A Technique for Solving Geometric Optimization Problems in 2 and 3 Dimensions

by

Catherine M. Levinson

B. Sc. Computing Science, University of Western Ontario, 1977

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY

1990

ISBN 0-315-[illegible]565-X

Approval

Name: Catherine M. Levinson

Degree: Master of Science

Title of Thesis: A Technique for Solving Geometric Optimization Problems Problems in 2 and 3 Dimensions

Examining Committee:

Chairman: Dr. Joseph G. Peters

Senior Supervisor: Dr. Binay K. Bhattacharya

Dr. Robert D. Cameron

External Examiner: Dr. Kamal K. Gupta
Engineering Science, SFU

Date Approved: Aug. 15, 1990

Title of Thesis/Project/Extended Essay

A Technique for Solving Geometric Optimization Problems in 2 and 3 Dimensions.

Author: ______________________

(signature)

Catherine Mary Levinson

(name)

____

(date)

Abstract

One of the continuing challenges of computational geometry is to find efficient and practical solutions for computational problems. Often, well-known problem-solving techniques, like divide-and-conquer, plane-sweep and prune-and-search are used in the creation of new algorithms. In this thesis, a new problem-solving-technique called ICT (iterative-convergent-technique) is introduced. ICT is an approximation technique that geometrically converges upon the exact solution. It begins by constructing a convex region that encloses the solution. Each iteration, a fixed fraction (approximately one-half) of the remaining region is chopped away, until the approximation is guaranteed to lie within ε of the exact solution, where ε is a parameter specified by the user.

To demonstrate the usefulness of this approach, we have applied ICT to a number of geometric optimization problems in 2 and 3 dimensions:

- determining the separability of two planar sets ;
- detecting the common intersection of the convex hulls of m sets of points ;
- Linear Programming ;
- the problem of finding the smallest enclosing sphere of n weighted points .

To my parents and to Ed's Dad,

and to my husband and best friend.

Acknowledgements

I would like to express my thanks to my senior supervisor, Binay K. Bhattacharya, for encouraging my creativity and independence. In addition to suggesting this topic, he has cheerfully assisted me whenever difficulties arose. I feel that I have learned a great deal from him.

I would also like to express my thanks to Robert D. Cameron for the guidance he gave me during the year that Binay was on sabbatical. There are a number of other people who have played an integral role in the development of this thesis, and to all of these people I say thanks. In particular, I would like to thank Mike Dyck, Sanjeev Mahajan, Andrew Kurn, Ken Collins, Ranabir Gupta, Jutta Bauer, Armin Bruderlin, Daniel Say and Ed Levinson. Mike has demonstrated a great talent for computational geometry problems, and in particular for finding counter-examples. Sanjeev has not only given me encouragement and ideas, but has helped me to understand techniques described in the literature. Andrew took on the onerous task of editing my first draft of Chapter 2. I would like to thank Ken for his interest in the ICT solution to LP. His enthusiasm for LP in general made this topic much more interesting. I would like to thank Jutta Bauer for translating parts of [Blaschke 49] from German into English, and also I would like to thank Armin for checking this translation. I would like to thank Daniel Say from the SFU library for guiding me through a literature search on his own time. Finally I would like to thank my husband Ed, one of the main reasons that I returned to school. This thesis is the result of his encouragement and help.

Table Of Contents

List Of Figures

Notation Summary

$a, b, c, \ldots$		scalars
$\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$		points or vectors
$\mathbf{ab}$		The line segment connecting the points $\mathbf{a}$ and $\mathbf{b}$
$\alpha, \beta, \gamma, \ldots$		lines or planes
$A, B, C, \ldots$		finite disjoint sets of objects
$\Gamma, \Delta, \Phi, \ldots$		bounded, continuous sets of points, which are referred to as *regions*
distance, ...		functions (See Appendix B).
$\{ \ldots \}$	–	sets
$\cap$	–	set intersection
$\cup$	–	set union
$-$	–	set difference
$\subseteq$	–	subset
$\in$	–	element of
$\notin$	–	not an element of

Specific Conventions:

S	–	initial set of objects (input)
n	–	cardinality of S
$\mathbf{p}_i$	–	subscript i indicates the i^{th} element of a set
w	–	weight associated with each object in S. For example w_i represents the weight associated with the i^{th} object in S. The unweighted version of each problem is obtained by letting $w_i = 1$ for all i.
k	–	dimension of the problem or set
E^k	–	k-dimensional Euclidean space. We will label the axes x, y and z if working in E^3.
iff	–	if, and only if

Chapter 1

Introduction

One of the continuing challenges of computational geometry is to find efficient and practical solutions for computational problems. This emphasis on both efficiency and practicality has resulted in an interesting mix of research. At one end of the spectrum, researchers strive to create more efficient algorithms, where 'more efficient' may imply that the 'worst-case' time or space requirements of the solution have been reduced, or that the algorithm behaves well in all 'expected' cases. At the other end of the spectrum, the problems faced during the implementation of these algorithms are examined. This includes reconciling the differences between the hardware that the algorithms will be implemented on and the abstract computer models that were used by the designers of these algorithms. For example, researchers study different ways of representing real numbers on discrete hardware, and also ways of automatically detecting and handling degenerate cases.

In this thesis, a new problem-solving-technique called ICT (iterative-convergent-technique) is presented, where a *problem-solving-technique* is simply an approach to solving a problem that is effective for a number of different problems. Algorithm designers have frequently made use of such techniques when creating new algorithms. For example, in their survey of computational geometry, [Lee and Preparata 84] have identified and described several that have proven useful in this respect, including plane-sweep, divide-and-conquer and prune-and-search. Often a great deal of ingenuity is required to demonstrate that a given problem can be solved using a particular approach. Sometimes complex preprocessing steps are required or perhaps a geometrical property of the problem can be exploited. The payoff for this effort is that usually the new solution has the time and space complexity of the problem-solving-technique.

The technique that is presented in this thesis is an approximation technique. This means that the algorithm will generate solutions that are within ε units of an exact solution, where ε is a parameter specified by the user. Usually approximation techniques are faster than exact techniques, making them attractive when the exact solution to a problem is not required. To demonstrate the power of ICT, it will be applied to the following geometric optimization problems:

- Linear Programming (LP) in 2 and 3 dimensions ;
- the problem of finding the smallest enclosing sphere of n weighted points in 2 and 3 dimensions (SES) ;
- detecting the common intersection of the convex hulls of m sets of points .

Exact linear solutions already exist for most of the problems listed above. The linear-time solution for LP was independently proposed by [Megiddo 83a] and [Dyer 84] . [Megiddo 84] extended this result to obtain a linear-time solution for LP in any fixed dimension. The unweighted 2-dimensional SES problem was solved in linear time by [Megiddo 83a] while the linear-time solution for the weighted SES problem was described by [Dyer 86] . Detecting whether the convex hulls of two sets of points overlap is equivalent to determining whether the two sets are linearly separable. This can be determined by solving two LP problems of fixed dimension [Edelsbrunner 87] (page 213) . So far, if $m > 2$, it is not possible to determine whether the convex hulls of m sets of points overlap in linear time.

Since a theoretically optimal solution already exists for most of these problems, it is apt to question the wisdom of suggesting yet another solution, in particular one that guarantees only an approximate solution. There are a number of reasons for examining ICT in detail.

- ICT is worth exploring because it is an approximation technique. As was stated earlier, often the exact solution to a problem is not required. ICT allows the user to specify the amount of error that is acceptable in the solution. If little precision is required, then ICT can terminate early.
- Each of the linear-time results referenced above utilizes the well-known *prune-and-search* technique, which was first introduced (independently) in [Megiddo 83a] and [Dyer 84] . One of the steps of this technique is to find the median of a set of numbers in linear time ([Blum, Floyd, Pratt, Rivest and Tarjan 72] , [Schonhage, Paterson and Pippenger 76]) . However, as [Edelsbrunner 87] (page 239) has noted, the worst-case optimal methods for finding the median of a set of numbers all suffer from poor average case behaviour. Instead he suggests that the simpler algorithms presented in [Floyd and Rivest 75] be considered for implementation since they

determine the median in a fast expected time. If Edelsbrunner's advice is followed, then the solutions to the above algorithms are no longer linear in the worst-case.

- The most common approach for representing real numbers on a computer is to approximate them by fixed-precision floating point numbers. If such a representation is used, then each ICT algorithm will have a linear worst-case time-complexity.

- ICT has the following striking feature. Changing the dimension from 2 to 3 or changing the problem from an unweighted to a weighted one are simply variations of the same problem. Often a solution for one of these variations sheds light on how the other variations can be solved. As we shall see when the history of some of the problems listed above are discussed, such variations have been treated as totally separate problems, making it difficult, if not impossible, to extend a 2-dimensional algorithm to 3 dimensions or to convert an algorithm for an unweighted solution into one that can solve the weighted version of the problem.

- ICT can be easily combined with other iterative approaches, including the prune-and-search technique. Examples of how this can be done will be presented later in the thesis.

- There are some problems that can be solved using ICT for which there are no known linear solution. For example, this is true of detecting the common intersection of the convex hulls of m sets of points.

- Since problem-solving-techniques have proven to be useful algorithmic design tools, it is worth considering a new one on this merit alone. Also, the geometry that ICT is based upon is interesting for its own sake. Thus ICT is interesting from a strictly theoretical point of view.

ICT will be described in the next few sections, first in abstract terms to emphasize the concepts behind our approach, and then more concretely by using it to solve a problem. The notational conventions listed in Appendix A have been used throughout this thesis.

1.1 ICT, An Iterative-Convergent-Technique

A problem must have a geometrical interpretation before ICT can be applied to it. Usually the exact solution is either a point or it can be directly determined from a point, given the constraints of the problem. The task of an ICT algorithm is to determine the location of this point. It begins by isolating the point's location to a particular *region* (see Appendix B) of space. Let Φ_0 denote the initial solution region. In each iteration, more of the solution region is chopped away, until what remains is small enough for all of its points to lie within ε of the exact solution, where ε is a parameter that is specified by the user. The method of chopping is based upon the following observation. If Φ and Γ are two convex regions that contain the exact solution $\mathbf{s}$, then $\mathbf{s} \in \Phi \cap \Gamma$. In each iteration, a new convex set (Γ_i) is determined, such that Γ_i contains $\mathbf{s}$ and such that the volume of $\Phi_i = \Phi_{i-1} \cap \Gamma_i$ is no larger than a fixed fraction of the volume of the previous solution region, Φ_{i-1}. Thus ICT converges towards $\mathbf{s}$ through a sequence of nested convex regions whose volumes decrease in a geometric progression.

In summary, each ICT algorithm has three distinct components:

- an *initialization step* during which the initial solution region Φ_0 is constructed ;
- an *iteration sequence* during which Γ_i is used to chop away a portion of the solution region. That is, $\Phi_i = \Gamma_i \cap \Phi_{i-1}$.
- a *termination predicate* which determines whether the current solution region is small enough to terminate or whether another iteration is required.

Note that one of the basic tenets of this thesis is that each of these components should require at most linear time. In order to help illustrate the above ideas, ICT will be applied to a sample problem in the next two sections.

1.2 The Smallest Enclosing Circle Of n Points In The Plane (SEC)

Finding the Smallest Enclosing Circle (SEC) of n points in the plane is a classic geometric optimization problem which over the years has been known by a number of different names including: minimum spanning circle, Euclidean distance facility location and the Euclidean one center (point) problem. Figure 1.1 illustrates the SEC problem. Suppose that we have been given a set of n points in the plane and we have been asked to find the smallest circle that encloses these points, where we define a circle with center $\mathbf{c}$ and radius r as follows: $\mathit{Circle}(\mathbf{c}, r) = \{ \mathbf{x} \in E^2 \mid \mathit{Distance}(\mathbf{c}, \mathbf{x}) \le r \}$. $\mathit{Distance}(\mathbf{c}, \mathbf{x})$ is a function that returns the Euclidean distance between the points $\mathbf{c}$ and $\mathbf{x}$.

Formally, SEC can be described as follows. Let $S = \{ \mathbf{p}_1 , \mathbf{p}_2 , \dots , \mathbf{p}_n \}$ denote a set of n points in the plane. We want to find the point $\mathbf{c}^*$ that minimizes r^*, where:

$$r^* = \operatorname*{maximum}_{i=1}^{n} \mathcal{D}istance(\mathbf{c}^* , \mathbf{p}_i) .$$

(a) S, the set of points (b) the smallest circle that encloses S.

Figure 1.1 An example of a smallest enclosing circle.

The smallest enclosing circle has a number of well-known properties, including:

Property 1: The smallest enclosing circle is unique;

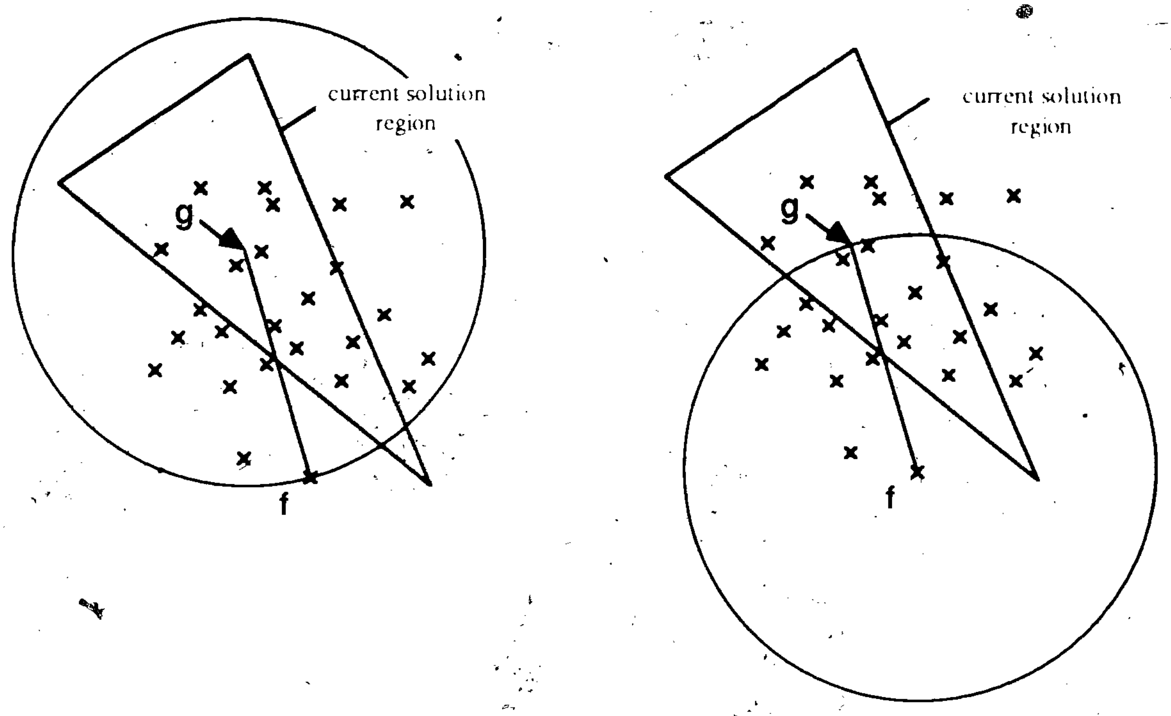
Property 2: Either two points of S define the endpoints of the diameter of the smallest enclosing circle, or three points of S form an acute triangle whose circumcircle is the smallest enclosing circle.

See Chapter 16 of [Rademacher and Toeplitz 57] for a proof of these two properties.

Property 2 states that $\mathbf{c}^*$ lies in the convex hull of at least two of the points of S. Therefore any bounding box that encloses the points of S will also enclose $\mathbf{c}^*$. We will make use of this knowledge when constructing the initial solution region. Now consider the problem of reducing the area of the solution region by a fixed fraction each iteration. Since the smallest enclosing circle is unique, any circle that encloses the points of S has a radius r such that, $r^* \le r$. Furthermore, it is well-known that

$$\mathbf{c}^* \in \bigcap_{\mathbf{p} \in S} Circle(\mathbf{p} , r) .$$

Let **g** denote a point that lies in the interior of the current solution region. Find the point $\mathbf{f} \in S$ that is furthest from **g**, and let $r = \mathit{Distance}(\mathbf{g}, \mathbf{f})$. It is easy to see that $\mathit{Circle}(\mathbf{g}, r)$ will enclose the points of S, as is shown in Figure 1.2.a. Therefore, it follows from the above that $\mathit{Circle}(\mathbf{f}, r)$, which is shown in Figure 1.2.b, encloses $\mathbf{c}^*$. Notice that **g** lies on the circumference of this circle.



(a) A circle that encloses S.

(b) A circle that encloses $\mathbf{c}^*$.

Figure 1.2 Part of the SEC iteration step

Now consider the half-plane whose boundary both passes through **g** and is tangent to $\mathit{Circle}(\mathbf{f}, r)$ (see Figure 1.3). Since this half-plane contains $\mathit{Circle}(\mathbf{f}, r)$, it must also contain $\mathbf{c}^*$. Thus we can construct the next solution region by intersecting the current solution region with this half-plane. Clearly the choice of **g** will affect the area of the region that is discarded. The following theorem will be used to guide our choice.

Winternitz's Theorem

A 2-dimensional convex figure is divided into two regions by a line that passes through its centre of gravity. The ratio of the area of these two regions always lies between the bounds $\frac{4}{5}$ and $\frac{5}{4}$ (inclusively).

See [Yaglom and Boltyanskii 61], page 160 for a proof of this theorem. Centre of gravity is defined in Appendix B. Each solution region is convex since it is constructed by intersecting half-planes. Therefore, by choosing **g** so that it coincides the the center of gravity of the current solution region, Winternitz's Theorem guarantees that the area of each successive region will be at most $\frac{5}{9}$ of the area of its predecessor, meeting the geometric reduction requirements of an ICT algorithm.

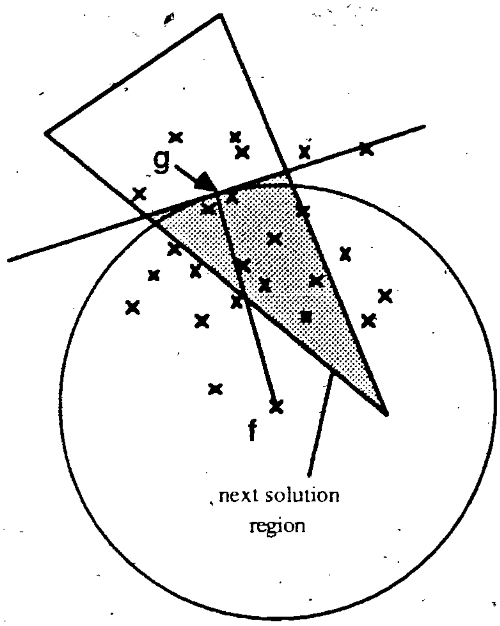


Figure 1.3 Reducing the area of the current solution region

Finally, consider the termination predicate. Suppose that $Circle(\mathbf{g}, r)$ is the approximate solution. There are two ways to interpret the termination criteria: either r should be within ε of r^* or else **g** should be within ε of $\mathbf{c}^*$, where ε is a parameter specified by the user. The basic idea behind the first criteria is to terminate once

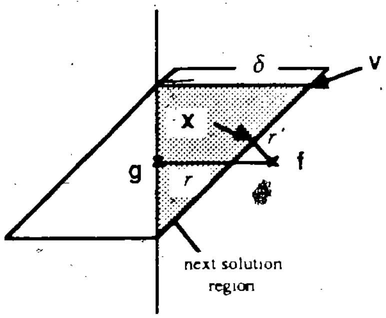
$$r - r^* < \varepsilon \;.$$

This can be achieved by finding a suitable $r' \leq r^*$ and terminating when $r - r' < \varepsilon$. Thus $r' < r^* \leq r$. There are several ways to find a suitable r'. For example, let **x** denote the point of the solution region that is closest to **f**. Clearly $Distance(\mathbf{x}, \mathbf{f}) \leq r^*$ since the solution region contains $\mathbf{c}^*$. Rather than finding $\mathbf{x}$ however, it will be easier to find the vertex $\mathbf{v}$ of the solution region that is of maximum (perpendicular) distance δ from the boundary of Γ_i. It is easy to see that

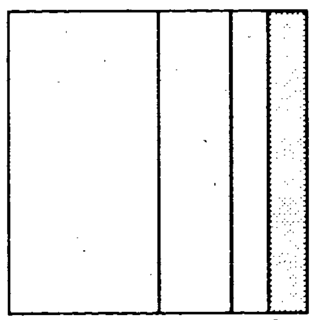
$$r - r' \leq \delta$$

(see Figure 1.4.a). Therefore, the algorithm will terminate once $\delta < \varepsilon$, since this ensures that $r - r' \leq \delta < \varepsilon$.

Now consider the second termination criteria. Clearly $\mathbf{g}$ is within ε of $\mathbf{c}^*$ if the current solution region is a subset of $Circle(\mathbf{g}, \varepsilon)$. This is easy to test – just make sure that each vertex of the solution region is within ε of $\mathbf{g}$. The difficulty is in ensuring that the solution region converges in all directions. For example, Figure 1.4.b illustrates four successive solution regions that are converging in the x-direction only. If this trend continues, then the solution region will never be a subset of $Circle(\mathbf{g}, \varepsilon)$. This problem is discussed further in Section 1.4.



(a) $r - r' < \varepsilon$



(b) an example of degenerate convergence

Figure 1.4 Terminating SEC

1.3 The ICT Algorithm For SEC

In this section, an ICT algorithm for finding the smallest enclosing circle of n points in the plane is described. It is assumed that the algorithm should terminate once r is within ε of r^*. (See Chapter 3 for an example of terminating once $\mathbf{g}$ is within ε of $\mathbf{c}^*$.) Before presenting the algorithm, some functions and definitions should be introduced. Let $COG(\Phi)$ denote a function that returns the center of gravity of the region Φ and let $Furthest(\mathbf{g}, S)$ denote a function that returns the point of S that is furthest from $\mathbf{g}$.

<u>Algorithm 1.1 : Finding the smallest enclosing circle of n points in the plane.</u>

1. **Initialization Step**

1.1 Let Φ_0 denote a bounding box for S ;

1.2 $r_0 := +\infty$;

2. **Iteration Step** ($i \geq 1$)

2.1 $\mathbf{g} := \mathcal{COG}(\Phi_{i-1})$;

2.2 $\mathbf{f} := \mathcal{Furthest}(\mathbf{g}, S)$;

2.3 $r_i := \mathcal{Minimum}(r_{i-1}, \mathcal{Distance}(\mathbf{g}, \mathbf{f}))$;

2.4 Let Γ_i denote the half-plane containing $\mathbf{f}$ whose boundary is tangent to $\mathcal{Circle}(\mathbf{f}, r_i)$ and perpendicular to the line segment $\mathbf{gf}$.

2.5 $\Phi_i := \Phi_{i-1} \cap \Gamma_i$;

3. **Termination Predicate**

3.1 Find the vertex $\mathbf{v}$ of Φ_i that is of maximum perpendicular distance from the boundary of Γ_i . Let δ denote this distance.

3.2 If $\delta < \varepsilon$

3.3 then { terminate reporting that $\mathcal{Circle}(\mathbf{g}, r_i)$ is the approximate solution }

3.4 else { continue to iterate. }

- end of algorithm -

Discussion and analysis of Algorithm 1.1

Let n denote the number of input points.

1. A rectilinear bounding box can be constructed in O(n) time. Therefore the initialization step requires O(n) time.

2. In each iteration the number of edges of the solution region will increase by at most one, due to the intersection of line 2.5 . Therefore, during the i^{th} iteration, the solution region will have O(i) edges.

3. The centre of gravity of Φ_{i-1} can be found in time linear to the number of edges of the region (see Section C.3) . Therefore line 2.1 can be performed in O(i) time.

4. Line 2.2 requires O(n) time since each point of S must be checked in order to find the one that is furthest from $\mathbf{g}$.

5. Line 2.3 may be a bit surprising. From the discussion in the previous section, one would expect it to be: r_i = *Distance*($\mathbf{g}$, $\mathbf{f}_i$). Of course this would be perfectly acceptable since the resulting area of Φ_i would be appropriately bounded. However, the goal of the above algorithm is to reduce the area of the solution region to the optimal point as fast as possible. By choosing r_i to be the minimum radius so far, we cut away even more of the solution region.

6. Clearly, Γ_i in line 2.4 can be determined in constant time, given $\mathbf{g}$, $\mathbf{f}$ and r_i .

7. The intersection $\Phi_i := \Phi_{i-1} \cap \Gamma_i$ can be performed in time linear to the number of edges of Φ_{i-1} (see Section C.4). Therefore line 2.5 can be performed in O(i) time .

8. Since the solution region is a convex polygon, it has the same number of vertices as edges. Clearly the distance between one vertex and the boundary of Γ_i can be determined in constant time (see [Bowyer and Woodwark 83] , page 107). Therefore the entire test can be performed in O(i) time.

9. There is an optimization that can be made to Algorithm 1.1 that has not been included for the sake of clarity. Each time a new minimum radius is discovered (line 2.3) , the edges of the solution region can be trimmed to reflect this new radius. For example, suppose that the edge e_j was added to the solution region during iteration j , where $j < i$. This edge can be trimmed from the solution region by intersecting the current solution region with a half-plane Γ_{ij} that contains $\mathbf{f}_j$, such that its boundary is parallel to that of Γ_j but which is a distance of r_i from the point $\mathbf{f}_j$ instead of r_j . (Note that $\Gamma_{ij} = \Gamma_{ij} \cap \Gamma_j$.) Γ_{ij} contains the exact solution since $r^* \le r_i$. Therefore this step will not discard the exact solution.

   The intersection routine (see Section C.4) can be customized for the trimming operation. The first step of this routine is to find a vertex of the solution region that does not lie in Γ_{ij} . Notice that either endpoint of e_j lies outside of Γ_{ij} , so there is no need to search for such a vertex during the trimming step. Starting from one such vertex, the intersection routine systematically traverses and deletes the edges of the solution region that lie outside of Γ_{ij} . Finally a new edge (e_{ij}) is added to the region in order to close the boundary of the polygon. Now consider the overall cost of the trimming step. Each edge can be added and deleted in O(1) time. Since at most O(i) edges can be deleted from the solution region and at most O(i) edges can be added to it, it is easy to see that the total running time for the trimming step is O(i) time. Therefore the trimming step further reduces the area of the solution region without increasing the asymptotical time-complexity of the algorithm. In addition, it helps to keep the solution region more 'centred' with respect to the furthest points.

Thus in summary,

- $O(n)$ time is required for the initialization step ;
- $O(\mathcal{M}aximum(i, n))$ time is required for the i^{th} iteration ;
- $O(i)$ time is required for the termination predicate during the i^{th} iteration.

Therefore, the total running time for Algorithm 1.1 is $O(t * \mathcal{M}aximum(n, t))$, where t is the total number of iterations performed by the Algorithm 1.1 . The size of t depends on ε and the area of the initial solution region. In the following it will be argued that the running time of Algorithm 1.1 is linear whenever fixed precision floating point numbers are used to approximate real numbers.

In most computer implementations, real numbers are approximated by a rational fraction limited to a certain fixed precision. This means that after a bounded number of iterations, say c_1, the area of the solution region will be less than the smallest discernable difference between two floating point numbers. If the algorithm has not already terminated, then at this point in time, the solution region will have been reduced to either a line segment or a single point (see Section C.4). If the vertices of the region are contained in the line that defines the boundary of Γ_i, then the algorithm will terminate, showing that t is bounded from above by c_1. However it is possible that the solution region has been reduced to a line segment that does not lie in the boundary of Γ_i. For example, this situation arises trivially when the initial solution region is a vertical line segment. The maximum length of this line segment is determined by the diameter of the initial solution region. When such a case arises, the algorithm will continue to iterate; each iteration the length of the line segment will be reduced by $\frac{1}{2}$ (see Section C.3). Thus after a bounded number of iterations, say c_2, the length of this line segment will be less than the smallest discernable difference between two floating point numbers. At this point the solution region will be reduced to a point and the algorithm will terminate. Thus, $t \leq c_1 + c_2$, where c_1 and c_2 are constants determined by the fixed precision of the real number representation.

Under the above assumptions, $t = O(1)$ since it is bounded from above by a constant. Furthermore, since it is expected that $t << n$, it is claimed that the running time of Algorithm 1.1 is $O(n)$.

1.4 Terminating ICT Algorithms

Although the termination of each ICT algorithm will be handled separately in this thesis, there are a few general comments that can be made. Suppose that $\mathbf{x}^*$ is the optimal solution of the problem and let $\mathbf{g}$ denote the centre of gravity of the current solution region. In this thesis, two methods of terminating ICT will be considered, which can be described loosely as follows:

(1) $\quad \left| \mathcal{F}(\mathbf{x}^*) - \mathcal{F}(\mathbf{g}) \right| < \varepsilon$

(2) $\quad \left| \mathbf{x}^* - \mathbf{g} \right| < \varepsilon$.

The meaning of these statements depends upon the problem being solved; their desirability depends upon the application. For example, if we consider the smallest enclosing circle problem studied in Section 1.2 , (1) refers to ensuring that the radius is within ε of r^* while (2) requires that $\mathbf{g}$ be within ε of $\mathbf{c}^*$. A type (1) termination involves finding an over- and underestimate of the optimal solution; when the two estimates are within ε of each other, then the algorithm can terminate. This type of termination was illustrated in Algorithm 1.1 . A type (2) termination requires that the solution region be a subset of *Circle*($\mathbf{g}$, ε) (or *Sphere*($\mathbf{g}$, ε) while solving a 3-dimensional problem). The problem of degenerate convergence arises only when a type (2) termination is required. Since this is the more difficult termination predicate to satisfy, the algorithms described in the rest of this thesis will consider this case only.

[Diaz and O'Rourke 89] have suggested an approach for handling degenerate convergence which may be applicable to ICT. Their approach involves finding the diameter of the solution region and splitting the region into two parts along this diameter. An iteration is then performed on both of the regions. A fixed fraction of both regions is cut away during the iteration, resulting in a fixed fraction of the total region being discarded. In addition, they show that for the problem of finding the centre of area of a convex polygon, this approach ensures that the diameter of the solution region converges to a point. It is likely that this property will also hold for ICT algorithms. However, since there exists no algorithm to compute the diameter of a convex polyhedron in linear time, this approach has not been pursued in this thesis. As was mentioned earlier, one of the basic tenets of this thesis is that each step of the algorithm should take at most linear time.

