几何画板与 Point Location 算法

——计算几何研究报告

小组成员

郝霄虹　2010210817　haoxiaohong.ivy@gmail.com
张　翔　2010311158　siang.zhang@gmail.com
谢凌曦　2010310509　198808xc@gmail.com

工作概述

这次的大作业中，我们计划研究一个软件与算法并重的课题，最后将我们的研究成果以软件演示的方法呈现给老师和各位同学。

我们的研究工作大致分为两个部分。

首先，我们打算构建起一个计算机几何系统（Computer Geometry System）的基础框架。由于时间有限，我们不太可能在几何系统上完成大量的科学计算。因此，我们将把主要的精力用于实现几何系统中的绘图演示部分。我们希望能够通过我们的努力，实现一个方便的演示系统。这样，我们可以将各种渠道获取的代码进行简单的处理后，以任意的分段演示的形式呈现在屏幕上。这非常有助于课堂演示或者算法效果的展示。如果过程比较顺利，我们还希望能够实现一个代码的评价系统。此时，教师可以通过这个系统发布一些算法的作业，学生通过实现某些函数来完成算法的设计。而系统可以利用共享内存中的数据进行分段的演示，并且判定算法的正确性。

其次，我们主要研究了计算几何里面的一个基础问题，平面正交区域的点定位问题。在这个问题上最新的研究成果是 Timothy M. Chan 在 2011 年的 SODA 上的一篇论文，该论文提出了平面正交区域点定位的查询时间复杂度$O(\log\log U)$，空间$O(n)$的算法，在理论上有很重要的价值。我们编程实现了该算法，并将该算法与已有的算法进行了分析比较。通过实现这个算法，不仅可以加深我们对计算几何以及某些数据结构的理解，而且还能够为上述系统的演示提供一个清晰的样例。

下面，报告将分软件方面的几何画板和平面正交区域点定位两个方面来介绍我们的工作。

简单几何画板软件

“自从 1946 年世界上第一台电子计算机问世以来，计算机应用的一个重要里程碑是 1962 年美国麻省理工学院发明了世界上第一台图形显示器。自此之后,计算机可以通过图形显示器直接输入、输出图形，并且可以在显示屏上通过光标的移动而直接修改图形。而在这之前，工程师是通过一厚叠纸上密密麻麻的数字来间接表达工程图形的。

1962 年被认为是美国和欧洲 CAD 开始发展的一年。首先的应用领域是汽车、飞机和造船工业。这 3 个行业，由于其产品的外形曲面特别复杂，要求特别苛刻，而成为 CAD 首先应用的领域。

与此同时，也就发展出了一门新兴学科——计算几何，它在美国常常被称为 CAGD(Computer Aided Geometric Design，计算机辅助几何设计)，专门研究“几何图形信息(曲面和三维实体)的计算机表示、分析、修改和综合”。1972 年在美国举行 CAGD 第一次国际会议,标志计算几何学科的形成。”

--Wikipidia

1. 问题背景

70 年代中期，CAD 在中国的造船和飞机制造工业中开始发展。在苏步青先生的组织和推动下，1982 年，中国召开了第一届计算几何和 CAD 学术会议。计算几何和 CAD 的应用领域覆盖了造船、航空、汽车、内燃机、模具、机械、服装和珠宝等广泛领域。

开发大型的 CAD 系统需要投入大量的人力和财力。无论在国际或国内，开发成功达到工业应用标准的大型 CAD 软件系统，而且在功能模块的理论能上有所创新的软件系统都是少见的。目前最常用的自动计算机辅助设计软件 AutoCAD，是由全球最大的 CAD 公司——美国 Autodesk 公司为计算机上应用 CAD 技术而开发的绘图程序软件包，现已经成为国际上广为流行的绘图工具。AutoCAD 拥有优秀的用户界面，集成了大量计算几何与其他领域的算法，具有成熟完备的功能，还提供了 AutoLisp 等语言绑定，使得用户能够用脚本语言进行一些复杂的自动化设计，或者针对特定的应用领域进行二次开发。

计算几何除了应用于 CAD(Computer Assited Design-计算机辅助设计)领域，还可以应用于 CAI(Computer Assited(aided) Instruction-计算机辅助教学)。计算机辅助教学改变了几百年来的一支粉笔、一块黑板的传统教学手段。它以生动的画面、形象的演示，给人以耳目一新的感觉。但就总体而言，计算机辅助教学不仅能替代一些传统教学的手段，而且能达到的传统教学无法达到的教学效果。比如利用计算机的动态特性表现一些动态画面、它的图画特性表现一些抽象的东西。这在一些辅助教学软件中已表现得淋漓尽致。如数学学科的《几何画板》、《数学实验室》等就是利用计算机的这些特点来实现良好的教学效果的。例如教学中演示算法的需要，几何对象的计算。目标是简化教学演示。

我们希望通过自己的一些努力，开发一个类似于《几何画板》的基本的计算机几何系统。以这个系统作为平台，用户只需按照一定的格式要求完成算法，将其载入系统后就能马上看到演示效果。不再需要花费大量的时间来学习如何设计界面，以及完成一些繁琐的工作。

2. 系统结构

系统由如下几个子模块组成：

- Java 图形界面 GeometryBoard

- 科学计算引擎内核 mU
- 计算机代数系统 maTHmU（使用旧版本内核 mU0）
- 嵌入式 Lisp-ECL
- 计算机代数系统 Maxima
- Lua 解释器

科学计算引擎内核 mU 的内容丰富、功能齐全，其独有的符号处理功能在国内处于领先地位，mU 设计了一套具有简洁语法和高效运行支持的编程语言，系统模块化程度高，可移植性强。mU 还提供了高度的可扩展性，使得在 mU 上进行二次开发非常灵活。

由于整个系统是由 C,C++，Java，mU0,mU,Common Lisp，Maxima 及 Lua 等八种语言写成的模块构成。为了实现跨语言模块之间的无缝连接，需要解决不同语言之间的函数调用，异常处理，基本数据类型和复合数据类型转换等诸多问题。为此我们首先为 mU 写了多个 C/C++扩展模块，分别实现了 mU 与 Java，ECL(Embed Common Lisp)以及旧版本 mU0 之间的无缝连接。进而通过 Java 与 Luajava 实现了与 Lua 模块之间的间接连接，通过 ECL 实现了与 Maxima 模块之间的间接连接。

3. 实现原理

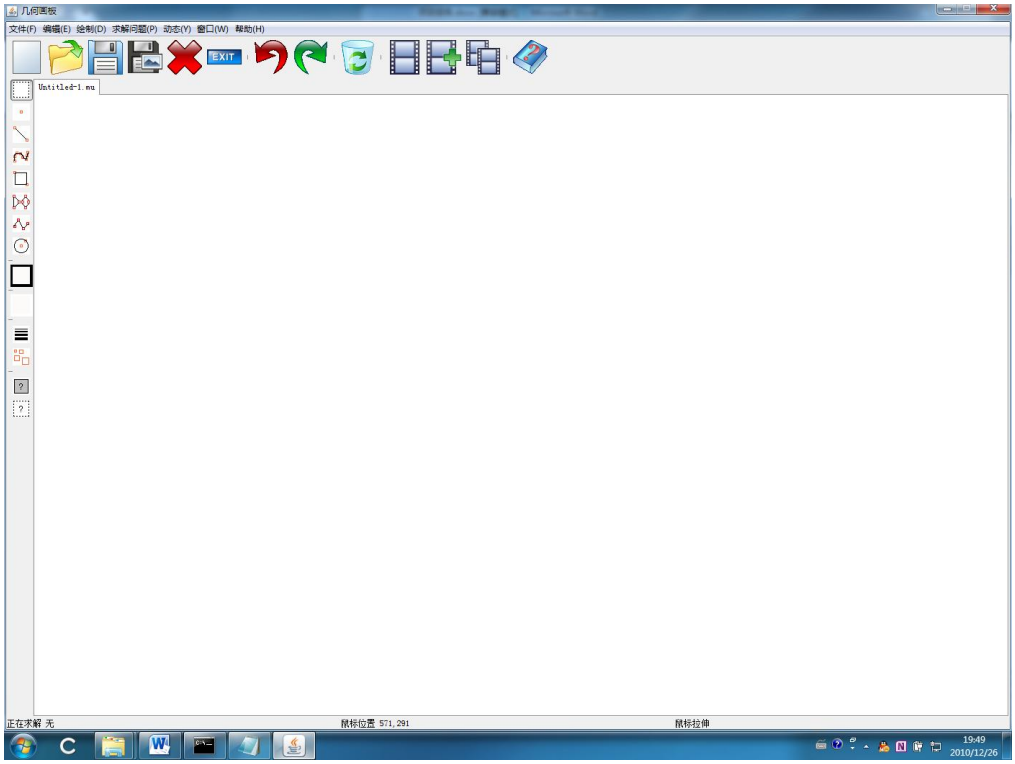
我们的几何画板程序功能非常复杂。其程序框架采用一般软件设计的通用架构，即模型-视图-控制器（Model-View-Controller, MVC）架构。其中，

