# Cost-Optimal Trees for Ray Shooting

Hervé Brönnimann[1⋆] and Marc Glisse[2⋆⋆]

[1] Computer and Information Science, Polytechnic University, Six Metrotech Center, Brooklyn, NY 11201, USA.
[2] Ecole Normale Suprieure Paris, 45 Rue d'Ulm, 75230 Paris, Cedex 5, France.

**Abstract.** Predicting and optimizing the performance of ray shooting is a very important problem in computer graphics due to the severe computational demands of ray tracing and other applications, e.g., radio propagation simulation. Aronov and Fortune were the first to guarantee an overall performance within a constant factor of optimal in the following model of computation: build a triangulation compatible with the scene, and shoot rays by locating origin and traversing until hit is found. Triangulations are not a very popular model in computer graphics, but space decompositions like kd-trees and octrees are used routinely. Aronov et al. [1] developed a cost measure for such decompositions, and proved it to reliably predict the average cost of ray shooting.
In this paper, we address the corresponding optimization problem, and more generally $d$-dimensional trees with the cost measure as the optimizing criterion. We give a construction of quadtrees and octrees which yields cost $O(M)$, where $M$ is the infimum of the cost measure on all trees, for points or for $(d-1)$-simplices. Sometimes, a balance condition is important. (Informally, balanced trees ensures that adjacent leaves have similar size.) We also show that rebalancing does not affect the cost by more than a constant multiplicative factor, for both points and $(d-1)$-simplices. To our knowledge, these are the only results that provide performance guarantees within approximation factor of optimality for 3-dimensional ray shooting with the octree model of computation.

## 1 Introduction

Given a set $S$ of objects, called a *scene*, the ray-shooting problem asks, given a ray, what is the first object in $S$ intersected by this ray. Solving this problem is essential in answering visibility queries. Such queries are used in computer graphics (e.g., ray tracing and radiosity techniques for photo-realistic 3D rendering), radio- and wave-propagation simulation, and a host of other practical problems.

A popular approach to speed up ray-shooting queries is to construct a space decomposition such as a quadtree in 2D and an octree in 3D. The query is then

⋆ Work on this paper has been supported by NSF ITR Grant CCR-0081964 and NSF CAREER Grant CCR-0133599.

⋆⋆ Work by the second author was performed while spending a semester at Polytechnic University, on leave from École Normale Supérieure, Paris, France.

answered by traversing the leaves of the tree as they are intersected by the ray, and for each cell in turn, testing for an intersection between the ray and the subset of objects intersecting that cell. The performance of such an approach greatly depends on the quality of that space decomposition.

Unfortunately, not much is understood about how to measure this quality. The ultimate criterion is running time, which also involves many factors (among which hardware acceleration and memory hierarchy side effects) and in any case is not directly amenable to analysis. Practioners use a host of heuristics and parameters of the scene, of which the object count is not the most important; more important seems to be the size of the objects in the scene, and other properties of the object distribution (density, depth complexity, surface area of the subdivision). Those parameters are used to develop automatic termination criteria for recursively constructing the decompositions (see Section 3). While they perform acceptably well most of the time, none of these heuristics performs better than the brute-force method in the worst case. More importantly, occasionally the termination criteria will produce a bad decomposition, and in any case there is no way to know the quality of the decomposition because lower bounds are hard to come by.

In [1], we proposed a measure for bounded-degree space decompositions, based on the surface area heuristic, which is a simplification (for practicality) of a more complicated but theoretically sound cost measure: under certain assumptions on the ray distribution, the cost measure provably reflects the cost of shooting an average ray using the space decomposition. So far, we have focused on experimentally proving that all the assumptions we make are warranted and do not greatly affect the effectiveness of the cost measure. Thus the cost measure, introduced in the next section, which is simple to compute and accurate, can guide us in optimizing the data structure. The topic of this paper is to present new algorithms and analyses for constructing provably (near-)optimal decompositions with respect to this cost measure.

*Related work* There has been a lot of work on quadtrees and octrees in the mesh generation and graphics community (see the book by Samet [23], the thesis of Moore [18], or the survey by Bern and Eppstein [7] for references). Since they are used for discretizing the underlying space, usual considerations include the tradeoff between the size of the tree and their accuracy with respect to a certain measure (that usually evaluates a maximum approximation error with respect to some surface). These are not usually relevant for ray shooting.

There is a rich history of data-structure optimization for ray tracing and other ray-shooting-related problems in computer graphics. Cost measures have been proposed for ray shooting in octrees by McDonald and Booth [16], Reinhard and coll. [21, 22], Whang and coll. [28]. Similar optimizations have been tried for related data structures, such as bounding volume hierarchies (BVH) [14, 24, 26], BSP-trees [20], uniform grids [12], and hierarchical uniform grids (HUG) [10]. We should also mention the work of Havran and coll. [15], who propose a method to determine experimentally the most efficient space subdivision for a given scene, by picking a similar scene (according to certain characteristics such as

size, number of objects, densities, etc.) in a database of scenes, for which the most efficient scheme has been determined. All of these approaches use heuristic criterion (sometimes very effectively) but none offer theoretical guarantees.

*Our results* In [9] and in this paper, we are interested in constructing trees with cost as low as possible, with a guaranteed approximation ratio. The only objects we consider are simplices (points and segments inside the unit square $[0,1]^2$ in $\mathbb{R}^2$, or points, segments and triangles inside the unit cube $[0,1]^3$ in $\mathbb{R}^3$). We however assume the Real-RAM model so as to avoid a discussion on the bit-length of the coordinates.

We give and analyze algorithms that produce trees with cost O(M), where M is a lower bound on the cost of any tree. The novelty from [9] is the extension to d=3 and higher of the results. we also examine the effect of rebalancing the tree on the cost measure, and prove that rebalancing only increases the cost by a constant multiplicative factor. While there are many subdivision criteria used by practitioners in computer graphics, our subdivision scheme is the first to our knowledge to guarantee an approximation ratio.

In a follow-up paper [2] to [1], we evaluated several heuristics, including those presented here, to optimize the cost value of an octree for a given scene. Both our algorithm and a simpler heuristic (namely, greedy without lookahead in this paper's terminology) gave the best cost. The optimal guarantees we obtain here nicely strengthen the experimental results and helped identify a simplification which offers a performance similar to the guaranteed algorithm proposed in this paper.

## 2 General cost measure results

The following cost measure was introduced by Aronov *et al.* [1] for the purpose of predicting the traversal cost of shooting a ray in $\mathbb{R}^d$, while using a quadtree (for $d = 2$) or an octree (for $d = 3$) to store $S$:

$$c_S(\mathcal{T}) = \sum_{\sigma \in \mathcal{L}(\mathcal{T})} (\gamma + |S \cap \sigma|) \times \lambda_{d-1}(\sigma), \tag{1}$$

where $\mathcal{L}(\mathcal{T})$ is the set of leaves of the quadtree, $S \cap \sigma$ is the set of scene objects meeting a leaf $\sigma$, and $\lambda_{d-1}(\sigma)$ is the perimeter length (if $d = 2$) or surface area (if $d = 3$) of $\sigma$.

