Representation and Visualization of Terrain Surfaces at Variable Resolution

P. Cignoni[†], E. Puppo[‡], R. Scopigno[⋆]

† Dipartimento di Informatica - Università di Pisa,

C.so Italia, 40 - 56100 Pisa ITALY - Email: `cignoni@di.unipi.it`

‡ Istituto per la Matematica Applicata – Consiglio Nazionale delle Ricerche

Via dei Marini, 6 (Torre di Francia) - 16149 Genova, ITALY - Email: `puppo@ima.ge.cnr.it`

⋆ CNUCE – Consiglio Nazionale delle Ricerche,

Via S. Maria 36, 56126 Pisa, ITALY - Email: `R.Scopigno@cnuce.cnr.it`

Abstract

We present a new approach for managing the multiresolution representation of discrete topographic surfaces. A Triangulated Irregular Network (TIN) representing the surface is built from sampled data by iteratively refining an initial triangulation that covers the whole domain. The refinement process generates triangulations of the domain corresponding to increasingly finer approximations of the surface. Such triangulations are embedded into a structure in a three dimensional space. The resulting representation scheme encodes all intermediate representations that were generated during refinement. We propose a data structure and traversal algorithms that are oriented to the efficient extraction of approximated terrain models with an arbitrary precision, either constant or variable over the domain.

1 Introduction

The search for multiresolution representation schemes has recently become very popular. Major applications involve generic surfaces embedded in 3D space [16, 8, 27], terrains in the context of Geographical Information Systems [10, 23, 12], 3D objects for classical CAD and recognition [9, 20, 21], and volume data [5, 31, 30]. All such models are based on the idea that a detailed digital model taken as input can be simplified into an approximate representation: appropriate measures of fidelity to the original model are taken as a quantitative mean to define multiple resolution levels.

There are two major challenges underlying the construction of multiresolution models [15]:

1. to find effective and efficient algorithms for automatically building an approximate model at a predefined level of resolution;

2. to structure models at different resolutions into a comprehensive framework that allows data to be manipulated at different resolutions according to the needs of a given application or task.

The main idea underlying all the works proposed in the literature to pursue the first goal is that a simplified model can be built based on a reduced set of data. The main approaches to the construction of approximate models are iterative, and can be classified into *simplification* methods [25, 28, 21, 17, 27] - i.e., methods that start from the full resolution and progressively reduce the dataset on which the model is based, in order to coarsen resolution; and *refinement* methods [13, 16, 8, 23, 12, 5] - i.e., methods that start from a very coarse approximation based on a very small dataset, and progressively refine it by inserting new data, in order to improve resolution.

Most methods for the approximate representation of surfaces use piecewise linear representations based on triangulations, because of their adaptivity. All methods to build approximate triangulated surfaces follow heuristics in trying to minimize the amount of data needed to achieve a given resolution. The most common approach is to base point selection on the impact, in terms of error reduction/increase, which is caused by the insertion/deletion of a point into/from the dataset. Many authors have also tried to preprocess data to extract meaningful features, in the form of points and lines [19, 29, 22, 26]. Other authors have attempted to improve results by shifting the vertices so that curvature within the triangles is nearly equal, or by removing unnecessary triangles [24].
Finding the best method to select from a set of sites the vertices or edges on which the triangulation has to be built is still an open issue. However, a recent theoretical result [1] suggests that the search for an optimal solution is hopeless in many cases; moreover, approximated algorithms that can guarantee a certain bound with respect to the optimal solution are far too slow to be of practical interest.

The models proposed in the literature to pursue the second goal can be broadly classified into *multi-layer* models [10, 5] - i.e., frameworks that relate a sequence of independent models of the whole object represented at different levels of resolution - and *tree* models [9, 20, 16, 8, 23, 12, 21, 27] - i.e., models in which a hierarchy of descriptions is represented by a tree, where each node corresponds to the description of (a portion of) an object at a given level of resolution, while each of its children represents a more detailed description of a piece of such an object. A comprehensive discussion of multiresolution models for terrain representation is presented in [11]. More recent advances in the formalization of multiresolution models for scalar fields in any dimension also outline the possibility of overcoming the above classification into multi-layer and tree, in order to obtain more compact models that incorporate different levels of resolution within a unified representation [2].

An interesting yet not much explored application of multiresolution models is rendering at variable resolutions over various zones of the surface/object/volume. A typical example is in landscape visualization for either flight simulators, or environmental assessment [18]: the detail of the terrain model presented to the user may be variable, depending on the distance from the point of view. Multiresolution allows a larger number of polygons to be rendered only in the areas where the visual impact is at its most significant. A similar approach has also been outlined in scientific visualization to sharpen resolution only in user-selected focus areas [5, 6]. The main problem in providing a representation with variable resolutions is to maintain the continuity of the surface where pieces of surface with different precisions meet. Most methods resolve this problem through rendering artifacts [6], or merely choose to ignore it.

