Octrees with near optimal cost for ray-shooting [1,3,4]

Hervé Brönnimann [a] and Marc Glisse [b]

[a] *Computer and Information Science Department, Polytechnic University, Brooklyn, NY 11201 USA; hbr@poly.edu.*

[b] *INRIA Lorraine - Loria , 615, rue du Jardin Botanique, B.P. 101 54602 Villers-lès-Nancy cedex, FRANCE; glisse@loria.fr.*

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 and coll. [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 on octrees with the same cost measure as the optimizing criterion. More generally, we solve the generalization for generalized octrees in any d dimensions with scenes made up of $(d-1)$-dimensional simplices. We give a construction of trees which yields cost $O(M)$, where M is the infimum of the cost measure on all trees. 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. These are the first and only known results that provide performance guarantees on the approximation factor for 3-dimensional ray shooting with this realistic model of computation. Our results have been validated experimentally by Aronov and coll. [2].

Key words: Ray shooting, cost model, cost prediction, average performance, octree, space decomposition, optimization

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 [15]), radio- and wave-propagation simulation [8], 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 or an octree in 3D. The query is then 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. Practioners use a host of heuristics and parameters of the scene, of which the object count is less important than, e.g., 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. This cost measure is a simplification of a more complicated but theoretically sound cost measure: under certain assumptions on the ray distribution, the more complicated cost measure provably reflects the cost of shooting an average ray using the space decomposition. The simpli-

[1] Research of the first author supported by NSF ITR Grant CCR-0081964 and in part by NSF CAREER Grant CCR-0133599.
[2] Research of the second author was performed during an internship at Polytechnic University, with partial support from Ecole Normale Supérieure, Paris, France.
[3] 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.
[4] A preliminary version of this paper authors appeared from the same authors under the title "Cost-optimal trees for ray shooting" in: Proc. LATIN 2004, LNCS Vol. 2796, p. 249-258, Springer, New York, 2004.

fied cost measure has a very similar predicting power for many scenes encountered in computer graphics. This has been experimentally verified [1, 2]. We conjecture it would take a very artificially constructed and unrealistic scene to bring forth a discrepancy. It thus makes sense to try and optimize the data structure with respect to this simplified cost measure.

In this paper, we are interested in constructing trees with cost as low as possible, with a guaranteed approximation ratio. First, we observe that tree construction heuristics used in computer graphics do not have a bounded approximation ratio. We give and analyze algorithms that produce trees with cost $O(M)$, where M is a lower bound on the cost of any tree for the given scene. 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. 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 $\mathbb{R}^3$, and $(d-1)$-simplices in $\mathbb{R}^d$. We however assume the real-RAM model so as to avoid a discussion on the bit-length of the coordinates. (This is also justified by our application.)

In a follow-up paper [2] to [1], we evaluate empirically several heuristics, including those presented here, to optimize the cost value of an octree for a given scene. Both our algorithm (*3-greedy* in this paper's terminology) and a simpler heuristic (which we call *1-greedy* or *greedy without lookahead*) give the best cost. In fact, we find that all the reasonable variants end up with approximately the same cost which must be within a factor $O(1)$ of optimal. There is no guarantee that this cost is within $(1+\varepsilon)$ of optimal, though. The polynomial-time algorithm suggested in the remark of Section 3.4 (full subdivision to depth $\log_2 n + C$ followed by dynamic programming) would be guaranteed to give a $O(1)$ approximation factor as well, and we'd even expect it to be very close to optimal, but it was too time- and space-consuming for us to try in [2]. Nevertheless, we find it hard to believe that the cost to which all these variants converge could be sub-optimal, and it thus appears experimentally that our procedure yields a near-optimal tree in practice.

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 [25], the thesis of Moore [20], 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, however, a rich history of data-structure optimization for ray shooting in computer graphics. (See the surveys in [1, 2, 12] and refs. therein.) Cost

measures have been proposed for ray shooting in octrees by MacDonald and Booth [18], Reinhard and coll. [24], Whang and coll. [31], and for other structures, such as bounding volume hierarchies [16, 27, 29], bintrees [18], BSP-trees [22], uniform grids [13] and hierarchical uniform grids [11]. We should also mention the work of Havran and coll. [17], 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 criteria (sometimes very effectively) but none offer theoretical guarantees. In particular, the cost function c of MacDonald and Booth is the same as ours (see Section 2.1 below) and their greedy procedure almost identical to our 1-greedy heuristic. They do not however derive the theoretically correct formulation c^* nor establish the connection to c, but they do explore a wider parameter space by allowing the subdivision of their bintree to be different than the spatial median (which we do not in this work).

2 General cost measure results

2.1 Motivation, definitions, and problem statement

In this paper we consider the problem of shooting rays into a scene consisting of solids, represented by their boundaries. In particular, we assume that these boundaries have already been subdivided into elements of constant combinatorial complexity, so that the only objects we consider are simplices of dimension at most $d-1$ in $\mathbb{R}^d$: points and segments inside the unit square $[0,1]^2$ in $\mathbb{R}^2$, points, segments and triangles inside the unit cube $[0,1]^3$ in $\mathbb{R}^3$, and $(d-1)$-simplices in $\mathbb{R}^d$.

The algorithm generally used for ray shooting with the help of a (convex) spatial subdivision $\mathcal{T}$ such as a quadtree (in 2D) or an octree (in 3D) is the following: when shooting a ray, the cells of that subdivision (in our context, the leaves of the octree) are traversed in the order they are encountered by the ray, starting with the cell that contains the origin of the ray. For each cell traversed, all the objects intersecting that cell are tested against the ray to find the first hit. If a hit is found inside the cell, then the traversal is stopped and the first such hit is returned, otherwise the algorithm proceeds to the next cell in the traversal. If it reaches the outside boundary of the enclosing volume, the ray is declared unoccluded and the algorithm returns "no hit".

