Quadtrees for Embedded Surface Visualization: Constraints and Efficient Data Structures

Laurent Balmelli and Jelena Kovačević
Bell Laboratories, Lucent Technologies
Murray Hill, NJ
{balmelli,jelena}@research.bell-labs.com

Martin Vetterli
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
Martin.Vetterli@epfl.ch

Abstract

The quadtree data structure is widely used in digital image processing and computer graphics for modeling spatial segmentation of images and surfaces. A quadtree is a tree in which each node has four descendants. Since most algorithms based on quadtrees require complex navigation between nodes, efficient traversal methods as well as efficient storage techniques are of great interest. In this paper, we first propose an efficient indexing scheme for a linear (pointerless) quadtree data structure. Such a quadtree is stored using a unidimensional array of nodes. Our indexing scheme has the property that the navigation between any pair of nodes can be computed in constant time. Moreover, the navigation across multiple quadtrees can be achieved at the same cost. We illustrate our results on applications in computer graphics. We first show how the problem of computing a so-called restricted quadtree *can be solved at optimal cost, e.g with a computational complexity having the order of magnitude of the problem size. Then, we explain how this problem can be solved in the case of surfaces modeled using multiple quadtrees. Finally, we show how a tessellated sphere can be implemented and navigated using our data structure.*

1 Introduction

The quadtree data structure is a tree in which each node has at most four children. In digital image processing, quadtrees are used to efficiently store image segmentations [3] (see Figure 1(a)) or to compress images by hierarchically storing color intensities [6]. In computer graphics, quadtrees and octrees (tridimensional generalization of quadtrees) are used as spatial partitioning representations. More recently, quadtrees have become popular to store triangulated surfaces [8, 4]. For example, a triangulated surface can represent a planar approximation of a bivariate function $f(u, v)$ (see Figure 1(b)). In this case, each node in the quadtree is used to store a subset of the triangles. An interesting problem is the computation of the so-called *restricted quadtree* [11]. Consider a quadtree used to store a triangulation as depicted in Figure 2. In the figure[1], the arrow links a node and its corresponding region in the triangulation. In this particular case, each leaf node represents a set of eight triangles. This triangulation is *embedded*, in that it can be generated by recursive subdivision of an initial square containing two triangles. Figure 3 shows

[1]Note that for clarity, we link together only the nodes having a common father, and located at the same level.

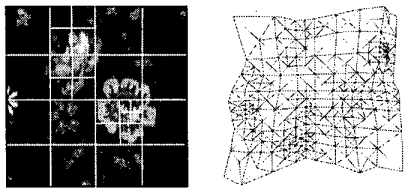

Figure 1: (a) Image partition. (b) Triangulated surface.

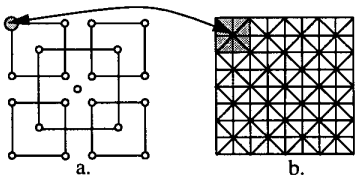

Figure 2: (a) Quadtree. (b) Triangulation.

the subdivision process as well as the vertices (black dots) used to construct the triangles. Each nonleaf node contains a basic set of eight triangles (see Figure 3(c)) embedded in a denser triangulation represented by its corresponding subtree. Due to its recursive nature, the quadtree is therefore adequate to store an embedded triangulation. Note that, although the triangulation in

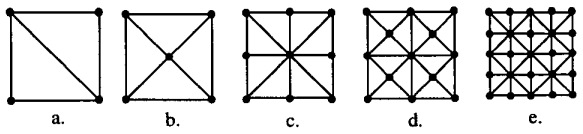

Figure 3: (a)-(e) Subdivision process yielding an embedded set of triangulations.

