Algorithmica (1990) 5: 421-457

Algorithmica

Computational Geometry in a Curved World[1]

David P. Dobkin[2] and Diane L. Souvaine[3]

Abstract. We extend the results of straight-edged computational geometry into the curved world by defining a pair of new geometric objects, the *splinegon* and the *splinehedron*, as curved generalizations of the polygon and polyhedron. We identify three distinct techniques for extending polygon algorithms to splinegons: the carrier polygon approach, the bounding polygon approach, and the direct approach. By these methods, large groups of algorithms for polygons can be extended as a class to encompass these new objects. In general, if the original polygon algorithm has time complexity $O(f(n))$, the comparable splinegon algorithm has time complexity at worst $O(Kf(n))$ where K represents a constant number of calls to members of a set of primitive procedures on individual curved edges. These techniques also apply to splinehedra. In addition to presenting the general methods, we state and prove a series of specific theorems. Problem areas include convex hull computation, diameter computation, intersection detection and computation, kernel computation, monotonicity testing, and monotone decomposition, among others.

Key Words. Computational geometry, Splinegon, Curve algorithm, Convexity, Monotonicity, Intersection, Kernel, diameter decomposition.

1. Introduction—the Need for Algorithms on Curvilinear Objects. As the name of the field suggests, computational geometry concerns the algorithmic aspects of geometric problems. As such, the span of the field should include algorithms for reasonable objects definable in reasonable geometries. Until recently, the majority of the results obtained have been restricted to a small class of geometric objects: points, lines, line segments, polygons, planes, and polyhedra. Despite the extensive body of algorithms and algorithmic techniques for objects defined with straight edges and flat faces, few of its results apply directly to problems of the real world. Solid modeling systems build objects by patching together surface patches that are defined via bicubic splines or quadratic splines [Re]. Motion-planning problems that need to be solved for the advancement of robotics typically involve motion of curved objects through barriers having curved shapes [HK]. Modern font design systems rely upon conic and cubic spline curves [Pa], [Pr], [Kn]. Numerous applications need efficient algorithms for processing curved objects directly [Sm], [Fo].

Prior to the second author's doctoral dissertation [So], written under the supervision of the first author, few algorithms treated curved objects, other than

[1] This research was partially supported by National Science Foundation Grants MCS 83-03926, DCR85-05517, and CCR87-00917.

[2] Department of Computer Science, Princeton University, Princeton, NJ 08544, USA.

[3] Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, USA. This author's research was also partially supported by an Exxon Foundation Fellowship, by a Henry Rutgers Research Fellowship, and by National Science Foundation Grant CCR88-03549.

Received October 29, 1987; revised August 12, 1988. Communicated by Bernard Chazelle.

circles and spheres, directly [SV], [HMRT]. Instead, the way to tackle arbitrary real objects has been to approximate them first as polygons or polyhedra of a sufficient number of vertices for the particular application. This process is generally quite unsatisfactory [Sm], [Fo]. This paper presents the main core of the results from [So]. Two additional papers contain other aspects of that work, as well as some further research in the area [DSV], [DS2].

This paper contains a set of recipes that can be used to determine if a result in the linear convex world can apply in the curvilinear world and if so to determine the best method of translating the result. We supplement this with numerous applications to existing algorithms, demonstrating that curved objects can indeed be processed efficiently. Furthermore, as new algorithms are developed for straight objects, it will often be possible to state them for curved objects with no extra machinery. These contributions should aid both producers and consumers of geometric algorithms.

We begin by stating the definition of a new geometric object, the *splinegon*, which is general enough to describe almost every closed curve and structured enough to allow large groups of algorithms for polygons to be extended as a class to encompass these new objects. We identify three general methods for translating groups of algorithms for linear objects to algorithms for splinegons: the carrier polygon approach, the bounding polygon approach, and the direct approach. In general, if the original polygon algorithm has time complexity $O(f(n))$, the comparable splinegon algorithm has time complexity at worst $O(Kf(n))$ where K represents a constant number of calls to a series of primitive procedures on individual curved edges.

2. The Splinegon and Its Properties. The extension of algorithms designed for the world of straight-edged objects into the world of curved objects requires the definition of a new abstract object which can mediate between these two worlds. We call this new object a *splinegon*. First, we give a formal definition of the object as a curved extension of a straight-edged polygon, and then we describe the process of structuring an arbitrary curved object as a splinegon, choosing vertices which relate the splinegon to an inferred polygon. We also isolate the few curved objects which cannot be formulated as splinegons.

A *splinegon* S can be formed from a polygon P on n vertices, $v_1, v_2, \ldots, v_n$, by replacing each line segment $\overline{v_iv_{i+1}}$ with a curved edge e_i which also joins v_i and v_{i+1} and which satisfies the following condition: the region $S\text{-}seg_i$ bounded by the curve e_i and the line segment $\overline{v_iv_{i+1}}$ must be convex.[4] The new edge need not be smooth; a sufficient condition is that there exists a left-hand and a right-hand derivative at each point p on the splinegon. If $S\text{-}seg_i \subseteq S$, then we say that the edge e_i is *concave-in*. Otherwise, we say that the edge e_i is *concave-out*. The polygon P is called the *carrier polygon* of the *splinegon* S.

Splinegons can be categorized much as polygons are. If the only edge intersections are those between two adjacent edges at their common vertex, then the

[4] Subscripts are always interpreted modulo n.

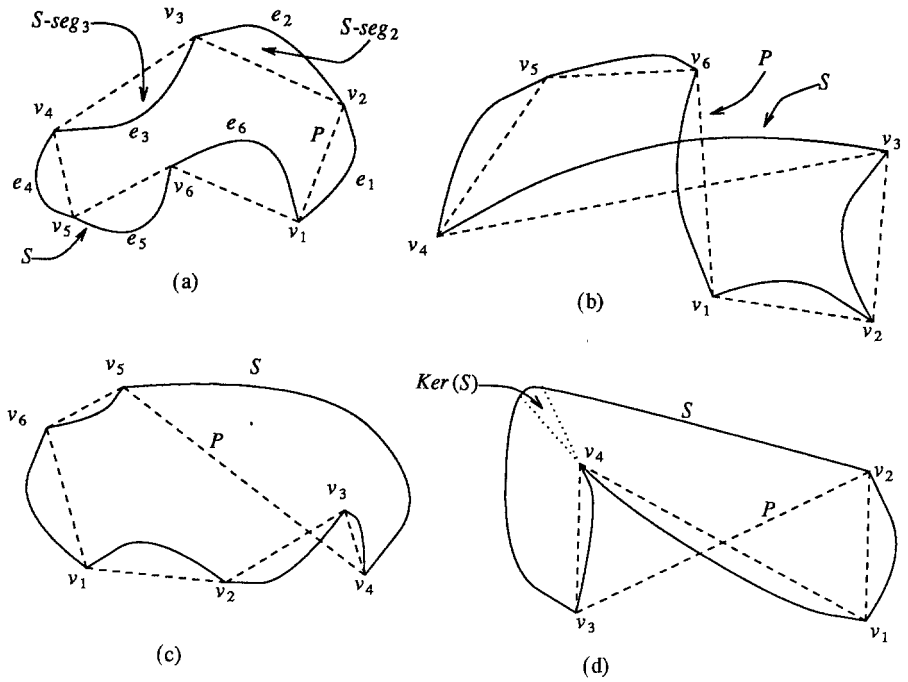

Fig. 1. (a) A simple splinegon with a simple carrier polygon; (b) a nonsimple splinegon; (c) a monotone splinegon in the x-direction with a nonsimple carrier polygon; and (d) a star-shaped splinegon and its kernel.

splinegon is said to be *simple.* If other edge intersections exist, then the splinegon is called *nonsimple.* A splinegon may be classified as a *monotone splinegon* in some distinguished direction $\vec{z}$ if it satisfies the following criterion: let m (resp. M) represent the point on the splinegon having the smallest (resp. largest) component in the $\vec{z}$ direction; the points m and M split the splinegon into two monotone chains of splinegon edges such that in traversing either chain from m to M the $\vec{z}$ component strictly increases. A *star-shaped splinegon* contains at least one point w in its interior so that each line segment from w to a point on the boundary of the splinegon lies within the splinegon. The collection of all such points w is called the *kernel* of the splinegon. The carrier polygons for splinegons in these four categories may or may not be simple (see Figure 1).

A *convex splinegon* S encloses a convex region. Clearly, any sequence of three or more points selected in order along the boundary of S defines a legitimate carrier polygon P. The convexity of S guarantees the convexity of P. We define a *triangle* to be a simple splinegon of three vertices. Since we have made no restriction that edges of splinegons be smooth, any arbitrary convex polygon of n vertices may be considered a splinegonal triangle. Although in the polygonal world a triangle is necessarily convex, a splinegonal triangle has no such restriction (see Figure 2).

We can now categorize the set of planar curves definable as splinegons:

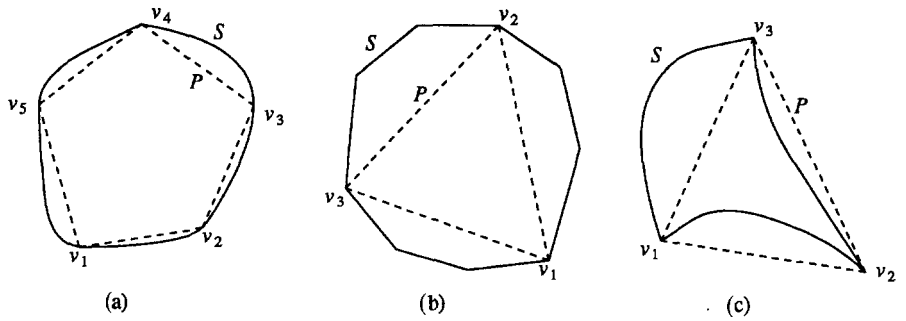

Fig. 2. (a) A convex splinegon, (b) a convex triangle, and (c) a nonconvex triangle.

THEOREM 1. *Any closed planar curve S can be considered a splinegon provided that the following two conditions are satisfied: S has only a finite number of inflection points; and any infinite line l intersects S in at most a finite number of points or line segments. If either condition is not met, no splinegon is possible.*

PROOF. To determine a carrier polygon for a curve S, we begin by tracing about the curve in counterclockwise order, inserting all inflection points as vertices of a tentative carrier polygon P. We now describe two methods for choosing additional vertices for the carrier polygon within each tentative e_i.

The first method requires less computation, but adds more vertices than necessary and yields an unwieldy carrier polygon, with overlapping, collinear edges. Trace S in counterclockwise order, moving from vertex to vertex. At every edge e_i, add as a vertex of P each single point and the endpoints of any line segments of the intersection of the line $\overleftrightarrow{v_i v_{i+1}}$ with e_i. Add all new vertices to P in the order in which they are encountered on tracing e_i from v_i to v_{i+1} (see Figure 3(a)).

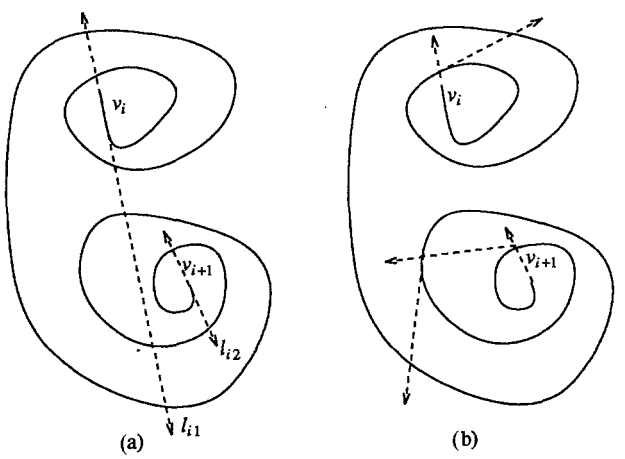

Fig. 3. An edge e_i of S when its carrier polygon P contains only inflection points. (a) The vertices added by method 1. (b) The vertices added by method 2.

The second method requires more computation, but adds fewer vertices and yields a more manageable carrier polygon. Let $\overleftrightarrow{l_{i1}}$ (resp. $\overleftrightarrow{l_{i2}}$) represent the line tangent to e_i at v_i (resp. v_{i+1}). Trace S in counterclockwise order, moving from vertex to vertex. Stop at each edge e_i. If the line $\overleftrightarrow{l_{i1}}$ intersects e_i in a single point (rather than a line segment) other than v_i, insert the first such point encountered on tracing e_i from v_i to v_{i+1} into the vertex list for P, making it the next vertex to be visited and splitting the current edge into two pieces. Repeat this process, tracing the curve in clockwise order and testing each edge e_i for intersection with $\overleftrightarrow{l_{i2}}$ (see Figure 3(b)).

Both methods terminate, provided that there are a finite number of inflection points and intersection points. If either condition is not met, no splinegon can be produced. □

It is possible to define planar curved objects which are not splinegons. For example, the following three segments define a closed curve which contains an infinite number of inflection points, all lying on the *x-axis*:

$$y = x \sin(1/x) \quad \text{for} \quad x \in (0, 2/\pi];$$
$$y = 2/\pi \quad \text{for} \quad x \in (0, 2/\pi];$$
$$x = 0 \quad \text{for} \quad y \in [0, 2/\pi].$$

The following four segments define a closed curve which intersects any line through the origin in an infinite number of distinct points:

$$r = 1/\theta \quad \text{for} \quad \theta \in [2\pi, +\infty);$$
$$r = 1/(\theta + \pi) \quad \text{for} \quad \theta \in [2\pi, +\infty);$$
$$\theta = 0 \quad \text{for} \quad r \in [1/3\pi, 1/2\pi];$$
$$(r, \theta) = (0, 0).$$

Neither of these two objects, however, arises in practice. Real-world planar curved objects can be constituted as splinegons.

For a closed planar curve S, either method of determining a carrier polygon produces a splinegon with far fewer vertices, in general, than any polygon which would adequately approximate the curve. Primitive procedures on splinegons, however, are more complicated than those on polygons. Determining the intersection of two line segments, for example, is a well-understood, and often implemented, process requiring constant time.[5] The complexity of determining the intersection of two curved segments depends on the complexity of the curves themselves. Consequently, our analyses of algorithm complexity are given both in terms of operations and calls to oracles for primitive procedures. We include here three-dimensional versions of the oracles which will be used in Section 7.

