

# Containment Algorithms for Objects in Rectangular Boxes \*

R. R. Martin      P. C. Stephenson  
Department of Computing Mathematics,  
University College Cardiff

## Abstract

A family of algorithms is presented for solving problems related to the one of whether a given object fits inside a rectangular box, based on the use of *Minkowski Sums* and convex hulls. We present both two and three dimensional algorithms, which are respectively linear and quadratic in their running time in terms of the complexity of the objects. In two dimensions, both straight sided and curved sided objects are considered; in three dimensions, planar faced objects are considered, and extensions to objects with curved faces are discussed.

## 1 Introduction

Computational Geometry has wider uses than just the design of solid objects. Here, we present a family of algorithms for solving a class of problems concerning whether a given object will fit into a rectangular box or not. The potential for application of such algorithms is widespread. If we are manufacturing two dimensional objects, our raw material will often come in rectangular sheets; similarly, in three dimensions, the starting point is often a rectangular block of material. We may need to tell which of several standard sizes of material is large enough for our object to be manufactured from. Related algorithms can be used to tell us what length of stock is required if the material comes in the form of a bar with rectangular cross section. A further possibility is that we wish to package an object in a rectangular case or crate, when the cost of the packing crate depends on its surface area, so we may wish to find the rectangular box of minimum area which encloses our object.

Further from end-user applications, but still of great importance, is the idea often used in solid modellers of placing boxes around objects as a crude filter when computing intersections between objects. Obviously, the closer each box is to the original object, the better the filtering will be, and the greater the saving in computational time. Thus algorithms which find minimal boxes containing given objects are also of direct use in geometric modelling systems.

We will give algorithms both in two and three dimensions for solving a variety of containment problems. The starting point will be the following: given an arbitrary polygon

---

\*This work was supported by an SERC ACME Grant, Number GR/D 59434

and a two-dimensional rectangular box, we wish to decide whether the polygon will fit into the box, and if so, how to translate and rotate the polygon so that it does fit.

We will then see how the basic ideas involved can be extended in a variety of ways, for example, to more complex objects with curved sides, and to solving related problems such as the minimum perimeter rectangular box the object will fit in. Finally a generalisation of the ideas to three-dimensional objects and boxes is presented.

We would like to point out that the algorithms we present are of direct practical use. They take linear time (in the number of sides of the object) in two dimensions, and quadratic time in three dimensions, as will be shown later. Furthermore, they are reasonably straightforward to implement even in three dimensions.

## 2 Related Work

Chazelle [4] gives a general algorithm for deciding whether a polygon  $P$  will fit into a convex polygon  $Q$ , which is obviously a more general case than the basic problem we are considering here (but is still not completely general — note that  $Q$  must be convex). His method takes time  $O(pq^2)$ , where  $P$  has  $p$  vertices and  $Q$  has  $q$  vertices. However, his method is quite complex, and he does not mention what happens if  $P$  has curved sides, or how his method might be extended to higher dimensions. Because we are dealing with the special case of a rectangular box, we are able to take advantage of this fact to produce a simpler algorithm.

Another piece of work by Toussaint [15] shows how to compute the minimum area bounding box of a polygon  $P$  in time  $O(p)$ . He uses a method which he calls “Rotating Calipers”, and an observation by Freeman and Shapira [5] that the polygon will have one of its edges (at least) touching one of the sides of the box. The method we give below has some similarities to the ideas Toussaint uses, although we prefer to think of the object as rotating, rather than the box. Our work goes further in that we have used our method to solve a wide range of problems, and also have shown how the ideas can be used in three dimensions.

Although Freeman and Shapira’s paper has the title “Determining the Minimum Area Encasing Rectangle for an Arbitrary Closed Curve”, they replace the curve by an approximating polygon; also their method requires quadratic running time.

O’Rourke [13] has given an algorithm for finding minimal enclosing boxes for a set of points in three dimensions, but his method takes time  $O(n^3)$  when there are  $n$  points, whereas our method requires only quadratic time in three dimensions. (The convex hull of a set of points can be found in less than quadratic time, so the fact that our method deals with polygons rather than point sets is not an important difference with regard to the running time.)

## 3 Vector Sums

This section considers some theoretical ideas for deciding whether an object will fit in a box, based on the concept of Vector Sums, or Minkowski Sums as they are sometimes called. We will only develop them here as far as is necessary for helping with boxing problems. However, they are of wide use in Computational Geometry and Geometric Modelling and interested readers may care to refer to [6, 7, 9, 11]. It is interesting to

note that we start from the vector sum, and use it to gain insight into boxing problems, whereas Toussaint [15] gives as one use of his rotating calipers idea the computation of the vector sum of two convex polygons.

The definition of the Vector Sum  $A \oplus B$  of two geometric objects  $A$  and  $B$  is all points which are the (vector) sum of some point in  $A$  with some point in  $B$ :

$$A \oplus B = \{\mathbf{a} + \mathbf{b} | \mathbf{a} \in A, \mathbf{b} \in B\}. \quad (1)$$

It should be noted that although the position of the resulting object depends on the initial choice of origin, the shape of the result does not.

We will also need to consider the complement  $A'$  of an object  $A$ , which is given by the set of all vectors not in  $A$ ,

$$A' = \{\mathbf{a} \notin A\}. \quad (2)$$

Let us look at the two dimensional problem of whether a polygon will fit inside a rectangular box. We will denote the box by  $A$ , and the object that we wish to fit into it by  $B$ . For the moment, let us just consider  $B$  in a single orientation. In the simple case that  $B$  is a convex polygon,  $A \oplus B$  can be found algorithmically by merging the edges of  $A$  and  $B$  in slope order. More interesting, however, is the observation that for general planar shapes  $B$ , if we form  $(A' \oplus B)'$ , the result will be a new rectangle which has shrunk horizontally and vertically by the width and height of  $B$ , assuming that  $A$  is large enough to contain  $B$  (otherwise the result will be the empty set). These ideas are illustrated in Figure 1. Perhaps an easier way of regarding the set  $(A' \oplus B)'$  is to think of it as telling us the amount of play there is when we put object  $B$  inside object  $A$ .

