Compressing Triangulated Irregular Networks

Leila De Floriani, Paola Magillo, Enrico Puppo
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova
Via Dodecaneso, 35 – 16146 Genova (Italy)

Abstract.
We address the problem of designing compact data structures for encoding a Triangulated Irregular Network (TIN). In particular, we study the problem of compressing connectivity, i.e., the information describing the topological structure of the TIN, and we propose two new compression methods which have different purposes. The goal of the first method is to minimize the number of bits needed to encode connectivity information: it encodes each vertex once, and at most two bits of connectivity information for each edge of a TIN; algorithms for coding and decoding the corresponding bitstream are simple and efficient. A practical evaluation shows compression rates of about 4.2 bits per vertex, which are comparable with those achieved by more complex methods. The second method compresses a TIN at progressive levels of detail and it is based on a strategy which iteratively removes a vertex from a TIN according to an error-based criterion. Encoding and decoding algorithms are presented and compared with other approaches to progressive compression. Our method can encode more general types of triangulations, such as those constrained by topographic features, at the cost of a slightly longer bitstream.

Keywords: Triangulated Irregular Networks, Geometric Compression, Terrain Modeling, Data Structures

1. Introduction

Huge terrain datasets are increasingly available in GIS. This fact gives rise to challenging problems in storage, transmission, and visualization of terrain models. In particular, the design of compact structures for encoding a terrain model as a *sequential bitstream* is important to reduce the amount of time needed to transmit it over a communication line (e.g., through Internet), to save disk storage space or to load a terrain from disk into memory.

While regular grids can be compressed through techniques similar to those used for compressing images [16], this is not true for Triangulated Irregular Networks (TINs). For TINs, the problem is different and more involved. While the connecting structure of a grid is fixed, the triangulation at the basis of a TIN is not uniquely defined by its vertices, and thus it must be encoded explicitly. Such triangulation is usually the result of some specialized algorithm which takes into account issues such as preservation of terrain features, interpolation of isolines, minimization of the approximation error, etc., and typically uses extra knowledge beside the position of vertices. Thus, it is not reasonable to encode just the vertices of a TIN, and recompute the underlying triangulation each time the TIN is loaded or received.

The study of techniques for producing compressed formats of triangle meshes (and, in particular, of TINs), generally referred to as *geometry compression*, has gained increasing attention in the last few years because of the widespread use of TINs as terrain models.

The problem of TIN compression involves two complementary tasks, which can be studied independently:

- *Compression of the vertices*, i.e., of the numerical information attached to each vertex of the TIN (location, elevation and, possibly, attributes).
- *Compression of connectivity*, i.e., of information describing the triangles of the TIN as triplets of vertices, and of adjacency information between pairs of triangles.

Compression of vertices and of connectivity involve different problems and different techniques. Thus, they can be treated separately. Vertex compression has been treated by some authors [3, 8, 18] by using combinations of lossy methods based on quantization and lossless methods based on entropy encoding.

In this paper, we do not address vertex compression, while we focus on compression schemes for TIN connectivity. We propose two compression methods. The first method is based on traversing a TIN in a *shelling order* (i.e., radially around a seed triangle); it encodes each vertex exactly once, and guarantees less than two control bits for each edge to encode connectivity. The second method compresses a TIN at progressive levels of detail and is based on a strategy which iteratively removes a vertex from a TIN according to an error-based criterion. Both methods encode both the triangles of a TIN and their adjacency relations directly in the bitstream.

This paper is organized as follows: in Section 2, we provide a brief survey of existing techniques for compressing connectivity; in Section 3, we describe our compression method based on shelling; in Section 4, we present our method for progressive compression. Section 5 contains some concluding remarks.

2. Previous Work

Our review of existing literature is focused on methods for compression of connectivity. We do not address here methods for vertex compression.

A TIN is usually maintained (both on disk and in main memory) in an *indexed format*: a list of vertices and a list of triangles are maintained and the connecting structure of the TIN is described by providing three vertex references for each triangle. Since the number of triangles in a TIN is roughly twice the number of vertices and since a vertex reference requires $\log n$ bits for a TIN with n vertices, such a scheme requires at least $6n \log n$ bits. The indexed format can be extended to encode triangle-to-triangle adjacencies by adding, for each triangle, the references to its three adjacent triangles. The total number of bits in this case raises to $12n \log n + 6n$. In the following we will refer to such structure as the *explicit encoding* for a TIN, and we will use it as a reference for evaluating the performance of compression methods.

Techniques for *compression of connectivity information* of a TIN can be roughly classified into:

- *Direct methods*, which are aimed at minimizing the number of bits needed to encode a given TIN "as it is". The bitstream must be read and processed completely by a decompression algorithm in order to obtain a description of the TIN.

- *Progressive methods*, which encode a coarse TIN representation and a sequence of details which must be added to it in order to recover the initial TIN. In this case, an interrupted bitstream provides an *approximation* of the terrain at a lower resolution.

Several compression methods proposed in the literature are not restricted to TINs but they have been developed for triangle meshes representing free-form surfaces in three-dimensional space. Thus, in our literature survey we sometimes refer to triangle meshes rather than to TINs.

2.1. Direct Methods

Classical direct methods proposed in the literature are based on the decomposition of a triangle mesh into *triangle strips*, or generalizations of triangle strips [3, 8, 9]. Such methods are especially well-suited for visualization, but less to TIN transmission or encoding on disk because they do not provide the full mesh topology.

A *triangle strip* is a path of triangles with alternated left and right turns; it is simply specified by listing the vertices of its triangles in a

