Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 124 (2016) 96–108

SAND2016-10992C

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable (IMR25)

Recursive Spoke Darts: Local Hyperplane Sampling for Delaunay and Voronoi Meshing in Arbitrary Dimensions

Mohamed S. Ebeida[a], Ahmad A. Rushdi[a,b]

[a]*Sandia National Laboratories, Albuquerque NM 87185, U.S.A.*
[b]*Institute for Computational Engineering and Sciences, University of Texas, Austin TX 78712, U.S.A*

Abstract

We introduce Recursive Spoke Darts (RSD): a recursive hyperplane sampling algorithm that exploits the full duality between Voronoi and Delaunay entities of various dimensions. Our algorithm abandons the dependence on the empty sphere principle in the generation of Delaunay simplices providing the foundation needed for scalable consistent meshing. The algorithm relies on two simple operations: line-hyperplane trimming and spherical range search. Consequently, this approach improves scalability as multiple processors can operate on different seeds at the same time. Moreover, generating consistent meshes across processors eliminates the communication needed between them, improving scalability even more. We introduce a simple tweak to the algorithm which makes it possible not to visit all vertices of a Voronoi cell, generating almost-exact Delaunay graphs while avoiding the natural curse of dimensionality in high dimensions.

© 2016 The Authors. Published by Elsevier Ltd.
Peer-review under responsibility of organizing committee of the 25th International Meshing Roundtable (IMR25).

Keywords: Delaunay Triangulation, Voronoi Meshes, High-Dimensional Spaces.

1. Introduction

Delaunay and Voronoi meshing techniques are very popular in several computational geometry [1,2] and meshing [3–5] problems for their rich geometric structures and appealing element properties. For a set of point seeds $\mathcal{X}$ in the d-dimensional Euclidean space $\mathcal{D}$, the Delaunay mesh is defined by the triangulation Del($\mathcal{X}$) where no point in $\mathcal{X}$ falls inside the circum-hypersphere of any simplex in Del($\mathcal{X}$) (e.g., in 2d, every triangle in a Delaunay triangulation is associated with an empty circumcircle passing through its vertices). The dual of Del($\mathcal{X}$) is the Voronoi mesh Vor($\mathcal{X}$) which splits $\mathcal{D}$ into cells where each cell contains all the points closer to its seed than any other seed in $\mathcal{X}$. Fundamentally, a Domain Decomposition (DD) method breaks down a domain into a finite number of elements, with certain desired properties, e.g., positive Jacobians, convex elements, and planar facets. A DD also defines how a point in the

* Corresponding author. Tel.: +1-505-844-0456
E-mail address: msebeid@sandia.gov

domain is assigned to an element (or more), and how each element in the DD identifies its neighbor element(s) in its close proximity.

For example, the Voronoi Domain Decomposition (VDD) decomposes a domain into convex elements (cells) bounded by planar convex facets. VDD cells are associated with generating sample points. Each cell is formed around a given seed, containing all points closer to that seed than any other seed in the domain. Despite their abundance of desired features, VDDs are not yet fully exploited in many applications. This can be mainly attributed to their unsolved computational challenges; generating true (unclipped) Voronoi cells that naturally conform to prescribed boundaries is an open computational geometry problem. In low-dimensions, state-of-the-art techniques rely on clipping, which lacks robustness and generates cells that are non-convex and not star-shaped. See Figure 1 for a 2d illustration of how clipping is employed into the evaluation of points neighborhood and consequently define Vor($\mathcal{X}$). Note the curse of dimensionality associated with this class of hyperplane-hyperplane trimming operations.

In high-dimensions, explicit generation of Voronoi cells, regardless of boundaries, is intractable. The number of corners of each Voronoi cell grows exponentially with dimension, imposing a lingering curse of dimensionality. Therefore, several Delaunay and Voronoi meshing approaches were tailored to specific problems with fixed dimensionality (e.g., 4d medical images [6] and scientific visualization [2,7]) or for a particular set of points (e.g., [8]).

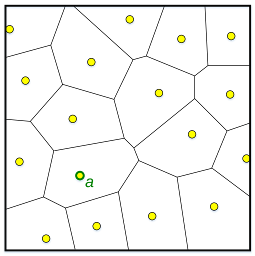
1.1. Related Work

Several methods rely on the duality between Del($\mathcal{X}$) and Vor($\mathcal{X}$) for their generation. Note that Del($\mathcal{X}$) is defined through the Delaunay simplices covering the *convex hull* of the points in $\mathcal{X}$. While a cell in Vor($\mathcal{X}$) contains all the points closer to its seed than any other seed in $\mathcal{X}$, a Delaunay simplex in Del($\mathcal{X}$) connects $d+1$ seeds, and is associated with an empty circumsphere (Delaunay sphere) that has no seeds inside it. The center of a Delaunay sphere is a corner of a Voronoi cell (Voronoi vertex). This feature is termed: the *empty sphere principle*, and is the basic rule to generate Del($\mathcal{X}$); see Figure 2 for a 2d example of how empty spheres develop as points are added to the domain and the resulting Del($\mathcal{X}$). An important remark at this point is that Vor($\mathcal{X}$) is uniquely defined for a given $\mathcal{X}$, while Del($\mathcal{X}$) is not due to potential ambiguous connectivity scenarios (e.g., when more than $d + 1$ seeds fall on the surface of the same Delaunay sphere). An ambiguous Delaunay simplex corresponds to a degenerate Voronoi entity of some dimension. There exist several algorithms to generate Vor($\mathcal{X}$):

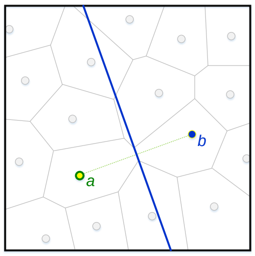
- *Direct algorithms*: compute the convex hull of $\mathcal{X}$, e.g., plane cutting [9], Fortune's [10], and Quickhull [11].
- *Indirect algorithms*: start by generating Del($\mathcal{X}$) and obtain Vor($\mathcal{X}$) as its dual, e.g., Bowyer—Watson [12–14].

In a nutshell, any exact algorithm must visit all Voronoi vertices, or equivalently visit the corresponding Delaunay spheres in order to construct a VDD or its dual Delaunay simplices. Note that the number of Voronoi vertices grows with dimensions introducing a *natural* curse of dimensionality. Therefore, both Voronoi and Delaunay DDs are completely abandoned in high-dimensional problems. Several computational challenges face each of the Voronoi and Delaunay decomposition approaches. On the direct methods front, plane cutting techniques go the extra mile of constructing the entire facets (not just vertices) of each Voronoi cell via successive hyperplane-hyperplane trimming operations. This approach enables the independent construction of different Voronoi cells; an appealing feature for scalable parallel implementations. However, as the number of dimensions grows, these operations become more complex and would require intractable extra storage capacity to track all bounding entities of the hyperplane being trimmed. This add-on algorithmic curse of dimensionality limits the implementation of plane cutting approaches to low dimensions (e.g., $d = 2, 3$) [15,16]. The Quickhull algorithm avoids the successive trimming operations by lifting the seeds onto $d+1$ paraboloid, deepening the curse of dimensionality and therefore increasing the computational and storage requirements exponentially. The implementation of Quickhull, Qhull, is the standard tool to build Voronoi and Delaunay meshes in arbitrary dimensions.

