一种基于缝隙码的区域填充算法

陈优广 顾国庆 王 玲
(华东师范大学信息科学技术学院,上海 200062)

摘 要 提出了一种基于缝隙码的区域填充算法。给出了单条缝隙码的填充算法,及多连通区域或整幅图像的快速填充算法,能填充任意复杂图像区域,对多连通区域或整幅图像填充时,算法只对图像区域填充,不用对区域外或区域内部的孔洞进行填充,对非二值图像,该算法不需要辅助内存空间。实验结果表明,对比现有的算法,本文算法具有速度快、效率高等优点。

关键词 缝隙码 区域填充 填充算法
中图法分类号: TP391.41 **文献标识码**: A **文章编号**: 1006-8961(2007)11-2086-07

A Region Filling Algorithm Based on Crack Chain Code Description

CHEN You-guang, GU Guo-qing, WANG Ling
(*School of Information Science and Technology, East China Normal University, Shanghai* 200062)

Abstract The paper proposed a region filling algorithm based on crack chain code description. The algorithm of single crack chain code and a fast filling algorithm about the complicated connecting region or the global image are given, in which only the image regions is filled. Compared with the conventional algorithms, this algorithm requires no storage and works well for any complex regions. Experimental results prove that it is faster and more efficacious.

Keywords crack chain code, region filling, filling algorithm

1 引 言

区域填充是计算机图形学的基本问题之一。基于链码的填充是图像图形处理的基本算法,且广泛应用于图像处理、目标分析、图像压缩及计算机图形学中。因此区域填充一直是人们研究的热点。

传统的填充算法主要有奇偶性检测[1]和种子填充[1,2]两大类。种子填充的缺点是需要栈结构,因而需要较大的存储空间以实现栈结构,而且在有多个对象需要填充时,种子点的逐个选取会降低效率,在某些复杂情况下,种子点的选择非常困难;在奇偶检测法中,由于交叉点可能为多重交点,会导致在统计交叉点数目时,容易出现错误,而且水平直线状的边缘线也会导致错误判断。

链码最早是由 Freeman 提出,分为 4 方向 Freeman 链码和 8 方向 Freeman 链码,随着链码的广泛应用,很多文献提出了当轮廓是以链码形式给出时的填充算法[3-9],且它们都能取得比传统算法正确而快速的填充结果,分析被填充区域轮廓的 8 方向链码编码属性,并将轮廓点分成孤立点、标志点和忽略点、不存在点,从而很好地解决了删除孤立点或将一个孤立点当作两个点的问题,这样就能避免产生错误的线段编号,从而取得正确的填充结果。

Chang 等人设计了一种从光栅表示到 Freeman 缝隙码表示的算法[10],本文简称之为缝隙码。它不同于 4 方向 Freeman 链码,Freeman 缝隙码记录区域边界像素边的 4 方向码值。虽然 Freeman 链码与 Freeman 缝隙码可以相互转换[11],但是把缝隙码转换为 Freeman 链码还是需要花费时间,所以研究基于缝隙码的填充算法有理论意义和实用价值的。为此给出了一种基于缝隙码的区域填充算法,给出单

收稿日期:2006-06-30;改回日期:2006-08-22

第一作者简介:陈优广(1971 ~),男,讲师。华东师范大学信息科学技术学院系统分析与集成专业博士研究生。主要研究方向为图像处理与模式识别。E-mail: ygchen@cc.ecnu.edu.cn

条缝隙码的填充算法,及多连通区域或整幅图像的快速填充算法,算法能自动选取种子。对比现有的算法,该算法不需要辅助内存空间,能填充任意复杂图像区域,尤其对多连通区域或整幅图像的填充时,不存在孔洞的重复填充问题。实验结果表明,本文算法具有速度快、容易实现等优点。

