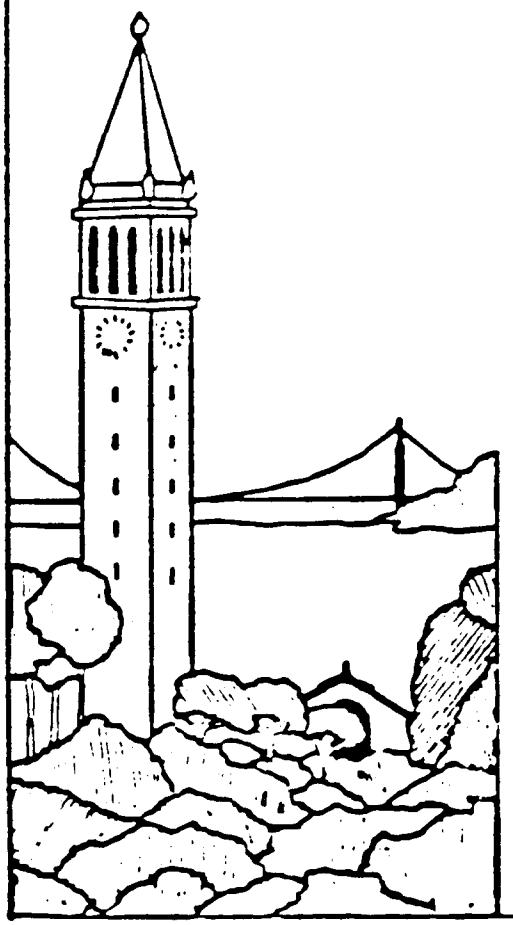
# Stabbing Isothetic Boxes and Rectangles in $O(n \lg n)$ Time

*Michael E. Hohmeyer and Seth J. Teller*



**Report No. UCB/CSD 91/634**

**June 1991**

**Computer Science Division (EECS)**
**University of California**
**Berkeley, California 94720**

# Stabbing Isothetic Boxes and Rectangles in $O(n \lg n)$ Time

Michael E. Hohmeyer
Seth J. Teller
Computer Science Division
University of California at Berkeley
Berkeley, California 94720

June 12, 1991

## Abstract

An algorithm is presented for determining in $O(n \lg n)$ time whether there exists a line that stabs each of $n$ polygons whose edges are from three sets of parallel lines. Using this algorithm, one can determine in $O(n \lg n)$ time whether there exists a line that stabs each of $n$ isothetic boxes or rectangles. If any stabbing line exists, the algorithm computes and returns one such stabbing line.

## 1 Introduction

Suppose one wishes to determine if it is possible to see into a region in $\mathbf{R}^3$ through a series of opaque planar barriers with polygonal holes. If it is possible to see into the region, then there must be a line that *stabs* all of the polygonal holes. Avis and Wenger present the first algorithm to determine if such a line exists and to calculate a representative line [1]. This algorithm runs in time $O(n^4 \lg n)$. Pellegrini presents an $O(g^2 n^2 \lg n)$ algorithm for this problem where $g$ is the number of distinct orientations of the normals of the polygons [5,4]. Thus, for the special case where there are a fixed number of distinct polygon normals, Pellegrini's algorithm runs in $O(n^2 \lg n)$ time. In practice it may be

the case that all of the polygons are *isothetic rectangles*, that is, rectangles whose edges are from lines parallel to one of the three principal axes [7]. In this paper we present an $O(n \lg n)$ algorithm that computes a stabbing line through $n$ isothetic rectangles or boxes, or determines that no such line exists.

