Deformable Spanners and Applications

Jie Gao* Leonidas J. Guibas* An Nguyen*

July 1, 2004

Abstract

For a set S of points in $\mathbb{R}^d$, an s-spanner is a graph on S such that any pair of points is connected via some path in the spanner whose total length is at most s times the Euclidean distance between the points. In this paper we propose a new sparse $(1+\varepsilon)$-spanner with $O(n/\varepsilon^d)$ edges, where ε is a specified parameter. The key property of this spanner is that it can be efficiently maintained under dynamic insertion or deletion of points, as well as under continuous motion of the points in both the kinetic data structures setting and in the more realistic blackbox displacement model we introduce. Our deformable spanner succinctly encodes all proximity information in a deforming point cloud, giving us efficient kinetic algorithms for problems such as the closest pair, the near neighbors of all points, approximate nearest neighbor search (aka approximate Voronoi diagram), well-separated pair decomposition, and approximate k-centers.

1 Introduction

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

*Department of Computer Science, Stanford University, Stanford, CA 94305. E-mail: jgao, guibas, anguyen@graphics.stanford.edu.

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 because our intent is to pursue a relatively new direction: spanner maintenance under point motion. The readers are referred to a number of survey papers for background material [2, 14, 32]. Extant spanner constructions are all static, based on sequential centralized algorithms. Our interest is in devising spanner data structures for points in a Euclidean space that can be maintained efficiently under dynamic insertion/deletion as well as continuous motion of the point set.

Maintaining proximity information is crucial in many physical simulations, as most forces in nature are short range — things interact when they are near. This is true across all scales, from smoothed particle hydrodynamics in astronomy to molecular dynamics in biology. We regard collision detection as a special case of proximity maintenance — indeed many extant approaches to collision detection already noted the similarity between that task and that of distance estimation between objects [28]. Of special importance to us is collision or self-collision detection among deformable objects. We came upon the deformable spanners that form the topic of this paper while searching for a lightweight combinatorial data structure that can address the needs of such simulations. Proximity is also important is many aspects of distributed mobile computing, as in *ad hoc* mobile communication networks. Nodes typically can communicate only where they are within a certain range. Proximity-based clustering has been widely used as a way to structure networks and economize on resources [17].

The *aspect ratio* of a point set S, defined by the ratio of the maximum pairwise distance to the minimum pairwise distance of points in S, is denoted by α. Our spanner structure, which we call a DefSpanner (or deformable spanner), is built on point sets with bounded aspect ratio, i.e., ones where α is polynomially bounded by the number of points. In terms of simulations, these points can be thought of as the centers of small elements on which the physical simulation model acts. The bounded aspect ratio condition naturally applies to modeling deformable shapes such as vines, ropes, cloth, tissue and proteins [31, 20, 29]. In all these cases, a deformable object is modeled as a connected collection of small non-overlapping elements of roughly the same size. Thus the aspect ratio is linear in the number of elements. Even when connectivity or disjointness is not required, other physical constraints usually prevent the elements from penetrating too much or drifting too far apart. An example of a spanner for the backbone of a protein is shown in Figure 1.

We propose in this paper a new deformable $(1+\varepsilon)$-spanner (given any $\varepsilon > 0$) for a set of n points in $\mathbb{R}^d$ under the Euclidean metric. We study the properties and applications of such a spanner. Our spanner has $O(n/\varepsilon^d)$ edges. If the point set has bounded aspect ratio, our spanner will have low degree and low weight,

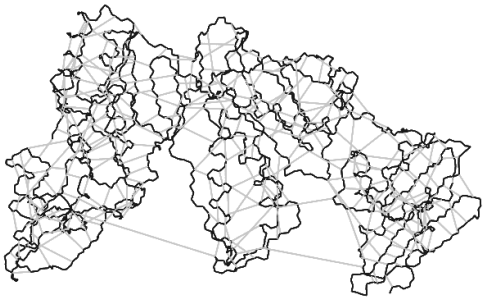

Figure 1. A spanner of a protein backbone. The spanner consists of the backbone edges (black) and a number of additional edges, the shortcuts (light gray). There is a path between any two backbone atoms with length at most 3 times their Euclidean distance.

i.e., the maximum number of spanner edges incident to any point is $O(\log \alpha/\varepsilon^d)$, and the total weight (length) of all edges is $O(\text{MST} \cdot \log \alpha/\epsilon^{d+1})$, where MST denotes the weight of the minimum spanning tree of the point set. Furthermore, the DefSpanner enjoys the additional advantage that it can be updated efficiently under both dynamic and kinetic situations. Most previously proposed algorithms to compute $(1+\varepsilon)$-spanners are all sequential and efficient updates are difficult. To be specific, in the DefSpanner, insertion or deletion of any point can be done in time $O(\log \alpha/\varepsilon^d)$ in the worst case. When the points move continuously, we study the kinetic properties of our spanner in the Kinetic Data Structure (KDS for short) framework [8, 21]. The kinetic spanner changes only at discrete times and has all the properties of a good KDS: efficiency, locality, responsiveness and compactness. To our knowledge, this is the the first kinetic spanner data structure. Under the assumption of bounded apsect ratio, $\log \alpha$ can be replaced by $\log n$ in all the above bounds.

It turns out that our DefSpanner construction only depends on a packing property of Euclidean metrics: a ball with radius r can be covered by at most a constant number of balls of radius $r/2$. Therefore, the spanner, as well as the applications on all the proximity problems, can be directly extended to the metrics with such properties, which were defined as *metrics with constant doubling dimension* [22]. Independently, Krauthgamer and Lee [25] proposed a quite similar hierarchical structure for proximity search in such metrics. They use the hierarchical structure to answer $(1+\varepsilon)$ nearest neighbor search in $O(\log \alpha + (1/\varepsilon)^{O(1)})$ time. Their data structure can be maintained so that each insertion and deletion takes $O(\log \alpha \log\log \alpha)$ time. That work, however, does not address any of the difficult maintenance under motion

issues that form the focus of this paper.

