PROXIMITY STRUCTURES FOR MOVING OBJECTS IN CONSTRAINED AND UNCONSTRAINED ENVIRONMENTS

A DISSERTATION
SUBMITTED TO THE PROGRAM IN SCIENTIFIC COMPUTING AND COMPUTATIONAL MATHEMATICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Menelaos I. Karavelas
August 2001

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Leonidas J. Guibas
Computer Science Department
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Ron Fedkiw
Computer Science Department

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani
Computer Science Department

Approved for the University Committee on Graduate Studies:

$$\pi \approx 3.14159265358979323846264$$

Ἀεί ὁ Θεός ὁ Μέγας Γεωμετρεί τό κύκλου μῆκος ἵνα Ὁρίσῃ διαμέτρῳ Παρήγαγεν ἀριθμόν ἀπέραντον καί ὅν φεῦ οὐδέποτε ὅλον θνητοί θά εὑρῶσι.

Dedicated to my parents Ioannis and Maria
and to my bother Dimitris

Αφιερώνεται στους γονείς μου Ιωάννη και Μαρία
και στον αδερφό μου Δημήτρη

Abstract

Proximity structures are of central importance in Computational Geometry. They appear in several applications such as robotics, motion planning, collision detection and also in simulations of virtual and physical systems. In the real world, in most of these applications the geometric objects involved are often moving. It is thus essential to be able to maintain in an efficient way proximity structures for moving objects.

In this thesis we deal with two kinds of proximity structures: sparse spanner graphs and Voronoi diagrams. We prove that bounded aspect ratio triangulations in two and three dimensions are spanner graphs. We also show that both conforming two-dimensional bounded aspect ratio triangulations and the Constrained Delaunay triangulation are spanner graphs.

Using the Kinetic Data Structures (KDS) framework, we show how to maintain near neighbors for moving points in two and three dimensions when the underlying structure is the Voronoi diagram. In the more realistic case of a set of possibly intersecting disks moving on the plane we show how to maintain the Euclidean Voronoi diagram. Using then the Voronoi diagram as a basis, we describe how to maintain the closest pair of the set of disks, a spanning sub-graph of the connectivity graph of the set of disks and near neighbors of disks.

Finally, we discuss the problem of handling kinetic simulations of geometric objects whose motion is represented as polynomials of high degree. In this setting, the moments in time that a change happens in the combinatorial structure of the attribute of interest are roots of polynomials of high degree. We present an algorithm that speeds up the kinetic simulations by representing the roots of the afore-mentioned polynomials as intervals, instead of computing them explicitly.

Acknowledgements

First of all, I would like to thank my advisor Leonidas J. Guibas for his constant support, help and encouragement throughout the years that I was working towards my Ph.D. Working with him has been a very rewarding experience on several levels. I am grateful to him.

Many thanks are due to Rajeev Motwani, Ron Fedkiw, Oussama Khatib and Leon Simon for being in my reading and oral committee. I would like to thank them for their advice and patience.

Many thanks to several members of Leonidas' group for their valuable help. They would always allocate some time to discuss with me, read my papers and provide helpful and constructive feedback. In particular, I would like to thank Julien Basch, João Comba, Olaf Hall-Holt and Li Zhang. Julien and Li saved me a lot of time by providing me their error-free kinetic data structures source code based on which I implemented my own kinetic data structures.

Andrew Stuart and Leonidas are probably the best two teachers that I met at Stanford. I had the opportunity and privilige to be a student and a teaching assistant for both. Their organization inside and outside the classroom are remarkable and their ability to trasmit knowledge is amazing. Although I never had the chance to work on the research level with Andrew, I would like to express my graditude for his encouragement and belief in me during my years at Stanford.

John Gerth has a special place in my heart. Not only was he there to solve any problems with my computer, but also, on numerous occasions, spent lots of time explaining how this or that worked. I would also like to thank Jutta, Evelyn, Hoa, Ada and Heather for being there and helping me resolve all sorts of administrative issues.

Stanford was not only a place to do graduate work, but also to meet new people and make new friends. Some of them are still here, while some others are gone. All of them, however, made life here much more pleasant and fun. In particular, I would like to thank Aris G., Artemis E., Athina M., Chrysoula T., Costas S., Costis M., Dimitris P., George C., George G., Maria G., Nikos F., Persa K., Phaedon K., Tasos G., Vasilis V., Yiannis K. and Yiannis O.

Finally, I would like to thank my parents and my brother who guided me in their own way throughout these past few years. They were always encouraging me to continue and finish up what I had started. I thank them for their love and constant support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Geometric objects that move with time appear in many applications. These applications include motion planning, geometric modeling, computer simulations of physical or virtual systems, robotics, collison detection, computer graphics and animation, mobile communications and ad hoc networks. The common aspect of all these applications is that the behavior of the geometric objects depends on their nearby environment. The aim is to be able to answer questions concerning proximity information among the geometric objects in an efficient and accurate way. In particular, we are interested in: (1) identifying geometric structures that can give us the wanted proximity information and (2) maintaining promixity structures in an efficient manner, as the geometric objects are moving.

Proximity information that may be of interest in the applications mentioned above is the closest pair of a set of geometric objects or the near neighbors of a reference object. In other cases we are interested in reporting reachability between pairs of objects that are static or moving. Shortest paths or approximate shortest paths between static or moving geometric objects are also of interest. Structures that can provide such promiximity information are called *Proximity Structures*. Such structures are the *closest pair* of a set of objects, the *nearest neighbor* of an object, *sparse spanner graphs*, the *Voronoi diagram*, or its dual the *Delaunay triangulation*, and others.

Proximity structures are of central importance in Computational Geometry. However, most of the work so far has focused on proximity structures for static geometric objects, or in the *dynamic* setting, in which we are interested in data structures and algorithms for efficiently inserting and deleting static geometric objects. However, in real world, objects are moving or deforming. The changes are mostly continuous, in contrast to the dynamic setting. In this thesis we focus on identifying classes of geometric structures that can provide proximity information and on studying and maintaining proximity structures for continuously moving objects.

1.2 Identifying Proximity Structures

As we mentioned in the previous section proximity structures are of central importance in Computational Geometry. One natural question then arises. What/which is a good proximity structure? Of course the answer depends on the application in mind.

For example, if we are interest in the extent of a set of geometric objects, we might want to know the farthest pair since then we can compute a bounding sphere for our objects. Ideally, in this context, we would like to know the convex hull of our set. In several applications, like collision detection, behavioral simulations or physics based simulations the critical information is the information about the nearby environment. In this context knowing the closest pair of the set of objects, or the objects that are within a certain distance from a reference object is really important. In mobile communications we are often interested in whether a node A in the network can be reached from another node B. We might also be interested in knowing the shortest path in the network between A and B or an approximate shortest path between these two nodes.

In this thesis we present several structures that can provide proximity information, such as near neighbors, approximate shortest paths, closest pair and connectivity between objects. In addition, we present how to maintain efficiently most of these structures as the geometric objects are moving.

1.3 Event-Driven Simulations

In real world applications involving geometric objects that are moving the motions of the objects may be complex. The representations of the motions may change throughout time, or may even be implicit. However, the typical characteristic of these motions is that they are continuous in time. Therefore, in order to design efficient algorithms that deal with moving geometric objects, we need to take advantage of this *temporal coherence* in motion.

The traditional approach taken in simulations of moving objects is the *time-discretization* approach. The time axis is discretized. At each time step the positions of the objects are computed and then the structure of interest is re-computed from scratch. The major drawback of this approach is that it does not take easily advantage of the temporal coherence in motion. Due to temporal coherence, we would expect the structure of interest not to change too much between two instances in time, and hence we waste computation time when re-computing the structure from zero. In this context there have been several attempts to use information about the objects' motion from the previous time step in order to compute the structure of interest at the current time step [6, 13, 21, 37, 39]. However, this approach is not systematic and does not have any guarrantees. Another major problem of the time-discretization approach is the choice of the time step. The fastest moving objects are typically the ones that impose the choice of the time step: the time step must be very fine, so at not to miss critical changes in the structure.

Another approach taken to handle motion is the *dimension-lifting* approach. The time axis in this approach is simply one more dimension for the problem. Hence, instead of solving a problem for moving objects in d dimensions, we solve a static problem in $d+1$ dimensions [18, 25, 47]. The first step is to convert the problem from d dimensions to $d+1$ dimensions, and then solve it. The problem with this approach stems mainly from the potential complexity of the motions. If the motions are complex, then also the $(d+1)$-dimensional objects will be complex. In addition, this approach assumes full knowledge of the objects' motion throughout time and cannot account for motion changes midstream. Finally, the method is not applicable in situations where we know the objects' motion implicitly.

Finally, we have the *event-driven* approach. In this approach the times of critical

events, that change the structure of interest, are computed, and we advance the simulation to the nearest, in time, critical event. In this thesis we are focusing on such event driven simulations and we present algorithms that take advantage of temporal coherence in the interest of maintaining proximity structures.

1.4 Kinetic Data Structures

Basch, Guibas and Hershberger [7, 8] proposed the *Kinetic Data Structures* (KDS) framework as a systematic way to design and analyze event-driven simulations and maintenance of geometric structures for moving objects. The main idea is that the correctness of a structure can be verified through a set of conditions. These conditions, which are called *certificates*, can be thought of as the choices that an algorithm makes to construct the geometric structure. Clearly, the certificates are functions of the geometric objects. If the objects are moving the certificates become functions of time and can be expressed as inequalities of the form $F(t) > 0$. For example, consider a set of n sorted (distinct) real numbers $x_1(t), \ldots, x_n(t)$. The $O(n)$ conditions that verify that the set of numbers is sorted are the conditions $x_i(t) > x_{i+1}(t)$, $i = 1, \ldots, n-1$. Equivalently, these conditions can be written as $x_{i+1}(t) - x_i(t) > 0$.

As long as the certificates remain valid, i.e., as long as the inequalities of the form $F(t) > 0$ remain true, we know that the combinatorial structure of interest is correct. When a certificate fails, we need to replace it by new certificates and get a new proof of correctness for the structure. Such a certificate failure is called an *event*. The idea in KDS is that we maintain this proof of correctness as time elapses. The critical events in the event-driven simulation are the times that the certificates fail and the purpose and aim of the KDS is to efficiently update the set of certificates when critical events happen. In contrast to the fixed-time methods, the KDS approach does not suffer from the dependance on the worst behaving object. The events that are being processed are the ones that are naturally suited to the problem addressed.

In principle every algorithm can be kinetized, using the choices that the algorithm makes when computing the structure of interest. However, such an approach may not result in efficient kinetic data structures. A KDS is considered well designed when it is of small space, can be updated efficiently when certificates fail or the objects'

motions are changed, and when it processes few events with respect to the number of combinatorial changes that the structure of interest undergoes as the objects are moving. We are going to discuss kinetic data structures and their efficiency issues later on in Section 2.3.

1.5 Motion Complexity

Typically, in Kinetic Data Structures the certificates are polynomial functions of the motions of the geometric objects. Thus, if the geometric objects move along polynomial trajectories, then the kinetic certificates are nothing but polynomial inequalities. Although in most cases the certificates are low degree functions of the motions of the objects, the motions themselves may be described as high degree polynomials, which suggests that the polynomials involved in the kinetic certificates are also of high degree.

Finding the roots of polynomials is not a trivial task, and for polynomials of degree greater than 5 there are no closed form solutions. Therefore, for the purposes of our kinetic simulations we may need to compute the roots of high degree polynomials in order to resolve the ordering of the critical events of the simulation. Unfortunately, root finding for polynomials typically involves the computation of eigenvalues of matrices and the existing algorithms can only provide us with both the complex and real roots of a real polynomial at the same time. The natural question that then arises is whether we can do better than computing all the roots of a polynomial in order to perform the time comparisons required by the event queue. Moreover, what if we do not want to use numerical techniques in order to compute these roots, but rather stay within a numerical error free symbolic environment?

The key observation with respect to this question is that, for the purposes of determining the sequence of events in a kinetic simulation, we do not really need to know the exact values of the critical event times, but rather their sequence. In addition, we would like to spend more computing resources for the event times that are in the near future than those that are far away.

1.6 Results and Contributions

This thesis is concerned with two basic classes of problems. The first one is existence results in the theory of sparse spanner graphs and the second is results in the theory of kinetic Voronoi diagrams.

Spanner graphs are, roughly speaking, graphs that provide good approximations to dense graphs with respect to path metrics. In this thesis we prove that bounded aspect ratio triangulations of point sets in two and three dimensions are spanner graphs with respect to the complete Euclidean graph (Section 3.1). We also prove that, given a *Planar Straight-Line Graph* (PSLG) in two dimensions, bounded aspect ratio constrained and conforming triangulations of the PSLG are spanner graphs with respect to the PSLG's visibility graph (Section 3.2). Finally, we prove that the Constrained Delaunay triangulation is a spanner graph with respect to the corresponding visilibility graph (Section 3.3).

In the theory of Kinetic Voronoi diagrams we design and analyze algorithms for the maintenance of near neighbors of moving points in two and three dimensions, using the Delaunay triangulation of the point set as the underlying structure (Section 4.2). The same idea applies also to the maintenance of the k-nearest neighbors of moving points again in two and three dimensions. We also show how to maintain the Constrained Delaunay triangulation using the KDS framework (Section 4.1). Using the CDT as the base structure, we can also maintain near neighbors of points when obstacles in the environment are present (Section 4.3). Finally, the CDT can also be used to maintain the relative convex hull of a set of points moving inside a simple polygon (Section 4.5).

In addition, we describe how to kinetize of the Euclidean Voronoi diagram for moving disks on the plane. The main contribution in this case is that the disks are allowed to intersect (Section 5.3). Using the Euclidean Voronoi diagram for disks, or its dual the Delaunay triangulation, we have devised algorithms for maintaining the closest pair of a set of planar moving disks as well as for maintaining a spanning subgraph of the connectivity graph of the set of moving disks (Sections 5.5 and 5.6). Finally, we present how to use the Euclidean Voronoi diagram for (non-intersecting) disks in order to maintain the near neighbors of a reference disk, or the k-nearest neighbors of a reference disk (Sections 5.7).

As mentioned in Section 1.5, computations in kinetic simulations can be very costly when the geometric objects are moving along trajectories that are high-degree polynomial functions of time. In this context, we present an algorithm for speeding-up these computions. The key observation that we exploit is that we do not really need the actual values of the event times, but rather their sequence. The certificates in this case are polynomial inequalities, and the event times are the real roots of these polynomials. The idea is to represent the event times by bounding intervals. Then we simply perform the needed event time comparisons using the bounding intervals. If the information encoded in the intervals does not permit us to decide the relative ordering of the event times, we refine the intervals appropriately until we acquire the needed ordering information (Chapter 6).

1.7 Related Work

1.7.1 Spanner Graphs

Let G be a connected n-vertex graph with arbitrary positive edge weights. A subgraph G' is a *t-spanner* if the length of the shortest path between any two vertices in G' is at most t times the length of the shortest path between the two vertices in G. The value t is the *stretch factor* associated with G'. Sparse spanner graphs have appeared as the underlying graph structure in communication networks [4, 43], as well as in biology [5]. They have also been of interest to computational geometers in the context of points in Euclidean spaces.

In particular, Dobkin, Friedman and Supowit [17] proved that the Euclidean Delaunay triangulation of a planar point set is a spanner graph with respect to the complete graph. The stretch factor that they found was approximately 5.08. Later, Keil and Gutwin [32] improved the stretch factor to approximately 2.42. Chew [14] proved that the Delaunay triangulation with repsect to the L_1 distance is also a spanner graph, and the corresponding stretch factor is $\sqrt{10}$. Finally, Chew [15] showed that the Delaunay triangulation with respect to a convex (triangle) distance metric is a spanner graph and the stretch factor in this case is 2.

Das and Joseph [16] proved that Euclidean graphs that satisfy a certain condition are spanner graphs. As a corollary they showed that the *greedy triangulation* and the

minimum weight triangulation are spanner graphs. They also presented an algorithm for constructing Euclidean linear size spanner graphs, the total edge weight of which is only a constant factor away from the weight of the minimum spanning tree of the corresponding point set. Levcopoulos and Lingas [36] also considered the problem of constructing Euclidean linear size spanner graphs the total edge weight of which is a constant factor away from the weight of the minimum spanning tree of the point set. In their paper they show how to construct spanner graphs with stretch factor $(1+\frac{1}{r})\frac{2\pi}{3\cos\frac{\pi}{6}}$, the total edge weight of which is at most $2r+1$ times the weight of the minimum spanning tree. Here the approximation parameter r is a positive rational. Their algorithm runs in linear time if the Delaunay triangulation is given. Keil [31] proposed what is known as θ-graphs. These are spanner graphs of linear size, which may have edge crossings. The stretch factor for these graphs is a function of the input angle θ, and in particular it is equal to $(\cos\theta(1-\tan\theta))^{-1}$.

Finally, Althöfer et al. [2] considered the more general problem of constructing a spanner graph with respect to a given graph with arbitrary positive weights. The measure of quality of the spanner graph does not only depend on the stretch factor, but also on its size and the ratio of the total edge weight of the spanner graph over the weight of the minimum spanning tree of the reference graph. In particular, they showed how to compute, in polynomial time, a $(2t+1)$-spanner of the original graph, the size of which is $O(n^{1+1/t})$ and its total weight is at most $1+n/2t$ times the weight of the minimum spanning tree of the reference graph. If the given graph is planar, they showed how to compute, in polynomial time, a $(2t+1)$-spanner, the size of which is at most $(n-1)(1+1/t)$ and the total weight of which is at most $(1+1/t)$ the weight of the minimum spanning tree of the reference planar graph.

1.7.2 Kinetic Voronoi Diagrams

There has been a tremendous amount of research and results in the theory of static Voronoi diagrams. The interested reader should refer to the survey paper by Aurenhammer [3] and the book by Okabe, Boots, Sugihara and Chiu [41].

In the theory of kinetic Voronoi diagrams, Guibas, Mitchell and Roos [22] showed how to kinetize the Euclidean Voronoi diagram for a set of points moving on the plane. They also proved a non-trivial upper bound on the number of combinatorial

changes that the Voronoi diagram can undergo, as the points are moving along pseudo-algebraic trajectories. The bound on the number of changes is roughly cubic on the number of points. Guibas and Zhang [24] described how to maintain the *Power diagram*, or its dual the *regular triangulation* for a set of non-intersecting moving disks on the plane. The Power diagram is the Voronoi diagram for a set of disks in which the distance of points on the plane from the disks is measured in the power metric. Zhang in his thesis [52] showed how to maintain the Power diagram for a set of non-intersecting spheres in three dimensions. Guibas, Snoeyink and Zhang [23] described how to maintain a compact version of the Voronoi diagram for a set of non-intersecting polygons moving on the plane. Finally, Gavrilova and Rokne [19] presented algebraic conditions as to when four non-intersecting moving disks have a common tangent disk and describe without many details how to maintain the Euclidean Voronoi diagram for a set of non-intersecting disks moving on the plane. However, their approach seems to work only in the case that the disks are roughly of the same size, and they did not provide any bounds on the number of combinatorial changes that the Voronoi diagram undergoes, as the disks are moving.

Chapter 2

Preliminaries

2.1 Models of Motion and Computation

There are many ways to represent motion. The position of a moving object can be represented explicitly as an analytic function of time. It can be represented implicitly as the solution of a differential equation, or even using a statistical model. In this thesis we deal with geometric objects whose motion is known explictly. In particular, we assume that geometric object move along *pseudo-algebraic* trajectories. A family of m unary functions $f_1(x), f_2(x), \ldots, f_m(x)$ are called pseudo-algebraic functions of degree d, if for any m-variate polynomial of degree d' the function $h(x) = g(f_1(x), f_2(x), \ldots, f_m(x))$ is either 0, or has at most dd' roots. It is easy to verify that polynomial or rational functions of constant degree are pseudo-algebraic functions. Moreover, given that certificates in KDSs are low degree functions of the objects' motion, the pseudo-algebraicity assumption ensures that a certificate will only fail a constant number of times throughout time. This is observation greatly helps our analysis and is the sole reason for choosing this motion model.

The computational model that we use is based on the *Algebraic Computational Tree* model (see [44, Section 1.4] for a brief introduction). For our purposes we need to augment the ACT model with additional capabilities. In particular, in addition to the usual arithmetic operations, we require that finding roots of polynomials of constant degree can be done in constant time. This enables us to compute failure times of certificates in constant time.

2.2 Envelopes and Davenport-Schinzel Sequences

One of the analysis tools that are used in this thesis are results on the complexity of the lower (upper) envelope of a set of pseudo-algebraic functions. Let $\mathcal{F} = \{f_i\}$ be a set of n pseudo-algebraic functions of maximum degree n. Then the lower envelope $\Gamma(\mathcal{F})$ of $\mathcal{F}$ is the function $\min_i f_i$. Correspondingly, the upper envelope of $\mathcal{F}$ is the function $\max_i f_i$. In the remainder of this section we only refer to lower envelopes. The results for upper envelopes are entirely analogous. The complexity of $\Gamma(\mathcal{F})$ is known to be related to the maximum length of Davenport-Schinzel sequences. A (n, s) Davenport-Schinzel sequence is a sequence composed of at most n symbols, with the additional properties that no two consecutive symbols in the sequence are the same and that any two symbols can have at most s alterations in the sequence. The length of the longest possible (n, s) Davenport-Schinzel sequence is denoted by $\lambda_s(n)$.

Let now $\mathcal{F}$ be a set of n pseudo-algebraic functions f_i of maximum degree s, defined on the entire real line. Then

Theorem 1 ([49]). *The complexity of the lower envelope $\Gamma(\mathcal{F})$ of a set $\mathcal{F}$ of pseudo-algebraic functions is $O(\lambda_s(n))$.*

If instead of pseudo-algebraic functions defined over the entire real line we have a set of n x-monotone pseudo-algebraic arcs defined on intervals of the real line, then

Theorem 2 ([49]). *The complexity of the lower envelope $\Gamma(\mathcal{F})$ of a set $\mathcal{F}$ of x-monotone pseudo-algebraic arcs is $O(\lambda_{s+2}(n))$.*

Clearly, if the functions f_i are polynomials of maximum degree s then the same results hold.

The functions $\lambda_s(n)$ are super-linear for $s \geq 3$, but glow very slowly with n. Let $\alpha(n)$ be the inverse function of the Ackermann function. Then

Theorem 3 ([49]). *The maximum length $\lambda_s(n)$ of a (n, s) Davenport-Schinzel sequence satisfies the following relations. Here s is assumed to be a constant.*

$$\begin{aligned}
\lambda_1(n) &= n, \\
\lambda_2(n) &= 2n - 1, \\
\lambda_3(n) &= \Theta(n\alpha(n)), \\
\lambda_4(n) &= \Theta(n \cdot 2^{\alpha(n)}), \\
\lambda_s(n) &\leq n \cdot 2^{(1+o(1))\alpha(n)^{\frac{s-2}{2}}}, \qquad \textit{if } s \textit{ is even, } s > 4, \\
\lambda_s(n) &\leq n \cdot 2^{(1+o(1))\alpha(n)^{\frac{s-3}{2}} \log \alpha(n)}, \qquad \textit{if } s \textit{ is odd, } s > 4.
\end{aligned}$$

Since $\alpha(n)$ is an extremely slowly growing function ($\alpha(n)$ is almost constant for all practical values of n), $\lambda_s(n)$ is very close to a linear function for constant s. In this thesis we sometimes use $\lambda(n)$ to denote $\lambda_s(n)$ for some constant s. We also define $\beta_s(n) = \lambda_s(n)/n$, and use $\beta(n)$ to denote $\beta_s(n)$ for some constant s.

2.3 The KDS Framework

Let us consider a set of geometric objects that are moving. Every object is assumed to have posted a *flight plan*, which provides information about its current motion. The geometric objects are assumed to move in a continuous way. However, they are allowed to change their motion description throughout time. Such a flight plan change is called an *update*, and it can occur because of interactions between the object and the environment or because of interactions with other moving objects.

The aim is to maintain a geometric attribute, which we are going to refer to as *configuration function*. The basic idea in the Kinetic Data Structures (KDS) framework is that the correctness of the configuration function can be established through a set of conditions. These conditions can be thought of as the choices that an algorithm makes for constructing the configuration function. The choices can be expressed as inequalities, which are typically low-degree algebraic sign conditions, and typically depend on the positions of only a small number of geometric objects.

We are going to call these inequalities *certificates*.

Once we allow the geometric objects to move, then the certificates become functions of time. As long as the sign conditions do not change sign, the certificates remain valid, and so does the configuration function. Although the geometric structure of the configuration function changes in a continuous way, the combinatorial structure of the configuration function changes only at discrete points in time. In particular, the only points in time that the combinatorial structure of the configuration function could possibly change are the times when the certificates change sign. We call these times the *kinetic events*. The kinetic events can be computed using the flight plans that the objects, involved in the corresponding certificate, have posted. Once a certificate fails, then we have to update the set of certificates and possibly update the configuration function. If the flight plan of an object changes, then we need to recalculate the times of sign change of all the certificates that the object is involved into.

Kinetic Data Structures are very similar to sweep-line or -plane techniques used extensively in Computational Geometry. In our case the dimension being swept over is time. In order to process the kinetic events we maintain an event queue on the certificates that verify the correctness of our configuration function. The priorities of the certificates are their *failure times*, i.e., the times they change sign. When a kinetic event takes place we need to update the event queue by deleting some of the certificates, inserting some new ones or updating the priorities of others. When flight plan of a geometric object is updated we have a similar scenario: the priorities of all the certificates involving the object, whose flight plan changed, need to be updated as well.

There are several measures to analyze and evaluate a KDS. First of all, we are interested in the cost of processing a single kinetic event. This cost is at least $\Omega(\log n)$, since whenever a kinetic event happens we need to insert and/or delete certificates in the event queue, or update the priorities of certificates in the event queue. If the cost of processing a single kinetic event is small, our KDS is called *responsive*. By small we mean that the cost of updating a single event is $O(\text{Polylog}(n))$ or $O(n^\epsilon)$, for every $\epsilon > 0$.

Moreover, we want kinetic data structures that are of low *cost*. The cost of a KDS

is defined as the worst-case number of events that we have to process when the motions of the geometric objects are pseudo-algebraic functions of time. We distinguish between the events that we need to process in the following way. The events that are related to actual changes in the combinatorial structure of the configuration function are called *external events*. These are events that we *must* process in order to maintain the configuration function correctly. There is also another type of events, the *internal events*, which are events that the KDS processes for its own internal needs, and do not correspond to changes in the combinatorial structure of the configuration function. Our aim is to develop kinetic data structures for which the worst-case total number of events processed is asymptotically of the same order, or slightly larger than, the worst-case number of external events. By slightly larger we mean that the ratio of the worst-case total number of events over the worst-case number of external events must be $O(n^\epsilon)$, for any $\epsilon > 0$. A kinetic data structure that has this property is called *efficient*.

Another measure of optimality of kinetic data structures is their *size*. By size we define the maximum number of events that the event queue contains at any time. A kinetic data structure is called *compact* is its size is nearly linear in the number of moving objects.

Finally, the *degree* of a kinetic data structure is the maximum number of events in the event queue that depend on a single object. This measure is crucial when we want to handle flight plan updates efficiently. A KDS is called *local* is the degree is polylogarithmic in the number of moving objects.

Chapter 3

Spanner Graphs

Let G be a connected n-vertex graph with arbitrary positive edge weights. A subgraph G' is a *t-spanner* if for any pair of vertices, their distance in G' is at most t times longer than their distance in G. The distance between two vertices in a graph is defined as the length of the shortest path in the graph between the two vertices. The value t is the *stretch factor* associated with G'. Sparse spanners are of particular interest to us for nearest neighbor queries. Given a reference point in a graph, we can perform a breadth first search on the associated spanner and prune the search using the current distance along the spanner and the known stretch factor. In the physical world, where motion is invariably present, we may be interested in maintaining nearest neighbors of certain or all the nodes as the underlying graph evolves over time. Indeed, the behavior of many physical or social systems can be modeled in terms of short-range interactions between the nodes, containment of some nodes by other groups of nodes, etc.

In this chapter we deal with the relationship between bounded aspect ratio triangulations and spanner graphs for a set of geometric points. First, we show that bounded aspect ratio triangulations in two and three dimensions are spanners with respect to the complete graph induced by the Euclidean distance between the points. Second, we extend the notion of spanners for environments with obstacles. More specifically, if G is a planar straight-line graph (PSLG), then the visibility graph $\mathcal{V}(G)$ of G is the graph that consists of all the edges of G, as well as all the edges

between points in G that do not properly intersect edges of G. Using $\mathcal{V}(G)$ we can define what is called the *geodesic distance* between two points in G, which is the length of the shortest path in $\mathcal{V}(G)$ between the two points. We show that any bounded aspect ratio triangulation that conforms with G is a spanner. Finally, we show that the Constrained Delaunay Triangulation (CDT) is also a spanner, with respect to the geodesic distance.

3.1 Fat Triangulations are Spanners

3.1.1 Triangulations in Two Dimensions.

Let abc be a triangle and let h be its longest side (hypotenuse) and v the corresponding height. The *aspect ratio* of abc is typically defined to be $A(abc) = h/v$ [12], a quantity that is always at least $2\sqrt{3}/3 \geq 1$. There exist other definitions for the aspect ratio of a triangle, which are roughly equivalent to the one we are using in this work. It can easily be shown that if θ is the smallest angle of abc, then

$$\frac{1}{\sin\theta} \leq A(abc) \leq \frac{2}{\sin\theta}\,. \tag{3.1}$$

Let $\mathcal{T}$ be a triangulation. We define the aspect ratio $A(\mathcal{T})$ to be the maximum of the aspect ratios of the triangles in $\mathcal{T}$. If θ_{min} is the minimum angle in $\mathcal{T}$ then the bounds (3.1) hold for $A(\mathcal{T})$ and θ_{min} :

$$\frac{1}{\sin\theta_{min}} \leq A(\mathcal{T}) \leq \frac{2}{\sin\theta_{min}}\,. \tag{3.2}$$

It is plausible to expect that the edges of convex partitions of the plane all of whose faces are 'fat' (by some measure) form a spanner graph of the partition vertices. This is so because for every straight shortcut through a fat face there is a path along the face boundary whose length is larger than the length of the shortcut by at most a constant factor. The main result of this subsection is to validate a special case of this intuition, by showing that bounded aspect ratio triangulations are spanner graphs of their vertices.

Theorem 4. *Let $\mathcal{T}$ be a triangulation of a point set S, such that $A(\mathcal{T}) \leq \alpha$. If a and b are two points in S, then $d_{\mathcal{T}}(a,b) \leq 2\alpha\, d(a,b)$, where $d_{\mathcal{T}}(a,b)$ denotes the length of the shortest path in $\mathcal{T}$ between a and b, and $d(a,b)$ is the Euclidean distance between a and b.*

Proof. Let a and b be two points in S. Without loss of generality we can assume that no point of S lies on the segment ab. If ab is an edge of $\mathcal{T}$ then $d_{\mathcal{T}}(a,b) = d(a,b) \leq 2\alpha d(a,b)$.

If not, then consider the triangles T_0, T_1, ..., T_k, T_{k+1} crossed by ab. The line ab separates the points of these triangles (except a and b) into two sets that lie in different (open) half-planes w.r.t. to ab. Moreover, there exists an ordering of the edges of the t_i's crossing ab, induced by the distance of their intersection with ab from a.

We construct a path from a to b zig-zagging above and below the line ab, as follows. From a go to either one of the points of t_0 incident to a. If we are at a point that is incident to b, then go to b. If we are at a point d_i not incident to b, consider all the edges incident to d_i that cross ab. Then d_{i+1} is the endpoint incident to d_i that corresponds to the edge of maximal order with respect to the ordering induced by ab.

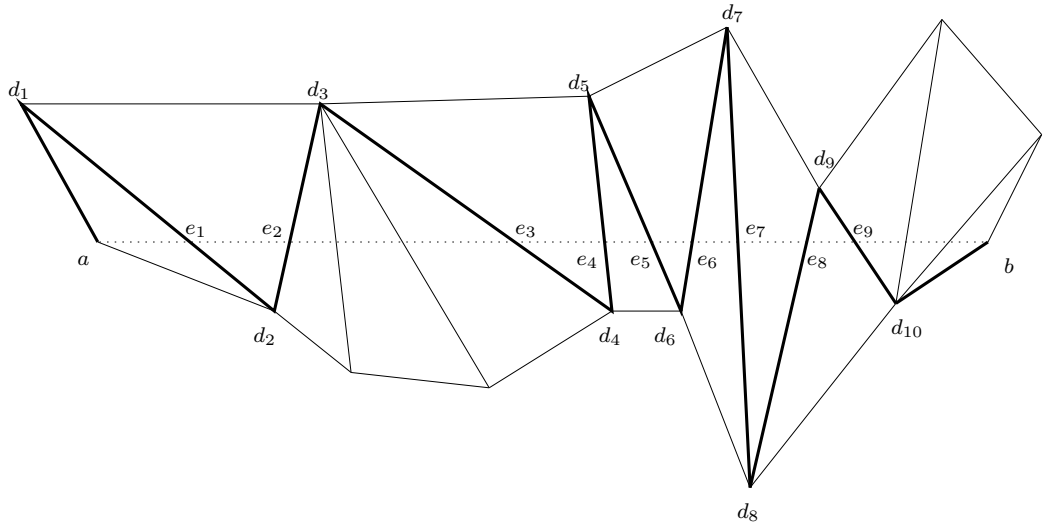


Figure 3.1: The zone of the segment ab, and the chosen path from a to b in the triangulation.