## 2 Dual of 2-D Problem

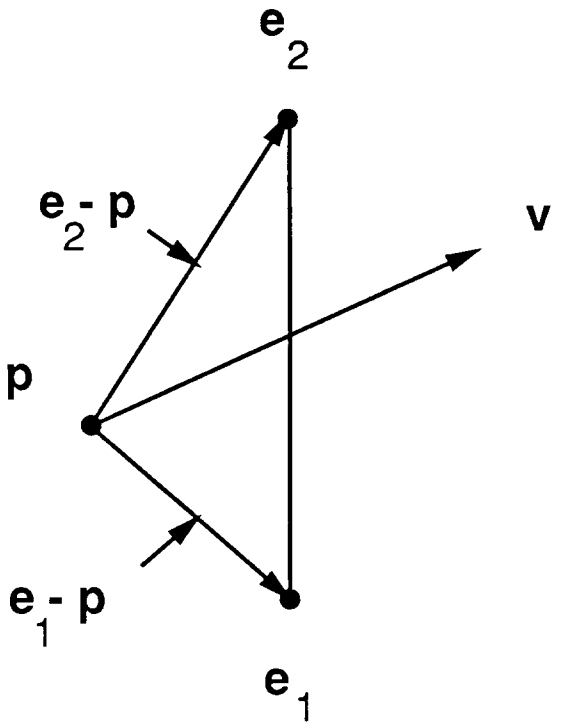



Figure 1: An oriented line.

Before considering the problem in three dimensions we consider it in two. In the plane, instead of stabbing rectangles we stab line segments. The stabbing line will be oriented; i.e., specified by a point $\mathbf{p}$ and a direction $\mathbf{v}$. The line segments are oriented in the sense that they are an ordered pair of end points $(\mathbf{e}_1, \mathbf{e}_2)$. A line is feasible if

$$(\mathbf{v} \times (\mathbf{e}_2 - \mathbf{p})) \cdot \hat{\mathbf{z}} \geq 0 \tag{1}$$

$$((\mathbf{e}_1 - \mathbf{p}) \times \mathbf{v}) \cdot \hat{\mathbf{z}} \geq 0 \tag{2}$$

as depicted in Figure 1. If the line segements are not isothetic, the problem of orienting them is quite difficult itself. However, when they are isothetic in 2-D one can fix the orientation of the vertical lines to admit solution lines going left to right and to orient all of the horizontal lines first to admit solution lines going upward and then to admit solution lines going downward. Thus, there are only 2 meaningful orientations in 2-D. Similarly in 3-D there are only 4, so that it is not too expensive to consider them all.

Since the line segments are axis-aligned, we can transform this into the following problem. Given two sets of points, $\{p_i\}$ and $\{q_i\}$, find a line that passes above the $\{p_i\}$ and below the $\{q_i\}$. We will refer to points that constrain a line to pass by another line in a particular fashion as *oriented* points. Thus we have reduced the problem to finding a feasible line passing by a set of oriented points in the correct fashion.

Given a set of $n$ oriented points in the plane we can describe the space of feasible lines. Refering to Figure 2, the points below the feasible region constrain lines to pass above them while the points above the feasible region constrain lines to pass below them. The feasible region appears as a bowtie-shaped shaded region. If we parametrize the feasible lines by their intercepts, $a$ and $b$, with the two vertical lines, $A$ and $B$, respectively, we see that each line in $x-y$ space corresponds to a point in $a-b$ space. The set of lines in $x-y$ space through a point in $x-y$ space corresponds to a line in $a-b$ space. The set of lines passing an oriented point in the correct fashion corresponds to a half plane in $a-b$ space, indicated by the arrows in Figure 2. The set of feasible lines is then the intersection of these half spaces, i.e. a convex polygon. This polygon has at most $n$ vertices. Each vertex of the convex hull of the set of feasible lines in $a-b$ space corresponds to a line in $x-y$ space.

A feasible line can be found in linear time using a linear programming algorithm [3,6] We recommend [6] since it is straightforward to implement and fast for low dimensions.

# 3 3-D Algorithm

Lines in $\mathbf{R}^3$ are oriented as follows. Let $L_1$ and $L_2$ be lines in $\mathbf{R}^3$. Let $\mathbf{p}_1$ be a point on $L_1$ and $\mathbf{v}_1$ be a vector along $L_1$, and similarly for $\mathbf{p}_2$, $L_2$ and $\mathbf{v}_2$, as depicted in Figure 3. The line $L_1$ will be feasible with respect to $L_2$ if and only if

$$\mathbf{v}_1 \times \mathbf{v}_2 \cdot (\mathbf{p}_1 - \mathbf{p}_2) \geq 0 \tag{3}$$

Note that if one line is considered fixed and the other variable, then the above constraint is quadratic.

