使用线段表实现线段编码与种子填充快速算法

陆宗骐 朱 煜
(华东理工大学信息科学与工程学院,上海 200237)

摘 要 在图像处理中常用水平线段集表示区域,提出了一种采用线段表结构来表示区域,并给出使用线段表改进传统像素标记与种子填充算法的快速算法。该算法中,线段表的每个表项对应一条水平线段,它包含 L、x_L、x_R、y 和 F 5 个参数,即表示上下线段连通关系的标记,左、右端点的 x 坐标,线段的 y 坐标与表示线段特征或类型的标志。具有相同标记的线段表表项构成连通区域。以线段表为基础,水平与垂直方向的连通检测可分别进行,水平方向可通过行程编码实现,垂直方向则通过比较上下线段的端点坐标来确定。线段编码是像素标记的改进,由行程编码、线段标记、统一标记与标记排序 4 个步骤组成。采用线段表结构后,线段编码中利用桶排序,种子填充新算法中,利用队列结构并避免重复扫描来提高效率,与轮廓填充算法相比较效率都可提高近一倍。

关键词 线段表 线段编码 种子填充 像素标记 轮廓填充 快速算法
中图法分类号: TP391.41 **文献标识码**: A **文章编号**: 1006-8961(2009)03-0499-06

Fast Algorithm of Line Segment Encoding and Seed Filling Based on Line Segment Table

LU Zong-qi, ZHU Yu
(*The School of Information on Science and Engineering*, *East China University of Science and Technology*, *Shanghai* 200237)

Abstract Horizontal line can be used to precisely describe the shape of a certain region. Line segment table is the data structure of line. In this paper, a fast algorithm of line segment coding and seed filling based on line segment table is proposed. Every element of the table describes a horizontal line. The five parameters of the table are L, x_L, x_R, y and F. They denote the symbol of relationship of up and down rows, the x coordinates of left and right end, the y coordinate of the row and a flag for line type respectively. Base on line segment table, the connectivity of rows and columns can be calculated conveniently. Fast algorithm of line segment encoding follows the steps of line searching, labeling and sorting. The fast algorithm achieved by the concept of line segment table is 1/2 faster than contour filling. Results show that the data structure of line segment table is much more effective and flexible for data storage and data processing.

Keywords line segment table, line segment encoding, seed filling, pixel labeling, contour filling, fast algorithm

1 引 言

Windows 95/NT 的图形设备接口(GDI)中增加了两种新的图形对象——路径和区域。路径表示一系列相互连接的直线和曲线,路径可以画出区域的轮廓或者被填充。区域是由曲线定义的区间,区域可由一个或多个形状(矩形、多边形或椭圆等)组成。区域常用于剪裁和击中测试。区域之间还可以作逻辑运算,如与、或、异或、求补及求反等操作来合并或分割。对于特定形状的区域,可以使用多个矩形来表示其大致形状。如果矩形足够小,一定数量的矩形就能够精确地表示区域的形状,GDI 中还有专门的函数来获得区域的矩形集。这两种图形对象的引入使得应用程序可以方便地建立复杂区域,绘制和填充不规则图形[1]。

收稿日期:2007-02-07;**改回日期**:2007-10-17

第一作者简介:陆宗骐(1945 ~),男。教授。主要研究领域为图像处理与模式识别。编著《C/C ++ 图像处理编程》和合作编著《Visual C ++ . NET图像处理编程》。E-mail: zongqilu@163.com

在图像处理中,应用广泛的表示区域轮廓的链码与 GDI 中的路径相对应,不同在于,构成路径的对象是图形元素(直线或曲线),而由链码表示的轮廓则是像素级的,它可以表示非常复杂并且不规则的形状。对于 GDI 中的区域在图像处理中目前尚无类似的数据结构。虽然,也有文献中出现采用水平线段表示区域的例子[2-3],但由于注意的侧重点不同,文献[2]中的线段编码用线段左端点坐标和其行程表示线段,文献[3]则侧重于线段间联系的结构关系,它们的可操作性还无法与 GDI 中的区域相比较。文献[4]则提出了用线段表作为表示区域的数据结构来实现线段编码,以提高像素标记的效率。给出了利用线段表分离外轮廓与孔,以及搜索种子点的例子,还给出了线段表与链码之间的转换算法。文献[5]则使用线段表实现了区域的腐蚀与膨胀。

本文介绍使用线段表实现线段编码与种子填充快速算法的原理及其实现方法。

2 线段表与线段编码

2.1 线段表及其应用

2.1.1 线段表的定义[4]