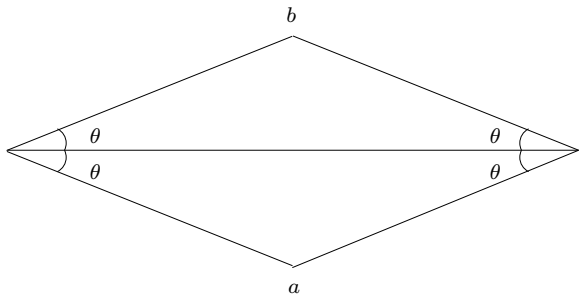


Figure 3.2: A construction that gives the lower bound for the optimal stretch factor c_{opt}.

Let $a = d_0, d_1, \ldots, d_s, d_{s+1} = b$ be the path defined above (see Fig. 3.1). This path has the property that, except at the endpoints, two consecutive vertices of the path lie on different sides of the ab. Let e_i be the intersection of $d_{i-1}d_i$ with the line ab, and let us focus on the triangle $e_i d_i e_{i+1}$. Let $\phi_i = \angle d_i e_i e_{i+1}$, $\omega_i = \angle e_i e_{i+1} d_i$ and $\theta_i = \angle e_i d_i e_{i+1}$. Clearly $\theta_{min} \leq \theta_i \leq \pi$.

If $\theta_i > \pi/2$, then

$$d(e_i, d_i) + d(d_i, e_{i+1}) \leq \frac{\pi}{2} d(e_i, e_{i+1}) \leq 2\alpha\, d(e_i, e_{i+1})\,.$$

If $\theta_i \leq \pi/2$, then using the sine law in the triangle $e_i d_i e_{i+1}$, as well as the lower bound for θ_i, we get

$$d(e_i, d_i) + d(d_i, e_{i+1}) = \frac{d(e_i, e_{i+1})}{\sin\theta_i} \left(\sin\omega_i + \sin\phi_i\right) \leq 2\alpha\, d(e_i, e_{i+1})\,.$$

Therefore,

$$d_{\mathcal{T}}(a, b) \leq \sum_{i=0}^{s} d(d_i, d_{i+1}) \leq 2\alpha \sum_{i=0}^{s} d(e_i, e_{i+1}) = 2\alpha d(a, b)\,,$$

which gives the desired result. □

Let c_{opt} be the optimal constant that bounds the ratio between the distances $d_{\mathcal{T}}(a, b)$ and $d(a, b)$. What we have just proved is that $c_{opt} \leq 2\alpha$. It is also easy to verify that $c_{opt} \geq \alpha/2$. Consider the triangulation in Fig. 3.2; the distance between the points a and b on the triangulation is $d(a, b)/\sin\theta$, which is greater than $\alpha d(a, b)/2$.

Hence :

Theorem 5. *Let c_{opt} be the optimal stretch factor for a bounded aspect ratio triangulation $\mathcal{T}$ of a two-dimensional point set S, such that $A(\mathcal{T}) \leq \alpha$. Then,*

$$\frac{\alpha}{2} \leq c_{opt} \leq 2\,\alpha\,.$$

3.1.2 Triangulations in Three Dimensions.

In three dimensions the aspect ratio of a tetrahedron is usually defined as the ratio of the radius R of the smallest containing sphere to the radius r of the largest sphere inscribed in the tetrahedron [40]. The aspect ratio $A(\mathcal{T})$ of a three dimensional triangulation $\mathcal{T}$ is defined as the maximum aspect ratio of any tetrahedron in the triangulation. An *interior angle* of the triangulation is an angle between two faces F and G where F and G are a facet and an edge, two facets, or two edges, that have a common intersection and that one face is not a subset of the other (see [40]). If θ_{min} is the minimum interior angle of the triangulation and α a bound on the aspect ratio of the triangulation, then there exist constants c_1' and c_2' such that (see [40])

$$\frac{c_1'}{\theta_{min}} \leq \alpha \leq \frac{c_2'}{\theta_{min}}\,.$$

It is easy to verify that if the above relations hold, then there exist constants c_1 and c_2, such that

$$\frac{c_1}{\sin\theta_{min}} \leq \alpha \leq \frac{c_2}{\sin\theta_{min}}\,.$$

Proving the spanner property for fat triangulations in three dimensions is more demanding. It requires two steps: first we approximate the straight line path by a path on the faces of the crossed tetrahedra, and then that latter path by another path following only the edges of the tetrahedra. The corresponding theorem is as follows:

Theorem 6. *Let $\mathcal{T}$ be a triangulation of a three dimensional point set S, such that $A(\mathcal{T}) \leq \alpha$. Then*

$$\frac{d_{\mathcal{T}}(a,b)}{d(a,b)} \leq \beta^2, \quad \beta = \max\{\frac{2\alpha}{c_1}, \frac{\pi}{2}\}\,,$$

where a, b are points in S, $d_{\mathcal{T}}(a, b)$ is the distance of the shortest path in $\mathcal{T}$ between a and b and $d(a, b)$ is the Euclidean distance between a and b.

Proof. We are going to describe a path on the tetrahedrization for which the suggested bound holds.

Consider two points $a, b \in S$ and consider all the triangles that intersect the interior of ab. The intersections of these triangles with ab induce an ordering for the set of triangles. Also, any two consecutive triangles, w.r.t. this ordering, share an edge. If more than two consecutive triangles share a common edge, we discard of all but the first and last triangle. In the remainder of the proof we shall deal with this reduced set of triangles $T_0, T_1, \ldots, T_s, T_{s+1}$.

Let $a = e_0, e_1, \ldots, e_s, e_{s+1} = b$ be the intersections of the triangles with the line ab. We can construct a two-leg polygonal path $Q_i = e_i f_i e_{i+1}$ that lies on the triangles T_i and T_{i+1}, that has the property

$$d_{Q_i}(e_i, e_{i+1}) \leq \beta\, d(e_i, e_{i+1}). \tag{3.3}$$

Let uv be the common edge of T_i, T_{i+1}. The idea is to choose f_i to be a point in uv, such that the angle $\theta_i = \angle e_i f_i e_{i+1}$ is bounded below by θ_{min}. Let Π_i, Π_{i+1} be the supporting half-planes of the triangles T_i, T_{i+1}, respectively (see Fig. 3.3). Let $e_i w e_{i+1}$ be the shortest path from e_i to e_{i+1} that lies on Π_i and Π_{i+1}. If w lies in the interior of uv (Fig. 3.3(top)), then choose f_i to be the point w. Then the angle $\theta_i = \angle e_i w e_{i+1}$ is greater or equal than the bihedral angle of Π_i and Π_{i+1}, which is greater than θ_{min}, i.e., $\theta_i \geq \theta_{min}$. If w does not lie inside uv, then we can assume without loss of generality that v is closer to w than u. Then project e_i and e_{i+1} on uv using lines parallel to vv' and vv'', respectively (Fig. 3.3(bottom)). Let e'_i and e'_{i+1} be these projections. In this case we choose f_i to be the one among e'_i and e'_{i+1} that is closer to v. Then $\theta_i \geq \angle v'vv'' \geq \theta_{min}$. Now, if $\theta_i > \pi/2$, then

$$d_{Q_i}(e_i, e_{i+1}) \leq \frac{\pi}{2} d(e_i, e_{i+1}) \leq \beta\, d(e_i, e_{i+1}). \tag{3.4}$$

If $\theta_i \leq \pi/2$, then using the sine law in the triangle $e_i f_i e_{i+1}$ and the lower bound for θ_i, we get

$$d_{Q_i}(e_i, e_{i+1}) \leq \frac{2\alpha}{c_1} d(e_i, e_{i+1}) \leq \beta\, d(e_i, e_{i+1}). \tag{3.5}$$

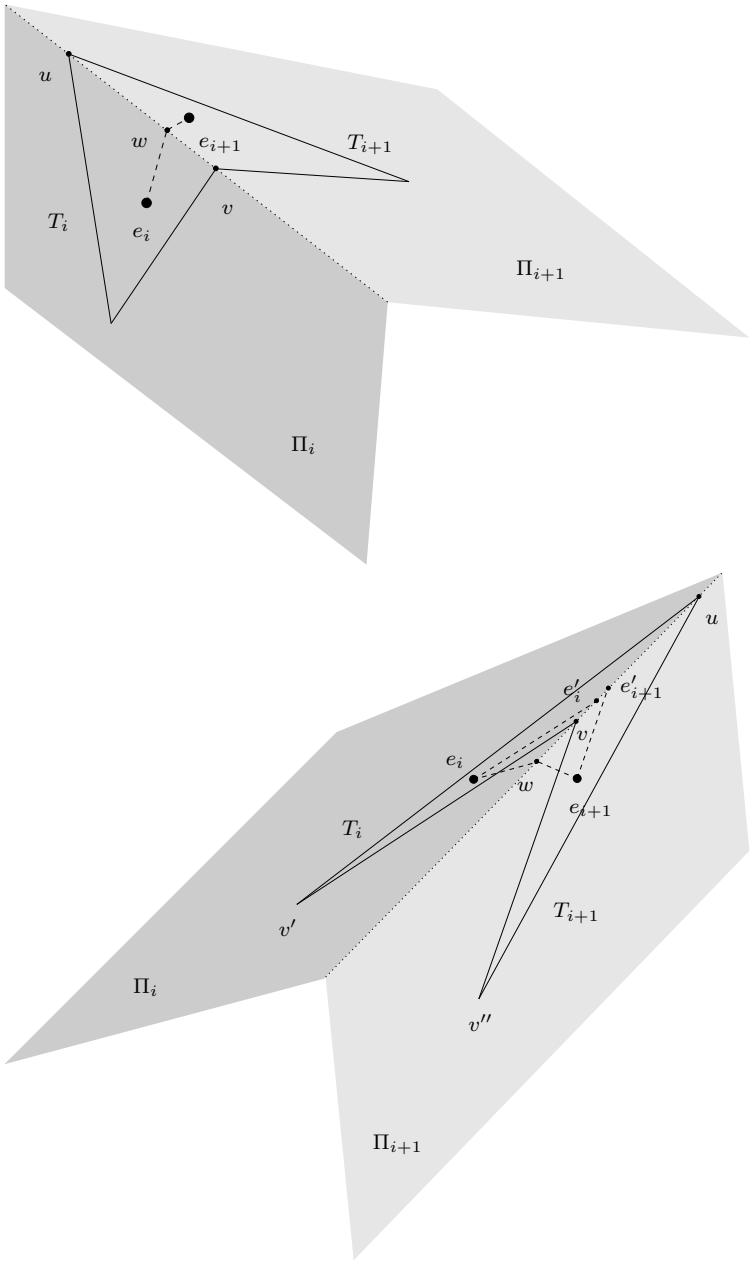


Figure 3.3: The construction of the polygonal path Q_i. Top: the shortest path between e_i and e_{i+1} crosses uv. Bottom: the shortest path between e_i and e_{i+1} does not cross uv.

Combining inequalitites (3.4) and (3.5) we establish (3.3).

Using the construction above, we have created a polygonal path Q with vertices

$a = e_0, f_1, e_1, \ldots, e_s,\ f_s,\ e_{s+1} = b$, that separates the endpoints of the edges of the triangles T_i in two disjoint sets (except for a and b), depending on which side of the polygonal path they reside (see Fig. 3.4). It also induces an ordering for the edges of the T_i's that intersect it. Construct a three-dimensional path from a to b using the edges of the T_i's as follows. From a go to either one of its two incident vertices in T_0. If we are at a point d_i that is incident to b, go to b. If we are at a point d_i not incident to b, consider all edges incident to d_i, that intersect Q. Among those edges choose the one of maximal order w.r.t. the ordering induced by Q; d_{i+1} is the endpoint of this edge incident to d_i. This construction yields a 3D path P with vertices $a = d_0, d_1, \ldots, d_k, d_{k+1} = b$, that goes back and forth across the polygonal line Q. Let $f'_1, f'_2, \ldots, f'_k$ be the subset of the f_i's corresponding to the edges $d_i d_{i+1}$, and let $f'_0 = a, f'_{k+1} = b$. Since the angles $\angle f'_i d_{i+1} f'_{i+1}$ are bounded from below by θ_{min}, we get the following bound, in exactly the same manner that we established bound (3.3) :

$$d(f'_i, d_{i+1}) + d(d_{i+1}, f'_{i+1}) \leq \beta\, d(f'_i, f'_{i+1}).$$

This in turn yields :

$$d_P(a,b) = \sum_{i=0}^{k} d(d_i, d_{i+1}) = \sum_{i=0}^{k} [d(f'_i, d_{i+1}) + d(d_{i+1}, f'_{i+1})] \leq \beta \sum_{i=0}^{k} d(f'_i, f'_{i+1}).$$

But

$$\sum_{i=0}^{k} d(f'_i, f'_{i+1}) \leq d_Q(a,b) = \sum_{i=0}^{s} d_{Q_i}(e_i, e_{i+1}) \leq \beta \sum_{i=0}^{s} d(e_i, e_{i+1}).$$

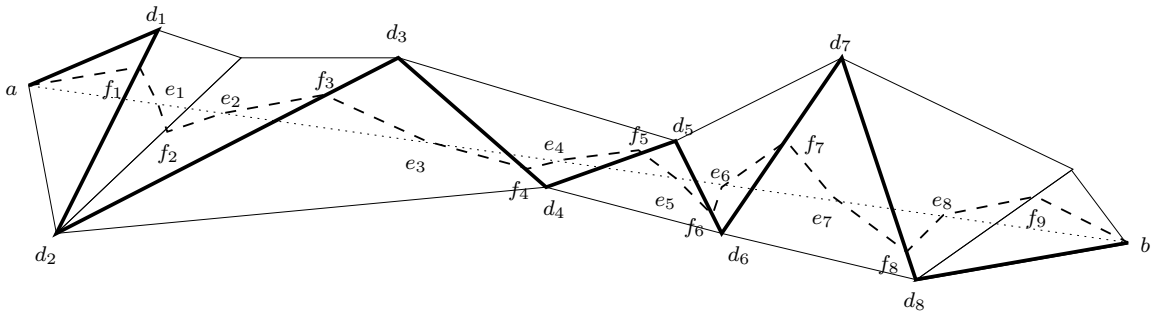


Figure 3.4: The reduced set of triangles intersecting ab and the paths Q (dashed line) and P (thick solid line).

Combining the above inequalities with (3.3) we get :

$$d_P(a,b) \leq \beta \sum_{i=0}^{k} d(f_i', f_{i+1}') \leq \beta^2 \sum_{i=0}^{s} d(e_i, e_{i+1}) = \beta^2 d(a,b)\,,$$

the result to be shown. □

We believe that similar ideas can be used to prove an analogous result for fat triangulations in any dimension.

If c_{opt} is the optimal stretch factor for bounded aspect ratio triangulations in three dimensions, we just proved that $c_{opt} \leq \beta^2$. It is easy to show that $c_{opt} \geq \alpha/c_2$. Let cde be an equilateral triangle on the xy-plane, the barycenter of which is the origin. Consider the triangulation formed by the two tetrahedra $acde$ and $bcde$, where a, b are the points $a = (0, 0, +\epsilon)$, $b = (0, 0, -\epsilon)$, where ϵ is such that the angles of the edges ac and bc with the xy-plane are θ. Let α be a bound on the aspect ratio of this triangulation. Clearly,

$$\frac{c_1}{\sin\theta} \leq \alpha \leq \frac{c_2}{\sin\theta}.$$

The length of the shortest path from a to b in this case is

$$d_{\mathcal{T}}(a,b) = d(a,c) + d(c,b) = \frac{d(a,b)}{\sin\theta} \leq \frac{\alpha}{c_2}\, d(a,b).$$

Hence :

Theorem 7. *Let c_{opt} be the optimal stretch factor for a bounded aspect ratio triangulation $\mathcal{T}$ of a three-dimensional point set S, such that $A(\mathcal{T}) \leq \alpha$. Then,*

$$\frac{\alpha}{c_2} \leq c_{opt} \leq \beta^2, \qquad \beta = \max\{\frac{2\alpha}{c_1}, \frac{\pi}{2}\}\,.$$

3.2 Environments with Obstacles

Let G be a PSLG. The graph G induces a subdivision $\mathcal{S}(G)$ of the plane into regions. Let also $\mathcal{V}(G)$ be the visibility graph associated with G. If v is a vertex of G, then we denote with F_v the set of faces of $\mathcal{S}(G)$ adjacent to v.

We focus on paths that lie entirely within one face of the subdivision $\mathcal{S}(G)$ and do not cross any constraining edges. The following definition captures these requirements (see also Fig. 3.5).

Definition 1. *A path P on the plane between two vertices u and w of G, such that $F_u \cap F_w \neq \emptyset$, is called* legal *if*

1. *the entire path P lies inside the closure of exactly one face of $\mathcal{S}(G)$.*
2. *we can find a path as close as we want to P that shares the same endpoints with P, and the interior of which lies in the interior of the same face as P.*

Definition 2. *The* geodesic distance *$d_G(u, w)$, with respect to the graph G, is the length of the shortest legal path between u and w on $\mathcal{V}(G)$, measured in the Euclidean metric.*

We call a triangulation $\mathcal{T}(G)$ *constrained* (with respect to G) if the vertices of $\mathcal{T}(G)$ are those of G and every edge in G is an edge in $\mathcal{T}(G)$. We call a triangulation *conforming* if every vertex in G is in $\mathcal{T}(G)$ and every edge in G is the union of some edges in $\mathcal{T}(G)$. Clearly a constrained triangulation is also conforming.

Theorem 8. *Let G be a PSLG and let $\mathcal{T}(G)$ be a conforming triangulation of G such that $A(\mathcal{T}) \leq \alpha$. If u and w are two vertices in G sharing a face of $\mathcal{S}(G)$, then $d_{\mathcal{T}(G)}(u, w) \leq 2\alpha\, d_G(u, w)$.*

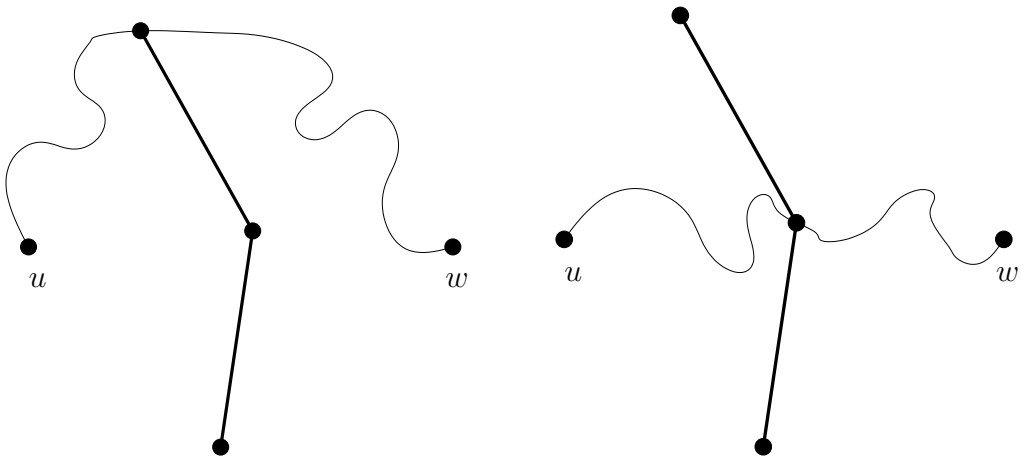


Figure 3.5: A legal (left) and a non-legal path (right).

Proof. Let $u = v_0, v_1, \ldots v_n = w$ be the sequence of vertices of G that consist of the shortest legal path in $\mathcal{V}(G)$. If $v_{k-1}v_k$ is a portion of a constrained edge, then, since $\alpha \geq 1$,

$$d_{\mathcal{T}(G)}(v_{k-1}, v_k) = d(v_{k-1}, v_k) \leq 2\alpha\, d(v_{k-1}, v_k)\,.$$

If $v_{k-1}v_k$ is not a portion of a constrained edge then consider the path from v_{k-1} to v_k described in the proof of Theorem 4. For this path we know that

$$d_{\mathcal{T}(G)}(v_{k-1}, v_k) \leq 2\alpha d(v_{k-1}, v_k).$$

Moreover, since v_{k-1} and v_k are visible from each other, it is easy to find a homeomorphism from this path to the segment $v_{k-1}v_k$, which implies that the path lies in the same face as $v_{k-1}v_k$. Therefore,

$$d_{\mathcal{T}(G)}(u, w) \leq \sum_{k=1}^{n} d_{\mathcal{T}(G)}(v_{k-1}, v_k) \leq 2\alpha \sum_{k=1}^{n} d(v_{k-1}, v_k) = 2\alpha\, d_G(u, w)\,.$$

□

3.3 The Constrained Delaunay Triangulation is a Spanner

Dobkin, Friedman and Supowit [17] have shown that the DT is a spanner graph of its vertices. The stretch factor M they proved was approximately 5.08. Later, Kiel and Gutwin [32] improved the stretch factor to approximately 2.42. It turns out that we can generalize the proof in [17] for the constrained case, and therefore show that the CDT is also a spanner, with respect to the geodesic distance — with the same stretch factor as in [17]. We begin with some definitions.

Definition 3. *The* CDT distance $CDT(u, w)$ *is the length of the shortest legal path between* u *and* w *on* $\mathcal{D}(G)$*, where* $\mathcal{D}(G)$ *is the CDT of* G*.*

Definition 4. *The* bounded Voronoi diagram *of* G *Vorb(G) is a planar subdivision that partitions the plane into regions* $R(v)$*, whwre* v *is a vertex in* G*, such that a point*

p belongs to the $R(v)$ if and only if pv is the shortest straight-line segment connecting p with a vertex of G that does not intersect any edge of G.

In order to prove that the CDT is a spanner graph, the following two properties of the bounded Voronoi diagram will be useful. Both of them have been proved in [38] and we state them here for completeness.

Lemma 1. *Let u, w be vertices of G, where $R(u)$ and $R(w)$ share an edge e in Vorb(G). Let also c be a circle passing through u and w with the center in a point p inside e. Then no vertex v of G visible from u or w is in the interior of the circle c.*

Lemma 2. *Let G be a PSLG. The straight-line dual of Vorb(G) is a subset of every CDT of G.*

We will prove our main result in the following way. If u and w are two vertices of G, we are going to find a path P from u to w on $\mathcal{D}(G)$ that is in the same face as the shortest path from to u to w on the plane. We shall then prove that the length of this path is at most M times $d_G(u, w)$, which gives the wanted result, since

$$CDT(u, w) \leq d_P(u, w) \leq M\, d_G(u, w).$$

Let $u = v_0, v_1, \ldots, v_n = w$ be the sequence of vertices of G that consist of the shortest legal between u and w path on $\mathcal{V}(G)$. If an edge $v_{k-1}v_k$ is a constrained edge then obviously it is an edge in the CDT and thus

$$CDT(v_{k-1}, v_k) = d(v_{k-1}, v_k) \leq M\, d(v_{k-1}, v_k)$$

and our result holds true for this portion of the path from a to b.

If $v_{k-1}v_k$ is not a constrained edge then v_{k-1} is visible from v_k and moreover, since $v_{k-1}v_k$ is an edge of $\mathcal{V}(G)$ there are no points of G on the segment $v_{k-1}v_k$ (since then $v_{k-1}v_k$ would not be an edge of $\mathcal{V}(G)$). For the purpose of proving our result it suffices to find a path on $\mathcal{D}(G)$ from v_{k-1} to v_k such that the inequality holds.

Let now a and b be two points of G. We assume that a and b lie on the x-axis, that $\mathbf{x}(a) < \mathbf{x}(b)$ and that a and b are the endpoints v_{k-1} and v_k of a non-constrained edge e_k of the shortest path on $\mathcal{V}(G)$ between two points on the plane. The proof

is a generalization of that in [17]. Let $a = b_0, b_1, \ldots, b_{m-1}, b_m = b$ be the vertices corresponding to the sequence of bounded Voronoi regions traversed by walking along the x-axis.

Definition 5. *The path $a = b_0, b_1, \ldots, b_m = b$ is called the* direct CDT path *from a to b.*

Let p_i be the point on the x-axis that also lies on the boundary between $Vorb(b_{i-1})$ and $Vorb(b_i)$, for $i = 1, \ldots, m$. By the definition of the bounded Voronoi diagram p_i is the center of a circle C_i passing through b_{i-1} and b_i.

Direct CDT paths have the following properties, some of which we prove here and some of which have already been proved in [17].

Lemma 3. *The segments $b_{i-1}b_i$, $i = 1, \ldots, m$ belong to the CDT of G.*

Proof. By Lemma 2, all the edges of the dual of $Vorb(\mathrm{G})$ are edges of the CDT. □

Lemma 4. $\boldsymbol{x}(p_{i-1}) \leq \boldsymbol{x}(p_i)$*, for $i = 1, \ldots, m$.*

Proof. It is a direct consequence of way the p_i's are constructed. □

Lemma 5 ([17]). $\boldsymbol{x}(b_{i-1}) \leq \boldsymbol{x}(b_i)$*, for $i = 1, \ldots, m$.*

Lemma 6. *For all $i = 0, \ldots, m$, b_i is contained within or on the boundary of* circle(a, b).

Proof. Let k be such that the midpoint c of (a, b) lies in the bounded Voronoi region of b_k. Then

$$\mathbf{x}(p_k) \leq \mathbf{x}(c) \leq \mathbf{x}(p_{k+1}) \tag{3.6}$$

and since c is in the bounded Voronoi region of b_k we have

$$d(b_{k-1}, c) \geq d(b_k, c), \qquad d(b_{k+1}, c) \geq d(b_k, c). \tag{3.7}$$

Now consider b_i, b_{i+1}, $0 \leq i \leq k-1$. Then $\mathbf{x}(p_{i+1}) \leq \mathbf{x}(c)$ and p_{i+1} is visible from both b_i and b_{i+1} and $d(b_i, p_{i+1}) = d(b_{i+1}, p_{i+1})$. But the portion of the circle with center p_{i+1} and radius $d(b_{i+1}, p_{i+1})$ to the left of $\mathbf{x}(b_i)$ lies entirely within the circle

with center c and radius $d(b_i, c)$. Hence, since b_{i+1} is on that portion of the circle, we have $d(b_{i+1}, c) \leq d(b_i, c)$. Similarly we can prove that

$$d(b_{i+1}, c) \leq d(b_i, c), \quad k \leq i \leq m-1. \tag{3.8}$$

Thus all the b_i's lie inside circle(a, b). □

Lemma 7. *If a and b are visible, then the direct CDT path between a and b lies in the same face as the segment ab.*

Proof. Let S_i be the triangle $p_i b_i p_{i+1}$. Since the entire segment $p_i p_{i+1}$ lies in the Voronoi region of b_i, we get that the triangle S_i lies in the Voronoi region of b_i, and thus it is empty of vertices or edges of G. Consider now the triangle $b_i p_{i+1} b_{i+1}$, which we call T_i. By Lemma 1, the interior of T_i does not contain any vertices of G that are visible from b_i and b_{i+1}. Suppose though that it contains a vertex v that is not visible by b_i and b_{i+1}, and let e be a constrained edge that separates v from b_i and b_{i+1}. Then e would have to cross the edges $b_i p_{i+1}$ and $p_{i+1} b_{i+1}$ of T_i, which cannot happen since p_{i+1} is visible from both b_i and b_{i+1}. Hence the interior of the triangle T_i, as well as the interior of the segments $b_i p_{i+1}$ and $p_{i+1} b_{i+1}$ are empty of vertices or edges of G. The union of the interiors of the T_i's and the S_i's, as well as the interiors of the segments $p_i b_i$ and $b_i p_{i+1}$, cover the interior of the region between the polygonal line $b_0 b_1 \ldots b_m$ and the x-axis and in fact that region is empty (see Fig. 3.6). Therefore,

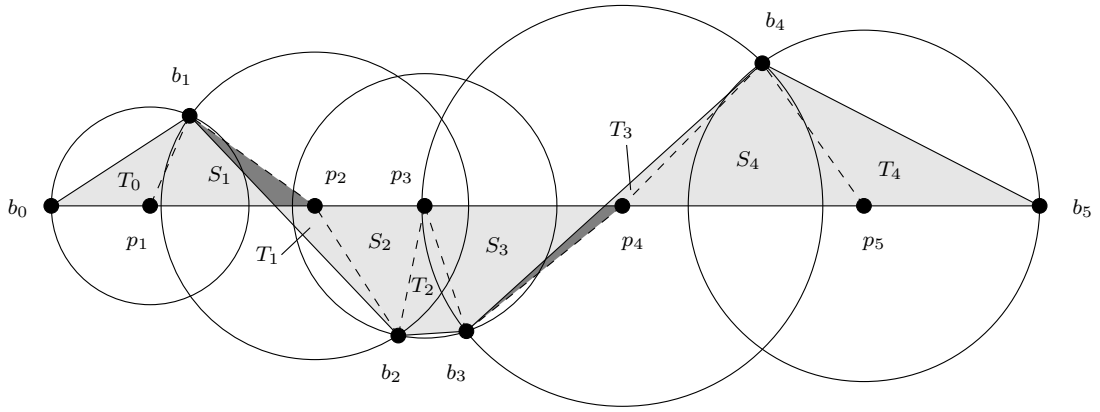


Figure 3.6: Proof of Lemma 7. The dark gray areas correspond to triangle intersections.

we can define a homeomorphism from b_0b_m to the polygonal line $b_0b_1 \dots b_m$, which implies that the direct CDT path lies in the same face as $ab \equiv b_0b_m$. □

Another lemma that will be useful is the following.

Lemma 8 ([17]). *Let $D_1, D_2, \dots, D_k$ be circles all centered on the x-axis such that $D = \bigcup_{1 \le i \le k} D_i$ is connected. Then* boundary*(D) has length at most $\pi\,(x_r - x_\ell)$, where x_ℓ and x_r are the least and greatest x-coordinates of D, respectively.*

We are now ready to prove the following theorem.

Theorem 9. *Let a, b be two points of G, that are visible from each other and no other point of G lies on the segment ab. Then there exists a CDT path from a to b of length $CDT(a, b)$, such that*

$$CDT(a, b) \le \frac{1 + \sqrt{5}}{2}\, \pi\, d(a, b)$$

Proof. Let us assume that all the b_i's happen to be above the x-axis. In this case we say that the direct CDT path from a to b is *one-sided* (see Fig. 3.7). Handling one-sided paths is easier than generic ones, since we can use Lemma 8 to bound the length of the path. In particular, for one-sided paths, the length of the path is bounded by half the length of boundary(C), that lies above the x-axis. Moreover, due to Lemma 7, the direct one-sided CDT path lies in the same face of $\mathcal{S}(G)$ as a and b. But then, due to Lemma 8, this is bounded above by $\frac{\pi}{2}\, d(a, b)$, which is consistent with our result.

The trouble arises when we do not have one-sided paths. As in [17], we try to stay above the x-axis. If the direct path dips below the x-axis, we determine how costly the dip will be. If the cost is not too expensive then we follow the direct path below the x-axis and then back up. Otherwise, we construct a shortcut between the two points above the x-axis. What remains to show is that the shortcut is not too long and that it is in the same face as the line ab.

Let $a = b_0, b_1, \dots, b_{m-1}, b_m = b$ be the direct CDT path from a to b. Assume that we are at point b_i of the path such that $\mathbf{y}(b_i) \ge 0$, $i < m$ and $\mathbf{y}(b_{i+1}) < 0$. Let j be the least number greater than i such that $\mathbf{y}(b_j) \ge 0$ (see Fig. 3.8). Let T_{ij} denote the

path along the boundary of C clockwise from b_i to b_j. Let w_{ij} denote the length of the projection of T_{ij} onto the x-axis, i.e.,

$$w_{ij} = \mathbf{x}(b_j) - \mathbf{x}(b_i), \tag{3.9}$$

and let

$$h_{ij} = \min\{\mathbf{y}(q) : q \text{ lies on } T_{ij}\}. \tag{3.10}$$

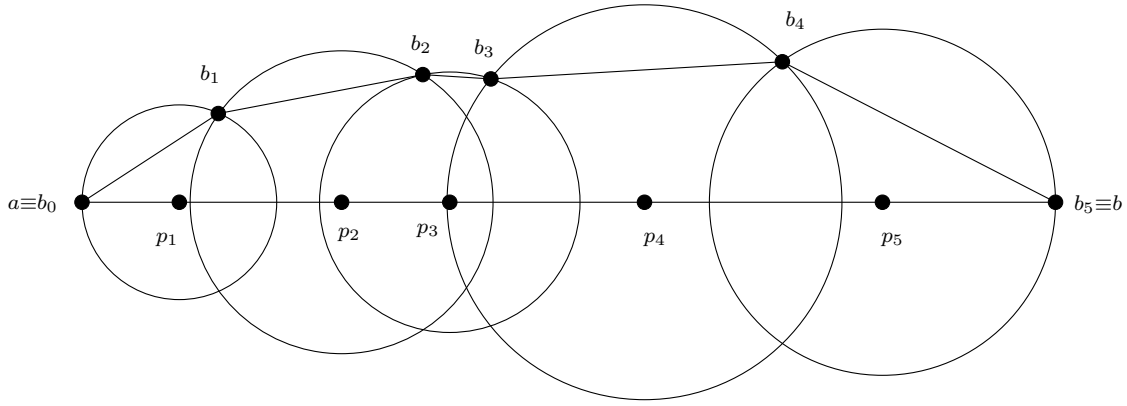


Figure 3.7: The direct CDT path from a to b is one-sided.