The worst-case cost of this algorithm depends on the stabbing number of the subdivision. While it is not hard to fabricate scenes with a worst-case linear

time ray, in a typical application of ray shooting such as ray tracing or wave propagation simulation, many rays are shot. The quantity of interest is really the average cost for a random ray following a certain distribution of rays. A common heuristic in computer graphics consists of approaching the density of rays hitting a convex cell by its perimeter area, as motivated by Crofton's formula in integral geometry [26]. In fact, this is rigorous for line traversal if the line distribution is the rigid-motion invariant distribution, but this is ill-defined for rays. In [1], we observed that for a uniform distribution of rays originating either outside the subdivision or on the surface of objects, the density of rays traversing a cell was proportional to the area of the perimeter plus the total area of the perimeter of the objects contained in the cell. The average cost of traversing a cell is proportional to the number of intersection tests performed multiplied by the density, plus some overhead γ for finding the next cell in the traversal, [5] and thus the following cost measure should be considered for modeling the total traversal costs [1]: [6]

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

where $\mathcal{L}(\mathcal{T})$ is the set of leaves of the tree, S_σ is the set of scene objects intersecting a leaf σ, $\lambda_{d-1}(\sigma)$ is the perimeter length (if $d = 2$) or surface area (if $d = 3$) of σ, and $\lambda_{d-1}(S_\sigma \cap \sigma)$ is the perimeter or area of the portion contained in σ of the objects in S_σ. The last term $\lambda_{d-1}(S_\sigma \cap \sigma)$ is somewhat complicated to evaluate, and so the following simplification was introduced in [1]:

$$c_S(\mathcal{T}) := \sum_{\sigma \in \mathcal{L}(\mathcal{T})} \Big(\gamma + |S_\sigma|\Big) \times \lambda_{d-1}(\sigma). \tag{2}$$

This simplified 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), and was already proposed earlier by MacDonald and Booth [18] but without the connection to c_S^*. MacDonald and Booth [18] (for bintrees) and Aronov and coll. [1,2] (for bounded-degree spatial subdivisions, with extensive experimentation for octree) argue and provide ample experimental evidence that this cost function also reflects the average cost of ray shooting using the spatial subdivision induced by the leaves of $\mathcal{T}$.

[5] Technically speaking, the cost functions make sense only for spatial subdivisions with bounded degree, so that the cost of finding the next cell in the traversal can be bounded by a constant γ. But this is also true in an amortized sense for octrees, see [1] for the technical details.

[6] Strictly speaking, the average cost of shooting a ray is $c_S^*(\mathcal{T})/\mu$ where μ is the total density of the ray distribution, i.e. the surface area of the outer perimeter of $\mathcal{T}$ plus the total surface area of the objects in S. For the purpose of optimizing ray shooting costs, however, if the outer bounding box and the scene are given, the denominator is constant and we need only concentrate on finding the best $\mathcal{T}$ optimizing $c_S^*(\mathcal{T})$ or $c_S(\mathcal{T})$.

The problem we then consider is, given a scene S, an overall bounding box $\mathcal{B}$, and a cost function such as Eq. (2), to construct an octree $\mathcal{T}$ subdividing $\mathcal{B}$ which minimizes $c_S(\mathcal{T})$. Note that all the definitions as phrased above are valid for $d \geq 4$ as well as for 2D and 3D, and for the sake of generality we consider below the problem for any *fixed* $d \geq 2$. Since S and $\mathcal{B}$ are also fixed, we will often omit them in the notation and use simply $c(\mathcal{T})$. In order to obtain theoretical guarantees, we restrain ourselves to split the octree nodes at their spatial median, although others have investigated other choices to heuristically improve costs. Even with this, we do not know if our problem is NP-complete or not. Nevertheless, we are not aware of any work providing optimality or even any sublinear bound on the approximation ratio. Namely, all the known termination criteria for subdividing an octree or other constructions mentioned earlier have no known sublinear bound on their approximation ratio.

2.2 Tree and object costs

Observe that the cost measure expressed in Eq. (2) can be decomposed into two terms: $c(\mathcal{T}) = c_t(\mathcal{T}) + c_o(\mathcal{T})$. The first term,

$$c_t(\mathcal{T}) := \gamma \sum_{\sigma \in \mathcal{L}(\mathcal{T})} \lambda_{d-1}(\sigma) = \gamma \lambda_{d-1}(\mathcal{L}(\mathcal{T})), \tag{3}$$

where $\mathcal{L}(\mathcal{T})$ denotes the set of leaves of $\mathcal{T}$ and λ_{d-1} is extended to sets of leaves by summation, is simply the tree total area (which is more than the surface area of the bounding box). We call c_t the *tree cost*. The second term $\sum_{\sigma \in \mathcal{L}(\mathcal{T})} |S_\sigma| \lambda_{d-1}(\sigma)$, can be computed object by object and this leads to

$$c_o(\mathcal{T}) := \sum_{s \in S} \lambda_{d-1}(\mathcal{L}_s(\mathcal{T})), \tag{4}$$

where $\mathcal{L}_s(\mathcal{T})$ denote the set of leaves of $\mathcal{T}$ intersected by s, and again λ_{d-1} is extended to sets of leaves by summation. We call c_o the *object cost*. Again, note that $\lambda_{d-1}(\mathcal{L}_s(\mathcal{T}))$ is more than the perimeter of the union $\cup_{s \in S} \mathcal{L}_s(\mathcal{T})$, as it measures the sum of the perimeters of all the leaves intersected by the object. (In particular, both sides of a face contribute if the face belongs to the interior of $\cup_{s \in S} \mathcal{L}_s(\mathcal{T})$.) It is useful to keep in mind the following simple observations: when subdividing a leaf σ, the total tree cost of its children is twice the tree cost of σ, and the contribution within σ to the object cost of an object is multiplied by $m/2^{d-1}$ where $m \in [1 \dots 2^d]$ is the number of children intersected by the object. Note that $m \leq 3$ for a segment in 2D and $m \leq 7$ 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.

