Vol. 13, No. 1: 57–69

Proximity Cluster Trees

Elena Jakubiak Hutchinson and Sarah Frisken
Tufts University Department of Computer Science

Ronald Perry
Mitsubishi Electric Research Laboratories

Abstract. Hierarchical spatial data structures provide a means for organizing data for efficient processing. Most spatial data structures are optimized for performing queries, such as intersection and containment testing, on large data sets. Set-up time and complexity of these structures can limit their value for small data sets, an often overlooked yet important category in geometric processing. We present a new hierarchical spatial data structure, dubbed a proximity cluster tree, which is particularly effective on small data sets. Proximity cluster trees are simple to implement, require minimal construction overhead, and are structured for fast distance-based queries. Proximity cluster trees were tested on randomly generated sets of 2D Bézier curves and on a text-rendering application requiring minimum-distance queries to 2D glyph outlines. Although proximity cluster trees were tailored for small data sets, empirical tests show that they also perform well on large data sets.

1. Introduction

Data sets can be searched faster when they are well-organized. For example, geometric objects, such as points and curves, can be arranged into hierarchical structures that expedite searches, such as nearest-neighbor queries (i.e., finding the closest object to a given point). Brute force can be used to search small data sets, but when the volume of queries on a small data set is high, the cumulative query times can limit overall performance. In such situations,

1086-7651/07 $0.50 per page

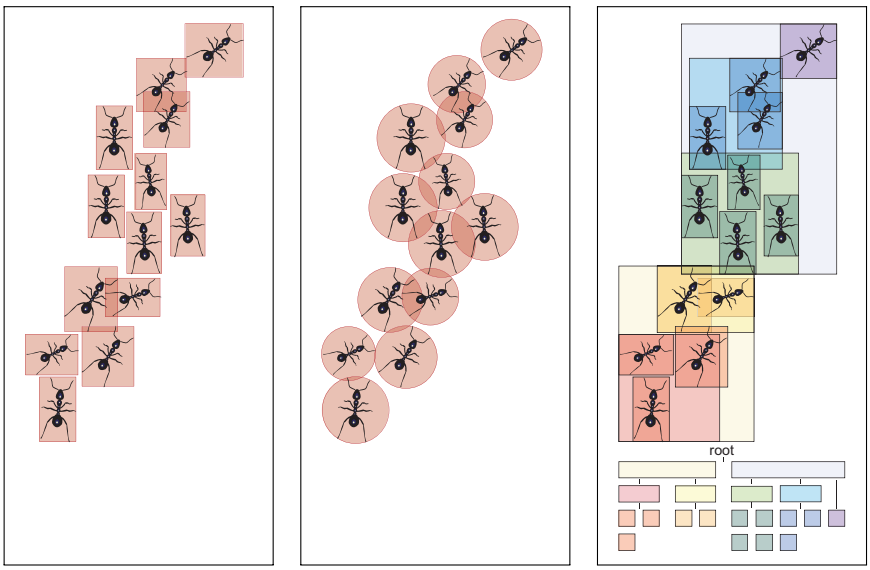
small data sets should be hierarchically organized. Classic hierarchical spatial data structures, however, were developed for querying large data sets and are inefficient when applied to small data sets.

Proximity cluster trees (PCTs) are hierarchical spatial data structures that organize spatial objects for fast minimum-distance computations. PCTs are constructed using a simple and efficient clustering heuristic that exploits natural spatial subdivisions in the data. Therefore, the structure of PCTs reflects the spatial layout of the data, making PCTs fast to query and ideal for applications that require many queries of small data sets. Moreover, empirical tests show that they also perform well on larger data sets. While our current algorithm is geared and implemented for querying 2D objects, we are currently investigating extending PCTs to three dimensions using 3D bounding elements such as bounding spheres and bounding boxes.