This cost function provably models the cost of finding all the objects intersected by a random line, with respect to the rigid-motion invariant distribution of lines [1]. Here's an overly simplified explanation why: when shooting a ray, the octree is traversed and all the objects in a traversed leaf are tested against the ray to find the first hit. The cost in a leaf $\sigma$ is thus $O(\gamma + |S \cap \sigma|)$. The coefficient $\gamma$ depends on the implementation, and models the ratio of the cost of the tree traversal (per cell) to that of a ray-object intersection test (per test).[1]

[1] In [2], we show how to choose $\gamma$ to reliably get the best cost possible.

But integral geometry tells us that a random ray will intersect $\sigma$ with probability $\lambda_{d-1}(\sigma)$ (this is not quite true; read [1] for the niceties). Hence the average cost of ray shooting is given by the weighted average, exactly what our cost measure computes.

*Tree and object costs.* Observe that the cost measure can be split into two terms: $c_t(\mathcal{T}) = \gamma\lambda_{d-1}(\mathcal{T}) = \gamma\sum_{\sigma\in\mathcal{L}(\mathcal{T})}\lambda_{d-1}(\sigma)$ (the *tree cost*), and $c_o(\mathcal{T}) = \sum_{s\in S}\lambda_{d-1}(s\cap\mathcal{T})$ (the *object cost*), where $s\cap\mathcal{T}$ denote the set of leaves of $\mathcal{T}$ crossed by $s$ and $\lambda_{d-1}$ is extended to sets of leaves by summation. It is useful to keep in mind the following simple observations: when subdividing a cell $\sigma$, the total tree cost of its children is twice the tree cost of $\sigma$, and the object cost of an object is multiplied by $m/2^{d-1}$ where $m\in[1\ldots 2^d]$ is the number of children intersected by the object. Note that $m\le 3$ for a segment in 2D and $m\le 7$ for a triangle in 3D (unless they pass through the center of the cell). As the tree grows finer, the tree cost increases while the object cost presumably decreases.

**Lemma 1.** *For any set $S$ of simplices in $[0,1]^d$, $c(\mathcal{T})\ge 2d\gamma+d\sqrt{2}\sum_{s\in S}\lambda_{d-1}(s)$, for any $d\ge 2$.*

**Proof.** The tree cost cannot be less than $\lambda_{d-1}([0,1]^d)\gamma = 2d\gamma$, and the object cost cannot be less than $\sum_{s\in S}\lambda_{d-1}(s)S$. We can improve this lower bound further by noting that any leaf $\sigma$ that is intersected by an object $s$ has area at least $d\sqrt{2}$ times $\lambda_{d-1}(s\cap\sigma)$. Indeed, the smallest ratio $\lambda_{d-1}(\sigma)/\lambda_{d-1}(s\cap\sigma)$ happens when $s$ maximizes $\lambda_{d-1}(s\cap\sigma)$; this happens for a diagonal segment of length $\sqrt{2}$ for the unit square (of perimeter 4), and for a maximal rectangular section of area $\sqrt{2}$ for the unit cube (of area 6). In fact, the maximal section of the unit $d$-cube is $\sqrt{2}$ [6], hence the ratio is at least $2d/\sqrt{2} = d\sqrt{2}$ in any dimension. ∎

## 3 Tree construction schemes

All we said so far was independent of the particular algorithm used to construct the tree. In this section, we introduce several construction schemes and explore their basic properties.

*Terminology and notation.* We follow the same terminology as [9], and generalize it to encompass any dimension. For the $d$-cube $[0,1]^d$ and the cells of the decomposition, we borrow the usual terminology of polytopes (vertex, facet, $h$-face, etc.). The *square* is a quadtree that has a single leaf (no subdivision), the *cube* is an octree with a single leaf, and the *$d$-cube* is a single-leaf tree (for any $d$). We call this tree the *unit cell* and denote it by $\mathcal{T}^{(\text{empty})}$. If we subdivide this leaf recursively until depth $k$, we get a *complete tree (of depth $k$)*, denoted by $\mathcal{T}_k^{(\text{complete})}$, and its leaves form a regular $d$-dimensional grid with $2^k$ cells on each side. In a quadtree, if only the cells incident to one facet (resp. $d$ facets sharing a vertex, or touching any of the $2d$ facets) of a cell are subdivided, and

this recursively until depth $k$, the subtree rooted at that cell is called a *$k$-side* (resp. *$k$-corner* and *$k$-border*) tree, and denoted by $\mathcal{T}_k^{(\text{side})}$ (resp. $\mathcal{T}_k^{(\text{corner})}$ and $\mathcal{T}_k^{(\text{border})}$); see Figure 1 for an illustration of the 2D case. In higher dimensions, there are other cases (one for each dimension between 1 and $d-2$). All this notation is extended to starting from a cell $\sigma$ instead of a unit cell, by substituting $\sigma$ for $\mathcal{T}$: for instance, the complete subtree of depth $k$ subdividing $\sigma$ is denoted by $\sigma_k^{(\text{complete})}$.

The subdivision operation induces a partial ordering $\prec$ on trees, whose minimum is the unit cell. Again, this partial ordering is extended to subtrees of a fixed cell $\sigma$.

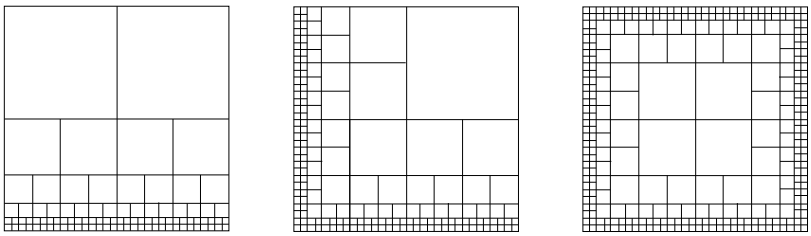

**Fig. 1.** The $k$-side quadtree $\mathcal{Q}_k^{(\text{side})}$ (left), a corner $\mathcal{Q}_k^{(\text{corner})}$ (center), and a border $\mathcal{Q}_k^{(\text{border})}$ (right).

We consider recursive algorithms for computing a tree of a given set $S$ of objects, which subdivide each cell until some given termination criterion is satisfied. In particular, we may recursively subdivide the unit cell until each leaf meets at most one object. We call this the *separation criterion*, and the resulting tree the *minimum separating tree*, denoted $\mathcal{T}^{(\text{sep})}(S)$, with variants where the recursion stops at depth $k$, denoted $\mathcal{T}_k^{(\text{sep})}(S)$. (Note that the depth of $\mathcal{T}^{(\text{sep})}$ is always infinite if any two simplices intersect.) In 3D, for non-intersecting triangles, a variant of [3] stops the recursive subdivision also when no triangle edge intersects the leaf (but any number of non-intersecting triangle interiors may slice the leaf). We will not analyze this variant in this paper.

