Computational Geometry*

D. T. Lee
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
e-mail: dtlee@ece.nwu.edu
+1-847-467-4144 (FAX)
+1-847-491-5007

1 Introduction

Computational geometry evolves from the classical discipline of design and analysis of algorithms, and has received a great deal of attention in the last two decades since its inception in 1975 by M. Shamos[108]. It is concerned with the computational complexity of geometric problems that arise in various disciplines such as pattern recognition, computer graphics, computer vision, robotics, VLSI layout, operations research, statistics, etc. In contrast with the classical approach to proving mathematical theorems about geometry-related problems, this discipline emphasizes the computational aspect of these problems and attempts to exploit the underlying geometric properties possible, *e.g.*, the metric space, to derive efficient algorithmic solutions.

The classical theorem, for instance, that a set S is convex if and only if for any $0 \leq \alpha \leq 1$ the convex combination $\alpha p + (1-\alpha)q = r$ is in S for any pair of elements $p, q \in S$, is very fundamental in establishing convexity of a set. In geometric terms, a body S in the Euclidean space is convex if and only if the line segment joining any two points in S lies totally in S. But this theorem *per se* is not suitable for computational purposes as there are infinitely many possible pairs of points to be considered. However, other properties of convexity can be utilized to yield an algorithm. Consider the following problem. Given a simple closed Jordan polygonal curve, determine if the interior region enclosed by the curve is convex. This problem can be readily solved by observing that if the line segments defined by all pairs of vertices of the polygonal curve, $\overline{v_i, v_j}, i \neq j, 1 \leq i, j \leq n$, where n denotes the total number of vertices, lie totally inside the region, then the region is convex. This would yield a straightforward algorithm with time complexity $O(n^3)$, as there are $O(n^2)$ line segments, and to test if each line segment lies totally in the region takes $O(n)$ time by comparing it against every polygonal segment. As we shall show below, this problem can be solved in $O(n)$ time by utilizing other geometric properties.

At this point, an acute reader might have come up with an $O(n)$ algorithm by making the observation: Since the interior angle of each vertex must be strictly less than π in order for the region to be convex, we just have to check for every consecutive three vertices v_{i-1}, v_i, v_{i+1} that the angle at vertex v_i is less than π. (A vertex whose internal angle has

*To appear in CRC Handbook of Computer Science and Engineering 1996.

a measure less than π is said to be *convex*; otherwise, it is said to be *reflex*.) One may just be content with this solution. Mathematically speaking, this solution is fine and indeed runs in $O(n)$ time. The problem is that the algorithm implemented in this straightforward manner without care may produce an incorrect answer when the input polygonal curve is *ill-formed*. That is, if the input polygonal curve is not *simple*, *i.e.,* it self-intersects, then the *enclosed* region by this closed curve is not well defined. The algorithm, without checking this simplicity condition, may produce a wrong answer. Note that the above observation that all the vertices must be *convex* in order to have a convex region is only a *necessary* condition. Only when the input polygonal curve is *verified* to be simple will the algorithm produce a correct answer. But to verify if the input polygonal curve self-intersects or not is no longer as straightforward. The fact that we are dealing with computer solutions to geometric problems may make the task of designing an algorithm and proving its correctness nontrivial.

An objective of this discipline in the theoretical context is to prove lower bounds of the complexity of geometric problems, and to devise algorithms (giving upper bounds) whose complexity *matches* the lower bounds. That is, we are interested in the *intrinsic* difficulty of geometric computational problems under a certain computation model, and at the same time are concerned with the algorithmic solutions that are provably optimal in the worst or average case. In this regard, the asymptotic time (or space) complexity of an algorithm is of interest. Due to its applications to various science and engineering related disciplines, researchers in this field have begun to address the *efficacy* of the algorithms, the issues concerning *robustness* and *numerical stability*[62], and the actual running times of their implementions.

In this chapter we concentrate mostly on the theoretical development of this field in the context of sequential computation. Parallel computation geometry is beyond the scope of this chapter. We will adopt the *real* RAM (Random Access Machine) model of computation in which all arithmetic operations, comparisons, kth-root, exponential or logarithmic functions take unit time. For more details refer to [56, 102, 108]. We first begin with a summary of problem solving techniques that have been developed[83, 105, 126], and then discuss a number of topics that are central to this field, along with additional references in the literature for further reading of these topics.

2 Problem Solving Techniques

We give an example for each of the eight major problem solving paradigms that are prevalent in this field. In subsequent sections we make reference to these techniques whenever appropriate.

2.1 Incremental Construction

This is the simplest and most intuitive method, also known as *iterative method*. That is, we compute the solution in an iterative manner by considering the input incrementally.

Consider the problem of computing the line arrangements in the plane. Given is a set $\mathcal{L}$ of n straight lines in the plane, and we want to compute the partition of the plane induced by $\mathcal{L}$. One obvious approach is to compute the partition iteratively by considering one line at a time[41]. As shown in Fig. 1, when line i is inserted, we need to traverse the regions that are intersected by the line and construct the new partition at the same time. One can show that the traversal and re-partitioning of the intersected regions can be done in $O(n)$ time per insertion, resulting in a total of $O(n^2)$ time. This algorithm is asymptotically optimal, since the running time is proportional to the amount of space required to represent the partition. This incremental approach also generalizes to higher dimensions. We conclude with the theorem[59].

Theorem 1 *The problem of computing the arrangement $\mathcal{A}(H)$ of a set H of n hyperplanes in $\Re^k$ can be solved iteratively in $O(n^k)$ time and space, which is optimal.*

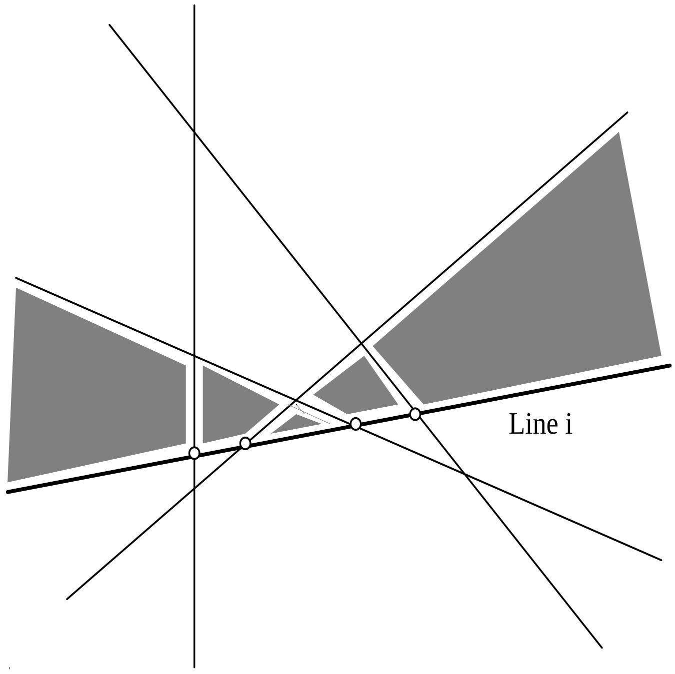


Figure 1: Incremental construction of line arrangement: Phase i

2.2 Plane Sweep

This approach works most effectively for 2-dimensional problems for which the solution can be computed incrementally as the entire input is scanned in a certain order. The concept can be easily generalized to higher dimensions[25]. This is also known as the *scan-line* method in computer graphics and used for a variety of applications such as shading, polygon filling, among others.

Consider the problem of computing the *measure* of the union of n isothetic rectangles, i.e., whose sides are paralle to the coordinate axes. We would proceed with a *vertical* sweep-line, sweeping across the plane from left to right. As we sweep the plane, we need to keep track of the rectangles that intersect the current sweep-line and those that are yet to be visited. In the meantime we compute the area covered by the union of the rectangles seen so far. More formally, associated with this approach there are two basic data structures containing all *relevant* information that should be maintained.

1. *Event schedule*, which defines a sequence of *event points* that the sweep-line status will change. In this example, the sweep-line status will change only at the left and right boundary edges of each rectangle.

2. *Sweep-line status*, which records the information of the geometric structure that is being swept. In this example the sweep-line status keeps track of the set of rectangles intersecting the current sweep-line.

The event schedule is normally represented by a *priority queue* and the list of events may change dynamically. In this case, the events are static; they are the x-coordinates of the left and right boundary edges of each rectangle. The sweep-line status is represented by a suitable data structure that supports insertions, deletions, and computation of the partial solution at each event point. In this example a *segment tree* attributed to Bentley[108] is sufficient. Since we are computing the area of the rectangles, we need to be able to know the *new* area covered by the current sweep-line between two adjacent event points. Suppose at event point x_{i-1} we maintain a partial solution $\mathcal{A}_{i-1}$. In Fig. 2 the shaded area S needs to be added to the partial solution, i.e., $\mathcal{A}_i = \mathcal{A}_{i-1} + S$. The shaded area is equal to the total measure, denoted sum_ℓ, of the union of vertical line segments representing the intersection of the rectangles and the current sweep-line times the distance between the two event points x_i and x_{i-1}. If the next event corresponds to the left boundary of a rectangle, the corresponding vertical segment, $\overline{p,q}$ in Fig. 2, needs to be inserted to the segment tree. If the next event corresponds to a right boundary edge, the segment, $\overline{u,v}$ needs to be deleted from the segment tree. In either case the total measure sum_ℓ should be updated accordingly. The correctness of this algorithm can be established by observing that the partial solution obtained for the rectangles to the *left* of the sweep-line is maintained correctly. In fact this property is typical of any algorithm based on the plane-sweep technique.

Since the segment tree structure supports segment insertions and deletions and the update (of sum_ℓ) operation in $O(\log n)$ time per event point, the total amount of time needed is $O(n \log n)$.

The measure of the union of rectangles in higher dimensions can also be solved by the plane-sweep technique with quad-trees, a generalization of segment trees.

Theorem 2 *The problem of computing the measure of n isothetic rectangles in k-dimensions can be solved in $O(n \log n)$ time, for $k \leq 2$ and in $O(n^{k-1})$ time for $k \geq 3$.*

The time bound is asymptotically optimal. Even in one-dimension, i.e., computing the total length of the union of n intervals requires $\Omega(n \log n)$ time (see [108]).

We remark that the sweep-line used in this approach is not necessarily a straight line. It can be a topological line as long as the objects stored in the sweep-line status are ordered, and the method is called *topological sweep*[11, 57]. Note that the measure of isothetic rectangles can also be solved using the divide-and-conquer paradigm (cf. Section 2.5).

2.3 Geometric Duality

This is a geometric transformation that maps a given problem into its equivalent form, preserving certain geometric properties so as to manipulate the objects in a more convenient manner. We'll see its usefulness for a number of problems to be discussed below. Here let us describe a transformation in k-dimensions, known as *polarity* or *duality*, denoted $\mathcal{D}$, that maps d-dimensional varieties to $(k-1-d)$-dimensional varieties, $0 \leq d < k$.

Consider any point $p = (\pi_1, \pi_2, \ldots, \pi_k) \in \Re^k$ other than the origin. The dual of p, denoted $\mathcal{D}(p)$, is the hyperplane $\pi_1 x_1 + \pi_2 x_2 + \cdots + \pi_k x_k = 1$. Similarly a hyperplane that does not contain the origin is mapped to a point such that $\mathcal{D}(\mathcal{D}(p)) = p$. Geometrically speaking, point p is mapped to a hyperplane whose normal is the vector determined by p

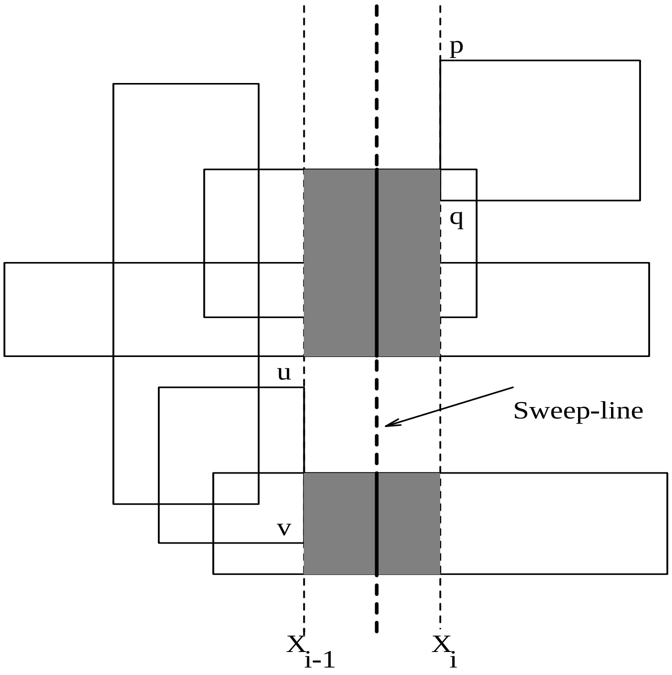


Figure 2: The plane-sweep approach to the measure problem in 2-dimensions.

and the origin, and whose distance to the origin is the reciprocal of that between p and the origin. Let S denote the unit sphere $S : x_1^2 + x_2^2 + \cdots + x_k^2 = 1$. If point p is external to S, then it is mapped to a hyperplane $\mathcal{D}(p)$ that intersects S at those points q that admit supporting hyperplanes h such that $h \cap S = q$ and $p \in h$. In 2-dimensions a point p outside of the unit disk will be mapped to a line intersecting the disk at two points q_1 and q_2 such that line segments $\overline{p, q_1}$ and $\overline{p, q_2}$ are tangent to the disk. Note that the distances from p to the origin and from the line $\mathcal{D}(p)$ to the origin are reciprocal to each other. Fig. 3(a) shows the duality transformation in 2-dimensions. In particular, point p is mapped to the line shown in boldface. For each hyperplane $\mathcal{D}(p)$ let $\mathcal{D}(p)^+$ denote the half-space that contains the origin and let $\mathcal{D}(p)^-$ denote the other half-space.

The duality transformation not only leads to dual of arrangements of hyperplanes and configurations of points and vice versa, but also preserves the following properties.

Incidence: point p belongs to hyperplane h if and only if point $\mathcal{D}(h)$ belongs to hyperplane $\mathcal{D}(p)$.

Order: point p lies in half-space h^+ (respectively h^-) if and only if point $\mathcal{D}(h)$ lies in half-space $\mathcal{D}(p)^+$ (respectively $\mathcal{D}(p)^-$).

Fig. 3(a) shows the convex hull of a set of points that are mapped by the duality transformation to the shaded region, which is the common intersection of the half-planes $\mathcal{D}(p)^+$ for all points p.

Another transformation using the unit paraboloid U, represented as $U : x_k = x_1^2 + x_2^2 + \cdots + x_{k-1}^2$, can also be similarly defined. That is, point $p = (\pi_1, \pi_2, \ldots, \pi_k) \in R^k$ is mapped to a hyperplane $\mathcal{D}_u(p)$ represented by the equation $x_k = 2\pi_1 x_1 + 2\pi_2 x_2 + \cdots + 2\pi_{k-1} x_{k-1} - \pi_k$. And each non-vertical hyperplane is mapped to a point in a similar manner such that $\mathcal{D}_u(\mathcal{D}_u(p)) = p$. Fig. 3(b) illustrates the 2-dimensional case, in which point p is mapped to a line shown in boldface. For more details see, e.g. [56, 108].

