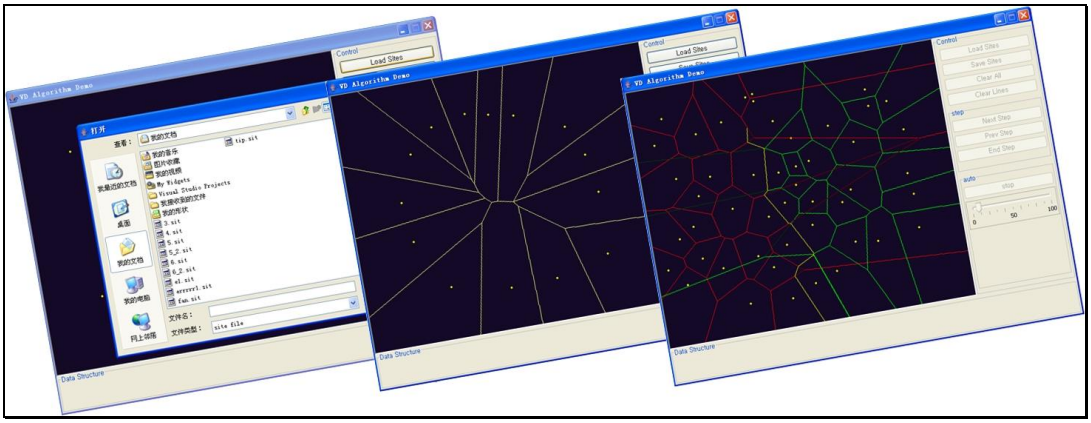
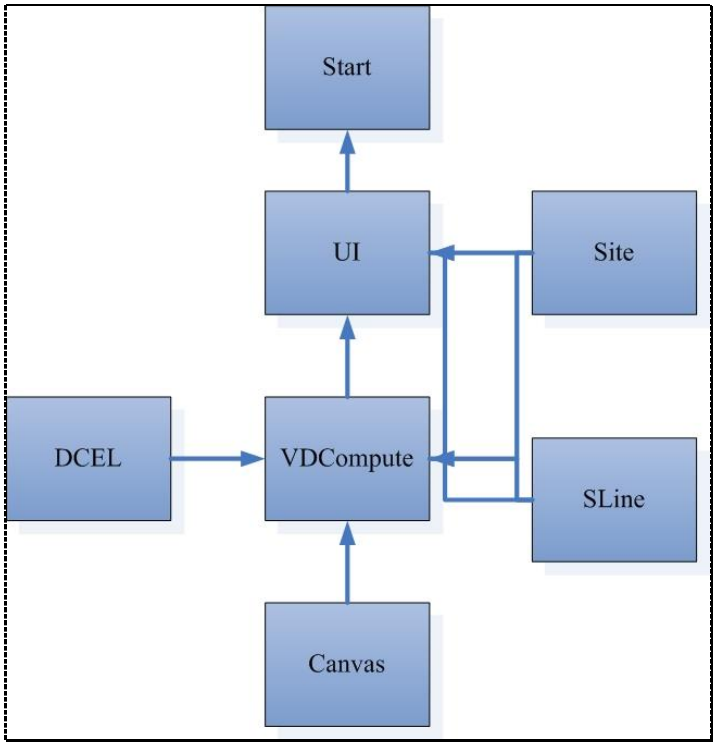
Voronoi Diagram 的分治算法演示报告

By 喻纯 & 张亮



【总体设计】



程序类关系图

我们的程序设计如图所示，解释如下：

类名	描述
Start	负责创建 UI
UI	控制核心，负责构建操作 Canvas 、VDCompute 以及界面相关的 button，panel 等控件
VDCompute	算法代码，负责计算 VD 分治算法，同时相应 UI 控制，实行界面交互
DCEL	Dcel 结构类，包括 face、edge、vertex 等描述及基本操作
SLine	Canvas 绘画的基本单元-线，在 canvas.updateUI()之前通过遍历 DCEL 以及 merge 过程临时变量结构获得
Site	Canvas 绘画的基本单元-点，用户输入
Canvas	绘图区，同时也为用户输入区

【算法说明】

首先，我们介绍一下我们的算法及其实现。

【算法流程】

VD 的分而治之算法采用典型的递归设计。对于一个 site 集 S，首先将其沿横轴按数量等分为左右两部分 S_L、S_R，然后对左右部分先后利用该算法求出它们的 VD 图，最后将它们通过 Merge 方法在线性时间内合并为一个大的 VD 图。递归代码如下：

