基于 Z-map模型的加工区域边界抽取算法研究

周 刚 邬义杰 潘晓弘
(浙江大学现代制造工程研究所, 杭州 310027)

摘 要 为了快速精确地进行加工区域边界抽取, 给出了一种 Z-map加工模型的加工区域边界抽取算法, 该算法首先把 Z-map模型下规则网格点阵转化为二元图进行边界抽取; 然后以基于段长的方式, 逐行扫描步长段, 并利用上下行段之间的关系确定段左右节点的连接, 以形成有向环, 从而确定边界为外轮廓或为内轮廓, 该算法时间复杂度为 $O(n)$, n为步长段的数量; 接着通过对环中段间的连接关系分析, 恢复了加工区域完整的边界信息; 最后给出了该算法时间与段、行、列数之间的关系, 同时与以前的算法进行了比较。结果表明, 该算在效率和实施难度上都较以前算法有了一定的提高。

关键词 边界抽取 段 Z-map 数控加工
中图法分类号: TP391.74 **文献标识码**: A **文章编号**: 1006-8961(2008)01-0151-07

Machining Region Boundary Extraction Algorithm Based on Z-map Model

ZHOU Gang, WU Yi-jie, PAN Xiao-hong
(*Institute of Production Engineering, Zhejiang University, Hangzhou* 310027)

Abstract For reducing the region boundary extraction algorithm complexity, in this study, Z-map model to binary image with value-range is transformed, then region problem is extracted from binary image. Via scanning Row and run, right and left node of run is connected by its relationship between Row, then the loop is constructed by foregoing connection and extract region boundary. According to the relationship of the run's connection, resume the region boundary information is resumed completely. The time complexity of extraction algorithm is $O(n)$, while n is the number of run. At last, the relation between the time consuming of the algorithm and the number of run, row and column were provided, while comparison of this arithmetic and before was also given.

Keywords boundary extraction, run, Z-map, NC machining

1 引 言

NC数控程序的编程目前一般在 CAM 中完成, 其大致可以分为以下 3个过程: (1)工艺参数的确定, 加工区域的细分; (2)加工边界的确定; (3)刀具轨迹的产生和干涉检查。目前的研究比较集中于步骤(1)与步骤(3)的研究, 而对于加工边界确定的研究则比较少。本文在研究了开发基于 STEP-NC 数据模型的数控系统过程中, 在分析特征的加工方式的基础上, 给出了一种基于 Z-map加工模型的加工区域边界抽取算法。

直接抽取自由曲面的边界非常困难, 其计算模型和曲面的拼接、细分是其瓶颈问题。在基于网格点阵为基础的加工模型中, 文献[1]提出了一种以曲面点集模型作为自由曲面的加工模型, 文献[2]将利用 Z方向上的点阵投影的方法应用于粗加工中, 陈杉等人将 Z留量模型应用于智能加工, 以解决自动清根等问题[3]。文献[4]以 Z-map加工模型为零件加工模型, 建立了在 Z-map加工模型上进行

基金项目: 国家"863"高技术研究发展计划项目(2006AA04Z233); 国家自然科学基金项目(50575205); 浙江省自然科学基金项目(Y105686)
收稿日期: 2006-03-20; **改回日期**: 2006-08-10
第一作者简介: 周刚(1980~), 男。博士生。主要从事数控技术、STEP-NC方向研究。E-mail: zghb@zju.edu.cn

加工区域边界确定[5]和刀具轨迹产生[6]等一系列方法。

关于二元图的边界抽取的算法比较丰富,在检索到的文献已有关于扫描边界的抽取算法,基于步长编码的边界抽取算法、基于单调链的边界抽取算法以及基于邻接编码的边界抽取算法等[7~10]。这些算法虽解决了二元图图形的特征边界抽取问题,但是其算法复杂度较高。近来,一些研究者对上述的算法进行了不断的改进,如文献[5]介绍了基于步长编码的边界抽取算法,其算法复杂度达到了$O(n)$,其中n为步长段的个数,而且可以在确定边界的同时,确定边界的走向。当扫描边界时,一个复杂图形需要首先搜索不定数量的搜索开始点,然后才能开始边界的搜索和连接开始点。文献[11]给出了算法复杂度为$O(n)$的边界抽取算法,其中图形长度为$n\times n$。其实际的算法次数为$3n$。但是其只是确定边界的模糊轮廓,而且其边界轮廓线只是在黑白单元间游走,不能完全确定精确的图形轮廓。