1.5 Other Related Work

ICT was inspired by an algorithm by [Castells and Melville 83], [Melville 85] which finds the smallest enclosing circle of a convex polygon. We can use Castells and Melville's algorithm to solve our problem by first finding the convex hull of S in O($n \log h$) time, where h is the number of points on the convex hull [Kirkpatrick and Seidel 86]. Let $H = \{ \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_h \}$ be the ordered set of points comprising the convex hull of S. Melville's algorithm differs from ours in the following ways.

- The initial solution region is constructed by intersecting h circles centered about each of the points of H. Normally this step would require O(h^2) time, but, because the points of a convex polygon are already sorted, Melville is able to achieve this step in O(h) time.

- The algorithm terminates once the area of the current solution region is less than the precision of the floating-point hardware being used. Since the amount of available precision is fixed under such conditions, Melville is able to bound the total number of iterations by a constant, leading to a linear running time.

ICT is an improvement over Castells and Melville's for a number of reasons. Firstly, the requirement that the original source points be sorted has been removed. This eliminates the need to determine the convex hull of the set of n points. Secondly, each iteration Castells and Melville's algorithm finds the intersection of n circles, while the ICT algorithm simply finds the intersection of a convex polygon and a half-space, where the polygon has O(i) edges during the i^{th} iteration. Therefore the cost of each of ICT iteration, at least initially, will be smaller than those of the other algorithm.

1.6 What Will Be Done In This Thesis?

In Chapter 2, Winternitz's 2-dimensional result will be extended to 3 dimensions, allowing ICT to be applied to 3-dimensional problems. In Chapter 3, ICT is applied to the problem of finding the smallest enclosing sphere of n weighted points in 3 dimensions. This is a generalization of the sample problem presented earlier in this chapter. In Chapter 4, the problem of detecting the common intersection of the convex hulls of m sets of points in 2 and 3 dimensions is examined. In Chapter 5, ICT is applied to Linear Programming in 2 and 3 dimensions. Finally, in Chapter 6 the conclusions are presented, along with some suggestions for future research.

As has already been mentioned, Appendix A describes the notational conventions used. In addition, Appendix B includes definitions of some of the mathematical terms that are used in the thesis. Some of the issues related to implementing ICT algorithms are discussed in Appendix C while Appendix D summarizes some of the functions that have been defined in this thesis.

Chapter 2

Extending Winternitz's Theorem To 3 Dimensions

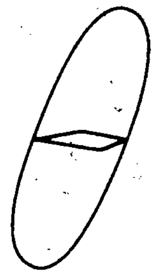
In section 1.2 we saw how the 2-dimensional version of Winternitz's theorem was used by ICT to solve the SEC problem. Similarly, 3-dimensional ICT problems require a 3-dimensional version of Winternitz's Theorem which is discussed in this chapter. Specifically, we show that, for a plane passing through the centre of gravity of a convex region,

$$\frac{27}{37} \leq r \leq \frac{37}{27},$$

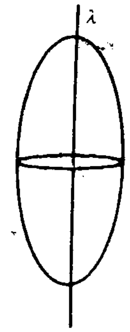
where r is the ratio of the volumes of the two regions determined by the plane. The crux of the argument is that for any 3-dimensional convex region, a right-angled cone can be constructed that has the same volume, such that, when partitioned by a plane passing through its centre of gravity, the ratio of the volumes of these two regions establishes the bounds for r. The construction of the cone involves several steps. First we apply the Schwarz construction to the original region, creating a region of the same volume that is axially symmetric about the z-axis. The cone is constructed from this symmetrical image. Schwarz's construction and the cone construction are described in the next two sections. In Section 2.3 we show that the right-angled cone establishes the range mentioned above. Finally, in Section 2.4, we present the proof of the theorem.

2.1 The Schwarz Construction

Given a closed convex region Φ and a line λ, the Schwarz construction constructs a closed convex region of equal volume, such that the new region is axially symmetric about λ [Blaschke 49]. Briefly, for every plane perpendicular to λ that intersects Φ, construct a closed circular disc about λ that is equal in area to the intersection of Φ and this plane. The constructed region is the union of these circular discs. (see Figure 2.1). We will refer to this new region as the *symmetrical image of Φ about λ*.



(a) Φ, the original convex region



(b) the symmetrical image of Φ about λ

Figure 2.1 An example of the Schwarz Construction.

Theorem 2.1 Let Ψ denote the symmetrical image of Φ about the z-axis. The centres of gravity of both Φ and Ψ lie in the same horizontal plane. Furthermore, the regions of Φ and Ψ that lie above this plane are equal in volume, as are the regions that lie below it.

Proof: Let ρ denote the horizontal plane that passes through the centre of gravity of Φ, and let Φ_1 denote the region of Φ that lies above ρ. The volume of Φ_1 is

$$\mathcal{V}(\Phi_1) = \int_{\Phi_1} dV = \int_{z_0}^{z_{max}} \mathcal{A}(z)\, dz , \qquad [2.1]$$

where $\mathcal{A}(z)$ is a function that returns the cross-sectional area of each differential slice, and the integration limits, z_0 and $z_{\max}$, refer to the z-coordinates of the lower and upper horizontal supporting planes, respectively, for Φ_1. Since ρ is perpendicular to the z-axis, the Schwarz construction partitions Φ_1 in exactly the same manner as the integration in equation [2.1] does. Therefore, $\mathcal{V}(\Phi_1) = \mathcal{V}(\Psi_1)$, where Ψ_1 is the region of Ψ that lies above ρ. Similarly, $\mathcal{V}(\Phi_2) = \mathcal{V}(\Psi_2)$, where Φ_2 and Ψ_2 denote the regions of Φ and Ψ that lie below the plane ρ. Now consider $\mathbf{g}$, the centre of gravity of Φ. By definition,

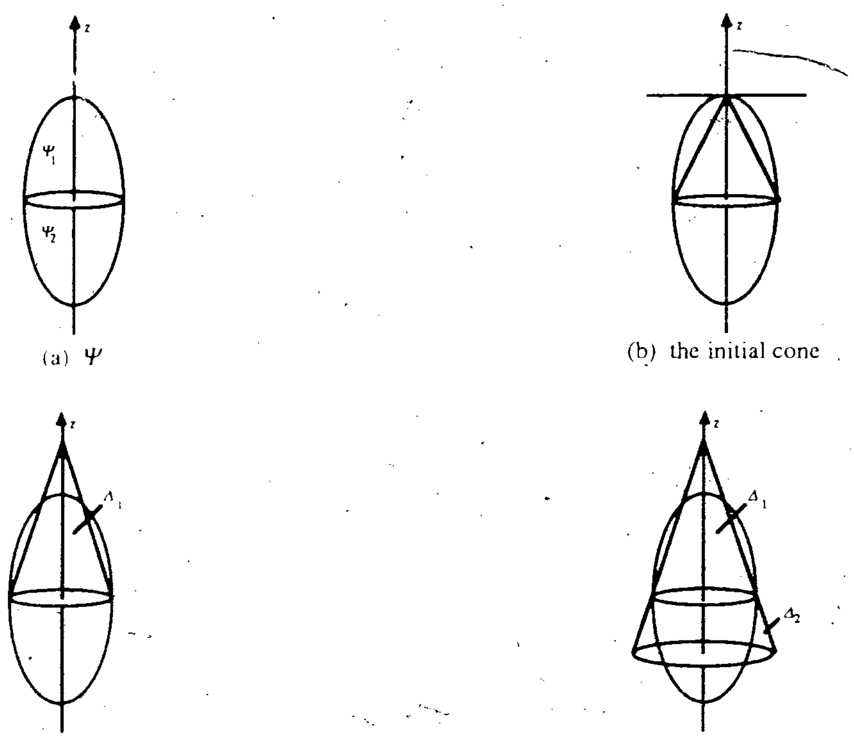
$$z_{\mathbf{g}} = \frac{\int_{z_{min}}^{z_{max}} z \; \mathcal{A}(z)\, dz}{\mathcal{V}(\Phi)}$$

where $z_{\min}$ is the z-coordinate of the lower horizontal supporting plane for Φ. The expression, $z\;\mathcal{A}(z)\,dz$, has the same value for corresponding differential slices of Φ and Ψ. Therefore $z_{\mathbf{g}}$

is the z-coordinate of the centre of gravity of Ψ, leading us to conclude that ρ passes through the centre of gravity of Ψ. ♦

2.2 The Cone Construction

Let Ψ denote a closed convex figure that is symmetric about the z-axis, and whose centre of gravity coincides with the origin. In this section, we will construct a (right-angled) cone that has the same volume as Ψ. First, we will construct a cone whose base is the intersection of Ψ and the plane $z = 0$ and whose apex is the point on the z-axis where Ψ is supported from above by a horizontal plane (Figure 2.2.b). The volume of this cone is less than or equal to the volume of Ψ_1, the upper part of Ψ. Now, gradually extend the apex of this cone up the z-axis, continuously increasing its volume, until it has the same volume as that of Ψ_1. Let Δ_1 denote this final cone (Figure 2.2.c). Finally, define a further extended cone Δ by extending the sides of Δ_1 downwards, shifting its base down the z-axis while keeping it perpendicular to the z-axis, until its volume is the same as that of Ψ. Let Δ_2 denote the region of the cone that lies below the plane $z = 0$ (Figure 2.2.d). Δ is the union of Δ_1 and Δ_2.



(a) Ψ (b) the initial cone

(c) $\mathcal{V}(\Delta_1) = \mathcal{V}(\Psi_1)$ (d) $\mathcal{V}(\Delta_2) = \mathcal{V}(\Psi_2)$

Figure 2.2 The cone construction technique

In the rest of this section, it will be argued that the centre of gravity of Δ does not lie below the centre of gravity of Ψ. First we show that the centre of gravity of Δ_1 does not lie below that of Ψ_1. Next, we will show that the centre of gravity of Δ_2 does not lie below that of Ψ_2. Finally we will combine these results to show that the centre of gravity of Δ does not lie below that of Ψ.

Lemma 2.2: The centre of gravity of Δ_1 does not lie below that of Ψ_1.

Proof: If Ψ_1 is a cone, then the two regions are identical, and hence have the same centre of gravity. Therefore, assume that Ψ_1 is not a cone. The union of Ψ_1 and Δ_1 can be partitioned into three regions: P_1, the set of points common to both Ψ_1 and Δ_1; P_2, the 'doughnut' or toroidal region that surrounds the cone and P_3, the points that lie solely in Δ_1 (Figure 2.3.a). Note that the intersection of P_2 and P_3 is a horizontal circle.

Let $\mathbf{g}_{\Psi_1}$, $\mathbf{g}_{\Delta_1}$, $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_3$ denote the centres of gravity of Ψ_1, Δ_1, P_1, P_2 and P_3, respectively. By Theorem B.2, the point $\mathbf{g}_{\Psi_1}$ lies on the line segment $\mathbf{g}_1\,\mathbf{g}_2$, dividing it in the ratio

$$\frac{\text{Length } (\mathbf{g}_1 \;\; \mathbf{g}_{\Psi_1})}{\text{Length } (\mathbf{g}_{\Psi_1} \;\; \mathbf{g}_2)} = \frac{\mathcal{V}(P_2)}{\mathcal{V}(P_1)}$$

Similarly, $\mathbf{g}_{\Delta_1}$ lies on the line segment connecting $\mathbf{g}_1$ and $\mathbf{g}_3$, dividing it in the same ratio since $\mathcal{V}(P_3) = \mathcal{V}(P_1)$. Therefore,

$$\frac{\text{Length } (\mathbf{g}_1 \;\; \mathbf{g}_{\Psi_1})}{\text{Length } (\mathbf{g}_{\Psi_1} \;\; \mathbf{g}_2)} = \frac{\text{Length } (\mathbf{g}_1 \;\; \mathbf{g}_{\Delta_1})}{\text{Length } (\mathbf{g}_{\Delta_1} \;\; \mathbf{g}_3)} \qquad [2.2]$$

Since each region is axially symmetric about the z-axis, the centres of gravity of these regions will lie on the z-axis. Let z_{Ψ_1}, z_{Δ_1}, z_1, z_2 and z_3 denote the z-coordinate of these points. We can rewrite equation [2.2] as follows:

$$\frac{\sqrt{(z_1 - z_{\Psi_1})^2}}{\sqrt{(z_{\Psi} - z_2)^2}} = \frac{\sqrt{(z_1 - z_{\Delta_1})^2}}{\sqrt{(z_{\Delta_1} - z_3)^2}}$$

$$\left| \frac{z_1 - z_{\Psi_1}}{z_{\Psi_1} - z_2} \right| = \left| \frac{z_1 - z_{\Delta_1}}{z_{\Delta_1} - z_3} \right|$$

$$-\frac{z_1 - z_{\Psi_1}}{z_{\Psi_1} - z_2} = -\frac{z_1 - z_{\Delta_1}}{z_{\Delta_1} - z_3} \qquad [2.3]$$

The last line follows because $\mathbf{g}_{\Psi_1}$ must lie between $\mathbf{g}_1$ and $\mathbf{g}_2$ and similarly, $\mathbf{g}_{\Delta_1}$ must lie between $\mathbf{g}_1$ and $\mathbf{g}_3$. Now translate the region $\Psi_1 \cup \Delta_1$ vertically so that z_1 coincides with the origin. This does not affect the relative positions of the above centres of gravity. Equation [2.3] now simplifies to

$$\frac{z_{\Psi_1}}{z_{\Psi_1} - z_2} = \frac{z_{\Delta_1}}{z_{\Delta_1} - z_3} \qquad [2.3]$$

or simply:

$$\frac{z_3}{z_2} = \frac{z_{\Delta_1}}{z_{\Psi_1}} \qquad [2.4]$$

Recall that, by construction, the regions P_2 and P_3 meet at a horizontal circle, and hence are separated by a horizontal plane. From this it follows that $\mathbf{g}_3$ does not lie below $\mathbf{g}_2$. To see this, consider the intersection of P_2 and the set of horizontal planes. This intersection partitions P_2 into a set of regions, each of which is radially symmetric about the z-axis and whose centre of gravity lies in the same horizontal plane. Since P_2 is the union of these regions, from Theorem B.2 we can conclude that $\mathbf{g}_2$ must lie on the line segment connecting the centres of gravity of the two regions that are extreme in the z-direction. Since all of P_2 lies below the horizontal plane that separates it from P_3, we can conclude that $\mathbf{g}_2$ does not lie above this plane. A similar argument can be used to show that $\mathbf{g}_3$ does not lie below this plane. Therefore we conclude that $\mathbf{g}_3$ does not lie below $\mathbf{g}_2$, and hence $z_3 \geq z_2$.

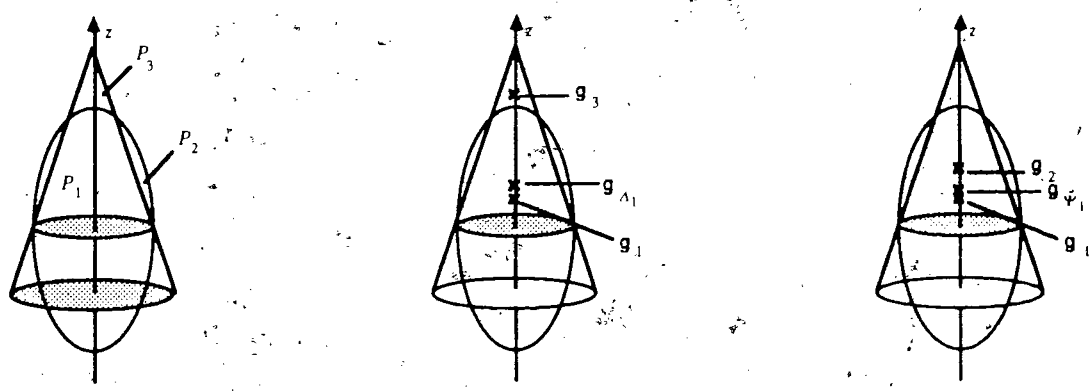
Now what remains is to show is that $\mathbf{g}_{\Delta_1}$ does not lie below $\mathbf{g}_{\Psi_1}$. (Recall that $\Psi_1 \cup \Delta_1$ have been translated vertically so that z_1 coincides with the origin.) There are three cases to consider:

(1) Suppose that z_2 and z_3 are both positive. This means that z_{Ψ_1} and z_{Δ_1} are both positive since they both lie on line segments whose one endpoint is the origin and whose other endpoint lies above the origin. Therefore, in order to satisfy equation [2.4], we conclude that $z_{\Delta_1} \geq z_{\Psi_1}$.

(2) Similarly, if z_2 and z_3 are both negative, then z_{Ψ_1} and z_{Δ_1} must both be negative, and again $z_{\Delta_1} \geq z_{\Psi_1}$ in order to satisfy equation [2.4].

(3) Finally, suppose that $z_2 \leq 0 \leq z_3$. By Theorem B.2, z_{Ψ_1} lies in the closed interval $[z_2 , 0]$. Similarly, z_{Δ_1} lies in $[0 , z_3]$. Therefore, once again, $z_{\Delta_1} \geq z_{\Psi_1}$.

Since in each case $z_{\Delta_1} \geq z_{\Psi_1}$, we conclude that $\mathbf{g}_{\Delta_1}$ does not lie below $\mathbf{g}_{\Psi_1}$. ♦



(a) partitioning $\Psi_1 \cup \Delta_1$ (b) calculating $\mathbf{g}_\Psi$ (c) calculating $\mathbf{g}_\Delta$

Figure 2.3 The centre of gravity of Δ_1 does not lie below that of Ψ_1

Lemma 2.3: The centre of gravity of Δ_2 does not lie below that of Ψ_2

The proof of this lemma is analogous to that of Lemma 2.2 and will not be repeated here. Note that a key point of this proof is that Δ_2 does not extend below Ψ_2.

Theorem 2.4: If $\mathbf{g}_\Delta$ and $\mathbf{g}_\Psi$ denote the respective centres of gravity of Δ and Ψ, then $\mathbf{g}_\Delta$ does not lie below $\mathbf{g}_\Psi$.

Proof: Let $s_1 = \mathcal{V}(\Psi_1) = \mathcal{V}(\Delta_1)$ and $s_2 = \mathcal{V}(\Psi_2) = \mathcal{V}(\Delta_2)$. By Theorem B.2, the point $\mathbf{g}_\Psi$ lies on the line segment $\mathbf{g}_{\Psi_1}$ and $\mathbf{g}_{\Psi_2}$, dividing it in the ratio:

$$\frac{\text{Length}(\mathbf{g}_{\Psi_1}\ \mathbf{g}_{\Psi})}{\text{Length}(\mathbf{g}_{\Psi}\ \mathbf{g}_{\Psi_2})} = \frac{s_2}{s_1} \quad [2.5]$$

Furthermore, the points $\mathbf{g}_{\Psi}$, $\mathbf{g}_{\Psi_1}$ and $\mathbf{g}_{\Psi_2}$ each lie on the z-axis. Therefore we can rewrite equation [2.5] as:

$$\frac{z_{\Psi_1} - z_{\Psi}}{z_{\Psi} - z_{\Psi_2}} = \frac{s_2}{s_1} . \quad [2.6]$$

(The absolute value signs are not needed since by construction, $z_{\Psi_2} \leq z_{\Psi} \leq z_{\Psi_1}$.) Rewriting equation [2.6] gives us:

$$z_{\Psi} = \frac{s_1\ z_{\Psi_1} + s_2\ z_{\Psi_2}}{s_1 + s_2} .$$

By a similar argument, we can show that:

$$z_{\Delta} = \frac{s_1\ z_{\Delta_1} + s_2\ z_{\Delta_2}}{s_1 + s_2} .$$

s_1 and s_2 are both positive since they denote volumes. In addition, $z_{\Psi_1} \leq z_{\Delta_1}$ (by Lemma 2.2) and $z_{\Psi_2} \leq z_{\Delta_2}$ (by Lemma 2.3) . Therefore,

$$s_1\ z_{\Psi_1} + s_2\ z_{\Psi_2} \leq s_1\ z_{\Delta_1} + s_2\ z_{\Delta_2} ,$$

implying that $z_{\Psi} \leq z_{\Delta}$. Therefore we conclude that $\mathbf{g}_{\Delta}$ does not lie below $\mathbf{g}_{\Psi}$. ♦

2.3 A Property of Right-Angled Cones

In this section, we will show that a plane parallel to the base of a right-angled cone and passing through its center of gravity partitions it into two regions such that the ratio of their volumes is $\frac{27}{37}$.

Theorem 2.5: Let Δ denote a right-angled cone that is partitioned into two regions by a plane that is parallel to the base of the cone and which passes through its centre of gravity. Let Δ_a denote the region of Δ that contains the apex and let Δ_b denote the other region. (See Figure 2.4) The ratio of the volumes of Δ_a and Δ_b is then

$$\frac{\mathcal{V}(\Delta_a)}{\mathcal{V}(\Delta_b)} = \frac{27}{37}$$

Proof: It is well-known that the volume of a right-angled cone is $\frac{1}{3}\pi r^2 h$, where r is the radius of the base and h is the height of the cone, and that the centre of gravity of a cone is $\frac{h}{4}$ from its base. Therefore Δ_a is a right-angled cone with height $h_a = \frac{3}{4}h$, radius $r_a = \frac{3}{4}r$ and thus, volume $\mathcal{V}(\Delta_a) = \frac{9}{64}\pi r^2 h$.

Hence,
$$\frac{\mathcal{V}(\Delta_a)}{\mathcal{V}(\Delta)} = \frac{\frac{9}{64}\pi r^2 h}{\frac{1}{3}\pi r^2 h} = \frac{27}{64},$$

and
$$\frac{\mathcal{V}(\Delta_b)}{\mathcal{V}(\Delta)} = 1 - \frac{27}{64} = \frac{37}{64}.$$

Therefore,
$$\frac{\mathcal{V}(\Delta_a)}{\mathcal{V}(\Delta_b)} = \frac{27}{37}.$$

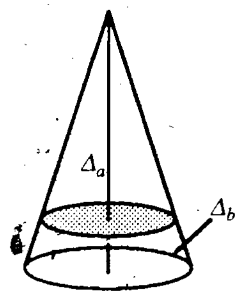


Figure 2.4 Partitioning the cone into two regions.

2.4 Proof of the Theorem

Finally, in this section we prove that Wintemitz's proof extends to 3 dimensions. The proof makes use of the results of the previous sections.

Theorem 2.6: Consider a 3-dimensional convex region Φ, which has been partitioned into two regions, Φ_1 and Φ_2, by a plane that passes through its centre of gravity. The ratio of the volumes of Φ_1 to Φ_2 obeys:

$$\frac{27}{37} \leq \frac{\mathcal{V}(\Phi_1)}{\mathcal{V}(\Phi_2)} \leq \frac{37}{27}.$$

Proof: Assume that $\mathcal{V}(\Phi_1) \leq \mathcal{V}(\Phi_2)$. Rotate the plane and Φ, such that the plane coincides with the $z = 0$ plane, and such that Φ_1 lies above it and Φ_2 lies below it.

(1) Construct Ψ by applying the Schwarz construction to Φ.

(2) Construct Δ by applying the cone construction to Ψ.

Recall that by construction, Δ is partitioned into two regions, Δ_1 and Δ_2, by the plane $z = 0$, such that Δ_1 lies above this plane and Δ_2 lies below it. Furthermore, $\mathcal{V}(\Delta_1) = \mathcal{V}(\Phi_1)$ and $\mathcal{V}(\Delta_2) = \mathcal{V}(\Phi_2)$. By Theorem 2.4, we know that the centre of gravity of Δ does not lie below the plane $z = 0$. A horizontal plane, γ, through the cone's centre of gravity partitions Δ into two regions, Δ_a and Δ_b, which respectively lie above and below γ. It is easy to see that

$$\mathcal{V}(\Delta_a) \leq \mathcal{V}(\Delta_1) \quad \text{and} \quad \mathcal{V}(\Delta_b) \geq \mathcal{V}(\Delta_2)$$

Thus, $\mathcal{V}(\Delta_a) \leq \mathcal{V}(\Phi_1)$ and $\mathcal{V}(\Delta_b) \geq \mathcal{V}(\Phi_2)$.

By Lemma 5, we know that $\dfrac{\mathcal{V}(\Delta_a)}{\mathcal{V}(\Delta_b)} = \dfrac{27}{37}$, and since by assumption, $\mathcal{V}(\Phi_1) \leq \mathcal{V}(\Phi_2)$, we conclude that

$$\frac{27}{37} = \frac{\mathcal{V}(\Delta_a)}{\mathcal{V}(\Delta_b)} \leq \frac{\mathcal{V}(\Phi_1)}{\mathcal{V}(\Phi_2)} \leq 1 .$$

Had we assumed $\mathcal{V}(\Phi_1) \geq \mathcal{V}(\Phi_2)$, we would have found by a similar argument that

$$1 \leq \frac{\mathcal{V}(\Phi_1)}{\mathcal{V}(\Phi_2)} \leq \frac{37}{27} .$$

Thus, without assumptions, we have $\dfrac{27}{37} \leq \dfrac{\mathcal{V}(\Phi_1)}{\mathcal{V}(\Phi_2)} \leq \dfrac{37}{27}$. ♦

Chapter 3

The Smallest Enclosing Sphere of n Weighted Points (SES)

Finding the smallest enclosing sphere (SES) of n weighted points in E^3 is a generalization of the smallest enclosing circle problem (SEC), which was discussed in detail in Sections 1.2 and 1.3. Since the intuitive discussion presented there extends directly to this problem, it will not be repeated here. One of the reasons for discussing this algorithm is to illustrate the ease with which a solution for an unweighted 2-dimensional problem can be modified to solve a weighted 3-dimensional version of the same problem. This extensibility is one of the strengths of the ICT approach. In addition, the first example of handling degenerate convergence is presented and discussed. We begin with a formal description of the problem, followed by a summary of some of the more recent history of both SEC and SES. Finally, the ICT algorithm for the weighted SES problem is presented and discussed.

Formally, let $S = \{ \mathbf{p}_i \mid i = 1, \ldots, n \}$ denote a set of points in E^3 and let w_i denote a weight associated with each point $\mathbf{p}_i$, such that $w_i \geq 0$. Finding the smallest enclosing sphere entails finding the point $\mathbf{c}^*$ that minimizes

$$\mathop{Maximum}_{i=1}^{n} \{ w_i \, Distance(\mathbf{c}^*, \mathbf{p}_i) \} .$$

The phrase *unweighted* will be used to distinguish problems for which each $w_i = 1$.

3.1 History Of SEC And SES

SEC is a well-studied problem, having been first introduced into the literature over one hundred years ago. In location theory, it is the minimax counterpart of the well-known Fermat problem [Francis and White 74].

The first published algorithm for solving the unweighted SEC problem was presented in [Sylvester 1857, 1860] and [Chrystal 1885]. This algorithm, which has come to be known as the Chrystal-Pierce algorithm, converges on the optimal solution by constructing a sequence of enclosing circles with decreasing radii. At least one point of S is discarded each iteration, leading to a worst case running time of $O(n^2)$. The expected running time for this algorithm is dependent upon the selection of the initial enclosing circle. Different initialization steps have been suggested by [Nair and Chandrasekaran 71] and by [Chakraborty and Chaudhuri 81]. [Hearn and Vijay 82] have reported that the initialization procedure described by [Chakraborty and Chaudhuri 81] seems to provide the best empirical results.

[Elzinga and Hearn 72a] have taken a different approach to solving the unweighted SEC problem. Rather than starting with a large circle that encloses all of S, they start with a circle that has a radius that is less than or equal to that of the optimal solution, converging upon the optimal solution through a sequence of circles with monotonically increasing radii. [Hearn and Vijay 82] have reported that the worst case running time for this algorithm is $O(h^3 n)$, where h is the number of vertices of the convex hull of S. Empirically they found that the algorithm has an $O(n)$ running time for randomized data.

[Elzinga and Hearn 72b] have presented two algorithms for solving unweighted SES (of any dimension). They have shown that the optimal solution for the k-dimensional problem is both unique and can be expressed as the convex combination of at most $k+1$ points of S. Their first algorithm transforms the original convex programming problem into an equivalent quadratic programming dual problem, solving it by using the Simplex method for quadratic programming in a finite number of steps. Their second algorithm is a generalization of the approach used by [Elzinga and Hearn 72a]. That is, the algorithm converges on the optimal solution by constructing a sequence of spheres with monotonically increasing radii. Since only a finite number of such spheres can be constructed, the algorithm terminates in finite time.

[Shamos and Hoey 75] have solved the unweighted SEC problem in $O(n \log n)$ time by making use of the Furthest Point Voronoi Diagram (FPVD). In 2 dimensions, the FPVD is a planar graph that partitions the plane into a set of convex regions, one region for each point of the convex hull of S. Each vertex of the FPVD is equally-distant from at least three points of S. Furthermore, a circle centered at a vertex of the FPVD whose radius is the same as the distance between the vertex and one of its defining points is an enclosing circle for S. Their algorithm begins by finding the diameter of S in

O($n \log n$) time. If the circle defined by this diameter does not enclose the points of S, then the FPVD is constructed in O($n \log n$) time. Each of the O(n) vertices of the FPVD are then checked to see which has the nearest defining points and hence is the radius of the smallest enclosing circle. The original algorithm proposed by [Shamos and Hoey 75] was incorrect in that it did not find the diameter of set initially. The requirement for this step has been described by [Bhattacharya and Toussaint 85].

[Hearn and Vijay 82] have solved the weighted SEC problem by extending both the Elzinga-and-Hearn and the Chrystal-Pierce algorithms mentioned above. They have reported that empirical testing of both of these algorithms, along with a third algorithm by [Jacobson 81] revealed that the weighted Elzinga-and-Hearn algorithm out-performed the other two algorithms substantially.

[Megiddo 83a] has presented a linear time solution for the unweighted SEC problem that utilizes a technique that has come to be known as the *prune-and-search* technique (see Section 5.6). Each iteration a fixed fraction of the source points are discarded, leading to the linear time result.

