Applied and
Computational
Mathematics
Division

NISTIR 5596

Computing and Applied Mathematics Laboratory

Inserting Line Segments into Triangulations and Tetrahedralizations

J. Bernal

March 1995

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Gaithersburg, MD 20899

NISTIR 5596

Inserting Line Segments into Triangulations and Tetrahedralizations

J. Bernal

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Applied and Computational Mathematics Division
Computing and Applied Mathematics Laboratory
Gaithersburg, MD 20899

March 1995

U.S. DEPARTMENT OF COMMERCE
Ronald H. Brown, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arati Prabhakar, Director

Inserting Line Segments into Triangulations and Tetrahedralizations

Javier Bernal

National Institue of Standards and Technology, Gaithersburg, MD 20899, U. S. A.

Abstract. In this paper, we further develop an algorithm by Bernal, De Floriani, and Puppo, for inserting a line segment into a Constrained Delaunay triangulation. The new version of the algorithm inserts the line segment in exactly the same manner in which the old version does but has the additional capability that it does not delete the triangles intersected by the line segment but transforms them through edge-swapping. Since the concept of edge-swapping generalizes to 3−dimensional space, a 3−dimensional version of the algorithm without the optimization step for the Delaunay property is also presented for attempting to insert a line segment into a tetrahedralization. It is shown that for certain cases the failure of the 3−dimensional algorithm to insert a line segment is an indication that it can not be done. Finally, 3−dimensional problems that can be approached algorithmically as 2−dimensional problems are identified.

Key words. computational geometry, constrained Delaunay triangulation, edge-swapping, empty circle criterion, locally equiangular, segment insertion, Voronoi diagram

AMS(MOS) subject classifications. 68U05

1. Introduction

A *triangulation for* a finite set of points S in the plane is a finite collection of triangles in the plane having pair-wise disjoint interiors, each of which intersects S exactly at its vertices, and the union of which is the convex hull of S. Given a triangulation T for S, we say that T is a *Delaunay triangulation for* S if for each triangle in T there does not exist a point of S inside the circumcircle of the triangle [10]. A *(Delaunay) tetrahedralization for* a finite set of points in 3−dimensional space is similarly defined with tetrahedra and spheres taking the place of triangles and circles.

A triangulation more general than the Delaunay triangulation can be defined. Let S be a finite set of points in the plane, and let E be a finite collection, possibly empty, of line segments that intersect only at points in S and whose endpoints belong to S. We say that a triangulation T for S is a *triangulation for S constrained by E* if for each e in E and each t in T, e does not intersect the interior of t. Given T, a triangulation for S constrained by E, we say that T is a *Delaunay triangulation for S constrained by E* if for each t in T there does not exist a point P of S inside the circumcircle of t such that no e in E intersects the interior of the convex hull of $t \cup \{P\}$.

Let S and E be as above. Given T, a triangulation for S constrained by E, we say that T is *locally equiangular constrained by E* if given any two triangles in the triangulation that share a common edge not contained in any edge belonging to E and whose union is a strictly convex quadrilateral, then replacement of the common edge by the alternative diagonal of the quadrilateral does not increase the minimum of the six angles in the two triangles making up the quadrilateral. That a triangulation is constrained Delaunay if and only if it is constrained locally equiangular has been proven in [8]. On the other hand, given S and E as above, T a triangulation for S constrained by E, we say that T *constrained by E satisfies the empty circle criterion on a local basis* if given any two triangles t, t' in T that share a common edge not contained in any edge belonging to E, then the vertex of t' not in t is not inside the circumcircle of t. That a triangulation is constrained locally equiangular if and only if it is constrained satisfying the empty circle criterion on a local basis has been proven in [4], [8].

Algorithms for the computation of a Delaunay triangulation for the vertices of a polygon constrained by the boundary of the polygon have been presented in [4], [8], [9], the algorithm in [8] having complexity $O(n \log n)$, where n is the number of vertices of the polygon. As for the general problem of computing a Delaunay triangulation for a set of n points constrained by a set of line segments, an $O(n^2)$ algorithm has been presented in [8], $O(n \log n)$ divide-and-conquer algorithms have been presented in [3], [12], and an $O(n \log n)$ plane-sweep algorithm has been presented in [11]. Each one of these algorithms has the disadvantage that the set

of line segments must be known before the execution of the algorithm.

In [5] a method has been presented for the incremental computation of a constrained Delaunay triangulation by stepwise insertion of points and line segments. Accordingly, algorithms are presented in [5] for point insertion and line segment insertion into a constrained Delaunay triangulation. Independently, the algorithm for line segment insertion was also presented in [1]. In the following section, we describe a new version of the segment insertion algorithm that works in the same manner in which the old one does, but that has the additional capability of not deleting the triangles intersected by the line segment, transforming them instead through edge-swapping (Lawson's transformation [7]). In Section 3, we take advantage of the fact that edge-swapping generalizes to 3−dimensional space and present what would be considered the generalization to 3−dimensional space of the new line insertion algorithm without the optimization step for the Delaunay property. It is shown there that for certain cases the failure of this algorithm to insert a line segment into a tetrahedralization is an indication that it cannot be done. Finally, in the same section, 3−dimensional problems are identified that can be approached algorithmically as if they are 2−dimensional.

2. Segment insertion by edge-swapping

Let T be a triangulation in the plane, not necessarily Delaunay, let P_1, P_2, $P_1 \neq P_2$, be vertices in T, and let T^* be the collection of triangles in T whose interiors are intersected by the line segment with endpoints P_1, P_2. We say that the line segment with endpoints P_1, P_2 *has been inserted into* T *producing* $\hat{T}$ if $\hat{T}$ is a triangulation for the vertices of T such that the line segment is the union of one or more edges in $\hat{T}$ and each triangle in $T \setminus T^*$ is also in $\hat{T}$. In what follows, and assuming that T is constrained Delaunay, we present procedure INSERT_SEGMENT which inserts the line segment with endpoints P_1, P_2 into the triangulation T by edge-swapping (Lawson's transformation [7]), producing a constrained Delaunay triangulation with the line segment as an additional constraint. Without any loss of generality, we assume that the line segment is not an edge in T and that its relative interior does not contain any vertices in T.

In [1] and [5] this algorithm was presented but without edge-swapping. This older version consists essentially of two steps. In the first step, the triangles whose interiors are intersected by the line segment are detected and deleted so that a non-triangulated region inside the convex hull of the original triangulation results. In the second step, this region is divided into two polygons separated by the line segment, and a Delaunay triangulation is then computed for each polygon. Each polygon satisfies the property that each point in the polygon is visible through the polygon from the line segment, i. e. given a point P in the

polygon but not in the line segment, there exists a point P' in the relative interior of the line segment such that the relative interior of the line segment with endpoints P, P' is contained in the interior of the polygon. Because of this property, each polygon can be triangulated through an incremental insertion of triangles in the polygon, and optimized for the Delaunay property with procedures based on the empty circle criterion. Outlines of this older version, justifications, optimization procedures, and related results can be found in [1], [2], [5], [6].

The new version of the algorithm presented here works essentially in the same manner in which the old one does, thus producing exactly the same triangles, but has the capability through edge-swapping of maintaining at all times a complete triangulation for the vertices in the original triangulation. This will be illustrated below with an example.

In the following, we list and describe, in the order of their first appearance in procedure INSERT_SEGMENT, procedures used there as primitives.

INTERSECTED_TRIANGLES(T, T^*, P_1, P_2, Q, t_F): Assuming that P_1, P_2, $P_1 \neq P_2$, are vertices in triangulation T, this procedure identifies T^*, $T^* \subseteq T$, which is made up of those triangles in T with interiors intersected by the line segment with endpoints P_1, P_2. It also locates t_F and Q, where t_F is the triangle in T^* with P_1 as one of its vertices and Q is any one of the vertices of t_F different from P_1.

NEXT_TRIANGLE(T, P_1, P_2, t_P, t_C): Assuming that P_1, P_2, $P_1 \neq P_2$, are vertices in triangulation T, that the line segment with endpoints P_1, P_2 intersects the interior of t_P, a triangle in T, and that P_2 is not a vertex of t_P, this procedure locates triangle t_C in T which shares a facet with t_P intersected by the line segment, and which is closer to P_2 than t_P in the direction of the line segment.

NEXT_VERTEX(t_P, t_C, P): Assuming that t_P and t_C are adjacent triangles in some triangulation, this procedure locates vertex P of t_C not in t_P.

