Kinetic Mesh Refinement in 2D

Umut A. Acar
Max Planck Institute
for Software Systems
Germany
umut@mpi-sws.org

Benoît Hudson
Autodesk, Inc.
Montréal, QC, Canada
benoit.hudson@autodesk.com

Duru Türkoğlu
Dept. of Computer Science
University of Chicago
Chicago, IL, USA
duru@cs.uchicago.edu

ABSTRACT

We provide a kinetic data structure (KDS) to the planar *kinetic mesh refinement* problem, which concerns computation of meshes of continuously moving points. Our KDS computes the Delaunay triangulation of a size-optimal well-spaced superset of a set of moving points with algebraic trajectories of constant degree. Our KDS is compact, requiring linear space in the size of the output. It is local, using a point in $O(\log \Delta)$ certificates. It is responsive, repairing itself in $O(\log \Delta)$ time per event. It is efficient, processing $O(n^2 \log^3 \Delta)$ events in the worst case; this is optimal up to a polylogarithmic factor. Also, our KDS is dynamic, responding to point insertions and deletions in $O(\log \Delta)$ time. In our bounds Δ stands for the geometric spread, the ratio of the diameter to the closest pair distance. To the best of our knowledge, this is the first KDS for mesh refinement.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and ProblemComplexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Theory

Keywords

Mesh refinement, Well-spaced point sets, Kinetic data structures, Deformable spanners, Self-adjusting computation

1. INTRODUCTION

Mesh refinement is an essential step in many applications such as surface reconstruction, physical simulations, more broadly in scientific computing, graphics etc. (e.g., [12]). The idea behind mesh refinement is to break up a physical domain into discrete elements, e.g., triangles, so that certain functions defined on the domain may be computed approximately by considering the discrete elements. For accuracy of the computed properties, it is important that the triangles be "well-shaped", i.e., no triangle should have small angles. When the input points admit no well-shaped triangulation, meshing algorithms can include additional *Steiner points* in their output mesh. Ideally, the output meshes will be *size-optimal*: their size (number of vertices and triangles) should be within a constant factor of the size of the smallest possible mesh consisting of well-shaped elements. A *quality* mesh is both well-shaped and size-optimal.

We distinguish three related problem statements. In the simplest, the input is a *static* set of points and we want to construct a quality mesh. The input may also be given *dynamically*, as a series of additions and removals from the input; the task is to maintain a quality mesh after each input change. Our main focus in this article is the case in which points have *kinetics*: each point has an associated velocity function and we maintain a quality mesh at all times. The first efficient static algorithm is due to Bern, Eppstein and Gilbert [9]. However, the first provably efficient algorithm for the dynamic problem was only recently demonstrated [4], and the kinetic version of the problem, which requires computation of quality meshes of changing set of moving objects, was open until the present work.

We develop our solution in the kinetic data structures framework, KDS in short [8, 15]. A KDS uses *certificates* to certify, i.e., witness the correctness of, the current mesh, and schedules *events* to repair the mesh when certificates fail. The structure can determine the time of an event by using its knowledge of the motion plans of the points. When an event happens, the data structure updates the mesh as well as the set of certificates. We analyze the performance of a KDS according to four properties: responsiveness, efficiency, compactness, and locality. *Responsiveness* requires the data structure to respond to events in time polylogarithmic in the size of the input. *Efficiency* requires the total number of events processed to be at most a polylogarithmic factor larger than the minimal number of combinatorial changes (the addition or removal of a Steiner point). *Compactness* requires the number of the certificates to be near-linear. *Locality* requires each input point to participate in a polylogarithmic number of certificates.

Previous work proposes KDS's for the related problem of kinetic triangulations, without the insertion of Steiner points. The most efficient kinetic Delaunay triangulation schemes [13, 16] are a linear factor less efficient than optimal. There are structures that efficiently maintain non-Delaunay triangulations in the plane [7, 20]. Agarwal *et al* [6] show how to maintain the *stable Delaunay graph*, namely the subset of the Delaunay triangulation that has large minimum angles. We can efficiently maintain the full Delaunay graph

SCG'11, June 13–15, 2011, Paris, France.

of our output point set by inserting Steiner points to ensure that all triangles have large minimum angles.

To see the challenges in providing an effective kinetic data structure for meshing, it helps to consider the techniques that work well for the static version of the problem. One approach based on *balanced quadtrees*, generates an appropriately refined quadtree over the input points and adds the corners of the quadtree squares as Steiner points [9]. In the kinetic setting, the quadtree, because it is fixed, can generate a large number of events. For example, if the input contains two close points that move along parallel linear trajectories preserving the distance, say ϵ, between them, then the quadtree may need to be restructured every time the points leave their quadtree cell, which is only $\Theta(\epsilon)$ distance. In other words, a quadtree approach cannot be *efficient*. Another approach computes Voronoi diagrams and inserts the corners of the Voronoi cells (equivalently, the Delaunay circumcenters) as Steiner points [18, 5]. The main difficulty in kinetizing these approaches is that the position function of a Steiner point depends on three points, some of which may themselves be Steiner points. The description length of the position function can thereby build up to be polynomial in n. Since computing just one event time could take polynomial time, such a structure cannot be *responsive*.

In this paper, we provide an effective kinetic data structure for computing meshes of a dynamically changing set of points consisting of points moving along algebraic trajectories of constant degree. Our KDS yields triangulations of size-optimal well-spaced point sets (Section 5). We analyze the responsiveness, efficiency, locality, and compactness of our data structure as functions of the input size and the geometric spread (the ratio of diameter to closest pair). Since the spread changes as time evolves, we define Δ to be the ratio of the maximum diameter of the input at any time, and the minimum distance between closest pair of input points at any time. If the spread is polynomially bounded by the input size, our data structure yields bounds all in the problem size with logarithmic factors. Our KDS guarantees:

Responsiveness. A certificate failure requires $O(\log \Delta)$ update time (Theorem 4.7).

Locality. A point participates in $O(\log \Delta)$ certificates (Lemma 5.1).

Compactness. Total number of certificates is $O(m)$, where m is the output size and $m \in O(n \log \Delta)$ (Lemma 5.1).

Efficiency. The number of events is $O(n^2 \log^3 \Delta)$ which is within a $O(\log^2 \Delta)$ factor of the optimal (Lemma 5.4).

Dynamic updates. A point insertion or deletion requires $O(\log \Delta)$ update time (Theorem 4.7).

At a high level, our solution is in essence a balanced quadtree method, replacing the quadtree with a variant of the deformable spanner of Gao, Guibas, and Nguyen [14]. Our KDS consists of a construction algorithm (Section 3) that computes a quality mesh of the input, and an update algorithm (Section 4) that enables kinetic motion simulation and dynamic changes. Given a set of input points, the construction algorithm first computes a *well-spaced* superset of the input in $O(\log \Delta)$ levels. A set of points M is *well-spaced* if for each point $p \in \mathsf{M}$, the ratio of the distance to the farthest point in the Voronoi cell of p divided by the distance to the nearest neighbor of p in M is small [23]. Intuitively, in a well-spaced point set, points have "well-shaped" Voronoi cells. The construction algorithm then computes a Delaunay triangulation of the well-spaced set by performing local computations only, yielding a Delaunay triangulation with no small angles. The construction algorithm hinges on a technique for determining the position and the motion plans for Steiner points. Upon a kinetic event or a dynamic modification to the input set, the update algorithm repairs the well-spaced superset and its Delaunay triangulation. The update algorithm propagates the changes through the construction algorithm, updating each of the $O(\log \Delta)$ levels, by performing amortized constant work at each level.

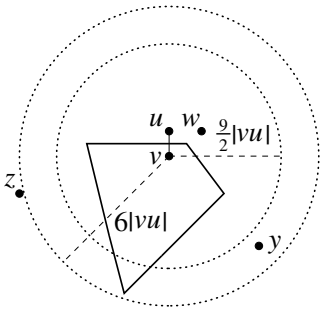

Figure 1: Given the set $\mathsf{M} = \{v, u, w, y, z\}$, the nearest neighbor distance of v, $\mathrm{NN}_{\mathsf{M}}(v)$, is $|vu|$. Thick lines depict the Voronoi cell of v, $\mathrm{Vor}_{\mathsf{M}}(v)$; v is 6-well-spaced, but not $\frac{9}{2}$-well-spaced.

The approach of developing a construction algorithm and then providing an update algorithm based on change propagation is inspired by recent advances on self-adjusting computation (e.g., [17, 21, 1]). In self-adjusting computation, programs can respond automatically to modifications to their data by invoking a general-purpose change propagation algorithm that can also utilize a certain (traceable) data structure to ensure asymptotic efficiency [2]. These techniques have been applied effectively to other computational geometry problems such as kinetic convex hulls in 3D [3] and dynamic well-spaced point sets [4].