In this paper, we present a multiresolution model for triangulated topographic surfaces, called a HyperTriangulation (HT), which is more compact and flexible than previous models. Our model is based on a structure that can maintain all significant refinement/simplification steps in passing from either a coarse representation to a refined one, or vice versa. Intermediate representations are maintained implicitly in the model: an efficient data structure allow "on the fly" representations to be retrieved at arbitrary resolutions between the minimum and maximum available. Moreover, representations can be efficently extracted at resolution variable over the domain, while still guaranteeing the continuity of the resulting surface.

The definition of the model is independent of its construction, provided that representations whose resolutions are close to each other can be related through local changes over the domain. Although we have based our construction algorithm on a refinement technique, it is straightforward to build it through a simplification technique.

The rest of the paper is organized as follows. Section 2 reviews the basic concepts about

Triangulated Irregular Networks, and the construction of approximate models. A general description of the ideas behind HyperTriangulations is given in Section 3. Section 4 describes a data structure to manage HT. The algorithms for the extraction of approximated models from HT are presented in Section 5. Our conclusions are drawn in Section 7.

2 Approximated Digital Terrain Representation

A natural terrain is mathematically described by an elevation function $\phi : D \subseteq \mathbb{R}^2 \rightarrow \mathbb{R}$, defined over a connected domain D on the XY plane. The surface described by the image of ϕ is often called a *topographic* or $2\frac{1}{2}D$ surface. In practical applications, function ϕ is sampled at a finite set of points $P = \{p_1, ..., p_n\} \subset D$, known as the set of representative points in the digital terrain. In this case the function ϕ can be defined piecewise over a subdivision Σ of D with vertices in P.
When a triangular subdivision is adopted to partition D, piecewise linear functions are a common choice to compute the elevation of points that are not in P. One such model is called a *Triangulated Irregular Network (TIN)*: TIN models of $2\frac{1}{2}D$ surfaces can be adapted to the characteristics of the surface, they can be built on scattered data, and they are widely used in many different fields, such as Geographical Information Systems, Finite Element Analysis, robotics, and computer graphics in general.
Since a TIN is fully characterized by the plane triangulation underlying it, plus the elevation value at each of its vertices, hereafter we will always work on the plane triangulation, by considering the triangles that form the actual surface only for the purpose of rendering or error testing.

2.1 Approximation error

As we pointed out in the introduction, the construction of an approximated representation is based on the possibility of selecting a significant subset of data from either a regular or scattered dataset. The selection is almost always based on some measure of the error in representing a given surface through a simplified model. In the case of TINs, different norms can be adopted to measure the distance between a surface represented by a TIN built over the whole dataset, and the surface corresponding to a reduced model based on a subset of data. A simple and common choice is to measure such errors by the maximum distance between the elevation at a datum and its approximate elevation in the reduced representation. The relevance of a given data point p in the current representation is related to the increase/decrease in the error as a consequence of the deletion/insertion of p from/into the model.

Another critical issue is the preservation of point and lineal features, such as ridges, valleys, peaks, and pits. Features can be identified by sharp discontinuities in the gradients of adjacent facets. Such features may be largely preserved if, while constructing the multiresolution representation, a measure of likelihood is adopted, which tends either to maintain or to insert points belonging to features. For instance, the selection heuristic can take care of the discontinuity on the gradient that is introduced/lost with the insertion/deletion of a point.

Below we describe a refinement technique on which we have based the construction of our model. In order to remain generic, we assume that at each step a *score* can be computed for each datum that is not a vertex of the model. This score may be dependent on the norm used to measure the approximation error, and on any other parameter involved in point selection, as discussed above. In order to preserve the efficiency of the method, it must be possible to compute such a score whenever the TIN is updated only at the points involved in changes. Moreover, for each such point p, it must be possible to compute its score in constant time, based only on local information (e.g., on the triangle of the current model covering p). We also assume that the approximation error of the TIN is updated while the score for each point is computed, at no extra cost.

In the simplest case, the score is the absolute value of the difference between the approximated and the actual elevation at p, while the current approximation error coincides with the maximum score over the triangulation. In a more sophisticated selection scheme, the score of p may be evaluated by weighting the surface error at p with the difference of the gradients of the three facets obtained by inserting p as a new vertex into the triangle containing it.

2.2 The Delaunay Selector

The method we adopted in this work builds an approximated TIN through a refinement technique based on the on-line Delaunay triangulation of points on the XY domain. This approach is called the *Delaunay Selector*, and it derived from an early method proposed in [13]. A 3D generalization of this method has also been used for multiresolution volume modeling and visualization [5]. Here, we give a brief description of the algorithm, based on an efficient implementation proposed in [11].