PREVIOUS_VERTEX(t_C, P_1, P_2, P, Q): Assuming that P is a vertex of triangle t_C and that the line segment with endpoints P_1, P_2 intersects exactly one of the edges of t_C with P as an endpoint, this procedure locates the vertex Q of t_C for which the line segment does not intersect the edge with endpoints P, Q.

STRICT_CONVEXITY($t_C, t_P, flag2$): Assuming that t_C and t_P are adjacent triangles in some triangulation, and that $flag2$ equasl 1, this procedure sets $flag2$ to zero whenever the

union of t_C and t_P is not a strictly convex quadrilateral.

EDGE_SWAP(t_C, t_P, Q, T, T^*): Assuming that t_C, t_P are adjacent triangles in T^*, $T^* \subseteq T$, whose union is a strictly convex quadrilateral, and that Q is one of the vertices that t_C and t_P have in common, this procedure transforms t_C, t_P, and therefore in the same manner T^* and T, through the replacement of the common edge by the alternative diagonal of the quadrilateral in such a way that Q is the vertex of the transformed t_P not in the transformed t_C.

OPTIMIZE(T, T^*, t_P, P, Q, R): Assuming that P, Q, R are the vertices of triangle t_P in T^*, $T^* \subseteq T$, this procedure is essentially the same as procedure UPDTRI in [1], which transforms T^*, and therefore in the same manner T, through edge-swapping in such a way that if after the execution of this procedure t is a triangle in T^* that either equals t_P or that was not in T^* before the execution of the procedure, then t satisfies the following properties:
(1) P is a vertex of t.
(2) The two edges of t with P as an endpoint intersect the line segment with endpoints Q, R.
(3) The circumcircle of t does not contain in its interior any vertex P' in T^* for which the relative interior of the line segment with endpoints P, P' lies entirely in the interior of the union of the triangles in T^* and intersects the edge of t that does not have P as an endpoint.

PREVIOUS_TRIANGLE(T, P_1, P_2, t_C, t_P): Assuming that P_1, P_2, $P_1 \neq P_2$, are vertices in triangulation T, that the line segment with endpoints P_1, P_2 intersects the interior of t_C, a triangle in T, and that P_1 is not a vertex of t_C, this procedure locates triangle t_P in T which shares a facet with t_C intersected by the line segment, and which is closer to P_1 than t_C in the direction of the line segment.

THIRD_VERTEX(t_C, R, P, Q): Assuming that R, P, $R \neq P$, are vertices of triangle t_C, this procedures identifies Q, a vertex of t_C with $Q \neq R$ and $Q \neq P$.

The outline of INSERT_SEGMENT follows. Throughout the procedure and for the purpose of keeping track of where in T the optimization procedure can be applied, a collection of triangles T^* is maintained, $T^* \subseteq T$, which is made up of those triangles or their transformations through edge-swapping (with procedures EDGE_SWAP and OPTIMIZE) that initially are intersected by the relative interior of the line segment with endpoints P_1, P_2. Also, throughout the procedure, a function F is defined on certain elements of $\{1, 2\} \times T^*$. Essentially, given t in T^* with interior intersected by the line segment, it is assumed that

the line segment intersects exactly two edges of t at different points. $F(1,t)$, $F(2,t)$ are then defined to represent the endpoints of the edge of t not intersected by the line segment, in such a way that $F(2,t)$ also represents an endpoint of the one of the two edges of t intersected by the line segment that is closer to P_2 in the direction of the line segment. Finally, we notice that if the optimization steps are eliminated in INSERT_SEGMENT (steps 20 and 32) then the procedure simply becomes one for inserting a line segment into a triangulation.

```
    procedure INSERT_SEGMENT(T, P1, P2)
        begin
1.      INTERSECTED_TRIANGLES(T, T*, P1, P2, Q, tF);
2.      F(1, tF) := P1; F(2, tF) := Q; flag1 := 1;
3.      while (flag1 = 1) do
            begin
4.          tP := tF;
5.          NEXT_TRIANGLE(T, P1, P2, tP, tC);
6.          NEXT_VERTEX(tP, tC, P);
7.          if (P ≠ P2) then
                begin
8.              PREVIOUS_VERTEX(tC, P1, P2, P, Q);
9.              tF := tC
                end
            else
                begin
10.             Q := F(2, tP); flag1 := 0
                end
11.         if (F(1, tP) = P1) then F(2, tP) := Q;
12.         F(1, tC) := Q; F(2, tC) := P; flag2 := 1;
13.         while (flag2 = 1) do
                begin
14.             STRICT_CONVEXITY(tC, tP, flag2);
15.             if (flag2 = 1) then
                    begin
16.                 R := F(1, tP); tL := tC;
17.                 EDGE_SWAP(tC, tP, Q, T, T*);
18.                 if (tF = tL) then tF := tC;
19.                 if (F(1, tC) = F(2, tP)) then
                        begin
```

```
20.                 OPTIMIZE(T, T*, t_P, P, Q, R);
21.                 F(1, t_C) := R; Q := R
                    end
                else
                    begin
22.                 F(1, t_C) := R; F(2, t_C) := F(2, t_P);
23.                 F(1, t_P) := Q; F(2, t_P) := P; t_C := t_P
                    end
24.                 if (R ≠ P_1) then
                    begin
25.                 PREVIOUS_TRIANGLE(T, P_1, P_2, t_C, t_P);
26.                 if (F(1, t_P) = P_1) then F(2, t_P) := Q;
27.                 if (P = P_2) then
                        begin
28.                     Q := F(2, t_P); F(1, t_C) := Q
                        end
                    end
                else
                    begin
29.                 flag2 := 0;
30.                 if (P = P_2) then
                        begin
31.                     THIRD_VERTEX(t_C, R, P, Q);
32.                     OPTIMIZE(T, T*, t_C, P, Q, R)
                        end
                    else
                        begin
33.                     NEXT_TRIANGLE(T, P_1, P_2, t_C, t_N);
34.                     F(2, t_C) := F(1, t_N)
                        end
                    end
                end
            end
        end
    end
```

Algorithm INSERT_SEGMENT has been mostly justified in [1], [6]. In what follows, we

illustrate with an example the way in which the algorithm works and the new aspects of its justification. Since the insertion of the line segment is the significant aspect of the algorithm we do not assume that the initial triangulation is Delaunay and ignore the optimization steps throughout. Starting with triangulation (i) in Figure 1, we enumerate and describe in the order of their executions the crucial steps of INSERT_SEGMENT that are executed in order to obtain the desired triangulation, triangulation (viii) in Figure 1, and all other intermediate triangulations, triangulations (ii)-(vii) also in Figure 1. Here, given points X, Y, Z, whenever we refer to the triangle XYZ it is implied that a triangle exists in the plane with vertices X, Y, Z, and that this is the order in which they appear in the boundary of the triangle in a counterclockwise direction around the interior of the triangle. Given points W, X, Y, Z, a similar assumption goes along with any reference to the quadrilateral $WXYZ$.

1. Step 1 (triangulation (i)): T^* is obtained and it is made up of the triangles intersected by line segment P_1P_2: P_1AF, FAE, EAB, EBD, DBC, DCP_2. The first triangle is $t_F = P_1AF$.

2. Step 5 (triangulation (i)): The next triangle is $t_C = FAE$.

3. Step 6 (triangulation (i)): The next vertex is $P = E$.

4. Step 14 (triangulation (i)): The quadrilateral P_1AEF is not strictly convex ($flag2$ is set to zero, no edge-swapping is possible, and a new P must be obtained).

5. Step 5 (triangulation (i)): The next triangle is $t_C = EAB$.

6. Step 6 (triangulation (i)): The next vertex is $P = B$.

7. Step 14 (triangulation (i)): The quadrilateral $FABE$ is strictly convex.

8. Step 17 (triangulation (ii)): Triangles FAE, EAB are transformed into triangles FAB, FBE through edge-swapping; t_C becomes FAB.

9. Step 14 (triangulation (ii)): The quadrilateral P_1ABF is strictly convex.

10. Step 17 (triangulation (iii)): Triangles P_1AF, FAB are transformed into triangles P_1AB, P_1BF through edge-swapping; t_C becomes P_1BF.

