Anchored Sweep: A Paradigm

Dissertation

zur Erlangung des akademischen Grades
"Doktor der technischen Wissenschaften"
in der Studienrichtung Informatik

verfaßt von
Petru Pau
am
Forschungsinstitut für symbolisches Rechnen
Technisch-Naturwissenschaftliche Fakultät
Johannes Kepler Universität Linz

März 2000

Erstar Begutachter: Univ. Doz. Dr. Sabine Schifter
Technisch-Naturwissenschaftliche Fakultät
Johannes Kepler Universität Linz

Zweiter Begutachter: Univ. Prof. Dr. Ştefan Măruşter
Mathematics Faculty
West University of Timişoara, România

Contents

Eidesstattliche Erklärung

Ich versichere, daß ich die Dissertation selbständig verfaßt habe, andere als die angegebenen Quellen und Hilfsmittel nicht verwendet und mich auch sonst keiner unterlaubten bedient habe.

Petru Pau
Linz, im März 2000

Zusamenfassung

Das Thema dieser Arbeit liegt im Bereich "Computational Geometry". Wir geben hier eine umfassende Darstellung der Anwendungsmöglichkeiten der "anchored sweep"-Technik. Wir beschränken uns dabei auf Anwendungen auf sogenannte *Optimalplazierungsproblemen*.

In Anlehnung an das wohlbekannte "line sweep"-Model wird "anchored sweep" auf Räume angewandt, die topolgisch zu einem Kreis oder einem abgeschlossenen Geradensegment homeomorph sind. Die Intuition dahinter besteht darin, ein Objekt tangential am Rand eines anderen Objekts entlangzubewegen, und die Ereignisse, die dabei auftreten, zu verarbeiten.

Wir verwenden hier die "anchored sweep"-Technik um spezielle Positionen konvexer Objekte im zwei- und dreidimensionalen Raum aufzuspüren, nämlich solche Positionen, in denen diese Objekte eine maximale Anzahl von Punkten einer vorgegebenen Menge enthalten. Diese Positionen werden durch Ausfuehrung einer Verschiebung oder einer Isometrie erreicht.

Auf diese Art erreichen wir Lösungen fuer folgende Probleme: Sei P ein ebenes konvexes Polygon oder ein dreidimensionales Polyhedron, und sei S eine Punktmenge in der Ebene bzw. im Raum. Gesucht ist eine Verschiebung oder eine Isometrie τ sodass $\tau(P)$ eine maximale Anzahl von Punkten in S enthält.

Eine erste Diskretisierung des Suchraumes ergibt sich durch sogenannte "stabile Plazierungen", das sind spezielle Transformationen, nach deren Ausführung der Rand von P eine Anzahl von Punkten aus S enthält. Wir zeigen, dass "stabile Plazierungen" durch das Schneiden einer Anzahl von Kopien von P erreicht werden können. Beim Schneiden einer Anzahl von Kopien von P entsteht ein Polygon, das danach als Suchraum verwendet wird.

Während der Ausfuehrung des "anchored sweep" wird P mit einer Ecke an diesem Durchschnittpolygons gehalten. Während P entlang des Durchmessers bewegt wird, passieren die Punkte aus S durch den Rand von P. Jene Position von P, die die meisten Punkte aus S enthält, wird gespeichert.

Wir zeigen ausserdem, dass die Mindestkomplexität für Probleme dieses Typs $\Omega(nlog(n))$ ist, wobei n die Anzahl der Ecken von P ist.

Abstract

The subject of this thesis is situated in the frame of computational geometry. We give here a unified exposition of the utilities of the *anchored sweep*. We focus on applications in problems of optimal placement.

Derived from the well established *line sweep* paradigm, the anchored sweep is used for spaces that are topologically homeomorphic to a circle or a segment – usually not a straight line. It has a very strong intuition behind it, as it consists basically in moving an object with a point anchored on the boundary of another object, and processing all events that occur during this movement.

In this thesis, the anchored sweep is used to detect special position of convex objects in two- and three-dimensional spaces, namely, positions in which these objects contain a maximum number of points from a given set. These positions can be attained by applying either a translation or a rigid motion.

Thus, we present solutions for the following problems: Let P be a convex planar polygon or a polyhedron in three-dimensional space, and let S be a set of points in plane respectively in space. Find a translation or a rigid motion τ such that $\tau(P)$ contains a maximum number of points in S.

A first discretization of the search space is given by the *stable placements*, i.e., special transformation under which the boundary of the convex object contains a number of points from the initial set. We relate the stable placements to the intersection of a number of certain translated copies of P. By intersecting a number of copies of the initial object a polygon is obtained; it will be used afterwards as the space that will be swept.

During the anchored sweep, P is kept with one vertex on the polygon given by the intersection. As P traverses the perimeter, the points in S traverse the boundary of P. The position of P with maximum S-points inside is stored.

It is shown here that the lower bound for problems of this type is $\Omega(n \log n)$, where n is the number of vertices of P.

Acknowledgements

I am extremely grateful to my supervisor, Univ. Doc. Dr. Sabine Stifter, for her continuous encouragements during my work on this thesis. I would also like to thank my Romanian supervisor Prof. Dr. Stefan Maruster, for his support and help.

Prof. Tudor Jebelean I would like to thank for suggesting me to study and do research at RISC, and for guiding me. Perhaps my most important gain here comes from the lecture "Thinking, Speaking, Writing" of Prof. Bruno Buchberger, whom I would like to express my appreciation and gratitude.

I am deeply grateful to Prof. Matt Dickerson from Middlebury College, USA: he started it all. Many thanks should be addressed to Wadham College, Oxford, UK, for giving me the opportunity to work there on some chapters of my thesis.

I thank my friends, Dipl. Ing. Artur and Luminita Pustai from Erlangen, Germany, who made it possible to me to write most of this thesis.

Chapter 1

Introduction

Although relatively new as a science, *computational geometry*[1] has already brought results that have pushed it in the first row of contemporary sciences. And not only its immediate, practical applications are of importance: The theory developed in its frame has solved a sum of other, older problems.

As many other sciences did, computational geometry has come up with *methods*. Some of these methods are of such a great importance that they are considered *paradigms*. Closely related to other branches of the computational and data processing sciences, computational geometry has continuously updated their tools and concepts.

From the mathematical point of view, computational geometry uses mainly theoretical knowledge from Euclidean geometry, projective geometry, topology and analysis. Our belief is that methods developed in computer algebra might prove extremely useful[2].

From the side of data processing, the systematic study of algorithms and data structures has provided tools for computational geometry to develop. The reverse is also true: many algorithms, data structures and optimality results have been discovered when studying geometrical problems.

[1] For a comprehensive historical perspective we refer the excellent book of Preparata and Shamos [PreparataShamos85], pp. 1-6.

[2] Since many geometrical relations can be expressed in terms of polynomial expressions, methods developed in computational algebra – for instance Groebner bases [Buchberger85] – can be applied

1.1 Paradigmatic Approach of Geometrical Problems

Practice has proven that a relatively small number of methods ensures a coverage of almost all the problems that have to be solved. In fact, these methods were applied thousands of years ago, in the most variate situations.

One step at a time (or incremental construction of the solution) is maybe the safest way to reach the result, although usually not the fastest. Its modern offsprings range from queuing the things to be done (in a *first-in-first-out* manner) to a treatment based on *assigning priorities*.

Divide and conquer has brought considerable speed in solving the problems. Indeed, by splitting the matters and solving each piece separately, and afterwards by putting together the results, one usually works easier and faster.

In computational geometry, some problems can be solved by incremental construction of the solution: convex hull, triangulation, intersections. Many of these problems can be solved in a divide-and-conquer manner, the solutions being generated much faster.

An extremely useful paradigm, suggested by the nature of geometrical problem, is *space sweep*: A certain k-dimensional Euclidean space is swept by a "vertical" hyperplane. During this motion, some points are encountered by the hyperplane: they are inserted in a data structure in the order in which they appear. Later, the data structure is processed in turn, of importance here being the order in which the points are processed.

Perhaps the most heavily used instance is the *plane sweep*: A vertical line sweeps the plane from left to right; its intersections with objects are somehow stored so that they can be processed afterwards. The main applications reside in the computation of intersections of planar shapes and geometry of rectangles. A slightly modified version is used to compute the convex hull of a planar set of points: here one uses a ray emanating from an internal point of the convex hull instead of a vertical line.

The *anchored sweep* is a method derived from the general stem of sweeping techniques. A geometric object is moved while preserving one of its points on a curve in the contour of another object. Again, during this sweep some events occur, and their informational counterpart is stored in a special data structure which is later processed.

1.2 A Class of Problems Solved with Anchored Sweep

We exemplify the usage of the anchored sweep by treating in detail problems of optimal placement. The most general case of such a problem is:

Problem 1.2.1. *Given a geometric object O in a universe populated with another geometric objects $o_1, o_2 \dots$ find a transformation τ such that $\tau(O)$ optimizes a certain measure μ that depends on τ, O and $o_1, o_2 \dots$.*

The object to be placed can be given either explicitly (e.g., by numerical values for its geometrical attributes, such as: radius, in the case of a circle; coordinates of the vertices, in the case of a polytope) or by the means of some properties (e.g., convexity). The transformation τ is usually a geometrical transformation (translation, rotation, homothety, scale reduction). The domain of the objects that populate the space ranges from sets of points (which can be finite or infinite) to other more complex objects (polyhedral domains, sets of spheres, etc.).

We will to give solution for instances of the problem:

Problem 1.2.2. *Given a convex polytope P and a set of points S, find a geometric transformation τ of the space such that $\tau(P)$ contains a maximal subset of S in the interior or on the boundary.*

Although this problem was first posed in very practical applications, solutions of some of its instances have been lately used as a preliminary step in some theoretical papers ([BarequetADG97]).

We will use **the anchored sweep** for solving this kind of problems: Intuitively, O will be moved continuously while keeping one particular point of its boundary (usually, a vertex) on a closed curve. This movement defines a space of positions of O, space which, in fact, is one-dimensional. We are looking for particular points in this space, like, for instance, points which ensures special position of O with respect to other objects.

We will present here solutions that are asymptotically faster than the previous approaches (or the naive ones, when they were not attacked at all).

1.3 Structure of the Thesis

The structure of this thesis is as follows:

Chapter 2 contains an exposition of the main results related to line sweep, anchored sweep and particular instances of problem 1.2.1.

Chapter 3 contains the theoretical background that supports the algorithms. We will prove here the existence of some very special transformations, that put a d-dimensional polyhedron in contact with a certain subset of the set of points. We will also relate here such transformation to the intersection of some translated copies of the polyhedron. It will also be proved here that there exists a rectangular box which includes the polyhedron and can be tiled up by a constant – up to the dimension of the space – number of translated copies of the polyhedron.

The chapter ends with some considerations about the complexity of the problem.

In Chapter 4 we will introduce the main data structures and sketch some methods that will be used as sub-algorithms. Their main purpose is to speed up the algorithm, by optimally performing the subtasks.

Thus, we will present here data structures designed for geometric algorithms: doubly connected edge lists, segment trees, chains of trees. We will also present in detail fast algorithms for finding intersections of convex shapes, in two and three dimensions.

Chapter 5 contains the main algorithms, for the planar and three-dimensional cases. For the planar case, we will present solutions for the problems of finding optimal translation and optimal rigid motion – i.e., composition of a translation and a rotation. For the three-dimensional case, we give a solution for finding the optimal translation.

Chapter 2

Historical Perspective

2.1 Space-Sweep Methods

The concept of space sweep has been introduced by Shamos and Hoey in the paper *Geometric Intersection Problems* [ShamosHoey76]. The method has been used later by many other researchers.

The book *Computational Geometry an Introduction* [PreparataShamos85] of Preparata and Shamos describe a wealth of solutions, for very different problems, which use space-sweep methods.

Reporting intersections

Perhaps the most famous application of the space-sweep is in the problem of detecting and reporting all the intersections in a set of plane segments. The paper of Shamos and Hoey has only given the initial impulse (and the main method) in its systematic study. Bentley and Ottmann, in [BentleyOttmann78], have offered answers to many problems posed by Shamos and Hoey in their paper. Red-blue segment intersection problem has been solved by Palazzi and Snoeyink ([PalazziSnoeyink94]) using a variant of the sweep algorithm. A provably optimal algorithm for reporting line segments intersections has been proposed by Chazelle and Edelsbrunner [ChazelleEdelsbrunner92]. The case when any degeneracy is enabled is presented by Freiseisen and Pau in [FreiseisenPau98].

The same paradigm is used when studying more complicated structures, like planar maps – their intersection is considered in [NievergeltPreparata82] and [MairsonStolfi83].

Basically, the method works as follows: A vertical line sweeps the plane from left to right. In each moment, the intersection of this line with the set of segments – its *status* – is maintained. This intersection changes whenever an endpoint of one of the segments is reached, or when the line touches an intersection point. These *events* are processed in turn, by changing the status of the sweeping line. A report is produced each time an intersection event is reached. The list of events, which initially contains the endpoints, is continuously updated by eliminating the events that have already been processed (points at the left of the sweeping line), or by inserting as new events the intersections. Figure 2.1 gives a snapshot from the evolution of such a method.

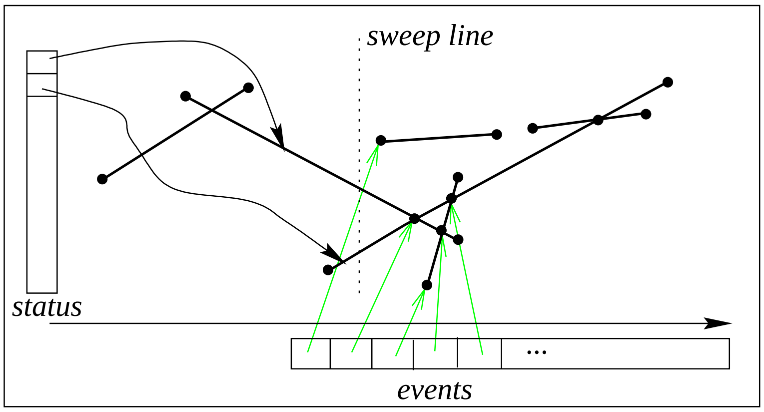

Figure 2.1: A snapshot in the evolution of the plane-sweep algorithm.

Point location

Planar point-location problems, whose systematic study began with the paper of Dobkin and Lipton [DobkinLipton76], are attacked with a wide range of methods. One has a subdivision of the plane, given as a planar graph the edges of which are represented as straight line segments. The problem is to find in which face of this planar graph a given query-point lies. Some very interesting solutions for this kind of problems use the space-sweep paradigm.

In *the slab method* ([PreparataShamos85], pp. 45), the plane is divided into horizontal slabs defined by the y-coordinate of the vertices of the planar graph (see Fig. 2.2, a). By sweeping the plane from bottom to top, one gets the instantaneous status consisting in the sequence of edges that intersect the current slab. When the slab containing the query-point is reached, the information stored in status of this slab is enough to reconstitute the planar face that contains the point.

The *chain method* ([PreparataShamos85], pp. 48), that solves the same problem of point-location, uses the plane-sweep technique in a preprocessing phase, for getting *vertically-monotone* chains. Here, it is essential that from each vertex (except the up-most) at least one edge outgoing upwards must exist. One such edge must be added for vertices that do not have any (in Fig. 2.2 b, with dashed line is drawn the edge that should be inserted). This process is carried out by a bottom-to-top plane sweep, during which the vertices without upward outgoing edges are detected, and new edges are inserted. After this process, the vertically-monotone chains can be constructed, and the point-location means checking the position of the query-point against these chains.

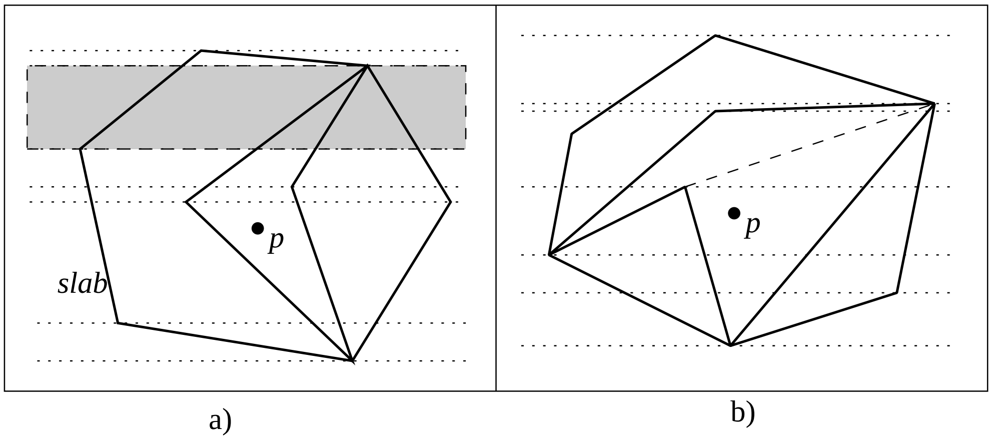

Figure 2.2: Plane-sweep used in point-location: the slabs method (a) and the chains method (b).

Geometry of rectangles

Last but not least, space sweep methods are heavily used in the geometry of rectangles. Here, a set of n-dimensional rectangles, with edges parallel to the axis, is given, and one has to compute a certain parameter of this set: the measure (volume – for three dimensions; area – in the planar case; length – in the linear case); a description of the boundary; number of connected components etc. An excellent chapter on this kind of problems is contained in [PreparataShamos85].

We exemplify here by sketching the algorithm for computing the area of a set of rectangles. Again, a sweeping line goes from left to right. The status, now, should be represented so that the intersection of the vertical line with any of the rectangles in the given set is easily extracted. Clearly, this status changes whenever the line reaches an abscissa that belong to a corner of a rectangle: from this point on, either the line intersects the rectangle, or it stops intersecting it. In Fig. 2.3, at the right of the point marked with a cross on the x axis, the line starts intersecting the shaded rectangle.

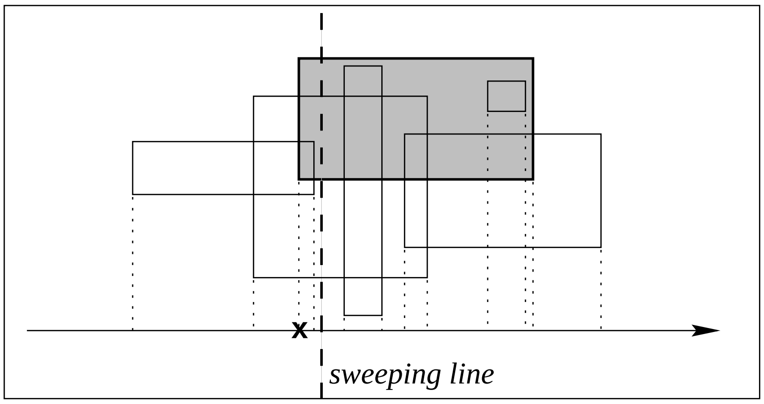

Figure 2.3: Plane-sweep used in computing the area of a set of rectangles.

For an efficient storage of the line status, these algorithms use the *segment tree*, a structure that statically stores segments on the real line; this concept has been invented by Bentley ([Bentley77]). In the same paper, he gives an optimal algorithm for computing the area of a set of rectangles. For multidimensional instances of the measure problem, algorithms have been proposed by Leeuwen and Wood ([LeeuwenWood81]).

All the methods mentioned so far make use of the space-sweep paradigm. Brilliant in its simplicity, this method provides an extremely powerful tool for attacking a broad and diverse spectrum of problems.

The technique that we want to present in this thesis is derived from the space sweep. Instead of sweeping a space with a vertical hyperplane, we sweep a restricted area with a shape anchored on a contour. However, the same concepts appear here as well: event points, status, etc. The main common point is *the linearity of the sweep:* the contour that holds the anchor is a one-dimensional space, topologically homeomorphic to a circle.

We exemplify the usage of anchored sweep by giving solutions to some instances of *optimal placement* problems. The following section gives an overview of the main questions (and their solutions) raised in this context.

2.2 Optimal Placement Problems

In general, one has a set of geometric objects and a given shape, and one wants to get a transformation of this shape so that a certain parameter, which depends on all the objects. The shape can be given either implicitly, by a property (for instance, convexity), or explicitly, by means of some numerical characteristics (coordinates of vertices for polygons/polyhedra, radius for circles, etc.).

Convex Hull

In a *Divide and conquer* approach, the points in the input set are split in two subsets of approximately equal size, the convex hull of these two sets are computed recursively, and then the polygons found in these way are merged into a convex shape.

Smallest Shape-**Type Problems**

Similar problems consider the smallest circle that covers the points in a give set ([Megiddo83]), a rectangle, or multidimensional box ([Toussaint83], [ORourke85]), triangles ([ORourkeAg86]). In all these cases, one has to find a minimal object of the corresponding shape that encloses the maximum number of points (or all points) in S. Except for the circle case, where the geometric object is given and a homothety should be computed, in the other two cases, one looks for an optimal polygon from a given family.

Each of these papers is interesting in its own, since it comes up with a new idea, which can be applied far beyond the scope of its initial usage. Megiddo presents his solution for finding the minimal enclosing circle in a paper that deals with optimal algorithms for two-and three-dimensional linear programming. Among other interesting ideas, he proposes a variant of *pruning techniques*: at each step of the computation, a subset of the data is removed from further considerations, as it does not contain relevant information for the solution.

Toussaint, in the paper [Toussaint83], proposes another interesting method: *rotating calipers*. While computing the smallest rectangle that encloses a set of points, a set of parallel lines rotates continuously; in every moment, the points in the set are *between* these lines. Some particular angles are of interest, and for these angles, a rectangle that covers the points is computed.

This method has been extended by O'Rourke ([ORourke85]) to the three-dimensional case.

When the shape of the polygon/object is preserved, one seeks a homothety (eventually composed with a rigid motion). Examples are finding the smallest homothetic copy of a convex polygon that encloses a set of points, and, closely related, finding the largest placement of one convex polygon inside another ([AgarwalAmentaSharir96]). Other papers consider the largest inscribed isothetic rectangle in a convex polygon ([AltHsuSnoeyink97], [FisherHöffgen94]). The smallest ellipse that covers a set of points is found by an algorithm given by Gärtner and Schönherr ([GaertnerSchoenherr97]).

Preserving the Object

When both the shape and the size of the polygon/geometrical object is preserved, we touch the problems that will be dealt with here. Now one looks for a isometry (be it a rigid motion or a translation) that ensures the maximization of the number of points contained. Of interest are problems of finding the best translation/isometry of a rectangle, circle, convex polygon that contain a maximum number of points from a given set. We will study some planar cases and a three-dimensional instance.

Planar instances of Problem 1.2.2, posed in the Introduction, have been solved in [Chazelle83], [BarequetDP97], [DickersonScharstein96].

The paper [BarequetDP97] gives two algorithms for solving the problem:

Problem 2.2.1. *Given a convex polygon P and a set S of points in plane, find a translation τ such that $\tau(P)$ covers a maximum number of points in S.*

The paper [DickersonScharstein96] contains a solution for the problem:

Problem 2.2.2. *Given a convex polygon P and a set S of points in plane, find a rigid motion τ such that $\tau(P)$ covers a maximum number of points in S.*

A three-dimensional instance of the problem 1.2.2 has been given in [Pau99b]:

Problem 2.2.3. *Given a convex polyhedron P and a set S of points, find a translation τ such that $\tau(P)$ covers a maximum number of points in S.*

Chapter 3

Theoretical Preliminaries

3.1 Usual Notations

We try to situate our theoretical discussions in the most general context in which they are true. Thus, unless stated otherwise, we always work in the Euclidean d-dimensional space E^d, for a finite d. For such a space, we will implicitly assume the existence of a support vector space: For two points p, q in E^d, $\mathbf{v} = q - p$ denotes the d-dimensional vector of source p and target q, and, for a point q and a vector $\mathbf{v}$, $p + \mathbf{v}$ is the unique point q with $\mathbf{v} = q - p$. We denote by $|\mathbf{v}|$ the *norm* of the vector $\mathbf{v}$. As a convention, with bold-faced letters we denote vectors, while the letters denoting points will be written with italics.

A *hyperplane* in a d-dimensional Euclidean space is a the set of all points x for which $< x, a >= c$, for $a \in \mathrm{E}^d$ and $c \in \mathbf{R}$ fixed; we denote such a hyperplane $H(a, c)$ (here $< x, a >$ is the notation for the scalar product of two d-dimensional vectors, $x - O$ and $a - O$). The vector $a - 0$ is the *normal* to the hyperplane. Any hyperplane defines two closed half-spaces, $H(a, c)^+ = \{(x \in \mathrm{E}^d| < a, x >\geq 0\}$ and $H(a, c)^- = \{(x \in \mathrm{E}^d| < a, x >\leq 0\}$.

The distance which we use is the Euclidean distance, computed as the sum of squared differences of coordinates. We will denote it with "dist". The distance from a point p to a geometrical object G (a set of points, or an l-dimensional plane, or a polyhedron etc.) is the minimum distance between p and any point in G.

The projection function will be denoted $\mathrm{proj}_S(O)$, with the meaning: projection of object O on subspace S.

The consecrated notion of *simplex* will be used here with the usual meaning: a d-dimensional simplex is a d-dimensional body with exactly $d+1$ affinely-independent vertices. Analytically, a simplex of vertices $p_1, \ldots, p_{d+1}$ is the set of all points p of the form $p = t_1p_1 + t_2p_2 + \cdots + t_dp_d + (1 - t_1 - t_2 - \cdots - t_d)p_{d+1}$, where $0 \leq t_i \leq 1$ for $1 \leq i \leq d$, and $t_1 + t_2 + \cdots + t_d \leq 1$. In general, d-dimensional simplices of vertices $p_1, p_2, \ldots, p_{d+1}$ are denoted $p_1p_2 \ldots p_{d+1}$.

For a d-dimensional simplex T, the *support* of T, denoted $\mathrm{supp}(T)$, is the hyperplane in the $d+1$-dimensional space that contains T. For instance, $\mathrm{supp}(p_1p_2)$ is the straight line in the plane that contains p_1 and p_2, $\mathrm{supp}(p_1p_2p_3)$ is the plane that contains the triangle $p_1p_2p_3$ in the three-dimensional space.

P is a bounded convex polyhedron (a *polytope*) with m vertices, denoted $p_1, \ldots, p_m$. A k-dimensional face of P will be called simply k-face. The 0-faces are called *vertices*, and the 1-faces - *edges*. The vertices of P are supposed to be in general position: the polytope is *simplicial* (each of its k-faces has exactly $k+1$ vertices). Without loss of generality, we consider the vertex p_1 as the origin (i.e. having coordinates (0,0, … ,0)).

The polytope is viewed as a closed set, that is, $P = \mathrm{interior}\,(P) \cup \mathrm{boundary}\,(P)$. The standard notation ∂P is used to represent its boundary.

A hyperplane $H(a, c)$ *supports* a polytope P if $P \cap H(a, c) \neq \emptyset$ and $P \subset H(a, c)^+$.

The letter τ denotes a translation of arbitrary vector; $\tau_{\mathbf{v}}$ is the translation of vector $\mathbf{v}$. We use the letter ρ for a general rigid motion (an *isometry*), which is the composition of a translation and a rotation. We will not work with symmetries: all the isometries will be in the *special orthogonal group* $SO(d)$, i.e., they can be expressed as $d \times d$ matrices with determinant 1.

We sometimes use the notation $P + \mathbf{v}$ for the polytope $\tau_{\mathbf{v}}(P)$.

We will also work with a set of $n \geq 1$ points: it is usually denoted with S, and its points, with $q_1, \ldots, q_n$. The number of points in a (finite) set A will be denoted by $\#(A)$. The *diameter* of a set of points is the largest distance between any two of these points.

The notion of *containment* of a set of points in a polytope is the same from set theory. We will also say that the polygon P *covers* a set S of points in plane if $S \subset P$.

Throughout this thesis, we will assume that some properties of *general position* are fulfilled:

- no k-face of P is parallel to another l-face of P, for $1 < k, l \leq d-1$;
- no straight-line segment with endpoints in S is parallel to a face of P;
- no k-dimensional subspace determined by $k+1$ points in S is parallel to an l-dimensional subspace determined by other $l+1$ points in S, for $1 \leq k, l \leq d$.

3.2 Theoretical Facts

The most important result, from the point of view of the problems that we try to solve here, is the following:

Theorem 3.2.1. *Let P be a polytope with m vertices in a d-dimensional space, and T be a n-vertex polytope contained entirely in the interior of P, such that no k-face of T is parallel with an l-face of P, for $1 \leq k, l \leq d-1$.*

There is a translation τ such that $\partial\tau(T) \cap \partial P$ contains at least d vertices of $\partial\tau(T)$ (if a vertex of $\partial\tau(T)$ is on a k-face of P, it is counted $d-k$ times).

Proof. We will iteratively determine i vertices of T $(1 \leq i \leq d)$, and translations τ_i, so that $\partial\tau_i(T) \cap \partial P$ contains the i vertices.

Let $q_1, \ldots, q_{d+1}$ denote the vertices of T.

$i = 1$**:** For each vertex q_i of T, let $q_i^* \in \partial P$ be so that

$$\mathrm{dist}(q_i, q_i^*) = \mathrm{dist}(q_i, \partial P).$$

(see Fig. 3.1).

Let $\mathbf{v}_i^* := q_i^* - q_i$, and let k_1 be so that

$$|\mathbf{v}_{k_1}^*| = \min_{1 \leq i \leq d+1} |\mathbf{v}_i^*|.$$

We set $\tau_1 := \tau_{\mathbf{v}_{k_1}^*}$. The image of q_{k_1} under this translation lies on ∂P, and, since $\mathbf{v}_{k_1}$ has the smallest length, no vertex of T is translated out of P.

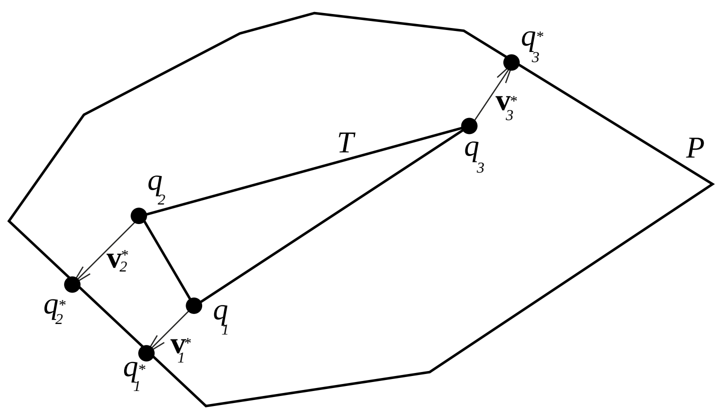

Figure 3.1: Finding the vertex of T closest to the boundary of P, in the 2-dimensional case.

