COMPUTER GRAPHICS AND IMAGE PROCESSING **19**, 303–334 (1982)

Determining the Spatial Containment of a Point in General Polyhedra

YEHUDA E. KALAY

CAD–Graphics Laboratory,
Institute of Building Sciences,
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Received July 21, 1981; revised September 21, 1981

Determining the inclusion of a point in volume-enclosing polyhedra (shapes) in 3D space is, in principle, the extension of the well-known problem of determining the inclusion of a point in a polygon in 2D space. However, the extra degree of freedom makes 3D point–polyhedron containment analysis much more difficult to solve than the 2D point–polygon problem, mainly because of the nonsequential ordering of the shape elements, which requires global shape data to be applied for resolving special cases. Two general $O(n)$ algorithms for solving the problem by reducing the 3D case into the solvable 2D case are presented. The first algorithm, denoted "the projection method," is applicable to any planar-faced polyhedron, reducing the dimensionality by employing parallel projection to generate planar images of the shape faces, together with an image of the point being tested for inclusion. The containment relationship of these images is used to increment a global parity-counter when appropriate, representing an abstraction for counting the intersections between the surface of the shape and a halfline extending from the point to infinity. An "inside" relationship is established when the parity-count is odd. Special cases (coincidence of the halfline with edges or vertices of the shape) are resolved by eliminating the coincidental elements and re-projecting the merged faces. The second algorithm, denoted "the intersection method," is applicable to any well-formed shape, including curved-surfaced ones. It reduces the dimensionality by intersecting the shape with a plane which includes the point tested for inclusion, thereby generating a 2D polygonal trace of the shape surface at the plane of intersection, which is tested for containing the trace of the point in the plane, directly establishing the overall 3D containment relationship. A particular $O(n)$ implementation of the 2D point-in-polygon inclusion algorithm, which is used for solving the problem once reduced in dimensionality, is also presented. The presentation is complemented by discussions of the problems associated with point–polyhedron relationship determination in general, and comparative analysis of the two particular algorithms presented.

1. INTRODUCTION

Determining whether a point in 3D space is "inside" or "outside" a volume-enclosing shape is, in principle, an extension to testing a point in 2D space for inclusion in a planar polygon. This problem occurs frequently in many applications such as VLSI circuit fabrication, computer vision and pattern recognition systems, computer aided design and other areas involving computational geometry.

Solutions to the 2D problem are well documented, and many implementations exist (see, for example, [1–6]). However, the extra degree of freedom makes establishing the 3D point–polyhedron containment relationship much more difficult than the 2D problem. A solution to a restricted class of shapes, namely determining the spatial inclusion of a point in convex, planar-faced polyhedra, was given by Lee and Preparata in [3].

0146-664X/82/080303-32$02.00/0
Copyright © 1982 by Academic Press, Inc.
All rights of reproduction in any form reserved.

This paper presents two general $O(n)$ solutions to the point–polyhedron problem. The first is applicable to any planar-faced polyhedra (including concave ones), and the applicability range of the second is extended over curved-surfaced shapes. The second algorithm is intuitively simpler than the first but more complicated to implement. Both solutions are based on reducing the dimensionality of the problem to the solvable 2D case. The first solution (denoted "the projection method") achieves that reduction by *projecting* the faces of the shape onto a plane, together with the point being tested for containment, yielding a 2D image consisting of polygons and a point. However, unlike the algorithm presented by Lee and Preparata which uses a projection to identify the horizontal location of the point relative to the shape (later to be complemented by a vertical relationship identification), the algorithm presented here uses projection only to facilitate a parity-count test based on the intersections between a halfline extending from the test point and the surface of the shape, which readily determines the overall containment relationship of the point and the shape. The second solution (denoted "the intersection method") reduces the dimensionality by intersecting the shape with a plane which is chosen to be coincidental with the point being tested for containment, yielding a polygonal trace of the shape surface and the point in the plane, to be processed by a point–polygon inclusion test. The additional information inherent in the third dimension is maintained and utilized in both methods to solve singularity cases. For the sake of completeness, the particular implementation of an $O(n)$ 2D point–polygon containment test, which is used to solve the problem once its dimensionality has been reduced, is also presented.

This presentation is organized as follows: the underlying definitions which are used throughout the paper are given in Section 2, then the general approaches for establishing 2D containment relationship are briefly discussed in Section 3, followed by a particular implementation in Section 4. The problems associated with the 3D case are discussed in Section 5, which also describes a direct, simplistic algorithm for their resolution. The projection and intersection methods are presented in Sections 6 and 7. A comparative analysis and discussion of the two methods concludes the presentation.

2. DEFINITIONS AND FUNDAMENTAL NOTIONS

It is useful to begin with recalling some standard geometric definitions and introducing the notions which will be used throughout the paper.[1]

DEFINITION 1. A *polygon* (Fig. 2.1) is the figure formed by choosing a sequence of n arbitrary points $A_1, \ldots, A_n$, joining each point with the next by a line segment, and joining the last point with the first one. The points $A_1, \ldots, A_n$ are the *vertices* of the polygon, and the segments $A_1A_2, \ldots, A_nA_1$ are its *sides* (a vertex bounds exactly two sides).

This definition, introduced by Poinsot, is general, since it does not imply planarity and simplicity of the figure. Neither does it imply that the sides are segments of straight lines. A polygon is said to be *planar* if all its sides lie in one plane, otherwise it is said to be *skew*. If sides intersect only at common bounding vertices, then it is

[1] Geometric definitions are given in many books. The ones used for the purposes of this presentation are *Fundamentals of Mathematics*, volume II (H. Behnke, F. Bachmann, K. Fladt and H. Kunle, Eds.), MIT Press, Cambridge, Mass., 1974; and *Graphs, Surfaces and Homology*, by P. J. Giblin, Chapman & Hall Mathematical series, Halsted Press, New York, 1977.

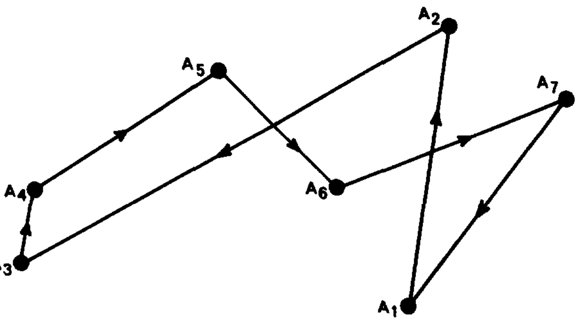

FIG. 2.1. A general polygon.

said to be *simple*. It is said to be *straight* if all its sides are segments of straight lines. Note however, that the definition of a polygon does imply an *ordering of the sides*, fixed by the sequence of joining the vertices in A.

DEFINITION 2. A *shape* is a system of *simple* polygons with the property that every side of each polygon is also a side of exactly one other polygon in the system. The polygons are called the *faces* of the shape, their sides are its *edges*, and their vertices are its *vertices*. The system as a whole is called the *surface* of the shape.

A corollary of Definition 2 is that every edge is bounded by exactly two vertices, and belongs to exactly two faces. The definition implies closedness of the surface, but does not imply it connectivity and orientability; neither does it require that the faces be planar. *Connectivity* means that there exists a path on the surface from any vertex to any other vertex of the shape. Not imposing this requirement means that a shape could be composed of multiple *disjoint shells*, each consisting of a *connected* set of faces. However, to be able to establish a containment relationship of a point in space and the shape, it is necessary to add the *orientability* condition, so that each shell of the shape will partition the 3D space into exactly two disjoint domains, exactly one of which is unbounded, commonly denoted "outside," and the other is bounded and denoted "inside." Orientability of a surface can be defined in different ways, for instance, by the *Möbius Edge Rule*. Consider an edge common to two adjacent faces. The orientation of each of these faces (being simple polygons) determines an order for the two endpoints (vertices) of the edge. If these two orders are *opposite* to each other, the faces are coherently oriented. For shapes this means that the normals to all their faces point to the same domain of the 3D space, as partitioned by their surface. Shapes whose surfaces are made of connected and orientable shells, and the points any pair of their faces share are only those of their common edge or vertices (that is, they do not self-intersect), are said to be *well formed*. All the discussions throughout this paper will assume well-formedness of the shapes.[2]

It is important to note that well-formedness is a *global* characteristic of the shape. It cannot be determined with local data alone, as encoded in the definitions of the edges and faces (namely, that each edge is bound by exactly two vertices, and is common to exactly two faces) because of the non-self-intersection requirement.

[2]For a more complete discussion of orientability of surfaces see Chapters 2 and 3 in [7]. For a discussion of well-formedness in general see [8].

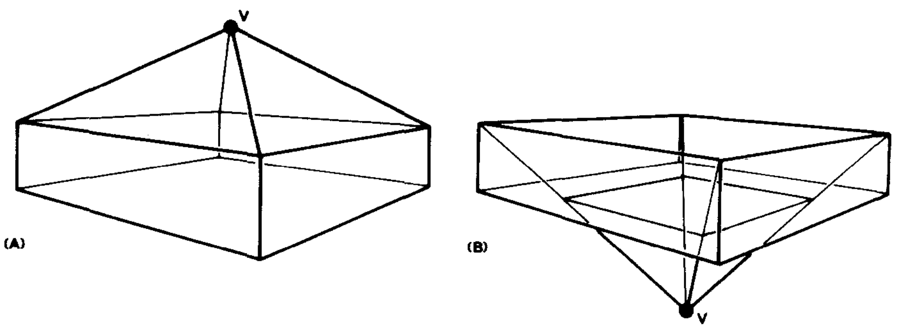

FIG. 2.2. Globality of the self-intersection condition.

Figure 2.2 shows why: topologically shapes *A* and *B* are the same, but geometrically they differ in the location of vertex *V* relative to the rest of the shape, making *A* a well-formed shape and *B* an ill-formed one.

Simple polygons also possess well-formedness characteristics (in terms of connectivity, orientability and non-self-intersection). However, for the purposes of this presentation it was found useful to relax the requirement of simplicity and to extend the definition of "inside" for polygons. First, a definition of another term is required:

DEFINITION 3. The *wrap number* of a point is a measure of the number of times the point is encircled by the set of ordered sides which make up the polygon. This number is positive or negative, depending on the sense of the cyclic order of the sides around the point (clockwise or counter-clockwise).

The "inside" and "outside" relationships of a point in a polygon can now be defined as follows:

DEFINITION 4. A point is *inside* a polygon if its wrap-number is odd, and it is *outside* otherwise.

Thus, points *A*, *B*, and *C* in Fig. 2.3 are considered to be "inside" the polygon, regardless of side orientation, while *D* is considered to be "outside." The rational for this relaxation will become clear when the projection method is discussed in Section 6. Note, however, that such a definition does not conflict with any of the traditional methods for establishing the containment relationship of a 2D point and a planar polygon, as discussed in the following section.

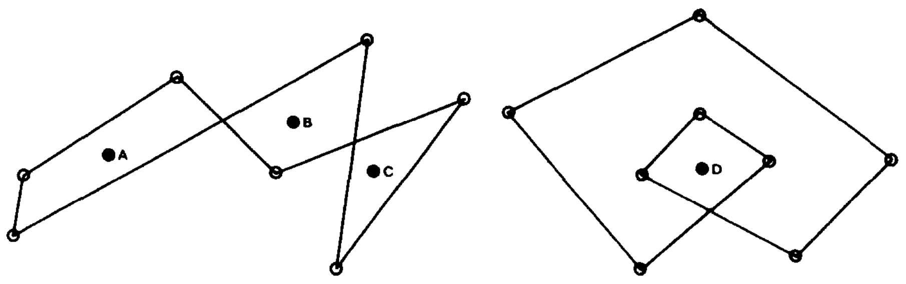

FIG. 2.3. Extension of "inside" definition for polygons.