[Megiddo 83b] has used presented a parallel algorithm for solving the weighted SEC problem in O($n (\log n)^3 (\log \log n)^2$) time, using a total of O($n (\log n)^2$) processors.

The algorithm presented by [Castells and Melville 83] and [Melville 85] for solving unweighted SEC has already been discussed in Section 1.5.

[Dyer 86] has presented an algorithm that solves the weighted SES problem in any fixed dimension in linear time. Dyer begins by linearizing the problem, transforming it to a ($k + 1$)-dimensional problem by adding a non-linear constraint. He then applies the prune-and-search technique to solve the problem in O($3^{(k+1)^2} n$).

[Oommen 87] has presented a variation of the Chrystal-Pierce algorithm which solves the unweighted SEC problem by optimizing the next circle to be used in the sequence of enclosing circles. He has reported that some very good empirical results have been achieved as a result of this optimization.

3.2 The ICT Algorithm For SES

In this section, ICT is applied to the weighted SES problem. It is shown that ICT can be used to optimize a convex function without transforming it into a problem of one higher dimension, as was done by [Dyer 86]. The algorithm presented below is almost identical to Algorithm 1.1, which solves the unweighted SEC problem. It differs in that a weighted distance is accommodated, and a different termination predicate has been implemented. Recall that Algorithm 1.1 terminated once r is within ε of r^*. Since this type of termination easily extends to the 3-dimensional weighted problem, it will not be repeated here. Instead the following algorithm terminates once $\mathbf{g}$ is within ε of $\mathbf{c}^*$, where ε is a user-specified parameter and $\mathbf{c}^*$ is the centre of the optimal solution. (This accounts for the addition of lines 2.5 to 2.7 below). Recall from Section 1.2 that in this case, degenerate convergence must be both detected and handled.

Let $COG(\Phi)$ denote a function that returns the center of gravity of the region Φ and let $f(\mathbf{g})$ denote the index of the point in S that is farthest (has the greatest weighted distance) from $\mathbf{g}$.

That is, $w_{f(c)}\ \mathcal{D}istance(\mathbf{c}, \mathbf{p}_{f(c)}) := \underset{i=1}{\overset{n}{\mathcal{M}aximum}} \{ w_i\ \mathcal{D}istance(\mathbf{c}, \mathbf{p}_i) \}$.

Algorithm 3.1 : Finding the smallest enclosing sphere of n weighted points in E^3

1. **Initialization Step**
   - 1.1 Let Φ_0 denote a bounding box for S;
   - 1.2 $r := +\infty$;
2. **Iteration Step** ($i \geq 1$)
   - 2.1 $\mathbf{g} := COG(\Phi_{i-1})$;
   - 2.2 $j := f(\mathbf{g})$;
   - 2.3 $r := \mathcal{M}inimum(r, w_j\ \mathcal{D}istance(\mathbf{g}, \mathbf{p}_j))$;
   - 2.4 Let Γ denote the half-space containing $\mathbf{p}_j$ such that the boundary of Γ is tangent to $\mathcal{S}phere\left(\mathbf{p}_j, \frac{r}{w_j}\right)$ and perpendicular to the line segment $\mathbf{g}.\mathbf{p}_j$.
   - 2.5 If (the dimension of $\Phi_{i-1} < 3$) and (Φ_{i-1} lies in the boundary plane of Γ)
   - 2.6 then { set Φ_i to the single point $\mathbf{g}$ }
   - 2.7 else { $\Phi_i := \Phi_{i-1} \cap \Gamma$ }
3. **Termination Predicate**
   - 3.1 Find the vertex $\mathbf{v}$ of Φ_i that is farthest from $\mathbf{g}$.
   - 3.2 If $\mathcal{D}istance(\mathbf{g}, \mathbf{v}) < \varepsilon$
   - 3.3 then { terminate reporting that $\mathcal{S}phere(\mathbf{g}, r)$ is the approximate solution }
   - 3.4 else { continue to iterate. }

– end of algorithm –

3.2.1 Discussion And Analysis Of Algorithm 3.1 (SES)

Let n denote the number of points in S.

1. Since $\mathbf{c}^*$ lies in the convex hull of from two to four points of S, it follows that it is contained in a rectilinear bounding box that encloses the points of S. Such a box can be constructed in $O(n)$ time. Therefore the initialization step requires $O(n)$ time.

2. Now consider the number of faces of the solution region. Φ_0 will have at most six faces. Each iteration, the intersection on line 2.7 will increase the number of faces by at most one. Therefore the solution region will have $O(i)$ faces during iteration i.

3. The centre of gravity of Φ_{i-1} can be found in time linear to the number of faces of the region (see Section C.3). Therefore line 2.1 can be performed in $O(i)$ time.

4. Line 2.2 requires $O(n)$ time since each point of S must be checked in order to find the one that is the furthest weighted distance from $\mathbf{g}$.

5. Notice that r on line 2.3 records the minimum weighted distance encountered so far. This weighted distance is converted to an unweighted one in order to construct Γ on line 2.5. Γ can be determined in constant time, given $\mathbf{g}$, $\mathbf{p}_j$, w_j and r.

6. Since $r \geq r^*$ and since each point on the surface of $Sphere\left(\mathbf{p}_j, \frac{r}{w_j}\right)$ is a weighted distance of r from $\mathbf{p}_j$, it follows that $\mathbf{c}^*$ is enclosed by this sphere, and also by Γ. Therefore the intersection on line 2.7 will not discard the optimal solution.

7. Line 2.5 tests for degenerate convergence. Degenerate convergence arises when the solution region does not converge in all possible directions. That is, instead of converging to a point, the solution region converges upon either a convex polygon or line segment that is not contained in $Sphere(\mathbf{g}, \varepsilon)$. In such a case, Algorithm 3.1 continues to iterate with a solution region that has a lower dimension. Recasting of the solution region to a lower dimension is automatically handled by the intersection routine (Section C.4). Furthermore, the determination of the centre of gravity of the region is based upon the dimension of the region, not the dimension of the problem (Section C.3). Therefore as long as we can ensure that a fixed fraction of the remaining solution region is cut away each iteration regardless of the dimension of this region, then degenerate convergence is not a problem. In the worst case, the solution region converges to a convex polygon, next to a line

segment, and finally to a point. After some time it is contained within $Sphere(\mathbf{g}, \varepsilon)$ and the algorithm terminates.

A fixed fraction of the remaining solution region is discarded as long as Φ_{i-1} is not contained in the plane that defines the boundary of Γ (for example, see Figure 3.1) . Now suppose that Φ_{i-1} is completely contained in the boundary plane of Γ . This means that Φ_{i-1} is tangent to $Sphere\left(\mathbf{p}_j , \frac{r}{w_j}\right)$ at the point $\mathbf{g}$. Thus $\mathbf{g}$ is the optimal solution and the algorithm terminates. This is signalled to the termination predicate by setting Φ_i to the single point $\mathbf{g}$ on line 2.6 .

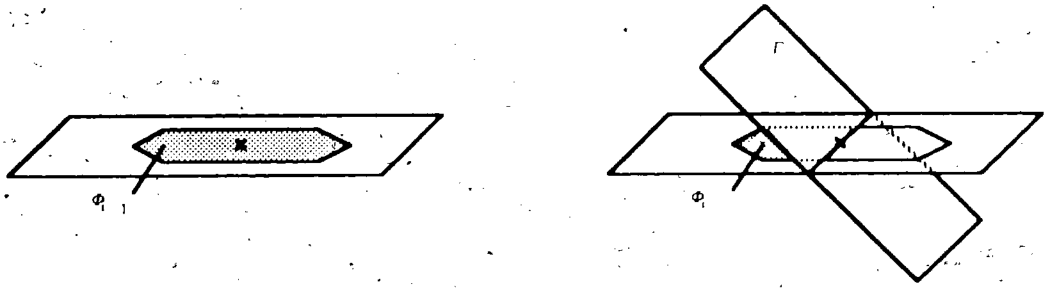
8. The dimension of Φ_{i-1} can be determined in $O(1)$ time (Section C.2) . Also, it can be determined whether Φ_{i-1} is contained in the boundary of Γ in constant time, since the data structure used to represent a 2-dimensional solution region also records the plane that the region lies in (Section C.2) . The intersection $\Phi_i := \Phi_{i-1} \cap \Gamma$ on line 2.7 can be performed in time linear to the number of faces of Φ_{i-1} (Section C.4) . Therefore the total cost of lines 2.5 to 2.7 is $O(i)$ time.

9. Since $\mathbf{g}$ either lies on the boundary of Γ or else is exterior to it, it follows that the current solution region will be reduced by a fixed fraction by the intersection on line 2.7 . If Φ_{i-1} is 3-dimensional, it follows from Theorem 2.6 that at least $\frac{27}{64}$ of the volume of Φ_{i-1} is discarded. If Φ_{i-1} is 2-dimensional, then Wintermitz's Theorem guarantees that at least $\frac{4}{9}$ of the area of Φ_{i-1} is discarded. If Φ_{i-1} is 1-dimensional, then half of the line segment will be thrown away.

10. Since the number of vertices of a convex polyhedron is linearly related to the number of faces of the region, the termination predicate requires $O(i)$ time.

11. As with Algorithm 1.1 , a trimming step can be added to the iteration step of the above algorithm. (See (9) of the discussion and analysis of Algorithm 1.1 in Section 1.3 .) This step has not been included for the sake of clarity. In 2 dimensions, all the the edges can be trimmed to reflect the new minimum radius in linear time. However, this same operation requires $O(i^2)$ time in the worst case in 3 dimensions (for example, consider the case where the solution region is a pyramid) . Therefore, instead of trimming all the edges of the solution region, it is suggested that Algorithm 3.1 keep track of the last three 'furthest' points that have been encountered along with the edges defined by these points. Each time a new minimum radius is discovered (line 2.3) the last three edges added would be trimmed to reflect the new minimum radius. Thus the solution region would be cut by a maximum of four half-spaces each iteration, and hence the operation can be performed in linear time. (Four half-spaces have been suggested since the $\mathbf{c}^*$ is a weighted distance of r^* from between two to four

points of S .) The advantage of this step is that it would further reduce the solution region without increasing the asymptotical time-complexity of the algorithm. It addition, it should also help to keep the solution region more 'centred'.



(a) Φ_{i-1} is 2-dimensional

(b) Φ_{i-1} does not lie in the boundary Γ

Figure 3.1 A fixed fraction of the current solution region is discarded.

Thus, in summary,

- O(n) time is required for the initialization step ;
- O($\mathcal{Maximum}(i , n)$) time is required for the i^{th} iteration ;
- O(i) time is required for the termination predicate during the i^{th} iteration.

Therefore, the total running time for Algorithm 3.1 is O($t * \mathcal{Maximum}(n , t)$) , where t is the total number of iterations performed by the Algorithm 3.1 .

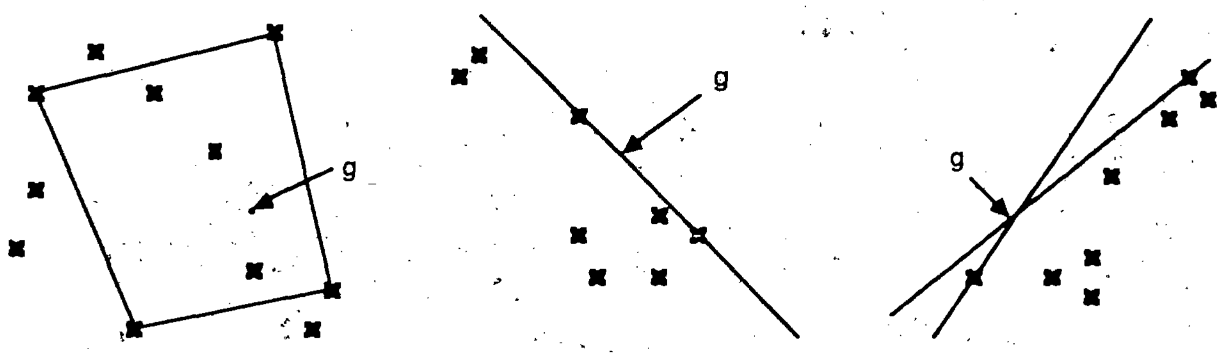
As was the case for Algorithm 1.1 , the size of t depends on ε and the area of the initial solution region. Recall that the running time of Algorithm 1.1 is linear provided that fixed precision floating point numbers are used to approximate real numbers (Section 1.3) . The same argument can be extended to show that Algorithm 3.1 is linear under this same condition. It was mentioned in (7) above that, in the worst case, the solution region converges to a convex polygon, next to a line segment, and finally to a point. The maximum area of the polygon along with the maximum length of this line segment can be determined from the initial solution region. Using an argument analogous to the one presented in Section 1.3 , three constants can be defined, c_1 , c_2 and c_3 , which respectively represent the maximum number of iterations required to reduce the solution region to a convex polygon, a line segment and finally to a point. Since $t \le c_1 + c_2 + c_3$, and since it is expected that $t << n$, we claim that the worst case time-complexity of Algorithm 3.1 is O(n) when fixed precision floating point numbers are used to approximate real numbers.

Chapter 4

Testing The Separability Of Sets Of Points

Suppose that we have been given m sets of points and have been asked to detect whether or not their m convex hulls share a common point. If $m = 2$, then the answer can be obtained in linear time by solving two linear programs [Edelsbrunner 87] (page 213). (Recall that linear programming of fixed dimension can be solved in linear time using the approach introduced independently by [Megiddo 83a] and [Dyer 84] .) If $m > 2$, then pairs of point sets would have to be compared and linearity is lost if we use the algorithm for the case $m = 2$. Alternately, the problem can be solved by finding the convex hull of each of the sets and then detecting whether the convex hulls overlap or not. The convex hull of a set can be found in $O(n \log h)$ time, where n is the number of points of the set and h is the number of vertices of the convex hull [Kirkpatrick and Seidel 86]. [Chazelle and Dobkin 87] have shown that it is possible to detect the overlap of two convex polygons in $O(\log n)$ time, while $O(\log^3 n)$ is required to detect the overlap of two convex polyhedrons. [Reichling 88] has extended their 2-dimensional result, showing that it is possible to detect whether m convex r-gons overlap in $O(m \log^2 r)$ time. In this chapter, an ICT algorithm is presented that does not need to construct the m convex hulls in order to determine if they share a point in common. This is of interest since improvements in speed are often obtained by eliminating unnecessary information. We believe the ICT solution will be very fast since each iteration approximately one half of the remaining solution region is discarded. If the convex hulls of the sets do overlap, then a point that is common to all of these convex hulls is reported. (Note that this point does not have to be an element of any of the given sets.) If the convex hulls of at least two of the sets do not overlap, then the algorithm terminates, reporting that there is no such common point.

Before the main problem of this chapter can be solved however, a technique for detecting whether a point lies in the convex hull of a sets of points S must be developed. This is sometimes referred to as the extreme point problem and has been solved in linear time using linear programming [Megiddo 83a]. In this thesis the extreme point problem will be solved by transforming it to a separability problem of one less dimension. Actually a slightly harder problem is solved – the algorithm distinguishes between points that lie interior, on the boundary or exterior to the convex hull of the set. Furthermore, information that supports this decision is returned to the calling routine. For example, suppose that **g** is the point being tested and S is a 3-dimensional set of points. If **g** lies in the interior of the convex hull of S then a maximum of $4(k-1)$ points of S are returned such that **g** also lies in the convex hull of this subset. This information will be used in Chapter 5 to construct the initial solution region for linear programming (LP) problem. If **g** lies on the boundary of the convex hull, then a half-space that contains S and whose boundary supports S at **g** is returned. If **g** lies exterior to the convex hull, then two half-spaces are returned – the boundary of each half-space supports S and passes through the point **g**. Furthermore, the intersection of the two half-spaces defines a wedge that contains the points of S. The wedge and half-space information will be used to reduce the current solution region. Examples of these cases are illustrated in Figure 4.1.



(a) **g** is interior to the convex hull (b) **g** lies on the boundary (c) **g** is exterior to the convex hull

Figure 4.1 Illustrating the information returned by PointInSet2D.

The approach used to solve the extreme point problem is similar to one suggested in [Megiddo 83a] (Appendix C) for solving the planar version of this problem. Figure 4.2 illustrates the hierarchy of routines that will be discussed in this chapter. 'PointInSet kD', ($k \leq 3$) is discussed in Section 4.1. This routine solves the extreme point problem by transforming it into a $(k-1)$-dimensional separability problem. The transformed problem, which is solved by 'SetSet kD' ($k \leq 2$), is discussed in Section 4.2. This is the routine that identifies and returns most of the supporting information illustrated by Figure 4.1. In fact a very tight coupling exists between the routines shown in Figure 4.2 – the supporting information gathered by SetSet1D is eventually used to construct the supporting information returned by PointInSet3D. (SetSet1D and SetSet2D are described in Section 4.2.1 and 4.2.2 respectively.) SetSet2D is an ICT algorithm which calls PointInSet2D twice each iteration. The routine main in Figure 4.2 refers to Algorithm 4.4 and is described in Section 4.3. This is the routine that determines whether the convex hull of m sets of points overlap or not.

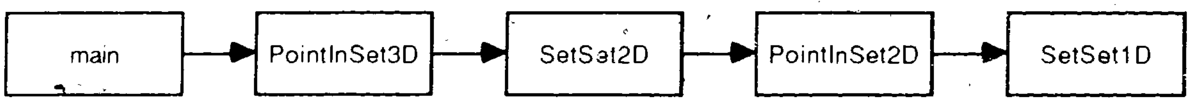


Figure 4.2 The hierarchy of the routines discussed in this chapter.

4.1 The Point-In-Set Problem

In this section, the separability of a point from a set of points is determined (the *point-in-set* problem). Let S denote a set of points and let $\mathcal{CH}(S)$ denote the convex hull of S. A point is said to be *strictly separable* from S if it is exterior to $\mathcal{CH}(S)$, *weakly separable* if it lies on its boundary and *inseparable* if it lies in the interior of $\mathcal{CH}(S)$. The point-in-set problem will be solved by transforming it into the problem of determining the separability of two sets of one less dimension (the *set-set* problem). Since this approach applies equally well to both 2 and 3-dimensional problems, only the 3-dimensional problem will be considered.

Two planar sets, S_1 and S_2, are said to be *strictly separable* if there exists a line such that both of the open half-planes defined by this line contains one of the sets but not the other. Similarly, S_1 and S_2, are said to be *weakly separable* if they are not strictly separable, but there exists a line such that each of the closed half-planes defined by this line contains one of the sets but not the other. S_1 and S_2 are said to be *inseparable* if they are neither weakly nor strictly separable. The set-set problem arises in pattern recognition. For example, see [Duda and Hart 73] (page 138) and [Jozwik 83]. [Dobkin and Reiss 80] have solved the set-set problem by first constructing a point-in-set problem; the constructed problem is then solved using linear programming. Although the constructed set is of the same dimension as the original two sets, its cardinality is $(n_1 \ n_2)$, where n_1 and n_2 denotes the cardinality of the two original sets. Thus it is possible for their approach to significantly increase the size of the problem.

We will begin by describing the mapping of a 3-dimensional set S to two planar sets, P_A and P_B. Without loss of generality, assume that the point to be tested coincides with the origin. First, partition the points of S into three sets, S_A, S_B and S_O, depending upon whether the point lies *above*, *below* or *on* the plane $z = 0$, respectively. For now, assume that S_O is empty.

Lemma 4.1 Given a plane that passes through the origin, points that lie in one half-space defined by this plane will be mapped to the other half-space under reflection about the origin. Correspondingly, points that lie on the plane will be mapped to points that lie on the plane.

Second, radially project each point of S_A and S_B onto the plane $z = 1$. That is, map each point $\mathbf{p}$ to the point where the line passing through the origin and $\mathbf{p}$ intersects the plane $z = 1$. Notice that the points of S_A will not be reflected through the origin by this projection, but the points of S_B will be. Let P_A and P_B denote the projected image of S_A and S_B respectively.

The following theorem states that determining the separability of the origin from S is equivalent to determining the separability of P_A and P_B. That is, the original point-in-set problem can be solved by transforming it into a set-set problem of one less dimension.

Theorem 4.2

(1) the origin is strictly separable from S iff P_A is strictly separable from P_B;

(2) the origin is weakly separable from S iff P_A is weakly separable from P_B;

(3) the origin is inseparable from S iff P_A is inseparable from P_B.

Proof: First consider the case where all the points of S lie to one side of the plane $z = 0$. Clearly S is strictly separable from the origin and since either P_A or P_B is empty, P_A is strictly separable from P_B in a trivial way. From now on assume that both S_A and S_B are non-empty.

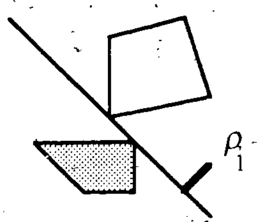
6

(1) Assume that the origin is strictly separable from S. By definition, there exists a plane ρ (different from $z = 0$), which passes through the origin and has S lying to one side of it. This implies that S_A and S_B both lie to the same side of ρ, and hence, from Lemma 4.1, it follows that P_B will lie on the opposite side of ρ, and hence that P_A and P_B will lie on opposite sides of the line determined by the intersection of ρ and the plane $z = 1$. This proves that the two sets are strictly separable. Now assume that P_A is strictly separable from P_B. By definition, there exists a line in the plane $z = 1$ such that P_A lies to one side of it and P_B lies to the other. There is a plane through this line and the origin, such that, all the points of S_A lie on the same side as those of S_A while all the points of S_B lie on the opposite side as those of P_B. Hence, S lies to one side of the plane defined by the origin and this line. Thus S is strictly separable from the origin whenever P_A is strictly separable from P_B.

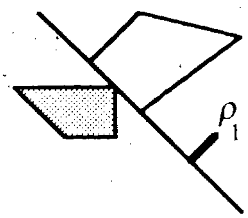
(2) Now assume that the origin is weakly separable from S and let ρ denote a supporting plane of S that passes through the origin. Let ρ_1 denote the line formed by the intersection of ρ and the plane $z = 1$. Here some of the points of S lie on ρ while the rest lie to one side of it. As in the previous case, the points that do not lie on ρ will be mapped to two planar sets, separated by the line ρ_1; the rest of the points will be mapped onto the line ρ_1 (by Lemma 4.1). Therefore, if the convex hull of P_A, $\mathcal{CH}(P_A)$, and $\mathcal{CH}(P_B)$ are to intersect at all, they must do so along the line ρ_1. (See Figure 4.3.) Recall that, by assumption, the origin lies on the boundary of $\mathcal{CH}(S)$. Since it has been assumed that S_O is empty, the origin must lie on either an edge or a face of $\mathcal{CH}(S)$. Consider the vertices of this edge or face. If the origin lies on a line passing through two of the vertices, then one of these points must belong to S_A and the other to S_B. Since their radial projection onto ρ_1 is to the same point, this proves that P_A is weakly separable from P_B. If the origin does not lie on such a line, then it must lie in the interior of a triangle defined by three vertices, say $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$. First assume that one vertex belongs to S_A, say $\mathbf{s}_1$, and that the other two belong to S_B. Consider the line that passes through $\mathbf{s}_1$ and the origin. Clearly $\mathbf{s}_2$ and $\mathbf{s}_3$ must

lie to either side of this line (in the plane ρ). From this it follows that the radial projection of $\mathbf{s}_2$ and $\mathbf{s}_3$ onto the line ρ_1 will lie to either side of the radial projection of $\mathbf{s}_1$. This means that a point of P_A lies on an edge of $\mathcal{CH}(P_B)$. Similarly, if one of the points belongs to S_B and the other to S_A , then a point of P_B lies on an edge of $\mathcal{CH}(P_A)$. Therefore it can be concluded that P_A is weakly separable from P_B whenever the origin is weakly separable from S . By reversing this argument, it can be shown that S is weakly separable from the origin whenever P_A is weakly separable from P_B.

(3) This last case follows directly from cases (1) and (2) and will not be shown. Thus S is inseparable from the origin whenever P_A is inseparable from P_B .



(a) strictly separable sets



(b) weakly separable sets

Figure 4.3 The convex hulls of two planar sets of points.

Finally, consider the case where S_O is not empty. Notice that points of S_O cannot be radially projected onto the plane $z = 1$. Furthermore, any point that coincides with the origin is a special point; it automatically guarantees that S is at most weakly separable from the origin. By rotating the points of S slightly about the origin, we can ensure that the only points of S that lie in the plane are the ones that coincide with the origin. If it is determined that S is strictly separable from the origin without knowledge of these coincident points, then the algorithm will report that S is weakly separable from the origin.

The following algorithm summarizes the results of this section. The function SetSet2D returns a record of type Separability (see Figure 4.4). Most of the information returned by this record is gathered during the call to SetSet2D , but there are two exceptions: line 3 below catches the degenerate case where all the points of S coincide with the origin while line 6 catches the case where at least one point of S coincides with the origin.

Algorithm 4.1 : Solving the 3-dimensional point-in-set problem.

Function **PointInSet3D** (**p** , S) : Separability ;

1. Translate both **p** and S so that **p** coincides with the origin. If necessary, rotate the points of S appropriately about the origin so that only points that coincide with the origin lie in the plane $z = 0$.
2. Partition S into S_A , S_O and S_B .
3. If both S_A and S_B are empty then { return a record that has class := weaklySeparable ; info := coincident and list := NIL }
4. Radially project S_A and S_B onto the plane $z = 1$, constructing the sets P_A and P_B .
5. result := SetSet2D(P_A , P_B) ;
6. If (result.class == strictlySeparable) and (S_O is not empty) then { result.class := weaklySeparable } ;
7. Restore the points of S to their original position, plus apply the same transformation to the points in result.list ; (The contents of result.list will be explained more fully when SetSet2D is discussed later in the chapter.)
8. Return(result) ;

– end of algorithm –

Aside from the call to SetSet2D , the above routine has a linear-time complexity, since each of the steps requires either linear or constant time. Thus the time-complexity for the above algorithm is then O(n) + T(n) , where T(n) is the time required to solve the SetSet2D problem when S_1 and S_2 contain a total of n points. In Section 4.2.1 it will be shown that SetSet1D requires O(n) time. Therefore, the worst case time complexity of PointInSet2D is O(n) .

The separability of two planar sets could be determined by using linear programming, as mentioned in the introduction of this chapter, and would lead us to conclude that the point-in-set problem is linear. However, our objective is to show that ICT can also be applied to the 2-dimensional set-set problem. Furthermore, if the two sets are inseparable, then the subset of points that prove this to be the case will be required in Chapter 5 . In Section 4.2.2.3 it will be shown that SetSet2D can be solved in O(t * $Maximum$ { n , t }) , where t is the number of iterations performed by the ICT algorithm. Therefore the the time-complexity of PointInSet3D is $O(t * Maximum\{n, t\})$, which is $O(t\,n)$ for $1 \le t << n$.

```
TYPE
      TypeOfSeparability = ( strictlySeparable , weaklySeparable , inseparable ) ;
      TypeOfInfo = ( unknown    , coincident , cone , halfPlane , halfSpace ,
                          wedge , inseparablePoints) ;

      pListEntry = ^ListEntry ;
      ListEntry =         RECORD
                          next : pListEntry ;
                          p : Point
                    END ;

      Separability = RECORD
                          class : TypeOfSeparability ;
                          info :  TypeOfInfo ;
                          list : pListEntry ;
                    END ;
```

Figure 4.4 A Pascal-like data structure for describing the supporting information

4.2 The Set-Set Problem

In this section, the following problems are considered in 1 and 2 dimensions:

1) determine whether two sets, S_1 and S_2 are strictly separable, weakly separable or else inseparable.

2) identify separator information for S_1 and S_2, which was discussed in the introduction of this chapter.

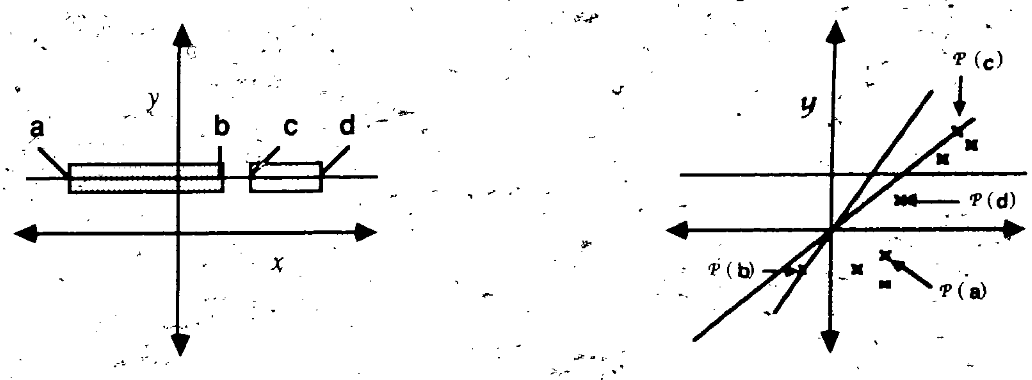
To facilitate the use of the results of this problem in the point-in-set problem, the following notation will be introduced. Let S denote the points of the original point-in-set problem. Suppose that $\mathbf{p} \in S$ and that $\mathbf{q}$ is the radial projection of $\mathbf{p}$ on the plane $z = 1$ (or the line $y = 1$ if S is 2-

dimensional). So far **q** has been referred to as the image of **p**. From now on, **p** will be referred to as the *originator* of **q**, and we define the function $\mathbf{p} = \mathcal{P}(\mathbf{q})$ to give us access to these original points. Thus if four points prove that S_1 and S_2 are inseparable, then from Theorem 4.2, the originators of these points prove that S is inseparable from the origin.

4.2.1 The 1-Dimensional Set-Set Problem

Determining the separability of two 1-dimensional sets is trivial. Since both sets lie on a line, their convex hulls can be represented by intervals. If the two intervals do not intersect, then the sets are strictly separable; if they meet at an endpoint then the two sets are weakly separable; otherwise they are inseparable. This test can be performed in constant time once the intervals have been determined in $O(n)$ time. What is of interest is the information that is captured concerning the location of S, which contains the originators of S_1 and S_2. The rest of this sub-section describes this information and presents a convention for returning it to the calling routine.