[5] As Forrest [Fo] and others remind us, however, in a world where round-off error exists, even line segment intersection is not a solved problem. We ignore such questions here. The interested reader is referred to [DSi].

A_1 (A_2) Compute the intersection of two curved edges (faces) or the maximum and minimum separation between them.
B_1 (B_2) Compute the intersection of a line with a curved edge (face).
C_1 (C_2) Given a curved edge (face) and either a direction or a point, report both the point and the direction of a line (plane) that supports the edge (face) at that point.
D_1 (D_2) Determine the line (plane) which supports a pair of curved edges (faces).
E_1 (E_2) Compute the intersection of a plane with a curved edge (face).
F_1 (F_2) Compute the area (volume) bounded by a curved edge (face) and by the corresponding edge (face) of the carrier polygon (polyhedron).

The complexity of each primitive procedure would be a constant in any domain where we restricted splinegon edges.

3. The Carrier Polygon Approach. The carrier polygon approach is particularly useful for processing convex splinegons. Algorithms for convex polygons often exploit a pair of key properties:

(a) Given any point x on the boundary of a planar convex object S, there exists a *supporting line* l through x which divides the plane into two half-planes: a closed half-plane containing all of S; and an open half-plane containing no point of S. If S is a polygon of n vertices, then the set of n lines formed by extending each of the n edges of S is sufficient to supply a supporting line for S at each point on its boundary.
(b) Given any direction in the plane, there exists a pair of lines m_1 and m_2 in that direction such that both m_1 and m_2 are *supporting lines* for S and the region bounded by m_1 and m_2 completely contains S. If S is a polygon of n vertices, then the intersection of each of m_1 and m_2 with S must include at least one of the n vertices.

Direct extension of these algorithms to splinegons requires an excessive number of operations on curved edges. Indeed, an uncountable number of lines are required to provide a line of support at each boundary point of a convex splinegon. Thus, in applying property (a), the line of support at a given point x would have to be computed individually. Likewise, an uncountable number of points are necessary to provide a boundary point of the splinegon lying on each possible supporting line. Thus, in applying property (b) to splinegons, finding a point of S lying on each of m_1 and m_2 would require calculations using curved edges.

Focusing on the carrier polygon avoids many of the direct manipulations of the curved edges. For each convex splinegon S, the carrier polygon P is itself convex, $P \subseteq S$ and $S = P \cup (\bigcup_i S\text{-}seg_i)$. The line defined by two adjacent vertices, $\overleftrightarrow{v_i v_{i+1}}$, divides the plane into two half-planes: the "outside" half-plane contains the convex region $S\text{-}seg_i$; the "inside" half-plane contains a splinegon $S_i = S - S\text{-}seg_i$ which is supported by $\overleftrightarrow{v_i v_{i+1}}$ along an edge. Each $S\text{-}seg_i$ lies within a

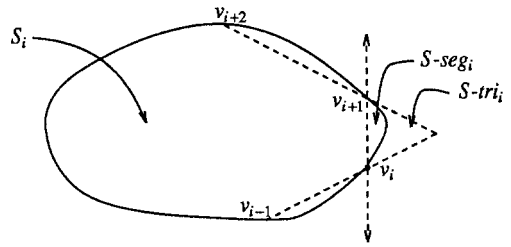

Fig. 4. A convex splinegon S divided into S_i and $S\text{-}seg_i$.

triangle $S\text{-}tri_i$ defined by $\overline{v_iv_{i+1}}$, $\overleftrightarrow{v_{i-1}v_i}$ and $\overleftrightarrow{v_{i+1}v_{i+2}}$ (see Figure 4).[6] $P \cup (\bigcup_i S\text{-}tri_i)$ forms a star-shaped polygon whose kernel is P.

OBSERVATION 1 (Carrier Polygons). The carrier polygon imposes sufficient structure on a convex splinegon that polygon algorithms can be extended to splinegons with the only modification being *ad hoc* procedures to allow for all possible behavior of $S\text{-}seg_i$ and its bounding triangle $S\text{-}tri_i$. Only infrequently will an examination of the precise behavior of e_i be necessary. Although the carrier polygon has particular use with convex polygons, if it is simple, then it also aids in processing monotone splinegons.

This observation enables us to adapt the algorithms of [CD] and [DK1] to process splinegons as well as to develop an original algorithm for point inclusion both for convex polygons and for convex splinegons.

THEOREM 2. *The intersection of a line with a convex splinegon of N vertices can be computed in* $O(B_1+\log N)$ *operations* (*see Figure* 5).

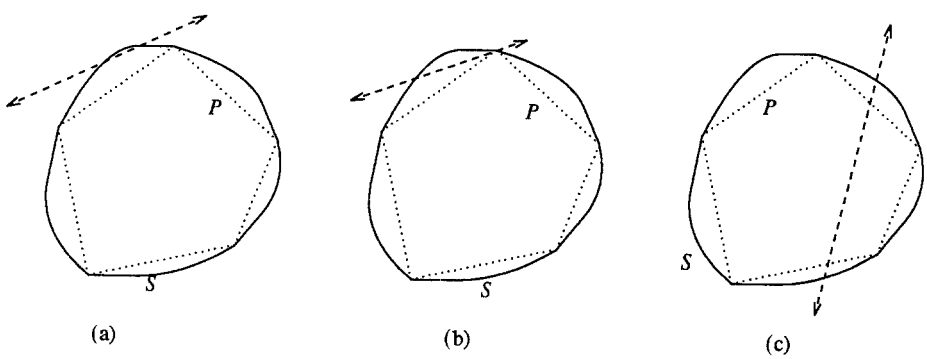

Fig. 5. Examples of Theorem 2.

[6] Note that $S\text{-}tri_i$ may be unbounded.

PROOF. To compute the intersection of a line l with a convex splinegon S, first run the Chazelle-Dobkin polygon algorithm [CD] on the carrier polygon P. If there is no intersection, that algorithm will report the vertex v_i of the polygon which is closest to l. Although l does not intersect P, it may still intersect one of $S\text{-}seg_{i-1}$ or $S\text{-}seg_i$. (The convexity of S dictates that l cannot intersect both.) Consequently, we use oracle B_1 to test l against e_{i-1} and e_i. In the case that the intersection of l with P consists of a single vertex v_i, again l may also intersect either $S\text{-}seg_{i-1}$ or $S\text{-}seg_i$. Testing the line against both e_{i-1} and e_i is sufficient to determine the splinegon-line intersection. If the polygon algorithm reports the intersection as $\overline{v_iv_{i+1}}$, an entire edge of P, or as $\overline{v_iv_j}$, a diagonal of P, then the intersection of l and S also consists exactly of that segment. If the polygon algorithm reports the intersection as a line segment with endpoints on two different edges of the carrier polygon, $\overline{v_iv_{i+1}}$ and $\overline{v_jv_{j+1}}$, testing the line against the corresponding curved edges e_i and e_j determines the endpoints of the segment forming the splinegon-line intersection. After running the polygon algorithm in $O(\log N)$ time, the subsequent special cases each require at most two calls to oracle B_1. Thus the entire process requires $O(B_1+\log N)$ time, using asymptotically fewer oracle calls than simple operations. □

Chazelle and Dobkin [CD] show that the inclusion of a point in a convex polygon of N vertices can be decided in $O(\log N)$ operations by their algorithm for computing the intersection of a line with a convex polygon. In recent years, however, the hierarchical searching method has emerged as a potent tool in the field of computational geometry [Ki], [DK2], [DS1]. The following dichotomy makes the convex splinegon an abstract object ideally suited for hierarchical processing. An algorithm for an N-sided polygon requires many iterations when N is large, but primitive operations on edges require only constant time. If an N-sided convex polygon is viewed as a splinegonal triangle, primitive operations on the "edges" require time proportional to N, but each algorithm has a constant number of iterations.

THEOREM 3. *The inclusion of a point in a convex polygon of N vertices can be decided in $O(\log N)$ operations using the hierarchical method.*

PROOF. Various proofs of this theorem exist (see, e.g., pp. 85-86 of [Me]). We include the proof here for completeness.

Given a convex polygon P on N vertices, $v_1, v_2, \ldots, v_N$, we develop a hierarchy of splinegons all having the same boundary as P, but having carrier polygons of fewer vertices. We assume for ease of explanation that $N=3(2^k)$ for some k. We recast P as a triangular splinegon S^k with a carrier polygon P^k whose three vertices v_1^k, v_2^k, v_3^k are all original vertices of P. The vertices v_i^k are chosen so that $N/3-1$ of P's original vertices lie in the interior of each of the three "curved" edges of S^k.

To create a splinegon S^{i-1} from a splinegon S^i in the hierarchy, insert each vertex v_j^i of S^i as a vertex of S^{i-1}. In addition, insert the median original vertex

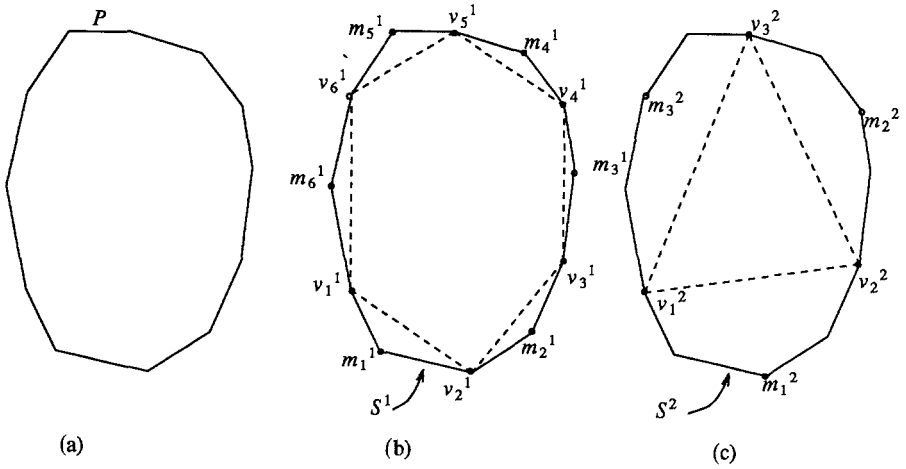

Fig. 6. (a) A sample polygon P on 12 vertices; (b) a splinegon S^1; and (c) a splinegon S^2.

m_j^i lying in the interior of each edge e_j^i of S^i. After performing this process k times, we achieve a splinegon S^0 which is identical to its carrier polygon P^0 and to the original polygon P (see Figure 6).

To test whether a point x lies within the convex polygon P, we begin by determining in constant time whether x lies within the straight-line triangle P^k. If it does, we are done. If not, then x must lie outside of at least one of the lines $\overleftrightarrow{v_j^k v_{j+1}^k}$. If x lies outside of two of the lines, then by convexity it cannot belong to P, as we discussed above. Suppose that x lies outside of the line $\overleftrightarrow{v_j^k v_{j+1}^k}$. Then we can determine whether x lies in the carrier polygon P^{k-1} by testing whether x belongs to the triangle $\Delta v_j^k m_j^k v_{j+1}^k$, which is identical to the triangle $\Delta v_{2j}^{k-1} v_{2j+1}^{k-1} v_{2j+2}^{k-1}$ defined by three adjacent vertices of P^{k-1}.

At each stage in the algorithm, we either determine that x belongs to some carrier polygon P^i, or that x lies outside of P, or select a triangular test which will determine whether x belongs to P^{i-1}. If we proceed through k stages without terminating, we know an index j such that x belongs to the original polygon P if and only if x belongs to the triangle $\Delta v_j v_{j+1} v_{j+2}$. At each stage in the algorithm, the point is tested for inclusion in a triangle. The first such test reduces the size of the problem by at least two-thirds. Each subsequent test divides the problem in half. □

COROLLARY. *The inclusion of a point in a convex splinegon of N vertices can be decided in $O(B_1+\log N)$ operations using the hierarchical method.*

PROOF. Given a convex splinegon S has P as its carrier polygon, define a hierarchy of splinegons all having the same boundary as S, but with vertices chosen as described above. In this case, the lowest-level splinegon S^0 is identically equal to S, and its carrier polygon P^0 is exactly P. At the final stage of the algorithm, we know an index j such that x belongs to the original carrier polygon

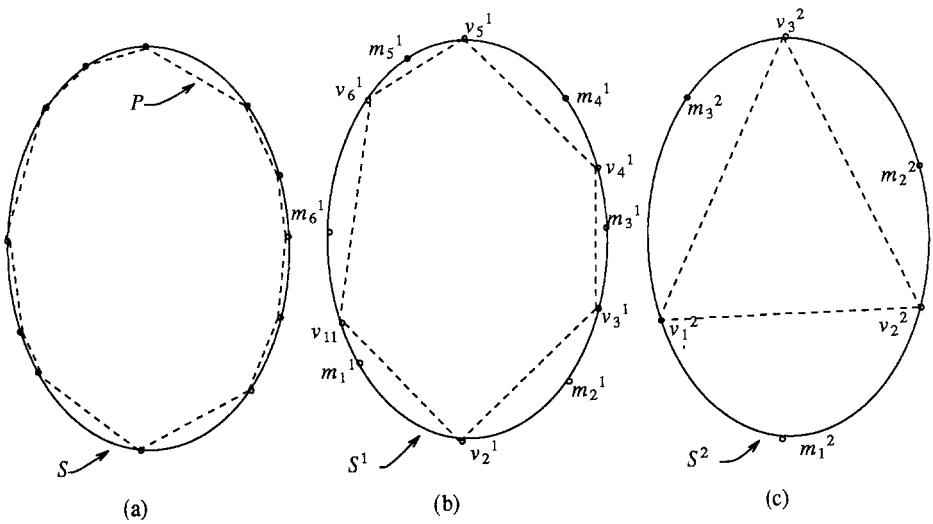

Fig. 7. (a) A splinegon S on 12 vertices; (b) S^1; and (c) S^2.