Let $\varepsilon \geq 0$ be a tolerance value, let P be a finite set of points in $\mathbb{R}^2$, and let ϕ be the elevation function known at the points of P. An initial triangulation Σ is built first, whose vertex set is

```
DelaunaySelector(SetofPoint P, var Triangulation $\Sigma$, var $\varepsilon$)
begin
    $\Sigma$ := BuildInitialTriangulation(P);
    while not Err ($\Sigma$) $\leq \varepsilon$
        p := SelectMaxScorePoint(P,$\Sigma$);
        $\Sigma$ := UpdateTriangulation($\Sigma$, p);
end
```

Figure 1: The Delaunay Selector Algorithm.

composed of all extreme points of the convex hull of P: such a triangulation covers the whole domain of the sampled data[1].

The triangulation is refined through the iterative insertion of new vertices, one at a time: at each iteration, the point of P with the highest score is inserted as a new vertex, and Σ is updated accordingly. The refinement process continues until $E(\Sigma) \leq \varepsilon$. A pseudo code description of the Delaunay Selector algorithm is shown in Figure 1.

Note that, as with most refinement techniques, the insertion of a single point during the Delaunay selector does not necessarily cause a decrease in the approximation error (simplification techniques have a symmetric behavior). In any case, the convergence of the method guarantees that the approximation will improve after some other vertices have been inserted. Hereafter we will call a *refinement step* a minimal sequence of consecutive point insertions such that the error of the resulting approximation is smaller than the error in the previous step. The area of the domain involved in a refinement step is always a polygonal region (which may be unconnected and/or multiply connected), which we will call the *refinement region.* The refinement region at a given refinement step is always bounded by edges that belong to both the triangulation before refinement, and the triangulation after refinement.

3 HyperTriangulation

Let us suppose that a Delaunay Selector is being run with an error tolerance $\varepsilon = 0$: the final structure generated by the algorithm will be a model at full resolution. If we consider all models

[1]An alternative is to start with a single triangle containing all data, whose vertices are ideally at infinity; the value of f at such dummy vertices is set to $+\infty$. Dummy vertices and triangles incident on them can easily be removed during a postprocessing phase.

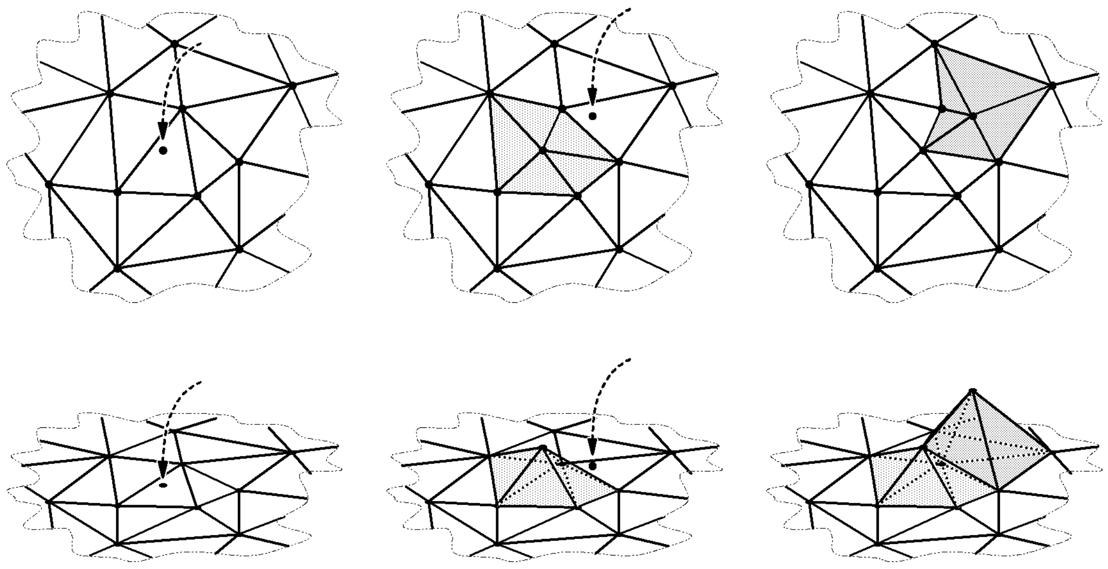

Figure 2: Two steps in the refinement process and the corresponding update on the current hypertriangulation.

built at intermediate refinement steps, we have a whole sequence of triangulations $\{\Sigma_0, \ldots, \Sigma_n\}$, where Σ_0 is the initial triangulation, $\Sigma = \Sigma_n$ is the full resolution model, and $\forall i = 0, \ldots, n$, the TIN associated with triangulation Σ_i approximates the full resolution with an error ε_i. The sequence of error tolerances monotonically decreases: $\varepsilon_0 > \varepsilon_1 > \ldots > \varepsilon_n = 0$.

The sequence of triangulations could be piled up into a layered model, such as the Delaunay Pyramid proposed in [10][2]. As pointed out in [2], the Delaunay pyramid has the disadvantage of replicating at each new layer all the portions of the triangulation that remain unchanged from the previous layer as well. This redundancy would involve an explosion in the complexity of the resulting structure when small refinements are performed from step to step, and resulting levels are a high number, as described above.

