基于边界相邻三点的区域遍历算法

谭明金
(解放军理工大学工程兵工程学院计算机室，南京 210007)

摘 要 首先基于边界上当前像素与前后两个相邻像素的坐标关系，定义并构造出边界像素与连通闭区域内，某些像素之间的联系，然后通过枚举各边界像素类型及其特点分析，提出了一种用于判定，并找出区域内与当前边界像素具有这种联系的像素系列，进而遍历出任意连通闭区域(像素边界线可为任意复杂曲线的单连通或者复连通闭区域)的像素遍历算法. 该算法在适应性及时间与空间性能等许多方面均很好地满足了诸如任意连通闭区域的填充和点在各种复杂区域的判定与跟踪等方面的应用需要，同时，它对闭区域像素的描述方法也为开展如何更有效地表示闭区域的研究提供了一种有益的参考.

关键词 计算机图形学(520·6030) 区域遍历 区域表示 区域填充
中图法分类号：TP301.6 **文献标识码**：A **文章编号**：1006-8961(2003)03-0322-06

An Algorithm for Ransacking Region Connection Based on Three Borderling Pixels

TAN Ming-jin
(*Computer Section, Engineering Institute of Engineering Corps, PLA Univ. Of Sci. & Tech., Nanjing* 210007)

Abstract Based on the coordinate relationship between the current borderline pixel and its two lockstep borderline pixels, a certain connection by which some pixels on the region connect with the current borderline pixel is defined and constructed here. On the basis of enumerating all the types of the borderline pixels and analyzing their features, an algorithm that ransacks any close region connection by deciding and finding out those pixels in the region corresponding to the current borderline one with that connection, is concluded. Any close region connection means that the close region connection can be either single connectivity one or complexity connectivity one with any borderline pixel curves. That is, the algorithm can deal with almost all close regions in a consistent way. So it has good applicability. Secondly, it gets higher performances in both time and space due to its particular judging rules, retrieving processes and data describing method. Additionally, the operations mainly converge on the integral comparison operation, and the algorithm is simple. In a word, the algorithm meets the region-concerned needs such as region filling, point domain juding, point domain tracking, and so forth. Simultaneously, due to the region pixels describing method, it presents a helpful use for reference for researching how to describing close region more effectively.

Keywords *Ransacking region, Describing region, Filling region*

0 引 言

二维图形在光栅化或称像素化[1](区域填充)之前，必须确定与区域对应的像素集(即区域遍历). 目前，区域遍历主要有基于扫描线边界求交或基于像素种子生长的两种基本算法类型[1]. 其中基于扫描线求交的区域遍历算法(以后简称扫描线求交算法)，因要进行交点计算、排序与判断等序列处理，故算法复杂，且性能差；而基于种子生长的区域遍历算法(以后简称种子算法)，又需要事先给定一个处在区域内的像素种子，才能对区域边界内外的像素进行区分，这不仅使随后的搜索过程很复杂，而且尤其使存储区域内的像素需要很大的空间.

本文提出一种利用边界相邻 3 个像素的关系，逐个判定，并检索出与所有边界像素对应的区域像

收稿日期：2002-04-18；改回日期：2002-10-09

素遍历的算法. 比较而言，这种算法显得简单，性能也比较好.

1　定义与记号

这里是在默认显示屏坐标系(以屏幕左上角为原点，水平向右和垂直向下分别为 x 轴和 y 轴正向的直角坐标系)的基础上讨论像素坐标的.

定义 1　边界[2]上当前像素记为 P_c，沿边界行进，前进方向上紧接当前像素的像素记为 P_f，后退方向上紧靠当前像素的像素记为 P_h. 约定用 x_{P_k} 与 y_{P_k} 的形式分别表示某像素 P_k 的 x 与 y 坐标值(下标 k 代表 c、f、h、a、b 与 w 等任一种像素类型).

定义 2　对于满足条件 $x_{P_a}>x_{P_b}$ 的两个像素 P_a 与 P_b，称 P_a 处在 P_b 的右位或 P_b 处在 P_a 的左位. 而满足条件 $x_{P_a}=x_{P_b}$ 的两个像素 P_a 与 P_b，互称处在中位.