The focus of this paper are the theoretical and combinatorial properties of this spanner construction. We plan to report elsewhere on an implementation and comparisons with other proximity maintenance methods. However, we discuss certain aspects of the use of the spanner in physical simulations when appropriate to motivate and justify the choices we have made. In particular, one of our goals has been to address a deficiency of kinetic data structures in the physical simulation setting.

In the classical KDS presentation, all objects are assumed to move according to known motion plans. This may be a good model for air-traffic, but it is not well-suited for modeling deformable objects. In a typical deformable simulation, elements are moved in discrete time steps by an integrator implementing the physical model, typically an ordinary or partial differential equation. Thus, after an integrator step, the KDS has to recover the structure being maintained, even though the intermediate motions of the elements are hidden from view and possibly multiple certificates have failed. We call this the *blackbox displacement* model: a 'black box' moves the points and we need to repair the spanner structure after these small displacements. The general issue of how to repair a geometric structure after small perturbations of its defining elements can be quite hard. We might hope that we can correlate the amount of repair needed in the structure to, for example, the size of the displacements or the number of failed KDS certificates. But this may not be always possible. As Figures 2 and 3 show, another well-known proximity structure, the Delaunay triangulation, behaves in highly discontinuous ways.

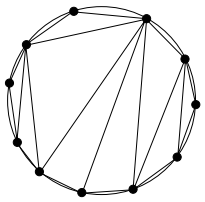

Figure 2. The points shown are nearly cocircular. The Delaunay triangulation could change dramatically even when the points move ever so slightly.

Unlike the Delaunay triangulation, however, our DEFSPANNER is a highly non-canonical structure: for a given configuration of points, many roughly equally good spanner structures are possible. Because of that, we can show that our spanner can be provably and efficiently updated in the blackbox displacement model. The essential reason is that, among all valid spanners for the current configuration we can always choose one that is very similar to the one from the previous time step. In fact, the DEFSPANNER is the first non-trivial data structure that can be provably maintained under the blackbox displacement model.

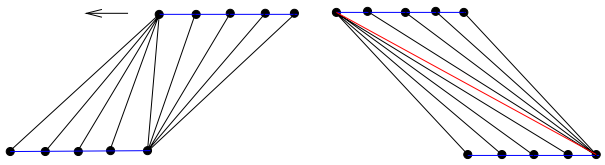

Figure 3. The points on top move to the left. While only one edge (the thick edge) in the triangulation after the motion fails the local Delaunayhood test, repairing the triangulation is very expensive – $\Omega(n^2)$ flips are required.

In addition to basic proximity maintenance, our DEFSPANNER can be used to give efficient kinetic and blackbox displacement algorithms for several related problems. 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 (aka get the functionality of approximate Voronoi diagrams). We also get the first kinetic algorithms for maintaining well-separated pair decompositions and approximate k-centers 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.

2 Specific contributions

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

Closest pair and collision detection. The crucial insight here is that before a pair of elements can collide in a deformable model, any spanner must put an edge between them (otherwise the bounded spanning ratio condition would be violated) — see Figure 4. Note also that the closest pair of elements must have an edge in any $(1+\varepsilon)$-spanner, if $\varepsilon < 1$. Thus the DEFSPANNER naturally contains the information we need for closest pair maintenance and collision detection.

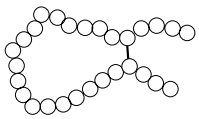

Figure 4. Before a collision happens, a spanner edge must connect the colliding elements.

Again, there is a huge literature on collision detection, but relatively little of it deals with collision detection for deformable objects. The standard approaches based on rigid bounding volume hierarchies do not extend easily to deformable

shapes. Such hierarchies would need to be recomputed as an object deforms, often at considerable cost. One can mitigate the frequency of recomputing the hierarchies by using larger or looser bounding volumes to allow for some deformation, but then the efficiency of the intersection tests suffers. Some efforts towards better deformable bounding volume hierarchies for 'linear' objects are the kinematic chains of Lotan *et al.* [29], where the hierarchy allows for quick updates after a single, or few, joints in the chain move, and the combinatorial sphere hierarchy of Guibas *et al.* [20], where bounding spheres are defined implicitly through feature points on the surface of an object. Both of these structures can perform intersection tests in $O(n^{4/3})$ time in 3-D. There has also been work based on deformable tilings of the free space among moving objects [1], but this is currently limited to 2-D.

Compared to these structures, the DefSpanner 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 DefSpanner, 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 [12]. A pair of point sets (A, B) is *s-well-separated* if the distance 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 [12], 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 [9, 10, 12, 11, 6, 2, 30, 27, 19, 15].

In fact, many of the spanner constructions for points in Euclidean space use the well-separated pair decomposition as a tool [6, 2, 30, 27]. The basic idea is this: the graph defined by taking an arbitrary edge connecting each s-well-separated pair must be a spanner [9]. The spanning ratio can be made arbitrarily close to 1 as long as we choose a large enough s. Here we show that the other direction is also true: the DefSpanner we build 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 [12]. 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.

$(1+\varepsilon)$-nearest neighbor query/approximate Voronoi diagram. The DefSpanner 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 $O(\log n/\varepsilon^d)$. There has been a lot of work on data structures to answer approximate nearest neighbor queries quickly [24, 3, 5, 4]. However, they all try to minimize the storage or query cost and do not consider points in motion.

k-centers. 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 [16]. However, the usual algorithms to compute approximate k-center are of a greedy nature [16, 18], and thus not easy to kinetize. For the dual problem of k-centers, i.e., minimizing the number of centers when the radius of each cluster is prespecified, efficient kinetic maintenance schemes are available [17, 23]. Here we show how to compute an 8-approximate k-center by using the DefSpanner. Furthermore, we are first to give a kinetic approximate k-center as the points move.