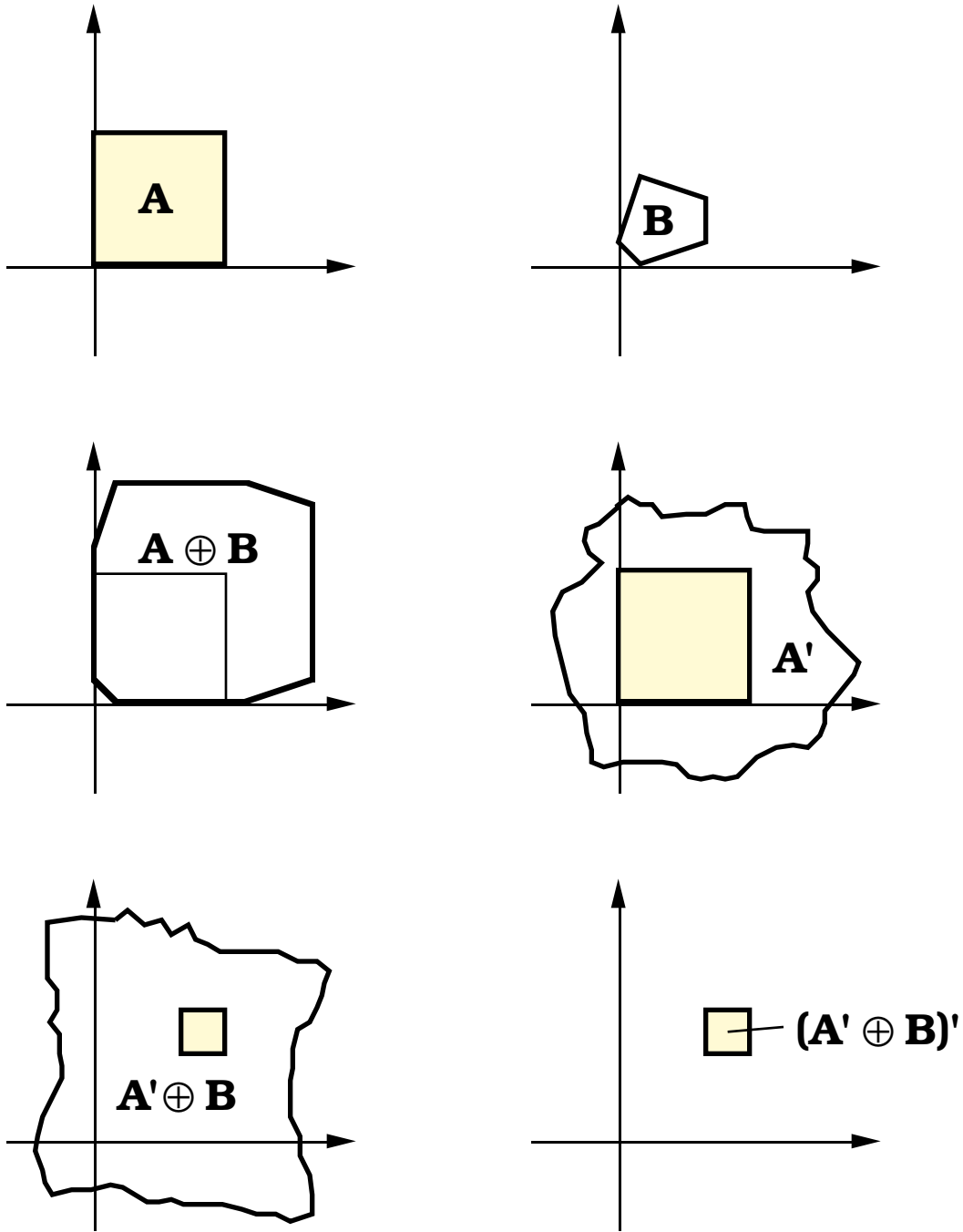
The problem we are trying to solve is rather more complex however, as we wish to see if  $B$  will fit inside  $A$  in *any* orientation. Conceptually we do this by forming the vector sum for every possible angle  $\theta$  as we rotate  $B$  through  $180^\circ$  (not  $360^\circ$ , because of the symmetry of the rectangle) while keeping  $A$  fixed. The result is a set in  $x, y, \theta$  space. If this set is not empty, a value for  $\theta$  can be found such that the object will fit in the box. Possible positions and orientations in which it does fit can easily be deduced from the set.

It is not too difficult to see that such ideas can readily be extended to tell whether a three dimensional object will fit inside a three dimensional box.

In practice, although this vector sum approach is useful for considering the problem, it turns out that we do not need vector sum algorithms for solving the problem. Instead, by considering the result of the vector sum described above, we can use as a basis for our algorithm the following observation: the object will fit in the box if and only if, simultaneously for some rotational orientation of the object, the width of the object is less than or equal to the width of the box *and* the height of the object is less than or equal to the height of the box.

## 4 Does a Polygon Fit in a Box?

As will be explained as we go along, the algorithm presented here for solving this problem has two main parts to it. There is an algebraic part which tells us for a given configuration whether there is a solution, and there is a control part which tells us when we need to change from one configuration to the next, which is determined by considering the vertices of the polygon.



**Figure 1 - Vector Sum Operations**

## 4.1 Algebraic Basis of the Algorithm

We will first present a simplified version of the problem which naturally leads to the more general version.

### 4.1.1 Does a Rectangle Fit in a Box?

The simpler problem which we will consider first is whether a given rectangle of width  $a$  and height  $b$  fits inside a given box of width  $w_{box}$  and height  $h_{box}$ . We start by making the trivial check of whether the rectangle fits without rotation, i.e.  $a \leq w_{box}$  and  $b \leq h_{box}$ . If these conditions are satisfied, we have no more to do.

If we rotate the rectangle through an angle  $\theta$ , then if there is a solution, there must be one in the range  $0^\circ \leq \theta \leq 90^\circ$ , by symmetry. Limiting ourselves to this angular range, we can express the new width  $w$ , and height  $h$  of the rectangle as

$$w = a \cos \theta + b \sin \theta, \quad h = b \cos \theta + a \sin \theta. \quad (3)$$

The rectangle will fit in the box for a given value of  $\theta$  provided that, simultaneously,  $w \leq w_{box}$  and  $h \leq h_{box}$ .

If we eliminate  $\theta$  (using Gröbner Basis Algorithms [2]) from these two equations we obtain

$$b^2 w^2 + a^2 w^2 - 4abwh + b^2 h^2 + a^2 h^2 - b^4 + 2a^2 b^2 - a^4 = 0. \quad (4)$$

If we let  $a = r \cos \phi$  and  $b = r \sin \phi$  then  $w = r \cos(\theta - \phi)$  and  $h = r \sin(\theta + \phi)$ , with both of these angles being positive, as can be seen by drawing a diagram. The signs of these angles tells us which of the roots gives the correct solution, namely

$$h = \frac{2abw + (a^2 - b^2)\sqrt{a^2 + b^2 - w^2}}{a^2 + b^2}. \quad (5)$$

We may now substitute  $w_{box}$  for  $w$  in this expression. If the resulting value of  $h$  is less than or equal to  $h_{box}$ , we have solved the problem, provided, of course, that we can find a corresponding value for  $\theta$  from these values of  $w$  and  $h$ . This requires that  $h$  must be in the range

$$\min(a, b) \leq h \leq \sqrt{a^2 + b^2} \quad (6)$$

This value of  $\theta$  gives a solution in which the rectangle fits exactly across the width of the box. Thus we can place the rectangle into the box by first rotating it through  $\theta$ , and then translating it so that its lowest and leftmost vertices touch the bottom and left hand sides of the box respectively.

