Fast Approximating Triangulation of Large Scattered Datasets*

Henrik Weimer   Joe Warren
Rice University
Computer Science Department
Houston, TX 77005-1892

September, 1997

Abstract

This report describes algorithms and data-structures for the fast construction of three-dimensional triangulations from large sets of scattered data-points. The triangulations have a guaranteed error bound, i.e. all the data-points lie within a pre-specified distance from the triangulation. Three different methods for choosing triangulation vertices are presented, based on interpolation, and L_2 and L_∞-optimization of the error over subsets of the data-points. The main focus of this report will be on devising a simple and fast algorithm for constructing an approximating triangulation of a very large set of points. We propose the use of adapted dynamic data structures and excessive caching of information to speed up the computation and show how the method can be extended to approximate multiple dependent datasets in higher-dimensional approximation problems.

Keywords: Delaunay triangulation, Voronoi diagram, computational geometry, scattered data approximation, multi-dimensional approximation, data-structures, linear programming, optimization.

1 Introduction and Previous Work

This report presents a simple algorithm to iteratively construct a piecewise linear approximation from a given set of points in 3D which satisfies a given error tolerance. The main focus of our presentation will be on making this simple algorithm run fast and efficiently such that very large data sets can be approximated in reasonable time. Applications of such a triangulation method include:

*This work was supported in part under National Science Foundation grant CCR-9500572, the Texas Advanced Technology Program grant 003604-010 and Western Atlas International, Inc.

- digitizing the shape of complex objects for reverse engineering, industrial design and analysis from range images[1]
- construction of a surface model from contour data in medical applications [15]
- modeling of geologic structures such as faults and horizon surfaces from seismic data in exploration geophysics [4]

Naturally such samples are represented by triples (x_i, y_i, z_i) where x_i and y_i identify the sample location and z_i denotes the estimated depth value at this location. For the sake of computational simplicity no two of these triples will coincide in both x_i and y_i, i.e. the samples can be considered coming from a functional surface.

At the end of this document we will introduce extensions of this approximation method to higher-dimensional dependent data-sets.

A fair amount of related work has been done for constructing piecewise linear approximations of sets of scattered points. Many popular methods for constructing a precise linear approximation of a set of scattered points first build the full triangulation of the whole set and then aim at reducing the complexity of this large complex.

Turk [21] introduced a method to derive several levels of detail from a polygonal model description by inserting new vertices into the original model and then removing the old vertices in a topology preserving manner.

Schroeder et al. [19] developed an algorithm to decimate the number of vertices in a triangle mesh. After vertices have been deleted from the original mesh the resulting holes are patched in a local re-triangulation process.

An algorithm which produces a triangulation to approximate a given set of points in 3D is described by Hoppe, DeRose et al. [11]. They define an error-function $f : D \to \mathbb{R}$ where D is an estimation of the shape of the objects and f is an approximation of the geometric distance of the approximation f from the real object. The approximation of the object can then be constructed form the isosurface $f = 0$. In a second step a discrete energy minimization which captures tight geometric fit and sparse representation can then be used to locally optimize the resulting triangulation [12].

Margaliot and Gotsman [16] suggest a similar approach. They approximate a smooth surface from scattered samples by constructing the Delaunay triangulation and then optimizing it to achieve better precision.

Hamann suggests in [9] a method which iteratively removes triangles in nearly planar surface regions of an object. The rationale behind his approach is that flat regions of an object can be represented well by large triangles. A curvature estimate of the triangulated object is suggested to identify such regions.

Klein and Liebich [13] use the Hausdorff distance between the original mesh and the resulting simplification to pick vertices of a polygonal object for elimination and re-triangulate locally.

An algorithm to create a sequence of continuous resolution representations of a complex polygonal object by iteratively collapsing edges of the finer mesh into a single vertex has been devised by Hoppe [10].

A very interesting approach for computing the overall shape of an object has been introduced by Edelsbrunner and Mücke [7]. They define the three dimensional α-shape for some given set of points and precision α as roughly the shape remaining in space when a ball of radius α is rolled over the cloud of points such that the ball only touches but never enters the data points.

All of the algorithms mentioned above were designed to handle small or moderately sized problem sets. We are focusing on handling data sets with several hundred thousand or a few million points and can not afford to run expensive approximation methods. The main focus of this report will thus be on devising a simple and fast algorithm for constructing an approximating triangulation of a very large set of points. We propose the use of adapted dynamic data structures and excessive caching of previously computed information to speed up the computation.

1.1 The Problem

The task is, given a set $S = \{(x_i, y_i, z_i), i = 1, \cdots, n\}$ of scattered points sampled from a surface and a tolerance $\epsilon > 0$, find a triangulation T such that for all $i = 1, \cdots, n$, point (x_i, y_i, z_i) lies within ϵ of the triangulation T. Let $d_T : S \to \mathbb{R}^+$ be a distance measure representing the distance of a point $p \in S$ from the triangulation T, then we can state the problem more formally as

$$d_T(x_i, y_i, z_i) \leq \epsilon \text{ for all } i = 1, \cdots, n. \tag{1}$$

Thus triangulation T will be an approximation of the sampled surface with the guaranteed error bound ϵ with respect to the error measure d_T. We would desire the approximation to be concise, i.e. it should have a small number of vertices and simplices. An example is given in figure 1: a set of 1 Million random samples from a parabolic surface can be approximated by a triangulation with 246 vertices and 431 triangles having all of the samples within 1% tolerance in 22:59 minutes on a 75MHz R8000-based workstation.

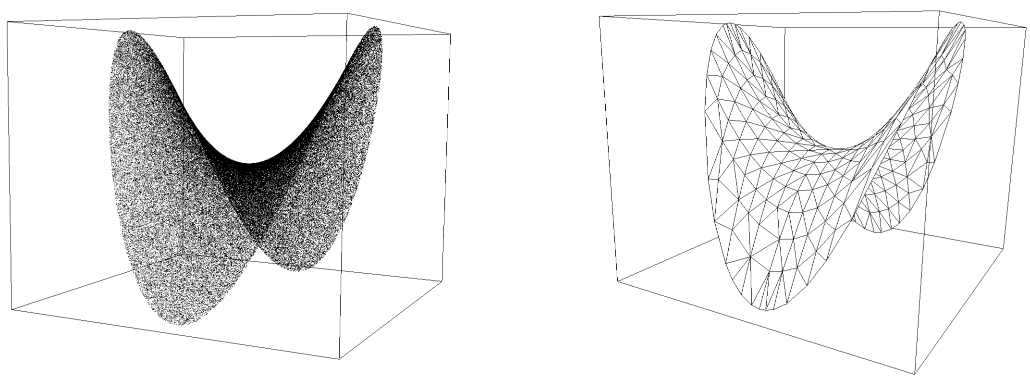


Figure 1: *We want to approximate very large sets of points by a concise triangulation.*

Several issues need to be addressed here. First, it is not clear which distance measure should be satisfied. The simple vertical distance is a common choice for d_T as it is compatible with many of the available methods for sampling data, e.g. all seismic imaging methods and range scanners follow this model. Furthermore vertical distance is extremely easy to compute. The greatest disadvantage of this choice is that we have to restrict the topological type of the approximated objects to be disks.

