Adaptive Thinning for Bivariate Scattered Data

N. Dyn, M. S. Floater, and A. Iske

Abstract: This paper studies adaptive thinning strategies for approximating a large set of scattered data by piecewise linear functions over triangulated subsets. Our strategies depend on both the locations of the data points in the plane, and the values of the sampled function at these points — *adaptive thinning*. All our thinning strategies remove data points one by one, so as to minimize an estimate of the error that results by the removal of a point from the current set of points (this estimate is termed "anticipated error"). The thinning process generates subsets of "most significant" points, such that the piecewise linear interpolants over the Delaunay triangulations of these subsets approximate progressively the function values sampled at the original scattered points, and such that the approximation errors are small relative to the number of points in the subsets. We design various methods for computing the anticipated error at reasonable cost, and compare and test the performance of the methods. It is proved that for data sampled from a convex function, with the strategy of convex triangulation, the actual error is minimized by minimizing the best performing measure of anticipated error. It is also shown that for data sampled from certain quadratic polynomials, adaptive thinning is equivalent to thinning which depends only on the locations of the data points — *non-adaptive thinning*. Based on our numerical tests and comparisons, two practical adaptive thinning algorithms are proposed for thinning large data sets, one which is more accurate and another which is faster.

Key words: thinning, adaptive data reduction, simplification, bivariate scattered data, triangulation, piecewise linear approximation

1. Introduction

When dealing with large sets of scattered bivariate data with corresponding function values, it is often desirable to generate a hierarchy of coarser and coarser representations of the data, for example when fast visualization is required. One such coarse representation is simply the piecewise linear interpolant to some subset of the data over a suitable triangulation. This paper addresses the problem of how to generate a sequence of subsets of "most significant" points, such that the corresponding hierarchy of piecewise linear interpolants over the Delaunay triangulations of these subsets are close to the original data, and such that the error of approximation gradually increases with the reduction in the number of points in the subsets. Our approach to this is to apply what we call an *adaptive thinning algorithm*: the recursive and greedy removal of "least significant" points from the data set, according to some criterion, which attempts to reduce the error incurred by the removal of a point.

The idea of thinning triangulated scattered data is not new, and is more commonly referred to as *decimation* or *simplification* in the literature. Heckbert and Garland [**8**] give an extensive survey of simplification methods both for terrain models (triangulated scattered data in the plane) and free form models (manifold surfaces represented by 3D triangle meshes). For a more recent survey paper, see [**7**].

The literature on simplification methods for triangle meshes is quite large, and among the many ideas there are two which, we believe, are crucial: (1) anticipating realistically, by a reasonable amount of computing, the error which is incurred by the removal of a point from a given set of points, and (2) using a priority queue in order to reduce the computational cost from $O(N^2)$ to $O(N \log N)$.

The thinning algorithm we study here is built on these two ideas. We have designed, tested and compared several methods for anticipating the error which is incurred by the removal of a point from a data set. Our algorithms choose the point to be removed as the one of minimal anticipated error. In our numerical experiments one method for anticipating the error is found to be superior (denoted by AT1). The conclusion of the paper is an efficient, adaptive thinning algorithm for generating multiscale approximations to a function sampled at scattered points. We also propose a second adaptive thinning algorithm (denoted by AT3) which is less accurate but much faster.

Though we have not found papers in which the thinning algorithm AT1 appears, it is close to the simplification algorithm of Soucy and Laurendeau [**12**], designed for free form 3D triangle meshes.

To gain some insight into the nature of our algorithm, it is proved that for data sampled from a convex function, with the strategy of convex triangulation, minimization of our preferred anticipated error yields the same sequence of removed points, as the minimization of the actual error incurred by the removal of a point. This theoretical result together with the observation that Delaunay triangulation is the same as convex triangulation for data sampled from

$$f(x) = A(x_1^2 + x_2^2) + Bx_1 + Cx_2 + D \ , \tag{1.1}$$

leads us to the conclusion that our algorithm can be expected to perform well, as is demonstrated by the numerical examples (in Section 6).

Another theoretical aspect of our algorithm relates adaptive thinning for the quadratic polynomials (1.1) to non-adaptive thinning (thinning that depends only on the locations of the bivariate points, and generates a hierarchy of subsets of the scattered bivariate points, which are close to being well distributed). This result can be a source for new strategies for non-adaptive thinning [**6**].

Similar considerations and results for the simpler case of univariate adaptive thinning are presented in [**3**].

For a given point set, the piecewise linear interpolant depends on the triangulation of the point set. Thus our adaptive thinning algorithms depend on the triangulation strategy. In this paper we use Delaunay triangulations [**9**], which possess many interesting theoretical properties and with which the complexity estimation of our adaptive thinning algorithm is easy. Yet it seems that for reducing the approximation error, a better strategy would be that of "data-dependent triangulations" [**4**]. This will be checked in future work.

The outline of the paper is as follows: In Section 2 we present in details the notion of adaptive thinning, and formulate our thinning algorithm. The two theoretical results on our adaptive thinning algorithm, mentioned above, are investigated in Section 3. Section 4 is concerned with implementation aspects and complexity aspects of our adaptive thinning algorithm. In Section 5 two less costly methods for anticipating the error are presented,

resulting in two additional adaptive thinning algorithms. Numerical experiments with the three adaptive thinning algorithms and one non-adaptive thinning algorithm are presented in Section 6. The performance of the four algorithms on a set of terrain data is compared, and practical conclusions are made.