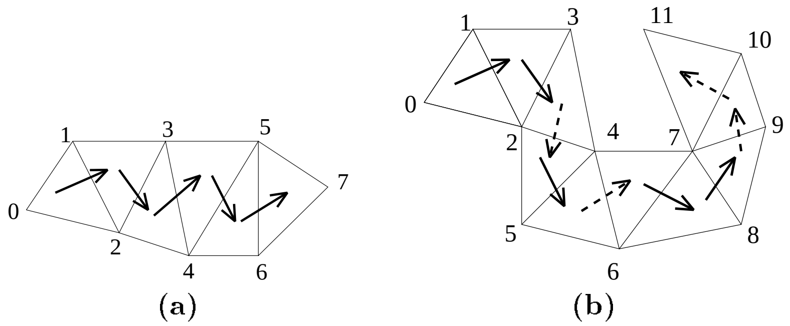


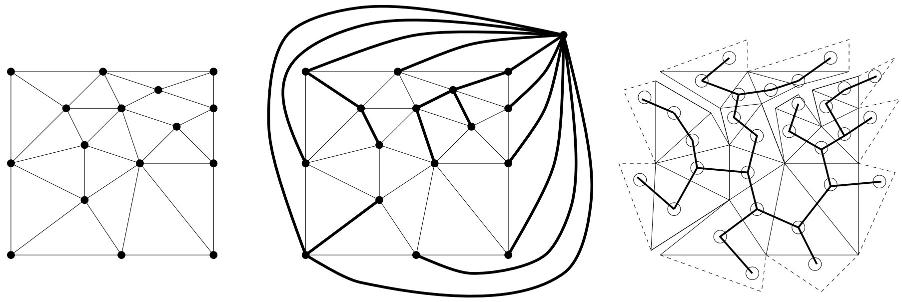
Figure 1. (a) A triangle strip; the corresponding vertex sequence is $0, 1, 2, 3, 4, 5, 6, 7$. (b) A generalized triangle strip; dashed arrow correspond to turns in the same direction as the previous one in the sequence; the correseponding vertex sequence with vertex replication is $0, 1, 2, 3, 4, 2, 5, 4, 6, 7, 8, 9, 7, 10, 11$.

zig-zag order (see Figure 1 (a)); each vertex appears only once in the sequence. A drawback of this simple structure is that a triangle mesh in general cannot be decomposed into strips with alternating left and right turns [9] .

A more general form of strip is given by so-called *generalized triangle strips*, which represent standard technology in graphics packages. In this case we do not have a zig-zag sequence, but each new vertex may correspond either to a right or to a left turn in the pattern (see Figure 1 (b)). The encoding bitstream either uses a binary code, called a swapcode, to specify how to connect each new vertex to the last encoded triangle, or it repeats the vertex in case of two successive turns with the same orientation occur (see Figure 1 (b)). A drawback is that several strips are necessary to cover a triangle mesh and vertices shared by more than one strip must be replicated.

On average each vertex appears twice in the resulting bitstream. It has been shown [9] that the problem of generating a minimum number of strips is NP-complete. Evans et al. [9] give several heuristics in which they try to maximize the number of triangles per strip, to minimize swaps, to avoid strips formed by just one triangle.

Other methods [1, 8] also allow reusing more of the past vertices than the last two ones (as in generalized triangle strips), by storing the last k vertices in a buffer; small codes indicating how to operate on such a buffer are interleaved in the bitstream. Deering [8] proposes a buffer of 16 positions and a syntax for the encoding, giving rise to the so-called *generalized meshes*. Bar-Yehuda and Gotsman [1] studied the extent to which we can increase the buffer size to reduce vertex duplication. They also show that the problem of minimizing the buffer size for a given mesh is NP-hard. Chow [3] gave algorithms for building generalized



(a) (b) (c)

Figure 2. (a) A TIN; (b) the same TIN with a vertex at infinity added, and the edges of the vertex spanning tree superimposed as thick lines; (c) the triangle spanning tree (the dashed lines denote triangles incident in the vertex at infinity).

meshes which try to optimize the use of the buffer through heuristics. Compression rates are about 11 bits of connectivity information per vertex in [3, 8] and from 2.5 to 7 in [9].

Methods based on triangle strips and their variants provide just a description of triangles and not of their mutual topological relations. Thus, if the decoded TIN must undergo operations that require adjacency relations (e.g., computation of visibility information, paths, drainage networks), these should be reconstructed separately with an extra cost. Other methods allow reconstruction of adjacency relations from the encoding bitstream [7, 19]. Our method presented in Section 3 belongs to this second class.

Topological surgery by Taubin and Rossignac [18] cuts a triangle mesh along a set of edges that correspond to a spanning tree of vertices; such tree is determined by selecting the minimum number of edges which connect all the vertices. If we imagine cutting the given TIN along the edges of the vertex spanning tree, we obtain another TIN which contains the same triangles as the original one, and whose dual graph is structured as tree, thus forming a spanning tree of triangles of the original TIN (see Figure 2).

The triangle spanning tree can be decomposed into a collection of triangle strips, called *runs*. Runs are connected through branching triangles, each of which connects three runs.

The bitstream produced by the method consists of the vertex and of the triangle spanning trees. A rather complex algorithm permits to reconstruct the original TIN. Through experiments on manifold meshes Taubin and Rossignac show a compression of 4 bits per vertex on average. The method produces longer strips than the previously discussed ones; it requires that all vertices are kept in main memory during both compression and decompression; both the coding and the decoding algorithms perform consistent graph manipulation and are not easy to implement.

The method by Touma and Gotsman [19], unlike the previous ones, *encodes triangle adjacencies* directly in the bitstream. The compression algorithm maintains an active list of edges bounding a (not necessarily simply-connected) polygon which separates triangles already encoded (located inside the polygon) from triangles that have still to be to coded (located outside). At each step, the polygon is enlarged by encoding all triangles lying outside it which are incident in one of its vertices. The decompression algorithm works in a completely similar way.

The resulting bitstream contains each vertex once, and three types of commands: ADD, SPLIT and MERGE, where the first two commands have one argument and the third command has two arguments. The argument of ADD is the degree d of a new vertex to be inserted (i.e., the number of triangles incident in such vertex), the arguments of SPLIT and MERGE are integer numbers with the meaning of indexes and offsets; such commands mark those points at which the number of boundary curves of the current polygon increases by splitting one curve into two, or decreases by merging two curves into one, respectively. With the exception of MERGE operations (which are rare), the compression and the decompression algorithms run in linear time in the number of mesh edges. Details about the complexity of MERGE are not provided in [19].

The method generates an ADD command for each vertex of the mesh. The number of SPLIT and MERGE commands depends on the mesh. The authors do not report any evaluation of the number of SPLIT or MERGE commands that can occur: they just observe that MERGE commands can occur only for meshes of positive genus (i.e., three-dimensional surfaces with through holes), and thus not for terrains.

If we assume that the degree of any vertex is less than b, the compressed sequence requires at least $2 + \log b$ bits per vertex: two bits for the command (since we have three possible commands) and $\log b$ bits for the degree: for instance, 5 and 6 bits, respectively, if $b = 8$ or if $b = 16$. The bits necessary to represent possible SPLIT and MERGE commands add to these. Touma and Gotsman observe that the sequence of codes can be compressed further. Since the average degree of a vertex is 6, there is a spread of degrees around this value; moreover, there are easily long subsequence repeating the same commands and the same arguments. Thus, they use entropy coding combined with run-length encoding, getting experimental compression rates from 0.2 to 2.4 bits per vertex for connectivity.