In practice, we may need to interchange the rôles of width and height to find a solution if the box is very wide. However, this will be ignored at present, as the control part of the algorithm will take care of this in the general case.

### 4.1.2 Does a Convex Quadrilateral Fit in a Box?

Let us now consider the problem of testing whether an arbitrary convex quadrilateral fits in the box. We start by performing the check for a trivial solution, as described above. Let the top, left, bottom, and rightmost points of the quadrilateral have coordinates  $(x_t, y_t)$ ,  $(x_l, y_l)$ ,  $(x_b, y_b)$  and  $(x_r, y_r)$  respectively. These points may not all be distinct.

For example, the same point may be both topmost and rightmost; this will not affect what follows. Slightly more problematical is the case when two (or more) points are equally rightmost (for example). In this case, we choose the lower one, because after a small rotation this point will remain rightmost, and the other(s) will not. Similar logic is applied to other special cases.

If we now rotate the quadrilateral through an angle  $\theta$ , small enough that the same points remain the top, left, bottom and rightmost points (which also implies that  $\theta$  is in the range  $0^\circ \leq \theta \leq 90^\circ$ , as before), then after rotation these points will have moved to  $(x'_t, y'_t)$  etc. where

$$x'_t = x_t \cos \theta - y_t \sin \theta, \quad y'_t = x_t \sin \theta + y_t \cos \theta, \quad (7)$$

and similarly for the other points. We can now express the width  $w$  and the height  $h$  of the quadrilateral as

$$w = x'_r - x'_l = (x_r - x_l) \cos \theta - (y_r - y_l) \sin \theta, \quad (8)$$

$$h = y'_t - y'_b = (x_t - x_b) \sin \theta + (y_t - y_b) \cos \theta. \quad (9)$$

This can be written in the form

$$w = a \cos \theta + b \sin \theta, \quad h = d \cos \theta + e \sin \theta \quad (10)$$

for short, which is now similar to the previous case. This time we find on eliminating  $\theta$  that

$$(e^2 + d^2)w^2 - 2(be + ad)hw + (a^2 + b^2)h^2 - a^2e^2 + 2abde - b^2d^2 = 0, \quad (11)$$

and that the root which gives the correct answer is

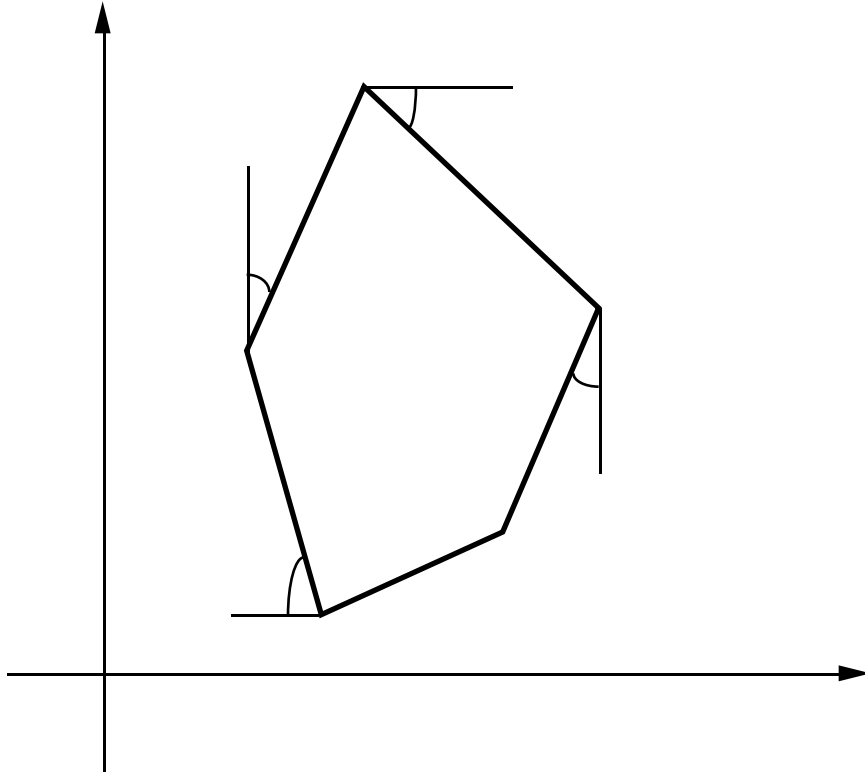
$$h = \frac{(ad + be)w + (ae - bd)\sqrt{a^2 + b^2 - w^2}}{a^2 + b^2}. \quad (12)$$

Again, if we can find a value for  $\theta$ , we can rotate and position the quadrilateral in the box, given one condition. This is that the value of  $\theta$  found must not be so large that any of the top, right, left or bottom points has been replaced by a different vertex of the quadrilateral. In other words, the topmost point must remain topmost and so on after this rotation. If this condition is not satisfied, we must repeat the process with a different configuration for the quadrilateral. This will be explained further in the next section as it is, in fact, the same as the control part of the algorithm used in the more general case.

Also, as we will again see in the next section, the calculations described above are sufficient to decide if *any* polygon will fit inside a rectangular box.

## 4.2 Control Part of the Algorithm

A simple, but useful, observation also made previously by Toussaint [15] is that if a given object will fit inside a rectangular box, then the convex hull of the object must also fit inside the box. This is easy to see. If any two points on the boundary of the object are taken, the straight line joining them lies inside the convex hull of the object; the line must also lie inside the box because the box is convex. Thus, as a first step, we take the



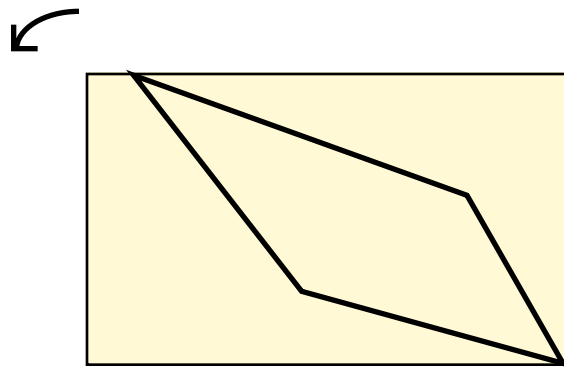
**Figure 2 - Finding the Next Active Vertex**

convex hull of the polygon we are testing. The convex hull of a polygon with  $n$  sides can be computed in  $O(n)$  time [8], thus allowing our final algorithm to still be linear.