定义 3　称 P_c 与 P_f 或 P_h 之间的 y 坐标之差为变化趋势，并记 $T_h=y_{P_c}-y_{P_h}$，$T_f=y_{P_f}-y_{P_c}$. 其中，T_f 与 T_h 分别简称为前势或后势. 其取值为负、零或正的趋势分别称为升势、平势或降势.

定义 4　$T_h=0$，且 $T_f=0$ 的像素称为中点. $T_h\neq0$，且 $T_f=0$ 的像素称为首点. $T_h=0$，且 $T_f\neq0$ 的像素称为尾点. $T_h\times T_f<0$ 的像素称为尖点. $T_h\times T_f>0$ 的像素称为普点.

定义 5　称离点 P_c 最近，且 y 坐标相同的左位边界点(不含 P_f 与 P_h)为 P_c 的偶点，记为 P_w. 记 P_s 表示区域中 y 坐标为 y_{P_c}，x 坐标处在 $[x_l,x_{P_c}]$ 区间内的像素系列. 其中，x_l 的取值为：当 P_c 存在偶点时，$x_l=x_{P_w}+1$；否则，$x_l=x_{P_c}$.

定义 6　观察者沿边界前行，离闭区域最近的连通部分(P_c、P_f 与 P_h 除外)总在其左边的行进方向称为正向. 对于单连通闭区域[2]来说，正向是逆时针方向；对于复连通闭区域[2]来说，最外边界的正向是逆时针方向，而内部边界的正向则是顺时针方向.

定义 7　称满足 $|x_{P_a}-x_{P_b}|<2$，且 $|y_{P_a}-y_{P_b}|<2$ 的两个不同像素 P_a 与 P_b 相邻或连续.

2　算法原理与实现

2.1　原　理

算法的基本思路是，通过对边界逐像素建立 P_s 像素系列的表达式，以实现区域遍历. 由此可以看出，算法的实现过程集中于判定，并找出边界像素的偶点. 不失一般性，这里通过研究从任意边界像素出发，正向行进来实现区域遍历的情况.

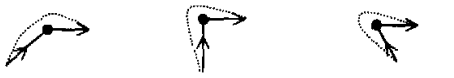
统计表明，按前后势的取值可以将所有边界像素 P_c 区分为中点、首点、尾点、尖点和普点等 5 大类. 进一步依据 P_h 和 P_f 与 P_c 的各种位置关系(包括相邻特性)，可以将每一大类细分为更多的小类. 像素分类的情况及其示意图如图 1 到图 9 所示. 对于这些分类有两点需要说明：其一，这里统一按 P_h 相对于 P_c 的位置、T_h、P_f 相对于 P_c 的位置与 T_f 等 4 个方面的顺序来描述 P_c，同时为叙述方便，还对各种属性进行了简化描述，并且描述时，位置和趋势信息可以同时或者不同时出现. 比如，用“左升右降”表示“P_h 在 P_c 左边、$T_h<0$、P_f 在 P_c 右边，且 $T_f>0$”的 P_c 类型；用“后升前降”表示“$T_h<0$，且 $T_f>0$”的 P_c 类型，而用“左左”表示“P_h 与 P_f 都在 P_c 左边”的 P_c 类型，等等；其二，在每种类型的图形表示中，实心圆点表示 P_c，指向实心圆点的箭头同时表示 P_h 和 T_h，指离实心圆点的箭头同时表示 P_f 和 T_f，虚线表示连通区域距离 P_c 最近的部分(P_c、P_f 与 P_h 除外). 不同类型像素点分类及其特性如下：