2　现有的基于链码的填充算法分析

CAI 最早给出了利用链码进行填充的算法,提出了对轮廓点按链码属性进行分类,来解决奇偶性填充算法存在的问题[3],算法用到了一个与图像等大的辅助内存,需要扫描图像中的每一个像素点。后来 Chang 和 Frank 提出了一种快速、低内存消耗的算法[4,5],算法较 CAI 算法提高了效率,然而其需用一个表来记录孤立点和标记点,对表的排序花费了大量的时间。Tang 则给出离散格伦定理[12]和基于此定理的填充算法[6],通过计算从某一点出发的整条轮廓线的积分,来判断该点是否是区域内部点,令像素 t 的坐标为 (x_t,y_t),那么判断 t 在区域内部或外部的计算公式如下:

$$\sum_{(x_p,y_p)\in L}\Delta(x_p-x_t,y_p-y_t)\times \mathrm{sign}(x_p,y_p)=\begin{cases}0 & (x_t,y_t)\text{ 在区域的外部}\\ 1 & (x_t,y_t)\text{ 在区域的内部}\end{cases}$$

其中,L 是边界点的有序集合。符号函数 $\mathrm{sign}(x,y)$ 根据点 (x,y) 的进出链码,其取值由表 1 给出,其中,-1 表示左端点,1 表示右端点。$\Delta(x,y)=\begin{cases}1 & x>0\text{ 且 }y=0\\ 0 & \text{其他}\end{cases}$。但 Tang 采用的是种子填充算法,需要很大的计算量。

Ren 给出了一种把链码分解成简单子链的填充方法[7],为了解决由于轮廓不连续光滑造成的一些点被跟踪多次的问题,该算法把链码分解成多条简单子链,通过简单子链的填充给出了整条链的填充算法,但算法没有给出区域内轮廓的填充算法,也无法同时填充多条轮廓。

Ren 最近又给出了一种新的快速的基于 Freeman 链码的填充算法[8],并在该文中对前 4 种方法的不足给出详尽描述,它提出了一种新的自动选取种子的方法,通过统计进出轮廓点的链码的纵坐标和来判断该点是填充起点、终点或是跳过点。用 sum 表示进出轮廓点的链码的纵坐标和。当

表 1　TANG 左右端点表

Tab.1　TANG_LUT

c_i	c_{i+1}							
	0	1	2	3	4	5	6	7
0	0	1	1	1	1	×	×	0
1	0	1	1	1	1	0	×	0
2	×	1	1	1	1	0	0	×
3	×	1	1	1	1	0	0	0
4	-1	×	×	0	0	-1	-1	-1
5	-1	0	×	0	0	-1	-1	-1
6	-1	0	0	×	×	-1	-1	-1
7	-1	0	0	0	×	-1	-1	-1

注:c_i,c_{i+1} 为相邻的两个缝隙码组合,× 为不存在的组合。

sum > 0 时,轮廓点为填充起点;当 sum < 0 时,轮廓点为填充终点;当 sum = 0 时,轮廓点为填充跳过点。该方法把种子填充和奇偶填充相结合,其思想是发现种子,按奇偶填充法进行填充,对每条链码的填充,算法复杂度为 $a+O(n)$,其中,a 为图像区域面积,n 为图像边界的长度。目前为止文献[10]中 Ren 的算法是最优的。

但是 Ren 的算法在对多连通区域的填充时,存在对洞的重复填充问题,导致时间的浪费,影响了效率,如图 1 所示。

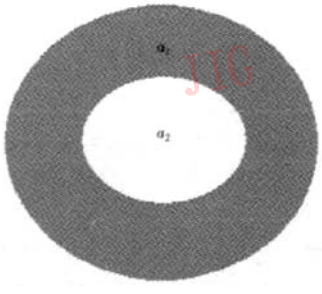

图 1　多连通区域

Fig.1　Complicated connecting region

令 n_1、a_1 表示区域的外边界和区域面积,n_2、a_2 表示区域的内边界和洞的面积,那么利用 Ren 的算法,需要分别对区域的外边界和内边界填充,对每一条区域边界,Ren 的算法要遍历边界点 4 遍,区域点一遍,如图 1 所示,需要填充的点为

$$4n_1+a_1+a_2+4n_2+a_2=4(n_1+n_2)+a_1+2a_2$$

本文给出了一种基于缝隙码的填充算法,针对多连通区域或整幅图像的填充时,给出了只需对图像区

域像素填充的快速算法，不需要对区域外（包括孔洞）像素进行填充，算法很好地解决了多连通区域洞的重复填充问题。

3 区域边界的缝隙码表示及其性质

按逆时针或顺时针方向沿着图像边界像素的边行走一周，依次记录其方向码，就得到图像边界的缝隙码，加上起始点坐标，边界就可以用缝隙码唯一确定下来。除非特殊说明，本文给出的缝隙码都是按逆时针方向行走的。