*Dynamic programming and greedy strategies.* As introduced in [9], the *dynamic programming algorithm* finds the tree that minimizes the cost over all trees with depth at most $k$, which we denote by $\mathcal{T}_k^{(\text{opt})}(S)$ (or $\sigma_k^{(\text{opt})}(S)$ if we start from a cell $\sigma$ instead of the unit cell): the algorithm starts with the complete tree $\mathcal{T}_k^{(\text{complete})}$, and simply performs a bottom-up traversal of all the nodes, while maintaining the optimum cost of a tree rooted at that node. The decision whether to keep the subtree of a cell or prune it is based on the cost of the cell vs. the sum of the optimum costs of the subtrees rooted at its $2^d$ children.

Unfortunately, the memory requirements of this algorithm are huge for large values of $k$ (although they remain polynomial if $k = \Theta(\log n)$; see next section). Therefore we also propose a greedy strategy with bounded lookahead: the algorithm proceeds by recursively subdiving the nodes with a greedy termination criterion: when examining a cell $\sigma$, we run the dynamic programming within $\sigma$ with depth $k$ ($k$ is a parameter called *lookahead*). If the best subtree $\sigma_k^{(\text{opt})}(S)$ does not improve the cost of the unsubdivided node $\sigma$, then the recursion terminates. Otherwise, we replace $\sigma$ by the subtree $\sigma_k^{(\text{opt})}(S)$ and recursively evaluate the criterion for the leaves of $\sigma_k^{(\text{opt})}(S)$. We call this the $k$-greedy strategy, and denote the resulting tree by $\mathcal{T}^{(k\text{-greedy})}(S)$ (or $\sigma^{(k\text{-greedy})}(S)$ if we start from a cell $\sigma$ instead of the unit cell). Note that unlike all the other quadtrees constructed up to now, that tree could be infinite. We use the notation $\mathcal{T}_\ell^{(k\text{-greedy})}$ to denote the tree constructed with the $k$-greedy lookahead criterion combined with a maximum depth of $\ell$.

With no lookahead ($k = 1$), the *greedy strategy* simply examines whether one subdivision decreases the cost measure. Below, we show that this does not yield good trees in general. We will analyze the greedy strategies without and with lookahead, first for points, then for simplices. But first, we must grapple with the issue of infinite depth.

*Pruning beyond a given depth.* The "optimal" tree may not have finite depth (it is conceivably possible to decrease the cost by subdividing ad infinitum), so we let $M$ denote the infimum of $c(\mathcal{T})$ over all trees $\mathcal{T}$ for $S$. (As a consequence of Lemma 1, $M \geq 2d\gamma > 0$.) In order to have an algorithm that terminates, we usually add an extra termination criterion such as a depth limit of $k$.

We now show that pruning a tree at a depth of $k$, for some choice of $k = \Theta(\log n)$ (to ensure that the tree has a polynomial size), increases cost at most by a constant factor. We first show it for arbitrary convex obstacles (simplices in particular). Then we improve on the result for the case of points. The proofs are no difficult and omitted for space consideration. Nevertheless, these considerations of depths are necessary to ensure that the computation is meaningful.

**Lemma 2.** *Let $\mathcal{T}$ be a $d$-dimensional tree which stores a set $S$ of $n$ convex objects of dimensions not more than $d-1$. For $k = \log_2 n + C$, let $\mathcal{T}_k$ be the tree obtained from $\mathcal{T}$ by removing every cell of depth greater than $k$. Then $c(\mathcal{T}_k) = O(c(\mathcal{T}))$ and the constant does not depend on $n$ nor $S$.*

*Remark* A choice of $k = \log_2 n + C$ ensures that $\mathcal{T}_k$ has at most $(2^k)^d = O(n^d)$ leaves, for any fixed $d$. Hence the algorithm which computes the full subdivision at depth $k$ and then applies the dynamic programming heuristic provably computes a tree whose cost is $O(M)$ in polynomial time, as a consequence of Lemma 2.

As a side note, with slightly more restrictive hypotheses on $\mathcal{T}$, the depth $k$ can be reduced for points so that $\mathcal{T}_k$ has size at most $O(n^{1+\frac{1}{d-1}}) = O(n^2)$ (for any $d \geq 2$) and cost as close as desired to that of $\mathcal{T}$.

**Lemma 3.** *Let $\mathcal{T}$ be a $d$-dimensional tree, which stores a set $S$ of $n$ simplices. Assume that $\mathcal{T}$ does not contain empty internal nodes (i.e. that are subdvided but do not contain any object). Let $\mathcal{T}_k$ be the tree obtained from $\mathcal{T}$ by removing every cell of depth greater than $k$. Then, for every $\varepsilon > 0$ there exists a $C$ (that depends only on $\varepsilon$ and $\gamma$ but not on $n$) such that, for $k = \frac{1}{d-1}\log_2 n + C$, we have*

$$c(\mathcal{T}_k) \leq (1+\varepsilon)c(\mathcal{T}).$$

## 4 Cost-optimal trees

The following lemma was proven in [9] for the case $d = 2$. Its statement and proof extend straightforwardly to higher dimensions.

**Lemma 4 ( [9]).** *The lookahead greedy strategy does not always give a cost-optimal tree. Specifically, for any $k$, there is a set $S$ of $n$ objects such that no tree of depth at most $k$ has cost less than $2d(\gamma + n)$, but some quadtree of depth at least $k+1$ has cost less than $2d(\gamma + n)$.*

The counterexample for $d = 2$ consists of $n$ copies of the segment $pq$, where $p = (1-2^{-m-1}, 2^{-m-1})$, and $q = (2^{-m-1}, 1-2^{-m-1})$, and considering the cost-optimal quadtree $\mathcal{Q}^{(2)}_{k,m}$ of depth at most $k$. As long as $k \leq m+1$, the situation is similar to the case where $pq$ is the whole diagonal and the cost-optimal quadtree of depth at most $k$ is the square, as can easily be verified. When $k$ becomes larger, however, it becomes interesting to subdivide the corners.

In this example, the ratio between the cost of the optimal tree and the cost of the initial tree is bounded below by a constant. Thus in that case the tree we obtained is a good approximation of the optimal solution. In fact, this can be proven for all sets of objects.

**Lemma 5.** *Given a set $S$ of flat objects in the unit cube, and let $M$ be the infimum of $c(\mathcal{T})$ over all trees $\mathcal{T}$. There is an integer $p$ (for $d = 2$ or $d = 3$, $p = 3$) such that the tree $\mathcal{T}^{(p\text{-greedy})}$ constructed by the $p$-greedy strategy has cost $c(\mathcal{T}^{(p\text{-greedy})}) = O(M)$.*

**Proof.** The intuition is that small objects behave well, and the cost of a big object is bounded below by a constant times its size so it cannot be reduced by very much. Let us look at a cell $\sigma$ of the tree $\mathcal{T}^{(p\text{-greedy})}$: we are going to show that, when the optimal decomposition of depth at most $p$ of a cell $\sigma$ does not improve on the cost of $\sigma$, then the cost of $\sigma$ is $O(M_\sigma)$ where $M_\sigma$ is the infimum cost of all the possible tree subdivisions of $\sigma$. If this holds true for every leaf $\sigma$ of the $p$-greedy strategy, then $c(\mathcal{Q}^{(p\text{-greedy})}) = O(M)$ as well.