- 模型部分负责存储表示文件的数据，并且经过解释器的解析，将文件转化为一棵文档树（DOM 树）。这是所有后续处理的基础所在。
- 视图部分通过对模型部分生成的文档树的解析，生成一棵视图树：它的结构与文档树的结构基本相同，只不过将绘图所需要的信息具体化，以方便绘制；随后，它调用视图树中每个节点的绘制方法，将整张图片绘制在屏幕上。
- 控制器部分负责接受用户的交互信息，并且生成程序能够接受的事件对象，供程序进行处理；在这一部分中，我们实现了丰富的处理机制，使得用户的输入能够在各种状态下呈现给程序。

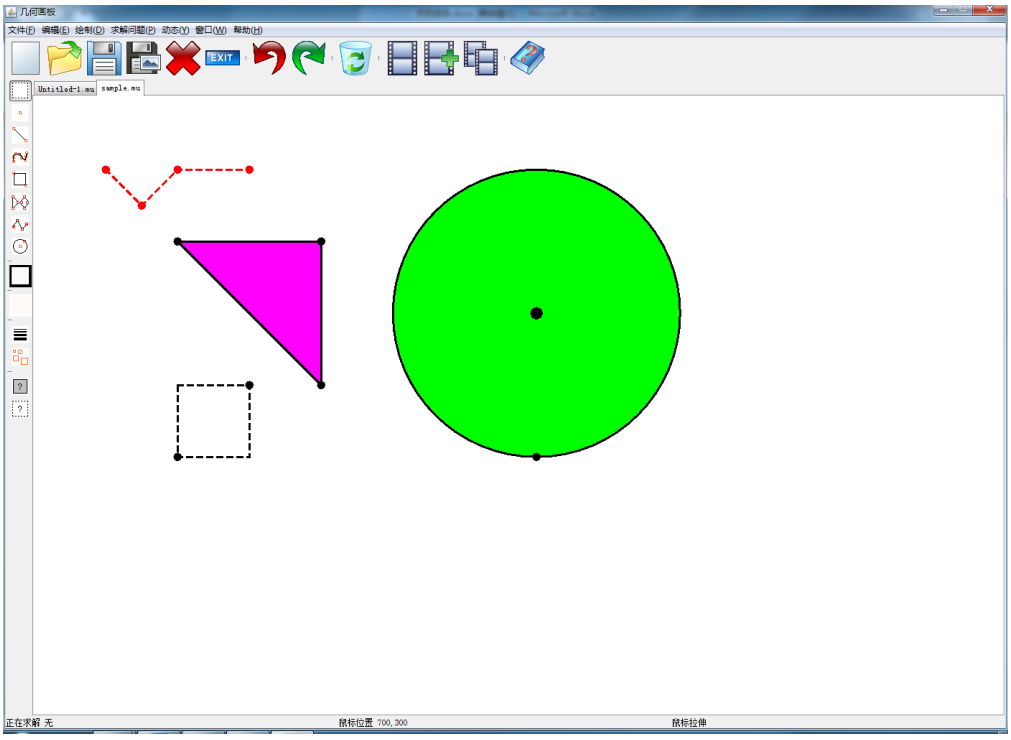
MVC 架构的实现，使得程序能够形成一个对外基本无关的整体，以方便地接受外界的查询，并向外界传递自身的信息。

4. 系统特性

3.1 基本图形绘制



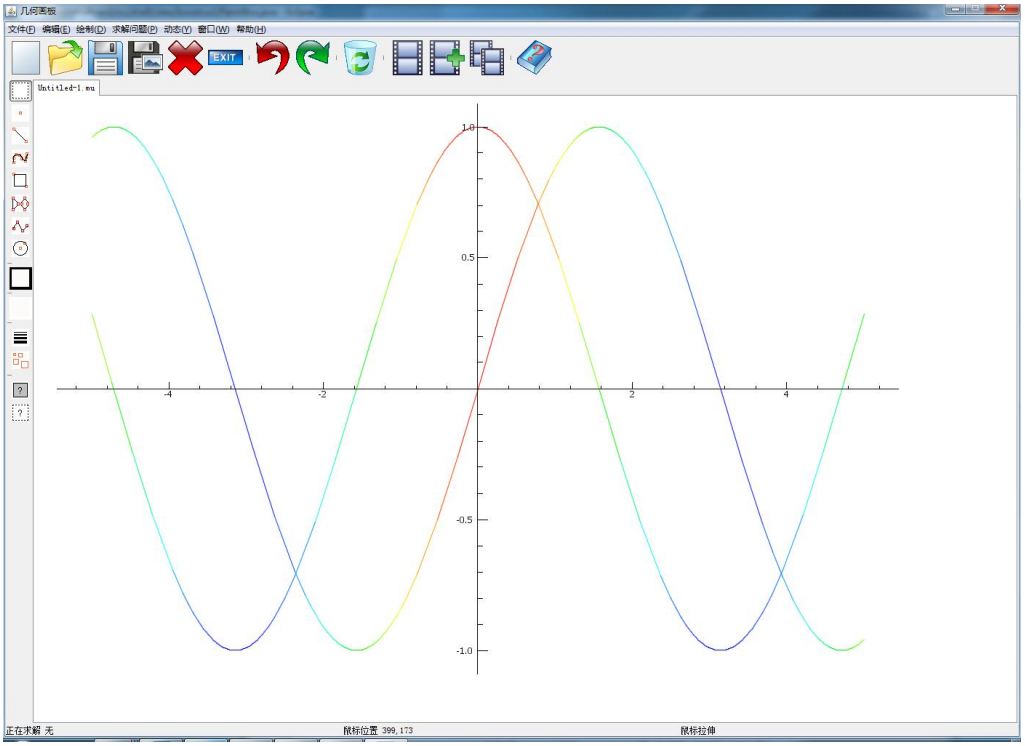
3.2 基本图形操作



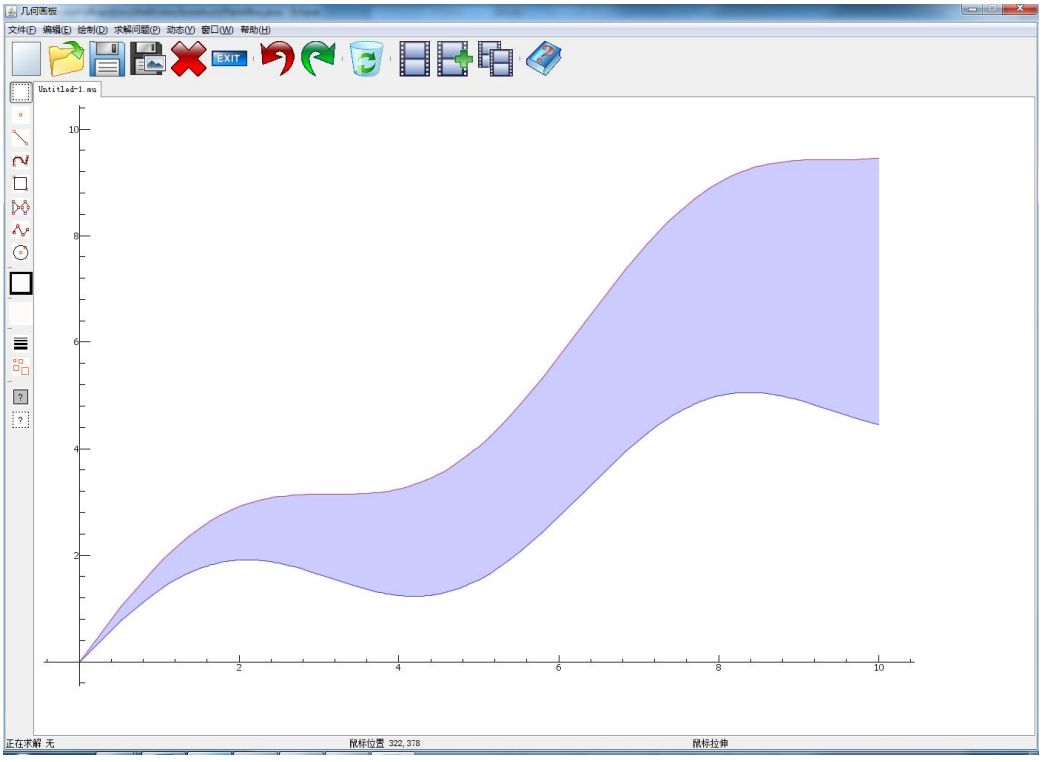
3.3 基本动画功能

(a) 二维函数曲线绘制

```
Plot[{Sin[x],Cos[x]},{x,-5,5},Options[ColorFunction[Cos[x]]]]
```