P if and only if x belongs to the triangle $\Delta v_j v_{j+1} v_{j+2}$. If x lies outside both $\overleftrightarrow{v_j v_{j+1}}$ and $\overleftrightarrow{v_{j+1} v_{j+2}}$, then x does not belong to S. If x lies outside just $\overleftrightarrow{v_j v_{j+1}}$, then x belongs to S if and only if it belongs to $S\text{-}seg_j$. Similarly, if x lies outside just $\overleftrightarrow{v_{j+1} v_{j+2}}$, then x belongs to S if and only if it belongs to $S\text{-}seg_{j+1}$ (see Figure 7). □

THEOREM 4. *The intersection of two convex splinegons of at most N vertices can be detected in $O(A_1 + B_1 + C_1 \log N)$ operations.*

PROOF. Given the problem of detecting the intersection of two convex splinegons P and Q of at most N vertices each, we assume that each splinegon is given as a list of vertices in a random access memory with each vertex pointing to a description of the curve which adjoins it to its neighbor. Otherwise, no sublinear algorithm is possible [CD]. We do no preprocessing of this description. The output should consist either of a point in the intersection or of a line which supports one of the splinegons and separates it from the other.

Our algorithm employs the binary search strategy used in the polygonal algorithm due to Dobkin and Kirkpatrick [DK1]. First, we split P (resp. Q) at the points having maximum and minimum y-coordinate into a left semi-infinite splinegon P_L (resp. Q_L) and a right semi-infinite splinegon P_R (resp. Q_R). The splinegons P and Q intersect if and only if P_L intersects Q_R and P_R also intersects Q_L. Thus, we need an algorithm for detecting the intersection of a left semi-infinite splinegon L with a right semi-infinite splinegon R. If L (resp. R) has n (resp. m) vertices $l_1, l_2, \ldots, l_n$ (resp. $r_1, r_2, \ldots, r_m$), let $i = \lceil m/2 \rceil$ and $j = \lceil n/2 \rceil$. R_i (resp. L_j) represents the line defined by the pair of vertices r_i, r_{i+1} (resp. l_j, l_{j+1}). An examination of the relative positioning of r_i, r_{i+1}, l_j, l_{j+1}, R_i and L_j allows the removal of half of the vertices of at least one splinegon from further consideration. After $O(\log n)$ iterations, one splinegon is reduced to three vertices. Split

that splinegon into two splinegons of two vertices each and compare each in turn against the second splinegon using the original strategy. In some instances, it will be necessary to add a single curved operation to guarantee the removal of half of the vertices of the larger splinegon. Nonetheless, in each case, $O(\log n)$ iterations will reduce the second splinegon to three vertices, and then any intersection can be detected using brute force. For details, see [So]. A sequel to this paper includes further details and complete implementation [DS2]. □

4. Bounding Polygon Approach. In standard form, a splinegon S is given as a circular list of N vertices, which completely defines its carrier polygon P, and a pointer from each vertex to a description of the edge joining that vertex to its neighbor. We now define a second polygon Q of $O(N)$ vertices called the *bounding polygon* of the *splinegon S*. Q contains all of the vertices of S, which are called *fixed vertices* of S. In addition, for each edge e_i of S which is not a line segment, the vertex list for Q contains a *pseudovertex* e_i^* of S between the fixed vertices v_i and v_{i+1}; e_i^* represents the point of intersection of the ray tangent to e_i at v_i with the ray tangent to e_i at v_{i+1}.[7] An edge of Q which joins two fixed vertices is called a *fixed edge*. An edge joining a fixed vertex with a pseudovertex is called a *pseudoedge*. A pseudoedge $\overline{v_i e_i^*}$ (resp. $\overline{e_i^* v_{i+1}}$) is considered *loose* if its only intersection with the curved edge e_i is at the vertex v_i (resp. v_{i+1}). If $\overline{v_i e_i^*}$ (resp. $\overline{e_i^* v_{i+1}}$) intersects e_i in a line segment, then the edge is considered *tight*.

The bounding polygon provides a useful tool for extending numerous algorithms on simple polygons to splinegons, particularly those algorithms which are vertex-based. The efficacy of the bounding polygon is due in large part to the fact that for many calculations on polygons, or on splinegons, attention need be given only to vertices at reflex angles, interior angles measuring more than 180°. In general, an edge e_i of S can be viewed as a polygonal chain of an arbitrary number of vertices leading from v_i to v_{i+1} whose first edge lies (either loosely or tightly) on $\overline{v_i e_i^*}$, whose last edge lies (either loosely or tightly) on $\overline{e_i^* v_{i+1}}$. A splinegon whose edges are all concave-in has reflex angles only at fixed vertices, and the adjacent edges of the bounding polygon accurately determine the angle at a fixed vertex. A curved edge which is concave-out corresponds to a polygonal chain composed entirely of reflex angles. The associated pseudovertex and the adjacent pseudoedges give a good approximation. During the execution of a polygon algorithm on the bounding polygon, processing of a pseudovertex e_i^* can include evaluation and insertion of pertinent information about the associated curve e_i.

[7] For some splinegon edges the two tangent rays might not intersect in the plane. The rays would intersect, however, if S were embedded on the surface of a sphere. We can allow the point e_i^* to represent the corresponding point on the projective plane at infinity (see Figure 8(a)). Alternately, such a splinegon edge can be broken into at most three pieces by the insertion of two new fixed vertices so that for each new edge the tangent rays will intersect at a point with finite coordinates (see Figure 8(b)). The bounding polygon Q of a simple splinegon S of N vertices has at most $2N$ vertices if we allow points at infinity, or at most $6N$ vertices otherwise. For the sake of clarity, we assume that all pseudovertices have finite coordinates.

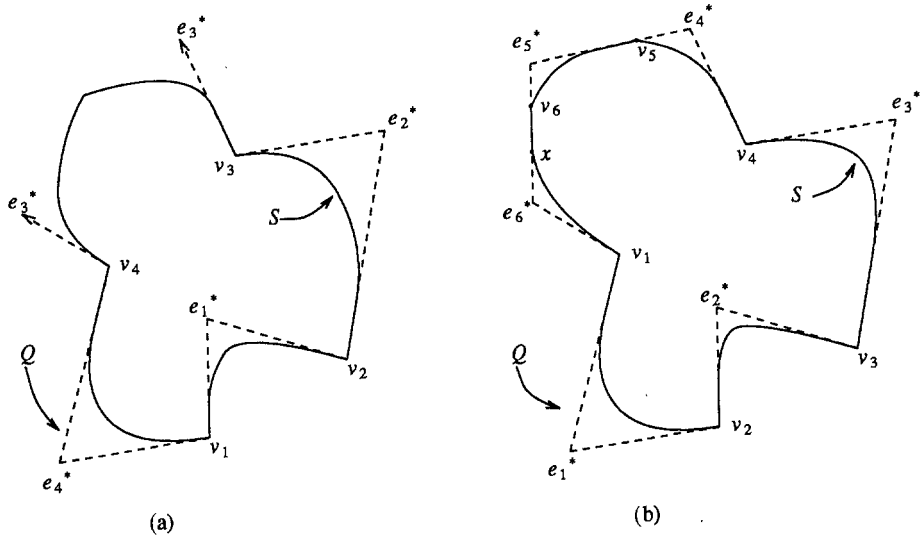

Fig. 8. A pair of splinegons and their bounding polygons: (a) has an infinite pseudovertex, whereas (b) does not. In (b), $v_6 e_6^*$ is a *tight* pseudoedge since it contains the line segment $\overline{v_6 x}$; all other pseudoedges are loose.

The splinegon shares several characteristic properties with its bounding polygon, as we shall prove below. A convex splinegon S with carrier polygon P is contained within its convex bounding polygon Q. (Remember that the polygon defined by $P \cup (\bigcup_i S\text{-}tri_i)$ is star-shaped rather than convex.) A splinegon S is monotone if its bounding polygon Q is monotone. Moreover, if S is monotone in a direction $\vec{z}$ for which Q is not, then $\vec{z}$ must be orthogonal to some loose pseudoedge of Q. A splinegon has a kernel only if the bounding polygon has a kernel.

Simplicity, however, is not a shared property. The bounding polygon of a simple polygon need not be simple. Moreover, a nonsimple splinegon may have a simple bounding polygon. Nonetheless, algorithms on simple polygons may extend to simple splinegons using the bounding polygon technique, whether or not the bounding polygon is simple. For example, as demonstrated below, we may compute the convex hull of a simple splinegon using a bounding polygon which, if computed explicitly, would not be simple.

Computing the bounding polygon explicitly often provides a useful approximation for the splinegon. Unfortunately, the specifications of the polygon Q are not readily available from the given description of the splinegon S. Computing each edge of Q costs $O(C_1)$ time. As a consequence, we postpone computing the actual coordinates of a pseudovertex until necessary. In some instances, it may be possible to compute the coordinates of only a small subset of the pseudovertices. In the convex hull algorithm given below, none of the coordinates for any e_i^* are ever computed. Instead the entry of e_i^* merely points to a description of the edge e_i. In such an instance, we are using the polygon in a topological sense, rather than a strict one [Km].

The bounding polygon provides an alternative tool for expressing the splinegon itself. It allows a vertex-based polygon algorithm to be extended to splinegons with its structure intact. It also allows the creators of new algorithms to write them in a general format which encompasses splinegons, but with the more restricted polygon algorithm clearly visible within. In either case, separate procedures exist for the processing of fixed vertices and for the processing of pseudovertices.

To summarize:

OBSERVATION 2 (Bounding Polygons). When the carrier polygon itself is insufficient, it may often be enhanced by adding further structure based solely on the local structure of the splinegon. This structure results in a bounding polygon with *fixed vertices* corresponding to vertices of the splinegon and *pseudovertices* corresponding to edges of the splinegon. Insofar as possible, computations involving the polygon in the linear case now involve the bounding polygon with only local attention to the splinegon's curved edges.

LEMMA 1. *A convex splinegon has a convex bounding polygon.*

PROOF. Suppose a given convex splinegon S has a nonconvex bounding polygon Q. Then there exists a reflex angle either at a fixed vertex v_i or at a pseudovertex e_i^*. But all edges of a convex splinegon are concave-in, and thus all pseudovertices are convex. Suppose that Q has a reflex angle at v_i. But then there exist points x_i and x_{i-1} lying on e_i and e_{i-1}, respectively, which are each arbitrarily close to v_i such that the interior angle $\angle x_{i-1}v_ix_i$ is also reflex. But then S is not convex. □

THEOREM 5. *The diameter of a convex splinegon of N vertices can be computed in $O((A_1+C_1)N)$ time.*

PROOF. The diameter of a convex polygon is realized by a pair of antipodal vertices. Shamos's algorithm for finding the diameter [Sh] determines all such vertex pairs, computes all of the distances, and keeps the maximum. The diameter of a convex splinegon is also realized by a pair of antipodal points, but although those points will lie on the boundary of the splinegon, they may not be vertices. To find the diameter of a convex splinegon S of N vertices, we apply a modified version of the Shamos algorithm to the bounding polygon, which by Lemma 1 is convex and contains S. After determining all antipodal vertex pairs for the bounding polygon, any pseudovertex can be replaced by the appropriate point on its corresponding edge to yield the pairs of antipodal points on the splinegon itself.

To run this algorithm, the entire bounding polygon must be computed, using $O(C_1N)$ time. Next, each edge of the bounding polygon is oriented as a vector and translated in turn to the origin: pseudoedges become vectors of the form $\overrightarrow{v_ie_i^*}$ and $\overrightarrow{e_i^*v_{i+1}}$; and fixed edges become vectors of the form $\overrightarrow{v_iv_{i+1}}$. Due to the

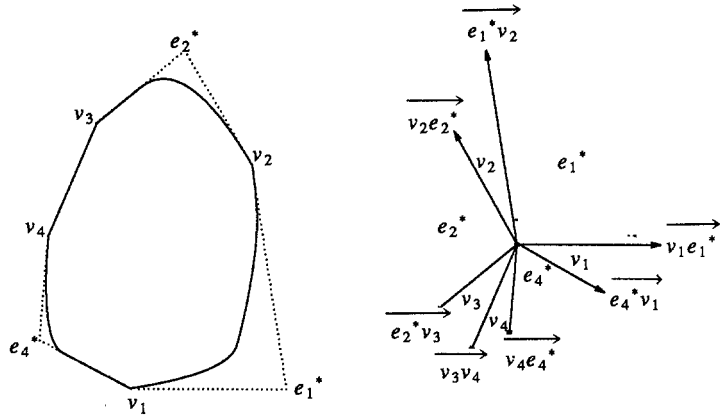

Fig. 9. A splinegon and its bounding polygon and the corresponding vertex sectors.

convexity of S, these vectors will be in nondecreasing order of polar angle.[8] The inclusive sector from a vector whose head is a fixed vertex v_i to a vector whose tail is v_i corresponds to that fixed vertex. If the two vectors have the same direction, then the fixed vertex will correspond to a sector consisting of a single ray. The sector strictly between a vector whose head is a pseudovertex e_i^* and a vector whose tail is e_i^* corresponds to that pseudovertex. A pseudovertex e_i^* was added to the bounding polygon only if the edge e_i was not straight. Thus, each pseudovertex corresponds to a nonempty open sector (see Figure 9).

Pick a line l passing through the origin, and in time $O(\log N)$ determine the two sectors in which it lies. To find all $O(N)$ antipodal pairs, rotate the line l counterclockwise. An antipodal pair changes only when l enters a new sector. Divide the vertex pairs reported into three groups: pairs having two fixed vertices; pairs with one fixed vertex and one pseudovertex; and pairs having two pseudovertices. The first group can be processed as in the original algorithm by computing the distance between the two vertices. The next two groups require the use of oracle A_1. For the second group, determine the point on that curved edge associated with the pseudovertex which lies at maximum distance from the fixed vertex. For the last group, determine the pair of points, one per curved edge, at maximum distance from each other. It is easy to see that the maximum distance over the three groups is the diameter. Thus, the entire algorithm runs in $O((A_1+C_1)N)$ time. □

LEMMA 2. *If all edges of the bounding polygon Q associated with a splinegon S are tight, then S is monotone in a direction $\vec{z}$ if and only if Q is monotone in the same direction. If S is monotone in a direction $\vec{z}$ for which Q is not, then $\vec{z}$ is orthogonal to some loose pseudoedge of Q.*

[8] If the boundary of S is smooth at a vertex v_i, then the vectors $\overrightarrow{e_{i-1}^* v_i}$ and $\overrightarrow{v_i e_i^*}$ have the same orientation.