Our alternative approach is to store a sort of history of the incremental refinement process. Structures that store the full history of the incremental construction of Delaunay triangulations were proposed in [3, 14], which depend on the construction algorithm, and whose main purpose is to improve point location either during construction or spatial query. Our model maintains

[2]Actually, in the Delaunay pyramid refinement steps are imposed a priori, on the basis of a decreasing sequence of tolerances, in order to reduce the number of layers. The pyramid also stores vertical interference links that are not discussed here.

a history that is independent of the construction algorithm (indeed, the same model can be built through a simplification technique that iteratively demolishes a triangulation), and is simplified with respect to the previous ones. Only triangles that are part of some intermediate triangulation Σ_i are maintained, while all triangles that only appear in the context of a single refinement step are discarded.

Let us consider the refinement region that is retriangulated in passing from Σ_{i-1} to Σ_i. Triangles of Σ_{i-1} and Σ_i can be classified as follows:

- *living triangles*: the triangles that are not changed during refinement (i.e., triangles outside the refinement region, that belong both to Σ_{i-1} and Σ_i);

- *dead triangles*: old triangles destroyed while updating the triangulation (i.e., the triangles of Σ_{i-1} that belong to the refinement region);

- *newborn triangles*: new triangles created while updating the triangulation (i.e., triangles of Σ_i inserted into the refinement region).

Note that usually most triangles are *living*, because the incremental insertion process acts only locally. By definition, the set of dead triangles and the set of newborn triangles respectively form two triangulations of the refinement region. Such triangulations share the edges that bound this region. Hence, instead of simply replacing the triangulation inside the refinement region, as in the standard Delaunay selector, we can "paste" a patch formed by the newborn triangles over the triangulation formed by the dead triangles, while saving the dead triangles below the newborn ones. The refinement proceeds by iteratively pasting a patch at each refinement step.

In order to make the whole structure understandable, we embed it in 3D space: the triangulation Σ_0 lies on the XY plane, while at refinement step i the new vertices inserted are raised up along the Z axis at elevation i, and the new patch is welded onto the old triangulation at the boundary of the influence region: this can be visualized as a "bubble" popping up from the triangulation (see Figure 2). The resulting structure is a 2D simplicial complex embedded in 3D space, such that at each step the current triangulation is formed by the triangles of the upper surface of the complex[3]. This structure is called a *HyperTriangulation (HT)*: it maintains both the geo-topological information collected during the refinement process, and information on the error of each triangle, which is useful for extracting representations at arbitrary resolutions.

[3]Note that the surface corresponding to the current triangulation on the HyperTriangulation has no relation to the terrain surface defined by its corresponding TIN.

Note that now different triangulations of the sequence $\Sigma_0, \dots \Sigma_n$ are not stored explicitly and independently, but they are interconnected in order to store only once any portion that is common to different triangulations. This fact makes the model quite compact. Each intermediate triangulation is encoded implicitly in HT. In order to show this, let us define the following two attributes for each triangle t in HT :

- ε_b: *birth error* the global error reached by the triangulation just before triangle t was created;
- ε_d: *death error* the global error of the triangulation just before t was destroyed (zero if the triangle is part of the complete triangulation).

The birth and death errors allow to detect those triangles in HT that were contained in the triangulation Σ_i, produced as an intermediate result of the refinement process of the Delaunay Selector, which satisfied approximation error ε_i. Consider a triangle t in HT, which satisfies the following inequality:

$$t.\varepsilon_d \leq \varepsilon_i < t.\varepsilon_b, \tag{1}$$

where $t.\varepsilon_b$ and $t.\varepsilon_d$ are the birth and death errors of t, respectively: t is called a ε_i*-alive* triangle. From the definition above and from (1) it follows that all ε_i-alive triangles must belong to Σ_i. We show that Σ_i is in fact formed only by such triangles.

Let p be a point in the domain D of the HyperTriangulation HT. For the sake of simplicity, let us assume that p does not lie on the projection of any edge of HT on the XY plane. Points that lie on projected edges can be treated exactly the same way, but the procedure is a little more technical. We define the set of triangles that cover p as:

$$T_p = \{\ t \in HT : p \in \hat{t}\ \} \tag{2}$$

where $\hat{t}$ is the projection of t on the XY plane. For each T_p there exists an ordering $t_1, t_2, ..., t_n$ on the set of its elements such that:

$$\forall i : \quad t_i.\varepsilon_b > t_i.\varepsilon_d = t_{i+1}.\varepsilon_b > t_{i+1}.\varepsilon_d \quad \text{where } 1 \leq i < n; \tag{3}$$

Indeed, whenever a newborn triangle containing p is generated during construction, the triangle containing p in the current triangulation must die, and the birth error and death error of the newborn and dead triangle, respectively, must coincide. More informally, for each point p of D there must exist only one triangle in HT whose projections in the XY plane contain