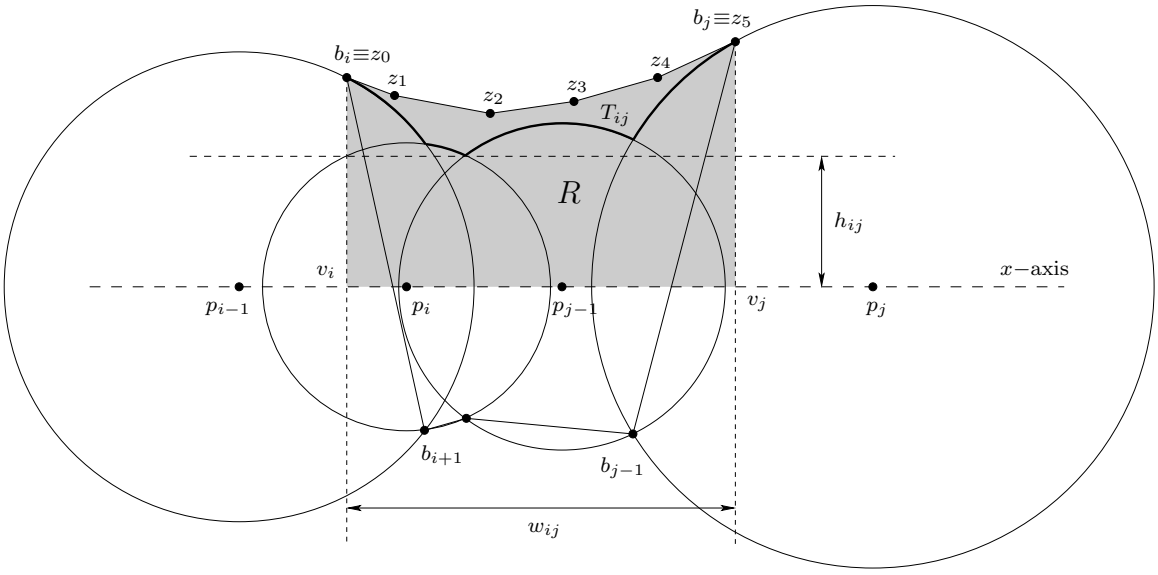


Figure 3.8: The shortcut from b_i to b_j.

Now if $h_{ij} \leq w_{ij}/4$ we continue along the direct path to b_j, otherwise we take a shortcut as follows. Construct the lower convex hull $b_i = z_0, \ldots, z_n = b_j$ of the set

$$\{q \in G : \mathbf{x}(b_i) \leq \mathbf{x}(q) \leq \mathbf{x}(b_j) \text{ and } \mathbf{y}(q) \geq 0 \text{ and } q \text{ lies under } b_i b_j\}.$$

Note that these convex hull edges are not part of the direct CDT path from a to b. The shortcut then is the path from z_k to z_{k+1}, for each $0 \leq k \leq n-1$. The key facts are the two following:

Lemma 9. *Let $z_k z_{k+1}$ be an edge of the lower convex hull described above and let z_k be visible from z_{k+1}. Then the direct CDT path from z_k to z_{k+1} is one-sided.*

Proof. The proof is identical to that of Lemma 4 in [17]. □

Lemma 10. *The direct CDT path from z_k to z_{k+1} lies in the same face as the segment ab.*

Proof. Consider the area $R_k = \{q : \mathbf{y}(q) \geq 0 \text{ and } q \text{ below } z_k z_{k+1}\}$. Let R be the union of the R_k's (shaded region in Fig. 3.8). There cannot be a vertex of G in R_k because $z_k z_{k+1}$ would not be an edge of the convex hull. Suppose that there exists a constrained segment e that separates $z_k z_{k+1}$ from the x-axis. Since the R_k's do not contain any vertices of G, the end points of e must either both be outside R or one of them is outside R and the other is one of the z_l's. In any of the two cases, since a and b are visible from each other, e must cross one of the segments $b_i v_i$ or $b_j v_j$, where v_i, v_j are the projections of b_i, b_j on the x-axis. But then e has to cross one of the segments $b_i b_{i+1}$ or $b_{j-1} b_j$, which cannot happen since they are segments of the CDT. Hence R has to be empty of vertices or edges of G. On the other hand, by Lemma 7 the direct CDT path between z_k and z_{k+1} lies in the same face as the segment $z_k z_{k+1}$. Therefore, the direct CDT path between z_k and z_{k+1} lies in the same face as ab. □

Using the two lemmas above the result then follows in exactly the same way as in [17]. □

Let's get back to our original setting. If $u = v_0, v_1, \ldots, v_n = w$ is the sequence of

vertices of G of the shortest legal path from u to w, then

$$d_G(u,w) = \sum_{k=1}^{n} d(v_{k-1}, v_k), \tag{3.11}$$

and all the segments $v_{k-1}v_k$ lie in the same face F of $\mathcal{S}(G)$. For each one of these segments we constructed a direct CDT path such that the entire path lies in F and moreover,

$$CDT(v_{k-1}, v_k) \leq \frac{1+\sqrt{5}}{2}\,\pi\, d(v_{k-1}, v_k). \tag{3.12}$$

Since

$$CDT(u,w) \leq \sum_{k=1}^{n} CDT(v_{k-1}, v_k) \tag{3.13}$$

we have proved the following.

Theorem 10. *Let u, w be two points of G that share a common face in $\mathcal{S}(G)$. Then*

$$\frac{CDT(u,w)}{d_G(u,w)} \leq \frac{1+\sqrt{5}}{2}\,\pi \tag{3.14}$$

If c_{opt} is the optimal stretch factor for the CDT, we have just shown that

$$c_{opt} \leq \frac{1+\sqrt{5}}{2}\,\pi\,.$$

It has been shown in [15] that a lower bound on the stretch factor for the DT is $\pi/2$. Since the CDT is a generalization of the DT, the same result holds for the CDT as well. Hence,

Theorem 11. *Let $c_{opt}(CDT)$ be the optimal stretch factor for the Constrained Delaunay Triangulation of a planar straight-line graph G. Then,*

$$\frac{\pi}{2} \leq c_{opt}(CDT) \leq \frac{1+\sqrt{5}}{2}\,\pi\,.$$

3.4 Conclusion

Theorems 5 and 7 provide lower and upper bounds on the optimal stretch factors for bounded aspect ratio triangulations in two and three dimensions. We do not know if these bounds are tight. It is thus an interesting open problem to devise constructions that will reduce the gap between the lower and upper bounds. Analogous results for higher dimensional bounded aspect ratio triangulations would also be desirable, and we believe that the approach described in this chapter can be used for proving such results.

The best known upper bound for the stretch factor for the Delaunay triangulation of a two-dimensional point set is [32]

$$\frac{2\pi}{3\cos(\frac{\pi}{6})}.$$

It has been conjectured by Chew [15] that the upper bound on the stretch factor of Delaunay triangulation is actually close to $\pi/2$. This is still an interesting open problem. It also of great interest to have a tighter result, than that of Theorem 11, for the Constrained Delaunay triangulation.

Three-dimenional Delaunay triangulations can be of $\Omega(n^2)$ complexity. In that respect, knowning whether the 3D Delaunay triangulation is a spanner graph is not of the same importance as in the 2D case. However, it is useful to have such a result, since in many cases the complexity of the Delaunay triangulation in three dimensions is subquadratic.

Finally, what has been presented here is a relationship between fat triangulations and spanner graphs. It is plausible that such a relationship holds for more general *fat subdivisions* of the plane. One interesting issue, for example, is whether planar subdivisions, in which the faces are of bounded aspect ratio, with respect to some measure, are spanner graphs as well.

Chapter 4

Kinetic Voronoi Diagrams and Applications

In this chapter we present how to maintain the CDT of a planar straight-line graph G using the *Kinetic Data Structures* (KDS) framework. We also deal with the problem of maintaining near neighbors of moving points. In particular, given a set of points in two or three dimensions, we show how to maintain the near neighbors of some reference points, or how to maintain the k-nearest neighbors of some reference points. We use the Delaunay triangulation of the point set as the underlying structure. On top of the DT we build an additional KDS for each one of the reference points, that keeps track of the near neighbors. Using the CDT as the underlying structure we also show how to maintain near neighbors in environments where obstables are present. Finally, we discuss how to use the CDT for maintaining the *relative convex hull* for a set of points moving inside a simple polygon.

4.1 Kinetic Constrained Delaunay Triangulation

Maintaining the Voronoi diagram, or its dual the Delaunay triangulation, for a set of points on the plane is done by exploiting the local property of the Delaunay triangulation. This property states that an edge in the triangulation is globally Delaunay if it is locally Delaunay. The certificates in this case are the `InCircle` tests for the edges of the triangulation. When a certificate fails we simply have to do an edge flip

to restore the correctness of the Delaunay triangulation [7].

In the case of the Constrained Delaunay triangulation the situation is analogous. The fact that a triangulation is the CDT can be established through a set of local conditions on the non-constrained edges of the CDT. In particular, it has been shown [11, Lemma 3] that if the non-constrained edges of the CDT are locally Delaunay then the triangulation is globally the CDT. The definition of local Delauniness is the same as in the case points but it only applies to the non-constrained edges of the triangulation.

For the sake of completeness we are going to prove that if a triangulation is locally Delaunay then it is the CDT, using a different approach than that in [11]. We start with a definition.

Definition 6. *Let $\mathcal{T}$ be a triangulation and let e be an edge in $\mathcal{T}$. Let T_1, T_2 be the triangles adjacent to e and let u, v be the endpoints of e. Finally let a, b be the vertices of T_1, T_2 that are not u or v. We say that e* passes the `InCircle` test *if and only if* `InCircle`(a, u, v, b) *is false.*

It is shown in [11, Lemma 3] that local `InCircle` tests establish the global CDT property. We prove the same result here using a different approach.

Lemma 11. *An edge uv is in the CDT if and only if for all a, b both visible from both u and v to the left and right of uv we have that* `InCircle`(a, u, v, b) *is false.*

Proof. Let $\{C_t\}$ be the one-parameter family of circles passing through u and v, where t denotes the distance of the center of C_t from the midpoint of uv. The edge uv is in the CDT if and only if there is a circle passing through u, v such that no points that are visible from u and v are in the interior of the circle. This is true if and only if every circle auv with a to the left of uv corresponds to a value of t less than of equal to that of any circle vub with b to the right of uv and a, b both visible from u and v. This proves our result. □

Using the above lemma we can now state and prove the relationship between the local and global CDT property.

Theorem 12. *A triangulation $\mathcal{T}(G)$ of a PSLG G is the CDT if and only if all the non-constrained edges of $\mathcal{T}$ pass the* `InCircle` *test.*

Proof. Let e be an edge of $\mathcal{T}$ that fails the `InCircle` test and let a and b as in Definition 6. Then it cannot be that e is an edge of the CDT, because a, b are both visible from u, v and `InCircle`(a, u, v, b) is true (see Lemma 11). Conversely, if $\mathcal{T}$ is not the CDT there exists an edge uv and two vertices a, b both visible from u, v, such that `InCircle`(a, u, v, b) is true, with a, b being on different sides of uv according to Lemma 11. Among all such quadruples choose the one for which the sum $\angle vau + \angle ubv$ is maximum. Clearly, no edge or vertex of $\mathcal{T}$ can enter the triangles auv or vub. Hence these are the triangles incident to uv and obviously uv fails the test. □

Therefore, in order to maintain the CDT we only need to check when a non-constrained edge fails its `InCircle` test; when this happens, a single edge flip restores the correctness of the CDT (see Fig. 4.1). If we assume that the moving vertices of the CDT do not hit constrained edges, then the only events are the edge flips. When such an event happens we need $O(\log n)$ time to update our KDS, i.e., the KDS for the CDT is responsive. The size of our KDS is proportional to the number of non-constrained edges, i.e., $O(n)$, and our KDS is compact. However, as in the DT case, the KDS is not local since a moving point may be associated with $\Omega(n)$ certificates. Finally, if the motions of the vertices are pseudo-algebraic, the total number of combinatorial changes in the CDT, which is also the number of events that we have to process, is $O(n^3\beta(n))$. The best lower bound on the number of

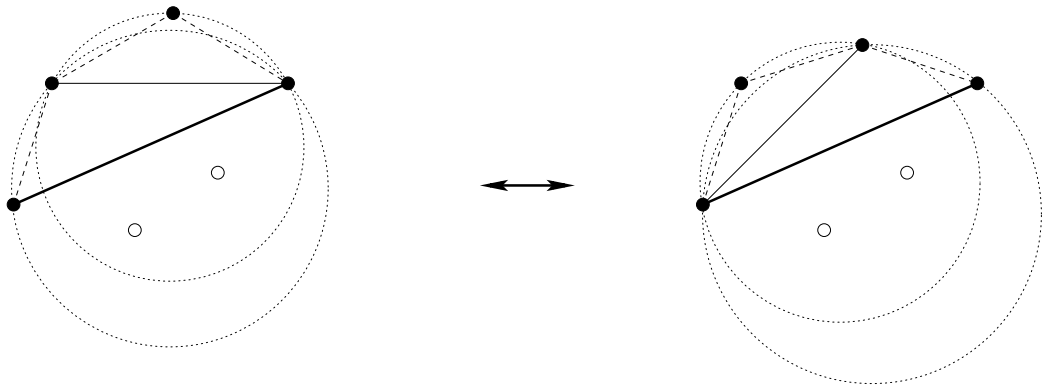


Figure 4.1: The flip-edge event. The think solid segment is a constrained edge. The thin solid line is the edge e that is flipped. The dotted circles are the circumcircles of the triangles adjacent to e. The white points are points of G not visible from e.

combinatorial changes of the DT, and thus the CDT, is $\Omega(n^2)$ [22]. Hence, the KDS presented above may not be efficient.

4.2 Near Neighbors in 2D and 3D

Suppose that we have a set V of moving points in two (three) dimensions and a point $p \in V$, for which we want to know the points in V that are within a certain distance R_p from p. The obvious approach is to maintain the distance from p to every other point in V and keep those that are within the prescribed distance. We show how to do better using the Delaunay triangulation of V. Let C_p be the ball centered at p with radius R_p. In two dimensions C_p is actually a circle, whereas in three dimensions it is a sphere. Our crucial observation is that, if we are maintaining the DT of V, the only points that enter or exit C_p are endpoints of edges of the DT crossing C_p exactly once (called crossing edges from now on). Hence, maintaining the near neighbors of p reduces to maintaining the DT and updating the set of crossing edges, whenever a point enters or exits C_p (see Fig. 4.2).

In this section we shall treat the two- and three-dimensional case together. The

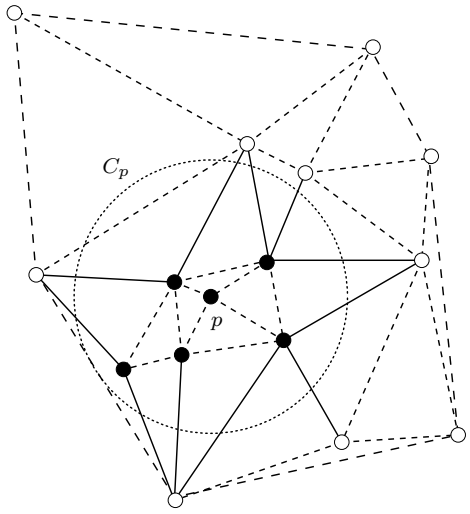


Figure 4.2: Keeping track of the points that may enter or exit C_p. The only edges that we have to look at are those that *properly intersect* C_p, i.e., the edges that cross C_p exactly once (solid edges in this figure).

two-dimensional case where obstacles in the environment are present will be addressed in the next section.

If we want to maintain the set of near neighbors for a set $S \subseteq V$ of points, we can apply the ideas described above for each one of the points in S separately. A noteworthy feature of our method is that, except for the overhead of maintaining the Delaunay triangulation, it is *motion-sensitive*: all other events processed by the structure reflect actual changes to the neighborhoods of the points of interest. Though the overhead of maintaining the Delaunay triangulation can be significant in the worst case, in practice it has nearly linear efficiency and it can be a useful piece of infrastructure for other applications as well, including clustering, communications, etc.

We shall generalize our setting in order to avoid the restriction that p has to be the center of C_p. Thus, let V be a set of points in two or three dimensions and let p be a point in V. Let $\mathcal{T}$ be a triangulation of V. The points in V are assumed to be moving. With p we associate a ball C_p, containing p, of radius R_p, which may be time varying. The ball C_p will contain the point p in its interior throughout time. In order to prove our main theorem that associates the edges of the DT with near neighbor maintenance we need the following definition.

Definition 7. *We say that an edge e of $\mathcal{T}$* properly intersects *C_p, if one endpoint of e lies outside of C_p and the other endpoint of e is inside C_p.*

The basis of the kinetization process is the following theorem, which holds true in both two and three dimensions.

Theorem 13. *Let $\mathcal{T}$ be the DT and let $p \in V$ be a point associated with a ball C_p. If a point $q \in V$ enters/exits C_p at some time t_0, then there exists an edge of $\mathcal{T}$ between q and a point inside C_p.*

Proof. At time t_0, q is on the boundary of C_p. Let $\{C_r\}$ be the family of balls with center r that pass through q, where r is a point on the segment pq. Consider the ball $C_{r'}$ such that r' is at maximal distance from q, and $C_{r'}$ contains no points of V in its interior. Note that because p is inside C_p throughtout time, such a circle $C_{r'}$ always exists. Due to the maximality of r', $C_{r'}$ touches a point $q' \in C_p$ that is inside C_p. Clearly the edge qq' is a DT edge (see Fig. 4.3). □

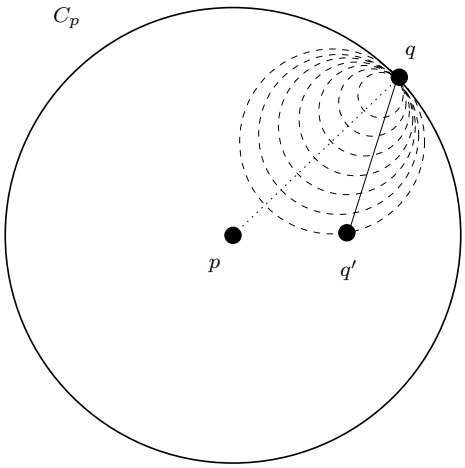


Figure 4.3: The proof of Theorem 13 in two dimensions.

4.2.1 The Kinetic Maintenance Algorithm

Let A_p be the set of points inside C_p. Let also E_p be the set of edges of the DT that properly intersect C_p. As we have already mentioned our goal is to maintain these two sets. In order to do that we have to handle two types of events:

1. events that correspond to points entering or exiting C_p,

2. events that correspond to the maintenance of the DT.

In two dimensions the DT can be maintained via edge-edge flips [7], whereas in three dimensions the DT can be maintained using face-edge or edge-face flips [30, 52].

When a point q enters C_p we have to look at q's neighbors. For those neighbors that are outside C_p we only need to add the corresponding edges to E_p. For those that are inside C_p, we need to remove the corresponding edges from the edge set E_p. Obviously, we also need to add q to the set A_p (see Fig. 4.4, from left to right).

When a point q exits C_p the situation is entirely symmetric: for all of q's neighbors that are outside C_p delete the corresponding edges from E_p. For the neighbors that are inside C_p we need to add the corresponding edges to the set E_p. Finally, we need to delete q from the set A_p (see Fig. 4.4, from right to left).

As far as the second type of event is concerned we shall distinguish between the two and three dimensional case. In two dimensions, whenever an edge-edge flip happens

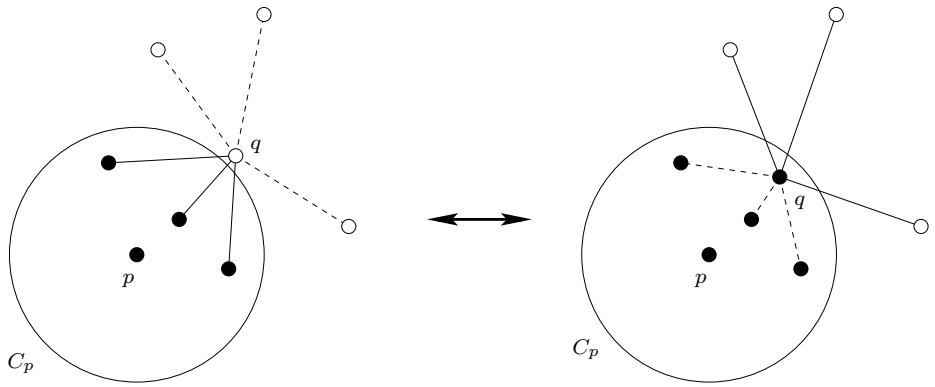


Figure 4.4: Maintaining the near neighbors of p as the point q enters (left to right) or exits (right to left) the circle C_p. The black points are in the set A_p. Solid edges belong to the set E_p.

we only have to update the set E_p. In particular, if the old edge was in E_p we need to delete it; if the new edge properly intersects C_p we need to add it to E_p. In three dimensions, if we have an edge-face flip and the edge to be deleted was in E_p, we simply need to delete it from the set E_p. If a face-edge flip is performed, we need to check if the newly created edge properly intersects C_p; if this is the case we add it to E_p.

The construction and algorithm described above can be directly generalized to any L_p metric with $1 < p < \infty$. In particular, we can maintain in exactly the same way near neighbors that are within distance R from a given point q in the L_p metric by maintaining the L_p-metric version of the DT.

4.2.2 Maintaining the k-nearest neighbors

A variant of the problem above is the one where we want to maintain the k-nearest neighbors of a point p. Suppose that we have initially computed which are these neighbors. Then the radius R_p of C_p is now the distance between p and its k-th nearest neighbor p_k – clearly in this case R_p is time varying. The set A_p now consists of all the points inside C_p including p_k. The set E_p consists of all the edges of the DT crossing C_p exactly once. The edges that are adjacent to p_k are considered to be crossing edges. In order to maintain the k nearest neighbors of p we have to handle

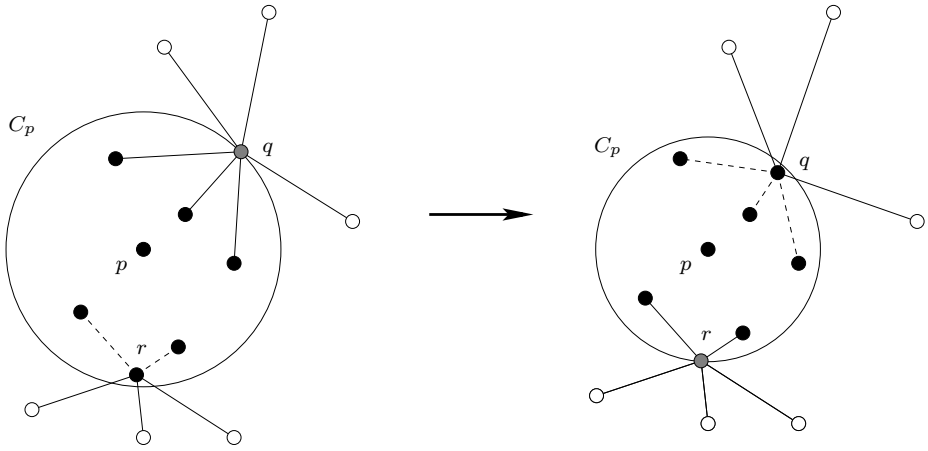


Figure 4.5: Maintaining the k-nearest neighbors of p as the $(k-1)$-th nearest neighbor becomes the k-th nearest neighbor.

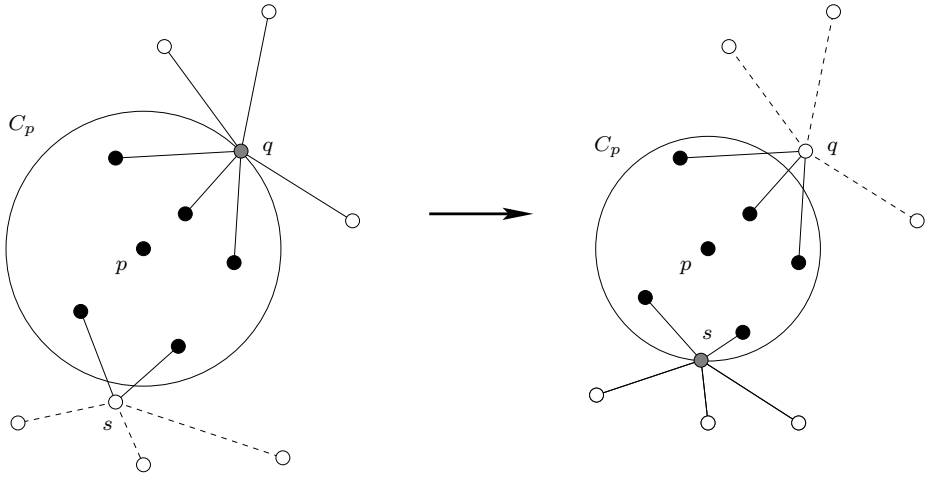


Figure 4.6: Maintaining the k-nearest neighbors of p as the $(k+1)$-th nearest neighbor becomes the k-th nearest neighbor.

three types of events:

1. the $(k-1)$-th nearest neighbor of p becomes the k-th nearest neighbor of p,
2. the $(k+1)$-th nearest neighbor of p becomes the k-the nearest neighbor of p.
3. events that correspond to the maintenance of the DT.

Let q be the k-th nearest neighbor of p and r the $(k-1)$-th nearest neighbor of p (see Fig. 4.5). When an event of the first type happens then all edges of the DT connecting r to points inside C_p are added to E_p. In addition, all edges of the DT connecting q to points inside C_p are deleted from E_p. Finally, we have to update how R_p changes with time.

Let now q be the k-th nearest neighbor of p and s be the $(k+1)$-th nearest of p (see Fig. 4.6). When an event of the second type happens we have to add s to the set A_p and remove q from it. All the edges of the DT connecting q to points outside C_p are deleted from E_p and all the edges of the DT connecting s to points outside C_p are added to E_p (see Fig. 4.6). Again, we need to update how R_p changes with time.

The third type of event is treated in exactly the same way as in our original problem.

4.3 Near Neighbors in Constrained Environments

The approach and algorithm of the preceeding section can be generalized for constrained two-dimensional environments represented as a PSLG G. A constrained edge e that intersects C_p twice is called a *blocking edge*. The points q that we keep track of are those that are inside C_p and not blocked from p by a blocking edge. It turns out that all such points can be approached from p using a path in the CDT of G that lies entirely inside C_p. Again, as in the unconstrained case, points of interest that enter or exit C_p are endpoints of edges of the CDT crossing C_p. Hence maintaining this point set of interest means maintaining the CDT, as well as maintaining the set of crossing edges.

In this section we shall precisely define the set of points that we want to maintain and prove that the CDT is a good triangulation to use to encapsulate proximity information between the points in our point set. We shall then provide the nearest neighbor maintenance algorithm, which essentially describes how to maintain the set of crossing edges described above.

Let $G(V, E)$ be a PSLG and p be a point in V. Let $\mathcal{T}(G)$ be a constrained triangulation of G. The points in V are assumed to be moving. Again, as in the unconstrained case, we associate with p a circle C_p, containing p, of radius R_p, which

may be time varying. The circle C_p will contain the point p in its interior throughout time.

Definition 8. *Let $\mathcal{T}(G)$ be a constrained triangulation of G. We call a point q in V* approachable *from p, if q is inside C_p and there exists a path from p to q in $\mathcal{T}(G)$ that lies entirely in C_p.*

Due to the existence of blocking edges, the definition of proper intersection has to be modified slightly. In particular,

Definition 9. *We say that an edge e of $\mathcal{T}(G)$* properly intersects *C_p, if one endpoint of e lies outside of C_p and the other endpoint of e is approachable from p.*

The difference here is that the point insize C_p is now required to be an approachable point instead of any point inside C_p.

The fact that the point set that we want to maintain is the set of approachable points w.r.t. the CDT is established by the following theorem.

Theorem 14. (the maximality property) *Let A be the set of points in $V \cap C_p$ that are not blocked from p by a blocking edge. Then A is the set of approachable points of p with respect to the CDT of G.*

Proof. Let $q \in A$ be a point not approachable from p. This implies that there exists an edge e with endpoints u and v, such that u, v are outside of C_p. The edge e splits C_p in two regions and p, q are in different regions. Consider the triangles that contain e, and let q' be the third vertex of the triangle that lies on the same half-space as q. Clearly e cannot be a constrained edge, since then q would not be in A. If q' is not inside C_p, then the circle passing through $q'uv$ must contain either p or q, or some other point in C_p that is visible from either u or v. This contradicts the CDT property for e. If q' is inside C_p then the circle passing through $q'uv$ must contain p or some other vertex in C_p that is visible from u or v. Again we have contradicted the CDT property for e. □

The following theorem is the basis of the kinetization process for the case of constrained environments.

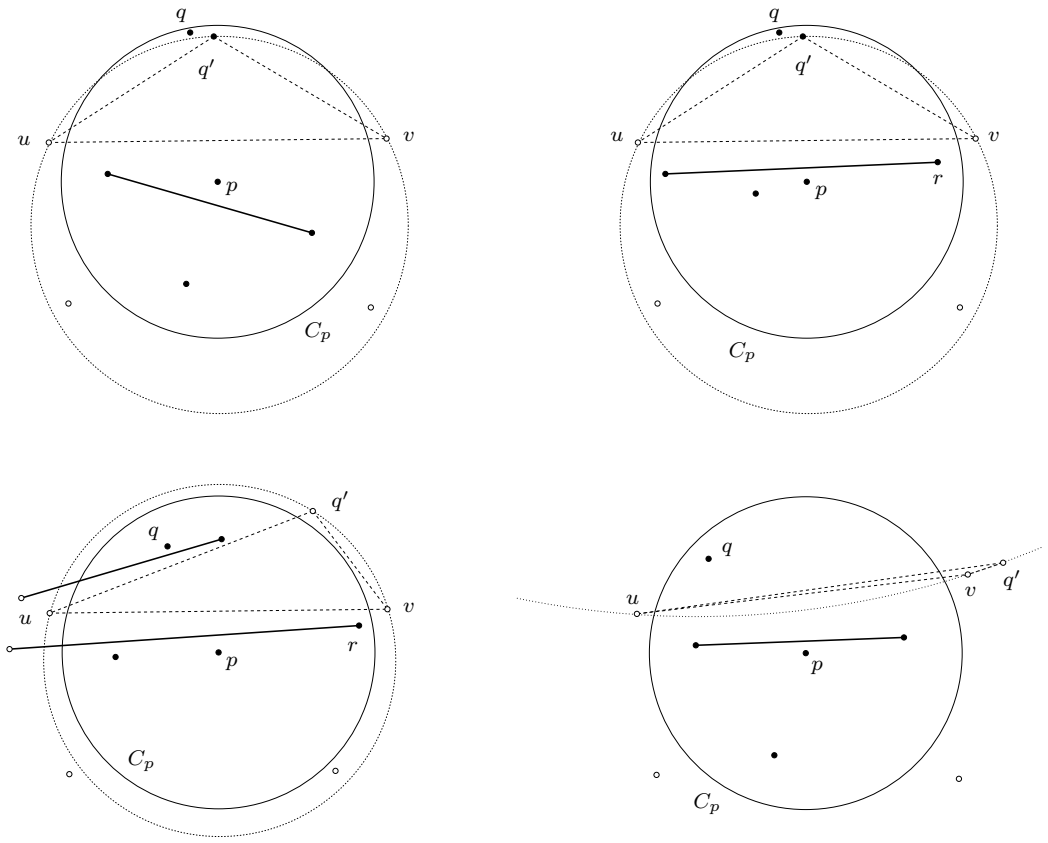


Figure 4.7: Some of the cases in the proof of Theorem 14. Top left: q' is inside C_p, the circle through $q'uv$ contains p. Top right: q' is inside C_p, the circle through $q'uv$ contains $r \neq p$; p is not visible from both u, v. Bottom left: q' is outside C_p, the circle through $q'uv$ contains $r \neq p, q$; both p and q are not visible from both u, v. Bottom right: q' is inside C_p and the circle $q'uv$ contains q but not p.