3. CONTAINMENT RELATIONSHIPS IN 2D SPACE

Following Definition 4, establishing whether a point lies inside or outside a planar polygon is the determination of its wrap-number. This being an important problem in many applications, several algorithms for solving it have been developed, based on the following three methods [6]:

1. If the polygon is known to be *convex*,[3] the coordinates of the point can be substituted in the line equation of each side. If the signs of all such substitutions are the same, the point is "inside" the polygon, otherwise it is "outside" (given that the coefficients in all the line equations are chosen consistently). Lee and Preparata have extended this method to include any planar straight-line graph, by replacing the side-equation substitution test with a partitioning of the graph (and the 2D space) into regions, and determining the region which includes the test-point by using the "chain method" [3].

2. However, for nonconvex planar polygons, the above method is inadequate and two other methods may be used. One is based on determining the parity of the number of intersections between the polygon and a halfline extending from the point to infinity. The point is "outside" if the parity is even, and it is "inside" if the parity is odd. Such an approach has been implemented, for example, by Barton, using a binary searchable polygonal representation of the boundary lines representing geographic regions [5].

3. The other method usable for nonconvex polygons is based on computing the sum of the angles subtended by each of the sides of the polygon as seen from the test point, and traversed in a consistent direction. This sum is always an integer multiple of 2π, the multiple being, in fact, the wrap-number of the point, thus establishing an "outside" point/polygon relationship if it is even. Figures 3.1 and 3.2 depict these two methods.

For straight polygons, practical implementation of neither of these last two methods needs to involve actual numerical calculations (that is, calculating the points of intersection for the parity-count method or the angles for the summation method). A nonnumerical implementation of the parity-count method is described in the following section. The wrap-number in the angle-summation method can be found, for instance, by partitioning the 2D space into four quadrants about the point, and identifying sequentially the location and slope of the polygon sides in them.

The particular implementation of these methods depends on various parameters for efficiency purposes, most important of which are the relative numbers of points and polygons to be tested, and the number of vertices in each polygon. The number of vertices in the polygon determines the profitability of using intelligent, complex algorithms rather than brute-force methods, and the number of points and polygons divides implementations into three classes:

1. The *single-point problem*, in which a single point is tested for inclusion in a single polygon.

[3]A polygon is said to be convex if any straight line intersects it in at most one continuous segment of that line.

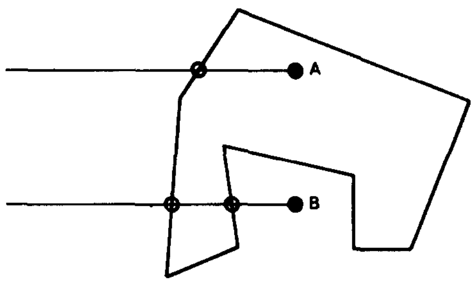

FIG. 3.1. The parity count method.

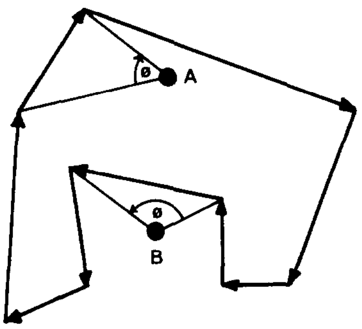

FIG. 3.2. The angle summation method.

2. The *multiple-point problem* ("range problem"), in which multiple points are tested for inclusion in the same polygon (as, for example, in the case of scan-converting a polygon for raster graphics, or computer-assisted legislative redistricting applications), in which case, as was shown by several researchers (see, for example, [2, 9]), some investment in preprocessing may considerably increase the overall efficiency.[4]

3. The *multiple-polygon problem*, in which a single point is tested for inclusion in multiple polygons. This is similar, in principle, to the single point problem except that the overall efficiency can be improved by making use of the global properties of the structure, such as the possibility to "envelope" several polygons into a single one to be tested for containing the point, or use Lee and Preparata's "chain method" to partition the 2D space into disjoint regions. This case will be further discussed in Section 6, as it is the predominant class when solving the singularities of the projection method.

The particular implementation chosen for the purposes of the algorithms in this presentation is discussed in the following section. It is based on the parity-count method and is extended to accommodate nonsimple polygons, namely, self-intersecting polygons, and polygons with coincidental edges and vertices.

[4]This class can be further subdivided into *interactive* and *batch* modes, depending on whether all the points to be tested are known in advance or they are presented to the algorithm one at a time. Batch mode can emulate the preprocessing in the actual containment establishing test.

4. THE 2D CONTAINMENT RELATIONSHIP (CR) ALGORITHM

As stated earlier, the parity of the intersections between the test-line (TLN) and the boundary of the polygon can be established without actually calculating the points of intersection, since it is the existence of these intersections which is of consequence rather than their actual location. Establishing their existence can be done by determining the location of each vertex relative to the test-point (TP), and then the relationship between the two endpoints of each side: Intersection exists when the endpoints lie on opposite sides of the TLN. Assuming a horizontal TLN, this means that the vertices lie "above" and "below" it. However, since the TLN is only a halfline, assuming it extends leftward from the TP, then sides of the polygon which lie completely to the right of the TP do not intersect it even if their endpoints lie on its opposite sides. Thus classifying the relative locations of vertices as "above", "below" or to the "right" of the TP, the 2D space is partitioned into 3 regions as depicted by Fig. 4.1.

This partitioning is, however, insufficient, since the correctness of the parity-count method is impaired by two kinds of singularities:

1. When one or more vertices of the polygon lie exactly on the TLN.
2. When one or more sides of the polygon are collinear with the TLN.

Misinterpretation of either of these cases will result in a wrong parity-count, turning the polygon "inside-out," in the sense that exterior points will be classified as being inside, and vice-versa (Fig. 4.2).

Singularities can either be avoided, or they can be solved directly. Avoiding singularities can be done by selecting a different TLN which does not coincide with any of the vertices, or by displacing the coincidental vertices in some predetermined, consistent manner, to remove the coincidence. A different TLN can be selected directly, or the polygon can be rotated about the TP (a method which has some

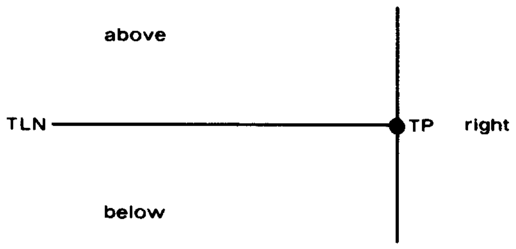

FIG. 4.1. Basic partitioning of the 2D space by the TP.

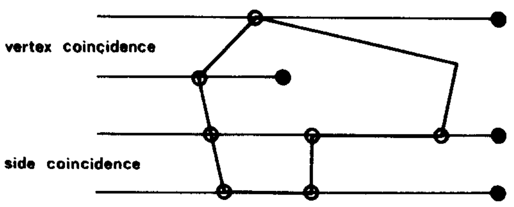

FIG. 4.2. Singularity conditions in the 2D case.

computational attractiveness). In either case, however, several iterations may be required until a nonsingular TLN is found, particularly if the polygon contains many vertices and concavities. Displacing coincidental vertices results in a temporary distortion of the polygon, and is very sensitive to numerical imprecisions and round-off errors, producing satisfactory results only if the vertices are represented using integer arithmetic (a method which is effectively used in polygon scan-conversion algorithms for raster displays, as described in [10]).

Solving singularities directly is relatively simple in the 2D case. It is predicated on the observation that "legal" intersections (representing a transition of the TLN from an internal to an external region of the 2D space as partitioned by the polygon) occur when the sides converging on the coincidental vertex (or side) lie on different sides of the TLN (above and below), whereas "illegal" intersections (representing a tangency of the TLN to the polygon) occur when the converging sides both lie on the same side of the TLN. This could also be detected by observing the directions of the successive sides at the singularity vertex: If their directions are monotonically upward or downward, the intersection is legal, otherwise it is not.

The algorithm presented here implements the convergence method, namely, identification of the relative locations of the preceding and succeeding vertices to the coincidental element. To do this, the basic partitioning of the 2D space presented earlier is refined by adding two more regions and sub-partitioning the "right" region: the added regions are the set of all points which are coincidental with the TLN except the TP itself, and the TP as a separate region. The "right" region is subpartitioned into "right-above," "right-line," and "right-below," with respect to the vertical coordinate of the TP (Fig. 4.3).

More formally, the 2D space is partitioned into the following seven regions, assuming TP $\leftarrow (u_{tp}, w_{tp})$ is the test-point, and $P \leftarrow (u, w)$ is a point in the 2D space:

1. $TP \equiv \{P_{u,w} \mid (u = u_{tp}) \wedge (w = w_{tp})\}$,
2. $AA \equiv \{\forall P_{u,w} \mid (u \geq u_{tp}) \wedge (w > w_{tp})\}$,
3. $TL \equiv \{\forall P_{u,w} \mid (u > u_{tp}) \wedge (w = w_{tp})\}$,
4. $BB \equiv \{\forall P_{u,w} \mid (u \geq u_{tp}) \wedge (w < w_{tp})\}$,
5. $RA \equiv \{\forall P_{u,w} \mid (u < u_{tp}) \wedge (w > w_{tp})\}$,
6. $RL \equiv \{\forall P_{u,w} \mid (u < u_{tp}) \wedge (w = w_{tp})\}$,
7. $RB \equiv \{\forall P_{u,w} \mid (u < u_{tp}) \wedge (w < w_{tp})\}$.

It can be easily verified that this partitioning includes all the points in the 2D space without overlaps. The notation of the sets was chosen such that the first letter

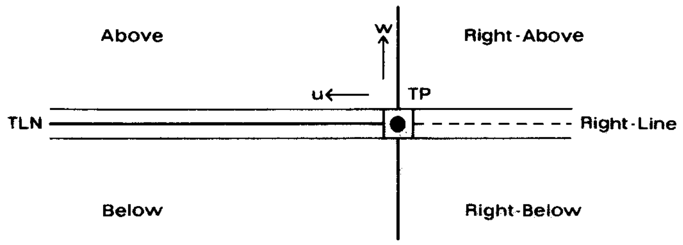

FIG. 4.3. Complete partitioning of the 2D space.

	TP	TL	RL	RB	RA	BB	AA
AA	5	4	1	3	1	2	1
BB	5	4	1	1	3	1	
RA	5	1	1	1	1		
RB	5	1	1	1			
RL	5	6	1				
TL	5	4					
TP	5						

FIG. 4.4. Endpoint relationships and side classification.

denotes *primary* region belonging (T, A, B, or R), and the second letter denotes *secondary* region belonging (P, L, A or B). This notation will be refered to later as the "color" of the vertices.

Depending on the relative locations of its two endpoints, a side of the polygon can be located in any one of these regions, or straddle across two or more of them. These locations determine whether it intersects the TLN, and in some cases, the overall result of the algorithm, such as when the TP is coincidental with a side or a vertex of the polygon. A complete enumeration of the relative locations of the two endpoints in the seven regions yields $C(\frac{2}{7}) + 7 = 28$ combinations, consisting of six different significant cases which are relevant to the containment analysis (Figure 4.4).

Case 1 has sides not intersecting the TLN, while Case 2 has sides which do.[5] Cases 3 and 4 identify singularities of two kinds, requiring further tests for complete resolution. Sides of Case 5 have at least one vertex which is coincidental with the TP, establishing its relationship with the polygon as "on-vertex." Likewise, sides of Case 6 establish its relationship as "on-side."