$1 < i \leq d$: We have a translation τ_{i-1} of vector $\mathbf{v}_{i-1}$ such that $\partial\tau_{i-1}(T)$ intersects $i-1$ $d-1$-faces of P; in other words, $i-1$ vertices of $\tau_{i-1}(T)$, not necessarily distinct, lie on the boundary of P.

We can safely consider that the points $q_1, \ldots, q_{i'} \in T$, for i' between 1 and $i-1$, are translated by τ_{i-1} on the boundary of P. The other $d+1-i'$ vertices of $\tau_{i-1}(T)$ lie in the interior of P.

Claim: *There exists a nonempty subset* $\mathbf{W}$ *of a* $(d-(i-1))$*-dimensional vector space so that, for all* $\mathbf{v} \in \mathbf{W}$*,*

$\tau_{\mathbf{v}}(\tau_{i-1}(T)) \subset P$*, and*

$\{\tau_{\mathbf{v}}(\tau_{i-1}(q_1)), \ldots, \tau_{\mathbf{v}}(\tau_{i-1}(q_{i'}))\} \subset \partial P$.

Proof of the Claim. For $1 \leq j \leq i'$, $\tau_{i-1}(q_j)$ belongs to a l-face F_j of ∂P ($l \leq d-1$). Let $\mathbf{V}_j$ be the supporting l-dimensional vector space of the l-plane π_j containing F_j; for at least one vector $\mathbf{v}$ in $\mathbf{V}_j$, $\tau_{\mathbf{v}}(\tau_{i-1}(q_j))$ is in F_j. Let $\mathbf{V}^*_{i-1} := \cap_{1 \leq j \leq i'} \mathbf{V}_j$. The general position of vertices of P ensures that $\mathbf{V}^*_{i-1}$ is a $d-i+1$-dimensional vector space. Moreover, if $\mathbf{v} \in \mathbf{V}^*_i$, $\tau_{\mathbf{v}}(\tau_{i-1}(T))$ keeps every point $\tau_{i-1}(q_j)$ in the plane π_j.

Now, since $\tau_{i-1}(q_j) \in \pi_j$, there is at least one vector in $\mathbf{V}^*_{i-1}$ so that the condition of the claim is fulfilled. This means that for a nonempty subset of vectors in $\mathbf{V}^*_{i-1}$, the corresponding translations keep T inside P, with $q_1, \ldots, q_{i'}$ anchored on the faces $F_1, \ldots, F_{i'}$, respectively.

Proof of Theorem 3.2.1 (continuation). For $i' + 1 \leq k \leq d + 1$, let $\mathbf{v}_k \in \mathbf{V}^*_{i-1}$ be (one of the vectors) with

$$|\mathbf{v}_k| = \min\{|\mathbf{v}| \; |\mathbf{v} \in \mathbf{V}^*_{i-1}, \tau_{i-1}(q_k) + \mathbf{v} \in \partial P\}.$$

Let k', $i' < k' \leq d+1$, be so that $|\mathbf{v}_{k'}| = \min_k |v_k|$. Then the translation $\tau_{\mathbf{v}_{k'}+\mathbf{v}_{i-1}}$ has the property that

$q_1, \ldots, q_{i'}$ are translated on $F_1, \ldots, F_{i'}$, correspondingly,

$q_{k'}$ is translated on the boundary of P, and

no vertex of T is translated in the exterior of P.

■

Remark that this theorem gives a constructive proof of the existence of some particular transformations. The i'th step in the proof suggests a way to iteratively compute the required translation. Intuitively, the computation is carried out by fixing $i - 1$ vertices of T (the *anchors*) on the boundary of P, and then translating T with these vertices anchored on the corresponding faces, until either a new vertex of T touches ∂P, or an anchor touches a lower dimensional face.

In fact, this will be only the first step of the **anchored sweep**: once that enough faces of P are stabbed by the vertices of T, the sweep is further done along a 1-dimensional space.

For the general case, of rigid motions, things are very similar:

Theorem 3.2.2. *Let P be a polytope with m vertices in a d-dimensional space, and T be a n-vertex polytope contained entirely in the interior of P.*

There is an isometry ρ such that $\partial\rho(T) \cap \partial P$ has at least $d+1$ vertices of $\partial\rho(T)$ (if a vertex of $\partial\rho(T)$ is on a k-face of P, it is counted $d - k$ times).

Proof. For the simplicity of presentation, we will give a proof in the planar case. Therefore let P be a convex polygon with m vertices, and T a convex polygon with n vertices contained entirely in the interior of P. From the previous theorem, we may assume that the vertices q_1 and q_2 of T lie on the edges e_1 and e_2 of P.

We start a complex movement of T, by rotating it while preserving, in any moment, q_1 and q_2 on the boundary of P, and the other points of T in the interior of P (see Fig. 3.2).

Again, there are two possibilities:

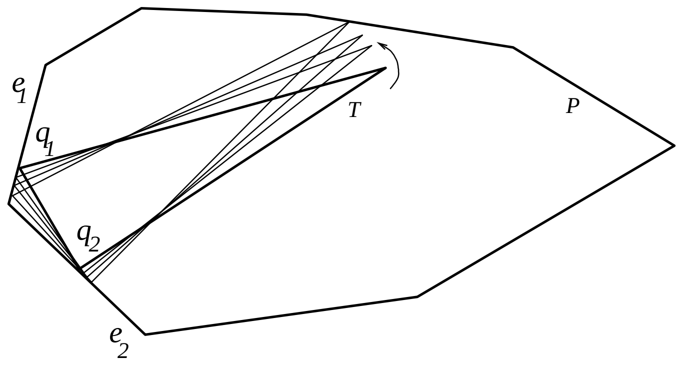

Figure 3.2: Complex movement of T inside P, preserving two of its vertices on ∂P.

- either a third vertex of T touches the boundary of P,
- or one of the anchors reach a vertex of P.

The position of a third vertex of T touching the boundary of P can be found by intersecting the trajectories of the vertices of T with ∂P: we give some details in Appendix 7.1. Since these trajectories are ellipses, they may intersect each edge of P at most two times.

The first intersection reached during this movement is chosen – after this intersection, the corresponding vertex of T leaves P.

If one of the two anchors is brought to a vertex of P, the search stops: that vertex is counted twice, and thus there are three contact vertices.

A similar movement can be defined for multi-dimensional cases. Now one starts with the position of T anchored with d vertices on the boundary of P, as ensured by the previous theorem. The movement is more complex here, since d points should be kept anchored on d $d-1$-dimensional faces. The trajectories of the other vertices of T are accordingly more complex. Again, one must choose the position in which one other vertex of T touches a face of P, while preserving T inside P. $\blacksquare$

Remark 3.2.1. A symbolic study of this movement is rather difficult. For the three-dimensional case, the trajectory of a fourth vertex of a tetrahedron T that moves inside a convex polyhedron P fills a *volume*. Of interest here is the intersection of this volume with the boundary of P: this set contains all the positions of the fourth vertex that correspond to placements of T with all four vertices on ∂P.

3.3 Stable Placements

Turning now back to the main problem that we want to solve, it is clear that there are a (continuous) infinity of transformations that fulfill the desired condition, of maximum containment. The results given in the previous section enable us to look for some very particular transformations, namely, those that anchor the polytope P with a certain subset of S on the boundary.

Definition 3.3.1. (TSP) *P' is a* translation stable placement *(TSP) of the d-dimensional polytope P, with respect to a set of points S, if*

- *there exists a translation τ with $P' = \tau(P)$,*
- *$S \subset P'$ and*
- *$S \cap \partial P'$ has at least d points, if S contains at least d points, or $|S|$, if S contains less than d points.*

Definition 3.3.2. (GSP) *P' is a* general stable placement *of the d-dimensional polytope P, with respect to a set of points S, if*

- *there exists a rigid motion ρ with $P' = \rho(P)$,*
- *$S \subset P'$ and*
- *$S \cap \partial P'$ has at least $d+1$ points, if S contains at least $d+1$ points, or $|S|$, if S contains less than $d+1$ points.*

The existence of TSP's has been proven by Theorem 3.2.1, and the existence of GSP's, by Theorem 3.2.2. In addition, we have the following corollaries:

Corollary 3.3.1. (Existence of TSP's) *Let τ^* be a translation such that*

$$\#(\tau^*(P) \cap S) = \max_{\tau \text{ translation}} \{\#(\tau(P) \cap S)\},$$

and let $S' := \tau^*(P) \cap S$. Then there exists a TSP of P with respect to S'.

Corollary 3.3.2. (Existence of TSP's) *Let ρ^* be a rigid motion such that*

$$\#(\rho^*(P) \cap S) = \max_{\rho \text{ isometry}} \{\#(\tau(P) \cap S)\},$$

and let $S' := \rho^*(P) \cap S$. Then there exists a GSP of P with respect to S'.

These two corollaries focus the field of research from the (uncountably) infinitely many translations/isometries to stable placements. Problem 1.2.2 can be now rewritten as:

Problem 3.3.1. (Stable Placement of Max Containment) *Given a convex polytope P and a set of points S, find a stable placement (TSP or GSP) P' of P so that P' contains a maximal subset of S.*

It remains to study the possibility of fast computation of stable placements. The basic idea that will be tracked is that if two points lie on ∂P, then the translations of P with p_1 in these two points intersect. The following section contains some useful results.

3.4 Characterization of Stable Placements via Intersection

We give in this section two theorems that relate translation stable placements to polygons/polyhedra intersections. In this way, we bring the matters on the familiar ground of computing intersections, instead of using the method suggested by Theorem 3.2.1.

Let q_1 and q_2 be two points in plane, chosen so that there exists at least a TSP of P with respect to them. The problem we want to solve in the following is *finding the TSP's of P with respect to q_1 and q_2.*

Figure 3.3 gives an idea of how TSP's can be related to intersections.

Assume that q_1q_2 is not parallel to any edge of the convex polygon P. Let $-P$ be the symmetric of P with respect to p_1. Translate $-P$ so that p_1 goes to q_1, and then to q_2. Let r be an intersection point of these two translated copies of $-P$.

Lemma 3.4.1. *$\tau_{r-p_1}(P)$ is in TSP with respect to q_1 and q_2.*

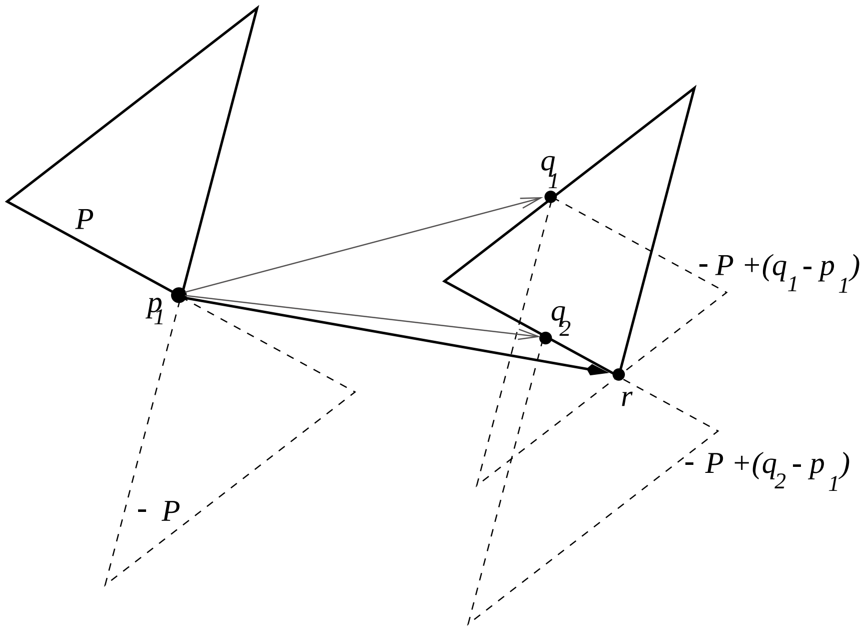

Figure 3.3: Finding a TSP of P with respect to q_1 and q_2.

Proof. Since r belongs to the boundary of $\tau_{q_1-p_1}(P)$, the vector $r-q_1$ joins p_1 with a point of $\partial(-P)$. Accordingly, $p_1+(q_1-r)$ will be on ∂P.

Same is true for q_2: The vector $p_1+(q_2-r)$ will be on ∂P.

Therefore,

$$q_1 = (p_1 + (r - p_1)) + (q_1 - r)$$

will be on $\partial\tau_{r-p_1}(P)$, and

$$q_2 = (p_1 + (r - p_1)) + (q_2 - r)$$

will be on $\partial\tau_{r-p_1}(P)$. ■

This result can be easily generalized:

Theorem 3.4.1. (Connecting Intersections with TSP's) *Let P be a d-dimensional polytope, $q_1,\ldots,q_d$ points in space, in general position, p_1 a vertex of P, and $-P$ the symmetric of P with respect to p_1. The vector $\mathbf{v}$ has the property that $\tau_{\mathbf{v}}(P)$ is in TSP with respect to $q_1,\ldots,q_d$ iff*

$$\mathbf{v} = r - p_1, \textit{ where } r \in \bigcap_{1\le i\le d} \tau_{q_i-p_1}(-P).$$

Proof. Again, the proof is based on symmetry arguments. We have:

$$\begin{gathered}\tau_{\mathbf{v}}(P) \text{ is in TSP with respect to } q_1, \ldots, q_d \\ \text{iff} \\ q_i = \tau_{\mathbf{v}}(p_1) + (q_i - \tau_{\mathbf{v}}(p_1)) \in \partial\tau_{\mathbf{v}}(P), \text{ for } 1 \leq i \leq d \\ \text{iff} \\ r := q_i + (\tau_{\mathbf{v}}(p_1) - q_i) \in \tau_{q_i - p_1}(-P), \text{ for } 1 \leq i \leq d \\ \text{iff} \\ \mathbf{v} = r - p_1, \text{ where } r \in \cap_{1 \leq i \leq d} \tau_{q_i - p_1}(-P).\end{gathered}$$

■

We have now a tool at hand for computing easily the TSP's of a convex polygon. It remains to discuss how intersections of convex polytopes can be computed, and how fast. The following subsection offers a survey of results in this area.

3.4.1 Intersections of Convex Shapes

Computing intersections of geometric objects is probably one of the most discussed topics in the literature. The results are extremely numerous, regarding all possible instances of the problem, be them planar or multidimensional, convex or non-convex, straight-lined objects or having other kinds of algebraic varieties as boundary.

The first linear-time algorithm for the intersection of two planar convex polygons has been invented by Shamos in his PhD thesis ([Shamos78]). Among other important results, we should mention the *star of rays* method of Shamos and Hoey ([ShamosHoey76]), the sickle method of O'Rourke et al ([ORourkeCh82]), both for planar cases, and, for the three-dimensional case, the results of Chazelle ([Chazelle92]). All these algorithms work in time linear with the number of vertices of the two shapes to be intersected.

In two dimensions we can take advantage of the specific situation: we have two identical convex polygons whose intersection we want to compute. Namely, we want to have a fast algorithm that solves the following problem:

Problem 3.4.1. (Fast Planar Intersection)
Let P be a convex polygon with m vertices, and let τ_1 and τ_2 be translations. Compute the intersections between $\delta\tau_1(P)$ and $\delta\tau_2(P)$.

This problem can be approached by using the tentative prune-and-search techniques of Kirkpatrick and Snoeyink ([KirkSnoeyink95]). In their paper, they give a proof of the following lemma:

Lemma 3.4.2. (Fast Intersection of Identical Convex Polygons) *The intersection of two copies of a m-vertices convex polygon can be found in $O(\log m)$ time.*

They show first that the fixed point of the composition of a monotone increasing piecewise-basic function and a monotone decreasing piecewise-basic function can be computed in $O(\log m)$ time, when both of them have $O(m)$ linear components. As an application, they give an $O(\log m)$ algorithm for computing both the longest vertical chord, and a vertical chord of given length. The problem of computing the intersection points of $\partial\tau_{\mathbf{v_1}}(P)$ and $\partial\tau_{\mathbf{v_2}}(P)$ can be reduced to finding the two chords of length $|\mathbf{v_2} - \mathbf{v_1}|$ and direction given by $\mathbf{v_2} - \mathbf{v_1}$.

Since the proof of this theorem is constructive, and consists in fact in an algorithm for computing intersections, we postpone it to Chapter 4, Section 4.2.1.

In three dimensions, the best algorithm for intersecting planar shapes was invented by Chazelle ([Chazelle92]). It works in linear time with respect to the number of vertices of the two polyhedra to be intersected. In Section 4.2.2, we will give an informal description of his algorithm. In the same section, we describe a slightly simpler algorithm dedicated to the computation of the intersection of two identical polyhedra; we shall give then a constructive proof of the following lemma:

Lemma 3.4.3. *The intersection of two translated copies of a m-vertex convex polyhedron can be computed in $O(m)$ time.*

3.5 Rectangular Boxes and Convex Polytopes

We give in this section a result concerning the relation between a rectangular box that inscribes a non-degenerate d-dimensional polytope P and a box inscribed in P. This result will be used to justify the speed-up gained by some of the algorithms presented in this paper, when the search is restricted to points contained in neighboring rectangular boxes that enclose the polytope.

Theorem 3.5.1. (Rectangular boxes) *There exists a rectangular box R_P that encloses P and is no more than $2^{d-1}d!$ times the volume of P, and there exists a rectangular box R_I that is included in P, is orthogonal to R_P, and is at least $1/2^{\frac{(d+4)(d-1)}{2}}$ the volume of R_P.*

Proof. For proving this theorem, we iteratively construct pairs of parallel hyperplanes that enframe the polytope, until a d-dimensional rectangular box is created. Then we show that a rectangular box included in P, parallel to the one just constructed, exists. The ratia among the volumes of the boxes and polytope are proven to be as claimed.

Let p_1 and p_2 be the most distant vertices of P. These points define the first pair of parallel hyperplanes: $\pi_1||\pi_1'$, with $p_1 \in \pi_1, p_2 \in \pi_1'$, and both of them orthogonal to the segment p_1p_2. The distance between these hyperplanes equals the length of the diameter of P.

Among the rest of P vertices, let p_3 be the most distant point to p_1p_2, and let p_3' be the projection of p_3 on p_1p_2. The second pair of parallel hyperplanes can be now defined: $\pi_2||\pi_2'$, $p_3 \in \pi_2$, $p_3p_3' \perp \pi_2, \pi_2'$, and $\mathrm{dist}(\pi_2, \pi_2') = \mathrm{length}(\mathrm{proj}_{\mathrm{supp}(p_3p_3')}(P))$. The distance between these hyperplanes is less than $2\mathrm{dist}(p_3, p_3')$; moreover, π_1, π_1' are orthogonal to π_2, π_2', as their normals, namely p_1p_2 and p_3p_3', are orthogonal.

Figure 3.4 contains a planar construction, and Figure 3.5 a three-dimensional one.

Iteratively, assume that $k-1$ pairs of hyperplanes, $\pi_1, \pi_1', \ldots, \pi_{k-1}, \pi_{k-1}'$, $1 < k < d$, have been defined, by constructing simplices as suggested in the previous paragraphs. That is,

$$\begin{aligned}
\pi_1, \pi_1' \perp p_1p_2, \quad & \mathrm{dist}(\pi_1, \pi_1') = \mathrm{dist}(p_1, p_2) = \\
& \quad = \max\{\mathrm{dist}(p_i, p_j) | p_i, p_j \text{ vertices of } P\}; \\
\pi_2, \pi_2' \perp p_3p_3', \quad & \mathrm{dist}(p_3, p_1p_2) = \max\{\mathrm{dist}(p, p_1p_2) | p \text{ vertex of } P\}, \\
& p_3' = \mathrm{proj}_{\mathrm{supp}(p_1p_2)}(p_3), \\
& \mathrm{dist}(\pi_2, \pi_2') = \mathrm{length}(\mathrm{proj}_{\mathrm{supp}(p_3p_3')}(P)); \\
& \ldots \\
\pi_{k-1}, \pi_{k-1}' \perp p_kp_k', \quad & \mathrm{dist}(p_k, p_1p_2 \ldots p_{k-1}) = \\
& \quad = \max\{\mathrm{dist}(p, p_1p_2 \ldots p_{k-1}) | p \text{ vertex of } P\}, \\
& p_k' = \mathrm{proj}_{\mathrm{supp}(p_1p_2 \ldots p_{k-1})}(p_k), \\
& \mathrm{dist}(\pi_{k-1}, \pi_{k-1}') = \mathrm{length}(\mathrm{proj}_{\mathrm{supp}(p_kp_k')}(P)).
\end{aligned}$$

Let p_{k+1} be the most distant vertex to $p_1p_2 \ldots, p_k$, and p_{k+1}' its projection on $\mathrm{supp}_{p_1p_2 \ldots p_k}$. The k'th pair of parallel hyperplanes, π_k, π_k', is orthogonal to $p_{k+1}p_{k+1}'$, $p_{k+1} \in \pi_k$, and the distance between π_k and π_k' is

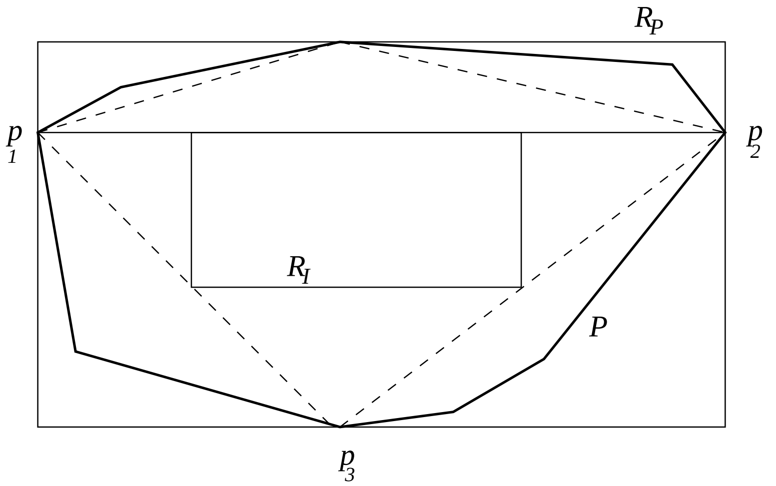

Figure 3.4: Construction of rectangles R_P, that encloses P, and R_I, contained in P.

$\text{length}(\text{proj}_{\text{supp}(p_{k+1}p'_{k+1})}(P))$.

We continue this process until d pairs of enframing P, parallel and reciprocally orthogonal hyperplanes will have been constructed. They define a d-dimensional rectangular box R_P that encloses P tightly: each $d-1$-dimensional facet of R_P touches at least one vertex of P.

We denote by l_i the distances between π_i and π'_i, for $1 \le i \le d$, and by h_i the distances between p_{i+1} and p'_{i+1}, for $2 \le i \le d$; we also use $\text{height}(R_P)$ for h_d. From the particular way of constructing the pairs of parallel hyperplanes, $l_1 = h_1 = \text{dist}(p_1, p_2)$, and, for $2 \le i \le d$, $l_i \le 2h_i$. The volume of $p_1 p_2 \dots p_{d+1}$ is

$$\underbrace{(\,\underbrace{(\,\underbrace{(h_1 \times h_2)/2}_{\text{Vol}(p_1p_2p_3)} \times h_3)/3}_{\text{Vol}(p_1p_2p_3p_4)} \times h_4)/4}_{\text{Vol}(p_1p_2p_3p_4p_5)} \times \cdots \times h_d)/d = \frac{1}{d!} \prod_{1 \le i \le d} h_i,$$

and the volume of R_P is

$$\prod_{1 \le i \le d} l_i.$$

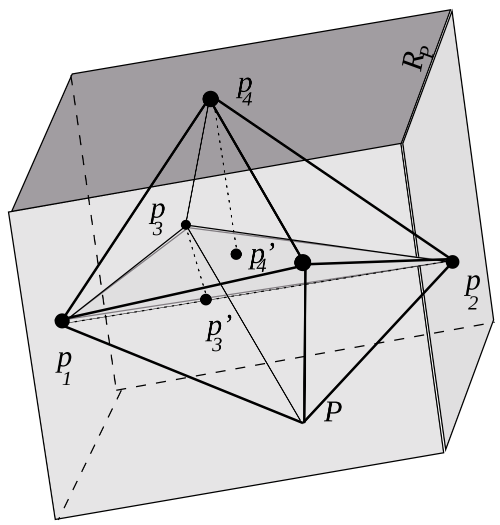

Figure 3.5: Construction of the rectangular box R_P that encloses P.

It follows:

$$\begin{aligned}\mathrm{Vol}(R_P) &= \prod_{1\le i\le d} l_i \\ &\le 2^{d-1}\prod_{1\le i\le d} h_i \\ &= 2^{d-1}d!\frac{1}{d!}\prod_{1\le i\le d} h_i \\ &= 2^{d-1}d!\cdot \mathrm{Vol}(p_1p_2\dots p_{d+1}).\end{aligned}$$

Since the simplex $p_1p_2\dots p_{d+1}$ is included in P, the volume of R_P is less than $2^{d-1}d!$ the volume of P, as claimed.

We construct now inductively the rectangular box R_I, included in P, with facets parallel to those of R_P.

For $d = 2$, the construction is suggested in Fig. 3.4. In triangle $p_1p_2p_3$, a segment parallel to p_1p_2, that contains the middle point of p_3p_3', with endpoints on edges p_1p_3 and p_2p_3, is drawn. This segment and its projection on p_1p_2 constitute two edges of R_I, and the segments that join their endpoints, the other two edges. The edges of R_I are parallel to corresponding edges of R_P, due to their mutual relation to the diameter of P, p_1p_2.

We generalize this construction to d dimensions. The hyperplane $H_{base} = \mathrm{supp}(p_1 p_2 \ldots p_d)$ will hold a $(d-1)$-dimensional base of R_I. Let T_{half} be the intersection of the simplex $p_1 p_2 \ldots p_{d+1}$ with the hyperplane parallel to H_{base} that contains the middle point of $p_{d+1}p'_{d+1}$. The base $R_{I_{base}}$ of R_I will be constructed, using the same method, in $\mathrm{proj}_{H_{base}}(T_{half})$. The height of R_I is $\mathrm{dist}(p_{d+1}, p'_{d+1})/2$. A three-dimensional example is drawn in Fig. 3.6.

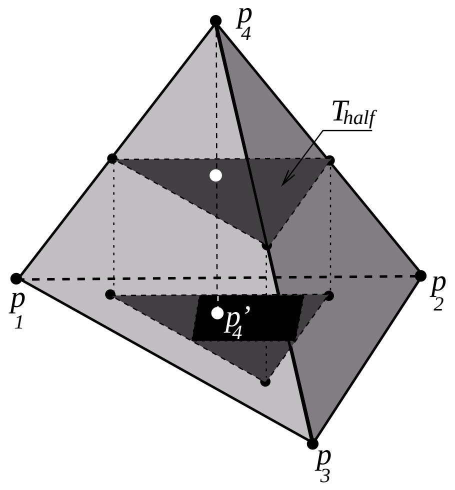

Figure 3.6: Construction of the rectangular box R_I contained in P.

For computing the ratio of volumes, we prove first, by induction on the number of dimensions d, that:

$$\mathrm{Vol}(R_I) = \frac{d!}{2^{2+3+\cdots+d}} \mathrm{Vol}(p_1 p_2 \ldots p_{d+1}),$$

for $d \geq 2$.

For $d = 2$, the height of R_I is half of $\mathrm{dist}(p_3, p'_3)$ (see Fig. 3.4), and the length of R_I is half of $\mathrm{dist}(p_1, p_2)$. Therefore:

$$\begin{aligned}
\mathrm{Vol}(R_I) &= 1/4 \cdot \mathrm{dist}(p_1, p_2) \cdot \mathrm{dist}(p_3, p'_3) \\
&= 1/2 \cdot (1/2 \cdot \mathrm{dist}(p_1, p_2) \cdot \mathrm{dist}(p_3, p'_3)) \\
&= \frac{1}{2} \mathrm{Vol}(p_1 p_2 p_3) \\
&= \frac{1 \cdot 2}{2^2} \mathrm{Vol}(p_1 p_2 p_3) \\
&= \frac{2!}{2^2} \mathrm{Vol}(p_1 p_2 p_3).
\end{aligned}$$

We assume the formula is true for $d-1$, i.e.,

$$\mathrm{Vol}(R_{I_{base}}) = \frac{(d-1)!}{2^{2+3+\cdots+(d-1)}}\mathrm{Vol}(T_{half}).$$

We can write:

$$\begin{aligned}\mathrm{Vol}(R_I) &= \mathrm{Vol}(R_{I_{base}}) \cdot \frac{\mathrm{dist}(p_{k+1}, p'_{k+1})}{2} \\ &= \frac{(d-1)!}{2^{2+3+\cdots+(d-1)}}\mathrm{Vol}(T_{half}) \cdot \frac{\mathrm{dist}(p_{k+1}, p'_{k+1})}{2},\end{aligned}$$

$$\begin{aligned}\mathrm{Vol}(p_1p_2\ldots p_{k+1}) &= \frac{1}{d}\mathrm{Vol}(p_1p_2\ldots p_k) \cdot \mathrm{dist}(p_{k+1}, p'_{k+1}), \\ \mathrm{Vol}(p_1p_2\ldots p_k) &= 2^{d-1} \cdot \mathrm{Vol}(T_{half}),\end{aligned}$$

therefore

$$\begin{aligned}\mathrm{Vol}(R_I) &= \frac{(d-1)!}{2^{2+3+\cdots+(d-1)}}\mathrm{Vol}(T_{half}) \cdot \frac{\mathrm{dist}(p_{k+1}, p'_{k+1})}{2} \\ &= \frac{(d-1)!}{2^{2+3+\cdots+(d-1)}} \cdot \frac{1}{2^{d-1}}\mathrm{Vol}(p_1p_2\ldots p_k) \cdot \frac{\mathrm{dist}(p_{k+1}, p'_{k+1})}{2} \\ &= \frac{(d-1)!}{2^{2+3+\cdots+(d-1)} \cdot 2^{d-1} \cdot 2} \cdot d \cdot \mathrm{Vol}(p_1p_2\ldots p_{k+1}) \\ &= \frac{d!}{2^{2+3+\cdots+d}}\mathrm{Vol}(p_1p_2\ldots p_{d+1}).\end{aligned}$$

We have already shown that

$$\mathrm{Vol}(p_1p_2\ldots p_{k+1}) \geq \frac{1}{d! \cdot 2^{d-1}}\mathrm{Vol}(R_P).$$

We have thus:

$$\begin{aligned}\mathrm{Vol}(R_I) &= \frac{d!}{2^{2+3+\cdots+d}}\mathrm{Vol}(p_1p_2\ldots p_{d+1}) \\ &\geq \frac{d!}{2^{2+3+\cdots+d}} \cdot \frac{1}{d! \cdot 2^{d-1}}\mathrm{Vol}(R_P) \\ &= \frac{1}{2^{\frac{(d+4)(d-1)}{2}}}\mathrm{Vol}(R_P).\end{aligned}$$

■

What we have proven here is the fact that, for every fixed dimension d, there are such boxes with volumes in constant ratia with respect to the volume of P. The formulae seem rather pessimistic; however, for $d = 2$ or $d = 3$, the constants are reasonable, for the purposes of our algorithms. Moreover, the rectangular boxes that we described are by no means the best choices: finding such pairs of boxes that satisfy some criteria of optimality is a problem in itself.

