A Solution of Polygon Containment, Spatial Planning, and Other Related Problems Using Minkowski Operations

PIJUSH K. GHOSH

National Centre for Software Technology, Gulmohar Cross Road No. 9, Juhu, Bombay 400 049, India

Received May 23, 1988; revised February 15, 1989

This paper gives a complete solution to the polygon containment problem under translation and other related problems for all kinds of two dimensional regions. The solution is achieved in three steps. First, it is shown that the containment and the related problems can be directly mapped to Minkowski decomposition and addition problems. Minkowski decomposition, which is intrinsically a geometric problem, is then reformulated in terms of set operations and set theoretic tools are used to reduce the computational complexity of the problem. Finally, a new technique, termed as decomposition boundary tracing technique, is devised and employed for the solution of the decomposition problem. This new technique brings out a unified algorithmic approach to solve all kinds of decomposition problems in a two dimensional space, and thereby the containment problem in its entirety.

1. INTRODUCTION

1.1. The Problem and the Motivation

The polygon containment problem is to decide whether a polygon B can be translated to fit inside another polygon S, and if it fits then to compute the set of safe positions in which B fits inside S. This set of safe positions is also known as feasible region.

The concept of polygon containment has a number of salient applications in the real world. An immediate application is in stockcutting, i.e., in cutting successively a polygonal object B out of a sheet of material S. The tolerance inspection problem often arises in a manufacturing process, in a vision system, or in template matching. The problem is as follows. In 2-dimension, the task is to generate a given polygonal object B, but one is getting a polygon B', not necessarily equal to B because of noise, fabrication, or other generation errors. Tolerances are given in terms of two polygons S₁ and S₂. One, say S₁, lies outside B and the other, i.e., S₂, lies inside B. If the generated polygon B' is completely contained in S₁, and S₂ is completely contained in B', then B' is considered to conform to the desired tolerances [1].

Another important application of polygon containment is seen in spatial planning. The spatial planning problem involves placing an object among other objects or moving it without colliding with nearby objects [9]. Let S be an object that contains n other objects H_j's. Let B be a new object. In a real situation, S may be the entire floor space, H_j's may be the obstacles in the floor, and B may be a robot vehicle or a machine tool. The spatial planning is the problem of finding safe positions for B inside S, such that there is no intersection between B and any of the obstacles H_j's. That means, $H_j \cap B = \emptyset$. The problem of finding a path for B to move it from one position to another position is an equivalent problem, since a path is nothing
but a connected sequence of safe positions. In 2-dimension, the spatial planning problem, therefore, is equivalent of computing the feasible region for a polygonal object B inside a polygonal object S that has n number of holes H_s in it. The problems like finding out the intersection of two polygons or determining the distance between two polygons are also special cases of the polygon containment problem.

Certain practical situations demand that the polygon containment problem must deal with both translation as well as rotation of B. For example, changing the orientation of the robot vehicle at some position might make it possible to move the vehicle through the obstacles without collision, which was otherwise not possible by translation alone. But in this paper, we tackle the problem by keeping the orientation of B fixed. The exact problem we handle here can be stated as follows.

**Problem 1** (polygon containment under translation). Two polygons S and B are given; the polygons might have holes in them. Compute the feasible region in which B can be translated to fit inside S.

We solve the problem by using Minkowski addition (vector addition) and Minkowski decomposition (inverse of addition) operations. Our approach is very different from the existing ones. The primary advantage of our approach is that it allows us to formulate a unified containment algorithm for all types of S and B polygons. The unified method can easily be extended to 3-dimensional as well. In addition, we also develop a theoretical framework for Minkowski decomposition (which is the primary tool for solving the containment problem) and show how set theoretic results can be used to reduce the complexity of a geometric problem. Our treatment of the problem is briefly sketched in Section 1.3.

### 1.2. Earlier Works

The problem had been studied by a number of researchers in various contexts. Vaishnavi and Wood [15] and later Lee and Preparata [8] investigated the rectangle containment problem that involves only rectangular polygons. Chazelle [1] reported a linear time algorithm in the case where both S and B polygons are convex. If the convex polygons S and B have N_S and N_B number of edges, respectively, and are given in slope-sorted order, Chazelle's algorithm computes the feasible region in time $O(N_S + N_B)$ or $O(N)$, where $N = \text{Total number of edges involved} = N_S + N_B$. Recently Baker et al. [1] also obtained the same complexity, but by using somewhat different techniques. They reported an $O((N_S N_B + N_S^2) \log N_S N_B)$ algorithm when B is convex but S is a simple (nonconvex) polygon.

Although the reported algorithms vary considerably in geometric techniques and data structures, the fundamental idea is same everywhere. Place B inside S such that B touches one of the edges of S; translate B continuously along that edge and test whether B crosses the boundary of S at any point; if B hits the boundary, then choose the next edge of S and translate B along that edge in the similar manner, and so on. The translational path of B gives the boundary of the feasible region (Fig. 1). All these algorithms are also similar in another regard. The basic module inside the hearts of these algorithms is a containment algorithm for two convex polygons. To solve the general problem, the two given polygons are first broken up into convex parts and at the final stage intermediary results are combined to obtain the solution. These kinds of algorithms can be termed as the move and test algorithms.
FIG. 1. The basic idea behind existing polygon containment algorithms.

In Fig. 1 we demonstrate how the $B$ polygon is translated along the edges of $S$ and show by dark dotted lines the boundary of the feasible region. Note the fact that a placement of $B$ is uniquely determined by a placement of a specific point of $B$, since only translational motion of $B$ is allowed. We may call this specific point the reference point $r$. The boundary of the feasible region is, in fact, the translational path or the locus of $r$. If we choose some other reference point, say $r'$, the feasible region is only shifted/translated, but its shape remains unchanged. The boundary of the feasible region with respect to $r'$ is shown by light dotted lines in Fig. 1. In this paper we are primarily concerned with the shape of the feasible region, without worrying much about the location of the reference point. Once the shape of the feasible region is obtained, it can be translated appropriately.

1.3. Our Approach and Objective

Our approach is completely different from the existing move and test approach. The problems concerning interactions between objects (for example, containment of one object within another, movement of an object among obstacle objects, distance between two objects, etc.) become relatively simpler if the moving object $B$ is a point instead of an object with finite size. Consider, for example, our polygon containment problem. If the given $B$ is a point, then the entire $S$ polygon itself becomes the feasible region. We, therefore, try to derive a transformation operation so that $B$ in the real space is transformed to a point, say to the reference point $r$, in the transformed space. That transformed space is commonly known as configuration space, or simply $C$-space. In the $C$-space, since $B$ shrinks to a point, the other objects must also shrink accordingly. On the other hand, if there are holes in the other objects, they must grow in accordance (Fig. 2).

In Fig. 2 the object $S$ is a polygon with a hole and we choose $r$ as the reference point of $B$. After transformation the polygon $B$ is transformed to the point $r$, while $S$ is transformed to a polygon $T$. The nature of the transformation must be such that when $r$ is inside the $T$ polygon in the $C$-space, $B$ must be completely contained in $S$ in the real space; on the other hand $B$ must not be completely contained in $S$ when $r$ turns out to be outside of $T$. The polygon $T$, therefore, represents the required feasible region.

However, the fundamental question is: what is that transformation operation? Because once we have that transformation operation, our task is merely to transform $S$ accordingly, and that transformed $S$, i.e., $T$ will be our feasible region.

We show that Minkowski decomposition operation takes care of the shrinking of an object exactly in the way we want to shrink them in the $C$-space. The growing of holes, in accordance with the shrinking of the object, is the inverse process, and is done through inverse of Minkowski decomposition. This inverse operation is commonly known as Minkowski addition or vector addition.
Our solution of the containment problem, therefore, reduces to two steps:

1. to establish first the formal relationship between Minkowski operations and the containment and other related problems;
2. to devise then the appropriate algorithms to carry out Minkowski operations on polygonal objects.

To achieve this aim, we organize the paper in the form:

1. In Section 2 we briefly state the primary notations and definitions that are followed in this paper. The formal definitions of Minkowski addition and decomposition are also included in this section.

2. The connections between Minkowski operations and the polygon containment, the spatial planning, the polygon intersection problems etc. are established in Section 3. It is shown that Minkowski decomposition of a polygon $S$ by a polygon $B$ is exactly the same as the containment of the polygon $B$ inside the polygon $S$. In other words, the containment problem can be directly mapped to the Minkowski decomposition problem. In this paper we decide to concentrate only on the decomposition problem and not on the Minkowski addition problem, since the latter operation is needed only if the polygons have holes. Moreover, the problem of Minkowski addition has been tackled in great detail in other places [3].

3. In Section 4 we start devising algorithms for Minkowski decomposition on various kinds of polygons. It may appear at first that the algorithms for decomposing a convex polygon by a convex polygon, a simple polygon by a convex polygon, a simple by a simple, etc., will all be different. Interestingly a new notion, termed as decomposition boundary tracing, emerges from the way we formulate these algorithms. This notion allows us to devise a unified algorithmic technique to carry out all kinds of decomposition within the same framework. This section is divided into three parts.

(a) In the first part we attack Minkowski decomposition problem from a set theoretic point of view. The decomposition problem is generalized as the decomposition of a set by another set in the $d$-dimensional real euclidean
space, instead of a polygon by a polygon. Then the problem is reformu-
lated in terms of set operations. A number of set theoretic results
concerning Minkowski decomposition of two sets are stated and proved.
We show how these set theoretic results could be effectively used to
reduce the computational complexity of our problem that is very much
geometric in nature. The results we obtain in this part are applicable for
general sets and thereby allow us to go beyond 2-dimensional space and
polygonal regions.