2. Adaptive Thinning

Suppose $X = \{x_1, \ldots, x_N\}$ is a given finite set of distinct points in $\mathbb{R}^2$ with convex hull $\Omega = [X]$, and that some unknown function f is sampled at these points, giving the values $f(x_1), \ldots, f(x_N)$. One of the central issues in this paper is finding a subset Y of X and a triangulation $\mathcal{T}_Y$ such that the piecewise linear interpolant $L(f, \mathcal{T}_Y)$ is close to the given data in the sense that the error

$$E(\mathcal{T}_Y; X; f) = \max_{x \in X} |L(f, \mathcal{T}_Y)(x) - f(x)| \tag{2.1}$$

is small. By a triangulation of Y we understand a collection of closed triangles $\mathcal{T}_Y = \{T_1, \ldots, T_n\}$ whose vertices comprise Y, and which satisfy the conditions

(i) $T_i \cap T_j$ is either empty, a common vertex or a common edge, for $i \neq j$,
(ii) $\bigcup_{i=1}^{n} T_i = [Y]$.

The piecewise linear interpolant to f over $\mathcal{T}_Y$ is denoted by $L(f, \mathcal{T}_Y)$. It is the function which is continuous over $[Y]$ and linear over each triangle in $\mathcal{T}_Y$. In order for (2.1) to make sense, we assume that $[Y] = \Omega = [X]$. For the sake of simplicity, we ensure this by only considering subsets Y of X for which $X_B \subset Y$, where $X_B = X \cap \partial\Omega$ with $\partial\Omega$ the boundary of Ω. We also define $X_I = X \setminus X_B$.

Ideally, for any given M which is less than N and greater than the number of points in X_B, namely $K = |X_B| < M < N$, we would like to find a subset Y of X of cardinality M, satisfying $X_B \subset Y$, for which the error in (2.1), for a fixed triangulation strategy, is minimal. However it is clearly impractical to search amongst all possible subsets and this motivates the more pragmatic approach of *thinning*.

The idea of thinning is to remove points from X_I one by one in order to reach a subset Y of a certain size. Our criterion for removing a point from the current subset is to minimize the error incurred by the removal, with respect to some measure of error. In general we want to remove a point of 'least' significance. In this sense the thinning algorithm can be regarded as a greedy algorithm, choosing the current step to do the optimal step in the current situation. More precisely, suppose we have decided on some triangulation strategy, and suppose Y is the current subset of X after several thinning steps. Ideally we would like to remove a point y from $Y_I = Y \setminus X_B$ which minimizes the error measure

$$e(y) = e(y; Y) = E(\mathcal{T}_{Y \setminus y}; X; f). \tag{2.2}$$

In practice, we consider an alternative error measure $e_a(y)$, which we call the *anticipated error*. This error measure is based on decremental Delaunay triangulation: by always using Delaunay triangulations, any new triangulation $\mathcal{T}_{Y \setminus y}$ can be computed from the previous

triangulation $\mathcal{T}_Y$ by making only a few local changes [**9**]. As will be established later, the minimization of the anticipated errors $e_a(y)$ implies, in certain cases, the minimization of the actual errors $e(y)$.

First, for any triangle T whose vertices are in X, we define e_T to be the maximum error over T,

$$e_T = \max_{x \in T \cap X} |L(f,T)(x) - f(x)|, \tag{2.3}$$

where $L(f,T)$ denotes the linear interpolant to f over T. Next let $T_1, \ldots, T_k$ be the k triangles in $\mathcal{T}_Y$ which contain a given vertex y in Y_I. We call the union of these triangles the *cell* $C(y)$. If we now remove y from $\mathcal{T}_Y$, then a Delaunay triangulation $\mathcal{T}_{Y \setminus y}$ can be constructed by retaining $\mathcal{T}_Y$ outside $C(y)$ and by a Delaunay retriangulation of the cell $C(y)$. Thus the triangles $T_1, \ldots, T_k$ are replaced by $k-2$ new triangles $T'_1, \ldots, T'_{k-2}$ (see Figure 1), so that

$$C(y) = \bigcup_{j=1}^{k} T_j = \bigcup_{j=1}^{k-2} T'_j.$$

We set the anticipated error to be

$$e_a(y) = e_a(y; Y) = \max_{j=1,\ldots,k-2} e_{T'_j}. \tag{2.4}$$

We note that since

$$e(y) = \max \left(e_a(y), \max_{T \in \mathcal{T}_Y \setminus \{T_1, \ldots, T_k\}} e_T \right), \tag{2.5}$$

$$e_a(y) \leq e(y).$$

We define our thinning algorithm by saying that a point y in Y_I is *removable* if

$$e_a(y) = \min_{z \in Y_I} e_a(z). \tag{2.6}$$

Figure 1a and 1b. Retriangulation of a cell.

Thinning Algorithm.