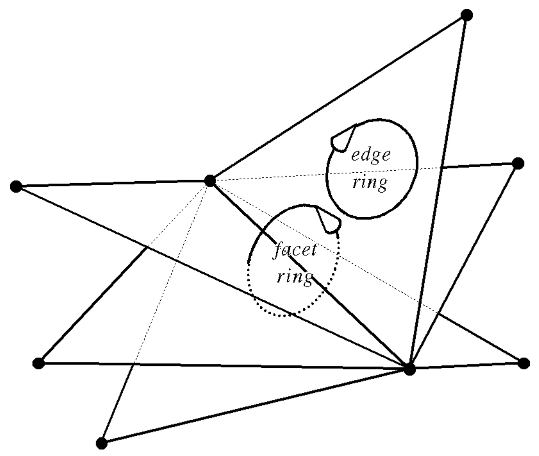

Figure 3: A FacetEdge belongs to two rings.

p and which is ε_i-alive. Hence, the set of ε_i-alive triangles cover the whole domain, and thus there cannot be other triangles in Σ_i.

Since the birth and death error of each triangle in HT will be used to efficiently extract terrain representations from HT (see Section 5), they will be encoded explicitly in the model.

4 A data structure for HyperTriangulation

In this section we describe a data structure for managing HyperTriangulations, which is based on the *facet-edge* structure described in [7] for representing cell complexes in three dimensions[4].

In the facet-edge data structure, an atomic entity is associated with each pair that is identified by a face and one of its edges: the so-called *facet-edge*. This structure is equipped with traversal functions that permit the complex to be traversed. Such traversal functions are *enext* and *fnext*. These functions are used to move from a facet-edge to an adjacent one, either by changing edge on the same face, or by changing face around the same edge (note that in 3D space more than two faces (triangles) may be incident at each edge).

Let t be a face (triangle) of a cell complex C, and let e be one of the edges of t. The facet-edge te denotes two rings in C: the *edge-ring* is formed by all the edges of the boundary of t; the *facet-ring* is formed by all the faces incident at e (see Figure 3). The traversal function

[4]Actually, we are only interested in the 2-skeleton of a three dimensional complex, i.e., the 2-simplicial complex formed by all triangles pasted into the HT during refinement.

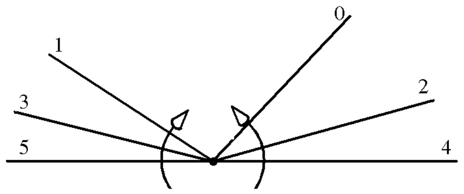

Figure 4: Facet-edge ordering respects the error ordering (arrows represent the fnext chain).

enext and **fnext** permit one to move from one facet-edge to the next along the edge-ring and the facet-ring, respectively.

We have attached two fields to each facet-edge to store the birth and death error of the triangle to which it belongs. Moreover, we have made some structural changes to make the data structure more suitable to our needs. Note that in the HT, the triangles incident at the same edge e can be subdivided into two groups, namely, those formed by faces whose projection on the XY plane lie either to the left or to the right of the projection of e, respectively. When traversing the HT, we may need to move through two different domains: the *spatial* domain D and the *error* domain. In other words, in the former case, we may need to cross an edge e from one triangle to another, which belongs to the group on the other side of e, and which has a compatible precision[5]. In the latter case, we may need to adjust the precision by moving to the facet that either preceeds or follows the current one in the facet-ring, while remaining on the same side of the edge.

We thus maintain the facet-ring split into two separate bidirectional chains, according to the side of the domain that the corresponding triangles cover. Moreover, we maintain one more link from each facet-edge to a facet-edge on the other side of the edge. Link `fother` connects a facet-edge te with the facet-edge on the opposite side of e that was either created together with te, or had a minimum error when te was created. Hence, for each facet-edge we encode the facet-ring with the following fields:

- `fnext`: next facet-edge in the facet-ring (with lower error);
- `fprev`: previous facet-edge in the facet-ring (with higher error);

[5]By *compatible precision* we mean that the error ranges spanned by the birth and death errors of the two triangles overlap.

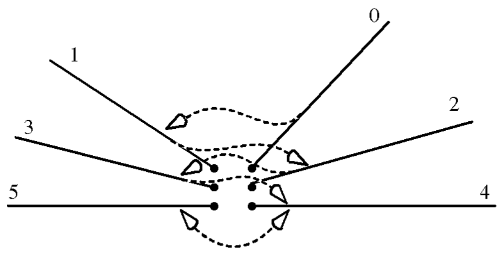

Figure 5: A facet-edge is directly connected with a facet-edge on the other side.

- `fother`: facet-edge on the other side.

Figure 4 shows a side view of the facet-edges in Figure 3. The two arrows represent the two parts of the edge-ring. Figure 5 shows the `fother` links; the numbers represent the value of birth error for the facet-edge. Note that the `fother` links are not necessarily symmetric.

5 Extracting triangulations from HT

Given a HyperTriangulation encoded with the structure described in the previous Section, it is possible to efficiently extract triangulations defining TINs such that:

- the approximation error is either constant **or** variable over the domain D according to a function defined on D, and indicating the error tolerance accepted at each point in the domain, **and**
- the continuity of the surface extracted is guaranteed everywhere.