2. PRELIMINARIES

We present some definitions used throughout the paper, describe the technique we use for selecting Steiner vertices, and present a brief overview of the deformable spanners.

2.1 Definitions

Kinetic mesh refinement is the problem of maintaining a quality triangulation of a superset M of an input point set N. We approach this problem in the dual setting: Voronoi diagrams and well-spaced point sets. We define a *domain* Ω to be a ball centered at an arbitrary input point with radius at least nine times the diameter of the given input N. For any given superset M of N, the *Voronoi cell of v in M*, written $\mathrm{Vor}_{\mathsf{M}}(v)$, consists of points $x \in \Omega$ such that for all $u \in \mathsf{M}, |ux| \geq |vx|$. The *nearest-neighbor distance of v in M*, written $\mathrm{NN}_{\mathsf{M}}(v)$, as the distance from v to the nearest other point in M. A point $v \in \mathsf{M}$ is *ρ-well-spaced* if the Voronoi cell of v is contained in the ball with radius $\rho\,\mathrm{NN}_{\mathsf{M}}(v)$ centered at v; M is ρ-well-spaced if every point in M is ρ-well-spaced. Figure 1 illustrates these definitions.

It is important that the output is as small as possible while a quality triangulation is achievable. We minimize the output size (up to a constant factor) by constructing a size-conforming output [22]. A set $\mathsf{M} \supset \mathsf{N}$ is *size-conforming* if there exists a constant c independent of N such that for all $v \in \mathsf{M}$, $\mathrm{NN}_{\mathsf{M}}(v) > c \cdot \mathrm{lfs}(v)$, where $\mathrm{lfs}(v)$, the *local feature size* of v, is the distance from v to the second-nearest point in N. For the rest of the paper, we use the term *point* to refer to any point in Ω and the term *vertex* to refer to the input and output points.

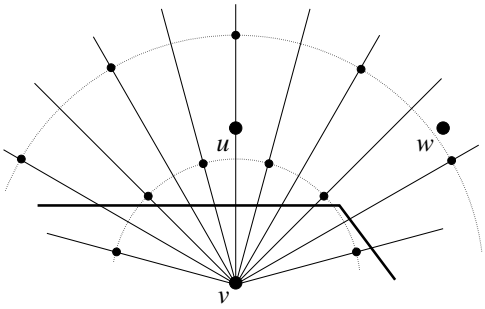

Figure 2: Zooming in on the point set M of Figure 1, first two orbits and some rays of v (nucleus) are displayed, with part of its Voronoi cell in thick lines. Intersections of odd rays with orbits at odd ranks and even rays with orbits at even ranks form satellites—potential Steiner vertices—shown in smaller dots.

2.2 Selecting Steiner Vertices

The key problem in mesh refinement is the selection of Steiner vertices, i.e., where they should be inserted and how they should move. To solve this problem, we propose a local template-based approach. Specifically, the template specifies for each input vertex v, a *nucleus*, an infinite number of *satellites* that may be inserted as Steiner vertices. A nucleus and its satellites form a well-spaced point set and the satellites move together with their nucleus: the position curves of the satellites are the same as their nucleus' position curve plus a fixed translation. To ensure a size-conforming output, we set the spacing between the satellites proportional to the distance to their nucleus, i.e., for each satellite, the distance to the nearest other satellite is within a constant multiplicative factor of the distance to their nucleus.

For the planar case, we use a particular template (illustrated in Figure 2) defined by a fixed set of rays emanating from a nucleus and their intersections with concentric circles of geometrically increasing radii. Consider 24 rays leaving each input vertex at angles $\theta, 2\theta, \ldots, 24\theta$, where $\theta = \pi/12$. Also, consider concentric circular orbits of radius 2^ℓ around every input vertex where $\ell \in \mathbb{Z}$ is the *rank* of these orbits. Defining odd (even) rays to be rays at angles that are odd (even) multiples of θ, we choose certain translations to define the satellites: the intersections of odd rays with orbits at odd ranks and the intersections of even rays with orbits at even ranks. In this template, a nucleus, a rank, and a ray with the same parity as that of the rank defines a unique satellite. Intuitively, this is a discrete polar coordinate system, where the nucleus defines the origin, the rank defines the radius (exponentially), and the ray defines the polar angle. In the rest of the paper, we use the term ℓ-satellite to refer to a satellite on an orbit at rank ℓ.

2.3 Deformable Spanners

Our algorithm uses the kinetic deformable spanners of Gao, Guibas, and Nguyen as a point location data structure [14]. In our algorithms, in order to generate a quality mesh, we insert certain satellites into the deformable spanner data structure. Taking advantage of the location of the satellites, we achieve accuracy and efficiency by extending the deformable spanners with exact nearest neighbor queries and more efficient vertex insertion procedures. In this section, we briefly overview the original deformable spanners data structure; we explain the extension in the next section. In the most general form, given a parameter $\varepsilon > 0$, the deformable spanners $(1+\varepsilon)$-approximates the Euclidean distance between the vertices. Throughout the paper, we use the spanner with $\varepsilon = 1$, guaranteeing 2-approximation. A spanner represents a hierarchical discretization of a vertex set at geometrically increasing scales. Given a vertex set M and any $s > 0$, a *discretization of* M *at scale* s is a subset $\mathsf{M}' \subseteq \mathsf{M}$ of vertices which satisfy the following two conditions: i) the minimum distance between any two vertices of M' is at least s, ii) the set of balls $\{B(v, s) \mid v \in \mathsf{M}'\}$ cover M. Note that there can be multiple discretizations of a vertex set. A spanner is based on a hierarchy of discretizations $\mathsf{M} = \mathsf{M}_\lambda \supseteq \mathsf{M}_{\lambda+1} \supseteq \cdots \supseteq \mathsf{M}_\Lambda$ that satisfy the following properties.

- λ is the minimum integer such that $2^{\lambda-1} \leq \delta < 2^\lambda$, where δ is the closest pair distance in M.
- For each $\ell \in \{\lambda+1, \lambda+2, \ldots, \Lambda\}$, M_ℓ is a discretization of $\mathsf{M}_{\ell-1}$ at scale 2^ℓ.
- M_Λ is the only singleton in the hierarchy.

We call M_ℓ a discretization at *rank*[1] ℓ and refer to its vertices as *discrete centers* or more specifically as ℓ*-centers*. A spanner connects discrete centers at the same and at consecutive ranks with neighbor, parent, and child pointers. Specifically, two ℓ-centers are ℓ*-neighbors* if the distance between them is at most $c \cdot 2^\ell$, where $c = 4 + 16/\varepsilon = 20$. If v is both an ℓ-center and an $(\ell+1)$-center, v is its own parent/child. Otherwise, an $(\ell+1)$-center w whose distance to v is at most $2^{\ell+1}$ is designated to be the *parent* of v; and v to be a child of w at rank ℓ. For a vertex v, we define its *maximum rank*, written Λ_v, to be the highest rank where v is a discrete center. We define its *minimum rank*, written λ_v, to be the lowest rank ℓ where v is a discrete center with at least one ℓ-neighbor. Now, we briefly explain the construction of the spanner using a somewhat different presentation than the one used by Gao et al. [14]. Also, we describe their certificates and summarize their update algorithm.

Construction. We start by assigning an arbitrary input vertex v to be the root of the spanner S and set the maximum rank Λ of S to $\max_{w \in \mathsf{N}} \lceil \lg |vw| \rceil$, i.e., we set $\mathsf{S}_\Lambda = \{v\}$. Furthermore, we temporarily set v to be the parent of every other input vertex. In a top-down pass, at each rank ℓ, we greedily determine the set of ℓ-centers (S_ℓ) as follows: initializing $\mathsf{S}_\ell = \mathsf{S}_{\ell+1}$ and performing a linear pass on the set of remaining vertices, $\mathsf{N} \setminus \mathsf{S}_{\ell+1}$, we insert a vertex w into S_ℓ if the ball $B(w, 2^\ell)$ does not contain an ℓ-center. This can be done by checking the cousins (parent's neighbors' children) of w. Otherwise, if w is 2^ℓ-close to an ℓ-center v, we update the parent of w to be v. We also insert neighbor edges between any two ℓ-centers within $c \cdot 2^\ell$ distance, again using cousins. We stop at rank $\ell = \lambda$, when $\mathsf{S}_\ell = \mathsf{N}$.

