Graphical Models **64,** 230–257 (2002)
doi:10.1006/gmod.2002.0578

Mesh Optimization Using Global Error with Application to Geometry Simplification

Laurent Balmelli[1]

Visual and Geometric Computing Group, IBM Research, T.J. Watson Center, Hawthorne, New York
E-mail: balmelli@us.ibm.com

Martin Vetterli[2]

Institute for Communication Systems, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
E-mail: Martin.Vetterli@epfl.ch

and

Thomas M. Liebling

Institute for Mathematics, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Received September 15, 2001; revised May 14, 2002

This paper presents a linear running time optimization algorithm for meshes with subdivision connectivity, e.g., subdivision surfaces. The algorithm optimizes a model using a metric defined by the user. Two functionals are used to build the metric: a rate functional and a distortion (i.e. error) functional. The distortion functional defines the error function to minimize, whereas the rate functional defines the minimization constraint. The algorithm computes approximations within this metric using jointly *global error* and an optimal vertex selection technique inspired from optimal tree pruning algorithms used in compression. We present an update mechanism, that we name *merging domain intersections* (MDIs), allowing the control of global error through the optimization process at low cost. Our method has application in progressive model decomposition, compression, rendering, and finite element methods. We apply our method to geometry simplification and present an algorithm to compute a decomposition of a model into a multiresolution hierarchy in $O(n \log n)$ time using

[1] This work was completed while the author was a Ph.D. student at the Laboratory for Audio-Visual Communications at the Ecole Polytechnique Fédérale de Lausanne, Switzerland. To whom correspondence should be addressed. Phone: +1 914 784 7762. Fax: +1 914 784 7667.

[2] Martin Vetterli is also with the Department of Electrical Engineering and Computer Science, UC Berkeley, Berkeley, California 94720.

1524-0703/02 $35.00

global error, where n is the number of vertices in the full-resolution model. We show that a direct approach, i.e. not using MDIs, recomputing global error has at least cost $O(n^2)$. We analyze the optimality of the algorithm and give several for its properties. We present results for semi-regular meshes obtained from approximation of subdivision surfaces whose connectivity is obtain from (triangulated) quadrilateral quadrisection (e.g. 4-8 or Catmull-Clark subdivision).

Key Words: rate-distortion optimal; mesh optimization; global error; subdivision surfaces; geometry simplification.

1. INTRODUCTION

1.1. Motivation

Meshes with *subdivision connectivity*, e.g., semi-regular triangulations constructed using iterated subdivision rules, are popular in many applications, such as visualization [10] and finite element mesh generation, to name a few. Their irregular counterparts have also been extensively studied; however, semi-regular meshes are preferred because of their superior performance and flexibility for editing [4], transmission [16], and compression [13].

In the strictly regular setting, e.g., subdivision of a single quadrilateral, meshes with subdivision connectivity are used to visualize terrain data [10, 20, 23], as shown in Fig. 1a. Terrain models are given as elevation matrices (i.e., the parameterization is implicit) and the mesh is used to *connect*, i.e., triangulate, the input matrix. Semi-regular meshes are used to represent surfaces with arbitrary topology. The topology is fixed using an initial coarse mesh with irregular connectivity (Fig. 2a). The vertices forming this mesh are called *extraordinary vertices*. Semi-regular meshes are used to compute approximations of subdivision surfaces (Fig. 1b). Subdivision surfaces are an increasingly popular representation for piecewise-smooth surfaces. Algorithms used to compute subdivision surfaces use recursive subdivision rules to *create* vertices. Examples of such rules are provided by Kobbelt ($\sqrt{3}$) [14], Loop [22], Catmull–Clark [6, 8], and Velho-Zorin (4-8) subdivision [28]. Today, the properties of subdivision surfaces are an important area of investigation (e.g., see [29]).

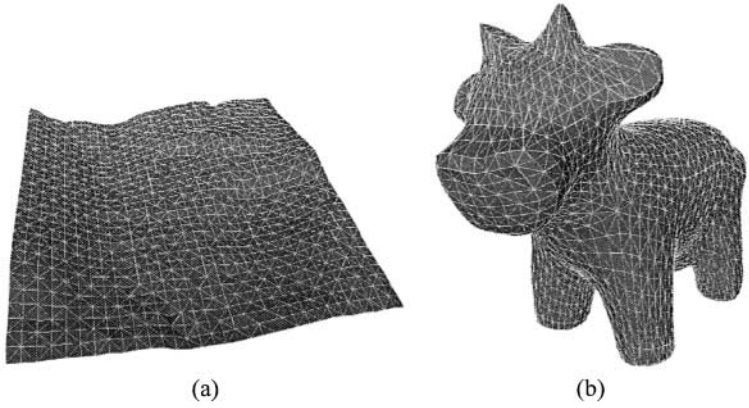


FIG. 1. Meshes with subdivision connectivity. (a) Regular setting. (b) Semi-regular setting: The arbitrary topology is obtained by subdividing an initial coarse mesh with irregular connectivity.

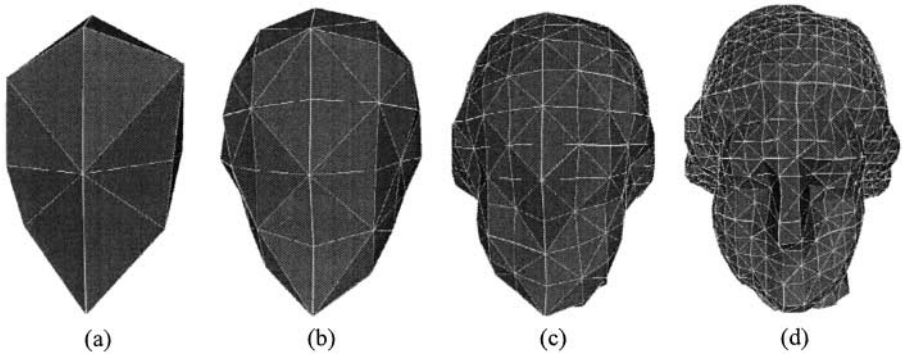


FIG. 2. Construction of a subdivision surface from an input model having irregular connectivity (Venus model). (a) Initial coarse mesh with irregular connectivity: The initial mesh fixes the topology of the subdivision surface. (b), (c), (d) Successive subdivision steps: At each step detail vectors computed from the input model are added to the vertices.

A particular class of meshes with subdivision connectivity are *4-8 meshes*. In these meshes, the valence of the vertices alternates between four and eight. In Figs. 2a–2d, we show an example of 4-8 subdivision: Fig. 2a shows the initial coarse mesh, and Figs. 2b–2d show successive subdivision steps. At each step, detail vectors computed from the original (i.e., irregular) model are added to the vertex locations. The computation of detail vectors is typically part of a *remeshing process* used to convert the irregular input model into a mesh having subdivision connectivity (e.g., see [18]). Finally, note that a 4-8 mesh is also used to triangulate the terrain in Fig. 1a, although in this case, as stated previously, the dataset given by the input matrix is just triangulated using 4-8 connectivity and no additional vertices are generated. In this context, 4-8 meshes are also called *quadtree triangulations* since quadtrees are often used to store them [17, 23, 25].

Optimization algorithms for meshes with subdivision connectivity hold a great interest for many applications. For example, geometry simplification algorithms are used to obtain adaptive, multiresolution representations and are an important topic of investigation [10, 20, 21, 23]. Other applications include compression [13], editing [4], finite element mesh generation, evaluation of bidirectional reflectance distribution functions, and radiance [24]. In this work, we give an application in geometry simplification leading to multiresolution hierarchy. Multiresolution 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 resource-constrained rendering, storage, or transmission.

An important concern is the error metric used to optimize the model. In order to keep low computational complexity, most current implementations use local error criteria. Methods using a global error metric are usually computationally more intensive or are simply infeasible without efficient processing tools. This work introduces a set of processing tools to optimize a mesh using global error while preserving low computational complexity.

1.2. Contributions and Plan

We propose an efficient linear running time optimization algorithm for meshes with subdivision connectivity. Our algorithm can be used to address several different problems

on these meshes, such as progressive model decomposition, compression, rendering, and finite element analysis. We present an application in geometry simplification of semi-regular 4-8 meshes, whose connectivity is obtained from triangulated quadrangular quadrisection. In this case, the algorithm decomposes the input mesh into an initial coarse mesh (e.g., as in Fig. 2a) plus a series of detail meshes forming a multiresolution hierarchy. Our algorithm uses a generalized vertex decimation technique (see below) and global error to optimize the representation. Global error metrics yield better approximation quality than heuristics based on local error, but are often computationally expensive. Decimation approaches also yield better results than their refinement counterparts [11]. We show that a direct algorithm using the same error criterion and vertex selection requires at least $O(n^2)$ time, where n is the number of vertices in the input mesh. In comparison our algorithm computes a single approximation in $O(n)$ time and transforms the input into a multiresolution hierarchy (i.e., a fully progressive decomposition is computed) in $O(n \log n)$ time. Therefore, the low computational complexity of our algorithm allows the use of global error at low cost for processing large models.

