Voronoi 图生成算法的实现、对比及演示实验报告

C 组
张哲 2012212882
唐磊 2012212861
陈帅 2012212906

1. 实验内容

对 Voronoi 图的生成算法进行实现、对比及演示。

2. 数据结构

采用 DCEL 结构（下面所列代码去除了函数、只留下成员变量，具体见源码中 basic_types.h 文件）：

```
class Site
{
public:
    Point p;
private:
    double x_, y_;  //coordinates of this site
    Face* incFace_; //the face that this site belongs to
};

class Vertex
{
public:
    Point p;
private:
    double x_, y_;      //coordinates of v
```

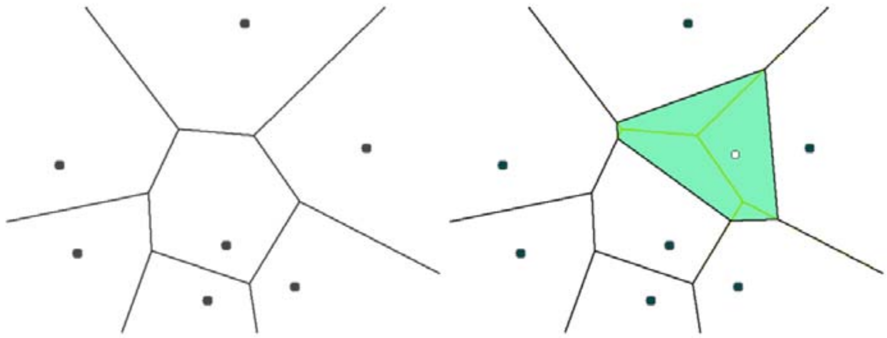
```
    Halfedge* incEdge_; //pionter to any outgoing incident halfedge
};


class Face
{
private:
    Site* site_;        //the site of this face
    Halfedge* incEdge_; //any incident halfedge
};

class Halfedge
{
public:
    bool hasDraw;

private:
    Halfedge* twinEdge_;  //pointer to twin halfedge
    Vertex*   oriVertex_; //pointer to origin vertex
    Face*     incFace_;   //pointer to left incident face
    Halfedge* prevEdge_;  //pointer to CCW previous halfedge
    Halfedge* nextEdge_;  //pointer to CCW next halfedge
    Point*    midPoint_;  //the midpoint of the two sites of this halfedge
    Vector*   direction_; //the direction of this halfedge
};
```

3. 算法描述

3.1 增量法

3.1.1 增量法概述

每次向 voronoi 图中增加一个点，寻找与这个点最近的 site，从这个 site 开始计算新加入的点所统治的区域的边界，删除旧的边并更新 DCEL 结构，直到所有点都加入到图中为止。



3.1.2 重点及难点

DCEL 结构的填充与更新：每插入一个新的点之后，对 DCEL 结构的更新都需要保证不出错。不能少更新或错更新，要保证 DCEL 结构的完整性及正确性。

多余边的删除与终止条件的判断：根据新加入的点位置的不同，对多余边的删除及终止条件的判断会有所不同，需要针对每种情况考虑清楚相应的对策。

边界跨越、其行进方向及终止条件的判断：同样的，由于新加入点的位置的不同（主要是在不在原来所有的点构成的凸包内），对边界的跨越、其行进方向及终止条件的判断都会有所不同。需要分别处理。

详细实现：

算法 1：递归分治

```
VoronoiDiagram* partition(vector<Point> &origin, int left, int right)
{
	if(right - left < 3)
	{
		VoronoiDiagram * result = smallVD(origin, left, right);//分别计算2或3个sites
		return result;
	}else
	{
		VoronoiDiagram * leftResult = partition(origin, left, (left+right)/2);
		VoronoiDiagram * rightResult = partition(origin, (left+right)/2 + 1, right);
		return mergeVD(leftResult, rightResult);
	}
}
```