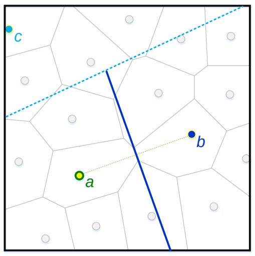
Indirect methods, on the other hand, rely on the empty sphere principle [17–20] and have been well-studied in the last few decades. However, most of the research efforts here were directed towards low dimensional problems (e.g., $d = 2, 3$). Tools using the empty sphere principle are usually dimension-specific. Furthermore, from an implementation perspective, state of the art tetrahedral Delaunay meshing software tools are inherently sequential, i.e., they insert points one after the other, which leads to poor parallel scalability. Moreover, changing the order of point insertions typically generates inconsistent meshes.



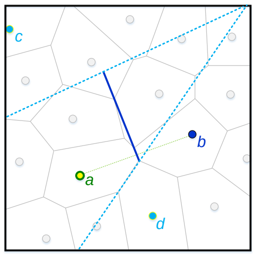
(a) Identifying the Voronoi neighbors of seed a.



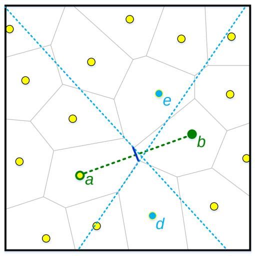
(b) Seed b is declared as a candidate neighbor of a.



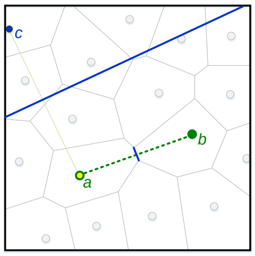
(c) Seed c trims the candidate facet associated w/ b.



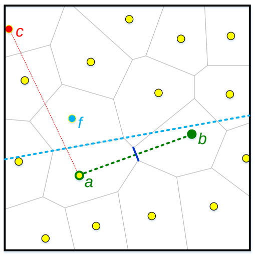
(d) Seed d further trims the candidate facet.



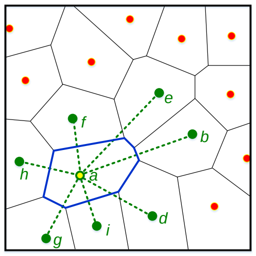
(e) Trimming using all seeds, b is a neighbor of a.



(f) Seed c is declared a new candidate neighbor of a.

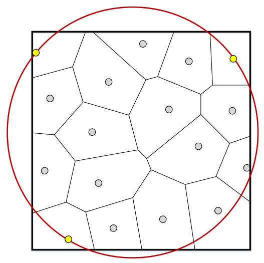


(g) Trimming with f, c can not be a neighbor of a.

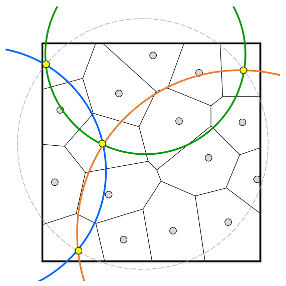


(h) After considering all seeds, Vor(a) is formed.

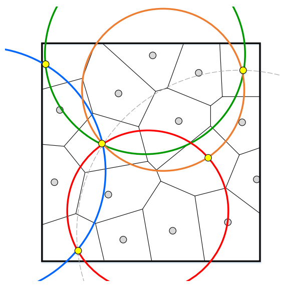
Figure 1: Constructing a 2-d Vor($\mathcal{X}$) via successive plane cutting operations. To find the exact cell edges around seed a, we need to identify its neighbor cells, which in turn requires successive hyperplane-hyperplane trimming operations, adding an algorithmic curse of dimensionality.



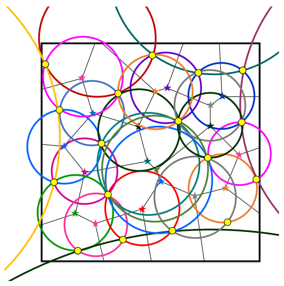
(a) Largest empty circumcircle passing through 3 points in $\mathcal{D}$.



(b) As a point is added, three empty circumcircles are created.

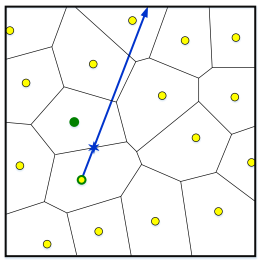


(c) Adding more points changes the circumcircles' arrangement.

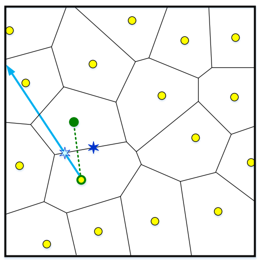


(d) All points are added, and circumcircles define Del($\mathcal{X}$).

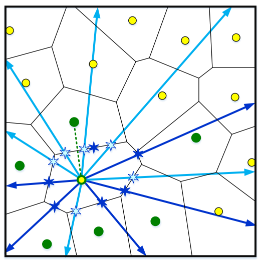
Figure 2: Constructing a 2-d Del($\mathcal{X}$) via the empty sphere principle: the circumcircles of all valid Delaunay triangles have empty interiors.



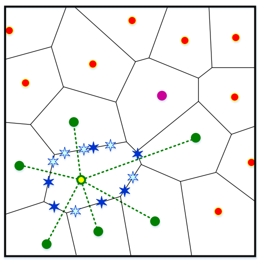
(a) A random line samples a point from a Voronoi facet.



(b) A redundant point sampled from the same facet.



(c) Local MC line sampling to find Delaunay neighbors.



(d) Neighbors sharing relatively small facets missed.

Figure 3: Constructing a 2-d Del($\mathcal{X}$) via local hyperplane sampling: identifying the significant neighbors of a given Voronoi cell can be done without imposing the curse of dimensionality associated with the hyperplane-hyperplane trimming operations.

1.2. Local Hyperplane Sampling

The idea of sampling a parameter space using a random well-spaced pointset is very appealing for high-dimensional applications, e.g., Uncertainty Quantification (UQ) and optimization. However, the state-of-the-art algorithms to generate it rely on background grids to track the uncovered regions for future sampling. This introduces a curse of dimensionality, bounding their implementations to about 6-d in practice. A non-traditional sampling approach uses hyperplane samples (e.g., lines, planes, etc) rather than simple 0-d points. For example, a Maximal-Poisson disk sampling (MPS) technique [21] aims to generate well-spaced samples that can achieve maximality in covering $\mathcal{D}$. Using hyperplane samples was found to have superior performance in finding the voids between MPS disks, and hence reach a maximal disk packing in high dimensions without suffering from the curse of dimensionality [22]. Coupling this hyperplane sampling approach with the duality between Voronoi facets and Delaunay edges enabled the first *implicit* Voronoi domain decomposition, where a Voronoi cell is not explicitly constructed via its vertices. Instead, a domain point is classified into a cell using a closest neighbor search over all seeds. This operation is not costly is it is independent of dimensionality. In the Spoke Darts algorithm [23], hyperplane sampling was used to solve complex global optimization problems in high dimensions. This approach relies on the duality between a Voronoi facet $\mathcal{F}_v$ and the dual Delaunay edge $\mathcal{E}_d$. The local hyperplane sampling approach does not visit any Voronoi vertex. Instead, it randomly samples points from a Voronoi facet to *witness* the dual Delaunay edge. The curse of dimensionality associated with visiting all Voronoi vertices is therefore broken. However, some Delaunay edges might be missed if their dual Voronoi facets are relatively small, hence the resulting Delaunay graph is only *approximate*. See Figure 3 for a 2d illustration of how the significant Delaunay neighbors with relatively-large shared facets are found via successive Monte Carlo (MC) sampling operations.

