PROGRESSIVE MESH DECOMPOSITION IN THE OPERATIONAL RATE-DISTORTION SENSE USING GLOBAL ERROR

Laurent Balmelli

Visual and Geometric Computing
IBM T.J. Watson Research Center, USA
balmelli@us.ibm.com

ABSTRACT

Given a semi-regular mesh whose subdivision connectivity is obtained with the 4-8 scheme, we propose an algorithm to decompose the mesh into a control mesh and a series of embedded detail meshes. Hence, the output representation is adaptive and progressive. We use a tree-driven, fine to coarse approach to simplify the mesh using vertex decimation. Previous approaches use local error and greedy strategies to simplify meshes. Our method uses global error and a generalized vertex decimation technique borrowed from optimal tree pruning algorithms used in compression. Although global error is used, our algorithm has cost $\Theta(n \log n)$. We show that a direct approach using the same error criterion has at least cost $\Theta(n^2)$. We have a rate-distortion framework: each approximation satisfies a constraint in rate (e.g. number of triangles) while minimizing the distortion (e.g. distance in l_2 norm with the input mesh). The algorithm aims at the optimal approximations in the rate-distortion sense. We analyze the optimality of the algorithm and give several proofs for its properties. Our algorithm can be applied to meshes obtained with other subdivision schemes (e.g. Loop, Catmull-Clark,...) and has also applications in compression and finite element analysis.

Keywords: rate-distortion optimal, mesh simplification, global error, subdivision connectivity

1. INTRODUCTION

1.1 Motivation

Meshes with *subdivision connectivity*, e.g. semi-regular triangulations constructed using iterated subdivision rules (Figures 1a-d), are popular in many applications, such as visualization [11] and finite element analysis [9], to name a few. Their irregular counterparts have also been extensively studied [16], but regular meshes are preferred because of their superior performance and flexibility for processing [13], transmission [17] and compression [15].

A particular class of semi-regular triangulations are *4-8 meshes*. In the strict regular setting, these meshes have been extensively used to visualize terrain data [3, 11, 19, 21]. In this context, 4-8 meshes are also called *quadtree triangulations* because quadtrees are often used to store them [18, 21, 22]. Terrain models are given as amplitude matrices (i.e. the parametrization is implicit) and 4-8 meshes are used to *connect* the vertices (Figures 1a-d). Recently, researchers have also used semi-regular 4-8 meshes to compute approximations of subdivision surfaces [24]. Subdivision surfaces are an increasingly popular representation for piecewise-smooth surfaces. Algorithms for subdivisions surfaces use recursive subdivision rules to *create* vertices from a coarse control mesh. Examples of such rules are provided by Loop [20], Catmull-Clark [6, 8] and Velho-Zorin [24]. Today, the properties of subdivision surfaces are an important area of investigation (e.g. [25]).

In both terrain visualization and subdivision surfaces, researchers often deal with large datasets. Therefore, *simplification algorithms* producing adaptive, multi-resolution representations are an important topic of investigation [11, 19, 21, 3]. Multi-resolution representations of meshes with subdivision connectivity have many advantages over their uniform counterparts. They allow for vertices to be concentrated in detailed regions, leading to efficient descriptions of the shape. Their multiple levels of resolution provide an efficient means to deal with resources-constrained rendering, storage or transmission.

1.2 Contributions and plan

We propose an efficient algorithm to produce adaptive representations of semi-regular 4-8 meshes using vertex decimation and global error. The algorithm decomposes the input into a control mesh plus a series of detail meshes. Global error metrics yield better approximation quality than heuristics based on local error, but are often computationally expensive. We shows that a direct approach using global error requires at least $\Theta(n^2)$ time, where n is the number of vertices in the input mesh. In comparison, our algorithm using the same error criterion has cost $\Theta(n \log n)$. Also, decimation approaches yield better results than their refinement counterparts [12]. In order to choose vertices to decimate, we use an optimal vertex selection technique borrowed from tree pruning algorithms used in compression [7].

We study our mesh simplification algorithm in an operational Rate-Distortion (RD) framework. We attach decimation costs to each vertex. More precisely, we measure a cost in rate, given in terms of triangles, and a cost in distortion, computed in l_2 norm with respect to the original mesh. We give an $\Theta(\log^2 n)$ algorithm to maintain global error estimates for the vertices during the optimization process. The algorithm takes advantage of the hierarchy imposed over the set of vertices by the 4-8 mesh construction (Section 1.4).

We discuss the optimality of the solutions and analyze how optimal vertices are chosen at each decimation step. We use our results to show that approximation errors are most of the time monotonic across rate. We prove that, under certain assumptions, monotonicity is achieved. We explain that suboptimal cases leading to nonmonotonicities exist. However, we show experimentally that the approximation errors returned by our algorithm behave almost always monotonically across rate. We compare our algorithm to a greedy counterpart and show that monotonicity is no more conserved in this case.

We show experimentally the superiority of using global error for the vertices over approaches based on local error. Then, we apply our algorithm to a database of 388 terrains [10] and give approximation results and timings. Timings are given for terrains containing up to two millions triangles.

The paper is organized as follows: In Section 1.3, we review previous work. In Section 1.4, we introduce the hierarchical construction of 4-8 meshes and explain the constraints imposed over the vertices. In Section 2, we introduce our approach: More precisely, in Section 2.1, we present our framework and give the algorithm. Then in Section 2.2, we explain the update method used to maintain global characteristics for the vertices. We evaluate the complexity of the algorithm in Section 2.3. We discuss the optimality of the solutions in Section 3 and give experimental results in Section 4. We summarize our results and conclude this paper in Section 5.

1.3 Previous work

Many simplification algorithms for regular 4-8 meshes have been given in the context of terrain visualization [11, 19, 21]. Unfortunately, all previous approaches use greedy strategies and local error criteria to simplify the model. Lindstrom *et al.* and Pajarola *et al.* use an greedy insertion approach [19, 21], whereas Duchaineau *et al.* adopt a strategy involving both greedy insertion and greedy decimation [11]. In Section 1.4, we explain the difference between greedy selections and more general approaches, as introduced in this work. In the semi-regular setting (e.g. subdivision surfaces), simplification algorithms using local error are also given (see Velho [23]). However, most implementations are based on nonadaptive representations to avoid the added complexity and performance penalty traditionally associated with adaptive schemes.

Each simplification step modifies the model's shape, and some errors must be recomputed. Efficient algorithms for update are necessary to keep acceptable computational cost. In previous work, algorithms are given in order to locally recompute errors after greedy insertions [19, 21] or greedy decimations [11]. However, no such algorithm is described for more general cases of decimation (Section 1.4). Moreover, no low-cost solution exists to efficiently maintain global error estimates for the vertices.

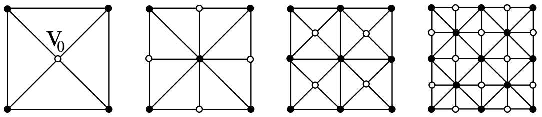
Approaches based on global error are usually more computationally expensive [14, 1]. A typical cost for such algorithm is $\Theta(n^2)$, where n is the number of vertices or triangles. Hence, the cost restricts the size of the dataset to be processed. In [4], we give an analysis in computational complexity of simplification operations, e.g. insertion and decimation, for 4-8 meshes. In the present paper, we use these results to obtain an $\Theta(n \log n)$ algorithm using global error.