Theorem 15. *Let $\mathcal{T}(G)$ be the CDT of G and let $p \in V$ be a point associated with a circle C_p. If a point $q \in V$ enters/exits the circle C_p at some time t_0 and is visible from at least one point inside C_p, then there exists an edge of $\mathcal{T}(G)$ between q and a point inside C_p.*

Proof. At time t_0, q is on the boundary of C_p. Let $\{C_r\}$ be the family of circles with center r that pass through q, where r is a point on the segment pq. Consider the circle $C_{r'}$ such that r' is at maximal distance from q, and $C_{r'}$ contains no points of V in its interior that are visible from q. Note that because the set of points of V that

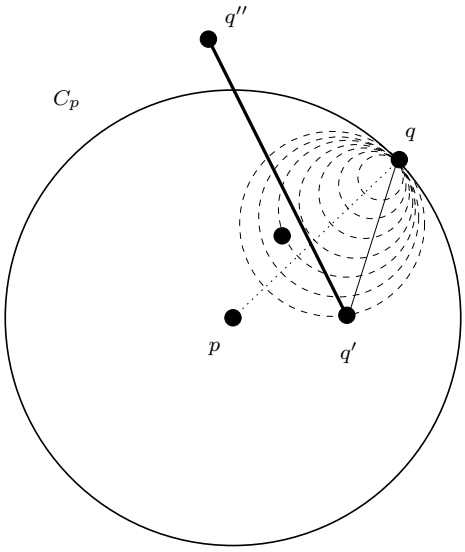


Figure 4.8: The proof of Theorem 15. The segment $q'q''$ is a constrained edge.

are visible from q at t_0 is non-empty by assumption, such a circle $C_{r'}$ always exists. Due to the maximality of r', $C_{r'}$ touches a point $q' \in C_p$ that is visible from q. Clearly the edge qq' is a CDT edge (see Fig. 4.8). □

4.3.1 The Kinetic Maintenance Algorithm

Let now A_p be the set of approachable points from p w.r.t. the CDT. Let also E_p be the set of edges of the CDT that properly intersect C_p. As we have already mentioned our goal is to maintain these two sets. In order to do that we have to handle two types of events:

1. edge flips that are required to maintain the CDT,

2. events that correspond to points entering or exiting C_p.

Whenever an edge flip happens we only have to update the set E_p. If the old edge was in E_p we need to delete it; if the new edge properly intersects C_p we need to add it to E_p.

When a point q enters C_p we have to look at q's neighbors. For those neighbors that are outside C_p we only need to add the corresponding edges to E_p. For those

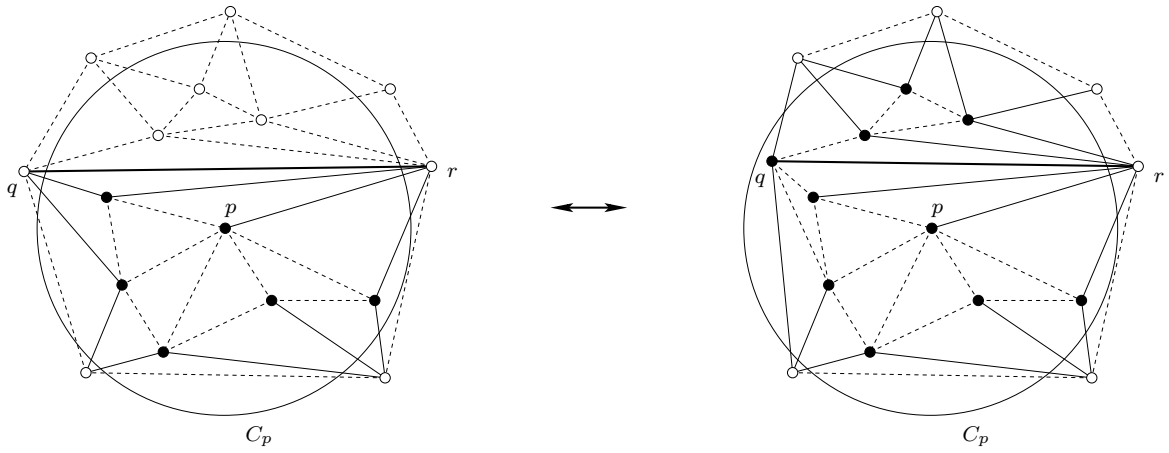


Figure 4.9: Maintaining the near neighbors of p as the point q enters (left to right) or exits (right to left) the circle C_p. The segment qr (thick solid line) is a constrained edge.

that are inside and in A_p we need to remove the corresponding edges from the edge set E_p. Finally for the neighbors that are inside but not in A_p we need to add them to the point set A_p and perform the same tests for their neighbors recursively (see Fig. 4.9, from left to right).

When a point q exits C_p the situation is entirely symmetric: for all the neighbors that are outside delete the corresponding edges from E_p. For the neighbors that are inside and remain approachable after the point exits, we need to add the corresponding edges to the set E_p. Finally as far as the remaining neighbors are concerned, we have to delete them from the set A_p of approachable neighbors, delete any edges in E_p that adjacent to them and recursively do the same for their neighbors (see Fig. 4.9, from right to left).

4.4 Cost Analysis

In this section we are going to provide a cost analysis of the algorithms presented here and compare our method with other methods for solving the same problems. Our analysis is applicable to the two-dimensional case in both constrained and unconstrained environments. We assume thoughout this section that the points in the

point set in question move along trajectories which are pseudo-algebraic functions of time.

Assume that we have a point set V of n points and that we want to maintain near neighbors for r points in V, $1 \leq r \leq n$. We are going to consider four different measures for evaluating the different methods : (1) the number of events that the KDS has to process, which is basically the efficiency of the KDS, (2) the number of certificates in the event queue, which is the compactness of the KDS, (3) the update cost per event, which is the responsiveness of the KDS, and (4) the number of certificates in which a single point participates; the maximum value is called the *locality* of the KDS.

The obvious way to maintain near neighbors of the reference points is to maintain the distances from every reference point to all other points in V. In the kinetic framework this translates to a KDS of size $\Theta(rn)$, since the number of certificates that we have to maintain per reference point is $\Theta(n)$. The cost per event is clearly $O(\log n)$ which is the cost of inserting a certificate in the event queue. The worst case number of events than this KDS has to process is $O(rn)$, which is the number of times that points enter or exit the reference circles. Finally, a single point in V appears in $\Theta(r)$ certificates, one for each reference point.

Another way of maintaining near neighbors is to maintain two *kinetic heaps* for every reference point, one for the points inside the reference circle and one for the points outside. A kinetic heap is nothing but a regular heap in which the priorities of the nodes in the heap are time varying. In our case the priorities are the squares of the distances of the points from the reference circle. Clearly, at every time instant the only points that are candidates for exiting or entering the reference circles are the roots of the two kinetic heaps. In this case, in addition to the events corresponding to points entering or exiting reference circles, we also have events associated with the kinetic heaps' maintenance. We know that the worst case number of events that a kinetic heap processes is $O(n\sqrt{n \log n})$, if the priorities of the heap nodes depent linearly on time [9]. A lower bound on the number of events is $\Omega(n \log n)$ [9]. However, for higher degree dependances, like in our case, where the priorities are second degree functions of time, we do not know an upper bound on the number of events processed. Hence the total number of events that our KDS processes in this case if $O(rn + rH)$,

where H is the worst case number of events that each kinetic heap processes. The cost per event is again $O(\log n)$, which is the time to schedule an event in the priority queue. The total number of certificates in this case is again $\Theta(rn)$, since for every reference point we have two kinetic heaps of total size $\Theta(n)$. Finally, the locality for this KDS is $\Theta(r)$ certificates, since every point participates in a constant number of certificates per reference point.

We now turn to the analysis of the KDS presented in this section of the paper. We call our method the kinetic Delaunay method. As we discussed in a previous subsection a bound on the number of combinatiorial changes of the Delaunay triangulation is $O(n^3\beta(n))$. This is the best known upper bound for the number of combinatorial changes of the DT, whereas the best known lower bound is $\Omega(n^2)$. Since the number of times that points can enter or exit the reference circles is $O(rn)$, we conclude that our KDS processes $O(rn + D)$ events in the worst case, where D is the number of Delaunay events, and $D = O(n^3\beta(n))$. Among those events, the ones that correspond to Delaunay edge-flips take $O(r+\log n)$ time to process, since we have to check whether the new edge crosses any of the reference circles, and schedule/reschedule in the event queue a constant number of certificates. However, when we check to see if the new edge crosses any of the r reference circles, we can actually restrict ourselves to only those reference circles that cross the edges of the quad in the triangulation surrounding the new edge. In practice, this implies that we need to check the new edge against many fewer than r reference circles. The events that correspond to points entering/exiting reference circles take $O(n)$ time to process. This is because we have to look at the neighbors of the point entering/exiting a reference circle, which can be $O(n)$ in the worst case. Clearly, the locality of the kinetic Delaunay method is $O(n)$.

It seems, at first, that the kinetic Delaunay method is not better, at least in the asymptotic sense from the other two methods. However, in many applications the moving points are uniformly distributed on the plane. In a situation like this, the expected degree of the vertices in the Delaunay triangulation is $O(1)$ and the expected number of events associated with combinatorial changes of the Delaunay triangulation is $O(n^{3/2})$ [53]. Hence the total number of events for the kinetic Delaunay method is $O(rn+n^{3/2})$. Moreover, if the size of the reference circles is small, w.r.t. the diameter of the point set, both the number of points inside the reference circle and the number

of crossing edges is constant. Hence the number of certificates in the event queue is $O(r+n) = O(n)$, and the locality of the kinetic Delaunay method is $O(r)$. The cost per event becomes $O(\log n)$, since the expected number of circles crossing the surrounding quad of the new edge, when an edge-flip happens, is constant. When the reference circles are large, say of the order of the diameter of the point set, then we expect the number of crossing edges per reference circle to be $O(\sqrt{n})$. In this case, the number of certificates in the event queue is $O(r\sqrt{n}+n)$, which is again better than the obvious and kinetic heap methods. The locality of the kinetic Delaunay method becomes $O(r)$ on the average, whereas the cost per event is still $O(\log n)$, since again we expect the number of reference circles crossing the quad of the new edge, when a Delaunay flip occurs, to be a constant. The situation becomes even more favorable for the kinetic Delaunay method when $r = \Theta(n)$.

4.5 The Relative Convex Hull

Relative convex hulls have been of interest in both the computer vision [50] and computational geometry community [26]. In this section we describe how to maintain the relative convex hull for a set of points S moving inside a simple polygon P.

Let R be the relative convex hull of S with respect to P. We will refer to the edges of P as p-edges and to the edges of R as r-edges. Note that a p-edge can be an r-edge, and also note that the graph G with vertices the set $P \cup S$ and edges the union of the set of p-edges and r-edges is a PSLG.

We want to construct and maintain the CDT of G and properly update both G and the triangulation whenever points need to be added or removed from R. There are two kinds of events that we need to handle other than the edge-flip events that we need to process in order to maintain the CDT.

The first kind happens when a point in $(S \cup P) \setminus R$ becomes a point of R. Let p be the point in $(S \cup P) \setminus R$ and let q and r be the endpoints of the r-edge that p hits (see Fig. 4.10(left to right)). It can easily be verified that when p becomes collinear with q and r, then the triangle pqr is a triangle of the CDT. What we have to do in this case is to remove qr, add qp and pr to the set of edges in G, and retriangulate the area around p. This can be done by triangulating the quadrangle created by the deletion

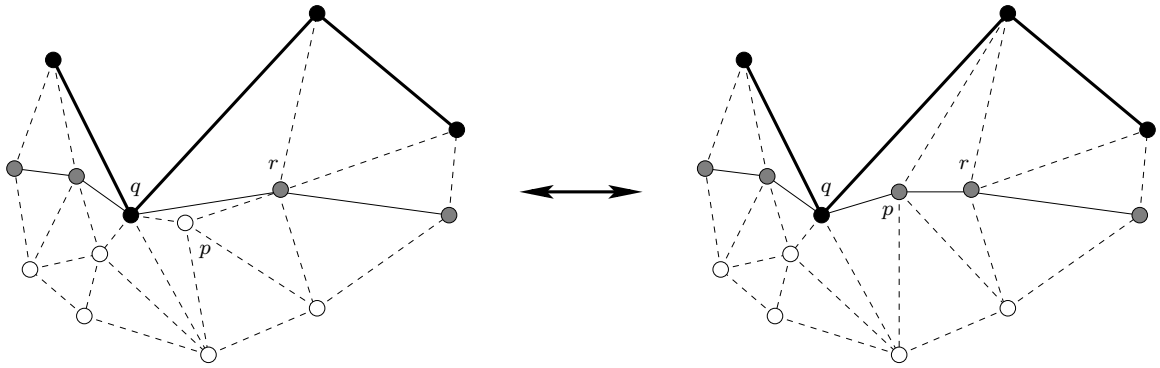


Figure 4.10: The Relative Convex Hull events. The thick solid line is the polygon P. The thin solid line is the relative convex hull R and the dashed edges are the non-constrained edges of the CDT. Left to right: p becomes a point of the RCH. Right to left: p is no longer a point of the RCH.

of qr, and then by simply invoking the standard edge-flip algorithm for producing the CDT given any triangulation, with the appropriate initial edge list [11].

The second kind of event is the symmetric one, when a point in R becomes a point of $(S \cup P) \setminus R$. Let p be the point in R and let q and r be the endpoints of the r-edges incident to p (see Fig. 4.10(right to left)). Such an event can be detected by scheduling `CCW` tests for all consecutive triplets in R. What we have to do in this case is to delete all the edges connecting p with points inside or outside R, depending on whether $p \in P$ or $p \in S$, respectively, add the edge qr in G, remove the r-edges qp and pr from G (but not from the CDT), triangulate the hole next to p and reconstruct the CDT using the edge-flip algorithm.

4.6 Conclusion

In this chapter we presented how to maintain the near neighbors of moving points in two and three dimensions. In particular, we described how to maintain all the points that are within a fixed (possibly time-varying) distance from a reference point and how to maintain the k-nearest neighbors of a reference point. We also showed how to maintain the Constrained Delaunay triangulation; using that as the underlying structure we discussed how to maintain near neighbors in two dimensions when obstacles are present. Finally, we presented an algorithm for maintaining the relative convex

hull for a set of points moving inside a simple polygon.

One very interesting problem is that of maintaining the set of points that are visible from a reference point. This problem seems much more complicated than that of maintaining near neighbors. The obvious solution is to try to maintain the visibility graph of the point set and the environment. However, the visibility graph can be of quadratic size, which is prohibitive. On the other hand, the CDT does not seem to contain enough information for efficiently maintaining the set of visible points. It would be really nice to have a structure for maintaining the visible set of a reference point, that lies somewhere in between these two extremes. All the algorithms involving obstacles that are presented in this chapter are tied to two dimensions. We would like to extend these structures to three or higher dimensions. Finally, the algorithm presented here for the maintenance of the relative convex hull does not seem to be efficient. It would be of interest to find other KDSs that can be used in a more efficient manner for the maintenance of the relative convex hull.

Chapter 5

Voronoi Diagrams for Moving Disks

In this chapter we tackle the problem of maintaining the Euclidean Voronoi diagram, or its dual the Delaunay Triangulation (DT) for a set of disks moving in the plane. The Euclidean Voronoi diagram of disks is defined in the same way as the Voronoi diagram for points: we assign every point on the plane to the disk to which it is closer. We measure the distance of a point from a disk as the Euclidean distance of the point from the center of the disk minus the radius of the disk. The set of points assigned to a disk is the Voronoi cell of that disk and the collection of the boundaries of the Voronoi cells is the Euclidean Voronoi diagram of the set of disks. It turns out that the Voronoi diagram of the disks is a subdivision of the plane. We can define its dual, called the Delaunay triangulation, by connecting the centers of the disks whose Voronoi cells are adjacent. We define the Voronoi diagram and its dual the Delaunay triangulation more precisely in Section 5.1.

The major contribution here is that the disks are allowed to intersect. This enables us to not only report collisions between disks, but also to report when the penetration depth between two disks achieves a certain value, or when a disk is wholly contained inside another disk. Moreover, our data structure can be used for maintaining the connectivity of the set of disks as the disks move, or to maintain near neighbors of disks.

The Voronoi diagram is maintained using the Kinetic Data Structure (KDS)

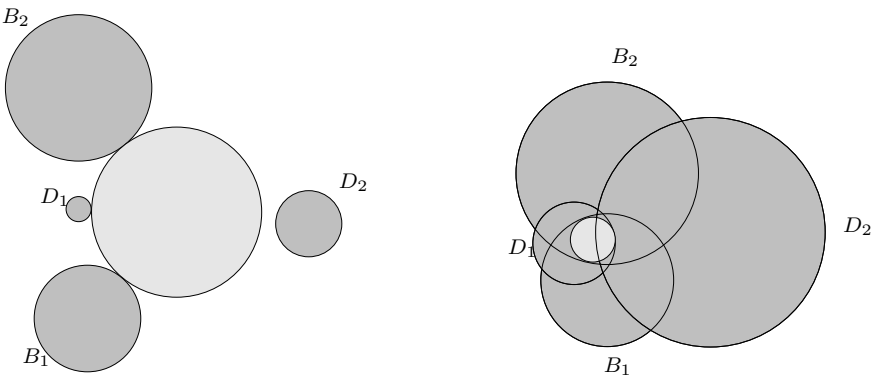


Figure 5.1: Top: the edge connecting B_1 and B_2 is locally Delaunay because the exterior tangent ball of B_1, B_2 and D_1 does not intersect D_2. Bottom: the edge connecting B_1 and B_2 is locally Delaunay because the interior tangent ball of B_1, B_2 and D_1 is not contained in D_2. The exterior/interior tangent ball of B_1, B_2 and D_1 is shown in light gray.

framework introduced in [8] and is discussed in more detail in Chapter 2. The kinetization process relies heavily on the fact that the local Delaunay property for the edges in the DT ensures that the triangulation is globally Delaunay. Let e be an edge connecting the disks B_1, B_2 and having the disks D_1, D_2 as its neighbors in the triangulation. The local Delaunay property states that the edge e is an edge of the DT if the exterior tangent ball of B_1, B_2 and D_1 does not intersect the disk D_2, or if the interior tangent ball of B_1, B_2 and D_1 is not contained in D_2 (see Fig. 5.1). The global Delaunay property states that there exists an edge in the DT between two disks B_1 and B_2 if there exists an exterior tangent ball to B_1 and B_2 that does not intersect any other disk or if there exists an interior tangent ball to B_1 and B_2 that is not contained in any other disk. In this chapter we prove the relationship between the global and local Delaunay properties.

Using the local Delaunay property, we can maintain the Voronoi diagram for the set of disks using three types of events, two of which appear only in the case of intersecting disks. The data structure that we use is called the *Augmented Delaunay Triangulation* (ADT) of the set of disks. It consists of the Delaunay triangulation of the disks augmented with some additional linear size data structure associated with the disks that do not contribute to the Voronoi diagram (see Fig. 5.2). We call these disks *trivial*. Since the edges in ADT $\setminus$ DT are associated with trivial disks, the ADT

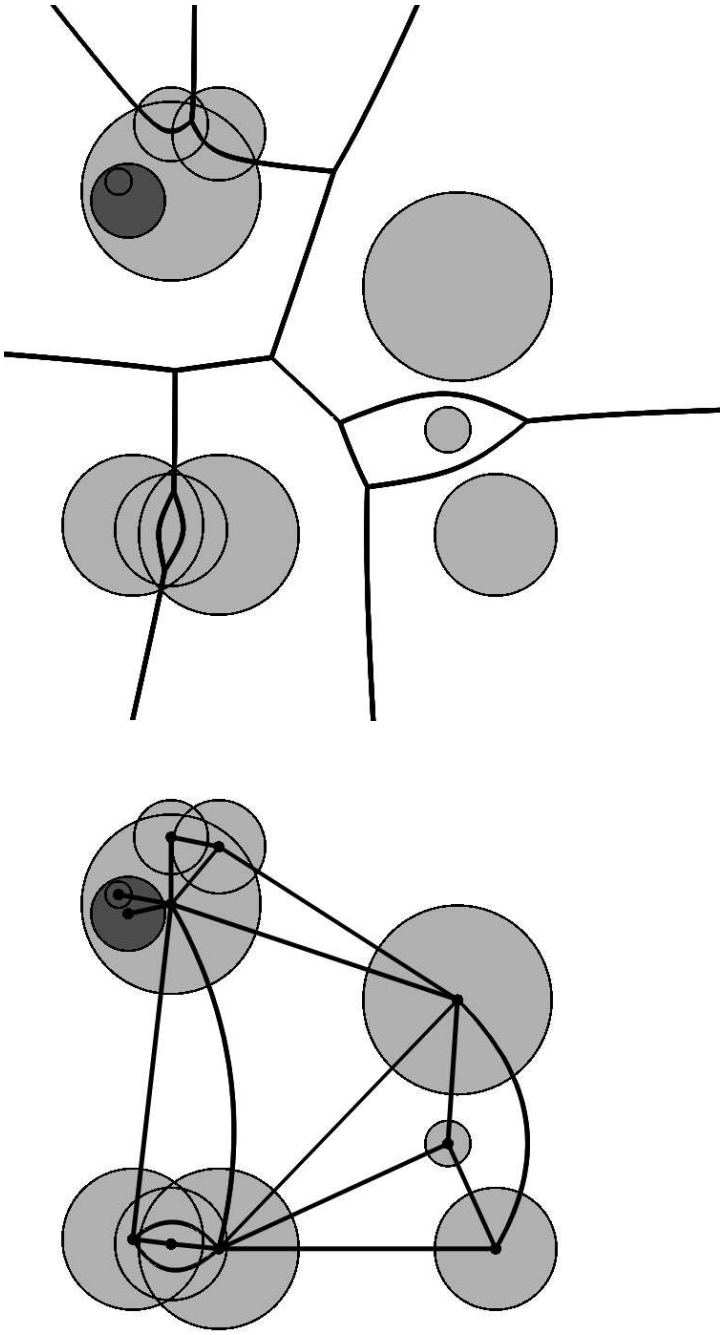


Figure 5.2: The Voronoi diagram for a set of disks (top) and the corresponding Augmented Delaunay Triangulation (bottom). The disks in dark gray are trivial.

differs from the DT only when we have trivial disks. In particular, if we do not allow disk intersections the DT and ADT coincide.

An interesting property of the ADT is that the closest pair of the set of disks is realized between two disks that share an edge in the ADT. Thus, knowing how to maintain the ADT enables us to maintain the closest pair of the set of disks using a tournament tree on the edges of the ADT. The distance between two disks B_1 and B_2 is defined as follows :

$$\delta(B_1, B_2) = \begin{cases} d(b_1, b_2) - r_1 - r_2, & B_1 \not\subseteq B_2 \text{ and } B_2 \not\subseteq B_1 \\ -2\min\{r_1, r_2\}, & B_1 \subseteq B_2 \text{ or } B_2 \subseteq B_1 \end{cases}, \tag{5.1}$$

where b_i are the centers of the disks B_i, r_i their radii, and $d(\cdot, \cdot)$ denotes the Euclidean metric. If the set of disks does not have any intersecting disks the distance function (5.1) gives us the closest pair in the usual sense. If there are intersecting disks, then the closet pair with respect to (5.1) is either the pair of non-trivial disks with *maximum penetration depth* among all intersecting pairs of disks, or the largest trivial disk and its container.

Another important property of the ADT is that a subgraph of the ADT is a spanning subgraph of the connectivity graph of the set of disks. Knowing how to maintain the ADT enables us to maintain the connectivity of the set of disks by maintaining the afore-mentioned spanning subgraph.

Finally, the DT of a set of non-intersecting disks has the property that if we want to find the near neighbors of a disk we only need to look at its neighborhood in the DT. Therefore, in order to maintain the near neighbors of a disk we simply need to look at its neighborhood in the DT and update this neighborhood as the DT changes. We make this statement more precise in Section 5.7.

The rest of the chapter is structured as follows. In Section 5.1 we introduce the Voronoi diagram for disks, and discuss some of its properties. Section 5.2 is devoted to proving the relationship between the global and local Delaunay properties. In Section 5.3 we show how to kinetize the Voronoi diagram. In Section 5.5 we prove that the closest pair of the set of disks is realized between disks that share an edge in the ADT and describe how to maintain the closest pair. In Section 5.6 we prove that a properly chosen subset of the ADT is a spanning subgraph of the connectivity

graph of the set of disks. Finally, in Section 5.7 we discuss the maintenance of near neighbors of disks.

5.1 The Voronoi Diagram for Disks and its Properties

Let S be a set of n disks B_j, with centers b_j and radii r_j. Let $d(\cdot,\cdot)$ be the Euclidean distance. We define the distance $\delta(p,B)$ between a point $p \in \mathbb{E}^2$ and a disk $B = \{b, r\}$, as $\delta(p,B) = d(p,b) - r$. We define the Voronoi diagram for the set S as follows. For each $j \neq i$, let $H_{ij} = \{y \in \mathbb{E}^2 : \delta(y,B_i) \leq \delta(y,B_j)\}$. Then we define the (closed) Voronoi cell of B_i to be the cell $V_i = \bigcap_{i \neq j} H_{ij}$. The *Voronoi diagram* VD(S) of S is defined to be the set of points which belong to more that one Voronoi cell. For simplicity we assume that no point in VD(S) belongs to more than three Voronoi cells; this is a generalization of the assumption made for point sets, in which no more than three points are cocircular. The Voronoi diagram just defined is a subdivision of the plane. It consists of straight or hyperbolic arcs and each Voronoi cell is star-shaped with respect to the center of the corresponding disk (see [48, Properties 3 and 4]).

In contrast to the Voronoi diagram for points, there may be disks whose corresponding Voronoi cell is empty. In particular, the Voronoi cell V_i of a disk B_i is empty if and only if B_i is wholly contained in another disk (see [48, Property 2]). A disk whose Voronoi cell has empty interior is called *trivial*, whereas a disk whose Voronoi cell has non-empty interior is called *non-trivial*.

We define the dual of VD(S) as follows. The vertices are the centers of the non-trivial disks. If $V_i \cap V_j \neq \emptyset$, we add an edge $[b_i, b_j]$ for every open arc α of $V_i \cap V_j$. It turns out that the dual graph is a planar graph [48, Property 6]. We can define a planar embedding of the dual graph by mapping every edge e_α to two straight line segments $b_i x$ and $x b_j$ where x is an interior point of α. This planar embedding has the property that all but its outer face contain at least three edges. This implies that the size of both the Voronoi diagram and its dual is $O(n)$ [48, Property 7]. The Voronoi diagram just defined may consist of more than one connected component (see [48, Property 9]). If it consists of a single connected component and the disks

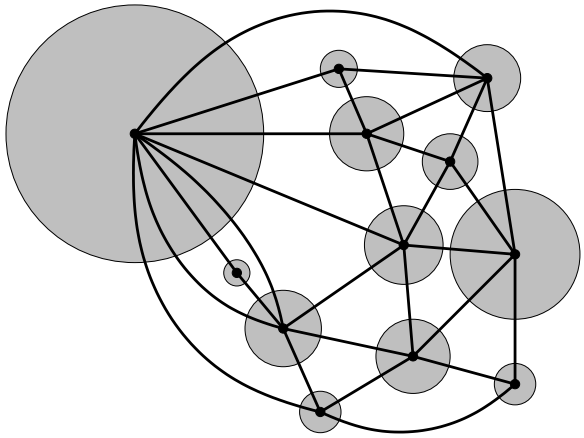


Figure 5.3: A case in which the Voronoi diagram consists of a single connected component. The dual is a (generalized) triangulation of the plane.

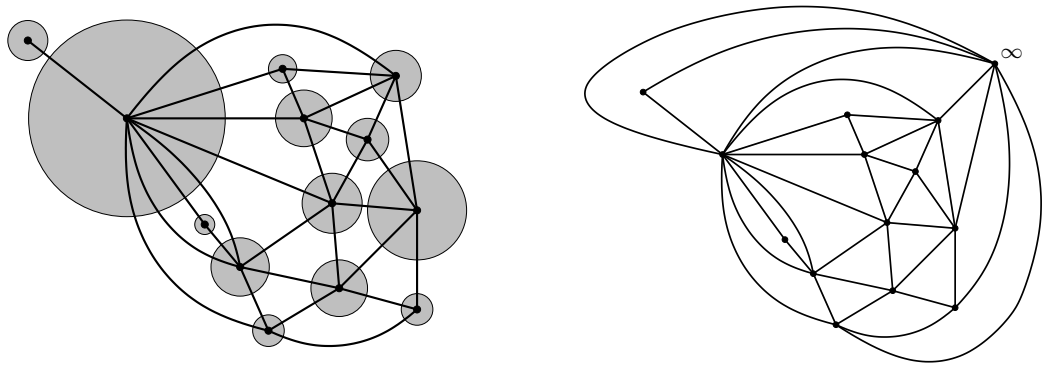


Figure 5.4: A case in which the Voronoi diagram consists of two connected components. Left: the dual of the Voronoi diagram is not a (generalized) triangulation of the plane. Right: the graph corresponding to the compactified version of the dual of the Voronoi diagram; in this case we have a (generalized) triangulation.

are in general position, the dual graph is a generalized triangulation of the plane. By generalized we mean that the edges of the triangles may be curved arcs or polygonal lines instead of straight line segments and that all but its outer face contain exactly three edges (see Fig. 5.3). If it consists of more than one connected component then we can add a disk at infinity and appropriately add edges from the disks on the convex hull of S to the disk at infinity. The compactified graph is a generalized triangulation, in which every face consists of exactly three edges (see Fig. 5.4).

We assume throughout the rest of the chapter that the Voronoi diagram consists of only one connected component. We shall refer to the dual graph of $\mathrm{VD}(S)$ as the *Delaunay Graph* $\mathrm{DG}(S)$ of S. If the disks are in general position we refer to the dual graph as the *Delaunay Triangulation* $\mathrm{DT}(S)$ of S.

Let B_i and B_j be two disks such that no disk is contained inside the other. A ball tangent to B_i and B_j that does not contain either of the two is an *exterior tangent ball*. A ball tangent to B_i and B_j that lies in $B_i \cap B_j$ is an *interior tangent ball*. The following theorem couples the existence of edges in $\mathrm{DG}(S)$ with exterior and interior tangent balls of disks in S.

Theorem 16 (Global Property). *There exists an edge $[b_i, b_j]$ in $DG(S)$ between B_i and B_j if and only if one of the following holds: (1) there exists an exterior tangent ball to B_i and B_j which does not intersect any disk $B_k \in S$, $k \neq i, j$; (2) there exists an interior tangent ball to B_i and B_j, which is not contained in any disk $B_k \in S$, $k \neq i, j$.*

Proof. Let $[b_i, b_j]$ be an edge of $\mathrm{DG}(S)$. Then the intersection of V_i and V_j consists of at least one arc α with non-empty interior. Let y be a point in the interior of α. Consider the ball C centered at y with radius $|\delta(y, B_i)| = |\delta(y, B_j)|$. Then C is tangent to both B_i and B_j. If $y \in B_i \cap B_j$ suppose that C is contained in a third disk B_k. If $y \notin B_i \cap B_j$ suppose that C intersects with a third disk B_k. In both cases, if C is not tangent to B_k, then $\delta(y, B_k) < \delta(y, B_i) = \delta(y, B_j)$, which contradicts the assumption that $y \in V_i \cap V_j$. If C is tangent to B_k, then y belongs to V_k as well. But this contradicts the fact that y is an interior point of α. Hence C is the desired ball. Conversely, assume there exists an exterior tangent ball C to both B_i and B_j, that does not intersect with any other disk B_k. Let y be the center of C. Then $y \in V_i \cap V_j$, since $\delta(y, B_i) = \delta(y, B_j)$ and $\delta(y, B_i) < \delta(y, B_k)$, for all $k \neq i, j$. Suppose that y is an end point of an arc in $\mathrm{VD}(S)$. Then there exists a third disk B_k that is tangent to C; this contradicts the assumption that C intersects with only B_i and B_j. Hence y has to be in the interior of some arc α of $\mathrm{VD}(S)$. But then the edge $[b_i, b_j]$ is an edge of $\mathrm{DG}(S)$. Assume now that C is an interior tangent ball to both B_i and B_j, which is not contained inside any disk B_k, $k \neq i, j$. Let y be the center of C. Then $\delta(y, B_i) = \delta(y, B_j)$ and since C is not contained inside another disk B_k, $k \neq i, j$, we have that $\delta(y, B_k) > \delta(y, B_i)$. Hence $y \in V_i \cap V_j$. If y was an end point of $V_i \cap V_j$,

then there would be a third disk B_k, $k \neq i, j$, with $\delta(y, B_k) = \delta(y, B_i)$. But then C would be contained in B_k, which contradicts our assumption. Hence y is an interior point of $V_i \cap V_j$, which implies that $[b_i, b_j]$ is an edge in $\mathrm{DG}(S)$. □

In order to account for the trivial disks, we augment the Delaunay triangulation with some additional edges. For a trivial disk D we add an edge between D and its container disk. If D has more than one container we need to add an edge to only one of its containers, chosen arbitrarily. We call this structure the *Augmented Delaunay Triangulation* $\mathrm{ADT}(S)$ of S. The set of additional edges forms a forest, and the root of each tree in the forest is a non-trivial disk. Clearly, the forest has linear size. Hence the size of $\mathrm{ADT}(S)$ is still $O(n)$.