Z-map加工模型是在规则的2维网格点阵中记录模型表面的Z值,文献[4]利用切片的方式来得到每一网格上模型表面的Z值。加工过程中,首先确定V_{low},如果模型表面的Z值大于V_{low},那么这一网格点为实点,否则为虚点。

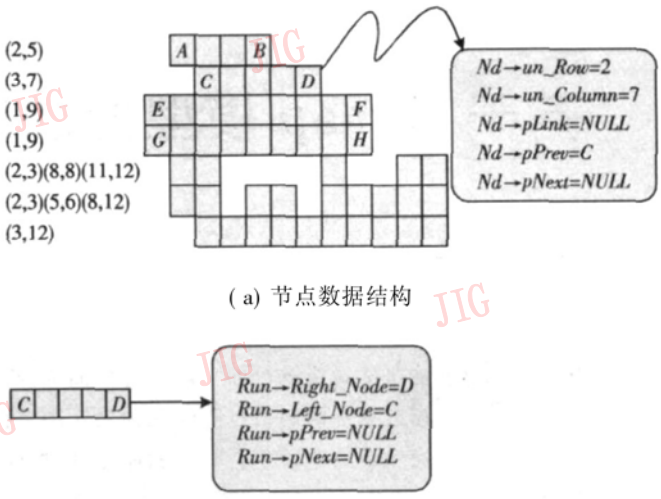
2 边界抽取算法概述

2.1 数据定义和描述

在阐述算法前,本文首先给出相应的数据定义和描述。

(1) 节点数据 节点是网格分析中的最小单位,也是Z-map模型中的网格单元。节点分为实节点和虚节点,实节点就是落在加工区域内或边界上的节点,虚节点为落在加工边界外的节点。通常此边界抽取算法中,并不是全部的节点都需要给出,大部分节点可在段生成过程中略去,只需保留段的左节点 left_Node和右节点 Right_Node,在数据存储中,应包括链表的前后指针$(\overrightarrow{Node}\ pPrev, \overrightarrow{Node}\ pNext)$及边界连接指针$\overrightarrow{Node}\ p_Link$。而每个节点包含其所在的行和列的信息$\overrightarrow{Node}\ un_Row$, $\overrightarrow{Node}\ un\_Column$(如图1(a)所示)。

(2) 段数据 如图1(b)所示,段由连续的实节点组成,用Run表示数据结构中包括左节点和右节点,其代表了段中最左、最右的两个实节点用$\overrightarrow{Run}\ Right_Node$和$\overrightarrow{Run}\ left_Node$表示。段在一行中的前后指针为$(\overrightarrow{Run}\ pPrev, \overrightarrow{Run}\ pNext)$。



(a) 节点数据结构

(b) 段数据结构

图 1 节点和段的数据结构描述

Fig 1 Data structure of node and run

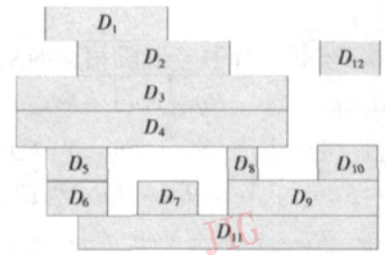
(3) 段间关系定义 首先根据上一行中的段与下一行中的段之间的位置关系确定以下两种关系:段相交和段分离。

段相交定义为上下两行中的段之间在某一列上同时为实节点;段分离定义为上下两行中的段之间在任何列上不同时存在实节点。

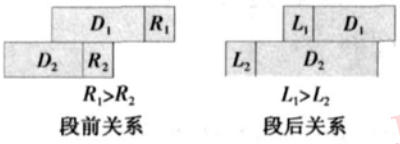
(4) 起始段、结束段和独立段定义 根据上下行中的段间关系,对段进行进一步的定义。假使某一段的上一行中没有与该段相交的段存在,那么称该段为起始段$Start_Run$,如果某一段的下一行中没有与该段相交的段存在,那么就称该段为结束段。如果该段既是初始段,又是结束段则称该段为独立段。如图2(a)中的D_1、D_7、D_{10}就是起始段,D_{11}为结束段,D_{12}为独立段。

