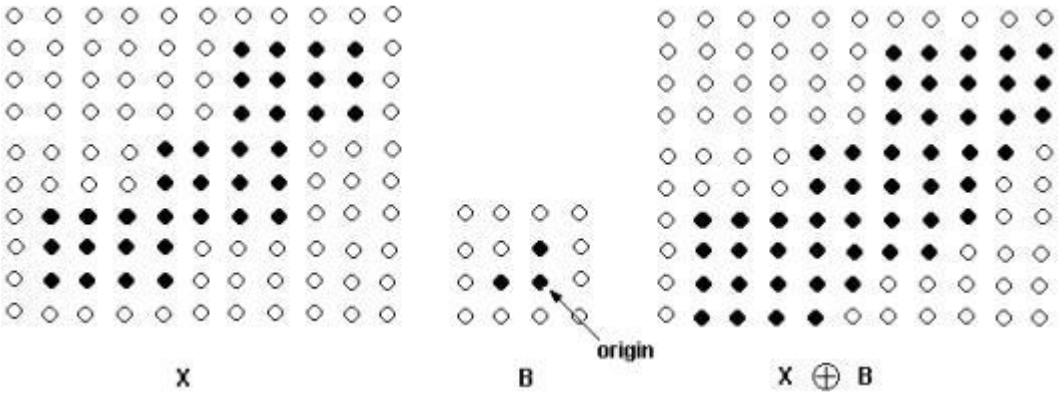

图 6.14　膨胀运算

图 6.15 为图 6.11 膨胀后的结果图，能够很明显的看出膨胀的效果。

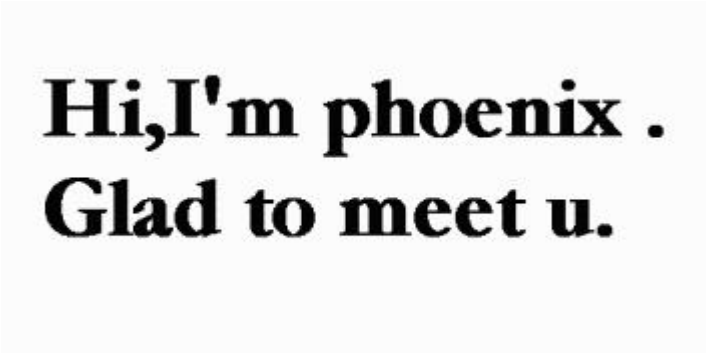

图 6.15　图 6.11 膨胀后的结果图

下面的这段程序，实现了上述的膨胀运算，针对的都是黑色点。参数中有一个 BOOL 变量，为真时，表示在水平方向进行膨胀运算，即结构元素 B 为 $[1\quad 1.\quad 1]$；否则在垂直方

向上进行膨胀运算，即结构元素 B 为 $\begin{bmatrix}1\\1.\\1\end{bmatrix}$。

6.3 开运算

先腐蚀后膨胀的过程称为开运算。
用来消除小物体、在纤细点处分离物体、平滑较大物体的边界的同时并不明显改变其面积。
先腐蚀后膨胀称为开(open)，即 OPEN(X)=D(E(X))。
让我们来看一个开运算的例子(见图 6.16)：

