计算简单多边形的核

——计算几何实验报告 **2003 年 12 月**

指 导 教 师： 邓 俊 辉

作　　　者： 万　　海 (035451)

蒋 弘 山 (035441)

吴 邦 欲（035454）

1. 问题描述

简单多边形的定义：设平面上n 个点v_0，v_1，…，v_{n-1} 按循环排序方法逆时针排列，v_0 在v_{n-1} 之后，又设$e_0=v_{n-1}\ v_0$，$e_1=v_0\ v_1$，$e_{n-1}=v_{n-2}v_{n-1}$，是连接点的n条线段，那么这些线段界定一个简单多边形，当且仅当：1、循环排序中相邻线段对的交是它们之间共有的单个点：$e_i\cap e_{i+1}=v_i$；2、不相邻的线段不相交：$e_i\cap e_j=\varnothing$，j≠i+1，i=1,2…n-1，$e_0=e_n$。

简单多边形 P 的核(kernel)，K(p)定义如下：K(p)由多边形内部的点构成，这些点与多边形的任何顶点相连所构成的线段完全包含在 P 中。

K(p)的定义还可以等价地描述为：如果将多边形P的边看作逆时针方向所构成的环，则K(p)是多边形的所有边的左半平面的交集。通过观察可以发现，K(P)或者为空或者为完全包含在P 内的有界凸集。

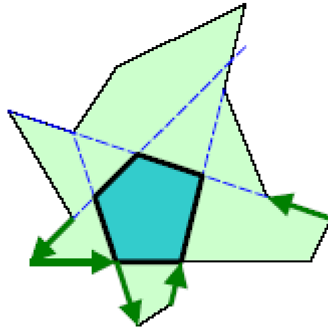


图 1 kernel 的定义

1976 年，Shamos 和 Hoey 提出了在 O(nlogn)的时间内完成 n 条边构成的简单多边形的 kernel 算法，该算法基于的思想是求 n 个一般的半平面的交需要的时间是 O(nlogn)，基本上是蛮干型。但经分析发现该下界并不适合求 kernel，这是因为在简单多边形中，所有半平面是按边的顺序有序排列的，而他们在设计算法时忽略了这个特性。

1994 年 Sven Schuierer 在论文"An O(log log n) Algorithm to Compute the Kernel of a Polygon"提出了利用 n/loglogn 个处理器并行处理点包含和直线求交的方法，将求 kernel 的复杂度降为 O(loglogn)。该算法基于按斜率排序的边数据结构，这篇论文的特点是将 kernel 的特性以及存在 kernel 的多边形的特性分析得非常透彻，但实现起来比较困难。

我们的实验是基于1978年D.T.LEE和F.P.PrePARATA提出的一个改进算法，在仔细阅读论文的过程中我们还发现图表和描述存在一些小的错误。这个算法充分利用了半平面有序排

列的特性，效率能达到线性时间，(与边的数目呈线性关系)。这种具有线性复杂度的算法称为核序列算法。算法依序扫描P 的顶点，同时构造一个凸多边形的核序列 $K_1, K_2, \cdots K_{n-1}$，这里每个多边形既可能有界也可能无界。K_i 是e_0 ， e_1 ，…， e_i 的 $i+1$ 个左半平面的交集， i=1,2 … n-1，因此有$K_{n-1} \subseteq \ldots \subseteq K_2 \subseteq K_1$，且$K_{n-1}$=K(P)。

2. 计算模型和基本概念

两种算法的计算模型是一致的。

简单多边形 P 的输入用一系列的顶点表示，v_0，v_1，…，v_{n-1}，其中 n≥4，顶点v_i用实数坐标表示(x_i, y_i)，多边形的边的表示方法为：e_i=(v_{i-1}, v_i)，i=1,2,…n，表示由顶点v_{i-1}和v_i构成的边。为方便表达，多边形 P 用顶点和相关的边形成的有序循环链表示 $v_0\ e_1\ v_1\ e_2\ e_{n-1}\ v_{n-1}\ e_0\ v_0$, 其中$e_0 = v_{n-1}\ v_0$同时我们给每条边赋予一个方向，使得多边形落在边的左侧，或者说多边形的边界定义为逆时针走向。

如果顶点v_{i+1}位于包含边e_i且与e_i通向的直线的右侧，则称v_i是凹点，v_i的内角大于180°。否则称v_i为凸点。在算法设计时，为了方便消除直线角(straight-angle)，我们将凸点的内角严格定义为小于 180°，消除直线角对多边形没有任何影响，可以在算法实施前在 O(n) 的时间内对 P 的边界做扫描来完成。

一些基本概念：

线的标记（Denotement of Line）：A 和B 是包含边e 的直线上的两个点，则A、B 之间与e 同方向的线段标记为AeB ；从B 发出的与e 同方向的射线标记为$\hat{}\ Be$ ；以B 为终点的与e 同方向的射线标记为$\hat{}eB$ 。

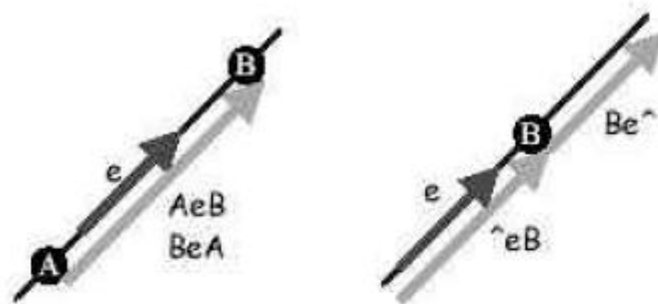


图 2 线段的标记

线性或循环表（Linear/Circular List）：在处理过程中， K_i 被维护成为顶点和顶点之间边的双链表，这个表为线性或者循环取决于K_i是开放还是封闭的，当它是线性表时，它的第一项和最后一项分别对应表头和表尾。

支持射线及支持点（Support Ray/Point）：如下图所示， K_i 关于顶点v_i 的支持射线为f_i 和l_i ，关于顶点v_i 的支持点为F_i 和L_i 。支持射线和支持点分别是某点到某凸集的

两条切线和两个切点。其中指定以 v_i 为起点的两条半线 f_i 到 l_i，在包含 k_i 的平面区域，从 f_i 到 l_i 的逆时针方向的角小于180^0。F_i 和 L_i 在以 k_i 为基础求 k_{i+1} 的过程中扮演重要角色。