(5) 段相交分类 段相交又可以划分为段前和段后两种形式。在段相交的基础上,如果上一行中的段的右节点在下一行中的段的右节点的右边,那么就称这两段的关系为段前;如果上一行中的段的左节点在下一行中的段的左节点的右边,那么就称这两段的关系为段后(如图2(b)所示)。

(6) 轮廓线定义 边界轮廓分为外环和内环,其中外环逆时针走向,内环顺时针走向,在段连接的同时确定环的走向。外环全部由段以及段端点连接而成(如图3(a)所示)。内环在列方向上由段的左右实节点连接而成,而在行方向上则用两端点的实节点以及实节点中间的虚节点表示(如图3(b)所示)。



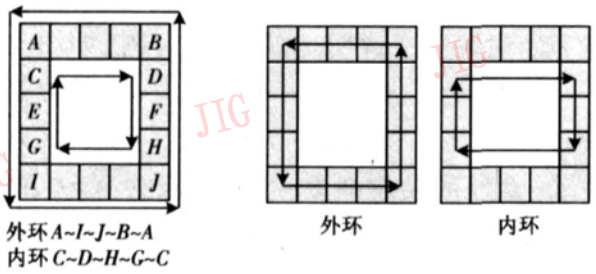
(a) 起始段、结束段和独立段



(b) 段前、段后关系

图 2　上下行段之间关系及段分类定义

Fig 2　Relationship definition between runs in contiguous row and run type



(a) 外环和内环的方向　　(b) 环的连接与连接方向

图 3　图形的边界轮廓线连接

Fig 3　Linked type of figure boundary

2.2　基于段连接的环构造算法

基于段连接的环构造算法的核心思想是采用段左右节点分别连接的方式, 并以行为单位, 逐行扫描, 完成区域边界的连接。其中左节点连接方向只包括向下和水平向左两种, 右节点连接方向包括向上和水平向右两种, 为了限定左右节点的连接方向可以在连接的同时确定环的走向, 并可根据环向确定环为内环或外环, 进一步就可以得到加工区域。单一行中左节点的连接可用右向传播子程序完成, 由于左节点的连接方向为向下和水平向左, 所以具体连接只需判断该行和下一行中的段之间的关系就可以确定。右节点与左节点的连接方式完全相同, 只是方向上有所区别。

下面给出具体的基于段连接的环构造算法 (以图 4中的图形为例)。

左节点平行连接子程序 (*Start_Run*, *Next_Row_Run*)

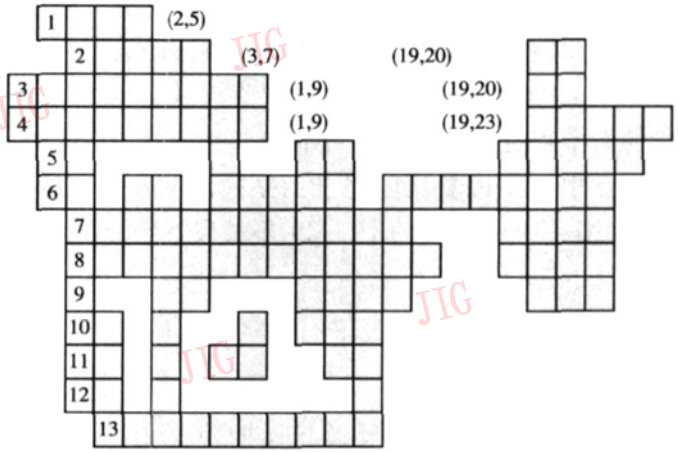


图 4　环构造示例图形

Fig 4　Example binary figure of loop constructing

Begin

While(*Start_Run* 的后一段 *Next_Run* 与 *Next_Row_Run* 是连接关系)

Repeat{*Next_Run*→*Left_Node*→*pLink* = *Start_Run*→*Right_Node*;

Start_Run = *Next_Run*; }

End

该子程序用于完成内环底部左节点的连接和选取第 6行和第 7行为连接的上下行。如图 5所示, 起始段 *Start_Run* 为 2~ 3段, 当连接到 14~ 18段时, 若该段的左节点 (18)大于 *Next_Row_Run* 段的左节点 (14), 则此次连接完成。在该行中再次调用左节点平行连接子程序, 即可完成 20~ 22段左节点的连接。