The vertical distance of the point (x_i, y_i, z_i) from the triangulation T can be expressed as follows. Let $T(x, y)$ be the depth value of triangulation T at the location (x, y), then $d_T(x, y, z) = |T(x, y) - z|$ represents the vertical distance measure between a point (x, y, z) and the triangulation T.

A second point of concern is the choice of triangulation T. The Delaunay triangulation [3] in the parameter plane is particularly interesting because the resulting piecewise linear interpolating surface has been shown by Rippa [18] to have minimal roughness among all piecewise linear interpolating surfaces for a given set of points. Furthermore Delaunay triangulations have many nice geometric properties (see [6, 17]) and efficient algorithms for their construction have been studied extensively (see [2, 5, 14]).

The Delaunay triangulation of a given set of vertices $V = \{v_i, i = 1, \cdots, n\}$ is best described using the Voronoi diagram for V [6]. The Voronoi diagram for V is the collection of all Voronoi regions $V(i)$. The Voronoi region $V(i)$ is the set of all points closer to v_i than to any other point of V. The Delaunay triangulation for V is defined as the straight line dual of the Voronoi diagram for V.

A triangulation T is a planar subdivision in which all bounded regions are triangles. Equation (1) implies that for any data point (x_i, y_i, z_i) there exists a triangle $t_j \in T$ such that $d_{t_j}(x_i, y_i, z_i) \leq \epsilon$ is true where d_{t_j} denotes the restriction of the distance measure d_T to the part of the plane covered by the triangle t_j. Thus triangle $t_j \in T$ is responsible for approximating a subset $S_j \subset S$ containing all points $p \in S$ which lie inside t_j in the projection into the parameter plane. We say the points in S_j are **covered** by triangle t_j, see figure 2. There should be at least three non-collinear data-points in such a set S_j as otherwise the orientation of the triangle t_j will be under-determined.

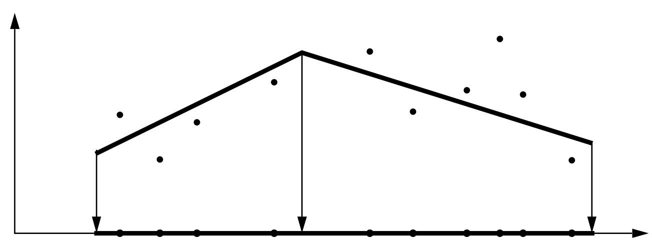


Figure 2: *A sample is* **covered** *by a triangle if it lies inside the triangle in the projection into the xy-plane.*

This report discusses three different approaches to determine the vertices of a triangle t_j given the set of points S_j covered by t_j. The first approach uses interpolation of data-points at the vertices of t_j. The second and third approaches use L_2 and L_∞-minimization of all the distances of the points in S_j from t_j respectively.

2 Constructing an Approximating Triangulation

The proposed algorithm for constructing the approximating triangulation T for a set S of scattered points in 3D and a tolerance ϵ is extremely simple and can be outlined as follows: starting off with a simple and coarse initial triangulation, the data-point with maximal error from the approximation is found. Then this data-point with maximal error is added as a new vertex into the triangulation and the resulting change in the quality of the approximation is assessed. This iterative refinement process continues until the maximal error at any data-point falls below the given tolerance ϵ. The algorithm is summarized in figure 3.

1. compute a coarse initial triangulation T_0 which covers all of S in the projection to the parameter plane
2. find a data-point $\hat{p} \in S$ with maximal distance from T_0, let $i = 0$
3. while $d_{T_i}(\hat{p}) > \epsilon$
   (a) refine T_i with respect to $\hat{p}$, call the resulting triangulation T_{i+1}
   (b) find a data-point $\hat{p} \in S$ with maximal distance from T_{i+1}, let $i = i + 1$

Figure 3: *The Iterative Refinement Algorithm.*

The first iterations of the algorithm are depicted on an example in figure 4. For this example we chose the initial triangulation to include only the first and last data-point and the vertices with maximal deviation from the current approximation are shaded gray. After two iterations the approximation has all samples within the desired tolerance and the final approximation will contain four vertices.

The suggested method is extremely simple. Constructing the Delaunay triangulation for all of S using the algorithm in figure 3 has time complexity $O(|S|^2)$ compared to $O(|S| \log |S|)$ for the optimal algorithm [17]. But if the tolerance ϵ is chosen to be non-zero and the set of samples S is dense, then the algorithm will in general require a number of iterations very much smaller than $|S|$. The remainder of this report will focus on efficient implementations of the presented algorithm such that the time complexity is dominated by the number of vertices in the resulting triangulation T and thus improving the expected running time of the algorithm to be much better than $O(|S| \log |S|)$.

The following sections will describe the steps of the above algorithm in more detail. Section 2.1 presents a simple method for finding the initial triangulation T_0. The following section 2.2 discusses efficient methods for finding the data-point $\hat{p} \in S$ with maximal distance from T_i. Finally section 2.3 suggests approaches for updating the triangulation T_i with respect to the new data-point point $\hat{p}$. The result of this update will be a hopefully more accurate approximation T_{i+1}.

2.1 The Initial Triangulation

Step 1 in the algorithm in figure 3 requires finding a coarse initial triangulation T_0 covering all of S. This triangulation should be easy to compute to start out in the algorithm.

