Kinetic Mesh Refinement in 2D

Umut A. Acar* Benoît Hudson† Duru Türkoğlu‡

Abstract

Mesh refinement is an essential step in many applications such as surface reconstruction, physical simulations, more broadly in scientific computing, graphics, etc. Kinetic mesh refinement is the problem of maintaining a quality mesh of continuously moving points; this problem has been open. In this paper, we provide a kinetic data structure (KDS) that maintains 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; it has linear output-sensitive memory consumption. It is responsive; it repairs itself in $O(\log \Delta)$ time per kinetic event. Also, it is efficient; in the worst case it processes $O(n^2 \log^3 \Delta)$ events, which is optimal up to a polylogarithmic factor. Here, Δ is the geometric spread of the input, the ratio of the diameter to the closest pair distance. Furthermore, our KDS is highly local; in addition to kinetic events, it also handles dynamic modifications, insertions and deletions of input points. Updating the structure after a dynamic modification takes $O(\log \Delta)$ time. To the best of our knowledge, this is the first KDS for mesh refinement.

1 Introduction

The idea behind mesh refinement is to break up a physical domain into discrete elements, e.g., triangles, so that properties of the domain may be computed by considering the discrete elements. For this approach to be effective, we need the elements constituting a mesh possess certain quality properties, e.g., no small angles, which we achieve by inserting additional points, called *Steiner* points, into the mesh. An important instance of this problem, known as *kinetic mesh refinement*, concerns computation of meshes of continuously moving points.

Our approach to kinetic mesh refinement is motivated by the well-spaced point sets problem. Formally, 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 [13]. In two dimensions, well-spaced point sets relate strongly to meshing: a well-spaced point set induces a Delaunay triangulation with no small angles and the Delaunay triangulation of a well-spaced point set can be computed efficiently by performing local computations only. Therefore, as a first step, we focus on the well-spaced point set problem, which we define as the construction of a well-spaced output that is a superset of a given set of input points. An important criterion in mesh refinement is to insert as few Steiner points as possible. The output is *size-optimal* if its size is within a constant factor of the size of the smallest well-spaced superset.

We assume that the input points move along algebraic trajectories of constant degree and analyze the problem within the Kinetic Data Structures (KDS) framework introduced by Basch et al [3, 9]. In general, a KDS maintains a geometric structure and a set of *certificates*, which certifies the validity of the geometric structure being maintained. When a certificate fails, the KDS repairs itself and updates the geometric structure, consequently, updates the set of certificates that certifies the new structure. A good KDS has four properties: (i) responsiveness, that is, it responds to a certificate failure efficiently; (ii) compactness, that is, it generates a small set of certificates; (iii) locality, that is, each point is associated with a small number of certificates so that the KDS can respond to motion plan changes quickly; (iv) efficiency, that is, the number of events that the KDS processes is only a polylogarithmic factor more than the worst case maximum number of topological changes in the geometric structure.

The mesh refinement problem has been studied since the 1980s [4, 6]. Although a lot of progress has been made on meshing static, non-moving point sets, to the best of our knowledge, there is no solution for kinetic mesh refinement, which is efficient and has quality guarantees. For example, for maintaining the Delaunay triangulation of a given set of points (without inserting Steiner points), the best proven upper bound on the number of topological changes is nearly cubic [7, 10], while the lower bound is nearly quadratic. On the other hand, there are recently proposed efficient triangulation schemes that suffer from the quality criteria [1, 2, 12].

For mesh refinement in the static case, two efficient approaches have been proposed for selecting Steiner points.

*Max-Planck Institute for Software Systems
†Autodesk Inc.
‡University of Chicago

The first approach, based on *balanced quadtrees*, generates an appropriately refined quadtree over the input points and adds the corners of the quadtree squares [5]. The second approach, based on *Voronoi diagrams*, maintains a Voronoi diagram of the set of points and iteratively adds Steiner points selected from the set of Voronoi nodes [11]. Both of these approaches run in $O(n \log \Delta)$ time, where Δ is the ratio of the diameter to the closest pair distance in the input. Naively kinetizing previous meshing algorithms fails to provide an efficient solution. In the case of quadtree techniques, the main problem is the quadtree. Since a quadtree partitions the space into fixed subspaces, two points at distance ϵ away from each other, moving along parallel linear trajectories, will cross into a new quadtree cell after moving a distance of only $\Theta(\epsilon)$, generating an arbitrary number of events. In the case of Voronoi diagram based techniques, the problem is the representation of the trajectories of Steiner points. The location of a Steiner point, being placed at the circumcenter of a simplex, depends on three points some of which may themselves be Steiner points. Thus the location of a Steiner point can have as large as polynomial complexity failing to meet responsiveness and locality requirements.