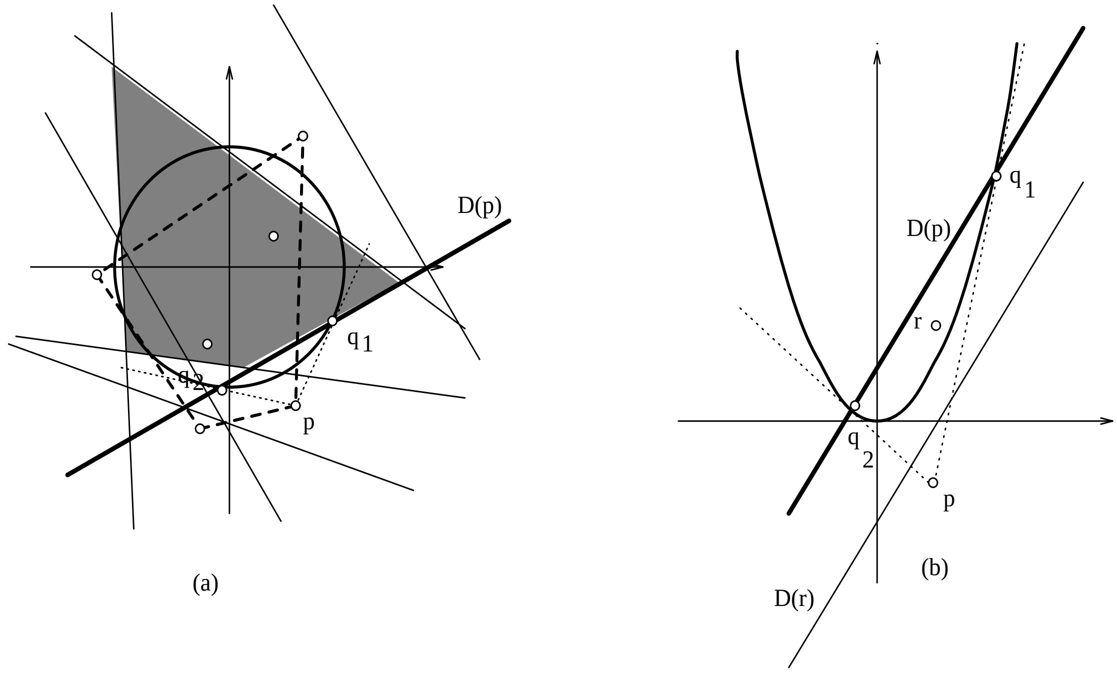


Figure 3: Geometric duality transformation in 2-dimensions.

2.4 Locus

This approach is often used as a preprocessing step for a geometric *searching* problem to achieve faster query-answering response time. For instance, given a *fixed* database consisting of geographical locations of post offices, each represented by a point in the plane, one would like to be able to efficiently answer queries of the form: "what is the nearest post office to location q?" for some query point q. The locus approach to this problem is to partition the plane into n regions, each of which consists of the locus of *query* points for which the *answer* is the same. The partition of the plane is the so-called *Voronoi* diagram (cf. Section 3.2.3.) In Fig. 7, the post office closest to query point q is site s_i. Once the Voronoi diagram is available, the query problem reduces to that of locating the region that contains the query, an instance of the point-location problem discussed in Section 3.3.

2.5 Divide-and-Conquer

This is a classical problem solving technique and has proven to be very powerful for geometric problems as well. This technique normally involves partitioning of the given problem into several subproblems, recursively solving each subproblem, and then combining the solutions to each of the subproblems to obtain the final solution to the original problem. We illustrate this paradigm by considering the problem of computing the common intersection of n half-planes in the plane. Given is a set S of n half-planes, h_i, represented by $a_i x + b_i y \leq c_i, i = 1, 2, \ldots, n$. It is well known that the common intersection of half-planes, denoted $CI(S) = \bigcap_{i=1}^{n} h_i$, is a convex set, which may or may not be bounded. If it is bounded, it is a convex polygon. See Fig. 4, in which the shaded area is the common intersection.

The divide-and-conquer paradigm consists of the following steps.

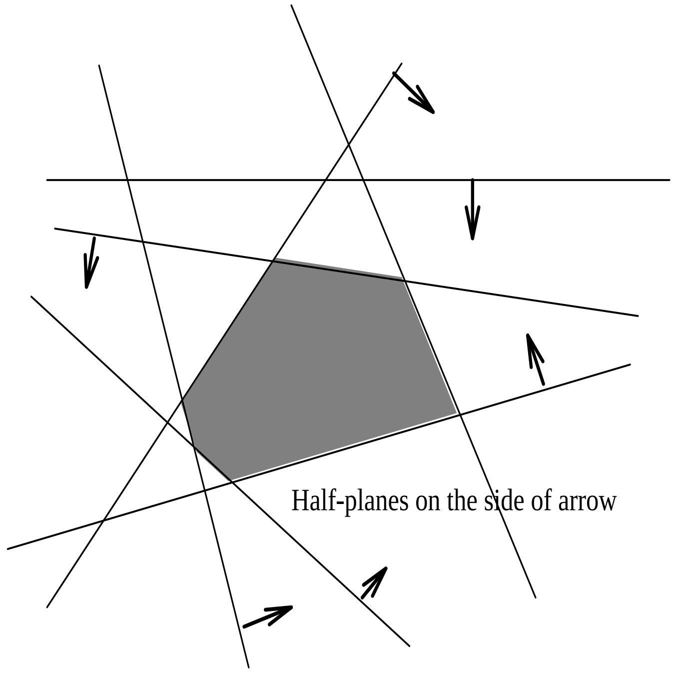


Figure 4: The common intersection of half-planes

Algorithm Common_Intersection_D&C(S)

1. If $|S| \leq 3$, compute the intersection $CI(S)$ explicitly. **Return** $(CI(S))$.
2. Divide S into two approximately equal subsets S_1 and S_2.
3. $CI(S_1) =$ **Common_Intersection_D&C**(S_1).
4. $CI(S_2) =$ **Common_Intersection_D&C**(S_2).
5. $CI(S) =$**Merge**$(CI(S_1), CI(S_2))$.
6. **Return** $(CI(S))$.

The key step is the **Merge** of two common intersections. Since $CI(S_1)$ and $CI(S_2)$ are convex, the *merge* step basically calls for the computation of the intersection of two convex polygons, which can be solved in time proportional to the size of the polygons (cf. Section 3.7). The running time of the divide-and-conquer algorithm is easily shown to be $O(n \log n)$, as given by the following recurrence formula, where $n = |S|$.

$$\begin{aligned} T(3) = & \quad O(1) \\ T(n) = & \quad 2T(\tfrac{n}{2}) + O(n) + M(\tfrac{n}{2}, \tfrac{n}{2}), \end{aligned}$$

where $M(\frac{n}{2}, \frac{n}{2}) = O(n)$ denotes the merge time (step 5).

Theorem 3 *The common intersection of n half-planes can be solved in $O(n \log n)$ time by the divide-and-conquer method.*

The time complexity of the algorithm is asymptotically optimal, as the problem of sorting can be reduced to it[108].

2.6 Prune-and-Search

This approach, developed by Dyer[54] and Megiddo[92, 93, 94], is a very powerful method for solving a number of geometric optimization problems, one of which is the well-known linear programming problem. Using this approach they obtained an algorithm whose running time is linear in the number of constraints. For more development of linear programming problems, see [95, 96]. The main idea is to prune away a fraction of *redundant* input constraints in each iteration while searching for the solution. We use a 2-dimensional linear programming problem to illustrate this approach. Without loss of generality we consider the following linear programming problem:

$$\begin{array}{ll} \textit{Minimize} & Y \\ \textit{subject to} & \alpha_i X + \beta_i Y + \gamma_i \leq 0, i = 1, 2, \ldots, n \end{array}$$

These n constraints are partitioned into three classes, C_0, C_+, C_-, depending on whether β_i is zero, positive or negative, respectively. The constraints in class C_0 define an X-interval $[x_1, x_2]$ which constains the solution, if any. The contraints in classes C_+ and C_- define, however, upward- and downward-convex piecewise linear functions $F_+(X)$ and $F_-(X)$ delimiting the feasible region[1]. (Fig. 5) The problem now becomes

$$\begin{array}{ll} \textit{Minimize} & F_-(X) \\ \textit{subject to} & F_-(X) \leq F_+(X) \\ & x_1 \leq X \leq x_2 \end{array}$$

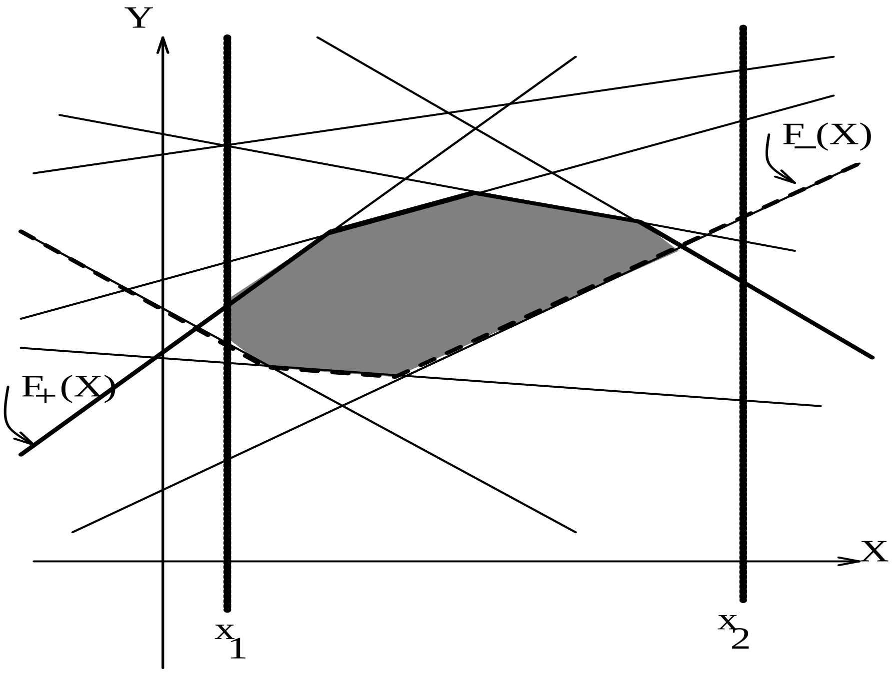


Figure 5: Feasible region defined by upward- and downward-convex piecewise linear functions.

Let λ^* denote the optimal solution, if it exists. The values of $F_-(\lambda)$ and $F_+(\lambda)$ for any λ can be computed in $O(n)$ time, based on the slopes $-\alpha_i/\beta_i$. Thus in $O(n)$ time one

[1] These upward- and downward-convex functions are also known as the upper and lower *envelopes* of the line arrangements for lines belonging to classes C_- and C_+ respectively.

can determine for any $\lambda' \in [x_1, x_2]$ if i) λ' is infeasible, and there is no solution, ii) λ' is infeasible, and we know a feasible solution is less or greaten than λ', iii) $\lambda' = \lambda^*$, or iv) λ' is feasible, and whether λ^* is less or greater than λ'.

To choose λ' we partition constraints in class C_- and C_+ into pairs and find the abscissa $\lambda_{i,j}$ of their intersection. If $\lambda_{i,j} \notin [x_1, x_2]$ then one of the constraints can be eliminated as redundant. For those $\lambda_{i,j}$'s that are in $[x_1, x_2]$ we find in $O(n)$ time[50]the median $\lambda'_{i,j}$ and compute $F_-(\lambda'_{i,j})$ and $F_+(\lambda'_{i,j})$. By the above arguments that we can determine where λ^* should lie, we know one half of the $\lambda_{i,j}$'s do not lie in the region containing λ^*. Therefore one constraint of the corresponding pair can be eliminated. The process iterates. In other words, in each iteration at least a fixed fraction $\delta = 1/4$ of the current constraints can be eliminated. Since each iteration takes $O(n)$ time, the total time spent is $Cn + C\delta n + \cdots = O(n)$. In higher dimensions we have the following result due to Dyer[54] and Clarkson[46].

Theorem 4 *A linear program in k-dimensions with n constraints can be solved in $O(3^{k^2} n)$ time.*

We note here some of the new recent developments for linear programming. There are several randomized algorithms for this problem, of which the best expected complexity, $O(k^2 n + k^{k/2+O(1)} \log n)$ is due to Clarkson[47], which is later improved by Matoušek et al.[91] to run in $O(k^2 n + e^{O(\sqrt{k \ln k})} \log n)$. Clarkson's algorithm[47] is applicable to work in a general framework, which includes various other geometric optimization problems, such as *smallest enclosing ellipsoid*. The best known deterministic algorithm for linear programming is due to Chazelle and Matoušek[42] which runs in $O(k^{7k+o(k)} n)$ time.

2.7 Dynamization

The techniques are developed for query-answering problems, classified as *geometric searching* problems, in which the underlying database is changing over (discrete) time. A typical geometric searching problem is the *membership* problem, i.e., given a set $\mathcal{D}$ of objects, determine if x is a member of $\mathcal{D}$, or the *nearest neighbor searching* problem, i.e., given a set $\mathcal{D}$ of objects, find an object that is *closest* to x according to some distance measure. In database area these two problems are referred to as the *exact match* and *best match* queries. The idea is to make use of good data structures for static database and enhance them with dynamization mechanisms so that updates of the database can be accommodated on-line and yet queries to the database can be answered efficiently.

A general query Q contains a variable of type $T1$ and is asked of a set of objects of type $T2$. The answer to the query is of type $T3$. More formally, Q can be considered as a mapping from $T1$ and subsets of $T2$ to $T3$, i.e., $Q : T1 \times 2^{T2} \rightarrow T3$. The class of geometric searching problems to which the dynamization techniques are applicable is the class of *decomposable searching problems*[20].

Definition 1 *A searching problem with query Q is decomposable if there exists an efficiently computable associative, and communtative binary operator @ satisfying the condition:*

$$Q(x, A \cup B) = @(Q(x, A), Q(x, B)).$$

In other words, the answer to a query Q in $\mathcal{D}$ can be computed by the answers to two subsets $\mathcal{D}_1$ and $\mathcal{D}_2$ of $\mathcal{D}$. The membership problem and the nearest neighbor searching problem mentioned above are decomposable.

To answer queries efficiently we have a data structure to support various update operations. There are typically three measures to evaluate a static data structure $\mathcal{A}$, i.e.,

1. $P_{\mathcal{A}}(N)$, the preprocessing tiem required to build $\mathcal{A}$,

2. $S_{\mathcal{A}}(N)$, the storage required to represent $\mathcal{A}$ and

3. $Q_{\mathcal{A}}(N)$, the query response time required to search in $\mathcal{A}$,

where N denotes the number of elements represented in $\mathcal{A}$. One would add another measure $U_{\mathcal{A}}(N)$ to represent the *update* time.

Consider the nearest neighbor searching problem in the Euclidean plane. Given a set of n points in the plane, we want to find the nearest neighbor of a query point x. One can use the Voronoi diagram data structure $\mathcal{A}$(cf. Section 3.2.3) and point location scheme (cf. Section 3.3) to achieve the following. $P_{\mathcal{A}}(n) = O(n \log n), S_{\mathcal{A}}(n) = O(n)$ and $Q_{\mathcal{A}}(n) = O(\log n)$. We now convert the static data structure $\mathcal{A}$ to a dynamic one, denoted $\mathcal{D}$, to support insertions, and deletions as well. There are a number of dynamization techniques, but we describe the technique developed by van Leeuwan and Wood[122] that provides the general flavor of the approach.

