

## 一种快速绘制剖面线的新算法(FDHP)

徐惠芳

(西北大学, 陕西 西安 710069)

**摘要:**文中给出了一种快速绘制剖面线的新算法(Fast Drawing Hatch Pattern, FDHP)。该方法无需对边界进行复杂而又不稳定的搜索,通过种子点对在其虚拟像空间((内存中自设一  $(X_{\max} - X_{\min} + 1) * (Y_{\max} - Y_{\min} + 1) / 4 + 1$  Bytes Buffer, 每个虚拟像素点(整个图形共计有  $(X_{\max} - X_{\min}) * (Y_{\max} - Y_{\min})$  个虚拟像素点)用二个 Bits 标识其状态)中进行区域填充以快速得到其所具有的最小拓扑区域,并在其区域进行剖面线绘制。该方法已在博士 CAD 系统中予以实现,大量用户常年使用,反应非常好。该算法新颖、高效、快速并具有非常好的鲁棒性等优点。

**关键词:**剖面线;算法;CAD;鲁棒性

**中图分类号:**TP301.6

**文献标识码:**A

**文章编号:**1673-629X(2006)11-0089-03

## A New Algorithm for Fast Drawing Hatch Pattern

XU Hui-fang

(Northwest University, Xi'an 710069, China)

**Abstract:** Describes a new algorithm for fast drawing hatch pattern(FDHP). There is no need for complex and unstable finding the boundary of an area. Based on the seed point, this algorithm uses fast improved flooding algorithm in the self-define memory buffer which can be set as  $(X_{\max} - X_{\min} + 1) * (Y_{\max} - Y_{\min} + 1) / 4 + 1$  Bytes (the whole picture will have  $(X_{\max} - X_{\min}) * (Y_{\max} - Y_{\min})$  virtual pixels), and each one only needs two bits to represent its status during calculation in size to get the minimum topological area for the hatch pattern. This algorithm has been implemented in Doctor-CAD system, and has been used in design and industry for a long time which has been proved to be very good and successful. This algorithm is very fast, efficient and very robust as well in the practical applications.

**Key words:** hatch; algorithm; CAD; robust

## 0 引言

当前 CAD 系统已经广泛地应用于工业设计的各个方面。而在任何一个商品化的 CAD 系统中,快速绘制剖面线都是一个十分重要而又常常难于解决的问题<sup>[1~3]</sup>。所以每一种商品化的 CAD 系统都用了大量的精力和时间来解决这个问题。目的是为用户提供一个好的绘制剖面线的方式和方法。由于其绘制剖面线方法常常涉及到一个商品化的 CAD 系统的核心技术,因此大都不予以公开发表。文献[2]上的方法比较复杂且在实际应用中存在着许多问题,文献[4,5]的方法存在着效率不高、鲁棒性差、实用性不够好的缺点。

对于一个商品化的二维 CAD 系统来说,一个实用、高效、鲁棒性好的剖面线填充方法是其成功的关键因素之一。文中给出的这种算法 FDHP 是一种新颖、高效、快速并具有很好的鲁棒性的剖面线填充算法。

该算法成功地应用于博士 CAD 系统中,其应用效果

十分理想。

## 1 对问题的抽象

在一般的 CAD 系统中,用户大都喜欢给定一个种子点来绘制剖面线。对于这种给定种子点的剖面线填充 FDHP 算法的基本思想是通过种子点对在其虚拟像空间(内存中自设一  $X_{\max} * Y_{\max} / 4$  Bytes Buffer, 每个虚拟像素点(整个图形共计有  $X_{\max} * Y_{\max}$  个虚拟像素点)用二个 Bits 标识其状态)中进行区域填充以快速得到其所具有的最小拓扑区域,从而确定起边界,在其区域内进行剖面线绘制。首先将二维图形及种子点由物空间转换到一虚拟像空间,在像空间进行操作然后将结果再转换到物空间。

一般二维图形都是由直线、多边形、圆、曲线等组成的。

设一个图形  $D = \{ \text{Line, Arc, Circle, Polygon, Curve, etc.} \}$  组成的集合。其中的子集合如下:

$\text{Line} = \{ L_i \} (i = 1, 2, 3, \dots, M_1)$

$\text{Arc} = \{ A_i \} (i = 1, 2, 3, \dots, M_2)$

$\text{Circle} = \{ C_i \} (i = 1, 2, 3, \dots, M_3)$

收稿日期:2006-02-27

作者简介:徐惠芳(1963-),女,浙江宁波人,讲师,主要从事 CAD 及工业控制方向研究。

Polygon =  $\{P_i\} (i = 1, 2, 3, \dots, M_4)$

Curve =  $\{V_i\} (i = 1, 2, 3, \dots, M_5)$

.....