右向传播子程序用于完成段左节点的连接, 存在以下 5种连接情况: ①为结束段的左节点连接, 结束段的左节点与该段的右节点相连结; ②左节点与下一行中与该段相连, 而且处于最左边的段的左节点的连接; ③左节点与该段处于同一行中的段的右节点的连接。当前一段的右节点和后一段的左节点同时位于下一行的段中间时, 则利用左节点平行连接子程序完成连接, 且后一段的左节点指向前一段

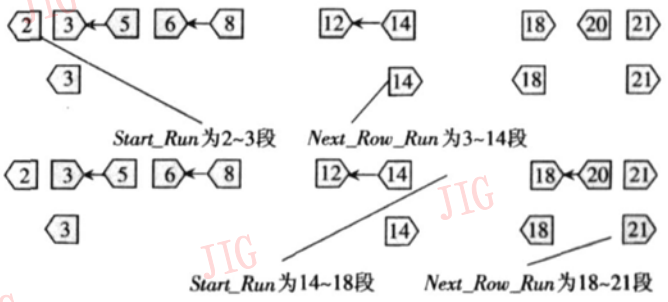
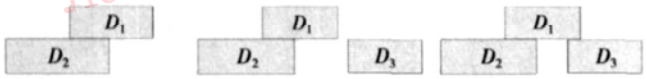


图 5　左节点平行连接子程序示例

Fig 5　Example of left-node parallel linking subprogram

的右节点; ④当上下行段的相交关系为段前关系时, 则可能出现 3种情况 (如图 6所示), 只有当在图 6 (c)的情况时, 下一步长段的左节点才有可能与该段的右节点连接; ⑤完成当前步长段在上下行之间与该行中的左边步长段完全独立的段的左节点的连接, 如图 4中第 2行的 19~ 20步长段。



(a) $D_2 \rightarrow Next = NULL$ (b) D_1 与 D_3 段分离 (c) D_1 与 D_3 段相交

图 6 段前关系在连接时出现的类型

Fig. 6 Three types in run-head relationship linking

右向传播子程序 *Start_Run*

Begin

B_Node = *Start_Run*$\rightarrow$*Left_Node*;

①**If**(*Start_Run* 为结束段)

{

B_Node$\rightarrow$*pLink* = *Start_Run*$\rightarrow$*Right_Node*;

Start_Run = *Start_Run*$\rightarrow$*pNext*;

右向传播子程序 *Start_Run*

return;

}

②*Next_Raw_Run* = 下一行中与该段相交的最左边段。

B_Node$\rightarrow$*pLink* = *Next_Raw_Run*$\rightarrow$*left_Node*;

③左节点平行连接子程序 (*Start_Run*, *Next_Raw_Run*);

④**while**(*Start_Run* 与 *Next_Raw_Run* 为段前关系)

Repeat{ *Next_Raw_Run* = *Next_Raw_Run*$\rightarrow$*pNext*;

If(*Start_Run* 与 *Next_Raw_Run* 是连接关系) {

左节点平行连接子程序 (*Start_Run*, *Next_Raw_Run*);

}}}

⑤**while**(*Start_Run*$\rightarrow$*pNext* 不为 *NULL*)

Repeat{*Start_Run* = *Start_Run*$\rightarrow$*pNext*;

右向传播子程序 *Start_Run*;

}

End

下面对图 4例子加以说明, 为了能清楚表示, 以下以字母代替数字说明。以第 6行为例 (如图 7所示), 此时 *Start_Node* 为 *A* ~ *B* 段, 因 *A* ~ *B* 段为非结束段, 而且与下一行的 *K* ~ *L* 段相交, 同时 *K* ~ *L* 段也是 *A* ~ *B* 段的最左端段, 故 *K* ~ *L* 段为 *Next_Raw_Run*, 那么 *A*$\rightarrow$*pLink* 就指向 *K*。进而调用左节点平