4.1. Solving the 2D singularities

Cases 3 and 4 divide the singularities into two groups, depending on the action to be taken for their resolution. Case 3 singularities involve diagonally opposite endpoint relationships (*AA* and *RB*, or *BB* and *RA*, see Fig. 4.3), and are denoted "simple singularities". They may be resolved into one of three possible side classifications, depending on their slope and position relative to the TP (Fig. 4.5):

- No intersection (Figs. 4.5a, d).
- Coincidence with the TP, in which case the algorithm yields *side-coincidence* resolution (Figs. 4.5b, e).
- Intersection, in which case the parity-counter is incremented (Figs. 4.5c, f).

Slope comparisons, shown in Fig. 4.6, determine which one of the possible relative positions is assumed by the side:

- $|a/b| > |c/d|$ —the side intersects the TLN.
- $|a/b| = |c/d|$ —the side is coincidental with the TP.
- $|a/b| < |c/d|$ —no intersection.

[5]Having chosen the partitioning of the 2D space as presented above, Case 2 includes a vertical side which may be coincidental with the TP. Another test must be performed to completely determine its classification.

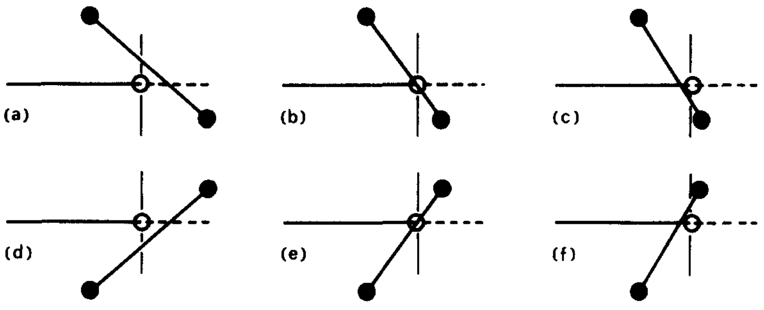

FIG. 4.5. Simple singularity cases.

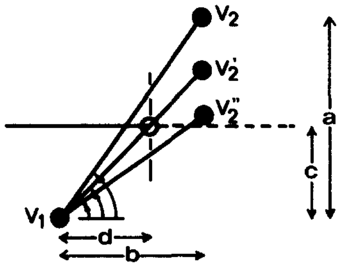

FIG. 4.6. Resolving simple singularities by slope comparisons.

Case 4 singularities, denoted "complex singularities," involve at least one polygon vertex which is coincidental with the TLN. They are resolved by identifying the first preceding and succeeding vertices which are not on the TLN, and comparing their Secondary Region Belongings (colors): if these colors are the same (both A or both B), there is no intersection; otherwise (if the colors are A and B), intersection exists. For the sake of simplicity the preceding vertex is guaranteed not to coincide with the TLN by starting the traversal with a non-coincidental vertex.[6] The first noncoincidental successor is found by repeatedly testing intermediate vertices, disregarding those which coincide with the TLN, until a noncoincidental vertex is encountered. Thus no distinction needs to be made between coincidental sides and vertices, or multiple consecutives side coincidences (however, care must be taken not to overlook a successor side which coincides with the TP, yielding a *side-coincidence* resolution).

4.2. The 2D CR Algorithm

The algorithm for determining the 2D containment relationship consists of traversing the vertices of the polygon in the order implied by their joining sides, determining the relative location of each consecutive pair by comparing each vertex coordinate to that of the TP, and incrementing the parity counter as required.

Let $P = (A_1,\dots,A_n)$ denote a straight, planar polygon, with vertices $A_i = (u_i, w_i)$, $i = 1,\dots,n+1$, such that $A_{n+1} = A_1$. Let $P_t = (u_t, w_t)$ be a test point the relative

[6]Since nonsimple polygons are allowed, a degenerate polygon, all whose vertices are coincidental with the TLN, may occur. In that case more elaborate starting and terminating conditions must be employed, and the containment relationship will be determined by the relative horizontal location of the TP and the polygon, or its coincidence with a vertex.

location of which, with respect to P, is to be determined. Let C denote the parity counter.

1. [*Initialize traversal vertices and parity counter.*] Set $i \leftarrow 1$; $j \leftarrow 2$; $C \leftarrow 0$. Determine the Primary and Secondary colors of A_i relative to P_t (A_i is a vertex not coincidental with the TLN).
2. [*Determine relationship of vertices.*] Determine the color of A_j. Determine the relationship r between A_i and A_j (r is an integer, $r \in [1:6]$), based on the colors of A_i and A_j.
3. [*No intersection case.*] If $r = 1$ then go to step 9.
4. [*Intersection case.*] If $r = 2$ then:
    a. [*Test vertical side coincidence.*] If A_i, A_j and P_t are vertically collinear, then P_t is coincidental with the side of P bounded by A_i and A_j. Go to step 11.
    b. [*Increment parity count.*] $C \leftarrow C + 1$; then go to step 9.
5. [*Simple singularity case.*] If $r = 3$ then determine the slopes S_a of A_iA_j and S_b of A_iP_t:
    a. [*Side coincidence.*] If $S_a = S_b$ then P_t is coincidental with the side of P bounded by A_i and A_j. Go to step 11.
    b. [*Intersection.*] If $S_a > S_b$ then (increment parity counter) $C \leftarrow C + 1$; then go to step 9.
    c. [*No intersection.*] Go to step 9.
6. [*Complex Singularity case.*] If $r = 4$ then resolve complex-singularity case:
    a. [*Set successor.*] Set $k \leftarrow j + 1$, and determine the color of A_k relative to P_t.
    b. [*Succeeding side coincides with PT.*] If A_jA_k is coincidental with P_t then P_t is coincidental with the side of P bounded by A_j and A_k. Go to step 11.
    c. [*Successor vertex coincides with PT.*] If A_k is coincidental with P_t then *halt*: Vertex coincidence is reported.
    d. [*Successor vertex is on TLN.*] If A_k is coincidental with the TLN then set $j \leftarrow j + 1$; then go back to step 6[a].
    e. [*Compare successor and predecessor vertices' Secondary colors.*] If the Secondary colors of A_i and A_k are the same (both A or both B) then (no intersection); otherwise (intersection) increment parity counter $C \leftarrow C + 1$.
    f. [*Traversal step.*] Set $j \leftarrow k$, then go to step 9.
7. [*Vertex coincidence case.*] If $r = 5$ then P_t is coincidental with P vertex A_i or A_j. Determine which one, then go to step 11.
8. [*Side coincidence case.*] P_t is coincidental with the side of P bounded by A_i and A_j. Go to step 11.
9. [*General termination and traversal step.*] If $i < n$ then: $i \leftarrow j$; $j \leftarrow j + 1$. Go to step 2.
10. [*Determine parity.*] If (the parity counter) C is ODD then P_t is *inside* P, otherwise it is *outside* P.
11. [*Exit.*] *halt.*

The above presentation of the 2D CR algorithm describes its main structure only. It does not cover all the possible cases, such as the case of a degenerate polygon, all whose vertices coincide with the TLN. Such cases require some programming elaboration in defining the general termination condition, which are easy to handle but introduce notation irrelevant to the comprehension of the algorithm itself. Likewise, the determination of the relative region belonging (color) of the vertices is a simple programming task.

4.3. *Analysis and Discussion*

The complexity of the 2D CR algorithm is dominated by the number of vertices n, establishing a time complexity $O(n)$ both for worst and expected cases (assuming side and vertex coincidences of the TP are statistically negligable). *Minmax* boxing, if maintained by the environment in which the algorithm is embedded, would reduce that to a lower bound constant 1 for trivially rejectable cases (where the point is far outside the polygon).

Since no particular data structure is required (except for storing the input polygon itself, the cost of which for most practical applications is assumed to be carried by the environment embedding the algorithm), the space complexity is a constant 3, since at any time the algorithm is concerned with three vertices at most.

Handling assemblies of multiple subjoint or disjoint polygons, such as in the case of a polygon with "holes" in it, can be done by calling the algorithm repeatedly, once for each polygon, treating the parity counter as a global variable.

It should be noted that the algorithm presented here delivers more information than a simple "inside" or "outside" answer, explicitly identifying the coincidental side or vertex if the TP coincides with one. This, as will be shown later, is used to resolve some of the singularities of the 3D case. However, for applications where such coincidences can a priori be resolved as "outside" or as "inside," the algorithm could be collapsed into a more compact representation.

5. CONTAINMENT RELATIONSHIP IN 3D SPACE

Extrapolating from the 2D case, establishing the containment relationship between a point and a volume enclosing, well-formed shape in 3D space could be accomplished by determining the parity of the number of intersections between the surface of the shape and a halfline (TLN) extending from the test point to infinity (Fig. 5.1).

As in the 2D case, the parity count will be impaired by singularities which arise in the 3D case when a shape element (vertex, edge or face) is coincidental with the TLN. Such a coincidence may resolve into one or two different cases: (1) it may be a legal intersection, in the sense that it represents a transition from an internal region of the 3D space to the external one (or vice versa), as partitioned by the surface of the shape, and therefore it should be counted for parity (Fig. 5.2a), or (2) it may be tangency condition, in which case the surface of the shape is not penetrated and no inside/outside transition occurs, therefore it should not be counted for parity (Fig. 5.2b). Misinterpretation of such singularities may result in a wrong parity count,

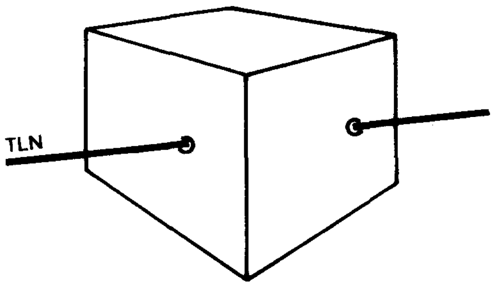

FIG. 5.1. Extending the parity-count method to the 3D case.

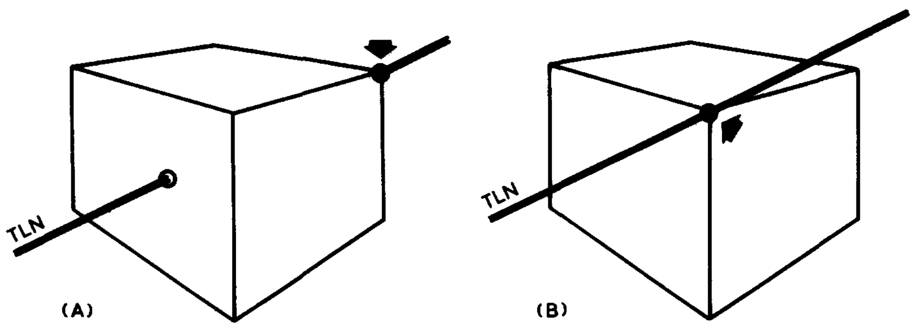

FIG. 5.2. Singularity conditions in the 3D case.

which, as in the 2D case, will turn the shape "inside-out" such that points which are inside it will be considered outside, and vice versa.

As in the 2D case, singularities can either be avoided, or they can be resolved directly (determined whether or not they should be counted for parity).

5.1. Avoiding the Singularities

While singularities could be avoided in the 2D case, avoiding them in the 3D case has undesirable side effects which have to be solved at a cost which is possibly higher than that of solving the singularities themselves. Singularities can be avoided in the following ways, which are an extension to the 2D case:

1. Use another TLN which does not create singularities, by avoiding coincidence with vertices and edges of the shape, or tangency with its faces.