We propose a KDS for maintaining the Delaunay triangulations of size-optimal well-spaced point sets under motion. We use the *deformable spanners* of Gao, Guibas, and Nguyen [8] as a point location structure and extend it to support faster insertions. To ensure that the trajectories of Steiner points can be represented efficiently, we use a picking strategy where the coordinates of a Steiner point depend only on one input point. We provide a construction algorithm that first constructs a well-spaced superset of its input and then determines the Delaunay triangulation of this superset. We propose an update algorithm for maintaining the structure under motion. Using these algorithms, we prove that our KDS yields triangulations of size-optimal well-spaced point sets and satisfies the criterion which determines the quality of a KDS. Assuming that the geometric spread stays within a constant factor of the initial geometric spread, updating after a certificate failure takes $O(\log \Delta)$ time (responsiveness). In total, we process $O(n^2 \log^3 \Delta)$ events in the worst case, whereas there exist examples where $\Omega(n^2 \log \Delta)$ points must be added and removed causing that many events (efficiency). We store $O(\log \Delta)$ certificates per point (locality), a total of $O(m)$ (compactness), where m is the size of the output, which is bounded by $O(n \log \Delta)$. Our structure also handles dynamic modifications to the input, insertion and deletions of input points. We show that the update algorithm employed in our KDS updates the mesh in $O(\log \Delta)$ time per insertion/deletion.

2 Overview

Preliminaries. Given a set of points N, we define the *geometric spread* (Δ) to be the ratio of the diameter to the distance between the closest pair in N. To define a notion of quality, we introduce a *domain* Ω to be a ball centered at an arbitrary input point with radius at least four times the diameter of the given input N. We use the term *point* to refer to any point in Ω and the term *vertex* to refer to the input and output points. Given a vertex set $\mathcal{M}$, the *nearest-neighbor distance of v in $\mathcal{M}$*, written $\mathrm{NN}_{\mathcal{M}}(v)$, is the distance from v to the nearest other vertex in $\mathcal{M}$. The *Voronoi cell of v in $\mathcal{M}$*, written $\mathrm{Vor}_{\mathcal{M}}(v)$, consists of points $x \in \Omega$ such that for all $u \in \mathcal{M}, |ux| \geq |vx|$. A vertex v is *ρ-well-spaced* if the Voronoi cell of v is contained in the ball with radius $\rho\,\mathrm{NN}_{\mathcal{M}}(v)$ centered at v; $\mathcal{M}$ is ρ-well-spaced if every vertex in $\mathcal{M}$ is ρ-well-spaced. Figure 1 illustrates these definitions.

To solve the kinetic mesh refinement problem in two dimensions, we consider an exponentially distributed set of potential Steiner vertices surrounding each input vertex, use the deformable spanners (Gao et al. [8]) as a point location data structure for proximity computation, and a greedy strategy for decimating output vertices so that the remaining vertices are well-spaced. Once we obtain a well-spaced output, we take advantage of the well-spacedness property and use the point location structure to identify the Voronoi neighbors of each output vertex, i.e., the Delaunay edges between the output vertices.

Steiner Vertex Selection. Steiner vertex selection in prior meshing algorithms has typically been based either on a global scheme (as in quadtree-based refinement), or on the shape of Voronoi cells (as in Voronoi refinement). Neither of these approaches fit into the kinetic setting well. The main difficulty is to generate a well-spaced superset that is in motion in a way that matches the input. We propose a solution using a local template approach. We assign an infinite number of potential Steiner vertices, which we call *satellites*, to every input vertex. For a satellite u of v, we say that v is the *nucleus* of u. The satellites are fixed to the local coordinate frame of their nucleus: the satellites' position curves are identical to their nucleus' position curve, plus a fixed translation (shift). In addition, we require the template to be well-spaced, so that if we create an infinite point cloud consisting of one vertex and all its satellites, we would have a well-spaced point set. Furthermore, we require the spacing between satellites to be proportional to their distance from the nucleus: $\mathrm{NN}_{\mathcal{M}}(u) \in \Theta(|uv|)$, where $\mathcal{M}$ is the infinite mesh. These properties allow us to produce a well-spaced, size-optimal output. We describe the location of the satellites of an input vertex as follows:

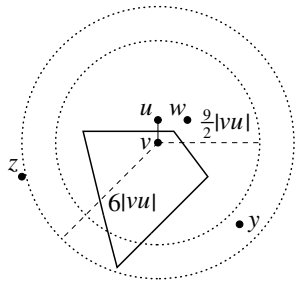

Figure 1: Let $\mathcal{M} = \{v, u, w, y, z\}$. The nearest neighbor distance of v, $\mathrm{NN}_{\mathcal{M}}(v)$, is $|vu|$. Thick lines depict the Voronoi cell of v, $\mathrm{Vor}_{\mathcal{M}}(v)$; v is 6-well-spaced, but not $\frac{9}{2}$-well-spaced.