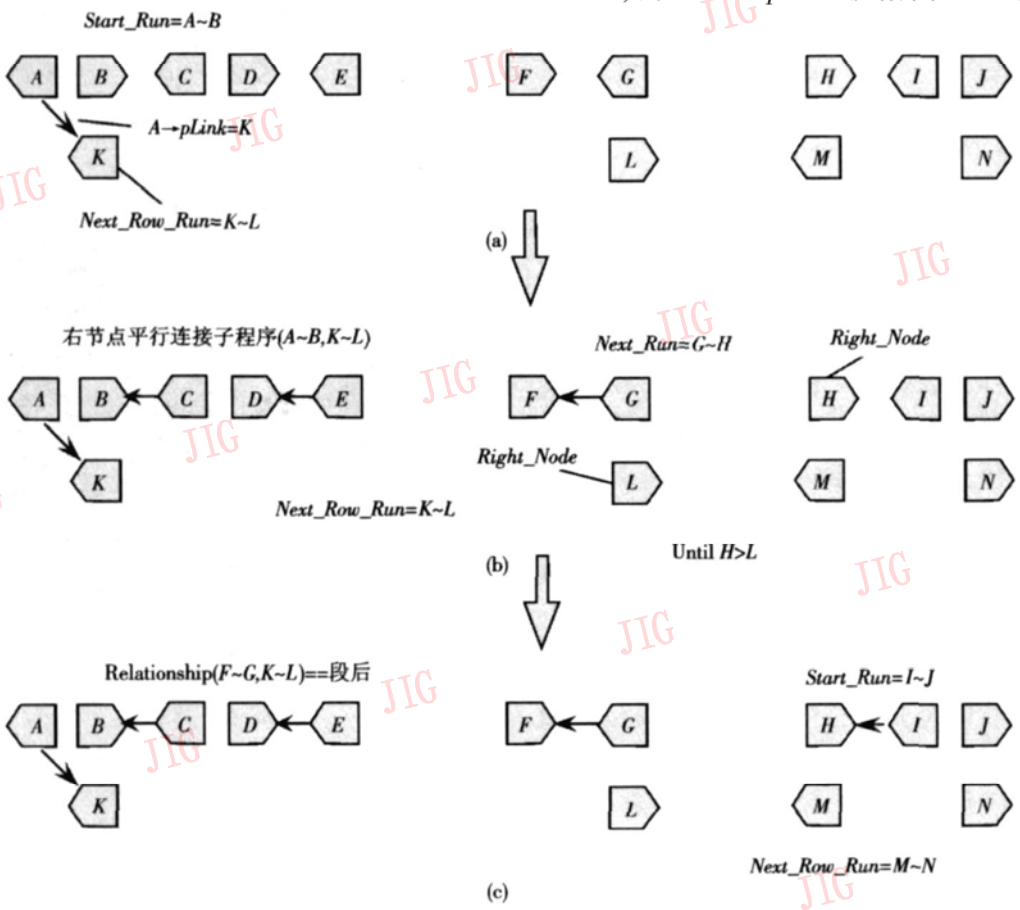


图 7 右向传播子程序示例

Fig. 7 Example of rightward linking subprogram

行连接子程序, 连接 $E^{\rightarrow} pLink=D$, $G^{\rightarrow} pLink=F$, 因段 $I\sim J$ 的 I 节点大于 $K\sim L$ 段的 L 节点, 故调用左节点平行连接子程序结束。判断 $G\sim H$ 与 $K\sim L$ 的段相交关系为段前关系, 而 $Next\_Raw\_Run$ 为 $M\sim N$ 段, 由于其与 $G\sim H$ 段是连接关系, 所以再次调用左节点平行连接子程序, 即可完成 $I^{\rightarrow} pLink=H$, 而 $I\sim J$ 步长段的 $pNext$ 就为 $NULL$。依此完成此行中所有段的左节点的连接。

左向传播子程序 *Start_Run*

Begin

B_Node= Start_Run→ Right_Node;

①**If**(*Start_Run* 为开始段) {

B_Node→ pLink= Start_Run→ Left_Node;

Start_Run= Start_Run→ pPrev;

左向传播子程序 *Start_Run*

return;

}

②*Next_Raw_Run*= 下一行中与该段相交的最右边段。

B_Node→ pLink=Next_Raw_Run→ Right_Node;

③右节点平行连接子程序 (*Start_Run, Next_Raw_Run*);

④**while**(*Start_Run* 与 *Next_Raw_Run* 为段后关系)

Repeat{*Next_Raw_Run=Next_Raw_Run→ pPrev;*

If(*Start_Run* 与 *Next_Raw_Run* 是连接关系) {

右节点平行连接子程序 (*Start_Run, Next_Raw_Run*);

} }

⑤**while**(*Start_Run* 不为 *NULL*)

