An Efficient Query Structure for Mesh Refinement

Benoît Hudson* Duru Türkoğlu†

Abstract

We are interested in the following mesh refinement problem: given an input set of points P in $\mathbb{R}^d$, we would like to produce a good-quality triangulation by adding new points in P. Algorithms for mesh refinement are typically incremental: they compute the Delaunay triangulation of the input, and insert points one by one. However, retriangulating after each insertion can take linear time. In this work we develop a query structure that maintains the mesh without paying the full cost of retriangulating. Assuming that the meshing algorithm processes bad-quality elements in increasing order of their size, our structure allows inserting new points and computing a restriction of the Voronoi cell of a point, both in constant time.

1 Introduction

A central task in scientific computing is to discretize a domain of interest into a simplicial mesh. Concretely, we investigate the following setting: we are given a point cloud P in $[0,1]^d$, typically $d = 2$ or 3. A meshing algorithm must produce a simplicial decomposition of $[0,1]^d$ (a triangulation in $d = 2$), in which all of the input points appear as vertices of the decomposition. Each simplex should have good aspect ratio (*quality*), since that is a necessary or at least sufficient condition for many scientific computing applications. To ensure this, the algorithm must *refine* the point set, creating new (*Steiner*) vertices. But it should not add too many vertices: if the smallest possible mesh of good aspect ratio has m vertices, the algorithm should output $O(m)$ vertices.

There are two categories of mesh refinement algorithms with provable guarantees: those based on quadtrees [BEG94, MV00], and those based on Delaunay triangulations [Rup95, She98]. Quadtrees can be constructed in $O(n \log n + m)$ time [BET99]. Traditional Delaunay methods produce substantially smaller meshes in practice, but are quadratically slow in the worst case. The runtime behaviour of these algorithms is dominated by the cost of maintaining the mesh as the output is incrementally computed. Two Delaunay refinement algorithms sidestep this cost and match the quadtree runtime: Har-Peled and Üngör [HPÜ05] hybridize Delaunay and quadtree methods, while Hudson, Miller, and Phillips [HMP06] carefully maintain a triangulation that is always sparse. Here, we extend the hybrid approach to three or more dimensions.

*Toyota Technological Institute at Chicago
†University of Chicago

INITIALIZE(P)	$O(n \log n + m)$
APPROXIMATENN(v)	$O(1)$
CLIPPEDVORONOI(v, β)	$O(1)$
ADDVERTEX(p, v)	$O(1)$

Fig. 1: The interface of our data structure, and runtimes assuming a bottom-up mesh refinement algorithm.

Mesh quality: In a mesh where all simplices have good aspect ratio, the vertices of the mesh are well-spaced in the following sense [Tal97]. Denote by $\mathrm{NN}(v)$ the distance from v to its nearest neighbour, and by $R(v)$ the distance from v to the farthest node of its Voronoi cell. Then we say that v is ρ-well-spaced if $R(v) \leq \rho NN(v)$.

Mesh spacing: The *local feature size* $\mathrm{lfs}(x)$ is the distance from x to its second-nearest input point [Rup95]. In a *size-conforming* mesh, at every output vertex v, $\mathrm{NN}(v) \in \Theta(\mathrm{lfs}(v))$. A size-conforming mesh has $O(m)$ vertices.

Bottom-up meshing: We say that a mesh refinement algorithm is bottom-up if it incrementally ensures that small Voronoi cells are well-shaped before processing large Voronoi cells. That is, there are constants ρ and γ such that before the algorithm inserts a new Steiner point p with a nearest other mesh vertex at distance $\mathrm{NN}(p)$, every mesh vertex v with $\mathrm{NN}(v) < \gamma\,\mathrm{NN}(p)$ is ρ-well-spaced.

Our contribution: Expanding upon the idea of Har-Peled and Üngör, we provide a data structure for a meshing algorithm to use as a black box to maintain information about the partial mesh as it is constructed. After initializing our structure, the algorithm has access to two queries and an update routine (see Figure 1). CLIPPEDVORONOI(v, β) computes the nearby portion of the Voronoi cell of a vertex v. We show in Section 2 that this clipped version of the Voronoi cell is sufficient to determine where to insert a Steiner vertex near v. Upon deciding to insert a new Steiner vertex at p, the algorithm can call ADDVERTEX(p, v) to effect the insertion. We prove in Theorem 7 that both CLIPPEDVORONOI and ADDVERTEX run in constant time, assuming the algorithm is a bottom-up meshing algorithm that produces a size-conforming mesh. To help the algorithm produce the bottom-up ordering, we provide APPROXIMATENN(v), which returns an approximation of the true distance from v to its nearest neighbour. Using our structure, a bottom-up mesh refinement algorithm can mesh a point cloud in $\mathbb{R}^d$ in $O(n \log n + m)$ time.