(b) In the second part, however, we decide to restrict ourselves to 2-dimen-
sion. We start by presenting an algorithm to decompose a convex
polygon by another convex polygon (will be referred to as the
convex-convex case for convenience). The important contribution at this
part is to introduce the notion of decomposition boundary tracing in
formulating the algorithm. We also show here that the complexity of our
algorithm does not increase because of our formulation. It is also linear,
as was obtained by the previous researchers.

(c) In the third part we intend to examine whether the notion of a decompo-
sition boundary tracing technique could be extended beyond convexity.
Therefore, we undertake the task of decomposing a simple polygon by a
convex polygon (will be referred to as the simple-convex case). The
tracing technique does not need any modification for the convex parts of
the simple polygon. The nature of output, i.e., the tracing also remains the
same as in the case of the convex-convex case. We show that for the
nonconvex parts of the simple polygon also, the tracing technique re-
mains exactly the same but only the nature of the tracing changes. By
following our approach we are also able to get an algorithm of the same
complexity as obtained by Baker et al. Moreover, it appears that the
average case complexity of our algorithm could be reduced considerably
by devising more efficient data structures. However, we decide not to try
out those finer details in this paper.

4. Once it is established that the same decomposition boundary tracing tech-
nique can be used for both convex-convex and simple-convex cases, the possibility
of forming a unified approach to all decomposition algorithms in 2-dimension
emerges. In Section 5 we analyze the basic concept behind the tracing technique and
propose how this concept can be universally used for all kinds of 2D decomposi-
tions. The concept behind the tracing technique is expressed in the form of two
propositions. The applicability of the tracing concept beyond convexity and also
beyond polygonally defined regions is more strongly established by devising algo-
rithms for the simple-simple case and for regions that are bounded by smooth
boundary curves.

5. Section 6 is the concluding section. We briefly indicate how our approach
could be extended to 3-dimensional and also suggest a few important problems for
further investigation.

We must also mention the following facts regarding our paper.

1. The term configuration space is not coined by us. Lozano-Perez [9] previ-
ously used the term and also the concept of growing obstacle objects in connection
with the spatial planning problems. Our definition of configuration space, however, is a more general concept that accommodates both growing as well as shrinking of objects, since they are just the inverses of each other. Thus the concept of our configuration space covers a wide range of problems concerning interactions among objects and their movements in space.

2. In recent years a general method for image processing, known as mathematical morphology, has become exceedingly popular and is growing rapidly as a separate discipline. Interestingly, Minkowski operations form the kernel of all the mathematical morphological operations. In fact, two of the morphological operations, known as dilation and erosion, are nothing but Minkowski addition and Minkowski decomposition, respectively. Although mathematical morphology is primarily concerned with the processing of digital images, in morphological literatures many results on Minkowski operations were stated for general sets of points in $d$-dimensional real euclidean space. Therefore, the statements of some of the theorems and equations presented in Sections 3 and 4.1 in this paper are not completely new. However, in certain cases, results were stated in morphological literature without any formal proof. Therefore, we decided to provide separate proofs for all of them which are also more algebraic in nature. The interested readers may refer to Matheron [10], Serra [12], or Harlick et al. [6] for alternative proofs of some of the theorems.

3. The proof of the proposed unified theory of decomposition is presented in a more informal way compared to the proofs of the theorems in the first part of the paper. It has served a number of useful purposes. First, quite a few interesting geometric properties of Minkowski decomposition have been unravelled during the discussion which could not have been done otherwise. Second, we are able to show how a general approach to polygon decomposition has gradually evolved as we progressively reduce the constraints on the input polygons. Moreover, the theory of Minkowski addition was developed rigorously by a number of researchers [4, 10, 5, 3]. Since Minkowski decomposition is the inverse of Minkowski addition, it is possible to formulate more formal proofs of the unified theory of decomposition along the same line of addition. But we feel it is more appropriate to introduce a different line of thought and, therefore, do not make use of any of the theorems of Minkowski addition. The decomposition theory also becomes more self-contained in that process.

2. BASIC NOTATIONS AND DEFINITIONS

2.1. Notations

The objects we consider in this paper are the sets of points in $E = R^d$, i.e., in $d$-dimensional real euclidean space. These sets are generally denoted by capital letters $A$, $B$, $S$, $T$, etc. We consider only the compact sets, i.e., which are both bounded and closed. Any point $p$ in such a set is a vector with $d$-tuple $(x_1, \ldots, x_d)$. Most of the results we prove will be applicable for such general sets of points. A point is generally denoted by a small letter such as $a$ or $p$ or $v$.

To distinguish between the boundary of a positive region and a hole, the notion of oriented boundary is introduced in Section 4.

The notations $\cup$, $\cap$, $\setminus$, and $^c$ are used to denote set union, intersection, difference, and complement, respectively. The notation $"+"$ is used to de-
note Minkowski/vector addition with a point, whereas "\( \oplus \)" is used for denoting Minkowski addition of two sets. Similarly, "\(-\)" and "\(\ominus\)" notations are used for Minkowski decompositions.

The letter \( N \) is kept reserved for denoting the number of vertices or edges of a polygon. The other definitions or notations that we use are stated at the places where they first occur in our presentation.

2.2. Minkowski Operations

2.2.1. Minkowski Addition

Minkowski addition can be regarded as a kind of \textit{growing} or \textit{dilation}. A precise definition of Minkowski addition can be given as follows.

\textbf{Definition 1 (Minkowski addition).} Let \( B \) and \( T \) be two arbitrary sets in \( \mathbb{R}^d \) space. The resultant set \( S \) is obtained by positioning \( B \) at every point of \( T \), i.e., vectorially adding all the points of \( B \) with those of \( T \). We denote this by

\[
S = B \oplus T = \{ b + t : b \in B, t \in T \},
\]

where "\( \oplus \)" stands for Minkowski addition.

A simple example in 2D may clarify the idea. Let \( B \) be an elliptical region and \( T \) be an open coplanar cubic curve. If \( B \) is considered as a brush and \( T \) as a trajectory, then the resulting figure \( S \) is obtained by moving the brush along the trajectory (Fig. 3). This process can also be viewed as growing of the curve \( T \) to \( S \), by means of \( B \). We consider that in Cspace \( B \) becomes the reference point \( r \). Note that \( B \oplus T \) in real space is equal to \( S \oplus \{ r \} \) in Cspace. The same concept is true in higher dimensions as well.

Minkowski addition of two sets can also be expressed in terms of set union operation. If \( A_p \) denotes translate of a set \( A \) by the vector \( p \), i.e., \( A_p = A \oplus \{ p \} \),
then it obviously follows that,

\[ S = T \oplus B = B \oplus T = \bigcup_{t \in T} B_t = \bigcup_{b \in B} T_b. \]  

(1)

2.2.2. Minkowski Decomposition

Minkowski decomposition is the inverse of Minkowski addition operation; it can be regarded as *shrinking* or *erasing*. It is denoted by the symbol "\( \ominus \)". Its precise definition can be given as follows:

**Definition 2 (Minkowski decomposition).** If \( S \) and \( B \) are two sets in \( \mathbb{R}^d \) space, then

\[ S \ominus B = \bigcap_{b \in B} S_{-b} = \bigcap_{-b \in \bar{B}} S_{-b}, \]  

(2)

where "\( \ominus \)" stands for Minkowski decomposition operation. We call \( S \ominus B \) as the decomposition of \( S \) by \( B \). (According to our previous notation, "\( S_{-b} \)" means the translate of \( S \) by the vector \( -b \), i.e., \( S \oplus \{ -b \} \). For the convenience of notation, \( S_{-b} \) is also written as "\( S - b \)."

The set \( \bar{B} = \{ -b : b \in B \} \) is generally known as the *symmetrical set* of \( B \) with respect to the origin. Since our final aim is to shrink \( B \) to the reference point \( r \), we choose reference point as the origin point (Fig. 4).

Let us clarify the idea of Minkowski decomposition by an example shown in Fig. 5. The polygon \( S \), we consider, is a trapezoidal region while the polygon \( B \) is a rectangular area. Since \( B \) is centre-symmetric, \( B \) and \( \bar{B} \) are the same in this case. For decomposing \( S \) by \( B \), \( S \) is placed at every point of \( \bar{B} \) and their common intersection region is determined. The resultant intersection is a triangular region as shown separately in the figure.

It might be difficult to conceive that the above definition of Minkowski decomposition is indeed the inverse of Minkowski addition. Our claim that the above definition is the inverse of addition can be established by showing the fact that

\[ (S \ominus B) \oplus B = S, \]  

provided there exists a set \( X \) such that \( S = X \oplus B \).

---

**Fig. 4.** A set \( B \) and its symmetrical set \( \bar{B} \) in 2D.
The above relation can be easily proved by using Eq. (1) and Eq. (2). One derivation of it is given in Ghosh [3]. Readers may also consult Matheron [10]. It is important to note here the necessity of the condition $S = X \oplus B$ for proving the relation. The necessity arises since we have no concept of independent existence of a negative region. Therefore, there does not always exist a set $X$ for any $B$ and $S$ such that $S = X \oplus B$. For example, no $X$ exists if $B$ is a circle and $S$ is a triangle. Now $S \ominus B$ is supposed to give us the $X$. If no such $X$ exists, then $(S \ominus B) \oplus B$ cannot be equal to $S$.