Our method is inspired from tree-pruning algorithms used in modeling of adaptive quantize for compression [5, 7, 15]. More precisely, our algorithm extends the optimization technique described by Chou *et al.* in [7] to geometry processing. We take advantage of the vertex hierarchy obtained from the subdivision process and apply an optimal selection technique. Our semi-regular models are stored in a hierarchical data structure, namely a forest of quadtrees. A vertex can be selected at any level in the hierarchy. Therefore a vertex, as well as all its descendants, can be considered at any optimization step. In our geometry simplification algorithm, this is equivalent to considering sets of vertices (as well as individual ones) for decimation. In this context, we call this operation *generalized decimation* as opposed to considering only one vertex at a time, usually done in previous work. We explain that only generalized decimation allows for making optimal choices in the operational Rate-Distortion (RD) sense (e.g. see Fig. 7).

We study our mesh optimization algorithm in an operational RD framework. The RD framework requires the user to define two functionals: a rate functional and a distortion functional. These functionals drive the optimization process and are defined according to the application. In order to perform geometry simplification, we define the rate functional as the number of vertices in the mesh and the distortion functional as the distance in l_2 norm between a simplified mesh and the input model. We give a $O(\log^2 n)$ algorithm based on *merge domain intersections* (MDIs) [2] to recompute global error estimates at the vertices during the optimization process.

It is important to note that our algorithm seeks to obtain the optimal solutions in the operational rate–distortion sense. It is well known in rate–distortion theory that these solutions cannot be obtained using refinement, i.e., vertex insertion methods [11]. Consequently our method uses a bottom-up, fine to coarse approach.

We discuss the optimality of the solutions and analyze how optimal vertices are chosen at each optimization step. In particular, we prove that generalized decimation leads to optimal selection of the vertices. We use our results to show that approximation errors are almost always monotonic across rate. We explain that monotonicity is achieved under a certain assumption. With this assumption and our proof for optimal selection, our algorithm reduces to the algorithm given by Chou *et al.* in [7]. In [7], the authors prove the optimality of their algorithm, which allows us to suggest the optimality of our method as well. Without assumption, 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. To demonstrate this property, we compare our geometry simplification algorithm to a limited version using standard decimation (i.e., decimating one vertex at a time) and show that monotonicity is no more conserved in this case. Then, we experimentally show the superiority of global error over approaches based on local error in two steps. First we run an experiment on a large set of polylines and compare several vertex selection approaches. Second, we apply our algorithm to a standard set of semi-regular meshes and compare it to a standard refinement approach as used in [20, 23].

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 optimization algorithm. Then in Section 2.2, we explain the update method used to maintain global error estimates at 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. In Section 5 we give a summary of our results and discuss additional applications for our optimization framework.

1.3. Previous Work

Many simplification algorithms for regular 4-8 meshes have been given in the context of terrain visualization [10, 20, 23]. Previous approaches use greedy strategies and local error criteria to simplify the model. Lindstrom *et al.* and Pajarola *et al.* use an efficient insertion approach [20, 23], whereas Duchaineau *et al.* adopt a strategy involving both insertion and greedy decimation [10]. Their techniques are elegant and very well suited for visualization of a terrain dataset. In particular, Lindstrom and Pascucci recently proposed many interesting improvements to handle very large terrain datasets [21]. Efficient out-of-core techniques are also described by Lindstrom in [19]. Simplification algorithms using local error are also given in the semi-regular setting, e.g., subdivision surfaces [27]. However, most implementations for subdivision surfaces are based on nonadaptive representations to avoid the added complexity and performance penalty traditionally associated with adaptive schemes. Although very efficient for visualization, these methods are less suited to address optimization problems requiring high quality approximations. Our algorithm gives a means to address optimally large optimization problems using global error at low computational cost. Typical applications are model decomposition, compression, and rendering. In contrast, our method may be less adapted for visualization since a fine-to-coarse approach is used to optimize the representation. Our approach is therefore complementary to previous work using refinement and local error.

Each simplification step modifies the model's shape, and some errors must be recomputed. Efficient update algorithms are necessary to maintain acceptable computational cost. In previous work, algorithms are given in order to locally recompute errors after insertions [20, 23] or decimations [10] of individual vertices. However, no such algorithm is described for more general cases of decimation, e.g., when sets of vertices are considered jointly, as in our method. Moreover, no low-cost solution existed prior to this work to efficiently maintain global error estimates for the vertices.

Greedy strategies using local error have linear complexity with the input when a single approximation is needed. If a full decomposition of the dataset is desired (e.g., to store the

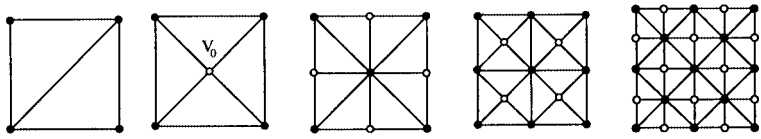
model in a progressive format), then the minimal cost is $O(n \log n)$ [10, 20, 23]. Wavelet filter algorithms such as the ones proposed by Schröder and Sweldens [26] can also be used to compute approximations (e.g., see [13]). In this case the minimal cost of computing a full decomposition is also $O(n \log n)$. Our algorithm has the same computational characteristics as the ones described above. The computational cost of these algorithms only differs in a constant, as discussed below.

Any algorithm performing a full decomposition of a mesh into a multiresolution hierarchy cannot do better than $O(n \log n)$, since this problem is equivalent to sorting an array. Standard wavelet algorithms are by far the most efficient since the use of filters requires almost no comparisons. However, adaptive filters (i.e., filters that vary locally) are more expensive since they need to query region characteristics in order to set up filtering parameters. Refinement approaches [20, 23] are usually computationally efficient. However, the efficiency depends on the "look-ahead" capabilities one wants to provide the algorithm with. As greedy approaches are shortsighted by nature, some care must be taken in order to avoid falling into local minima of the optimized error function.

As said previously, our algorithm is inspired from optimal tree pruning algorithms used to compute adaptive quantizers for compression. A quantizer is often represented with a binary tree whereas 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: The hierarchy imposed over the vertices must be conserved. In addition, a mesh must be conforming in order to be a valid solution, imposing additional constraints on the optimization process. Therefore, our algorithm extends previous methods used for quantizers [7] to handle additional constraints in geometry processing.

1.4. 4-8 Meshes and Constraints

We present a simple construction of a regular 4-8 mesh connecting an elevation 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 Figs. 3a–3e. Initially, an *coarse mesh* formed with two triangles is connected using the four corner vertices. Hence, this mesh is a single triangulated quadrilateral (quad). Then, each triangle hypotenuse is bisected to connect a vertex at the



(a) $l = 0$ (b) $l = 1$ (c) $l = 2$ (d) $l = 3$ (e) $l = 4$

FIG. 3. Triangulation of an elevation matrix z using 4-8 connectivity. Initially, an initial coarse 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.

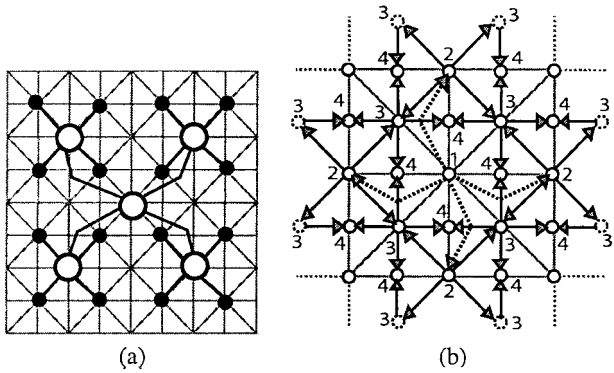


FIG. 4. Hierarchies obtained by 4-8 construction. (a) The set of hierarchical meshes is naturally represented using a quadtree: Each tree node represents a triangulated quadrilateral (e.g. top arrow). A subtree (bottom arrow) corresponds to a larger region. (b) The hierarchical set of vertices is represented with a directed acyclic graph (DAG).

midpoint. We denote each connection step by l and Figs. 3a–3e depict steps $l = 0, 1, 2, 3$, and 4, respectively. After $2d$ connection steps, the mesh contains $n = 4^d + 2^{d+1} + 1$ vertices. The unique vertex inserted at step $l = 1$ (Fig. 3b) is called the *root vertex* and is denoted by v_0.

The same iterative procedure is used for subdivision surfaces in the semi-regular settings. However, in this case new vertices are generated using subdivision rules [6, 8, 14, 22, 28]. Each face of the initial coarse mesh (e.g., Fig. 2a) is similar to the quad shown in Fig. 3a. Figures 2a–2d show steps $l = 2$, 4, and 6, respectively.

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 obtained at the next step. Hence, the construction defines a hierarchy of triangulations (e.g., Figs. 3a–3e), as well as a hierarchical set of vertices. The hierarchy of triangulations is naturally described using a quadtree. In Fig. 4a, we represent a quadtree and its associated triangulation. Each node represents a triangulated quad such as the one shown in Fig. 3c. The root node corresponds to the mesh obtained at step $l = 2$ (Fig. 3c), whereas its four descendants represent the mesh in Fig. 3e. Therefore, the mesh in Fig. 4a is represented by the leaves of the quadtree (black nodes) depicted on top of it.

