平面简单多边形的直骨架实现

定义

2D 简单多边形的直骨架（straight skeleton），它是一种新的多边形的骨架，是仅由简单多边形的一些边的角平分线所组成的（如图 1 所示）。

轮廓边（contour edge），原简单多边形上的边。图 1 中红色的边。

骨架边（skeleton edge），由原多变形的边与边的角平分线所组成的边。图 1 中蓝色和绿色的边。

轮廓顶点（contour vertex），就是原多边形上的顶点。图 1 中红色轮廓边相交的顶点

骨架顶点（skeleton vertex）,是由骨架边相交而成的顶点。图 1 中绿色边和绿色边，蓝色边和蓝色边，或绿色边和蓝色边相交而成的顶点。

轮廓平分线（contour bisector），产生该种平分线的两条边在轮廓是相邻的。图 1 中绿色的边。

内部平分线（inner bisector），产生该种平分线的两天边在轮廓上不是相邻的。图 1 中蓝色的边。

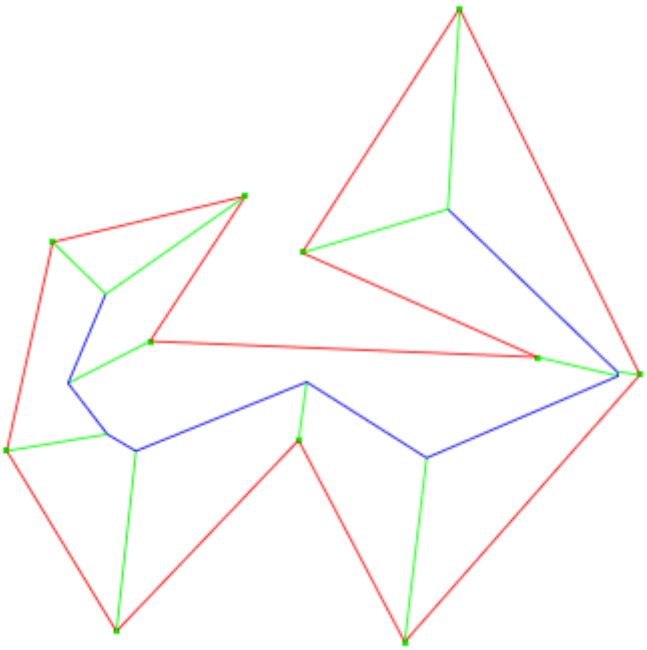


图 1 简单多边形及其直骨架

实验内容

本次作业的主要内容是实现 2D 简单多边形（包括凸多边形，非凸多边形，带洞多边形）的直骨架生成算法，以及在骨架生成之后，提供了骨架等高线的生成过程演示和简单多边形的三维屋顶演示效果。

实验目的

2D 简单多边形的直骨架有着广泛的应用，它相对于其它一般的骨架更加具有实际的应用价值，因此实验该种骨架的生成还是十分具有学习意义的。

实验步骤

AICHHOLZER 在[1]中提出了一种 2D 简单多边形直骨架的概念，在该论文中他给多边形直骨架的概念给了一个明确、清晰地定义：

从每一条多边形的轮廓边上向多边形内部穿过一个三维的平面，所有的这些平面与多边形的夹角都是相等的(我们演示的时候是设为 45°)，这些平面在三维空间上相交，相交所产生的三维的包围多面体上的边就相当于是多边形的屋顶的棱（可以认为原简单多边形是你家房子的墙壁，然后所生成直骨架就相当于你家的屋顶），然后包围多面体上的所有边和顶点在原简单多边形上进行投影，投影所得到的点和边就是简单多边形的骨架边和骨架顶点，也就是我们实验要最终求得的点和边。

数据结构设计

多向链表：由于轮廓顶点与骨架顶点之间的拓扑关系十分复杂，一个轮廓顶点不但要指向原多变形上 prev 和 next 轮廓顶点，还要指向其相关的骨架上的顶点，这就要求我们设计使用多向链表来维护整个多边形及其骨架的拓扑关系。见图 2。

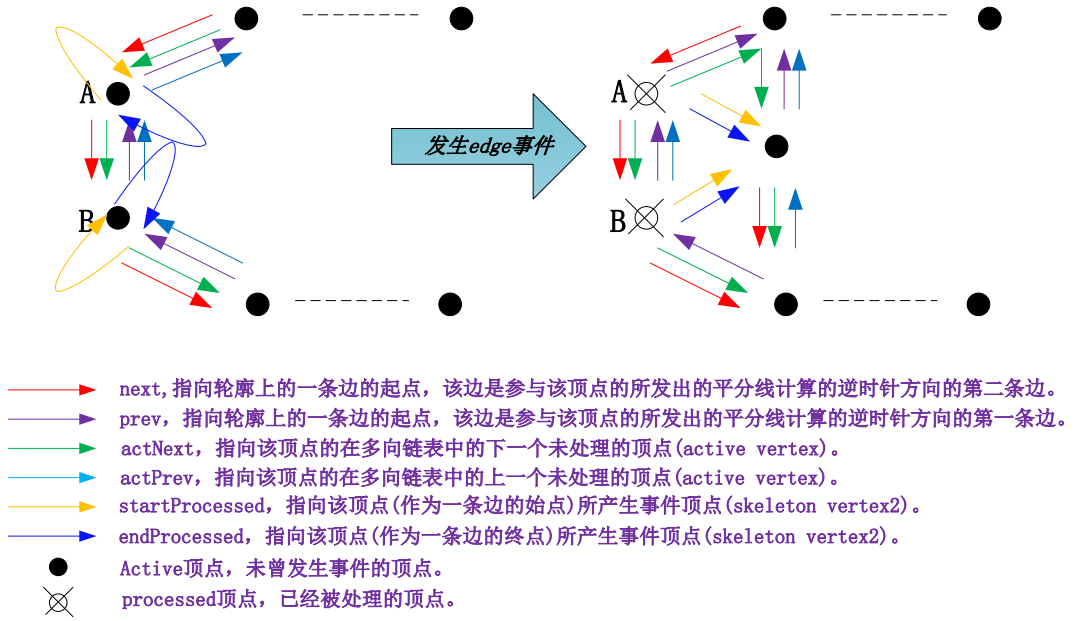


图 2 多向链表在发生 edge 事件的前后变化

优先队列：由于整个直骨架的构造算法是基于扫描线算法，因此优先队列这种数据结构是主要用来在 O(lgn)的时间内插入新的事件点，在 O(1)的时间内取出下一个事件点。在我们的算法实现的过程中，由于针对于每一个事件点都只有两种操作插入和取出，不存在在算法执行的过程中某个事件点的键值发生改变的情况，因此用 STL 中提供的 priority_queue 就可以满足需求了。

算法介绍

简单多边形的判定

我们实现的这个直骨架生成算法不仅能针对凸多边形，而且能针对非凸多边形，甚至可以针对带洞的多边形。简单多边形的一个很重要的判定原则本可以用 Chazelle 提出的算法来在 O(n)的时间内解决，但是为了方便用户在输入多边形的点的时候一旦发生了非多边形的输入可以及时改正，我们就采用了用户每输入一个新的顶点就判断一次是否是简单多边形，每次判定都在 O(n)时间内完成。

带洞的简单多边形，这个洞必须要完全位于多边形外轮廓的内部，而且存在多个洞的情况，洞与洞之间不能重叠，更重要的是不能存在洞中有洞的情况。

洞位于外轮廓的内部的判定

首先判定当前输入的洞上顶点是否位于外轮廓的内部，我们使用的是射线法，从当前输

入的顶点向某一个方向发射一条无穷长的射线，如果该与该射线相交的顶点数为奇数，则认为该顶点位于多边形外轮廓的内部，而且还要判定该洞顶点所产生的内轮廓边是否和其他边相交，如果不相交则说明该内轮廓边位于外轮廓的内部，等到一个洞画完，而且每个洞上的顶点都能通过以上判定，则可以断定该洞位于外轮廓的内部，否则一旦发现某个顶点不符合要求，就取消该顶点，提示用户重新输入。