3. RELATIONSHIP BETWEEN MINKOWSKI OPERATIONS AND CONTAINMENT AND OTHER RELATED PROBLEMS

3.1. Polygon Containment Problem

Let us first define the polygon containment problem more formally. Let $T$ denote the set of safe positions, i.e., the feasible region in which a given $B$ fits inside a given $S$. That essentially means that, if $B$ is translated to any point $t \in T$, the translate polygon $B_t$ must be within $S$, i.e.,

$$T = \{ t : B_t \subseteq S \}.$$

In the polygon containment problem, our task is to compute the set $T$.

We now prove the following relation to establish the connection between the containment problem and Minkowski operations.

$$S \ominus B = \text{Feasible region} = \{ t : B_t \subseteq S \}. \quad (3)$$

Proof. Let

$$x \in S \ominus B$$

$$\iff x \in \forall_b S_{-b}, \quad \text{where } b \in B$$

$$\iff \forall_b (x + b) \in S$$

$$\iff B_x \subseteq S$$

Thus,

$$S \ominus B = \{ t : B_t \subseteq S \}.$$
Therefore, the problem of computing the feasible region \( T \) is the same as computing the set \((S \ominus B)\). In other words, the containment problem can be directly mapped to Minkowski decomposition problem. Problem 1 can, therefore, be re-stated in the following form.

**Problem 1a (Minkowski decomposition).** *Given two sets \( S \) and \( B \) in \( R^d \), compute the set \( S \ominus B \), where \( \ominus \) denotes Minkowski decomposition.*

Note that the statement of Problem 1a is much stronger than that of Problem 1. \( S \) and \( B \) are two planar polygons in the former, whereas they can be general sets in \( R^d \) space in the latter. However, in this paper, we emphasize computing \( S \ominus B \) when both \( S \) and \( B \) are polygons in \( R^2 \).

### 3.2. Spatial Planning Problem

It is already pointed out in Section 1 that the spatial planning problem is equivalent to the following problem. Let \( S \) be a set (the entire floor space, for example) having a hole \( H \) (equivalent of an obstacle) in it. The task is to find for a set \( B \) (say, a moving object), the set of safe positions (i.e., the feasible region) in which \( B \) fits inside the positive region in \( S \). In Fig. 6 we show a typical example. The feasible region for \( B \) is shown dotted. Note the forbidden region around the obstacle \( H \), where \( B \) cannot be placed. It may be viewed as if the obstacle \( H \) has grown to the forbidden region when \( B \) has shrunk to the reference point \( r \). The forbidden region can be considered as a hole inside the feasible region.

Now the positive region, say \( S' \), in \( S \) is equal to the entire \( S \) minus the hole \( H \), i.e.,

\[
S' = S \setminus H = S \cap H^c.
\]

Therefore, according to our previous notation,

\[
\text{Feasible region} = S' \ominus B = (S \cap H^c) \ominus B.
\] (4)

Equation (4) can be simplified by using two equations

\[
S \ominus B = (S^c \ominus \bar{B})^c
\] (5)

\[
(S_1 \cap S_2) \ominus B = (S_1 \ominus B) \cap (S_2 \ominus B).
\] (6)
Proof of equation (5). We first claim that,

\[(S^c)_{-b} = (S_{-b})^c.\]

Since,

\[
x \in (S^c)_{-b} \\
\iff x + b \in S^c \\
\iff x + b \in S \\
\iff x \not\in S_{-b} \\
\iff x \in (S_{-b})^c.
\]

Now,

\[S^c \oplus \tilde{B} = \bigcup_{-b \in \tilde{B}} (S^c)_{-b} = \bigcup_{-b \in \tilde{B}} (S_{-b})^c \quad \text{(according to the above result)}.\]

Therefore,

\[(S^c \oplus \tilde{B})^c = \bigcap_{-b \in \tilde{B}} S_{-b} = S \ominus B.
\]

The relation is proved.

Proof of Equation (6). The relation can be proved directly from the definition of Minkowski decomposition. One may also prove it by using Eq. (5).

Now applying Eq. (5) and Eq. (6) in Eq. (4), we obtain

Feasible region = \((S \cap H^c) \ominus B\)

\[= (S \ominus B) \cap (H \oplus \tilde{B})^c\]

or

\[\text{Feasible region} = P \setminus Q, \quad (4a)\]

where

\[P = (S \ominus B) \quad \text{and} \quad Q = (H \oplus \tilde{B}).\]

The physical meaning of Eq. (4a) is: if the moving object \(B\) becomes a point, the entire floor space shrinks from \(S\) to \(P\) and the obstacle grows from \(H\) to \(Q\). Figure 6 demonstrates the same fact.

In solving spatial planning problem, researchers, in general, are more interested in computing the forbidden region around the obstacle \(H\), i.e., in computing \(Q = (H \oplus \tilde{B})\), rather than computing the entire feasible region \((P \setminus Q)\). Therefore, the spatial planning problem can be reduced to the following problem.

**Problem 2 (Minkowski addition).** Given two sets \(T\) and \(B\) in \(R^d\), compute the sum set \(T \oplus B\), where \(\oplus\) denotes Minkowski addition.
Problem 2 was elaborately discussed and tackled by Ghosh [3] as well as by Guibas et al. [5] for 2D regions. Therefore, we shall not handle this problem in this paper.

3.3. Polygon Intersection Problem/Distance between Two Polygons Problem

The polygon intersection problem is to find out whether two given polygons \( T \) and \( B \) intersect or not. We have seen in the previous subsection that if the reference point \( r \) is outside the region \( T \oplus B \), the polygon \( B \) does not intersect \( T \). On the other hand, if \( r \) is inside, \( B \) intersects \( T \). Note that in computing \( \hat{B} \), we have chosen the reference point \( r \) as the origin. Therefore,

Two polygons \( T \) and \( B \) intersect if and only if the origin is inside the sum polygon \( T \oplus \hat{B} \).

(See Guibas et al. [5] for more detail.) Note that the intersection problem now reduces to the point inclusion problem.

Similarly the minimum distance between two polygons \( T \) and \( B \) is the distance of the polygon \((T \oplus \hat{B})\) from the origin. Therefore, these two problems are essentially Minkowski addition problem as stated in Problem 2.

4. SOLUTION OF PROBLEM 1a: COMPUTATION OF \( S \oplus B \)

4.1. Building Up the Set Theoretic Framework

The definition of \( S \oplus B \) immediately suggests a computational procedure. Place the set \( S \) on every point of \( \hat{B} \) and find out the common intersection of all the instances (i.e., translates) of \( S \).

The computational cost could be tremendously reduced if it suffices to place \( S \) only on the boundary points \( \partial \hat{B} \) of \( \hat{B} \), instead of placing it on all the interior as well as on the boundary points. By limiting ourselves on a restricted class of sets \( S \), we can make is possible. Note the following theorem.

**Theorem 1** (boundary reduction theorem). If \( S \) is a simply connected set (i.e., it has no hole in it), then

\[
S \oplus B = S \oplus \partial B,
\]

where \( \partial B \) denotes the boundary of \( B \).

**Proof.** In the first place note the fact that,

\[
(\partial B \subseteq S) \Rightarrow (B \subseteq S),
\]

provided \( S \) contains no hole.

Refer to Fig. 7. The set \( S \), shown hatched in the figure, has no hole in it. The boundary \( \partial B \) of \( B \) is drawn in bold. The topological argument says that if \( \partial B \) can be placed within a simply connected set \( S \), it is then possible to continuously shrink \( \partial B \) up to a point by keeping it always within \( S \). During this topological-shrinking process the different instances of \( \partial B \) covers all the interior points of \( B \). Thus the above implication is correct.
Now,
\[ x \in S \ominus \partial B \]
\[ \equiv (\partial B)_x \subseteq S \]
\[ \equiv B_x \subseteq S, \quad \text{provided } S \text{ is simply connected} \]
\[ \equiv x \in S \ominus B. \]

Thus the theorem is proved.

In order to improve the computational efficiency in decomposition, we have proposed to limit the set \( S \) to be a simply connected set. But that does not prevent us altogether from handling a multiply connected \( S \). In that case, we have already shown through Eq. (4a), that a hole in \( S \) can be treated separately and the intermediary results finally can be combined together by the set difference operation. But a more important question that crops up at this point is: "Is it possible to increase the computational efficiency further by limiting to a more restricted class of sets \( S \)?" The answer is, "Yes, indeed."

Let us now consider \( S \) to be restricted to a convex set. We can then show a considerable improvement in computational efficiency. The results of this investigation are presented through Theorems 2 to 5.

**Theorem 2** (theorem on convex decomposition by line segment). If \( S \) is a convex set and \( L \) is a line segment whose end points are \( b_s \) and \( b_e \), then
\[ S \ominus L = S \ominus \{b_s, b_e\}. \]

**Proof.** Let \( L = \{b_s, b_1, \ldots, b_i, \ldots, b_e\} \), where \( b_i \) denotes any intermediate point on the line segment. Any \( b_i \) can be expressed as
\[ b_i = r \cdot b_s + (1 - r) \cdot b_e, \quad \text{where } 0 < r < 1 \quad (a) \]

From the definition of the decomposition,
\[ S \ominus L = \left[ (S - b_s) \cap (S - b_e) \right] \cap \left[ (S - b_1) \cap \cdots \cap (S - b_i) \cdots \right]. \]

Theorem 2 will be proved if we can show that any \( t \in [(S - b_s) \cap (S - b_e)] \) is also an element of \( (S - b_i) \).

