
Algorithms for classifying and constructing the boundary of a shape

Rudi Stouffs*

Faculty of Architecture,
Delft University of Technology,
Delft, The Netherlands
E-mail: r.m.f.stouffs@tudelft.nl
*Corresponding author

Ramesh Krishnamurti

School of Architecture,
Carnegie Mellon University,
Pittsburgh, USA
E-mail: ramesh@cmu.edu

Abstract: This paper continues with the subject matter that we introduced previously (Krishnamurti and Stouffs, 2004). Here, we describe algorithms for classifying the boundary of a shape with respect to another, coequal, shape and for constructing the description of a shape given parts of the boundary that make up the shape. Specifically, algorithms for classification and construction of shapes in U_{23} (plane shapes) and in U_{33} (volume shapes) are described in this paper. These procedures form a unified basis for shape arithmetic.

Keywords: shape; shape arithmetic; geometrical modelling; algorithms; computational complexity.

Reference to this paper should be made as follows: Stouffs, R. and Krishnamurti, R. (2006) 'Algorithms for classifying and constructing the boundary of a shape', *J. Design Research*, Vol. 5, No. 1, pp.54–95.

Biographical notes: Rudi Stouffs is Associate Professor at the Design Informatics Chair, Faculty of Architecture, Delft University of Technology. He holds an MSc in architectural engineering from the Free University, Brussels, an MSc in Computational Design and a PhD in Architecture from Carnegie Mellon University (CMU). He has been Assistant Professor at the Department of Architecture at CMU and Research Coordinator at the Chair for Architecture and CAAD at ETH Zurich. His research interests include computational issues of description, modelling and representation for design in the areas of information exchange, collaboration, shape recognition and generation, geometric modelling and visualisation.

Ramesh Krishnamurti is a Professor in Computational Design in the School of Architecture at Carnegie Mellon University. He holds a PhD in Systems Design from the University of Waterloo. His research focuses on the formal, semantic, generative and algorithmic issues in computational design. Past research include work in spatial grammars, spatial algorithms, geometrical modelling, analyses of design styles, knowledge-based design systems, integration of graphical and natural language, graphic environments, computer simulation and

war games. His current projects deal with sensor-based model reconstruction, shape grammar implementations, building information models, and green CAD.

1 Basic problems

In a companion paper (Krishnamurti and Stouffs, 2004), we laid out a unified framework for computing the boundary of a shape, for applications where structured descriptions of shapes are important, as, for example, in the implementation of shape grammars (Krishnamurti and Stouffs, 1993; for an extended motivation we refer to Krishnamurti and Stouffs, 2004). The structured descriptions have two parts: carrier, a shape that embeds the given shape, and boundary, a shape that specifies the form of the shape. The structured descriptions concern shapes as composed of segments: a shape is a segment if it has no nonempty proper subshape (i.e., part) the boundary of which is a part of the boundary of the segment. A segment is thus a shape with a ‘minimal’ boundary with respect to the shape. We showed that the boundary of a shape can be classified with respect to another shape by splitting the boundary segments into disjoint classes so that each split segment can be identified as inner, outer, shared in the same way or shared oppositely with respect to the other shape. Further, for any shape operation, the segments that make up the boundary of the resulting shape consist of segments from specific subsets of classes as indicated in Table 1. The letters I , O , M , and N respectively denote classes of inner, outer, same-shared, and oppositely-shared segments of the specified shape with respect to the other.

Table 1 Classified boundary segments that make up the boundary of the shape resulting from a shape operation

Operation:*	$X + Y$	$X \cdot Y$	$X - Y$	$X \oplus Y$
Boundary: $boundary [X * Y]$	$O_X + O_Y + M$	$I_X + I_Y + M$	$O_X + I_Y + N$	$I_X + I_Y + O_X + O_Y$

*Any of the four operations ‘+’, ‘·’, ‘-’ and ‘ \oplus ’.

The following classes of problems form the basis of these operations:

- classifying a segment with respect to a coequal shape where a carrier of the segment is embedded within the carrier of the shape
- constructing the representation of a shape – that is, its maximal segments – from the given parts of the boundary of the shape obtained from the classification.

We present classification and construction algorithms for shapes in algebra U_3 (Stiny, 1991), that is, linear shapes in a 3-dimensional Euclidean space. The original versions of these algorithms were first presented in Stouffs’ dissertation (Stouffs, 1994), and rely implicitly on an isomorphism between U_3 and subsets of $\wp(E^3)$. As such, for our presentation, we operate simultaneously on hyperplanes in E^3 and segments in U_3 (as if these objects exist in the same space); for example, the intersection of a line and line segment refers, specifically, to the point of intersection of the line with the point set in E^3 isomorphic to the given line segment. In our development, we freely borrow

from conventional geometrical modelling methods; at the same time, we note that, although it is possible to find similar algorithms based on more conventional boundary representations for solids (see, for example, Hoffman, 1989; Mäntylä, 1988), these algorithms generally require additional special procedures to handle specific properties – for example, non-manifold shapes – if such properties are present. Moreover, the algorithms require adaptation to cater for the kinds of spatial problems that we are interested in. For instance, conventional boundary representations do not readily deal with arbitrary subshapes of a shape. Alternatively, algorithms may reflect on operations that are not closed within one dimensionality and, instead, are defined across dimensionalities (Gursoz et al., 1991). In this case, shapes cannot be defined to share boundary.

2 Preliminary

We rely on the following general representation for a maximal shape as a unique set of maximal segments each represented by a carrier-boundary pair. The carrier is identified by a tuple, its *co-descriptor*, which, for a segment x is denoted by $co[x]$. Typically, this is a representation of the equation of the carrier shape. A boundary shape consists of a set of simple boundaries $\{b_i\}$ (that is, ‘minimal’ boundary shapes), each a maximal shape; the boundary shape is represented as a set, $boundary[x] = \cup b_i$. For any boundary segment k of a shape X , $inside[k]$ represents its neighbourhood relationship with respect to X . The decomposition of a maximal shape into its maximal segments is unique. So too is the decomposition of a boundary shape into its simple boundaries.

We impose a total order on shapes according to their constituent segments. In turn, we impose a total order on the segments according to their co-descriptors, and for coequal segments according to their boundaries. The same total order applies to the simple boundaries that define a segment. In the particular case of comparing two discontinuous (that is, disjoint and no shared boundary segments) segments x and y , it suffices to compare the first segment from $boundary[x]$ and $boundary[y]$.

Procedures such as SUM, PRODUCT, DIFFERENCE, CLASSIFY and CONSTRUCT are used independent of the algebra of their arguments, even though their implementation may differ with each algebra. For example, $SUM(X, Y)$ returns the sum of two shapes X and Y , where X , Y and the resulting shape all belong to the same shape algebra.

As a convention, we use capital letters to denote shapes, that is, sets of (maximal) segments, and lower case letters for single segments. Thus, a segment x considered as shape would be written as $\{x\}$. We adopt the following convention: letters s and t denote volume segments; f and g plane segments; k and l line segments; and p and q points. The co-descriptor for any segment x is a description of the equation of its carrier, and is represented in two ways: as a functional by $co[x]$, and as a value by $co-x$. [It should be noted that for any segment there are infinitely many carriers, and we consider the maximal carrier that carries all other carriers of the segment for algorithmic purposes.] The following vector notation is employed: pq denotes the vector from point p to q ; d_l denotes the direction vector for a line or line segment l ; and n_f denotes the normal vector for a plane or plane segment f . Vectors can be positive or negative depending on whether or not $v > 0$, specified in the usual way. The norm of a vector v , denoted as $\|v\|$, is a normalisation of both its magnitude and sign; that is, $\|-v\| = \|v\|$. The capital letters

I , O , M and N are reserved to denote classes of inner, outer, same-shared, and oppositely-shared segments with a single subscript to denote the shape this class pertains to. For example, I_X denotes the class of inner segments of shape X , with respect to the other shape. The symbols $+$, $-$ and \cup respectively denote the set operations join, delete and merge (not removing any duplicate elements), the latter operating on sorted sets. The operators $+$ and $-$ should not be confused with the shape operations of sum and difference which, as algorithms, are represented by the procedures SUM and DIFFERENCE.

Procedures for shape operations: SUM, PRODUCT, DIFFERENCE and SYMMETRIC-DIFFERENCE, and shape relations: CONTAIN, OVERLAP, SHARE-BOUNDARY and DISCONTIGUOUS, that we describe in Krishnamurti and Stouffs (2004), rely on two procedures for their result: CLASSIFY and CONSTRUCT. CLASSIFY operates on two coequal shapes and classifies the boundary segments of each shape with respect to the other into the four classes. CONSTRUCT takes as input a set of (line or plane) segments that defines the boundary of a coequal shape and constructs this maximal shape. Classification and construction algorithms for U_{23} (*plane shapes*) and U_{33} (*volume shapes*) are described in this paper. The conventions and form of the English like pseudo-code for the algorithms are due to Cormen et al. (1990). We use the symbols ' \leftarrow ' and '=' to represent the assignment operator and equality relation respectively. Moreover, unless specifically referenced, data structures are also taken from Cormen et al. (1990). Comments within the algorithms are expressed italicised.

We use other procedures to assist in the development of the algorithms. We list in Table 2 basic procedures that are common to the subsequent algorithms. All other procedures are defined upon use.

Table 2 Common basic procedures

INTERSECTION ($co-k, co-l$)	Returns the point, if any, isomorphic to the point of intersection of the lines l and k
INTERSECTION-2 ($l, co-k$)	Returns the point, if any, isomorphic to the point of intersection of all carriers of the line segment l and line k ; p must be coincident with l
LINE-SEGMENT (p, q)	Returns a line segment with endpoints p and q
INTERSECTION-LINE ($co-f, co-g$)	Determines the co-descriptor of the line of intersection of the planes f and g
NORMAL-PLANE ($co-l, co-f$)	Determines the co-descriptor of a plane normal to the plane f and through the line l
PARALLEL ($co-x, co-y$)	Compares two co-descriptors of segments or shapes in the same algebra and returns TRUE if they are either equal or represent parallel carriers and FALSE otherwise

Procedures SUM, PRODUCT and DIFFERENCE as well as relations CONTAIN, OVERLAP, SHARE-BOUNDARY and DISCONTIGUOUS on shapes in U_0 (point shapes) are trivial. The same procedures on shapes in U_1 (comprising line segments) are well-known (see Chase, 1989; Krishnamurti, 1980). The arithmetic of plane segments is proven in Krishnamurti (1992). An overview of shape arithmetic is given in Krishnamurti (1980). We note that SUM, PRODUCT and DIFFERENCE take linear time in the size of

their input, for point and line shapes. Here, we restrict our treatment to plane and volume shapes.

Since we are also interested in the computational complexity of the algorithms, we employ the standard functional notation, namely: Θ , O and Ω , to refer, respectively, to the worst case, asymptotic upper bound and asymptotic lower bounds (see Cormen et al., 1990).

The procedures in Table 3 take $O(n \log n)$ time, where n is the number of segments.

Table 3 Basic procedures on shapes (sets of segments) that take $O(n \log n)$ time, where n is the number of segments

MAXIMAL (X)	Converts an unsorted set of line segments that may share boundary, but do not overlap, X , into the corresponding maximal shape
SORT (X)	Sorts a set of segments, X , in correspondence to the total order defined on segments. Note that for any given set of discontinuous segments, the time complexity of SORT is dependent only on the size of the set and not on the sizes of the boundary shapes

The procedures in Table 4 all take linear time in the size of their input.

Table 4 Basic procedures on shapes (sets of segments) that take time linear in the number of segments

REDUCE (X)	Removes multiple occurrences of elements in a sorted set X
REDUCE-DUPLICATES (X)	Removes pairs of duplicate elements in a sorted set X
REMOVE-MULTIPLES (L)	Removes multiple occurrences of coincident line segments from a line shape
REMOVE-DUPLICATES (L)	Removes pairs of duplicate or coinciding line segments from a line shape

Where needed, we take, as given, that a line segment l is specified by a pair of endpoints, $tail[l]$ and $head[l]$, with $tail[l] \leq_c p \leq_c head[l]$, where \leq_c denotes lexicographical ordering on their coordinates. Whenever a segment is embedded within the carrier of a shape, we say that the segment is *coincident* with the shape, e.g., a line segment lying (or embedded) in a plane segment is coincident with the plane segment.

The remainder of this paper is divided into two parts. These two parts deal, respectively, with plane shapes and volume shapes. Each part describes the classification of the boundary of a shape with respect to another and the construction of the maximal element representation of a shape given its constituent parts.

3 Part I: classifying the boundary of a plane shape

3.1 Classification of line segments

The first instance of the problem is in classifying a line with respect to a plane shape. We consider a coequal plane shape F and an infinite line l coincident with the carrier of F . Boundary segments of F have associated shape neighbourhoods that define insides and outsides in relationship to F . In this case, the boundary of F defines an inside and outside

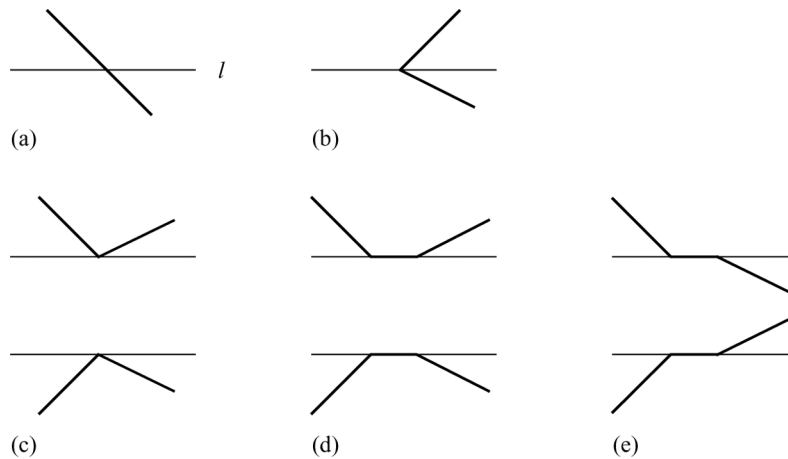
region in the carrier plane such that any part of l that does not intersect the boundary of F can be deemed inner or outer with respect to F . Without considering degenerate cases, each point of intersection of l and the boundary of F is an endpoint to two disjoint parts (segments) of l , one of which is deemed inner, while the other is deemed outer, with respect to F . As such, the set of intersection points of l and the boundary of F defines an alternating sequence of inner and outer segments. Since both infinite ends of the line l can be considered outer and each intersection point alternates the classification, the total number of intersection points of l and the boundary of F must be exactly even.

Procedure A

The points of intersection of the boundary of a coequal plane shape F and an infinite line l within the carrier of F , when sorted, define an alternating sequence of inner and outer segments, starting with an inner segment.