2. Displace the shape elements which coincide with the TLN in some predetermined manner such that their coincidence will disappear.

Determining how to pick a different TLN which does not create singularities may not be a trivial task if the shape is complicated, that is, has many elements, and thus the likelihood of coincidences is great. Moreover, singularity may not be encountered until after many of the elements have been tested, which in the worst case may result in

$$(n_1V + n_2E + n_3F)^2 \cdot 1$$

tests (where V is the number of vertices, E is the number of edges, and F is the number of faces. n_1, n_2, n_3 are coefficients denoting the number of coincidences with each kind of element). This means that a singularity is detected only when the last element is tested, and this happens for every kind of element of the shape n times.

A predetermined displacement (in terms of direction and magnitude) of the coincident elements, which results in a (temporary) distortion of the shape elements(s) involved, may not always remove the coincidence condition: for instance, if the displacement is a predetermined $+0.5$ increment in direction Z, then a TLN which coincides with an upper vertex of a cube may become coincidental with an edge of that cube which converges on the displaced vertex (Fig. 5.3). Moreover, such a

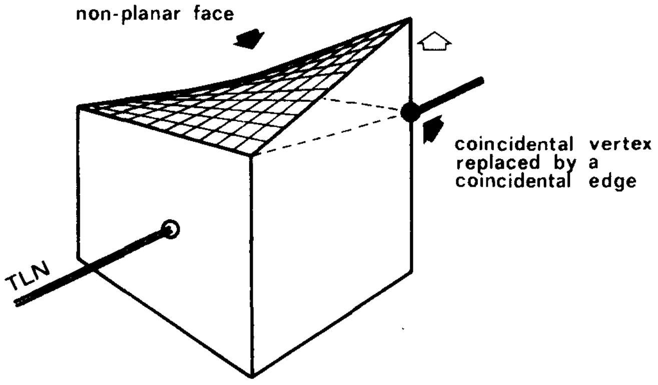

FIG. 5.3. Avoiding singularities by displacement.

displacement may render faces nonplanar, hindering the possibility of using simple means for determining whether or not the face is interested by the TLN. Rather it will involve the complexity of a containment relationship analysis between a point and a polygon on a curved surface. Round-off errors and numerical imprecisions may render such a method impractical for implementation.

5.2. *Resolving the Singularities*

3D singularities differ from 2D singularities in more than just the complexities resulting from the additional degree of freedom. Establishing the containment relationship between a point and a polygon in the 2D case was made possible due to (1) finding a common representation for both participating elements (the test point and the polygon), and (2) the sequential nature of the ordering of the polygon elements (sides and vertices). The common representation was achieved by increasing the dimensionality of the test point from 0 to 1 by making it into a test-line (TLN), bringing about the additional effect of partitioning the 2D space into regions, and by considering each side of the polygon as a line-segment which could be tested for relationship with the TLN.[7] Sequential ordering made it possible to resolve singularities based on local data alone, considering only the relationship of succeeding and preceding elements to the coincidental vertices.

When considering the containment relationship between a point and a shape in 3D space, the gap between the representations of the participating elements is widened by the additional degree of freedom; and there is no particular sequence in which the faces of the shape are ordered. The problem is, therefore, twofold: (1) identify the elements of the shape which are relevant for the resolution of a particular singularity case, and (2) find a representation which will capture their relative relationship in a way that can be uniquely interpreted. These two problems are not separable: unless a suitable representation is used, it is not possible to determine which of the elements are relevant for resolving the singularity. Therefore finding the correct representation is the predominant task before the singularities can be resolved.

[7]Or, considering the dual representation, the polygon sides were considered as pairs of points whose relationship with the TP determined the existence or absence of an intersection.

5.3. A Direct Approach Algorithm

Identifying the shape elements (faces) relevant for resolving the singularity can be based on adjacency to the coincidental element. It is the unique interpretation of their relative orientations which is more of a problem. A computationally attractive solution to this problem can be based on the *normals* to the faces, which uniquely identify their orientations. Considering the orientations of the normals to *all* the faces adjacent to the coincidental element, the following rule can be used to determine the nature of the singularity:

If the senses of the vector components of the normals on the TLN of all the faces which are directly adjacent to the coincidental element are the same, then the TLN penetrates the surface of the shape (the intersection should be considered for parity); otherwise it is only tangent to it.

The application of this rule is straightforward when the vector components on the TLN are nonzero, as shown in Fig. 5.4.

Complications arise when one or more of the faces are parallel to the TLN, their normal components on it being *null*. Examples of such singularity cases are depicted by Figs. 5.5a, b; in both cases an edge and two vertices are coincident with the TLN. The local conditions at each of the respective vertices are similar (though not the same): V_1^a and V_1^b have exactly the same local conditions, and from a local point of view, this is also true for V_2^a and V_2^b (F_2^a and F_2^b are tangent at a vertex to the TLN).

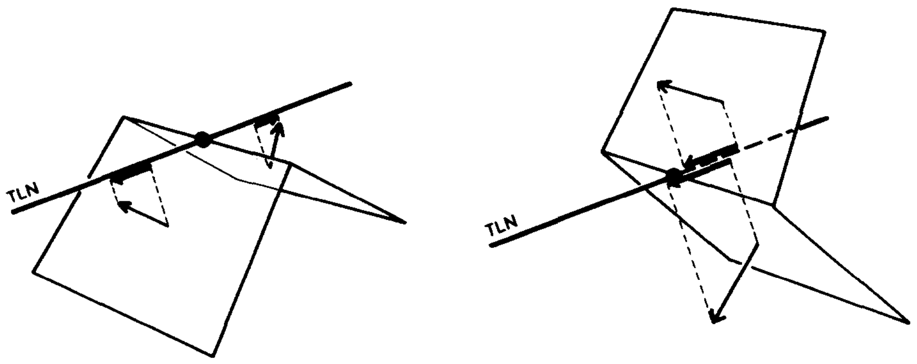

FIG. 5.4. Resolving singularities based on normal orientation.

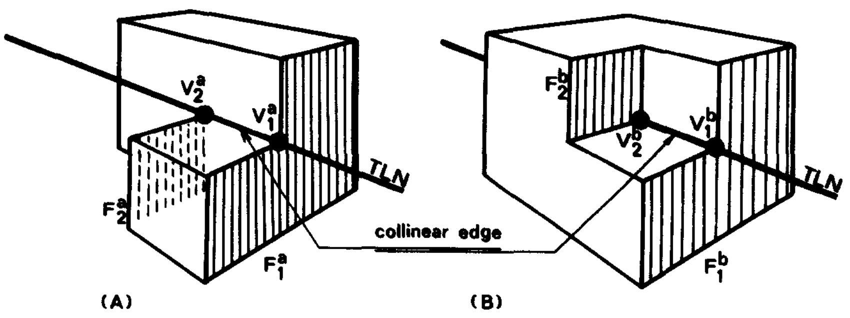

FIG. 5.5. Insufficiency of local data for solving singularities in 3D.

It is only the relative orientations of F_1 and F_2 in each shape that makes 5.5b a penetration and 5.5a a tangency.

It is necessary in such cases to incorporate the conditions prevailing at both endpoints into the resolution mechanism, thereby *expanding the locality* of the shape-data being used over formerly nonadjacent faces (which are, nevertheless, adjacent to the element which coincides with the TLN).

Embedding the above rule and the locality expansion principle in the 2D extrapolated parity-count method, an algorithm for establishing 3D containment relationship emerges, presented informally as follows:

1. Identify the point where the TLN intersects each face; this point may be inside, outside, or coincident with a vertex or an edge, or the TLN may be coplanar with the face.

2. If the point is inside or outside the face, no further processing is necessary; the parity-counter is incremented in case of "inside" relationship only.

3. If the TLN is coplanar with the face, then identify the faces adjacent to the currently tested one at its edges (or vertices) which coincide with the TLN, and consider their normal components on the TLN. If the senses of normal components are the same, then the TLN does intersect the surface of the shape and the parity counter is incremented. The faces which have thus been used in the locality expansion will not be tested again for intersection with the TLN, as their contribution to the parity count has been determined.

4. If the point is coincidental with an edge or a vertex of the face (but the TLN is not coplanar with it), then identify the faces adjacent to the coincidental element, and consider their normal components on the TLN. If their senses are the same, then the TLN does intersect the surface of the shape and the parity counter is incremented. The faces thus tested will not be tested again.

5. Having traversed and tested all the faces of the shape, if the parity of the counter is even then the point is outside the shape, and it is inside it otherwise.

Establishing the senses of the vector normal components on the TLN is facilitated by the rules of vector algebra which allow expressing any vector in 3D space in terms of its three cartesian components. Choosing the TLN parallel to the X axis, it is only the sense of the X component which is of consequence to the problem at hand, and it can be established by the sign of its direction cosine.

It is the establishing of the point of intersection between the face and the TLN which requires most of the computation and is most sensitive to numerical imprecision and round-off errors. It can nevertheless be accomplished by a two-step method: (1) determine the point of intersection between the TLN and the plane of which the face is part (the "face-plane"), then (2) determine the containment relationship between that point and the polygon defined by the face in that plane, using the 2D CR algorithm presented earlier.

Round-off errors, which increase as the face becomes closer to parallel to the TLN, can be eliminated and the computations made simpler (some of them completely omitted, such as determining the point of intersection between the TLN and the face-plane), if the *projected image* of the face on one of the major planes is used for 2D CR testing rather than the face-plane itself. However, such a projection has additional properties which can be exploited if a more global approach to the problem is taken, making homogeneous the representations of the point and the shape such that the resolution of the singularities can be done in a single logical framework, as will be discussed presently.

5.4. Homogeneous Representation of the Participating Elements

It has already been shown that the containment relationship can be established for a point and a planar polygon. Thus it should be possible to establish the containment relationship for a point and a shape in 3D space, if the dimensionality of the shape is reduced to 2D, while carrying the additional information generated by the third dimension in some different manner. Such a reduction in dimensionality can be achieved through two different methods, using the tools provided by affine and projective geometry:

1. By means of *projection* (a coincidence preserving mapping) that will generate a 2D image of the shape (or selected parts of it) on a plane.

2. By means of *intersecting* the shape with a plane, resulting in a 2D polygonal trace of the shape surface at the plane of the intersection.

Both reduction methods will be meaningful for solving the problem at hand only if the test point can also be incorporated in the resulting 2D representation in some unique manner. For the projection method this can be achieved by using the same mapping on the test point that was used on the shape, resulting in its image on the plane containing the 2D image of the shape (the image plane). The 3D relationship can then be established by means of the parity of the number of intersections between the surface of the shape and a TLN extending from the test point. The point is considered inside the shape if the parity is odd, and outside if it is even.

For the intersection method this can be done by selecting the intersection plane such that it includes the test point, so that the resulting 2D representation will contain the trace of the test point in addition to that of the intersecting shape surface, solvable directly by the 2D CR algorithm.

The major difference between the two methods is in that intersecting a shape with a plane produces a directly interpretable result, whereas the projection method may produce a number of overlapping polygons with complex relationships among themselves that need to be further processed before they can be interpreted for containment relationship. Thus this method has to be applied selectively and iteratively to each face of the shape, and the individual containment test results have to be combined into a single composite parity count that establishes the overall containment relationship between the point and the shape. This difference has direct bearings on the manner of handling singularities in every case, which also determines the environments in which each method can be used.