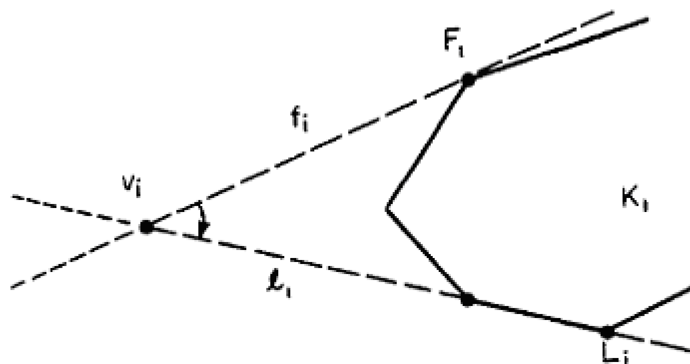


图 3 F_i 和 L_i 定义的图示

3. 算法思想

依次扫描多边形 P 的顶点，构造一个序列的凸多边形集 K_1，K_2，… K $n-1$，其中必然存在一个数 r<n，使得 K_1，K_2，…，K_{r-1} 是开放的，而 K_r, …, K_n–1 是封闭的。

所以，如果 w_i 和 w_{i+1} 是 P 的边 e_x 所在的直线上的两点，则称 wi e_x w_{i+1} 贡献了 w_i 和 w_{i+1} 之间的与 e_x 同方向的一条线段。如果 k_i 是开放的，则必然存在两条半线边(half-lines)，^ew 贡献的半线与 e 同向，终止于点 w，we^贡献的是补充半线，从点 w 出发，与 e 同向。

如果 P 中没有凹点，那么 P 是一个凸集，很显然 K(P)=P，所以在算法开始之前的预处理时首先搜索各顶点的内角，直到找到一个凹点，我们设为 v_0。图示如下：

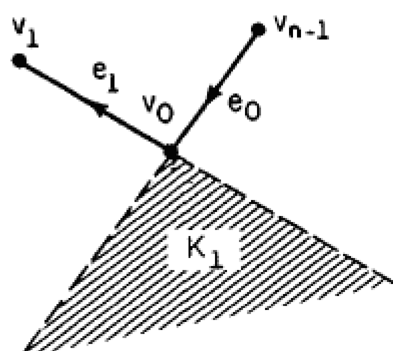


图 4 多边形 k_1 的图示

- 初始步改进：

  原算法在只有找到凹点时才开始构造 kernel。改进为从任意点开始都可以。初始步，首先设置 K_1 为 e_0 和 e_1 的两个左半平面的交，之后分两类情况设置 F_1 和 L_1：

  a、当顶点 v_0 对应的角为凹角时，F_1 和 L_1 分别被设置成 $\hat{}e_1v_0$ 和 $v_0e_0\hat{}$ 的无穷远点；

  b、当顶点 v_0 对应的角为凸角时，F_1 被设置成 v_0，L_1 被设置成 $v_0e_1\hat{}$ 的无穷远点。

如下图 5 所示：

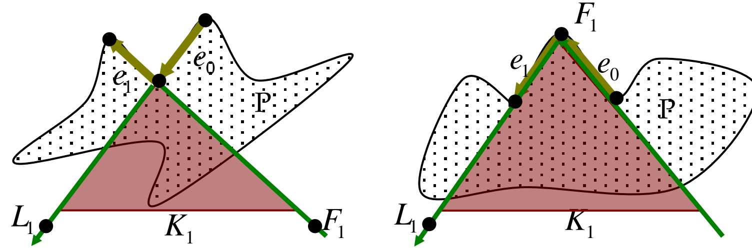


图 5 以任意点为起点时 F_1 和 L_1 的图示

- 循环递增步

在 K_i 的基础上求 K_{i+1}，假定我们得到了 K_i（它由以逆时针方向排列的顶点序列 w_1， w_1，…及相应的边来表示），为了构造 K_{i+1}，我们区分如下两大类情况：

a、 顶点 v_i 对应的角为凹角，即 $-\pi < e_i e_{i+1} < 0$，此时

i) 若支持点 F_i 位于 $\hat{}e_{i+1}v_i$ 的左侧，即 $\hat{}e_{i+1}v_i$ 与 k_i 无交点的情况。则将 K_{i+1} 置为 K_i，同时分别从 F_i 和 L_i 沿着逆时针方向找 K_{i+1} 相对于顶点 v_{i+1} 的支持点 F_{i+1} 和 L_{i+1}，如图 **6(a)**所示；

ii) 若支持点 F_i 位于 $\hat{}e_{i+1}v_i$ 上，即 $\hat{}e_{i+1}v_i$ 与 k_i 相切的情况。则将 K_{i+1} 置为 K_i，置 F_{i+1} 为 F_i，同时从 L_i 沿着逆时针方向找 K_{i+1} 相对于顶点 v_{i+1} 的另一支持点 L_{i+1}；

iii) 若支持点 F_i 位于 $\hat{}e_{i+1}v_i$ 的右侧，此时若 L_i 位于 $\hat{}e_{i+1}v_i$ 的右侧或其上，则 $K_{i+1} = \varnothing$，从而 ker(P)亦为空，结束运行；否则即是 $\hat{}e_{i+1}v_i$ 与 K_i 相交的情况。此时 $\hat{}e_{i+1}v_i$ 至多与 K_i 的边界 ∂K_i 交于两个点，按 ∂K_i 中线段的顺序可得 w'' 和 w' 两点，从 F_i 沿着逆时针方向搜索 ∂K_i 上的交点 w'，搜索终止于 L_i，若 w' 不存在，证明 K_i 位于 $\hat{}e_{i+1}v_i$ 的右侧，$K_{i+1} = \varnothing$，从而 ker(P)亦为空，结束运行。（理论上说这种情况不会发生，因为通过判断 L_i 与 $\hat{}e_{i+1}v_i$ 的位置关系已经能够得出交点一定存在的结论，但是考虑到实际运算时存在舍入误差，故加上这一步判断）。否则，若 K_i 开放，则另一个交点一定不存在，将另一交点置为 e_i 方向上的无穷远点 w''；否则，即若 K_i 封闭，则再从 F_i 沿着顺时针方向搜索 ∂K_i 上的交点 w''。连接 w'' w'，丢弃 K_i 中位于 $w'' e_{i+1} w'$ 右侧的部分，剩余部分构成 K_{i+1}，置 F_{i+1} 为 w''，从 L_i 沿着逆时针方向找 K_{i+1} 相对于顶点 v_{i+1} 的另一支持点 L_{i+1}，如图 **6(b)**所示。