Let us express \( t \) as follows:
\[ t = s_j - b_s = s_k - b_e, \quad \text{where } s_j, s_k \in S. \]
(The representation "s_j - b_i" means "s_j + (-b_i)" etc.) Therefore,

\[ b_e - b_i = s_k - s_j. \]  

(\text{b})

Let us assume that, \( t = x - b_i \), where \( x \) is any arbitrary vector. If we can now show that \( x \in S \), then \( (x - b_i) \), i.e., \( t \) becomes an element of \((S - b_i)\), and that will complete the proof of the theorem.

Since,

\[ t = x - b_i = s_k - b_e, \]

therefore,

\[ x = r \cdot s_j + (1 - r) \cdot s_k \]  

(\text{using Eq. (a) and Eq. (b)}).

Since \( S \) is a convex set and \( s_j, s_k \in S \), therefore, \( r \cdot s_j + (1 - r) \cdot s_k \) must be an element of \( S \), i.e., \( x \in S \). Thus the theorem is proved.

The importance of the theorem may be felt better if we restate it as follows.

\textbf{THEOREM 2a.} If \( S \) is a convex set, then the decomposition of \( S \) by two points is equal to the decomposition of \( S \) by the entire line segment joining these two points.

Theorem 2 serves as a basis for obtaining some more interesting results. Consider the following theorem.

\textbf{THEOREM 3 (theorem on reduction to convex hull).} If \( S \) is a convex set, then

\[ S \oplus B = S \oplus \text{conv}(B), \]

where \( \text{conv}(B) \) denotes the convex hull of \( B \).

\textbf{Proof.} Let \( B \) be a set of points in \( R^d \) space denoted by

\[ B = \{ b_i; \, i = 1, 2, \ldots, n \}. \]

The convex hull \( \text{conv}(B) \) is the set of all points \( p \) which can be written in the form (known as convex combination form):

\[ p = \alpha_1 b_1 + \alpha_2 b_2 + \cdots + \alpha_n b_n, \]

where,

\[ \alpha_i \in \mathbb{R}, \quad \alpha_i \geq 0, \quad \alpha_1 + \alpha_2 + \cdots + \alpha_n = 1. \]

Any point \( p \) of the convex hull can also be rewritten in the form

\[ p = \alpha_1 b_1 + (1 - \alpha_1) p', \]
where,
\[ p' = \frac{\alpha_2}{(1 - \alpha_1)} b_2 + \frac{\alpha_3}{(1 - \alpha_1)} b_3 + \cdots + \frac{\alpha_n}{(1 - \alpha_1)} b_n. \]

According to the above representation of \( p \), it is an intermediate point on the line segment joining \( b_1 \) and \( p' \). And Theorem 2 says, every such intermediate point \( p \) will be automatically included within the decomposition \( S \ominus \{ b_1, p' \} \).

The point \( p' \) can likewise be expressed in terms of \( b_2 \) and \( p'' \), etc., and by proceeding in this manner we finally get that any point \( p \in \text{conv}(B) \) becomes automatically included in the decomposition \( S \ominus \{ b_1, b_2, \ldots, b_n \} \), i.e., in \( S \ominus B \). Thus the theorem is proved.

The implication of this theorem is significant. The theorem states that if \( S \) is convex, then independent of the type of \( B \), the decomposition of \( S \) by \( B \) can be carried out by an algorithm for the convex–convex decomposition only (provided an algorithm to compute the convex hull of a set is available).

In fact, it is possible to simplify our problem further. Note the following theorem.

**Theorem 4** (theorem on reduction to vertices). If \( S \) and \( C \) are both convex sets, then
\[ S \ominus C = S \ominus \text{vert}(C), \]
where \( \text{vert}(C) \) denotes the set of vertices of \( C \).

**Proof.** The proof turns out to be easy enough if we use the following definition of \( \text{vert}(C) \): the set \( \text{vert}(C) \) is the smallest subset of a convex set \( C \) having the property that \( \text{conv}(\text{vert}(C)) = C \).

Therefore, if
\[ \text{vert}(C) = \{ v_i : i = 1, 2, \ldots, m \}, \]
then any point \( c \in C \) can be expressed as
\[ c = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_m v_m \quad (\alpha_i \in \mathbb{R}, \alpha_i \geq 0, \alpha_1 + \cdots + \alpha_m = 1). \]

The rest of the proof then follows from the proof of Theorem 3.

For the purpose of concise representation, we may combine Theorems 3 and 4 by introducing the notion of the set of extreme points of a set \( B \), usually denoted by \( \text{ext}(B) \). Mathematically, \( \text{ext}(B) \) is the set of vertices of \( \text{conv}(B) \). See Fig. 8 in which \( \text{ext}(B) \) is shown in boldface.

The proposed combined theorem may be stated as follows.

**Theorem 5** (theorem on reduction to extreme points). If \( S \) is a convex set, then
\[ S \ominus B = S \ominus \text{ext}(B), \]
where \( \text{ext}(B) \) denotes the set of extreme points of \( B \).

An idea of the reduction in size from the set \( B \) to the set \( \text{ext}(B) \) may be obtained from the typical example depicted in Fig. 8. The improvement in the computational cost, in the average case, can also be easily guessed.
4.2. Decomposition Algorithms in 2D

All the results we have proved so far are valid for any decomposition in $R^d$ space, $d$ being equal to or greater than 2. However, we shall now concentrate in devising decomposition algorithms in 2-dimension. That is because our primary aim is to solve the polygon containment problem and not the decomposition problem in the general $d$-dimensional space.

The techniques and complexities of the algorithms depend directly on the types of $S$ and $B$ polygons. For the convenience of the readers, we write down the different combinations of $S$ and $B$ that are considered in this paper.

1. Simply connected polygons.

A1. Convex–Convex. Here both $S$ and $B$ are convex polygons. Our approach is different from the conventional ones and our technique is discussed in detail.

A2. Convex–Simple. $S$ is a convex polygon and $B$ is a simple polygon. In that case Theorem 3 suggests that first we have to determine the convex hull of $B$ and then algorithm A1 can be used.

A3. Simple–Convex. $S$ is a simple polygon and $B$ is a convex polygon. In that case an algorithm can be devised by making use of the formula,

$$S = \text{conv}(S) \setminus \text{(holes)}$$

$$= S_1 \setminus H.$$

The idea is to represent a simple polygon as a convex polygon minus some holes. We can simplify the above expression by using the result of Eq. (4a),

$$S \ominus B = (S_1 \ominus B) \cap (H \ominus \bar{B}) = (S_1 \ominus B) \setminus (H \ominus \bar{B}).$$

Since both $S_1$ and $B$ are convex, the $(S_1 \ominus B)$ polygon can be obtained by using the algorithm A1. The polygon $(H \ominus \bar{B})$ can be obtained using any of the standard algorithms given in Ghosh, Guibas et al., Schwartz, etc. [3, 5, 14].

In this paper, however, we take another approach in the line of the algorithm A1. This algorithm together with algorithm A1 finally pave the way for a unified algorithmic approach to solve all decomposition problems, i.e., the polygon containment problems in 2-dimension.
A4. Simple–Simple. Here both S and B are simple polygons. The algorithm we present is derived from the unified approach.

A5. Smooth–Smooth. Instead of polygonal boundaries, we consider both S and B having smooth boundary curves. This algorithm is briefly presented, primarily to demonstrate the applicability of the unified approach beyond polygonal boundaries.

2. Multiply connected polygons. No decomposition algorithm for multiply connected polygons (i.e., polygons that have holes in them) is presented in this paper. However, we must mention that some of the theorems and results we proved so far can be effectively used for that purpose. For example, Theorem 3 states that the hole in B polygon can be ignored during decomposition if S is convex. Similarly, Eq. (4) can be used if S polygon has a hole in it. (Unfortunately, relationships similar to Eq. (4) do not hold true if B has holes, since

\[ S \ominus (B \cap H^c) \supseteq (S \ominus B_1) \cup (S \ominus H^c). \]

Note that the equality does not hold true always).

4.2.1. Convex–Convex Case: Algorithm A1

Let S be a convex \( N_S \)-gon and B be a convex \( N_B \)-gon. Our task is to determine \( S \ominus B \) in this case. The definition of decomposition in Eq. (2) and Theorem 4 together suggest that our task is equivalent to solving the following problem.

**Problem A1** (Minkowski decomposition for convex polygons). Given two convex polygons S and B, translate and place S polygon at every vertex of \( B^c \) (\( B \) is the symmetrical set of B) and determine the common intersection among all the instances of S.

The algorithms for intersection of two convex polygons are well known [11]. Our task is to find out the common intersection of \( N_B \) number of convex polygons; each are identical (only translated by different amounts) having \( N_S \) number of edges. Preparata et al. [11] stated that the common intersection of \( K \) number of convex \( N \)-gons can be found in \( \mathcal{O}(NK \log K) \) time. In our case, it will, therefore take \( \mathcal{O}(N_S N_B \log N_B) \) time.

But could we not expect a better result than that, since in our case all the \( N_B \) polygons are exactly identical? Problem A1 can indeed be solved in linear time, i.e. i.e., in time \( \mathcal{O}(N_S + N_B) \). There are several ways of achieving this bound by following conventional techniques [1]. However, we present here a somewhat different method. This new technique finally enables us to arrive at a unified method of decomposition in 2-dimension.