6. THE PROJECTION METHOD

A projection which is suitable for reducing the dimensionality and solving the representation and singularity problems is a *parallel* projection in the direction of the TLN, which maps the faces of the shape onto the image-plane (which is chosen to be perpendicular to the TLN). Since parallel projection is a special case of *singular affinities* (noninvertible, coincidence preserving, axial transformations), the definitions and theorems of affine geometry can be applied directly to prove that the 2D image of the projected shape elements forms an equivalence figure with the corresponding shape elements in 3D space, preserving the structural relationships (topology) between the elements.[8]

The invertibility which is destroyed by the reduction in dimensionality can be retained artificially by storing generator-image relationships for every element, thus converting the singular affinity into an *automorphism*, a one-to-one linear mapping between the shape and its 2D image.

In the simple case, when the TLN intersects a face of the shape without causing singularity conditions with any of its constituent elements (edges and vertices), the result of the projection (as implied by the nature of the affinity) is a polygon. This polygon, together with the 2D image of the test point, can be analysed by means of the 2D CR algorithm to establish whether or not the point lies inside the polygon, thus determining whether or not the global parity counter should be incremented (Fig. 6.1).

However, to resolve a singularity condition caused by coincidence of the TLN with a vertex or an edge of the shape, *all* the faces adjacent to the coincidental element have to be projected also, according to the locality expansion principle. Such a projection will produce a 2D image which consists of multiple polygons (a planar graph), all of which are tangent at a vertex (or an edge) to the 2D image of the test point (Fig. 6.2).

The complexity of such an image hinders its direct interpretation for containment analysis. However, the image could be interpreted if the *external contour* (envelope) of the participating polygons, which is itself a polygon, is considered for containment analysis rather than the individual face projections. The graph could be converted into such a composite polygon by *removing the sides common to the generating faces* in 3D space (and which also coincide with the TLN), thereby *merging* the faces into a single, possibly non-planar face, and re-projecting it to create a 2D image of a single polygon.

The conversion of the graph into the composite polygon can be done by iterating through a suitable projection-elimination algorithm for each face of the shape. However, a convention that will resolve some of its possibly ambiguous outcomes needs to be adopted:

CONVENTION 1. A point which is *on* the surface of the shape is considered to be *outside* it.

The informal description of the algorithm follows.

6.1. *The Projection–Elimination Algorithm*

1. A face is projected onto the image plane together with the test point.
2. The resulting polygon and point are tested for containment relationship by means of the 2D CR algorithm. This test will yield one of the following results:
   a. The point is inside the polygon.
   b. The point is outside the polygon.
   c. The point is coincidental with an edge of the polygon.
   d. The point is coincidental with a vertex of the polygon.

[8]Affine geometry is described in many books. This presentation relies on F. Flohr and F. Raith—Affine and Euclidean Geometry, and R. Lingenberg and A. Bauer—Affine and Projective Planes, in *Fundamentals of Mathematics*, volume II (H. Behnke, F. Bachmann, K. Fladt and H. Kunle, Eds.), MIT Press, Cambridge, Mass., 1974.

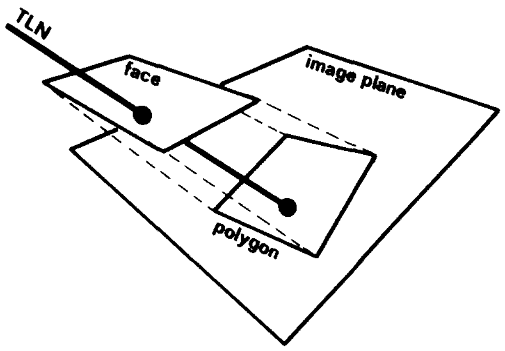

FIG. 6.1. Simple projection case.

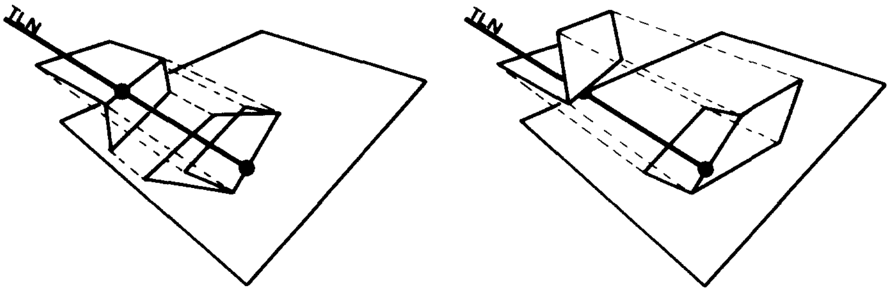

FIG. 6.2. Complex projection cases.

3. Cases (a) and (b) are readily interpretable, the global parity count being incremented for (a) only.
4. Cases (c) and (d) denote the existence of a singularity case; however, since the same coincidence-preserving projection was used to generate both the polygon and the point, their coincidence in the image-plane indicates that the TLN coincides with an edge (or a vertex) of the shape in 3D space. The automorphism (which is artificially maintained) is used to identify these corresponding 3D elements, and they are *removed* from the shape. For Case (c) an edge is removed, and for (d) the vertex and all the edges which converge on it are removed. This will cause *merging* of the faces which were adjacent to the removed elements into a single, possibly non-planar, face.
5. Having thus removed the cause of the singularity, the composite face is projected again (together with the test point), possibly resulting in a nonsimple polygon, which is the *contour* of the merged faces, and which can be tested for including the image of the TP by step 2 of this algorithm.

This iteration may be repeated several times until step 3 is encountered and the relationship is correctly established. The removal of edges and vertices, and therefore the merging of the faces, eliminates the singularity and expands its locality over formerly nonadjacent faces. Therefore it facilitates both the identification of the shape elements which are relevant for solving a particular singularity, and it brings global shape data to bear on the problem (Fig. 8.1 depicts such an iterative process). The faces used in the locality expansion process need not be tested again, as their contribution to the parity-count has been determined. A formal presentation of the algorithm now follows.

6.2. 3D Containment Relationship Algorithm: The PROJECTION METHOD

For clarity, the description of the algorithm is given on two levels: the outer level describes the main control structure of the *shape traversal and parity count determination*, while the inner level describes the actual *2D image generation* of a face (or a collection of faces), to be tested for including the 2D image of the TP. Trivial calculations such as establishing the existence of penetration between the plane of which the face is part (face-plane) and the TLN (which is a halfline), are omitted. The image plane is chosen to be the YZ plane of the coordinate system.

Let $S = (F_1, \ldots, F_n)$ denote a planar-faced shape, with faces $F_i = \{E \mid$ the edges of S which constitute a cycle on $F_i\}$, $i = 1, \ldots, n$, where $E_j = \{P_1, P_2 \mid$ the endpoints of $E_j\}$, $j = 1, \ldots, r$, are the edges of S, and $P_k = (x_k, y_k, z_k)$, $k = 1, \ldots, m$, are the vertices of S. Let $P_t = (x_t, y_t, z_t)$ be a test point the relative location of which, with respect to S, is to be determined. Let C denote the parity counter, M a set containing marked edges, and T a set containing faces which have been traversed and tested.

Shape traversal and parity count determination:

1. [*Initialize traversal, parity counter, and sets M and T.*] Set $i \leftarrow 0$; $C \leftarrow 0$; $M \leftarrow \{\ \}$; $T \leftarrow \{\ \}$.

2. [*Project test point.*] Let $I_{tp} \leftarrow (y_t, z_t)$ denote the 2D image of TP on the image plane.

3. [*General traversal step and termination condition.*] Set $i \leftarrow i + 1$. If $i = n + 1$ then go to step 10.

4. [*Get candidate face for projection.*] If $F_i \notin T \wedge$ (*the face-plane of* F_i *is penetrated by the* TLN) then let TF $\leftarrow F_i$ denote the currently traversed face, and (mark F_i) set $T \leftarrow T \cup \{F_i\}$.

5. [*Project traversed face.*] Generate a polygon I_{tf}, the 2D image of *TF* (use the *2D image generation* algorithm).

6. [*Determine 2D containment relationship.*] Use the *2D CR Algorithm* to determine the containment relationship *cr* of I_{tp} and I_{tf}.

7. [*Intersection case.*] If *cr* = *inside* then (increment parity counter) $C \leftarrow C + 1$; then go to step 3.

8. [*No intersection case.*] If *cr* = *outside* then go to step 3.

9. [*Singularity case.*] (At this point in the algorithm *cr* = *onVertex* or *onSide*. Use the explicit vertex references A_1 and A_2 returned by the *2D CR Algorithm* to determine and mark the edges of S which converge on elements coincidental with TLN.) Set $M \leftarrow M \cup \{\forall E_j \mid E_j(P_1 \vee P_2) = (A_1 \vee A_2), j = 1, \ldots, r\}$; then go to step 5.

10. [*Determine parity.*] If (the parity counter) C is ODD then P_t is *inside S*, otherwise it is *outside* S. *Halt.*

2D Image Generation. The following assumptions concerning the data structure of the faces and the edges facilitate the representation of the 2D image generation algorithm. *Faces* are assumed to be traversed in a clockwise manner, when viewed from outside the shape. *Edges* are directed, as defined by their endpoints (P_1, P_2).

For every edge, this direction agrees with that of the traversal direction of one of its adjacent faces, and is opposite to that of the other adjacent face.[9] Let *dir* (an integer [1 : 2]) denote the direction in which edge E is being traversed, where $dir = 1$ denotes edge traversal P_1P_2 and $dir = 2$ denotes edge traversal P_2P_1:

1. [*Initialize traversal.*] Set (traversal starting edge) $TSE \leftarrow$ an *unmarked edge* of TF. Set (traversal edge) $TE \leftarrow TSE$; and determine *dir* (the direction of traversing TE while on face TF); Set $i \leftarrow 0$.

2. [*Vertex projection.*] If (unmarked edge) $TE \notin M$ then set $i \leftarrow i + 1$; and (add a vertex to the image polygon) $A_i(u, w) \leftarrow TE(y_{dir}, z_{dir})$ (the y, z coordinates of endpoint *dir* of edge TE); then go to step 4.

3. [*Switch to adjacent face across TE.*] Set $TF \leftarrow$ *adjacent face across TE*; (mark face TF for having been traversed) $T \leftarrow T \cup \{TF\}$; and set $dir \leftarrow (dir \text{ MOD } 2) + 1$.

4. [*Traversal step and termination condition.*] Set $TE \leftarrow$ next edge on TF. If $TE \neq TSE$ then go to step 2; otherwise $I_{tf} \leftarrow (A_1,\ldots,A_n)$ is a straight, planar polygon, with vertices $A_i = (u_i, w_i)$, $i = 1,\ldots,n + 1$, such that $A_{n+1} = A_1$, which is the result returned by the *2D Image Algorithm*. *Halt.*

6.3. *Analysis and Discussion*

Dividing the algorithm representation into two levels does not imply nestedness with respect to iterations. Rather, there is a continuum of tradeoffs between the two levels, maintained by means of set T which is used to mark faces which have been tested, and is updated in both levels (steps 4 in the outer level and 3 in the inner level). Thus singularities which cause the inner level to project other faces besides the one being traversed at the outer level do not change the overall complexity: these additional faces will not be tested again. Therefore, the number of tests performed in the worst case is bounded above by n, the number of faces in shape S, yielding an overall complexity $O(n)$.

There is, however, a difference between an outer-level worst case (when there are no singularities and every face-plane is penetrated by the TLN), and an inner-level worst case (when all the faces of the shape are coincidental with the TLN at least in one element). In the first case each face is projected exactly once, whereas in the second case resolving the singularity of the first face involves repeated re-projections until all the faces of S are projected simultaneously. However, the reprojections account only for a small fraction of the complexity measure, which is dominated by the number of 2D CR tests, which is bounded above by n.