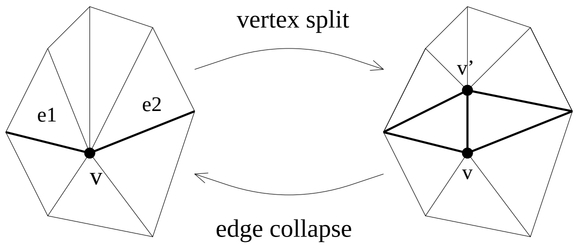


Figure 3. Edge collapse and its reverse vertex split.

2.2. Progressive Methods

Progressive methods encode a coarse approximation of a mesh and a sequence of modifications which incrementally refine it into the original mesh [12, 13, 17]. An interrupted bitstream provides an approximation of the whole terrain, which can be seen as a form of *lossy* compression. Progressive compression methods are especially useful for mesh transmission or as compact formats for secondary storage since they allow a trade-off between loading/transmission time and loss of accuracy. This additional feature is paid with compression rates that are usually lower than those of direct methods.

The basic technique for building progressive representations of meshes come from iterative techniques for mesh simplification based on *local modifications* of a mesh (see, e.g., [14, 11]).

A sequence of progressive resolutions for a mesh is generated by iteratively applying a *destructive operator*, which removes details somewhere in the mesh. A destructive operator has an inverse *constructive operator*, which recovers such details. A compressed bitstream contains the coarse mesh (i.e., the result of a sequence of destructive operations) along with the sequence of constructive operations necessary to recover the original mesh.

Progressive Meshes, developed by Hoppe [12, 13], provide a progressive compression method where the destructive operator is *edge collapse*. Edge collapse replaces an edge e with one of its endpoints v, and the two triangles sharing e with two edges incident at v. The corresponding constructive operator is *vertex split*, which expands a vertex v into an edge $e = vv'$ and two edges e_1 and e_2 among those incident at v into two triangles (see Figure 3). Each vertex split in the sequence is compactly encoded by providing the new vertex v', a reference to an old vertex v, and a code specifying the position of e_1 and e_2 around v. The number of bits necessary to encode connectivity is $n(\log n + \log(b(b-1)))$ for a mesh of n vertices, if b is the maximum degree of a vertex, since the position of e_1 is a number in the range $0 \ldots b$ and that of e_2 a number in the range $0 \ldots b-1$ (as one position has already been chosen for e_1).

For instance, if $n = 2^{16}$, and $b = 8$ ($b = 16$), we have 21 ($23n$) bits of connectivity information.

Snoeyink and Van Kreveld [17] have developed a progressive method which applies to TINs based on Delaunay triangulations, and guarantees that all intermediate meshes are Delaunay triangulations. The destructive operator removes a vertex from a Delaunay triangulation; the corresponding constructive operator re-inserts such vertex. The compression algorithm performs several stages starting from the given mesh: each stage removes a maximal set of independent vertices (i.e. vertices not connected by an edge) from the current Delaunay triangulation. The process is guaranteed to terminate in a logarithmic number of stages. The bitstream contains the final coarse mesh plus, for each stage in reverse order, the sequence of removed vertices (which must be re-inserted by the decoding algorithm). The sequence of vertices removed in a stage is sorted in the same order in which the triangle containing each vertex is encountered in the mesh traversal algorithm of de Berg et al. [4]. Thus, the decoding algorithm at each stage can locate all the triangles containing the new vertices in linear time in the size of the current triangulation. The length of the resulting bitstream is $O(\log^2 n)$ where n is the number of vertices and the constant is not known. A disadvantage of this method is in requiring heavy numerical calculations (i.e., the incremental reconstruction of a Delaunay triangulation) for decoding, which may cause not only time overhead, but also numerical and inconsistency problems (e.g., different triangulation of co-circular points in the original and in the reconstructed mesh). No experimental results are available.

Note that both methods described above assume a specific strategy for retriangulating the mesh after the insertion of each vertex (i.e., vertex split in [12, 13], and Delaunay update in [17]). In Section 4, we propose a progressive method which does not depend on the retriangulation strategy.

3. A Direct Compression Method Based on Shelling

In this section we describe a new compression method based on shelling. This method is designed for mesh transmission or disk storage because it encodes both triangles and their adjacency relations.

A *shelling sequence* of a TIN Σ is a sequence $\sigma_1, \sigma_2, \ldots, \sigma_n$ of triangles, containing each triangle of Σ once, and such that, for any prefix subsequence $\sigma_1, \sigma_2, \ldots, \sigma_m$ (with $m \leq n$), the boundary of the union of the set of triangles $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is a simple polygon. A mesh is *shellable* if it admits a shelling sequence; it is *extendably shellable* if it is *shellable* and every shelling sequence defined for a subset of its triangles can be extended to a shelling sequence for the whole mesh. Extendably shellable meshes include TINs and all the triangulations of surfaces homeomorphic to a sphere or a disk; surfaces with a non-null genus (i.e., with through holes) are not extendably shellable [2]. The method we present here applies to triangle meshes that are *extendably shellable*.

The bitstream implicitly encodes a shelling sequence of the given TIN. It contains the vertices of the TIN interleaved with control codes providing connectivity information. There are four control codes: VERTEX, SKIP, LEFT, and RIGHT, each represented on two bits. Coding and decoding algorithms are very simple and work in time linear in the size of the TIN without performing any numerical computation. Both the triangles and their adjacency links are recovered from the bitstream without extra processing. Moreover, only the boundary of the current polygon and triangles incident to its edges have to be kept in memory, while compressed / decompressed triangles lying completely inside the polygon can be discarded.

The method starts from an arbitrary triangle and traverses the TIN according to a shelling order. At each iteration, it produces a two-bit code and, possibly, a vertex. Each vertex of the TIN is encoded exactly once.

3.1. Compression and Decompression Algorithms

The compression and decompression algorithms maintain the *simple polygon* bounding the currently encoded / decoded portion of the triangulation in a doubly-linked list; they also keep a *queue of active edges*, which are those edges of the current boundary that have not been tested yet to compress / decompress the triangle externally adjacent to them; a control code is sent/read for each active edge.