2.1 Result summary

We summarize the results below. All the algorithms/data structures are deterministic and we consider the worst-case behavior. For n points S in $\mathbb{R}^d$, we have,

- A $(1+\varepsilon)$-spanner with $O(n/\varepsilon^d)$ edges, $O(\log \alpha/\epsilon^d)$ degree, and $O(\text{MST} \cdot \log \alpha/\epsilon^{d+1})$ weight;

- 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 with size $O(n/\varepsilon^d)$;
- An $O(n)$ structure for $(1+\varepsilon)$-nearest neighbor queries in $O(\log \alpha/\varepsilon^d)$ time, where α is the aspect ratio of S;
- An $O(n)$ data structure for closest pair and collision detection;
- An 8-approximate k-center, for any $0 < k \leq n$.

Furthermore, if the point set also has bounded aspect ratio, we have efficient kinetic and dynamic maintenance for the spanner so that each operation takes $O(\log \alpha/\varepsilon^d)$ time — in fact $\log \alpha$ can be replaced by $\log n$ in all the above bounds. The kinetic data structures for maintaining the $(1+\varepsilon)$-spanner, the $(1/\varepsilon)$-well-separated pair decomposition, the 8-approximate k-center, have the four desirable kinetic properties of efficiency, compactness, locality, and responsiveness.

3 The Deformable spanner

In this paper we focus on a set S of points in the Euclidean space $\mathbb{R}^d$. Recall that the *aspect ratio* α of S is defined by the ratio of the maximum pairwise distance to the minimum pairwise distance of two points in S. Without loss of generality, we assume that the closest pair of points has distance 1, so the furthest pair of S has distance α.

3.1 Spanner definition

A set of *discrete centers* with radius r for point set S is defined as a maximal subset $S' \subseteq S$ such that the balls with radius r centered at the discrete centers contain all the points of S but any two centers are of a distance at least r away from each other. Notice that the set of discrete centers is generally not unique.

We construct a hierarchy of discrete centers such that S_0 is the original point set S and S_i is a set of discrete centers of S_{i-1} with radius 2^i, for $i > 0$. Intuitively, the hierarchical discrete centers are well-distributed samplings of the point set at different spatial scales.

The DefSpanner G on S is constructed as follows: we first construct the hierarchy of discrete centers S_i and then add edges of length no more than $c \cdot 2^i$ between points in S_i to the graph G, where $c = 4 + 16/\varepsilon$. These edges connect each center to other centers in the same level whose distances are comparable to the radius at that level. As pointed out later in Lemma 3.1(3), the edges also connect each center to the points (or centers) it covers in the next lower level.

We use the following notations throughout the paper. Since a point p may appear in many levels in the hierarchy, when it is not clear, we use $p^{(i)}$ to denote the point p in level S_i. A center q in S_i is said to *cover* a point p in S_{i-1} if $|pq| \leq 2^i$. A point p in S_{i-1} may be covered by many centers in S_i. We denote by $P(p)$ one of those centers and call it the parent of p. The choice of $P(p)$ is arbitrary but fixed. We also call p a child of $P(p)$. We recursively define $P^{j-i}(p)$ as the ancestor in level S_j of p by $P^0(p) = p$, $P^{j-i}(p) = P(P^{j-i-1}(p))$, and call $C_{i-1}(p) = \{q \in S_{i-1} | P(q) = p\}$ the set of children of p in S_{i-1}. We denote $N_i(p) = \{q \in S_i \mid |pq| \leq c \cdot 2^i\}$ the set of neighbors of p in S_i.

3.2 Spanner property

We first prove some properties about the discrete center hierarchy and the spanner.

Lemma 3.1. *1.* $S_i \subseteq S_{i-1}$.

2. *For any two points* $p, q \in S_i$, $|pq| \geq 2^i$.
3. *If* $q \in C_i(p)$ *and* $q \neq p$, *then* $q \in N_i(p)$, *i.e. there is an edge from each point* q *to its parent.*
4. *The hierarchy has at most* $\lceil \log_2 \alpha \rceil$ *levels.*
5. *For any point* $p \in S_0$, *its ancestor* $P^i(p) \in S_i$ *is of a distance at most* 2^{i+1} *away from* p.

Proof: The first three claims are obvious. For the fourth claim, an i-th level center $p \in S_i$ has radius 2^i. So if $2^i \geq \alpha$, the ball centered at p contains all the points in S. Therefore the height of the hierarchy h is at most $\lceil \log_2 \alpha \rceil$. The last claim holds because there is a path from $p \in S_0$ to $P^i(p)$ with total length of at most $2 + 2^2 + \cdots + 2^i$. ☑

We are now ready to prove that G is a spanner.

Theorem 3.2. *G is a $(1+\varepsilon)$-spanner.*

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 i-th parents $P^i(p)$ and $P^i(q)$. Denote $p_i = P^i(p), q_i = P^i(q)$, $p_{i-1} = P^{i-1}(p), q_{i-1} = P^{i-1}(q)$. To prove that G is a spanner, we show that the path connecting p, q via p_i, q_i has length at most $(1+\varepsilon)|pq|$.

First, we have that $|p_i q_i| \leq c \cdot 2^i$ and $|p_{i-1} q_{i-1}| > c \cdot 2^{i-1}$. By Lemma 3.1, $|p p_{i-1}| \leq 2^i$, $|q q_{i-1}| \leq 2^i$. So $|pq| \geq |p_{i-1} q_{i-1}| - |p p_{i-1}| - |q q_{i-1}| >$

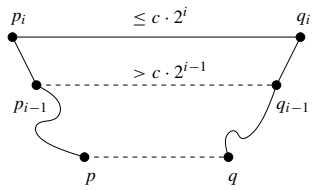

Figure 5. There exists a path in G between any two points p and q with length at most $(1+\varepsilon)|pq|$.

