Progressive TINs: Algorithms and Applications*

Anil Maheshwari Pat Morin Jörg-Rüdiger Sack

School of Computer Science, Carleton University
{*maheshwa,morin,sack*}*@scs.carleton.ca*

Abstract

Transmission of geographic data over the Internet, rendering at different resolutions/levels of detail, or processing at unnecessarily fine detail pose interesting challenges and opportunities. In this paper we explore the applicability to GIS of the notion of progressive meshes, introduced by Hoppe [13] to the field of computer graphics. In particular, we describe progressive TINs as an alternate method to hierarchical TINs, design algorithms for solving GIS tasks such as selective refinement, point location, visibility or line of sight queries, isoline/contour line extraction and provide empirical results which show that our algorithms are of considerable practical relevance. Moreover, the selective refinement data structure and refinement algorithm solves a question posed by Hoppe.

1 Introduction

In the field of Computer Graphics, very recently, Hoppe [13], proposed the *progressive mesh* (PM) representation, which can represent a triangulated mesh at a continuous level of resolution. In essence, a mesh in the PM format consists of a coarse mesh, M_0, and a sequence of operations which successively transform the coarse mesh into progressively finer meshes, $\{M_1, M_2, M_3, \ldots, M_n\}$, until the high level detailed input mesh, M_n, is (re)created.

The progressive mesh representation naturally supports some operations not found in standard representations of triangulated meshes. *Level-of-Detail (LOD) Approximation* allows for viewing and manipulating the mesh at varying levels of detail (number of vertices). *Progressive transmission* allows for the transmission of a mesh across a network connection, so that the receiver can immediately display an approximation of the mesh, and refine this display as more data is received. *Selective refinement* is the ability to view and manipulate part of the mesh at a high level of detail while the remainder of the mesh remains at a low level of detail.

*This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and Almerco Inc.

In this paper we are interested in *progressive triangulated irregular networks*, progressive TINs, as an alternate representation to hierarchical TINs as described in [3, 5, 4, 2]. A TIN is one of the fundamental data structures for representing geographical data where a triangulated set of points is stored together with e.g., its elevation. TINs were first introduced in 1978 (see [21, 22, 6]).

Storing data at different levels of detail is becoming increasingly important due to changes for GIS in users, devices, and communication technology. Today's users have access to geographic data at a level of detail orders of magnitude greater than just a few years ago. Frequently, there is no need to work with a TIN at a high level of detail to obtain a desired result, or it is inefficient or even computationally infeasible. Users often obtain or share data remotely via the Internet or other communication lines. In a progressive scheme, a user can interrupt the communication process at any time, with an approximate solution, without waiting for the final result.

In this paper we introduce the concept of progressive TINs and present schemes for selective refinement[1], which in turn helps us in addressing a number of GIS applications such as exact and approximate point location queries, visibility queries, and isoline/contour extraction (conversion to contour lines). Empirical results are presented for some of our algorithms which show that they are of considerable practical relevance. Furthermore, we sketch representations of progressive TINs in external memory.

2 Related Work

The problem of level of detail approximation in TINs has received much attention in the field of GIS. This research takes the form of Hierarchical TINs; surveys of these schemes are included in [1] and [12]. Each of these schemes use a *tree* [3, 5, 4] or *directed acyclic graph (DAG)* [2] structure to store the triangles of a triangulation in a hierarchical manner. These schemes support LOD approximation by only refining the mesh to a certain level in the hierarchy, and support efficient selective refinement by searching the structure in a top-down manner. Although not described in the literature, these schemes can also support a limited form of

[1] In this paper we present a solution to the problem of efficient selective refinement, originally posed in [13]. A similar solution to the problem of selective refinement for meshes has been independently and concurrently proposed by Hoppe [14] with Computer Graphics applications in mind.