11. Step 29 (triangulation (iii)): Because P_1 is a vertex of t_C, $flag2$ is set to zero (a new P must be obtained).

12. Step 5 (triangulation (iii)): The next triangle is $t_C = EBD$.

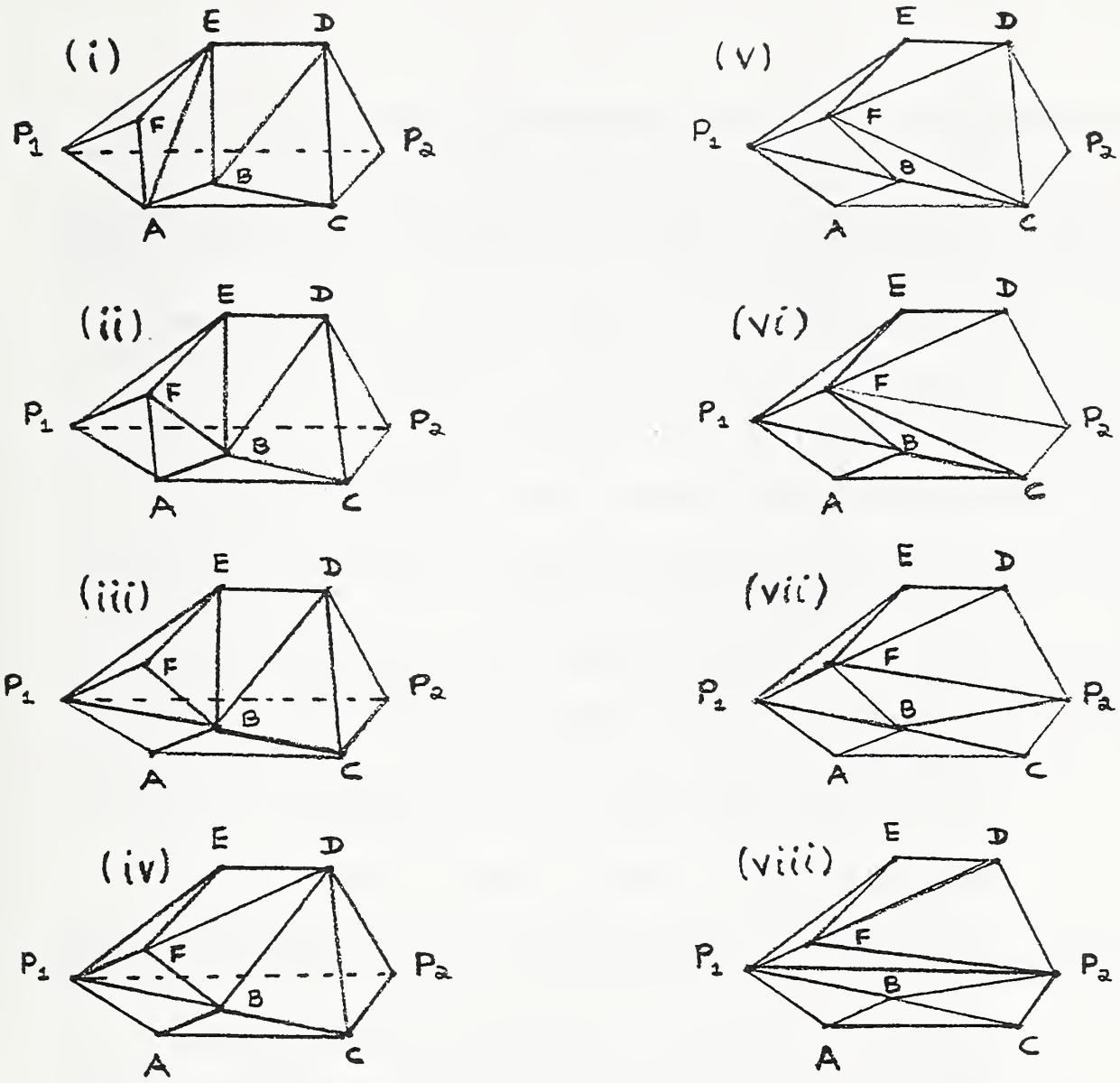

Figure 1: A triangulation and the insertion into it of line segment with endpoints P_1, P_2.

13. Step 6 (triangulation (iii)): The next vertex is $P = D$.

14. Step 14 (triangulation (iii)): The quadrilateral $FBDE$ is strictly convex.

15. Step 17 (triangulation (iv)): Triangles FBE, EBD are transformed into triangles FBD, FDE through edge-swapping; t_C becomes FBD.

16. Step 14 (triangulation (iv)): The quadrilateral P_1BDF is not strictly convex.

17. Step 5 (triangulation (iv)): The next triangle is $t_C = BCD$.

18. Step 6 (triangulation (iv)): The next vertex is $P = C$.

19. Step 14 (triangulation (iv)): The quadrilateral $FBCD$ is strictly convex.

20. Step 17 (triangulation (v)): Triangles FBD, BCD are transformed into triangles FBC, FCD through edge-swapping; t_C becomes FBC.

21. Step 14 (triangulation (v)): The quadrilateral P_1BCF is not strictly convex.

22. Step 5 (triangulation (v)): The next triangle is $t_C = DCP_2$.

23. Step 6 (triangulation (v)): The next vertex is $P = P_2$.

24. Step 10 (triangulation (v)): $flag1$ is set to zero (this is the last P).

25. Step 14 (triangulation (v)): The quadrilateral FCP_2D is strictly convex.

26. Step 17 (triangulation (vi)): Triangles FCD, DCP_2 are transformed into triangles FCP_2, FP_2D through edge-swapping; t_C becomes FCP_2.

27. Step 14 (triangulation (vi)): The quadrilateral $FBCP_2$ is strictly convex.

28. Step 17 (triangulation (vii)): Triangles FBC, FCP_2 are transformed into triangles FBP_2, BCP_2 through edge-swapping; t_C becomes FBP_2.

29. Step 14 (triangulation (vii)): The quadrilateral P_1BP_2F is strictly convex.

30. Step 17 (triangulation (viii)): Triangles P_1BF, FBP_2 are transformed into triangles P_1BP_2, P_1P_2F through edge-swapping (line segment P_1P_2 has now been inserted).

31. Step 29 (triangulation (viii)): $flag2$ is set to zero (combined with $flag1$ equal to zero, it signifies that the execution of the algorithm must come to an end).

We can use the same example to illustrate the significant new aspects of the justification of the algorithm. A rigorous proof would involve showing that the triangles produced by the old version of the algorithm are also produced by the new one in exactly the same manner. Here we avoid a rigoruous proof and concentrate our efforts on simply pointing out the reasons why triangulation (i) in Figure 1 is transformed into triangulation (iii), thus producing triangle P_1AB which is a triangle that the old version would also produce. Equivalently, we point out the reasons why the quadrilaterals $FABE$ and P_1ABF are strictly convex under the assumptions that the quadrilateral P_1AEF is not strictly convex and that the triangle P_1AB is also produced by the old version of the algorithm. The latter assumption is equivalent to the fact that the internal angle of triangle P_1AB at A measures less than π radians. In order to show that the quadrilateral $FABE$ is strictly convex we must show that each of its internal angles measures less than π radians. The internal angles of the quadrilateral at F and B satisfy this property because they are also internal angles, respectively, of triangles FAE and EAB. Since the internal angle of the quadrilateral at A is contained in the internal angle of triangle P_1AB at A it follows that it must also satisfy the property. Finally, because of the way in which line segment P_1P_2 intersects the quadrilateral P_1AEF and the fact that the quadrilateral is not strictly convex it follows that the ray with origin E through F must intersect the line segment. This, combined with the fact that the ray with origin E through B also intersects the line segment (E is above it, B is below it), implies that the internal angle of the quadrilateral at E measures less than π radians. That the quadrilateral P_1ABF is also strictly convex follows through essentially similar arguments.

3. The 3−dimensional version of the algorithm

Let T be a tetrahedralization, not necessarily Delaunay, let P_1, P_2, $P_1 \neq P_2$, be vertices in T, and let T^* be the collection of tetrahedra in T each of which is intersected by the line segment with endpoints P_1, P_2 at either its interior or the relative interior of one of its facets. We say that the line segment with endpoints P_1, P_2 *can be inserted into* T if a tetrahedralization $\hat{T}$ for the vertices of T exists such that the line segment is the union of one or more edges in $\hat{T}$ and each tetrahedron in $T \setminus T^*$ is also in $\hat{T}$. In what follows, we take advantage of the fact that edge-swapping generalizes to 3−dimensional space and present procedure 3D_INSERT_ATTEMPT which attempts to insert the line segment with endpoints P_1, P_2 into T, and which can be considered as the generalization to 3−dimensional space of procedure INSERT_SEGMENT of the previous section without the optimization step for the Delaunay property. We notice that only the case for which the relative interior of the line segment with endpoints P_1, P_2 does not intersect any edges in the tetrahedralization is addressed in what follows.