1. Set $X_N = X$ and compute a Delaunay triangulation $\mathcal{T}_N$ of X_N.
2. For $i = N, N-1, \ldots, K+1$,
   2. (a) locate a removable point x in $(X_i)_I$

2. (b) let $X_{i-1} = X_i \setminus x$,
2. (c) compute a Delaunay triangulation $\mathcal{T}_{i-1}$ of X_{i-1} from $\mathcal{T}_i$.

The result of the thinning algorithm is a hierarchical sequence of subsets of X,

$$X_B = X_K \subset X_{K+1} \subset \cdots \subset X_N = X,$$

where $|X_i| = \mathrm{i}$, and a sequence of corresponding triangulations $\mathcal{T}_K, \ldots, \mathcal{T}_N$.

This thinning algorithm will be the subject of most of this paper so let us pause to make some observations.

Remarks

1. The basic principles of this thinning algorithm are similar to those of the algorithm proposed by Soucy and Laurendeau [**12**] for the simplification of free form triangle meshes in 3D, where the retriangulation of a cell is done by a local projection into a locally defined plane.
2. By changing the definition of a removable point and/or by changing the strategy of triangulation, various alternative thinning algorithms can result.
3. If our aim is purely to minimize approximation error, a data-dependent triangulation strategy may at first seem more appropriate than Delaunay triangulation. While Delaunay triangulation depends only on the locations of the points in $\mathbb{R}^2$, data-dependent triangulation takes into account the values of f at these points, and aims to reduce the error between the piecewise linear interpolant on the triangulation and the function f. Various strategies for data-dependent triangulation are investigated in [**4**]. However, for data-dependent triangulation, the topological changes required for decremental triangulation are not guaranteed to be local. Thus, we leave the study of the use of data-dependent triangulation strategies to future work.
4. In case the Delaunay triangulation is used, it is possible to use the actual error $e(y)$ as the criterion for the removal of a point instead of the anticipated error $e_a(y)$. The complexity of the thinning algorithm is not changed but there are significant additional costs (see the details in Section 4).
5. In [**5,6**] various definitions of a removable point were proposed which tend to favour well-distributed sets X_i in the plane. In these definitions of a removable point the criterion depends only on the locations of the points in the plane.

3. Theoretical Aspects

In this section, we analyze some properties of adaptive thinning. First, we study adaptive thinning for quadratic polynomials of the form (1.1). For any triangle $T = [v_1, v_2, v_3]$ and a point $x \in T$, let $\lambda_i = \lambda_i(x)$, $i = 1, 2, 3$, be its (non-negative) barycentric coordinates with respect to T.

Lemma 3.1. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be any quadratic polynomial and let $L(f, T)$ be the linear interpolant to f over T. Then*

$$L(f,T)(x) - f(x) = \frac{1}{2}\left(\lambda_1\lambda_2 D^2_{v_2-v_1} f + \lambda_1\lambda_3 D^2_{v_3-v_1} f + \lambda_2\lambda_3 D^2_{v_3-v_2} f\right),$$

where $D_d^2 f$ denotes the second order directional derivative of f in the direction $d = (d_1, d_2) \neq 0$.

Proof: We can express f as a quadratic Bernstein polynomial over T,

$$f(x) = c_{11}\lambda_1^2 + c_{22}\lambda_2^2 + c_{33}\lambda_3^2 + 2\left(c_{12}\lambda_1\lambda_2 + c_{13}\lambda_1\lambda_3 + c_{23}\lambda_2\lambda_3\right).$$

Further, using degree elevation, we have

$$\begin{aligned} L(f,T)(x) &= c_{11}\lambda_1 + c_{22}\lambda_2 + c_{33}\lambda_3 \\ &= c_{11}\lambda_1^2 + c_{22}\lambda_2^2 + c_{33}\lambda_3^2 + \\ &\quad (c_{11}+c_{22})\lambda_1\lambda_2 + (c_{11}+c_{33})\lambda_1\lambda_3 + (c_{22}+c_{33})\lambda_2\lambda_3, \end{aligned}$$

and so

$$L(f,T)(x) - f(x) = (c_{11} - 2c_{12} + c_{22})\lambda_1\lambda_2 + (c_{11} - 2c_{13} + c_{33})\lambda_1\lambda_3 + (c_{22} - 2c_{23} + c_{33})\lambda_2\lambda_3,$$

and therefore, using the well-known formulas for derivatives of Bernstein polynomials over triangles, the result follows. ∎

We will use this result to study the anticipated errors when f has the form (1.1). Indeed, in this case,

$$D_d^2 f = d_1^2 f_{xx} + 2d_1 d_2 f_{xy} + d_2^2 f_{yy} = 2A(d_1^2 + d_2^2) = 2A||d||^2,$$

and so from Lemma 3.1 we find that

$$L(f,T)(x) - f(x) = A\left(\lambda_1\lambda_2||v_2 - v_1||^2 + \lambda_1\lambda_3||v_3 - v_1||^2 + \lambda_2\lambda_3||v_3 - v_2||^2\right).$$

Thus recalling the expression for $e_a(y)$ in equation (2.4) we conclude as follows.

Proposition 3.2. *If f is the quadratic polynomial in (1.1), then the adaptive thinning is independent of the values of f at the data points.*

Moreover, since all (normalized) second order directional derivatives of f in (1.1) are equal, we expect the points generated by the thinning algorithm applied to f to be well-distributed. This is confirmed by a numerical example in Section 6.