Assume there are $a$ objects meeting at most $C_d(2^p)$ cells, and $b$ other objects. The cost of $\sigma$ is $(\gamma + a + b)\lambda_1(\sigma)$. Since $\sigma_p^{(\text{complete})}$ has cost at most $\left(2^p\gamma + a\frac{C_d(2^p)}{2^{p(d-1)}} + 2^p b\right)\lambda_1(\sigma)$, which we assumed to be at least $c(\sigma)$, we have

$c(\sigma) = (\gamma + a + b)\,\lambda_1(\sigma) \leq \left(2^p\gamma + a\frac{C_d(2^p)}{2^{p(d-1)}} + 2^p b\right)\lambda_1(\sigma)$, which implies that $a \leq (\gamma + b)(2^p - 1)\left(1 - \frac{C_d(2^p)}{2^{p(d-1)}}\right)^{-1}$. We will need a technical lemma:

**Lemma 6.** *For every $d$ and $k$, there exist constants $C_d(k)$ and $S_d(k) > 0$ such that for any convex object $s$ of dimension at most $d-1$, either $s$ intersects at most $C_d(k)$ cells of the regular grid of side $k$, or else $\lambda_{d-1}(s) \geq S_d(k)$. We may take $C_d(k) \leq d^2k^{d-2}$ and $C_3(k) = 7k - 6$.*

Let $p$ be the smallest integer such that $C_d(2^p) < (2^p)^{d-1}$. By lemma 6, an object which belongs to more than $C_d(2^p)$ cells has measure at least $S_d(2^p)$ so its contribution to the cost is at least $(d\sqrt{2}S_d(2^p))\lambda_{d-1}(\sigma)$. The optimal cost $M_\sigma$ is then greater than $(\gamma + bd\sqrt{2}S_d(2^p))\lambda_{d-1}(\sigma)$. We have then proved that $M_\sigma$ is at least a fixed fraction of the cost of $\sigma$, and the lemma follows. ∎

Already in 2D, the separating quadtree strategy does not work as well for segments as for points, especially since it is not able to distinguish between a segment that barely intersects the corner of the square and the diagonal (in the first case it is usually good to subdivide, and in the second case it is not). The lookahead strategy is then a true improvement.

*The case of points.* Arguably, the case of points is of theoretical interest only, but has relevance since simplices are usually very small in a graphics scene (when they come from a subdivision surface), and can be thought of as points. This is lent credence by a recent trend: point cloud data (PCD) is becoming an important primitive in computer graphics, and several algorithms for rendering them have been given of late, which are amenable our cost measure. In addition, points are also produced by sampling, 3D scanners, etc.; even though ray shooting makes little sense for these applications the points can be in an octree and the cost may have other significance there.

We first examine the simple termination criteria. Take the example of $n$ points in a corner, $S_n$, described in our configuration **2**. In 2D, with no subdivision, the cost is $4\,(\gamma + n)$. With full subdivision to depth $k$, the cost is $2^{k+2}\gamma + n2^{2-k}$ which is at least $8\sqrt{\gamma n}$ for any value of $k$. Now the separation criterion stopped at depth $k$ gives a cost of $12\gamma + (4n - 3\gamma)2^{-k}$ (this is also what the greedy algorithm leads to). For large values of $n$, these are respectively $\Theta(n)$, $\Omega(\sqrt{n})$ and $O(1)$. Thus no subdivision or a complete subdivision don't approximate the optimum very well.

In the plane, the 2-greedy strategy may produce a quadtree of cost $\Theta(n)$ times the optimal cost, as can be seen by placing $n$ points in the center of the square: the cost after two subdivisions is $12 + 4(\gamma + n)$, more than the initial cost of $4(\gamma + n)$; hence the tree is not subdivided, yet the cost would tend to 24 if subdivided ad infinitum. In any dimension, the same example shows that the 1-greedy strategy also has a bad approximation ratio of $\Theta(n)$. Nevertheless, with one more level of lookahead, everything works near-optimally. We simply state the lemma and omit the proof, similar the the one given above for simplices.

**Lemma 7.** *The 3-greedy strategy in the plane, and the 2-greedy strategy in $d$ dimensions ($d \geq 3$) produce near-optimal trees for points. Namely, if $S$ be a set of $n$ points in the unit $d$-cube, and $M$ is the infimum of $c(\mathcal{T})$ over all trees $\mathcal{T}$, then $c(\mathcal{T}^{(3\text{-greedy})}) = O(M)$ for all $d \geq 2$, and $c(\mathcal{T}^{(2\text{-greedy})}) = O(M)$ for all $d \geq 3$.*

As for finding the optimal quadtree, the question is still open whether for given values of $\gamma$ (and maybe of $n$) there exists a $k$ such that the lookahead strategy yields the optimal result. All we know is that if $n = 5$ and $\gamma < 1$ tends to 1, the required $k$ tends to infinity. We can also mention that if every point belongs to at most one cell, then $k = 1$ leads to the optimal tree.

## 5 Rebalancing quadtrees and octrees

Quadtrees are used in meshing for computer graphics, and octrees are used as a space subdivision method for ray casting and radiosity methods, for instance. Both quadtrees and octrees are used in scientific computing numerical simulations (e.g., finite element methods). These are but a few applications where quadtrees and octrees have appeared. In all these applications, it can be important to maintain aspect ratio (hence starting with a unit cell) and to ensure that two neighboring cells don't have wildly differing sizes. This has led several authors to propose balancing for trees. Also from our perspective, since the cost measure of [1] provably relates to the cost of traversal only for balanced trees, we are interested in balancing trees as well. In this section, we prove that rebalancing does not affect the cost by more than a multiplicative constant factor.

*Definitions.* Two leaves are *$k$-adjacent* if they intersect in a convex portion of dimension $k$. A tree is called *$k$-balanced* if the depths of any two $k$-adjacent leaves differ by at most one. Notice that when considering two $k$-balanced trees, their intersection, constructed from the unit cell by subdividing all and only cells that are subdivided in both trees, is $k$-balanced. Thus for a tree $\mathcal{T}$, there is a unique balanced tree $\text{bal}_k(\mathcal{T}) = \min\{\mathcal{T}' \succ \mathcal{T} : \mathcal{T}' \text{ is } k\text{-balanced}\}$, which is called the $k$-rebalancing of $\mathcal{T}$.

For instance, 0-balanced quadtrees are what Moore called *smooth* quadtrees [18], and 1-balanced what he called *balanced* and others called 1-*irregular* or *restricted* [8, 18, 25].

*Cost analysis* While the size of $\text{bal}_k(\mathcal{T})$ is known to increase by at most a constant factor from the size of $\mathcal{T}$, the final cost $c(\text{bal}_k(\mathcal{T}))$ is unknown, however. Our main result concerning cost analysis is the following, when objects in $\mathcal{S}$ are either points or segments.