Our algorithm is inspired from optimal tree pruning algorithms used to compute adaptive quantizers for compression [5, 7]. A quantizer is often represented with a binary tree and a partial tree corresponds to an adaptive quantizer. The efficiency of a quantizer is evaluated using a rate functional, e.g. returning the average bitrate of the quantizer, and a distortion functional, e.g. measuring the deviation between the original and the quantized samples. Both functionals are evaluated on the tree representation. The algorithm given by Chou *et al.* in [7] computes partial trees achieving minimal distortion for a given rate. They prove that the algorithm finds the optimal quantizers by iteratively decimating the tree. This problem is very similar to our mesh decimation approach, however the hierarchy imposed over the vertices must be conserved and a mesh must be conforming in order to be a valid solution, posing additional constraints to the optimization process.

1.4 4-8 meshes and constraints

We present a simple construction of a regular 4-8 mesh connecting an amplitude matrix z (e.g. terrain data), i.e. the coordinates x, y are implicit. For the sake of clarity, we represent our meshes as tilings of the plane $\mathbb{R}^2$. A 4-8 mesh connecting the dataset is created using the recursive procedure depicted in Figures 1a-d. Initially, a *control mesh* formed with two triangles is connected using the four corner vertices. Hence, the control mesh is a single triangulated quadrilateral (quad). Then, each triangle hypothenuse is bisected to connect a vertex at the midpoint. We denote each connection step by l and Figures 1a-d depict steps $l = 1, 2, 3, 4$, respectively. After $2d$ connection steps, the mesh contains $n = 2 \cdot 4^d$ triangles. The unique vertex inserted at step $l = 1$ (Figure 1a) is called the *root vertex* and is denoted by v_0.

Subdivision surfaces are used to generate semi-regular 4-8 meshes, i.e. with arbitrary topology [24]. A coarse control mesh composed of a small set of triangulated quads (as in Figure 1a) fixes the topology and is used as an initial mesh. Then, subdivision rules are used to create new vertices connected on each quad, as in Figures 1a-d.



(a) $l = 1$ (b) $l = 2$ (c) $l = 3$ (d) $l = 4$

Figure 1. Connection of a matrix of amplitudes z using the 4-8 scheme: Initially, a control mesh formed by two triangles is created using the corner vertices. Then, triangle hypothenuses are bisected to connect a vertex at the midpoint. Each connection step is denoted by l and (a),(b),(c) and (d) show steps $l = 1, 2, 3$ and 4, respectively. At each step, the newly connected vertices are depicted in white.

The iterative procedure used to connect the vertices naturally yields *hierarchical constraints* over the set of vertices. The vertices at each level form a set of triangles, eventually embedded in a set of finer triangles formed by the vertices connected at the next step. Hence, the construction defines a hierarchy of triangulations (e.g. Figure 1a-d), as well as a hierarchical set of vertices. Both hierarchies can be efficiently modeled with a quadtree [2]. However, since neighbor quads share common vertices on their edges, nodes in the tree of vertices share common children. Hence, subtrees are actually locally connected to their neighbors. Consequently, pruning operations have to be defined accordingly [2]. Our algorithm operates on vertices and we refer to their hierarchy simply as tree for simplicity.

Consider the root vertex v_0 connected in Figure 1a. A decimation *perserving the hierarchy* operates as follows: When v_0 is decimated (e.g. in the mesh of Figure 1d), the pair of triangles split by v_0 (Figure 1a) is recovered. Call v a vertex, then M_v denotes the set of vertices that must be removed jointly in order to recover the original pair of triangles and then preserve the hierarchy. We call this set *merging domain*. This set forms a forest of subtrees in the tree of vertices. Consequently when decimating v_0, the root of the tree of vertices, all the vertices in the mesh are also decimated, i.e. M_{v_0} contains all the vertices in the mesh. Assume that $|M_v|$ counts the number of vertices in a domain, then $|M_{v_0}| = n$. A merging domain is attached to each vertex in the mesh. For the vertices v connected at the step depicted in Figure 1d, $M_v = \{v\}$ since it suffices to remove v to recover the corresponding pair of triangles in Figure 1c. Such decimation is referred to as a *greedy* case of decimation (as used in [11]) and $M_v = \{v\}$ is a leaf in the tree of vertices. In contrast, a *general* case of decimation refers to when an arbitrary domain (i.e. with $|M_v| > 1$), formed by a forest of subtrees, can be removed. We explain in Section 2.1 that allowing general decimation is the key to perform optimal choices in the rate-distortion sense.

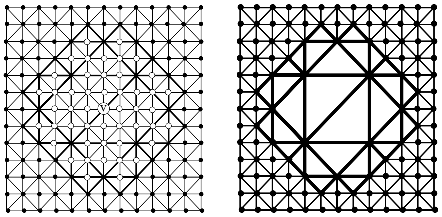

(a) (b)

Figure 2. General decimation: (a) Example of merging domain M_v. The white vertices depict the vertices in the domain. (b) Support of the merging domain: Set of remaining triangles when M_v is decimated.

Figure 2a depicts an example of general decimation: The white vertices belong to the domain M_v attached to the central vertex v. Assume that $2d$ steps were used to construct the mesh, then v was connected at step $2d-4$ in order for the domain to have the size shown in the figure. Therefore the coarser the connection step, the larger the merging domain. Figure 2b depicts the triangulation when M_v is decimated. We call *support* the remaining set of triangles tiling the merging domain. Moreover, we denote by $\breve{M}_v$ a decimated merging domain. Note that the decimation preserves the hierarchy and that the resulting mesh is conforming.

Preserving the hierarchy imposed by the construction constrains simplification algorithms to search a smaller set of possible approximations. However, this approach has the following advantages: Each simplified mesh is represented by a partial tree, hence no effort is needed to retriangulate the dataset after removing a vertex, since all successive approximations are embedded. Such representation is naturally progressive and the connectivity can be stored in a compact way. The resulting meshes are efficient for compression.

The most important benefit for preserving the hierarchy is that global error estimates for the vertices can be computed at low cost. In Section 2, we explain that this can be done in $\Theta(\log^2 n)$. In Section 4.1, we show experimentally that our hierarchy-preserving method using global error leads to far superior results in quality than their counterparts using local error (e.g. [11, 19, 21, 23]). Moreover, we show that there is only a small degradation when compared to algorithms not preserving the hierarchy, i.e. searching a larger space of approximations. Recall that these algorithms do not benefit from the properties described above.