The general principle is to decompose $\mathcal{A}$ into a collection of separate data structures so that each update can be confined to one or a small, fixed number of them; however to avoid degrading the query response time we cannot afford to have excessive fragmentation since queries involve the entire collection.

Let $\{x_k\}_{k\geq 1}$ be a sequence of increasing integers, called *switch points*, where x_k is divisible by k and $x_{k+1}/(k+1) > x_k/k$. Let $x_0 = 0, y_k = x_k/k$, and n denote the current size of the point set. For a given *level* k, $\mathcal{D}$ consists of $(k+1)$ static structures of the same type, one of which, called *dump* is designated to allow for insertions. Each substructure $\mathcal{B}$ has size $y_k \leq s(\mathcal{B}) \leq y_{k+1}$, and the dump has size $0 \leq s(dump) < y_{k+1}$. A block $\mathcal{B}$ is called *low* or *full* depending on if $s(\mathcal{B}) = y_k$ or $s(\mathcal{B}) = y_{k+1}$ respectively, and is called *partial* otherwise. When an insertion to the dump makes its size equal to y_{k+1}, it becomes a full block and any non-full block can be used as the dump. If all blocks are full, we switch to the next level. Note that at this point the total size is $y_{k+1} * (k+1) = x_{k+1}$. That is, at the beginning of level $k+1$, we have $k+1$ *low* blocks and we create a *new* dump which has size 0. When a deletion from a low block occurs, we need to borrow an element either from the dump, if it is not empty, or from a partial block. When all blocks are low and $s(dump) = 0$, we switch to level $k-1$, making the low block from which the latest deletion occurs the *dump*. The level switching can be performed in $O(1)$ time. We have the following:

Theorem 5 *Any static data structure $\mathcal{A}$ used for a decomposable searching problem can be transformed into a dynamic data structure $\mathcal{D}$ for the same problem with the following performance. For $x_k \leq n < x_{k+1}$, $Q_{\mathcal{D}}(n) = O(kQ_{\mathcal{A}}(y_{k+1}))$, $U_{\mathcal{D}}(n) = O(C(n) + U_{\mathcal{A}}(y_{k+1}))$ and $S_{\mathcal{D}}(n) = O(kS_{\mathcal{A}}(y_{k+1}))$, where $C(n)$ denotes the time needed to look up the block which contains the data when a deletion occurs.*

If we choose for example x_k to be the first multiple of k greater than or equal to 2^k, i.e., $k = \log_2 n$, then y_k is about $n/\log_2 n$. Since we know there exists an $\mathcal{A}$ with $Q_{\mathcal{A}}(n) = O(\log n)$ and $U_{\mathcal{A}}(n) = P_{\mathcal{A}}(n) = O(n \log n)$ we have the following corollary.

Corollary 1 *The nearest neighbor searching problem in the plane can be solved in $O(\log^2 n)$ query time, and $O(n)$ update time. (Note that $C(n)$ in this case is $O(\log n)$).*

There are other dynamization schemes that exhibit various query-time/space and query-time/update-time tradeoffs. The interested reader is referred to [45, 56, 97, 107, 108] for more information.

2.8 Random Sampling

Randomized algorithms have received a great deal of attention recently because of their potential applications. See Chapter 5 for more information. For a variety of geometric problems randomization techniques help in building geometric subdivisions and data structures to quickly answer queries about such subdivisions. The resulting randomized algorithms are simpler to implement and/or asymptotically faster than those previously known. It is important to note that the focus of randomization is *not* on random input, such as a collection of points randomly chosen uniformly and independently from a region. We are concerned with algorithms that use a source of random numbers and analyze their performance for an arbitrary input. Unlike *Monte Carlo* algorithms, whose output may be incorrect (with very low probability), the randomized algorithms, known as *Las Vegas* algorithms, considered here are guaranteed to produce a correct output.

There are a good deal of newly developed randomized algorithms for geometric problems. See [52] for more details. Randomization gives a general way to divide and conquer geometric problems and can be used for both parallel and serial computation. We will use a familiar example to illustrate this approach.

Let us consider the problem of nearest neighbor searching discussed in Section 2.7. Let $\mathcal{D}$ be a set of n points in the plane and q be the query point. A simple approach to this problem is:

Algorithm S:

- Compute the distance to q for each point $p \in \mathcal{D}$.
- Return the point p whose distance is the smallest.

It is clear that *Algorithm S*, requiring $O(n)$ time, is not suitable if we need to answer many queries of this type. To obtain faster query response time one can use the technique discussed in Section 2.7. An alternative is to use the *random sampling* technique as follows. We pick a random sample, a subset $\mathcal{R} \subset \mathcal{D}$ of size r. Let point $p \in \mathcal{R}$ be the nearest neighbor of q in $\mathcal{R}$. The open disk $K_{\mathcal{R}}(q)$ centered at q and passing through p does not contain any other point in $\mathcal{R}$. The answer to the query is either p or some point of $\mathcal{D}$ that lies in $K_{\mathcal{R}}(q)$.

We now extend the above observation to a finite region G in the plane. Let $K_{\mathcal{R}}(G)$ be the union of disks $K_{\mathcal{R}}(r)$ for all $r \in G$. If a query q lies in G, the nearest neighbor of q must be in $K_{\mathcal{R}}(G)$ or in $\mathcal{R}$. Let us consider the Voronoi diagram, $\mathcal{V}(\mathcal{R})$ of $\mathcal{R}$ and a triangulation, $\Delta(\mathcal{V}(\mathcal{R}))$. For each triangle T with vertices a, b, c of $\Delta(\mathcal{V}(\mathcal{R}))$ we have $K_{\mathcal{R}}(T) = K_{\mathcal{R}}(a) \cup K_{\mathcal{R}}(b) \cup K_{\mathcal{R}}(c)$, shown as the shaded area in Fig. 6. A probability lemma [47] shows that with probability at least $1 - O(1/n^2)$ the candidate set $\mathcal{D} \cap K_{\mathcal{R}}(T)$ for all $T \in \Delta(\mathcal{V}(\mathcal{R}))$ contains $O(\log n)n/r$ points. More precisely, if $r > 5$ then with probability at least $1 - e^{-C/2+3\ln r}$ each open disk $K_{\mathcal{R}}(r)$ for $r \in \mathcal{R}$ contains no more than Cn/r points of $\mathcal{D}$. If we choose r to be $\sqrt{n}$, the query time becomes $O(\sqrt{n}\log n)$, a speedup from **Algorithm S**. If we apply this scheme recursively to the candidate sets of $\Delta(\mathcal{V}(\mathcal{R}))$, we can get a query time $O(\log n)$[47].

There are many applications of this random sampling techniques. Derandomized algorithms were also developed. See e.g. Chazelle and Friedman[38] for a deterministic view of random sampling and its use in geometry.

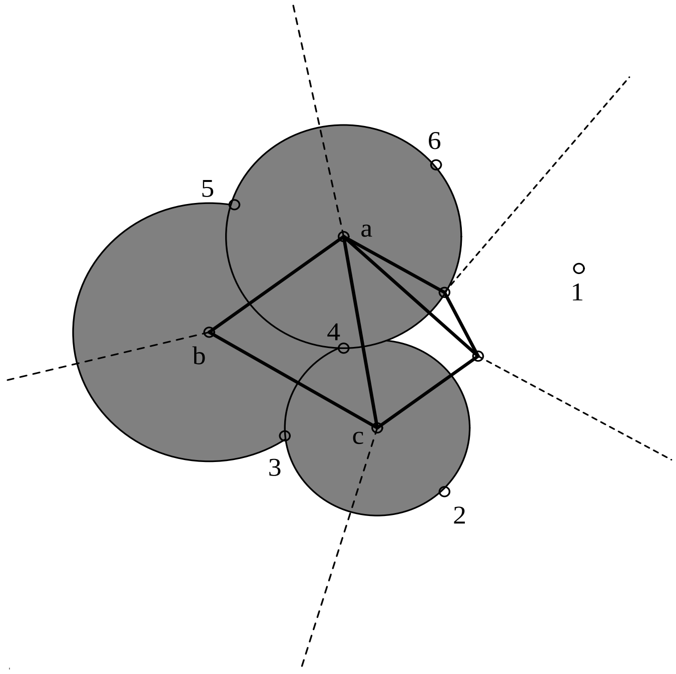


Figure 6: A triangulation of the Voronoi diagram of 6 sites and $K_{\mathcal{R}}(T)$, $T = \Delta(a, b, c)$.

3 Classes of Problems

In this section we aim to touch upon classes of problems that are fundamental in this field, and describe solutions to them, some of which may be nontrivial. The reader who needs further information about these problems is strongly encouraged to refer to the original articles cited in the references.

3.1 Convex Hull

The convex hull of a set of points in $\Re^k$ is the most fundamental problem in computational geometry. Given is a set of points, and we are interested in computing its convex hull, which is defined to be the smallest convex body containing these points. Of course the first question one has to answer is how to represent the convex hull. An implicit representation is just to list all the extreme points[2], whereas an explicit representation is to list all the extreme d-faces of dimensions $d = 0, 1, \ldots, k-1$. Thus the complexity of any convex hull algorithm would have two parts, computation part and the output part. An algorithm is said to be *output-sensitive* if its complexity depends on the size of output.

Definition 2 *The convex hull of a set S of points in $\Re^k$ is the smallest convex set containing S. In 2-dimensions, the convex hull is a convex polygon containing S; in 3-dimensions it is a convex polyhedron.*

3.1.1 Convex Hulls in 2-and 3-Dimensions

For an arbitrary set of n points in 2- and 3-dimensions, we can compute the convex hull using the *Graham scan*, *gift-wrapping* or *divide-and-conquer* paradigm, which are briefly

[2]A point in S is an extreme point if it cannot be expressed as a convex combination of other points in S. In other words, the convex hull of S would change when an extreme point is removed from S.

described below.

Recall that the convex hull of an arbitrary set of points in 2-dimensions is a convex polygon. The *Graham Scan* computes the convex hull by (i) sorting the input set of points with respect to an interior point, say O, which is the centroid of the first three non-collinear points, (ii) connecting these points into a star-shaped polygon P centered at O, and (iii) performing a linear scan to compute the convex hull of the polygon[108]. Step (i) being the dominating step, the Graham scan, takes $O(n \log n)$ time.

One can also use the *gift-wrapping* technique to compute the convex polygon. Starting with a vertex that is known to be on the convex hull, say the point O, with the smallest y-coordinate, we sweep a half-line emanting from O counterclockwise. The first point v_1 we hit will be the next point on the convex polygon. We then march to v_1, repeat the same process, and find the next vertex v_2. This process terminates when we reach O again. This is similar to wrapping an object with a *rope*. Finding the next vertex takes time proportional to the number of points remaining. Thus the total time spent is $O(n\mathcal{H})$, where $\mathcal{H}$ denotes the number of points on the convex polygon. The gift-wrapping algorithm is output-sensitive, and is more efficient than Graham scan if the number of points on the convex polygon is small, i.e. $o(\log n)$.

One can also use the divide-and-conquer paradigm. As mentioned in Section 2.5, the key step is the **Merge** of two convex hulls, each of which is the solution to a subproblem derived from the recursive step. In the division step we can recursively separate the set into two subsets by a vertical line L. Then the *merge* step basically calls for computation of two common tangents of these two convex polygons. The computation of the common tangents, also known as *bridges* over line l, begins with a segment connecting the rightmost point l of the left convex polygon to the leftmost point r of the right convex polygon. Advancing the endpoints of this segment in a *zig-zag* manner we can reach the top (or the bottom) common tangent such that the entire set of points lies on one side of the line containing the tangent. The running time of the divide-and-conquer algorithm is easily shown to be $O(n \log n)$.

A more sophisticated output-sensitive and optimal algorithm which runs in $O(n \log \mathcal{H})$ time has been developed by Kirkpatrick and Seidel[75]. It is based on a variation of the divide-and-conquer paradigm. The main idea in achieving the optimal result is that of eliminating *redundant* computations. Observe that in the divide-and-conquer approach after the common tangents are obtained, some vertices that used to belong to the left and right convex polygons must be deleted. Had we known these vertices are not on the final convex hull, we could have saved time not to compute them. Kirkpatrick and Seidel capitalized on this concept and introduced the *marriage-before-conquest* principle. They construct the convex hull by computing the upper and lower hulls of the set; the computations of these two hulls are symmetric. It performs the *divide* step as usual that decomposes the problem into two subproblems of approximately equal size. Instead of computing the upper hulls recursively for each subproblem, it finds the common tangent segment of the two *yet-to-be-computed* upper hulls, and proceeds recursively. One thing that is worth noting is that the points known not to be on the (convex) upper hull are discarded before the algorithm is invoked recursively. This is the key to obtaining a time bound that is both output-sensitive and asymptotically optimal.

The divide-and-conquer scheme can be easily generalized to 3-dimensions. The merge step in this case calls for computing common supporting faces that *wrap* two recursively computed convex polyhedra. It is observed by Preparata and Hong[108] that the common supporting faces are computed from connecting two *cyclic* sequences of edges, one on each polyhedron. The computation of these supporting faces can be accomplished in linear time, giving rise to an $O(n \log n)$ time algorithm. By applying the marriage-before-conquest principle Edelsbrunner and Shi[60] obtained an $O(n \log^2 \mathcal{H})$ algorithm.

The gift-wrapping approach for computing the convex hull in 3-dimensions would mimic the process of wrapping a gift with a piece of paper and has a running time of $O(n\mathcal{H})$.

3.1.2 Convex Hulls in k-Dimensions, $k > 3$

For convex hulls of higher dimensions, a recent result by Chazelle[33] showed that the convex hull can be computed in time $O(n \log n + n^{\lfloor k/2 \rfloor})$, which is optimal in all dimensions $k \geq 2$ in the worst case. But this result is insensitive to the output size. The gift-wrapping approach generalizes to higher dimensions and yields an output sensitive solution with running time $O(n\mathcal{H})$, where $\mathcal{H}$ is the total number of i-faces, $i = 0, 1, \ldots, k-1$ and $\mathcal{H} = O(n^{\lfloor k/2 \rfloor})$[56]. One can also use *beneath-beyond* method of adding points one at a time in ascending order along one of the coordinate axis[3]. We compute the convex hull $CH(S_{i-1})$ for points $S_{i-1} = \{p_1, p_2, \ldots, p_{i-1}\}$. For each added point p_i we update $CH(S_{i-1})$ to get $CH(S_i)$ for $i = 2, 3, \ldots, n$ by deleting those t-faces, $t = 0, 1, \ldots, k-1$, that are internal to $CH(S_{i-1} \cup \{p_i\})$. It is shown by Seidel[56] that $O(n^2 + \mathcal{H} \log n)$ time is sufficient. Most recently Chan[27] obtained an algorithm based on the gift-wrapping method that runs in $O(n \log \mathcal{H} + (n\mathcal{H})^{1-1/(\lfloor k/2 \rfloor + 1)} \log^{O(1)} n)$ time. Note that the algorithm is optimal when $k = 2, 3$. In particular, it is optimal when $\mathcal{H} = o(n^{1-\epsilon})$ for some $0 < \epsilon < 1$.

