EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Communication Systems Department
Laboratory for Audiovisual Communications

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Progressive Meshes in an Operational Rate-Distortion Sense with Application to Terrain Data

submitted to *IEEE Transactions on Visualization and Computer Graphics*

L. Balmelli and J. Kovačević and M. Vetterli
April 19th 2000

DSC Technical Report - DSC/2000/019

Abstract

This paper presents an efficient simplification method for regular meshes obtained with a binary subdivision scheme. Our mesh connectivity is constrained with a quadtree data structure. We propose a quadtree built especially for this class of meshes having a *constant-time traversal property*. We introduce a rate-distortion (RD) framework to decimate the mesh and build a progressive representation for the model. We propose to achieve the RD-optimal solutions for our quadtree-restricted setting: to obtain the optimal solutions, we show how to find the vertex in the quadtree yielding the best RD trade-off and then perform *optimizations at variable rate*, where the rate is given by a cost function (for example the number of triangles). All previous methods are restricted to constant rate optimization only. We compare the optimal approach to its greedy counterpart. We give computationally optimal formulations for all our algorithms on the quadtree. We apply our technique to a large dataset of terrains and give extensive experimental results.

Index terms: meshing, decimation, optimality, rate-distortion, terrain data, quadtree, regular subdivision scheme, progressive meshes.

Contents

1 Introduction

1.1 Simplification of meshes with application to terrain data

Highly detailed models in computer graphics require extensive resources to be rendered. Researchers have been looking for simplification algorithms to reduce the quantity of model information while attempting to preserve its geometric features as well as possible. We present a framework to decompose a model into a progressive mesh using a quadtree data structure. Although this approach has already been taken in [10], the novelty of our solution is that the optimization is done at variable rate in an operational rate-distortion (RD) framework (see below), where the rate is given by a cost function (for example the number of triangles). We apply our method to *Digital Elevation Terrain Data* (DETD) which are usually large datasets requiring extensive computational resources to be processed. This type of dataset is central to Geographical Information System (Figure 1). Let us

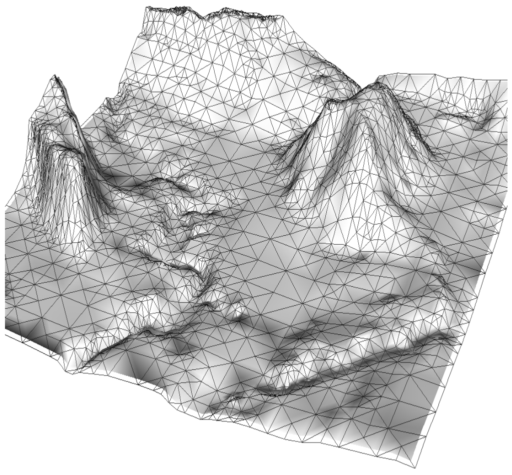

Figure 1: Rendered DETD using a quadtree triangulation counting 6000 triangles.

define our approach more formally: we consider a set of N vertices obtained from a parametric representation $f[x, y]$ (as in DTEDs) stored in a quadtree data structure. We use a so-called *quadtree triangulation* as in [16] for the connectivity of the mesh[1]. Our method also applies to sets of vertices in $\mathbb{R}^3$ if the connectivity is conserved. In this case, a remeshing technique (see for example [15]) is applied if no parameterization is given for the vertices. We propose an RD-optimal $\Theta(N \log^2 N)$ algorithm to decimate the vertices in the mesh, or equivalently to prune the quadtree.

The fundamental difference between our operational RD framework and previous solutions is that at *each optimization step, the algorithm chooses to decimate the vertex in the tree hierarchy giving the best trade-off between rate and distortion*. The rate is defined as a *cost for the approximation* and the distortion as a *distance with the original model (i.e. full resolution)*. Namely, each simplification *maximizes the decrease in rate while minimizing the increase in distortion*. In previous works [5, 19], candidate vertices were restricted to lie in leaf nodes. In [17], the author give the same refinement algorithm as in [16], but claim a better selection threshold for the vertex. In our case, each optimization step ends up pruning a *set of nodes* in the quadtree [2], therefore each simplification is represented by a set of subtrees. Decimating any vertex in the quadtree raises two questions: *how to insure the spatial continuity of the mesh* and *how to efficiently recompute the error of the vertices after the simplification?* A subtree representing an approximation is known as a *restricted quadtree* [18]. Such a tree represents a spatially continuous mesh, in the sense that all triangle edges are coplanar. This problem has also been mentioned as the *coplanarity problem* by Samet in [18]. While several researchers have proposed to solve the problem, the solutions were either for particular cases [5, 19] or computationally suboptimal [17]. We propose a computationally optimal algorithm to solve the problem of computing a restricted quadtree. As for updating the

[1]More formally, our mesh connectivity is obtained with a binary subdivision scheme.

[2]Consider the case where the vertex giving the best trade-off is not in a leaf node, then additional vertices have to be decimated jointly in order to conserve the tree hierarchy.

errors of the mesh in minimal computational time, to the authors knowledge no solution to the general case has been given. We explain then *how to find the minimal set of vertices for which the simplification changes the error* and *give a computationally optimal algorithm to update the mesh characteristics*.

The vertices in the quadtree are iteratively decimated to decompose the model, yielding a set of embedded subtrees. Therefore, a progressive mesh is computed in the sense that the original model is split between a base mesh (represented by the root of the quadtree) and a set of detail meshes. Each detail mesh is represented by the set of subtrees pruned at each simplification step. Note that the quadtree constrains the dataset since it imposes a hierarchy on the vertices. Our RD-optimal solutions are given for this restricted setting. In contrast, the general problem of selecting the K "best" vertices[3] without imposing a hierarchy is NP-hard [1]. Our algorithm has the following interesting property: the decrease in distortion is *quasi*-monotonic[4] across rates in the progressive representation. We show that this property cannot be guaranteed by a greedy algorithm as in [5, 6, 10, 11, 16]. Finally, the computational efficiency of our framework relies on our linear quadtree data structure [14], which has a *constant-time traversal property*. This property, based on a node indexing scheme that we named *z-ordering*, allows browsing of the data-structure in minimal time and yields computationally optimal implementations for all optimization steps. Our solution provides a fast simplification algorithm as well as an efficient data structure and is therefore suited for processing large datasets. We compare our algorithm to its greedy counterpart (which considers only vertices in leaf nodes) and give results for a large database of DETDs. Note that even our greedy version is novel compared to previous works using the same connectivity [5, 16] since our *update mechanism* allows to recompute the errors of the mesh after a decimation in *minimal computational time*.

1.2 State of the art

Simplification algorithms and complexities Given a model of N vertices, an approximation is constructed by selecting a subset of $K < N$ vertices representing the original model as accurately as possible. Strategies to achieve this task vary as research gains momentum. Two main optimization approaches exist: decimation and refinement. The former consists in removing vertices from the mesh whereas the latter inserts vertices in a coarse approximation. Two related techniques on the quadtree triangulation were presented by Lindstrom *et al.* [16] and Duchaineau *et al.*[5]: the former consists in an $\Theta(K \log N)$ algorithm to refine the mesh. Any vertex can be inserted, eventually requiring additional triangle *forced-split* to conserve the mesh continuity, and their algorithm is in fact the refinement counterpart of our algorithm. The later provides a dynamic mesh representation based on an equivalent formulation on a binary tree (like in [2]). An $\Theta(K \log N)$ algorithm is proposed to insert and decimate vertices in the model. Two priority queues are used for this purpose, but their ordering require the distortion measure to be monotonic preventing to use the l_2 or l_∞ norm. Only vertices in the tree leaves are inserted or decimated, therefore each optimization split or merge two triangles. Methods for semi-regular and irregular meshes were presented by Gross *et al.*[8], Garland *et al.*[6] and Hoppe *et al.*[10, 11]. Their algorithms have complexities $\Theta(N^2)$, $\Theta((K+N)\log K)$ and $\Theta(N \log N)$ respectively.

Key issues Besides [16], all above methods support only a constant increase or decrease in triangles at each optimization step. However in [16], the optimization algorithm does not take into account forced-splits to choose the best vertex to insert. For TINs [6, 10, 11], the lack of hierarchy in the model requires to add or remove a fixed number of triangles at each optimization step. The above review led us to identify a set of basic ingredients for an efficient mesh simplification framework: having an efficient data structure is essential, since many processing steps strongly rely on the ability to efficiently query the dataset. Examples include the recomputation of the connectivity and the update of the errors after a simplification step. An efficient data structure must then provide: information on the spatial orientation of the model, fast and simple access to the dataset and minimal storage. Moreover, the efficiency of the data structure is tightly coupled with the mesh connectivity. For example, a mesh with regular connectivity lowers the storage complexity and allows to better design data access mechanisms. A mesh with simple connectivity does not usually achieve the quality of irregular triangulations, but provides a more flexible framework in terms of optimization and efficiency. The processing of large meshes requires scalable algorithms, as described in [11], with low computational complexity. The algorithm must also be able to handle multiple error metrics. Usually, a simplification algorithm evaluates the error as a displacement of vertices either in world-space or in screen-space. However, more sophisticated error metrics including, for example, the

[3]The mesh built on theses vertices minimizes the distortion criterion.

[4]The monotonicity is conserved at best.

attributes of the vertices (such as color, shading, texture) should be considered. Our framework is built under these considerations.

1.3 Problem formalization and approach

We propose an optimal decimation algorithm for quadtree-based meshes optimizing the model in an operational RD framework. Decimation approaches as in [8, 10, 11] yield better approximation quality than refinement algorithms [5, 6, 16] as reported in [2, 11]. Refinement algorithms are classified as greedy and cannot in general achieve optimal solutions (see [7] for more details). The connectivity of our model is given by the quadtree triangulation, which is obtained with a *binary subdivision scheme* as used in [5, 16]. We represent a simplified mesh M simply by a set of vertices (possibility having attributes), since the connectivity is implicit for our class of meshes[5]. The set M_0 (i.e. *original mesh*) contains information about all the vertices forming the mesh. Our approach states the simplification of the mesh as an optimization problem as follows: we start with the original mesh M_0 built on N vertices, or $M_0 = \{v_0, \dots, v_{N-1}\}$. A simplified model M is denoted $|M| < |M_0|$ since it contains less vertices than the original model. We define mesh functionals u as functions $u : M \to \mathbb{R}$. We use two mesh functionals: a cost functional C and a distortion functional D. The vector-valued function $\mathbf{u}(M) = (C(M), D(M))$ represents the cloud of all possible simplified models in the RD plane. The operational RD function

$$D_T(C) = \min_{|M| \leq |M_0|} \{D(M) | C(M) \leq C\} \tag{1}$$

specifies the optimal trade-off between rate and distortion in the restricted setting. For a cost budget C, the solution $(C(M), D(M))$ returned by $D_T(C)$ satisfies the constraint at minimal incurred distortion. The set of optimal meshes $|\mathcal{B}| < \dots < |M_1| < |M_0|$, where $\mathcal{B}$ denotes to the base mesh, corresponds to a series of embedded restricted subtrees. Each optimal configuration is represented by a couple $(C(M_i), D(M_i))$ on the curve bounding the convex hull of all configurations (Figure 2). This curve is called the *constrained operational rate-distortion curve*.

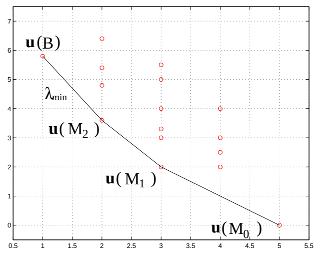

Figure 2: Mesh simplification in the RD plane: each possible simplified model is represented by a point in the plane. The optimal configurations are on the curve bounding the convex hull.

1.4 Contribution

This section gives an overview of our contributions and the characteristics of our framework.

Operational rate-distortion framework Our models are optimized in an operational rate-distortion framework. At each optimization step, the optimal algorithm chooses to decimate the vertex giving the best trade-off between rate and distortion. Namely, the vertex *maximizing the decrease in rate while minimizing the increase in distortion* is decimated and optimizations at variable rate are performed.

Progressive description A mesh is decomposed into a base mesh and a set of detail meshes. Our algorithm ensures at best that the distortion decreases monotonically through successive refinements of the model.

[5]Although we use the converse and more efficient solution to store the mesh: for datasets with implicit parameterization, it suffices to store one float per vertex and spend a few bits on the connectivity to describe the mesh univocally (see [9]).

Optimal solutions in an operational rate-distortion sense Our algorithm returns the optimal approximations in an operational RD sense and outperforms greedy approaches. We compare our algorithm with its greedy counterpart and explain the advantages of our approach.

Efficient data structure Our framework takes advantage of our efficient linear quadtree data structure having a constant-time traversal property. This feature allows to access and process the dataset in minimal computational time.

Computationally optimal update algorithm We state and implement a computationally optimal algorithm to update the errors of the mesh after a simplification step. This mechanism can also be used in [5, 16].

Compact representation A property of regular meshes is the compactness of the representation: for models having implicit parametric information (as for DETD), the vertices can be expressed with a single float and few bits are spent on encoding the connectivity.

Low complexity and scalable algorithm The complete optimal decomposition in a progressive mesh is computed in $\Theta(N \log^2 N)$ time and an approximation of K vertices is obtained in $\Theta(K \log^2 N)$. We compare our algorithm to its greedy counterpart which has complexities $\Theta(N \log N)$ and $\Theta(K \log N)$ respectively. The low computational complexity allows us to process large datasets.

Spatial continuity Each simplification step must preserve the spatial continuity of the mesh, or equivalently the subtree corresponding to the approximation must be restricted. We give the size of the problem of computing such a tree and solve it in optimal computational time using our quadtree data structure.

Support of multiple simplification metrics The simplification metric can be selected by the user according to the application. We show that our framework only requires the cost functional to be monotonically increasing with the tree size, while the distortion functional can be arbitrary.

1.5 Outline of the paper

This paper is organized as follows: in Section 2.1 we briefly introduce the quadtree triangulation, or binary subdivision scheme. In Section 2.2 and 2.3, we explain how the constant-time traversal property is obtained for the tree and how the mesh characteristics can be stored without redundancy. We present then our operational RD framework in Section 3 and give an RD-optimal algorithm to obtain the solutions minimizing the distortion criteria. Section 4 analyses the algorithm properties and the optimality of the solutions is discussed in Section 4.3. In Section 5, we present experimental results for a large database of DETD. Finally, in Section 6, we comment our results and propose future investigations.

