IMPLICIT BOUNDING VOLUMES AND BOUNDING VOLUME HIERARCHIES

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

An Nguyen
September 2006

Abstract

Computing and maintaining proximity information between objects are crucial tasks in modeling and simulating the world around us, as in robotic motion planning, physics-based simulation, molecular dynamics, etc. The information is important as objects in real life do not normally penetrate and most of the interactions between objects happen when they are near each other.

Popular methods for answering proximity and collision queries use bounding volume hierarchies (BVHs). In these methods a bounding volume hierarchy is constructed for each object so that the object is captured in more and more details as one goes down the hierarchy. When answering a proximity query between two objects, their hierarchies are traversed from top down and the proximities of their bounding volumes are examined. Bounding volume hierarchies allow one to determine quickly if two objects are not in close proximity. The further apart the objects are, the less traversal the methods have to do and thus the less work in determining the proximity between the objects.

Current bounding volume hierarchy methods use explicit bounding volumes like spheres, axis aligned bounding boxes (AABB), oriented bounding boxes (OBB), k-discrete oriented polytopes (k-DOP), and rectangle swept spheres (RSS), etc. All these bounding volumes have simple geometry and are defined explicitly with respect to the ambient space containing the objects. While working well for rigid objects, existing bounding volume hierarchies do not work well when the objects deform substantially.

This dissertation presents results that show the power of implicit bounding volumes and implicit bounding volume hierarchies for proximity queries and collision detection. The first part of the dissertation studies the use of zonotopes (Minkowski sums of line segments) as bounding volumes. By defining bounding volumes implicitly through generating segments, complicated shapes can be captured tightly with a small number of segments. Efficient algorithms for computing exact and approximate optimal bounding volumes, as well as algorithms to detect interference between zonotopes are provided.

The second part of this dissertation studies a way to exploit prior structure of the underlying objects in constructing implicit bounding volume hierarchies that are stable when the objects undergo large deformations. More specifically, the linear structure of deforming necklaces is exploited to construct wrapped bounding sphere hierarchies attached to the objects. The bounding sphere hierarchies bound the necklaces tightly yet do not change often when the necklaces deform. The hierarchies offer the first sub-quadratic time self-collision detection for predefined hierarchies on necklaces.

The third part of this dissertation studies a geometric data structure called the discrete center hierarchy. The structure can be thought of as an implicit bounding volume hierarchy, and can be efficiently constructed and maintained under motion, giving the first lightweight maintainable bounding sphere hierarchy for a set of moving points. The structure naturally provides other maintainable proximity structures such as an $(1+\epsilon)$-well separated pair decomposition and an $(1+\epsilon)$-spanner for any $\epsilon > 0$, an approximate Voronoi diagram, and an approximate k-center structure.

Acknowledgements

I would like to thank my advisor Leonidas Guibas for his guidance and encouragement, his funding, and especially his patience in advising me. He has given me the freedom to to explore various research areas outside of this dissertation. This dissertation would not be possible without his help.

I would also like to thank my collaborators Leonidas Guibas, Pankaj Agawal, Li Zhang, Jie Gao, and Daniel Russel for the work related to this dissertation, and Mark Yim, Julien Basch, David Hsu, Niloy Mitra, Yusu Wang, Stefan Funke, Qing Fang, Primoz Skraba, and Nikola Milosavljevic for my work in other research areas. It is very enjoyable to work with these talented individuals.

Finally, I would like to thank my family for their encouragement and their mental support during my study. Special thanks to my wife for helping me deal with the many difficult moments I have been through.

Contents

List of Figures

Chapter 1

Introduction

Simulating objects around us to understand and to predict their motion and behavior is a major goal of many disciplines. With our growing desires to model complicated objects, to simulate physical laws governing our world accurately, and to interact with the simulations in real time, we need not only more computational power but also new efficient algorithms.

Physical objects often have the simple property that they do not overlap or interpenetrate. While objects in nature effortlessly bounce off each other upon contact, avoiding overlapping each other, virtual objects in our computations do not automatically do so. We instead have to spend great effort to detect collisions between virtual objects and handle the collisions when they happen. Collision detection is a bottleneck in many computations, including those in physical simulations, robotic motion planning, virtual surgery, etc.

A more general problem that is closely related to the collision detection problem is that of proximity query. Given a collection of objects, we are interested in finding all pairs of objects that are within some distance threshold from each other. The collision detection problem is a specific case of the proximity query problem when the distance threshold is zero. The proximity problem is of great interest because objects in nature often influence other objects near them. Take for example, in molecular dynamics simulations, forces asserted by atoms on each other are significant only when the atoms are close enough, within some threshold distance called the Lennard-Jones cut off distance. Forces between pairs of atoms further apart can be safely ignored to reduce the computation cost without affecting too much the quality of the simulations.

While being more general, the proximity query problem is essentially a collision detection problem, as checking whether two objects are within some distance threshold from one another is the same as checking appropriately fattened copies of the objects for collision. In this dissertation, we treat the two problems identically and focus mostly on the collision detection problem.

Collision detection is often done in two phases. In the *broad phase* when there are many objects in the scene, fast and simple heuristics are used to identify pairs of objects that potentially overlap. In this phase, objects are projected on one or more axes, and only pairs of objects having their projections in all axes overlap are selected for further collision checking [CLMP95, PML97]. In the *narrow phase*, pairs of objects identified in the broad phase are examined more closely to find out whether they indeed overlap. In this dissertation, we mainly concern with the narrow phase. Specifically, given two objects, we would like to check whether they intersect one another. When the objects are deformable, our primary interest is to check for self-collision, i.e. collision between different parts of the same object.

Virtual objects in simulations are often modeled as collections of primitive elements such as spheres, triangles, quadric patches, etc. As the primitive elements have simple shapes, the more complex the objects are, the more primitive elements that the objects contain. To verify that two virtual objects do not overlap, we could naively check each primitive element in one object against each primitive element in the other object. For self-collision, we could check for collision between each primitive element against all other primitive elements in the object. The cost of this naive approach to collision detection grows quadratically when object complexity increases, making collision detection prohibitively expensive for complex objects. Fast and efficient algorithms are thus desirable to handle the collision detection needs in large scale applications.

Typically two objects only collide with each other at a small number of locations, and most pairs of primitives from the two objects are far apart. We can attempt to identify quickly most pairs of elements that cannot possibly intersect and naively check remaining pairs of objects for collision.

1.1 Bounding volume hierarchies

A popular and effective way to filter out most non-colliding pairs of elements without spending a lot of computation is to use *bounding volume hierarchies*. A bounding volume hierarchy covering an object is simply a tree in which each node is associated with a bounding volume, see Figure 1.1. The bounding volume at the root node of the hierarchy covers the whole object, and the bounding volumes at the children of each node together cover the portion of the object that the bounding volume at that node covers. The collection of bounding volumes in each level of a bounding volume hierarchy covers the entire object, and each successive level of the hierarchy gives a tighter and tighter covering of the object.

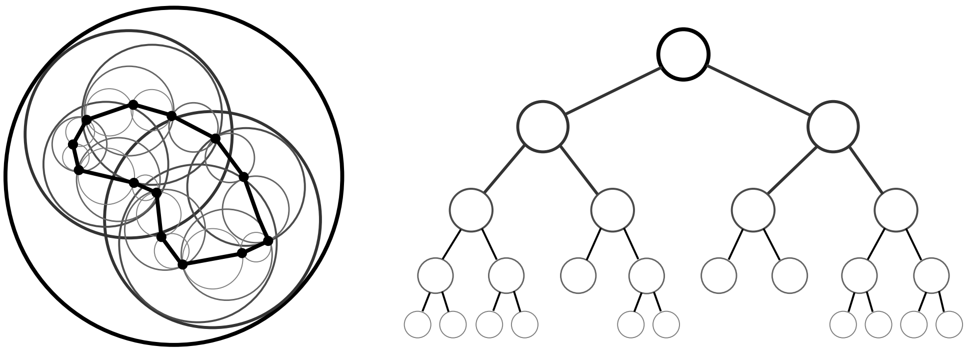

Figure 1.1: A bounding sphere hierarchy of a closed polyline. The top level sphere is a coarse approximation of the object, and the collection of spheres at the leaves of the hierarchy provides a much tighter approximation. In this example, the bounding volumes at the leaves of the hierarchy are diametral spheres, and each interior bounding volume is the smallest sphere containing the bounding volumes of their children.

Bounding volume hierarchies have been used in practice for a long time. Clark [Cla76] suggested to use bounding volume hierarchies as a way to represent virtual objects in a scene in a hierarchical fashion to render them quickly from any particular view point. This representation enables efficient clipping algorithm to identify parts of the scene that is not in the viewing window so that no further work is spent on rendering them. The representation also allows efficient sorting of objects in a scene according to their distance to the view point, enabling fast computation of the visible surface. Bounding volume hierarchies are also used to compute quickly the point where a light ray first intersects a scene, speeding up rendering in ray tracing [RW80, WHG84, KK86].

When used for collision detection, bounding volume hierarchies allow us to find out quickly many pairs of elements that cannot possibly intersect. Given two objects and their bounding volume hierarchies, if we check the bounding volumes at the root nodes of the two hierarchies for intersection and find that these two bounding volumes do not intersect, we can quickly conclude that all pairs of elements from the two objects do not intersect. If the top bounding volumes intersect, the objects potentially intersect. In that case, we can refine one of the two hierarchies, replacing the hierarchy by its subtrees rooted at the children of its original root node, effectively recursively check for collision between one object and components of the other object that are covered by the bounding volume hierarchies rooted at the children nodes, see Algorithm 1.

Algorithm 1 Detect collision between BVHs H_1 and H_2

```
1:  $B_1 \leftarrow \mathrm{root}(H_1)$
2:  $B_2 \leftarrow \mathrm{root}(H_2)$
3:  if $B_1$ does not intersects $B_2$ then
4:      RETURN FALSE
5:  else
6:      if $B_1$ is bigger than $B_2$ then
7:          SWAP $B_1$ and $B_2$ {Ensure $B_2$ is bigger than $B_1$}
8:      end if
9:      {Refine the big bounding volume hierarchy $B_2$}
10:     for all child $C$ of $B_2$ do
11:         if Detect collision between $C$ and $B_1$ then
12:             RETURN TRUE
13:         end if
14:     end for
15:     RETURN FALSE
16: end if
```

When the objects are far from each other and the bounding volumes tightly bound the objects, we often do not have to refine the hierarchies too many times before knowing for sure that the objects do not intersect. If the objects are overlapping or close to each other, we may have to traverse the hierarchies all the way to the leaf nodes where collisions between the objects may be detected. We then perform the naive collision checking between pairs of elements for those pairs of leaf nodes. The work involved in the collision detection is much less than the quadratic running time as in the naive collision checking algorithm for many practical cases.

While the concept behind bounding volume hierarchies is simple, there is a huge literature on the topic. Work on this topic often addresses one or more issues from the following major topics on bounding volume hierarchies: the choice of bounding volumes used in the hierarchies, the construction of bounding volume hierarchies, the update of the bounding volume hierarchies under motion (and more generally, deformation), and the exploitation of motion coherence in updating the bounding volume hierarchies. As the literature is so vast, and our approach is more theoretical in nature than most of the previous work, we do not attempt to give a complete survey of the topic but rather give a brief introduction to each of the topics in the following subsections. Interested readers are recommended to read [LG98, HEV$^+$04, LM04, TKZ$^+$04] for surveys.

1.1.1 Bounding volume shapes

One major issue in designing a bounding volume hierarchy for a particular application is selecting the shape of bounding volumes used in the hierarchy. A number of shapes have been used in practice: spheres [Qui94, Hub95, PG95], axis-aligned bounding boxes (AABB) [CLMP95, vdB97, Zac02], arbitrarily oriented bounding boxes (OBB) [GLM96], k-discrete oriented polytopes (k-DOPs) [KHM$^+$98], line and rectangle sweep spheres [LGLM00], convex hulls [EL01, OL03], quantized orientation slabs with primary orientation (QuOSPO) [He99], spherical shells [KGL$^+$98], cylinders [Bes03], etc., see Figure 1.2. Each shape has been shown to be effective for some particular scenarios.

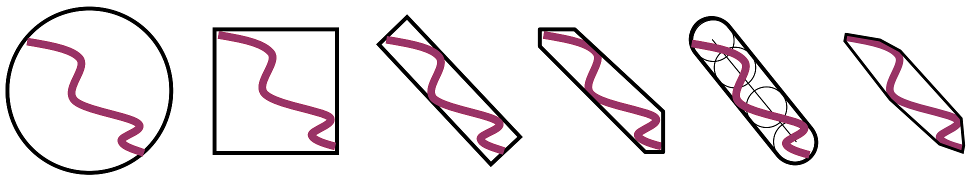

Figure 1.2: Popular bounding volumes. From left to right, the bounding volumes are sphere, axis aligned bounding box, oriented bounding box, k-DOP, line sweep sphere, and convex hull.

The effectiveness of using a given class of shapes in bounding volume hierarchies in general depends on the ability of that class of shapes to bound tightly objects in the application being considered and the efficiency of collision checking between two shapes in that class. On the one hand, the more tight the bounding volumes, the less we encounter false positives during collision detection – cases when two bounding volumes intersect but the corresponding parts of the objects covered by the bounding volumes do not. Tight bounding volumes also decrease the number of levels needed in a bounding volume hierarchy to approximate an object to a given error threshold, and thus reduce the cost of traversing the hierarchy during collision detection.

On the other hand, tight bounding volumes often have complicated shapes, leading to costly collision checking between the bounding volumes. To lower the cost of collision checking between the bounding volumes, simple shapes such as spheres are more favorable. The choice of a particular class of shapes in bounding volume hierarchies involves a complex trade off between the tightness and the collision complexity of that class of shapes.

1.1.2 Construction of bounding volume hierarchies

Once we have selected a particular shape for the bounding volumes, we can construct bounding volume hierarchies with bounding volumes of that shape. One way to do so, see [Qui94, GLM96], is to use a top-down divide and conquer algorithm. Given an object, we can split it into two or more roughly equal components, then recursively construct a bounding volume hierarchy for each of those components. The bounding volume of the root can then be chosen so that it covers tightly the entire object, or so that it covers all the bounding volumes of its children in the hierarchy. While computing a bounding volume covering only the object in the subtree gives a tighter bounding volume, computing a bounding volume covering all children bounding volumes is cheaper and allows interruptible collision detection in time-critical application [Hub96].

Bounding volume hierarchies can also be constructed bottom-up [BCG$^+$96]. Each element in the objects are first covered by a bounding volume, then nearby bounding volumes are merged together to create bounding volumes at higher levels until only one bounding volume is left.

Other approaches have been proposed to build bounding volume hierarchies, exploiting more the geometry of the objects to obtain tight bounding volumes. Tan et al. [TCL99] and Otaduy and Lin [OL03] use model simplification to merge leaf level bounding volumes from bottom up in the construction of bounding volume hierarchies of objects. Hubbard [Hub95] and Bradshaw and Sulivan [BO04] adaptively compute the medial axes of objects and use them to obtain good refinements of the spheres in a top down construction of bounding sphere hierarchies.

1.1.3 Motion and deformation

The objects are under motion and deformation in most applications involving collision detection. In these cases, before we can use bounding volume hierarchies for collision detection, we need to make sure that they are currently valid. The cost of updating the hierarchies may be significant, and thus we must take into account this cost in the collision detection.

If the objects move rigidly, the update of the bounding volume hierarchies is simple. When an object moves to a new location and orientation, we can simply transform bounding volumes in the bounding volume hierarchies of the object to the new location and orientation. Care must be taken, however, when the bounding volumes are aligned with the global frame of reference, as in the case of axis-aligned bounding boxes or k-DOPS. In these cases, the bounding volumes after an arbitrary rigid transformation are no longer aligned with the global frame. We can recompute the bounding volumes from scratch, which is expensive but gives tight bounding volumes, though if the rotation is small, we can quickly compute a new bounding volume aligning with the global frame by modifying the previous bounding volumes [CLMP95]. The realigning is either done by a local walking on the features of the object to obtain new tight bounding volumes, or by computing looser bounding volumes that cover the rotated version of the previous bounding volumes, see Figure 1.3.

An alternative method to global realignment of bounding volumes was proposed by Zachmann [Zac02]. In this work, the axis-aligned bounding boxes were not aligned to the new frame at each step to reduce the update cost. A more expensive collision test based on the separating plane [GLM96] must be used however to detect collision between aligned and non-aligned boxes.

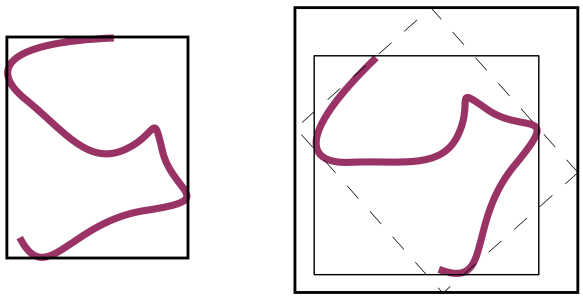

Figure 1.3: Updating axis aligned bounding box under rotation. When an object rotates, a new bounding box can be computed from the previous bounding box by a local walk. A looser bounding box can be computed more quickly as an axis-aligned bounding box of the rotated version of the previous bounding box.

Updating bounding volume hierarchies becomes more complicated when the objects are moving and deforming at the same time. In this case, both the tree structures and the bounding volumes of the hierarchies must be updated to ensure their effectiveness. The cost of updating is high, so only bounding volumes with simple shapes such as spheres [JP04], AABBs [vdB97], and k-DOPs [KHM$^+$98] are used.

The deformations considered in existing literature are limited. Klowsowski et al. [KHM$^+$98] recomputed and refit bounding volume hierarchies when objects undergo small deformation without providing any theoretical bound. Lotan et al. [LSHL02] proposed a new bounding volume hierarchies for kinematic chains, chains form by rigid links connecting to each other via rotational joints. They gave a method for updating the bounding volume hierarchies when one or more joints in the chains underwent a large angle change. The work was independent, yet closely related to our work in chapter 4. James and Pai [JP04] used *BDtree*, a bounding volume hierarchy inspired by the work in chapter 4 and showed that the bounding volumes in a BDtree can be conservatively updated so that the total cost of updating is sublinear.

Ganovelli et al. [GDO00] also proposed to maintain bounding volume hierarchies when objects deform. The hierarchies are based on oct-trees, and when the objects deform, the oct-trees are updated and bounding volumes in the hierarchies are updated to take into account changes in the oct-tree and in the positions of the elements. No theoretical performance for the scheme was provided, however.

1.1.4 Exploiting coherence

In most applications, collision detection has to be done repeatedly. As the displacement of the objects from one step to another step is small, the collision detection operation in two consecutive steps are often highly correlated. We can cache certain information from one step to speed up later steps. For example, asserting that two oriented bounding boxes (or convex hulls in general) are not overlapping involves a computation of a separating axis, a direction such that the projections of the boxes (or convex hulls) on that axis do not overlap. Caching such direction can help us to quickly verify whether the direction is still a separating axis for the two objects after a small motion, potentially saving us the cost of computing a new separating axis [GLM96]. Similarly, the bounding volume test tree [LGLM00], a structure encoding the collision interface between two bounding volume hierarchies, can be cached so that the collision checking in future time steps are more efficient, zooming directly to where the potential overlapping regions and thus saving time on hierarchy traversal.

We would like to point out that bounding volume hierarchies are only one of the several methods used to filter out pairs of non-intersecting pairs of elements. Various other ways have been proposed for this purpose. One simple way to do the filtering is to hash elements in the objects into regularly spacing boxes called voxels [LK02, THM$^+$03], observing that only pairs of elements that are in the same voxel could potentially intersect each other. Another way to reduce the number of pairs of primitives to check for collision is via visibility-based-overlap query using graphic hardware [GLM04, GRLM03]. The objects are rendered from different view directions, and pairs of objects whose images are disjoint are guaranteed to be disjoint. For self-collision detection of clothes, Volino and Thalmann [VT94] use surface curvature tests to rule out self intersections of certain patches on a smoothly discretized surface.

Just like bounding volume hierarchy methods for collision detection, most work on these methods focuses on practical aspects of collision detections and rarely provide any rigorous theoretical performance analysis. The collision detection algorithms used are simple tools to achieve fast enough running time for the simulations of interest. There is much less emphasis on the scalability and accuracy of the collision detection algorithms and on the worst case performance for arbitrary scenarios.

1.2 Implicit bounding volume hierarchies

It is well known that there is a wide gap between the theoretical and practical performance of collision detection using bounding volume hierarchies. Consider the collision detection between two polyhedra as in Figure 1.4. If we consider the faces of the polyhedra as elements and construct bounding volume hierarchies for the polyhedra with those elements as leaf nodes, as done in popular implementations such as SOLID [vdB97, vdB99] and RAPID [GLM96], the cost of the collision detection is quadratic in the number of faces. This is the case since the two polyhedra potentially intersect each other at quadratically many places, forcing the collision detection algorithms to visit quadratically many pairs of leaf nodes. There is little gain in using bounding volume hierarchies for collision detection in this worst case.

While there is a vast literature on practical use of bounding volume hierarchies for collision detection, there is surprisingly little work on bounding volume hierarchies from the theoretical side. Zhou and Suri et al. [ZS99] and Suri et al. [SHH98] showed that if the objects are fat, the number of overlapping pairs of objects are roughly proportional to the number of overlapping pairs of bounding boxes of the objects. This result justifies the usage of bounding boxes to cull away pairs of non-overlapping objects. Erickson [Eri05] studied a class of polyhedra that he called *local* polyhedra and showed that the intersection between two *local* polyhedra can be computed in $O(n \log n)$ time using hierarchies of axis-aligned bounding boxes.

This dissertation contributes to the understanding of bounding volume hierarchies from the theoretical side. We use theoretical analysis to analyze various aspects of the performance of collision detection using bounding volume hierarchies. We focus on implicit representations of bounding volume hierarchies and show the advantage of keeping the

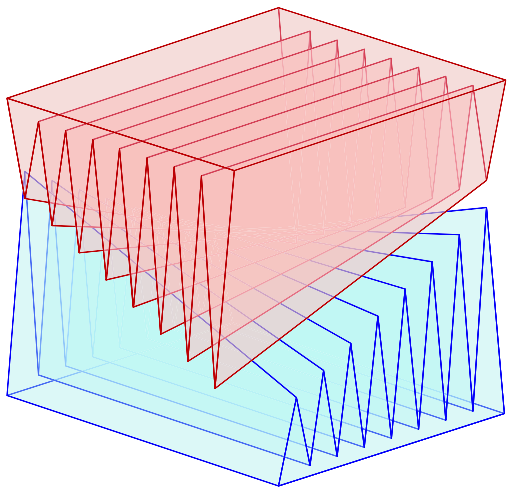

Figure 1.4: Chazelle's polyhedra from [Eri05]. Two polyhedra could potentially intersect each other at quadratically many places. Algorithms using bounding volume hierarchies will not help to reduce the quadratic cost of collision detection in this case.

bounding volumes or bounding volume hierarchies as combinatorial objects instead of representing them explicitly as geometric objects as in the previous works.

In particular, we show implicit representations of bounding volumes give us a better trade off between tightness and complexity. Instead of representing the bounding volumes using explicitly defined shapes such as spheres, AABBs, OBBs, etc., we consider using zonotopes, complex shapes with simple descriptions as Minkowski sums of line segments, as bounding volumes. Zonotopes have complicated explicit shapes allowing them to cover objects tightly. On the other hand, an implicit representation of zonotopes as collections of segments enables efficient algorithms operating on zonotopes.

We also show two different implicit representations of bounding volume hierarchies. In both representations, the bounding volume hierarchies are combinatorial objects attached to the objects and not to the ambient space as in the previous works. When the objects deform, even though the bounding volume hierarchies change continuously, the combinatorial structures representing them only change at discrete moments, at which time the structures can be updated. We rigorously analyze various qualities of the data structures we use and the efficiency of our algorithms in the worst case using asymptotic analysis. We show that implicit representations are more stable and they enable smooth update of the bounding volume hierarchies when the underlying objects deform.