We conclude this section with the following theorem[27].

Theorem 6 *The convex hull of a set S of n points in $\Re^k$ can be computed in $O(n \log \mathcal{H})$ time for $k = 2$ or $k = 3$, and in $O(n \log \mathcal{H} + (n\mathcal{H})^{1-1/(\lfloor k/2 \rfloor + 1)} \log^{O(1)} n)$ time, where $\mathcal{H}$ is the number of i-faces, $i = 0, 1, \ldots, k-1$.*

3.2 Proximity

In this section we address proximity related problems.

3.2.1 Closest Pair

Consider a set S of n points in $\Re^k$. The closest pair problem is to find in S a pair of points whose distance is the minimum, *i.e.*, find p_i and p_j, such that $d(p_i, p_j) = \min_{k \neq l}\{d(p_k, p_l)$, for all points $p_k, p_l \in S\}$, where $d(a, b)$ denotes the Euclidean distance between a and b. (The result below holds for any distance metric in Minkowski's norm.) The brute force method takes $O(d \cdot n^2)$ time by computing all $O(n^2)$ interpoint distances and taking the minimum; the pair that gives the minimum distance is the closest pair. In 1-dimension, the problem can be solved by sorting these points and then scanning them in order, as the two closest points must occur consecutively. And this problem has a lower bound of $\Omega(n \log n)$ even in 1-dimension following from a linear time transformation from the *element uniqueness problem*. See [108].

But sorting is not applicable for dimension $k > 1$. Indeed this problem can be solved in optimal time $O(n \log n)$ by using the *divide-and-conquer* approach as follows. Let us first consider the case when $k = 2$. Consider a vertical cutting line λ that divides S into S_1

[3]If the points of S are not given *a priori*, the algorithm can be made *on-line* by adding an extra step of checking if the newly added point is internal or external to the current convex hull. If internal, just discard it.

and S_2 such that $|S_1| = |S_2| = n/2$. Let δ_i be the minimum distance defined by points in $S_i, i = 1, 2$. Observe that the minimum distance defined by points in S can be either δ_1, δ_2, or defined by two points one in each set. In the former case, we are done. In the latter, these two points must lie in the vertical strip of width $\delta = \min\{\delta_1, \delta_2\}$ on each side of the cutting line λ. The problem now reduces to that of finding the closest pair between points in S_1 and S_2 that lie inside the strip of width 2δ. This subproblem has a special property, known as the *sparcity* condition, *i.e.*, the number of points in a box[4] of length 2δ is bounded by a constant $c = 4 \cdot 3^{k-1}$, since in each set S_i, there exists no point that lies in the interior of the δ-ball centered at each point in $S_i, i = 1, 2$[108]. It is this sparcity condition that enables us to solve the bichromatic closest pair problem (cf. Section 3.2.2 for more information.) in $O(n)$ time. Let $\overline{S}_i \subseteq S_i$ denote the set of points that lie in the vertical strip. In two dimensions, the sparcity condition ensures that for each point in $\overline{S}_1$ the number of candidate points in $\overline{S}_2$ for the closest pair is at most 6. We therefore can scan these points $\overline{S}_1 \cup \overline{S}_2$ in order along the cutting line λ and compute the distance between each point scanned and its six candidate points. The pair that gives the minimum distance δ_3 is the bichromatic closest pair. The minimum distance of all pairs of points in S is then equal to $\delta_S = min\{\delta_1, \delta_2, \delta_3\}$.

Since the merge step takes linear time, the entire algorithm takes $O(n \log n)$ time. This idea generalizes to higher dimensions, except that to ensure the sparcity conidition the cutting hyperplane should be appropriately chosen to obtain an $O(n \log n)$ algorithm[108].

3.2.2 Bichromatic Closest Pair

Given two sets *red* and *blue* points, denoted R and B respectively, find two points, one in R and the other in B, that are closest among all such mutual pairs.

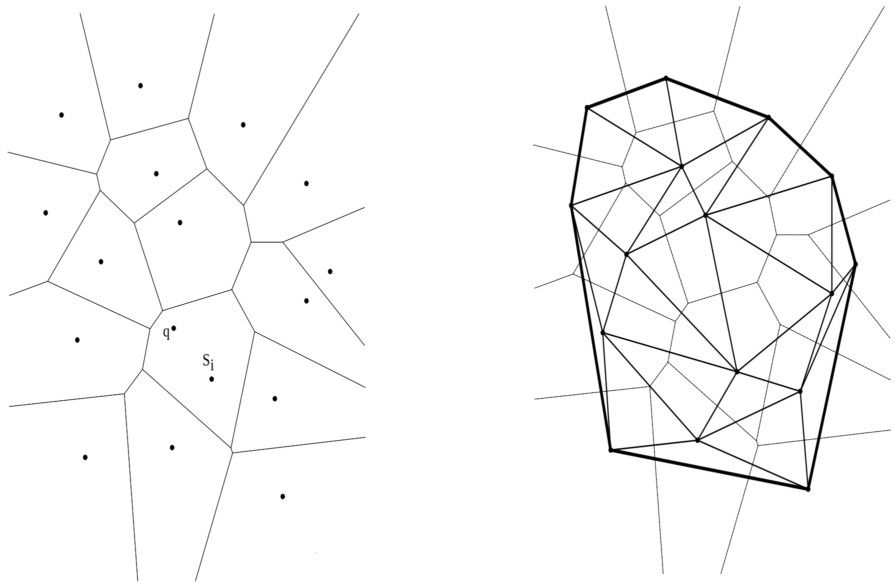
The special case when the two sets satisfy the sparcity condition defined above, can be solved in $O(n \log n)$ time, where $n = |R| + |B|$. In fact a more general problem, known as *fixed radius all nearest-neighbor problem in a sparse set* [19, 108], *i.e.*, given a set M of points in $\Re^k$ that satisfies the sparsity condition, find all pairs of points whose distance is less than a given parameter δ, can be solved in $O(|M| \log |M|)$ time[108]. The bichromatic closest pair problem in general, however, seems quite difficult. Agarwal *et al.*[1] gave an $O(n^{2(1-1/(\lceil k/2 \rceil+1))+\epsilon})$ time algorithm and a randomized algorithm with an expected running time of $O(n^{4/3} \log^c n)$ for some constant c. Chazelle et al.[36] gave an $O(n^{2(1-1/(\lfloor k/2 \rfloor+1))+\epsilon})$ time algorithm for the bichromatic farthest pair problem, which can be used to find the diamter of a set S of points by setting $R = B = S$.

A lower bound of $\Omega(n \log n)$ for the bichromatic closest pair problem can be established. (see e.g. [108].) However, when the two sets are given as two simple polygons, the bichromatic closest pair problem can be solved relatively easily. Two problems can be defined. One is the *closest visible vertex pair* problem, and the other is the *separation problem.* In the former, one looks for a red-blue pair of vertices that are visible to each other and are the closest; in the latter, one looks for two boundary points that have the shortest distance. Both the closest visible vertex pair problem and the separation problem can be solved in linear time[6, 7]. But if both polygons are convex, the separation problem can be solved in $O(\log n)$ time[34, 55].

Additional references about various variations of closest pair problems can be found in [21, 24, 73, 111, 117].

---

[4] A box is also known as a hypercube.



(a) (b)

Figure 7: The Voronoi diagram of a set of 16 points in the plane.

3.2.3 Voronoi Diagrams

The Voronoi diagram $\mathcal{V}(S)$ of a set S of points, called *sites*, $S = \{s_1, s_2, \ldots, s_n\}$ in $\Re^k$ is a partition of $\Re^k$ into Voronoi cells $V(s_i)$, $i = 1, 2, \ldots, n$, such that each cell contains points that are closer to site s_i than to any other site $s_j, j \neq i$, *i.e.,*

$$V(s_i) = \{x \in R^k | d(x, s_i) \leq d(x, s_j) \forall s_j \in R^k, j \neq i.\}$$

Figure 7(a) shows the Voronoi diagram of 16 point sites in 2-dimensions. Figure 7(b) shows the straight-line dual graph of the Voronoi diagram, which is called the Delaunay triangulation.

In 2-dimensions, $\mathcal{V}(S)$ is a planar graph and is of size linear in $|S|$. In dimensions $k \geq 2$, the total number of d-faces of dimensions $d = 0, 1, \ldots, k-1$, in $\mathcal{V}(S)$ is $O(n^{\lceil d/2 \rceil})$.

Construction of Voronoi Diagram in 2-Dimensions

The Voronoi diagram possesses many properties that are proximity related. For instance, the closest pair problem for S can be solved in linear time after the Voronoi diagram has been computed. Since this pair of points must be adjacent in the Delaunay triangulation, all one has to do is to examine all adjacent pairs of points and report the pair with the smallest distance. A divide-and-conquer algorithm to compute the Voronoi diagram of a set of points in the Euclidean plane was first given by Shamos and Hoey[108] and generalized by Lee[108] to L_p-metric for all $1 \leq p \leq \infty$. A *plane-sweep* technique for constructing the diagram is proposed by Fortune[61] that runs in $O(n \log n)$ time. There is a rich body of literature concerning the Voronoi diagram. The interested reader is referred to a recent survey by Fortune (in [52], pp.192-234).

Although $\Omega(n \log n)$ is the lower bound for computing the Voronoi diagram for an arbitrary set of n sites, this lower bound does not apply to special cases, *e.g.,* when the sites are on the vertices of a convex polygon. In fact the Voronoi diagram of a convex polygon can be computed in linear time[3]. This demonstrates further that additional property of the input can help reduce the complexity of the problem.

Construction of Voronoi Diagrams in Higher Dimensions

The Voronoi diagrams in $\Re^k$ are related to the convex hulls $\Re^{k+1}$ via a geometric transformation similar to duality discussed in Section 2.3. Consider a set of n sites in $\Re^k$, which is the hyperplane $\mathcal{H}^0$ in $\Re^{k+1}$ such that $x_{k+1} = 0$, and a paraboloid $\mathcal{P}$ in $\Re^{k+1}$ represented as $x_{k+1} = x_1^2 + x_2^2 + \cdots + x_k^2$. Each site $s_i = (\mu_1, \mu_2, \ldots, \mu_k)$ is transformed into a hyperplane $\mathcal{H}(s_i)$ in $\Re^{k+1}$ denoted as $x_{k+1} = 2\sum_{j=1}^{k} \mu_j x_j - (\sum_{j=1}^{k} \mu_j^2)$. That is, $\mathcal{H}(s_i)$ is tangent to the paraboloid $\mathcal{P}$ at a point $\mathcal{P}(s_i) = (\mu_1, \mu_2, \ldots, \mu_k, \mu_1^2 + \mu_2^2 + \cdots + \mu_k^2)$, which is just the vertical projection of site s_i onto the paraboloid $\mathcal{P}$. The half-space defined by $\mathcal{H}(s_i)$ and containing the paraboloid $\mathcal{P}$ is denoted as $\mathcal{H}^+(s_i)$. The intersection of all half-spaces, $\bigcap_{i=1}^{n} \mathcal{H}^+(s_i)$ is a convex body, and the boundary of the convex body is denoted $CH(\mathcal{H}(S))$. Any point $p \in \Re^k$ lies in the Voronoi cell $V(s_i)$ if the vertical projection of p onto $CH(\mathcal{H}(S))$ is contained in $\mathcal{H}(p)$. In other words, every κ-face of $CH(\mathcal{H}(S))$ has a vertical projection on the hyperplane $\mathcal{H}^0$ equal to the κ-face of the Voronoi diagram of S in $\mathcal{H}^0$.

We thus obtain the result which follows from Theorem 6[56].

Theorem 7 *The Voronoi diagram of a set S of n points in $\Re^k$, $k \geq 3$ can be computed in $O(n^{\lceil (k+1)/2 \rceil})$ time and $O(n^{\lceil k/2 \rceil})$ space.*

For more results concerning the Voronoi diagrams in higher dimensions and duality transformation see [14].

3.2.4 Farthest Neighbor Voronoi Diagram

The Voronoi diagram defined in Section 3.2.3 is also known as the *nearest neighbor* Voronoi diagram. A variation of this partitioning concept is a partition of the space into cells, each of which is associated with a site, and which contains all points that are farther from the site than from any other site. This diagram is called the *farthest neighbor* Voronoi diagram. Unlike the *nearest neighbor* Voronoi diagram only a subset of sites have a Voronoi cell associated with them. Those sites that have a non-empty Voronoi cell are those that lie on the convex hull of S. A similar partitioning of the space is known as the order κ-*nearest neighbor* Voronoi diagram, in which each Voronoi cell is associated with a subset of κ sites in S for some fixed integer κ such that these κ sites are the closest among all other sites. For $\kappa = 1$ we have the *nearest neighbor* Voronoi diagram, and for $\kappa = n - 1$ we have the *farthest neighbor* Voronoi diagram. The higher-order Voronoi diagrams in k-dimensions are related to the levels of hyperplane arrangements in $k + 1$ dimensions using the paraboloid transformation[56].

Since the *farthest neighbor* Voronoi diagram is related to the convex hull of the set of sites, one can use the *marriage-before-conquest* paradigm of Kirkpatrick and Seidel[73] to compute the *farthest neighbor* Voronoi diagram of S in 2-dimensions in time $O(n \log \mathcal{H})$, where $\mathcal{H}$ is the number of sites on the convex hull.

3.2.5 Weighted Voronoi Diagrams

When the sites are associated with weights such that the distance function from a point to the sites is weighted, the structure of the Voronoi diagram can be drastically different than the unweighted case.

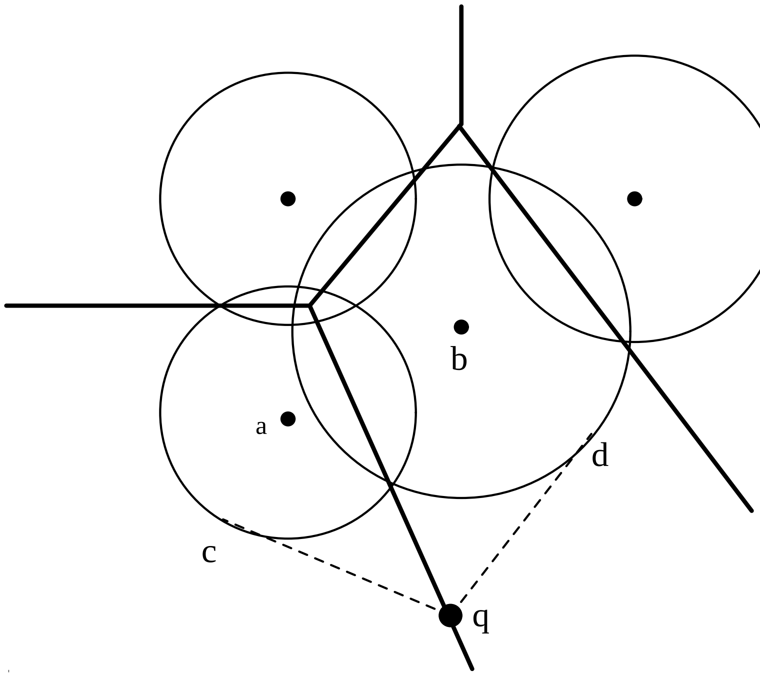


Figure 8: The power diagram in two dimensions; solid lines are equidistant to two sites.