Extraction at constant approximation The simplest case in this context is the extraction of a triangulation at a constant approximation which corresponds to an error ε in the range of available resolutions (i.e., between error zero and the error corresponding to the bottom level of HT). One such triangulation can be extracted from HT through a topological visit, starting from a triangle t that satisfies $t.\varepsilon_d \leq \varepsilon < t.\varepsilon_b$, and moving to adjacent triangles that satisfy the same relation. A pseudo-code of the algorithm is presented in Figure 6. The algorithm traverses the HT, moving from triangle to triangle through primitive `fother`, and it keeps the set of facet-edges which still have to be visited in two stacks: `BubbleStack`, which holds the facet-edges on the same bubble, and `ActiveStack`, which holds the other facet-edges. The next triangle to visit is selected according to the fixed error `eps`.

The time complexity of this algorithm depends on the number of triangles of the extracted triangulation and the number of facet-edges visited to find the one that satisfies the error and is adjacent to the current facet-edge.

The triangulation at a constant error ε is formed by all triangles that were contained in the *top* surface of the HT when the refinement process reached the first (and largest) among all errors ε_i such that $\varepsilon_i < \varepsilon$. We extract all such triangles by navigating such a top surface through adjacencies. Suppose we start from a ε-alive triangle, if we move on the top of the HT we only need to follow the fnext and fother links. Indeed, when we reach the border of a bubble, the fother field links the current facet-edge e with the facet-edge e' adjacent to e that was on the top of the HT when e was created. Since e' was created before e, it must have a birth error that is larger than that of e. Hence, if e' is not ε-alive, this is certainly because it has a death error lower than ε; so, we must follow the fnext link from e' to climb another bubble.

A fair strategy to find the first ε-alive triangle is to pick an edge of the boundary of HT. By construction, an ε-alive triangle exists on this edge and we can retrieve its facet-edge through a simple scan of the fnext chain. A more efficient alternative is to keep all the errors $\varepsilon_0, \ldots, \varepsilon_n$ sorted in a balanced tree (or in an array with direct access), and to maintain one pointer from each such error ε_i to a triangle in the HT that is ε_i-alive: for any ε, the proper value of ε_i and, hence, the corresponding initial triangle can thus be found in a time logarithmic in the total number of errors spanned by the HT.

Moreover, we visit the top surface of the HT by climbing each bubble only once. This simply entails visiting all the facet-edges of each bubble before moving to another bubble. This can be done by following with higher priority the facet-edges that satisfy the error without the needs of following the fnext link. Since each bubble will be climbed at most once during the visit, the number of times that the algorithm follows the fnext link is bounded by the number of bubbles. Let n_t be the total number of triangles in the extracted triangulation, and let n_s be the number of steps in the refinement process. If we note that the number of bubbles is n_s, we can state that the worst case time complexity of this algorithm is $O(n_t + n_s)$.

Extraction at variable approximation The extraction of a surface with a variable approximation error over the domain requires a different HT traversing strategy. Here, we describe an

```
ExtractFixedErr(HyperTriang HT, float eps, var Triangulation $T_{eps}$)
begin
    fe=FindFirstFE(HT, eps);
    InsertFace(fe,$T_{eps}$);
    for i=1 to 3
        Mark(fe);
        Push(fe, ActiveStack);
        fe=fe.enext;
    while(ActiveStack is not empty)
        fe = Pop(ActiveStack);
        Push(fe,BubbleStack);
        while(BubbleStack is not empty)
            fe = Pop(BubbleStack);
            fe=fe.fother;
            while (fe.deatherr > eps) fe=fe.fnext;
            if not Marked(fe)
                InsertFace(fe,$T_{eps}$);
                for i = 1 to 2
                    fe = fe.enext;
                    if not Marked(fe)
                        Mark(fe);
                        if fe.fother.deatherr <= eps < fe.fother.birtherr
                            then Push(fe,BubbleStack);
                            else Push(fe,ActiveStack);
end;
```

Figure 6: The algorithm for extracting a terrain model with error `Eps` from a HyperTriangulation.

algorithm that is suitable whenever the approximation error follows a function $E : D \rightarrow \mathbb{R}$ of the class:

$$E(p) = f_e(d(v_p, p)) \tag{4}$$

where $f_e : \mathbb{R} \rightarrow \mathbb{R}$ is a monotonically increasing function, $d()$ is the standard Euclidean distance in $\mathbb{R}^2$, and v_p is a fixed point called the *viewpoint*. This error function is adequate for applications such as flight simulators: the farther we are from the viewpoint, the larger the tolerance error that can be accepted for terrain representations.