2 Mesh connectivity and storage

2.1 Binary subdivision scheme

The connectivity of our meshes is obtained with a *binary subdivision scheme* (BSS) like in [5, 16]. Four steps of the subdivision scheme, each denoted l, are depicted in Figure 3. Assume that $d = 2l$, then after l steps the mesh contains $n = 2 \cdot 4^d$ triangles. This construction naturally maps the amplitude matrix of a DETD, where the parameterization is implicit. The successive meshes generated by recursive subdivision are embedded, in the sense that we can obtain any mesh from a coarser approximation by simply splitting some of the triangles, which leads to the following definition:

Definition 2.1 (Embedding)
Given two meshes obtained with a BSS M_i and M_j, we say that M_j is embedded in M_i iff $|M_j| < |M_i|$ and $M_j \subset M_i$. Note that when $M_i \equiv M_0$, the inclusion is not necessary.

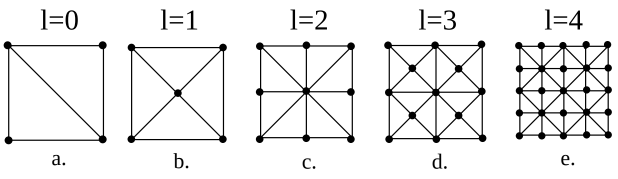

Figure 3: Binary subdivision scheme: (a) the base is represented by a square formed by two triangles. At each subdivision step l, (b)-(e) each triangle is subdivided in two.

This class of regular meshes are commonly stored in a quadtree [16] and also known as *quadtree triangulations*. In Figure 4, we depict the quadtree-mesh correspondence. To have a clear representation of the quadtree in the figure, we link together only the nodes having a common father, and located at the same level.

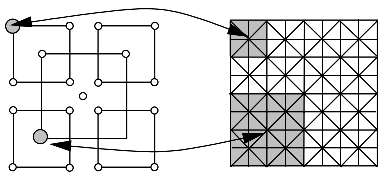

Figure 4: Quadtree structure storing a mesh obtained with a binary subdivision scheme. To have a clear representation of the quadtree in the figure, we link together only the nodes having a common father, and located at the same level. The arrows link nodes to their corresponding regions in the mesh.

2.2 Z-ordering of the quadtree nodes

A quadtree like the one in Figure 4 can be spatially organized by assigning a spatial location for the child nodes with respect to their father. Figure 5a shows a possible spatial organization for the child node indices, whereas Figure 5b shows an alternate one, named *z-ordering* (see solid arrows). We organize a quadtree node p and its children as in Figure 5b. This particular ordering has the property that the difference between any pair of horizontal nodes is 1, whereas the difference between any pair of vertical nodes is 2 (see dashed arrows).

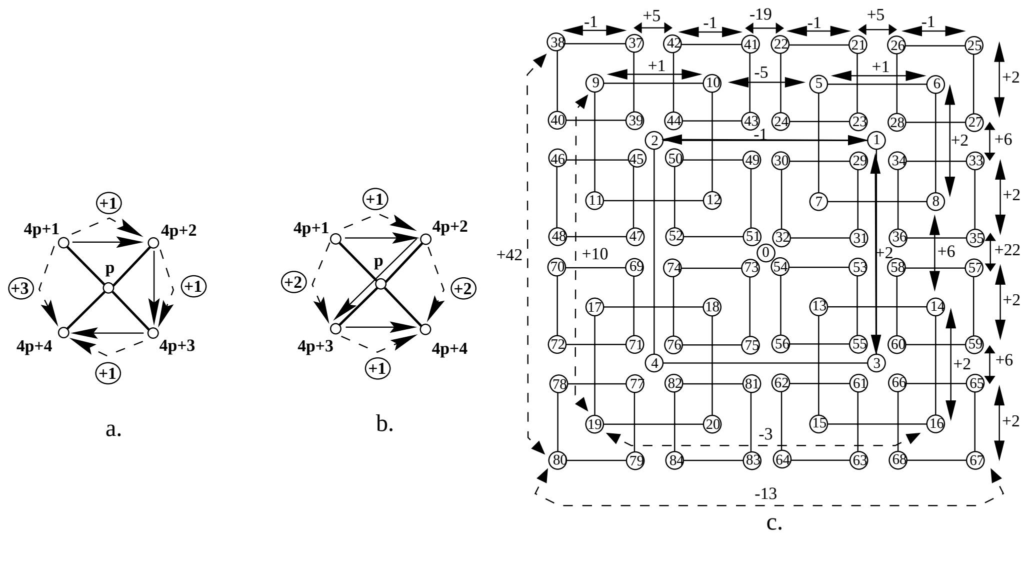

Figure 5: Spatial organization of the nodes: (a) Example. (b) z-ordering. (c) Spatial index organization using the z-ordering for a quadtree of depth 4. The index differences δ are given on top of the arrows. The dashed arrows represent the distances assuming a toroidal structure (see Theorem 1).

We construct the indexing using the ordering of Figure 5b for the odd quadtree levels, and a vertically mirrored version for the even levels. Figure 5c shows the resulting indexing for a quadtree of depth 4.

Once recursively applied, the z-ordering yields the following key property: *the horizontal/vertical difference between the indices of neighboring pairs is constant for a particular column/row.* In Figure 5c, the index differences, denoted δ, are given on top of the arrows. Call now the *relative level distance* (RLD) the distance r between two neighbor nodes p_1 and p_2 at the same level in the quadtree evaluated as the number of levels to traverse to find their common father, as depicted in Figure 6. Given two such nodes, this distance is obtained with

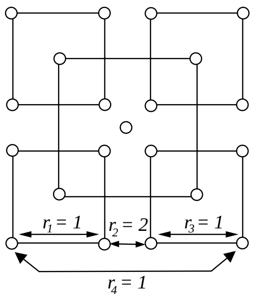

Figure 6: Illustration of the *relative level distance* (RLD) between two neighbor nodes at the same level in the quadtree. The value r on the arrows gives the number of levels to traverse in order to find a common father for the two neighbors.

$$\lfloor \frac{p_1 - 1}{4^r} \rfloor = \lfloor \frac{p_2 - 1}{4^r} \rfloor. \tag{2}$$

We can now derive equations expressing the vertical and horizontal differences δ between the node indices in Figure 5c as a function of r. Moreover, a similar difference (see the dashed arrows in the figure) can be computed for the border nodes with their opposite neighbors, which is equivalent to assume that the quadtree has a toroidal structure. In Appendix A.1, we show that r is obtained with a simple recurrence equation. Theorem 1 give the differences δ and the proofs are given in Appendix A.2.

Theorem 1 (z-ordering)
The horizontal/vertical difference between the indices of neighboring pairs having a relative level distance r is constant for a particular column/row. The horizontal differences are

$$\delta_h(r) = \frac{6}{5}4^r + \frac{1}{5}(-1)^{r+1}, \tag{3}$$

$$\delta_{ht}(r) = \frac{1}{5}4^{r+1} + \frac{1}{5}(-1)^r. \qquad \textit{(toroidal)} \tag{4}$$

The vertical differences are

$$\delta_v(r) = \frac{4}{3}4^r + \frac{2}{3}, \tag{5}$$

$$\delta_{vt}(r) = \frac{2}{3}4^{r+1} - \frac{2}{3}. \qquad \textit{(toroidal)} \tag{6}$$

The recursive nature of the quadtree allows us to compute and store vectors for the solution to (2). We obtain then one vector per quadtree level. Then, (3)-(6) are evaluated on these vectors yielding two vectors per level (one per orientation) to compute all index differences in the quadtree. The z-ordering yields a *constant-time node traversal (CNT) property* for the linear quadtree (see Appendix A.3). The index difference between two arbitrary nodes (not necessarily at the same level) can be obtained by combining (3)-(6), as shown in Appendix A.4.

The CNT property naturally extends to the navigation of a *forest* of quadtrees. It suffices to use (4) and (6) to traverse nodes between quadtrees. Therefore, any mesh can be subdivided into tiles, each stored in a separate quadtree. The partitioning of the mesh allows us to perform *block-based simplification* as in [11], and meshes of arbitrary size can be handled (Figure 7a). Also, meshes with more complex topologies can be constructed in this framework: the simplest example is the storage of a subdivided octahedron [6] using two quadtrees (Figure 7b).

[6]A subdivided octahedron is used to obtain a tessellation for the sphere.

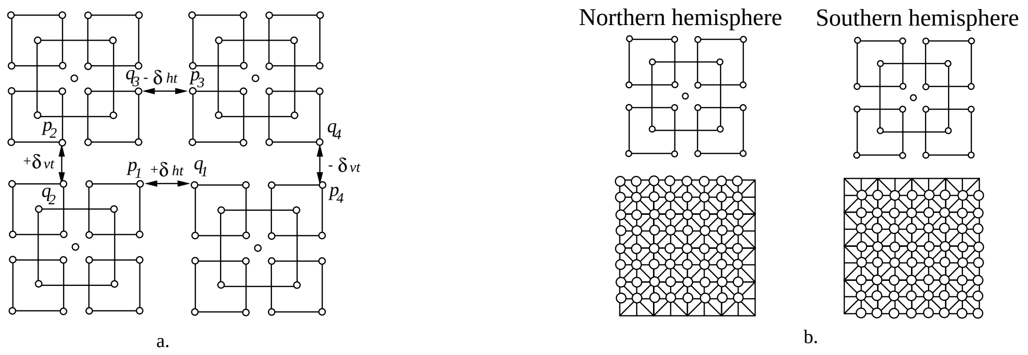

Figure 7: Data structure scalability: (a) Navigation of a quadtree forest. The starting node is p_i and the destination q_i, $i = 1 \dots 4$. The index difference δ is the value to add to p_i to obtain q_i, i.e. $q_i = p_i + \delta$. (b) Storage of a subdivided octahedron. Each hemisphere is represented by a separate mesh, and the octahedron is obtained by merging them one on top of the other using the border vertices. The vertices stored for each hemisphere are represented with a white dot.

2.3 Efficient storage of the mesh

In the quadtree, each node can potentially describe the characteristics (i.e. value and connectivity) of five vertices. However, since neighbor squares share common vertices on their edges, such a description would be highly redundant. Lindstorm *et al.* pointed out this problem in [16]. Their quadtree storage requires to duplicate shared vertices information leading to expensive storage even for small meshes. To avoid redundancy, we store only the description of three vertices in the nodes (the central vertex and two adjacent border vertices). Therefore each node stores 3 floats plus 3 bits (each bit checks the vertex availability) to completely encode the vertices and the connectivity. The quadtree is sufficient to store all descriptions but the ones of the rightmost and bottommost vertices. It suffices to use two additional binary trees to store the characteristics of the remaining vertices. Then each node in the binary tree stores one float plus one bit. In the case of the storage of a subdivided octahedron, no binary tree is required since two quadtrees suffice to store the information of all vertices without redundancy (Figure 7b).

3 Algorithm

3.1 Introduction

In this section, we present an $\Theta(N \log^2 N)$ algorithm to find the solutions to (1) for the full range of possible constraints (leading to a progressive description). Our algorithm is derived from the g-BFOS algorithm [4], an optimal tree pruning algorithm used as an optimization tool in compression [13]. Namely, determining $D(C)$ is a discrete convex programming problem, and the algorithm minimizes the functional $J(M) = D(M) + \lambda C(M)$ over all simplified models M of M_0. The Lagrangian multiplier λ corresponds to the slope bounding the convex hull at the optimal setting $(C(M_i), D(M_i))$ (Figure 2). We present now the constraints incurred when decimating an arbitrary vertex in the tree hierarchy. These constraints must be satisfied for each approximation in order to obtain a restricted quadtree, or equivalently a spatially continuous mesh.

3.2 Preserving the spatial continuity of the mesh

A simplified mesh M is obtained by removing vertices from the original set. Once a vertex is removed, the two triangles originally split by this vertex during the BSS are merged. This constraint insures that, in the series of successive approximations, we can always *construct a mesh from a coarser approximation by simply splitting some of the triangles*. In other words, the tree structure is conserved through the simplification process and a progressive decomposition is given by a set of embedded subtrees. The above assertion implies that, after the decimation of a particular vertex, additional vertices will likely be decimated jointly in order to conserve the mesh spatial continuity. In Figure 8a, we represented the merging of the two triangles originally split by vertex v (Figure 3d), however the mesh is not spatially continuous at this point since the merged triangle edges share vertices with smaller neighboring triangles. In Figure 8b, additional vertices are removed jointly such that the resulting mesh is continuous.

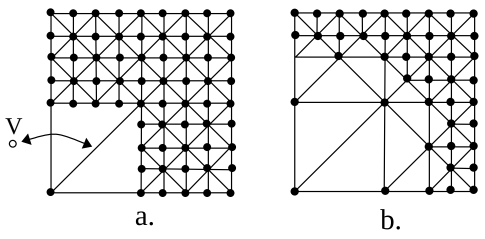

Figure 8: Vertex decimation: (a) The vertex v is removed and the two triangles originally split by this vertex (see Figure 3d) are merged, but the mesh is not yet spatially continuous. (b) Then, additional vertices are decimated jointly yielding a spatially continuous mesh.

We say that each vertex defines a *merging domain*, which is formed by the vertices to be removed jointly with this one. In Figure 9a, the white vertices depict the merging domain of vertex v (the vertex at the center of the set), whereas the black ones connect the *support of the domain*. The support is defined as *the minimal set of triangles covering the domain* when all the vertices inside the merging domain have been decimated (Figure 9b). In Figure 9c, we show the influence, on a rendered mesh, of decimating a vertex in a quadtree node close to the root.

Consider a vertex v and denote by M_v the merging domain attached to the vertex (Figure 9a). The error incurred when decimating vertex v must be evaluated on its merging domain. This consideration is also valid when computing the error incurred by the insertion of a vertex. In [16], the error is only evaluated on v and further variations of the global error caused by the forced splits are not taken into account. The problem of computing the merging domain of a vertex is equivalent to find a *restricted quadtree* [18], which defines a tree corresponding to a continuous mesh.

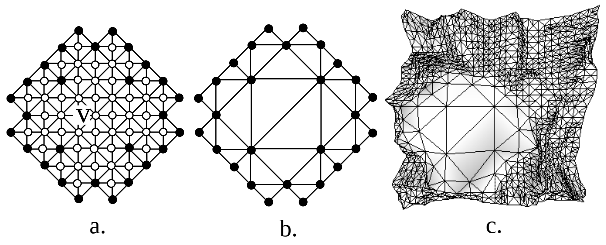

Figure 9: Merging domain: (a) The white vertices represent the merging domain M_v of vertex v. (b) After decimating v (i.e. $|M_v| = 0$), only the black vertices connecting the support of M_v in M remain. (c) The effect of decimating a vertex close to the root in a rendered mesh.