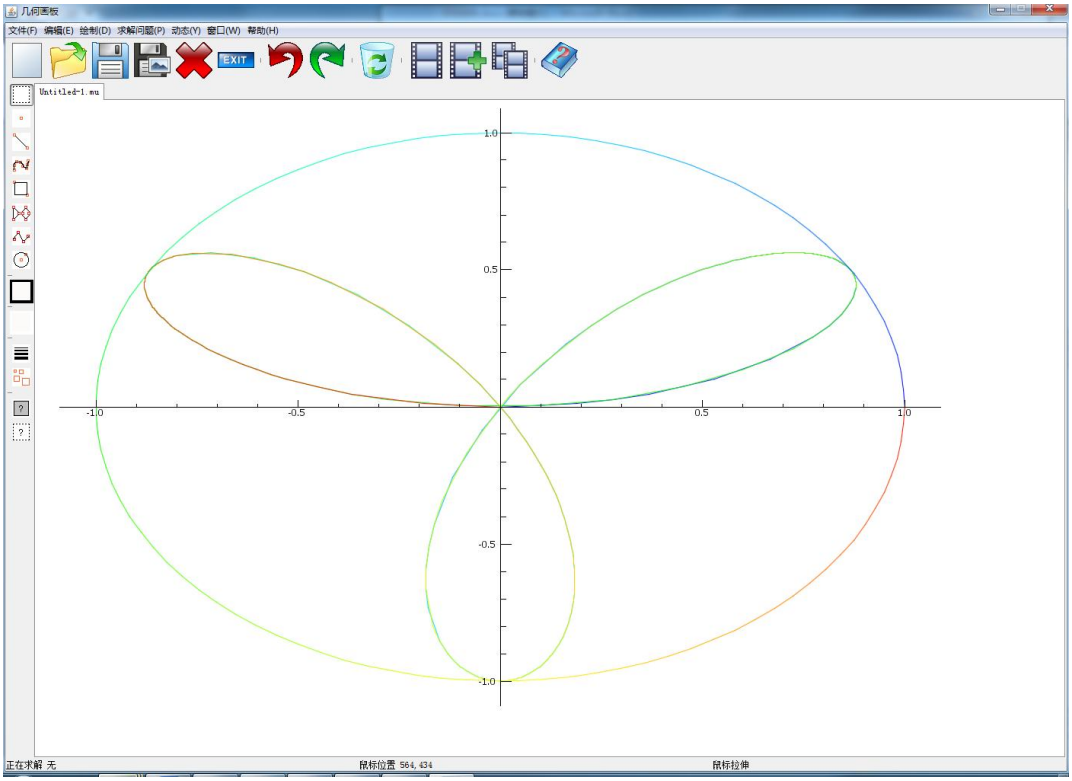



```
Plot[{Sin[x]+x/2,Sin[x]+x},{x,0,10},Options[FillToCurve->{0,1}]]
```

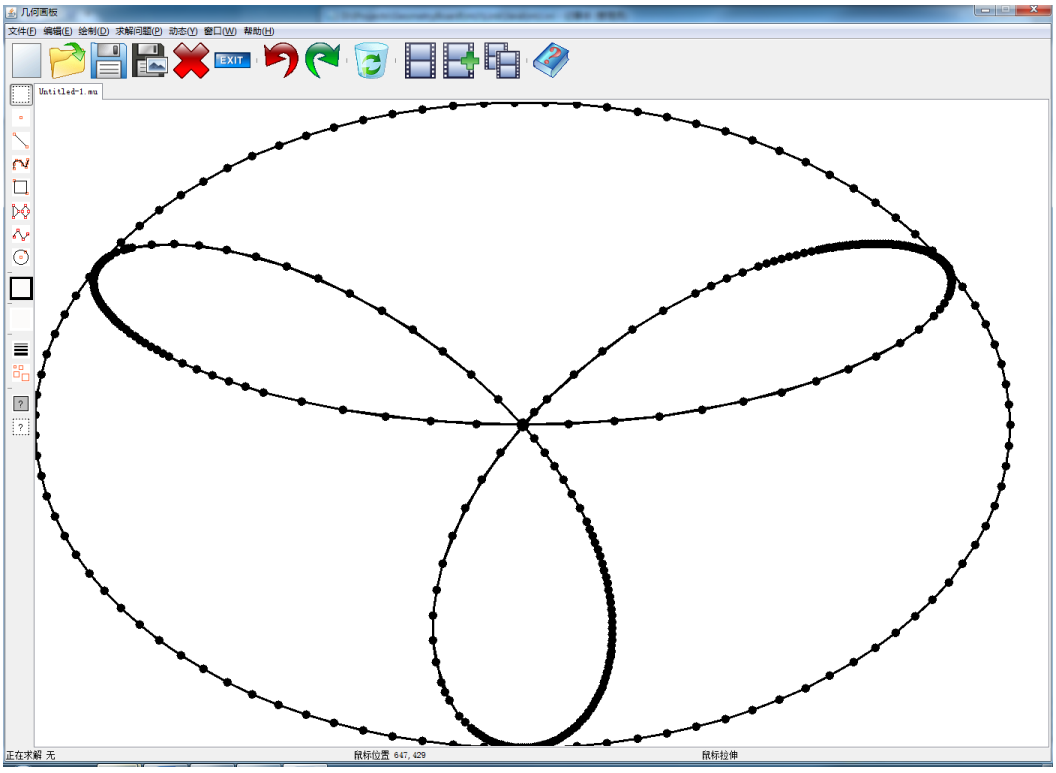


(b) 二维函数曲线的动态转换与修改

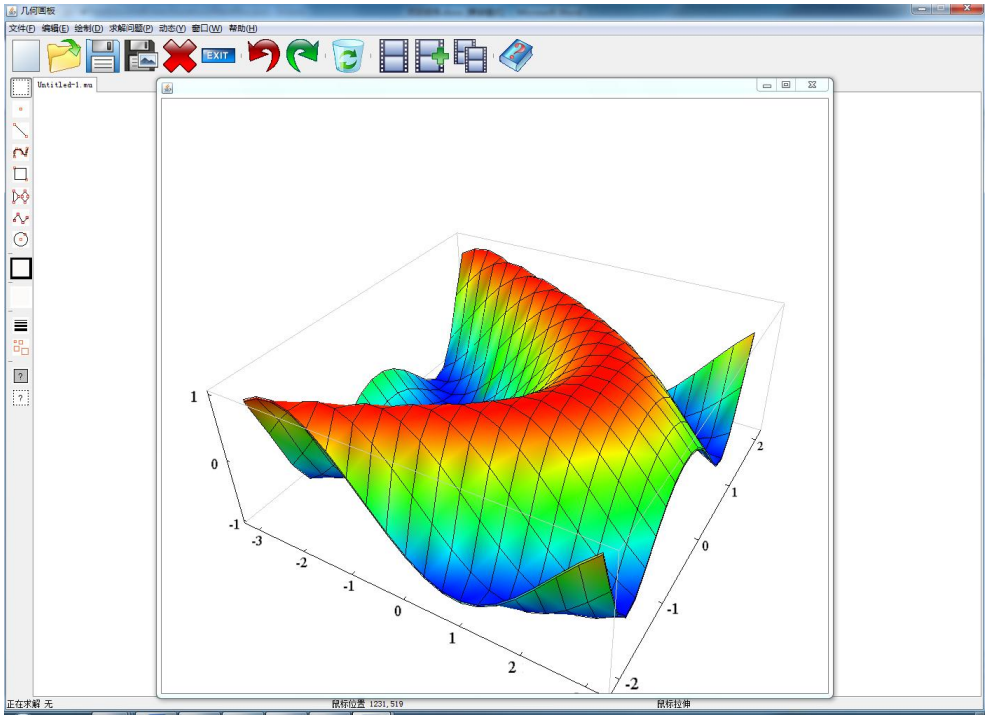
```
PolarPlot[{Sin[3t],1},{t,0,2Pi},Options[System`ColorFunction[2t]]]
```




```
PlotExpr[PolarPlot[{Sin[3t],1},{t,0,2 System`Pi}].mU[]]
```