The compression algorithm starts from an arbitrary triangle of the TIN, whose edges define the initial polygon, and sends its three vertices to the output stream. Then, it examines each edge e of the current polygon in turn and tries to enlarge the polygon by including the triangle σ externally adjacent to e. It performs the following actions on e:

1. if the third vertex v of σ does not lie on the polygon, then include σ and send a VERTEX code followed by vertex v (see Figure 4 (a));

2. if σ does not exist (i.e., edge e is on the boundary of the TIN), or the third vertex v of σ lies on the polygon, and it is not the other vertex of one of the edges adjacent to e along the polygon (i.e., σ cannot be

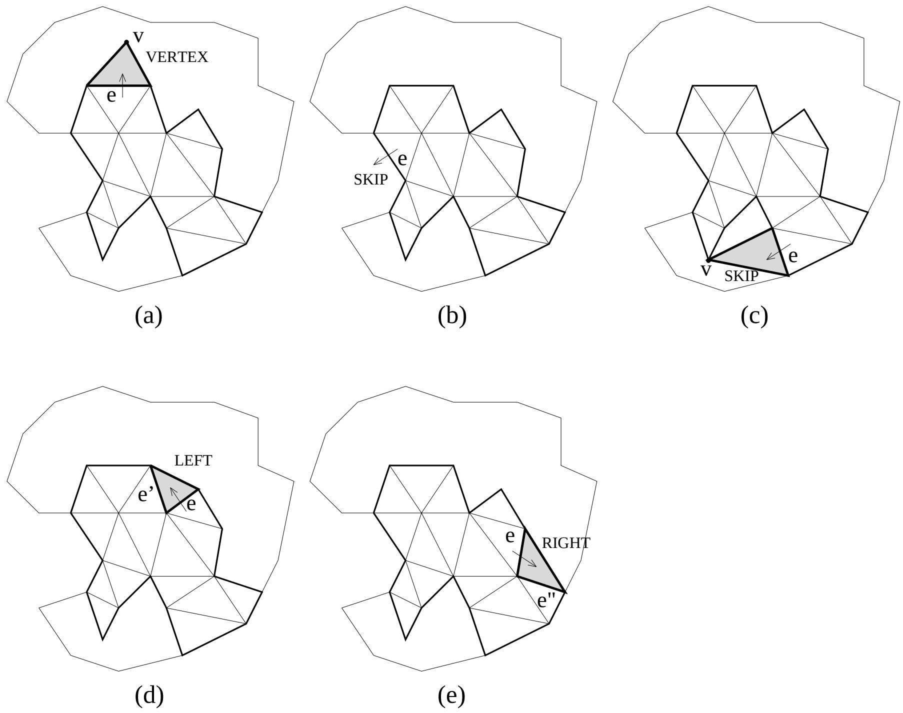


Figure 4. The five possible situations for the current edge e in the TIN compression algorithm: (a) the triangle externally adjacent to e brings a new vertex V; (b) the triangle externally adjacent to e does not exist; (c) the triangle externally adjacent to e exists but cannot be included because it violates shelling; (d,e) the triangle externally adjacent to e has another edge on the current polygon, and such edge is the one lying to the left and to to right of e, respectively.

included since it makes the polygon not simple), then send a SKIP code (see Figure 4 (b) and (c)); edge e will not be examined again;

3. if the third vertex v of σ lies on the polygon, and it is the other vertex of the edge adjacent to the left of e, then include σ and send a LEFT code (see Figure 4 (d));

4. if the third vertex v of σ lies on the polygon, and it is the other vertex of the edge adjacent to the right of e, then include σ and send a RIGHT code (see Figure 4 (e)).

The loop terminates when all the triangles have been compressed.

The decoding algorithm works in a similar way. It builds the first triangle from the three vertices found at the beginning of the bitstream. Then, it examines each edge e of the current polygon in turn, and tries to enlarge the current polygon by creating a new triangle adjacent to e, according to the directions contained in the next control code read from the bitstream:

1. in case of a VERTEX code: read a vertex v from the bitstream, create a new triangle σ from e and v, and set an adjacency link between σ and the triangle adjacent to the other side of e;

2. in case of a SKIP code: no new triangle is created; edge e will not be examined again;

3. in case of a LEFT code: create a new triangle σ from e and the edge e' adjacent to the left of e along the polygon; set adjacency links between σ and the triangles adjacent to the other side of e and e';

4. in case of a RIGHT code: create a new triangle σ from e and the edge e' adjacent to the right of e along the polygon; set adjacency links between σ and the triangles adjacent to the other side of e and e'.

The loop terminates when the bitstream has been read completely.

3.2. Analysis

Since a TIN is extendably shellable, at each step one of the following two conditions occurs: either all triangles have been processed, or there exists a triangle which can be added to the current polygon while maintaining it simple; at least one edge of such a triangle must lie on the current polygon. In the first case, the process terminates. In the latter case we show that the compression algorithm can find a new triangle to encode.

Each step of the main loop examinates an edge e of the current polygon until it finds one such that the triangle externally adjacent to it can be included. It is easy to see that, if e generates a triangle (cases VERTEX, LEFT, and RIGHT) then such triangle preserves the simplicity of the current polygon and, if e does not generate a triangle (case SKIP) then either e is on the TIN boundary, or the triangle t_e externally adjacent to e does not preserve simplicity because the vertex of t_e opposite to e belongs to the boundary of the current polygon. Edges of this last type will not be examined again (they are "virtually removed" from the current polygon). Therefore, the algorithm will scan edges of the current polygon until a non-SKIP edge is found. Since the triangulation is extendably shellable, we are always sure that one such edge exists until the whole triangulation has been traversed.

It remains to be shown that a triangle t_e externally adjacent to a SKIP edge e at some stage will be generated at some later stage. Since all three vertices of t_e already belong to the current polygon, it will be possible to include t_e only after having another edge e' of t_e on the polygon. Let t' be the neighbor of t_e along e'. Since we are following a shelling order, t', hence e', will be included in the current polygon at some time. Thus, while examining e', either a LEFT or a RIGHT code will be sent, and t_e will be included.

Thus, we have shown that the compression algorithm always terminates, after having traversed all the triangles of the given TIN in shelling order.

At each step, the number of operations performed on the current edge is bounded by a constant. Each TIN edge is examined at most once in the main loop, hence the time complexity for both compression and decompression is linear in the number of edges, which is about $3n$, i.e., $O(n)$.

It is easy to see that the actions performed by the decompression algorithm, on each code read from the bitstream, reproduce exactly the same triangle configuration from which that code has been generated by the compression algorithm. Thus, the original TIN is reconstructed correctly.

Each vertex is sent exactly once by the compression algorithm, and a code is sent for every examined edge, i.e., *at most one* code for each TIN edge. Indeed, not all edges are examined: LEFT and RIGHT codes eliminate one edge each, which therefore does not generate a code; moreover, the vertices of the boundary remaining after having processed all the triangles do not generate any code.

