Set Operations on Sculptured Solids

Spencer W. Thomas

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF UTAH

UUCS-87-004

Set Operations on Sculptured Solids

Spencer W. Thomas*
Computer Science Department
University of Utah
Salt Lake City, Utah 84112
USA

March 4, 1987

Abstract

A general method is presented for performing Boolean set operations on solids represented by a sculptured surface boundary model. The method is developed first for solids with closed boundary surfaces. It is then shown to apply also to "partially bounded" solids, which have incompletely specified bounding surfaces. The method is demonstrated using solids with B-spline bounding surfaces.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling – *solid object representations, geometric algorithms.* J.6: Computer-Aided Engineering – *Computer-aided Design.*

General Terms: Algorithms, Theory.

Additional Key Words and Phrases: Solid modeling, set operations, sculptured surfaces.

*This work was supported in part by the National Science Foundation (DCR-8506115, DCR-8203692 and DCR-8121750), the Defense Advanced Research Projects Agency (DAAK11-84-K-0017), and the Office of Naval Research (N00014-82-K-0351). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

1 Introduction

Solid Modeling deals with the problem of creating a computer model of a solid object. In principle, any operation performed upon the model, or any query directed to the model should yield the same result that would be received from the real object. Most research in the solid modeling area has been directed towards solids composed of primitive shapes and polyhedra[1,2,3,5,7,9,13]. The process of building up complex shapes as combinations of primitive solids is commonly referred to as *constructive solid geometry* (*CSG*).

Boolean set operations are generally a central component of a solid modeling system, as they provide the primary means of building up complex shapes from simpler ones. This is a fundamental operation in systems where the only shapes supported directly are simple quadric solids (spheres, cylinders, etc.) and polyhedral forms; set operations are also important to a system using sculptured surfaces. The three basic set operations, *union*, *intersection*, and *difference* are illustrated in Figure 1.

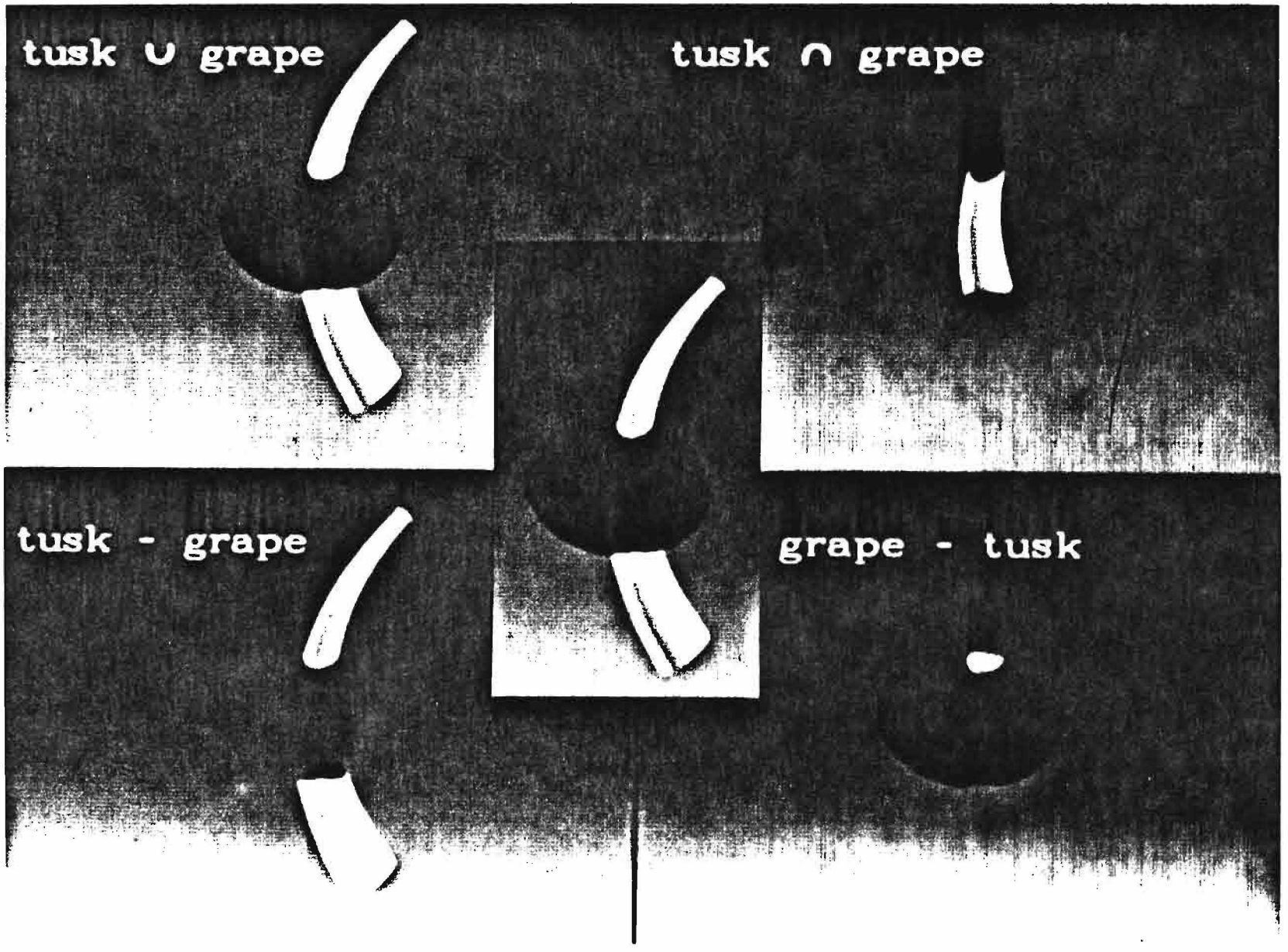
One of the reasons that most solid modeling systems use simple shapes is the difficulty of performing Boolean set operations on models using sculptured surface boundaries. The difficulty arises from two factors. First, the intersection of two sculptured surfaces may not be derivable in a useful form, and must therefore be approximated. Second, design with free-form surfaces is complicated by a requirement, inherent in all previous algorithms for performing set operations, that the boundaries of the operand bodies be closed. Independent solutions to the first problem, using a subdivision approximation, are described by Carlson, Peng, and Thomas[4,8,11]. A solution using ray-tracing is described by Kitaoka[6]. We present here a solution to the second problem.

An example of the use of set operations in a sculptured solid model is illustrated in Figure 2. The object modeled is the root of a turbine blade. In the figure, the top of the root (to which the airfoil would be attached) is at the bottom of the picture. The basic shape of the root is defined by a single B-spline surface. The pockets are removed from the sides of the root by subtracting a pair of pocket shaped surfaces. The flare at the bottom of the cooling passage is also formed independently, and then subtracted from the root assembly. The complex curve at the bottom of the root arises naturally from this operation. The platform at the top of the root is part of a tapered cylindrical surface with a large radius. If the entire cylinder were modeled, as would be necessary in a conventional CSG based modeling system, it would dwarf the rest of the root.

Note that in this design none of the original surfaces are closed, although the final result is bounded by a closed surface, as it must be to represent a real turbine blade. It is distracting and irrelevant to force a designer to arbitrarily complete a boundary for the sole purpose of satisfying the aritficial systems' requirements for effecting set operations. It should only be necessary to design those pertinent regions of the surface which affect the final shape of the object.

Figure 1: The basic set operations

Figure 2: Design of root of turbine blade


tusk ∪ grape
tusk ∩ grape
tusk - grape
grape - tusk

The converse of this principle is that, as much as is possible, the modeling system should not produce unexpected results when presented with such data.

This paper provides a method for performing set operations on solids bounded by free-form, sculptured surfaces. In many cases, the description of intermediate results need not be complete; it is not necessary to provide a closed bounding surface. Criteria for the validity of a set operation expression involving *partially bounded*[1] objects are presented. Although B-spline surfaces were used in an implementation of these ideas, they are applicable to any boundary model, and apply to higher dimensional spaces as well.

2 Background

Using set operations is a common approach to combining objects into more complex objects. They are fairly simple to understand, and provide a natural way of describing objects that will be manufactured by machining. The difference operator, for example, directly models material removal by such manufacturing operations as drilling and milling. Some geometric features of an object can arise from interactions between independently designed pieces. For example, the curve where the wing of an airplane joins the fuselage is not designed, but is determined by where the shapes of the wing and fuselage intersect.

2.1 Objects as Regular Sets

To perform set operations on objects, they must be represented as sets of points in space. We desire to model manufacturable, or *realizable*, objects. This is tied in with the concept of *model validity*. Intuitively, a valid model represents a realizable object, but we require a more precise definition. In general, validity corresponds to a formal statement about properties of a set, which can be verified, and which can be maintained throughout the modeling process. A good characterization of a valid set is that it has no isolated points or "dangling edges," and neither does its complement. For example, the infinitely thin square represented by the set of points

$$\{(x, y, z) \mid 0 \leq x \leq 1, 0 \leq y \leq 1, z = 0\}$$

is not a realizable object, and should not be included in the class of valid sets. A class of sets which satisfies this characterization is the class of *regular sets*. A set can be *regularized* by taking the closure of its interior. Figure 3 illustrates this process, starting in *a* with a set with a number of dangling edges, isolated points, and a missing point, or hole. *b* shows the interior of the set, the dangling edges and isolated points are gone, but the hole remains. *c* is the closure of the interior, the hole has been filled in and the boundary has been added. A regular

[1] See Section 5.

