Cost-Optimal Trees for Ray Shooting*

Hervé Brönnimann† Marc Glisse†

1 Introduction

This paper deals with a cost measure on space decompositions for a given set of objects (points, segments in 2D, triangles in 3D). This cost measure was introduced in [1] to predict the cost of ray shooting. In a previous paper [2], we proposed a construction of quadtrees which yields cost $O(M)$, where M is the infimum of the cost measure on all quadtrees, for either points or segments in 2D. In this paper, we examine the case of octrees, and more generally d-dimensional trees with points and $d-1$-dimensional simplices, with respect to their cost measure.

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 neighboring leaves have similar size.) We also state and analyze a rebalancing algorithm for trees in arbitrary dimension, and show that rebalancing does not affect the cost by more than a constant multiplicative factor, for both points and $(d-1)$-simplices. We give explicit small constants for quadtrees and octrees.

For lack of space, we only state the results, without proof (included in the long version).

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:

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

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 σ, and $\lambda_{d-1}(\sigma)$ is the perimeter length (if $d = 2$) or surface area (if $d = 3$) of σ. The coefficient γ 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). When S is fixed, we simply denote the cost by $c(\mathcal{T})$.

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 λ_{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 σ, the total tree cost of its children is twice the tree cost of σ, 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 $d \ge 2$.*

The "optimal" tree may not have finite depth (it is 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, $M \ge 2d\gamma$.) In order to have an algorithm that terminates, we usually add an extra termination criterion such as a depth limit of k. When we say that a subdivision scheme is *optimal*, we mean that its cost when restricted to depth k tends to M as k tends to ∞. The expression *near optimal* will be used when the cost is $O(M)$ instead of M.

Let $\mathcal{T}$ be a d-dimensional tree which stores a set S of n convex objects of dimensions not more than $d-1$. Let $\mathcal{T}_k$ be the tree obtained from $\mathcal{T}$ by removing every cell of depth greater than k.

Lemma 2 *For $k = \log_2 n + C$, we have $c(\mathcal{T}_k) = O(c(\mathcal{T}))$ and the constant does not depend on n nor S.*

*Work on this paper has been supported by NSF ITR Grant CCR-0081964. Research of Hervé Brönnimann has also been supported in part by NSF CAREER Grant CCR-0133599. Marc Glisse was supported by École Normale Supérieure during his stay at Polytechnic University. This paper extends results presented at the Canadian Conference on Computational Geometry, CCCG'02.

†CIS, Polytechnic University, Six Metrotech, Brooklyn, NY 11201, USA. Email: `hbr@poly.edu`, `marc.glisse@ens.fr`.

3 Tree construction schemes

We follow the same terminology as [2], and generalize it to encompass any dimension. The root of all trees is the d-cube $[0,1]^d$. 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. All this notation is extended to starting from a cell σ instead of a unit cell, by substituting σ for $\mathcal{T}$: for instance, the complete subtree of depth k subdividing σ is denoted by $\sigma_k^{(\text{complete})}$.

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. If we recursively subdivide the unit cell until each leaf meets at most one object (the *separation criterion*), the resulting tree the *minimum separating tree*, denoted $\mathcal{T}^{(\text{sep})}(S)$, with variants where the recursion stops at depth k, denoted $\mathcal{T}_k^{(\text{sep})}(S)$.

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), can be computed by bottom-up dynamic programming, starting with the complete tree $\mathcal{T}_k^{(\text{complete})}$. Unfortunately, the memory requirements of this algorithm are huge for large values of k (although they remain polynomial if $k = \Theta(\log n)$.) 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, if the subtree $\sigma_k^{(\text{opt})}(S)$ does not improve the cost of the unsubdivided node σ, then the recursion terminates. Otherwise, we replace σ 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 σ instead of the unit cell). With no lookahead ($k = 1$), the *greedy strategy* simply examines whether one subdivision decreases the cost measure.

The following are the results we can prove for k-greedy:

Lemma 3 *Given a set S of flat objects in the unit cube, and let M be the infimum of $c(\mathcal{T})$ over all trees $\mathcal{T}$. For each d there exists a constant 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 $c(\mathcal{T}^{(p\text{-greedy})}) = O(M)$.*

4 Rebalancing quadtrees and octrees

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 $\mathcal{T}$.

Given a tree $\mathcal{T}$, there are simple algorithms to produce $\text{bal}_k(\mathcal{T})$ or a refinement thereof. We propose one that runs in time linear in the size of $\text{bal}_k(\mathcal{T})$. A very clever algorithm of Moore subdivides the cells of the quadtree while visiting in three passes, one forward, then backward, then forward [3], and produces the minimal k-balanced quadtree ($k = 0$ or 1).

For quadtrees, it is known that the number of leaves of $\text{bal}_0(\mathcal{Q})$ is at most 9 times that of $\mathcal{Q}$, and 8 times only for $\text{bal}_1(\mathcal{Q})$ (those constants are tight); in higher dimensions, more work remains to be done, but the bound of 3^d is an upper bound for the size of $\text{bal}_k(\mathcal{T})$ and any k. Thus, for any fixed d, the time to construct $\text{bal}_k(\mathcal{T})$ is proportional to the size of $\mathcal{T}$.

While the size of $\text{bal}_k(\mathcal{T})$ is in check, the final cost $c(\text{bal}_k(\mathcal{T}))$ is unknown, however. We prove the following, when objects in $\mathcal{S}$ are either points or segments.

Lemma 4 *Let $\mathcal{T}$ be a tree storing both points and/or simplices in the unit cube. Then for any k, $0 \le k < d$,*

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

Remark. The constant 3^d in Lemma 4 is tight for the number of leaves, but not for the cost of 0-balanced trees. When considering k-balanced trees, $k \ge 1$, it may be overly pessimistic. Indeed Moore shows that in the plane, a factor of 9 is tight for the size of 0-balanced quadtrees, but 8 is tight for 1-balanced quadtrees.

References

[1] B. Aronov, H. Brönnimann, A.Y. Chang, and Y.-J. Chiang. Cost prediction for ray shooting. *Proc. of Eighteenth ACM Symp. on Geom. Comput.*, 2002, Barcelona, Spain.

[2] H. Brönnimann, M. Glisse, and D. Wood. Cost-optimal quadtrees for ray shooting. To appear in *Canad. Conf. on Comput. Geom.* (CCCG'02), Lethbridge, Alberta.

[3] D.W. Moore. *Simplicial Mesh Generation with Applications.* Ph.D dissertation, Cornell University, 1992.