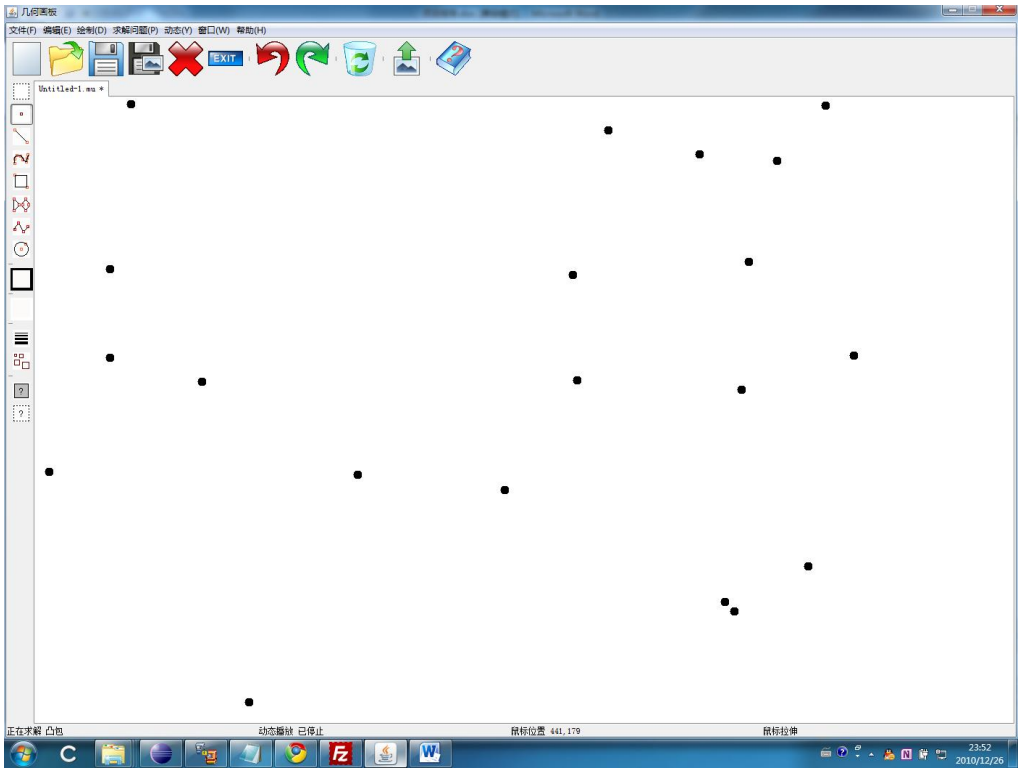
(c) 三维函数图像绘制



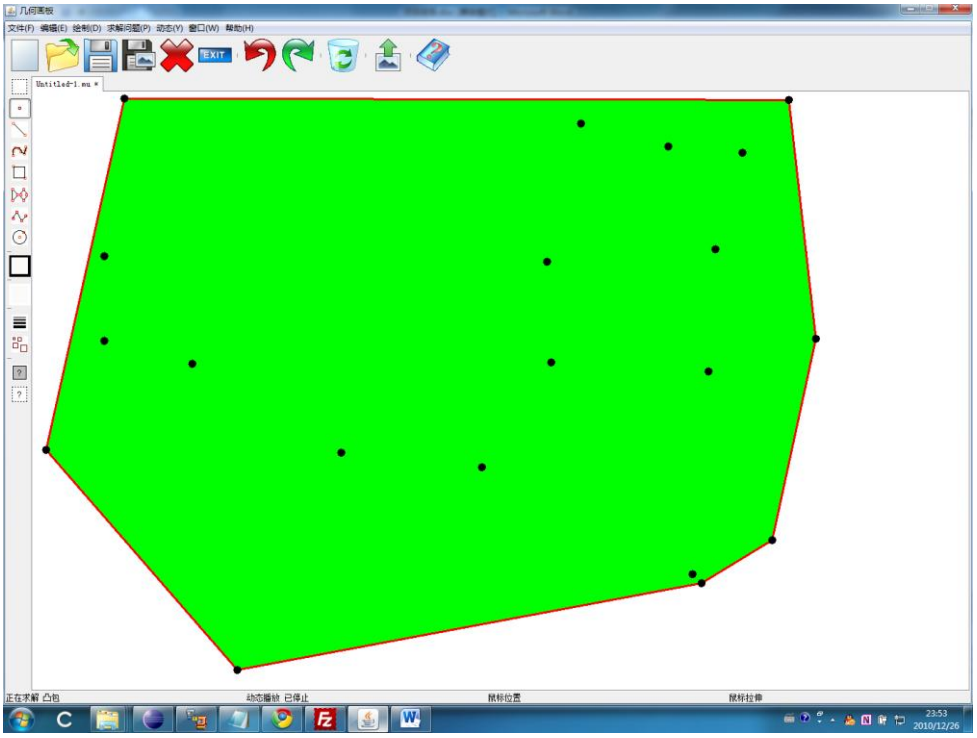
(d) 三维函数图像视角变换及动态控制

(e) 新算法的动态载入

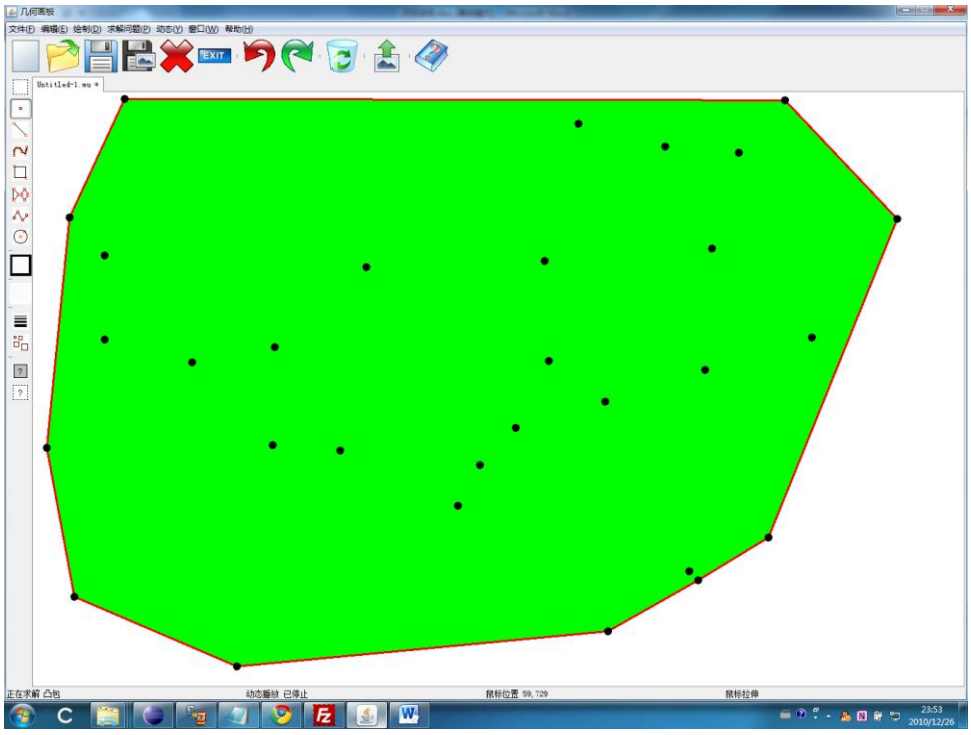
在菜单栏中，点击“求解问题”、“凸包”、“生成数据”，拖动改变生成点的数量。例如我们生成20个数据点：