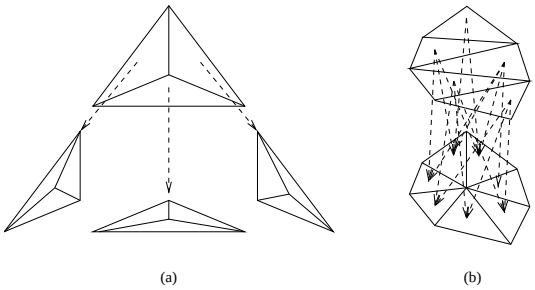

Figure 1: An example of (a) the thin triangles formed by tree based schemes and (b) the high complexity of DAG based schemes.

progressive transmission by transmitting the levels of the hierarchy one at a time.

Unfortunately, both the tree-based and DAG-based schemes have their drawbacks (see Figure 1). Tree-based schemes either tend to form long thin triangles (since triangles are nested inside each other) which can lead to numerical instabilities, or they form surface discontinuities. DAG-based schemes tend to be complex due to the variable (possibly high) degree of their internal nodes. This also makes them less space efficient than tree-based schemes. DAG based schemes also tend to be less efficient in practice since they are usually based on the Kirkpatrick hierarchy [16] which is known to have large constants.

Recently, Snoeyink and van Kreveld [24] presented an algorithm for permuting the vertices of a planar point set so that Delaunay triangulation (or any canonical triangulation) of this point set can be constructed incrementally in linear time. It is unlikely that their scheme can support selective refinement.

The progressive TIN representation avoids the thin triangles associated with tree based hierarchical schemes since it does not require triangles to be nested inside each other. The constants in progressive TIN algorithms are also smaller than those in DAG based schemes. Furthermore, the progressive TINs naturally support efficient selective refinement and, as shown here, lend themselves to the development of a number of GIS applications.

3 The Progressive Mesh Representation

The progressive mesh (PM) representation is based on the *edge collapse* transformation and its inverse the *vertex split*. These transformations have been described by Gold [7] for the dynamic maintenance of Voronoi diagrams, and Hoppe [13] in the context of progressive meshes. An edge collapse involves collapsing an edge by identifying its two incident vertices, v_1 and v_2, into a single aggregate vertex v. The two adjacent faces (v_1, v_2, v_l), and (v_1, v_2, v_r) vanish in the process. An edge collapse and its inverse vertex split are illustrated in Figure 2.

In the progressive mesh representation, a mesh M, is represented as a pair, $(M_0, vsplits)$, where M_0 is a coarse-grained mesh and *vsplits* is a list of vertex splits which will reproduce the original mesh when applied in order. A vertex split transformation $vsplits(v, v_1, v_2, v_l, v_r, A)$ splits the aggregate vertex v into two vertices v_1 and v_2 and adds the edges (v_1, v_l), (v_1, v_r), (v_2, v_l), and (v_2, v_r). A stores attribute information for the neighbourhood of the transformation, including but not limited to, the positions of the two new vertices.

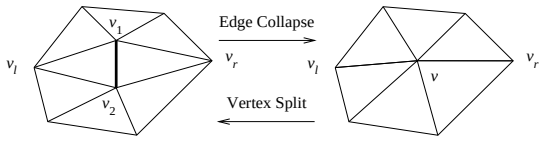

Figure 2: The edge collapse transformation and its inverse, the vertex split

We call v the *parent* of v_1 and v_2, and we call v_1 and v_2 the *children* of v. We say that a vertex u is an *ancestor* of a vertex v if $u = v$ or if u is an ancestor of the parent of v. In this case we call v a *descendent* of u. We denote by M_i the mesh obtained by performing the first i elements of *vsplits* on the coarse grained mesh M_0. The original mesh, M, can be obtained by applying all the elements of *vsplits* in order, i.e., $M_{|vsplits|} = M$.

The progressive mesh construction algorithm takes a mesh, M, performs a series of edge collapse operations to obtain the mesh M_0, and sets *vsplits* to the list of vertex split operations that invert the edge collapses performed. The order in which the edge collapses are performed is determined by an application specific fitness function (e.g., minimizing the geometric error). The edges of the mesh are placed in a priority queue based on their fitness and are removed one at a time as they are collapsed. Edges in the neighbourhood of a collapse may have their priorities updated. A fitness function which considers the geometry, discrete attributes and scalar attributes of the mesh is described in [13]. Sufficient requirements for an edge collapse to be valid are given in [15].

