

Computational analysis of 4-8 meshes with application to surface simplification using global error

Laurent Balmelli ^{a,1} Thomas Liebling ^b Martin Vetterli ^{c,d}

^a*Visual and Geometric Computing, IBM Research, T.J Watson Center, USA*

^b*EPFL, Mathematics Dept., Ecole Polytechnique Fédérale, Lausanne, Switzerland*

^c*EPFL, Communication Systems Dept., Ecole Polytechnique Fédérale, Lausanne, Switzerland*

^d*Dept. of EECS, UC Berkeley, Berkeley, USA*

Abstract

Mesheres with subdivision connectivity are popular in applications such as visualization and finite element analysis. We consider a particular class of such meshes known as *4-8 meshes*. We present computational results when computing approximations of 4-8 meshes using vertex decimation or vertex insertion. We first use our results to show that algorithms using local error need $\Theta(n \log n)$ operations to fully decompose or refine a 4-8 mesh. Then, we show that a full decomposition of a 4-8 mesh using global error is obtained with an $\Theta(n \log n)$ decimation algorithm. The solution involves *merging domain intersections* to obtain an efficient mechanism to compute global error estimates for the vertices. In comparison, a direct algorithm using the same error criterion requires $\Theta(n^2)$ operations for the same task. Our results apply to 4-8 meshes connecting vertex sets or approximating subdivision surfaces. For example, our algorithm using global error produces progressive representations of terrain data or subdivision surfaces in \mathbb{R}^3 having arbitrary topology.

Key words: Subdivision connectivity; 4-8 meshes; Multiresolution hierarchies; Surface simplification algorithm using global error; Quadtree triangulation

¹ Contact email: balmelli@us.ibm.com. This work was completed while L. Balmelli was with the Laboratory for Audio-Visual Communications, Communication Systems Dept., Ecole Polytechnique Fédérale, Lausanne, Switzerland.

1 Introduction

1.1 Motivation

This paper presents several computational results when processing a class of meshes with *subdivision connectivity* known as 4-8 meshes. Meshes with subdivision connectivity are regular triangulations constructed using iterated subdivision rules. We look at algorithms using vertex insertion or vertex decimation approaches in order to generate progressive representations for these meshes.

Meshes with subdivision connectivity are popular in many applications, such as visualization [7] and finite element analysis [6], to name a few. Their irregular counterparts have also been extensively studied [11], but regular meshes are preferred because of their superior performance and flexibility for processing [9], transmission [12] and compression [10].

For example, 4-8 meshes have been extensively used to visualize terrain data [2,7,15,17]. In this context, these meshes are also called *quadtree triangulations* because quadtrees are often used to store them [14,17,18]. Terrain models are given as amplitude matrices (i.e. the parametrization is implicit) and 4-8 meshes are used to *connect* the vertices (Figures 2a-e). Recently, researchers have also used 4-8 meshes to compute approximations of subdivision surfaces [19]. Subdivision surfaces are an increasingly popular representation for piecewise-smooth surfaces. Algorithms for subdivisions surfaces use recursive subdivision rules to *create* vertices from a coarse control mesh. Examples of such rules are provided by Loop [16], Catmull-Clark [3,5] and Velho [19]. The properties of subdivision surfaces are an important area of current investigation (e.g. [20]).

In both terrain visualization and subdivision surfaces, researchers often deal with large datasets. Therefore, *simplification algorithms* producing adaptive, multiresolution representations are an important topic of investigation [7,15,17,2]. Multiresolution representations of meshes with subdivision connectivity have many advantages over their uniform counterparts: They allow for vertices to be concentrated in detailed regions, leading to efficient descriptions of shapes. Also, their multiple levels of resolution provide an efficient means to deal with resources-constrained rendering, storage or transmission. However, adaptivity comes at a price: These representations are more complex to process and to store. Also, basic operations such as vertex queries are more difficult to implement. In the next section, we review previous work on simplification algorithms, in particular for 4-8 meshes.

This paper is organized as follows: We review previous work on simplification algorithms for 4-8 meshes in Section 1.2. Then, we give our contributions in Section 1.3. Section 2 introduces 4-8 meshes and the constraints attached to their processing. In Section 3, we analyze simplification operations. Section 4 explains how to

compute *merging domain intersections* and Section 5 presents an application, i.e. surface simplification using global error. We conclude this paper in Section 6.

1.2 Previous work on simplification algorithms

Simplification algorithms, leading to multiresolution representations, aim to select a subset of the original vertices in order to efficiently represent the shape. We consider approaches either based on vertex decimation (e.g. when starting from a dense mesh) or vertex insertion (e.g. when starting from a coarse mesh). Note that alternative approaches exist (e.g. edge split, edge collapse) and are also investigated. In Section 2.2, we explain that a hierarchy (very similar to the hierarchy in a tree structure) is imposed over the vertices by the 4-8 connection (Figures 2a-e). Hence, decimating/inserting an arbitrary vertex often implies jointly decimating/inserting additional vertices to preserve the hierarchy (think of pruning a tree node and all its descendants, or inserting a leaf node and all its parents). For any vertex, its set of descendants is called its *merging domain* and its set of parents is called its *splitting domain*.

Many simplification algorithms for 4-8 meshes based on decimation or insertion have been given in the context of terrain visualization [7,15,17]. However, these methods are based on insertion only [15,17] or on restricted cases of decimation [7] (see below). For subdivision surfaces, most implementations are based on non-adaptive representations to avoid the added complexity and performance penalty traditionally associated with adaptive schemes. When simplifying a mesh, an error criterion is used to select vertices to insert or decimate. For example, an error can be computed at each vertex according to local variations in curvature over its merging domain or splitting domain, depending on if the aim is to decimate or insert the vertex, respectively. Therefore, each simplification step modifies the model's shape, and some errors must be recomputed. In previous works, algorithms were given in order to recompute errors after a vertex insertion [15,17] or restricted decimation [7]. However, no such algorithm is described in the general case of decimation.² Also, all previous algorithms have used local error metrics.

Overall, no general computational analysis of common 4-8 mesh operations (e.g. decimation, insertion, update of the modified errors) is available to the best of the authors' knowledge. A detailed analysis of 4-8 mesh properties is useful in many aspects: It provides tools to design algorithms and forecast their cost. It also allows for more elaborated error metrics to be built, improving algorithm performances. As explained above, simplification algorithms reduce the number of vertices whereas algorithms for subdivision surfaces generate vertices using subdivision rules. Hence an analysis of 4-8 mesh properties helps provide a unifying framework for simplification and subdivision methods.

² We explain the difference between restricted and general decimation in Section 2.2.

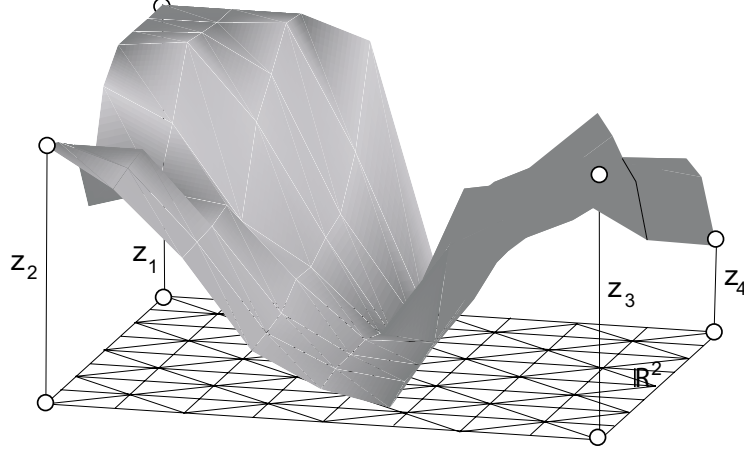


Figure 1. Analysis of the properties of 4-8 meshes: The figure depicts a rendered surface in \mathbb{R}^3 and its underlying triangulation is projected in the plane \mathbb{R}^2 . The amplitudes z of the surface are depicted only for the corner vertices for the sake of clarity. The properties analyzed in this paper derive from the vertex hierarchy and the particular connectivity of 4-8 meshes. Hence, an analysis of the mesh as a tiling of \mathbb{R}^2 is sufficient.

The properties of 4-8 meshes analyzed in this paper derive from the vertex hierarchy and the particular connectivity of the vertex set. Hence, an analysis of the mesh as a tiling of \mathbb{R}^2 is sufficient. In Section 2.1, we present a construction connecting a matrix of amplitudes z . Figure 1 depicts a rendered surface constructed with such a matrix and its underlying triangulation is projected in \mathbb{R}^2 . The tiling is obtained with the procedure shown in Figures 2a-e. Our results also apply to meshes approximating subdivision surfaces in \mathbb{R}^3 (as in [19]), except at *extraordinary vertices* forming the coarse control mesh. These vertices are similar to the four vertices used to connect the initial square of two triangles in Figure 2a. Note that in Figure 1, an amplitude z_i is attached to these extraordinary vertices. Each face of the control mesh is then subdivided as shown in Figures 2a-e. Since the control mesh can be irregular, meshes with arbitrary topology are obtained. In the next section, we explain our contributions and the organization of this paper.

1.3 Contributions

In this paper, we present computational complexity results for processing 4-8 meshes. We summarize our contributions below and refer to the corresponding sections.