**Theorem 1.** *Let $\mathcal{T}$ be a tree storing both points and/or simplices in the unit cube. Then for any $k$, $0 \leq k < d$, and $d \leq 3$,*

$$c(bal_k(\mathcal{T})) \leq 3^d c(\mathcal{T}).$$

*The result becomes $c(bal_k(\mathcal{T})) = O(d^2 4^d) c(\mathcal{T})$ in higher dimensions.*

The following lemma was first proven by Weiser in 2D, and by Moore for any dimension $d \geq 2$.

**Lemma 8 ( [18]).** *Let $\mathcal{T}$ be a tree. There is a 0-balanced tree $\mathcal{T}' \succ \mathcal{T}$ such that $c_t(bal_0(\mathcal{T})) \leq 3^d c_t(\mathcal{T})$.*

Since $\mathcal{T}'$ must be a refinement of $\mathrm{bal}_0(\mathcal{T})$, which is itself a refinement of $\mathrm{bal}_k(\mathcal{T})$, for any $k > 0$, this implies that $c_t(\mathrm{bal}_k(\mathcal{T})) \leq c_t(\mathrm{bal}_0(\mathcal{T})) \leq 3^d c_t(\mathcal{T})$. Note that the same construction also implies the factor $3^d$ on the number of leaves.

Next we prove in Lemma 9 that the object cost of $\mathrm{bal}_k(\mathcal{T})$ is at most twice (for points) and some constant $B_d < S^d$ (for simplices) times that of $\mathcal{T}$. The proof is omitted for lack of space.

**Lemma 9.** *Let $\mathcal{T}$ be a tree, and consider the object cost of a single object $s \in S$ both in $\mathcal{T}$ and in $bal_k(\mathcal{T})$. If $s$ is a point, then $\lambda_{d-1}(s \cap bal_k(\mathcal{T})) \leq 2\lambda_{d-1}(s \cap \mathcal{T})$. If $s$ is a convex object of dimension at most $d-1$ (e.g. a $(d-1)$-simplex), then $\lambda_{d-1}(s \cap bal_k(\mathcal{T})) \leq B_d \lambda_{d-1}(s \cap \mathcal{T})$.*

Compounding all these costs together, we have $c_o(\mathrm{bal}_k(\mathcal{T})) \leq 3^d c_o(\mathcal{T})$ for any $0 \leq k < d$, which ends the proof of the theorem.

*Remark.* It could also very well be that rebalancing actually decreases the cost. We don't know that, and we don't need it since we are mostly interested in trees for which $c(\mathcal{T}) = O(M)$. In any case, we can ask if there is a reverse theorem (lower bound on $c(\mathrm{bal}_k(\mathcal{T}))$ in terms of $c(\mathcal{T})$).

## 6 Conclusion

In this paper we have proved that instead of considering the optimal octree, and without increasing the cost too much, we may consider the octree given by the lookahead strategy. Still, this may yield an infinite subdivision. In order to have an effective algorithm, we need to add a termination criteria such as a depth limit of $\log_2 n$. As we have also proven, this increases again the cost at most by a constant factor. In practice, we find that greedy with or without lookahead yield near-optimal octrees, hence the approximation ratio seems close to one.[2]

All the results stated in this paper should extend easily to recursive grids and simplicial trees as well, in two and higher dimensions, with only small differences. However, the constants involved in the analysis would be even higher than they are here.

We conclude with a few open problems: first, is it true that by pruning at depth $k = \Theta(\log n)$, we can approach the cost to within $1 + \varepsilon$? Since the optimal tree might be infinite, there is little sense in asking for an algorithm

[2] Actually, they both yield octrees of same cost which are the lowest cost we observe with other heuristics; we find it hard to believe that they would all be $c$ times optimal, for some constant $c > 1$.

that constructs the optimal tree. But if the answer to the first question were true, it would be nice to have a PTAS with respect to the cost measure. We don't know if the greedy strategy for high enough lookahead would fit the bill.

Lastly, the cost measure considered here is simple but does not model the average traversal cost during ray shooting. For this, the following cost measure should be considered [1]:

$$c^*(\mathcal{T}) = \sum_{\sigma} (\gamma + |S \cap \sigma|) \times (\lambda_{d-1}(\sigma) + \lambda_{d-1}(S \cap \sigma)), \tag{2}$$

x where $\lambda_{d-1}(\mathcal{S} \cap \sigma)$ measures the portion of the objects within $\sigma$. Our only result here is that the greedy strategy does not work.

## Acknowledgments

This research was initiated at the McGill-INRIA Workshop on Computational Geometry in Computer Graphics, February 9-15, 2002, co-organized by H. Everett, S. Lazard, and S. Whitesides, and held at the Bellairs Research Institute of McGill University. We thank Robin Chapman of sci.math for providing reference [6]. Many thanks go to Boris Aronov for his very detailed comments, suggestions, and for interesting discussions.

## References

1. B. Aronov, H. Brönnimann, A.Y. Chang, and Y.-J. Chiang. Cost prediction for ray shooting. In *Proc. of Eighteenth ACM Symp. on Geom. Comput.*, pages 293–302, 2002, Barcelona, Spain.
2. B. Aronov, H. Brönnimann, A.Y. Chang, and Y.-J. Chiang. Cost-driven octree construction schemes: an experimental study. In *Proc. of Nineteenth ACM Symp. on Geom. Comput.*, 2003, San Diego, CA.
3. B. Aronov and S. Fortune. Approximating minimum weight triangulations in three dimensions. *Discrete Comput. Geom.*, 21(4):527–549, 1999.
4. B. Aronov and R. Schiffenbauer. Manuscript, 2001.
5. J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A.S. Glassner, editor, *An Introduction to Ray Tracing*, pages 201–262. Morgan Kaufmann Publishers, Inc., 1989.
6. K. Ball. *Cube slicing in* $\mathbb{R}^n$. Proc. Amer. Math. Soc. 97 (1986), no. 3, 465–473.
7. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, Lecture Notes Series on Computing, Volume 4, pages 47–123, World Scientific, Singapore, 1992.
8. Bank, Sherman and Weiser. Refinement algorithms and data structures for regular local mesh refinement. In *IEEE First Symposium on Scientific Computing*, R. Stepleman, editor, North Holland Publishing Company, pages 3–17, 1983.
9. H. Brönnimann, M. Glisse, and D. Wood. Cost-optimal quadtrees for ray shooting. In *Proc. Canad. Conf. on Comput. Geom.* (CCCG'02), Lethbridge, Alberta.
10. F. Cazals and C. Puech. Bucket-like space partitioning data structures with applications to ray tracing. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 11–20, 1997.

