A Polynomial Time Algorithm for Multivariate Interpolation in Arbitrary Dimension via the Delaunay Triangulation

Tyler H. Chang
Dept. of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA
thchang@vt.edu

Layne T. Watson
Depts. of Computer Science, Mathematics, and Aerospace & Ocean Engineering
Virginia Polytechnic Institute and State University

Thomas C. H. Lux
Bo Li
Dept. of Computer Science
Virginia Polytechnic Institute and State University

Li Xu
Dept. of Statistics
Virginia Polytechnic Institute and State University

Ali R. Butt
Kirk W. Cameron
Dept. of Computer Science
Virginia Polytechnic Institute and State University

Yili Hong
Dept. of Statistics
Virginia Polytechnic Institute and State University

ABSTRACT

The Delaunay triangulation is a fundamental construct from computational geometry, which finds wide use as a model for multivariate piecewise linear interpolation in fields such as geographic information systems, civil engineering, physics, and computer graphics. Though efficient solutions exist for computation of two- and three-dimensional Delaunay triangulations, the computational complexity for constructing the complete Delaunay triangulation grows exponentially in higher dimensions. Therefore, usage of the Delaunay triangulation as a model for interpolation in high-dimensional domains remains computationally infeasible by standard methods. In this paper, a polynomial time algorithm is presented for interpolating at a finite set of points in arbitrary dimension via the Delaunay triangulation. This is achieved by computing a small subset of the simplices in the complete triangulation, such that all interpolation points lie in the support of the subset. An empirical study on the runtime of the proposed algorithm is presented, demonstrating its scalability to high-dimensional spaces.

KEYWORDS

Delaunay triangulation, multivariate interpolation, high-dimensional triangulation

ACM Reference Format:
Tyler H. Chang, Layne T. Watson, Thomas C. H. Lux, Bo Li, Li Xu, Ali R. Butt, Kirk W. Cameron, and Yili Hong. 2018. A Polynomial Time Algorithm for Multivariate Interpolation in Arbitrary Dimension via the Delaunay Triangulation. In *ACM SE '18: ACM SE '18: Southeast Conference, March 29–31, 2018, Richmond, KY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3190645.3190680

ACM SE '18, March 29–31, 2018, Richmond, KY, USA

ACM ISBN 978-1-4503-5696-1/18/03…$15.00
https://doi.org/10.1145/3190645.3190680

1 INTRODUCTION

With the rise in big data analytics, accurate techniques for performing multivariate approximations that scale to high-dimensional domains have become increasingly valuable. Well-known solutions to the multivariate approximation problem include nearest-neighbor interpolation, least squares approximation, spline based approximation functions, and a host of machine learning regression techniques. Delaunay triangulations are fundamental to computational geometry and are commonly used to perform piecewise linear multivariate interpolation in geographic information systems (GIS), civil engineering, physics, and computer graphics applications [5, 7, 15]. Delaunay triangulations have also been proposed as an effective means for learning nonlinear functions in the context of machine learning [13], but never achieved mass popularity likely due to their impractical computational complexity for high-dimensional data.

Given an underlying function $f : \mathbb{R}^d \to \mathbb{R}^m$, a finite set of points P in $\mathbb{R}^d$, and the response values $f(p)$ for all $p \in P$, an interpolant $\hat{f}$ is an approximation of f that satisfies $\hat{f}(p) = f(p)$ for all $p \in P$. In the case where $d > 1$, $\hat{f}$ is a multivariate interpolant. One method for constructing a multivariate interpolant is to define a mesh of simplices that are disjoint except on their boundaries, have vertices in P, and whose union is the convex hull of P, denoted $CH(P)$. Such a mesh is a d-dimensional triangulation of P.

Let $T(P)$ be a d-dimensional triangulation of P. To define an interpolant in terms of $T(P)$, let $q \in CH(P)$ be an interpolation point, and let S be a simplex in $T(P)$ with vertices $s_1, \ldots, s_{d+1}$ such that $q \in S$. Then there exist unique convex weights $w_1, \ldots, w_{d+1}$ such that $q = \sum_{i=1}^{d+1} w_i s_i$, $\sum_{i=1}^{d+1} w_i = 1$, and $w_i \geq 0$ for $i = 1, \ldots, d+1$. Then the interpolant $\hat{f}_T$ is given by

$$\hat{f}_T(q) = f(s_1)w_1 + f(s_2)w_2 + \ \ldots \ + f(s_{d+1})w_{d+1}. \qquad (1)$$

Note that $\hat{f}_T$ is well-defined, since q is in multiple simplices if and only if q lies on a shared face. In such a case, q can be expressed as a convex combination of only the vertices defining this mutual face. Therefore, the weights associated with all nonmutual vertices must be zero.

The Delaunay triangulation is a specific triangulation that enjoys several properties considered optimal for interpolation [14].

For computing two- and three-dimensional Delaunay triangulations, several $O(n \log n)$ time algorithms exist [16]. However, for higher-dimensional Delaunay triangulations, the computational complexity grows exponentially [11]. This paper considers the case where a user seeks to interpolate at a finite set of points Q using (1) with the Delaunay triangulation of P. A new algorithm is proposed for doing so that runs in polynomial time with respect to the dimension, number of input points, and number of points to be interpolated.

The paper is organized as follows. Section 2 provides relevant definitions and a short summary of related work and challenges. Section 3 contains a detailed description and analysis of the operations performed in the proposed algorithm. Section 4 introduces the proposed algorithm along with an analysis of its complexity. Section 5 raises several issues pertaining to numerical stability and describes how they are addressed in the algorithm's implementation. Section 6 briefly describes a Fortran implementation of the algorithm and presents an empirical analysis of its run time. Section 7 concludes this paper and outlines future work.

