EFFICIENT ALGORITHMS FOR EMBEDDED RENDERING OF TERRAIN MODELS

Laurent Balmelli, Serge Ayer, and Martin Vetterli

Laboratory for Audio-Visual Communications
Swiss Federal Institute of Technology
1015 Lausanne, Switzerland

ABSTRACT

Digital terrains are generally large files and need to be simplified to be rendered efficiently. We propose to build an adaptive embedded triangulation based on a binary tree structure to generate multiple levels of details. We present a $O(n \log n)$ decimation algorithm and a $O(n \log n)$ refinement algorithm, where n is the number of elevation points. We compare them in a rate-distortion (RD) framework. The algorithms are based on an improved version of the optimal tree pruning algorithm G-BFOS allowing to deal with constrained tree structures and non-monotonic tree functionals.

1. INTRODUCTION AND MOTIVATION

Digital Elevation Models (DEM), representing real terrains have become popular in recent years. Such data is usually given under the form of large uniform sample sets, or elevations. Its usage has a wide spread, from urban planning to the generation of dedicated maps in Geographical Information Systems (GIS). For this reason, rendering this data is an important problem in computer graphics. In order to achieve this task, a solution is to generate a planar approximation of the surface using triangles. However, due to the amount of data, rendering the full model is still an issue. Researchers have studied many optimization methods to improve this task. Most techniques are based on selecting a subset of the elevations, or possibly generate more relevant ones, in order to lower the amount of triangles to be rendered. Triangulations are classified into two categories: regular and irregular. Regular triangulations, or *Regular Triangulated Grid* (RTG) are simple to build with a uniform data set. However, this approach fails when dealing with irregular samples sets. For this reason, many authors have proposed algorithms to triangulate such data, leading to irregular triangulations, often called *Triangulated Irregular Network* (TIN) [1]. In this paper, we propose an approach which starts from a RTG and builds an adaptive grid handling irregular sample sets. We propose an efficient mechanism for generating multiple levels of details. Our method presents several advantages over TINs. First, we don't need to recompute the triangulation between successive levels of details, secondly the successive triangulations are embedded and well adapted for multiresolution. These characteristics are not met by TIN. The embedded data structure allows progressive rendering and transmission, when used in a network context. We present two different approaches: the first one is based on the optimal tree pruning algorithm G-BFOS (first proposed in [2]) and provides a decimation scheme, whereas the second one provides a fast refinement scheme. We compare their performances in a rate-distortion (RD) framework.

2. REGULAR TRIANGULATED GRID

2.1. Construction of the Triangulation

The triangulation is built in a top-down fashion. Consider a sample set of size $n = N \times N$, where $N = 2^p + 1$, then starting with an initial triangulation composed of two triangles, one recursively subdivides each of them. One will be able to do $p + 1$ such subdivisions. Figure 1 shows three triangulation levels (level 0 is not represented) and the sampled elevation points (black dots). A binary tree stores the subdivisions, where each node represents a triangle. For n elevations, the tree depth is $h = 2 \log_2(N - 1)$ and the levels range within $0..h$.

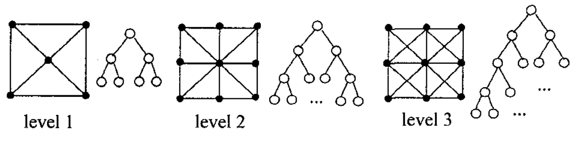

Figure 1: Triangulation and binary tree

2.2. Sibling Triangles

In the generated structure, some triangles share the same hypotenuse, thus a common midpoint. We call these ones *sibling triangles.* One has to consider these triangles in order to avoid shape discontinuities while computing an arbitrary level of details. Consider the simple example of Figure 2, the triangles at level 1 in the tree must be merged (or split) jointly in the structure to avoid a surface crack (also known as *T-vertex*).

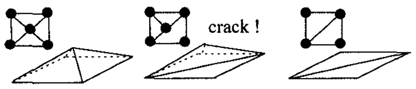

Figure 2: Surface Crack (T-Vertex)

Sibling triangles are present at almost every levels in the tree structure and create triangles splitting/merging dependencies. Figure 3 and Figure 4 depicts respectively the necessary operations to split/merge triangles in the structure. In both figures, the black node is split/merged. The gray nodes show the influenced ones.