Next let us consider the special case of convex data. It turns out that we can analyze the adaptive thinning algorithm of Section 2, modified to the convex data case in the sense that the Delaunay triangulation is replaced by a convex triangulation. It has been established in [**10**] and [**2**] that if f is any convex function, there is a triangulation $\mathcal{T}_X$ of X, which is called a *convex triangulation*, for which the piecewise linear interpolant $L(f, \mathcal{T}_X)$ is convex. The convex triangulation $\mathcal{T}_X$ is unique if no four data points $(x, f(x))$ lie in a plane, and the interpolant $L(f, \mathcal{T}_X)$ is unique in any case. Moreover the convex interpolant $L(f, \mathcal{T}_X)$ lies above f, pointwise,

$$f(x) \leq L(f, \mathcal{T}_X)(x), \qquad x \in \Omega. \tag{3.1}$$

It is also known that in the special case that f is a quadratic polynomial of the form (1.1), with $A > 0$, the convex triangulation and the Delaunay triangulation are one and the same. For a discussion of these and further issues see [**1**].

Clearly if convex data is thinned, a natural triangulation $\mathcal{T}_Y$ for any subset Y of X is also a convex one. Therefore we wish next to discuss a modified thinning algorithm called the *convex thinning algorithm*. This is identical to the thinning algorithm defined in Section 2 except that we always triangulate with a convex triangulation. It is well known (see for example [**1**]) that, similar to Delaunay triangulation, decremental convex triangulation requires only retriangulating the corresponding cell $C(y)$.

Now observe that since $L(f, \mathcal{T}_Y)$ is a convex interpolant to the data $(Y \setminus y, f)$ as well as to the data (Y, f), inequality (3.1) implies

$$f(x) \le L(f, \mathcal{T}_Y)(x) \le L(f, \mathcal{T}_{Y \setminus y})(x), \qquad x \in \Omega,$$

and so, recalling the notation of Section 2,

$$e_a(y) \ge \max_{j=1,\dots,k} e_{T_j}.$$

This together with (2.5) implies

Lemma 3.3. *Suppose f is convex and let Y be any subset of X and $\mathcal{T}_Y$ its convex triangulation. Then for any y in Y_I,*

$$e(y) = \max\{e_a(y), E(\mathcal{T}_Y; X; f)\}.$$

This lemma leads to the following important result:

Proposition 3.4. *Suppose f is convex and let Y be any subset of X and $\mathcal{T}_Y$ its convex triangulation. Then for any y and z in Y_I,*

$$e_a(y) \le e_a(z) \qquad \Longrightarrow \qquad e(y) \le e(z).$$

Thus the convex thinning algorithm minimizes the actual approximation error at each step.

4. Algorithmic Aspects

In this section we study the complexity of the adaptive thinning algorithm and various aspects of its implementation. Recall that the thinning algorithm requires initially computing the Delaunay triangulation $\mathcal{T}_X$ of X. This can be done in $O(N \log N)$ operations, where $N = |X|$ [**9**]. In addition, as in [**6**], we store the interior nodes of $\mathcal{T}_X$ in a heap. A heap is a binary tree which can be used for the implementation of a priority queue. Each node x in the heap bears its anticipated error $e_a(x)$ as its significance value. Due to the heap condition, the significance of a node is smaller than the significances of its two children. Therefore, the root of the heap contains a removable point. It is well-known [**11**] that each insertion, removal, or update of one node in the heap costs $O(\log n)$ operations,

where n is the number of nodes in the heap. In consequence, building the initial heap costs $O(N \log N)$ operations.

As regards the thinning itself, we assume that in every triangulation computed by the thinning algorithm, the number of triangles in every cell is bounded above, i.e. is $O(1)$. Now suppose Y is the current subset and it has size $n = |Y|$. The number of points already removed is $N - n$. We perform Step 2 of the thinning algorithm of Section 2 as follows.

(1) Pop the root y^* from the heap and update the heap.
(2) Remove the node y^* from the triangulation $\mathcal{T}_Y$ and compute $\mathcal{T}_{Y\setminus\{y^*\}}$ by replacing the current triangles $T_1, \ldots, T_k$ in the cell $C(y^*)$ with new triangles $T'_1, \ldots, T'_{k-2}$.
(3) Attach each point $x \in X \cap C(y^*)$ that has previously been removed, in particular y^* itself, to a triangle T'_j which contains x and compute $e_{T'_j}$, $1 \le j \le k-2$.
(4) For each neighbouring vertex y of y^* in $\mathcal{T}_Y$, update its anticipated error $e_a(y)$ and its position in the heap.

Thus, during the performance of the adaptive thinning algorithm, each of the already removed points, including the currently removed point, is attached to a triangle in the current triangulation. These attachments facilitate the computation of the anticipated error of the points that have not been removed yet. As regards number of operations, Step (1) requires $O(\log n)$, step (2) requires $O(1)$, and step (4) takes $O(\log n)$. As regards step (3), we can assume that the current number of triangles is proportional to n and so, under the assumption that the $N - n$ removed points are uniformly distributed over these triangles, the number of operations in step (3) is of the order of $(N - n)/n$. Therefore summing the costs of steps (1) to (4) for all n, we find that the total cost of the thinning algorithm is $O(N \log N)$.