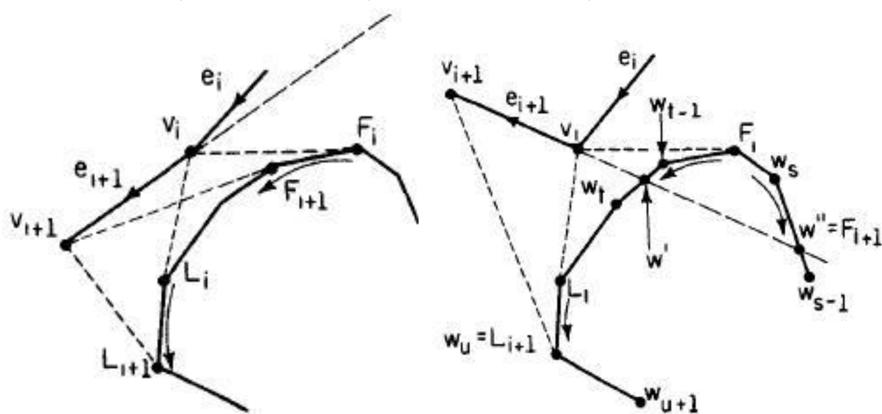


图 6 (a) **图 6 (b)**

b、顶点 v_i 对应的角为凸角，即 $0 < e_i e_{i+1} < \pi$ ，此时

i) 若支持点 L_i 位于 $v_i e_{i+1}$^ 的左侧或 $v_i e_{i+1}$^ 上，则将 K_{i+1} 置为 K_i ，同时分别从 F_i 和 L_i 沿着逆时针方向找 K_{i+1} 相对于顶点 v_{i+1} 的支持点 F_{i+1} 和 L_{i+1} ，如图 **7(a)** 所示；

ii) 若支持点 L_i 位于 $v_i e_{i+1}$^ 的右侧，此时若 F_i 位于 $v_i e_{i+1}$^ 的右侧或其上，则 $K_{i+1} = \varnothing$ ，从而 ker(P)亦为空，结束运行；否则，即 F_i 位于 $v_i e_{i+1}$^ 的左侧，则 $v_i e_{i+1}$^ 与 K_i 的边界 ∂K_i 交于一个或两个点，首先从 L_i 沿着顺时针方向搜索 ∂K_i 上的交点 w' ，若 K_i 开放，则根据 $v_i e_{i+1}$^ 的方向判定是否存在另一个交点 w'' ，如果 $v_i w'$^ 位于 $v_i F_i$^ 和 $v_i e_{i+1}$^ 之间，则从 L_i 沿着逆时针方向搜索 ∂K_i 上的交点 w'' ，否则 w'' 为 $v_i w'$^ 方向上的无穷远点，连接 $w'\,w''$ ，丢弃 K_i 中位于 $w'\,w''$ 右侧的部分，剩余部分构成 K_{i+1} 。接着更新 K_{i+1} 和 L_{i+1} 的值，若 $v_{i+1} \in [v_i e_{i+1} w']$ ，则从 F_i 沿着逆时针方向找 K_{i+1} 相对于顶点 v_{i+1} 的另一支持点 F_{i+1} ， L_{i+1} 设为 w'' ；否则，若 $v_{i+1} \in (w' e_{i+1} w'')$ ，则置 $F_{i+1} = w'$ ， $L_{i+1} = w''$ ；否则，置 $F_{i+1} = w'$ ，并从 w'' 沿着逆时针方向 ∂K_i 上的支持点 L_{i+1} ，如图 **7(b)**所示。

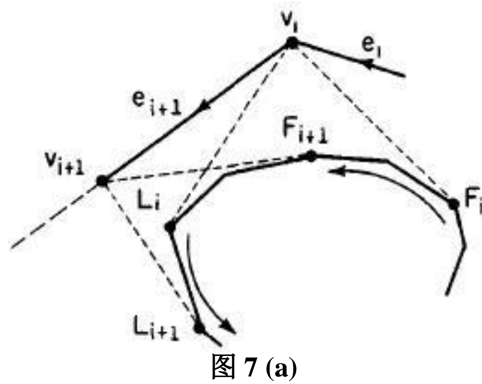


图 7 (a)

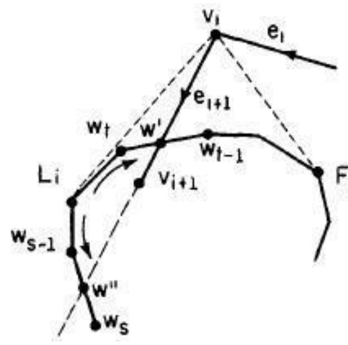


图 7 (b)

4．我们对论文中算法的改进和修正

- 原论文中的图 Fig3 和 Fig4 画法有错。因当 kernel 未封闭时，首尾两条半线应是发散的，不应该画成收敛状态。
- 初始步改进：

原算法在只有找到凹点才开始构造 kernel。改进为从任意点开始都可以。原算法在多边形为凸包时效率高，但一般情况下需要耗费 O(n)的时间来查找凹点。改造后的算法在一般情况下节省了找凹点的时间，但当多边形为凸包时则不如原算法预处理效率高。我们做此修改的原则是算法具有普遍性。

- 循环递增步改进：

(1) 当 v_i 是凹点时，kernel 不可能由 unbounded 状态转换为 bounded 状态。

(2) 当 v_i 是凸点时，kernel 不可能与 $v_i e_{i+1}$^ 交于两点而仍保持 unbounded 状态。

也就是说在本文的图 6(b)和图 7(b)中，直线^ e_{i+1} ^不可能与 K_i 上以 L_i 为起点逆时针方向的边有交点。

改进分析：减少了复杂情况的判断，进一步提高效率。理论证明见本文后半部分。

5. 理论证明

引理 1 Hi 定义为位于直线^e_i^的左边的半平面，则k_{i+1}是 $H_0,H_1,\ldots,H_{i+1}$，i=0，1，…，n-2 的交集。

证明 采用归纳法。$k_1=H_1\bigcap H_2$，假定$k_i=H_0\bigcap H_1\bigcap\ldots\bigcap H_i$，由于算法实现基于$k_{i+1}=k_i\bigcap H_{i+1}$，因此得到 $K_{i+1}=H_0\bigcap H_1\bigcap\ldots\bigcap H_i\bigcap H_{i+1}$。

引理 1 保证了构造 K(P)的算法的正确性。但经分析发现，有可能存在这样的 P，且 K(P)＝∅，但是循环递增时却要花费 $O(n^2)$的开销，因此需要仔细分析研究 kernel 的特性，以便尽早发现 K(P)＝∅，从而算法终止。