5.2 The Local Property of the Delaunay Triangulation

In this section we present the local Delaunay property for a set of possibly intersecting disks and we show that the local Delaunay property is a sufficient and necessary condition for a (generalized) triangulation of the set of disks to be globally Delaunay. We only consider non-trivial disks, since trivial disks do not contribute to the Voronoi diagram. The relationship between the local and global Delaunay property of the triangulation is central in the kinetization process, since this is the property that enables us to describe a global property though a set of local conditions.

Let π_{ij} be the bisector of B_i and B_j. The bisectors are lines or hyperbolas which can be oriented. We define the orientation to be such that b_i is to the left of π_{ij}. Let $\prec$ be the linear ordering on the points of π_{ij}. Let o_{ij} be the midpoint of the subsegment of $b_i b_j$ that lies either in free space or in $B_i \cap B_j$. We can parameterize π_{ij} as follows: if $p \prec o_{ij}$ then $\zeta_{ij}(p) = -(\delta(p, B_i) - \delta(o_{ij}, B_i))$; otherwise $\zeta_{ij}(p) = \delta(p, B_i) - \delta(o_{ij}, B_i)$. When we have no degeneracies, the function ζ_{ij} is a 1–1 and onto mapping from π_{ij} to $\mathbb{R}$. Let $\omega(p, B_i)$ be the ball with center p and radius $|\delta(p, B_i)|$. Clearly, $\omega(p, B_i)$ is tangent to B_i and B_j, and $\omega(p, B_i)$ either does not contain any of B_i and B_j or lies in $B_i \cap B_j$.

Let B_i, B_j and B_k be three disks such that no disk is contained inside another.

The three disks may have up to eight common tangent balls (see Fig. 5.5). Among those we are interested in only two kinds: those balls that do not contain any of the three disks, which we call *exterior tangent balls* and those that are contained entirely in the intersection of the three disks, which we call *interior tangent balls* (see Fig. 5.6). Let P_i, P_j, P_k be the points of tangency of the disks B_i, B_j, B_k with their common tangent ball. If $\texttt{CCW}(P_i, P_j, P_k) > 0$ we call the common tangent ball the

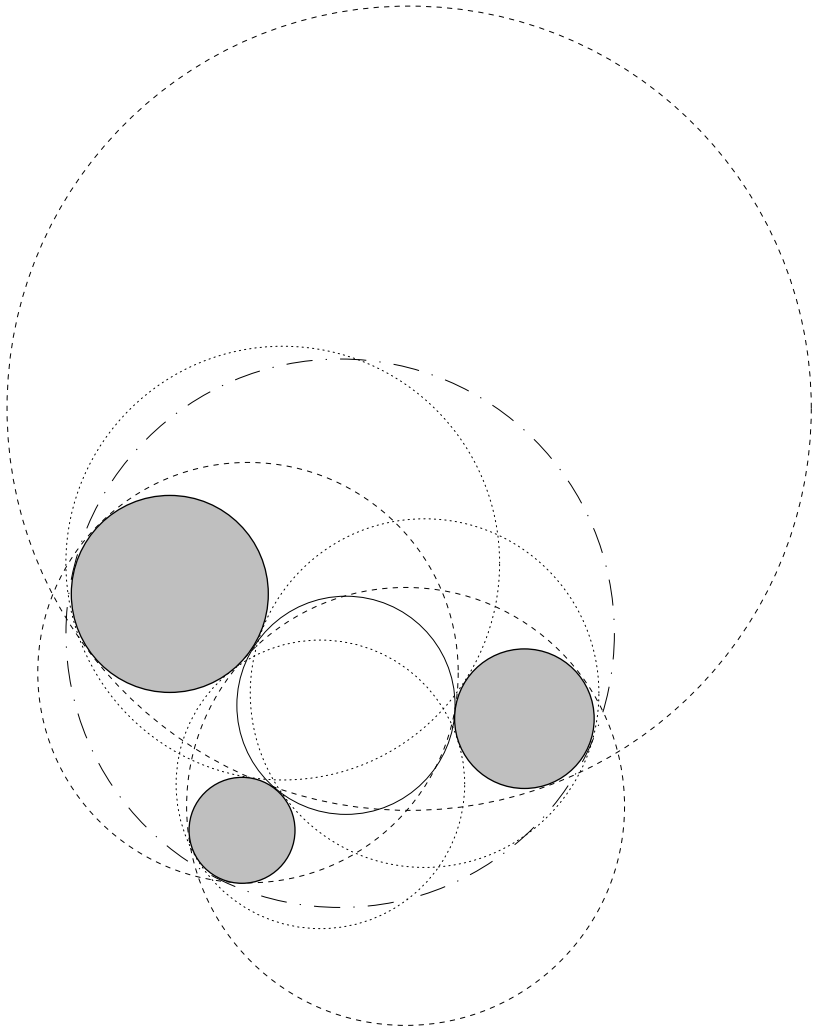


Figure 5.5: Three disks and their eight common tangent balls. The solid tangent ball does not contain any of the disks. The three dotted tangent balls contain exactly one disk. The three dashed tangent balls contain eactly two disks. The dash-dotted tangent ball contains all three disks.

left tangent ball of the triple B_i, B_j, B_k. If $\texttt{CCW}(P_i, P_j, P_k) < 0$, we call the common tangent ball the *right tangent ball* of the triple B_i, B_j, B_k. Note that three disks have at most one left/right exterior/interior tangent ball (see Fig. 5.6). Finally, we define $\zeta_{ij}^L(B_k)$ to be the parameter value of the center $c \in \pi_{ij}$ of the left tangent ball of B_i, B_j and B_k. Correspondingly, $\zeta_{ij}^R(B_k)$ is the parameter value of the center of the right tangent ball of B_i, B_j and B_k.

Let $\mathcal{T}(S)$ be a (generalized) triangulation of S that is constructed as follows. The vertices of $\mathcal{T}(S)$ are the centers of the disks in S. An oriented edge e_{ij}^{kl} in $\mathcal{T}(S)$ is an edge that connects the disks B_i and B_j and has, as neighbors in $\mathcal{T}(S)$, the disks B_k and B_l to its left and right, respectively. It is possible that the disks B_k and B_l are the same. The disk B_k is called the *left neighbor* of e_{ij}^{kl} and the disk B_l is called the *right neighbor* of e_{ij}^{kl}. Note that the quadruple (i, j, k, l) uniquely defines edges in $\mathcal{T}(S)$, i.e., there can be at most one oriented edge in the triangulation starting from B_i, ending at B_j and having B_k and B_l to its left and right, respectively. The left tangent ball of the triple B_i, B_j, B_k is called the *left (tangent) ball* of e_{ij}^{kl}, and similarly, the right tangent ball of the triple B_i, B_j, B_l is called the *right (tangent) ball* of e_{ij}^{kl}. We assume that for every edge in $\mathcal{T}(S)$ its left and right tangent balls exist. Then we can embed e_{ij}^{kl} with a two-leg polygonal line $b_i x b_j$, where x is a point on π_{ij} with parameter value $\zeta_{ij}(x)$ in between $\zeta_{ij}^L(B_k)$ and $\zeta_{ij}^R(B_l)$. For every triangle $\Delta_{ijk} \in \mathcal{T}(S)$ that connects the disks B_i, B_j and B_k, in counterclockwise order, we associate the left tangent ball $\tilde{\Delta}_{ijk}$ of the triple B_i, B_j, B_k. This is called the *Delaunay ball* of Δ_{ijk}. Note that there is an 1–1 correspondance between triangles Δ in $\mathcal{T}(S)$ and their Delaunay balls $\tilde{\Delta}$.

An edge e_{ij}^{kl} in $\mathcal{T}(S)$ is called *locally Delaunay* if the predicate $\texttt{InCircle}(B_i, B_j, B_k, B_l)$ is false. A triangle Δ in $\mathcal{T}(S)$ is called locally Delaunay if all its edges are locally Delaunay. The `InCircle` predicate is defined below. We are going to discuss the algebraic conditions associated with the `InCircle` predicate in Section 5.8.

Definition 10. *Let B_i, B_j, B_k, B_l be four disks. The predicate $\texttt{InCircle}(B_i, B_j, B_k, B_l)$ is true if $k \neq l$ and either B_l intersects the exterior left tangent ball of B_i, B_j and B_k, or B_l contains the interior left tangent ball of B_i, B_j and B_k.*

Note that if an edge e_{ij}^{kl} is locally Delaunay then $\zeta_{ij}^L(B_k) > \zeta_{ij}^R(B_l)$. This imples that if a triangle Δ is locally Delaunay, then the center $c_{\tilde{\Delta}}$ of its Delaunay ball $\tilde{\Delta}$ lies

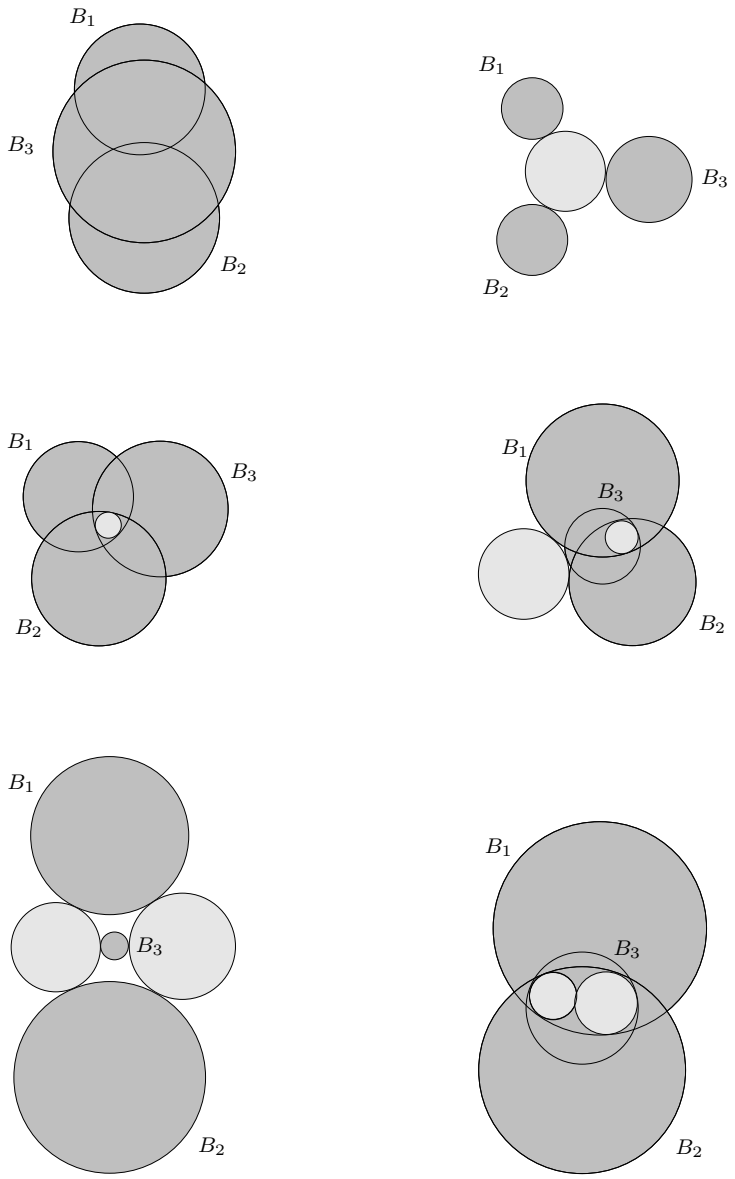


Figure 5.6: The various cases that can possibly happen with respect to the number of exterior or interior tangent balls of three disks B_1, B_2 and B_3. The tangent balls are shown in light gray. From left to right and top to bottom: no exterior or interior tangent balls; only one exterior tangent ball; only one interior tangent ball; one exterior and one interior tangent ball; two exterior tangent balls; two interior tangent balls.

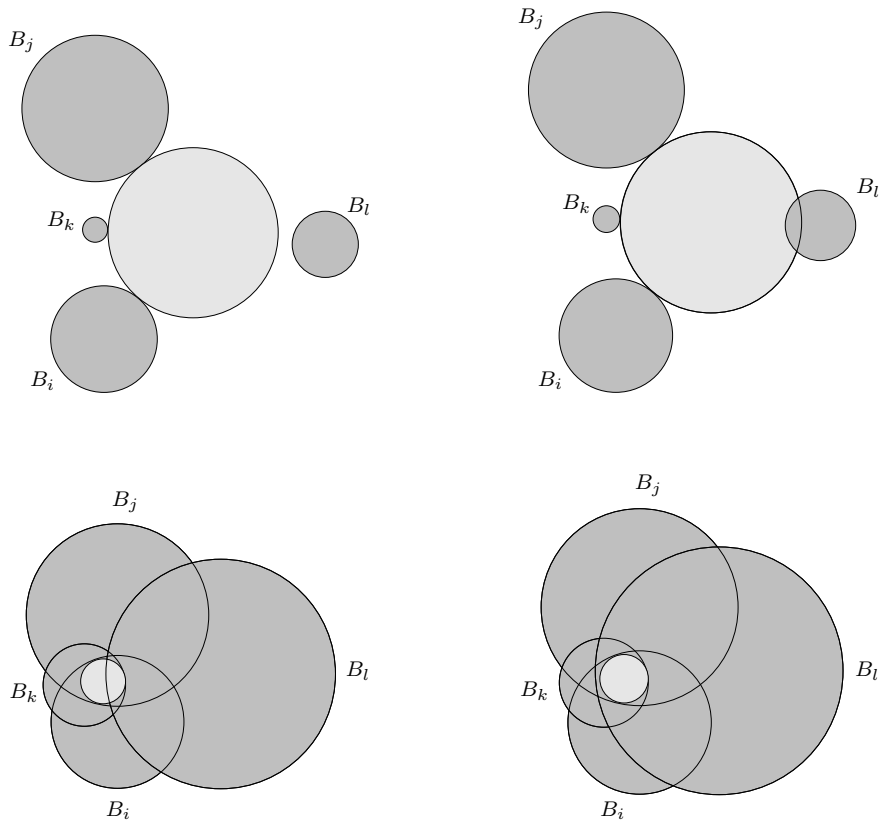


Figure 5.7: The four cases (assuming $k \neq l$) for the predicate `InCircle`(B_i, B_j, B_k, B_l). Left: the predicate is false. Right: the predicate is true.

in the interior of Δ. We are now ready to prove the main result of this section, namely the relationship between the local and global property of the Delaunay triangulation.

Theorem 17 (Local Property). *A (generalized) triangulation $\mathcal{T}(S)$ is the Delaunay triangulation of S if and only if all the triangles in $\mathcal{T}(S)$ are locally Delaunay.*

Proof. It is straightforward to verify that if a triangulation is globally Delaunay then it is locally Delaunay as well.

Suppose now that we have a triangulation $\mathcal{T}(S)$ that is locally Delaunay but not globally. We assume without loss of generality that for all triangles the corresponding Delaunay balls are interior. If this is not the case we can increase the radii of all the disks by a sufficiently large quantity. The triangulation $\mathcal{T}(S)$ is not affected by this change, other than that all the Delaunay balls become interior.

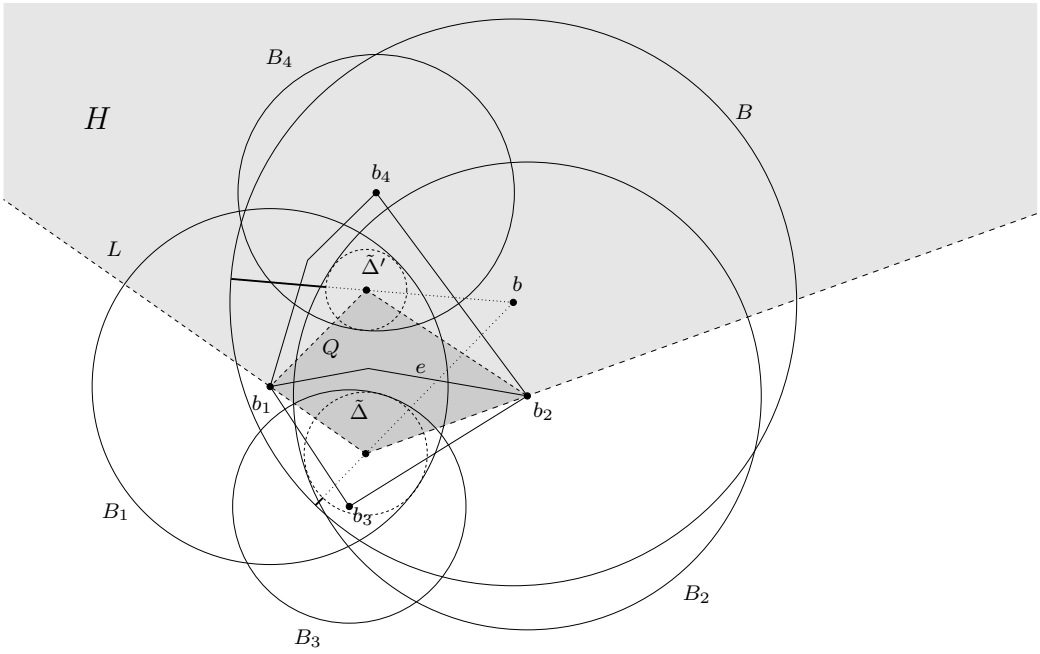


Figure 5.8: Proof of the local property.

Since $\mathcal{T}(S)$ is not globally Delaunay there exists a triangle Δ that is locally Delaunay but its Delaunay ball $\tilde{\Delta}$ is contained inside some disk $B = \{b, r\}$. Consider the distance between the disk B and the Delaunay ball $\tilde{\Delta}$. This distance is

$$\delta(\tilde{\Delta}, B) = d(b, c_{\tilde{\Delta}}) + r_{\tilde{\Delta}} - r\,,$$

where $c_{\tilde{\Delta}}$ and $-r_{\tilde{\Delta}}$ are the center and radius of $\tilde{\Delta}$ (interior Delaunay balls are considered to have negative radius). Among all triangles Δ for which $\tilde{\Delta} \subset B$, choose Δ to be the one for which $\delta(\tilde{\Delta}, B)$ is minimized.

Let $e = [b_1, b_2]$ be the (oriented) edge of Δ that the segment $c_{\tilde{\Delta}}b$ intersects (see Fig. 5.8). Let L be the two-leg polygonal line $b_1 c_{\tilde{\Delta}} b_2$. Let Δ' be the left neighboring triangle of e in $\mathcal{T}(S)$. Since $c_{\tilde{\Delta}}b$ intersects e, b must lie in the half-plane H bounded by L that contains e. Since both Δ and Δ' are locally Delaunay the quad $Q = b_1 c_{\tilde{\Delta}} b_2 c_{\tilde{\Delta}'}$ is contained inside $\Delta \cup \Delta'$, and clearly b cannot lie inside Q. But then we have $\tilde{\Delta}' \subset B$, and moreover $\delta(\tilde{\Delta}', B) < \delta(\tilde{\Delta}, B)$, which contradicts the fact that $\delta(\tilde{\Delta}, B)$ is minimal. □

5.3 Kinetizing the Delaunay Triangulation

Maintaining the Voronoi diagram or its dual, the Delaunay triangulation, for a set of points moving on the plane is straightforward [22]. This is due to the local property of the Delaunay triangulation, which states that if the triangles of the Delaunay triangulation are locally Delaunay, then the triangulation is the Delaunay triangulation. When one of the conditions fails we simply have to do an *edge-flip* operation to restore the correctness of the Delaunay triangulation. The same principle is exploited to maintain the power diagram of non-intersecting moving disks [24] and the Voronoi diagram for rigidly moving polygons [23].

In the case of non-intersecting disks the very same ideas can be used. The local Delaunay property is also true for the Delaunay triangulation of disks, as we showed in the preceding section, and thus the critical events happen at times when four disks become cocircular or when three disks lying on the convex hull of S have a common tangent line. In fact if we add a disk at infinity and connect every disk lying on the convex hull of S with the disk at infinity, the compactified version of the Delaunay triangulation of S consists of triangles only and every triangle has exactly three neighboring triangles. In this setting, the case of three disks having a common tangent reduces to a cocircularity event with one of the disks being a disk at infinity. When such a cocircularity event happens we only need to perform an edge-flip operation to restore the correctness of the Delaunay triangulation, and its dual the Voronoi diagram.

However, when we allow disk intersections the situation changes considerably. Unlike the points' case, there are disks that are not associated with a particular Voronoi cell, namely the trivial disks. We need to account for these disks, since as the disks move some of the trivial disks may become non-trivial and vice versa. This is done by considering the Augmented Delaunay triangulation instead of the Delaunay triangulation. There are three types of events that change the combinatorial structure of $\mathrm{ADT}(S)$: the *cocircularity*, the *appearance* and the *disappearance* event. All three events are associated with edges of the $\mathrm{ADT}(S)$. In particular, an edge in $\mathrm{DT}(S)$ is associated with a cocircularity and a disappearance event. An edge in $\mathrm{ADT}(S) \setminus \mathrm{DT}(S)$ is associated with an appearance event. We now discuss each type of event separately.

5.3.1 The Cocircularity Event

The cocircularity event happens when four distinct disks have a common exterior or interior tangent ball. Let B_i, $i = 1, 2, 3, 4$ be the four disks associated with the cocircularity event and let $[b_1, b_3]$ be the edge to be flipped. We need to delete that edge and add the edge $[b_2, b_4]$ (see Fig. 5.9).

The cost of this event is $O(\log n)$. We need $O(1)$ time to perform the edge-flip, $O(\log n)$ time to schedule the cocircularity event for the new edge, $O(\log n)$ time to schedule the disappearance event for the new edge, and $O(\log n)$ time to reschedule the four cocircularity events of the edges surrounding the new edge.

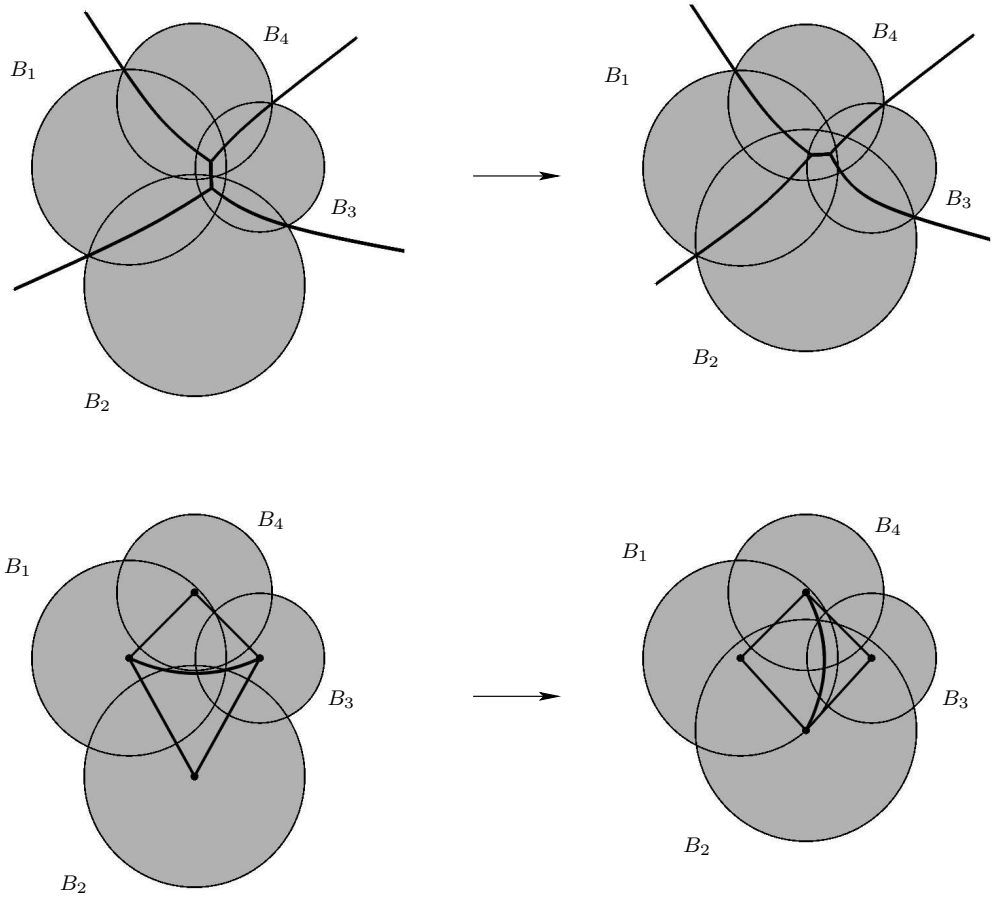
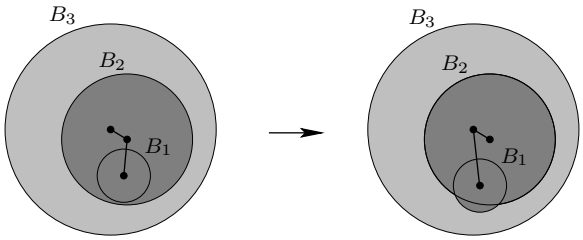


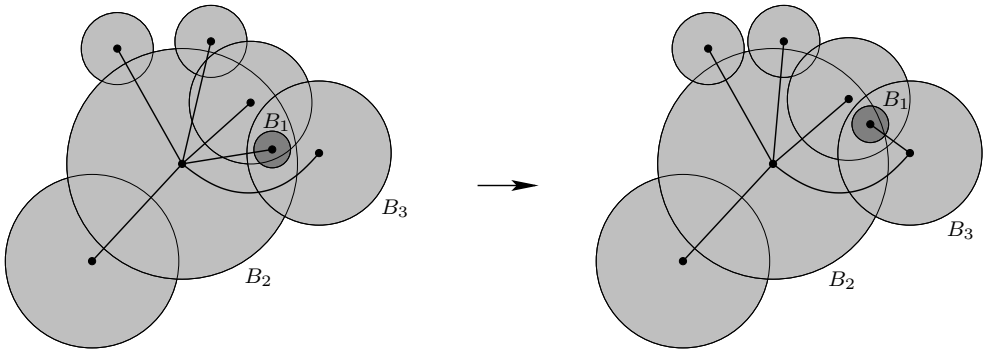
Figure 5.9: The cocircularity event. Top: the Voronoi diagram. Bottom: the Delaunay triangulation.

5.3.2 The Appearance Event

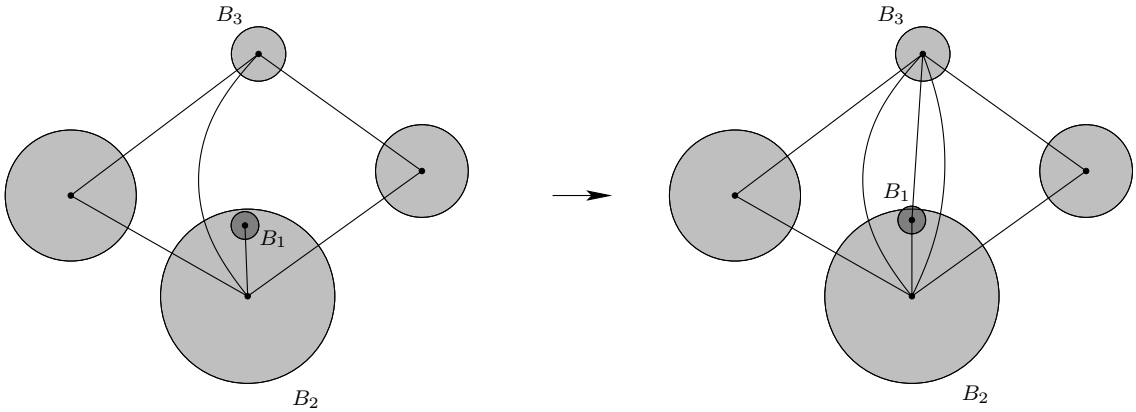
The appearance event occurs when a disk B_1 contained inside a disk B_2 is no longer wholly contained inside B_2. There are two possibilities when this happens: (1) B_2



(a) B_2 is a trivial disk.



(b) B_2 is non-trivial and B_1 remains trivial.



(c) B_2 is not-trivial and B_1 becomes non-trivial.

Figure 5.10: The appearance event.

is a trivial disk and (2) B_2 is a non-trivial disk. If B_2 is a trivial disk we delete the edge $[b_1, b_2]$ and add the edge $[b_1, b_3]$, where B_3 is the container disk of B_2 (see Fig. 5.10(a)). If B_2 is a non-trivial disk we need to first check its neighbors in the DT to see if B_1 is contained in any one of them. If such a neighbor B_3 exists we delete the edge $[b_1, b_2]$ and add the edge $[b_1, b_3]$ (see Fig. 5.10(b)). If B_1 is not contained in any of the neighbors of B_2, we need to identify the edge $[b_2, b_3]$ that corresponds to the edge of the Voronoi cell of B_2 that the half-line b_2b_1 intersects. Then duplicate this edge and add the edge $[b_1, b_3]$, thus creating two new triangles in $\mathrm{DT}(S)$ (see Fig. 5.10(c)).

The cost of the appearance event in the first case is $O(\log n)$, since we need $O(1)$ time to update the ADT and $O(\log n)$ time to schedule the new appearance event. In the second case, the cost of the appearance event is $O(n)$, since we have to look at all the neighbors of B_2 to find if B_1 is contained in any one of them. Hence the overall cost of this event is $O(n)$.

There is one issue, however, that we did not discuss in detail. In particular, it was suggested above implicitly, that when B_1 exits B_2 and is contained inside another non-trivial disk B_3, then this disk is a neighbor of B_2. The answer to this question is formally answered by the following theorem.

Theorem 18. *Let B_1 be a trivial disk in S that at time t_0 exits its non-trivial container disk B_2. If B_1 remains trivial at time t_0, then there exists a non-trivial disk $B_3 \neq B_2$ that contains B_1, and B_3 is a neighbor of B_2 in $DT(S)$.*

Proof. We will prove the theorem by contradiction. Suppose than at time t_0 none of the non-trivial disks than contain B_1 is a neighbor of B_2 in $\mathrm{DT}(S)$. Let B such a non-trivial disk, that contains B_1 at time t_0 and does not share an edge with B_2 in $\mathrm{DT}(S)$. Consider the ball $C(B_2, B)$ than is an interior tangent ball to both B_2, B and has its center on the line b_2b_1 (see Fig. 5.11). Let $r(B_2, B)$ be the (absolute value of the) radius of $C(B_2, B)$. Among all non-trivial disks B that contain B_1, choose B to be the one for which $r(B_2, B)$ is maximal. Since B_2 and B do not share an edge in the $\mathrm{DT}(S)$, there must be a non-trivial disk B' that contains the ball $C(B_2, B)$. But then $B_1 \subset B'$, and moreover $r(B_2, B') > r(B_2, B)$, which contradicts the maximality of $r(B_2, B)$. □

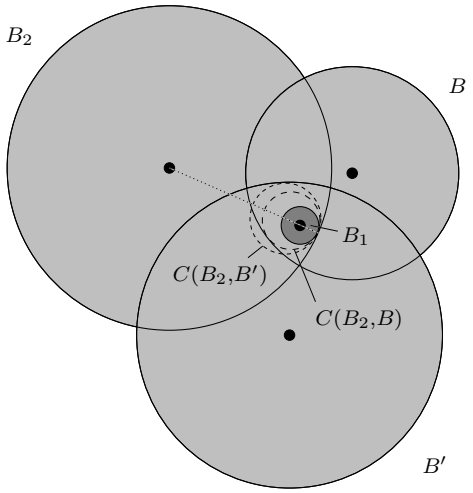


Figure 5.11: Proof of Theorem 18.

5.3.3 The Disappearance Event

The disappearance event takes place between two disks B_1 and B_2 when, e.g., B_1 becomes wholly contained in B_2. When the disappearance event occurs, the edge $[b_1, b_2]$ belongs to two triangles with a common third point b_3 corresponding to a disk B_3. We simply need to delete the edge $[b_1, b_3]$, and identify the two edges $[b_2, b_3]$, thus deleting two triangles from $\mathrm{DT}(S)$ (see Fig. 5.12).

The cost to process the disappearance event is $O(\log n)$. It takes $O(1)$ time to delete the two triangles from the triangulation, $O(\log n)$ time to schedule the appearance event for the edge $[b_1, b_2]$ and $O(\log n)$ time to reschedule the cocircularity events for the edge $[b_2, b_3]$ and its four surrounding edges.