(1)中点

(a) 左平右平类　　(b) 右平左平类

图 1　$T_h=0$，且 $T_f=0$ 中点类像素

(2)首点



(a) 左升右平类　(b) 中升右平类　(c) 右升右平类

(d) 左升左平类　(e) 中升左平类　(f) 右升左平类

图 2　$T_h<0$，且 $T_f=0$ 首点类像素

(a) 左降右平类　(b) 中降右平类　(c) 右降右平类

(d) 左降左平类　(e) 中降左平类　(f) 右降左平类

图 3　$T_h>0$，且 $T_f=0$ 首点类像素

(3)尾点

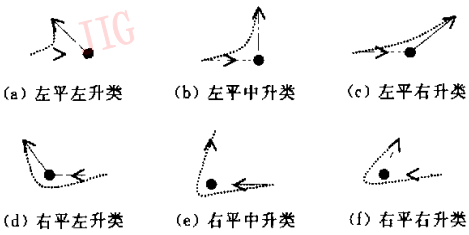


图 4 $T_h=0$,且 $T_f<0$ 尾点类像素

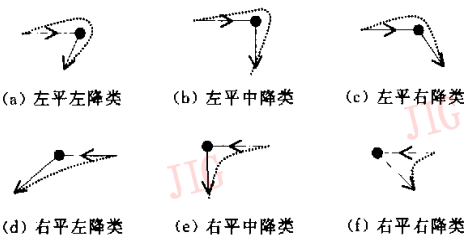


图 5 $T_h=0$,且 $T_f>0$ 尾点类像素

(4)尖点

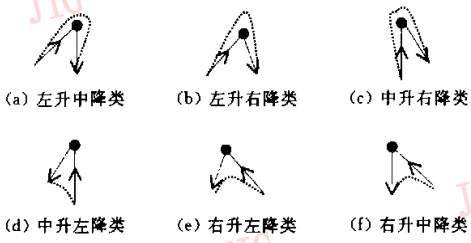


图 6 $T_h<0$,且 $T_f>0$ 尖点类像素

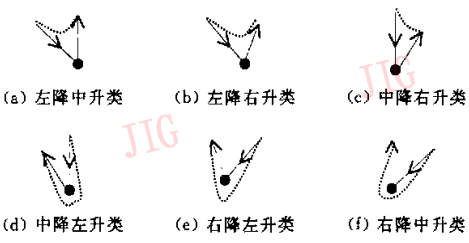


图 7 $T_h>0$,且 $T_f<0$ 尖点类像素

(5)普点

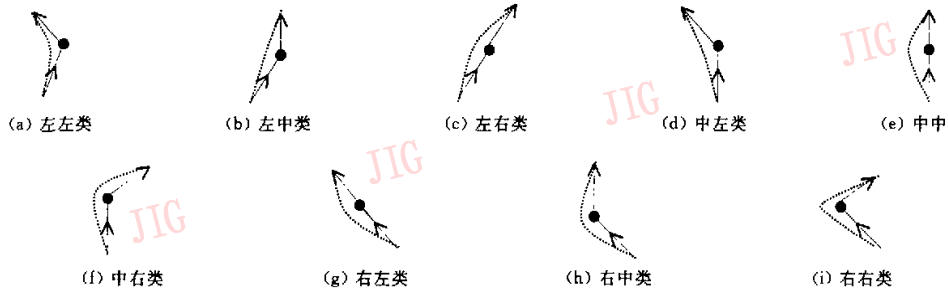


图 8 $T_h<0$,且 $T_f<0$ 普点类像素

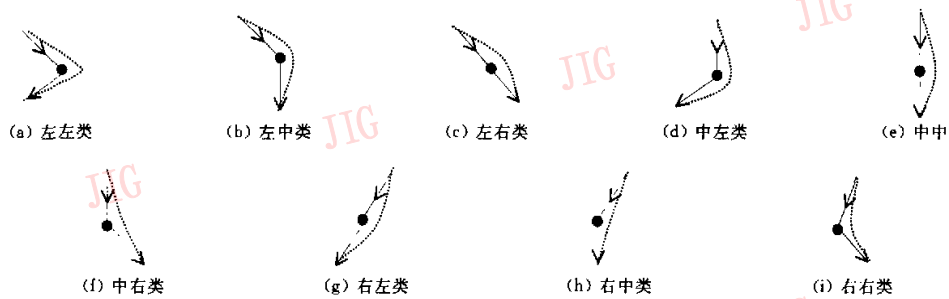


图 9 $T_h>0$,且 $T_f>0$ 普点类像素

从示意图以及根据偶点的定义容易看出,只有那些从面向屏幕的用户观察,在 P_c 左边具有虚线的 P_c,才可能具有偶点.具体地说,只有“后升右平”类首点、“右平前升”类尾点、P_h 在 P_f 左边的“后升前降”类尖点或 P_h 在 P_f 右边的“后降前升”类尖点以及“后升”类普点,才可能具有偶点,而其他类型的 P_c 不可能有偶点.