DAC 算法核心代码

```
Private boolean dacVD(int i,int j)
{
    int diff = j-i;
    if(diff<2)
        return false;
    else if(diff<=3)
    {
        if(!trivialVD(i, j))
        {
            return false;
        }
    }
    int t = (i+j)/2;
    dacVD(i,t);
```

```
dacVD(t,j);
newMerge(i,t,t,j);
return true;
```

当 site 数量下于等于 3 并且大于 1 的时候，直接对当前图进行 **trivialVD**，它不是递归算法，直接生成当前 site 集的 VD 图。

该算法中最重要的就是 Merge 子过程，它将两个可以用垂直线隔开的 VD 图在线性时间内合并为一个 VD 图。算法伪代码如下：

Merge 算法伪代码

```
Merge（L1,L2,R1,R2）
//说明：L1,L2 表示左部分的 site 点下标范围；R1，R2，同理。
1：寻找左右两部分凸包的上公切线，用它和左右两部分的交点 f1tp,f2tp 表示。
        f1=S_L 中的最右点；
        f2=S_R 中的最左点；
        while(true)
        {
            f1up = nextCVFace(f1,counterwise);
            f2up = nextCVFace(f2,clockwise);
            if(!toleft(f1up.site,f2.site,f1.site))
            {
                f1 = f1up;
                continue;
            }
            if(toleft(f2up.site,f1.site,f2.site))
            {
                f2 = f2up;
                break;
            }
        }
2：从 f1tp 和 f2tp 形成的点对开始，顺序求得将左右两部分切割开的轮廓线 C 的每一个分段。
        f1 = f1tp;
        f2 = f2tp;
        lcp = 无穷远；
        Edge xe1 = lineXcell(f1,f2,f1,lcp);
        Edge xe2 = lineXcell(f1,f2,f2,lcp);
        if（xe1==null&&xe2==null）
            return;
        p1 = lineXEdge(f1,f2,xe1);
        p2 = lineXEdge(f1,f2,xe2);
        if(p1.y>p2.y)
        {
            C 和左部相交；
        }
```

```
            Else if(p1.y<p2.y)
            {
                C 和右部相交；
            }
            else
            {
                C 和左右部同时相交；
            }
```

3：依据轮廓线对左右两部分进行裁减，形成最终结果。

【数据结构】

DCEL 数据结构

```
class Face
{
    int id;
    Site site;
    Edge incEdge;
}
class Edge
{
    int id;
    Edge twinEdge;
    Vertex oriVertex;
    Face incFace;
    Edge preEdge;
    Edge nextEdge;
}
class Vertex
{
    double x;
    double y;
}
public class Site
{
    public double x;
    public double y;
}
```