3.3 Greedy against optimal decimation

Although it would be much simpler to decimate only vertices described by the leaf nodes (like in [5, 19]), as in this case the merging domain is trivial to compute, consider the following extreme case: assume that the input mesh of N vertices is totally flat, and would therefore be represented with no increase in distortion by the two initial triangles (Figure 3a). An algorithm considering only the tree leaves still performs N simplification steps in this case. However, the optimal algorithm chooses to decimate the merging domain M_v of the unique vertex v inserted at step $l = 1$ (Figure 3b) and the *whole mesh is simplified to a model of two triangles in a single iteration because removing all vertices is the best rate-distortion trade-off*. Being able to decimate any vertex in the hierarchy allows to perform optimizations at variable rate[7] and allows to obtain the optimal configurations in the RD plane (Figure 10). We compare our optimal algorithm to its greedy counterpart, namely by forcing the algorithm to decimate vertices located in tree leaves. In other words, the greedy version performs optimizations at fixed cost. A greedy algorithm only considers *the short term effects of inserting or decimating a single vertex*, that

[7]Generally defined as the number of triangles.

is, it only considers what happens with the addition or deletion of a pair of triangles in the current mesh. In the next section, we explain how to evaluate the best trade-off between rate and distortion to find the optimal vertex.

3.4 Rate-distortion optimization

Recall that we defined a mesh functional $u(\cdot)$ as a real-valued function on the set of vertices, or $u : M \to \mathbb{R}$. Note that $u(\cdot)$ is also defined for a singleton set, i.e. when M_v contains only one vertex. We further define the *variation* of a mesh functional as $\Delta u(M_v) = u(M_v) - u(\breve{M}_v)$, where $u(M_v)$ is the value of the functional evaluated on the vertices of the merging domain (these vertices are depicted in Figure 9a) and $\breve{M}_v$ is the merging domain M_v when all vertices have been decimated, i.e. $\breve{M}_v = \emptyset$ (Figure 9b). Recall that we use two mesh functionals: the cost C of an approximation and its distortion D with respect to the original model. Then, the variation $\Delta C(M_v)$ is the change in cost and $\Delta D(M_v)$ is the change in distortion when v is decimated. We now present the simplification algorithm.

The algorithm takes a mesh stored in the quadtree as input. For each vertex v, the tuple $\{\Delta C(M_v), \Delta D(M_v), \lambda(v)\}$ is stored. In each quadtree node, an additional $\lambda(v_{\min})$ stores the minimal value $\lambda(v)$ for the subtree rooted at this node. The output of the algorithm is a progressive set of approximations $\mathcal{B} \subset \ldots \subset M_1 \subset M_0$ (see definition 2.1), where $\mathcal{B}$ denotes the base mesh of two triangles (Figure 3a). For all approximations M_i, the distortion value $D(M_i)$ is minimized for the cost $C(M_i)$. The algorithm decomposes the mesh until a minimal cost $C_{\min}$ is reached. To obtain the full decomposition, we set $C_{\min} = C(\mathcal{B})$. For example, when the cost is defined as the number of triangles, then $C_{\min} = 2$ (Figure 3a). We give below the pseudo-code for the algorithm. Each important step refers either to a later section or its computational complexity is given (see the marginal notes on the right hand-side). Note that our formulation assumes that the cost function C is monotonically increasing with the tree size. This point will be discuss later in Section 4.2.

ALGORITHM

```
▷ Input: THE FULL RESOLUTION MESH $M_0$.
◁ Output: A PROGRESSIVE SET OF APPROXIMATIONS $\mathcal{B} \subset \ldots \subset M_1 \subset M_0$
• Initialization:
1 FOR ALL VERTICES $v$, COMPUTE $\lambda(v) = \frac{-\Delta D(M_v)}{\Delta C(M_v)}$.          Section 4.2.3
2 $M \longleftarrow M_0$
3 while $C(M) > C_{\min}$
    4 SEARCH THE OPTIMAL VERTEX $v^\star = \arg\min_{v \in M} \lambda(v)$.      $\Theta(\log N)$
    5 DECIMATE $M_{v^\star}$, I.E. COMPUTE $M \longleftarrow M - M_{v^\star}$.     Section 4.2.1
    6 UPDATE THE ANCESTORS $a$ OF $M_{v^\star}$, I.E. $\forall v \in M_{v^\star}$,      Section 4.2.2
        • $\Delta\mathbf{u}(M_a) \longleftarrow \Delta\mathbf{u}(M_a) - \Delta\mathbf{u}(M_v)$, WHERE $a$ IS A FATHER VERTEX OF $v$.
        • RECOMPUTE $\lambda(a)$.
    7 STORE $M_{v^\star}$.
8 end
```

At the initialization phase (line 1), the magnitude $\lambda(v) = \frac{-\Delta D(M_v)}{\Delta C(M_v)}$ of the slope for every configurations obtained by decimating M_v is computed (Figure 10a). The value $\lambda(v)$ represents the trade-off between cost and distortion. In contrast, a greedy algorithm which only considers the decimation of one single vertex (i.e. in a leaf node) actually imposes $\Delta C(M_v) = c$, where c is a constant given by the cost functional. At each step, the optimal vertex $v^\star$ with minimal slope $\lambda_{\min}$ is found (line 4) and $M_{v^\star}$ is decimated (line 5), as depicted in Figure 10b. At the end of each iteration, we search for a set of *ancestors* $A_{M_{v^\star}}$ of $v^\star$, defined as the *vertices in the mesh which error has been modified after the simplification step*. Recall that an efficient aspect of our algorithm is *its ability to find these vertices and update their characteristics in the simplified mesh*, therefore for each ancestor $a \in A_{M_{v^\star}}$, its value $\Delta\mathbf{u}(M_a)$ is updated as $\Delta\mathbf{u}(M_a) \longleftarrow \Delta\mathbf{u}(M_a) - \Delta\mathbf{u}(M_v)$, where a is a father vertex of v and $v \in M_{v^\star}$ (line 6). Two categories of ancestors for $M_{v^\star}$ are identified: the *father vertices*, i.e. the vertices a such that $M_{v^\star} \subset M_a$, and the *neighbor vertices*, i.e. the vertices a such that $M_{v^\star} \cap M_a \neq \emptyset$. After the decimation of $M_{v^\star}$, *only the error of the father and neighbor vertices have been modified.* We explain how to find the ancestors of $M_{v^\star}$ in Section 4.2.2. Finally, line 7 records the simplification step to reconstruct the progressive model after the decomposition. Figures 10b-d depict three iterations of the algorithm. In Section 4, we prove that the total cost to compute the complete decomposition is $\Theta(N \log^2 N)$.

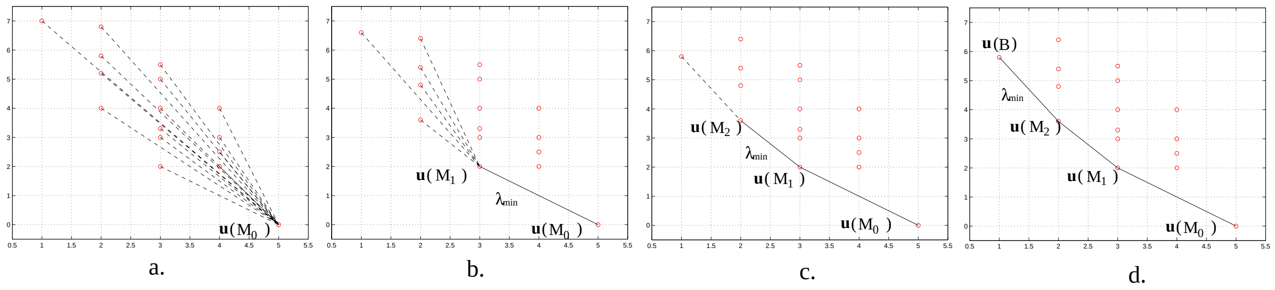

Figure 10: Algorithm iterations: (a) initialization (line 1), (b)-(d) the algorithm iterates to find the vertices on the convex hull (lines 4-7).

4 Analysis and complexity

This section analyses the algorithm and validates our results. We examine each step in detail and compute the total computational complexity. Then, we address the optimality of the solutions.

4.1 Preliminaries

Polynomial algorithm Once $M_{v^\star}$ has been decimated, we need to find a set of *ancestor vertices* $A_{M_{v^\star}}$ to update the errors of the mesh. The set of ancestors can be seen as the vertices which merging domains intersect or contain the one of the decimated vertex (i.e. their characteristics have been modified by the decimation of $M_{v^\star}$). From a quadtree point a view, the father vertices are located in the branches toward the root, whereas the neighbor vertices are distributed in the neighborhood of the decimated vertex. This neighborhood is composed of nodes at the same level, as well as nodes of lower branches. Figure 11a unveils the quadtree structure storing a part of the mesh: intuitively, the mesh layer has been replaced by its corresponding quadtree. The dark region depicts the nodes influenced by the decimation of $M_{v^\star}$. The arrows represent a proportion of the extensive traversal required in the quadtree to visit the neighbor vertices. Therefore $|A_{M_{v^\star}}|$ can only have polynomial size if $M_{v^\star}$ remains local around $v^\star$ in a mesh of increasing size. Figure 11b represents the merging domain of vertex $v^\star$ by its support [8]. In a mesh of increasing size, the domain converges to the irregular octagon represented by the dark shape. Proposition 1 states that $M_{v^\star}$ remains concentrated around $v^\star$ which implies that $|A_{M_{v^\star}}|$ has polynomial size since there is a finite number of merging domains intersecting and containing $M_{v^\star}$. The proof of the property is purely geometric and aims at expressing the growth of the octagonal support as a pair of geometric series (one per radius), see Appendix A.5 for more details. Using this property, we can conclude that the problem of updating the vertex characteristics has polynomial computational complexity as well.

Proposition 1 (Limit surface of the merging domain)
The domain covered by $M_{v^\star}$ in a mesh of increasing size is bounded by the irregular octagon with dimensions $r_{\max} = \sqrt{2}$ and $r_{\min} = \frac{3}{2}$ in Figure 11b.

[8] In the figure, vertex $v^\star$ is only represented for clarity, although it is not part of the mesh anymore.

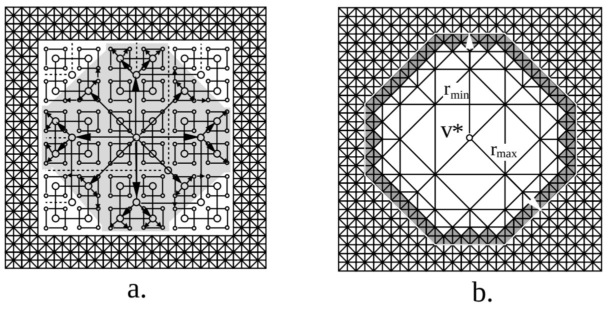

Figure 11: (a) The figure unveils the quadtree structure storing a part of the mesh. The dark region represents the nodes influenced by the decimation of a vertex contained in the central node. The arrows illustrate the extensive traversal required to visit the neighbor vertices. (b) Limit surface of the merging domain of vertex $v^\star$: the white region represents the support for the actual mesh density, whereas the dark region shows the limit surface in mesh of infinite density.

Error metric Although a perceptual metric for the error would be preferable, the l_2 and l_∞ norms are commonly used distortion measures to compute the distance between an approximation and the original model. In [11], Hoppe *et al.* conclude that the use of the l_2 norm is inadequate. However in [6], Garland *et al.* find the l_2 norm to be a better measure of approximation than the l_∞ norm because it leads to steadier RD curves. Actually, Garland's conclusion on stability is related to the monotonicity of the norms: both l_2 and l_∞ norms are nonmonotonic distortion functionals. Although it is a well-known fact in compression, the monotonicity issue is rarely addressed in computer graphics. Practically, a norm is nonmonotonic when an approximation with K vertices has a smaller distortion than an approximation with $L > K$ vertices which can happen when the vertex locations cannot be modified. The added value of preserving the monotonicity for a progressive decomposition is that it insures that each refinement of the model will decrease the distortion. The fact cannot be guaranteed by a greedy algorithm. In Appendix A.6, we give an example of nonmonotonicity in 1D.

Evaluation of a mesh functional A mesh functional is evaluated on a merging domain M_v. The size of the merging domain can be given in term of vertices, denoted $|M_v|$ or in term of triangles forming the support, denoted $|M_v|_\triangle$. During the simplification process, the size of M_v varies since vertices are decimated from the original mesh M_0. We give the size in two cases: first, when the domain is complete (i.e. no vertex has been decimated) or equivalently when the domain is fully triangulated (Figure 9a). Then, we give the size when all vertices have been decimated. In this case, only $|M_v|_\triangle$ is non null and the support is obtained (Figure 9b). The sizes of M_v are given in term of triangles and the two cases are respectively denoted $\max|M_v|_\triangle$ (fully triangulated) and $\min|M_v|_\triangle$ (support). Their asymptotical behavior gives important clues about the complexity of the algorithm.

A mesh of N vertices, where $\sqrt{N} = 2^d + 1$, has $n = 2 \cdot 4^d$ triangles, where d is the depth of the quadtree. N and n are linked by

$$n = (N-1)(\frac{1}{2} + \frac{1}{2^d})^{-1}. \tag{7}$$

The size of M_v is a function of the total number of triangles n in the mesh and the step l of the BSS at which the vertex was inserted (Figure 3a-e). The asymptotical sizes $\max|M_v|_\triangle$ and $\min|M_v|_\triangle$ are given by Theorem 2. The proof of this theorem is given in Appendix A.7. The theorem below gives the sizes in the worst case and in expectation. The worst case is computed by considering only the unique vertex inserted at step 1 of the BSS (Figure 3b) in an unbounded and infinitely dense mesh. This case is only important theoretically, however we are more interested in the expected case: the values in expectation are computed as the average sizes over all the vertices in the tree. The computational complexity obtained in practice is given by the expected cases.

Theorem 2 (Asymptotical size of the merging domain)
Consider a mesh containing $n = 2 \cdot 4^d$ triangles with $d > 0$. The maximum and minimum numbers of triangles covering the merging domain are given by

$$\max|M_v|_\triangle(n) \in \Theta(n), \tag{8}$$

$$\min|M_v|_\triangle(n) \in \Theta(\sqrt{n}) \tag{9}$$

in the worst case and

$$E[\max|M_v|_\triangle(n)] \in \Theta(\log n), \tag{10}$$
$$E[\min|M_v|_\triangle(n)] \in \Theta(c) \tag{11}$$

in expectation.

4.2 Mesh decimation and update

4.2.1 Computing the restricted quadtree in optimal time

We explain now how to decimate a vertex $v^\star$ and its merging domain $M_{v^\star}$. In the greedy case where only the vertices in the leaves are considered, we have $M_{v^\star} = \{v^\star\}$. In the optimal case (where we can have $|M_{v^\star}| > 1$), we must ensure that after the decimation of $M_{v^\star}$ the mesh is still spatially continuous, or equivalently that the quadtree is restricted. As depicted in Figure 11a, computing the restricted quadtree requires extensive navigation. However, our framework provides a data structure having a constant-time traversal property. Since each vertex in $M_{v^\star}$ has to be visited, the cost of the computationally optimal algorithm is $\Theta(\log N)$ (Theorem 2). By using Theorem 1, we construct an algorithm to solve the problem exactly in this size. The computational complexity to visit the entire domain and compute the restricted quadtree is therefore $\Theta(\log N)$. Note that in the greedy case, the complexity is simply $\Theta(1)$.