The space complexity, assuming that the shape data are carried by the environment in which the algorithm is embedded, is only that which is required to store the 2D image of the polygon generated when projecting faces. This, in the worst case, is $O(m)$ (m denoting the number of vertices in the image polygon).

The lower bound on the algorithm, when none of the face-planes is penetrated by the TLN, consists only of the cost of establishing the nonpenetration condition for each face, which is of complexity $O(n)$. This cost could, however, be reduced to a constant 1 if the environment embedding the algorithm maintains *minmax* boxing information of the shape (for other purposes as well), as is the case in the shape

[9]By the *Möbius Edge Rule:* see Section 2.

operations algorithm (performing the Boolean operations of union, intersection and difference on shapes) described in [11].

In practice, the more complicated the shape and the larger its number of faces the less likely it is to experience inner-level worst-case conditions, and its singularities will typically involve a small number of simultaneous face projections only.

7. The Intersection Method

The intersection method is another approach to establishing the containment relationship for a point in a shape by reducing the dimensionality of the problem into 2D. That is done by generating zero or more planar polygons which represent the trace of the shape surface where it is intersected by a plane. This intersecting plane (IP) is chosen to contain the test point, so that the generated polygons can be analyzed for containment relationship by means of the 2D CR algorithm. Well-formedness of the shape, as defined in Section 2, guarantees the closedness of the contour polygons and their simplicity, and makes it possible to directly relate the 2D CR test result to the 3D case[10]: the point is outside the shape if it is also outside the 2D intersection contour polygons (ICP).

The ICP is generated by identifying the intersection line segments between each face of the shape and the IP (*minmax* boxing may be used for fast rejection of nonintersecting faces). Nonsequential ordering of the faces on the shape will produce these line segments in a somewhat arbitrary order, so they must be sorted by endpoint matching to form a polygon.

The resulting ICP may, however, not be simple: for example, in case of a face which is coplanar with the IP, the resulting ICP will include a 2D element (an area), possibly in addition to simple line-segments, as depicted by Fig. 7.2. This will disable the use of the 2D CR algorithm, which can only be applied to proper polygons, as they were defined in Section 2. Thus singularity conditions will have a different effect in the intersection method than in the projection method: rather than cause the algorithm to yield wrong results, they will prevent its operation altogether. Therefore, these singularities must either be avoided, or they must be solved directly. First, however, it is necessary to identify the singularity conditions for the intersection method, and isolate the troublesome cases.

7.1. Possible Spatial Relationships between the Shape and the IP

An enumeration of all the (generic) possible spatial relationships between a shape and the IP reveals five different cases, which may appear in various combinations (Fig. 7.3):

1. No intersection (the shape is completely on one side or the other of the IP).
2. One or more nonadjacent vertices are coincident with the IP.
3. One or more nonadjacent edges are coplanar with the IP.
4. One or more faces are coplanar with the IP.
5. A number of faces intersect the IP with none of the above singularities.

[10] By extending the *Jordan Curve Theorem* which states, roughly speaking, that a simple closed polygon always separates the plane into two regions [7] which, in this case, are homomorphic to those regions of the 3D space partitioned by the surface of the shape.

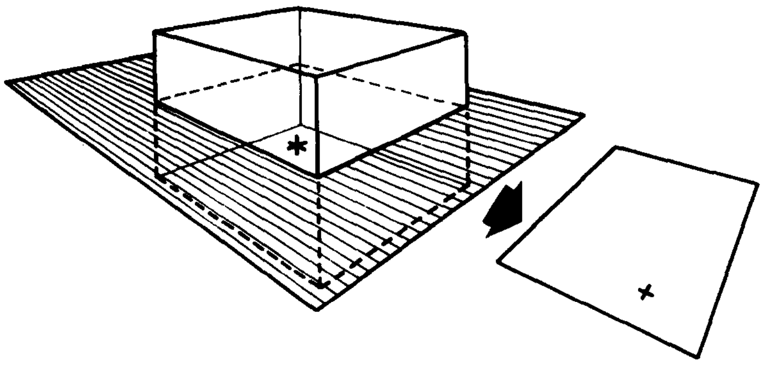

FIG. 7.1. The intersection method.

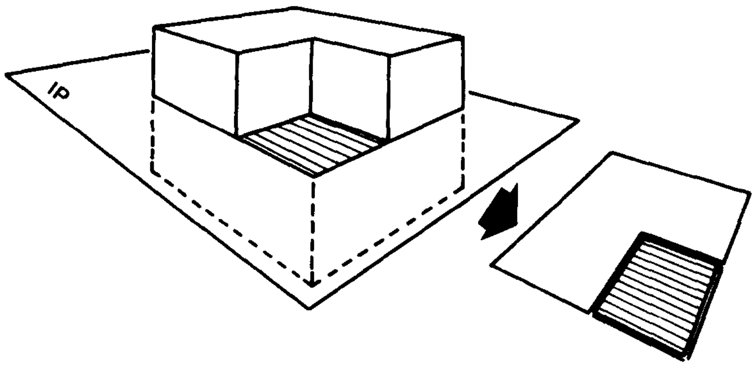

FIG. 7.2. The ICP of a shape including a face coplanar with the IP.

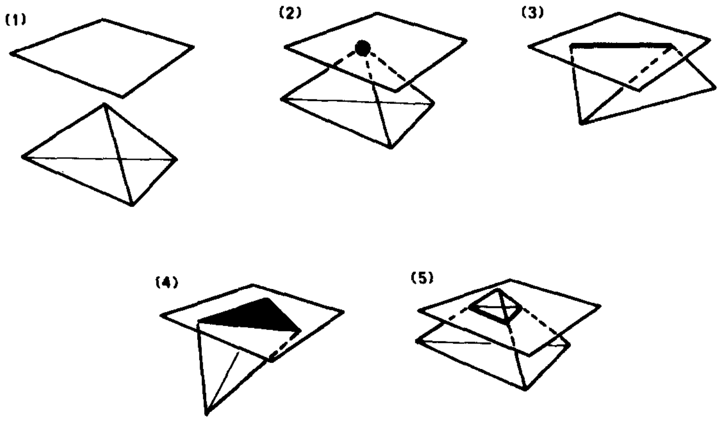

FIG. 7.3. Enumeration of generic spatial relationships between a shape and the IP.

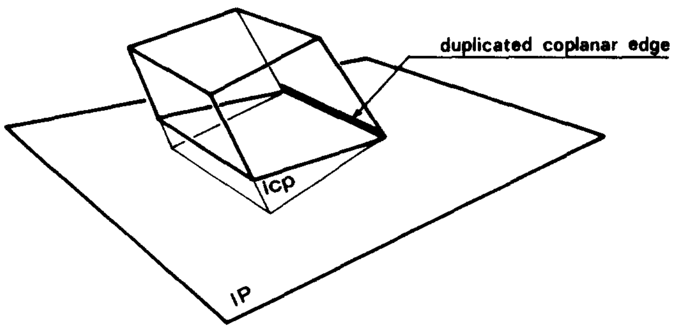

FIG. 7.4. Duplication of an ICP edge caused by a coplanar edge.

Relationship cases of type (1) and (2) have no effect on the 2D CR algorithm, and can be easily detected and eliminated.[11] Case (5) is the general ICP generator, with no associated singularities. It is cases (3) and (4) which cause the troublesome singularities of the intersection method. Case (4) generates the 2D elements in the ICP, whereas Case (3) causes duplication of some of the ICP edges: traversal of the shape faces encounters each edge *twice*,[12] including coplanar edges which, by definition, generate and coincide with ICP edges. This, in most cases, will result in topological "pockets" (though with laminated edges) in the polygon, hindering the use of the 2D CR algorithm by disrupting the parity count, turning the shape inside out. Arbitrary elimination of one of the duplicated edges in each pair does not always produce a desirable result, since it may generate an "open polygon" (a polygonal arc).

These two singularity cases are often combined, increasing the possible complexity even further (Fig. 7.5).

7.2. *Handling Singularities of the Intersection Method*

Both singularity cases can be avoided by selecting another IP or by displacing the coplanar shape elements, as discussed for the projection method. Either method will, however, generate undesirable side effects and will make it necessary to deal with them instead.

Duplicate ICP edges can be eliminated by adopting another convention:

CONVENTION 2. A coplanar edge is considered an ICP edge generator iff the two faces adjacent to it have together at least two points which are strictly on opposite sides of the IP (i.e., together they are "above" and "below" the IP).

Note that this convention will also eliminate edge duplication caused by faces coplanar with the IP.

It is possible now to eliminate singularities altogether by decomposing the ICP into two (or more) polygons, each consisting of elements of a single type (Fig. 7.6), so the 2D elements can be treated as simple polygons for applying the 2D CR test:

[11]Case (2) means that the shape is tangent at a vertex to the IP, generating a degenerate polygon composed of a single vertex. By convention 1 this case will always result in an "outside" containment relationship.

[12]Because each edge is adjacent to two faces, as defined in Section 2.

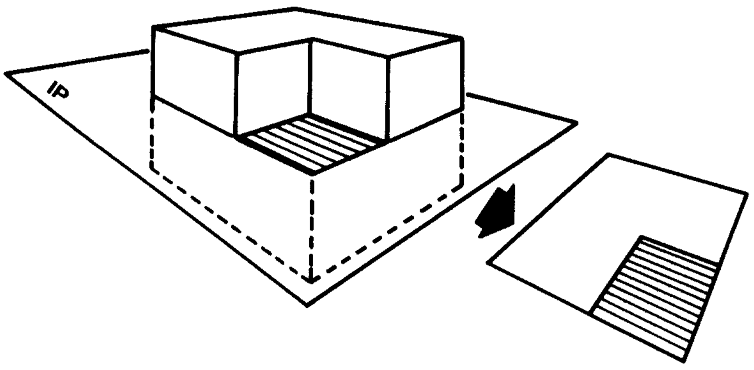

FIG. 7.5. Combined singularities.

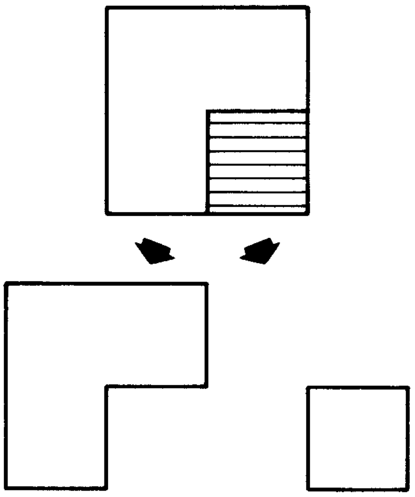

FIG. 7.6. Decomposing the ICP into homogeneous elements.

The TP is considered to be inside the shape iff it is inside one of the noncoplanar polygons of the ICP, and outside otherwise.[13]

7.3. Generating and Decomposing the ICP

The ICP is thus generated by identifying or calculating its edges as a combination of the following intersection cases between the shape and the IP:

1. The intersection line(s) of faces completely intersecting the IP.
2. The coplanar edges (of noncoplanar faces) which comply with convention 2.
3. All the edges of coplanar faces.

Identifying the line(s) of intersection between each face and the IP for generating the ICP can be done in different ways, none of which is trivial to implement, and are

[13]If it is inside a 2D polygon then it is on the surface of the shape, which by convention 1 means that it is outside the shape.

beyond the scope of this paper. One such method, which first identifies the points of intersection between each edge of the face and the IP, generating a list of points later to be pairwise-sorted into intersection line segments, is described in [11]. Other methods, such as calculating the intersection line first then sectioning it into the appropriate actual intersection segments, are also possible. Both methods will however fail for curved-surfaced faces, for which more complicated methods, such as approximating the line of intersection by means of the Lane-Riesenfeld algorithm [12], will have to be employed.