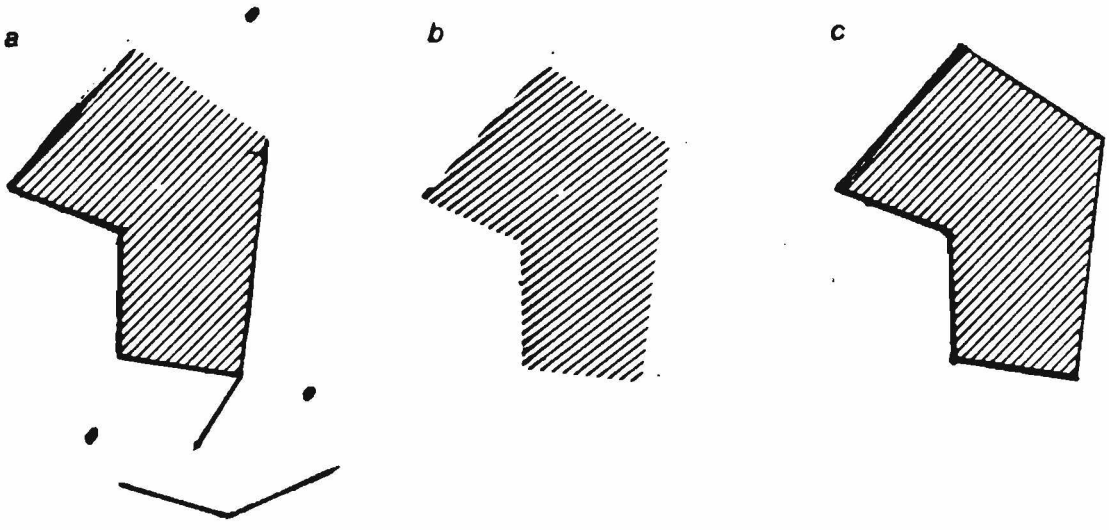


Figure 3: Set regularization

set is defined as a set which is equal to its regularization[10]. Regularity is a formal property of sets that can serve as the basis for definition of a valid model.

The class of regular sets is not closed under set operations. This would seem to make it useless for modeling. However, it is possible to redefine the standard set operations to achieve operations which, when applied to regular sets, always produce regular sets. These are called *regularized set operations*. The result of a regularized set operation on two (regular) sets is just the regularization of the result of the corresponding standard set operation on the original two sets. Regularized set operations obey the laws of Boolean algebra, and are the proper analogue of the standard set operations for regular sets. Unless stated otherwise, all sets and set operations will be assumed regular.

2.2 Set Membership Classification

We consider next the problem of representing sets in a computer. Any regular set contains an uncountable number of points, so direct enumeration is impossible. An indirect representation of a set is provided by a *set membership classification function*. Given a point p and a set S, the membership classification function $M(p, S)$ returns **in** if the point is a member of the set, and **out** if it is not. A membership classification function is a complete representation of a set; in theory, all questions about the set may be answered by reference to it. Of course, it may be necessary to ask infinitely many questions, but theoretically it is possible.

Tilove[12] developed a method for determining the result of a set operation on sets described by set membership classification. The set membership classification function for the result is determined directly from the membership functions for the operands. The classification of a given point with respect to the result thus depends only on the classifications of the point with respect to the operands.

In order to correctly model set operations on regular sets using this technique, it is necessary to extend the set membership classification function to provide extra information about boundary points. In essence, it is necessary to know on "which side" of the boundary point the set lies. Tilove associates a *neighborhood* with each boundary point. The neighborhood is defined as the intersection of an arbitrarily small open ball about the point with the set. If the neighborhood of a point p with respect to a set S is denoted by $N(p, S)$, then the set membership classification function for the intersection of two sets A and B can be written

$$
\begin{aligned}
M(p, A \cap B) = \quad & \mathbf{in}, && \text{if} \quad M(p, A) = \mathbf{in} \text{ and } M(p, B) = \mathbf{in}, \\
& \mathbf{bdy}, && \text{if} \quad M(p, A) = \mathbf{bdy} \text{ and } M(p, B) = \mathbf{in}, \\
& && \text{or} \quad M(p, A) = \mathbf{in} \text{ and } M(p, B) = \mathbf{bdy}, \\
& && \text{or} \quad M(p, A) = \mathbf{bdy} \text{ and } M(p, B) = \mathbf{bdy} \text{ and} \\
& && \quad N(p, A) \cap N(p, B) \neq \emptyset, \\
& \mathbf{out}, && \text{otherwise.}
\end{aligned}
$$

$$
N(p, A \cap B) = N(p, A) \cap N(p, B).
$$

Note that neighborhoods have a lower dimensionality than the original sets, so that the above definition of intersection is not infinitely recursive[2].

Figure 4 shows an example of this process. Note that the point p_1 is on the boundary of $A \cap B$, since its neighborhoods in A and B are not disjoint, while the point p_2 has neighborhoods in C and D that are disjoint (in a regular sense), and is therefore outside $C \cap D$.

3 Set Operations on Boundary Models

The set operation method using a membership classification function described above does not work well on boundary models, for which it is traditionally difficult to formulate a membership classification function. The result of a set operation on boundary models is the boundary of the result, so a method of performing set operations on boundary models can be derived by considering the **bdy** case in the membership classification functions. Such a method is described with a dimensionally recursive formulation by Putnam and Subrahmanyam[9]. They describe a *boundary classification* function, that classifies the boundaries of one set with respect to another. The function $C(A, B)$ classifies the boundary

[2]E.g., the neighborhood of a point on the boundary of a planar set can be represented as a set of angular ranges, and can therefore be considered a one-dimensional set.

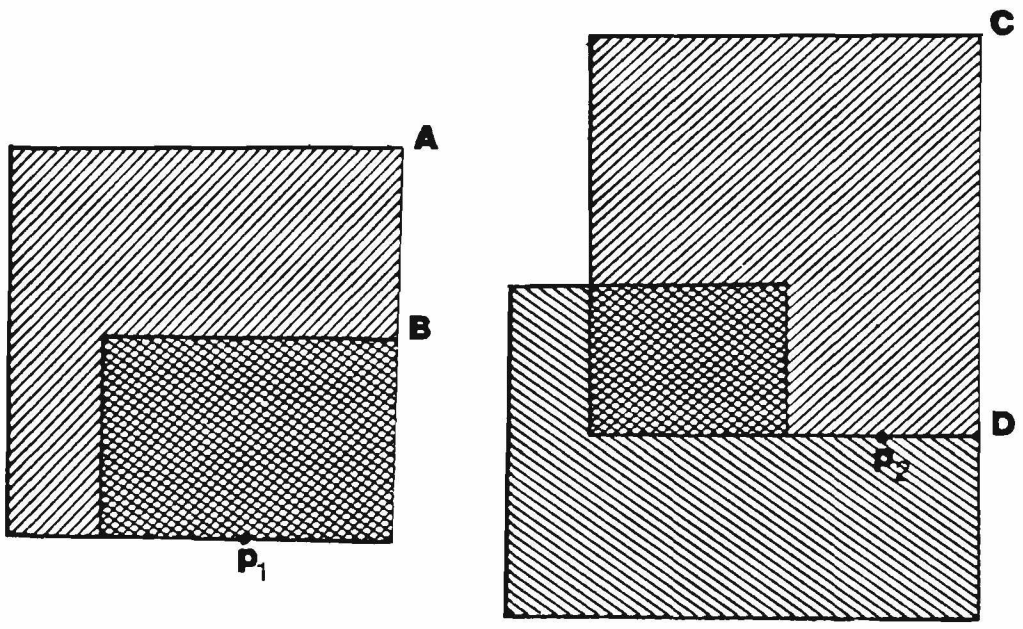


Figure 4: Regular set operations

of A with respect to the set B, and returns a 4-tuple of subsets of the boundary of A.

$$C(A,B) = \{AinB, AoutB, AsharedB, AantisharedB\}$$

The *shared* and *anti-shared* cases arise from coincident sections of the boundaries, and correspond to the cases where the neighborhoods are, respectively, overlapping and disjoint. An example of two sets and the classifications of their boundaries with respect to each other is shown at the top of Figure 5.

Note that the boundaries must be *oriented*, with the orientation indicating where the "inside" of the boundary is, and thus which region comprises the set itself. The orientation of the results of boundary classification is taken to be identical to the orientation of the boundary of the set mentioned first, i.e., $AantisharedB$ is oriented as a piece of the boundary of A, while $BantisharedA$ is oriented as a piece of the boundary of B. Both have the same surface geometry, but they have opposite orientations.

From the boundary classification, we find that, e.g., the boundary of the intersection of two sets is given by

$$\partial(A \cap B) = AinB \cup BinA \cup AsharedB$$

The three basic set operations are illustrated at the bottom of Figure 5.

3.1 Boundary Expressions

We wish to consider not just simple set operations, but complete expressions involving set operations (*set expressions*). First, we will derive an extension of the boundary classification method that has several advantages:

- It uses standard Boolean notation, allowing us to easily manipulate expressions.
- It is extensible to deriving boundary classifications directly from set expressions.
- It will allow us to handle *partial boundaries*.

We write rules for the boundary of the result of a set operation in terms of set operations between boundaries and sets, instead of using the classification function. We will not consider the cases of coincident boundaries, but only the "in" and "out" cases[3]. There is a one-to-one mapping between simple expressions of this form and the boundary classification function:

$$\begin{aligned} AinB &= \partial A \cap B \\ AoutB &= \partial A \cap \overline{B} \end{aligned}$$

[3]We see in Section 6 how to deal with coincident surfaces in this context.