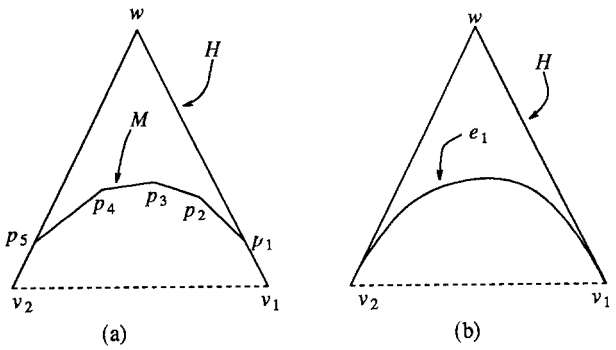

Fig. 10. (a) A convex polygonal chain inscribed in a triangle; and (b) a curved edge inscribed in a triangle.

PROOF. A splinegon, like a polygon, is monotone in a direction $\vec{z}$ if it can be decomposed into two chains monotone in directions $\vec{z}$. A monotone chain can be traversed from one endpoint to the other with the $\vec{z}$ component strictly increasing. Thus, a straight edge or line is monotone in every direction except its orthogonal. A chain consisting of two edges is monotone in every direction which has an orthogonal lying in the interior of the convex angle defined by the two edges. But to determine if a polygonal chain is monotone, it is not necessary to examine every edge. Suppose that the points $v_1, p_1, p_2, \ldots, p_n, v_2$ form a convex polygonal chain M and that the points v_1, w, v_2 form a triangle such that p_1 lies on the line segment $\overline{v_1 w}$ and p_n lies on $\overline{v_2 w}$. Then the chain M and the chain H defined by v_1, w, v_2 are both monotone in the direction $\vec{z}$ if and only if a ray originating at w and orthogonal to $\vec{z}$ lies in the interior of $\angle v_1 w v_2$ (see Figure 10(a)). Similarly, a splinegon edge e_1 extending from v_1 to v_2 for which $\overline{v_1 w}$ and $\overline{v_2 w}$ are tight pseudoedges is monotone in a direction $\vec{z}$ if and only if H is monotone. If the pseudoedge $\overline{v_1 w}$ (resp. $\overline{v_2 w}$) is loose, however, then in general the monotonicity of e_1 coincides with the monotonicity of H. But e_1 is monotone in the direction orthogonal to $\overline{v_1 w}$ (resp. $\overline{v_2 w}$), and H is not (see Figure 10(b)). □

THEOREM 6. *The directions in which a simple splinegon of N vertices is monotone can all be determined in $O(C_1 N)$ time.*

PROOF. According to Lemma 2, the monotonicity of a simple splinegon can be determined by deciding the monotonicity of the bounding polygon and then paying special attention to the loose pseudoedges. The Preparata-Supowit algorithm [PS] for the polygonal case uses an approach similar to the Shamos diameter algorithm [Sh]. Each edge of the polygon is oriented as a vector, translated in turn to the origin. By noting whenever a sector has been swept over more than once in moving from one vector to the next, they isolate the directions of monotonicity. We use the Preparata-Supowit [PS] method, but integrate the special procedures for loose pseudoedges into the main algorithm by making the following modification. Instead of processing all of the directions from $\overrightarrow{v_i e_i^*}$ to

$\overrightarrow{e_i^* v_{i+1}}$, inclusive, as a group, process the directions between the two vectors as a group and process the vectors $\overrightarrow{v_i e_i^*}$ and $\overrightarrow{e_i^* v_{i+1}}$ separately according to whether the corresponding edges were tight or loose. Orient all edges of the bounding polygon as vectors. Assign each fixed edge a key of 1. If pseudoedge $\overrightarrow{v_i e_i^*}$ (resp. $\overrightarrow{e_i^* v_{i+1}}$) is tight, associate with it a key of 1; otherwise, assign it a key of 0. We calculate the edges of the bounding polygon, one by one, and enqueue the vectors as translated to the origin, onto the queue L. After enqueuing all of the vectors, push a second copy of the first. Creating the queue L costs $O(C_1 N)$ time.

We process the list of at most $2N+1$ vectors one by one, retaining the significant information in a new list M of vectors ordered by polar angle. Pop the first vector from L and insert it into the empty list M along with its key and with three tags all initialized to 0: the forward, the backward, and the self. These tags may assume values in the set $\{0, 1, 2\}$. If a tag having a value of 2 is "incremented," it retains the value 2. The last vector inserted into M is called the current vector.

If the top vector of L describes the same polar angle as does the current vector, compare their keys. If both vectors have the same key, delete the top vector. Otherwise keep as the current vector whichever one has the larger key and delete the other. Consider the angle from the current vector to the top vector. If it belongs to the interval $(0, \pi)$ (resp. $(-\pi, 0)$), we shall move forward (resp. backward) through M in order to insert the top vector. Begin by incrementing the forward (resp. backward) tag on the current vector. Increment the self tag only if the current vector has a key of 1 or if this move does not represent a change in direction. Then move forward (resp. backward) through M, incrementing all three tags on every vector and deleting any vector having three identical tags, until locating the position for the new vector. Pop it from L and insert it into M, making the backward (forward) tag match the forward (backward) tag of the vector preceding it, and the forward (backward) and self tags match the backward (forward) tag of the vector ahead of it (see Figure 11).

Each vector is inserted into M once, requiring constant time. Each subsequent time it is processed, all three of its tags are incremented using constant time. But when all three tags on a vector equal 2, the vector is deleted. As each of the

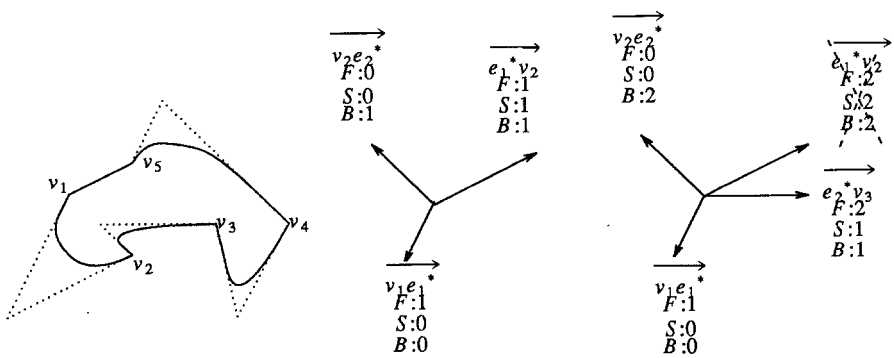

Fig. 11. (a) A splinegon S and its bounding polygon Q. (b) M after three insertions. (c) M after four insertions.

$O(N)$ vectors will be processed at most three times, the cost of creating the list M is at most $O(N)$.

At the termination of the above algorithm, we have the partition of the polar range $[0, 2\pi)$ by $O(N)$ vectors, each labeled with either a 1 or a 2, into $O(N)$ sectors which can be identified as a 1 or a 2 by the forward and backward tags on the vectors bounding it. Pick a pair of rays r_1, r_2 which form a straight angle at the origin. In $O(\log N)$ time, determine whether r_1 (r_2) contains a vector in M or lies in a sector between two vectors and assign it the appropriate label. Rotate r_1 and r_2 in tandem counterclockwise around the origin, changing the labels whenever either intersects a new vector or enters a new sector and recording every polar-angle interval in which both rays are assigned a 1. Since the labels change at most $O(N)$ times, there are at most $O(N)$ intervals reported. Thus this process requires $O(N)$ time. Whenever both rays are assigned a 1, T is monotonic in the direction normal to the two rays. If there is no angle at which both rays are assigned a 1, then T is not monotone. □

LEMMA 3. *The kernel of a splinegon S is equal to the intersection of S and the visible regions defined by the edges of its bounding polygon Q.*

PROOF. For a point within a splinegon to be visible from an edge e_i which is concave-out, the point must lie within the wedge at the pseudovertex e_i^* defined by the extension of the rays $\overrightarrow{v_i e_i^*}$ and $\overrightarrow{v_{i+1} e_i^*}$ (see Figure 12(a)). Thus, a concave-out curved edge from v_i to v_{i+1} defines the same visible region as would the pair of straight pseudoedges $\overline{v_i e_i^*}$ and $\overline{e_i^* v_i}$. If the edge is concave in, then a visible point must lie within the convex region defined by the rays $\overrightarrow{e_i^* v_i}$ and $\overrightarrow{e_i^* v_{i+1}}$ and by the curved edge e_i from v_i to v_{i+1} (see Figure 12(b)). Thus, a concave-in curved edge from v_i to v_{i+1} defines a somewhat smaller visible region than that determined by the pair of straight pseudoedges $\overline{v_i e_i^*}$ and $\overline{e_i^* v_i}$. That smaller region, however, is exactly equal to the intersection of the splinegon S with region determined by the pseudoedges. □

LEMMA 4. *Given a simple splinegon S, at most one connected component of a particular curved edge e_i can lie on the boundary of the kernel of S.*

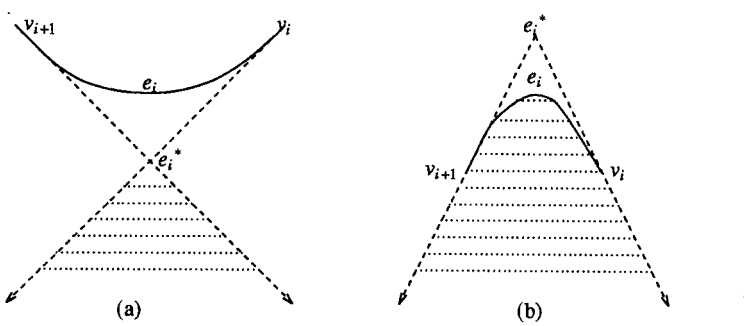

Fig. 12. (a) The region visible from an edge e_i which is concave-out. (b) The region visible from an edge e_i which is concave-in.

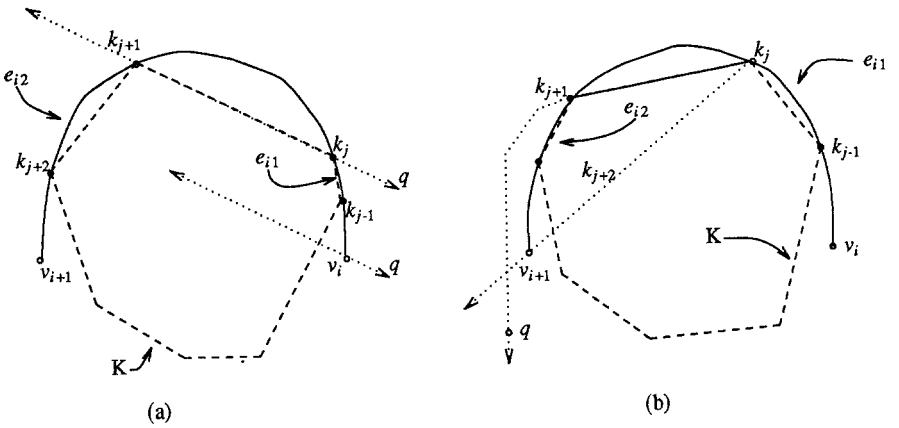

Fig. 13. Examples for Lemma 4.

PROOF. Suppose that e_{i1} and e_{i2} are two distinct segments of e_i, in counterclockwise order, both of which belong to the boundary of the kernel of S. The two segments are joined by a convex chain of straight edges. Let $\overline{k_j k_{j+1}}$ represent the straight edge which immediately follows e_{i1}. The line $\overleftrightarrow{k_j k_{j+1}}$ must contain some vertex q of the bounding polygon Q lying on a chain of edges which extends in counterclockwise order from v_{i+1} to v_i.

(1) Suppose that $\overrightarrow{k_{j+1} k_j}$ contains q (see Figure 13(a)). Then some chain of edges must join v_i to q. At best, q lies nearly at the point at infinity and a single edge connects v_i and q. Thus $\overleftrightarrow{v_i q}$ is nearly parallel to $\overleftrightarrow{k_j k_{j+1}}$. Even so, the edge $v_i q$ prevents e_{i1} from participating in the boundary of K.
(2) Suppose instead that $\overrightarrow{k_j k_{j+1}}$ contains q (see Figure 13(b)). Then some chain of edges joins v_{i+1} to q. But for that chain of edges to permit e_{i1} to participate in the boundary of K, q must lie to the left of $\overrightarrow{k_j v_{i+1}}$. Then, however, $\overline{k_j k_{j+1}}$ prevents e_{i2} from participating in the boundary of K. □

THEOREM 7. *The kernel of a simple splinegon P of N vertices can be determined in $O((B_1 + C_1)N)$ time.*

PROOF. To compute the kernel of a simple polygon of N vertices, Lee and Preparata [LP] developed a vertex-based algorithm which runs in $O(N)$ time, making a single pass around the polygon while maintaining a tentative kernel K based on the vertices seen thus far. We review this algorithm here and then give the modifications so that it will work for splinegons. By Lemma 3, this involves running the original algorithm on the bounding polygon, but providing special processing for pseudovertices, and the adjacent pseudoedges, associated with concave-in splinegon edges. In particular, in the modified algorithm, whenever it becomes clear that a curved edge may not contribute to the boundary of the kernel, it is marked *vacuous*. Otherwise, the edge is marked *potent*. By Lemma 4, at most one piece of a particular curved edge e_i can lie on the boundary of

the kernel of S. Thus, for each *potent* e_i, we maintain pointers delimiting the portion of the tentative kernel K where a segment of e_i may belong.