2 BACKGROUND

2.1 Definitions

For completeness, first consider the following geometric definitions. A $(d-1)$-sphere in $\mathbb{R}^d$ with center v and radius $r \geq 0$ is given by $C(v,r) = \{x \mid \|x - v\|_2 = r\}$, and the (geometric) interior of C is the open ball $B(v,r) = \{x \mid \|x - v\|_2 < r\}$. Note that $d+1$ affinely independent points in $\mathbb{R}^d$ define a unique $(d-1)$-sphere that passes through these points.

The Delaunay triangulation is defined as the geometric dual of the Voronoi diagram, also known as the Dirichlet tessellation [5, 7]. To obtain a Delaunay triangulation from the Voronoi diagram of a set of n points $P \subset \mathbb{R}^d$, take the set of d-simplices defined by the $d+1$ closest points to each Voronoi vertex. By definition of a Voronoi vertex, there will always be at least $d+1$ points equidistant from each vertex. If there are more than $d+1$ points equidistant from one or more Voronoi vertices, then the division of those points into two or more space-filling d-simplices is arbitrary, and any Delaunay triangulation of P is not unique.

Note that each Voronoi vertex v is the center of a $(d-1)$-sphere $C(v,r)$ through the $d+1$ or more points defining v, such that $B(v,r) \cap P = \emptyset$. This is often referred to as the *empty circumsphere property*. Formalizing this property, the following alternative definition of a Delaunay triangulation is generally preferred. See Figure 1 for a visual.

Definition 2.1. A Delaunay triangulation $DT(P)$ of a finite set of points $P \subset \mathbb{R}^d$ is any triangulation of P such that for each d-simplex $S \in DT(P)$, the $(d-1)$-sphere $C(v,r)$ circumscribing S satisfies $B(v,r) \cap P = \emptyset$.

Note that if all $p \in P$ are contained in some lower-dimensional linear manifold (or equivalently, if $CH(P)$ has zero volume), then all Voronoi edges extend without bound, and there are no Voronoi vertices. Therefore, $DT(P)$ (and in general, any full-dimensional triangulation of P) does not exist. It should be noted that for any set of n points in $\mathbb{R}^d$, if $n < d+1$ then all of the points lie in a $(n-1)$-dimensional linear manifold trivially and no triangulation can exist.

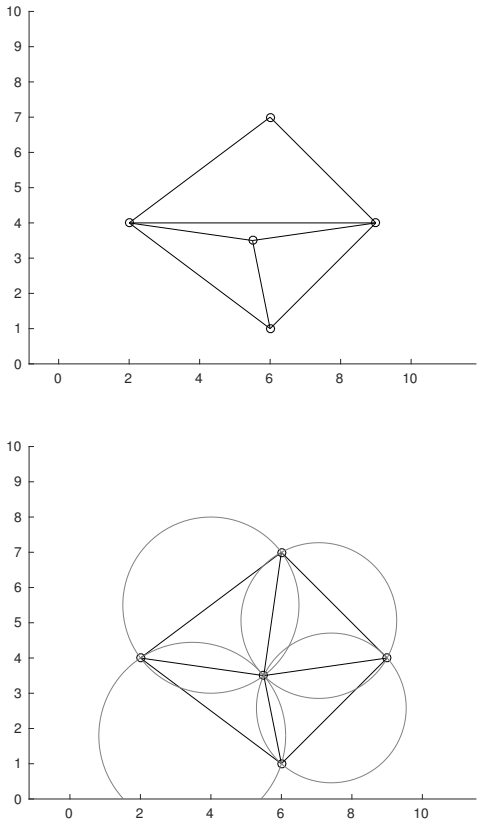

Figure 1: A triangulation in $\mathbb{R}^2$ (top) and the Delaunay triangulation (bottom).

In the context of interpolation, the case where $n \geq d+1$ and still no triangulation exists can be interpreted as an over-parameterization of the underlying function f.

Except in degenerate cases, there will be exactly $d+1$ points on the boundary of every $(d-1)$-sphere, and $DT(P)$ is unique. Generally, if $n \geq d+1$, it can be assumed that $DT(P)$ exists and is unique. Since the existence and uniqueness of $DT(P)$ is generally presumed, it is customary to reference *the* Delaunay triangulation of a set of points. This can be formalized with the following definition.

Definition 2.2. A set of points P in $\mathbb{R}^d$ is said to be in *general position* if P does not lie in some $(d-1)$-dimensional linear manifold, and if no $d+2$ points in P lie on the same $(d-1)$-sphere.

Recall that every Delaunay simplex is circumscribed by a $(d-1)$-sphere $C(v,r)$ satisfying $B(v,r) \cap P = \emptyset$. Additionally, consider any set $s = \{s_1, \ldots, s_{d+1}\}$ of $d+1$ points in P that defines a $(d-1)$-sphere $C(v,r)$ with $B(v,r) \cap P = \emptyset$. Then no points in $P \setminus s$ are closer to v than the points in s. Therefore v must be a Voronoi vertex, and s is the set of vertices of a d-simplex in $DT(P)$. This leads to the following useful equivalence.

Observation 2.3. A set of $d+1$ points $s \subset P$ is the set of vertices of a d-simplex $S \in DT(P)$ if and only if s defines a $(d-1)$-sphere $C(v,r)$ with $B(v,r) \cap P = \emptyset$.

2.2 Optimality of the Delaunay Interpolant

In this paper, the Delaunay interpolant will refer to the function $\hat{f}_{DT}$, as defined in (1), where DT is a Delaunay triangulation. The Delaunay triangulation has several favourable properties that make $\hat{f}_{DT}$ optimal with respect to all piecewise linear interpolants of the form $\hat{f}_T$. The following results, originally shown by Rajan [14], demonstrate some of these favourable properties, many of which are analogous to properties associated with "quality" finite element meshes [5].