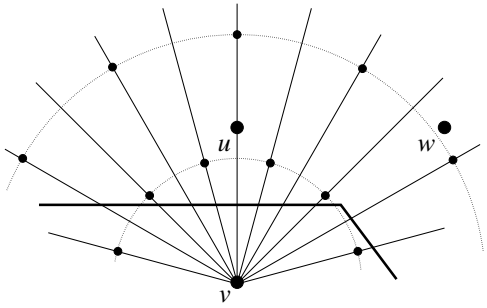

Figure 2: Zooming on $\mathcal{M}$, first two orbits and some rays of v (nucleus) are displayed. In total there are 24 rays with equal angles in between. Part of the Voronoi cell of v is displayed in thick lines. The innermost orbit has rank equal to the size of v. Particular intersections of the rays with orbits form satellites.

Consider concentric circles of radius 2^ℓ around every input vertex where $\ell \in \mathbb{Z}$ is the *rank* of these circles. We define the *size* of an input vertex v, written $\langle v \rangle$, to be the largest rank ℓ such that the circle at rank ℓ (of radius 2^ℓ) does not contain any other input vertex. For each input vertex v of size ℓ, we define its *orbits* to be the circles centered at v that are at ranks $\geq \ell$. Furthermore, consider 24 *rays* leaving each input vertex at angles $\theta, 2\theta, \ldots, 24\theta$, where $\theta = \pi/12$. 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. Intuitively, this template defines a discrete polar coordinate system, where the nucleus defines the origin, the rank defines the radius (exponentially), and the ray defines the polar angle. More formally, for given $v \in \mathsf{N}$, $\ell \in \mathbb{Z}$, and $r \in \{1, \ldots, 12\}$, we use $v_{\ell,r}$ to denote the satellite which has nucleus v, is on the orbit at rank ℓ, and on the ray at angle $2r\theta$ if ℓ is even, or on the ray at angle $(2r-1)\theta$ if ℓ is odd. For shorthand, we say that $v_{\ell,r}$ is the ℓ-satellite of v on ray r. Figure 2 illustrates these definitions.

Construction Algorithm. We build a triangulation in two stages: first, we construct a well-spaced superset of the input by inserting certain satellites and eliminating rest of them, then, we identify the Delaunay triangulation of the well-spaced superset of the first stage. In the first stage, we insert those satellites that are guaranteed not to cause small features. More specifically, for any ℓ-satellite s, defining the *certificate ball* of s to be the ball of radius $2^{\ell-2}$ centered at s, we insert s if and only if its certificate ball is empty. In order to ensure that the certificate balls of the inserted satellites remain empty, we process the satellites in increasing rank order. In the second stage, we identify potential Voronoi neighbors of each vertex, which by the well-spacedness property is guaranteed to be of constant size. We determine whether if each of these potential neighbors is actually a neighbor or not.

```
KineticWSP (N) =
  Construct point location structure S for N
  D ← ApproxDiameter(S)
  for each v ∈ N do
    ⟨v⟩ ← ApproxSize(S, v)
  for ℓ = min_{v∈N} ⟨v⟩ to ⌈lg 4D⌉ do
    for each v ∈ N do
      if ⟨v⟩ ≤ ℓ and Active(v, ℓ) then
        for r = 1 to 12 do
          if BallEmpty(S, v_{ℓ,r}, 2^{ℓ−2}) then
            InsertSteiner(S, v_{ℓ,r})
    for each v_{ℓ,r} ∈ S do
      CertifySteiner(S, v_{ℓ,r})
  for each v ∈ N do
    for each inserted satellite s of v do
      SetVoronoiNeighbors(S, s)
```

Figure 3: The psuedo-code for the construction algorithm

Processing satellites in increasing rank order allows us to progress towards a well-spaced output incrementally. More specifically, the satellites inserted at the current rank bound the Voronoi cells of the satellites inserted at the previous rank, making them well-spaced. Using this property, we prove the invariant that at the end of each rank all satellites inserted at the previous rank become and remain well-spaced. Another important observation is that if none of the previous rank satellites of an input vertex is inserted then it is unnecessary to insert any of the satellites of that vertex at the current rank. This observation proves to be helpful in achieving efficient dynamic and kinetic updates, thus in our algorithm, we only consider the satellites of the *active* input vertices: an input vertex v is active at its first rank $\langle v \rangle$ and it continues to be active at higher ranks as long as one of its satellites gets inserted at the current rank.