算法 2：merge

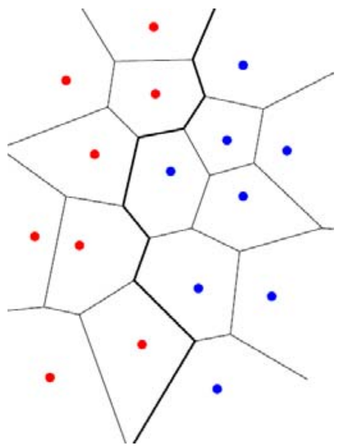
```
 VoronoiDiagram* mergeVD(VoronoiDiagram* left, VoronoiDiagram* right)
{
	VoronoiDiagram * result = new VoronoiDiagram();
	vector<DevideChain> devideChain;
	left->convex_hull = GeometryTool::getConvexHullUseGrahamScan(left->sites);
	right->convex_hull = GeometryTool::getConvexHullUseGrahamScan(right->sites);
	tangentLine(left->convex_hull,	right->convex_hull,	*leftMax,	*leftMin, *rightMax, *rightMin);//通过凸包分别得到 upper bound 和 lower bound，构造后面循环结束的条件.
	bool tobottom = false;
	do
	{
		tobottom = ((*leftMax).p == (*leftMin).p && (*rightMax).p == (*rightMin).p);
		chain = bisector(leftMax, rightMax);//得到 leftmax 和 rightMax 中垂线
		leftPoint = computeIntersectionPoint(leftMax, chain);//计算 chain 与左 site 对应 face 的交点
		rightPoint = computeIntersectionPoint(rightMax,chain);//计算 chain 与右 site 对应 face 交点
		if(leftPoint.y > rightPoint.y)//先与左边 site 相交
		{
			leftMax = IntersectionEdge.twin.face.site//更新左 site
			devideChain.push_back(IntesectionEdge,chain);
		}else if (rightPoint.y > leftPoint.y) //先与右边 site 相交
		{
			rightMax = IntersectionEdge.twin.face.site//更新右 site
			devideChain.push_back(IntesectionEdge,chain);
		}else
		{
			//循环结束
		}

	 }while(!tobottom);
	 connectWithChain(devideChain, left, right);//连接 chain
	 result = (*right) + (*left);
	 return result;
}
```

注意在连接的过程中删除边时的操作.若刚好被删除的边是标识某个 face 时，要更新此 face 的边.

3.2 分治法

3.2.1 分治法概述

- Let *P* be a set of *n* points in the plane.
- If the points are vertically partitioned into two subsets *R* and *B*.
    - consider the Voronoi diagram of the sets *R* and *B*.
    - then the Voronoi diagram of *P* substantially coincides with the Voronoi diagrams of *R* and *B*!
- In fact, there exists a monotone chain of edges of *V or*(*P*) such that *V or*(*P*) coincides with *V or*(*R*) to the left of the chain, and it coincides.



3.2.2 算法思路

1. Sort the points of *P* by abscissa (only once) and vertically partition *P* into two subsets *R* and *B*, of approximately the same size.
2. Recursively compute *V or*(*R*) and *V or*(*B*).
3. Compute the separating chain.
4. Prune the portion of V or(R) lying to the right of the chain and the portion of V or(B) lying to its left.

如何计算中间分隔链：

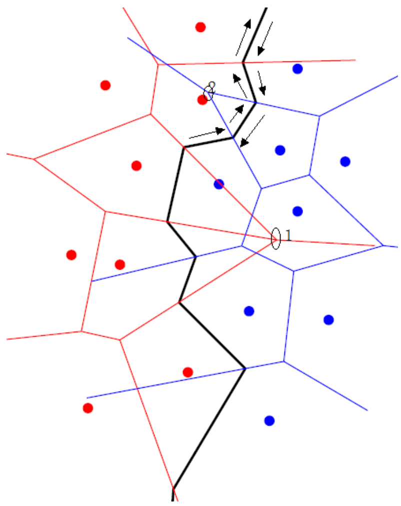
- Initialization-Find the two halflines(依据凸包计算).
- Starting with one of the halflines, and until getting to the other one, do:
- Each time an edge e ∈ b(R,B) begins, such that e ⊂ bij, pi ∈ R and pj ∈ B, do:
    - Detect its intersection with V orR(pi).

    - Detect its intersection with V orB(pj).
    - Choose the first of the two intersection points.
    - Detect the site pk corresponding to the new starting region

Replace pi or pj (as required) by pk.