Computational analysis of mesh operations. We compute the number of operations required to decimate and insert an arbitrary vertex in the hierarchy and show that, on average, these operations can be performed in $\Theta(\log n)$ time (Section 3.1). Call *ancestors* the vertices whose domain connectivity is modified after a decimation or an insertion. We explain how to find these vertices and show that $\Theta(\log n)$

exist in each case (Section 3.2).

Merging domain intersections. We explain that an interesting problem is to determine *which* are the removed vertices in the merging domain of ancestors after decimating a vertex. This problem requires the computation of *merging domain intersections*, and its solution is the most important contribution of this paper. The solution to this problem enables the building of a global error metric for algorithms using general decimation. We explain how to describe merging domain intersections (Section 4.1), and we provide a model for the problem (Section 4.2). We compute its cost in closed form (Section 4.3) and show that, on average, $\Theta(\log^2 n)$ operations are sufficient to compute the intersections between the merging domain of a decimated vertex and the merging domains of all its ancestors (Section 4.4).

Computational lower bounds for algorithms using local error Consider simplification algorithms based on local error using either general decimation or insertion. We use our results to prove computational lower bounds for their execution (Section 5.1). More precisely, we show that $\Theta(n \log n)$ operations are necessary to fully decompose or refine a surface.

Decimation algorithm using global error. We propose an algorithm to produce adaptive representations of 4-8 meshes using general decimation and global error (Section 5.2). Global error metrics yield better approximation quality than heuristics based on local error, but are often computationally expensive. Also, decimation approaches yield better results than their refinement counterparts. In particular, general decimation is usually needed to obtain optimal solutions [8]. Using merging domain intersections, we obtain a $\Theta(n \log n)$ decimation algorithm that uses global error (Section 6). In comparison, we show that a direct approach using the same error criterion requires $\Theta(n^2)$ operations. Interestingly, our results show that the algorithm using global error has the same computational complexity as its counterpart using local error.

We conclude this paper with a summary of our results in Section 6.

2 Background

2.1 4-8 mesh construction

We present a simple construction of a 4-8 mesh connecting an amplitude matrix z (e.g. terrain data), i.e. the coordinates x, y are implicit. For the sake of clarity, we represent our meshes as tilings of the plane \mathbb{R}^2 . A 4-8 mesh connecting the dataset is created using the recursive procedure depicted in Figures 2a-e. Initially, a quadrilateral, or more simply quad, formed with two triangles is connected using

the four corner vertices. Then, each triangle's hypotenuse is bisected to connect a vertex at the midpoint. We denote each connection step by l and Figures 2b-e depict steps $l = 1, 2, 3, 4$, respectively. After $l = 2d$ connection steps, the mesh contains $n = 2 \cdot 4^d$ triangles. The unique vertex inserted at step $l = 1$ (Figure 2b) is called the *root vertex* and is denoted by v_0 .

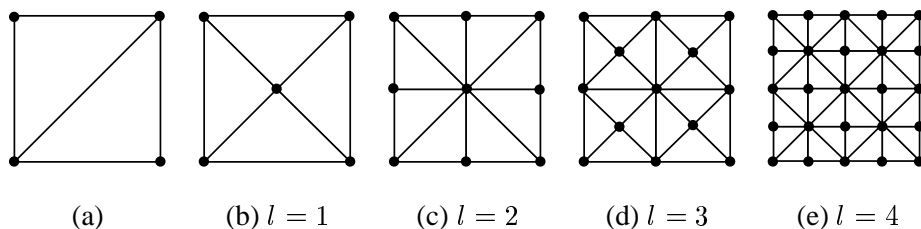


Figure 2. Connection of a matrix of amplitudes z using the 4-8 scheme: (a) Initially, a square formed by two triangles is created using the corner vertices. Then, triangle hypotenuses are bisected to connect a vertex at the midpoint. Each connection step is denoted by l and (b),(c),(d) and (e) show steps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

The 4-8 scheme takes its name from an instance of regular tilings studied by Laves in Cristallography [13]. A 4-8 mesh corresponds to a $[4 \cdot 8^2]$ tiling. The notation suggests that each triangle has one vertex of valence 4 and two vertices of valence 8. In Figures 2a-e, this is verified at even steps l .

Subdivision surfaces are used to generate 4-8 meshes with arbitrary topology [19]. A coarse *control mesh* composed of a small set of triangulated quads (as in Figure 2a) fixes the topology and is used as an initial mesh. Then, subdivision rules are used to create new vertices connected on each quad, as in Figures 2b-e.

2.2 Constraints when simplifying 4-8 meshes

The iterative procedure used to connect the vertices imposes *hierarchical constraints* over the set of vertices. The hierarchical structure is fixed by the connection step l assigned to the vertices. When computing an adaptive representation of the mesh (e.g using decimation or refinement), the hierarchy between vertices must be preserved.

For example, consider the root vertex connected in Figure 2b. A decimation *perserving the hierarchy* operates as follows: When this vertex is decimated (e.g. in the mesh of Figure 2e), the edge originally split by the vertex (the diagonal in Figure 2a) must be recovered. Consequently, all the vertices in the mesh are also decimated. Call v a vertex, then M_v denotes the set of vertices that must be removed jointly in order to recover the original edge and thus preserve the hierarchy. We call the set M_v *merging domain*. A merging domain is attached to each vertex in the mesh. For the vertices v connected at the step depicted in Figure 2e, $M_v = \{v\}$ since it suffices to remove v to recover the corresponding edge in Figure 2d. We

refer to such decimation as a *restricted* case of decimation (as used in [7]). In contrast, a general case of decimation refers to the removal of an arbitrary vertex (i.e. with $|M_v| > 1$).

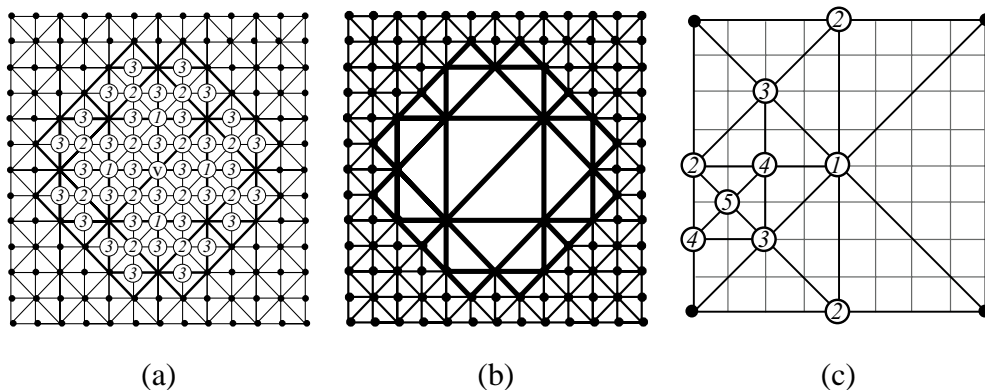


Figure 3. Mesh operations: (a) The white vertices represent the set M_v . (b) Support of the merging domain (set of triangles tiling the domain after M_v is decimated). (c) Insertion of a vertex v at a location connected at step $l = 5$. The white vertices form the splitting domain and the numbers correspond to the connection steps. The underlying grid depicts the parametrization of a 9×9 matrix of amplitudes.

We look now at the hierarchical constraints incurred when decimating and inserting a vertex in more detail. Consider the example in Figure 3a, and assume that v is connected at step l . Then the vertices in M_v connected at steps $l + 1$, $l + 2$ and $l + 3$ are labeled with the index 1, 2 and 3, respectively. The set M_v is decimated as follows: First, v is decimated. Then we remove the four vertices on the edges of the triangulated quad split by v (vertices with index 1). These vertices are called *descendants* of v , and we explain how to find them in Section 3.2. Then for each of these vertices, we decimate their four descendants (vertices with index 2). Note that the descendants split the four quads at the next level surrounding each edge of the previous quad. We repeat the procedure until the descendants connected at the last step are reached (step $l + 3$ in the example). The resulting mesh is shown in Figure 3b. The set of triangles tiling the merging domain in the figure is called *support*.

We also attach to each vertex v a *splitting domain*, denoted by S_v . The domain contains the vertices to insert jointly to v in order to preserve the hierarchy. Figure 3c shows an example of insertion: The vertex v lies at a location connected at step $l = 5$. An insertion preserving the hierarchy operates as follows: First, we connect v to its *parents*³ at step $l = 4$. Then, each of these vertices is connected to its parents at step $l = 3$ and so on. We repeat until no more vertices need to be inserted. At most this bottom-up traversal is stopped at the root vertex. The underlying grid in the figure represents the parametrization of a 9×9 matrix of amplitudes.

Preserving the hierarchy has two consequences for simplified meshes: First, it ensures that successive decimations/insertions yield a set of embedded meshes. For

³ We explain how to find the parents in Section 3.2.

example, consider a coarse mesh obtained after *a series of decimations*. If the series of meshes is embedded, it is always possible to reconstruct a mesh from a coarser version *only* by splitting a set of triangles. Second, the resulting mesh is *conforming* [6], i.e. no triangle has a vertex of another triangle in the interior of one of its edges. In other words, the mesh is a triangulation. This condition must be fulfilled in order to render the surface without cracks, i.e. to avoid shape discontinuities.