The hierarchical set of vertices forms a directed acyclic graph (DAG). Figure 4b shows the DAG connecting the vertices in Fig. 3e. The index next to each vertex is the connection step l. These indices correspond to the ones in Fig. 3. Each vertex is linked to its descendants by arrows. Dotted lines are used to suggest vertices in neighbor quads. We can see that, except for terminal vertices (i.e., without descendants), each vertex has four descendants (not taking into account border effects).

When a vertex is decimated we remove all its descendants as well. This condition ensures that the resulting approximation is conforming, i.e., free of cracks. Note that the DAG connecting the vertices is still very similar to a tree structure. Since vertices are hierarchically connected as a DAG, our geometry simplification algorithm actually performs graph pruning. This is precisely how our algorithm extends previous tree pruning algorithms used in compression [5, 7, 15].

Both hierarchies (i.e., triangles and vertices) are duals. Any pruned DAG of the vertices also corresponds to a partial quadtree representing an adaptive and conforming triangulation.

The corresponding quadtree is known as *restricted* in the literature [25]. Pajarola explains how to obtain a restricted quadtree from a selected set of vertices and use it for terrain visualization [23]. Since the optimization operations described in the paper preserve the vertex hierarchy, our meshes can always be represented with restricted quadtrees. More precisely since our algorithm operates on semi-regular meshes, our meshes are represented by a restricted forest of restricted quadtrees. Note that a restricted forest ensures that the mesh is conforming across quads forming the semi-regular representation. For simplicity, in the rest of this paper we simply refer to a restricted quadtree as a tree.

Consider the root vertex v_0 connected in Fig. 3b. A decimation *preserving the hierarchy* operates as follows: When v_0 is decimated (e.g., in the mesh of Fig. 3e), the pair of triangles split by v_0 (Fig. 3b) is recovered. More generally, 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 a *merging domain*. This set includes v as well as all its descendants. The descendants are found as suggested in the DAG depicted in Fig. 4b. Consequently when decimating v_0, all the vertices in the mesh are also decimated; i.e., M_{v_0} contains all the vertices in the mesh. Let us denote by $|M_v|$ 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 Fig. 3e, $M_v = \{v\}$ since it suffices to remove v to recover the corresponding pair of triangles in Fig. 3d. This is equivalent to removing a terminal vertex in the vertex hierarchy (Fig. 4b). In contrast, a generalized case of decimation refers to when an arbitrary domain (i.e., with $|M_v| > 1$) formed by a DAG of vertices can be removed. We explain in Section 2.1 that allowing generalized decimation is the key to performing optimal choices in the rate–distortion sense.

Figures 5a–5b depict an example of generalized decimation: The gray vertices belong to the domain M_v attached to the central vertex v. These vertices are part of the DAG rooted at vertex v. Assume that $2d$ steps are used to construct the mesh; then v is 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 5b depicts the triangulation when M_v is decimated. We call *support* the remaining set of triangles tiling the merging domain (gray shade in Figs. 5a and 5b). We denote by $\check{M}_v$ the set containing the vertices used to connect the triangles tiling the support (depicted by the white vertices in Fig. 5b). Note that the decimation preserves the hierarchy and that the resulting mesh is conforming.

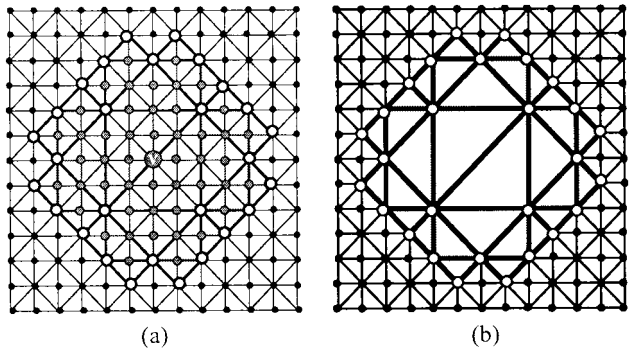


FIG. 5. Generalized decimation. (a) Example of merging domain M_v: The gray vertices belong to the domain. (b) Support of the merging domain: Set of remaining triangles when M_v is decimated. The set $\check{M}_v$ is represented by the white vertices and connect the triangles tiling the support.

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 a representation is naturally progressive and the connectivity can be stored in a compact way. The resulting meshes are efficient for compression: Most recent works in this field [13] use this constraint to avoid encoding explicitly the model connectivity and obtain high compression ratios. Works in terrain by Pajarola [23] and Lindstrom *et al.* [20, 21] also preserve the hierarchy of the vertex set.

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 $O(\log^2 n)$. 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., [10, 20, 23, 27]).

2. ALGORITHM

2.1. General Approach

This section introduces our algorithm based on generalized decimation and global error. We apply our algorithm to semi-regular meshes having 4-8 connectivity. Our models are stored using the forest of quadtrees described in [2]. More precisely, each subdivided face forming the initial coarse mesh (Figs. 2a–2d) is stored using a quadtree, and any approximation of the model is represented by a partial forest. Recall that all quadtrees in the forest are restricted, as well as the forest itself in order to obtain a conforming mesh. Also, 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. The algorithm uses two mesh functionals: a distortion functional and a rate functional. These functionals define an optimization metric. Let us denote by D and R these functionals, respectively, and call $M_0 = \{v_0, \ldots, v_{N-1}\}$ the input mesh and M an optimized version; then the problem solved by the algorithm is

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

where r denotes a constraint. Hence, D is the function to minimize under constraints imposed by R. We prove in Section 3.1 (Lemma 3) that the only condition imposed on the functionals is that R must be monotonically decreasing with the size of M_v, i.e., $|M_{v_1}| \le |M_{v_2}| \Rightarrow R(M_{v_1}) \le R(M_{v_2})$, whereas D can be arbitrary. The latter condition leaves the greatest freedom to define the functional D according to the application.

The definition of a mesh functional is tailored to the application. Typically, the functionals are used to evaluate properties of the mesh structure (number of vertices, errors, etc.). Also, when a function is defined over the mesh, the mesh functionals can be used to optimize the model according to associated properties (texture, environment maps, etc.). Note that since our optimization method is based on a hierarchical structure (tree or DAG), the functionals can be efficiently evaluated.

We apply our algorithm to geometry simplification. Hence in our application, R counts the number of vertices, whereas D measures the distance in the l_2 norm between the original surface and an approximation (Appendix A). For each v we compute the vector value $\underline{u}(M_v) = (R(M_v), D(M_v))$. A progressive representation (i.e., multiresolution hierarchy)

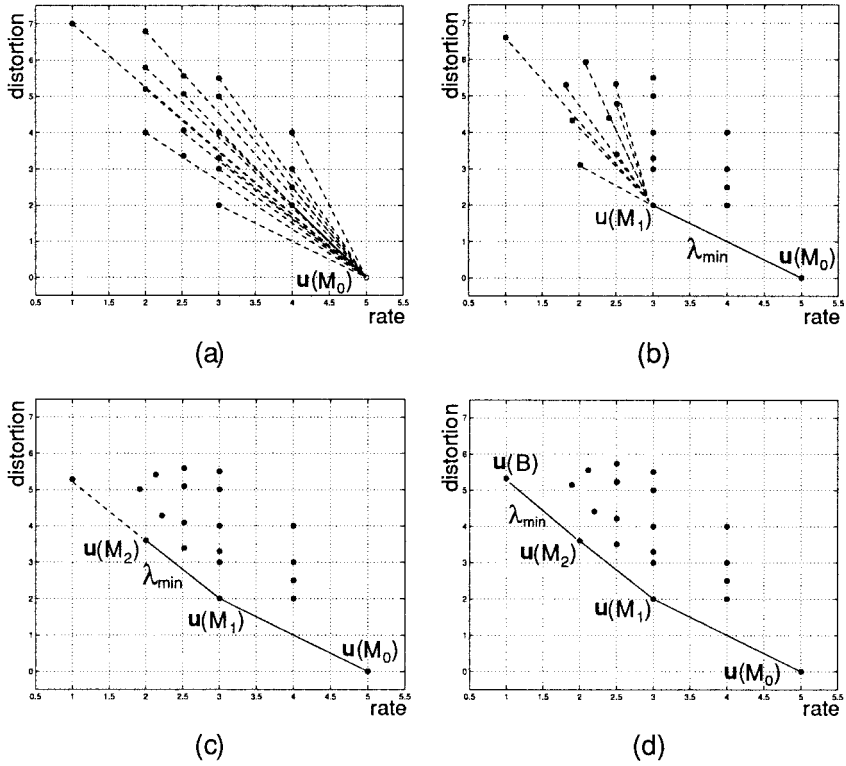


FIG. 6. Algorithm. (a) Initially, the variations $\Delta \underline{u}(M_v)$ and the slopes $\lambda(v)$ are computed for each merging domain M_v. (b) The domain with minimal slope $\lambda(v) = -\Delta D(M_v)/\Delta R(M_v)$ is the optimal candidate for optimization. The RD characteristics of the ancestor 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.

for M is found by solving Eq. (1) 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

$$|\mathcal{B}| < \cdots < |M_1| < |M_0|, \tag{2}$$

where $\mathcal{B}$ is the initial coarse mesh (e.g., Fig. 2a), 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 the *rate–distortion (RD) plane* (Figs. 6a–6d). 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 lowerbounding the convex hull of all configurations (Fig. 6d). The approximations on this curve are optimal in the operational RD sense.