2. ALGORITHM

2.1 General approach

This section introduces our $\Theta(n \log n)$ algorithm based on general decimation and global error. We show in Section 2.3 that it computationally outperforms a direct approach using the same error criterion. We apply it to a regular mesh built on a matrix of amplitudes z, e.g. terrain data. The same algorithm applies to semi-regular meshes. In this case, extraordinary vertices are not decimated in order to preserve the mesh topology. We use *mesh functionals* $u : M_v \to \mathbb{R}$ to compute properties for v *over its merging domain* M_v. We use two mesh functionals R and D: R is called the *rate* and counts the number of triangles, whereas D measures the distance in l_2 norm between the original surface and an approximation (Appendix A). Hence, for each v we compute the vector value $\underline{u}(M_v) = (R(M_v), D(M_v))$.

Call $M_0 = \{v_0, \ldots, v_{N-1}\}$ the input mesh and M a simplified version, then the problem to solve is

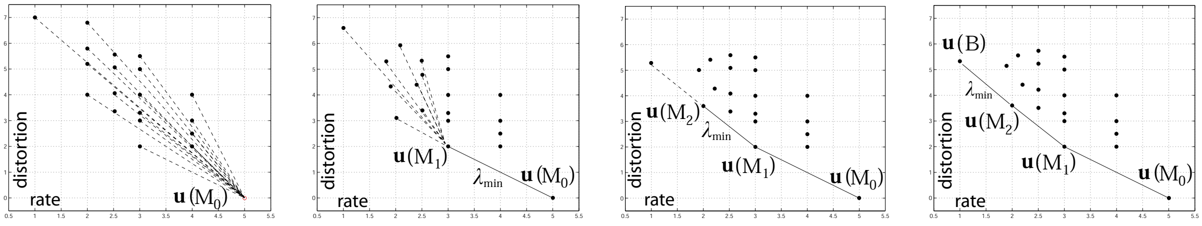
$$D(R) = \min_{|M| \leq |M_0|} \{D(M) | R(M) \leq r\}, \qquad (1)$$

where r denotes a constraint in rate. A progressive representation for M is found by solving the problem for all values $2 \geq r \geq n$. For a rate budget r, the solution $(R(M_i), D(M_i))$ returned by $D(R)$ satisfies the constraint at minimal incurred distortion. The set of solutions, denoted by

$$|\mathcal{B}| < \ldots < |M_1| < |M_0|, \qquad (2)$$

where $\mathcal{B}$ is the control mesh, corresponds to a series of embedded approximations. The solutions are embedded in the sense that any approximation can be reconstructed from a coarser solution only by splitting a set of triangles.

Each simplified mesh $(R(M_i), D(M_i))$ can be represented as a position in the space of values spanned by R and D. This space is called *rate-distortion (RD) plane* (Figures 3a-d). The set of all possible approximations is a cloud of positions in the RD plane. Each optimal configuration is represented by a position $\underline{u}(M_i) = (R(M_i), D(M_i))$ on the curve bounding the convex hull of all configurations (Figure



(a) (b) (c) (d)

Figure 3. Algorithm: (a) Initially, the variations $\Delta\underline{u}(M_v)$ and the slopes $\lambda(v)$ are computed for each vertex. (b) The vertex with minimal slope $\lambda(v) = -\Delta D(M_v)/\Delta R(M_v)$ is chosen and decimated. The RD characteristics of the ancestor vertices of M_v are updated, hence the corresponding positions in the RD plane are displaced. (c),(d) The algorithm is iterated. The algorithm aims at the solutions on the curve lowerbounding the set of all possible configurations. These approximations are optimal in the operational RD sense.

3d). This curve is called the *operational RD curve* and the approximations on this curve are optimal in the operational RD sense.

We define the *variation* of a functional as

$$\begin{aligned}\Delta\underline{u}(M_v) &= \underline{u}(M_v) - \underline{u}(\breve{M}_v)\\ &= (R(M_v) - R(\breve{M}_v), D(M_v) - D(\breve{M}_v))\\ &= (\Delta R(M_v), \Delta D(M_v)).\end{aligned} \tag{3}$$

The variation $\Delta\underline{u}(M_v)$ is the change in rate and distortion when M_v is decimated. Therefore, a vector $\Delta\underline{u}(M_v)$ links two configurations in the RD plane. More precisely, given a mesh over which $\Delta\underline{u}(M_v)$ is computed, the vector leads to the configuration obtained by decimating M_v. Hence, $\lambda(v) = -\Delta D(M_v)/\Delta R(M_v)$ is the trade-off between rate and distortion when M_v is decimated and represents a slope in the RD plane (Figure 3a).

The algorithm proceeds as follows: Initially, the variations $\Delta\underline{u}(M_v)$ and the slopes $\lambda(v)$ are computed (Figure 3a) and stored for each vertex. Note that $\Delta D(M_v) < 0$ (Appendix A), hence $\lambda(v) > 0$. Additionally, we use a variable $\lambda_{\min}$ at each vertex to store the minimal slope amongs all its descendants. At each iteration the vertex v with minimal $\lambda(v)$ is chosen and M_v is decimated (Figure 3b). General decimation allows us to select the optimal M_v in the rate-distortion sense: The selection *minimize the increase in distortion while maximizing the decrease in rate*. The decimation changes the characteristics (i.e. in rate and distortion) of a set of vertices. We call these vertices *ancestors*[1] and denote this set by A_{M_v}. Two types of ancestors a exist: the vertices such that $M_v \subset M_a$ (i.e. towards the root) and the vertices such that M_v and M_a partially overlap. In [2], we explain how to find these vertices efficiently. In particular, we prove that

$$|A_{M_v}| \in \Theta(\log n). \tag{4}$$

Also, we show that $\Theta(\log^2 n)$ operations are sufficient to update all the ancestor values. We explain our update mechanism in Section 2.2. Once the RD characteristics of the ancestor vertices are updated, the corresponding positions $\underline{u}(M_a)$ in the RD plane are displaced. The algorithm is iterated until the configuration with minimal rate is reached (Figures 3c-d).

We give the algorithm below. In our application, we use a regular 4-8 mesh and the configuration with minimal rate has two triangles (subdivided by the root vertex v_0 as shown in Figure 1a). As pointed out in Section 1.4, M_{v_0} contains all the vertices in the mesh. Therefore, since our characteristics are global, the global rate and the global distortion are given by $R(M_{v_0})$ and $D(M_{v_0})$, respectively. Hence, in line 7 we use $R(M_{v_0})$ to test the rate of the current approximation. Similarily, we could use $D(M_{v_0})$ to obtain configurations satisfying a maximum error. The total complexity of the algorithm is computed in Section 2.3.