The following lemma was given in [9] for quadtrees, and rephrased here to encompass any dimension as well.

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

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 $\sum_{s\in S}\lambda_{d-1}(s)$. We can improve this lower bound further by noting that any leaf σ 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 have said so far is independent of the particular termination criterion or algorithm used to construct the tree. In this section, we introduce several construction schemes and explore their basic properties.

3.1 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 *unit* and denote it by $\mathcal{T}^{(\text{unit})}$. 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 intervals along each direction. Note that the root has depth 0 and its children have depth 1. In a tree, 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 3.1 for an illustration of the 2D case. In higher dimensions, there are other cases (one for each dimension between 1 and $d-2$). Generally speaking, we call the tree obtained by recursively subdividing until depth k all the cells touching the facets adjacent to given h-face a *k-corner of order h* (in dimension d), and denote it $\mathcal{T}_k^{(h,\,d\text{-corner})}$. A k-corner is of a corner of order 0, and a k-side is a corner of order $d-1$. All this notation is extended to starting from a cell

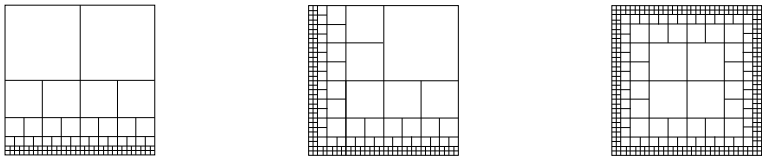

Fig. 1. The k-side quadtree $\mathcal{Q}_k^{(\text{side})}$ (left), a corner $\mathcal{Q}_k^{(\text{corner})}$ (center), and a border $\mathcal{Q}_k^{(\text{border})}$ (right).

σ instead of a unit tree, by substituting σ for $\mathcal{T}$: for instance, the complete subtree of depth k subdividing σ is denoted by $\sigma_k^{(\text{complete})}$.

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

We consider algorithms for computing a tree for a given set S of objects, which subdivide each cell recursively until some given termination criterion is satisfied. In particular, we may recursively subdivide the unit cube until each leaf meets at most one object (or any fixed C). 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 $\mathcal{T}_k^{(\text{sep})}(S)$, or $C > 1$ denoted $\mathcal{T}_C^{(\text{sep})}(S)$. (Note that the depth of $\mathcal{T}_C^{(\text{sep})}$ is always infinite if more than C 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.

3.2 Examples of cost computation

As examples and for completeness, we compute the costs of some of the configurations given above.

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

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

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 $0 \le i \le d-1$, and that

a k-border is made of 2^d half-scaled $(k-1)$-corners:

$$\begin{aligned} c_t(\mathcal{T}_k^{(\text{side})}) &= 2d\gamma(k+1), \\ c_t(\mathcal{T}_k^{(\text{corner})}) &= \frac{2d\gamma}{(2^{d-1}-1)^2} \times \left(k\left(2^{2d-1} - 2^{1+d} + 2\right) - 2^{d-1} + 1 + \frac{2^{d-1}-1}{2^{k(d-1)}} \right), \\ c_t(\mathcal{T}_k^{(\text{border})}) &= 2c_t(\mathcal{T}_{k-1}^{(\text{corner})}) \end{aligned}$$

Let S_n denote n distinct points very close to one corner of the unit cell, let's say the origin. 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. After k levels of recursively subdividing the incident cell, we obtain the tree $\mathcal{T}_k^{(\text{sep})}(S_n)$ whose cost is $c(\mathcal{T}_k^{(\text{sep})}(S_n)) = 12\gamma + (n-2\gamma)2^{2-k}$ for $d=2$; 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 $\mathcal{T}^{(\text{unit})}$ for any value of k depends on n and γ. In particular, $\mathcal{T}_k^{(\text{sep})}(S_n)$ has cost lower than the unit 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 γ.

3.3 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 σ 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 σ, we run the dynamic programming within σ with depth p (p is a parameter called *lookahead*). If the best sub-

tree $\sigma_p^{(\mathrm{opt})}(S)$ does not improve the cost of the unsubdivided node σ, then the recursion terminates. Otherwise, we replace σ by the subtree $\sigma_p^{(\mathrm{opt})}(S)$ and recursively evaluate the criterion for the leaves of $\sigma_p^{(\mathrm{opt})}(S)$. We call this the p-greedy strategy and denote the resulting tree by $\mathcal{T}^{(p\text{-greedy})}(S)$ (or $\sigma^{(p\text{-greedy})}(S)$ if we start from a cell σ instead of the unit cube). Note that unlike all the other trees constructed up to now, that tree could be infinite. We use the notation $\mathcal{T}_k^{(p\text{-greedy})}$ to denote the tree constructed with the p-greedy lookahead criterion combined with a maximum depth of k.

With one level of lookahead ($p = 1$), the *greedy strategy* simply examines whether subdivision at one level decreases the cost measure. Below, we show that this does not always yield good trees. We will analyze the greedy strategies with one or more levels of lookahead, first for points, then for simplices. But first, we must grapple with the issue of infinite depth.

3.4 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. Indeed, this is the case for $n > 2\gamma$ in the example of $\mathcal{T}_k^{(\mathrm{sep})}(S_n)$ given at the end of section 3.2. So we let M denote the infimum of $c(\mathcal{T})$ over all trees $\mathcal{T}$ for a scene S. (As a consequence of Lemma 1, $M \geq 2d\gamma$.) In order to have an algorithm that terminates, we usually add an extra termination criterion such as a maximum depth k.