In the following, we list and describe, in the order of their first appearance in procedure 3D_INSERT_ATTEMPT, procedures used there as primitives.

FIRST_TETRAHEDRON(T, P_1, P_2, Q, t_F): Assuming that P_1, P_2, $P_1 \neq P_2$, are vertices in tetrahedralization T, this procedure identifies t_F, where t_F is the tetrahedron in T with P_1 as one of its vertices and interior intersected by the line segment with endpoints P_1, P_2. It also identifies Q, where Q is any one of the vertices of t_F different from P_1.

NEXT_TETRAHEDRON(T, P_1, P_2, t_P, t_C): Assuming that P_1, P_2, $P_1 \neq P_2$, are vertices in tetrahedralization T, that the line segment with endpoints P_1, P_2 intersects the interior of t_P, a tetrahedron in T, and that P_2 is not a vertex of t_P, this procedure locates tetrahedron t_C in T which shares a facet with t_P intersected by the line segment, and which is closer to P_2 than t_P in the direction of the line segment.

NEXT_VERTEX(t_P, t_C, P): Assuming that t_P and t_C are adjacent tetrahedra in some tetrahedralization, this procedure locates vertex P of t_C not in t_P.

PREVIOUS_VERTEX(t_C, P_1, P_2, P, Q): Assuming that P is a vertex of tetrahedron t_C and that the line segment with endpoints P_1, P_2 intersects exactly one of the facets of t_C with P as a vertex, this procedure locates the vertex Q of t_C for which the line segment does not intersect the facets of t_C that have in common the edge with endpoints P, Q.

STRICT_CONVEXITY($t_C, t_P, flag2$): Assuming that t_C and t_P are adjacent tetrahedra in some tetrahedralization, and that $flag2$ equals 1, this procedure sets $flag2$ to zero whenever the union of t_C and t_P is not a strictly convex hexahedron.

COMMON_VERTEX(t_C, t_P, Q, S, U): Assuming that t_C and t_P are adjacent tetrahedra in some tetrahedralization, and that Q and S are vertices, not necessarily different, common to t_C and t_P, this procedure locates U, a vertex common to t_C and t_P such that $U \neq Q$ and $U \neq S$.

TWO_THREE(T, t_C, t_P, P, R, Q, U): Assuming that t_C, t_P are adjacent tetrahedra in T whose union is a strictly convex hexahedron, that P is the vertex of t_C not in t_P, that R is the vertex of t_P not in t_C, and that Q, U, $Q \neq U$, are vertices that t_C and t_P have in common, this procedure transforms T by transforming t_C and t_P into the three tetrahedra that have in common the edge with endpoints P, R, in such a way that t_C becomes the one

of the three tetrahedra that does not have Q as a vertex, and t_P the one that has Q and U as vertices.

FACET_INTERSECT$(P, R, U, P_1, P_2, flag2)$: Assuming that P, R, U are the vertices of a facet of some tetrahedron, and that $flag2$ equals 1, this procedure sets $flag2$ to zero whenever the line segment with endpoints P_1, P_2 does not intersect the relative interior of the triangle with vertices P, R, U.

PREVIOUS_TETRAHEDRON(T, P_1, P_2, t_C, t_P): Assuming that P_1, P_2, $P_1 \neq P_2$, are vertices in tetrahedralization T, that the line segment with endpoints P_1, P_2 intersects the interior of t_C, a tetrahedron in T, and that P_1 is not a vertex of t_C, this procedure locates tetrahedron t_P in T which shares a facet with t_C intersected by the line segment, and which is closer to P_1 than t_C in the direction of the line segment.

The outline of 3D_INSERT_ATTEMPT follows. Throughout the procedure, and serving the same purpose as in procedure INSERT_SEGMENT of the previous section, a function F is defined on certain elements of $\{1, 2\} \times T$. Essentially, given t in T with interior intersected by the line segment with endpoints P_1, P_2, it is assumed that the line segment intersects exactly two facets of t at different points. $F(1, t)$, $F(2, t)$ are then defined to represent the endpoints of the edge of t that is shared by the two facets of t that the line segment does not intersect, in such a way that $F(2, t)$ also represents a vertex of the one of the two facets of t intersected by the line segment that is closer to P_2 in the direction of the line segment. Also, throughout the procedure, a variable $flag$ is defined which at the end of the execution of the procedure equals 1 if the line segment has been inserted, zero otherwise. We point out here that in the procedure, tetrahedra can only be transformed at steps 21 and 27 with procedure TWO_THREE which, when possible, transforms two adjacent tetrahedra into three in the obvious fashion.

procedure 3D_INSERT_ATTEMPT$(T, P_1, P_2, flag)$
begin
1. $flag := 0$;
2. FIRST_TETRAHEDRON(T, P_1, P_2, Q, t_F);
3. $F(1, t_F) := P_1$; $F(2, t_F) := Q$; $flag1 := 1$;
4. **while** $(flag1 = 1)$ **do**
begin
5. $t_P := t_F$;
6. NEXT_TETRAHEDRON(T, P_1, P_2, t_P, t_C);

```
7.      NEXT_VERTEX(t_P, t_C, P);
8.      if (P ≠ P_2) then
            begin
9.          PREVIOUS_VERTEX(t_C, P_1, P_2, P, Q);
10.         t_F := t_C
            end
        else
            begin
11.         Q := F(2, t_P); flag1 := 0
            end
12.     if (F(1, t_P) = P_1) then F(2, t_P) := Q;
13.     F(1, t_C) := Q; F(2, t_C) := P; flag2 := 1;
14.     while (flag2 = 1) do
            begin
15.         STRICT_CONVEXITY(t_C, t_P, flag2);
16.         if (flag2 = 1) then
                begin
17.             R := F(1, t_P); S := F(2, t_P);
18.             COMMON_VERTEX(t_C, t_P, Q, S, U);
19.             if (F(1, t_C) = F(2, t_P)) then
                    begin
20.                 t_L := t_C;
21.                 TWO_THREE(T, t_C, t_P, P, R, Q, U);
22.                 if (t_F = t_L) then t_F := t_C;
23.                 F(1, t_C) := R; Q := R
                    end
                else
                    begin
24.                 FACET_INTERSECT(P, R, U, P_1, P_2, flag2);
25.                 if (flag2 = 1) then
                        begin
26.                     t_L := t_C;
27.                     TWO_THREE(T, t_C, t_P, P, R, Q, U);
28.                     if (t_F = t_L) then t_F := t_C;
29.                     F(1, t_C) := R; F(2, t_C) := F(2, t_P);
30.                     F(1, t_P) := Q; F(2, t_P) := P; t_C := t_P
```

```
                    end
                end
31.             if (flag2 = 1) then
                    begin
32.                 if (R ≠ P1) then
                        begin
33.                     PREVIOUS_TETRAHEDRON(T, P1, P2, tC, tP);
34.                     if (F(1, tP) = P1) then F(2, tP) := Q;
35.                     if (P = P2) then
                            begin
36.                         Q := F(2, tP); F(1, tC) := Q
                            end
                        end
                    else
                        begin
37.                     flag2 := 0;
38.                     if (P = P2) then flag := 1
                        else
                            begin
39.                         NEXT_TETRAHEDRON(T, P1, P2, tC, tN);
40.                         F(2, tC) := F(1, tN)
                            end
                        end
                end
            end
        end
    end
```

Experiments show that procedure 3D_INSERT_ATTEMPT seldom succeeds in inserting a line segment. However, this may just be an indication that it is seldom possible to insert a line segment into a tetrahedralization. Let T, P_1, P_2, T^* be as above. In what follows, we show that for a certain kind of T^* the failure of the procedure simply signifies that the line segment can not be inserted into T. Namely, we prove the following proposition.

Proposition 1. If points Q_1, Q_2 exist, $Q_1 \neq Q_2$, that are vertices of every tetrahedron in T^*, then at the end of the execution of 3D_INSERT_ATTEMPT, variable *flag* equals 1 if

and only if the line segment can be inserted into T.