1.3 Overview and summary of results

The dissertation is divided into 3 parts. In chapter 3, we propose to use zonotopes, complex polytopes that can be represented implicitly as Minkowski sums of line segments, as bounding volumes. As complex polytopes, zonotopes can be used to capture objects tightly, yet with their simple implicit representation as sets of line segments, they allow efficient algorithms to operate on them. We give algorithms to compute tight bounding zonotopes containing a set of points in 2D and in higher dimensions, and give an efficient algorithm to do collision detection in 3D. Given a set of points, we show an $O(n \log^2 n)$ algorithm to compute the minimum area enclosing zonotope in $\mathbb{R}^2$, and an algorithm to compute a $(1+\varepsilon)$-approximate minimum length zonotope in $\mathbb{R}^d$ in time $O(n\epsilon^{-(d-1)^2} + \epsilon^{-O(d^2)})$. We provide an algorithm for detecting collision between two implicitly represented zonotopes in $\mathbb{R}^3$ in time $O(n \log^2 n)$.

We also show that zonotopes can be used to facilitate collision detection in space time domain, a technique to ensure that the collisions of objects are always detected during their continuous motion.

In chapters 4 and 5, we address the issue of updating the hierarchy during motion. In chapter 4, we consider bounding sphere hierarchy on deformable necklaces, linear structures such as ropes, strings, muscles, macro-molecules, etc. We exploit this linear property to define implicit bounding volume hierarchies on necklaces and show how we can maintain them when the necklaces deform. We prove that self collision detection of the necklaces using our implicit bounding volume hierarchies can be done in subquadratic time, the first subquadratic time bound for self collision checking using predefined hierarchies.

In chapter 5, we define data structures called *discrete center hierarchies* on point sets. We provide algorithms to construct the structures and algorithms to update the structures when points are added or removed. When the points move, assuming that the points move coherently (which we will define precisely later) we provide algorithms to update the structure, fully exploiting the coherence in the motion of the points. We analyze our algorithm in the kinetic data structure framework, a theoretical framework for algorithms dealing with motion, and show that our algorithm has all the properties of a nearly optimal kinetic data structure.

The maintainability of the discrete center hierarchies under motion also implies the maintainability of a number of other proximity structures of interest in computational geometry. We show how to maintain an $(1+\epsilon)$-well separated pair decomposition and an $(1+\epsilon)$-spanner for any $\epsilon > 0$, an approximate Voronoi diagram, an approximate k-center data structure under motion, and we analyze the performance of the maintenance for those structures.

We provide background material in chapter 2, and conclude in chapter 6.

1.4 References

The results in chapter 3 were published in the proceedings of Symposium on Discrete Algorithms (SODA) in 2003 with title “Zonotopes as Bounding Volumes”, pages 803–812. The work was coauthored by Leonidas Guibas and Li Zhang.

The results in chapter 4 were published in Symposium on Computational Geometry (SoCG) in 2002 with title “Collision detection for deforming necklaces”, pages 33–42. It was republished in “Computational Geometry: Theory and Applications”, volume 28(2-3), pages 137–163, in 2004. The work was coauthored by Pankaj Agarwal, Leonidas Guibas, Daniel Russel, and Li Zhang.

Parts of the material in chapter 5 have been published in “Symposium on Computational Geometry (SoCG)” in 2004 with title “Deformable Spanners and their Applications”, pages 179–199 and were republished in “Computational Geometry: Theory and Application”, volume 35, pages 2–19, in 2006. This was a joint work with Jie Gao and Leonidas Guibas.

Chapter 2

Background

In this chapter we present background material for the rest of the dissertation.

2.1 Points and spheres as primitives

In this dissertation, the primitives we consider are points and spheres to simplify basic geometric calculations so that we can focus on the combinatorial issues that form our main interest. This choice of primitives is natural when the objects themselves are collections of points or spheres. In many other cases, even when the primitives are not points or spheres, we can frequently formulate the collision detection problems involved as being on points or spheres. For example, when the objects are polytopes, bounding volumes for a collection of polytopes could be thought of as bounding volumes of the vertices of the polytopes. As another example, in simulations using finite element methods [Red93] objects are often partitioned into small tetrahedra, each can be approximated by a sphere. There is a literature on using spheres in engineering modeling [DB00] and for biomolecules spheres are obviously the right choice.

We can also use the technique in [Qui94] to approximate the objects (or their surface area) with collections of spheres, see Figure 2.1. The smaller the size of the spheres, the more accurate the collections approximate the objects. The collision detection between objects can then be approximated as a proximity query between the centers of the spheres in the sphere collections.

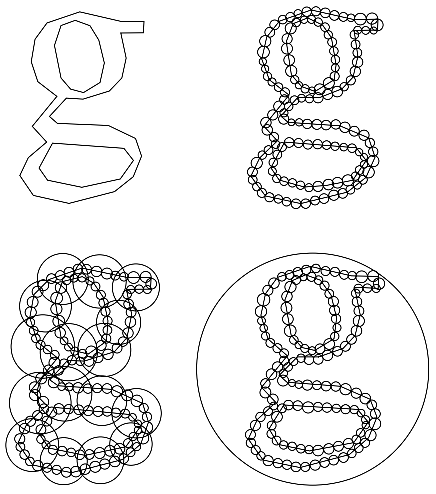

Figure 2.1: A covering using spheres of the surface of an objects and a bounding volume hierarchy of that covering, in [Qui94]

.

2.2 KDS motion model

There has been increased interest recently in modeling time-varying phenomena and this has naturally led to the study of geometric objects under motion. Certain assumptions on the motion of the underlying objects are necessary so that algorithms for moving objects can be analyzed.

Early work on moving objects in computational geometry, initiated by Atallah [Ata85], focused on bounding the number of combinatorial changes in geometric structures as the objects move along prescribed trajectories [AS00]. In the late 1980s algorithms were developed for detecting and computing the actual changes in geometric structures as objects undergo motion, e.g. [ST95, AGMR98]. However, these results assume an off-line model of motion in which the trajectories are given in advance, which is a relatively uncommon situation.

More recently, the *Kinetic Data Structures Framework* (or KDS for short) [Gui98, BGH99] was proposed. In this framework, each object has a known flight plan, the trajectory that each follows. However, the objects are allowed to change the flight plan when necessary due to interactions with other objects or with the environment.

KDS works by maintaining a proof over time that a combinatorial structure has certain desired properties, for example the current data structure is a valid bounding volume hierarchy of an object, or that a given pair of points is the nearest pair of points. Each proof, called *certificate*, is often an assertion about certain relationship between the underlying objects. An *event* happens when a certificate fails, making the proof invalid. At each event, we need to update the KDS, changing the combinatorial structure if necessary, and find a new proof that the structure continues to have the desired properties.

An event that leads to a change in the combinatorial structure (say the pair of nearest point change) is called an *internal event*. An event that requires changes in the proof without changing the data structure is called an *external event*. Note that an external event is considered non-essential, as there are potentially some other KDS's having the same desired properties that does not have an event at that time.

As the trajectories of the objects are known precisely, the certificate failure time can be predicted. Between certificate failure times, all the proofs remain valid, and thus the desired property still holds. As the result, the KDS changes only at those certificate failure time even though the objects move continuously.

A KDS maintains an event queue containing the failure times of its current certificates. When the earliest event happens, the KDS must invoke some repair mechanism to generate a new set of proofs and associated certificates for the desired property that we wish to maintain.

The quality of a KDS is measured by the following 4 properties:

- *Compactness*. The compactness of KDS measures the size of the KDS in the worst case. We call a KDS compact if the number of certificates in the KDS is at most linear or near linear to the number of objects, i.e. when there are N objects, the KDS has at most $O(N)$ or $O(N \operatorname{polylog} N)$ certificates.

- *Locality*. The locality of a KDS measures the number of certificates in which any object participates in the worst case. We call a KDS local if any object is involved in at most $O(\operatorname{polylog} N)$ certificates. When a KDS is local, a flight change involves the update of a small number of certificate failure times.

- *Responsiveness*. The responsiveness of a KDS measures how fast the KDS can be updated in the worst case when a certificate fails. A KDS is called responsive if the

KDS can be updated in time $O(\mathrm{polylog}\, N)$.

- *Efficiency*. The efficiency of a KDS measures the ratio of the number of external events over the number of internal events. A KDS is called efficient if this ratio is at most $O(\mathrm{polylog}\, N)$ in the worst case, i.e. the KDS does not spend too much time handling non-essential events.

There are KDSs to maintain various geometric structures such as convex hulls, nearest pairs [BGH97], Delaunay triangulations [BGSZ97], multi-dimensional range search trees [BGZ97], etc. One of the key results in this dissertation is a KDS for bounding volume hierarchies.

2.3 Black box motion model

Though the KDS view has provided an elegant framework for the analysis of motion algorithms, certain requirements limit its applicability, especially the need to predict certificate failure times. In many real-world situations where a physical system is evolving according to an ordinary or partial differential equation, the motion plans of the simulation elements are not known in closed form and thus cannot be modeled properly in the KDS framework.

We consider a more practical motion model that we call *black box motion model*. In this model, we assume that at certain times (the time steps of the simulation) an oracle moves the elements according to the some underlying physic laws and reports their new position back to us. We assume that the time steps chosen by the simulator are such that at each step the motion of the elements is small when compared to the overall scale of the simulation. Unlike the kinetic data structure setting where we have explicit motion paths and can predict certificate failures, here we are called upon to repair a geometric structure after small displacements of its defining elements.

Maintaining geometric structures under black box motion model is significantly more difficult compare to those under the KDS motion model. For example, maintaining a Delaunay triangulation under the black box motion model is quite complicated with no performance guarantees whatsoever [GR04], even though the same data structure can be maintained easily and cheaply in KDS setting.

We note that the repair mechanism for all existing KDS relies on a strong assumption that the repair is done when the *first* certificate fails. It is not known whether any existing KDS can be repaired when there are multiple certificate failures, or even when there is only one failed certificate but this certificate is not guaranteed to be the first certificate that fails since the last repair, situations that frequently occure in black box motion model.

In this dissertation, we show that our bounding volume hierarchy can be provably updated in the black box motion model, making it the first data structure that is maintainable in this more practical setting.

Chapter 3

Zonotopes as Implicit Bounding Volumes

3.1 Introduction

Zonotopes have long been studied in combinatorial geometry, polyhedral combinatorics, algebraic geometry, and other parts of mathematics. Yet, except for their use in solving systems of polynomial equations [HS95], their usefulness in applications of interest to science and engineering has been limited. In this chapter we propose to develop the use of zonotopes (and especially zonogons and zonohedra, the $\mathbb{R}^2$ and $\mathbb{R}^3$ cases) as versatile bounding volumes for pieces of underlying geometry in modeling applications. Unlike other bounding volumes proposed in previous works, zonotopes are complicated objects, yet they have simple implicit representations, and efficient algorithms are available to operate directly on these implicit representations.

A zonotope Z is defined by line segment generators $s_1, s_2, \ldots, s_n$ in $\mathbb{R}^d$. The zonotope is simply the Minkowski sum $s_1 \oplus s_2 \oplus \cdots \oplus s_n$ of its line segment generators. Equivalently, a zonotope is simply an affine image of the unit cube from $\mathbb{R}^n$ to $\mathbb{R}^d$. Well known zonotopes are parallelograms and regular hexagons in $\mathbb{R}^2$, and parallelepipeds, dodecahedrons, and truncated octahedrons in $\mathbb{R}^3$, see Figure 3.1.

It is obviously a convex polytope and its facets are parallelepipeds defined by $(d-1)$-tuples of its generators. Note that a zonotope must be a centrally symmetric convex polytope. While in 2-D any centrally symmetric convex polygon is a zonogon, that is no longer the case in 3-D or higher dimensions. However, many familiar polyhedra are zonotopes,

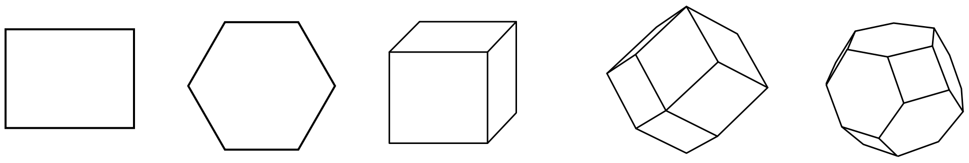

Figure 3.1: Rectangles, regular hexagons, cubes, dodecahedrons, and truncated octahedrons are all zonotopes. They have 2, 3, 3, 4, and 5 segment generators respectively.

including cubes and parallelepipeds, truncated octahedra, and rhombic dodecahedra. In 3-D the facets of a zonotopes are parallelograms defined by pairs of segment generators. The collection of all those facets sharing a particular segment generator form a band (zone) wrapping around the zonotope — a fact which justifies the name *zonotope*, see Figure 3.2

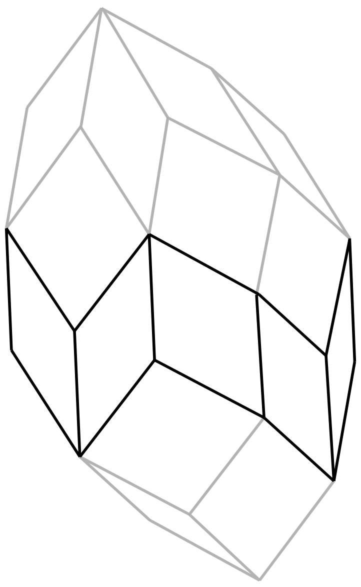

Figure 3.2: A zonotope and its band corresponding to the vertical segment.

Since a zonotope is always centrally symmetric, every facet has an opposite congruent facet on the other side. The combinatorics of the faces of a zonotope are equivalent to those of the vertices in an arrangement of great hypercircles in a sphere of one less dimension. This can be most easily seen by considering the space of d-dimensional hyperplanes tangent to the zonotope. The space of all their d-dimensional normal unit vectors can be seen as a unit sphere, equivalent to an oriented projective $(d-1)$-space. For any given unit vector, there is a unique hyperplane normal to the vector and tangent to the zonotope at some face. Under this map each generator gives rise to a great circle; thus the facets of the zonotope are in 1-1 correspondence with the vertices of this spherical arrangement under this tangent space map. Note that all facets belonging to a zone, map to the vertices on the great circle defined by their shared generator.

Spherical arrangements can be mapped to hyperplane arrangements by a simple projective map. Because of these two correspondences, we can both estimate the size and construct zonotopes by using the corresponding classical bounds for hyperplane arrangements. In particular, zonotopes in $\mathbb{R}^d$ have complexity (including number of facets) that is $O(n^{d-1})$, and can be constructed within the same time bound. In particular, a zonohedron has complexity $O(n^2)$. If the generating segments are in general positions, i.e. no d segments lie on any hyperplane in $\mathbb{R}^d$, the complexity of the zonotopes is precisely $\Theta(n^{d-1})$.

Several facts make zonotopes an intriguing possibility as a bounding volume:

- Zonotopes are closed under Minkowski sum and difference; this implies that testing for intersection between two zonotopes can be implemented by testing for point inclusion in their Minkowski difference.

- The list of generators is an efficient *implicit* representation of the zonotope; for example, in $\mathbb{R}^3$, a zonotope of size $O(n^2)$ can represented by only n generators. Furthermore, operations such as Minkowski sum and difference are trivial to express in terms of generator lists.

- Zonotopes allow for bounding volumes of *variable complexity*, within a unified framework. For example, when constructing a bounding volume hierarchy, one can use zonotopes with more generators at higher levels in the hierarchy, where there are few hierarchy nodes but the complexity of the enclosed geometry might lead to a bad fit using only few generators. As we will show, mixing space and space-time volumes [Cam90, Hub93, Hub95] is another example.

3.2 Summary of the results

In this chapter we present efficient algorithms for finding tight zonotopes enclosing some underlying polyhedral geometry and for using zonotopes as bounding volumes in collision detection applications. Throughout, we aim to represent zonotopes via their implicit representation as collections of generators, and not as explicit polytopes.

Specifically:

- We give an $O(n \log^2 n)$ algorithm for computing the minimum area zonotope enclosing a set of n points in $\mathbb{R}^2$.
- We give an algorithm to find a zonotope enclosing n given points in $\mathbb{R}^d$ with generators along k given directions and minimizing the sum of the generator lengths, in time $O(nk^{d-1} + k^{O(d)})$.
- We give an algorithm to find a zonotope enclosing n given points in $\mathbb{R}^d$ whose total generator length is within $(1 + \epsilon)$ of the optimum, in time $O(n(\epsilon/d)^{-(d-1)^2} + (\epsilon/d)^{-O(d^2)})$.

Furthermore, given zonotopes in $\mathbb{R}^3$ specified by their generators, we can:

- Decide whether two zonotopes with n generators in total intersect or not, in time $O(n \log^2 n)$ time.
- When repeated intersection testing is required, as in physical simulations, we describe how to implement efficiently some of the classical methods (such bounding volume hierarchies, or tracking closest feature pairs) using only the implicit description of zonotopes via their generators.
- We show how to easily build bounding *space-time volumes* for zonotopes, for use in collision detection applications where it is critical that no collisions be missed.

3.3 Zonotope fundamentals

Formally, a zonotope is a Minkowski sum of a finite set of line segments. An alternative view is to define a zonotope by its center and generator vectors. The zonotope Z centered

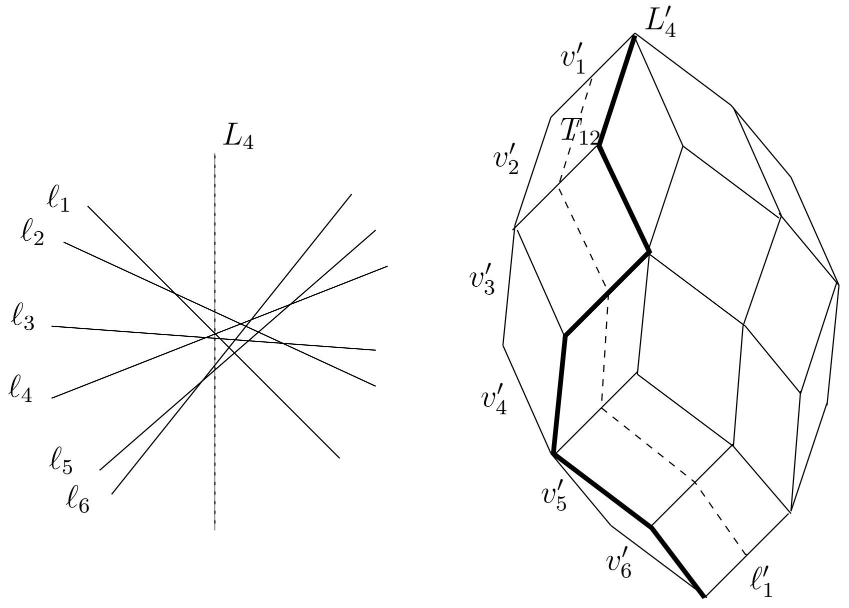

Figure 3.3: The duality between the line arrangement Π and the tiling $\mathcal{T}$.

at p with generators $v_1, v_2, \cdots, v_n$ is the point set $\{p + \sum_i \delta_i v_i \mid -1 \leq \delta_i \leq 1$ for each $1 \leq i \leq n\}$. We write $Z = (p, \langle v_1, v_2, \cdots, v_n \rangle)$. For simplicity, we assume throughout the paper that the zonotopes are non-degenerate, i.e. any d generators are linearly independent. In d-dimensions, a zonotope with n generators has complexity $O(n^{d-1})$. As we remarked, the topology of the boundary of a zonotope matches the topology of a hyperplane arrangement in any dimension [Zie94]. We describe this duality in three dimensions.

In three dimensions, the faces on a zonotopes are parallelograms. Let us take the z-axis as pointing up. The boundary of a zonotope can be decomposed into two pieces: the upper hull Z^+ and the lower hull Z^-. We project Z^+ on the xy plane and obtain the tiling $\mathcal{T}$ of a convex polygon Q, the projection of the vertical silhouette of Z^+. Each tile T_{ij} in $\mathcal{T}$ is the projection of a face on Z^+ and is a translation of parallelogram $\{xv'_i + yv'_j \mid -1 \leq x, y \leq 1\}$ where v'_i denotes the projection of v_i on the xy-plane.

Now consider the plane $P = \{z \mid z = 1\}$. For each generator v_i of Z, we draw a plane P_i passing through the origin and perpendicular to v_i. Let ℓ_i be the line $P_i \cap P$. Denote by Π the arrangement of $\{\ell_i \mid 1 \leq i \leq n\}$. The dual diagram of $\mathcal{T}$ is isomorphic to the line arrangement Π: each parallelogram T_{ij} in $\mathcal{T}$ corresponds to a vertex between ℓ_i and ℓ_j; each vertex in $\mathcal{T}$ to a face in Π; and each edge in $\mathcal{T}$ to an edge in Π (Figure 3.3).

In the later sections, we will exploit this duality between line arrangements and zonotopes to design efficient algorithms for zonotope intersection testing.

3.4 Smallest enclosing zonotopes

In this section we focus on algorithms for computing smallest enclosing zonotopes for some underlying geometry in $\mathbb{R}^d$. Many possible definitions of 'smallest' can be used, including volume, surface area, total length of generators, etc. A first basic observation is that it suffices to consider only the convex hull of the underlying geometry, since a zonotope is convex. Since we are only concerned with polyhedral geometry, from now on we will focus on computing the optimal enclosing zonotope of a set of points, which we may assume to be the vertices of a given convex polytope. We consider the cases $d = 2$ and $d \geq 3$ separately.

3.4.1 Minimum area enclosing zonotope in $\mathbb{R}^2$

In $\mathbb{R}^2$ we can give a fast algorithm to compute the minimum area enclosing zonotope. Our input can be assumed to be a convex polygon $\mathcal{H}$ of n vertices $P_i, 1 \leq i \leq n$, in $\mathbb{R}^2$. We show that the smallest area zonotope $\mathcal{Z}$ that contains all the points P_i can be computed in $O(n \log^2 n)$ time.

We first look at a much simpler problem, when the center O of the zonotope is specified. In $\mathbb{R}^2$, a zonotope is simply a centrally symmetric polygon, or *zonogon*. If the zonogon has a center at O and contains all the points P_i, it must also contain all the reflection points P'_i of P_i through O, and thus it contains the convex hull of the set of $2n$ points P_i and P'_i. This convex hull is centrally symmetric around O, and so it is the minimum area zonogon centered at O and containing all the points P_i. We call this zonogon $\mathcal{Z}(O)$.

In the general setting, only the points P_i are given. We need to find the center O that minimize the area of $\mathcal{Z}(O)$. For notational simplicity, we will allow indices outside the range $[1 \mathinner{.\,.} n]$ and identify P_i with P_{i+n} and P_{i-n} for each i.

For a given center O, the vertices of $\mathcal{Z}(O)$ are either original points P_i or reflected points P'_i. By grouping the original points together, and respectively, the reflected points, we can describe $\mathcal{Z}(O)$ as a circular sequence of vertices in a counterclockwise order of the form: $\mathcal{S} = P_{a_1}...P_{b_1}P'_{a'_1}...P'_{b'_1} \cdots P_{a_k}...P_{b_k}P'_{a'_k}...P'_{b'_k}$. We call this sequence the *combinatorial description* of $\mathcal{Z}(O)$.

We denote by $[\mathcal{P}]$ the area of a polygon $\mathcal{P}$, and define the function $f : \mathbb{R}^2 \to \mathbb{R}$, $f(O) = [\mathcal{Z}(O)]$, for each point $O \in \mathbb{R}^2$. Note that given $\mathcal{H}$ and O, we can construct $\mathcal{Z}(O)$ in $O(n)$ time, and thus f can be evaluated at any point $O \in \mathbb{R}^2$ in $O(n)$ time. To find the global minimum of f, we first establish some properties of f.