Since the number of VERTEX codes is fixed, the bitstream is shorter when there are many LEFT and RIGHT codes and few SKIP codes. Implementing the list of active edges as a queue results in a polygon that tends to grow in a breadth first order, thus providing many LEFT and RIGHT codes. We found that an implementation based on a stack, hence on a depth first order, produces much more SKIP codes, and, thus, longer bitstreams.

In the worst case, the number of control codes is bounded from above by the number of edges in the given TIN, i.e., about $3n$ (equivalent to $6n$ bits), where n is the number of TIN vertices. Experimental results give better compression rates (see Section 3.3).

3.3. Experiments and Discussion

Our compression method takes into account only the connectivity structure of a TIN, not its geometry. Thus, in order to prove the performance of our approach on a wide range of inputs, we show experiments on TINs having different kinds and sizes of triangulations, rather than on TINs representing different terrains.

Table I. Results of the compression algorithm based on shelling. U1–4 are TINs at uniform accuracy, A1–4 and B1–4 are TINs at variable accuracy. Times are in seconds.

TIN	vert	tri	code bits	bits / vert	encoding time (tri/sec)	decoding time (tri/sec)
U1	42943	85290	182674	4.2538	3.82 (22327)	2.33 (36605)
U2	28510	56540	123086	4.3173	2.53 (22347)	1.55 (36477)
U3	13057	25818	57316	4.3897	1.15 (22450)	0.71 (36363)
U4	6221	12240	27180	4.3690	0.56 (21857)	0.33 (37090)
A1	15389	30566	64678	4.2029	1.34 (22810)	0.83 (36826)
A2	15233	30235	63958	4.1986	1.36 (22231)	0.82 (36871)
A3	15515	30818	65210	4.2030	1.36 (22660)	0.84 (36688)
A4	15624	31042	65520	4.1935	1.39 (22332)	0.85 (36520)
B1	5297	10570	22392	4.2273	0.46 (22978)	0.28 (37750)
B2	5494	10959	23468	4.2716	0.49 (22365)	0.30 (36530)
B3	5397	10768	23060	4.2727	0.47 (22910)	0.29 (37131)
B4	5449	10874	23136	4.2459	0.48 (22654)	0.29 (37496)

All TINs used here are approximated models built by a surface simplification algorithm from an original grid dataset representing the area of Mt. Marcy (NY), courtesy of U.S. Geological Survey. Data U1–4 are TINs approximating original data at a uniform error value, equal to 1, 2, 5 and 10 units, respectively; all triangles in a single TIN have about the same area. Conversely, TINs A1–4 and B1–4 are characterized by a great difference in triangle density of different zones: in A1–4 (B1–4), one fourth (one 16th) of the area is represented with an elevation error equal to 0.5 meters, while the rest is represented at the coarsest possible detail.

Experiments have been performed on a Pentium Pro (200 Mhz, 64 Mb RAM). Results are shown in Table I. Compression rates are lower than 4.5 bits per vertex. Execution times are about 22K and 37K triangles per second in compression and decompression, respectively.

The main advantages of our method are those of being very simple conceptually, and easy to implement, and at the same time, of giving good compression rates.

The method is as simple as (generalized) triangle strips [3, 8, 9] and achieves higher compression rates, comparable to those of more complicated algorithms [18]; moreover, such algorithms do not provide adjacency information.

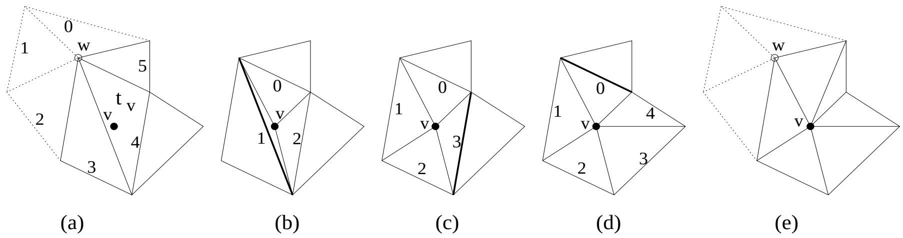


Figure 5. The triangulation after (a) and before (e) removing a vertex v, and the related sequence of edge flips (b,c,d), to be read left-to-right to restore situation (e) from situation (a). In (a), the edges around w, and in (b,c,d) the edges around v are numbered. Triangle t_v is encoded as the pair $(w, 4)$ and the sequence of edge flips is 1,3,0.

The algorithm proposed in [19] also encodes adjacency information, and is more complicated than ours since it manages a multiply-connected polygon instead of a simple one. The bitstream generated by such method (disregarding the postprocessing step that is applied in [19] to further compress it) requires at least $5n$ or $6n$ bits (depending on the maximum degree of a vertex in the TIN), plus a variable number of SPLIT and MERGE codes, which depends on the input mesh. Thus, the length of the two bitstreams is comparable on average.

4. A Progressive Compression Method based on Edge-Flips

In this section we present a new method for TIN compression at progressive levels of detail. We use a destructive operator which consists of removing a vertex v of degree bounded by a constant b from the current TIN and of retriangulating the corresponding polygonal hole π. The inverse constructive operator inserts vertex v and recovers the previous triangulation of π. We recall that the degree of a vertex is the number of triangles incident in it.

The old triangulation is recovered from the new one by first splitting the triangle t_v, containing v, in the current TIN, into three triangles, and then applying a sequence of *edge flips*. An edge flip is an operation that replaces an edge, shared by two triangles whose union forms a convex polygon, with the other diagonal of such polygon.

Our bitstream encodes, for each removed vertex v, both the starting triangle t_v and the sequence of edge flips to be performed (see Figure 5):

- The starting triangle t_v is given by a reference to a vertex w of t_v and by the index of t_v in the star of triangles incident at w. The reference to w requires $\log n$ bits; the index of t_v requires $\log b$ bits.

– A flip operation is specified by an edge opposite to v (in its star of triangles), which is involved in the flipping operation. If triangles incident at v are numbered in a conventional order in the data structure encoding the TIN, such edge can be uniquely identified with an index. Note that there are three edges incident at v at the beginning and their number increases by one at each flip: there will be $3+i$ incident edges after i flips, and $k-1$ incident edges before the last flip, where k is the degree of the removed vertex ($k \leq b$).