点击“求解问题”、“凸包”、“求解”，自动获得问题的解。



我们还可以进行实时修改，如图所示。



(f) 界面的脚本控制

(g) 算法快速原型开发、测试框架

平面正交区域的点定位算法

1. 引言

平面点定位问题是计算几何方面的一个基本问题。它在计算机图形学、计算机辅助设计、地理信息系统等许多领域有着广泛的应用。Preparata 和 Shamos 曾对这个方面早期的研究成果做过综述。这个问题的解决方法多种多样，其中有四种各自不同的方法，性能都可以达到$O(logn)$的平均查询时间、$O(n)$的存储空间水品。这些方法是 Edelsbrunner 等人基于线段树(segment tree)和分散层叠(fractional cascading)等技术提出的链方法(chain method)；Kirkpatrick 提出的三角形细分法(triangle refinement method)；Sarnak 和 Tarjan 以及 Cole 提出的基于持续性(persistency)的方法；还有 Mulmuley 所提出的随机增量算法。

平面正交区域的点定位问题是平面点定位问题的一个特例，因此，用上面所介绍的几种方法都可以做到$O(logn)$的查询时间、$O(n)$的存储空间来解决该问题。然而，我们仍然会有疑问，这是不是就是这个算法的最后结果了。下面，我们将介绍一种在所有的坐标取值为$\{1,2 \dots U\}$中的整数点的条件下，平面正交区域点定位的一个单点最坏查询复杂度可以达到$O(loglogU)$的算法。

2. 平面正交区域点定位问题

2.1 问题定义

在二维空间中，点定位问题可以被定义为：在一个平面上给定 n 条垂直或水平的线段，将平面分成若干个多边形区域（有限或者无限）。给定一个查询点的坐标，目标是找到包含该点的多边形。

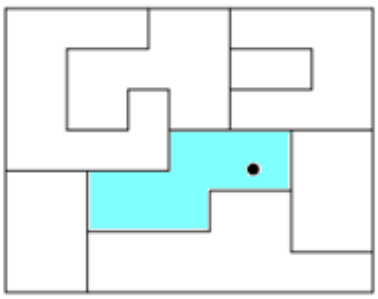


图 2.1 点定位问题：染色部分表示查询到的正交多边形区域

2.2 问题分析

2.2.1 相关数据结构--Van Emde Boas Tree

Van Emde Boas Trees 是一种可以在$O(loglog\ U)$时间内实现增、删、查找、后继及前趋节点的数据结构。给定一 n 个元素集合 S，所有元素的取值都在$\{1 \dots U\}$范围内。对于一个给定的 U，前趋查找的$O(log\ logU)$查询时间界对于任意$O(npolylog\ n)$存储空间的数据结构都是紧的。

在前趋查找中，我们需要找到集合 S 中比给定值（不一定在集合中）小的最大的元素。HASH 表可以在$O(1)$时间完成增、删、查找等操作，但是后继及前趋节点需要遍历整个 HASH 表，需要时间$O(U)$。而一般的二叉树结构，可以在$O(logn)$的时间内完成各种操作。那么$O(loglog\ n)$是怎么来的呢？

对于一个大小为 u 的区间查询，这里我们用到了类似二叉树的结构，但是每一次分叉不是两个，而是$\sqrt{u}$。比如有u个节点的结构，分成$\sqrt{u}$组，每份大小为$\sqrt{u}$，对于这些每一个子结构继续递归地分组，直到最后只有一个数。我们用二进制来表示任意一个数x(e.g 55, 在u=256 时是00110111_2)。

我们定义$low(x) = lowoder\ half\ of\ bits\ (e.g.0111_2)$

$hight(x) = highorder\ half\ of\ bits\ (e.g.0011_2)$

所以，$low(x) = x\ mod\ \sqrt{u}$

$hight(x) = \lfloor x/\sqrt{u} \rfloor$

如果我们把分组考虑进去，那么也就是

$low(x)$ 组内序号

$hight(x)$ x 所在组的编号

因为在每一个子结构中都是可以随机访问(RAM)，知道了组内序号和组的编号，我们很容易得到查询的复杂度是：

$$T(u) = T\left(\sqrt{u}\right) + O(1)$$

$$T'(log\ u) = T'\left(log\sqrt{u}\right) + O(1)$$

$$T'(logu) = T'\left(\frac{1}{2}logu\right) + O(1)$$

所以 T(u)的复杂度是$O(loglog\ u)$

我们得到任意一个结构 S: 其中 summary[s]表示该节点信息，表明区间是否数。

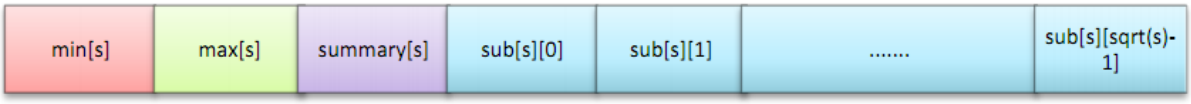


图 2.2　van Emde Boas Tree 节点结构

我们可以得到如下的 van Emde Boas tree 的结构，如下图所示

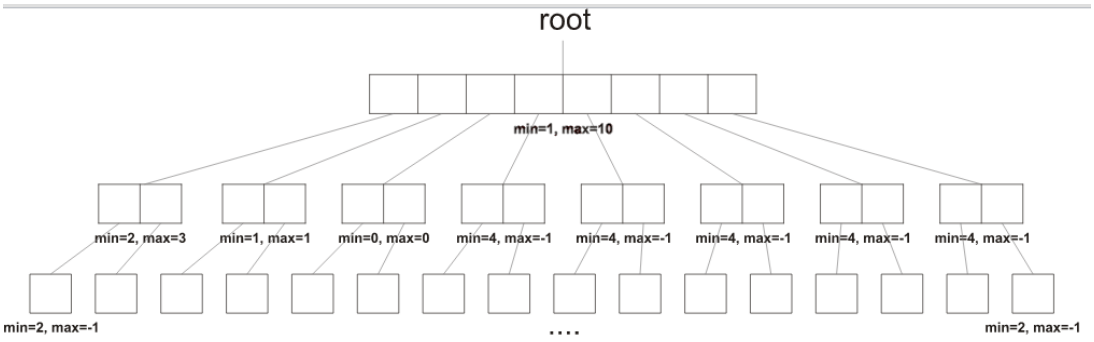


图 2.3 U = 16, 插入 1,2,3,5,8,10 之后的 Van Emde Boas tree

2.2.2 平面正交区域点定位问题

平面正交区域的点定位问题可以看作是一维的前趋搜索在几何意义上的扩展：给定平面上正交线段组成的O(n)个子区域，建立一个数据结构使得我们可以快速查询到给定点所在的子区域。通过垂直剖分，我们可以得到若干个互不相交的矩形区域。则问题可以转化为在平面上给定O(n)个不相交矩形的点定位问题。对于这个问题，de Berg, van Kreveld 和 Snoeyink 在之前的研究中提出了线性空间的$O((loglog\ U)^2)$的算法。

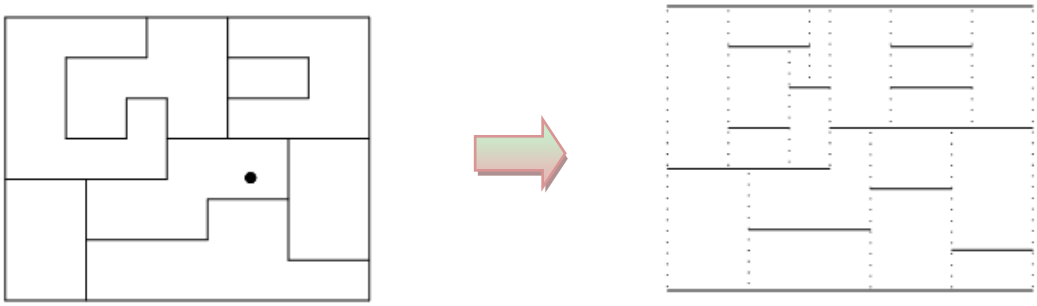


图 2.4　将区域竖直剖分成若干个互不相交的矩形

关于$O((\log\log U)^2)$的复杂度算法，其中一种方法就是用“两层”的数据结构，也就是说在 y 方向建立一个 Van Emde Boas tree，再在这棵树上的每一个节点上建立一个 x 方向的 Van Emde Boas tree。De Berg 等人用了这种思想来达到$O((\log\log U)^2)$的查询时间。