洞与洞之间是否重叠或是洞中有洞的判定

由于我们对用户每次输入的顶点都进行的是否存在边相交的情况，因此洞与洞相交的情况也就判定了。但是有可能存在洞中有洞(一个洞完全包含在另一个洞里)的情况（见图 3），我们采用了也是射线法，在目前绘制的这个洞结束时，在每个其它的每个洞中，任意的挑一个顶点来进行射线法测试，如果当前绘制的洞完全包含其它洞，则射线相交的次数为偶数次（目前绘制的洞完全包含在其它洞中的情况，在每个顶点判定是否位于外轮廓的内部就可以判定了，因为这种情况射线相交的次数也同样为偶数次）。

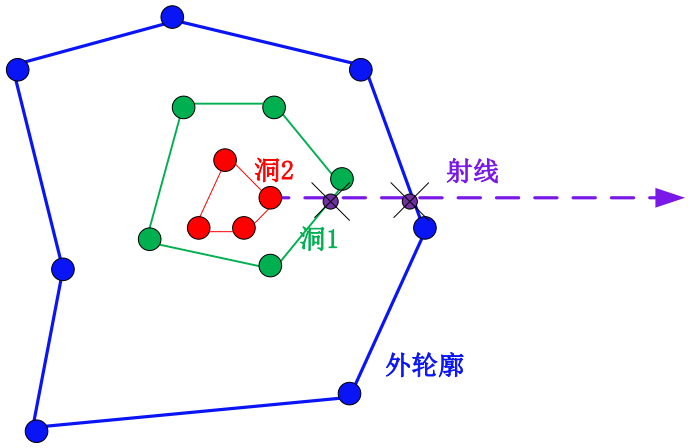


图 3 洞 1 完全包含洞 2

角平方线(bisector)的计算

传统意义上的角平分线就是相邻两条边的夹角平分线，在我们的算法执行的过程中，需要计算各种类型的角平分线，而且不仅是针对相邻的两条边，不相邻的两条边同样也可以进行计算角平分线，图 4 列出了 5 情况下角平分线的计算。

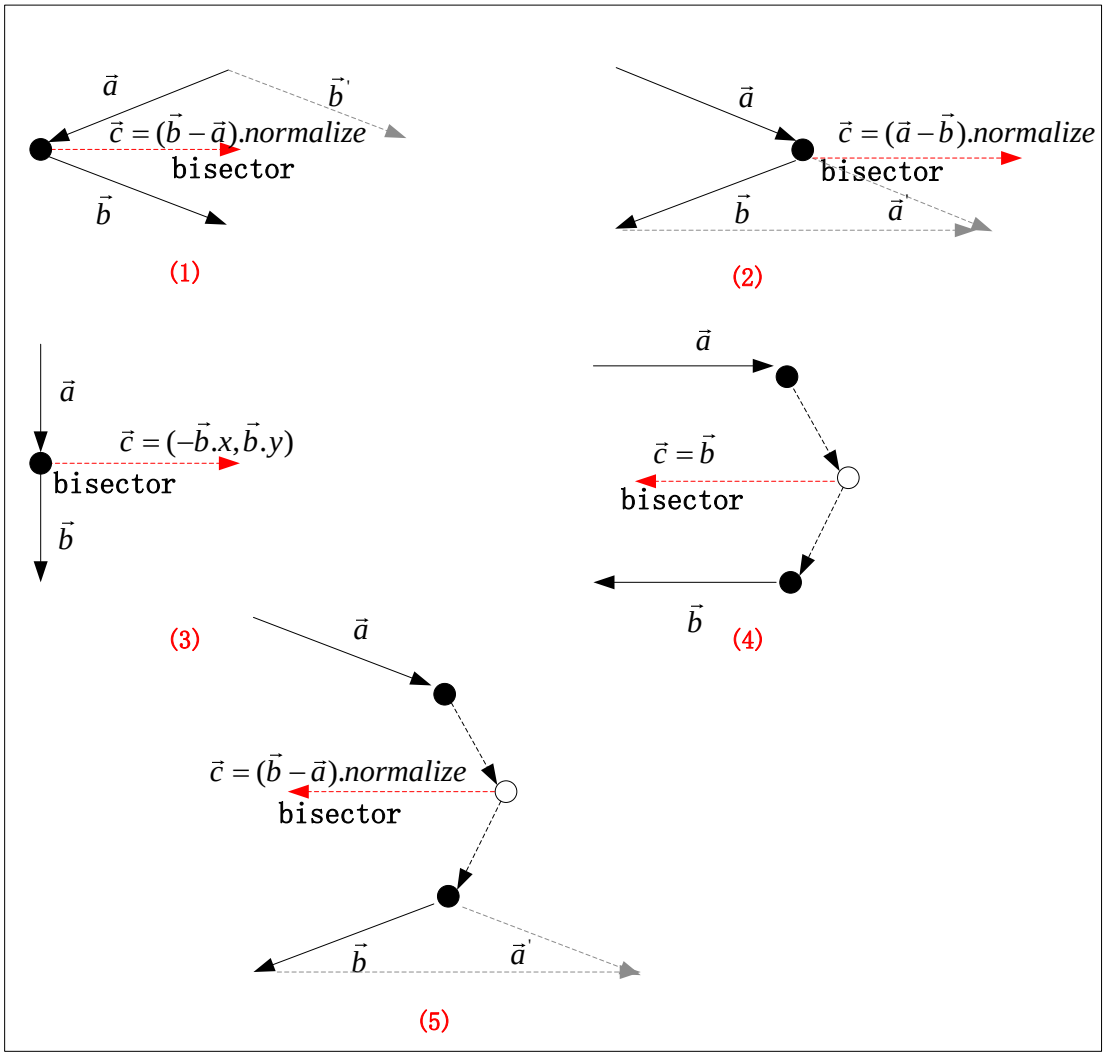


图 4 两条轮廓边的角平分线计算（情况（1）两条边是相邻非平行，且顶点是凸点；情况（2）的两条边是相邻非平行，但顶点是凹点；情况（3）的两条边是相邻且平行；情况（4）的两条边非相邻且平行；情况（5）的两条边是非相邻且非平行）。

扫描线算法

初始化

初始化工作主要是计算每个顶点的角平分线，以及算出该角平分线与相邻两个角的角平分线的交点哪个离对应的边更近，选取距离较近的交点（图 5 中的点 V_1）压入优先队列中，这是凸点的情况（图 5 中的点 A）。

如果是凹点的话（图 5 中的点 B），还要计算该点可能发生 split 事件的分裂点到对应的边的垂直距离，选择距离最短的分裂点（图 5 中的点 V_2）压入到栈中。计算这个距离的过程略微复杂，详细过程在后面会进行描述。

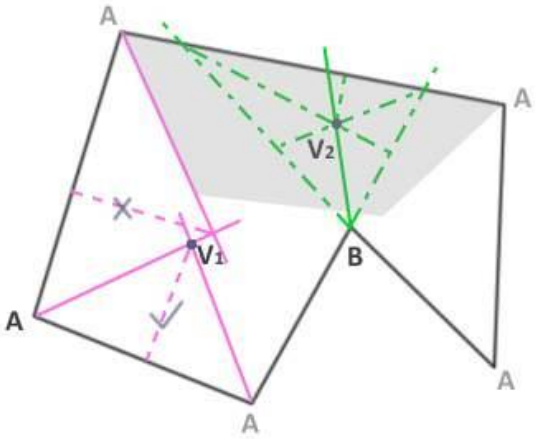


图 5　V1 和 V2 为被放入优先队列的交点（其它 A 点对应的交点未在该图中画出）

此外，初始化的工作还包括将轮廓上的顶点按顺序通过指针 prev，next，actPrev，actNext 相邻，在初始化的时候 prev 和 actPrev 同样指向逆时针方向的上一个顶点，next 和 actNext 同样指向逆时针方向的下一个顶点。在计算骨架的过程中，actNext 和 actPrev 指针会指示出当前活动的顶点，而 next 和 prev 指针会指示出与当前点相关的轮廓上的点的位置，原本就在轮廓上的点的 next 和 prev 指针从初始化之后一直保持不变。