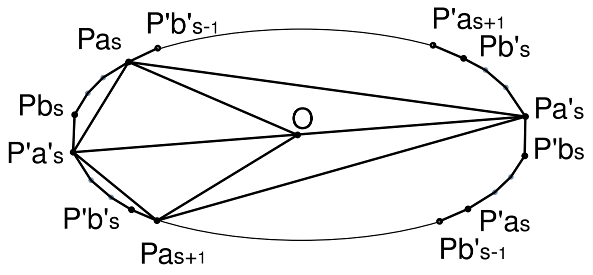

Figure 3.4: Combinatorial description of a zonogon

In a region where the combinatorial description of $\mathcal{Z}(O)$ is a constant $\mathcal{S}$, exploiting the symmetry of $\mathcal{Z}(O)$, we have that (see Figure 3.4):

$$\begin{aligned} f(O) &= \sum_{s=1}^{k} \Big([OP_{a_s}P'_{a'_s}P_{a_{s+1}}] + \\ &\quad [P_{a_s}...P_{b_s}P'_{a'_s}] + [P'_{a'_s}...P'_{b'_s}P_{a_{s+1}}] \Big) \\ &= \sum_{s=1}^{k} \Big([P_{a'_s}P_{a_s}OP_{a_{s+1}}] + 2\,[P_{a_s}...P_{b_s}P'_{a'_s}] \Big) \\ &= \sum_{s=1}^{k} \Big([P_{a'_s}P_{a_s}P_{a_{s+1}}] - [OP_{a_s}P_{a_{s+1}}] \\ &\quad +4\,[P_{a_s}...P_{b_s}O] - 2\,[P_{a_s}...P_{b_s}P_{a'_s}] \Big) \\ &= \sum_{s=1}^{k} \Big([P_{a'_s}P_{a_s}P_{a_{s+1}}] + 4\,[P_{a_s}...P_{b_s}O] \\ &\quad -2\,[P_{a_s}...P_{b_s}P_{a'_s}] \Big) - [P_{a_1}...P_{a_k}]\,. \end{aligned}$$

It follows from the above equation, in each region where $\mathcal{S}$ is a constant, f is an affine function of O. The coefficient of the linear term of that affine function comes from the term $4\,[P_{a_s}...P_{b_s}O]$ and thus depends only on the edges of $\mathcal{H}$ appearing on $\mathcal{Z}(O)$.

For an edge P_tP_{t+1} of $\mathcal{H}$, let s be such that P_s be the point furthest from P_tP_{t+1} among all the points P_i. Let ℓ_t be the directed line connecting the midpoint of P_sP_t to the midpoint of P_sP_{t+1}.

It is easy to see that the edge P_tP_{t+1} of $\mathcal{H}$ is an edge of $\mathcal{Z}(O)$ iff all the points P'_i are on the left side of the directed line P_tP_{t+1}, i.e. iff P'_s is on the left side of P_tP_{t+1}, and thus, iff O is on the left side of ℓ_t. If $\mathcal{A}$ denotes the arrangement of the lines ℓ_t, then it is clear that f is a piecewise affine function over $\mathcal{A}$. We have thus shown:

Lemma 3.4.1.

The area function f is piecewise affine, and its domain decomposition is given by an arrangement of n lines.

Given a line ℓ, let us consider the restriction f_ℓ of f onto ℓ. It is clear that f_ℓ is a piecewise affine function as well. We can compute the intersections of ℓ with the lines ℓ_t, and sort these intersections in $O(n \log n)$ time. After that, when O moves along ℓ, we can track the edges of $\mathcal{H}$ appearing on or disappearing from $\mathcal{Z}(O)$. This way, we can easily compute the function f_ℓ in $O(n)$ additional time. Thus,

Lemma 3.4.2.

The restriction of f onto any line ℓ can be computed in $O(n \log n)$ time.

By rotating the plane, we can assume, without lost of generality, that the line ℓ is a vertical line. We further assume that the line ℓ and the points P_i are in general position, and thus ℓ is not parallel to any of the edges in $\mathcal{H}$. When the point O is at $-\infty$ along ℓ, the edges in $\mathcal{H}$ appearing on $\mathcal{Z}(O)$ are edges on the upper hull of $\mathcal{H}$. When O moves upward and crosses one of the line ℓ_t, if that line corresponds with an edge in the upper hull of $\mathcal{H}$, that edge disappears from $\mathcal{Z}(O)$. If the line ℓ_t corresponds with an edge in the lower hull of $\mathcal{H}$, that edge appears on $\mathcal{Z}(O)$. Observe that in both cases, the slope of f_ℓ increases. As the result, the slope of f_ℓ is monotonically increasing. The point O_m where the slope changes sign from negative to positive must be unique and is the only minimum of f_ℓ. Thus,

Lemma 3.4.3.

The restriction f_ℓ of f onto any line ℓ is a unimodal function, i.e. a function with only one local minimum.

Corollary 3.4.4. *The function f is unimodal, i.e. it has only one local minimum.*

Proof: If f has at least two local minima, let ℓ is a line passing through at least two minima of f. The restriction of f on ℓ has at least two local minima, a contradiction to the previous lemma. $\square$

Let ℓ be a vertical line, and let O_ℓ be the point where f_ℓ achieves its minimum. From Lemma 3.4.2, O_ℓ can be computed in $O(n \log n)$ time. If we compute the linear coefficient of f at O_ℓ, we can decide whether O_ℓ is the global minimum of f, and if not, using the unimodal property of f, we can tell whether the global minimum must be on the left or the right of ℓ.

As f is piecewise affine on $\mathcal{A}$, it has a global minimum at one of the vertices of $\mathcal{A}$. We can use binary search, with the help of the slope selection algorithm of [CSSS89], to locate that vertex. There are $O(n^2)$ candidate vertices at the beginning. In each search step, we reduce the number of candidate vertices in half, by first running the slope selection algorithm to obtain a vertical line separating the set of candidates into 2 subsets of equal size, then deciding which of the subsets contains the global minimum. There are $O(\log n)$ search steps, each costs $O(n \log n)$ time for running the slope selection algorithm to obtain a vertical line, and another $O(n \log n)$ time to decide what side of that vertical line the optimal vertex is on. The total cost of locating the global minimum of f is $O(n \log^2 n)$. Thus,

Theorem 3.4.5. *The minimum area zonotope containing a set of n points in $\mathbb{R}^2$ can be computed in $O(n \log^2 n)$.*

In $\mathbb{R}^d$, $d \geq 3$, the computation of the optimal enclosing zonotope for a point set is more complicated, because not every centrally symmetric polyhedral body is a zonotope. In fact, most centrally symmetric bodies cannot be approximated arbitrarily closely by zonotopes — those that can are know as *zonoids*. Unlike in $\mathbb{R}^2$, we know of no simple way to find an optimal enclosing zonotope even when the zonotope center is given. The volume of a zonotopes in $\mathbb{R}^d$ is also more complicated, being the sum of the volumes of all possible parallelepiped formed by d-tuples of the generators. We consider instead for $d \geq 3$ two simpler problems. In both problems, we use the sum of the total length of the generators as the measure of optimality. In the first problem, we consider the case where the directions of the generators are given, and in the second problem, we consider the task of finding an approximately optimal enclosing zonotope.

3.4.2 Discrete oriented enclosing zonotope

Given a set of points $P = \{p_1, p_2, ..., p_n\}$, and a set of unit vectors $V = \{v_1, v_2, ..., v_k\}$, we would like to compute a point p and coefficients $c_1, c_2, ..., c_k$ such that the discrete oriented zonotope $\mathcal{Z}_d = (p, \langle c_1v_1, c_2v_2, ..., c_kv_k\rangle)$ contains all points in P.

What makes this problem easier is the fact that the combinatorial structure of a zonotope depends on the direction of its generating vectors, and not on the their length, and thus the combinatorial structure of all discrete oriented zonotopes with the same direction vector set are the same. We can compute the hyperplane arrangement dual to $\mathcal{Z}_d$ in $O(k^{d-1})$ time and then, for any $(d-1)$-tuple of directions, we can find the normal direction of the two zonotope faces corresponding to that tuple, and find the two extreme points among P along that direction. It is clear that we only need to look at these extreme points when computing $\mathcal{Z}_d$.

By rearranging the points if necessary, we can assume without lost of generality that the first n' points $p_1, p_2, ..., p_{n'}$ are the extreme points, $n' = O(k^{d-1})$.

To compute the coefficients c_i's, we solve the following linear program:

$$\begin{aligned}
&min \sum_{i=1}^{k} c_i, \\
&\text{subject to:} \\
&p_j = p + \sum_{i=1}^{k} b_{ij}v_i, \; \forall j, \; 1 \leq j \leq n', \text{ and} \\
&-c_i \leq b_{ij} \leq c_i, \; \forall i, j, \; 1 \leq i \leq k, 1 \leq j \leq n'.
\end{aligned}$$

We can solve this linear program in polynomial time using interior point methods [Wri92]. There are $O(k^d)$ equations and constraints, and therefore the cost of solving this linear programming is $O(k^{O(d)})$. Thus,

Theorem 3.4.6. *The minimum total length discrete oriented zonotope $\mathcal{Z}_d$ having generators along k given directions and containing a given set of n points can be computed in $O(nk^{d-1} + k^{O(d)})$ time.*

3.4.3 Approximate minimum total length zonotopes

In this subsection, we would like to compute a zonotope $\mathcal{Z}_a$ containing the set of points $P = \{p_1, p_2, ..., p_n\}$ such that the total length of the generators of $\mathcal{Z}_a$ is within a $O(1+\epsilon)$ factor of the the total length of the minimum total length zonotope $\mathcal{Z} = (p, \langle w_1, w_2, ..., w_r\rangle)$ enclosing P.

We consider the set of unit vectors $S = \{v_1, v_2, ..., v_k\}$ that tessellates the sphere of directions so that for any unit vector u, there is a vector v_j in S such that $|u - v_j| < \epsilon/d$, i.e. any unit vector can be closely approximated by a vector in S. It is intuitively clear that we can do so with $O((\epsilon/d)^{-(d-1)})$ vectors. Let $e_1, e_2, ..., e_d$ be the unit vectors along the axes of some coordinate system.

For each generator vector w_i of $\mathcal{Z}$, let $v_{j(i)}$ be a vector in S that closely approximates $w_i/|w_i|$, i.e. $\left|v_{j(i)} - w_i/|w_i|\right| < \epsilon/d$. From this and the fact that the unit cube contains the unit sphere, $w_i/|w_i| \subset v_{j(i)} \oplus_{i=1}^{d} ((\epsilon/d)\cdot e_d)$. Thus

$$\begin{aligned}
\mathcal{Z} &= p \oplus \oplus_{i=1}^{r} w_i \\
&\subset p \oplus \oplus_{i=1}^{r} \big(|w_i| \cdot (v_{j(i)} \oplus_{i=1}^{d} ((\epsilon/d)\cdot e_d))\big) \\
&= p \oplus \oplus_{i=1}^{r} (|w_i| \cdot v_{j(i)}) \oplus_{i=1}^{d} \Big(\big((\epsilon/d)\sum_{i=1}^{d} |w_i|\big)\cdot e_i\Big).
\end{aligned}$$

Let $R = S \cup \{e_1, e_2, ..., e_d\}$, then $\mathcal{Z}$ is contained inside a discrete oriented zonotope with direction vector set R. We call this zonotope $\mathcal{Z}'$. Let $D = \sum_{i=1}^{d} |w_i|$, the total length of generators in Z, then the total length of generators in $\mathcal{Z}'$ is at most $D + \epsilon D = (1+\epsilon)D$.

We compute $\mathcal{Z}_a$ with direction vector set R having the minimum total length. The total length of generating vectors of $\mathcal{Z}_a$ is less than the total length of $\mathcal{Z}'$, and thus is within $1+\epsilon$ the total length of $\mathcal{Z}$. Thus,

Theorem 3.4.7. *Given a set of n points in $\mathbb{R}^d$, and an $\epsilon > 0$. An approximate enclosing zonotope of the point set with total length of generators within $1 + O(\epsilon)$ of the optimal one can be computed in $O(n(\epsilon/d)^{-(d-1)^2} + (\epsilon/d)^{-O(d^2)})$.*

3.5 Collision detection between two zonotopes

We now describe algorithms for testing if two zonotopes intersect in $\mathbb{R}^3$. We consider two scenarios. In the first scenario, we only need to detect the collision between two static zonotopes. In the other scenario, we need to repeatedly detect the collision between two zonotopes in different positions or orientations. In latter case, preprocessing is allowed to accelerate the subsequent collision detection. This scenario is often encountered in applications dealing with dynamic scenes such as moving objects. All of these problems have been studied extensively for convex bodies. Of course, a zonotope is a convex object and thus any algorithm for convex objects applies to zonotopes as well. However, the explicit representation of a zonotope with n generators needs $\Theta(n^2)$ storage. Direct application of the existing algorithms to zonotopes becomes inefficient. Therefore, the major challenge is to design efficient algorithms that work for implicitly represented zonotopes. We show in this section that many algorithms developed for convex bodies have efficient counterparts for zonotopes.

3.5.1 Static collision detection

For this problem we need to ‘anchor’ zonotopes at particular points of space. Thus we will specify zonotopes by giving their center, followed by a list of their line segment generators. We treat the segment generators as vectors emanating from the origin. Note that when the center coincides with the origin, a zonotope coincides with its centrally symmetric reflection through the origin.

Given two zonotopes $Z_1 = (p, \langle v_1, \cdots, v_k\rangle)$ and $Z_2 = (q, \langle w_1, \cdots, w_m\rangle)$, we wish to decide whether Z_1 intersects Z_2, i.e. whether $Z_1 \cap Z_2 = \emptyset$. The following is well-known.

Lemma 3.5.1. *$Z_1 \cap Z_2 \neq \emptyset$ iff $q - p$ is in the zonotope $(0, \langle v_1, \cdots, v_k, w_1, \cdots, w_m\rangle)$.*

The above lemma reduces the collision detection between zonotopes to the point membership problem of a zonotope: given a point p and a zonotope $Z = (0, \langle u_1, u_2, \cdots, u_n\rangle)$ (here $n = k + m$), determine whether $p \in Z$. Of course, we may compute the explicit representation of this zonotope and apply a standard algorithm for checking whether a point is inside a convex body. This algorithm, however, will run in at least $\Omega(n^2)$ time as the number of vertices of a zonotope can be quadratic in terms of the number of generators. In this section, we present an algorithm for intersection detection with only $O(n \log^2 n)$ running time.

Recall that the boundary of Z can be decomposed into the upper hull Z^+ and the lower hull Z^-. A point is in Z iff it is directly below Z^+ and directly above Z^-. We therefore further reduce the problem to determining whether a point is directly below Z^+ or/and above Z^- (the problems are symmetric). Let Q be the boundary of the projection of Z^+ and the tiling $\mathcal{T}$ of Q be the projection of Z^+ on the xy plane (Figure 3.3). For a point p, consider its projection p_0 on the xy plane. If p_0 is outside of Q, p is not directly below Z^+. Otherwise, we locate the parallelogram T that contains p_0 and decide if p is above or below the corresponding facet on Z^+.

The tiling $\mathcal{T}$ can be viewed as a monotone planar subdivision. We will use a method similar to [EGS86] to locate the point. Namely, we perform a binary search on the monotone separators to determine the two adjacent separators that sandwich the point. We will show below that each separator consists of n line segments and can be computed in time $O(n \log n)$. Since we perform $O(\log n)$ comparisons against separators in total, the algorithm runs in time $O(n \log^2 n)$. In what follows, we shall show how to construct a separator in $O(n \log n)$ time.

We will now exploit the previously mentioned tangent space duality between zonotopes and arrangements. For a zonotope Z, let Π be the line arrangement defined in Section 3.3. Order all the vertices in Π according to their x coordinates, and index the vertices according to their order. We can assign the same index to the corresponding parallelogram in $\mathcal{T}$.

Clearly, the order is consistent with the "right-of" relationship[1] in $\mathcal{T}$. Denote by L_k the vertical line that just on the right of the k-th vertex in Π. Suppose that L_k crosses the lines $\ell_{i_1}, \ell_{i_2}, \cdots, \ell_{i_n}$ from left to right (Figure 3.3). Define the corresponding vertical pseudoline L'_k in the tiling $\mathcal{T}$ as follows. We start from the top vertex of Q and form a monotone chain S by extending $v'_{i_1}, v'_{i_2}, \cdots, v'_{i_n}$ one by one. Since L_k separates all the vertices with indices $\leq k$ from those $> k$, L'_k separates all the faces with indices $\leq k$ from those $> k$, i.e L'_k is the k-th separator, from left to right, in $\mathcal{T}$. We can compute L_k in $O(n \log n)$ time by the optimal slope selection algorithm [CSSS89]. Therefore, we have that:

Theorem 3.5.2. *For any two zonotopes with n generators in total, we can decide whether they intersect in $O(n \log^2 n)$ time.*

3.5.2 Repetive collision detection

In many settings such as in motion planning, dynamic simulation, and computer animation, we may need to perform intersection testing between two objects repetitively when they are in different configurations. When the objects are represented as polytopes, typical methods used for such collision detections include the Minkowski sum method for translational motions, hierarchical methods, and local walking techniques. We will describe how to implement those methods efficiently for implicitly represented zonotopes.

Minkowski sum method. When only translation is allowed, one standard technique is the Minkowski sum method[2]. In such a method, we compute the Minkowski sum of two convex objects and reduce the collision detection problem to a point containment problem in the Minkowski sum. According to the earlier duality, this is very similar to point location in line arrangements. Point location in line arrangements is a very well studied topic in Computational Geometry. The best known trade-off between preprocessing and query time is roughly $O(\frac{n}{\sqrt{m}})$ query time by using $O(m)$ preprocessing time and space. Unfortunately, we are unable to achieve the same bound for our problem. Instead, we have the following weaker trade-off.

[1]We use "right-of" instead of "above" notation just for exposition convenience.

[2]The Minkowski sum method works for rotations too, but it raises the dimension from three to six and increases the complexity significantly.

Lemma 3.5.3. *For any zonotope with n generators, for any $n \leq m \leq n^2$, we can preprocess it into a data structure with $O(m)$ space so that the membership query can be answered in time $O(\frac{n^2}{m}\log^2 n)$.*

Proof: We compute a $(1/r)$-cutting Δ of the dual arrangement: a set of $O(r^2)$ interior disjoint trapezoids that refine the arrangement of r lines so that each trapezoid is crossed by $O(n/r)$ lines. We then map the cutting to the tiling $\mathcal{T}$ of Q. Each line ℓ_i is mapped to a pseudoline ℓ'_i which is the bisector of the strip corresponding to the generator v_i (Figure 3.3). We perturb each vertical thread to its right and map to its corresponding vertical pseudoline as defined before. This way, we obtain a planar subdivision Δ' of Q. Each cell in the subdivision corresponds to a trapezoid in Δ. Further, the line-trapezoid incidence relationship in (Π, Δ) is preserved. Thus, we can compute and store a $O(\log n)$ query time point location data structure for Δ' and associate the crossing lines with each cell in Δ'. For any query point, we first locate it in Δ' and then use the algorithm as shown in Theorem 3.5.2 to locate the point. The first step takes $O(\log n)$ time, and the second step takes $O(\frac{n}{r}\log^2 n)$ time as each cell is crossed by $O(n/r)$ lines. As for the preprocessing, since each pseudoline has complexity $O(n)$, the complexity of the arrangement of r pseudolines is $O(nr)$. In addition, each cell needs to store $O(n/r)$ lines and there are $O(r^2)$ cells. The storage in total is $O(nr)$. By setting $r = m/n$, we obtain the bound as claimed. $\square$

The above algorithms can also be used to compute the polyhedral distance defined by the zonotope.

Corollary 3.5.4. *Given a polyhedral metric defined by a zonotope with n generators, for any $n \leq m \leq n^2$, we can construct a data structure using $O(m)$ storage and in $O(m \log n)$ time, so that the distance between any two points can be computed in time $O(\frac{n^2}{m}\log^2 n)$.*

Proof: For any two points p, q, we locate the face intersected by the ray from the origin to $q - p$ and then compute the Minkowski distance. The intersection can be reduced to point location in the mapping of the boundary of the zonotope to a sphere centered at the origin. The same technique and bound apply. $\square$

Hierarchical method. In the hierarchical method [DK83, EGSZ99], a series of bounding volumes are computed to approximate the object with higher and higher accuracy. The collision detection between two objects is by starting from the coarsest level of the bounding volumes and descending until we separate two bounding volumes or detect the collision between the two objects. Here, we wish to emulate the hierarchical method for implicitly represented zonotopes.

Denote by $H(A, B)$ the Hausdorff distance between two point sets A and B and by $D(A)$ the diameter of a point set A. What is crucial in bounding volumes construction in [EGSZ99] is a the well-known approximation property of convex bodies: for any convex object A in three dimensions and for any $\epsilon > 0$, there exists another convex body B with $O(1/\epsilon)$ vertices so that $H(A, B) \leq \epsilon D(A)$ [Dud74]. There are similar results for zonotopes. In [BM83], it is shown that a unit ball in $\mathbb{R}^3$ can be approximated within Hausdorff distance ϵ by a zonotope with $O(1/\epsilon)$ generators. The proof is constructive but only works for Euclidean balls. In [BLM89], it is proven that in d-dimensions, any zonoid A can be approximated within $\epsilon D(A)$ by a zonotope with $O(1/\epsilon^{2+\tau} d)$ generators, for any $\tau > 0$. But the proof is non-constructive. In the following, we show that for any zonotope with n generators, we can construct an approximation efficiently.

Lemma 3.5.5. *For any zonotope A in $\mathbb{R}^3$ and for any $\epsilon > 0$, there exists a zonotope B with $O(1/\epsilon^2)$ generators, so that $H(A, B) \leq \epsilon D(A)$. Further, B can be computed in $O(n)$ time where n is the number of the generators of A.*

Proof: Suppose that $A = (0, \langle v_1, v_2, \cdots, v_n \rangle)$. By symmetry, we can assume that all the v_i's have positive z components. We normalize every v_i and each v_i corresponds to a point p_i on the unit hemisphere. Now, we subdivide the unit hemisphere into k patches so that for any two points p, q in the same patch, the angle between op and oq is bounded by $O(1/k^{1/2})$. Suppose that the patches are $C_1, C_2, \cdots, C_k$. For each C_j, pick a point q_j in C_j and denote by $n_j = \overrightarrow{oq_j}$.

Now divide all the v_i's into clusters according to the patches they are in. Define $V_j = \{v_i | p_i \in C_j\}$. For each vector $v_i \in V_j$, we form two vectors: $v_i^1 = (v_i \cdot n_j) n_j$ is the projection of v_i on the direction n_j, and $v_i^2 = v_i - v_i^1$. Set $u_j = \sum_{v_i \in V_j} v_i^1$, for $1 \leq j \leq k$. Consider the zonotope $Z = (0, \langle v_1^2, v_2^2, \cdots, v_n^2 \rangle)$. Suppose that $(0, \langle w_1, w_2, w_3 \rangle)$ is the tightest axis aligned bounding box of Z. (w_1, w_2, w_3 can be computed easily by projecting each v_i^2 to the x, y, z axes). Now, consider the zonotope $B = (0, \langle u_1, u_2, \cdots, u_k, w_1, w_2, w_3 \rangle)$. We claim that $H(A,B) \le \epsilon D(A)$ if $k = c/\epsilon^2$ for some constant $c > 0$.