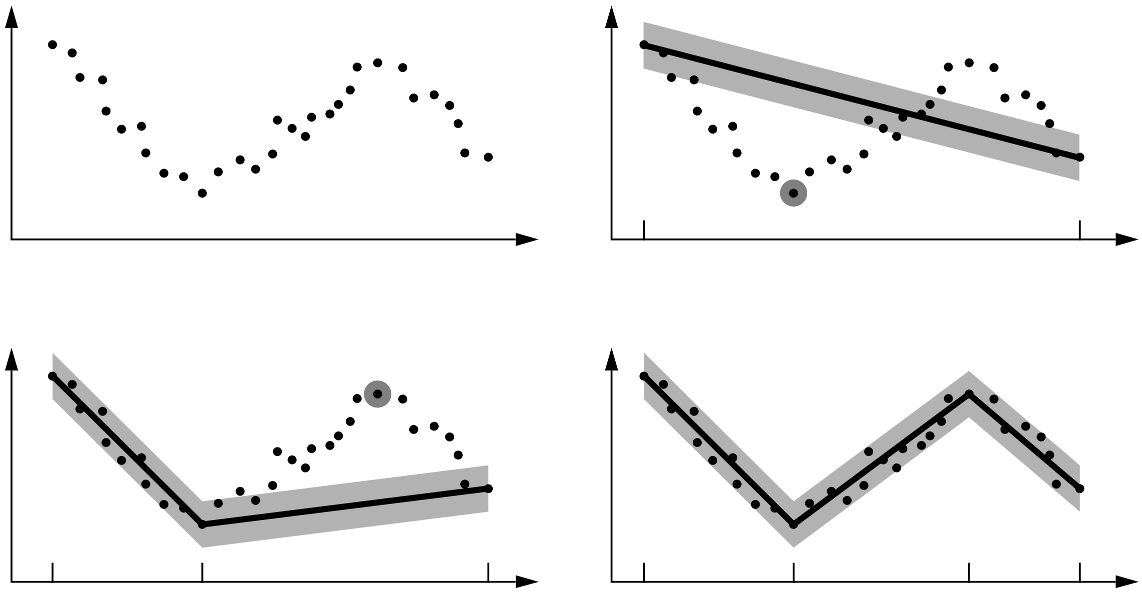


Figure 4: *An example of the first steps of the iterative refinement algorithm.*

T_0 can be determined by finding the bounding box of S and cutting it in half by two triangles parallel to the parameter-plane. The corners of this cutting plane are offset somewhat to the outside of the bounding box such that all the projections of points in S are known to lie properly inside the boundaries of these triangles. Figure 5 shows this construction.

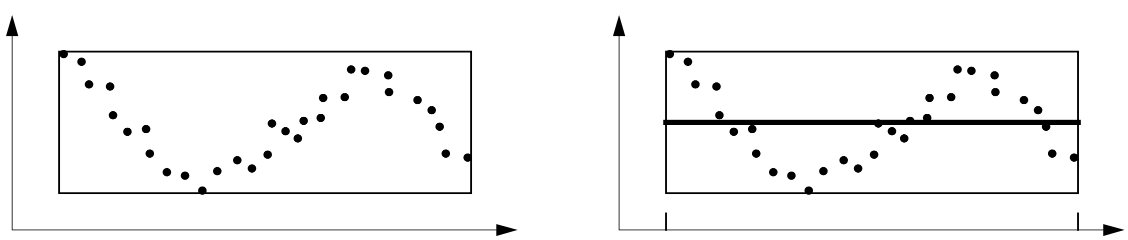


Figure 5: *Initial Triangulation.*

Alternatively an arbitrary set of vertices can be specified to define the knots of the initial triangulation.

2.2 Finding the Maximal Error Point

The refinement step 3a of the algorithm in figure 3 makes use of a data-point $\hat{p}$ in S with maximal distance from the current approximating triangulation T_i. A naive approach for finding this point would simply evaluate the error measure d_{T_i} for every point in S and finding a maximal value among these which requires $O(|S|)$ time in every iteration.

This approach can be vastly outperformed by exploiting the observation that adding a new vertex to the approximation only entails a local change of the triangulation. As pointed out by Bowyer [2], adding a vertex to a Delaunay triangulation requires removing all triangles which contain the new vertex in their circumcircle. This limits the extent of change in the approximation to the data-points covered by these deleted triangles.

A heap [8] containing all points $p \in S$ ordered by their error $d_{T_i}(p)$ can be used to keep track of these changes from iteration to iteration. Upon insertion of a new vertex into the triangulation, the heap needs to be updated for those points in S which were covered by the deleted triangles. This will lead to a considerable speedup as typically the number of points covered by a single triangle will be small and also the number of triangles affected by inserting one vertex will be small. Let n_i denote the number of points which were covered by triangles erased in iteration i. Then this approach has time-complexity $O(n_i \log |S|)$ for the update of the heap in iteration i of the iterative refinement algorithm.

Further improvement can be achieved. Observe that the point $\hat{p} \in S$ with maximal error is covered by some triangle $\hat{t}$ in the current triangulation T_i. Furthermore the point $\hat{p}$ has also maximal error among all points covered by triangle $\hat{t}$, see figure 6. Thus it is sufficient to only consider one data-point per triangle for this heap of errors. Thus for every triangle t in the approximation T_i only one data-point with maximal error covered by the triangle t needs to be stored in the heap. When updating the triangulation T_i by inserting a new vertex, heap entries coming from erased triangles are deleted and one single entry for every newly created triangle is added to the heap with the maximal error of any point covered by the new triangle. Thus the size of the heap in iteration i is the number of simplices in triangulation T_i.

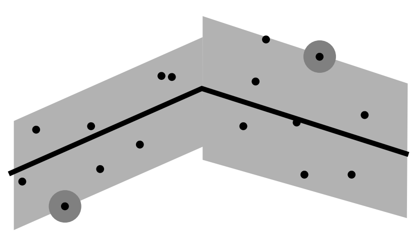


Figure 6: *It is sufficient to keep track of one sample with maximal error per triangle in the approximation.*

Let $n_{del}^{\hat{p}}$ denote the set of triangles which are removed from T_i upon insertion of vertex $\hat{p}$, and let $n_{ins}^{\hat{p}}$ denote the number of triangles which are created in step i. Thus the time-complexity for one update of the heap is $O(n_{del}^{\hat{p}} \log |T_i| + n_{ins}^{\hat{p}} \log |T_{i+1}|)$ which is generally significantly smaller than $O(|S|)$ and $O(n_i \log |S|)$.

2.3 Updating the Triangulation

To update the triangulation T_i with respect to vertex $\hat{p}$ in step 3a of the iterative refinement algorithm in figure 3, the corresponding Delaunay triangulation in the parameter plane has to be modified. This requires erasing all simplices which contain the new vertex in their circumcircle and re-triangulating in this region. The removals expose an un-triangulated region in T_i with vertices on its boundary which are all visible from the new vertex $\hat{p}$, i.e. a straight line from $\hat{p}$ to any vertex on this boundary does not intersect any of the remaining triangulation edges. In the literature this area is called a star-shaped region [17].

This star-shaped region can be found in an outward search in the current triangulation starting at the simplex covering $\hat{p}$. To facilitate this search a reference to the covering triangle can be stored with every point in S. Upon insertion of a new vertex for data-point $\hat{p}$ into the triangulation, the triangle $t_{\hat{p}}$ which covers $\hat{p}$ can be found using this reference. Trivially $t_{\hat{p}}$ has $\hat{p}$ inside its circumcircle as $\hat{p}$ lies in the interior of the triangle. Thus triangle $t_{\hat{p}}$ has to be erased from the triangulation. Starting at $t_{\hat{p}}$ a search outward in the triangulation for other triangles with $\hat{p}$ in their circumcircle can be performed. In this outward search only triangles which are vertex-adjacent to some triangle that has the new vertex in its circumcircle need to be considered [2]. Figure 7 contains an example: The circumcircles containing the new vertex are drawn in solid lines, the triangles which need to be removed are shaded in gray and can be found in an outward search starting at the triangle containing the new vertex. A stack can then be used to keep track of adjacent triangles which still need to be considered in the search. After finding the set of triangles to be erased, the Delaunay triangulation can be modified locally by deleting these triangles and adding edges from the inserted vertex $\hat{p}$ to any vertex on the boundary of the star-shape.