As a consequence, for every d-dimensional polyhedron P, there exists a box R_P that contains it and that is contained into the union of a constant (up to the dimension d) number c of translated copies of P. Therefore, if k given points lie inside R_P, there exists a translation of P that covers at least k/c points. The following corollary is now immediate:

Corollary 3.5.1. *Let S be a set of points and P a d-dimensional polytope. Then there exists a rectangle R_P enclosing P with the following property: If there exists a translation $\tau(R_P)$ of R_P containing k points of S, then there exists a translation $\tau^*(P)$ of P containing $\Omega(k)$ points of S.*

3.6 Layers in Arrangements

Although it does not fall completely under the scope of this investigation, we will present an algorithm for the planar translational instance of Problem 1.2.2. It has been proposed in the paper of Barequet, Dickerson & Pau [BarequetDP97], and it has the advantage of working with consecrated structures in computational geometry – *chains of edges* – and of applying a consecrated paradigm – *plane sweep*. The algorithm is presented in Section 5.1.4.

For a comprehensive view on arrangements of geometrical figures, we refer to the excellent book of Edelsbrunner [Edelsbrunner87]. The theoretical result that will be used in order to estimate the efficiency of the algorithm has been published by Sharir ([Sharir91]).

Usually, an *arrangement* of geometrical figures is a set of such figures disposed in a certain way (more often at random). In the case of planar polygons, the arrangement can be pretty complicated, if they intersect. Here we talk about a set of n copies of a unique convex polygon P, translated so that a vertex goes to points from a set S. The *depth* of a point, with respect to this arrangement, is the number of polygons to which it belongs. In Fig.

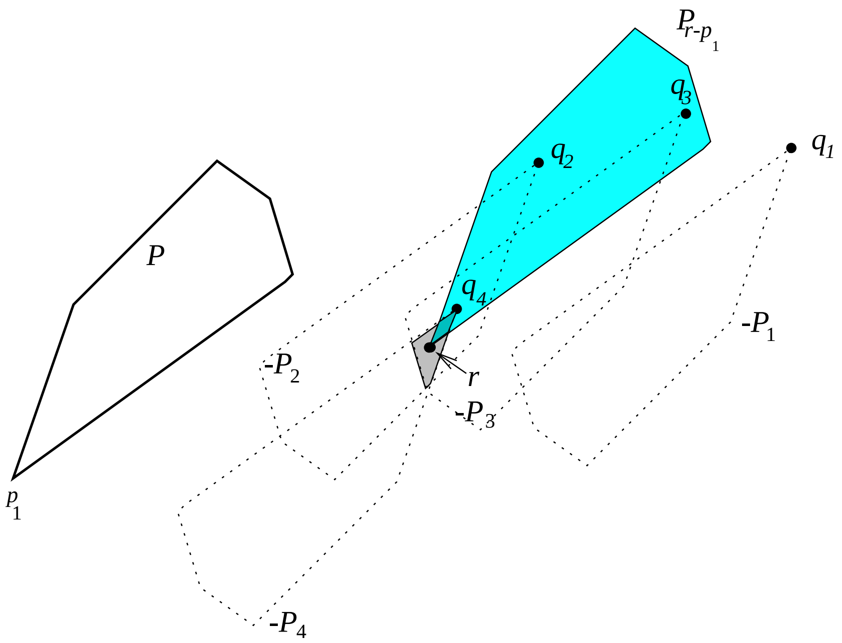

Figure 3.7: Arrangements of polygons in the plane.

3.7, point r belongs to three polygons, $-P_2, -P_3, -P_4$: its depth if 3, as is the depth of all points in the shaded region.

The colored polygon P_r contains q_2, q_3 and q_4. The following lemma gives a formal explanation of this fact:

Lemma 3.6.1. *Let $-P$ be the symmetric of P with respect to its vertex p_0, and let $-P_i := \tau_{q_i - p_0}(-P)$, $1 \leq i \leq n$. Any point r in the region of maximum depth in the arrangement of $-P_i$ has the property that $\tau_{r-p_0}(P)$ contains a maximum number of points in S.*

Proof. The proof of this lemma is immediate. Indeed, assume r is a point in the region of maximum depth in the arrangement of $-P_i$ (see Fig. 3.7).

It is clear that, for a certain index k between 1 and n, $r \in -P_k$ iff $q_k \in \tau_{r-p_0}(P)$ (the argument is similar to the one used in Lemma 3.4.1). If $r \in -P_j$, for $1 \leq j \leq l, l \leq n$, then, as before, each q_j lies in $\tau_{r-p_0}(P)$.

Since r is a point in a region of maximum depth, it belongs to a maximum number of translated copies of P_f; accordingly, a maximum number of points in S will lie in $\tau_{r-p_0}(P)$. ∎

A bound on the number of intersection points among a set of n copies of P is given by the following theorem:

Theorem 3.6.1. *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n translated copies of a convex polygon P. If the maximum depth of any point in the subdivision of the plane determined by these polygons is k, then the number of boundary intersections is $\leq cnk$, for a positive constant c.*

This theorem is a direct consequence of the following theorem, given by Sharir ([Sharir91]):

Theorem 3.6.2. *Let $\mathcal{A}$ be an arrangement formed by n shapes in the plane, having the property that the boundaries of each pair of shapes intersect at most twice. The number of intersection points of depth $\leq k$ is $\leq 26nk$.*

We do not insert here the complete proof, which uses probabilistic and combinatorial methods. We only present a sketch of the proof, in a few sentences:

- for an intersection point p and a randomly chosen subset R of $r = \lfloor \frac{n}{k} \rfloor$ shapes, the expected value of the number of intersections among shapes in R that do not fall inside other shape in R (denoted by $f_0(R)$) is computed in two ways;
- first, by adapting Clarkson and Shor's [ClarksonShor89] probabilistic techniques, the author shows that

$$\mathrm{E}(f_0(R)) \geq \frac{r(r-1)}{4n(n-1)} \cdot \sum_{j=0}^{k} \#(S_j),$$

  where S_j denotes the set of all intersections among shapes in $\mathcal{A}$ that fall under exactly j other shapes in $\mathcal{A}$;
- from a result of Kedem et al [Kedem86], $f_0(R) \leq 6r$;

and, by a simple computation, the inequality in the theorem is revealed.

3.7 Complexity Bounds

This section contains a result about the absolute lower bound of any instance of Problem 1.2.2. This results has been published in [Pau99a].

First of all, we introduce the general framework of such problems of complexity, presenting the techniques used in proving the bound. Subsection 3.7.2 contains the main result, and its proof.

3.7.1 Theoretical Techniques for Determining Lower Bound

Any problem $\mathcal{P}$ for which one proposes an algorithm challenges the researcher with the question:

Problem 3.7.1. *How fast is the* fastest *algorithm which solves $\mathcal{P}$?*

The number of operations of the optimal algorithm – this is, in fact, the answer to the problem 3.7.1 – is called *lower bound*; an important branch of the theory of algorithms studies bounds and optimality. It is very important to know the lower bound; it may justify further research on the problem, to generate faster algorithms; on the other hand, an algorithm whose order equals the lower bound is certainly optimal, and any attempt to improve its asymptotic behavior would be futile.

The techniques used to determine the lower bound are very specialized: *binary decision tree*, *oracles*. A good introduction in Lower Bound Theory is presented in [HorowitzSahni-a82]. We will use here some arguments given by Shamos and Preparata ([PreparataShamos85]).

One way to find out the lower bound for a certain problem is by "transforming" that problem (formally) to a problem whose lower bound is already known. There are quite a few such known problems: they are named *computational prototypes*, and their usefulness is out of any doubt. The computational time for such prototypes has been determined using that kind of techniques mentioned before (oracles, etc.).

The following definition gives a formal shape to transformability:

Definition 3.7.1. (Transformability)
The problem $\mathcal{P}_1$ is $t(N)$-transformable in the problem $\mathcal{P}_2$ if

- *the INPUT of the problem $\mathcal{P}_1$ can be converted in $t(N)$ operations into a suitable INPUT for the problem $\mathcal{P}_2$*
- *the OUTPUT of the problem $\mathcal{P}_2$ can be transformed in $t(N)$ operations in a correct OUTPUT for the problem $\mathcal{P}_1$, OUTPUT that corresponds to the initial INPUT.*

Here N is the size of the problem $\mathcal{P}_1$.

The next two propositions are fundamental in the lower bound theory. Indeed, they characterize the power of this technique. Their proof is evident.

Theorem 3.7.1. *If $T(N)$ operations are required to solve the problem $\mathcal{P}_1$ and $\mathcal{P}_1$ is $t(N)$-transformable in $\mathcal{P}_2$, then $\mathcal{P}_2$ requires at least $T(n) - O(t(N))$ time*[1].

Theorem 3.7.2. *If problem $\mathcal{P}_2$ can be solved in $T(N)$ time and $\mathcal{P}_1$ is $t(N)$-transformable in $\mathcal{P}_2$, then $\mathcal{P}_2$ can be solved in at most $T(n) + O(t(N))$ operations.*

3.7.2 A Lower Bound for Problems of Optimal Placement

It is easy to see that the lower bound for the one-dimensional instance of Problem 1.2.2 will be *a fortiori* a lower bound for higher dimensions.

The computational prototype that will be used in determining the desired lower bound is the ϵ-closeness problem.

In [PreparataShamos85], the problem of ϵ-closeness is defined as follows:

Problem 3.7.2. *Given n real numbers and $\epsilon > 0$, determine whether any two of them are at distance less than ϵ.*

The authors present the result:

Lemma 3.7.1. *Any algorithm that solves the problem 3.7.2 requires $\Omega(n \log n)$ operations.*

Based on this lemma, we will prove the following:

[1] In other words, its complexity is $\Omega(T(N) - O(t(N)))$

Theorem 3.7.3. *Any algorithm that solves Problem 1.2.2 requires at least $\Omega(n \log n)$ operations (n is the number of points in S).*

Proof. It is clear that the complexity of the polytope P decreases with he dimension of the space: in the one-dimensional case, P will be a closed interval. More precisely, the problem 1.2.2 reduces to:

Problem 3.7.3. *Let S be a set of n real numbers and P a closed interval. Determine a real value v such that a maximum number of elements in S lies in $P + v$.*

Let A an optimal algorithm for the problem 3.7.3 and let ϵ a positive real number. For a set S of real numbers and the interval $[0, \epsilon]$, after applying the algorithm A, a value v will be computed, such that the interval $[v, v + \epsilon]$ contains a maximum possible number of elements in S. If $[v, v + \epsilon]$ contains more than a S number, then a positive answer is obtained for the ϵ-closeness problem.

The number of operations necessary for the transformations is constant, for the input for the ϵ-closeness problem is transmitted as it is to the problem 3.7.3, and the output of problem 3.7.3 – consisting in the optimal geometric transformation and the number of points contained – may be obtained in a constant number of operations from the result of the ϵ-closeness problem.

Therefore the ϵ-closeness problem is $O(1)$-transformable to the problem 3.7.3. It follows that this last problem requires $\Omega(n \log n)$ operations. ∎

The lower bound determined before is **absolute**; for a specific dimension d and for a certain type of geometric transformation, one expects that the algorithms solving these instances of problem 1.2.2 will require more operations.

For the one-dimensional case, there exists a simple algorithm that solves Problem 3.7.3. It consists in sorting the points in S, followed by a special sweep of these points, in which the interval P moves from left to right, recording the position of maximum cover. A detailed description of this algorithm is presented in Section 5.3.

For the two-dimensional case, this lower bound is attained by an algorithm that finds an optimal placement of a rectangle. Basically a plane-sweep type algorithm, it is presented in Section 5.4.

Chapter 4

Basic Algorithms and Data Structures

In this chapter, we give an overview of some general algorithms used to side-processings the input data, or which, due to their efficiency, will be used in order to get some speed-ups. We also present some data structures, both basic and geometric.

We start by mentioning a few general data structures, together with the associated operations. More specific data structures are then given in details, structures dedicated to store geometric information.

We also present here generic algorithms that are practically well known – like sorting, searching etc. For these, we give a short description and we refer to the most prominent books where the reader can find them. More specific algorithms are presented in some details, algorithms that handle geometric data.

4.1 Data Structures

The data structures dedicated to storing geometrical objects should be complex enough as to express the geometrical relationships, as well as able to permit the necessary speed to the algorithms that consult and modify them. There are some underlying structures that are used in most of the cases, structures, in fact, well studied. Clearly, apart from the basic operations, the geometrical nature of the data imposes some specific processings.

We give in the following a short exposé of the data structures used in our algorithms. We begin with a short collection of consecrated structures – *lists* and versions of lists. The *trees*, more exactly *binary trees*, and *hash-tables* are also mentioned.

More specific data structures described here are *doubly connected edge lists, queues of events, segment trees*, and *trees of chains*.

4.1.1 General Purpose Data Structures

Perhaps the most common data structure is **the list**. The relation imprinted in this structure is *succession*. Presented in a large variety, it is used to store ordered or unordered sets of points, vertices of polygons (in *circular lists*) or more complicated collections of objects.

We refer to the books [AhoHU83], [CormenLR90] and [HorowitzSahni-d82] for excellent expositions of lists and operation on lists. In fact, these books contain all the material related to the other structures that will be discussed here.

The simplest variant of lists allows only to iteratively process their elements. The operations defined on *linear lists* are: insertion of elements (only at one end or at both ends), removal of elements, and iterative access (for processing) to every element in turn. Circular lists have in addition the property that after the last element, the first one comes.

Insertion and deletion of elements in lists take constant time, when they are done at the ends. Insertion and deletion of elements inside the list take usually linear time with respect to the length of the list.

The **trees** are more complicated structures, involving a certain relation among the information recorded in their *nodes*. The most interesting, from our point of view, are *binary trees*: every element is linked to at most two *descendents* and at most one *ancestor*. The node without ancestor is called *root*, and the nodes without descendants are called *leaves*. Following the links between nodes, *branches* that connect the root to leaves are created. Nodes are naturally layered into *levels*, defined by their distance to the root. Usually, the set of nodes created by the transitive closure of *descendent* relation are disjoint, for distinct nodes on the same level. Every node is the root of the tree that has its descendants only – this is called a *subtree*.

When the nodes are arranged as close to the root as possible, the tree has a minimum number of levels, and the branches will be small. *Traversing* the tree – i.e., passing from the root to a leaf, following the links ancestordescendant, is very fast: in fact, it is proportional to the number of levels. It is obvious that, in a binary tree with n nodes, the number of levels can be as small as $\lfloor \log_2(n) \rfloor + 1$. This observation makes the trees ideal for fast retrieval of information, when a certain relation among the records is defined (usually an ordering), and when insertion/elimination of nodes should also be provided.

There are many variants of binary trees, when the relation among the nodes is an order. *Balanced trees* have minimum height (i.e., number of levels), thus minimum traversal time: these are the most attractive, from the performance point of view. The order can be materialized in different ways. In an *ordered tree*, for every non-leaf node, the descendants in the left subtree are smaller than itself, and it is smaller that its descendants in the right subtree. In a *priority queue*, the root is the smallest element, and each node is the smallest in its subtree.

Basic operations on trees are *insertion and removal* of nodes, traversal and search. It is no trivial task to design these operations, in order to preserve the structure of the tree. However, these operations take logarithmic time with respect to the number of elements in the structures, in other words, proportional to the length of the longest branch.

Hash tables are, in a way, extensions of arrays. The *indexing* process is now more complicated. The position of an element in a hash table depends on the value of a certain key. This position is computed using a *hash function.*

The main issue of this structure resides in the fact that each entry in a hash table can store a (rather small) number of elements, usually bounded by a constant, and the position of the entry that corresponds to an element is computed in constant time, using the key and the hash function. The main operations are insertion, deletion and search. All these operations are done in constant time.

The structures introduced so far can be extended, so that they fit to applications where more complex operations are necessary. For instance, a priority queue can store points in an Euclidean space, and the ordering criterion can be the order on a certain coordinate. Also, in an ordered tree, the nodes can store complex objects. The next section presents some dedicated structures.

4.1.2 Storing Geometric Data

The algorithms that will be presented in the thesis work with convex polygons and polyhedra, that have to be stored somehow in the memory of a computer. In order to allow fast access to relevant information in the description of these objects, special data structures have been designed. We present in the following some dedicated data structures for storing polygons and polyhedra. We also present some special data structures meant to be used in the sweep methods.

Data Structures for Polygons and Polyhedra

From an informational point of view, a convex polygon is a very nice object. It is known that the vertices occur in a certain order, and it is also known that, at each vertex, the internal angle has the same sign (or is less than 2π). Therefore, for storing a convex polygon, it would be enough to put the geometric information associated to each vertex (i.e., the coordinates) into a linear structure, letting the order of data reflect the order in which the vertices occur. A circular queue modeled as an array will do. It is easy to simulate the circularity of vertices – the fact that the first vertex *follows* after the last – by taking the indices of elements in the structure from Z_m, where m is the number of vertices.

The edges are now immediately accessible, as segments determined by two consecutive vertices.

Due to the linearity of the structure, which reflects the geometric linearity, and due to the convexity of the polygon, fast detection of vertices with certain properties is enabled: in $O(\log m)$ time one can get the leftmost, rightmost, upper or lower vertex; all these are done by binary searches. It should also be mentioned that the k'th vertex of the image of the polygon under an isometry is obtained in constant time – if the formula for the isometry is known.

The situation changes dramatically for polyhedra. In their case, there is no intrinsic order among the elements. This time, the data structure must keep – and provide – information related to *incidence* and *adjacency* properties.

The facial structure of convex polyhedra is very similar to the one of *planar graphs*: The image of a convex polyhedron under a stereographic projection *is* a planar graph. This is the reason why, for storing convex polyhedra, a structure dedicated to represent planar maps is used: *doubly-*

connected edge list. The first description of this data structure appears in [MullerPreparata78].

A planar map consists of a collection of polygonal non-intersecting faces. It can be seen as the embedding in the plane of a planar graph, with edges represented as straight-line segments. For effectively handling a structure that stores a planar map, information related to *points, edges, faces and adjacency/incidence relations among them* has to be stored. The doubly-connected edge list (shortly DCEL) fulfills these requirements.

The structure comprises an array of entries that record information associated to edges. For each edge, the following information must be kept:

- the coordinates of the vertices;
- the names of the adjacent faces – to the left respectively to the right of the edge;
- two pointers, one to the first edge encountered on the left face, and one to the next edge on the right face, when one proceed counterclockwise around the first respectively second vertex.

Additionally, arrays of headers for vertices and faces should be created, in which pointers to edges adjacent respectively incident are stored. Having this information, in linear time one can find:

- the set of edges on the contour of a face, in clockwise order;
- the set of edges incident to a vertex, in counterclockwise order.

This structure can be used verbatim for storing convex polyhedra. Indeed, each face is a convex polygon – it has thus the structure of a face of a planar graph – and an order can be naturally defined among the edges incident to a vertex.

Data Structures for Space-Sweep Techniques

Queues of Events. Especially necessary for space-sweep techniques, the *queues of events* record points where the intersection of the sweeping subspace with the universe changes.

We shortly recall the idea of space sweep: There is a d-dimensional universe populated with geometrical objects. A certain property of this universe is sought. For finding it, a subspace of dimension $d-1$ is moved along a straight line. During this sweep, the subspace intersects some of the objects. In certain positions, the configuration of the intersection changes. These particular positions are the so-called *event points*.

As an example, we refer to the first problem mentioned in Section 2.1: reporting intersections in a set of line segments. Here the event points are endpoints of segments and intersection points.

At the beginning, only the endpoints are known: the queue of events will contain them. As soon as intersection points are detected, they are inserted in the queue. Therefore, this structure should be *dynamic*, it should allow fast retrieval/elimination of the *first*, i.e. the *smallest*, element, and fast insertion of a new element, into its appropriate place. The data structure suited for such operations is *priority queue*. The underlying order is the order on the x coordinate.

The elements of the queue record some information specific to the algorithm, usually pointers to objects to which the points that generated the event belong.

Segment Trees. Originally introduced by J. L. Bentley ([Bentley77]), the segment tree is a data structure meant to maintain intervals on the line, in a *dynamic* fashion. The extremes of the intervals are known from the beginning, and are not subject to changes. These abscissae are stored in a *static* structure, designed as a rooted, ordered binary tree.

The abscissae of the interval endpoints will be ordered; we can work with the indices in their ordered array, instead of the actual values. We exemplify a segment tree that stores intervals that have as endpoints the integer numbers in $[1, N]$, with $N \geq 1$.

Each node v of the tree records the following information:

- the endpoints $B[v], E[v]$ of the interval associated to the node;
- pointers to the left and right descendants, LSON and RSON;
- some problem-specific data allocated to the interval.

(We have used here the terminology from Preparata & Shamos book [PreparataShamos85], pp. 13-15.)

The interval associated to the root is $[1, N]$; for a node v, the interval $[B[v], E[v]]$ is split to the descendants: the interval of $LSON[v]$ is $[B[v], (B[v]+$

$E[v])/2]$, and the interval of $RSON[v]$ is $[(B[v]+E[v])/2, E[v]]$. The leaves of the tree store intervals of the form $[k, k+1]$ – these are called *elementary intervals*. The intervals associated to nodes that are not leaves in this three are called *standard intervals*.

It is easy to see that the segment tree of root $[1, N]$ will have height $\lceil \log_2(N) \rceil$, and is balanced. Any interval $[b, e] \subseteq [1, N]$ will be partitioned into at most $\lceil \log_2(b-e) \rceil + \lfloor \log_2(b-e) \rfloor$ standard intervals. This property is remarkable, because when an interval has to be inserted into or removed from the tree, the information associated only to $O(\log N)$ nodes must be modified.

"Insertion" and "removal" have somehow a different meaning, since all the endpoints are already contained in the data structure. Practically, at a certain moment in the evolution of the algorithm, the segment tree stores the interval (which might not be connected) that has a given property – for instance, during the sweep of a set of rectangles, the intersections of the sweeping line with the rectangles are recorded into such a structure. The advantage comes from the fact that, although a linear number of elementary intervals may compose such an intersection, only a logarithmic number of standard intervals will have the data field updated; this brings a considerable speed-up.

4.1.3 Trees of Chains of Edges

For situations in which the possible intersections between the sweeping line and the objects are *not* known from the beginning, a dynamic structure will be used. Usually designed as a balanced ordered tree, this structure allows fast insertion/removal and access to neighbors of an element.

In one of the algorithms that will be presented here, the objects which populate the plane are *chains of edges* defined on the boundary of the polygon P (translated in different positions) by the leftmost and rightmost vertices. Every such chain is x-monotone (every vertical line intersects it in at most one point). For a given abscissa x_0, the vertical order among the chains of edges is well defined - except when they intersect. A pictorial description of such a situation in presented in Fig. 4.1.

For the purposes of the algorithm, the structure will be rather simple. All the x-monotone chains are copies of the upper and lower chains of edges of the same convex polygon, translated with along certain vectors. It would then be enough to store only the vectors of translation, instead of the information

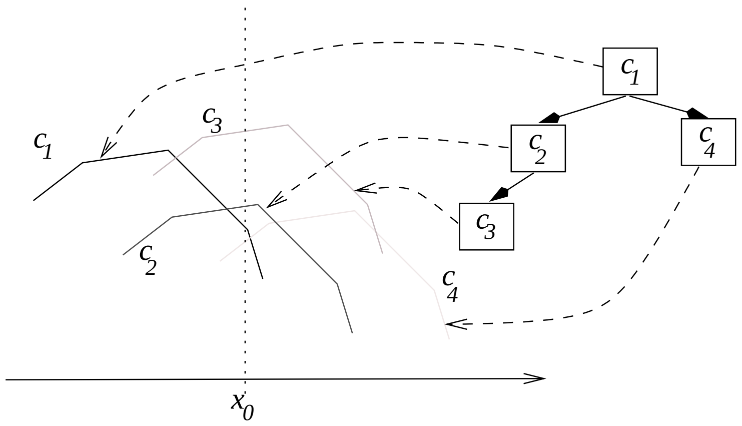

Figure 4.1: Chains of edges and the corresponding tree.

associated to the whole chains. Access to any member of a chain – be it a vertex or an edge – is guaranteed in constant time: constant time is required to gain access to a member of the array of vertices that stores the chain, and the translated of that vertex is computed in constant time, too.

4.2 Algorithms

Algorithms for sorting and searching are nowadays the ABC of any course in data processing. We only say here that, for sorting a n-elements array, $O(n \log n)$ operations are necessary, while for accessing an element in an n-sized sorted structure, $O(\log n)$ operations are necessary.

The following sections contain descriptions of the line-sweep technique and of some algorithms for intersection. The line-sweep will be used in one of the planar algorithms, and intersection of convex shapes is a basic ingredient in the main algorithms.

4.2.1 Intersections of Convex Shapes: Planar Case

Computing intersections of two convex polygons has been proven to require linear time with respect to their numbers of vertices. We present here a method for computing the intersection of two copies of a convex polygon. This method works in logarithmic time with respect to the number of vertices, thus it is much faster than the general method.

For the simplicity of the presentation, consider that the two copies are disposed in plane like in Fig. 4.2:

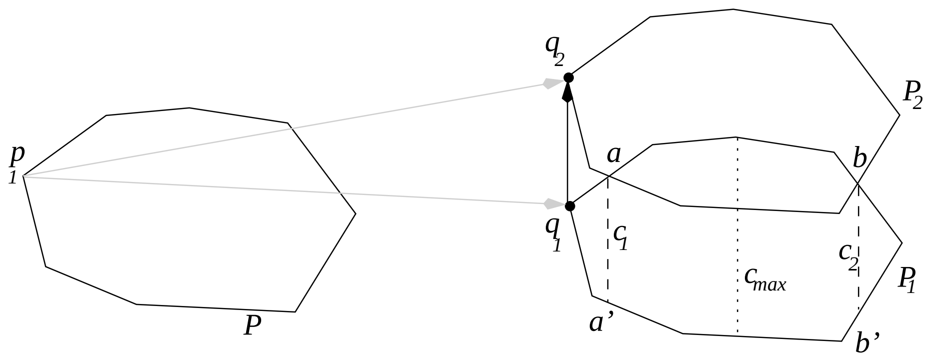

Figure 4.2: Intersection of two copies of a convex polygon.

The two polygons P_1 and P_2 have been obtained by translating P so that its vertex p_1 is put on q_1 and respectively q_2. We assume that q_1 and q_2 have the same abscissa, without losing generality, and the y-coordinate of q_1 is smaller than the y-coordinate of q_2.

The boundary of each polygon can be partitioned into two chains of edges, monotone with respect to the x axis: these chains are delimited by the leftmost and respectively rightmost vertices (which are easy to compute, since they have the minimum and maximum abscissae). If P_1 and P_2 intersect, the upper chain of P_1 and the lower chain of P_2 intersect.

On the other hand, the lower chain of P_2 can be obtained by translating the lower chain of P_1 with the vector $q_2 - q_1$. If a is an intersection point of P_1 and P_2, then there exists a' on the lower chain of P_1 so that $a = \tau_{q_2-q_1}(a')$, or, in other words, $a = a' + (q_2 - q_1)$. Notice that the segment $a'a$ is a vertical chord in P_1, of length $|q_2 - q_1|$.

If the vector $q_2 - q_1$ is long enough, P_2 and P_1 do not intersect: in order that P_1 and P_2 intersect, $|q_2 - q_1|$ must not be greater than the length of the longest vertical chord.

In the case of a proper intersection, the two intersection points lie on one side and the other of the longest chord. Clearly, there exists the situation when the longest chord has its endpoints on parallel edges of P; this is a degeneracy case – the segment q_1q_2 is parallel to an edge of P; we do not consider this degeneracy here, as we have stated, from the beginning, the general position hypothesis.

The two intersection points are, in fact, endpoints of vertical chords of equal lengths, namely, of length $|q_2 - q_1|$. If we can find the abscissae of these chords, we easily get the intersection points. We have to solve the following problem:

Problem 4.2.1. *Find the two vertical chords of length l in convex polygon P.*

A solution of this problem has been given by Kirkpatrick and Snoeyink [KirkSnoeyink95] using tentative prune-and-search techniques for computing fixed-points. They show that the vertical chords of given length can be computed in $O(\log m)$ time, for a m-vertex polygon. We present in the following their method.

Finding the Longest Vertical Chord

First of all, the longest vertical chord has to be computed. Once the longest chord is found, the two vertical chords of given length are sought on each side of the longest chord.

For the longest vertical chord, the following result gives a useful characterization:

Theorem 4.2.1. *Let c be a vertical chord in P of endpoints A, B (A under B). There are parallel tangents to A and B iff c is the longest vertical chord.*

Proof. Assume first that c is not the longest chord. In order to show that there are no parallel tangents to P through A and B, we split the boundary of P into two horizontally monotone chains U and L. U is in turn split into two vertically monotone sub-chains, U_i (upper-increasing) and U_d (upper-decreasing), and L is split into L_d and L_i (see Fig. 4.3).