2. Background

It is common to search spatial data sets, such as those encountered in geographic information systems, for the closest object to a given point (e.g, what is the closest lake to my house). Applications that use distance-based queries are common in human-computer interaction (e.g., on which object did a user click), robotics (e.g., which objects are within reach of a mechanical arm), and collision detection (e.g., which objects are most likely to collide in the next time interval). Computer science literature is rich with research that addresses how to speed up distance-based queries for large data sets [Guttman 84, Hjaltason and Samet 99, Roussopoulos et al. 95]. Spatial data sets containing a small number of objects (e.g., fewer than 100 objects) are commonly searched using brute-force approaches. In some instances, however, brute-force searches of small data sets are too slow, especially when there is a high volume of queries (e.g., over 30,000 queries) and when the queries are part of a time-critical processing step.

There are two general approaches to speed up queries on spatial data sets: (1) geometric objects can be processed in terms of associated, simplified bounding regions, and (2) objects can be organized into spatial hierarchies. Simplified bounding regions, such as circles and rectangles in 2D or spheres and boxes in 3D, can be used to determine the general location of spatial objects without having to process the often complex geometric details of these objects (see Figures 1(a) and 1(b)). This approach is commonly used in collision detection, where objects collide only when their bounding regions overlap. Consequently, unnecessary, often expensive tests to determine if two objects have collided can be avoided by first checking if their bounding regions overlap.



(a) (b) (c)

Figure 1. Bounding regions can be used to simplify object representations for faster processing. (a) Axis-aligned bounding boxes are easy to compute, provide a relatively tight fit, and have a reasonably fast point-to–bounding region distance computation. (b) Bounding circles have a fast point-to–boundary region distance computation, but there may be excessive overlap between bounding regions. (c) For added efficiency, bounding regions can be organized into a hierarchical spatial data structure.

Using bounding regions alone, however, still requires an exhaustive test between the bounding regions of all objects. Such exhaustive searches can be avoided by organizing objects into a hierarchical spatial data structure (see Figure 1(c)). Hierarchical spatial data structures are often used in areas such as robotics (e.g., for motion planning), computer graphics (e.g., for ray tracing), and computational geometry (e.g., for intersection and containment testing). The hierarchical spatial data structure arranges objects into a tree-like structure with a root node that contains a bounding region of all the objects, internal nodes that contain a subset of the geometric objects and a bounding region that encloses all of these objects, and leaf nodes that contain a single geometric object and a bounding region enclosing that object. During a distance-based query, it is possible to prune entire branches of the tree, thereby eliminating objects contained in the leaves of the pruned branches from further testing. This can significantly reduce the number of distance computations.

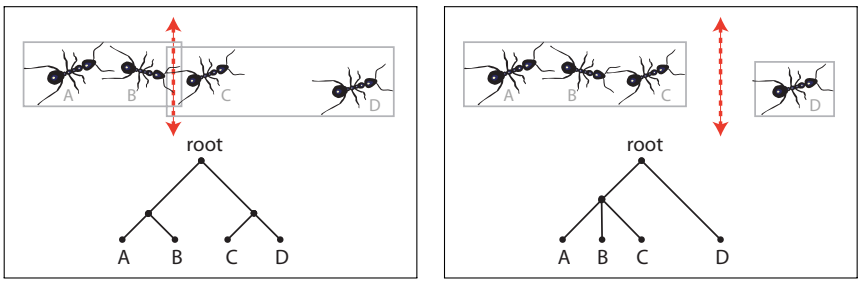


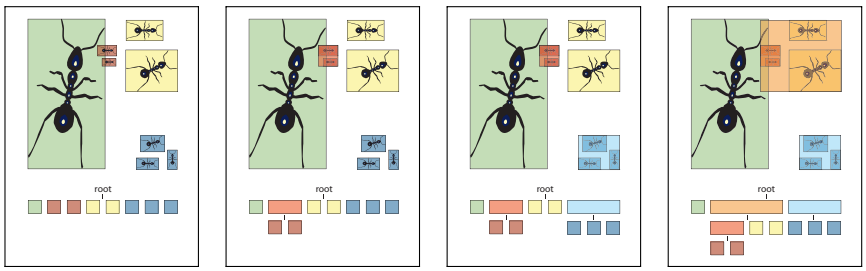
Figure 2. An example illustrating how a fixed number of children can impose unnatural subdivisions. (a) Using a binary tree as a model for a hierarchical spatial data structure can force poor grouping. (b) Allowing the number of children to vary leads to more natural clustering.