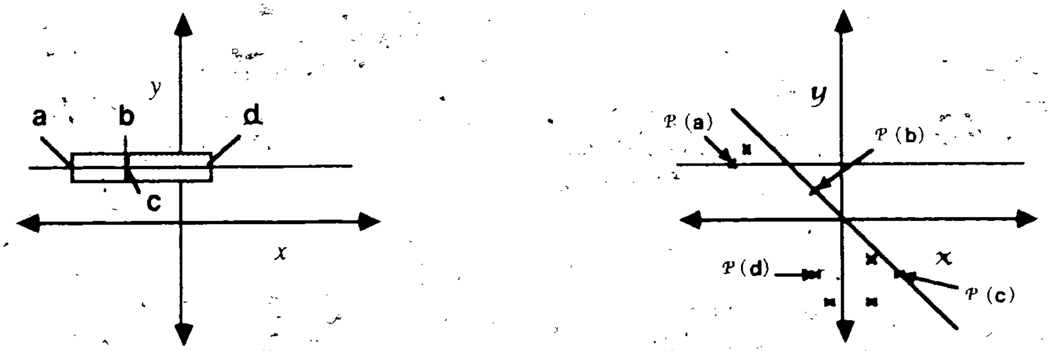
Let **a**, **b**, **c** and **d** denote the extreme points of the two intervals, ordered in non-decreasing order. First suppose that S_1 and S_2 are strictly separable. Each of the points on the line $y = 1$ that lie between **b** and **c** separates the two sets (see Figure 4.5.a). In fact, the points of S are contained in the cone whose vertex is at the origin and whose extreme rays are defined by $\mathcal{P}(\mathbf{b})$ and $\mathcal{P}(\mathbf{c})$, respectively (see Figure 4.5.b). If either S_1 or S_2 is empty, then the two sets are strictly separable in a trivial sense. In this case S is contained in the cone whose vertex is at the origin and whose extreme rays are defined by the originators of the extreme points of the non-empty set. If the two intervals are weakly separable, then the points of S lie in a closed half-plane whose boundary passes through $\mathcal{P}(\mathbf{b})$ and $\mathcal{P}(\mathbf{c})$ (see Figure 4.6). The correct half-plane can be identified by recording the originator of another point that does not lie on this line. If S_1 and S_2 are inseparable then the originators of **a**, **b**, **c** and **d** determine that the origin is inseparable from S (see Figure 4.7). Notice that any three points that prove that the two sets are inseparable will do. The only case where four points are required is when the two intervals coincide.



(a) S_1 and S_2 are strictly separable.

(b) The cone that encloses S is shaded

Figure 4.5 Example 1 of SetSet1D – strictly separable sets.



(a) S_1 and S_2 are weakly separable.

(b) The half-plane that encloses S is shaded

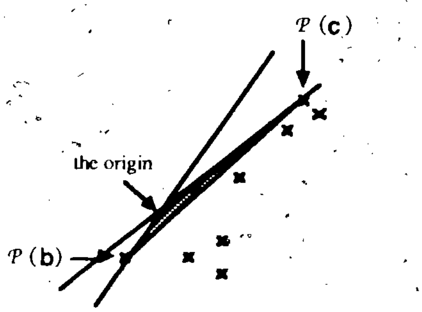
Figure 4.6 Example 2 of SetSet1D – weakly separable sets.

(a) S_1 and S_2 are inseparable.

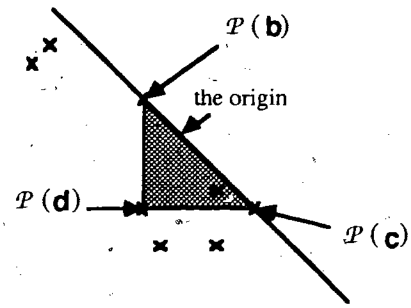
(b) The rectangle that encloses the origin is shaded

Figure 4.7 Example 3 of SetSet1D – inseparable sets.

The above information will be returned to the calling routine via a record of type Separability (which was described in Figure 4.4). Consider each of the fields of this record in turn: 'class' indicates the separability of the two sets; 'info' describes the contents of 'list'. If S_1 and S_2 are strictly separable, then 'list' will contain three points, the origin plus $\mathcal{P}($**b**$)$ and $\mathcal{P}($**c**$)$, ordered in a clockwise direction such that the origin is the second point in the list. As Figure 4.8.a illustrates, the cone that encloses S can be determined from the triangle defined by these points. If S_1 and S_2 are weakly separable, then 'list' will contain three points, $\mathcal{P}(\mathbf{b})$ and $\mathcal{P}(\mathbf{c})$ plus one other point of S that lies in the interior of the half-plane. These points will be ordered in a clockwise direction such that $\mathcal{P}(\mathbf{b})$ and $\mathcal{P}(\mathbf{c})$ are the first two elements of the list. Notice in Figure 4.8.b that the half-plane that contains S can be determined from the triangle defined by these points. Both triangles will be used in the next section to determine the separability of two planar sets. If the two sets are inseparable, then 'list' will point to the originators of three or four points that determine this fact.



(a) S_1 and S_2 are strictly separable.



(b) S_1 and S_2 are weakly separable.

Figure 4.8 The triangles that encode the supporting information.

4.2.2 The Planar Set-Set Problem

ICT will be used to solve the planar set-set problem. Briefly, the algorithm will test the separability of a point $\mathbf{g}$ from both of the sets by calling PointInSet2D twice, once for each set. Based upon the results of this test, the algorithm will either terminate immediately or it will make use of the separator information returned by PointInSet2D to reduce the area of the solution region. First the algorithm will be presented, followed by a description of 'InseparableOrWeaklySeparableTest' and 'FormatStrictlySeparableInfo', two routines called by 'SetSet2D'. Finally a detailed analysis of the algorithm is presented in Section 4.2.2.3.

Algorithm 4.2 : Solving the planar set-set problem.

Function **SetSet2D** (S_1 , S_2) : Separability ;

1. Initialization Step

1.1 Find a rectilinear bounding box that encloses S_1 and one that enclose S_2 .

1.2 Let Φ_0 denote the intersection of these two boxes.

1.3 If Φ_0 is empty, then {

1.4 Let **g** denote a point that lies between the two boxes (see Figure 4.9) ;

1.5 info1 := PointInSet2D(**g** , S_1) ;

1.6 info2 := PointInSet2D(**g** , S_2) ;

1.7 return (FormatStrictlySeparableInfo ((info1 , info2 , S_1 , S_2))

}

2. Iteration Step ($i \geq 1$)

2.1 **g** := $COG(\Phi_{i-1})$;

2.2 Φ_i := Φ_{i-1} ;

2.3 info1 := PointInSet2D(**g** , S_1) ;

2.4 info2 := PointInSet2D(**g** , S_2) ;

2.5 result := InseparableOrWeaklySeparableTest (info1 , info2) ;

2.6 if result.class == unknown then

2.7 { ReduceRegion (info1 , VAR Φ_i) ;

2.8 ReduceRegion (info2 , VAR Φ_i) ; }

3. Termination Predicate

3.1 If (result.class == unknown) and (Φ_i contains only a single point , **g**)

3.2 then { result := FormatStrictlySeparableInfo((info1 , info2 , S_1 , S_2)) }

3.3 If (result.class == unknown) then continue to iterate else return (result) ;

– end of algorithm –

Procedure **ReduceRegion** (info , VAR Φ_i) ; /* reduce the solution region */

1. if info.class == weaklySeparable then

   { let Φ_i denote the intersection of Φ_i and the half-plane described by info.list }

2. else if info.class == strictlySeparable then

   { let Φ_i denote the intersection of Φ_i and the cone described by info.list }

– end of algorithm –

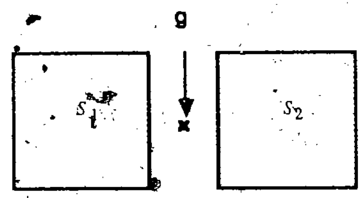


Figure 4.9 The bounding boxes of S_1 and S_2 are strictly separable.

4.2.2.1 Testing If The Sets Are Weakly Separable Or Inseparable

SetSet2D is unusual for an ICT algorithm since it continues to iterate until an exact solution has been reached. Thus, it is important to identify the separability of the two sets as soon as possible. The routine InseparableOrWeaklySeparableTest on line 2.5 of the Algorithm 4.2 is responsible for determining if the two sets are either weakly separable or inseparable from each other and if so, for formatting the supporting information that is returned. The results of this sub-section are summarized in Figure 4.10. (The columns labelled S_1 and S_2 in this diagram are interchangeable.)

Separability from g		Separability of S_1 and S_2
S_1	S_2	
inseparable	inseparable	inseparable
inseparable	weakly separable	inseparable
weakly separable	weakly separable	either weakly separable or inseparable
inseparable	strictly separable	unknown
weakly separable	strictly separable	unknown
strictly separable	strictly separable	unknown

Figure 4.10 Determining the separability of S_1 and S_2 based upon their separability from g.

Obviously, if both S_1 and S_2 are inseparable from g then S_1 and S_2 are inseparable. Similarly, if g is inseparable from one of S_1 and S_2 and weakly separable from the other, then S_1 and S_2 are inseparable. Both of these cases can be determined in constant time by testing 'info1.class' and 'info2.class' (see Section 4.2.1). Now consider the case where g is weakly separable from both sets. (For example, see Figure 4.11.) Is there enough information available to determine the separability of the two sets? Theorem 4.3 asserts that there is.

(a) convex hulls of two inseparable sets

(b) convex hulls of two weakly separable sets

Figure 4.11 Distinguishing between weakly separable and inseparable sets.

Theorem 4.3 Suppose that **g** is weakly separable from both S_1 and S_2. Let T_1 and T_2 denote the triangles described by 'info1.list' and 'info2.list', respectively (see Section 4.2.1). S_1 and S_2 are inseparable iff the triangles denoted by T_1 and T_2 are inseparable.

Proof: First consider two degenerate cases. If both sets coincide with **g** then S_1 and S_2 are inseparable. If the points of one set coincide with **g** but not the other, then S_1 and S_2 are weakly separable. From now on assume that both sets contain some points that do not coincide with **g**. Assume that T_1 and T_2 are inseparable. Recall that, by construction, $T_1 \subseteq \mathcal{CH}(S_1)$ and $T_2 \subseteq \mathcal{CH}(S_2)$. Therefore, if T_1 and T_2 are inseparable, then S_1 and S_2 must also be inseparable. Now assume that S_1 and S_2 are inseparable, but that T_1 and T_2 are not. Recall that the points of T_1 define either a cone or a half-plane that encloses the points of S_1. (A cone arises when a vertex of the convex hull of the set coincides with the point **g**.) If the points of T_1 define a cone, then **g** is the vertex of the cone; otherwise **g** lies on the boundary of the half-plane (see Figure 4.12). In either case, any line that supports T_1 at **g** must also support S_1. The same can be said of T_2 and S_2. Therefore, any line that separates T_1 and T_2 at **g** must also separate S_1 and S_2. This contradicts our assumption that these two sets are inseparable. Therefore we conclude that S_1 and S_2 are inseparable iff T_1 and T_2 are inseparable. ♦

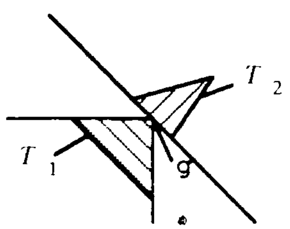
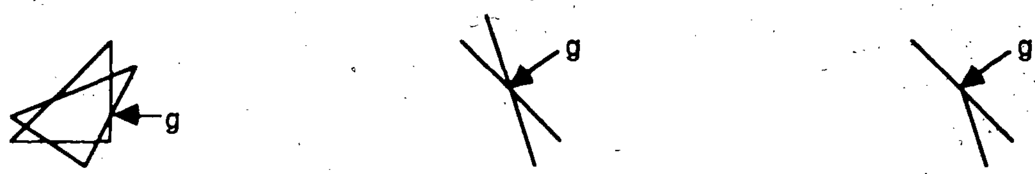


Figure 4.12 S_1 is contained in a cone while S_2 is contained in a half plane.

The triangles T_1 and T_2 are inseparable if any one of the following occurs: they are coincident; a vertex of one triangle lies in the interior of the other, and finally, if there is a crossing edge (see Figure 4.13.a). It can be determined if the two triangles are coincident in constant time, but the test must be sure to handle degenerate triangles properly (see Figures 4.13.b and 4.13.c). Degenerate triangles arise when the points of S_1 and S_2 are collinear.



a) inseparable triangles (b) inseparable degenerate triangles c) weakly separable degenerate triangles

Figure 4.13 Examples of inseparable and weakly separable triangles.

In summary, if **g** is not strictly separable from either of the sets, then the separability of the two sets can be determined in constant time. Otherwise, the algorithm will continue to iterate until the solution region has been reduced to a single point.

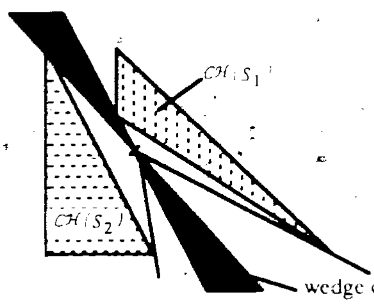
Now consider the supporting information that will be returned via a record of type Separability (Figure 4.4). If S_1 and S_2 are inseparable, then the fields of this record will be set to: class := inseparable , info := inseparablePoints and list will point to the originators of { info1 list ∪ info2 list } . If S_1 and S_2 are weakly separable, then it follows from Theorem 4.3 that 'info1 list' and 'info2 list' describe two triangles that are weakly separable at **g** . (Recall that **g** lies on the boundary of both triangles.) Since any line that separates the triangles also separates S_1 from S_2, it follows from Theorem 4.2 that a plane that passes through this line and the origin will separate S from the origin. Such a line can be found in constant time. Thus if the two sets are inseparable, then the fields of this record will be set to class := weaklySeparable , info := halfSpace and list will point to a set of four points that define a half-space that contains S . If the separability of the two sets is not known then the returned record will have class := unknown .

4.2.2.2 Formatting Supporting Information Of Strictly Separable Sets

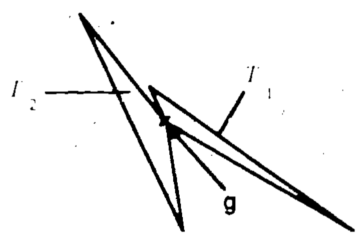
The function 'FormatStrictlySeparableInfo', which is called on lines 1.7 and 3.2 of Algorithm 4.2, is responsible for formatting the supporting information once it has been determined that S_1 and S_2 are strictly separable from each other. In such a case, SetSet2D should return two half-spaces whose intersection defines a wedge that contains S.

There are two conditions under which 'FormatStrictlySeparableInfo' is called: either **g** is strictly separable from both S_1 and S_2 or else it is strictly separable from one of the sets and weakly separable from the other. (It is impossible for the solution region to be reduced to a single point when **g** is inseparable from one of the sets.) In both cases, 'info1.list' and 'info2.list' describe two triangles, T_1 and T_2, which are weakly separable at **g**.

First assume that **g** is strictly separable from both S_1 and S_2. In this case T_1 and T_2 define two cones that contain S_1 and S_2 respectively. Furthermore, the boundary of a wedge of separating lines for the two triangles can be determined from the edges of T_1 and T_2 (see Figure 4.14). Since any line that separates the triangles also separates S_1 from S_2, it follows from Theorem 4.2 that a plane that passes through this line and the origin will separate S from the origin. Thus the half-spaces that define the boundary of the wedge that contains S can be determined from T_1 and T_2 in constant time.



(a)



(b) T_1 and T_2 are the white triangles in (a).

Figure 4.14 Illustrating the wedge of separators for S_1 and S_2.

Now consider the case where **g** is strictly separable from one of the sets and weakly separable from the other. A problem arises since only one separator exists for the two triangles, even though the two sets are strictly separable. Algorithm 4.3 describes the manner in which this will be handled.

Algorithm 4.3 : Formatting the supporting information for strictly separable sets.

```
Function  FormatStrictlySeparableInfo ( info1 , info2 , S1 , S2 ) :  Separability ;
1.    result.class := strictlySeparable ;
2.    result.info := wedge ;
3.    If ( info1.class == weaklySeparable ) or ( info2.class == weaklySeparable ) then
4.       { Without loss of generality, assume that T1 defines a cone and T2 defines a half-plane,
         as shown in Figure 4.12 ;
5.       Let u denote the point of S1 that is closest to the boundary line of the half-plane
         defined by T2 and let v denote a point that is half-way between u and this line.
6.       info1 := PointInSet2D( v , S1 ) ;
7.       info2 := PointInSet2D( v , S2 ) ;                }
8.    Since v is strictly separable from both sets, construct the wedge information as described above
      and return it through result.list ;
-- end of algorithm --
```

In analyzing the above algorithm, lines 1 – 4 and 8 can be performed in constant time, while lines 5 - 7 require O(n) time each, where n is the total number of points of S_1 and S_2. Therefore the time-complexity of this algorithm is O(n) time.

4.2.2.3 Analysis and Discussion of Algorithm 4.2 (**SetSet2D**)

Finally, the analysis of Algorithm 4.2 is presented. Assume that S_1 has n_1 points and that S_2 has n_2 and let $n = n_1 + n_2$.

1. First consider the initialization step. A point that lies in the convex hull of a set will lie in the rectilinear bounding box that encloses any one of the sets. Both boxes can be found in O(n) time; the intersection of the two boxes can be performed in constant time.

2. Now consider the case where the intersection is empty. In this case, the two sets are strictly separable but the supporting information still needs to be determined. This will be handled by finding a point **g** that lies between the two bounding boxes. **g** can be found in constant time by considering the vertices of the two bounding boxes. PointInSet2D , the 2-dimensional algorithm for solving the point-in-set problem, requires time linear in the number of points of the set (Section 4.1). Therefore the calls to PointInSet2D on lines 1.5 and 1.6 require O(n_1) and O(n_2) time. The call to

FormatStrictlySeparableInfo will require constant time since **g** is strictly separable from both sets.

3. Therefore the total cost of the initialization step is O(n) time.

4. Now consider the number of edges of a solution region. Φ_0 will have at most four edges. During each iteration, the number of edges of the solution region will be increased by at most two because the point **g** lies on the apex of each cone, or on the boundary of each half-plane used to cut the solution (lines 2.7 and 2.8). Therefore, in the worst case, the solution region will have O(i) edges at the beginning of iteration i.

5. The centre of gravity of the solution region can be found in a time linear to the number of edges of the solution region (Section C.3). Therefore, the centre of gravity can be found in O(i) time.

6. The two calls to PointInSet2D on lines 2.3 and 2.4 require a total of O(n) time.

7. As was described in Section 4.2.2.2, the call to InseparableOrWeaklySeparableTest can be performed constant time.

8. Since a cone can be thought of as the intersection of two half-planes, the next solution region will be constructed by intersecting the current solution region with from one to four half-planes (lines 2.6 to 2.8). The intersection of a convex polygon and a half-plane can be computed in time linear in the number of edges of the convex polygon (Section C.4). Since the solution region will have at most O(i) at the beginning of iteration i, this step can be performed in O(i) time.

9. Since **g** lies on the boundary of each half-plane that intersects the solution region, it follows from Winternitz's theorem that the area of the solution region is reduced by at least a fixed fraction each iteration.

10. If the solution region has been reduced to a single point **g** , and **g** is strictly separable from either S_1 or S_2, then S_1 and S_2 are strictly separable. This follows from the fact that the intersection on line 2.7 does not cut away any of the convex hull of S_1. Similarly, the intersection on line 2.8 does not cut away any of the convex hull of S_2. Therefore, if the solution region has been reduced to a single point, and this point does not lie in the convex hull of one of the sets, then the two sets must be strictly separable.

11. If the algorithm is continuing to iterate, then the termination predicate requires constant time since the call to FormatStrictlySeparableInfo ensures that the algorithm will terminate. In the worst case,

this call will require $O(n)$ time (see previous section). Therefore the total cost of the termination predicate is $O(n)$ time.

12. There is a termination test that could be added to the algorithm which results in early termination in some cases. Recall that any line segment that connects two points of a sets lies in the convex hull of the set. Since 'info1.list' and 'info2.list' (from line 2.3 and 2.4) both contain some points of S_1 and S_2 respectively, the lines connecting these points can be tested to see if they prove the sets are inseparable. This can be thought of as a generalization of Theorem 4.3. For example, in Figure 4.15.a, the algorithm would terminate immediately if this test were implemented since the line segment **ab** crosses the line segment **cd**, proving the two sets are inseparable. Without the test, the algorithm would continue to iterate with the solution region shown in Figure 4.15.b.

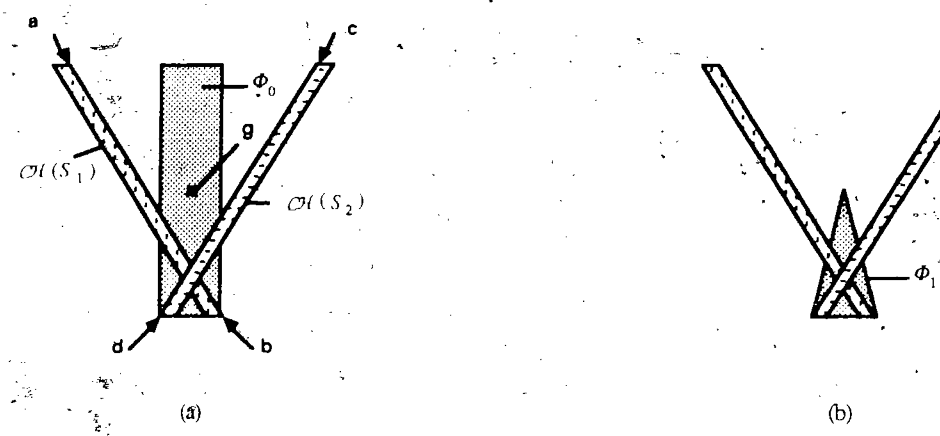


Figure 4.15 Illustrating the solution region after one iteration of Algorithm 4.2.

Thus, in summary,

- $O(n)$ time is required for the initialization step.
- In the worst case, the i^{th} iteration requires

  $\mathcal{Maximum}\ \{\ O(n),\ O(i),\ O(1)\ \} = O(\ \mathcal{Maximum}\ \{\ n,\ i\ \}\ )$ time.
- $O(n)$ is required for the termination predicate.

Therefore, the total running time for Algorithm 4.2 is $O(\ t * \mathcal{Maximum}\ \{\ n,\ t\ \}\ )$ time, where t is the number of iterations performed. As was argued at the end of Section 1.3 and Section 3.2.1, the number of iterations of an ICT algorithm can be bounded from above by a constant provided that the algorithm is implemented using fixed-precision, floating point arithmetic. Under this assumption, t is bounded from above by a constant. In this case, the running time for Algorithm 4.2 is $O(n)$.

4.3 Detecting If The Convex Hulls Of m Sets Of Points Overlap

Finally, we are ready to solve the main problem of this chapter. In this section, ICT is used to detect whether the convex hulls of m sets of points overlap or not. The approach used to solve this problem is much the same as that used in SetSet2D (Algorithm 4.2). Algorithm 4.4 , which is presented below, has two ways of terminating: either the algorithm identifies a point that lies in the convex hull of each of the m sets, or else it reduces the solution region to a single point that does not lie in the convex hull of at least one of the sets. In the former case, the algorithm reports, 'YES, the convex hulls of the m sets do overlap', and in the latter, it reports 'NO, they do not overlap' .

During each iteration of Algorithm 4.4 , PointInSet3D is called a total of m times, in order to determine whether the centre of gravity of the current solution region (**g**) lies in the convex hull of each of the m sets. If it turns out that **g** is either inseparable or weakly separable from each of the sets, then the algorithm terminates, reporting **g** is common to the convex hull of each of the m sets.

Determining that the sets do not overlap is a more difficult problem, since the solution region must be reduced to a single point. If **g** does not lie in the convex hull of each of the sets, then it must be strictly separable from at least one of the sets. In this case, the wedge returned by PointInSet3D can be used to reduce the volume of the solution region for the next iteration. However, theoretically, it is not possible for a single wedge to reduce the solution region to a single point, since the centre of gravity of a convex region lies in its interior and since **g** lies on the boundary of the wedge. The most naive solution to this problem is to intersect the current solution region with each of the half-spaces returned by the m calls to PointInSet3D . However, this would require intersecting the solution region with from 2 to (2 m) half-spaces each iteration, an operation that requires more time than we are willing to spend. Also only the first intersection guarantees that the solution region will be reduced by a fixed fraction. The rest of the intersections may have limited benefit. Instead, a test has been developed that determines whether the result of the intersection would be a single point, if the intersection did take place. If so, the current solution region is replaced with one that contains only the point **g** . Otherwise, the solution region is reduced by a fixed fraction by intersecting it with the last wedge that has been returned by PointInSet3D .

The test consists of mapping each of the h half-spaces to points on the surface of a unit sphere, and mapping $\mathbf{g}$ to its origin (the centre of the sphere). Let $\mathbf{n}_1$, $\mathbf{n}_2$, ..., $\mathbf{n}_h$ respectively denote the h outward unit normals. Place each normal so that its tail coincides with the origin. This yields a total of h points on the surface of the unit sphere. In Section 4.3.1 , it will be shown that if the mapped points are inseparable from the origin, then the intersection of the corresponding half-spaces will contain just one point, $\mathbf{g}$.

Algorithm 4.4 will be presented first, followed by a discussion of ReduceSolutionRegion in Section 4.3.1 . (ReduceSolutionRegion is responsible for performing the termination test described above and for reducing the solution region appropriately. In addition, it ensures that the problem of degenerate convergence does not arise.) Finally, in Section 4.3.2 , a detailed discussion and analysis of Algorithm 4.4 is presented.

Let S_0 , S_1 , ... , S_{m-1} denote m sets of points in 3 dimensions. In the following algorithm, N denotes the list of points resulting from the mapping of the half-spaces, while SaveNormals is the routine responsible for constructing this list.

<u>Algorithm 4.4 : Detecting whether the convex hulls of m sets of points overlap.</u>

Program **main** (S_1 , S_2) ;

1. **Initialization Step**

   1.1 Find a rectilinear bounding box that encloses each of the sets. Let Φ_0 denote the intersection of the m boxes.

   1.2 If Φ_0 is empty, then terminate, reporting that the convex hulls do not overlap.

2. **Iteration Step** ($i \geq 1$)

```
2.1   g := COG( Φ_{i-1} ) ;
2.2   gInEachConvexHull := true ;
2.3   N := nil ;
2.4   for j := 0 , to  m - 1  do {
2.5           result := PointInSet3D( g , S_j ) ;
2.6           if result.class == strictlySeparable then {
```

2.7 glnEachConvexHull := false ;

2.8 Let Γ_1 and Γ_2 denote the two half-spaces described by result.list ; }

2.9 if result.class $\neq$ inSeparable then SaveNormals(result , **g** ; VAR N) ;

} /* end of for loop */

2.10 if glnEachConvexHull == false then

Φ_i := ReduceSolutionRegion (**g** , N , Φ_{i-1} , Γ_1 , Γ_2) ;

3. Termination Predicate

3.1 If glnEachConvexHull == true ,

3.2 then { terminate, reporting 'YES, the convex hulls overlap at the point **g** ' }

3.3 else if (Φ_i contains only a single point , **g**)

3.4 then { terminate, reporting 'NO, they do not overlap '}

3.5 else { continue to iterate. } ;

– end of algorithm –

Procedure **SaveNormals** (result , **g** ; VAR N) ;

/* Recall that since **g** is either weakly or strictly separable from S_j , that 'result.list' describes either one or two half-spaces that contain all of the points of S_j . /*

1. for each half-space Ψ (described by result.list) do {

2. Let **n** denote the outward unit normal for the half-space Ψ . Let **q** denote the point that is yielded when the tail of **n** coincides with the origin .

3. Append **q** to N ; }

– end of algorithm –

4.3.1 Reducing The Solution Region For Algorithm 4.4

Three topics will be discussed in this section. First, it will be shown that given the set N , a set of points on the surface of a unit sphere centred at the origin, it is possible to determine whether the set { **g** } is the result of intersecting the half-spaces that were used to create N . (The mapping between points and half-spaces, and the construction of N was described in the previous section.) Second, the strategy that will be used by ReduceSolutionRegion to avoid degenerate convergence will be discussed. Finally, the algorithm for ReduceSolutionRegion will be presented, along with an analysis of this algorithm.

Theorem 4.4 If the points of N are inseparable from the origin, then $\{ \mathbf{g} \}$ is the result of intersecting the half-spaces used to create the set N.

Proof: Assume that the points of N are inseparable from the origin. From this it follows that the origin lies in the interior of the convex hull of these points. Now map this convex hull onto the surface of the sphere as follows: map each edge to the smallest arc of the great circle determined by its vertices. This mapping creates a spherical subdivision on the surface of the sphere, consisting of faces, arcs and vertices. Consider an arbitrary face of the subdivision. The vertices of this face correspond to half-spaces in the original space whose intersection is an unbounded pyramid, Δ, which has $\mathbf{g}$ as its apex. Let $\mathbf{a}$ denote a point that lies in the interior of this spherical face. Notice that $\mathbf{a}$ corresponds to a half-space Ψ (in the original space) such that $\Delta = \Delta \cap \Psi$. Thus Ψ can be added to the original set of half-spaces without affecting the result of their intersection. Let Ψ^- denote a second half-space that has the same boundary as Ψ but extends in the opposite direction. Observe that $\{ \mathbf{g} \} = \Delta \cap \Psi^-$. This is the critical observation of the proof: if both Ψ and Ψ^- can be added to the set of half-spaces, without affecting the result of the intersection, then result of this intersection must be $\{ \mathbf{g} \}$. Let $\mathbf{b}$ denote the point on the unit sphere that corresponds to Ψ^- ($\mathbf{a}$ and $\mathbf{b}$ are diametrically opposite). Three cases may occur: (i) if $\mathbf{b}$ coincides with a vertex of the spherical subdivision, then the half-space Ψ^- is already one of the original set of half-spaces; (ii) if $\mathbf{b}$ lies in the interior of a face of the subdivision, then it follows from above that Ψ^- can be added to the original set of half-spaces without affecting the result of original intersection; (iii) $\mathbf{b}$ lies on an arc of the subdivision, but is not one of the endpoints of this arc. The vertices of this arc correspond to half-spaces in the original space whose intersection determines a wedge that has $\mathbf{g}$ lying on its boundary. Let Γ denote this wedge. Observe that $\Gamma = \Gamma \cap \Psi^-$. Thus, Ψ^- can be added to the original set of half-spaces without affecting the result of their intersection. Since in all three cases, Ψ^- can be added without affecting the result of the intersection of these half-spaces, it follows that the result of intersecting the original set of half-spaces must be $\{ \mathbf{g} \}$. ●

PointInSet3D will be used to test whether the origin is inseparable from N. If it is, then the current solution region will be replaced with a solution region that contains only the point $\mathbf{g}$. The correctness of this approach follows directly from Theorem 4.4. However, a question arises as to the efficiency of this approach. That is, is it possible for the intersection of the set of half-spaces to be $\{ \mathbf{g} \}$ when the origin is separable from N? The answer is no. To show this, first assume that the origin is weakly separable from N. From this it follows that all the points of N lie in a hemisphere of the unit sphere. Furthermore, the great circle that defines the boundary of this hemisphere passes through a subset of N, such that the origin lies in the interior of the convex hull of this subset. The intersection of the half-spaces associated with this subset will result in a line. A point that does not lie on this great circle corresponds to a half-space that will reduce the line to a half-line. However, no further reduction is possible since there is no point in the opposite hemisphere. Thus if the origin is weakly separable from the origin, then the result of intersecting the corresponding half-spaces will either be a line or a half-line. If the origin is strictly separable, then the result of the intersection will an unbounded region with volume.