Power Diagrams

Suppose each site s in $\Re^k$ is associated with a non-negative weight, w_s. For an arbitrary point p in $\Re^k$ the weighted distance from p to s is defined as

$$\delta(s,p) = d(s,p)^2 - w_s^2$$

If w_s is positive, and if $d(s,p) \geq w_s$, then $\sqrt{\delta(s,p)}$ is the length of the tangent of p to the ball, $b(s)$, of radius w_s and centered at s. $\delta(s,p)$ is also called the *power of* p with respect to the ball $b(s)$. The locus of points p equidistant from two sites $s \neq t$ of equal weight, will be a hyperplane called the *chordale* of s and t. See Fig. 8. Point q is equidistant to sites a and b and the distance is the length of the tangent line $\overline{q,c} = \overline{q,d}$.

The power diagram of two dimensions can be used to compute the contour of the union of n disks, and the connected components of n disks in $O(n \log n)$ time, and in higher dimensions it can be used to compute the union or intersection of n axis-parallel cones in $\Re^k$ with apexes in a common hyperplane in time $O(CH_{k+1}(n))$, the multiplicative weighted nearest-neighbor Voronoi diagram (defined below) for n points in $\Re^k$ in time $O(CH_{k+2}(n))$, and the Voronoi diagrams for n spheres in $\Re^k$ in time $O(CH_{k+2}(n))$, where $CH_\ell(n)$ denotes the time for constructing the convex hull of n points in $\Re^\ell$[13]. For the best time bound for $CH_\ell(n)$ consult Section 3.1.2.

Multiplicative-Weighted Voronoi Diagrams

Each site $s \in \Re^k$ has a positive weight w_s, and the distance from a point p to s is defined as

$$\delta_{multi-w}(s,p) = d(p,s)/w_s$$

In 2-dimensions, the locus of points equidistant to two sites $s \neq t$ is a circle, if $w_s \neq w_t$, and a perpendicular bisector of line segment $\overline{s,t}$, if $w_s = w_t$. Each cell associated with a site

s consists of all points closer to s than to any other site and may be disconnected. In the worst case the nearest neighbor Voronoi diagram of a set S of n points in 2-dimensions can have an $O(n^2)$ regions and can be found in $O(n^2)$ time. In 1-dimension, the diagram can be computed optimally in $O(n \log n)$ time. However, the farthest neighbor multiplicative-weighted Voronoi diagram has a very different characteristic. Each Voronoi cell associated with a site remains connected, and the size of the diagram is still linear in the number of sites. An $O(n \log^2 n)$ time algorithm for constructing such a diagram is given in [85]. See [110] for more applications of the diagram.

Additive-Weighted Voronoi Diagrams

The distance of a point p to a site s of a weight w_s is defined as

$$\delta_{add-w}(s,p) = d(p,s) - w_s$$

In 2-dimensions, the locus of points equidistant to two sites $s \neq t$ is a branch of a hyperbola, if $w_s \neq w_t$, and a perpendicular bisector of line segment $\overline{s,t}$ if $w_s = w_t$. The Voronoi diagram has properties similar to the ordinary unweighted diagram. For example, each cell is still connected and the size of the diagram is linear. If the weights are positive, the diagram is the same as the Voronoi diagram of a set of spheres centered at site s and of radius w_s and in 2-dimensions this diagram for n disks can be computed in $O(n \log^2 n)$ time[79, 113] and in $k \geq 3$ one can use the notion of power diagram to compute the diagram[13].

3.2.6 Other Generalizations

The sites mentioned so far are point sites. They can be of different shapes. For instance, they can be line segments, disks, or polygonal objects. The metric used can also be convex distance function or other norms. See [4, 23, 76, 127] for more information.

3.3 Point Location

Point-location is yet another fundamental problem in computational geometry. Given a planar subdivision, and a query point, one would like to find which region contains the point in question.

In this context we are mostly interested in fast response time to answer repeated queries to a fixed database. An earlier approach is based on the *slab method*[108], in which parallel lines are drawn through each vertex, thus partitioning the plane into parallel slabs. Each parallel slab is further divided into subregions by the edges of the subdivision that can be ordered. Any given point can thus be located by two binary searches; one to locate the slab containing the point among the $n+1$ horizontal slabs, and followed by another to locate the region defined by a pair of consecutive edges which are ordered from left to right. This requires preprocessing of the planar subdivision, and setting up suitable search tree structures for the slabs and the edges crossing each slab. We use a three tuple, $(P(n), S(n), Q(n))$= (preprocessing time, space requirement, query time) to denote the performance of the search strategy (cf. Section 2.7). The slab method gives an $(O(n^2), O(n^2), O(\log n))$ algorithm. Since preprocessing time is only performed once, the time requirement is not as critical as the space requirement. The primary goal of the query processing problems is to minimize the query time and the space required.

Lee and Preparata[108] first proposed a *chain decomposition* method to decompose a monotone planar subdivision with n points into a collection of $m \leq n$ monotone chains organized in a complete binary tree. Each node in the binary tree is assoiated with a monotone chain of at most n edges, ordered in y-coordinate. Between two adjacent chains there are a number of disjoint regions. Each query point is compared with the node, hence the associated chain, to decide on which side of the chain the query point lies. Each chain comparison takes $O(\log n)$ time, and total number of nodes visited is $O(\log m)$. The search on the binary tree will lead to two adjacent chains and hence identify a region that contains the point. Thus the query time is $O(\log m \log n) = O(\log^2 n)$. Unlike the slab method in which each edge may be stored as many as $O(n)$ times, resulting in $O(n^2)$ space, it can be shown that each edge in the planar subdivision, with an appropriate chain assignment scheme, is stored only once. Thus the space requirement is $O(n)$. The chain decomposition scheme gives rise to an $(O(n \log n), O(n), O(\log^2 n))$ algorithm. The binary search on the chains is not efficient enough. Recall that after each *chain comparison*, we will move down the binary search tree to perform the next chain comparison and start over another binary search on y-coordinate to find an edge of the chain, against which a comparison is made to decide if the point lies to the left or right of the chain. A more efficient scheme is to perform binary search of the y-coordinate at the root node and to spend only $O(1)$ time per node as we go down the chain tree, shaving off an $O(\log n)$ factor from the query time[58]. This scheme is similar to the ones adopted by Chazelle and Guibas[40] in fractional cascading search paradigm and by Willard[124] in his range tree search method. With the linear time algorithm for triangulating a simple polygon due to Chazelle[31] (cf. Section 3.6.1) we conclude with the following optimal search structure for planar point-location.

Theorem 8 *Given a planar subdivision of n vertices, one can preprocess the subdivision in linear time and space such that each point location query can be answered in $O(\log n)$ time.*

The point location problem in arrangements of hyperplanes is also of significant interest. See, e.g., [39]. Dynamic versions of the point location problem have also been investigated. See [45] for a survey of dynamic computational geometry.

3.4 Motion Planning — Path Finding Problems

The problem is mostly cast in the following setting. Given are a set of obstacles O, an object, called *robot*, and an initial and final position, called source and destination respectively. We wish to find a path for the robot to move from the source to the destination avoiding all the obstacles. This problem arises in several contexts. For instance, in robotics this is referred to as the *piano movers' problem*[128] or *collision avoidance* problem, and in VLSI routing this is the *wiring* problem for 2-terminal nets. In most applications we are searching for a collision avoidance path that has a *shortest* length, where the distance measure is based on the Euclidean or L_1-metric. For more information regarding motion planning see e.g., [5, 128].

3.4.1 Path Finding in 2-Dimensions

In 2-dimensions, the Euclidean shortest path problem in which the robot is a point, and the obstacles are simple polygons, is well studied. A most fundamental approach is by using the notion of *visibility graph*. Since the shortest path must make turns at polygonal vertices, it is sufficient to construct a graph whose vertices are the vertices of the polygonal obstacles and the source and destination, and whose edges are determined by vertices that are mutually *visible*, i.e., the segment connecting the two vertices does not intersect the interior of any obstacle. Once the visibility graph is constructed with edge weight equal to the Euclidean distance between the two vertices, one can then apply the Dijkstra's shortest path algorithms[108] to find a shortest path between the source and destination. The Euclidean shortest path between two points is referred to as the *geodesic* path and the distance as the *geodesic* distance. The computation of the visibility graph is the dominating factor for the complexity of any visibility graph based shortest path algorithm. Research results aiming at more efficient algorithms for computing the visibility graph and for computing the geodesic path in time proportional to the size of the graph have been obtained. In [63] Ghosh and Mount gave an output sensitive algorithm that runs in $O(E + n \log n)$ time for computing the visibility graph, where E denotes the number of edges in the graph.

Mitchell[98] used the so-called *continuous Dijkstra* wave front approach to the problem for general polygonal domain of n obstacle vertices and obtained an $O(n^{5/3+\epsilon})$ time algorithm. He constructed a *shortest path map* that partitions the plane into regions such that all points q that lie in the same region have the same vertex sequence in the shortest path from the given source to q. The shortest path map takes $O(n)$ space and enables us to perform shortest path queries, i.e., find a shortest path from the given source to any query points, in $O(\log n)$ time. Hershberger and Suri[67] on the other hand used plane subdivision approach and presented an $O(n \log^2 n)$-time and $O(n \log n)$-space algorithm to compute the *shortest path map* of a given source point. They later improved the time bound to $O(n \log n)$. If the source-destination path is confined in a simple polygon with n vertices, the shortest path can be found in $O(n)$ time[108].

In the context of VLSI routing one is mostly interested in rectilinear paths (L_1-metric) whose edges are either horizontal or vertical. As the paths are restricted to be rectilinear, the shortest path problem can be solved more easily. Lee et al.[87] gave a survey on this topic.

In a two-layer VLSI routing model, the number of segments in a rectilinear path reflects the number of *vias*, where the wire segments change layers, which is a factor that governs the fabrication cost. In robotics, a straight line motion is not as costly as *making turns*. Thus the number of segments (or *turns*) has also become an objective function. This motivates the study of the problem of finding a path with the smallest number of segments, called the *minimum link path problem*[101, 118].

These two cost measures, length and number of links, are in conflict with each other. That is, a shortest path may have far too many links, whereas a minimum link path may be arbitrarily long compared with a shortest path. Instead of optimizing both measures *simultaneously* one can seek a path that either optimizes a linear function of both length and the number of links or optimizes them in a lexicographical order. For example, we optimize the length first, and then the number of links, i.e., among those paths that have the same shortest length, find one whose number of links is the smallest and vice versa.

A generalization of the collision avoidance problem is to allow collision with a cost. Suppose each obstacle has a weight which represents the cost if the obstacle is *penetrated*. Mitchell and Papadimitriou[100] first studied the weighted region shortest path problem. Lee et al.[86] studied a similar problem in the rectilinear case. Another generalization is to include in the set of obstacles some subset $F \subset O$ of obstacles, whose vertices are *forbidden* for the solution path to make turns. Of course, when the weight of obstacles is set to be ∞, or the forbidden set $F = \emptyset$, these generalizations reduce to the ordinary collision avoidance problem.

3.4.2 Path Finding in 3-Dimensions

The Euclidean shortest path problem between two points in a 3-dimension polyhedral environment turns out to be much harder than its 2-dimensional counterpart. Consider a convex polyhedron P with n vertices in 3-dimensions and two points s, d on the surface of P. A shortest path from s to d on the surface will cross a sequence of edges, denoted $\xi(s, d)$. $\xi(s, d)$ is called the *shortest path edge sequence* induced by A and B and consists of distinct edges. If the edge sequence is known, the shortest path between s and d can be computed by a planar unfolding procedure so that these faces crossed by the path lie in a common plane and the path becomes a straight line segment.

Mitchell et al.[99] gave an $O(n^2 \log n)$ algorithm for finding a shortest path between s and d even if the polyhedron may not be convex. If s and d lie on the surface of two different polyhedra, Sharir[114] gave an $O(N^{O(k)})$ algorithm, where N denotes the total number of vertices of k obstacles, In general, the problem of determining the shortest path edge sequence of a path between two points among k polyhedra is NP-hard[26].

3.4.3 Motion Planning of Objects

In the previous sections we discuss path planning for moving a point from the source to a destination in the presence of polygonal or polyhedral obstacles. We now briefly describe the problem of moving a polygonal or polyhedral object from an initial position to a final position subject to translational and/or rotational motions.

Consider a set of k convex polyhedral obstacles, $O_1, O_2, \ldots, O_k$, and a convex polyhedral robot, R in 3-dimensions. The motion planning problem is often solved by using the so-called *configuration space*, denoted $\mathcal{C}$, which is the space of parametric representations of possible robot placements[88]. The free placement FP is the subspace of $\mathcal{C}$ of points at which the robot does not intersect the interior of any obstacle. For instance, if only translations of R are allowed, the free configuration space will be the union of the Minkowski sums $M_i = O_i \oplus (-R) = \{a - b | a \in O_i, b \in R\}$ for $i = 1, 2, \ldots, k$. A *feasible* path exists if the initial placement of R and final placement belong to the same connected component of FP. And the problem is to find a continuous curve connecting the initial and final positions in FP. The combinatorial complexity, i.e., the number of vertices, edges, and faces on the boundary of FP, largely influences the efficiency of any $\mathcal{C}$-based algorithm. For translational motion planning, Aronov and Sharir[8] showed that the combinatorial complexity of FP is $O(nk \log^2 k)$, where k is the number of obstacles defined above and n the total complexity of the Minkowski sums $M_i, 1 \leq i \leq k$.

Moving a ladder (represented as a line segment) among a set of polygonal obstacles of size n can be done in $O(K \log n)$ time, where K denotes the number of pairs of obstacle vertices whose distance is less than the length of the ladder, and is $O(n^2)$ in general[116]. If the moving robot is also a polygonal object, Avnaim et al.[17] showed that $O(n^3 \log n)$ time suffices. When the obstacles are *fat*[5] Van der Stappen and Overmars[121] showed that the above two 2-dimensional motion planning problems can be solved in $O(n \log n)$ time, and in 3-dimensions the problem can be solved in $O(n^2 \log n)$ time, if the obstacles are ℓ-fat for some positive constant ℓ.

[5]An object $O \subseteq R^k$ is said to be *ℓ-fat*, if for all hyperspheres S centered inside O and not fully containing O we have $\ell \cdot volume(O \cap S) \geq volume(S)$.

3.5 Geometric Optimization

The geometric optimization problems arise in operations research, pattern recognition and other engineering disciplines. We list some representative problems.

3.5.1 Minimum Cost Spanning Trees

The minimum (cost) spanning tree, *MST*, of an undirected, weighted graph $G(V, E)$, in which each edge has a nonnegative weight, is a well studied problem in graph theory and can be solved in $O(|E| \log |V|)$ time[108]. When cast in the Euclidean or other L_p-metric plane in which the input consists of a set S of n points, the complexity of this problem becomes different. Instead of constructing a *complete* graph whose edge weight is defined by the distance between its two endpoints, from which to extract an *MST*, a sparse graph, known as the *Delaunay triangulation* of the point set, is computed. It can be shown that the *MST* of S is a subgraph of the Delaunay triangulation. Since the *MST* of a planar graph, can be found in linear time[108], the problme can be solved in $O(n \log n)$ time. In fact, this is asymptotically optimal, as the closest pair of the set of points must define an edge in the *MST*, and the closest pair problem is known to have an $\Omega(n \log n)$ lower bound, as mentioned in Section 3.2.1.