具体算法（步骤）如下：

A. 将轮廓的顶点数据按顺序存放在一条多向链表中，并给指针 prev，next，actPrev，actNext 赋值，使得这些顶点正确相连，如果多边形内部有洞的话，每一个洞上的顶点都存为一条链表，最后将所有链的头指针保存在一个头指针列表中；

B. 对所有的顶点（也包括洞上的顶点）计算角平分线；

C. 建立一个“距离”越小优先级越高的“交点”优先队列 Q ，对所有的顶点，若该点为凸点，则计算它所对应的角平分线与相邻的两个顶点的角平分线的交点，选取到对应边的垂直距离较小的顶点放入 Q 中，如图 5 中粉色的线条所示，我们称这种顶点对应的事件为 edge 事件；若该点为凹点，则除了要进行上述操作以外，还要针对可能发生的 split 事件进行分裂点的计算，如图 5 中绿色的线条所示，从 B 点沿两侧的边的反方向发出两条射线，对所有的轮廓上的边进行相交测试，对于满足如下条件的边：

1) 与两条射线均相交；

2) 方向正确，例如在图 5 中，B 点的对边 AA 的方向必须是从右至左的。

分别计算两条射线与它们中的每一条边围成的三角形的角平分线的交点，此时需要进一步的判断，判断计算出的交点是否在对边的两个顶点对应的角平分线包围的区域里，即图 5 中的

判断 V_2 是否在灰色区域里，若不在，则排除该点，若在，则该点为一个候选点。得到所有的候选点之后，选出其中到达对边的垂直距离最小的点，放入 Q 中。

计算骨架

计算骨架这部分主要的流程是每次从优先队列 Q 中 pop 出一个顶点，然后对其进行处理，将产生出的骨架的边记录下来，并更新相关顶点在多向链表中的连接关系。

具体算法（步骤）如下：

重复以下步骤直至队列 Q 为空：

A. 从 Q 中 pop 出一个顶点 V，如果该顶点为 edge 事件产生的交点且产生该交点所对应的两个顶点至少有一个已经被处理过的话，那么重复步骤 A；如果该顶点为 split 事件产生的分裂点且产生该分裂点所对应的凹点已经被处理过的话，那么重复步骤 A；

B. 如果 V 为 edge 事件产生的交点：

1) 设产生 V 的两个顶点为 V_a 和 V_b，输出边 (V, V_a) 和 (V, V_b) 到记录骨架的边列表中，并标记 V_a、V_b 为已访问。如果 V_a 的 actPrev 的 actPrev 指针指向 V_b 的话，说明 V_a，V_b 的活动链上只剩下三个顶点了，那么也将边(V, V_a->actPrev)进行输出，并标记 V_a->actPrev 为已访问；

2) 更新链表指针：

   V->next ← V_b->next

   V->prev ← V_a->prev

   V->actNext ← V_b->actNext

   V_b->actNext ← V

   V->actPrev ← V_a->actPrev

   V_a->actPrev ← V

3) 对新顶点 V 计算角平分线，并比较它与活动链上相邻的两个顶点（即 actPrev 和 actNext 指针所指向的顶点）的平分线的交点到对应的轮廓上的边的垂直距离，将较小的放入队列 Q 中；

C. 如果 V 为 split 事件产生的分裂点：

1) 设产生 V 的凹点 B 为 V_a，对边的起点为 V_b，输出边(V, V_a) 到记录骨架的边列表中，并标记 V_a 为已访问；

2) 由于 split 事件会将 V_a 所在的活动链分成两个链，因此建立两个新顶点 V_1 和 V_2，它们的位置是相同的，但是在链表中的连接关系是不同的，两个点属于不

同的活动链，如图 6 所示。

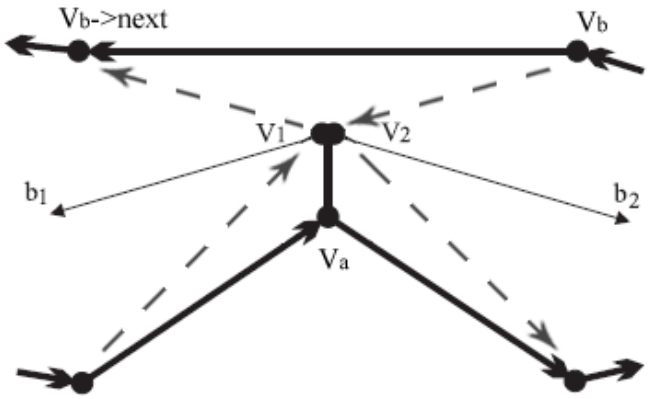


图 6　split 事件处理示意图，虚线为活动链

更新链表指针：

图 6 所示是一种最简单的情况，在这种情况中 V_b 和 V_b->next 还尚未被处理，但是在多数时候，V_b 或者 V_b->next 可能已经被处理，不在活动链上了，这时候的处理过程会比较复杂，有时候还会出现多次 split 的现象，关于上述这些复杂情况的处理方法，请见接下来的轮廓边被多个凹点所分裂的情况分析，在这里只描述如图 6 所示的最简单的情形下链表的连接关系。

V_1->next ← V_b->next

V_1->prev ← V_a->prev

V_1->actNext ← V_b->next

V_b->next->actPrev ← V_1

V_1->actPrev ← V_a->actPrev

V_a->actPrev->actNext ← V_1

V_2->next ← V_a

V_2->prev ← V_b

V_2->actNext ← V_a->actNext

V_a->actNext->actPrev ← V_2

V_2->actPrev ← V_b

V_b->actNext ← V_2

3) 如果 V_1 的 actPrev 的 actPrev 指向 V_1 自己，则说明在 V_1 所在的活动链中只剩下两个顶点，那么输出边(V_1，V_1->actPrev)到骨架的边列表中，并标记 V_1、

V_1->actPrev 为已访问；如果 V_2 的 actPrev 的 actPrev 指针指向 V_2 自己，则执行相似的操作。如果 V_1，V_2 都被处理了，那么重复步骤 A；

4) 对于 V_1，V_2（在上一步中被处理的顶点忽略该操作）分别计算所对应的角平分线（图 6 中的 b_1，b_2），并比较它与活动链上相邻的两个顶点（即 actPrev 和 actNext 指针所指向的顶点）的平分线的交点到对应的轮廓上的边的垂直距离，将较小的放入队列 Q 中。

5)

退化情况分析

两条角平分线平行时（bisector）计算交点到相应轮廓边的垂直距离时产生的问题。这时用两条直线的求交计算得到交点到轮廓边的距离，能得到一个用 double 类型无法表示的数正数，但是也有可能得到一个用 double 类型无法表示的负数，而我们产生一个新的事件点，加入优先队列时所使用的键值是用距离来衡量的，然后得到的是一个大负数，我们也同样把该事件点加入事件队列中，以至于这个事件点位于优先队列的最小堆的顶部，且接下来就会发生该事件，这样会产生一个错误的事件，以至于生成的骨架不正确。为了保持算法的正确执行，我们再遇到这种情况的时候，不让其加入优先队列，这样就可以避免这种错误。如图 7 中的V_a，V_b所产生的角平分线Bi_a, Bi_b平行，当计算得到的交点到相应的轮廓边的距离$H_a = -\infty$或$H_b = -\infty$时，如果不排除就会产生问题。