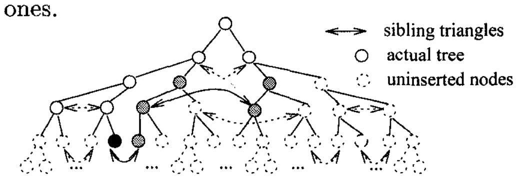

Figure 3: Example of Triangle Splitting

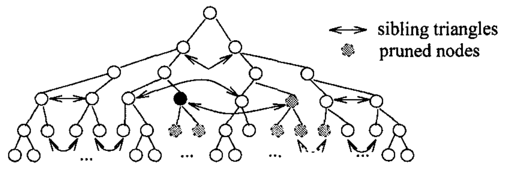

Figure 4: Example of Triangle Merging

We analyze now some properties of the triangulation that will be used to calculate algorithms complexities. The tree nodes dependencies are static and can be expressed in closed-form. For any triangle, one can then compute its *splitting domain* and its *merging domain.* A triangle splitting/merging domain is formed by the set of all triangles that one has to split/merge jointly with this particular triangle. Figure 5 shows an example of splitting domain and merging domain. In the left hand-side of the figure, one can see that splitting the filled triangle t requires 6 forced splits (domain size is 6). In the right hand-side, one can see that merging triangle t requires 29 forced merges (domain size is 29).

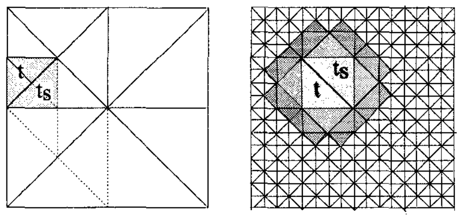

Figure 5: Splitting and Merging Domain

One can see that triangle t and its sibling t_s share the same merging domain. The function $M[i]$ in Figure 6 shows the average merging domain size per level for different triangulations sizes. This function will influence the decimation algorithm performances. One can see that the triangles at level 3 have maximum domain size, but there are only 8 such triangles! Moreover, half of the total number of triangles have minimum domain size. Assume now that the *cost* $C(T)$ of the tree T is the sum of the costs of its nodes. If a node (or equivalently a triangle) has a domain of size $m > 0$, then its cost is $m + 1$, otherwise ($m = 0$) its cost is 1. For any unconstrained tree of depth D, $C(T) = \sum_{i=0}^{D} 2^i$. The constraints induce then a surplus cost for the tree. One can see that, given our constraints (left hand-side of Figure 6), this surplus cost fades out as the tree size increases. This results is important and shows that asymptotically, evaluating a constrained tree do not cost more than evaluating an unconstrained tree.

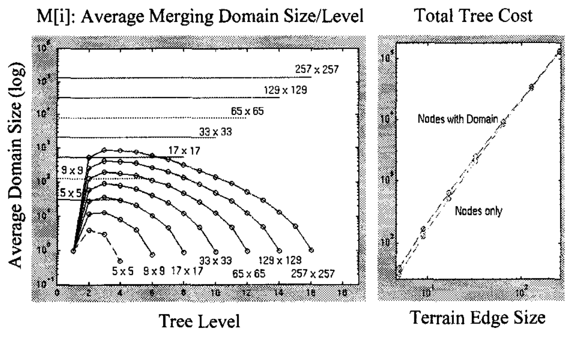

Figure 6: Merging Domain Statistics

Due to overlaps, a particular triangle can belong to several different merging domains. The function $V[i]$ (see Figure 7) gives the percentage of triangles per intersection count. This function is important since it will weight the cost of an iteration step of the decimation algorithm (see Section 3.1).

Unlike in the previous case, a triangle t and its sib-

O[i]: Intersection Count

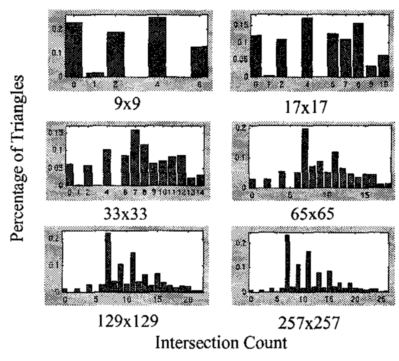

Figure 7: Percentage of Triangles/Intersection Count

ling t_s do not share the same splitting domain. The function in Figure 8 shows the maximum and the average splitting domain size per level. For each triangle in the structure, the highest possible number of forced splits was computed.