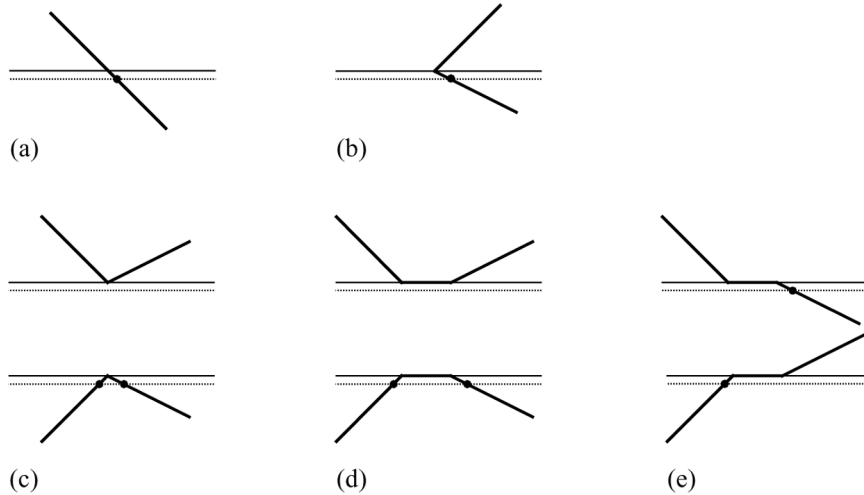
Figure 1 illustrates the different cases for the intersection of a (horizontal) line l and the boundary of a plane shape F , where the line and shape are coplanar. Cases (b)–(e) are degenerate in that the point of intersection coincides with the endpoint of (at least) two boundary segment. Other degenerate cases are compositions of the cases. In order for the cases to be consistent with Procedure A for a set of intersection points on l , cases (a) and (b) must constitute a single point of intersection, case (c) zero or two coincident points of intersection, case (d) zero or two non-coincident points of intersection, and case (e) again a single point of intersection.

Figure 1 Simple cases of intersection of an infinite line with the boundary of a plane shape. Cases (b)–(e) are degenerate



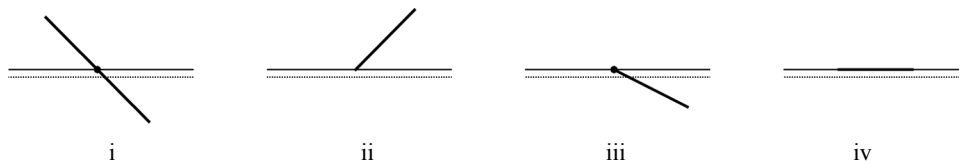
Consider a line l' parallel to the given line l at an arbitrarily small distance from l , within the carrier of F . For any boundary segment of the plane shape F intersected by l in one of its endpoints, the segment either intersects l' in a single point that is not an endpoint or does not intersect at all. Furthermore, whether there are zero or one intersection points depends solely on which side the boundary segment is with respect to the line l , within the carrier of F . Such a line l' always exists, and none of the degenerate cases for l can be a degenerate case for l' . Figure 2 illustrates the resulting cases for l' .

Figure 2 All degenerate cases are resolved by translating the line of intersection over an arbitrarily small distance perpendicular to its axis



If we consider only those intersection points for l that correspond to the intersection points of l' , and remove pairs of coincident points of intersection (lower diagram in case (c)), then all cases are consistent with Procedure A. However, the class of inner segments determined from these points of intersection using Procedure A, may contain shared segments, i.e., those boundary segments of F coequal with l . Then, the segments that are deemed inner with respect to the shape F result from taking the difference of this class of inner segments with the class of shared segments previously determined. Whether a single boundary segment has an intersection point with l does not depend on the particular case. As a result, we distinguish four basic cases (i–iv in Figure 3) for the intersection of a single boundary segment with a line. Any degenerate case is a composition of two or more of these four cases. When comparing the degenerate cases ii–iv, we note that an intersection point is retained only if the boundary segment lies on one side of the line l and not on the other, nor if it is a part of l . It suffices to check only for one point on the segment, for instance, the other endpoint.

Figure 3 Four cases for the intersection of a line segment with a line. Cases ii–iv are degenerate. Only cases i and iii result in an intersection point



The following inequality constitutes a simple condition to determine whether the point of intersection of a line segment k on the boundary of F is to be included:

$$\text{DOT-PRODUCT}(co[tail[k]], co-g) > 0 \text{ or } \text{DOT-PRODUCT}(co[head[k]], co-g) > 0.$$

The plane g is normal to F through l . This condition holds for all cases including degenerate ones. For case i, the endpoints of k lie on different sides; for case iv, the DOT-PRODUCT is zero for both endpoints; for cases ii and iii, one endpoint has a DOT-PRODUCT different from zero, and one of the conditions must be positive while the other is negative.

This discussion is summarised in procedure CLASSIFY-LINE (Figure 4) with input, the co-descriptor $co-l$ of a line l and a coequal plane shape F , with the condition that l is coincident with the carrier of F . The procedure returns the classes of inner and shared line segments, with respect to F , with the given co-descriptor. The influence of Procedure A can be seen in the loop defined by steps 11–13. The dot product condition on the inclusion of the point of intersection of a line segment k is captured in steps 6–8.

Figure 4 CLASSIFY-LINE: classifying a line l with respect to a coequal plane shape F with l coincident with the carrier of F

```

CLASSIFY-LINE (co-l, F)
1  I ← M ← P ← ∅
2  co-g ← NORMAL-PLANE (co-l, co[F])
   Classify l against each line segment k in the boundary
3  for each line segment k ∈ boundary[F]
4     if co[k] = co-l
5     then M ← M + {k}
   Is there a point of intersection?
6     else p ← INTERSECTION-2 (k, co-l)
7         if p ≠ 0 and (DOT-PRODUCT (co[tail[k]], co-g) > 0
   or DOT-PRODUCT (co[head[k]], co-g) > 0)
8         then P ← P + {p}
9  SORT (P)
   Remove duplicate points
10 REDUCE-DUPLICATES (P)
   Procedure A: alternating sequence of inner and outer segments
11 for each point p ∈ P
12     I ← I + { LINE-SEGMENT (p, next[p]) }
13     P ← P - { p, next[p] }
14 MAXIMAL (M)
15 I ← DIFFERENCE (I, M)
16 return (I, M)

```

Let n denote the number of maximal boundary segments of maximal segments of F ; $n = |\text{boundary}[F]|$. Then, the sizes of both M and P , and hence I , are $O(n)$. SORT takes $O(n \log n)$ time; REDUCE-DUPLICATES takes $O(n)$ time; for line shapes, DIFFERENCE takes time linear in the input size; MAXIMAL takes $O(n \log n)$ time, and all other procedures take constant time. The $-$ operator, used to remove the first two elements of a set, is achieved in constant time. Thus, the time complexity of procedure CLASSIFY-LINE with input size n is $O(n \log n)$. The following then holds:

- (1) *The inner and shared segments of a line embedded within the carrier of a plane shape F , with respect to F , can be determined in $O(n \log n)$ time and $\Theta(n)$ space where $n = |\text{boundary}[F]|$.*

The following result is immediate:

- (2) *The line segments of intersection for a pair of plane shapes F and G , each coequal, can be determined in $O(n \log n)$ time and $\Theta(n)$ space where $n = |\text{boundary}[F]| + |\text{boundary}[G]|$.*

This can be shown by considering procedure INTERSECTION (Figure 5) with two plane shapes F and G , each coequal, as input. The result is the shape of line segments of intersection from F and G . If the carrier planes of F and G are either identical or parallel, the resulting shape of intersection is empty. Otherwise, we define the (infinite) line of intersection and use CLASSIFY-LINE to find the inner and shared segments of this line with each shape (steps 2, 3 and 6). The product of the two sets of segments is the set of line segments of intersection of both shapes (which is returned in step 9). The line of intersection l of two non-parallel planes f and g is easily determined from their normal vectors. The time complexity stems from the fact that PARALLEL and INTERSECTION-LINE each take constant time, CLASSIFY-LINE takes $O(n \log n)$ time, and returns a shape of $O(n)$ size. Procedures SUM and PRODUCT each take linear time in the size of their input, for shapes of line segments.

Figure 5 INTERSECTION: lines of intersection of two plane segments or shapes (each coequal)

```

INTERSECTION (F, G)
1  return  $\emptyset$  if PARALLEL (co[F], co[G])
   Otherwise, find the line of intersection
2  co-l  $\leftarrow$  INTERSECTION-LINE (co[F], co[G])
3  (IF, MF)  $\leftarrow$  CLASSIFY-LINE (co-l, F)
4  MAXIMAL (MF)
   IF is necessarily maximal
5  IF  $\leftarrow$  SUM (IF, MF)
6  (IG, MG)  $\leftarrow$  CLASSIFY-LINE (co-l, G)
7  MAXIMAL (MG)
   IG is necessarily maximal
8  IG  $\leftarrow$  SUM (IG, MG)
9  return PRODUCT (IF, IG)

```

In a few cases we are interested in finding the intersection of a plane segment with a plane (e.g., see CLASSIFY-FACE in Figure 20). Procedure INTERSECTION-2 (Figure 6) takes as input, a plane segment and the co-descriptor of a plane. The complexity is necessarily the same as that of CLASSIFY-LINE.

Figure 6 INTERSECTION-2: intersection of a plane segment with a plane

```

INTERSECTION-2 (f, co-g)
1  return  $\emptyset$  if PARALLEL (co[f], co-g)
   Otherwise, find the line of intersection
2  co-l  $\leftarrow$  INTERSECTION-LINE (co[f], co-g)
3  return CLASSIFY-LINE (co-l, {f})

```

Another instance of the classification problem arises when classifying a boundary line segment with respect to a coequal plane shape. We consider procedure

CLASSIFY-EDGE (Figure 7) with input a line segment l that is a boundary segment to some plane shape and a coequal plane shape F . The results of the procedure are the classes of inner, outer, same-shared and oppositely-shared segments of l with respect to F . We borrow steps from CLASSIFY-LINE with arguments $co[l]$ and F to determine the inner and shared line segments with co-descriptor $co[l]$. Taking the product of these inner segments with l yields the class of inner segments of l . The class of same-shared segments results from the product of l and the boundary segments k of F for which $inside[k]$ equals $inside[l]$. Similarly, the class of oppositely-shared segments equals the product of l and those boundary segments k of F for which $inside[k]$ differs from $inside[l]$. Finally, the class of outer segments of l equals the difference of l with the classes of inner and shared segments.

Figure 7 CLASSIFY-EDGE: classifying a boundary line segment l with respect to a coequal plane shape F

```

CLASSIFY-EDGE (l, F)
1  I ← M ← N ← P ← ∅
2  co-g ← NORMAL-PLANE (co[l], co[F])
3  for each line segment k ∈ boundary[F]
4      if co[k] = co[l]
5          Find out type of shared segment
6          then if inside[k] = inside[l]
7              then M ← M + PRODUCT ({k}, {l})
8              else N ← N + PRODUCT ({k}, {l})
9          else p ← INTERSECTION-2 (k, co[l])
10             if p ≠ 0 and (DOT-PRODUCT (co[tail[k]], co-g) > 0
11                 or DOT-PRODUCT (co[head[k]], co-g) > 0)
12                 then P ← P + {p}
13 SORT (P)
14 Remove duplicate points
15 REDUCE-DUPLICATES (P)
16 Procedure A: an alternating sequence of inner and outer segments
17 for each point p ∈ P
18     I ← I + { LINE-SEGMENT (p, next[p]) }
19     P ← P - { p, next[p] }
20 I is necessarily maximal
21 MAXIMAL (M)
22 R ← DIFFERENCE ({l}, M)
23 MAXIMAL (N)
24 R ← DIFFERENCE (R, N)
25 I ← PRODUCT (R, I)
26 O ← DIFFERENCE (R, I)
27 return (I, M, N, O)

```

The complexity of CLASSIFY-EDGE is easily determined. Let n denote the size of the boundary of F . Procedures PRODUCT, DIFFERENCE and MAXIMAL applied to line shapes all take time linear in the size of their arguments. Therefore, PRODUCT ($\{k\}$, $\{l\}$) (see steps 6 and 7) takes constant time, while all other instances take $O(n)$ time. It follows then that the complexity is identical to the complexity for procedure CLASSIFY-LINE; from which, we claim that:

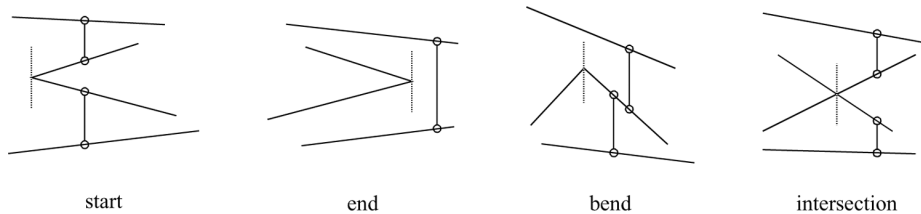
- (3) *Classifying a boundary line segment with respect to a coequal plane shape F with the carrier of the line segment coincident with the carrier of F takes $O(n \log n)$ time and requires $\Theta(n)$ space where $n = |\text{boundary}[F]|$.*

A simple generalisation of the above yields an $O(n^2 \log n)$ algorithm to classify the boundary of one shape with respect to another shape. We use a plane-sweep algorithm (Bentley and Ottman, 1979) to determine the points of intersection of the boundary line segments of shapes F and G and to classify the split segments with respect to the other shape. Consider a vertical line sweeping the carrier plane of F and G from left to right. At any position, the *sweep-line* defines a cross section of the shape composed of the line segments of F and G . Define the topology of a cross-section to be the ordering of these segments intersecting the sweep-line, about this cross-section. This topology remains invariant except at a finite number of *transition points*. These include the endpoints of the line segments and the points of intersection of segments from F and G . Two consecutive transition points define a *slice* of the plane. The topology of each slice is encoded in the *status* of the sweep-line, at the time it sweeps this slice. This status is updated at each transition point.

Initially, we consider only endpoints as transition points. For line segment l with endpoints $\text{tail}[l]$ and $\text{head}[l]$, we say that l leaves from $\text{tail}[l]$ and arrives at $\text{head}[l]$. Then, two line segments intersect at a point p only if the two segments are either consecutive, within the slice that immediately precedes transition point p , or separated by one or more segments arriving at p . In the former case, an inspection of the sweep-line status at the preceding transition point, after the update, reveals both line segments to be consecutive. In the latter case, p constitutes an initial transition point. At each transition point, the status of the sweep-line is updated by removing the line segments arriving at p and, subsequently, inserting the line segments leaving from p , in the correct order. The only segments affected by this update, with respect to possible forthcoming intersection points, are the two segments immediately above and below p , not containing p .

Nievergelt and Preparata (1982) distinguish four basic types of transition points: *start*, *end*, *bend* and *intersection*. Figure 8 illustrates, for each basic type, the pairs of line segments that need to be examined for forthcoming intersection points. These consist of the two segments immediately above and below p , not containing p , together with their immediate predecessor (below) and successor (above), respectively. As a result, at most two new transition points may be revealed at each transition point.

Figure 8 Pairs of line segments to be examined for forthcoming intersection points, for each of the four basic types of transition points



A plane-sweep algorithm operates on two basic structures: the *task schedule*, which contains the sorted list of transition points known thus far, and the sweep-line status. Let H denote the structure representing the task schedule. Transition points are removed as these are processed, in order, and, upon determination, the intersection points are

inserted. Thus, the structure H requires the functionality of extracting the minimum element from H , inserting an element into H , and checking membership in H . Such a structure may be considered as a *priority queue* (Cormen et al., 1990). Let D denote the structure representing the sweep-line status. At each transition point p , the line segments arriving at p are removed from, and the line segments leaving from p are inserted into D . Line segments intersecting at p are split and, subsequently, replaced by their right sub-segments, in reverse order. Then, the left sub-segments as well as any removed segments are classified into the classes of inner, outer, same-shared and oppositely-shared segments with respect to either shape. Thus, the structure D is a dynamic set that requires the operations of *insert*, *delete*, *search* and *reverse* to be supported.