The Lee-Preparata algorithm assumes that the first vertex v_1 of the polygon P is reflex, for if no vertex were reflex, the polygon would be convex and thus serve as its own kernel. It also assumes that the vertices are numbered in counterclockwise order around the boundary of P. The algorithm begins with the first vertex and then moves from vertex to vertex. Upon reaching a vertex v_i, the following information is available:

(1) A doubly linked list of vertices which describes the boundary of the convex region K which is visible to all edges from v_N to v_i. If K is unbounded, the list is linear, and the vertices at the list tail and at the list head are both points in the projective plane at infinity associated with a particular direction. If K is bounded, then the list is circular and all of its vertices are finite.

(2) A pair of vertices F and L from K at maximum distance from v_i such that $\overrightarrow{v_iF}$ and $\overrightarrow{v_iL}$ both support K and such that the clockwise wedge from $\overrightarrow{v_iF}$ to $\overrightarrow{v_iL}$ contains K. If K is bounded, then F and L always represent finite points in the plane. If K is unbounded, however, F (resp. L) may represent the point at infinity at the tail (resp. head) of K's list.

The computation to be performed at a vertex v_i depends upon whether that vertex is reflex or convex. We describe the reflex case below, but omit the details for the convex case as they are exactly symmetric and thus can be easily inferred.

Suppose that v_i is reflex and that F lies on or to the left of $\overrightarrow{v_{i+1}v_i}$. Then trace the boundary of K from F to L in the counterclockwise direction. Stop upon finding a point k' where $\overrightarrow{v_{i+1}v_i}$ intersects the boundary of K. If no such point is found, then the kernel of P is null, so the algorithm halts. Otherwise, insert k' in the appropriate position as a vertex of K.

Next, trace the boundary of K in the clockwise direction from k' until reaching a second point k'' of the intersection of K and $\overrightarrow{v_{i+1}v_i}$. If we reach a point at infinity at the list tail without discovering a point k'', then let k'' be the point at infinity having direction $\overrightarrow{v_{i+1}v_i}$. In either case, insert k'' in the appropriate position as a vertex of K, and set $F = k''$. Delete all vertices of K between k' and k'' in clockwise order (see Figure 14).

Suppose that v_i is reflex and that F lies to the right of $\overrightarrow{v_{i+1}v_i}$. In this case, K remains unchanged.

In all cases, whether v_i is reflex or convex, before proceeding to the next vertex v_{i+1}, the algorithm performs a final update on both L and F.[9] Trace the boundary of K counterclockwise beginning with L (resp. F) until finding a vertex k_j such that either k_{j+1} lies to the left of (resp. on or to the right of) $\overrightarrow{v_{i+1}k_j}$ or such that k_j is the point at infinity at the list head. Set $L = k_j$ (resp. $F = k_j$).

Lee and Preparata show that the algorithm runs in linear time since all but two of the edges traced in attempting to revise K are always removed, since F

[9] In general, whenever one of F or L is set to k'' above, this final update will leave that value unchanged. The exception is the special case where v_i is convex and the line segment $\overline{v_iv_{i+1}}$ contains both k' and k''. In this instance, L must be revised a second time.

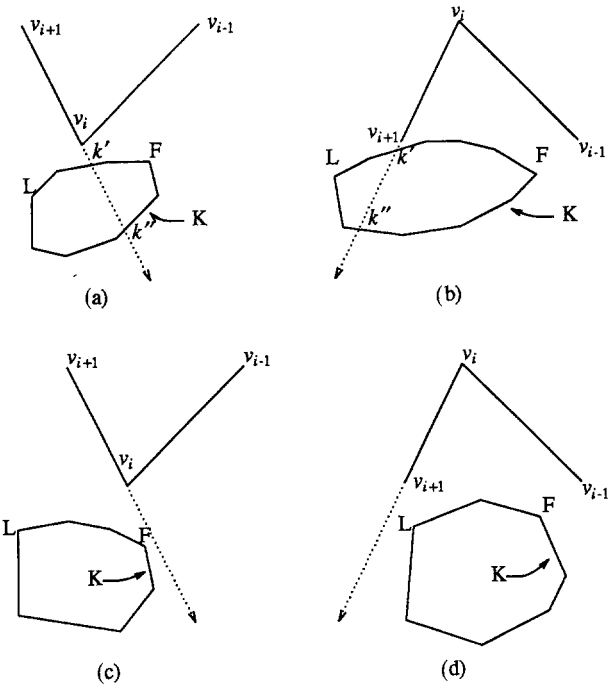

Fig. 14. (a) v_i is reflex and F lies on or to the left of $\overrightarrow{v_{i+1}v_i}$. (b) v_i is convex and L lies on or to the right of $\overrightarrow{v_iv_{i+1}}$. (c) v_i is reflex and F lies to the right of $\overrightarrow{v_{i+1}v_i}$. (d) v_i is convex and L lies to the left of $\overrightarrow{v_iv_{i+1}}$.

and L move around K only in a counterclockwise direction, and since, for each v_{i+1} for which there exists a $p \in K$, $\sum_{j=1}^{i} \alpha_j < 3\pi$, where α_j represents the interior angle of the triangle $\Delta p v_j v_{j+1}$ at p.

We make the following three modifications to the Lee-Preparata algorithm:

(1) Upon reaching a fixed vertex v_i which precedes a convex pseudovertex e_i^*, stop after revising K but before making final revisions to F and L. Perform the following computation before continuing. Suppose that the current value of L is k_j. Test k_j in constant time to determine whether it lies to the right of $\overrightarrow{v_iv_{i+1}}$. If not, then mark the curved edge e_i in the representation of S as vacuous, for in no way can it participate in defining the kernel of S (see Figure 15(a)). If it does, then test k_j in $O(B_1)$ time to determine whether it lies either on or to the right of e_i. If this second test fails, then mark the curved edge e_i in S as potent and assign it a pointer to the edge $\overline{k_jk_{j+1}}$ in K. Also mark the edge $\overline{k_jk_{j+1}}$ in K as the tail edge for e_i. The curved edge e_i may or may not participate in defining the kernel of S, but a search beginning at k_j and moving in the counterclockwise direction will yield the answer (see Figure 15(b)). If the second test succeeds, trace the boundary of K in the clockwise direction from k_j until discovering an edge $\overline{k_mk_{m+1}}$ which intersects either e_i or $\overrightarrow{v_iv_{i+1}}$. If no such edge exists, then the kernel of S is null and we halt (see Figure 15(c)). If the reported edge crosses $\overrightarrow{v_iv_{i+1}}$ but does not cross e_i, then mark e_i in S as vacuous (see Figure 15(d)). If

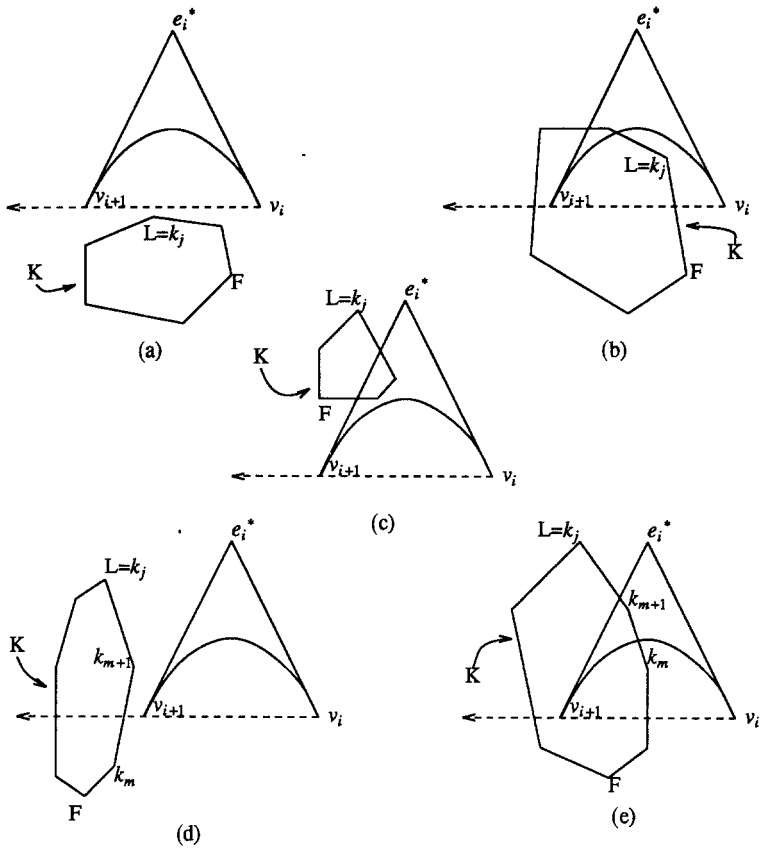

Fig. 15. The five possible scenarios after revising K at a fixed vertex which precedes a convex pseudovertex.

the reported edge does cross e_i, then mark e_i in S as potent, and mark $\overline{k_m k_{m+1}}$ in Q as the tail edge for e_i (see Figure 15(e)). In either of these last two cases, delete all vertices of K strictly between k_j and k_{m+1} in the clockwise direction. If e_i is vacuous, then these edges would have been deleted anyway in the processing of pseudovertex e_i^*. If e_i is potent, then e_i prevents these edges from contributing to the boundary of the kernel of S.

(2) When reaching a fixed vertex v_{i+1} after having just processed a convex pseudovertex e_i^*, determine whether e_i in S has been marked potent. If so, perform the following computation before proceeding with the algorithm. Suppose that the current value of F is k_j. Since e_i has been marked potent, k_j must lie to the right of $\overrightarrow{v_i v_{i+1}}$. Test k_j in $O(B_1)$ time to determine whether it lies either on or to the right of e_i.

(a) If not, then add an extra pointer in the representation of the curved edge e_i in S to the edge $\overline{k_{j-1}k_j}$ in K, and label the edge $\overline{k_{j-1}k_j}$ as the head edge for e_i. The curved edge e_i may or may not participate in defining the kernel of S, but a search beginning at the tail edge, moving in the counterclockwise direction, and ending at the head edge will yield the answer (see Figure 16(a)).

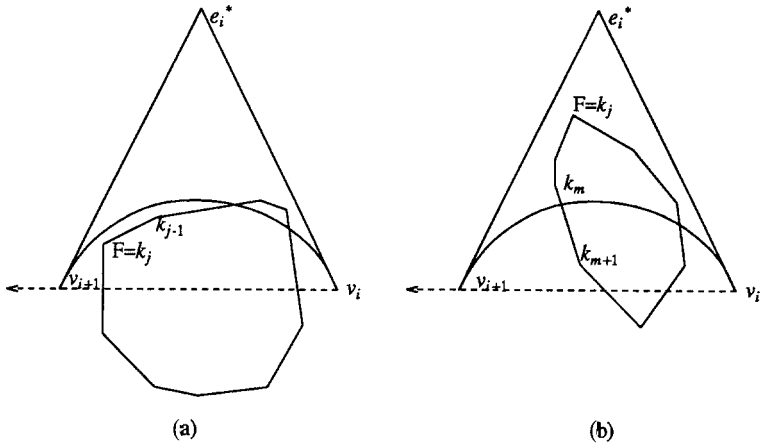

Fig. 16. The two possible scenarios after processing a pseudovertex associated with a potent curved edge of S.

(b) If so, trace the boundary of K in the counterclockwise direction from k_j until discovering an edge $\overline{k_m k_{m+1}}$ which intersects e_i. Such an edge must exist. Add a pointer from e_i in S to the edge $\overline{k_m k_{m+1}}$ and also mark $\overline{k_m k_{m+1}}$ in Q as the head edge for e_i (see Figure 16(b)). In this case, delete all vertices of K strictly between k_j and k_m in the counterclockwise direction. The curved edge e_i prevents these edges from contributing to the boundary of the kernel of S.

(3) Next, we must guarantee that, as K is repeatedly revised, labels and pointers to tail edges and head edges of potent curved edges are updated. Also, edges which become vacuous must be so identified. The only instances in which these updates must be made are those in which the deleted edges of K include some portion of either one or both of the tail edge $\overline{k_t k_{t+1}}$ and head edge $\overline{k_h k_{h+1}}$ for some curved edge e_i.

(a) Suppose the deleted portion runs in the counterclockwise direction from a point k', which lies between k_{h+1} and k_t, and ends at a point k'', which lies between k_{h+1} and k'. In other words, the tail edge and the head edge and all intervening edges are all deleted. In this case, mark the curved edge e_i as vacuous (see Figure 17(a)).

(b) Suppose the deleted portion runs in the counterclockwise direction from a point k', which lies between k_{t+1} and k_h, and ends at a point k'', which lies between k_{t+1} and k'. In other words, both the tail edge and the head edge are deleted, but some of the intervening edges remain. Test $\overline{k'k''}$ for intersection with e_i. If the entire segment lies to the right of e_i, then the kernel of S is null. Otherwise, mark $\overline{k'k''}$ both as the new head edge and as the new tail edge. Adjust the pointers at e_i (see Figure 17(b)).

Use as many of the following as pertain, if and only if neither of the above cases apply.

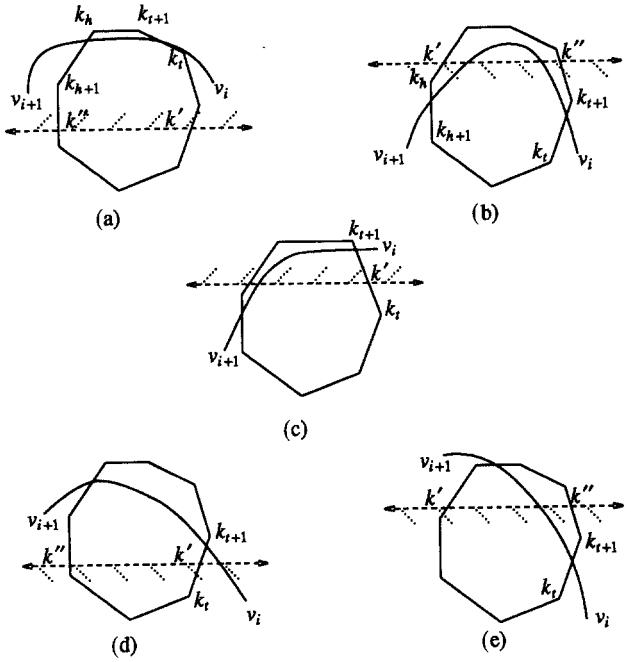

Fig. 17. Examples of preserving accurate labels for curved edges and their tail and head edges.