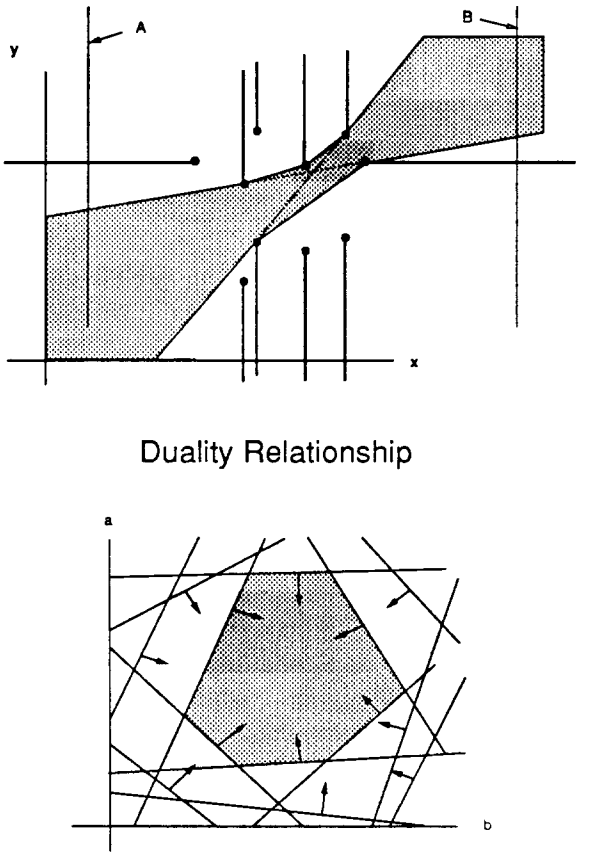



Figure 2: Lines in $\mathbf{R^2}$ and their dual representation.

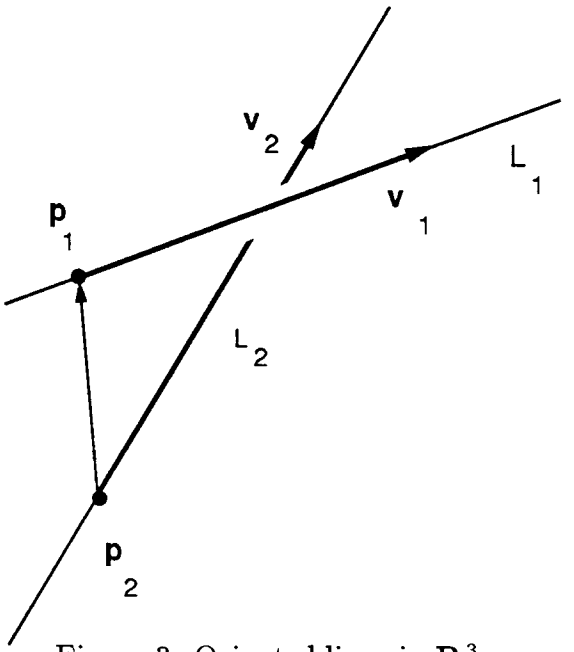



Figure 3: Oriented lines in $\mathbf{R}^3$.

This is not simply a case of choosing the parametrization of the variable line badly; there is no coordinatization of feasible lines so that the feasible region is bounded by hyperplanes. To see this, consider the line constraints that arise from requiring a line to pass through the axis-aligned rectangles pictured in Figure 4. There is no way to smoothly move from one of the solutions pictured to the other. If we could parametrize the space of feasible lines in such a way that the constraints were linear then the space of feasible lines would not only have to be connected but, in fact, convex. This example shows that no parametrization exists which causes the constraints to be linear. Thus, an application of linear programming as in the 2-D case is ruled out.