Repeat{*Start_Run= Start_Run→ pPrev;*

左向传播子程序 *Start_Run*; }

End

完成了左节点的连接后, 再逆序完成右节点的连接。同样右节点平行连接子程序在算法上与左节点平行连接子程序完全相同, 只是在顺序上变为从右到左。跟右向传播子程序类似, 本文给出的左向传播子程序的示例分析如图 8所示, 即首先进入左向传播子程序 (*J*), 由于 $I\sim J$ 为非起始段, $U\sim V$ 为其最右端段, 所以 $J^{\rightarrow} pLink=V$。右节点平行连接子程序 ($I\sim J$, $U\sim V$)是首先连接 $H^{\rightarrow} pLink=I$; 然后调

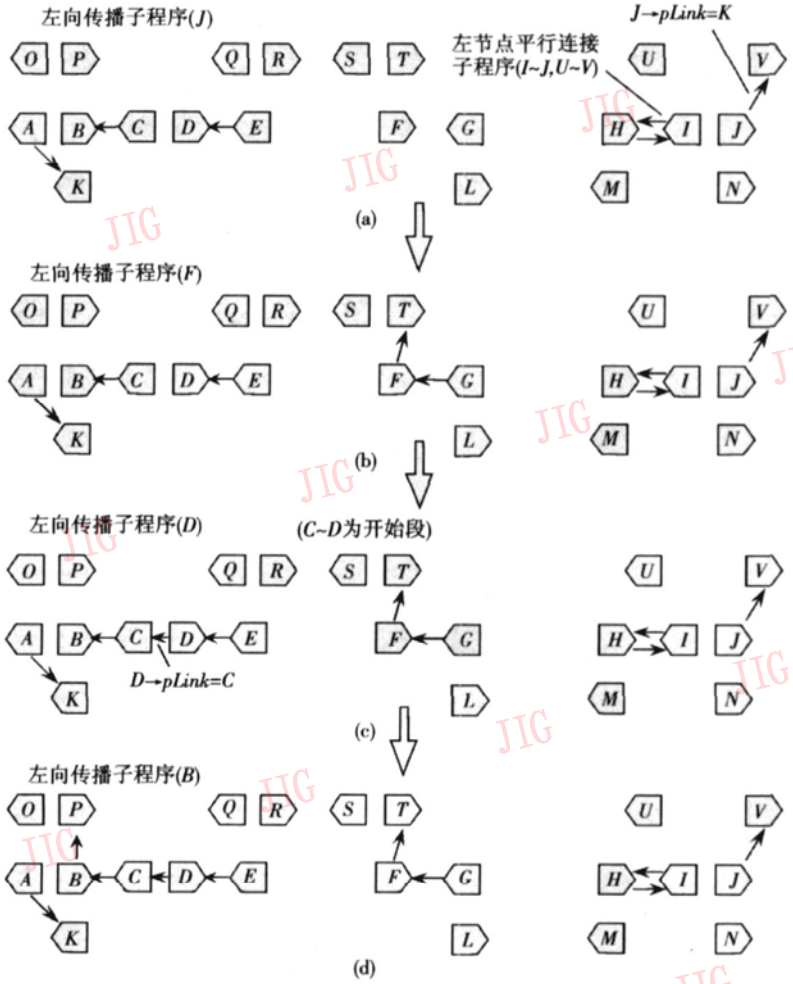


图 8　左向传播子程序示例

Fig. 8　Example of leftward linking subprogram

用第 ⑤步的左向传播子程序 (*F*), 以完成 $F^{\rightarrow} pLink = T$。继续调用左向传播子程序 (*D*), 由于 $C \sim D$ 是开始段, 所以 $D^{\rightarrow} pLink = C$。继续调用左向传播子程序 (*B*), 以完成 $B^{\rightarrow} pLink = P$。由此第 6行的左节点和右节点的连接全部完成。

基于上述左向传播和右向传播子程序, 可以给出以下主程序:

段连接构造程序 ( )

Begin

N = 1;

Repeat{*Start_Run* = 第 *N* 行的最左段;

右向传播子程序 (*Start_Run*);

左向传播子程序 (*Start_Run*);

N + +;

}**Until**(*N* 大于网格行数)