Classic hierarchical spatial data structures were developed for large data sets. When used for small data sets, the set-up time for constructing the spatial hierarchies can outweigh the time savings of querying well-organized data. Furthermore, most hierarchical spatial data structures fix the number of children that a node may have for part or all of the construction process [van den Bergen 97, Gottschalk et al. 96, Sappa and García 04, Xavier 96], thereby frequently forcing unnatural subdivisions between objects (see Figure 2). PCTs require minimal set-up time and do not fix the number of children that a node may have, thereby promoting a natural spatial clustering of objects.

3. Proximity Cluster Trees

3.1. Structure of a Proximity Cluster Tree

PCTs hierarchically group objects using axis-aligned bounding boxes as the bounding regions. PCTs reflect the spatial layout of the objects they contain. Larger objects, which are likely to be processed more frequently during a query, can exist at the same level in the tree as similarly sized clusters of objects. Internal nodes may contain a variable number of children (see Figure 1(c)). This natural organization of objects results in a "well-formed" spatial hierarchy where (1) objects processed more frequently during a query reside higher up in the tree, (2) overlap between the bounding regions of clusters is minimized, and (3) the tree has as few levels as possible given the previous two constraints. These features of well-formed spatial hierarchies promote faster distance-based queries.



(a) (b) (c) (d)

Figure 3. An example illustrating PCT construction. (a) All objects are initialized as children of the root node. (b) During the first application of the proximity clustering algorithm, the two dark-red nodes are identified as nearest neighbors and are clustered into the new light-red node. (c) Later during the first application, the three dark-blue nodes are clustered with their nearest neighbors, resulting in the new light-blue node. (d) During the second application of the proximity clustering algorithm, the clustering distance is increased. The light-red node and the yellow nodes are clustered into the new orange node.

3.2. Constructing a Proximity Cluster Tree

PCTs are fast and easy to construct. The initial PCT has only two levels—a root node and its children (see Figure 3(a)). Each child contains a single geometric object. The PCT is built by iteratively applying a proximity clustering algorithm to the tree. The proximity clustering algorithm creates internal nodes by clustering children of the root node based on their size and proximity.

The distance between the geometric centers of the bounding regions of any two nodes N_1 and N_2 (the "center-to-center" distance) reflects both the separation distance and the size of the bounding regions of N_1 and N_2. The center-to-center distance can be computed quickly and does not require user-tuned parameters. Each application of the proximity clustering algorithm clusters children of the root node whose center-to-center distances are less than a specified clustering distance (see Figures 3(b) and 3(c)). The specified clustering distance is increased with each application of the proximity clustering algorithm (see Figure 3(d)). This ensures that nodes that are of the same size and in the same proximity are clustered together for creating a well-formed spatial hierarchy. A PCT can be constructed using the following steps:

1. Create an initial PCT with two levels—a root node and its children, where each child contains a single geometric object.