We wish to solve the following problem: given a set S of $n$ oriented, axis-aligned lines in $\mathbf{R}^3$ representing constraints, we wish to determine if there is a satisfying line. We will present the $O(n \lg n)$ algorithm first as a slower algorithm and then show how it can be made more efficient. We begin by making the following observation:

**Lemma 1** *If there is any feasible line, then there must be a feasible line intersecting four constraint lines.*

**Proof**: Coordinatize the all lines by their intercepts with the planes containing two

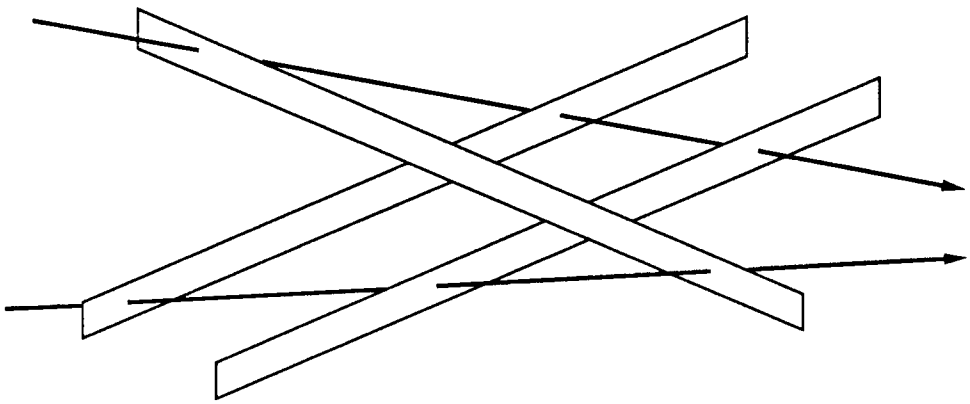


Figure 4: The space of feasible lines can be disconnected.

constraint rectangles. Call the coordinates of the feasible lines $S_4 \subset \mathbf{R}^4$ The edges of the rectangles bound the coordinates $S_4$. The constraints are not strict inequalities so that $S_1$ is closed and thus compact. Thus, there must be a feasible line $L$ which minimizes the first coordinate of $S_4$. Clearly $L$ must be on the boundary of the $S_4$. Thus, it intersects a constraint line, $C_1$. Now coordinatize a subset of the feasible lines by their intercepts with $C_1$ and their intercept with a plane $P$ containing a rectangle $R$ and not containing $C_1$. Call the coordinates of the feasible lines $S_3 \subset \mathbf{R}^3$. The edges of the rectangle that meet $C_1$ bound the first coordinate of $S_3$. $R$ bounds the second two coordinates of $S_3$. $S_3$ is again compact. The feasible line which minimizes the first coordinate of $S_3$ must be on the boundary of $S_3$, i.e., intersecting a second constraint line $C_2$.

Case 1: If $C_1$ and $C_2$ intersect then coordinatize the set of lines passing through $C_1$ and $C_2$ by their intercepts with a plane $P$ containing a rectangle $R$ and not containing $C_1$ and $C_2$. Call the coordinates of the feasible lines $S_2 \subset \mathbf{R}^2$. $S_2$ is bounded by $R$ and is again compact. The feasible line minimizing the first coordinate of $S_2$ must intersect a third constraint line $C_3$. Coordinatize the lines passing through $C_1$, $C_2$ and $C_3$ by their intercept with $C_3$. Call the coordinates of the feasible lines $S_1 \subset \mathbf{R}$. $S_1$ is bounded by the rectangle edges that meet $C_3$. Thus $S_1$ is compact. The feasible line in $S_1$ minimizing its coordinate must intersect a fourth line $C_4$.

Case 2: If $C_1$ and $C_2$ do not intersect then coordinatize the set of lines passing through $C_1$ and $C_2$ by their intercepts with $C_1$ and $C_2$. Call the coordinates to the feasible lines $S_2 \subset \mathbf{R}^2$. The first coordinate of $S_2$ is bounded by the rectangle edges that meet $C_1$ and the second coordinate is bounded by the rectangle edges that meet $C_2$. Thus $S_2$ is compact. The feasible line which minimizes the first coordinate of $S_2$ must intersect a third constraint line $C_3$. Coordinatize the lines passing through $C_1$, $C_2$ and $C_3$ by their intercept with $C_1$. Call the coordinates of the feasible lines $S_1 \subset \mathbf{R}$. $S_1$ is bounded by the rectangle edges that meet $C_1$. Thus $S_1$ is compact. The feasible line in $S_1$ minimizing its coordinate must intersect a fourth line $C_4$.