We define the *variation* of a functional as

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

In our application, the variation $\Delta\underline{u}(M_v)$ is the change in rate and distortion when M_v is decimated. More generally, $\Delta\underline{u}(M_v)$ computes the difference between the functionals evaluated on the domain M_v and those evaluated on the support $\breve{M}_v$. 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 tradeoff between rate and distortion when M_v is decimated and represents a slope in the RD plane (Fig. 6a).

The algorithm proceeds as follows: Initially, the variations $\Delta\underline{u}(M_v)$ and the slopes $\lambda(v)$ are computed (Fig. 6a) and stored for each vertex. Note that $\Delta D(M_v) < 0$ (Appendix A), and hence $\lambda(v) > 0$. Additionally, we use a variable $\lambda_{\min}$ at each vertex to store the minimal slope among all its descendants. At each iteration the vertex v with minimal $\lambda(v)$ is chosen and M_v is decimated (Fig. 6b). Generalized decimation allows us to select the optimal M_v in the rate-distortion sense: The selection minimizes 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 domains M_a and their values $\underline{u}(M_a)$ must be updated. We call these vertices *ancestors* and denote this set by A_{M_v}. These vertices are easily found by backtracking the DAG of vertices shown in Fig. 4b for each vertex in M_v. Two types of ancestors a exist: the vertices such that $M_v \subset M_a$ (i.e., toward 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 the following lemma:

LEMMA 1 (Size of the ancestor set). *Let A_{M_v} denote the set of ancestors of the domain M_v; then*

$$\left|A_{M_v}\right| \in O(\log n) \tag{4}$$

on average. Moreover, there exists a $O(\log^2 n)$ algorithm to update all the ancestor values after modifying $\underline{u}(M_v)$.

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 (Figs. 6c–6d).

We give the optimization algorithm below. In our application, we use semi-regular 4-8 meshes and the configuration with minimal rate is given by the initial coarse mesh (e.g., see Fig. 2a). We denote by $|\mathcal{B}|$ the rate of this mesh. As pointed out in Section 1.4, M_{v_0} contains all the vertices in the mesh. Therefore, since we use global error, the global rate and the global distortion of the mesh 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 update of the global error (line 10) is explained in Section 2.2. The total complexity of the algorithm is computed in Section 2.3.

OPTIMIZATION ALGORITHM (FULL DECOMPOSITION).

1 **initialization:**
  2 **for all** v
    3 COMPUTE $\Delta D(M_v)$, $\Delta R(M_v)$
    4 $\lambda(v) \leftarrow \frac{-\Delta D(M_v)}{\Delta R(M_v)}$

iteration:
  $i = 1$ (*counter for the approximations.*)
  while $R(M_{v_0}) > |\mathcal{B}|$
    $v^\star = \arg\min_{v \in M} \lambda(v)$
    $M_i \longleftarrow M_{i-1} \setminus M_{v^\star}$
    UPDATE $\Delta D(M_a)$ AND $\Delta R(M_a), \forall a \in A_{M_v}$
  end
end

Suppose that two slopes $\lambda(v_1)$ and $\lambda(v_2)$ have equal value and that v_1 and v_2 are connected at the same step l (Figs. 3a–3e). Then according to Lemma 3, the distortion of the optimal approximations decreases monotonically as the rate of the model increases. Therefore if $\lambda(v_1) = \lambda(v_2)$ is the minimal value at some iteration, decimating M_{v_1} increases all updated slopes $\lambda(a)$, where $a \in A_{M_{v_1}}$. Since v_2 cannot be part of $A_{M_{v_1}}$ (v_1 and v_2 are connected at the same step l), then $\lambda(v_2)$ is the next minimal value and M_{v_2} is decimated right after M_{v_1}. Suppose now that v_1 and v_2 are connected at different steps; then the domain attached to the vertex with the coarsest connection step l is preferred, since the decrease in rate of the model is larger. This condition is easily implemented since the minimal value is searched in a hierarchical way in the forest of quadtrees used to store the model.

Since the algorithm performs generalized decimation, a solution might not exist for every rate. Not having a solution for all rates is actually not a problem, since the algorithm returns an approximation satisfying the constraint as stated in (1). Consider the following example: Assume that the input model is a triangulated flat plane (Fig. 7a). Therefore, the error at all vertices is zero. The optimal choice in the operational RD sense is to decimate M_{v_0} (the merging domain at the root) since this choice maximizes the decrease in rate of the model while minimizing the increase in distortion. In other words, all the vertices in the plane are decimated in a single optimization step (Fig. 7b). In this extreme case, only two rates are available: $R(M_{v_0}) = n$ and $R(\breve{M}_{v_0}) = 4$ (Figs. 7a and 7b). Any approximation having between n and 4 vertices would not optimally satisfy the constraint in (1). Also, such an approximation would be rather useless in our application.

2.2. Update of Global Error

In this section, we present the algorithm used to update the functional variations of the vertex characteristics, i.e., recompute the global error. The algorithm has cost $O(\log^2 n)$ and

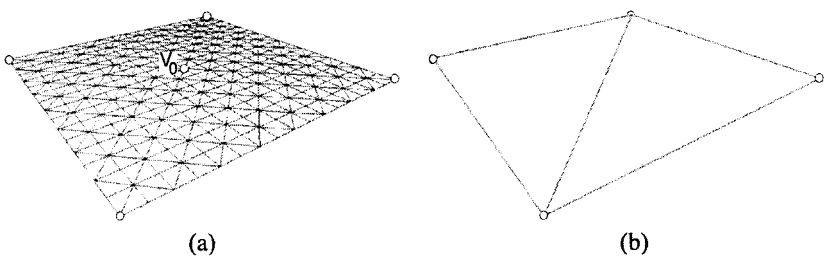


FIG. 7. Optimal choice in the operational rate–distortion sense. (a) For a flat model, all vertex errors are zero. (b) The algorithm chooses to decimate M_{v_0}, i.e., the merging domain attached to the root vertex, because the decrease in rate is maximized for a minimal increase in distortion a zero in this case.

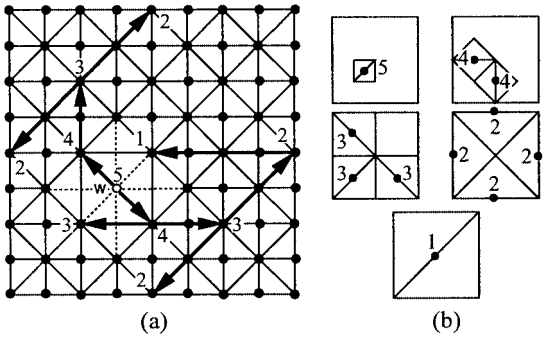


FIG. 8. Parents of vertex w (A_w). (a) To find the parents of w (connected at $l = 5$), the DAG of vertices is traversed fine to coarse. The path (arrows) backtracks recursively the DAG of vertices (Fig. 4b) toward the root. Each traversed vertex splits a pair of triangles depicted in (b). In both (a) and (b), the index next to each vertex indicates the connection step (Figs. 3a–3e).

is derived from an algorithm based on an inclusion–exclusion principle used to compute merging domain intersections (MDIs) presented by Balmelli *et al.* in [2]. 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

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

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 optimization algorithm in Section 2.1) during the mesh optimization. In the algorithm, the updated ancestor functionals are denoted by $\Delta \underline{u}'(M_a)$. 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 DAG of vertices. An example of a traversal is shown in Fig. 8. The index next to each vertex is the connection step l (Figs. 3a–3e). The larger the index, the finer the connection step. Finally, note that an important property related to the parents of a vertex is

$$\forall w \in M_v, \quad v \in A_w; \tag{6}$$

i.e., all the vertices $w \in M_v$ have v as a parent. This can easily be seen in the DAG in Fig. 4b.

UPDATE ALGORITHM.