This algorithm can be modified to use the actual error (2.2) instead of the anticipated error (2.4), without changing the complexity, by maintaining two heaps. One heap, the e-heap, consists of $\{e_a(y) : y \in Y_I\}$, with $e_a(y)$ defined in (2.4). In this heap the root y^* points to the minimal element, i.e. $e_a(y^*) = \min_{y\in Y_I} e_a(y)$. The second heap, the T-heap, consists of $\{e_T : T \in \mathcal{T}_Y\}$ with e_T defined in (2.3). In this heap the root T^* points to the maximal element, i.e. $e_{T^*} = \max_{T\in\mathcal{T}_Y} e_T$. From (2.5), it is easy to see that there must be a removable point among y^* and the three vertices of the triangle T^*. As to the complexity of the modified algorithm, note that we have to update both the T-heap and the e-heap, and our estimates show that the number of required operations in the modified step (4) is significantly increased. Thus the modified algorithm still requires $O(N \log N)$ operations, but with a larger constant.

5. Alternative Adaptive Thinning Algorithms

In this section we discuss some possible simplifications of the thinning algorithm of Section 2, which we refer to as AT1, by simplifying the method for computing the anticipated error of a point.

When considering which point y to remove from the current subset Y, one could simply ignore all the points that have previously been removed. Thus we would replace $e_a(y)$ in (2.4) by the simpler error measure

$$e_a(y) = E(\mathcal{T}_{Y\setminus y}; Y; f) = |L(f, \{T'_1, \ldots, T'_{k-2}\})(y) - f(y)|.$$

We will call this simpler thinning algorithm AT2. We have also explored a faster thinning algorithm, AT3, which not only ignores the points already removed but also computes an anticipated error for each point without needing to temporarily retriangulate its cell.

The basic idea behind AT3 is to define the anticipated error $e_a(y)$ as the maximum of the 'directional' anticipated errors at y in a certain sample of directions. For each neighbouring vertex z in $\mathcal{T}_Y$ of y, we consider the unique point p lying at the intersection of the boundary of $C(y)$ and the straight line passing through z and y (other than z itself). Such a point exists, since $C(y)$ is a star-shaped polygon. The point p moreover, is either a vertex of $\partial C(y)$ or a point on one of its sides. In either case, p lies on at least one edge of $\partial C(y)$. Let us denote such an edge by $[z_2, z_3]$; see Figure 2. Then the triangle $T_z = [z, z_2, z_3]$, with vertices in $Y \setminus y$, contains y. We call T_z a *directional triangle* of y. We then let

$$e_a^z(y) = |L(f, T_z)(y) - f(y)|$$

be the (unique) directional anticipated error of y in the direction $z - y$, and define the *anticipated error* at y as

$$e_a(y) = \max_{z \in V_y} e_a^z(y).$$

where V_y is the set of all neighbouring vertices of y in $\mathcal{T}_Y$.

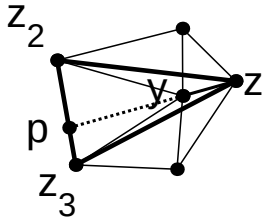


Figure 2. Directional triangle of y.

6. Numerical Examples

We have implemented the thinning algorithms AT1, AT2, and AT3 of the previous section, together with one non-adaptive thinning algorithm, NAT, of [**6**]. The algorithm NAT ignores the samples $f(x_i)$ and simply favours evenly distributed subsets of the points x_i. Our implementation only removes interior points, though all the algorithms could easily be extended so as to remove also (non-extremal) boundary points. In this section we compare the performance of these four algorithms in terms of both approximation quality and computational cost. To this end, we have considered using one specific example from terrain modelling. The corresponding data set, *Hurrungane*, contains 23,092 data points. Each data point is of the form $(x, f(x))$, where $f(x)$ denotes the terrain's height value sampled at the location $x \in \mathbb{R}^2$. This data set is displayed in Figures 3a and 3b.

For all four thinning algorithms, we have recorded both the required seconds of CPU time (without considering the computational costs required for building the initial Delaunay triangulation and the heap) and the sequence of approximation errors $E(\mathcal{T}_Y; X; f)$ after the removal of $n = 1000, 2000, \ldots, 22000$ points. Not surprisingly, we found that