3 Analysis of simplification operations

3.1 Decimation and insertion of a vertex

We first evaluate the cost of decimating a vertex. To do so, we compute the number of triangles connected with at least one vertex in M_v . This number is linearly proportional to the number of vertices in M_v . We address two cases: (1) before the decimation of M_v (e.g. Figure 3a), and (2) after the decimation of all vertices in M_v . In the first case, The number of triangles is denoted by $|M_v|_\Delta$. In the second case, the empty set M_v is denoted by \check{M}_v and the number of triangles is denoted by $|\check{M}_v|_\Delta$. Note that $|\check{M}_v|_\Delta$ counts the number of triangles tiling the support of the domain (e.g. Figure 3b).

Both $|M_v|_\Delta$ and $|\check{M}_v|_\Delta$ are functions of the size of the mesh n and the connection step l of the vertex. Then, Proposition 1 gives the sizes $|M_v|_\Delta(l, n)$ and $|\check{M}_v|_\Delta(l, n)$, i.e. as functions of n and l . These functions return the sizes of a “fully expanded” merging domain, i.e the mesh boundaries are ignored. Hence the largest overestimate is obtained for the root vertex, e.g. $|M_v|_\Delta(1, n) > n$. The sizes are correct for vertices close to the center of the mesh and having a sufficiently large l . The proof is given in Appendix A.

Proposition 1 (Size of the merging domain)

Consider a uniform 4-8 mesh containing $n = 2 \cdot 4^d$ triangles with $d > 0$. The number of triangles $|M_v|_\Delta(l, n)$ for a vertex v connected at step $1 \leq l \leq 2d$ is given by

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

where $c_1(l, n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$ and $c_2(l, n) = \frac{2^{-l} n}{16}$. The number of triangles

$|\check{M}_v|_\Delta(l, n)$ is given by

$$|\check{M}_v|_\Delta(l, n) = \begin{cases} 2^{1-l}n - 2, & l > 2d - 4, \\ 32(2^{-(l+4)}n \cdot c_1^{-1} + \frac{3}{2} \cdot c_1) & \\ -16(\log_4 n - \frac{l}{2}) - 18, & l \leq 2d - 4, \end{cases} \quad (2)$$

with $c_1(l, n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$.

For n arbitrary, the maximum of (1) and (2) is obtained for $l = 1$. We use this value to obtain their asymptotical behavior with respect to n : For (1) we have that $c_1(1, n) \in \Theta(\sqrt{n})$ and $c_2(1, n) \in \Theta(n)$, hence

$$|M_v|_\Delta(1, n) \in \Theta(n), \quad (3)$$

whereas for (2), replacing $1/8\sqrt{n/2} \leq c_1(1, n) \leq 1/4\sqrt{n/2}$ yields

$$(4 + 3\sqrt{2})\sqrt{n} - 16 \log_4 n - 26 \leq |\check{M}_v|_\Delta(1, n) \leq (2^{\frac{7}{2}} + 6\sqrt{2})\sqrt{n} - 16 \log_4 n - 26, \\ |\check{M}_v|_\Delta(1, n) \in \Theta(\sqrt{n}). \quad (4)$$

Since the size of M_v depends on the connection step of v , we compute the average size, denoted by $E(\cdot)$, over all vertices. The functions $|M_v|_\Delta(l, n)$ and $|\check{M}_v|_\Delta(l, n)$ decrease exponentially when the connection step increases. More precisely, each time l is incremented, the size of M_v is roughly divided by 4. Hence⁴,

$$E[|M_v|_\Delta(n)] \in \Theta(\log n), \quad (5)$$

and

$$E[|\check{M}_v|_\Delta(n)] \in \Theta(c). \quad (6)$$

We compute now the cost for inserting a vertex. To do so, we calculate the number of triangles $|S_v|_\Delta$ connected with at least one vertex in S_v . Finding S_v only requires a bottom-up traversal of the mesh structure (exactly how this traversal is performed will be explained in the next section). Moreover, each vertex splits two triangles in the mesh (Figure 3c). Therefore, for the vertices connected at the last step $l = 2d$, we have

$$|S_v|_\Delta \in \Theta(\log n). \quad (7)$$

The function $|S_v|_\Delta$ increases linearly with the connection step, and $|S_v|_\Delta$ is minimum for the root vertex. Therefore, averaging $|S_v|_\Delta$ over all vertices yields again

$$E[|S_v|_\Delta] \in \Theta(\log n), \quad (8)$$

⁴ For technical details see [1].

3.2 Sets of ancestor vertices

Consider a decimation algorithm: We denote by A_{M_v} the set of vertices whose merging domain connectivity is modified after decimating a vertex M_v . Note that in simplification algorithms, where an error is computed at each vertex, the set A_{M_v} contains the vertices whose error is modified after decimating M_v . Similarly, consider an insertion algorithm: We denote by A_{S_v} the set of vertices whose splitting domain connectivity is modified after inserting a vertex v , i.e. inserting S_v . Both vertices in A_{M_v} and A_{S_v} are called *ancestors*. The set A_{M_v} refers to vertices not yet decimated, whereas A_{S_v} refers to vertices not yet inserted.

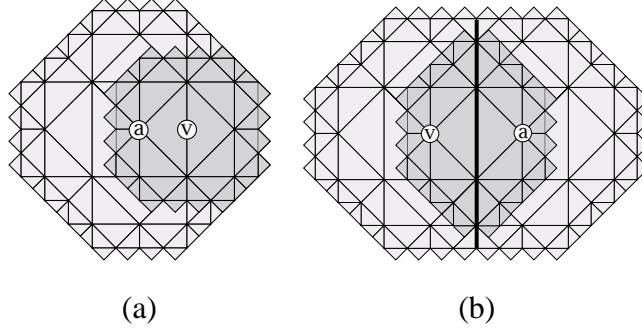


Figure 4. Visual representation of the ancestors of M_v : (a) $M_v \subset M_a$. (b) Overlap between M_v and M_a . The dark region depicts the domains' overlap, and the thick line is the intersection boundary.

We explain first how to find the ancestor sets A_{M_v} : We are looking for vertices a such that $M_v \subset M_a$ (Figure 4a), and for vertices a whose domain M_a partially overlaps M_v (Figure 4b). In the latter figures, we depict the merging domains using their support. The decimation of M_v has removed vertices in the merging domain of both types of vertices a as defined above.

An important property of 4-8 meshes is obtained by construction: When the mesh is subdivided, the merging domain of a vertex v is embedded in at most two merging domains of vertices (call the vertices a_1 and a_2) connected at the previous step. These vertices are the *parents* of v . Each vertex has two parents at the previous step, except for the border vertices, which have only one parent. Figures 5a-d depict four connection steps. The root vertex (Figure 5a) has no parents by definition. For steps $l > 1$ (Figures 5b-d), an arrow links each vertex to its parents (at the previous step l). Symmetrically reversing the arrows would link a vertex to its *descendants*. To find a chain of ancestors, denoted by A_v , for any vertex v , the arrows linking v to its parents are recursively followed until the root vertex is reached. This results in a bottom-up traversal of the mesh. For example, in Figure 6a we depict the chain of ancestors $A_v = \{a_i\}$, $i = 1 : 10$ of vertex v . In the example, a_{10} is the root vertex v_0 . The ancestor with the smallest connection step is always the root vertex

v_0 . Hence for any vertex v

$$\forall a_i \in A_v, M_v \subset M_{a_i} \subset M_{v_0} \quad (9)$$

Moreover, $\forall i, a_i \notin M_v$. Note that the set S_v (e.g. Figure 3c) is found using the ancestor chain of v . However, the set S_v is composed of vertices not yet inserted, hence recursions (e.g. as shown in Figure 6a) are stopped when a vertex already in the mesh is met.

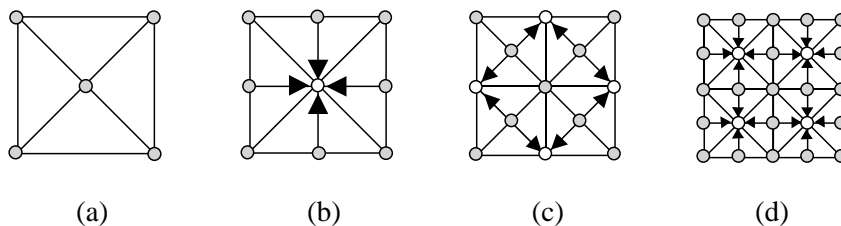


Figure 5. Finding the parent vertices: In each figure, the parents are represented in white and an arrow points from each vertex to its parents. (a) The root vertex has no parents by definition. Parents of the vertices inserted at steps (b) $l = 1$, (c) $l = 2$ and (d) $l = 3$.

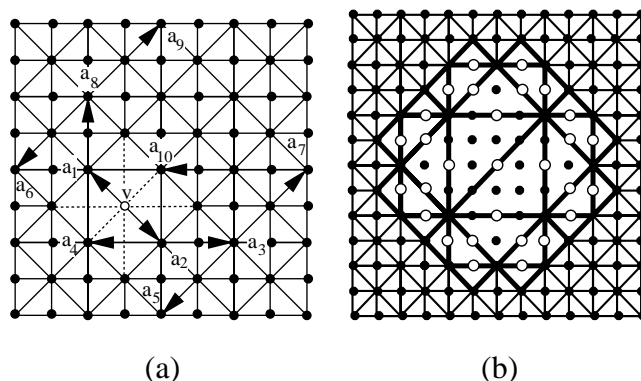


Figure 6. Ancestor vertices: (a) The chain of ancestors a_i built from v by recursively finding its parents towards the root vertex. Note that $a_{10} = v_0$. (b) The white vertices in M_v are the only ones with one parent not in M_v .