Figure 2(b) is uniform, the one in Figure 1(b) (represented as a top-viewed rendered surface) is not. This is because the former corresponds to a balanced quadtree (each node has exactly four children), whereas the latter corresponds to an unbalanced quadtree. Moreover, each node may contain from four up to eight triangles. Therefore, a nonuniform triangulation can be computed by deleting nodes of its quadtree representation. This leads to a *restricted quadtree*[11]: consider the deletion a quadtree node, or

0-7803-5467-2/99/ $ 10.00 © 1999 IEEE

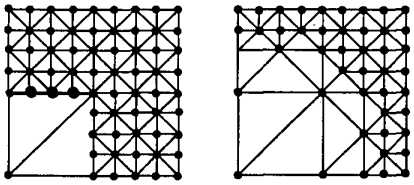

Figure 4: (a) Simplified triangulation leading to discontinuities in the rendered surface. (b) Simplified triangulation corresponding to a restricted quadtree.

equivalently the removal of all triangles represented by the subtree rooted at this particular node. If no care is taken, the triangle edges in the resulting simplified triangulation might not be coplanar, as depicted in Figure 4(a). Therefore, once rendered, the triangulated surface will have discontinuities. To ensure coplanarity, additional nodes have to be pruned at the same time, leading to a restricted quadtree. A restricted quadtree can be seen as the tree satisfying the coplanarity requirement (Figure 5). Although this problem has been studied previously, either only particular cases such as deleting only leaf nodes have been solved [5], or computationally suboptimal algorithms based on a quadtree structure using pointers have been given [10]. We attribute this to a lack of adapted and efficient navigation method for the quadtree data structure. We start with the basic proper-

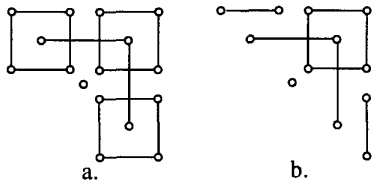

Figure 5: (a) Quadtree corresponding to Figure 4a. (b) Restricted quadtree (corresponding to Figure 4b.)

ties of quadtrees. Then, we introduce an indexing scheme for the quadtree, called *z-ordering*, allowing to navigate between any pair of nodes in constant time. Note that the navigation is not restricted to neighbor nodes, or does not require a particular index format for the node as in previous works [12, 8]. Our indexing scheme allows us to extend the navigation across multiple quadtrees with no increase in cost. We illustrate our results with applications in computer graphics. We first give an algorithm to compute a restricted quadtree having a computational cost of the order of magnitude of the problem size. Then, we explain how this problem can be solved in the case of surfaces modeled using multiple quadtrees. Finally, we show how a tessellated sphere can be implemented and navigated using our data structure.

2 Linear quadtree

A quadtree of depth d contains $\sum_{i=0}^{d-1} 4^i$ nodes, with indices ranging from $0, \ldots, \frac{1}{3}(4^d - 1) - 1$. Given a particular node $p > 0$ and a tree of depth $d > 0$, the following is true:

$$\text{Parent node index:} \quad \lfloor \frac{p-1}{4} \rfloor, \tag{1}$$

$$\text{Children node indices:} \quad 4p + i, \quad i = 1 \ldots 4, \tag{2}$$

$$\text{Level of the node:} \quad \lfloor \log_4(3p+1) \rfloor. \tag{3}$$

The quadtree is simply stored as a linear array of nodes (no explicit index for the nodes needs to be stored). Equations (1)-(3) do not reveal the spatial organization of the nodes, since they are valid for any permutation of the child indices. We organize a quadtree node and its children as in Figure 6(b). Figure 6(a) shows a possible spatial organization for the child node indices, whereas Figure 6(b) shows an alternate one, named *z-ordering* (see solid arrows). 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). We construct the indexing using the ordering of Figure

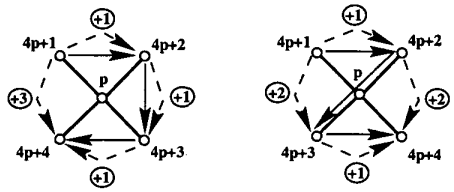

Figure 6: Child node indices: (a) Example. (b) z-ordering.