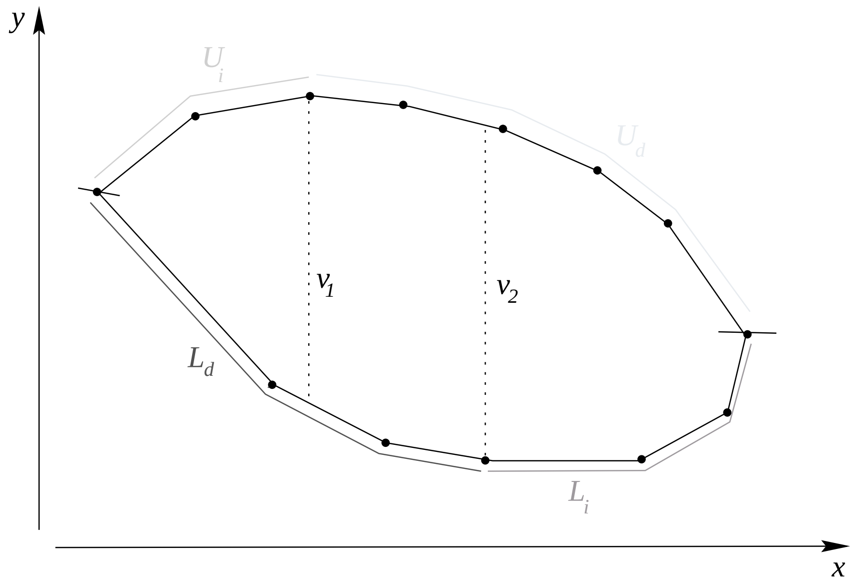

Figure 4.3: Monotone chains of edges on the boundary of a convex polygon.

Without loss of generality, we can assume that the rightmost point on U_i is to the left of the rightmost point on L_d. Let now $c = AB$ be a chord that is not the longest.

1. *Case B on U_i:* In this case, any tangent to A has a negative slope (L_d can be seen as the graph of a monotonously decreasing continuous function), and any tangent to B has a positive slope. Note that this is true even when A or B are vertices: in this case, there exist a pencil of tangents, but this pencil is bounded by the incident edges (see Fig. 4.4); these edges have negative slopes on the lower chain, and positive slopes, on the upper chain.

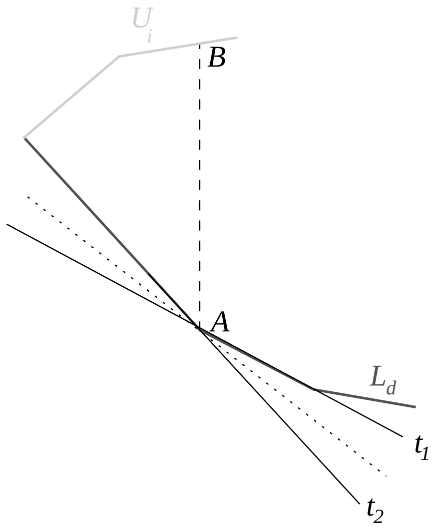

Figure 4.4: Every line between t_1 and t_2 is tangent to A with negative slope.

Remark that, if the longest chord has the endpoint B on U_i, it must be the rightmost point of U_i.

2. *Case A on L_i:* This case is similar to the first one, except that the slopes of tangents to A are positive, and the slopes of tangents to B are negative (we have assumed that the rightmost point on U_i is to the left of the rightmost point on L_d). Also remark that, if the longest chord has the endpoint A on L_i, it must be the leftmost point of L_i.

3. *Case A on L_d and B on U_d:* Since the upper chain U is a concave function (the slope of tangents decreases), and the lower chain L is a convex function, the length of the chord, which can be computed as the difference between U and L seen as function of the abscissa, is again a concave function. It will have then a unique maximum, either at the endpoints of the chains or somewhere else.

   We will treat here the case when the lower endpoint A_l of the longest chord c_l lie on L_d and its upper endpoint B_l lie on U_d. Let c be a chord to the left of c_l. Here, the length of the chord is an increasing function of abscissa, therefore the speed of growth, which is precisely the difference of the slopes of the tangents at the endpoints, is different from 0 (for endpoints that are identical to vertices of P, we apply the same argument as before: the tangents can be chosen in a pencil bounded by the incident edges). This means that c cannot have tangents with the same slope, i.e., parallel.

The same argument can be used when c lie to the right of the longest chord, except that here the slopes of the tangents decrease unequally.

We have shown so far that there cannot be parallel tangents to the endpoints of a chord that is not the longest. Let now c_l be the longest chord; we will show that c_l admits parallel tangents to its endpoints.

There cannot be that no endpoint of c_l is a vertex of P: if e_1 and e_2 are the edges of P on which the endpoints of c_l lie, from the assumption of general position, e_1 and e_2 are not parallel, and thus the longest chord with endpoints on them must touch a vertex.

We have now two cases, depicted in Fig. 4.5.

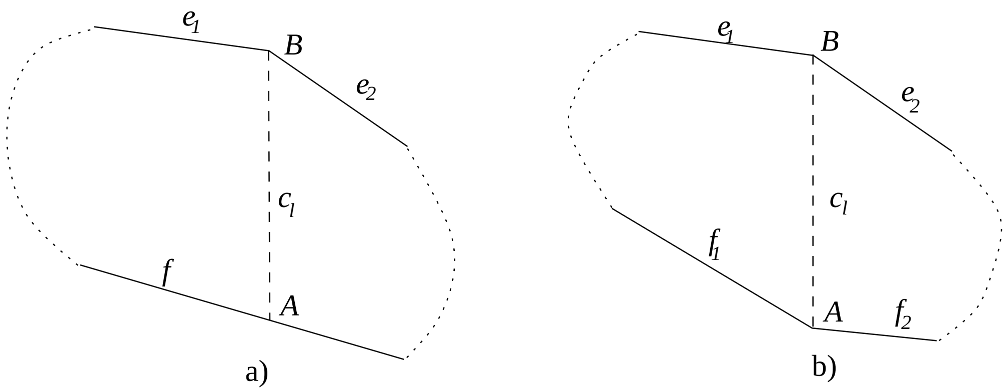

Figure 4.5: The longest chord can have one endpoint at a vertex (a) or both endpoints (b).

In the first case, c_l has one endpoint, say B, at a vertex. We show that there exist a tangent to B parallel to the edge f that contains the other endpoint, A.

Any straight line that contains B and has a slope between the slopes of e_1 and e_2 is a tangent for P. On the other hand, the slope of e_1 is greater than the slope of f (because B is vertically more distant to f than any other point on e_1), and the slope of e_1 is smaller than the slope of f. This means that there exists a tangent to P parallel to f in the pencil delimited by e_1 and e_2.

For case (b), let I_1 be the interval of slopes of tangents to A, and I_2 the interval of slopes of tangents to B. We show that $I_1 \cap I_2 \neq \emptyset$.

Let f_1, f_2 be the edges incident to A, and e_1, e_2 the edges incident to B, as in figure. From the convexity of the lower chain and concavity of the upper chain of edges, we have:

$$\begin{aligned} \text{slope}(e_1) &> \text{slope}(e_2) \\ \text{slope}(f_1) &< \text{slope}(f_2) \end{aligned}$$

The chord c_l is the longest vertical chord, therefore the edges e_1 and f_1 intersect to the left of c_l; it follows that

$$\text{slope}(e_1) > \text{slope}(f_1).$$

Using a similar argument,

$$\text{slope}(e_2) < \text{slope}(f_2).$$

Now, $I_1 = [\text{slope}(f_1), \text{slope}(f_2)]$ and $I_2 = [\text{slope}(e_2), \text{slope}(e_1)]$, and, since $\text{slope}(e_2) < \text{slope}(f_2)$ and $\text{slope}(f_1) < \text{slope}(e_1)$, the two intervals have nonempty intersection. ■

This theorem gives a method for finding the longest vertical chord: we have to find the vertical chord that admits parallel tangents to its endpoints. For this, we will define two piecewise-linear functions so that the fixed point of their decomposition gives the abscissa of the longest chord. Finding a fixed point of two such functions can be done in logarithmic time with respect to the number of their linear components.

In the following, we see the chains U and L as functions of their abscissae: in this way, we introduce a parameterization of these chains. Without loss of generality, we can assume the parameter range over $[0, 1]$:

$$U : [0, 1] \to \mathbf{R},$$

$$L : [0, 1] \to \mathbf{R}.$$

U and L are piecewise-linear, and the total number of their linear components is m. For the simplicity of notation, we will use the same letters, U and L,

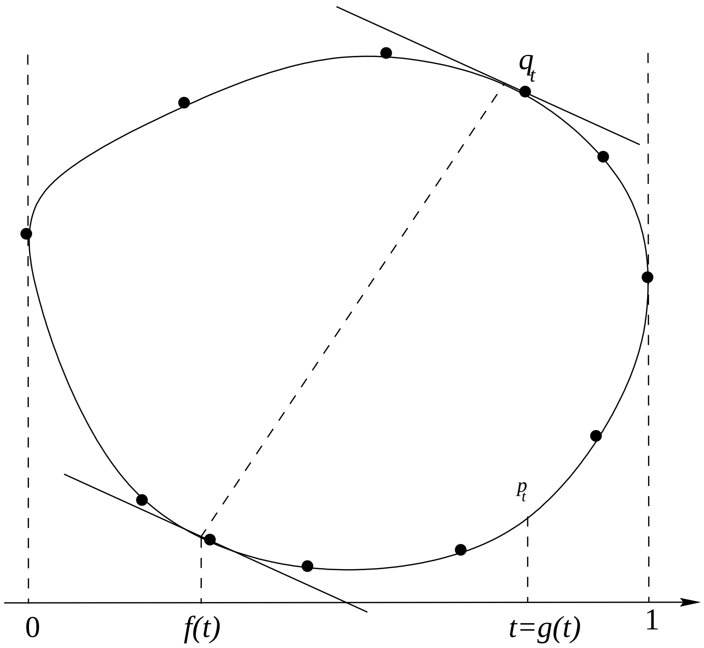

Figure 4.6: Defining f and g.

for the graphs of these functions. We associate to L and U two functions, $g : [0,1] \to \mathbf{R}$, $f : [0,1] \to \mathbf{R}$, defined as follows: g is the identity, i.e., for a point p in L of parameter t, $g(t) := t$ is the parameter of the upper endpoint of the vertical chord of abscissa t. For a point $q \in U$ of parameter t, $f(t)$ is the parameter of the point q' in L with the same tangent as q (see Fig. 4.6 for an intuitive view). The function f is not well defined when q or q' is a vertex: there are many tangents to such points. This problem can be avoided by "smoothing" the polygon: we approximate P by a family of smooth splines indexed on the degree of approximation accuracy, and we take P as the limit of this family.

Now, f is decreasing and g is increasing on $[0,1]$; moreover, $f([0,1]) = [0,1], g([0,1]) = [0,1]$. For such functions we have the following lemmata:

Lemmata 4.2.1. *Let f, g be two piecewise-linear continuous functions, $f : A \to A$ monotone decreasing and $g : B \to B$ monotone increasing, where A and B are intervals on the real axis. If $B \subseteq f(A)$ and $A \subseteq g(B)$ then $f \circ g$ and $g \circ f$ have fixed points in the corresponding domains.*

Proof. Clearly, $f \circ g : B \to f(A) \supseteq B$ is monotone decreasing. Moreover, $B \subseteq f(A) \subseteq f(g(B))$, thus $f \circ g$ is onto.

Let $h := f \circ g - 1_B$, $h(x) = f(g(x)) - x$; h is monotone, because $f \circ g$ is monotone; h has different signs at the ends of B, because $f \circ g$ is onto; h is continuous, because $f \circ g$ and 1_B are continuous. From the Mean Value Theorem, h has a root on B, which means there exists x in B so that $f(g(x)) = x$. This is a fixed point for $f \circ g$.

The proof that $g \circ f$ has a fixed point is identical. ∎

Let $t_0 \in [0, 1]$ be such that $(g \circ f)(t_0) = t_0$. The points on L and U of parameter t_0 have the same abscissa, i.e., they belong to the same vertical chord. Since g is the identity, $g(f(t_0)) = t_0$ iff $f(t_0) = t_0$.This means that there are two parallel tangents to points on U and L of abscissa t_0; in other words, the vertical chord of abscissa t_0 admits parallel tangents. By Theorem 4.2.1, this vertical chord is the longest.

Thus, by finding a fixed point of the composition of f and g, we find a longest vertical chord. We will show now that such a fixed point can be found in $O(\log m)$ time.

The method resembles a binary search. At each step k, a sub-interval J_k of $[0, 1]$ is kept, with $g \circ f$ having a fixed point in J_k.

Let the abscissae of the vertices in L be $A := \{a_1, \ldots, a_l\}$ and in U be $B := \{b_1, \ldots, b_u\}$, with $l + u = m - 2$ (the initial and final vertices coincide); recall that $a_1 = b_1 = 0$, $a_l = b_u = 1$. There is an interval $[b_k, b_{k+1}]$ that contains a fixed point of the composition $g \circ f$. Let $a_{mid} := a_{\lfloor \frac{l+1}{2} \rfloor}$, $b_{mid} := b_{\lfloor \frac{u+1}{2} \rfloor}$. In constant time we can compare $f(b_{mid})$ and a_{mid}, $g(a_{mid})$ and b_{mid}. There can be one of the following four situations (see Fig, 4.7):

Case (a): $f(b_{mid}) \geq a_{mid}$ and $g(a_{mid}) \geq b_{mid}$: The interval $[b_1, b_{mid})$ is discarded, because, if $b \in [b_1, b_{mid})$, then $f(b) > a_{mid}$, and $g(f(b)) > g(a_{mid}) \geq b_{mid}$, thus, no fixed point can be here.

Case (b): $f(b_{mid}) \geq a_{mid}$ and $g(a_{mid}) \leq b_{mid}$: The interval $[a_1, a_{mid})$ is discarded, because $f(b) \in [a_1, a_{mid})$ only if $b > b_{mid}$, and $g(f(b))$ will then be smaller than b_{mid}.

Similarly, in the other two cases, halves of the number of intervals of either a's or $b's$ are discarded. The process continues until two intervals, $[a_i, a_{i+1}]$ and $[b_j, b_{j+1}]$ are found, so that $g \circ f$ has a fixed point in $[b_j, b_{j+1}]$. This fixed

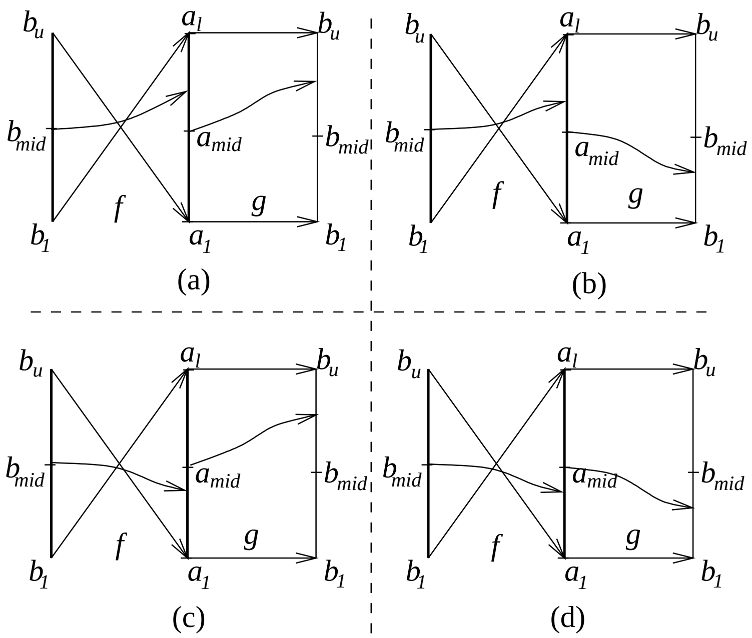

Figure 4.7: Discarding halves of a-intervals and b-intervals.

point can be found in constant time, as the solution of the (linear) equation $g(f(x)) = x$.

At each step, the number of elements in either A or B halves, hence the total number of steps is $O(\log|A| + \log|B|)$; since $|A| + |B| = m - 2$, a simple computation reveals that there are $O(\log m)$ steps.

Finding Vertical Chords of Given Length

In order to find vertical chords of given length l, one has to make sure that l is smaller than the length of the longest vertical chord. If this condition is fulfilled, the search continues on each side of the longest chord. Clearly, there are cases when the longest vertical chord is an edge of the polygon; in such a situation, there will be only one vertical chord of length l.

Theorem 4.2.2. (Chord of given length) *The at most two vertical chords of given length l in a convex polygon P with m vertices can be found in $O(\log m)$ computing time.*

Proof. In $O(\log m)$ time, the longest chord C_l can be computed. If the length l is greater than the length of C_l, then there is no chord of P of length l.

If l is equal to the length of C_l, the computation stops, and C_l is reported.

When l is smaller than the length of C_l, the two vertical chords of length l should be sought to the left and right of C_l – if C_l is not an edge of P itself, in which case there is only one chord of length l.

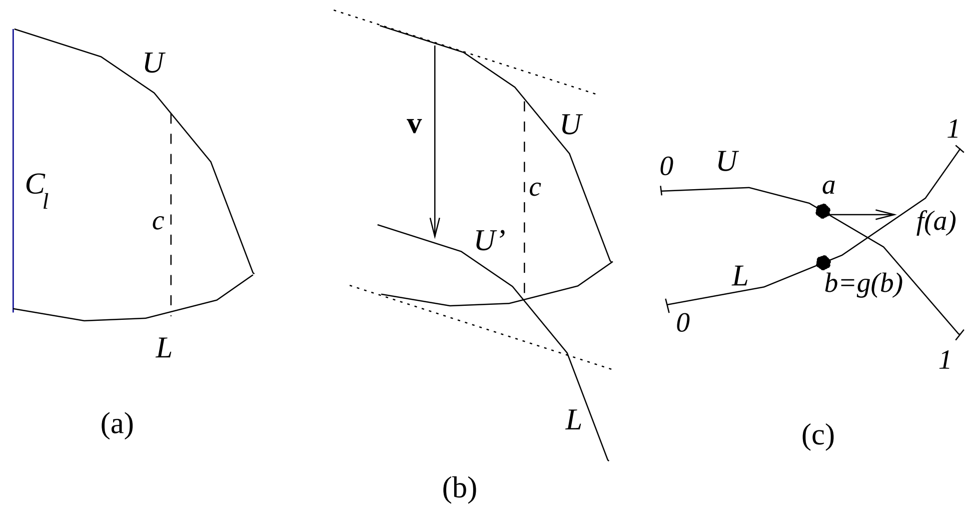

Figure 4.8: Two x-monotone chains of edges on ∂P to the right of the longest chord (colored blue).

We study in the following the search of a chord of length l – let us call it c – to the right of C_l (see Fig. 4.8, (a)). The boundary of P to the right of C_l can be split into two x-monotone chains, U (upper) and L (lower).

The lower endpoint of c is the intersection of L with $U' := \tau_{\mathbf{v}}(U)$, where $\mathbf{v}$ is the vector of coordinates $(0, -l)$ (see Fig. 4.8, (b)). We can focus to fast detection of this intersection point.

The slope of any tangent to U' is dominated by the slope of the tangent to the upper endpoint of C_l (drawn with dotted line in Fig. 4.8, (b)); accordingly, the slope of any tangent to L dominates the slope of the tangent to the lower endpoint of C_l. Due to this fact, U' and L can be subjected to a skew transformation, that does not change the x coordinate of their intersection point, but makes them both x and y-monotone, as in Fig. 4.8, (c).

We have reduced thus the initial problem to finding the intersection point of two x and y monotone chains, U and L; both are considered parameterized by x-coordinate, with x ranging over $[0, 1]$. We define two functions, f, g :

$[0,1] \to [0,1]$, so that a fixed point of their composition gives the abscissa of $L \cap U$.

The function g will be the identity, and, for $x \in [0,1]$, $f(x)$ will be the abscissa of the point on L having the same y-coordinate as $U(x)$. Clearly, $g \circ f(x_0) = x_0$ if and only if $U(x_0) = L(x_0)$, i.e., x_0 is the parameter of the intersection point.

Now, since:

- g is piecewise-linear and monotone-increasing,
- f is piecewise-linear and monotone decreasing,
- the cumulate number of linear pieces of f and g is dominated by m,

the fixed point of $g \circ f$ can be found in $O(\log m)$ time, as we have shown in the previous paragraph.

■

4.2.2 Intersection of Convex Shapes: Spatial case

The fastest algorithm for the intersection of two three-dimensional polyhedra has been given by B. Chazelle ([Chazelle92]). This algorithm uses a very intelligent improvement of the hierarchical representation of a convex polyhedron of Dobkin and Kirkpatrick ([DobkinKirkpatrick83]). We insert here only an informal description of this algorithm, for it is much too complex to be exposed in details, and much too complicated for the particular case that we have to deal with.

A convex polyhedron P with m vertices is the first in a descending chain of polyhedra in which the last has a constant number of vertices, and two consecutive polyhedra differ by peeling off small polyhedral cones: this is the idea behind the hierarchical representation of Dobkin and Kirkpatrick. Chazelle pushed this idea to the extreme by:

- discarding all but a constant number of polyhedra from the hierarchy;
- designing a similar incomplete hierarchy for the exterior of the polygon; more precisely, since the exterior of the polygon is non-convex, he switched to the dual space, where the set of planes that avoid P are mapped to a convex polyhedron.

First, if the two polyhedra intersect, a witness of this intersection is generated using Dobkin and Kirkpatrick algorithm for detecting intersections ([DobkinKirkpatrick83]). The time used for this initial step is $O(m)$.

Then the intersection of boundaries is computed, by a walk along edges, consecutively on the boundary of P and Q. Beware that in the general case the boundaries may intersect along many disjoint cycles of edges. However, once that a point on such a chain is found, the whole connected component can be determined in linear time. The total number of edges on all these cycles cannot be larger than the number of edges of the intersection polyhedron, which is linear in the number of vertices of the two polyhedra.

Chazelle's algorithm finds first a vertex on a connected component of the intersection, and then generates in turn every other cycle of edges. The delicate phases of his method – finding a vertex on the intersection, and finding the next cycle of edges on the intersection – use the ingenious data structures to perform in linear time the first task, and also in total linear time the detection of all cycles.

We have to mention here that in our algorithm `Max-3dcontainment-by-Anchored-Sweep` (page 89) the intersection of two translated copies of the same polyhedron has to be detected – to be specific, the intersection of boundaries. This simplifies a lot the assumptions of the problem, because (as will be shown below) there is only one cycle of straight-line segments in the intersection of boundaries. Therefore, the first problem is to find a point on this cycle, and the second problem, to generate the edges of the intersection, using the point detected before as starting point on a walk on the boundaries of the two polyhedra.

However fast the modified algorithm can be, it cannot go under $O(m)$. This is due to the fact that there can be $O(m)$ edges in the boundary intersection – see Fig. 4.9 for an example. Examples of simplicial polyhedra with the intersection of boundaries of the same magnitude can be easily imagined.

In all what follows, we deal with a convex polyhedron P with m vertices and two points in space, q_1 and q_2. The vector $q_2 - q_1$ will be denoted by $\mathbf{v}$.

First of all, the following Lemma states the properties of the intersection of boundaries:

Lemma 4.2.1. (Intersection of Translated Polyhedra) *Let P be a three-dimensional convex polyhedron with vertices $\{p_1, \ldots, p_m\}$ in general position, and let q_1, q_2 be two points such that no k-face of P is parallel to q_1q_2*

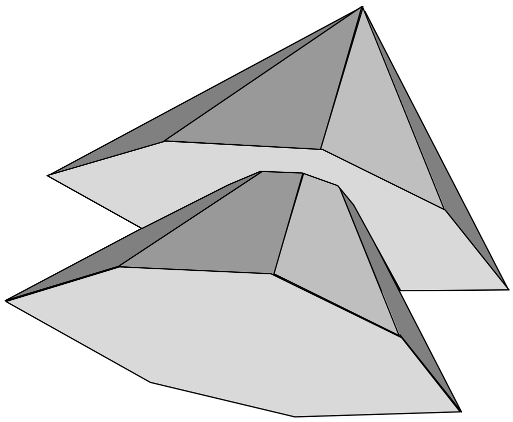

Figure 4.9: The surfaces of two identical pyramids with m-vertex bases can intersect along an m-vertex polygonal line.

$(k = 1, 2)$. Then

$$\gamma = \tau_{q_1-p_1}(P) \cap \tau_{q_2-p_1}(P)$$

is a closed polygonal line in space whose projection on a plane orthogonal to $q_2 - q_1$ is a convex polygon.

Proof. If $\tau_{q_1-p_1}(P)$ and $\tau_{q_2-p_1}(P)$ have non-empty intersection, the intersection will be a convex polyhedron; we denote this intersection by $P_{1,2}$.

Let π be a plane orthogonal to $q_2 - q_1$. The convex hull of $\mathrm{proj}_\pi(P_{1,2})$ is a convex polygon $H_\cap$; due to the convexity of $P_{1,2}$, each edge of $H_\cap$ is in fact the projection of an edge of $P_{1,2}$. The edges of $P_{1,2}$ that project on $H_\cap$ form a closed polygonal line: they split the boundary of P_{q_1,q_2} in two pieces, one convex-downward – we denote it C_{top}, and the other convex-upward – denoted C_{bottom}.

We shall prove that $C_{top} \cap C_{bottom} = \gamma_{1,2}$.

In the following, we consider, without loss of generality, that the vector $q_2 - q_1$ is vertical and points upwards: $\tau_{q_1-p_1}(P)$ is under $\tau_{q_2-p_1}(P)$ (see Fig.4.10). We denote by H_P the convex hull of the projection on π of any of these polyhedra. The edges of $\tau_{q_1-p_1}(P)$ that project onto edges in H_P form a closed polygonal line; this line splits its boundary into two pieces, one convex-downward (we call it $P_{1,top}$) and one convex-upward ($P_{1,bottom}$). Similarly, the boundary of $\tau_{q_2-p_1}(P)$ is split into two pieces, $P_{2,top}$ and $P_{2,bottom}$.

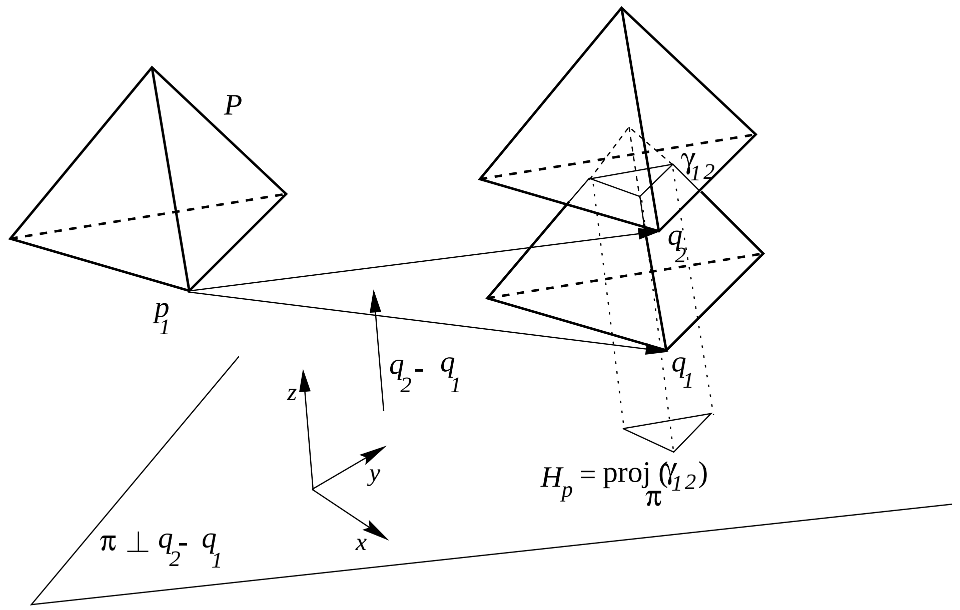

Figure 4.10: The intersection of the boundaries of two translates of P.

Now, $P_{2,top} = \tau_{q_2-q_1}(P_{1,top})$ and $P_{2,bottom} = \tau_{q_2-q_1}(P_{1,bottom})$; this implies that $P_{1,top} \cap P_{2,top} = \emptyset = P_{1,bottom} \cap P_{2,bottom}$, and thus only $P_{1,top}$ and $P_{2,bottom}$ contribute to the intersection. Moreover, since C_{top} is convex-downward, it cannot contain portions from $P_{2,bottom}$, which is convex-upward; nor can C_{bottom} contain portions from $P_{2,top}$.

This means that $C_{top} \cap C_{bottom} = P_{1,top} \cap P_{2,bottom} = \tau_{q_1}(P) \cap \tau_{q_2}(P) = \gamma_{1,2}$; since C_{top} and C_{bottom} were determined by splitting the surface of $P_{1,2}$ after a chain of edges that project onto $H_\cap$, it follows that $H_\cap = \text{proj}_\pi \gamma_{1,2}$. ■

The problem of computing the intersection of two copies of a convex polyhedron is reduced, by the preceding Lemma, to the computation of the intersection of two convex surfaces. For this,

- an initial point on the polygonal chain $\gamma_{1,2}$ must be found,
- then the whole cycle of edges in $\gamma_{1,2}$ must be constructed.

Finding an Initial Point on $\gamma_{1,2}$. This phase in the polyhedral intersection detection can be solved with the elegant algorithm of separation of

Dobkin and Kirkpatrick ([DobkinKirkpatrick85]). At the core of this method stands a *hierarchical* description of the polyhedron, consisting in $O(\log m)$ polyhedra that gives a progressively better approximation. The total amount of memory for storing this array of polyhedra is $O(m)$.

The hierarchical representation of a polyhedron P with m vertices is a sequence of polyhedra $H(P) = P_1, \ldots, P_k$ with the following properties:

1. $P_1 = P$ and P_k is a simplex – a tetrahedron;
2. $P_{i+1} \subset P_i$, for $1 \leq i < k$;
3. all vertices of P_{i+1} are also vertices of P_i;
4. the vertices in P_i that are not in P_{i+1} are non-adjacent in P_i (they are not connected with edges).