We now show that pruning a tree beyond depth k for some choice of k increases the 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.

Lemma 2 *Let $\mathcal{T}$ be a d-dimensional tree which stores a set S of n convex objects of dimensions at most $d-1$. For any fixed constant $C \geq 0$ and $k = \log_2 n + C$, let $\mathcal{T}_k$ be the tree obtained from $\mathcal{T}$ by removing every cell of depth greater than k. Then $c(\mathcal{T}_k) = O(c(\mathcal{T}))$ with a constant depending on d, C and γ (not on S nor n).*

Proof. First, the tree cost only increases when 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 those cells of $\mathcal{T}_k$ is at most the object cost of $\mathcal{T}$. It remains to bound the object cost of the leaves of $\mathcal{T}_k$ that have 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 $\mathcal{G}$ 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. We could not find a bound in the literature, so we include it here for completeness:

Lemma 3 *Let $\mathcal{G}$ be a $K \times \cdots \times K$ d-dimensional grid subdividing the unit cube, with K^d cells. Any convex object s of dimension at most $d-1$ intersects at most $(d+1) \cdot \left(4dK^{d-2} + K^{d-1}\lambda_{d-1}(s)\right)$ cells of $\mathcal{G}$.*

Proof of Lemma 3. Since the dimension of s is at most $d-1$, it belongs to a hyperplane π: consider the coordinate n_i of largest absolute value of a vector $\mathbf{n}$ normal to π, and project the object s, the vector $\mathbf{n}$, and the grid $\mathcal{G}$ onto a $(d-1)$-dimensional object s', vector $\mathbf{n}'$, and grid $\mathcal{G}'$ in the hyperplane $x_i = 0$. Note that $\lambda_{d-1}(s') \leq \lambda_{d-1}(s)$. By our choice of projection, each cell of $\mathcal{G}'$ intersected by s' is the projection of at most $d+1$ cells of $\mathcal{G}$ intersected by s. This is most easily seen by rewriting the equation of π as $x_i n_i + \mathbf{x}'\mathbf{n}' = \pi_0$, or $\Delta x_i n_i + \Delta\mathbf{x}'\mathbf{n}' = 0$, which means that $|\Delta\mathbf{x_i}| \cdot |n_i| \leq |\Delta\mathbf{x}' \cdot \mathbf{n}'| \leq \sum_{j\neq i} |\Delta x_j| \cdot |n_j| \leq |n_i| \sum_{j\neq i} |\Delta x_j|$, and thus when $\Delta\mathbf{x}'$ is in a cell of $\mathcal{G}'$, $|\Delta x_i|$ is at most $d-1$; in the worst case, this can span $d+1$ cells (the bottom and top cells being only touched by a single point). The $(d-2)$-dimensional boundary of s' is also convex, and may intersect at most $K^{d-1} - (K-4)^{d-1} \leq 4dK^{d-2}$ cells (by convexity, the worst case occurs when the boundary is largest possible, hence intersects all the border cells of $\mathcal{G}'$ and their neighbors). The relative interior of s' cannot contain more than $K^{d-1} \cdot \lambda_{d-1}(s')$ cells of $\mathcal{G}'$, and the lemma follows. □

Proof of Lemma 2 (cont.) The area of a cell of $\mathcal{G}$ is $2dK^{1-d}$. The cost of the object associated to depth k cells is then at most $2d(d+1)\,(4d/K + \lambda_{d-1}(s))$. Summing over all objects gives a total of at most $2d(d+1)\,(4dn/K + \sum_s \lambda_{d-1}(s))$. 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 $2d(d+1)\left(d \cdot 2^{2-C} + \sum_s \lambda_{d-1}(s)\right)$. By Lemma 1, this is at most $\max\left\{\sqrt{2}(d+1), d(d+1)2^{2-C}/\gamma\right\}$ times the cost of $\mathcal{T}$. Putting everything together, $c(\mathcal{T}_k) \leq \left(1 + \sqrt{2}(d+1) + d(d+1)2^{2-C}/\gamma\right) c(\mathcal{T})$. □

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.

Unfortunately, this cannot be extended to yield approximations within $(1+\varepsilon)$. But as a side note, for scenes consisting of points only and with slightly more restrictive hypotheses on $\mathcal{T}$, the depth k can be chosen so that $\mathcal{T}_k$ has size $O(n^{1+\frac{1}{d-1}}) = O(n^2)$ (for any $d \geq 2$) and cost as close as desired to that of $\mathcal{T}$.

Lemma 4 *Let $\mathcal{T}$ be a d-dimensional tree, which stores a set S of n points. 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 d, ε and γ but not on S nor 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 σ which contains n_σ points and has depth $k = \frac{1}{d-1}(\log_2 n + C)$ is $c(\sigma) = (\gamma + n_\sigma)2d \cdot 2^{-k(d-1)} = 2d(\gamma + n_\sigma)/(2^C n)$. The proof for points hinges on the fact that, unlike an arbitrary simplex, a point belongs to at most 2^d leaves,[7] and that to be subdivided, a cell needs to contain at least one point. Hence, in $\mathcal{T}_k$ there are at most $2^d n$ leaves that 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 C. 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. □

This result may seem somewhat anecdotic, but scenes consisting of points only have some relevance (see Section 4.2).

4 Constructing trees whose costs approach the optimal

4.1 General case: simplices

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

Lemma 5 *The lookahead greedy strategy does not always give (asymptotically) 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 tree of depth at least $k+1$ has cost less than $2d(\gamma + n)$.*

[7] If this seems like nitpicking, consider that "point" might mean a very small simplex. Thus this result for points applies equally to any collection of small objects, where "small" means to be contained in some cube of side 2^{-k}. See also Section 4.2.