  Therefore, for $k > 3$, the whole sequence of edge flips is identified by a sequence of $k-3$ indices $s_0, \ldots, s_{k-4}$, where s_i is in the range $[0, 3+i]$. It is easy to see that for a removed vertex of degree k, there exist $3 \cdot 4 \cdots (k-1) = \frac{(k-1)!}{2}$ possible sequences of flip indices. A given sequence can be encoded compactly with a unique number in the range $[0, \frac{(k-1)!}{2}]$, called a *flip code*, which requires $\lceil \log(\frac{(k-1)!}{2}) \rceil$ bits and is given by the formula:

$$C \equiv \sum_{i=0}^{k-4} s_i \frac{(i+2)!}{2}.$$

  This flip code can be unpacked with a standard method: s_{k-4} is given by $c \operatorname{div} \frac{(k-2)!}{2}$; for a generic i,

$$s_i = (c - \sum_{j=i+1}^{k-4} s_j \frac{(j+2)!}{2}) \operatorname{div} \frac{(i+2)!}{2}$$

  where div denotes the integer division operation.

Our bitstream encodes each removed vertex once and, for each vertex, a vertex reference, a triangle index, the number of flips to be performed and a flip code, as described above. Disregarding the cost of the initial TIN, which is always very small, our method produces $n(\lceil \log n \rceil + 2\lceil \log b \rceil) + \sum_{k=3}^{b} (n_k \lceil \log(\frac{(k-1)!}{2}) \rceil)$ bits of connectivity information, where n_k is the number of removed vertices of degree k. Since, on average, the degree of a vertex in a TIN is six, this can be roughly estimated as $n(\lceil \log n \rceil + 2\lceil \log b \rceil + 6)$. Experiments we made on real data confirm such an estimate.

4.1. Compression and Decompression Algorithms

The *compression algorithm* works as follows:

1. Initialize the current TIN as the input TIN.

2. While the current TIN is not "small enough" select a vertex v of degree at most b according to some criterion (to be explained next):

   – remove v and retriangulate the polygonal hole π;
   – locate the triangle t_v of the new triangulation of π which contains v;
   – choose a vertex w of t_v, having degree at most equal to b and compute the index of t_v around w;
   – simulate the reconstruction of the old triangulation of the hole (the one containing v) from the current one (the one without v), and encode the corresponding sequence of edge flips.

3. Send the current TIN, in a directly compressed format (e.g., as explained in Section 3), followed by the sequence of removed vertices in reverse order; for each vertex v, output also the index of the associated vertex w, the index of t_v in the set of triangles incident in w, the number of flips to be performed and the flip code.

The compression algorithm has two key points: the choice of the vertex v to be removed at each iteration of loop 2, and the retriangulation of the corresponding hole π. *Any criterion* can be used for both purposes, without affecting the format of the bitstream, or the decompression algorithm. We will discuss such issues further in Section 4.2.

The *decompression algorithm* performs the following steps:

1. Initialize the current triangulation with the coarse triangulation read from the bitstream.

2. Read the next vertex v with associated information. Get vertex w from the list of vertices in the current TIN and retrieve triangle t_v in the star of w. Insert v into the current triangulation by splitting triangle t_v at v; unpack the flip code and perform the given sequence of edge flips.

3. The process can be stopped either when the bitstream is finished (in this case the original TIN is completely reconstructed, and we have a lossless decompression), or when the current TIN has a given number vertices, or when a certain time has elapsed (in the two latter cases, a TIN approximating the original one is built, and we have a lossy decompression).

During both compression and decompression, the current TIN is maintained in a data structure corresponding to the indexed format with triangle-to-triangle adjacencies (see Section 2), augmented with partial vertex-to-triangle incidence information: for each vertex w in the TIN, a link to one of its incident triangles is maintained. Such a data structure is updated according to fixed conventional rules, which allow us to obtain exactly the same indexing scheme during both compression and decompression. This is fundamental to uniquely identify, at each step, triangle t_v in the star of vertex w and each edge to be flipped through the corresponding flip index.

4.2. Analysis

The correctness of the method follows from basic properties of triangulations well known from computational geometry:

1. if we remove a vertex, together with all its adjacent triangles, from the interior of a plane triangulation, we make a star-shaped polygonal hole that can be always filled with triangles without adding extra vertices (we have a slightly different situation if the vertex is removed from the boundary of the triangulation);

2. given a plane triangulation T and a new vertex v, if t_v is the triangle of T containing v, then we can obtain a new triangulation by replacing t_v with three new triangles incident at v (triangle split);

3. any triangulation of a given set V of vertices can be transformed into any other triangulation of the same set of vertices through a sequence of edge flips;

4. each edge flip applied to a boundary edge of the star of a vertex v increases the degree of v by one unit.

Property 1 ensures that at each step of the compression algorithm we obtain a new (simplified) triangulation. Let T be the current triangulation at a generic step, and T' be the triangulation obtained by eliminating vertex v from T. Property 2 ensures that locating t_v in T' provides sufficient information to obtain a new triangulation T'' containing v with degree three, while property 3 ansures that T can be obtained from T'' through a sequence of edge flips. Since T differs from T'' only in the portion corresponding to the hole left by removing v from T, then edge flips will affect only that portion. Each edge flip given by a flip index introduces an edge incident at v which belongs to T, and such an edge is never removed by a subsequent flip, because each flip affects an edge on the boundary of the current star of v. Hence, if the degree of v in T is k, then T will be recovered by exactly $k-3$ flip operations, which are specified by the flip code.

The core of the compression algorithm is the main loop in Step 2. As already mentioned, we have two degrees of freedom: in the selection of the vertex v to be removed and in the retriangulation of the resulting hole.

Any criterion can be used to retriangulate the hole π. Since π has a bounded number b of edges, retriangulating it requires a constant number of operations in any case. For instance, an *ear cutting* procedure might be applied, which corrsponds to iteratively flipping edges incident at v, until just three incident edges are left. Cutting an ear of π corresponds to flipping an edge incident at v, thus the sequence of edge flips to be encoded is provided directly. Otherwise, no matter what algorithm has been used to triangulate π, the sequence of flips can be obtained as follows. Let T_π be the triangulation of π. Vertex v is located first in T_π and triangle t_v is found. An influence polygon is initialized with t_v, and its edges are scanned. Each time an edge e is found, which does not lie on the boundary of T_π, then e is deleted and the influence polygon grows by incorporating the triangle externally adjacent to e. The index of e is encoded as a split index. This process continues until the influence polygon covers the whole π (i.e., all edges are on the boundary of T_π).

Any criterion can also be used for selecting the vertex to be removed, under the constraint that its degree is not higher than b. For a sufficiently large value of b (e.g., $b = 16$) this constraint has no impact on real world cases. The specific implementation choice affects both the complexity of the compression algorithm and the efficiency of the resulting bitstream for the purpose of progressive transmission. We suggest to select the vertex causing the least increase in the approximation error (as it was also done by other authors, e.g., [14, 11]). This requires that the error caused by removing a vertex of the current TIN is known at each iteration.