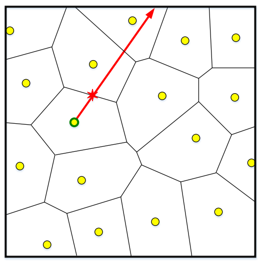
2. Algorithm Overview

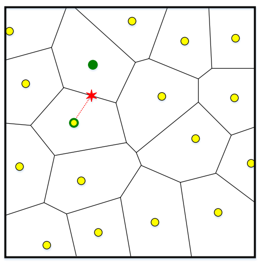
The local hyperplane sampling technique using by Spoke Darts in 1.2 only exploits the duality between Voronoi facets and Delaunay edges. This indeed breaks the curse of dimensionality and avoids the usage of the empty sphere principle, yet the generated Delaunay graph is only approximate. For an exact solution in arbitrary number of dimensions, we introduce a *recursive* local hyperplane sampling approach: Recursive Spoke Darts (RSD), which exploits the full duality between the entities of the Voronoi and Delaunay meshes. RSD uses the relations between Voronoi's facets $\mathcal{F}_v$ and edges $\mathcal{E}_v$) and Delaunay's edges $\mathcal{E}_d$ and facets $\mathcal{F}_d$ to find both Vor($\mathcal{X}$) and Del($\mathcal{X}$) in arbitrary dimensions and generate their exact meshes in a robust fashion.

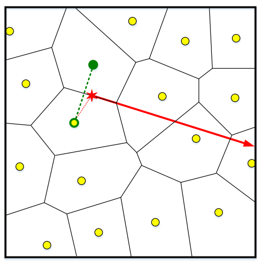
In brief, our algorithm starts at an arbitrary random Voronoi seed, samples random spokes (rays) from a series of $d, d-1, \ldots, 1$-dimensional spaces (aligned with various Voronoi entities) in order to identify valid Voronoi vertex at which we sample d deterministic lines along the Voronoi edges connected to it to identify its neighbor vertices. For illustration and without loss of generality, we present the steps of the RSD algorithm in a 3-dimensional space. Similar steps are employed in other dimensions.

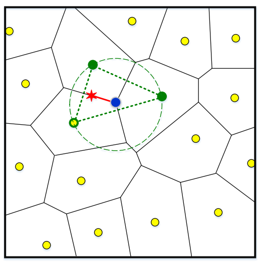
1. *Initial Random Seed Selection*: Pick an arbitrary Voronoi seed x_0 to start from.
2. *3-d Random Spoke Sampling*: Sample a random spoke in the 3-d space starting from x_0. This ray identifies a point x_1 on a Voronoi facet $\mathcal{F}_v$ which in turn identifies the dual Delaunay edge $\mathcal{E}_d$ of $\mathcal{F}_v$.
3. *2-d Random Spoke Sampling*: Starting from point x_1, sample another random spoke from the 2-d space of $\mathcal{F}_v$, identifying point x_2 along a Voronoi edge $\mathcal{E}_v$ that bounds $\mathcal{F}_v$. Point x_2 identifies the Delaunay facet $\mathcal{F}_d$; which is the dual of the Voronoi edge $\mathcal{E}_v$.
4. *1-d Random Spoke Sampling*: Sample another random spoke starting at point x_2 along the 1-d edge $\mathcal{E}_v$ to identify a Voronoi vertex x_3. Vertex x_3, in turn, identifies its dual Delaunay simplex connecting x_0 and 3 neighbor seeds.
5. *Deterministic Line Sampling*: Once a vertex is identified, 3 lines are sampled along the Voronoi edges connected to it in order to retrieve its neighbor vertices in the same cell, and their corresponding dual Delaunay simplices.
6. *Propagation*: Continue this propagation until each Voronoi vertex identifies its neighbor vertices on the same cell, completing the exact construction of that cell and its dual Delaunay simplices.

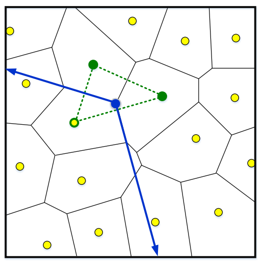
See Figure 4 and Figure 5 for illustrative examples in 2-d and 3-d spaces, respectively.

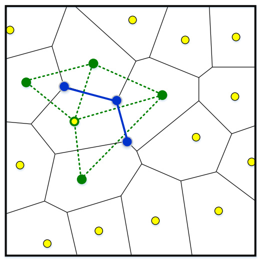

(a) A random ray samples a point from a 2-d space.

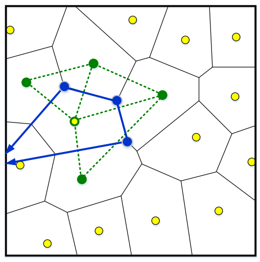

(b) A point is identified as edge and neighbor witness.


(c) A random ray samples a point from a 1-d edge.


(d) A delaunay simplex around the retrieved vertex.


(e) Two line samples from vertex along connected edges.


(f) Neighbor vertices and dual simplices identified.


(g) More line samples to identify other verteces.

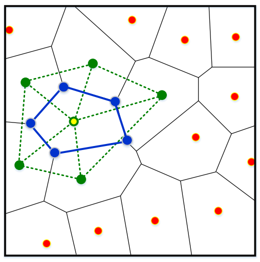
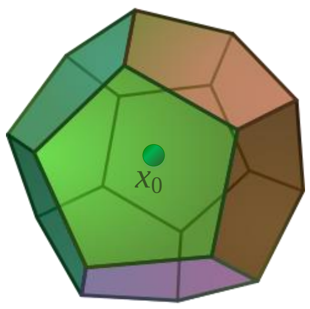
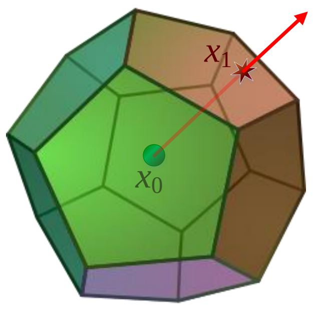

(h) All vertices and their dual simplices identified.

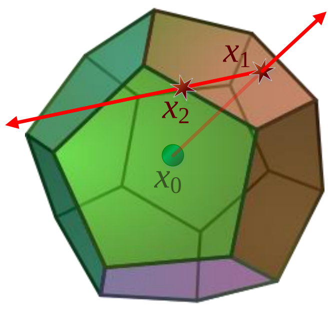
Figure 4: Constructing exact 2-d Del($\mathcal{X}$) and Vor($\mathcal{X}$) using our Recursive Spoke Darts (RSD) approach.