The line segments that define the status of the sweep-line, partition this line into inner and outer segments, by Procedure A. Similarly, by the same procedure, regions of the plane defined by these line segments, in the proximity of the sweep-line, can be classified into *inside* and *outside* regions, with respect to either shape. Then, a split line segment is deemed inner if it lies in a region classified as inside with respect to the shape that does not contain this segment, and outer if it lies in an outside region. Two coincident segments, belonging to different shapes, are deemed same-shared if either region, neighbouring the coincident segments, is classified equal with respect to both shapes, and are deemed oppositely-shared otherwise.

The preceding discussion is summarised in procedure CLASSIFY (Figure 9) with input two coequal plane shapes F and G . The results of the procedure are the classes of inner and outer segments of each shape's boundary with respect to the other shape and the classes of same-shared and oppositely-shared boundary segments of both shapes.

Figure 9 CLASSIFY: classifying two coequal plane shapes (or segments)

```

CLASSIFY (F, G)
1  R ← ∅
2  shared ← FALSE
3  H ← the endpoints of the boundary segments from F and G
   H is sorted lexicographically according to their X- and Y-coordinates
   For each point p in H, L[p] is the set of boundary segments leaving p
4  D ← {−∞, ∞}
   For each transition point in the plane sweep
5  while H ≠ ∅
6     p ← EXTRACT-MIN (H)
7     (low, high) ← NEIGHBOURING-SEGMENTS (D, p)
   For the segments above and below p
8     for each l ∈ D between low and high, low and high not included
9         m ← succ[l]
10        if head[l] = p
11            then k ← l
12                DELETE (D, l)
13            else k ← CREATE-SEGMENT (tail[l], p)
14                tail[l] ← p
15                CLASSIFY-SEGMENT (R, k, m, shared)
16        REVERSE (D, succ[low], pred[high])
17        INSERT (D, l) for each l ∈ L[p]
18        tagF ← tag[low, F]
19        tagG ← tag[low, G]

```

Figure 9 CLASSIFY: classifying two coequal plane shapes (or segments) (continued)

For the segments above and below p determine inside and outside tags

```

20  for each  $l \in D$  between low and high, low and high not included
21      if  $l \in F$ 
22          then tagF  $\leftarrow$  TOGGLE (tagF)
23          tagG  $\leftarrow$  TOGGLE (tagG)
24          tag[l, F]  $\leftarrow$  tagF
25          tag[l, G]  $\leftarrow$  tagG
26      q  $\leftarrow$  INTERSECT (low, succ[low])
27      INSERT (H, q) if not ELEMENT (H, q)
28      q  $\leftarrow$  INTERSECT (high, pred[high])
29      INSERT (H, q) if not ELEMENT (H, q)
30  return R

```

Find the segments immediately above and below the transition point, p

```

NEIGHBOURING-SEGMENTS (D, p)
1  k  $\leftarrow$  SEARCH (D, p, COMPARE-POINT-WRT-LINE)
2  k  $\leftarrow$  pred[k] if COMPARE-POINT-WRT-LINE (p, k)  $\leq$  0
3  l  $\leftarrow$  succ[k]
4  l  $\leftarrow$  succ[l] while COMPARE-POINT-WRT-LINE (p, l) = 0
5  return (k, l)

```

Classify a split segment with respect to the other shape

```

CLASSIFY-SEGMENT (R, k, l, shared)
1  ( $I_F, I_G, M, N, O_F, O_G$ )  $\leftarrow$  R
2  if shared
3      then shared  $\leftarrow$  FALSE
4      else if tail[k] = tail[l]
5          then shared  $\leftarrow$  TRUE
6          if tag[k, F] = tag[k, G]
7              then Insert k into M
8              else Insert k into N
9          else if k  $\in F$ 
10             then if tag[k, G] = INSIDE
11                 then Insert k into  $I_F$ 
12                 else Insert k into  $O_F$ 
13             else if tag[k, F] = INSIDE
14                 then Insert k into  $I_G$ 
15                 else Insert k into  $O_G$ 
16  R  $\leftarrow$  ( $I_F, I_G, M, N, O_F, O_G$ )

```

Toggling a segment with respect to the region

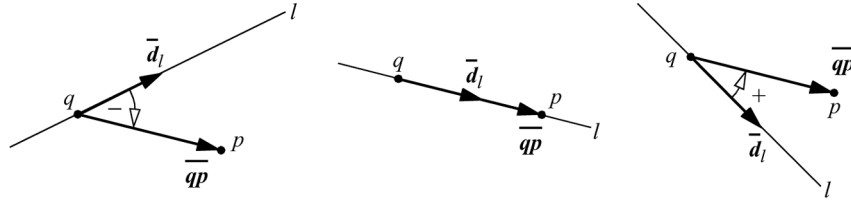
```

TOGGLE (tag)
1  return OUTSIDE if tag = INSIDE
2  return INSIDE

```

Procedure NEIGHBOURING-SEGMENTS (see Figure 9) determines the segments immediately above and below the current transition point, not containing this point. It invokes procedure COMPARE-POINT-WRT-LINE, which compares the location of a point p with respect to a line (segment) l . In particular, it returns the direction. This results in 0 if p lies on l , a positive value if p lies ‘above’ l , and a negative value otherwise (Figure 10).

Figure 10 Comparison of the location of a point p with respect to a line segment l by checking the sign of the cross-product $\vec{d}_l \times \vec{qp}$



If we associate each region, as defined by two consecutive line segments in the status of the sweep-line, with the lower of these two bounding segments, then, we can augment the status of the sweep-line with the classifications of each region (whether it is inside or outside with respect to either shape). $tag[l, F]$ denotes the classification of the region associated with l , with respect to F , which, during a split, is alternatively TOGGLE'd from INSIDE to OUTSIDE and vice versa (see CLASSIFY, steps 22 and 23). CLASSIFY-SEGMENT utilises this information to classify each split segment with respect to the other shape.

The operations, on the structure H , of extracting the minimum element (EXTRACT-MIN), inserting an element (INSERT), and checking membership are bounded by $O(\log |H|)$ time, where $|H|$ denotes the size of H , when H is implemented as a heap or balanced tree. Procedures INSERT, DELETE and SEARCH, on the structure D , take time bound $O(\log |D|)$, when D is implemented as a balanced tree or a splay tree (Sleator and Tarjan, 1985). By augmenting the tree structure to include predecessor ($pred[]$) and successor ($succ[]$) links, the procedure REVERSE takes time linear in the number of segments to be reversed.

Each boundary segment of F and G is inserted into and removed from D exactly once. Let n denote the total number of boundary segments of both F and G . At each transition point, D contains at most $n + 2$ segments. Thus, insertion and deletion of the n segments take $O(n \log n)$ time. Let m denote the number of intersection points between segments of F and segments of G ($m = O(n^2)$). Then, updating the sweep-line status at the m intersection points takes time linear in m . Initialising the task schedule H is performed in $O(n \log n)$ time. The total number of transition points is at most $m + n$ and therefore, processing the task schedule takes $O((m + n) \log (m + n))$ time. CLASSIFY-SEGMENT takes constant time. Thus, the entire plane-sweep, given m and n as defined above, takes $O((m + n) \log n)$ time with $m = O(n^2)$.

This gives the main result of this section:

- (4) *The boundaries of two coequal plane shapes F and G can be classified with respect to each other in $O((m + n) \log n)$ time and $\Theta(m + n)$ space where $n = |boundary[F]| + |boundary[G]|$, and $m = O(n^2)$ is the number of intersection points between the boundary segments of F and G .*

Remark: The resulting shapes I_F, I_G, O_F, O_G, M and N that correspond respectively to the classes of inner and outer (with respect to either shape), same-shared and oppositely-shared segments are not necessarily maximal. On the other hand, applying procedure MAXIMAL to each of these line shapes will not affect the asymptotic running time.

On practical considerations for classifying boundary shapes

It is important to note that the algorithms can be improved to run faster in practice without affecting their asymptotic time bounds. For instance, through preprocessing, line (and plane) segments can be arranged as interval trees (de Berg et al., 1997) to eliminate from consideration those pairs of segments that definitely will not intersect. More specifically, CLASSIFY-EDGE can be improved by storing, in P , just those intersection points that lie between $tail[l]$ and $head[l]$ while keeping a tally of the intersection points to the left of $tail[l]$. This count determines whether or not the first segment starting at $tail[l]$ is inner or outer with respect to F . We can still use Procedure A to classify all subsequent segments. However, this tactic will not alter the complexity of the algorithm. Such improvements and other considerations such as effective storage management, while essential for practical implementations, neither add to the understanding of the algorithms nor alter their complexity.

3.2 Constructing the boundary of a plane shape

Once the given plane shapes have been classified, the shape resulting from a shape operation has to be constructed from its defining segments. In the procedures below we assume that we are always given a set of line segments that forms the boundary of a plane shape.

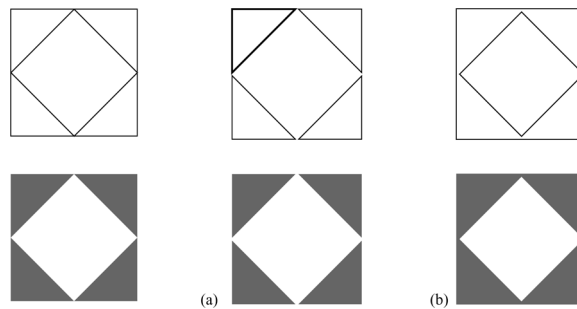
Construction of plane segments

The first instance of the problem occurs when extracting simple boundaries from a set of non-intersecting boundary line segments. We assume that the given set of line segments forms the boundary of a plane shape. We assume that the given line segments neither intersect (except at endpoints), nor overlap, although they may be coincident (i.e., identical). The result is a division of the given segments into subsets of line segments each of which defines a simple boundary as a maximal line shape.

However, there may not be a unique solution to the partitioning of a set of line segments into (non-intersecting) simple boundaries. Figure 11 illustrates an example of a plane shape that allows for two distinct interpretations of its boundary:

- made up of four outer boundaries
- made up of an outer and one inner boundary.

Figure 11 Two possible interpretations for the boundary of a plane segment: either (a) four outer boundaries or (b) one outer and one inner boundary. For a maximal segment representation of shapes we adopt interpretation (a)



For maximal segments, we employ the former interpretation. We note that two polygons are allowed to share more than one endpoint only when they both represent boundaries of the same type, either inner or outer.

We describe procedure EXTRACT-POLYGONS (Figure 12) that satisfies the requirement on the input line segments as well as the interpretation of Figure 11(a). However, line segments may be coincident, that is, cases such as those shown in Figure 13 are possible. The need to handle these cases derives from procedure SPLIT which applies to plane segments (see Figure 30). In Figure 13(a), the result of procedure EXTRACT-POLYGONS is a set of two simple boundaries that overlap. In Figure 13(b), each pair of coinciding segments results in a trivial boundary cycle that is removed accordingly (see step 23 of procedure CYCLES in Figure 12). The following outline formalises this distinction. It assumes that all simple boundaries are traversed in a counterclockwise manner.

Figure 12 EXTRACT-POLYGONS: extracting the simple boundaries for a plane shape using a depth-first search

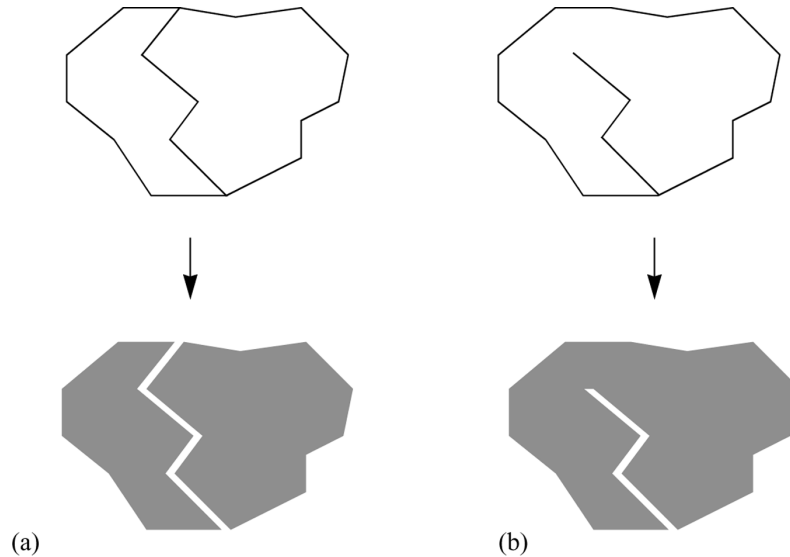
```

EXTRACT-POLYGONS (L)
1  C ← ∅
   Build the graph corresponding to L in sorted order
2  G ← ADJACENCY-GRAPH (L)
3  for each vertex v ∈ V[G]
4      C ← C + CYCLES (G, v) while Adj[v] ≠ ∅
5  for each shape L ∈ C
6      MAXIMAL (L)
7  return C

Determining the simple cycles of a planar graph
CYCLES (G, u)
1  P ← C ← ∅
2  v ← u
3  PUSH (P, v)
4  (v, w) ← STARTING-EDGE (v)
   Remove directed edge (v, w) from E[G]
5  Adj[v] ← Adj[v] - {w}
6  while P ≠ ∅
7      PUSH (P, w)
8      (w, t) ← CONTINUATION-EDGE (w, (w, v))
9      Adj[w] ← Adj[w] - {v}
10     Adj[w] ← Adj[w] - {t}
11     v ← w
12     w ← t
   If a cycle is found
13     if w ∈ P
14         then L ← ∅
15         while P ≠ ∅ and w ≠ top[P]
16             l ← LINE-SEGMENT (point[t], point[top[P]])
17             if tail[l] = point[t]
18                 then inside[l] = 1
19                 else inside[l] = -1
20             L ← L + {l}
21             t ← POP (P)
22         L ← L + {LINE-SEGMENT (point[t], point[w])}
   If line segments do not coincide, insert L at the front of the set
23         C ← {L} + C if |L| > 2
24     Adj[w] ← Adj[w] - {v}
25     for each shape L ∈ rest[C]
26         for each segment l ∈ L
27             inside[l] ← -inside[l]
28     return C

```

Figure 13 The role of coinciding line segments in the determination of simple boundaries: (a) the double concatenation of coinciding line segments defines the overlap of two boundaries and (b) redundant boundary segments in a single boundary



Procedure B

- 1 *Starting from the bottom left-most endpoint, proceed along the line segment closest to the bottom in a counterclockwise order about that endpoint.*
- 2 *At each endpoint on the path, proceed along the line segment that is closest to the last segment in a clockwise order about the endpoint.*

It is important to note that the notion of clockwise and counterclockwise angle is independent of any particular line segment under consideration. For three points p , q and r , the angle $\angle qpr$ is said to be *counterclockwise* if $\mathbf{pq} \times \mathbf{pr} \geq 0$, and *clockwise*, otherwise. Thus, for any set of coplanar line segments, the vector product of the direction vectors of any two line segments is a vector with a fixed direction (apart from the sign).