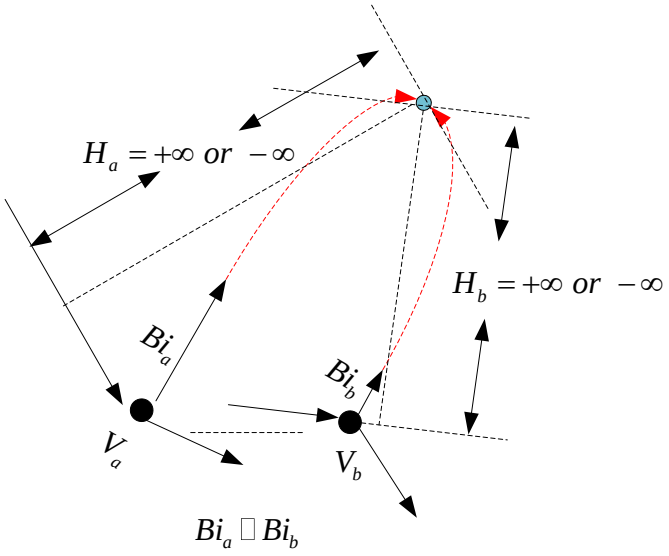


图 7 Bisector 平行产生的问题

轮廓边被多个凹点所分裂的情况分析

在多边形的轮廓边在扫描线算法执行的过程中往里面推的时候，同一个轮廓上的边被多个凹点产生的 split 事件所分开时，这时候可能把所有的 active 所组成的环分为多个子环，如图 8 所示，轮廓边 YZ 第一次被中间的那个凹点所分裂在 X 点，那么如果接下来左边的凹点 V 点又把 YZ 边所分裂，这时候你无法知道 V 点在数据结构上所分裂的环是左边的子环 YXMN 还是右边的子环，如果是遍历整个每个子环，然后确定 V 点要分割的子环，这时需要 O(N)的时间复杂度。

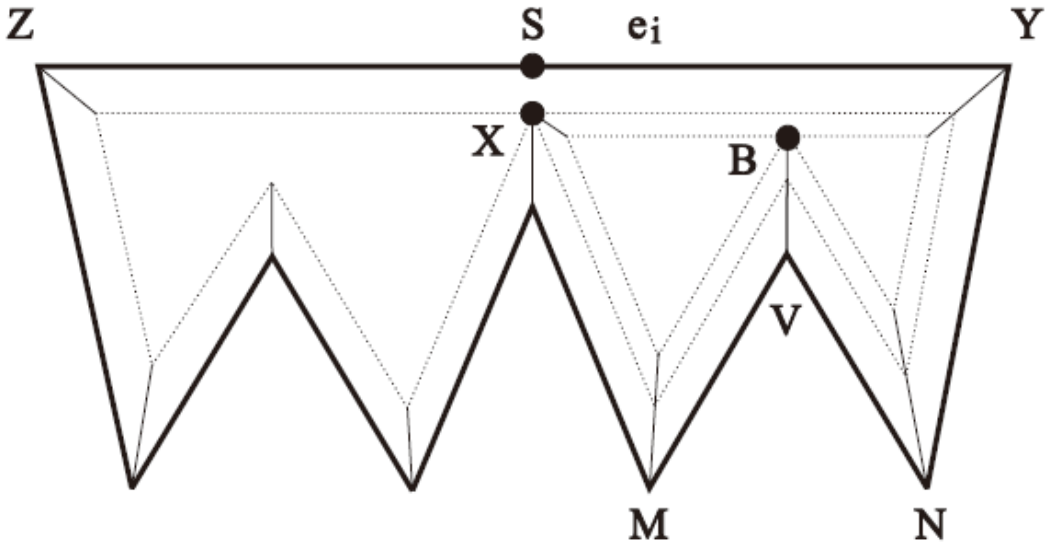


图 8 轮廓边 YZ 在往里面推的时候被多个凹点所分裂

但是通过观察可以发现，通过事件点 B 在 YZ 上的投影到 Y 的距离可以很快的定位 V 点所要分裂的环（如图 9 所示，为了计算方便我们再算法的实现过程中，把 Y 点的 t 值设为负无穷，把 Z 点的 t 值设为正无穷），我把这些 t 值存储成一个 BBST，我们使用了 STL 中的 set 来方便的进行操作，这样的话在 O(lgn)的时间内就能定位到所分裂的环上，大大的提高了算法执行的效率。

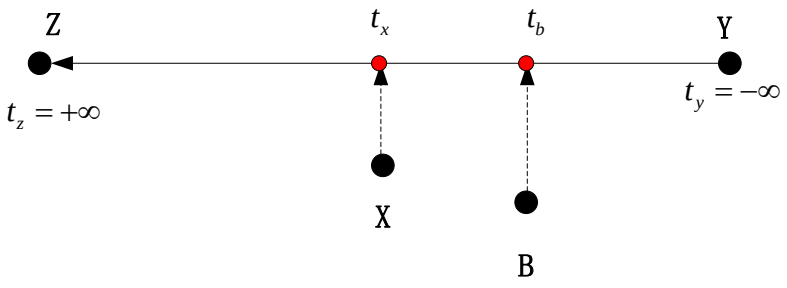


图 9 事件点在要分裂的轮廓边上的投影

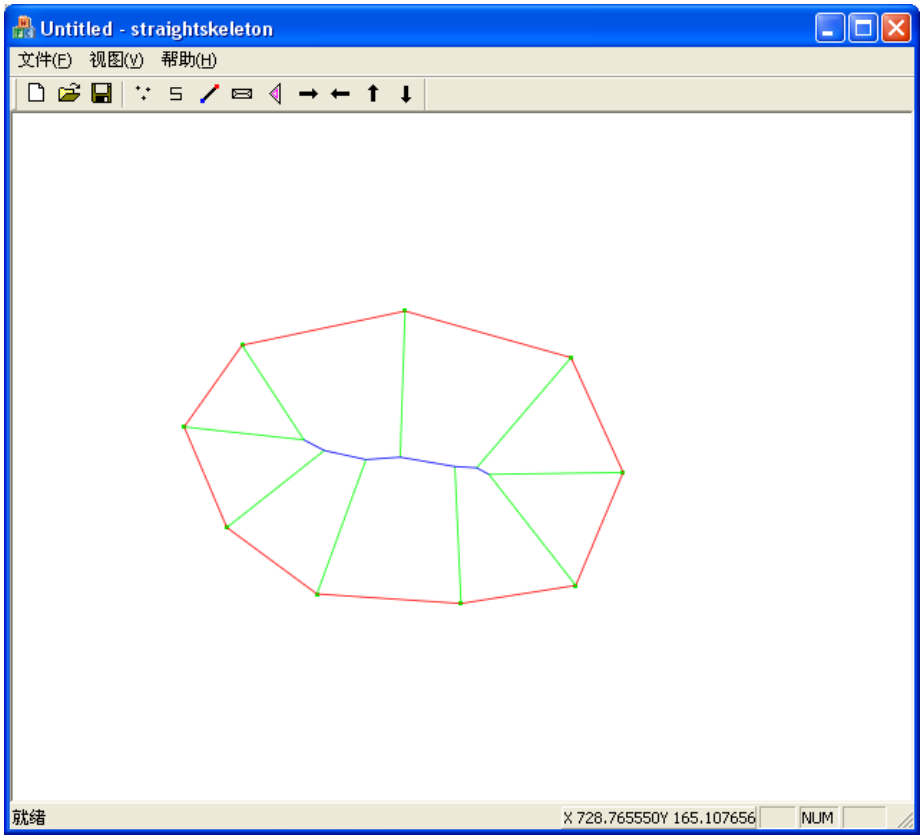
时间复杂度分析

在初始化的时候，我们要对每个凹点计算其可能产生产生 split 事件的轮廓边，每次都要遍历整个链表，因此总的时间为 O(mn)，m 为凹点的个数，n 为顶点个数。优先队列的建立使用的最小二叉堆，而且初始化的时候事件的个数为 O(n)，因此优先队列初始化为 O(n)。每次从优先队列中去下一个将要发生的事件需要 O(lgn)的时间，每个事件发生后所产生的新的事件点为 O(1)个，因此每个事件发生后所产生的新事件插入到优先队列的事件复杂度为 O(1)*O(lgn)=O(lgn)。由于整个多边形的骨架点数量级为 O(n)，因此扫描线算法复杂度为 O(nlgn)，所以整个算法的时间复杂度为 O(mn+nlgn)。