Because $v_i = v_i^1 + v_i^2$, we have that

$$\begin{aligned} A &\subset (0, \langle v_1^1, v_2^1, \cdots, v_n^1\rangle) \oplus (0, \langle v_1^2, v_2^2, \cdots, v_n^2\rangle) \\ &\subset (0, \langle u_1, u_2, \cdots, u_k\rangle) \oplus Z \\ &\subset (0, \langle u_1, u_2, \cdots, u_k\rangle) \oplus (0, \langle w_1, w_2, w_3\rangle) = B\,. \end{aligned}$$

Denote by $\|v\|$ the Euclidean length of the vector v. For any point $p \in B$ of the form $p = \sum_i \alpha_i u_i + \sum_j \beta_j w_j$, let $q = \sum_i \alpha_i \sum_{v_j \in V_i} v_j \in A$. It suffices to prove that $\|pq\| \le \epsilon D(A)$ if $k = c/\epsilon^2$ for some constant $c > 0$. First, it is easy to verify that $\|pq\| \le c_0 \sum_i \|v_i^2\|$ for some $c_0 > 0$. According to the property of the subdivision, we have that $\|v_i^2\| \le c_1/k^{1/2}\|v_i\|$ for some $c_1 > 0$. Therefore $\|pq\| \le c_2/k^{1/2} \sum_i \|v_i\|$. Further, it is not hard to see that $D(A) \ge c_3 \sum_i \|v_i\|$ for some constant $c_3 > 0$. Set $c = (\frac{c_2}{c_3})^2$. Then we have that $\|pq\| \le \epsilon D(A)$, if $k = c/\epsilon^2$. Therefore, $H(A,B) \le \epsilon D(A)$, and B has $O(1/\epsilon^2)$ generators. □

Local walking method. In a local walking method [LC91, Mir97], the closest pair of features (vertices, edges, or faces) between two convex objects is tracked for two objects. It is shown in [LC91] that a simple local walking strategy is guaranteed to find the closest pair of features between two convex volumes. If the motion is small, then the closest pair at any time step should not be "far" from the previous step, and therefore the walking should terminate in a small number of steps. Now, we show that the local walking method can be applied to zonotopes as well. What is crucial in Lin-Canny's method is the ability to discover the neighboring features of any given feature. This is easy for an explicitly represented polyhedron. However, again we cannot afford to construct the explicit representation of a zonotope. Instead, we show that it is easy to construct the neighborhood of any feature on the fly. For simplicity, consider the walking from face to face in $\mathcal{T}$. On each face of $\mathcal{T}$, there are four choices to choose to which neighbor to exit. We have that

Lemma 3.5.6. *For any zonotope with n generators, after preprocessing with $O(n)$ space and $O(n \log n)$ time, we can perform the face-to-face walking in $O(\log^2 n)$ time per step.*

Proof: Again, by duality, the walk is to determine the vertices adjacent to the vertex dual to a face in $\mathcal{T}$. This can be done by maintaining a dynamic convex hull data structure. The classical algorithm by Overmars and van Leeuwen [OvL80] gives us the desired bound. $\square$

3.6 Zonotopal space-time volumes

As we mentioned, an important benefit of zonotopes is that their description complexity can be varied or adjusted according to the application needs. In this section we illustrate how this can be exploited for collision detection involving *space-time volumes* [Cam90, Hub93, Hub95].

Consider a simple scenario where we have two zonotopes P and Q moving rigidly in $\mathbb{R}^3$. We are interested in verifying that their paths do not collide. In a typical implementation, the dynamics of P and Q are controlled by an integrator. At each time time step the positions of the bodies are updated by the dynamics module, and a new collision test is performed, using (say), the algorithm described in Section 3.5. Note that in this approach, the rate of collision checking is determined entirely by the system dynamics. It is possible that collision may be missed, if P and Q overlap, and then stop overlapping, within a single time step. It is also possible that unnecessary collision checks are done, as when the two bodies are far away.

A way to address both of these concerns is to do collision checking not on P and Q, but on the portions of space swept by P and Q during a period of time Δt, the so-called space-time volumes of the two bodies. In this section we show how to enclose these space-time volumes in bounding zonotopes. Note that if these bounding volumes are disjoint, then P and Q cannot collide at any time during the interval Δt. If the volumes intersect (either the bounding zonotopes, or the actual space-time volumes swept by the bodies), then P and Q may or may not collide during Δt. Note that P and Q collide if they occupy the same space at the same time. Our space-time volumes are 3-D and are the spatial projections of the real 4-D space-time volumes. Thus, if P collides with Q's location at a different time during Δt, this will lead to a space-time volume intersection, even though P and Q have not collided. When such collisions are detected, the interval Δt can be cut in half and the process repeated, until either a real-collision is detected, or non-intersection is confirmed. We omit further details here.

If the body P just translates during the interval Δt, then its translation vector v can just be added to the list of generators for P to produce a zonotopal description for the exact space-time volume swept by P. The new zonotope needs to be anchored at the origin of P, translated by $v/2$. This simple case illustrates the power of the zonotope description. Of course we must handle the case of a more general rigid motion during Δt. Besides translation, there can be a rotational component as well. Let z be the rotation axis and θ the rotation angle; we assume that $\theta < 180°$—a condition that should be easy to satisfy since in general Δt is small. The rotational component causes each vertex of P to move along a circular arc centered on the z axis, on a plane normal to the axis. Figure 3.5 below depicts these vertex arcs, projected onto a plane normal to the z-axis and moved to a common origin.

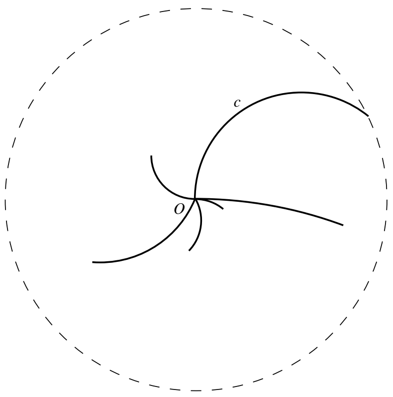

Figure 3.5: The rotational motion of the vertices of P.

If we can enclose these arcs in a tight-fitting parallelogram (a 2-D zonotope), we augment the generators of P with the translation vector v and these two 'rotational' generators, to produce a space-time bounding volume for the rigid motion of P. The resulting zonotope needs to be centered at the center of P, offset by $v/2$ and the offset between the common origin of the arcs and the parallelogram center in Figure 3.5. The fact that this zonotope bounds all placements of P during the rotation follows, because the space-volume contain all vertices of each such placement, and therefore (by convexity) all of P throughout Δt.

Since we expect Δt to be small, we also expect the set of arcs we need to enclose to be small in length. However, the number of such arcs can be $\Theta(n^2)$ (where n denotes the number of generators of P), so we wish to avoid looking at all these arcs individually.

Looking at Figure 3.5, we claim that a particular arc c is contained in a circle centered at the origin and passing through its other endpoint, because by assumption each arc spans an angle of less than $180°$. Thus all arcs are fully enclosed in a circle centered at the origin, whose radius is the distance to the the most distant endpoint of any of the arcs. This circle can in turn be enclosed in a square, which forms our bounding parallelogram.

It remains to show how to compute the distance of the vertex of P most distant from the rotation axis z. To do so it suffices to project all generators of P onto a plane normal to z and simply compute the 2-D zonotope generated by them in $O(n \log n)$ time, then select the most distant of the $2n$ vertices thus formed. Thus we have shown that:

Theorem 3.6.1. *Given a rigid motion of P over interval Δt, a space-time bounding zonotope for all placements of P can be computed by adding three generators to P. These generators can be computed in $O(n \log n)$ time.*

Note that, since our space-time volumes are spatial projections of the true 4-D space time volumes, we need not assume that P moves with constant velocity and angular acceleration during its rigid motion. All that matters is the set of spatial positions occupied and not the times at which they were. It would be interesting to explore the idea of working directly with 4-D zonotopes that are bounding volumes in true space-time, but we have not explored that path since we currently lack an efficient intersection test for 4-D zonotopes.

Chapter 4

Implicit Bounding Volume Hierarchies for Deformable Necklaces

4.1 Introduction

An aspect of motion that has not been adequately modeled in previous work is that objects in the world are often organized into groups and hierarchies and the motions of objects in the same group are highly correlated. For example, though not all points in an elastic bouncing ball follow exactly the same rigid motion, the trajectories of nearby points are very similar and the overall motion is best described as the composition of a global rigid motion with a small local deformation. Similarly, the motion of an articulated figure, such as a man walking, is most succinctly described as a set of relative motions, say that of the upper right arm relative to the torso, rather than by giving the trajectory of each body part in world coordinates. All theoretical analysis to-date are based on the assumption of independently moving objects. By ignoring such motion coherence we run the danger of developing sub-optimal algorithms that do not exploit well the structure of the problem. A similar call for realistic input models in geometric algorithms was made in [dBKvdSV97].

In this chapter we propose to study a model for deformable 'linear' objects such as macro-molecules, muscles, ropes, etc. Though our objects live in $\mathbb{R}^2$ or $\mathbb{R}^3$, they have an essential one-dimensional character that we heavily exploit in our algorithms. It is worth noting that, though modeling some aspects of linear objects is simpler than modeling surfaces or solids, linear objects can come into self-proximity and self-contact in more elaborate ways than their higher-dimensional counterparts. So from a certain point of view, dealing with collisions for deformable linear objects is the hardest case.

We call our linear objects *necklaces* and the spherical elements used to model them *beads*. The exact way in which a necklace moves and deforms depends on the physical model used and is application dependent. We focus on tracking different geometric attributes of a necklace, in particular, an bounding volume hierarchy of the necklace.

One of the key contributions made in this chapter is an implicit bounding volume hierarchy for our deforming necklaces. We exploit the linear structure of the necklaces to obtain bounding volume hierarchies that are simply combinatorial structures attached on the necklaces instead of on the ambient space. By representing the bounding volume hierarchy implicitly as a combinatorial structure, the bounding volume hierarchy is stable under motion and can be maintained efficiently. Furthermore, the bounding volume hierarchy remains tightly fitting throughout the motion.

We also consider to use power diagrams (generalized Voronoi diagrams) to track proximities between the beads in a necklace. The power diagram can be readily maintained under the KDS setting, and it provides a complement to our bounding volume hierarchies, working well when the bounding volume hierarchies does not.

4.2 Our approach and results

In this chapter, we investigate bounding sphere hierarchies for proximity maintenance and collision detection of deforming necklaces. Given that our atomic elements are themselves balls, our choice of spheres as our bounding volumes is natural. Spheres do not bound as tightly as oriented bounding boxes (in the limit they have linear as opposed to quadratic convergence to an underlying smooth shape [GLM96]), but intersection/containment tests among them are especially simple and their symmetry makes rigid motions straightforward to implement.

The various hierarchies discussed above for static geometry aggregate bounding volumes based on spatial proximity. When an object undergoes large deformations, however, spatial proximity is variable over time and cannot be used as a reliable guide to aggregation.

We have decided to base the hierarchy we will use on *topological proximity* in the object, because this notion of proximity is better preserved. For our linear necklace this gives us an especially simple rule for aggregation: we build a balanced binary tree on the sequence of beads $\{B_1, B_2, \cdots, B_n\}$ so that the intermediate aggregates correspond to the sets of leaves that are descendants of internal nodes in the tree.

The specific sphere hierarchy we use is based on bounding volumes which are the smallest spheres containing relevant parts of the original geometry — which are contiguous substrings of beads in our setting. This gives each bounding sphere a small combinatorial description: such a sphere is determined by at most four of the enclosed beads (3 in $\mathbb{R}^2$). As our necklace deforms, the geometry of this sphere hierarchy also changes continuously. However, the combinatorial descriptions of these bounding spheres (in other words, the lists of beads defining them) change at only discrete events. This discrete nature of the updates makes it possible to maintain the bounding spheres efficiently under motion, in contrast to non-combinatorially defined bounding volumes

In general, a bounding volume hierarchy is formed by creating a balanced recursive partitioning of the underlying geometry and computing a bounding volume enclosing each group. Once the partitioning is determined, there are two ways to form the bounding volume hierarchy. One is, as we do, to compute a tight bounding volume on the geometry in each group, which we call the 'wrapped hierarchy', and the other is to compute the bounding volume of the bounding volumes of the children subgroups, which we call the 'layered hierarchy'. Clearly, the wrapped hierarchy is always tighter than the layered hierarchy. In this paper, we study first the relationship between the two hierarchies. We show the somewhat surprising result that, in the worst case, a bounding sphere in the layered hierarchy is at most a factor of $\sqrt{\log n}$ bigger than the corresponding one in the wrapped hierarchy, and that this bound is tight. Furthermore, the bound holds in any dimension.

The most important application of bounding hierarchies is in collision and self-collision checking. While such methods work well in practice, in the worst case nothing better than the trivial quadratic bound was previously known. This bound arises in the case where both hierarchies are traversed completely and all leaves of one have to be checked for intersection against all leaves of the other. We show that, with a simple heuristic, the folklore self-collision checking method using the sphere hierarchy and local refinement as necessary achieves sub-quadratic time bounds: $O(n \log n)$ in two dimensions, and $O(n^{2-2/d})$ in d-dimensions for $d \geq 3$ — to our knowledge, this is the first sub-quadratic worst-case bound for collision detection algorithms using bounding volume hierarchies[1].

The power diagram is another tool that people often use to deal with balls. While it has been known that the closest pair of a set of disjoint balls defines an edge in the power diagram ([GZ98]), that result does not apply directly to our problem since we allow adjacent spherical elements to overlap. We show that the power diagram can be used to compute the closest pair in a deforming necklace as well. It is interesting to note that the worst-case for the sphere hierarchy occurs for highly packed necklaces, while these are actually very favorable cases for the power diagram — in such cases the power diagram size is linear in all dimensions [Eri02].

In Section 4.3 we present the formal setting of our beads and necklaces. Section 4.4 discusses the precise bounding sphere hierarchy we have chosen to implement and the reasons for our choice. We present verification and repair algorithms for maintaining the hierarchy as the necklace deforms, and compare the tightness of the wrapped and layered hierarchies. Section 4.5 presents a number of combinatorial results about collision detection using the sphere hierarchy or the power diagram.

4.3 Beads and necklaces — definitions

A *necklace* consists of a sequence of n closed balls $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, called *beads*, in the Euclidean space $\mathbb{R}^d$. We assume that only adjacent balls along the necklace may intersect and no ball is fully contained in another set of balls. In some contexts we make further assumptions. These include:

uniformity: there is a constant $\alpha \geq 1$ such that the ratio of the radii of any two balls in a necklace is in the interval $[1/\alpha, \alpha]$;

[1]A similar bound was reported concurrently with our work for oriented bounding boxes in [LSHL02]. As already mentioned, they studied kinematic chains similar to our necklaces, although with a different motivation.

limited overlap: two consecutive balls B_i and B_{i+1} along a necklace are allowed to overlap, but the angle of their normals at a common point on their surfaces is bounded below by β.

connectivity: the beads form a connected set — in other words, any two consecutive beads along the necklace have a point in common.

Whatever conditions we adopt, we assume that they are maintained by the underlying physics causing a necklace to move or deform. We remark that similar 'necklace conditions' were studied for establishing the optimality of tours in the plane [ERW89].

We define the following terms to be used in later sessions. In a sphere hierarchy built on top of a necklace, the internal nodes of the hierarchy are called *cages*. When the cage is defined as the minimum enclosing sphere (MES) of the beads it contains, the geometry of the cage is fully determined by a constant number of the beads which we call the *basis beads*.

The sphere hierarchy is based on a balanced binary tree built on top of the necklace, which each bead of the necklace is a leaf in the tree. Let $L(t)$ be the set set of beads corresponding to leaves of the subtree rooted at node t. Note that $L(t)$ is a contiguous subchain of the original necklace, which we will call the *canonical subnecklace*. Each cage corresponds to an internal node, t, of the tree and bounds all the beads in $L(t)$. This is one instance where we heavily use the *a priori* known structure of the type of object we are modeling.

4.4 The wrapped hierarchy

4.4.1 Definition and properties

We define the *wrapped hierarchy* of a necklace to be the sphere hierarchy corresponding to the balanced tree described above, where the sphere corresponding to each internal node is the *minimum enclosing sphere* (MES) of the beads in the canonical sub-necklace associated with that node. We call these bounding spheres corresponding to internal nodes *cages*. Note that this allows the cages of the children of a node in the hierarchy to stick out of the cage of the parent. We call the sphere hierarchy defined by making the cage of a parent to be the MES of the cages of its two children the *layered hierarchy* [Qui94]. Though the wrapped hierarchy is slightly more difficult to compute than the layered hierarchy, it is tighter fitting and most importantly it can be maintained more easily under deformation — a fact that at first seems counter-intuitive. An example of each type of hierarchy is shown in Figure 4.1.

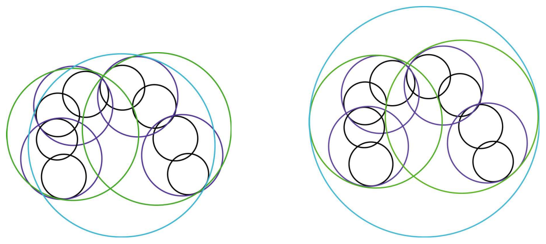

Figure 4.1: Wrapped (left) and layered (right) sphere hierarchies. The base beads are black. Notice that each cage in the wrapped hierarchy is supported by 2 or 3 beads.

The key property of the wrapped hierarchy that is of interest to us is that each cage, being a minimum enclosing sphere of a set of beads, is fully determined by a small number (two, three, or four in $\mathbb{R}^3$) of the basis beads in the associated canonical sub-necklace. Note that the cage of an internal node is also the minimum enclosing sphere of its basis beads. When a necklace deforms, the basis of each cage remains constant for a period. At certain discrete events the basis of a cage changes typically by a pivoting step in which (1) an old basis bead leaves the basis, and (2) a new bead from the enclosed sub-necklace enters the basis. At times only one of these events may happen, but the total number of basis beads will always remain between two and four. Thus, although during continuous necklace deformation the cages deform continuously, their combinatorial descriptions stay constant and change only at discrete events. This combinatorialization of a continuous phenomenon is an insight analogous to what is exploited in kinetic data structures.

We expect that under smooth deformation the combinatorial description of the cages (i.e. their basis beads) will stay constant for a fairly long time, and when finally the basis of a cage needs to change, that change will be easy to detect and the *basis update* simple to perform. For instance, in Figure 4.2 we show an example in $\mathbb{R}^2$ of the upper layers

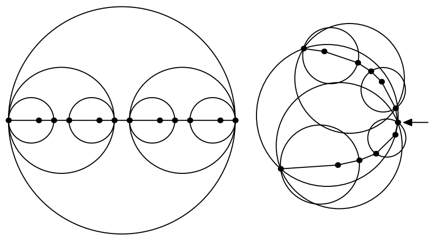

Figure 4.2: A combinatorially defined sphere hierarchy is stable under deformation. Only the top level cage differs between the two conformations.

of such a hierarchy in two quite different configurations of a deforming necklace. It so happens that all the hierarchy cages except for the root cage continue to have the same combinatorial description at all intermediate configurations.

While the wrapped hierarchy is always tighter than the layered hierarchy, it is interesting to know exactly how much difference there can be between the two. In the following, we consider the case where all the beads are points (or equivalently equal radius spheres), and in arbitrary position. We have the following result:

Theorem 4.4.1. *For any given set of n points in any dimension and any binary tree with depth $\lceil \log_2 n \rceil$ on the points, if we denote by τ_W, τ_L the radii of the root spheres for the wrapped and layered hierarchies of the point set, respectively, then $\tau_L \leq \tau_W \sqrt{\lceil \log_2 n \rceil}$. The bound is almost perfectly tight, as we can construct a set of points so that $\tau_L \geq \tau_W \sqrt{\lfloor \log_2 n \rfloor}$.*

We denote the minimum enclosing sphere (MES) of a set of beads S by $M(S)$ and the MES corresponding to a node t by $M(t)$. The upper bound result is implied by the following lemma.

Lemma 4.4.2. *Let O and R be the center and the radius of $M(S)$, and O_ℓ and R_ℓ the center and the radius of a ball on level ℓ in the layered hierarchy of S. Let $d_\ell = |OO_\ell|$. Then $R_\ell^2 \leq \ell(R^2 - d_\ell^2)$.*

Proof: Without lost of generality, let us assume that $R = 1$. We prove the lemma using induction. The lemma is clearly true for the 0-th level, as $R_0^2 \leq (1 - d_0)^2 \leq 1 - d_0^2$. We assume that the lemma holds for all balls at level ℓ in the layered hierarchy and show that the lemma still holds for balls at level $\ell + 1$.

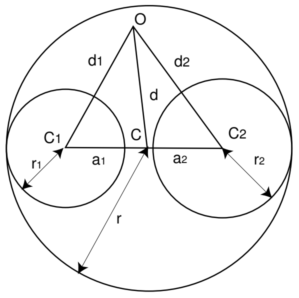

Figure 4.3: A ball and its two children in a layered hierarchy. Lemma 4.4.2 proves that the farther the centers of the children are from the center of their parent, the smaller their radii must be in comparison.

Let C and r be the center and the radius of a ball at level $\ell+1$ in the layered hierarchy, and let C_1, C_2 and r_1, r_2 the centers and radii of its two children. Let $d = |OC|$, $d_1 = |OC_1|$, and $d_2 = |OC_2|$, see Figure 4.3. By the induction assumption, $r_1^2 \leq \ell(1-d_1^2)$ and $r_2^2 \leq \ell(1-d_2^2)$. We would like to show that $r^2 \leq (\ell+1)(1-d^2)$. This is clearly true when the parent ball at C is identical to one of its children balls. We consider the general case when the parent ball is bigger than both of its children.

From $\cos(\angle OCC_1) = -\cos(\angle OCC_2)$ and the law of cosines, we obtain:

$$\begin{aligned}
\frac{a_1^2+d^2-d_1^2}{2a_1 d} &= -\frac{a_2^2+d^2-d_2^2}{2a_2 d}, \text{ and therefore} \\
a_2(a_1^2+d^2-d_1^2) &= -a_1(a_2^2+d^2-d_2^2), \text{ or} \\
(a_1+a_2)d^2 &= (a_2 d_1^2 + a_1 d_2^2) - (a_2 a_1^2 + a_1 a_2^2), \text{ or} \\
d^2 &= \frac{a_2 d_1^2 + a_1 d_2^2}{a_1+a_2} - a_1 a_2 \,. \qquad (4.1)
\end{aligned}$$

Using Equation (4.1) we have

$$\begin{aligned}
d^2 &\leq \frac{(a+\delta)(1-r_1^2/\ell)+(a-\delta)(1-r_2^2/\ell)}{2a} - (a-\delta)(a+\delta) \\
&= \frac{2a - a(r_1^2+r_2^2)/\ell - \delta(r_1^2-r_2^2)/\ell}{2a} - (a^2-\delta^2) \\
&= 1 - \frac{1}{2\ell}(r_1^2+r_2^2) - \frac{\delta}{2a\ell}(r_1^2-r_2^2) - (a^2-\delta^2) \\
&= 1 - \frac{s^2+\delta^2}{\ell} - \frac{2s\delta^2}{a\ell} - (a^2-\delta^2)\,.
\end{aligned}$$

Thus,

$$(\ell+1)(1-d^2)-r^2 \ \geq\ (\ell+1)\big[\frac{s^2+\delta^2}{\ell}+\frac{2s\delta^2}{a\ell}+(a^2-\delta^2)\big]-(s+a)^2 \,.$$

Simplifying the right hand side of the above inequality, we get:

$$(\ell+1)(1-d^2)-r^2 \geq \frac{(\ell a-s)^2a_1a_2+(s+a)^2\delta^2}{\ell a^2} \geq 0\,.$$

Thus, $r^2 \leq (\ell+1)(1-d^2)$. This completes the inductive proof. □

The above lemma immediately implies the upper bound in Theorem 4.4.1 as the depth of the tree is bounded by $\lceil \log_2 n \rceil$. Furthermore, we can show that the inequality in Lemma 4.4.2 can be made tight, and we can construct a set of points to attain the upper bound.