4 Selective Refinement

In this section we describe a new method of performing selective refinement in progressive meshes. The method is simple, numerically robust, and easy to implement. Due to space limitations we only sketch the selective refinement data structures and algorithms. For details, refer to [18].

Our approach can be summarized as follows: In a preprocessing phase, we associate with each vertex, v, a *region of influence* outside of which splitting v has no effect. By using the parent/child relationships between vertex split records these regions of influence can be organized as a forest of rooted binary trees. When the records are organized in this manner it is possible to efficiently identify all the vertex split records whose regions of influence intersect a given query region q. Once the relevant vertex split records have been identified, they are applied in the correct order.

We treat the *vsplits* list as a dependency graph in which a vertex v_1 depends on a vertex v if v is split and one of the resulting vertices is v_1. Observe that the dependency graph is a forest of rooted binary trees whose roots are the vertices of the coarse TIN, M_0, and whose leaves are the vertices of

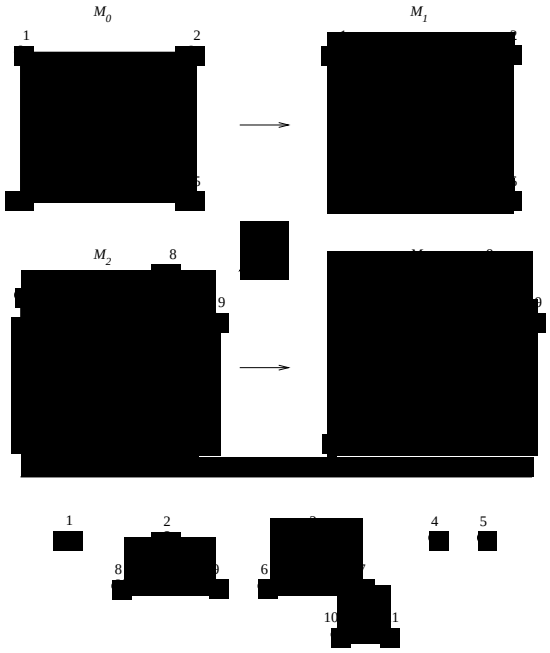

Figure 3: A sequence of vertex splits and its associated dependency graph.

the original mesh, M (see Figure 3 for an illustration).

With each node v in the dependency graph we associate an axis parallel 3-dimensional bounding box, denoted $roi(v)$ which represents the region outside of which this vertex split has no influence. We define $roi(v)$ recursively as follows: if v is a leaf then $roi(v)$ is the smallest box which encloses all the neighbours of v in M, otherwise $roi(v)$ is the smallest box which encloses $roi(v_1)$ and $roi(v_2)$ where v_1 and v_2 are the two children of v. These boxes, i.e., the roi's can be computed in a bottom-up fashion during the progressive TIN construction procedure. Alternatively, these boxes can be constructed using a post order traversal of the forests.

Once the *vsplits* list is augmented with the roi information, selective refinement can be carried out in the following manner: For each vertex of M_0 we search the tree rooted at M_0 and retrieve all vertex splits, v, for which $roi(v)$ overlaps the query region, q. If $roi(v)$ does not intersect q then we do not need to search the subtree rooted at v, since all the regions of influence of the children of v are contained in $roi(v)$. Therefore, retrieving the pertinent vertex splits can be done in time $O(|M_0| + k)$, where k is the number of splits retrieved.

Once the *vsplits* records are retrieved they need to be sorted and applied in order. The sorting could be done by a standard sorting algorithm, but this would take $O(k \log k)$ time. In the full version of this paper [18], we show that the sorting of the records as well as the *vsplits* transformations can be applied in $O(kd)$ time, where d is the average degree of the vertices that are split, using topological sort.

Preprocessing requires $O(n)$ time, since it consists of post-order traversals of the dependency graph, which is a tree. The running time of the selective refinement algorithm is dominated by the time it takes to sort the retrieved vertex splits and applying them and is therefore $O(kd)$.