To update the references to covering triangles which are associated with data points the vertices on the boundary of the star-shape can then be sorted radially with respect to the newly added vertex and data-points previously covered by erased triangles are reassigned to the covering triangles efficiently using a binary search in the radially sorted boundary of the star-shape.

After all points have been reassigned to a covering triangle, one single pass over the newly added triangles is necessary to find the data-point with maximal error for each of them. This data-point together with its error and a pointer to the covering triangle is then inserted into the

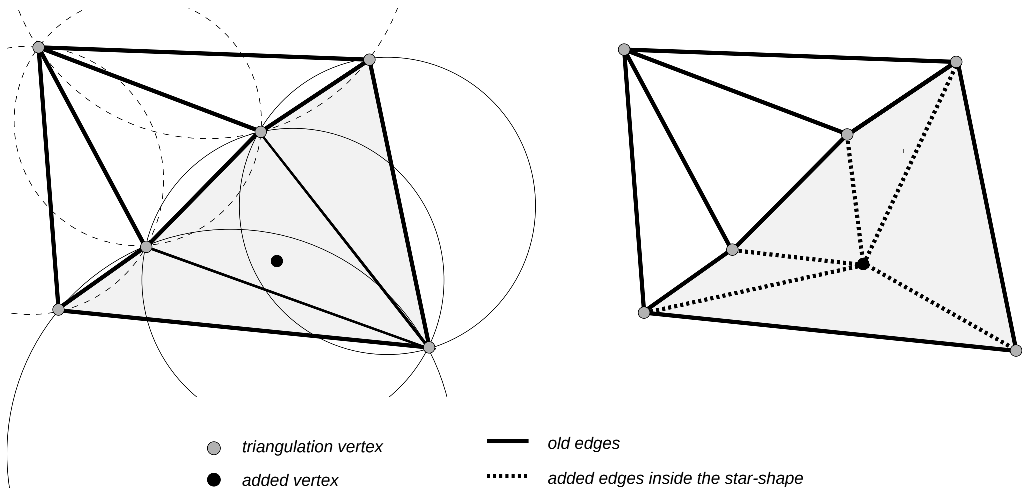


Figure 7: *Identifying and re-triangulating the star-shaped area of triangles on insertion of a new vertex.*

heap of maximal errors. After the update of the heap is completed, the point with maximal error is stored at the root of the heap and can thus be found easily in step 3b of the iterative refinement algorithm in figure 3.

3 Interpolation vs. Approximation at the Vertices

The previous section described how the triangulation in the parameter plane can be refined once the vertex to be added next has been determined. It is not immediately clear, what the effect on the three-dimensional approximating triangulation should be: The connectivity of the approximating triangulation will be the same as in the parameter plane, but how should the z-coordinate for the newly added vertex be chosen and what will be the actual position of the triangles in 3D? The following sections will focus on answering these questions.

3.1 Interpolation

The simplest solution for finding this z-value for a new vertex is to simply take the z-value of the data-point which led to the insertion of this vertex. This approach will lead to an approximating triangulation which interpolates data-points at the triangulation vertices and approximates only on the interior of triangles.

Choosing interpolation at the triangulation vertices is simple and fast. No additional work needs to be done to update the approximation once the triangulation in the parameter plane has been determined. Also the approximating triangulation is well defined as all data-points are coming from a functional surface by assumption, i.e. no two data-points have common x and y component.

The disadvantage of choosing interpolation at the vertices in the triangulation is that convergence may be far less than optimal. The problem may be, that the given set of data points could be approximated within tolerance with a single simplex, but additional refinement steps may be necessary just because interpolation at the vertices was used. An example of this behavior in two dimensions is given in figure 8.

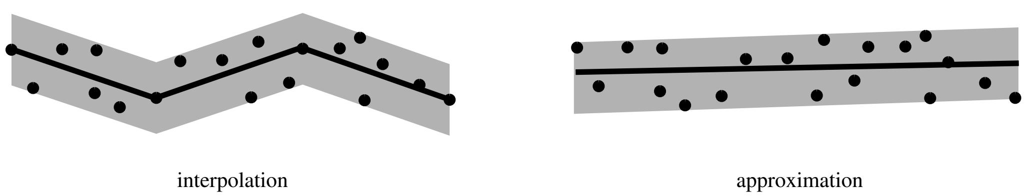


Figure 8: *Interpolation may require more simplices than approximation.*

3.2 Approximation

Approximation at the vertices can be used with the hope of achieving faster convergence of the iterative refinement algorithm to a pre-specified tolerance. If the vertices are no longer interpolated data-points, the z-value at a triangulation vertex obviously has to be found by some other means.

Consider insertion of vertex v into the triangulation. As described in section 2.3 this means inserting a star-shape of triangles into the triangulation. Call this set of new triangles T_{new}.

Optimization of the distance measure d_T for all the data-points covered by triangles adjacent to the added triangulation vertex can be used to determine approximating planes which are optimal under some yet to be determined criterion. Following the idea presented in [22] the z-value at vertices can be chosen to the average of z-values of adjacent approximating planes. After the data-points have been reassigned to their covering triangle in T_{new} according to the radial angle of their projection into the parameter plane, the z-value for vertices of triangles in T_{new} can be found using the algorithm in figure 9.

Figure 10 contains an example in two dimensions: The z-values at vertices are found to be the average of the z-values of adjacent L_2-optimal approximating planes. The following sections will focus on possible choices for the approximating planes computed in step 1 of this algorithm.

The construction in step 2b in the algorithm in figure 9 computes the z-value at a triangulation

1. for every triangle $t \in T_{new}$ compute an approximating plane for the set $\tilde{S} \subset S$ of data-points covered by t which minimizes the L_k-norm of d_t on $\tilde{S}$ (for $k = 2$ or $k = \infty$).
2. let
$$V = \bigcup_{t \in T_{new}} \{\text{vertices of } t\} \ .$$
3. for each $\hat{v} \in V$
   (a) let $T_{\hat{v}}$ be the set of triangles adjacent to $\hat{v}$ (including but not limited to those in T_{new})
   (b) compute the new z-component for vertex $\hat{v}$ to
$$\hat{v}_z = \frac{1}{|T_{\hat{v}}|} \sum_{t \in T_{\hat{v}}} t(\hat{v}) \ .$$

Figure 9: *Algorithm for updating the z-values.*

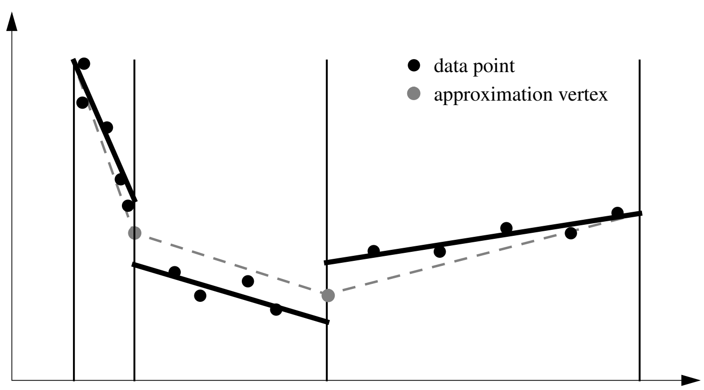


Figure 10: *Computing z-values at vertices from approximating planes.*

vertex as the average of the z-values of all L_k-optimal planes for triangles adjacent to the vertex. Note that inserting a vertex into the triangulation leads to updates of z-values of all vertices of triangles in the star-shape, including the boundary. This implies that triangles vertex-adjacent to star-shape triangles will also change and thus the heap of errors needs to be updated for points covered by these triangles, too.