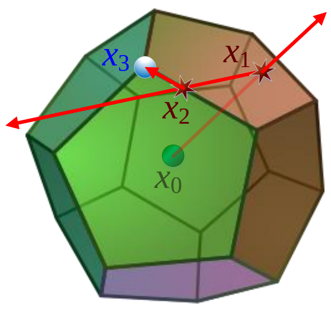
(a) Initial seed selection.



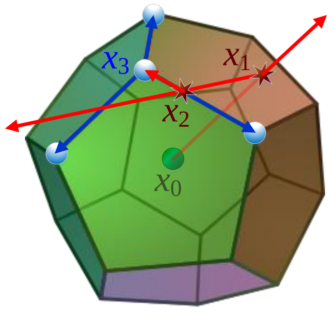
(b) Random 3-d spoke identifies x_1.



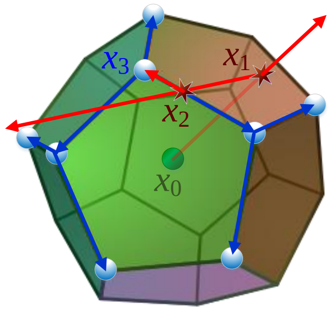
(c) Random 2-d spoke identifies x_2.



(d) Random 1-d spoke identifies x_3.



(e) 3 lines identify neighbor vertices.



(f) Further line propagation.

Figure 5: Constructing exact 3-d Del($\mathcal{X}$) and Vor($\mathcal{X}$) using our Recursive Spoke Darts (RSD) approach.

3. Algorithm Significance

The RSD algorithm has following significant properties, which makes it appealing especially for scalable parallel meshing applications:

1. *Abandoning the dependence on the empty sphere principle.* The empty sphere principle has serious limitations. In specific, it imposes a sequential seed insertion, and results in different meshes in case of Delaunay ambiguities if the order of insertion changes. RSD abandons the usage of the empty sphere principle in the generation of Delaunay simplices providing the foundation needed for scalable consistent meshing.
2. *Scalable Parallel Meshing.* The RSD approach guarantees scalability as multiple processors can operate on different seeds at the same time. Moreover, generating consistent meshes across processors via unified operations eliminates the communication needed between them even at interfaces, improving scalability even more.
3. *Robust Dimension-Independent Implementation.* From an implementation point of view, the RSD algorithm relies on two simple operations: a) line-hyperplane trimming, and b) spherical range search. The first operation recursively exploits the full Voronoi-Delaunay duality, eliminates the need for intractable hyperplane-hyperplane trimming associated with direct methods, and enables dimension-independent implementations. The second operation spots the equidistant seeds from a retrieved vertex to robustly construct the dual Delaunay simplices in a canonical form.
4. *Consistent Delaunay Meshing.* Conflicts between ambiguous Delaunay entities are handled using integer operators on seed indices. This guarantees mesh consistency regardless of the number of processors/partitions utilized, as long as the order of seeds' indices is locally preserved in each partition. Similar operators eliminate *slivers* on the fly, ensuring a high-quality mesh.
5. *Simplification for Tractability in High-Dimensions.* To avoid the natural curse of dimensionality associated with Delaunay and Voronoi meshing in high dimensions, RSD makes it possible not to visit all vertices. This can be simply achieved by limiting the propagation from a Voronoi vertex to others, still improving the accuracy of the approximate Delaunay graph over that resulting from the non-recursive version.

4. Implementation Details

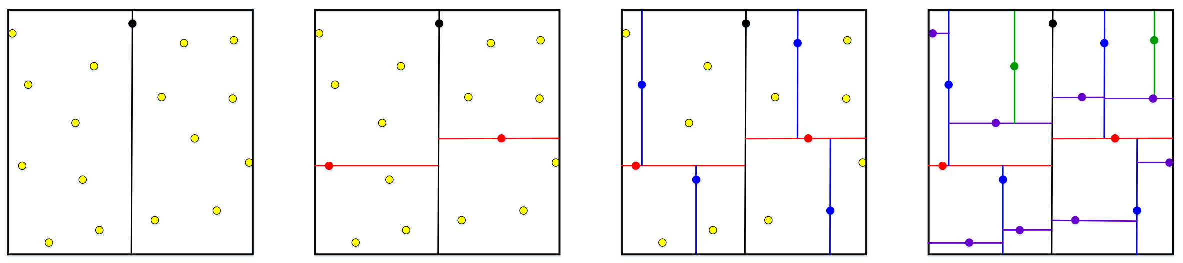
In this section, we list further details about the implementation of our algorithm. In specific, we present our initial data structure (a k-d tree) for neighborhood searches, illustrate how we use it to trim random spokes, and how we resolve ambiguities in a consistent way so that all processors would make the same resolution decision with no need for communication between them.

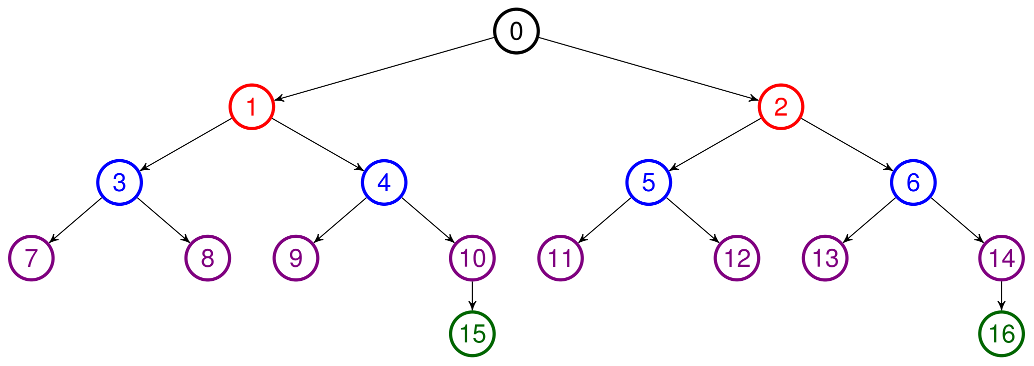
4.1. Initial Data Structure

We use balanced k-d trees [24] for our initial domain decomposition as they do not suffer from the curse of dimensionality and can also be implemented in parallel [25,26]. On average, balanced k-d trees use $O(\log N)$ to find the nearest neighbor within a tree of N nodes. Our k-d trees are *balanced* in the sense that at any level in the tree: a pivot hyperplane in some sub-domain is chosen to pass through the point at or closest to the median of the points in that sub-domain. See Figure 6 for a 2-d example of their construction. We rely on this class of binary trees to find the closest point to the end of the random spoke and hence make the right spoke-trimming decision as we explain next. We use `qsort` [27] to speed up the sorting process along the tree construction.

4.2. Trimming Spokes

At any Voronoi seed x_i in dimension d, whenever a random spoke $\vec{s}(x_i)$ is sampled, it needs to be trimmed at the boundaries of that Voronoi cell. For this purpose, we use the k-d tree to collect the closest neighbors $\{x_j\}$ of that Voronoi seed and use them to trim the spoke. At least $d + 1$ neighbors are needed. Our algorithm collects neighbors using a *neighborhood sphere* centered at x_i, and the radius of that sphere is parametrized by α; the global targeted aspect ratio of Voronoi cells, and r_{min}; the distance between x_i and its closest neighbor $x*$. The radius of the


(a) Binary tree splitting.