11. A.Y. Chang. A survey of geometric data structures for ray tracing. Technical Report TR-CIS-2001-06, CIS Department, Polytechnic University, 2001.
12. J.G. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4:65–83, 1988.
13. A.S. Glassner, editor. *An Introduction to Ray Tracing*. Morgan Kaufmann Publishers, Inc., 1989.
14. J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, pages 14–20, May 1987.
15. V. Havran, J. Bittner, and J. Prikryl. The Best Efficiency Scheme `http://www.cgg.cvut.cz/GOLEM/proposal2.html`.
16. J.D. MacDonald and K.S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6:153–166, 1990.
17. J.S.B. Mitchell, D.M. Mount, and S. Suri. Query-sensitive ray shooting. *Int. J. Comput. Geom. & Appls.*, 7(3):317–347, August 1997.
18. D.W. Moore. *Simplicial Mesh Generation with Applications*. Ph.D dissertation, Cornell University, 1992.
19. D.W. Moore. *The Cost of Balancing Generalized Quadtrees*. In *Proceedings of the Third Symposium on Solid Modeling and Applications (SMA'95)*, pages 305–312, 1995.
20. B. Naylor. Constructing good partitioning trees. In *Proceedings of Graphics Interface '93*, pages 181–191, Toronto, Ontario, may 1993. Canadian Information Processing Society.
21. E. Reinhard, A.J.F. Kok, and A. Chalmers. Cost distribution prediction for parallel ray tracing. In *Second Eurographics Workshop on Parallel Graphics and Visualisation*, pages 77–90. Eurographics, September 1998.
22. E. Reinhard, A.J.F. Kok, and F.W. Jansen. Cost prediction in ray tracing. In P. Hanrahan and W. Purgathofer et al., editors, *Rendering Techniques '97*, pages 42–51. Porto, Portugal, 1996.
23. H. Samet. *Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
24. K.R. Subramanian and D.S. Fussell. Automatic termination criteria for ray tracing hierarchies. In *Proc. of Graphics Interface '91*, June 3-7 1991.
25. Von Herzen and A. Barr. Accurate triangulations of deformed intersecting surfaces. In *Proceedings of 21st Annual ACM Symposium on Computer Graphics* (SIGGRAPH'87), pages 103–110, 1987.
26. H. Weghorst, G. Hooper, and D.P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984.
27. A. Weiser. *Local-Mesh, Local-Order, Adaptive Finite Element Methods with a Posteriori Error Estimators for Elliptic Partial Differential Equations*. Tech. Rep. 213, Dept. Computer Science, Yale University, New Haven, 1981.
28. K. Y. Whang, J. W. Song, J. W. Chang, J. Y. Kim, W. S. Choand, C. M. Park, and I. Y. Song. Octree-R: An adaptive octree for efficient ray tracing. *IEEE Trans. Visual and Comp. Graphics*, 1:343–349, 1995.

## Appendix

In this appendix, provided only for the reviewers, we include the missing proofs so that reviewers can examine the truths of our statements if they desire. We intend it only for the consideration of the reviewers and not as part of the submission.

## A Examples of cost computation

As examples, we compute the cost of the same configurations as [9], but in any dimension $d \geq 2$.

**0.** With no subdivision, the cost of the unit cell $\mathcal{T}^{(\text{empty})}$ is $c(\mathcal{T}^{(\text{empty})}) = 2d(\gamma + n)$.

**1.** For points, the cost of a full subdivision at depth $k$, $\mathcal{T}_k^{(\text{complete})}$, is at least $2d(2^k\gamma + \frac{n}{2^{k(d-1)}})$ (this is the exact value if all the points fall in a single leaf of $\mathcal{T}_k^{(\text{complete})}$; this happens if the binary expansions of all point coordinates always have at least $k+1$ bits), and at most $2d(2^k\gamma + \frac{2^d n}{2^{k(d-1)}})$ (if each point belongs to $2^d$ leaves).

**2.** As an exercise, the tree costs of the $k$-side, $k$-corner, and $k$-border, and more generally of the $k$-corner of order $h$ can be computed readily by noting that a $k$-side is a half-scaled copy of $2^{d-1}$ empty trees and $2^{d-1}$ $(k-1)$-sides, that a corner is made of $\binom{d}{i}$ half-scaled $k$-corners of order $i$, for every $0 \leq i \leq d-1$, and that a $k$-border is made of $2^d$ half-scaled $(k-1)$-corners: **3.** Let $S_n$ denote $n$ distinct points very close to one corner of the unit cell, let's say the origin. After $k$ levels of recursively subdividing the incident cell,[3] we obtain the tree $\mathcal{T}_k^{(\text{sep})}(S_n)$ whose cost is $c(\mathcal{Q}_k^{(\text{sep})}(S_n)) = 12\gamma + (n - 2\gamma)2^{2-k}$. For any $d$, the cost is

$$c(\mathcal{T}_k^{(\text{sep})}(S_n)) = \frac{2dn}{2^{(d-1)k}} + 2d\gamma\left(1 + \frac{1 - 2^{(1-d)k}}{1 - 2^{1-d}}\right)$$

Whether this is an improvement over $\mathcal{T}^{(\text{empty})}$ for any value of $k$ depends on $n$ and $\gamma$. In particular, $\mathcal{T}_k^{(\text{sep})}(S_n)$ has cost lower than the unsubdivided tree for large values of $k$ only if $n > 2\gamma$ in 2D, and $n > \frac{4}{3}\gamma$ in 3D ($n > \frac{2^{d-1}}{2^{d-1}-1}\gamma$ in general).

This example tells us that whether subdivision strategies based on the number of objects in a cell—like the separation criterion—produce optimal or near-optimal trees, depends strongly on the value of $\gamma$.

[3] Here, 'very close' means always within the cell incident to that vertex. It's then pointless to subdivide the other cells: since they do not contain points of $S$, their cost would be doubled.

## B Pruning (Lemmas 2 and 3)

**Lemma 10.** *Let $\mathcal{T}$ be a $d$-dimensional tree which stores a set $S$ of $n$ convex objects of dimensions not more than $d-1$. For $k = \log_2 n + C$, let $\mathcal{T}_k$ be the tree obtained from $\mathcal{T}$ by removing every cell of depth greater than $k$. Then $c(\mathcal{T}_k) = O(c(\mathcal{T}))$ and the constant does not depend on $n$ nor $S$.*

**Proof.** First, the tree cost will only increase while further subdividing, so that $c_t(\mathcal{T}_k) \leq c_t(\mathcal{T})$. The cells of depth less than $k$ are the same in $\mathcal{T}$ and $\mathcal{T}_k$, hence the object cost of $\mathcal{T}$ is an upper bound for all the cells of $\mathcal{T}_k$ with depth less than $k$. It remains to bound the object cost of the leaves of $\mathcal{T}_k$ at depth $k$.