如何进行 merge：

- 得到 chain 后，先连接 chain，分两个方向
- 加入到 left 和 right 的边中.
- chain 中 segment 是跟 left 相交还是 right 相交：
    - Left:相交的边 next 在 segment 的右，递归删
    - Right:相交的边 prev 在 segment 的左，递归删
    - 注意：被删除的某边可能属于某个 face 指向的边



3.2.3 详细实现：

算法 1：递归分治

```
VoronoiDiagram* partition(vector<Point> &origin, int left, int right)
{
	if(right - left < 3)
	{
		VoronoiDiagram * result = smallVD(origin, left, right);//分别计算 2 或 3 个 sites
		return result;
	}else
	{
		VoronoiDiagram * leftResult = partition(origin, left, (left+right)/2);
		VoronoiDiagram * rightResult = partition(origin, (left+right)/2 + 1, right);
		return mergeVD(leftResult, rightResult);
	}
}
```

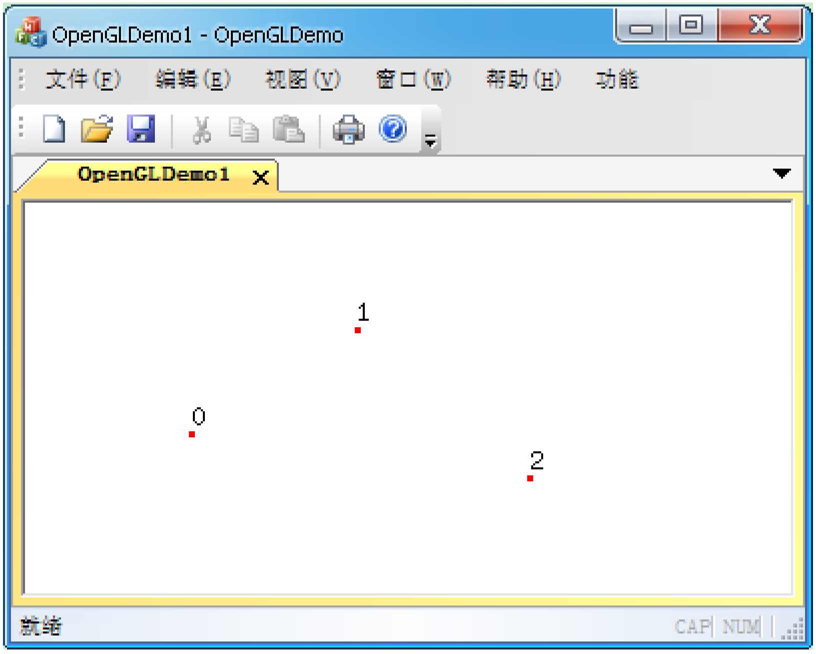
算法 2：merge

```
 VoronoiDiagram* mergeVD(VoronoiDiagram* left, VoronoiDiagram* right)
{
    VoronoiDiagram * result = new VoronoiDiagram();
    vector<DevideChain> devideChain;
    left->convex_hull = GeometryTool::getConvexHullUseGrahamScan(left->sites);
    right->convex_hull = GeometryTool::getConvexHullUseGrahamScan(right->sites);
    tangentLine(left->convex_hull, right->convex_hull, *leftMax, *leftMin,
*rightMax, *rightMin);//通过凸包分别得到 upper bound 和 lower bound，构造后面
循环结束的条件.
    bool tobottom = false;
    do
    {
        tobottom = ((*leftMax).p == (*leftMin).p && (*rightMax).p == (*rightMin).p);
        chain = bisector(leftMax, rightMax);//得到 leftmax 和 rightMax 中垂线
        leftPoint = computeIntersectionPoint(leftMax, chain);//计算 chain 与左 site 对
    应 face 的交点
        rightPoint = computeIntersectionPoint(rightMax,chain);//计算 chain 与右 site
    对应 face 交点
        if(leftPoint.y > rightPoint.y)//先与左边 site 相交
        {
            leftMax = IntersectionEdge.twin.face.site//更新左 site
            devideChain.push_back(IntesectionEdge,chain);
        }else if (rightPoint.y > leftPoint.y) //先与右边 site 相交
        {
            rightMax = IntersectionEdge.twin.face.site//更新右 site
            devideChain.push_back(IntesectionEdge,chain);
        }else
        {
            //循环结束
        }

     }while(!tobottom);
     connectWithChain(devideChain, left, right);//连接 chain
     result = (*right) + (*left);
     return result;
}
```