Now consider the problem of degenerate convergence. As was stated in step 7 of Section 3.2.1, degenerate convergence arises when the solution region does not converge in all possible directions. That is, instead of converging to a point, it converges to a line segment or a convex polygon. Under the assumptions of Appendix C (that fixed-precision, floating-point numbers will be used to approximate real numbers), the solution region will be recast to a lower dimension by the intersection routine described in Section C.4, once it has been determined that the volume is effectively zero. In such a case, Algorithm 4.4 will continue to iterate with a solution region that has a lower dimension. As was mentioned in Section 3.2.1, no problems arise as long as a fixed fraction of the remaining solution region is cut away each iteration. In the worst case, the solution region will converge to a convex polygon, next to a line segment, and finally to a point. Thus in practice, it is possible for the solution region to be reduced to a single point by intersecting it with a single wedge each iteration, even though this is not possible theoretically.

So the following question naturally arises. Is it is possible for solution region to be not reduced by a fixed fraction each iteration? Recall that the solution region is intersected with a wedge that is defined by the intersection of Γ_1 and Γ_2, two half-spaces that are supplied to ReduceSolutionRegion via input parameters, each of which have $\mathbf{g}$ lying on their boundary. It was argued in Section 3.2.1 that, as long as the solution region is not completely contained in the boundary of both half-spaces, then the solution region will be reduced by a fixed fraction. (For example, see Figure C.1 .) If it is, then the current solution region must be a line segment that lies in the line determined by the intersection of the boundaries of Γ_1 and Γ_2. Such a case cannot be ignored, since if it arises, Algorithm 4.4 will go into an infinite loop: successive centres of gravity will coincide leading to the same choice of Γ_1 and Γ_2 in the iterations that follow. Clearly such a situation can be detected in O(1) time. However, what should be done once it is detected? In the following it will be shown that in such a case, the calling routine (Algorithm 4.4) should terminate since no point of the current solution region lies in the convex hull of at least one of the m sets. This will be signalled to the calling routine by returning $\Phi_i = \{ \mathbf{g} \}$.

Assume that the current solution region is contained by the line determined by the intersection of the boundaries of Γ_1 and Γ_2. Let λ denote this line. Recall from Algorithm 4.4 , that the routine ReduceSolutionRegion is called only if $\mathbf{g}$ is strictly separable from at least one of the sets. Let S_j denote the last set that was determined to be strictly separable from $\mathbf{g}$. In this case, the half-spaces Γ_1 and Γ_2 determine a wedge that contains all the points of S_j. Without loss of generality, assume that $\mathbf{g}$ coincides with the origin. Therefore λ passes through the origin since $\mathbf{g}$ lies on the boundary of both Γ_1 and Γ_2. In the following it will be shown that that there exists a plane that contains λ but does not intersect the convex hull of S_j. Therefore no point of λ can lie in the convex hull of S_j and hence, by assumption, neither can the a point of the current solution region. Consider the point $\mathbf{g}_1$, determined by the intersection of λ and the plane $z = 1$. Recall that PointInSet3D constructs two sets, S_{j1} and S_{j2} and then calls SetSet2D to determine their separability. The point $\mathbf{g}_1$ is last the centre of gravity determined by the SetSet2D algorithm, and it is through this point that the wedge of separators shown in Figure 4.14 pass. It is easy to see that any line that lies in the interior of this wedge strictly separates S_{j1} from S_{j2}. Therefore, by Theorem 4.2, the plane determined by this line and the origin for the 3-dimensional problem will have all the points S_j lying to one side of it. Thus, such a plane cannot intersect the convex hull of S_j. Furthermore, since this plane contains both the origin and $\mathbf{g}_1$, it must also contain the line λ. Therefore we conclude that no point of the current solution region lies in the convex hull of S_j.

The following algorithm is a summary of the above comments. Let $\mathbf{o}$ denote the origin.

<u>Algorithm 4.5 : Reducing the solution region for Algorithm 4.4 .</u>

Function **ReduceSolutionRegion** ($\mathbf{g}$, N , Φ_{i-1} , Γ_1 , Γ_2) : Φ_i ;

1. tmpResult := PointInSet3D($\mathbf{o}$, N) ;
2. if tmpResult.class == inseparable
3. then { return (Φ_i set to the single point $\mathbf{g}$)
4. else { if Φ_{i-1} is completely contained in the planes that define the boundaries of Γ_1 and Γ_2 ,
5. then { return (Φ_i set to the single point $\mathbf{g}$) }
6. else { return ($\Phi_i := \Phi_{i-1} \cap \Gamma_1 \cap \Gamma_2$) } ;

} /* else */

– end of algorithm –

Recall that the set N has at most $(2m)$ points. Let f denote the number of faces of Φ_{i-1}. In analyzing Algorithm 4.5 ,

- O(t_m * $\mathcal{Maximum}$ { m , t_m }) time is required for line 1 , where t_m is the number of iterations of this ICT algorithm for PointInSet3D .
- O(f) time is required to delete Φ_{i-1} and replace it with Φ_i = { $\mathbf{g}$ } . Thus lines 3 and 5 require O(f) time.
- It is easy to see that the test on line 3 can be performed in O(1) time.
- The intersection of Φ_{i-1} and one half-space requires O(f) time (see Section C.4). Therefore the intersection described on line 6 can be performed in O(f) time.

Thus, Algorithm 4.5 requires a total of

$$O(\ Maximum\ \{\ (t_m * Maximum\ \{\ m\ ,\ t_m\ \})\ ,\ f\ \}$$

time. As was argued at the end of Section 1.3 and Section 3.2.1, the number of iterations of an ICT algorithm can be bounded from above by a constant provided that the algorithm is implemented using fixed-precision, floating point arithmetic. Under this assumption, t_m is bounded from above by a constant. In this case, the running time for Algorithm 4.2 is $O(\ Maximum\ \{\ m\ ,\ f\ \}\ )$.

4.3.2 Analysis and Discussion of Algorithm 4.4

Finally, the analysis of Algorithm 4.4 will be presented. In the following discussion, assume that the sets $S_0, S_1, \ldots, S_{m-1}$ have $n_0, n_1, \ldots, n_{m-1}$ points respectively, and that $n = n_0 + n_1 + \ldots + n_{m-1}$.

1. First consider the initialization step. A point that lies in the convex hull of all of the sets lies in the rectilinear bounding box that encloses each one of the sets. Therefore if Φ_0 is empty, the algorithm can terminate reporting that the convex hulls do not overlap. All of the bounding boxes can be found in $O(n)$ time, and their intersection, Φ_0, can be computed in $O(m)$ time. Since each set must have at least one element, $m < n$. Therefore the initialization step requires $O(n)$ time.

2. Now consider the number of faces of the solution region. Φ_0 has at most six faces. Each iteration the number of faces of the solution region is increased by at most two. Therefore, in the worst case, the solution region has $O(i)$ faces at the beginning of iteration i.

3. The centre of gravity of a 3-dimensional solution region can be found in time linear to the number of faces of the region (Section C.3). Therefore, the centre of gravity can be found in $O(i)$ time.

4. The for-loop, defined on lines 2.4 to 2.9, is responsible for testing whether **g** lies in the convex hull of each of the sets. First consider the correctness of this loop. Before entering the loop, gInEachConvexHull is initialized to true and N is initialized to nil. The sets S_0 to S_{m-1} are each tested in order. If **g** does not lie in the convex hull of a set, say S_j, then gInEachConvexHull is set to false and the two half-spaces defining the wedge that contains S_j are saved in Γ_1 and Γ_2. If **g** is not inseparable from a set, then the half-space(s) described by result.list are mapped to points on the unit sphere by SaveNormals, which are then appended to the

list N. If the for-loop terminates with gInEachConvexHull set to true, then **g** lies in the convex hull of all m sets.

Now consider the time requirements for the for-loop during the i^{th} iteration. All the steps except the call to PointInSet3D can be performed in constant time. Recall from Section 4.1 that PointInSet3D requires $O(t * \mathcal{M}aximum\{ n, t \})$, where t is the number of iterations performed by the ICT algorithm on an input of size n. Therefore, one iteration of the for-loop requires $O(t_{i,j} * \mathcal{M}aximum\{ n_j, t_{i,j} \})$, where $t_{i,j}$ is the number of iterations performed by PointInSet3D for set S_j. Thus, during each iteration, the for-loop will require

$$\sum_{j=0}^{m-1} O(t_{i,j} * \mathcal{M}aximum\{ n_j, t_{i,j} \}) \text{ time.} \qquad [4.1]$$

By assuming $t_{i,j} << n_j$, equation [4.1] simplifies to

$$\sum_{j=0}^{m-1} O(t_{i,j} * n_j) \qquad [4.2]$$

Furthermore, equation [4.2] can be rewritten as

$$O\left(n * \sum_{j=0}^{m-1} t_{i,j} \right) \text{ time,} \qquad [4.3]$$

since each $n_j \leq n$.

5. 'ReduceSolutionRegion' (line 2.10) requires

$$O(\mathcal{M}aximum\{ (t_{i,m} * \mathcal{M}aximum\{ m, t_{i,m} \}), f_i \})$$

time to reduce the solution region, where f_i is the number of faces of the region during iteration i and $t_{i,m}$ is the total number of iterations of the call made to PointInSet3D. Since the solution region has $O(i)$ faces during the i^{th} iteration and since $m \leq n$, this step requires

$$O(\mathcal{M}aximum\{ i, t_{i,m} * \mathcal{M}aximum\{ n, t_{i,m} \} \}) \text{ time.}$$

6. Finally, the termination predicate can be performed in $O(1)$ time.

In summary,

- $O(n)$ time is required for the initialization step;

- The i^{th} iteration requires:

  $O(i)$ time to determine the centre of gravity of the region ;

  $O\left(n * \sum_{j=0}^{m-1} t_{i,j} \right)$ time for the for-loop

  $O(\mathcal{M}aximum\{ i, t_{i,m} * \mathcal{M}aximum\{ n, t'_{i,m} \} \})$ time for ReduceSolutionRegion .

  Therefore, in total, the i^{th} iteration requires:

  $$O\left(\mathcal{M}aximum\left\{ i, n * \sum_{j=0}^{m} t_{i,j} \right\} \right) \text{ time.}$$

- $O(1)$ time is required for the termination predicate during the i^{th} iteration.

Let $T(n)$ denote the total running time for Algorithm 4.4 and let t denote the total number of iterations performed. In this case,

$$T(n) = O(n) + O(x) + O(t), \qquad [4.4]$$

where, $x = \sum_{i=1}^{t} O\left(\mathcal{M}aximum\left\{ i, n * \sum_{j=0}^{m} t_{i,j} \right\} \right)$.

By letting, $t' = \sum_{i=1}^{t} \sum_{j=0}^{m} t_{i,j}$,

we find that $x = O\left(\mathcal{M}aximum\{ t^2, t' n \} \right) = O(t' n)$ Substituting this back into [4.4] gives

$$T(n) = O(n) + O\left(\mathcal{M}aximum\{ t^2, t' n \} \right) + O(t).$$

Assuming that $t << n$, the running time of the Algorithm 4.4 is $O(t' n)$ since $t \le t'$. As was argued at the end of Section 1.3 and Section 3.2.1, the number of iterations of an ICT algorithm can be bounded from above by a constant provided that the algorithm is implemented using fixed-precision, floating point arithmetic. Under this assumption, each $t_{i,j}$ is bounded from above by a constant, as is t. In this case, the running time for Algorithm 4.2 is $O(n)$.

Chapter 5

Linear Programming In 2 and 3 Dimensions

The Linear programming (LP) model minimizes a linear function subject to a set of linear equations and inequalities (constraints). In this chapter we will present an ICT algorithm that solves LP in 2 and 3 dimensions. [Edelsbrunner 87] (page 239) has noted that efficient solutions for low-dimensional LP problems have a large potential to lead to efficient solutions for other common geometric problems. For example, Kirkpatrick and Seidel's O($n \log h$) convex hull algorithm (h is the number of points on the convex hull) exploits the fact that 2-dimensional LP can be solved in O(n) time. [Kirkpatrick and Seidel 86]. In fact many geometric problems can be expressed directly as linear programming problems of low dimension. This is true of the Chebyshev line fitting problem and the smallest enclosing circle, which will be described later in the thesis. Other examples can be found in [Edelsbrunner 87] (pages 213, 236–239) and [Dobkin and Reiss 80].

In Section 5.1, a description of the geometric interpretation of LP in 2 and 3 dimensions is presented, followed by a brief history of some of the research that has taken place in this area (Section 5.2). A detailed discussion of the ICT solution for LP is presented in Section 5.3 . The most challenging aspect of this solution has been the creation of the initial solution region, which is described in Section 5.3.1 . The approach described there constructs a point-in-set problem from the constraints, and then uses the routines described in Chapter 4 to identify a small number of constraints whose intersection is bounded in all directions. (Note that it is possible that no such subset exists. In this case, the ICT algorithm for LP terminates immediately, indicating the direction in which the problem is unbounded. In the other case, that is, when the solution is finite but the set of feasible points is infinite (for example, see Figure 5.1.a), it is expected that the user will add a constraint that results in a bounded region before restarting the process.) The iteration component of the algorithm is discussed in Section 5.3.2 ; two methods of reducing the solution region are discussed. The first approach is the easiest to explain and implement, but may lead to degenerate convergence. The second approach is a much stronger result: given an arbitrary line, it is possible to ensure that the next solution region will extend to only one side of a plane that contains this line. Thus, if the algorithm detects that the solution region is not converging in a particular direction, then a line perpendicular to this direction will be supplied, ensuring convergence in that direction. Termination of the algorithm is discussed in Section 5.3.3, followed by the actual ICT algorithm in Section 5.4. Finally in Section 5.5, it is shown that ICT can be combined with the *prune-and-search* technique, which was independently introduced by [Megiddo 83a] and [Dyer 84], resulting in linear-time algorithm that produces an exact solution.

5.1 Linear Programming (LP) In 2 and 3 Dimensions

The dimension of an LP problem is determined by the maximum number of independent variables in a constraint. The 3-dimensional LP problem can be stated formally as follows:

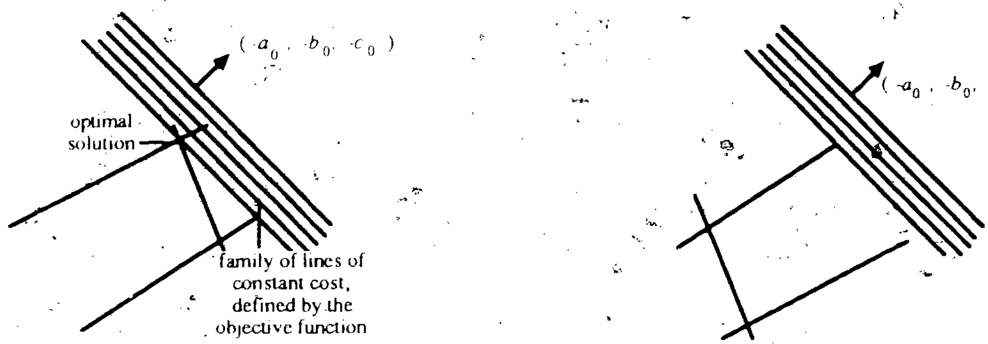
$$\begin{array}{ll} \underset{x,\,y,\,z}{\text{minimize}} & a_0\,x + b_0\,y + c_0\,z \\ \text{subject to} & a_i\,x + b_i\,y + c_i\,z \le d_i \;,\quad i = 1, \ldots n \;. \end{array} \qquad [5.1]$$

The linear form $a_0\,x + b_0\,y + c_0\,z$ is called the *objective* or *cost* function, while each of the n inequalities are called *constraints*. Notice that no equalities have been included in the above constraints since an equality is easily represented as two inequalities. For example, the plane described by the equation $a_j\,x + b_j\,y + c_j\,z = d_j$ is also described by the inequalities: $a_j\,x + b_j\,y + c_j\,z \le d_j$ and $a_j\,x + b_j\,y + c_j\,z \ge d_j$.

Feasible solutions correspond to those points that satisfy all constraints. The role of the objective function is to formalize the criteria for choosing the best feasible solution, for example, one that minimizes the cost of production. Geometrically, each constraint represents a closed half-space, and the intersection (F) of the n half-spaces corresponds to the set of feasible points. If F is empty, then the problem is said to be *infeasible*. That is, no point satisfies all of the constraints. Otherwise, the optimal solution is the point of F which lies furthest in the direction determined by the vector $(-a_0, -b_0, -c_0)$. If F is *unbounded* in this direction, then the optimal solution is at infinity.

Otherwise the optimal solution is a point $\mathbf{p} \in F \cap \lambda$, where λ is the supporting plane of F that is a member of the linear functions of constant cost that are defined by the objective function, such that λ bounds F in the direction $(-a_0, -b_0, -c_0)$ (see Figure 5.1). We will refer to λ as the *objective supporting plane*. Notice that the slope of this plane can be determined directly from the objective function.

F is convex since it is the intersection of n half-spaces. This restricts λ to 'touching' F in one of the following ways: if it touches F at a single point then this vertex is the optimal solution for the problem; if it intersects either a face or an edge of F, then many optimal solutions exist, all of them equally good.



(a) a bounded 2D LP problem (b) an unbounded 2D LP problem

Figure 5.1 Examples of Feasible LP problems with the same objective function.

In summary, there are three types of LP problems, each having a geometric interpretation:

- infeasible problems;
- feasible problems that are unbounded in the direction $(-a_0, -b_0, -c_0)$;
- bounded feasible problems, guaranteed to have at least one finite optimal solution.

5.2 History of LP

Linear programming was first introduced in the late 1940's by George B. Dantzig, who was trying to mechanize some of the planning processes for the U.S. Air Force. Besides presenting the LP model, Dantzig designed the 'simplex method' which is still the most widely used method for solving general LP problems. There are many texts that describe techniques for solving the general LP problem, for example, [Chvátal 83] or [Papadimitriou and Steiglitz 82]. Since we will consider only problems that are of low dimension, and since the techniques used to solve LP in low dimensions efficiently are fundamentally different from techniques used to solve the general LP problem ([Edelsbrunner 87] , page 238) , we will note only a few of the results obtained for the general LP problem.

- [Klee and Minty 72] have shown that the simplex method's worst case time-complexity is exponential in the size of input m, where m is the total number of variables and constraints (although it runs very fast on average) .
- [Khachiyan 79] has presented the *ellipsoid method* for solving LP which has a worst case running time that is polynomial. This result is mainly of theoretical interest since the typical number of iterations seems to be very large even on reasonably small problems, and each individual iteration may be prohibitively laborious ([Chvátal 83] , page 451) .
- [Karmarkar 84] has presented a variation of Khachiyan's algorithm which is expected to be efficient in the 'expected' case also.

One approach to solving the 2 or 3-dimensional LP problem is to find the intersection of the n constraints in $O(n \log n)$ time ([Preparata and Muller 79]) and then find the supporting line that determines the optimal solution in $O(\log n)$ time ([Shamos 78], Section 3.3.6). Thus the total running time for this approach is $O(n \log n)$. [Guibas, Stolfi and Clarkson 87] have augmented this approach to solve a slightly different problem, one where the constraints of the LP problem are relatively stable, but the objective function changes frequently. They preprocess the constraints into a structure such that, given any linear objective function, they can report the point(s) in space that minimize this function in $O(\log n)$ time. Their preprocessing step has two stages: first they find the polyhedron defined by the intersection of the constraints in $O(n \log n)$; second, they map this polyhedron onto a unit sphere in $O(n)$ time. Once this has be done, then an $O(\log n)$ point location algorithm ([Kirkpatrick 83] , [Edelsbrunner, Guibas and Stolfi 86]) can be used to determine the optimal solution. It is interesting to note that their method of mapping the polyhedron onto the unit sphere has also been used by [O'Rourke 85] and [Zorbas 86] to obtain supporting line information for the polyhedron.

An elegant linear time solution for LP in 2 and 3 dimensions has been presented by [Dyer 84] and independently by [Megiddo 83a]. They managed to achieve this efficiency by not constructing the convex hull of F. Instead, during each iteration, a fixed fraction of the remaining constraints are pruned away. Thus the cost of each iteration decreases in a geometrical progression leading to the linear time result. This approach will be discussed in more detail in Section 5.4.

[Megiddo 84] has shown that the $O(n)$ 3-dimensional LP result can be generalized to solve any LP problem of fixed dimension in linear time. The time-complexity of his algorithm is $O\left(2^{2^k} n\right)$, where k is the dimension of the problem and n is the number of constraints. [Dyer 86] and independently, [Clarkson 86] improved this result so that the time-complexity is $O\left(3^{(k+1)^2} n\right)$. Notice that even for $k = 2$ or 3, the above constants are quite large.

5.3 The ICT Approach

As might be expected, LP will be solved by constructing an initial solution region that encloses the optimal solution; each iteration the volume of the remaining solution region will be reduced by at least a fixed fraction until the termination predicate has been satisfied. Each of these steps will be discussed in detail before the algorithm is presented in Section 5.4.

5.3.1 Constructing The Initial Solution Region

The task of creating an initial solution region for LP has been unexpectedly challenging, even though only 2 and 3-dimensional LP problems have been considered. Most algorithms that solve LP do not need to bound the solution region. One exception is the ellipsoid method of [Khachiyan 79]. The following description has been taken from [Chvátal 83], pages 447–448.

Let
$$\sum_{j=1}^{k} a_{ij} x_j \le b_i, \quad \text{where } i = 1, \ldots, n$$

represent the n constraints of the problem, each having k variables. If the problem has any solution at all, then it has a solution such that:

$$-2^{D} \leq x_j \leq 2^{D}, \quad \text{where } j = 1, \ldots, k \qquad [5.2]$$

with D standing for the total number of binary digits in the $n(k+1)$ integers a_{ij} and b_i. Thus the polyhedron defined by [5.2] will enclose the optimal solution if there is one. Notice that even when $k \leq 3$ the value of D can be very large since it is dependent on the number of constraints.

A different approach will be used to construct the initial ICT solution region. Briefly, the region will be constructed by intersecting a subset of at most $4(k-1)$ constraints, where $k \leq 3$. Clearly this is an improvement since the size of the solution region is not dependent upon the number of constraints. However, there is a drawback. Recall from Figure 5.1.a that even when an LP problem is considered to be bounded, the set of feasible points need not be. If the intersection of all of the constraints is unbounded, then clearly initial region will also be unbounded, which plays havoc with any claims of convergence. This situation can be handled in one of two ways: either the algorithm can add a constraint to the problem which results in a bounded solution region, without affecting the optimal solution, or else the algorithm can terminate, allowing the user to add the required constraint. The latter approach has been adopted in this thesis.

Thus the main result of this section is that the boundedness of the set of feasible points can be determined by mapping each constraint to a point on a unit sphere; the centre of this sphere will be inseparable from the mapped points if, and only if the set is bounded in all directions (see Theorem 5.4). First the mapping of the constraints to points on the unit sphere will be described, followed by Lemma 5.2 and 5.3, which describe tests that enable us to determine if the mapped constraints are bounded in a particular direction. Finally the main result of the section (Theorem 5.4) is proven.

First consider the two half-planes shown in Figure 5.2.a. It is easy to see that their intersection is bounded from above by any line that is parallel to a line that supports the intersection at **a**. Notice that it is not necessary to know the location of the half-planes in order to determine this information. In other words, each half-plane can be arbitrarily translated without affecting the set of directions that their intersection is bounded in. Each 2-dimensional constraint will be translated so that its boundary is tangent to a unit circle centered at the origin and such that the origin lies in the interior of the half-plane (Figure 5.2.b). This mapping has two side-effects: first, it transforms infeasible problems into feasible ones (for example, see Figure 5.3), and second, each non-redundant constraint now contributes one edge to the feasible region. We will ignore both of these side-effects since in the end, the solution region will be constructed by intersecting the original, untranslated constraints. If the intersection of these constraints turns out to be empty, then the LP problem is infeasible.

Figure 5.2 Translating The Constraints

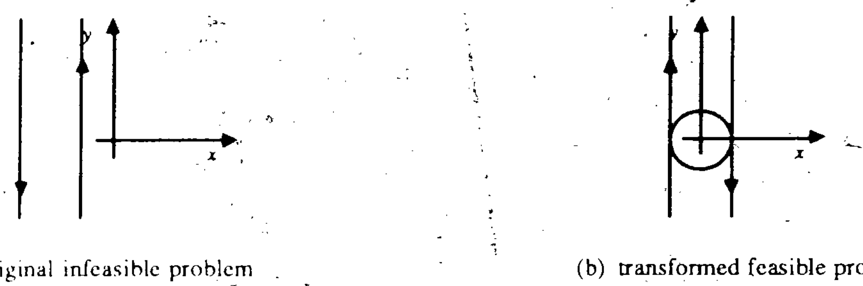


Figure 5.3 An infeasible problem is transformed into a feasible problem by the translation.

It is not difficult to see that a similar mapping can be applied to two half-spaces without affecting the set of directions that their intersection is bounded in. In this case, each constraint is translated so that its boundary is tangent to the unit sphere centered at the origin and such that the origin lies in the interior of the half-space.[1]

Lemma 5.1 The set of feasible points for a 2 or 3-dimensional LP problem is bounded if, and only if, the intersection of translated constraints is bounded, provided that the set of feasible point is non-empty.

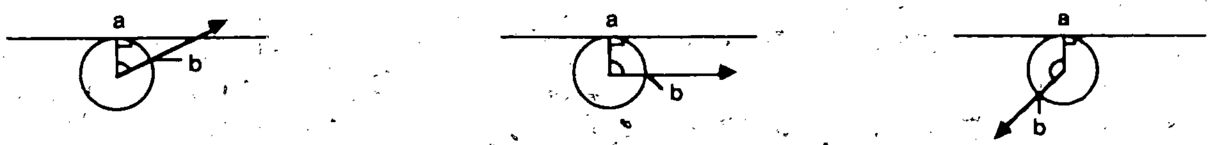
---

1 This is similar to the first step of the mapping used by [Guibas, Stolfi and Clarkson 87], which was mentioned in Section 4.2.

This follows from what has been said above. Before presenting Theorem 5.4, which describes the test that will be used to determine whether the set of translated constraints is bounded or not, two lemmas that will be used to prove this theorem will be introduced.

Lemma 5.2 Consider a unit circle that is centered at the origin and let λ denote a line that is tangent to this circle at the point $\mathbf{a}$. Now consider a ray whose endpoint coincides with the origin, and which intersects the unit circle at the point $\mathbf{b}$. This ray will intersect λ if, and only if, the length of the shorter arc connecting $\mathbf{a}$ and $\mathbf{b}$ is less than $\frac{\pi}{2}$.

Figure 5.4 illustrates each of the three possibilities. Notice that the arc length is the same as the angle that is shown since the circle has unit radius.



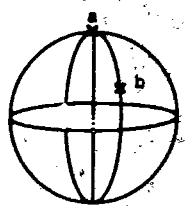
(a) arc length $< \frac{\pi}{2}$ (b) arc length $= \frac{\pi}{2}$ (c) arc length $> \frac{\pi}{2}$

Figure 5.4 Illustrating the three cases of Lemma 5.2.

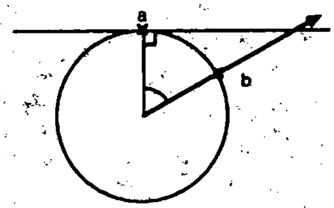
Now consider the 3-dimensional case of this lemma. Two distinct points on a sphere which are not extremities of a diameter lie on one and only one great circle (for example, $\mathbf{a}$ and $\mathbf{b}$ in Figure 5.5.a). The shorter arc connecting these two points is the shortest curve on the surface of the sphere that connects them.

Lemma 5.3 Suppose that the plane λ is tangent at the point $\mathbf{a}$ to the unit sphere centered at the origin. Let $\mathbf{b}$ denote the point where a ray whose endpoint coincides with the origin intersects the unit sphere. This ray will intersect λ if, and only if the length of the shortest arc on the surface of the sphere connecting $\mathbf{a}$ and $\mathbf{b}$ is less than $\frac{\pi}{2}$.

Proof: Consider the plane defined by the points $\mathbf{a}$ and $\mathbf{b}$ and the origin. Its intersection with λ results in a line that is tangent to the unit sphere at the point $\mathbf{a}$, and, its intersection with the unit circle results in a great circle that has **a** and **b** lying on its circumference. Furthermore, notice that the ray in question also lies in this plane. Thus by Lemma 5.2, the ray will intersect λ if, and only if, the length of the arc connecting **a** and **b** is less than $\frac{\pi}{2}$ (see Figure 5.5.b). Since this arc lies on a great circle, it must be the shortest arc on the surface of the sphere that connects **a** and **b**. ●



(a) two points on a great circle



(b) a 2-dimensional view of the intersection

Figure 5.5 Illustrating Lemma 5.3.