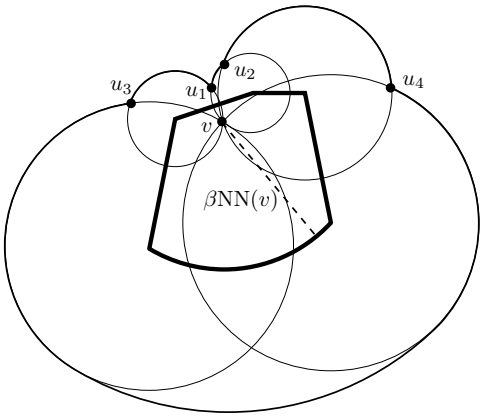

Fig. 2: The β-clipped Voronoi cell of v and its certificate region, bounded by neighbours u_1 through u_4.

2 Choosing Steiner Points

Regardless of runtime considerations, the fundamental question in mesh refinement is about where to put the Steiner points. Traditional solutions compute Steiner points based on the Delaunay triangulation, which is too expensive to maintain, so we need a more local way of choosing a Steiner point. The following region includes the points defined by two prior proposals for Steiner point choices — the *off-center* [Üng04] in 2D, and the CoreDisk [JÜ07] in 3D:

Definition 1 *The β-**clipped Voronoi cell** of a vertex v, denoted $V^\beta(v)$, is the intersection of the Voronoi cell of v, and the ball centered at v with radius $\beta\,\mathrm{NN}(v)$.*

Any point x in $V^\beta(v)$ is in the Voronoi cell of v; therefore, the open ball centered at x with radius $|vx|$ is empty of any mesh vertices. We call these **certificate balls**, and define the **certificate region** of $V^\beta(v)$ as the union of these balls (Figure 2). An algorithm that correctly computes $V^\beta(v)$ must have verified that the entire certificate region is empty; otherwise, the algorithm could falsely report x as being in the clipped Voronoi when it lies outside. Furthermore, an algorithm must verify that for all x on the boundary, either x is equidistant to v and another vertex, or x is at the clipping distance $\beta\,\mathrm{NN}(v)$ from v; otherwise, the algorithm could falsely report the clipped Voronoi cell is smaller than it is.

3 Data structure

Our data structure is based on the leaves of a quadtree. Each leaf square (or hypercube) stores the set of vertices contained in the square, and pointers to the neighbouring squares. Also, each vertex v stores a pointer to the square in which it lies, which we denote square(v). Our proofs of fast runtime depend on the quadtree having certain properties: **(A)** Each leaf square should only have a bounded number of neighbours. **(B)** Each leaf square should be sized in proportion to the local feature size of the points in the cell. That is, if a quadtree square has sides of length l, then for all x in the square, $\mathrm{lfs}(x) \in \Theta(l)$. The balanced quadtree of Bern, Eppstein, and Gilbert [BEG94] satisfies our requirements, though in practice it will be preferable to split the squares more coarsely.

It takes INITIALIZE $O(n \log n + m)$ time to compute the quadtree leaves [BET99]. The mesh spacing requirement on the meshing algorithm, and requirement (B) on the quadtree, together guarantee that the square size is within a constant factor of $\mathrm{NN}(v)$. This allows APPROXIMATENN(v) to return the size of square(v), in constant time. To implement ADDVERTEX(p, v), we first look up square(v), then walk from square to square along the segment pv. Upon reaching the square that contains p, we add p to its list of vertices. The description of CLIPPEDVORONOI merits its own section.

4 Clipped Voronoi Computation

We represent the β-clipped Voronoi cell of a vertex v using a set U of mesh vertices (e.g. $U = \{u_1, u_2, u_3, u_4\}$ in Figure 2). To find them, we perform a scan starting at v and growing a ball outward up to a maximum radius: at time t the ball has radius $t < t_{\max} = \beta\,\mathrm{NN}(v)$. When our scan reaches a vertex u at time t, we add u to U_t if it is a Voronoi neighbour, and $|uv| \leq 2t_{\max}$ — i.e. u is on the boundary of the certificate region. At time $t_{\max}$, we will know that we have covered the entire certificate region, and that $U_{t_{\max}}$ is an accurate representation of $V^\beta(v)$.