ALGORITHM

```
1  initialization:
```

---

[1] Recall that subtrees in the tree of vertices are locally connected to their neighbors (Section 1.4). Hence, in contrast with common definitions for ancestors in trees, our ancestors are not strictly confined to vertices towards the root.

```
2   for all $v$
        3   COMPUTE $\Delta D(M_v)$, $\Delta R(M_v)$
        4   $\lambda(v) \leftarrow \frac{-\Delta D(M_v)}{\Delta R(M_v)}$
5 iteration:
    6   $i = 1$   (counter for the approximations.)
    7   while $R(M_{v_0}) > 2$
            8   $v^\star = \arg\min_{v \in M} \lambda(v)$
            9   $M_i \longleftarrow M_{i-1} \setminus M_{v^\star}$
            10  UPDATE $\Delta D(M_a)$ AND $\Delta R(M_a)$, $\forall a \in A_{M_v}$
    11  end
12 end
```

2.2 Update of global error

In this section, we present the algorithm used to update the functional variations of the vertex characteristics. The algorithm has cost $\Theta(\log^2 n)$ and is derived from an algorithm based on an inclusion-exclusion principle used to compute merging domain intersections [4]. Assume that a domain M_v is decimated, then for each vertex $w \in M_v$, we find a set of parents $a \in A_w$ (see below) and the variations $\Delta\underline{u}(M_a)$ are replaced by

$$\Delta\underline{u}(M_a) - \Delta\underline{u}(M_w), \forall a \in A_w, \forall w \in M_v. \quad (5)$$

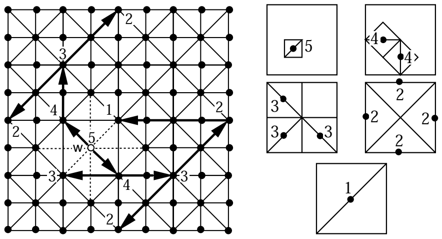
The unions of all sets A_w, $\forall w \in M_v$ is a set of ancestors of M_v. We explain how to find the sets A_w below.

We use the algorithm below to update the functional variations computed at the initialization (lines 1-4 of the algorithm in Section 2.1) during the mesh optimization. In the algorithm, the updated ancestor functionals are denoted $\Delta\underline{u}'(M_a)$. The algorithm finds the set of ancestors A_{M_v} and updates the characteristics using (5). More precisely, a set of *parents* for each vertex $w \in M_v$, denoted by A_w, is traversed. The parents are found using a fine to coarse traversal of the tree of vertices. An example of traversal is shown in Figure 5. The index next to each vertex is the connection step l (Figures 1a-d). The larger the index the finer the connection step. Finally, note that an important property related to the parents of a vertex, is

$$\forall w \in M_v, v \in A_w, \quad (6)$$

i.e. all the vertices $w \in M_v$ have v as a parent.

The update (5) is performed for each parent. Once the set of parents for w has been visited, w is decimated. Hence the algorithm is used to update the ancestor variations and to jointly decimate the domain M_v. Hence, it replaces lines 9 and 10 of the algorithm given in Section 2.1. Consider the decimation of M_v and the update of A_{M_v}. Furthermore, assume that v is connected at step l. The set of vertices $w \in M_v$ has to be iteratively decimated starting at the vertices in the domain having the *finest connection step*. From a tree point of view, this is equivalent to iteratively prune the subtrees formed by M_v starting at the leaves. Therefore, if M_v spans m levels, then the vertices at step $l+m$ are decimated first, followed by the vertices at step $l+m-1$, and so on.



(a) (b)

Figure 5. Parents of vertex w (A_w): (a) To find the parents of w (connected at $l = 5$), the vertex hierarchy is traversed fine to coarse. The path (arrows) follows recursively the tree of vertices towards the root. Each traversed vertex split a pair of triangles depicted in (b). In both (a) and (b), the index next to each vertex indicates the connection step (Figure 1a-d).

ALGORITHM

```
for ALL AVAILABLE VERTICES $w \in M_v$ CONNECTED AT STEP $l + m \ldots l$
    for ALL $a \in A_w$
        $\Delta\underline{u}'(M_a) = \Delta\underline{u}(M_a) - \Delta\underline{u}(M_w)$
    end
    decimate $w$
end
```

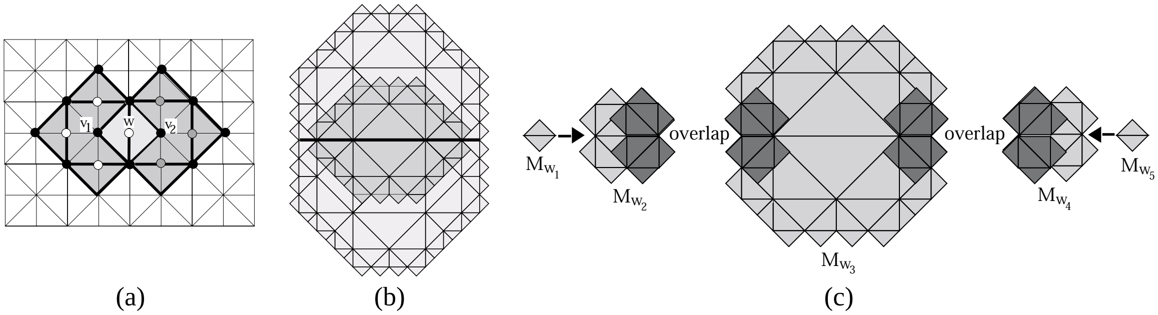


Figure 4. Update of the global error: (a) When M_{v_1} and M_{v_2} are decimated, the global characteristics (rate and distortion) of the mesh are $\underline{u}(M_{v_0}) - \Delta\underline{u}(M_{v_1} \cup M_{v_2})$. (b) Intersection between two large merging domains. For clarity, the merging domains are represented using their support. The intersection is depicted in dark shade and the thick line represents the boundary between the domains. (c) Decomposition of the intersection in part (b): The intersection is expressed as the union of a set of smaller domains M_{w_i}, $i = 1:5$. Note that the pairs of domains M_{w_2},M_{w_3} and M_{w_3},M_{w_4} overlap (represented in dark shade).

We explain now how global characteristics are maintained using the above algorithm. We start with a simple example, then we address the general case. To do so, we summarize the problem of finding merging domain intersections. A complete analysis is found in [2].

Consider the following example: After the initialization phase (lines 1-4 of the algorithm in Section 2.1), the functional values $\Delta\underline{u}(M_v)$ are global since no domain has been yet decimated. Consider M_{v_1} and M_{v_2} as depicted in Figure 4a. Clearly, after decimating both domains, the global characteristics of the mesh (rate and distortion) are

$$\Delta\underline{u}(M_{v_0}) - \Delta\underline{u}(M_{v_1} \cup M_{v_2}), \tag{7}$$

where v_0 denotes the root vertex. Recall that v_0 is used to measure the characteristics of the complete mesh since M_{v_0} contains all the vertices. To evaluate (7), we need to compute $\Delta\underline{u}(M_{v_1} \cup M_{v_2})$. Unfortunately, we have $M_{v_1} \cap M_{v_2} \neq \emptyset$, thus

$$\Delta\underline{u}(M_{v_1} \cup M_{v_2}) < \Delta\underline{u}(M_{v_1}) + \Delta\underline{u}(M_{v_2}). \tag{8}$$

However, $M_w = M_{v_1} \cap M_{v_2}$, as shown in Figure 4a. Hence, the surplus of $\Delta\underline{u}(M_{v_1}) + \Delta\underline{u}(M_{v_2})$ is $\Delta\underline{u}(M_w)$ because every triangle tiling the support of M_w is also a triangle of either the support of M_{v_1} or M_{v_2}. Therefore,

$$\Delta\underline{u}(M_{v_1} \cup M_{v_2}) = \Delta\underline{u}(M_{v_1}) + \Delta\underline{u}(M_{v_2}) - \Delta\underline{u}(M_w). \tag{9}$$