Following the above discussion, the ancestors a of M_v such that $M_a \subset M_v$ are simply found by building a chain of ancestor starting at v . How can we find ancestors when M_v and M_a partially overlap? These ancestors are found by building ancestor chains from a selected set of vertices in M_v . Again, denote by a_1, a_2 the parents of a vertex (found by following the arrows in Figures 5a-d). For some vertices in M_v , $a_1, a_2 \in M_v$. Therefore, these vertices must be avoided. Only a small set of vertices in M_v have one parent which does not belong to M_v . These vertices are depicted in white in Figure 6b. There is exactly one such vertex per triangle tiling the support of the merging domain. Therefore, with (6) we know that, on average, we have a constant number of such vertices. Then, the ancestors a are found by building an ancestor chain from these particular vertices, starting at the parent not belonging to M_v .

A bottom-up traversal of the mesh to find ancestors requires $\Theta(\log n)$ steps. Hence, with (6) we have, on average, that

$$|A_{M_v}| \in \Theta(c) \cdot \Theta(\log n) \in \Theta(\log n). \quad (10)$$

We explain now how to find A_{S_v} : A property of the vertices in M_v is

$$\forall w \in M_v, v \in A_w, \quad (11)$$

i.e. all the vertices $w \in M_v$ have v as an ancestor. For A_{S_v} we are looking for the vertices a whose splitting domain connectivity has changed after inserting v , i.e. at least one vertex was inserted in S_a . More precisely, the vertices such that $\exists w \in S_v, w \in S_a$. Therefore, we have to find a subset of vertices w_i in A_v (the ancestor chain built from v) with the smallest connection step, such that no pairs w_i, w_{i+1} verifies $M_{w_i} \subset M_{w_{i+1}}$. Otherwise, the set $\{w_i\}$ is redundant. Call these vertices $\min_l(A_v)$, then following (11), we have

$$A_{S_v} = \bigcup_{w \in \min_l(A_v)} M_w. \quad (12)$$

Using (5), we can conclude that, on average,

$$|A_{S_v}| \in \Theta(\log n). \quad (13)$$

4 Merging domain intersections

In the following section, we propose a method for finding merging domain intersections. Recall that two types of ancestors exist for M_v : the vertices a such that $M_v \subset M_a$ and the vertices a whose domain M_a partially overlaps M_v . When $M_v \subset M_a$, then $M_v \cap M_a = M_v$. Therefore, we are interested in finding the intersection in the second case. We proceed in two steps: First, we compute the size of an intersection. The metric used to compute the size is defined in the next subsection. Second, we provide an algorithm that can be used to find $M_v \cap M_a$ for all ancestors a whose domain partially overlaps M_v .

4.1 How to describe an intersection

We describe the intersection between two merging domains as *the union of a set of (smaller) merging domains*. Using merging domains as building blocks provides a compact, efficient description for intersections. Finding the vertices in M_v only requires searching around v using a single pattern, whereas finding $M_v \cap M_a$ is difficult due to the multiplicity of cases: Just consider all the possible locations for

neighboring ancestors a . Hence, Figure 4b is just a particular example of arrangement for $M_v \cap M_a$. Following (10), in total we have $\Theta(\log n)$ such arrangements.

Consider the following example: In Figure 7b, the intersection $M_v \cap M_a$ is the single domain M_w . In general, more than one domain is needed to represent the intersection. Consider then $M_v \cap M_a$ in Figure 7c: In this case, the intersection is

$$M_{w_1} \cup M_{w_2} \cup M_{w_3}, \quad (14)$$

which makes sense, since the vertices contained in the intersection belong to the domains M_{w_i} , $i = 1, 2, 3$.

We would like to represent the intersection as an *exclusive* set of vertices, i.e. as in (14). Consider $M_v \cap M_a$ in Figure 7d: The domain M_{w_3} overlaps with the domains M_{w_2} and M_{w_4} . We write the intersection as

$$\bigcup_{i=1}^5 M_{w_i} = \bigoplus_{i=1}^5 M_{w_i} \setminus D, \quad (15)$$

where the operator \bigoplus “gathers” the vertices in the sets M_{w_i} , and D denotes the set of vertices to remove in order to obtain an exclusive set – in this case, the vertices in $M_{w_3} \cap M_{w_2}$ and $M_{w_3} \cap M_{w_4}$. To minimize the number of terms in the union (15), the domains M_{w_i} should be as large as possible (e.g. as depicted in Figure 7d). Finally, the set D in (15) is also expressed as a union of smaller domains. Therefore, computing this term again involves removing redundant vertices. This suggests that finding an intersection often requires recursively adding (\bigoplus) and subtracting (\setminus) domains (inclusion-exclusion principle).

We address the problem as follows: We identify a worst case, i.e. the pair of neighbor vertices v and a with the largest intersection. Then, we propose a model to compute the size of the intersection (Section 4.2 and 4.3). The size is given in terms of domains to add or subtract, e.g. as in (15), in order to obtain an exclusive set of vertices. Finally, we provide an algorithm for computing all possible intersections of a merging domain M_v with its neighbors a in A_{M_v} (Section 4.4).

4.2 Modeling the intersection between a pair of merging domains

Consider two vertices v and a arranged as in Figure 7a. The intersection size is maximum between domains of central vertices in two horizontal (or vertical) adjacent quads (Figure 7a). Figures 7b to 7d depict the union between two domains⁵ attached to vertices connected respectively at step $2d - 1$, $2d - 3$ and $2d - 5$ for a mesh of size $n = 2 \cdot 4^d$ (recall that the subdivision steps l range between 1 and

⁵ In the figures, we choose to depict M_a as triangulated to explicitly show the density of triangles needed for the intersection.

2d). These vertices are located at the center of a quad. The intersection between the domains is shaded. We denote by I_{2d-1} , I_{2d-3} and I_{2d-5} these unions, hence

$$\begin{aligned} I_{2d-1} &= (M_v \oplus M_a) \setminus M_{w_1}, \text{ with } v \text{ and } a \text{ as in Figure 7b,} \\ I_{2d-3} &= (M_v \oplus M_a) \setminus (\oplus_{i=1}^3 M_{w_i}), \text{ with } v \text{ and } a \text{ as in Figure 7c.} \end{aligned} \quad (16)$$

Assume that $C(\cdot)$ is an operator measuring the cost to find I_{2d-j} , $j \geq 1$, as defined in the previous section. Then, we have that $C(I_{2d-1}) = 2$ and $C(I_{2d-3}) = 4$. To verify this, simply count the number of times an operator \oplus or \setminus is used in the above equations.

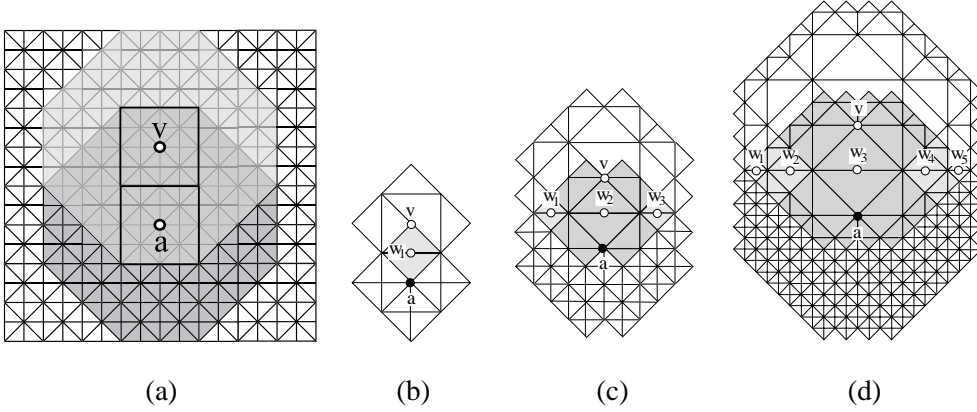


Figure 7. The intersection between merging domains in vertical position: (a) The intersection is maximum for direct vertical (as depicted) and horizontal neighbors. The shaded parts in (b) I_{2d-1} , (c) I_{2d-3} and (d) I_{2d-5} depict the intersections between the domains M_v and M_a .