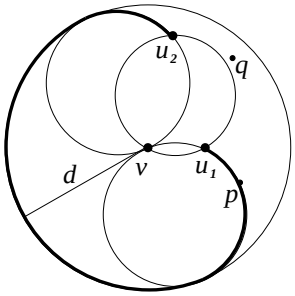

Fig. 3: The thick curve is the set of points p with distance $d_v^U(p) = d$. No empty ball reaches q, so $d_v^U(q) = \infty$.

For efficiency, we need to ensure that the scan does not exceed the certificate region. Therefore, we use a distance function different from the Euclidean one. For any point p, we define the distance $d_v^M(p)$ as the diameter of the smallest certificate ball for p. If no such ball exists then we say $d_v^M(p) = \infty$.

Computing $d_v^M(p)$ exactly is inefficient. However, we can relax the requirement that the balls be empty of any mesh vertices, and instead at time t compute $d_v^{U_t}(p)$ — the diameter of the smallest ball with v and p on its surface that includes no vertex of U_t in its interior. This distance function is non-decreasing in time: adding vertices can only make it harder for a ball to be empty. Therefore, $d_v^{U_t}(p) \leq d_v^M(p)$ for all t. Interestingly, when the scan reaches p, the efficiently-computed lower bound is tight:

Lemma 2 *If point p has $d_v^{U_t}(p) \leq t$, then $d_v^{U_t}(p) = d_v^M(p)$.*

Proof. The assumption of the lemma implies that there exists a ball B of diameter $d_v^{U_t}(p)$ that contains no vertex of U_t. Consider a point q strictly inside B. It has a strictly smaller

CLIPPEDVORONOI(v, β)
1: Let $Q \leftarrow \{\text{square}(v)\}, t_{\max} \leftarrow \infty$, and $U \leftarrow \emptyset$
2: **while** Q contains an element q with $d_v^U(q) < t_{\max}$ **do**
3: $q \leftarrow$ Minimum of Q with respect to d_v^U
4: **if** q is a vertex **then**
5: Add q to U
6: **if** q is the first vertex **then** $t_{\max} \leftarrow \beta|vq|$
7: **if** q is a quadtree square **then**
8: Add each vertex in the square to Q
9: Add every unvisited neighbouring square to Q
10: **return** U

Fig. 4: The CLIPPEDVORONOI algorithm, which performs a scan through the quadtree and mesh vertices.

ball that is empty: $d_v^{U_t}(q) < t$. Because the computed distance function is nondecreasing in t, we must already have visited q, and discovered whether or not q is a vertex of M. In fact, we know it is not because if it were it would be both a vertex in U_t and inside B, contradicting that B is empty. This applies for all $q \in B$: all of B is in fact empty of vertices of M. Monotonicity further implies that B is the smallest empty ball, which proves $d_v^{U_t}(p) = d_v^M(p)$. □

As a corollary, this shows that the $U_{t_{\max}}$ computed by the scan faithfully represents the β-clipped Voronoi cell: The scan visits the entire certificate region, because all points in the certificate region have distance less than $t_{\max}$. It also visits no more than the certificate region, because any point we visit has an empty ball with its center in $V^\beta(v)$.

Implementation (see Figure 4): We discretize the scan using quadtree squares. Starting at square(v), we explore outward from v using a queue Q of events — vertices and squares. Upon processing a vertex, we update the current Voronoi cell U; and upon processing a square, we enqueue the vertices it contains, and the squares it neighbours. We compute $d_v^U(p)$ using a convex program: we find a point c that is the center of an empty ball of minimum radius.

$$\begin{array}{ll} \text{minimize} & |cv| \\ \text{subject to} & |cv| = |xp| \\ & |cv| \geq |cu_i| \text{ for all } u_i \in U \end{array}$$

For a quadtree square, the distance is the minimum distance to any p in the square; this corresponds to letting p be free variables in the above program, and, in dimension d, adding $2d$ constraints on the coordinates of p.