Lemma 4.4.3. *There is a set of n points in the plane such that $\tau_L \geq \tau_W\sqrt{\lfloor \log n \rfloor}$, where τ_W, τ_L denote the radius of the root ball in the wrapped and layered hierarchy, respectively.*

Proof: It suffices to consider the case $n = 2^k$. We will construct a collection of points such that their wrapped hierarchy has radius 1 and their layered hierarchy has radius $\sqrt{k}$. The construction is done incrementally. We first fix any point O and place a point at O_0 such that $|OO_0| = 1$.

Suppose that we have constructed the set S_ℓ, the first 2^ℓ points. Let O_ℓ be the center of the ball covering S_ℓ in the layered hierarchy. To construct the set $S_{\ell+1}$, we first find the point $O_{\ell+1}$ such that (1) $\angle OO_{\ell+1}O_\ell = 90°$, and (2) $|O_\ell O_{\ell+1}| = 1/\sqrt{k}$. We can then construct S'_ℓ by flipping all the points in S_ℓ about the line $OO_{\ell+1}$. Finally we set $S_{\ell+1} = S_\ell \cup S'_\ell$. See Figure 4.4.

It is straightforward to show that in the above incremental construction, the ball covering S_ℓ in the layer hierarchy has radius $\ell/\sqrt{k}$, and $|OO_\ell| = \sqrt{1-\ell/k}$. Thus we can always find the point $O_{\ell+1}$, as long as $\ell < k$. It is clear that the root ball of the wrapped hierarchy has unit radius as all the points constructed are on the same circle, and the root ball of the layered hierarchy has radius $\sqrt{k}$.

Theorem 4.4.1 is the combination of Lemmas 4.4.2 and 4.4.3. □

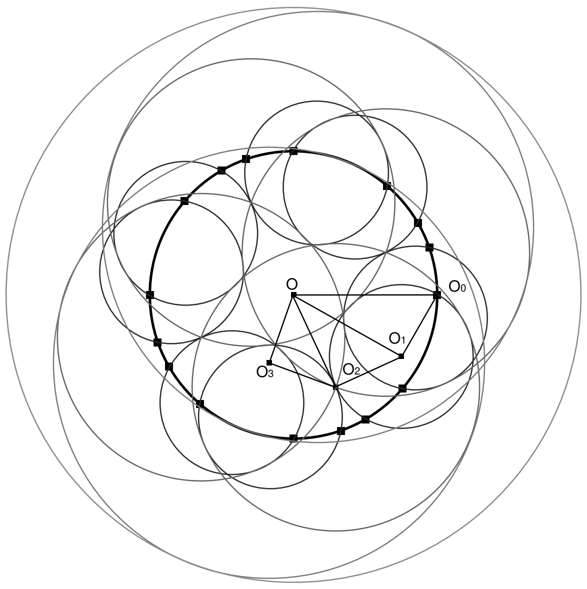

Figure 4.4: The construction of a set of 16 points on a circle of radius 1 such that the root circle of their layered hierarchy has radius $\log(\sqrt{16}) = 2$. The point O_0 is chosen arbitrarily. Right triangles OO_0O_1, OO_1O_2, OO_2O_3 are then constructed such that $|O_0O_1| = |O_1O_2| = |O_2O_3| = 1/2$. The point set is constructed as the closure of the singleton set $\{O_0\}$ with respect to the reflections over the lines OO_1, OO_2, OO_3, and the refection over the point O. The circles in the layered hierarchy are also shown.

4.4.2 Construction and maintenance

It is straightforward to construct the wrapped hierarchy by directly computing the minimum enclosing sphere (MES) for the canonical set of each cage. There is a complex algorithm for computing MES of a set of points or equal radius spheres in $O(n)$ time in the worst case [Meg82] and a simple one running in expected $O(n)$ time [Wel91]. If the basis beads are balls of different radii, the algorithm is slightly more complicated but with the same overall running time [KF03]. Therefore, it takes $O(n \log n)$ time to construct initially the wrapped hierarchy of the necklace of n beads.

To maintain the wrapped hierarchy, we need to verify the correctness of the hierarchy, i.e. the correctness of the basis beads at each internal node, after a simulation time step, and update those which are no longer correct. This task is simplified by the following observation:

Lemma 4.4.4. *Let $\mathcal{B}$ and $\mathcal{S}$ be sets of beads with $\mathcal{B} \subseteq \mathcal{S}$. If all beads $B \in \mathcal{S}$ are contained in $M(\mathcal{B})$, then $M(\mathcal{S}) = M(\mathcal{B})$. Thus, the basis for $M(\mathcal{S})$ is the same as the basis for $M(\mathcal{B})$.*

Note that the basis can shrink, so if $\mathcal{B}$ was the basis of $M(\mathcal{S})$ before an update, and after the update $M(\mathcal{S}) = M(\mathcal{B})$, then some (not necessarily proper) subset of $\mathcal{B}$ is the new basis for $M(\mathcal{S})$.

The verification is done in a hierarchical manner, bottom up, with a top down pass from each tree node; we call this method the *cascade verification*. Suppose that we have checked the validity of the nodes of a subtree under a node c. We first compute the minimum enclosing sphere M of the basis beads from the previous time step. According to Lemma 4.4.4, in order to check the correctness of c, it is sufficient to check that all the beads in $L(c)$, are contained in $M(c)$. This can be either done directly in linear time, which we call *naive verification* or indirectly as follows. Maintain a frontier, F, initially containing the children of c. At each step we take a node d out of the frontier and check if $M(d)$ is contained in M. If it is not, there are two cases — if the node is an internal node we add its children to F; otherwise it is a leaf, so we know that the node has escaped and the basis of node c is no longer valid and needs to be updated. If we can continue the above process until F becomes empty without encountering an escaped leaf node, we know that the basis for $M(c)$ is still valid. It is tempting to try to accelerate the above process by noting that the geometry belonging to a cage must be contained in the intersection of the cages on the path from that node to the root, and checking to see if this volume is contained in the cage being verified. However, in practice the extra complexity of this check more than outweighs its benefits. While in the worst case, the above procedure may take $\Theta(n \log n)$ time if all paths need to be traversed, our experiments suggest than in most instances the actual time is closer to linear, as only paths to the basis beads need to be checked.

When beads escape from an enclosing cage c, a basis update must be performed. At least one of the escaped beads must be a basis bead for the new $M(c)$. The LP-type algorithm [Wel91] allows easy exploitation of this knowledge in a natural manner, as well as easy use of other heuristic information about which beads are likely to be basis beads. The cost of the update is expected to be linear in the size of the canonical subchain.

4.5 Collision and self-collision detection

4.5.1 Separating necklaces using the wrapped hierarchy

Bounding volume hierarchies are often used for collision detection algorithms between two objects or for self-collision detection. Typically, such an algorithm tries to derive a non-intersection proof by finding a sufficient set of *separating bounding volume pairs* while walking down the hierarchy. It reports a collision if it fails to produce such a set of pairs. More precisely, suppose that B is a set of n disjoint objects $\{B_1, B_2, \ldots, B_n\}$ (beads in our hierarchy) and T is a tree built on the elements of B. Each internal node t in T corresponds to a bounding volume containing the objects at the leaves of the subtree rooted at t. This way, we create a collection P of objects, among which n are basic objects. A set C of pairs $\{(c_i, c_i')\}$, where $c_i, c_i' \in P$, is called a collection of separating pairs for B if:

1. for any i, c_i and c_i' are disjoint, and
2. for any i, j, where $i \neq j$, there exists a k so that $B_i \in c_k$ and $B_j \in c_k'$.

Such a separating pair set serves as a proof of the non-collision of the set B, i.e. it exists if and only if all the objects in B are mutually disjoint. The minimum number of pairs needed to separate a set of objects is crucial as it is a lower bound of the cost of a collision detection algorithm taking such an approach. In our problem, the objects are beads and the bounding volumes are cages, and the hierarchical structure is the wrapped hierarchy. We also need to relax requirement 2 to $i \neq j-1, j, j+1$ as we allow adjacent beads to intersect each other. There has been some prior research on separating a set of balls. When we are allowed to group balls arbitrarily, there always exist a set of $O(n)$ separating pairs for n disjoint balls [HSS83, CK95a]. However, to our knowledge, the separating pairs for predefined hierarchical structures have not been studied combinatorially before. Here, we will show that for the wrapped hierarchy in d dimensions, there always exists a separating pairs collection (a *separating family*) with $O(\max\{n \log n, n^{2-2/d}\})$ members, if there is no collision between any pair of two non-adjacent beads. In the following, we do not distinguish a node in the tree and the cage built for that node.

Theorem 4.5.1. *Let $\mathcal{B} = \{B_1, B_2, \cdots, B_n\}$ be a sequence of n beads in d-dimensions (for $d \geq 2$), satisfying the uniformity and limited overlap assumptions. Then there exists a separating family for $\mathcal{B}$ of size $O(\max\{n \log n, n^{2-2/d}\})$ among the cages in its wrapped hierarchy.*

Proof: By the uniformity and limited overlap assumption, we have that there exists a constant $\alpha \geq 1$ such that the distance between the centers of two adjacent beads is in the interval $[1, \alpha]$. We give an algorithm for constructing a separating family Σ. Throughout the proof, $\log$ is understood as $\log_2$.

Fix an integer i such that $0 \leq i \leq \log n - 1$. Set $r_i = 2^i$. Let D_j be the cage in the wrapped hierarchy that encloses the beads $B_{(j-1)r_i+1}, \ldots, B_{jr_i}$. Clearly, the radius of D_j is at most $R_i = r_i\alpha$. Let $\mathcal{D}^i = \{D_1, D_2, \ldots\}$ be the resulting set of balls. For each $D_j \in \mathcal{D}^i$, let K_j be the set of points that are distance at most $8R_i$ from a point in D_j, i.e., $K_j = \{x \mid \exists y \in D_j \quad |xy| \leq 8R_i\}$. K_j is a ball with radius at most $9R_i$ and concentric with D_j. For any ball $D_k \in \mathcal{D}^i$ that is contained in K_j and is disjoint from D_j, we report the pair (D_j, D_k). We define Σ_j^i to be the set of pairs reported for D_j. We repeat this process for all balls in $\mathcal{D}^i$ and set $\Sigma^i = \bigcup_j \Sigma_j^i$. Set $\Sigma = \bigcup_{i=0}^{\log n-1} \Sigma^i$.

We claim that Σ is a separating family for B. It is obvious from the construction that all the pairs in Σ are disjoint. We need to argue that it covers all the pairs of beads. Consider a pair of disjoint beads (B_ℓ, B_m). Denote by D_ℓ^i and D_m^i the i-th level (the level increases bottom up starting from 0) cages that contain B_ℓ and B_m, respectively. Let $t = \max\{i \mid D_\ell^i \cap D_m^i = \emptyset\}$. It is easy to see that every point in D_ℓ^t is within distance $4R_{t+1} = 8R_t$ from the center of D_m^t because D_ℓ^{t+1} and D_m^{t+1} intersect. Therefore $(D_\ell^t, D_m^t) \in \Sigma$. Hence, Σ is a separating family.

Next, we bound the size of Σ. Note that

$$\sum_{i=\frac{1}{d}\log n+1}^{\log n-1} |\mathcal{D}^i| = O(n^{1-1/d}),$$

because each ball in this set corresponds to the cage at a node in the wrapped hierarchy of

B whose depth is less than $(1/d)\log n$. Hence,

$$\sum_{i=\frac{1}{d}\log n+1}^{\log n-1} |\Sigma^i| = O(n^{2-2/d})\,.$$

It suffices to bound $|\Sigma^i|$ for $0 \le i \le (1/d)\log n$.

Every pair (D_j, D_k) in Σ^i_j "covers" $r_i^2 = 2^{2i}$ pairs of beads. For any pair of beads (B_u, B_v) covered by this pair, their centers are within distance $9R_i$ because D_k is contained in K_j. Therefore, Σ^i covers $2^{2i}|\Sigma^i|$ pairs of beads with each pair is within distance $9R_i$. But, by a packing argument, there are at most $(9R_i)^d = O(2^{di})$ beads whose centers are within distance $9R_i$ from the center of B_u. Hence, $2^{2i}|\Sigma^i| = O(n2^{di})$, which implies that $|\Sigma^i| = O(n2^{(d-2)i})$. Therefore,

$$\begin{aligned} |\Sigma| &= O(n^{2-2/d}) + \sum_{i=0}^{\frac{1}{d}\log n} O(n2^{(d-2)i}) \\ &= O(\max\{n\log n, n^{2-2/d}\})\,. \end{aligned}$$

This completes the proof of the theorem. $\square$

Remark. The above bound is tight in the worst case. Consider an $n^{1/d} \times \cdots \times n^{1/d}$ d-dimensional grid. We can form a necklace B by tracing and connecting the segments parallel to the x-axis in the grid. By the construction, B contains $n^{1-1/d}$ x-axis aligned segments, each with length $n^{1/d}$. Any separating family of B in its wrapped hierarchy has size $\Omega(\max\{n\log n, n^{2-2/d}\})$. For two parallel segments at distance δ where $\delta \in [i, i+1)$, we need $\Theta(n^{1/d}/i)$ pairs to cover the beads on them. According to the bounds on the number of lattice points in spherical shells, we know that there are $\Omega(n^{1-1/d}i^{d-2})$ such pairs of segments. So the number of pairs in any separating family is

$$\sum_{i=1}^{n^{1/d}} \Omega(ni^{d-3}) = \Omega(\max\{n\log n, n^{2-2/d}\})\,.$$

From Theorem 4.5.1, it follows that there always exists a collection of sub-quadratically many ($O(n\log n)$ in two dimensions and $O(n^{2-2/d})$ in d-dimensions for $d \ge 3$) separating pairs if any two non-adjacent balls are disjoint. The above constructive proof also suggests the following simple heuristic for collision detection using wrapped hierarchies: when two cages intersect, we always split the one containing more beads, breaking ties arbitrarily. This way, the cages we compare always have similar number of beads inside them, and their parent cages intersect. Therefore, the above proof applies — the number of pairs examined by the algorithm is bounded by Theorem 4.5.1. That is, such a collision detection algorithm takes time $O(n \log n)$ in two dimensions and $O(n^{2-2/d})$ in higher dimensions, in particular, it is $O(n^{4/3})$ in three dimensions.

4.5.2 Collision detection with the power diagram

Theorem 4.5.1 gives us a sub-quadratic bound on the running time of the collision detection method using the wrapped hierarchy. While the bound is $O(n \log n)$ in two dimensions, it is $\Theta(n^{4/3})$ in three dimensions. We observe that the wrapped sphere hierarchy is good for self-collision detection of a deforming necklace when the necklace is not too entangled. When the string of beads is highly packed, however, many cages in the hierarchy overlap with each other. We then have to traverse deeply down the hierarchy before being able to locate disjoint cages. According to a recent result by Jeff Erickson [Eri02], the Delaunay triangulation has linear complexity for a dense set of points. Although that result does not directly apply to the power diagrams, we may still expect that a similar result holds for the power diagram of a dense set of balls with comparable sizes. It was shown in [GZ98] that the closest pair of balls are neighbors in the power diagram if all the balls are disjoint, and therefore we may use the power diagram for collision detection of a set of disjoint balls. In our problem, however, consecutive beads may overlap and that result no longer applies. In the following, we first show that we can still use the power diagram to perform collision detection, even when we allow some partial ball overlaps. Then, we discuss briefly how to maintain the power diagram under our motion model.

We prove our result about the power diagram in a more general setting. Let $\mathcal{B}$ be a collection of balls and P be a set of pairs of balls. We say that $\mathcal{B}$ is disjoint with respect to P if every pair of balls are disjoint, unless it is in P. For example, a necklace is disjoint with respect to the set $P = \{(B_i, B_{i+1}) | 1 \le i \le n-1\}$. We define the closest pair of balls in $\mathcal{B}$ to be the pair with the minimum Euclidean distance among all the pairs *not* in P.

For any given P, a ball A is a neighbor of B if the pair (A, B) is in P. A ball A is a neighbor of a pair (B, C) if A is a neighbor of either B or C. A ball B is called a *proper connector* if for every neighboring ball A of B, there is another neighbor C of B such that A and C are disjoint. Balls that are not proper connectors are called *improper*. Note that when the balls are disjoint, there are no improper balls. For a necklace, only the first and last beads are improper. We then have the following result.

Theorem 4.5.2. *The closest pair of $\mathcal{B}$ must either (1) share an edge in the power diagram of $\mathcal{B}$, or (2) have a common neighbor, or (3) have an improper ball as a neighbor.*

Before we prove the theorem, let us briefly mention some terminology often used in the power diagram literature. Two balls are *orthogonal* if they intersect each other and the angle between the tangent planes at any of the common points is $90°$. When the two balls penetrate each other even deeper, they *intersect more than orthogonally*. It is well know that given two balls B_i and B_j, there is an edge between B_i and B_j in the power diagram if there is no ball intersecting both B_i and B_j more than orthogonally.

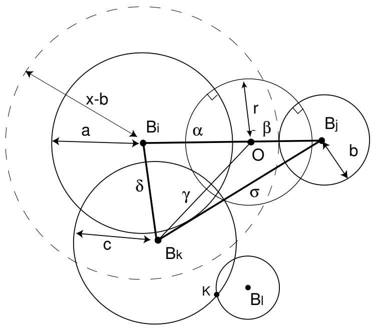

Figure 4.5: The power diagram can be used for collision detection in necklaces. The ball B_k cannot intersect ball O more than orthogonally, certifying the power diagram edge B_iB_j. x is the distance between B_i and B_j.

Proof: Suppose that (B_i, B_j) is the closest pair of balls in $\mathcal{B}$ with respect to P. Assume that (B_i, B_j) does not satisfy (2) and (3). We would like to show that B_iB_j is an edge in the power diagram of $\mathcal{B}$.

Let O be the minimum ball orthogonal to both balls B_i and B_j, and let r, a, b be the radii of O, B_i, and B_j respectively (Figure 4.5.2). Consider another ball B_k, where $k \neq i, j$, in $\mathcal{B}$. We would like to show that B_k does not intersect O more than orthogonally. According to [GZ98], it is sufficient to consider those balls intersecting either B_i or B_j. Since B_i and B_j do not share a common neighbor, we assume, without loss of generality, that B_k intersect B_i but disjoint from B_j. Let r be the radius of B_k. We denote the length of the line segments $OB_i, OB_j, OB_k, B_iB_k, B_jB_k$ by $\alpha, \beta, \gamma, \delta$, and σ, respectively, and the length of B_iB_j by $x = \alpha + \beta$. We list the following conditions that those quantities have to satisfy:

1. $\alpha^2 = r^2 + a^2$, and $\beta^2 = r^2 + b^2$ (by orthogonality).
2. $\sigma \geq x - a + c$ (by the fact that that the distance between ball B_i and ball B_j should be no more than the distance between ball B_j and ball B_k).
3. $\delta \geq c - a$ (by the triangle inequality $\delta \geq \sigma - x$, and condition 2).
4. $\delta \geq x - b - c$ (since B_k is proper, there is another neighbor ball B_l of B_k so that B_i, B_l are disjoint, and so the distance between ball B_i and ball B_l is no less than the distance between ball B_i and ball B_j. If K is the intersection of balls B_k and B_l, then $\delta + c \geq |B_iK| \geq x - b$).

Given the above relations, we would like to derive that the ball B_k does not intersect ball O' more than orthogonally, i.e. $\gamma^2 \geq c^2 + r^2$.

From Equation (4.1) in the proof of Lemma 4.4.2, we have

$$\gamma^2 = \frac{\beta\delta^2 + \alpha\sigma^2}{x} - \alpha\beta\,.$$

We substitute $x - a + c$ for σ and the larger of $c - a$ or $x - b - c$ for δ to find a lower bound for γ^2.

1. When $c - a \geq x - b - c$, i.e. $c \geq (x + a - b)/2$, we obtain

$$\begin{aligned}
\gamma^2 &\geq \frac{\beta(c-a)^2 + \alpha(c+x-a)^2}{x} - \alpha\beta \\
&= c^2 + \frac{2c}{x}(-\beta a + \alpha(x-a)) + \frac{1}{x}(\beta a^2 + \alpha(x-a)^2) - \alpha\beta \\
&= c^2 + 2c(\alpha - a) + a^2 + \alpha x - 2\alpha a - \alpha\beta \\
&= c^2 + 2c(\alpha - a) + (\alpha - a)^2 \\
&\geq c^2 + (\alpha - a)(x + a - b) + (\alpha - a)^2 \\
&= c^2 + (\alpha - a)(\alpha + a) + (\alpha - a)(\beta - b) + (\alpha - a)^2 \\
&= c^2 + r^2 + (\alpha - a)(\beta - b) + (\alpha - a)^2 \\
&\geq c^2 + r^2.
\end{aligned}$$

2. When $c < (x + a - b)/2$, similarly, we get

$$\gamma^2 \geq \frac{\beta(x-b-c)^2 + \alpha(x-a+c)^2}{x} - \alpha\beta\,.$$

Manipulating the right hand side of the above inequality, we obtain:

$$\gamma^2 \geq c^2 + \tfrac{2c}{x}(\alpha(\alpha - a) - \beta(\beta - b)) + (\alpha - a)^2 + (\beta - b)^2 + r^2\,.$$

If $\alpha(\alpha - a) \geq \beta(\beta - b)$, it is clear that $\gamma^2 \geq c^2 + r^2$. If not, using $c > (x + a - b)/2$, we have that

$$\begin{aligned}
\gamma^2 > \;& c^2 + r^2 + \frac{1}{\alpha + \beta}((\alpha + a) + (\beta - b))(\alpha(\alpha - a) - \beta(\beta - b)) \\
& + (\alpha - a)^2 + (\beta - b)^2\,.
\end{aligned}$$

Simplifying the right hand side of the above inequality, we obtain

$$\gamma^2 \quad > \quad c^2 + r^2 + (\alpha - a)^2 + (\beta - b)(\alpha - a).$$

In all cases, $\gamma^2 \geq c^2 + r^2$. □

Theorem 4.5.2 gives us a way to check self-collision for a necklace: we can first compute the power diagram of all the beads and then check every pair for each power diagram edge. In addition, we check those pairs which share common neighbors, i.e. pairs (B_i, B_{i+2}) for $1 \leq i < n - 1$, and those pairs involving B_2 or B_{n-1} (the only beads having an improper neighbor in the necklace). Clearly, the number of additional pairs is $O(n)$, and the cost of the checking is dominated by the complexity of power diagram, which we expect to be linear for a densely packed necklace.

Under the KDS motion model, it is easy to maintain the power diagram [GXZ01]. For the discrete time step model, however, it can be too expensive to recompute the power diagram at each time step. However, we can use a variety of techniques to update the diagram, similar to what was done in [GR04] to update the Delaunay triangulation. A simple and fast solution is to remove some subset of the balls such that the power diagrams of the remaining balls before and after the time step are combinatorially identical, move the balls to the final locations, and then reinsert the removed balls. Another way is to convert a discrete time step into a continuous one by inventing an artificial motion which interpolates between the initial and final locations of the balls and then apply the KDS technology. We are allowed to pick whatever motion so that the certificate failure times are easy to compute, or the number of events is small. For example, we can continuously change the size of beads so that each bead moves on a straight line in the standard lifting space.

Chapter 5

Discrete Center Hierarchies and Applications

5.1 Introduction

Bounding volume hierarchies for necklaces described in the previous chapter are stable under motion, and when the hierarchies become invalid, we can often update them quickly. Unfortunately, the cost of updating the hierarchies in the worst case is high, taking $O(n \log n)$ time. The work in this chapter is an attempt to address that problem, introducing a new type of bounding volume hierarchies that can be efficiently updated when the underlying points move.

Given a set S of points, a *set of discrete centers* of S with a separation radius r is a sampling of S such that the distance between any pair of samples is more than r and any point in S is within a distance of r from some sample. A *discrete center hierarchy* (DCH) on S is simply a hierarchy of discrete center sets where the bottom level of the hierarchy is S itself, and each level is a discrete center set of the level below, with the separation radius increasing exponentially with respect to the level number.