4.2.1.1. Two notations. The algorithm we are going to propose needs the following two notions.

(i) Oriented boundary curve. If the boundary curve encloses a region, we consider this region as a positive area and assign counterclockwise orientation to the boundary curve. On the other hand, if the boundary curve encloses a hole, we consider this as a void (or negative) area and give clockwise orientation to the boundary curve. The basic idea is, if we move along the boundary curve in the
direction of its orientation, the region on its "left" is a positive region and on its "right" is void. In our presentation, hereafter, we assume that all the edges and line segments are directed without explicitly mentioning this (refer to Fig. 9). A region and its oriented boundary representation are shown in Fig. 9a. On the other hand, in Fig. 9b we take an oriented boundary curve and show the region it represents. Note that the triangular portion at the vertex $p_1$ is discarded at the time of region representation, since its orientation specifies it as a void region. In reality a void region cannot exist in isolation; it must be always surrounded by a positive region to make it a hole.

(ii) Anti-supporting line of a convex figure. A convex figure $F$ can have two supporting lines in any given direction. (A supporting line $l$ of a convex figure $F$ passes through a point(s) on the boundary of $F$ such that the interior of $F$ lies entirely on one side of $l$.) If we assign an orientation to the supporting lines, then one of the two, say $l_1$, will follow the orientation of the boundary curve of $F$ while the other one, say $l_2$, will be exactly opposite to it (Fig. 10). We call the second one an anti-supporting line while the first one can be called simply a supporting line. With respect to the supporting line, the figure $F$ lies on its left, while $F$ lies on the right of its anti-supporting line. The intersections $F \cap l_1$ and $F \cap l_2$ can be called supporting point and anti-supporting point, respectively.

4.2.1.2. Basic idea of the algorithm. We can now come back to our algorithm. The situation is as follows. The $B$ polygon has $N_B$ number of vertices. The $S$ polygon is placed on every one of those $N_B$ vertices. Therefore, corresponding to every edge, say $e$, of $S$, there will be $N_B$ number of line segments—all of which are equal, parallel, and similarly oriented to the edge $e$, only translated differently. We

---

**FIG. 9.** Oriented boundary curves.

**FIG. 10.** Supporting and anti-supporting lines of convex figure.
A SOLUTION OF POLYGON CONTAINMENT

Fig. 11. Decomposition of a convex polygon by a convex polygon.

may consider all these line segments as one bunch corresponding to the edge $e$ of $S$. We call it the $e$-bunch. For example, refer to the typical example depicted in Fig. 11a. The corresponding situation when $S$ is placed on the vertices of $\bar{B}$ is shown in Fig. 11b. The $e_1$-bunch corresponding to the edge $e_1$ of $S$ consists of $N_\bar{B}$ line segments which are shown by bold lines. Clearly, if $S$ has $N_S$ edges, there will be $N_S$ such $e$-bunches corresponding to every of its edges.

A crucial observation. The line segments in an $e$-bunch could be obtained by placing $\bar{B}$ to the two end points of the edge $e$ and then joining the corresponding vertices of the two instances of $\bar{B}$. This is demonstrated separately in Fig. 11c for the $e_1$-bunch. This observation will be used later in our algorithm.

Therefore, in order to determine the common intersection of all the instances of $S$, our task boils down to:

1. Select from each of the $e$-bunch, the left-most line segment. We call it the Le-seg. In general a Le-seg is an oriented line segment of length $e$. But since $S$ is convex in this case, we may safely assume that each Le-seg here is an infinite line and thereby describes a half plane. The final outcome of this selection step will be $N_S$ number of Le-segs corresponding to all the edges of $S$. In Fig. 11d they are shown by bold lines.

2. Find out the intersections among those Le-segs. In this particular case, it will be equivalent to determining the common intersection of the $N_S$ half planes they describe.
4.2.1.3. First step of the algorithm: Selection step. Selecting the $Le$-seg from an $e$-bunch is not difficult. The $Le$-seg must be the anti-supporting line of the $B$ polygon (or a translated instance of $B$ polygon as shown in Fig. 11c).

The proof of the above statement follows from the arguments given below. If a line segment from the $e$-bunch is an anti-supporting line of $B$, then all the points of $B$ will be on its right by definition. Therefore all other line segments of the $e$-bunch will be also on its right (refer to our observation in a previous paragraph). That means, that line segment must be the left-most line segment of the lot, i.e., the $Le$-seg. On the other hand, if the $Le$-seg is not the anti-supporting line, then there must be at least one vertex of $B$ that will be on the left of the $Le$-seg. Therefore, there will be at least one line segment in the $e$-bunch that will be on the left of the $Le$-seg. This is clearly a contradiction. Therefore, the $Le$-seg must be anti-supporting line.

Step 1 of the algorithm can, therefore, be accomplished in two substeps:

1a. Identify, for each edge $e$ of $S$, the corresponding anti-supporting point at $B$.

1b. Translate $B$ to the two vertices of $S$ corresponding to the edge $e$ and join the two instances of the anti-supporting point at $B$ by a line segment. According to our above arguments, this line segment is the desired $Le$-seg. For more clarification, refer to Fig. 11d.

Repeat the process for every edge of $S$.

We can now find out the time bound of an algorithm for executing step 1. If the edges of $S$ and $B$ are given in slope-sorted order, then the time bound for step 1a is $O(N_S + N_B)$, since it is equivalent of merging two ordered set of elements. Step 1b needs $O(N_S)$ time. Therefore, step 1 takes $O(N_S + N_S)$ time.

4.2.1.4. Second step of the algorithm: Intersection step. Once we obtain these $N_S$ $Le$-segs, the task of step 2, in this particular case, is to find out the common intersections of the $N_S$ half planes described by these lines. This can be done in $O(N_S \log N_S)$ time [11]. However, we can obtain a faster algorithm since these line segments are parallel to the edges of $S$ and are known, therefore, in slope-sorted order. This order can be exploited to get an $O(N_S)$ algorithm.

Baker et al. [1] indicated an algorithm that indeed takes $O(N_S)$ time. We present here a different algorithm of the same complexity with our ultimate intention of unifying all decomposition algorithms.

The crux of our algorithm lies in describing these $Le$-segs through a different representation scheme which can be termed as a decomposition boundary tracing scheme. (The decomposition boundary tracing of a decomposition will be denoted by the letter $D$.) The scheme is, essentially, traversing or tracing the $Le$-segs in a continuous manner.

The new representation of the $Le$-segs through tracing is done as follows. Start with any of the $Le$-segs, say $Le_1$-seg corresponding to the edge $e_1$ of $S$ (see Fig. 12 as well as Fig. 11d), and move along it following its orientation. According to our previous argument we can assume that the end points of $Le_1$-seg lie on the corresponding anti-supporting point at $B$ (or at its translate), say $u_1$. Try now to trace the next $Le$-seg, say $Le_2$-seg, and let us assume that its corresponding
EXAMPLE I

EXAMPLE II

FIG. 12. Two typical decomposition boundary tracings. (A line segment $B$ must be considered as a 2D rectangle of zero width.)

anti-supporting point at $\hat{B}$ is $v_2$. If $v_1$ and $v_2$ do not happen to be the same point of $\hat{B}$, then first move along the edge(s) of $\hat{B}$ to reach from $v_1$ to $v_2$, and then move along $Le_2$-seg. Continue this process until all the $Le$-segs are traced in this way. Obviously we are bound to reach $Le_1$-seg where we started. Two typical decomposition boundary tracings are shown in Fig 12. In the figure, the edges in the tracings corresponding to the edges of $\hat{B}$ are marked as $r'$. These $r'$-edges are keeping the continuity in tracing the $Le$-segs.
Surely we have to resolve a point before proceeding further. We find that, in any tracing $D$, some edges of $\hat{B}$, i.e., the $r'$-edges are also included. That happens when we move from one of the anti-supporting points at $\hat{B}$ to another anti-supporting point along the edges of $\hat{B}$. Now in which way should we have to move? Should we follow the orientation of $\hat{B}$ or move in the opposite direction? For example, consider the first example of Fig. 12. In moving from $v_1$ to $v_2$, we may either move along $v_1v_2$ by following the orientation of $\hat{B}$, or we may go along $v_1v_4v_3v_2$, in the opposite direction.

In the convex–convex case, it is a matter of choice, since both will result into the same $S \ominus B$ as shown in Fig. 12c. Without explaining at this point we state that the easy rule, as long as $\hat{B}$ is convex, is to move in the opposite direction.

The easy rule, however, may introduce inefficiency: there may be too many self-intersections in the decomposition boundary tracing at times; also the total number of edges in the tracing might become unnecessarily large as demonstrated in Fig. 12.

The exact rule is as follows. Check whether the direction of turn of one Le-seg to the next Le-seg in the tracing is clockwise or counterclockwise. Choose that path along the edges of $\hat{B}$ where you can follow the same orientation. An explanation behind the formation of these rules is clarified in Section 5.1. Until we reach that section, we shall follow the easy rule.

A few immediate observations about decomposition boundary tracing could be noted now:

(a) A decomposition boundary tracing $D$ is a closed, oriented, nonsimple (self-crossing) polygon.
(b) The $D$ polygon is essentially a travel along the Le-segs in a continuous manner. In that respect $D$ may be regarded as a new representation of the Le-segs as one single polygon.
(c) The positive region, say $T$, enclosed by the polygon $D$ is the required common intersection of different instances of $S$ placed on $\hat{B}$, i.e., the intended decomposition $S \ominus B$.