如图 2 所示，像素顶点坐标在 u-v 坐标系取值为整数，像素坐标在 x-y 坐标系下取整数，u-v 坐标系可由 x-y 坐标系向左向上平移半个像素获得。图 2 所示的区域边界的缝隙码可表示为 $\{(u_0,v_0)$ 32323230300300100101121232211 2$\}$。(u_0,v_0) 是区域边界像素的顶点 P 在 u-v 坐标系下的坐标。

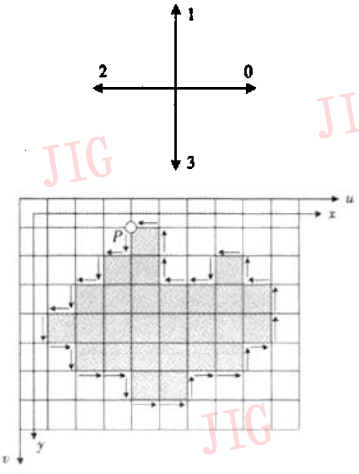

图 2 4 方向码与区域轮廓的缝隙码

Fig. 2 Four directions code and crack code of contour

缝隙码记录的是封闭的区域边界的 4 方向码序列，令 S_0、S_1、S_2、S_3 分别表示缝隙码中码值为 0、1、2、3 的个数，那么有 $S_0=S_2$，$S_1=S_3$。由区域边界的连通性原理，有以下性质成立。

性质 用水平直线与区域边界相交，令 M_1 与 M_3 分别表示水平直线与区域边界缝隙码相交的码值为 1 和 3 的个数，那么有 $M_1=M_3$，且方向码 1 和方向码 3 是交互出现的。

与 Freeman 链码不同，缝隙码记录的是经过边界像素边的方向码，起始点为边界像素的顶点。表 2 定义了像素顶点坐标随方向码变化的坐标偏移量。

表 2 坐标偏移表 OFFSET_LUT

Tab. 2 OFFSET_LUT

c	0	1	2	3
du	1	0	−1	0
dv	0	−1	0	1

注：c 为缝隙码，du、dv 为 c 从当前位置迁移到下一个位置的坐标增量。

已知起始点和缝隙码，根据表 2 给出的坐标偏移表 OFFSET_LUT，可以很容易获得区域边界像素顶点坐标。像素坐标值与其左上顶点在 u-v 坐标系下的坐标相等。令 (x,y) 为边界像素坐标，(u,v) 为该像素的一个顶点坐标，e 为以 (u,v) 为始点的方向码，那么像素坐标 (x,y) 与 (u,v) 和方向码 e 的关系 $(x,y)=f(u,v,e)$ 如下：

当 $e=0$ 时，$(x,y)=(u,v-1)$

当 $e=1$ 时，$(x,y)=(u-1,v-1)$

当 $e=2$ 时，$(x,y)=(u-1,v)$

当 $e=3$ 时，$(x,y)=(u,v)$

缝隙码表示的边界轮廓可以看作是一个多边形，其角度变化可由表 3 给出。

表 3 链码角度变化表 Angle_LUT(A)