(b) k-d tree connections.

Figure 6: Building a balanced k-d tree as an initial domain partitioning data structure for our algorithm. Example shows a 2-d tree of 16 nodes.

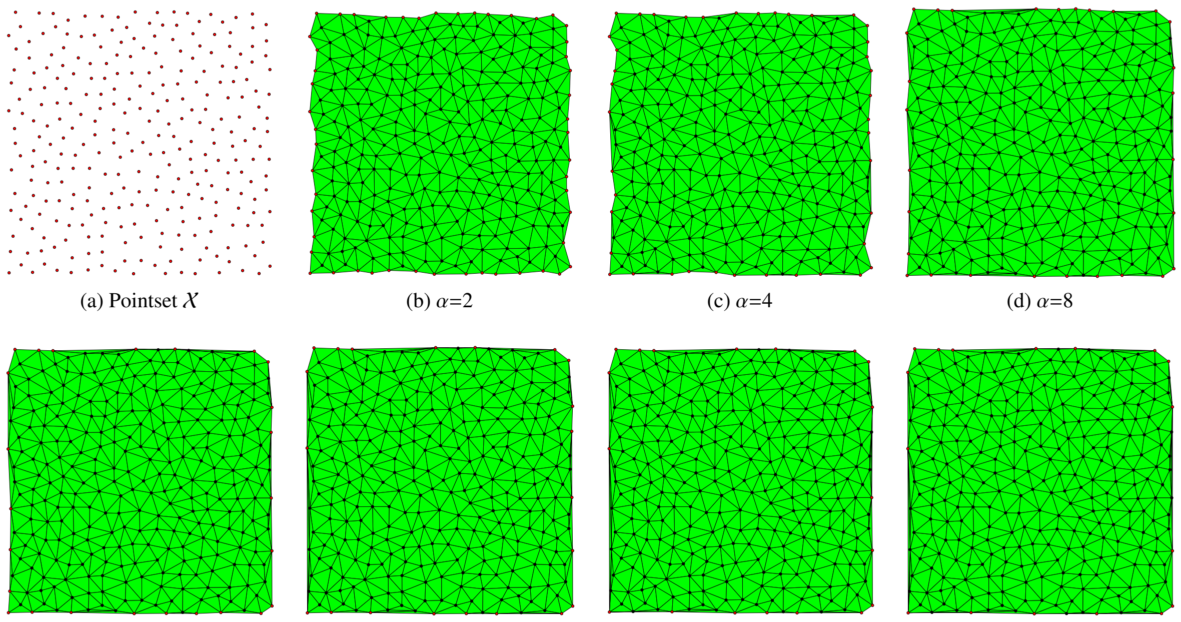
neighborhood sphere around x_i is then given by $r_i = \alpha * r_{min}$. The spoke is then trimmed using the Voronoi hyperplane between x_i and the closest point in the sphere to the end of the spoke x^c.

The process goes on until no point in the sphere can trim the spoke anymore; at which the dimensionality of the next spoke is reduced by 1. To reduce the dimensionality at each spoke step, we employ the Grahm-Schmidt approach [28]: given a set of orthogonal basis vectors $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{d-1}\}$ in the space $\mathbb{R}^d$, we can construct another orthogonal basis set $\{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{d-1}\}$ by successive projection:

$$\begin{aligned} \mathbf{v}_0 &= \mathbf{u}_0, \\ \mathbf{v}_k &= \mathbf{u}_k - \sum_{j=1}^{k-1} \mathrm{proj}_{\mathbf{v}_j} \mathbf{u}_k \end{aligned}$$

Using this approach, we construct the basis vectors $\mathcal{D}^*$ of the upcoming spoke, orthogonal to the basis of prior spokes $\mathcal{D}^{**}$. This continues until the dimensionality of future spoke is 0; indicating a Voronoi vertex, at which the propagation step kick in along associated Voronoi edges until all Voronoi vertices around x_i are visited. Obviously, the point distribution also impacts the construction of Delaunay simplices. Figure 7 shows the impact of increasing α during the successive phases of our algorithm for a set of well-spaced points. Similarly, Figure 8 shows the impact of increasing α on the construction of the Delaunay graph when the initial 2-d pointset is Monte-Carlo sampled.

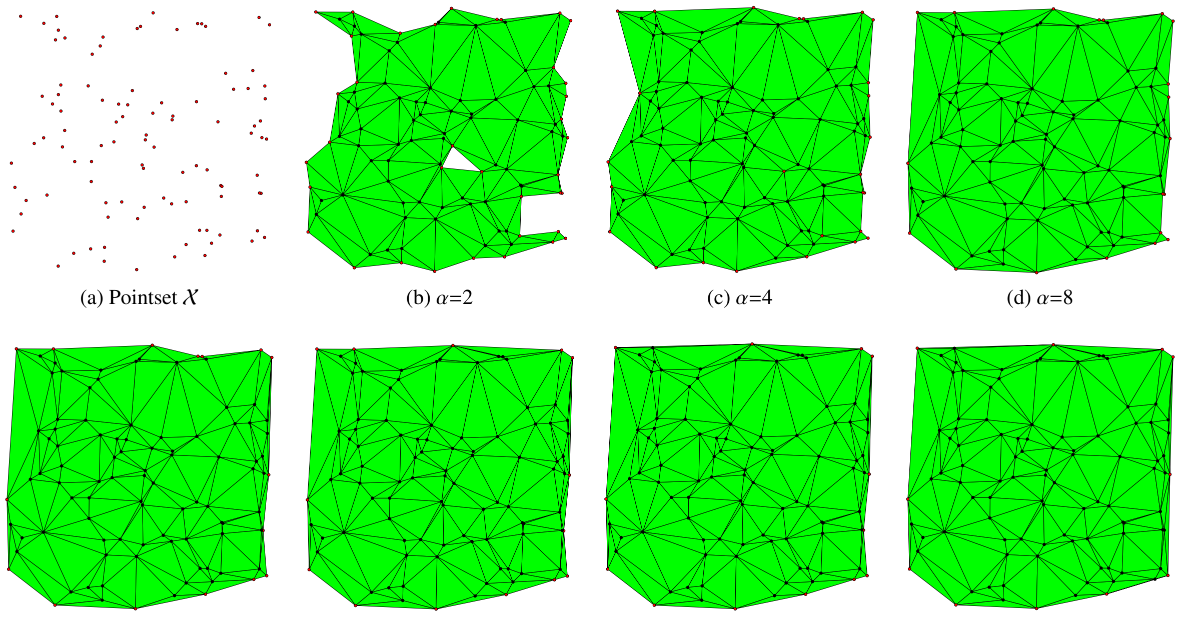
In general, our algorithm progressively increases α as illustrated in Figure 7 and Figure 8. Initially, α starts at the value of 2 (sufficient for a uniform well-spaced pointset). The value of α gets doubled in each phase, searching for Voronoi vertices of cells with matching (or less) aspect ratios. Experimentally, we chose a termination value of $\alpha_{max} = 2^{10}$. This limiting value of α can be increased to handle suspected cases on Voronoi cells with higher aspect ratios, with no significant increase in meshing time. Even with Monte Carlo pointsets, the value of α_{max} was sufficient in all our experiments.