S[i]: Average Splitting Domain Size/Level

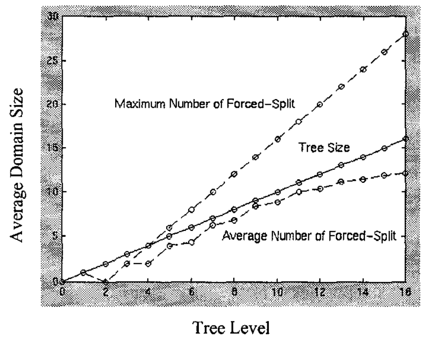

Figure 8: Splitting Domain Statistic

3. APPROXIMATIONS IN A RATE-DISTORTION FRAMEWORK

3.1. Decimation Algorithm

The decimation algorithm is based on a modified implementation of G-BFOS, an optimal tree pruning algorithm [3]. A previous work [2] has shown the opportunity to use G-BFOS for decimation of terrain data, however, the authors did not consider sibling triangles. This consideration is definitively important to preserve the continuity of the terrain and to evaluate correctly the tree functionals (see below). Moreover, the planar approximation of the terrain is produced by interpolating linearly its elevations to generate triangles. This model leads to a non-monotonic behavior of the distortion metric (L_2 norm).

We recall now the G-BFOS algorithm and present the modifications to support constrained trees. We first recall the notations used by Gersho [3]: if T represents a binary tree, we denote by $S \leq T$ any pruned subtree rooted at the same node. More generally, S_t represents a subtree rooted at node t. A tree functional $u(\cdot)$ is a real-valued function on trees, or $u : S \to \mathcal{R}$. The variation $\Delta u(S_t)$ of a tree functional is defined as $\Delta u(S_t) = u(S_t) - u(t)$. Finally, we will consider *linear tree functionals* defined as $u(S) = \sum_{t \in \tilde{S}} u(t)$, where $\tilde{S}$ is the set of leaves of the subtree S. The algorithm works as follow: given two tree functionals u_1 and u_2 and considering the vector-valued function $\mathbf{u}(S) = (u_1(S), u_2(S))$, where u_1 is monotonically increasing ($\Delta u_1(S) \geq 0$) and u_2 is monotonically decreasing ($\Delta u_2(S) \leq 0$), the output of the algorithm is a set of embedded trees $t_0 \leq S_n \leq \ldots \leq S_2 \leq S_1 \leq T$ corresponding to the vertices $\mathbf{u}(t_0), \mathbf{u}(S_n), \ldots, \mathbf{u}(S_1), \mathbf{u}(T)$ bounding the convex hull of all possible tree configurations (optimal solutions), as shown in Figure 9. If $r(S)$ and $d(S)$ are tree functionals returning respectively the rate and the distance in the L_2 norm (distortion) between two subtrees, then typically $u_1(S) = r(S)$ and $u_2(S) = d(S)$. However $d(S)$ is not a monotonically increasing function in our case since, given a particular subtree S_t, it may exist a pruned tree $S_t^\star < S_t$ such that $\Delta d(S_t)\Delta d(S_t^\star) \leq 0$ (non-monotonicity). The algorithm starts at the point $\mathbf{u}(T)$ which is on the convex hull, computes the magnitude $\lambda(t) = \frac{-\Delta d(S_t)}{\Delta r(S_t)}$ of the slope to every other configurations (S_t is the subtree to prune to obtain the new tree configuration) and choose the minimal one. The corresponding subtree is pruned and the magnitudes λ are updated to take into account the new tree configuration. The algorithm is iterated to obtain the successive approximations. It has been shown that configurations on the convex hull are embedded [4]. We prove in Appendix that the non-monotonicity of the functional $d(\cdot)$ do not affect the optimality of the solutions.

As said in section 2.2, merging two triangles (or equivalently pruning a subtree) requires merging all the triangles of its domain. Given a subtree S_t where its root node corresponds to a particular triangle, we denote its domain by $\hat{S}_t$. One can see that, for the algorithm to operate correctly, the tree functionals *must be also evaluated on each triangle domain.* Then, the value λ is redefined as