At this point we note that it is also possible to generalize this scheme to perform selective refinement on any mesh $M_t \in \{M_0, \ldots, M_{|vsplits|}\}$. In this case, the query region is returned at exactly the level of detail at which it appears in M_t. To achieve this generalization, we simply "prune" the search when a vertex split record is reached whose index is greater than t. In analyzing the running time of the generalized algorithm, the value of k is defined in the same manner as above, but with respect to the vertex split sequence $v_1, \ldots, v_t$.

The preprocessing and selective refinement procedures are quite simple, and only small constants are hidden in the "big-O" notation. Another merit of this scheme is that since it uses only axis parallel bounding boxes, the search procedure can be implemented using only comparison operations and is therefore not subject to the rounding errors inherent in floating point arithmetic computations.

5 Progressive TINs

In this section, we suggest the progressive TIN representation as a viable alternative to traditional hierarchical schemes. Combined with the selective refinement scheme presented in Section 4 the progressive TIN representation avoids the problem of thin triangles, as well as the large constants associated with the Kirkpatrick hierarchy. Also, as the following sections show, the progressive TIN representation naturally support a number of common TIN operations. Furthermore, these operations require little or no additional preprocessing or storage overhead.

Because of the small constants involved in the PM representation, the algorithms presented in this section are competitive with existing algorithms which operate on TINs in a standard representation. The advantage of these algorithms over these existing algorithms are:

1. these algorithms work on TINs in the PM representation, maintaining all the advantages of the PM representation,
2. these algorithms require little or no additional preprocessing or storage ($O(n)$ preprocessing and $O(n)$ storage) making them, in some cases, more efficient than existing solutions, and
3. by using the "pruning" technique described in Section 4, all the algorithms presented here can be used to efficiently solve approximate versions of the problem in question, i.e., the problem can be solved on any of the meshes $M_0, \ldots, M_{|vsplits|}$.

Throughout the remainder of this section we use T to denote the TIN being operated on in the same manner as we have previously used M to denote an arbitrary mesh.

5.1 Point Location

Point location in a TIN is a well studied problem in GIS and a number of algorithms exist for this problem. Given a TIN, T, and a query point q, the point location problem is to determine the face of T in which q lies. Although theoretically

optimal algorithms ($O(n)$ preprocessing time and $O(\log n)$ query time e.g., [16]) exist for the point location problem, these algorithms have large constants which makes them less useful in practice. (Most real commercial implementations use schemes that are theoreticallly sub-optimal but which work usually well in practice [8, 19].)

To perform point location on a progressive TIN, we perform selective refinement in which the query region is a single point, namely the query point q. The selective refinement algorithm is run and the triangle in which the query point lies is found. The running time of this algorithm is clearly $O(|M_0| + kd)$ where k and d are defined as in Section 4.

If our objective is to do largely point location queries in the TIN, then the fitness function that can be used in the construction algorithm to obtain a progressive TIN is to always collapse the edge (v_1, v_2) such that the resulting region of influence is the smallest among all possible edge collapses. This heuristic seems to perform very well in practice (see Section 6). (A similar heuristic is used in the construction of R-Trees [11].)

5.2 Isoline Extraction

The problem of isoline or contour line extraction is stated as follows: given a query elevation h, return all triangles of T which occur at height h. Given the triangles which occur at elevation h, the polygons and polygonal chains which form the isolines can be found in $O(k)$ time (where k is the number of triangles) by starting at an arbitrary triangle and walking along triangles until a triangle already visited is reached, or the border of T is reached. When this occurs, another unvisited triangle is chosen, and the same procedure is applied. The problem of extracting isolines from TINs has been addressed in [25] and from hierarchical TINs in [5].

For answering isoline queries on a progressive TIN, we associate with each vertex v, information about the minimum and maximum elevation affected by the splitting of that vertex. We denote this information by $minh(v)$ and $maxh(v)$, respectively. If v is a vertex of T, this information consists of the minimum and maximum elevation which occurs in the immediate neighbourhood of v in T. If v is obtained by collapsing (v_1, v_2), we let $minh(v) = \min\{minh(v_1), minh(v_2)\}$ and $maxh(v) = \max\{maxh(v_1), maxh(v_2)\}$. Note that interval $[minh(v), maxh(v)]$ is analogous to the region of influence $roi(v)$ defined in Section 4 and can be computed in linear time with a post-order traversal of the dependency graph.