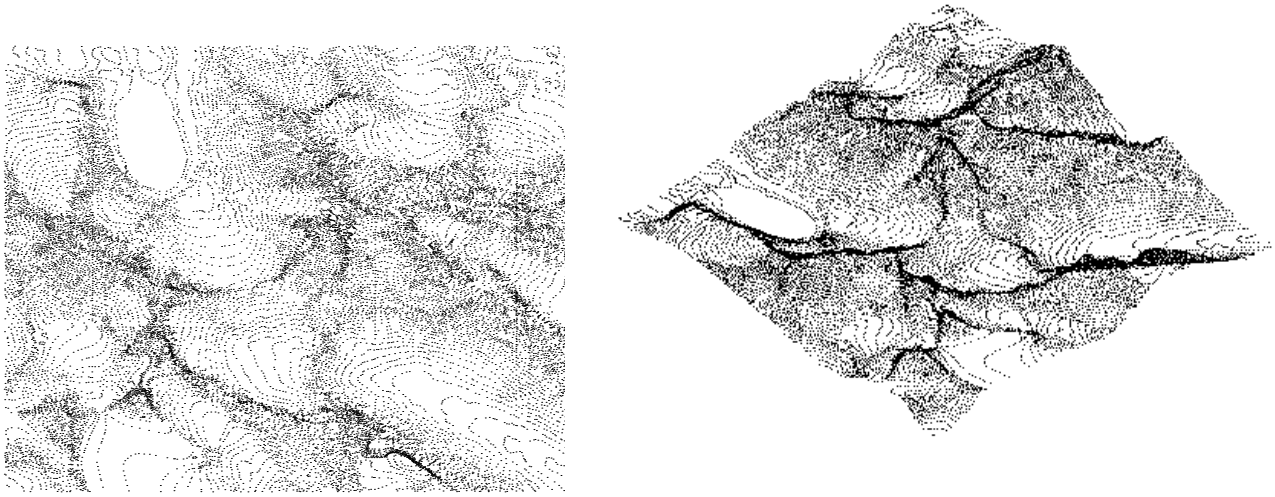


Figure 3a and 3b. Hurrungane: 2D and 3D view

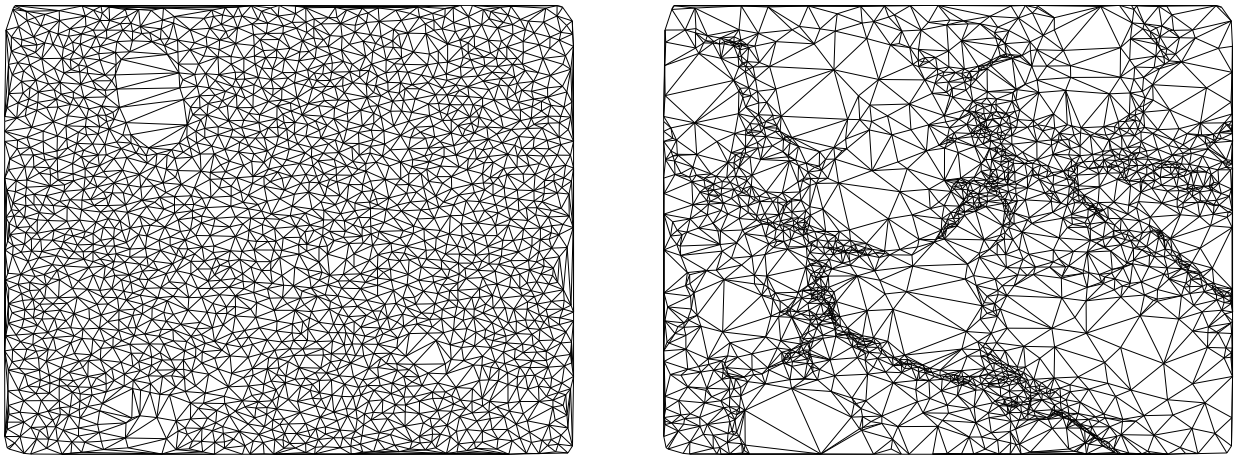


Figure 4a and 4b. Thinned Hurrungane with 1092 points — 2D view
NAT (left) and AT1 (right)

NAT is the fastest method but also the worst one in terms of its approximation error. For example, for $n = 22000$ the algorithm AT1 takes 247.53 seconds of CPU time, whereas NAT takes only 11.37 seconds. On the other hand, we obtain in this particular example $E(\mathcal{T}_Y; X; f) = 278.61$ for NAT, but only $E(\mathcal{T}_Y; X; f) = 30.09$ when using AT1. The two corresponding triangulations $\mathcal{T}_Y$ output by NAT and AT1 are displayed in Figures 4a and 4b (2D view) and 4c and 4d (3D view). In Figures 5a and 6a the approximation error as a function of the number of removed points is plotted for the different thinning algorithms, while in Figures 5b and 6b the corresponding seconds of CPU time are displayed.

The graphs show that, with respect to approximation error, the three adaptive thinning algorithms AT1, AT2, and AT3 are much better than NAT. Among the three adaptive thinning algorithms, AT1 is the best, followed by AT3, and lastly AT2. Note that by defi-