4.2.2 Updating the errors of the mesh

After each decimation, the characteristics $\Delta\mathbf{u}$ of the mesh have to be recomputed. In this section, we explain an efficient update mechanism to achieve this task. Recall that $A_{M_{v^\star}}$ denotes the ancestor vertices of $v^\star$ and that this set corresponds to the vertices a which $\Delta\mathbf{u}(M_a)$ have been modified by the decimation of $M_{v^\star}$. To update the mesh in the general decimation case, we need therefore to find $A_{M_{v^\star}}$ as defined below:

Definition 4.1 (Ancestor vertices of $M_{v^\star}$)
The vertices a such that $\Delta\mathbf{u}(M_a)$ is modified after the decimation of $M_v^\star$ are given by

$$A_{M_{v^\star}} = \bigcup_{v \in M_v^\star} A_v, \tag{12}$$

where $A_v = \{a \mid v \in M_a\}$ are the father vertices of v. In particular, $A_{v^\star}$ is called the set of father vertices of $M_{v^\star}$ and

$$\forall a \in A_{v^\star}, M_{v^\star} \subset M_a, \tag{13}$$

whereas for all $v \in M_{v_\star}$ with $v \neq v^\star$, the vertices in the sets A_v are called neighbor vertices of $M_{v^\star}$ and

$$\forall a \in \bigcup_{v \in M_{v^\star}, v \neq v^\star} A_v, \quad M_{v^\star} \cap M_a \neq \emptyset. \tag{14}$$

We give now an algorithm to compute A_v. The set $A_{M_{v^\star}}$ is simply found by iterating the algorithm for all vertices $v \in M_{v^\star}$, as stated in the above definition.

Construction of the set of father vertices A_v We give now an algorithm to compute A_v: consider a vertex v inserted at step l of the BSS, then in M_0, there are at most two vertices a_1, a_2 inserted at step $l-1$ such that $M_v \subset M_{a_1}$ and $M_v \subset M_{a_2}$. This key fact is a consequence of the BSS and is the essence of the constraints to solve in order to update the mesh characteristics. Note that, for a vertex located on the border, only one such vertex exists. The set of father vertices is therefore built recursively from v until the root vertex (i.e. the unique vertex inserted at step 1, see Figure 3a) is reached. The construction is illustrated in Figure 12a. The algorithm requires extensive navigation (Figure 12b) in the quadtree and can be written efficiently using the constant-time traversal property given by Theorem 1. Finally, since the size of the bottom up traversal has $\Theta(\log N)$ steps, we have that $|A_v| \in \Theta(\log N)$. Therefore considering Definition (4.1) and (10) in Theorem 2, we have therefore that

$$|A_{M_{v^\star}}| \in \Theta(\log N)\Theta(\log N) \in \Theta(\log^2 N), \tag{15}$$

in the general decimation case where $|M_{v^\star}| > 1$.

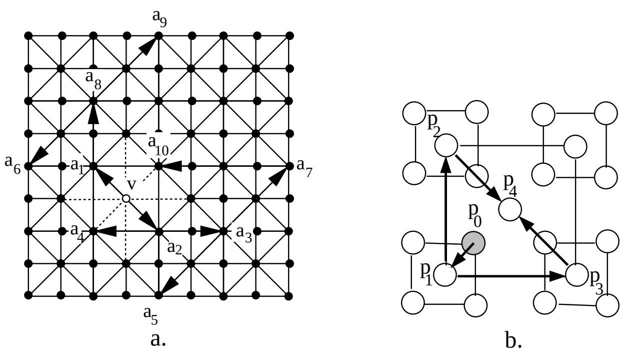

Figure 12: (a) The father path for vertex v is given by the vertices a_i, $i = 1, \ldots, 10$. (b) The algorithm on the quadtree constructing the father set A_v achieves a bottom-up traversal of the data structure.

Update of the set of ancestor vertices $A_{M_{v^\star}}$ The decimation of $M_{v^\star}$ has to be done according to the hierarchy imposed on the vertices by the BSS. Otherwise, the update of the ancestor vertices will be redundant and incorrect, as shown in the following example: consider $v_1, v_2 \in M_{v^\star}$, then for $v_1, v_2 \neq v^\star$ we have in general that (Figure 13a) $M_{v_1} \cap M_{v_2} \neq \emptyset$. Assume now that v_1 and v_2 have a common ancestor a, then the update $\Delta\mathbf{u}(M_a) - \Delta\mathbf{u}(M_{v_1}) - \Delta\mathbf{u}(M_{v_2})$ is not redundant if and only if for all the vertices s such that $M_s \subset M_{v_1}$ and $M_s \subset M_{v_2}$ we have $M_s = \emptyset$, i.e. $\Delta\mathbf{u}(M_{v_1})$ and $\Delta\mathbf{u}(M_{v_2})$ have already been updated. For example, consider v_1 and v_2 in Figure 13a and 13b inserted at step $l = 2d - 1$. Then both vertices have 4 vertices (the vertices s and the vertex w in the figure) which merging domain is contained in M_{v_1} or M_{v_2}. So in the example we have $M_{v_1} \cap M_{v_2} = \{w\}$. The intersection of M_{v_1} and M_{v_2} is represented by the light gray region in the figure. If the update of $\Delta\mathbf{u}(M_a)$ is carried on before the update of $\Delta\mathbf{u}(M_{v_1})$ and $\Delta\mathbf{u}(M_{v_2})$, then *the value of* $\Delta\mathbf{u}(M_w)$ *will be subtracted twice to* $\Delta\mathbf{u}(M_a)$. A correct update is achieved as follow: $\Delta\mathbf{u}(M_{v_1})$ and $\Delta\mathbf{u}(M_{v_2})$ are first updated (the update is depicted by the arrows in Figure 13a), followed by the update of $\Delta\mathbf{u}(M_a)$ (Figure 13b). The following proposition summarizes the procedure to correctly update the mesh:

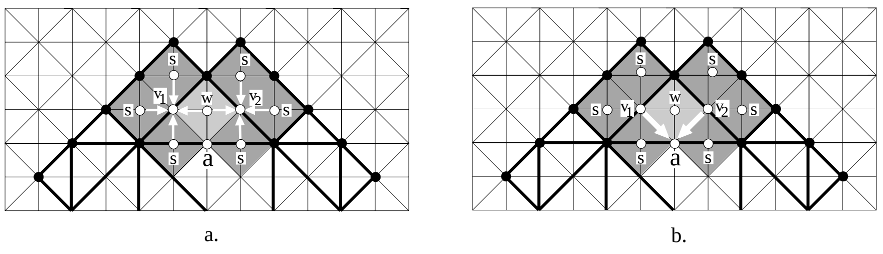

Figure 13: Avoiding redundant updates: (a) the merging domains M_{v_1} and M_{v_2} intersects at vertex w. To update $\Delta\mathbf{u}(M_a)$ without redundancy, $\Delta\mathbf{u}(M_{v_1})$ and $\Delta\mathbf{u}(M_{v_2})$ are first updated, (b) followed by the update of $\Delta\mathbf{u}(M_a)$.

Proposition 2 (Update of the mesh)
Consider a vertex v inserted at step l of the BSS, then an update of $\Delta\mathbf{u}(M_v)$ is not redundant if and only if for all vertices s inserted at step $l+1$ such that $M_s \subset M_v$, $\Delta\mathbf{u}(M_s)$ has been updated.

4.2.3 Complexity

Initialization It is easy to evaluate the complexity of the initialization step using Theorem 2. For all N vertices, we need to compute the value of the slope $\lambda(v) = \frac{-\Delta D(M_v)}{\Delta C(M_v)}$. This operation requires to evaluate both operators $C(\cdot)$ and $D(\cdot)$ on the merging domain of all vertices. This operation takes $N \cdot E[\max |M_v|_\triangle(n)]$, which proves that the initialization has order $\Theta(N \log N)$.

Decimation and update Assume that $v^\star$ is the optimal vertex to decimate, then we have shown that $|A_{M_{v^\star}}|$ has size $\Theta(\log^2 N)$ is expectation. Note that it makes sense to use the expected size for $\max |M_v|_\triangle$ since the worst case exists only for the unique vertex inserted at step 1 of the BSS (Figure 3b) and the complexity must be evaluated for a complete decimation of the tree. Therefore, the computational complexity to decimate and update the mesh for one iteration has order $\Theta(\log^2 N)$ for the optimal algorithm. For the greedy algorithm, where $|M_{v^\star}| = 1$, then this cost is simply $\Theta(\log N)$

Total complexity Both algorithms are executed in $\Theta(N)$ steps. In the case of the greedy algorithm, exactly N steps are completed, whereas in the optimal case less steps can be executed since the optimal trade-off might not be obtained with a vertex in a leaf node. In the optimal case, $\Theta(\log^2 N)$ operations are performed for each step, compared to $\Theta(\log N)$ in the greedy case. Therefore, the algorithm complexities in general are $\Theta(N \log^2 N)$ for the optimal algorithm and $\Theta(N \log N)$ for the greedy algorithm, to obtain the complete decomposition of a mesh of N vertices.

4.3 Optimality

In this section, we first discuss the conservation of the monotonicity through the simplification process, then address the optimality of the solutions.

Monotonicity Assume two vertices v_0 and v_1 such that $v_1 \in M_{v_0}$ (then $\Delta C(M_{v_1}) < \Delta C(M_{v_0})$) and that $\Delta D(M_{v_1}) > \Delta D(M_{v_0})$. Such case exists with the l_2 or the l_∞ norms since both are nonmonotonic. Consider now that v_1 is decimated before v_0, then the slope $\lambda(v_0)$ will change sign after the update (see Section 3.4). In the optimal algorithm, the choice of the vertex having the minimal slope λ implicitly translates into the condition given by Proposition 3. To be candidate for decimation, a vertex must fulfill this condition. We give the proposition below and prove it in Appendix A.8.

Proposition 3
Consider two vertices v_0 and v_1 inserted respectively at step l and $l+1$ of the BSS and that $v_1 \in M_{v_0}$. Assuming that the cost functional C is monotonically increasing (i.e. $\Delta C(M_v) \geq 0$), then the merging domain M_{v_1} is decimated before M_{v_0} if and only if

$$\frac{\Delta D(M_{v_0})}{\Delta D(M_{v_1})} > \delta > 1, \tag{16}$$

where $\delta = \Delta C(M_{v_0})/\Delta C(M_{v_1})$. When v_0 does not meet condition (16), then M_{v_0} is called a nonmonotonic merging domain.

Following our initial example, the above proposition ensures that vertex v_0 will be chosen before v_1 since it results in a better RD tradeoff. We examine now the property requirements for the mesh functionals: Proposition 4 states that the monotonicity of the cost functional is sufficient to insure the monotonicity of the rate-distortion curve. Recall that this assumption was made in the pseudo code of the algorithm. The proof is given in Appendix 88.

Proposition 4 (Monotonicity of the RD Curve)
Assume an iteration of the algorithm where $\forall v$, $\Delta C(M_v) \geq 0$ and $\Delta D(M_v)$ is arbitrary, then given a vertex $v^\star$ with $\lambda(v^\star) = \min_{v \in M} \lambda(v)$ and the set of ancestor $A_{M_{v^\star}}$, for all updated vertices $a \in A_{M_{v^\star}}$ we have $\lambda(a) > \lambda(v^\star)$.

Let us now complicate slightly our initial example: consider that an ancestor $a \in A_{M_{v_0}}$ is decimated before v_0 and unfortunately $v_1 \in M_a$ (Figure 14). Then the update $\Delta D(M_{v_0}) - \Delta D(M_{v_1})$ will take place and the nonmonotonicity will not be avoided. Therefore Proposition 4 can only be proved under the assumption that no such vertex exists. By the following example, we have shown that because of the overlapping of the domains, nonmonotonicities cannot be technically avoided even in the general decimation case. However, the above situation is rare in practice because: *the sets A_{M_v} are small compared to the total number of vertices (15)* and *the datasets (i.e. the vertices) are locally smooth in general*. Our experiments will show that the optimal RD curve is still very stable compared to the greedy one, even in very “chaotic” situations like the one depicted in Figure 19.

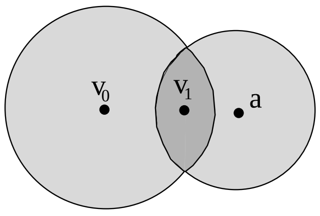

Figure 14: Suboptimal case of the algorithm: $v_1 \in M_a \cap M_{v_0}$ and M_{v_0} is a nonmonotonic merging domain (see Proposition 3). Decimating M_a before M_{v_0} will provoke a nonmonotonicity of the RD curve if $\Delta D(M_{v_1}) > \Delta D(M_{v_0})$.

Optimality of the solutions Our algorithm is derived from an optimal tree pruning algorithm given in [4]. The authors give an algorithm to optimally prune trees without restrictions, i.e. a tree does not have to be restricted to be a valid solution. In our case, we considered only the solutions given by a restricted quadtree to insure the spatial continuity of the mesh. The optimality of their algorithm has been proven in [4]. The example given in the previous section gives us clues whether the optimal solutions achieved in the quadtree-restricted case. We have seen that the overlapping between domains prevented to make the optimal choice in some rare cases. Therefore, we conclude that most of the time the optimal solutions are obtained, however the overlapping can lead to slightly suboptimal solutions in some particular situations.

5 Experimental results

5.1 Terrain dataset and metric

In this section, we compare our optimal and greedy algorithms using a large set of DETD. Our dataset represents the southwest region of Switzerland (the Alps) and is composed of 388 terrains of size 257×257. For the experiments, we use the squared l_2 norm as a measure of distance between the original mesh and the approximations (Figure 15a). When using this norm, each amplitude z_i is projected in the triangle reconstructed when the vertex is decimated. For example, z_1' is obtained by projecting z_1 in the triangle formed by vertices $(x_5, y_5, z_5), (x_6, y_6, z_6), (x_9, y_9, z_9)$, as depicted in Figure 15b. The error for each vertex is then given by $\delta_i = |z_i - z_i'|^2$. To evaluate the error $D(\breve{M}_{v^\star})$, all the vertices in the merging domain (Figure 9a) are projected in the support once $M_{v^\star}$ is decimated (Figure 9b). The distortion functional is therefore defined by $D(\breve{M}_{v^\star}) = \sum_{v \in M_{v^\star}} \delta_v$. We evaluate the cost of an approximation in term of triangles.