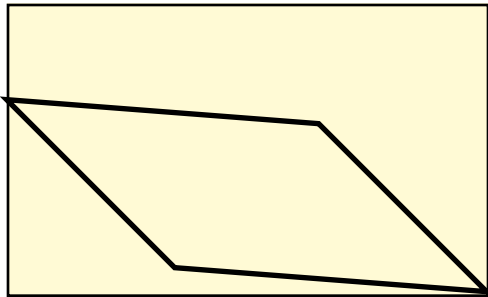
There are several advantages in finding the convex hull as a preprocessing step, even though it may be possible to devise an algorithm which does not explicitly do so. The first of these is that it is simpler to only consider convex polygons, both when designing the algorithm, and when implementing it. The second is that the convex hull of a complex polygon may have considerably fewer vertices than the polygon itself, which means that the fit testing algorithm will have less work to do.

A key observation which makes the quadrilateral algebra of the previous section especially important is as follows. At any given orientation of the polygon, we can tell whether the polygon will fit inside the box by examining *only* four of its vertices, which are the topmost, leftmost, bottommost and rightmost ones. The positions of the other vertices are not important, as they can not prevent the polygon fitting in the box, providing that these four extremal vertices lie inside it. Thus, we can use the algebra given above for a quadrilateral to tell if the four extreme vertices lie inside the box. The only problem we now have to solve is how to decide which are the extremal vertices, and at what angles of rotation this set of extreme vertices changes. We shall refer to these as the *active vertices*.

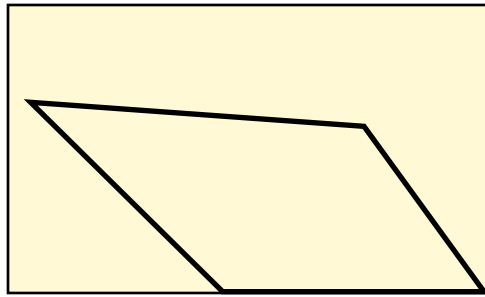
A simple scan through the vertices of the polygon enables us to find the initially active vertices, which we check for a trivial fit. Next, taking the initial orientation of the polygon, we define the solution region as all angles between zero, and the minimum one required such that one (or more) of the four active vertices is replaced by a new vertex as the polygon is rotated (anticlockwise). This is the minimum of the four angles shown in Figure 2.



**Initial State**



**Case A**



**Case B**

**Figure 3 - Coping with Narrow Objects**

We now use the quadrilateral algebra to look for a solution using the initial four active vertices. If the algebra gives a solution for  $\theta$ , we must check that it lies within the solution region. If  $\theta$  lies outside this region, it does not correspond to a useful solution, as it implies that the polygon has been rotated so far that the same set of four vertices are no longer the active ones. On the other hand, if we do find a solution within the solution region, we have solved the problem. To fit the polygon in the box, we rotate it through the angle found, and place the leftmost and lowest vertices against the left and bottom sides of the box. Alternatively, if we find no solution, or we find a solution outside the solution region, we must then rotate the original polygon through the minimum angle found above, giving a new set of active vertices, in which one (or more) of the original vertices is replaced by its clockwise neighbour. This process is repeated until either we find a solution, or we have rotated the original polygon through more than  $180^\circ$ , when by symmetry we know there can be no solution.

The algorithm as it stands has a small difficulty. The polygon may be so narrow that for any orientation the set of active vertices can not become as wide as the box, and so looking for solutions with  $w = w_{box}$  is not valid. However, we may avoid this problem as follows. Let us orient the box so that we may assume  $w_{box} \geq h_{box}$ . Now, consider an object which fits into the box as shown in the first diagram in Figure 3, touching the box at top and bottom. If we rotate the object anticlockwise, one of two things must happen, depending on the exact shape of the object. In Case A, where the object can become as wide as the box, we may make the assumption that  $w = w_{box}$  as before. In Case B, the object will have either its top or bottom side running along the top or bottom side of the box, and will still be inside the box.

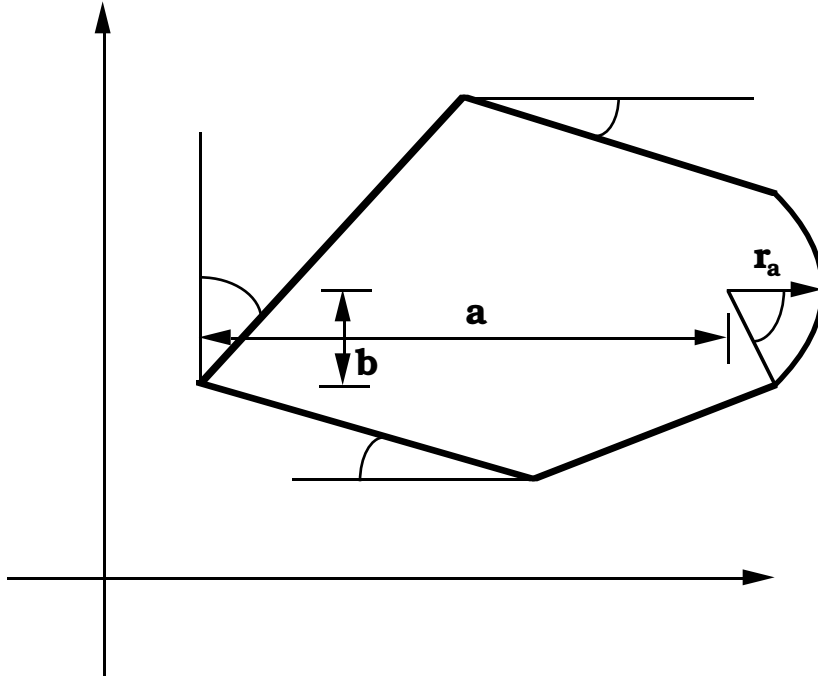
This leads to two modifications of the original version of the algorithm. At each change of active vertices we must perform a trivial test to decide whether the object is already inside the box to allow for Case B. Secondly, when we solve for  $h$  after setting  $w = w_{box}$ , and find the corresponding value of  $\theta$ , we must substitute this value of  $\theta$  back into the expression for  $w$  and check that it is indeed equal to  $w_{box}$ . This will ensure that the solution we have found is a valid one, and that the solution is not a spurious one based on a set of vertices for which  $w$  cannot be equal to  $w_{box}$ .

It is easy to prove that this algorithm takes linear time in the number of vertices of the polygon. Each configuration of active vertices is tested in constant time. We only need to consider the immediately adjacent vertex to each active vertex when finding the next set of active vertices because we are testing a convex polygon, so this step, too, takes constant time as there are always four active vertices. Finally, as the polygon is rotated, each edge can become horizontal or vertical only once, so the number of possible sets of active vertices is linear.

It should be noted that the only non-trivial calculations involved are square roots which means that the problem can be solved directly, without the need for an iterative method. This fact combined with the linear time complexity of the algorithm makes it fast in practice.