(a) Pointset $\mathcal{X}$ (b) α=2 (c) α=4 (d) α=8

(e) α=16 (f) α=32 (g) α=64 (h) α=128

Figure 7: RSD construction of 2-d Delaunay meshes on well-spaced pointset $\mathcal{X}$ as the target aspect ratio parameter α is increased.



(a) Pointset $\mathcal{X}$ (b) α=2 (c) α=4 (d) α=8

(e) α=16 (f) α=32 (g) α=64 (h) α=128

Figure 8: RSD construction of 2-d Delaunay meshes on Monte Carlo pointset $\mathcal{X}$ as the target aspect ratio parameter α is increased.

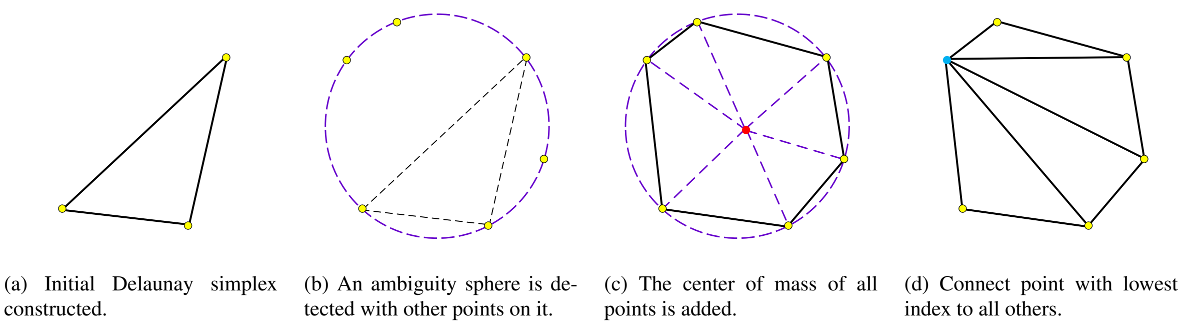

(a) Initial Delaunay simplex constructed.
(b) An ambiguity sphere is detected with other points on it.
(c) The center of mass of all points is added.
(d) Connect point with lowest index to all others.

Figure 9: Consistent resolution of Delaunay ambiguities. If an ambiguity is detected, the center of mass of the ambiguous points is temporarily added, resolving the ambiguity and deducing the connections between these points. Removing the point and keeping the connections, further connections are added by connecting the point with the lowest index to all others.

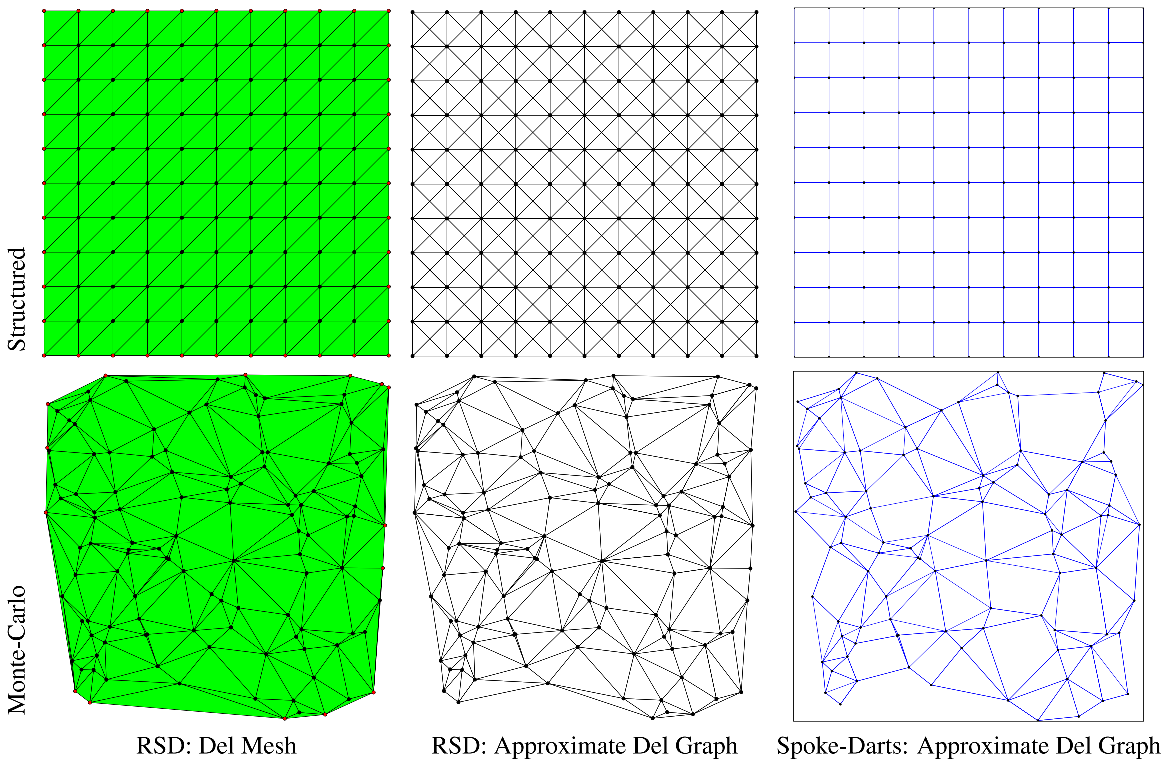

Structured
Monte-Carlo
RSD: Del Mesh
RSD: Approximate Del Graph
Spoke-Darts: Approximate Del Graph

Figure 10: Using 121 structured and Monte-Carlo points in 2-d, we distinguish the construction of the Delaunay mesh Del($\mathcal{X}$) using RSD, an almost-exact approximation of the Delaunay graph using the approximate version of RSD, and an approximation of the Delaunay graph using Spoke-Darts [23] with several missing Delaunay edges.

4.3. Resolving Ambiguities and Conflicts

Our algorithm can consistently resolve Delaunay ambiguities without any communication load between processors. If an ambiguity is detected, we add a point in the center of the ambiguous points, temporarily resolving the ambiguity and deducing the connections between these points. Removing the center of mass point and keeping the connections, we add further connections by connecting the point with the *lowest index* to all others. Since this rule is known ahead of time, different processors would resolve an ambiguity in the same exact way, reaching the same exact mesh. See Figure 9 for an illustrative example.

4.4. Approximate RSD Implementation in High Dimensions

In high dimensions, the number of Voronoi vertices and Delaunay simplices grow exponentially; a major bottleneck for discretizing high dimensional spaces. However, the number of Delaunay edges relies only on the number of input points (and not dimension), hence can be enumerated efficiently. Similarly, although the number of Voronoi vertices becomes intractable in high dimension, the number of Voronoi cells has a one-to-one mapping to the number of input points (Voronoi seeds). Coupling these observations makes it possible to decompose high dimensional spaces in a tractable way. In this case, our RSD algorithm can still proceed to some precision and stop the propagation step before all vertices are collected. Once a Voronoi vertex is retrieved, the equi-distant Voronoi seeds to that vertex are connected in the approximate Delaunay graph. Visiting a Voronoi vertex that adds no new Delaunay edges to the approximate graph is considered a *miss*. We stop the approximate RSD procedure around some seed when some number of successive misses occur (e.g., 100 in our experiments). This approximate version of RSD samples a subset of the Voronoi vertices, rather than visit all of them. Our experiments show that the approximate RSD algorithm generated an almost exact Delaunay graph despite not visiting all vertices.