5 Runtime Proof

We prove that CLIPPEDVORONOI and ADDVERTEX are fast in two parts. First, we present a basic geometric fact about empty balls in partially-constructed meshes. This then implies that the certificate region intersects a bounded number of quadtree squares. From there it is an easy corollary to show that CLIPPEDVORONOI and ADDVERTEX run in constant time.

5.1 Points in an empty ball have large lfs

We first show that empty balls do not contain points with small local feature size. We adapt a prior argument that assumed the entire mesh was well-spaced [HMP06]; in the present version, we only require that vertices with small nearest neighbour distance are well-spaced, so that our result will apply in the intermediate stages of a bottom-up algorithm. For clarity, we define a *local mesh size* function induced by the set of vertices the algorithm has output so far (including all of the input points). We say that $\text{lms}_M(p)$ is the distance from p to the second-nearest vertex of the partial mesh M. This differs from the local feature size $\text{lfs}(p)$ in that the local feature size only considers input points, not output vertices. In particular, $\text{lfs}(p) \geq \text{lms}_M(p)$.

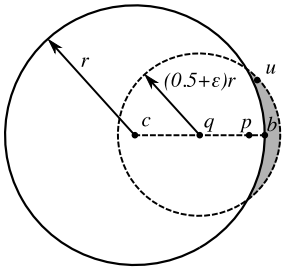

Fig. 5: Setup for the proofs of Section 5.1.

Theorem 3 *Given a mesh M and an empty ball of radius r, assume every vertex u in M with $\text{NN}(u) \leq \gamma r$ is ρ-well-spaced. Then there is a constant ϵ that depends only on γ and ρ such that at any point p in the empty ball, $\text{lms}_M(p) \geq \epsilon r$.*

Proof. If p is at distance at least ϵr from all vertices, we are done. Otherwise, we have the situation in Figure 5. Let c be the center of the empty ball in question. We identify two points on the ray from c to p: q is at distance $r/2$ from the center, and b is on the boundary. Finally, u is the vertex nearest q: q is in the Voronoi cell of u. Lemma 4 will show that $\text{NN}(u) \geq \min(\gamma, \frac{1}{2\rho})r$. This then implies Lemma 5: for appropriate ϵ, u is also the vertex nearest p. The distance from any point in the Voronoi cell of u to a second vertex is minimized at the point x equidistant between u and its nearest neighbour; there, $\text{lms}_M(x) = \text{NN}(u)/2$. Therefore, $\text{lms}_M(p) \geq \frac{\eta}{2}r$. Since $\eta/2 > \epsilon$, the result holds. □

Lemma 4 *The vertex nearest u is at distance $\text{NN}(u) \geq \eta r$, where $\eta = \min(\gamma, \frac{1}{2\rho})$.*

Proof. If $\text{NN}(u) \geq \gamma r$, we are done. Otherwise, we know that u is well-spaced: $\text{NN}(u) \geq R(u)/\rho$. At the same time, we know that $R(u) \geq |qu|$ since q is in the Voronoi cell; and that $|qu| \geq r/2$ because u is outside the empty ball. □

Lemma 5 *If there is some vertex within ϵr of p, then u, the nearest vertex to q, is also the nearest vertex to p.*

Proof. We know there is a vertex u' within distance ϵr of p, and we know that $|pq| \leq r/2$. By the triangle inequality, $|qu'| \leq (1/2 + \epsilon)r$. Then u lies somewhere within the ball centered at q, of radius $(1/2 + \epsilon)r$, since u is nearer to q than is u'. We also know that u must be outside the ball

centered at c, of radius r. These two constraints define a crescent, shown shaded in Figure 5. Consider the plane that goes through c, b, and u. Said plane also contains q since c, b, and q are collinear. We can conformally transform the plane so that c is the origin, $q = \langle 0.5, 0\rangle$, $b = \langle 1, 0\rangle$, and $u = \langle x, y\rangle$. Under this transformation, $r = 1$. The farthest u can be while still lying in the crescent is the point of the crescent where $|cu| = 1$ and $|qu| = 1/2 + \epsilon$. We rewrite the first equality as $x^2 + y^2 = 1$, while the second gives us that $x = 1-\epsilon-\epsilon^2$. Then we can conclude that $|bu|^2 \leq 2-2x = 2(\epsilon+\epsilon^2)$. Recall from the prior lemma that the nearest neighbour of u is at distance η; therefore, every point at distance $\eta/2$ from u is within the Voronoi cell of u. Solving for $|bu| = \eta/2$ we see that by setting $\epsilon = \frac{1}{4}\sqrt{4+2\eta^2} - 1/2$, we ensure that b lies in the Voronoi cell of u. Since both b and q lie in the same convex set, p also lies in it. □