We show now that the algorithm computes (9) after the successive decimation of M_{v_1} and M_{v_2} (the order has no importance). For M_{v_1}, the algorithm first decimates w and the three remaining vertices connected at the same step (depicted in white in Figure 4a). Hence, following (6), the updated functional characteristics at v_1, v_2 and v_0 (root vertex) are, respectively,

$$\begin{aligned}
&\Delta\underline{u}(M_{v_1}) - \Delta\underline{u}(M_w) - \sum_{k \in M_{v_1}, k \neq v_1, k \neq w} \Delta\underline{u}(M_k),\\
&\Delta\underline{u}(M_{v_2}) - \Delta\underline{u}(M_w),\\
&\Delta\underline{u}(M_{v_0}) - \Delta\underline{u}(M_w) - \sum_{k \in M_{v_1}, k \neq v_1, k \neq w} \Delta\underline{u}(M_k).
\end{aligned} \tag{10}$$

Then, the algorithm decimates v_1 and the updated value at v_0 is

$$\Delta\underline{u}(M_{v_0}) - \Delta\underline{u}(M_{v_1}). \tag{11}$$

Note that v_2 is not affected by the decimation of v_1 since $v_2 \notin A_{v_1}$. Now M_{v_2} is decimated, starting with the three available vertices $k \in M_{v_2}, k \neq w$, depicted in gray in Figure 4a. The updated values at v_2 and v_0 are, respectively,

$$\begin{aligned}
&\Delta\underline{u}(M_{v_2}) - \Delta\underline{u}(M_w) - \sum_{k \in M_{v_2}, k \neq v_2, k \neq w} \Delta\underline{u}(M_k),\\
&\Delta\underline{u}(M_{v_0}) - \Delta\underline{u}(M_{v_1}) - \sum_{k \in M_{v_2}, k \neq v_2, k \neq w} \Delta\underline{u}(M_k).
\end{aligned} \tag{12}$$

Finally, the algorithm decimates v_2 and the updated value at v_0 is

$$\Delta\underline{u}(M_{v_0}) - \underbrace{(\Delta\underline{u}(M_{v_1}) + \Delta\underline{u}(M_{v_2}) - \Delta\underline{u}(M_w))}_{\Delta\underline{u}(M_{v_1} \cup M_{v_2})}, \tag{13}$$

which shows that (9) is obtained, i.e. the characteristics computed at v_0 are global.

The above example shows that the algorithm uses an inclusion-exclusion principle to compute the global error at each vertex. We presented a simple example where the intersection between the decimated domains $\breve{M}_{v_1}$ and $\breve{M}_{v_2}$ is the singleton domain $M_w = \{w\}$. In general, intersections between domains are more complex. For example, consider the intersection between the two domains in Figure 4b. In the figure, the domains are depicted using their support for clarity. Following (4), we have $\Theta(\log n)$ possible arrangements for intersections. Hence the examples in Figures 4a and 4b are just particular cases. Figure 6 depicts another example of arrangement.

The intersection in Figure 4b can be decomposed in terms of smaller merging domains, as shown in Figure 4c. The number of domains is proportional to the size of the intersection. In the example of Figure 4a, a single domain M_w is sufficient to express the intersection, whereas in Figure 4c, the intersection is written

$$M_{w_1} \cup M_{w_2} \cup M_{w_3} \cup M_{w_4} \cup M_{w_5}. \tag{14}$$

Unfortunately, the domains M_i forming the intersection overlap (Figure 4c), i.e. $M_{w_2} \cap M_{w_3} \neq \emptyset$ and $M_{w_3} \cap M_{w_4} \neq \emptyset$. Therefore to compute (14), we use an inclusion-exclusion approach to resolve all embedded intersections. In [4], we call this problem *merging domain intersections* and we show that (14) is computed in $\Theta(\log n)$ time. Also, we show that the intersections between a decimated domain $\breve{M}_v$ and the domains of all its ancestors A_{M_v} are computed in $\Theta(\log^2 n)$ time.

The update algorithm given at the beginning of this section automatically computes all intersections between M_v and the domains of its ancestors. As a result, the computed values at each vertex v are *the global variation in RD characteristics after M_v is decimated*. Hence, the characterstics $\Delta\underline{u}(M_{v_0})$, i.e. at the root vertex, are the global RD characteristics of the mesh.

We conclude now this section with the following general example: Assume that all vertices in M_v are decimated except for v. Therefore, following (6) the updated variations at v and v_0 are, respectively,

$$\begin{array}{l} \Delta\underline{u}(M_v) - \sum_{w \in M_v, w \neq v} \Delta\underline{u}(M_w), \\ \Delta\underline{u}(M_{v_0}) - \sum_{w \in M_v, w \neq v} \Delta\underline{u}(M_w). \end{array} \tag{15}$$

Assume that v is now decimated, then using (5), the variation at v_0 is now

$$\Delta\underline{u}(M_{v_0}) - \Delta\underline{u}(M_v), \tag{16}$$

which corresponds to the global characteritics of the mesh after the decimation of M_v.

2.3 Complexity

The cost of the algorithm in Section 2.1, i.e. computing a complete decomposition of the mesh, is found as follows: In [4], we show that merging domains have size $\Theta(\log n)$ on average. Thus, assuming a mesh of n triangles, the initialization has cost $\Theta(n \log n)$. At each iteration, the optimal vertex $v^\star$ (having minimal slope $\lambda(v^\star)$) is found in $\Theta(\log n)$ operations using the values $\lambda_{\min}$. The cost to decimate M_v and update the variations for the vertices in A_{M_v} is $\Theta(\log^2 n)$. Also, $\Theta(\log n)$ values $\lambda(v)$ and $\lambda_{\min}$ are recomputed and the algorithm is iterated. On average, $n/\Theta(\log n)$ steps are necessary to decompose the mesh, since at each step, $\Theta(\log n)$ vertices on average are decimated (average size of merging domains). Hence, the cost to compute the full decomposition is $\Theta(n \log n)$.

A direct algorithm needs to recompute the global error over each ancestors' domain. A lowerbound for this update is obtained as follows: We have roughly $\Theta(4^{l+1})$ vertices at step l and $l \in \Theta(\log n)$ ancestors exists. Call a any such ancestor, then $|M_a|_\triangle(i,n) \approx n/4^{i-1}$, $1 \leq i \leq l$. Therefore, a lowerbound for the complexity is

$$\sum_{i=0}^{\log_4 n} 4^i \sum_{j=0}^{i} \frac{n}{4^j} = \frac{16}{9}n^2 - \frac{1}{3}n\log_4 n - \frac{7}{9}n \in \Theta(n^2). \tag{17}$$

Note the above approximation accounts only for the ancestors a such as $M_v \subset M_a$. Accounting for the update of the ancestors whose domain partially overlaps does not change the order of magnitude. However, this evaluation is complex due to the $\Theta(\log n)$ cases of overlap, i.e. arrangements for intersections, one has to deal with (Section 2.2).