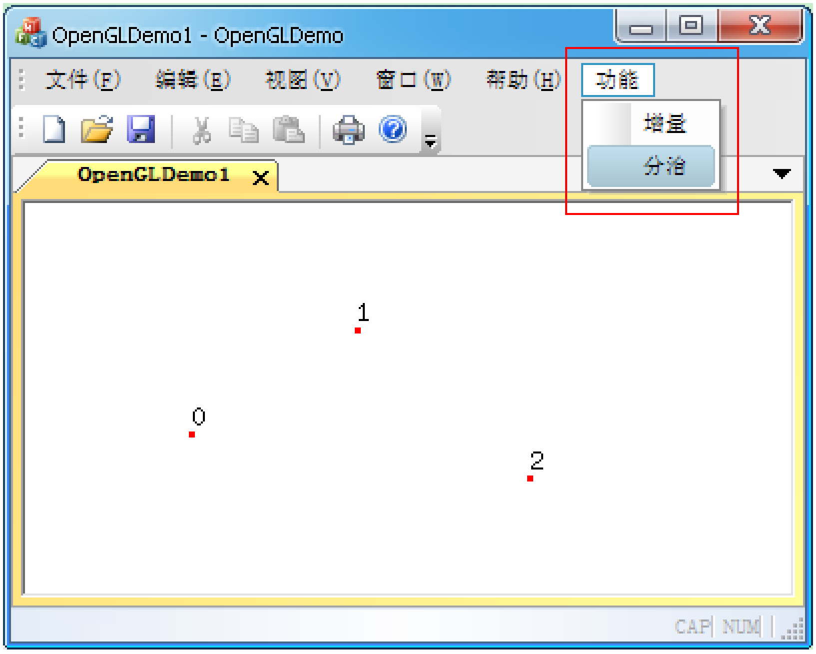
注意在连接的过程中删除边时的操作.若刚好被删除的边是标识某个 face 时，要更新此 face 的边.