Certificates. To certify the spanner, Gao et al. describe four kinds of certificates. These are: i) *parent-child* certificates for certifying $|vu| \leq 2^{\ell+1}$ for an ℓ-center v and its parent u at rank $\ell+1$. ii) *separation* certificates for certifying $|vu| > 2^\ell$ for ℓ-neighbors v and u. iii) *edge* certificates for certifying $|vu| \leq c \cdot 2^\ell$ for ℓ-neighbors v and u. iv) *potential neighbor* certificates for certifying $|vu| > c \cdot 2^\ell$ for two ℓ-centers v and u, whose parents are $(\ell+1)$-neighbors.

[1]Note that this definition coincides with the rank definition that is used to describe the satellites—both definitions provide a logarithmic scale with base 2. One could use different bases; for simplicity, we chose base 2 for both.

```
Construct a spanner S for N

∀v ∈ N, activation rank ρ_v ← λ_v
for ℓ = λ to Λ + 4
  ∀v ∈ N, v is active if
    ρ_v = ℓ or ∃ a converted (ℓ − 1)-satellite of v
  for each ℓ-satellite s of each active v ∈ N
    if BallEmpty(S, s, 2^{ℓ−2}) then Convert(S, s, ℓ)
  for each Steiner vertex s ∈ S_ℓ
    TryToPromote(S_{ℓ+1}, s)

for each rank ℓ
  for each s ∈ N with ρ_s = ℓ or ℓ-satellite s ∈ M \ N
    V ← ∪_{ℓ'∈{ℓ−6,...,ℓ−3}} ℓ'-neighbors of s
    IdentifyVoronoiNeighbors(s, V)
```

Figure 3: The pseudo-code for the construction

Maintenance. When some certificates fail, we update the discretizations in a top-down pass. Assuming that the spanner at ranks above ℓ is updated, we fix the spanner at rank ℓ. First, we check if any of the parent-child certificates at rank ℓ has failed. If an ℓ-center v is no longer a child of an $(\ell+1)$-center w, we check to see if there is another $(\ell+1)$-center, an $(\ell+1)$-neighbor of w, that is $2^{\ell+1}$-close to v. If there is, we assign that vertex as the new parent of v, if not we *promote* v to rank $\ell+1$ and repeat the promotion procedure until we assign a new parent to v. In promoting to a higher rank $\ell' > \ell+1$, we use the rank ℓ' ancestor of v for locating v, e.g., parent of w for $\ell' = \ell+2$. Next, if a separation certificate has failed, i.e., an ℓ-center v becomes a child of an $(\ell+1)$-center, we remove v from ranks $\ell+1$ and above. This leaves the children of v at ranks $\ell+1$ and above without a parent. We promote these children as necessary, by applying the above procedure. In repairing both types of certificate failures, we insert and remove neighbor edges in order to maintain a valid spanner structure. In addition, we update the neighbor edges, if any of the edge or potential-neighbor certificates has failed. During maintenance, besides updating the spanner, we update the set of certificates as necessary in order to certify the updated spanner.

3. CONSTRUCTION ALGORITHM

Our construction algorithm builds a mesh in three stages (Figure 3). First, it constructs a spanner for the input vertices by running the construction algorithm described in Section 2.3; second, it constructs a well-spaced superset M of the input by inserting certain satellites into the spanner; third, it constructs the Delaunay triangulation of M.

In the second stage, our algorithm iterates over ranks and determines a set of *active* input vertices and applies a *conversion* and a *promotion* process at each rank. In the conversion process, it selects certain satellites of active input vertices and inserts them into the spanner as Steiner vertices. In the promotion process, it inserts certain Steiner vertices into the discretization at the next rank in order to represent the current superset correctly at the next rank.

As part of determining the active status of input vertices, our algorithm starts the second stage by assigning each input vertex an *activation rank*, which is defined as the minimum rank in the initial spanner. At each rank ℓ, our algorithm then determines the active input vertices—an input vertex is *active* at rank ℓ if its activation rank is ℓ or it was active at rank $\ell-1$ and one of its $(\ell-1)$-satellites was converted to a Steiner vertex. It continues by applying the conversion process on the ℓ-satellites of active input vertices. Specifically, iterating through each ℓ-satellite s of each active vertex, our algorithm converts s if and only if s has an empty *certificate ball*, which is defined as the ball $B(s, 2^{\ell-2})$. When converting an ℓ-satellite s, our algorithm inserts s into the spanner hierarchy beginning at S_{λ_s} and promotes s to the highest possible rank up to rank ℓ. Once the algorithm is done with the conversion process, it updates the discretization at the next rank ($\mathsf{S}_{\ell+1}$) and prepares the spanner for the next iteration by promoting certain ℓ-centers. In this promotion process, by calling `TryToPromote` (details below) for each Steiner vertex $s \in \mathsf{S}_\ell$, the algorithm determines whether there is an $(\ell+1)$-center $2^{\ell+1}$ close to s. If there is one, the algorithm assigns it to be the parent of s. Otherwise, the algorithm inserts s into $\mathsf{S}_{\ell+1}$, i.e., promotes s to rank $\ell+1$, and determines the $(\ell+1)$-neighbors of s.

In the third stage, the algorithm identifies the Voronoi neighbors of the vertices. For each Steiner vertex s at rank ℓ and each input vertex with activation rank ℓ, it first generates a candidate set consisting of the spanner neighbors of s at ranks $\{\ell-6, \ldots, \ell-3\}$. It then computes the Voronoi cell of s in this candidate set in $O(1)$ time; this computation yields the true Voronoi cell of s in the well-spaced superset M (Lemma 3.9). Then, the complete Voronoi diagram yields the Delaunay triangulation of M.

In the rest of this section, we prove the correctness and the $O(n \log \Delta)$ runtime bound of our algorithm. For realizing the run-time bound, we focus on efficiently locating any ℓ-satellite s in the spanner using its nucleus v. We extend the spanner data structure with $O(1)$ time functions, `BallEmpty`, `Convert`, and `TryToPromote`, details of which we describe in the next two paragraphs. Taking advantage of these $O(1)$ time functions, we bound the total runtime converting satellites to Steiner vertices by $O(n \log \Delta)$ as there are $O(n \log \Delta)$ many satellites to consider. Then, we bound the runtime of the discretization step by $O(n \log \Delta)$, by proving that there are $O(n)$ ℓ-centers in the spanner at each rank ℓ (Lemma 3.11) and $O(\log \Delta)$ ranks. In the third stage, since the computation of the Voronoi cell of each vertex in M takes $O(1)$ time, we achieve the desired runtime bound.

In order to efficiently locate an ℓ-satellite s of an active vertex v, we define the notion of ℓ-sites. An *ℓ-site* of s is an input vertex $w \in \mathsf{S}_\ell$ (at rank ℓ) that satisfies the condition $|sw| \leq c \cdot 2^{\ell-2}$. By Lemma 3.1, the rank ℓ ancestor of v, say w, is an ℓ-site of s. A naive approach locates w in $O(\ell - \rho_v)$ time by walking up the parent chain of v. If the activation rank of v is ℓ, this is efficient, however, it might be as costly as $O(\log \Delta)$ if ℓ is significantly higher than the activation rank ρ_v. In this case, the rank ℓ ancestor, say w', of the last converted satellite u of v can be located in $O(1)$ time (Lemma 3.10) and by modifying the arguments of Lemma 3.1 slightly, one can prove that w' is $c \cdot 2^{\ell-2}$ close to s. Once locating an ℓ-center (w or w') that is $c \cdot 2^{\ell-2}$ close to s, all ℓ-sites of s can be located in $O(1)$ time by checking its ℓ-neighbors.

We describe the $O(1)$ time implementations of `BallEmpty`, `Convert`, and `TryToPromote` using ℓ-sites. `BallEmpty` first finds all ℓ-sites of s, then computes the ℓ-neighbors of s in $O(1)$ time by checking the ℓ-neighbors of the cousins of an ℓ-site of s (Lemma 3.2). It then computes the ℓ'-neighbors of s for ranks $\ell' = \ell-1$ down to $\ell' = \ell-6$ by checking the ℓ'-neighbors of the children of the $(\ell'+1)$-neighbors of s. It

determines whether the certificate ball of s is empty or not by checking whether there is a neighbor of s within $2^{\ell-2}$ distance of s or not (Lemma 3.4). If there are no neighbors, the algorithm converts s by calling `Convert`; otherwise, it discards s. `Convert` inserts s into the spanner at its minimum rank λ_s and promotes it up to rank ℓ. At the end of each rank, `TryToPromote` promotes certain Steiner vertices to the next rank. For a given ℓ-center s, it uses an $(\ell+1)$-site u of s, the parent of an ℓ-site of s to be precise, to determine if an $(\ell+1)$-center is close enough to be the parent of s. It checks all $(\ell+1)$-neighbors of u; if there is an $(\ell+1)$-center within $2^{\ell+1}$ distance of s, it assigns that vertex to be the parent of s. If no such $(\ell+1)$-center exists, it includes s into $\mathsf{S}_{\ell+1}$ as an $(\ell+1)$-center. Also, it locates all $(\ell+1)$-neighbors of s by checking $(\ell+1)$-neighbors of cousins of u.