$$\lambda(t) = -\Delta d(S_t + \hat{S}_t)/\Delta r(S_t + \hat{S}_t)$$

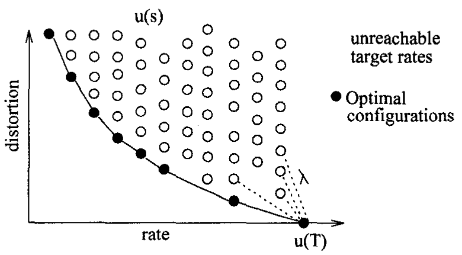

Figure 9: Rate-Distortion Plan and the Convex Hull

The complexity of the initialization step, which consists in computing the initial tree functionals values for each triangle is proportional to $O(n)$ (see Section 2.2). The algorithm complexity is influenced by the size of the domains, since for each triangle belonging to $\hat{S}_t$ one needs also to update all the magnitudes λ of its ancestor nodes (see [3]). The cost of such update depends on the triangle domain size m, thus in total $(m+1)\log n$. Experimentations permit us to evaluate the practical value of the factor $c = m+1$. Since most prunings are done near the leaves, its value remains small (see Figure 10). We conjecture then that $c \log n \approx \log n$. One needs also to update the triangles which domain contains one of the merged triangles. Function I[i] (Figure 7) shows that the expected cost is $mE(I) \approx \log n$. Assuming that we need to merge the n nodes, the expected complexity of the algorithm is $O(n \log n)$.

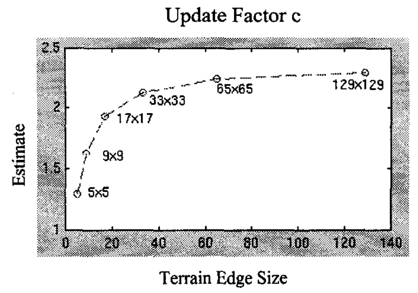

Figure 10: Estimation of the Update Factor c

3.2. Refinement Algorithm

The refinement algorithm is simply based on a greedy tree growing method. It starts with an initial approximation of two triangles (Figure 1) and progressively refines the approximation choosing the available nodes having maximum λ. In this case, the initialization step consists simply in computing the lambda value for the available nodes. Once a node is split, one evaluates the tree functionals for each of its leaves and update the λ magnitude for the triangles which domain contains one of the split triangles. Since this operation is directly dependent of the triangle splitting domain, its cost is approximatively $\log n$ (see Figure 8). Assuming that each triangle is split, the expected complexity of the algorithm is $O(n \log n)$.

Fig	*Rate±5 tri.*	*RTG*	*Refinement*	*Decimation*
a	512	22.54	23.12	24.82
b	1024	25.97	26.64	28.18
c	2048	29.23	29.32	31.43
d	4096	31.80	32.32	34.48

Table 1: Signal-to-Noise Ratio (dB) Results

3.3. Performances and Conclusion

We present here the simplifications of a 257x257 DEM of Zermatt in Switzerland. The original DEM produces a RTG of 131'072 triangles. Figure 12 and 13 present (top-viewed) approximations at different rates whose results are reported in Table 1. The SNR is computed as $10\log_{10}(e_{max}/e_m)$, where e_{max} is the maximum squared error (approximation with two triangles) and e_m is the measured squared error. We added, for comparison, the results obtained using a RTG (uniform grid). Figure 11 compares the RD curves of the refinement (bottom curve) and decimation (top curve) methods.

The decimation algorithm gives good results in term of RD. An interesting question is how far from the optimal solutions are the approximations? Albeit our algorithm was based on an optimal tree pruning algorithm for unconstrained trees, it did not permit us to conclude about optimality. The tree constraints lead to a complex data structure, and pruning optimally such structure is still an open problem. Although the refinement algorithm does not perform much better than a simple planar approximation (RTG) in term of RD, visually significant details (like peaks for instance) are very rapidly emphasized, which makes it very interesting from this point of view (see Figure 12).

Appendix

In this section, we give the proofs for optimality of G-BFOS when using a non-monotonic distortion metric.

Proposition 3.1 Pruning Candidate Necessary Condition
Consider two nodes t_0 and t_1 with level(t_0) < level(t_1),