- The Delaunay triangulation uniformly minimizes the radius of the min-containment sphere over all simplices, with respect to all other triangulations.
- Consider any point $x \in P$ and let $B_x^T = B_1^T \cup B_2^T \cup \ldots \cup B_k^T$ where $\{B_i^T\}$ is the set of circumballs defined by all simplices in T containing x as a vertex. Then for all $T(P) \neq DT(P)$
$$B_x^{DT} \cap CH(P) \subset B_x^T \cap CH(P).$$
- Define an edge of a simplex to be a 1-face of that simplex, and let the length of an edge denote the Euclidean distance between the two vertices defining the 1-face. Then the Delaunay triangulation minimizes the weighted sum of squares of all edge lengths where the weight is given by the sum of the volumes for all simplices incident to that edge.
- Define a simplex S to be "self-centered" if its circumsphere $C(v,r)$ satisfies $v \in S$. Then if a triangulation $T(P)$ exists such that all $S \in T(P)$ are self-centered, then $T(P) = DT(P)$.

2.3 Related Works and Limitations

Efficient solutions exist for computing Delaunay triangulations in two- and three-dimensions, running in $O(n \log n)$ time [16]. However, in $\mathbb{R}^d$, the worst case size of the Delaunay triangulation is known to be $O(n^{\lceil d/2 \rceil})$ [11]. Even in the common case, the Delaunay triangulation still tends to grow exponentially with the dimension. This phenomenon is often referred to as the curse of dimensionality.

In spite of this, many attempts have been made to compute Delaunay triangulations in arbitrary dimension d. The earliest algorithm proposed for computing arbitrary-dimensional Delaunay triangulations was proposed independently by both Bowyer [4] and Watson [17] in the same issue of *The Computer Journal*. In the Bowyer-Watson algorithm, points are inserted incrementally; and with each insertion, the triangulation is refined to be Delaunay.

Quickhull [1] is one of the most time efficient methods for computing high-dimensional Delaunay triangulations. It also boasts a robust and numerically stable implementation, which is currently used as the intrinsic for high-dimensional Delaunay triangulation computation in `Matlab`, `SciPy`, and `R`. Quickhull takes advantage of the Delaunay triangulation's relationship with convex hulls, lifting P onto the bottom of a bowl in $\mathbb{R}^{d+1}$, computing the convex hull of the lifted set, then projecting the facets of the convex hull back down into the original space to get the Delaunay triangulation in $\mathbb{R}^d$.

The Delaunay Graph approach [2] is loosely based off the algorithm described in [3], which is at its core, a randomized variation of the Bowyer-Watson algorithm. However, to avoid the explosion in storage overhead, this algorithm (at the expense of computation time) forces the Delaunay triangulation into a graph format that can be stored in $O(n^2)$ space. This algorithm is currently used in the Computational Geometry Algorithms Library (CGAL).

The final algorithm of interest is the Delaunay Wall (DeWall) algorithm [6]. The DeWall algorithm is a *divide-and-conquer* based approach that offers an interesting strategy for guided construction of a Delaunay simplex "wall."

It should be noted that none of the above mentioned algorithms are believed to scale past eight dimensions for large data sets.

3 NECESSARY OPERATIONS

The goal of this work is a Delaunay interpolation algorithm that scales polynomially to high-dimensional spaces. Recall from Section 2.3 that the worst case size of the Delaunay triangulation is $O(n^{\lceil d/2 \rceil})$. Therefore, any algorithm that requires the computation of the complete Delaunay triangulation cannot scale. To circumvent this limitation, the following observation is necessary.

Observation 3.1. Given a finite set of points P in $\mathbb{R}^d$, the Delaunay interpolant $\hat{f}_{DT}(q)$ for some $q \in CH(P)$ can be exactly computed given the vertices $s = \{s_1, \ldots, s_{d+1}\}$ of any simplex $S \in DT(P)$ such that $q \in S$.

This reduces the problem of computing $DT(P)$ to that of computing a specific simplex in $DT(P)$. Given an interpolation point q and a point set P, the proposed algorithm computes a polynomial sized subset of $DT(P)$ such that q lies in the support of that subset. Given the vertices of a simplex in $DT(P)$ containing q, the computation of $\hat{f}_{DT}(q)$ via (1) is trivial.

In this section, the machinery for computing the previously described containing Delaunay simplex is developed. The correctness of the proposed algorithm will follow from the correctness of these operations. There are three basic operations that need to be defined and proven. They are the construction of an initial Delaunay simplex, the completion of an open Delaunay facet, and the visibility walk. The construction of a Delaunay simplex and the completion of an open facet are also described as the basis for both *incremental construction* and *divide-and-conquer* paradigms in [6]. The visibility walk finds wide use in several other Delaunay triangulation and point location algorithms [2–4, 12, 17].

3.1 Growing a Delaunay Simplex

To begin, the following definition of a Delaunay face is useful.

Definition 3.2. Let P be a set of points in $\mathbb{R}^d$. Let F be a k-face with vertices in P where $0 \leq k \leq d$. Then F is a *Delaunay face* if the smallest (radius) $(d-1)$-sphere $C(v,r)$ circumscribing F satisfies $B(v,r) \cap P = \emptyset$.

Note that if $k = d$, then by Observation 2.3, F is a Delaunay simplex. The following lemma shows how to grow a complete Delaunay simplex from an arbitrary Delaunay face.

Lemma 3.3. *Let P be a set of points in $\mathbb{R}^d$ in general position, and let F be a Delaunay k-face with vertices $\phi \subset P$ where $k < d$. Let*

$\phi^* = \phi \cup \{p^*\}$ where $p^* \in P \setminus \phi$ minimizes the radius of the smallest $(d-1)$-sphere $C_{\phi \cup \{p\}}$ through the points in $\phi \cup \{p\}$, over all $p \in P \setminus \phi$. Then F^*, the $(k+1)$-face with vertices ϕ^*, is also a Delaunay face.