2. Define `MAX_APPS`, the maximum number of times the proximity clustering algorithm is applied to the tree.
3. Define `MAX_CHILDREN`, the maximum allowable number of children of the root node.
4. Compute P_{avg}, the average perimeter of all the bounding regions of the objects.
5. Initialize N_{apps}, the number of applications of the proximity clustering algorithm, to 1.
6. While the root node has more than `MAX_CHILDREN` and $N_{\text{apps}} \leq$ `MAX_APPS`, apply the proximity clustering algorithm (Steps 6(a)–6(c)) and increment the number of applications of the proximity clustering algorithm (Step 6(d)) as follows:
   (a) Compute the clustering distance, $D_{\text{cluster}} = N_{\text{apps}} * \frac{P_{\text{avg}}}{\texttt{MAX\_APPS}}$.
   (b) Label all children of the root node as "old" nodes.
   (c) For each old child node C_i of the root node,
      i. Find its sibling with the closest center-to-center distance.
      ii. If the center-to-center distance between C_i and its closest sibling is less than D_{cluster}, merge the nodes in one of two ways:
         A. If the closest sibling is another old node, create a "new" child of the root node and move both C_i and its closest sibling from being children of the root node to being children of the new node.
         B. If the closest sibling is a new node, move C_i from being a child of the root node to being a child of its closest sibling.
   (d) Increment N_{apps}.

The shape of the PCT is affected by the two user parameters `MAX_APPS` and `MAX_CHILDREN` (see Table 1 and Figure 5 for typical values). During each application of the proximity clustering algorithm, several children of the root node may be clustered into a single node, and some children of the root node may not be clustered at all, providing a natural clustering that reflects the layout of the spatial data.

This simple clustering heuristic favors grouping nodes that are of the same size and in the same proximity, leading to a well-formed spatial hierarchy where (1) objects processed more frequently during a query reside higher up in the tree, (2) overlap between the bounding regions of clusters is minimized, and (3) the tree has as few levels as possible given the previous two constraints. In rare cases, the proximity clustering algorithm fails to group nodes that are

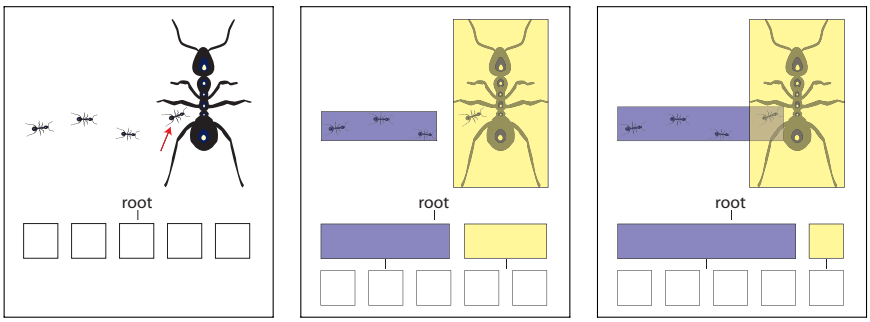


Figure 4. An example illustrating a PCT that fails to group nodes that are of the same size and in the same proximity. (a) The geometric center of the leaf node containing the indicated small ant is closest to the geometric center of the leaf node containing the large ant. (b) The proximity clustering algorithm groups the small ant with the large ant instead of with the other small ants. This leads to a PCT that is less efficient to query because the leaf node containing the small ant will be processed more often than necessary during a query. (c) Grouping all of the small ants together leads to a PCT that is more efficient to query.

of the same size and in the same proximity. This occurs when the bounding regions of the two nodes are of drastically different sizes yet their geometric centers are close (see Figure 4(a)). In such cases, a proximity cluster tree is still constructed but may be less efficient to query because the smaller node, which should be processed less frequently during a query, is processed as frequently as the larger node (see Figures 4(b) and 4(c)).

3.3. Querying a Proximity Cluster Tree

Although both depth-first and breadth-first searches are possible for top-down querying of hierarchical spatial data structures, both use a pre-determined search order and hence can be inefficient. Instead, we use a best-first search similar to that of Hjaltson and Samet, which proceeds by choosing to investigate the candidate node most likely to contain the desired object [Hjaltason and Samet 99]. For a minimum-distance query, with the goal of finding the closest object to a query point p, we proceed as follows:

1. Compute the distance from p to the bounding box of each child of the root node.
2. Insert the children of the root node into a sorted list L_{nodes}, where the nodes of L_{nodes} are sorted in ascending distance from p to their respective bounding boxes, as computed in Step 1.