## 5 Does a Curved Object Fit in a Box?

Most theoretical computational geometers tend to restrict themselves to polygonal (or polyhedral) objects. However, many objects of real (industrial) interest have curved sides



**Figure 4 - Including Curved Edges**

(or faces). Thus, we feel that it is important to show how our algorithm can be extended to cope with such objects. In particular, we will look at objects whose sides consist of straight lines and circular arcs, and we will also briefly discuss more generally curved shapes.

Fortunately, many of the same ideas can be used as before. Again, we start by computing the convex hull of the object. One of the few authors who *has* considered theoretical computational geometry for curved objects is Souvaine. She gives an algorithm for finding convex hulls of “splinegons” (polygons generalised to include possibly curved sides) in linear time, using an approach based on bounding polygons [14].

Let us initially consider a polygon which includes a single circular arc edge, as shown in Figure 4. The rightmost point of the shape now lies on the circular arc, and slides around it as we rotate the polygon, rather than being fixed at a vertex of the polygon. Thus, we need to generalise the idea of active vertices to the *active points*, which are still the extrema of the object in the horizontal and vertical directions. From the diagram, we can see that the width of the object as it is rotated is given by

$$w = a \cos \theta + b \sin \theta + r_a, \quad (13)$$

where  $a$  and  $b$  are measured to the point at the centre of the circular arc, and  $r_a$  is its radius.

More generally, any or all of the active points of the shape may lie on such curved sides, and the height and width of the object are given by

$$w = a \cos \theta + b \sin \theta + g, \quad h = d \cos \theta + e \sin \theta + z \quad (14)$$

where  $a, b, d$  and  $e$  are measured either to vertices, or the centres of circular arcs as appropriate, while  $g$  and  $z$  are the sums of the radii of any circular arcs involved in the appropriate direction.

As before, we can eliminate  $\theta$  from these equations to find an expression for  $h$  in terms of  $w$ . The appropriate solution is

$$h = \frac{(a^2 + b^2)z + (ad + be)(w - g) + (ae - bd)\sqrt{a^2 + b^2 - (w - g)^2}}{a^2 + b^2}. \quad (15)$$

Having now established the necessary algebraic conditions for finding a solution, we must now see how the controlling part of the algorithm is affected by allowing curved sides. If one of the active points lies on a curve, the angle used for this point to determine the solution region is now just the angle subtended by the arc from the point to the end of the arc, as shown in Figure 4. A further possibility which did not arise in the straight edged case is that two successive edges may now join tangentially to each other. If this is so, we need not consider the end of the edge as a vertex in its own right, as there is only a single orientation (not a range) for which this vertex may take part in a solution. (The vertex can only instantaneously become active as the object is rotated.) Finally, a curved edge may turn through more than  $90^\circ$ , in which case it may contain both the leftmost and rightmost points, for example. A simple way of avoiding any problems which might arise, both algebraically and algorithmically, is to use a preprocessing step which splits such arcs into two adjacent segments.

By a simple extension of Figure 3 we can see that cases where the object is not able to become as wide as the box can always be detected at a changeover of solution regions, as in the polygonal case.

If we allow objects with more generally curved sides, we need to find analogous expressions for width and height as a function of angle, which would mean expressing the curves in polar form. Because the radius of the curve would now depend on  $\theta$ , it is unlikely that we would be able to eliminate  $\theta$  from this pair of expressions. Instead, an iterative method would be required to find  $\theta$  and  $h$  simultaneously given  $w$ .

On the other hand, the control part of the algorithm would remain unchanged. Indeed, one of the strengths of our method is that the control part of the algorithm is clearly separated from the geometric part, allowing different geometries to be used in conjunction with the same control mechanism.

## 6 Related Algorithms

In fact, by changing the algebra used, the same control mechanism also can be used to solve other related problems, as well as allowing for different geometry. Some typical examples are given below.

### 6.1 Minimum Area Bounding Rectangle Of A Polygon

We can use the theorem proved by Freeman and Shapira [5] quoted above to solve this problem without *any* algebra at all! Because the minimum bounding box for a polygon is guaranteed to contain the polygon with one of the polygon's edges lying along one of the box's edges, all we have to do is to rotate the polygon so that its edges are successively vertical or horizontal. This is exactly what the control part of the algorithm does, so all we have to do is compute the bounding rectangle and its area at each changeover of active vertices, and remember the smallest rectangle and its corresponding configuration as we

go along. In this particular case, our method is then identical to the method described by Toussaint [15].

## 6.2 Minimum Area Bounding Rectangle Of A Curved Object

Unfortunately, this simple trick no longer works when we have curved edges. Instead, we proceed as follows, using the control algorithm as if we were trying to find whether the object would fit inside a rectangle, but we use different algebra to search for a solution between changeovers.

As stated before, the width and height of the object are given by

$$w = a \cos \theta + b \sin \theta + g, \quad h = d \cos \theta + e \sin \theta + z. \quad (16)$$

Thus, the area  $A$  of a bounding rectangle is given by  $A = wh$ . This will be minimum when  $dA/d\theta = 0$ . This condition leads to the following equation for  $\theta$ :

$$(be - ad) \sin 2\theta + (ae + bd) \cos 2\theta - (az + dg) \sin \theta + (bz + eg) \cos \theta = 0 \quad (17)$$

giving the minimum rectangle between changeovers. This can be reduced to the quartic equation below where  $u = \tan \theta/2$ ; note that once again no iteration is required to find a solution:

$$(bz + eg - ae - bd)u^4 + 2(az + dg - 2(ad - be))u^3 + 6(ae + bd)u^2 + 2(az + dg + 2(ad - be))u - (bz + eg + ae + bd) = 0. \quad (18)$$

However, we need to be careful that we have found a local minimum, and not a maximum for the area. This can be assured by checking that  $d^2A/d\theta^2 > 0$ , i.e.

$$-2(ae + bd) \sin 2\theta + 2(be - ad) \cos 2\theta - (eg + bz) \sin \theta - (dg + az) \cos \theta > 0. \quad (19)$$

In summary, we thus use the control algorithm as before (excluding the initial triviality check and box reorientation), and compute the minimum enclosing rectangle between changeovers. We also must find the enclosing rectangle at each changeover. Again, we record and update the smallest solution found so far as the algorithm proceeds, until we have rotated the object through  $180^\circ$ .