5.2 Queries and updates are fast

Lemma 6 *Assume we are given a size-conforming mesh M, a value r with the guarantee that every vertex u in M with $\mathrm{NN}(u) \leq \gamma r$ is ρ-well-spaced, and a quadtree that satisfies condition (B). Then the certificate region of any vertex v with $\mathrm{NN}(v) \leq r$ intersects $O(1)$ leaf squares of the quadtree.*

Proof. At any x in the certificate region, there is an empty ball containing x that has diameter at least $\mathrm{NN}(v)$. Therefore, Theorem 3 shows that $\mathrm{lfs}(x) \in \Omega(\mathrm{NN}(v))$. Any quadtree square that intersects the query region necessarily includes at least one such x, which, conjoined with condition (B) on the quadtree, implies that the squares each have side length in $\Omega(\mathrm{NN}(v))$. Finally, a volume packing argument applies: within a ball of volume $O(\beta\,\mathrm{NN}(v))^d$, every visited quadtree square consumes $\Omega(\mathrm{NN}(v))^d$ volume. □

Theorem 7 *When a bottom-up mesh refinement algorithm calls* AddVertex *or* ClippedVoronoi*, our data structure responds in constant time.*

Proof. In AddVertex(p, v), if p lies within the β-clipped Voronoi cell of v, then every square visited by the call intersects the certificate region. Upon finding the destination square, AddVertex simply appends to a list. Thus AddVertex does constant work. When the ClippedVoronoi algorithm visits a quadtree square, that square either intersects the certificate region, or is a neighbour of a square that intersects the certificate region. Thanks to the guarantee that squares have a bounded number of neighbours, the latter outnumber the former by at most a constant factor. Thus ClippedVoronoi visits only $O(1)$ squares. The function also does work iterating over the vertices each square contains. By the mesh spacing requirement, a mesh vertex v has a nearest neighbour no closer than $\Omega(\mathrm{lfs}(x))$; meanwhile, the quadtree square that contains v has sides of length $O(\mathrm{lfs}(v))$. Therefore, each square only hosts $O(1)$ mesh vertices, and the total work is $O(1)$. □

6 Higher-dimensional input features

In the present work, we showed how to support a class of mesh refinement algorithms with a search structure that could find appropriate new Steiner points in constant time, assuming the input is a point cloud. Typically, engineers and graphic artists will want the mesh to respect more than just a point cloud: it should respect some meaningful segments and polygons, and perhaps also curves and curved surfaces. Of course, the client application could simply specify a denser packing of points on the surfaces, but ideally the meshing algorithm could handle the features directly. Nothing fundamental blocks the extension of our structure to handle the case of higher-dimensional features (even curved, and with small input angles), although we would need need to enrich the interface our data structure presents.

References

[BEG94] Marshall Bern, David Eppstein, and John R. Gilbert. Provably good mesh generation. *J. Computer and System Sciences*, 48(3):384–409, 1994.

[BET99] Marshall Bern, David Eppstein, and Shang-Hua Teng. Parallel construction of quadtrees and quality triangulations. *IJCGA*, 9(6):517–532, 1999.

[HMP06] Benoît Hudson, Gary L. Miller, and Todd Phillips. Sparse Voronoi Refinement. In *IMR*, pages 339–356, 2006.

[HPÜ05] Sariel Har-Peled and Alper Üngör. A time-optimal Delaunay refinement algorithm in two dimensions. In *SoCG*, pages 228–236, 2005.

[JÜ07] Ravi Jampani and Alper Üngör. Construction of sparse well-spaced point sets for quality tetrahedralizations. In *IMR*, pages 63–80, 2007.

[MV00] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Computing*, 29(4):1334–1370, 2000.

[Rup95] Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[She98] Jonathan Richard Shewchuk. Tetrahedral Mesh Generation by Delaunay Refinement. In *SoCG*, pages 86–95, 1998.

[Tal97] Dafna Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 1997.

[Üng04] Alper Üngör. Off-centers: A new type of Steiner point for computing size-optimal quality-guaranteed Delaunay triangulations. In *LATIN*, pages 152–161, 2004.