(c) Suppose the deleted portion runs in the counterclockwise (resp. clockwise) direction from a point k', which lies between k_t and k_{t+1} (resp. k_h and k_{h+1}). In other words, the interior part of the tail edge (resp. head edge) is deleted. In this case, mark $k_t k'$ (resp. $k' k_{h+1}$), the remaining portion, as the new tail (resp. head) edge for e_i. Update the pointer at e_i (see Figure 17(c)).

(d) Suppose the deleted portion runs in the clockwise (resp. counterclockwise) direction from a point k', which lies between k_t and k_{t+1} (resp. k_h and k_{h+1}), to a point k''. In other words, the exterior part of tail edge (resp. head edge) is deleted. Test the point k' to determine whether it lies to the right of e_i. If so, then mark $k'k''$ as the new tail (resp. head) edge. If not, then mark $\overline{k'k_{t+1}}$ (resp. $\overline{k'k_h}$) as the new tail (resp. head) edge. Update the pointer at e_i (see Figure 17(d)).

(e) Suppose the deleted portion runs from a point k' through both k_t and k_{t+1} (resp. k_h and k_{h+1}), and ends at a point k''. In other words, the entire tail (resp. head) edge is deleted. Mark $\overline{k'k''}$ as the new tail (resp. head) edge and update the pointer at e_i (see Figure 17(e)).

These three routines provide the basis for the theorem. Determining the bounding polygon requires $O(C_1 N)$ time. The entire Lee-Preparata algorithm runs in $O(N)$ time. If no tracing is done, then modification (1) requires constant time. If n edges are traced, then $n-1$ edges are deleted. Thus the tracing and the deleting may be charged to those edges, and only $O(B_1)$ time needs be charged to each call to modification (1). The same argument applies to modification (2). Whenever the original algorithm revises K by deleting all vertices in a particular

direction between k' and k'', it explicitly traces all of the intervening edges. Thus the information about the relative positioning of head and tail edges required by modification (3) can be computed at the same time and all marks and pointers may be updated, incurring at most a constant charge per edge. When the main algorithm is complete, we perform one final trace around K. We test the edges between each tail-head pair for intersection with the respective curved edge, and update K accordingly. This single pass around K and S requires $O(B_1N)$ time. □

In the next algorithm the structure of the bounding polygon is used more than the polygon itself. In fact, neither the pseudoedges nor the pseudovertices are ever explicitly determined, for the approximation they would provide for curved edges is not sufficiently accurate to decide which edges participate in the convex hull. The vertex list for the bounding polygon, however, does contain an entry for each nonstraight edge of the splinegon. Thus we apply the original Graham-Yao polygon algorithm [GY] to the bounding polygon and yield the convex hull of the splinegon merely by adding special procedures for processing those vertices which were really pointers to curves rather than vertices:

THEOREM 8. *The convex hull of a simple splinegon of N vertices can be computed in time and space* $O((B_1+C_1+D_1)N)$.

PROOF. The Graham-Yao algorithm assumes that the vertices $v_1, v_2, \ldots, v_m, v_{m+1}, \ldots, v_N$ of the simple polygon P are given in clockwise order around the boundary, that v_1 is the vertex of minimum x-coordinate, and that v_m is the vertex of maximum x-coordinate. Define the path along P from v_i to v_j as a *pocket* of P if no vertex along the path lies to the left of the directed line segment $\overrightarrow{v_iv_j}$; call $\overrightarrow{v_iv_j}$ the *top* of the pocket; say that a vertex lies inside (resp. outside) the pocket if it lies (resp. does not lie) in the closed region bounded by the pocket and its top. Graham and Yao characterize the task of finding the convex hull of P as that of identifying a circular list of vertices such that each consecutive pair delimits a pocket of P and such that the pocket tops and the vertices form a convex polygon (see Figure 18(a)). The set of vertices of the convex hull must include both v_1 and v_m and may not include any vertex lying inside a pocket of P, except for its endpoints. Thus the convex hull problem can be divided into two symmetric pieces: compute the top hull (resp. bottom hull) of P, which corresponds to the *left hull* of the oriented chain $v_1, v_2, \ldots, v_{m-1}, v_m$ (resp. $v_m, v_{m+1}, \ldots, v_N, v_1$).

The *left hull* algorithm maintains a stack Q of candidate hull vertices, where q_0 (resp. q_t) represents the bottom (resp. top) element of the stack, with the invariants that $q_0, q_1, \ldots, q_t$ always form a convex polygon and, for $2 \le i \le t$, q_{i-1}, q_i always delimit a pocket of P. To find the top hull of P, the algorithm begins by setting $q_0 = v_m$, $q_1 = v_1$, and q_2 to be the first vertex lying to the left of the directed line segment $\overrightarrow{v_1v_m}$. After pushing a vertex v_i onto the stack, the algorithm moves from vertex to vertex along the chain, searching for the first vertex x outside the current convex polygon. If v_{i+1} lies to the left of $\overrightarrow{q_{t-1}q_t}$, then x is v_{i+1}. Otherwise, the algorithm tests whether v_{i+1} belongs to the pocket with

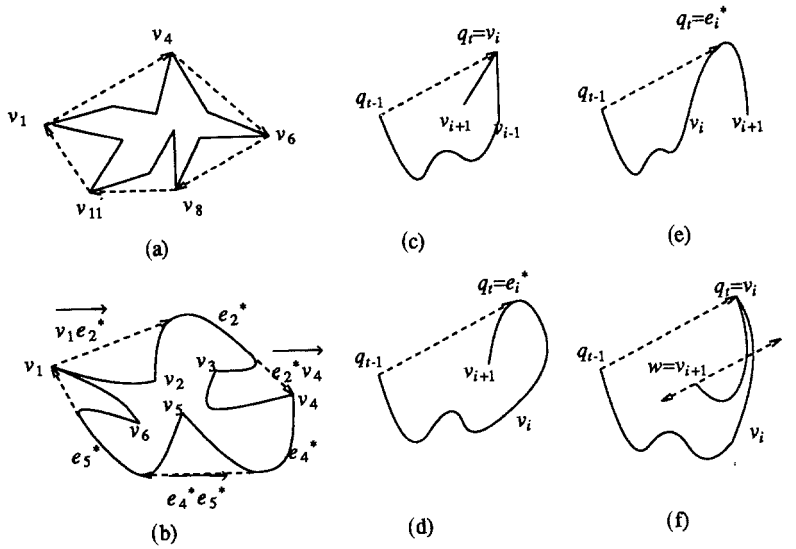

Fig. 18. (a) and (b) The convex hull of a simple polygon and a simple splinegon. (c) The polygon pocket test. (d)–(f) The splinegon pocket test for convex, reflex, and fixed q_t.

endpoints q_{t-1} and q_t. If so (resp. not), then x will be the first successor of v_{i+1} to lie to the left of $\overrightarrow{q_{t-1}q_t}$ (resp. $\overrightarrow{q_tq_0}$). Before inserting x into the stack, as many vertices are popped from the stack as necessary so that x lies to the right of the new $\overrightarrow{q_{t-1}q_t}$. The algorithm uses linear time and space, as each vertex not rejected outright is inserted into the stack exactly once and deleted at most once. At termination, the path from q_t to q_0 along P also forms a pocket, and thus Q describes the top hull of P.

The splinegon algorithm is nearly identical to the polygon algorithm, but it must consider both the fixed vertices and the pseudovertices of the bounding polygon. In this application, however, we never compute the coordinates of the pseudovertices explicitly; each pseudovertex merely points to a description of the corresponding curved edge. Thus, we need to define what we mean by the directed line segment $\overrightarrow{vw}$ where at least one of v and w is a pseudovertex:

(1) $\overrightarrow{v_ie_j^*}$ represents the directed line segment of maximum length which extends from v_i to a point y on e_j and which supports e_j so that each point of e_j lies on or to the right of $\overrightarrow{v_ie_j^*}$.
(2) $\overrightarrow{e_i^*v_j}$ represents the directed line segment of maximum length which extends from a point x on e_i to v_j and which supports e_i at x so that each point of e_i lies on or to the right of $\overrightarrow{e_i^*v_j}$.
(3) $\overrightarrow{e_i^*e_j^*}$ represents the directed line segment which extends from a point x on e_i to a point y on e_j and which supports e_i at x and e_j at y so that each point of either e_i or e_j lies on or to the right of $\overrightarrow{e_i^*e_j^*}$.

Next, we augment our definitions of what it means for a vertex x to lie to the left of $\overrightarrow{vw}$:

(1) e_j^* lies to the left of $\overrightarrow{vw}$ if any portion of e_j lies to the left of $\overrightarrow{vw}$.

(2) If $\overrightarrow{ve_j^*}$ (resp. $\overrightarrow{e_j^*w}$) intersects e_j only at v_{j+1}, we shall consider v_{j+1} to lie to the left of the respective directed line segment.

Given these augmented definitions, the convex hull of a simple splinegon P can now be identified by a circular subsequence of the vertices of the bounding polygon Q such that each consecutive pair delimits a pocket and such that a convex splinegon is formed by the pocket tops, the fixed vertices, and the portions of those edges identified by pseudovertices which join adjacent pocket tops (see Figure 18(b)).

As in the polygon case, the circular list of vertices is determined by two applications of the *left hull* algorithm. However, we must first insert the points of minimum and maximum x-coordinate as fixed vertices and renumber the vertices accordingly. We must also define the test to determine whether a vertex v immediately succeeding q_t and lying on or to the right of $\overrightarrow{q_{t-1}q_t}$ lies inside a pocket delimited by q_{t-1} and q_t. In the polygon case, the pocket test is simple: for $q_t = v_i$, v_{i+1} is v and it lies inside (resp. outside) the pocket if it does (resp. does not) lie to the left of the directed line segment $\overrightarrow{v_{i-1}v_i}$ (see Figure 18(c)). In the splinegon case, we have multiple cases. If q_t is a reflex pseudovertex, then v belongs to the pocket (see Figure 18(d)). If q_t is a convex pseudovertex, then v does not belong to the pocket (see Figure 18(e)). If q_t is a fixed vertex v_i, let w represent whichever of v_{i-1} and v_{i+1} lies closest to the line l containing $\overrightarrow{q_{t-1}q_t}$. Find the intersection with both e_{i-1} and e_i of the line passing through w parallel to l. The intersection either consists of one component from each edge, or of both w and a second point from one edge and one component from the other. In the latter case, discard w. Now, if the one component from e_i is to the left (resp. right) of the one component from e_{i-1}, then v does (resp. does not) belong to the pocket (see Figure 18(f)).

As modified, the Graham–Yao algorithm will provide a list of vertices describing the left hull of each half of the splinegon. That list will include both fixed vertices and pseudovertices. A single transversal of that final list can determine which portion of each of the curved edges associated with a listed pseudovertex actually lies on the convex hull. Then the convex hull is formed by linking fixed vertices and curved segments with straight segments. □

We note that Schäffer and Van Wyk [SV] have achieved the same result by a different approach, as we describe in the next section.

5. Direct Approach. In the previous two sections we have presented two distinct methods for extending polygon algorithms. The carrier polygon approach primarily applies to extensions of algorithms on convex polygons. The bounding polygon approach has particular application in the extension of vertex-based algorithms. In general, however, edge-based algorithms need neither the artifice of focusing on the carrier polygon nor the artifice of focusing on the bounding polygon. Where the original algorithm considered the line segment from v_i to v_{i+1}, the revised algorithm considers the curved edge e_i which joins v_i to v_{i+1}.

All that is needed is a revised procedure for processing the individual edges which accounts for the greater freedom enjoyed by curves.

Straight polygon edges enjoy many properties which curved splinegon edges do not. Two edges of a polygon intersect at most in a single point, or perhaps a single line segment, whereas two splinegon edges may intersect arbitrarily often. A line supporting two edges of a polygon passes through at least one vertex of each edge, whereas a line supporting two splinegon edges may contain just one interior point from each edge. A polygon edge is always monotone in every direction but one. Some splinegon edges are not monotone in any direction, although every splinegon edge can be divided into at most three pieces such that each piece is monotone in the chosen direction. Two nonhorizontal polygon edges intersect in their interiors if and only if, when the endpoints are ordered by y-coordinate, the edges intersect the horizontal lines through the middle two endpoints in different order. For y-monotone splinegon edges, this test establishes the parity of the number of crossings, but nothing more. If both endpoints of a polygon edge lie in the interior of a convex object, the entire edge lies in the interior of that object. If both endpoints of a splinegon edge lie in the interior of a convex object, the edge may still intersect the boundary of the object and part of the edge may lie outside.

OBSERVATION 3 (The Direct Approach). Edge-based polygon algorithms can be extended directly to splinegons, provided that those assumptions about the behavior of edges which apply only to straight line segments are removed, additional tests are inserted to accommodate the more general edges, and splinegon edges are split into monotone pieces as necessary.