Proof. The set of centers of $(d-1)$-spheres $C_\phi(v, r)$ through ϕ is a $(d-k)$-dimensional linear manifold $\mathcal{M}_\phi$ (the solution set of k linear equations), and similarly for ϕ^*, the $(d-k-1)$-dimensional linear manifold $\mathcal{M}_{\phi^*}$. The center v^* of the smallest $(d-1)$-sphere $C_{\phi^*}(v^*, r^*)$ through ϕ^* is the projection of p^* onto $\mathcal{M}_{\phi^*}$ (the closest point in $\mathcal{M}_{\phi^*}$ to p^*). If there were a $p \in P \setminus \phi$ with $p \in B(v^*, r^*)$, then $\|p - v^*\|_2 < r^*$. Then by continuity $B(v^*, r^*) \cap \mathcal{M}_\phi$ contains a point z equidistant from all the points in ϕ and p, with $\|z - p\|_2 < r^*$, and hence there exists a $(d-1)$-sphere centered at z through $\phi \cup \{p\}$ of radius less than r^*, which contradicts the definition of p^* and r^*. Therefore F^*, the simplex with vertices ϕ^*, must be a Delaunay face. □

The smallest $(d-1)$-sphere containing a single point is simply the point itself, whose open ball is the empty set. Therefore, any point in P is a Delaunay 0-face trivially. So, starting with an arbitrary point in P and applying the following algorithm d times, the vertices of a full Delaunay simplex S are found. The correctness of this approach follows immediately from Lemma 3.3.

Algorithm 1, computes the vertices $\phi^* \supset \phi$ of a Delaunay $(k+1)$-face from the vertices ϕ of a Delaunay k-face.

Let P be a set of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$ in general position.
Let $\phi \subset P$ be the vertices of a Delaunay k-face where $k < d$.
Let r_i denote the minimum radius of a $(d-1)$-sphere containing ϕ and the point p_i.

$r_{min} := \infty$;
for $i := 1, \ldots, n$ **do**
  if $r_i < r_{min}$ **and** $p_i \notin \phi$ **then**
    $\hat{p} := p_i$;
    $r_{min} := r_i$;
  end if
end for
return $\phi^* := \phi \cup \{\hat{p}\}$;

3.2 Completing an Open Facet

Before presenting the algorithm for completing an open facet, the following observation is helpful. A visual representation of Observation 3.4 is presented in Figure 2.

Observation 3.4. Let $DT(P)$ be the Delaunay triangulation of a set of points P in $\mathbb{R}^d$. Given a facet F of some simplex $S \in DT(P)$, let $p_1, \ldots, p_n$ be a sequence of points in P and in a halfspace H with hyperplane boundary containing F. Define the circumspheres $C_1, \ldots, C_n$ with corresponding open balls $B_1, \ldots, B_n$ such that each C_k contains σ and p_k. Assume the sequence $p_1, \ldots, p_n$ satisfies $p_k \in B_{k+1}$ for all $1 \le k < n$. Then $B_1 \cap H \subset B_2 \cap H \subset \ldots \subset B_n \cap H$.

The proof of this observation has been omitted but follows from a simple continuity argument. Given Observation 3.4, the following algorithm is proposed for completing an open facet of a Delaunay simplex.

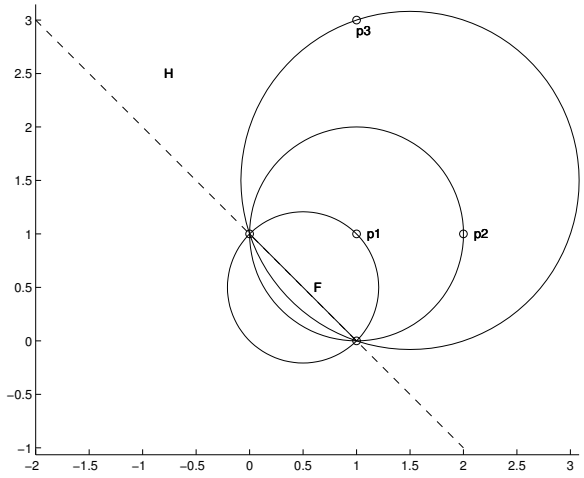

Figure 2: A 2D visualization of Observation 3.4. The solid line segment is a facet F, H is the halfspace above the dashed line, and $\{p1, p2, p3\}$ is a sequence satisfying $p1 \in B_2$ and $p2 \in B_3$.

Algorithm 2, computes the set of vertices s of a Delaunay simplex in the halfspace H from the vertices σ of a Delaunay facet F lying in the boundary hyperplane of H.

Let P be a set of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$ in general position.
Let σ be the vertices of a facet F of a Delaunay simplex.
Let H be a halfspace with hyperplane boundary containing F.
Let $P_H = (P \setminus \sigma) \cap H$.
Let B_i denote the open ball defined by the $(d-1)$-sphere containing σ and p_i.

$B := H$;
$\hat{p} := p_0$; Note, p_0 is a dummy vertex.
for $i := 1, \ldots, n$ **do**
  if $p_i \in P_H$ **and** $p_i \in B$ **then**
    $\hat{p} := p_i$;
    $B := B_i$;
  end if
end for
if $\hat{p} = p_0$ **then**
  σ must define a facet of $CH(P)$
  return $\emptyset$;
else
  return $s := \sigma \cup \{\hat{p}\}$;
end if

Claim 3.5. *Let P be in general position. Then Algorithm 2 correctly computes the vertices of a Delaunay simplex.*

Proof. First, note that if $\emptyset$ is returned, then the set $P_H = (P \setminus \phi) \cap H$ was empty. Then the algorithm has correctly determined that σ defines a facet of $CH(P)$, and no simplex can exist in H. Now suppose a set of vertices s was returned. Since P is in general position, for every pair of points $p_1, p_2 \in P_H$, it must be that either $p_1 \in B_2$ or $p_2 \in B_1$. Therefore, by Observation 3.4, the added vertex $\hat{p}$ is in the open ball defined by the circumsphere of *every* other

point in P_H. So, $\hat{p}$ is the *only* point in P_H that could define the vertices for a Delaunay simplex in conjunction with the set σ. It has been given that σ is the set of vertices of a Delaunay facet and the Delaunay triangulation exists since the points are in general position. Therefore, $s = \phi \cup \{\hat{p}\}$ must be the vertices of a Delaunay simplex. □

3.3 Visibility Walk

The final ingredient in the proposed algorithm will be a visibility walk. Begin with the following definition.