Proof. That $flag$ equal to 1 implies that the line segment can be inserted into T follows trivially. Thus, it remains to be shown that if $flag$ equals zero then the line segment can not be inserted into T.
For some positive integer n, let t_i, $i = 1, \ldots, n$, be the tetrahedra in T^* in the order in which they are intersected by the line segment from P_1 to P_2.
At the end of the execution of the procedure let T^{**} be the collection of tetrahedra in T that are intersected by the relative interior of the line segment, and for some positive integer m, let t'_i, $i = 1, \ldots, m$, be the tetrahedra in T^{**} in the order in which they are intersected by the line segment from P_1 to P_2.
Clearly, $n \geq m$, and since $flag$ equals zero it follows that $m \geq 3$.
Let R_0 equal P_1, and, inductively, for each i, $i = 1, \ldots, n$, let R_i be the vertex of t_i different from R_{i-1}, Q_1, and Q_2. Similarly, points R'_i, $i = 0, \ldots, m$ are defined with respect to t'_i, $i = 1, \ldots, m$. Figure 2 illustrates an example of the facets of the tetrahedra in T^* that do not have Q_2 as a vertex, and of R_i, $i = 0, \ldots, n$, R'_i, $i = 0, \ldots, m$, where n equals 14 and m equals 5.
We define a function f from $\{0, \ldots, m\}$ into $\{0, \ldots, n\}$ in such a way that for each i, $i = 0, \ldots, m$, R'_i equals $R_{f(i)}$. Based on this definition, for each i, $i = 1, \ldots, m$, we then define sets $W_i \subset \{R_0, \ldots, R_n\}$, by

$$W_i \equiv \{R_{f(i-1)} = R'_{i-1}, R_{f(i-1)+1}, \ldots, R_{f(i)} = R'_i\}.$$

From the definition of T^{**} it follows that given i, $2 \leq i \leq m$, the union of t'_{i-1} and t'_i is not a strictly convex hexahedron (step 15 of procedure). Thus, it is not possible to insert the line segment and at the same time to have a new tetrahedron in T with vertices Q_1, R'_{i-2}, R'_{i-1}, R'_i. The same is true for a tetrahedron with vertices Q_2, R'_{i-2}, R'_{i-1}, R'_i. From this and the fact that it is always true that $F(1, t_C)$ equals $F(2, t_P)$ in step 19 of the procedure, it follows that for each i, $i = 2, \ldots, m$, it is not possible to insert the line segment and at the same time to have a new tetrahedron with one vertex equal to either Q_1 or Q_2, two vertices in W_{i-1}, and one vertex in $W_i \setminus \{R'_{i-1}\}$.
In what follows, we assume that the line segment can be inserted into T. Thus, we must assume that T^* has been transformed in such a way that the line segment is one of its edges. Clearly, in the transformed T^*, which we denote by $\hat{T}^*$, only one tetrahedron can have both Q_1 and Q_2 as vertices, namely the tetrahedron with vertices Q_1, Q_2, P_1, and P_2. All other tetrahedra with either Q_1 or Q_2 as a vertex have in addition three vertices of the form R_j, R_k, R_l, $0 \leq j < k < l \leq n$.

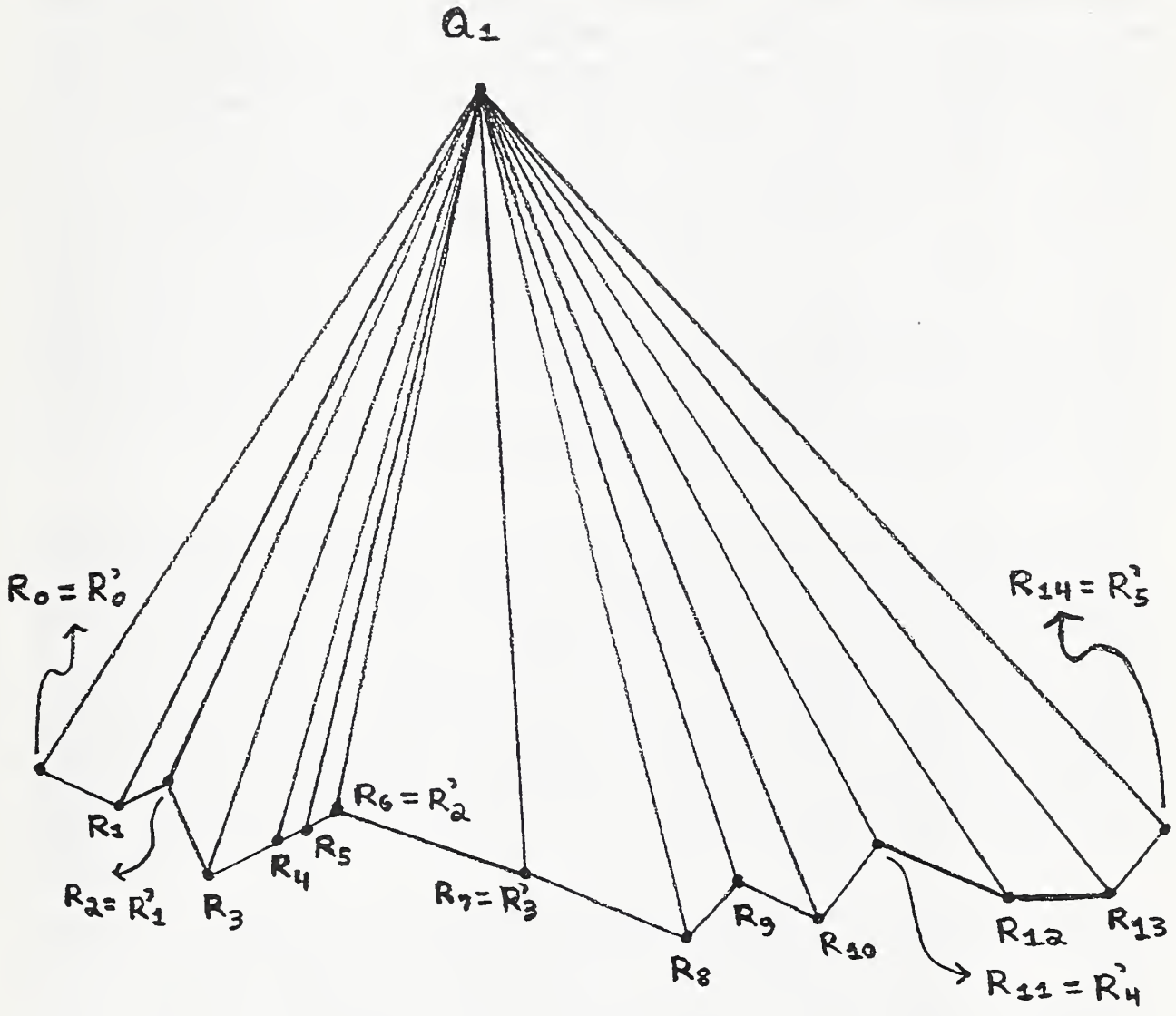

Figure 2: Facets of tetrahedra in T^* that do not have Q_2 as a vertex, and points R_i, $i = 0, \ldots, n$, R'_i, $i = 0, \ldots, m$.

For some integer n', $1 \leq n' < n$, we define integers h_i, l_i, $i = 0, \ldots, n'$, as follows. We let h_0 and l_0 equal 0 and n, respectively. Inductively, given i, $i > 0$, we assume integers h_{i-1}, l_{i-1}, $0 \leq h_{i-1} < l_{i-1} \leq n$, have been defined such that for integers j, k, $1 \leq j < k \leq m$, $R_{h_{i-1}} \in W_j$, $R_{l_{i-1}} \in W_k$, $R_{h_{i-1}} \neq R'_j$, $R_{l_{i-1}} \neq R'_{k-1}$, and the triangle with vertices Q_1, $R_{h_{i-1}}$, $R_{l_{i-1}}$ is a facet of a tetrahedron in $\hat{T}^*$. Then from the geometry of T^* and the last fact about the triangle with vertices Q_1, $R_{h_{i-1}}$, $R_{l_{i-1}}$, it follows that integers h_i, l_i exist, $h_{i-1} < h_i < l_i \leq l_{i-1}$, for which $R_{h_i} \in W_j$, $R_{l_i} \notin W_j$, and the tetrahedron with vertices Q_1, $R_{h_{i-1}}$, R_{h_i}, R_{l_i} belongs to $\hat{T}^*$. If R_{l_i} belongs to W_{j+1} then we let n' equal i. That for some i, $1 \leq i < n$, and some j, $1 \leq j < m$, R_{l_i} belongs to W_{j+1}, while $R_{h_{i-1}}$, R_{h_i} belong to W_j, follows from the fact that $\{h_i\}$ is an increasing sequence of integers bounded above by $\{l_i\}$ which is itself a non-increasing sequence of integers. Thus, n' is well defined. However, this is a contradiction, for it implies for some j, $1 \leq j < m$, the existence of a tetrahedron in $\hat{T}^*$ with one vertex equal to Q_1, two vertices in W_j, namely $R_{h_{n'-1}}$ and $R_{h_{n'}}$, and one vertex in $W_{j+1} \setminus \{R'_j\}$, namely $R_{l_{n'}}$. This completes the proof of the proposition.