6(b) for the odd quadtree levels, and a vertically mirrored version for the even levels. Figure 7 shows the resulting indexing for a quadtree of depth 4. Once recursively applied, the z-ordering

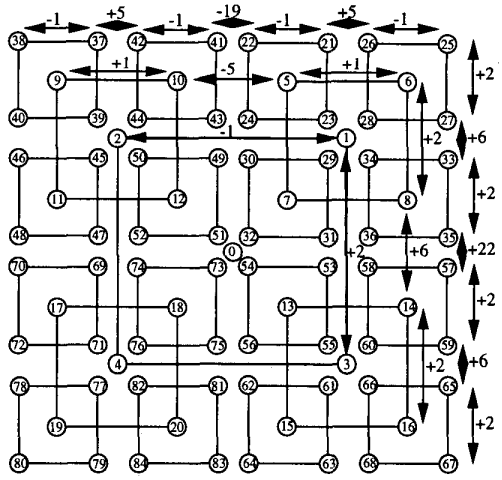

Figure 7: Spatial index organization using z-ordering for a quadtree of depth 4.

yields the following key property: the horizontal/vertical difference between the indices of neighbor pairs is constant for a particular column/row. In Figure 7, the index differences are denoted next to the arrows. It is now possible to derive general expressions

for the distance between two neighbor nodes located at the same level. However, to do this, we first need to find the *closest common father* of the two nodes. In other words, we need to seek the root of the smallest subtree containing both nodes. Consider two nodes p_1 and p_2 located at level r. We call the unique integer v solving (4) the *relative level distance* (RLD) between p_1 and p_2. Their closest common father is located at level $r - v$. The RLD is found solving the following equation

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

We can then derive equations expressing the vertical and horizontal differences between the indices. Assume now that the quadtree has a toroidal structure, e.g that the rightmost nodes, at a particular level, are neighbors with the leftmost nodes, and that the topmost nodes are neighbors with the bottommost nodes. Then, additional equations can be derived to find the neighbor nodes assuming such structure. We will see later that these equations can be used to derive a single expression for the distance between any pair of nodes (not necessarily neighbors). We express now the general distances between the node indices (the proofs are given in the appendix).

Property 2.1. z-ordering
The horizontal/vertical difference between the indices of neighbor pairs, having a relative level distance v, is constant for a particular column/row. The horizontal differences are

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

$$\delta_{th}(v) = \frac{1}{5}4^{v+1} + \frac{1}{5}(-1)^v. \qquad \textit{(toroidal)} \tag{6}$$

The vertical differences are

$$\delta_v(v) = \frac{4}{3}4^v + \frac{2}{3}, \tag{7}$$

$$\delta_{tv}(v) = \frac{2}{3}4^{v+1} - \frac{2}{3}. \qquad \textit{(toroidal)} \tag{8}$$

In the next section, we explain how to use Property 2.1 to navigate the quadtree in constant time.

3 Navigation

Due to the recursive nature of the quadtree, we can state a set of equations to express the relative level distances v^i, e.g solving (4), for any level i of the quadtree:

$$\begin{array}{l} v^1 = (\ 0 \quad \phi_1 \quad 0\ ), \qquad \phi_1 = (\ 0\ ), \\ v^i = (\ i-1 \quad \phi_i \quad i-1\ ), \\ \phi_i = (\ \phi_{i-1} \quad i-1 \quad \phi_{i-1}\ ). \end{array} \tag{9}$$

These equations generate a vector for each level of the quadtree. The elements of the vector are accessed by a node to find its relative level distance with the western, eastern, northern and southern neighbors. The vectors containing the index difference between the columns and the rows of the spatial quadtree can be derived from (9) and (5)-(8). The horizontal indexing differences between the columns of the quadtree Δh^i, at level i, are given by

$$\Delta h^1 = (\ \delta_{th}(0) \quad -\delta_h(0) \quad \delta_{th}(0)\ ), \tag{10}$$

$$\phi_1 = (\ -\delta_h(0)\ ), \tag{11}$$

$$\Delta h^i = (\ \delta_{th}(i-1) \quad \phi_i \quad \delta_{th}(i-1)\ ), \tag{12}$$

$$\phi_i = (\ -\phi_{i-1} \quad -\delta_h(i-1) \quad -\phi_{i-1}\ ), \tag{13}$$

whereas the vertical indexing differences between the rows Δv^i, at level i, are given by

$$\Delta v^1 = (\ -\delta_{tv}(0) \quad \delta_v(0) \quad -\delta_{tv}(0)\ ), \tag{14}$$