THEOREM 9. *The intersection of two convex splinegons of at most N vertices each can be computed in $O(A_1 N + C_1 \log N + B_1)$ time.*

PROOF. We extend the method of [Sh] to compute the intersection of the two convex splinegons P and Q, each of at most N vertices. First, we use the result of Theorem 4 to locate a point x in the intersection of P and Q, provided one exists. Let x be the origin of a polar coordinate system, and draw rays from x through each of the vertices of P, dividing the plane into sectors. Pick a vertex of Q and determine in which sector of P it lies. Scan around Q once, testing each edge of Q for intersection with the relevant edges of P. Since no backtracking is done, all intersection points can be determined in $O(A_1 N)$ time. The intersection consists of chains taken alternately from splinegons P and Q with the intersection points in between. For further details, see either [So] or [DS2]. □

THEOREM 10 [TV]. *The internal horizontal vertex visibility information for a simple splinegon can be computed in $O(N \log\log N + (B_1 + C_1)N)$ time.*

PROOF. Tarjan and Van Wyk [TV] give an $O(N \log\log N)$-time algorithm for computing the internal horizontal vertex visibility information for a simple

polygon of N vertices. The horizontal line segments which join a vertex to its visible edge or edges define a partition of the polygon into trapezoids. As they note, their algorithm extends directly to splinegons. To do so, first add additional vertices to guarantee that each splinegon edge is monotone in the y-direction, using $O(C_1 N)$ time, and adding at most $2N$ new vertices. Once this modification has been made, the algorithm runs unchanged except for the fact that it computes the intersection of horizontal lines with curved edges rather than straight edges, an $O(B_1)$-time process, and reports "trapezoids" bounded by a pair of horizontal line segments and a pair of y-monotone curved edges. □

THEOREM 11 [TV]. A splinegon of N vertices can be tested for simplicity in $O(N \log\log N + (A_1 + B_1 + C_1)N)$ time.

PROOF. By using their $O(N \log\log N)$-time algorithm to compute both the internal and the external horizontal vertex visibility information for a polygon, Tarjan and Van Wyk can detect whether a polygon is simple in $O(N)$ additional time. They note that the algorithm extends directly to splinegons. The splinegon version requires a final stage: if the splinegon still appears to be simple after running the original algorithm, test the pair of curved side-edges from each of the trapezoids reported in either iteration of visibility testing for intersection; if no intersections are found, the splinegon is indeed simple. This revised algorithm runs in $O(N \log\log N + (A_1 + B_1 + C_1)N)$ time. □

THEOREM 12 [DSV]. *A simple splinegon of N vertices can be decomposed into the union of monotone pieces with simple carriers in $O(N \log\log N + (B_1 + C_1)N)$ time. The total number of vertices in the decomposition is $O(N)$.*

PROOF. See [DSV]. □

THEOREM 13 [DSV]. *The boundary intersection and/or the area intersection of two N-sided simple splinegons can be detected in $O(N \log\log N + (A_1 + B_1 + C_1)N)$ time.*

PROOF. See [DSV]. □

THEOREM 14 [SV]. *The convex hull of a simple splinegon of N vertices can be computed in linear time and space.*

PROOF. To extend the algorithm of Graham and Yao [GY] to compute the convex hull of piecewise-smooth Jordan curves, a subset of the simple splinegons, Schäffer and Van Wyk [SV] first revised the Graham–Yao vertex-based algorithm to run as an edge-based algorithm. Thus, instead of maintaining a stack of vertices which belong to the convex hull, the Schäffer–Van Wyk algorithm maintains a stack of edges which participate in the convex hull. The main calculation on edges in the revised algorithm consists of computing a half-plane of desired

orientation which contains both edges and whose bounding line supports both edges. All half-planes possessing the final two properties have bounding lines defined by endpoints of the edges. In the curved world, however, determining the desired half-plane will require computing tangents to curves. Nonetheless, this algorithm is extended directly to piecewise-smooth Jordan curves by adapting the procedures for processing edges. See [SV] for details. □

6. Limitations of Splinegons. In the previous sections we have described numerous problems where the algorithms for straight-edged polygons can be extended to work for splinegons. Unfortunately, some things explicitly cannot be done. Many geometric algorithms begin by decomposing simple polygons into a disjoint set of monotone pieces, convex pieces, or triangles without adding any new vertices. Fournier and Montuno [FM] show that polygon decomposition into the union of convex polygons, of star-shaped polygons, of monotone polygons, and of triangles are all linear-time equivalent to solving the all vertex-edge horizontal visibility problem. As discussed in the previous section, Tarjan and Van Wyk [TV] have recently demonstrated that all horizontal-visibility information can be computed in time $O(N \log\log N)$. Thus, decomposition of a simple polygon into convex polygons, star-shaped polygons, monotone polygons, and triangles can all be accomplished in $O(N \log\log N)$ time.

In general, however, splinegons do not have the same flexibility. As described in previous sections, we can determine whether a given simple splinegon is convex, star-shaped, or monotone in $O(N)$ time. The splinegon extensions of the Tarjan–Van Wyk algorithms allow us either to decide whether a given splinegon is simple or decompose a simple splinegon into the union of monotone pieces all having simple carriers in $O(N \log\log N)$ time. Decomposition of splinegons into convex pieces, however, is problematic. Some splinegons are inherently nonconvex. For example, a splinegon with a single edge which is concave-out can never be decomposed as a union of convex pieces. The only efficient solution is to decompose the original splinegon into the union of monotone pieces, and then decompose each one into the union and difference of a collection of convex pieces [see DSV]. The decomposition will be expressed in the form $\bigcup_j (\bigcup_i A_{ij} - \bigcup_i B_{ij})$, where j ranges over the number of monotone splinegons and the A's and B's describe the decomposition of each individual monotone splinegon. As a linear number of vertices may be added in the process, the size of the minimum decomposition does not depend solely on the number of reflex angles. Algorithms dependent on convex decompositions have been designed to handle unions well. In many, difference can be easily accommodated. The restricted ordering of the union and difference operations, however, raises questions about the usefulness of this decomposition.

Triangulation is even more problematic. Dividing an N-sided convex polygon P into triangles is a simple linear-time procedure. By convexity, any diagonal, an open line segment joining two nonadjacent vertices, lies in the interior of P. Any collection of $N-2$ nonintersecting diagonals divides P into triangles. Triangulating a convex splinegon S is equally easy. Any collection of diagonals

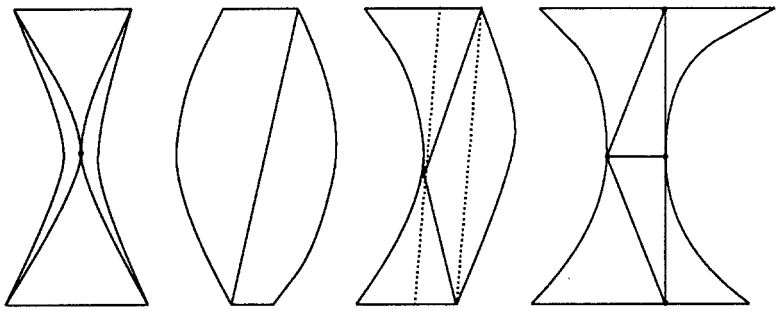

Fig. 19. The triangulation of trapezoids.

which triangulates P also triangulates S, and each triangle in the decomposition is convex.

Triangulation of simple splinegons, however, is complicated. Many splinegons cannot be triangulated without the addition of Steiner points. For example, the splinegonal trapezoid depicted in Figure 19(a) can never be triangulated merely by adding straight edges between existing vertices. Nor will curved edges without inflection points between existing vertices suffice. The minimum-size triangulation results from stretching copies of the two longer curved edges toward each other until they touch at a point. Insert that point and the resulting four curved edges.

If triangulation requires the creation of new curved edges, then its usefulness becomes questionable. Each splinegon, however, can be triangulated using a linear number of line segments and a linear number of new vertices. Computing horizontal visibility information yields a decomposition of an arbitrary splinegon into a linear number of trapezoids. No vertex produces more than two new vertices in the trapezoidal decomposition. A trapezoid whose side-edges are both concave-in is convex and can be triangulated by adding either diagonal; no new vertex is necessary (see Figure 19(b)). If one side-edge is concave-in and one is concave-out, determine the point on the concave-out edge which is closest to the line segment defined by the vertices of the concave-in edge; add line segments from those vertices to the new point (see Figure 19(c)). If both side-edges are concave-out, add the points of minimum separation on the two curved edges as vertices. Find a line supporting one of the side-edges at its new vertex, and add the portion of that line which connects the top and bottom edge of the trapezoid. Add the two points of contact with the trapezoid as vertices; finally, triangulate the interior polygon formed by the new vertices (see Figure 19(d)). At most four vertices are added to triangulate any trapezoid.

In polygon algorithms, triangulation has two major selling points: all regions can be triangulated without adding any new vertices; triangles are always convex. A splinegon can also be triangulated efficiently, but the triangles may not be convex, and a linear number of new vertices may be required. The lack of convexity and the potential size of the new decomposition may prevent the efficient extension of polygon algorithms dependent on triangulation.

A prime example of a polygon algorithm dependent on triangulation is Kirkpatrick's optimal algorithm for planar point location [Ki].[10] He begins by triangulating every region of the given planar subdivision. A hierarchy is then established by removing an independent set of vertices and retriangulating. The efficiency of the algorithm depends on the fact that the number of vertices constantly decreases, and, for each vertex discarded, the number of triangles decreases by two. In the splinegon case, even allowing curved triangulations, there is no such guarantee. First, new vertices may be required to triangulate the original planar subdivision, an acceptable one-time charge. But, as vertices are discarded, we can still get arbitrarily complicated new regions to triangulate. The retriangulation might well add more vertices than had been discarded.

Triangulation and convex decomposition represent a class of algorithms which may not extend profitably to splinegons. In the graphics world the extension of polygon edges form a convex decomposition of the plane. From any viewing point within one convex region, the same edges are visible. The list of visible edges need be updated only when the viewing point crosses a boundary. One technique used in motion planning entails unfolding polyhedral objects until they are planar. A straight path can be chosen in the plane, and then wrapped back along the surface of the polyhedron. Duality transformations which map lines to points, or planes to points have become an increasingly powerful tool within computational geometry. None of these methods extend easily into the curved world.

Does the existence of methods which do not extend to the splinegon world mean that there are a class of problems which require asymptotically more time in the splinegon world than they do in the polygonal world? Not necessarily. There is some indication that alternative methods can be substituted which allow the asymptotic complexity to remain unchanged. Edelsbrunner *et al.* [EGS] and Sarnak and Tarjan [ST] have provided optimal algorithms for planar point location which do in fact extend to splinegons. Not only do these algorithms equal the Lipton-Tarjan algorithm and the Kirkpatrick algorithm in time and space complexity, but they surpass the older algorithms in practicality. The Edelsbrunner *et al.* result depends on monotone pieces rather than triangles [EGS]. The monotone decomposition of splinegons is efficient and clean. Consequently, their algorithm extends directly to splinegons, as they expect. The Sarnak-Tarjan algorithm depends solely on the monotonicity of the individual edges [ST].

It is hard to assess the degree to which alternative methods can compensate for those methods which extend poorly from the straight world to the curved world. It may be that monotone decomposition, for example, can make up for whatever power convex decomposition and triangulation will lack and thus will emerge as an increasingly powerful tool in the polygonal world as well as in the splinegonal world. Further study is needed.

[10] Lipton and Tarjan developed an optimal algorithm for this problem some years earlier [LT1], [LT2]. A significant theoretical achievement, their algorithm is much harder to implement than Kirkpatrick's.

7. Higher-Dimensional Extensions. The extension of the development above to three dimensions raises interesting mathematical questions. The basic question is how to characterize three-dimensional curved objects. This question is not fully resolved since two reasonable alternatives are possible, and yet neither accommodates the full spectrum of real-world three-dimensional objects. This section presents these two definitions of a *splinehedron S* as a modification of a *carrier polyhedron P.* We discuss the effect of the choice of definition both on the range of objects accommodated and on the applicability of each of the approaches used in two dimensions, and sketch one concrete result using the first model.

We begin by defining a *splinehedron* as a modification of a *carrier polyhedron* P in which each face of P is replaced by a curved surface bounded by the same vertices and edges. The ith face f_i of S together with the corresponding face p_i of P must enclose a convex region $S\text{-}seg_i$. A *convex splinehedron* both encloses a convex region and has a convex carrier polyhedron.

This splinehedron model limits us to objects whose curved faces join in straight line segments but is attractive in that it allows direct extension of the three main splinegonal methods into the three-dimensional world. The carrier polyhedron approach still works. Given a convex splinehedron S, the plane defined by the ith face p_i of the carrier polyhedron P divides space into two half-spaces. The "outside" half-space contains the convex region $S\text{-}seg_i$, and the "inside" half-space contains the convex splinehedron $S_i = S - S\text{-}seg_i$. S_i can be considered a convex polyhedron which is supported by the given plane along a face. Furthermore, the convexity of S dictates that $S\text{-}seg_i$ is enclosed in the solid defined by the "outside" half-space determined by p_i of P and by the "inside" half-spaces determined by the faces adjacent to p_i. Consequently, without any direct manipulation of curved faces, the behavior of S can be reasonably approximated.

This splinehedron model readily accommodates the bounding polyhedron approach. Given an arbitrary splinehedron S, we create a bounding polyhedron Q. Q contains all of the original vertices and all of the original edges of S, the fixed edges and fixed vertices. For each triangular face f_i of S which is not planar, let f_i^* represent the point of intersection of the three planes each of which supports f_i along an edge.[11] As in the two-dimensional case, this point may have finite coordinates, or may represent a point at infinity. The pyramid defined by p_i and the pseudovertex f_i^* contains $S\text{-}seg_i$. Insert the pseudovertex f_i^* into Q along with pseudoedges joining it to each of the fixed vertices of f_i. A face bounded only by fixed edges is called a fixed face. Otherwise, it is a pseudoface. A pseudoface is considered *loose* if its only intersection with the curved face it supports is the fixed edge. If the intersection has positive area, the pseudoface is considered *tight.*

A face-based polyhedron algorithm can be extended to a face-based splinehedron algorithm in this model using the direct approach. A face of the splinehedron

[11] If f_i has more than three vertices, then the tangent planes defined by its edges may not intersect in a single point. This irregularity does not present a problem. We still insert a single pseudovertex f_i^* into Q, but f_i^* will represent the collection of vertices defined by the intersection of the tangent planes as well as the line segments which connect them.

resembles a face of a polyhedron in that it is bounded by a collection of vertices and line segments. In extending a polyhedron algorithm, however, all assumptions based on the flatness of the faces (e.g., the monotonicity of faces, that the intersection of two faces consists of a single component) must be updated.