Finally, in what follows we shed more light on the fundamental differences between the 2−dimensional and the 3−dimensional line insertion problems by identifying those 3−dimensional problems that can be approached algorithmically as 2−dimensional problems. In particular, we look at the following problem: Let T be a triangulation in the $x - y$ plane of 3−dimensional space, let P_1, P_2, $P_1 \neq P_2$, be vertices in T such that the line segment with endpoints P_1, P_2 is not an edge in T and its relative interior does not contain any vertices in T, and let T^* be the collection of triangles in T that are intersected by the relative interior of the line segment. Let $\hat{T}$ be a collection of contiguous tetrahedra in 3−dimensional space that have a vertex $\hat{Q}$ in common and let $\bar{T}$ be the collection of 2−dimensional triangles in 3−dimensional space that are the facets of the tetrahedra in $\hat{T}$ that do not have $\hat{Q}$ as a vertex. Assume that the perpendicular projection of $\bar{T}$ onto the $x - y$ plane equals T^* and that each triangle in T^* is the perpendicular projection of only one triangle in $\bar{T}$. Assume, in addition, that P'_1, P'_2 are the vertices in $\bar{T}$ whose perpendicular projections onto the $x - y$ plane are P_1, P_2, respectively, and, without any loss of generality, for the purpose of executing 3D_INSERT_ATTEMPT with $\hat{T}$, P'_1, P'_2 as input, that $\hat{T}$ is a complete tetrahedralization for its vertices. What conditions must the tetrahedra in $\hat{T}$ satisfy so that the relative interior of the line segment with endpoints P'_1, P'_2 lies in the interior of the union of the tetrahedra in $\hat{T}$ and the line segment can then be inserted into $\hat{T}$ with procedure 3D_INSERT_ATTEMPT in a manner that mimics exactly what the 2−dimensional algorithm (without the optimization steps) does when inserting into T the line segment with endpoints P_1, P_2?

In order to answer the above question we further formalize the notation. For a positive

integer n, let P_i, $i = 1, \ldots, n$, be distinct points in the $x-y$ plane of 3−dimensional space, and for each i, $i = 1, \ldots, n$, let x_i, y_i be the $x-$ and $y-$coordinates, respectively, of P_i. Given a triangulation T for the set of points P_i, $i = 1, \ldots, n$, we say that a collection T' of distinct 2−dimensional triangles in 3−dimensional space is a *triangulation that generalizes* T (a *generalized triangulation* for short) if numbers z_i, $i = 1, \ldots, n$, exist such that if points P_i', $i = 1, \ldots, n$, in 3−dimensional space are defined by setting P_i' equal to (x_i, y_i, z_i) for each i, $i = 1, \ldots, n$, then the set of vertices of T' equals the set of points P_i', $i = 1, \ldots, n$, and the perpendicular projection onto the $x-y$ plane of T' is T.

Let P_i, P_i', x_i, y_i, z_i, $i = 1, \ldots, n$, T, T' be as above. Assume that the line segment with endpoints P_1, P_2 is not an edge in T and that its relative interior does not contain any vertices in T. Let T^* be the collection of triangles in T that are intersected by the relative interior of this line segment, and let $\bar{T}$ be the collection of triangles in T' whose perpendicular projection onto the $x-y$ plane is T^*. For arbitrarily large positive z we let $\hat{Q}$ represent the point $(0, 0, z)$, and $\hat{T}$ the collection of tetrahedra obtained by computing the convex hulls of $\hat{Q}$ together with each of the triangles in $\bar{T}$. In what follows, we say that the line segment with endpoints P_1', P_2' *can be inserted into* $\hat{T}$ if a collection of tetrahedra $\tilde{T}$ exists such that the tetrahedra in $\tilde{T}$ have pair-wise disjoint interiors, the relative interior of the line segment is contained in the interior of the union of the tetrahedra in $\tilde{T}$, the line segment is an edge in $\tilde{T}$, $\tilde{T}$ and $\hat{T}$ have the same set of vertices, and the union of the tetrahedra in $\tilde{T}$ equals the union of the tetrahedra in $\hat{T}$. Accordingly, we say that the line segment with endpoints P_1', P_2' *can be inserted into the positive side of* T' if it can be inserted into $\hat{T}$. Based on these definitions, we notice that if the line segment with endpoints P_1', P_2' *satisfies the prerequisite for insertion into* $\hat{T}$, i. e. its relative interior lies entirely in $\hat{T}$ and does not intersect any edges of tetrahedra in $\hat{T}$, then an attempt can be made to insert it into $\hat{T}$ with procedure 3D_INSERT_ATTEMPT even though $\hat{T}$ is not necessarily a complete tetrahedralization for its vertices.

We assume that the line segment with endpoints P_1', P_2' satisfies the prerequisite for insertion into $\hat{T}$, that procedure INSERT_SEGMENT (without the optimization steps) has been executed for inserting into T the line segment with endpoints P_1, P_2, and that procedure EDGE_SWAP (step 17 of INSERT_SEGMENT) has been executed m times during the insertion. Similarly, we assume that procedure 3D_INSERT_ATTEMPT has been executed for attempting to insert into $\hat{T}$ the line segment with endpoints P_1', P_2' and that procedure TWO_THREE (steps 21 and 27 of 3D_INSERT_ATTEMPT) has been executed m' times during the attempt.

We define functions a, e from $\{1, \ldots, m\}$ into $\{(i, j) : 1 \leq i < j \leq n\}$ as follows: Given l, $1 \leq l \leq m$, we set $a(l)$ and $e(l)$ equal to (h, k) and (q, r), respectively, where h, k, q, r, $1 \leq h < k \leq n$, $1 \leq q < r \leq n$, are those integers for which after the l^{th} execution of

EDGE_SWAP in INSERT_SEGMENT the edge with endpoints P_h, P_k is the new edge in the triangulation and the edge with endpoints P_q, P_r is the edge that has been eliminated. Correspondingly, assuming $m' > 0$, we also define functions a', e' from $\{1, \ldots, m'\}$ into $\{(i,j) : 1 \leq i < j \leq n\}$ as follows: Given l, $1 \leq l \leq m'$, we set $a'(l)$ and $e'(l)$ equal to (h,k) and (q,r), respectively, where h, k, q, r, $1 \leq h < k \leq n$, $1 \leq q < r \leq n$, are those integers for which after the l^{th} execution of TWO_THREE in 3D_INSERT_ATTEMPT the edge with endpoints P'_h, P'_k is the edge that the three new tetrahedra have in common and the edge with endpoints P'_q, P'_r is the edge that the two eliminated tetrahedra had in common and that does not have $\hat{Q}$ as an endpoint. Clearly, $a(m)$ equals $(1,2)$, and if 3D_INSERT_ATTEMPT is successful then $m' > 0$ and $a'(m')$ also equals $(1,2)$.

Finally, in what follows, given integers h, k, q, r, $1 \leq h < k \leq n$, $1 \leq q < r \leq n$, we say that (h,k) *crosses* (q,r) if the relative interiors of the line segment with endpoints P_h, P_k and of the line segment with endpoints P_q, P_r have one and only one point in common. Assuming (h,k) crosses (q,r), let $\hat{x}$, $\hat{y}$ be the $x-$ and $y-$coordinates, respectively, of the point at which the line segment with endpoints P_h, P_k intersects the line segment with endpoints P_q, P_r, and let $\hat{z}_{hk}$, $\hat{z}_{qr}$ be the numbers for which the points defined by $(\hat{x}, \hat{y}, \hat{z}_{hk})$, $(\hat{x}, \hat{y}, \hat{z}_{qr})$ belong, respectively, to the line segment with endpoints P'_h, P'_k, and the line segment with endpoints P'_q, P'_r. Based on these definitions, we say then that (h,k) *is below* (q,r) if $\hat{z}_{hk} < \hat{z}_{qr}$. We are now ready to answer the question formulated above.