The approximation error of the current TIN is the maximum approximation error of its triangles. For each triangle t, its approximation error is equal to the maximum vertical distance of a removed vertex, whose vertical projection falls in t, from the plane of t. In order to compute approximation errors of triangles we associate with each triangle t in the current triangulation the set of vertices already deleted whose vertical projections fall in t.

The error introduced by removing a vertex v is evaluated by simulating the deletion of v: the hole left by v is retriangulated and the error is measured by locating v and all points lying inside its incident triangles with respect to the new triangles that fill the hole, computing the errors of such triangles, and choosing the maximum error. The error of each vertex v is stored together with v, and it is updated whenever a vertex adjacent to v is removed. Note that it is sufficient to evaluate the error only at vertices whose degree is smaller than b, because they are the only

Table II. General information and theoretical bound for the three TINs considered in our experiments.

terrain	#vertices	$\log_2 n$	$\log_2 b$	bits/vert (theor.)
Devil Peak	38115	16	4	30
Mount Marcy	16384	14	4	28
San Bernardino	16384	14	4	28

candidates for removal. The triangulation of a hole, necessary to evaluate the error, is performed only at vertices of bounded degree, hence it takes constant time at each vertex. Vertices of bounded degree are maintained in a priority queue based on their errors. At each deletion, error must be re-evaluated only for a constant number of vertices, i.e., the neighbors of v. Location of points inside a given triangle t occurs at most four times, namely for evaluating the cost of removing each of its three vertices, and when one of such vertices is actually removed (when t is removed as well). During the whole coarsening process, a removed vertex may fall, and need to be relocated, in $O(n)$ different triangles in the worst case. However, following [10], it is easy to show through amortized analysis that the expected cost of point location is still $O(\log n)$. It follows that the total cost for computing approximation errors is $O(n \log n)$. This equals the cost of managing the priority queue, since each operation on the queue costs $O(\log n)$, and a constant number of operations are performed for each removed vertex. Therefore, the total time complexity of the compression algorithm in this case is $O(n \log n)$.

The selection criterion affects the quality of the intermediate triangulations obtained from an interrupted bitstream (i.e., the ratio between accuracy and number of vertices used), and thus the efficiency of the sequence for *progressive transmission*. In other words, if the bitstream is interrupted after k vertices, the approximation error of the currently reconstructed TIN is either larger or smaller, depending on the policy used within the compression algorithm for selecting vertices that must be removed. One can trade-off the complexity of the selection procedure with the quality of the compressed sequence.

Other possible techniques for vertex selection, different from the one described above, have been discussed and experimented in [5]. Such alternative methods include the selection of a maximal set of independent vertices of bounded degree at each time, still based on an error-driven criterion, and the random selection of a maximal set of independent vertices (which runs in $O(n)$, but gives poor quality of intermediate triangulations in the bitstream).

Table III. Length of the bitstream with the three encoding algorithms.

terrain	algo	#bits	bits/vert	max deg	avg deg
Devil Peak	IR	1127k	29.58	11	5.64
Devil Peak	IE	1125k	29.53	11	5.62
Devil Peak	SE	1159k	30.42	11	5.97
Mount Marcy	IR	450k	27.51	10	5.61
Mount Marcy	IE	449k	27.43	11	5.58
Mount Marcy	SE	462k	28.22	11	5.89
San Bernardino	IR	450k	27.50	11	5.61
San Bernardino	IE	450k	27.50	11	5.60
San Bernardino	SE	465k	28.39	11	5.95

Table IV. Number of vertices which must be decoded to obtain a TIN within a certain approximation error (expressed as a percentage on the heigth range of the data set), with the three encoding algorithms.

Terrain	algo	0.25 %	0.5 %	1 %	2 %	5 %	10 %
Devil Peak	IR	33766	29473	22385	13515	4060	1349
Devil Peak	IE	30845	26113	18525	10086	2830	925
Devil Peak	SE	32566	29154	20910	10512	2483	805
Mount Marcy	IR	12323	9049	5578	3205	1368	818
Mount Marcy	IE	10540	7278	4168	2378	936	527
Mount Marcy	SE	10022	7037	3915	2068	692	230
San Bernardino	IR	13302	11836	9863	7389	3414	1296
San Bernardino	IE	12083	10613	8660	6070	2579	1296
San Bernardino	SE	11222	10613	7311	4849	1965	657

We experimented three different encoding algorithms on three terrains (datasets courtesy of the U.S. Geological Survey). The characteristics of the three TINs are summarized in Table II. Tables III and IV show the results. The method denoted as IR removes a maximal set of mutually Independent vertices chosen in a Random way; IE removes a maximal set of mutually Independent vertices chosen in an Error-driven way; SE removes a Single vertex chosen in an Error-driven way. In all cases, the bound on the degree of vertices to be removed is $b = 11$.

All heuristics give a bitstream length close to the theoretical bound: it is slightly lower for methods that remove independent vertices, and slightly higher for the method that removes a single vertex. Method IR needs more vertices to be decoded in order to achieve a given precision in the resulting TIN, with respect to error-driven methods IE and SE. At low resolution, SE always outperforms the other two heuristics, and generally behaves better than IE in most cases.

The decompression algorithm works in linear time, since n vertices are read, and the insertion of any vertex v into the current triangulation involves a search for t_v in the star of a vertex w of degree bounded by b, and a number of flips bounded by $b-1$, with b a predefined constant. Note that no numerical computations are needed in reconstructing the TIN.

4.3. Discussion

The main advantage of our method is in being more general and flexible than other progressive methods [12, 13, 17]. In fact, it does not impose any specific criterion for removing vertices and for retriangulating after vertex removal. It can incorporate different criteria for choosing the vertices to remove, trading off between compression time and quality of the intermediate meshes. The only constraint is on the maximum degree of removed vertices (similar to Hoppe's). Hoppe does not take into account the approximation error when selecting the next edge to collapse. Also, Snoeyink and Van Kreveld do not select vertices with an error-based criterion; moreover, deleting independent sets of vertices at once puts a strong constraint, hence reducing adaptivity: the number of vertices necessary to achieve a given approximation error may be much higher.

In our approach, the specific criterion used for retriangulation is encoded in the bitstream itself rather than being predefined. On the contrary, [17] is restricted to Delaunay triangulations and [12, 13] to triangulations obtainable through edge collapse (for instance, it cannot guarantee a Delaunay triangulation at each step).