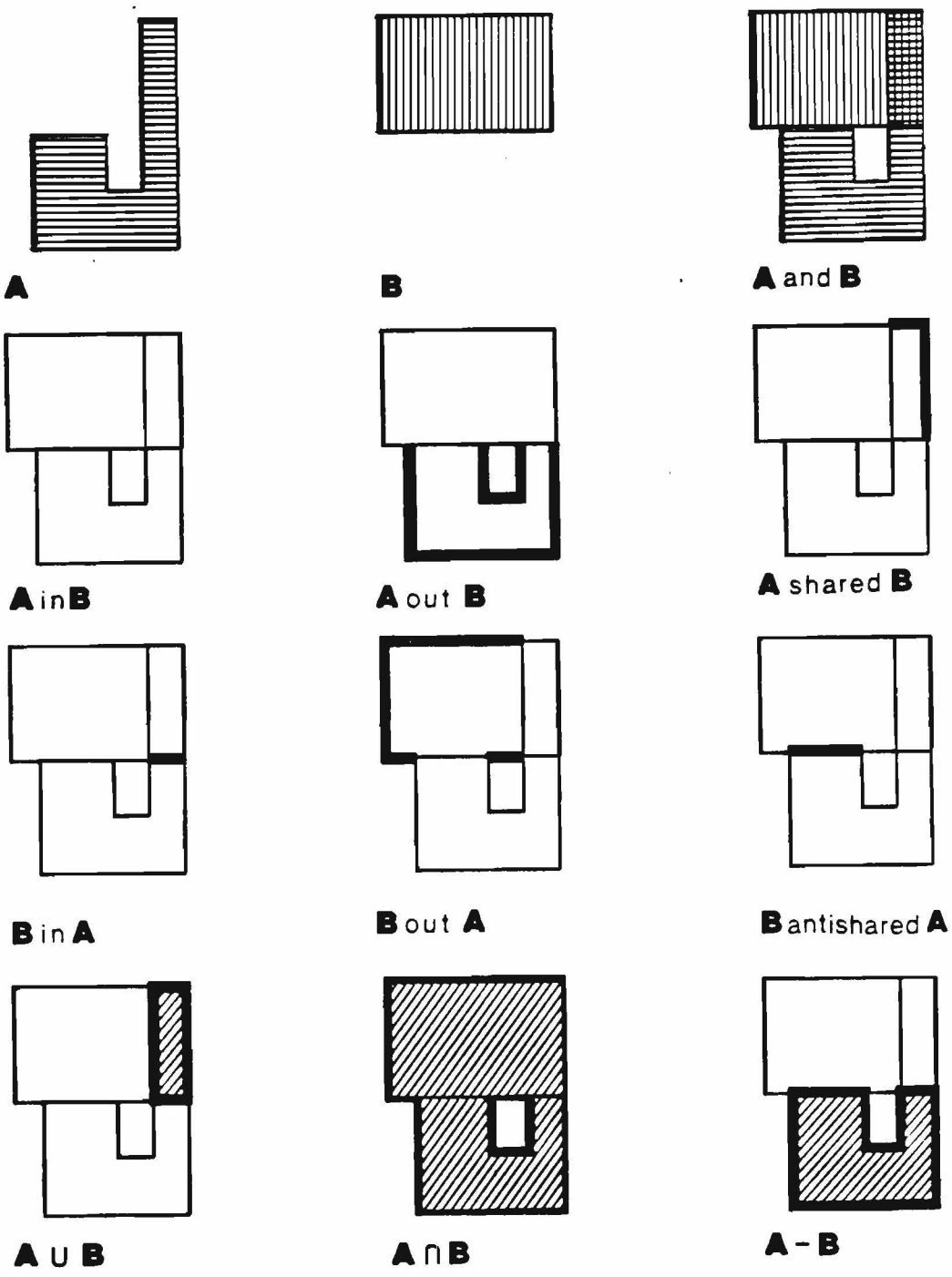


Figure 5: Boundary classification and set operations

Thus, we have

$$\partial(A \cap B) = [\partial A \cap B] \cup [\partial B \cap A] \quad (1)$$
$$\partial(A \cup B) = [\partial A \cap \overline{B}] \cup [\partial B \cap \overline{A}] \quad (2)$$

Let us first compute a simple example using this method. Consider

$$(A \cap B) \cap C$$

We get the boundary of the result by the following steps

$$\begin{aligned}
&\partial[(A \cap B) \cap C] \\
&\quad = [\partial(A \cap B) \cap C] \cup [(A \cap B) \cap \partial C] \\
&\quad = \{[(\partial A \cap B) \cup (A \cap \partial B)] \cap C\} \cup [(A \cap B) \cap \partial C] \\
&\quad = [\partial A \cap B \cap C] \cup [\partial B \cap A \cap C] \cup [\partial C \cap A \cap B)]
\end{aligned}$$

We have reduced the single expression into three expressions of lower dimensionality. The expression

$$\partial A \cap B \cap C$$

means that we should compute the intersection of the sets B and C as restricted to the boundary of A. The result of this intersection is the piece of the boundary of A that will appear in the boundary of $(A \cap B) \cap C$. We call this expression the *boundary expression* for A.

The restricted set operations may also be performed by the same method, so that if we denote

$$\begin{aligned}
B_a &= \partial A \cap B \\
C_a &= \partial A \cap C,
\end{aligned}$$

then we can write

$$\begin{aligned}
\partial A \cap B \cap C &= B_a \cap C_a \\
\partial(B_a \cap C_a) &= [\partial B_a \cap C_a] \cup [\partial C_a \cap B_a]. \quad (3)
\end{aligned}$$

Note that the boundary of, e.g., B_a is exactly the intersection of the boundary of B with the boundary of A. We can therefore easily evaluate the expressions in Equation 3.

A simple algorithm for finding the boundary of a complex Boolean expression is

Algorithm 1

1. Reduce the expression to a sum of products form (an expression which is a union of terms, each of which is the intersection of objects or their negation). Reduce each term as much as possible, and eliminate redundant terms.

2. Repeatedly apply the simple boundary rules in Equations 1 and 2 to the sum of products form to get the boundary of the result.
3. Collect terms in each boundary separately and simplify. Note that a boundary and its negation should be treated separately in this step.

When we try to apply this algorithm we encounter some difficulties. Consider applying it to the fairly simple expression below:

$$\begin{aligned}
&\partial[(A \cap B) \cup (A \cap C)] \\
&= [\partial(A \cap B) \cap \overline{(A \cap C)}] \cup [\partial(A \cap C) \cap \overline{(A \cap B)}] \\
&= [\partial A \cap B \cap (\overline{A} \cup \overline{C})] \cup [\partial B \cap A \cap (\overline{A} \cup \overline{C})] \\
&\quad \cup [\partial A \cap C \cap (\overline{A} \cup \overline{B})] \cup [\partial C \cap A \cap (\overline{A} \cup \overline{B})] \\
&= \{\partial A \cap [(B \cap \overline{A}) \cup (B \cap \overline{C}) \cup (\overline{B} \cap C) \cup (\overline{A} \cap C)]\} \\
&\quad \cup [\partial B \cap (A \cap \overline{C})] \cup [\partial C \cap (A \cap \overline{B})]
\end{aligned}$$

The term in the result for ∂A contains some occurrences of $\overline{A}$. It is not clear how to evaluate an expression like $\partial A \cap \overline{A}$: Is the boundary of a set inside or outside the complement of the set? Let us investigate this question.

Using the above procedure, all terms involving the boundary of a set and the set itself (e.g. $\partial A \cap \overline{A}$) will arise from set union operations[3], e.g.,

$$\begin{aligned}
\partial(A \cup A) &= \partial A \cap \overline{A} \\
&= \partial A \\
\partial(A \cup \overline{A}) &= \partial A \cap A \\
&= \partial \emptyset \\
&= \emptyset
\end{aligned}$$

Therefore, we can define

$$\begin{aligned}
\partial A \cap \overline{A} &= \partial A, \text{and} \\
\partial A \cap A &= \emptyset.
\end{aligned}$$

Algorithm 1 can lead to fairly complex intermediate expressions, so we will modify it to obtain an algorithm that will tell us only how a particular boundary contributes to the boundary of the result. The following algorithm will produce the boundary expression for the set A from a set expression involving A. We treat A and $\overline{A}$ as independent sets here; they will have different boundary expressions.

Algorithm 2

1. As above, reduce to a sum of products form.

[3]This is true only because of the normalization of form in step 1.

2. Collect terms involving A using the inverse distributive law. Note that no terms involving $\overline{A}$ should be included in this collection.
3. Replace $\overline{A}$ with the universe in any terms of which it is a component. Therefore, if $\overline{A}$ is the only component of a term, the boundary of A will not appear in the result at all.
4. The expression is now in the form

$$(A \cap (Y_1 \cup Y_2 \cup \cdots)) \cup Z_1 \cup Z_2 \cup \cdots \quad ,$$

where none of the terms Y_i, Z_i involve the set A (or $\overline{A}$). Let

$$\begin{aligned} Y &= (Y_1 \cup Y_2 \cup \cdots), \\ Z &= (Z_1 \cup Z_2 \cup \cdots). \end{aligned}$$

Then

$$\begin{aligned} \partial[(A \cap Y) \cup Z] &= [\partial(A \cap Y) \cap \overline{Z}] \cup [\partial Z \cap \overline{(A \cap Y)}] \\ &= [\partial A \cap Y \cap \overline{Z}] \cup [\partial Y \cap A \cap \overline{Z}] \cup [\partial Z \cap \overline{(A \cap Y)}] \end{aligned}$$

Since Y and Z are independent of A, no part of the boundary of A can appear in the boundaries of Y or Z. Therefore, the portion of the boundary of A that appears in the result must be

$$\partial A \cap (Y \cap \overline{Z})$$

We can prove the correctness of Algorithm 2 by showing that $Y \cap \overline{Z}$ defines exactly the region in which the boundary of A constitutes the boundary of the result of the original expression. In other words, within $Y \cap \overline{Z}$, a point is inside or outside the result exactly as it is inside or outside of A. Outside $Y \cap \overline{Z}$, the result is independent of A, so that a point is inside or outside the result exactly as it is inside or outside $Y \cup Z$. The proof appears in Figure 6.