3. 算法描述

这里，我们的算法主要应用了 Van Emde Boas tree 将问题划分成$\sqrt{U}$的算法思想，使得最终的复杂度可以达到$O(loglog\ U)$。

首先，我们先把算法进行限定，需要在“离线”处理所有的线段更新，每一次的操作只是进行点定位的查询。我们可以将问题表示为：给定一个有 n 个不相交矩形的集合 S，S 中所有的定点都是整数，属于$[0, W) \times [0, H)$，初始时 W=H=U，所有矩形的边都是水平或者竖直的。我们的目的是建立一个数据结构可以查询到 S 中包含查询点 p 的矩形。

3.1 通过网格进行区域化简

我们运用递归来解决这个问题。第一个递归策略基于一种简单的想法：将区域等分成网格。特别的，我们用$\lceil W/K \rceil$行$\lceil H/L \rceil$列的网格来划分区域。第 i 列指的是$[iK, (i+1)K) \times [0, H)$，第 j 列表示$[0, W) \times [jL, (j+1)L)$，网格$(i, j)$表示$[iK, (i+1)K) \times [jL, (j+1)L)$。我们说一个矩形边 s 与网格相交当且仅当 s 的一个顶点在网格内，一个顶点在网格外。

数据结构表示为：

- 如果网格被某个矩形$s \in S$完全包含，T[i,j]=”covered by s”;
- 否则如果网格与 S 中某条竖直边相交，则 T[i,j]=”vertical”;

- 否则如果网格与 S 中某条水平边相交，则 T[i,j]=”horizontal”;
- 否则 T[i][j]= ”vertical”或者”horizontal”.

存储表格 T（如图），对于每一个网格点（i, j）

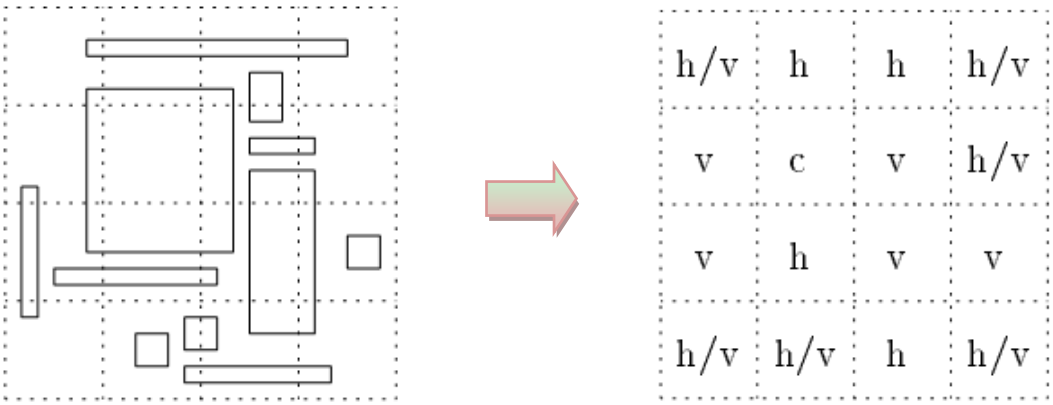


图 2.5　网格化后的存储表格结构

我们选择$K = \lceil W/\sqrt{n} \rceil, L = \lceil H/\sqrt{n} \rceil$，使得最后的网格数目为$O(n)$

查询算法：给定一个询问点$q(x, y)$

- $i = \lceil x/K \rceil, j = \lceil y/L \rceil$
- if T[i,j]=”covered by s”, then　return s
- If T[i,j]=”vertical” then　recursively query the data structure for column i;
- If T[i,j]=”horizontal” then　recursively query the data structure for row j;

算法的正确性很容易证明，比如说，如果 T[i,j]=”vertical”,那么没有水平边切过网格(i , j)，又因为所有的矩形是不相交的，所以包含 q 的矩形一定有一个定点在第 i 列。

算法的空间、时间复杂度可以如下计算

$$S(n, W, H) \le \sum_i S\left(m_i, \left\lceil \frac{W}{\sqrt{n}} \right\rceil, H\right) + \sum_j S\left(n_j, W, \left\lceil \frac{H}{\sqrt{n}} \right\rceil\right) + O(n)$$

$$Q(n, W, H) \le \max\left\{\max_i Q\left(m_i, \left\lceil \frac{W}{\sqrt{n}} \right\rceil, H\right), \max_j Q\left(n_j, W, \left\lceil \frac{H}{\sqrt{n}} \right\rceil\right)\right\} + O(1) \qquad (1)$$

3.2　通过哈希进行区域化简

我们考虑另一种递归策略。仅考虑 y 轴方向上的化简时，通过 van Emde Boas, 我们可以把 y 轴区间[0,H)分成大概$\sqrt{H}$行，每一行高度是$\sqrt{H}$。我们建立一个”top”数据结构来将非空行当做元素进行处理，一个”bottom”数据结构处理每一个非空行的元素。然而，这种方法并不能解决我们的问题。因为一个元素可能会出现在多行（也就是说，一个矩形的竖直边可能与多个网格相交）。我们的新想法是用一个新的”bottom”数据结构来处理这种情况，通过”collapsing”操作。

我们用$\lceil H/L \rceil$行的网格来表示一个 y 方向的划分，第 j 行表示区域$[0, W) \times [jL, (j+1)L)$

如下图，我们可以建立一个图的”top”,”bottom”结构：

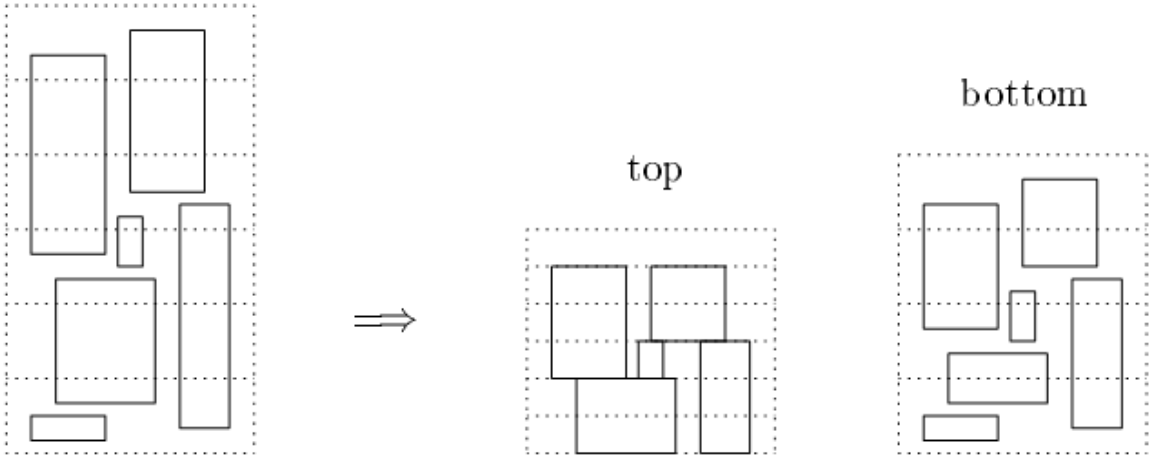


图 2.6　top 及 bottom 结构图

数据结构：

- 建立一个字典 D 存储所有包含 S 中定点的行的索引。将 D 排序，并且用序数函数$f(z)$表示元素$z \in D$的序数。
- “Round” 对于每一个 y 坐标先下去取整到它的行标号。对于每一个矩形$[x_1, x_2) \times [y_1, y_2)$表示为$[x_1, x_2) \times [\lfloor y_1/L \rfloor, \lfloor y_2/L \rfloor)$ ，递归建立 top 数据结构。
- “Collapse” 去掉所有不包含顶点的行。也就是说，将一个矩形$[x_1, x_2) \times [y_1, y_2)$映射到$[x_1, x_2) \times [f(\lfloor y_1/L \rfloor)]L + (y_1 \bmod L), [f(\lfloor y_2/L \rfloor)]L + (y_2 \bmod L)$。递归建立 bottom 数据结构。