1 **for** ALL AVAILABLE VERTICES $w \in M_v$ CONNECTED AT STEP $l + m \ldots l$
  2 **for** ALL $a \in A_w$
    3 $\Delta \underline{u}'(M_a) = \Delta \underline{u}(M_a) - \Delta \underline{u}(M_w)$
    4 $\lambda(a) \leftarrow -\Delta D(M_a)/\Delta R(M_a)$
  5 **end**
  6 **decimate** w
7 **end**

The update (5) is performed for each parent. In our geometry simplification algorithm, once the set of parents for w has been visited, w is decimated. Hence in this case the update

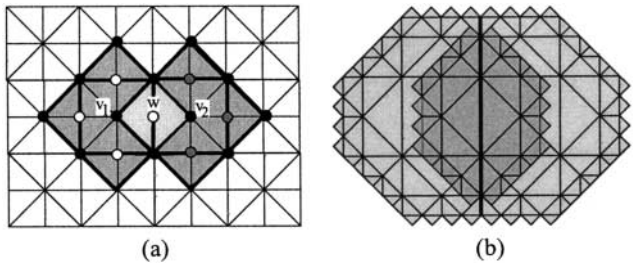


FIG. 9. Update of the global error. (a) When M_{v_1} and M_{v_2} are decimated, the global characteristics (rate and distortion) of the mesh are $\underline{u}(M_{v_0}) - \Delta\underline{u}(M_{v_1} \cup M_{v_2})$. (b) Intersection between two large merging domains: For clarity, the merging domains are represented using their support. The intersection is depicted by the darkly shaded area and the thick line represents the boundary between the domains.

algorithm is used to recompute the ancestor variations and to jointly decimate the domain M_v. The update algorithm replaces lines 9 and 10 of the optimization algorithm given in Section 2.1. Suppose that M_v is decimated and that A_{M_v} must be updated. Furthermore, assume that v is connected at step l. Then the set of vertices $w \in M_v$ has to be iteratively decimated starting at the vertices in the domain having the largest connection step (vertices at the finest level). This is equivalent to iteratively pruning the DAG of vertices spanning M_v starting at the terminal vertices. 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.

We explain now how global error is recomputed using the above algorithm. We start with a simple example and then we address the general case. To do so, we summarize the problem of finding MDIs. A complete analysis is found in [2].

Consider the following example: After the initialization phase (lines 1–4 of the optimization algorithm in Section 2.1), the functional values $\Delta\underline{u}(M_v)$ are global since no domain has yet been decimated. Consider M_{v_1} and M_{v_2} as depicted in Fig. 9a. Clearly, after decimating both domains, the global characteristics of the mesh (rate and distortion) are

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

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,

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

However, $M_w = M_{v_1} \cap M_{v_2}$, as shown in Fig. 9a. 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 the support of either M_{v_1} or M_{v_2}. Therefore,

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

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 Fig. 9a). Hence following

(6), the updated functional characteristics at v_1, v_2, and v_0 (root vertex) are, respectively,

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

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

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

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 Fig. 9a. The updated values at v_2 and v_0 are, respectively,

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

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

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

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 domains M_{v_1} and 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 Fig. 9b. In the figure, the domains are depicted using their support for clarity. According to Lemma 1, we have $O(\log n)$ possible arrangements for intersections. Hence the examples in Figs. 9a and 9b are just particular cases. Figure 11 depicts another example of arrangement.

The intersection in Fig. 9b can be decomposed in terms of smaller merging domains, as shown in Fig. 10. The number of domains is proportional to the size of the intersection. In the example of Fig. 9a, a single domain M_w is sufficient to express the intersection, whereas in Fig. 10, 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 (Fig. 10); 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 [3], we call this problem *merging domain intersections* (MDIs) and we show that (14) is computed in $O(\log n)$ time. Also, we show that

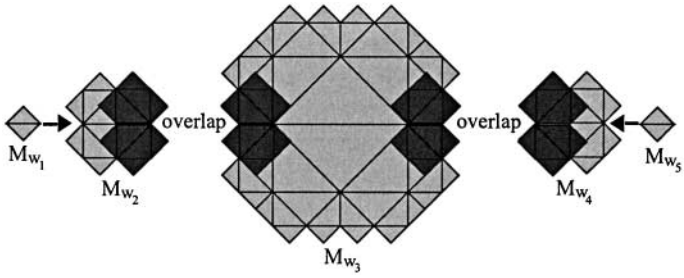


FIG. 10. Decomposition of the intersection in Fig. 9b. The intersection is expressed as the union of a set of smaller domains M_{w_i}, $i = 1:5$. Note that the pairs of domains M_{w_2}, M_{w_3} and M_{w_3}, M_{w_4} overlap (represented by darkly shaded area).

the intersections between M_v and the domains of all its ancestors A_{M_v} are computed in $O(\log^2 n)$ 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 of the RD characteristics when M_v is decimated. Hence, the characteristics $\Delta\underline{u}(M_{v_0})$, i.e., at the root vertex, are the global RD characteristics of the mesh.

We now conclude 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,

$$\begin{aligned} &\Delta\underline{u}(M_v) - \sum\nolimits_{w\in M_v, w\neq v} \Delta\underline{u}(M_w), \\ &\Delta\underline{u}\big(M_{v_0}\big) - \sum\nolimits_{w\in M_v, w\neq v} \Delta\underline{u}(M_w). \end{aligned} \tag{15}$$

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

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

which corresponds to the global rate and error 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 full decomposition of the mesh, is found as follows: In [3], we show that merging domains have size $O(\log n)$ on average. Thus, assuming a mesh of n vertices, the initialization has cost $O(n \log n)$. At each iteration, the optimal vertex $v^\star$ (having minimal slope $\lambda(v)$) is found in $O(\log n)$ operations using the values $\lambda_{\min}$. The cost to decimate M_v and update the variations for the vertices in A_{M_v} using MDIs is $O(\log^2 n)$. Also, $O(\log n)$ values $\lambda(v)$ and $\lambda_{\min}$ are recomputed and the algorithm is iterated. On average,

$$n/O(\log n) \tag{17}$$

steps are necessary to decompose the mesh, since at each step $O(\log n)$ vertices on average are decimated (average size of merging domains). Hence, the cost to compute the full

decomposition is

$$O(n \log n). \tag{18}$$

Without using MDIs, a direct algorithm needs to recompute the global error over each ancestor domain. A lowerbound for this update is obtained as follows: We have roughly $O(4^{l+1})$ vertices at step l and $O(\log n)$ ancestors exist. Call a any such ancestor; then

$$|M_a|(i, n) \approx n/4^{i-1}, \quad 1 \le i \le l. \tag{19}$$

Therefore, the complexity is at least

$$\sum_{i=0}^{\log_4 n} 4^i \sum_{j=0}^{i} \frac{n}{4^j} = \frac{16}{9}n^2 - \frac{1}{3}n \log_4 n - \frac{7}{9}n \in O(n^2). \tag{20}$$

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 $O(\log n)$ cases of overlap, i.e., arrangements for intersections, one has to deal with (Section 2.2). We summarize our result with the following lemma.

LEMMA 2 (Algorithm complexity). *Given an input mesh M containing n vertices, the algorithm using generalized decimation and global error computes a single approximation of M in $O(n)$ time and full decomposition of M in*

$$O(n \log n) \tag{21}$$

time when merging domain intersections are used to recompute the global error.

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 optimization step (Section 3.1). Second, we discuss issues related to intersections between domains and how optimal choices are affected (Section 3.2).

3.1. Optimal Choice

Recall that our rate functional measures the number of vertices; 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

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

Such a case is possible with the l_2 or the l_∞ norm since these norms are nonmonotonic with the mesh size [2, 12]. Recall that $\Delta D(M_{v_1}) < 0$ and $\Delta D(M_{v_2}) < 0$ (Section 2.1). Then if v_2 is decimated, following (5) and (6) we have that

$$\Delta D\left(M_{v_1}\right) - \Delta D\left(M_{v_2}\right) > 0, \tag{23}$$

and the new slope is

$$\lambda(v_1) = \frac{-\left(D\left(M_{v_1}\right) - D\left(M_{v_2}\right)\right)}{\left(\Delta R\left(M_{v_1}\right) - \Delta R\left(M_{v_2}\right)\right)} < 0; \tag{24}$$

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 generalized decimation as follows: If the decimation of a domain M_v provokes a nonmonotonicity at a parent of v, then the algorithm decimates the domain of the parent instead. This is implicitly done when choosing the minimal value $\lambda(v)$; hence no implementation is necessary to enforce this condition. In Lemma 3 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 in the operational RD sense.

LEMMA 3 (Optimal candidate in the operational RD sense). *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 optimized before M_{v_1} if and only if*

$$\frac{\Delta D\left(M_{v_1}\right)}{\Delta D\left(M_{v_2}\right)} > \delta > 1, \tag{25}$$

where $\delta = \Delta R(M_{v_1})/\Delta R(M_{v_2})$. When v_1 does not meet condition (25), *the domain M_{v_1} is said to be nonmonotonic with respect to M_{v_2}.*

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

$$\Delta D\left(M_{v_1}\right)\Delta R\left(M_{v_2}\right) > \Delta R\left(M_{v_1}\right)\Delta D\left(M_{v_2}\right). \tag{26}$$

Since the functional R is monotonically increasing, we can write

$$\Delta R\left(M_{v_1}\right) = \delta \Delta R\left(M_{v_2}\right), \qquad \delta > 1. \tag{27}$$

Then, replacing (27) in (26) yields

$$\Delta D\left(M_{v_0}\right) > \delta \Delta D\left(M_{v_1}\right). \tag{28}$$

■

3.2. Intersection between Domains and Optimal Choice

In Section 2.1, we explained that a type of ancestors comprises the vertices a such that M_v and M_a partially overlap. Figure 11 illustrates such a case. Assume now that M_v is the optimal domain to optimize 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 generalized decimation cannot avoid nonmonotonicities across rate.