由于所有非直线类图形都是由一个以直线段逼近的显示直线段列表,所以可以说所有的图形都是由直线段组成的。

因此任何一个 CAD 图形均可转换成只有  $N$  条直线段组成的集合  $L = \{L_i, i = 1, 2, 3, \dots, N\}$ , 如此以来, 无论图形如何复杂多变, 问题都可以归结为: 在一个物空间中已知一个种子点和  $N$  条直线段所组成的图形, 并对其图形进行剖面线填充, 即问题都可以抽象为已知如下信息的剖面线填充。

种子点:  $S = \{X, Y\}$

图形:  $D = \{L_i, i = 1, 2, 3, \dots, N\}$ , 每条直线段由两个点组成, 即  $Line = \{P_s, P_e\}$ , 而点由其二维坐标组成, 即  $P_s = \{X_s, Y_s\}, P_e = \{X_e, Y_e\}$ 。当然还有剖面线的填充模式 Hatch Pattern。这样的抽象就大大简化了其复杂性和多变性, 有利于整体效率和鲁棒性的提高。

## 2 算法的基本思想

设已知种子点及直线段集合, 算法由以下几个步骤组成:

1) 首先将图形进行坐标变换, 使其剖面线为一水平线。

2) 计算出该图形的最大最小区域 (Min - Max - Box), 并将所有的坐标圆整到像空间, 可以一对一圆整, 或放大某一给定的系数  $S$  圆整,  $S$  大于零, 一般情况下是 1, 但是如果所需绘制剖面线的区域很小时可以给一大于 1 的系数, 如 2, 5 或 10 等。当然值得说明的是这样的变换经过其逆变换后并不影响其结果。也就是说其剖面线绘制结果与其系数无关, 引入系数的目的主要是区域很小时其 Flood Fill 算法可能失效, 所以需要做放大处理。实际绘图中这种情况很少出现, 尽管如此博士 CAD 系统还是引入了此系数, 以保证用户在任何情况下都能绘出所需的剖面线。

3) 判定种子点是否在 Min - Max - Box 之内。若否, 则无剖面线。

4) 计算并分配一个内存 Buffer 作为虚拟像空间 (这里虚拟像空间用以区别屏幕显示空间), 每一个虚拟像素占 2 个 bits, 这样可以表示 4 种状态。

所需要的内存 Buffer 可由如下公式计算出:

Size (in Bytes) =  $(X_{\max} - X_{\min} + 1) * (Y_{\max} - Y_{\min} + 1) / 4 + 1$  (Bytes)。

在虚拟像空间, 每一个像素点的状态将可能是以下 3 种状态之一:

\* 边界点 (直线及 Min - Max - Box 边界所填的点, 值为 1);

\* 内部点 (由种子点起所填充的点, 值为 2);

\* 空间点 (未填充的空间点, 值为 0)。

5) 将内存 Buffer 的每一位置为零。

6) 首先需要设计并实现一个自己的直线的绘制算法, 在内存 Buffer 的虚拟像空间中绘制直线段, 并将其所有的直线段 (用值 1) “画” 在其内存 Buffer 中, 再将 4 条最大边界线也 “画” 于其中。

7) 设计并实现一个自己的 Flood Fill 算法, 在内存 Buffer 的虚拟像空间中从种子点起作快速填充, 每个像素点的值置为 2。

8) 快速填充完后, 从最小的  $Y_1$  坐标起找到内部点 (值为 2 的点), 并向两边搜索找出其起点  $X_s$  和终点  $X_e$ 。这样就得到一段剖面线的两个点 ( $X_s, Y_1$ ) 和 ( $X_e, Y_1$ )。

9) 由于步骤 8) 计算出的点是在虚拟像空间, 不是在物空间, 所以不是准确的交点, 还需用直线  $Y = Y_1$  与所有的直线段求交, 计算出精确的  $X_s$  和  $X_e$ 。