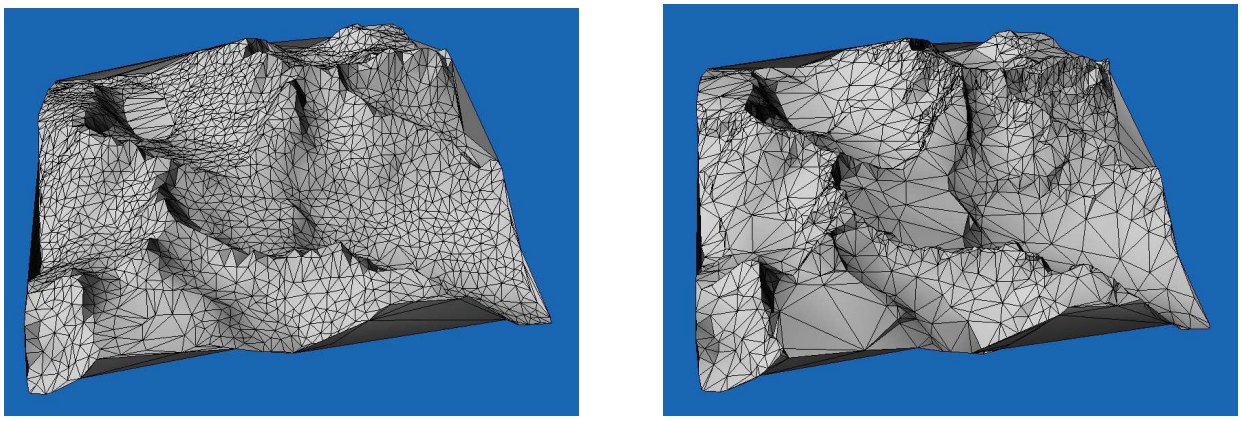


Figure 4c and 4d. Thinned Hurrungane with 1092 points — 3D view
NAT (left) and AT1 (right)

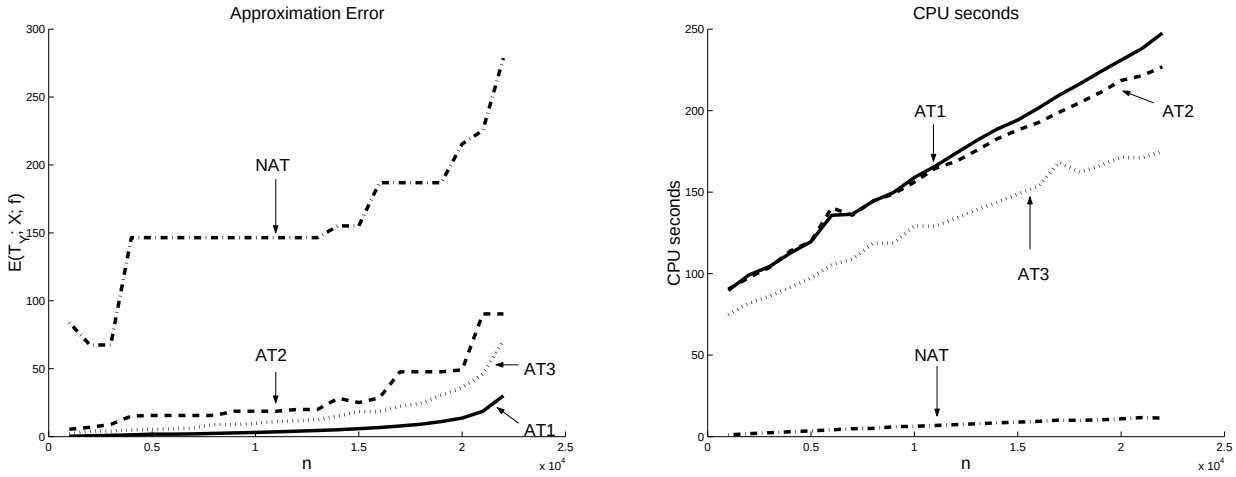
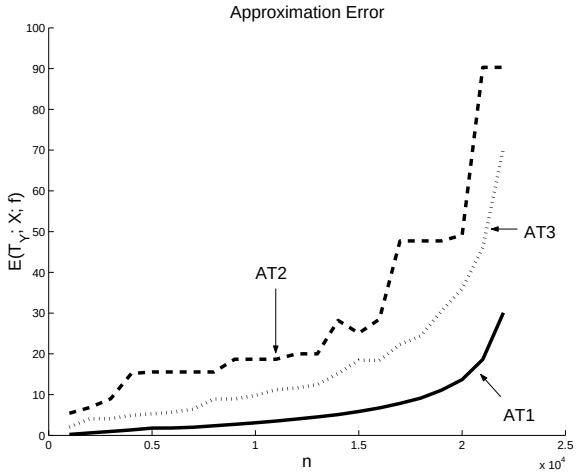


Figure 5a and 5b. Hurrungane: comparison between NAT (dash-dot line), AT1 (solid), AT2 (dashed), and AT3 (dotted), approximation error (left) and seconds of CPU time (right).

nition AT3 can only be inferior to AT2 after one removal. In the numerical examples AT3 has continued to be inferior for about 50 removal steps, after which its approximation error is smaller than that of AT2.

As to the computational costs for the adaptive thinning algorithms, AT3 is the fastest, and AT1 the slowest, cf. Figures 5b and 6b. Our conclusion is that AT1 is our recommended thinning algorithm. But if computational time is a critical issue, AT3 is a good alternative.

In a further example, we have sampled the function $f(x) = x_1^2 + x_2^2$ at 2000 randomly chosen points in the unit square, displayed in Figure 7a. We then removed 1500 points from this data set using the algorithm AT1. Figure 7b shows the 500 remaining points. Recall that according to Proposition 3.2, the algorithm AT1 does not depend on the function values of f. The distribution of the points in Figure 7b is fairly even. Our numerical computations show that the distribution is very similar to that obtained by NAT.