引理 2 P 是一个简单多边形，假定 K(P)≠ ∅，那么对任意点 p∈K(P)，以及 P 的边界上的任意点，线段 pu 必然包含在 P 内。

证明 设 u 是边 ej=(v_{j-1} v_j)上的点，假设 pu 不在 P 中，设 q 是 pu 落在 P 外部的任意点。由于 p 和 u 都位于 P 内部，而 pu 只是部分落在 P 内部，则必然存在多边形的两条边，设为 er 和 es 在 q 的两边分别从不同方向穿越 pu。又由于 p∈K(P),根据 kernel 的定义，p 与 P 的顶点的连线都位于 P 内部，因而 P 中的边不可能穿过线段(pv_{j-1})和(pv_j)，而 P 的边界本身构成环，er 和 es 应属于v_j和v_{j-1}之间的边所构成的链 C 上，Cej 是封闭的，因而我们推导出结论 P⊂(pv_{j-1} v_j)，进而得出结论：不可能(pv_{j-1})和(pv_j)同时是 Cej 上的边，至少有一条落在 P 的外部，这与 kernel 的定义矛盾，假设不成立。

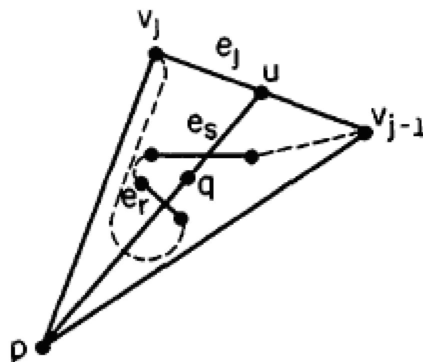


图 8 如果 p∈K(P),则在(pu)上不存在落在 P 外部的点

引理 2 的结论实际上强化了 kernel 的定义，也就是表明，kernel 中的任意点到 P 的边界上的任意点间的连线都位于 P 的内部。

现在我们着重考虑从 P 中的点v_0引出的针对k_i的支持线的半线 f 和 l，指定从 f 到 l 的顺时针角为凸的，f^*为v_0和距v_0最远的支持点间的线段，f`为 f
的补充半线，l^*和 l`的定义类同。

定理 1 设k_{i+1}非空，e_{i+1}穿越 l^*或者 f`，且$v_{i+1}$位于由 l 和 f`所构成的凸区域中，那

么 K(P)=∅。

证明 假设e_{i+1}穿越 1^*(穿越 1^*的下方怎么办？准确的描述应该为 1^*是一条半线)，则可以说 P 的边界将v_0和k_{i+1}分离开了(图 9(a))，这个现象在 P 的边界同时穿越 f^*和 1^*时很明显，如果不是同时穿越，例如，图示中v_i位于 f^*和 1^*所围成的区域，v_i不可能位于阴影区，否则与e_{i+1}穿越 1^*和 f`矛盾，或者说 P 的边界穿越 1^*两次，会导致k_{i+1}为空。假定 K(P)≠∅，令 p∈K(P)，所以 p∈k_{i+1}，依据引理 2，线段(pv_0)应位于 P 的内部，但实际上，(pv_0)跨越了 P 的边界，矛盾，所以 K(P)=∅。

假设e_{i+1}穿越 1`(图 9(b))，我们说 P 的边界将 f`和 1`构成的凸域分开了，这个现象在 P 的边界同时穿越 f`和 1`时很明显，如果不是同时穿越，例如，图示中$v_i$位于 f`和 1`所围成的区域，$v_i$不可能位于阴影区，否则与$e_i$穿越 1*和 f`矛盾，或者说 P 的边界穿越 f`两次，会导致k_{i+1}为空。假定 K(P)≠∅，令 p∈K(P)⊆ k_{i+1}，半线 p(pv_0)^与 P 的边界交于点 u，根据引理 2，线段(pu)应包含在 P 中，但事实上它在v_0点穿越边界，矛盾，所以 K(P)=∅。

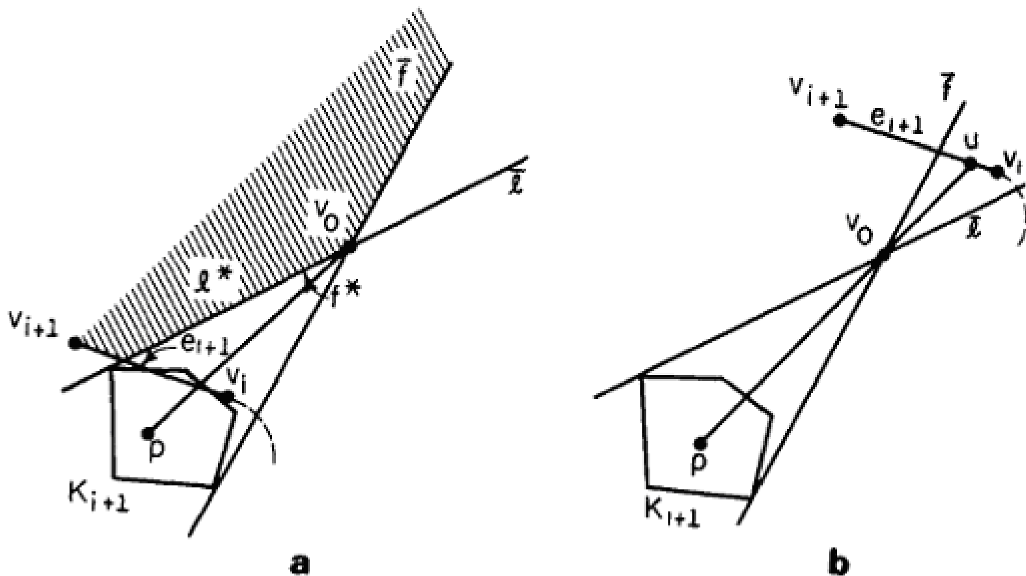


图 9 定理 1 证明的图示

根据上面的定理，我们按如下方式修改循环递增步：

计算k_{i+1}，F_{i+1}，L_{i+1}之前，测试是否e_{i+1}穿越 l*或 f`，同时测试$v_{i+1}$是否位于 f`和 l 界定的阴影区，如果条件成立，则终止算法，令 K(P)=∅。

(1) 在循环递增步的 a(iii)和 b(ii)，令 F*和 L*为 f 和 l 上的支持点，在确定k_{i+1}、F_{i+1}和L_{i+1}

后，更新 F*和 L*。在计算k_1时，令 F*和 L*分别指向各自半线的无线远处的点。如果在从k_i形成k_{i+1}时，F*或者 L*没有被删除，不必更新，否则必须根据k_{i+1}生成新的针对v_0的支持点。如果v_i是凹的：当v_0在^e_{i+1} v_{i+1}的左边，则 F*←w'，L*←w"；当v_0在^e_{i+1} v_{i+1}的右边，则 F*←w"，L*←w'；如果v_i是凸的：当v_0在 vie_{i+1}^的左边，则 L*←w'，F*←w"；当v_0在v_i e_{i+1}^的右边，则 L*←w"， F*←w'。其中 w'和 w"是在 a(iii)和 b(ii)中求出，w"可能是半线上无穷远处的点。