To answer an isoline query, *vsplits* is searched in the same manner as is done for selective refinement except that rather than testing if the region of influence of a vertex, v, intersects a query region, the test is whether h lies in the interval $[minh(v), maxh(v)]$. The splits retrieved are then applied in order. At this point, the TIN is fully refined around the contour lines at height h, and very coarse in other areas. To complete the query processing, it remains only to report the triangles whose minimum and maximum elevations include h. The running time of this algorithm is $O(|M_0| + kd)$ where k is the number of vertex splits whose height intervals overlap h, and d is defined as in Section 4.

From this, it follows that the edge collapse sequence can be optimized for isoline extraction by always collapsing the edge which produces the smallest interval in the resulting supervertex. This heuristic is similar to that suggested in Section 5.1 for optimizing the edge collapse sequence for point location.

Progressive TINs also support progressive transmission of isolines. To perform progressive transmission of isolines, we note that a vertex split operation can only affect isolines in the neighbourhood of the split vertex, and so maintaining isolines under the vertex split operation is a localized matter. Therefore, progressive transmission of isolines involves first transmitting the coarse grained TIN, T_0, and then transmitting the vertex split records which affect the elevation(s) in question. At the receiving end, extracting the isolines from T_0 can be done using a linear time brute force method, and these isolines can be maintained using only local operations as vertex splits records are received.

5.3 Visibility Queries

Two points on a TIN, T, are said to be visible if the line segment joining them does not intersect T. The visibility query problem is the following: given two query points p and q, is p visible from q?

A straightforward solution to the visibility query problem is as follows. Locate the triangle in which p lies and then walk along triangles in the direction of q until either (1) an edge is crossed that occludes q in which case the answer is negative, or (2) the triangle containing q is reached in which case the answer is positive. The preprocessing required by this algorithm is the same as that for point location. The query time is the time to locate q plus the number of edges, k, which intersect the segment $\overline{pq}$.

Alternatively, on a progressive TIN, a visibility query between a pair of points can be answered by selectively refining the TIN along the line segment $\overline{pq}$ while at the same time locating p, and then walking from p to q along the segment $\overline{pq}$. This requires no additional preprocessing or storage. The running time is proportional to the number of *vsplits* records whose regions of influence intersect the line segment $\overline{pq}$.

The above procedure can be made more efficient, at the cost of a small increase in the storage (i.e., two real values) per triangle. For each triangle t in the progressive TIN, let $maxh(t)$ (respectively, $minh(t)$) denote the maximum (respectively, minimum) height, over all points in t. (Note that, in general, at any stage in the progressive TIN, a triangle t intersects several triangles at finer resolution.) To determine whether the query points p and q are visible, we follow the following steps:

1. Locate p and q in the coarse triangulation T_0 and then walk from p to q, collecting all triangles in the coarse representation that intersects the segment $\overline{pq}$.
2. Let t be one of the coarse triangles obtained in the previous step. Let s be the portion of the line segment $\overline{pq}$ that lies in t. Let $maxh(s)$ (respectively, $minh(s)$ denote the maximum (respectively, minimum) y-coordinate for the points on the segment s. Observe that if $maxh(s) < minh(t)$, then p and q are not visible. In this case, we stop and report that p and q are not visible. Moreover, if $minh(s) > maxh(t)$ then p and q are visible with respect to t, and there is no need to do further refinement of t.

3. Let S denote the set of coarse triangles at the end of Step 2, for which $maxh(s) > minh(t)$ and $minh(s) < maxh(t)$, i.e., those triangles which could possibly be visible. Which triangles should be refined in the set S? We propose a simple heuristic, which refines first those triangles in S whose $minh$ or $maxh$ value is close to the critical values. These are the triangles, for which after a few steps of the refinement, we are likely to conclude whether they block visibility between p and q, or they do not block the visibility.

Presently, we are pursuing the implementation of the above schemes for the visibility problem in progressive TINs.