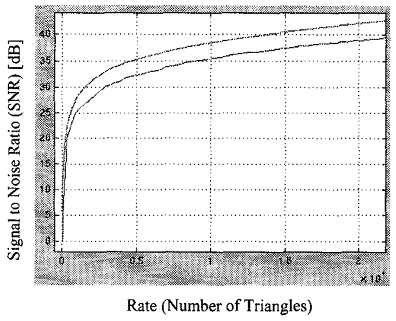

Figure 11: Rate-Distortion Curves for Zermatt

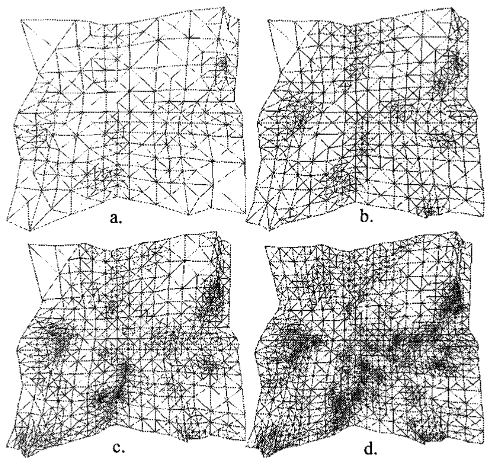

Figure 12: Approximations using Refinement

then assuming a monotonically increasing rate functional r (e.g $\Delta r(S) \geq 0$), the subtree S_{t_1} will be pruned before the subtree S_{t_0} only and only if

$$\frac{\Delta d(S_{t_0})}{\Delta d(S_{t_1})} > \delta > 1$$

where $\delta = \Delta r(S_{t_0})/\Delta r(S_{t_1})$.

proof *For node t_1 to be pruned before node t_0, we need to have*

$$\Delta d(S_{t_0})\Delta r(S_{t_1}) > \Delta d(S_{t_1})\Delta r(S_{t_0})$$

since r is monotonically increasing, we can express $\Delta r(S_{t_0})$ as

$$\Delta r(S_{t_0}) = \delta \Delta r(S_{t_1}) \tag{1}$$

with $\delta > 1$. Then, by replacing the above term, we have that

$$\Delta d(S_{t_0}) > \delta \Delta d(S_{t_1})$$

which gives a necessary condition on node t_0 so that node t_1 is a pruning candidate.

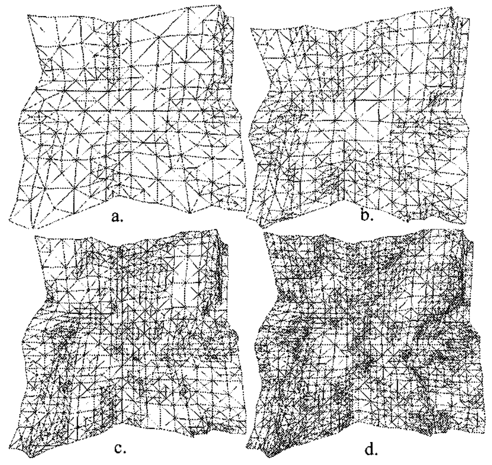

Figure 13: Approximations using Decimation

Proposition 3.2 *Given a node $t^\star$ with $\lambda(t^\star) = min_{t \in S}\lambda(t)$ and the set of ancestor $\mathcal{A} = \{\ t \mid t^\star \in S_t\}$, for all updated nodes $t \in \mathcal{A}$ we have*

$$\lambda(t) > \lambda(t^\star)$$

proof *We have to show that $\lambda(t^\star)$ is a lower bound for all others $\lambda(t) \in S$. We only have to consider the updated nodes (e.g the nodes $t \in \mathcal{A}$), since $t^\star$ is the node with minimal slope in the current tree configuration. We know that node $t^\star$ satisfy proposition 3.1, then for any node $t \in \mathcal{A}$*

$$\lambda(t) = \frac{\Delta d(S_t) - \Delta d(S_{t^\star})}{\Delta r(S_t) - \Delta r(S_{t^\star})} > \frac{\delta \Delta d(S_{t^\star}) - \Delta d(S_{t^\star})}{\delta \Delta r(S_{t^\star}) - \Delta r(S_{t^\star})}$$

$$\frac{(\delta-1)\Delta d(S_{t^\star})}{(\delta-1)\Delta r(S_{t^\star})} = \frac{\Delta d(S_{t^\star})}{\Delta r(S_{t^\star})} = \lambda(t^\star)$$

4. REFERENCES

[1] Michael Garland and Paul Heckbert. Fast polygonal approximation of terrain and height fields. *Internal Report CMU-CS-95-181*, September 1995.

[2] Scott P.Oswald, Kannan Ramchandran, and Thomas S.Huang. Efficient terrain data representation for 3d rendering using the generalized bfos algorithm. *ICIP*, 1997.

[3] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[4] P.A. Chou, T.Lookabaugh, and R.M Gray. Optimal prunning with application to tree-structured source coding and modeling. *IEEE Trans. Information Theory*, 35:299–315, March 1989.