This particular observation is crucial for our future course and therefore may be clarified a little further. We knew that the positive region $T$ (i.e., the common region left of all the Le-segs) enclosed by the Le-segs is the required $S \ominus B$. In the $D$ polygon, in addition to the Le-segs, we have also added some edges of the $\hat{B}$ polygon which are denoted as $r'$-edges. Since all the Le-segs are the anti-supporting lines of $\hat{B}$, the polygon $\hat{B}$ (or its translates) will be always on the right of the Le-segs. Therefore, no edge of $\hat{B}$ can go to the left of the Le-segs, and thereby the positive region enclosed by the Le-segs remains unaltered even after including a few edges of the $\hat{B}$ polygon.

(d) For convex–convex decomposition, only the Le-segs of $D$ corresponding to the edges of $S$ contribute to the boundary $\partial T$ of the positive region $T$. The total number of such Le-segs is $N_S$. In computing $T$, the $r'$-edges of $D$ appear to be redundant.

(e) It can be argued that, in the case of convex–convex decomposition, the polygon $D$ consists of $(N_S + N_B)$ number of edges corresponding to every edge of $S$ and $B$, provided we follow the exact rule for traversing the edges of $\hat{B}$. 
According to our formulation, now the task of step 2 is to find out the self-intersections of the edges of $D$, i.e., the intersections of the Le-segs. That will define the boundary $\partial T$ of $T$. We observe that two kinds of situations may happen.

**Case I.** When all the Le-segs of $D$ are present in the boundary $\partial T$ (Fig. 12c, Example I), then step 2 of our algorithm can be accomplished in time $O(N_S)$. Because the task is merely to determine the intersection points between the adjacent Le-segs which are given in slope-sorted order.

**Case II.** When only $K$ out of the $N_S$ Le-segs ($K < N_S$) are present in the boundary $\partial T$ (Fig. 12b, Example II), then first we have to identify the $(N_S - K)$ Le-segs that are absent or redundant. To check the redundancy, we may proceed as follows.

Let $l_{i-1}$, $l_i$, and $l_{i+1}$ denote three consecutive Le-segs given in slope-sorted order. Let the intersection points of the lines $l_{i-1}$ and $l_i$ be $a$, and that of $l_i$ and $l_{i+1}$ be $b$. The line $l_i$ will be redundant if the direction from $a$ to $b$ does not follow (i.e., is opposite to) the direction of the line $l_i$. See Fig. 13.

The logic is easy to see. If the lines $a$ and $l_i$ are oppositely directed, then the intersection point $c$ of $l_{i-1}$ and $l_{i+1}$ must be on the left of $l_i$. Because if $c$ is on the right of $l_i$, then the slope of $l_{i+1}$ (which passes through the points $b$ and $c$) must be less than the slope of $l_i$, and that is a contradiction. Now if the lines $l_{i-1}$, $l_i$, and $l_{i+1}$ describe respectively the half planes $\text{HP}_{i-1}$, $\text{HP}_i$, and $\text{HP}_{i+1}$, then the fact that the point $c$ is on the left of $l_i$ means $(\text{HP}_{i-1} \cap \text{HP}_{i+1}) \subseteq \text{HP}_i$. Therefore, $\text{HP}_i$ (i.e., in turn $l_i$) becomes redundant since the common intersection is nothing but 

$$
\cdots \cap \text{HP}_{i-1} \cap \text{HP}_i \cap \text{HP}_{i+1} \cdots
$$

Hence by a constant time testing a redundant Le-seg can be identified and removed. Therefore, the removal of all the redundant Le-segs can be done in $O(N_S)$ time.

Once we identify only the $K$ number of Le-segs that all are present in the boundary $\partial T$, their self-intersections can be determined in time $O(K)$ by finding out the intersection points of the adjacent Le-segs.

Therefore, step 2 can be accomplished in $O(N_S)$ time. So the whole algorithm takes $O(N_S + N_B)$ time, which is linear.

The technique of this algorithm can be summarized in the following form:

1. Form the decomposition boundary tracing $D$.
2. Determine the positive region enclosed by $D$.

We can term this technique as the *decomposition boundary tracing* technique, or simply the $D$-tracing technique.
4.2.2. Simple-Convex Case: Algorithm A3

Let $S$ be a simple $N_S$-gon. $B$ remains a convex $N_B$-gon as before. Our task is to determine $S \ominus B$.

4.2.2.1. Relative supporting line. The first step involves characterization of convex and simple polygons in terms of a supporting line. Generally a convex polygon is distinguished from a simple polygon in the following way. For a convex polygon all its vertices are convex, whereas for a simple polygon only some of its vertices are convex, the others are reflex or concave. A vertex is a concave vertex if the internal angle at the vertex is more than $180^\circ$; otherwise it is convex.

We observe that every vertex of a convex polygon can have at least one supporting (or anti-supporting) line, whereas a vertex, particularly the concave vertex, of a simple polygon may not have any. Mathematicians, to remedy this situation, extended the notion of a supporting line, known as a relative supporting line. A line passing through a point $p$ on the boundary of the figure $F$ can be considered as a relative supporting line if it is a supporting line not compared with the whole figure $F$, but with $F$ in some neighborhood of $p$. We shall use this general notion of relative supporting line to tackle the problems with nonconvex figures. (For convenience, we shall hereafter write simply supporting (or anti-supporting) line/point to mean relative.)

4.2.2.2. A subproblem: Decomposition by a line segment. As a first step toward solving the main problem, we investigate a situation where $S$ is a simple polygon and $B$ is a line segment such that it is a relative supporting line at some concave vertex $v$ of $S$. Such a situation is depicted in Fig. 14a. Our task is to determine $S \ominus B$.

Since $S$ is not a convex polygon, we have to place $S$ at every point of $\hat{B}$ and then compute the common intersection as shown in Fig. 14a. Unlike the convex case, placing $S$ only at the two end points of the line segment $\hat{B}$ would result into a region which is a superset of $S \ominus B$ (Fig. 14b).

We now try to compute $S \ominus B$ through our decomposition boundary tracing technique. But the fundamental question is: Will that same technique developed for the convex-convex case work? It needs no explanation to state that for determining the common intersection of the instances of $S$, we have to identify, in every direction, the left-most parts of those instances. Therefore, we have to examine whether the $D$-tracing technique ensures choosing the left-most parts always. To simplify the problem, we examine the situation in two parts.

Part 1. For the Convex parts of $S$. As long as $B$ is convex (as is the case here) and we are concerned with the convex parts of $S$ (i.e., on the edges of $S$ incident only on the convex vertices), the principle of formation of $D$ does not need any modification. Because the task remains exactly the same as before—from each of the $e$-bunch corresponding to the convex edge $e$ of $S$, choose the $Le$-seg.

Part 2. For the Nonconvex parts of $S$. But the problem arises for the nonconvex parts of $S$, i.e., for the edges that are incident to its concave vertices. The problem is due to relative supporting line. Note that any convex edge $e$ of $S$ is a global supporting line of $S$. Therefore, the whole of the common intersected region (i.e., the region of our interest) will lie at the left of the corresponding $e$-bunch, and the $Le$-seg of that $e$-bunch will be the “sole” contributing edge of the boundary of the
intersected region in that particular direction. On the other hand, any nonconvex edge \( e' \) of \( S \) is a relative supporting line of \( S \) within a small neighborhood. Therefore, unlike the convex case, only a part of the intersected region, and not the whole, is supposed to lie at the left of the corresponding \( e' \)-bunch. And clearly the corresponding \( Le' \)-seg "cannot" be the sole contributing edge in that direction. Fig. 14a demonstrates this situation, where we see that not only the \( Le' \)-segs, but all the segments in each of the two \( e' \)-bunches contribute to the intersected boundary at the vicinity of the concave vertex \( u \). We again may look into this problem in two parts.

**Part A. When \( B \) is relative supporting line at \( v \).** Let us assume that the line segment \( B \) in our problem is a relative supporting line at the concave vertex \( v \). It is a typical situation that we have chosen in our example in Fig. 14a. Certainly, in this case too, the \( Le' \)-segs, from each of the two bunches corresponding to the two nonconvex edges incident on \( v \), contribute to the boundary of the common intersection. But in addition to these two segments, "some parts" of the other segments of the two bunches also contribute. But can we characterize those "some parts"? Since \( S \) is placed at every point of \( \bar{B} \), the vertex \( v \) is the only contributing point at any instant. And the sum total contribution is the locus or instances of \( v \) as \( S \) is placed continuously at every point of \( \bar{B} \). And that locus is nothing but the line segment \( \bar{B} \) itself. This situation is clearly demonstrated in Fig. 14a.

Now if we follow the \( D \)-tracing technique, we shall also arrive at the same result. The resulting \( D \) for our example and the enclosed positive region \( T \) (which conforms to \( S \Theta B \)) is shown in Fig. 15.
Note. One significant difference between the nature of the tracings in the previous convex-convex case and this simple-convex case must be noted. In the former only the edges of $S$ were effectively contributing to the boundary of the $T$ region, while in the latter we find an edge, say $ab$ in Fig. 15, which appears to be an edge of $B$, is also contributing. The edge $ab$ is, in fact, the instances or the locus of the concave vertex $v$.