By associating with each discrete center in each level it participates in a ball with radius proportional to the minimum separation between the centers in that level, the discrete center hierarchies implicitly define bounding volume hierarchies. We call those balls *implicit balls*. We study this structure and its various applications. We show that the DCH is maintainable when the underlying points in S move in KDS setting. The DCH can also be maintained in blackbox motion model, making it the first non-trivial data structure that does so.

Given an implicit bounding volume hierarchy associated with a DCH, we can run the self-collision detection algorithm on that bounding volume hierarchy. This effectively gives the pairs of intersecting balls at the bottom level of the bounding volume hierarchy, and thus all pairs of points in S within some threshold of each other. The set of all pairs of intersecting balls found in the process form a graph called *conflict graph* of that bounding volume hierarchy.

We show that the discrete center hierarchies together with their conflict graphs enable many data structures for proximity-related problems that are maintainable under motion. For example, we can maintain the closest pair of points and thus have a collision detection mechanism. We can maintain the near neighbors of all points (to within a specified distance), and perform approximate nearest neighbor searches (or get the functionality of approximate Voronoi diagrams). We also get the first kinetic algorithms for maintaining well-separated pair decompositions, geometric spanners, and approximate k-center of our point set. So this one simple combinatorial structure provides a ‘one-stop shopping’ mechanism for a wide variety of proximity problems and queries on moving points.

5.2 Specific contributions

We now discuss in greater detail the specific problems we address and the contributions that the DCH data structure makes.

Maintainable bounding volume hierarchy. We show that by keeping the conflict graph as an auxiliary data structure, the DCH is maintainable when the points in S move, either in the KDS framework or in blackbox motion model.

Closest pair and collision detection. We show that there is always an edge between the closest pair of points in S in its conflict graph. The conflict graph thus naturally contains the information we need for closest pair maintenance and collision detection.

Compared to the structures in previous chapter and in [LSHL02], the conflict graph is much lighter weight. It is a purely combinatorial structure (edges, specified by pairs of points) of size $O(n/\varepsilon^d)$ that allows self-collision detection in $O(n)$ time.

All near neighbors search. The all near neighbors search problem is to find all the pairs of points with distance less than a given value r, i.e., for each point, we must return the list of points inside the ball with radius r. In physical simulations such search is often used to limit interactions to only pairs of elements that are sufficiently near each other. For example, most molecular dynamics (MD) systems maintain such 'neighbor-lists' for each atom and update them every few integration steps.

A typical MD algorithm performs this task by voxelizing space into tiles of size comparable to the size of a few atoms and tracks which atoms intersect which voxels. Since many voxels may be empty, a hash table is normally used to avoid huge voxel arrays. Atoms are reallocated to voxels after each time step. The simplicity of this method is appealing, but its performance is intimately tied to a prespecified interaction distance.

With the conflict graph, to find all the points within a certain distance r from a point p, we start from p and follow the spanner edges until the total length is greater than $s \cdot r$. We then filter the points thus collected and keep only those that are within distance r of p. We can show that the cost of this is $O(n + k)$, where k is the number of pairs in the answer set — thus the method is output sensitive and the cost of filtering does not dominate.

Well-separated pair decompositions. The concept of a well-separated pair decomposition for a set of points in $\mathbb{R}^d$ was first proposed by Callahan and Kosaraju [CK95a]. A pair of point sets (A, B) is *s-well-separated* if the distance[1] between A, B is at least s times the diameters of both A and B. A *well-separated pair decomposition* (WSPD) of a point set consists of a set of well-separated pairs that 'cover' all the pairs of distinct points, i.e. any two distinct points belong to the different sets of some pair of the decomposition. In [CK95a], Callahan and Kosaraju showed that for any point set in a Euclidean space and for any positive constant s, there always exists an s-well-separated pair decomposition with linearly many pairs. This fact has been very useful in obtaining nearly linear time algorithms for many problems such as computing k-nearest neighbors, n-body potential fields, geometric spanners and approximate minimum spanning trees [CK93, Cal93, CK95a, CK95b, AMS94, ADM$^+$95, NS00, LNS02, GLNS02, Eri02].

[1]The distance between two point sets A, B is defined as the minimum distance of two points p, q, with $p \in A$ and $q \in B$.

We show that the DCH and its conflict graph can be used to generate an s-well-separated pair decomposition, for any positive s — it suffices to take $\varepsilon = 4/s$ in the spanner construction. The size of the WSPD is linear, which matches the bound by Callahan and Kosaraju [CK95a]. Since the spanner can be maintained in dynamic and kinetic settings, the well-separated pair decomposition can also be maintained efficiently for a set of moving points.

Spanner. A subgraph G' is a *spanner* of a graph G if $\pi_{G'}(p, q) \leq s \cdot \pi_G(p, q)$ for some constant s and for all pairs of nodes p and q in G, where $\pi_G(p, q)$ denotes the shortest path distance between p and q in the graph G. The factor s is called the *stretch factor* of G' and the graph G' an s-spanner of G. If G is the complete graph of a set of n points S in a metric space $(S, |\cdot|)$ with $\pi_G(p, q) = |pq|$, we call G' an s-spanner of the metric $(S, |\cdot|)$. We will focus on collections of points in $\mathbb{R}^d$, in settings where proximity information among the points is important. Spanners are of interest in such situations because sparse spanners with stretch factor arbitrarily close to 1 exist and provide an efficient encoding of distance information. In particular, continuous proximity queries requiring a geometric search can be replaced by more efficient graph-based queries using spanners.

There is a vast literature on spanners that we will not attempt to review in any detail here. The readers are referred to a number of survey papers for background material [ADM$^+$95, Epp00, Pel00]. Extant spanner constructions are all static, based on sequential centralized algorithms. We show that the DCH structure also provides us a spanner data structure for points in a Euclidean space.

$(1 + \varepsilon)$-nearest neighbor query/approximate Voronoi diagram. The DCH structure we propose can be used to output the approximate nearest neighbor of any point $p \in \mathbb{R}^d$ with respect to the point set S, in time $O(\lg n/\varepsilon^d)$. There has been a lot of work on data structures to answer approximate nearest neighbor queries quickly [IM98, AM02, AMN$^+$98, AMM02]. However, they all try to minimize the storage or query cost and do not consider points in motion.

k-center. For a set S of points in $\mathbb{R}^d$, the k-center is a set K of points, $K \subseteq S$, $|K| = k$, such that $\max_{p \in S} \min_{q \in K} |pq|$ is minimized. The geometric k-center problem, where the points lie in the plane and the Euclidean metric is used, is approximable within 2, but is not approximable within 1.822 [FG88]. However, the usual algorithms to compute approximate k-center are of a greedy nature [FG88, Gon85], and thus not easy to kinetize. For the dual problem of k-center, i.e., minimizing the number of centers when the radius of each cluster is prespecified, efficient kinetic maintenance schemes are available [GGH$^+$03, Her03]. Here we show how to compute an 8-approximate k-center by using the conflict graph. Furthermore, we are first to give a kinetic approximate k-center as the points move.

5.2.1 Result summary

We summarize the results below. All the algorithms/data structures are deterministic and we consider the worst-case behavior. For a set S of n points in $\mathbb{R}^d$, let the *aspect ratio* α of S be the ratio of the maximum pairwise distance to the minimum pairwise distance between points in S. We have:

- An algorithm to construct a DCH and its conflict graph.
- An algorithm to insert points to and remove points from a DCH
- An algorithm to maintain a DCH and its conflict graph

Furthermore, we show that the above algorithm could be adopted to obtain the following data structures, all are maintainable when the underlying points move:

- An $O(n)$ structure for finding all near neighbors in time $O(k + n)$, where k is the size of the output;
- A $(1/\varepsilon)$-well-separated pair decomposition of size $O(n/\varepsilon^d)$;
- A $(1 + \varepsilon)$-spanner with $O(n/\varepsilon^d)$ edges, maximum degree bounded by $O(\lg \alpha/\varepsilon^d)$, and total weight bounded by $O(\text{MST} \cdot \lg \alpha/\varepsilon^{d+1})$, where MST is the weight of the minimum spanning tree of S;
- An $O(n)$ structure for $(1 + \varepsilon)$-nearest neighbor queries in $O(\lg \alpha/\varepsilon^d)$ time;
- An $O(n)$ data structure for closest pair and collision detection;
- An 8-approximate k-center, for any $0 < k \leq n$.

The KDS obtained all have the desirable kinetic properties of efficiency, compactness, locality, and responsiveness.

5.3 Discrete center hierarchies

In this chapter we focus on a set S of points in the Euclidean space $\mathbb{R}^d$. Without loss of generality, we assume that the closest pair of points has distance 1 so that the furthest pair of S has distance α, the aspect ratio of S.

5.3.1 Definition

A set of *discrete centers* with radius r of a given point set S is defined as a maximal subset $S' \subseteq S$ such that the balls with radius r centered at points in S' contain all points in S but any two points in S' are of a distance greater than r away from each other. The points in S' are also referred to as *centers*. From this definition, for each point p in S, there is a center q in S' such that the distance $|pq| \leq r$. We call q the parent of p, and p is a child of q. We also say that q covers p. When there are more than one centers in S' within a distance r from p, we pick the parent of p arbitrarily so that each point in S has exactly one parent in S'.

Given a set of points S, we can construct a sequence of sets $S_0, S_1, \cdots, S_{h-1}$ such that S_0 is the original point set S and S_{i+1} is a set of discrete centers of S_i with radius 2^i, for $i \geq 0$. The sequence terminates when there is only 1 point left, i.e. $|S_{h-1}| = 1 < |S_{h-2}|$.

The sequence of sets together with the parent-child relations from the discrete centers yield a tree structure on S that we call a *discrete center hierarchy* (DCH). We call h the height of the hierarchy and call the sets $S_i, 0 \leq i < h$ the levels of the hierarchy. We note that there are potentially many different discrete center hierarchies on the same set S.

Since a point $p \in S$ may appear in many levels of the discrete center hierarchy, when it is not clear, we use $p^{(i)}$ to denote the node p in level S_i. We denote $P(p^{(i)})$ the parent of $p^{(i)}$, and when the level of p is clear from the context, we simply denote its parent as $P(p)$. We denote by $C_{i-1}(p)$ the set of p's children in S_{i-1}. We call $P^{(i)}(p) \in S_i$ the ancestor of p. Note that if p is in S_i, then $p^{(i)}$ must be the parent of $p^{(i-1)}$, and thus p is the ancestor of itself in all levels $j \leq i$.

Since the distance between a node $p \in S_i$ and its parent is less than 2^i, if we associate with $p \in S_i$ a ball $B(p)$ with radius 2^i, its easy to see that the ball $B(p)$ is contained within $B(P(p))$. A discrete center hierarchy thus implicitly defines a bounding volume hierarchy

H of a collection of balls of radius 1 centered at points in S. Denote this collection of balls as $\mathcal{C}$. The ball $B(p)$ is called an *implicit ball* at p.

More generally, given a parameter $c \geq 1$, we can associate with each node $p \in S_i$ an implicit ball $B_c(p)$ with radius $c \cdot 2^i$. We denote $\mathcal{C}_c$ the collection of balls of radius c centered at points in S. The discrete center hierarchy then implicitly defines a bounding volume hierarchy H_c on $\mathcal{C}_c$. The bounding volume hierarchy H_1 is simply H.

Using the terminology in the previous chapter, the bounding volume hierarchy H_c is a layered hierarchy, i.e. the implicit ball at any node contains all the implicit balls at its children nodes. It is neither a tight nor a canonical bounding volume hierarchy, though as we will show later, H (or H_c) can be fully maintained under motion and self-collision detection using the DCH is optimal. We note that as c becomes large, the ball $B_c(p)$ becomes well inside $B_c(P(p))$, and intuitively, the implicit bounding volume hierarchy becomes more stable when the points in S are moving. We will make this intuition more precise and exploit this property later in subsection 5.5.3.

We let the *conflict graph* G_c be a multi-graph on S such that for each pair of intersecting implicit balls $B_c(p^{(i)})$ and $B_c(q^{(i)})$, $p \neq q$, there is an edge between p and q in G_c. We say that p and q are neighbors in level i of G_c, and the edge between them an edge in level i of G_c. We denote by $N_i(p)$ the set of neighbors of p in level i. If we wish to detect the self-collision of $\mathcal{C}_c$ using the bounding volume hierarchy H_c, i.e. finding out all pairs of intersecting balls among the $\mathcal{C}$, the edges in the conflict graph G_c are precisely the pairs of balls that we check for collision. The conflict graph G_c essentially encodes the proximity information between all points in S in a hierarchical fashion.

For simplicity, we write $G \equiv G_1$. Besides using G as a self-collision data structure for $\mathcal{C}$, we will show that the conflict graph G can be used as an auxiliary data structure to update the DCH when points are added or removed from S, and to maintain the DCH in the KDS framework. We can further use the graph G_c for $c > 1$ to maintain the DCH in the blackbox motion model.

5.3.2 Basic properties

We first prove some properties about the discrete center hierarchy and its conflict graph G_c.

Lemma 5.3.1.

1. *$S_i \subseteq S_{i-1}$.*
2. *For any two points p and $q \in S_i$, $|pq| > 2^{i-1}$.*
3. *G_c has an edge in level i between p and $q \in S_i$, iff $|pq| \leq c \cdot 2^{i+1}$.*
4. *If $q^{(i)} \in C_i(p^{(i+1)})$ and $q \neq p$, then $q^{(i)} \in N_i(p^{(i)})$, i.e. there is an edge in G_c from each point q to its parent in the DCH.*
5. *p and q are neighbors only if they have the same parent or $P(p)$ and $P(q)$ are neighbors.*
6. *The hierarchy has at least $\lfloor \lg \alpha \rfloor + 1$ levels and at most $\lceil \lg \alpha \rceil + 2$ levels.*

Proof: The first 3 claims follow from the definition of the DCH and the conflict graph G_c. The fourth claim follows from the fact that if $p^{(i)}$ is a child of $q^{(i+1)}$, $|pq| \leq 2^i < c \cdot 2^{i+1}$. The fifth claims is true because the implicit ball at p (or respectively q) is completely contained in the implicit ball at $P(p)$ (or $P(q)$ respectively). The last claim follows from the fact that the top level implicit ball has diameter $2^h > \alpha$ and that points in level $h-2$ are more than 2^{h-3} apart which implies $\alpha > 2^{h-3}$. □

5.3.3 Size of the conflict graph

The DCH clearly has linear size. We show that the conflict graph G_c also has linear size. We first prove a simple result that is used repeatedly in this chapter.

Lemma 5.3.2 (Packing Lemma). *If all points in a set $U \subset \mathbb{R}^d$ are of at least a distance r away from each other, then there are at most $(2R/r+1)^d$ points in U within any ball X of radius R.*

Proof: Let X' be a ball co-centric with X with radius $R + r/2$. The balls of radius $r/2$ centered at points of U inside X are all disjoint and are inside X'. By a volume argument, there can be at most $((R + r/2)/(r/2))^d = (2R/r + 1)^d$ such balls. □

Using Lemma 5.3.2, we have:

Lemma 5.3.3. *Each point in S_i covers at most 5^d points in S_{i-1}.*

Note that the bound in Lemma 5.3.3 and Lemma 5.3.4 can be improved. A more careful analysis, e.g., by Sullivan [Sul94], shows that the maximum number of points in S_{i-1} covered by a point in S_i is 19 in $\mathbb{R}^2$; 87 in $\mathbb{R}^3$; and $O(2.641^d)$ in dimension d.

From Lemma 5.3.2, we also have:

Lemma 5.3.4. *A point $p \in S_i$ has fewer than $(8c+1)^d - 1$ edges with other points of S_i in G_c.*

Lemma 5.3.5. *The maximum degree of G_c is $((8c+1)^d - 1)(\lceil \lg \alpha \rceil + 2)$.*

Proof: It follows from Lemma 5.3.4 and Lemma 5.3.1 (6). $\square$

Lemma 5.3.6. *There are less than $2(8c+1)^d \cdot n$ edges in G.*

Proof: Note that if p is a point in a DCH that does not have any children and p has k edges in the conflict graph G, removing p from the DCH gives another DCH with one less vertex, and the conflict graph G' corresponding with the new DCH has k less edges. The lemma follows if we can show that we can always find a childless point p in the DCH that is incident to at most $2(8c+1)^d$ edges.

Let p and q be the closest pair of points in G, and let k be such that $2^k \geq |pq| > 2^{k-1}$. Since $|pq| \leq 2^k$, p and q cannot be both in level S_{k+1}. Without lost of generality, assume that p is not in S_{k+1}. As p and q is the nearest pair of points, all other points are more than 2^{k-1} from p. By Lemma 5.3.2, p is incident on at most $(1 + 8c/2^j)^d - 1$ edges in level S_{k-j} for each $0 \leq j \leq log_2 c$, and thus at most $\sum_{j=0}^{log_2 c}((1 + 8c/2^j)^d - 1) \leq 2(8c+1)^d$. $\square$

5.4 Construction and dynamic maintenance

In this section we show that we can efficiently construct the DCH and its conflict graph G_c in $O(n \lg \alpha)$ time, where n is the number of points and α is the aspect ratio of the point set. We also show that we can dynamically insert or remove a point from the DCH and its associated conflict graph at a cost of $O(\lg \alpha)$ for each operation. In realistic settings where α is a polynomial function of n, the construction of the hierarchy is $O(n \lg n)$, and dynamic update operations are done in $O(\lg n)$ time each.

To describe the construction and maintenance of the DCH, we adopt a slightly different scenario. We assume that the aspect ratio α is always bounded by a polynomial of the number of points. However, as the points are inserted and deleted, the minimum separation of the point set may change. To address this, we imagine that we virtually keep sets of points S_i for $-\infty < i < \infty$, such that S_i is a set of discrete centers of S_{i-1} with radius 2^i. Since the aspect ratio is bounded, there exist m and M such that $S_i = S_m$ for all $i \leq m$ and $S_i = S_M$ for all $i \geq M$. We refer to S_m and S_M the bottom and the top levels of the DCH respectively. The set S_M is a singleton, and the node in S_M is the root of the DCH. For each point p other than the root of the hierarchy, we store the maximum number M_p such that level S_{M_p} contains p, and store its parent $P(p^{(M_p)})$. We also store the minimum number m_p such that p has a neighbor in S_{m_p}, non-empty lists of neighbors of p in each of the levels between S_{m_p} and S_{M_p}, and non-empty lists of children of p in all levels below S_{M_p}. We also store the value of m and M for the hierarchy. Notice that we can always scale the point set so that the minimum separation is 1 and thus return to the previous setup.

5.4.1 Construction

We construct the hierarchy incrementally by inserting points one by one. Suppose that we already have a hierarchy of $n-1$ points. To insert the n-th point p, we find a parent of p by computing the lowest level S_i and a point q in that level such that $|pq| < 2^{i-1}$.

Observe that p is inside an implicit ball $B(q)$ in a DCH only if it is inside the implicit ball $B(P(q))$. We can traverse down the hierarchy and find in each level S_i all implicit balls containing p, and among them, all potential parents of p if p is in level S_{i-1}. We can then select among all potential parents a parent for p and insert p as a child of that node.

In each level S_i, the points are 2^{i-1} apart, and thus by Lemma 5.3.2, there are at most 3^d implicit balls containing any given point p. The cost of inserting p is thus at most $3^d \lg \alpha$, and the cost of computing the DCH by incremental insertion is $O(3^d n \lg \alpha)$.

We note that once the DCH is computed, we can traverse it from top down to compute its conflict graph. The cost of the computation of the conflict graph is proportional to its size, and thus we can additionally compute the conflict graph G_c in linear time.

5.4.2 Dynamic updates

From the previous subsection, it is clear that we can insert points into a DCH at the cost $O(\lg \alpha)$ each. The removal of points from a DCH is more complicated, and we need a supporting data structure to make the removal possible. This supporting structure is the conflict graph G of the DCH. In this section, we show that the DCH and its conflict graph G can be updated when points are inserted and deleted, costing $O(\lg \alpha)$ per point.

It is clear that once a point p has been inserted into a DCH, the edges incident on p in the conflict graph can be computed simply by traversing down the DCH. This is the case due to a simple observation that any given level S_i, the ball $B(p^{(i)})$ can only intersect with ball $B(q^{(i)})$ if $B(P(p^{(i)}))$ intersects $B(P(q^{(i)}))$. The cost incurs in each level is $O(1)$, and thus the total cost of updating the conflict graph is $O(\lg \alpha)$.

To remove a point p from a DCH, we remove p level by level from bottom up. In level i, if p has no children (except itself), we can simply remove p. The conflict graph can then be updated by removing all edges incident on p. If p has children, its children would become *orphans*, and we need to find new parents for them before we can remove p. We assume $q^{(i-1)}$ is a child of $p^{(i)}$, $q \neq p$. From (4) in Lemma 5.3.1, $q^{(i-1)}$ is a neighbor of $p^{(i-1)}$ on level S_{i-1}.

From (5) in Lemma 5.3.1, we know the parent of $q^{(i-1)}$ must be a neighbor of the parent of $p^{(i-1)}$, i.e., $p^{(i)}$. If there is a neighbor w of $p^{(i)}$ that covers $q^{(i-1)}$, we set $q^{(i-1)}$'s parent to be w and we are done. If $q^{(i-1)}$ is not covered by any centers on level S_i, it must be inserted into S_i. Notice that the newly inserted node $q^{(i)}$ must be a neighbor of $p^{(i)}$, we can recursively either find a parent for $q^{(i)}$ or promote q further up. The neighbors of q can then be computed from top down in a way similar to point insertion in the previous subsection.

Note that the cost of raising a child of p up one level is $O(1)$, and as the child may end up in the top level, the cost of fixing a child of p is $O(\lg \alpha)$. Since p could appear in $O(\lg \alpha)$ levels and has $O(\lg \alpha)$ children, a trivial bound on the cost of removing p is $O(\lg^2 \alpha)$. However notice that for any level S_{i-1}, all children of p on or below the level are inside a disk of radius 2^{i-1}, and the minimum separation in S_i is 2^{i-2}. By Lemma 5.3.2, at most $O(1)$ points among p's children can end up being in S_i. The total number of times all children of p are raised up one level is $O(\lg \alpha)$, and thus the cost of removing p is $O(\lg \alpha)$.

Once the DCH has been updated, the edges associated with p can be removed. The edges incident on the children being raised can be computed at the cost of $O(1)$ per child per raise, and thus the total cost of updating the conflict graph is also $O(\lg \alpha)$.

Theorem 5.4.1. *Dynamic insertion and deletion of points in a DCH and its conflict graph G take $O(\lg \alpha)$ each, where α is the aspect ratio. The DCH can be constructed in time $O(n \lg \alpha)$. Given a DCH, its conflict graph G can be computed in $O(n)$ additional time.*

5.5 Maintenance under motion

We analyze the maintenance of the DCH in the kinetic data structure (KDS) framework [BGH99, Gui98]. A KDS tracks an attribute of a moving system over time by maintaining a set of certificates as a proof of attribute value correctness. In our case, we would like to maintain a set of certificates showing that the discrete centers hierarchy is valid. When the points move, the certificates may become invalid, at which time we need to update the certificate set and possibly the DCH. We show that our scheme to maintain the DCH has all properties of a good KDS, and will also show that the DCH can be maintained in a more practical black-box motion framework.

5.5.1 KDS maintenance

To maintain the DCH in the KDS framework, we again use its conflict graph G as an auxiliary data structure. We maintain both the DCH and the conflict graph and use the conflict graph to help repairing the DCH when the DCH becomes invalid. We use four kinds of certificates for this purpose, the first two to certify the validity of the DCH, and the last two to certify the validity of the conflict graph.

1. A *parent-child certificate* certifies that a node p in level S_i is within distance 2^i of $P(p)$.

2. A *separation certificate* certifies that nodes q and p in level S_i is at least 2^{i-1} away from each other.

3. An *edge certificate* certifies that neighbor nodes p and q in level S_i are near each other, that the implicit balls at p and q intersect.

