A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams

Welson Jacometti

Relatório Técnico DCC–04/92

Junho de 1992

A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams

Welson Jacometti*

Keywords: Quad-Edge data structure, Voronoi Diagram, Delaunay Triangulation, Farthest-Neighbor Voronoi Diagram, Farthest-Neighbor Triangulation

In [GS85], Guibas and Stolfi introduced the Quad-Edge data structure and applied it to a divide-and-conquer algorithm which computes the Delaunay Triangulation and the Nearest-Neighbor Voronoi Diagram of a set of n points in the plane [PS85]. We note that, in fact, few modifications on their original algorithm can produce one for the computation of the Farthest-Neighbor Voronoi Diagram from its dual (a triangulation of the convex hull of the set of points) that runs in optimal $\Theta(n \log n)$ time and linear space in the worst case, likewise the original.

The Modified Algorithm

The InCircle Test

The first modification required is to replace each of the three ocurrences of the **InCircle** test by an **OutCircle** test; the **InCircle** test determines

*Departamento de Ciência da Computação, Universidade Estadual de Campinas, Campinas, SP, Brazil

whether a given point is *interior* to the circle spanned by a certain triplet of points, while the **OutCircle** tests whether a given point is *exterior* to that circle[1] (see [GS85] for more details about the **InCircle** test). In other words, let A, B, C be three points in the plane. Given a fourth point D, **OutCircle** returns *true* if and only if

$$\mathcal{D}(A,B,C,D) = \begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{vmatrix} < 0.$$

Merging Two Convex Structures

Another necessary modification refers to the merging of two disjoint Farthest-Neighbor Triangulations. As only the points of the convex hull participate in these triangulations, merging two of them is equivalent to merging two convex polygons and hence the points not in the resulting hull must be discarded. Note that since in the original algorithm the inferior supporting line is already being calculated, all that has to be added to accomplish this task in linear time is to calculate the superior supporting line of the triangulations, and then discard the points that lie between these lines.

These modifications arise from observing the property that every triangle in the Farthest-Neighbor Triangulation has a circle through its vertices that encloses the entire set, which is the dual of the property in which the original algorithm is based (these observations first surfaced while doing experimentations within GeoLab – a programming environment for geometric algorithms currently under development [Jac92]).

In the algorithm below, the modifications to the original are printed in italic font.

[1] Note that points on the boundary of the circle are considered *exterior* by the **InCircle** test and *interior* by the **OutCircle**.

```
1 PROCEDURE FVorTriangulation [S] RETURNS [le, re]:
2   IF |S| = 2 THEN
3     a <- MakeEdge[]; a.Org <- s1; a.Dest <- s2;
                                  RETURN [a, a.Sym]
4   ELSEIF |S| = 3 THEN
5     IF COLINEAR[s1, s2, s3] THEN
6       a <- MakeEdge[]; a.Org <- s1; a.Dest <- s3;
                                    RETURN[a, a.Sym]
7     ELSE
8       a <- MakeEdge[]; b <- MakeEdge[]; Splice[a.Sym, b];
9       a.Org <- s1; a.Dest <- b.Org <- s2; b.Dest <- s3;
10      IF CCW[s1, s2, s3] THEN c <- Connect[b, a];
                                   RETURN [a, b.Sym]
11      ELSEIF CCW[s1, s3, s2] THEN c <- Connect[b, a];
                                      RETURN [c.Sym, c]
12      FI
13    FI
14  ELSE
15    [ldo, ldi] <- FVorTriangulation[L];
      [rdi, rdo] <- FVorTriangulation[R];
16    {set uldi and urdi that are used to calculate the
       superior supporting line}
17    uldi <- ldi.Onext; urdi <- rdi.Oprev;
18    guard.ldi <- nil; guard.rdi <- nil;
19    DO
20      IF NOT RightOf[rdi.Org, ldi] AND
            guard.ldi <> ldi.Lnext THEN
21        IF guard.ldi = nil THEN guard.ldi = ldi FI;
22        ldi <- ldi.Lnext
23      ELSEIF NOT LeftOf[ldi.Org, rdi] AND
                guard.rdi <> rdi.Rprev THEN
24        IF guard.rdi = nil THEN guard.rdi = rdi FI;
25        rdi <- rdi.Rprev
26      ELSE EXIT FI
```

```
27      OD;
28      {calculate the superior supporting line}
29      guard.uldi <- nil; guard.urdi <- nil;
30      DO
31        IF NOT LeftOf[urdi.Org,uldi] AND
              guard.uldi <> uldi.Rprev THEN
32          IF guard.uldi = nil THEN guard.uldi = uldi FI;
33          uldi <- uldi.Rprev
34        ELSEIF NOT RightOf[uldi.Org,urdi] AND
                  guard.urdi <> urdi.Lnext THEN
35          IF guard.urdi = nil THEN guard.urdi = urdi FI;
36          urdi <- urdi.Lnext
37        ELSE EXIT FI
38      OD;
39      basel <- Connect[rdi.Sym, ldi];
40      {remove points that do not lie in
         the final convex hull}
41      ubaselOrg <- urdi.Sym.Dest; ubaselDest <- uldi.Org;
42      lcand = basel.Rprev.Sym.Onext;
        rcand = basel.Oprev.Sym.Oprev;
43      WHILE NOT lcand.Org = ubaselDest DO
44        t <- lcand.Rprev;
45        remove all edges out of lcand
46        lcand <- t;
47      OD;
48      WHILE NOT rcand.Org = ubaselOrg DO
49        t <- rcand.Lnext;
50        remove all edges out of rcand
51        rcand <- t;
52      OD;
53      IF ldi.Org = ldo.Org THEN ldo <- basel.Sym FI;
54      IF rdi.Org = rdo.Org THEN rdo <- basel FI;
55      DO
56        lcand <- basel.Sym.Onext;
57        IF Valid[lcand] THEN
```

```
58          WHILE OutCircle[basel.Dest, basel.Org,
                    lcand.Dest, lcand.Onext.Dest] DO
59            t <- lcand.Onext; DeleteEdge[lcand];
              lcand <- t;
60          OD
61        FI;
62        rcand <- basel.Oprev;
63        IF Valid[rcand] THEN
64          WHILE OutCircle[basel.Dest, basel.Org,
                     rcand.Dest, rcand.Oprev.Dest] DO
65            t <- rcand.Oprev; DeleteEdge[rcand];
              rcand <- t;
66          OD
67        FI;
68        IF NOT Valid[lcand] AND NOT Valid[rcand] THEN
            EXIT
          FI;
69        IF NOT Valid[lcand] OR
70          (Valid[rcand] AND
71            OutCircle[lcand.Dest, lcand.Org,
                rcand.Org, rcand.Dest])
72          THEN
73            basel <- Connect[rcand, basel.Sym]
74          ELSE
75            basel <- Connect[basel.Sym, lcand.Sym]
76          FI
77        OD;
78        RETURN [ldo, rdo]
79      FI
80    END FVorTriangulation.
```