One subtle problem comes up when relaxing the triangulation vertices to not interpolate data-points. It now is possible that a data-point has maximal error in two different iterations (e.g. in figure 12). The first time the data-point $\hat{p}$ has maximal error, the iterative refinement algorithm picks $\hat{p}$ as a new site for a triangulation vertex. If the same point $\hat{p}$ has maximal error a second time it is possible that the iterative refinement algorithm runs in an infinite loop and thus does not converge because adding a vertex to the triangulation a second time does not change the approximation at all. This problem can be circumvented by forcing the approximation to interpolate any data-point

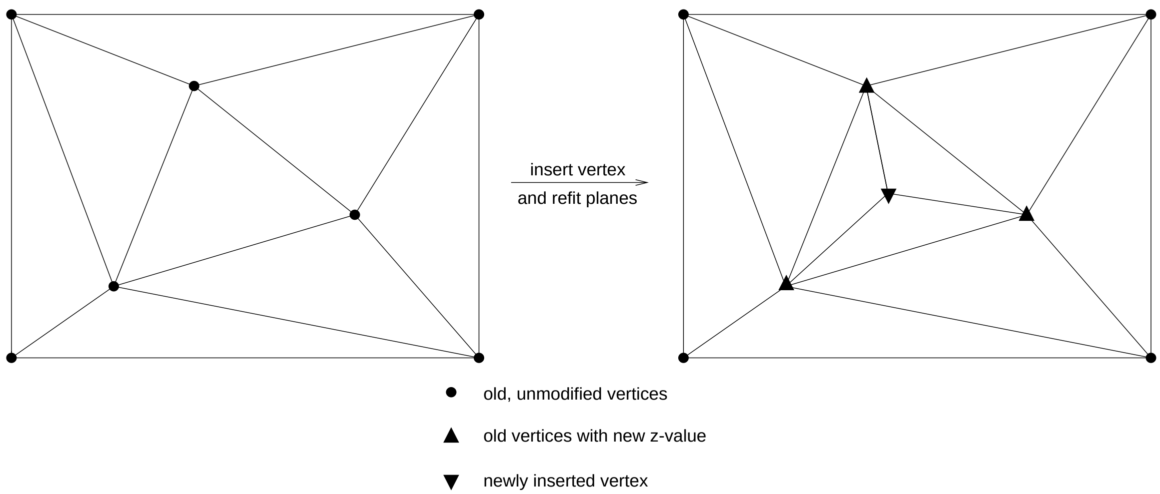


Figure 11: *Area in the triangulation affected by insertion of a new vertex.*

which is found to have maximal error more than once.

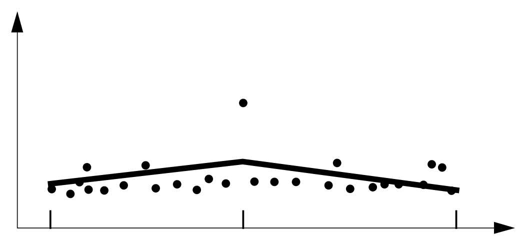


Figure 12: *If approximation is used a data-point can have maximal error even though it is a knot in the triangulation.*

3.2.1 L_2-Approximation

Step 1 in the algorithm in figure 9 requires finding an approximating plane for a given set of points. One possible choice for such an approximation of a set of points is the L_2-optimal plane which minimizes the sum of the squares of the distances of all data-points from the plane. The rationale behind minimizing the L_2-norm of a set of data-points over triangles is to minimize the average error of points covered by the approximation triangle. Let $P = \{(x_i, y_i, z_i), i = 1, \cdots, n_P\}$ be a set of points in space. Then the L_2-optimal plane F for P is a mapping $F : \mathbb{R}^2 \to \mathbb{R}^3$ which minimizes

$$\min_F \sum_{i=1}^{n_P} |F(x_i, y_i) - z_i|^2 \tag{2}$$

The plane F can be represented as $F(x, y, z) = ax + by + c$. Differentiation of (2) with respect to a, b and c leads to a system of three linear equations in the three unknowns which can be solved symbolically. The resulting expressions for the unknowns a, b and c contain sums of quadratic expressions in the x_i, y_i and z_i. Thus finding the L_2-optimal plane for a given set P takes $O(|P|)$ time.

3.2.2 L_∞-Approximation

L_2-optimization over the set of data-points covered by one triangle was used to minimize the overall error for the triangle. Conversely an L_∞-minimization over the sets of covered data-points can be used with the hope that this will lead to faster convergence of the approximation. The rationale here is that the maximal error of any data-point should be minimized to achieve fast convergence.

In this framework a linear programming problem is set up to minimize the maximal distance of any point in $P = \{(x_i, y_i, z_i), i = 1, \cdots, n\}$ from the plane $F(x, y) = ax + by + c$, i.e. we want to find the minimum

$$\min_{a,b,c} \max_i |F(x_i, y_i) - z_i| \ . \tag{3}$$

This problem can be restated by introducing an objective variable δ defined by

$$|F(x_i, y_i) - z_i| \leq \delta \qquad \text{for all} \qquad i = 1, \cdots, n. \tag{4}$$

The absolute values can then be eliminated by having two constraints per data-point $(x_i, y_i, z_i) \in P$

$$\begin{aligned} F(x_i, y_i) - z_i &\geq -\delta \\ F(x_i, y_i) - z_i &\leq \delta \ . \end{aligned}$$

Replacing the definition of F and rewriting a, b and c as $a = a_+ - a_-$,$b = b_+ - b_-$ and $c = c_+ - c_-$ allows us then to use any linear programming solver (e.g. see [20]) to find a L_∞-optimal solution for these $2n$ linear inequalities in 7 variables, $n = |P|$. The result will be the coefficients for an approximating plane which has minimal maximal distance for any point in P with respect to the distance measure d_T.