## 6.3 Minimum Perimeter Bounding Rectangle Of A Curved Object

In two dimensional problems concerning wrapping, we may be more interested in perimeters than areas. By performing a similar analysis to the previous one, we start by expressing the perimeter  $P$  of the box is given as  $P = 2(w + h)$ . This is an extremum when  $dP/d\theta = 0$ , which requires us to solve

$$\tan \theta = \frac{e + b}{d + a}. \quad (20)$$

This extremum will be a minimum when  $d^2P/d\theta^2 > 0$ , and thus when

$$(e + b) \sin \theta + (a + d) \cos \theta < 0 \quad (21)$$

which simplifies using the previous equation to

$$\frac{1}{\cos \theta} < 0. \tag{22}$$

Because we are rotating the object anticlockwise, and because we have split any arcs which subtend an angle of more than 90 degrees, any solution of interest must be in the range  $0^\circ \leq \theta \leq 90^\circ$ . This contradicts the requirement that  $1/\cos \theta < 0$ . Thus, as in the case for finding the minimum bounding area rectangle for a polygon, the solution must lie *at* one of the orientations where we change from one solution region to another, and not between them. Thus, we proceed as in that case, by finding the perimeter of the bounding rectangle at each changeover, and recording the minimum value found (and the corresponding rectangle) as we rotate the object through 180 degrees.

## 6.4 Minimum Height Rectangle Of A Given Width

Another problem assumes that we wish to cut an object (possibly curved) from a rectangular strip of material, and we wish to know what is the minimum height (or as more usually stated, length) of strip required. The solution can be found by a simple extension of the original problem of whether the object will fit in a given box. The only difference now is that instead of stopping when we the first value of  $h \leq h_{box}$ , we continue, remembering the minimum value of  $h$  found as we rotate through  $180^\circ$ .

By analogy with a possibility mentioned in connection with the original algorithm, the problem is complicated by the fact that the object may be so small that it is smaller than the width of the strip. In this special case, the problem reduces to finding the minimum height of the object as it rotates. In fact, by using a very similar argument to the one used for finding minimum perimeter boxes, we can show that a minimum height can only occur at the changeovers between solution regions, and not between them. Thus, in fact, the algorithm given above *will* solve the overall problem for any object provided that we take care to include the calculation of the height of the box *at* changeovers of active points, as well as between them.

## 6.5 Does a Box Fit Inside an Object?

Just to show that we do not think that our algorithm can solve every problem, we will briefly mention one seemingly similar problem for which our methods are not appropriate. This is the problem of whether a box fits *inside* a given object, which in some senses is the opposite problem to the original one. The basic problem is that we have relied on the convexity of the box to generate the algorithm, and a general object may well not be convex. A second and weaker point is that we have also used the fact that a box has only four sides to guarantee the linear time complexity of the algorithm.

Baker [1] gives an algorithm for deciding if a convex polygon will fit inside a generally non-convex one, but with the very restrictive condition that rotation is not permitted.

## 7 Does a Polyhedron Fit Inside a Box?

If the above ideas are going to be of wide use, they must also be extended to cope with three dimensional objects. We will concentrate on the simple case of deciding whether a

polyhedral object fits in a box. Similar generalisations may be used to cope with objects with curved faces as curved arcs in two dimensions. However, spherical patches are much less frequently used in three dimensions than circular arcs in two dimensions, so a simple repetition of the arguments is not particularly helpful. The emphasis here will be on the control part of the algorithm, to which the user may add the algebra appropriate to variations on the basic problem.

## 7.1 Convex Hull of a Polyhedron

Various algorithms exist for finding convex hulls in three dimensions. If we start with a set of  $n$  points, it can be shown in this case to require at least  $O(n \log n)$  time [12]. However, it still seems to be an open question if we can do better than that if we start with a simple polyhedron rather than an arbitrary set of points. One method which is fairly easy to understand, and not too difficult to implement, even if not the most theoretically efficient, is the “Gift-Wrapping” method of Chand and Kapur [3]. It can be used in both two and three dimensions, and has the advantage of being readily adapted to cope with curved objects if necessary.

## 7.2 Algebraic Basis of the Algorithm

To see if a polyhedron will fit in a box, we must consider rotations about two different axes, which obviously makes the algebra more complicated (and the control part of the algorithm too). In fact, the extra difficulties with the algebra are greater than they may appear at first sight, because not only do we have three relations now for height, width, *and* thickness, but in fact two of them are inequalities rather than equations. If we position the object so that it just fits in across the height of the box, we must then check whether it also fits in across the width and thickness simultaneously. We start by generalising the two dimensional equations for width and height of the object as it is rotated to a set of three equations for its width, height and thickness, where thickness is measured in the  $z$  direction. We assume the first rotation is through an angle  $\theta$  about the  $z$ -axis, and the second through an angle  $\phi$  about the  $y$ -axis. The resulting expressions for the extent of the object in terms of the original coordinates of the active vertices are

$$w = a \cos \theta \cos \phi + b \sin \theta \cos \phi + c \sin \phi, \quad (23)$$

$$h = d \cos \theta + e \sin \theta, \quad (24)$$

$$t = f \cos \theta \sin \phi + g \sin \theta \sin \phi + k \cos \phi. \quad (25)$$

We shall call the two new active vertices “far” and “near”; their respective coordinates are  $(x_f, y_f), (x_n, y_n)$ , with the near point having the largest  $z$  coordinate. The values of the constants given above are

$$a = x_r - x_l, \quad b = y_l - y_r, \quad c = z_l - z_r, \quad (26)$$

$$d = y_t - y_b, \quad e = x_t - x_b, \quad (27)$$

$$k = z_n - z_f, \quad f = x_n - x_f, \quad g = y_f - y_n. \quad (28)$$

If the polyhedron fits inside the box in some orientation, it must simultaneously satisfy  $w \leq w_{box}$ ,  $h \leq h_{box}$ , and  $t \leq t_{box}$ . By using a similar approach to the method used in

two dimensions, we start by setting  $h = h_{box}$ , choosing  $h$  because it depends on only one of the angles. (The permissibility of this choice will be justified when we consider the control part of the algorithm.) This gives a value for  $\theta$  which can then be substituted into the other two inequalities. Unfortunately, in three dimensions, the method no longer only involves square roots, but inverse trigonometric functions must be computed:

$$\theta = \sin^{-1} \left( \frac{h_{box}}{\sqrt{e^2 + d^2}} \right) - \tan^{-1} \left( \frac{d}{e} \right). \quad (29)$$

When we substitute this value into the equations for the width and the thickness, it can be seen that we end up with expressions of the form

$$w = \alpha \cos \phi + \beta \sin \phi, \quad t = \gamma \cos \phi + \delta \sin \phi. \quad (30)$$

The problem is now to find a value of  $\phi$  which causes both  $w \leq w_{box}$  and  $t \leq t_{box}$  to be satisfied simultaneously. However, the form of these equations is exactly the same as those we already know how to solve for the two dimensional problem. We thus use the following strategy. First, we check if at  $\phi = 0$ , before rotation about the second axis, both  $w \leq w_{box}$  and  $t \leq t_{box}$ , in which case we already have a solution. Secondly, we can set  $w = w_{box}$  and solve for  $t$ . If  $t \leq t_{box}$  we have a solution, otherwise we finally set  $t = t_{box}$  and solve for  $w$ , checking if  $w \leq w_{box}$ . (In each case, we must check that the values of  $\theta$  and  $\phi$  found do lie within the solution region.) If none of these cases are satisfied, there is no solution within this solution region, so the control part of the algorithm must step on to the next solution region. Otherwise, we now have values for  $\theta$  and  $\phi$  which will fit the object inside the box.