推论 1　假设k_i不空，p∈k_i，αj 是三角形(pv_{j-1} v_j)在点 p 的内部角，定义如果按逆时针方向(pv_j)在(pv_{j-1})之后，则角的值为正，否则为负，其中 j=1,2,…，i，我们可以有结论：$\sum_{j=1}^{i}$ αj<3π。

证明　反证法。假设$\sum_{j=1}^{i}$ αj≥3π。这意味着 P 的边界从v_0开始，如图 10 所示围绕k_i中的任一点转圈，k_i是封闭的，(a)图中，P 的边界在支持点两边从两个方向跨越 l，(b)图中，P 的边界跨越 f`，根据定理 1，K(P)=∅，矛盾，假设不成立。

根据我们的分析，认为原算法中有一点考虑不完善。即当v_i是凹点时，不可能将原理不封闭的kernel构造成封闭的kernel。也就是说原论文算法中(1.1)的(ii)最后一种情况不存在。为此，引入了下面的引理、定理和推论。

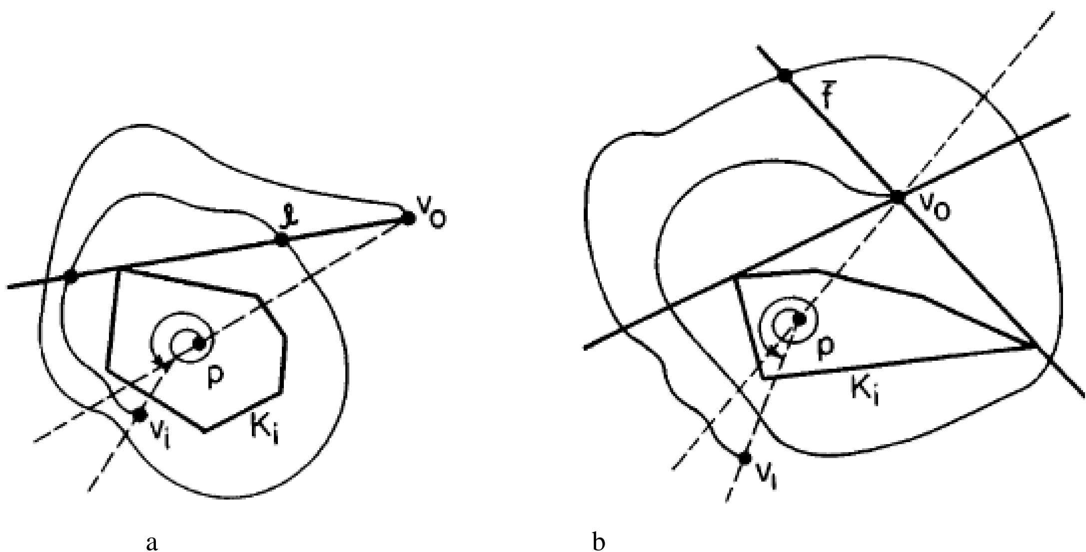


图 10 推论 1 证明的图示

以下的定理和推论是我们在分析问题时得出来的，根据推论我们优化了论文中的算法，丢弃了有两个交点时，逆时针查找第二个交点的情况。但尚未经过 D.T.Lee 的认可。

定理 2　在图 11 中，假设处理到点 v_i 时 k_i 尚未封闭，start 和 stop 分别是首尾两条半线，那么 v_i 只能出现在直线 stop 的右边或者在直线 stop 上。

证明　假设 v_i 出现在 stop 的左边，那么由于 stop 必是两相邻点构成的线段所在的直线，设为 v_j 和 v_{j+1}，那么由于 j+1<i，必然存在一条路径从 v_{j+1} 能够到达 v_i，在这些路径形成的直线中，一定存在这样的直线，或者使 K 落在它的右侧，导致 K(P)=∅，或者与 K 相交，导致 K 的形状发生变化。因而假设不成立。

推论 2 不论 v_i 是凸点还是凹点，如图 6(b)和 7(b)所示，简单多边形 P 的边^ e_{i+1} ^在与 k_i 交于两点时，第二个交点 w``一定位于 k_i 上从 w`开始沿 k_i 的顺时针方向上的边或初始半线上。

证明　如图 11 所示，由于 v_i 必定位于 stop 的右方，过点 v_i 做 start 和 stop 的平行线以及 k_i 的支持线，三条直线将平面划分位 6 个区域，当 v_{i+1} 位于区域 I，则 k_i 落在 e_{i+1} 的右边，k_{i+1} 为空；当 v_{i+1} 位于 II 和 V 区域，那么 e_{i+1} 与 k_i 只有一个交点，且交点落在 start 平行线的左方；如果 v_{i+1} 位于 III 和 VI 区域，则 e_{i+1} 与 k_i 有两个交点，均落在 start 平行线的右方；当 v_{i+1} 位于 IV 区域，则 k_i 落在 e_{i+1} 的左边，k_{i+1} 不变。从上面的分析得出结论，无论 v_{i+1} 出现在 v_i 的哪一个方位，边^ e_{i+1} ^在与 k_i 交于两点时，第二个交点 w``一定位于 k_i 上从 w`开始沿 k_i 的顺时针方向上的边或初始半线上。

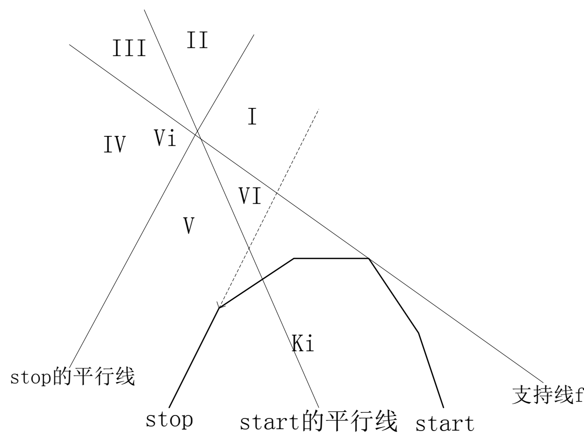


图 11 定理 2 证明的图示

6. 算法实现

(1) 主要数据结构及算法

程序共分为两个部分：

- 界面模块。该模块实现如下功能
  - ✓ 有序点集的输入。保证得到的多边形是简单多边形（不相邻的线段不相交），保证相邻的线段不共线。
  - ✓ 有序点集的导入和导出。以文件的形式保存输入的点集，并能够从文件中得到输入的点集
  - ✓ 可视化演示算法的流程（即 kernel 的生成过程的可视化演示）
    - ✧ 单步演示（前进，后退）
    - ✧ 连续演示，连续演示的暂停和恢复
    - ✧ 演示中间的特定步
    - ✧ 特例演示，如螺旋型等
- 核心算法模块。该模块实现如下功能
  - ✓ 针对从界面模块得到的数据，按照 kernel 的构造算法，生成每一步的相应数据（用于界面模块的显示）
  - ✓ 将生成 kernel 的每一步的相关数据保存在数据结构 MyKernels 中