Although each of the carrier polyhedron, bounding polyhedron, and direct approaches applies to this model, our only example uses the carrier polyhedron approach:

THEOREM 15. *The intersection of two preprocessed convex splinehedra of at most N vertices each can be detected in $O((A_2+B_1)\log N+(C_2+E_2)\log^2 N)$ operations.*

PROOF. The algorithm for detecting the intersection of two splinehedra follows that of the polyhedron algorithm of [DK1]. Each splinehedron will be represented as a sequence of parallel splinegonal cross-sections, one per vertex, and all their connecting faces and edges. Each cross-section of the splinehedron forms a splinegon having the corresponding cross-section of the carrier polyhedron as a carrier polygon. Each pair of adjacent splinegonal cross sections and all of their connecting edges and faces describe a splinedrum whose side faces are curved patches. The carrier polygons for these adjacent splinegonal cross sections together with their connecting edges and faces describe a carrier drum for the splinedrum (see Figure 20). Thus a splinehedron can be viewed as a sequence of splinedrums. Each splinedrum can be specified by a circular list of its side-edges, pointers to the description of the individual curved side-faces, and the planes containing the top and bottom faces. The algorithm centers around detecting the intersection of the two middle splinedrums. In each instance in which the two splinedrums do not intersect, half of one splinehedron may be removed from future consideration. See either [So] or [DS2] for details. □

In the model described above, a splinehedron is defined from its carrier polyhedron: the vertex list and the edge list remain unchanged; each face entry is modified to contain an equation of the surface in which the face lies. Under this definition, adjacent curved faces must join together at a straight line segment—

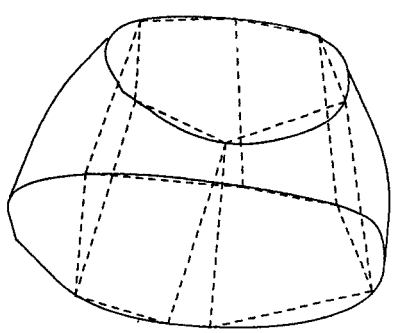

Fig. 20. A splinedrum defined on two adjacent splinegonal cross sections and its carrier drum defined on two adjacent polygonal cross sections.

not what happens in practice! Our second model allows each edge entry to be modified to include an equation of the planar curve which joins its two vertex-endpoints and which separates the two faces.[12] The ith face of the splinehedron S together with the corresponding face of the carrier polyhedron P and each plane defined by a curved edge of S and the corresponding straight edge of P bound a convex region $S\text{-}seg_i$.

Allowing each face of P to be replaced by a curved surface containing the vertices of the original face but having curved edges more adequately reflects the real world. This model, however, dramatically alters the efficacy of the three methods of polyhedron extension. A splinehedron in this model still has a carrier polyhedron, but it no longer approximates the splinehedron as well as it did in the restricted model. Given a convex splinehedron S, the plane defined by the ith face p_i of its carrier polyhedron P divides space into two half-spaces. The "outside" half-space contains the convex region $S\text{-}seg_i$, but also some portion of each of the adjacent S-*seg*'s and possibly some part of an unlimited number of other neighboring S-*seg*'s. The "inside" half-space contains the remainder, a convex splinehedron which cannot be defined explicitly without specific calculations for each instance of S and i. In addition, the solid defined by the "outside" half-space determined by p_i of P and by the "inside" half-spaces determined by the faces adjacent to p_i no longer contains $S\text{-}seg_i$. Whenever f_i represents a curved face all of whose edges are curved, then each of those edges as well as a neighboring region will be excluded from the solid.

The bounding polyhedron approach is even more problematical. A bounding polyhedron should contain all of the original vertices of a splinehedron as well as a collection of pseudovertices which approximate the faces. The previous method of defining the bounding polyhedron is no longer valid. Once edges are defined as curves, there no longer exists a single plane which supports a face along an entire edge, approximating the face in the neighborhood of that edge. Only one alternative method seems promising. For each triangular face f_i of S which is not a planar polygon, let f_i^* represent the point of intersection of the three planes each of which is tangent to f_i at a vertex. The pseudovertices of the form f_i^* only form a subset of the new vertices which must be inserted into Q. Neighboring pyramids will intersect each other forming numerous new vertices and edges. The bounding polyhedron defined in this fashion could have as many vertices as the sum of the number of faces of S with three times the number of vertices.

The direct approach is also adversely affected, but not to the same extent. A face of a splinehedron under this model has a smaller resemblance to a face of a polyhedron. Not only is the face not flat, it also does not have a planar boundary, let alone a piecewise-straight planar boundary. The modifications which must be made to a polyhedron algorithm become far more complicated.

See [DS2] for a description of the effect of the alternative model on intersection detection.

[12] We restrict an edge to being a planar curve because a nonplanar edge would dramatically complicate the model.

8. Conclusion. We give here a series of observations which provide an algorithmic basis for extending linear algorithms into curved space. These observations, though simply stated, give rise to powerful results in the computational geometry of curved objects.

To recapitulate:

OBSERVATION 1 (Carrier Polygons). The carrier polygon imposes sufficient structure on a convex splinegon that polygon algorithms can be extended to splinegons with the only modification being *ad hoc* procedures to allow for all possible behavior of $S\text{-}seg_i$ and its bounding triangle $S\text{-}tri_i$. Only infrequently will an examination of the precise behavior of e_i be necessary. Although the carrier polygon seems to be most useful for convex polygons, it can also be an important tool in processing monotone splinegons, but only if it is simple.

OBSERVATION 2 (Bounding Polygons). When the carrier polygon itself is insufficient, it may often be enhanced by adding further structure based solely on the local structure of the splinegon. This structure results in a bounding polygon with *fixed vertices* corresponding to vertices of the splinegon and *pseudovertices* corresponding to edges of the splinegon. Insofar as possible, computations involving the polygon in the linear case now involve the bounding polygon with only local attention to the splinegon's curved edges.

OBSERVATION 3 (The Direct Approach). Edge-based polygon algorithms can be extended directly to splinegons, provided that those assumptions about the behavior of edges which apply only to straight line segments are removed, additional tests are inserted to accommodate the more general edges, and splinegon edges are split into monotone pieces as necessary.

We have presented three strategies for generalizing straight-line algorithms to apply to curved objects. The results listed in Table 1 demonstrate the efficacy of these approaches and suggest a broader applicability. Numerous other extant algorithms for polygons could be generalized by carefully applying the techniques given here, and we encourage readers of this paper to do so. In the future, however, we hope that developers of new algorithms for computational geometry problems will apply these strategies at the outset in order to state their results as generally as possible.

Despite the efficacy of these techniques, we conjecture that there are other two-dimensional problems of geometry for which none of the approaches we describe is sufficient. Furthermore, many issues in three-dimensional curved geometry remain unresolved. Additional research is necessary. There also remain the open problems of finding techniques to realize the splinegon oracles. These would be particularly interesting for specific cases of curves (e.g., piecewise cubic, or polynomial in degree d). Recent work by Bajaj and Kim addresses this issue [BK].

Table 1

Problem	Approach	Time complexity
Intersection detection of a line with a convex splinegon	Carrier	$O(B_1+\log N)$
Intersection detection of two convex splinegons	Carrier	$O(A_1+B_1+C_1 \log N)$
Intersection detection of convex splinegon and splinedrum	Carrier	$O((C_1+E_2)\log N+A_1+B_1+E_1)$
Intersection detection of two convex splinedrums	Carrier	$O(C_2 \log N+A_2)$
Intersection detection of two convex splinehedra	Carrier	$O((A_2+B_1)\log N+(C_2+E_2)\log^2 N)$
Testing point inclusion for a convex polygon using hierarchy	Carrier	$O(\log N)$
Testing point inclusion for a convex splinegon	Carrier	$O(B_1+\log N)$
Area computation for an arbitrary splinegon	Carrier	$O(F_1 N)$
Diameter computation for a convex splinegon	Bounding	$O((A_1+C_1)N)$
Monotonicity determination for a simple splinegon	Bounding	$O(C_1 N)$
Kernel computation for a simple splinegon	Bounding	$O((B_1+C_1)N)$
Convex hull computation for a simple splinegon	Bounding	$O((B_1+C_1+D_1)N)$
Intersection computation for a convex splinegon	Direct	$O(A_1 N+C_1 \log N+B_1)$
Horizontal visibility computation for a simple splinegon [TV]	Direct	$O(N \log\log N+(B_1+C_1)N)$
Simplicity testing for an arbitrary splinegon [TV]	Direct	$O(N \log\log N+(A_1+B_1+C_1)N)$
Monotone decomposition of a simple splinegon [DSV]	Direct	$O(N \log\log N+(B_1+C_1)N)$
Intersection detection for two simple splinegons [DSV]	Direct	$O(N \log\log N+(A_1+B_1+C_1)N)$

Acknowledgment. We are indebted both to Chris Van Wyk and to an anonymous referee for their careful reading of this paper and their numerous constructive suggestions.

References

[BK] Bajaj, C., and Kim, M.-S., Algorithms for planar geometric models, *Proceedings of the 15th ICALP*, Tampere, Finland, LICS 317, Springer-Verlag, Berlin, 1988.

[CD] Chazelle, B., and Dobkin, D., Intersection of convex objects in two and three dimensions, *Journal of the Association for Computing Machinery*, **34**, 1987, 1-27. A preliminary version of this paper, entitled "Detection is easier than computation," appeared in the *Proceedings of the ACM Symposium on Theory of Computing*, Los Angeles, CA, May 1980, pp. 146-153.

[DK1] Dobkin, D., and Kirkpatrick, D., Fast detection of polyhedral intersections, *Theoretical Computer Science*, **27**, 1983, 241-253.

[DK2] Dobkin, D., and Kirkpatrick, D., A linear algorithm for determining the separation of convex polyhedra, *Journal of Algorithms*, **6**, 1985, 381-392.

[DSi] Dobkin, D., and Silver, D., Recipes for geometry and numerical analysis—part I: an empirical study, *Proceedings of the ACM Symposium on Computational Geometry*, Urbana, IL, 1988, pp. 93–105.

[DS1] Dobkin, D., and Souvaine, D., Computational geometry—a user's guide, in *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics* (J. T. Schwartz and C. K. Yap, eds.), Erlbaum, Hillsdale, NJ, 1987, pp. 43–93.

[DS2] Dobkin, D., and Souvaine, D., Detecting and computing the intersection of convex objects, submitted for publication.

[DSV] Dobkin, D., Souvaine, D., and Van Wyk, C., Decomposition and intersection of simple splinegons, *Algorithmica*, **3**, 1988, 473–486.

[EGS] Edelsbrunner, H., Guibas, L. J., and Stolfi, J., Optimal point location in a monotone subdivision, DEC System Research Report, October, 1984.

[Fo] Forrest, A. R., Invited talk on computational geometry and software engineering, ACM Symposium on Computational Geometry, Yorktown Heights, NY, June, 1986.

[FM] Fournier, A., and Montuno, D. Y., Triangulating simple polygons and equivalent problems, *ACM Transactions on Graphics*, **3**, 1984, 153–174.

[GY] Graham, R. L., and Yao, F. F., Finding the convex hull of a simple polygon, *Journal of Algorithms*, **4**, 1983, 324–331.

[HMRT] Hoffman, K., Mehlhorn, K., Rosensthiehl, P., and Tarjan, R., Sorting Jordan sequences in linear time using level-linked search trees, *Information and Control*, **68**, 1986, 170–184.

[HK] Hopcroft, J., and Kraft, D., The challenge of robotics, in *Advances in Robotics 1: Algorithmic and Geometry Aspects of Robotics* (J. T. Schwartz and C. K. Yap, eds.), Erlbaum, Hillsdale, NJ, 1987, pp. 7–42.

[Km] Kim, M., private communication, November 17, 1987.

[Ki] Kirkpatrick, D., Optimal search in planar subdivisions, *SIAM Journal on Computing*, **12**, 1983, 28–35.

[Kn] Knuth, D., *Computers and Typesetting, Vol. C: The METAFONTbook*, Addison-Wesley, Reading, MA, 1986.

[LP] Lee, D. T., and Preparata, F., An optimal algorithm for finding the kernel of a polygon, *Journal of the Association for Computing Machinery*, **26**, 1979, 415–421.

[LT1] Lipton, R., and Tarjan, R., A separator theorem for planar graphs, *SIAM Journal of Applied Mathematics*, **36**, 1979, 177–189. A preliminary version of this paper was presented at the Waterloo Conference on Theoretical Computer Science, Waterloo, Ontario, August, 1977.

[LT2] Lipton, R., and Tarjan, R., Applications of a planar separator theorem, *Proceedings of the IEEE FOCS Conference*, Providence, RI, October 1977, pp. 162–170.

[Me] Mehlhorn, K., *Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.

[Pa] Pavlidis, T., Curve fitting with conic splines, *ACM Transactions on Graphics*, **2**, 1985, 1–31.

[Pr] Pratt, V., Techniques for conic splines, *Computer Graphics*, **19**, 1985, 151–159.

[PS] Preparata, F., and Supowit, K., Testing a simple polygon for monotonicity, *Information Processing Letters*, **12**, 1981, 161–164.

[Re] Requicha, A., Representations for rigid solids: theory, methods and systems, *Computing Surveys*, **12**, 1980, 437–464.

[ST] Sarnak, N., and Tarjan, R. E., Planar point location using persistent search trees, *Communications of the ACM*, **29**, 1986, 669–679.

[SV] Schäffer, A. A., and Van Wyk, C. J., Convex hulls of piecewise-smooth Jordan curves, *Journal of Algorithms*, **8**, 1987, 66–94.

[Sh] Shamos, M., Geometric complexity, *Proceedings of the ACM Symposium on Theory of Computing*, Albuquerque, NM, May, 1975, pp. 224–233.

[Sm] Smith, A. R., Invited talk on the complexity of images in the movies, ACM Symposium on Computational Geometry, Yorktown Heights, NY, June, 1986.

[So] Souvaine, D. L., Computational Geometry in a Curved World, Ph.D. Thesis, Princeton University, October, 1986.

[TV] Tarjan, R. E., and Van Wyk, C. J., An $O(n \log \log n)$-time algorithm for triangulating simple polygons, *SIAM Journal of Computing*, **17**, 1988, 143–178.