This problem in dimensions 3 or higher can be solved in subquadratic time. For instance, in 3-dimensions, $O((n \log n)^{1.5})$ time is sufficient[28] and in $k \geq 3$ dimensions $O(n^{2(1-1/(\lceil k/2 \rceil+1))+\epsilon})$ time suffices[1].

3.5.2 Minimum Diameter Spanning Tree

The minimum *diameter* spanning tree, *MDST*, of an undirected weighted graph $G(V, E)$ is a spanning tree such that the total weight of the longest path in the tree is minimum. This arises in applications to communication network where a tree is sought such that the maximum delay, instead of the total cost, is to be minimized. A graph-theoretic approach yields a solution in $O(|E||V| \log |V|)$ time[66]. In [68] Ho *et al.* showed that by the triangle inequality there exists an *MDST* such that the longest path in the tree consists of no more than *three* segments. Based on this an $O(n^3)$ time algorithm was obtained.

Theorem 9 Given a set S of n points, the minimum diameter spanning tree for S can be found in $\theta(n^3)$ time and $O(n)$ space.

We remark that the problem of finding a spanning tree whose total cost **and** the diameter are both bounded is NP-complete[68]. Another similar problem that arises in VLSI clock tree routing is to find a tree from a source to multiple sinks such that every source-to-sink path is the shortest and the total wire length is to be minimized. This problem is still not known to be solvable in polynomial time or NP-hard. Recently we have shown that the problem of finding a minimum spanning tree such that the longest source-to-sink path is bounded by a given parameter is NP-complete[112].

3.5.3 Minimum Enclosing Circle Problem

Given a set S of points the problem is to find a smallest disk enclosing the set. This problem is also known as the (unweighted) 1-center problem. That is, find a center such that the maximum distance from the center to the points in S is minimized. More formally, we need to find the center $c \in \Re^2$ such that $\max_{p_j \in S} d(c, p_j)$ is minimized. The weighted

1-center problem, in which, the distance function $d(c, p_j)$ is multiplied by the weight w_j, is a well-known minimax problem, also known as the *emergency center problem* in operations research. In 2-dimensions, the 1-center problem can be solved in $O(n)$ time[54, 93]. The minimum enclosing ball problem in higher dimensions is also solved by using linear programming technique[93, 94].

3.5.4 Largest Empty Circle Problem

This problem, in contrast to the minimum enclosing circle problem, is to find a circle centered in the interior of the convex hull of the set S of points that does not contain any given point and the radius of the circle is to be maximized. This is mathematically formalized as a maximin problem, the minimum distance from the center to the set is maximized. The weighted version is also known as the *obnoxious center* problem in facility location. An $O(n \log n)$ time solution for the unweighted version can be found in [108].

3.5.5 Minimum Annulus Covering Problem

The *minimum annulus covering problem* is defined as follows. Given a set of S of n points find an annulus (defined by two concentric circles) whose center lies internal to the convex hull of S such that the *width* of the annulus is minimized. The problem arises in mechanical part design. To measure if a circular part is *round*, an ANSI standard is to use the width of an annulus covering the set of points obtained from a number of measurements. This is known as the *roundness* problem[77]. It can be shown that the center of the annulus is either at a vertex of the nearest neighbor Voronoi diagram, a vertex of the farthest neighbor Voronoi diagram, or at the intersection of these two diagrams[77]. If the input is defined by a simple polygon P with n vertices, and the problem is to find a minimum-width annulus that contains the boundary of P, the problem can be solved in $O(n \log n + k)$, where k denotes the number of intersection points of the *medial axis* of the simple polygon and the boundary of P[77]. When the polygon is known to be convex, a linear time is sufficient[119]. If the center of the smallest annulus of a point set can be arbitrarily placed, the center may lie at infinity and the annulus degenerates to a pair of parallel lines enclosing the set of points. This problem is different from the problem of finding the *width* of a set, which is to find a pair of parallel lines enclosing the set such that the distance between them is minimized. The *width* of a set of n points can be found in $O(n \log n)$ time, which is optimal[84]. In 3-dimensions the *width* of a set is also used as a measure for flatness of a *plate — flatness* problem. Houle and Toussaint[69] gave an $O(n^2)$ time algorithm, and Chazelle et al.[36] improved it to $O(n^{8/5+\epsilon})$.

3.6 Decomposition

Polygon decomposition arises in pattern recognition in which recognition of a shape is facilitated by first decomposing it into simpler parts, called *primitives*, and comparing them to templates previously stored in a library via some similarity measure. The primitives are often convex with the simplest being the shape of a triangle.

We consider two types of decomposition, *partition* and *covering*. In the former type, the components are pairwise disjoint except they may have some boundary edges in common. In the latter type, the components may overlap. A minimum decomposition is one such that the number of components is minimized. Sometimes additional points, called *Steiner* *points*, may be introduced to obtain a minimum decomposition. Unless otherwise specified, we assume that no Steiner points are used.

3.6.1 Triangulation

Triangulating a simple polygon or in general, triangulating a planar straigth-line graph, is a process of introducing noncrossing edges so that each face is a triangle. It is also a fundamental problem in computer graphics, geographical information systems, and finite element methods.

Let us begin with the problem of triangulating a simple polygon with n vertices. It is obvious that for a simple polygon with n edges, one needs to introduce at most $n-3$ diagonals to triangulate the interior into $n-2$ triangles. This problem has been studied very extensively. A pioneering work is due to Garey et al., which gave an $O(n \log n)$ algorithm and a linear algorithm if the polygon is monotone[105, 108]. A recent breakthrough linear time triangulation result of Chazelle[31] settled the longstanding open problem. As a result of this linear triangulation algorithm, a number of problems can be solved in linear time. For example, the simplicity test, defined in Section 3.7.1 and many other shortest path problems inside a simple polygon[64]. Note that if the polygons have holes, the problem of triangulating the interior requires $\Omega(n \log n)$ time[10].

Sometimes we want to look for *quality* triangulation, instead of just an arbitrary one. For instance, triangles with large or small angles is not desirable. It is well-known that the Delaunay triangulation of points in general position is unique and it will maximize the minimum angle. In fact, the characteristic angle vector[6] of the Delaunay triangulation of a set of points is *lexicographically maximum*[78]. The notion of Delaunay triangulation of a set of points can be generalized to a planar straight-line graph $G(V, E)$. That is, we'd like to have G as a subgraph of a triangulation $G'(V, E'), E \subseteq E'$, such that each triangle satisfies the *empty circumcircle* property: no vertex *visible* from the vertices of triangle is contained in the interior of the circle. This *generalized* Delaunay triangulation was first introduced by Lee[78] and an $O(n^2)$ (respectively $O(n \log n)$) algorithm for constructing the generalized triangulation of a planar graph (respectively a simple polygon) with n vertices was given in [80]. Chew[44] later improved the result and gave an $O(n \log n)$ time algorithm using divide-and-conquer. Triangulations that minimize the maximum angle or maximum edge length were also studied. But if constraints on the measure of the triangles, for instance, each triangle in the triangulation must be nonobtuse, then Steiner points must be introduced. See Bern and Eppstein (in [52], pp. 23-90) for a survey of different criteria of triangulations and discussions of triangulations in 2- and 3-dimensions.

The problem of triangulating a set P of points in $\Re^k$, $k \geq 3$ is less studied. In this case the convex hull of P is to be partitioned into $\mathcal{F}$ nonoverlapping simplices, the vertices of which are points in P. A simplex in k-dimensions consists of exactly $k+1$ points, all of which are extreme points. Avis and ElGindy[15] gave an $O(k^4 n \log_{1+1/k} n)$ time algorithm for triangulating a simplicial set of n points in $\Re^k$. In $\Re^3$ an $O(n \log n + \mathcal{F})$ time algorithm was presented and $\mathcal{F}$ is shown to be linear if no three points are collinear, and at most $O(n^2)$ otherwise. See [52] for more references on 3-dimensional triangulations and Delaunay triangulations in higher dimensions.

[6]The characteristic angle vector of a triangulation is a vector of minimum angles of each triangle arranged in nondescending order. For a given point set the number of triangles is the same for all triangulations, and therefore each of them is associated with a characteristic angle vector.

3.6.2 Other Decompositions

Partitioning a simple polygon into shapes such as convex polygons, star-shaped polygons, spiral polygons, monotone polygons, etc. has also been investigated[120]. A linear time algorithm for partitioning a polygon into star-shaped polygons was given by Avis and Toussaint[16] after the polygon has been triangulated. This algorithm provided a very simple proof of the traditional art gallery problem originally posed by Klee, i.e., $\lfloor n/3 \rfloor$ vertex guards are always sufficient to see the entire region of a simple polygons with n vertices. But if a minimum partition is desired, Keil[74] gave an $O(n^5 N^2 \log n)$ time, where N denotes the number of reflex vertices. However, the problem of *covering* a simple polygon with a minimum number of star-shaped parts is *NP*-hard[81]. The problem of partitioning a polygon into a minimum number of convex parts can be solved in $O(N^2 n \log n)$ time[74]. The minimum covering problem by star-shaped polygons for rectilinear polygons is still open. For variations and results of art gallery problems the reader is referred to [104, 115]. Polynomial time algorithms for computing the minimum partition of a simple polygon into simpler parts while allowing Steiner points can be found in [10, 120]

The minimum partition or covering problem for simple polygons becomes NP-hard when the polygons are allowed to have *holes* [74, 106]. Asano et al.[9] showed that the problem of partitioning of a simple polygon with h holes into a minimum number of trapezoids with two horizontal sides can be solved in $O(n^{h+2})$ time, and that the problem is NP-complete, if h is part of the input. An $O(n \log n)$ time 3-approximation algorithm was presented. Imai and Asano[70] gave an $O(n^{3/2} \log n)$ time and $O(n \log n)$ space algorithm for partitioning a rectilinear polygons with holes into a minimum number of rectangles (allowing Steiner points). The problem of covering a rectilinear polygon (without holes) with a minimum number of rectanges, however, is also NP-hard[48].

The problem of minimum partition into convex part and the problem of determining if a nonconvex polyhedron can be partitioned into tetrahedra without introducing Steiner points are NP-hard[106, 109].

3.7 Intersection

This class of problems arises in architectual design, computer graphics[51], etc. and encompasses two types of problems, *intersection detection* and *intersection computation.*

3.7.1 Intersection Detection Problems

The intersection detection problem is of the form: Given a set of objects, do any two intersect? The intersection detection problem has a lower bound of $\Omega(n \log n)$[108]. The pairwise intersection detection problem is a precursor to the general intersection detection problem.

In 2-dimensions the problem of detecting if two polygons of r and b vertices intersect was easily solved in $O(n \log n)$ time, where $n = r + b$ using the red-blue segment intersection algorithm[89]. However, this problem can be reduced in linear time to the problem of detecting the self intersection of a polygonal curve. The latter problem is known as the *simplicity* test and can be solved optimally in linear time by Chazelle's linear time triangulation algorithm[31]. If the two polygons are convex, then $O(\log n)$ suffices[34, 55]. We remark here that although detecting if two convex polygons intersect can be done in logarithmic time, detecting if the boundary of the two convex polygons intersect requires $\Omega(n)$ time[34].

In 3-dimensions, detecting if two convex polyhedra intersect can be solved in linear time by using a hierarchical representation of the convex polyhedron, or by formulating it as a linear programming problem in 3 variables[34, 49, 53, 93].

For some applications we would not only detect intersection, but also *report* all such intersecting pairs of objects or *count* the number of intersections, which is discussed next.

3.7.2 Intersection Reporting/Counting Problems

One of the simplest such intersecting reporting problems is that of *reporting* all intersecting pairs of line segments in the plane. Using the plane sweep technique one can obtain an $O((n+\mathcal{F})\log n)$ time, where $\mathcal{F}$ is the output size. It is not difficult to see that the lower bound for this problem is $\Omega(n\log n+\mathcal{F})$; thus the above algorithm is $O(\log n)$ factor from the optimal. Recently this segment intersection reporting problem was solved optimally by Chazelle and Edelsbrunner[35] who used several important algorithm design and data structuring techniques, as well as some crucial combinatorial analysis. In contrast to this asymptotically optimal *deterministic* algorithm, a simpler randomized algorithm for this problem that takes $O(n\log n+\mathcal{F})$ time but requires only $O(n)$ space (instead of $O(n+\mathcal{F})$) was obtained[52]. Balaban[18] most recently reported a deterministic algorithm that solves this problem optimally both in time and space.

On a separate front, the problem of finding intersecting pairs of segments from different sets was considered. This is called *bichromatic line segment* intersection problem. Nievergelt and Preparata[103] considered the problem of *merging* two planar convex subdivisions of total size n and showed that the resulting subdivision can be computed in $O(n\log n+\mathcal{F})$ time. This result[103] was extended in two ways. Mairson and Stolfi[89] showed that the bichromatic line segment intersection reporting problem, can be solved in $O(n\log n+\mathcal{F})$ time. Guibas and Seidel[65] showed that merging two convex subdivisions can actually be solved in $O(n+\mathcal{F})$ time using topological plane-sweep.

Most recently Chazelle et al.[37] used *hereditary segment trees* structure and *fractional cascading*[40] and solved both segment intersection reporting and counting problems optimally in $O(n\log n)$ time and $O(n)$ space. (The term $\mathcal{F}$ should be included for reporting.)

The *rectangle intersection reporting* problem arises in the design of VLSI circuitry, in which each rectangle is used to model a certain circuitry component. This is a well-studied classical problem and optimal algorithms ($O(n\log n+\mathcal{F})$ time) have been reported (see [83] for references). The k-dimensional hyperrectangle intersection reporting (respectively, counting) problem can be solved in $O(n^{k-2}\log n+\mathcal{F})$ time and $O(n)$ space (respectively, in time $O(n^{k-1}\log n)$ and space $O(n^{k-2}\log n)$).

3.7.3 Intersection Computation

Computing the actual intersection is a basic problem, whose efficient solutions often lead to better algorithms for many other problems.

Consider the problem of computing the common intersection of half-planes discussed in Section 2.5. Efficient computation of intersection of two convex polygons is required. The intersection of two convex polygons can be solved very efficiently by plane-sweep in linear time, taking advantage of the fact that the edges of the input polygons are ordered. Observe that in each vertical strip defined by two consecutive sweep-lines, we only need to compute the intersection of two trapezoids, one derived from each polygon[108].

The problem of intersecting two convex polyhedra was first studied by Muller and Preparata[108], who gave an $O(n \log n)$ algorithm by reducing the problem to the problems of intersection detection and convex hull computation. From this one can easily derive an $O(n \log^2 n)$ algorithm for computing the common intersection of n half-spaces in 3-dimensions by the divide-and-conquer method. However, using geometric duality and the concept of separating plane Preparata and Muller[108] obtained an $O(n \log n)$ algorithm for this problem, which is asymptotically optimal. There appears to be a difference in the approach to solving the common intersection problem of half-spaces in 2- and 3-dimensions. In the latter we resorted to geometric duality instead of divide-and-conquer. This *inconsistency* was later resolved. Chazelle[32] combined the hierarchical representation of convex polyhedra, geometric duality and other ingenious techniques to obtain a linear time algorithm for computing the intersection of two convex polyhedra. From this result several problems can be solved optimally: (1) the common intersection of half-spaces in 3-dimensions can now be solved by divide-and-conquer optimally, (2) the merging of two Voronoi diagrams in the plane can be done in linear time by observing the relationship between the Voronoi diagram in 2-dimensions and the convex hull in 3-dimensions (cf. Section 3.2.3), and (3) the medial axis of a simple polygon or the Voronoi diagram of vertices of a convex polygon can be solved in linear time.