LEMMA 3.1. *For any $\ell' \geq \ell$, the rank ℓ' ancestor of the nucleus of an ℓ-satellite s is an ℓ'-site of s.*

PROOF. Let v be the nucleus of s and w be the rank ℓ' ancestor of v. Since $|sv| = 2^{\ell}$ and $|vw| \leq 2^{\ell'+1}$, by the triangle inequality, we have $|sw| < 2^{\ell'+2} < c \cdot 2^{\ell'-2}$. □

LEMMA 3.2. *Given an ℓ-satellite s, for some rank $\ell' \geq \ell$, let u be an ℓ'-site and w be an ℓ'-neighbor of s. Then, w is either a cousin of u (u's parent's neighbors' child) or an ℓ'-neighbor of one of the cousins of u.*

PROOF. Let v be the nucleus of s, u' be the parent of u and therefore an $(\ell'+1)$-site of s. If w is an input vertex, let w' be its parent, otherwise, let w' be one of its $(\ell'+1)$-sites. In both cases, $|ww'| \leq c \cdot 2^{\ell'-1}$. Our claim is proven if we can show that u' and w' are $(\ell'+1)$-neighbors. Since w is an ℓ'-neighbor of s, $|sw| \leq c \cdot 2^{\ell'}$, and since u' is an $(\ell'+1)$-site of s, $|su'| \leq c \cdot 2^{\ell'-1}$. Using the triangle inequality, we have $|u'w'| \leq c \cdot 2^{\ell'+1}$, that is, u' and w' are $(\ell'+1)$-neighbors. □

The correctness proof relies on some technical lemmas. In our main lemma, we prove that our algorithm progresses towards a well-spaced superset incrementally (Lemma 3.3). In order to prove this lemma, we show that the bottom-up processing order over the ranks ensure that querying only certain neighbors of a satellite is enough to determine whether the certificate ball of that satellite is empty or not (Lemma 3.4). Furthermore, converting, again in a bottom-up order, only the satellites with empty certificate balls guarantees that their certificate balls remain empty—we never convert a satellite that lies in the certificate ball of a converted satellite (Lemma 3.6). After proving well-spacedness, we prove that the output is size-optimal as well (Lemma 3.8).

LEMMA 3.3. *At the end of rank ℓ, all input vertices with activation ranks $\leq \ell$ and all satellites converted at ranks $< \ell$ are $\frac{9}{2}$-well-spaced.*

For the base case, at the beginning of rank λ, there are no active input vertices and no converted satellites, therefore the claim is trivially true. For the inductive hypothesis, we assume that at the end of rank $\ell-1$, all input vertices with activation ranks $\ell-1$ and below and all satellites converted at ranks $\ell-2$ and below are $\frac{9}{2}$-well-spaced. We break down the inductive proof into several steps and conclude it later.

LEMMA 3.4. *The certificate ball of an ℓ-satellite s is empty of input vertices and earlier converted satellites iff it does not contain an ℓ'-neighbor of s at ranks $\ell' \in \{\ell-6, \ldots, \ell-3\}$.*

PROOF. The only if part of the proof is trivial; if the certificate ball of s contains a neighbor, clearly, it is not empty. For the if part, for any rank $\ell' \geq \ell-2$ and any ℓ'-neighbor u of s, $|su| > 2^{\ell-2}$ because u and s are both ℓ'-centers. Therefore u cannot be inside the certificate ball of s. For the case that s has an ℓ'-neighbor u for some rank $\ell' \leq \ell-7$, we have $|su| \leq c \cdot 2^{\ell'}$. Using the triangle inequality, the rank $\ell-6$ ancestor, say u', of u satisfies $|su'| \leq c \cdot 2^{\ell'} + |uu'| < c \cdot 2^{\ell-7} + 2^{\ell-5} < 2^{\ell-2} < c \cdot 2^{\ell-6}$. Therefore u' being an $(\ell-6)$-neighbor of s would be inside the certificate ball of s. Now, given the premise of the lemma, assume towards a contradiction that a vertex not neighboring s lies inside the certificate ball and let v be the nearest one.

Since the spanner has a 2-approximation guarantee, s has a neighbor within distance $2|sv| \leq 2^{\ell-1} < c \cdot 2^{\ell-5}$, thus, s has an $(\ell-5)$-neighbor. Since v is not a neighbor of s, v cannot be among the children of $(\ell-5)$-neighbors of s. More specifically, v is not an $(\ell-6)$-center; if it were, its parent would have been an $(\ell-5)$-neighbor of s. Therefore, since v is not an $(\ell-6)$-center, we have $\mathrm{NN}(v) < 2^{\ell-6}$. Hence, for some $\ell' < \ell$, v is either an ℓ'-satellite or an input vertex with activation rank ℓ'. By the inductive hypothesis, v is $\frac{9}{2}$-well-spaced. Since v is the exact nearest neighbor of s, s lies in the Voronoi cell of v. Therefore, we have $|sv| \leq \frac{9}{2}\,\mathrm{NN}(v) < \frac{9}{2} \cdot 2^{\ell-6} < 2^{\ell-3}$. Let w be the rank $\ell-5$ ancestor of v, then $|vw| < 2^{\ell-4}$, and using the triangle inequality, we get $|sw| < 2^{\ell-2} < c \cdot 2^{\ell-5}$. Being that close, w must be an $(\ell-5)$-neighbor of s. This is a contradiction to our premise because w is in the certificate ball of s. □

In order to better explain our construction algorithm, we relate the problem of determining which satellites to be converted to the maximal independent subset problem. At a given rank ℓ, before converting any satellites, consider the set of ℓ-satellites whose certificate balls are empty. Let the *proximity graph* at rank ℓ be the undirected graph on this set with edges connecting two ℓ-satellites if and only if they are within distance $2^{\ell-2}$ from each other. Then we prove:

LEMMA 3.5. *At each rank ℓ, the algorithm converts a maximal independent subset of the proximity graph.*

PROOF. First, we prove independence. Pick any satellite s that is converted at rank ℓ. By Lemma 3.4, since we never convert a satellite before ensuring that its certificate ball is empty, s cannot be $2^{\ell-2}$ close to any existing vertex prior to its conversion. Similarly, the algorithm cannot convert an ℓ-satellite that is $2^{\ell-2}$ close to s. Thus, in the proximity graph, none of the satellites adjacent to s are converted. For maximality, consider an ℓ-satellite s, none of whose neighbors in the proximity graph is converted. This implies that s has an empty certificate ball, therefore s would be converted when the algorithm tries to insert s. □

Using the above lemma, we state a corollary that is useful in our analysis: the certificate ball of each ℓ-satellite s of each active input vertex contains a vertex, either the converted vertex s or a vertex that prohibits the conversion of s.

LEMMA 3.6. *For any rank $\ell' \leq \ell$, there is an empty ball of radius $2^{\ell'-2}$ around any converted ℓ'-satellite and around any input vertex with activation rank ℓ'.*

PROOF. First, consider a converted ℓ'-satellite s. Using Lemma 3.4, we know that at rank ℓ', the certificate ball of s

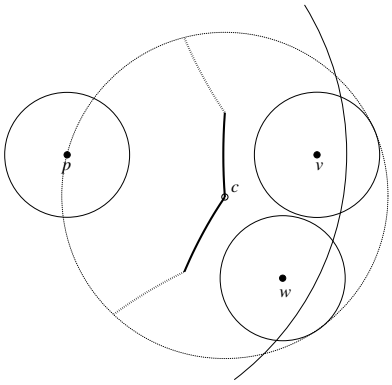

Figure 4: The points v and w are ℓ-satellites of an input vertex p. Each of the hyperbolic thick curves depicts the locus of the points whose distance to an ℓ-satellite is $2^{\ell-2}$ less than its distance to p. The Voronoi cell of p is a subset of the weighted Voronoi cell of p defined by these hyperbolic curves.