Let us also show formally that the elimination, in step 3, of $\overline{A}$ from the expression is valid. This involves showing that the membership function will give the same result as above, even when $\overline{A}$ is present. First, we note that any set expression can be rewritten in the form

$$(A \cap Y) \cup (\overline{A} \cap Z_1) \cup Z_2.$$

The boundary expression for A computed by Algorithm 2 is $A \cap (Y \cap \overline{[Z_1 \cup Z_2]})$. The proof is in Figure 7. Note that while the result of the set expression may be said to be "independent" of A outside $Y \cap \overline{Z}$, it may be dependent on $\overline{A}$ (as determined by the boundary expression for $\overline{A}$).

(1) to show:	for all p in $(Y \cap \overline{Z})$,
	$M(p, A) = M(p, [A \cap Y] \cup Z)$
$M(p, [A \cap Y] \cup Z) =$	**in**, if $M(p, A \cap Y) =$ **in** or $M(p, Z) =$ **in**
	out, otherwise
for all p in $(Y \cap \overline{Z})$,	$M(p, Z) =$ **out**,
	so $M(p, [A \cap Y] \cup Z) = M(p, A \cap Y)$.
$M(p, A \cap Y) =$	**in**, if $M(p, A) =$ **in** and $M(p, Y) =$ **in**
	out, otherwise
for all p in $(Y \cap \overline{Z})$,	$M(p, Y) =$ **in**,
	so $M(p, A \cap Y) = M(p, A)$.
qed	

(2) to show:	for all p in $\overline{(Y \cap \overline{Z})}$,
	$M(p, [A \cap Y] \cup Z) = M(p, Y \cup Z)$
note:	$\overline{(Y \cap \overline{Z})} = (\overline{Y} \cup Z)$
$M(p, [A \cap Y] \cup Z) =$	**in**, if $M(p, A \cap Y) =$ **in** or $M(p, Z) =$ **in**
	out, otherwise
$M(p, Y \cup Z) =$	**in**, if $M(p, Y) =$ **in** or $M(p, Z) =$ **in**
	out, otherwise
for all p in $\overline{Y}$,	$M(p, Y) =$ **out**,
	so $M(p, Y \cup Z) = M(p, Z)$, and
	$M(p, A \cap Y) =$ **out**,
	so $M(p, [A \cap Y] \cup Z) = M(p, Z)$
for all p in Z,	$M(p, Z) =$ **in**,
	so $M(p, Y \cup Z) =$ **in** $= M(p, Z)$, and
	$M(p, [A \cap Y] \cup Z) =$ **in** $= M(p, Z)$,
therefore	$M(p, Y \cup Z) = M(p, Z) =$
	$M(p, (A \cap Y) \cup Z)$, for p in $\overline{(Y \cap \overline{Z})}$.
qed	

Figure 6: Proof of correctness for algorithm 2

(3) to show: for all p in $(Y \cap \overline{(Z_1 \cup Z_2)})$,
$M(p, A) = M(p, [A \cap Y] \cup [\overline{A} \cap Z_1] \cup Z_2)$

$M(p, [A \cap Y] \cup [\overline{A} \cap Z_1] \cup Z_2) =$
in, if $M(p, A \cap Y) =$ **in**
or $M(p, \overline{A} \cap Z_1) =$ **in**
or $M(p, Z_2) =$ **in**
out, otherwise

$M(p, \overline{A} \cap Z_1) =$ **in**, if $M(p, \overline{A}) =$ **in** and $M(p, Z) =$ **in**
out, otherwise

for all p in $(Y \cap \overline{(Z_1 \cup Z_2)})$,
$M(p, Z_1) =$ **out**, and $M(p, Z_2) =$ **out**
so $M(p, \overline{A} \cap Z_1) =$ **out**,
so $M(p, [A \cap Y] \cup [\overline{A} \cap Z_1] \cup Z_2) =$
$M(p, A \cap Y)$

Conclusion follows as in (1)

Figure 7: Proof of correctness for algorithm 2, part 3

4 Set Expression Evaluation

In this section we will describe evaluation of set expressions using boundary expressions and the intersections between the boundaries. The intersections between boundaries will be called *cut points* or *cut curves* because they cut the boundaries into **in** and **out** regions. The cut points or curves inherit their orientation from their parent boundaries, and are themselves boundaries of the restricted sets.

Consider the set expression $A \cup B$ shown in Figure 8. To find the boundary of the result, first compute the boundary expression for each surface:

$$(\partial A \cap \overline{B}) \cup (\partial B \cap \overline{A})$$

Then intersect the boundaries ∂A and ∂B to get cut points in each, as shown in Figure 8*a*. The arrows show the orientation of the cut points in each boundary.

The cut points bound sets in the original boundaries. Figure 8*b* shows the set bounded by the cut points in ∂A. If this set is called B_a, then the right hand portion of the boundary expression for ∂A becomes a set expression restricted to ∂A. It evaluates to the part of ∂A not in the set B_a, as shown in *c*. The boundary expression for ∂B is similarly evaluated.

A simple recursive algorithm for computing the boundary of the result of a set expression involving n dimensional sets is given below.

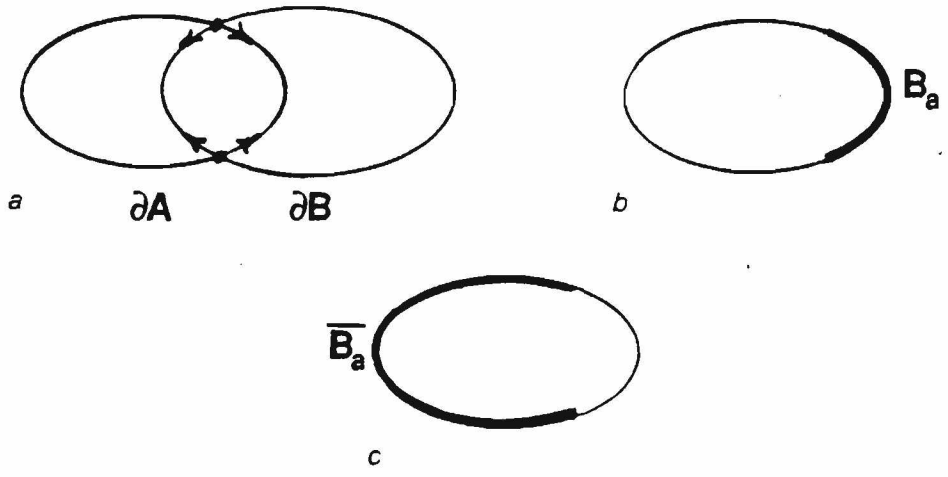


Figure 8: Evaluation of $A \cup B$ using cut points

Algorithm 3

1. For each set, X, in the set expression do:
   (a) Generate its boundary expression.
   (b) Intersect the $n-1$ dimensional boundary ∂X with that of each other object to get $n-2$ dimensional cut "curves."
   (c) Treating the boundary expression as a set expression restricted to the boundary ∂X, with the $n-1$ dimensional sets bounded by the cut curves, evaluate it by this algorithm to get the curves bounding the piece of ∂X which bounds the result of the original set expression.
2. Assemble the resulting boundary pieces into the boundary of the result of the set expression.

As a further example, consider the sets shown in Figure 9*a*. We wish to evaluate the expression

$$[(A \cup B) \cap C] \cup (A \cap D)$$

on these sets, giving the result shown at the right. Figure 9*b* shows the cut points for the boundary ∂A. Each point is labeled according to the boundary with which it is the intersection, with an arrow showing its orientation. The cut points bound sets B_a, C_a, and D_a in ∂A, as shown in *c*. The boundary expression for ∂A is