5.4 External Memory Progressive TINs

External memory methods for TINs are of significant practical relevance to the field of GIS, as the amount of geographic data currently available exceeds the capacity of internal memories. This motivates the development of an external memory storage scheme for progressive TINs. With such a scheme, the coarse grained TIN, T_0, could be stored in internal memory, and user could perform selective refinement to refine a small portion of this TIN and work with it.

A number of external memory spatial data structures exist which can be used to store progressive TINs. We believe that the data structure which is well suited for this application is the R-tree [11] or one of its variants, as R-trees are designed specifically for storing axis parallel boxes. (A variety of other spatial index structures exist see e.g. [23, 9, 10, 17] and also consult the new survey article by Nievergelt and Widmayer [20].) In particular, one variant of the R-tree, the packed R-tree constructs an R-tree in a bottom up fashion from a static set of axis aligned boxes.

Thus in order to construct an external memory representation of a progressive TIN, we perform the progressive TIN construction algorithm and then build a packed R-tree on the list of vertex splits. When performing selective refinement we use the packed R-tree to extract the relevant vertex splits and then proceed in the manner described in Section 4 to sort and apply the splits.

Using this representation, the algorithms for point location, elevation queries, and visibilities queries can all be applied to progressive TINs stored in external memory.

6 Empirical Results

In this section we present empirical results for our selective refinement algorithm. Test were performed on randomly generated TINs as well as two TINs representing an actual geographical area, containing 10000 and 20000 points, each. Random TINs were generated by choosing uniformly distributed points in the unit square and then computing a Delaunay triangulation of these points. All results for random TINs were compared with those of the two real TINs with the same number of vertices, and were found to be nearly identical.

Figure 4 shows the ratio between the number of vertices in a query region and the number of vertex splits which are retrieved when selectively refining the region. The tests

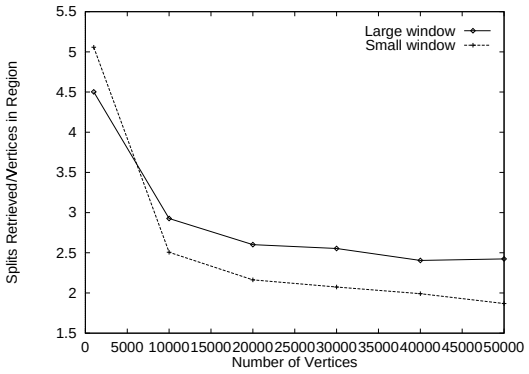

Figure 4: Performance of selective refinement algorithm.

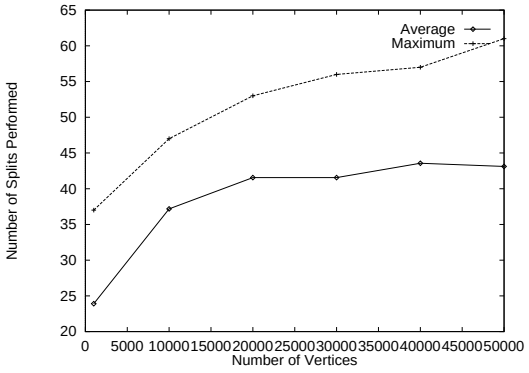

Figure 5: Performance of point location algorithm.

take a query window of a fixed size and places it at 2500 regularly spaced locations on the TIN and perform selective refinement at each location. Both small (1/25 of the TIN's surface area) and large (1/4 of the TINs surface area) query windows were tested. The main results of these tests is that the ratio between the number of splits performed and the number of vertices in the query window converges to a small constant (< 3) as n increases. This confirms that the running time of the selective refinement algorithm is directly proportional to the complexity of the query region.

Figure 5 shows results for the planar point location technique described in Section 5.1. The query point was placed at 2500 regularly spaced location on the TIN and point location was performed. Figure 5 shows the results for TINs with up to 50000 vertices, and shows that even the worst-case running times tend to be logarithmic in the size of T. Also of interest are the absolute values in Figure 5 since these show that the constants are quite small. In no case does the number of vertex splits performed exceed 70.