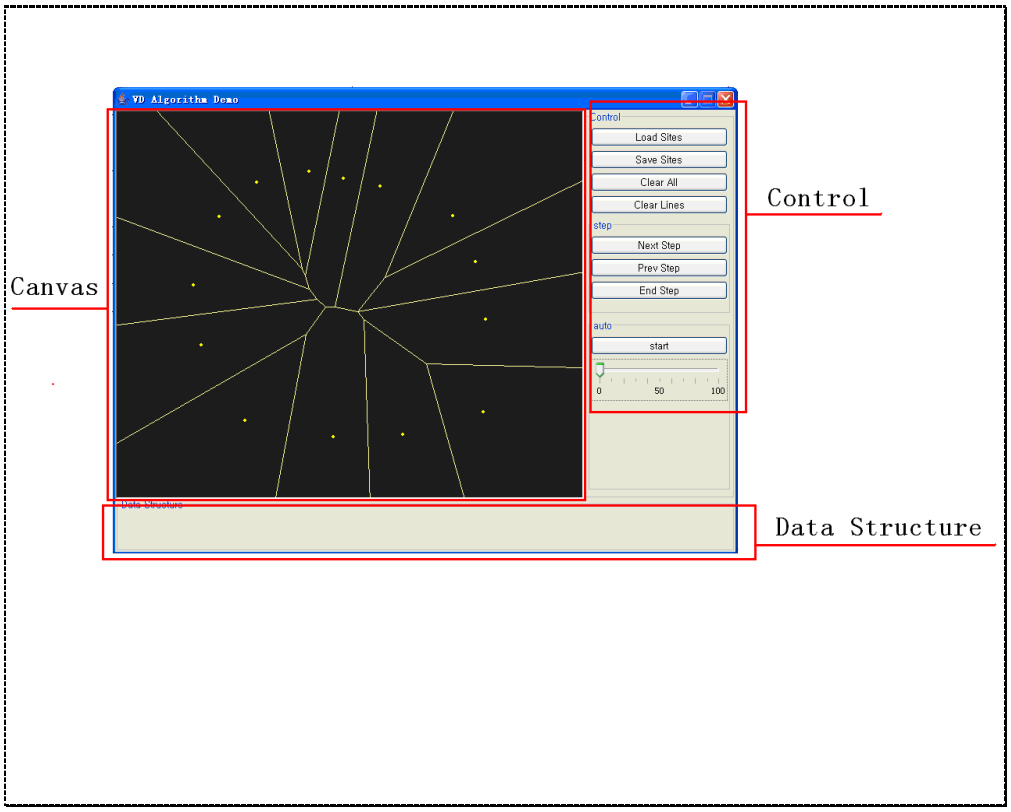
【代码实现】

代码较长，见源码。

【界面说明】

这一部分我们介绍一下我们演示界面设计及其效果。

【界面设计】



界面全貌

如图所示，界面空间分为三个区域：

1. Canvas 视图区，
2. Control 控制区，
3. Data Structure 数据描述区（暂不展现）

【功能介绍】

操作及描述

名称	描述
点输入	在 Canvas 区通过鼠标左键输入点

Load Sites	将保存过的点列信息导入
Save Sites	保存点列信息
Clear All	清空 Canvas 内的内容，包括线、点
Clear Lines	清空 Canvas 内的线
NextStep	单步执行时的下一步
PrevStep	单步执行时，Merge 时的上一步
EndStep	结束单步执行
Start	根据 speepslider 确定的时间间隔，连续演示算法过程

【代码实现】

VDCompute 是一个线程，与 UI 通过 notify() & wait()进行线程建通信，并实现“下一步”、“上一步”以及“连续”等控制

受控体 VD Compute 需要停留位置插入代码

```
if(ui.threadstate == 1)
{//auto连续执行时
    try
    {
        ui.lines = getLines();
        ui.canvas.updateUI();
        sleep(ui.speedSlider.getValue()*20);

    } catch (InterruptedException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
else if(ui.threadstate == 2)
{// 单步执行状体
    System.out.println("A3");
    ui.lines = getLines();
    ui.canvas.updateUI();
    synchronized (ui)
    {
        while(ui.isNext == false && ! ui.threadToStop)
        {// 等待nextStep button按下
            try
            {
```

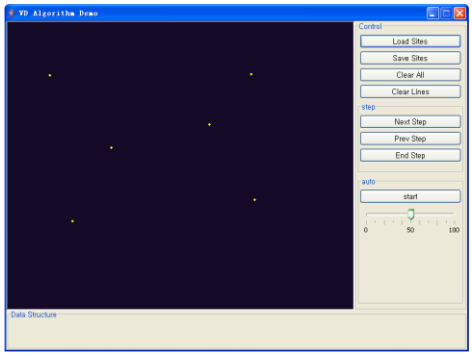
```
                        ui.wait();
                    }
                    catch(Exception e)
                    {}
                }
                ui.isNext = false;
            }
        }
```

UI nextstep 的 ActionListerner 代码

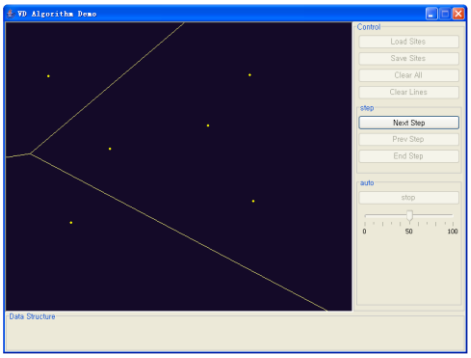
```
        canvas.updateUI();
        if(threadstate == 2)
        {
            synchronized (this)
            {
                isNext = true;
                notify();
            }
        }
        if(threadstate == 0)
        {
            Site[] tempS = new Site[sites.size()];
            for(int i = 0; i < tempS.length;i++)
            {
                tempS[i] = new Site(sites.get(i).x,sites.get(i).y);
            }
            vd = new VDCompute(this); //第一次按下创建VDCOmpute线程
            threadstate = 2;
            vd.start();
        }
```