3.8 Geometric Searching

This class of problems is cast in the form of query-answering as discussed in Section 2.7. Given is a collection of objects, with preprocessing allowed, one is to find objects that satisfy the queries. The problem can be static or dynamic, depending on if the database is allowed to change over the course of query-answering sessions, and it is studied mostly in modes, *count-mode* and *report-mode*. In the former case only the *number* of objects satisfying the query is to be answered, whereas in the latter the actural identity of the objects is to be reported. In the report-mode the query time of the algorithm consists of two components, *search time* and *output*, and expressed as $Q_{\mathcal{A}}(n) = O(f(n) + \mathcal{F})$, where n denotes the size of the database, $f(n)$ a function of n, and $\mathcal{F}$ the size of output. It is obvious that algorithms that handle the report-mode queries can also handle the count-mode queries ($\mathcal{F}$ is the answer). It seems *natural* to expect that the algorithms for count-mode queries would be more efficient (in terms of the order of magnitude of the space required and query time), as they need not search for the objects. However, it was argued that in the report-mode range searching, one could take advantage of the fact that since reporting takes time, the more to report, the *sloppier* the search can be. For example, if we were to know that the ratio $n/\mathcal{F}$ is $O(1)$, we could use a sequential search on a linear list. Chazelle in his seminal paper[29] on filtering search captializes on this observation and improves the time complexity for searching for several problems. As indicated below the count-mode range searching problem is harder than the report-mode counterpart.

3.8.1 Range Searching Problems

This is a fundamental problem in database applications. We'll discuss this problem and the algorithm in 2-dimensions. The generalization to higher-dimensions is straightforward using a known technique[19]. Given is a set of n points in the plane, and the ranges are specified by a product $(l_1, u_1) \times (l_2, u_2)$. We would like to find points $p = (x, y)$ such that $l_1 \leq x \leq u_1$ and $l_2 \leq y \leq u_2$. Intuitively we want to find those points that lie inside a query rectangle specified the range. This is called *orthogonal range searching*, as opposed to other kinds of range searching problems discussed below. Unless otherwise specified, a range refers to an orthogonal range. We discuss the static case, as this belongs to the class of decomposable searching problems, the dynamization transformation techniques can be applied. We note that the range tree structure mentioned below can be made dynamic by using a weight-balanced tree, called a $BB(\alpha)$ tree[97, 125].

For count-mode queries this problem can be solved by using the locus method as follows. Divide the plane into $O(n^2)$ cells by drawing horizontal and vertical line throug each point. The answer to the query q, i.e., find the number of points dominated by q, (those points whose x- and y-coordinates are both no greater than those of q) can be found by locating the cell containing q. Let it be denoted by $Dom(q)$. Thus the answer to the count-mode range-queries can be obtained by some simple arithmetic operations of $Dom(q_i)$ for the four corners of the query rectangle. We have $Q(k,n) = O(k \log n)$, $S(k,n) = P(k,n) = O(n^k)$. To reduce the space requirement at the expense of query time has been a goal of further research on this topic. Bentley introduced a data structure, called *range-trees*[19]. Using this structure the following results were obtained: for $k \geq 2$, $Q(k,n) = O(\log^{k-1} n)$, $S(k,n) = P(k,n) = O(n \log^{k-1} n)$. (see [83] and [124] for more references.)

For report-mode queries, Chazelle[29] showed that using filtering search technique the space requirement can be further reduced by a $\log \log n$ factor. In essence we use less space to allow for more objects than necessary to be found by the search mechanizm, followed by a filtering process leaving out unwanted objects for output. If the range sataifies additional conditions, e.g., *grounded* in one of the coordinates, say $l_1 = 0$, or the aspect ratio of the intervals specifying the range is fixed, less space is needed. For instance, in 2-dimensions, the space required is linear (a saving of $\log n / \log \log n$ factor) for these two cases. By using the so-called functional approach to data structures Chazelle[30] developed a *compression* scheme to encode the "downpointers" used by Willard[124] to reduce further the space requirement. Thus in k-dimensions, $k \geq 2$, for the count-mode range queries we have $Q(k,n) = O(\log^{k-1} n)$ and $S(k,n) = O(n \log^{k-2} n)$ and for report-mode range queries $Q(k,n) = O(\log^{k-1} n + \mathcal{F})$, and $S(k,n) = O(n \log^{k-2+\epsilon} n)$ for some $0 < \epsilon < 1$.

3.8.2 Other Range Searching Problems

There are other range searching problems, called the simplex range searching problem, and the half-space range searching problems that have been well-studied. A simplex range in $\Re^k$ is a range whose boundary is specifed by $k+1$ hyperplanes. In 2-dimensions it is a triangle.

The report-mode half-space range searching problem in the plane is optimally solved by Chazelle et al.[41] in $Q(n) = O(\log n + \mathcal{F})$ time and $S(n) = O(n)$ space, using geometric duality transform. But this method does not generalize to higher dimensions. For $k = 3$, Chazelle and Preparata[43] obtained an optimal $O(\log n + \mathcal{F})$ time algorithm using $O(n \log n)$ space. In [2] Agarwal and Matoušek obtained a more general result for this problem: for $n \leq m \leq n^{\lfloor k/2 \rfloor}$, with $O(m^{1+\epsilon})$ space and preprocessing, $Q(k,n) = O((n/m^{1/\lfloor k/2 \rfloor}) \log n + \mathcal{F})$. As the half-space range searching problem is also decomposable (cf. Section 2.7) standard dynamization techniques can be applied.

A general method for simplex range searching is to use the notion of *partition tree.* The search space is partitioned in a hierarchical manner using cutting hyperplanes, and a search structure is built in a tree structure. Willard[123] gave a sublinear time algorithm for count-mode half-space query in $O(n^\alpha)$ time using linear space, where $\alpha \approx 0.774$, for $k = 2$. Using Chazelle's cutting theorem Matoušek showed that for k-dimensions, there is a linear space search structure for the simplex range searching problem with query time $O(n^{1-1/k})$, which is optimal in 2-dimensions and within $O(\log n)$ factor of being optimal for $k > 2$. For more detailed information regarding geometric range searching see [90].

The above discussion is restricted to the case in which the database is a collection of points. One may consider other kinds of objects, such as line segments, rectangles, triangles, etc., depending on the needs of the application. The inverse of the orthogonal range searching problem is that of *point enclosure searching problem.* Consider a collection of isothetic rectangles. The point enclosure searching problem is to find all rectangles that contain the given query point q. We can cast these problems as *intersection searching* problem, i.e., given a set S of objects, and a query object q, find a subset $\mathcal{F}$ of S such that for any $f \in \mathcal{F}, f \cap q \neq \emptyset$. We have then the rectangle enclosure searching problem, rectangle containment problem, segment intersection searching problem, etc. We list only a few references about these problems[22, 71, 82]. Janardan and Lopez[72] generalized the intersection searching in the following manner. The database is a collection of *groups* of objects, and the problem is to find all groups of objects intersecting a query object. A group is considered to be intersecting the query object if any object in the group intersects the query object. When each group has only one object, this reduces to the ordinary searching problems.

4 Conclusion

We have covered in this chapter a wide spectrum of topics in computational geometry, including several major problem solving paradigms developed to date and a variety of geometric problems. These paradigms include incremental construction, plane-sweep, geometric duality, locus, divide-and-conquer, prune-and-search, dynamization and random sampling. The topics included here, i.e., convex hull, proximity, point location, motion planning, optimization, decomposition, intersection, and searching, are not meant to be exhaustive. Some of the results presented are classical, and some of them represent the state of the art of this field. But they may also become classical in months to come. The reader is encouraged to look up the literature in major computational geometry journals and conference proceedings given in the references. We have not discussed parallel computational geometry which has an enormous amount of reseach findings. Atallah gave a survey on this topic[12].

We hope that this treatment will provide sufficient background information about this field and that researchers in other science and engineering disciplines may find it helpful and apply some of the results to their own problem domains.

5 Further Information

We remark that there are new efforts put into the applied side of the algorithm development. A library of geometric software including visualization tools, and applications programs is under development at the Geometry Center, University of Minnesota, and a concerted effort is being put together by researhers in Europe and in the U.S. to organize a system library containing primitive geometric abstract data types useful for geometric algorithm developers and practitioners.

Those who are interested in the implementations or would like to have more information about available software, may consult the Proceedings of the Annual ACM symposium on

Computational Geometry, which has a video session, or the WWW page on *Geometry in Action* by David Eppstein (http://www.ics.uci.edu/~eppstein/geom.html).

References

[1] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf, E. Welzl, "Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs," *Discrete & Comput. Geometry.*, 6(1991), pp. 407-422.

[2] P. Agarwal and J. Matoušek, "Dynamic Half-Space Range Reporting and Its Applications," *Algorithmica*, 13,4 (April 1995), 325-345.

[3] A. Aggarwal and L. J. Guibas and J. Saxe and P. W. Shor, "A Linear-Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon," *Discrete & Comput. Geometry*, 4,6 (1989), 591-604.

[4] H. Alt and O. Schwarzkopf, "The Voronoi Diagram of Curved Objects," *Proc. 11th Annual ACM Symp. Comput. Geometry*, 1995, 89-97.

[5] H. Alt and C. K. Yap, "Algorithmic Aspect of Motion Planning: A Tutorial, Part 1 & 2," *Algorithms Review,* 1 (1990), 43-77.

[6] N. Amato, "Determining the Separation of Simple Polygons," *Int'l J. Comput. Geom. & Applications,* 4,4 Dec. 1994, 457-474.

[7] N. Amato, "Finding a Closest Visible Vertex Pair between Two Polygons," *Algorithmica,* 14,2 (August 1995), 183-201.

[8] B. Aronov and M. Sharir, "On Translational Motion Planning in 3-Space," *Proc. 10th Annual ACM on Computational Geometry*, June 1994, 21-30.

[9] Ta. Asano, Te. Asano and H. Imai, "Partitioning a Polygonal Region into Trapezoids," *J. ACM* 33,2 (April 1986), 290-312.

[10] Ta. Asano, Te. Asano and R. Y. Pinter, "Polygon Triangulation: Efficiency and Minimality," *J. Algorithms*, 7 (1986), 221-231.

[11] Te. Asano, L. J. Guibas and T. Tokuyama, "Walking on An Arrangement Topologically," *Int'l J. Comput. Geometry & Appl.*, 4,2 (June 1994), 123-151.

[12] M. J. Atallah , "Parallel Techniques for Computational Geometry," *Proceedings of IEEE*, 80,9 (Sept. 1992), 1435-1448.

[13] F. Aurenhammer, "Power Diagrams: Properties, Algorithms and Applications," *SIAM J. Comput.*, 16,1 (Feb. 1987), 78-96.

[14] F. Aurenhammer, "A New Duality Result Concerning Voronoi Diagrams," *Discrete & Comput. Geometry*, 5 (1990), 243-254.

[15] D. Avis and H. ElGindy, "Triangulating Point Sets in Space," *Discrete & Comput. Geometry*, 2 (1987), 99-111.

[16] D. Avis and G. T. Toussaint, "An Efficient Algorithm for Decomposing a Polygon into Star-Shaped Polygons," *Pattern Recognition*, 13 (1981), 395-398.

[17] F. Avnaim, J. D. Boissonnat, and B. Faverjon, "A Practical Exact Motion Planning Algorithm for Polygonal Objects amidst Polygonal Obstacles," *Proc. Geometry and Robotics Workshop*, J. D. Boissonnat and J. P. Laumond Eds., LNCS Vol. 391 (1988), 67-86.

[18] I. J. Balaban, "An Optimal Algorithm for Finding Segments Intersections," *Proc. 11th Annual Symp. Comput. Geometry,* June 1995, 211-219.

[19] J. L. Bentley, "Multidimensional Divide-and-Conquer," *Comm. ACM*, 23,4 April 1980, 214-229.

[20] J. L. Bentley and J. B. Saxe, "Decomposable Searching Problems I: Static-to-Dynamic Transformation," *J. Algorithms*, 1 (1980), 301-358.

[21] S. N. Bespamyatnikh, "An Optimal Algorithm for Closest Pair Maintenance," *Proc. 11th Annual Symp. Comput. Geometry*, June 1995, 152-161.

[22] V. Bistiolas and D. Sofotassios and A. Tsakalidis, "Computing Rectangle Enclosures," *Comput. Geometry: Theory & Appl.*, 2,6 (1993), 303-308.

[23] J.-D. Boissonnat, M. Sharir, B. Tagansky and M. Yvinec, "Voronoi Diagrams in Higher Dimensions Under Certain Polyhedra Distance Functions," *Proc. 11th Annual ACM Symp. Comput. Geometry*, 1995, 79-88.

[24] P. Callahan and S. R. Kosaraju, "Algorithms for Dynamic Closests Pair and n-Body Potential Fields," *Proc. 6th ACM-SIAM Symp. Discrete Algorithms*, 1995, 263-272.

[25] H. Bieri and W. Nef, "A Recursive Plane-Sweep Algorithm, Determining All Cells of a Finite Division of R^d," *Computing,* 28 (1982), 189-198.

[26] J. Canny and J. R. Reif, "New Lower Bound Techniques for Robot Motion Planning Problems," *Proc. 28th Annual Symp. Foundations of Computer Science,* Oct. 1987, 49-60.

[27] T. M. Chan, "Output-Sensitive Results on Convex Hulls, Extreme Points, and Related Problems," *Proc. 11th ACM Annual Symp. on Computational Geometry,* June 1995, 10-19.

[28] B. Chazelle, "How to Search in History," *Info. & Control*, 64 (1985), 77-99.

[29] B. Chazelle, "Filtering Search: A New Approach to Query-Answering," *SIAM J. Computing*, 15,3 (August, 1986) 703-724.

[30] B. Chazelle, "A Functional Approach to Data Structures and Its Use in Multidimensional Searching," *SIAM J. Computing*, 17,3 (June, 1988) 427-462.

[31] B. Chazelle, "Triangulating a Simple Polygon in Linear Time," *Discrete & Comput. Geometry*, 6 (1991), 485-524.

[32] B. Chazelle, "An Optimal Algorithm for Intersecting Three-Dimensional Convex Polyhedra," *SIAM J. Comput.*, 21,4 (1992), 671–696.

[33] B. Chazelle, "An Optimal Convex Hull Algorithm for Point Sets in Any Fixed Dimension," *Discrete & Computational Geometry*, 8 (1993), 145-158.

[34] B. Chazelle and D. P. Dobkin, "Intersection of Convex Objects in Two and Three Dimensions," *J. ACM*, 34 (1987), 1-27.

[35] B. Chazelle and H. Edelsbrunner, "An Optimal Algorithm for Intersecting Line Segments in the Plane," *J. ACM*, 39 (1992), 1-54.

[36] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir, "Diameter, Width, Closest Line Pair, and Parametric Searching," *Discrete & Computational Geometry*, 8 (1993), 183-196.

[37] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir, "Algorithms for Bichromatic Line-Segment Problems and Polyhedral Terrains," *Algorithmica*, 11,2 (Feb. 1994), 116-132.