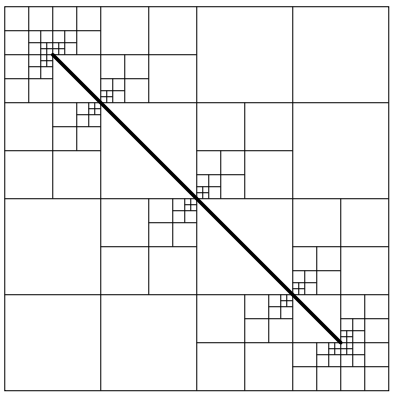

Fig. 2. An example of quadtree $\mathcal{Q}_{k,m}^{(2)}$; here $m = 2$ and $d = 4$.

The counterexample for $d = 2$ consists [9] 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 $\mathcal{Q}_{k,m}^{(2)}$ 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 more cost-effective to subdivide the corners, as shown by the quadtree in Figure 2 whose cost is less than $2d(\gamma + n)$.

Although the lookahead greedy strategy does not produce the optimal solution, in the counter-example above it does give a good approximation. In fact, this can be proven for all scenes.

Theorem 6 *Given a set S of convex objects of dimensions at most $d-1$ in the unit cube, let M be the infimum of $c(\mathcal{T})$ over all trees $\mathcal{T}$ storing S. There is an integer p which depends only on d ($p = 3$ for $d \leq 3$) 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 σ 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 σ does not improve on the cost of σ, then the cost of σ is $O(M_\sigma)$ where M_σ is the infimum cost of all the possible tree subdivisions of σ. If this holds true for every leaf σ of the p-greedy strategy, then $c(\mathcal{Q}^{(p\text{-greedy})}) = O(M)$ as well. We will need a technical lemma:

Lemma 7 *Let $\mathcal{G}$ be a $K \times \cdots \times K$ d-dimensional grid subdividing the unit cube, with K^d cells. For every d and K, there exist constants $G_d(K)$ and $C_d(K) > 0$ such that for any convex object s of dimension at most $d-1$, either s intersects at most $G_d(K)$ cells of $\mathcal{G}$, or else $\lambda_{d-1}(s) \geq C_d(K)$.*

Proof of Lemma 7. In two dimensions, a point intersects at most 4 cells, and any segment intersecting at least five cells of $\mathcal{G}$ must have length at least $1/K$, hence $G_2 = 4$ and $C_2 = 1/K$ suffices. Intuitively, in three dimensions, the maximum a segment can intersect is by joining two opposite endpoints of the unit cube, leading to $G_3(K) = 7K - 6$; in order to intersect any more cells, it must subtend a triangle of height at least half a diagonal of the cells of $\mathcal{G}$, thus have an area of at least $C_3(K) = \sqrt{3} \times (\sqrt{3}/2K)/2 = 3/(4K)$. The same argument in higher dimension leads to $G_d(K) = 2^d(K-1)+1$ and some $C_d(K) > 0$. Unfortunately, a formal proof is much harder. We give an easier proof which leads to slightly worse constants but works in any dimension.

Using the same notation as for Lemma 2, s projects onto some hyperplane $x_i = 0$ into an object s', and s intersects at most $(d+1)$ cells of $\mathcal{G}$ for every cell of $\mathcal{G}'$ intersected by s'. We now argue that in $d-1$ dimensions, either s' entirely contains a cell of $\mathcal{G}'$ or else it is contained in a slab of width $w = 2d\sqrt{d}/K$. Recall that any convex set has a pair of ellipsoids (called the Löwner-John ellipsoids) $e' \subseteq s' \subseteq e''$ such that $s' \subseteq de'$. If the width of s' is at least $2d\sqrt{d}/K$ in any direction, then the width of e' is at least $2\sqrt{d}/K$ in any direction as well and thus e' contains a sphere of diameter $2\sqrt{d}/K$, which in turn must entirely contain a cell of $\mathcal{G}'$ (of diameter $\sqrt{d}/K$). In that case, $\lambda_{d-1}(s) \geq \lambda_{d-1}(s') \geq \lambda_{d-1}(e') \geq 1/K^{d-2}$. Otherwise, s' is contained in a slab of width w as claimed. Any of the N cells of $\mathcal{G}'$ intersected by this slab is entirely contained in a slab of width $w + 2\sqrt{d}$, whose intersection with the cube must be of $(d-1)$-volume at least N/K^{d-1} and at most $w\sqrt{2}$, since the maximum section of a cube is $\sqrt{2}$ [6]. Thus $N \leq 2d\sqrt{d}K^{d-2}$. We may thus take $C_d(K) = 1/K^{d-2}$ and $G_d(K) = 2d(d+1)\sqrt{d}K^{d-2}$. □

Proof of Theorem 6 (cont.) We let $K = 2^p$ in the lemma, and choose p as the smallest integer such that $G_d(2^p) < (2^p)^{d-1}$. Assume there are a objects intersecting at most $G_d(2^p)$ cells, and b other objects. The cost of σ is $(\gamma + a + b)\lambda_1(\sigma)$. Since $\sigma_p^{(\text{complete})}$ has cost at most $\left(2^p\gamma + a\frac{G_d(2^p)}{2^{p(d-1)}} + 2^p b\right)\lambda_1(\sigma)$, which we assumed to be at least $c(\sigma)$, we have $c(\sigma) = (\gamma + a + b)\,\lambda_1(\sigma) \leq \left(2^p\gamma + a\frac{G_d(2^p)}{2^{p(d-1)}} + 2^p b\right)\lambda_1(\sigma)$, which implies that $a \leq (\gamma+b)(2^p-1)\left(1 - \frac{G_d(2^p)}{2^{p(d-1)}}\right)^{-1}$.