See Figure 10 for comparisons between Spoke Darts, RSD, and RSD approximate version. Using 121 structured and Monte-Carlo points in 2-d, we distinguish the construction of the Delaunay mesh Del($\mathcal{X}$) using RSD, an almost-exact approximation of the Delaunay graph using the approximate version of RSD, and an approximation of the Delaunay graph using Spoke-Darts [23] with several missing Delaunay edges. Note that a Delaunay meshing technique has to resolve Delaunay ambiguities when more than $d+1$ points fall on the same Delaunay sphere. Some of the Delaunay edges have to be discarded to avoid self-intersecting elements. RSD does that successfully and consistently as explained in section 4.3 and illustrated in Figure 9. The Delaunay graph, however, does not need these conflict resolutions, as shown in the structured case of the approximate version of RSD of Figure 10.

5. Experimental Results

In this section, we compare the performance of RSD to standard Delaunay and Voronoi meshing techniques in different dimensions in terms of the meshing runtime. In specific, we compare RSD to the state-of-the-art Delauanay meshing techniques: Triangle [18,29], TetGen [30], and Qhull [31]. Triangle generates exact Delaunay triangulations and Voronoi diagrams in 2-d. TetGen is the state of the art method to generate tetrahedral meshes on polyhedral domains in 3-d. Qhull computes the convex hull using the Quickhull algorithm, and hence finds the Delaunay triangulation and Voronoi diagram in arbitrary dimensions.

Figure 11 shows comparison results illustrating how RSD would perform with a variable number of processors, compared to Triangle, Qhull, and TetGen in 2-d, 3-d, and 4-d. Plots show the runtime of each method against the output number of Delaunay simplices. For a well-spaced set of points with numerous Delaunay ambiguities, RSD was slightly slower than Triangle in 2-d, however, it successfully outperformed TetGen and Qhull in 2-, 3-, and 4-d. Resolving ambiguities turned out to be a challenge for Qhull in various dimensions. In 2-d, Qhull did not produce valid meshes with number of simplices larger than 10M. To demonstrate the consistent meshing capability of RSD, we also ran our serial implementation on a number of processors p with no communication between them. Each processor loaded and constructed the k-d tree for the entire input pointset and generated the output Delaunay simplices associated with a subset of the input vertices; as illustrated in Figure 12. We show the performance of $p = 10$ and $p = 50$ in Figure 11. Applying domain subdivision to assign points to these processors would further improve the parallel performance. For the case of Monte-Carlo set of points in the right column of Figure 11, the aspect ratio of the elements deteriorate significantly. Although this impacted the runtime performance of RSD, it was still capable of generating exact Delaunay meshes in linear time. A thorough analysis of this capability including domain subdivisions is to be addressed in a follow-up paper.

To validate and check our final meshes, we rely on the "Empty Sphere Principle" to check our meshes and validate the final results. For example, we would construct the circumsphere of every output Delaunay simplex and check that its center matches the corresponding dual Voronoi vertex retrieved by our algorithm.

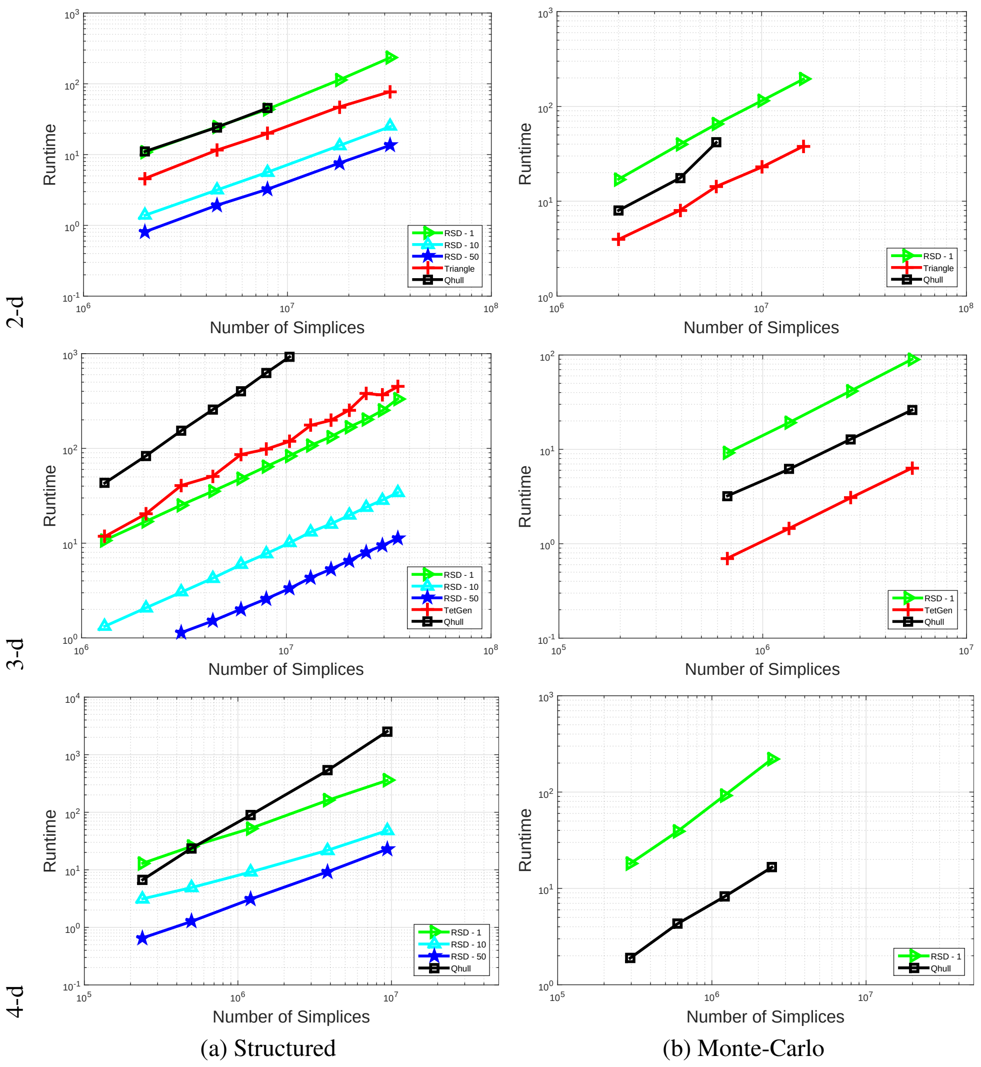


Figure 11: Runtime comparisons between Triangle, Qhull, TetGen, and RSD with a variable number of processors (where RSD-p indicates the utilization of p processors). Rows present different dimensionality: 2-d, 3-d, and 4-d, while columns present different point distribution: well-spaced structured and Monte-Carlo.