5.4 Combinatorial Changes of the Voronoi Diagram

In this section we study the number of combinatorial changes of the Voronoi diagram for disks. For the purposes of our analysis, we assume that the disks move in pseudo-algebraic motions with constant degree. Since the conditions corresponding to certificate failure times are algebraic functions of the disk motions, each certificate involving the same set of disks can fail only a constant number of times during the

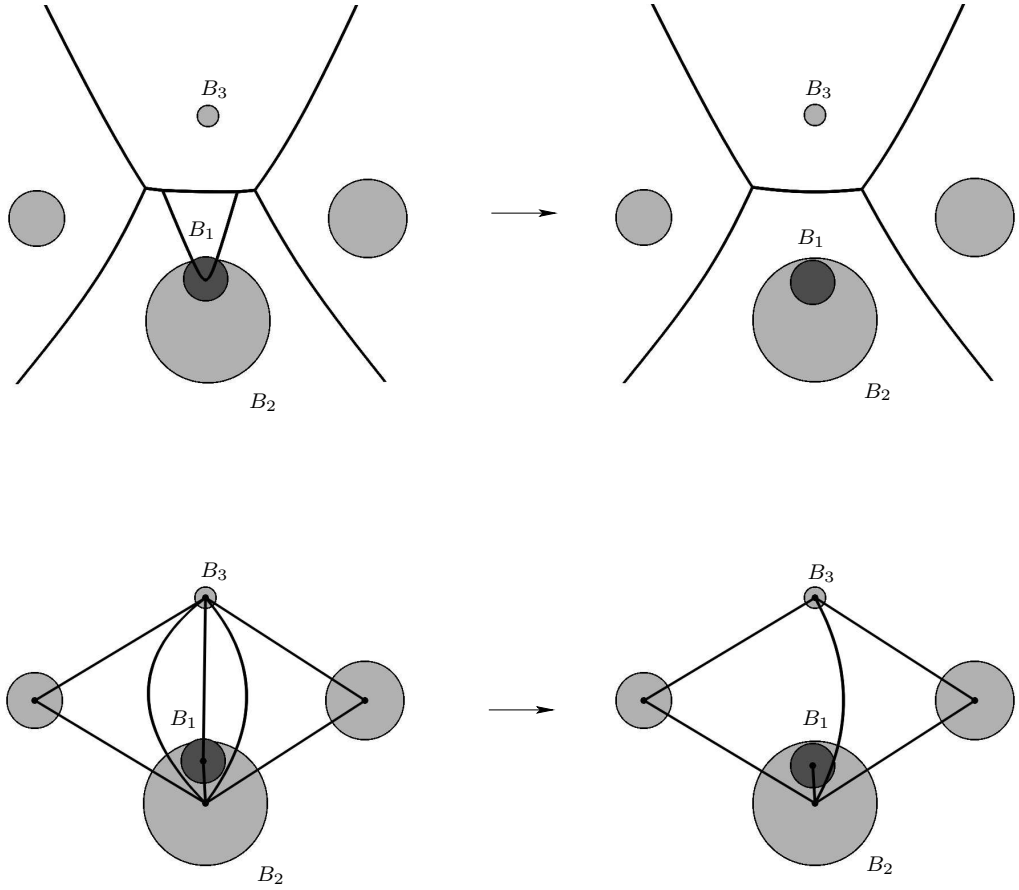


Figure 5.12: The disappearance event. Top: the Voronoi diagram. Bottom: the Delaunay triangulation.

entire motion process. The analysis that follows is very similar to that in [23] for the case of moving polygons. Our analysis, however, applies to the case of intersecting disks, whereas the analysis in [23] is done for non-intersecting polygons only.

Let $p \in \pi_{ij}$. If $p \notin B_i \cap B_j$, we say that p is *shaded* by B_k if $\omega(p, B_i) \cap B_k \neq \emptyset$. If $p \in B_i \cap B_j$, we say that p is *shaded* by B_k if $\omega(p, B_i) \subseteq B_k$. Let $S_{ij,k}$ be the set of points on π_{ij} shaded by B_k. Let $\tilde{S}_{ij,k}$ be the set of parameter values represented by the shaded portion, i.e., $\tilde{S}_{ij,k} = \{\zeta_{ij}(p) \mid p \in S_{ij,k}\}$. Since π_{ij} and π_{ik} intersect at most twice (see [48, Property 5]), $\tilde{S}_{ij,k}$ must have the form $\emptyset$, $(-\infty, a]$, $[b, \infty)$, $(-\infty, a] \cup [b, \infty)$, $[a, b]$ or $(-\infty, \infty)$, where a and b correspond to the parameter values of the Voronoi vertices defined by B_i, B_j and B_k. When $\tilde{S}_{ij,k}$ has the form

$(-\infty, a]$ or $(-\infty, a] \cup [b, \infty)$, we say that π_{ij} is *half-shaded* by B_k at a. Notice that if neither π_{ij} nor π_{ji} is half-shaded by B_k, then $\tilde{S}_{ij,k}$ must have the form $\emptyset$ or $[a, b]$, where $a < b$. Assume that B_k is not contained inside B_i or B_j. A consequence of the above analysis is the fact that follows.

Fact 1. *The shaded set $\tilde{S}_{ij,k}$ is of the form $[a, b]$, where $a < b$, if and only if B_k is not contained in B_i or B_j, and B_k lies completely inside the region defined by the disks B_i and B_j and their common outer tangent lines.*

Clearly a point in VD(S) lies in one of the π_{ij}'s, and in particular it cannot be a shaded point. This is the essence of the following fact.

Fact 2. *A point $p \in \pi_{ij}$ is in VD(S) if and only if $\zeta_{ij}(p)$ is not in the interior of $\tilde{S}_{ij,k}$, $\forall\ k \neq i, j$.*

As we discussed in Section 5.3, events occur when three disks have a common tangent line or four disks have a common tangent ball, or when two disks become tangent. Since the total number of times that two disks can become tangent is $O(n^2)$ and the total number of times that three disks have a common tangent line is $O(n^3)$, the number of events in our KDS is dominated by the total number of times that four disks have a common exterior or interior tangent ball. We can get the immediate upper bound of $O(n^4)$ for constant degree pseudo-algebraic motions. However, we show a significantly smaller upper bound of $O(n^3\beta(n))$.

Our main task is to prove the following theorem.

Theorem 19. *Suppose that S is a set of n disks. When the disks in S move pseudo-algebraically, the Voronoi diagram VD(S) changes $O(n^3\beta(n))$ times.*

In order to prove our result we need the following facts.

Fact 3. *The Voronoi diagram of three disks changes $O(1)$ times.*

Fact 4. *The distance function $\delta(B_i, B_j)$ consists of $O(1)$ rational arcs with constant degree when B_i and B_j move pseudo-algebraically with constant degree.*

For three disks B_1, B_2 and B_3 recall that $S_{12,3}$ is the set of points that are on π_{12} and shaded by B_3. The parameters $\tilde{S}_{12,3}$ of this shaded set may be of the form

$\emptyset$, $(-\infty, a]$, $[b, +\infty)$, $(-\infty, a] \cup [b, +\infty)$, $[a, b]$ or $(-\infty, +\infty)$. We define the function $\phi_{12,3}(t)$ as follows. If at time t, π_{12} is half-shaded by B_3 at a, then $\phi_{12,3}(t)$ is defined to be a. Otherwise, it is undefined. Since an end point of $\tilde{S}_{12,3}$ corresponds to the parameter value of a Voronoi vertex when considering B_1, B_2 and B_3 only, Facts 3 and 4 say that $\phi_{12,3}$ consists of $O(1)$ pieces of rational arcs. Likewise, we define the function $\phi_{ij,l}$ for each triplet i, j, l. For a pair of disks B_i and B_j, we have a family of functions $\Phi_{ij} = \{\phi_{ij,l} \mid l \neq i, j\}$. Let $\Gamma(\Phi_{ij})$ be the upper envelope of a set of functions Φ_{ij}. We first show that

Lemma 12. *A cocircularity event can be charged to a break point on $\Gamma(\Phi_{ij})$ or the overlay between $\Gamma(\Phi_{ij})$ and $-\Gamma(\Phi_{ji})$, for some $i \neq j$.*

Proof. Suppose that at time t, a cocircularity event happens to B_1, B_2, B_3 and B_4. We claim that among those four disks, there always exist two, say B_1 and B_2, so that $\tilde{S}_{12,3}$ and $\tilde{S}_{12,4}$ are not closed intervals with the form $[a, b]$. This claim stems from Fact 1.

Suppose v is the coincident Voronoi vertex at time t. Let $x = \zeta_{12}(v)$. By Fact 2, at time t, there cannot be any other B_i ($i \neq 1, 2, 3, 4$) such that x is in the interior of $\tilde{S}_{12,i}$. Since $\tilde{S}_{12,3}$ is not a closed interval, either $\phi_{12,3}(t) = x$ or $\phi_{21,3}(t) = -x$. The same argument applies to $\phi_{12,4}(t)$ and $\phi_{21,4}(t)$. The fact that v is not shaded by any other B_i allows us to charge such an event either to a break point on $\Gamma(\Phi_{12})$ or $\Gamma(\Phi_{21})$ or to an intersection between $\Gamma(\Phi_{12})$ and $-\Gamma(\Phi_{21})$. □

Given the above lemma we can prove our main theorem.

Proof. (Theorem 19) As we have already mentioned the Voronoi diagram changes when four disks become cocircular, i.e., they have a common tangent ball. By Lemma 12, these events can be charged to break points on the lower or upper envelopes of Φ_{ij}'s or their overlay. Since each $\phi_{ij,l}$ consists of $O(1)$ pieces of rational arcs, the complexity of $\Gamma(\Phi_{ij})$ is bounded by $\lambda_s(n) = n\beta(n)$, for some constant s. The overlay between two envelopes has the same complexity. Thus, the number of cocircularity events is bounded by $\sum_{i,j} O(1)n\beta(n) = O(n^3\beta(n))$. □

5.5 Closest Pair Maintenance

In this section we discuss how to maintain the closest pair of a set S of disks. The distance function that we use is given by relations (5.1). The trivial way to do the maintenance is to consider all $\binom{n}{2}$ pairs of disks and maintain the one of minimum distance with respect to the distance function (5.1). It turns out, however, that there always exists an edge in ADT(S) between the two disks comprising the closest pair. This suggests that we only need to look at $O(n)$ edges in order to determine the closest pair. We assume throughout this section that the closest pair is unique, i.e., if B_1, B_2 is the closest pair in S then $\delta(B_1, B_2) < \delta(B_i, B_j)$, for all $(i, j) \neq (1, 2)$, with $i \neq j$.

We first state three facts for the closest pair with respect to the distance function (5.1).

Lemma 13. *The closest pair in S is realized either between two non-trivial disks or between a trivial and a non-trivial disk.*

Proof. Let B_1, B_2 be the closest pair in S. Suppose that both B_1 and B_2 are trivial disks. We assume, without loss of generality, that $r_2 \leq r_1$. By the definition of the disk distance function we have $\delta(B_1, B_2) \geq -r_2$. Let D be a container of B_1. Then we have $\delta(B_1, D) = -r_1 \leq -r_2 = \delta(B_1, B_2)$, which contradicts the fact that B_1, B_2 is the closest pair. Hence B_1 cannot be a trivial disk. □

Lemma 14. *If the closest pair is realized between a trivial and a non-trivial disk, the non-trivial disk is the only container of the trivial disk.*

Proof. Let B_1, B_2 be the closest pair in S and let B_2 be the trivial disk. Let D be a container of B_2. Suppose that B_1 is not a container of B_2. Then by the definition of the disk distance function we have that $\delta(B_1, B_2) > -r_2 = \delta(D, B_2)$, which contradicts the fact that the disks B_1, B_2 are the closest pair. Hence B_1 is a container of B_2. The uniqueness of B_1 as a container of B_2 stems from the uniqueness assumption on the closest pair. □

Lemma 15. *If the closest pair is realized between a trivial and a non-trivial disk, then the trivial disk is the largest trivial disk in S.*

Proof. Let B_1, B_2 be the closest pair in S and let B_2 be a trivial disk. By Fact 14, B_1 is a container of B_2, i.e., $\delta(B_1, B_2) = -r_2$. Suppose B_3 is another trivial disk in S such that $r_3 > r_2$. Let D be a container of B_3. Then $\delta(D, B_3) = -r_3 < -r_2 = \delta(B_1, B_2)$, which is a contradiction. Hence B_2 is the largest trivial disk in S. □

We are now ready to state and prove the main theorem of this section.

Theorem 20. *Let B_1, B_2 be the closest pair in S. Then there exists an edge $[b_1, b_2]$ in ADT(S).*

Proof. We will prove the theorem by contradiction. Assume that B_1 and B_2 is the closest pair, but there does not exist an edge $[b_1, b_2]$ in ADT(S).

Let $\delta(B_1, B_2) \geq 0$ (see Fig. 5.13(top)). By Lemmas 13 and 14 both B_1, B_2 must be non-trivial. Let q_1, q_2 be the points of tangency of $\omega(o_{12}, B_1)$ with B_1, B_2, respectively (see Fig. 5.13(top)). Suppose that there exists a third disk $B_3 \in S$ such that $B_3 \cap \omega(o_{12}, B_1) \neq \emptyset$. Let $q_3 \in B_3 \cap \omega(o_{12}, B_1)$. Since $q_3 \in \omega(o_{12}, B_1)$, the edge q_1q_2 is the hypotenuse of the triangle $q_1q_2q_3$. Hence,

$$\delta(B_1, B_3) \leq d(q_1, q_3) \leq d(q_1, q_2) = \delta(B_1, B_2),$$

which contradicts the assumption that B_1, B_2 is the closest pair. Therefore $B_i \cap \omega(o_{12}, B_1) = \emptyset$, for all $i \neq 1, 2$, which by Theorem 16 implies that there exists an edge $[b_1, b_2]$ in ADT(S).

Let $\delta(B_1, B_2) < 0$, and let both B_1 and B_2 be non-trivial (see Fig. 5.13(bottom)). Since B_1, B_2 do not share an edge in DT(S), there exists a non-trivial disk B_3 that contains $\omega(o_{12}, B_1)$ (see Fig. 5.13(bottom)). Let C be the interior tangent ball of B_1 and B_3 that has its center on the line b_1b_2, and let r be the (absolute value of the) radius of C. Let r_{12}, r_{13} be the (aboslute values of the) radii of $\omega(o_{12}, B_1)$ and $\omega(o_{13}, B_1)$, respectively. Then

$$\delta(B_1, B_3) = -2r_{13} \leq -2r \leq -2r_{12} = \delta(B_1, B_2),$$

which contradicts the fact that B_1, B_2 is the closest pair in S. Hence B_1 and B_2 share an edge in ADT(S).

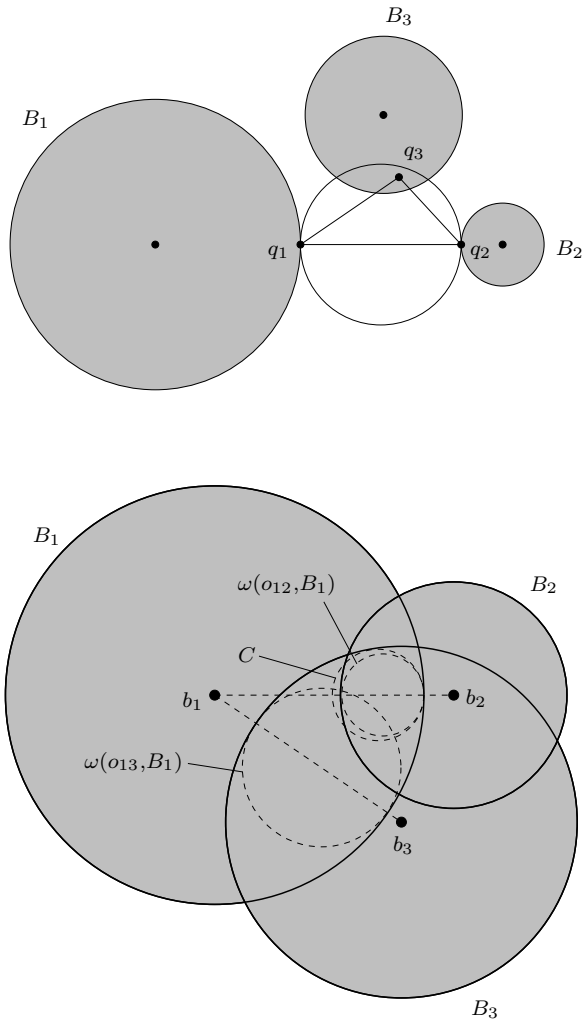


Figure 5.13: Proof of Theorem 20. Top: the case $\delta(B_1, B_2) \geq 0$. Bottom: the case $\delta(B_1, B_2) < 0$ and both B_1, B_2 are non-trivial.

Finally, let $\delta(B_1, B_2) < 0$ and let B_2 be a trivial disk. Then, by Lemmas 13 and 14, B_1 has to be non-trivial and B_1 is the only container of B_2. Clearly, there exists an edge $[b_1, b_2]$ in $\mathrm{ADT}(S) \setminus \mathrm{DT}(S)$. □

Now that we have established that we only need to look at the edges of the $\mathrm{ADT}(S)$ to find the closest pair, we simply need to maintain a *tournament tree* T on the edges of $\mathrm{ADT}(S)$. Before describing how to actually maintain T we need some definitions. Let t_1 and t_2 be two nodes of T. We say that $t_1 \prec t_2$ if the depth of t_1 is smaller than

the depth of t_2 in T, or if t_1, t_2 are of the same depth and t_1 is to the left of t_2 in T. A node t_1 is *adjacent* to a node t_2 in T if they have the same parent. A node t is a *loser* if its parent is its adjacent node. Finally, a node t is a *winner* if its parent is itself.

The certificates associated with T are the winner-loser relationships. T changes due to changes in the winner-loser relationships or due to changes of the ADT(S), because of cocircularity, appearance and disappearance events. When a winner-loser relationship changes we simply propagate the new winner up the tree, deschedule the old winner-loser relationships and schedule the new ones. During this propagation we visit $O(\log n)$ nodes of the tree and schedule/deschedule $O(\log n)$ certificates in total. Hence the cost per winner-loser relationship change is $O(\log^2 n)$ (see Fig. 5.14). When an edge disappears we replace the corresponding leaf node with the last loser leaf node and delete the last winner and loser leaf nodes. Then we propagate the last loser leaf node up the tree as in the case of a winner-loser relationship change. Again this takes $O(\log^2 n)$ time (see Fig. 5.15). Finally, when an edge appears we create two new leaf nodes in the tree: one for the new edge and one for the first leaf node. We attach the two new nodes under the current first leaf node and propagate the

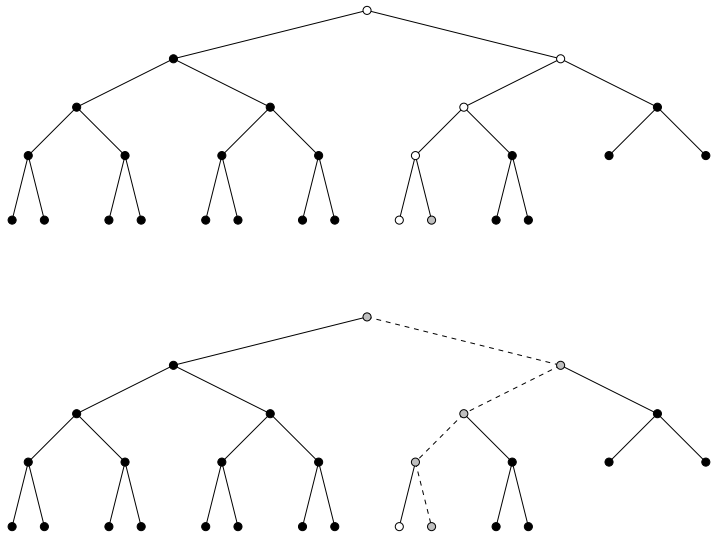


Figure 5.14: A case in which the a winner-loser relationship changes. Top: the white node is the closest pair. Bottom: the gray node becomes the closest pair; the gray node is propagated all the way up the tournament tree.

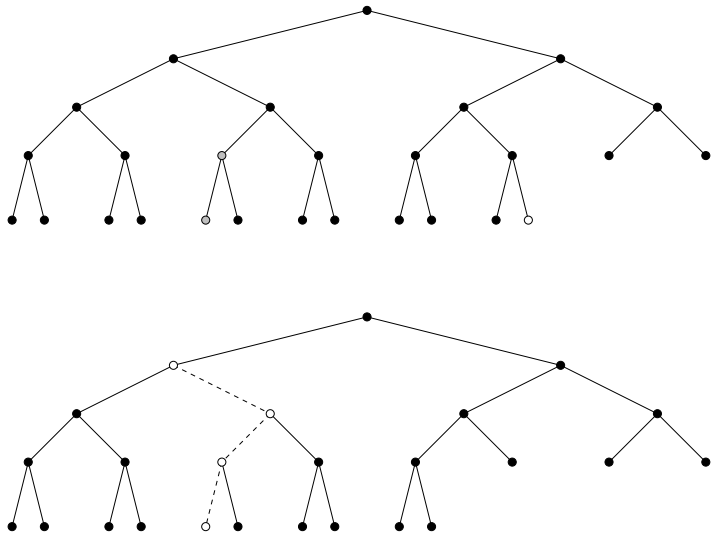


Figure 5.15: Removing a node from the tournament tree. The gray node is the one to be deleted. The white node is the last loser leaf node. Top: the tree before the removal of the gray node. Bottom: the tree after the removal of the gray node; the white node is propagated up the tree.

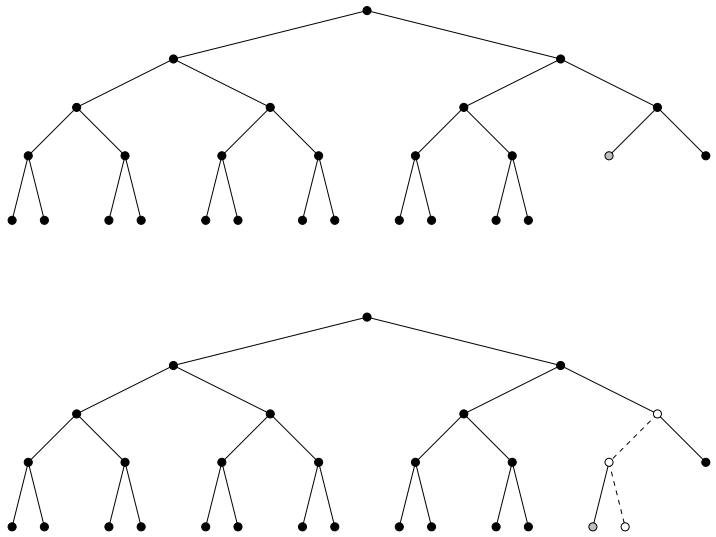


Figure 5.16: Adding a node to the tournament tree. The gray node is the the first leaf node. The white node is the new node. Top: the tree before the addition of the new node. Bottom: the tree after the addition of the new node; the new node is propagated up the tree.

winner between these two nodes up the tree (see Fig. 5.16). Once again this takes $O(\log^2 n)$ time.

The number of changes in a single winner-loser relationship is constant for disks moving along pseudo-algebraic trajectories of constant degree. Hence the number of events that we have to process is dominated by the number of combinatorial changes of ADT(S), which is $O(n^3\beta(n))$. All, but the appearance event, are processed in $O(\log^2 n)$ time; the appearance event is processed in $O(n)$ time.

5.6 Kinetic Connectivity of Disks

In this section we discuss how to kinetically maintain the connectivity for a set of disks of different radii. The problem for unit disks has already been studied in [20].

The connectivity graph K of a set of disks S is defined as follows. The vertices of K are the centers of the disks in S. Two disks share an edge in K if they intersect. Let G be the subgraph of ADT(S) defined as follows. An edge e in ADT(S) belongs to G if and only if it is an edge between two non-trivial intersecting disks or between a trivial disk and its container disk. Clearly, G is a subgraph of the connectivity graph K of S (modulo multiple edges between two disks in DT(S)). We show that G captures all the connectivity infromation that K has. This is really important since the size of K is $\Omega(n^2)$ in the worst case, whereas the size of G is $O(n)$. The main result of this section is the following.

Theorem 21. *If $B_1, B_2 \in S$ belong to the same connected component in K, then there exists a path in G that connects B_1 and B_2.*

In order to prove the above theorem we need a few definitions and a preparatory lemma. Let G_o be the subgraph of G that consists of the edges connecting non-trivial intersecting disks. Let G_r be the subgraph of G that consists of the edges that connect trivial disks with their containers. Let us consider first the case that the disks B_1, B_2 are non-trivial and intersect. Then the following lemma gives the desired result.

Lemma 16. *Let B_1 and B_2 be two non-trivial intersecting disks. Then there exists a path in G_o that connects B_1 and B_2.*

Proof. If there exists an edge $e_{12} \in \mathrm{DT}(S)$ between B_1 and B_2, then $e_{12} \in G_o$, and we are done. If such an edge does not exist, consider the line segment $[b_1, b_2]$ connecting the centers of B_1 and B_2 and let $V_1, \ldots, V_k$ be the sequence of Voronoi cells that $[b_1, b_2]$ intersects. Let $B_1 \equiv C_1, \ldots, C_k \equiv B_2$ be the corresponding disks. Consider the intersection p of the Voronoi edge corresponding to the pair C_i, C_{i+1} with the segment $[b_1, b_2]$. Since $p \in V_i \cap V_{i+1}$, there exists an edge $[c_i, c_{i+1}]$ in $\mathrm{DT}(S)$. Since B_1 and B_2 intersect $p \in B_1$ or $p \in B_2$. We can assume without loss of generality that $p \in B_1$. This implies that $\delta(p, B_1) \leq 0$. Since $p \in V_i$ we must have $\delta(p, C_i) \leq \delta(p, B_1) \leq 0$. Similarly, $\delta(p, C_{i+1}) \leq 0$. This implies that p lies in $C_i \cap C_{i+1}$, i.e., the disks C_i and C_{i+1} intersect. Hence the edge $[c_i, c_{i+1}] \in \mathrm{DT}(S)$ is in G_o. Moreover since $p \in B_1 \cap C_i \cap C_{i+1}$, the disks C_i and C_{i+1} lie in the same connected component as B_1. Consider the path defined by the edges $[c_i, c_{i+1}]$, $i = 1, \ldots, k-1$. This is a path that lies entirely in G_o, since $[c_i, c_{i+1}] \in G_o$, for all $i = 1, \ldots, k-1$. □

Now we are ready to prove the main theorem.

Proof. (Theorem 21) Let us assume first that B_1 and B_2 are non-trivial. Since B_1 and B_2 belong to the same connected component in K, there exists a sequence of non-trivial disks $B_1 \equiv C_1, \ldots, C_k \equiv B_2$ such that $C_i \cap C_{i+1} \neq \emptyset$. By Lemma 16 there exists a path P_i in G_o that connects C_i with C_{i+1} for all $i = 1, \ldots, k-1$. Consider the union U of these paths. Clearly, U is a path in G_o that connects B_1 to B_2.

If B_1 and/or B_2 are trivial, then find the roots R_1 and R_2 of the trees in G_r that B_1 and B_2 belong to. Since R_1, R_2 are non-trivial, there exists a path P_o in G_o that connects R_1 and R_2. Let P_1, P_2 be the paths in G_r that connect B_1 to R_1 and R_2 to B_2, respectively. Then the path $P \equiv P_1 P_o P_2$ is a path in G that connects B_1 to B_2. □

We can maintain the connectivity of the disks using the dynamic graph data structure of Holm, de Lichtenberg and Thorup [27]. This data structure supports edge insertions and deletions in $O(\log^2 n)$ amortized time, and connectivity queries in $O(\log n / \log\log n)$ time. The graph that we maintain is the graph G defined above. Once we have $\mathrm{ADT}(S)$, maintaining G is really simple. First we color the edges of $\mathrm{ADT}(S)$ as follows: edges between non-intersecting non-trivial disks are *green*, edges between intersecting non-trivial disks are *orange* and edges between trivial disks and

their containers are *red*. Clearly, G is the union of orange and red edges. The color of an edge changes if the corresponding disks become tangent. In particular, whenever a green edge becomes an orange edge or whenever an orange or a red edge appears we simply add it to G. Whenever an orange edge becomes a green edge or whenever an orange or a red edge disappears we simply delete it from G. Since the cost per insertion/deletion of edge in G is $O(\log^2 n)$, in the amortized sense, the cost per update of G is $O(\log^2 n)$ (amortized), except when we have an appearance event, in which case the update cost is $O(n)$. The number of times that two disks, moving along pseudo-algebraic trajectories of constant degree, can become tangent is constant. This implies that the number of events due to disk tangencies is $O(n^2)$. The total number of events for maintaining G is thus dominated by $O(n^3\beta(n))$, which is the number of times that the Delaunay triangulation of the set of disks can change combinatorially.

5.7 Near Neighbor Maintenance

Suppose that we have a set S of *non-intersecting* moving disks in two dimensions. Let P be a disk in S for which we want to know the disks in S that are within a certain, possibly time varying, distance R_P from P. Let C_P be the disk centered at P with radius R_P. The obvious approach is to maintain the distance from P to every other disk in S and keep those that intersect C_P. In fact, we can do better than that. If we are maintaining the DT of S, the only disks that can enter or exit C_P are those that are end points of edges of the DT crossing C_P exactly once. This is the essence of the following theorem.

Theorem 22. *Let $\mathcal{T}(S)$ be the DT(S) and let $P \in S$. If a disk $Q \in S$ enters/exits the disk C_P at some time t_0, then there exists an edge in $\mathcal{T}(S)$ between Q and some disk that intersects C_P.*

Proof. At time t_0, Q is tangent to C_P. Let $\{C_R\}$ be the family of balls with center R that are tangent to Q, lie inside C_P and their center lies on the segment $c_Q K_P$, where c_Q, K_P are the centers of Q and C_P, respectively (see Fig. 5.17). Consider the ball $C_{R'}$ such that R' is at maximal distance from Q and the ball $C_{R'}$ does not

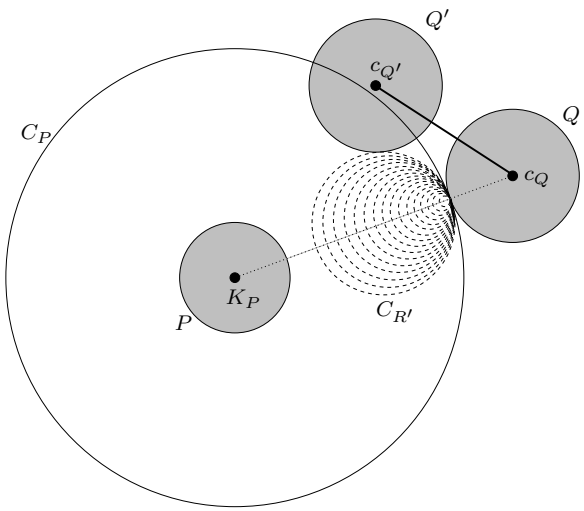


Figure 5.17: Proof of Theorem 22.

intersect the interior of any disk in S. Let $Q' \neq Q$ be the disk that $C_{R'}$ is tangent to. Q' always exists since the set A is non-empty ($P \in A$). Let $c_{Q'}$ be the center of Q'. The edge $[c_Q, c_{Q'}]$ is an edge in DT(S), since $C_{R'}$ does not intersect the interior of any disk in S. □

Maintaining the near neighbors of P then reduces to maintaining the DT of S and updating the set E_P of DT edges, one end disk of which intersects C_P and the other does not. The set E_P changes when disks enter or exit C_P. Edge flips due to the maintenance of DT(S) may also change E_P. In case we want to maintain the k-nearest neighbors of P the same algorithm applies with two slight modifications: (1) the distance R_P is defined to be the distance of the center of P from P_k, where P_k is the k-th nearest neighbor of P and (2) the edges of DT(S) adjacent to P_k are all included in E_P.

We shall omit the details of the algorithms since they are essentially the same as the corresponding algorithms for points in Section 4.2.

5.8 Conditions for the Cocircularity Event

In this section we present the algebraic conditions associated with cocircularity events. A cocircularity event happens when four distinct disks have a common exterior or

interior tangent. If one of the four disks is the disk at infinity, the cocircularity event is equivalent to requiring that three distinct disks have a common tangent line. The conditions of Theorem 24 have been derived in [19] for the case of non-intersecting disks. It turns out that they are still valid in the case of intersecting disks. Although the proofs of Theorems 23 and 24 can be directly derived using the analysis presented in [19, Theorem 1], we present them here for the shake of completeness. Note that the conditions presented below are algebraic functions of the disk motions.