In order to better understand the underlying process, we relate the problem of determining which satellites to be inserted to the maximal independent subset problem. At a given rank ℓ, before inserting any satellites, consider the ℓ-satellites whose certificate balls are empty. Let the *proximity graph* at rank ℓ be the undirected graph on these ℓ-satellites with edges connecting two of them if and only if they are within distance $2^{\ell-2}$ from each other. What our algorithm does is equivalent to inserting a maximal independent subset of these satellites. Indeed, we prove that choosing any maximal independent subset would be sufficient to construct a well-spaced, size-optimal output.

Our construction algorithm uses the point location data structure to build a set of certificates that certify the well-spacedness of the output and the validity of its Delaunay triangulation. These certificates confirm the activity status on all input vertices and prove that the output is composed of the input and a maximal independent subset of the proximity graph at each rank.

Dynamic and Kinetic Maintenance. Upon a certificate failure or a dynamic modification, we update the well-spaced superset, its triangulation, and the set of certificates. Following the structure of the construction algorithm we iterate over ranks, repair the proximity graph and update the maximal independent subset of satellites chosen at each iteration. During the update, we may need to remove and insert satellites at each rank. In order to determine a maximal independent subset of the new proximity graph, the decisions we make at that rank should not be affected by higher rank satellites present in the spanner. Therefore, at rank ℓ, instead of using the complete deformable spanner S, we use the structure induced by the vertices of rank ℓ or less. We call the induced structure the *restriction of* S *to rank* ℓ, and denote it by $\mathsf{S}|_\ell$. For the *restriction of* S *to the input (vertices)*, we use $\mathsf{S}|_0$. We keep two sets of vertices at each rank for affected satellites. The first set, $\mathcal{F}$, tracks the satellites that may be required to be inserted or removed, we call them *fully affected*. The second set, $\mathcal{P}$, tracks the affected satellites which are previously inserted and whose certificates that certify their insertion remain unaffected. These satellites, which we call *partially affected*, are not required to be removed; however, we do need to fix their representations in the data structure by reinserting them.

Our update algorithm starts by repairing $\mathsf{S}|_0$. Then, it computes the current diameter and updates the sizes of affected input vertices. Iterating through each rank ℓ, it updates the well-spaced superset in three phases: *remove*, *repair*, and *insert*. In the remove phase the algorithm removes the fully affected satellites from the spanner, $\mathsf{S}|_\ell$. In the repair phase, it reinserts the partially affected satellites so that $\mathsf{S}|_\ell$ is repaired. In the insert phase, similar to the construction algorithm, it tries to insert all fully affected satellites (including the ones that are removed earlier) into the data structure. The last step of the insert phase is certification; in this step the algorithm (re)certifies that the certificate balls of all affected satellites in the data structure are empty. Then the update aglorithm moves to the second stage, updating the triangulation. Having repaired the well-spaced superset, the update algorithm repairs the Voronoi cells (Delaunay edges) of each of the affected output vertices.

It is important to note that our update algorithm handles batch updates, therefore, we can handle any possible degeneracies (certificates that have the same failure times) that might occur due to translations of same input vertices. More importantly, we prove that we can bound the size of these degeneracies by a constant, therefore, we process each kinetic update, a single event or a batch of constant size, asymptotically in $O(\log \Delta)$ time.

References

[1] Pankaj K. Agarwal, Jie Gao, Leonidas J. Guibas, Haim Kaplan, Vladlen Koltun, Natan Rubin, and Micha Sharir. Kinetic stable delaunay graphs. In *SCG '10: 26th Annual Symposium on Computational Geometry*, 2010.

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

[3] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[4] Marshall Bern, David Eppstein, and John R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, 1994.

[5] Marshall W. Bern, David Eppstein, and Shang-Hua Teng. Parallel construction of quadtrees and quality triangulations. *Intl Journal of Comput. Geom. and App.*, 9(6):517–532, 1999.

[6] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[7] Jyh-Jong 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.

[8] Jie Gao, Leonidas J. Guibas, and An Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Applications*, 35(1):2–19, 2006.

[9] Leonidas J. Guibas. Kinetic data structures: a state of the art report. In *WAFR '98: 3rd Workshop on alg. found. of robotics*, pages 191–209, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[10] Leonidas J. Guibas, Joseph S. B. Mitchell, and Thomas 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.

[11] Benoît Hudson, Gary L. Miller, and Todd Phillips. Sparse Voronoi Refinement. In *15th Intl. Meshing Round.*, pages 339–356, 2006.

[12] Haim Kaplan, Natan Rubin, and Micha Sharir. A kinetic triangulation scheme for moving points in the plane. In *SCG '10: 26th Annual Symposium on Computational Geometry*, 2010.

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