Tab. 3 Angle_LUT(A)

c_i	c_{i+1}			
	0	1	2	3
0	0	90	×	−90
1	−90	0	90	×
2	×	−90	0	90
3	90	×	−90	0

令

$$S_A=A[c_{n-1}][c_0]+\sum_{i=0}^{n-1}A[c_i][c_{i+1}]$$

其中，$A[i][j]$ 由表 3 给出。

如果 $S_A=360$，那么轮廓为区域外边界，否则轮廓为内边界。

4 基于缝隙码的区域填充

只考虑经过边界像素的左右边的方向码。那么有 3 种可能，如图 3。

定义 若像素的左边有方向码 3 经过且其右边

图 3　边界像素与左右缝隙码的 3 种位置关系

Fig. 3　Three types of position of crack code and contour pixel

没有方向码经过,称该像素为 L_像素,若像素的右边有方向码 1 经过且其左边没有方向码经过,称该像素为 R_像素,若边界像素的左右两边都有方向码经过,那么称像素为 LR_像素。如图 3 所示,标记 L 的为 L_像素,标记 R 的为 R_像素,标记 LR 的为 LR_像素。

令 $P(x,y)$ 为区域边界上的像素点,c_P 为经过该像素某边的方向码。

If $c_P=3$ then 点 P 为 L_像素

If $c_P=1$ then 点 P 为 R_像素

If 点 P 为 L_像素 and 点 P 为 R_像素

then 点 P 为 LR_像素。

定理 1　用一水平直线与区域边界相交,那么水平直线与区域边界相交的 L_像素和 R_像素个数相等,且 L_像素和 R_像素相互间隔出现。

证明　用水平直线与区域边界相交,由性质,缝隙码与该直线相交的方向码 1 和方向码 3 的个数相同,且是交互出现的,由定义 1,除去 LR_像素,剩下的方向码 1 和方向码 3 就分别对应着边界的 L_像素和 R_像素,且交互出现。

如图 4 所示,虚线与区域边界相交的 L_像素与 R_像素个数相等,且交互出现。

由定理 1,L_像素可以看作填充种子点,L_像素和 R_像素又满足奇偶填充原理。对于单条缝隙码的填充算法可以描述为:首先遍历缝隙码,确定边界的 L_像素和 R_像素,并计算围线角度,确定围线方向;然后对所有的 L_像素,作为填充起始点,对区域外边界,从左向右填充,对区域内边界从右向左填充,直到 R_像素结束;最后填充边界像素。算法需要扫描边界像素顶点 3 遍,区域内部点一遍。

定义围线缝隙码类:

```
struct Crackchain{
  int u0,v0;      //缝隙码的起始点
  long n;         //缝隙码的长
  int * c;        // 缝隙码
}
```

算法 1:单链的填充

sampleChainFill(img,w,h,ck, col)

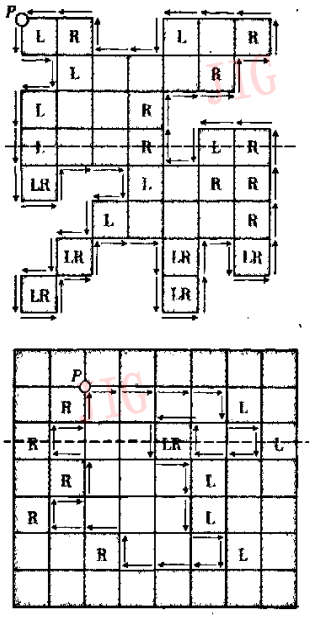

图 4　单条轮廓的左右边界像素示意图

Fig. 4　A demo of left and right pixel of single contour

其中,img 为填充图像,w 和 h 分别为图像的宽和高,ck 为缝隙码类,col 为填充色。

(1) 用 l,r,lr 分别表示边界上的 L_像素、R_像素和 LR_像素,定义中间变量,并初始化。用 S_A 来统计链码角度变化,$S_A=0$。

(2) 遍历链码,标记边界的左右像素,并判断其围线方向

```
u=ck.u0;v=ck.v0;
pc=ck.c[n-1];
for(i=0;i<ck.n;i++){
  nc=ck.c[i];
  S_A+=A[pc][nc]; //统计链码角度变化
  (x,y)=func(u,v,nc); //坐标变换
    if(nc==3){//标记边界上的左右像素
    if(img[x][y]==r) img[x][y]=lr;
      else img[x][y]=l;}
  if(nc==1){
      if(img[x][y]==1) img[x][y]=lr;
        else img[x][y]=r;}
    pc=nc;
    u=u+du[nc];v=v+dv[nc];
    }
  if(S_A>0)flg=1; else flg=-1;
```