Theorem 23. *Let $B_i = \{(x_i = x_i(t), y_i = y_i(t)), r_i\}$, $i = 1, 2, 3$, be three distinct disks moving on the plane. The first time that these disks have a common tangent line such that all disks lie on the same side of the line is the smallest root t_0 of the equation*

$$a^2 + b^2 - c^2 = 0, \tag{5.2}$$

where

$$a = \begin{vmatrix} x_1 & r_1 & 1 \\ x_2 & r_2 & 1 \\ x_3 & r_3 & 1 \end{vmatrix}, \quad b = \begin{vmatrix} y_1 & r_1 & 1 \\ y_2 & r_2 & 1 \\ y_3 & r_3 & 1 \end{vmatrix}, \quad c = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

Proof. Let $L = \{\alpha x + \beta y + \gamma = 0,\ \alpha^2 + \beta^2 = 1\}$ be the line that is tangent to all three disks B_i, $i = 1, 2, 3$. The condition that the line L is tangent to B_i, $i = 1, 2, 3$, can be written as

$$\alpha x_i + \beta y_i + \gamma = r_i, \quad i = 1, 2, 3.$$

Using Cramer's rule we can immediately find the solution of this system in terms of α, β, γ. In particular,

$$\alpha = \frac{1}{D} \begin{vmatrix} r_1 & y_1 & 1 \\ r_2 & y_2 & 1 \\ r_3 & y_3 & 1 \end{vmatrix}, \quad \beta = \frac{1}{D} \begin{vmatrix} x_1 & r_1 & 1 \\ x_2 & r_2 & 1 \\ x_3 & r_3 & 1 \end{vmatrix}, \quad \gamma = \frac{1}{D} \begin{vmatrix} x_1 & y_1 & r_1 \\ x_2 & y_2 & r_2 \\ x_3 & y_3 & r_3 \end{vmatrix},$$

where

$$D = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

However, in order for L to exist α and β must satisfy $\alpha^2 + \beta^2 = 1$, which immediately

gives relation (5.2). □

Theorem 24. *Let $B_i = \{(x_i = x_i(t), y_i = y_i(t)),\ r_i\}$, $i = 1, 2, 3, 4$, be four distinct disks moving on the plane. The first time that these disks have a common exterior or interior tangent disk is the smallest root t_0 of the equation*

$$a^2 + b^2 - c^2 = 0, \tag{5.3}$$

satisfying the conditions

$$\frac{ax_i^* + by_i^*}{dp_i} \leq 1, \quad i = 1, 2, 3, \tag{5.4}$$

where

$$a = \begin{vmatrix} y_1^* & r_1^* & p_1 \\ y_2^* & r_2^* & p_2 \\ y_3^* & r_3^* & p_3 \end{vmatrix}, \quad b = \begin{vmatrix} x_1^* & r_1^* & p_1 \\ x_2^* & r_2^* & p_2 \\ x_3^* & r_3^* & p_3 \end{vmatrix}, \quad c = \begin{vmatrix} x_1^* & y_1^* & p_1 \\ x_2^* & y_2^* & p_2 \\ x_3^* & y_3^* & p_3 \end{vmatrix}, \quad d = \begin{vmatrix} x_1^* & y_1^* & r_1^* \\ x_2^* & y_2^* & r_2^* \\ x_3^* & y_3^* & r_3^* \end{vmatrix},$$

and $x_i^* = x_i - x_4$, $y_i^* = y_i - y_4$, $r_i^* = r_i - r_4$, $p_i = (x_i^*)^2 + (y_i^*)^2 - (r_i^*)^2$, $i = 1, 2, 3$.

Proof. We can assume without loss of generality that B_4 is the disk of smallest radius r_4. Let $r_i^* = r_i - r_4$, $i = 1, 2, 3, 4$. Consider the disks $Z_i = (z_i, r_i^*)$, $i = 1, 2, 3, 4$, where $z_i = (x_i, y_i)$ is the center of the disk Z_i. Clearly, the problem of determining the times that the disks B_i, $i = 1, 2, 3, 4$, have a common tangent ball is equivalent to the problem of finding the times for which there exists a ball that is tangent to Z_i, $i = 1, 2, 3$, and passes through the point Z_4. We think of the disks Z_i as embedded on the complex plane. Let (z, r) be the ball that is tangent to Z_i, $i = 1, 2, 3$, and passes through Z_4. Note that (z, r) is an exterior tangent ball to all Z_i, $i = 1, 2, 3$. Consider the transformation $W = W(z) = 1/(z - z_4)$. This tranformation maps circles on the Z-plane that do not pass through z_4 to circles on the W-plane, and circles on the Z-plane that pass through z_4 to lines on the W-plane. Under this transformation the disks Z_i, $i = 1, 2, 3$, map to the disks $W_i = (w_i, \rho_i) = ((u_i, v_i), \rho_i)$, $i = 1, 2, 3$, where

$$w_i = \frac{\overline{z_i - z_4}}{p_i}, \quad \rho_i = \frac{r_i^*}{p_i}, \quad p_i = |z_i - z_4|^2 - (r_i^*)^2.$$

Note that all $p_i > 0$, since no one of the four original disks is contained inside another. The ball (z, r) is mapped to a line L. Let $L = \{\alpha u + \beta v + \gamma = 0, \alpha^2 + \beta^2 = 1\}$ be the line that is tangent to all W_i, $i = 1, 2, 3$. Note that under the transformation $W = W(z)$, the point at infinity is mapped to zero. Since the infinite point and all Z_i, $i = 1, 2, 3$, lie on the same side of the ball (z, r), the circle W_i, $i = 1, 2, 3$, and zero must lie on the same side of the line L. This requirement can be formulated as

$$\frac{\alpha u_i + \beta v_i + \gamma}{\gamma} \geq 1, \quad i = 1, 2, 3,$$

or equivalently

$$\frac{\alpha u_i + \beta v_i}{\gamma} \geq -1, \quad i = 1, 2, 3. \tag{5.5}$$

The condition that L is tangent to the disks W_i, $i = 1, 2, 3$, can be written as

$$\alpha u_i + \beta v_i + \gamma = \rho_i, \quad i = 1, 2, 3.$$

Using Cramer's rule we immediately get the following solution in terms of α, β and γ.

$$\alpha = \frac{1}{D}\begin{vmatrix} \rho_1 & v_1 & 1 \\ \rho_2 & v_2 & 1 \\ \rho_3 & v_3 & 1 \end{vmatrix}, \quad \beta = \frac{1}{D}\begin{vmatrix} u_1 & \rho_1 & 1 \\ u_2 & \rho_2 & 1 \\ u_3 & \rho_3 & 1 \end{vmatrix}, \quad \gamma = \frac{1}{D}\begin{vmatrix} u_1 & v_1 & \rho_1 \\ u_2 & v_2 & \rho_2 \\ u_3 & v_3 & \rho_3 \end{vmatrix},$$

where

$$D = \begin{vmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{vmatrix}.$$

However, α and β must satisfy $\alpha^2 + \beta^2 = 1$, which implies

$$\begin{vmatrix} \rho_1 & v_1 & 1 \\ \rho_2 & v_2 & 1 \\ \rho_3 & v_3 & 1 \end{vmatrix}^2 + \begin{vmatrix} u_1 & \rho_1 & 1 \\ u_2 & \rho_2 & 1 \\ u_3 & \rho_3 & 1 \end{vmatrix}^2 = \begin{vmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{vmatrix}^2.$$

Substituting u_i, v_i and ρ_i in terms of x_i^*, y_i^*, r_i^* and p_i, we get relation (5.3).

Finally, it is easy to verify that

$$\alpha = -\frac{1}{D'}\begin{vmatrix} y_1^* & r_1^* & p_1 \\ y_2^* & r_2^* & p_2 \\ y_3^* & r_3^* & p_3 \end{vmatrix}, \quad \beta = \frac{1}{D'}\begin{vmatrix} x_1^* & r_1^* & p_1 \\ x_2^* & r_2^* & p_2 \\ x_3^* & r_3^* & p_3 \end{vmatrix}, \quad \gamma = \frac{1}{D'}\begin{vmatrix} x_1^* & y_1^* & r_1^* \\ x_2^* & y_2^* & r_2^* \\ x_3^* & y_3^* & r_3^* \end{vmatrix},$$

where

$$D' = \begin{vmatrix} x_1^* & y_1^* & r_1^* \\ x_2^* & y_2^* & r_2^* \\ x_3^* & y_3^* & r_3^* \end{vmatrix}.$$

Substituting the above expressions in equalities (5.5) and expressing u_i and v_i in terms of x_i^*, y_i^* and p_i, we get inequalities (5.4). □

5.9 Conclusion

In this chapter we showed how to kinetically maintain the Voronoi diagram for a set of disks moving in the plane. The key steps in the kinetization process where the introduction of the Augmented Delaunay triangulation and the establishment of the relationship between the local and global Delaunay properties. We also proved that the closest pair of the set of disks is realized between two disks that share an edge in the ADT, and that a properly chosen subset of the ADT is a spanning subgraph of the connectivity graph of the set of disks. Based on these properties, and using the ADT as the underlying structure, we showed how to maintain the closest pair and the connectivity of the disks as the disks move. Finally, we showed how to maintain the disks that are within a prescribed distance from a reference disk and how to maintain the k-nearest neighbors of a reference disk.

We strongly believe that the results presented in this paper can be generalized to general *additively weighted Voronoi diagrams*, in which the weights can be positive as well as non-positive. We would also like to extend the results presented here to general smooth convex objects or to environments where obstacles are present. Finally, the best known lower bound on the number of combinatorial changes of the DT is $\Omega(n^2)$, whereas our upper bound is $O(n^3\beta(n))$. Given this upper bound, the

algorithms presented here for maintaining the DT, the closest pair and disk connectivity are not efficient; it would be of interest to find kinetic data structures that solve these problems efficiently, or prove a tighter lower or upper bound on the number of combinatorial changes of the DT.

Chapter 6

Interval Methods for Kinetic Simulations

Discrete-event simulation is commonly used by geometric algorithms to solve a variety of problems, in both the static and kinetic settings. Many classic geometric algorithms are of the sweep-line type (or sweep-plane, etc, in higher dimensions), in which a sweep is used to reduce a static problem in a given dimension to a dynamic problem in one less dimension. A sweep algorithm typically maintains an event queue, where the event times are the moments when the sweep line needs to stop and perform updates to the data structures it maintains, including the event queue itself. A famous classic example of such an algorithm is the Bentley-Ottmann line-sweep algorithm [10] for detecting all intersections among a set of line segments in the plane. All kinetic data structures are also based on an event queue, where the event times are certificate failures associated with the proof of correctness of a computation of the KDS attribute of interest. When a certificate fails, the proof, and with it the event queue, needs to be updated. In both cases a priority queue on event times is maintained and the algorithm repeatedly advances the clock to the next event and updates the queue. We will refer to both of these scenarios as *kinetic simulations*, because they involve the continuous evolution of a system punctuated by discrete events.

The calculation of an event time is frequently a non-trivial computational task. For example, it may involve computing the intersection of three-surfaces in the sweep-plane case, or the time when moving points become coplanar or co-spherical, in the

kinetic case. The cost of such computations cannot always be justified in terms of the final result that needs to be computed. For example, almost all kinetic simulations involve the *de-scheduling of events* — these are events that will not happen because the associated certificates were removed from the proof, and the computational resources that went into their event-time calculation will be wasted. In general, of course, it is hard to know, at the time an event is scheduled, that it will be de-scheduled at some future time in the kinetic simulation. Moreover, in many sweep and kinetic problems the exact time when events happen may not be needed, as long as we can guarantee that the correct sequence of events will be generated. This is exactly the case in the classic Bentley-Ottmann sweep, where the output is a purely combinatorial list of intersecting pairs of segments.

In this chapter we present an approach for reducing the cost of event-time calculations in kinetic simulations, through the use of interval methods, akin to interval arithmetic [1]. Instead of exact event times, we will focus on time intervals guaranteed to contain one or more events. The key intuition is that we do not need to know very precisely events scheduled to happen far into the future. We want to devote computational resources to refining the intervals associated with these events only as they get closer to the present time in the simulation. By using intervals for all event-queue operations as well, we are often able to resolve comparisons between event times without further refinement of the associated intervals. The result is that we are able to generate the correct sequence of events for the kinetic simulation, but at a substantial savings in the cost of the event-time calculations.

To realize and evaluate experimentally this idea, we concentrate in this chapter on events whose event times can be calculated by solving polynomial equations. Since almost without exception kinetic certificates are low degree polynomial functions of attributes (e.g., positions) of a small number of the moving bodies (e.g. the `CCW` or `InCircle` tests), this restriction covers the case where the motions themselves are polynomial (in the sweep case an equivalent condition is that the curves or surfaces involved are polynomial functions). Polynomials are an attractive class of functions to consider, because efficient root isolation methods for them have been well studied. We specifically make use of the technique of *standard sequences* to determine intervals containing the roots (event times) of interest. Furthermore, more general

functions can often be well approximated by polynomials in certain ranges (e.g., Taylor expansions). Our techniques can be extended to this case by including additional certificates whose failure indicates that a particular approximation is invalid and a new polynomial approximation needs to be generated.

The remaining sections of the chapter are as follows. In Section 6.1 the overall framework of a kinetic simulation is described in more detail. In Section 6.2 we present the algebraic and analytic tools used to develop our algorithm, i.e., we present the notion of standard sequences, as well as Sturm's and Bolzano's theorems. Then in Section 6.3 we present the details of the interval-based kinetic scheduler, including the refinement and update policies for intervals isolating polynomial roots, and the priority queue maintenance using intervals as opposed to exact event times. In Section 6.4 we provide an informal theoretical justification of the advantages of our approach. In Section 6.5 we provide a framework for comparing the interval and ordinary schedulers and present empirical data on the superiority of the interval approach. In Section 6.6 we discuss a trade-off between degree and number of pieces for splined motion trajectories. Finally we conclude in Section 6.7 with some additional remarks.

6.1 Kinetic Simulations

The inner loop of a kinetic simulation is the maintenance of the associated event queue. The entries of the event queue are the future failure times of the certificates currently in the kinetic proof — we will call these the *active* certificates. At each step of the kinetic simulation the next certificate to fail is obtained from the priority queue and the kinetic proof is updated to accommodate the altered state of the world. As a result, typically a number of active certificates leave the proof (and event queue) and a number of other new certificates enter the proof and become active.

Each certificate is typically a simple algebraic inequality on the positions/poses of a small number of features of the moving objects. In fact, in most kinetic simulations only a small number of different types of certificates are ever used (for example, a kinetic Voronoi/Delaunay simulation for point sites can be done using only the `InCircle` test).

The above considerations motivate the following formulation of the problem: let $\mathcal{S}$ be a set of polynomials $\{f_1(t), f_2(t), \ldots, f_k(t)\}$ (corresponding to the certificates in the KDS), the real roots of which represent possible events in our simulation. There is a notion of a current time t_0 and we are interested in quickly finding the time t_1, which is the smallest root of any of the f_i greater that t_0. Then we perform some changes in the set $\mathcal{S}$ and advance in time by setting $t_0 \leftarrow t_1$.

The naive solution to this problem is the following: for each polynomial f_i compute all its roots to the required precision, discard those that are complex and insert its smallest real root greater than the current time into the event queue (we can think of the event queue as a priority queue implemented using a heap). Some methods for computing the roots of a polynomial are the Jenkins/Traub method [29], the eigenvalue method [51, 45] in which we construct the companion matrix of the polynomial and compute its eigenvalues, Muller's and Laguerre's methods [45] and a more recent method by Lang and Frenzel [35]. Among these methods the last one, although very accurate for high degree polynomials, is rather expensive. Muller's, Laguerre's and the Jenkins/Traub methods, however, are not stable enough for polynomials of degree greater than 60 or so. Thus we decided to adopt the eigenvalue method for both our theoretical analysis as well as the implementation of the KDS. As already mentioned, our goal is to avoid spending resources in computing real roots that correspond to events that may never happen, or complex roots that are of no interest for our simulation. In addition, we want to compute the real roots of the polynomials only to such accuracy as required to resolve root comparisons and determine which polynomial among the two being compared has the earliest failure time.

The approach that we employ in this work is to use the *standard sequence* [28] of a polynomial f in the queue to maintain an ordered list of intervals that contain and isolate its real roots. The leftmost among these intervals is the one that represents the certificate for f in the priority queue. The comparison of two polynomials in the queue is done by comparing the intervals and splitting them as necessary. This process of resolving comparisons, as well as the forward stepping in time related to the update of the current time t_0, cause us to refine these interval lists and obtain tighter bounds on the roots of the polynomials.

The main advantages of this approach are as follows. First of all, operations on the interval list are only performed when needed, that is when the information obtained so far about the roots is not sufficient to resolve root comparisons, and thus to determine the relative priority of the two polynomials in the queue. These operations are focussed on the smallest root of each polynomial, rather than all the roots at the same time, thus avoiding spending computation time on possible events that may eventually not happen. Moreover, if we were to use a symbolic algebra system for performing computations, then our algorithm could be implemented with exact operations — unlike the naive method which must always resort to numerical techniques.

6.2 Mathematical Preliminaries

Let $y = \{y_1, y_2, \ldots, y_m\}$ be a finite sequence of non-zero numbers. We define the *number of variations in sign* of y to be the number of indices i, $1 \leq i \leq m-1$, such that $y_i y_{i+1} < 0$. If $y = \{y_1, y_2, \ldots, y_m\}$ is an arbitrary sequence of numbers, then we define the number of variations in sign of y to be that of the subsequence y' obtained by dropping the zeros in y. For the example the number of sign variations of $\{4.5, 0, 0, 0.5, -1.3, 0, 10^{-30}, 4, -200\}$ is 3.

Let now $f(x)$ be a polynomial of positive degree with real coefficients. Then the sequence of polynomials $\{f_0(x),\ f_1(x),\ \ldots,\ f_s(x)\}$ defined by repeated division as:

$$
\begin{aligned}
f_0(x) &= f(x) && (6.1)\\
f_1(x) &= f'(x) && (6.2)\\
f_0(x) &= q_1(x) f_1(x) - f_2(x) && (6.3)\\
&\vdots\\
f_{i-1}(x) &= q_i(x) f_i(x) - f_{i+1}(x) && (6.4)\\
&\vdots\\
f_{s-1}(x) &= q_s(x) f_s(x) \quad (\text{i.e., } f_{s+1} = 0), && (6.5)
\end{aligned}
$$

where the degrees of the f_i monotonically decrease, is called the *standard sequence*

for $f(x)$ [28]. As it can easily be verified, the $f_i(x)$, $i \geq 2$ are obtained by modifying Euclid's algorithm for finding the g.c.d. of $f(x)$ and $f'(x)$ in such a way that the last polynomial obtained at each stage is the negative of the remainder[1] in the division process:

$$f_{i+1}(x) = -f_{i-1}(x) \bmod f_i(x), \quad i = 1, \ldots, s-1. \tag{6.6}$$

In view of the above, $f_s(x)$ is the g.c.d. of $f(x)$ and $f'(x)$. In particular, if $f_s(x)$ is a constant polynomial then $f(x)$ has no multiple roots. Moreover, if $f_s(x)$ is not a constant polynomial, set $g_i(x) = f_i(x)/f_s(x)$, $0 \leq i \leq s$. Then $g_0(x)$ has the same roots as $f(x)$, but now all the roots of $g_0(x)$ are simple. Moreover the sequence $\{g_i(x)\}_{i=0}^{s}$ is the standard sequence for $g_0(x)$.

Using the notion of standard sequences just described we are ready to state *Sturm's theorem*, which addresses the problem of counting the number of real roots of a polynomial in an interval of the real line:

Theorem 25 (Sturm's Theorem [2]). *Let $f(x)$ be a polynomial of positive degree with real coefficients and let $\{f_0(x) = f(x),\ f_1(x) = f'(x), \ldots,\ f_s(x)\}$ be the standard sequence for $f(x)$. Assume $[a, b]$ is an interval of the real line such that $f(a) \neq 0$, $f(b) \neq 0$. Then the number of distinct real roots of $f(x)$ in (a, b) is $V_a - V_b$ where V_c denotes the number of variations in sign of $\{f_0(c), f_1(c), \ldots, f_s(c)\}$.*

It is shown in [54] that the roots of $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ lie in $[-\alpha, \alpha]$, where

$$\alpha = 1 + \frac{\max\{|a_{n-1}|, \ldots, |a_0|\}}{|a_n|}. \tag{6.7}$$

Hence, if $\mu = \alpha + \epsilon$ (for some $\epsilon > 0$) and $\{p_0(x) = p(x),\ p_1(x) = p'(x), \ldots,\ p_s(x)\}$ is the standard sequence of $p(x)$, then the total number of distinct real roots of $p(x)$ is $V_{-\mu} - V_{\mu}$, where, as before, V_c is the number of variations in sign of $\{p_0(c), p_1(c), \ldots, p_s(c)\}$.

Another very well-known theorem that will be of use is *Bolzano's theorem*:

Theorem 26 (Bolzano's Theorem). *Let $f(x)$ be a continuous real valued function in the interval $[a, b]$, and assume that $f(a)$ and $f(b)$ have opposite signs, i.e.,*

[1] An algorithm for finding such division remainders for two polynomials with real coefficients can be found in [34].

[2] Sturm's theorem holds true for polynomials with coefficients in any real closed field R. The statement of the theorem that we present deals only with the case of real coefficients (see [28] for the generic statement).

$f(a)f(b) < 0$. *Then there is at least one* c *in the open interval* (a, b) *such that* $f(c) = 0$.

Based on the above two theorems we can define the two primitives we will use, $T_f(a, b)$ and $S_{f,N}(a, b)$. The primitive $T_f(a, b)$ counts the number of distinct real roots of the polynomial $f(x)$ in (a, b), provided that $f(a)f(b) \neq 0$. The primitive $S_{f,N}(a, b)$ gives the number of (distinct) real roots of $f(x)$ in (a, b) provided that $f(a)f(b) \neq 0$ and that the number of real roots of $f(x)$ in (a, b) does not exceed N.

The purpose of introducing the second primitive is that if we know that $N = 1$ and that $f(x)$ has only simple roots, if any, in (a, b), then we can determine the number of real roots of $f(x)$ in (a, b) by simply checking whether $f(a)f(b) < 0$ or not. This test is much cheaper, especially for high degree polynomials, than the one suggested by *Sturm's Theorem*. Note that if $f(x)$ has only simple roots in (a, b) and $\deg f(x) \geq 2$, then $T_f(a, b)$ is equivalent to $S_{f,N}(a, b)$ for $N = \deg f(x)$.

Finally, given two polynomials $f(x)$ and $g(x)$ and an interval $[a, b]$ we can determine, using the above machinery, whether they have a common real root in $[a, b]$. This can be done by computing the g.c.d. h of f and g, and then using the above mentioned predicates to determine if h has a real root in $[a, b]$.

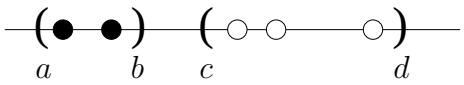
6.3 The Interval-Based Kinetic Scheduler

In order to reduce the cost of event time calculations in the event queue, we keep intervals that contain one or several of those event times. This choice enables us to avoid wasting computing resources when the comparison between two event times can be resolved by considering their corresponding intervals. Moreover, by our approach, we avoid spending computing time on events that are scheduled to happen in the future, and which may be de-scheduled before their time of occurrence (e.g., because the *flight plans* of the objects in the kinetic simulation have changed in the meantime, or because of other changes in the kinetic proof). By using intervals we are able to focus only on the first event time associated with a certain certificate that is greater than the current time; computations that have to do the remaining event times related to the certificate in question are postponed until later, when they are actually needed. Clearly, the better estimates we have for our event times, the easier the comparisons

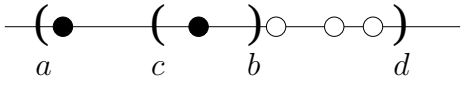
are. For this purpose, if during the process of comparing event times based on their interval representation we get finer bounds on these times we store and use these to facilitate other comparisons that are performed during the process of updating the event queue.

Let $f(x)$ be a polynomial that represents one of the certificates in our kinetic simulation. With each such polynomial we associate an ordered interval list (a_1, b_1), ..., (a_m, b_m) such that $b_i \leq a_{i+1}$, $i = 1, \ldots, m-1$, $f(a_i)f(b_i) \neq 0$ and $T_f(a_i, b_i) > 0$, $i = 1, \ldots, m$. All real roots of $f(x)$ greater than the current time t_c are contained in one of these intervals (initially the list consists of a single interval containing all the real roots of $f(x)$). Suppose now that we want to determine, among two polynomials $p(x)$ and $q(x)$, which is the one that corresponds to the earliest event time, i.e., which is the one that has the smallest real root (greater than t_c). We can also think of these event times as the *priorities* of $p(x)$ and $q(x)$ in the event queue; the question, therefore, is which, among $p(x)$ and $q(x)$, has the greater priority. Let (a, b) and (c, d) be the leftmost intervals in the lists of $p(x)$ and $q(x)$, respectively. We can assume that the roots of these polynomials are simple, since otherwise we can replace the polynomials, as described in the previous section, with others that have only simple roots. Without loss of generality we can also assume that $a \leq c$ (otherwise we can interchange the roles of $p(x)$ and $q(x)$). The procedure that we follow to determine the relative priority of $p(x)$ and $q(x)$ is described below. Note that we test a condition only if all the previous ones have failed. The black dots in the figures below depict the real roots of $p(x)$, whereas the white dots depict the real roots of $q(x)$.

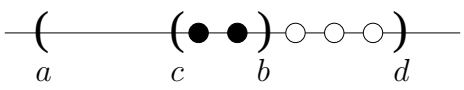
1. if $b \leq c$ then the smallest root of $p(x)$ will be smaller than the smallest root of $q(x)$, and thus $p(x)$'s priority will be greater than that of $q(x)$.



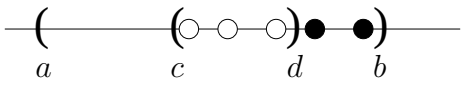
2. if $p(x)$ has any roots in $(a, c]$ then `priority`$(p(x))$ > `priority`$(q(x))$.



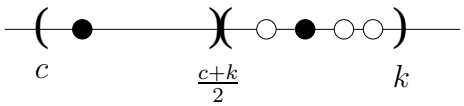
3. if $b \leq d$ and $q(x)$ has all its roots in $[b, d)$, then the priority of $p(x)$ is greater than that of $q(x)$.



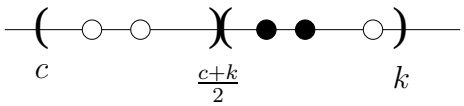
4. if $b > d$ and all the roots of $p(x)$ are in the interval $[d, b)$, then `priority`$(p(x))$ $<$ `priority`$(q(x))$.



5. if both polynomials have some of their roots in the interval (c, k), where $k = \min\{b, d\}$, then we need to employ a subdivision-like approach: we split (c, k) in the middle and check if the real roots of $p(x)$ and $q(x)$ are distributed in such a way in the two resulting intervals that we can directly determine their relative priority. For example, if $p(x)$ has roots in $(c, \frac{c+k}{2})$ and $q(x)$ does not then `priority`$(p(x))$ $>$ `priority`$(q(x))$.



If $p(x)$ does not have any roots in $(c, \frac{c+k}{2})$ and $q(x)$ does then `priority`$(p(x))$ $<$ `priority`$(q(x))$.



We recursively continue this subdivision process until we can determine which of the two polynomials has higher priority.

If the smallest roots of the two polynomials are not the same then the subdivision procedure terminates; on the other hand if they share that root then something else has to be done. What we do is the following test: if the two polynomials have only

one root in the interval of interest we check to see if that root is a common one. To do so we compute their g.c.d. and check if it has a root in the interval of interest. The natural question that arises is how we can be sure that the polynomials will have only one root in that interval.

At various points during the algorithm an interval (α, γ), associated with a polynomial $f(x)$, needs to be split in two parts (α, β) and (β, γ). If $f(\beta) \neq 0$, and $f(x)$ has roots in both intervals then we replace (α, γ) with (α, β) and (β, γ). If only one of the two intervals contains roots of $f(x)$ then we simply update the corresponding endpoint. If $f(\beta) = 0$, then find a point β' to the left or to the right of β, such that $f(\beta') \neq 0$ and do the splitting using that point. Since we have to do these interval splits anyway in order to determine the relative priority of the two polynomials we basically get for free better bounds on the roots of both polynomials. However, these better approximations of the roots are only computed when the information obtained so far is not sufficient to determine which polynomial is of higher priority. Moreover, splitting the intervals results in interval lists that will eventually contain only a single real root of the polynomial in question, which is important for determining if two polynomials have a common real root.

So far we did not properly take into account that we have a current time t_c and that we are interested only in real roots larger than that time. In addition, the current time is a critical event of the kinetic simulation, and thus it is represented as a root of some polynomial $c(x)$. Therefore we do not know it exactly, but rather we have an interval (α_c, β_c) in which it lies. This interval is the first interval in the interval list associated with $c(x)$. We can actually assume that the current time is the only root of the associated polynomial in that interval, since otherwise we can use the midpoint of the interval to split it; we can continue this recursively until we get a list in which the first interval contains only one root, which is going to be t_c. To take account of this fact, the interval list of a polynomial $p(x)$ needs some preprocessing which has to do with discarding the roots that are smaller than t_c. This procedure is as follows:

1. discard all of the intervals (a_i, b_i) such that $b_i \leq \alpha_c$ (if any) and then renumber the remaining ones.

2. if $\beta_c \leq a_1$ there is nothing more to be done: we have already kept all those roots that are greater than t_c.

3. if $\beta_c > a_1$, we check if t_c is in $(\alpha_c, a_1]$; in that case we simply update our bounds for t_c.

4. if $b_1 \leq \beta_c$ then

   (a) if t_c is in $[b_1, \beta_c)$, then delete (a_1, b_1), update α_c, and proceed in the same manner with the new (a_1, b_1).

   (b) if t_c is not in $[b_1, \beta_c)$, then it has to be in (r, b_1), where $r = \max\{a_1, \alpha_c\}$, in which case we update the bounds for t_c and employ the subdivision approach in (r, b_1).

5. if $b_1 > \beta_c$ and $p(t)$ has at least one root in $(r, \beta_c]$, where r is defined as above, then we employ the subdivision approach in $(r, \beta_c]$.

A similar pruning approach can be applied when we want to run the simulation up to a time T_{max}, where T_{max} is assumed not to be an event time. In that case we discard of all the intervals (a_k, b_k) such that $T_{max} \leq a_k$. Let (a_ℓ, b_ℓ) be the last interval in the list; clearly, $a_\ell < T_{max}$. If $b_\ell \leq T_{max}$ then we do nothing; otherwise, we just replace (a_ℓ, b_ℓ) with (a_ℓ, T_{max}), if (a_ℓ, T_{max}) contains any roots of the associated polynomial, or discard it altogether.

On several occasions we have talked about determining whether a polynomial $f(x)$ has real roots in a certain interval (a_i, b_i) or about how many roots there are. The primitive that we can use in these cases is $T_f(a_i, b_i)$, the most generic one among the two we introduced in the previous section. There are instances, however, where we can do better. If we store the number n_i of real roots of the interval (a_i, b_i), then we can use the primitive $S_{f,n_i}(c, d)$, whenever $a_i \leq c \leq d \leq b_i$, which is always the case when we split intervals. This way we can take advantage of the very simple test that the primitive S incorporates if $n_i = 1$ and $f(x)$ has simple roots in (a_i, b_i).

6.4 A Theoretical Justification

In this section we present a very simple model for the distribution of the event times and perform a worst-case analysis for two methods: our interval-based approach and

a method which computes all roots of the certificate polynomials by computing the eigenvalues of the corresponding companion matrix.

In particular, let s be the number of active polynomials and let us assume that each polynomial has d real roots which are random i.i.d. variables in $[0, T_{max}]$, where T_{max} is the time until when we run our kinetic simulation. Let also m be the total number of KDS events occurring during the simulation, and assume that at each event k old certificates (polynomials) leave the event queue and k new certificates enter the queue. Then the expected separation between the event times, i.e., the roots of the polynomials is $\frac{T_{max}}{ds}$, whereas the expected separation between roots of the same polynomial is $\frac{T_{max}}{d}$. We will also assume that the event queue is implemented using a heap-like structure, so that insertions and deletions in the queue take logarithmic time in the queue size.