■

Let $X$, $Y$, and $Z$ be the set of x-aligned, y-aligned, and z-aligned constraint lines, respectively. Since each constraint line belongs to one of $X$, $Y$, or $Z$, it is clear that there must be a feasible line intersecting two lines from the same set.

This is the key to exploiting the fact that the lines come from three sets of parallel lines. Consider a set of lines through two parallel lines $A$ and $B$. Coordinatize the lines by their intercepts with $A$ and $B$. If we restrict the set of lines to be feasible with respect to an additional line in $\mathbf{R}^3$, the restriction can be expressed as a linear constraint on the coordinates of the lines. This is because the solution lines lie in a plane through $A$ and $B$. This would not be true if $A$ and $B$ were skew.

We can devise an $O(n^3)$ algorithm as follows:

**Stabbing_ Line_1** $(X, Y, Z)$
*(1)* $S = X \cup Y \cup Z$
*(2)* **for** $G = X, Y, Z$
*(3)* $F = S - G$
*(4)* **for** *each pair* $(A, B) \in G$
*(5)* *determine the plane* $P$ *defined by* $A$ *and* $B$
*(6)* **for** *each* $C \in F$
*(7)* *let* $p_C$ *be the point where* $C$ *intersects* $P$
*(8)* **endfor**
*(9)* *use a linear programming algorithm to determine if there exists a line* $L$ *in* $P$ *satisfying the* $p_C$*'s.*
*(10)* **If** *there exists an* $L$ **return** *(L)*
*(11)* **endfor**
*(12)* **endfor**
*(13)* **return** *(infeasible)*

The outer loop (from line 2 to 12) is executed 3 times, the next (from line 4 to 11) is executed $O(|G|^2)$ times, the innermost loop (lines 6 to 8) is executed $O(|F|)$ times, and the linear programming algorithm takes $O(|F|)$ time. Thus, the total running time is $O(n^3)$.

The first optimization to this algorithm is to note that the loop over all pairs in $G$ (from line 4 to 11) is considering too many pairs. Consider, for example, the case when $G = X$ (on line 2), as depicted in Figure 5. Let $Q$ be a plane perpendicular to the x-axis. Let $T$ be the set of intersections of the lines in X with the plane $Q$. A line $L$ in $\mathbf{R}^3$ will be feasible with respect to the set of lines $S$ only if the projection $l$ of $L$ onto $Q$ is feasible with respect to the points $T$. Let $a$ and $b$ be the intersections of the ($x$-aligned) lines $A$ and $B$ with the plane $Q$. All of the lines intersecting $A$ and $B$ will project to the same line in $Q$, namely the line $l$ that intersects $a$ and $b$. For many of the pairs $(A, B) \in G$ the projection of $L$ will not be feasible with respect to $T$. Thus, many of the pairs $(A, B)$ can be removed from the second loop (lines 4 to 11) in **Stabbing_Line_1**. At most $|G|$ lines defined by pairs of points from $T$ will be feasible with respect to $T$, as explained in Section 2.