There is one small problem with the method as it stands. When we try to find  $\theta$ , it may be the case that we are unable to compute the  $\sin^{-1}$  required, as its argument may be greater than 1. This will happen if the object is not large enough to be able to attain a height  $h_{box}$  on rotation through *any* value of  $\theta$ . We must trap such cases to prevent them from causing arithmetic errors, but otherwise they are ignored because the control part of the algorithm will ensure that solutions involving such configurations are still handled correctly.

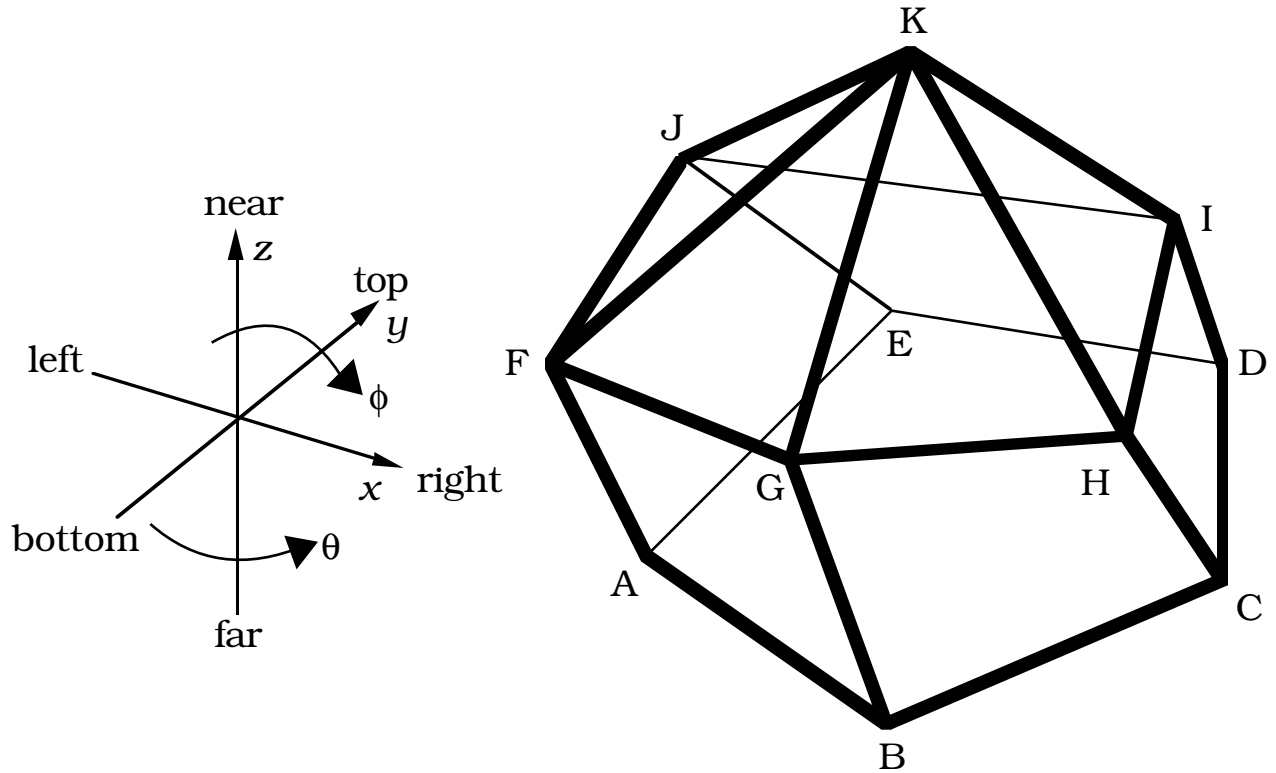
### 7.3 Control Part of the Algorithm

Having now discussed the algebra necessary for finding a solution if it exists within a single solution region, i.e. with a given set of active vertices, we must now examine the control part of the algorithm which tells when to change sets of active vertices.

As in the two dimensional case, it is necessary that if the object fits in the box at all, either the object fits inside the box with a pair of its vertices touching a pair of opposite faces of the box (which we assumed to be the top and bottom in the previous section) or it will fit inside the box with one of its faces lying in contact with one of the faces of the box. The second possibility will be detected like the analogous case in two dimensions by testing for a fit when we change over between active vertices.

The simplest way of ensuring that we do not miss any solutions is to repeat the following process with each of the box's three principal axes considered to be the height of the box in turn.

We take each face of the object in turn, and position the object on that face on the  $x$ - $y$  plane, which is the far plane of the box. (Let such a face of the object be  $ABCDE$  in



**Figure 5 - Fitting a Three Dimensional Object in a Box**

Figure 5.) Any of the vertices of that face may be the far vertex after rotation. We also determine which vertex is the near vertex (vertex  $K$  in Figure 5). There may be more than one vertex which can be the near vertex, in the case when a near edge or near face of the object is parallel to the far face. In such circumstances they must all be considered. We are going to make a rotation first about  $\theta$ , then about  $\phi$ . Thus, as long as we do not rotate too far about  $\theta$ , the rightmost of the vertices on the far plane (vertex  $C$  in the figure) must be the far vertex in any solution we find using our algebraic method. Similarly, if there is more than one possible near vertex, the leftmost one will be the near vertex after the  $\phi$  rotation, provided we do not rotate too far about  $\theta$ .

Thus, given the object in its initial orientation, we first check to see if the object already fits without rotation. We then solve the algebra given above to obtain a potential solution for  $\theta$  and  $\phi$ . We must now test whether the same vertices remain active after rotation through these angles. This can be done using the following important observation. *All* changeovers are now determined by the  $\theta$  rotation. The far vertex changes when a new vertex on the far face becomes rightmost (when vertex  $C$  is replaced by vertex  $B$  in the figure). A similar test is used to find the  $\theta$  rotation necessary for the near vertex to change if there is more than one candidate near vertex. Finally, we can tell when the leftmost, rightmost, top and bottom vertices change by considering those vertices connected by edges running in a direction of decreasing  $\theta$  from the current leftmost, rightmost, top and bottom vertices. (Vertex  $F$  is originally the leftmost vertex; after rotation vertices  $A$  and  $J$  are the ones which may replace it, for example.) We can now see that none of the active vertices will change provided that we restrict ourselves to the minimum of the three  $\theta$  values found above.