The size of $H(P)$ is the sum of the numbers of vertices, edges and faces of all polyhedra $P_1, \ldots, P_k$; k will be called the height of $H(P)$.

Starting from the DCEL (structure described in Section 4.1) representation of P, in $O(m)$ time a hierarchical representation of P can be constructed:

1. $P_1 = P$; $i = 1$;
2. while P_i is not a simplex do
   (a) $S_i :=$ any maximal subset of non-adjacent vertices of P_i, each having at most b incident edges;
   (b) $P_{i+1} :=$ the convex hull of the remaining vertices of P_i, after eliminating S_i;
   (c) $i := i + 1$

If b is sufficiently large, the maximal subset S_i of vertices with degree (i.e., number of incident edges) smaller than or equal to b will be within at least a constant ratio with the number of vertices of P_i (for instance, from Euler's formula, for $b \leq 11$, $m/\#(S_1) \geq 24$). This means that the numbers of vertices of polyhedra in $H(P)$ decreases, following a law of the form $|P_{i+1}| < c|P_i|$; here by $|P_i|$ we have denoted the number of vertices of P_i.

This is the reason why k, the total number of polyhedra in $H(P)$, is $O(\log m)$, and also, why the total number of vertices of $H(P)$ in $O(m)$. It remains only to show how P_{i+1} can be constructed from P_i in $O(|S_i|)$ time.

The construction of P_{i+1} from P_i is done sequentially, by eliminating one vertex in S_i at the time. By this removal, a constant number of edges and faces must be eliminated from the DCEL representation, and a constant number of edges and faces must be added. With a careful handling of the components of a DCEL, these operations can be done in constant time.

The hierarchical representation of P has some intrinsic geometric properties, from which the most important is the following: *if π is a plane such that $P_{i+1} \subset \pi^+$, then either $P_i \subset \pi$, or there is a unique vertex v of P_i such that $v \in \pi^-$.* As a consequence, if π is any hyperplane that supports P_{i+1}, then either it supports P_i, or $\pi^- \cap P_i$ is a pyramid whose apex has at most b incident edges.

We are now in position to give the algorithm for retrieving a common point of ∂P and $\partial\tau_{\mathbf{v}}(P)$, if they have nonempty intersection. The algorithm is presented in Figure 4.11. Informally, the closest points in P_i and $\tau_{\mathbf{v}}(P_i)$ are computed, for $i = k, k-1, \ldots, 1$; when these points happen to be identical, the algorithm stops.

Input a DCEL representation of a polyhedron P with m vertices;
a vertical vertex $\mathbf{v}$;
Output a point in $\partial P \cap \partial\tau_{\mathbf{v}}(P)$, if the polyhedra intersect;
$\emptyset$, if the polyhedra do not intersect;
begin
1. compute the hierarchical representation $H(P) = (P_1 = P, \ldots, P_k)$;
2. compute the closest pair of points (p_k, q_k)
 with $p_k \in \partial P_k$, $q_k \in \partial\tau_{\mathbf{v}}(P_k)$;
3. **if** $p_k = q_k$ **return** p_k;
4. **for** $i = k-1$ **downto** 1
5. compute (p_i, q_i), the closest pair of points,
 with $p_i \in \partial P_i$, $q_i \in \partial\tau_{\mathbf{v}}(P_i)$;
6. **if** $p_i = q_i$ **return** p_i;
 end for
7. **return** $\emptyset$
end

Figure 4.11: Algorithm `Point-In-Polyhedral-Intersection`.

A few words should be said about steps 2 and 5 of the algorithm. In step

2, the closest points of two three-dimensional simplices are computed; this is done in constant time, by simply taking all pairs (vertex, vertex), (vertex, edge), etc., of elements in the two polyhedra, and computing their closest points.

Step 5, however, requires a more in-depth analysis. Let H_P be a hyperplane orthogonal to $p_{i+1}q_{i+1}$ that supports P_{i+1}, H_{vP} a hyperplane orthogonal to $p_{i+1}q_{i+1}$ that supports $\tau_{\mathbf{v}}(P_{i+1})$, and H_P and H_{vP} separate P_{i+1} and $\tau_{\mathbf{v}}(P_{i+1})$. We have mentioned earlier that a unique vertex v_P of P_i lies in H_P^-, and a unique vertex of $\tau_{\mathbf{v}}(P_i)$ lies in H_{vP}^-. Therefore, the closest points of P_i and $\tau_{\mathbf{v}}(P_i)$ can be:

- either p_{i+1} and q_{i+1}, or
- the closest points of $P_i \cap H_P^-$ and $\tau_{\mathbf{v}}(P_i)$, or
- the closest points of $\tau_{\mathbf{v}}(P_i) \cap H_{vP}^-$ and P_i.

The last two possibilities can be explored in $O(|P_i|)$ operations, because the polyhedra $P_i \cap H_P^-$ and $\tau_{\mathbf{v}}(P_i) \cap H_{vP}^-$ have a constant number of elements.

The analysis of the algorithm is quite simple: The preprocessing requires $O(m)$ operations, as well as the total number of operations in the **for** cycle, lines 4-6, yielding a total time of $O(m)$.

So far, we have shown how a common point of P and $\tau_{\mathbf{v}}(P)$ can be obtained. This point can be on their surface intersection, but it can be interior as well.

If the point thus generated – let us call it p_{int} – is not on $\gamma_{1,2}$ (this that can be checked by a $O(m)$ verification of containment in every face of δP and $\delta\tau_{\mathbf{v}}(P)$), we consider any vertical plane π_{vert} that contains p_{int}. The intersection of π_{vert} with the two polyhedra consists in two identical convex polygons, because $\pi_{vert} \cap \tau_{\mathbf{v}}(P) = \tau_{\mathbf{v}}(\pi_{vert} \cap P)$. Since $p_{int} \in \pi_{vert}$, $\pi_{vert} \cap \tau_{\mathbf{v}}(P)$ and $\pi_{vert} \cap P$ have nonempty intersection. Obviously, $\partial(\pi_{vert} \cap \tau_{\mathbf{v}}(P)) \cap \partial(\pi_{vert} \cap P) \in \gamma_{1,2}$.

What we have now is precisely the problem of finding the intersection of two identical convex polygons. We have shown in Section 4.2.1 how this problem can be solved in $O(\log m)$ operations.

It remains only to show how the intersection of π_{vert} with P can be efficiently computed. The plane π_{vert} will be defined as the vertical plane that contains the origin, i.e., the vertex p_1 of P, and p_{int}; any other vertical plane that contains p_{int} would do.

The point p_1 will be a vertex of $\pi_{vert} \cap P$; one of the two edges of $\pi_{vert} \cap P$ incident to p_1 is sought as the intersection of one of the faces of P incident to p_1.

Once this edge is computed, the same process continues with the other endpoint. Figure 4.12 contains the algorithm.

Input. a DCEL representation of a convex polyhedron P
  with m vertices $\{p_1, \ldots, p_m\}$, p_1 at the origin;
 a point $p_{int} \in P$;
Output. a polygon Q defined as the intersection of P
  with the vertical plane that contains p_1 and p_{int};
begin
1. compute the equation of the vertical plane π_{vert}
  that contains the origin and p_{int};
2. $v_1 := p_1$; $k := 1$;
3. find a face f_1 incident to p_1 with $f \cap \pi_{vert} \neq \emptyset$;
4. **repeat**
5. compute e_2, the edge of f_1
   that contains the other endpoint v of $\pi_{vert} \cap f_1$;
6. **if** $v \neq v_1$
7.   **then** $k := k + 1$; $v_k := v$;
8. $f_2 := f1$;
9.   find the face f_1 adjacent to $f2$,
     having e_2 as common edge;
10. $e_1 := e_2$;
11. **end if**
12. **until** $v = v_1$;
13. **return** $\{v_1, \ldots, v_k\}$
end.

Figure 4.12: Algorithm `Intersection-Plane-Polyhedron`.

Since each face of P contributes with at most one edge, the total number of vertices that will be generated is less than or equal to m. For the first vertex, at most $O(m)$ faces are scanned, in line 3. For the rest of v_i's, the face adjacent to the edge of P that contains them is detected in constant time from the DCEL, since each edge of P has pointers to its adjacent faces; thus line 9 contributes with constant time. Line 5 can be done by a mere sweep of the edges of f_1; in the worst case, all edges of P are swept two times, as there are two faces adjacent to each edge. Thus the total time spent for line 5 is $O(m)$.

Summing up, the algorithm makes $O(m)$ operations for line 3, and $O(m)$ operations for the cycle in lines 4-12; all the other operations are done in constant time. Thus the intersection of a m-vertex polyhedron with a plane can be obtained in $O(m)$ operations.

Generating the edges of $\gamma_{1,2}$. In the previous paragraph we have shown how to find a point on $\gamma_{1,2}$. The generation of the whole cycle of edges can be done in a manner similar to the algorithm in Fig. 4.12. The difference, here, is that the role of plane π is taken by faces of $\tau_{\mathbf{v}}(P)$.

It is easy to modify the algorithm `Intersection-Plane-Polyhedron` and the data structures so that each vertex v_i of $P \cap \pi$ has attached a pointer to the edges of P to which it belongs, so that the point p on $\gamma_{1,2}$ comes with a pointer to the faces of P and $\tau_{\mathbf{v}}(P)$ that contain it. This information will constitute the starting point of the walk along faces of P and $\tau_{bfv}(P)$, during which $\gamma_{1,2}$ is generated.

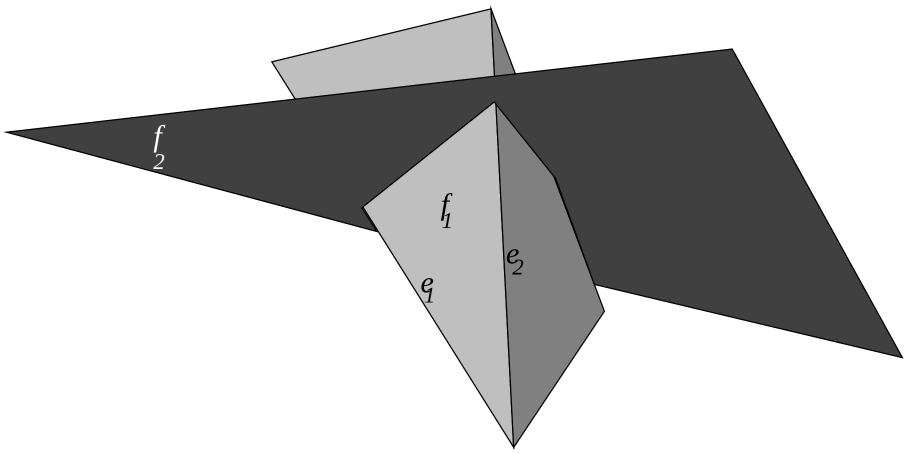

Figure 4.13: Advancing in the algorithm of $\gamma_{1,2}$-detection.

In every moment, there is a current face f_1 in P and a current face f_2 in $\tau_{\mathbf{v}}(P)$ that intersect, and an edge e_1 of one of these faces that is intersected by the other face. The algorithm detects the next edge intersected – either of f_1 or of f_2 – and this will give the next generated vertex of $\gamma_{1,2}$. Assume that this edge, let us call it e_2, belongs to f_1; then f_1 will be replaced by the other face adjacent to e_2, in P – see Figure 4.13.

Figure 4.14 contains the algorithm.

Input. a DCEL representation of a convex polyhedron P
a vector $\mathbf{v}$;
a point $p_{int} \in \partial P \cap \partial \tau_{\mathbf{v}}(P)$,
and the faces $f_1 \in \partial P$ and $f_2 \in \partial \tau_{\mathbf{v}}(P)$ to which it belongs;
Output. a cycle of straight-line segments in space, $\gamma_{1,2}$,
defined as $\partial P \cap \partial \tau_{\mathbf{v}}(P)$
begin
1. compute one endpoint, v, of the segment $f_1 \cap f_2$
2. $v_1 := v$; $k := 1$;
3. **repeat**
4. compute v', the other endpoint of $f_1 \cap f_2$ ($vv' = f_1 \cap f_2$)
5. **if** $v' \neq v_1$
6. **then** $k := k + 1$; $v_k := v'$;
7. **if** v' is on an edge e that belongs to f_1
8. $f_1 :=$ the other face on ∂P incident to e;
9. **else**
10. $f_2 :=$ the other face on $\partial \tau_{\mathbf{v}}(P)$ incident to e;
11. **end if**
12. $v := v'$;
13. **end if**
14. **until** $v = v_1$;
15. **return** $\{v_1, \ldots, v_k\}$
end.

Figure 4.14: Algorithm `Intersection-Polyhedra`.

This algorithm is linear in the number m of vertices of P. Indeed, apart from line 4, the **repeat** cycle in lines 3-14 loops as many times as edges in $\gamma_{1,2}$ are, i.e., $O(m)$ times: since $P \cap \tau_{\mathbf{v}}(P)$ is the intersection of $2m$ half-spaces, the number of its elements is $O(m)$. For lines 1 and 4, in the worst case, when all faces of ∂P and $\partial \tau_{\mathbf{v}}(P)$ are tested, the points v' are computed by walking along edges on the boundaries of faces; these walks go not more that two times over each edge, therefore the total time of line 1 and 4 is again $O(m)$. Thus, in $O(m)$ time the cycle of edges $\gamma_{1,2}$ can be computed.

4.2.3 Plane-Sweep Techniques

In Section 2.1 we have given a history of line-sweep techniques, and a short description. We present in the following, in some detail, the method.

The idea is simple, and is derived from the geometrical nature of the working space: Some geometrical objects (straight-line segments, rectangles, convex polygons etc.) lie in the plane (this is the *scene*). A vertical line sweeps the plane from left to right. In every moment, the intersection of the line with the set of objects (the *status* of the line) is maintained. This status changes at some abscissae (for instance, when the first point of an object is touched, during the sweep). The points where the status changes (the *event-points*) are also stored, usually in a priority queue, as fast retrieval of the leftmost point and fast insertion of new points are required. At the beginning, only information characteristic to each object is inserted in the event queue (for instance, the endpoints of the segment, when the objects are segments); new events are added during the sweep, as soon as they are detected (usually, these are points of intersection between objects).

The general skeleton of this method is depicted in Figure 4.15.

```
collect the initial information from the scene:
    initialize the event queue;
    initialize the line status;
while there are any events in the queue
    extract the leftmost event from the queue;
    process the geometrical information in the scene,
        to the right of this event:
            update the line status;
            update the event queue.
```

Figure 4.15: Sketch of the plane-sweep algorithm.

This method is especially efficient when the line status allows fast insertion and retrieval, i.e., when an order can be defined on every vertical section of the scene. If the scene contains a set of line segments, this order is well defined (except at the intersection points; however, at these points the order changes, and for most applications it is enough to consider this order for the adjacent intervals). If the scene contains convex polygons, some x-monotone chains of edges should be taken as basis objects. When the problem comes from the geometry of rectangles, there are special structures that handle the sweep – e.g., *segment trees.*

In Chapter 5, Section 5.1.4, we make use of a plane sweep of a scene that contains copies of convex polygons. The goal is to find a region of maximum depth in an arrangement of translated copies of a convex polygon. The line status will store more complicated objects – chains of edges.

Anchored Sweep Algorithms

Chapter 5

Translation of Optimal Containment

In this chapter, we describe solutions to instances of Problem 1.2.2:

- translational placement of a convex polygon to cover a maximum number of points from a given set;
- translational placement of a convex polyhedron to contain a maximum number of points from a given set.
- optimal translational placement of a segment on the real line to contain a maximum number of values from a given set;
- optimal translational placement of a rectangle to cover a maximum number of points from a given set;

The first two algorithms make use of the anchored sweep paradigm. Solutions for the first instance have been published in [BarequetDP97]. Pau, in [Pau99b], presents a solution for the three-dimensional case.

The basic idea behind these solutions is:

1. fix a subset of the set of points to be covered – in 1- and 2-dimensional cases, this subset contains one point; in three dimensions, it will contain two points;
2. translate the convex polygon/polyhedron/segment/rectangle with the same vertex to each point of this subset;

3. compute the common intersection of the translated copies – this intersection must be topologically homeomorph to a circle;

4. start a "sweeping movement" of the polytope with one vertex anchored to the intersection mentioned before;

5. keep track of the "events", i.e., of the points that enter/exit the moving polytope; record the position with most points inside.

The last two algorithms are optimal, in the sense that their number of operations is asymptotically equal to the lower bound, proven in Section 3.7. These algorithms make use of the space-sweep paradigm.

5.1 Translating Convex Polygons to Contain a Maximum Number of Points

The problem that will be solved in this section can be stated as following:

Problem 5.1.1. *Given a convex polygon P with m vertices and a set of n points in plane S, find a translation τ so that $\tau(P)$ contains a maximum number of points in S.*

5.1.1 Preliminaries

The vertices of the convex polygon P, $p_1, \ldots, p_m$, are stored in a list, in clockwise order. Without loss of generality, we assume that p_1 is at the origin. The edge from p_i to p_{i+1} will be denoted e_i, for $1 \leq i < n$; e_n denotes the edge from p_n to p_1. This structure can be enriched with an indexing method, so that the i'the vertex can be accessed in constant time.

We have to find a maximal subset S' of a set S of n points so that a translated copy of P contains S'. In general, if S' is contained in a translated copy of P, there exists a (non-countable) infinity of translations τ with $\tau(P)$ still containing S. On the other hand, the problem of mere decision whether a given set of points can be covered by a translated copy of P is not trivial.

5.1.2 Developing the algorithm

First Idea: Translation Stable Placement

In Chapter 3 we have proved a series of results that provide us with theoretical tools for attacking the problem. According to Corollary 3.3.1, if a subset S' of S is contained in $\tau(P)$, for a certain translation τ, then there exists a translation τ^* so that $\tau^*(P)$ is in Translation Stable Placement (TSP) with respect to S' – i.e., at least two points in S' are on the boundary of $\tau^*(P)$, or at least one point in S' lies on a vertex of $\tau^*(P)$. It follows that, in order to decide whether there exists a translation τ so that the subset S' is contained in $\tau(P)$, it is enough to consider all the TSP's of P and check if there is one that cover S'.

For the computation of TSP's, Theorem 3.4.1 gives as the necessary tool: given two points q_i and q_j in S, the TSP of P with respect to these points can be determined by computing the intersection between $\tau_{q_i-p_1}(P)$ and $\tau_{q_j-p_1}(P)$.

It seems, thus, that we have all tools necessary to solve Problem 5.1.1 at hand. An immediate method would:

- extract subsets S' of points in S, of cardinals ranging from $|S|$ to 2;
- check whether there exists a translation τ of P, with $\tau(P)$ in TSP, so that $S' \subset \tau(P)$;
- stop, when such a translation if found; the current S' is reported.

Well, this method is rather time-consuming – in the worst case, the number of subsets to be examined is exponential in the size of S. This is why, in the following, we try to develop a faster algorithm.

Second Idea: Restrict the Search to Pairs of Points

We start from the TSP's: for a subset $S' \subset S$, if there exists a TSP of P with respect to S', then, for a certain translation τ, we have $S' \subset \tau(P)$, and there are two points, say q_1 and q_2, in S', so that $q_1, q_2 \in \partial\tau(P)$. This sentence is also true for the maximal subset of S that can be covered by a translated of P. In other words, among the set of pairs of points that admit a TSP of P, there are pairs corresponding to the translation of maximum containment.

This idea leads us to a faster algorithm, sketched in the following:

- for each pair of points $q_i, q_j \in S$
  - check whether there exists a translation τ such that $\tau(P)$ is in TSP with respect to $\{q_i, q_j\}$;
  - if yes, then count the number of points in S contained in $\tau(P)$;
  - record the translation of maximum containment;
- report the translation recorded.

This method is much faster than the previous one, but it can still be improved.

Third Idea: Anchored Sweep

In the algorithm from above, whenever two pairs of points are relatively close, the corresponding TSP's of P will contain almost the same subset of S;

however, the subset covered is re-computed from scratch. At this point, we can bring in the convexity properties of P in order to develop an anchored-sweep method.

The idea is to keep track of a set of translations whose effect is to continuously move P, keeping it anchored with one point in S on the boundary. For some specific translations, the boundary of P will "touch" another point in S – therefore, a new TSP is reached.

The following easy result characterizes the space of translations of P that keep it anchored with vertex p_1 at a point q:

Lemma 5.1.1. *A vector* $\mathbf{v}$ *has the property that*

$$q \in \partial\tau_{\mathbf{v}}(P)$$

if and only if there exists a point

$$r \in \partial\tau_{q-p_1}(-P)$$

such that

$$\mathbf{v} = r - p_1.$$

Here $-P$ *is the notation for the symmetrical of* P *around* p_1*.*

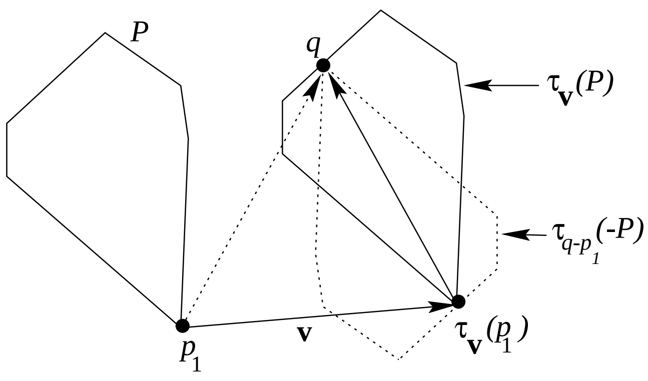

Figure 5.1: The space of translations of P anchored on q is $\partial\tau_{q-p_1}(-P)$.

Proof. Let $\mathbf{v}$ be a vector with $q \in \partial\tau_{\mathbf{v}}(P)$ (see Fig. 5.1). The vector $q-\tau_{\mathbf{v}}(p_1)$ joins two points on the boundary of P, namely p_1 and $\tau_{-\mathbf{v}}(q)$; therefore, the vector $\tau_v(p_1) - q$ will join two points on the boundary of $-P$, thus the point

$$p_1 + \mathbf{v} = p_1 + (q - p_1) + (\tau_{\mathbf{v}}(p_1) - q)$$

will be on $\partial(\tau_{q-p_1}(-P))$.

Let now r be a point of $\partial\tau_{q-p_1}(-P)$, and let $\mathbf{v} = r - p_1$. Similarly, $r - q$ is a vector that joins p_1 with another point on $\partial\tau_{q-p_1}(-P)$, therefore $p_1+(q-r)$ will join p_1 with a point on ∂P. It follows that

$$q = p_1 + (r - p_1) + (q - r)$$

is on $\partial\tau_{\mathbf{v}}(P)$. ■

Assume, for instance, that q_1 is the point in S that anchors P. Let q_2 be another point in S that can be swept through by the boundary of P. Let $\mathbf{v}$ be the vector $q_2 - q_1$, and assume the longest chord with direction $\mathbf{v}$ is longer than (q_1, q_2). There are precisely *two* chords of length and direction $\mathbf{v}$, on each side of the longest chord. It follows that there are exactly two translations τ_1 and τ_2 so that $q_1, q_2 \in \partial\tau_1(P), \partial\tau_2(P)$ (see Fig. 5.2).

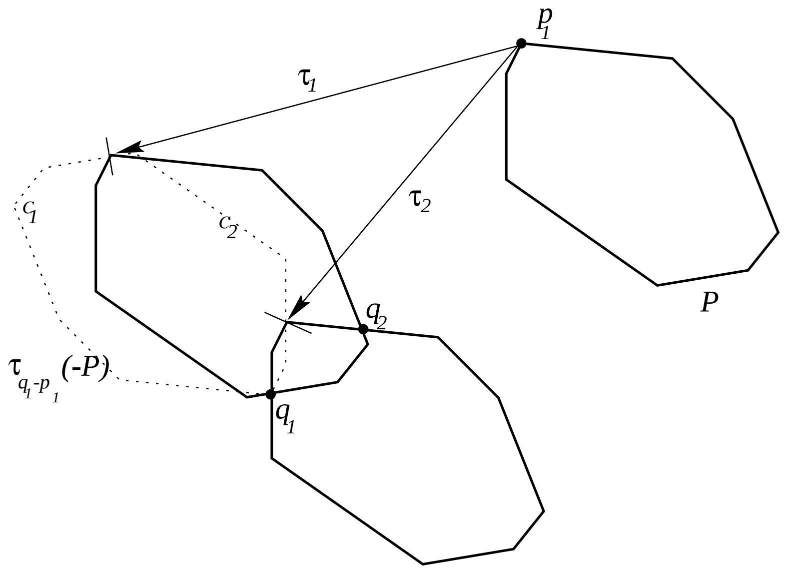

Figure 5.2: The two TSP's of P with respect to q_1 and q_2.

Thus, for two specific positions of P anchored on q_1, the point q_2 is on the boundary. These two positions are determined by two points, r_1 and r_2, on $\partial\tau_{q_1-p_1}(-P)$:

$$q_1, q_2 \in \partial\tau_{r_1-p_1}(P) \cap \partial\tau_{r_2-p_1}(P).$$

The points r_1 and r_2 split $\partial\tau_{q_1-p_1}(-P)$ into two chains, $\mathcal{C}_1$ and $\mathcal{C}_2$:

- for every point $r \in \mathcal{C}_1$, $q_2 \notin \tau_{r-p_1}(P)$;
- for every point $r \in \mathcal{C}_2$, $q_2 \in \tau_{r-p_1}(P)$;

(Recall that q_1 is always on $\tau_{r-q_1}(P)$, for $r \in \mathcal{C}_1 \cup \mathcal{C}_2$.) The endpoints of these two chains correspond to TSP's of P with respect to q_1 and q_2. For vectors $\mathbf{v}$ that translate p_1 on $\mathcal{C}_2$, the segment q_2q_1 is a portion of a chord (with q_1 on the boundary) of $\tau_v(P)$, while for vectors $\mathbf{v}$ that translate p_1 on $\mathcal{C}_1$, the segment q_2q_1 is longer than the chord with one endpoint q_1. This means that, during the sweep of P with p_1 anchored on q_1, the point q_2 traverses two times the boundary, once for entering and once for leaving P. This is the essential fact that enables us to develop the next algorithm.

We consider that the boundary of P is parameterized, with a parameter ranging over an interval of the reals. Without loss of generality, we can consider the parameterization as a function $\sigma : [0,1] \to \partial P$; $p_1 = \sigma(0) = \sigma(1)$.

Fix a point $q_1 \in S$; for every other point $q_2 \in S$, the algorithm computes the two values of the parameter t, t_1 and t_2, so that $q_1 = \tau_1(\sigma(t_1)) = \tau_2(\sigma(t_2))$ and $\tau_1(P)$ and $\tau_2(P)$ are the two TSP's of P with respect to q_1 and q_2.

These values are collected into a sorted structure, which is then swept. Each element of this structure corresponds to a *change* in the configuration: each value corresponds to a translation where a point of S, different from q_1, leaves or enters P. By keeping track of these changes, and of the points in S that are covered by P for each value of the parameter, the position of maximum containment can be detected.

Intuitively, the values of the parameter that correspond to TSP's of P determine on the parameter space a set of intervals; for each point $q_i \in S, q_i \neq q_1$, there exists such an interval, $[t_{b_i}, t_{e_i}]$, with the property that, for all $t \in [t_{b_i}, t_{e_i}]$, $q_i \in \tau_{q_1-\sigma(t)}(P)$ (see Fig. 5.3).

Once these intervals are computed, it remains to scan the sequence of their endpoints and find a region which is covered by the most of them. Figure 5.4 contains a formal description of the algorithm.

Lines 2-7 prepare the queue of events: here, the events are values of the parameter t. Each such value gives a position on the boundary of P; the translation that put P with q_i at that position will be TSP's with respect to q_i and yet another point q_j. When t passes through such a position, the polygon "moves" so that a new q_j enters or leaves.

Line 8 initializes the structure that is changed during the sweep, namely,

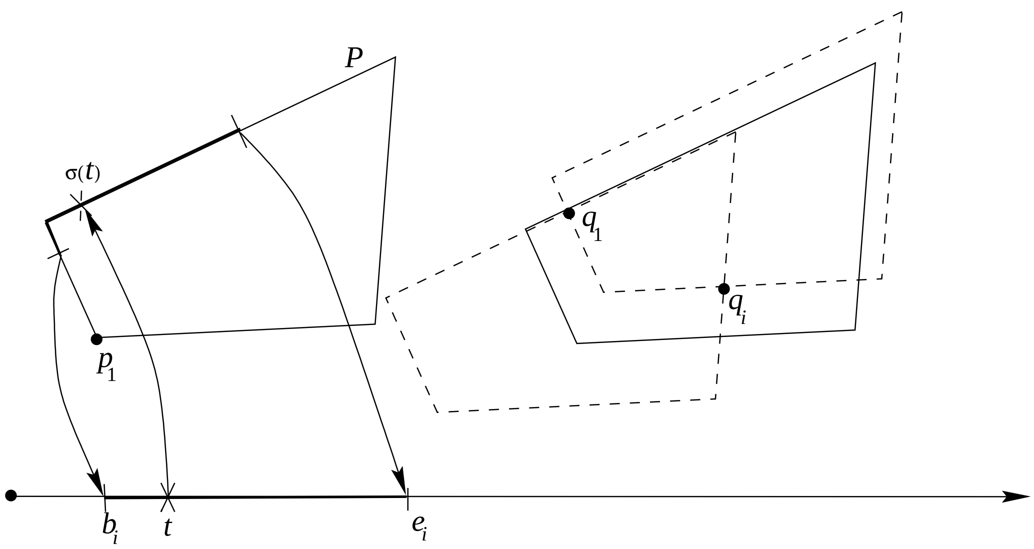

Figure 5.3: Interval of containment for point q_i, in the parameter space of all translations that keep P anchored on q_1. The TSP's of P with respect to q_1 and q_i are drawn with dashed lines. With continuous line the image of P under the translation that puts $\sigma(t)$ on q_1 is drawn, for a value of t in the parameter space.

the inclusion/exclusion information added to the points in S: At the beginning, with P translated along the vector $q_i - p_1$, some points will be contained in $\tau_{q_i - p_1}(P)$, other points will be in the exterior.

Lines 10-15 do the sweep. At each value of the parameter, a new point changes its position against P. If this point will be contained in P, it is counted to the number of points already contained. The maximum number of points contained is continuously updated (line 13).