区域可用水平线段的集合精确地进行描述,这个集合的数据结构就是线段表。线段表的每个表项对应一条水平线段,它包含 L(label)、x_l、x_r、y 和 F(flag)5 个参数,即表示上下线段间连通(邻接)关系的标记、左、右端点的 x 坐标,线段的 y 坐标和表示线段特征或类型的标志。这些参数足以表示水平线段的主要特征并有助于简化处理程序。

2.1.2 上下线段的连通性[4]

连通区域的搜索是图像处理中一种使用十分频繁的操作,它的基础是像素之间的连通性。对于水平线段而言则是上下线段的连通性,这可以通过比较线段的端点坐标来实现,即用下列逻辑关系进行判别:

$$\text{IF}(x_{1r} >= x_{2l} - N)\ \text{AND}\ (x_{1l} <= x_{2r} + N)\ \text{THEN}$$

$$T = \text{TRUE}$$

$$\text{ELSE}\ T = \text{FALSE}$$

其中,(x_{1l}, x_{1r})、(x_{2l}, x_{2r}) 分别表示相邻的上下行中两条线段左、右端点的 x 坐标。T 等于 TRUE 表示两线段连通;T 等于 FALSE 表示不连通。在两种连通条件下仅常数 N 取不同的值,4 连通时取 0,8 连通时取 1。

2.1.3 线段表的应用

击中测试是图形图像应用中经常使用的操作。一个简单的解决办法是建立一个模板图像,将所需区域填上专有颜色,然后通过检查输入位置在模板图像上的数值来判定是否击中。有了线段表也可完成这种判别,只要遍历整个线段表,看输入点是否落在其上即可。这种方法虽然在处理速度上不及使用模板图像,但较为节省内存空间。

图像的逻辑运算也可看作区域的合并、求交与求反等操作。利用集合运算的基本定律(交换律、分配律与结合律)可以证明,它们也可以看成各行内线段分量同类操作结果之并。因此,区域的逻辑运算也可利用线段表实现。

2.2 线段编码

2.2.1 像素标记[3,6]

图像经过阈值处理得到一幅二值化图像,图中常常包括多个区域,需要通过标记把它们区分开来。像素标记法就是检查各个像素与其相邻像素的连通性实现的。标记按扫描方式从左到右,从上到下依次进行。每个像素从它的上面或左面邻接像素获得标记,如果像素的上面或左面邻接像素中没有已标记的像素,则赋予新的标记。当上面或左面邻接像素的标记不同,发生冲突时,则记下这些标记,第 1 次标记结束后再将它们统一为同一标记。

像素标记法的标记对象是像素,由于相邻像素的邻点存在重叠,不可避免地引起重复操作,降低处理效率。同时,内存中的成块操作优于逐字节操作。因此,若能先在水平方向将相邻的像素检测出来,然后再对它们进行标记必定可以提高效率,这就是线段编码法。线段编码法标记的对象则是水平线段,由此可以通过扫描先得到各连通区域的线段表,进而对各表项进行标记与排序。

2.2.2 线段编码的步骤[4]

线段编码过程可分为行程编码、线段标记、统一标记和标记排序 4 个步骤,其中中间两个阶段与像素标记相似。线段编码的实现步骤如下:

(1)逐行扫描得到上面定义的初始线段表,即获取区域内所有水平线段端点的坐标值,并对标记 L 和标志 F 进行初始化,通常先赋予 0 值。

(2)当前线段与上面相邻行的线段做连通检测,并作相应处理:

① 若两线段相连通且当前线段尚未做标记,则将上面线段的标记赋予当前线段。

② 若两线段相连通且当前线段已做了标记,则记下这两条线段标记间的等价关系。

③ 若当前线段与上一行中的所有线段都不连通,则赋予此线段新的标记。

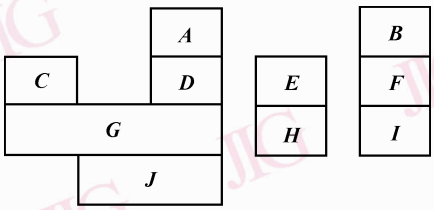
(3)将具有等价标记的连通线段统一为相同的标记,使得同一连通区域内的所有线段都采用相同的标记。