2.2 算法实现

2.2.1 边界存储

单连通闭区域(如图 10 所示)只有一条闭合边界(如图 11 所示). 复连通闭区域(如图 12 所示)有两条以上的闭合边界(如图 13 所示).

图 10　单连通区域

图 11　单连通区域边界

图 12　复连通区域

图 13　复连通区域边界

在边界存储时，需要将单复连通区域的边界统一考虑，即把连通区域的一或多条边界看成为一条闭合边界. 单连通区域本身就只有一条边界，而对于复连通区域，则需将一条边界的终点与另一条边界的起点相连(通过存储方式实现)而链接成一条闭合边界链. 比如，针对图 13 所示的情形，可以将边界 A、B、C、D、E 依次链接成一条边界链. 链接时，需要注意如下两个方面的内容：

(1)为遵循边界像素要一致正向存储的原则，应将外部边界(如图 13 所示的边界 A)上的边界像素按逆时针方向存储；而对所有内部边界(如图 13 所示的边界 B、C、D、E)上的边界像素，则应按顺时针方向存储.

(2)为保持每条边界各自的闭合特性，同时为消除边界链中，上一条边界的终点和下一条边界的起点的相互影响，需要对因为链接存储而形成的“存储边界”进行单独处理. 其处理方式是，在存储每条边界时，应在第 1 个位置存储边界的终点像素，然后再存储起点像素，并正向顺次存储剩下的边界像素，存储完终点像素以后，在下一个位置还要保存该边界的起点像素. 这样，就保持了每条边界各自的封闭特性. 为了避免边界链中一条边界的终点与下一条边界的起点的相互干扰，应在一条边界存储结束之后，再存储一个坐标为(−1,0)的“像素”(这里假定所有屏幕可显示像素坐标都按非负值映射)作为“存储边界”的判断条件. 这样在进行遍历时，只对每条边界的第 2 个存储位置到倒数第 2 个位置之间的像素进行处理. 遇到“存储边界”像素时，则开始对新一条边界进行处理，直到将所有边界像素处理完为止. 并可以在存储位置的最后，存储一个坐标为(−1,−1)的“像素”，作为全部边界存储结束的判断条件.

至于复连通区域的多条边界相互之间，按照什么先后次序进行存储，并没有什么特别规定，只要做到将所有边界都存储起来就行了.

另外，虽然单连通区域，由于只有一条边界，因而可以特殊处理，但如果将它与复连通区域统一看待，则只需按只有一条外边界的复连通区域存储边界就行了.

将边界按以上要求存储起来，就可以顺次判断，并从开头搜索存储位置，以搜索出偶点(判定偶点存在时). 搜索偶点是算法中唯一比较耗时的操作.

2.2.2 实例

这里给出算法的一个通用实例.

像素系列 ***P***，类型定义：

```
typedef struct_PIXELSET
{ int x;int y; int xl; }PIXELSET;
```

输入：正向存储边界像素坐标的 ***P***，类型数组首地址，以(−1,−1,0)作为结束标志值.

输出：用作输入参数的数组也用作输出参数.