End

图 4所示的最终连接如图 9所示。这样通过对每一段进行扫描, 就完成了段与段之间的端节点连接。

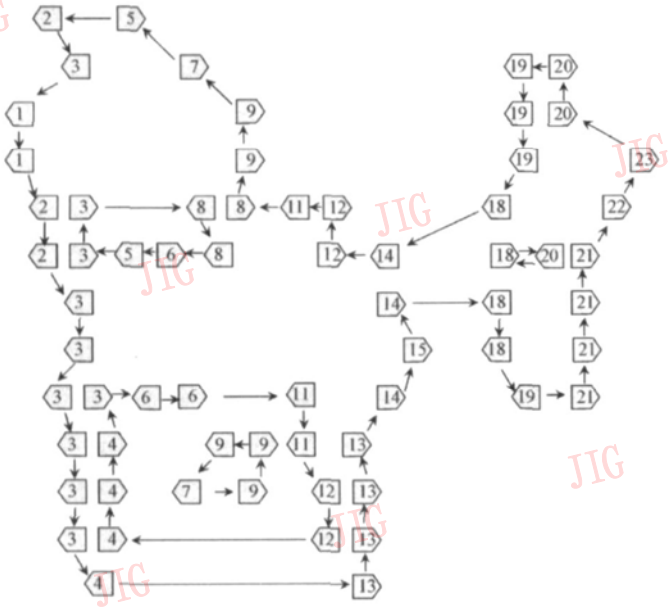


图 9 主程序段左右节点连接结果

Fig 9 Result of after right- and- left node linked

2 3 加工区域边界的恢复

利用上述算法产生的连接环需要通过边界的恢复才能完全表达区域的边界信息。在图 9产生的环中, 上下行有 6种连接表示 (如图 10(b)所示)。根据其连接关系可分别给出边界恢复的方式 (如左上表示的连接方式), 由于其实际的连接边界应是左上方向, 所以其走向应分解为先上后左两个过程。而在正下方向的连接上, 如果按连续两个正下方向的方式进行连接, 那么就合并为单个正下方向连接 (如图 10(a)所示)。

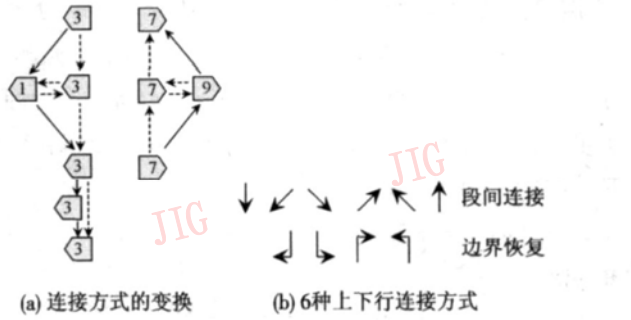


图 10 加工区域边界的恢复

Fig 10 Recognition of machining region boundary

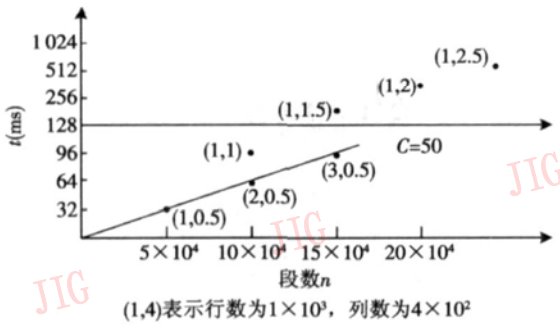
3 算法分析与数值试验

通过 C+ +编程, 给出算法的运算时间与段、行、列数目之间的关系, 结果如表 1所示。算法所花费的时间随着行数的增加呈线性的增加, 但是由于随着列数的增加, 所需时间的增长非常快, 所以有效分割图形的列数, 可以大大减少算法所需的时间。如果在段产生时就判定段与段之间的关系, 那么其算法运行时间可大大减少, 其与段、行、列数的关系如图 11(a)所示, 由图 11(a)可见, 其运行时间大致随着段数的增加而呈线性增加。同文献 [5]比较, 算法所需时间差别不大 (段与段之间的段前、段后关系可在段产生时确定, 不过需要增加段的关系数据结构才能完成段关系的确定), 两者关系如图 11(b)所示, 本算法可以适用于所有步长段之间关系的连接。文献 [5]主程序的算法复杂度为 *O* (*n*), 每段需要两次连接判断, 而且需要先寻找没有连接的节点, 然后才能完成这些节点的连接, 而采用逐行扫描的方式, 则可省略寻找开始点所需的消