Finding I_{2d-5} requires a little more work. Call *basic domains* the domains forming an intersection in I_{2d-j} . For example, I_{2d-1} , I_{2d-3} and I_{2d-5} have one, three and five basic domains, respectively. Then, Figure 8a depicts the intersection in I_{2d-5} and a decomposition into a set of *basic domains* M_{w_i} , $i = 1 : 5$ is shown in Figure 8b. Unlike I_{2d-1} and I_{2d-3} , some of the basic domains intersect and the left part in Figure 8b shows that $M_{w_2} \cap M_{w_3}$ is an instance of I_{2d-1} . Symmetrically, the same observation can be made for $M_{w_3} \cap M_{w_4}$. Therefore, to find I_{2d-5} , we must first deal with the embedded I_{2d-1} 's. Hence, I_{2d-5} can be written as

$$I_{2d-5} = \underbrace{(M_v \oplus M_a)}_2 \setminus \underbrace{(\oplus_{i=1}^5 M_{w_i})}_4 \setminus \underbrace{\overbrace{I_{2d-1}}^{\text{left}} \overbrace{I_{2d-1}}^{\text{right}}}_{2(C(I_{2d-1})+1)}, \quad (17)$$

where “left” and “right” above the equation stand for the left and right I_{2d-1} 's in Figure 8b. The costs of the individual part of (17) are given below the equation. How is computed the cost $C(I_{2d-5})$ in this case? First, we account for the cost of each I_{2d-1} and the cost to subtract them from $\oplus_{i=1}^5 M_{w_i}$. Then, we add the cost for adding the five basic domains forming the intersection and the cost for subtracting

them to $M_v \oplus M_a$. We called this latter part of the cost *basic cost* because it does not account for embedded intersections. Hence, the total cost to compute (17) is

$$C(I_{2d-5}) = \underbrace{6}_{\text{basic cost}} + 2 \cdot (C(I_{2d-1}) + 1) = 6 + 2 \cdot (2 + 1) = 12. \quad (18)$$

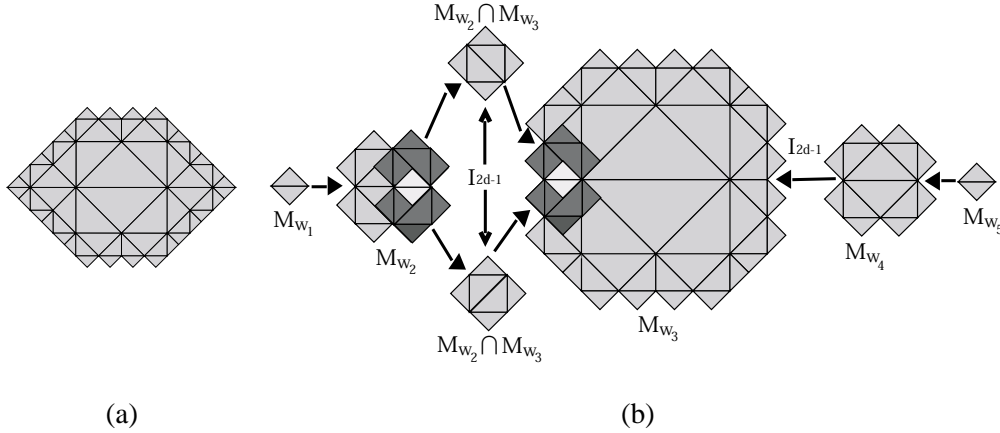


Figure 8. Decomposition of the intersection in I_{2d-5} : (a) The intersection in I_{2d-5} . (b) Decomposition of the intersection into a set of *basic domains* M_{w_i} , $i = 1 : 5$. The embedded intersection $M_{v_2} \cap M_{v_3}$ is split further. This intersection is an instance of I_{2d-1} (Figure 7b).

More generally, finding I_{2d-j} 's with $j \geq 5$ always involves dealing with smaller embedded I_{2d-j} 's. The tree in Figure 9 efficiently models the problem: Each level, as well as each node, represents an instance of I_{2d-j} . For example, I_{2d-5} is represented by the first level in the tree. The two nodes at this level depict the symmetrical embedding of I_{2d-1} 's as represented in Figure 8b. Hence, the tree is recursive: Consider for example I_{2d-9} , which contains two instances of I_{2d-5} . Then, each instance embeds I_{2d-1} 's and is represented by the first level of the tree.

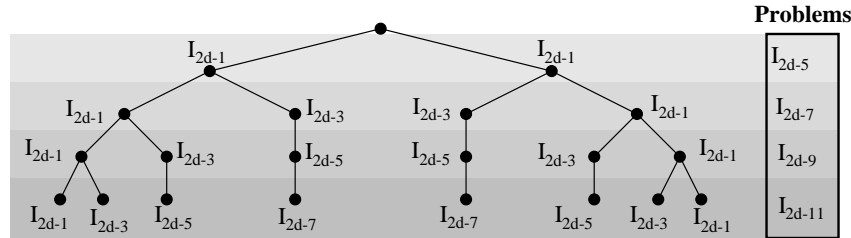


Figure 9. Embedding of intersection problems using a recursive tree: Each level of the tree, as well as each node, represents an instance of embedded intersection in I_{2d-j} , $j \geq 5$.

Only I_{2d-1} and I_{2d-3} do not contain embedded intersections to resolve. We can obtain a nonrecursive formulation of the tree as follows: First, we replace each node representing an instance of I_{2d-j} , $j \geq 5$, by the level representing its embedded instances I_{2d-1} and I_{2d-3} . We call $T_{1,3}$ the resulting tree. The right half of $T_{1,3}$ after substitution is shown in Figure 10a. The left half is a vertically mirrored version.

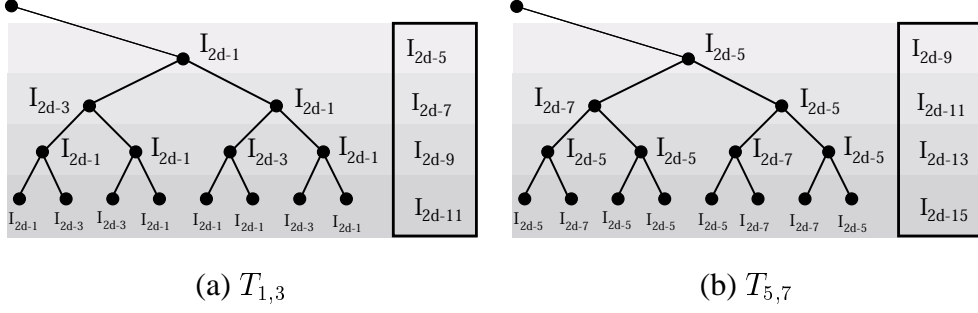


Figure 10. Trees for the nonrecursive model: (a) The tree $T_{1,3}$ in the figure is obtained by replacing the embedded I_{2d-i} 's, $i \geq 5$ in Figure 9 by their associated levels. (b) The tree $T_{5,7}$ in the figure models the occurrences of embedded I_{2d-5} and I_{2d-7} in I_{2d-i} 's, $i \geq 9$.

Then, we need a new tree $T_{5,7}$ to represent pairs of instances I_{2d-5} and I_{2d-7} , embedded in I_{2d-j} , $j \geq 9$ (Figure 10b). Also, we need a tree $T_{9,11}$ to account for pairs I_{2d-9} and I_{2d-11} in I_{2d-j} , $j \geq 13$, etc... More generally, we need a set of trees $T_{2d-j, 2d-j-2}$ to represent instances of pairs I_{2d-j} and I_{2d-j-2} , with $j \geq 2d - j + 2$. Therefore the recursive tree model in Figure 9 is replaced by a set of trees.

We give now an example: How can we find $C(I_{2d-9})$ with our nonrecursive set of trees? Computing $C(I_{2d-9})$ involves two trees: First, we account for the set of embedded instances I_{2d-1} and I_{2d-3} in $T_{1,3}$ (third level in the tree of Figure 10a). Then, we account for the embedded instances of I_{2d-5} in $T_{5,7}$ (first level in the tree of Figure 10b). Finally, the basic domains forming the intersection in I_{2d-9} are taken into account.

4.3 Computational cost

The advantage of the nonrecursive formulation is that the appearance pattern of any pair I_{2d-j} , I_{2d-j-2} is represented by a single generic tree $T_{2d-j, 2d-j-2}$. It suffices then to study this tree in order to evaluate the asymptotical cost for finding I_{2d-j} .

Let us denote by Φ and Ψ the *number of pairs* of problems I_{2d-j} and I_{2d-j-2} , respectively. The repetition pattern is given by the following recurrence equations

$$\begin{aligned}
 \Phi(k) &= \Phi(k-1) + 2\Psi(k-1), \\
 \Psi(k) &= \Phi(k-1), \\
 k &\geq 0, \Phi(0) = 1, \Psi(0) = 0.
 \end{aligned} \tag{19}$$

The case $k = 0$ corresponds to the first level in the tree, where we have a single pair of subproblems I_{2d-j} (Figures 10a-b), therefore $\Phi(0) = 1$ and $\Psi(0) = 0$. The

system (19) has the solution

$$\begin{aligned}\Phi(k) &= \frac{1}{3}(-1)^k + \frac{2}{3}2^k, \\ \Psi(k) &= -\frac{1}{3}(-1)^k + \frac{1}{3}2^k,\end{aligned}\tag{20}$$

where $k \geq 0$.