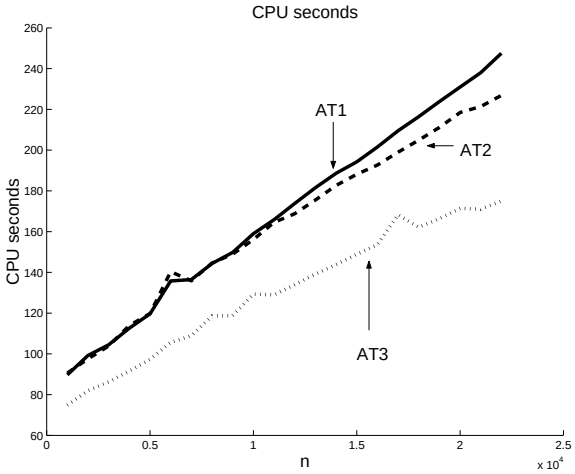
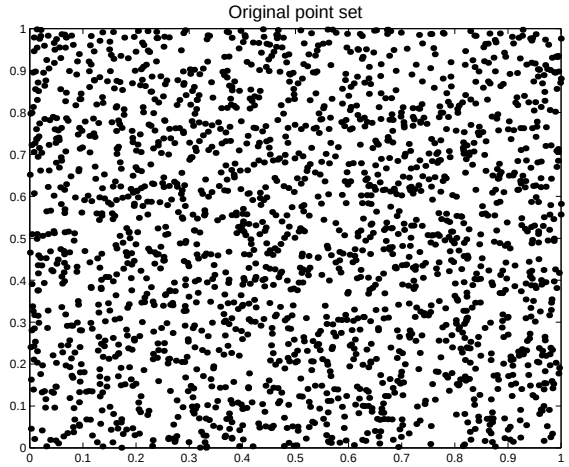


Figure 6a and 6b. Hurrungane: comparison between AT1 (solid line), AT2 (dashed), and AT3 (dotted), approximation error (left) and seconds of CPU time (right).



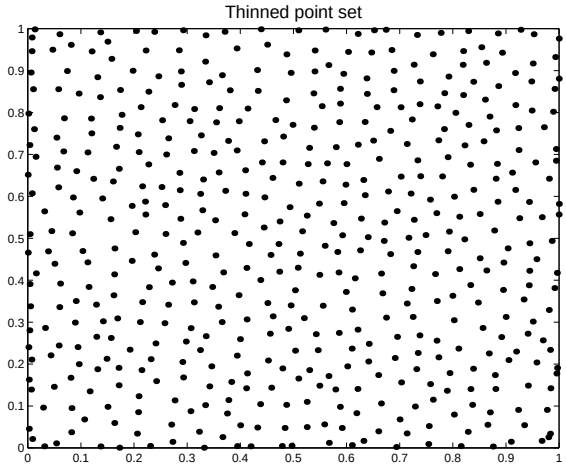


Figure 7a and 7b. $f(x) = x_1^2 + x_2^2$ sampled at 2000 randomly chosen points (left) and thinned by AT1 to 500 points (right).

Acknowledgement: The authors were partly supported by the European Union within the project MINGLE (Multiresolution in Geometric Modelling), contract no. HPRN-CT-1999-00117.

References

1. J. M. Carnicer and M. S. Floater, Piecewise linear interpolants to Lagrange and Hermite convex scattered data, Numerical Algorithms **13** (1996), 345–364.
2. W. Dahmen and C. A. Micchelli, Convexity of multivariate Bernstein polynomials and box spline surfaces, Stud. Sci. Math. Hung. **23** (1988), 265–287.
3. N. Dyn, M. S. Floater and A. Iske, Univariate adaptive thinning, Mathematical Methods for Curves and Surfaces: Oslo 2000, T. Lyche and L. L. Schumaker (eds.), Vanderbilt University Press, Nashville, 2001, 123–134.
4. N. Dyn, D. Levin and S. Rippa, Data dependent triangulations for piecewise linear interpolation, IMA J. Numer. Anal. **10** (1990), 137–154.
5. M. S. Floater and A. Iske, Multistep Scattered Data Interpolation using Compactly Supported Radial Basis Functions, J. Comp. Appl. Math. **73** (1996), 65–78.
6. M. S. Floater and A. Iske, Thinning algorithms for scattered data interpolation, BIT **38** (1998), 705–720.
7. C. Gotsman, S. Gumhold and L. Kobbelt, Simplification and Compression of 3D Meshes, preprint.
8. P. S. Heckbert and M. Garland, Survey of Surface Simplification Algorithms, Technical Report, Computer Science Dept., Carnegie Mellon University, 1997.
9. F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer, New York, 1985.
10. D. S. Scott, The complexity of interpolating given data in three space with a convex function of two variables, J. Approx. Theory **42** (1984), 52–63.
11. R. Sedgewick, *Algorithms*, Addison-Wesley, Reading, MA, 1983.
12. M. Soucy and D. Laurendeau, Multiresolution Surface Modeling Based on Hierarchical Triangulation, Computer Vision and Image Understanding **63** (1996), 1–14.

Nira Dyn
Tel-Aviv University
School of Mathematical Sciences
Tel Aviv 69978, ISRAEL
niradyn@math.tau.ac.il

Michael S. Floater
SINTEF
Post Box 124, Blindern
N-0314 Oslo, NORWAY
mif@math.sintef.no

Armin Iske
Technische Universität München
Zentrum Mathematik
D-80290 München, GERMANY
iske@ma.tum.de