We will now use the above lemma to show that we can test whether Φ, the intersection of the translated constraints is bounded or not. Since the approach applies equally well to both 2 and 3 dimensions, we will describe only the 3-dimensional case.

Theorem 5.4 Let U denote the set of points at which the n translated constraints are tangent to the unit sphere. Φ is bounded if, and only if the origin is inseparable from U.

Proof: Φ is unbounded if, and only if, it contains a ray ([Grünbaum 67], page 23). Assume that Φ is bounded, but that the origin is separable from U. As we saw in the previous chapter, this means that there exists a plane that passes through the origin that has all the points of U lying either on the plane or in one of the two half-spaces defined by the plane. Without loss of generality, assume that all the points of U lie on or above the plane $z = 0$. Consider the ray whose endpoint coincides with the origin and passes through the point **b** = (0, 0, -1). (See Figure 5.6.) Clearly the shortest arc on the surface of the sphere that connects **b** to any of the points of U must be greater than or equal to $\frac{\pi}{2}$. Therefore, by Lemma 5.3, the ray does not intersect any of the tangent planes that are associated with the points of U. However, this contradicts our assumption that Φ is bounded, since the boundary of Φ is determined by these planes. Therefore the origin is inseparable from U whenever Φ is bounded.

Now assume that the origin is inseparable from U, which means that the origin lies in the interior of the convex hull of no more than $(2 * k) = 6$ points of U [Gustin 47]. Furthermore, assume that Φ is not bounded. This means that there exists some ray whose endpoint coincides with the origin that does not intersect any of the tangent planes. Consider the point where this ray intersects the unit sphere. From Lemma 5.3, we know that the length of the shortest arc on the surface of the sphere that connects this point with any of the points of U must be greater than or equal to $\frac{\pi}{2}$. However, this means that all the points of U must lie in one hemisphere, which contradicts our assumption that the points are inseparable from the origin. Therefore Φ is bounded whenever the origin is inseparable from U. ♦

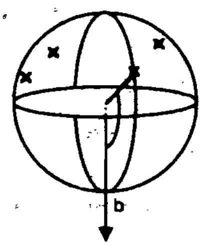


Figure 5.6 Illustrating Theorem 5.4

The routine 'PointInSet3D' (see Chapter 4) can be used to determine whether the origin is inseparable from U. Recall that not only does this routine return the separability of the origin from the set, but it also returns the following information: if the origin is either strictly or weakly separable, then 'PointInSet3D' returns either a wedge or a half-space that contains the points of U; if the set is inseparable from the origin, then it returns a maximum of $4(k-1)$ points of U such that the origin is interior to the convex hull of this subset. The former information can be used to provide the user with some indication of the direction that problem is unbounded in; the latter will be used to construct the initial solution region. By Theorem 5.4 and Lemma 5.1, the intersection of the constraints that defined this subset of points is bounded. The different stages of the process requires time as follows:

- $O(n)$ time is required to map the original constraints to U;

- the routine PointInSet3D requires $O(t_0 n)$ time to test whether the origin is inseparable from U, where t_0 is the total number of iterations of the algorithm;

- $O(1)$ time is required to intersect a maximum of $4 * (k - 1) = 8$ constraints.

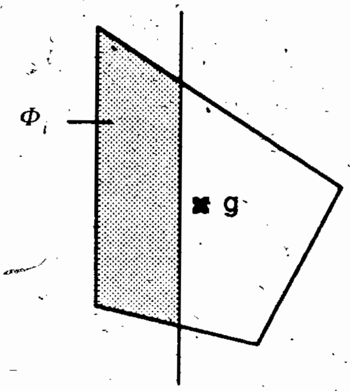
Therefore, the initial solution region can be constructed in

$$\mathcal{Maximum}\ \{ O(n), O(t_0 n), O(1) \} \quad ,$$

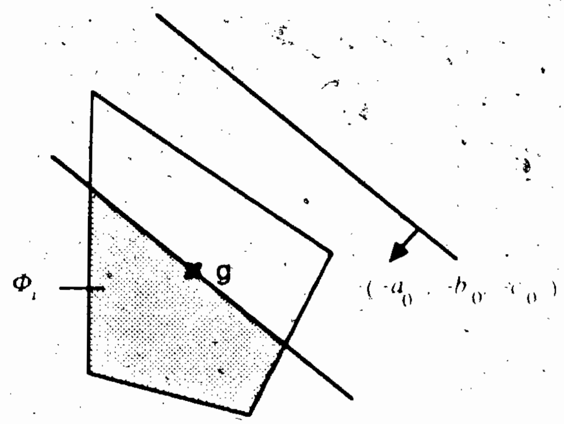
that is, in $O(t_0 n)$ time.

5.3.2 The Iteration Step

It is the responsibility of the iteration step to ensure that the volume of the solution region is reduced by a fixed fraction each iteration without discarding the optimal solution in the process. Suppose that the solution region has been reduced by intersecting it with one of the constraints of the LP problem. Clearly the optimal solution will not discarded in this case since it lies in the intersection of all of the constraints. Furthermore, as long as **g** (the centre of gravity of current solution region) lies in the region that is discarded, then from Theorem 2.6 it follows that the volume of the solution region will be reduced by at least a fixed fraction. (For example, see Figure 5.7.a .) But what if **g** is a feasible point and there is no such constraint? In this case, the solution region can be cut by a plane that is parallel to the objective supporting plane and which passes through **g**. Such a plane will divide the current solution region into two regions, one with objective values greater than that of **g** and one (containing the optimal solution) with objective values less than that of **g**. The former region can be discarded. (Figure 5.7.b illustrates such a case, assuming that objective function is being minimized.) Since **g** lies on the boundary of the half-space, this ensures that the volume of the next solution region will at most be a fixed fraction of that of the current region.



(a) **g** is infeasible

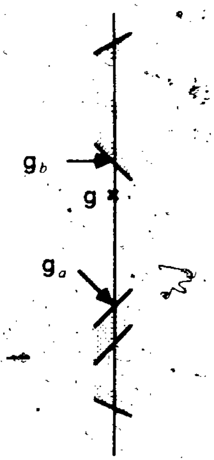


(b) **g** is feasible

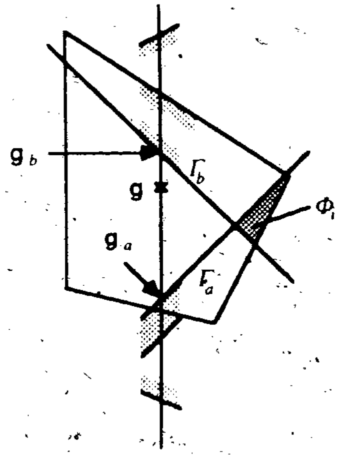
Figure 5.7 Reducing the solution region for LP

Although the above approach works reasonably well, it is possible to achieve greater control over the convergence of the algorithm with some more effort. The main result of this section is that given an arbitrary line, it is possible to ensure that the next solution region will extend to only one side of a plane that contains this line. Thus, if the algorithm detects that the solution region is not converging in a particular direction, then a line perpendicular to this direction will be supplied, ensuring convergence in that direction. Furthermore, if the line passes through the centre of gravity of the current solution region, then it follows from Theorem 2.6 that volume of the next solution will be a fixed fraction of that of its predecessor.

A 1-dimensional LP problem will be constructed and solved in order to determine the half-spaces that will be used to reduce the solution region. The 1-dimensional problem is formed by intersecting the given line with the original constraints. The same objective function is used for both problems; the 1-dimensional problem is feasible if the given line intersects the set of feasible points for the original problem. Some examples are shown in Figure 5.8 and 5.9. Note that in these diagrams, it is assumed that the objective supporting plane is horizontal, bounding the set of feasible points from below.

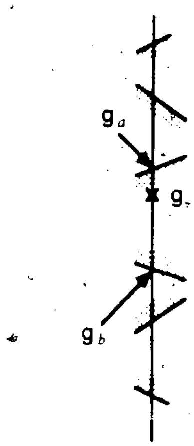


(a) an infeasible 1 dimensional problem

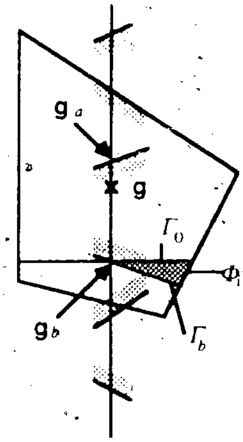


(b) cutting the solution region

Figure 5.8 An infeasible 1-dimensional LP problem.



(a) a feasible 1-dimensional problem

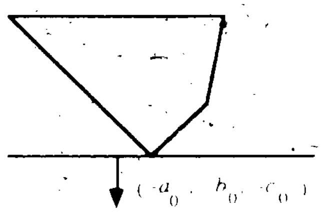


(b) cutting the solution region

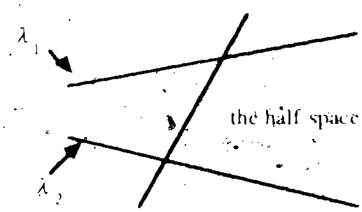
Figure 5.9 A feasible 1-dimensional LP problem.

The 1-dimensional problem shown in Figure 5.8.a is infeasible and the constraints that prove this to be the case will be used to reduce the volume of the current solution region (see Figure 5.8.b). In Figure 5.9.a, the feasible point that minimizes the objective function ($\mathbf{g}_b$) is identified. Since $\mathbf{g}_b$ is a feasible point of the original LP problem, a half-space whose boundary is parallel to the objective supporting plane will be used to reduce the volume of the current solution region, along with the constraint that defined this point.

Some definitions and notation will be introduced before the algorithm is presented. Without loss of generality, assume that the objective supporting plane is horizontal, bounding the set of feasible points from below (see Figure 5.10.a). Consider an arbitrary line λ. λ partitions the constraints of the LP problem into three groups: those whose boundaries are parallel to the line; those that contain one end of the line and those that contain the other end of the line. For clarity, some notation will be introduced that distinguishes these three groups. If λ is not horizontal, then one end extends to $+\infty$ in the z-direction and the other extends to $-\infty$. The former will be referred to as the *top* of the line while the latter as the *bottom*. If λ is horizontal, then it is assumed that top and bottom are assigned to the two ends of the line in some systematic fashion. Now consider a half-space that represents a constraint of the LP problem. If λ is parallel to the boundary of this half-space, then the constraint is *parallel* to λ; if it contains the top of λ, then the constraint is *bounded from below* with respect to λ and if it contains the bottom then it is *bounded from above* with respect to λ. Thus the constraint shown in Figure 5.10.b is bounded from below with respect to λ_1 and bounded from above with respect to λ_2.



(a) Objective supporting plane



(b) the half space determined by the constraint

Figure 5.10

The following algorithm constructs and solves the 1-dimensional problem in time linear to the number of constraints. In addition it reduces the volume of the current solution region, Φ_{i-1}, by intersecting it with the selected constraints. After an analysis and discussion of the algorithm, a proof will be given that argues that the reduced solution region extends ~~to~~ only one side of a plane that contains λ. Note that in the following that η_a and η_b denote the half-lines that result from respectively intersecting the constraints that are bounded from above and below with λ. For example, in Figures 5.8 and 5.9, $\mathbf{g}_a$ is the endpoint of η_a and $\mathbf{g}_b$ is the endpoint of η_b.

Algorithm 5.1 : Reducing the volume of the solution region for 3D LP.

ReduceRegion(λ , Φ_{i-1} ; VAR Φ_i , feasiblePointFound , optimal);

1 Let η_a η_b - λ ;

2 feasiblePointFound := false ;

3 For each constraint Γ {

3.1 If Γ is parallel with respect to λ ,

3.2 then { if λ is not contained by Γ , then { $\Phi_i := \Phi_{i-1} \cap \Gamma$; return }

3.3 else if Γ is bounded from above with respect to λ, then $\eta_a = \eta_a \cap \Gamma$

3.4 else $\eta_b = \eta_b \cap \Gamma$ }

4. Let $\mathbf{g}_a$ and $\mathbf{g}_b$ denote the endpoints of η_a and η_b respectively and let Γ_a and Γ_b denote the constraints that defined $\mathbf{g}_a$ and $\mathbf{g}_b$, respectively.

5. case 1 $\eta_a \cap \eta_b$ is empty /* In this case the 1-dimensional problem is infeasible. */

Φ_i $\Phi_{i-1} \cap \Gamma_a \cap \Gamma_b$; /* see Figure 5.8 /*

case 2 $\eta_a \cap \eta_b$ is not empty /* see Figure 5.9 /*

/* In this case, each point of $\eta_a \cap \eta_b$ is a feasible point for the original LP problem. */

feasiblePointFound := true ;

optimal := $\mathbf{g}_b$;

If the boundary of Γ_b is parallel to the objective supporting plane,

then set Φ_i to the single point $\mathbf{g}_b$.

Otherwise,

{ a plane drawn through $\mathbf{g}_b$ that is parallel the objective supporting plane will cut the set of feasible points into two regions; the objective values of one region will be greater than the objective values of the other region. Let Γ_0 denote the half-space that contains the region with smaller objective values. $\Phi_i := \Phi_{i-1} \cap \Gamma_0 \cap \Gamma_b$; }

6. return ;

end of algorithm

Analysis and Discussion of Algorithm 5.1

Assume that either the original LP problem is infeasible or else that Φ_{i-1} contains the optimal solution:

1. Notice that feasiblePointFound is initialized to false on line 2. The only time it is set to true is in case2 of step 5 after a feasible point has been found. Note that if a feasible point has been found, then $\mathbf{g}_b$ is the optimal solution for the 1-dimensional LP problem.

2. Step 3 constructs and solves the 1-dimensional LP problem. It begins by examining each of the constraints and determining if it is parallel, bounded from above or bounded from below with respect to λ. It is easy to see that there must be at least one constraint that is bounded from above and one that is bounded from below since, as was shown in the previous section, the initialization process ensures that the set of feasible points is bounded in all directions. Line 3.2 catches the case where a parallel constraint does not contain λ. Once such a case has been identified, there is no need to look at any more constraints since λ cannot intersect the set of feasible points. Lines 3.3 and 3.4 are responsible for determining η_a and η_b.

3. Consider case 1 of Step 5. The 1-dimensional problem is infeasible, so the constraints that prove this to be the case (Γ_a and Γ_b) are used to reduce the solution region. Note that Φ_i may be empty as a result of this intersection.

4. Now consider case 2 of Step 5. First consider the case where Γ_b is parallel to the objective supporting plane. Clearly, λ is not horizontal since otherwise Step 2.1 would have determined that Γ_b is a parallel constraint. It is easy to see that $\mathbf{g}_b$ (which lies in Γ_b) is the point of $\eta_a \cap \eta_b$ that has the minimum z-value. Furthermore, there cannot be a feasible point with a lower z-value since Γ_b is bounded from below. Therefore the algorithm concludes that $\mathbf{g}_b$ is an optimal solution for the LP problem. This is indicated to the calling routine by returning Φ_i as the single point $\mathbf{g}_b$.

   Now consider the case where Γ_b is not parallel to the objective supporting plane. Notice that this is the one place where Φ_{i-1} is intersected with a half-space that is not a constraint of the problem. Clearly, Γ_0 contains the optimal solution since its boundary passes through a feasible point and it extends in the direction that minimizes objective values.

5. Since the volume of the solution region is reduced by intersecting with half-spaces that contain the optimal solution, the optimal solution will not be discarded by this algorithm.

6. Note that it is possible that either Γ_a or Γ_b has been used in a previous iteration to diminish the size of the solution region, and hence will not reduce it any further. An example of this is shown in Figure 5.11. The diagram illustrates two successive calls to Algorithm 5.1. In both cases, λ is a vertical line that intersects the solution region and in both cases, Γ_b is the same constraint.

Nothing is gained by intersecting Γ_b with the solution region the second time around. This constraint cannot be deleted after its first use since otherwise we could not guarantee that $\mathbf{g}_b$ is a feasible point in case 2 of Step 4 . Instead, the intersection routine (Section C.4) should be modified so that it tags each constraint that is intersected with the solution region. That way, the routine can detect whether it has encountered a constraint previously, and ignore it if it has. Note that since Γ_0 is constructed afresh each time, it will always be used to reduce the solution region.

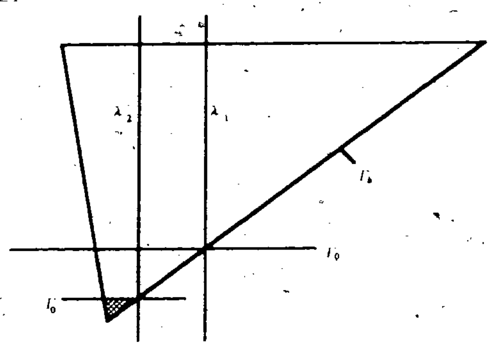


Figure 5.11 Illustrating two successive calls to Algorithm 5.1 .

Now consider the running time of Algorithm 5.1 . It is easy to see that this algorithm will increase the number of faces of solution region by at most two. Therefore each intersection operation can be performed in $O(f_i)$ time, where f_i is the number of faces of Φ_{i-1} . Thus the total running time for the algorithm is $O(\, \mathit{Maximum} \{ O(n), O(f_i) \})$, where n is the number of constraints of the problem.

Theorem 5.5 Algorithm 5.1 ensures that there exists a plane that contains λ such that Φ_i extends to only one side of this plane.

Proof: In order to prove the theorem, it will be shown that there exists at least one plane that separates Φ_i from λ . Consider each of the ways of constructing Φ_i .

(1) On line 3.2 , Φ_i is constructed by intersecting Φ_{i-1} with a constraint whose boundary is parallel to λ but does not contain λ . Clearly, the boundary of this half-space will separate Φ_i and λ .

(2) If the boundaries of Γ_a and Γ_b are parallel to each other in case 1 of Step 5 , then Φ_i will be empty and hence the theorem is trivially true. Otherwise, $\Gamma_a \cap \Gamma_b$ will define a wedge

that does not intersect λ. (Otherwise, $\eta_a \cap \eta_b$ would not be empty.) Therefore, $\Phi_i = \Phi_{i-1} \cap \Gamma_a \cap \Gamma_b$ will not intersect λ either, which means there exists a plane that separates Φ_i from λ.

(3) Finally, consider case 2 of Step 5. Clearly, if Φ_i is set to the single point $\mathbf{g}_b$ (when the boundary of Γ_b is parallel to the objective supporting plane) then the theorem is trivially true. Therefore assume that this is not the case, but instead $\Phi_i = \Phi_{i-1} \cap \Gamma_0 \cap \Gamma_b$. Two situations arise.

(a) If the line λ is horizontal, then the boundary of Γ_0 contains λ and since Γ_0 extends to only one side of this line, the same will be true of Φ_i.

(b) Assume that λ is not horizontal. In this case, Γ_0 is bounded from above with respect to λ. It is clear from above that the boundary of Γ_b is not parallel to the objective supporting plane, and hence is not parallel to the boundary of Γ_0. Thus $\Gamma_0 \cap \Gamma_b$ is a wedge and the point $\mathbf{g}_b$ is a point on the edge of this wedge. Thus the point $\mathbf{g}_b$ divides λ into two half-lines, one of which is completely contained by Γ_0 and one of which is completely contained by Γ_b. Thus $\Gamma_0 \cap \Gamma_b \cap \lambda$ is the point $\mathbf{g}_b$, which means that there exists a plane that weakly separates λ from this wedge. Since Φ_i is constructed by intersecting Φ_{i-1} with this wedge, clearly this same plane will separate Φ_i from λ.

Thus the theorem holds for each of the cases. ♦

5.3.3 The Termination Predicate

Let $\mathbf{x}^*$ denote the optimal solution for the LP problem and let ε denote a parameter specified by the user. In this section, the following termination conditions are discussed:

(1) $\quad | \mathcal{F}(\mathbf{x}^*) - \mathcal{F}(\mathbf{g}) | < \varepsilon$

(2) $\quad | \mathbf{x}^* - \mathbf{g} | < \varepsilon$.

Consider (1) first. Let $\mathcal{F}(\mathbf{p})$ denote a function that returns the value of the objective function at the point $\mathbf{p}$ and let $\mathbf{h}$ and $\mathbf{l}$ denote the points of the current solution region that respectively minimize and maximize this function. Recall that $\mathbf{x}^*$ lies somewhere within the solution region as long as the LP problem is feasible. Therefore, $| f(\mathbf{g}) - f(\mathbf{x}^*) | \leq | f(\mathbf{h}) - f(\mathbf{l}) |$. Clearly $\mathbf{h}$ and $\mathbf{l}$ can be determined in $O(v)$ time, where v is the number of vertices of the solution region. Therefore the algorithm can terminate once $| f(\mathbf{h}) - f(\mathbf{l}) | \leq \epsilon$. Now consider case (2). It is easy to see that this case is satisfied if $\Phi_i \subseteq Sphere(\mathbf{g}, \epsilon)$. This can be checked by simply testing whether each vertex of Φ_i is within $\mathbf{g}$ of ϵ. Hence this test also requires $O(v)$ time.

Note that a small solution region does not indicate that a problem is either feasible or infeasible. The only way to tell for sure is to have either encountered a feasible point during one of the calls to Algorithm 5.1 or else to have the solution region completely disappear. For some applications this may not be a problem since it may be known in advance that every problem is feasible. If this distinction is crucial to the application, then the tests described by (1) and (2) above should not be made until a feasible point has been found. Algorithm 5.1 helps to reveal infeasible problems by optimizing the choice of constraints used to reduce the solution region. For example, in Figure 5.12, the constraint Γ_a will be chosen to reduce the solution region, which will result in it immediately being set to empty. If it is not desirable to wait until a feasible point has been found, then the approach described by Algorithm 5.2 could be used instead.

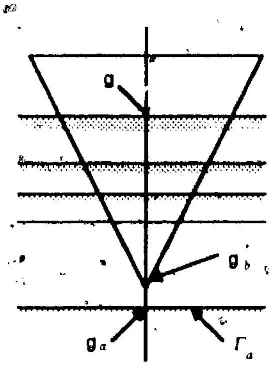


Figure 5.12. An infeasible LP problem.

Algorithm 5.2 : Testing whether a small solution region has a feasible solution.

Construct a 2-dimensional LP problem with arbitrary objective function as follows. Let ρ denote a plane that is parallel to the objective supporting plane and which passes through a point that is half way between $\mathbf{h}$ and $\mathbf{l}$. Construct the initial solution region for the new LP problem by intersecting Φ_i and ρ. Similarly, construct the constraints for the new problem by intersecting the original constraints with ρ. If ρ does not intersect each of the constraints, then conclude that the original problem is infeasible. Apply ICT to this new problem, terminating as soon as a feasible point has been found or when $| \mathcal{F}(\mathbf{l}) - \mathcal{F}(\mathbf{h}) | \le \varepsilon$. If a feasible point has been found, then clearly, the original problem is feasible. Otherwise, use a similar technique to recast the 2 dimensional problem as a 1-dimensional one. If no feasible point is found, then conclude that the problem is infeasible.

– end of algorithm –

Analysis and Discussion of Algorithm 5.2

Assume that n is the number of constraints of the problem.

1. The initial (planar) solution region can be constructed by intersecting the current solution region by two half-spaces that share the same boundary plane but extend to opposite sides of this plane. The intersection of a convex polyhedron and a half-space can be determined in $O(f)$ time, where f is the number of faces of the polyhedron (Section C.4). Therefore the initial solution region can be constructed in $O(f)$ time. As will be shown in the next section, the solution region for LP can have at most $(n + 1)$ faces. Therefore the initialization step requires at most $O(n)$ time.

2. ICT can solve the 2-dimensional LP problem in $O(t_1 n)$, where t_1 is the number of iterations of the algorithm.

3. As was seen in the previous section, a 1-dimensional LP problem can be solved in $O(n)$ time.

Thus the total running time for Algorithm 5.2 is $O(t_1 n)$. Note that it is still possible that the algorithm will lead us to conclude that a problem is infeasible when in fact it is feasible. This situation arises when λ does not intersect the set of feasible points. The likelihood of such an occurrence will depend upon the application. Once again, if this is not a suitable approach, then the termination test should not be made until at least one feasible point has been found.

5.4 The ICT Algorithm For 3-Dimensional LP

Finally it is time to summarize the above results by presenting the ICT algorithm for LP. Since the approach applies equally well in both 2 and 3 dimensions, only the 3-dimensional problem will be considered. It is assumed that Algorithm 5.3 should terminate once $| \mathbf{x}^* - \mathbf{g} | < \varepsilon$. As will be seen, the routine 'ReduceRegion' (Algorithm 5.1) is called twice each iteration, once with a line that passes through the point $\mathbf{g}$ (the centre of gravity of the current solution region) and once with a line that passes through $\mathbf{h}$, the point of the current solution region that minimizes the objective function. The former call to 'ReduceRegion' ensures that the volume of the solution region is reduced by a fixed fraction each iteration. The latter call is an optimization step. If $\mathbf{h}$ turns out to be feasible, then the algorithm terminates immediately, since $\mathbf{h}$ is the optimal solution for the LP problem. If it is not feasible, then the region will be further reduced, which should help to reveal the optimal solution sooner.

<u>Algorithm 5.3 : 3-dimensional LP</u>

1. Initialization Step

1.1 Without loss of generality, assume that the objective supporting plane is horizontal, bounding the set of feasible points from below.

1.2 Construct the initial solution region Φ_0 using the method described in Section 5.3.1 . If the solution region is unbounded in some direction, then terminate, reporting this to the user.

1.3 If Φ_0 is empty, then terminate, reporting that the problem is infeasible.

1.4 foundFeasiblePoint := false;

1.5 slopeOfLine := vertical ;

2. Iteration Step ($i \geq 1$)

2.1 Let $\mathbf{h}$ denote a point of Φ_i that minimizes the objective function.

2.2 Let $\lambda_{\mathbf{h}}$ denote a vertical line through the point $\mathbf{h}$.

2.3 ReduceRegion($\lambda_{\mathbf{h}}$, Φ_{i-1} ; VAR Φ_i , feasible , $\mathbf{h}_b$);

2.4 If Φ_i is empty, then terminate, reporting that the problem is infeasible.

2.5 If feasible then

2.6 { if $\mathbf{h}_b == \mathbf{h}$

2.7 then { terminate, reporting that $\mathbf{h}$ is the optimal solution }

2.8 else { foundFeasiblePoint := true }

}

2.9 $\mathbf{g}$ = COG(Φ_{i+1}) .

2.10 Let $\lambda_{\mathbf{g}}$ denote a line that passes through the point $\mathbf{g}$ that has slope 'slopeOfLine'

2.11 ReduceRegion($\lambda_{\mathbf{g}}$, Φ_i , VAR Φ_i , feasible , $\mathbf{g}_b$).

2.12 If Φ_i is empty, then terminate, reporting that the problem is infeasible

2.13 If feasible then foundFeasiblePoint := true .

3. Termination Predicate

3.1 Find the vertex of Φ_i that is farthest from $\mathbf{g}$. If the distance between these two points is greater than ε then set slopeOfLine so that it is normal to the line passing through these two points. Continue to iterate.

3.2 Otherwise {

3.3 if foundFeasiblePoint then report that $\mathbf{g}$ is the approximate solution

3.4 else { Execute Algorithm 5.2 to determine if a feasible point can be found

3.5 If so, then report that this point is the approximate solution

3.6 Otherwise report that the problem is infeasible }

3.7 Terminate algorithm. }

end of algorithm

Discussion And Analysis Of Algorithm 5.3

Let n denote the number of constraints of the problem. Recall that the routine 'ReduceRegion' is Algorithm 5.1 .

1. Line 1.1 has been included to ease the discussion of the algorithm. If the objective supporting plane is not horizontal bounding the set of feasible points from below, then the constraints can be rotated so that this is the case in O(n) time. If the rotation is not performed, then terms like 'top' and 'bottom' (see Section 5.3.2) will need to be defined more carefully, so as to reflect the orientation of the objective supporting plane.

2. The construction of the initial solution region (line 1.2) has been discussed in Section 5.3.1 . Recall that Φ_0 will be constructed by intersecting at most 8 half-spaces. Identification of these half-spaces requires O(t_0 n) time, where t_0 is the number of iterations of the PointInSet3D routine. The intersection of at most 8 half-spaces can be performed in constant time. Thus the entire step requires O(t_0 n) time.

3. It is possible to determine if the current solution region is empty in constant time. An empty solution proves that the problem is infeasible (see Section 5.3.2). Therefore the program can terminate, as is shown in line 1.3.

4. The variable 'foundFeasiblePoint' is simply a flag that records whether a feasible point has been encountered during the execution of the algorithm. Notice that it is set to false on line 1.4 and is only is only set to true if the routine 'ReduceRegion' reports that a feasible point has been found (lines 2.8 and 2.13). This flag is tested on line 3.3 to determine if a feasible point has been encountered during the course of the algorithm.

5. The variable 'slopeOfLine' describes the slope of the line that should be passed to ReduceRegion to ensure the solution region converges in all directions. Initially it is vertical, but the direction is reset on line 3.1 based upon knowledge of the point of the solution region that is farthest from $\mathbf{g}$. Note that had a different termination predicate been used, for example,

$$|\mathcal{F}(\mathbf{x}^*) - \mathcal{F}(\mathbf{g})| < \varepsilon \quad \text{instead of} \quad \|\mathbf{x}^* - \mathbf{g}\| < \varepsilon$$

(see Section 5.3.3), then it would be reasonable to use a line with the same slope each iteration. In this case, the initialization step could preprocess the constraints into three groups, those bounded from above, those bounded from below and those parallel to the line (see Section 5.3.2). This information could be supplied to the routine 'ReduceRegion', saving that routine the work of partitioning the constraints. Since this routine is called twice each iteration, such an optimization would enhance the overall performance of the algorithm.

6. Now consider the number of faces of the solution region. The initial solution region, Φ_0, will have at most 8 faces. Each call to 'ReduceRegion' may increase this number by at most two. Since ReduceRegion is called twice per iteration, the number of faces of the solution region will be $O(i)$ during the i^{th} iteration. However, recall from Section 5.3.2 that the solution region is reduced by either intersecting it with a constraint of the problem or else with a half-space whose boundary is parallel to the objective supporting plane. Therefore the solution region will have most $n + 1$ faces.