3. DISCUSSION OF OPTIMALITY

In this section, we discuss the optimality of the algorithm. First, we explain how an optimal vertex is chosen at each decimation step (Section 3.1). Second, we discuss issues related to intersections between domains and how optimal choices are affected (Section 3.2). Third, we explain how the monotonicity of the approximation errors (i.e. distortions) are conserved across rate (Section 3.3).

3.1 Optimal choice

Recall that our rate functional measures the number of triangles, hence the functional is monotonically increasing with the mesh size. Consider now the following example: Consider two vertices v_1 and v_2 such that $v_2 \in M_{v_1}$, i.e. $\Delta R(M_{v_2}) < \Delta R(M_{v_1})$. Furthermore, assume that

$$\Delta D(M_{v_2}) > \Delta D(M_{v_1}). \tag{18}$$

Such a case is possible with the l_2 or the l_∞ norms since both are nonmonotonic with the mesh size [14, 2]. Recall that $\Delta D(M_{v_1}) < 0$ and $\Delta D(M_{v_2}) < 0$ (Section 2.1). If v_2 is decimated, then following (5) and (6), we have that

$$\Delta D(M_{v_1}) - \Delta D(M_{v_2}) > 0, \tag{19}$$

and the new slope

$$\lambda(v_1) = \frac{-(D(M_{v_1}) - D(M_{v_2}))}{(\Delta R(M_{v_1}) - \Delta R(M_{v_2}))} < 0, \tag{20}$$

i.e. the sign of the slope changes. In consequence, M_{v_1} will be the optimal domain to decimate at the next iteration, and the global error will decrease, i.e the RD curve will be nonmonotonic. We say that M_{v_1} is a *nonmonotonic merging domain* with respect to M_{v_2}, i.e. decimating M_{v_2} creates a nonmonotonicity at v_1.

The algorithm avoids the above situation using general decimation (Section 1.4) as follows: If the decimation of a domain M_v provokes a nonmonotonicity at a parent of v, then the algorithm will decimate the domain of the parent instead. In Proposition 3.1, we show that only the rate functional needs to be monotonic and that the distortion functional can be arbitrary (i.e. monotonic or nonmonotonic), both with respect to the mesh size, in order for the algorithm to make the optimal choice.

Proposition 3.1. *Given v_1 and v_2, such that $v_2 \in M_{v_1}$, and $\Delta R(M_v) \geq 0$ (monotonicity of the rate functional), then M_{v_2} is decimated before M_{v_1} if and only if*

$$\frac{\Delta D(M_{v_1})}{\Delta D(M_{v_2})} > \delta > 1, \tag{21}$$

where $\delta = \Delta R(M_{v_1})/\Delta R(M_{v_2})$. When v_1 does not meet condition (21), the domain M_{v_1} is said to be nonmonotonic with respect to M_{v_2}.

Proof. For M_{v_2} to be decimated before M_{v_1}, we need to have

$$\Delta D(M_{v_1})\Delta R(M_{v_2}) > \Delta R(M_{v_1})\Delta D(M_{v_2}). \tag{22}$$

Since the functional R is monotonically increasing, we can write

$$\Delta R(M_{v_1}) = \delta \Delta R(M_{v_2}), \delta > 1 \tag{23}$$

Then, replacing (23) in (22) yields

$$\Delta D(M_{v_0}) > \delta \Delta D(M_{v_1}). \tag{24}$$

□

3.2 Intersection between domains and optimal choice

In Section 2.1, we explained that a type of ancestors is the vertices a such that M_v and M_a partially overlap. Figure 6 illustrates such a case. Assume now that M_v is the optimal domain to decimate at some iteration and that M_a is nonmonotonic with respect to M_w. Since $w \in M_v$, M_w is decimated jointly to M_v. Following (6), the decimation creates a nonmonotonicity at M_a. The above example shows that, due to the overlap between domains, the algorithm using general decimation cannot avoid nonmonotonicities across rate.

We perform experiments using matrices of amplitudes z (terrain data [10]) and compare our algorithm to a greedy counterpart using global error, i.e. we force the algorithm to perform greedy decimation only (Section 1.4). Hence, only "leaf" domains M_v, i.e. $M_v = \{v\}$, are decimated, preventing the algorithm to make optimal choices as explained in Section 3.1. We find

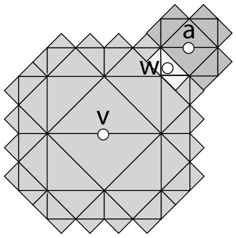


Figure 6. Suboptimal choice of the algorithm: $w \in M_v \cap M_a$ and M_a is a nonmonotonic merging domain with respect to M_w (see Proposition 3.1). Decimating M_w provokes a nonmonotonicity at M_a.

that, although the algorithm using general decimation cannot avoid monotonicity, the RD curve (top curve in Figure 7) is very stable compared to the one obtained with its greedy counterpart (bottom curve in Figure 7). For the greedy version, nonmonotonicities often occur at low rate. We conclude that, for our dataset, few cases of nonmonotonicity are observed.

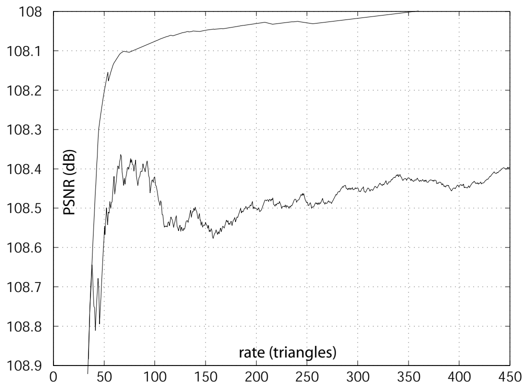


Figure 7. Comparison of nonmonotoncities of the RD curve between the algorithm using general decimation (top curve) and its greedy counterpart (bottom curve): The curve obtained with the greedy version is unstable at low rates.