Let us consider an arbitrary object, and since we are only concerned with depth $k$, let us consider the full subdivision of depth $k$ instead of $\mathcal{T}_k$. We let $K$ be $2^k$. This is a $d$-dimensional grid of side $K$, with $K^d$ cells. Our first purpose is to give an upper bound on the number of cells of this grid that the object can intersect. Since the dimension of the object is at most $d-1$, the object belongs to a hyperplane: consider the coordinate of largest absolute value of the normal vector to this hyperplane, and project both the object and the grid along the corresponding direction. By our choice of projection, each $(d-1)$-cell intersected by the projection of the object is the projection to at most $d+1$ $d$-cells intersected by the objects. The $(d-2)$-dimensional boundary of the projection is also convex, and may intersect at most $K^{d-1} - (K-2)^{d-1} \leq 2dK^{d-2}$ cells (by convexity, the worst case occurs when the boundary is largest possible, hence intersects all the border cells of the grid). The $(d-1)$-dimensional relative interior cannot contain more than $K^{d-1} \cdot \lambda_{d-1}(s)$ inner cells. The total number of cells met by the object is then at most $(d+1) \cdot \left(2dK^{d-2} + K^{d-1}\lambda_{d-1}(s)\right)$.

Now, the measure of a cell is $2dK^{1-d}$. The cost of the object associated to depth $k$ cells is then at most $2d(d+1)\left(2d/K + \lambda_{d-1}(s)\right)$. Summing over all objects gives a total of at most $2d(d+1)\left(2dn/K + \sum_s \lambda_{d-1}(s)\right)$. Substituting $2^{\log_2 n + C}$ for $K$ shows that the combined object costs of the leaves of $\mathcal{T}_k$ at depth $k$ is at most $2d(d+1)\left(d \cdot 2^{1-C} + \sum_s \lambda_{d-1}(s)\right)$. By Lemma 1, this is at most $\max\left\{\sqrt{2}(d+1), d(d+1)2^{1-C}/\gamma\right\}$ times the cost of $\mathcal{T}$. Putting everything together, $c(\mathcal{T}_k) \leq \left(1 + \sqrt{2}(d+1) + d(d+1)2^{1-C}/\gamma\right) c(\mathcal{T})$. ∎

**Lemma 11.** *Let $\mathcal{T}$ be a $d$-dimensional tree, which stores a set $S$ of $n$ simplices. Assume that $\mathcal{T}$ does not contain empty internal nodes (i.e. that are subdvided but do not contain any object). Let $\mathcal{T}_k$ be the tree obtained from $\mathcal{T}$ by removing every cell of depth greater than $k$. Then, for every $\varepsilon > 0$ there exists a $C$ (that depends only on $\varepsilon$ and $\gamma$ but not on $n$) such that, for $k = \frac{1}{d-1}\log_2 n + C$, we have*

$$c(\mathcal{T}_k) \leq (1+\varepsilon)c(\mathcal{T}).$$

**Proof.** The cost of a cell $\sigma$ which contains $n_\sigma$ points and has depth $k = \frac{1}{d-1}(\log_2 n + C)$ is

$$c(\sigma) = (\gamma + n_\sigma)2d \cdot 2^{-k(d-1)} = \frac{2d}{2^C n}(\gamma + n_\sigma).$$

The proof for points hinges on the fact that, unlike an arbitrary simplex, a point belongs to at most $2^d$ leaves, and that to be subdivided, a cell needs to contain at least one point. Hence, there are at most $2^d n$ leaves in $\mathcal{T}_k$ which contain a point, and $\sum_\sigma n_\sigma \leq 2^d n$. Summing over all leaves at depth $k$ which still contain points, one gets

$$\sum_{\sigma \in \mathcal{L}_k(\mathcal{T})} c(\sigma) \leq \frac{2d}{n}(\gamma + n_\sigma) \cdot \frac{d2^{d+1}(\gamma+1)}{2^C},$$

which can be made as small as $\varepsilon 2d\gamma$ for an appropriate value of $K$. By Lemma 1, this implies that the cost of the non-empty leaves at depth $k$ in $\mathcal{T}_k$ is at most $\varepsilon c(\mathcal{T})$. The leaves of $\mathcal{T}_k$ at depth less than $k$ also belong to $\mathcal{T}$, and the same holds as well for the leaves at depth $k$ which do not contain any point. Hence their total cost is at most $c(\mathcal{T})$ and the lemma follows for points. ∎

## C *p*-Greedy for points

**Lemma 12.** *The 3-greedy strategy in the plane, and the 2-greedy strategy in $d$ dimensions ($d \geq 3$) produce near-optimal trees for points. Namely, if $S$ be a set of $n$ points in the unit $d$-cube, and $M$ is the infimum of $c(\mathcal{T})$ over all trees $\mathcal{T}$, then $c(\mathcal{T}^{(\text{3-greedy})}) = O(M)$ for all $d \geq 2$, and $c(\mathcal{T}^{(\text{2-greedy})}) = O(M)$ for all $d \geq 3$.*

**Proof.** We prove that 3-greedy is near-optimal, for $d \geq 2$. Then we indicate what changes for 2-greedy when $d \geq 3$. The cost of a cell $\sigma$ is $(\gamma + |S \cap \sigma|)\lambda_1(\sigma)$. The cost of a complete subdivision $\sigma_3^{(\text{complete})}$ of $\sigma$ to depth 3 is $(8\gamma + \frac{1}{8^{d-1}} \sum_{p \in S \cap \sigma} n_p)\lambda_{d-1}(\sigma)$, where $n_p$ is the number of leaves of that subdivision that contain $p$. Since $n_p$ is at most $2^d$, the latter cost is at most $8\gamma + |S \cap \sigma| 2^{3-2d})\lambda_{d-1}(\sigma)$. If $\sigma$ is a leaf of $\mathcal{T}^{(\text{3-greedy})}$, this means that $\sigma_3^{(\text{opt})}$ is just the leaf $\sigma$, then certainly $c(\sigma_3^{(\text{complete})})$ is greater than $c(\sigma)$. Hence $8\gamma + |S \cap \sigma| 2^{3-2d} \geq \gamma + |S \cap \sigma|$, i.e. $|S \cap \sigma| = \frac{7}{1-2^{3-2d}}\gamma = C_d$. The only case when the 3-greedy strategy fails, is when the lookahead says it is wrong to subdivide whereas it is not. In that case, the number of points is at most $C_d\gamma$. Then the cost is at most $(1 + C_d)\gamma\lambda_{d-1}(\sigma)$, and because of Lemma 1 this is $O(M_\sigma)$, where $M_\sigma$ is the infimum cost of all the possible quadtree subdivisions of $\sigma$. Since lookahead only subdivides a cell if it has a subtree that actually improves the cost, necessarily $\mathcal{T}^{(\text{3-greedy})} \prec \mathcal{T}^{(\text{opt})}$ (where $\mathcal{T}^{(\text{opt})}$ is any optimal[4] tree). Thus every leaf $\sigma$ of $\mathcal{T}^{(\text{3-greedy})}$ is a cell of $\mathcal{T}^{(\text{opt})}$, and since $c(\sigma) = O(M_\sigma)$ then $c(\mathcal{T}^{(\text{3-greedy})}) = O(\mathcal{T}^{(\text{opt})}) = O(M)$ as well.