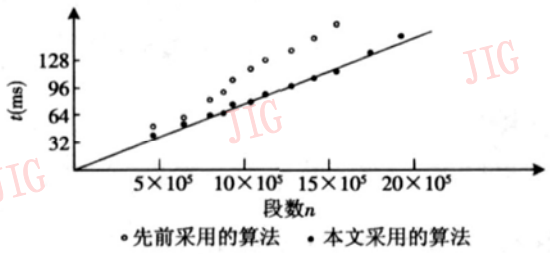
表 1 算法时间与段、行、列数目之间的关系

Tab 1 Relationship between time consumed and numbers of run, row and column

序号	行方向	列方向	时间 (ms)	段
1	1 000	50	32	40 694
2	2 000	50	62	81 384
3	3 000	50	94	121 949
4	1 000	100	96	80 632
5	1 000	200	344	161 100



(a) 算法时间包括搜索时间情况



(b) 算法时间不包括搜索时间情况

图 11　算法时间与段、行、列数的关系

Fig 11　Relationship between time consumed and numbers of run, row and column

耗。其准备时期所需的比较, 算法运算也是比较费时, 因为在各行中需确定 *Rear(Front)_Left(right)_Node*, 而且需要不断更新。本文对每段的连接, 同样需要两次判断 (左节点和右节点连接), 算法复杂度同样为 $O(n)$。

4　结　论

本文给出了一种 Z-map加工模型下的加工区域边界抽取的算法, 该算法先通过基于段的方法, 以逐行扫描的方式根据上下行段间的关系连接段的左右节点, 然后根据节点的连接关系恢复加工区域的边界连接, 以获得加工区域的边界。该算法不仅解决了复杂图形的边界连接问题, 而且算法复杂度达到 $O(n)$, 其中 n 为段数, 同时给出了算法的运行时间与段、行、列数目之间的关系图表。在单行上, 段的数量对算法运行时间的影响很大, 但在加工区域细分的基础上, 使用该算法可以达到大大降低算法运行时间的效果, 因为在段产生时确定段之间的关系可以有效降低算法程序的运行时间。

参考文献 (References)

1　Jerard R B, Angleton J M, Drysdal R L. The use of surface point sets for generation simulation Verification and Automatic Correction of NC Machining Programs[A]. In: Proceedings in NSF Design and Manufacture System Conference[C], Chicageo, IL, USA, 1989. 143~148.

2　Yan Guang-rong, Zhu Xin-xiong. A new approach for rough cutting [J]. Journal of Engineering Graphics, 2000, **21**(1): 28~35. [闫光荣, 朱心雄. 基于点阵投影的粗加工新方法[J]. 工程图学学报, 2000, **21**(1): 28~35.]

3　Chen Shan, Yan Guang-rong. The intelligent machining based on stock-remaining model[J]. Computer Aided Engineering, 2000, **9**(3): 25~32. [陈杉, 闫光荣. 基于留量模型的智能加工[J]. 计算机辅助工程, 2000, **9**(3): 25~32.]

4　Choi B K, Jerard R B. Sculptured surface machining[M]. Dordrecht Netherlands: Kluwer, 1998.

5　Park S C, Chio B K. Boundary extraction algorithm for cutting area detection[J]. Computer-Aided Design, 2001, **33**(8): 571~579.

6　Park S C. Tool path generation for Z-constant contour machining[J]. Computer-Aided Design, 2003, **35**(1): 27~36.

7　Cederburg R T. Chain-link coding and segmentation for raster scan devices[J]. Computer Graphics and Image Processing, 1979, **10**(3): 224~234.

8　Chakravarty I. A single-pass, chain Generating algorithm for region boundaries[J]. Computer Graphics and Image Processing, 1981, **12**(2): 182~193.

9　Kim S D, Lee J H, Kim J K. A new chain-coding algorithm for binary images using run-length codes[J]. Computer Vision Graphics and Image Processing, 1988, **41**(1): 114~128.

10　Sobel I. Neighborhood coding of binary images for fast contour following and general binary array processing[J]. Computer Graphics and Image Processing, 1978, **9**(1): 127~135.

11　Chia Tsong Lin, Wang Kuang Bor, Chen Lay Rong. A parallel algorithm for generating chain code of objects in binary images[J]. Information Sciences, 2003, **149**(4): 219~234.