We compute the cost $C(I_{2d-j})$ as follows: First, we rename each cost $C(I_{2d-j})$ by $C(I_i)$, where $i = \lfloor j/2 \rfloor$. This allows for computing the costs as a single-parameter function and simplifies our computation. Then, we weight the number of pairs $\Phi(k), \Psi(k)$ of problems I_{2d-j} and I_{2d-j-2} with their basic costs, since the non-recursive model lets us use a summation across a set of trees in order to account for embedded intersections. In general, the basic cost for I_{2d-j} is $j + 1$, or equivalently $2i + 2$ in our single-parameter cost function. The first problem involving two trees is $C(I_{2d-7})$, i.e. $C(I_3)$. Therefore the cost for $i > 2$ is

$$C(I_i) = 2i + 2 + 2 \sum_{j=1}^p (4j - 1) \Phi(i - 2j) + 2 \sum_{j=1}^q (4j + 1) \Psi(i - 2j), \tag{21}$$

where $p = i - 1 - \lfloor \frac{i-1}{2} \rfloor$ and $q = \lfloor \frac{i-1}{2} \rfloor$. To obtain the asymptotic behavior, we sum the equation, leading to

$$\begin{aligned}C(I_i) = & 2i + \frac{90}{27}2^i + 1 - \frac{2^i}{4^{p+1}} \left[\frac{64}{9}(p+1) + \frac{16}{27} \right] - \frac{2^i}{4^{q+1}} \left[\frac{32}{9}(q+1) + \frac{56}{27} \right] + \\ & \frac{4}{3}(-1)^i \left[(p+1)^2 + 1 - (q+1)^2 - \frac{3}{2}(p+1) + \frac{q+1}{2} \right].\end{aligned}\tag{22}$$

A quick analysis is performed by observing the magnitude of each term:

$$i, p, q \in \Theta(\log n), \quad 2^i \in \Theta(n), \quad 4^{-p}, \quad 4^{-q} \in \Theta\left(\frac{1}{n}\right).\tag{23}$$

Therefore,

$$C(I_i) \in \Theta(n), \tag{24}$$

since 2^i is the dominant term in (22). As for (1) or (2), the cost $C(I_i)$ decreases exponentially when i increases. Hence averaging (24) over all vertices yields

$$E[C(I_i)] \in \Theta(\log n).\tag{25}$$

4.4 Algorithm computing all intersections

We give now an inclusion-exclusion algorithm to compute in the sets D in (15) between M_v and *all* the ancestors in A_{M_v} .

Algorithm 1. Compute all intersections between M_v and M_a , with $a \in A_{M_v}$

- (1) **for** all vertices w connected at step $2d \dots l$
 - (a) **for** all $a \in A_w$
 - (i) $M_a \leftarrow M_a \setminus w$
 - (ii) **if** $a \notin M_v$ **then** $D_a \leftarrow D_a \oplus w$
 - (b) **end**
- (2) **end**

To illustrate the algorithm with a simple example, we compute the term D in (15). Recall the decomposition in Figure 8b. Then, $D = (M_{w_2} \cap M_{w_3}) \oplus (M_{w_3} \cap M_{w_4})$. We restrict our example to computing the first term of D . The intersection $M_{w_2} \cap M_{w_3}$ is shown in Figure 7b, hence in our example $M_{w_2} \cap M_{w_3} = M_a \cup M_v$. Assume that a and v are connected at step l . Then, w in Figure 7b is connected at level $l + 1$ and $A_w = \{a, v, \dots\}$, where the dots suggest additional vertices. The algorithm iteratively decimates vertices starting at the ones with the largest connection step ($l + 1$ in our example). Each vertex is removed from all the merging domains of its ancestors. Assume that D gathers the removed vertices forming $M_{w_2} \cap M_{w_3}$, then the algorithm proceeds as follows: First, $\forall a \in A_w$, decimate w from M_a , i.e. $M_a \setminus \{w\}$. The same is done for all other vertices connected at step $l + 1$ in M_a and M_v (see the dots in D below). Then after the first step, we have

$$D = \{w, \dots\}, \quad M_a = \{a\}, \quad M_v = \{v\}. \quad (26)$$

At the second step, the vertices connected at step l are considered. Hence, the vertices a and v are decimated and $D = \{w, a, v, \dots\}$. The set D contains only one w and an exclusive set is obtained.

Algorithm 1 computes all intersections between M_v and M_a , with v connected at step l and $a \in A_{M_v}$. As suggested before, our algorithm finds the intersection by decimating M_v , although this is not mandatory to implement the algorithm. At each ancestor $a \in A_{M_v}$, a set D_a gathers the vertices in the intersection between M_v and M_a . Note that the decimation must be performed step-wise and starts at vertices with the largest step l . Since M_v contains $\Theta(\log n)$ vertices on average and $\Theta(\log n)$ operations are required to find the ancestor chain A_v , the cost of the algorithm is $\Theta(\log^2 n)$.

We summarize our results in the following proposition:

Proposition 2

The vertices in the intersections of a merging domain M_v with the domains of its ancestors A_{M_v} can be found in $\Theta(\log^2 n)$ operations.

5 Application

This section is organized as follows: First, we briefly evaluate the computational costs of insertion and decimation algorithms using local error with the results obtained in Section 3. Then, we introduce our decimation algorithm using global error.

5.1 Cost of algorithms using local error

Consider an algorithm using general decimation whose input is a dense 4-8 mesh of vertices in \mathbb{R}^3 . A progressive representation is computed using iterated decimation. An error in l_2 norm is computed for each vertex v as the sum of the squared differences between the vertices in M_v and their projection in the domain's support averaged by $|M_v|_\Delta$. Then, at each step we need $\Theta(\log n)$ operations (5) to decimate the vertices, and $\Theta(\log n)$ operations (10) to find the ancestors. For each ancestor, $\Theta(\log n)$ vertex errors (5) have to be locally recomputed, hence the cost for updating all errors is $\Theta(\log^2 n)$. On average, the algorithm requires $n/\Theta(\log n)$ steps to fully decompose the mesh; therefore, the minimal cost is $\Theta(n \log n)$.

Now consider an algorithm using insertion. The input mesh has minimal resolution (e.g. Figure 2a) and is iteratively refined using vertex insertion. Then, at each step we need $\Theta(\log n)$ operations (7) to insert the vertices, and $\Theta(\log n)$ operations (13) to find the ancestors. Again, for each ancestor, $\Theta(\log n)$ vertex errors (7) have to be locally recomputed, hence the cost for updating all errors is $\Theta(\log^2 n)$. On average, the algorithm requires $n/\Theta(\log n)$ steps on to fully refine the mesh, therefore the minimal cost is again $\Theta(n \log n)$. We conclude with the following Proposition:

Proposition 3

On average, an algorithm based on local error (evaluated over the vertex domains) and using general decimation or insertion requires $\Theta(n \log n)$ operations to fully decompose or refine a 4-8 mesh with n triangles.

5.2 Algorithm based on general decimation and global error

This section introduces a decimation algorithm based on global error and shows that it computationally outperforms a direct approach using the same error criterion. Our algorithm is inspired from an algorithm used to compute adaptive quantizers for compression presented by Chou et al.[4]. Note that in the context of adaptive quantizers, less constraints are incurred since no notion of “conforming” solutions is defined. We apply our algorithm to meshes built on matrices of amplitude z , e.g. terrain data. We use *mesh functionals* $u : M_v \rightarrow \mathbb{R}$ to compute properties for v over its domain M_v . We use two mesh functionals R and D : R is called

the *rate* and counts the number of triangles, whereas D measures the distance in l_2 norm between the original surface and an approximation. Note that $R(M_v)$ can be computed in closed form using (1) and (2) if overestimates are allowed for the vertices at the mesh boundaries. Hence, for each v we compute the vector value $\mathbf{u}(M_v) = (R(M_v), D(M_v))$.

Call M_0 the input mesh and M a simplified version, then the problem to solve is

$$D(R) = \min_{|M| \leq |M_0|} \{D(M) | R(M) \leq r\}, \quad (27)$$

where r denotes a constraint in rate. A progressive representation for M is found by solving the problem for all values $2 \geq r \geq n$. We further define the *variation* of a functional as $\Delta \mathbf{u}(M_v) = \mathbf{u}(M_v) - \mathbf{u}(\tilde{M}_v)$. Hence, $\Delta \mathbf{u}(M_v) = (\Delta R(M_v), \Delta D(M_v))$. The variation $\Delta \mathbf{u}(M_v)$ is the change in rate and distortion when M_v is decimated. Therefore, a vector $\Delta \mathbf{u}(M_v)$ corresponds to a simplified mesh in the space of values spanned by R and D (this plane is usually called *rate-distortion* plane in the compression literature [8]). The ratio $\lambda(v) = -\Delta D(M_v)/\Delta R(M_v)$ is the trade-off between rate and distortion when M_v is decimated and represents a slope in the rate-distortion plane.

The algorithm proceeds as follows: Initially, the variations $\Delta \mathbf{u}(M_v)$ for each vertex are computed. Recall that M_{v_0} contains all the vertices in the mesh. Therefore, $R(M_{v_0})$ and $D(M_{v_0})$ measure the rate and the distortion of the entire mesh, respectively. At each iteration the vertex v with minimal $\lambda(v)$ is chosen. Then M_v is decimated, and the ancestor errors are updated, as described below.