$(c-4)\cdot 2^{i-1}$. Also the length of the path that connect p, q via p_i, q_i is at most $2^{i+1} + |p_i q_i| + 2^{i+1} \leq 8 \cdot 2^i + |pq| \leq (1 + 16/(c-4))|pq| = (1+\varepsilon)|pq|$. This proves that G is a $(1+\varepsilon)$-spanner. ☑

3.3 Size and weight of the spanner

By a standard packing argument, we have,

Lemma 3.3. *Each point in S_i covers at most 5^d points in S_{i-1}.*

Lemma 3.4. *The number of edges any point $p \in S_i$ has with other points of S_i is at most $(1+2c)^d - 1$.*

Note that the bound in Lemma 3.3 could be improved. A more careful analysis, see Sullivan [34], shows that in $\mathbb{R}^2$, the maximum number of children is 19, and in $\mathbb{R}^3$, 87. In higher dimensions, the number of children is at most $O(2.641^d)$.

Lemma 3.5. *The maximum degree of G is $(1+2c)^d \lceil \log_2 \alpha \rceil$.*

Proof: It follows from Lemma 3.4 and Lemma 3.1 (4). ☑

Lemma 3.6. *The total number of edges in G is less than $2(1+2c)^d \cdot n$.*

Proof: Note that if G is a DefSpanner and p is a point in G that does not have any children, then removing p and all edges incident on p from G gives us another DefSpanner G' with one less vertex. The lemma follows if we can show that we can always find a childless point p in G that is incident to at most $2(1+2c)^d$ edges.

Let p and q be the closest pair of points in G and let $k = \lfloor \log_2 |pq| \rfloor$. As p and q cannot be both in S_{k+1}, we assume further that p is not in S_{k+1}. Since p is at least 2^k apart from all points, it does not have any children in level $k-1$ or below, and thus it is childless. By a packing argument similar to that in Lemma 3.4, p is incident to at most $(1+2c/2^j)^d - 1$ edges in S_{k-j}, for all $0 \leq j \leq \log_2 c$. The total number of edges incident on p is thus at most $\sum_{j=0}^{\log_2 c} \left((1+2c/2^j)^d - 1\right) \leq 2(1+2c)^d$. ☑

Lemma 3.7. *The total weight of the spanner is* $O(MST \log \alpha/\epsilon^{d+1})$.

Proof: First we bound the total weight of all edges in a certain level S_i. We charge the edges in S_i to the MST of S_i: an edge incident on $p^{(i)}$ is charged to some MST edges incident on $p^{(i)}$. Each edge of the MST is charged at most $O(1/\epsilon^d)$ times, and the weight of the edges in S_i is at most $c = O(1/\epsilon)$ times the weight of the edges in the MST of S_i. Thus the total weight of the edges in S_i is at most $O(1/\epsilon^{d+1} \cdot \text{MST}(S_i))$, and from that, the total weight of the spanner is $O(\text{MST} \cdot \log \alpha/\epsilon^{d+1})$. ☑

To summarize, we have,

Theorem 3.8. *For a set of n points in $\mathbb{R}^d$ with aspect ratio α, we can construct a $(1+\varepsilon)$-spanner G so that the total number of edges in G is $O(n/\varepsilon^d)$, the maximum degree of G is $O(\log_2 \alpha/\varepsilon^d)$, and the total weight of G is $O(\log_2 \alpha/\epsilon^{d+1} \cdot MST)$.*

Remark. Notice that the hierarchy has at most $\lceil \log_2 \alpha \rceil$ levels. We can replace the logarithm base with any number greater than 1. Specifically if we choose $\beta > 1$, we can build the hierarchy so that for any two points p, q in S_i, $|pq| \geq \beta^i$. The hierarchy has at most $\lceil \log_\beta \alpha \rceil$ levels. Similar to Theorem 3.2, we can show that the graph constructed is a $(1+\epsilon)$ spanner when $c = \max\left(\beta, \frac{4\beta^2}{(\beta-1)\varepsilon} + \frac{2\beta}{\beta-1}\right)$.

4 Construction and dynamic maintenance

The previous section defines a set of properties such that a graph with those properties is a spanner. In this section we show that we can efficiently construct the spanner in $O(n \log_2 \alpha)$ 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 our hierarchy, at a cost of $O(\log_2 \alpha)$ for each operation. In practical settings where α is a polynomial function of n, the construction of the hierarchy is $O(n \log_2 n)$, and dynamic update operations are done in $O(\log_2 n)$ time each.

To describe the dynamic maintenance of the spanner, we adopt a slightly different setting. 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 a set of points S_i, $-\infty < i < \infty$, 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 there are spanner edges only on S_i, $m \leq i \leq M$, $M - m = O(\log n)$. We refer to S_m and S_M the bottom and the top of the hierarchy respectively. 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.

We begin with the following simple yet crucial observation which is used repeatedly in this section. It asserts that if there is an edge between two nodes, then there is also an edge between their parents:

Lemma 4.1. *If $q \in N_i(p)$ then $P(q) \in N_{i+1}(P(p))$.*

Proof: If $q \in N_i(p)$, $|pq| \leq c \cdot 2^i$. Thus $|P(p)P(q)| \leq |P(p)p| + |pq| + |q\,P(q)| \leq (c+4) \cdot 2^i \leq c \cdot 2^{i+1}$. ☑

4.1 Spanner construction

We construct the hierarchy incrementally by inserting points one by one. Suppose that we already have a hierarchy of $n-1$ points. The n-th point p is inserted as follows. We first pretend that p appears in all levels of the hierarchy and insert p into the hierarchy from the top level to the bottom level; $p^{(i)}$'s parent is $p^{(i+1)}$. From Lemma 4.1, p only connects to nodes on level i whose parents on level $i+1$ have already been connected to p. So we compute $N_i(p)$ in each level i in the hierarchy by checking the distance from p to all its 'cousins', i.e., the children of the neighbors of its parent. We stop if p does not have any neighbor. If p has a neighbor in the bottom level, we check whether p has any neighbors in even lower levels and if necessary, decrease m and extend the hierarchy downward. Intuitively, in this step we do point location from top down by using Lemma 4.1 and connect edges no longer than $c \cdot 2^i$ to p on each level i.