这里 flg=1 时,边界为区域外边界,否则为内边界。$A[i][j]$ 的值由表 3 给出,函数 (x,y)=func(u,

v,nc)见 3 节定义,以下同。

(3) 填充区域内部像素

```
u = ck. u0;v = ck. v0;
for(i = 0;i < ck. n;i + + ){
  nc = ck. c[i];
  (x,y) = func(u,v,nc);
   if(img[x][y] = = l){
     j = x;
     while(img[j][y]! = r){
     mg[j][y] = col;
     j + = flg;
   }
   u = u + du[nc];v = v + dv[nc];
 }
```

(4) 填充边界像素

```
u = ck. u0;v = ck. v0;
for(i = 0;i < ck. n;i + + ){
  nc = ck. c[i];
  (x,y) = func(u,v,nc);
  img[j][y] = col;
u = u + du[nc];v = v + dv[nc];
}
```

对整幅图像填充时,如果图像区域都是单连通区域,算法 1 就能很好地填充,但是如果图像中存在多连通区域,边界存在包含关系,如果用算法 1 填充,存在洞的重复填充问题。另外,对于复杂的图像,当获得的边界缝隙码混杂时,利用算法 1 填充还会导致填充混乱。考虑到上面的问题,下面给出多连通区域或整幅图像的填充算法,算法只需对区域内像素填充,对区域外像素(包括孔洞)不进行填充,且对单连通区域也适用。

定理 2 用水平直线与区域相交,那么水平直线与区域所有边界相交的 L_像素和 R_像素个数相等,且 L_像素和 R_像素相互间隔出现。

证明 定理 2 可以看作定理 1 的推广,考虑区域边界所有缝隙码,经过边界像素的左右边的缝隙码仍为如图 3 所示的 3 种类型,由于所有边界的缝隙码是不相交的,由性质,水平直线与所有缝隙码相交的方向码 3 和方向码 1 的个数相等,且是交互出现的,除去 LR_像素,剩下的方向码 3 和方向码 1 决定了边界上的 L_像素和 R_像素,所以定理成立,如图 5 所示。

下面给出多连通区域或整幅图像的填充算法。其基本思想是:首先遍历所有链码,确定边界像素的

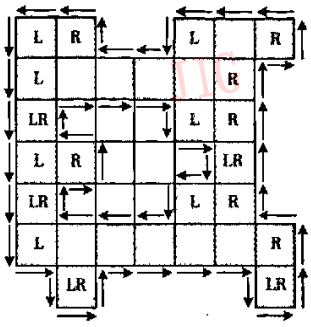

图 5 多连通区域的左右边界像素示意图

Fig. 5 A demo of left and right pixel of complicated connecting region

L_像素和 R_像素;然后,遍历所有区域边界像素,如果像素为 L_像素,那么从左向右填充,直到遇到 R_像素结束,填充图像区域内像素点;最后,填充所有区域边界像素。

算法 2 多连通区域或整幅图像的填充算法

multiChainFill(img,w,h,ca[ ],m,col)

其中,img 为填充图像,w 和 h 分别为图像的宽和高,ca 为缝隙码类数组,col 为填充色。

(1) 遍历所有区域边界,标记所有边界的 L_像素和 R_像素。

```
for(k = 0;k < m;k + + ){
  ck = ca[k];
  u = ck. u0;v = ck. v0;
  for(i = 0;i < ck. n;i + + ){//遍历链码
    nc = ck. c[i];
    (x,y) = func(u,v,nc);
  if(nc = = 3){
    if(img[x][y] = = r)img[x][y] = lr;
    else img[x][y] = l;}
  if(nc = = 1){
    if(img[x][y] = = l)img[x][y] = lr;
      else img[x][y] = r;}
  u = u + du[nc];v = v + dv[nc];
  }
}
```