7 Conclusions

The progressive TIM appears to be a suitable structure to represent terrains at different levels of detail. It has low overhead, is conceptually simple and easy to implement. We have given a number of important GIS applications of this

data representation as well as some empirical results. Other applications we have considered include finding the maximum height value in a region, or refining selectively in a moving rectangle. A natural question which arises out of our work is to find the properties of the class of problems where selective refinement is an effective tool.

References

[1] Andrews, D. Simplifying terrain models and measuring terrain model accuracy. Tech. rep., University of British Columbia Computer Science Department, 1996.

[2] de Berg, M., and Dobrindt, K. On levels of detail in terrains. In *Proc. 11th Annual ACM Symp. on Computational Geometry (SCG '95)* (1995).

[3] de Floriani, L. A pyramidal data structure for triangle based surface description. *IEEE Computer Graphics and Applications 9* (1989), 67–78.

[4] de Floriani, L., Magillo, P., Puppo, E., and Bertolotto, M. Variable resolution operators on a multiresolution terrain model. In *4th ACM Workshop on Advances in Geographic Information Systems* (1996), pp. 123–130.

[5] de Floriani, L., Mirra, D., and Puppo, E. Extracting contour lines from a hierarchical surface model. In *Eurographics '93* (1993), pp. 249–260.

[6] Gold, C. M. The practical generation and use of geographic triangular element data. *Harvard Papers on Geographic Information Systems 5* (1978).

[7] Gold, C. M. Problems with handling spatial data—the Voronoi approach. *CISM Journal 45*, 1 (1991), 65–80.

[8] Goodrich, M. T., Orletsky, M., and Ramaiyer, K. Methods for achieving fast query times in point location data structures. In *Proc. 8th ACM Symp. on Discrete Algorithms (SODA '97)* (1997).

[9] Guenther, O. The design of the cell tree: an object-oriented index structure for geometric databases. In *Proceedings of the 5th IEEE Conference on Data Engineering* (1989), pp. 598–615.

[10] Güting, R. H. An introduction to spatial database systems. *The VLDB Journal 3* (1994), 357–399.

[11] Guttman, A. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference 1984* (1985), pp. 47–57.

[12] Heckbert, P. S., and Garland, M. Multiresolution modelling for fast rendering. In *Proc. Graphics Interface '94* (1994), pp. 43–50.

[13] Hoppe, H. Progressive meshes. In *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996), pp. 99–108.

[14] Hoppe, H. View-dependent refinement of progressive meshes. In *Computer Graphics (SIGGRAPH '97 Proceedings)* (1997).

[15] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuezle, W. Mesh optimization. *Computer Graphics (SIGGRAPH '95 Proceedings)* (1995), 247–254.

[16] Kirkpatrick, D. G. Optimal search in planar subdivisions. *SIAM Journal on Computing 12* (1979), 18–27.

[17] Laurini, R., and Thompson, D. *Fundamentals of Spatial Information Systems.* Academic Press, London, 1992.

[18] Maheshwari, A., Morin, P., and Sack, J.-R. Progressive TINs: Algorithms and applications. Tech. rep., Carleton University School of Computer Science, 1997.

[19] Näher, S. The LEDA manual, Version R-3.4.1. Tech. rep., Max-Planck-Institut für Informatik, 1996.

[20] Nievergelt, J., and Widmayer, P. Spatial data structures: concepts, design, and choices. In *Handbook of Computational Geometry*, J. Urrutia and J.-R. Sack, Eds. Elsevier Sciences, to appear.

[21] Peucker, T. K. Data structures for digital terrain models. *Harvard Papers on Geographic Information Systems 5* (1978).

[22] Peucker, T. K., Fowler, R. J., Little, J. J., and Mark, D. M. The triangulated irregular network. In *Proceedings DTM Symposium American Society of Photogrammatry - Amercian Congress on Survery and Mapping* (1978), pp. 24–31.

[23] Samet, H. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[24] Snoeyink, J., and van Kreveld, M. Linear-time reconstruction of Delaunay triangulations with applications. In *ESA '97: European Symposium on Algorithms* (1997).

[25] van Kreveld, M. Efficient algorithms for isoline extraction from a TIN. *Int. Journal of GIS 10* (1996), 523–540.