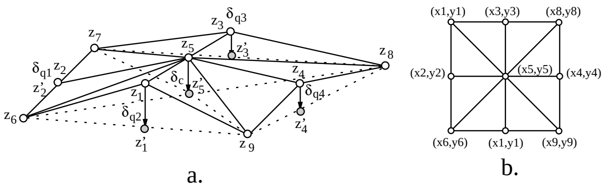

Figure 15: Computation of the error in squared l_2 norm for a triangulated cell stored in a quadtree node. (a) Side view, (b) top view.

Consider $e_{\max}$ the maximum error obtained by decimating all the vertices in the mesh and e_r the error measured between an approximation of rate r and M_0. We compute the peak-to-signal noise ratio, or *PSNR* between the two meshes. The error, denoted E_r, is given in dB and is defined as

$$E_r = 10 \log_{10}\left(\frac{e_{\max}}{e_r + 1}\right). \qquad \text{(dB)} \tag{17}$$

For section 5.4, we use an alternate definition given by

$$E_r = -10 \log_{10}(e_r + 1). \qquad \text{(dB)} \tag{18}$$

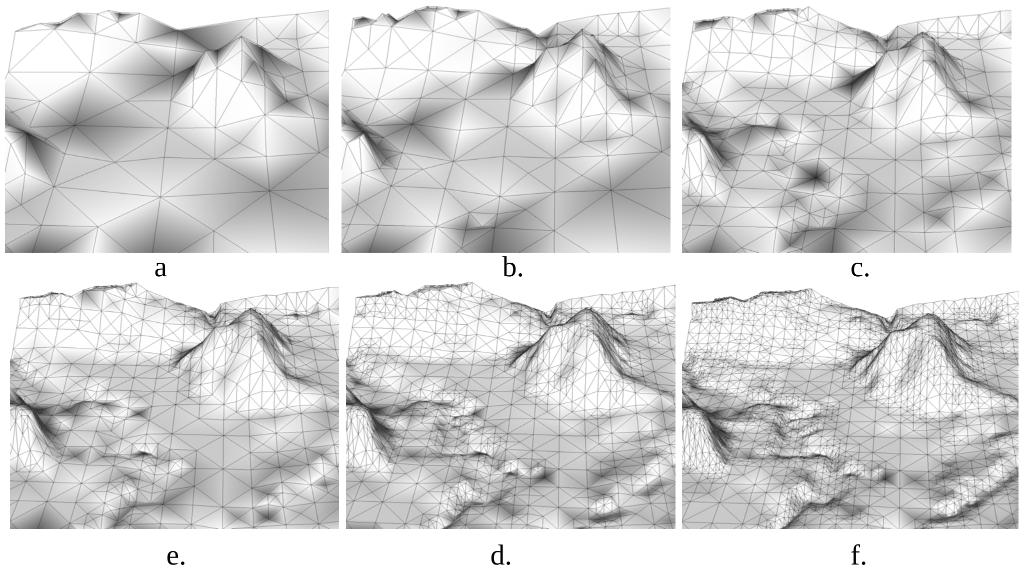

Figure 16: Progressive refinement of a terrain model: (a) a coarse resolution of the mesh is generated with 200 triangles (PSNR 14.9 dB), successively refined to (b) 400 (PSNR 18.46 dB), (c) 800 (PSNR 22.2 dB), (d) 1600 (PSNR 25.914 dB), (e) 3200 (PSNR 29.81 dB) and finally (f) 6400 triangles (PSNR 33.8 dB). Each approximation minimizes the global squared l_2 error for each triangle budget. The mesh in f. contain 5% of the total number of triangles $n = 131072$.

5.2 Progressive decomposition

We first give an example of progressive refinement using an optimal decomposition of a 257×257 DETD. Figure 16a-f shows its progressive refinement starting from a coarse approximation of 200 triangles (Figure 16a), successively refined to 400 (Figure 16b), 800 (Figure 16c), 1600 (Figure 16d), 3200 (Figure 16e) and finally 6400 triangles (Figure 16f). Each approximation minimizes the global error. The final approximation uses 5% of the total number of triangles and achieves a PSNR of 33 dB. The individual PSNRs for each approximation are given in the figure caption. In Figure 17a-d, we give the RD curves obtained with the optimal and the greedy algorithms for the mesh in Figure 16a-f for the corresponding rate range. In each figure, the top curve is the greedy RD curve, whereas the bottom curve is the optimal RD curve. Also, the rate is given by the magnitude d of the mesh density $n = 2 \cdot 4^d$ and the error is computed using (17). Figure 17a shows the global error at low rate. We can see here that the greedy RD curve is highly nonmonotonic, whereas the optimal RD curve is quasi-monotonic across rates. The nonmonotonicity obtained with the optimal curve illustrates the example in Section 4.3. Figure 17b-c depicts the error at increasing rate respectively.

5.3 Average and maximum gain

Table 1: Gain comparison for the optimal versus the greedy algorithm using 257×257 terrain tiles

ranges in triangles	max. gain (dB)	rate (tri.)	max. expected gain (dB)	expected rate (tri.)
100 - 2600	2.805	203	0.983	439
2600 - 26300	0.73	25106	0.254	16822
26.3k - 130k	3.68	110942	1.154	102991

We compare now the greedy and optimal approaches on the full dataset of 388 terrains. For each terrain, a full decomposition is generated using the greedy and the optimal algorithms. We compare the performances in three cases: at low rate (between 0.1% and 2% of n), at average rate (between 2% and 20% of n) and finally at high rate (between 20% and 99.9% of n). Since each terrain have different characteristics, their approximation error is normalized by the maximum error $e_{\max}$ (obtained at minimal rate $r = 2$). We fix the maximum gain to 50 dB, then

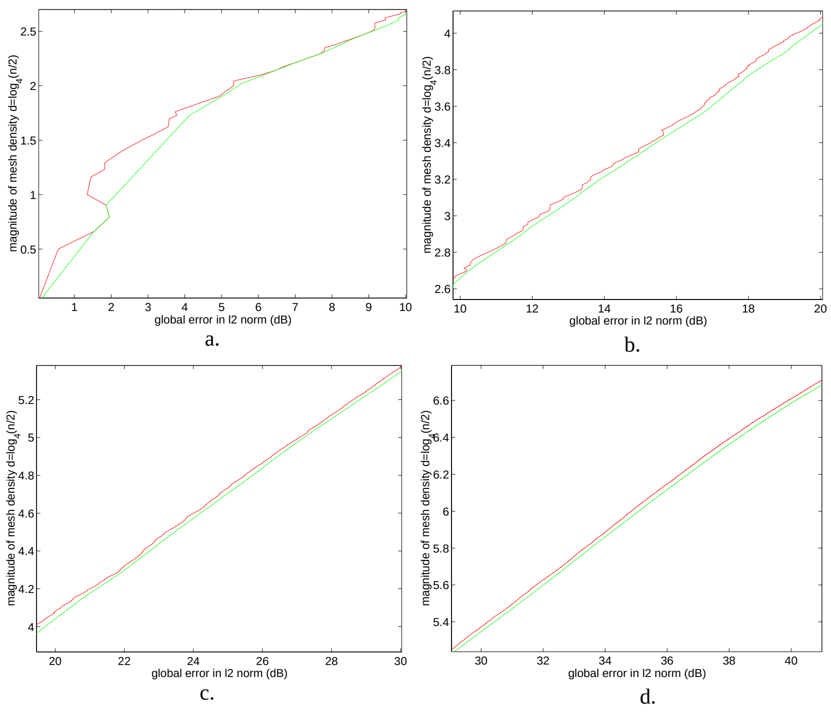

Figure 17: Rate-distortion curves for the decomposition of the model in Figure 16. In each figure, the top curve is the greedy RD curve, whereas the bottom curve is the optimal RD curve. The rate is given by the magnitude d of the mesh density $n = 2 \cdot 4^d$: (a) $0 < d < 2.5$ (i.e. $2 < n < 64$), (b) $2.6 < d < 4$ (i.e. $73 < n < 512$), (c) $4 < d < 5.3$ (i.e. $512 < n < 3104$), (d) $5.4 < d < 6.7$ (i.e. $3565 < n < 21618$).

the normalized errors are further multiplied by 10^5. Consider E_r as defined in (17), then we have $0 \le E_r \le 50$ for all terrain measurements $0 \le e_r \le e_{\max}$. The normalization allows to compare the decomposition of all the terrains in the database. In Table 1, we give for each rate range: the *maximum gain* observed and the *maximum expected gain* of the optimal decomposition over the greedy approach. For each value, we give respectively the rate and the expected rate for which these values were obtained. Figure 18a-c show the average optimal and greedy rate-distortion curves obtained for each rate range. In each figure, the optimal and greedy curves are respectively represented by the bottom and top curves.

5.4 Conservation of monotonicity

We show now that the error of the successive approximations given by the greedy algorithm is rarely monotonically decreasing (see Garland *et al* for more examples [6]). In Figure 19a, the top and bottom curves show respectively the optimal and greedy RD curves obtained for the decomposition of the terrain in Figure 19b. The curves are shown for rates lower than 450 triangles (horizontal axis in the graph), and the error on the vertical axis is given by (18). The error curve given by the greedy algorithm is frequently unstable at low rates, which is mainly due to shortsighted optimizations. The optimal algorithm conserves much more efficiently the monotonicity of the curve, as shown in the figure. Other common norms like l_∞ are also been reported as unstable [6].

5.5 Timings

We give now the timings obtained to fully decompose a mesh into a progressive representation with the optimal algorithm using a C++ implementation running on an Intel Pentium III at 500 Mhz. We give both the time spent on the initialization and the time spent on the decomposition of the mesh. Recall that the initialization step consists in computing for each vertex the global error variation incurred when the vertex is decimated and that this step has cost $\Theta(N \log N)$. The optimal algorithm to decompose the model has magnitude $\Theta(N \log^2 N)$. Table 2 gives the measured values.

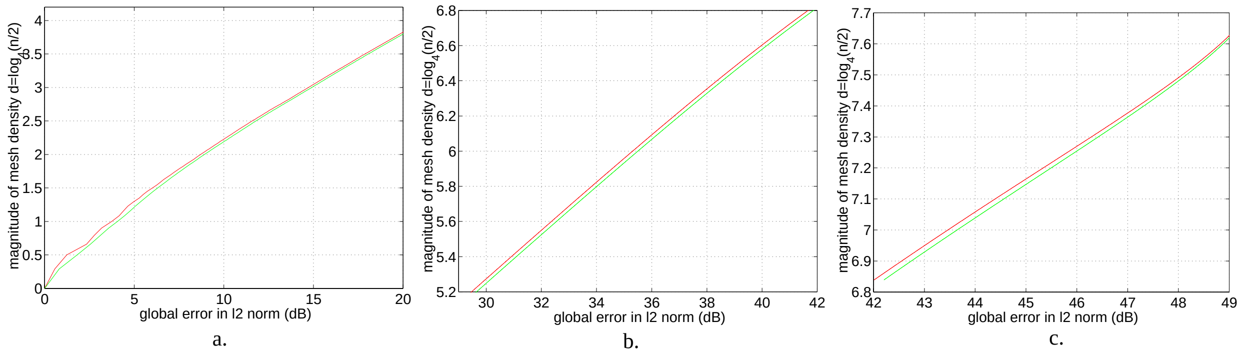

Figure 18: Average rate-distortion curve for the optimal and greedy algorithm: curves for rate (a) between 100 - 2600 triangles, (b) between 2600 - 26300 triangles, and (c) between 26.3k - 130k triangles. In each figure, the optimal and greedy curves are respectively represented by the bottom and top curves.

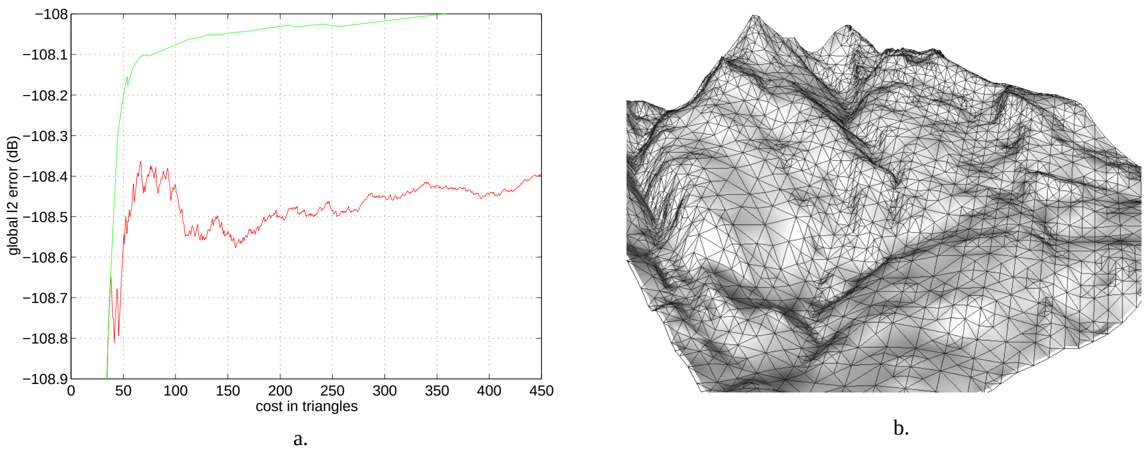

Figure 19: Example of nonmonotonicity of the greedy algorithm at low rate. (a) The top curve is given by the optimal algorithm whereas the bottom one is given by the greedy decimation. (b) The DETD used to generate the RD curves.

6 Conclusion and future work

We presented a scalable simplification framework for meshes obtained with a binary subdivision scheme. We proposed a compact and efficient quadtree data structure and optimized meshes in an operational RD sense. we explained how the RD-optimal solutions could be achieved by performing *optimizations at variable rate* as well as how to update the errors of the mesh after a simplification step at minimal computational cost. Our solution have several advantages:

- the optimal algorithm always outperforms the greedy optimization, i.e. the optimal solutions achieve the lowest distortion at few additional cost,
- the variable-rate optimization approach allows to conserve at best the monotonicity of the distortion across rates,
- minimal requirements for the specification of the cost and distortion functionals provide high flexibility for the user to target their definition according to the application,
- the functionals are only evaluated during the initialization stage, allowing to keep their utilization apart from the optimization process using an efficient update mechanism,
- our framework is computationally efficient and can handle large datasets.

We delineate now future areas of investigation that could take advantage of our framework:

Table 2: Timings measured for a C++ implementation of the optimal algorithm

$\sqrt{N}$	$N \log N$	$N \log^2 N$	mesh density (tri.)	init. time (s)	decomposition time (s)
33	1	1	2048	0.003	0.057
65	4.6	5.5	8192	0.011	0.30
129	21.2	29.5	32766	0.05	1.7
257	96	153	131072	0.24	11.4
513	431	770	524288	1.07	48
1025	1913	3792	2097152	4.81	238

sophisticated metrics Computer graphics models are usually heterogenous sources of informations. For example, often textures are defined additionally to the mesh. An interesting area of research would be the development of metrics to optimize jointly the mesh and its attributes in order to control their contribution to the overall quality of the approximation.

compression Recently, compression techniques for regular subdivision schemes have shown promising [12]. A compression scheme taking advantage of our variable-rate optimization method as well as our hierarchical data structure would provide a mean to encode meshes in an operational RD sense.