Proposition 2. The line segment with endpoints P'_1, P'_2 satisfies the prerequisite for insertion into $\hat{T}$, m equals m', and for each integer l, $l = 1, \ldots, m$, $a(l)$ equals $a'(l)$, and $e(l)$ equals $e'(l)$ so that the line segment can be inserted into $\hat{T}$ if and only if for each integer l, $l = 1, \ldots, m$, $e(l)$ is below $a(l)$.

Proof. The 'only if' part follows easily. In order to prove the 'if' part it suffices to prove that for each integer l, $l = 1, \ldots, m$, $e(l)$, which obviously crosses $(1,2)$, is below $(1,2)$. This will imply that the line segment satisfies the prerequisite for insertion in $\hat{T}$, and that $flag2$ always equals 1 in step 25 of 3D_INSERT_ATTEMPT (after the execution of procedure FACET_INTERSECT in step 24).

Let T^* be as defined above, and let T_0^* equal T^*. Inductively, for each l, $l = 1, \ldots, m$, let T_l^* be the collection of triangles in the $x-y$ plane of 3−dimensional space which is the transformation of T_{l-1}^* after the l^{th} edge swap.

Let $\bar{T}$ be as defined above. For each l, $l = 0, \ldots, m$, let $\bar{T}_l$ be the collection of distinct 2−dimensional triangles in 3−dimensional space whose perpendicular projection onto the $x-y$ plane equals T_l^*, and whose set of vertices equals that of $\bar{T}$.

For each l, $l = 0, \ldots, m$, we define a real-valued function f_l with domain the union of the triangles in T^* as follows. Given a point P in a triangle in T^* we let $\hat{x}$, $\hat{y}$ be the $x-$ and $y-$coordinates, respectively, of P, and let $f_l(P)$ be the unique number for which the point defined by $(\hat{x}, \hat{y}, f_l(P))$ belongs to a triangle in $\bar{T}_l$.
Given an integer l, $1 \leq l \leq m$, let h, k, q, r, $1 \leq h < k \leq n$, $1 \leq q < r \leq n$, be those integers for which $a(l)$ equals (h,k) and $e(l)$ equals (q,r). By definition T_l^* is the transformation of T_{l-1}^* obtained by replacing the edge with endpoints P_q, P_r by the edge with endpoints P_h, P_k. Clearly, the replaced edge is shared by two triangles in T_{l-1}^* whose union is a strictly convex quadrilateral and the new edge is the alternative diagonal of this quadrilateral. These observations and the fact that $e(l)$ is below $a(l)$ imply that f_{l-1} equals f_l everywhere except in the relative interior of the aforementioned quadrilateral in which f_{l-1} is strictly less than f_l. In particular, given a point P in the relative interior of the replaced edge, it then follows that $f_{l-1}(P) < f_l(P)$. Thus, since the edge with endpoints P_1, P_2 belongs to T_m^*, given an integer l, $1 \leq l \leq m$, and a point P which is the intersection of the edge with endpoints P_1, P_2 and the edge replaced in T_{l-1}^* during the l^{th} edge swap, it must follow that $f_{l-1}(P) < f_l(P) \leq f_m(P)$. Hence, $e(l)$ is below $(1,2)$ and the proof of the proposition is complete.

We notice that Proposition 2 provides conditions for identifying 3−dimensional problems that can be approached algorithmically as if they are 2−dimensional. However, it does not provide a method or procedure for selecting the vertices of $\hat{T}$ so that these conditions are satisfied. In what follows, we describe one such method that is based on the order in which the 2−dimensional swapping of edges occurs. As will be pointed out below, the implementation of this method requires that procedures INSERT_SEGMENT and 3D_INSERT_ATTEMPT be somewhat modified.

Let P and Q be variables as they appear in procedure INSERT_SEGMENT. We define variable W as the pair (P,Q). During the execution of INSERT_SEGMENT, W takes on different values, each value being taken on by W only once. Let W' be one such value and assume that while W equals W' procedure EDGE_SWAP in step 17 of INSERT_SEGMENT is executed at least once. Accordingly, we notice that W ceases to equal W' in one of two ways: when an execution of procedure STRICT_CONVEXITY returns a value of zero for $flag2$, and when $F(1, t_C)$ equals $F(2, t_P)$ in step 19 of INSERT_SEGMENT. When the first possibility occurs it signifies that while W equaled W' the swapping of edges (through the execution of EDGE_SWAP) did not lead to the creation of a triangle that does not intersect the line segment with endpoints P_1, P_2 (since the second possibility never occurred). Thus, we can think of the swapping of edges as being unnecessary while W equals a value for which the first

possibility occurs, and point out that procedure INSERT_SEGMENT can be modified so that such a value can be identified as soon as W is set to it (before any swapping of edges occurs) and W can then be set to its next value. Correspondingly, we can define variable W in the same manner with respect to variables P and Q in procedure 3D_INSERT_ATTEMPT. Here, again assuming that W' is one of the values that W takes on, we consider the possibility that W ceases to equal W' because an execution of procedure STRICT_CONVEXITY in step 15 of 3D_INSERT_ATTEMPT returns a value of zero for $flag2$. In a manner similar to what we did for the 2−dimensional algorithm, we point out that procedure 3D_INSERT_ATTEMPT can be modified so that such a value can be identified as soon as W is set to it (before any executions of procedure TWO_THREE in step 27 of 3D_INSERT_ATTEMPT take place while W is set to the value) and W can then be set to its next value.

In what follows, we assume that INSERT_SEGMENT and 3D_INSERT_ATTEMPT have been modified as just described. We also assume that T, T^*, n, P_i, P_i', $i = 1, \ldots, n$, $\hat{T}$, m, a, e are defined as above and that the modified version of INSERT_SEGMENT (without the optimization steps) has been executed for inserting into T the line segment with endpoints P_1, P_2. We let S^* be the set of vertices of T^*, and without any loss of generality we let P_3 be the point in S^* which is the last value that Q takes on during the execution of INSERT_SEGMENT (obtained through the execution of THIRD_VERTEX in step 31). We let n^* be the number of points in S^* and notice that $1 \leq n^* - 3 \leq m$ since for each point in S^* different from P_1, P_2, P_3, one execution of EDGE_SWAP with Q equal to the point takes place that produces a triangle t_P that does not intersect the line segment with endpoints P_1, P_2. We define a one-to-one function g from $\{1, \ldots, n^*\}$ onto S^* by setting $g(n^*)$, $g(n^* - 1)$, $g(n^* - 2)$ equal to P_1, P_2, P_3, respectively, and by setting for each i, $i = 1, \ldots, n^* - 3$, $g(i)$ equal to the point in S^* which is the value of Q the i^{th} time during the execution of INSERT_SEGMENT that the execution of EDGE_SWAP produces a triangle t_P that does not intersect the line segment with endpoints P_1, P_2. In addition, we define functions ini, fin from $\{1, \ldots, n^* - 3\}$ into $\{1, \ldots, m\}$, by setting for each i, $i = 1, \ldots, n^* - 3$, $ini(i)$ equal to the positive integer l for which the l^{th} time EDGE_SWAP is executed it is also the first time it is executed with Q equal to $g(i)$, and $fin(i)$ equal to the positive integer k for which the k^{th} time EDGE_SWAP is executed it is also the last time it is executed with Q equal to $g(i)$.

Assume that procedure INSERT_SEGMENT has been further modified so that it computes and returns variables that correspond to m, a, e, n^*, g, ini, fin. Based on this information, we now describe a procedure for computing for each integer i, $i = 1, \ldots, n^*$, a number z_i that can be used as the z−coordinate for the vertex of $\hat{T}$ that corresponds to $g(i)$ so that for each integer l, $l = 1, \ldots, m$, $e(l)$ is below $a(l)$. Essentially, the procedure consists of two steps. In the first step, the numbers z_{n^*-2}, z_{n^*-1}, z_{n^*} are selected arbitrarily from the set of real numbers. In the second step, inductively, for each i, $i = 1, \ldots, n^* - 3$, assuming

that the numbers $z_{i+1}, \ldots, z_{n^*}$ have been computed, z_i is selected arbitrarily from the set of possible values of z_i for which $e(l)$ is below $a(l)$ for each integer l, $l = ini(i), \ldots, fin(i)$. In order to see that this set can be computed under the given assumption in a manner consistent with the required conditions, it suffices to show that its computation depends solely on $z_{i+1}, \ldots, z_{n^*}$. For this purpose, let l, h, k, q, r, $ini(i) \leq l \leq fin(i)$, $1 \leq h < k \leq n$, $1 \leq q < r \leq n$, be integers such that $a(l)$ equals (h, k) and $e(l)$ equals (q, r). Then from the definitions of g and e, $g(i)$ must equal either P_q or P_r, and assuming without any loss of generality that $g(i)$ equals P_q, then from the definitions of g, a, and e, integers $i1$, $i2$, $i3$ must exist, $i + 1 \leq i1, i2, i3 \leq n^*$, such that P_h equals $g(i1)$, P_k equals $g(i2)$, and P_r equals $g(i3)$. Thus, the computation of the set of possible values of z_i for which $e(l)$ is below $a(l)$ depends solely on $z_{i+1}, \ldots, z_{n^*}$, and the assertion follows.