$$\partial A \cap [\overline{(B \cap C)} \cap (C \cup D)]$$

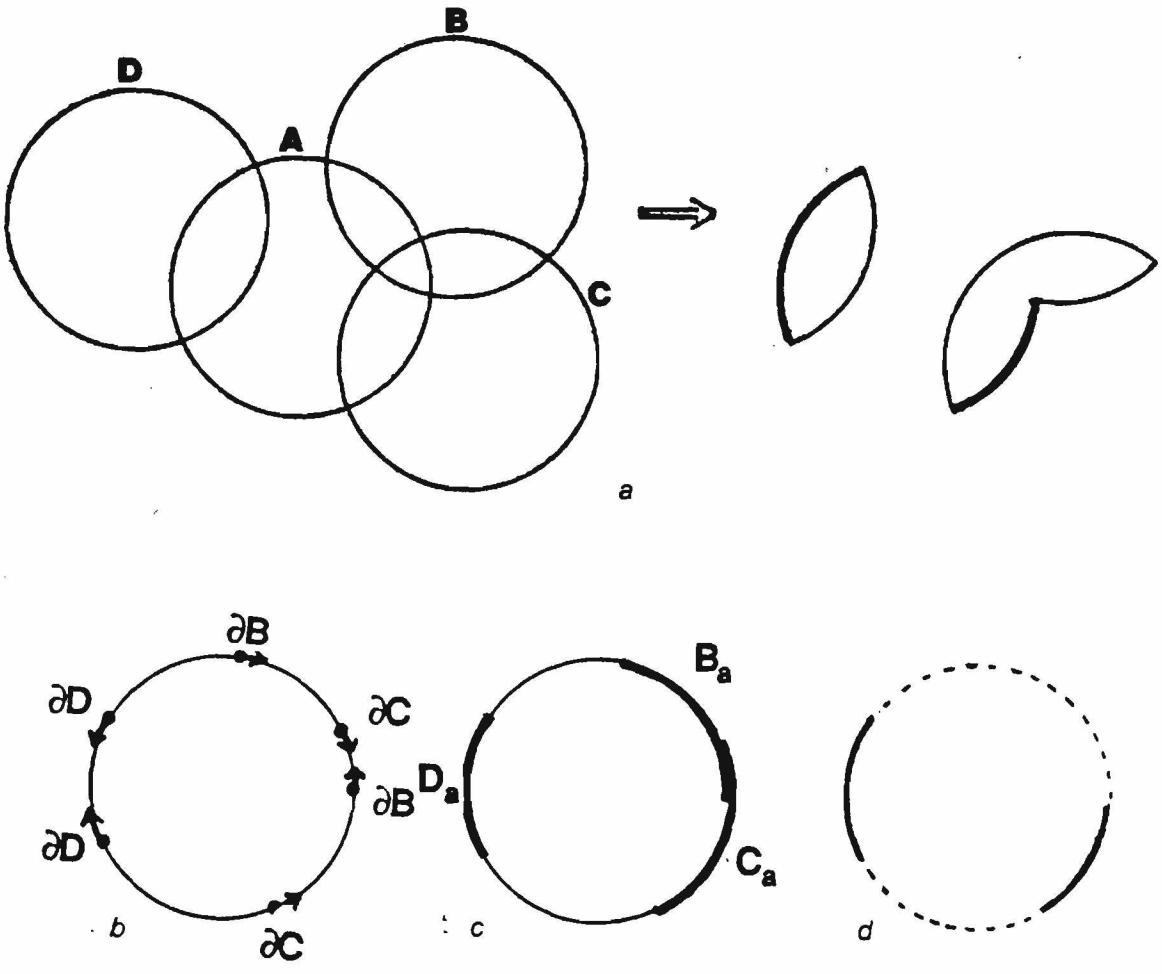


Figure 9: Cut points and sets

which, when evaluated as a set expression, gives the result shown in Figure 9*d*.

In Algorithm 3, the recursion never terminates. At some level, it is necessary to explicitly evaluate a set expression. If the sets are one dimensional, i.e., parts of a curve, the intersection of the zero dimensional boundaries would lead to -1 dimensional intersections, a patent absurdity. It is reasonable, then, to stop when $n = 1$. For one dimensional sets, the algorithm becomes

Algorithm 4

1. For each set, X, do:
   (a) Calculate the boundary expression for X.
   (b) Evaluate the boundary expression at each boundary point, x_i, of X to determine whether x_i is inside the result or not. If it is inside, keep it as a bounding point of the result, otherwise, discard it.
2. The points which were kept bound the result of the set expression.

Look at Figure 10, which repeats the situation shown in Figure 9*c* as a one dimensional set expression. The set expression to be evaluated is

$$\overline{(B \cap C)} \cap C \cup D.$$

Applying the transformation procedure gives the boundary expression below for the boundaries of the set $\overline{B}$:

$$\begin{aligned} & \overline{(B \cap C)} \cap (C \cup D) \\ &= (\overline{B} \cup \overline{C}) \cap (C \cup D) \\ &= \{\overline{B} \cap (C \cup D)\} \cup (\overline{C} \cap D) \\ &\rightarrow \partial\overline{B} \cap \{\overline{(\overline{C} \cap D)} \cap (C \cup D)\} \\ &\rightarrow \partial\overline{B} \cap C \end{aligned}$$

At b_1, the expression C evaluates to **out**, but at b_2 it evaluates to **in**. Thus, the point b_2, with its orientation reversed, bounds part of the result, as can be seen in Figure 9.

Why use this complicated procedure over simple binary evaluation? There are several reasons:

- It can be more efficient. Computing the boundary expression for each set is a relatively inexpensive operation compared to actually performing the evaluation of a set operation. If an expression is evaluated pairwise, more boundary evaluations will be necessary than if the region of each boundary that will appear in the result of the expressions is computed once.
- It is necessary when using partially bounded sets (see below). For example, consider the "wavy" cube in Figure 11. It is formed as the intersection

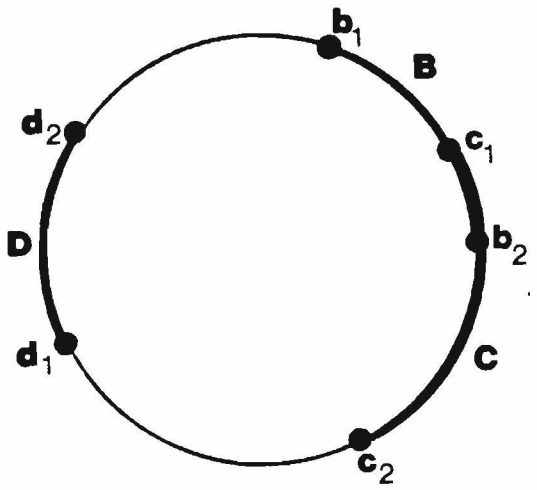


Figure 10: Cutting in one dimension

of six partially bounded sets. If any pair of boundaries is considered, the boundary of their intersection is indeterminate. By evaluating the boundary expression for each considering all the sets at once, the boundary can be found.

- We may be able to discover features of the solution that can simplify the evaluation process. In the example illustrated in Figure 9, the boundary of B is unaffected by D (and D by B), so that B and D need not be intersected with each other. We are not able to tell this easily from the original expression.

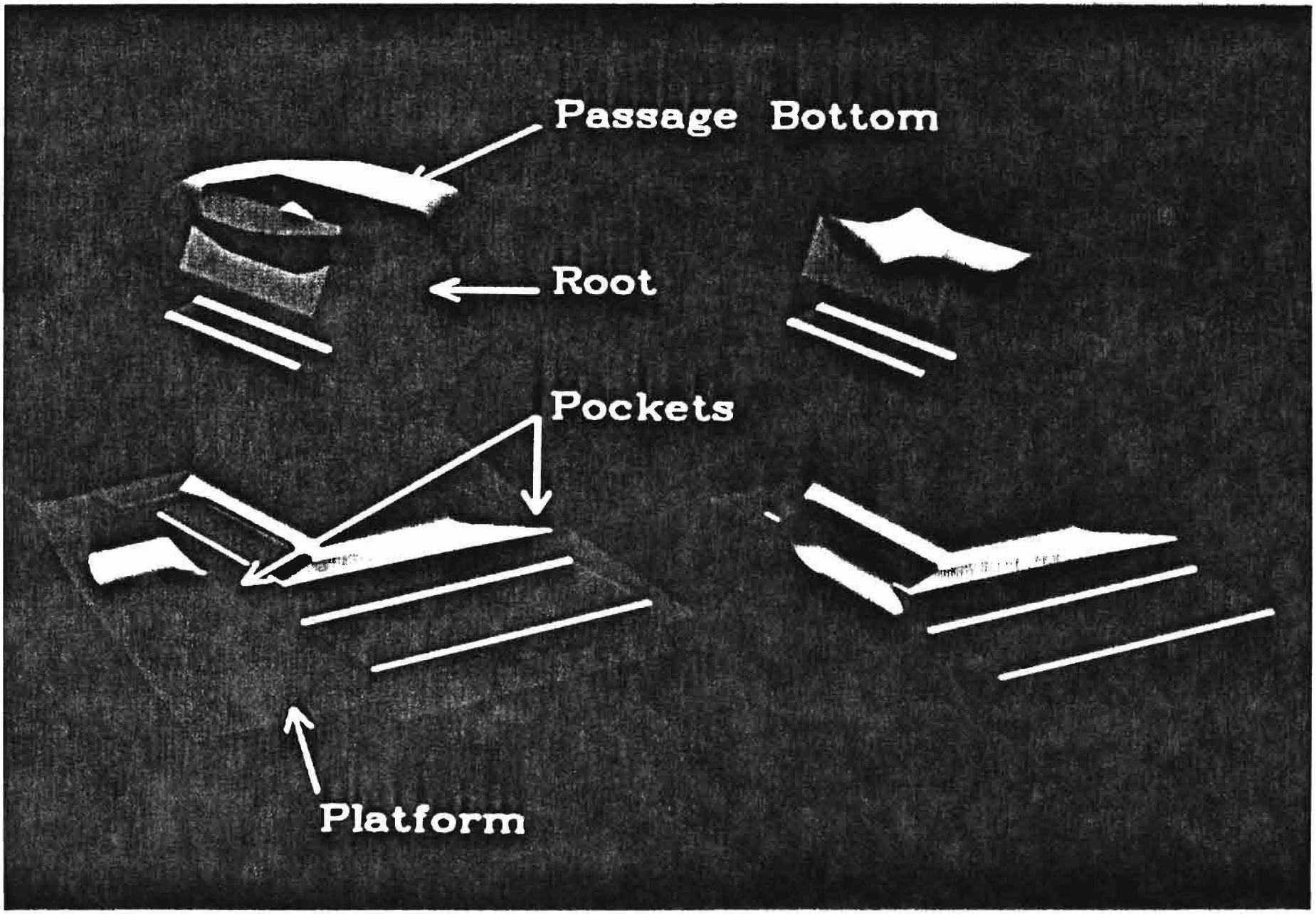
- Sometimes the solution may be independent of a set, so that it need not be evaluated for that set at all.