Consider three line segments a , b and c , no two overlapping, with a common endpoint p . Let the order of the line segments about p be represented as a triple, such that (a, b, c) be a clockwise ordering and (a, c, b) counterclockwise. All other permutations of $\{a, b, c\}$ are cyclic permutations of these two. Thus, we need only consider cycles (a, b, c) and (a, c, b) . Either defines three angles about p (the sum of which add up to 360°) and only one can be greater or equal to 180° . Figure 14 illustrates clockwise and counterclockwise configurations in the cases when all angles are less than 180° or when a single angle is greater or equal to 180° . Table 5 formalises these results. We conclude that three line segments a , b and c are configured clockwise about a common endpoint if at least two of the angles $\angle ab$, $\angle bc$ and $\angle ca$ are clockwise (cases i–iv), and otherwise, are configured counterclockwise (cases v–viii).

Figure 14 Clockwise (cases i–iv) and counterclockwise (cases v–viii) configurations of three line segments a , b and c about a common endpoint. Cases i and v indicate the situations when all angles are less than 180° . Cases ii–iv and vi–viii indicate the situations when a single angle is greater or equal to 180°

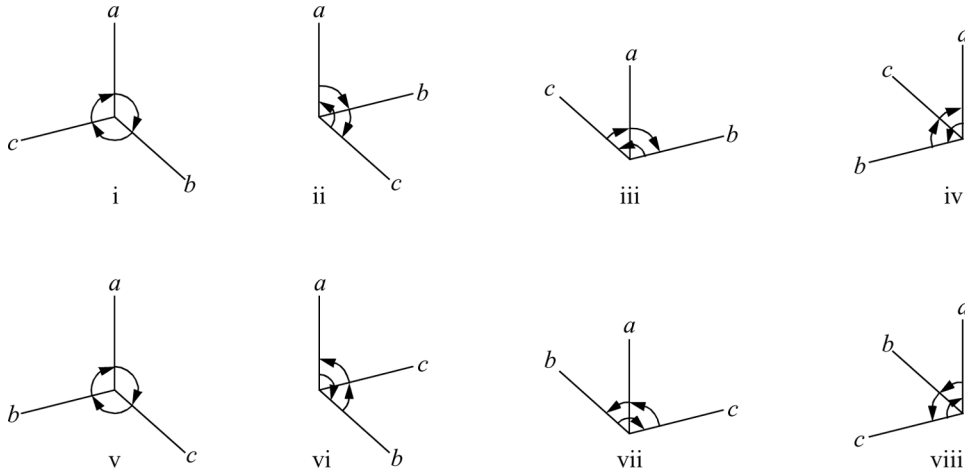


Table 5 Combinations of clockwise \curvearrowright and counterclockwise \curvearrowleft angles for $\angle ab$, $\angle bc$ and $\angle ca$ grouped with respect to the overall ordering (a, b, c) or (a, c, b)

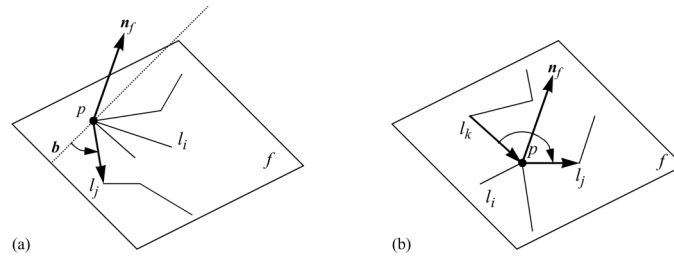
	$\angle ab$	$\angle bc$	$\angle ca$
(a, b, c)			
i	\curvearrowright	\curvearrowright	\curvearrowright
ii	\curvearrowright	\curvearrowright	\curvearrowleft
iii	\curvearrowright	\curvearrowleft	\curvearrowright
iv	\curvearrowleft	\curvearrowright	\curvearrowright
(a, c, b)			
v	\curvearrowleft	\curvearrowleft	\curvearrowleft
vi	\curvearrowright	\curvearrowleft	\curvearrowleft
vii	\curvearrowleft	\curvearrowright	\curvearrowleft
viii	\curvearrowleft	\curvearrowleft	\curvearrowright

Let $l_i, i \geq 1$, denote all line segments that have p as an endpoint. For each segment l_i , let q_i denote the other endpoint ($q_i \neq p$). Consider step 2 in Procedure B. If the last segment is l_k , then the continuation segment is l_j ($j \neq k$) if and only if (l_k, l_j, l_i) defines a clockwise ordering for all $i \neq j, i \neq k$.

The bottom left-most endpoint is the greatest lower bound of the set of all endpoints under lexicographical ordering, \leq_c . Let p denote the bottom left-most endpoint. Let f denote the carrier plane of all line segments. Consider the line of intersection of f and a plane parallel to YZ through a point p . If f is parallel to YZ , then consider the line of intersection of f and a plane parallel to XZ through p . This line of intersection defines the

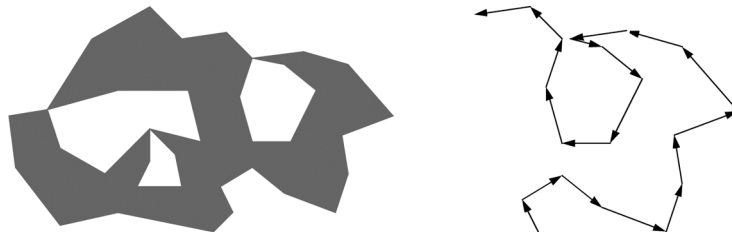
bottom direction b (see Figure 15(a)). b denotes the direction vector of this line. Irrespective of its sign, for any line segment l_i with endpoint p , one of the angles $\angle bl_i$ and $\angle l_i b$ is counterclockwise and the other clockwise. Therefore, the starting segment is l_j if and only if (b, l_j, l_i) defines a counterclockwise ordering, i.e., $\angle l_j l_i$ is counterclockwise, for all $i \neq j$.

Figure 15 Two examples illustrating the (a) start and (b) continuation steps in procedure B to extract the simple polygons from a set of boundary segments



The greedy approach outlined above yields the smallest enclosed surface for the given starting segment. Figure 15 illustrates both steps in the algorithm. Let G denote the graph derived from the set of line segments by associating a vertex with each (unique) endpoint and an undirected edge with each line segment. Then, the simple boundaries correspond to simple cycles in the planar graph G , which are extracted using a depth-first search on the graph. Starting from the bottom left-most vertex and proceeding as described above, a cycle or boundary is found when a vertex or point is reached that has been visited earlier in the traversal. If this vertex is the starting vertex, the traversal is concluded; other cycles may then be found by traversing the remaining graph, from a (possibly) new vertex. Otherwise, the search is continued to find other cycles, until the starting vertex is reached. If more than one cycle is determined within a single traversal, all but the last one necessarily represent inner boundaries for the shape defined by the boundaries from this traversal (see Figure 16). However, since all are treated as outer boundaries, the insides of the inner boundaries' line segments are inverted accordingly. For each line segment l added to the cycle, its inside is indicated by $\mathbf{n}_f \times \mathbf{d}_l$ provided l has retained the direction given in the counterclockwise traversal (see steps 17–19 of procedure CYCLES, Figure 12). Let $V[G]$ denote the vertex set and $E[G]$, the edge set. Each undirected edge is represented as two directed edge-halves, in opposite directions; each directed edge-half (u, v) is defined as an entry in the adjacency list, denoted $Adj[u]$, of the vertex u .

Figure 16 An exemplar result of the procedure CYCLES, consisting of four simple cycles of which all but one represent an inner boundary for the defined shape. Shown in detailed is part of the traversal



The complexity of EXTRACT-POLYGONS is easily determined. Let n denote the number of line segments. Procedure STARTING-EDGE takes time linear in the size of the adjacency list. Determining the continuation edge with respect to the current edge (procedure CONTINUATION-EDGE) is achieved in constant time, so too is the deletion of an edge from $E[G]$ (i.e., from an adjacency list). All other steps in procedure CYCLES take constant time. Each time a vertex is pushed on the stack P , an edge is traversed and both directed edge-halves are consequently removed from the graph. Each vertex popped from the stack results in a single line segment inserted into the cycle L . Extracting a single cycle c takes $O(|c|)$ time, and summed over all cycles or simple boundaries, the total time taken is $O(n)$. Since $|V[G]| \leq |E[G]|$, initialisation of the graph (ADJACENCY-GRAPH) requires $\Theta(n \log n)$ time; procedure MAXIMAL takes $O(n \log n)$ time in total. Therefore:

- (5) *For a set L of non-intersecting (except at their endpoints) line segments that specifies the boundary of a plane segment or plane shape F , constructing the simple boundaries that specify the plane segments of F takes $\Theta(n \log n)$ time and requires $\Theta(n)$ space where $n = |L|$.*

The next step is to distinguish the outer and inner boundaries from a set of simple boundaries and thereby, construct the corresponding plane segments. EXTRACT-POLYGONS gives the simple boundaries. CLASSIFY-POLYGONS, implemented using a plane-sweep, performs the classification of these boundaries. Thus, we have the two-step procedure for CONSTRUCT giving a maximal plane segment representation of a plane shape. The algorithm is shown in Figure 17. L is a set of line segments, in which neither the lines intersect (except at their endpoints) nor overlap, though may be coincident, that forms the boundary of a plane shape. CLASSIFY-POLYGONS takes $O(n \log n)$ time where n denotes the number of line segments in the subsets of L . Hence:

- (6) *For a set L of non-intersecting (except at their endpoints) line segments that defines the boundary of a plane segment or plane shape F , constructing the plane segments that make up F takes $\Theta(n \log n)$ time and requires $\Theta(n)$ space where $n = |L|$.*

Figure 17 CONSTRUCT: construct maximal plane segment representation from its set of boundary line segments

```

CONSTRUCT (L)
1  L ← EXTRACT-POLYGONS (L)
2  return CLASSIFY-POLYGONS (L)    using a plane-sweep

```

EXTRACT-POLYGONS can be modified so as to classify cycles as they are returned by procedure CYCLES. Consider the set of all simple boundaries extracted from the set L of line segments and the shape defined by this set of boundaries. Because of the choice of the starting segment, the first cycle returned represents an outer boundary for the defined shape and all other cycles returned from the same traversal represent inner boundaries. Consider the remaining set of simple boundaries and the shape defined by this set. Again, because of the choice of the starting segment, the first cycle returned represents an ‘outer’ boundary for the defined shape and all other cycles returned from the same traversal represent ‘inner’ boundaries. Moreover, this ‘outer’ boundary is either an outer boundary for the overall shape, in which case nothing changes, or an inner boundary, in which case

the ‘inner’ boundaries become outer boundaries for the overall shape. Thus, each set of cycles returned by procedure CYCLES can be classified at that moment, independently of any cycles subsequently found (see arguments leading up to (12) on a similar approach to the classification of simple boundaries in U_2). In the worst case, each traversal determines only a single cycle that has to be classified with respect to all of the cycles previously found. In such a case, the plane-sweep yields the best overall result.

We can use the previous result when determining the maximal shape given a set of plane segments that may share boundary (but neither overlap nor one contains another). Procedure CLASSIFY (see Figure 9) when applied to (sets of) plane segments, takes as arguments two coequal (maximal) plane shapes. We know that the boundary segments of a maximal shape do not overlap (nor one contains another) nor intersect. As such, at any transition point (in the plane-sweep) at most two line segments intersect, one from each shape. However, no such assumption is included in the algorithm to procedure CLASSIFY. For example, when examining the segments immediately above and below the transition point for forthcoming intersection points, no distinction is made as to whether both segments that may intersect belong to different shapes or not (see steps 26 and 28 of CLASSIFY). Also, no restriction is made on the number of line segments that are split at each transition point (steps 8–15), nor the number of segments reversed (step 16). Therefore, if either shape contains intersecting boundary segments, their points of intersection are found and inserted into the list of transition points and, subsequently, the respective segments split. Procedure SPLIT is a variation on procedure CLASSIFY: it takes as its argument a single set of line segments and converts this into a set of non-intersecting segments, at the same time, extracting overlapping segments (as they are classified either same-shared or opposite-shared). Note that it is possible to combine the functionality of procedures SPLIT and CONSTRUCT into a single plane-sweep.

This is summarised in procedure MAXIMAL (Figure 18) with input a set F of possibly overlapping plane segments; the result is the plane shape as a set of maximal plane segments upon removing pairs of coincident segments. We assume the set F is sorted such that all coequal segments are consecutive and all classes are in the correct order.

Figure 18 MAXIMAL: converts a (sorted) set of plane segments into its maximal segment representation

```

MAXIMAL (F)
1  G ← ∅
2  for each class  $C_F \subset F$ 
3    (L, M) ← SPLIT (boundary[ $C_F$ ])
4    G ← G + CONSTRUCT (L)
5  F ← G

```

Let n denote the number of line segments in L . Let m denote the number of intersection points between segments of L ($m = O(n^2)$). Procedure SPLIT has the same computational complexity as procedure CLASSIFY when applied to plane shapes, i.e., $O((m + n) \log n)$. The same time bound also holds for the procedure CONSTRUCT. Whence:

- (7) Converting a (sorted) set F of plane segments into its maximal segment representation takes $\Theta((m + n) \log n)$ time and requires $\Theta(m + n)$ space, where $n = |\text{boundary}[F]|$ and m is the number of points of intersection of the boundary line segments of the plane segments of F .

4 Part II: Classifying the boundary of a volume shape

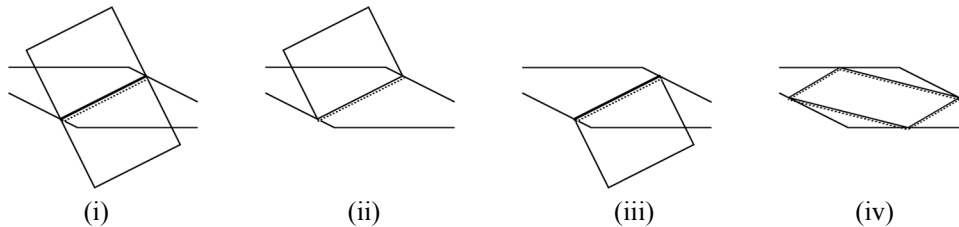
4.1 Classification of plane segments

Unlike line segments, the boundary of a plane segment does not have a fixed size. However, given a volume shape S , the total number of boundary line segments of boundary plane segments of S is related to the number of boundary plane segments of S . For a manifold solid, by the Euler-Poincaré equation: $v - e + f = 2(s - g)$, where v, e, f, s and g are respectively the number of vertices, edges, faces, shells and genus (handles), we have $f < e$. Thus, $f = \Theta(e)$. We represent volume segments and shapes as manifolds; therefore, $n = \Theta(e)$, $m = \Theta(f)$ and, thus, $m = \Theta(n)$. In other words, for a volume shape S , $|\text{boundary}[S]| = \Theta(|\text{boundary}[\text{boundary}[S]|]$.

We consider (finite) plane segments only.

In order to classify a plane segment against a volume shape, we need to consider the line segments of intersection of a plane with the boundary segments of the shape. Figure 19 illustrates the four possible cases of a plane segment intersecting a plane. It is a simple exercise in vector arithmetic to determine whether the inside of a plane segment with respect to a boundary line segment lies on one or the other side of a plane through this line segment.