We then traverse the hierarchy from the bottom up and clean up the hierarchy of discrete centers. We find the highest level S_i and the point $q \in N_i(p)$ such that $|pq| < 2^i$. We set the parent of p in S_{i-1} to be q, and remove p from all levels S_i and above. If p still remains in the top level, we increase M and extend the hierarchy upward.

Note that we are making two passes through the hierarchy. In each level in each pass, the work is at most $5^d(1+2c)^d = O(1/\varepsilon^d)$, and thus the cost of one insertion is $O(h/\varepsilon^d)$, and the total cost of the construction is $O(nh/\varepsilon^d)$, where $h = O(\log_2 \alpha)$ is the height of the hierarchy.

4.2 Dynamic updates

From the previous subsection, it is clear that we can insert points into the hierarchy at the cost of $O(\log_2 \alpha/\varepsilon^d)$ each. In this section, we show that points can be removed from the hierarchy, again at the cost of $O(\log_2 \alpha/\varepsilon^d)$ each.

To remove a point p from the hierarchy, we remove p from bottom up. If p has no children (except itself), we can simply remove p and all edges incident on p in each level. 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 $q^{(i)}$ is a child of $p^{(i+1)}$, $q \neq p$. From (3) in Lemma 3.1, $q^{(i)}$ is a neighbor of $p^{(i)}$ on level i.

From Lemma 4.1, we know the parent of $q^{(i)}$ must be a neighbor of the parent of $p^{(i)}$, i.e., $p^{(i+1)}$. If there is a neighbor w of $p^{(i+1)}$ that covers $q^{(i)}$, we set $q^{(i)}$'s parent to be w and we are done. If not $q^{(i)}$ is not covered by any centers on level $i+1$, and thus it must be inserted into level $i+1$. Notice that $q^{(i+1)}$ is a neighbor of $p^{(i+1)}$, we can recursively either find a parent for $q^{(i+1)}$ or promote q further up. The neighbors of q can then be computed from top down in a way similar to point insertion in previous subsection.

Note that the cost of raising a child of p up one level is $O(1/\varepsilon^d)$, and as the child may end up in the top level, the cost of fixing a child of p is $O(\log_2 \alpha/\varepsilon^d)$. Since p could appear in $O(\log_2 \alpha)$ levels and has $O(\log_2 \alpha)$ children, removing p may cost $O(\log^2 \alpha/\varepsilon^d)$. However notice that for any level S_i, all children of p on or below the level are inside a disk of radius $2 \cdot 2^i$, and the minimum separation in S_i is 2^i. By a packing argument, at most $O(1)$ of the children can end up being in S_i. The total cost of removing a point is thus $O(\log_2 \alpha/\varepsilon^d)$.

Theorem 4.2. *Dynamic insertion and deletion of points in the spanner takes $O(\log \alpha/\varepsilon^d)$ each, α is the aspect ratio. The spanner can be constructed in time $O(n \log \alpha/\varepsilon^d)$.*

5 Maintenance under motion

5.1 Kinetic data structure overview

We analyze the maintenance of the spanner in the kinetic data structure (KDS) framework [8, 21]. 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, and that the edges are connecting precisely the appropriate nearby pairs of points in each level. When the points move, the certificates may become invalid. A KDS event happens when a certificate fails. At each event, we need to update the certificate set and possibly the spanner. In the KDS framework, the

motion of the points are assumed to be explicitly known, so that the failure times of the certificates can be predicted precisely. The KDS processes the certificate failures in order of the failure time, jumping from one event to the next.

We will show how we maintain the spanner in the KDS framework and vaerify that our spanner enjoys all the desirable properties of a good KDS. We will then discuss how to maintain the spanner in practice, when the motion of the points are given by a black box.

5.2 KDS maintenance

To maintain the spanner G in the KDS framework, we need to maintain the discrete centers hierarchy and the edges between the centers at each level. First, we keep the neighborhood information for each node p. We have three kinds of certificates for this purpose. A *parent-child certificate* guarantees that a child p is within distance 2^{i+1} from its parent in level $i+1$. An *edge certificate* guarantees that a neighbor q of p at level i is within distance $c \cdot 2^i$. A *separation certificate* guarantees that a neighbor q of p at level i is of distance 2^i away. These three certificates prove the validity of the discrete centers hierarchy and also detect when the near neighbors move further away. However, the more difficult part is to detect when two currently far away points move close to each other for the first time. The key observation on the spanner hierarchy is that before two points can become neighbors at some level i, their parents are already neighbors at level $i+1$, as shown in Lemma 4.1. Therefore we only need to keep track of the *potential neighbors* of a point p, which are the 'cousins' of p, i.e., the centers that are not neighbors of p but their parents are $P(p)$'s neighbors. A fourth certificate, *potential neighbor certificate*, guarantees that a potential neighbor of p at level i is of distance $c \cdot 2^i$ further away. All certificates are simple distance comparisons among pairs of points. To summarize, the four kinds of certificates make sure that for each center p in level i, the values $P(p)$ (if $P(p) \neq p$), $N_i(p)$, and $C_{i-1}(p)$ we maintain are valid. The failure of four types of certificates generates five types of events, which are discussed separately as follows. Let p be a point in level i:

1. **Addition of a spanner edge.** When a potential neighbor certificate fails, i.e., a potential neighbor q of p comes within a distance $c \cdot 2^i$ of p, 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 a spanner edge.** When an edge certificate fails, i.e., a neighbor q of p moves such that it is further than $c \cdot 2^i$ from p, we drop the edge between p and q, making them potential neighbors. 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, $|pq| > 2^{i+1}$. p becomes an orphan, and we need to find a new parent for p or promote it into higher levels. We deal with orphans 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 comes within a distance of 2^i, 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.