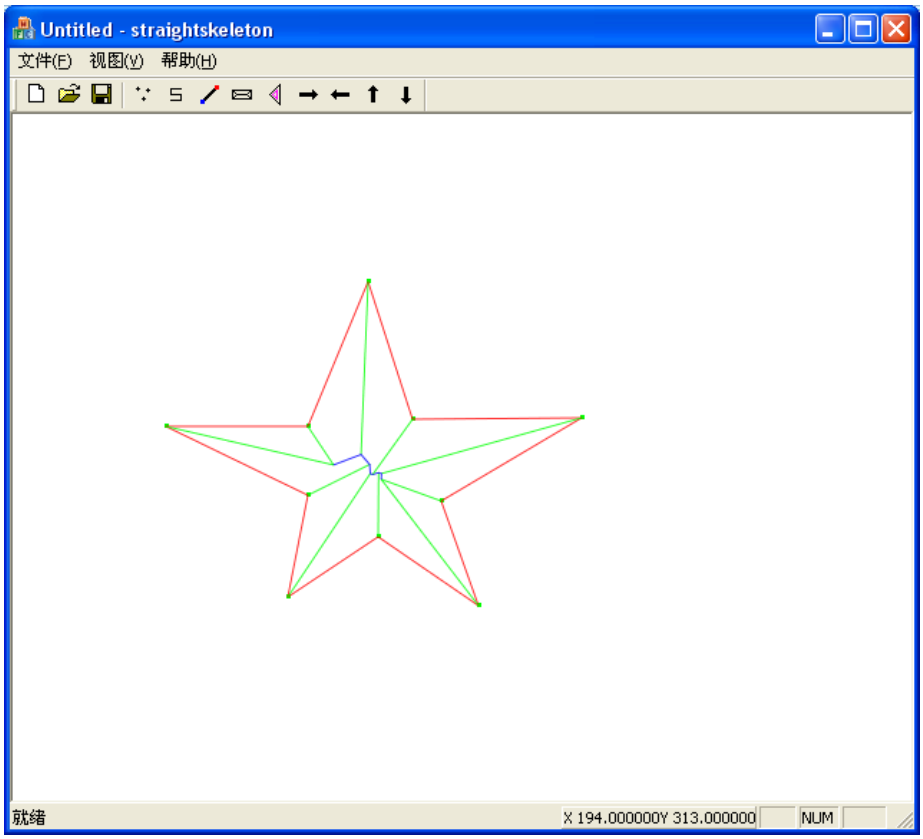
实验结果

平面简单多边形直骨架算法结果

a) 凸多边形情况

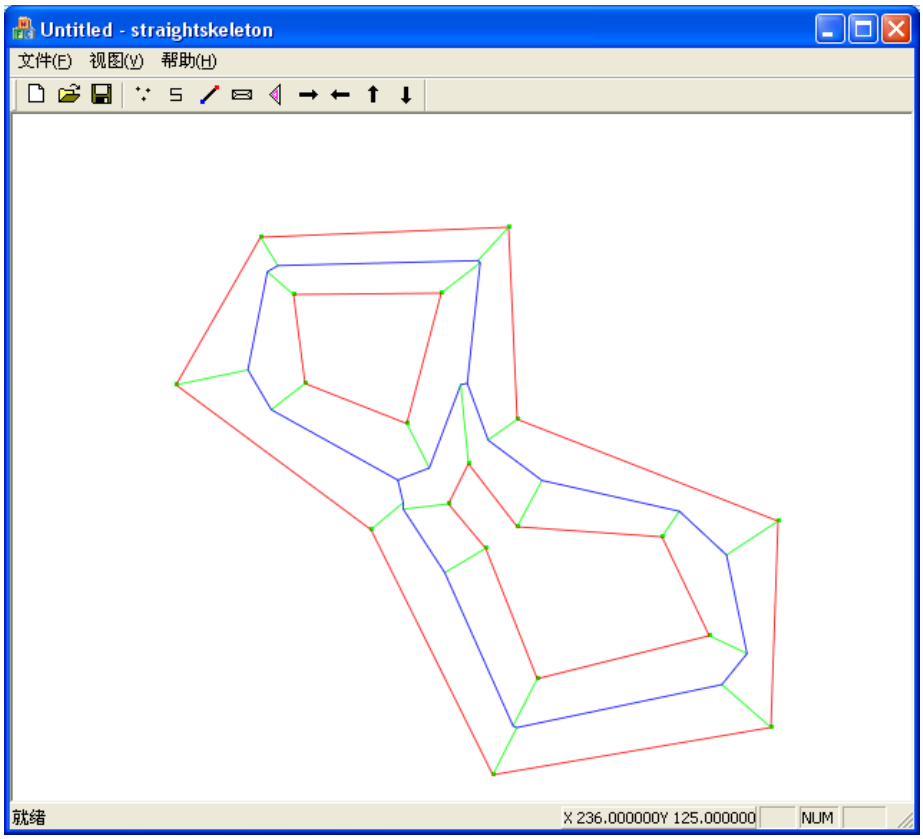


b) 凹多边形情况

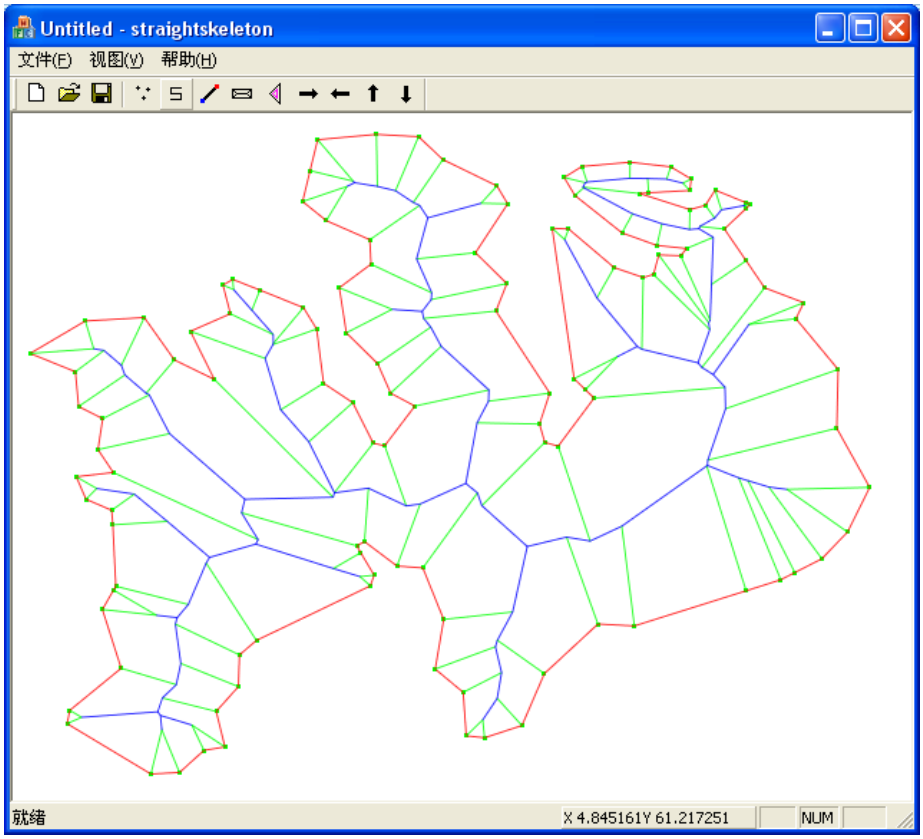


c) 带洞多边形情况

对于带洞情况，外轮廓的线方向为逆时针，而内轮廓的线方向为顺时针。这样就可以把外轮廓和内轮廓统一起来。

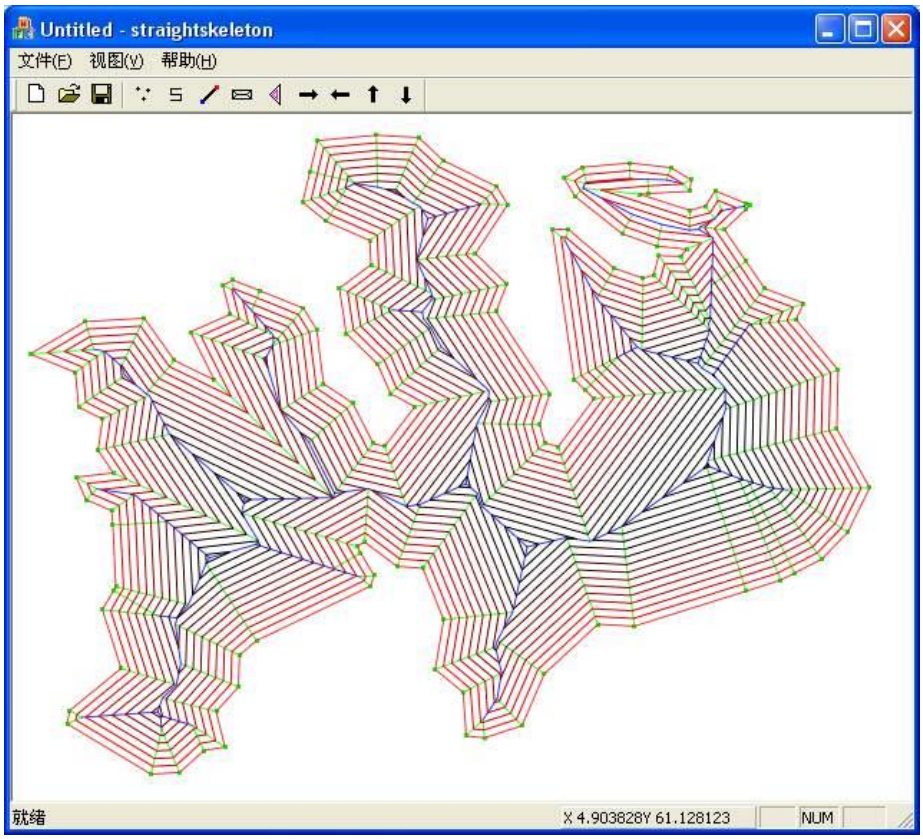


d) 较复杂多边形情况



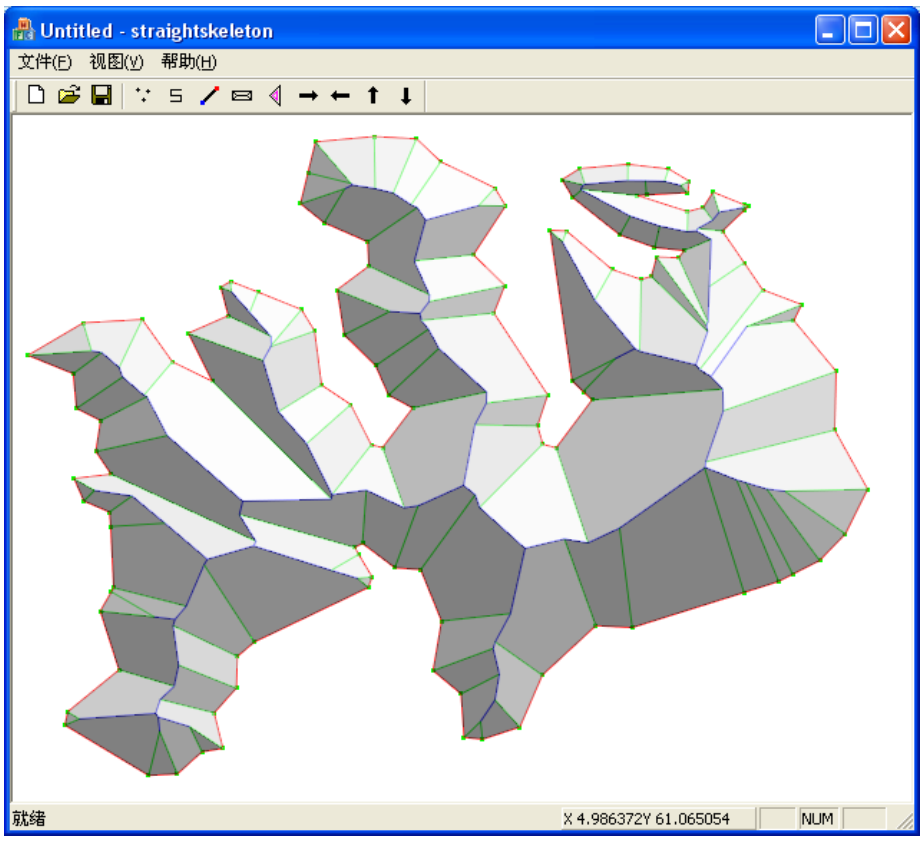
e) 等高线演示结果

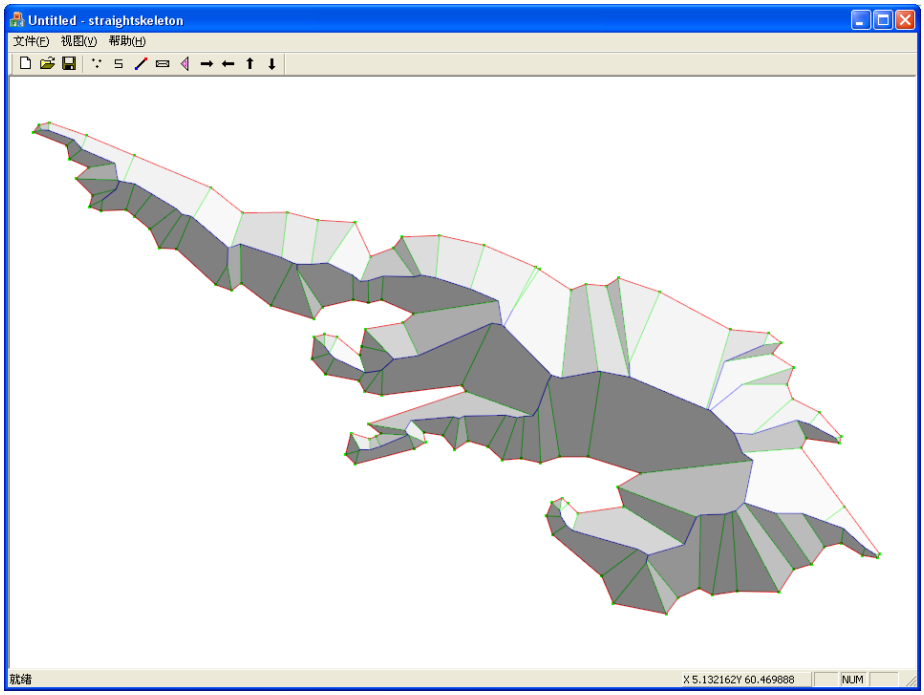
通过等高线动态生成的演示，可以形象地看出直骨架算法形象的另一面。