Figure 19 Four cases for the intersection of a plane segment with a plane. Cases ii–iv are degenerate. Only cases i and iii result in a line segment of intersection



We consider procedure CLASSIFY-FACE (Figure 20) with input, a plane segment f that is a boundary segment of some volume shape and a (necessarily coequal) volume shape S . The results of the procedure are the classes of inner, outer, same-shared and oppositely-shared segments of f with respect to S .

Figure 20 CLASSIFY-FACE: classifying a plane segment against a volume shape

```

CLASSIFY-FACE (f, S)
1  L ← M ← N ← ∅
2  for each plane segment g ∈ boundary[S]
   Work out the intersection of g and f according to Figure 19
3    if co[g] = co[f]
4    then if inside[g] = inside[f]
5      then M ← M + PRODUCT ({g}, {f})
6      else N ← N + PRODUCT ({g}, {f})
7    else (L', K) ← INTERSECTION-2 (g, co[f])
8      L ← L + L' if L' ≠ ∅
9      for each line segment l ∈ K
10         co-h ← NORMAL-PLANE (co[l], co[g])
11         L ← L + {l} if inside[l] * DOT-PRODUCT (co-h, co[f]) > 0
12  SORT (L)
13  REMOVE-DUPLICATES (L)
14  I ← CONSTRUCT (L)
15  MAXIMAL (M)
16  R ← DIFFERENCE ({f}, M)
17  MAXIMAL (N)
18  R ← DIFFERENCE (R, N)
19  I ← PRODUCT (R, I)
20  O ← DIFFERENCE (R, I)
21  return (I, M, N, O)

```

Let n denote the number of boundary line segments of boundary plane segments of S . Let k denote the size of the boundary of f . The procedure employs an auxiliary set of line segments L constructed in steps 7–11.

First, consider the construction of the inner segments I . For a given boundary segment g of S , let n_g denote the size of the boundary of g . Procedure INTERSECTION-2 (Figure 6), intersecting g with respect to all carriers of f , then takes $O(n_g \log n_g)$ time, for each boundary segment g , and the resulting sets L' and K have $O(n_g)$ size. Note that the line segments in K have received their *inside*[l] information from g through procedure CLASSIFY-LINE, called from within INTERSECTION-2. Summed over all boundary segments g of S , steps 7–11 take $O(n \log n)$ time and result in a set L of $O(n)$ size. Sorting the set L , removing duplicate line segments and constructing the set of plane segments I , steps 12–14, all take $O(n \log n)$ time. The resulting set I still has $O(n)$ size.

Next, consider the sets of shared segments, M and N . The product of f and g involves at most kn_g points of intersection and, therefore, takes $O(kn_g)$ time. The resulting sets M and N have $O(kn)$ size and take $O(kn)$ time to be assembled. By (7), procedure MAXIMAL (see Figure 18) applied to sets of plane segments takes $O(kn \log(kn))$ time for input size $O(kn)$. Steps 16 and 18 compute the difference of f with the classes of same-shared and oppositely-shared plane segments. Since $M \leq f$, $N \leq f$ and $M \cdot N = 0$, these steps can be replaced by the following (partial) algorithm.

```

L ← boundary[M] U boundary[N]
REMOVE-DUPLICATES (L)
L ← L U boundary[f]
REMOVE-DUPLICATES (L)
R ← CONSTRUCT (L)

```

Since $\text{boundary}[f]$ has size k and both $\text{boundary}[M]$ and $\text{boundary}[N]$ have $O(kn)$ size, the preceding algorithm takes $O(kn \log(kn))$ time.

Steps 19 and 20 compute the classes of inner and outer segments, respectively, resulting from the product and difference of R and I . Using the characteristics of the classification approach, these steps can be replaced by the following (partial) algorithm.

```
(IR, II, M, N, OR, OI) ← CLASSIFY (R, I)
L ← IR ∪ II ∪ M
I ← CONSTRUCT (L)
L ← OR ∪ II ∪ M
O ← CONSTRUCT (L)
```

The computational complexity of CLASSIFY-FACE applied to plane shapes is dominated by the number of intersection points between boundary segments from both shapes. In this specific case, these intersections can only occur between boundary segments of f and segments of I or at the vertices of S . Thus, the number of intersection points is at most $O(kn)$ and the algorithm takes $O(kn \log(kn))$ time.

- (8) *A boundary plane segment f can be classified with respect to a volume shape S into classes of inner, outer, same-shared and oppositely-shared segments in $O(kn \log(kn))$ time and $O(kn)$ space where $n = |\text{boundary}[S]|$ and $k = |\text{boundary}[f]|$.*

We next consider procedure CLASSIFY (Figure 21) with input two coequal volume shapes S and T .

Figure 21 CLASSIFY: classifying two coequal volume shapes

```
CLASSIFY (S, T)
1  IS ← IT ← M ← N ← OS ← OT ← ∅
2  for each plane segment  $f \in \text{boundary}[S]$ 
3      (I, M', N', O) ← CLASSIFY-FACE (f, T)
4      IS ← IS + I
5      M ← M + M'
6      N ← N + N'
7      OS ← OS + O
8  SORT (IS)
9  SORT (M)
10 SORT (N)
11 SORT (OS)
12 for each plane segment  $f \in \text{boundary}[T]$ 
13     (I, M', N', O) ← CLASSIFY-FACE (f, S)
14     IT ← IT + I
15     OT ← OT + O
16 SORT (IT)
17 SORT (OT)
18 return (IS, IT, M, N, OS, OT)
```

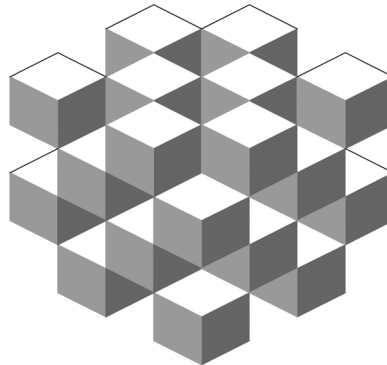
This procedure relies on CLASSIFY-FACE (see Figure 20). Each plane segment of S , quite simply, is classified with respect to T (steps 2–11) and vice versa (steps 12–17). The results of CLASSIFY are the classes of inner and outer segments of each shape's boundary with respect to the other shape and the classes of same-shared and oppositely-shared boundary segments of both shapes.

For volume shape X , let $n_X = |\text{boundary}[\text{boundary}[X]]|$. For a boundary segment f of S , let n_f denote the size of the boundary of f . Then, CLASSIFY-FACE (f, T) takes $O(n_f n_T \log(n_f n_T))$ time; summed over all boundary segments f of S this becomes $O(n_S n_T \log(n_S n_T))$, and similarly for the boundary plane segments of T with respect to S . Since the sizes of all I_S, I_T, M, N, O_S and O_T are $O(n_S n_T)$, all sorting takes $O(n_S n_T \log(n_S n_T))$ time.

- (9) *The boundaries of two coequal volume shapes S and T can be classified with respect to each other in $O(n \log n)$ time and $O(n)$ space where $n = |\text{boundary}[S]| \times |\text{boundary}[T]|$.*

We can improve upon this result by partitioning the boundary segments into coequal classes. Let n denote the size of the boundary of a volume shape S . Let k denote the number of classes upon partitioning the boundary of S into coequal classes. Even though $k = O(n)$, it does not hold that $k = \Theta(n)$. Figure 22 illustrates a volume shape where $k = \Omega(\sqrt[3]{n})$, due to Karasick (1988). The following procedure demonstrates the improvement through using this distinction. Additionally, it incorporates multiple applications of the INTERSECTION-2 procedure into a single plane-sweep algorithm.

Figure 22 A volume shape consisting of $4^{3/2} = 32$ segments (cubes), $4^3 \times 3 = 192$ boundary plane segments, but only $3 \times (4 + 1) = 15$ coequal classes of boundary segments



Source: Karasick (1988)

We consider procedure CLASSIFY (Figure 23) with input two coequal volume shapes S and T . The results of the procedure are the classes of inner and outer segments of each shape's boundary with respect to the other shape and the classes of same-shared and oppositely-shared boundary segments of both shapes.

Procedure PARTITION partitions the set of boundary segments into a set of coequal classes. Each class can be parted further into two sub-classes, denoted $in[F]$ and $out[F]$, where $in[F]$ contains all boundary segments s with $inside[s]$ equal to $+1$, and $out[F]$ contains all other segments (with $inside[]$ equal to -1). Then, the class of same-shared boundary plane segments of S and T equals the sum of the products of $in[F]$ and $in[G]$, and $out[F]$ and $out[G]$, for all coequal classes F and G in S and T , respectively (step 19). Similarly, the class of oppositely-shared boundary plane segments equals the sum of the products of $in[F]$ and $out[G]$, and $out[F]$ and $in[G]$ (step 20).

Figure 23 CLASSIFY: classifying two coequal volume shapes (improved)

```

CLASSIFY (S, T)
1  IS ← IT ← M ← N ← OS ← OT ← ∅
2  CS ← PARTITION (boundary[S])
3  CT ← PARTITION (boundary[T])
4  for each class F ∈ CS + CT
5      lines[F] ← same[co[F]] ← opp[co[F]] ← segments[F] ← ∅
   Determine line segments of intersection
6  for each class F ∈ CS
7      for each class G ∈ CT
8          co-l ← INTERSECTION-LINE (co[F], co[G])
9          if co-l ≠ 0
10             then lines[F] ← lines[F] + {co-l}
11                 lines[G] ← lines[G] + {co-l}
12 for each class F ∈ CS + CT
13     SORT (lines[F])
14     REDUCE (lines[F])
15     (inner[F], shared[F]) ← CLASSIFY-LINES (lines[F], F)
   Determine shared segments and collect line segments of intersection
16 for each class F ∈ CS
17     for each class G ∈ CT
18         if co[F] = co[G]
19             then same[co[F]] ← PRODUCT (in[F], in[G])
20                 + PRODUCT (out[F], out[G])
21                 opp[co[F]] ← PRODUCT (in[F], out[G])
22                     + PRODUCT (out[F], in[G])
23         else co-l ← INTERSECTION-LINE (co[F], co[G])
24             if co-l ≠ 0
25                 then segments[F] ← segments[F] + EXTRACT (inner[G], co-l)
26                     K ← EXTRACT (shared[G], co-l)
27                     for each line segment l ∈ K
28                         co-h ← NORMAL-PLANE (co[l], co[G])
29                         if inside[l] * DOT-PRODUCT (co-h, co[F]) > 0
30                             then segments[F] ← segments[F] + {l}
31                             segments[G] ← segments[G]
32                                 + EXTRACT (inner[F], co-l)
33                     K ← EXTRACT (shared[F], co-l)
34                     for each line segment l ∈ K
35                         co-h ← NORMAL-PLANE (co[l], co[F])
36                         if inside[l] * DOT-PRODUCT (co-h, co[G]) > 0
37                             then segments[G] ← segments[G] + {l}
   Construct inner segments, collect shared segments and determine outer segments
38 for each class F ∈ CS
39     SORT (segments[F])
40     REMOVE-DUPLICATES (segments[F])
41     I ← CONSTRUCT (segments[F])
42     R ← DIFFERENCE (F, same[co[F]])
43     R ← DIFFERENCE (R, opp[co[F]])
44     IS ← IS + PRODUCT (R, I)
45     OS ← OS + DIFFERENCE (R, I)
46     M ← M + same[co[F]]
47     N ← N + opp[co[F]]
48 for each class G ∈ CT
49     SORT (segments[G])
50     REMOVE-DUPLICATES (segments[G])
51     I ← CONSTRUCT (segments[G])
52     R ← DIFFERENCE (G, same[co[G]])
53     R ← DIFFERENCE (R, opp[co[G]])
54     IT ← IT + PRODUCT (R, I)
55     OT ← OT + DIFFERENCE (R, I)
56 return (IS, IT, M, N, OS, OT)

```

Each line of intersection between two classes of boundary segments, one from S and one from T (step 8), is classified with respect to either class to yield inner and shared segments (step 15). Procedure CLASSIFY-LINES is functionally similar to CLASSIFY-LINE (see proof of (1)), except that it classifies multiple lines with respect to a single coequal shape. However its asymptotic running time can be improved upon by using a plane-sweep approach similar to the procedure CLASSIFY applied to sets of line segments (see proof of (4)), when classifying all lines in a single sweep.

Given a class F from S and the corresponding set of inner and shared segments for all intersecting classes from the shape T , this set, upon removing the shared segments on one side of the carrier of F (see discussion leading to (8)), defines the boundary of the planar section of T by this carrier. Then, the product of F with the shape corresponding to this section determines the class of inner segments, including some shared segments. Let I denote the shape corresponding to this section (steps 38 and 48). Consider the shape R that is the difference of the class F and the shared segments of F (with respect to classes of T). The product of R and I determines the inner segments of F with respect to T (steps 41 and 51); the difference of R and I determines the outer segments of F with respect to T (steps 42 and 52). Same-shared and oppositely-shared segments are gathered from the classes of S or T (steps 43 and 44). Thus, all classes of inner, outer, same-shared and oppositely-shared segments can be found by simple arithmetic on plane shapes.

Let n denote the sum of the sizes of the boundaries of (the boundaries of) S and T . Partitioning both sets of boundary segments into coequal classes takes $O(n \log n)$ time. Let k denote the total number of classes. Determining the lines of intersection between classes of S and T takes $\Theta(k^2)$ time; classifying these lines of intersection with respect to each of its defining classes takes $O(kn \log n)$ time in total (steps 6–15). The resulting sets of line segments total $\Theta(kn)$ size. Determining the same-shared and oppositely-shared line segments takes time linear in n (steps 18–20). Procedure EXTRACT extracts the line segments with a given co-descriptor from a set of line segments that results from a call to procedure CLASSIFY-LINES. This can be done in $O(\log k)$ time if this set is subdivided into sets of coequal segments and these subsets are stored in a tree with depth $\log(k)$. Then, collecting the line segments of intersection for each class takes $O(k^2 \log k + kn)$ time in total (steps 16–34). Finally, constructing the classes of inner and outer segments for either shape, and collecting the classes of same-shared and oppositely-shared segments takes $O(kn \log n)$ time. The number of intersection points, m , is on the same order as the number of line segments of intersection. The sizes of the resulting classes is on the order of the number of boundary segments n and the number of line segments of intersection m , i.e., $O(m + n)$ with $m = O(kn) = O(n^2)$. This establishes the main result of this section:

- (10) *The boundaries of two coequal volume shapes S and T can be classified with respect to each other in $O(kn \log n)$ time and $\Theta(kn)$ space, where $k = |\text{classes}[\text{boundary}[S]]| + |\text{classes}[\text{boundary}[T]]|$, and $n = |\text{boundary}[S]| + |\text{boundary}[T]|$.*

Remark: Note that, unlike in procedure CLASSIFY (Figure 9) applied to line segments (see proof of (4)), for plane segments, the resulting shapes I_F , I_G , O_F , O_G , M and N corresponding to classes of inner and outer (with respect to both shapes), same-shared and oppositely-shared segments are necessarily maximal.