(of radius $2^{\ell'-2}$) is empty. Subsequent conversions require empty certificate balls of radius at least $2^{\ell'-2}$, thus, the certificate ball of s remains empty. Now, consider an input vertex v with activation rank ℓ'. We know that all other input vertices are at least $2^{\ell'}$ far away. Therefore, by the triangle inequality, for $k < \ell'$, all k-satellites of other input vertices are at least $2^{\ell'} - 2^k \geq 2^{\ell'-1}$ far away from v. Moreover, for $k \geq \ell'$, all converted k-satellites have empty balls of radius $2^{k-2} \geq 2^{\ell'-2}$, hence, our result follows. □

We prove the inductive step of Lemma 3.3, that all input vertices with activation rank ℓ and Steiner vertices converted at rank $\ell - 1$ are $\frac{9}{2}$-well-spaced. Figure 4 displays an input vertex p with activation rank ℓ. By the corollary to Lemma 3.5, there is a vertex inside the certificate ball of every ℓ-satellite of p. We know that the vertex q inside the certificate ball of a given ℓ-satellite, say v, is within $2^{\ell-2}$ distance of v. Considering the locus of the points whose distance to v is $2^{\ell-2}$ less than its distance to p, the collection of these hyperbolas defines a weighted Voronoi cell of p. Observe that none of the points that lies in the half that contains v can be in the Voronoi cell of p. Consequently, the weighted Voronoi cell bounds the Voronoi cell of p. The extreme points of this region are the intersections of two hyperbolas, which correspond to the circumcenters of the circles that are tangent to p and to two certificate balls on the outside. One can show that these circumcenters (e.g., c in Figure 4) lie within $9 \cdot 2^{\ell-3}$ distance of p. Then, the proof of $\frac{9}{2}$-well-spacedness follows from Lemma 3.6, that the nearest neighbor of p is at least at $2^{\ell-2}$ distance.

In order to prove the $\frac{9}{2}$-well-spacedness of the Steiner vertices, we illustrate an example in Figure 5 displaying an input vertex p, its $(\ell-1)$-satellites u, v, w, its ℓ-satellites y and z, and their certificate balls. Assuming that v is converted to a Steiner vertex, we prove that its Voronoi cell lies within the big ball $B(v, 9 \cdot 2^{\ell-4})$ displayed in the figure. Intuitively the proof considers the output vertices in each of the certificate balls of u, w, y, and z. We prove that p and these four vertices bound the Voronoi cell of v as desired. Once again, we consider the weighted Voronoi cell of v, which upper bounds the actual Voronoi cell of v, where each of the

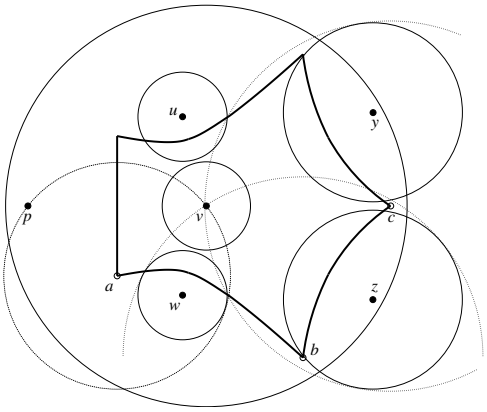

Figure 5: Illustration of the $\frac{9}{2}$-well-spacedness of a converted $(\ell-1)$-satellite v of an input vertex p. The weighted Voronoi cell defined by the thick hyperbolic curves bound the Voronoi cell of v.

satellites u, w, y, and z has weight equal to the radius of its certificate ball. Then, we prove that the corners of the weighted Voronoi cell, or the circumcenters, e.g., a, b, c, lie within $9 \cdot 2^{\ell-4}$ distance of v. Again Lemma 3.6 states that the nearest neighbor of v is at least at $2^{\ell-3}$ distance; this implies the $\frac{9}{2}$-well-spacedness of v.

LEMMA 3.7. *All Steiner vertices converted at the final rank are $\frac{9}{2}$-well-spaced.*

PROOF. We sketch the proof because of space restrictions. Given an $(\Lambda+4)$-satellite s of an input vertex v, we know $|sv| = 2^{\Lambda+4}$. Since the diameter D is less than $2^{\Lambda+2}$, only one of the satellites on the same ray as that of s can be inserted. We prove that for each of the 12 rays, the $(\Lambda+4)$-satellite of exactly one of the active vertices is converted to a Steiner vertex: any satellite s' that is converted to a Steiner vertex at rank $\Lambda+3$ is at distance $2^{\Lambda+3}$ away from its nucleus v' and by the triangle inequality, $|ss'| \geq |sv| - (|s'v'| + |v'v|) \geq 2^{\Lambda+4} - (2^{\Lambda+3} + D) > 2^{\Lambda+2}$; the certificate ball of s is empty of output vertices. Then arguments similar to those in *Lemma* 3.3 proves well-spacedness. □

LEMMA 3.8. M *is size-conforming with respect to* N.

PROOF. For any $v \in \mathsf{M}$, we show $\mathrm{NN}_{\mathsf{M}}(v) \in \Omega(\mathrm{lfs}(v))$. First, we analyze the vertices by ranks and upper bound their local feature sizes. If v is an input vertex with activation rank ℓ, we have $\mathrm{lfs}(v) = \mathrm{NN}_{\mathsf{N}}(v) \in O(2^{\ell})$. If v is an ℓ-satellite with nucleus u, by the Lipschitz condition, we have $\mathrm{lfs}(v) \leq \mathrm{lfs}(u) + |uv|$. Since the activation rank of u is at most ℓ, we have $\mathrm{lfs}(u) \in O(2^{\ell})$, and since $|uv| = 2^{\ell}$, we deduce that $\mathrm{lfs}(v) \in O(2^{\ell})$. In both cases, by Lemma 3.6, we know that $\mathrm{NN}_{\mathsf{M}}(v) \geq 2^{\ell-2}$, therefore, our result follows. □

After the algorithm constructs a $\frac{9}{2}$-well-spaced size-optimal superset M of the input N, it constructs the Delaunay triangulation of M. Let $s \in \mathsf{M}$ be either an input vertex with activation rank ℓ or a Steiner vertex converted at rank ℓ. Using Lemma 3.9 the algorithm generates a candidate set by iterating through each neighbor of s at ranks $\{\ell-6, \ldots, \ell-3\}$ and checking if it lies within $9 \cdot 2^{\ell-2}$ distance of s. With this candidate set (of constant size), the algorithm applies half-space tests to determine exactly which of these candidates are indeed Voronoi neighbors of s.

LEMMA 3.9. *Let s be a converted ℓ-satellite or an input vertex with activation rank ℓ. If v is a Voronoi neighbor of s in M, v and s are ℓ'-neighbors at a rank $\ell' \in \{\ell-6, \ldots, \ell-3\}$.*

PROOF. By Lemma 3.6, we know that there is an empty ball of radius $2^{\ell-2}$ around s. Since s is $\frac{9}{2}$-well-spaced, the Voronoi cell of s must be confined in a ball of radius $\frac{9}{2} \cdot 2^{\ell-2}$, which implies $|vs| \leq 9 \cdot 2^{\ell-2}$. Because of the 2-approximation guarantee, both v and s have neighbors in the spanner within $18 \cdot 2^{\ell-2}$ distance. Thus, the minimum ranks of v and s are less than $\ell - 1$. Because of the empty ball, s is an $(\ell-2)$-center and a discrete center at a lower rank ℓ' as long as s has an ℓ'-neighbor. Since $9 \cdot 2^{\ell-2} \geq |vs| > 2^{\ell-2}$, there is some $k \in \{\ell-6, \ldots, \ell-3\}$ such that $c \cdot 2^k \geq |vs| > c \cdot 2^{k-1}$. Then the Voronoi cell of v includes a point at least $c \cdot 2^{k-2}$ far from v. Since v is $\frac{9}{2}$-well-spaced, there is an empty ball of radius at least $\frac{2}{9} \cdot c \cdot 2^{k-2} > 2^k$ around v. Therefore, v is a k-center and consequently a k-neighbor of s. □

LEMMA 3.10. *Considering a converted ℓ-satellite s, its minimum rank λ_s satisfies $\ell - 6 \leq \lambda_s \leq \ell - 3$.*

PROOF. By definition of λ_s, s has a λ_s-neighbor u within distance $c \cdot 2^{\lambda_s}$. Since s is converted to a Steiner vertex, its certificate ball must be empty. Thus, $2^{\ell-2} < |su| < c \cdot 2^{\lambda_s}$, that is, $2^\ell/4c < 2^{\lambda_s}$. This implies $\ell - 7 < \lambda_s$. For the upper bound, observe that the nucleus of s is 2^ℓ away from s. Again, by definition of λ_s, we know that any λ_s-neighbor of s is at least $c \cdot 2^{\lambda_s - 1}$ away from s. Gao et al. proves that the nearest of those λ_s-neighbors provides a 2-approximation for the exact nearest neighbor of s [14]. Since the nucleus of s is at 2^ℓ distance, we have $c \cdot 2^{\lambda_s - 1} \leq 2 \cdot 2^\ell$, that is, $2^{\lambda_s} \leq \frac{4}{c} \cdot 2^\ell$. We conclude, $\lambda_s < \ell - 2$. □