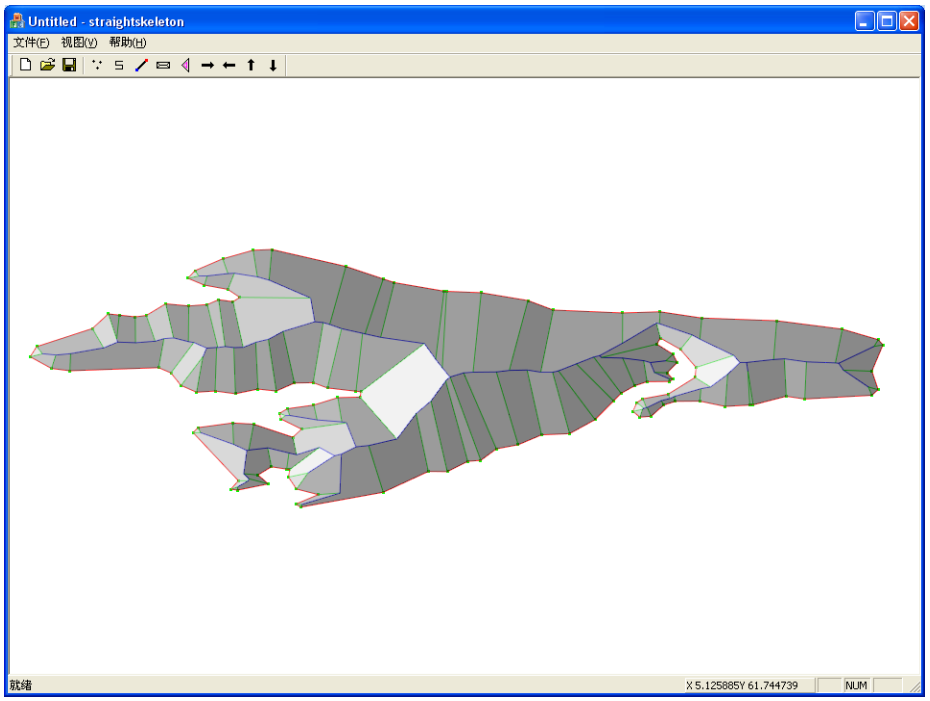


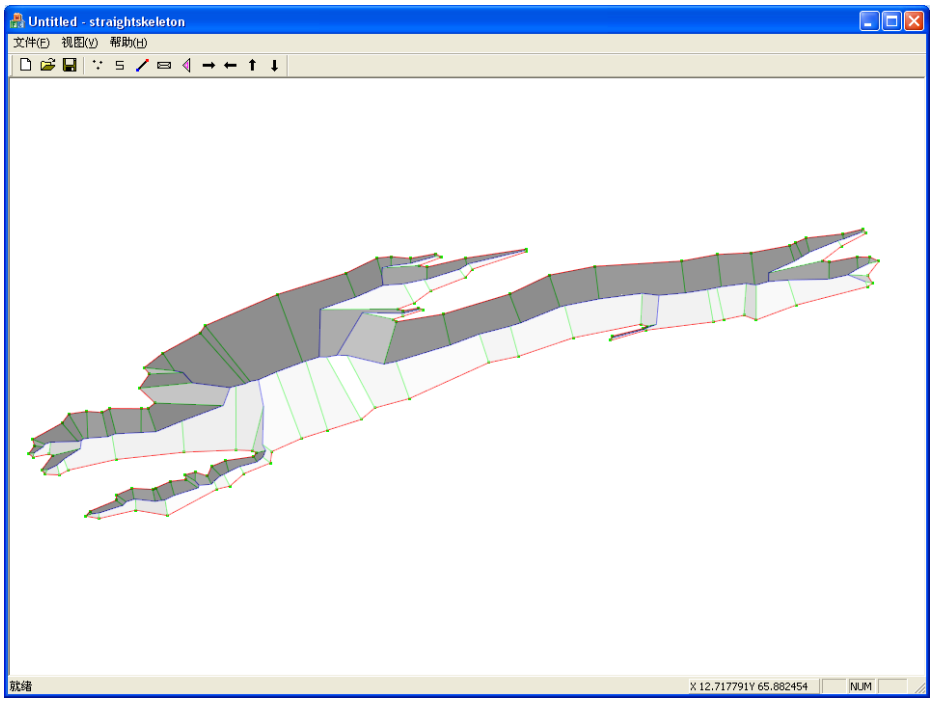
f)　三维屋顶演示效果

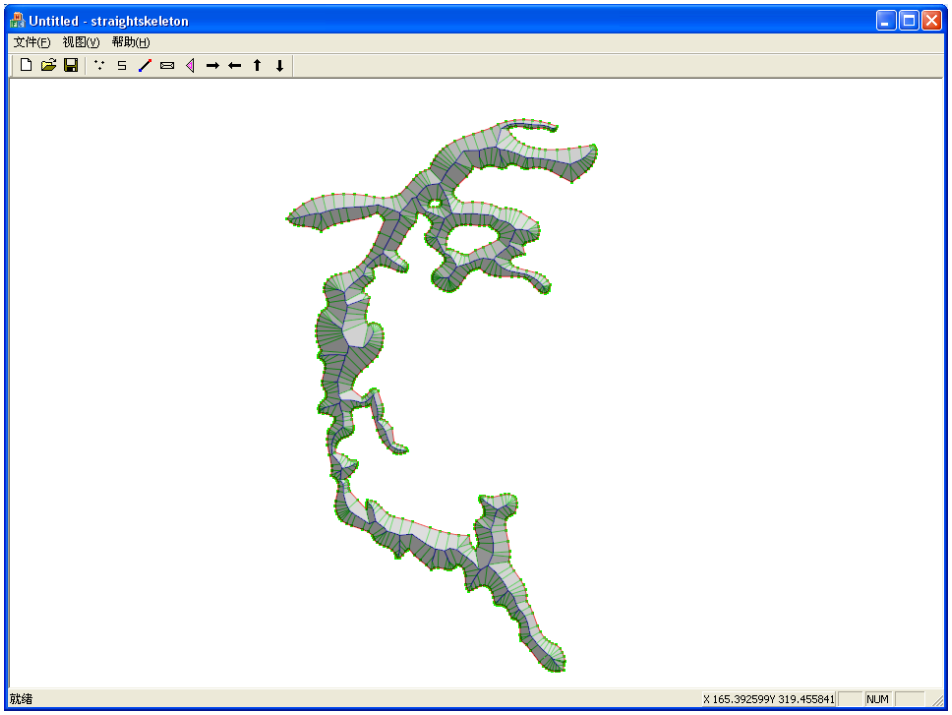
直骨架算法可以运用到三维地形的制作中。通过此效果演示，大致可以表现出凹凸感。在这里，先由面的轮廓边算出该面的三维空间中的法线，然后结合平行光的方向，即可得出光照效果图。系统支持改变光源方向，更加形象的表现此效果。

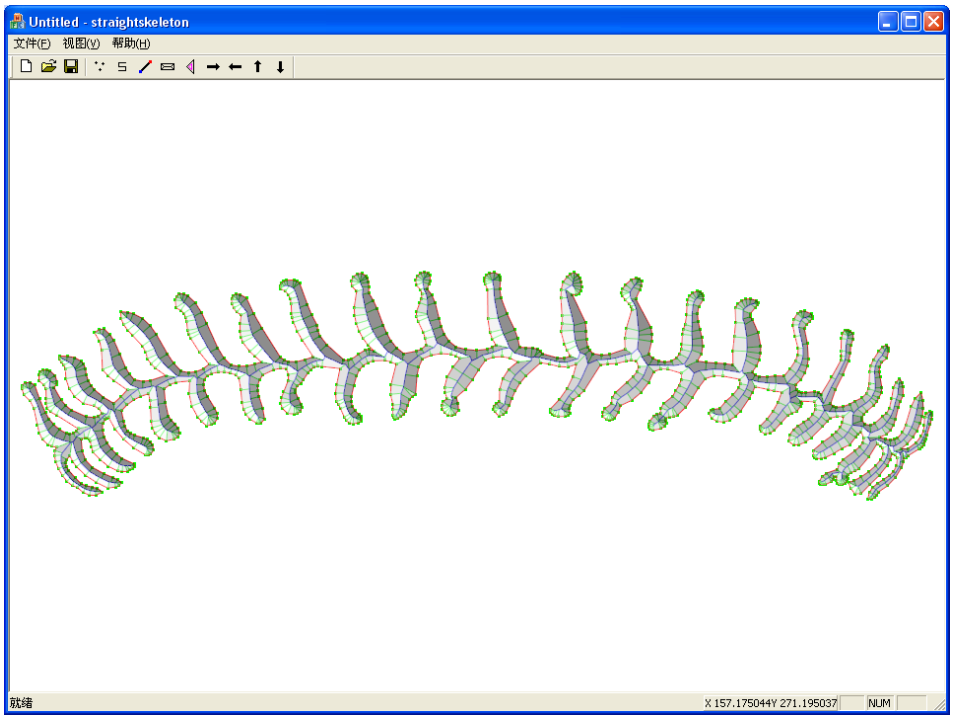





Untitled - straightskeleton
文件(F) 视图(V) 帮助(H)
就绪
X 5.125885Y 61.744739
NUM


Untitled - straightskeleton
文件(F) 视图(V) 帮助(H)
就绪
X 12.717791Y 65.882454
NUM





实验操作视频演示网址为：http://v.youku.com/v_show/id_XMTM5ODMyMDA0.html

算法性能测试：

论文[1]指出，该算法的时间复杂度为 O(nm+nlogn)，其中 n 为多边形的总顶点数，m 为凹点(reflex vertices)数目。因为 m <= n，所以 O(nm+nlogn)的上限为 O(n^2)。

本次实验对性能测试所采取的多边形生成方法为：

```
for (int i = 0; i < count; ++i)
```

```
{
    double x = i + random();
    double y = random() + random();
}
for (int i = count-1; i >= 0; --i)
{
    double x = i + random();
    double y = -random() - random();
}
```