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, 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.

[3] G. Chartrand and L. Lesniak. *Graphs and Digraphs*. Chapman-Hall, 1996.

[4] 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.

[5] 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.

[6] M. Garland and P.S. Heckbert. Fast polygonal approximation of terrain and height fields. *Internal Report CMU-CS-95-181*, September 1995.

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

[8] M.H. Gross, O.G. Staadt, and R.Gatti. Efficient triangular surface approximation using wavelets and quadtree data structure. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):1–13, June 1996.

[9] I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. *to appear in proceedings of SIGGRAPH*, 2000.

[10] H. Hoppe. Progressive meshes. *Proceedings of SIGGRAPH*, pages 99–108, 1996.

[11] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. *Proceedings of IEEE Visualization*, pages 35–42, October 1998.

[12] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. *to appear in proceedings of SIGGRAPH*, 2000.

[13] K.Ramchandran and M.Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Transactions on Image Processing*, 2(2):160–175, April 1993.

[14] 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.

[15] A.W.F. Lee, W. Swelden, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parametrization of surfaces. *Proceedings of SIGGRAPH*, pages 95–104, 1998.

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

[17] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. *Proceedings of IEEE Visualization*, 1998.

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

[19] M. Tamminen and F.W. Jansen. An integrity filter for recursive subdivision meshes. *Computer Graphics 9*, 4(1):351–363, 1985.

A Appendix

A.1 Recurrence equations for the relative level distance

To use (3)-(6), we need to provide the RLD r. The RLD is depicted in Figure 6 and is found for a pair of nodes p_1, p_2 solving (2). The regular structure of the quadtree allows to find a closed-form expression for the RLD. Call $\mathbf{r}^i$ the RLDs for level i (for example the series $r_i, i = 1, \ldots, 4$ in Figure 6), then we have

$$\mathbf{r}^1 = \begin{bmatrix} 0 & \phi_1 & 0 \end{bmatrix}, \qquad \phi_1 = \begin{bmatrix} 0 \end{bmatrix}, \tag{19}$$

$$\mathbf{r}^i = \begin{bmatrix} i-1 & \phi_i & i-1 \end{bmatrix}, \qquad \phi_i = \begin{bmatrix} \phi_{i-1} & i-1 & \phi_{i-1} \end{bmatrix}. \tag{20}$$

It suffices to compute one vector par level and evaluate (3)-(6) on this vector: call $\Delta_h^i(\mathbf{r}^i)$ and $\Delta_v^i(\mathbf{r}^i)$ the vectors giving the index differences for pairs of horizontal and vertical neighbor nodes respectively, then (3)-(6) are evaluated as follows:

$$\Delta_h^1(\mathbf{r}^1) = \begin{bmatrix} \delta_{ht}(0) & -\delta_h(0) & \delta_{ht}(0) \end{bmatrix}, \qquad \Phi_1 = \begin{bmatrix} -\delta_h(0) \end{bmatrix}, \tag{21}$$

$$\Delta_h^i(\mathbf{r}^i) = \begin{bmatrix} \delta_{ht}(i-1) & \Phi_i & \delta_{ht}(i-1) \end{bmatrix}, \qquad \Phi_i = \begin{bmatrix} -\Phi_{i-1} & -\delta_h(i-1) & -\Phi_{i-1} \end{bmatrix}, \tag{22}$$

$$\Delta_v^1(\mathbf{r}^1) = \begin{bmatrix} -\delta_{vt}(0) & \delta_v(0) & -\delta_{vt}(0) \end{bmatrix}, \qquad \Phi_1 = \begin{bmatrix} \delta_v(0) \end{bmatrix}, \tag{23}$$

$$\Delta_v^i(\mathbf{r}^i) = \begin{bmatrix} -\delta_{vt}(i-1) & \Phi_i & -\delta_{vt}(i-1) \end{bmatrix}, \qquad \Phi_i = \begin{bmatrix} \Phi_{i-1} & \delta_v(i-1) & \Phi_{i-1} \end{bmatrix}. \tag{24}$$

For example, for Δ_h^1, Δ_h^2 and Δ_h^3 we have respectively

$$\Delta_v^1 = \begin{bmatrix} +1 & -1 & +1 \end{bmatrix}, \tag{25}$$

$$\Delta_v^2 = \begin{bmatrix} +3 & +1 & -5 & +1 & +3 \end{bmatrix}, \tag{26}$$

$$\Delta_v^3 = \begin{bmatrix} +13 & -1 & +5 & -1 & -19 & -1 & +5 & -1 & +13 \end{bmatrix}, \tag{27}$$

whereas for Δ_v^1, Δ_v^2 and Δ_v^3 we have respectively

$$\Delta_v^1 = \begin{bmatrix} -2 & +2 & -2 \end{bmatrix}, \tag{28}$$

$$\Delta_v^2 = \begin{bmatrix} -10 & +2 & +6 & +2 & -10 \end{bmatrix}, \tag{29}$$

$$\Delta_v^3 = \begin{bmatrix} -42 & +2 & +6 & +2 & +22 & +2 & +6 & +2 & -42 \end{bmatrix}, \tag{30}$$

as depicted in Figure 5c.

A.2 Proof of Theorem 1

The index differences in the quadtree are proved by induction. For each proofs, we show that the case $r = 0$ correspond to the distance in Figure 5 and then prove the equation in the general case.

For δ_h, we have: $\delta_h(0) = 1$ which corresponds to the horizontal distance between two nodes having the same father. Equation (3) can be can be rewritten

$$\delta_h(r) = 4^r + \sum_{n=0}^{r-1} 4^n(-1)^{n+r+1}. \tag{31}$$

Using the induction step

$$\delta_h(r) = 4\delta_h(r-1) + (-1)^{r-1}, \tag{32}$$

we show that

$$\begin{aligned} \delta_h(r) &= 4(4^{r-1} + \sum_{n=0}^{r-2} 4^n(-1)^{n+r}) + (-1)^{r-1} \\ &= \frac{6}{5}4^r + \frac{1}{5}(-1)^{r+1}, \end{aligned} \tag{33}$$

For δ_v, we have: $\delta_h(0) = 2$ which corresponds to the horizontal distance between two nodes having the same father. Equation (5) can be can be rewritten

$$\frac{\delta_v(r)}{2} = 4^r - \sum_{n=0}^{r-1} 4^n. \tag{34}$$

Using the induction step

$$\delta_v(r) = 4\delta_v(r-1) - 2, \tag{35}$$

we show that

$$\begin{aligned} \frac{\delta_v(r)}{2} &= 4(4^{r-1} - \sum_{n=0}^{r-2} 4^n) + 1 \\ &= \frac{2}{3}4^r + \frac{1}{3} \end{aligned} \tag{36}$$

For δ_{th}, we have: $\delta_{th}(0) = 1$ which is the index difference between two horizontal nodes assuming a torus structure. Equation (4) can be rewritten as

$$\delta_{th}(r) = \sum_{n=0}^{r} 4^n (-1)^{n+r}. \tag{37}$$

Using the induction step

$$\delta_{th}(r) = 4\delta_{th}(r-1) + (-1)^r, \tag{38}$$

we show that

$$\begin{aligned} \delta_{th}(r) &= 4(\sum_{n=0}^{r-1} 4^n (-1)^{n+r-1}) + (-1)^r \\ &= \frac{1}{5}4^{r+1} + \frac{1}{5}(-1)^r, \end{aligned} \tag{39}$$

Finally for δ_{vh}, we have: $\delta_{tv}(l=0) = 2$ which is the index difference between two vertical nodes assuming a torus structure. Equation (6) can be rewritten as

$$\delta_{tv}(r) = 2\sum_{n=0}^{r} 4^n. \tag{40}$$

Using the induction step

$$\delta_{tv}(r) = 4\delta_{tv}(r-1) + 2, \tag{41}$$

we show that

$$\begin{aligned} \delta_{tv}(r) &= 8(\sum_{n=0}^{r-1} 4^n) + 2, \\ &= \frac{2}{3}4^{r+1} - \frac{2}{3}, \end{aligned} \tag{42}$$

which concludes the proof. $\surd$

A.3 Constant-time navigation

We explain now how to use the results in the previous sections to achieve constant-time navigation. Assume knowing the index of a node p_s in the tree and need to access p_t from this node. Nodes p_s and p_t are further called *the starting node* and *the target node* respectively. Constant-time traversal is achieved whenever we can find Δ such that

$$p_t = p_s + \Delta, \tag{43}$$

with p_s and p_t arbitrary. The difference Δ is a function of p_s and a geometric relation $\mathsf{G}(p_s, p_t)$ between both nodes. Examples of geometric relations are depicted in Figure 20. In Theorem 1, we gave a mean to implement G_0 (i.e. horizontal) and G_1 (i.e. vertical) in Figure 20. Then,

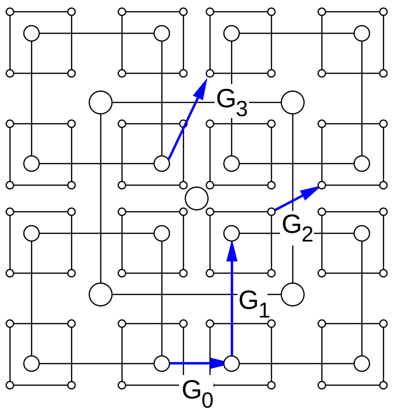

Figure 20: Geometric relations between pair of nodes

in Appendix A.1, we explained how to use these relations for a pair of nodes given their RLD. The geometric relations are *additive*, then in Figure 20 we have

$$\mathsf{G}_2 = \mathsf{G}_0 \oplus \mathsf{G}_1, \tag{44}$$

where $\oplus$ denotes the successive application of G_0 and G_1. Similarly, G_3 and G_4 can be implemented with a series of relations G_0 and G_1. Note that for G_3 and G_4, we also need to traverse levels in the tree.

Consider first the relation G_0 and G_1. To find Δ as a function of p_s, we proceed as follows: each level $0 \le i \le d-1$ in the quadtree can be seen as a $2^i \times 2^i$ grid of nodes. Consider a node p at level i, we can compute a position $[g_h, g_v]$ for the node on the level grid. Then, the coordinates g_h and g_v are used to access Δ_h^i and Δ_v^i respectively. Note that Δ_h^i and Δ_v^i have both size $2^i + 1$. Therefore, for p_s

- its *western neighbor* is given by $p_t = p_s - \Delta_h^i[g_h]$,
- its *eastern neighbor* is given by $p_t = p_s + \Delta_h^i[g_h + 1]$,
- its *northern neighbor* is given by $p_t = p_s - \Delta_v^i[g_v]$,
- its *southern neighbor* is given by $p_t = p_s + \Delta_v^i[g_v + 1]$.

Finding the coordinates $[g_h, g_v]$ The coordinates are found by expressing the local index of p_s in the level grid in base 4, named *qword* of p_s. Then, we define a coordinate system for the grid and find the location $[g_h, g_v]$ in this system with a simple sum of 2×2 matrices. Setting the origin to the top leftmost node leads the following four matrices

$$\mathsf{F}_0 = \mathbf{0}, \qquad \mathsf{F}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \mathsf{F}_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \mathsf{F}_3 = \mathbf{I}_2, \tag{45}$$

where each matrix corresponds to a coefficient in base 4. The qword, denoted $\mathbf{q}_{p_s} = \{q_{i-1}, \dots, q_0\}$ has length i and is computed with $p'_s = p_s - \frac{1}{3}(4^i - 1)$, i.e. the local index of p_s in the grid at level i. In Section 2.2, we explained that the z-ordering is flipped between odd and even levels. Therefore, (46) depicts the transformation τ to apply to odd bits q in $\mathbf{q}_{p_s}$:

$$\begin{aligned} q : 0 &\xrightarrow{\quad\tau\quad} 1, \\ q : 1 &\xrightarrow{\quad\tau\quad} 0, \\ q : 2 &\xrightarrow{\quad\tau\quad} 3, \\ q : 3 &\xrightarrow{\quad\tau\quad} 2. \end{aligned} \tag{46}$$

Then $[g_h, g_v]$ is given by

$$\begin{bmatrix} g_h \\ g_v \end{bmatrix} = \sum (\mathsf{F}_{q_{2i}} + 2 \cdot \mathsf{F}_{\tau(q_{2i+1})}) \begin{bmatrix} 4^i \\ 4^i \end{bmatrix}. \tag{47}$$

Consider the following example: node 8 is located at level 2 and has local index 3. Hence, its qword is $\mathbf{q} = \{0, 3\}$. Therefore,

$$\begin{aligned} \begin{bmatrix} g_h \\ g_v \end{bmatrix} &= (\mathsf{F}_3 + 2 \cdot \mathsf{F}_{\tau(0)}) \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \\ \begin{bmatrix} g_h \\ g_v \end{bmatrix} &= (\mathsf{F}_3 + 2 \cdot \mathsf{F}_1) \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}. \end{aligned} \tag{48}$$

Finally using (29), for node 8

- its *western neighbor* is $p_t = 8 - \Delta_h^2[3] = 7$,
- its *eastern neighbor* is $p_t = 8 + \Delta_h^2[4] = 11$,
- its *northern neighbor* is $p_t = 8 - \Delta_v^2[1] = 6$,
- its *southern neighbor* is $p_t = 8 + \Delta_v^2[2] = 14$,

as it can be verified in Figure 5c. Note that a eastern neighbor is found assuming a toroidal structure for the quadtree.

A.4 Dealing with arbitrary geometric relations

In the previous section, we show how to achieve constant-time navigation for simple geometric relations such as G_0 and G_1. We show now how to deal with relations like G_2 and G_3 using additivity: a straightforward way to achieve arbitrary traversals is to apply G_0 and G_1 iteratively until reaching the target node. However, for each step of the traversal, we would need to recompute (47). A much efficient way is to compute $[g_h, g_v]$ only for p_s, and then deduce the coordinates of the subsequent nodes in the traversal path. For example, given node p_s at level i having coordinates $[g_h, g_v]$, its northern neighbor has coordinates $[g_h, g_v - 1]$, its north-eastern child node has coordinates $[2g_h + 1, 2g_v]$, etc... Therefore, G_2 is simply implemented as

$$p_t = p_s + \Delta_h^i[g_h + 1] - \Delta_v^i[g_v], \tag{49}$$

whereas G_3 is given by

$$p_t = 4 \cdot p_s + k + \Delta_h^{i+1}[2g_h + 2] - \Delta_v^{i+1}[2g_v], \tag{50}$$

where $k = 1, 2$ whether i is odd or even respectively (see Figure 5b). We give a example for (50): assume that $p_s = 12$ (level 1), then G_3 leads to $p_t = 24$ (Figures 20 and 5c). Namely we have $[g_h, g_v] = [1, 1]$ and

$$p_t = 4 \cdot 12 + 1 + \Delta_h^3[4] - \Delta_v^3[2] = 49 - 19 - 6 = 24. \tag{51}$$

Finally, note that equations such as (3)-(6) in Theorem 1 can be found to directly implement relations such as G_2. In this case, (49) is implemented using a single addition instead of the two obtained using additivity. However, this case is more limiting to implement arbitrary geometric relations.