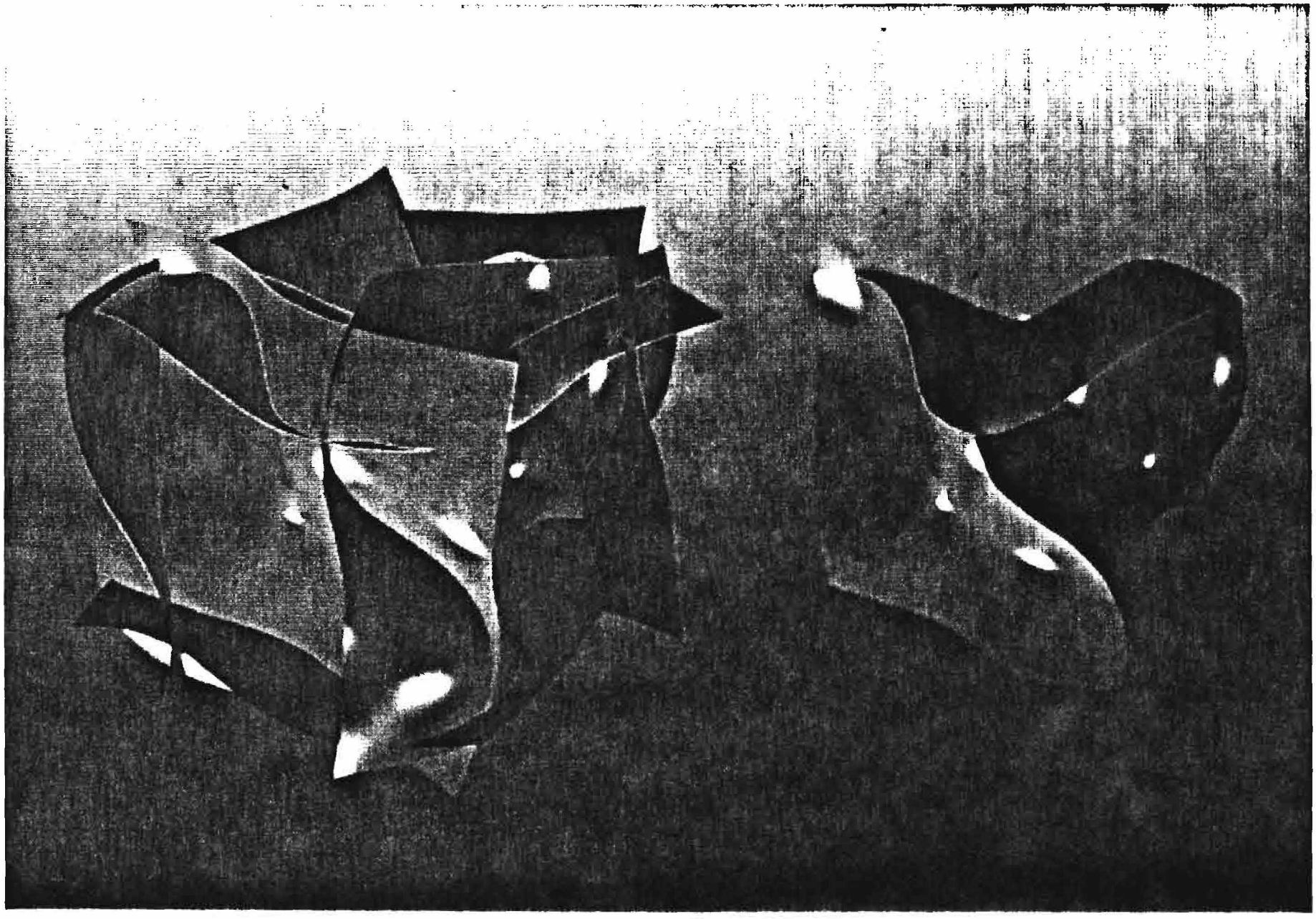
5 Partially Bounded Objects

The surfaces comprising a sculptured boundary model may not form a closed boundary. Furthermore, it may be difficult to create new bounding surfaces so that the boundary is closed. But, it would seem that we must have closed boundaries in order to do set operations. We shall see that this is not true, that set operations can be performed on partially bounded sets, provided that they satisfy some fairly simple criteria. The boundary set expressions derived in the previous section provide a solution method.

Set operations on partially bounded objects can be interpreted in terms of the corresponding operations on completely bounded objects. Conceptually, the partial boundaries are completed, the complete objects are operated upon,

Figure 11: A "wavy" cube


Passage Bottom
Root
Pockets
Platform



University of Utah
Computer Science

CS-238

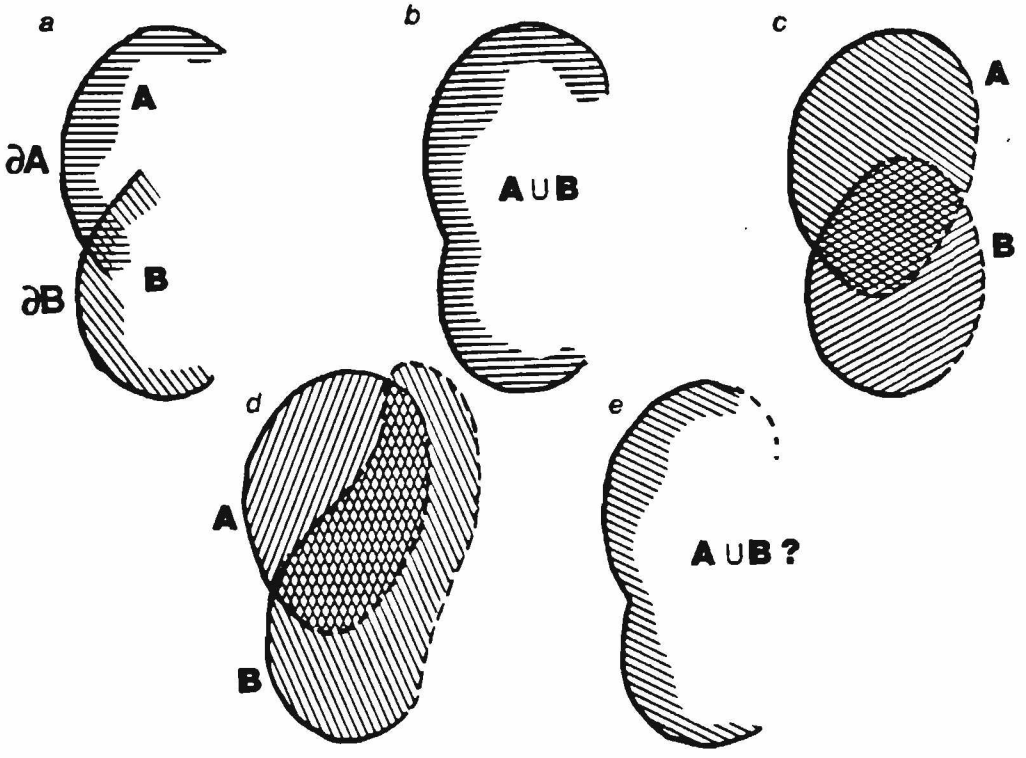


Figure 12: Completion of partial boundaries in set operations

and any portions of the completions which remain in the result are removed. The completion does not actually take place, so it must not affect the final result. In a *valid completion*, the completing portions of the boundaries must not intersect the original partial boundaries, except in those areas of the partial boundaries which do not appear in the final result. One way to ensure this is to use only the intersections between the original boundaries to bound the boundary regions that appear in the result. Thus, in the two dimensional case only the intersection points between the given boundaries may be used to derive the results.

Figure 12*a* shows two boundaries ∂A and ∂B, which partially bound objects A and B, respectively. The shading shows the boundary orientations. The partial boundary of the union of A and B is shown in *b*. A possible valid completion is shown in *c*. The completing portions of the boundaries are dashed. The completion shown in *d* is not valid because it would lead to the boundary shown in *e* as the partial boundary of the union of A and B. The missing piece of the boundary is shown as a dashed curve.

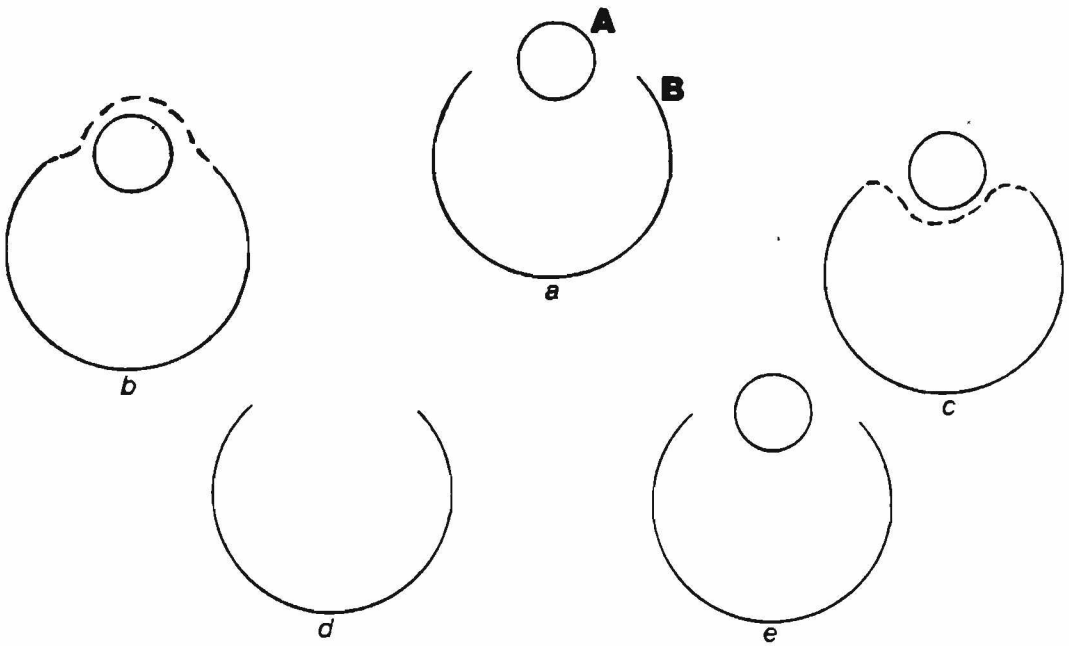


Figure 13: Ambiguous results

For consistency and reproducibility of the operation, any valid completion of the partially bounded objects should produce the same final result. If there are valid completions which produce different results, then the original data is *ambiguous*. An example of ambiguous data is shown in Figure 13. The two partially bounded sets A and B shown in a have an ambiguous union (or intersection). b and c show two possible completions, which result in the boundaries for the union shown in d and e, respectively. In the absence of extra information, either solution is correct.