Having generated all the edges which make-up the ICP, they undergo endpoint-matching sort and decomposition into "2D" and simple polygons. The resulting polygons are tested for containment relationship with the 2D trace of the test point (by means of the 2D CR algorithm), yielding a result which establishes the overall containment relationship between the shape and the point. The order of testing the resulting polygons is the basis for correct resolution of the singularities: "2D" polygons need to be tested *first*, since if the TP is included within any of them, meaning that it is *on* the surface of the shape, the overall containment result is established as an *outside* relationship (by convention 1), whereas if it is inside a simple ICP polygon, the overall containment test result is an *inside* relationship.

Basically, the endpoint matching and decomposition mechanism is a simple exhaustive search of the ICP list of edges. It is, however, complicated by the non-uniqueness of the successive edges, if coplanar face generated edges are combined with simple intersection edges (Fig. 7.5), resulting in a multi-looped graph of endpoint matched edges, rather than a single cycle (Fig. 7.7).

Local resolution of these ambiguities would require much computation, or a rather complicated data structure (possibly "winged edges" such as described in [13]). Globally, however, it makes no difference which edges will be substituted for closing the polygon, provided the "2D" polygons are tested first.

The substitution can thus be done arbitrarily by order of appearance in the list, using the following algorithm: The edges resulting from faces coplanar with the IP are flagged. The decomposition is then done by repeatedly searching the ICP to find endpoint-matched succeeding edges that form a cycle (without backtracking). Having found a matching successor, if unflagged it is removed from the ICP and used

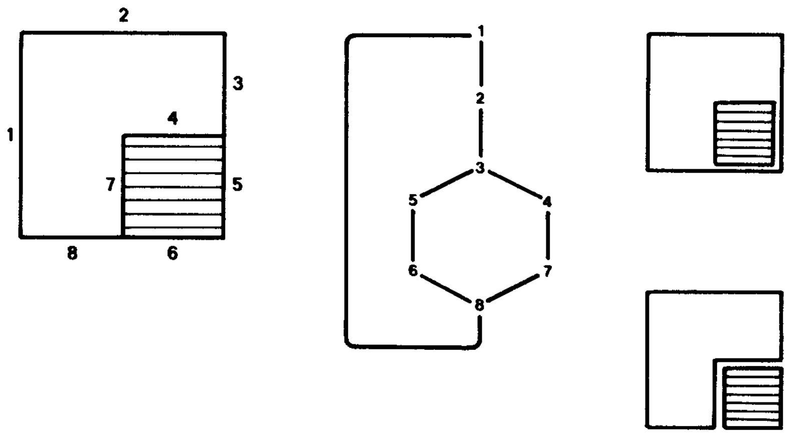

FIG. 7.7. Graph representation of the ICP.

directly as a polygon side, else (if flagged) it is copied into the polygon (Fig. 7.8), and marked in the ICP for having been encountered. Flagged edges are removed from the ICP when encountered the second time. If there is no matching successor in the ICP then the current polygon is "closed" and a new one is "opened." The process is terminated when the ICP is empty or all its edges have been encountered at least once. A more formal description of the algorithm now follows.

7.4. Sort-Decomposition Algorithm

Let ICP denote the unsorted set of Intersection Contour Polygon edges, with $E \in$ ICP an edge such that $E = \{P_1, P_2, f, t\}$ is a four-tuple where (P_1, P_2) is the pair of endpoints of E, f is a Boolean flag set to *true* if E was generated by a coplanar face, and t is a Boolean flag set to *true* after E has been traversed once. Let $P_{out} = \{A_j, f \,|\, (A_j = (u_j, w_j), j = 1,\ldots,n+1$, such that $A_{n+1} = A_1)$; and (f is a Boolean flag set to *true* if P was generated from flagged edges only)$\}$ $out = 1,\ldots,m$, denote the output of the algorithm which is a list of polygons comprised of sorted lists of ICP vertices. $out = 0$ denotes a zero-length list. Let $dir = \{1, 2\}$ denote the two endpoint subscripts of an edge.

1. [*Initialize output polygon-list.*] Set $i \leftarrow 0$; $out \leftarrow 0$.

2. [*Termination condition.*] If (ICP = *empty*) $\vee$ (*all its edges were encountered at least once*) then *halt*: P_{out}, $out = 0,\ldots,i$ is the list of polygons to be tested for containing the test point (by means of the *2D CR Algorithm*). Otherwise (open a new output polygon) set $i \leftarrow i + 1$; $out \leftarrow i$.

3. [*Initialize output polygon traversal.*] (set EP to an unflagged edge in the ICP (if one exists), else flag polygon and set EP to any side) $j \leftarrow 0$; and (endpoint identifier) $dir \leftarrow 1$. Let $EP \leftarrow (E, 0 \,|\, (E \in$ ICP, such that $E(f) = false) \vee (E = 0$ if

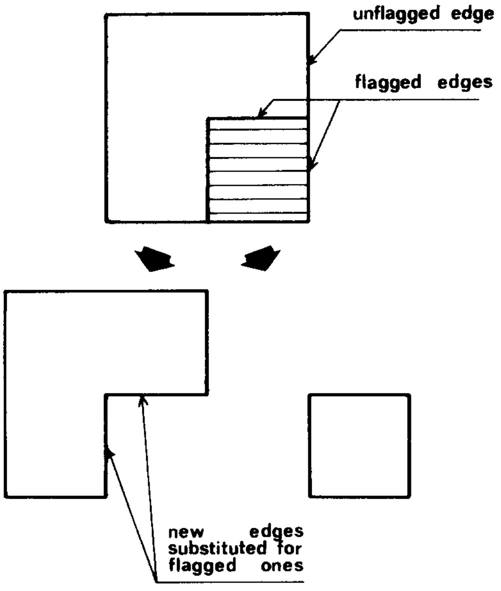

FIG. 7.8. Substituting flagged edges.

there is no unflagged edges in ICP)}. If $EP = 0$ then (flag polygon) $P_{out}(f) \leftarrow true$, and $EP \leftarrow E$ (any side in ICP).

4. [*Output polygon traversal step.*] Set $j \leftarrow j + 1$, and $A_j(u_j, w_j) \leftarrow EP(P_{dir})$.

5. [*Removal or flagging of ICP side.*] If $EP(t) = true$ or $EP(f) = false$ then (remove EP from ICP) ICP $\leftarrow$ ICP $-$ $\{EP\}$; otherwise (flag EP) set $EP(t) \leftarrow true$.

6. [*Matching successor edge.*] $EP \leftarrow \{(E \mid E \in$ ICP the *dir* endpoint matched successor edge to $EP) \vee (0 \mid$ if no such edge in ICP) $\vee$ $(0 \mid$ if $E = EP)\}$. If $EP = 0$ then (current output polygon is closed) go to step 2; otherwise go to step 4.

7.5. *Analysis and Discussion*

Generating the ICP edges by intersecting the faces of the shape with the *IP* is of complexity $O(n)$ at most (n denoting the number of faces in the shape). Additional costs are incurred by sorting the ICP list once it has been generated, at a worst-case cost of $O(2r \log 2r)$ (r denoting the number of ICP edges and it is doubled since one edge may be used for two polygons), then by testing its decomposed polygons for containing the test point (by means of the 2D CR algorithm), at a cost of $O(\Sigma m_i)$(m_i denoting the number of the vertices in each polygon).

A worst case of the sort-decomposition algorithm (when all the faces of the shape are coplanar with the IP, thereby generating an ICP containing all the edges of the shape) will produce n polygons when decomposed after $O(2R \log 2R)$ length search (R denoting the total number of edges of the shape). These polygons will undergo at most $O(n)$ 2D CR tests, resulting in an overall complexity of $O(n + 2R \log 2R + \Sigma nm_i)$, or $O(n + 2R \log 2R + nM)$ for (average) trihedral-vertex shapes (M denoting the total number of vertices of the shape). An average case of the sort-decomposition algorithm, when each face of the shape intersects the IP without singularities, will produce an ICP of m edges constituting a single polygon (and hence, a single 2D CR test), resulting in an overall cost of $O(m(m + 3))$ (this estimate, however, does not provide for faces which produce more than a single intersection edge). However, the complexity of the sort-decomposition algorithm and the 2D CR tests are negligible in comparison with that of generating the ICP list of edges in the first place, and could therefore be considered as constants, establishing the overall time complexity of the intersection algorithm at $O(n)$.

The space complexity, assuming that the shape data structure is carried by the environment, is dominated by the number of ICP edges, and is therefore $O(r)$. It is complemented by the space required for storing the polygons for the 2D CR test, but since there is an almost complete overlap between the two (the polygons are generated while the ICP edges are deleted), the overall space complexity is unchanged.

8. CONCLUSION AND COMPARATIVE EVALUATION

It has been demonstrated that the added complexity of the 3D point–polyhedron containment analysis, over its 2D point–polygon counterpart, stems from two problems related to the nonsequential ordering of the shape elements: (1) identification of the shape elements which are relevant for resolving a particular singularity condition, and (2) finding a common representation for these elements and the test point which will capture their relative relationships in a way that can be uniquely interpreted. The principle of expanding the locality of the shape data brought to bear on a particular singularity resolution was explained in Section 5, based on the global nature of the well-formedness characteristics of a shape, which are the predominant factor in establishing inside/outside relationship between a point in space and the surface of a shape.

Two algorithms, the *projection method* and the *intersection method*, were presented and analyzed independently, both solving the relevancy and representation problems by reducing the dimensionality of the problem into the solvable 2D point–polygon case. Both have been implemented on a VAX 11/780 computer in the PASCAL programming language, and were found to be effective in establishing the containment relationship between a point and complex shapes. Some examples are shown in Figs. 8.1 and 8.2.

It is time now to draw some comparative conclusions from the two algorithms. Figure 8.3 shows a timing analysis of six representative cases, all using a shape which consists of a two-unit cube centered at the origin, one of its octants notched out (the one in the positive octant of the coordinate system). The location of the test point relative to this shape was changed as shown in Fig. 8.4, generating different singularity conditions in each case.

It is evident from examining Fig. 8.3 that neither algorithm is clearly superior to the other; a particular case which causes singularity conditions when applying one method may not cause such conditions when applying the other method. Consider, for instance, Cases 1 and 6. Case 1 shows the test point located inside the shape, on

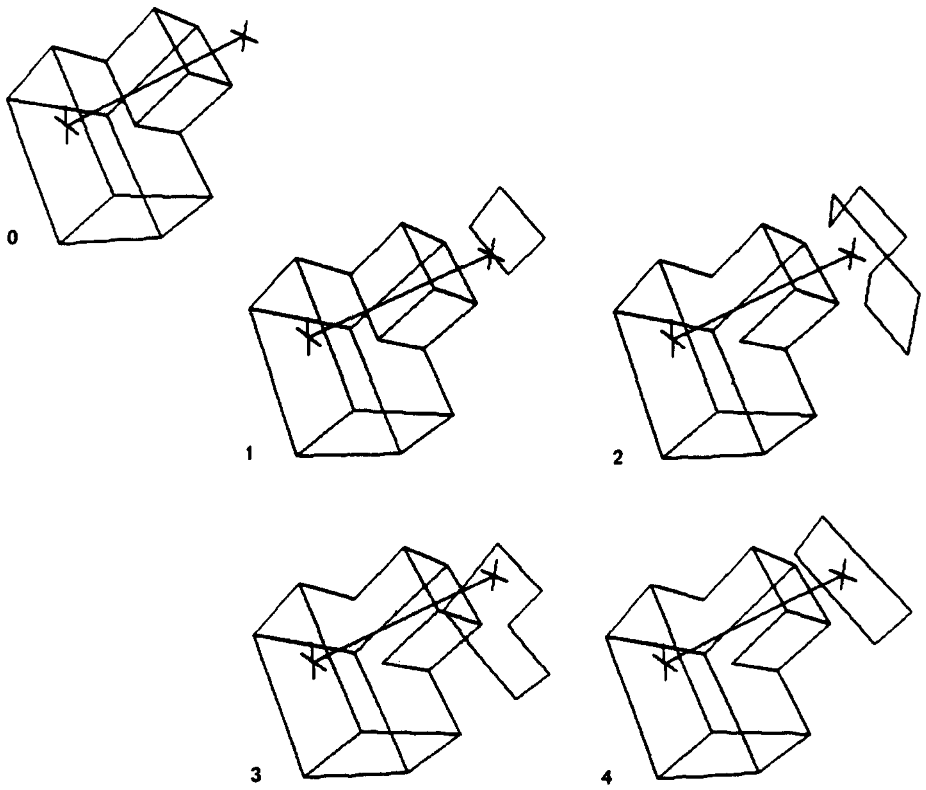

FIG. 8.1. Example of iterative application of the projection method.