(4)整个线段表按标记值的大小重新进行排序,同一标记值线段的顺序不作改变。

这样图像中的各个区域依次排列在线段表中,每个区域对应线段表中连接在一起的一段。

在实际使用时,步骤(3)通常看作步骤(2)的附属操作,并不单独使用。因此,整个线段编码过程可分为行程编码、线段标记(包括统一标记)和标记排序 3 个步骤,它们的结果都可独立使用。

图 1(a)给出了一幅有 3 个区域的图像,$A \sim J$ 表示组成区域的水平线段。它的编码过程见图 1(b),图中的每一行表示处理的一个步骤,其中的下划线是提示需要引起注意的地方。由得到的线段表可知,图中共有 3 个区域,并可分别计算出它们的面积,或进行其他后续处理。



(a) 有3个连通区域的图像

*A*0,*B*0,*C*0,*D*0,*E*0,*F*0,*G*0,*H*0,*I*0,*J*0

*A*1,*B*2,*C*3,*D*1,*E*4,*F*2,*G*1(1,3),*H*4,*I*2,*J*1

*A*1,*B*2,*C*1,*D*1,*E*4,*F*2,*G*1,*H*4,*I*2,*J*1

*A*1,*C*1,*D*1,*G*1,*J*1,*B*2,*F*2,*I*2,*E*4,*H*4

(b) 线段编码过程

图 1　连通区域与线段编码

Fig. 1　The connected region and line encoding

2.2.3　桶排序[7]

线段编码中的第 4 步标记排序可将同一连通区域的表项集中排列在一起。由于线段表各项有 5 个参数,数据较长,排序的效率与排序方法有着密切的关系。本算法中,采用了排序效率较高的桶排序。

桶排序是 2 维数据,如坐标数据的一种高效排序方法。多边形填充中的 y 桶排序算法为每条扫描线建立一个存储单元或者称为桶。将每条线段的信息置入与该线段的 y 值相对应的桶中。桶本身是有序的,因此排序操作只牵涉存放在同一桶中的数据,从而大大提高了排序的效率。

以标记为桶的排序步骤如下:

(1)统计每个标记的表项(线段)数。

(2)建立新的线段表,确定每个桶的位置。

(3)将各表项依次传入新线段表相应的桶中。

由于搜索线段是以光栅扫描方式进行的,故桶内无需再重新排序。图 1(b)中第 3 行至第 4 行的转换是桶排序的一个典型实例。

原来的线段编码算法[4]没有注意效率采用了简单的冒泡排序算法,改用桶排序后避免了大量的数据交换,排序效率提高了 50 ~ 100 倍,而使线段编码的总效率也能有近十倍的提高,如表 1 所示。另外,改进算法中行程编码与统一标记部分也有所改进。

表 1　不同标记排序算法的运行时间

Tab. 1　Processing times of different label sorting methods

单位:ms

		PIII 664M	CM 1.6G	PIV 2.66G
原算法	冒泡排序	45.9	19.9	11.5
	4 步骤合计	57.8	23.7	13.8
改进算法	桶排序	0.89	0.14	0.11
	4 步骤合计	8.0	2.1	1.4

原来的线段编码算法在效率上远低于种子填充与轮廓填充算法,改进后则成了其中效率最高的算法,如表 2 所示。

表 2　各算法在不同类型计算机上的运行时间

Tab. 2　Processing times of various algorithms on different CPUs

单位:ms

方法		PIII664M	CM1.6G	PIV2.66G
线段编码	行程编码	4.0	1.2	0.8
	线段标记	0.87	0.46	0.29
	标记排序	0.89	0.14	0.11
	线段填充	2.22	0.30	0.24
	合计	8.0	2.1	1.4
种子填充				
算法 2	堆栈法	37.0	14.9	10.1
算法 3	队列法	32.6	12.3	8.1
算法 4	快速算法	10.8	2.7	1.8
轮廓填充	取图时间	2.8	0.2	0.4
	轮廓跟踪	8.6	3.7	2.0
	轮廓填充	3.7	1.6	1.0
	合计	15.1	5.5	3.4

3 种子填充及其快速算法

3.1 种子填充

3.1.1 传统的种子填充算法[7]

种子填充算法假设区域中至少有一个像素是已知的,它被称为种子。然后设法找到区域内所有其他像素,并对它们进行填充。初期的种子填充采用递归算法,它在填充种子点的同时不断地将它的邻点作为新的种子压入堆栈,填充过程一直进行到堆栈空为止。