4 Extension to Higher Dimensional Approximation

The algorithms presented so far can be used to construct an approximating triangulation T from a set of points $S \subset \mathbb{R}^3$. The final triangulation T will satisfy a preset tolerance criterion according to (1) where $d_T : S \to \mathbb{R}^+$ is some error measure. In many applications the data-points S are scattered samples from some real object and the resulting triangulation T is used as a model for this object which is then used in later processing. For reasons motivated in section 1.1, d_T was chosen as the vertical distance between the triangulation T and the data-point, i.e. $d_T(x, y, z) = |T(x, y) - z|$.

An extension to the stated approximation problem would be to allow data-points from an arbitrary dimensional object $S \subset \mathbb{R}^d,\ d \geq 3$ and to construct a triangulation T which approximates this higher dimensional object within a given error bound ϵ. Hence the problem statement (1) can be reformulated to

$$d_T(p) \leq \epsilon \quad \text{for all } p \in S \tag{5}$$

where $d_T : S \to \mathbb{R}$ is again a suitable error measure which is now defined on a subset of $\mathbb{R}^d$. Such an approximation can be used to accurately represent higher dimensional data-sets and they arise in many modeling problems, e.g. in oil reservoir modeling where the first two dimensions specify a location and the following dimensions represent things like depth, speed of sound, porosity etc. For subsequent finite element analysis it is now essential to construct a mesh that represents all of the dependent values to the desired precision and has a common set of knots for the approximation.

The methods presented earlier can be extended to handle such a multi-dimensional approximation problem by replacing the error measure d_T which was based on vertical distance by an error measure for the higher dimensional problem. The iterative refinement algorithm in figure 3 now has to consider the point with maximal error with respect to this new error measure.

If we assume that the samples come from a functional two-dimensional object in $\mathbb{R}^d$ then the error measure d_T can be redefined based on an arbitrary vector metric for R^d, i.e.

$$d_T(x_1, x_2, \cdots, x_d) = \|(x_3, x_4, \cdots, x_d) - T(x_1, x_2)\|_* \tag{6}$$

where $\|\cdot\|_*$ denotes any error metric assessing the quality of the approximation. One very reasonable choice for $\|\cdot\|_*$ is the maximums norm on R^{d-2},i.e. $\|(t_1, t_2, \cdots, t_{d-2})\|_\infty = \max_i t_i$.

Equation (6) implies together this choice for $d_T$that the resulting approximation T satisfies the error constraint in every coordinate dimension.

The methods to enhance the run time efficiency of the iterative refinement algorithm based on the heap of errors per simplex in the approximation (section 2.2) and the binning of data-points to the covering triangles (section 2.3) easily extend to this multi-dimensional setting.

5 Results

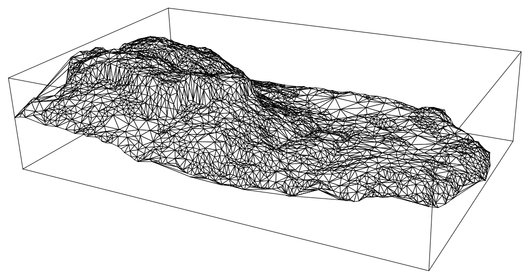
The three approaches to construct approximating triangulations were implemented and tested on three-dimensional sets of scattered points sampled from surfaces. Figure 13 contains as an example approximations created from a small set of 7644 scattered data-points which have been sampled off the base of a salt surface in the Gulf of Mexico. All three approximations contain any of the data-points within 1% vertical tolerance. Figure 13a contains the approximation constructed using interpolation at the triangulation vertices, figure 13b has been constructed using L_2-optimization and 13c using L_∞-optimization at the vertices. The approximation algorithms ran on these problems on a 75MHz R8000 based workstation with 256MB of memory for 46.76 seconds, 1 minute 26.46 seconds and 10 minutes 15.17 seconds respectively.

As expected, the method using interpolation at the triangulation vertices runs fastest. The approximating methods require a very large amount of work constructing the L_2 and L_∞-optimal approximating planes and thus run much slower.

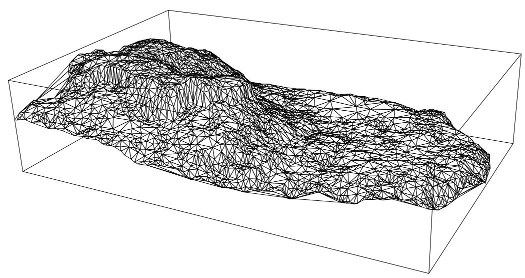
But running time is not the only concern. As the resulting approximating triangulations will be used as input to subsequent algorithms such as numerical simulation, the number of simplices in the approximation is of critical importance. Reducing the size of the triangulation by a considerable amount would be worth investing more time in finding this simpler approximation.

As the iterative approximation algorithm inserts one vertex at a time, the number of simplices in the triangulation is proportional to the number of iterations the algorithm runs. Thus it would be desirable to achieve a given error bound using a smaller number of vertices.

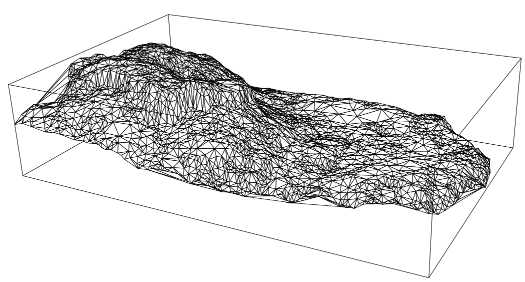
Experience on a large number of data sets with a wide range of sizes seems to indicate that L_2 and L_∞-approximation do not lead to significantly smaller approximations. Thus the interpolation method seems overall preferable as the approximations can be found in orders of magnitude less time.



a.: interpolation: 3471 vertices, 6805 triangles



b.: L_2-optimization: 3301 vertices, 6462 triangles



c.: L_∞-optimization: 3167 vertices, 6210 triangles

Figure 13: *Approximation of a geophysical dataset. The original set has 7644 scattered data-points. The three surfaces have all the data-points within 1% vertical tolerance.*

Figures 14 and 15 compare the three paradigms for handling triangulation vertices: interpolation, L_2 and L_∞-approximation. These graphs are typical for all datasets that we ran the algorithms on. The graphs show the number of iterations along the horizontal axis and the maximal relative vertical error at every step along the vertical axis. Thus the curves shown here demonstrate the rate of convergence against a given error bound.