The limit for the  $\phi$  rotation can be found by considering the possible successor vertices to the near, far, leftmost and rightmost vertices. Again, the possibilities are the ones which are connected to them by an edge, in this case the ones which run in a direction of decreasing  $\phi$ . (In the figure,  $K$  will be replaced as the far vertex by one of  $F$ ,  $G$ , or  $J$ , while  $C$  must be replaced by  $H$  as rightmost.)

A solution is sought with the given set of active vertices. If one is not found, we rotate the object through the minimum  $\theta$  found above to move to the next configuration, and try again. We continue doing this until either we find a solution, or we have increased  $\theta$  through  $360^\circ$ . Note in this case we cannot use the  $180^\circ$  symmetry of the box as we are insisting that the  $\phi$  rotation must be positive.

As mentioned above, this process is repeated for each face of the object until either we have found a solution, or we have tried every possibility. In the latter case, the object must be too large to fit in the box.

We can justify that all possible configurations have been tested as follows. Although not all vertices may become the near vertex as various faces are laid flat on the far face of the box, all possible pairs of vertices which *can* occur as a pairing of far and near vertices will be tested at least once. In other words, some such pairings will be tested both ways round (with both possibilities for one vertex occurring as far and the other as near as different faces are laid flat on the far plane), but *all* such pairings will be tested at least *one* of the ways round. To see this, consider the object in some position with a given vertex as the near one, and another vertex as the far one. There is now a minimum angle of rotation such that one of the faces (or both) containing one of these vertices becomes parallel with the  $x$ - $y$  plane. This pair of vertices will be a far-near pair when that face is the far face: the vertex of the pair which is not in that face must still be the near vertex, because of the way we chose the face.

## 7.4 Time Complexity of the Algorithm

As stated in the introduction, the algorithm presented above has quadratic running time, which we will now prove. We will also make an observation that will avoid recomputing various quantities more often than necessary. In the following discussion, we assume that the object has  $f$  faces,  $e$  edges, and  $v$  vertices.

In the outer loop of the algorithm, each face is chosen as the far face in turn, and in the inner loop the object is rotated through  $\theta$ . As we do this, we find the minimum  $\theta$  for new active vertices, which involves searching the vertices connected by an edge from the current top, bottom, leftmost and rightmost vertices. However, as we rotate the object, each vertex can be in each active position only once. If we sum the number of edges leading away from *every* vertex of the object over all vertices, we just get twice the number of edges, as each edge is connected to exactly two vertices. Thus, at most we only need to perform  $O(e)$  tests for  $\theta$  limits for each face. (This spreading of the cost over all possible vertices is an example of an *amortized analysis*. For further examples in the field of Computational Geometry, see Guibas and Stolfi [10].)

The same technique can be applied to the  $\phi$  tests for the leftmost and rightmost vertices. However, the  $\phi$  test for the far and near vertices requires a little more care. The reason is that as we rotate the object in  $\theta$ , the same vertex may remain the near vertex, and it may be connected to  $O(v)$  other vertices, which have to be tested every time that we consider each of the perhaps  $O(v)$  possibilities for the other active vertices. This problem

which if unsolved would increase the time complexity of our algorithm can be avoided as follows. For each choice of the far face, before we start carrying out the  $\theta$  rotations, we can determine just once all of the neighbours of the set of possible near vertices, and decide which one will be active over various ranges of  $\theta$ . Secondly, we can decide which of *its* neighbours will be the one which will determine the  $\phi$  limit as  $\theta$  changes. By removing these decisions from the inner loop, we can just update a pointer to this list of possibilities as  $\theta$  increases. There are obviously  $O(e)$  edges to consider in making this list; the list can be created in linear time on the assumption that we are dealing with a polyhedron, and so the vertices around a face, and the edges around a vertex are already sorted.

Similar ideas can be applied to the  $\phi$  limit for the possible far vertices.

Thus, both types of limit tests within the loop can be performed in order  $O(e)$  time, while we have a total number of faces  $O(f)$  to check as candidates for the far face. Thus, overall, our algorithm has quadratic running order  $O(fe)$  as claimed.

## References

- [1] B. S. Baker, S. J. Fortune, S. R. Mahaney. *Polygon Containment Under Translation*. J. Algorithms, **7**, 532-548, 1986.
- [2] B. Buchberger. *Applications of Gröbner Bases in Non-Linear Computational Geometry* Proc. Workshop on Scientific Software, Minneapolis, 1987. IMA Volumes in Mathematics and its Applications 14, Springer Verlag 1987.
- [3] D. R. Chand, S. S. Kapur. *An Algorithm For Convex Polytopes*. J. ACM, **17**, (1), 78-86, 1970.
- [4] B. Chazelle. *The Polygon Containment Problem*. Advances in Computing Research, **1**, 1-33, JAI Press, 1983.
- [5] H. Freeman, R. Shapira. *Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve*. Comm. ACM, **18**, (7), 409-413, 1975.
- [6] P. K. Ghosh, S. P. Mudur. *Theoretical Framework for Shape Representation and Analysis*. Proc. NATO ASI Fundamental Algorithms for Computer Graphics, Ed. R. A. Earnshaw, Springer Verlag, 1985.
- [7] P. K. Ghosh. *A Computational Theoretic Framework for Shape Representation and Analysis Using The Minkowski Addition and Decomposition Operators* Ph. D. Thesis, University of Bombay, 1986.
- [8] R. L. Graham, F. F. Yao. *Finding the Convex Hull of a Simple Polygon*. J. Algorithms **4**, 324-331, 1983.
- [9] L. Guibas, L. Ramshaw, J. Stolfi. *A Kinetic Framework for Computational Geometry*. IEEE 24th Annual Conference on the Foundations of Computer Science, COMPUS '83, 100-111, 1983.
- [10] L. Guibas, J. Stolfi. *Ruler, Compass and Computer - The Design and Analysis of Geometric Algorithms*. Proc. NATO ASI Theoretical Foundations of Computer Graphics and CAD, Lucca, Italy, July 1987.

- [11] A. E. Middleditch. *The Representation and Manipulation of Convex Polygons*. Proc. NATO ASI Theoretical Foundations of Computer Graphics and CAD, Lucca, Italy, July 1987.
- [12] F. P. Preparata, M. I. Shamos. *Computational Geometry*. Springer Verlag, New York, 1985.
- [13] J. O'Rourke. *Finding Minimal Enclosing Boxes* Int. J. Computer and Information Science, 14 (3) 183-199, 1985.
- [14] D. L. Souvaine. *Computational Geometry in a Curved World*. Princeton University, Department of Computer Science, Ph.D. Thesis, 1986.
- [15] G. T. Toussaint. *Solving Geometric Problems with the "Rotating Calipers"*. Proc. IEEE MELECON'83, Athens, Greece, 1983.