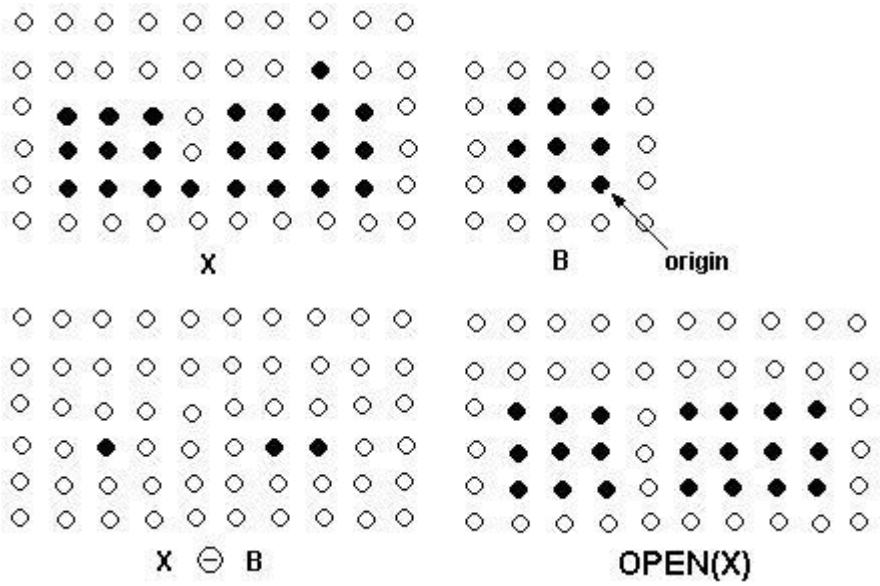

图 6.16 开运算

在图 16 上面的两幅图中，左边是被处理的图象 X(二值图象，我们针对的是黑点)，右边是结构元素 B，下面的两幅图中左边是腐蚀后的结果；右边是在此基础上膨胀的结果。可以看到，原图经过开运算后，一些孤立的小点被去掉了。一般来说，开运算能够去除孤立的小点，毛刺和小桥(即连通两块区域的小点)，而总的位置和形状不变。这就是开运算的作用。要注意的是，如果 B 是非对称的，进行开运算时要用 B 的对称集 Bv 膨胀，否则，开运算的结果和原图相比要发生平移。图 6.17 和图 6.18 能够说明这个问题。