4. 程序使用方法

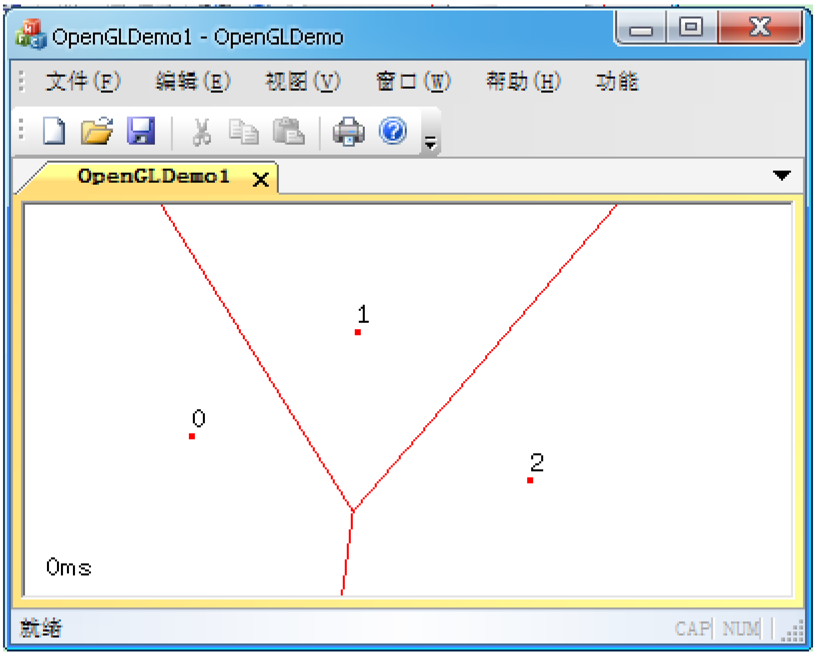
直接用鼠标左键在空白处单击输入点：



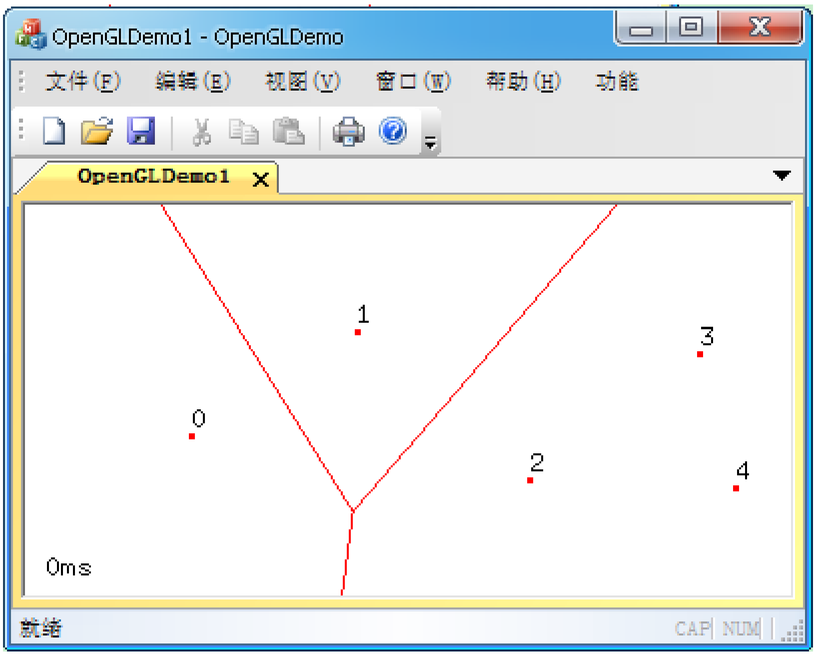
选择菜单“功能”下的“增量”或“分治”选项来生成 voronoi 图：



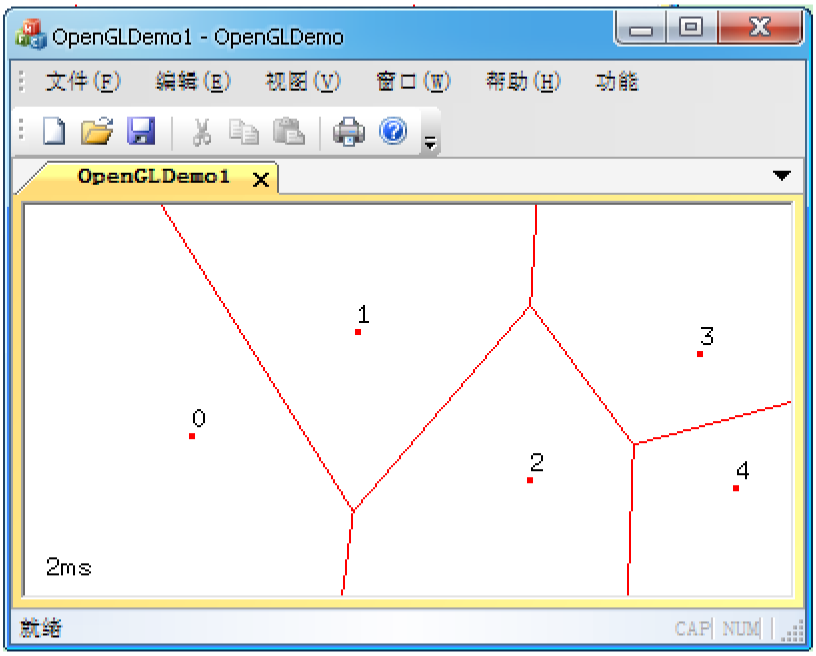
生成的 voronoi 图如下：



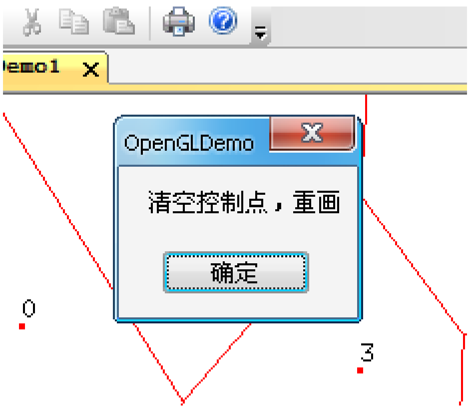
界面左下角显示了算法生成 voronoi 图所用的时间，以毫秒为单位。之后可以继续在界面上添加新的点：