[4] The cost $M$ is an infimum and may not be achieved by any finite-depth tree. But for this proof, $\mathcal{T}^{(\text{opt})}$ can be allowed to have infinite depth. Alternately, one may consider an increasing sequence of trees $\mathcal{T}_k^{(\text{opt})}$ whose costs converge towards $M$.

Redoing the computation with lookahead $k = 2$ for $d \geq 3$, we find that $C_d = \frac{7}{1-2^{2-d}}$, and the rest of the proof follows similarly. ∎

On the example given in that section, the separation criterion is near optimal. In fact, Aronov and Schiffenbauer (private communication) have also and independently proven that $\mathcal{Q}^{(\text{AS-sep})}(S)$ is near-optimal, where $\mathcal{Q}^{(\text{AS-sep})}$ uses a slightly different separation criterion: subdivide a cell whenever it covers more than one point of $S$, and also if any of its orthogonal (resp. diagonal) neighbors has depth which differs by more than 1 (resp. 2). Note that their construction is valid only if all the points in $S$ are distinct.

**Lemma 13.** ***(Aronov,Schiffenbauer)*** *Let $S$ be a set of $n$ points in the unit square, and let $M$ be the infimum of $c(\mathcal{Q})$ over all quadtrees $\mathcal{Q}$. Then $\mathcal{Q}^{(\text{AS-sep})}(S)$ is near-optimal, namely, $c(\mathcal{Q}^{(\text{AS-sep})}(S)) = O(M)$.*

## D Rebalancing

The proof of the lemma 8 uses the concept of *barrier* [18]. A barrier (depicted is a configuration of cells around a designated cell that guarantees that the designated cell never has to split to achieve balance. See Figure 2 for an example with a given quadtree. It is not hard to see that $\mathcal{T}'$ contains the barrier of every leaf of $\mathcal{T}$, and that the number of leaves of $\mathcal{T}'$ is at most $3^d$ that of $\mathcal{T}$.

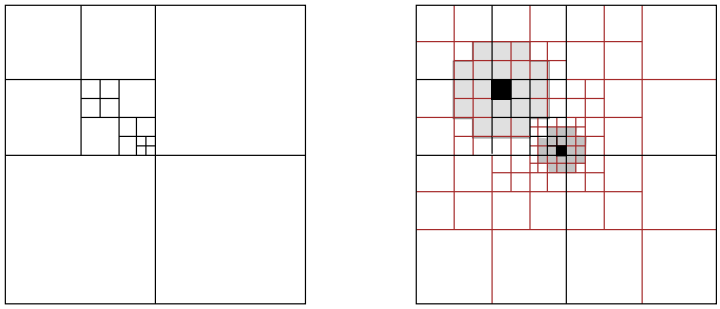

**Fig. 2.** Illustration of Lemma 8: A quadtree $\mathcal{T}$ (left) and its corresponding $\mathcal{T}'$ (right).

**Lemma 14.** *Let $\mathcal{T}$ be a tree, and consider the object cost of a single object $s \in S$ both in $\mathcal{T}$ and in $bal_k(\mathcal{T})$. If $s$ is a point, then $\lambda_{d-1}(s \cap bal_k(\mathcal{T})) \leq 2\lambda_{d-1}(s \cap \mathcal{T})$. If $s$ is a convex object of dimension at most $d-1$ (e.g. a $(d-1)$-simplex), then $\lambda_{d-1}(s \cap bal_k(\mathcal{T})) \leq B_d \lambda_{d-1}(s \cap \mathcal{T})$.*

**Proof.** For a point, the worst case is when it falls exactly on the center of a cell: it then belongs to all the children of the subdivided cell, and thus its contribution to the object cost is at most doubled. Note that further subdivision will never increase that contribution.

The situation is more complicated for simplices, but we can prove that the contribution of an object within a leaf $\sigma$ of $\mathcal{T}$ is never increased by more than a constant factor. The proof relies in that, when rebalancing, $\sigma$ is only subdivided along its boundary, because only its neighbors can force it to subdivide. Thus, the subtree $\sigma'$ of $\sigma$ in $\mathrm{bal}_k(\mathcal{T})$ must satisfy $\sigma' \prec \sigma_\infty$, where $\sigma_\infty$ is the maximum subdivision of $\sigma$ along its boundary (an infinite border). By the following lemma and appropriate scaling, the object cost $\lambda_{d-1}(s \cap \mathrm{bal}_k(\mathcal{T}))$ is at most $B_d \lambda_{d-1}(s \cap \mathcal{T})$. ∎

**Lemma 15.** *The object cost of a flat object in any tree $\mathcal{T} \prec \mathcal{T}_\infty^{(\mathrm{border})}$ is bounded by a constant $2dB_d$. Moreover, one can choose $B_d \leq 3^d$ for $d \leq 3$.*

**Proof.** We first prove the lemma for sides, then for borders. Let $d$ be fixed, and let $s_k$ (resp. $c_k$, $b_k$) be the maximum object cost of a simplex in $\mathcal{T}_k^{(\mathrm{side})}$ (resp. $\mathcal{T}_k^{(\mathrm{corner})}$, $\mathcal{T}_k^{(\mathrm{border})}$).

By construction, a $k$-side is made up of $2^{d-1}$ half-scaled empty cells, $2^{2(d-1)}$ quarter-scale empty cells, and $2^{2(d-1)}$ quarter-scale $(k-2)$-sides. If the simplex intersects all the $(k-2)$-sides, then it cannot intersect the half-scaled empty cells, and it thus intersect only $2^{d-1}$ half-scale $(k-1)$-sides, thus $s_k \leq s_{k-1}$ (the factor $2^{d-1}$ disappears because of scaling). On the other hand, if it misses any of the $(k-2)$-sides, then the object cost decreases geometrically (but it may intersect all the empty cells, of total measure $4d$). More explicitly,

$$s_k \leq \max\left(s_{k-1}, 4d + \frac{2^{2d-2}-1}{2^{2d-2}} s_{k-2}\right).$$

This recurrence shows that $s_k$ is bounded by $S_d = d4^d$ for all $k$.

We can then consider a border as the union of $2d2^{d-1}$ half-scaled sides (the superpositions only increase the cost), so that the cost is at most $d^2 2^{2d-1}$.

The constant $B_d$ thus obtained is larger than $3^d$. A more careful case analysis of a depth three subdivision is too long to include here, but would show that $B_d$ can indeed be chosen less than $3^d$, at least for $d \leq 3$. The conjecture is $B_d = 2^{d-1}$. ∎

*Remarks. 1* The constant $3^d$ in Lemma 8 is tight for the number of leaves, but not for the cost of 0-balanced trees. When considering $k$-balanced trees, $k \geq 1$, it may be overly pessimistic. Indeed Moore shows that in the plane, 9 is tight for 0-balanced quadtrees, but 8 is tight for 1-balanced quadtrees.

*2* Aronov and Schiffenbauer independently proved that for any quadtree $\mathcal{Q}$ (not necessarily balanced), there is a balanced quadtree $\mathcal{Q}'$ such that $c(\mathcal{Q}') = O(c(\mathcal{Q}))$Their construction may subdivide more than necessary.