10) 将所计算出的剖面线端点 ( $X_s, Y_1$ ) 和 ( $X_e, Y_1$ ) 进行逆坐标变换, 使其从虚拟像空间回到物空间。这样一条剖面线的绘制就完成了,  $Y_1 + 1$  循环 8) 到 10) 的操作, 直到  $Y_{\max}$  结束。

11) 对于复杂的剖面线图案其思想是一样的, 只不过重复其以上的操作并把图案中所有的线段全部进行一遍即可完成。

## 3 博士 CAD 系统中的应用实例

博士 CAD 系统是 20 世纪 90 年代中期开发的国内具有自主知识产权的 CAD/CAM 集成系统中的 CAD 子系统。博士 CAD 系统是国家九五火炬计划项目, 系统荣获一九九八年陕西省科技进步一等奖。该系统在全国拥有众多的用户。该系统功能强大、稳定可靠, 是 20 世纪 90 年代中期国内著名的 CAD 系统之一。

以下是博士 CAD 系统中定义的部分 Hatch Pattern 及应用实例。

;; Ver. 4.2 DOCTOR - CAD Hatch Pattern File

\* ANGLE, Angle steel

0, 0, 0, 0, .275, .2, -.075

90, 0, 0, 0, .275, .2, -.075

\* ANSI31, ANSI Iron, Brick, Stone masonry

45, 0, 0, 0, .125

\* ANSI32, ANSI Steel

45, 0, 0, 0, .375

45, .176776695, 0, 0, .375

\* ANSI33, ANSI Bronze, Brass, Copper

45, 0, 0, 0, .25

45, .176776695, 0, 0, .25, .125, -.0625

\* ANSI34, ANSI Plastic, Rubber

45, 0, 0, 0, .75

45, .176776695, 0, 0, .75

45, .353553391, 0, 0, .75

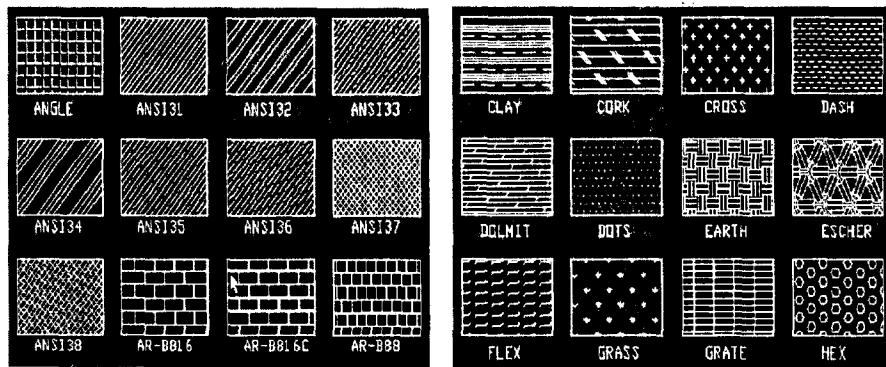


图 1 博士 CAD 系统的部分剖面线填充图案

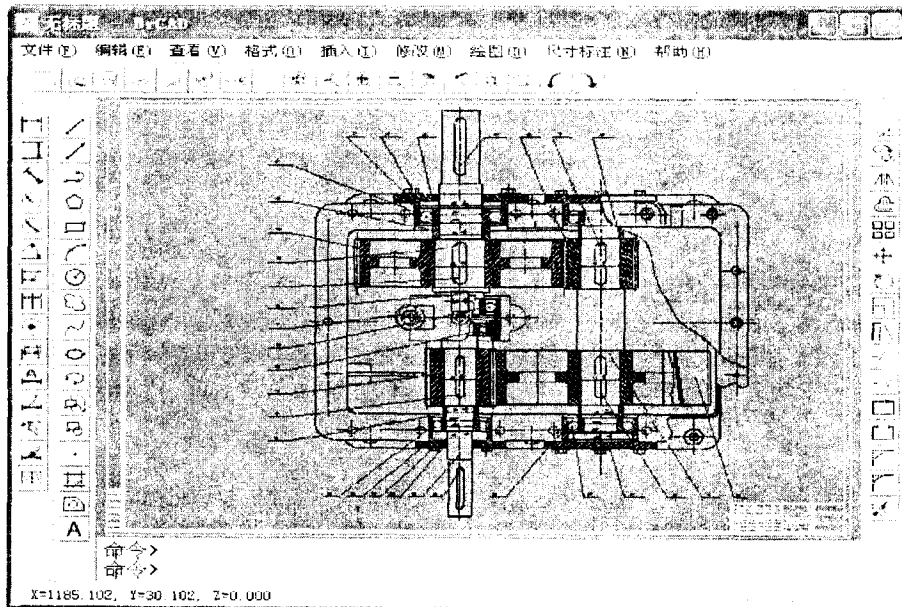


图 2 博士 CAD 系统生成的工程图纸(其中有大量的剖面线)