Definition 3.6. Let S, U be simplices in $DT(P)$. Then $S <_x U$, (x is visible to S with respect to U), if every ray ℓ drawn from x that has a nonempty intersection with both S and U intersects S "before" U, i.e., for all $y \in S \cap \ell$ and for all $z \in U \cap \ell$, y is between x and z on ℓ.

A visibility walk is defined by a sequence of simplices $S_1, \ldots, S_n$ that satisfy $S_n <_x S_{n-1} <_x \ldots <_x S_1$. Using Algorithm 2, a visibility walk could be performed from an arbitrary simplex to the simplex containing an interpolation point $q \in CH(P)$, by completing a shared facet between neighboring S_i and S_{i+1} such that $S_{i+1} <_q S_i$. From the definition, it is clear that if F_i is the shared facet between S_i and S_{i+1} and S_{i+1} is on the same side of the hyperplane containing F_i as q, then $S_{i+1} <_q S_i$. Therefore, a "next step" in the simplex walk always exists unless S_i contains q. However, it is not clear whether a given simplex walk will ever terminate, (i.e., whether it will ever arrive at a S_n containing q). Clearly, if a transitivity property can be established, then the walk terminates since there are finitely many simplices in any given triangulation. The following definition formalizes this.

Definition 3.7. A triangulation $T(P)$ is said to be acyclic with respect to some viewpoint x if for all $S, U, V \in T(P)$, $S <_x U$ and $U <_x V$ implies $U <_x V$.

In general, an arbitrary triangulation $T(P)$ is not acyclic, and a visibility walk to locate some $S \in T(P)$ containing $q \in CH(P)$ may go into an infinite loop and fail to terminate. However, it is known that Delaunay triangulations are acyclic.

Theorem 3.8. *Delaunay triangulations are acyclic with respect to a fixed viewpoint x.*

This theorem was originally shown by Edelsbrunner, and a detailed proof is in his original paper [8]. However, for the sake of completeness, consider the following abbreviated interpretation of Edelsbrunner's proof.

Proof. If there exists a function $\Phi_x : DT(P) \to \mathbb{R}$ such that $S <_x U \Rightarrow \Phi_x(S) < \Phi_x(U)$, then the proof follows trivially since it cannot be that

$$\Phi_x(S_1) < \Phi_x(S_2) < \ \ldots \ < \Phi_x(S_k) < \Phi_x(S_1).$$

Define $\Phi_x(S) = \|x - v_S\|_2^2 - r_S^2$, where $C(v_S, r_S)$ is the $(d-1)$-sphere circumscribing S. Assume $S, U \in DT(P)$ such that $S <_x U$. Then any ray originating at x passes through S before U. Since S and U have mutually empty circumballs (with respect to each others' vertices), it follows that $\Phi_x(S) < \Phi_x(U)$. □

4 DELAUNAY INTERPOLATION ALGORITHM AND ANALYSIS

4.1 Algorithm Description

Using the three operations outlined in Section 3, it is possible to grow an initial Delaunay simplex then "walk" toward the simplex containing an interpolation point $q \in CH(P)$ by completing open facets such that the sequence $S_1, \ldots, S_k$ of constructed Delaunay simplices satisfies $S_k <_q S_{k-1} <_q \ldots <_q S_1$.

Note, that for every $S \in DT(P)$ and $q \in \mathbb{R}^d$, q is a *unique* affine combination of the vertices of S. Furthermore, if q is *in* S, then this affine combination will also be convex by the definition of a simplex. Therefore, one can terminate based on the nonnegativity of the affine weights for q with respect to the vertices of the current simplex S. Furthermore, if q is not in S, then at least one of the weights will be negative, corresponding to a point that should be dropped to advance "closer" to q with respect to $<_q$. Conveniently, upon termination, the convex weights for q are the exact weights needed to linearly interpolate at q within S, as defined in (1). Pseudo code for the proposed algorithm is provided below.

Algorithm 3, computes the value of the Delaunay interpolant $\hat{f}_{DT}(q)$ for $q \in CH(P)$.

Let P be a set of n points in $\mathbb{R}^d$ in general position.
Let $q \in CH(P)$ be a point to interpolate at.
Let $f_i = f(p_i)$ be known for all $p_i \in P$, and let $f(s)$ denote the set $\{f_{i_1}, f_{i_2}, \ldots, f_{i_{d+1}}\}$ for a set s of $d+1$ points $\{p_{i_1}, p_{i_2}, \ldots, p_{i_{d+1}}\}$ in P.
Let MakeFirstSimplex(P) be a function that grows the set of vertices of a Delaunay simplex from points in P as described in Algorithm 1.
Let CompleteSimplex(σ, q, P) be a function that completes the facet defined by the set of vertices σ with a point from P that is on the same side of the hyperplane containing σ as q; as described in Algorithm 2.
Let AffineWeights(q, s) be a function that returns the affine weights that give q as a combination of the vertices s.
Let MinIndex(w) return the index of the most negative element w_i.
Let DropVertex(s, i) return the set $s \setminus \{s_i\}$ where s_i is the ith element of s.

```
s := MakeFirstSimplex(P);
w := AffineWeights(q, s);
while w_i < 0 for some 1 ≤ i ≤ d + 1 do
    i := MinIndex(w);
    σ := DropVertex(s, i);
    s := CompleteSimplex(σ, q, P);
    w := AffineWeights(q, s);
end while
return f̂_DT(q) := ⟨w, f(s)⟩ (inner product);
```

The correctness of Algorithm 3 follows from the correctness of the operations in Sections 3.1 - 3.3. The simplices constructed will always be Delaunay by Algorithms 1 and 2, and the while loop will terminate in finite time by Theorem 3.8.