4.2 Constructing the boundary of a volume shape

Construction of volume segments

When applied to the construction of volume segments, boundary traversal becomes a tree traversal. Suppose we are given a set, say F , of plane segments that forms the boundary of a volume shape. We assume the plane segments in F neither overlap nor intersect, except at their boundary line segments. Furthermore, we assume the boundary line segments neither intersect nor overlap, but they may be coincident, that is, be identical. (This assumption ensures that each boundary line segment can be considered as a single entity, either as a part of the boundary of a segment, or disjoint from that boundary). We can divide F into subsets of plane segments, each of which defines a simple boundary as a maximal shape. We define the *horizon* in a boundary traversal to be the set of boundary line segments that have been reached but not yet completed. The following is an outline of the algorithm to create polyhedral shells of plane segments.

Procedure C

- 1 *Starting from a left-most boundary line segment, proceed along the plane segment that is closest to the bottom direction.*
- 2 *Insert the boundary line segments of the current plane segment into the horizon, remove any duplicate line segments (in the horizon).*
- 3 *Take any line segment in the horizon, proceed along the plane segment that makes the smallest inside angle with the shell, about the line segment.*
- 4 *Proceed from 2.*

Note that for three non-overlapping plane segments f , g and h that share a boundary line segment l , their ordering is identical to the configuration of the vectors \mathbf{v}_f , \mathbf{v}_g and \mathbf{v}_h about a point p on l . See Figure 24. The angle $\angle fg$ is equal to the angle defined by \mathbf{v}_f and \mathbf{v}_g about p , which is counterclockwise if $\mathbf{v}_f \times \mathbf{v}_g \geq 0$.

Consider the set S of all boundary line segments that have the bottom left-most endpoint p as an endpoint. A left-most line segment l is a segment in S that makes the smallest angle with a plane a parallel to the YZ plane through p . The angle of a line segment l with a equals the angle between l and the normal projection of l on a (see Figure 25(a)). (Since the sine of the angle is proportional to the length of the vector product it suffices to calculate the length $|\mathbf{d}_l \times \mathbf{t}|$ for each line segment l in order to find the segment with the smallest angle). The bottom direction is defined by a plane b through l and a line perpendicular to l within a . Then, \mathbf{v}_b is given by $\|\mathbf{n}_b \times \mathbf{d}_l\| = \|\mathbf{n}_a \times \mathbf{t}\|$, where \mathbf{t} denotes the normal projection of l . The selection of the starting segment proceeds in a similar way as for the boundary traversal of plane segments (see proof of (5)), that is, the selected segment makes a smallest angle, e.g., counterclockwise, with the bottom direction plane b . The inside of the shell under construction with respect to the starting segment is dependent upon the actual angle between the bottom direction plane and the starting segment f and the direction of its normal vector \mathbf{n}_f . That is, $inside[f]$ equals 1 if $\mathbf{v}_b \times \mathbf{v}_f$ and $\mathbf{v}_f \times \mathbf{n}_f$ are simultaneously positive or negative, and equals -1 , otherwise.

Figure 24 The configuration of three plane segments f , g and h about a common boundary line segment l is identical to the configuration of the vectors v_f , v_g and v_h about a point p on l , where v_f denotes a vector perpendicular to d_l and n_f within f (and likewise for g and h). Counterclockwise (+) and clockwise (-) are defined relative to d_l

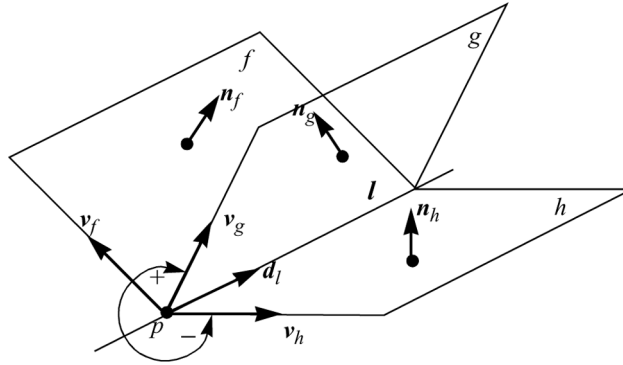
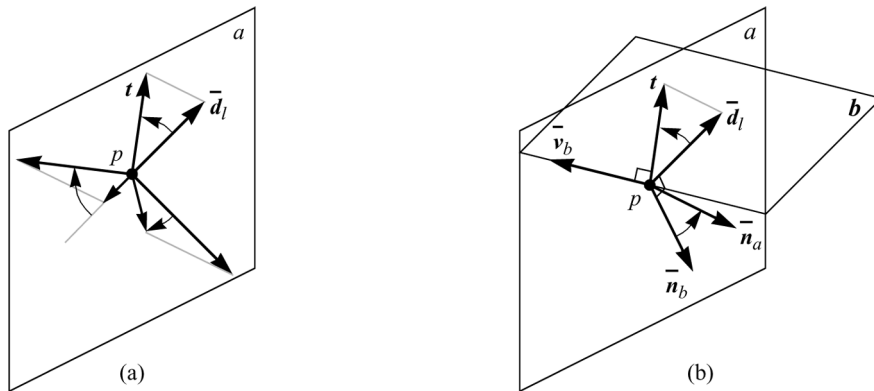


Figure 25 Illustrations of (a) a left-most line segment l and (b) the bottom direction plane b with respect to l . The plane a is parallel to YZ and contains p . l is a segment that makes the smallest angle with a . The bottom direction vector v_b is perpendicular to the direction vector of l and the normal vector of a



The determination of the continuation segment is dependent upon the inside of the plane segment that it continues from. Again, consider the configuration of plane segments f , g and h shown in Figure 24. Suppose f is the reference segment, that is, the segment continued from. Let $c_f (= \text{inside}[f] n_f)$ indicate the inside of the shell being constructed with respect to f . If c_f defines a counterclockwise angle from v_f , then the continuation segment is the segment closest to the reference in a counterclockwise order about l , e.g., segment g . The inside of the shell with respect to g is indicated by a vector c_g clockwise from v_g ; that is, $\text{inside}[g] = +1$ if $n_g = c_g = v_g \times \|d_l\|$, and -1 otherwise. Similarly, if c_f defines a clockwise angle from v_f , then the continuation segment is the segment closest to the reference in a clockwise order about l and the orientation of the inside of the shell with respect to the continuation segment is ‘counterclockwise’.

Consider the graph derived from the given set of plane segments by associating a vertex with each unique boundary line segment and an edge with each plane segment. Given that this set of plane segments constitutes the boundary of a volume shape, we say that the graph *defines* a boundary shape. We denote a *simple shell* any subgraph

that defines a simple boundary; we denote a *composite shell* any subgraph that defines a boundary shape other than a simple boundary. Thus, the polyhedral boundaries that define the volume segments of the resulting maximal shape, correspond to simple shells in the graph, that are extracted using a tree-traversal process: starting from a left-most vertex and proceeding as described in the algorithm outlined above, a shell is completed whenever the horizon is empty. However, similar to the boundary traversal in the construction of plane segments, a single traversal may not yield a simple shell, but a (composite) shell that defines a shape composed of a single outer boundary and zero, one or more inner boundaries. Unlike the two-dimensional problem, the recognition of the simple shells that make up the current construction is not an obvious task (see Figures 26 and 27).

Figure 26 Shells: (a) a shape defined by six outer boundary shells; (b) an exploded view of the same shape and (c) a shape defined by one outer and six inner boundary shells

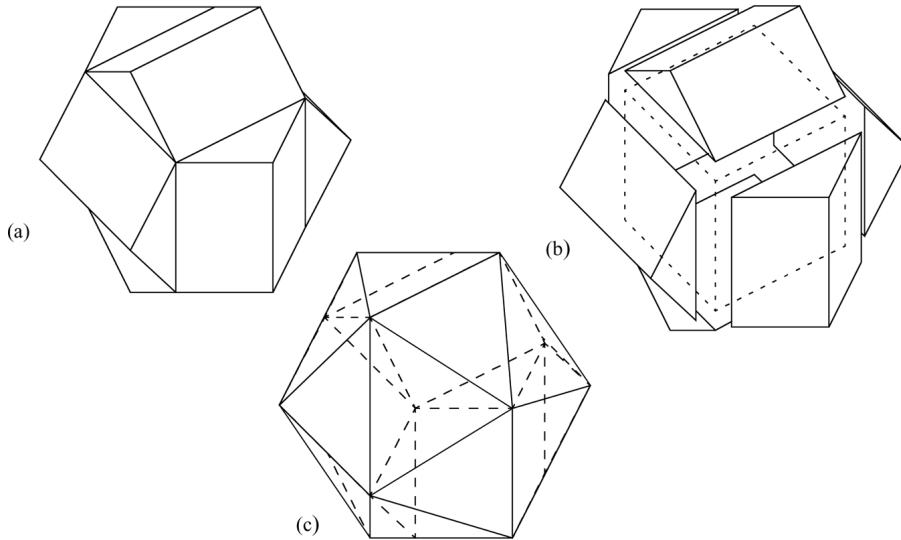
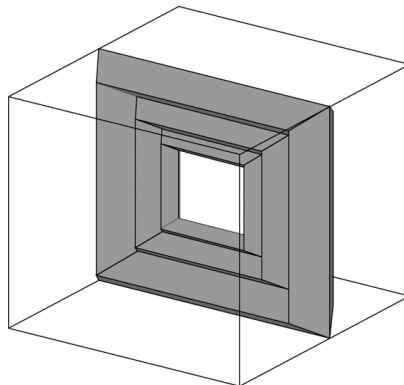
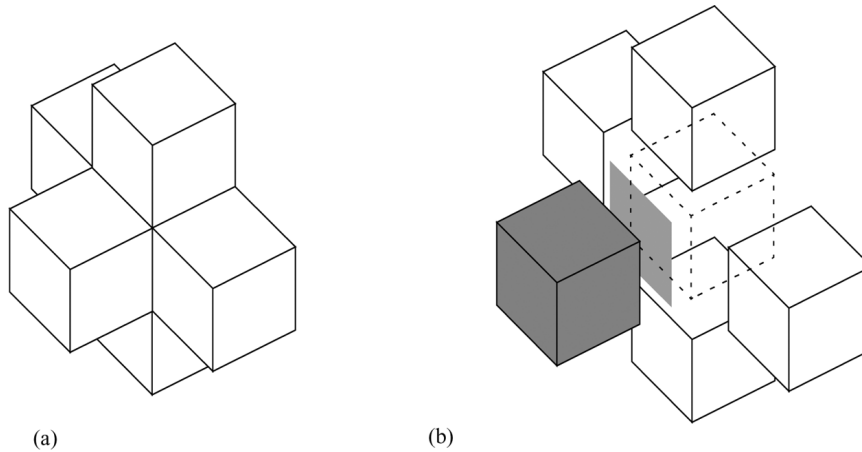


Figure 27 A shape defined by one outer boundary and three inner boundaries. Each boundary, with the exception of a single inner boundary, is constructed as two partial shells



Let the *multiplicity* of a vertex denote the number of edges it adjoins, that is, the number of boundary plane segments of which the corresponding boundary line segment is a part. The multiplicity of a vertex is, necessarily, even. Let a *partial shell* denote a maximal part of a shell that can be constructed by traversing vertices with multiplicity equal to 2, only (see Figure 28). Therefore, the horizon of a partial shell defines a closed concatenation of boundary line segments corresponding to vertices with multiplicity greater than 2. Consider the partial shells resulting from a single application of Procedure C on the given graph. Then, the combined horizon of these partial shells, under the set operation of symmetric difference, is empty. Let each partial shell be represented by a single (composite) edge and consider the (sub)graph of these edges joined by the common vertices in their horizons. Then, in a second traversal, within this (sub)graph, we consolidate partial shells that have a common vertex of multiplicity equal to 2 into new, partial or complete, simple shells (with an empty horizon). In the next cycle of the algorithm, we repeat both traversals on the remaining graph, and such until the entire graph is traversed and all simple shells are determined.

Figure 28 Partial shells: (a) a composite shell and (b) an exploded view of the constituting partial shells. The emphasised partial shells are constructed in a single cycle



For this algorithm to succeed, it is imperative that, at each step in the second traversal, a vertex of multiplicity equal to 2 exists within the current (sub)graph. Suppose no such vertex exists. Then, at least some of the boundaries must form a meta-shell as illustrated in Figure 26. However, in case (a), because of the choice of the continuation edge, the construction of each outer boundary results in a separate cycle of the algorithm. Also, in case (c), since the outer boundary is defined by a single partial shell, the construction of this outer boundary results in a separate cycle of the algorithm, after which the construction is reduced to case (a).

We consider procedure EXTRACT-POLYHEDRA (Figure 29) with input a set F of plane segments that forms the boundary of a volume shape.

Figure 29 EXTRACT-POLYHEDRA: creating a volume shape from its boundary plane segments

Extracting the simple boundaries of a volume shape
EXTRACT-POLYHEDRA (F)

```

1  S ← ∅
   Build the graph corresponding to F
2  G ← ADJACENCY-GRAPH (F)
3  for each vertex v ∈ V[G]           in sorted order
4      while Adj[v, WHITE] ≠ ∅
5          S ← S + SHELLS (G, v)
6  for each shape F ∈ S
7      MAXIMAL (F)
8  return S

```

Extracting partial shells (first traversal)
SHELLS (G, u)

```

1  R ← S ← ∅
2  e ← STARTING-EDGE (u)
3  a ← PARTIAL-SHELL (G, e, R)
   First partial shell is a part of outer shell
4  outer[a] ← TRUE
5  if horizon[a] = ∅
6  then for each plane segment f ∈ shell[a]
7      UNREGISTER (R, edge[f])
8  return {shell[a]}           a single, simple shell
9  H' ← H ← horizon[a]
10 for each vertex v ∈ H'
11     e ← CONTINUATION-EDGE (v, last-edge[v])
12     a ← PARTIAL-SHELL (G, e, R)
13     if horizon[a] = ∅
14     then S ← S + {shell[a]}
15         for each plane segment f ∈ shell[a]
16             UNREGISTER (R, edge[f])
17     else H ← H ∪ horizon[a]
18         H' ← H' ⊕ horizon[a]

```

Consolidating partial shells (second traversal)
PRIORITY-QUEUE (H, GREY)

```

20 while H ≠ ∅
21     v ← MINIMUM (H)
22     (d, e) ← Adj[v, GREY]
23     d ← RETRIEVE (R, d)
24     e ← RETRIEVE (R, e)
25     H' ← horizon[d] ∩ horizon[e]
   Remove common vertices from graph
26     for each vertex v ∈ H'
27         for each edge e' ∈ Adj[v, GREY]
28             if RETRIEVE (R, e') ∈ {d, e}
29                 then Adj[v] ← Adj[v] - {e'}
30             if Adj[v, GREY] = ∅
31                 then DELETE (H, v)
32     F ← shell[d] + shell[e]
33     H' ← horizon[d] ⊕ horizon[e]
34     if H' = ∅
35     then if outer[d] ∨ outer[e]
36         then S ← {F} + S           place outer boundary in front of list
37         else S ← S + {F}
38     for each plane segment f ∈ F
39         UNREGISTER (R, edge[f])
40     else a ← COMPOSITE-EDGE (F, H')
41         outer[a] ← outer[d] ∨ outer[e]
42         REGISTER (R, (d, a))
43         REGISTER (R, (e, a))
44 for each shape F ∈ rest[S]
45     for each segment f ∈ F
46         inside[f] ← -inside[f]
47 return S

```

Figure 29 EXTRACT-POLYHEDRA: creating a volume shape from its boundary plane segments (continued)

Extracting a partial shell as a composite edge

```

PARTIAL-SHELL (G, e, R)
1  H ← ∅ horizon as a priority queue
2  a ← COMPOSITE-EDGE ({segment[e]}, ∅)
3  outer[a] ← FALSE
4  REGISTER (R, (e, a))
5  for each line segment l ∈ boundary[segment[e]]
6      last-edge[vertex[l]] ← e
7      INSERT (H, vertex[l])
8  while H ≠ ∅ and |Adj|MINIMUM (H)| = 2
9      v ← MINIMUM (H)
10     e ← CONTINUATION-EDGE (v, last-edge[v])
11     REGISTER (R, (e, a))
12     shell[a] ← shell[a] + {segment[e]}
13     for each line segment l ∈ boundary[segment[e]]
14         if vertex[l] ∈ H
15             then Adj[vertex[l]] ← Adj[vertex[l]] - {e, last-edge[vertex[l]]}
16             DELETE (H, vertex[l])
17         else last-edge[vertex[l]] ← e
18             INSERT (H, vertex[l])
19 horizon[a] ← ORDERED-SET(H)
20 return a

```

Let G denote the graph derived from the set of plane segments by associating a vertex with each (unique) boundary line segment and an edge with each plane segment. Note that each edge links at least three vertices. Let $V[G]$ denote the vertex set and $E[G]$, the edge set. Each edge is represented as an entry in each of the adjacency lists of the vertices corresponding to the boundary line segments of the edge's plane segment. Assume each adjacency list to be a cyclic list with the edge-halves ordered clockwise about the common vertex. A colour scheme is used to distinguish the edges that partake in each stage of the construction.