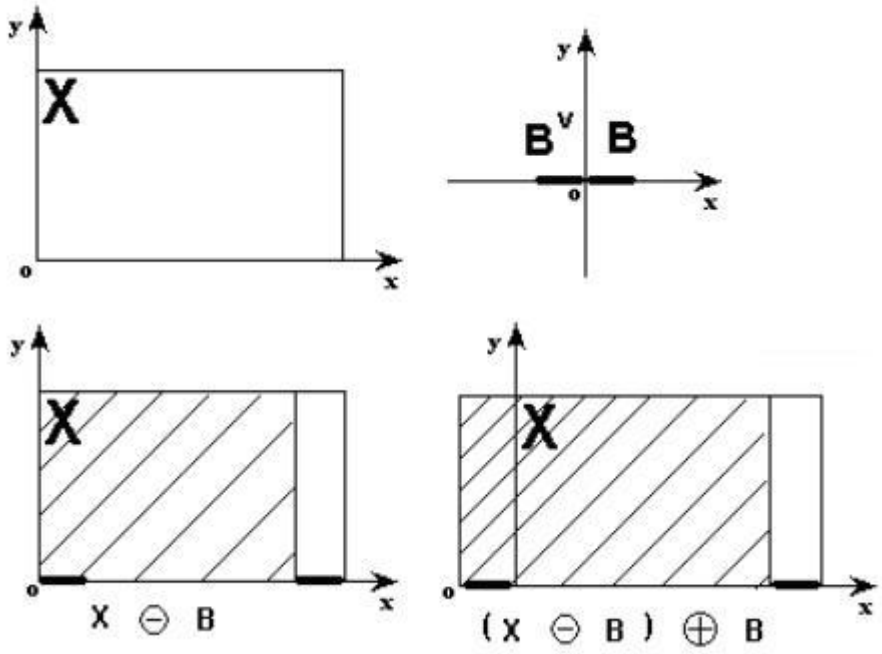

图 6.17 用 B 膨胀后，结果向左平移了

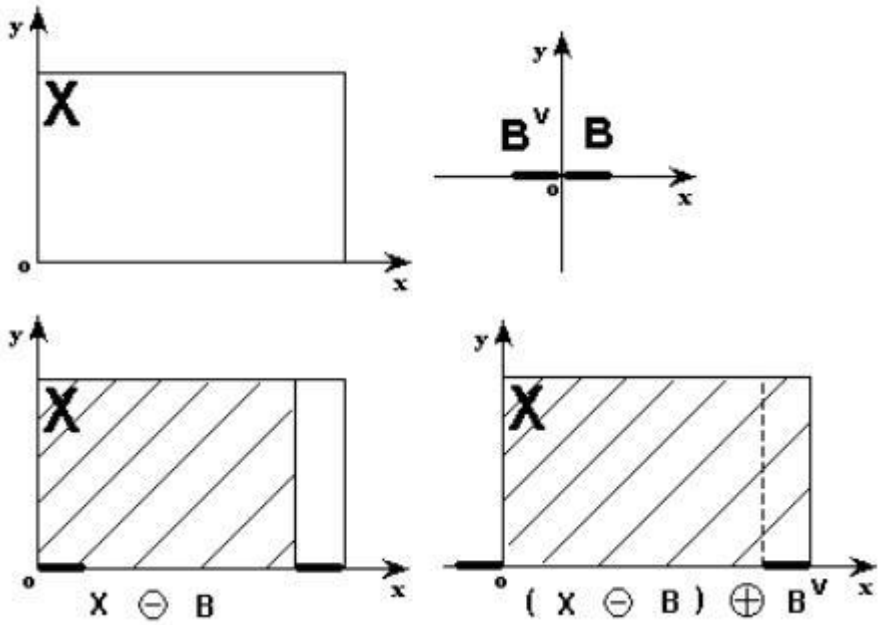

图 6.18 用 Bv 膨胀后位置不变

图 6.17 是用 B 膨胀的，可以看到，OPEN(X)向左平移了。图 18 是用 Bv 膨胀的，可以看到，总的位置和形状不变。

图 6.19 为图 6.11 经过开运算后的结果。

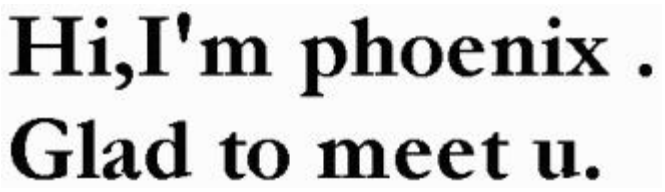

图 6.19 图 6.11 经过开运算后的结果

开运算的源程序可以很容易的根据上面的腐蚀，膨胀程序得到，这里就不给出了。

6.4 闭运算

先膨胀后腐蚀称为闭(close)，即 CLOSE(X)=E(D(X))。
让我们来看一个闭运算的例子(见图 6.20)：

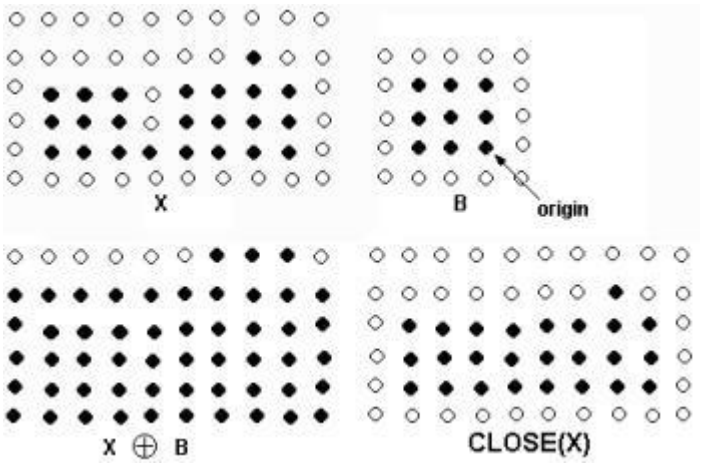

图 6.20　闭运算

在图 6.20 上面的两幅图中，左边是被处理的图象 X(二值图象，我们针对的是黑点)，右边是结构元素 B，下面的两幅图中左边是膨胀后的结果，右边是在此基础上腐蚀的结果可以看到，原图经过闭运算后，断裂的地方被弥合了。一般来说，闭运算能够填平小湖(即小孔)，弥合小裂缝，而总的位置和形状不变。这就是闭运算的作用。同样要注意的是，如果 B 是非对称的，进行闭运算时要用 B 的对称集 Bv 膨胀，否则，闭运算的结果和原图相比要发生平移。