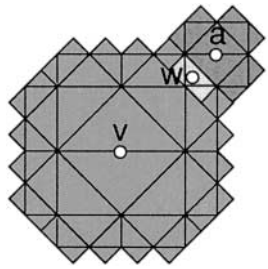


FIG. 11. Suboptimal choice of the algorithm. $w \in M_v \cap M_a$ and M_a is a nonmonotonic merging domain with respect to M_w (see Lemma 3). Decimating M_w provokes a nonmonotonicity at M_a.

We perform experiments using matrices of amplitudes z (terrain data [9]) and compare our algorithm to a limited version using global error but with the restriction that only terminal vertices in the DAG of vertices can be candidates for decimation; i.e., vertices are decimated one at a time. Hence, all decimated merging domains have size one (i.e., $|M_v| = 1$). This prevents the algorithm from making optimal choices in the sense of Lemma 3 (Section 3.1). We find that although the algorithm using generalized decimation cannot avoid monotonicity, as explained in the above paragraph, the RD curve (top curve in Fig. 12) is very stable compared to the one obtained with the limited version (bottom curve in Fig. 12).

With the assumption that no suboptimal choice is made (which basically assumes the independence of vertex choices) and using Lemma 3, our algorithm reduces to the algorithm given by Chou *et al.* in [7]. In [7], the authors prove the optimality of their algorithm, which allows us to suggest that our method be optimal as well. More precisely, they point out that a remarkable property of their algorithm is that the optimal approximations are found using iterated decimation steps. We do not give a formal proof for optimality in this paper and leave the question open. It is clear that even if our algorithm was proven optimal,

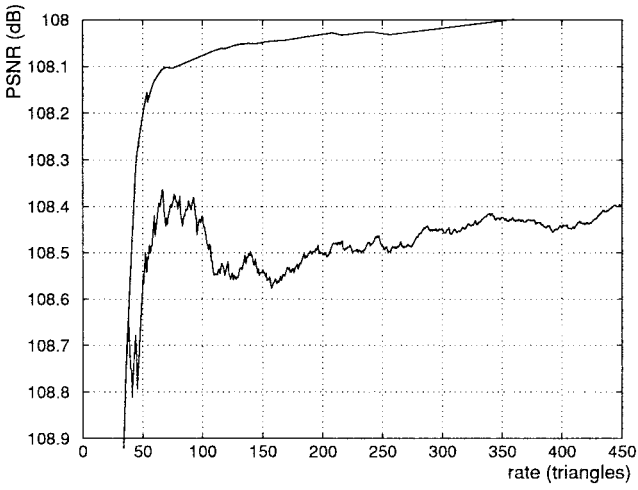


FIG. 12. Comparison of nonmonotonicities of the RD curve between the algorithm using generalized decimation (top curve) and its limited version using standard decimation (bottom curve). In the latter case, the algorithm cannot conserve monotonicity.

the optimality would be restricted to the case of (quad) tree-constrained approximations. The more general problem of optimal vertex selection without constraints is known to be NP-Hard [1].

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., a piecewise-linear polynomial. This allows us to run a large number of experiments and to compare our approach using generalized decimation and global error with several standard optimization strategies. In Section 4.2, we apply the algorithm to a standard set of semi-regular meshes as well as terrain data. Finally in Section 4.3 we discuss the increase in cost due to the use of generalized decimation and global error.

4.1. Experiments with Polylines

Recall the hierarchy imposed over the vertices by the 4-8 mesh construction (Section 1.4). A similar hierarchy can be applied to the polyline model using a binary tree. Hence, any approximation generated by a decimation or insertion algorithm can be represented with a partial tree. Figure 13 shows an example of hierarchical approximation: The top curve is the original one and has seven 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.

We compare a 1D counterpart of our algorithm to two standard strategies using local error: vertex insertion and vertex decimation. 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 also compare our results to the optimal approximations obtained using dynamic programming. In the context of mesh simplification, these approximations can be seen as the optimal solutions using irregular triangulations. Agarwal *et al.* [1] have demonstrated that, for meshes, finding these solutions is NP-Hard. 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.

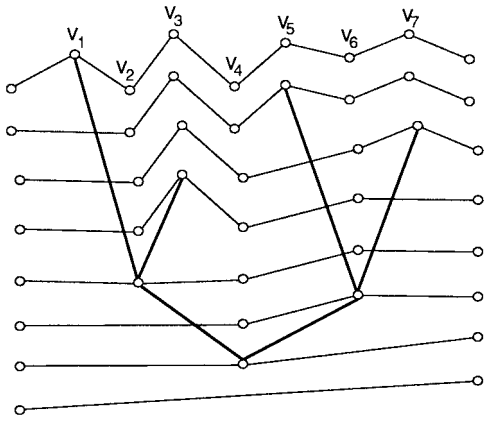


FIG. 13. Successive approximations of a polyline using decimation. Knots are iteratively decimated from top to bottom. The binary tree constraining the decimation is depicted using bold lines.

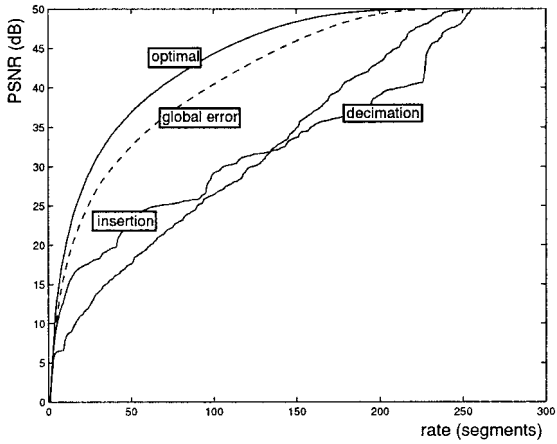


FIG. 14. Comparison of RD curves. The rate is computed as the number of segments forming an approximation and the distortion is computed in l_2 norm with respect to the original curve. The top curve is obtained using dynamic programming. The dashed curve is obtained with our algorithm. Finally, the two bottom curves compare the decimation and insertion approaches using local error.

The results are shown in Fig. 14: The top curve shows the errors of the optimal approximations found using dynamic programming. The dashed curve is obtained with our algorithm. 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. The figure shows that the approximations preserving the hierarchy obtained with our algorithm perform very well compared to the optimal solutions obtained using dynamic programming: The curves show less than a 5 dB difference. Recall that the approximations preserving the hierarchy have many additional properties compared to irregular ones such as compact storage and efficient processing due to the natural parameterization.

4.2. Experiments Using Semi-regular Meshes

We apply our geometry simplification algorithm to a standard set of semi-regular meshes with 4-8 connectivity. These meshes are generated using 4-8 subdivision during a remeshing process. We compare our algorithm to a standard greedy insertion algorithm such as used in [20, 23]. Note that our version operates on semi-regular meshes, whereas in [20, 23] the algorithms work on terrain data, i.e., regular setting. Our meshes are stored using the pointerless forest of quadtrees described in [2]. This data structure has efficient navigation properties and allows our implementation to be computationally efficient. We summarize the characteristics of our models in Table 1. The semi-regular models are formed by sets of subdivided quads. The terrain model [9] is composed of 15 patches of size 256×256. In all experiments, we normalize the errors and fix the gain to 50 dB.

Results are presented in Figs. 15–19. Note that since our approach uses decimation, our RD curves are computed from right to left, whereas the insertion strategy computes the approximations from left to right. Our algorithm shows up to a 15 dB improvement compared to the greedy strategy (e.g., in Fig. 18b), whereas we measure an average gain of 8 dB with our set of models. We comment on our results in more detail below.

TABLE 1
Models' Characteristics

Model	Number of faces	Total triangles	Figure
Cow	50	102400	15
Wallace	87	178176	16
Venus	24	49152	17
Armadillo	102	208896	18
Terrain	15	1966080	19

We can see that the greedy insertion strategy performs closely to our algorithm at low rates (about 10% of the total number of vertices). Then, it quickly accumulates errors through iterations. For example in Fig. 15b, the greedy curve varies very slowly across rates, except for some peaks. This behavior is characteristic of the algorithm being blocked in local minima of the error function. This can also be observed in the other figures.

The greedy algorithm performances also depend on the smoothness of the model. Our algorithm shows great improvements on models with frequent changes in curvature such as the Armadillo model (Fig. 18). With this model, improvements up to 18 dB are observed. On the other hand, smoother models are better suited to greedy approaches. For example, on the Venus model (Fig. 17) the algorithm shows fewer improvements. However, we still have an improvement of almost 10 dB, around 60% of the total number of vertices for this model. An example of terrain approximation is shown in Fig. 20. Figure 21a shows the Armadillo model at full resolution, whereas Fig. 21b shows a simplified version. The face is still finely triangulated in high curvature regions, whereas the rate of the model is reduced in flat regions such as internal ears and shoulders.

The most noticeable result is the stability of the RD curve returned by our algorithm. The use of generalized decimation and global error provides a steady increase in quality as the rate of the model increases. As explained above, because of local minima, the greedy optimization technique cannot guarantee a constant increase in quality across rates.