LEMMA 3.11. *For any rank ℓ, the number of ℓ-centers in the spanner is bounded by $O(n)$, i.e., $|\mathsf{S}_\ell| \in O(n)$.*

PROOF. By Lemma 3.10, if an ℓ'-satellite s of an input vertex v is converted to a Steiner vertex, its minimum rank in the spanner is at least $\ell' - O(1)$. In particular, for s to be an ℓ-center, ℓ' must be bounded by $\ell + O(1)$, in other words, $|sv|$ must be bounded by $O(2^\ell)$. Since two ℓ-centers are at least 2^ℓ far apart, a packing argument shows that there can be at most $O(1)$ satellites of v that are ℓ-centers. The proof follows using the fact that there are n input vertices. □

THEOREM 3.12. *Our algorithm constructs a $\frac{9}{2}$-well-spaced, size-optimal superset M of the input N, and computes the Delaunay triangulation of M in $O(n \log \Delta)$ time.*

PROOF. The quality proof follows from the Lemmas 3.3 and 3.7. Lemma 3.8 shows that the output is size-conforming, which is sufficient to prove size optimality [22]. Finally, Lemma 3.9 proves that the algorithm correctly computes the Voronoi diagram, i.e., the Delaunay triangulation of the superset M. Runtime proof follows from previous discussions based on Lemmas 3.10 and 3.11. □

4. DYNAMIC & KINETIC MAINTENANCE

We present an update algorithm that, given a dynamic modification or a kinetic event, updates the set of Steiner vertices and the corresponding Delaunay triangulation so that the output remains to be a quality, size-optimal triangulation. The update algorithm is a change-propagation algorithm: it maintains a set of affected satellites for each rank

```
Update the spanner S restricted to N
for each rank ℓ, F_ℓ = P_ℓ = ∅
for each affected input vertex v
  ρ'_v ← ρ_v, ρ_v ← λ_v (in S restricted to N)
  if ρ_v ≠ ρ'_v then
    for each rank ℓ between ρ_v and ρ'_v (inclusive)
      F_ℓ ← F_ℓ ∪ {ℓ-satellites of v}
for each violated ball-empty/ball-not-empty certificate
  F_ℓ ← F_ℓ ∪ {s}, where s is the ℓ-satellite involved
for each affected Steiner vertex s
  P_ℓ ← P_ℓ ∪ {s}, where ℓ is the rank of affection

for each rank ℓ with F_ℓ ∪ P_ℓ ≠ ∅
  for each s ∈ F_ℓ ∩ S Remove(S, s)
  for each s ∈ F_ℓ
    v ← nucleus of s
    if (ρ_v = ℓ or ∃ a converted (ℓ−1)-satellite of v)
      and BallEmpty(S, s, 2^{ℓ−2}) then Convert(S, s, ℓ)
  for each nucleus v of each satellite s ∈ F_ℓ
    if an ℓ- or (ℓ+1)-satellite of v is converted then
      F_{ℓ+1} ← F_{ℓ+1} ∪ {(ℓ+1)-satellites of v}
  S_{ℓ+1} ← UpdatePromotion(F_ℓ ∪ P_ℓ)

for each rank ℓ and each affected s ∈ N with ρ_s = ℓ
                or each ℓ-satellite s ∈ M ∩ (F_ℓ ∪ P_ℓ)
  V ← ∪_{ℓ'∈{ℓ−6,...,ℓ−3}} ℓ'-neighbors of s
  IdentifyVoronoiNeighbors(s, V)
```

Figure 6: Psuedo-code for the update algorithm.

and repairs each rank consecutively. The algorithm therefore is trivially $O(\log \Delta)$ "stable" at the level of abstraction of the ranks, i.e., it has to repair only $O(\log \Delta)$ ranks to update the output. To bound the total update runtime, we show that there are $O(1)$ affected satellites at each rank (Lemma 4.5) and that repairing each rank requires amortized $O(1)$ time, yielding our final bound on responsiveness (Theorem 4.7).

As required by the KDS framework, we certify both the spanner structure and our extension of it. For the spanner, we use the same set of certificates (parent-child, edge, separation, and potential neighbor) used by Gao et al. [14]. For the extension, we generate our own certificates. We certify the conversion decisions for each ℓ-satellite s considered. If the certificate ball of s is not empty, we generate a *ball-not-empty* certificate to certify that a vertex $v \in \mathsf{M}$ lies inside $B(s, 2^{\ell-2})$. Otherwise, we generate a *ball-empty* certificate to certify that $v \notin B(s, 2^{\ell-2})$ for each ℓ'-neighbor v of s for $\ell' < \ell$. For certifying the Delaunay triangulation, we observe by Lemma 3.9 that, for each converted ℓ-satellite s, neighbors of s at ranks $\{\ell-6, \ldots, \ell-3\}$ constitute its potential Voronoi neighbors. We certify the halfspace tests performed for determining the Voronoi cell of s considering this candidate set. These discussions allow us to state:

LEMMA 4.1. *The certificates generated by our construction algorithm certifies that the output is the Delaunay triangulation of a size-optimal and well-spaced superset of the given input set.*

Upon a certificate failure or a dynamic modification, the update algorithm whose pseudo-code is shown in Figure 6 updates the output and the set of certificates. Following the structure of the construction algorithm, in three stages, it repairs the proximity graph and updates the maximal independent subset of satellites chosen at each rank. It keeps two sets of vertices at each rank for affected satellites. The

first set, $\mathcal{F}$, tracks the satellites that may be required to be removed or converted to Steiner vertices, we call them *fully affected*. The second set, $\mathcal{P}$, tracks the satellites which are previously converted and whose ball-empty certificates remain unaffected. These satellites, which we call *partially affected*, are not required to be removed; however, we may need to promote or demote these vertices in the spanner.

In the first stage, the update algorithm updates the spanner data structure restricted to the input vertices using the update procedure described in Section 2.3. It updates the diameter and the activation ranks of the input vertices if needed. Each of these updates may partially/fully affect certain satellites. The algorithm then initializes $\mathcal{F}_\ell$ and $\mathcal{P}_\ell$ lists. For each input vertex whose activation rank has changed, it marks all satellites at ranks between the previous and current activation ranks as fully affected as these satellites may need to be removed or considered for conversion. Also, for Steiner vertices, if any of the ball-empty or ball-not-empty certificates has failed, it marks the corresponding satellite fully affected. It marks all other satellites related to failed certificates as partially affected at the rank at which the certificate is defined.

In the second stage, the update algorithm iterates through each rank, updating the structure in three phases: *remove*, *convert*, and *repair*. In the remove phase, it removes the fully affected satellites from the spanner by calling `Remove`, which removes a satellite s in a bottom-up pass starting from its minimum rank λ_s until its maximum rank Λ_s by removing it from the neighbors, parent, and child lists. This function runs in $O(\Lambda_s - \lambda_s)$ time. In the convert phase, similar to the construction algorithm, the update algorithm tries to convert all fully affected satellites (including the ones that are removed earlier) to Steiner vertices, provided that their nuclei are active and their certificate balls are empty. Finally, in the repair phase, the update algorithm repairs the discretization at the next rank by updating the promotion decisions for the affected Steiner vertices using `UpdatePromotion`, which performs the discretization step only on the affected Steiner vertices. This function takes linear time in the number of affected Steiner vertices, which we prove to be of constant size. After the update algorithm is done with the second stage, it updates the Delaunay triangulation in the third stage by computing the up-to-date Voronoi neighbors of the affected vertices.

If an update affects a ball-empty/ball-not-empty certificate of an ℓ-satellite s, it enqueues s into the $\mathcal{F}_\ell$ list. The only exception to this rule is that the update algorithm never enqueues a satellite into $\mathcal{F}$ lists if its ball-empty certificate is affected during a call to `Remove`. If an update affects an edge/potential-neighbor certificate of an Steiner vertex s at rank ℓ, the algorithm enqueues s into the $\mathcal{P}_\ell$ list. Similarly, if an update affects a parent-child/separation certificate of two vertices at consecutive/same ranks, it enqueues these vertices in $\mathcal{P}_\ell$ ($\mathcal{P}_{\ell+1}$) list. Finally, if an update affects a Voronoi certificate, it enqueues the ℓ-satellite for which the Voronoi cell is being computed to $\mathcal{P}_\ell$ list. Based on our update algorithm, we state following lemma without a proof.