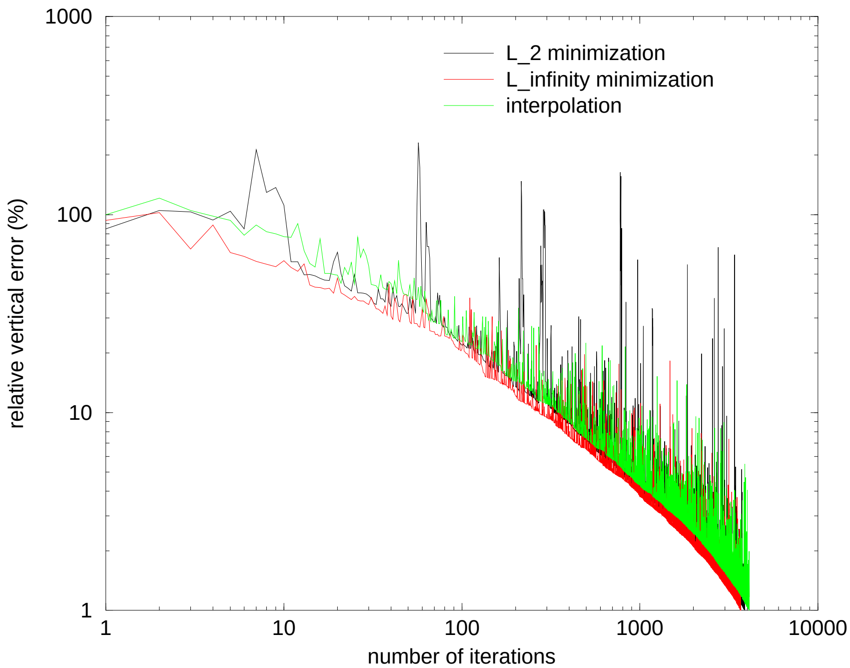


Figure 14: *Comparison of the three paradigms (top dataset, 7640 points)*

We observe that for the approximating methods we get a much greater variability in the convergence. The graphs for L_2 and L_∞-approximation show glitches of very high vertical error, but these glitches are typically accommodated within one or two iterations and the curve comes back to the old relative error. Especially for the L_2-approximating method the glitches are significant.

This variability can be explained by the fact, that in many cases the approximating plane for one triangle is not a good solution for the adjacent triangles. Averaging of the z-values of all planes from adjacent triangles at the triangulation vertices can thus lead to poor approximation. Hence if one of the planes has a slope far off from the sampled structure, the z-value at vertices can be strongly deviating from the z-value of points sampled close to this vertex (but which are covered by different triangles). The approximating triangulation will then reflect this peak in the z-coordinate and have a very high vertical error. This problem is typically resolved within a very small number of iterations, but it leads to slowdown in the convergence towards the given tolerance.

The interpolating method does not exhibit these glitches of temporarily high error as all vertices are completely determined by the data itself. If the sampled object is sufficiently smooth inserting interpolated vertices does not increase the deviation of the approximation from the samples.

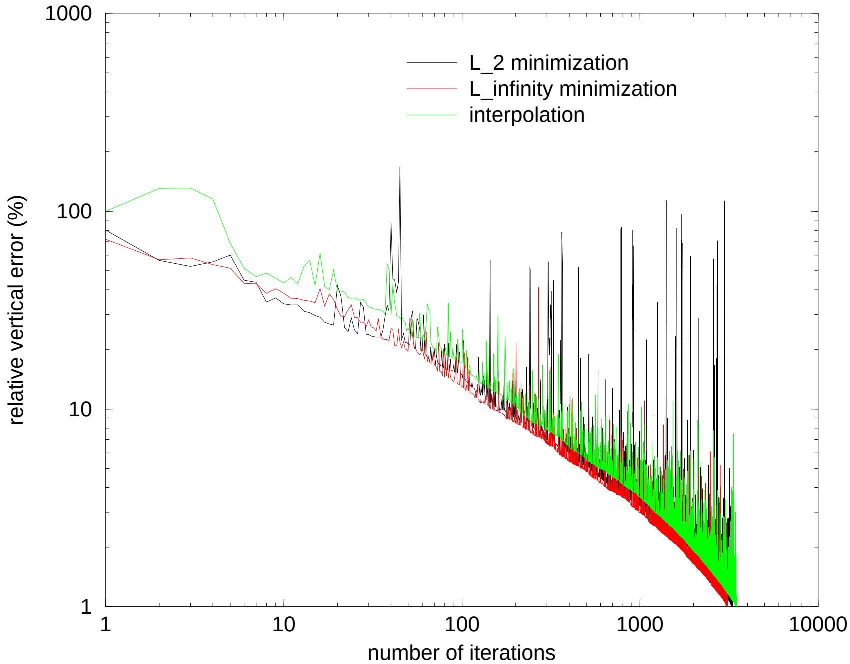


Figure 15: *Comparison of the three paradigms (base dataset, 7644 points)*

A second important observation is that for both graphs the three curves have the same overall rate of convergence. L_∞ converges the fastest, L_2 converges slightly slower and interpolation converges the slowest. But the differences are miniscule except for the first 10 or so iterations where the relative error is anyway still very high. For the first few iterations, the approximating techniques have a significantly better relative error, but during this time their overhead is also maximal. During the first few iterations the number of points covered by one triangle is still very large (in the first iteration about $\frac{1}{2}$ the number of data-points) and thus it is very expensive to construct approximating planes at this stage. Also the relative error is still so high, that an approximation with that few vertices is unacceptable because of exceeding tolerance.

The observation that the general rate of convergence is almost the same for the three methods is significant. The much higher computation effort to find the approximating triangulations would be acceptable if the resulting triangulations were significantly smaller, but the experiments suggest that a comparable number of simplices is required in all three approaches and thus the L_2 and L_∞-approximating algorithms will lead to triangulations with comparable size for a given tolerance. Hence using the algorithm with interpolation at the triangulation vertices is warranted as it is by orders of magnitude computationally simpler and the results are almost as good as with the approximating approaches.