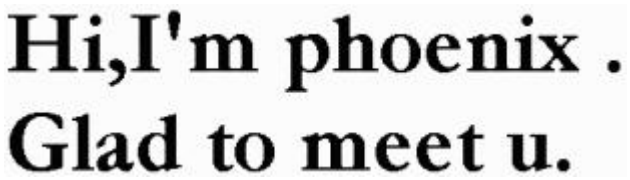

图 6.21　图.611 经过闭运算后的结果

闭运算的源程序可以很容易的根据上面的膨胀，腐蚀程序得到，这里就不给出了。
你大概已经猜到了，开和闭也是对偶运算，的确如此。用公式表示为(OPEN(X))c=CLOSE((Xc))，或者(CLOSE(X))c =OPEN((Xc))。即 X 开运算的补集等于 X 的补集的闭运算，或者 X 闭运算的补集等于 X 的补集的开运算。这句话可以这样来理解：在两个小岛之间有一座小桥，我们把岛和桥看做是处理对象 X，则 X 的补集为大海。如果涨潮时将小桥和岛的外围淹没(相当于用尺寸比桥宽大的结构元素对 X 进行开运算)，那么两个岛的分隔，相当于小桥两边海域的连通(对 Xc 做闭运算)。

6.5 细化运算

细化(thinning)算法有很多，我们在这里介绍的是一种简单而且效果很好的算法，用它就能够实现从文本抽取骨架的功能。我们的对象是白纸黑字的文本，但在程序中为了处理的方便，还是采用 256 级灰度图，不过只用到了调色板中 0 和 255 两项。
所谓细化，就是从原来的图中去掉一些点，但仍要保持原来的形状。实际上，是保持原图的骨架。所谓骨架，可以理解为图象的中轴，例如一个长方形的骨架是它的长方向上的中轴线；

正方形的骨架是它的中心点；圆的骨架是它的圆心，直线的骨架是它自身，孤立点的骨架也是自身。文本的骨架嘛，前言中的例子显示的很明白。那么怎样判断一个点是否能去掉呢？显然，要根据它的八个相邻点的情况来判断，我们给几个例子(如图 6.22 所示)。

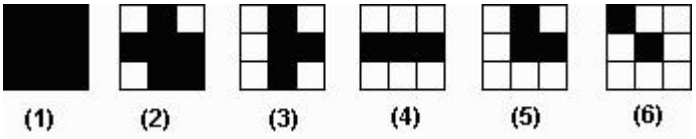

图 6.22　根据某点的八个相邻点的情况来判断该点是否能删除

图 6.23 经过细化后，我们预期的结果是一条水平直线，且位于该黑色矩形的中心。实际的结果确实是一条水平直线，但不是位于黑色矩形的中心，而是最下面的一条边。为什么会这样，我们来分析一下：在从上到下，从左到右的扫描过程中，我们遇到的第一个黑点就是黑色矩形的左上角点，经查表，该点可以删。下一个点是它右边的点，经查表，该点也可以删，如此下去，整个一行被删了。每一行都是同样的情况，所以都被删除了。到了最后一行时，黑色矩形已经变成了一条直线，最左边的黑点不能删，因为它是直线的端点，它右边的点也不能删，因为如果删除，直线就断了，如此下去，直到最右边的点，也不能删，因为它是直线的右端点。所以最下面的一条边保住了，但这并不是我们希望的结果。解决的办法是，在每一行水平扫描的过程中，先判断每一点的左右邻居，如果都是黑点，则该点不做处理。另外，如果某个黑点被删除了，那么跳过它的右邻居，处理下一个点。这样就避免了上述的问题。