(2) 主要数据结构和算法

基本数据结构：

- 点的表示：

```
public class MyPoint {
    float x,y;            该点的坐标
    MyPo  int next,pre;            向前向后指针，分别指向前一个点和后一个点
    }
```

- kernel 的数据结构表示：

其实上，kernel 的数据结构就是一个双向循环链表，head 指向链表的第一个元素。其中_poly[]数组为产生该 kernel 的简单多边形。

在该数据结构中并没有显式的表示直线和线段，代之以双向链表中相邻两个点所确定的直线或线段。

```
public class MyPointList {
```

double Min_Value = 0.0001;　　误差容限
MyPoint head;　　指向链表头
MyPoint F , L , F_0, L_0, V_0; 分别指当前处理点的支持点，V_0 的支持点及 V_0.
MyPoint Vi , Viplus1 ,Visub1; 指 V_i,V_{i-1},V_{i+1}
int maxSteps;　生成 kernel 的最大步数
MyPoint _poly[];　　输入并经过预处理的简单多边形的顺序点集。
}

支持的操作：

- 向双向链表中添加和删除点或点集
- 点与直线的位置关系判断：三角形面积法
- 直线与直线，直线与线段，线段与线段的位置关系的判断（即判断是否有交点），如果有交点，则能求出交点的坐标
- 更新相应数据，如，F，L，F_0，L_0 等
- 判断一个点是凹点还是凸点

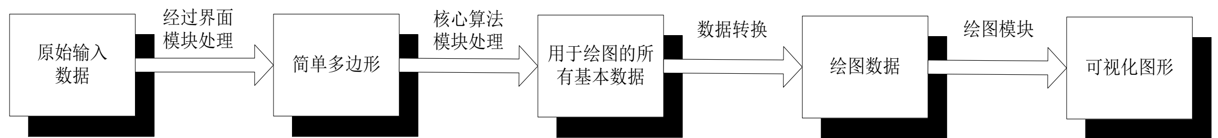
用以上的操作集和数据集可以由 Ki（第 i 个 kernel）生成 K_{i+1}（第 i＋1 个 kernel）。

- 用于演示的数据结构：
  public class MyKernels
  该数据结构中包含以下数据：
  - _Kernels[]数组，用于存放由核心算法传递过来的 Ki，以及相应的 F，L，L_0，F_0，V_i，V_{i+1}
  - 一些用于显示控制的变量，
    boolean showF0L0V0；　是否显示 F_0，L_0，V_0
    boolean showFL；是否显示 F，L
    boolean showShdowFL；是否显示 F，L 间的阴影
    boolean showHalfS；是否显示半平面的阴影
    boolean showKernelsBefore；是否显示前面生成的 kernel

  该数据结构中包含的操作
  - 生成用于绘制第 i 个 kernel 的相关数据
- 用于绘图的数据结构
  public class canvas
  该数据由 MyKernels 产生。
  操作：
  - 画点
  - 画线
  - 画内部添充的多边形（即 kernel）
  - 绘支持线 F，L，F_0，L_0
  - 绘制 FL 间的阴影
  - 绘制半平面的阴影

- 与经典算法的不同：
  将 K_0 限制为一矩形区域（遍历简单多边形，生成一能覆盖该简单多边形的最小矩形）。此操作使初始的 K_0 封闭，从而排除了无穷远点的处理和 kernel 不封闭时的处理。

（3） 算法流程



7. 性能分析

算法分析集中在两个主要操作上，首先是 kernel 的修改，即求 k_i 与半平面^ e_{i+1} ^的交，从而得到 k_{i+1}；其次是 F_i 和 L_i 的更新，包括求支持点以及交点时对 k_i 进行的扫描操作。在 a(iii)和 b(ii)的情况下，对 k_i 的修改也就意味着对支持点的修改。

我们从考虑求交入手，在 a(iii)的情况下，从 F_i 开始逆时针和顺时针扫描 k_i，从而得到 ^ e_{i+1} ^与 k_i 的两个交点 w'和 w''，(当然同时也能得到 F_{i+1})，令 N_i 是扫描到的边的总条数，所以一次这样的操作意味着从 k_i 要删除 N_i -2 条边，又由于被删除的边与多边形的边在同一条直线上，我们有 $\sum$ (N_i -2)≤n，所以 $\sum$ N_i，在算法(1.1)中要访问的顶点总数的上界为 3n，O(n)数量级。对(2.1)的算法分析同理。

现在我们考虑对求交过程中不能显示获得的支持点的修改，因为有些交点可以直接作为支持点，不必再加以分析。在(1.1)、(1.2)、(2.1)和(2.2)的情况下，L_i 会被修改，在 a(iii)、b(ii)的情况下，F_i 会被修改。分析一下在求 L_{i+1} 时需要访问的 k_{i+1} 的边的集合，我们称在找到 v_{i+1} 后，紧跟在它后面的点为 overshoot，很显然，多边形中的每个顶点至多一个 overshoot，所以，overshoot 的总数为 O(n)。其次，k_i 中的每条边最多被访问两次，否则根据推论 1，K(P)就会为空。因此这部分的上界也是 3n，O(n)数量级。