(2) 遍历所有区域边界,选择 L_像素为填充起始点填充图像区域

```
for(k = 0;k < m;k + + ){
  ck = ca[k];
  u = ck. u0;v = ck. v0;
  for(i = 0;i < ck. n;i + + ){
    nc = ck. c[i];
```

```
    (x,y) = func(u,v,nc);
      if(img[x][y] == =1){j = x;
      while(img[j][y]! = r){
        img[j][y] = col; j + = 1;}
      }
      u = u + du[nc];v = v + dv[nc];
    }
  }
```

(3) 给所有边界点填色

```
for(k = 0;k < m;k + + ){
  ck = ca[k];
  u = ck. u0;v = ck. v0;
  for(i = 0;i < ck. n;i + + ){
    nc = ck. c[i];
    (x,y) = func(u,v,nc);
    img[j][y] = col;
    u = u + du[nc];v = v + dv[nc];
  }
}
```

利用算法 2 对多连通区域填充时,只需遍历区域外边界和内边界 3 遍,区域内部点一遍。对比算法 1 和 Ren 的算法,明显提高了速度。

5　实验结果

本文算法与文献[9]中 Ren 的填充算法通过实验进行了比较,表 4 给出了算法性能比较。表 5 给出了不同类型图像的填充时间。

表 4　算法性能比较表

Tab. 4　Theoretical algorithm performance

算法	填充点数
Ren 算法	4 × 边界点 + 区域面积 + 2 × 洞面积
算法 1	3 × 边界点 + 区域面积 + 2 × 洞面积
算法 2	3 × 边界点 + 区域面积

表 5　测试图像填充时间比较

Tab. 5　The performance time comparison by testing image

单位:ms

图号	图像尺寸	Ren 算法	算法 1	算法 2
图 6	272 × 265	40	30	30
图 7	315 × 346	40	40	20
图 8	360 × 477	190	180	160
图 9	520 × 488	190	200	150
图 10	679 × 450	110	100	90
图 11	921 × 401	271	240	131

由表 4 和表 5 可以看出,算法 1 虽然边界点少遍历一遍,但区域边界的缝隙码个数比 Freeman 链码个数多,算法 1 和算法 2 花费时间基本上接近,由于算法 2 不对多连通区域的洞填充,明显提高了填充速度。

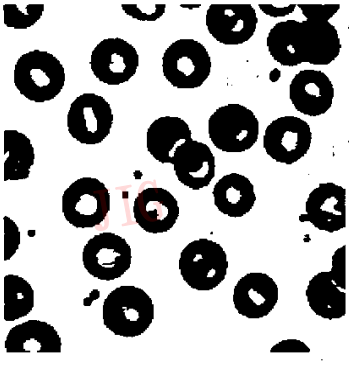

图 6　细胞图像填充例

Fig. 6　Filling a cell image

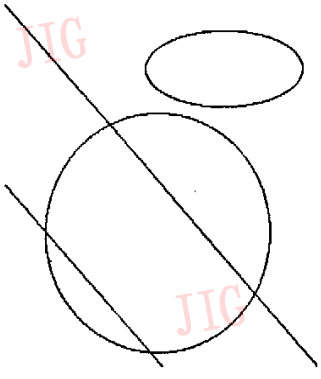

图 7　线图像填充例

Fig. 7　Filling a line image

图 8　龙图像填充例

Fig. 8　Filling a dragon image

图 9 人物图像填充图例

Fig. 9 Filing a figure image

$$\mathbf{H}=\left(\mathbf{K}\left(\mathbf{R}-\frac{\mathbf{t}\mathbf{n}_{\pi_1}^{\mathrm{T}}}{d_{\pi_1}}\right)\mathbf{K}^{-1}\right)^{-1}\left(\mathbf{K}\left(\mathbf{R}-\frac{\mathbf{t}\mathbf{n}_{\pi_2}^{\mathrm{T}}}{d_{\pi_2}}\right)\mathbf{K}^{-1}\right). \tag{6}$$

being $\mathbf{n}_{\pi_1}$, $\mathbf{n}_{\pi_2}$ the normals and d_{π_1}, d_{π_2} the distances to the planes.