其中 random()产生 0 到 1 之间的随机数，通过此方法可以得到 2*count 个点，先后连接后组成一个简单多边形，并且凹点数目相当多。将此多边形输入到系统，得到 10 组测试数据为表格 1。通过时间消耗和多边形顶点数目的关系图可以看出，时间消耗和顶点数目的二次方近似成正比。

表格 1 算法时间复杂度

点数	1k	2k	3k	4k	5k	6k	7k	8k	9k	10k
时间	66	250	561	964	1512	2173	2966	3875	4887	6023

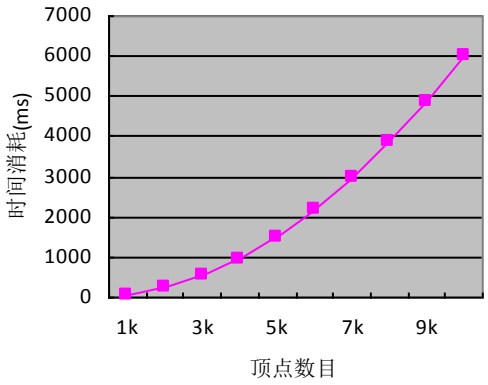


图 10 时间消耗和顶点数目的关系图

用户手册

界面整体印象

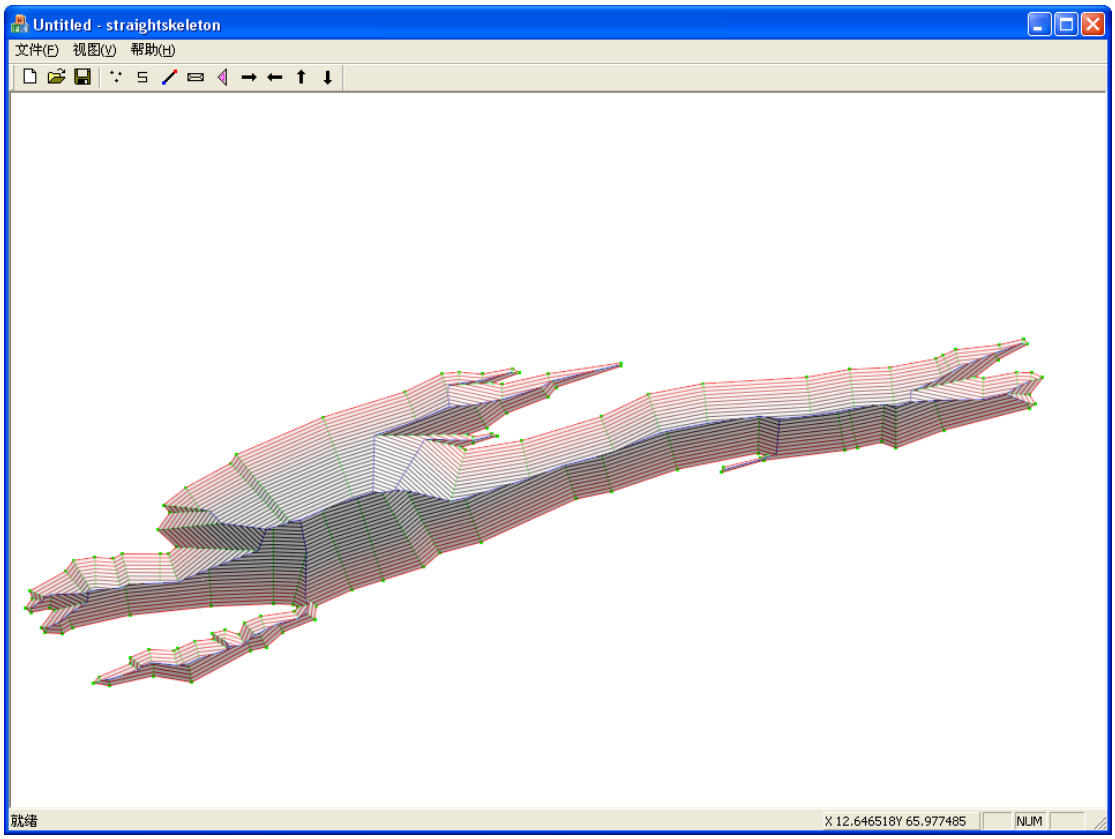


图 11 界面整体印象

界面操作说明

- （或者“文件”->“新建”）：将已存在的数据和界面图案进行清除，重新初始化。
- （或者“文件”->“打开”）：打开和读入保存在文件中的平面简单多边形的数据，此为输入简单多边形的方式之一。
- （或者“文件”->“保存”）：保存界面中显示的简单多边形数据到文件中。文件格式见下面说明。
- ：手动输入简单多边形数据。点击此图标后，就可以在界面上单击鼠标，先后输入多边形的顶点，结束时单击右键，就可以把最后一个点和该多边形的第一个边连接起来。支持输入有空多边形，在输入外环多边形后，使用相同方法在外环多边形里面进行输入内环多边形（即多边形的洞）。
- ：简单多边形直骨架算法。在简单多边形数据输入后，点击此图标，就可以在界面上显示直骨架的结果。红色线代表轮廓线，绿色线代表跟轮廓线上点相连的骨架线，蓝色线代表不与轮廓线上点相连的骨架线。
- ：对界面图形进行平移。为了方便用户观察结果，点击此图标，按住鼠标左键拖动，就可以实现对界面图像的平移操作。
- ：三维屋顶演示效果。在进行直骨架算法操作后，点击此图标，就可以显示直骨架

结果的三维模拟效果。

-  ：显示简单多边形等高线生成过程。在进行直骨架算法操作后，点击此图标，就可以动态显示多边形的等高线组生成的过程。生成方向为从轮廓边往多边形内部推进。
-  ：改变平行光方向。在三维屋顶演示的时候，分别选择这几个图标，就相应改变平行光方向，使得三维模拟效果更加明显。另外，在三维屋顶演示的时候，还可以按住鼠标左键拖动来改变光源的方向，更加方便显示三维效果。
- 鼠标滚轮操作：对界面图形进行放缩操作。此操作和平移操作相结合，可以方便用户观察整体效果和局部细节。

简单多边形数据文件格式说明

(内外轮廓边总数) [换行] * integer
(外轮廓顶点数) [换行] * integer
(x-coord) (y-coord) [换行] * double
(x-coord) (y-coord) [换行] * double
(x-coord) (y-coord) [换行] * double
……
(EOF) 或者 (0 号洞顶点数) [换行] * integer
(x-coord) (y-coord) [换行] * double
(x-coord) (y-coord) [换行] * double
(x-coord) (y-coord) [换行] * double
……
(EOF) 或者 (1 号洞顶点数) [换行] * integer
(x-coord) (y-coord) [换行] * double
(x-coord) (y-coord) [换行] * double
(x-coord) (y-coord) [换行] * double
……

一个例子为：

```
3   # 轮廓边总数
5   # 外轮廓
383.000000000000000 105.000000000000000
666.000000000000000 189.000000000000000
628.000000000000000 417.000000000000000
315.000000000000000 497.000000000000000
176.000000000000000 379.000000000000000
4   # 0 号洞
291.000000000000000 398.000000000000000
415.000000000000000 424.000000000000000
415.000000000000000 333.000000000000000
326.000000000000000 326.000000000000000
4   # 1 号洞
428.000000000000000 280.000000000000000
```

547.000000000000000 323.000000000000000
613.000000000000000 203.000000000000000
437.000000000000000 218.000000000000000

参考文献

[1] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. G artner. A novel type of skeleton for polygons. Journal of Universal Computer Science, http://www.iicm.edu/jucs 1 12, Institute for Image Processing and Computer Supported New Media, 1(12):752{761, 1995.
[2] P. Felkel, et al. Straight Skeleton Implementation. Proceedings of Spring Conference on Computer Graphics 1998, Budmerice, Slovakia, pp. 210-218.

其他说明

1） 系统开发平台为 vs2005，使用了基本的 OpenGL 库，对于编译和连接无特殊要求。
2） 提交文件夹 data 中包含我们设计和 CGAL 提供的几组测试数据。
3） 测试实验的配置环境为：Intel Core 2 Duo CPU 2.2GHz, 2.00GB RAM。