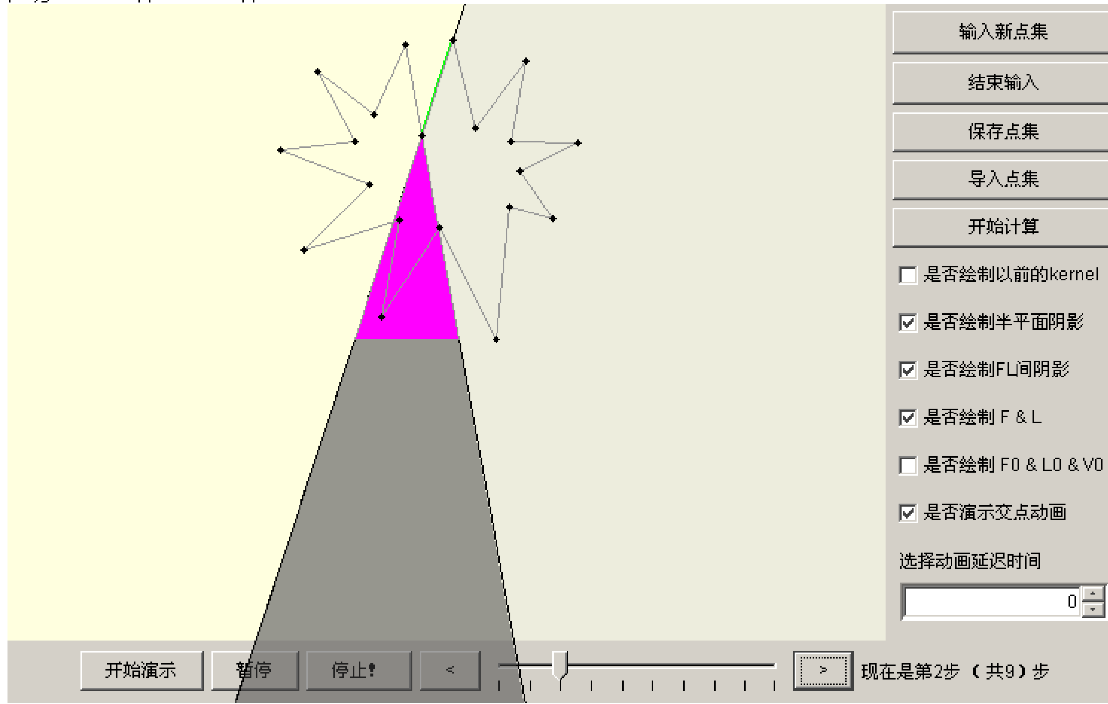
最后，每次形成 k_i 后，要针对 v_0 求支持点，复杂度分析同求 F*和 L*，实际操作时还要简单一些。

综上所述，算法复杂度为 O(n)，含常量因子。

8. 测试

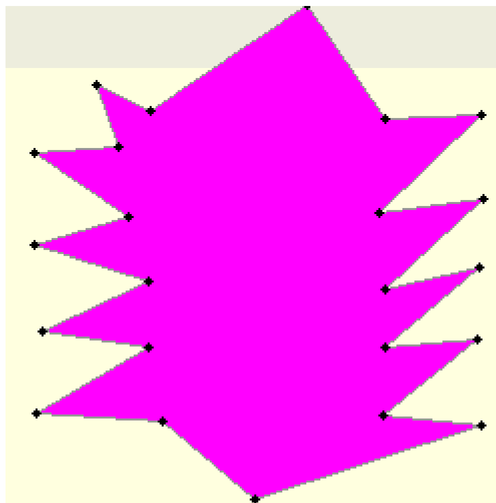
- 界面

polygonkernel.AppletUI will appear below in a Java enabled browser.