4. A *potential edge certificate* certifies that non-neighbor nodes p and q in level S_i are far from each other, i.e. the implicit balls in level i at p and q do not intersect.

All certificates are simple distance comparisons among pairs of points. There are clearly $O(n)$ parent-child and edge certificates. Note p and q in level S_i cannot become within 2^{i-1} unless they are already neighbors in S_i, we only need to keep separation certificates between neighboring pairs of nodes, and thus we only have only $O(n)$ separation certificates.

Similarly, note that the implicit balls at p and q in level S_i cannot intersect unless the implicit balls at $P(p)$ and $P(q)$ intersect, we only need to keep potential edge certificates between pairs of *potential neighbors*, nodes that are not neighbors but their parents are neighbors. Using an argument similar to that in Lemma 5.3.6, we can show that the number of potential certificates is also linear.

The failure of the four types of certificates generates four types of events, which are discussed separately as follows.

1. **Addition of an edge.** When a potential edge certificate fails, i.e., when the implicit balls at p and q in level S_i intersect, we add an edge between p and q, making q a neighbor of p. We also update the list of potential neighbors of the children of p and q.

2. **Deletion of an edge.** When an edge certificate fails, i.e., when the implicit balls at p and q in level S_i no longer intersect, we simply remove the edge between p and q. We also update the list of potential neighbors of the children of p and q.

3. **Promotion of a node.** When a parent-child certificate fails, i.e., $q = P(p)$ no longer covers $p \in S_i$, $|pq| > 2^i$. p becomes an orphan, and we need to find a new parent for p or promote it into higher levels. We deal with orphans in the same way as in dynamic updates. We then update the potential neighbors of p in level i and above.

4. **Demotion of a node.** When a separation certificate fails, i.e., a neighbor q of p in level S_i comes within a distance of 2^{i-1}, we need to remove one of the two points from level i. Assume without lost of generality that p is not in level $i+1$. We *demote* p, i.e., removing p from level i. Each former child t of p in level $i-1$ becomes an orphan, and we deal with each of them as in the previous event.

5.5.2 Quality of the kinetic discrete center hierarchy

There are four desirable properties that a good KDS should have [BGH99, Gui98]: (i) compactness: the KDS has a small number of certificates; (ii) responsiveness: when a certificate fails, the KDS can be updated quickly; (iii) locality: each point participates in a small number of certificates so that when the motion plan of that point changes, the KDS can be updated quickly; (iv) efficiency: there are not too many certificate failures compared with the number of combinatorial changes of the attribute being tracked in the worst case.

We show that the kinetic DCH has all desirable properties of a good KDS. As shown in the previous subsection, the total number of certificates in the kinetic DCH is linear, and thus the kinetic DCH is compact.

When an edge certificate or a potential edge certificate fails, we need to remove or add an edge in the conflict graph, and thus the cost of repairing such certificate failure is clearly $O(1)$. When a parent-child certificate or a separation certificate fails, we need to demote or promote a node, and the cost of such repair is $O(\lg \alpha)$. If the spread of the point set is assumed to be polynomial in the number of points n, then the repair cost for any certificate failure is at most $O(\lg \alpha) = O(\lg n)$. The kinetic DCH thus can be updated quickly when a certificate fails, i.e. it is responsive.

As the degree of each node in the conflict graph is $O(\lg \alpha)$, it is easy to see that each node is involved in $O(\lg \alpha)$ edge certificates, $O(\lg \alpha)$ potential edge certificates, $O(\lg \alpha)$ separation certificates, and at most 1 parent-child certificates. Thus, when the motion plan of a node changes, we can update the failure time of all certificates involving that node in time $O(\lg \alpha)$. If we assume that the spread of the point set is polynomial in n, the cost of updating the motion plan is $O(\lg n)$, and thus the kinetic DCH is local.

To show that the kinetic DCH is efficient, we first observe that each certificate is between two points, and when it starts to fail, the distance between those 2 points is an integral power of 2. Thus each pair of points can generate at most $O(\lg \alpha)$ certificate failures, and the total number of certificate failures in a kinetic DCH is at most $O(n^2 \lg \alpha)$. We now show a case when any DCH must change at least $\Omega(n^2)$ times.

We consider a necklace of balls in the plane. The necklace consists of three segments. For the two segments close to the ends, each contains n/c bumps, where each bump has height c and the distance along the necklace between adjacent bumps is $2c$. The two segments are connected by a bent segment with $2n$ balls. The total number of balls in the necklace is $10n$. The top segment with bumps is moving linearly towards the left; the bottom segment remains static. The balls on the middle segment move accordingly to keep the necklace connected. Figure 5.1 shows the configuration of the necklace at the starting and ending point.

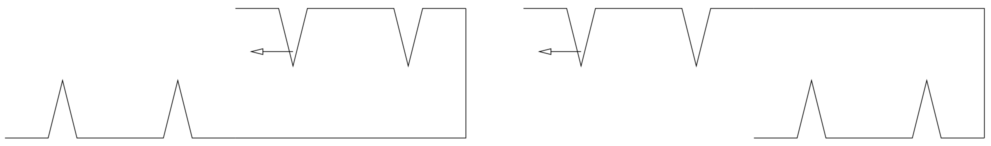

Figure 5.1: Motion of the points.

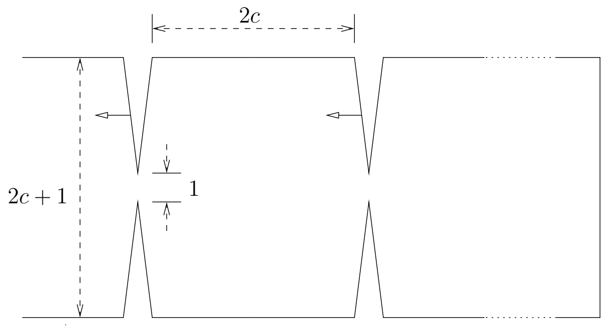

Figure 5.2: There are at least $\Omega(n^2)$ changes to any discrete center hierarchy during the motion.

Consider the time when a top bump is directly above a bottom bump, so that the distance between their peaks is 1. See Figure 5.2. Let p and q be the nodes at the peaks such that $|pq| = 1$. If we choose c such that $c > 2$, p and q are far from all other points, and as the result, either p is the parent of q or q is the parent of p in any discrete center hierarchy of S. Without lost of generality, let p be the parent of q. Some time later during the motion, the distance between the points p and q becomes 2 or more, and thus q is either promoted or q changes its parent. Thus there must be at least one event for the pair of points p and q, and any kinetic DCH for this necklace must have $\Omega(n^2)$ events.

We have established:

Theorem 5.5.1. *The kinetic DCH is efficient, responsive, local and compact. Specifically, the total number of events in maintaining a kinetic DCH is $O(n^2 \lg \alpha)$ under pseudo-algebraic motion. Each event can be updated in $O(\lg \alpha)$ time. A flight-plan change can be handled in $O(\lg \alpha)$ time. Each point is involved in at most $O(\lg \alpha)$ certificates. When the aspect ratio of the point set is bounded, the $\lg \alpha$ in the above formulas can be replaced by $\lg n$.*

5.5.3 Maintenance in a physical simulation setting

In practice, the motion of the points may not be known in advance. Instead, the new point positions are given by some physics black box after every time step, and we are called in to repair the DCH. When the points move randomly in a step, the best option is to rebuild the DCH from scratch. If the points do not move much, however, we can attempt to modify the DCH in the previous step to get a DCH for the points in the next step.

We first verify and update the hierarchy from the top down, using update operations as in the KDS setting. Suppose that we have updated all levels above level i and we would like to update level i. First we verify that all centers in level i are still covered by their parent centers in level $i+1$. For each center in level i that became an orphan, we find a new parent for it. Then for each pair of neighbors in level i that are closer than 2^i, we demote one of the two centers and find new parents for the orphans. Edges of the conflict graph G_c in level i are then verified and updated if necessary.

To show that the hierarchy is correct after the update, we need the following lemma which extends Lemma 5.3.1(5).

Lemma 5.5.2. *Let p, q be centers in level i and $r = P(p), s = P(q)$ in a time frame. If*

p, q, r, s do not move more than $(c-1) \cdot 2^{i-1}$ in each time step, then in the next time frame p and q are neighbors in G_c only if $r = s$ or r and s are neighbors in G_c.

Proof: Let p_1, q_1, r_1, s_1 and p_2, q_2, r_2, s_2 be the positions of the centers in the two time frames respectively. If p_2 and q_2 are neighbors, then $|r_2s_2| \leq |r_2r_1| + |r_1p_1| + |p_1p_2| + |p_2q_2| + |q_2q_1| + |q_1s_1| + |s_1s_2| \leq 4(c-1) \cdot 2^{i-1} + 2 \cdot 2^i + c \cdot 2^{i+1} = c \cdot 2^{i+2}$, and thus $r_2 = s_2$ or r_2 and s_2 are neighbors. $\square$

Note that the cost of the update consists of the cost of traversing the hierarchy, the cost of verifying all edges, and the cost of fixing orphans. The cost of traversing and the cost of verifying all edges is proportional to the number of edges, which is $O(n)$. The total cost of the update is thus $O(n + k \lg \alpha)$ where k is the number of changes to the hierarchy.

If we know more about the motions of the points, for example a bound on the velocity or acceleration of the points, we can compute conservative bounds on the failure times of the certificates. These conservative bounds can be used to avoid verifying all edges and thus to lower the cost of the DCH maintenance.

5.6 Applications

5.6.1 Spanner

A subgraph G' is a *spanner* of a graph G if $\pi_{G'}(p, q) \leq s \cdot \pi_G(p, q)$ for some constant s and for all pairs of nodes p and q in G, where $\pi_G(p, q)$ denotes the shortest path distance between p and q in the graph G. The factor s is called the *stretch factor* of G' and the graph G' an s-spanner of G. If G is the complete graph of a set of n points S in a metric space $(S, |\cdot|)$ with $\pi_G(p, q) = |pq|$, we call G' an s-spanner of the metric $(S, |\cdot|)$.

In this subsection, we show that the conflict graph G_c is a $(1+\epsilon)$-spanner where $\epsilon = 4/(c-1)$. As the conflict graph G_c can be maintained under dynamic insertion and deletion and in the KDS setting, this $(1+\epsilon)$-spanner is maintainable in both dynamic and kinetic change. We note that existing spanner constructions are all static and based on sequential centralized algorithms, and the update of those spanners when the underlying points move, added, or deleted is often infeasible.

Theorem 5.6.1. *G_c is a $(1+\varepsilon)$-spanner where $\epsilon = 4/(c-1)$.*

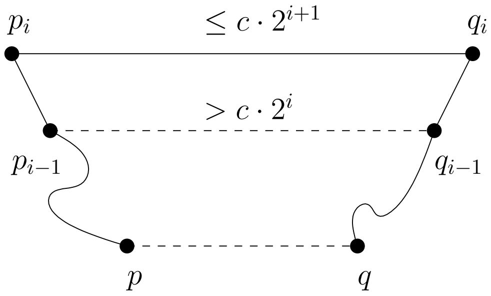

Figure 5.3: There exists a path in G between any two points p and q with length at most $(1+\varepsilon)|pq|$.

Proof: For a pair of points $p, q \in S_0$ we find the smallest level i so that there is an edge between their ancestors $P^{(i)}(p)$ and $P^{(i)}(q)$. Define $p_i = P^{(i)}(p), q_i = P^{(i)}(q), p_{i-1} = P^{(i-1)}(p), q_{i-1} = P^{(i-1)}(q)$. We take the path $\Lambda(p,q)$ as the concatenation of the parent chain from p to p_i, the edge between p_i, q_i and the parent chain between q_i and q. To prove that G is a spanner, we show that the path $\Lambda(p,q)$ has length at most $(1+\varepsilon)|pq|$.

First, we have that $|p_iq_i| \leq c \cdot 2^{i+1}$ and $|p_{i-1}q_{i-1}| > c \cdot 2^i$. By Lemma 5.3.1, $|pp_{i-1}| \leq 2^{i-1}$, $|qq_{i-1}| \leq 2^{i-1}$. So $|pq| \geq |p_{i-1}q_{i-1}| - |pp_{i-1}| - |qq_{i-1}| > (c-1) \cdot 2^i$. Also the length of $\Lambda(p,q)$ is at most $2^i + |p_iq_i| + 2^i \leq |pq| + 4 \cdot 2^i \leq |pq| + 4|pq|/(c-1) = (1+\varepsilon)|pq|$. This proves that G is a $(1+\varepsilon)$-spanner. $\square$

Given a graph on S, the *weight* of that graph is the total length of all edges of that graph. The minimum spanning tree (MST) of S is a connected graph on S having the smallest weight. We show that the spanner G_c is a graph with small weight.

Lemma 5.6.2. *The total weight of the conflict graph G_c is $O(MST \cdot \lg \alpha / \varepsilon^{d+1})$.*

Proof: First we bound the total lengths of all edges in a certain level S_i. We charge the edges in S_i to the minimum spanning tree (MST) of S_i as follows. An edge incident on $p^{(i)}$ is charged to one of the MST edges incident on $p^{(i)}$. Since each node in S_i has at most $(8c+1)^d - 1 = O(1/\varepsilon^d)$ edges with other nodes in S_i, each edge of the MST is charged at most $O(1/\varepsilon^d)$ times. Since the edges of G_c in level S_i have lengths at most $c \cdot 2^{i+1}$

and the points of S_i are at least 2^{i-1} away from each other, the length of an edge in S_i is at most $4c = O(1/\varepsilon)$ times the length of the charged edge in the MST of S_i. Thus the total lengths of the edges in S_i is at most $O(1/\varepsilon^{d+1} \cdot \text{MST}(S_i))$. The weight of the MST of S_i is at most twice the weight of the minimum Steiner Tree[2] (MStT) of S_i [Vaz01]. Furthermore if a point is added, the weight of the MStT can only become larger. Thus we have $\text{MST}(S_i) \leq 2\text{MStT}(S_i) \leq 2\text{MStT}(S) \leq 2\text{MST}(S)$. It follows that the total weight of the G_c is at most $O(\text{MST} \cdot \lg \alpha/\varepsilon^{d+1})$. □

To summarize, we have,

Theorem 5.6.3. *For a set of n points in $\mathbb{R}^d$ with aspect ratio α, we can construct a $(1+\varepsilon)$-spanner G_c so that the total number of edges in G_c is $O(n/\varepsilon^d)$, the maximum degree of G_c is $O(\lg \alpha/\varepsilon^d)$, and the total weight of G_c is $O(\lg \alpha/\varepsilon^{d+1} \cdot MST)$. Points can be added or removed from G_c in time $O(\lg \alpha)$ each. Furthermore, G_c can be maintained when the underlying points move, either in the KDS or blackbox motion model. The KDS spanner is efficient, local, compact, and responsive.*

5.6.2 Well-separated pair decomposition

A pair of point sets (A, B) is *s-well-separated* if the distance[3] between A, B is at least s times the diameters of both A and B. A *well-separated pair decomposition* (WSPD) of a point set consists of a set of well-separated pairs that 'cover' all the pairs of distinct points, i.e. any two distinct points belong to the different sets of some pair of the decomposition. In [CK95a], Callahan and Kosaraju showed that for any point set in a Euclidean space and for any positive constant s, there always exists an s-well-separated pair decomposition with linearly many pairs. This fact has been very useful in obtaining nearly linear time algorithms for many problems such as computing k-nearest neighbors, n-body potential fields, geometric spanners and approximate minimum spanning trees [CK93, Cal93, CK95a, CK95b, AMS94, ADM$^+$95, NS00, LNS02, GLNS02, Eri02].

[2]For a set of points S in the plane, the minimum Steiner tree is a tree in the plane with minimum total weight that connects all the points in S.

[3]The distance between two point sets A, B is defined as the minimum distance of two points p, q, with $p \in A$ and $q \in B$.

In this subsection, we show that the DCH induces a linear size well-separated pair decomposition. The maintainability of the DCH under dynamic insertion and deletion and under motion can be easily extended to show that the new well-separated pair decomposition is also maintainable.

Lemma 5.6.4. *The spanner can be turned into an s-well-separated pair decomposition, so that $s = c - 1 = 1/\varepsilon$. The size of this well-separated pair decomposition is $O(n/\varepsilon^d)$.*

Proof: For nodes $p^{(i)}$ and $q^{(i)}$ in the DCH, we denote by P_i and Q_i, respectively, the sets of all decedents of $p^{(i)}$ and $q^{(i)}$ including $p^{(i)}$ and $q^{(i)}$. Consider the set C of pairs (P_i, Q_i) where $p^{(i)}$ and $q^{(i)}$ are not neighbors in level i, but their parents in level $i+1$ are neighbors. We now argue that C is an s-well-separated pair decomposition with $s = c - 1 = 1/\varepsilon$.

Note that all points in P_i (or Q_i) are within a distance of 2^i from p_i (or q_i), and thus the diameter of P_i (or Q_i) is at most 2^{i+1}. Since $p^{(i)}$ and $q^{(i)}$ are not neighbors in S_i, $|pq| > c \cdot 2^{i+1}$, and thus the distance between P_i and Q_i is at least $(c-1)2^{i+1}$. It follows that P_i and Q_i are s-separated, where $s = c - 1$. On the other hand, for each pair of points in S, there is only one level i in the hierarchy such that their ancestors on level i is connected by an edge but their ancestors on level $i+1$ are not connected by an edge. Thus any pair of points is covered by exactly one pair (P_i, Q_i) in C. This shows that C is an s-well-separated pair decomposition.

By Lemma 5.3.3, each point covers at most 5^d points in one level below. The number of well-separated pairs in C equals to the number of non-connected cousin pairs in the DCH, which is at most a factor of 5^{2d} the number of edges in conflict graph G_c. Thus C has size $O(n/\varepsilon^d)$. □

Theorem 5.6.5. *The s-well-separated pair decomposition can be maintained by a KDS which is efficient, responsive, local and compact.*

Proof: We construct and maintain the conflict graph G_c. By using G_c as a supporting data structure, we maintain the well-separated pair decomposition implicitly by marking the pairs (P_i, Q_i) where $p^{(i)}$ and $q^{(i)}$ are not connected by an edge in some level i, but their parents in level $i+1$ are connected by an edge. They only change when the edges are inserted/deleted. So the total number of events is $O(n^2 \lg \alpha)$, as per Theorem 5.5.1. Upon request, a well-separated pair can be output in time proportional to the number of points it covers.

On the other hand, there exists a set of n points such that any linear-size c-well-separated pair decomposition has to change $\Omega(n^2/c^2)$ times. For the setting in Figure 5.2, there must be a well-separated pair that contains only the points of the upper bump and lower bump. The total number of such pairs is $\Omega(n^2/c^2)$, so is the total number of changes. $\square$

5.6.3 All near neighbors query

The near neighbors query for a set of points, i.e., for each point p, returning all the points within distance ℓ from p, has been studied extensively in computational geometry. A number of papers use spanners and their variants to answer near neighbors query in almost linear time [AS97, DDS92, LS95, Sal92]. Specifically, on a spanner, we do a breadth-first search starting at p until the graph distance to p is greater than $s \cdot \ell$, where s is the stretch factor. Due to the spanning property, this guarantees that we find all the points within distance ℓ from p. Furthermore, we only check the pairs with distance at most $s \cdot \ell$. Notice that unlike the previous papers that focus only on static points, the conflict graph can be maintained under motion, so the near neighbors query can be answered at any time during the movement of the points.

Before we bound the query cost of the algorithm, we first show that the number of pairs within distance $s \cdot \ell$ will not differ significantly with the number of pairs within distance ℓ. A similar result has been proved in [Sal92]. The following theorem is more general with slightly better results and the proof is much simpler.

Theorem 5.6.6. *For a set S of points in $\mathbb{R}^d$, denote by $\chi(\ell)$ the number of ordered pairs (p, q), $p, q \in S$ such that $|pq| \le \ell$, then $\chi(s \cdot \ell) \le (2(2s+3)^d + 1)\chi(\ell) + n(2s+3)^d$.*

Proof: We first select a set of discrete centers $S_{\ell/2}$ with radius $\ell/2$ from points S. We then assign a point q to a center p if $|pq| \le \ell/2$. A point can be within distance $\ell/2$ of more than 1 centers. In this case, we assign it to one of them arbitrarily. Any point is assigned to one and only one discrete center. We say q is covered by p if p is the assigned center for q. The set of points covered by $p \in S_{\ell/2}$ is denoted by $H(p)$. Define $k(p) = |H(p)|$.

Define the distance between two sets A, B of points as the minimum distance between two points in each set. We consider the set Ψ of ordered pairs $(H(p), H(q))$, $p \neq q$ such that the distance between $H(p)$ and $H(q)$ is at most $s \cdot \ell$. By triangular inequality, $|pq| \leq (s+1) \cdot \ell$. Using Lemma 5.3.2, the number of such q's that $(H(p), H(q)) \in \Psi$ is at most $(2s+3)^d$. Thus Ψ has at most $(2s+3)^d n$ ordered pairs.

An edge $p'q'$ is said to be covered by $(H(p), H(q))$ if $p' \in H(p)$ and $q' \in H(q)$. We note that any pair of points within the same $H(p)$ for any p are within a distance of ℓ of each other. Thus we have

$$\chi(\ell) \geq \sum_{p \in S_{\ell/2}} k(p)(k(p)-1). \tag{5.1}$$

Furthermore, any ordered pairs (p', q') with $\ell < |p'q'| \leq s \cdot \ell$ are covered by some pair $(H(p), H(q)) \in \Psi$. Thus we have

$$\chi(s \cdot \ell) - \chi(\ell) \leq \sum_{(H(p),H(q)) \in \Psi} k(p)k(q). \tag{5.2}$$

Using the inequality $ab \leq a(a-1) + b(b-1) + 1$ for all real numbers a and b, $k(p)k(q) \leq k(p)(k(p)-1) + k(q)(k(q)-1) + 1$. Summing this inequality over all pairs in Ψ, we obtain

$$\sum_{(H(p),H(q)) \in \Psi} k(p)k(q) \leq 2(2s+3)^d \sum_{p \in S_{\ell/2}} k(p)(k(p)-1) + n(2s+3)^d. \tag{5.3}$$

Combining (5.3) with inequalities (5.1) and (5.2), we have $\chi(s \cdot \ell) - \chi(\ell) \leq 2(2s+3)^d \chi(\ell) + n(2s+3)^d$, which implies $\chi(s \cdot \ell) \leq (2(2s+3)^d + 1)\chi(\ell) + n(2s+3)^d$. $\square$

Theorem 5.6.7. *For a set S of points in $\mathbb{R}^d$, we can organize the points into a structure of size $O(n)$ so that we can perform the near neighbors query, i.e., for each point p, find all the points within distance ℓ of p, in time $O(k+n)$, if the size of the output is k.*

Proof: As we described before, we traverse the s-spanner G_c by a breadth-first search and collect the pairs with distance at most $s \cdot \ell$ that include all pairs with distance no more than ℓ. We then filter out unnecessary pairs and only keep the pairs within distance ℓ. From Theorem 5.6.6, $\chi(s\ell) \leq (2(2s+3)^d + 1)k + n(2s+3)^d$, where k is the size of the output.

For a fixed s, G_c has linear size by Theorem 5.6.3, and the cost of traversing and filtering is $O(k + n)$. $\square$

We remark that this output sensitivity is not valid on a per point basis. Figure 5.4 shows an example situation where for point p the number of neighbors within distance $s \cdot \ell$ is not proportional to those within distance ℓ.

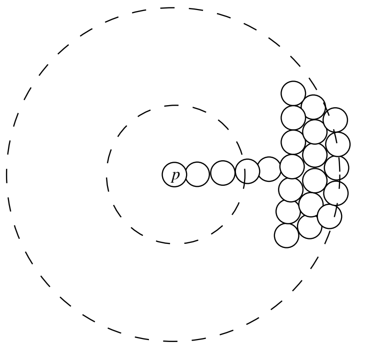

Figure 5.4: The number of neighbors of a point p increases abruptly.