4.2 Time Complexity

The construction of the first simplex, as defined in Algorithm 1, can be formulated as a sequence of least squares (LS) problems ranging in size from $2 \times d$ to $d \times d$. Each LS problem can be solved in $O(d^3)$ time. At all $d-1$ sizes, one must solve up to n LS problems, taking the point that produces the minimum residual as the next vertex. Therefore, the total computation time for the first simplex will be $O(nd^4)$.

To complete a simplex (in one iteration of the visibility walk) requires at most n linear solves, performed in $O(nd^3)$ total time. Therefore, the total time complexity of the visibility walk is given by $O(nd^3k)$, where k is the number of iterations required to converge on the interpolation point. Bowyer [4] claims that for uniformly distributed input points P, the expected length of a walk starting from the center of the Delaunay triangulation is $O(n^{1/d})$. However, no proof is provided so this remains speculative. Mücke, Saias, and Zhu [12] prove that in up to three dimensions, a randomized variant of the visibility walk converges in $O(n^{\frac{1}{d+1}})$ iterations, but the proof does not generalize past three dimensions.

Assuming Bowyer's claim holds, using the proposed algorithm to interpolate at m points from a set of n points in $\mathbb{R}^d$ will take $O(mn^{1+\frac{1}{d}}d^3 + mnd^4)$ expected time. This is a significant improvement over the exponential time required for computing the entire Delaunay triangulation.

4.3 Space Complexity

Recall from Section 2.3 that the size of the Delaunay triangulation grows exponentially with the dimension. Therefore, space complexity is equally as concerning as time complexity since, for large d, one cannot store the exponentially sized triangulation in memory. Another advantage of the proposed algorithm is that any computed simplex that does not contain any interpolation point can be discarded immediately after one of its open facets has been completed.

Therefore, the required space for computing the Delaunay interpolant is reduced to:

- $O(nd)$ space for the n input points in $\mathbb{R}^d$;
- $O(md)$ space for storing the m interpolation points in $\mathbb{R}^d$, the m containing simplices of size $d+1$ each, and the m convex coordinate vectors of size $d+1$ each;
- $O(d^2)$ space for storing the $d \times d$ matrices involved in performing linear solves;
- Other temporary storage arrays that require $O(d)$ space.

This makes the total space complexity $O(nd + md + d^2)$. Since no triangulation can exist unless $n > d$, the space complexity can be further reduced to $O(nd + md)$, which is approximately the same size as the input.

4.4 Optimizations

There are several optimizations that are easily implemented to improve the performance of the proposed algorithm. First, in a slight modification to Algorithm 3, one could identify the point $\hat{p} \in P$ that is a nearest neighbor to q with respect to Euclidean distance in $O(n)$ time and build the first simplex off of $\hat{p}$ instead of an arbitrarily chosen point. As seen in Table 1, for interpolating at a single point, this typically leads to location of the simplex containing q in $O(d \log d)$ iterations of the visibility walk with very little dependency on n.

Table 1: Average number (with a sample size of 20) of Delaunay simplices computed in a simplex walk from the simplex built off the nearest neighbor to q for n pseudo-randomly generated points in d dimensions.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
$d = 2$	3.05	2.90	3.25	3.10
$d = 8$	23.75	24.75	24.30	23.10
$d = 32$	95.25	125.60	131.85	150.10
$d = 64$	171.95	221.85	248.35	280.60

When interpolating many points, it is often the case that some simplex or simplices in the walk contain interpolation points that have not yet been resolved. With no increase in total time complexity, it is possible to check if the current simplex contains *any* of the unresolved interpolation points. If the points being interpolated are tightly clustered, it is typical for them to all be contained in a small number of Delaunay simplices, significantly reducing total computation time.

Furthermore, as will be described in Section 5, when numerical stability is considered, construction of the first simplex becomes a dominant factor. Then it often becomes optimal to "daisy chain" visibility walks, walking from each solution simplex to the next without ever recomputing an initial simplex. Note that when this approach is taken, the previous complexity analyses regarding the length of the visibility walk no longer apply, and not much can be said about the length of the walk. However, in practice, for a moderate number of interpolation points the walk still tends to locate all simplices in relatively few steps and significantly less time than what would be required to compute the complete triangulation.

4.5 Extrapolation

Up until this point, the proposed algorithm has only covered interpolation cases (when q is in $CH(P)$). Often, however, it is reasonable to make a prediction about some extrapolation points Z that are *slightly* outside $CH(P)$. In these cases, it is most reasonable to project each $z \in Z$ onto $CH(P)$ and interpolate at each projection $\hat{z}$, provided the residual $r = \|z - \hat{z}\|_2$ is small. The projection of z onto $CH(P)$ can be easily reformulated as an inequality constrained least squares problem, whose efficient solution is described in [10]. Note that the time and space complexity for performing the projection could be expensive compared to what is required for computing the Delaunay interpolant.

5 ISSUES IN STABILITY

An important assumption that has been made up until this point is that P is in general position. However, in real world applications, it is possible that P could be some degenerate set. There are two cases that lead to meaningful degeneracies.

- P could be contained in some lower-dimensional linear manifold.

- There could exist $d+2$ or more points in P that lie on the same $(d-1)$-sphere.

A special case of the second listed degeneracy, is the case where $k+1$ points in P lie on some $(k-1)$-dimensional linear manifold *and* on some $(k-2)$-sphere embedded in that manifold. This leads to a situation where Algorithms 1 and 2 can fail by selecting points that minimize the radius of the resulting $(d-1)$-sphere but define a degenerate k-face.