种子填充算法是计算机图形学中使用十分广泛的区域填充算法,若将填充的颜色看作标记则也可用于图像处理中连通区域的标记。

有两类不同的种子填充算法,即边界填充和泛填充。填充边界定义区域的算法称为边界填充算法,填充像素定义区域的算法称为泛填充算法。前者需在区域内部选择种子点,种子的选择较后者困难,后者则可将区域上的任一像素选作种子点,包括边界像素,因此,只要通过扫描方式找到区域的任一像素即可填充整个区域。计算机图形学中主要使用边界填充算法,而在图像处理中主要使用泛填充算法。以像素为单位的种子填充算法效率较低,往往采用效率较高的经过改进的基于搜索水平线段的扫描线种子填充法。

3.1.2 根据行程分析的标记方法[6]

根据行程分析的标记方法是扫描线种子填充算法中的一种,这里所说的行程就是水平线段。它的实现步骤如下:

(1)用光栅扫描方式找到最初遇到的对象像素,并给包含这个像素的行程分配新的标号。

(2)把与已分配了标号的行程上、下(4-/8-)连接的所有行程的最左端像素的坐标记录在堆栈中(忽略已经分配了标号的行程)。

(3)若堆栈已空,则返回步骤(1)。若不空,则取出栈顶的坐标,给包含该像素的行程分配标号后,返回步骤(2)。

以上种子填充算法有两个缺点,一是只对区域内的像素做了标记,并没有得到可供后续处理直接使用的区域描述。因此,区域做了标记后,还需要采用轮廓跟踪等算法再将所需区域提取出来。二是算法中存在重复操作,效率不高。如在寻找种子点与实际填充时需两次扫描同一线段。

3.1.3 利用队列结构保存区域参数

上面采用递归算法的填充过程使用后进先出的堆栈结构,操作总在栈顶进行,前面数据出栈后即被新进的数据所破坏。所以区域填充结束无法保留完整的区域数据。

新算法将填充过程改为顺序处理,即将堆栈改为队列,改后进先出为先进先出,以达到既可控制填充过程又便于保留所得区域参数的目的。每检测得到一条水平线段就将它的参数存入线段表。当此线段作为种子从处理队列中取出时,再给图像上的相应线段填充(做标记),然后检查它的上下两行邻接部分搜索新的线段。这样,就可以克服传统种子填充算法的第一个缺点,填充结束同时能得到区域的线段表。

3.2 快速算法

3.2.1 避免重复扫描

在 3.1.2 和 3.1.3 中介绍的种子填充算法由于没有注意垂直方向的搜索方向,为了算法的通用性,需在每一种子线段的上下行都进行扫描。因此,存在重复操作,影响处理速度。图 2 列出了 4 邻接向下搜索时避免重复的 4 种情况。图中黑色线段为以前检测到的线段,灰色线段为新检测到的线段,空心矩形为待检测的种子线段。在这种情况下,前面检测到的线段已做过处理,以后无需再做检测,故不用再包含在种子线段中。

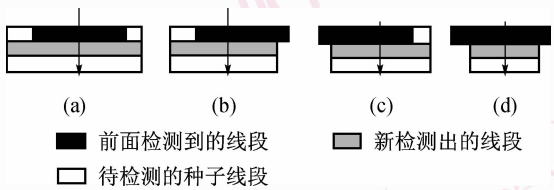


图 2 4 邻接向下扫描时的种子线段

Fig. 2 The seed line segment while 4-neighborhood scanning-down

为了叙述方便起见,将图 2 中与搜索方向一致的种子线段称为主种子线段,将与搜索方向反向的种子线段称为辅种子线段。显然,每一线段必有一条主种子线段,并且与此线段等长。辅种子线段的情况则比较复杂,在种子线段的数量和长度方面都不一样,可有 0 ~ 2 条不等,共 4 种情况。

3.2.2 种子线段端点的确定

假设 x_{11} 与 x_{1r} 为前面检测到的线段左、右端点的坐标、x_{21} 与 x_{2r} 为当前检测到的线段左、右端点的坐

标,***A*** 为主种子线段,***B***、***C*** 为辅种子线段,如图 3 所示。粗线框为 4 邻接情况下的种子线段,8 邻接时还需加上线段两端细线框表示的那个像素。