A further condition on the operands and set expression arises from the rearrangement of the expression that is done in deriving the boundary set expressions. That is, that the result of the set expression must not depend on the order in which the expression is evaluated. This is always true for closed boundaries, but may not be true for some partially bounded objects.

An example is shown in Figure 14. We wish to compute the result of $(A - C) \cup B$, as shown in e. The boundary expression for B is $\partial B \cap (\overline{A} \cup C)$. The restricted set expression in the boundary of B is thus $\overline{A_b} \cup C_b$, with the sets A_b and C_b illustrated in b. The boundary expression for $\overline{A_b}$ is $\overline{A_b} \cap \overline{C_b}$. This cannot be evaluated unambiguously at the points bounding $\overline{A_b}$, as C_b has only one bounding point.

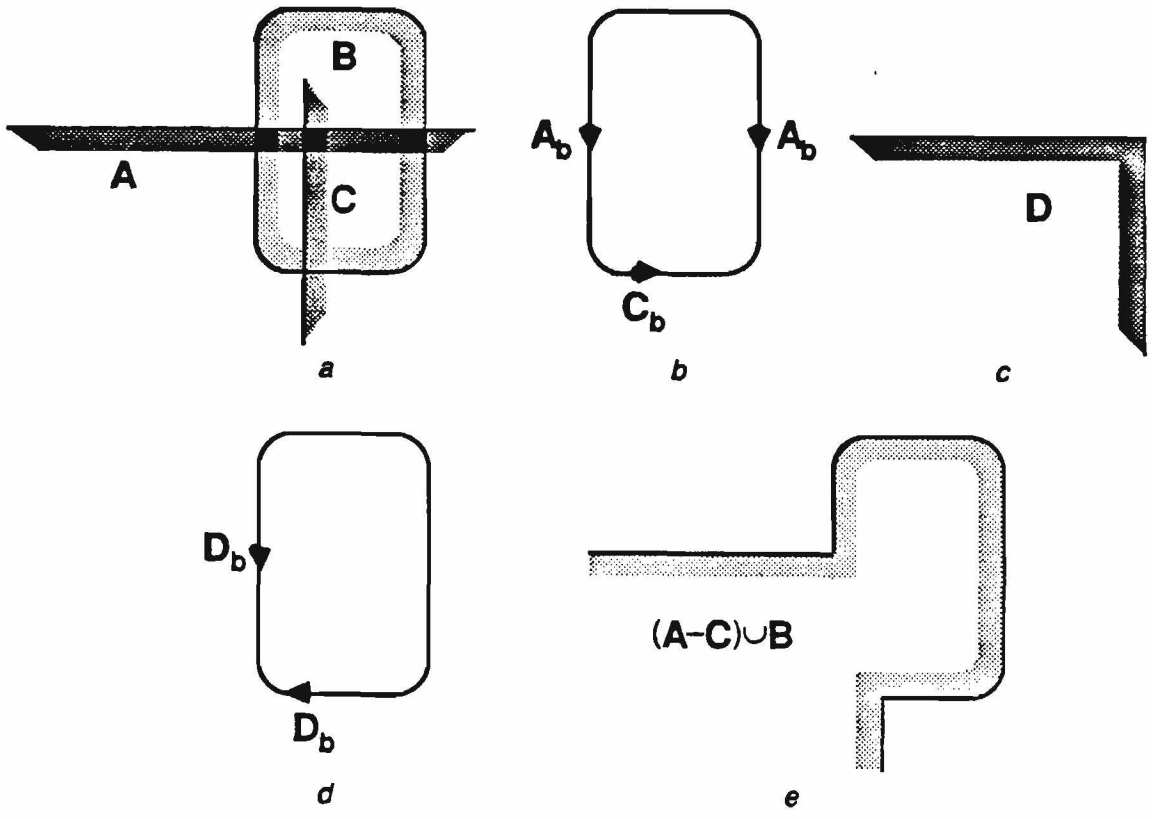


Figure 14: Importance of evaluation order

However, if we first evaluate $A - C$, we obtain the set D shown in *c*. The boundary expression for B is then $\partial B \cap \overline{D}$, which induces the restricted set on ∂B shown in *d*. This can be easily evaluated to give the final result in *e*.

The example shown in Figure 15 demonstrates why it is sometimes necessary to evaluate an entire expression, rather than evaluating the individual operations pairwise. When computing the result of the expression $A \cap B \cap C$, one could first compute $A \cap B$, and then intersect that result with C. However, in this example, it is not possible to compute the intersection of A and B, since the intersection curve of ∂A with ∂B does not completely divide ∂A into separate regions. However, the expression $\partial A \cap (B \cap C)$ can be evaluated by recursively evaluating the expression $B \cap C$ in the boundary of A. Letting B_a and C_a represent the restrictions of the sets B and C to the boundary of A, we get

$$\partial(B_a \cap C_a) = [\partial B_a \cap C_a] \cup [\partial C_a \cap B_a].$$

The two boundary expressions above can be easily evaluated, yielding the boundary of $B_a \cap C_a$ shown in Figure 15*c*. The final result is shown in *d*.

Another such example was illustrated in Figure 11. No pair of surfaces in this figure divide each other into disjoint regions; the expression can only be evaluated by considering all surfaces simultaneously.

6 Coincident surfaces

So far, we have ignored the case of coincident or shared boundaries. For full utility, we must rectify this lack, and integrate shared boundaries into the algorithm. We will do this by essentially removing the shared boundaries from the problem, by redefining them as boundaries of new objects. How is this done?

Figure 16 shows two sets, A and B, and their intersection. The boundary of the intersection contains a section of boundary shared between ∂A and ∂B. Since the boundary expression method operates on the boundary of each set independently of the others, it would produce either two coincident boundary pieces, one from each of a and b, or no boundary at all in that region. The shared boundary piece needs to be identified and isolated as a separate boundary, allowing it to be cut normally.

In Figure 17*a* the shared boundary piece has been removed from the boundaries ∂A and ∂B, and is labeled ∂C. The remaining pieces of ∂A and ∂B are labeled $\partial A'$ and $\partial B'$, respectively. The boundary piece ∂C can be considered to partially bound a set C, $\partial A'$ a set A', and $\partial B'$ a set B', as shown in Figure 17*b*. Then,

$$\begin{aligned} A &= A' \cup C, \text{ and} \\ B &= B' \cup C, \end{aligned}$$

as illustrated in *c*.