Using the Sherman–Morrison formula [12], in a similar way than in Ref. [13], it turns out that

$$\left(\mathbf{R}-\frac{\mathbf{t}\mathbf{n}_{\pi_1}^{\mathrm{T}}}{d_{\pi_1}}\right)^{-1}=\mathbf{R}^{-1}+\frac{(\mathbf{R}^{-1}\mathbf{t})\left((\mathbf{n}_{\pi_1}^{\mathrm{T}}/d_{\pi_1})\mathbf{R}^{-1}\right)}{1+\mathbf{n}_{\pi_1}^{\mathrm{T}}\mathbf{R}^{-1}\mathbf{t}}. \tag{7}$$

图 10 文档图像填充例

Fig. 10 Filling a document image

时间 / 教室 / 节次 / 星期	一	二	三	四	五	六	七	八	晚
一									
二									
三							数字图象变换 理 610		
四									
五									

图 11 表格图像填充例

Fig. 11 Filling a table image

6 结 论

本文给出了一种基于缝隙码的填充算法，利用种子填充和奇偶填充相结合，根据经过像素边的缝隙码，确定填充的开始像素和终止像素，给出了单条链码的填充算法，还给出了多连通区域或整幅图像的填充算法，对比现有的基于链码的填充算法，本文给出的算法在填充多连通区域时只需填充区域内像素，不用填充孔洞区域，明显提高了效率，对非二值图像，算法不用辅助内存。实验结果表明，本算法具有容易实现、速度快等优点，所以本算法具有较广泛的用途。

参考文献(References)

1 Pavlidis T. Filling algorithms for raster graphics [J]. Computer Graphics Image Process, 1979, 10(2): 126 ~ 141.

2 Pavlidis T. Algorithms for Graphic and Image Processing [M]. Rockville, MD: Computer Science Press, 1982.

3 Cai Zu-guang. Restoration of binary images using contour direction chain codes description[J]. Computer Vision, Graphics, and Image Processing, 1988, 41(1): 101 ~ 106.

4 Chang Long-wen, Leu Kuen-long. A fast algorithm for the restoration of images based on chain codes description and its application[J]. Computer Vision, Graphics, and Image Processing, 1990, 50(3): 296 ~ 307.

5 Frank Y Shih, Wai-Tak Wong. An improved fast algorithm for the restoration of images based on chain codes description[J]. Computer Vision, Graphics, and Image Processing, 1994, 56(3): 348 ~ 351.

6 Tang G Y. Region Filling with the use of the discrete Green theorem [J]. Computer Vision, Graphics, and Image Processing, 1988, 42(3): 297 ~ 305.

7 Ren Ming-wu, Yang Jing-yu, Sun Han. A new contour filling algorithm based on Chain codes description[J]. Journal of Image and Graphics, 2001, 6(4): 348 ~ 352. [任明武,杨静宇,孙涵. 一种新的基于链码描述的轮廓填充方法[J]. 中国图象图形学报, 2001, 6(4): 348 ~ 352.]

8 Ren Ming-wu, Yang Wan-kou, Yang Jing-yu. A new and fast contour-filling algorithm[J]. Pattern Recognition, 2005, 38(12): 2564 ~ 2577.

9 Li Hua, Zhu Guang-xi, Zhu Yao-ting. A method for restoration of binary image based on Chain coding [J]. Journal of Image and Graphics, 2000, 5(6): 474 ~ 478. [李华,朱光喜,朱耀庭. 一种利用方向链码重建二值图像的新方法[J]. 中国图象图形学报, 2000, 5(6): 474 ~ 478.]

10 Chang Fu, Chen Chun-jen, Lu Chi-jen. A linear-time component-labeling algorithm using contour tracing technique [J]. Computer Vision and Image Understanding, 2004, 93(2): 206 ~ 220.

11 Chen You-guang, Zhang Wei, Gu Guo-qing. Transformations among several chain codes in square lattice[J]. Mini-Micro Systems, 2005, 26(12): 2190 ~ 2193. [陈优广,张薇,顾国庆. 矩形点阵上链码的转换算法[J]. 小型微型计算机系统, 2005, 26(12): 2190 ~ 2193.]

12 Tang G Y. A discrete version of green's theorem [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1982, 4(3): 242 ~ 250.