It may well happen that the points in S be distributed on an area much larger than P, so that, for a lot of pairs of points q_i and q_j in S, there is no translation of P that has them on the boundary. However, our algorithm attempts to find such a translation.

Fourth Idea: Limiting the Search Space More

For pairs of points that are too distant, it is futile to try to find a TSP of P. In order to limit the number of pairs of points that are examined, we use

Input. a convex polygon P, given as the sequence of its vertices,
  $\{p_1, \ldots, p_m\}$;
 a set of n points $s = \{q_1, \ldots, q_n\}$;
Output. a vector $\mathbf{v}$ with the property that
  $\tau_{\mathbf{v}}(P) \cap S$ contains a maximum possible number of points;
begin
1. $nMax := 0$; $vMax := (0, 0)$;
2. **for** each point $q_i \in S$ **do**
3. $nCon := 0$;
4.   **for** each $j \neq i$ **do**
5.   compute TSP's of P with respect to q_i and q_j;
6.   compute parameters t_1 and t_2 of the (at most) two points
     on ∂P translated to q_i by these TSP's ;
7.   insert (t_1, j) and (t_2, j) in a priority queue Q,
      ordered after the first elements in the pairs;
8.   **if** q_j is contained in $\tau_{q_i - p_1}(P)$
     /* *this translation corresponds*
      to the value $t = 0$ *of the parameter* */
     then Mark q_j "IN" ; $nCon := nCon + 1$
     else Mark q_j "NOT IN";
     end if
9.   **end for**
10.   **while** Q not empty **do**
11.   extract (t, j) from the front of Q
12.   **if** q_j is not "IN"
     then $nCon := nCon + 1$; mark q_j "IN"
     else $nCon := nCon - 1$; mark q_j "NOT IN"
     end if
13.   **if** $nCon > nMax$ **then** $nMax := nCon$; $vMax = q_i - \sigma(t)$;
14.   **end while**
15. **end for**
16. report the vector $vMax$
end.

Figure 5.4: Algorithm `Max-2Dcontainment-By-Anchored-Sweep`.

bucketing: We make use here of Corollary 3.5.1 (section 3.5) for distributing the points into buckets. According to this corollary, we can find a rectangle R_P that encloses P, and, if k is the maximum number of S points that can be covered by this rectangle, there exists a translation τ so that $\tau(P)$ contains $\Omega(k)$ points.

The plane will be partitioned by a grid whose cells are identical to R_P. Then S is partitioned, all points that lie in the same cell of the grid will belong to the same bucket; we get in this way a finite number of buckets. For a bucket b, we define its *neighborhood* B as being the buckets corresponding to the eight cells that surround it, plus itself. Clearly, for a point $q_i \in b$, a tentative pair will lie in B. We denote the neighborhood of the bucket that contain the point q by B_q.

With this improvement, line 4 of the algorithm changes with:

4. **FOR** each $q_j \in B_{q_i},\ j \neq i$ **DO BEGIN**

and a preprocessing phase is required, in which the points are partitioned into buckets. The most suitable structure for storing the buckets is a hash table. Here is a description of the preprocessing:

1. compute the diameter D of P;
   let p_i and p_j be the endpoints of D;
   reorient the axes so that $D||Ox$;
2. determine the vertices of P of maximum distances
   to the supporting line of D;
   let d_1 and d_2 be the distances of these points to support(D);
   let $h = d_1 + d_2$, l=length(D);
3. compute the smallest area rectangle R_S so that
   $S \in R_S$ and has two edges parallel to D;
   let (x_0, y_0) be the coordinates of the lower-left corner of R_S;
4. for each point $q = (x_0, y_0) \in S$
5. compute $(i_q, j_q) := \left(\frac{x_q - x_0}{l}, \frac{y_q - y_0}{l}\right)$;
6. insert q in the hash table on the position corresponding to (i_q, j_q).

5.1.3 Analysis

We now present an asymptotic analysis of the time required by Algorithm 1, improved as discussed in the previous paragraph.

The Preprocessing Phase. We recall that P has m vertices, and S has n points. We denote by k the maximum number of points in S that can be contained in a translated copy of P.

Steps 1 and 2 of the preprocessing algorithm require $O(m)$ operations.

Step 3 require $O(n)$ operations.

The size of the hash table that stores the buckets is of the order $\frac{\text{Area}(R_S)}{\text{Area}(R_P)}$, and this can grow pretty large. However, there are at most n buckets that can store points in S; the size of the hash table cannot grow over $O(n)$.

The Algorithm. The computation of TSP's of P with respect to two points, in line 5 of the algorithm, requires $O(\log m)$ operations, as explained in Section 4.2.1.

The computation of the parameters of two points (line 6) on the boundary of P can be done in constant time.

The insertion of an element into a priority queue requires logarithmic time with respect to the size of the queue. Now, there can be as many intersection points as two times the number of points in a neighborhood – i.e., in a block of nine adjacent buckets. By corollary 3.5.1, the number of points that can lie in one bucket is $O(k)$; thus, any neighborhood can contain $O(k)$ points. This means that, for a point q, at most $O(k)$ events are to be inserted in Q: each insertion requires then $O(\log k)$ time.

So far, we have computed the time needed by the **for** loop in lines 4 – 9: $O(k(\log m + \log k))$.

The inner steps of the **while** loop in lines 10 – 14, executed $O(k)$ times, are constant-timed.

The outer **for** loop (lines 2 – 15) is executed n times; we have just calculated that each cycle requires $O(k \log(m + k))$ operations. This algorithm requires therefore a total of $O(nk \log(m + k))$ time.

5.1.4 Another Approach

For the planar case, there exists another, conceptually different algorithm for solving Problem 5.1.1. This algorithm is based on the plane-sweep technique.

We use Lemma 5.1.1, that relates the space of translations of P anchored on a point $q \in S$ to $\tau_{q-p_1}(-P)$, where $-P$ is the symmetric of P with respect to its vertex p_1. It is easy to see that for every point $r \in \tau_{q-p_1}(-P)$, $q \in \tau_{r-p_1}(P)$; $\tau_{q-p_1}(-P)$ is the containment region of q (see Fig. 5.5).

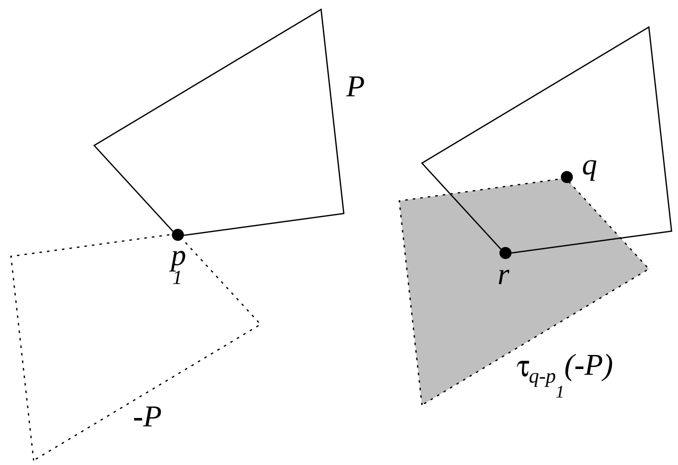

Figure 5.5: The containment region of a point q.

Let now $q_i, \ldots, q_k \in S$, and assume that

$$Q = \bigcap_{1 \leq i \leq k} \tau_{q_i - p_1}(-P) \neq \emptyset.$$

Then for every point $r \in Q$ and for every $i \in \{1, \ldots, k\}$, $\tau_{r-p_1}(P)$. We are led to the following truth:

Lemma 5.1.2. *A vector* $\mathbf{v}$ *corresponding to a translation of* P *that contains a maximum number of points in* S *has the property that* $p_1 + \mathbf{v}$ *is a point in a region of maximum depth in the arrangement of polygons* $\tau_{q_i-p_1}(-P)$, $1 \leq i \leq n$.

This region can be determined by a line-sweep algorithm. We give here a short description of this algorithm, together with its analysis.

We are given a set of $n > 0$ polygons, $\mathcal{P} = \{P_1, \ldots, P_n\}$, each of them obtained by translating a convex polygon P along a certain vector. We are looking for a region in the plane that belongs to a maximum number of polygons in $\mathcal{P}$. We can define a function on the plane, *depth*, that associates to each point p the number of polygons in $\mathcal{P}$ to which p belongs; we seek thus a region in the plane whose points have maximum depth.

The boundary of every polygon is split into two x-monotone chains of edges. A point p belongs to two polygons if it is *under* both their upper chain, and *above* both their lower chain. The intersections of the chains of edges with a vertical line can be ordered by the y-coordinate, defining a linear order among the chains – except at an intersection point: here, the order changes.

This order changes also when the leftmost and rightmost points of polygons in $\mathcal{P}$ are reached.

We have now all the elements to required by a plain plane-sweep algorithm. The line status will consist of the chains intersected by the sweeping line, ordered according to their y-coordinate. The information that will be stored comprises the chains of edges (ordered), and the depth of the regions between consecutive chains. This structure will change when a leftmost or rightmost point of a polygon is encountered, and at intersections of chains.

There are two types of events: the abscissae of leftmost and rightmost points of polygons in $\mathcal{P}$; these events are known from the beginning, and they initialize the event queue. Events of the second type are intersections of chains; they are computed dynamically, during the sweep.

The algorithm is presented in Figure 5.6.

Analysis

For storing the events, a priority queue will be used. Both insertion of a new event and extraction of the first event take logarithmic time with respect to the length of the queue. There are $2n$ leftmost and rightmost points of polygons in P; as about the estimated number of intersections, the analysis of M. Sharir ([Sharir91], see also Chapter 3, Section 3.6 of this thesis) states that there are $O(nk)$, where k is the maximum depth. Thus, the total amount of time for handling the event queue is of order $O(nk \log n)$.

The line status structure records the chains of edges sorted according to their y-coordinate, at a certain abscissa. Updating the line status consists in either inserting new neighboring chains, eliminating neighboring chains,

Input. a convex polygon P of $\{p_1, \ldots, p_m\}$;
a set of n points $S = \{q_1, \ldots, q_n\}$;
Output. a vector $\mathbf{v}$ with the property that
$\tau_{\mathbf{v}}(P) \cap S$ contains a maximum possible number of points;
begin
initialize the event queue with the abscissae
of leftmost and rightmost points of polygons in $\mathcal{P}$;
initialize the empty line status
$\mathbf{v}$:=(0,0); $nMaxCover := 0$;
while there are any events in the queue
extract the leftmost event e from the queue;
if e is the leftmost point of a polygon P_l in $\mathcal{P}$
then insert the chains of P_l into the line status;
update the depth of the regions between these chains;
if the depth d of one of these regions is $> nMaxCover$
then $nMaxCover := d$;
$\mathbf{v} := r - p_1$, for a point r in this region;
end if;
check the chains for intersection against their neighbors;
if there are any intersection points
then insert them into the event queue;
end if
end if
if e is the rightmost point of a polygon P_l in $\mathcal{P}$
then eliminate the chains of P_l from the line status;
update the depth of the region between their neighboring chains;
check the neighboring chains for intersection;
if there are any intersection points with greater abscissa
then insert them into the event queue;
end if
end if
if e is an intersection point
then change places of the chains in the line status;
check these chains for intersection against their neighbors;
if there are any intersection points with greater abscissa
then insert them into the event queue;
end if
end if
end while
report $\mathbf{v}$.
end.

Figure 5.6: Algorithm `Max-2Dcontainment-By-Line-Sweep`.

or changing places of neighboring chains. If a balanced ordered tree is used, these operations take logarithmic time with respect to the size of the structure. Since there are at most $2n$ chains, the total time for updating the line status is of order $O(n \log n)$.

It remains to study the time spent on computing intersections. Et each event point, chains of edges are tested for intersection – thus an intersection computation is launched $O(nk)$ times. As described in Subsection 4.2.1, for each intersection $O(\log |P|)$ operations are done. If P has m vertices, $O(nk \log m)$ time goes with computing intersections.

Summing up, the overall time of this algorithm is $O(nk \log mn)$. Seeminlgy the time is asymptotically greater than for the first algorithm presented here. However, the technique uses only consecrated data structures, and can be easily implemented.

5.2 Translating Convex Polyhedra to Contain a Maximum Number of Points

In this section we give a solution for the problem:

Problem 5.2.1. *(***3-D Translation of Max Containment***) Let P be a polyhedron in the three-dimensional space and S a set of points. Determine a translation τ such that $\tau(P)$ contains the maximum number of points in S.*

5.2.1 Preliminaries

In all what follows, P is a convex polyhedron with m vertices in the three-dimensional space. Unless stated otherwise, its vertices are supposed to be in general position: the polyhedron is simplicial (each of its faces is a triangle). Without loss of generality, we consider the vertex p_1 as the origin (i.e. having coordinates (0,0,0)).

S is a set of n points, $q_1, \ldots, q_n$, that are also supposed in general position: no four of them are coplanar; no edge or face of P is parallel with the line determined by any two of them.

According to Corollary 3.3.1, **Existence of TSP's**, if a subset $S' \subseteq S$ has all its points contained in P, there exists a TSP of P with respect to S'. This means, there exists a translation τ with the property that $\#(\partial\tau(P) \cap S') \geq 3$. The boundary of P consists of faces, edges and vertices. If a point in S' lies on an edge, it will be counted two times, because it is contained in two faces; if it lies in a vertex, it will be counted three times.

There are two questions that must be answered:

Question 5.2.1. (Find a 3-anchored Translation)
Given three points in S, how can one find a translation τ such that $\tau(P)$ contains these points on the boundary, if such a translation exists?

Question 5.2.2. *(***Number of 3-anchored Translations)**
Given three points in S, how many translations τ are there, so that $\partial(\tau(P))$ contains these points?

The answer to Question 5.2.1 is provided by Theorem 3.4.1, **connecting intersections with TSP's**. Its three-dimensional formulation is:

Lemma 5.2.1. (3-Anchored Positions) *The points q_1, q_2, q_3 are on $\partial\tau_{\mathbf{v}}(P)$ iff $\mathbf{v} = q - p_1$, for $q \in \gamma_{1,2,3} := \partial\tau_{q_1-p_1}(-P) \cap \partial\tau_{q_2-p_1}(-P) \cap \partial\tau_{q_3-p_1}(-P)$, and $-P$ is the symmetric of P with respect to p_1.*

Now, if we want to find a translation τ of P such that three points q_1, q_2, q_3 are on $\partial\tau(P)$, we will compute the intersection $\partial\tau_{q_1}(-P) \cap \partial\tau_{q_2}(-P) \cap \partial\tau_{q_3}(-P)$. If we translate P with p_1 in any point in the set determined as before, we can be sure that it will contact all three initial points on its boundary.

If we weaken the conditions, i.e., if we consider only two points, q_1 and q_2, the set of all translations τ such that $\partial\tau(P)$ contains q_1 and q_2 can be also computed using a similar argument. The following result, a corollary of Lemma 4.2.1, **intersection of translated polyhedra**, deals with this kind of translations; it will be used both for theoretical purposes – for giving an answer to Question 5.2.2 – and for devising the algorithm.

Corollary 5.2.1. (2-Anchored Positions) *The set of all points q such that $\partial\tau_{q-p_1}(P)$ contains q_1 and q_2, if not empty, is a simple polygonal line in space whose projection on a plane orthogonal to $q_2 - q_1$ is a convex polygon.*

Proof. The fact that all translations that put P in positions with q_1 and q_2 on the boundary put p_1 on points in $\tau_{q_1-p_1}(-P) \cap \tau_{q_2-p_1}(-P)$ can be proven using the same argument as in Theorem 3.4.1. The fact that this intersection, if not empty, is a closed polygonal line with the properties stated before has been proven in Lemma 4.2.1. ∎

In all what follows, by $\gamma_{1,2}$ we denote the polygonal line obtained by intersecting $\tau_{q_1-p_1}(-P)$ and $\tau_{q_2-p_1}(-P)$.

Remarks.

1. The polyhedron $\tau_{q_1-p_1}(-P) \cap \tau_{q_2-p_1}(-P)$ has, as its boundary, subsets from the boundaries of $\tau_{q_1-p_1}(-P)$ and $\tau_{q_2-p_1}(-P)$; due to the convexity of these two, the number of elements of its boundary (faces, edges and vertices) is $O(m)$.

   Since $\gamma_{1,2}$ is a closed chain of edges on the boundary of $\tau_{q_1-p_1}(-P) \cap \tau_{q_2-p_1}(-P)$, the number of its edges is also of order $O(m)$.

2. Theoretically, this lemma establishes a very important fact: *the convexity of the projection of $\gamma_{1,2}$ on the plane π orthogonal on $q_2 - q_1$.*

Consider another closed polygonal line in space – for instance, the chain $\gamma_{1,3}$, the intersection of the boundaries of $\tau_{q_1-p_1}(-P)$ and $\tau_{q_3-p_1}(-P)$. In general, this polygonal line projects on π into a non-convex (and even non-simple) polygon. However, due to the convexity of $\gamma_{1,2}$, the number of points in $\mathrm{proj}_\pi(\gamma_{1,2}) \cap \mathrm{proj}_\pi(\gamma_{1,3})$ is $\mathrm{O(m)}$.

Now, clearly, if $p \in \gamma_{1,2} \cap \gamma_{1,3}$, then $\mathrm{proj}_\pi(p) \in \mathrm{proj}_\pi(\gamma_{1,2}) \cap \mathrm{proj}_\pi(\gamma_{1,3})$. We have thus the following lemma:

Lemma 5.2.2. (Number of 3-anchored Translations) *The number of TSP's of P with respect to three given points is $O(m)$.*

We have to emphasize here that all this discourse is valid only for points in general position. There are, certainly, configurations in which the number of TSP's of P with respect to three given points is infinite – for example, when all three points q_1, q_2, q_3 lie on a face of P; we have excluded such situations from the very beginning, as we have imposed as a condition for general position the fact that no two points in S lie on a straight line parallel to an edge or a face of P.

3. We invoke Corollary 5.2.1 to justify another technique that will be used in the algorithm: *the local linearization of the space of translations.* For two points in S, q_1 and q_2, let $\gamma^p_{1,2} := \mathrm{proj}_\pi(\gamma_{1,2})$, where π is a plane orthogonal to $q_2 - q_1$.

   Since $\gamma^p_{1,2}$ is a convex polygon, it can be parameterized linearly with a parameter t in, say, the interval $[0, 1]$. We again invoke the general position of points in S and vertices of P: no edge in $\gamma_{1,2}$ is orthogonal to π. It follows that $\gamma^p_{1,2}$ is a one-to-one and onto image of $\gamma_{1,2}$, therefore we can see the parameterization as a parameterization of $\gamma_{1,2}$.

   Now $[0, 1]$ is a linear space that can be swept.

   The intersections of $\gamma_{1,2}$ and $\gamma_{1,3}$ correspond to some particular values of t. They also correspond to translations that put P in TSP with respect to q_1, q_2 and q_3: When translating P anchored with p_1 on $\gamma_{1,2}$, at certain points the boundary of P will touch q_3. These points are exactly $\gamma_{1,2} \cap \gamma_{1,3}$. It is obvious that, between two such intersections, q_3 is either inside P or outside it. In fact, during a sweep of P with p_1 anchored on $\gamma_{1,2}$, at each point in $\gamma_{1,2} \cap \gamma_{1,3}$, q_3 changes its position from the exterior to the interior of P and reversely.

We can then record the intersections of $\gamma_{1,2}$ with $\gamma_{1,i}$, for $i \neq 1, 2$, $i \in \{1, 2, \dots, m\}$ by storing the corresponding values of the parameter t. Then, by sweeping the domain of t, we obtain the translation of P of maximal containment, so that q_1 and q_2 lie on the boundary.

The third remark from before contains the essence of our algorithm. Basically, pairs of points in S are considered, and for such a pair (q_i, q_j), the polygonal line $\gamma_{i,j}$ is computed. Then for every other point q_k, the contour $\gamma_{i,k}$ is intersected with $\gamma_{i,j}$, and the intersection points are collected in a structure ordered after their position on the $\gamma_{i,j}$. Finally, the anchored sweep is simulated, by taking in turn each of the intersection points. At such a point, the configuration changes: one of the q_k's enters P or exits from P. During this process, the position of P that contains the maximum number of q_k's is captured.

In order to gain some speed-up, we shall use another result, necessary for reducing the search space.

The algorithm will take pairs of points in S, (q_i, q_j), and produce the corresponding chains of straight-line segments $\gamma_{i,j}$; if the two points in such a pair are too distant, no translated of P will have them both on the boundary – in other words, the two translates of P with p_1 in each of the two points will not intersect.

We use a similar argument as in section 5.1 to reduce the number of pairs that will be considered: *bucketing*. The space containing the points in S is split into rectangular boxes (the buckets) shaped to enclose P. Every point in S will be assigned to the bucket that contains it. When considering pairs of points, they are taken from neighboring buckets. Moreover, only buckets that contain at least one point will be effectively stored.

We are now in position to give the algorithm and its analysis.

5.2.2 The Algorithm

Details related to the storage of the polyhedron P have been presented in Chapter 4, Section 4.1.2. For an algorithm for computing the intersection of two convex polyhedrons, we refer to the same chapter, Section 4.2.2. We consider here as a black-box the algorithm for determining the position of a point against a convex polyhedron.

We use the following data structures to store the objects that the algorithm deals with:

- the set of points S is stored in an array indexed with the first natural numbers;
- two arrays of points store the vertices of $\gamma_{i,j}$ and $\gamma_{i,k}$, for $i \neq j \neq k \in \{1, \ldots, n\}$;
- a priority queue Q with elements of the form (x, i), where x is a point parameterized by a scalar value t and i is a natural number. The underlying order for the elements of Q is given by the values of the parameter t. The queue may be implemented, for instance, as an AVL tree (see [AhoHU83]);
- a hashing table whose elements represent the *buckets*.

For each point i, we denote by B_i its *neighborhood*; it consists of its own bucket and of the buckets with which its bucket shares a face, an edge or a vertex. It is clear that if a translated copy of P contains two points on the boundary, these points must lie in neighboring buckets, if not in the same.

For a point p on a chain of edges $\gamma_{i,j}$, parameterized with the value t_0, we will denote the translation of P of vector $p - p_1$ by τ_{t_0}. All the intersections of $\gamma_{i,j}$ with some other $\gamma_{i,k}$ will be stored in the priority queue Q according to the value of their parameter.

During the sweep of P anchored in the points q_i and q_j, its boundary will touch another point q_k in S; this position of P corresponds to some element of the form (x, k) stored in Q. When the sweep proceeds further, the point q_k changes its position against P (from outside to inside and vice-versa).

This movement is reflected by the work with the elements of Q. At each moment, the first element in Q corresponds to the first point that will be touched by P. The change of position of the points q_k when an element (x, k) in Q is processed enables us to keep track of the points that are contained in P, and thus to store the position of P having a maximum number of points in S inside.

5.2.3 The Algorithm

The sketch of the algorithm is presented in figure 5.7.

Input. a polyhedron P of vertices $p_1, \ldots, p_m$;
a set S of m points in space, $\{q_1, \ldots, q_m\}$;
Output. a vector $\mathbf{v}$ with the property that $\tau_{\mathbf{v}}(P) \cap S$
contains a maximum possible number of points;

preprocess S by distributing its points into buckets;
$max := 0$;
for each point $q_i \in S$ **do**
    $Q := \emptyset$;
    for each point q_j in B_i **do**
        compute $\gamma_{i,j}$
        $c := 0$ /* *c denotes the number of points contained in P* */
        if $\gamma_{i,j}$ is nonempty
        then for each point q_k in B_i **do**
                compute $\gamma_{i,k}$
                compute $\gamma_{i,k} \cap \gamma_{i,j}$
                insert (x, k) into Q, for each $x \in \gamma_{i,k} \cap \gamma i, j$
                if $q_k \in \tau_0(P)$
                then mark q_k "IN"; $c := c + 1$;
                    if $c > max$
                    then $max := c$; store the vector $x - p_1$
                    end if
                else mark q_k "NOT IN"
                end if
            end for
            while Q not empty **do**
                delete (x, k) from front of Q
                if q_k is not "IN"
                then $c := c + 1$; mark q_k "IN"
                else $c := c - 1$; mark q_k "NOT IN"
                end if
                if $c > max$
                then $max := c$; store the vector $x - p_1$.
                end if
            end while
        end if
    end for
end for
report the vector stored;
end.

Figure 5.7: Algorithm `Max-3dcontainment-By-Anchored-Sweep`.

5.2.4 Analysis

We will consider in turn each cycle from the algorithm, starting from the innermost. By d we denote the maximum number os S points that can be contained by a translated copy of P.

To begin with, let i, j be fixed. According to Lemma 5.2.2, for each point $q_k \in B_i$ there are $O(m)$ intersections between $\gamma_{i,j}$ and $\gamma_{i,k}$. There are $O(d)$ S points in the neighborhood B_i of q_i, by Corollary 3.5.1. Therefore, the queue Q may store $O(md)$ elements, and thus, the time spent by line 12 is $O(m \log md)$.

The time for the computation in line 10 is $O(m)$, using the algorithm presented in Chapter 4, Section 4.2.2).

The intersection in line 11 is computed with an algorithm based on the convexity of the projection of $\gamma_{i,j}$ which works in in $O(m \log m)$ time. The other operations in the innermost **for** loop – lines 9-20 – are done in constant time. All in all, for this loop, the number of operations is $O(m \log md)$, and it loops $O(d)$ times.

The **while** loop in lines 21-30 mainly consists in executing a constant number of operations for each element in a priority queue of size $O(md)$, thus the time for this cycle is $O(md)$.

Now, the number of pairs q_i, q_j that lead the cycles in lines 3-33, is $O(nd)$ – since, again, q_j is taken from the neighborhood of q_i – therefore the total time of the algorithm is $O(nmd^2 \log md)$.

The space used by the algorithm is $O(md + n)$. Indeed, there should be stored: the set S of points (n) and the buckets $(O(n))$; the polyhedron P $(O(m))$; the points in $\gamma_{i,j}$ and $\gamma_{i,k}$ $(O(m))$; the elements of Q, whose size could be as large as $O(md)$.

5.3 Optimal Placement on the Line

In the one-dimensional instance of Problem 1.2.1, the convex polytope is reduced to a closed interval of given length l, and the vector $\mathbf{v}$ has only one coordinate. In other words, given a set of n reals S, one looks for a real value v such that $[v, v + l] \cap S$ is of maximal cardinality.

We have shown in Section 3.7 that an optimal algorithm would solve this problem in $O(n \log n)$ time. We present in the following such an algorithm.

First of all, the values in S are sorted: this is the most time-consuming operation, requiring $O(n \log n)$ operations.

Then, the algorithm starts sweeping the array of points: The number of points closer to $\min(S)$ than l is computed – by a binary search, the position of $\min(S) + l$ in the sorted array S is determined; intuitively, a segment of length l has been placed with the leftmost endpoint at $\min(S)$. Then, this segment slides to the right, until either a new (bigger) point is reached, or a small value is left behind. This process continues until $\max(S)$ is reached, continuously updating the position in which the interval contains the maximum number of points.

Figure 5.8 contains a formal descritption of the algorithm.

Theorem 5.3.1. *Algorithm* **LinearOptimalPlacement** *computes a correct solution of Problem 3.7.3.*

Proof. Theorem 3.2.1 ensures the fact that there is a value v such that either v or $v+l$ are among the elements of S, and the interval $[v, v+l]$ is of maximum containment. In fact, it is easy to show that for the 1-dimensional case, there exists a point q in S such that $[q, q + l]$ contains a maximum number of S points.

If this point q is the smallest, it is detected in lines 1-3. If not, it is detected in the **while** loop in lines 4-12.

Indeed, in these lines, for every $q_{k_1} \in S$ that is a potential leftmost endpoint of the interval, the number of points in $[q_{k_1}, q_{k_1+l}]$ is correctly computed. Clearly, the point that ensures maximal containment is thus detected and recorded. ∎

Analysis. Apart from line 1, the rest of the algorithm works in linear time with respect to the number of points in S: Each cycle of the **WHILE** loop in lines 4 – 12 performes at most n increments for k_1 and at most n increments for k_2.

Input. a set $S = \{q_1, \ldots, q_n\}$ of real numbers;
a real value l;
Output. a real value v with the property:
$[v, v+l] \cap S$ has maximal cardinality;
begin
1. sort the points in S; let sorted S be $\{q_1, q_2, \ldots, q_n\}$;
2. by a binary search, compute the number k_2 of S points smaller than $q_1 + l$;
3. $nCovered = nMaxCovered = k_2$; $vMaxCover = q_1$;
$k_1 = 1$;
$\{$*the points* $q_{k_1}, \ldots, q_{k_2}$ *are contained in the currently translated interval*$\}$
4. **while** $k_2 < m$
5. $k_1 = k_1 + 1$; $nCovered = nCovered - 1$;
6. **while** $k_2 + 1 \leq m$ **and** $q_{k_1} + l > q_{k_2+1}$
7. $nCovered = nCovered + 1$; $k_2 = k_2 + 1$;
8. **end while**
9. **if** $nCovered > nMaxCovered$
10. **then** $nMaxCovered = nCovered$;
$vMaxCovered = q_{k_1}$;
11. **end if**
12. **end while**
13. report v.
end

Figure 5.8: Algorithm **LinearOptimalPlacement**.

The total time of the algorithm is therefore $O(n \log n)$.

5.4 Optimal Placement of Rectangles

In this section we will prove that the lower bound can be achieved for the two-dimensional case, when the polygon P is a rectangle. Using a plane-sweep technique, a region in the plane is found, with the property that every translation that puts a vertex of P in a point in the region is a translation of optimal containment.

After a short theoretical discussion, we present the main data structure used here, the *segment tree*, and the processings defined on it. The algorithm follows, and its analysis.

It has been shown in Section 3.6, more precisely in Lemma 3.6.1, that a region of maximum depth in the arrangement of symmetrics of P with respect to a vertex corresponds to translations of maximum containment. We will use here this result, and we will seek such a region in the arrangement obtained by translating P with one of its vertices to every point in S.

Let p_1, p_2, p_3 and p_4 denote the four vertices of the rectangle P, which, without loss of generality, is assumed with edges parallel to axes. The symmetric of P with respect to p_1 is a rectangle identical to P, so we could as well work directly with P. By translating P with p_3 to all points in the set S, an arrangement $\mathcal{A}$ of rectangles is obtained (see Fig. 5.9). Let R be one of the regions of maximum depth in this arrangement; by translating P with p_1 in any point of R, a maximum number of points in S is covered.