代码实例：

```
void RegionRansack(PIXELSET *Matrix){
int uIndex, HandledNum, xL, froTrend, hroTrend;
bool bPixelSet;
PIXELSET curPixel, froPixel, hroPixel, cplPixel;
HandledNum=1;
while (Matrix[HandledNum+2].y>-1) {/*沿边界正向行进遍历区域的全部像素*/
if(Matrix[HandledNum+1].x<0) HandledNum+=3;
/*进入新边界，后移3个存储位置*/
curPixel.x=Matrix[HandledNum].x;
curPixel.y=Matrix[HandledNum].y;
hroPixel.x=Matrix[HandledNum-1].x;
hroPixel.y=Matrix[HandledNum-1].y;
froPixel.x=Matrix[HandledNum+1].x;
froPixel.y=Matrix[HandledNum+1].y;
hroTrend=curPixel.y-hroPixel.y; froTrend=froPixel.
y-curPixel.y; xL=curPixel.x;
/*若bPixelSet为True，则该当前像素具有偶点.依此加入对首点、尾点、尖点和普点的判断*/
bPixelSet=(hroTrend<0&&froPixel.x-curPixel.x>
0&&froTrend==0);
```

```
bPixelSet=bPixelSet || (hroPixel.x-curPixel.x>0&&
hroTrend==0&&froTrend<0);
bPixelSet = bPixelSet || ((hroTrend * froTrend<0)&&
(hroPixel.x-froPixel.x>0&&hroTrend>0 ||
  hroPixel.x-froPixel.x<0&& hroTrend<0));
  bPixelSet=bPixelSet||((hroTrend * froTrend>0)&&
(froTrend<0));
    if ( bPixelSet ) { uIndex = 1;
cplPixel.x=-1; cplPixel.y=-1;
   while(Matrix[uIndex+2].y>-1) { /* 找出当前
像素的偶点 */
    if ( Matrix [ uIndex ]. y = = curPixel. y&&
Matrix[uIndex].x<curPixel.x)
       if (curPixel.x-cplPixel.x>curPixel.x-Matrix
[uIndex].x){
          cplPixel. x = Matrix [ uIndex ]. x;
cplPixel.y=Matrix[uIndex].y;}
    uIndex++; }
    xL=cplPixel.x+1; }
Matrix[HandledNum++].xL=xL; }
HandledNum=1;uIndex=0;
while (Matrix[HandledNum+2].y>-1) { /* 去除数组
中的"存储边界" */
  if(Matrix[HandledNum+1].x<0)
HandledNum+=3; /* 进入新边界 */
   Matrix [ uIndex ]. x = Matrix [ HandledNum ]. x;
Matrix[uIndex].y=Matrix[HandledNum].y;
    Matrix [ uIndex + + ]. xL =
Matrix[HandledNum++].xL; }
Matrix[uIndex].x=-1; Matrix[uIndex].y=-1; }
```

2.2.3 说明

算法要求先得到区域的边界像素，并顺次存储，这是容易做到的. 根据边界描述(如边界方程)对边界线进行像素化[3]或者进行扫描处理，是区域填充及其相关应用几乎都需要的步骤，这种算法已经相当成熟；至于正向存储的条件，也容易得到满足. 其实，在存储边界像素的过程中，只需沿边界顺次一致地进行像素化，并将得到的像素保存起来，而不必关心像素是按正向还是逆向进行的存储. 而存储方向(正向或逆向)的认定则取决于对边界像素的遍历顺序(文中实例代码没有考虑这点，而只是简单地按正向存储的顺序处理边界像素).

更进一步的代码实现，可以先按约定的正向存储进行处理，一旦发现不一致(根据算法规则判定的第 1 个具有偶点的边界像素，在搜索完全部边界像素以后，仍然无法找到一个有效取值的偶点，此时若取负值，则表明像素的存储方向与处理过程所遍历的前后顺序方向不一致)，则按相反方向重新遍历，就可以得到正确的结果. 这样，就解决了如何确定存储方向的问题.

2.3 算法分析

2.3.1 正确性

首先，按照文中指定方式描述的边界像素只可能有前面列举的那些类型，而利用组合数学的知识可以算出这些类型的数目. 具体情况是，对于中点，P_h 的位置按组合运算可以表示为 C_2^1 种取值(2 种情况择其一，取值为 2)，P_h 确定后，P_f 的位置有 C_1^1 种取值，其后前势都只能有 C_1^1 种取值，将各取值相乘，则中点一共有 $C_2^1C_1^1C_1^1$ 种，即有 2 种有效类型；对于首点，P_h 的位置有 C_3^1 种取值，其后势有 C_2^1 种取值，P_h 确定后，P_f 的位置只能有 C_2^1 种取值，其前势只能有 C_1^1 种取值，将各取值相乘，则首点只能有 $C_3^1C_2^1C_1^1$ 种，即有 12 种有效类型；依此类推，可以算出尾点有 $C_3^1C_2^1C_1^1$ 种，即有 12 种有效类型；尖点有 $C_3^1C_2^1C_1^1$ 种，即有 12 种有效类型；普点有 $C_3^1C_2^1C_3^1C_1^1$ 种，即有 18 种有效类型. 总共加起来，边界像素一共有 56 种有效类型；其次，由算法的原理可知，区域内的任何像素显然必定属于，且只属于边界上与某个像素对应的像素系列 P_i. 综合起来，算法能遍历出区域的全部像素，也就是说，算法是正确的.