LEMMA 4.2. *After the iteration at rank ℓ, the spanner contains a maximal independent subset of the satellites in the up-to-date proximity graph at rank ℓ.*

LEMMA 4.3. *At rank ℓ, the number of active vertices in a ball of radius $O(2^\ell)$ is bounded by $O(1)$.*

PROOF. A vertex v inside the given ball may be active for two reasons. Either its activation rank is ℓ or one of its satellites at rank $\ell - 1$ is converted. In the first case, we bound the nearest neighbor distance of v by $\Omega(2^\ell)$. Thus, a packing argument bounds the number of vertices that fall into the first case by $O(1)$. In the second case, let s be a converted $(\ell-1)$-satellite of v. Then, there is an empty ball around s with radius $\Omega(2^\ell)$. At a fixed rank and ray, all satellites are shifted versions of input vertices. Therefore, if the $(\ell-1)$-satellite on the same ray (same polar angle) of another vertex u is converted, we have $|uv| \in \Omega(2^\ell)$. Similar packing arguments bound the number of such vertices by $O(1)$ for each ray r. Since there is a constant number of rays, the result follows. □

In order to bound the runtime of our updates, we define the *focus* of the dynamic update or the kinetic event as one of the input vertices and prove that all modifications take place around the focus. For a dynamic modification the focus is the vertex being inserted or deleted. For a kinetic event, consider the vertices/satellites involved in the certificate failure. Representing the satellites with their nuclei and input vertices by themselves, the focus is any of the two representations of the points involved in the certificate.

LEMMA 4.4. *Consider a kinetic event or a dynamic modification and let p be the focus. For any rank ℓ, any satellite in $\mathcal{F}_\ell$ or $\mathcal{P}_\ell$ lists is $O(2^\ell)$ away from p.*

PROOF. We sketch the proof this lemma. We use induction over the order in which the algorithm inserts satellites into these lists. For the base case, we consider the affected lists at the end of the first stage. Fix rank ℓ and consider a satellite s satisfying the premises. Construction and the update algorithms of Gao et al. perform local operations at each rank. More specifically, given the focus p, if the update removes or inserts an edge in the spanner at rank ℓ, the vertices incident to the edge are within $O(2^\ell)$ distance of p. Therefore, if s is enqueued into these lists while repairing the initial spanner structure then $|sp| < O(2^\ell)$. Similar to the base case, we prove that if applying an update on a satellite s_1 affects another satellite s_2 and s_2 is either fully affected at or partially affected at rank ℓ', then $|s_1 s_2| < O(2^{\ell'})$. Geometrically relating the affected satellites defines a dependence path from each affected satellite to the focus p. Except for a constant overhead, all the effects grow in ranks, therefore, using the triangle inequality on any of the paths from p to s, we can bound the distance between p and s by a geometric series with constantly many repetitions of the terms. Using the fact that a geometric series is dominated by its last term, and the fact that the last term is $O(2^\ell)$, we conclude our result. □

LEMMA 4.5. *For any rank ℓ, there are $O(1)$ satellites in $\mathcal{F}_\ell$ and $\mathcal{P}_\ell$ lists.*

PROOF. Let s be an affected satellite, by Lemma 4.4, s lies inside a ball of radius $O(2^\ell)$ around the focus p. We analyze the two cases, $s \in \mathcal{F}_\ell$ and $s \in \mathcal{P}_\ell$ and prove that in each case there is a constant number of such satellites. For the first case, assume that $s \in \mathcal{F}_\ell$, i.e., s is an ℓ-satellite of an active vertex. The nucleus lies inside a slightly larger ball around p of radius $O(2^\ell)$. Lemma 4.3 proves that there are only $O(1)$ many vertices; since a vertex has $O(1)$ satellites at a given rank, we conclude the first case. For the second

case, assume that $s \in \mathcal{P}_\ell$, i.e., s is an ℓ-center. Therefore, s is at least 2^ℓ far from another ℓ-center. Since s is $O(2^\ell)$ close to p, a packing argument bounds the number of such vertices by $O(1)$ as well. □

LEMMA 4.6. *The total runtime of* `Remove` *calls made by the update algorithm is* $O(\log \Delta)$.

PROOF. By Lemma 4.4, for each rank ℓ, all of the fully affected satellites are within $O(2^\ell)$ distance of the focus p. The runtime required to remove these affected satellites is $T < \sum_{\ell=\lambda}^{\Lambda} \sum_{s \in \mathcal{F}_\ell} \alpha \cdot (\Lambda_s - \lambda_s)$, where λ and Λ are the minimum and the maximum rank of the spanner and α is the constant in the big-Oh notation hidden in the runtime of the `Remove` function. Using Lemma 3.10 to bound λ_s, we rewrite $T < O(\log \Delta) + \alpha \cdot \sum_{\ell=\lambda}^{\Lambda} \sum_{s \in \mathcal{F}_\ell} (\Lambda_s - \ell)$. Rearranging, we get $T < O(\log \Delta) + \alpha \cdot \sum_{\ell=\lambda}^{\Lambda} \sum_{\ell'=\ell}^{\Lambda} |\{s \mid s \in \mathcal{F}_\ell, \Lambda_s > \ell'\}| = O(\log \Delta) + \alpha \cdot \sum_{\ell'=\lambda}^{\Lambda} \sum_{\ell=\lambda}^{\ell'} |\{s \mid s \in \mathcal{F}_\ell, \Lambda_s > \ell'\}|$. For any given ℓ', we claim that $\sum_{\ell=\lambda}^{\ell'} |\{s \mid s \in \mathcal{F}_\ell, \Lambda_s > \ell'\}|$ is bounded by $O(1)$: for any $\ell \leq \ell'$, the Steiner vertices in $\mathcal{F}_\ell$ are within $O(2^{\ell'})$ distance of the focus p and any vertex with $\Lambda_s > \ell'$ is an ℓ'-center; each of them has an empty ball of radius $2^{\ell'}$. Then, a packing argument proves our claim and we get $T = O(\log \Delta)$. □

THEOREM 4.7. *Given a kinetic event or a dynamic modification, the update algorithm repairs the spanner structure and the well-spaced, size-optimal superset in* $O(\log \Delta)$ *time.*

PROOF. At each rank, ensuring that the output contains a maximal independent subset proves well-spacedness (Lemmas 3.3, 3.7, and 4.2). Since we update the minimum ranks of affected input vertices, we can apply Lemma 3.8 to ensure size-optimality as well. Lemma 4.5 bounds the total number of affected satellites by $O(\log \Delta)$. Processing each of the affected satellites takes $O(1)$ time except for removal. Lemma 4.6 bounds the total runtime of the removal of the satellites by $O(\log \Delta)$. Thus, our result follows. □

5. QUALITY OF THE KDS

In order to prove the efficiency of our kinetic data structure we show that our KDS is responsive, local, compact, and efficient. We proved responsiveness in the previous section. In this section, we prove that our KDS is compact and local, i.e., it has near linear total number of certificates, and each input vertex participates in a logarithmic number of certificates. Then, we prove that our KDS is efficient, i.e., there are not too many certificate failures compared to the number of combinatorial changes required in the worst case.

LEMMA 5.1. *Every input vertex participates in* $O(\log \Delta)$ *certificates. In total, our kinetic data structure maintains* $O(m)$ *certificates, where* m *is* $|\mathsf{M}|$.

PROOF. Consider an input vertex v and fix a rank ℓ. In Lemma 3.11, we prove that v has $O(1)$ converted satellites that are ℓ-centers in the final spanner. Since there are $O(\log \Delta)$ ranks and since each of the ℓ-centers is associated with a constant number of certificates, each input vertex, through its satellites, participates in $O(\log \Delta)$ certificates. For the total number of certificates, we already know that the resulting spanner has $O(m)$ neighbor edges and that there are $O(m)$ ball-empty certificates. We are left to prove that there are $O(m)$ ball-not-empty certificates. For each

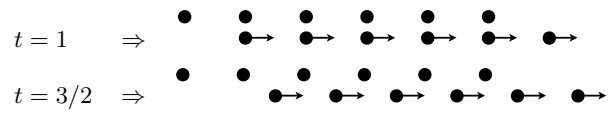

Figure 7: Consider a horizontal line of $2k$ evenly-spaced vertices: $(0,0)$, $(1,0)$, $\ldots (2k,0)$, and a second line ϵ above the first line: $(0,\epsilon)$, $(1,\epsilon)$, $\ldots (2k,\epsilon)$. Assign a fixed velocity vector $(1,0)$ to the vertices of the lower line. The upper line does not move.

satellite that is not converted, we charge its ball-not-empty certificate to an inserted satellite at the previous rank or if no such satellite exists to its nucleus. Using this approach, every Steiner vertex or input vertex is charged at most $O(1)$ ball-not-empty certificates. There are $O(m)$ vertices in the output, therefore, the result follows. □

We know that each input vertex may introduce $O(\log \Delta)$ many satellites, a total of $O(n \log \Delta)$. Most of our certificates are distances between pairs of vertices and there are $O(\log \Delta)$ different distances we consider, therefore, our algorithm processes a total of at most $O(n^2 \log^3 \Delta)$ events through these types of certificate failures. For certifying the Delaunay triangulation, by the above analysis, we know that the candidate Voronoi neighbor set of a given Steiner vertex changes $O(n \log^2 \Delta)$ many times. There are $O(1)$ halfspace tests performed to determine the exact Voronoi cell, hence, our algorithm processes a total of at most $O(n^2 \log^3 \Delta)$ events altogether. We show that there exist examples for which maintaining a size-optimal well-spaced point set requires $\Omega(n^2 \log \Delta)$ combinatorial changes. Specifically, consider the example shown in Figure 7 with $n = 4k$ points and $\epsilon = 1/k$ and let time evolve from $t = 0$ to k. The diameter is $\Theta(n)$ and the minimum pairwise distance oscillates between $\Theta(1)$ at half-integer times and $\Theta(\epsilon)$ at integer times. The spread is therefore $\Delta \in \Theta(n/\epsilon)$, which implies $\log \Delta \in \Theta(\log \epsilon^{-1})$.