The cost of the eigenvalue method is $O(d^2K)$ where K is the number of iterations performed [51]. In particular, if we want to compute the eigenvalues with accuracy equal to ε, then

$$K = O(\frac{\log \varepsilon}{\log \max_{1 \le i \le d-1} \frac{|\lambda_{i+1}|}{|\lambda_i|}}) \tag{6.8}$$

where λ_i are the roots of the polynomial satisfying $|\lambda_{i+1}| \le |\lambda_i|$, $1 \le i \le d-1$. In our case we can assume that $\lambda_i \ge 0$, $\forall i$ (the roots represent time values). In view of our assumption that the roots are evenly distributed and that their distance is $\frac{T_{max}}{d}$ in expectation, we get that

$$\max_{1 \le i \le d-1} \frac{|\lambda_{i+1}|}{|\lambda_i|} \le \frac{d-1}{d} \le \frac{d}{d+1} \tag{6.9}$$

which implies that

$$K = O(\frac{\log \varepsilon}{\log \frac{d}{d+1}}) = O(d \log \frac{1}{\varepsilon}) \tag{6.10}$$

Since the roots are expected to be $\frac{T_{max}}{ds}$ apart from each other, we only need an accuracy $\varepsilon = \Theta(\frac{T_{max}}{ds})$ which implies that

$$K = O(d(\log d + \log s))$$

Hence, the total cost per event using the eigenvalue method is $O(kd^3(\log d + \log s))$.

Consider now the interval-based method. This method needs $O(\log ds)$ steps to resolve the comparison between two polynomials (because of the subdivision-like approach that we employ) and at each step the cost is $O(d^2)$ (this is the cost to compute the predicates $T_f(a, b)$ or $S_{f,N}(a, b)$). Therefore the total cost per event is $O(kd^2(\log d + \log s))$, which is a factor of d better than that of the eigenvalue method. As we will see in the next section the numerical experiments support the above theoretical calculation.

6.5 Numerical Experiments

We implemented the algorithm that was described in Section 6.3 for two KDSs: one for maintaining the Delaunay triangulation (DT) and one maintaining the closest pair (CP) of a set of points moving on the plane. The points are moving on trajectories of the form $(x(t), y(t))$ where both $x(t)$ and $y(t)$ are polynomials of degree d. The coefficients of these polynomials are chosen uniformly from $[-1, 1]$, except their constant term which is chosen uniformly from $[0, 1]$. In the case of the DT the certificates correspond to `InCircle` tests of quadruples of points, hence the degree of the certificates is at most $4d$. In the case of the CP the certificates are polynomials of degree $2d$ or d, that corresponding to comparisons of squared distances for points in the plane or to comparisons of the projections of points along certain (fixed) directions. The details for the certificates for both simulations can be found in [7].

In our examples the number n of moving points is between 10 and 20, whereas the degree d of their motion varies from 2 to 40 in the DT simulation, and from 2 to 80 in the CP simulation. For every pair (n, d) we computed the running times using three different approaches:

(a) the "naive" method, in which we compute all the roots of a polynomial, throw away those that are complex, and use the real ones to resolve the event time comparisons; the roots of a polynomial are computed by constructing the companion matrix and computing its eigenvalues [45, 51],

(b) the interval-based approach that we have already described, and

(c) a hybrid method, in which we isolate the real roots of the polynomial using the

predicates $T_f(a, b)$ and $S_{f,N}(a, b)$ and then use a standard root finding technique, like the bisection method [45], to compute the root of the polynomial in each interval.

For each pair (n, d) we started with 10 different initial configurations of points. The experiments were performed on an SGI workstation using an R10000/195 MHz processor.

What we can see from the results, as shown in Figures 6.1 and 6.2, is that the eigenvalue method in the DT case, is superior for motion degrees up to 6, whereas for the CP case it is superior for motion degrees up to 13. This should be attributed to the overhead of the interval method due to the evaluation of the standard sequence for each polynomial. However for certificates of higher degree the interval-based method is superior to the eigenvalue method. In fact the data shows that we gain a speed-up factor of order d, where d is the degree of the motion, independently of n. The same can be observed when comparing the eigenvalue and the hybrid methods. The hybrid method, however seems to be a constant factor worse that the interval-based method; this can be attributed to two facts: first of all, the hybrid method computes *all* the real roots of each certificate and not only those that are after the current time; secondly, the roots are computed to greater accuracy than needed in order to resolve the comparisons in the priority queue.

6.6 Degree vs. Events

The cost of a kinetic simulation is an increasing function of the algebraic degree of the motions — more complex motions imply more time-consuming event-time calculations. At the same time, this cost is also an increasing function of the number of events that have to be processed. In this section we consider a trade-off between these two costs. By approximating a high-degree motion by a sequence of lower-degree motions, we can reduce the cost of event-time calculations, while at the same time adding the cost of processing the flight plan updates that must happen at motion segment boundaries. We can actually view this issue backwards as well: if we approximate splined polynomial motions with single polynomials of high degree, then we eliminate events that have to do with flight plan updates, but at the same time we increase the

cost of processing the simulation events. Of course kinetic simulations are chaotic systems and there is absolutely no guarantee that the approximated system will have the same sequence of events as the original. Nevertheless, according to our experience, approximations such as the above do preserve the overall character of the simulation as well as various global statistics, and thus are meaningful and useful under certain circumstances.

To examine this trade-off, consider n points moving each along a single parametric polynomial trajectory of degree d_H. We approximate the motion of the points, with motions of lower degree d_L, $d_L < d_H$ in the following manner: we densely sample each higher order trajectory and then perform a constrained least squares fit to the sampled data. The number of time samples is equal to m and these are uniformly distributed in the time interval of interest; we will discuss the choice of m in the sequel. We impose the constraint that the original and the approximating motions must coincide at the endpoints of each of the time intervals of the approximation (we need to maintain at least C^0 continuity for the splined motion). The interval of approximation is initially the entire time interval for which we run our simulation. We obtain a measure of closeness between the original and approximating trajectories by combining the distances between corresponding points on the two trajectories at each of the sampled times using the L_∞-norm. If this closeness measure fails to be below some prespecified threshold value, then we split the interval of approximation at the sample time of maximum error and recursively repeat the approximation procedure for the two subintervals. The cost of each approximation step is $\Theta(md_L^2)$, dominated by the constrained least squares calculation. The number of times we need to repeat the process will be analyzed below.

In our setting, we want to compute a polynomial segment of degree d_L that approximates a segment of degree d_H to within an error ϵ in the L_∞-norm (or split the interval if that is not possible). We accept a single segment only when we can establish that it meets this criterion. Peetre [42] shows that the error introduced by doing a discrete instead of a uniform polynomial approximation is $O(1/m^2)$, where m is the number of points used to perform the discrete approximation. Since we calculate the L_∞-norm over a discrete set of m sampled time values, we choose[3] $m = 10/\sqrt{\epsilon}$ so as

[3]The constant 10 here was chosen arbitrarily. The correct constant can be estimated if we have

to guarantee via Peetre a maximum error of $\epsilon/2$ between the discrete and uniform norms. We also make the threshold discussed in the previous paragraph to be $\epsilon/2$. In this way a low degree segment is accepted only when it is known to be within ϵ of the original in the L_∞-norm.

When we approximate a polynomial of degree d_H with one of degree $d_L < d_H$, over the interval $[0, t]$, the error of the approximation is of order $O(t^r)$, where $r = d_L + 1$. Since we want this error to be at most ϵ, we require that $t = O(1/\epsilon^{1/r})$. From this estimate for t we see that the number of low degree polynomial pieces required to approximate the original curve in our simulation will roughly be $O(T_{max}/\epsilon^{1/r})$, where T_{max} is the stop time of the simulation.

The cost of the kinetic simulation, in the model above, is of two types:

1. the cost of resolving comparisons between certificates that have to do with the maintenance of the geometric attribute of interest, and

2. the cost of approximating and updating the trajectories of the moving points.

The first part of the cost is assumed to be equal to the product of the number of events n_{ev} scheduled and descheduled in the priority queue due to the changes in the geometric attribute of interest, times the mean cost of resolving a comparison between event-times. We saw in Section 6.4 that this cost is $O(d^2 \log(ds))$, where d is the degree of the certificates and s is the size of the priority queue. Since the degree of the certificates is typically a constant multiple of the degree of the motion, the total cost due to the maintenance of the geometric attribute is $O(n_{ev} d_L^2 \log(d_L s))$. The second part of the cost has to do with the scheduling and descheduling of events that correspond to changes in the motion, as well as the cost to approximate the original motion with one of lower degree. The cost for these priority queue updates is $O(\log s)$, yielding a total cost of $O(n_{mc}[\log s + d_L^2/\sqrt{\epsilon}\,])$ for this second part, where n_{mc} is the number of events associated with the motion changes. Following the analysis in the previous paragraph, $n_{mc} = \Theta(n\, T_{max}/\epsilon^{1/r})$, where $r = d_L + 1$; thus, the total cost for our simulation is $O(n_{ev} d_L^2 \log(d_L s) + n\, T_{max}\, [\log s + d_L^2/\sqrt{\epsilon}\,]/\epsilon^{1/r})$. In many KDSs the size of the proof to be maintained is linear with respect to the number of objects in the simulation, therefore s is taken to be equal to the number of points n.

a priori knowledge of the maximum acceleration of the particles.

The expression for the total cost of the simulation then becomes $O(n_{ev} d_L^2 \log(d_L n) + n\, T_{max}\, [\log n + d_L^2/\sqrt{\epsilon}\,]/\epsilon^{1/r})$.

Assuming that the approximation is accurate enough, the number of events n_{ev} is only a function of d_H. Under this assumption, and for fixed ϵ, it is clear that as d_L decreases, we expect the cost of updating the geometric attribute to decrease and the remaining cost to increase. However, the remaining cost consists of two different parts which behave differently as d_L changes. In particular, for small d_L, since we have a lot of low degree polynomials, the cost of updating the event queue is large; for large d_L, the cost of the approximation dominates. Moreover, we can expect a monotone increase in the cost of the simulation as the error ϵ decreases.

In order to examine the validity of the above analysis we considered $n = 5$ moving points at 10 initial random positions. The geometric attribute that we want to maintain in this experiment is the Delaunay triangulation of the points. The degree of their original trajectory is $d_H = 32$ and the degrees d_L of the approximate trajectories vary from 30 to 2. The stop time for the simulations is $T_{max} = 1$. Our earlier assumptions hold true, namely that the degree of the certificates d is a constant multiple of the degree of the motion d_H or d_L and that the size of the priority queue s is linear in the number of points n. Figure 6.3 depicts the average running times as a function of the degree of the motion d_L for several errors ϵ. The square corresponds to the simulation where the original trajectory is used. The interval-based approach described in this paper was used to do the simulations.

The main observations are the following :

1. The cost of the simulation increases monotonically as we increase the accuracy. This is in agreement with our model.

2. For fixed accuracy and for decreasing d_L, the cost of the simulation at first decreases, reaches a minimum and then increases. Initially the cost of the simulation is dominated by the cost of computing the approximating motions (due to their large degree); as the degree d_L decreases further, the number of curve pieces needed for the approximation starts to go up and the cost now is dominated by the updates of the motion in the event queue. This, again, is a behavior consistent with the model presented above. The bumps appearing in the graphs should be attributed to the changes in the combinatorial structure

of the simulation and the fact that the point where we perform the split in our recursive subdivision of the approximation intervals may not be exactly optimal.

3. The degree of minimal total cost, when we do the approximation, increases as the accuracy increases. Furthermore, for low accuracy, this minimal cost is smaller than the cost when we do not approximate at all, while for high accuracy this optimal cost is higher than the original. This can also be explained by our model since the cost of the approximation increases, both in terms of the number of polynomial pieces required to approximate the original trajectory and the cost of the least squares fit, which, as we saw, depends on the imposed accuracy.

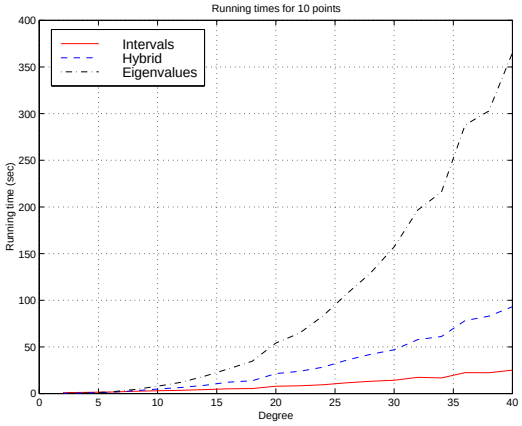
The lesson from these experiments is that, if we require very accurate approximate trajectories, then we are better off performing the simulation without doing the approximation and taking advantage of the speed-up provided by our interval approach. If accuracy is not an issue, then a smaller degree will be advantageous and the above analysis and experimental data offer some guidance on the choice of the optimal degree.

6.7 Conclusion

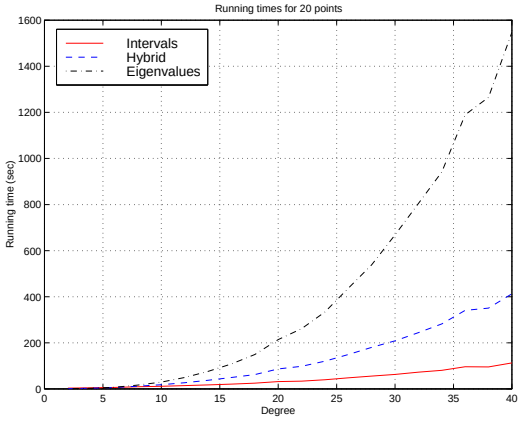
In this chapter we have presented an interval-based method for maintaining kinetic simulations of objects that move on polynomial trajectories. The major idea of the method is to use intervals that contain the event times of the simulation in order to resolve the comparisons between events times in the event queue, thus avoiding wasting time on computing event times for events that may never occur, or computing them more accurately than needed. Experimental results, as well as a simple theoretical analysis, show that by using the interval-based method we gain a speed-up of d, where d is the degree of motion, over the naive approach.

Although polynomial motions constitute a common class of motions, we would like to extend our approach to more general motions. In particular, we would like to explore the possibility of applying our algorithm to motions that are solutions of ordinary differential equations either by exploiting existing theoretical results (see

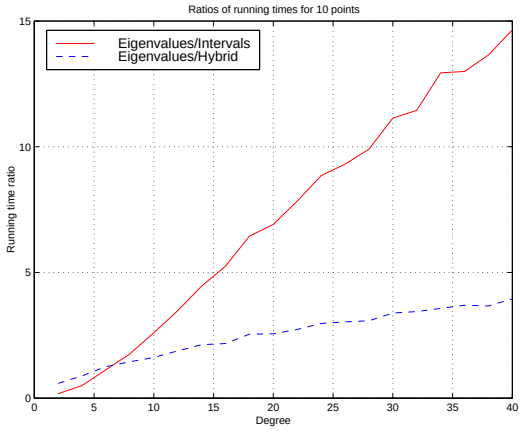
[33]) or by approximating the solution of the o.d.e. by a polynomial function and then adding additional certificates in the kinetic simulation corresponding to the times that the particular approximations are no longer valid. Another possible direction of research is to use interval arithmetic techniques to obtain bounds on function values (see [46]) for non-polynomial functions, and use these bounds as the basis for our approach.



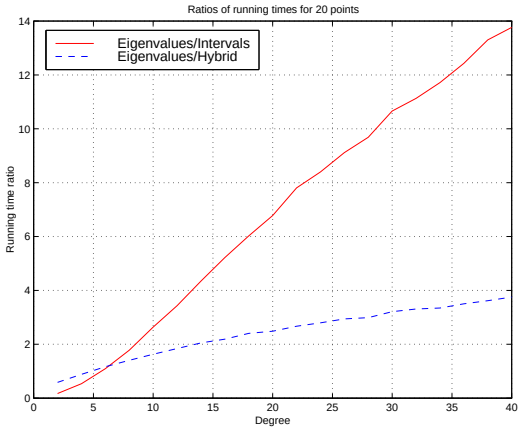
(a) DT: running times for 10 points



(b) DT: running times for 20 points

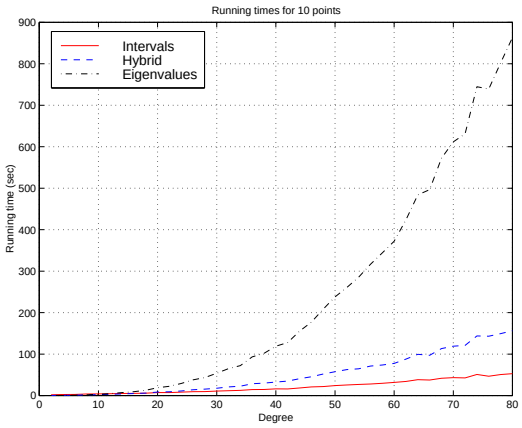


(c) DT: ratios of running times for 10 points

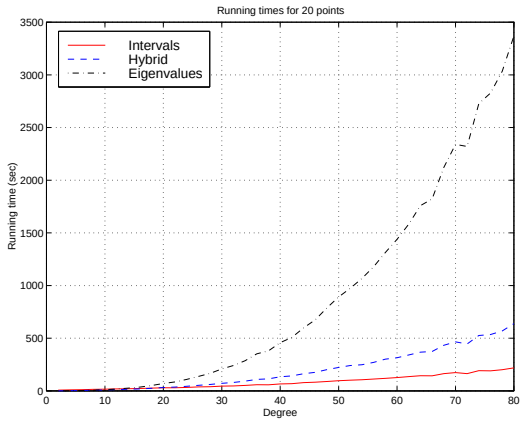


(d) DT: ratios of running times for 20 points

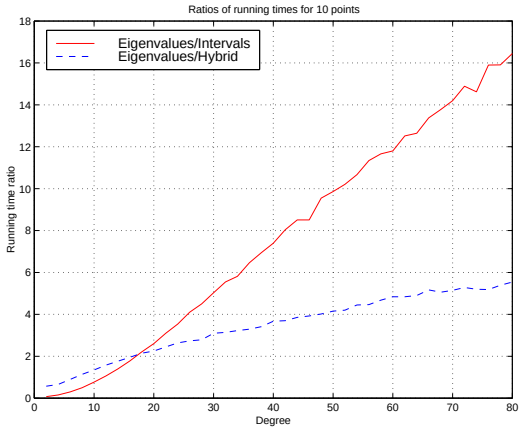
Figure 6.1: Mean running times in seconds and ratios of running times for maintaining the Delaunay triangulation of 10 and 20 points on a plane using the three different methods for handling the events times: the interval-based, the eigenvalue one and a hybrid one; 10 initial configurations were used for each point set.



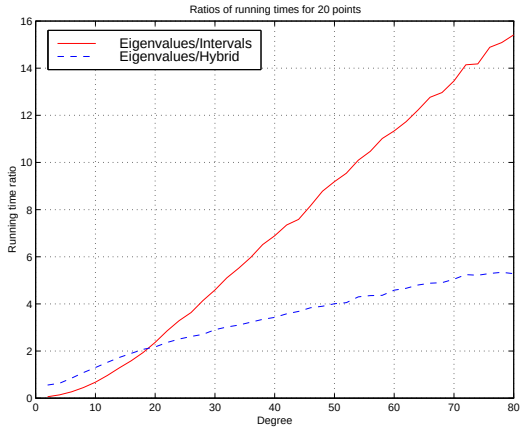
(a) CP: running times for 10 points



(b) CP: running times for 20 points



(c) CP: ratios of running times for 10 points



(d) CP: ratios of running times for 20 points

Figure 6.2: Mean running times in seconds and ratios of running times for maintaining the closest pair of 10 and 20 points on a plane using the three different methods for handling the events times: the interval-based, the eigenvalue one and a hybrid one; 10 initial configurations were used for each point set.

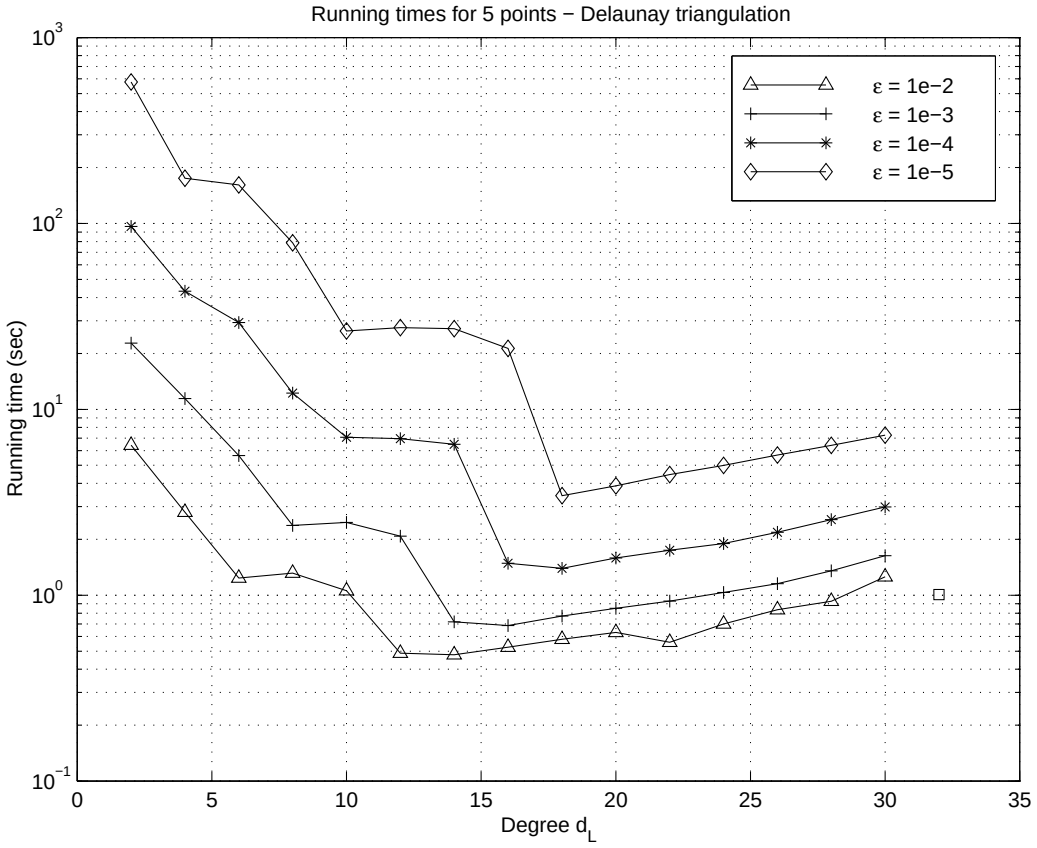


Figure 6.3: Mean running times in seconds for maintaining the Delaunay triangulation of 5 moving points as a function of the degree d_L of the approximate splined motions. The points are moving originally on polynomial trajectories of degree $d_H = 32$; the running time for the simulation using the original trajectory is shown by a square. Four different values for ϵ are considered: 10^{-i}, $i = 2, 3, 4, 5$. The stop time is $T_{max} = 1$. 10 initial configurations are used for each point set. The interval-based method is applied.

Chapter 7

Conclusion

In this thesis we presented several results in the theory of sparse spanner graphs and kinetic Voronoi diagrams. In particular, we showed that two classes of triangulations, namely bounded aspect ratio triangulations and the Constrained Delaunay triangulation, are spanner graphs. We also presented algorithms for efficiently maintaining near neighbors of points in unconstrained or constrained environments. I used the Kinetic Data Structures (KDS) framework for maintaining the geometric structures of interest as the points move. Using the same framework we described how to maintain the Euclidean Voronoi diagram for possibly intersecting disks moving on the plane. Given the Voronoi diagram of the set of disks, we can also maintain other proximity structures such the closest pair of the disks or near neighbors of disks. Finally, we showed an algorithm for speeding up kinetic simulations when the representations of the motions of the geometric objects are polynomials of high degree. The algorithm is based on representing the roots of the certificates of the kinetic simulation, which are high degree polynomials in this case, using intervals.

7.1 Approximate Shortest Path Maintenance

In many applications we are interested in knowning shortest paths between geometric objects. However, the objects in many cases are moving, in which case we would like to be able to maintain these shortest paths. This problem seems to be very difficult, so alternatively what we would like to do is maintain approximate shortest paths, i.e.,

paths that are within a constant factor of the optimal ones. In this context sparse spanner graphs are of great importance. Sparse spanner graphs guarrantee that they contain such a path and since they are of small size they should be much more easily maintainable that denser graphs. Although, we know how to maintain some spanner graphs, such as the Delaunay triangulation, we do not know how maintain approximate shortest paths on them.

In the same context, small stretch factors are important if we want to have good approximations to the actual shortest paths. Hence it would be of interest to find the optimal stretch factor for the Delaunay triangulation, or close the gap between the upper and lower bounds for the optimal stretch factor for bounded aspect ratio triangulations presented in Chapter 3.

7.2 Kinetic Bounded Aspect Ratio Triangulations

Although bounded aspect ratio triangulations are important as spanner graphs, they are of great interest independently due to the fact that they are heavily used in meshing applications. In fact, in many of these applications the meshes are moving, which gives rise to the problem of maintaining, in a kinetic setting, bounded aspect ratio triangulations. Another issue in meshing applications in the existence of boundaries, which suggests that we need to look at ways of maintaining constrained or (most probably) conforming bounded aspect ratio triangulations.

There are several questions that one could pose. For example, one issue is the number of Steiner points that we are allowed to use. Another issue is whether existing algorithms for producing bounded aspect ratio triangulations, such as Delaunay refinement, are easily kinetizable. Another problem is the size of the output triangulation as a function of the bound on the aspect ratio and the motion of the points.

7.3 Euclidean Voronoi Diagram for Spheres in 3D

In Chapter 5 we dealt with the problem of maintaining the Euclidean Voronoi diagram for a set of possibly intersecting disks. In would be of great interest to extend the results and algorithms to three dimensions. One would expect that once we know

how to maintain the Euclidean Voronoi diagram for three-dimensional spheres, we would also be able to maintain their closest pair, connectivity and near neighbors, in exactly the same manner as in two dimensions.

The extension does not seem at all straightforward. The Voronoi diagram in 3D consists of conic surfaces and Voronoi cells may not even contain Voronoi vertices (0-dimensional simplices). It would be very helpful, both from the kinetic perspective, but also from the viewpoint of understanding the Voronoi diagram itself, to first establish local conditions that guarrantee the global correctness of the structure. It would be really nice to verify that the dual of the Voronoi diagram is indeed a (generalized) tetrahedrization and to associate all the possible combinatorially different types of Voronoi cells with tetrahedra. Insight in this direction could also possibly yield an optimal algorithm for the construction of the Voronoi diagram using only local operations, just like the (non-optimal) flip-edge algorithm on the plane.

Bibliography

[1] Götz Alefeld and Jürgen Herzberger. *Introduction to interval computations*. Academic Press, New York, 1983.

[2] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9:81–100, 1993.

[3] Franz Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[4] B. Awerbuch. Complexity of network synchronization. *J. ACM*, pages 804–823, 1985.

[5] H. J. Bandelt and A. W. M. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Adv. in Appl. Math.*, 7:309–343, 1986.

[6] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Proc. SIGGRAPH '90*, pages 19–29, 1990.

[7] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms*, pages 747–756, 1997.

[8] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 1:1–28, 1999.

[9] Julien Basch, Leonidas J. Guibas, and G. D. Ramkumar. Sweeping lines and line segments with a heap. In *Proc. 13th Annual Symp. on Comput. Geom.*, pages 469–471, 1997.

[10] J. L. Bentley and Th. Ottman. Algorithms for reporting and counting geometric intersections. *IEEE Trans. on Computers*, C-28(9):643–467, Sept. 1973.

[11] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry*, pages 47–123. World Scientific, 2nd edition, 1995.

[12] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comput. System Sci.*, 48:384–409, 1994.

[13] S. Cameron. Enhancing GJK: computing minimum and penetration distances between convex polyhedra. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 1997.

[14] L. P. Chew. There is a planar graph almost as good as the complete graph. In *Proc. 2nd ACM Symp. on Comput. Geom.*, pages 169–177, 1986.

[15] L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39:205–219, 1989.

[16] G. Das and D. A. Joseph. Which triangulations approximate the complete graph? In *Proc. Internat. Symp. on Optimal Algorithms*, volume 401 of *LNCS*, pages 168–192. Springer-Verlag, Berlin, 1989.

[17] David P. Dobkin, Steven J. Friedman, and Kenneth J. Supowit. Delaunay graphs are almost as good as complete graphs. In *28th Annual Symp. Found. of Comput. Sci.*, pages 20–26, 1987.

[18] D. Eppstein and J. Erickson. Raising proofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 58–67, 1998.

[19] M. Gavrilova and J. Rokne. Swap conditions for dynamic Voronoi diagrams for circles and line segments. *CAGD*, 16:89–106, 1999.

[20] L. J. Guibas, J. Hershberger, S. Suri, and L. Zhang. Kinetic connectivity for unit disks. In *Proc. 16th ACM Symp. on Computat. Geom.*, pages 331–339, 2000.

[21] L. J. Guibas, D. Hsu, and L. Zhang. H-walk: hierarchical distance computation for moving convex bodies. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 344–351, 1999.

[22] L. J. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In G. Schmidt and R. Berghammer, editors, *Proc. 17th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 570 of *LNCS*, pages 113–125. Springer, June 1991.

[23] L. J. Guibas, J. Snoeyink, and L. Zhang. Compact Voronoi diagrams for moving convex polygons. In *Proc. 7th SWAT*, 2000. To appear.

[24] L. J. Guibas and L. Zhang. Euclidean proximity and power diagram. In *Proc. 10th CCCG*, 1998.

[25] P. Gupta, R. Janardan, and M. Smid. Fast algorithms for collision and proximity problems involving moving geometric objects. *Comput. Geom. Theory Appl.*, 6:371–391, 1996.

[26] J. Hershberger. Optimal parallel algorithms for triangulated simple polygons. In *Proc. 8th Annual Symp. on Comput. Geom.*, pages 33–42, 1992.

[27] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proc. 30th ACM STOC*, pages 79–89, 1998.

[28] Nathan Jacobson. *Basic Algebra I.* W. H. Freeman, New York, 2nd edition, 1985.

[29] M. A. Jenkins. Algorithm 493 zeros of a real polynomial. *ACM Transactions on Mathematical Software*, 1:178–, 1975.

[30] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Comput. Aided Geom. Design*, 8:123–142, 1991.

[31] J. M. Keil. Approximating the complete Euclidean graph. In *Proc. SWAT*, volume 318 of *LNCS*, pages 208–213. Springer-Verlag, Berlin, 1988.

[32] J. Mark Keil and Carl A. Gutwin. The Delaunay triangulation closely approximates the complete Euclidean graph. In *Proc. WADS*, volume 382 of *LNCS*, pages 47–56. Springer-Verlag, Berlin, 1989.

[33] A. G. Khovanskiĭ. *Fewnomials*, volume 88 of *Translations of Mathematical Monographs*. Americal Mathematical Society, Providence, Rhode Island, 1991.

[34] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 3rd edition, 1998.

[35] Markus Lang and Bernhard-Christian Frenzel. Polynomial root finding. *IEEE Signal Processing Letters*, 1994.

[36] C. Levcopoulos and A. Lingas. There are planar graphs almost as good as the complete graphs and as short as minimum spanning trees. In *Proc. Internat. Symp. on Optimal Algorithms*, volume 401 of *LNCS*, pages 9–13. Springer-Verlag, Berlin, 1989.

[37] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proc. IEEE Intern. Conf. Robot. Autom.*, volume 2, pages 1008–1014, 1991.

[38] Andrzej Lingas. Voronoi diagrams with barriers and the shortest diagonal problem. *Inform. Process. Lett.*, 32:191–198, 1989.

[39] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* PhD thesis, Univ. California, Berkeley, CA, 1996.

[40] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in three dimensions. In *Proc. 8th ACM Symp. on Comput. Geom.*, pages 212–221, 1992.

[41] Atsuyuki Okabe, Barry Boots, Kōkichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: concepts and applications of Voronoĭ diagrams*. John Wiley & Sons Ltd., Chichester, 2nd edition, 2000.

[42] Jaak Peetre. Approximation of norms. *J. Approx. Theory*, 3(3):243–260, 1970.

[43] P. Peleg and J. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18:740–747, 1989.

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

[45] William H. Press, Saul A. Teukolsky, William T. Vettering, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 2nd edition, 1992.

[46] Helmut Ratschek and Jon Rokne. *Computer Methods for the Range of Functions*. John Wiley & Sons, New York, 1984.

[47] E. Schömer and C. Thiel. Efficient collision detection for moving polyhedra. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 51–60, 1995.

[48] M. Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. of Comput.*, 14(2):448–468, May 1985.

[49] Micha Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[50] F. Sloboda, B. Zat'ko, and P. Ferianc. Minimum perimeter polygon and its application. In R. Klette and W. G. Kropatsch, editors, *Theoretical Foundations of Computer Vision*, pages 59–70. Akademie-Verlag, Berlin, 1992.

[51] Lloyd Nicholas Trefethen and David Bau, III. *Numerical Linear Algebra*. SIAM, 1997.

[52] Li Zhang. *Kinetic Maintenance of Proximity Structures*. PhD thesis, Stanford University, 2000.

[53] Li Zhang, Harish Devarajan, Julien Basch, and Piotr Indyk. Probabilistic analysis for combinatorial functions of moving points. In *Proc. 13th Annual Symp. on Comput. Geom.*, pages 442–444, 1997.

[54] R. Zippel. *Effective Polynomial Computation*. Kluwer Academic Publishers, 1993.