A.5 Proof of Proposition 1

The support of $M_{v^\star}$ in Figure 11 grows like a geometric series along the radiuses $r_{\min}$ and $r_{\max}$. Denote by i each step of such series, then the triangles at step i are twice smaller than the triangles at the preceding step $i-1$. Series $r_{\min}$ and $r_{\max}$ have common ratio $\frac{1}{2^i}$ and $\frac{1}{\sqrt{2}}$ respectively. Therefore the dimension of the octagon are given by

$$r_{\min} = \lim_{n\to\infty} \frac{1}{2} + \sum_{i=1}^{n} \frac{1}{2^i} = \frac{3}{2} \tag{52}$$

and

$$r_{\max} = \lim_{n\to\infty} \frac{1}{\sqrt{2}} + \sum_{i=1}^{n} \frac{1}{2^{i+\frac{1}{2}}} = \sqrt{2} \tag{53}$$

$\surd$

A.6 Example of non-monotonicity in 1D

Consider the 1D function M_0 of Figure 21a. The simplification is constrained by the tree in Figure 21b, in the sense that the decimation of vertex v_0 requires the decimation of vertices v_1 and v_2, or equivalently, the 1D merging domain of v_0 is $M_{v_0} = \{v_0, v_1, v_2\}$. Recall that the border vertices are not decimated such that the successive approximations conserve the same support. This scheme is similar to the constraints imposed on the dataset by the quadtree in our problem. In this example, the cost functional C computes the number of segments of the approximation, whereas the distortion functional D computes the distance in l_2 norm between the approximation and the original model. We first consider a greedy algorithm decimating only vertices located in leaf nodes. In this case, the algorithm will successively decimate the vertices v_1, v_2 and v_0. Note that the order between v_1, v_2 can be exchanged since they provoke the same increase in distortion. At the bottom of Figure 21a, we show the approximation M_3 and M_2 and Figure 21c shows the corresponding vertices $(C(\cdot), D(\cdot))$ in the rate-distortion plane given by

$$(C(M_0), D(M_0)) = (4,0), \qquad (C(M_2), D(M_2)) = (2,8), \qquad (C(M_3), D(M_3)) = (1,6). \tag{54}$$

By only considering the leaf nodes, the greedy algorithm cannot avoid the nonmonotonicity occurring between M_2 and M_3 as shown in the top curve of Figure 21b. The optimal algorithm will decimate the vertex giving the best trade-off between rate and distortion. This trade-off is given by $\lambda(v) = \frac{-\Delta D(M_v)}{\Delta R(M_v)}$. For the first iteration, we have then the following values

$$\lambda(v_0) = \tfrac{3}{3} = 1, \quad \lambda(v_1) = \tfrac{4}{2} = 2, \quad \lambda(v_2) = \tfrac{4}{2} = 2. \tag{55}$$

Therefore, decimating v_0 represents the best trade-off and avoids the nonmonotonicity. The convex hull corresponding to the optimal solutions is represented by the bottom curve in Figure 21c.

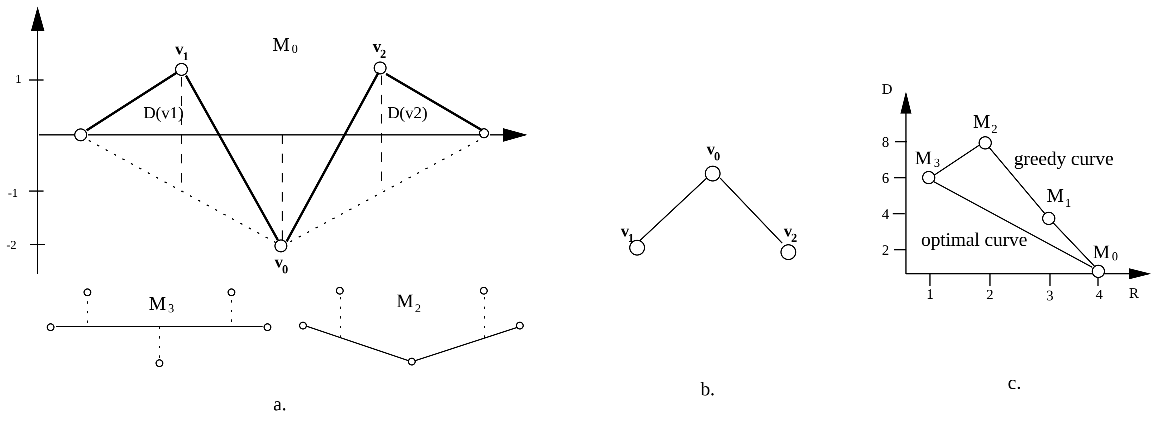

Figure 21: Example of nonmonotonicity in 1D: (a) M_0 is the initial function, whereas M_2 and M_3 are two approximations. (b) The binary tree constraining the dataset. (c) The top curve is the greedy RD curve obtained by successively decimating v_2,v_1 and v_0, whereas the bottom curve is the optimal curve.

A.7 Proof of Theorem 2

The proof is organized as follows: we first explain how the regular grid naturally mapped by the BSS can be split into a quincunx and a Cartesian grid. We will use this fact to study the asymptotical behavior of the merging domain sizes, i.e. the size of the support (Figure 9a) and the size of the fully triangulated domain (Figure 9b). Then, we calculate the number of the triangles in a support of increasing size. For this task, we consider the support as a planar triangulated graph and use its dual representation [3], i.e. a scaffolding tree where each node represents a triangle. We can then compute $\min |M_v|_\triangle$ by simply counting the number of nodes, whereas a weighted evaluation of the same tree leads $\max |M_v|_\triangle$. Both sizes are given in closed-form. Then, we show how an asymptotical analysis of the sizes can be achieved to derive the result of Theorem 2.

Quincunx and Cartesian grid The BSS induces a natural hierarchy on the vertices in M. We illustrate it by splitting the regular grid Ω into the so-called *quincunx* and *Cartesian* grids (Figure 22a and 22b). The sets are denoted Q and C respectively. Since two subdivision steps are necessary for the mesh to map a uniform grid, we have $d = \frac{l}{2}$ with l even. The total size of each grid after $d \geq 0$ subdivisions is respectively given by

$$|Q|^d = \frac{1}{6}4^{d+1} + 2^{d+1} - \frac{8}{3} \qquad \text{and} \qquad |C|^d = \frac{1}{12}4^{d+1} - \frac{1}{3}. \tag{56}$$

In Figures 22a and 22b, the number next to a vertex in the grid indicates at which step of the BSS this vertex is inserted. Note that the four vertices with $l = 0$, forming the two initial triangles (Figure 3a), are actually not inserted since they are used to construct the initial base mesh. Thus we have $|M|^d = |Q|^d + |C|^d + 4$, where d denotes the number of subdivision steps. Finally, note that the quincunx grid is just a rotation of the Cartesian one by $\frac{\pi}{4}$ and their superposition results in the uniform grid Ω (Figure 22c).

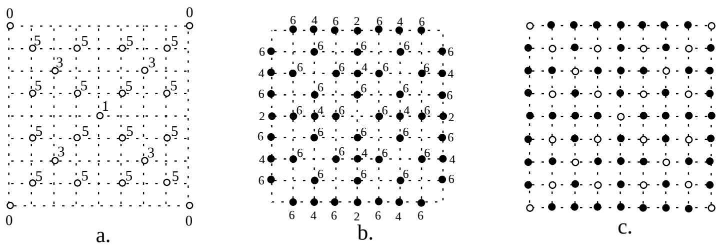

Figure 22: Vertex hierarchy: a. Cartesian grid. b. Quincunx grid. c. Superposition of both grids. In a. and b. the label next to each vertex corresponds to the step at which the vertex is inserted during the BSS.

Scaffolding of the support In Figure 23a-e, we depict a support of increasing size as well as the scaffolding tree representing the dual graph. We first compute the number of triangles $\min |M_v|_\triangle(i)$ as a parameter of the support size i. To do this, it suffices to evaluate the number of nodes in the tree.

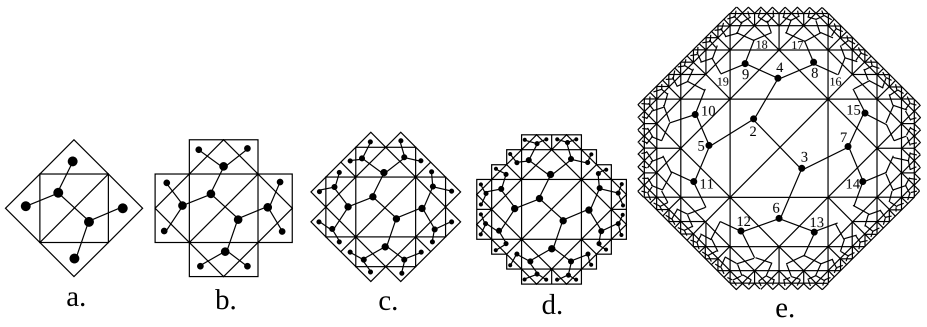

Figure 23: Support of increasing size and its dual graph (scaffolding): support at size (a) $i = 2$, (b) i = 3, (c) i = 4, (d) i = 5, (e) i = 8.

In the series of supports in Figure 23a-e, the tree is balanced (i.e. each node has the same number of child nodes) until $i = 4$. In Figure 23d, some nodes have only one child node. We need then to understand how the tree is unbalanced to compute $\min |M_v|_\triangle(i)$. We further denote $\min |M_v|_{\triangle \mathbf{b}}$ and $\min |M_v|_{\triangle \mathbf{u}}$ the number of triangles in the balanced and unbalanced part of the tree respectively. For $\min |M_v|_{\triangle \mathbf{b}}(i)$, we immediately have that

$$\min |M_v|_{\triangle \mathbf{b}}(i) = \sum_{m=1}^{i} 2^m - 2 = 2^{i+1} - 2. \qquad i \leq 4 \tag{57}$$

We explain how to find $\min |M_v|_{\triangle \mathbf{u}}$ below.

In Figure 24, we represent the scaffolding from node 4 in Figure 23e and enlightened the unbalanced part in the dark zone. We further define $j = i - 4$ and compute $\min |M_v|_{\triangle \mathbf{u}}(j)$. Observe that

- At level $2j - 1$ with $j > 0$, we have 2^j nodes with two children and $2^{j+1} - 2$ nodes with one child node and
- for every level $2j$ with $j > 0$, there are 2^{j+1} nodes with two children and $2^{j+1} - 2$ nodes with one child node.

Consider now that l and k denote respectively the number of odd and even indices. In other words, $l = j - \lfloor\frac{j}{2}\rfloor$ and $k = \lfloor\frac{j}{2}\rfloor$. For clarity,

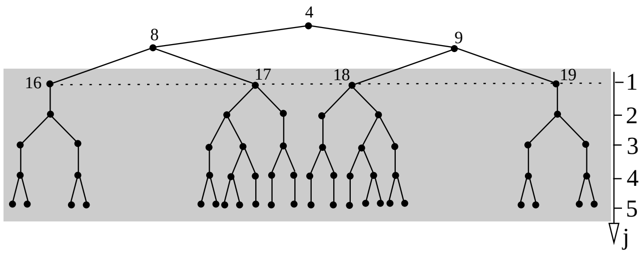

Figure 24: Portion of the tree scaffolding the support from node 4 in Figure 23e.

we compute two separate sums: one for j odd, denoted by $o(l)$, and one for j even, denoted by $e(k)$. We have then

$$o(l) = \sum_{m=1}^{l} (2^{m+2} - 2) = 2^{l+3} - 2l - 8, \tag{58}$$

$$e(k) = \sum_{m=1}^{k} (2^{m+2} + 2^{m+1} - 2) = 2^{k+3} + 2^{k+2} - 2k - 12. \tag{59}$$

To select between (58) and (59) for j arbitrary, we replace l and k in their respective equation and sum (58) and (59), then

$$\begin{aligned} o(j) + e(j) &= 2^{j - \lfloor \frac{j}{2} \rfloor + 3} - 2(j - \lfloor \frac{j}{2} \rfloor) + 2^{\lfloor \frac{j}{2} \rfloor + 3} + 2^{\lfloor \frac{j}{2} \rfloor + 2} - 2\lfloor \frac{j}{2} \rfloor - 20, \\ &= 8(2^{j - \lfloor \frac{j}{2} \rfloor} + 3 \cdot 2^{\lfloor \frac{j}{2} \rfloor - 1}) - 2j - 20. \end{aligned} \tag{60}$$

Therefore, for $i > 4$ we have

$$\min |M_v|_{\triangle \mathbf{u}}(i) = \min |M_v|_{\triangle \mathbf{b}}(4) + 4 \cdot (o(i-4) + e(i-4)), \tag{61}$$

$$= 32(2^{i - \lfloor \frac{i-4}{2} \rfloor - 4} + 3 \cdot 2^{\lfloor \frac{i-4}{2} \rfloor - 1}) - 8i - 18. \tag{62}$$

Finally, we obtain $\min |M_v|_{\triangle}(i)$ by gathering (57) and (61):

$$\min |M_v|_{\triangle}(i) = \begin{cases} 2^{i+1} - 2 & 1 \leq i \leq 4, \\ 32(2^{i - \lfloor \frac{i-4}{2} \rfloor - 4} + 3 \cdot 2^{\lfloor \frac{i-4}{2} \rfloor - 1}) - 8i - 18 & i > 4. \end{cases} \tag{63}$$

Weighting of the tree To find $\max |M_v|_{\triangle}$, we need to weight each node of the tree with the density of triangles contained in the triangle represented by the node. Figure 25a-b represents the support and the fully triangulated domain respectively. Again, we split $\max |M_v|_{\triangle}$ into a sum $\max |M_v|_{\triangle \mathbf{b}}$ for the balanced part and a sum $\max |M_v|_{\triangle \mathbf{u}}$ for the unbalanced part of the tree. For $\max |M_v|_{\triangle \mathbf{b}}$, we weight (57) with $w(m) = 2^{i-m+1}$, and have

$$\max |M_v|_{\triangle \mathbf{b}}(i) = \sum_{m=1}^{i} 2^m \cdot 2^{i-m+1} - 2 = i \cdot 2^{i+1} - 2. \qquad i \leq 4 \tag{64}$$

For $\max |M_v|_{\triangle \mathbf{u}}$, we reuse (58) and (59) and use the weights:

$$w_o(m) = 2^{l+k-2m+2}, \qquad w_e(m) = 2^{l+k-2m+1}, \tag{65}$$

for $o(l)$ and $e(k)$ respectively. Then,

$$\begin{aligned} w_o o(l) + w_e e(k) &= \sum_{m=1}^{l} (2^{m+2} - 2) w_o(m) + \sum_{m=1}^{k} (2^{m+2} + 2^{m+1} - 2) w_e(m), \\ &= 24 \cdot 2^{l+k} + \frac{8}{3} 2^{k-l} - 16 \cdot 2^k - 12 \cdot 2^l + \frac{4}{3} 2^{l-k}. \end{aligned} \tag{66}$$

Again, we successively replace l, k, j and gather (64) and (66) to obtain:

$$\max |M_v|_{\triangle}(i) = \begin{cases} i \cdot 2^{i+1} - 2, & i \leq 4, \\ 128 \cdot c_2 + 4(c_2 \cdot (24 - 12 \cdot c_1^{-1} + \frac{4}{3} c_1^{-2}) & \\ + \frac{8}{3} c_2^{-1} c_1^2 - 16 \cdot c_1), & i > 4, \end{cases} \tag{67}$$

where $c_1 = 2^{\lfloor \frac{i-5}{2} \rfloor}$ and $c_2 = 2^{i-5}$.