$$\phi_1 = (\ \delta_v(0)\ ), \tag{15}$$

$$\Delta v^i = (\ -\delta_{tv}(i-1) \quad \phi_i \quad -\delta_{tv}(i-1)\ ), \tag{16}$$

$$\phi_i = (\ \phi_{i-1} \quad \delta_v(i-1) \quad \phi_{i-1}\ ). \tag{17}$$

To access the indexing differences (or distance) vectors, we need to compute, for a particular node, its position (n_x, n_y) on a $2^i \times 2^i$ grid, where i is the level at which the node is located in the quadtree. This position can be found using the local index $p_{local} = p - (4^i - 1)/3$ of the node within its level, and examine its corresponding expression in base 4. Assume $\mathbf{q}_l$ to be the coefficients of p_{local} in base 4. Then, since each local node index p_{local} is unique within its level, the coefficients $\mathbf{q}_l = \{q_{l-1}, \ldots, q_0\}$ provide a unique path to its location on the grid. More details can be found in [2]. Using the coordinates (n_x, n_y) and the vectors given by (12) and (16), the distances from node p to its neighbors k's are

$$\text{western neighbor: } k - p = -\Delta h^d[n_x], \tag{18}$$

$$\text{eastern neighbor: } k - p = \Delta h^d[n_x + 1], \tag{19}$$

$$\text{northern neighbor: } k - p = -\Delta v^d[n_y], \tag{20}$$

$$\text{southern neighbor: } k - p = \Delta v^d[n_y + 1], \tag{21}$$

where $\Delta h^d[i]$ and $\Delta v^d[i]$ denote the access to the ith coefficient of the respective vector. Consider, for example, node 8 located at the second level in the quadtree (Figure 7). The local index of this node is $p_{local} = 8 - \frac{1}{3}(4^2 - 1) = 3$ and its expression in base 4 is the sequence $\{q_1, q_0\} = \{0, 3\}$. The location of the node within the level grid is $(n_x, n_y) = (4, 2)$. We can use the distance vectors Δh^2 and Δv^2 (for level 2) to find its neighbors:

$$\text{western neighbor is } k = 8 - \Delta h^2[4] = 7,$$

$$\text{eastern neighbor is } k = 8 + \Delta h^2[5] = 11,$$

$$\text{northern neighbor is } k = 8 - \Delta v^2[2] = 6,$$

$$\text{southern neighbor is } k = 8 + \Delta v^2[3] = 14,$$

as one can verify in Figure 7. Note that node 8 is located on the "edge" of the quadtree at its level. It does not therefore have an eastern neighbor (if we do not consider the quadtree a torus). Consequently, node 11 is node's 8 eastern neighbor using (6).

Equations (5)-(8) can be used in an iterative way to derive a single expression for an arbitrary distance in the quadtree (Figure 8). The idea is to follow a path of nodes, starting from the initial node, leading to the destination node. Note that only the coordinates (n_x, n_y) for the initial node need to be computed, since the subsequent coordinates, for the nodes along the path, can be

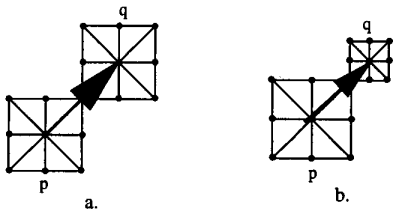

Figure 8: Distances between: (a) two diagonal nodes at the same level. (b) two diagonal nodes at different levels.

easily derived from (n_x, n_y). Consider the node p in Figure 8(a), the distance to its neighbor q is

$$q - p = \Delta h^d[n_x + 1] - \Delta v^d[n_y].$$

For example, if $p = 13$ (then $n_x = 3$ and $n_y = 3$), the north-eastern neighbor q at the same level is

$$q = 13 + \Delta h^3[4] - \Delta v^3[3] = 13 + 1 - 6 = 8.$$

On the other hand, the distance between node p and its neighbor q in Figure 8(b) is given by

$$q - p = 3p + 1 + \Delta h^{d+1}[2n_x + 1] - \Delta v^{d+1}[2n_y - 1].$$

As previously, when $p = 13$, the neighbor q at the next level is

$$q = 13 + 40 + \Delta h^3[7] - \Delta v^3[5] = 53 + 5 - 22 = 36.$$

Both examples can be verified in Figure 7.