The number of certificates for a point in any level it participates is $O(1/\varepsilon^d)$, and thus the total number of certificates is $O(n/\varepsilon^d)$, and the number of certificates associated with any point is $O(\log_2 \alpha/\varepsilon^d)$. Assuming that the motion preserves the bounded aspect ratio α, the total number of events in maintaining the spanner under pseudo-algebraic motion is bounded by $O(n^2 \log_2 \alpha)$ since an event only happens when the distance between two points becomes either 2^i or $c \cdot 2^i$ for $i = 0, \cdots, \log_2 \alpha$.

Note that both the dynamic and kinetic maintenance can also be done exactly in the same way for spanners with hierarchy expansion ratio $\beta > 1$ and $c > \max(\beta, 2\beta/(\beta - 1))$.

5.3 Quality of the kinetic spanner

There are four desirable properties that a good KDS should have [8, 21]: (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 of that point is changed, the KDS can be updated quickly; (iv) efficiency: there are not too many certificate failures in the sense that the number of events is not too large comparing to the number of times the attribute being tracked changes combinatorially in the worst case.

As pointed out in the previous subsection, our kinetic spanner has linear number of certificates, and that each point participates in at most $O(\log_2 n/\varepsilon^d)$ certificates. It is also easy to see that the cost to repair the KDS when a certificate fails is either $O(1)$ or $O(\log_2 n/\varepsilon^d)$. Our spanner KDS is thus compact, responsive, and local. As for the efficiency, we first have the following result:

Lemma 5.1. *There exists a set of n points so that any linear-size c-spanner has to change $\Omega(n^2/c^2)$ times.*

Proof: 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 moves accordingly to keep the necklace connected. Figure 6 shows the configuration of the necklace at the starting and ending point.

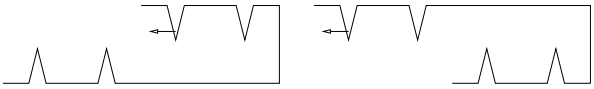

Figure 6. Motion of the points.

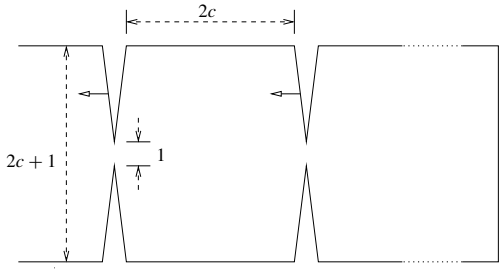

Figure 7. Lower bound $\Omega(n^2/c^2)$ for the changes of any linear-size spanner.

Consider the time when a top bump is directly above a bottom bump, so that the distance between their peaks is 1. See Figure 7. For any c-spanner, there must be a path connecting the two peaks with length no more than c. Therefore there must be an edge between some point in the top bump and some point in the bottom bump, since otherwise any path between the two peaks will be longer than c. So the total number of edges in the c-spanner that ever appear during the motion is at least $\Omega(n^2/c^2)$. Since the spanner starts with $O(n)$ edges. there must be at least $\Omega(n^2/c^2)$ changes of any linear-size c-spanner. ☑

As the number of events that our spanner KDS has to handle is $O(n^2 \log_2 \alpha) = O(n^2 \log_2 n)$, we have thus established:

Theorem 5.2. *The kinetic spanner is efficient, responsive, local and compact. Specifically, the total number of events in maintaining G is $O(n^2 \log_2 n)$ under pseudo-algebraic motion. Each event can be updated in $O(\log_2 n/\varepsilon^d)$ time. A flight-plan change can be handled in $O(\log_2 n/\varepsilon^d)$ time. Each point is involved in at most $O(\log_2 n/\varepsilon^d)$ certificates.*

5.4 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 spanner. If the points move a lot, it can be difficult to repair the structure. However, when the points' motion is small, we can repair the spanner efficiently, since the spanner depends only on the pairwise distances of the points.

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 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 4.1.

Lemma 5.3. *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 $(c/4-1)\cdot 2^i$ in each time step, then in the next time frame p and q are neighbors only if $r = s$ or r and s are neighbors.*

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 $|r_2s_2| \le |r_2r_1| + |r_1p_1| + |p_1p_2| + |p_2q_2| + |q_2q_1| + |q_1s_1| + |s_1s_2| \le (c-4)\cdot 2^i + 2\cdot 2^{i+1} + c\cdot 2^i = c\cdot 2^{i+1}$, and thus $r_2 = s_2$ or r_2 and s_2 are neighbors. ☑

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 $O(n + k\log\alpha)$ where k is the number of changes to the hierarchy. If we know more about the motions of the points, we can lower the spanner update cost by computing conservative bounds on the failure times of various spanner certificates and using the bounds to avoid examining these

certificates for a series of steps. The topic of tighter coupling between the spanner maintenance and the integrator module will be discussed in a future paper.

6 Applications

6.1 Spanners and well-separated pair decomposition

We show by the following theorem that the spanner implies a linear size well-separated pair decomposition.

Lemma 6.1. *The spanner can be turned into an s-well-separated pair decomposition, so that $s = c/4 - 1 = 4/\varepsilon$. The size of the WSPD is $O(n/\varepsilon^d)$.*

Proof: For each node p_i in the spanner, we denote P_i be the set of all decedents of p_i and p_i itself. Now we consider the set C of 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. $|p_i q_i| > c \cdot 2^i$. Note that all nodes in P_i (or Q_i) are within a distance of 2^{i+1} from p_i (or q_i), and thus, the distance between P_i and Q_i is at least $(c-4)2^i$. The diameter of P_i (or Q_i) is at most 2^{i+2}. Thus P_i and Q_i is s-separated, where $s = (c-4)/4$. Note that each pair of points in the hierarchy is covered by one and exactly one pair (P_i, Q_i) in C. It follows that C is an s-well-separated pair decomposition. The number of pairs in C is $O(n/\varepsilon^d)$. ☑

Theorem 6.2. *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 spanner. By using the spanner 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 $O(n^2 \log n)$, as per Theorem 5.2. 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 7, 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. ☑

6.2 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 [7, 13, 26, 33]. 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 DefSpanner can be maintained under motion, so the near neighbors query can be answer 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 [33]. The following theorem is more general with slightly better results and the proof is much simpler.