Initially, all edges are white. The first traversal constructs partial shells using only white edges. When an edge becomes part of the current construction its colour is altered to grey. Upon completion of the first traversal, all grey edges compose the subgraph to be used during the consolidation process. As shells are completed, the composing edges are coloured black. These edges no longer participate in the process.

Procedure SHELLS returns a set of boundary shells, which is composed of a single outer boundary and zero, one or more inner boundaries; PARTIAL-SHELL extracts a partial shell as a composite edge containing two parts, a shell and a horizon. The shell is the shape of plane segments that is defined by the partial shell. The structure H is a priority queue (prioritised on the vertices' white or grey multiplicity), that represents the horizon under construction, and supports the operations INSERT, DELETE and MINIMUM. Procedure ORDERED-SET (Figure 29, PARTIAL-SHELL: step 19) converts the structure H into an ordered set, representing the horizon. This supports the set operations \cup (union), \cap (intersection) and \oplus (symmetric difference).

Procedure PRIORITY-QUEUE (Figure 29, SHELLS: step 19) rebuilds the set into a priority queue, using only the grey edges to determine the multiplicity for each vertex. For each vertex inserted in the horizon the *last-edge* field is updated to reference the edge to which the vertex belonged at the time of the insertion, as is necessary in order to determine the continuation edge at a later time.

R denotes a registration table that supports procedures REGISTER, UNREGISTER and RETRIEVE. Registration links an edge (whether composite or not) to a composite edge of which the former now makes a part. Registration removes the need for updating the edge for each of its vertices.

During the first traversal (SHELLS: steps 1–18), H' constitutes the set of vertices that have been reached (an odd number of times) but not yet completed (an even number of times). Whenever a new partial shell is constructed, its horizon is added to H' and duplicate vertices removed (through the symmetric difference operator). H constitutes the global horizon and guides the second traversal (SHELLS: steps 19–43). When two partial shells are consolidated, their horizons are merged and duplicate vertices removed. If the resulting horizon is empty, then a complete shell has been constructed. Otherwise, a new composite edge is created for the combined shell and horizon. As a result of the choice of the starting edge, we know that the first partial shell must define a part of an outer boundary (with respect to the shape defined by the current composite shell). The *outer* field of the composite edge encodes this information. As partial shells are consolidated, the outer information is passed on to the new composite edge. At any time, only one composite edge represents an outer shell.

Consider a single execution of procedure SHELLS. Let n denote the number of plane segments processed, with the number of boundary line segments equal to $\Theta(n)$. Procedure CONTINUATION-EDGE takes constant time and so do procedures COMPOSITE-EDGE, REGISTER, UNREGISTER, MINIMUM and the set operator \oplus .

Consider the calls to procedure PARTIAL-SHELL. On a priority queue of size h , procedures INSERT and DELETE take $O(\log h)$ time. Since each vertex is inserted or deleted exactly once for each edge it joins, processing the priority queue H takes $O(n \log h)$ time. Converting the queue into an ordered set takes $\Theta(h \log h)$ time for an arbitrary ordering. Since $h = O(n)$, the combined time for all calls to PARTIAL-SHELL is $O(n \log n)$.

Procedure STARTING-EDGE is invoked only once and has $O(n)$ as an upper bound. Unregistration of the edges (plane segments) of the completed shell takes $\Theta(n)$ time in total. The set operations \cup , \cap and \oplus take linear time in the size of the sets, $O(n)$. Let K denote the number of incomplete, partial shells. As the set operations are executed possibly once for each partial shell, the combined time is $O(nK)$. Each time two partial shells are consolidated, retrieving any simple edge belonging to either partial shell takes one more step. Thus, in a dumb way, retrieving an edge from the registration table takes $O(K)$ time. However, if the information is propagated backwards upon retrieving the final edge, the retrieval time can be reduced probably to almost constant time. Converting a set into a priority queue takes $\Theta(n \log n)$ time. Thus, the total time taken to execute SHELLS once is $O(nK + n \log n)$. K is a measure of the complexity of the resulting set of shells. In many instances, K will equal 1; in the worst case, K is in the order of the number of simple boundaries (see Figure 27).

Consider procedure EXTRACT-POLYHEDRA. Let n denote the input size, let k denote a characteristic of the complexity of the traversed shells. The initialisation of the graph requires $\Theta(n \log n)$ time. Extracting the shells takes $O(n (\log n + K))$ time. Procedure MAXIMAL takes $O(n \log n)$ time in total. Hence:

- (11) For a set F of non-intersecting plane segments that defines the boundary of a volume segment or volume shape S , constructing the simple boundaries that define the volume segments of S requires $O(nK + n \log n)$ time and $\Theta(n)$ space where $n = |F|$ and K is the number of simple boundaries.

The next step is to distinguish the outer and inner boundaries from a set of simple polyhedral boundaries and to construct the corresponding volume segments. We consider procedure CONSTRUCT (Figure 30), with input, a set F of plane segments that forms the boundary of a volume shape. We assume that neither two plane segments in F overlap nor one contains the other. The result of the procedure is a volume shape as a set of maximal volume segments.

Figure 30 CONSTRUCT: constructing the volume segment from a shape of boundary plane segments

```

CONSTRUCT (F)
1  SPLIT (F)
2  R ← EXTRACT-POLYHEDRA (F)
3  T ← NIL
4  for each shape F ∈ R                               in order
5      INSERT (T, F, ENCLOSURE)
6  return EXTRACT-SEGMENTS (T)

Splitting a set of plane segments at their lines of intersection
SPLIT (F)
1  G ← ∅
2  for each class CF ⊂ F
3      split[CF] ← ∅
4  for each class CF ⊂ F                               in order
5      for each class CG ⊂ F with co[CG] > co[CF]
6          L ← INTERSECTION (CF, CG)
7          split[CF] ← split[CF] + L
8          split[CG] ← split[CG] + L
9      SORT (split[CF])
10     REMOVE-MULTIPLES (split[CF])
11     L ← boundary[CF]
12     M ← DIFFERENCE (split[CF], L)
13     (L, M) ← SPLIT (L U M U M)
14     G ← G + CONSTRUCT (L U M U M)
15     SORT (G)
16     F ← G

Extracting the volume segments from a tree of polyhedral boundaries
EXTRACT-SEGMENTS (t)
1  S ← ∅
2  while t ≠ NIL
3      u ← child[t]
4      R ← {shape[t]}
5      while u ≠ NIL
6          S ← S + EXTRACT-SEGMENTS (child[u])
7          for each segment x ∈ shape[u]
8              inside[x] ← -inside[x]
9          R ← R + {shape[u]}
10         u ← sibling[u]
11         S ← S + SEGMENT (R)
12         t ← sibling[t]
13     SORT (S)
14     return S

```

Figure 30 CONSTRUCT: constructing the volume segment from a shape of boundary plane segments (continued)

```

ENCLOSURE (F, G)
    Number of piercing points is either odd or even
    1  odd ← FALSE
    2  f ← first[F]
    3  l ← first[boundary[f]]
    4  for each plane segment g ∈ G
    5      if co[f] ≠ co[g]
    6          Check for intersection with boundary[g]
    7          then k ← first[boundary[g]]
    8              p ← 0
    9              while k ≠ 0 and p = 0
    10                 p ← INTERSECTION-2 (k, co[l])
    11                 k ← next[k] if p = 0
    12             if k ≠ 0
    13                 then if p = tail[k] or p = head[k]
    14                     Intersection point is an endpoint
    15                     then if VALID-ENDPOINT (p, g, l, co[f])
    16                         then odd ← not odd
    17                         else if VALID-INTERSECTION (k, co[g], l, co[f])
    18                             then odd ← not odd
    19                     else p ← PIERCING-POINT (co[l], g)
    20                     odd ← not odd if p ≠ 0
    21 return odd

VALID-ENDPOINT (p, g, l, co-f)
    1  co-m ← INTERSECTION-LINE (co[g], co-f)
    2  j ← 0
    3  for each line segment k ∈ rest[boundary[g]]
    4      if tail[k] = p or head[k] = p
    5          then if j = 0
    6              then j ← k
    7              else if DOT-PRODUCT (co[k], co-m) < DOT-PRODUCT (co[j], co-m)
    8                  then j ← k
    9  if DOT-PRODUCT (co[j], co-m) = 0
    10 then return COMPARE-INSIDE-2 (co-f, j, co[g])
    11 else return COMPARE-INSIDE-1 (co-m, j, co[g])

VALID-INTERSECTION (k, co-g, l, co-f)
    1  co-m ← INTERSECTION (co-g, co-f)
    2  if co[k] = co-m
    3  then return COMPARE-INSIDE-2 (co-f, k, co-g)
    4  else return not (COMPARE-INSIDE-1 (co-m, k, co-g)
    5                      xor COMPARE-INSIDE-1 (co-m, l, co-f))

```

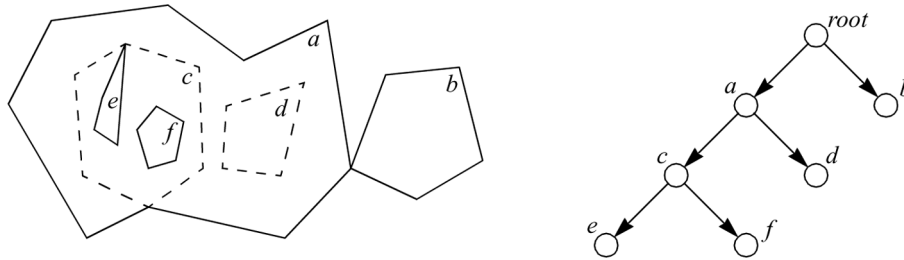
Once we have extracted the simple boundaries from the set F , we need to distinguish the inner and outer boundaries and, subsequently, create the boundary shapes as these define the maximal volume segments of the resulting shape. We say that a boundary x *encloses* a boundary y if the shape defined by x , $\Gamma(x)$, contains the shape defined by y , $\Gamma(y)$. Moreover, if $\Gamma(x) \times \Gamma(y) \neq 0$ for two extracted boundaries x and y , then either $\Gamma(x) \leq \Gamma(y)$ or $\Gamma(y) \leq \Gamma(x)$ (Krishnamurti and Stouffs, 2004).

Given the set of simple boundaries resulting from procedure EXTRACT-POLYHEDRA (Figure 29), consider the *enclosure-tree* of these boundaries, defined as follows. Let $vertex[x]$ denote the vertex in the tree representing boundary x . If a boundary x encloses a boundary y , then, $vertex[x]$ is an ancestor of $vertex[y]$ (and $vertex[y]$ is a descendant of $vertex[x]$). If $vertex[x]$ is the parent of $vertex[y]$, then, boundary x encloses boundary y and any boundary that encloses y (with the exception of y itself) also encloses

x . Consider an imaginary root that encloses all boundaries. Then, all children of the root represent outer boundaries that are not enclosed by any other boundaries. Consequently, the grandchildren of the root represent inner boundaries that are enclosed by one of the previous boundaries.

Let the level of a vertex denote the distance from the root and consider the root as an inner boundary. Then, all vertices on even levels represent inner boundaries while all vertices on odd levels represent outer boundaries, with respect to the resulting shape. Thus, the set of boundaries from a single vertex on an odd level and its children, defines a maximal segment of this resulting shape. Figure 31 illustrates the enclosure-tree for a plane shape.

Figure 31 The enclosure-tree of an exemplar shape defined by six simple boundaries. Inner boundaries are drawn dashed



We use the following algorithm to check enclosure for a single point within x , with respect to y .

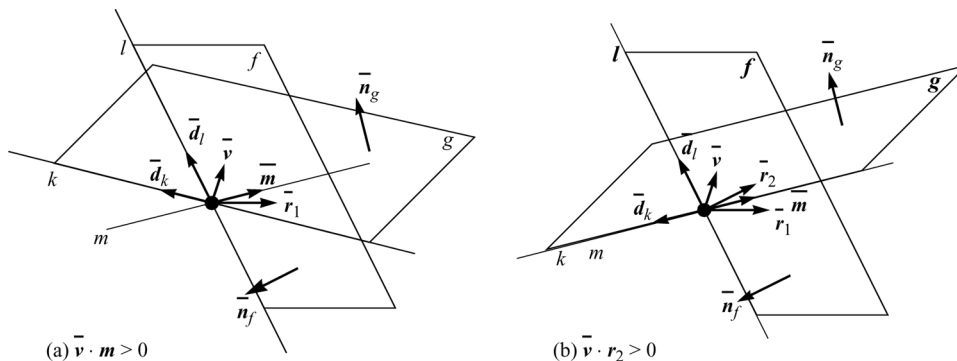
Procedure D

- 1 Consider the carrier of any boundary line segment l of a segment of x .
- 2 Determine the number of piercing points of this carrier with the segments of y that are to the left to or equal to the tail of the line segment l .
- 3 y encloses x only if this number is odd.

The piercing point of a line and a plane segment is the point of intersection of this line and the point set isomorphic to this plane segment. When determining the piercing point of a line with a plane segment, we distinguish whether the line pierces (the inside of) the plane segment, intersects (the inside of) the boundary of the plane segment, or contains an endpoint of a boundary segment of the plane segment; the latter two constitute ‘degenerate’ cases. In order to resolve these degenerate cases, we define two reference directions with respect to the carrier line. That is, we consider translating the carrier line over an arbitrarily small distance along directions perpendicular to this line. This is similar to the determination of the points of intersection of a line with the boundary of a plane shape or a plane with the boundary of a volume shape (see the algorithms in Figures 4 and 20).