4 Applications

In this section, we illustrate our results with two applications in computer graphics. We first give an optimal algorithm to compute a restricted quadtree using a linear implementation of the quadtree. Then, we show how a data structure modeled with multiple quadtrees can be efficiently navigated. A Java applet illustrating the first example is available at `http://lcavwww.epfl.ch/Triangulation`.

Recall the problem introduced in Section 1. The quadtree is used to store an embedded triangulation. This class of triangulations is widely used to model triangulated surfaced from terrain elevation data [7, 9, 1]. When modeled using a quadtree, triangles can be removed from the original triangulation by deleting quadtree nodes (Figure 4). As seen in Section 1, not all quadtree configurations yield a triangulation in which all triangle edges are coplanar (Figure 4 and 5). Such a "valid" quadtree is called a restricted quadtree [11]. We consider here the computation of a restricted quadtree resulting from the deletion of an arbitrary quadtree node. We start by computing the size of the problem and then state an algorithm solving the problem with a computational complexity having the order of magnitude of the problem size. The algorithm is therefore optimal and solves the problem at minimal cost.

According to Figure 4, the amount of nodes to delete with a particular quadtree node to obtain a restricted quadtree, is proportional to the level at which this particular node is located. We evaluate the size of the problem by counting the number of nodes to visit to avoid noncoplanar triangle edges. This number is linearly proportional to the number of triangles contained in the simplified region. Figure 9 depicts the shape of a simplified region. Note that, in Figure 4(b), the region is incomplete because of the node location. In [2], we prove that, in the worst case, the size of the region is $\Theta(\sqrt{n})$, where n is the total number of triangles contained in the quadtree. We also show that, in expectation (i.e. when choosing an arbitrary node), the region has constant size $\Theta(c)$. An optimal algorithm solving the problem with the same complexity magnitudes can be implemented as depicted in Figure 9(b). Using (5)-(8), we construct an algorithm visiting once and only once the nodes containing the triangles to delete jointly, thus solving the problem in its size.

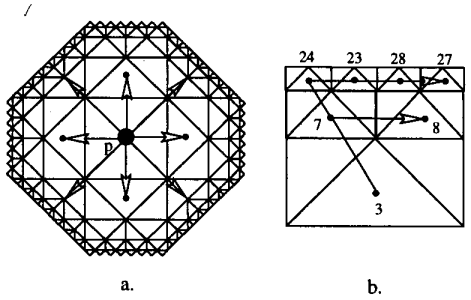

Figure 9: (a) Simplified region after deletion of node p. (b) An example of node traversal.

Consider again the case in which a quadtree is used to store an embedded triangulation. Consider further that the triangulation is distributed among multiple quadtrees, as depicted in Figure 10. In this figure, the triangulation is stored across nine quadtrees and the root node of the central quadtree has been deleted. The simplifications required to obtain a set of restricted quadtrees can be computed using (6) and (8) implementing a toroidal structure. Using the same argument, the same data structure can be used to

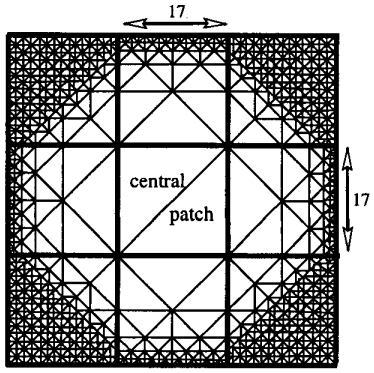

Figure 10: Triangulation formed by multiple quadtrees.

model closed objects such as the sphere. A sphere can be tessellated by radially projecting the vertices of a subdivided octahedron (Figure 11). Such an object can be constructed using two

embedded triangulations, therefore using two quadtrees. To preserve the sphere topology, the root nodes must contain at least the triangles obtained at the second step of the subdivision process (Figure 3(b)).