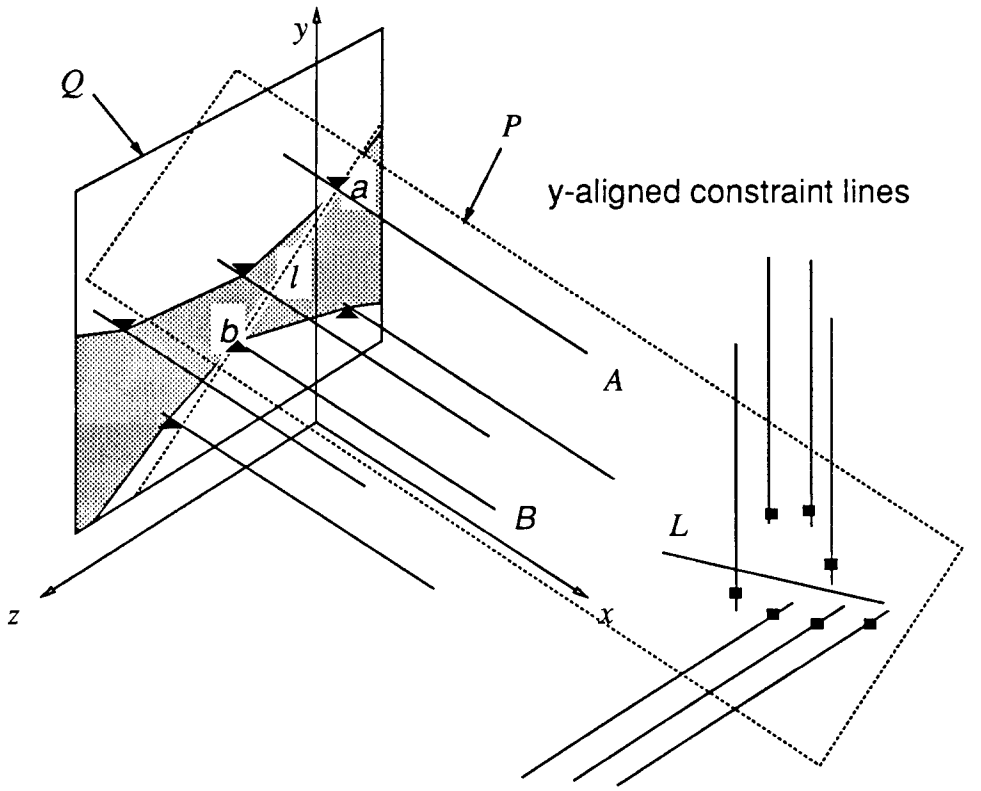



Figure 5: Many planes determined by $A$'s and $B$'s contain only infeasible lines.

We can now describe an $O(n^2)$ algorithm:

```
Stabbing_Line_2  (X, Y, Z)
(1)    S = X ∪ Y ∪ Z
(2)    for G = X, Y, Z
(3)            F = S − G
(4)            let Q be a plane perpendicular to the lines in G
(5)            let T be the intersections of G with Q
(6)            let H be the convex hull (in the dual space) of the
                       lines in Q feasible with respect to T
(7)            for each l ∈ H
(8)                    determine the plane P defined by l and the
                               direction of the lines in G.
(9)                    for each C ∈ F
(10)                           let p_C be the point where C intersects P
(11)                   endfor
(12)                   use a linear programming algorithm to
                       determine if there exists an L in P satisfying
                       the p_C's.
(13)                   If there exists an L return ( L)
(14)           endfor
(15)   endfor
(16)   return (infeasible)
```

Let $X'_P$ be the intersection of the lines in $X$ with the plane $P$, and similarly define $Y'_P$ and $Z'_P$. Let $\mathcal{X}(P)$ be the convex hull of the lines in $P$ feasible with respect to $X'_P$, and similarly define $\mathcal{Y}(P)$ and $\mathcal{Z}(P)$.

Consider again the case when $G = X$ (on line 2). In the inner loop we are testing $|\mathcal{X}|$ times whether $\mathcal{Y}(P)$ and $\mathcal{Z}(P)$ intersect. Each test takes $O(|Y| + |Z|)$ time if we use a linear programming algorithm as in **Stabbing_Line_2**. The key to reducing this cost is to note that while the specific coordinates of $\mathcal{Y}(P)$ and $\mathcal{Z}(P)$ depend on the plane $P$, the *structure* of $\mathcal{Y}(P)$ and $\mathcal{Z}(P)$ does not depend on $P$ as depicted in Figure 6. In [2] an $O(\lg\ n)$ algorithm is described for testing whether convex polygons intersect. We can build the necessary query structures outside of all of the loops. When a particular value in the query structure must be used it can be computed in constant time. Thus, we can compute whether $\mathcal{Y}(P)$ intersects $\mathcal{Z}(P)$ in $O(\lg(|\mathcal{Y}| + |\mathcal{Z}|))$ time. The final $O(n\ \lg\ n\ )$ algorithm is as follows:

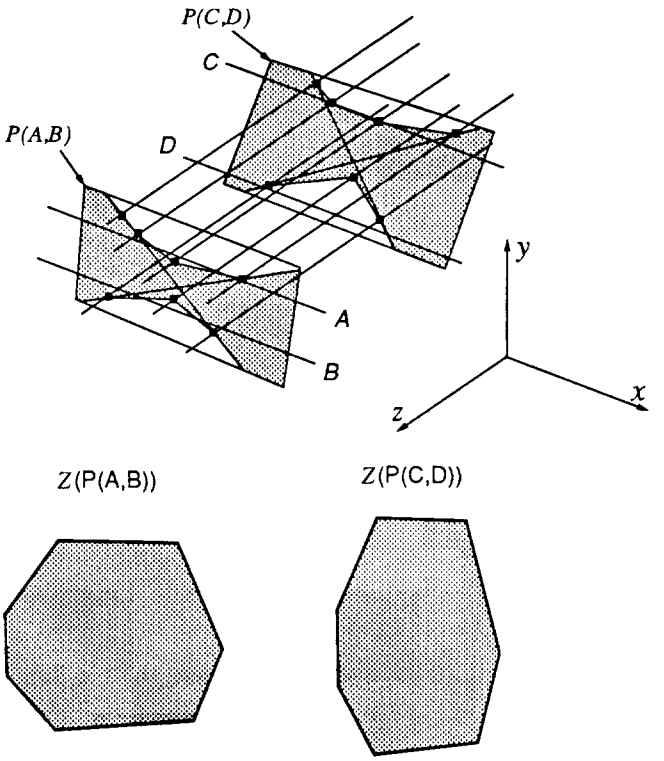



Figure 6: The structure of the region of feasible lines does not depend on P.

**Stabbing_Line_3** $(X, Y, Z)$
*(1)* *construct* $\mathcal{X}()$, $\mathcal{Y}()$, *and* $\mathcal{Z}()$
*(2)* *Let* $\mathcal{S}[0] = \mathcal{X}()$
*(3)* *Let* $\mathcal{S}[1] = \mathcal{Y}()$
*(4)* *Let* $\mathcal{S}[2] = \mathcal{Z}()$
*(5)* **for** $i = 0, 1, 2$
*(6)* **for** *each vertex* $L$ *of* $\mathcal{S}[i]$ *($L$ is a line)*
*(7)* *determine the plane* $P$ *defined by* $L$ *and the direction of the lines in* $\mathcal{S}[i]$.
*(8)* **If** $\mathcal{S}[(i+1) \bmod 3](P)$ *intersects* $\mathcal{S}[(i+2) \bmod 3](P)$ **return** *(the line corresponding to the point of intersection)*
*(9)* **endfor**
*(10)* **endfor**
*(11)* **return** *(infeasible)*

# 4 Stabbing Isothetic Boxes

When one has polygons that are not axis-aligned, then the following approximation might be used. The polygons to be stabbed can be enclosed in bounding boxes and the following question posed instead: "Is there a line that stabs all of the boxes?" A line that stabs all of the boxes must stab at least one face from each box. Thus, one could run **Stabbing_Line_3** on each of the $6^n$ $n$-tuples of faces. This would lead to a $6^n n \lg n$ algorithm. Fortunately we can do better.

Any line, whether it stabs all of the boxes or not, must fall into one of the four following categories.

1. $dy/dx > 0$, $dz/dx > 0$
2. $dy/dx > 0$, $dz/dx \leq 0$
3. $dy/dx \leq 0$, $dz/dx > 0$
4. $dy/dx \leq 0$, $dz/dx \leq 0$ .

For each category of lines we can pick out a set of six *significant edges* as depicted in Figure 7. A line of a certain type will stab the box if and only if the line passes correctly around each of the box's significant edges. For instance, in the coordinate system of