5.1 Dealing with Degeneracies

The situation where all points lie on a lower-dimensional linear manifold will always result in a situation where no new point can be "added" to the set of vertices during the construction of the first Delaunay simplex without making some face degenerate. This situation is easily detected via a check for rank deficiency and need not be handled since users can apply dimension reduction techniques to construct an equivalent non degenerate problem.

In the case where more than $d+1$ points lie on some $(d-1)$-sphere, one would like to still obtain *a* Delaunay interpolant, though it will no longer be unique. Exactly which Delaunay interpolant is obtained is actually unimportant, since all Delaunay interpolants are equally optimal. Since degeneracies occur with zero probability, one solution to this problem could be to perturb each $p \in P$ by some small random amount such that the perturbed set $\rho(P)$ is in general position [9].

Alternatively, one could handle degeneracies by constantly checking that each Delaunay k-face created is *not* embedded in some $(k-1)$-dimensional manifold (including the case where $k=d$ so the k-face is a d-simplex). This would mean checking while growing the simplex that no k-face is degenerate, and checking during the walk that each added point is not directly on the hyperplane containing the current facet. Implementing these checks ensures that a legal Delaunay simplex is ultimately found (as opposed to a "flat" simplex with zero volume). Finally, in the case where two points p_1 and p_2 would define simplices with the same circumsphere in conjunction with the current vertices ϕ, the choice between adding p_1 and p_2 can be made arbitrarily since both solutions produce legal Delaunay simplices.

In this work, the latter solution is preferred over ad hoc perturbation since it allows for degeneracies to be handled at minimal computational cost and is easily adapted to handle floating-point error. A drawback of this method is that in the degenerate case, the simplices obtained containing two nearby interpolation points (though both simplices will be Delaunay) may not come from the *same* Delaunay triangulation. This can create discontinuities in the "surface" defined by the interpolant, but does not detract from interpolation accuracy.

5.2 Dealing with Imprecision

The degeneracy problem is only exacerbated in the context of a floating-point paradigm. Let ϵ be the working precision of the floating-point environment. In this section, the case is considered where an error of magnitude ϵ could occur in any computation.

Most notably, instead of checking whether the vertices of a Delaunay k-face lie *exactly* in a $(k-1)$-dimensional linear manifold, it is now necessary to check whether they are *nearly* in a $(k-1)$-dimensional linear manifold. Consider the $d \times k$ matrix A whose ith column is given by $A_i = p_{i+1} - p_1$ where p_i is the ith vertex of the current face. Then the vertices of the defining k-face are "nearly" in a $(k-1)$-dimensional linear manifold if the ratio between the smallest and largest singular values σ_k/σ_1 is less than ϵ.

Note that singular value decompositions are expensive and generally computed via iterative methods. So to save time, a check for rank deficiency should only be performed after constructing the complete first simplex. Then, if $\sigma_d/\sigma_1 < \epsilon$, the entire simplex must be reconstructed with a check after each insertion so that a vertex that causes a degeneracy can be immediately discarded. The case where all vertices cause degeneracy at step k of the initial simplex construction implies that P is contained in a $(k-1)$-dimensional linear manifold "up to the working precision" (where $k \leq d$) so no triangulation exists. After constructing the initial simplex, stability can be ensured by only considering points that are a distance of ϵ off the hyperplane containing the current facet.

It is also possible that a set of points that do *not* define a Delaunay face could be computed as such due to floating-point error. For the purpose of interpolation, this is not considered to be an issue since the vertices of the resulting simplex must be within a perturbation of magnitude ϵ of defining a Delaunay simplex. Therefore, the computed simplex and the true Delaunay simplex can be thought of as having the same circumspheres up to the working precision, so the choice between them can be made arbitrarily.

Finally, points that are on or near the boundary of their containing simplex could be perturbed outside that simplex by floating-point error. This could result in an infinite loop if the point is not perturbed identically when checked for in the adjacent simplices. Therefore, a simplex should be accepted as containing q if the affine weights computed in Algorithm 3 are all greater than $-\epsilon$ (i.e., positive up to the working precision).

On a final note, the tolerance ϵ should be proportional to the machine precision and should be scale/shift invariant. To achieve scale/shift invariance, the input data set is translated and rescaled to be centered about the origin with a radius of one.

6 IMPLEMENTATION

A serial numerically stable implementation of the proposed algorithm has been coded in ISO Fortran 2003. Note that the value chosen for the working precision was $\epsilon = \sqrt{\mu}$ where μ is the unit roundoff. This code was tested for correctness on over 10,000 data sets ranging in size from $n = 3$ to $n = 800$ and ranging in dimension from $d = 1$ to $d = 4$. For each data set, anywhere from one to several hundred interpolation points were predicted using both the described algorithm and the standard implementation of Quickhull. On the non degenerate data sets, the code performed identically to Quickhull up to a modest multiple of the machine precision. On the degenerate data sets, a small sample of the largest discrepancies were hand checked and confirmed to be resulting from degeneracies. The described real world data sets were gathered at Virginia Tech for usage in the VarSys project. The data was also tested on a small number of pseudo-random data sets in five dimensions, and due to the lack of degeneracy in the random data, the algorithm

always computed simplices from Quickhull's computed Delaunay triangulation.

Using the proposed algorithm, run times were gathered for pseudo-randomly generated point sets in up to 64 dimensions. Note that the problem of computing a small number of simplices in the triangulation is inherently easier than the problem of computing an exponentially large triangulation. Subsequently, the proposed algorithm can be timed in much higher dimensional spaces, where computation via its competitors is either painfully slow or totally infeasible. However, previous studies have managed to evaluate the runtime of Quickhull and several other competitors in five dimensions, and the runtime tables in Section 5 of [2] could be compared to the data in Tables 2 and 3.