Our method is less compact than those in [12, 13] and [17]. Disregarding the cost of the initial triangulation, which is always very small, our method produces $n(\lceil \log n \rceil + \lceil 2 \log b \rceil + 6)$ bits of connectivity information. Hoppe's method requires $n(\log n + \log(b(b-1)))$ bits. For $n = 2^{16}$ and $b = 8$ ($b = 16$), Hoppe's method costs $21n$ ($23n$) bits, while our bitstream requires about $28n$ ($30n$) bits. Thus our method has an overhead of about 30% with respect to Hoppe's with the advantage of encoding more general TINs (e.g., Delaunay, data-dependent, etc.) and of being based on vertex removal/insertion, which is probably the most common technique used in GIS for terrain simplification.

The bitstream of [17] describes connectivity in just $O(\log^2 n)$ bits; no experimental results are known. However, their decoding algorithm involves heavy numerical calculations, which are time-consuming and subject to floating point inaccuracies; our method does not perform any numerical computation in decoding.

5. Concluding Remarks

We have proposed two new methods for compressing TINs. The first method is a direct method based on traversing a TIN in a shelling order. It sends each vertex once, and produces less than 4.5 bits of connectivity information for each vertex, which is advantageous with respect to other methods [3, 8, 9, 19]. Moreover, the adjacency relations between triangles are reconstructed directly from the bitstream at no additional processing cost. Because of this property, it is better suited to TIN analysis than methods based on strips (which are essentially optimized for rendering purposes).

The main advantages of this compression technique are the following: it provides triangle adjacency information, it is conceptually simple and easy to implement, and it gives good compression rates.

The algorithm proposed in [19], although more complicated than ours, has similar features: it works expanding a polygon through a TIN and provides triangle adjacency information. In [19], Touma and Gotsman achieve improved compression rates by further compressing the bitstream through entropy encoding and run-length encoding. A similar signal compression strategy can be applied to the sequence generated by our shelling-based algorithm as well. We plan to test such possibility in the near future, and we believe that this could lead to improvements in compression rates comparable to those of [19].

Here, we have proposed our compression method for extendably shellable triangle meshes and we have applied it to TINs. The method can be extended to triangle meshes that are not extendably shellable. If the input mesh, representing a free-form surface in space, is not extendably shellable, then it is automatically partitioned into shellable patches and each patch is encoded independently. This adds just a small overhead due to the replication of vertices that belong to more than one patch. Based on a similar principle, the method also extends to higher dimensions to compress tetrahedral meshes which, in general, are not extendably shellable. In this case, the number of possible control codes and, thus, the number of bits to represent them, increases: four bits are needed to compress tetrahedral meshes. Some results on generic triangle meshes and on tetrahedral meshes can be found in [15].

The second method belongs to the class of compression methods providing progressive approximations for a TIN. It allows transmission of a TIN according to criteria related to the approximation error of the terrain representation. It is more flexible than Hoppe's method [12, 13] since it can deal with Delaunay triangulations, and it has a broader applicability than the one by Snoeyink and van Kreveld [17], since it can deal with data dependent and constrained Delaunay triangulations. On the other hand, it is just a little more expensive than Hoppe's method. From the encoded bitstream a multiresolution model of a TIN, the Multi-Triangulation, can be constructed, which makes it possible to perform terrain analysis at variable resolution [6]. It applies to generic surfaces in space, other than TINs. We are currently studying an alternative encoding scheme for the flip sequence that might be more compact and, furthermore, be easily reversed in order to manage dynamic processes in which a mesh can be either refined or coarsened on-line.

We have shown experimental results for the first method, while the second one is currently under implementation. Compression algorithms with different error-driven selection techniques have been experimented already in [6].

Acknowledgements

This work has been developed while Leila De Floriani was visiting the University of Maryland Institute for Advanced Computer Studies (UMIACS); her research has been supported by National Science Foundation (NSF) Grant "The Grand Challenge" under contract BIR9318183.

This work has been partially supported by the Strategic Project of The Italian National Research Council "Symbolic Computation – Systems for Geometric Modeling" under contract number 97.04800.ST74.

References

1. R. Bar-Yehuda and C. Gotsman. Time/space tradeoffs for polygon mesh rendering. *ACM Transactions on Graphics*, 15(2):141–152, 1996.
2. H. Bruggeser and P. Mani. Shellable decompositions of cells and spheres. *Math. Scand.*, 29:197–205, 1971.
3. M.M. Chow. Optimized geometry compression for real-time rendering. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97 Proceedings*, pages 347–354. IEEE Press, 1997.

4. M. de Berg, R. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *International Journal of Geographic Information Science*, 11, 1997.
5. L. De Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Proceedings IEEE Visualization 97*, pages 103–110, Phoenix, AZ (USA), October 1997.
6. L. De Floriani, P. Magillo, and E. Puppo. VARIANT - processing and visualizing terrains at variable resolution. In *Proceedings 5th ACM Workshop on Advances in Geographic Information Systems*, Las Vegas, Nevada, 1997.
7. L. De Floriani, P. Magillo, and E. Puppo. Compressing TINs. In *Proceedings 6th ACM Workshop on Advances in Geographic Information Systems*, 1998.
8. M. Deering. Geometry compression. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '95), ACM Press*, pages 13–20, 1995.
9. F. Evans, S. Skiena, and A Varshney. Optimizing triangle strips for fast rendering. In *Proceedings IEEE Visualization '96*, pages 319–326, 1996.
10. L.J. Guibas, D.E. Knuth, and M. Sharir. Randomized incremental construction of the Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.
11. P. Heckbert and M. Garland. Survey of surface simplification algorithms. Technical Report , Department of Computer Science, Carnegie Mellon University, 1997.
12. H. Hoppe. Progressive meshes. In *ACM Computer Graphics Proc., Annual Conference Series, (SIGGRAPH '96)*, pages 99–108, 1996.
13. H. Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22:27–36, 1998.
14. J. Lee. A drop heuristic conversion method for extracting irregular networks from digital elevation models. In *Proceedings GIS/LIS'89*, pages 30–39, Orlando, FL, USA, 1989.
15. P. Magillo. *Spatial Operations on Multiresolution Cell Complexes.* PhD Thesis, Dept. of Computer and Information Sciences, University of Genova (Italy), 1999.
16. B.P. Pennbaker and J.L. Mitchell. *JPEG, Still image compression standard.* Van Nostrand Reinhold, 1993.
17. J. Snoeyink and M. van Kreveld. Linear-time reconstruction of Delaunay triangulations with applications. In *Proc. 5th European Symposium on Algorithms*, 1997.
18. G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.
19. C. Touma and C. Gotsman. Triangle mesh compression. In *Proceedings Graphics Interface'98*, pages 26–34, 1998.