7. The call to ReduceRegion (line 2.3 and 2.11) requires $O(\mathit{Maximum}\{O(n), O(f_i)\})$ time to reduce the solution region (Section 5.3.2), where f_i is the number of faces of the solution region. Since the solution region can have at most $O(n)$ faces, each call to ReduceRegion requires $O(n)$ time. Notice that after each call, the algorithm tests to see whether the new solution region is empty. If it is, then the algorithm terminates

immediately, since the problem has proven to be infeasible (line 2.4 and 2.12). This test can be performed in constant time. Notice that the first call to ReduceRegion (line 2.3) tests the line passing through **h**. If it is determined that **h** is a feasible point then the algorithm terminates immediately since **h** is the optimal solution (lines 2.5 to 2.7).

8. The centre of gravity of the solution region (line 2.9) can be found in time linear to its number of faces (Section C.3). Therefore it can be determined in $O(n)$ time.

9. On line 3.1, each vertex of the solution region is checked in order to find the one that is farthest from **g**. Since the number of vertices of a convex polyhedron is linearly related to its number of faces, this step can be performed in $O(n)$ time.

10. If the test on line 3.1 indicates that the solution region is sufficiently small, the algorithm will terminate. Before doing so however, it first decides whether the LP problem is feasible or infeasible. The flag 'foundFeasiblePoint' indicates if a feasible point has been encountered at some point during the execution of the algorithm. If one has, then problem is reported to be feasible (line 3.3). If none has been found so far, then Algorithm 5.2 constructs a 2-dimensional LP problem and searches for a feasible point. This requires $O(t_2 n)$ time, where t_2 is the number of iterations required by this algorithm. Thus in total, this step requires $O(t_2 n)$ time.

Thus in summary,

- $O(t_0 n)$ time is required for the initialization step.
- $O(n)$ time is required for the i^{th} iteration step
- $O(n)$ time is required to test if the solution region is small enough. This test is performed once per iteration.
- $O(t_2 n)$ time is required to test if the solution region contains a feasible point. This test will be performed at most once.

Thus a total of $O(tn)$ time is required to execute this algorithm, where $t = Maximum\{t_0, t_1, t_2\}$ and t_2 is the number of iterations performed by Algorithm 5.3. As was argued at the end of Section 1.3 and Section 3.2.1, the number of iterations of an ICT algorithm can be bounded from above by a constant provided that the algorithm is implemented using fixed-precision, floating point arithmetic

Under this assumption, t_0, t_1 and t_2 and hence t are bounded from above by a constant. In this case, the running time for Algorithm 5.3 is $O(n)$.

5.6 Exact Linear-Time Solution

In this section ICT will be combined with the prune-and-search technique for solving LP, which was introduced independently by [Dyer 84] and [Megiddo 83a]. The prune-and-search algorithm is an iterative procedure; in each iteration, a fixed fraction of remaining constraints are pruned away. First consider the 2-dimensional algorithm and the half-planes shown in Figure 5.13. It is not difficult to see that α can be discarded in Figure 5.13.a since it will never define the optimal solution. Similarly, if the optimal solution lies to the left of λ in Figure 5.13.b, then β can be discarded; if it lies to the right then α can be discarded and if it lies on λ then neither can be discarded since these may be the constraints that determine the optimal solution.

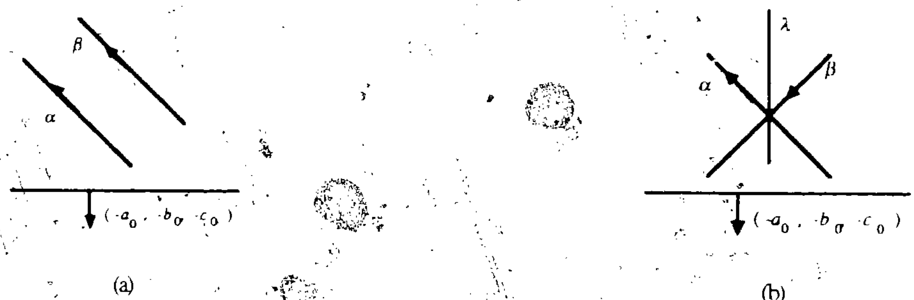


Figure 5.13 Identifying constraints that can be discarded.

Without loss of generality, assume that the objective supporting plane is horizontal, bounding the set of feasible points from below. Furthermore, assume that the constraints have been partitioned into three groups, S_a, S_b and S_v: each element of S_a is bounded from above by its boundary plane; each element of S_b is bounded from below, and S_v contains those constraints that are bounded by vertical planes. Let H_a, H_b and H_v denote the boundary planes for the constraints in S_a, S_b and S_v, respectively. Figure 5.14 describes the iteration loop for the 2-dimensional prune-and-search LP algorithm. (see Chapter 10 of [Edelsbrunner 87] for more details.)

Repeat (* prune-and-search iteration loop *)

1. Construct_Pairs : Arrange the elements of H_a in pairs, and the elements of H_b in pairs;
   If any pair is parallel to each other, then discard one of the constraints.
   Otherwise, determine the point of intersection of the two lines.
2. λ_x := Find_TST: Consider the set of intersection points, constructed in the previous step.
   Let **x** denote the point with median x-coordinate and let λ_x denote the vertical line that passes through **x**.
3. Bisect(λ_x) : Determine to which side of λ_x that the optimal solution lies on.
4. Prune(λ_x) : Discard one constraint from each pair, if possible.

until m constraints remain, where m is some constant ; Solve the problem directly.

Figure 5.14 Iteration Loop of the LP prune-and-search algorithm.

The point **x** can be determined in linear time ([Blum, Floyd, Pratt, Rivest and Tarjan 72] , [Schonhage, Paterson and Pippenger 76]) . However, as [Edelsbrunner 87] (page 239) has noted, the worst-case optimal methods for finding the median of a set of points all suffer from poor average case behaviour. Instead he suggests that the simpler algorithms presented in [Floyd and Rivest 75] be considered for implementation since they determine the median in a fast expected time. Bisect determines to which side of λ_x the optimal solution lies. In 2-dimensions, this usually involves solving one 1-dimensional LP problem, but in very degenerate cases, it may involve solving a total of three such problems. Prune performs the pruning that was described above. In each iteration, at least $\frac{1}{4}$ of the remaining constraints are pruned away, leading to the overall linear result.

The benefit of combining ICT with the prune-and-search algorithm is that it may eliminate the calls to both Find_TST and BISECT from each iteration. (The extra overhead of adding one iteration of the ICT algorithm is relatively small.) Since in this case there is no need to worry about degenerate convergence, Algorithm 5.3 will be modified to pass only vertical lines to 'ReduceRegion' (Algorithm 5.1) . Thus by Theorem 5.5 (see Section 5.3.2) , the current solution region and hence the optimal solution will always lie to one side of λ_g . Hence ICT can be combined with the prune-and-search technique as follows:

Algorithm 5.4 : Combining ICT and the prune-and-search technique.

Repeat

1. Perform one iteration of the iteration step for the 2-dimensional ICT algorithm for LP.
2. Prune(λ_g) ;
3. If less than $\frac{1}{4}$ of the remaining constraints have been pruned away, then {

    Perform one iteration of the repeat loop described in Figure 5.14 ;

    Let ρ denote the closed half-plane with boundary λ_x such that λ_x contains the optimal solution ;

    Let $\Phi_i := \Phi_i \cap \rho$;

    }

until m constraints remain, where m is some constant ;

Solve the problem directly.

- end of algorithm -

Notice that as the solution region gets very small, it is unlikely that step 3 will ever be executed. It may seem odd that we can combine ICT with a technique that discards constraints. Recall from Section 5.3.2 that it was stated specifically that this could not be done. The difference in this case is that the prune-and-search technique discards redundant constraints. Recall that in order for the ICT algorithm to behave correctly, $\mathbf{g}_b$ must a feasible point for the original problem in case 1 of Algorithm 5.1 . Discarding redundant constraints will not affect the choice of $\mathbf{g}_b$. For example, consider the half-planes shown in Figure 5.13.a once again. It is easy to see that α will never define $\mathbf{g}_b$, so it does not matter if this constraint is discarded. Now, consider Figure 5.13.b . It is easy to see from Algorithm 5.4 that the current solution region will always lie to one side of λ or to the other. Thus so will the all further $\mathbf{g}_b$. Hence it's choice will also not be affected by the discarded constraint.

The above approach also extends to 3-dimensions. The 3-dimensional prune-and-search algorithm for LP follows the same pattern that was described in Figure 5.14 , except that steps 2 - 3 are much more involved. We will not describe these steps, except to note that this time, Prune requires as input two vertical planes, whose intersection defines a wedge that contains the optimal solution. To determine this information, both Find_Tst and Bisect are called twice (each iteration) . In the 3-dimensional case, λ is a vertical plane and Bisect :

- determines if λ intersects the range that contains the optimal solution. This amounts to solving two 2-dimensional LP problems, whose constraints are defined by the intersection of S_v and the plane $z = 0$.
- determines $\mathbf{x^*}$, which amounts to solving a 2-dimensional LP problem whose constraints are defined by the intersection of λ and S_a and S_b.
- decides on which side of λ the optimal solution lies. This involves solving two 2-dimensional LP problems, whose constraints are the tight constraints which defined $\mathbf{x^*}$.

Hence, even if ICT is only combined with the 2-dimensional LP problem, the 3-dimensional algorithm is expected to run faster. However, it is easy to combine ICT with the 3-dimensional algorithm. As long as $\lambda_{\mathbf{g}}$ is vertical, Theorem 5.5 ensures that there exists a half-space that contains Φ_i such that the boundary of this half-space is vertical and contains $\lambda_{\mathbf{g}}$. Such a half-space can be determined in constant time by considering the constraints that ensure this property (see proof of Theorem 5.5). Let ρ_i denote such a half-space for Φ_i. A vertical wedge for Prune can be constructed by intersecting ρ_i with one of the half-spaces that defined the wedge for the previous iteration. If at least $\frac{1}{16}$ of the remaining constraints are discarded by this call to Prune, then the prune-and-search iteration step can be ignored this iteration. If not, the current solution region can be further reduced by intersecting it with the two half-spaces determined by the Find_Tst and Bisect routines.

Chapter 6

Conclusions

A new approximation technique has been proposed for solving geometric optimization problems in 2 and 3 dimensions. The technique, called Iterative Convergent Technique (ICT), converges to the optimal solution geometrically, terminating once the approximate solution is within ε of the optimal one, where ε is a parameter specified by the user. Two termination predicates have been considered:

$$(1) \quad \left| \mathcal{F}(\mathbf{x^*}) - \mathcal{F}(\mathbf{g}) \right| < \varepsilon \; ;$$

$$(2) \quad \left| \mathbf{x^*} - \mathbf{g} \right| < \varepsilon \; ,$$

where, $\mathbf{x^*}$ denotes the optimal solution, $\mathbf{g}$ denotes the approximate solution and $\mathcal{F}(\mathbf{p})$ represents some function that is meaningful to the problem, evaluated at that point $\mathbf{p}$. In addition, the question of degenerate convergence has been considered and handled separately for each of the problems listed below. Degenerate convergence, which is only a problem if termination predicate (2) is applied, arises when the solution region does not converge in all directions.

To illustrate the power of the technique, ICT has been applied to the following problems:

- detecting the common intersection of the convex hulls of m sets of points in 2 and 3 dimensions ;
- determining the separability of two planar sets ;
- Linear Programming (LP) in 2 and 3 dimensions ;
- finding the smallest enclosing sphere of n weighted points in 2 and 3 dimensions (SES) .

In the process, algorithms have been developed that can be used to solve the following problems:

- the extreme point problem in 2 and 3 dimensions ;
- origin point interior (determining whether the origin is extreme with respect to a set of points) ;
- hemisphere problem (determining if a set of points lies interior to some hemisphere) ;
- determining if the intersection of a set of half-spaces is bounded ;
- finding a hyperplane that separates a point from a set of points ;
- finding a hyperplane separating two sets .

This last set of problems are different applications of the separability problems discussed in Chapter 4 . All of these problems have been described in [Dobkin and Reiss 80] .

The time-complexity for LP is O($t\ n$), where t is the number of iterations performed. The time-complexity for the rest of the problems is O(t * $\mathcal{Maximum}$ (n , t)) . The size of t depends upon the volume of the initial solution region, ε , the precision of the machine (*macheps*) and the type of termination predicate used. It has been shown that t is bounded from above by a constant whenever fixed-precision floating point arithmetic is used to approximate real arithmetic. Under this assumption, which is currently the most common approach to representing real numbers, the running time for each algorithm is O(n) .

It has been demonstrated that ICT can be combined with the prune-and-search technique, developed independently by [Megiddo 83a] and [Dyer 84] . In addition, the application of ICT to SES demonstrates that it can be used to optimize a convex programming problem. Furthermore, a comparison of Algorithm 1.1 and Algorithm 3.1 illustrates the ease with which an algorithm for an unweighted 2-dimensional problem can be converted to a solution for a weighted 3-dimensional problem. This extensibility is one of the strengths of the ICT approach.

6.1 Other Problems That ICT May Be Applied To

It is conjectured that ICT can be applied to the following problems:

- the weighted Chebyshev or L_1 line fitting problem (after first applying Brown's Dual to the source points [Brown 78]) ;
- finding the smallest enclosing sphere of a set of spheres of differing radii ;
- constrained versions of the problems studied in the thesis, where the optimal solution is constrained to lie in a convex region ;
- versions of the smallest enclosing sphere that use an L_1 or L_∞ metric instead of the L_2 metric, which has been used in this thesis.

6.2 Suggestions For Future Research

Since the results of this thesis hold in both 2 and 3 dimensions, it is likely that ICT can be extended to any arbitrary dimension. To prove this would involve showing that a hyperplane passing through the centre of gravity of a k-dimensional convex region divides it into two regions, such that the ratio of the volumes of the two regions would always lie between fixed limits, thus guaranteeing that the size of the solution region decreases geometrically with the number of iterations. It is conjectured that in the general case, these limits will be:

$$\frac{k^k}{(k+1)^k - k^k} \quad \text{and} \quad \frac{(k+1)^k - k^k}{k^k}$$

Algorithms for finding the volume of a k-dimensional convex region already exist. For example, [Cohen and Hickey 79] determine the volume of such a region by partitioning it into simplices. Converting such an algorithm so that it also finds the centre of gravity of the region should be relatively straightforward. The intersection algorithm by [Seidel 81], which is described in Section C.4, already handles convex regions of arbitrary dimension.

Degenerate convergence has been handled on a problem-by-problem basis. It would be advantageous to have one general approach for solving degeneracies.

Currently the routine that creates the initial solution region for LP requires that the set of constraints be bounded in all directions. This is not necessary however, since it is possible to add a constraint that does not affect the optimal solution. Once this generalization has been added, the ICT solution for LP will be applicable to many more situations.

So far it can only be claimed that ICT is expected to run very fast. It would be useful to implement ICT and empirically compare its running time with that of other algorithms. In particular, it would be useful to implement the set of routines described in Appendix C as a set of library routines.

In this thesis, the solution region has been reduced by a fixed fraction by exploiting a property of the centre of gravity of a convex region. It would be of interest to examine other ways of ensuring that the solution region is reduced by a fixed fraction in each iteration. For example, [Diaz and O'Rourke 89] have examined properties of the centre of area of a convex polygon.

Appendix A – Notation Conventions

All sets of points considered will be subsets of k-dimensional Euclidean space (E^k), where $k \leq 3$. An orthogonal coordinate system will be used, with the axes normally labeled x, y and z as shown in Figure A.1 .

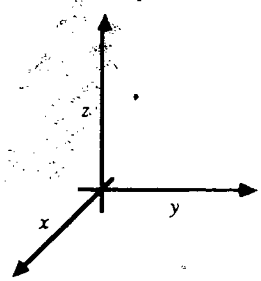


Figure A.1 Labelling of the coordinate system

Points and vectors will be denoted by lower case bold letters in the Helvetica font ($\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$). Scalars will be denoted by lower case italic letters in the Times font ($a, b, c, \ldots$). For example, $\mathbf{a} = (x_{\mathbf{a}} , y_{\mathbf{a}} , z_{\mathbf{a}})$ is a point in E^3. Provided that the meaning is clear, we will denote the coordinates of the point $\mathbf{p}_i$ as simply (x_i , y_i , z_i). However, if any ambiguity arises, we will revert to $(x_{\mathbf{p}_i} , y_{\mathbf{p}_i} , z_{\mathbf{p}_i})$ instead. We will use the terms *above* and *below* as follows: $\mathbf{a}$ lies above $\mathbf{b}$ implies that $z_{\mathbf{a}} > z_{\mathbf{b}}$ while $\mathbf{a}$ lies below $\mathbf{b}$ implies $z_{\mathbf{a}} < z_{\mathbf{b}}$.

A line segment will be denoted by its endpoints. For example, $\mathbf{a}\,\mathbf{b}$ is the line segment with endpoints $\mathbf{a}$ and $\mathbf{b}$. Lines and planes will be denoted by lower case italic Greek letters ($\alpha, \beta, \gamma, \ldots$). A line is considered to be *horizontal* if it is parallel to the $z = 0$ plane and *vertical* if it is perpendicular to this plane. A plane is *horizontal* if it is perpendicular to the z-axis and *vertical* if parallel to the z-axis.

Functions will be denoted by the Zapf Chancery font ($\mathcal{A}$, *a*, $\mathcal{B}$, *b*, ...). For example, *Distance*($\mathbf{a}, \mathbf{x}$) is a function that returns the Euclidean distance between the points $\mathbf{a}$ and $\mathbf{x}$. Appendix C summarizes the functions defined in this thesis.

Two parallel vertical bars will be used denote the absolute value of an expression. For example, $| -2 | = 2$.

Braces { ... } have been used for two purposes. Within algorithms, they have been used to delimit statement lists, similar to the C programming language convention. Everywhere else, they have been used to denote sets.

We will use the symbols $\cap$, $\cup$, $-$, $\subseteq$, $\in$, $\notin$ to denote the set operations of intersection, union, set difference, subset of, element of, not an element of, respectively. Finite sets of disjoint objects will be denoted by upper case italic letters in the Times font ($A, B, C, \ldots$). For example, $S = \{ \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n \}$ is a set of points. Regions, which are defined in Appendix B to be bounded continuous sets of points will be denoted by upper case italic Greek letters ($\Gamma, \Delta, \Phi, \ldots$). For example, $\Phi = \{ \mathbf{x} \in E^3 \mid \mathit{Distance}(\mathbf{a}, \mathbf{x}) \leq r \}$ is a sphere centred about the point $\mathbf{a}$ with radius r.

We will use the following notation to represent an iterative selection process:

$$\mathop{operation}_{i=1}^{n} \; expression \; .$$

For example, if $S = \{ \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n \}$, then the expression:

$$\mathop{\text{maximum}}_{i=1}^{n} \; \mathit{Distance}(\mathbf{c}, \mathbf{p}_i)$$

returns the point $\mathbf{p}_i \in S$ which is furthest from the point $\mathbf{c}$.

Appendix B – Some Definitions

This appendix includes definitions for some of the mathematical terms that have been used in this thesis. The reader is referred to [Preparata and Shamos 85] for definitions of more familiar geometrical terms like *half-space* or *on the same side of the line.*

B.1 Convex Set

A set of points Φ is defined to be convex if for every pair of points $\mathbf{a}, \mathbf{b} \in \Phi$, the line segment joining $\mathbf{a}$ and $\mathbf{b}$ is also contained in Φ.

B.2 Region

We define a *region* to be a bounded continuous set of points, where a set is said to be bounded if it can be enclosed by a sphere with a fixed radius. A *solution region* is simply a region that contains the exact solution of the problem that we are considering.

B.3 Affine Set

μ is said to be an affine set if for every pair of distinct points $\mathbf{a}, \mathbf{b} \in \mu$, the infinite line joining $\mathbf{a}$ and $\mathbf{b}$ is also contained in μ. Thus lines, planes and 3-space are affine sets and we define their *dimension* to be 1, 2 and 3 respectively. Trivially, a points is an affine space with dimension 0.

B.4 Dimension of a Region

We define the dimension of a region Φ to be the minimum dimension of the affine sets that contain Φ.

B.5 Hyperplane

We define a hyperplane to be an affine set of dimension $k - 1$, where k is the dimension of the space.

B.6 Interior, Exterior and Boundary Points

Points of space can be divided into three classes with respect to a region: interior, exterior and boundary points. We define a *neighbourhood* of the point $\mathbf{p}$ to be

$$\mathcal{N}(\mathbf{p}, r) = \{ \mathbf{x} \in E^k \mid \mathcal{D}istance(\mathbf{p}, \mathbf{x}) < r \} .$$

$\mathbf{p}$ is an *interior* point of the region Φ if $\mathcal{N}(\mathbf{p}, r) \subseteq \Phi$ for some sufficiently small r.

$\mathbf{p}$ is an *exterior* point of Φ if $\mathcal{N}(\mathbf{p}, r) \cap \Phi$ is empty for some sufficiently small r.

$\mathbf{p}$ is a *boundary* point of Φ if it is neither an interior nor exterior point.

B.7 Closed Region

A region is said to be closed if it contains all of its boundary points.

B.8 Supporting Hyperplane

A hyperplane μ supports a region Φ if the following two conditions are satisfied:

(1) μ contains at least one point of the boundary of Φ

(2) Φ lies in only one of the two closed half-spaces defined by μ.

B.9 Separating Hyperplane

The regions Φ_1 and Φ_2 are separated by the hyperplane μ if one of the open half-spaces bounded by μ contains Φ_1 and the other open half-space contains Φ_2.

B.10 Volume

The volume of a region Φ is defined to be $\mathcal{V}(\Phi) = \int_{\Phi} dV$. Normally the volume of a 1-dimensional region is referred to as its *length* and the volume of a 2-dimensional region as its *area*. We will use a subscript when we want to stress the dimension of the space we are working in. For example, $\mathcal{V}_2(\Phi)$ indicates that we desire the area of Φ. The following properties of volume are important to us.

1) If Φ has dimension k and $j > k$, then $\mathcal{V}_j(\Phi) = 0$.

2) If $\Phi_1 \subseteq \Phi_2$, then $\mathcal{V}(\Phi_1) \leq \mathcal{V}(\Phi_2)$.

B.11 Centre of Gravity

Usually only physical objects have a centre of gravity. For example, the centre of gravity of a planar figure cut out of sheet metal is the place where the point of a pin must be placed in order to balance the figure horizontally. In this thesis, we will use a geometric interpretation of the centre of gravity. That is, we shall assume that all regions considered are constructed from a homogeneous material with unit density[1]. We define the centre of gravity of a region Φ to be the point:

$$\mathbf{g} = \frac{\int_{\Phi} \mathbf{x} \, dV}{\int_{\Phi} dV}$$

Theorem B.1 Consider a region Φ which has been partitioned into r regions, $\Phi_1, \Phi_2, \ldots, \Phi_r$ with respective centres of gravity $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_r$. The centre of gravity of Φ is the point:

$$\mathbf{g} = \frac{\mathbf{g}_1 \mathcal{V}(\Phi_1) + \mathbf{g}_2 \mathcal{V}(\Phi_2) + \ldots + \mathbf{g}_r \mathcal{V}(\Phi_r)}{\mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2) + \ldots + \mathcal{V}(\Phi_r)}$$

Proof: Since Φ has been partitioned into r regions, the volume of Φ can be rewritten as: $\mathcal{V}(\Phi) = \mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2) + \ldots + \mathcal{V}(\Phi_r)$ and also

$$\int_{\Phi} \mathbf{x} \, dV = \int_{\Phi_1} \mathbf{x} \, dV + \int_{\Phi_2} \mathbf{x} \, dV + \ldots + \int_{\Phi_r} \mathbf{x} \, dV \, .$$

Therefore, if $\mathbf{g}$ denotes the centre of gravity of Φ then,

$$\mathbf{g} = \frac{\int_{\Phi} \mathbf{x} \, dV}{\mathcal{V}(\Phi)} = \frac{\int_{\Phi_1} \mathbf{x} \, dV + \int_{\Phi_2} \mathbf{x} \, dV + \ldots + \int_{\Phi_r} \mathbf{x} \, dV}{\mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2) + \ldots + \mathcal{V}(\Phi_r)} \qquad [1]$$

1 The technical term usually used for this notion is *centroid*. We use the more intuitive phrase for clarity.

Consider any one of the r regions of Φ, say Φ_i. By definition, the centre of gravity of Φ_i is the point, $\mathbf{g}_i = \dfrac{\int_{\Phi_i} \mathbf{x}\, dV}{\mathcal{V}(\Phi_i)}$ which means $\int_{\Phi_i} \mathbf{x}\, dV = \mathbf{g}_i\, \mathcal{V}(\Phi_i)$.

Substituting this into [1] leads us to conclude that

$$\mathbf{g} = \frac{\mathbf{g}_1\, \mathcal{V}(\Phi_1) + \mathbf{g}_2\, \mathcal{V}(\Phi_2) + \ldots + \mathbf{g}_r\, \mathcal{V}(\Phi_r)}{\mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2) + \ldots + \mathcal{V}(\Phi_r)} \quad \blacklozenge$$

Theorem B.2: Consider a region Φ, which has been partitioned into two regions, Φ_1 and Φ_2, with respective centres of gravity $\mathbf{g}_1$ and $\mathbf{g}_2$ (Figure B.1). The centre of gravity of Φ, $\mathbf{g}$, lies on the line segment joining $\mathbf{g}_1$ and $\mathbf{g}_2$, dividing it in two so that the ratio of the two parts is:

$$\frac{\text{Length}(\mathbf{g}_1\,\mathbf{g})}{\text{Length}(\mathbf{g}\,\mathbf{g}_2)} = \frac{\mathcal{V}(\Phi_2)}{\mathcal{V}(\Phi_1)}$$

Proof: Any point $\mathbf{p} = \mathbf{g}_1 + t(\mathbf{g}_2 - \mathbf{g}_1)$ lies on the line segment $\mathbf{g}_1\mathbf{g}_2$ if and only if t is a real number in the range $0 \le t \le 1$.

Let $t = \dfrac{\mathcal{V}(\Phi_2)}{\mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2)}$. Clearly, $0 \le t \le 1$, so the point:

$$\mathbf{p} = \mathbf{g}_1 + t(\mathbf{g}_2 - \mathbf{g}_1) = \frac{\mathbf{g}_1\, \mathcal{V}(\Phi_1) + \mathbf{g}_2\, \mathcal{V}(\Phi_2)}{\mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2)}$$

lies on the line segment $\mathbf{g}_1\mathbf{g}_2$, and $\dfrac{\text{Length}(\mathbf{g}_1\,\mathbf{p})}{\text{Length}(\mathbf{p}\,\mathbf{g}_2)} = \dfrac{t}{1-t} = \dfrac{\mathcal{V}(\Phi_2)}{\mathcal{V}(\Phi_1)}$

From Theorem B.1 we know that $\mathbf{g} = \dfrac{\mathbf{g}_1\, \mathcal{V}(\Phi_1) + \mathbf{g}_2\, \mathcal{V}(\Phi_2)}{\mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2)}$.

Therefore we conclude that $\mathbf{g}$ coincides with $\mathbf{p}$, dividing the line segment $\mathbf{g}_1\mathbf{g}_2$ in the ratio

$$\frac{\mathcal{V}(\Phi_2)}{\mathcal{V}(\Phi_1)} \quad \blacklozenge$$

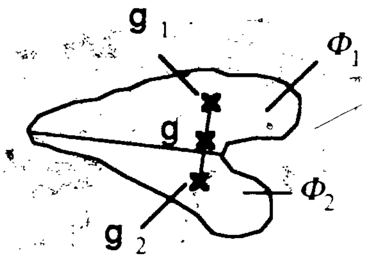


Figure B.1 Illustrating Theorem B.2

Appendix C – Some Implementation Details

In this appendix, issues that may arise when ICT is implemented are discussed, such as representing the solution region and finding its volume and centre of gravity. In addition, we will discuss some issues that arise when real numbers are approximated using a finite precision, floating point representation. It is a common practice in computational geometry to design algorithms for hypothetical computing environments that support real numbers. This clarifies the computational model plus provides a rich set of tools for solving problems. However, there is a drawback to this approach; many theoretically correct algorithms are not completely robust in practice. In this appendix we will try to identify some of the limitations and problems that might arise for ICT algorithms, once they have been implemented. Some of these problems are data dependent, which means that ICT may be applicable to some applications but not for others. Clearly it is useful to try to identify such problems early.

C.1 Numbers and Limitations

The advantage of representing real numbers as finite precision, floating point numbers is the speed with which computations can be performed: floating point hardware has been highly optimized. However, this approach does have some well-known drawbacks. In this section, we will consider a few of these limitations, and describe further limitations imposed by ICT. More information can be obtained from any introductory numerical analysis book. For example, see [Dennis and Schnabel 83].

The foremost limitation is that both the magnitude and the precision of the numbers that can be represented is limited. ICT further reduces this magnitude since it frequently computes the distance between two points. It is impossible to represent the distance between a point that has the biggest possible x-coordinate and one that has the smallest possible one, since distance is represented by a positive number. Also, it is often desirable to optimize the distance function by not evaluating the square root. This approach is applicable when relative distances are of interest, like finding a data point that is furthest from the centre of gravity of a region. In this case, the algorithm needs to be able to represent a number that is the square of the distance, which is double the normal order of magnitude. (Note that when applying this approach to weighted problems, the weight must also be squared.) In some cases, it may be possible to scale the input data so that the spread between points will be reduced to an acceptable range. However this approach reduces the precision of the solution and hence may not be acceptable.

Recall from Chapter 1 that the solution generated by an ICT algorithm should be within ε units of an exact solution, where ε is a parameter specified by the user. Clearly t is not possible to generate this solution if ε is smaller than the available precision. Numerical analysts often introduce the concept of machine epsilon so that the precision of a representation can be discussed without tying the discussion to a specific machine. Machine epsilon (*macheps*) is defined to be the smallest positive number a such that $(1 + a) > 1$. Thus each ICT algorithm should ensure that ε is greater than macheps before proceeding to search for a solution.