Figure 3 can be used to illustrate the way in which information must be gathered in order to compute the numbers z_i, $i = 1, \ldots, n^*$, as just described. Triangulation (i) in Figure 3 is the initial triangulation into which the line segment with endpoints P_1, P_2 is to be inserted, triangulation (vii) is the desired triangulation obtained from the execution of the modified version of INSERT_SEGMENT, and triangulations (ii)-(vi) are intermediate triangulations in the order in which they are computed during the execution of INSERT_SEGMENT. Each triangulation, except triangulation (i), corresponds to one instance of the swapping of edges, the dotted edge in each triangulation being the one that is eliminated. It is easy to see that for this example m equals 6, n^* equals 8, the values for $e(i)$, $i = 1, \ldots, 6$, are $(5,6)$, $(4,8)$, $(4,7)$, $(5,7)$, $(3,8)$, $(3,7)$, respectively, and for $a(i)$, $i = 1, \ldots, 6$ are $(1,7)$, $(3,7)$, $(3,5)$, $(1,3)$, $(2,7)$, $(1,2)$, respectively. In addition, from triangulations (ii), (iv)-(vii), respectively, the values for $g(i)$, $i = 1, \ldots, 5$, are obtained (the edge swapping depicted in triangulation (iii) does not produce a triangle that is not intersected by the line segment) and they are, respectively, the points P_6, P_4, P_5, P_8, P_7. Finally, we notice that for this example the values for $ini(i)$, $i = 1, \ldots, 5$, are, respectively, 1, 2, 4, 5, 6, and for $fin(i)$, $i = 1, \ldots, 5$, are, respectively, 1, 3, 4, 5, 6 (P_4, the value for $g(2)$, is the only point to which two instances of the swapping of edges are assigned: those depicted in triangulations (iii) and (iv)). Based on this information, the numbers z_i, $i = 1, \ldots, 8$, are then computed as described above.

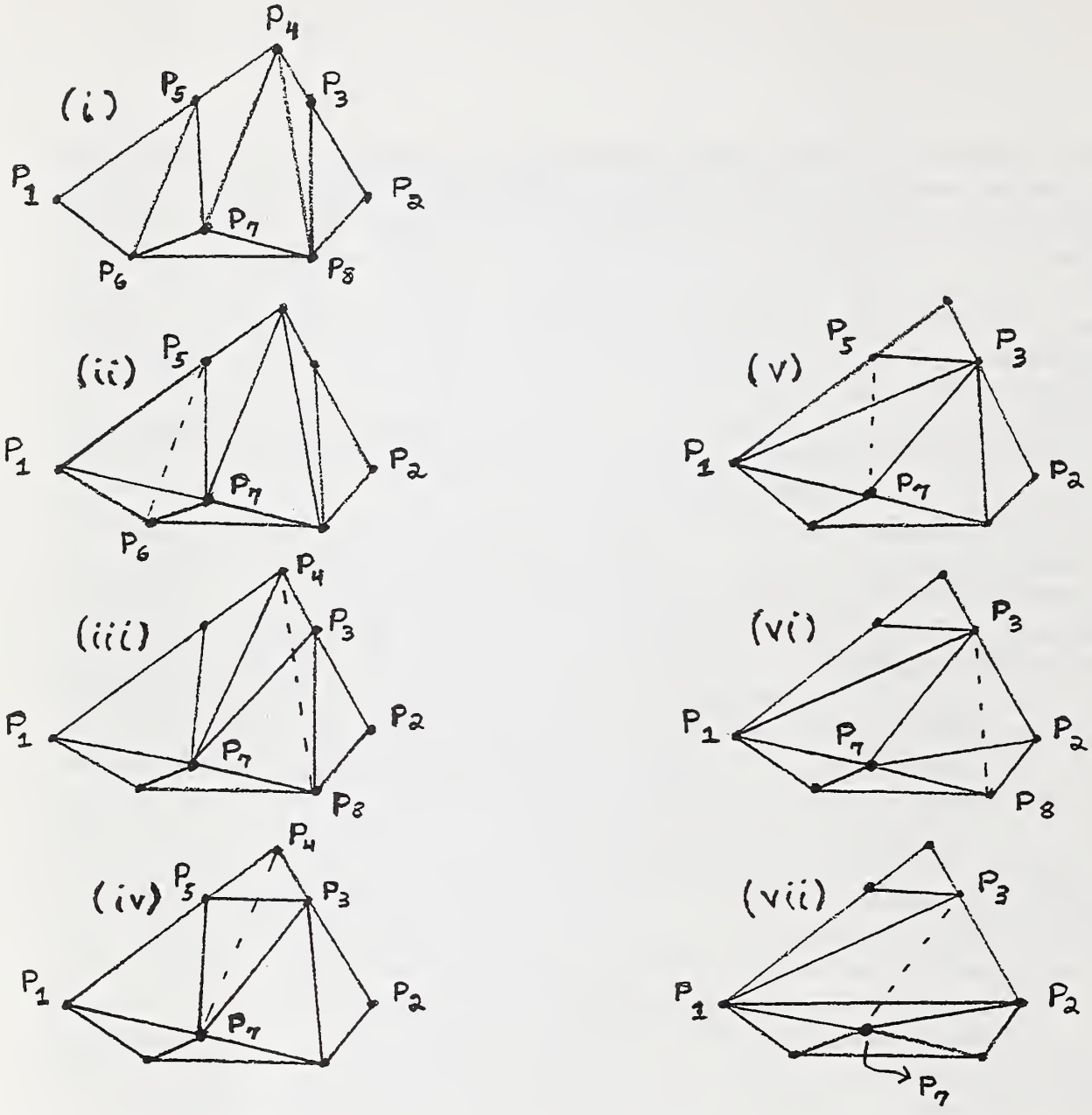

Figure 3: A triangulation and the swapping of edges during the insertion into it of line segment with endpoints P_1, P_2.

References

[1] J. Bernal, On constructing Delaunay triangulations for sets constrained by line segments, National Institute of Standards and Technology Technical Note 1252 (1988).

[2] J. Bernal, Computing Delaunay triangulations for comet-shaped polygons, National Institute of Standards and Technology Internal Report 4716 (1991).

[3] L. P. Chew, Constrained Delaunay triangulations, Algorihtmica 4 (1989), 97-108.

[4] L. De Floriani, B. Falcidieno, and C. Pienovi, Delaunay-based representation of surfaces defined over arbitrarily shaped domains, Computer Vision, Graphics, and Image Processing 32 (1985), 127-140.

[5] L. De Floriani and E. Puppo, Constrained Delaunay triangulation for multiresolution surface description, Proc. 9^{th} International Conference on Pattern Recognition (1988), 566-569.

[6] L. De Floriani and E. Puppo, A dynamic incremental algorithm for constrained Delaunay triangulation, Istituto per la Matematica Applicata Tech. Rep. (1988).

[7] C. L. Lawson, Transforming triangulations, Discrete Math. 3 (1972), 365-372.

[8] D. T. Lee and A. K. Lin, Generalized Delaunay triangulation for planar graphs, Discrete Comput. Geom. 1 (1986), 201-217.

[9] B. A. Lewis and J. S. Robinson, Triangulation of planar regions with applications, The Comput. J. 21 (1978), 324-332.

[10] F. P. Preparata, M. I. Shamos, Computational Geometry - An Introduction, Springer-Verlag, New York (1985).

[11] R. Seidel, Constrained Delaunay triangulation and Voronoi diagrams with obstacles, Rep. 260, IIG-TU Graz, Austria (1988), 178-191.

[12] C. A. Wang and L. Schubert, An optimal algorithm for constructing the Delaunay triangulation of a set of line segments, Proc. 3^{rd} Ann. ACM Symp. on Computational Geometry (1987), 223-232.