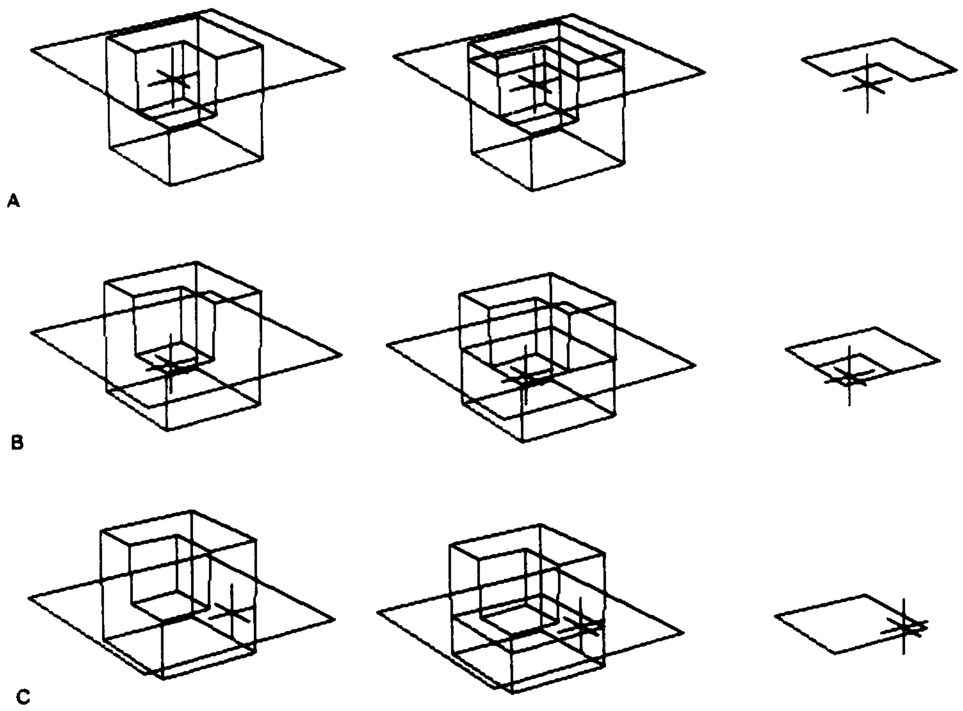

FIG. 8.2. Examples of applications of the intersection method.

	Test point	Method		P/I	Projections	ICP
		P	I			
1.	(−0.5, −0.5, 0)	15	109	7.26	Single face projected; no singularities	ICP contains 2D element and six sided polygon
2.	(0.5, 0.5, 0.5)	23	77	3.34	Two faces projected; no singularities	ICP contains six-sided polygon; no singularities
3.	(1, 0, 0)	63	105	1.66	TLN collinear with an edge; five faces projected, one re-projection	ICP contains 2D element and six-sided polygon (same as Case 1)
4.	(1, 1, 0.5)	68	76	1.11	TLN coincidental with two edges; four faces projected, two re-projections	ICP contains six-sided polygon; no singularities
5.	(1, 1, −0.5)	70	46	0.65	TLN coincidental with two edges; four faces projected, two re-projections (same as Case 4)	ICP contains four-sided polygon; no singularities
6.	(1, 1, −1)	61	28	0.46	TLN collinear with an edge; four faces projected, one re-projection	Empty ICP

FIG. 8.3. Timing analysis for the projection and intersection methods. The times quoted above should be considered for comparative evaluation only, since some program overhead is involved.

the lower horizontal face-plane of the notched octant. It presents no singularities for the projection method, but generates a coplanar face when applying the intersection method, causing the *projection* method to be 7.26 times faster than the intersection method. On the other-hand, Case 6 shows the test point located on one of the lower vertices of the shape, causing a collinearity of an edge with the TLN when applying the projection method (which is solved with one re-projection), while generating an *empty* ICP when applying the intersection method (according to convention 2), causing the *intersection* method to be 2.17 times faster than the projection method.

Figure 8.3 also shows, as was estimated in Section 6, that the number of reprojections is relatively insignificant (consider cases 3 and 4), and that the number of intersecting faces is the predominant factor in determining the speed of the intersection method (consider cases 4 and 5).

The significant difference between the two methods is in the range of shapes to which each can be applied. The reliance on planarity conditions, which is inherent in the underlying projections, and the use of the composite polygon contour for resolving singularities, limits the applicability of the projection method to planar-faced polyhedra only. This limitation, together with the indirectness of the projection

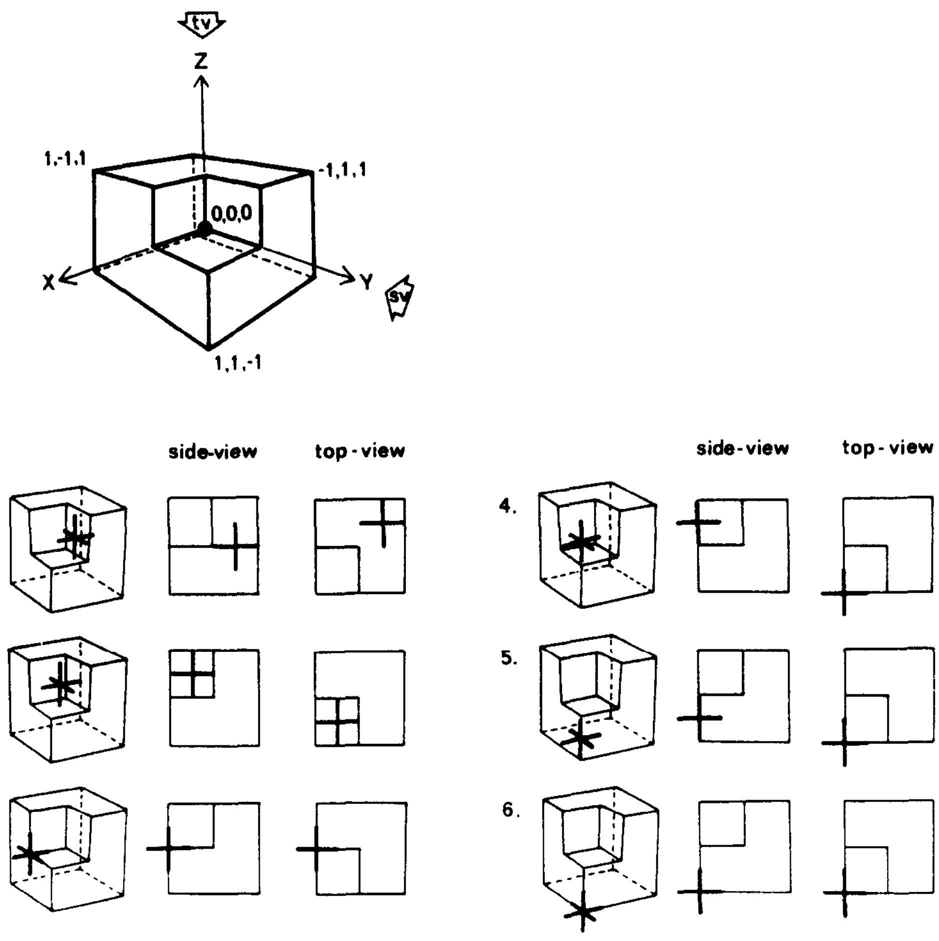

FIG. 8.4. Test cases for timing analysis.

method for establishing the containment relationship, led to the investigation of the intersection method, whose independence of planarity conditions makes it applicable for establishing the containment relationship between a point and curved-surfaced shapes, provided the 2D CR algorithm is extended to handle planar curves as well as straight lines.

Both methods have merits and drawbacks, promoting perhaps the desirability of pre-processing each case and determining, based on the type of the singularities, which one of the two algorithms should be used. However, such preprocessing is time consuming and will be justified only in cases where the shape is extremely complex, requiring many iterations when applying one algorithm but not the other. Still, for planar-faced polyhedra, the use of the projection method seems advantageous, since its speed gains may reach a factor of 8 over the intersection method, whose speed gains reach only a factor of 2 over the projection method, in the examples presented here.

ACKNOWLEDGMENTS

The author is grateful to Charles M. Eastman for his comments and many helpful suggestions in developing the ideas and algorithms in this paper. This work was supported by NSF Grant MCS-78-22328.

REFERENCES

1 M. I. Shamos, Computational Geometry, Ph.D. thesis, Yale University, 1978.
2. J. L. Bentley and W. Carruthers, Algorithms for Testing the Inclusion of Points in Polygons, 18th Annual Allerton Conference on Communication, Control and Computing, ACM, 1980.
3. D. T. Lee and F. P. Preparata, Location of a point in a planar subdivision and its applications, *SIAM J. Comput.* **6**(3), September, 1977, 594–606.
4. W. Burton, Representation of many-sided polygons and polygonal lines for rapid processing, *Comm. ACM* **20**(3), March, 1977, 166–171.
5. E. E. Barton and I. Buchanan, The polygon package, *Computer Aided Design* **12**(1), January, 1980, 3–11.
6. I. E. Sutherland, R. F. Sproull, and R. A. Shumacker, A characterization of ten hidden-surface algorithms, *Comput. Surveys* **6**(1) March, 1974, 1–55.
7. P. J. Giblin, *Graphs, Surfaces and Homology*, Chapman and Hall/Halsted Press, New York, 1977.
8. A. Baer, C. M. Eastman, and M. Henrion, Geometric modeling: A survey, *Computer Aided Design* **11**(5), September, 1979, 253–272.
9. F. P. Preparata, *A New Approach to Planar Point Location*, Technical Report, Coordinated Science Laboratory, Univ. of Illinois, August, 1979.
10. W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, second ed., McGraw–Hill, New York, 1979.
11. Y. E. Kalay and C. M. Eastman, *Shape Operations: An Algorithm For Spatial-Set Manipulation Of Solid Objects*, Technical Report 10, Institute of Building Sciences, Carnegie–Mellon Univ., July, 1980.
12. J. M. Lane and R. F. Riesenfeld, A theoretical development for the computer generation and display of piecewise polynomial surfaces, *IEEE Trans. Patt. Anal. Mach. Intell.* **PAMI-2**(1) January 1980, 35–46.
13. W. Baumgart, *Winged Edge Polyhedron Representation*, Technical Report CS-320, Stanford Artificial Intelligence Laboratory, October, 1972.