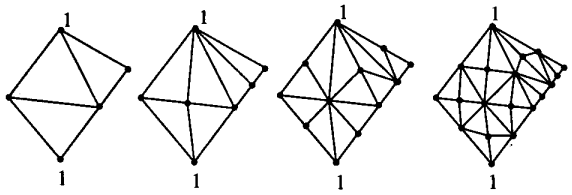

Figure 11: Regularly subdivided octahedron constructed with two triangulations. The radial projection of the vertices can be used to obtain a uniform tessellation of the sphere. Note that the vertices labeled 1 must remain in the model to preserve the topology.

5 Appendix

Proof of Property 2.1 We first prove by induction the horizontal distance $\delta_h(l)$: consider $l = 0$, which gives the horizontal distance between two nodes having the same father node. We then have $\delta_h(l = 0) = 1$, which is consistent with the z-ordering of the indices (see Figure 6(b)). Equation (5) can be rewritten as

$$\delta_h(l) = 4^l + \sum_{n=0}^{l-1} 4^n(-1)^{n+l+1}$$

by using the following:

$$\delta_h(l) = 4\delta_h(l-1) + (-1)^{l-1}$$

and we show that

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

which proves the equation. Due to the lack of space, we only give the induction steps for the three remaining equations. The same procedure can be repeated for the vertical distance $\delta_v(l)$. When $l = 0$ (relative vertical distance between two nodes having the same father node), we have $\delta_h(l = 0) = 2$, which is consistent with the z-ordering of the indices. Equation (7) can be rewritten as

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

Therefore, the induction step $\delta_v(l) = 4\delta_v(l-1) - 2$ can be used to prove the equation. The horizontal torus is given by 6. One can verify that $\delta_{th}(l = 0) = 1$, which is still compliant with the z-ordering. Equation (6) can be rewritten as

$$\delta_{th}(l) = \sum_{n=0}^{l} 4^n(-1)^{n+l}$$

and the induction step $\delta_{th}(l) = 4\delta_{th}(l-1)+(-1)^l$ can be used to prove the equation. Similarly, we can verify that for the vertical torus distance $\delta_{tv}(l)$, we have $\delta_{tv}(l = 0) = 2$, which is still compliant with the z-ordering. Equation (8) can be rewritten as

$$\delta_{tv}(l) = 2\sum_{n=0}^{l} 4^n,$$

and the induction step $\delta_{tv}(l) = 4\delta_{tv}(l-1) + 2$ can be used to prove the equation. ◇

6 References

[1] L. Balmelli, S. Ayer, and M. Vetterli. Efficient algorithms for embedded rendering of terrain models. *Proc. Int. Conf. Image Processing (ICIP)*, October 1998.

[2] L. Balmelli, J. Kovačević, and M. Vetterli. Progressive and adaptive meshes in a rate-distortion framework. *submitted to IEEE Transaction on Image Processing*, 1999.

[3] R. Gonzalez and P. Wintz. *Digital Image Processing*. Addison Wesley, 1987.

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

[5] B. Von Herzen and A.H. Barr. Accurate triangulation of deformed, intersecting surfaces. *Computer Graphics 21, also Proceeding of ACM SIGGRAPH 87*, 21(4):103–110, July 1987.

[6] F.S. Hill, S. Walker, and F. Gao. Interactive image query system using progressive transmission. *Proceedings of ACM Siggraph*, pages 323–333, 1983.

[7] P. Hughes. Building a terrain renderer. *Computers in Physics*, pages 434–437, July/August 1991.

[8] M. Lee and H. Samet. Navigating through triangle meshes implemented as linear quadtrees. *Technical Report, Computer Science Dept., Center for Automation Research, University of Maryland*, CS-TR-3900, April 1998.

[9] 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. *ACM SIGGRAPH*, pages 109–118, 1996.

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

[11] H. Samet. *Application of Spatial Data Structures*. Addison-Wesley Publishing Company, 1989.

[12] G. Schrack. Finding neighbors of equal size in linear quadtrees and octrees in constant time. *CVGIP: Image Understanding*, 55(3):221–230, May 1992.