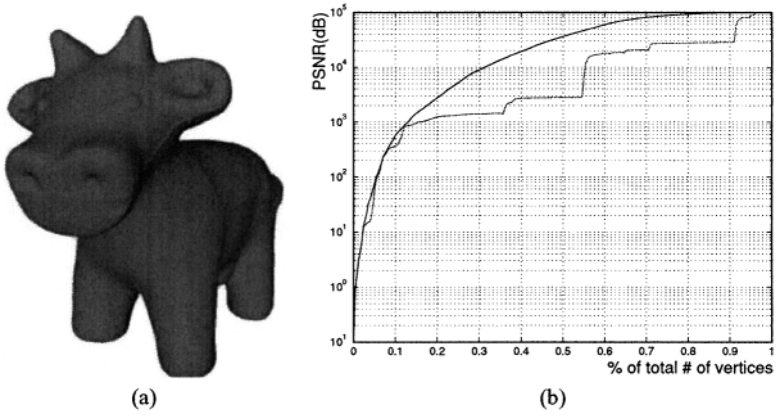
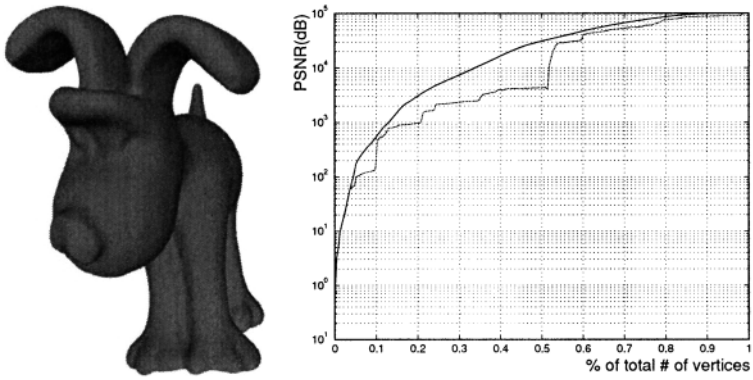
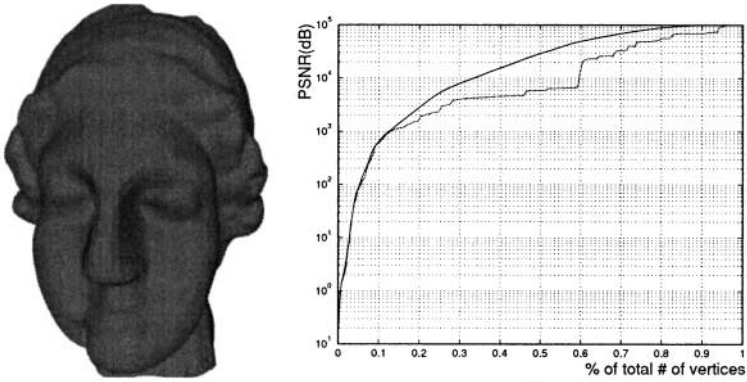


FIG. 15. Cow model. (a) Rendered model. (b) Rate–distortion curves: The top curve is obtained with our algorithm, whereas the bottom one is obtained with a greedy insertion approach.



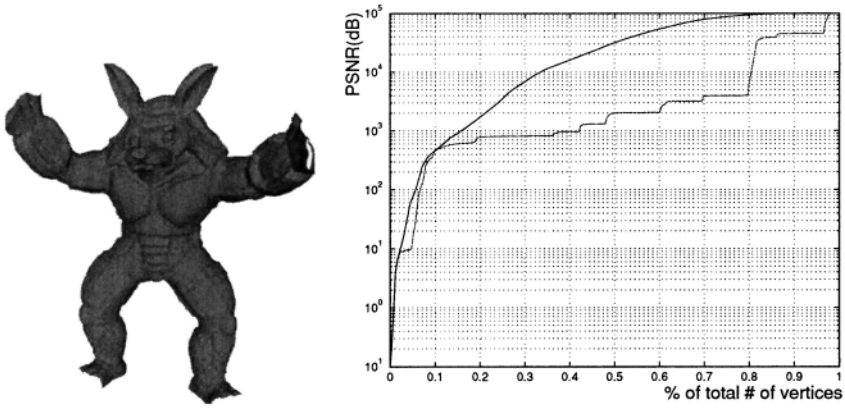
(a) (b)

FIG. 16. Wallace model. (a) Rendered model. (b) Rate–distortion curves: The top curve is obtained with our algorithm, whereas the bottom one is obtained with a greedy insertion approach



(a) (b)

FIG. 17. Venus model. (a) Rendered model. (b) Rate–distortion curves: The top curve is obtained with our algorithm, whereas the bottom one is obtained with a greedy insertion approach.



(a) (b)

FIG. 18. Armadillo model. (a) Rendered model. (b) Rate–distortion curves: The top curve is obtained with our algorithm, whereas the bottom one is obtained with a greedy insertion approach.

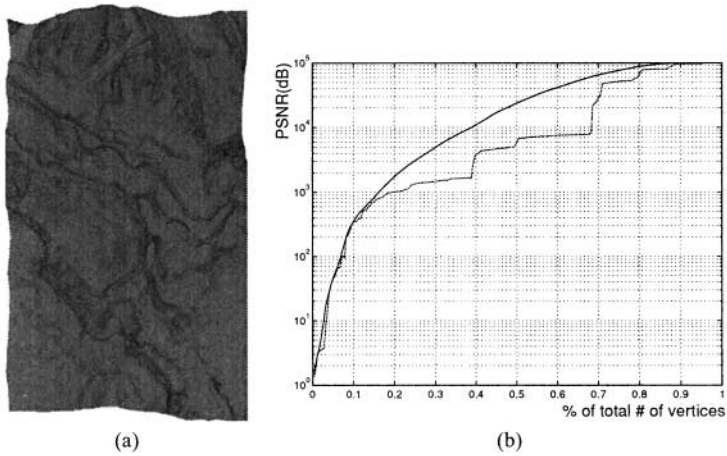


FIG. 19. Terrain model. (a) Rendered model (top view). (b) Rate–distortion curves: The top curve is obtained with our algorithm, whereas the bottom one is obtained with a greedy insertion approach.

4.3. Computational Cost

Since our method uses generalized decimation, a larger space of possible approximations is searched compared to greedy approaches. This inherently involves more comparisons and slightly increases the total cost of the algorithm. However, since our mesh hierarchy is based on a tree structure (the models are stored in a forest of quadtrees), the cost of finding a minima in the tree does not change much whether the search is limited to the leaves (for example in a greedy decimation approach) or expanded to the whole tree. Greedy insertion strategies are faster to generate approximations containing few vertices faster since these algorithms start with a coarse representation. However, their cost rapidly increases when an approximation containing a large number of vertices is computed.

An advantage of our method is that the functionals R and D are only evaluated during the initialization phase of the optimization algorithm described in Section 2.1. During the optimization phase, the algorithm does not need to evaluate the functionals. Therefore few operations are necessary to update the global error estimates duing the optimization process.

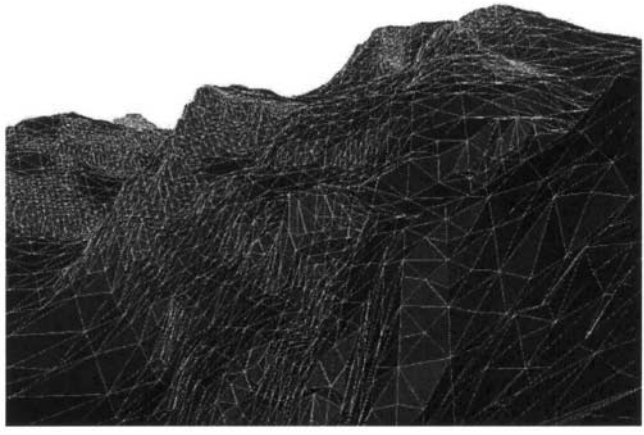


FIG. 20. Approximation of the terrain model.

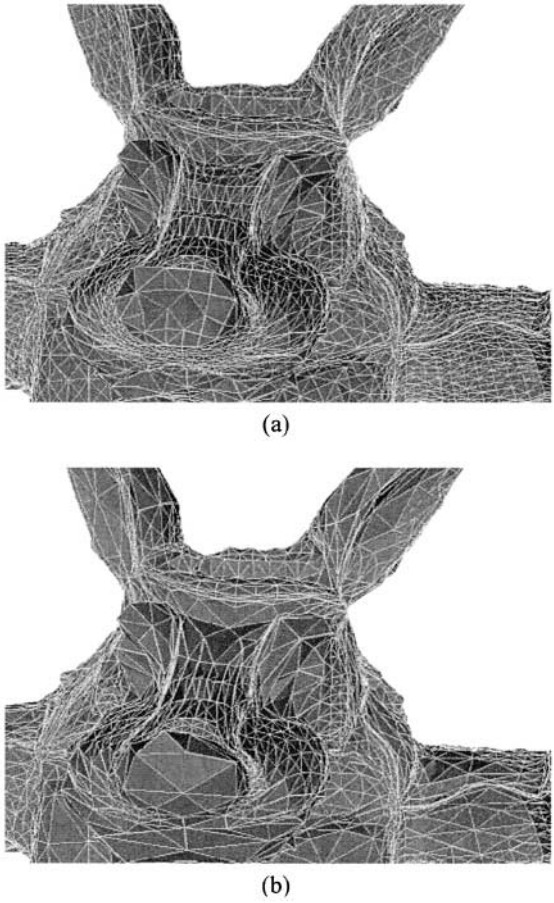


FIG. 21. Approximation of the Armadillo model. (a) Original mesh. (b) Approximation: The ears are simplified, whereas the face is tessellated finely in high curvature regions.