By lemma 7, an object which belongs to more than $G_d(p)$ cells has measure at least $C_d(p)$ so its contribution to the cost is at least $(d\sqrt{2}C_d(2^p))\lambda_{d-1}(\sigma)$. The optimal cost M_σ is then greater than $(\gamma + bd\sqrt{2}C_d(2^p))\lambda_{d-1}(\sigma)$. We have then proved that M_σ is at least a fixed fraction of the cost of σ, and the theorem 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 p-greedy strategy is thus a true improvement.

4.2 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 to our cost measure. (A good introductory reference is the proceedings of the first Symposium on Point-Based Graphics [14].)

In the plane, the 1- and 2-greedy strategy may produce a quadtree of cost $\Theta(n)$ times the optimal cost, and so does 1-greedy in higher dimensions. Indeed, this can be seen by placing n points in the center of the square. With no subdivision, the cost is $4\,(\gamma + n)$. With full subdivision to depth $k \geq 1$, the cost is $24\gamma + 16(n - 2\gamma)/2^k$ which tends to 24γ when k tends to infinity. After one or two subdivisions, then, the cost is $8(n + \gamma)$ or $16\gamma + 4n$, both of which are higher than with no subdivision, hence the tree is not subdivided according to the 1-greedy or the 2-greedy heuristic. This shows that the approximation ratio is $(n+\gamma)/6 = \Theta(n)$. In any dimension, the same example shows that the 1-greedy strategy also has a bad approximation ratio of $\Theta(n)$. Nevertheless, 2-greedy (for $d \geq 3$) and 3-greedy (for $d = 2$) both work near-optimally.

Lemma 8 *Given a set S of n points in the unit d-cube, if 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 σ is $(\gamma + |S_\sigma|)\lambda_1(\sigma)$. The cost of a complete subdivision $\sigma_3^{(\text{complete})}$ of σ to depth 3 is $(8\gamma + \frac{1}{8^{d-1}} \sum_{p \in S_\sigma} n_p)\lambda_{d-1}(\sigma)$, 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 $(8\gamma + |S_\sigma|2^{3-2d})\lambda_{d-1}(\sigma)$. If σ is a leaf of $\mathcal{T}^{(\text{3-greedy})}$, this means that $\sigma_3^{(\text{opt})}$ is just the leaf σ, then certainly $c(\sigma_3^{(\text{complete})})$ is greater than $c(\sigma)$. Hence $8\gamma + |S_\sigma|2^{3-2d} \geq \gamma + |S_\sigma|$, i.e. $|S_\sigma| = \frac{7}{1-2^{3-2d}}\gamma = C_d$. 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_σ is the infimum cost of all the possible quadtree subdivisions of σ. Since lookahead only subdivides a cell if it has a subtree that actually improves the cost, necessarily $\mathcal{T}^{(\text{3-greedy})}$ is a subtree of $\mathcal{T}^{(\text{opt})}$ (where $\mathcal{T}^{(\text{opt})}$ is any optimal[8] tree). Thus every leaf σ of $\mathcal{T}^{(\text{3-greedy})}$ is a cell of $\mathcal{T}^{(\text{opt})}$, and since

[8] The cost M is an infimum and may not be achieved by any finite-depth tree. But

$c(\sigma) = O(M_\sigma)$ then $c(\mathcal{T}^{(\text{3-greedy})}) = O(\mathcal{T}^{(\text{opt})}) = O(M)$ as well.

Redoing the computation with lookahead $p = 2$ 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 [4] 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 and produces smooth quadtrees (see next section).

As for finding the optimal tree, the question is still open whether for given values of γ (and maybe of n) there exists a p such that the lookahead greedy strategy yields the *optimal* result, namely, the sequence $c(\mathcal{T}_k^{(p\text{-greedy})})$ converges towards M as k tends to infinity. All we know is that if $n = 5$ and $\gamma < 1$ tends to 1, the required p tends to infinity. We can also mention that if every point belongs to at most one cell, then $p = 1$ leads to the optimal tree.

5 Rebalancing quadtrees and octrees

In all the applications of quadtrees and octrees, it can be important to maintain aspect ratio (hence starting with a unit cube) and to ensure that two neighboring cells don't have wildly differing sizes. This has led several authors to propose balancing for trees. 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. Rebalancing is known to increase size by at most a constant factor. In this section, we prove that rebalancing also does not affect the cost by more than a multiplicative constant factor.

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 tree 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 $\text{bal}_k(\mathcal{T}) = \min\{\mathcal{T}' : \mathcal{T} \prec \mathcal{T}' \text{ and } \mathcal{T}' \text{ is } k\text{-balanced}\}$, which is called the *k-rebalancing* of $\mathcal{T}$. For instance, 0-balanced quadtrees are what Moore called *smooth* quadtrees [20], and 1-balanced what he called *balanced* and others called 1-*irregular* or *restricted*.

for this proof, $\mathcal{T}^{(\text{opt})}$ can be allowed to have infinite depth. Alternately, one may consider an increasing sequence of trees $(\mathcal{T}_k^{(\text{opt})})_{k\to\infty}$ whose costs converge to M.

Theorem 9 *Let $\mathcal{T}$ be a tree storing both points and/or simplices in the unit cube. Then for any k, $0 \leq k < d$, $c(bal_k(\mathcal{T})) = O(c(\mathcal{T}))$.*

Since the cost $c(\mathcal{T}) = c_t(\mathcal{T}) + c_o(\mathcal{T})$, the proof will be a simple consquence of the next two lemmas. The following lemma was first proven by Weiser in 2D, and by Moore for any dimension $d \geq 2$, for a cost function which is simply the number of leaves. Here, the cost function is the total surface area of the leaves but their proof applies as well.

Lemma 10 *Let $\mathcal{T}$ be a tree. Then for any $k \geq 0$, $c_t(bal_k(\mathcal{T})) \leq 3^d c_t(\mathcal{T})$.*

Proof sketch. The construction suggested by Moore [20] starts with $\mathcal{T}' \leftarrow \mathcal{T}$ and subdivides every leaf node of $\mathcal{T}'$ which has a 0-adjacent neighbor whose size is smaller than half, or equivalently whose depth is greater by more than one. The process iterates until no further subdivision is necessary. The process must terminate with $\mathcal{T}' = \mathrm{bal}_0(\mathcal{T})$.