5.6.4 $(1 + \varepsilon)$-nearest neighbor

An s-approximate nearest neighbor of a point $p \in \mathbb{R}^d$ with respect to a point set S is a point $q \in S$ such that $|pq| \le s \cdot |pq^*|$, where q^* is the nearest neighbor of p. We first show that for a given parameter $c > 1$, the conflict graph G_c can be used a data structure to maintain $(1 + \varepsilon)$-nearest neighbors of points in S, where $\varepsilon = 4/(c - 1)$.

Let p be a point in S, q is its nearest neighbor in G_c, and q^* be the nearest point in S to p. Clearly $|pq| \ge |pq^*|$. As G_c is a $(1 + \varepsilon)$-spanner, the shortest path between p and q has length at most $(1 + \varepsilon) \cdot |pq| \le (1 + \varepsilon) \cdot |pq^*|$. As q is the nearest neighbor of p, any path involved p must have length at least $|pq|$, and thus $|pq| \le (1 + \varepsilon) \cdot |pq^*|$, i.e. q is the an approximate nearest neighbor of q in S.

Suppose i is the lowest level where the edge pq appears, $c \cdot 2^{i-1} < |pq| \le c \cdot 2^i$. In any level $j \le i - 1$, there is no edge attached with point p as if there is such an edge, it would be shorter than pq. Thus to find the nearest neighbor q in G_c for a point $p \in S$, we simply find the lowest level i where p has an edge, and take q to be the nearest neighbor of p among all the level i edges incident on p. q is then also an $(1 + \varepsilon)$-approximate neighbor of p.

If we keep for each point its shortest edge in G_c, we can get the $(1+\varepsilon)$-nearest neighbor of any point $p \in S$ by a single lookup. The maintenance of the DCH and the conflict graph G_c implies the maintenance of the $(1+\varepsilon)$-nearest neighbor information as well. So we have,

Theorem 5.6.8. *For a set S of points in $\mathbb{R}^d$, we can maintain a kinetic data structure of size $O(n/\varepsilon^d)$ that keeps the $(1+\varepsilon)$-nearest neighbor in S of any node $p \in S$. The structure is efficient, responsive, local and compact.*

Proof: All we need to prove is the efficiency of the KDS. The example in subsection 5.5.2 shows that any linear-size structure maintaining the $(1+\varepsilon)$-approximate neighbor has to change $\Omega(n^2\varepsilon^2)$ times. $\square$

So far we use the conflict graph G_c as a data structure on some point set S to maintain the $(1+\varepsilon)$-approximate nearest neighbor query for a specific ε, and the query points must be in S. The approximate $(1+\varepsilon)$-nearest neighbor query can be done in a more general setting where the parameter ε is not given during the preprocessing, and the query points can be arbitrary. In this setting, we can still answer the query efficiently, though the query time is no longer $O(1)$ as in the previous case.

Theorem 5.6.9. *For a set S of points in $\mathbb{R}^d$, we can organize the n points into a structure of size $O(n)$ so that we can perform the $(1+\varepsilon)$-nearest neighbor query in $O(\lg \alpha/\varepsilon^d)$ time, i.e., given a point $p \in \mathbb{R}^d$, find a point q in S such that $|pq| \leq (1+\varepsilon)|pq^*|$, where q^* is the nearest neighbor of p.*

Proof: Given an $\varepsilon > 0$ and a query point p, we let $t = 1 + 2/\varepsilon$. To answer the $(1+\varepsilon)$ approximate nearest neighbor of p, we traverse the DCH top down and keep track of the set $W_i = \{q \mid q \in S_i, |pq| < t \cdot 2^i\}$ as the level i decreases.

First of all, we notice that $|W_i| = O(t^d)$ for any i, since the points in S_i are at least distance 2^{i-1} apart. Secondly, we observe that for a point $q \in S_i$, if $|pq| < t \cdot 2^i$, by triangular inequality $|pP(q)| \leq |pq| + |qP(q)| < t \cdot 2^i + 2^i \leq t \cdot 2^{i+1}$. Therefore W_i must be included in the set of the children of W_{i+1}. So we can construct W_i from W_{i+1} in $O(t^d)$ time, by checking the children of all the points in W_{i+1}. The total running time of such a traversal is bounded by $O(t^d \lg \alpha) = O(\lg \alpha/\varepsilon^d)$.

At the end of the traversal of the DCH, let q be the point closest to p encountered. We will show that q is a $(1+\varepsilon)$-nearest neighbor of p. Let q^* be the closest point to p among all points in S. If q^* is in encountered during the traversal, then clearly $|pq| = |pq^*|$, and we are done. If not, let j be the level such that $P^{(j-1)}(q^*) \notin W_{j-1}$ and $P^{(j)}(q^*) \in W_j$. j exists because the root of the DCH is an ancestor of q and by virtually extending the hierarchy if necessary, the root of the DCH is always visitted during the traversal.

By definition of q and Lemma 5.3.1, $|pq| \leq |pP^{(j)}(q^*)| \leq |pq^*| + |q^*P^{(j)}(q^*)| \leq |pq^*| + 2^j$. On the other hand, $|pq^*| \geq |pP^{(j-1)}(q^*)| - 2^{j-1} \geq (t-1)\cdot 2^{j-1}$. Thus $|pq| \leq (1 + 2/(t-1))|pq^*| = (1+\varepsilon)|pq^*|$. The theorem is proved. □

5.6.5 Closest pair and collision detection

We observe that if p and q is the closest pair of points in S, and let i be such that $2^{i-1} < |pq| \leq 2^i$, then p and p must be in level S_i, and as $|pq| \leq 2^{i+1}$, there is an edge between p and q in the conflict graph G, i.e. there is an edge in G between the shortest pair of points in S. By maintaining G, we can maintain this closest pair of points.

Theorem 5.6.10. *For a set S of moving points in $\mathbb{R}^d$, we have a linear-size kinetic data structure to maintain the closest pair of the point set. The KDS is efficient, local, compact and responsive.*

Proof: We first construct and maintain a DCH of S and its conflict graph G. The edge between the closest pair is the shortest edge in G. As G has only linear number of edges, we simply use a kinetic tournament tree [Gui98] to keep track of the shortest edge among all the spanner edges. The kinetic tournament tree is known to be efficient, local, compact and responsive. Thus our KDS to maintain the closest pair, by using the kinetic DCH and the kinetic tournament tree, has all those good properties as well. Namely, the total size of the KDS is of linear size in the number of points. Each point is involved in $O(\lg n + \lg \alpha/\varepsilon^d)$ certificates. The update cost on a certificate failure is $O(\lg^2 n)$, as there are $O(\lg n)$ edge changes in the conflict graph G, and each edge change in the conflict graph triggers an edge insertion or deletion in the kinetic tournament tree which costs $O(\lg n)$.

There are at most $O(n^2 \lg \alpha)$ edge changes in the DCH and its conflict graph G. The kinetic tournament tree for an input with size m processes $O(m \lg m)$ events. By the efficiency of the kinetic tournament tree, the total number of events processed altogether can be bounded by $O(n^2 \lg \alpha \lg n)$. The number of times the shortest pair of points can change is $O(n^2)$, using the same example as in Figure 1.4. Thus our KDS for maintaining the closest pair is efficient. □

5.6.6 k-center

For a set S of points in $\mathbb{R}^d$, we choose a set K of points, $K \subseteq S$, $|K| = k$, and assign all the points in S to their closest point in K. The k-center problem is to find a K such that the maximum radius of the k-center, $\max_{p \in S} \min_{q \in K} |pq|$, is minimized.

The DCH implies an 8-approximate k-center for any k. We take the lowest level i such that $|S_i| \leq k$. If $|S_i| = k$, then we take $K = S_i$. If $|S_i| < k$, we also add some (arbitrary) children of S_i to K so that $|K| = k$.

Lemma 5.6.11. *K is an 8-approximation of the optimal k-center.*

Proof: On level $i-1$ we have more than k points with distance at least 2^{i-1} pairwise apart. So in the optimal solution, at least two points in S_{i-1} are assigned to the same center. Thus the optimal k-center has maximum radius at least 2^{i-2}. But K has radius at most 2^{i+1}. So K is an 8-approximation to the optimal k-center solution. □

Theorem 5.6.12. *For a set S of n points in $\mathbb{R}^d$, we can maintain an 8-approximate k-center. The KDS is responsive, local and compact. Furthermore, for any fixed integer $t \geq 1$, our KDS for the 8-approximate $(n-t)$-center is efficient.*

Proof: We first maintain a DCH under motion. When the nodes in S_i move, we update the approximate k-center as well. If a node $p \in S_i$ is deleted from level i, we add children of S_i to K to keep $|K| = k$. The only problem that needs to be clarified is when the number of centers at level $i-1$ becomes k or less. However, since $S_i \subseteq S_{i-1}$ and we take K to be S_i and some children of S_i, we thus smoothly switch from level i to level $i-1$. The other event is when $|S_i| = k + 1$, we simply take S_{i+1}. Notice that $S_{i+1} \subseteq S_i \subseteq S_{i-1}$ so the update to K is $O(1)$ per event. And K is changed at most $O(n^2 \lg \alpha)$ times.

To prove the efficiency of our KDS for $k = n - t$ where $t \geq 1$ is a constant, we show that there exists a setting in which any c-approximate $(n - t)$-center, where $c > 1$, has to change $\Omega(n^2)$ times.

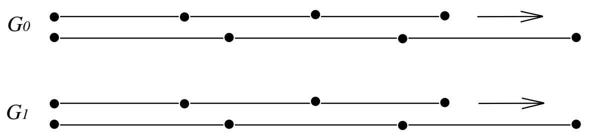

Figure 5.5: Any c-approximate $(n - t)$-center must change $\Omega(n^2/t^2)$ times.

Let $m = n/(2t)$ and fix a value $\gamma > c$. We arrange n points on a plane as in Figure 5.5. First we group the n points into t groups $G_i, 0 \leq i < t$. Each group G_i consists of m *lower* points at positions $(\gamma jm, 2\gamma ti), 0 \leq j < m$, and m *upper* points at positions $(\gamma j(m - 1), 2\gamma ti + 1), 0 \leq j < m$. Note that there are exactly t *matched* pairs of points that are exactly 1 unit distance apart in this arrangement, and the distance between all other pairs of points are at least $\gamma > c$. As the result, the radius of the optimal $(n - t)$-center is 1, and any c-approximate $(n - t)$-center must contain exactly one point in each of the t matched pairs (and the remaining $n - 2t$ points are the unmatched points).

If we fix the lower points in all groups and let the upper points move to the right with the same velocity, every time the upper points move a distance $x\gamma$ for each integer value of x between 1 and $m(m - 1)$, we have a different set of t matched pairs of points, and thus any c-approximate $(n - t)$-center must change. It follows that any c-approximate $(n - t)$-center must change at least $m(m - 1) = \Omega(n^2/t^2)$ times. $\square$

Remark Notice that the spanner actually gives an approximate solution to a set of k-center problems with different k simultaneously. We can maintain j subsets $K_1, K_2, \cdots, K_j$ such that K_i is an 8-approximation of the optimal k_i-center, where $0 \leq k_i \leq n$. The update cost per event is $O(j + \lg \alpha)$.

We also note that there exists a situation and a certain k where the optimal k-center undergoes $\Omega(n^3)$ changes, as the example in [GGH+03]. In fact, that example shows that there is a value k such that k-center with approximation ratio < 1.5 has to change $\Omega(n^3)$

times. On the other hand, any point is a 2-approximation 1-center and thus 2-approximation 1-center does not have to change when the points move. The efficiency of our KDS for approximate k-center for a full spectrum of k is still not well understood.

Chapter 6

Conclusions

We have shown in this dissertation the power of using implicit representations of bounding volumes and bounding volume hierarchies. Implicit bounding volumes such as zonotopes allow one to achieve a good tradeoff, having compact descriptions yet bounding tightly at the same time. Implicit bounding volume hierarchies are more stable when underlying primitives move, and smooth updates of the hierarchies become possible, not only in the kinetic data structure framework but also in the blackbox motion model.

The work is not complete however. While the algorithms proposed have good asymptotic running time in theory, the constant factors behind the asymptotic behavior are sometimes large, making their performance in practice not as desirable. Bridging the gap between theory and practice for bounding volume hierarchies, coming up with algorithms with good worst case theoretical bound that also work well in practice is still an open challenge.

We finally remark that even though we only show that implicit descriptors help in designing data structures for bounding volumes and bounding volume hierarchies, we expects that they also help in designing data structures for other purposes.

Bibliography

[ADM$^+$95] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 489–498, 1995.

[AGMR98] G. Albers, Leonidas J. Guibas, Joseph S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points. *Internat. J. Comput. Geom. Appl.*, 8:365–380, 1998.

[AM02] Sunil Arya and Theocharis Malamatos. Linear-size approximate voronoi diagrams. In *Proc. of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 147–155, 2002.

[AMM02] Sunil Arya, Theocharis Malamatos, and David M. Mount. Space-efficient approximate voronoi diagrams. In *Proc. of the 34th ACM Symposium on Theory of Computing*, pages 721–730, 2002.

[AMN$^+$98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[AMS94] S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 703–712, 1994.

[AS97] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17:33–54, 1997.

[AS00] Pankaj K. Agarwal and Micha Sharir. Arrangements and their applications. In J. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers, 2000.

[Ata85] M. J. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, 11(12):1171–1181, 1985.

[BCG$^+$96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Comput. Graph. Forum*, 15(3):C387–C396, C484, September 1996. Proc. Eurographics'96.

[Bes03] Sergei Bespamyatnikh. Cylindrical hierarchy for deforming necklaces. In *Proc. 9th Ann. Internat. Conf. Computing and Combinatorics (COCOON'03)*, LNCS 2697, pages 20–29, 2003.

[BGH97] J. Basch, Leonidas J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, 1997.

[BGH99] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *J. Alg.*, 31(1):1–28, 1999.

[BGSZ97] J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.

[BGZ97] Julien Basch, Leonidas J. Guibas, and Li Zhang. Proximity problems on moving points. In *Symposium on Computational Geometry*, pages 344–351, 1997.

[BLM89] J. Bourgain, J. Lindenstrauss, and V. Milman. Approximation of zonoids by zonotopes. *Acta Mathematics*, 162:73–141, 1989.

[BM83] U. Betke and P. McMullen. Estimating the sizes of convex bodies from projections. *Journal of London Mathematics Society*, 27:525–538, 1983.

[BO04] Gareth Bradshaw and Carol O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. Graph.*, 23(1):1–26, 2004.

[Cal93] Paul B. Callahan. Optimal parallel all-nearest-neighbors using the well-seated pair decomposition. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 332–340, 1993.

[Cam90] S. Cameron. Collision detection by four-dimensional intersection testing. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 291–302, 1990.

[CK93] Callahan and Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1993.

[CK95a] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[CK95b] Paul B. Callahan and S. Rao Kosaraju. Algorithms for dynamic closest-pair and n-body potential fields. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 263–272, 1995.

[Cla76] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[CLMP95] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. ACM Interactive 3D Graphics Conf.*, pages 189–196, 1995.

[CSSS89] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.

[DB00] S. De and K. J. Bathe. The method of finite spheres. *Computational Mechanics*, 25:329–345, 2000.

[dBKvdSV97] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 294–303, 1997.

[DDS92] M. T. Dickerson, R. L. Drysdale, and J. R. Sack. Simple algorithms for enumerating interpoint distances and finding k nearest neighbors. *Internat. J. Comput. Geom. Appl.*, 2(3):221–239, 1992.

[DK83] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoret. Comput. Sci.*, 27(3):241–253, December 1983.

[Dud74] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10:227–236, 1974.

[EGS86] H. Edelsbrunner, Leonidas J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[EGSZ99] J. Erickson, L. J. Guibas, J. Stolfi, and L. Zhang. Separation sensitive collision detection for convex objects. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 327–336, 1999.

[EL01] Stephan A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3), pages 500–510. Blackwell Publishing, 2001.

[Epp00] David Eppstein. Spanning trees and spanners. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[Eri02] Jeff Erickson. Dense point sets have sparse Delaunay triangulations. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 125–134, 2002.

[Eri05] Jeff Erickson. Local polyhedra and geometric graphs. *Comput. Geom. Theory Appl.*, 31(1-2):101–125, 2005.

[ERW89] H. Edelsbrunner, Günter Rote, and Emo Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theoret. Comput. Sci.*, 66:157–180, 1989.

[FG88] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.

[GDO00] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Spring Conference in Computer Graphics (SCCG)*, pages 156–163, 2000.

[GGH$^+$03] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. *Discrete and Computational Geometry*, 30(1):45–63, 2003.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Comput. Graph.*, 30:171–180, 1996. Proc. SIGGRAPH '96.

[GLM04] Naga K. Govindaraju, Ming C. Lin, and Dinesh Manocha. Fast and reliable collision culling using graphics hardware. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 2–9, New York, NY, USA, 2004. ACM Press.

[GLNS02] Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate distance oracles for geometric graphs. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 828–837, 2002.

[Gon85] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[GR04] Leonidas Guibas and Daniel Russel. An empirical comparison of techniques for updating delaunay triangulations. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 170–179, New York, NY, USA, 2004. ACM Press.

[GRLM03] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 25–32, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Gui98] L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.

[GXZ01] L. J. Guibas, F. Xie, and L. Zhang. Kinetic data structures for efficient simulation. In *Proc. IEEE International Conference on Robotics and Automation (Vol. 3)*, pages 2903–2910, 2001.

[GZ98] Leonidas J. Guibas and Li Zhang. Euclidean proximity and power diagrams. In *Proc. 10th Canadian Conference on Computational Geometry*, pages 90–91, 1998.

[He99] Taosong He. Fast collision detection using quospo trees. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 55–62, New York, NY, USA, 1999. ACM Press.

[Her03] John Hershberger. Smooth kinetic maintenance of clusters. In *Proc. ACM Symposium on Computational Geometry*, pages 48–57, 2003.

[HEV$^+$04] Sunil Hadap, Dave Eberle, Pascal Volino, Ming C. Lin, Stephane Redon, and Christer Ericson. Collision detection and proximity queries. In *GRAPH '04: Proceedings of the conference on SIGGRAPH 2004 course notes*, page 15, New York, NY, USA, 2004. ACM Press.

[HS95] B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. *Math. of Computation*, 64:1541–1555, 1995.

[HSS83] J. E. Hopcroft, J. T. Schwartz, and Micha Sharir. Efficient detection of intersections among spheres. *Internat. J. Robot. Res.*, 2(4):77–80, 1983.

[Hub93] Philip M. Hubbard. Space-time bounds for collision detection. Technical Report CS-93-04, Dept. of Computer Science, Brown University, 1993.

[Hub95] Philip M. Hubbard. Collision detection for interactive graphics applications. *IEEE Trans. Visualization and Computer Graphics*, 1(3):218–230, September 1995.

[Hub96] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph.*, 15(3):179–210, July 1996.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[JP04] Doug L. James and Dinesh K. Pai. Bd-tree: output-sensitive collision detection for reduced deformable models. *ACM Trans. Graph.*, 23(3):393–398, 2004.

[KF03] B. Gärtner K. Fischere. The smallest enclosing ball of balls: Combinatorial structure and algorithms. In *Proc. 19th Annual ACM Symposium on Computational Geometry*, pages 292–301, 2003.

[KGL$^+$98] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and accurate contact determination between spline models using shelltrees. *Computer Graphics Forum*, 17(3):315–326, 1998.

[KHM$^+$98] J. Klosowski, M. Held, Joseph S. B. Mitchell, K. Zikan, and H. Sowizral. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Trans. Visualizat. Comput. Graph.*, 4(1):21–36, 1998.

[KK86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer* *graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1986. ACM Press.

[LC91] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 1008–1014, 1991.

[LG98] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. pages 37–56, 1998.

[LGLM00] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *International Conference on Robotics and Automation*, pages 3719–3726, 2000.

[LK02] Orion Sky Lawlor and Laxmikant V. Kalée. A voxel-based parallel collision detection algorithm. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 285–293, New York, NY, USA, 2002. ACM Press.

[LM04] Ming Lin and Dinesh Manocha. Collision and proximity queries. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 787–807. CRC Press LLC, Boca Raton, FL, 2004.

[LNS02] Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002.

[LS95] H. P. Lenhof and M. Smid. Sequential and parallel algorithms for the k closest pairs problem. *Internat. J. Comput. Geom. Appl.*, 5:273–288, 1995.

[LSHL02] Itay Lotan, Fabian Schwarzer, Dan Halperin, and Jean-Claude Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 43–52, New York, NY, USA, 2002. ACM Press.

[Meg82] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 329–338, 1982.

[Mir97] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, MERL, 201 Broadway, Cambridge, MA 02139, USA, July 1997.

[NS00] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM J. Comput.*, 30:978–989, 2000.

[OL03] Miguel A. Otaduy and Ming C. Lin. Clods: dual hierarchies for multiresolution collision detection. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 94–101, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[OvL80] M. H. Overmars and J. van Leeuwen. Dynamically maintaining configurations in the plane. In *Proc. 12th Annu. ACM Sympos. Theory Comput.*, pages 135–145, 1980.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PG95] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Comput. Graph. Forum*, 14(2):105–116, June 1995.

[PML97] Madhav K. Ponamgi, Dinesh Manocha, and Ming C. Lin. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51–64, 1997.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[Red93] J. N. Reddy. *An Introduction to the Finite Element Method*. McGraw-Hill, 1993.

[RW80] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 110–116, New York, NY, USA, 1980. ACM Press.

[Sal92] J. S. Salowe. Enumerating interdistances in space. *Internat. J. Comput. Geom. Appl.*, 2:49–59, 1992.

[SHH98] Subhash Suri, Philip M. Hubbard, and John F. Hughes. Collision detection in aspect and scale bounded polyhedra. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 127–136, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[ST95] Elmar Schömer and Christian Thiel. Efficient collision detection for moving polyhedra. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 51–60, 1995.

[Sul94] John M. Sullivan. Sphere packings give an explicit bound for the besicovitch covering theorem. *The Journal of Geometric Analysis*, 2(2):219–230, 1994.

[TCL99] Tiow-Seng Tan, Ket-Fah Chong, and Kok-Lim Low. Computing bounding volume hierarchies using model simplification. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 63–69, New York, NY, USA, 1999. ACM Press.

[THM$^+$03] M. TESCHNER, B. HEIDELBERGER, M. MUELLER, D. POMERANETS, and M. GROSS. Optimized spatial hashing for collision detection of deformable objects. In *VMV'03: Proceedings of Vision, Modeling, Visualization*, pages 47–54, 2003.

[TKZ$^+$04] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, and W. Strasser. Collision detection for deformable objects.

In *Eurographics State-of-the-Art Report (EG-STAR)*, pages 119–139. Eurographics Association, Eurographics Association, 2004.

[Vaz01] V. V. Vazirani. *Approximation Algorithms*. Universitext. Springer-Verlag, 2001.

[vdB97] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.

[vdB99] Gino van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools: JGT*, 4(2):7–25, 1999.

[VT94] Pascal Volino and Nadia Magnenat Thalmann. Efficient selfcollisionvdetection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166, 1994.

[Wel91] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes Comput. Sci.*, pages 359–370. Springer-Verlag, 1991.

[WHG84] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Trans. Graph.*, 3(1):52–69, 1984.

[Wri92] M. H. Wright. Interior methods for constrained optimization. In A. Iserles, editor, *Acta Numerica 1992*, pages 341–407. Cambridge University Press, New York, USA, 1992.

[Zac02] Gabriel Zachmann. Minimal hierarchical collision detection. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 121–128, New York, NY, USA, 2002. ACM Press.

[Zie94] G. M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, 1994.

[ZS99] Yunhong Zhou and Subhash Suri. Analysis of a bounding box heuristic for object intersection. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 830–839, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.