Part B. When $B$ is not a relative supporting line at $v$. If the line segment $B$ is not a supporting line at the concave vertex, the situation turns out to be much simpler. The $Le'$-seg for a nonconvex edge $e'$ of $S$ becomes the "sole" contributing factor in that particular direction, and the $D$-tracing technique does not need any modification. A typical such situation is shown in Fig. 16.
From the above discussion we can, therefore, conclude that the $D$-tracing technique, described in Section 4.2.1.4, is applicable even if $S$ is a simple polygon and $B$ is a line segment.

4.2.2.3. The generalization: Decomposition by a convex polygon $B$. Let us now come back to our original problem. We claim, from the study of the above subproblem, that the $D$-tracing technique does not need any modification even if $B$ is a convex polygon instead of a line segment. In the first place, recall the fact that placing $S$ on the boundary edges $\partial B$ of $B$ alone is sufficient, according to Theorem 1. We can now argue in the following way. Since $B$ is a convex polygon, the technique of tracing the part of $D$ corresponding to the convex parts (say, corresponding to the edges $e_1$, $e_2$, etc. in Fig. 17) remains exactly similar to that of the convex–convex case. Corresponding to the nonconvex parts (i.e., a concave vertex $v$ and its incident edges, say, $e_3$, $e_4$ in Fig. 17), the boundary of the common intersection contains the $Le'$-segs from the nonconvex $e'$-bunches (i.e., $Le_3$-seg, $Le_4$-seg in Fig. 17), plus it contains the left-most part of the locus of the vertex $v$ as it moves along $\partial B$. The vertex $v$ and thereby its locus comes in between those two $Le'$-segs. That essentially means that the edge(s) of $\tilde{B}$ whose slope(s) in the reverse direction falls in between the slopes of $e_3$ and $e_4$ must also be included in the boundary of the common intersection, in addition to the $Le_3$-seg and $Le_4$-seg. Our tracing technique also does exactly the same thing. (A different proof of this part is given in Ghosh [3], by exploiting the fact that a nonconvex part of a polygon can be described through the convex hole $H$. Therefore, its contribution to the boundary of the common intersection can be determined from the boundary of the sum $H \oplus \tilde{B}$. To keep the paper self-complete the proof is not included since it uses the concept of boundary tracing for Minkowski addition of polygons.) Once we form the $D$ polygon, we can determine the positive region $T$ enclosed by $D$. $T$ is the required $S \ominus B$.

4.2.2.4. Complexity analysis for the Algorithm A3. The simple-convex algorithm, like the convex–convex case, has also two basic steps:

Step 1. Formation of $D$ which is a self-crossing polygon.

Step 2. Determination of the positive region $T$ enclosed by $D$. 

![Fig. 17. Decomposition of a simple polygon $S$ by a convex polygon $B$.](image-url)
Let us first find out the maximum number of edges in $D$. Let that number be $N$.

$$N = \text{number of edges contributed by } S + \text{number of edges contributed by } \tilde{B}$$

$$= N_S + (\text{number of vertices of } S \times \text{max numbers of edges of } \tilde{B} \text{ contributing at any vertex})$$

$$= N_S + N_S \times (N_B - 1)$$

$$= N_S \cdot N_B.$$

Next we have to find out the maximum number of self-intersections between these $N$ edges. To do this, we consider these $N$ edges as a collection of $N_S$ number of sickles: each of these sickles consists of $N_B$ number of line segments, of which $(N_B - 1)$ segments are contributed by the edges of the $\tilde{B}$ polygon (shown dotted in Fig. 18) and the rest, say $l$, is a Le-seg contributed by one of the edges of $S$ (shown boldface in Fig. 18).

Note that each such sickle is a convex chain, since $\tilde{B}$ is convex and the line segment $l$ is an anti-supporting line of $\tilde{B}$. Since we have $N_S$ such sickles, therefore, the maximum number of self-intersections among them could be of the order of $N_S^2$.

If the edges of $S$ and $B$ are given in slope-sorted order, step 1 can be carried out in $O(N)$, i.e., in $O(N_S \cdot N_B)$ time. Step 2 can be executed by using the method suggested by Bently and Ottman [11] for reporting all $K$ intersections of an arbitrary set of $N$ line segments that takes $O((N + K)\log N)$ time. (More recently, Chazelle [11] reported an $O(N \log^2 N / \log \log N + K)$ algorithm for the same problem. Therefore, step 2 takes $O((N_S^2 + N_S N_B)\log N_S N_B)$ time in our case (or, $O(N_S N_B \log^2 N_S N_B / \log \log N_S N_B + N_S^2)$, if one chooses to use Chazelle’s method).

(Note that Algorithm A3 could be made more efficient by not forming the complete $D$-polygon explicitly, but computing only those parts of $D$ that may contribute to the positive region $T$. For example, in case of the convex vertices of $S$, the edges of $\tilde{B}$ need not be traced explicitly as is shown in A1. However, we shall not concern ourselves with those finer details in this paper.)

5. A UNIFIED APPROACH AND ITS DEMONSTRATIONS

5.1. The Approach

The previous discussions bring out the fact that, in determining $S \Theta B$ the major task is to form the decomposition boundary tracing $D$. Once we are able to form the $D$ polygon, we can use any standard algorithm to compute the boundary of the positive region $T$ that is enclosed by $D$. $T$ is the required $S \Theta B$.

But does the $D$-tracing technique work for all kinds of polygons? If we can show that it indeed works, then we arrive at a unified approach to solve all 2D
A SOLUTION OF POLYGON CONTAINMENT

29

(a) A part of S polygon and rotating supporting lines
(b) Corresponding anti-supporting points at B
(c) Formation of the tracing D

FIG. 19. Analysis of the decomposition boundary tracing D.

decompositions. For that purpose, we must analyse first the steps we follow in the D-tracing technique.

First, any edge of a polygon can be characterized as the intersection of the polygon and the supporting line of the polygon parallel to that edge. Therefore, according to our notations, an edge of a polygon is the supporting points at the polygon with respect to the supporting line parallel to that edge. (According to our notion of oriented line, “parallel” means “similarly oriented” also.)

Now recall that a D polygon consists of Le-segs corresponding to each edge e of S, and r'-edges corresponding to the edges of B which keep continuity between two adjacent Le-segs. Any Le-seg, we know, is the edge e of S translated to its corresponding anti-supporting point at B (refer to Section 4.2.1.3). This is equivalent of vectorially adding the edge e to its corresponding anti-supporting point at B. In other words, this is equivalent of vectorially adding the supporting points at S to the anti-supporting point at B with respect to the supporting line parallel to the edge e of S.

Second, in forming D, we move along the edges of B to reach from the anti-supporting point for one edge of S to the anti-supporting point for the next edge of S. This step is equivalent to assuming that the supporting line parallel to one edge of S is continuously changing its direction (i.e., rotating continuously) to finally become parallel to the next edge of S, and at every stage the supporting point at S (it is, in fact, the vertex of S where the two edges meet) and the corresponding anti-supporting points at B are being vectorially added. (This continuous movement of the supporting line ensures continuity of the D polygon.) Figure 19 might clarify the idea better. In Fig. 19a we show how the supporting line is gradually rotating from the direction of the edge $e_1$ to the edge $e_2$. At all times the supporting point at S remains fixed at its vertex v, but the corresponding anti-supporting point at B (Fig. 19b) changes from c to a along the edge ca. For every direction of the supporting line, we vectorially add the supporting point at S to the corresponding anti-supporting point at B which finally forms an r'-edge (Fig. 19c).

(Our formulation of the exact rule for traversing the edges of the B polygon must be clear to the readers at this point. The easy rule had also worked since we finally discarded the negative regions enclosed by the D polygon.)
The above analysis could be summarized in the following form. The tracing $D$ is essentially the vector sum of the supporting points at $S$ and the corresponding anti-supporting points at $\bar{B}$ in every possible direction. But does the tracing $D$ thus formed ensure correct $S \oplus B$ in all situations? Yes, it does. Because for determining $S \oplus B$, we have to first identify, in every direction, the left-most parts of the instances of $S$ when placed on the boundary points of $\bar{B}$. The supporting point(s) at $S$ in any given direction is actually its boundary point(s) in that direction. The anti-supporting point(s) at $\bar{B}$ is the left-most point(s) of $\bar{B}$ in that direction. Therefore, vector addition of the two will surely give us the left-most part of the instances of $S$ in that given direction.

We can now abstract out our approach to decomposition in terms of two propositions stated below.

**PROPOSITION 1** (supporting line proposition of 2D decomposition). *Given two plane figures $S$ and $B$ in $R^2$, the decomposition boundary tracing $D$ of $S \oplus B$ is the set of all points $s + (-b)$ (i.e., vectorial addition of the points $s$ and $-b$), such that $s \in S$ and $-b \in \bar{B}$ are the supporting point(s) and corresponding anti-supporting point(s) respectively with respect to a supporting line in a given direction, and the direction of the supporting line varies continuously from 0 to 360° covering all directions in the plane.*

Note that varying the direction of a supporting line from 0 to 360° is equivalent to varying it continuously along the boundary of $S$. However, varying it in the latter way has certain computational advantages, and therefore, has been adopted by us.