并继续使用菜单上的功能生成 voronoi 图：



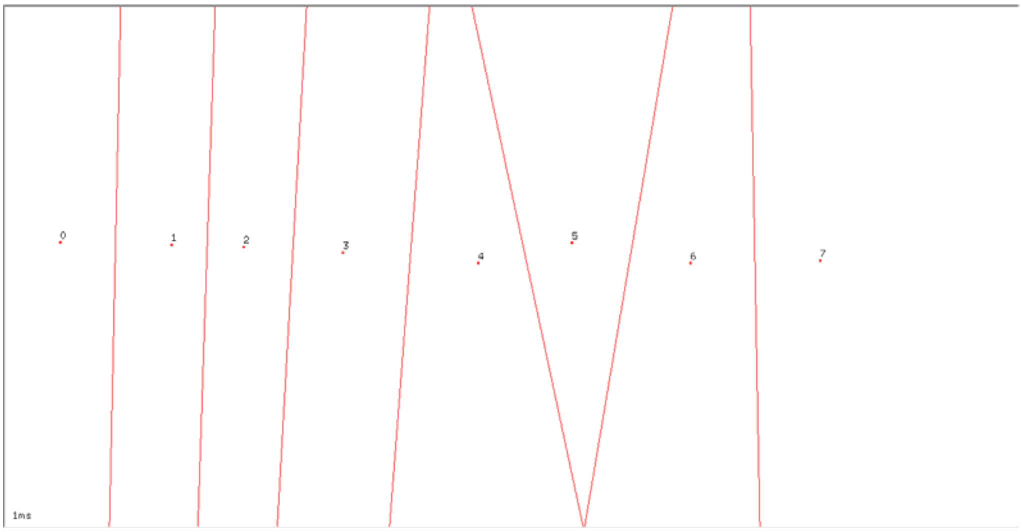
同时还可以双击界面空白处清空当前所有点，然后重新输入点：



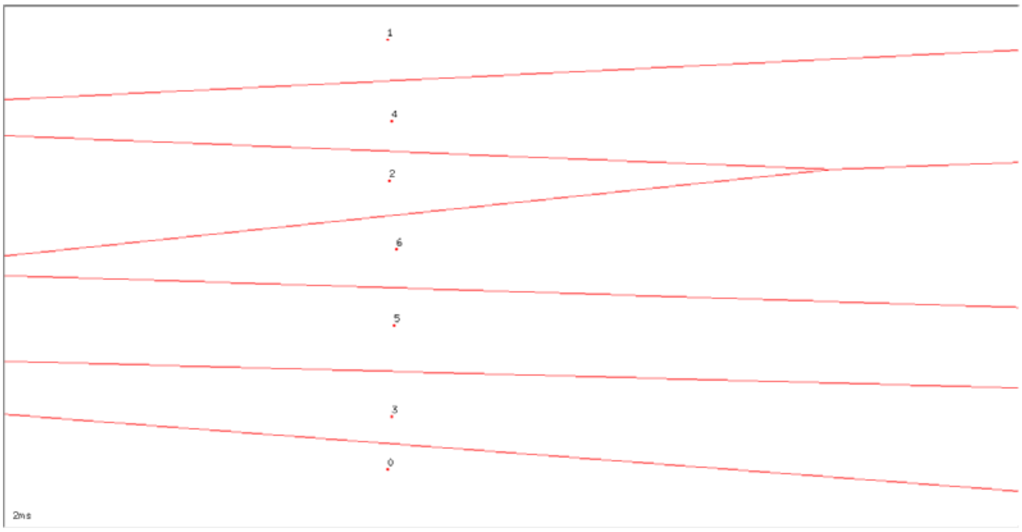
另外，还可以使用菜单中的保存与打开功能将当前的点集保存成文本文件，以供下次打开使用。