If needed, complex functional values can be computed during a preprocessing phase, e.g., using the result of a simulation, and stored jointly with the mesh properties.

Finally, our current implementation uses a pointerless forest of quadtrees having efficient navigation properties [2]. More precisely, the merging domain of a vertex is efficiently computed using constant-time neighbor search operations in the forest of quadtrees. The largest model (the terrain model in Fig. 19a) is fully decomposed—all successive optimal approximations in the operational RD sense are computed—in less than 10 s on a Pentitum III 800 MHz laptop, whereas for smaller models the processing lasts only a few seconds. The greedy algorithm performs in about half the time of our algorithm. However, note that our greedy implementation is very simple. If elaborated mechanisms are used in order to avoid local minima, then performances are likely to degrade rapidly.

5. CONCLUSION

This paper introduces an optimization technique for meshes with subdivision connectivity, i.e., hierarchical datasets. Our method allows regular as well as semi-regular models

(e.g., subdivision surfaces) to be optimized using global error in $O(n \log n)$, improving on direct $O(n^2)$ techniques. We apply our algorithm to 4-8 meshes to perform geometry simplification. We introduce a selection technique allowing sets of vertices to be optimized jointly. In our application, this results in performing generalized decimation of the model.

Our application comprises a metric to compute the rate of an approximation in vertices and its distortion in l_2 norm with respect to the original model. Different metrics can be used to solve other problems: For example, view-dependant geometry simplification can be achieved by simply changing the definition of the distortion functional. More precisely, the approximation error for each domain M_v will be computed given a viewpoint. Compression algorithms operating in the RD plane can also be implemented. In this case, the rate functional returns the cost in bits for encoding a merging domain. For example, the vertices in the domain can be processed using wavelet filters and the number of bits is given by the encoding of the resulting coefficients. The distortion functional computes the reconstruction error. When a function is defined on the mesh (e.g., by using a texture map), the functionals can be used to perform mesh optimization using attributes. Hence, several problems such as fast evaluation of bidirectional reflectance distribution functions and radiosity can be efficiently addressed. Finally, our algorithm can be used in finite element analysis to obtain high-quality triangulations.

The decimation approach used by the algorithm returns better approximation quality than standard refinement techniques. Our method gives excellent results using semi-regular meshes at low computational cost. However, decimation approaches require more storage than refinement techniques in general. This problem can be addressed in several ways: Our algorithm can be used patch-wise in order to use fixed-size storage. In this case, vertices on patches' boundaries must be aligned in order to obtain a conforming model. Out-of-core methods [19] are an active area of research and our algorithm can benefit from these techniques. In conclusion, our algorithm is well suited to solve at low cost complex mesh optimization problems using global error.

APPENDIX A: EVALUATION OF MESH FUNCTIONALS

We compute the costs in rate for each domain, measured as the number of vertices, in closed form using the results in [3]. More precisely, in [3] 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 22 shows

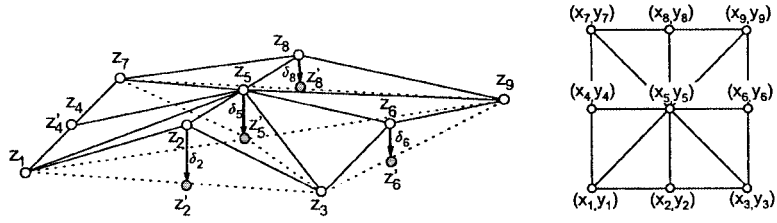


FIG. 22. Computation of the error in squared l_2 norm for a triangulated quad. The left part shows a top view of the triangulated quad.

how errors are measured on a triangulated quadrilateral. The total distortion is evaluated similarily on the domain M_v.

Consider the triangulated quadrilateral in the left-hand side of Fig. 22. 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 Fig. 22). 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., initially $\Delta D(M_v) < 0$ for all the vertices.

ACKNOWLEDGMENTS

We thank the reviewers for their comments and their help for improving the presentation of this paper. We also thank Anthony Edward for his help and support in proofreading this paper. Laurent Balmelli also personally thanks his PhD advisor, Professor Martin Vetterli, and his co-author, Professor Thomas Liebling, for their support and encouragement during his doctoral studies at the Ecole Polytechnique Fédérale de Lausanne in Switzerland, the period during which most of this work has been completed. The authors thank the following people for providing the models used in their experiments: Professor Denis Zorin from New York University and Dr. Ioana Martin at IBM Research for providing the models in Figures 15 and 16, Cyberware Inc. for the Venus model shown in Figure 17, and Prof. Mark Levoy from Stanford University for the Armadillo model in Figure 18.

REFERENCES

1. P. K. Agarwal and P. K. Desikan, An efficient algorithm for terrain simplification, in *Proceedings ACM-SIAM Symposium Discrete Algorithms, 1997*, pp. 139–147.
2. L. Balmelli, *Rate-Distortion Optimal Mesh Simplification for Communications*, Ph.D. dissertation 2266, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland, 2000.
3. L. Balmelli, T. M. Liebling, and M. Vetterli, Computational analysis of meshes simplification using error, to appear in *Computational Geometry: Theory and Application.* Preprint available at http://www.balmelli.net/download/cgta2002.pdf.
4. H. Biermann, I. Martin, F. Bernadini, and D. Zorin. Cut-and-Paste editing of multiresolution surfaces, in *Proceedings of SIGGRAPH* (to appear), July 2002.
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 Trans. Inform. Theory* **35**, 1989, 299–315.
8. J. H. Clark, A fast algorithm for rendering parametric surfaces, in *Proceedings of SIGGRAPH, 1979*, pp. 289–299.
9. Office Federal de Topographie. Pixelkarte 1:25000 cd rom 1, 2, 3. CH-2084 Wabern, May 1997.
10. M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, Roaming terrain: Real-time optimally adapting meshes, in *Proceedings of IEEE Visualization, 1997*.
11. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, Dordrecht/Norwell, MA, 1992.
12. P. S. Heckbert and M. Garland, *Survey of Polygonal Surface Simplification Algorithms*, Technical Report, Carnegie Mellon University, May 1997.
13. A. Khodakovsky, P. Schröder, and W. Sweldens, Progressive geometry compression, in *Proceedings of SIGGRAPH, 2000*, pp. 271–278.
14. L. Kobbelt, $\sqrt{3}$ subdivision, in *Proceedings of SIGGRAPH, 2000*, pp. 103–112.

15. K. Ramchandran and M. Vetterli, Best wavelet bases in a rate–distortion sense, IEEE Trans. On Image Processing **2**, 1993, 160–175.

16. U. Labsik, L. Kobbelt, R. Schneider, and H.-P. Seidel, Progressive transmission of subdivision surfaces, *Comput. Geom.* **15**, 2000, 25–39.

17. L. Balmelli, J. Kovačević, and M. Vetterli, Quadtree for embedded surface visualization: Constraints and efficient data structures, *Proc. IEEE Int. Conf. Image Process.* (ICIP) **2**, 1999, 487–491.

18. A. W. F. Lee, W. Swelden, P. Schröder, L. Cowsar, and D. Dobkin, Maps: Multiresolution adaptive parameterization of surfaces, *Proceedings of SIGGRAPH, 1998*, pp. 95–104.

19. P. Lindstrom, Out-of-core simplification of large polygonal models, in *Proceedings of SIGGRAPH, July 2000*.

20. 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, in *Proceedings of SIGGRAPH, 1996*, pp. 109–118.

21. P. Lindstrom and V. Pascucci, Visualization of large terrains made easy, in *Proceedings of IEEE Visualization, October 2001*.

22. C. Loop, *Smooth Subdivision Surfaces Based on Triangles*, Master's thesis, Department of Mathematics, University of Utah, 1987.

23. R. Pajarola, Large scale terrain visualization using the restricted quadtree triangulation, in *Proceedings of IEEE Visualization, 1998*, pp. 299–305.

24. H. Rushmeier, Realistic image synthesis for scenes with radiatively participating media, Ph.D. thesis, Cornell University, 1988.

25. H. Samet, *Application of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA, 1990.

26. P. Schröder and W. Sweldens, Spherical wavelets: Texture processing, in *Rendering Techniques '95* (P. Hanrahan and W. Purgathofer, Eds.), Springer-Verlag, Vienna, New York, August 1995.

27. L. Velho, Four-face cluster simplification, in *Proceedings of Shape Modeling International, 2001*.

28. L. Velho and D. Zorin, 4-8 subdivision, *Comput. Aided Geom. Design* **18**, 2001, 397–427, Special Issue on Subdivision Techniques.

29. D. Zorin, A method for analysis of C^1-continuity of subdivision surfaces, *SIAM J. Numer. Anal.* **37**, 2000, 1677–1708.