Figure 1: The modified algorithm.

Intuitively, in the merge phase of our modified algorithm, we can think of the "rising bubble" [GS85] as running on the outside of the set, ensuring that to each triangle created the above property holds. Figure 2 illustrates this behavior.

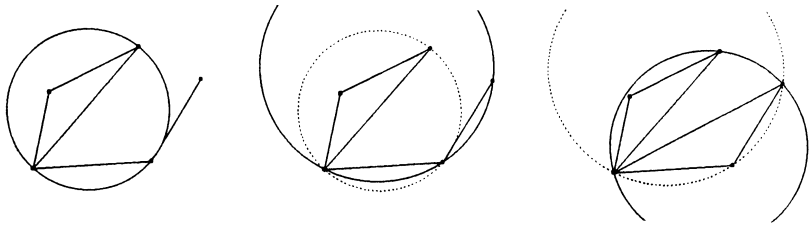

Figure 2: Three steps of the merging phase.

Lines 16–17, 28–38, 28–38 achieve a clean-up of the quad-edge data structure by removing points not in the hull of the original set. Therefore, these lines could be eliminated provided that the input points form a convex polygon (or, equivalently, one might preprocess the input set via any optimal convex hull construction, e.g., Kirkpatrick and Seidel's algorithm [KS86]). Lines 5–13, 18–27 were modified from the original algorithm to correctly handle colinear points (by skipping then).

As for the correctness of the above algorithm, the reader will have no difficulties in verifying that the proof presented in [GS85] holds, provided that **InCircle** is replaced by **OutCircle**. It is also easy to see that all changes introduced take linear time (within loops of $\log n$ depth). Hence, we have obtained an optimal $\Theta(n \log n)$ time (and linear space) algorithm for the construction of the Farthest-Neighbor Voronoi Diagram of a set of n given points in the plane.

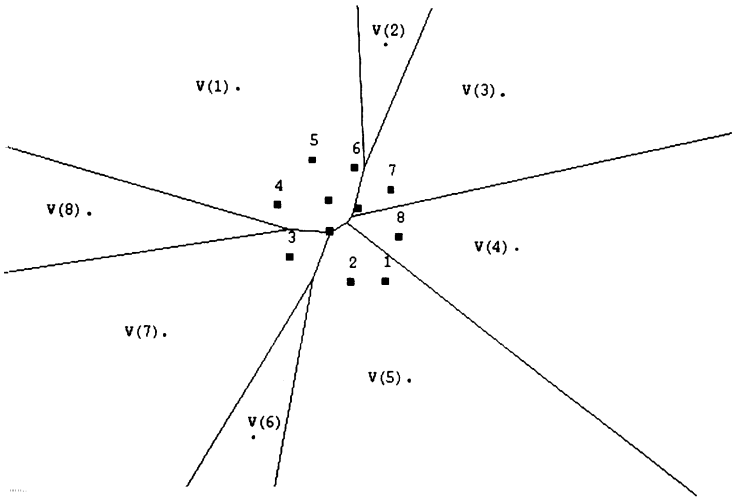

Figure 3: Farthest-Neighbor Voronoi diagram for a set of eight points (internal points discarded).

References

[GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, April 1985.

[Jac92] W. R. Jacometti. GeoLab – Um Ambiente para Desenvolvimento de Algoritmos em Geometria Computacional. Master's thesis, DCC - IMECC - UNICAMP, 1992. to appear.

[KS86] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(2):287–299, 1986.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer Verlag, New York, NY, 1985.

Relatórios Técnicos

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`