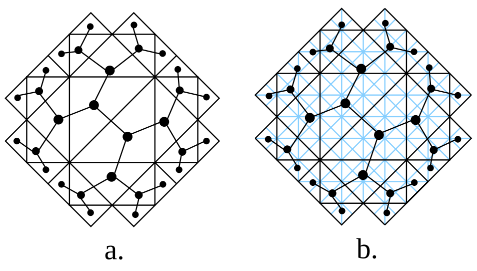

Figure 25: Weighting of the scaffolding tree: (a) the scaffolding covers the support. (b) Each triangle represented by a node in the tree is weighted by the density of triangles.

Expressions as a function of n and l To compute the sizes in M_0, we new to express (63) and (67) as a function of the complexity n of the mesh and the step l of the BSS at which the vertex was inserted. The parameters n, l and i are linked by

$$2^i = n \cdot 2^{-l}. \tag{68}$$

We give now an intuitive argument for (68): for simplicity, consider that $l = 1$ (which is equivalent to assume a scaffolding for the support of the root vertex). Then each time two levels of nodes are added to the scaffolding, the mesh complexity increases by four. Recall that $n = 2 \cdot 4^d$, then replacing n in (68) yields $4^{\frac{i}{2}} = 4^d$, which proves (68). We replace then $i = 2 \cdot \log_4 n - l$ in (63) and (67) to obtain:

$$\max |M_v|_\triangle(l,n) = \begin{cases} (2\log_4 n - l)2^{1-l}n, & l > 2log_4\frac{n}{2} - 4, \\ 128 \cdot c_2 + 4(c_2 \cdot (24 - 12 \cdot c_1^{-1} + \frac{4}{3}c_1^{-2}) & \\ +\frac{8}{3}c_2^{-1}c_1^2 - 16 \cdot c_1), & l \leq 2log_4\frac{n}{2} - 4, \end{cases} \tag{69}$$

where $c_1(l,n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$ and $c2(l,n)) = \frac{2^{-l}n}{16}$. The number of triangles $\min |M_v|_\triangle(l,n)$ is given by

$$\min |M_v|_\triangle(l,n) = \begin{cases} 2^{1-l}n - 2, & l > 2log_4\frac{n}{2} - 4, \\ 32(2^{-(l+4)}n \cdot c_1^{-1} + \frac{3}{2} \cdot c_1) & \\ -16(\log_4 n - \frac{l}{2}) - 18, & l \leq 2log_4\frac{n}{2} - 4, \end{cases} \tag{70}$$

with c_1 as in (69). In both cases, l satisfies $0 < l \leq 2d$. In Figure 26a, we depict $\max |M_v|_\triangle$ (middle curve) and $\min |M_v|_\triangle$ (bottom curve)

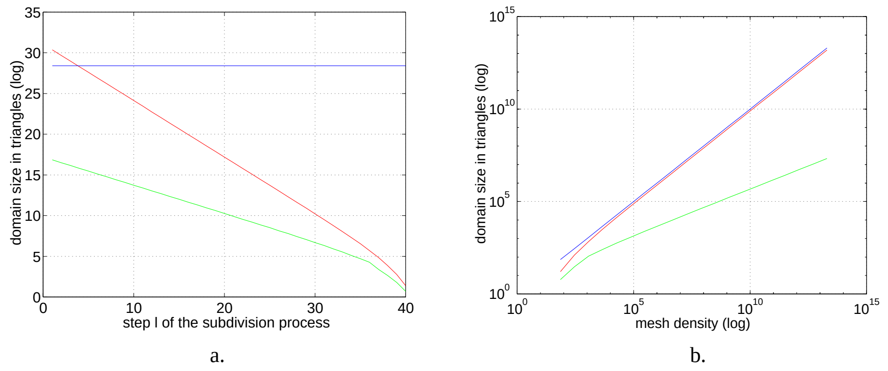

Figure 26: Asymptotical behavior of the merging domain: (a) Merging domain sizes as a function of l ($n = 2 \cdot 4^{20}$). The top constant value depicts the mesh size. The middle and bottom curves represent $\max |M_v|_\triangle$ and $\min |M_v|_\triangle$ respectively. (b) Merging domain size as a function of n ($l = 1$). The top curve represent the mesh complexity n, whereas the middle and the bottom curves are given for $\max |M_v|_\triangle$ and $\min |M_v|_\triangle$ respectively.

as a function of the subdivision step l posing $n = 2 \cdot 4^{20}$. The top curve represents the constant mesh size n. Note that since the mesh is bounded, the maximum value at $l = 1$ is greater than n. Figure 26b shows the asymptotical behavior of $\max |M_v|_\triangle$ and $\min |M_v|_\triangle$ (posing l=1) for $n \to \infty$. The top curve represents the mesh size, whereas the middle and the bottom curves are given respectively by $\max |M_v|_\triangle$ and $\min |M_v|_\triangle$.

Asymptotical sizes of $\max |M_v|_\triangle$ and $\min |M_v|_\triangle$ We give now one more result necessary to derive the asymptotical behavior of $\max |M_v|_\triangle$ and $\min |M_v|_\triangle$ theoretically.

For both $\min |M_v|_\triangle$ and $\max |M_v|_\triangle$, the following properties are obtained from the BSS:

$$|M_v|_\triangle(2^{i+1}-1, n) = |M_v|_\triangle(1, \frac{n}{4^i}), \tag{71}$$

$$|M_v|_\triangle(2^{i+1}, n) = |M_v|_\triangle(2, \frac{n}{4^i}), \tag{72}$$

where (71) holds for all vertices of the Cartesian grid (i.e. l is odd) and (72) holds for all vertices of the quincunx grid (i.e. l is even).

We derive now the asymptotical behavior of $\max |M_v|_\triangle(l,n)$ and $\min |M_v|_\triangle(l,n)$ or equivalently their size when $n \to \infty$ (Figure 26b). We first compute the size in the worst case (posing $l = 1$). We use the short-cuts $\max |M_v|_\triangle(n)$ and $\min |M_v|_\triangle(n)$ to denote respectively $\max |M_v|_\triangle(1,n)$ and $\min |M_v|_\triangle(1,n)$. We have

$$\max |M_v|_\triangle(n) = \begin{cases} (2\log_4 n - 1)n, & n < 128, \\ 7n + \frac{1024}{3}c_1^2 - 64c_1 - \frac{3}{2}c_1^{-1} + \frac{1}{6}c_1^{-2}, & n \geq 128, \end{cases} \tag{73}$$

and

$$\min |M_v|_\triangle(n) = \begin{cases} 2^{1-l}n - 2, & n < 128, \\ n \cdot c_1^{-1} + 48 \cdot c_1 - 16\log_4 n - 26, & n \geq 128, \end{cases} \tag{74}$$

with $c_1(n) = 2^{\lfloor \log_4 n - \frac{5}{2} \rfloor}$. We need now to bound the term $c_1(n)$, thus

$$\frac{1}{8}\sqrt{\frac{n}{2}} \leq c_1(n) \leq \frac{1}{4}\sqrt{\frac{n}{2}}. \tag{75}$$

Immediately, we can conclude for (73) that

$$\max |M_v|_\triangle(n) \in \Theta(n), \tag{76}$$

since n is the dominant term. Using (75), we can now lower bound (74) to obtain:

$$\begin{aligned} \min |M_v|_\triangle(n) &\geq 4\sqrt{n} + 3\sqrt{2}\sqrt{n} - 16\log_4 n - 26, \\ &\in \Omega(\sqrt{n}). \end{aligned} \tag{77}$$

Similarly, we upper bound (74):

$$\begin{aligned} \min |M_v|_\triangle(n) &\leq 2^{\frac{7}{2}}\sqrt{n} + 6\sqrt{2}\sqrt{n} - 16\log_4 n - 26, \\ &\in O(\sqrt{n}). \end{aligned} \tag{78}$$

Using (77) and (78) we have that

$$\min |M_v|_\triangle(n) \in \Theta(\sqrt{n}) \tag{79}$$

Both results (76) and (79) are given in the worst case and are confirmed by Figure 26b.

We compute now the complexities in expectation. To do so, we proceed as follow: 1. we compute the merging domain size for each l and $n \to \infty$, 2. we calculate their weighted sum using the quantity of vertices (56) inserted at each subdivision step l. Finally, the sum is averaged by the total number of vertices N. For the first task, we simply use (71) and (72) which allows to restrict us to two cases, namely $l = 1$ (Cartesian) and $l = 2$ (quincunx) and we vary the complexity n of the mesh instead. This twist allows us to reuse the results (76) and (79). For the second task, we weight the merging domain sizes using the approximation 4^i for the vertices quantities, with $i = 1 \ldots d$ where is $d = \log_4(n) - \frac{1}{2}$, since 4^i is the dominant term in (56). Finally, using (7) we have that

$$\lim_{d\to\infty} \frac{n}{N} = \lim_{d\to\infty} \frac{(N-1)(\frac{1}{2} + \frac{1}{2^d})^{-1}}{N} = 2. \tag{80}$$

Therefore $\frac{n}{2}$ is a good asymptotical value for the number of vertices in the mesh. For $E[\max |M_v|_\triangle(n)]$ we have

$$\begin{aligned} E[\max |M_v|_\triangle(n)] &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^{i+1} m(1, \frac{n}{4^i}) + \sum_{i=0}^{d-1} 4^{i+1} m(2, \frac{n}{4^i})), \\ &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^{i+1}\Theta(\frac{n}{4^i}) + \sum_{i=0}^{d-1} 4^{i+1}\Theta(\frac{n}{4^i})), \\ &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^{i+1} b_1 \cdot \frac{n}{4^i} + \sum_{i=0}^{d-1} 4^{i+1} b_2 \cdot \frac{n}{4^i}), \\ &= 16(b_1 + b_2)(d-1), \\ &= 16(b_1 + b_2)(\log_4 n - \frac{3}{2}) \end{aligned} \tag{81}$$

which proves that

$$E[\max |M_v|_\triangle(n)] \in \Theta(\log n) \tag{82}$$

Similarly, for $E[\min |M_v|_\triangle(n)]$ we have

$$\begin{aligned} E[\min |M_v|_\triangle(n)] &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^{i+1} m(1, \frac{n}{4^i}) + \sum_{i=0}^{d-1} 4^{i+1} m(2, \frac{n}{4^i})), \\ &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^{i+1} \Theta(\sqrt{\frac{n}{4^i}}) + \sum_{i=0}^{d-1} 4^{i+1} \Theta(\sqrt{\frac{n}{4^i}})), \\ &= \frac{2}{n}(\sum_{i=0}^{d-1} 4^{i+1} b_1 \cdot \sqrt{\frac{n}{4^i}} + \sum_{i=0}^{d-1} 4^{i+1} b_2 \cdot \sqrt{\frac{n}{4^i}}), \\ &= 4(b_1 + b_2)\frac{2^{d+\frac{3}{2}}}{n} \sum_{i=0}^{d-1} 4^{\frac{i}{2}}, \\ &= \frac{4(b_1+b_2)}{n} \underbrace{2^{d+\frac{3}{2}}(2^d - 1)}_{(\star)}. \end{aligned} \tag{83}$$

Finally, we compute the order of the term $(\star)$ by replacing d

$$2^{d+\frac{3}{2}}(2^d - 1) = \sqrt{2}n - 2\sqrt{n} \quad \in \Theta(n), \tag{84}$$

yielding

$$E[\min |M_v|_\triangle(n)] \in \frac{\Theta(n)}{n} \in \Theta(c). \tag{85}$$

which proves that $\min |M_v|_\triangle(n)$ has constant size in expectation. $\surd$

A.8 Proof of Proposition 3

For M_{v_1} to be pruned before M_{v_0}, we need to have

$$\Delta D(M_{v_0})\Delta C(M_{v_1}) > \Delta D(M_{v_1})\Delta C(M_{v_0}). \tag{86}$$

Since C is monotonically increasing, we can write

$$\Delta C(M_{v_0}) = \delta \Delta C(M_{v_1}), \tag{87}$$

with $\delta > 1$. Then, replacing (87) in (86) yields

$$\Delta D(M_{v_0}) > \delta \Delta D(M_{v_1}), \tag{88}$$

which proves the proposition. $\surd$

A.9 Proof of Proposition 4

Assume that $\nexists a \in A_{M_{v^\star}}$ such that M_a is nonmonotonic (in the sense of Proposition 3) and $v \in M_a \cap M_{v^\star}$ such that $\Delta D(M_a) < \Delta C(M_v)$, then we have to show that $\lambda(v^\star)$ is a lower bound for $\{\lambda(v)\}_{v\in M}$. We only have to consider the updated vertices (i.e. $a \in A_{M_{v^\star}}$). Moreover, $v^\star$ satisfies Proposition 3, then we have for a:

$$\begin{aligned} \lambda(a) &= \frac{\Delta D(M_a) - \Delta D(M_{v^\star})}{\Delta C(M_a) - \Delta C(M_{v^\star})} > \frac{\lambda \Delta D(M_{v^\star}) - \Delta D(M_{v^\star})}{\lambda \Delta C(M_{v^\star}) - \Delta C(M_{v^\star})}, \\ &> \frac{(\lambda - 1)\Delta D(M_{v^\star})}{(\lambda - 1)\Delta C(M_{v^\star})} > \frac{\Delta D(M_{v^\star})}{\Delta C(M_{v^\star})} > \lambda(v^\star), \end{aligned} \tag{89}$$

which ends the proof. $\surd$