We use Algorithm 1 to decimate M_v and update the functional variations. For each ancestor a , the variation of vector-functional values are replaced by

$$\Delta \mathbf{u}(M_a) - \Delta \mathbf{u}(M_w), w \in M_v \quad (28)$$

where $w \in M_v$. The variations $\Delta \mathbf{u}(M_a)$ updated with $\Delta \mathbf{u}(M_w)$ are found by constructing an ancestor chain A_w . Hence (28) replaces step (i) in Algorithm 1 and step (ii) is no longer necessary. After the initialization step, the variations $\Delta \mathbf{u}(M_v)$ are global, since no vertex is yet decimated. The following example illustrates how this property is maintained after the update: Assume that all vertices in M_v are decimated except v . Therefore, following (11) the updated variation at v and v_0 are

$$\Delta \mathbf{u}(M_v) - \sum_{w \in M_v, w \neq v} \Delta \mathbf{u}(M_w) \quad , \quad \Delta \mathbf{u}(M_{v_0}) - \sum_{w \in M_v, w \neq v} \Delta \mathbf{u}(M_w). \quad (29)$$

Assume that v is now decimated, then using (28), the variation at v_0 is now

$$\Delta \mathbf{u}(M_{v_0}) - \Delta \mathbf{u}(M_v), \quad (30)$$

which shows that the global error is used.

Algorithm 2. Surface simplification**initialization:****for all** v

compute $\Delta D(M_v)$, $\Delta R(M_v)$ and $\lambda(v) = \frac{-\Delta D(M_v)}{\Delta R(M_v)}$.

iteration:

$i = 1$ (counter for the approximations.)

while $R(M_{v_0}) > 2$

search the optimal vertex $v^* = \arg \min_{v \in M} \lambda(v)$.

$M_i \leftarrow M_{i-1} \setminus M_{v^*}$.

update $\Delta D(M_a)$ and $\Delta R(M_a)$, $\forall a \in A_{M_{v^*}}$.

end**end**

5.3 Complexity

Algorithm 2 is used to compute a full decomposition of the mesh. The output is a progressive representation of the input dataset (e.g. terrain). An example of approximation is shown in Figure 11.

The cost of the algorithm is computed as follows: Initially, the values $\Delta \mathbf{u}(M_v)$ and $\lambda(v)$ are stored for each vertex. Additionally, we use a value λ_{\min} at each vertex to store the minimal slope amongs all its descendants. Following (5) and considering that the number of vertices in the mesh is proportional to n , this initialization has cost $\Theta(n \log n)$. At each iteration, the optimal vertex v^* (having minimal slope $\lambda(v^*)$) is found in $\Theta(\log n)$ operations using the values λ_{\min} . The cost to decimate M_v and update the variations for the vertices in A_{M_v} is $\Theta(\log^2 n)$. The values $\lambda(v)$ and λ_{\min} are also recomputed and the algorithm is iterated. On average, $n/\Theta(\log n)$ steps are necessary to decompose the mesh. Hence, the cost to compute the full decomposition is $\Theta(n \log n)$.

Following (11), a direct algorithm needs to recompute the global error over each ancestors' domain. A lower bound for this update is obtained as follows: We have roughly $\Theta(4^{l+1})$ vertices at step l and $l \in \Theta(\log n)$ ancestors exists. Call a any such ancestor, then $|M_a|_{\Delta}(i, n) \approx n/4^{i-1}$, $1 \leq i \leq l$. Therefore, a lower bound for the complexity is

$$\sum_{i=0}^{\log_4 n} 4^i \sum_{j=0}^i \frac{n}{4^j} = \frac{16}{9}n^2 - \frac{1}{3}n \log_4 n - \frac{7}{9}n \in \Theta(n^2). \quad (31)$$

Note the above approximation accounts only for the ancestors a such as $M_v \subset M_a$. Accounting for the update of the ancestors whose domain partially overlaps does not change the order of magnitude. However, this evaluation is complex due to the $\Theta(\log n)$ cases of overlap one has to deal with (Section 4.1). We conclude with the following proposition:

Proposition 4

On average, an algorithm based on global error and using general decimation requires $\Theta(n \log n)$ operations to fully decompose a 4-8 mesh with n triangles when merging domain intersections are used to update the vertex errors.

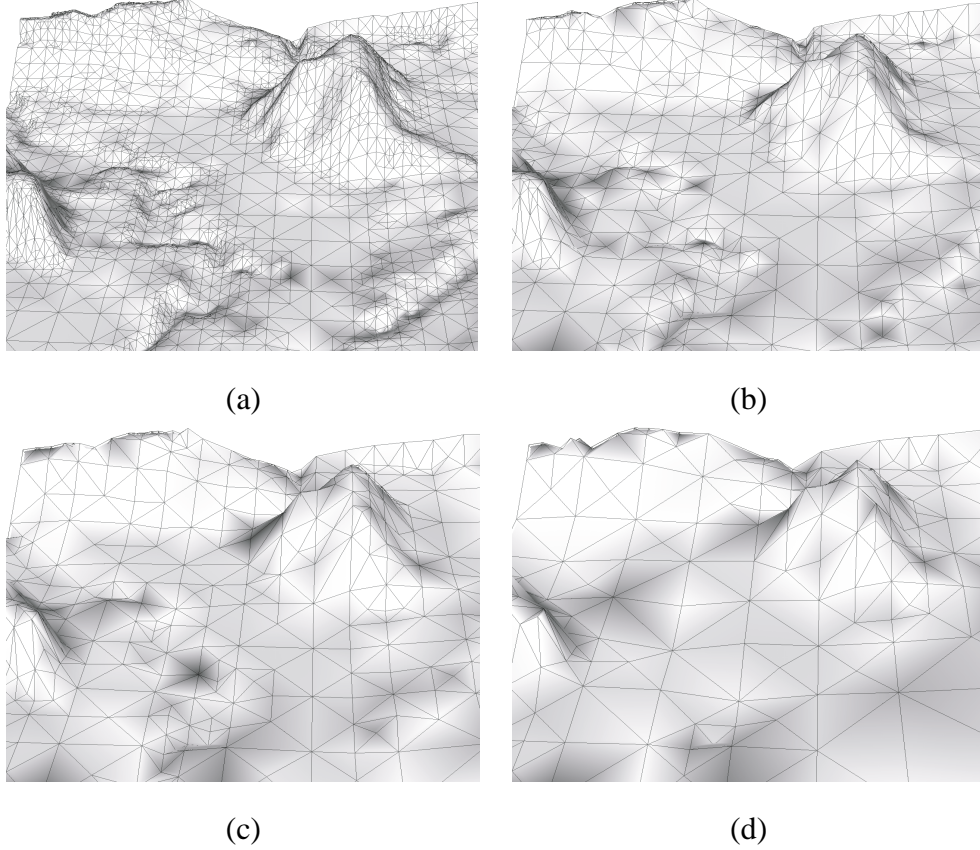


Figure 11. Example of terrain decomposition: Series of models at decreasing rate using approximatively (a) 6400, (b) 1600, (c) 800 and (d) 400 triangles.

6 Conclusion

We presented several results in computational complexity for simplification algorithms processing 4-8 meshes. We have shown that $\Theta(\log n)$ operations are necessary to decimate or insert a vertex in the mesh while preserving the hierarchy over the vertex set. These operations yield a conforming mesh, hence the represented surface can be rendered without shape discontinuity. We have shown how to efficiently update the vertex errors when decimating or inserting vertices. More precisely, the latter operations change the errors at $\Theta(\log n)$ vertices, and on average, $\Theta(\log^2 n)$ operations are needed to update them. Since $n/\Theta(\log n)$ steps are necessary to decompose or refine a mesh of n triangles, the total cost of the algorithm is $\Theta(n \log n)$.

We addressed the problem of finding merging domain intersections and provided a model for obtaining a closed form for the computational cost of this operation. More precisely, we have shown that $\Theta(\log n)$ operations are required to compute an intersection and that all intersections between the merging domain of a vertex and the domain of its ancestors can be found in $\Theta(\log^2 n)$ operations. We used these results to provide an algorithm using general decimation and global error to decompose a mesh in $\Theta(n \log n)$. We explained that a direct algorithm using the same error criterion would need $\Theta(n^2)$ operations to perform the decomposition.

A Proof of Proposition 1

We compute the number of triangles $|M_v|_\Delta$ and $|\check{M}_v|_\Delta$ as follows: We construct a dual representation of the support using a tree structure. Hence, each triangle corresponds to a node and the tree expands towards the boundaries of the support (Figure A.1a).

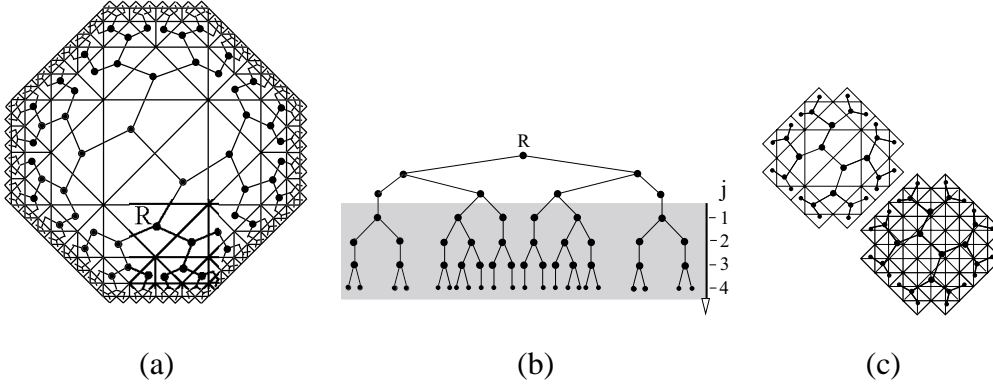


Figure A.1. Computing $|M_v|_\Delta$ and $|\check{M}_v|_\Delta$: (a) A dual representation of the support is constructed using a tree structure. (b) Tree structure expanding towards the boundaries. The node labeled “R” corresponds to the node in part (a). (c) The top part depicts a support and the bottom part is its triangulated counterpart. The dual tree is weighted using the number of triangles embedded in the triangles corresponding to the nodes.