Theorem 6.3. *For a set S of points in $\mathbb{R}^d$, denote by $\chi(\ell)$ as the number of ordered pairs (p, q), $p, q \in S$ such that $|pq| \leq \ell$, then $\chi(s \cdot \ell) \leq 2(2s+3)^d \chi(\ell) + n(2s+3)^d/2$.*

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 $|pq| \leq \ell/2$. 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 $C(p)$, and $N(p) = |C(p)|$. Note that any pair of points within the same $C(p)$ for any p are within a distance of ℓ of each other.

We consider the pairs of sets $C(p)$ and $C(q)$ such that the distance between $C(p)$ and $C(q)$ is at most $s \cdot \ell$. Clearly $|pq| \leq (s+1) \cdot \ell$, and thus each center participates in at most $(2s+3)^d$ such pairs. Since all the pairs (p', q') with $\ell < |p'q'| \leq s \cdot \ell$ are covered by at least one pair. We try to charge the long 'inter-distances' to the short 'intra-distances'.

From the inequality $ab \leq a(a-1) + b(b-1) + 1$ for all real values a and b, summing over all pairs of p and q such that the distance between $C(p)$ and $C(q)$ is at most $s \cdot \ell$, we obtain $\sum_{p,q} N(p)N(q) \leq 2(2s+3)^d \sum_p N(p)(N(p)-1) + n(2s+3)^d$, and thus $2\chi(s \cdot \ell) \leq 4(2s+3)^{2d} \chi(\ell) + n(2s+3)^d$. ☑

Theorem 6.4. *For a set S of points in $\mathbb{R}^d$, we 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 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 6.3, $\chi(s\ell) \leq 2(2s+3)^d k + n(2s+3)^d/2$, where k is the size of the output. We choose s as a constant, the size of the spanner G is $O(n)$, from Theorem 3.8. ☑

We remark that this output sensitivity is not valid on a per point basis. Figure 8 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.

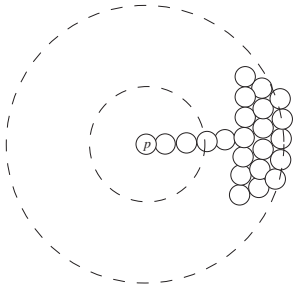

Figure 8. The number of neighbors of point p increases abruptly.

6.3 $(1+\varepsilon)$-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 $|pq| \leq s \cdot |pq^*|$, where q^* is the nearest neighbor of p. We first show that a $(1+\varepsilon)$ nearest neighbor is embedded in a $(1+\varepsilon)$ DefSpanner.

Lemma 6.5. *For a $(1+\varepsilon)$ DefSpanner on a set S of points in $\mathbb{R}^d$, we can perform the $(1+\varepsilon)$-nearest neighbors query in $O(\log \alpha/\varepsilon^d)$ 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: We construct the $(1+\varepsilon)$ DefSpanner G as before. Firstly, we do a fake insertion of p. Assume q is the direct neighbor of p in the spanner with the closest distance, $|pq| = x$, and q^* is the nearest neighbor of p. From the spanner property we know that $\pi_G(p, q^*) \leq (1+\varepsilon) \cdot |pq^*|$. On the other hand, since pq is the

shortest edge attached with p in the graph G, then we must have $\pi_G(p, q^*) \geq |pq|$. This implies that $|pq| \leq (1+\varepsilon) \cdot |pq^*|$.

We find such a q, i.e., the closest neighbor of p on the spanner, as follows. Take the lowest level i where edge pq appears, $c \cdot 2^{i-1} < |pq| \leq c \cdot 2^i$. Then for the level $j \leq i-1$, there is no edge attached with point p. Otherwise that edge would have shorter distance than pq. Therefore for each point $p \in S$, we take the lowest level i where p has an edge in the spanner. We take the shortest edge pq among all the level i edges. q is the $(1+\varepsilon)$-approximate neighbor of p. The theorem then follows from Theorems 3.8 and 4.2. ☑

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

Theorem 6.6. *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 Lemma 5.1 shows that any linear-size structure maintaining the $(1+\varepsilon)$-approximate neighbor has to change $(n^2\varepsilon^2)$ times. ☑

So far we build a $(1+\varepsilon)$ DefSpanner to answer and maintain the $(1+\varepsilon)$-approximate nearest neighbor query for a specific ε. In fact, to answer the $(1+\varepsilon)$-approximate nearest neighbor query, we can decouple the dependency of the DefSpanner on the parameter ε by using a $O(1)$ DefSpanner as an auxiliary structure.

Theorem 6.7. *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 $O(\log \alpha/\varepsilon^d)$ 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: Fix a constant $c > 4$ and construct a DefSpanner using that constant. Given an $\varepsilon > 0$ and a query point p, we let $t = 2 + 4/\varepsilon$. To answer the $(1+\varepsilon)$ approximate nearest neighbor of p, we traverse the DefSpanner top down and keep track of the set $K_i = \{q \mid q \in S_i, |pq| < t \cdot 2^i\}$ as the level i decreases.