注意到 top 数据结构的矩形范围的宽度是$[H/L]$，而 bottom 的矩形范围宽度最大是 nL，我们取$L = \left\lceil \sqrt{H/n} \right\rceil$，使得两种数据结构的宽度都是$\sqrt{nH} + n$。

查询算法：给定一个查询点$q(x, y)$

- If $\lfloor y/L \rfloor \notin D$ then recursively query the top data structure for the point $(x, \lfloor y/L \rfloor)$
- If $\lfloor y/L \rfloor \in D$ then recursively query the bottom data structure for the point $(x, [f(\lfloor y/L \rfloor)]L + (y \bmod L))$

正确性很容易证明。注意到每一次测试元素是否在 D 中只需要 O(1)的时间。

算法的时间空间复杂度可以表示为

$$S(n, W, H) \le 2S\left(n, W, \sqrt{nH} + n\right) + O(n)$$
$$Q(n, W, H) \le Q\left(n, W, \sqrt{nH} + n\right) + O(1) \quad (2)$$

在 x 方向上也同理，

$$S(n, W, H) \le 2S\left(n, \sqrt{nW} + n, H\right) + O(n)$$
$$Q(n, W, H) \le Q\left(n, \sqrt{nW} + n, H\right) + O(1) \quad (3)$$

3.3　两种递归方法结合

尽管两个递归策略分别来看都不够好，但是它们的结合却可以得到惊人的完美。第一种方法在 n 比较大的时候表现很好，而第二种方法在 n 比较小的时候有较大优势。因此，一个简单的结合就可以达$O(\log\log U)$的查询时间。

我们提出如下的算法

- If $n > \sqrt{W}, \sqrt{H}$ adopt the first strategy, using (1)
- If $n \le \sqrt{H}$ adopt the second strategy, using (2)
- If $n \le \sqrt{W}$ adopt the second strategy, using (3)

这样的设置平衡了$W/\sqrt{n}$和$\sqrt{nW}$, $H/\sqrt{n}$和$\sqrt{nH}$, 在这种情况下，我们得到时间、空间表达式

$$S(n,W,H) \le \sum_i S\left(m_i, O(W^{\frac{3}{4}}), H\right) + \sum_j S\left(n_j, W, O(H^{\frac{3}{4}})\right) + O(n)$$

$$Q(n,W,H) \le \max\left\{\max_i Q\left(m_i, (W^{\frac{3}{4}}), H\right), \max_j Q\left(n_j, W, (W^{\frac{3}{4}})\right)\right\} + O(1)$$

最后我们可以得到时间复杂度$Q(n,U,U) \le O(\log\log U)$，并且空间复杂度$S(n,U,U) \le O\left(4^{\log_{\frac{4}{3}}\log U} n\right) \le O(n\log^{10} U)$

算法流程图为

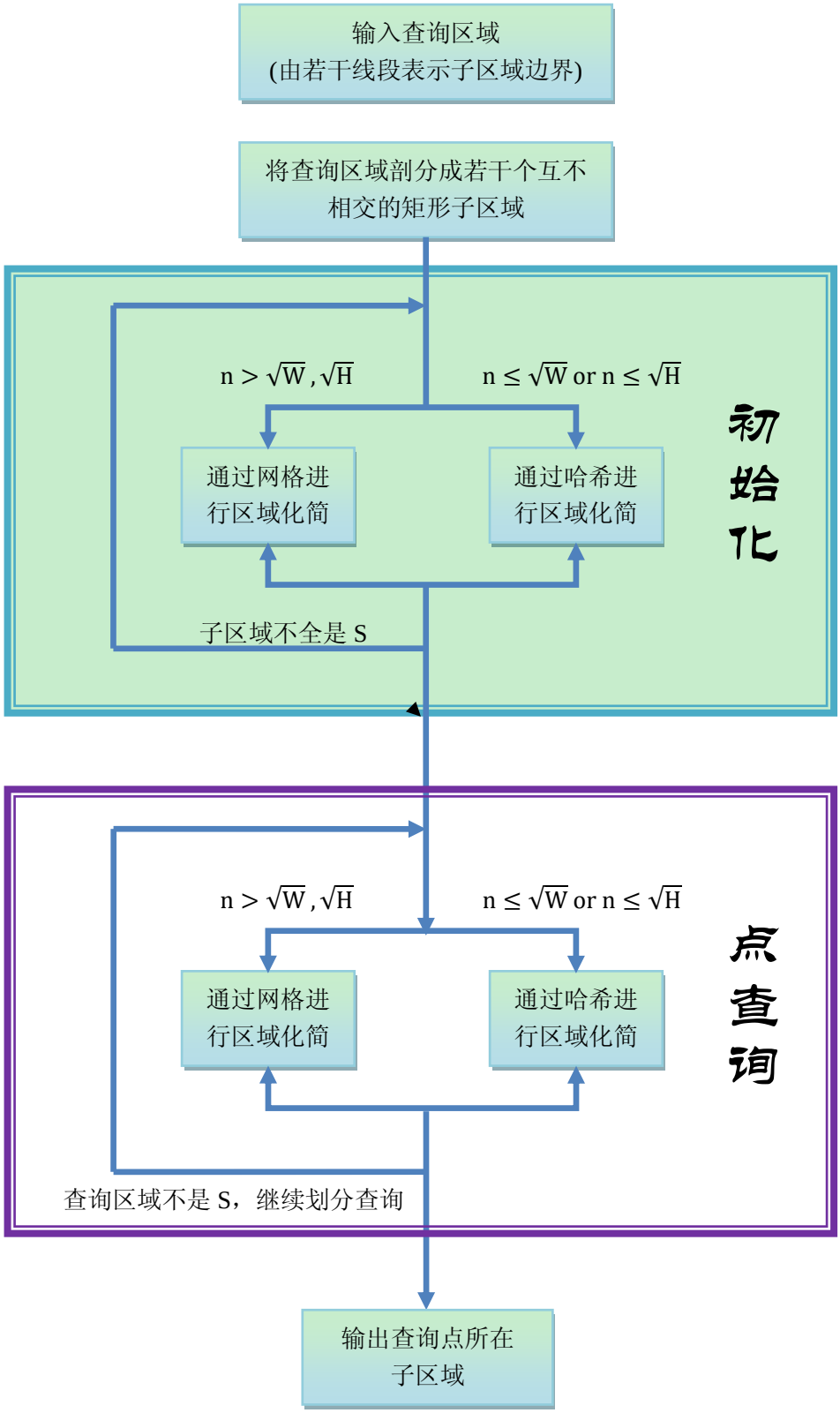
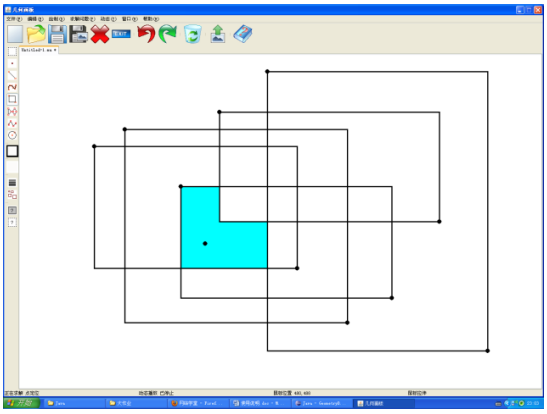