Since there is only a fixed number of bits available to represent each real number, a difference may exist between a real number and its floating point representation. Numerical analysts use macheps to describe this difference. If $float(x)$ denotes the floating point representation of a real number x, then

$$x(1 - \text{macheps}) < float(x) < x(1 + \text{macheps})$$

Similarly, the value of zero lies in the range,

$$-\text{macheps} < float(0) < \text{macheps}.$$

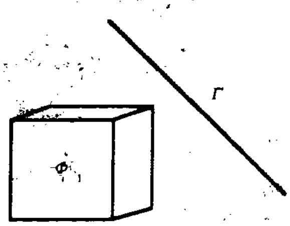
Some calculations are sensitive to small changes in numbers. For example, the intersection of two nearly parallel lines can be drastically affected by small differences in the numbers used to define the lines. Thus LP problems whose optimal solution are defined by constraints with nearly parallel boundaries will be affected by this. However, this particular problem applies to any algorithm that solves LP and not just ICT. [Bowyer and Woodwark 83] suggest representations for lines and planes and ways of performing intersections that help to minimize this type of sensitivity.

Some geometric algorithms suffer from accumulated errors, which arise when computations are based on the results of previous computations. The first computation may differ slightly from the true solution; this difference can easily be magnified by next computation. This should not be a problem for the

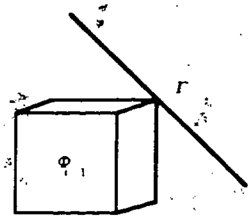
the ICT algorithms presented in this thesis, since the half-space used to reduce the volume of the solution region is determined by referring to the original input data each iteration.

C.2 A Data Structure For The Solution Region

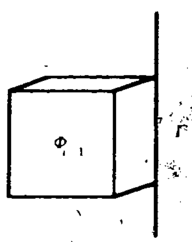
Each solution region will be constructed in a similar fashion; for 2-dimensional problems, it will be constructed by intersecting half-planes, and for 3-dimensional problems, it will be constructed by intersecting half-spaces. Notice that the bounding box that was used to construct the initial solution region in Algorithm 1.1 can be thought of as the intersection of four half-planes. Within this framework, the solution region can take on a variety of shapes: it may be empty, a point, a line segment, a convex polygon or else a convex polyhedron. Figure C.1 demonstrates situations where these different shapes may arise.



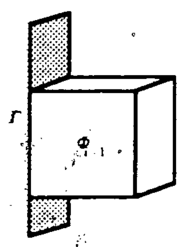
(a) Φ_i is empty



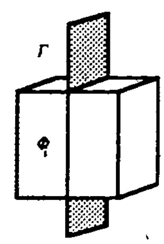
(b) Φ_i is a single point



(c) Φ_i is a line segment



(d) Φ_i is a convex polygon



(e) Φ_i is a convex polyhedron

Figure C.1 Determining $\Phi_i = \Phi_{i-1} \cap \Gamma$ for a 3-dimensional problem.

An incidence graph will be used to represent the solution region. This is a versatile data structure that can be used to represent a polytope of any dimension. The description that follows has been taken from [Edelsbrunner 87] (page 141). Since we need only be concerned with polytopes of 3 or less dimensions, some of the generality has been omitted.

Let Φ denote a convex polytope with non-empty interior in E^k, $k \le 3$. The terms 0-face, 1-face and 2-face will be used as synonyms for *vertex*, *edge* and *facet*. For convenience, the interior of Φ will be defined to be the only k-face of Φ, unless the interior is empty. Furthermore, the empty set will be defined to be the only (-1) - face of Φ. For $-1 \le j \le k-1$, a j - face f and a $(j+1)$ - face g are *incident* if f belongs to the boundary of g; in this case, f is called a *subface* of g and g is called a *superface* of f. Two faces of Φ are *adjacent* if they are incident upon a common edge, and two vertices are adjacent if they are incident upon a common edge. The *incidence graph* of Φ is an undirected graph whose nodes are in one-to-one correspondence with the j - faces of Φ, such that an arc exists between two nodes if for a j - face and a $(j+1)$ - face, their corresponding faces are incident. Figure C.2 illustrates an incidence graph for a tetrahedron.

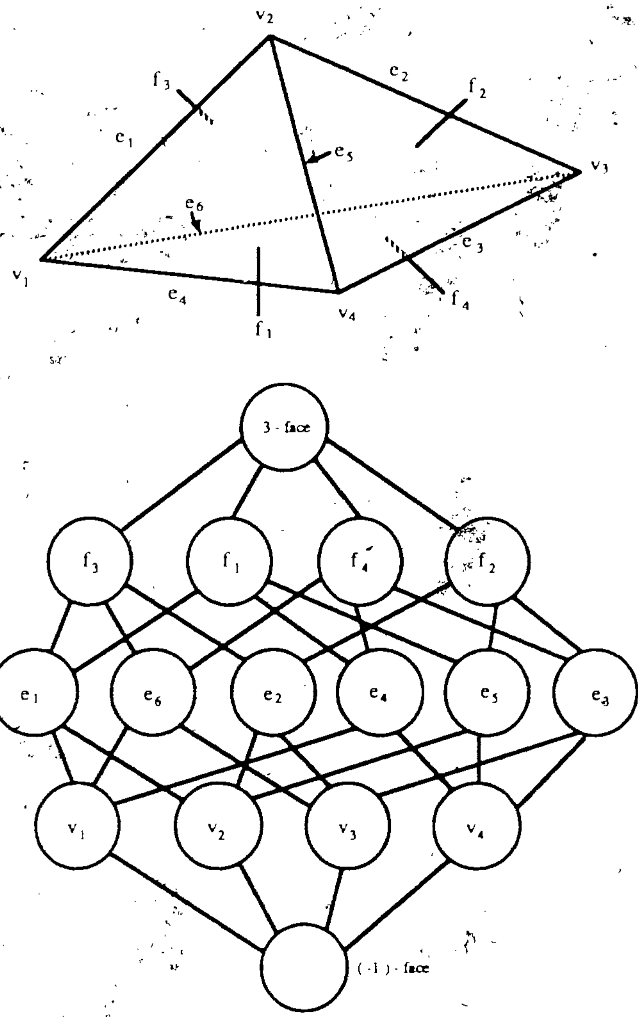


Figure C.2 The incidence graph for the tetrahedron shown above.

In order to store Φ using its incidence graph, some additional information is added to fix the location of Φ in space. For example, each 0-face records the coordinates of the vertex and each 2-face node describes the plane that it lies in. Note that the number of levels of the graph can be used to determine the dimension of the polytope. It is also worth noting that the list of edges incident upon a facet are in no particular order.

Let v, e and f denote the number of vertices, edges and faces of the solution region. [McMullen 71] has shown that for any polytope Φ of fixed dimension k, the amount of space required to store the incidence graph of Φ is $O(v^{\lfloor k/2 \rfloor})$, where v is the number of vertices of the polytope. Since in our case, $k \leq 3$, $O(v)$ is required to represent the solution region. Alternately, we can say $O(e)$ space is required if $k = 2$ since $v = e$, and if $k = 3$, we can say $O(f)$ space is required, since $v \leq 2f - 4$ [Grünbaum 67] (page 173).

C.3 Finding The Center Of Gravity Of The Solution Region

The centre of gravity of a point is the point itself; the centre of gravity of a line segment coincides with its midpoint. In this section we will consider the problem of finding the centre of gravity of a convex polygon and convex polyhedron. In both cases, the approach suggested by Theorem B.1 will be used, which states that if Φ is a region that has been partitioned into r regions, Φ_1, Φ_2, ..., Φ_r with respective centres of gravity $\mathbf{g}_1$, $\mathbf{g}_2$, ..., $\mathbf{g}_r$; then the centre of gravity of Φ is the point:

$$\mathbf{g} = \frac{\mathbf{g}_1\, \mathcal{V}(\Phi_1) + \mathbf{g}_2\, \mathcal{V}(\Phi_2) + \ldots + \mathbf{g}_r\, \mathcal{V}(\Phi_r)}{\mathcal{V}(\Phi_1) + \mathcal{V}(\Phi_2) + \ldots + \mathcal{V}(\Phi_r)} \quad \text{[C.1]}$$

In 2 dimensions, $\mathcal{V}(\Phi)$ denotes the area of Φ while in 3 dimensions, it represents the volume. Thus, the centre of gravity of Φ can be determined by first partitioning it into simplices (triangles or tetrahedrons) and then by finding the volume and centre of gravity of each simplex. These results can then be substituted into equation [C.1].

The area and centre of gravity of a triangle can be determined directly from its vertices. Let Φ_i denote a triangle with vertices $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$, where $\mathbf{v}_j = (x_j , y_j)$. The following formula determines the area of Φ_i:

$$\mathcal{V}(\Phi_i) = \left| \frac{1}{2} \mathit{Determinant} \begin{bmatrix} (x_2 - x_1) & (x_3 - x_1) \\ (y_2 - y_1) & (y_3 - y_1) \end{bmatrix} \right| .$$

Its centre of gravity can be determined by:

$$\mathbf{g}_i = \frac{\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3}{3} .$$

Similarly, let Φ_i denote a tetrahedron with vertices $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$, where $\mathbf{v}_j = (x_j, y_j, z_j)$. The volume of Φ_i can be determined by

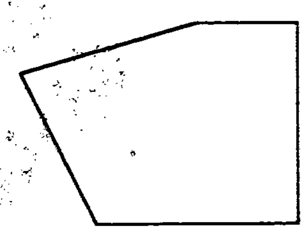
$$\mathcal{V}(\Phi_i) = \left| \frac{1}{6} \mathit{Determinant} \begin{bmatrix} (x_2 - x_1) & (x_3 - x_1) & (x_4 - x_1) \\ (y_2 - y_1) & (y_3 - y_1) & (y_4 - y_1) \\ (z_2 - z_1) & (z_3 - z_1) & (z_4 - z_1) \end{bmatrix} \right| \quad ;[C.2]$$

and its centre of gravity by

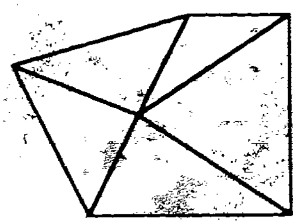
$$\mathbf{g}_i = \frac{\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3 + \mathbf{v}_4}{4} . \qquad [C.3]$$

[Bowyer and Woodwark 83] have presented optimized code for evaluating these equations. Each can be evaluated in constant time, given the vertices.

It is straight-forward to partition a convex polygon into triangles. For example, find a point $\mathbf{v}$ that is interior to Φ and connect the vertices of each edge to $\mathbf{v}$ as shown in Figure C.3. The same approach can be used to partition a convex polyhedron into tetrahedrons. That is, first connect each vertex to $\mathbf{v}$ (see Figure C.4.a). This partitions Φ into f pyramids, where f is the number of facets of Φ. (Each internal face borders exactly two pyramids, leaving no space unaccounted for.) Then partition each pyramid into tetrahedrons by triangulating its base which is a convex polygon, as shown in Figure C.4.b.

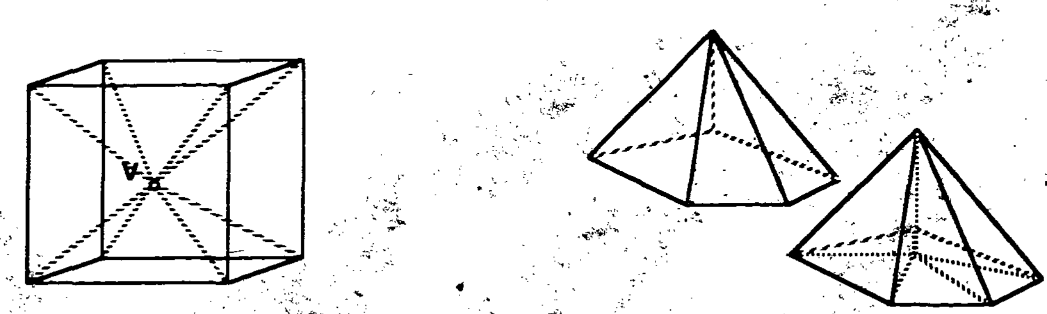


(a) before triangulation



(b) after triangulation

Figure C.3 Triangulating a convex polygon.



(a) partitioning Φ into pyramids

(b) partitioning a pyramid into tetrahedrons

Figure C.4. Partitioning a convex polyhedron.

First let's restrict our attention to finding the centre of gravity and area of a convex polygon. Recall that the edges of a polygon are connected to a 2-face node in the incidence graph. If this 2-face node has only three edges incident on it, then the polygon is a triangle Therefore assume that it has more than three edges incident on it. Consider any two of these edges, plus the four subfaces (vertices) of these two edges. At least three of the vertices will be non-identical. Select any such three and let $\mathbf{v}$ denote their centre of gravity. For each edge incident upon the 2-face node, calculate the centre of gravity and area of each triangle determined by the vertices of this edge and $\mathbf{v}$, and accumulate the values required by equation [C.1] above. Once each edge has been processed, determine the centre of gravity from the accumulated information. Since $\mathbf{v}$ can be determined in constant time, and the centre of gravity and area of each triangle can be found in constant time, the entire process requires $O(e)$ time, where e is the number of edges of the region.

Now consider a convex polyhedron. Recall that the faces of a polyhedron are connected to a 3-face node in the incidence graph. If this 3-face node has only four faces incident on it, then the polyhedron is a tetrahedron. Therefore suppose that it has more than four faces incident on it and consider the problem of finding a point interior to the polyhedron. Select an arbitrary vertex of the polyhedron. There will be at least three edges incident upon this vertex, and at most, only two of these edges will be incident upon the same face. Find the other endpoint of each of the three edges and let $\mathbf{v}$ denote the centre of gravity of these four points. Since the tetrahedron formed by these four points is contained by Φ, $\mathbf{v}$ will lie in its interior. Now access each of the facets of the polyhedron in turn, through the 3-face node. Since each facet is a convex polygon, it can be partitioned into triangles as was described above. Each edge will form another tetrahedron. Clearly the information required by equation [C.1] above can be accumulated as each edge of the face is processed. Since each edge of the polyhedron is incident on exactly two faces, each edge will be used exactly twice during the triangulation process. Thus the number of tetrahedrons examined will be linear in the number of edges of the polyhedron. Since equations [C.2] and [C.3] can both be evaluated in constant time, the total time required to determine the centre of gravity of a convex polyhedron is linear its number of edges of polyhedron. Since $e \leq 3f - 6$, where e and f are the number of edges and faces of the polyhedron, [Grünbaum 67] (page 173), the centre of gravity of a convex polyhedron can be found in time $O(f)$ time.

C.4 Reducing The Solution Region

Consider the problem of finding $\Phi_i = \Phi_{i-1} \cap \Gamma$, where Φ_{i-1} is a convex polytope and Γ is a half-space. [Seidel 81] has solved this problem for arbitrary dimension, in time proportional to the amount of change from the incidence (or facial) graph of Φ_{i-1} to the incidence graph of Φ_i.

Now consider the amount of change that can result from the intersection, that is the maximum number of deletions and additions to the graph. Let v, e and f denote the number of vertices, edges and faces of Φ_{i-1}. At most, one $(k-1)$-face will be added to Φ_{i-1}. Consider the size of the incidence graph of this addition. As was mentioned in Section C.2, the amount of space required to store its incidence graph of a k-dimensional polytope is $O(v^{\lfloor k/2 \rfloor})$ [McMullen 71]. From this formula, it is easy to see that the additional $(k-1)$-face will require $O(1)$ space if $k \leq 2$. If $k = 3$, then the new facet will be a convex polygon having at most f edges and f vertices. Therefore, $O(f)$ space will be required to store the graph and hence, $O(f)$ time will be required to create it. At the other extreme, Φ_i will be empty, which means that all but one node of the incidence graph for Φ_{i-1} will be deleted. $O(v^{\lfloor k/2 \rfloor})$ time will be required to delete this graph [Seidel 81]. As was mentioned at the end of Section C.2, this can alternately be stated as $O(e)$ time when $k = 2$ and $O(f)$ time when $k = 3$. Thus in summary, the intersection $\Phi_i = \Phi_{i-1} \cap \Gamma$ can be performed in $O(1)$ time if $k \leq 1$, $O(e)$ time if $k = 2$ and $O(f)$ time if $k = 3$.

Note that it is possible that the result of the intersection will be so thin that it should be considered to be of a lower dimension. This can be detected by finding the perpendicular distance between each vertex of Φ_i and the boundary of Γ. If the maximum (perpendicular) distance is less than some function of *macheps*, then this routine will ensure that the solution region is reduced to a lower dimension by intersecting it with a plane. This result can be achieved by intersecting Φ_i with two half-spaces whose boundaries are defined by the same plane but which extend to opposite sides of this plane. Since Φ_i is already the result of intersecting Φ_{i-1} with Γ, the natural choice of a second half-space is Γ_1, which has the same boundary as Γ but extends to the opposite direction. Since before this last intersection Φ_i has at most one more face than Φ_{i-1}, this last intersection can be performed in time $O(f)$ time, where f is the number of faces of Φ_{i-1}.

Appendix D – A Summary Of The Functions

$\mathcal{Circle}(\mathbf{c}, r)$

defines the set of points enclosed be a circle with centre $\mathbf{c}$ and radius r.

$$\mathcal{Circle}(\mathbf{c}, r) = \{\, \mathbf{x} \in E^2 \mid \mathcal{Distance}(\mathbf{c}, \mathbf{x}) \le r \,\}$$

$\mathcal{CH}(S)$

the convex hull of S, a set of points.

$\mathcal{COG}(\Phi)$

returns the point that is the center-of gravity of the region Φ.

$$\mathcal{COG}(\Phi) = \frac{\int_{\Phi} \mathbf{x}\, dV}{\int_{\Phi} dV}$$

$\mathcal{Distance}(\mathbf{a}, \mathbf{b})$

returns the Euclidean distance between two points, $\mathbf{a}$ and $\mathbf{b}$.

$$\mathcal{Distance}(\mathbf{a}, \mathbf{b}) = \sqrt{\left(\sum_{i=1}^{k} (a_i - b_i)^2 \right)}$$

$\mathcal{Furthest}(\mathbf{g}, S)$

returns the point of S that is furthest from the point $\mathbf{g}$.

$$\mathcal{Furthest}(\mathbf{g}, S) = \{\, \mathbf{p} \in S \mid \mathcal{Distance}(\mathbf{g}, \mathbf{p}_j) \le \mathcal{Distance}(\mathbf{g}, \mathbf{p}),\ j = 1, \ldots, n \,\}$$

$\mathcal{Sphere}(\mathbf{c}, r)$

defines the set of points enclosed be a sphere with centre $\mathbf{c}$ and radius r.

$$\mathcal{Sphere}(\mathbf{c}, r) = \{\, \mathbf{x} \in E^3 \mid \mathcal{Distance}(\mathbf{c}, \mathbf{x}) \le r \,\}$$

References

[Bhattacharya and Toussaint 85]
B. Bhattacharya, G. Toussaint, "On Geometric Algorithms That Use The Furthest-Point Voronoi Diagram", *Computational Geometry*, G.T.Toussaint (Editor), Elsevier Science Publishers B.V. (North-Holloand), pp. 43 - 61, 1985.

[Blaschke 49]
W. Blaschke, *Kreis und Kugel*, Chelsea Publishing Company, N.Y, 1949.

[Blum, Floyd, Pratt, Rivest, and Tarjan 72]
M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, "Time Bounds for Selection", *J. Comput. Systems Sci.*, vol. 7, pp. 448-461, 1972.

[Bowyer and Woodwark 83]
A. Bowyer, J. Woodwark, *A programmer's geometry*, Butterworths, Toronto, 1983.

[Brown 78]
K. Q. Brown, "Fast intersection of half spaces", Carnegie-Mellon University, Pittsburgh, Pennsylvania, Department of Computer Science, Report CMU-CS-78-129, 1978.

[Castells and Melville 83]
C. Castells, R. Melville, "An Unusual Algorithm for the Minimum Spanning Circle Problem", Technical Report 17, Department Electrical Engineering and Computer Science, Johns Hopkins University, Balitmore, 1983.

[Chakraborty and Chaudhuri 81]
R. K. Chakraborty, P. K. Chaudhuri, "Note on Geometrical Solution for Some Minimax Location Problems", *Transportation Science*, vol. 15, pp. 164-166, 1981.

[Chazelle and Dobkin 87]
B. Chazelle and D. P. Dobkin, "Intersection of Convex Objects in Two and Three Dimensions", *JACM*, vol. 34, pp. 1-27, 1987.

[Chrystal 1885]
G. Chrystal, "On the Problem to Construct the Minimum Circle Enclosing n Given Points in the Plane", *Proc. Edinburgh Math. Soc.*, vol. 3, pp. 30-33, 1885.

[Chvátal 83]
V. Chvátal, *Linear Programming*, W. H. Freeman and Company, 1983.

[Clarkson 86]
K. L. Clarkson, "Linear programming in $O\left(3^{(k+1)^2}\, n\right)$ time", *Information Processing Letters*, vol. 22, pp. 21-24, 1986.

[Cohen and Hickey 79]
J. Cohen, T. Hickey, "Two Algorithms for Determining Volumes of Convex Polyhedra", *Journal of the ACM*, vol. 26, no. 3, July 1979, pp. 401 - 414 .

[Dennis and Schnabel 83]
J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Inc., New Jersey, 1983.

[Dobkin and Reiss 80]
D. P. Dobkin, S. P. Reiss, "The Complexity of Linear Programming", *Theoretical Computer Science,* vol. 11, pp. 1-18, 1980.

[Dantzig 82]
G. B. Dantzig, "Reminiscences about the origins of Linear Programming", *Operations Research Letters,* vol. 1, no. 2, pp. 43-48, April, 1982.

[Diaz and O'Rourke 89]
M. Diaz, J. O'Rourke, "Computing the center of area of a polygon", *Lectures in Computer Science*, Volume 382, Proceedings of the Algorithms and Data Structure Workshop, Ottawa, Aug. 1989.

[Duda and Hart 73]
R. Duda and P. Hart, *Pattern Classification And Scene Analysis,* A Wiley-Interscience Publication, John Wiley & Sons, New York, 1973.

[Dyer 84]
M. E. Dyer, "Linear time algorithms for two- and three-variable linear programs," *SIAM Journal on Computing,* vol. 13, no. 1, pp. 31-45, Feb. 1984.

[Dyer 86]
M. E. Dyer, "On a multidimensional search technique and its application to the Euclidean one-center problem", *SIAM Journal on Computing*, vol. 15, pp. 725-738, 1986.

[Edelsbrunner 87]
H. Edelsbrunner, *Algorithms in Computational Geometry*, Volume 10 of EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1987.

[Edelsbrunner, Guibas and Stolfi 86]
H. Edelsbrunner, L. Guibas, J. Stolfi, "Optimal point location in a monotone subdivision, *SIAM J. Computing,* vol. 15, no. 2, May 1986, pp. 317 - 340 .

[Elzinga and Hearn 72a]
J. Elzinga, D.W. Hearn, "Geometrical Solutions for Some Minimax Location Problems", *Transportation Science,* vol. 6, pp. 379—394, 1972.

[Elzinga and Hearn 72b]
J. Elzinga, D.W. Hearn, "The Minimum Covering Sphere Problem", *Management Science,* vol. 19, no. 1, pp. 96-104, Sept. 1972.

[Floyd and Rivest 75]
R. W. Floyd and R. L. Rivest, "Expected Time Bounds for Selection", *Comm. ACM*, vol. 18, pp. 165-172, 1975.

[Francis and White 74]
R. L. Francis, J. A. White, *Facility Layout and Location, An Analytical Approach,* Prentice-Hall, Inc., New Jersey, 1974.

[Grünbaum 67]
B. Grümbaum, *Convex polytopes* , Wiley Interscience, New York, 1967.

[Guibas, Stolfi and Clarkson 87]
L. Guibas, J. Stolfi, K. Clarkson, "Solving related two- and three-dimensional linear programming problems in logarithmic time", *Theoretical Computer Science*, vol. 49, 1987, pp. 81 - 84.

[Guggenheimer 77]
H. W. Guggenheimer, *Applicable Geometry, Global and Local Convexity*, Robert E. Krieger Publishing Company, Inc., Hunnington, New York, 1977.

[Gustin 47]
W. Gustin, "On the Interior of the Convex Hull of a Euclidean Set", *The Bulletin of the American Mathematical Society*, vol., 53, 1947, pp. 299-301 .

[Hearn and Vijay 82]
D. Hearn, J. Vijay, "Efficient Algorithms for the (Weighted) Minimum Circle Problem", *Operations Research*, vol. 30, No. 4, pp. 777-795, July-August, 1982.

[Jacobsen 81]
A. Jozwik, "A Recursive Method for the Investigation of the Linear Separability of Two Sets", *Pattern Recognition*, vol. 16, no. 4, pp. 429-431, 1983.

[Jozwik 83]
S. K. Jacobsen, "An Algorithm for the Minimax Weber Problem", *Euorpean Journal of Operational Research*, vol. 6, pp. 144-148.

[Karmarkar 84]
N. Karmarkar, "A New Polynomial-Time Algorithm for Linear programming", *Combinatorica*, vol. 4, pp. 373-395, 1984.

[Khachiyan 79]
L. G. Khachiyan, "A polynomial algorithm in linear programming", (in Russian), *Doklady Academiia Nauk SSSR 244* : 1093—1096. [English translation: *Soviet Mathematics Doklady* vol. 20, pp. 191—194.]

[Kirkpatrick 83]
D. Kirkpatrick, "Optimal search in planar subdivisions, *SIAM J. Computing*, vol. 12, 1983, pp. 28 - 35 .

[Kirkpatrick and Seidel 86]
D. Kirkpatrick, R. Seidel, "The ultimate planar convex hull algorithm?", SIAM Journal of Computing, vol. 15, 1986, pp. 287-299.

[Klee and Minty 72]
V. Klee, G. J. Minty, "How good is the simplex algorithm?", in *Inequalities-III*, editor: O. Shisha, Academic Press, New York, 1972, pp. 159-175,

[Lee and Preparata 84]
D. T. Lee, Franco P. Preparata, "Computational Geometry—A Survey," *IEEE Transactions on Computers*, vol. c—33, no. 12, pp. 1072-1101, Dec. 1984.

[McMullen 71]
P. McMullen, "The maximum number of faces of a convex polytope", *Mathematika*, Vol. 17, 1971, pp. 179-184.

[Megiddo 83a]
N. Megiddo, "Linear time algorithm for linear programming in R^3 and related problems," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 759-776, Nov. 1983.

[Megiddo 83b]
N. Megiddo, "The Weighted Euclidean 1-center problem," *Mathematics of Operations Research*, vol. 8, pp. 498-504, 1983.

[Megiddo 84]
N. Megiddo, "Linear programming in linear time when the dimension is fixed," *J. ACM*, vol 31, no. 1, pp. 114-127, Jan. 1984.

[Melville 85]
R. C. Melville, "An Implementation Study of Two Algorithms for the Minimum Spanning Circle Problem", *Computational Geometry*, Vol 2 of Machine Intelligence and Pattern Recognition, editor G. T. Toussaint, North-Holland, 1985, pp. 267-294.

[Nair and Chandrasekaran 71]
K. Nair, R. Chandrasekaran, "Optimal Location of a Single Service Center of Certain Types", *Naval Res. Logist. Quart.*, vol. 18, 1971, pp. 503-510.

[O'Rourke 85]
J. O'Rourke, "Finding Minimal Enclosing Boxes", *International Journal of Computer and Information Sciences*, vol. 14, no. 3, 1985, pp. 183 - 199 .

[Oommen 87]
B. John Oommen, "An efficient geometric solution to the minimum spanning circle problem," *Operations Research*, vol. 35, no.1, pp. 80-86, January-February 1987.

[Papadimitriou and Steiglitz 82]
C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, N. J. , 1982.

[Preparata and Muller 79]
F. P. Preparata, D. E. Muller, "Finding the intersection of n half-spaces in time O(n log$n$)", *Theoretical Computer Science*, vol. 8, no. 1, pp. 45-55, Jan. 1979.

[Preparata and Shamos 85]
F. P. Preparata, M. I. Shamos, *Computational Geometry An Introduction*, Springer-Verlag New York Inc., New York, 1985.

[Rademacher and Toeplitz 57]
H. Rademacher, O. Toeplitz, *The Enjoyment of Mathematics* (translated from *Von Zahlen und Figuren*, Ed. 2, 1933, Julius Springer, Berlin), Princeton University Press, Princeton, N.J., 1957.

[Reichling 88]
M. Reichling, "On the Detection of a Common Intersection of k Convex Objects in the Plane", *Information Processing Letters*, Vol. 29, pp. 25–29, 1988.

[Schonhage, Paterson and Pippenger 76]
A. M. Schonhage, M. S. Paterson and N. Pippenger, "Finding the Median", *J. Comput. System Sci.* , vol. 13, pp. 1884-199, 1976.

[Seidel 81]
R. Seidel, *A convex hull algorithm optimal for point sets in even dimensions*, M.Sc. Thesis, University of British Columbia, 1981.

[Shamos 78]
M. I. Shamos, Computational Geometry, Ph.D. Thesis, Yale University, 1978.

[Shamos and Hoey 75]
M. I. Shamos, D. Hoey, "Closest-Point Problems", Proceedings of the 16th Annual EEE Symposium on Foundations of Computer Science, 1975, pp. 151-162.

[Sylvester 1857]
J. J. Sylvester, "A Question in the Geometry of Situation", *Quart. J. Pure Appl. Math,* vol. 1, pp. 79, 1857.

[Sylvester 1860]
J. J. Sylvester, "On Poncelet's Approximate Linear Valuation of Surd Forms", *Philosophical Magazine* vol. 20 (fourth Series), pp. 203-222.

[Yaglom and Boltyanskii 61]
I. M. Yaglom and V. G. Boltyanskii, *Convex Figures*, Translated by: P. J. Kelly and L. F. Walton, Translation copyright 1961, New York: Holt, Rinehart and Winston.

[Zorbas 86]
J. Zorbas, *Applications of Set Addition to Computational Geometry and Robot Motion Planning*, M.Sc. Thesis, School of Computing Science, Simon Fraser University, Burnaby, B.C., May 86.