The basic observation is that, if all the neighbors of a leaf node σ have same or smaller depth, then σ never has to be subdivided [20]. Intuitively, σ will never force any further subdivision of these neighbors, and so they can only be subdivided as a side, corner, or border, on the opposite side of σ. Using this, Moore argues that every subdivision of an internal node σ in $\mathcal{T}'$ which is not an internal node of $\mathcal{T}$ can be "blamed" on a 0-adjacent internal node of $\mathcal{T}$ at the same depth as σ, and since a node in $\mathcal{T}$ can only be blamed by its 0-adjacent neighbors in $\mathcal{T}'$ at the same depth (at most 3^d of them), the total number m of nodes of $\mathcal{T}'$ can only be 3^d times the total number of nodes of $\mathcal{T}$. This also holds for the number ℓ of leaves, since by induction it is not hard to see that $\ell = 1 + (2^d - 1)(m - 1)/2^d$; indeed, subdividing a leaf σ adds 2^d extra leaves to m but σ is no longer counted in ℓ, so ℓ changes only by $2^d - 1$. Consult [20] for the details.

Regarding the cost, the above proof adapts naturally by monitoring, instead of the total number of nodes, the *total area* $c_a(\mathcal{T})$ of the tree (internal and external nodes). By induction, it is not hard to see that $c_a = 2c_t - 2d$; indeed, subdividing σ adds $2\lambda_{d-1}(\sigma)$ to c_a but c_t changes only by $\lambda_{d-1}(\sigma)$ since σ is no longer a leaf. When subdividing σ in $\mathcal{T}'$, the cost of σ's children can be "blamed" on a 0-adjacent internal node of $\mathcal{T}$ at the same depth as σ, and since a node in $\mathcal{T}$ can only be blamed by its 3^d neighbors in $\mathcal{T}'$ at the same depth, we easily have $c_a(\mathcal{T}') \leq 3^d c_a(\mathcal{T})$ which implies $c_t(\mathcal{T}') \leq 3^d c_t(\mathcal{T})$.

Since $\mathcal{T}' = \mathrm{bal}_0(\mathcal{T})$, which is a refinement of $\mathrm{bal}_k(\mathcal{T})$, for any $k > 0$, this implies that $c_t(\mathrm{bal}_k(\mathcal{T})) \leq c_t(\mathrm{bal}_0(\mathcal{T})) \leq 3^d c_t(\mathcal{T})$. □

Next we prove in Lemma 11 that the object cost of $\mathrm{bal}_k(\mathcal{T})$ is at most twice (for points) and some constant $B_d \leq 2d^2 4^d$ (for simplices) times that of $\mathcal{T}$. We prove this for a single object whose object cost is $\lambda_{d-1}(\mathcal{L}_s(\mathcal{T}))$, since the

object cost is just the sum over all objects of that quantity. We conjecture that the best value is $B_d = 2^{d-1}$.

Lemma 11 *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 $\lambda_{d-1}(s \cap bal_k(\mathcal{T})) \leq 2\lambda_{d-1}(\mathcal{L}_s(\mathcal{T}))$. If s is a convex object of dimension at most $d-1$, then there is a constant B_d such that $\lambda_{d-1}(s \cap bal_k(\mathcal{T})) \leq B_d \lambda_{d-1}(\mathcal{L}_s(\mathcal{T}))$.*

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 σ of $\mathcal{T}$ is never increased by more than a constant factor. The proof relies in that, when rebalancing, σ is only subdivided along its boundary, because only its neighbors can force it to subdivide. Thus, the subtree σ' of σ in $\mathrm{bal}_k(\mathcal{T})$ must satisfy $\sigma' \prec \sigma_\infty$, where σ_∞ is the maximum subdivision of σ along its boundary (an infinite border). By the next lemma and appropriate scaling, the object cost $\lambda_{d-1}(s \cap \mathrm{bal}_k(\mathcal{T}))$ is at most $B_d \lambda_{d-1}(\mathcal{L}_s(\mathcal{T}))$. □

Lemma 12 *The object cost of a flat object in $\mathcal{T} \prec \mathcal{T}_\infty^{(\mathrm{side})}$, and in $\mathcal{T} \prec \mathcal{T}_\infty^{(\mathrm{border})}$ respectively, is bounded by constants $S_d = d4^d$ and $B_d = 2d^2 4^d$ respectively.*

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 and 2^{d-1} half-scale $(k-1)$-sides. Going one level further, it is made 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 with $s_0 = 2d$ shows that s_k is bounded by $S_d = d4^d$ for all k. We can then consider a k-border as the union of $2d$ half-scaled k-sides (the superpositions only increase the cost), so that the cost is at most $2d^2 4^d$. □

Remark 1. The constant 3^d in Lemma 10 is tight for the number of leaves, but we do not know if it is tight for the cost of 0-balanced trees. When considering

k-balanced trees, $k \geq 1$, it certainly is overly pessimistic. Indeed Moore shows that in the plane, 9 is tight for 0-balanced quadtrees, but 8 is tight for 1-balanced quadtrees.

Remark 2. Aronov and Schiffenbauer independently proved [4, Th. 2] 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.

Remark 3. 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 the same approximation ratio.

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 $1 + \varepsilon$ for simplices? Since the optimal tree might be infinite, there is little sense in asking for an algorithm that constructs the optimal tree. But if the answer to this 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 $c(\mathcal{T})$ but the theoretically sound one that does model the average traversal cost during ray shooting is $c^*(\mathcal{T})$ (see Eq. 1). Our only result here is that the greedy strategy does not work.