3.3 Monotonicity

In this section, we show that a rate functional monotonic with the mesh size is necessary and sufficient for the approximation errors to be monotonic across rate. To do so, we assume that no suboptimal choice, as explained in Section 3.3, is made. Under the above assumptions, we actually show that the slopes λ, corresponding to optimal choices made during the mesh decomposition, monotonically decrease across rates. This fact is stated in the following proposition:

Proposition 3.2 (Monotonicity of the RD Curve). *Consider that $\Delta R(M_v) \geq 0$ and $\Delta D(M_v)$ is arbitrary, $\forall v$. Call $M_{v^\star}$ the optimal domain to decimate and assume that there is no ancestor $a \in A_{M_{v^\star}}$ such that M_a is a nonmonotonic domain with respect to a vertex $w \in M_{v^\star} \cap M_a$. Then, for all updated vertices $a \in A_{M_{v^\star}}$ we have*

$$\lambda(a) > \lambda(v^\star). \tag{25}$$

Proof. We have to show that $\lambda(v^\star)$ is a lowerbound for $\{\lambda(v)\}_{v \in M}$. We only have to consider the updated vertices $a \in A_{M_v^\star}$, i.e.

$$\lambda(a) = \frac{\Delta D(M_a) - \Delta D(M_{v^\star})}{\Delta R(M_a) - \Delta R(M_{v^\star})}, \forall a \in A_{M_v^\star} \tag{26}$$

Moreover, $M_{v^\star}$ satisfies Proposition 3.1, then

$$\begin{aligned} \lambda(a) &> \frac{\delta \Delta D(M_{v^\star}) - \Delta D(M_{v^\star})}{\delta \Delta R(M_{v^\star}) - \Delta R(M_{v^\star})} \\ &> \frac{(\delta - 1)\Delta D(M_{v^\star})}{(\delta - 1)\Delta R(M_{v^\star})} \\ &> \frac{\Delta D(M_{v^\star})}{\Delta R(M_{v^\star})} > \lambda(v^\star). \end{aligned} \tag{27}$$

□

4. EXPERIMENTAL RESULTS

We organize our experimental results as follows: In Section 4.1, we first demonstrate the efficiency of our global error estimate using a simpler graphic model, namely *the polyline*, i.e piecewise-linear polynomial. This allows us to run a large number of experiments and to compare our approach, i.e. using general decimation and global error, with several algorithms.

Recall that the hierarchy imposed over the vertices by the 4-8 mesh construction restricts the space of approximations (Section 1.4). This constraint can be applied to the polyline model using a binary tree, i.e. a decimation (or insertion) algorithm must preserve the tree hierarchy when optimizing the model. We refer to these algorithms as *constrained* (Section 4.1). We compare our algorithm to two constrained approaches using local error: vertex insertion and vertex decimation.

We also compare our results to the optimal, *unconstrained*, approximations obtained using dynamic programming. In the mesh case, this could be seen as the optimal solutions using irregular triangulations. However, Agarwal *et al.* [1] have demonstrated that finding these solutions is NP-Hard. In [2], we explain and compare approaches to approximate polylines in detail. Finally, we apply our algorithm to terrain data (Section 4.2).

4.1 Comparison with methods using local error

We use the polyline model to test the efficiency of our global error estimate for the vertices. Figure 8 shows an example of constrained decimation. The top curve is the original one and has 7 interior knots. These knots are iteratively decimated and the bottom curve can be seen as the *control curve*. The binary tree constraining the decimation is depicted using bold lines in the figure.

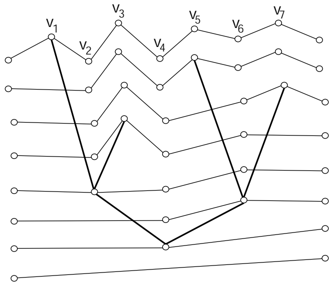


Figure 8. Successive approximations of a polyline using constrained knot decimation. Knots are iteratively decimated from top to bottom. The binary tree constraining the decimation is depicted using bold lines.

Our experimental results are shown in Figure 9: The graph shows a comparison of the RD curves. The rate is computed as the number of segments forming an approximation and the distortion is computed in l_2 norm with respect to the original curve. We run the experiment using 256 curves obtained from terrain data and we average the results of each algorithm. To compute the average, we normalize the errors and fix the gain to 50 dB.

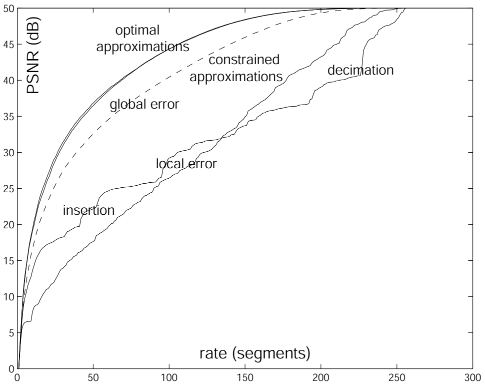


Figure 9. Comparison of RD curves: The rate is computed as the number of segments forming an approximation and the distortion is computed in l_2 norm with respect to the original curve. The top curve is obtained using dynamic programming. The dashed curve is obtained with our algorithm. Finally, the two bottom curves compare the decimation and insertion approaches using local error.

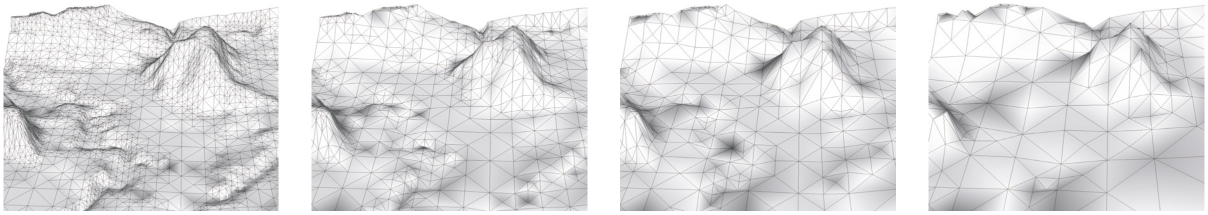
The top curve shows the errors of the optimal approximations found using dynamic programming. The dashed curve is obtained with our algorithm using general decimation and global error. The two bottom curves are obtained with greedy insertion and greedy decimation using local error. Both approaches accumulate errors through the iterative approximation process. Hence, the insertion method achieves better quality than the one using decimation at low rates and, symmetrically, the decimation method achieves better quality than the one using insertion at high rates.

4.2 Decomposition of terrain data

We apply now our algorithm to terrain data. We run experiments on 388 terrains [10]. Each matrix of amplitudes has size 257×257, hence each model has 131’072 triangles. The errors obtained using each terrain is averaged using the maximum error computed as the distance in l_2 norm between the control mesh and the original mesh. Then, the maximum gain is fixed to 50 dB and the average curve is shown in Figure 11. An example of terrain decomposition is shown in Figure 10.

We give now some timings when computing a complete decomposition. We use a PC equipped with a Pentium III 500 Mhz and 256 Mb of RAM. The implementation is in C++ and is not optimized. We store the 4-8 meshes with the quadtree described in [18]. For the timings, we use a subset of the available terrains in [10]. We use 50 terrains with sizes up to two millions triangles.

The results in Table 1 are obtained by measuring the average decomposition time for each terrain. Separate measurements are given for the initialization and the



(a) (b) (c) (d)

Figure 10. Progressive mesh decomposition using global error: Approximation using (a) 6400 triangles (PSNR 33.8 dB), (b) 1600 triangles (PSNR 25.9 dB), (c) 800 triangles (PSNR 22.2 dB) and (d) 400 triangles (PSNR 18.46 dB).