[38] B. Chazelle and J. Friedman, "A Deterministic View of Random Sampling and Its Use in Geometry," *Combinatorica*, 10,3 (1990), 229-249.

[39] B. Chazelle and J. Friedman, "Point Location among Hyperplanes and Unidirectional Ray-Shooting," *Comput. Geom. Theory Appl.* 4 (1994) 53-62.

[40] B. Chazelle and L. J. Guibas, "Fractional Cascading: I. A Data Structuring Technique," *Algorithmica,* 1 (1986) 133-186.

[41] B. Chazelle and L. J. Guibas and D. T. Lee, "The Power of Geometric Duality," *BIT*, 25 (1985), 76–90.

[42] B. Chazelle and J. Matoušek, "On Linear-Time Deterministic Algorithms for Optimization Problems in Fixed Dimension," *Proc. 4th ACM-SIAM Symp. Discrete Algorithms*, (1993), 281-290.

[43] B. Chazelle and F. P. Preparata, "Halfspace Range Search: An Algorithmic Application of k-Sets," *Discrete & Computational Geometry*, 1 (1986), 83-93.

[44] L. P. Chew, "Constrained Delaunay Triangulations," *Algorithmica*, 4,1 (1989), 97-108.

[45] Y.-J. Chiang and R. Tamassia, "Dynamic Algorithms in Computational Geometry," *Proceedings of IEEE*, 80,9 (Sept. 1992), 1412-1434.

[46] K. L. Clarkson, "Linear Programming in $O(n3^{d^2})$ Time," *Info. Proc. Lett.,* 22 (1986), 21-24.

[47] K. L. Clarkson, "A Randomized Algorithm for Closest-Point Queries," *SIAM J. Comput.,* 17,4 (August 1988), 830-847.

[48] J. C. Culberson and R. A. Reckhow, "Covering Polygons is Hard," *Proc. 29th Annual IEEE Symp. Found. Comput. Sci.*, 1988, 601-611.

[49] D. P. Dobkin and D. G. Kirkpatrick, "A Linear Algorithm for Determining the Separation of Convex Polyhedra," *J. Algorithms*, 6 (1985), 381-392.

[50] D. P. Dobkin and J. I. Munro, "Optimal Time Minimal Space Selection Algorithms," *J. ACM*, 28,3 (July 1981) 454-461.

[51] S. E. Dorward, "A Servey of Object-Space Hidden Surface Removal," *Int'l J. Computational Geometry & Applications* 4,3 (Sept. 1994), 325-362.

[52] D. Z. Du and F. K. Hwang, Computing in Euclidean Geometry, Eds., World Scientific Publishing Co., Singapore, 1992.

[53] M. E. Dyer, "Linear Programs for Two and Three Variables," *SIAM J. Comput.*, 13, 1984, 31-45.

[54] M. E. Dyer, "On a Multidimensional Search Technique and Its Applications to the Euclidean One-Center Problem," *SIAM J. Comput.*, 15, 1986, 725-738.

[55] H. Edelsbrunner, "Computing the Extreme Distances between Two Convex Polygons," *J. Algorithms*, 6 (1985) 213-224.

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

[57] H. Edelsbrunner and L. J. Guibas, "Topologically Sweeping an Arrangement," *J. Comput. Syst. Sci.,* 38 (1989), 165-194; Corrigendum in 42 (1991), 249–251.

[58] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, "Optimal Point Location in a Monotone Subdivision," *SIAM J. Computing,* 15,2 May 1986, 317-340.

[59] H. Edelsbrunner, J. O'Rourke and R. Seidel, "Constructing Arrangements of Lines and Hyperplanes with Applications," *SIAM J. Comput.*, 15 (1986), 341-363.

[60] H. Edelsbrunner and W. Shi, "An $O(n \log^2 h)$ Time Algorithm for the Three-Dimensional Convex Hull Problem," *SIAM J. Comput.,* 20,2, (April 1991), 259-269.

[61] S. Fortune, "A Sweepline Algorithm for Voronoi Diagrams," *Algorithmica*, 2 (1987), 153-174.

[62] S. Fortune, "Computational Geometry," in Directions in Computational Geometry, R. Martin, Ed., Information Geometers, 1993.

[63] S. K. Ghosh and D. M. Mount, "An Output-Sensitive Algorithm for Computing Visibility Graphs," *SIAM J. Comput.* 20,5 (Oct. 1991), 888-910.

[64] L. J. Guibas and J. Hershberger, "Optimal Shortest Path Queries in a Simple Polygon," *J. Comput. Syst. Sci.*, 39 (1989) 126-152.

[65] L. J. Guibas and R. Seidel, "Computing Convolutions by Reciprocal Search," *Discrete & Comput. Geometry*, 2,2 (1987) 175-193.

[66] G. Y. Handler, and P.B. Mirchandani, Location on Networks : Theory and Algorithm, MIT press 1979.

[67] J. Hershberger and S. Suri, "Efficient Computation of Euclidean Shortest Paths in the Plane", *34th Symp. on Foundations of Computer Science*, 1993, 508- 517.

[68] J. M. Ho, C. H. Chang, D. T. Lee and C. K. Wong, "Minimum Diamter Spanning Tree and Related Problems," *SIAM J. Comput.,* 20,5 (October 1991), 987-997.

[69] M. E. Houle and G. T. Toussaint, "Computing the Width of a Set," *IEEE Trans. Pattern Anal. Mach. Intelligence,* PAMI-10,5 (May 1988), 761-765.

[70] H. Imai and Ta. Asano, "Efficient Algorithms for Geometric Graph Search Problems," *SIAM J. Comput*, 15,2 (May 1986), 478-494.

[71] H. Imai and Ta. Asano, "Dynamic Orthogonal Segment Intersection Search," *J. Algorithms,* 8 (1987), 1-18.

[72] R. Janardan and M. Lopez, "Generalized Intersection Searching Problems," *Int'l J. Comput. Geom. & Appl.*, 3,1 (March 1993), 39-69.

[73] S. Kapoor and M. Smid, "New Techniques for Exact and Approximate Dynamic Closest-Point Problems," *SIAM J. Computing* to appear.

[74] J. M. Keil, "Decomposing a Polygon into Simpler Components," *SIAM J. Comput.*, 14 (1985), 799-817.

[75] D. G. Kirkpatrick and R. Seidel, "The Ultimate Planar Convex Hull Algorithm?" *SIAM J. Comput.,* 15,1 (Feb. 1986), 287-299.

[76] R. Klein, Concrete and Abstract Voronoi Diagrams, LNCS Vol. 400, Springer-Verlag, 1989.

[77] V. B. Le and D. T. Lee, "Out-of-Roundness Problem Revisited," *IEEE Trans. Pattern Analysis & Machine Intelligence*, 13, 3 (March 1991), 217-223.

[78] D. T. Lee, "Proximity and reachability in the plan," Ph.D. Thesis, Tech. Rep. R-831, Coordinated Sciece Lab., Univ. of Illinois, Urbana, IL, 1978.

[79] D. T. Lee and R. L. Drysdale, III, "Generalization of Voronoi Diagrams in the Plane," *SIAM J. Computing,* 10,1 (Feb. 1981), 73-87.

[80] D. T. Lee and A. K. Lin, "Generalized Delaunay Triangulation for Planar Graphs," *Discrete & Comput. Geometry*, 1 (1986), 201-217.

[81] D. T. Lee and A. K. Lin, "Computational Complexity of Art Gallery Problems," *IEEE Trans. Info. Theory*, 32 (1986), 276-282.

[82] D. T. Lee and F. P. Preparata, "An Improved Algorithm for the Rectangle Enclosure Problem," *J. Algorithems*, 3 (1982), 218-224.

[83] D. T. Lee and F. P. Preparata, "Computational Geometry: A Survey," *IEEE Trans. Comput.,* C-33 (Dec. 1984), 1072–1101.

[84] D. T. Lee and Y. F. Wu, "Geometric Complexity of Some Location Problems," *Algorithmica,* 1, 1986, 193-211.

[85] D. T. Lee and V. B. Wu, "Multiplicative Weighted Farthest Neighbor Voronoi Diagrams in the Plane," *Proc. Int'l Workshop on Discrete Mathematics and Algorithms*, Hong Kong, Dec. 1993,, pp. 154-168.

[86] D. T. Lee, C. D. Yang and T. H. Chen, "Shortest Rectilinear Paths among Weighted Obstacles," *Int'l J. Comput. Geometry & Applications*, 1,2, 1991, 109-124.

[87] D. T. Lee, C. D. Yang and C. K. Wong, "Rectilinear Paths among Rectilinear Obstacles," *Perspectives in Discrete Applied Math.*, K. Bogart, Ed., to appear.

[88] T. Lozano-Pérez, "Spatial planning: A Configuration Space Approach," *IEEE Trans. Comput.,* C-32 (1983), 108-120.

[89] H. G. Mairson and J. Stolfi, "Reporting and Counting Intersections Between Two Sets of Line Segments," *Proc. Theoretical Foundations of Computer Graphics and CAD,* Vol. F40, Springer-Verlag, 1988, 307-325.

[90] J. Matoušek, "Geometric Range Searching," *ACM Computing Survey,* 26 (1994), 421-461.

[91] J. Matoušek, M. Sharir and E. Welzl, "A Subexponential Bound for Linear Programming," *Proc. 8th Annual ACM Symp. Comput. Geometry*, 1992, 1-8.

[92] N. Megiddo, "Applying Parallel Computation Algorithms in the Design of Serial Algorithms," *Journal of ACM*, 30 (1983), 852-865.

[93] N. Megiddo, "Linear Time Algorithm for Linear Programming in R^3 and Related Problems," *SIAM J. Comput.*, 12,4 (Nov. 1983), 759-776.

[94] N. Megiddo, "Linear Programming in Linear Time when the Dimension is Fixed," *J. ACM*, 31, 1984, 114-127.

[95] N. Megiddo, "Towards a Genuinely Polynomial Algorithms for Linear Programming," *SIAM J. Comput.*, 12 (1983), 347-353.

[96] N. Megiddo, "New Approaches to Linear Programming," *Algorithmica*, 1 (1986), 387-394.

[97] K. Mehlhorn, Data Structures and Algorithms, Vol. 3, Multi-dimensional Searching and Computational Geometry, Springer-Verlag, 1984.

[98] J. S. B. Mitchell, 'Shortest Paths among Obstacles in the Plane", *Proc. 9th ACM Symp. on Comput. Geometry*, May 1993, 308-317.

[99] J. S. B. Mitchell, D. M. Mount and C. H. Papadimitriou, "The Discrete Geodesic Problem," *SIAM J. Comput.*, 16,4 (August 1987), 647-668.

[100] J. S. B. Mitchell and C. H. Papadimitriou, "The Weighted Region Problem: Finding Shortest Paths through a Weighted Planar Subdivision", *J. ACM*, 38,1, January 1991, 18-73.

[101] J. S. B. Mitchell, G. Rote and G. Wōginger, "Minimum Link Path among Obstacles in the Planes," *Algorithmica*, 8 1992, 431-459.

[102] K. Mulmuley, Computational Geometry: An Introduction through Randomized Algorithms, Prentice Hall, Englewood Cliffs, NJ, 1994.

[103] J. Nievergelt and F. P. Preparata, "Plane-Sweep Algorithms for Intersecting Geometric Figures," *Comm. ACM*, 25 (1982) 739-747.

[104] J. O'Rourke, Art Gallery Theorems and Algorithms, Oxford University Press, New York, NY, 1987.

[105] J. O'Rourke, Computational Geometry in C, Cambridge University Press, New York, NY, 1994.

[106] J. O'Rourke and K. J. Supowit, "Some NP-Hard Polygon Decomposition Problems," *IEEE Trans. Inform. Theory*, IT-30 (1983), 181-190,

[107] M. H. Overmars, The Design of Dynamic Data Structures, LNCS Vol. 156, Springer-Verlag, 1983.

[108] F. P. Preparata and M. I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, 1985.

[109] J. Ruppert and R. Seidel, "On the difficulty of Triangulating Three-Dimensional Non-convex Polyhedra" *Discrete & Comput. Geometry*, 7 (1992), 227-253.

[110] B. F. Schaudt and R. L. Drysdale, "Multiplicatively Weighted Crystal Growth Voronoi Diagrams," *Proc. 7th Annual ACM Symp. Comput. Geometry*, 1991, 214-223.

[111] C. Schwartz, M. Smid, and J. Snoeyink, "An Optimal Algorithm for the On-Line Closest-Pair Problem," *Algorithmica*, Vol. 12, No. 1, July 1994, pp. 18-29.

[112] D. Y. Seo and D. T. Lee, "On the Complexity of Bicriteria Spanning Tree Problems for a Set of Points in the Plane," Technical Report, Dept. EE/CS, Northwestern University, June 1995.

[113] M. Sharir, "Intersection and Closest-Pair Problems for a Set of Planar Discs," *SIAM J. Computing*, 14,2 (May 1985), 448-468.

[114] M. Sharir, "On Shortest Paths amidst Convex Polyhedra," *SIAM J. Comput.*, 16,3 (June 1987), 561-572.

[115] T. C. Shermer, "Recent Results in Art Galleries," *Proceedings IEEE*, 80,9 (Sept. 1992), 1384-1399.

[116] S. Sifrony and M. Sharir, "A New Efficient Motion Planning Algorithm for a Rod in Two-Dimensional Polygonal Space," *Algorithmica*, 2 (1987) 367-402.

[117] M. Smid, "Maintaining the Minimal Distance of a Point Set in Polylogarithmic Time," *Discrete & Comput. Geometry.*, 7(1992), pp. 415-431.

[118] S. Suri, "On Some Link Distance Problems in a Simple Polygon," *IEEE Trans. Robotics and Automation,* 6 (1) 1990, 108-113.

[119] K. Swanson, D. T. Lee, and V. L. Wu, "An Optimal Algorithm for Roundness Determination on Convex Polygons," *Comput. Geometry: Theory & Appl.*, 5,4 Nov. 1995, 225-235.

[120] G. T. Toussaint, Computational Geometry, Ed., North-Holland, 1985.

[121] A. F. Van der Stappen and M. H. Overmars, "Motion Planning amidst Fat Obstacle," *Proc. 10th Annual ACM on Computational Geometry*, June 1994, 31-40.

[122] J. van Leeuwen and D. Wood, "Dynamization of Decomposable Searching Problems," *Info. Proc. Lett.*, 10 (1980), 51-56.

[123] D. E. Willard, "Polygon Retrieval," *SIAM J. Comput.*, 11 (1982), 149-165.

[124] D. E. Willard, "New Data Structures for Orthogonal Range Queries," *SIAM J. Computing*, 14,1 (Feb. 1985) 232-253.

[125] D. E. Willard and G. S. Lueker, "Adding Range Restriction Capability to Dynamic Data Structures," *J. ACM*, 32 (1985), 34-43.

[126] F. F. Yao, "Computational Geometry," in Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity, J. van Leeuwen, Ed., 1994, 343-389.

[127] C. K. Yap, "An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments" *Discrete & Comput. Geometry,* 2 (1987), 365-393.

[128] C. K. Yap, "Algorithmic Motion Planning," in Advances in Robotics, Vol I: Algorithmic and Geometric Aspects of Robotics, J. T. Schwartz and C. K. Yap, Eds., Lawrence Erlbaum, London, 1987, 95-143.