**PROPOSITION 2** (proposition on decomposition boundary tracing). *In $R^2$, $S \oplus B$ is the positive region $T$ enclosed by the decomposition boundary tracing $D$ of $S \oplus B$. (

*Note. Complete algebraic proofs of these two propositions can be obtained by making use of Shephard's theorem on Minkowski addition [4, 3] and the result stated in Eq. (5).*

We consider these two propositions as the fundamental algorithmic concepts of 2D Minkowski decomposition, since they state the underlying principle behind all our algorithms for 2-dimensional decomposition. However, this does not mean that the efficient algorithms for all kinds of $S$ and $B$ will be exactly the same. To make an algorithm efficient, one must exploit the characteristics of the input figures. Therefore, the actual computational procedure for decomposition varies depending on the types of $S$ and $B$. We already demonstrated, for example, that in the case of the convex-convex decomposition, there is no need to form the decomposition boundary tracing explicitly, although the underlying principle remains the same. In the next section we present two more algorithms to establish this very point. Moreover, these two algorithms will bring our study on 2-dimensional decomposition to a certain level of completion.

### 5.2. Two More Algorithms

#### 5.2.1. Simple-Simple Case: Algorithm $A4$

Let $S$ and $B$ both be simple polygons as shown in Fig. 20a. the task is to compute $S \oplus B$ by making use of the previous two propositions.

In tracing the $D$ polygon we face no problem for the convex edges of $S$, where we can assume decomposition by $\text{conv}(B)$ instead by $B$ (see Theorem 3). But the same
assumption is not applicable for the nonconvex edges of $S$. Now, since $B$ is a simple polygon in this case, it may happen that for some nonconvex edge(s) of $S$, there may be more than one “disconnected” anti-supporting point(s) at $B$. See our example figure where, for the supporting line parallel to the edge $e_4$ of $S$, both the vertices $a$ and $d$ of $B$ are the corresponding anti-supporting points. Clearly, $a$ and $d$ are disconnected points. According to Proposition 1 if we add $\{e_4\} \oplus \{d, a\}$ to form $D$, the tracing $D$ becomes disconnected at this point. Although it can be proved that even such a disconnected tracing will eventually generate the correct $S \ominus B$, it is definitely preferable for various reasons to generate a completely connected $D$ polygon.
To achieve this aim, a number of remedies could be suggested. We suggest the following method. (It must be mentioned that our technique does not guarantee the best optimal algorithm.) The basic idea is to consider only one anti-supporting point at each time, to perform decomposition operation as many times as there are multiple anti-supporting points to determine the decomposed regions, and finally to take the intersections of all those intermediary decomposed regions to obtain the actual $S \ominus B$.

In our example figure (Fig. 20a), we have already stated that the edge $e_4$ has two anti-supporting points $a$ and $d$ at $\bar{B}$. Therefore, the decomposition operation has to be performed twice. The first time we assume that $e_4$ has only one anti-supporting point, say $a$, and determine the tracing $D_1$ (Fig. 20b) and subsequently the corresponding positive region $T_1$ (Fig. 20c). The second time we choose $d$ as the only anti-supporting point of $e_4$. The corresponding $D_2$ and $T_2$ are determined (shown in Figs. 20d and e, respectively). Finally we take the intersection of $T_1$ and $T_2$ to obtain the required $S \ominus B$.

The above procedure essentially means that we are assuming

$$B_1 = B \setminus d \quad \text{(consider } B \text{ without the vertex } d)$$

$$B_2 = B \setminus a \quad \text{(consider } B \text{ without the vertex } a)$$

And we are claiming that

$$S \ominus B = (S \ominus B_1) \cap (S \ominus B_2).$$

This is a valid claim, since for any general sets $S$, $B_1$, and $B_2$ in $\mathbb{R}^d$, the following relation holds:

$$S \ominus (B_1 \cup B_2) = (S \ominus B_1) \cap (S \ominus B_2).$$

And we have chosen $B_1$ and $B_2$ in such a way that

$$B = B_1 \cup B_2.$$

Note again that the algorithm stated above is only a conceptual model. In actual practice, it may not be necessary to compute all the $(S \ominus B_i)$ regions explicitly. It is possible to determine beforehand, by examining the nature of the multiple supporting points, whether some of the $(S \ominus B_i)$'s are proper subsets of the others. However, we avoid going into those details in this paper.

5.2.2. Smooth-Smooth Case: Algorithm A5

The task of Algorithm A5 is to compute $S \ominus B$ when both $S$ and $B$ have smooth boundary curves (Fig. 21).

Note that for a smooth boundary curve, the supporting line and the directed tangent line become synonymous. Therefore, the task of the formation of $D$ in this case reduces to the following sequence of operations. For every point $s$ of the boundary of $S$, find the direction of the tangent line and then determine the corresponding point $(-b)$ at the boundary of $\bar{B}$, where the tangent line is oppositely
directed; add \( s \) vectorially with \((-b)\), i.e., \( s + (-b) \). The method is depicted in Fig. 21b.

We may go further. It is often possible to obtain an algebraic solution of \( D \) in this case. Let the boundary curves of \( S \) and \( B \) be represented respectively in the parametric forms

\[
\partial B = \left[ B_x(u), B_y(u) \right],
\partial S = \left[ S_x(v), S_y(v) \right],
\]

where \( u \) and \( v \) are two scalar quantities and are closed intervals on the \( u \)-axis and \( v \)-axis, respectively. We assume that both \( B \) and \( S \) are oriented in the sense corresponding to an increase in the parameters \( u \) and \( v \), respectively. Clearly,

\[
\partial \tilde{B} = \left[ \tilde{B}_x(u), \tilde{B}_y(u) \right] = \left[ -B_x(u), -B_y(u) \right].
\]

Let \( s \in S \) for some \( v \), i.e., \( s = \partial S(v) \). To obtain the corresponding anti-tangent point \((-b)\) at \( \tilde{B} \), we have to solve the following equation for \( u \):

\[
\begin{bmatrix}
\frac{\delta \tilde{B}_x}{\delta u} \\
\frac{\delta \tilde{B}_y}{\delta u}
\end{bmatrix} = \begin{bmatrix}
\frac{\delta S_x}{\delta v} \\
\frac{\delta S_y}{\delta v}
\end{bmatrix}
\]

Let the solution for \( u \) be \( u = f(v) \), where "\( f \)" denotes some function. Therefore, the analytical expression for the \( D \)-curve is

\[
D = \left[ S_x(v) - B_x(f(v)), S_y(v) - B_y(f(v)) \right].
\]

To obtain more details in this regard readers may refer to Ghosh [2].

6. A FEW CONCLUDING REMARKS

In this paper we have introduced three important concepts:

1. We established the direct relationships between the Minkowski operations and the containment, spatial planning, and other related problems.

2. We reformulated Minkowski decomposition (as well as addition) problem, which is intrinsically a geometric problem, as a set theoretic problem and then used
the set theoretic tools to reduce the computational complexity of the problem. This set theoretic reformulation also allows us to go beyond convexity as well as beyond 2-dimensional.

3. We introduced the notion of decomposition boundary tracing that eventually helped us in proposing a unified theory for Minkowski decomposition algorithms in $R^2$.

Over and above those concepts, the paper also suggested (indirectly) a number of new directions that seem promising for future research. We may explicitly list here a few of them:

(a) The way we developed the theoretical framework and the 2D algorithms clearly allows us to go beyond 2-dimensional. First, the set theoretic results (Section 4.1) that simplify the geometric problem are applicable for general sets in $R^d$. Second, Propositions 1 and 2, which are the basis for all the 2D algorithms can be extended to higher dimensions simply by generalizing the concept of an oriented supporting line. The general notion is that of a supporting hyperplane with an orientation such that the direction of the outer normal of the supporting hyperplane matches that of the corresponding surface of the object.

We face one problem here. The problem is in choosing the order of traversal of the surfaces of $S$ and $\hat{B}$. In forming the $D$-polygon we traversed the edges of $S$ and $\hat{B}$ in such a way that at every stage of augmentation, the tracing is connected. The choice of the traversal order was easy for us, since the vertices and the edges of a planar polygon are intrinsically ordered. On the other hand, the vertices, edges, and faces of a 3-dimensional polyhedron are not ordered accordingly. The problem could be overcome by defining some order of traversal to maintain connectedness of the tracing at each stage. Obviously, that will lead to more complex data structures and algorithms.

(b) The problem was investigated by allowing only translation of the polygon—but not rotation. It seems that efficient algorithms allowing both translation and rotation may be devised by formulating the problem completely in terms of supporting points and decomposition boundary tracing. (Some investigations in this direction by the author have indeed produced some encouraging results.)

(c) It also appears that the boundary tracing technique could be used effectively in solving polygon union and intersection problems. We must note that some work in this direction was once initiated by Guibas et al. [5].

(d) After living with the subject for a long time we strongly feel that the significance of Minkowski operators is tremendous and need further investigation. Not only their importance is being felt in solving spatial planning, containment, and other related problems, but also in the areas like image processing, geometric modelling, biological form description and analysis, graphic arts and typesetting, etc. [13, 3, 7, 2]. Besides their impact on classical problems, the efficient computation of Minkowski operations give rise to theoretical and algorithmic problems of their own and can significantly enrich the discipline of computational geometry.

ACKNOWLEDGMENTS

I would like to express my gratitude to Professor R. Narasimhan, Dr. S. Ramani, Dr. S. P. Mudur, Dr. S. K. Ghosh, Ms. L. Parida, Mr. T. M. Vijayaraman, Mr. P. R. Chandrasekharan, and particularly to Mr.
A SOLUTION OF POLYGON CONTAINMENT

P. Sadanandan for their help at various stages of preparation of the manuscript. And, finally, I thank the referees of this paper for constructive suggestions.

REFERENCES