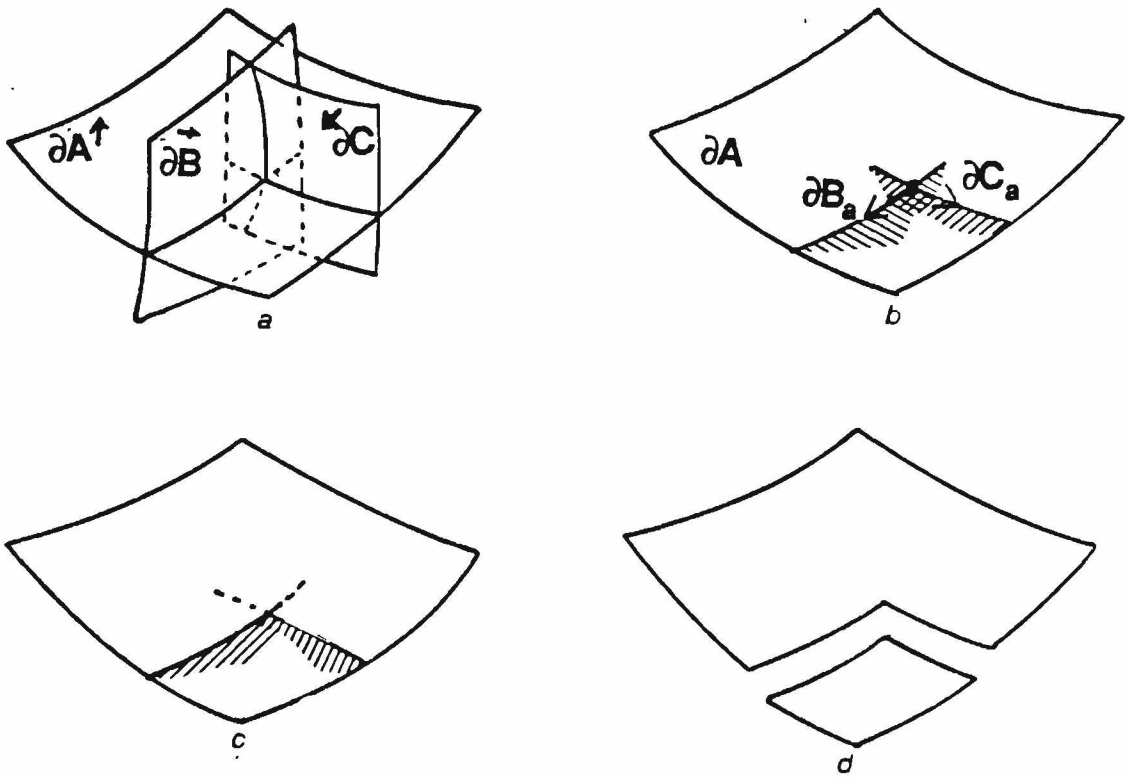


Figure 15: Dangling edges

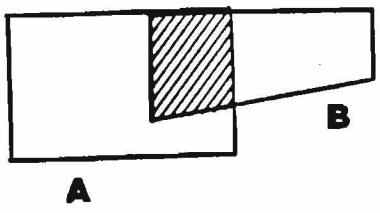


Figure 16: Intersection of two sets with shared boundaries

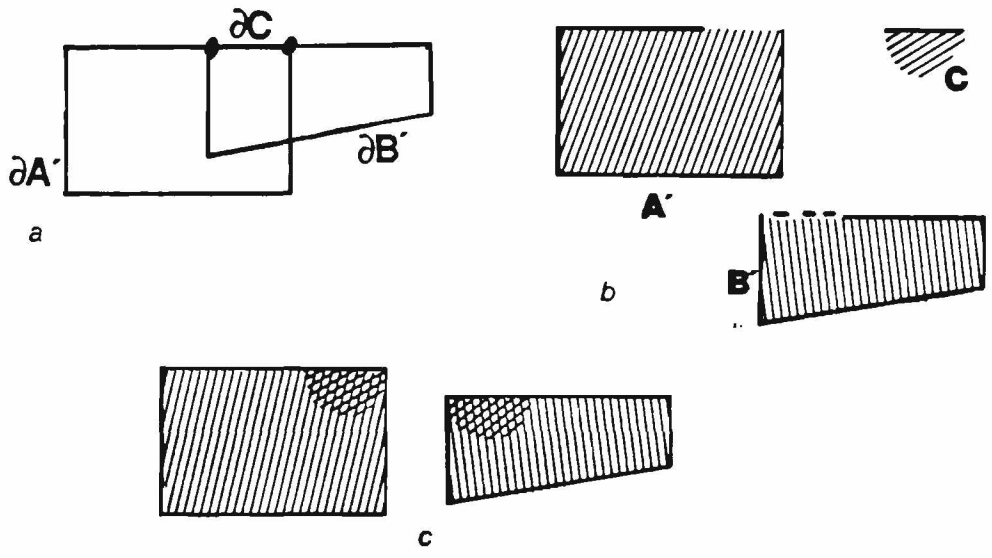


Figure 17: Isolating the shared boundary section

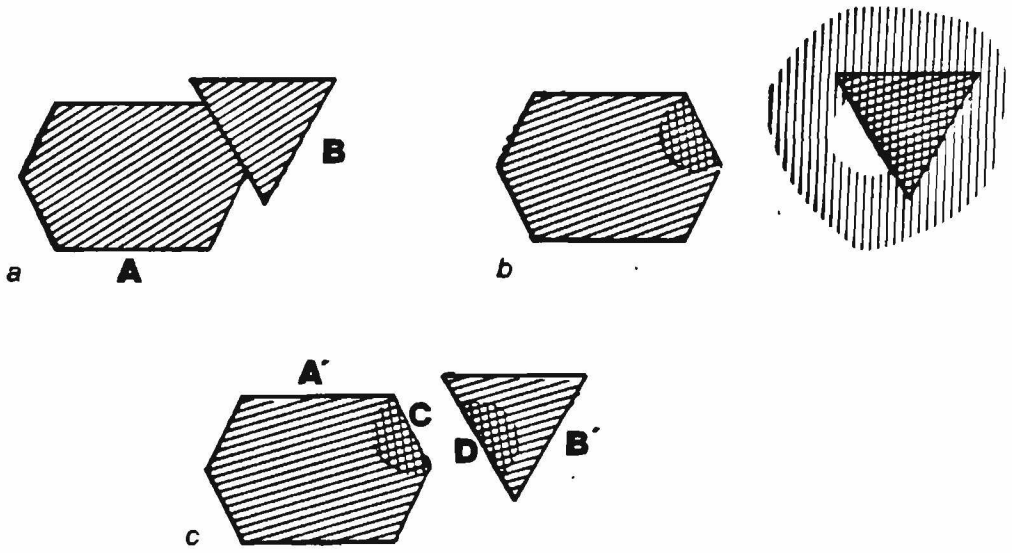


Figure 18: Union with antishared boundaries

With these changes, the set expression becomes

$$A \cap B = (A' \cup C) \cap (B' \cup C).$$

Note that ∂C does not cut either $\partial A'$ or $\partial B'$, so that when computing the boundary expression for A', C may be dropped from the expression wherever it appears. Similarly, C is not intersected by either A' or B', and they may be discarded from the boundary expression for C. Thus, the boundary expressions for A', B', and C are

$$\begin{array}{l} \partial A' \cap B' \\ \partial B' \cap A', \text{ and} \\ \partial C \cap \mathcal{U} \end{array}$$

Now, consider Figure 18, in which A and B have an antishared boundary section. Again, the region is detected and isolated, but now the question of orientation must be considered. If a single partially bounded set C is created, it is not true that

$$\begin{aligned} A &= A' \cup C, \text{ and} \\ B &= B' \cup \overline{C}, \end{aligned}$$

as might be expected by analogy with the shared case. Figure 18*b* illustrates the problem. No matter how a completion is chosen for ∂C, one of the unions

above will give the wrong result. Instead, two partially bounded sets must be created, one with a boundary shared with ∂A, and the other with boundary shared with ∂B. In *c*, these two sets have been labeled C and D, respectively, and it can be seen that

$$\begin{aligned} A &= A' \cup C, \text{ and} \\ B &= B' \cup D. \end{aligned}$$

Note that C and D have the property that

$$\partial(C \cup D) = \partial(C \cap D) = \emptyset$$

with respect to the existing partial boundaries.

Again, when creating boundary expressions for A' and B', C and D can be discarded, and A' and B' can be discarded from the boundary expressions for C and D. In the boundary expression for C, D must be treated as $\overline{C}$ and vice-versa.

Algorithm 2 in Section 4 is modified to include treatment of shared and antishared surfaces below.

Algorithm 5

1. Find all surface pieces shared, regardless of orientation, between two or more surfaces. Give the surface piece an orientation.
2. For each surface piece Q found in step 1, for each surface X containing Q,
   (a) Remove the piece Q from the surface X, giving X'.
   (b) If Q is shared with X, replace all occurrences of X in the expression with $X' \cup Q$.
   (c) If Q is antishared with X, replace all occurrences of X with $X' \cup R$, where R is identical to Q, but with opposite orientation.
3. Proceed as in Algorithm 2, except that when building the cut expression for X, discard all occurrences of Q and R. When building the cut expression for Q or R, discard all occurrences of X' and $\overline{X'}$, and treat R as $\overline{Q}$ or Q as $\overline{R}$, respectively.

Since all the shared and antishared surface pieces are eliminated and replaced by independent surfaces, the boundary algorithm can be directly applied to generate the desired result.

We must show that simply discarding the shared segment from the set expression when deriving the boundary expression for a given surface does indeed produce the correct boundary expression. Consider a surface ∂X that is the disjoint (regular) union of two surfaces ∂Y and ∂Z (Figure 19). The surface ∂X is also the boundary of the set which is the union of the sets partially bounded by ∂Y and ∂Z, in one of the possible classes of completions. Therefore, X can be

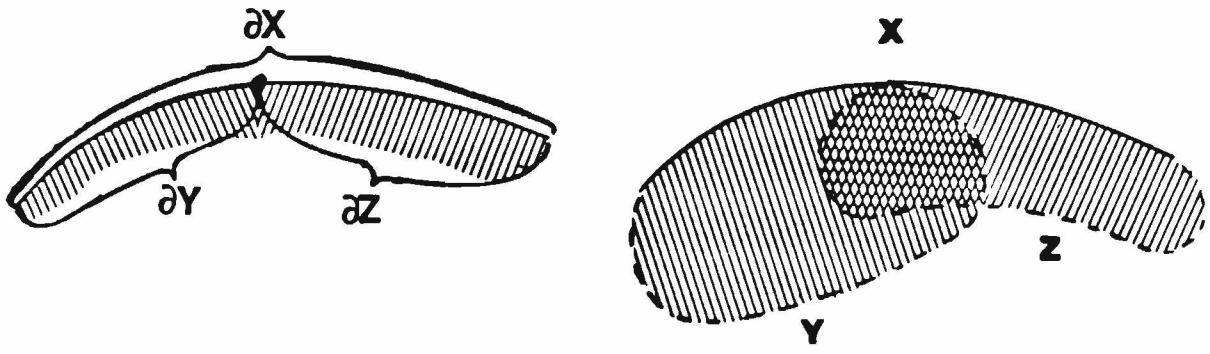


Figure 19: A boundary as a union of parts

replaced by $Y \cup Z$ without affecting the result in any expression containing it. The above procedure replaces all shared and antishared boundary regions with independent boundaries in a set expression. Since ∂Y and ∂Z are disjoint surfaces, removing Z from the boundary expression for Y cannot affect the result, but merely simplifies the calculations necessary to achieve it.

Note that the test for shared boundary components needs to be carried out at each dimensional level. Two surfaces that are not coincident may nevertheless intersect a third surface along a shared curve.

7 Conclusions

We wish to create solid models of sculptured, free-form objects. An essential part of a solid modeling system is the ability to combine objects with volumetric set operations. We have presented a method for evaluating the result of expressions of set operations on free-form boundary models using *boundary expressions.*

A boundary expression, when evaluated, gives exactly the portion of a given boundary that forms part of the boundary of the result of a set expression. It is evaluated by recursively deriving new boundary expressions of lower dimension. A set expression is evaluated by evaluating the boundary expression for each of the operand sets. The use of boundary expressions for evaluation of set expressions provides several advantages over pairwise evaluation. These include:

- increased efficiency,
- simplified evaluation process, and
- ability to use partially bounded sets.

Sculptured surfaces are not of infinite extent, nor do they typically form closed boundaries. For ease of use by a designer, it is important to recognize this fact, and to be able to use *partially bounded sets* in set expressions. Provided that a completion criterion is satisfied, expressions involving partially bounded sets may be evaluated using the boundary expression method.

Treatment of coincident boundaries is always problematic. We are able to eliminate shared boundary regions in a preprocessing step by treating them as separate, partial boundaries.

The work described here provides a method for integrating sculptured, free-form surface models into a solid modeling system. It extends the range of objects that are representable to include many important, previously unmodeled classes of parts.