References

[1] B. Aronov, H. Brönnimann, A. Y. Chang, and Y.-J. Chiang, Cost prediction for ray shooting, in: Proc. 18th Annu. ACM Sympos. Comput. Geom. (ACM, New York, 2002) 293–302. To appear in: Comput. Geom.: Theory and Appl..

[2] B. Aronov, H. Brönnimann, A. Y. Chang, and Y.-J. Chiang, Cost-driven octree construction schemes: an experimental study, in: Proc. of 19th Annu. ACM Sympos. Comput. Geom. (ACM, New York, 2003) 227–236. Comput. Geom.: Theory and Appl. 21 (1-2) (2005) 127–148.

[3] B. Aronov and S. Fortune, Approximating minimum weight triangulations in three dimensions, Discrete Comput. Geom. 21 (4) (Springer, New York, 1999) 527–549.

[4] B. Aronov and R. Schiffenbauer, unpublished manuscript (2001).

[5] J. Arvo and D. Kirk, A survey of ray tracing acceleration techniques, in: A.S. Glassner, ed., An Introduction to Ray Tracing (Morgan Kaufmann Publishers, San Francisco, 1989) 201–262.

[6] K. Ball, *Cube slicing in* $\mathbb{R}^n$, in: Proc. Amer. Math. Soc. 97 (3) (AMS, Providence, 1986) 465–473.

[7] M. Bern and D. Eppstein, Mesh generation and optimal triangulation, in: Computing in Euclidean Geometry, Lecture Notes Series on Computing, Vol. 4 (World Scientific, Singapore, 1992) 47–123.

[8] H. L. Bertoni, Radio Propagation for Modern Wireless Systems, (Prentice-Hall, Upper Saddle River, NJ, 2000).

[9] H. Brönnimann, M. Glisse, and D. Wood, Cost-optimal quadtrees for ray shooting, in: Proc. Canad. Conf. on Comput. Geom. CCCG'02, (Univ. Lethbridge, AB, 2002).

[10] H. Brönnimann and M. Glisse, Cost-optimal trees for ray shooting, in: Proc. LATIN 2004: Theoretical Informatics, LNCS Vol. 2976 (Springer, New York, 2004) 349–358.

[11] F. Cazals and C. Puech, Bucket-like space partitioning data structures with applications to ray tracing, in: Proc. 13th Annual ACM Symposium on Computational Geometry (ACM, New York, 1997) 11–20.

[12] A. Y. Chang, A survey of geometric data structures for ray tracing, Technical Report TR-CIS-2001-06, CIS Department (Polytechnic University, 2001).

[13] J. G. Cleary and G. Wyvill, Analysis of an algorithm for fast ray tracing using uniform space subdivision, The Visual Computer 4 (1988) 65–83.

[14] Eurographics Symposium on Point-Based Graphics 2004, M. Alexa, M. Gross, H. Pfister, S. Rusinkiewicz, eds. (Eurographics, 2004).

[15] A. S. Glassner, ed., An introduction to ray tracing (Morgan Kaufmann Publishers, San Francisco, 1989).

[16] J. Goldsmith and J. Salmon, Automatic creation of object hierarchies for ray tracing, IEEE Comput. Graphics Appls. (1987) 14–20.

[17] V. Havran, J. Bittner, and J. Prikryl, The Best Efficiency Scheme, http://www.cgg.cvut.cz/GOLEM/proposal2.html.

[18] J. D. MacDonald and K. S. Booth, Heuristics for ray tracing using space subdivision, The Visual Computer 6 (1990) 153–166.

[19] J. S. B. Mitchell, D. M. Mount, and S. Suri, Query-sensitive ray shooting, Int. J. Comput. Geom. & Appls. 7 (3) (1997) 317–347.

[20] D. W. Moore, Simplicial Mesh Generation with Applications, Ph.D Dissertation (Cornell University, 1992).

[21] D. W. Moore, The Cost of Balancing Generalized Quadtrees, in: Proc. 3rd Symposium on Solid Modeling and Applications (1995) 305–312.

[22] B. Naylor, Constructing good partitioning trees, in: Proc. Graphics Interface GI'93 (Morgan Kaufmann Publishers, San Francisco, 1993) 181–191.

[23] E. Reinhard, A. J. F. Kok, and A. Chalmers, Cost distribution prediction for parallel ray tracing, in: Proc. EUROGRAPHICS (Eurographics, 1998) 77–90.

[24] E. Reinhard, A. J. F. Kok, and F. W. Jansen, Cost prediction in ray tracing, in: P. Hanrahan and W. Purgathofer et al., eds., Rendering Techniques'97 (Porto, Portugal, 1996) 42–51.

[25] H. Samet, Design and Analysis of Spatial Data Structures (Addison-Wesley, 1990).

[26] L. Santaló, Integral Geometry and Geometric Probability, 2nd edition (Cambridge University Press, 2002).

[27] K. R. Subramanian and D. S. Fussell, Automatic termination criteria for ray tracing hierarchies, in: Proc. Graphics Interface GI'91 (Morgan Kaufmann Publishers, San Francisco, 1991).

[28] B. Von Herzen and A. H. Barr, Accurate triangulations of deformed intersecting surfaces, in: Proc. SIGGRAPH'87 (ACM, New York, 1987) 103–110.

[29] H. Weghorst, G. Hooper, and D. P. Greenberg, Improved computational methods for ray tracing, ACM Trans. Graphics 3 (1) (1994) 52–69.

[30] 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).

[31] K. Y. Whang, J. W. Song, J. W. Chang, J. Y. Kim, W. S. Cho, C. M. Park, and I. Y. Song, Octree-R: An adaptive octree for efficient ray tracing, IEEE Trans. Visual. Comput. Graphics 1 (1995) 343–349.