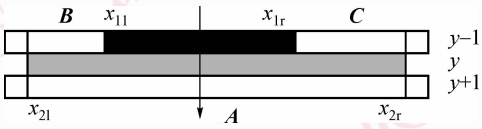


图 3　主、辅种子线段

Fig. 3　The host and assistant seed line segments

根据图 3 可以直接确定种子线段的位置。

线段 ***A*** 端点:$(x_{2l}-N,\ y+1)$、$(x_{2r}+N,\ y+1)$

线段 ***B*** 端点:$(x_{2l}-N,\ y-1)$、$(x_{11}-1,\ y-1)$

线段 ***C*** 端点:$(x_{1r}+1,\ y-1)$、$(x_{2r}+N,\ y-1)$

其中,4 邻接下 N 为 0,8 邻接下 N 为 1。向上搜索时主、辅种子线段中 y 值增量的正负号则与图 3 相反。当辅种子线段的左端点坐标大于右端点坐标时此线段不存在。即当 $x_{2l}-N>x_{11}-1$ 时没有线段 ***B***;当 $x_{1r}+1>x_{2r}+N$ 时没有线段 ***C***。

3.2.3　新线段的检测

种子线段是种子像素的集合,它们是搜索的出发点,新线段也可能超出它的范围。同时,由同一种子线段也可能搜索得到多条新线段,如图 4(b)所示。

图 4　种子线段及其检测结果

Fig. 4　The seed line segment and detected result

在检测新线段时,种子线段无主、辅之分,它们的处理方法是相同的。检测新线段的关键是其左端点的确定。新线段的检测过程如下:

(1)检查种子线段左端点是否是对象点,若是则分别向左、向右搜索新线段的左、右端点,如图 4(b)中的 ***D***。否则转过程(2)。

(2)向右搜索新线段的左端点,左端点的搜索范围不超出种子线段。找到新线段的左端点后,继续向右搜索它的右端点,如图 4(b)中的 ***E***。右端点的搜索范围则不受种子线段的限制,如图 4(b)中的 ***F***。

(3)重复过程(2)直到检测点超出种子线段范围。

3.2.4　描述种子线段的数据结构

由种子线段的确定方法可知,种子线段与新搜索得到的线段并不完全一致,但它们之间存在密切的关联。利用线段表结构可以很好地将它们结合在一起。此时标记 Label 同时表示新检测得到的线段与主种子线段,用 0 值标记来表示辅种子线段。用标志 Flag 来表示种子线段与当前线段 Y 方向上的坐标差,它由搜索方向决定,向下搜索时取 +1,向上搜索时取 −1,此时坐标差兼表方向。以图 3 向下搜索为例,主、辅种子线段的线段表表项分别为

线段 ***A*** 表项:$L,\ x_{2l},\ x_{2r},\ y,\ +1$

线段 ***B*** 表项:$0,\ x_{2l},\ x_{11},\ y,\ -1$

线段 ***C*** 表项:$0,\ x_{1r},\ x_{2r},\ y,\ -1$

端点调整值 N 可在处理过程中补上。

在整个填充过程中,只有初始化时输入的种子点所在的主、辅种子线段的长度相等,即

线段 ***A*** 表项:$L,\ x_{11},\ x_{1r},\ y,\ +1$

线段 ***B*** 表项:$0,\ x_{11},\ x_{1r},\ y,\ -1$

种子填充结束,清除线段表中的所有辅种子线段表项,留下的部分就是区域的线段表。

4　测试结果及效率分析

测试图像如图 5 所示,分辨率为 752 × 560,图中有 68 个形状复杂、多孔、大小不等、有许多长行程的区域。

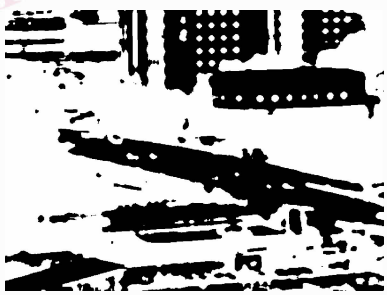


图 5　测试图像(黑白反相)

Fig. 5　The test image(black-white reversal)

算法 1 为线段编码法,作为参照的传统种子填充算法采用行程分析标记方法分 3.1.2 的堆栈法(算法 2)和 3.1.3 的队列法(算法 3)两种情况。算法 4 为种子填充快速算法。作为参照的轮廓跟踪填充方法采用文献[8]、[9]提出的轮廓跟踪和轮廓填

充算法(算法 5),此算法基于种子填充中的边界填充算法,它以区域边界的左端点作为种子点。区域边界由轮廓跟踪得到,得到轮廓后先绘制每一行程的右端点,然后从左端点出发填充到右端点,它是目前轮廓填充方面处理效率最高的算法之一。各算法在不同类型计算机上的运行时间如表 2 所示。