Using the dual representation for the support, $|\check{M}_v|_\Delta$ is found by summing the tree nodes; $|M_v|_\Delta$ is a weighted version of the sum. At the center of the support, the tree is balanced, i.e. each node has two children. However, the tree becomes unbalanced towards the boundaries. Figure A.1b depicts the tree at the bottom part of the support (note that the same tree expands towards the other cardinal directions). The label “R” in Figure A.1a and A.1b points out the sibling nodes. The shaded region in Figure A.1b shows the unbalanced part of the tree.

We count the tree nodes as follows: We compute two sums, one for the balanced part and one for the unbalanced part. Assume that i counts the tree levels and denote

by $|\check{M}_v|_{\Delta_b}(i)$ the number of triangles in the balanced part, then

$$|\check{M}_v|_{\Delta_b}(i) = \sum_{k=1}^i 2^k = 2^{i+1} - 2. \quad i \leq 4. \quad (\text{A.1})$$

Assume now that j counts the unbalanced levels (vertical axis in Figure A.1b), i.e. $j = i - 4$. Then for the unbalanced part, we use the following observation: For j odd 2^j nodes have two children and $2^{j+1} - 2$ nodes have one child. Moreover for j even, 2^{j+1} nodes have two children and $2^{j+1} - 2$ nodes have one child. Then, the sum of nodes for the unbalanced part is again split into two sums: one over odd j and one over even j . For any j , we have $j - \lfloor j/2 \rfloor$ odd indices and $\lfloor j/2 \rfloor$ even indices. We denote by $|\check{M}_v|_{\Delta_u}(j)$ the the number of nodes in the unbalanced part, then for $j \geq 1$

$$\begin{aligned} |\check{M}_v|_{\Delta_u}(j) &= 4 \left(\sum_{k=1}^{j-\lfloor j/2 \rfloor} (2^{k+2} - 2) + \sum_{k=1}^{\lfloor j/2 \rfloor} (2^{k+2} + 2^{k+1} - 2) \right), \\ &= 32(2^{j-\lfloor \frac{j}{2} \rfloor} + 3 \cdot 2^{\lfloor \frac{j}{2} \rfloor - 1}) - 8j - 80. \end{aligned} \quad (\text{A.2})$$

Hence, for $i > 4$

$$\begin{aligned} |\check{M}_v|_{\Delta}(i) &= |\check{M}_v|_{\Delta_b}(4) + |\check{M}_v|_{\Delta_u}(i - 4), \\ &= 32(2^{i-\lfloor \frac{i-4}{2} \rfloor - 4} + 3 \cdot 2^{\lfloor \frac{i-4}{2} \rfloor - 1}) - 8i - 18. \end{aligned} \quad (\text{A.3})$$

We now compute $|M_v|_{\Delta}$ using a weighted version of the sums (A.1) and (A.2). Each tree node is weighted using the number of triangles embedded in the triangle represented by the node. This number is a function of the total number of tree levels i and the level k of the tree node. More precisely, the weight is given by $w_k = 2^{i-k+1}$. Hence, the weighted sum of (A.1) yields

$$|M_v|_{\Delta_b}(i) = \sum_{k=1}^i 2^{i-k+1} 2^k = \sum_{k=1}^i 2^{i+1} = i \cdot 2^{i+1}. \quad i \leq 4. \quad (\text{A.4})$$

Since we used two sums for the unbalanced part, we use $w_k^0 = 2^{j-2k+1}$ for the sum over even j 's and $w_k^1 = 2^{j-2k+2}$ for the sum over odd j 's. Hence,

$$\begin{aligned} |M_v|_{\Delta_u}(j) &= 4 \left(\sum_{k=1}^{j-\lfloor j/2 \rfloor} (2^{k+2} - 2) 2^{j-2k+2} + \sum_{k=1}^{\lfloor j/2 \rfloor} (2^{k+2} + 2^{k+1} - 2) 2^{j-2k+1} \right), \\ &= 24 \cdot 2^j + \frac{8}{3} 2^{2\lfloor \frac{j}{2} \rfloor - j} - 16 \cdot 2^{\lfloor \frac{j}{2} \rfloor} - 12 \cdot 2^{j-\lfloor \frac{j}{2} \rfloor} + \frac{4}{3} 2^{j-2\lfloor \frac{j}{2} \rfloor}. \end{aligned} \quad (\text{A.5})$$

Hence, for $i > 4$

$$\begin{aligned}
|M_v|_{\Delta}(i) &= |M_v|_{\Delta_b}(4) + |M_v|_{\Delta_u}(i-4), \\
&= 128 \cdot c_2 + 4(c_2 \cdot (24 - 12 \cdot c_1^{-1} + \frac{4}{3}c_1^{-2}) \\
&\quad + \frac{8}{3}c_2^{-1}c_1^2 - 16 \cdot c_1),
\end{aligned} \tag{A.6}$$

where $c_1 = 2^{\lfloor \frac{i-4}{2} \rfloor}$ and $c_2 = 2^{i-4}$. To conclude, we need to express (A.3) and (A.6) in terms of the total number of triangles n and the connection step l . The parameter i is linked to n and l as $2^i = n \cdot 2^l$, or equivalently $n = 4^{(i+l)/2}$. For example, a triplet i, n, l is found as follows: In a uniform mesh of $n = 8$ triangles, $i = 2$ tree levels are needed to compute $|M_v|_{\Delta}(l = 1, n = 8)$ or $|\check{M}_v|_{\Delta}(l = 1, n = 8)$. Hence, we replace $i = 2 \cdot \log_4 n - l$ in (A.1), (A.3), (A.4) and (A.6). Therefore, for $|M_v|_{\Delta}(i)$ we have

$$|M_v|_{\Delta}(l, n) = \begin{cases} (2 \log_4 n - l)2^{1-l}n, & l > 2d - 4, \\ 128 \cdot c_2 + 4(c_2 \cdot (24 - 12 \cdot c_1^{-1} + \frac{4}{3}c_1^{-2}) \\ + \frac{8}{3}c_2^{-1}c_1^2 - 16 \cdot c_1), & l \leq 2d - 4, \end{cases} \tag{A.7}$$

with $c_1(l, n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$, $c_2(l, n) = \frac{2^{-l}n}{16}$, $1 \geq l \geq 2d$ and $n = 2 \cdot 4^d$.

whereas for $|\check{M}_v|_{\Delta}(i)$, we have

$$|\check{M}_v|_{\Delta}(l, n) = \begin{cases} 2^{1-l}n - 2, & l > 2 \log_4 \frac{n}{2} - 4, \\ 32(2^{-(l+4)}n \cdot c_1^{-1} + \frac{3}{2} \cdot c_1) \\ -16(\log_4 n - \frac{l}{2}) - 18, & l \leq 2 \log_4 \frac{n}{2} - 4. \end{cases} \tag{A.8}$$

and $c_1(l, n) = 2^{\lfloor \log_4 n - \frac{l+4}{2} \rfloor}$, $1 \geq l \geq 2d$ and $n = 2 \cdot 4^d$. \square

References

- [1] L. Balmelli. Rate-distortion optimal mesh simplification for communications. *Ph.D dissertation no 2260, Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland.*, 2000.
- [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] E. Catmull. A subdivision algorithm for computer display of curved surfaces. *Ph.D dissertation, Report UTEC-CSS-74-133, Computer Science Department, University of Utah*, December 1974.

- [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] J.H. Clark. A fast algorithm for rendering parametric surfaces. *Proceedings of SIGGRAPH*, pages 289–99, 1979.
- [6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, algorithms and applications*. Springer-Verlag, 2000.
- [7] 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.
- [8] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.
- [9] I. Guskov, W. Swelden, and P. Schröder. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH*, pages 325–334, 1999.
- [10] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. *proceedings of SIGGRAPH*, pages 271–278, 2000.
- [11] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry*, 14(1-3):5–24, 1999.
- [12] U. Labsik, L. Kobbelt, R. Schneider, and H.-P. Seidel. Progressive transmission of subdivision surfaces. *Computational Geometry*, 15(1-3):25–39, 2000.
- [13] F. Laves. Die Bau-zusammenhänge innerhalb der Kristallstrukturen. *Zeitschrift für Kristallographie*, 73:202–265, 1930.
- [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] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH*, pages 109–118, 1996.
- [16] C. Loop. Smooth subdivision surfaces based on triangles. *Master’s thesis, University of Utah, Department of Mathematics*, 1987.
- [17] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. *Proceedings of IEEE Visualization*, pages 299–305, 1998.
- [18] H. Samet. *Application of Spatial Data Structures: Computer Graphics, Image Processing and GIS*. Addison-Wesley Publishing Company, 1990.
- [19] L. Velho and D. Zorin. 4-8 subdivision. *Computer-Aided Geometric Design, Special Issue on Subdivision Techniques.*, 2001.
- [20] D. Zorin. A method for analysis of C^1 -continuity of subdivision surfaces. *SIAM Journal of Numerical Analysis*, 37(4):1677–1708, 2000.