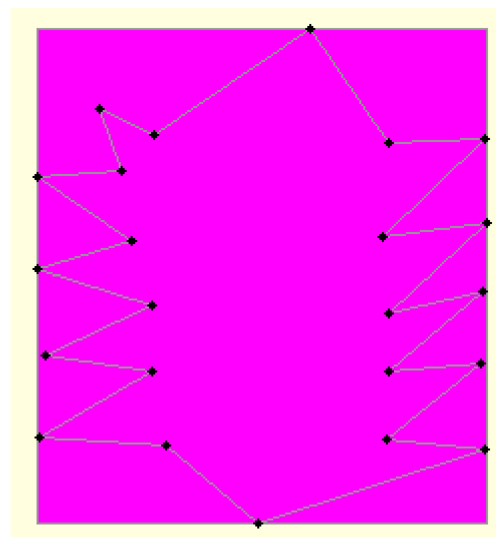


- 典型简单多边形的 kernel 生成示例

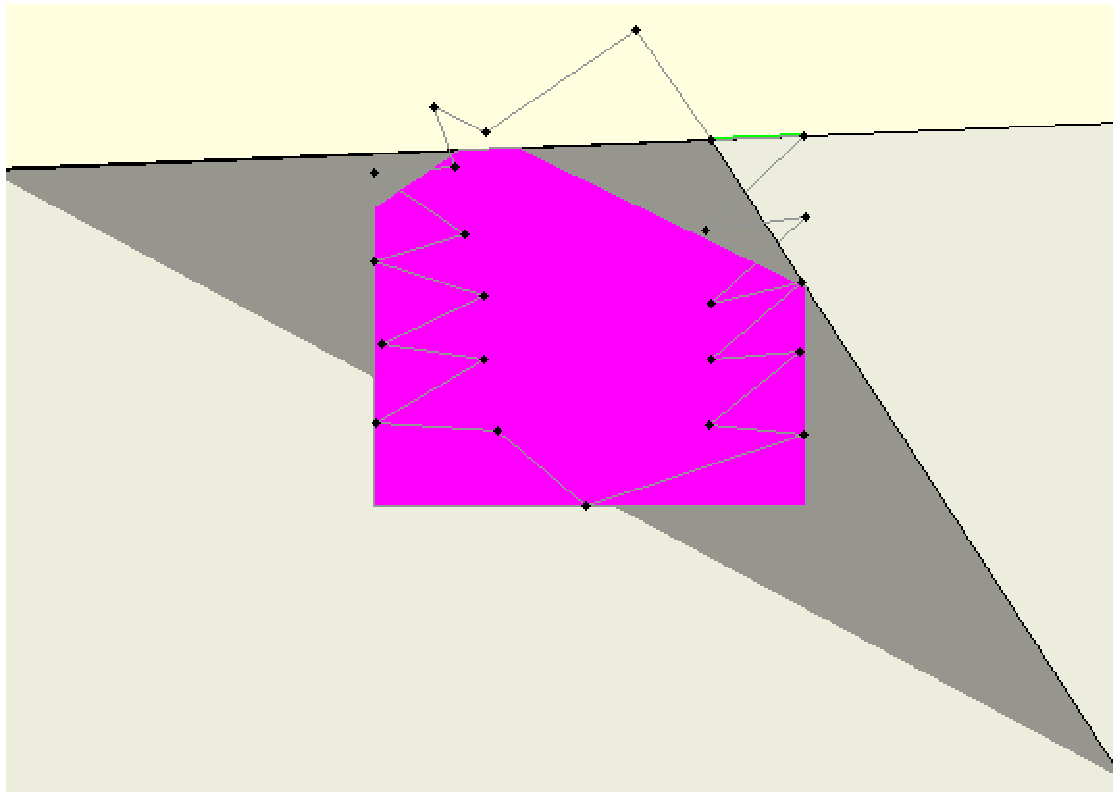
    - 多边形 1



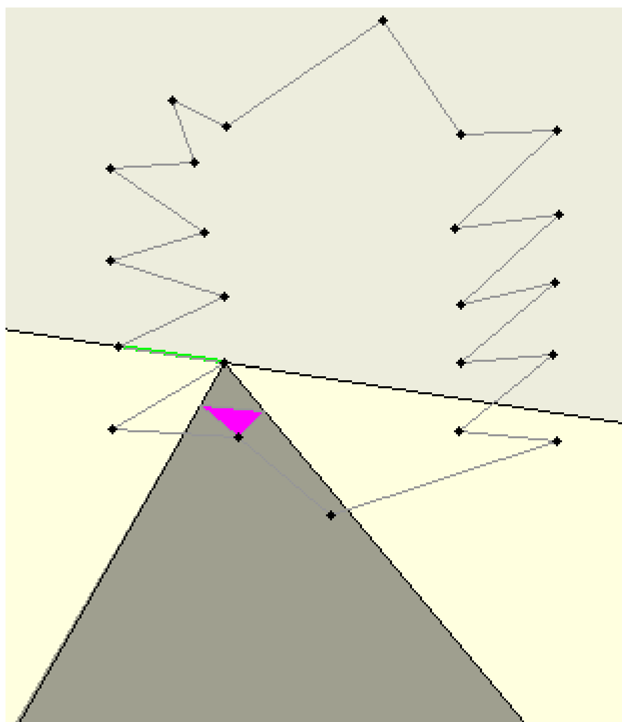
输入的简单多边形 1



初始矩形 kernel

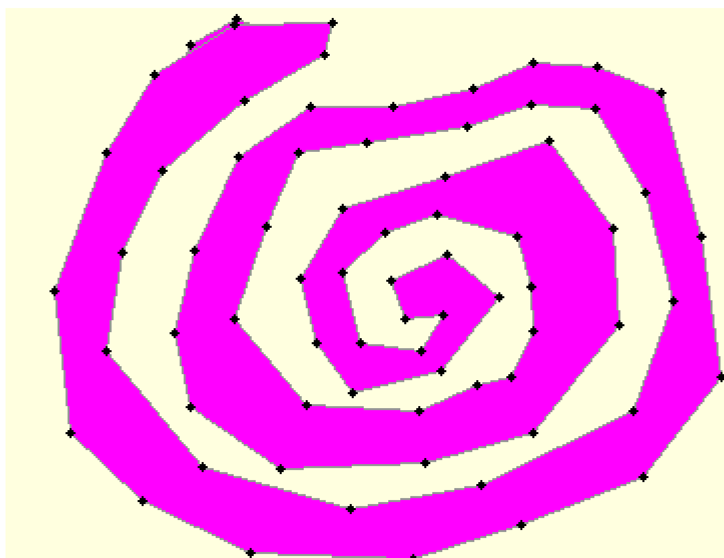


第 3 步生成的 kernel

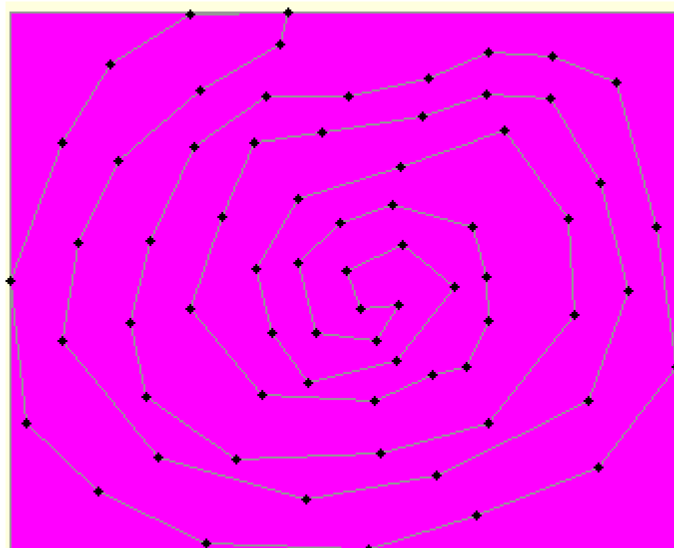


倒数第二步的 kernel,此时已能判断出 kernel 为空

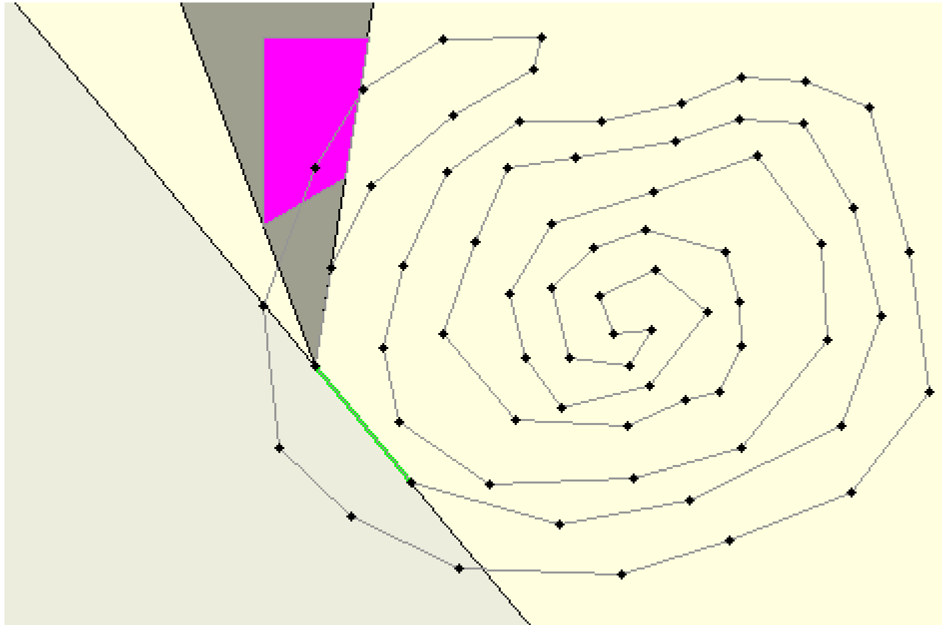
- 多边形 2



输入的多边形 2



初始的矩形 kernel



第 5 步即能判断 kernel 为空

9．实现过程中碰到的问题

由于采用了 JBuilder 开发环境，导致数据计算时误差比较严重，解决时引入了变量 MIN_VALUE，缺省取值 0.0001，如果两个点的距离小于 MIN_VALUE，则认为这两个点是重合的。在求点与直线的位置关系时，如果最终确定的值在一定的误差范围内，这认为点就位于直线上。

目前没有尚未解决的问题

致谢：

……。

参考文献

[1] S.Schuierer An O(loglogn) Time Algorithm to Compute the Kernel of Polygon。

[2] D. T. LEE AND F. P. PREPARATA，An Optimal Algorithm for Finding the Kernel of a Polygon， Journal of the Association for Computmg Machinery, Vol 26, No 3, July 1979, pp 415.-421

引用算法：无