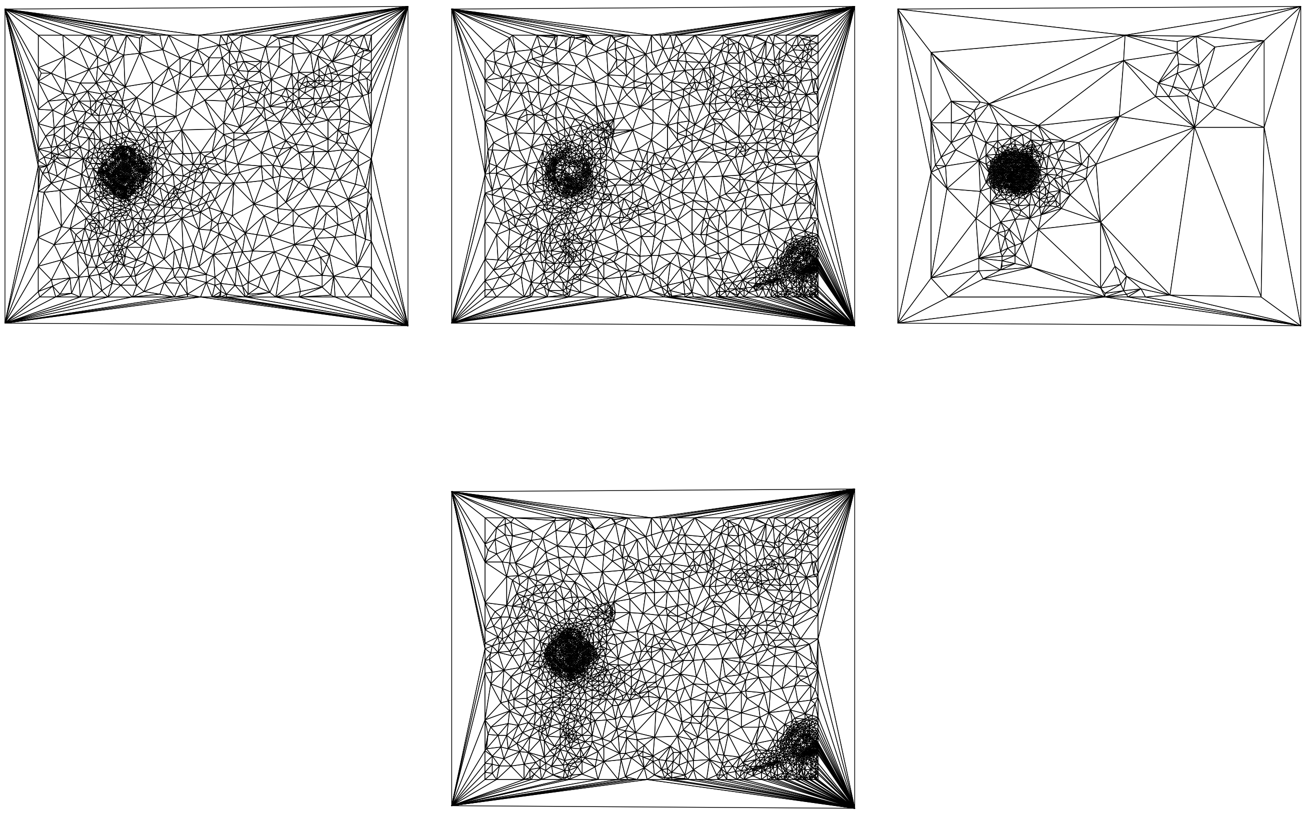


Figure 16: *An example of multi-dimensional approximation in 5D.*

Figure 16 depicts an example for the multi-dimensional approximation method. The input data-set contained 67221 samples of the form x, y, v_1, v_2, v_3 where x and y specify a location and v_1, v_2, and v_3 are dependent, functional data. The first row of figures depicts the triangulations which are constructed for each of the dimensions independently using 1% vertical tolerance. They have 1441, 1938 and 987 vertices respectively. The second row shows the multi-variate approximation using the infinity norm also with 1% vertical tolerance. This approximation has 2329 vertices which is far less than the sum of the sizes of the individual approximations. The multi-dimensional approximation accurately represents the significant structures for all three dependent input data sets on one common set of triangulation knots and can thus be used to generate a mesh representing the three dependent values simultaneously.

6 Conclusion and Future Work

This report suggested a very simple iterative algorithm for constructing an approximating triangulation of a set of scattered points which satisfies a preset tolerance criterion. The efficiency of the simple algorithm has been significantly improved by the use of dynamic data-structures and excessive caching such that even large data-sets with several million samples can be approximated within reasonable time.

The approximation is constructed without ever building the Delaunay triangulation of the full data-set. Several approaches for choosing the actual triangulation vertices have been presented based on interpolation, L_2 and L_∞-optimization over subsets of the data-points. The experimental results suggest that all three methods will lead to comparably large approximations but the algorithm based on interpolation at the vertices finds an approximation the fastest.

Future work should address the question why the algorithms with approximation at the triangulation vertices do not exhibit significantly faster convergence than the interpolation method. There is hope that finding the reason for slower convergence will hint to algorithms with better convergence rates.

Similarly it seems worth investigating how the restriction on the input data being two-dimensional, functional data can be alleviated. Although the simple iterative refinement algorithm extends to higher dimensions in a straight forward manner this extension is non-trivial. The methods for improving the runtime efficiency rely heavily on the parameter domain being two-dimensional and further work is needed to adapt these ideas to higher dimensions.

Finally we would want to address the question whether triangulations other than the Delaunay triangulation can be used to generate smaller approximations. As the resulting approximation is used as an input to a mesh generator the triangulation has to have certain guarantees on the aspect ratio of the simplices, or numerical instability problems will arise inevitably.

Acknowledgments

The authors would like to thank the people at Western Geophysical for their great support and many helpful discussions. Special thanks goes to Uwe Albertin, Jane Troutner and Wendell Wiggins. The data used in the evaluation of our algorithms is courtesy of Western Geophysical and the Society of Exploration Geophysicists.

References

[1] Paul J. Besl. Acitve, optical range imaging sensors. *Machine Vision and Applications*, 1(2):127–152, 1988.

[2] A. Bowyer. Computing dirichlet tesselations. *The Computer Journal*, 24(2):162–166, 1981.

[3] B. Delaunay. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS, VII Série, Class des Sciences Mathématiques et Naturelles*, pages 793–800, 1934.

[4] Milton B. Dobrin and Carl H. Savit. *Introduction To Geophysical Prospecting*. McGraw-Hill, fourth edition, 1988.

[5] Rex A. Dwyer. A faster divide-and-conquer algorithm for constructing Delaunay triangulations. *Algorithmica*, 2:137–151, 1987.

[6] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[7] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[8] Robert W. Floyd. Algorithm 245: Treesort3. *Communications of the ACM*, 7:701, 1964.

[9] Bernd Hamann. Data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

[10] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[11] Hugues Hoppe, Tony DeRose, Tony Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[12] Hugues Hoppe, Tony DeRose, Tony Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Computer Graphics*, pages 19–25, 1993.

[13] Richard Klein, Gunther Liebich, and W. Straßer. Mesh reduction with error control. In *Proceedings Visualization '96*, pages 311–318. IEEE Computer Society, 1996.

[14] D. T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980.

[15] Bill Lorensen, Ron Kikinis, John Flynn, Arie Kaufman, and Sandy Napel. Surface rendering versus volume rendering in medical imaging: Techniques and applications. In *Proceedings Visualization 1996*, pages 439–440. IEEE Computer Society, 1996.

[16] Michael Margaliot and Craig Gotsman. Piecewise-linear surface approximation from noisy scattered samples. In *Proceedings Visualization 1994*, pages 61–68. IEEE Computer Society, 1994.

[17] Franco P. Preparata and Michael I. Shamos. *Computational Geometry - An Introduction.* Springer-Verlag, 1985.

[18] Samuel Rippa. Minimal roughness property of the Delaunay triangulation. *Computer Aided Geometric Design*, 7:489–497, 1990.

[19] J. William Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, 1992.

[20] Paul R. Thie. *An Introduction to Linear Programming and Game Theory.* John Wiley and Sons, second edition edition, 1988.

[21] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics*, 26(2):55–64, 1992.

[22] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 311–318. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.