It remains to study the possibility of efficiently finding the region of maximum depth. As we have already said, a plane-sweep will be applied. A vertical line will sweep the plane from left to right; in every moment, its intersection with the set of rectangles in the arrangement $\mathcal{A}$ is maintained. The task of the algorithm is facilitated by the fact that the status of the sweeping line changes only when a vertical edge of a rectangle is reached, and this change applies to an interval on the line which corresponds to that rectangle. This means that:

- All events are given by abscissae of the form x_q or $x_q - w$, where x_q is the abscissa of a point in S, and w the width of P.

- All possible y-coordinates of intervals determined by $\mathcal{A}$ on the sweeping line are of the form y_q or $y_q - h$, where y_q is the y-coordinate of a point in S, and h is the height of the rectangle P.

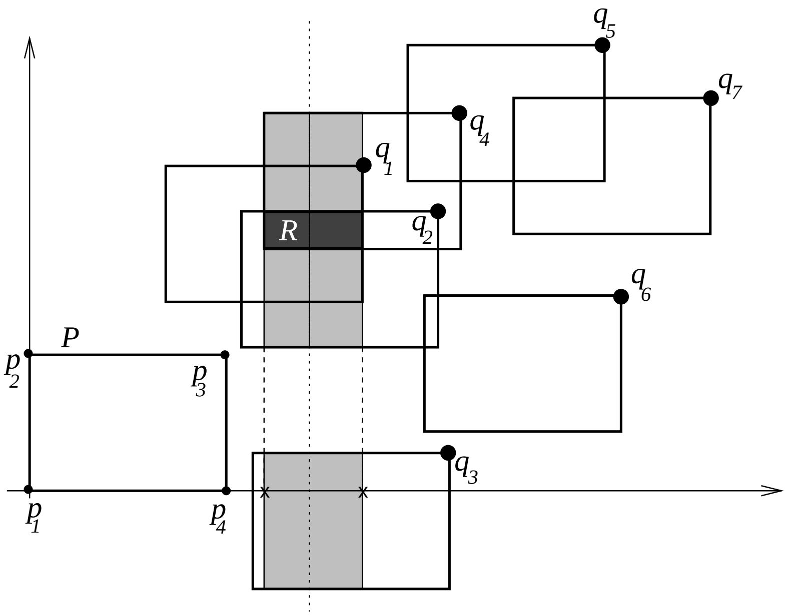

Figure 5.9: Arrangement of rectangles constructed by translating P with the vertex p_3 at every point in $\{q_1, \ldots, q_6\}$. The dark-shaded area corresponds to the region of maximum depth. The light-shaded area emphasizes a zone between two event points, during the sweep.

The two main data structures involved in the sweep, i.e., the *event queue* and the *line status*, are described in the following paragraphs.

5.4.1 Event Queue

Both the events and the edpoints of the segments on the sweeping line are known from the beginning. Moreover, there are no insertions – and practically no deletions – of events; therefore the event queue can be stored in an array with $2n$ elements. The sweep is modeled by processing in turn each element of the queue. The elements of this array must have the form (index i of a point q_i in S, left/right corner of the rectangle $\tau_{q_i - p_3}(P)$). With this information, the abscissa of an event can be computed in constant time.

The queue is assumed sorted; the sorting operation must be accomplished, at a cost of $O(n \log n)$ operations.

5.4.2 Line Status

The line status will be stored in a segment tree. In Section 4.1.2 we have presented some basic features of such a structure. We will develop here the presentation, by detailing the supported operations.

The endpoints of the segments determined by the scene (the arrangement of rectangles) on the sweeping line constitute a fixed set, known from the beginning: the set of all ordinates of vertices of the rectangles in $\mathcal{A}$. Using these values, the segment tree is constructed. The segments themselves belong to *regions of different depths* in the arrangement: their depths will be associated to nodes in the segment tree.

During the sweep, when a new rectangle is encountered, the segment determined by it on the sweeping line contributes by *increasing* the depth of all the standard segments in which it is decomposed. When the rightmost abscissa of a rectangle is encountered, the depth of all standard segments covered by this rectangle decreases by 1. The corresponding operations are detailed in the algorithms in Figure 5.10. We recall that each node v of the segment tree stores the following fields:

- $B[v], E[v]$ – the edpoints of the interval associated to v;
- $LSON[v], RSON[v]$ – pointers to the descendats of v;
- $depth[v]$ – the depth of the segment $B[v], E[v]$, measured as the number of rectangles which cover the segment.

There are some global variables managed by the algorithm:

1. *currentAbscissa* – the abscissa of the current event point;
2. a quadruple ($xMaxDepth$, $yMaxDepth$, $dMaxDepth$, $hMaxDepth$), storing relevant data for the region of maximum depth:
    - $xMaxDepth, yMaxDepth$ are the coordinates of the leftmost point of this region,
    - $dMaxDepth$ is the depth in the arrangements,
    - $hMaxDepth$ is the height of the region (all points in this region have their y-coordinates between $yMaxDepth$ and $yMaxDepth + hMaxDepth$).

```
procedure insert(b, e, v)
Input.     b, e - the endpoints of an interval;
           v - a node in the segment tree;
begin
1.    if (b ≤ B[v] and E[v] ≤ e)
      then
2.        depth[v] := depth[v] + 1
3.        if depth[v] > maxDepth
4.        then xMaxDepth := currentAbscissa;
               yMaxDepth := E[v];
               dMaxDepth := depth[v];
               hMaxDepth := E[v] − B[v];
5.        endif
      else
6.        if (b < ⌊(B[v]+E[v])/2⌋)
7.            then insert(b,e,LSON(v));
8.        end if
9.        if (⌊(B[v]+E[v])/2⌋ < e)
10.           then insert(b,e,RSON(v));
11.       end if
12.   end if
end

procedure delete(b, e, v)
Input.     b, e - the endpoints of an interval;
           v - a node in the segment tree;
begin
1.    if (b ≤ B[v] and E[v] ≤ e)
2.    then depth[v] := depth[v] − 1
      else
3.        if (b < ⌊(B[v]+E[v])/2⌋)
4.            then delete(b,e,LSON(v));
5.        end if
6.        if (⌊(B[v]+E[v])/2⌋ < e)
7.            then delete(b,e,RSON(v));
8.        end if
9.    end if
end
```

Figure 5.10: Procedures for updating the segment tree.

Input. $S = \{q_1, \ldots, q_n\}$ - a set of points in plane;
(h, l) - height and width of the rectangle P;
Output. (xR, yR) - coordinates of the lower-left corner
of the region of maximum depth;
(hR, lR) - height and width of the region of maximum depth;

begin
1. $\{$*construct the event queue* E $\}$
1.a. **for** $i = 1$ **to** n
1.b. insert $(x_{q_i}, i, 'begin')$ and $(x_{q_i} - l, i, 'end')$ into the event queue E;
1.c. **end for**
1.e. sort E after the first member in each pair;
2. $\{$*construct the segment tree of root* $R\}$
2.a. **for** $i = 1$ **to** n
2.b. insert y_{q_i} and $y_{q_i} - h$ into an array Y;
2.c. **end for**
2.e. sort Y;
2.f. recursively construct the segment tree of root R;
2.g. $B[R] = Y[1]; E[R] := Y[2n]; depth[R] := 0$;
2.h. construct $LSON(R)$ as the segment tree
with values in $\{Y[1], \ldots, Y[n]\}$;
2.i. construct $LSON(R)$ as the segment tree
with values in $\{Y[n+1], \ldots, Y[2n]\}$;
3. $(xMaxDepth, yMaxDepth, dMaxDepth, hMaxDepth) := (0, 0, 0)$;
4. $\{sweep\}$
4.a. **for** $i = 1$ **to** $2n$
4.b. $(x, k, u) := Y[i]$;
4.c. **if** $u =' begin'$
4.d. **then** insert($y_{q_i} - h, y_{q_i}$,R);
4.e. **else** delete($y_{q_i} - h, y_{q_i}$,R);
4.f. **end if**
4.g. **end for**
5. find the index k of $xMaxDepth$ in E;
6. $lMaxDepth := E[k+1].x - E[k].x$;
7. **return** $(xMaxDepth, yMaxDepth), (lMaxDepth, hMaxDepth)$;
end

Figure 5.11: Algorithm `Rectangles-Region-of-Maximum-Depth`.

5.4.3 The Algorithm

Having at hand the procedures for udating the line status, the algorithm is immediate: it is written in Figure 5.11.

5.4.4 Analysis

Sorting the array Y of $2n$ elements takes $O(n \log n)$ operations, as well as constructing the segment tree of the $2n$ y-coordinates. In lines 4.a. – 4.g., at each of the $2n$ steps in the **for** loop, one insertion/deletion of an element in/from the segment tree is accomplished; this is done by procedures insert and delete in $O(\log n)$ time per operation.

This gives an overall of $O(n \log n)$ time, which is asimptotically optimal.

Chapter 6

Rigid Motions of Optimal Containment

For the sake of completitude in presentation, we insert here a short description of an algorithm given by Dickerson and Scharstein ([DickersonScharstein96]) for the more difficult problem of finding an isometry (a *rigid motion*) that moves a convex polytope P in a position where it contains a maximum number of points from a given set S.

This method pushes the anchored sweep one step further: intuitively, the polygon is translated withe boundary anchored on a point in plane, then, pinned at that point, it is rotated with angles between 0 and 2π. The space of these rigid motions is two dimensional: the translation can be described by one parameter, and the rotation angle is the other parameter. Particular rigid motions in this space are captured by a so-called *rotation diagram*, which will be described in the following.

6.1 Description of the Problem

A planar *rigid motion* ρ is an affine transformation of the plane with special properties. The attribute preserved by this kind of transformations is *distance*; consequently, angles, volumes, incidence, collinearity are also preserved.

It has been proved that any planar rigid motion can be written as the composition of a translation and a rotation. This property is of utmost importance for the discourse in this section.

The problem that will be discussed can be stated as follows:

Problem 6.1.1. *(Optimal Polygon Placement) Given a convex polygon P and a planar point set S, find a rigid motion ρ that maximizes the number of points contained by $\rho(P)$.*

We will sketch here the solution given by Dickerson and Scharstein in 1998 [DickersonScharstein96]. In this algorithm, the authors make heavy usage of the anchored sweep.

The results that support theoretically the method is Theorem 3.2.2, that claims the existence of a family of special rigid motions that place P in special positions, when P covers all points in S, and Corollary 3.3.2, that allows the search of special rigid motions for a general polygon and a general set of points. These results have been proven in Chapter 3.

The main concept introduced in these results was that of *general stable placements* (GSP): rigid motions that transform P into a polygon with at least three points in S on the boundary. It has been proved that it is enough to look for such rigid motions: if a rigid motion of optimal containment exists, then a GSP of optimal containment exists.

This fact suggests a first approach: the brute force. For each triple of points in S, compute all stable placements of P, and for each placement count the number of points contained. Chazelle [Chazelle83] has shown that all stable placements of an m-vertex polygon P on three points can be computed in $O(m^2)$ time. This leads to an $O(n^4 m^2 \log m)$ time algorithm for Problem 6.1.1: there are $\binom{n}{3} = O(n^3)$triples of points in S that have to be considered, and, for each triple, $O(m^2)$ time is taken for computing the stable placements. Each stable placement is tested against S, i.e., the $n-3$ remaining points in S are checked, with an easy $O(\log m)$ algorithm, whether they fall inside the polygon or not.

The algorithm that will be described here constructs a bi-dimensional object – the *rotation diagram* – in the space of all possible rigid motions of P, anchored with one point q_i in S on the boundary. As we have shown a few times so far, the boundary of a symmetric of P, translated with one vertex at a point in S, gives the anchoring space; a vertex of the polygon sweeps along this space, and the positions in which the polygon's boundary encounters other points in S were the stable positions. The sweeping space is thus one-dimensional, since the circumference of the symmetric of P can be parameterized over a real interval. In the case of rigid motions, for each position of P during this sweep, one should consider all the rotations, described by an additional parameter that ranges from 0 to 2π.

When another point q_j in S enters the game, the rotation diagram will record all the regions in the parameters plane that correspond to rigid motions of P which cover both q_i and q_j. It is shown that the boundaries of these regions have a simple geometric structure.

The algorithm computes a region of maximum depth in the rotation diagram associated to point q_i. This region corresponds to a GSP of P of maximal containment that contains q_i on the boundary. Clearly, by computing the GSP of P of maximal containment with respect to every point in S and taking the best one, a solution of Problem 6.1.1 is detected.

The algorithm avoids taking into account points that are too distant to be contained in a GSP of P by using a *bucketing* approach. In this way, the working time of the algorithm becomes output-sensitive, the total number of operations being of order $O(n^2km^2\log(nm))$.

6.2 Rotation Diagrams

A slightly modified version of Lemma 3.4.1 is used:

Lemma 6.2.1. *Let P be a convex polygon of vertices $\{p_1,\ldots,p_m\}$, q_1,q_2 points in plane. A point x on $\partial\tau_{q_1-p_1}(P)$ is a point of intersection between $\partial\tau_{q_1-p_1}(P)$ and $\partial\tau_{q_2-p_1}(P)$ if and only if q_1 is on $\partial\tau_{q_2-x}(\tau_{q_1-p_1}(P))$.*

Proof. Assume first that $x\in\partial\tau_{q_1-p_1}(P)$. Then, since translation τ_{q_2-x} translates x to q_2, the point q_2 will be on $\partial\tau_{q_2-x}(\tau_{q_1-p_1}(P))$ (see Fig. 6.1).

Point q_1 is also on $\partial\tau_{q_2-x}(\tau_{q_1-p_1}(P))$, because $\tau_{q_2-x}\circ\tau_{q_1-p_1}=\tau_{q_1-x}\circ\tau_{q_2-p_1}$, and:

$$\begin{aligned}
q_1 &= \tau_{q_2-x}(\tau_{q_1-p_1}(p_1)+(x-q_2)\\
&= (p_1+(q_1-p_1)+(q_2-x))+(x-q_2)\\
&= (p_1+(q_2-p_1)+(x-q_2))+(q_1-x)\\
&= \tau_{q_1-x}(x).
\end{aligned}$$

The inverse implication is immediate, by a similar argument. ∎

As a consequence of this lemma, points on $\partial\tau_{q_1-p_1}(P)$ can be used to represent TSP's of P with q_1 and another point in S on the boundary.

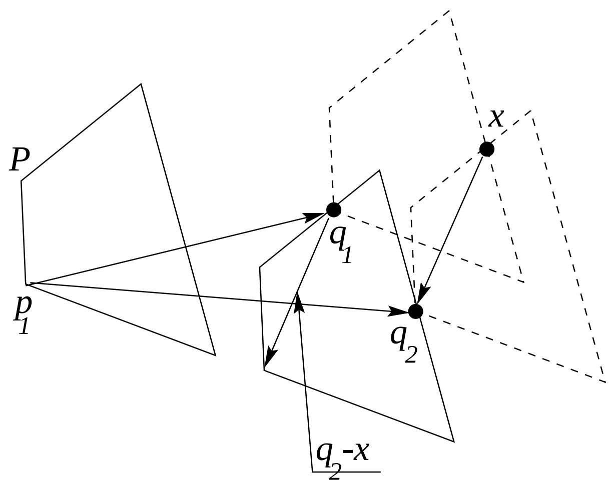

Figure 6.1: Another approach for finding a TSP of P with respect to q_1 and q_2.

Since there are two intersection points of $\partial\tau_{q_1-p_1}(P)$ and $\tau_{q_2-p_1}(P)$, these intersections determine on the boundary of $\tau_{q_1-p_1}(P)$ two disjoint components. Points in one component correspond to translations of P anchored on q_1 and containing q_2, whereas points in the other component correspond to positions of P anchored on q_1 that do not contain q_2.

The following lemma is a generalization of Lemma 6.2.1:

Lemma 6.2.2. *Let P be a convex polygon of vertices $\{p_1, \ldots, p_m\}$, q_1, q_2 points in plane. A point x is in $(\tau_{q_1-p_1}(P) \cap \tau_{q_2-p_1}(P))$ iff $\tau_{q_2-x}(\tau_{q_1-p_1}(P))$ contains both q_1 and q_2.*

These lemmas can be used to define a slightly modified variant of Algorithm `Max-2Dcontainment-By-Anchored-Sweep` (Fig. 5.4). Instead of sweeping along the boundary of a translated copy of $-P$, this algorithm sweeps the boundary of $\tau_{q_i-p_1}(P)$. The intersections points of $\tau_{q_i-p_1}(P)$ with translates of P with p_1 in other points in S determine of the boundary of the first one an arrangement of regions. The algorithm searches a region of maximum depth in this arrangement by applying a one-dimensional sweep. The events in this sweep are precisely the intersections of two boundaries.

Now, the problem in which both translations and rotations are allowed cannot use this method as it is, because every translation can be composed with a rotation of any angle. However, the set of all rotations can be discretized by partitioning it into a finite set of classes, which are equivalent in some sense. In other words, for a finite set of rotations of P around its vertex p_1, a similar algorithm is used to perform an anchored sweep of the rotated polygon.

These rotations constitutes landmarks in the bidimensional parameter space of all possible rigid motions of P. A plane-sweep method can be applied over this space, in which these landmarks are the events.

For each point $q_i \in S$, the rigid motion ρ such that q_i is on $\partial\rho(P)$, and $\rho(P)$ contains a maximal number of S points, is sought. The search space is, thus, bidimensional: On the horizontal axis the angles of rotation are represented, ranging from 1 to 2π. On the vertical axis the boundary of P is represented, ranging from 0 to the perimeter of P. Both axes wrap around, 0 and 2π, as well as 0 and the length of the boundary of P, are identified.

A point (θ, x) in this space corresponds to the rigid motion composed by a rotation of P around p_1 with θ degrees and a translation that puts the point at position t on the boundary of P to q_i.

This space is the *rotation diagram* associated to q_i.

Definition 6.2.1. (Rotation Diagram) *The rotation diagram of polygon P associated to point q, denoted $R_{p,q}$, is the space of all rigid motions η that can be written as $\tau \circ \rho$, where ρ is a rotation around p_1, and τ a translation with the property that $\tau(t) = q$, for a point t on the boundary of P.*

Let q_j be another point in S, and let θ be an angle, $0 \leq theta < 2\pi$; denote by P_θ the polygon $\rho_\theta(P)$, obtained by rotating P around p_1 with angle θ. By Lemma 6.2.1, there are at most two points t_1 and t_2 on ∂P_θ so that $\tau_{t_1-p_1}(P_\theta)$ and $\tau_{t_2-p_1}(P_\theta)$ contain both q_1 and q_2 on the boundary. In R_P, these two points correspond to two points on the vertical line of abscissa θ.

By Lemma 6.2.2, all points on the vertical at θ that lie between t_1 and t_2 correspond to translations of P_θ that keep the polygon anchored at q_i and covering q_j. It remains to study the shape of the figures obtained by allowing θ to vary.

Intuitively, these regions can be obtained by translating P with p_1 at q_i and q_j and rotating the two polygons with the same angles. By Lemma 6.2.1, at any given angle of rotation, the intersections between the two polygons determine the translations of P which would place both q_1 and q_2 on the boundary of P.

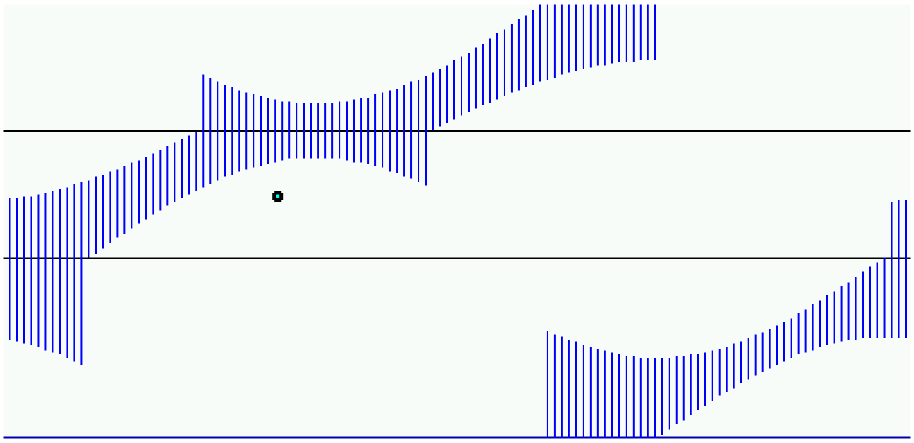

Figure 6.2: Containment region of point q_2 in the rotation diagram of triangle P associated to point q_1. The scene is shown in Figure 6.3.

During this rotation, the corresponding translations trace out curves in the rotation diagram R_{P,q_1}. By Lemma 6.2.2, these curves bound a (not necessarily connected) *containing region* in R_{P,q_1}, corresponding to all placements of P that contain q_2 and have q_1 on the boundary. Figure 6.2 shows the containing region of q_2 in the rotation diagram R_{P,q_1}; the configuration of P, q_1 and q_2 is presented in Figure 6.3. The containing region has been generated by a computer program. The point marked in Figure 6.2 corresponds to the particular position depicted in Figure 6.3. Horizontal lines in the rotation diagram mark the positions of vertices of P on its perimeter.

The containment region of point q_j in R_{P,q_i} will be denoted A_{q_j}.

For almost all angles θ, the rotated polygon P_θ has exactly two TSP's with respect to q_i and q_j, revealed in R_{P,q_i} by the endpoints of the intersection of A_{q_j} with the vertical through θ. There are some particular angles for which the continuity of curves described by these endpoints is broken. For this angles, called *critical*, there exists a continuous range of translations that keep P_θ in critical position: both q_i and q_j lie on a line parallel to an edge e of P_θ, and thus there are many translations that keep P_θ with q_i and q_j on e.

Another category of angles which spoil the regularity of the curves are those at which the intersection between the two boundaries is a vertex of

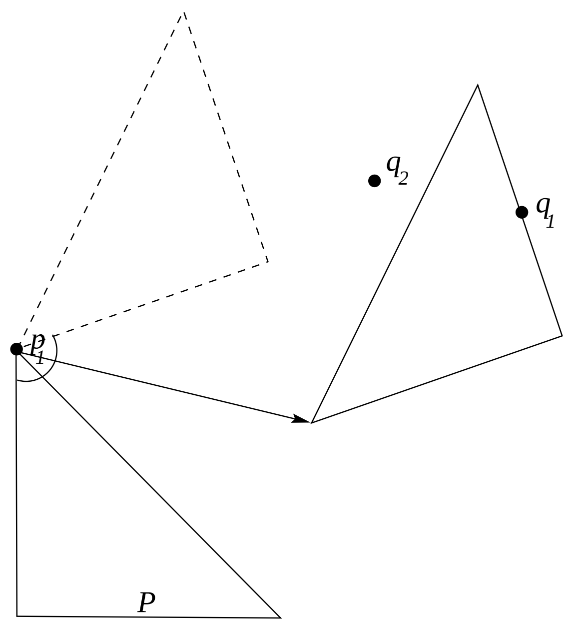

Figure 6.3: Triangle P and points q_1, q_2 that correspond to the rotation diagram presented in Figure 6.2.

one of the polygons. In this case, the *smoothness* of the curve is broken: two different edges, with different slopes, are adjacent to the vertex involved in such a situation, and their slope, as will be seen, determine a different variation law. The angles with these property will be also called critical.

Critical angles decompose therefore the containment region into smaller parts, which, hopefully, have a simple mathematical description. Indeed, between these critical angles, the equation of the curves described by the two endpoints is of the form stated in the following lemma:

Lemma 6.2.3. *The upper and lower curves bounding the decomposed regions of A_{q_2} are sine curves of the form $c_1 + c_2 \sin(\theta + c_3)$*

Proof. The upper and lower curves are generated by rotating in tandem two polygons, $P_i := \tau_{q_i - p_1}(P)$ and $P_j := \tau_{q_j - p_1}(P)$, around q_i and respectively q_j.
Claim. *During this rotation, the movement of P_2 relative to P_1 is the translation of P_2 on a circle around P_1.*
Proof of the claim. The two polygons keep the same relative orientation during the rotation in tandem: for every angle θ, $\rho_\theta(P_j) = \tau_{q_j - q_i}(\rho_\theta(P_i))$.

Moreover, for every vertex of P_i, the distance to the corresponding vertex of P_j is constant, equal to $\mathrm{dist}(q_i, q_j)$. This means that the relative position of $\rho_\theta(P_i)$ against $\rho_\theta(P_i)$ is given by the relative position of only one point of $\rho_\theta(P_j)$. Therefore it would be enough to study the dependency of the coordinates of q_j in the system of coordinates associated to P_i.

It will be easier to work with polar coordinates, the origin at q_i. The coordinates of q_j are of the form (r, φ), with $r = \mathrm{dist}(q_i, q_j)$, and φ the angle made by the ray through q_j emanating from the origin with the horizontal.

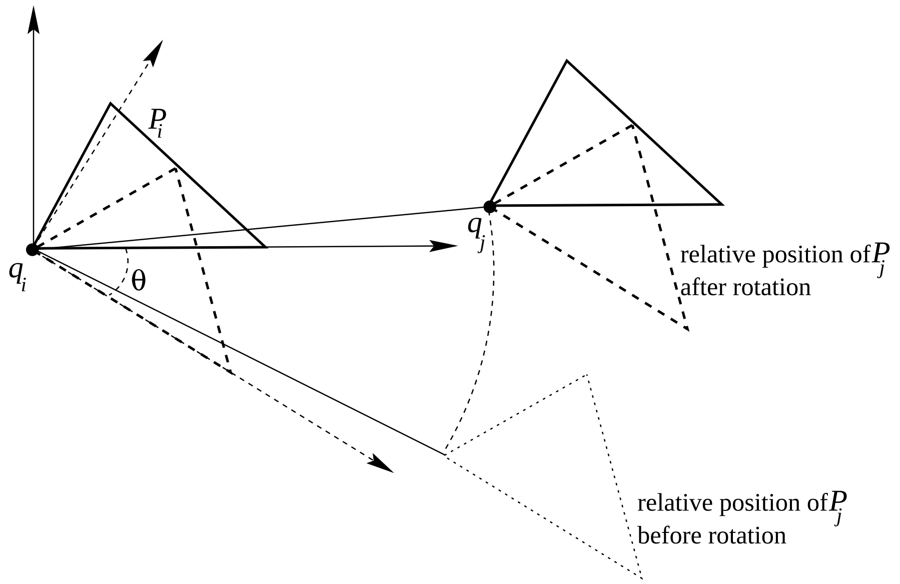

Figure 6.4: Positions of a polygon relative to the other, when both are rotated in tandem with the same angle.

If the system of coordinates rotates with angle θ, the angle φ will change into $\varphi - \theta$ (see Fig. 6.4). The distance r remains the same. This means that the positions taken by q_j during the rotation of the system of coordinates are on a circle of radius r. ■

Proof of Lemma 6.2.3 (continuation). It remains to study the variation of the intersection points of the boundaries of two polygons, when one of them if translated along a circle around the other.

The boundary of polygon P_i is considered parametrized. A good parameterization associates to each point p on ∂P_i the distance from p to p_1, along the boundary. Thus, if point p lies on the edge of endpoints p_k, p_{k+1}, for $1 \leq k \leq m$, then the parameterization of p is

$$\text{dist}(p_1, p_2) + \text{dist}(p_2, p_3) + \cdots + \text{dist}(p_{k-1}, p_k)\text{dist}(p_k, p).$$

What is needed now is an expression that allows writing (parametrically) a point of intersection of P_i and P_j, p_{int}, as a function of θ. Clearly, for angles θ for which the intersection point is a vertex of either one of the polygons, the expression changes. Between such angles, the function is continuous.

Let now e_1 be an edge of P_i that intersects e_2, an edge of P_j, at p.

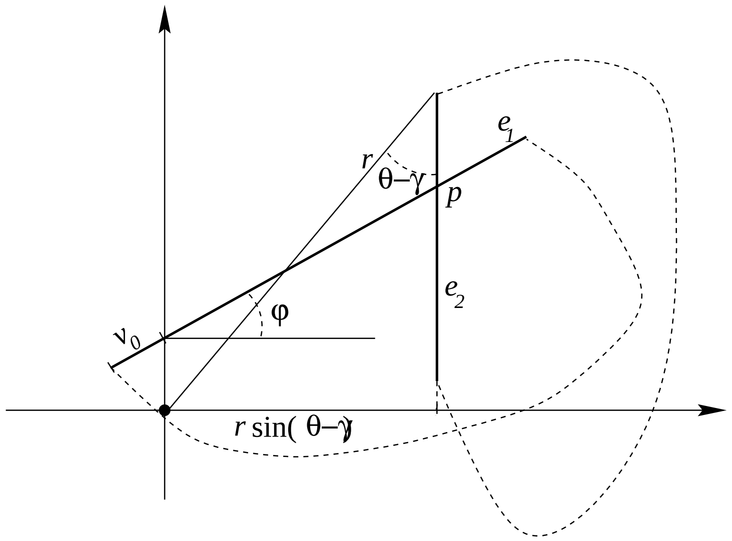

Figure 6.5: Variation of one of the intersection points of two polygons.

After rotating the whole plane with an angle γ around q_i, so that e_2 becomes vertical, the situation depicted in Figure 6.5 is reached. The parameterization of p is the sum of:

- the length l of all edges from p_1 to e_1;
- the length v_0 of the portion of e_1 that lies to the left of the y-axis;
- the length v of the portion of $e1$ between the y-cut and p.

During the translation of P_j along a circle around q_i, the only thing that varies is v. Now this value depends on the angle $\theta - gamma$, and the relationship is given by

$$v = \frac{r}{\cos(\phi)} \cdot \sin(\theta - \gamma),$$

that is, of the form stated in the Lemma. ∎

Next question would be how broad the decomposed regions can be; the answer is given by the following Lemma:

Lemma 6.2.4. *Each decomposed region is at most π wide.*

Proof. The bounds of a decomposed region are given by events that happen during the translation of P_2 along a circle around q_1. These events can be:

- a vertex of P_2 sweeps through an edge of P_1;
- an edge of P_2 passes over a vertex of P_1.