Let us consider the algorithm in Figure 6 and let us suppose that we are in an intermediate step in the extraction of the triangulation with constant approximation error `eps`. Consider the facet-edges currently contained in the ActiveStack: all of them are certainly `eps`-alive, i.e. for each facet-edge e in ActiveStack, the following holds:

$$e.\varepsilon_d \leq \texttt{eps} < e.\varepsilon_b \tag{5}$$

Let B_{Min} be the minimum birth error of these active facet-edges, then:

$$e.\varepsilon_d \leq B_{Min} < e.\varepsilon_b \tag{6}$$

Hence, active facet-edges are the right choice also for extracting a triangulation with constant error B_{Min}. Following this criterion we can progressively increase the error during the extraction. Therefore, this technique can be used to extract a triangulation whose precision adapts to error function $E()$.

The algorithm visits the HT starting from the triangle that is closest to (or, possibly, contains) the viewpoint v_p, and satisfies the error function $E()$. Then, it proceeds by visiting HT and increasing the error of extraction at each step of the visit. Active facet-egdes are maintained in a priority queue MBH (Minimum Birth Heap) which is ordered with respect to their minimum birth error. At each step, the next facet-edge to be visited is selected by extracting the minimum element from the heap. Moreover, the algorithm tries to adjust the tolerance `eps` that controls the extraction of triangles according to the error function on the extracted facet-edge and to the minimum birth error of facet-egdes.
Hence, the main steps of the algorithm are:

- extract facet-edge with minimum birth error e;
- check how much the tolerance can be raised, according to the current minimum birth error and to the value of $E()$;

- find the correct facet-edge e' on the other side of e and put the other facet-edges on the edge ring of e' into the heap.

To start the extraction we must detect the triangle in HT having error 0 and which cointains (or, is nearest) to viewpoint v_p. A classical point location query can be efficiently solved on the HT structure by exploiting the HT hierachical structure (which represent the history of the on-line construction of the triangulation). If we look to a particular class of applications, e.g. flight simulators, the knowledge of the problem allow us to avoid to cope with a generic point location query. In this case, at time t_i the viewpoint location is generally near to the location of the viewpoint at time t_{i-1}. The triangle containing the current viewpoint can be therefore simply detected by a topologic visit of HT starting from the triangle containing the previous viewpoint.

Figure 7 shows a pseudo-code of the algorithm: heap `BMH` maintains the active facet-edges ordered according to minimum birth error.

The time complexity of this algorithm depends on the number of triangles of the extracted triangulation. We can assume that the number of facet-edges we visit is proportional to the number n_t of facets in the extracted triangulation. For each visiting step we make an access to an heap with a cost that is logarithmic in terms of the number of object stored in the heap. Therefore, the overall complexity of the extraction of a surface with variable approximation error is $O(n_t \log n_t)$.

6 Conclusions and future works

We have presented a new model for the multiresolution representation of triangulated terrain surfaces. The model is more compact than other models described in the literature, and it allows continuous surface representations to be extracted either at an arbitrary fixed resolution, or at variable resolution over the domain.

Although we have given a description of the model based on a refinement construction technique, the same structure can be obtained through any technique that either refines or simplifies a surface defined on the basis of a discrete dataset.

We are currently implementing the algorithms described in the paper, and we plan to test them on real world data. The efficient manipulation of surface information achieved by the model should allow us to obtain a dynamic visualization of landscapes with high quality images in real time.

```
ExtractVarError(HyperTriang HT, function Err(float d), Point vp, var Triangulation T)
begin
    fe=FindFirstFace(HT, vp);
    InsertFace(fe,T);
    for i=1 to 3
        Mark(fe);
        InsertHeap(fe, MBH);
        fe=fe->enext;
    while(MBH is not empty)
        fe=Min(MBH);
        Bmin=fe.birtherr;
        d=dist(fe, vp);
        eps=min(Err(d), Bmin);
        fe=fe.other;
        while (fe.birtherr <= eps) fe=fe.fprev;
        while (fe.deatherr > eps) fe=fe.fnext;
        InsertFace(ef,T);
        for i = 1 to 2
            fe = fe.enext;
            if not Marked(fe);
                Mark(fe);
                InsertHeap(fe, MBH);
end;
```

Figure 7: The algorithm for extracting a model with variable error.

A straightforward extension of the model to handle volume data can be defined through an analogous structure built on tetrahedra embedded in 4D space. The same structure can generally be defined for multidimensional data through a d-simplicial complex embedded in $\mathbb{R}^{d+1}$. The generalization of the fact-edge data structure proposed in [4] extends the same efficient data structure for HyperTriangulations in any dimension. We plan to develop the details of this multidimensional model in the near future, and to develop and test a system based on a three-dimensional version for applications to volume visualization.

References

[1] P.K. Agarwal and S. Suri. Surface approximation and geometric partitions. In *Proceedings 5th ACM-SIAM Symposium On Discrete Algorithms*, pages 24–33, 1994.

[2] M. Bertolotto, L. De Floriani, and P. Marzano. Pyramidal simplicial complexes. In *Proceedings ACM Solid Modeling '95*, page (in print), Salt Lake City, Utah, U.S.A., May 17-19 1995.