【demo】

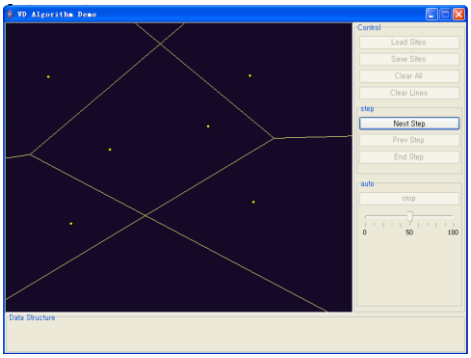
1. 在 Canvas 区域输入如图所示 6 个点



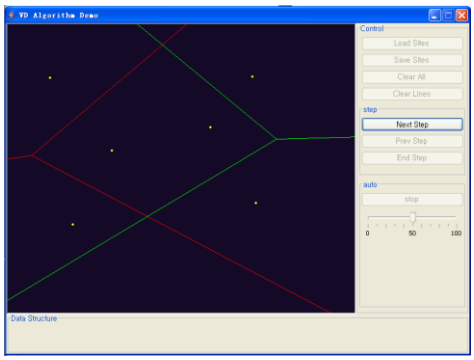
2. 按 NextStep 按钮后，分割左侧



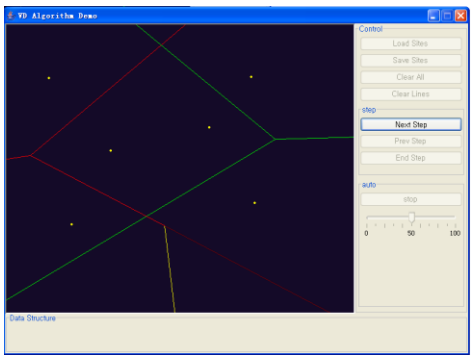
3. 按 NextStep 按钮后，分割右侧



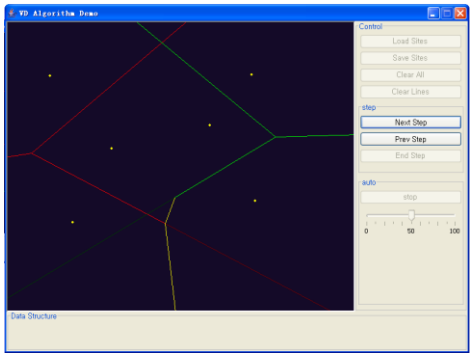
4. 按 NextStep 按钮后，将两侧颜色以不同颜色展现，表示 Merge 开始



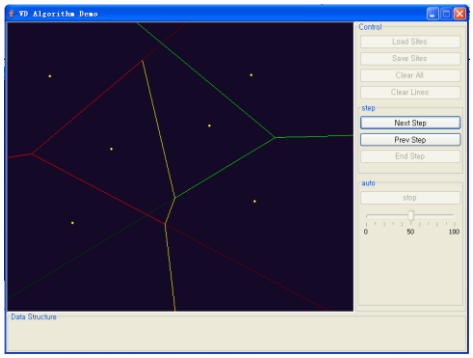
5. 按 NextStep 按钮后，开始绘制切线，以及相交线变化，被去掉线变淡



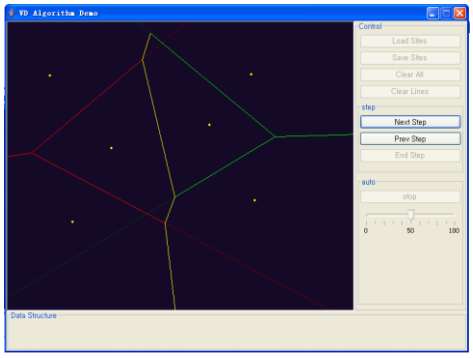
6. 按 NextStep 按钮后，绘制切线，以及相交线变化，被去掉线变淡



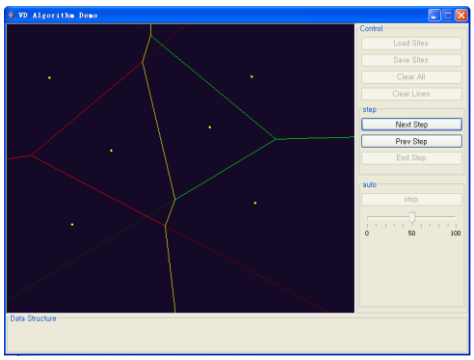
7. 按 NextStep 按钮后，绘制切线，以及相交线变化，被去掉线变淡



8. 按 NextStep 按钮后，绘制切线，以及相交线变化，被去掉线变淡



9. 按 NextStep 按钮后，绘制切线，以及相交线变化，被去掉线变淡



10. 按 NextStep 按钮后，完成 Merge 得到 VD 图。