First of all, we notice that $|K_i| = O(t^d)$, for any i, since the points in S_i are at least distance 2^i apart. Secondly, we observe that for a point $q \in S$, if $|pq| < t \cdot 2^i$, then $|pP(q)| < t \cdot 2^{i+1}$. This is due to the triangular inequality:

$|pP(q)| \leq |pq| + |qP(q)| < t \cdot 2^i + 2^{i+1} \leq t \cdot 2^{i+1}$. Therefore K_i must be included in the set of the children of K_{i+1}. So we can construct K_i from K_{i+1} in $O(t^d)$ time. The total running time of such a traversal is bounded by $O(t^d \log \alpha) = O(\log \alpha / \varepsilon^d)$.

At the end of the traversal of the DEFSPANNER, let q be the point closest to p in K_0. We will show that q is a $(1+\varepsilon)$-nearest neighbor of p. Let $q^* \in S_0$ be the closest point to p among all points in the spanner. If q^* is in K_0, then clearly $|pq| = |pq^*|$, and we are done. If not, let j be such that $P^{j-1}(q^*) \notin K_{j-1}$ and $P^j(q^*) \in K_j$. By definition of q and Lemma 3.1, $|pq| \leq |pP^j(q^*)| \leq |pq^*| + 2^{j+1}$. Since $|pq^*| \geq |pP^{j-1}(q^*)| - 2^j > (t-2) \cdot 2^{j-1}$, $|pq| < (1 + 4/(t-2))|pq^*| = (1+\epsilon)|pq^*|$. The theorem is proved. ☑

We note that while we need $c > 4$ in order to construct and maintain the DEFSPANNER, if we are only interested in static nearest neighbor queries, a DEFSPANNER with $c > 2$ would be suffice, even though a DEFSPANNER may not be a spanner when $c \leq 4$.

6.4 Closest pair and collision detection

Theorem 6.8. *For a set S of points in $\mathbb{R}^d$, we have a structure of size $O(n)$ to output the closest pair of the point set.*

Proof: We construct the $(1+\varepsilon)$-spanner G as before. If we take $\varepsilon < 1$, then the closest pair pq must have an edge in G. Otherwise, since the shortest path between p, q in G contains at least 2 edges, each of them is longer than $|pq|$, $\pi_G(p,q) > 2|pq|$. This contradicts with the spanner property. To maintain the closest pair, we simply use a kinetic priority queue [21] to keep the shortest edge among all the spanner edges. ☑

6.5 k-center

For a set S of points in $\mathbb{R}^d$, we choose a set K of points, $K \subseteq S$, $|K| = k$, we assign all the points in S to the 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.

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 6.9. *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. ☑

Theorem 6.10. *For a set S of points in $\mathbb{R}^d$, we can maintain an 8-approximate k-center. The KDS is efficient, responsive, local and compact.*

Proof: We first maintain a constant-spanner under motion. When the nodes in S_i changes, we update the approximate k-center as well. If a node $p \in S_i$ is deleted from level i, we add a 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 $O(n^2 \log_2 \alpha)$ times.

In addition, there exists a situation and a certain k where the optimal k-center undergoes $\Omega(n^3)$ changes, as the example in [17]. In fact, that example shows that any approximate k-center with approximation ratio < 1.5 has to change $\Omega(n^3)$ times. For a general approximation factor c, there exists an example showing an $\Omega(n^2)$ bound. The details are omitted. So the KDS to maintain 8-approximate k-center is efficient.

☑

Remark Notice that the spanner actually gives an approximate solution to a set of k-center problems with different k simultaneously. We can maintain t subsets $K_1, K_2, \cdots, K_t$ such that K_i is an 8-approximation of the optimal k_i-center, where $0 \le k_i \le n$. The update cost per event is $O(t + \log_2 n)$.

7 Conclusion

In this paper we have introduced a hierarchical construction that yields a spanner for a set of n points in $\mathbb{R}^d$ under the Euclidean metric. The remarkable property of this spanner is that is can be maintained easily under point insertion, deletion, or motion. We have shown that the DefSpanner allows a wide variety of proximity queries to be answered efficiently and provides the first known structure so capable that can be maintained under motion.

Acknowledgements: This research was supported in part by NSF grants CCR-0204486 and ITR-0205671, ONR MURI grant N0014-02-1-0720, and the Defense Advanced Research Projects Agency (DARPA) under contract number F30602-00-C-0139 through the Sensor Information Technology Program. Jie Gao was also supported by an IBM Ph.D fellowship. The authors wish to thank John Hershberger for useful discussions and in particular his contributions to Section 6.2, as well as an anonymous reviewer for suggesting relevant previous work.

References

[1] P. Agarwal, J. Basch, L. Guibas, J. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection. *International Journal of Robotics Research*, 21(3):179–197, 2002.

[2] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th ACM Symposium on Theory Computing*, pages 489–498, 1995.

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

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

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[6] S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 703–712, 1994.

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

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

[9] 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.

[10] P. B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proc. 34th IEEE Symposium on Foundations of Computer Science*, pages 332–340, 1993.

[11] P. B. Callahan and S. R. Kosaraju. Algorithms for dynamic closest-pair and n-body potential fields. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 263–272, 1995.

[12] 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.

[13] 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.

[14] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

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

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

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

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

[19] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graphs. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 828–837, 2002.

[20] L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. In *Proc. 18th ACM Symposium on Computational Geometry*, pages 33–42, 2002.

[21] 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.

[22] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proc. IEEE Symposium on Foundations of Computer Science*, 2003.

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

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

[25] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

[26] 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.

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

[28] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *IEEE International Conference on Robotics and Automation*, pages 1008–1014, Apr. 1991.

[29] I. Lotan, F. Schwarzer, D. Halperin, and J.-C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *Proc. of the 18th ACM Symposium on Computational geometry*, pages 43–52, 2002.

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

[31] D. K. Pai. STRANDS: Interactive simulation of thin solids using cosserat models. In *Eurographics*, 2002.

[32] D. Peleg. *Distributed Computing: A Locality Sensitive Approach*. Monographs on Discrete Mathematics and Applications. SIAM, 2000.

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

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