[3] J.D. Boissonnat and M. Teillaud. A hierarchical representation of objects: the delaunay tree. In *Proceedings 2nd ACM Symposium on Computational Geometry*, pages 260–268, 1986.

[4] E. Brisson. Representing geometric structures in d dimensions: topology and order. In *Proceedings 5th ACM Symposium on Computational Geometry*, pages 218–227. ACM Press, 1989.

[5] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution Modeling and Rendering of Volume Data based on Simplicial Complexes. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 19–26. ACM Press, October 17-18 1994.

[6] P. Cignoni, C. Montani, and R. Scopigno. MagicSphere: an insight tool for 3D data visualization. *Computer Graphics Forum*, 13(3):317–328, 1994. (Eurographics '94 Conf. Proc.).

[7] D.P. Dobkin and M.J. Laszlo. Primitives for the manipulation of three–dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[8] N. Dyn, D. Levin, and J.A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, April 1990.

[9] G. Fekete and L.S. Davis. Property spheres: a new representation for 3-d object recognition. In *Proceedings Workshop on Computer Vision: Representation and Control*, pages 192–201, Los Alamitos, CA, 1984. CS Press.

[10] L. De Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Comp. Graph. & Appl.*, 9(2):67–78, March 1989.

[11] L. De Floriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. Technical Report 1-94, Department of Information and Computer Sciences, University of Genova, ITALY, 1994.

[12] L. De Floriani and E. Puppo. A hierarchical triangle–based model for terrain description. In *Theories and Methods of Spatio–Temporal Reasoning in Geographical Space, LNCS n. 639*, pages 236–251. Springer-Verlag, Pisa, (Sept. 1992).

[13] R.J. Fowler and J.J. Little. Automatic extraction of irregular network digital terrain models. *ACM Computer Graphics*, 13(3):199–207, Aug. 1979.

[14] L.J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoy diagrams. In *Automata, Languages and Programming, LNCS N.443*, pages 414–431. Springer-Verlag, 1990.

[15] P. Heckbert and M. Garland. Multiresolution Modeling for Fast Rendering. In *Graphics Interface '94 Proceedings*, pages 43–50, 1994.

[16] B. Von Herzen and A.H. Barr. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics*, 21(4):103–110, 1987.

[17] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH '93(Anaheim, CA, August 1-6). In Computer Graphics Proceedings, Annual Conference series, ACM SIGGRAPH*, pages 19–26, 1993.

[18] K. Kaneda, F. Kato, E. Nakamae, T. Nishita, H. Tanaka, and T. Noguchi. Three dimensional terrain modeling and display for environment assessment. *ACM Computer Graphics*, 23(3):207–214, July 1989.

[19] T.K. Peucker and D.H. Douglas. Detection of surface-specific points by local parallel processing of discrete terrain elevation data. *Computer Graphics and Image Processing*, 4:375–387, 1975.

[20] J. Ponce and O. Faugeras. An object centered hierarchical representation for 3d objects: the prism tree. *Computer Vision, Graphics and Image Processing*, 38(1):1–28, Apr. 1987.

[21] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. In T. Kunii B. Falcidieno, editor, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, October 17-18 1993.

[22] L. Scarlatos. An automatic critical line detector for digital elevation matrices. In *Proceedings 1990 ACSM-ASPRS Annual Convention, Denver, CO*, volume 2, pages 43–52, March 18-23 1990.

[23] L. Scarlatos and T. Pavlidis. Hierarchical Triangulation Using Cartographic Coherence. *CVGIP: Graphical Models and Image Processing*, 34(2):147–161, March 1992.

[24] L. Scarlatos and T. Pavlidis. Optimizing Triangulations by Curvature Equalization. In A. Kaufman G. Nielson, editor, *Visualization '92 Proceedings*, pages 333–339. IEEE Press, 1992.

[25] W.J. Schroeder, J.A. Zarge, and W. Lorensen. Decimation of triangle mesh. *ACM Computer Graphics*, 26(2):65–70, July 1992.

[26] D.A. Southard. Piecewise linear surface models from sampled data. In *Proceedings of Computer Graphics International '91*, pages 667–680, June 22-28 1991.

[27] D.C. Taylor and W.A. Barrett. An Algorithm for Continuous Resolution Polygonalization of a Discrete Surface. In *Graphics Interface '94 Proceedings*, pages 33–42, 1994.

[28] G. Turk. Re–Tiling Polygonal Surfaces. *ACM Computer Graphics*, 26(2):55–64, July 1992.

[29] L.T. Watson, T.J. Laffey, and R.M. Haralick. Topographic classification of digital image intensity surfaces using generalized splines and the discrete cosine transformation. *Computer Vision, Graphics and Image Processing*, 29:143–167, Apr. 1985.

[30] R. Westermann. A Multiresolution Framework for Volume Rendering. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 51–58. ACM Press, October 17-18 1994.

[31] J. Wilhelms and A. Van Gelder. Multi-dimensional Trees for Controlled Volume Rendering and Compression. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 27–34. ACM Press, October 17-18 1994.