2.3.2 实验结果

图 10 所示的单连通闭区域是利用图 11 所示的边界像素经过前面给定的算法实例填充得到的，图 12 所示的复连通闭区域则是由图 13 所示的边界像素填充得到的. 实验结果证明，算法能够正确地填充复杂的连通闭区域.

2.3.3 性能比较

鉴于区域填充算法大多非常复杂，且算法适用性不一样，因此许多算法都存在多个变体，且变体的不同实现会对算法的性能构成很大影响[11]，这里在对本文算法与扫描线求交算法及种子算法进行性能比较时，只给以定性的分析.

总的来说，本文算法是一个综合性能比较好的算法，主要表现在以下几个方面：

首先，该算法适应性强，不仅能应用于任何复杂边界的连通闭区域，还能顺序一致地处理区域填充和点所在区域的判断问题. 相比之下，扫描线求交算法主要适用于边数较少的多边形闭区域或者诸如圆这样的边界比较规则的闭区域，并且需要区别对待

不同类型的边界；而种子算法由于需要事先给定一个处在区域内部的像素种子，因此它主要用于根据像素颜色属性的不同来进行区域填充的场合，其对于点是否属于某个区域的随机判定应用价值不大.

其次，本文算法的运算量主要集中在针对每个边界像素判断和搜索偶点这类简单的比较运算方面，代码量少，若以边界像素的数目 n 为输入规模，且以一个边界像素的处理为计量单位，则需要 n 个存储单位，若平均进行约 $n^2/4$ 个单位的比较运算，则时间复杂性的上界为 n^2，可记为 $O(n^2)$；而扫描线求交算法则需要区别各种不同情况来求出交点，由于还要进行建立与维护活性表、交点排序与配对等多种处理，运算复杂，代码量大，因此时间和空间开销都比较大，并且性能会随区域边界复杂性的不同而表现出很大的差异；当然，简易的种子算法也比较简单，同时能适应各种复杂边界闭区域的填充需要，但要对像素进行大量重复的压栈与出栈等操作，这要消耗很多的内存和时间，如果对种子算法引入扫描线求交步骤，则会遇到扫描线求交算法所涉及的一些问题.

再次，由于本文算法只对区域内的每个像素访问一次，因此输入输出量较小，且同时适合于软件和硬件实现；虽然扫描线求交算法的输入输出量也小，但不太适合于硬件实现；种子算法的情形与此大致类似.

另外，本文算法还有一点与扫描线求交算法不同的是，它不按扫描线顺序，而是沿边界线来对区域进行填充，这与种子算法从种子像素开始，按某种方式对区域进行填充也是不一样的.

3　结　论

本文提出了一种利用基于区域边界 3 个相邻像素的坐标关系来遍历区域的有效算法. 由于该算法适用于任何连通闭区域的像素遍历，因而对于任何连通闭区域的填充（区域光栅化）以及对点在各种复杂区域的判定与跟踪等方面（如程序各种热区界面的设计）都具有很好的应用价值. 同时，算法遍历的结果是，用边界像素的 P_n 形式表示了整个区域的全部像素坐标，即整个连通闭区域按边界线性化了，这对于开展如何更有效地表示封闭区域的研究具有潜在的参考价值.

参 考 文 献

1　孙家广. 计算机图形学（第三版）[M]. 北京：清华大学出版社，1999：165～190.

2　同济大学数学教研室. 高等数学（下册）[M]. 北京：高等教育出版社，1978：1～4，180.

3　刘勇奎. 计算机图形学的基础算法 [M]. 北京：科学出版社，2001：1～209.

谭明金　1972 年生，先后于 1993 年和 1996 年获南京理工大学数字控制专业学士和硕士学位，现为解放军理工大学工程兵工程学院讲师. 主要研究方向为计算机图形学、计算机安全.