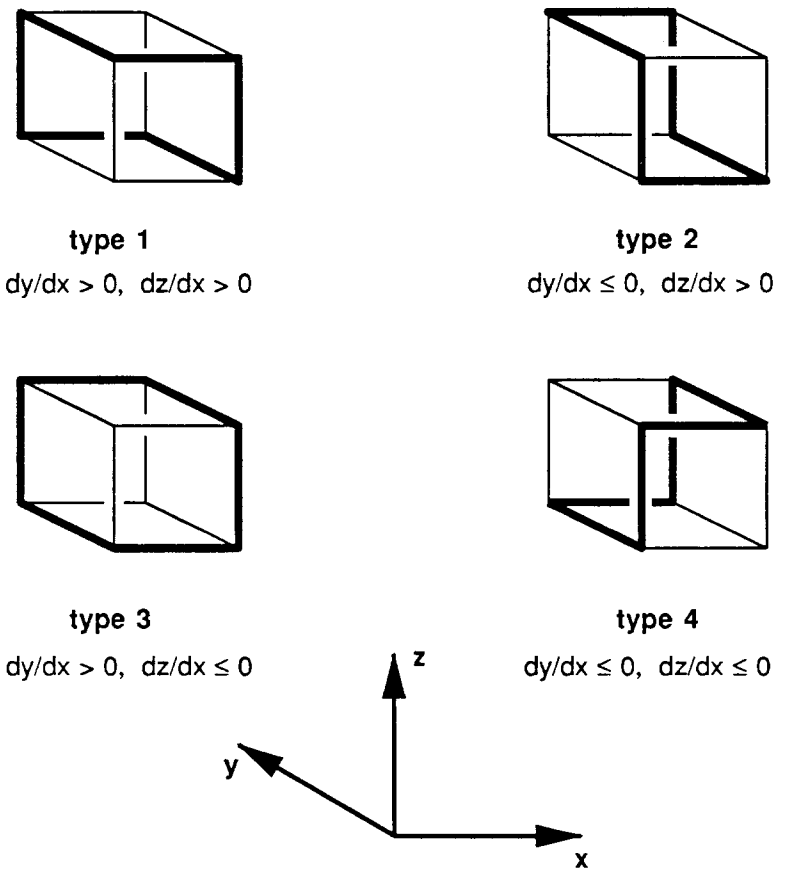



Figure 7: Significant edges for the four categories of line.

Figure 7, all of the lines of sight from the reader's eye are of type 3. Thus, the significant edges for this case are exactly the silhouette of the box.

One can determine if there exists a line that stabs each of $n$ boxes by running **Stabbing_Line_3** first on the edges appropriate for lines of type 1 then on those edges for type 2 lines and so on until either a stabbing line has been found or one has tried all four types. Since there are only four types, this algorithm requires at most $O(n \lg n)$ time.

If one knows beforehand that the solution lines can only be from a smaller set of the types of lines then clearly the others need not be checked. For example, if a pair of boxes $B_1$ and $B_2$ are are such that the $x$-minimum of $B_2$ is greater than the $x$-maximum of $B_1$ and the $y$-minimum of $B_2$ is greater than the $y$-maximum of $B_1$ then any feasible line must be either type 1 or type 2. In general, if two of the coordinates of a pair of boxes are separate then only two types must be checked, and if three of the coordinates of a pair of boxes are separate then only one type must be checked.

# 5 Acknowledgements

The authors thank Nina Amenta, Raimund Seidel, and Carlo Séquin for their helpful discussions regarding this work.

# References

[1] D. Avis and R. Wenger. Algorithms for line traversals in space. *Proceedings of the 3rd annual symposium on computational geometry*, pages 300–307, 1987.

[2] David P. Dobkin and Diane L. Souvaine. Detecting the intersection of convex objects in the plane. Technical Report 89-9, DIMACS, 1989.

[3] N. Megiddo. Linear-time algorithms for linear programming in $R^3$ and related problems. *SIAM Journal of Computing*, 12:759–776, 1983.

[4] M. Pellegrini and P. Shor. Finding stabbing lines in 3-dimensional space. *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 24–31, 1990.

[5] Marco Pellegrini. Stabbing and ray-shooting in 3-dimensional space. Technical Report 540; Robotics Report No. 230, New York University Courant Institute of Mathematical Sciences, Computer Science Division, 251 Mercer St., NY NY, 10012, 1990.

[6] Raimund Seidel. Linear programming and convex hulls made easy. In *ACM Symposium on Computational Geometry*, pages 211–215. ACM Press, 1990.

[7] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In *SIGGRAPH '91 Conference Proceedings*, August 1991. To appear.