Let l denote the carrier line, and r_1 and r_2 denote two unit reference direction vectors, with d_l , r_1 and r_2 mutually perpendicular. Consider the half-plane f with boundary l such that r_1 indicates the inside of f with respect to l . The normal vector of f equals the normalised vector product of d_l and r_1 , i.e., $n_f = \|d_l \times r_1\|$. We have $n_f = \pm r_2$. Also, $inside[f]$ equals +1 if $n_f \times d_l = r_1$ and -1, otherwise. This is illustrated in Figure 32.

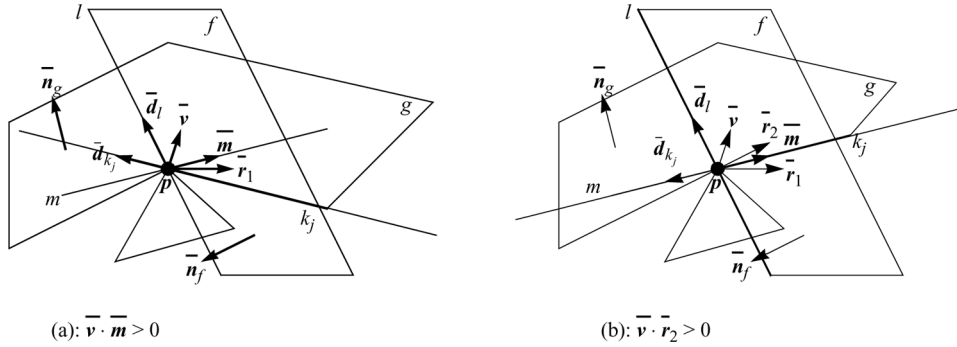
Figure 32 A point on the inside of a boundary segment k of a plane segment g is a valid piercing point: (a) if the direction vector of the line of intersection m of g and a reference half-plane f indicates the inside of g with respect to k , or (b) if m coincides with the carrier of k and the second reference vector indicates the half-space with boundary f that contains g



Consider the situation when l intersects the inside of a boundary line segment k of a plane segment g of y , also illustrated in Figure 32. Consider the line m of intersection of the carrier of g and f . Assume m is not co-linear with either l or the carrier of k . Then, the piercing point of l and g is a valid piercing point only if the (direction) vector \mathbf{m} of the line m indicates the inside of g with respect to k , that is, if $\text{inside}[k] (\mathbf{n}_g \times \mathbf{d}_k) \times \mathbf{m} > 0$. We have $\mathbf{m} = \pm(\mathbf{n}_f \times \mathbf{n}_g)$, where the sign is chosen such that $\mathbf{m} \times \mathbf{r}_1 > 0$, that is, \mathbf{m} also indicates the inside of f with respect to l . If $(\mathbf{n}_g \times \mathbf{d}_k) \times \mathbf{m} = 0$, then line m coincides with the carrier of k . In this case, we use the second reference direction vector \mathbf{r}_2 to determine the validity of the piercing point. That is, the piercing point of l and g is valid only if the vector resulting from the projection of \mathbf{r}_2 onto the carrier of g indicates the inside of g with respect to k , i.e., if $\text{inside}[k] (\mathbf{n}_g \times \mathbf{d}_k) \times \mathbf{r}_2 > 0$. If additionally $(\mathbf{n}_g \times \mathbf{d}_k) \times \mathbf{r}_2 = 0$, then, l lies on the carrier of g and there exists no piercing point.

Consider the case when l contains an endpoint p of a boundary line segment of a plane segment g of y , as depicted in Figure 33. Then, there exist at least two boundary line segments of g with endpoint p . Let k_i , $i \leq n$ denote all boundary line segments of g that have p as an endpoint. The scalar product of the direction vectors \mathbf{d}_{k_i} and \mathbf{m} is a measure of the cosine of the angle $\angle k_i m$ between the segments k_i and m (about p , in the carrier plane of g). If $\mathbf{d}_{k_i} \times \mathbf{m}$ is minimal for $i = j$ ($1 \leq i \leq n$), then, the line segment k_j makes the smallest angle with m , whether clockwise or counterclockwise. If the carrier of k_j and m do not coincide, i.e., $\mathbf{d}_{k_j} \times \mathbf{m} \neq 0$, then, \mathbf{m} indicates the inside of g with respect to k_j . That is, the point p is a valid piercing point if $\text{inside}[k_j] (\mathbf{n}_g \times \mathbf{d}_{k_j}) \times \mathbf{m} > 0$. Let COMPARE-INSIDE-1 ($\text{co}[m]$, k_j , $\text{co}[g]$) denote the result of the previous comparison (either TRUE or FALSE). If $(\mathbf{n}_g \times \mathbf{d}_{k_j}) \times \mathbf{m} > 0$, then, $k_j \times \mathbf{m} = 0$ and m coincides with the carrier of k_j . Thus, we use \mathbf{r}_2 to determine the validity of the piercing point, i.e., p is a valid piercing point if $\text{inside}[k_j] (\mathbf{n}_g \times \mathbf{d}_{k_j}) \times \mathbf{r}_2 > 0$. Let COMPARE-INSIDE-2 ($\text{co}-r$, k_j , $\text{co}[g]$) denote the result of the previous comparison (either TRUE or FALSE), where the co-descriptor $\text{co}-r$ represents a plane with normal vector \mathbf{r}_2 .

Figure 33 An endpoint of a boundary segment k_j of a plane segment g , where k_j makes the smallest angle with the line of intersection m of g and a reference half-plane f , is a valid piercing point: (a) if the direction vector of m indicates the inside of g with respect to k_j , or (b) if m coincides with the carrier of k_j and the second reference vector indicates the half-space with boundary f that contains g



We chose the following reference direction vectors: Given a plane segment f with boundary segment l , the first reference vector indicates the inside of f with respect to l and, therefore, is perpendicular to the direction vector of l . The second reference vector is the normal vector of f :

$$\begin{cases} \vec{r}_1 = \text{inside}[l](\vec{n}_f \times \vec{d}_l) \\ \vec{r}_2 = \vec{n}_f \end{cases}$$

Before we can extract the polyhedral boundaries from the set of (plane) segments F , we have to ensure that no two plane segments intersect. Procedure SPLIT, when applied to a set of plane segments, determines the line segments of intersection for each plane segment with respect to all other segments and, subsequently, creates the sub-segments as defined by these line segments of intersection (see proof of (7) for procedure SPLIT when applied to a set of line segments). In order to construct the simple boundaries of the sub-segments of a plane segment f , we use a single copy of the boundary segments of f together with two copies of the line segments of intersection of f as input to procedure CONSTRUCT (see Figure 30).

After the plane segments are split and the polyhedral boundaries subsequently extracted, the enclosure-tree is constructed for the resulting simple boundaries. The *sibling* and *child* fields link a vertex with its siblings and children, respectively. The tree is initially empty. Each boundary is inserted in the tree in the order of appearance in the set of boundaries. We know from the algorithm for EXTRACT-POLYHEDRA (Figure 29) that every boundary is discovered before any boundaries it encloses.

Procedure ENCLOSURE compares two simple boundaries F and G and returns TRUE if G encloses F and FALSE, otherwise. Procedure VALID-ENDPOINT checks the validity of a piercing point that coincides with an endpoint of a boundary segment. Similarly, procedure VALID-INTERSECTION checks the validity of a piercing point that coincides with an endpoint of a boundary segment. Procedure PIERCING-POINT determines the point of intersection of a line l and plane segment g , in all but the above two cases. We use the fact that the points of intersection of the boundary of a coequal shape and an infinite line within the carrier of the shape defines an alternating sequence (if sorted) of inner and outer segments to determine the validity of the piercing point,

given the number of intersection points of an arbitrary line through the proposed piercing point and within the carrier of g (e.g., the line of intersection of this carrier and a normal plane through the line l), with the boundary segments of g . Finally, the tree is traversed and the boundaries extracted and grouped as they define the maximal segments of the resulting shape (EXTRACT-SEGMENTS).

Let n denote the size of (the boundary of) F . Consider procedure SPLIT. Determining the line segments of intersection between the classes of plane segments of F and G takes $O(kn \log n)$ time where k is the total number of classes of F and G (steps 4–8). Constructing the plane segments from the set of boundary segments and line segments of intersection (two copies) of a class of segments takes $O((m+n) \log n) = O(kn \log n)$ time, where m is the number of line segments of intersection, i.e., $m = O(kn)$ (steps 9–14). This bound is also the overall complexity for the procedure SPLIT. The resulting size of F is $O(m+n)$.

Extracting the simple boundaries (EXTRACT-POLYHEDRA) from F takes $O((m+n)(\log n + K))$ time, where K denotes the number of resulting simple boundaries, and results in a set R of $O(m+n)$ size. Consider the procedure ENCLOSURE. Determining the point(s) of intersection of a line segment with the boundary of a plane segment takes linear time in the size of this boundary (steps 6–10). This time complexity holds also for the procedure PIERCING-POINT, as well as VALID-ENDPOINT and VALID-INTERSECTION (constant time, actually). Since, we repeat this computation for each segment of G , procedure ENCLOSURE takes $O(\alpha)$ time, where α denotes the sum of the sizes of both simple boundaries. Upon insertion of a boundary into the tree T (CONSTRUCT: step 5), each boundary already in the tree may need to be examined for possible enclosure, in the worst case. Therefore, building the tree takes $O(K(m+n))$ time for K simple boundaries. Extracting the segments (EXTRACT-SEGMENTS) from T takes $O(K \log K + m+n)$ time. The resulting time complexity of procedure CONSTRUCT is thus $O(kn \log n + Km)$. Hence:

- (12) *For a (sorted) set F of non-overlapping plane segments that defines the boundary of a volume shape S , constructing the volume segments that make up S takes $O(Km + kn \log n)$ time and $\Theta(m+n)$ space, where $n = |F|$, $k = |\text{classes}[F]|$, m is the number of (non-boundary) line segments of intersection between segments of F and K is the number of simple boundaries of S .*

This algorithm may be improved by considering the simple boundaries that result from EXTRACT-POLYHEDRA, grouped as they are returned from separate calls to the procedure SHELLS (see Figure 29). This procedure returns a set of one or more simple boundaries of which all but the first are, relatively, inner boundaries with respect to the first one. As such, these ‘inner’ boundaries no longer need to be classified. However, in the worst case, each call to the procedure SHELL returns a single boundary and the time complexity of the procedure CONSTRUCT remains the same.

We can use the previous result when determining the maximal shape corresponding to a set of volume segments that may share boundary, neither overlap nor one contains another. The time complexity of procedure MAXIMAL (Figure 34) is necessarily the same as for CONSTRUCT, i.e., $O(Km + kn \log n) = O(kn(\log n + K))$, where $n = |\text{boundary}[S]|$, $k = |\text{classes}[\text{boundary}[S]]|$, m is the number of line segments of intersection between segments of $\text{boundary}[S]$ and K is the number of simple boundaries of S .

Figure 34 MAXIMAL: maximal shape corresponding to a set of volume segments that may share boundary

```

MAXIMAL (S)
1  SPLIT (boundary[S])
2  REDUCE (boundary[S])
3  R ← EXTRACT-POLYHEDRA (boundary[S])
4  T ← NIL
5  for each shape F ∈ R           in order
6     INSERT (T, F, ENCLOSURE)
7  S ← EXTRACT-SEGMENTS (T)

```

5 Concluding remarks

We have presented algorithms for the classification and subsequent construction of the boundary of a shape. These algorithms form the core of a three-dimensional shape grammar system under implementation. We have also considered, though not reported here, the computational complexity for shape operations and shape relations, when applied to a pair of coequal segments, and found these are bounded by some polynomial function f of the size of the boundaries of the segments. The particular polynomial f depends on the shape algebra, U_i , $i \geq 0$, under consideration (see Stiny, 1991, for a description of the different shape algebras). The running times of arithmetic operations and relations on shapes are then asymptotically bound by some function of f . The examination of this computational complexity of arithmetic in shape algebras U_i , $0 \leq i \leq 3$, along with a comparison to results found in literature for similar algorithms is the topic of a forthcoming paper. Algorithms for other geometrical realisations of shape algebras for shape grammar applications remain open, in particular, the treatment of curved shapes although some progress has been reported by Jowers et al. (2004).

References

- Bentley, J.L. and Ottman, T.A. (1979) 'Algorithms for reporting and counting geometric intersections', *IEEE Transactions on Computers*, Vol. 28, pp.643–647.
- Chase, S.C. (1989) 'Shapes and shape grammars: from mathematical model to computer implementation', *Environment and Planning B: Planning and Design*, Vol. 16, No. 2, pp.215–242.
- Cormen, T.H., Leiserson, C.E. and Rivest, R.L. (1990) *Introduction to Algorithms*, The MIT Press, Cambridge, Mass.
- de Berg, M., van Kreveld, M. and Overmars, M. (1997) *Computational Geometry: Algorithms and Applications*, Springer, Berlin.
- Gursoz, E.L., Choi, Y. and Prinz, F.B. (1991) 'Boolean set operations on non-manifold boundary representation objects', *CAD Computer-Aided Design*, Vol. 23, No. 1, pp.33–39.
- Hoffman, C.M. (1989) *Geometric and Solid Modeling: An Introduction*, Morgan Kaufmann, San Mateo, California.
- Jowers, I., Prats, M., Earl, C. and Garner, S. (2004) 'On curves and computation with shapes', in Akin, O., Krishnamurti, R. and Lam, K.P. (Eds.): *Generative CAD Systems*, Carnegie Mellon University, Pittsburgh, Pa, pp.439–457.

- Karasick, M. (1988) *On the Representation and Manipulation of Rigid Solids*, PhD Dissertation, Department of Computer Science, McGill University, Montreal. (Technical report TR89-976, Dept. of Computer Science, Cornell University, Ithaca, N.Y., 1989)
- Krishnamurti, R. (1980) 'The arithmetic of shapes', *Environment and Planning B: Planning and Design*, Vol. 7, pp.463–484.
- Krishnamurti, R. (1992) 'The arithmetic of a maximal planes', *Environment and Planning B: Planning and Design*, Vol. 19, pp.431–464.
- Krishnamurti, R. and Stouffs, R. (1993) 'Spatial grammars: motivation, comparison and new results', in Flemming, U. and van Wyk, S. (Eds.): *CAAD Futures '93*, North-Holland, Amsterdam, pp.57–74.
- Krishnamurti, R. and Stouffs, R. (2004) 'Classifying the boundary of a shape', *The Journal of Design Research*, Vol. 4, No. 1 (<http://jdr.tudelft.nl/>).
- Mäntylä, M. (1988) *An Introduction to Solid Modeling*, Computer Science Press, Rockville, Md.
- Nievergelt, J. and Preparata, F.P. (1982) 'Plane-sweep algorithms for intersecting geometric figures', *Communications of the ACM*, Vol. 25, pp.739–747.
- Sleator, D.D. and Tarjan, R.E. (1985) 'Self-adjusting binary search trees', *Journal of the Association for Computing Machinery*, Vol. 32, pp.652–686.
- Stiny, G. (1991) 'The algebras of design', *Research in Engineering Design*, Vol. 2, pp.171–181.
- Stouffs, R. (1994) *The Algebra of Shapes*, PhD Thesis, Department of Architecture, Carnegie Mellon University, Pittsburgh, Pa.