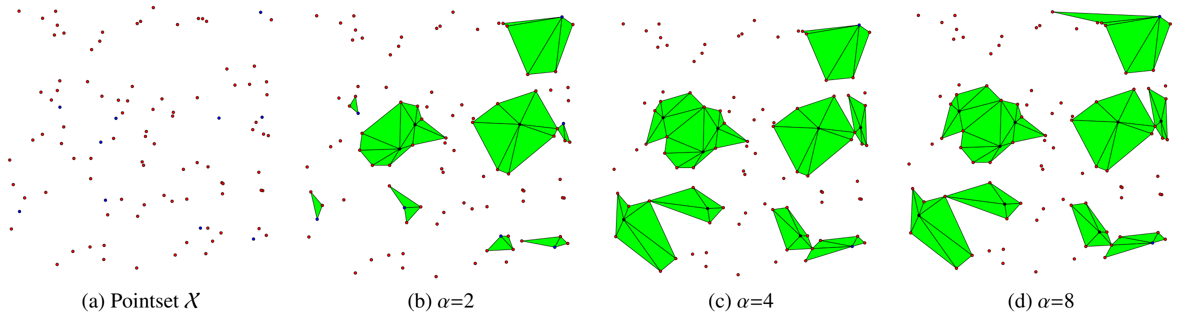


Figure 12: RSD parallel construction of 2-d Delaunay meshes on 10 processors, with 100 MC points as α is increased.

6. Conclusions

We introduced a novel recursive hyperplane sampling algorithm to construct exact Delaunay meshes in low-dimensions and approximate Delaunay graphs in high-dimensions. Our approach exploits the full duality between Voronoi and Delaunay entities of various dimensions, and abandons the dependence on the empty sphere principle in the generation of Delaunay simplices providing the foundation needed for scalable consistent meshing, resulting in a communication-free implementation. We demonstrated the capabilities of our algorithm when compared to state-of-the-art Delaunay meshing tools (Triangle, TetGen, and Qhull). We showed that the construction of an almost-exact Delaunay graph is tractable using the approximate version of our algorithm. Our future step include a complete analysis of the accuracy of that approximate version in high dimensions. We will also investigate various techniques to improve the RSD performance when handling random pointsets.

References

[1] C. D. Toth, J. O'Rourke, J. E. Goodman, Handbook of discrete and computational geometry, CRC press, 2004.

[2] J.-R. Sack, J. Urrutia, Handbook of computational geometry, Elsevier, 1999.

[3] S.-W. Cheng, T. K. Dey, J. Shewchuk, Delaunay mesh generation, CRC Press, 2012.

[4] P. Frey, P.-L. George, Mesh generation, John Wiley & Sons, 2013.

[5] H. Edelsbrunner, Geometry and topology for mesh generation, Cambridge University Press, 2001.

[6] P. Foteinos, N. Chrisochoides, 4d space–time delaunay meshing for medical images, Engineering with Computers 31 (2015) 499–511.

[7] C. Wang, J. Tao, Graphs in scientific visualization: A survey, in: Computer Graphics Forum, Wiley Online Library, 2016.

[8] E. Karabelas, M. Neumüller, Generating admissible space-time meshes for moving domains in $d + 1$-dimensions, arXiv preprint arXiv:1505.03973 (2015).

[9] C. H. Rycroft, G. S. Grest, J. W. Landry, M. Z. Bazant, Analysis of granular flow in a pebble-bed nuclear reactor, Physical review E 74 (2006) 021306.

[10] S. Fortune, A sweepline algorithm for Voronoi diagrams, Algorithmica 2 (1987) 153–174.

[11] C. B. Barber, D. P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Transactions on Mathematical Software (TOMS) 22 (1996) 469–483.

[12] A. Bowyer, Computing Dirichlet tessellations, The Computer Journal 24 (1981) 162–166.

[13] D. F. Watson, Computing the n-dimensional delaunay tessellation with application to Voronoi polytopes, The Computer Journal 24 (1981) 167–172.

[14] H. Si, Tetgen, a delaunay-based quality tetrahedral mesh generator, ACM Transactions on Mathematical Software (TOMS) 41 (2015) 11.

[15] C. Rycroft, Voro++: A three-dimensional Voronoi cell library in C++, Lawrence Berkeley National Laboratory (2009).

[16] M. S. Ebeida, S. A. Mitchell, Uniform random Voronoi meshes, in: Proceedings of the 20th International Meshing Roundtable, Springer, 2012, pp. 273–290.

[17] L. P. Chew, Guaranteed-quality Delaunay meshing in 3d (short version), in: Proceedings of the thirteenth annual symposium on Computational geometry, ACM, 1997, pp. 391–393.

[18] J. R. Shewchuk, Delaunay refinement algorithms for triangular mesh generation, Computational geometry 22 (2002) 21–74.

[19] T. K. Dey, J. A. Levine, Delaunay meshing of piecewise smooth complexes without expensive predicates, Algorithms 2 (2009) 1327–1349.

[20] J. R. Shewchuk, Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator, Applied computational geometry towards geometric engineering (1996) 203–222.

[21] M. S. Ebeida, S. A. Mitchell, A. Patney, A. A. Davidson, J. D. Owens, A simple algorithm for maximal Poisson-disk sampling in high dimensions, Computer Graphics Forum 31 (2012) 785–794.

[22] M. S. Ebeida, A. Patney, S. A. Mitchell, K. R. Dalbey, A. A. Davidson, J. D. Owens, *k*-d darts: Sampling by *k*-dimensional flat searches, ACM Transactions on Graphics (2014). In press. Also at arXiv:1302.3917 [cs.GR]; URL `http://arxiv.org/abs/1302.3917`.

[23] M. S. Ebeida, S. A. Mitchell, M. A. Awad, C. Park, L. P. Swiler, D. Manocha, L.-Y. Wei, Spoke darts for efficient high dimensional blue noise sampling, arXiv preprint arXiv:1408.1118 (2014).

[24] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (1975) 509–517.

[25] L. Hu, S. Nooshabadi, M. Ahmadi, Massively parallel kd-tree construction and nearest neighbor search algorithms, in: Circuits and Systems (ISCAS), 2015 IEEE International Symposium on, IEEE, 2015, pp. 2752–2755.

[26] N. Rajani, K. McArdle, I. S. Dhillon, Parallel k nearest neighbor graph construction using tree-based data structures, in: 1st High Performance Graph Mining workshop, Sydney, 10 August 2015, 2015.

[27] C. Allison, Sorting with qsort, C Users J. 11 (1993) 107–ff.

[28] L. N. Trefethen, D. Bau III, Numerical linear algebra, volume 50, Siam, 1997.

[29] J. R. Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, in: M. C. Lin, D. Manocha (Eds.), Applied Computational Geometry: Towards Geometric Engineering, volume 1148 of *Lecture Notes in Computer Science*, Springer-Verlag, 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry.

[30] H. Si, Tetgen, a delaunay-based quality tetrahedral mesh generator, ACM Trans. Math. Softw. 41 (2015) 11:1–11:36.

[31] C. B. Barber, D. P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Trans. Math. Softw. 22 (1996) 469–483.