图 2.6　算法流程图

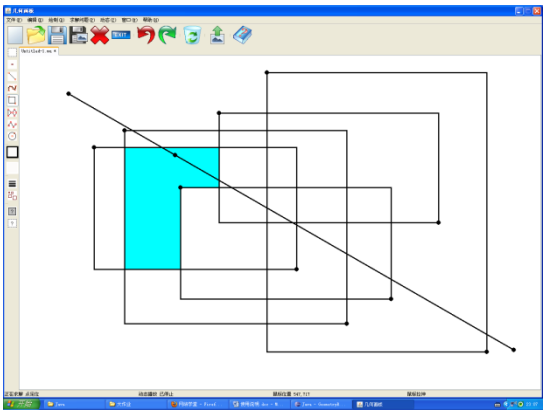
4. 实验结果

4.1 算法正确性

(a) 定点测试



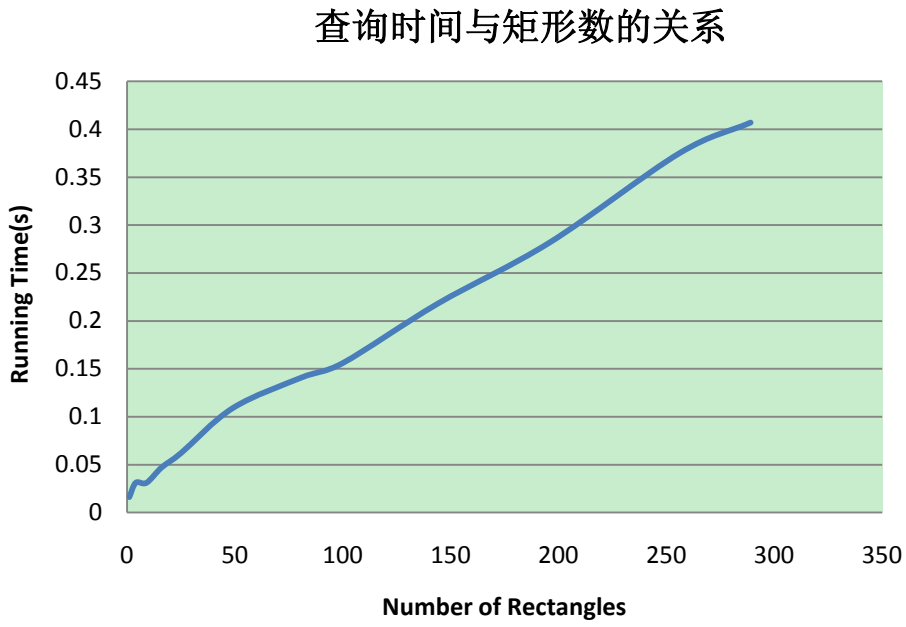
(b) 动点测试（点沿直线走）



4.2 算法效率

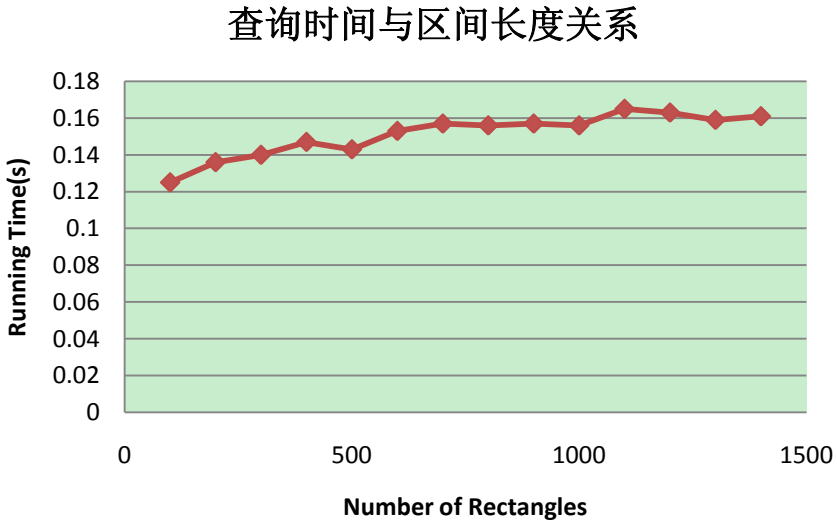
(I) 运行时间分析

算法运行时间与不相交矩形数目的关系



我们可以看到，尽管在计算复杂度时查询时间上没有 n 的因子，但是实际运行时间上和 n 是有关系的。分析其原因在于 O(log log U)是算法运行的上界，在 n 比较小的搜索树的高度比较低，因此运行时间短。但是从复杂度上来说算法还是 O(log log U)量级的。图中 U=500,查询点数为 10000 个。

算法运行时间与 U 的关系



我们可以看到，运行时间随着 U 的增长增加缓慢。符合 O(log log U)的查询时间的关系。图中取 n=100，查询点数为 10000 个。

(II) 算法对比

(a) 对比算法(Algorithm A)

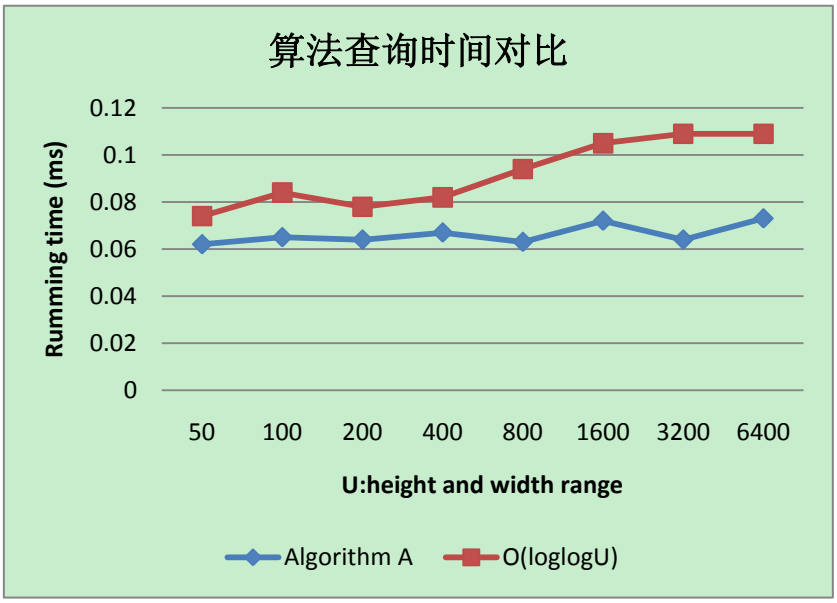
- 水平、竖直离散化，将原始图离散成$O(n^2)$个矩形
- 二分查找矩形所在区域

对比算法的单点查询时间O(logn)，预处理时间O(nlogn)，存储空间$O(n^2)$

(b) 随机数据生成程序

随机生成n 条长度取值为[1..U]的水平或者竖直线段。

取线段数 n=50 时，对比算法 A 和论文中O(loglog U)的算法



由上图我们可以看到，虽然在复杂度上O(loglogU)的更加优秀，但是在实际效果上它并不比简单的算法 A 表现好。算法 A 的复杂度是 O(log n)与 U 无关，所以不会随着 U 的增长而增长。而本文中的算法复杂度 O(log log U)，运行时间与 U 有关，但是不会随着 U 的增长而快速增长，这一点是和分析相符的。

5. 结论与展望

通过对比试验，我们会发现尽管该算法在理论上有很好的查询时间复杂度，但是在实践中该算法的表现还是不如简单的对比算法好的。分析其原因，我想主要是我们通常在计算时间复杂度的时候，往往会忽略掉常数因子。但是在这里，loglog U 和 log n 本身差别就小的情况下，常数因子对于时间的影响是非常大的。因此在很多时候，许多理论上面非常完美的算法往往在实际中表现不是很好。这些理论上好的算法只有在数据量非常大的时候才可以体现出来。因此，我们在实际中，应该根据数据规模选择合适的算法，而不是以为追求理论的完美。

从另一方面说，这个算法在理论上还有值得改进的地方，那就是它的空间复杂度。同样，在 1-d van Emde Boas tree 这种冗余也是存在的。将 van Emde Boas tree 转化为“x-fast”或者“y-fast”树，可以将复杂度由 O（U）将为 O(n)。用同样的思想，Chan 和 Patrascu 在论文[1]的介绍了 3 种降低复杂度的方法：random sampling, planar graph separators, 和基于 1-d exponential search tree 的方法。最终

可以将其空间复杂度降到线性，使得时间和空间上都达到理论上的最优解。

参考文献

[1] Timothy. M. Chan : Persistent Predencessor Search and Orthogonal Point Location on the Word RAM, SODA 2011
[2] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. Inform. Process Lett., 6:80-82, 1977.
[3] P. van Emde Boas, R.Kaas and E. Zijlstra. Design and implementation of an efficient priority queue. Mathematical Systems Theory, 10:99-127.1977
[4] 邓俊辉. 计算几何讲义