5. 例子演示

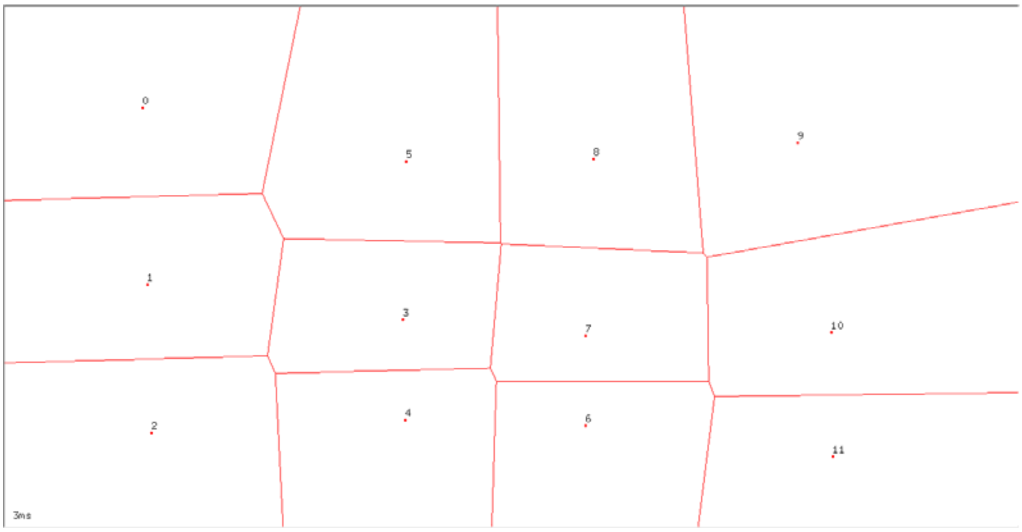
下面所示的几个例子均可读取附带的 example0x.txt 文件重现。



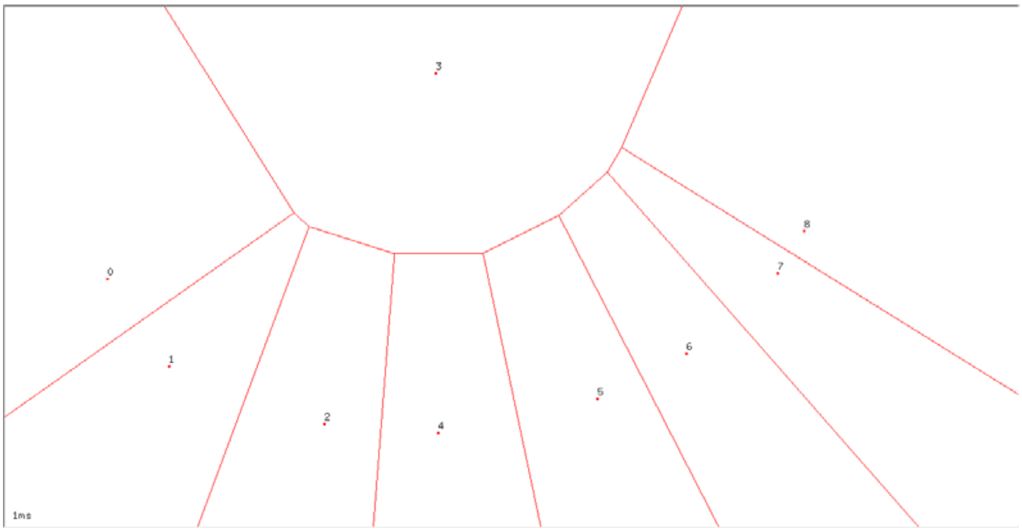
example01.txt



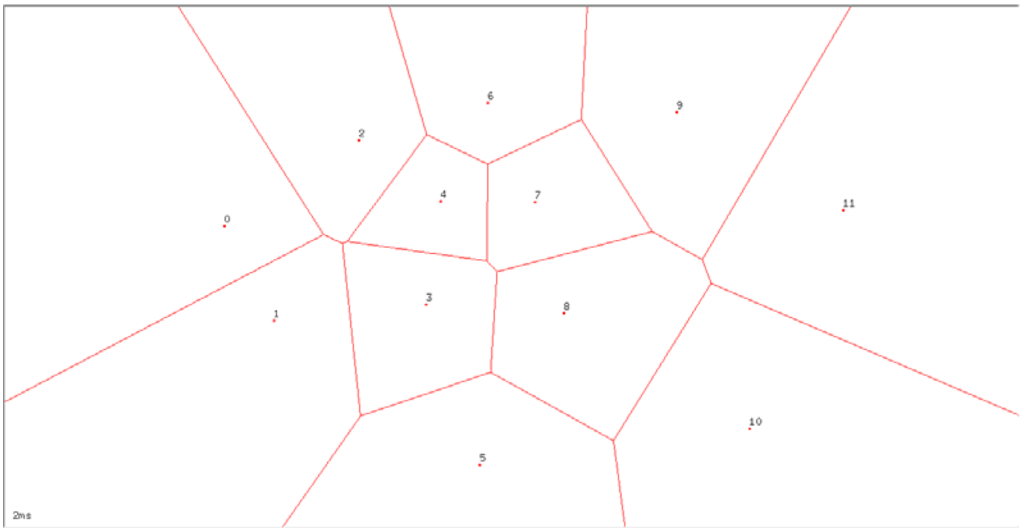
example02.txt



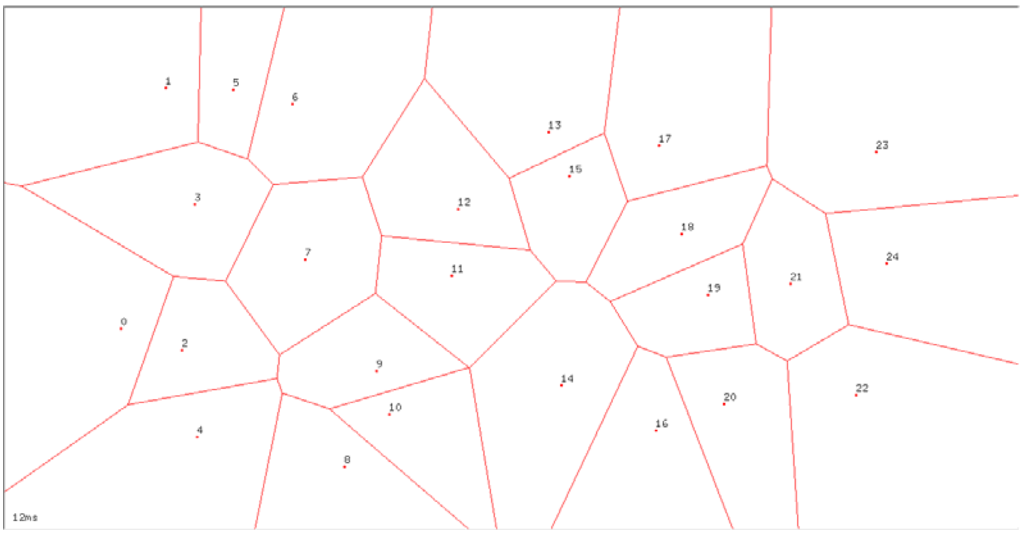
example03.txt



example04.txt



example05.txt



example06.txt

6. 性能与鲁棒性分析

实验平台：

操作系统：Windows XP SP3 32-bit；

处理器：Intel Core 2 Duo T6570, 2100 MHz；

内存：2GB DDR2-800　x2；

点数与算法运行时间如下：

点数	2	4	8	16	32	64	128	256
增量	0 ms	0 ms	1 ms	3 ms	10 ms	27 ms	80 ms	277 ms
分治	0 ms	1 ms	2 ms	8 ms	22 ms	65 ms	*	*

*分治法由于还存在一些 bug，点数过多时无法完成计算。

鲁棒性上，由于包括求交在内的全部代码均为我们手写，没有调用任何库，故对一些平行的情况可能处理的不够好，在随机生成点的情况下，增量法未观测到任何问题，分治法在点数较多的情况下会出现错误。

7. 实验总结

本次实验完成的并不是很成功，经过我们的反思，认为主要问题在于前期对选题的把握不够好。三人分别单独完成三种不同的算法任务量略大，并且初期互相交流不够充分，导致进度缓慢。同时，调试上的困难也致使我们后期花费了大量时间在 debug 上，对程序性能和鲁棒性的测试不够充分。

然而，本次实验也带给了我们不少的收获，例如对 voronoi 图生成算法的实现细节的深入理解、程序调试能力的锻炼与提升等等。

8. 参考文献

1. 计算几何课程讲义，邓俊辉

2. Aurenhammer, Franz. "Voronoi diagrams—a survey of a fundamental geometric data structure." ACM Computing Surveys (CSUR) 23.3 (1991): 345-405.

3. Klein, Rolf. "Abstract Voronoi diagrams and their applications." Computational Geometry and its Applications. Springer Berlin Heidelberg, 1988. 148-157.

4. Fortune, Steven. "A sweepline algorithm for Voronoi diagrams." Algorithmica 2.1-4 (1987): 153-174.