从表中可以看出,线段编码与种子填充快速算法优于轮廓填充,运行时间都减少了近一半;传统种子填充法不及轮廓填充,运行时间约为后者的一倍;种子填充快速算法较传统算法速度提高 3 ~ 5 倍,这是因为每条线段可减少一半扫描时间,再算上每个层次的"复利"关系,故总的运行时间可少于原来的 1/3。种子填充传统算法中队列法又优于堆栈法,不仅速度略快,还可得到区域的线段表。

同时,线段编码除了步骤(1)是在图像上进行外,后面步骤都是在线段表上进行的,因此区域并非一定要做填充,这样实际效率还可以更高些。

影响处理效率的主要因素是重复操作。如算法 2、3 没有利用上下线段的"遮掩"作用,不分情况在线段的上下做同样的扫描,而算法 5 为逐点填充,且内孔需填充两次。

计算机的运行效率不仅与软件,即采用的算法、操作次数以及选择的编译条件有关。也与硬件条件,如指令集、内存与各级缓存的大小也有很大关系。因此,选择了 3 种不同类型的计算机进行测试。从表 2 可以看出,结论相类似,线段编码与种子填充快速算法都优于轮廓填充算法。并且计算机性能愈好算法效率的提高愈明显,两种快速算法中则线段编码优于种子填充算法。

5 结 论

在图像处理中,常用水平线段集表示区域,本文提出采用线段表结构来表示区域,并给出使用线段表改进传统像素标记与种子填充算法的快速算法。

在算法上提高程序效率的关键,首先是避免重复操作,例如水平与垂直方向的连通检测分别进行,水平方向只需检测单侧邻点的连通性,上下线段的连通性则通过比较两条线段的端点坐标来确定。其次,在线段编码中利用桶排序减少排序过程中的大量数据交换,在种子填充快速算法中利用队列结构并避免重复扫描来提高效率。

本文提出的两种快速算法中,种子填充快速算法适用于指定(单个)区域的提取,线段编码则适用于全部区域的标记。同时,利用线段表表示区域可以参与区域的各种运算,例如逻辑运算和形态学运算[5]等。由此可见,线段表在图像处理中是一个如同链码一样有效的数据结构。

致 谢 任明武教授提供了相关的论文、演示程序与测试图像,深受启发,特致谢意。

参考文献(References)

1 White E. GDI + Programming Design [M]. Beijing: Tsinghua University Press, 2002:141-159. [White E. GDI + 程序设计[M]. 北京:清华大学出版社,2002: 141-159.]

2 Castleman K R. Digital Image Processing [M]. Beijing: Publishing House of Electronics Industry, 1998:480-481.

3 Zhang Yu-jin. Image Engineering—Image Processing and Analyzing [M]. Beijing: Tsinghua University Press, 1999:205-207. [章毓晋. 图象工程(上册):图象处理和分析[M]. 北京:清华大学出版社,1999: 205-207.]

4 Lu Zong-qi. C/C ++ Programming of Image Processing [M]. Beijing: Tsinghua University Press, 2005:319-360. [陆宗骐. C/C ++ 图像处理编程[M]. 北京:清华大学出版社,2005: 319-360.]

5 Lu Zong-qi, Zhu Yu. Fast algorithm of line segment encoding and seed filling [A]. In: Proceedings of the 13th ICIG [C], Beijing: Tsinghua University Press, 2006:306-311. [陆宗骐, 朱煜. 数学形态学腐蚀膨胀运算的快速算法[A]. 第十三届全国图像图形学学术会议论文集[C],北京:清华大学出版社,2006: 306-311.]

6 Tamura H. Computer Image Processing (in Chinese) [M]. Beijing: Science Press, 2004: 127-129. [田村秀行. 计算机图像处理[M]. 北京:科学出版社,2004:127-129.]

7 Rogers D F. Procedural Elements for Computer Graphics (Second Edition) [M]. Beijing: China Machine Press, 2002: 73-106. [Rogers D F. 计算机图形学的算法基础(第 2 版)[M]. 北京:机械工业出版社,2002: 73-106.]

8 Ren M, Yang J, Sun H. Tracing boundary contours in a binary image [J]. Image and Vision Computing, 2002, **20**(2): 125-131.

9 Ren M, Yang W, Yang J. A new and fast contour-filling algorithm [J]. Pattern Recognition, 2005, **38**(12): 2564-2577.