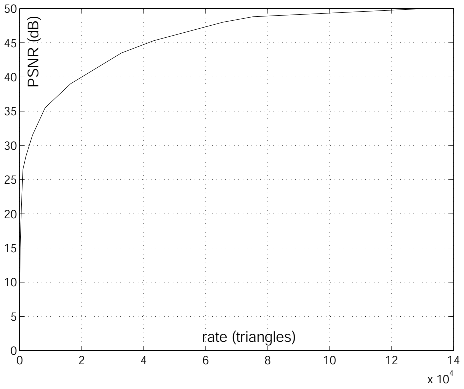


Figure 11. Average RD curve: We apply our algorithm to a database of 388 terrains [10]. To compute the average curve, the errors obtained using each terrain are normalized and the maximum gain is fixed to 50 dB.

decomposition (lines 1-4 and lines 5-12 of the algorithm in Section 2.1, respectively).

triangles n	init. time (s)	decomposition (s)
2’048	0.003	0.057
8’192	0.011	0.30
32’766	0.05	1.7
131’072	0.24	11.4
524’288	1.07	48
2’097’152	4.81	238

Table 1. Average decomposition times: The timings are obtained using a PC equipped with a Pentium III 500 Mhz and 256 Mb of RAM.

5. CONCLUSION

We presented an efficient $\Theta(n \log n)$ simplification algorithm to compute progressive decomposition of 4-8 meshes. Our algorithm uses general decimation and global error. We showed experimentally that, using global error, high quality approximations are obtained. Moreover, the low computational cost of the algorithm allows us for processing large datasets.

We showed that global error estimates for the vertices can be computed in $\Theta(\log^2 n)$ time during the mesh optimization. We gave an algorithm based on the solution to a problem called *merging domain intersections* presented in [4].

Our algorithm aims at finding the approximations whose RD characteristics are the curve bounding the convex hull of all possible solutions. This curve is called operational RD curve and the approximations are optimal in the operational RD sense. Although we did not prove the optimality of our solutions, we discussed the choices made by the algorithm during the optimization process. We showed experimentally that the RD curve corresponding to our solutions is most of the time monotonic across rate. Also, we showed that under certain assumptions monotonicity across rate is obtained.

A. EVALUATION OF MESH FUNCTIONALS

We compute the costs in rate for each domain, measured as the number of triangles, in closed form using the results in [4]. More precisely, in [4] we give closed-forms to compute $R(M_v)$ and $R(\breve{M}_v)$.

We use the squared l_2 norm as a measure of distortion between the original mesh and the approximations. More precisely, each vertex is projected in its corresponding triangles in the support of the domain. Consider the simple case of a matrix of amplitudes z. Figure 12 shows how errors are measured on a triangulated quadrilateral. The total distortion is evaluated similarly on the domain M_v.

Consider the triangulated quadrilateral in the left-hand side of Figure 12. Each amplitude z_i is projected in a triangle of the support. For example, z_2 is projected in the triangle formed by vertices $(x_1, y_1, z_1), (x_5, y_5, z_5), (x_3, y_3, z_3)$ (right-hand side of Figure 12). Denote by z_i' a projected amplitude, hence the error for each vertex is then given by $\delta_i = |z_i - z_i'|^2$. To evaluate the error $D(\breve{M}_v)$, all the vertices in the domain are projected in the support of M_v. The distortion functional is computed as $D(\breve{M}_v) = \sum_{v \in M_v} \delta_v$ and $D(M_v) = 0$. hence $\Delta D(M_v) = -D(\breve{M}_v)$, i.e. intially $\Delta D(M_v) < 0$ for all the vertices.

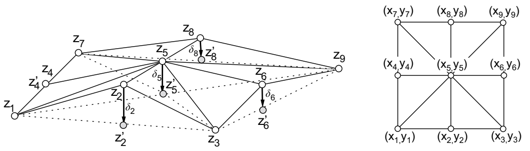


Figure 12. Computation of the error in squared l_2 norm for a triangulated quad. The left part shows a top view of the triangualted quad.

REFERENCES

[1] P.K. Agarwal and P.K. Desikan. An efficient algorithm for terrain simplification. *Proceedings ACM-SIAM Sympo. Discrete Algorithms*, pages 139–147, 1997.

[2] L. Balmelli. Rate-distortion optimal mesh simplification for communications. *Ph.D dissertation, Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland.*, 2000.

[3] L. Balmelli, S. Ayer, and M. Vetterli. Efficient algorithms for embedded rendering of terrain models. *Proceedings of IEEE Int. Conf. Image Processing (ICIP)*, 2:914–918, October 1998.

[4] L. Balmelli, T. Liebling, and M. Vetterli. Computational analysis of 4-8 meshes with application to surface simplification using global error. *Proceeding of the 13th Canadian Conference Computational Geometry*, August 2001.

[5] L. Breiman, J.H. Freidman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. The Wadsworth Statistics/Probability Series, Belmont,CA; Wadsworth, 1984.

[6] E. Catmull. A subdivision algorithm for computer display of curved surfaces. *Ph.D dissertation, Report UTEC-CSs-74-133, Computer Science Department, University of Utah*, December 1974.

[7] P. Chou, T. Lookabaugh, and R. Gray. Optimal pruning with application to tree-structured source coding and modeling. *IEEE Transactions on Information Theory*, 35(2):299–315, March 1989.

[8] J.H. Clark. A fast algorithm for rendering parametric surfaces. *Proceedings of SIGGRAPH*, pages 289–99, 1979.

[9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, algorithms and applications*. Springer-Verlag, 2000.

[10] Office Federal de Topographie. Pixelkarte 1:25000 cd rom 1,2,3. *CH-2084 Wabern*, Mai 1997.

[11] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. *Proceedings of IEEE Visualization*, 1997.

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

[13] I. Guskov, W. Swelden, and P. Schröder. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH*, pages 325–334, 1999.

[14] P.S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. *Carnegie Mellon University Technical Report*, May 1997.

[15] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. *proceedings of SIGGRAPH*, pages 271–278, 2000.

[16] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry*, 14(1-3):5–24, 1999.

[17] U. Labsik, L. Kobbelt, R. Schneider, and H.-P. Seidel. Progressive transmission of subdivision surfaces. *Computational Geometry*, 15(1-3):25–39, 2000.

[18] L.Balmelli, J.Kovačević, and M. Vetterli. Quadtree for embedded surface visualization: Constraints and efficient data structures. *Proceedings of IEEE Int. Conf. Image Processing (ICIP)*, 2:487–491, October 1999.

[19] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH*, pages 109–118, 1996.

[20] C. Loop. Smooth subdivision surfaces based on triangles. *Master's thesis, University of Utah, Department of Mathematics*, 1987.

[21] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. *Proceedings of IEEE Visualization*, pages 299–305, 1998.

[22] H. Samet. *Application of Spatial Data Structures: Computer Graphics, Image Processing and GIS*. Addison-Wesley Publishing Company, 1990.

[23] L. Velho. Four-face cluster simplification. *Proceedings of Shape Modeling International*, 2001.

[24] L. Velho and D. Zorin. 4-8 subdivision. *Computer-Aided Geometric Design, Special Issue on Subdivision Techniques.*, 2001.

[25] D. Zorin. A method for analysis of C^1-continuity of subdivision surfaces. *SIAM Journal of Numerical Analysis*, 37(4):1677–1708, 2000.