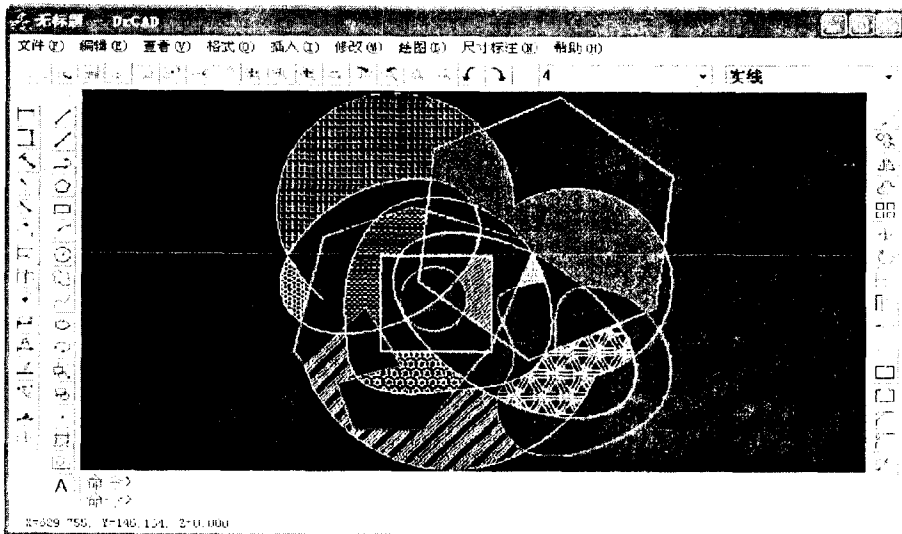


图 3 博士 CAD 系统在一个任意复杂的拓扑区域内生成的各种不同类型的剖面线

45, .530330086, 0, 0, .75

\* ANSI35, ANSI Fire brick, Refractory material

45, 0, 0, 0, .25

45, .176776695, 0, 0, .25, .3125, -.0625, 0, -.0625

图 1~图 3 是博士 CAD 系统的部分样图。

#### 4 结 论

文中所论述的 FDHP 算法无需对边界进行复杂而又不稳定的搜索,通过种子点对在其虚拟像空间中进行区域填充以快速得到其所具的最小拓扑区域,从而确定起边界,在其区域内进行剖面线绘制。该算法新颖、高效、快速并具有非常好的鲁棒性等优点。

该方法已在博士 CAD 系统中予以实现,从实际应用的情况来看,效果十分良好,是一种优秀的剖面线填充算法,可广泛应用于任何 CAD 及相关的图形系统中。

#### 参考文献:

- [1] 彭群生, 鲍虎军, 金小刚. 计算机真实感图形的算法基础[M]. 北京: 科学出版社, 1999.
- [2] 孙家广, 杨长贵. 计算机图形学[M]. 北京: 清华大学出版社, 1997.
- [3] 唐荣锡. CAD/CAM 技术[M]. 北京: 北京航空航天大学出版社, 1994.
- [4] 王志强, 肖立瑾. 一种高效可靠的剖面线参数化绘制技术[J]. 机械科学与技术, 1998, 17(4): 673-675.
- [5] 陈正鸣, 吴玉光. 剖面线的快速绘制技术[J]. 计算机工程与设计, 1999, 20(5): 47-51.

(上接第 88 页)

社, 2003.

- [5] Bovet D P, Cesati M. Understanding the Linux Kernel[M]. 2nd ed. 北京: 中国电力出版社, 2004.

- [6] Love R. Linux Kernel Development[M]. 2nd ed. 北京: 机械工业出版社, 2005.

- [7] Corbet J, Rubini A. Linux Device Driver[M]. 3rd ed. 南京: 东南大学出版社, 2005.