The following run times were gathered on an Intel i7-3770 CPU @3.40 GHz running CentOS release 7.3.1611. All averages are based on a sample size of 20 runs, each performed on a different pseudo randomly generated data set. Table 2 details average run times for interpolating at uniformly distributed interpolation points on a five-dimensional uniformly distributed data set. Table 3 details average run times for interpolating at clustered interpolation points (limited to a hypercube with 10% of the original point-set's diameter) on a five-dimensional uniformly distributed data set. Table 4 details average run times for interpolating at a single point in up to 64 dimensions over uniformly distributed data sets ranging in size from 2K points up to 32K points. All input data sets consist of pseudo-randomly generated points in the unit hypercube, generated using the Fortran intrinsic random number generator. Times were recorded with the Fortran intrinsic `CPU_TIME` function, which is accurate up to either microsecond resolution or the precision of the system clock.

Table 2: Average runtime in seconds for interpolating at uniformly distributed interpolation points for n pseudo-randomly generated input points in 5 dimensions.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
32 interp. pts	0.3 s	2.7 s	9.6 s	35.7 s
1024 interp. pts	2.5 s	11.6 s	28.9 s	79.1 s

Table 3: Average runtime in seconds for interpolating at clustered interpolation points for n pseudo-randomly generated input points in 5 dimensions.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
32 interp. pts	0.2 s	2.2 s	8.4 s	33.0 s
1024 interp. pts	0.2 s	2.5 s	9.2 s	35.2 s

Note that in the numerically stable implementation described, expensive checks for rank deficiency, computation of the diameter of P (an $O(n^2)$ operation), and daisy-chaining the simplex walk as described in Section 4.4, leads to computation times that do not scale as predicted in Section 4. However, the resulting algorithm is still able to robustly compute the Delaunay interpolant on a moderately sized data set in up to 64 dimensions, and the empirical results clearly indicate a sub-exponential time complexity.

Table 4: Average runtime in seconds for interpolating at a single point for n pseudo-randomly generated input points in d-dimensional space .

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
$d = 2$	0.1 s	1.7 s	6.8 s	27.0 s
$d = 8$	0.2 s	2.5 s	9.6 s	37.9 s
$d = 32$	1.4 s	9.5 s	29.7 s	101.1 s
$d = 64$	13.2 s	60.1 s	138.6 s	349.1 s

7 CONCLUSION AND FUTURE WORK

In this paper, a new Delaunay triangulation algorithm is proposed for interpolating in point clouds in arbitrary dimension d. This is achieved by computing a relatively small number of simplices from the complete Delaunay triangulation. A robust numerically stable implementation is empirically shown to scale to high-dimensional spaces. In future work, the methods used to construct and locate a single Delaunay simplex for the purpose of interpolation could be extended for other Delaunay triangulation applications. In particular, knowledge of a Voronoi cell can be achieved by computing the star of simplices incident at a given vertex.

REFERENCES

[1] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. 1996. The Quickhull Algorithm for Convex Hulls. *ACM Transactions on Mathematical Software (TOMS)* 22, 4 (1996), 469–483.

[2] Jean-Daniel Boissonnat, Olivier Devillers, and Samuel Hornus. 2009. Incremental Construction of the Delaunay Triangulation and the Delaunay Graph in Medium Dimension. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*. ACM, 208–216.

[3] Jean-Daniel Boissonnat and Monique Teillaud. 1993. On the randomized construction of the Delaunay tree. *Theoretical Computer Science* 112, 2 (1993), 339–354.

[4] Adrian Bowyer. 1981. Computing Dirichlet tessellations. *Comput. J.* 24, 2 (1981), 162–166.

[5] Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. 2012. *Delaunay Mesh Generation*. CRC Press.

[6] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. 1998. DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in $\mathbb{E}^d$. *Computer-Aided Design* 30, 5 (1998), 333–341.

[7] Mark de Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (third ed.). Springer-Verlag Berlin Heidelberg.

[8] Herbert Edelsbrunner. 1989. An acyclicity theorem for cell complexes in d dimensions. In *Proceedings of the fifth annual symposium on Computational geometry*. ACM, 145–151.

[9] Herbert Edelsbrunner and Ernst Peter Mücke. 1990. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics (TOG)* 9, 1 (1990), 66–104.

[10] Richard J Hanson and Karen H Haskell. 1982. Algorithm 587: Two Algorithms for the Linearly Constrained Least Squares Problem. *ACM Transactions on Mathematical Software (TOMS)* 8, 3 (1982), 323–333.

[11] Victor Klee. 1980. On the complexity of d-dimensional Voronoi diagrams. *Archiv der Mathematik* 34, 1 (1980), 75–80.

[12] Ernst P Mücke, Isaac Saias, and Binhai Zhu. 1999. Fast randomized point location without preprocessing in two-and three-dimensional Delaunay triangulations. *Computational Geometry* 12, 1-2 (1999), 63–83.

[13] Stephen M Omohundro. 1990. Geometric Learning Algorithms. *Physica D: Nonlinear Phenomena* 42, 1-3 (1990), 307–321.

[14] VT Rajan. 1994. Optimality of the Delaunay Triangulation in $\mathbb{R}^d$. *Discrete & Computational Geometry* 12, 2 (1994), 189–202.

[15] WE Schaap and R Van De Weygaert. 2000. Continuous Fields and Discrete Samples: Reconstruction through Delaunay Tessellations. *Astronomy and Astrophysics* 363 (2000), L29–L32.

[16] Peter Su and Robert L Scot Drysdale. 1995. A Comparison of Sequential Delaunay Triangulation Algorithms. In *Proceedings of the eleventh annual symposium on Computational geometry*. ACM, 61–70.

[17] David F Watson. 1981. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Comput. J.* 24, 2 (1981), 167–172.