LEMMA 5.2. *At integer times, a well-spaced superset requires* $\Omega(n \log \Delta)$ *Steiner vertices to be inserted.*

PROOF. Ruppert proves [22] that even the smallest well-spaced superset of an input has $\int_\Omega \mathrm{lfs}^{-d}(x)dx$ vertices, where Ω denotes the domain. To bound the integral, for each pair (u_i, v_i) of ϵ-close vertices, take their midpoint p_i. We define a set of non-overlapping balls $B(p_i, 1/2)$ for each of the at least k pairs of ϵ-close vertices. The integral over all of Ω is lower-bounded by the sums of the integrals over each ball:

$$\int_\Omega \mathrm{lfs}^{-d}(x)dx \geq \sum_{i=1}^{k} \int_{B(p_i,1/2)} \mathrm{lfs}^{-d}(x)dx$$

At the midpoint, $\mathrm{lfs}(p_i) = \epsilon/2$. Because lfs is 1-Lipschitz, $\mathrm{lfs}(x) \leq |p_i x| + \epsilon/2$. Then the integral over each ball is at least $\Omega(\log \epsilon^{-1})$. Since $k = n/4$, the sum is $\Omega(n \log \epsilon^{-1})$. □

LEMMA 5.3. *At half-integer times, inserting* $O(n)$ *Steiner vertices is sufficient for the lower bound example of Figure 7.*

PROOF. This proof requires some results from the field of curve reconstruction. We refer to the book by Dey [11] for an introduction. The key result we establish is that the input vertices form a good sample of a smooth curve: it is sufficiently dense so that no point of the curve is too far from a vertex, but sufficiently sparse so that no two vertices

are too close to each other. Density and sparsity are both relative to the distance from the curve to its medial axis. We can fit a sinusoidal curve through the vertices as they are arranged at half-integer times; the curve has amplitude $\epsilon/2$. Any point on the curve is at distance less than $1/2$ from an input vertex. Meanwhile, the medial axis of the curve is at distance $\frac{1}{8\epsilon} + \frac{\epsilon}{2} = \Theta(1/\epsilon)$ from the curve. Thus the vertices form an $O(\epsilon)$-dense sample of the curve. Vertices are at distance $\sqrt{\epsilon^2 + 1/4} \approx 1/2$ from each other, thus they form an $\Omega(\epsilon)$-sparse sample. Hudson, Miller, Phillips and Sheehy [19] show that such a vertex set has a well-spaced superset of size $O(n)$. □

LEMMA 5.4. *The lower bound example of Figure 7 requires $\Omega(n^2 \log \Delta)$ Steiner vertex insertions and deletions.*

PROOF. At integer times, the input requires $\Omega(n \log \Delta)$ Steiner vertices to be made well-spaced. At half-integer times, the input requires that there be no more than $O(n)$ Steiner vertices to be size-optimal. Thus, any algorithm that maintains a size-optimal well-spaced superset must add and subsequently remove $\Theta(n \log \Delta)$ vertices for every unit of time, a total of $\Omega(n^2 \log \Delta)$ such changes. □

6. CONCLUSIONS

This paper presents an effective kinetic data structure for kinetic mesh refinement in the plane. Our approach is inspired by self-adjusting-computation techniques and critically relies on deformable spanners [14]. Our bounds refer to the spread Δ, which *a priori* has no relationship to the input size n. However, if the points form an ϵ-net of a manifold, then the spread is at worst linear in n. This is because in an ϵ-net, no point of the manifold is further than ϵ from an input point, which bounds the diameter by $O(n\epsilon)$, and no two input points are at distance $o(\epsilon)$ from each other, which bounds the closest pair by $\Omega(\epsilon)$. Furthermore, the output size m is in $O(n)$ [19]. In other words, if points are a sample taken from a moving manifold, then our KDS has linear size and is efficient in the usual sense of being within a polylog(n) factor of optimal.

Our result applies only to the planar case, though it is very promising for arbitrary-dimension extension. The two missing pieces are: (1) a definition of satellites in higher dimensions, which will let us kinetically maintain well-spacing; (2) a kinetization of a method to convert a well-spaced point cloud into a quality mesh (e.g. [10]), since the direct correspondence between well-spacing and Delaunay mesh quality only applies in two dimensions.

7. REFERENCES

[1] U. A. Acar, G. E. Blelloch, M. Blume, R. Harper, and K. Tangwongsan. An experimental analysis of self-adjusting computation. *ACM Trans. Prog. Lang. Sys.*, 32(1):3:1–3:53, 2009.

[2] U. A. Acar, G. E. Blelloch, R. Ley-Wild, K. Tangwongsan, and D. Türkoğlu. Traceable data types for self-adjusting computation. In *Programming Language Design and Implementation*, 2010.

[3] U. A. Acar, G. E. Blelloch, K. Tangwongsan, and D. Türkoğlu. Robust kinetic convex hulls in 3D. In *European Symposium on Algorithms*, September 2008.

[4] U. A. Acar, A. Cotter, B. Hudson, and D. Türkoğlu. Dynamic well-spaced point sets. In *Symposium on Computational Geometry*, 2010.

[5] U. A. Acar, B. Hudson, G. L. Miller, and T. Phillips. SVR: Practical engineering of a fast 3d meshing algorithm. Oct 2007. Proceedings of the Sixteenth International Meshing Roundtable (IMR).

[6] P. K. Agarwal, J. Gao, L. J. Guibas, H. Kaplan, V. Koltun, N. Rubin, and M. Sharir. Kinetic stable delaunay graphs. In *Symposium on Computational Geometry*, 2010.

[7] P. K. Agarwal, Y. Wang, and H. Yu. A two-dimensional kinetic triangulation with near-quadratic topological changes. *Discrete & Computational Geometry*, 36(4):573–592, 2006.

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

[9] M. Bern, D. Eppstein, and J. R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, 1994.

[10] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver Exudation. *Journal of the ACM*, 47(5):883–904, 2000.

[11] T. K. Dey. *Curve and Surface Reconstruction*. Cambridge Univ. Press, 2006.

[12] H. Edelsbrunner. Triangulations and meshes in computational geometry. *Acta Numerica*, 9:133–213, 2000.

[13] J.-J. Fu and R. C. T. Lee. Voronoi diagrams of moving points in the plane. In *FST and TC 10: 10th Conf. on Found. of soft. tech. and Theor. comp. sci.*, pages 238–254, New York, NY, USA, 1990.

[14] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Applications*, 35(1):2–19, 2006.

[15] L. J. Guibas. Kinetic data structures: a state of the art report. In *Workshop on algo. found. of robotics*, pages 191–209, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[16] L. J. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In *17th Intl. Workshop Graph-Theoretic Concepts Computer Science*, pages 113–209. Springer-Verlag, Inc., 1992.

[17] M. A. Hammer, U. A. Acar, and Y. Chen. CEAL: a C-based language for self-adjusting computation. In *Programming Language Design and Imp.*, June 2009.

[18] B. Hudson, G. L. Miller, and T. Phillips. Sparse Voronoi Refinement. In *15th International Meshing Roundtable*, pages 339–356, 2006.

[19] B. Hudson, G. L. Miller, T. Phillips, and D. R. Sheehy. Size complexity of volume meshes vs. surface meshes. In *Symposium on Discrete Algorithms*, 2009.

[20] H. Kaplan, N. Rubin, and M. Sharir. A kinetic triangulation scheme for moving points in the plane. In *Symposium on Computational Geometry*, 2010.

[21] R. Ley-Wild, U. A. Acar, and M. Fluet. A cost semantics for self-adjusting computation. In *Principles of Programming Languages*, 2009.

[22] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[23] D. Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, August 1997.