In both cases, between events, the elements of P_2 describe circular arcs around q_1. Such an arc should be entirely inside P_1, therefore its length is bounded by the maximum angle that a convex polygon can have: this angle cannot be greater than π. ∎

It remains to give an upper bound for the number of decomposed regions of A_{q_j}.

Lemma 6.2.5. *A_{q_j} is decomposed into a most $O(m^2)$ regions.*

Proof. Polygon P has m vertices and edges. Each vertex can sweep through an edge of the other polygon at most twice. As there are m^2 combinations edge-vertex and vertex-edge, the number of critical angles, and thus the number of decomposed regions of A_{q_j}, is $O(m^2)$. ∎

The bound given above is tight; an example that achieves this bound is not difficult to give.

So far, the properties of the containment region of a point in the rotation diagram associated to $q_i \in S$ have been enumerated. These properties are used both in defining the algorithm and in its analysis.

In fact, the rotation diagram associated to point q_i will maintain all containment regions for the other points in S – points which are close enough to q_i, so that these regions are not empty. All these containment regions together form an arrangement of shapes in the plane. Recall that the points in the containment region of q_j correspond to rigid motions of P that keep q_i on the boundary, and q_j in the interior or on the boundary. This means that the more containment regions a point p belongs to, the more S points are covered by the rigid motion of P corresponding to p.

Therefore, a solution for the problem:

Problem 6.2.1. *Find a point of maximum depth in the arrangement of containment regions in the rotation diagram associated to a point $q_i \in S$ and polygon P.*

is a possible candidate for a solution of Problem 6.1.1. Indeed, if one takes a point of maximum depth among solutions of Problem 6.2.1, for rotation diagram associated to all points in S, then one gets a solution for Problem 6.1.1.

6.3 The Algorithm

Figure 6.6 presents the algorithm for the solution to Problem 6.1.1.

As in the purely translational case, the points can be first preprocessed using a bucketing technique, so that the algorithm will not attempt to compute containment regions for points q_j that are too distant from q_i, in the rotation diagram associated to q_i. In other words, for points q_i in a certain bucket, only points from the *neighborhood* B_i of that bucket is considered, i.e., points from the eight neighboring buckets, plus the bucket itself.

The following paragraphs detail the algorithm, namely lines 6 and 8.

Building the rotation diagram associated to point $q_i \in S$ means constructing the containment regions for all the other (at most) $n-1$ points in the neighborhood of the bucket to which q_i belongs.

As was explained above, for another point q_j, the containment region A_{q_i} consists of at most $O(m^2)$ sub-regions, bounded horizontally by verticals through critical angles, and vertically by sine curves. The horizontal broadness of such a sub-region is smaller than π.

Input. a convex polygon P, given as a list of its vertices $\{p_1, p_2, \ldots, p_m\}$;
a set of points S;
Output. a rigid motion given as $(\mathbf{v}, \theta)$:
the composition of a rotation of angle θ around p_1
and a translation of vector $\mathbf{v}$;
begin
preprocess points into buckets;
set max := 0;
for each point $q_i \in S$ **do**
initialize R_{P,q_i}, the rotation diagram for P and q_i;
for each $j \neq i$ and $q_j \in B_i$ **do**
compute the containment region A_{q_j} in R_{P,q_i};
end for
use line sweep to compute point x of maximum depth d
in the arrangement of all containing regions A_{q_j};
if $d >$ max
then set max := d; store x;
end if
end for
compute $\mathbf{v}$ as the point on ∂P
parameterized by the second coordinate of x;
return ($\mathbf{v}$, the first coordinate of x);
end

Figure 6.6: Algorithm `Rigid-motion-of-maximum-containment`.

This means that generating a containment region means computing the critical angles and the parameters of the sine curves between two consecutive critical angles.

A critical angle corresponds to a rotation of $\tau_{q_i - p_1}(P)$ and $\tau_{q_j - p_1}(P)$ that put them in a position in which a vertex on the boundary of one of them touches the boundary of the other. Now, rotating in tandem the two polygons is equivalent to keeping $\tau_{q_i - p_1}(P)$ fixed and translating $\tau_{q_j - p_1}(P)$ along a circle around q_1. The critical angles correspond to all positions of $P_{q_j - p_1}(P)$ at which a vertex of $\tau_{q_j - p_1}(P)$ touches an edge of $\tau_{q_i - p_1}(P)$, or a vertex of $\tau_{q_i - p_1}(P)$ touches an edge of $\tau_{q_j - p_1}(P)$. The angles corresponding to these particular position can be computed by intersecting circles with line segments; more precisely, for each vertex of $\tau_{q_j-p_1}(P)$, the intersections of its circle around q_1 with the edges of $\tau_{q_1-p_1}(P)$ are collected, and for each vertex of $\tau_{q_i-p_1}(P)$, the intersections of its circle around q_1 with every edge of $\tau_{q_j-p_1}(P)$ are collected.

This list of angles, sorted, gives the abscissae that decompose the containment region into sub-regions. For such a sub-region, the upper and lower bounding curves are sine curves. The parameters of these curves can be determined in constant time, by simple computation, using the only things known to the algorithm – coordinates of the intersecting edges.

From the efficiency point of view, the time spent for Line 6 is $O(m^2)$ – there are $m \times m$ pairs vertex-edge and edge-vertex to be checked, giving $O(m^2)$ critical angles; the number of decomposed regions are of the same order, and for each such region a constant number of operations are made for computing the parameters of the bounding curves.

In line 8 of the algorithm, a line-sweep is invoked, to seek a region of maximum depth in the arrangement of containment regions. Now, this is a rather peculiar situation, because the scene is made of regions bounded by sine curves, and not straight-line segments.

However, as the bounding curves of containment regions are x-monotone, a space-sweep technique can be used here. Critical angles give the events; another type of events are the intersections of the sine curves that bound the regions.

The event queue will be initialized with all critical angles in the rotation diagram – that is, critical angles for at most $n-1$ containment regions. This structure should provide fast insertion of new events – the intersections of sine curves, detected dynamically – and retrieval of the first element – a *priority queue* is appropriate. Its initialization, with the $O(nm^2)$ critical angles, requires sorting, and thus it is $O(nm^2 \log(mn))$.

The events that are dynamically generated do not exceed $O(n^2m^2)$. Indeed, For two containment regions A_{q_k} and A_{q_l}, the critical angles are sorted, and they determine $O(m^2)$ intervals. By Lemma 6.2.4, any such interval can be no greater than π. This means that, between two critical angles, the boundaries of A_{q_k} and A_{q_l} can intersect in at most a constant number of points.

Since there are $\binom{n-1}{2}$ pairs of containment regions in a rotations diagram, the number of intersection is $O(n^2m^2)$. Of this order is then the size of the queue of events; the time required for inserting a new event is $O(nm)$.

Using bucketing, the number of containment regions decreases. With the same argument used in Section 5.1, it can be shown that, if k is the number of points that are covered by an optimal placement of P, then, by using bucketing, $O(k)$ containment regions occur in any rotation diagram, and thus:

- the number of initial events is $O(nm^2)$;
- the number of intersections detected during the sweep is $O(k^2m^2$.

The line status consists of the intersections of the vertical line with the set of containment regions. At any angle, at most $n-1$ decomposing regions can be intersected, that is, at most $2(n-1)$ sine curves. The structure that stores the line status must allow fast access to any element, fast insertion and deletion, and also the possibility to switch places for two adjacent element. A tree of chains (see Section 4.1.3, page 42) can be used here. At any moment, the tree has at most $2(n-1)$ nodes ($O(k)$, when bucketing is used), therefore an insertion/deletion/switch of places takes $O(\log n)$ operations (respectively $O(\log k)$).

6.3.1 Analysis

- the preprocessing phase can be performed in time $O(n)$;
- the computation of a containment region, in line 6, is executed in time $O(m^2)$;
- for the line-sweep in line 8,
  - there are $O(k^2m^2)$ events;
  - for each event, $O(\log(km))$ time is spent for inserting the new intersections found into the queue, and $O(\log k)$ time is spent for updating the line status;
- the **for** loop in lines 5-10 works then in $O(k^2m^2\log(km))$ time, and
- the **for** loop in lines 3-11 works in $O(nk^2m^2\log(km))$ time.

The total running time is therefore $O(nk^2m^2\log(km))$.

Chapter 7

Appendices

7.1 On General Stable Placements in Plane

This appendix is dedicated to a discussion on the movement of a triangle with two vertices sliding on non-parallel lines. The facts presented here can be used for developping a symbolic-numeric algorithm for finding general stable placements.

Again, P will be a convex polygon, and $S = \{A, B, C\}$ a set of three points so that there are isometries ρ with $S \subset \rho(P)$.

Applying the algorithm for computing TSP's, a translation of P with two points in S on the boundary can be determined without dificulties. We will assume, from now on, that A and B are on two edges of P, say e_1 and e_2, not necessarily adjacent.

We further simplify the problem: we are to find a symbolic description of the trajectory of the point C, while A qnd B slide along e_1 and respectively e_2 – see Fig. 7.1.

Thus, let the points be $A(x_A, y_A), B(x_B, y_B), C(x_C, y_C)$, assume for simplicity that e_1 and e_2 are adjacent, and by a change of coordinates make their common endpoint be the origin, while e_1 is on the x-axis. A slides along e_1, therefore $y_A = 0$. B slides along e_2, therefore $y_B = m \cdot x_B$, with $m \neq 0$ a real number. Denote by a, b, c the lengths of the edges of the triangle ABC.

Let $f_1 = 0, f_2 = 0, f_3 = 0$ be the distance relation among the vertices,

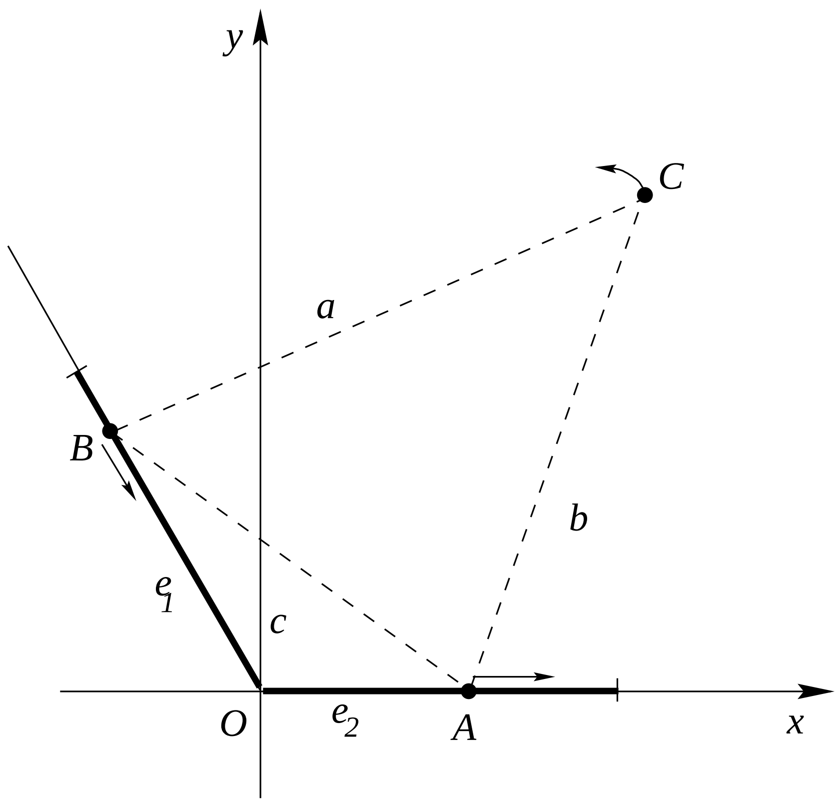

Figure 7.1: Grafical representation of the hypotheses of the problem of finding GSP's in the plane.

where f_1, f_2, f_3 have the following expressions:

$$
\begin{aligned}
f1 &= (x_B - x_A)^2 + m^2 x_B^2 - c^2;\\
f2 &= (x_B - x_C)^2 + (m x_B - y_C)^2 - a^2;\\
f3 &= (x_A - x_C)^2 + (y_C)^2 - b^2.
\end{aligned}
$$

We want first to find a relation between x_c and y_c – the implicit form of the curve described by C when A and B move along their corresponding admissible paths.

For this, we apply Buchberger's algorithm for computing Gröbner Basis (see [Buchberger85]), implemented in Maple by Frederik Chizak. The Gröbner Basis of the ideal generated by f_1, f_2, f_3, using an eliminating ordering with x_C and y_C with greatest weights, contains a set of 8 polynomials, from wich only one does not contain x_A and x_B:

$$
\begin{aligned}
g \;=\;& 16m^4b^4x_C^4 - 32m^3b^4x_C^3y_C - 32m^3b^2c^2x_C^3y_C + 32a^2m^3b^2x_C^3y_C \\
&-32a^2m^2c^2x_C^2y_C^2 + 16m^4b^4x_C^2y_C^2 + 16m^2b^4x_C^2y_C^2 - 32m^4b^2c^2x_C^2y_C^2 \\
&-32a^2m^2b^2x_C^2y_C^2 + 16m^4c^4x_C^2y_C^2 + 16a^4m^4x_C^2y_C^2 - 32a^2m^4c^2x_C^2y_C^2 \\
&+16a^4m^2x_C^2y_C^2 + 16m^2c^4x_C^2y_C^2 + 64m^2b^2c^2x_C^2y_C^2 - 32mc^4x_Cy_C^3 \\
&-32mb^2c^2x_Cy_C^3 - 32m^3c^4x_Cy_C^3 - 32m^3b^4x_Cy_C^3 + 32a^2m^3c^2x_Cy_C^3 \\
&+32a^2m^3b^2x_Cy_C^3 + 32a^2mc^2x_Cy_C^3 + 64m^3b^2c^2x_Cy_C^3 + 16a^4m^4y_C^4 \\
&+16m^2c^4y_C^4 + 16b^4m^2y_C^4 - 32b^2c^2m^2y_C^4 + 16c^4y_C^4 + 16a^4m^2y_C^4 \\
&-32a^2m^2b^2y_C^4 + 8a^4m^2b^2x_C^2 - 8a^4m^4b^2x_C^2 - 16a^2m^4b^4x_C^2 \\
&+16m^4b^4c^2x_C^2 + 8m^2b^6x_C^2 - 8m^4b^6x_C^2 + 16a^2m^4b^2c^2x_C^2 - 16a^2m^2b^4x_C^2 \\
&-8m^4b^2c^4x_C^2 - 16a^2m^2b^2c^2x_C^2 + 8m^2b^2c^4x_C^2 - 16m^2b^4c^2x_C^2 \\
&-24a^4mb^2x_Cy_C + 8a^6mx_Cy_C + 8a^6m^3x_Cy_C + 8mb^4c^2x_Cy_C \\
&+24a^2mb^4x_Cy_C + 24a^2m^3c^4x_Cy_C - 24a^4mc^2x_Cy_C + 24m^3b^6x_Cy_C \\
&-8a^2m^3b^4x_Cy_C - 24a^4m^3b^2x_Cy_C + 16a^2c^2mb^2x_Cy_C + 24a^2mc^4x_Cy_C \\
&-8mb^6x_Cy_C - 8c^6mx_Cy_C + 40m^3c^4b^2x_Cy_C - 56m^3b^4c^2x_Cy_C \\
&-16a^2c^2m^3b^2x_Cy_C - 24a^4m^3c^2x_Cy_C - 8c^6m^3x_Cy_C + 8mb^2c^4x_Cy_C \\
&-8a^6m^2y_C^2 - 32m^2b^2c^4y_C^2 + 8c^6y_C^2 + 8a^4c^2y_C^2 + 16a^4c^2m^4y_C^2 \\
&+24a^2b^4m^2y_C^2 + 8b^4c^2y_C^2 - 24a^2m^2c^4y_C^2 + 16a^2b^2m^4c^2y_C^2 - 8a^6m^4y_C^2 \\
&-16b^2c^4y_C^2 - 16a^4m^4b^2y_C^2 - 8a^2b^4m^4y_C^2 - 16b^6m^2y_C^2 + 24a^4m^2c^2y_C^2 \\
&-16a^2c^4y_C^2 - 8a^2m^4c^4y_C^2 + 8m^2c^6y_C^2 - 16a^2b^2c^2y_C^2 + 40b^4c^2m^2y_C^2 \\
&-4a^2b^6 - 4a^2c^6m^4 - 4a^6m^4c^2 - 4c^6b^2 + m^4c^8 - 8a^2c^6m^2 + b^8 \\
&+6c^4m^4a^4 + 4a^2c^4b^2 + 4a^2c^2b^4 + c^8 - 8b^6m^2c^2 - 4m^4b^6c^2 + 6m^4c^4b^4 \\
&+m^4b^8 + 2m^2b^8 + 6c^4b^4 + a^8 - 4c^2b^6 - 4m^4c^6b^2 + 12b^4c^4m^2 \\
&-8a^6m^2c^2 - 8a^6m^2c^2 + 6a^4b^4 - 4a^2c^6 - 4a^6c^2 + 6b^4m^4a^4 + 4b^6m^4a^2 \\
&+12c^4a^4m^2 - 8c^6m^2b^2 + 12c^4m^4a^2b^2 - 12c^2m^4a^4b^2 - 12c^2m^4a^2b^4 \\
&+16c^4a^2m^2b^2 + 2c^8m^2 + a^8m^4 + 4a^4c^2b^2 + 4a^6b^2m^4 + 2a^8m^2 \\
&-8b^4a^2m^2c^2 - 4a^6b^2 + 6a^4c^4 - 8b^2a^4m^2c^2 - 4b^4m^2a^4.
\end{aligned}
$$

If we instantiate the parameters with, for instance, $m = a = b = c = 1$, we get:

$$g_{1,1,1,1} = 1 - 4y_C^2 - 8x_C^2 + 4x_Cy_C + 8x_C^2y_C^2 + 4x_C^4$$

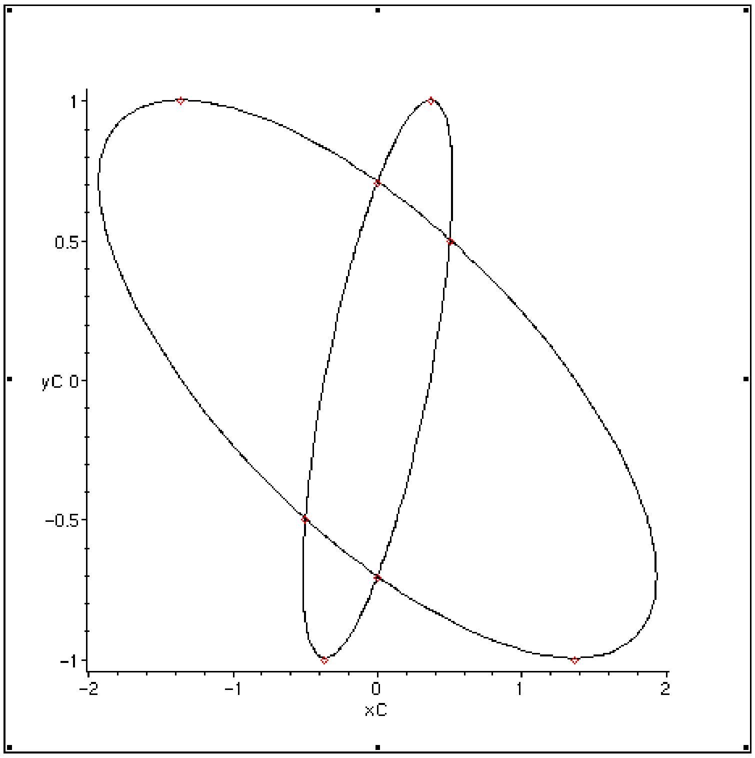

Figure 7.2: Trajectory of the third vertex of a triangle, when the other two vertices slide along two straight lines.

$$-8x_C y_C^3 + 8x_C^3 y_C + 4y_C^4.$$

The package **CASA** ([CASA]) gives a nice plot of the algebraic set $g_{1,1,1,1} = 0$ – Figure 7.2.

This figure suggests that $g_{1,1,1,1}$ is the product of two second-degree polynomials. Indeed, it can be factored over the field of real algebraic numbers. Each factor contributes to the variety with one ellipse. However, for our purposes only one ellipse must be considered, because the other is drawn by the third vertex of a symmetric of the initial triangle.

Therefore, the GSP's of a triple of points with respect to a triangle can be found by computing symbolically the trajectory of a point when the other two points slide along two sides of the triangle, and by intersecting this trajectory with the third edge of the triangle. The output will consist in at most two points – the intersection of a straight line with an ellipse.

Bibliography

[AgarwalAmentaSharir96] P. K. Agarwal, N. Amenta, M. Sharir - *Largest Placement of One Convex Polygon inside Another*, to appear in Discr. Comp. Geom.

[AhoHU83] A.Aho, J.Hopcroft, J.Ullmann - *Data structures and alorithms*, Springer-Verlag, 1983.

[AltHsuSnoeyink97] H. Alt, D. Hsu, and J. Snoeyink - *Computing the largest inscribed isothetic rectangle*, accepted to CGTA 1997.

[BarequetDP97] G.Barequet, M.Dickerson, P.Pau - *Translating a convex polygon to cover a maximum number of points*, Computational Geometry 8 (1997), 167-179.

[BenOr83] M. Ben-Or - *Lower bounds for algebraic computation trees*, in Proc. 15th Ann. ACM Sympos. Teory Comput. 1983, 80-86.

[Bentley77] J.L. Bentley - *Algorithms for Klee's Rectangle Problem*, Carnegie- Mellon Univ., Pittsburgh, Pennsilvania, Dept. of Comp. Sci., unpublished notes, 1977.

[BentleyOttmann78] J.L. Bentley, T. Ottmann - *Algorithms for Reporting and Counting Geometric Intersections*, IEEE Trans. on Computers, vol. C-28, no. 9, 1979.

[BentleyShamos78] J.L. Bentley, M.I. Shamos - *Divide and conquer for linear expected time*, Inform. Proc. Lett. 7(1978), 87-91.

[BarequetADG97] G.Barequet, A.Briggs, M.Dickerson, M.Goodrich - *Offset-Polygon Annulus Placement Problems*, LNCS 1272, Springer 1997, pp. 378-391.

[Buchberger85] B. Buchberger - *Gr¨ obner Bases: An Algorithmic MEthod in Polynomial Ideal Theory*, in Recent Trends in Multidimensional System Theory, chapter 6, pp. 184-232, D. Reidel, Dordrecht, 1985.

[CASA] F. Winkler *Constructive Algebraic Geometry with CASA*, RISC Report 93-26, Research Inst. for Symb. Comp., Johannes Kepler Univ., Linz, Austria, 1993.

[ChandKapur70] D.R. Chand, S.S. Kapur - *An algorithm for convex polytopes*, J. Assoc. Comput. Mach. 17 (1970), 78-86.

[ChazelleEdelsbrunner92] B. Chazelle, H. Edelsbrunner - *An Optimal Algorithm for Intersecting Line Segments in the Plane*, J. of ACM, vol. 39, no. 1, jan. 1992. pp. 1-54.

[Chazelle83] B. Chazelle - *The Polygon Placement Problem*, Adv. in Comp. Res. vol. 1, JAI Press 1983, pp. 1-34.

[Chazelle92] B. Chazelle *An optimal algorithm for intersecting three-dimensional convex polyhedra*, SIAM J. Comp. 21(4):671, 1992.

[ChazelleLee86] B. Chazelle, and D.T. Lee - *On a Circle Placement Problem, Computing* **36** (1986) 1–16.

[ClarksonShor89] K. Clarkson, P. Shor - *Applications of Random Sampling in Computational Geometry*, II, Discrete Comp. Geom. 4 (1989), 387-421.

[CormenLR90] T. Cormen, C. Leiserson, R. Rivest - *Introduction to Algorithms*, MIT Press, 1990.

[DickersonScharstein96] M.Dickerson and D. Scharstein, *Optimal Placement of Convex Polygons to Maximize Point Containment*, Discrete and Computational Geometry 11 (1998) 1–16.

[DobkinLipton76] D.P.Dobkin, R. Lipton - *Multidimensional searching problems*, SIAM J. Comp. 5(2), 181-186, 1976.

[DobkinKirkpatrick83] D.P.Dobkin, D. G. Kirkpatrick - *Fast Detection of Polyhedral Intersection*, Theoret. Computer Sci. 27 (1983), pp. 241-253.

[DobkinKirkpatrick85] D.P.Dobkin, D. G. Kirkpatrick - *A Linear Algorithm for Determining the Separation of Convex Polhedra*, J. of Alg. 6 (1985), pp. 381-392.

[Edelsbrunner87] H. Edelsbrunner - *Algorithms in Combinatorial Geometry*, Springer Verlag, 1987.

[EppsteinErickson] D. Eppstein and J. Erickson - *Iterated Nearest Neighbors and Finding Minimal Polytopes*, Discr. Comp. Geom. 11 321-350.

[FisherHöffgen94] P. Fisher, K.-U. Höffgen - *Computing a maximum axis-aligned rectangle in a convex polygon*, Inf. Proc. Letters 51, pp. 189-194, 1994.

[FreiseisenPau98] W. Freiseisen, P.Pau - *A Generic Plane-Sweep for Intersecting Line Segments*, RISC Report no. 98-18, Research Inst. for Symb. Comp., Johannes Kepler Univ., Linz, Austria, 1998.

[GaertnerSchoenherr97] B. Gärtner, S. Schönherr - *Smallest Enclosing Ellipses - Fast and Exact*, Institut für Informatik, Freie Universität Berlin, Techn. Rep. B 97-03, 1997.

[Graham72] R.L. Graham - *An efficient algorithm for determining the convex hull of a finite planar set*, Inf. Proc. Lett. 1 (1972), 132-133.

[HorowitzSahni-d82] E.Horowitz, S.Sahni - *Fundamentals of Data Structures*, Computer Science Press 1982.

[HorowitzSahni-a82] E.Horowitz, S.Sahni - *Fundamentals of Computer Algorithms*, Computer Science Press 1984.

[Jarvis73] R.A. Jarvis - *On the identification of the convex hull of a finite set of points in the plane*, Inv. Proc. Lett.2 (1973), 18-21.

[Kedem86] K. Kedem, R. Livne, J. Pach, M. Sharir - *On the union of Jordan regions and colision-free translational motion amidst plygonal obstacles*, Discr. Comp. Geom., 1 (1986), 59-71.

[KirkSeidel86] D.G. Kirkpatrick, R. Seidel - *The ultimate planar convex hull algorithm?*, Siam J. Comput. 12 (1983), 28-35.

[KirkSnoeyink95] D. Kirkpatrick, J. Snoeyink - *Tentative Prune-and-Search for Computing Fixed-Points with Applications to Geometric Comutation*, J. *Fundamenta Informaticæ*, 22(4):353–370, 1995.

[LeeuwenWood81] J. van Leeuwen, D. Wood - *The measure problem for rectangular ranges in d-space*, J. Algorithms 2, 282-300 (1981).

[Megiddo83] N. Megiddo - *Linear-time algoriths for linar programming in* R^3 *and related problems*, Siam J. Comput., Vol 12, No. 4, Nov. 1983, pp.759-774.

[MullerPreparata78] D. E. Muller, F. P. Preparate - *Finding the Intersection of Two Convex Polyhedra*, Theoretical Computer Science 7(2), 217-236, 1978.

[NievergeltPreparata82] J. Nievergelt, F. Preparata - *Plane-sweep algorithms for itersecting geometric figures*, Comm. ACM 25(10), pp. 739-747 (1982).

[MairsonStolfi83] H.G. Mairson, J. Stolfi - *Reporting and Couning Intersections Between Two Sets of Line Segments*, Proc. NATO Adv. Study Inst. on Theoret. Found. Comput. Graphics and CAD, Springer-Verlag, 1987.

[ORourkeAg86] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin *An optimal algorithm for finding minimal enclosing triangles*, J. Algorithms **7** (1986) 258–269.

[ORourkeCh82] J. O'Rourke, C.-B. Chine, T. Olson and D. Naddor, *A new linear algorithm for intersecting convex polygons*, Comp. Graphics and Image Processing 19, 384-391 (1982).

[PalazziSnoeyink94] L. Palazzi, J. Snoeyink - *Counting and Reporting Red/ Blue Segments Intersections*, CVGIP: Graphical Models and Image Processing vol 56, no. 4, pp. 304-310, 1994.

[Pau99a] P. Pau - *On the complexity of an optimal placement problem*, to appear in Annals of the University of Timişoara.

[Pau99b] P.Pau - *Translating Convex Polyhedra to Contain a Maximum Number of Points*, to appear in Annals of the University of Timişoara, Romania.

[ORourke85] J. O'Rourke *Finding minimal enclosing boxes*, International J. of Computer and Information Sciences, Vol. 14, No. 3, June 1985.

[PH77] F.P. Preparata, S.J. Hong - *Convex hulls of finite sets of points in two and three dimensions*, Comm. ACM 20 (1977), 87-93.

[PreparataShamos85] F.P. Preparata and M.I. Shamos *Computational Geometry: an Introduction*, Springer-Verlag (1985).

[Seidel81] R. Seidel - *A convex hull algorithm optimal for point sets in even dimensions*, Rep. 81-14, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, 1981.

[Seidel85] R. Seidel - *A method for proving lower bounds for certain geometric problems*, in Comput. Geom., G.T.Toussaint, ed. (1985), 319-334, North-Holland, Amsterdam.

[Shamos78] M.I. Shamos - *Computational Geometry* - PhD. thesis, Dept. Comput. Sci, Yale Univ., New Haven, Conn., 1978.

[ShamosHoey76] M.I. Shamos, D.J. Hoey - *Geometric Intersection Problems*, Proc. 17'th Annu. Conf. Found. of Comp. Sci., pp. 208-215, Oct. 1976.

[Sharir91] M. Sharir - *On k-sets in arrangements of curves and surfaces*, Discr. Comp. Geom. 6 (1991), 593-613.

[SteeleYao82] J.M. Steele, A.C. Yao - *Lower bound for algebraic decision trees*, J. Algorithms 3 (1982), 1-8.

[Toussaint83] G.T. Toussaint *Solving geometric problems with the rotating calipers*, Proc. IEEE MELECON '83 Athens, Greece (1983).

[Yao81] A. C. Yao - *A lower bound for finding convex hulls*, J. assoc. Comput. Mach. 28 (1981), 780-787.