3. Initialize the current minimum distance $D_{\min}$ to a large positive value.

4. While the minimum distance to the first node in L_{nodes} is less than $D_{\min}$, remove the first node from L_{nodes} and process it as follows:

   (a) When the node is a nonleaf node, unpack its child nodes and place them into L_{nodes} in sorted order.

   (b) When the node is a leaf node, compute the distance from p to the object contained in the leaf node and update $D_{\min}$ if appropriate.

The algorithm terminates when the minimum distance to the bounding box of the first node in L_{nodes} is greater than $D_{\min}$ since all of the objects contained in nodes at and beyond this point are guaranteed to be farther from p than the current closest object.

4. Results

4.1. Overview

PCTs were developed to accelerate the generation of adaptively sampled distance fields (ADFs) [Frisken et al. 00] from outline-based glyphs (e.g., TrueType fonts) for representing and rendering high-quality text [Frisken and Perry 06, Perry and Frisken 05]. Glyph outlines typically consist of line segments and a relatively small number of quadratic Bézier curves (e.g., 30) that are reasonably well-distributed spatially. Although our implementation was tailored for processing glyph outlines, experiments show that PCTs also perform well on larger data sets of randomly located quadratic Bézier curves. However, if the objects are very sparsely distributed spatially (e.g., the distance between objects is significantly larger than the perimeters of the objects), the method for computing the clustering distance needs to be tailored to the data.

We compared PCTs to several methods for performing distance queries: a brute force approach, pruning using various simple bounding regions (i.e., axis-aligned bounding boxes, the triangular convex hulls of quadratic Bézier curves, and bounding circles), and organizing the objects into a spatial hierarchy using the trees presented by García, Sappa, and Basañez (GSBTs) [García et al. 99], adapted to use axis-aligned bounding boxes in place of bounding circles.

We considered many existing algorithms that were developed to expedite distance-based queries. However, most algorithms fail to address the need for both quick initialization and well-formed trees. GSBTs were selected as the most promising existing approach for querying the minimum distance to a

small set of geometric objects since GSBTs are constructed using a straightforward approach that considers the general layout of objects before grouping them. GSBTs are created by first constructing an initial minimal spanning tree (a binary tree) using a proximity and size-based cost function. An n-ary tree is then formed from the initial binary tree by merging neighboring clusters according to their costs.

4.2. Simple Bounding Regions

We tested the use of a variety of simple, non-hierarchical bounding regions for pruning quadratic Bézier curves from the minimum distance query. These tests were performed on glyphs from a variety of typefaces. Results are presented for Arial and Times New Roman in Table 1; tests performed on other typefaces produced similar results. Axis-aligned bounding boxes, bounding circles, and bounding triangles (i.e., the convex hull of each quadratic Bézier curve) were used as bounding regions. Of these three bounding regions, axis-aligned bounding boxes produced the best results because of simple distance computations and sufficiently tight bounding regions.

4.3. Hierarchical Bounding Regions

Using bounding regions alone requires that for each query point, the point-to–bounding region distance is computed for every geometric object. Placing bounding regions in a hierarchical spatial data structure can significantly reduce the number of point-to–bounding region queries as demonstrated by PCTs and GSBTs [García et al. 99]—both methods significantly improve times for generating ADFs from glyph outlines (see Table 1). Figure 5 compares initialization and query times for these two methods applied to sets of randomly generated quadratic Bézier curves and randomly generated query points.

For small data sets, such as the quadratic Bézier curves of a glyph's outlines, both the initialization and query times are slightly better for PCTs than for GSBTs, resulting in font-generation times that are 5–15% faster (see Table 1). With randomly generated curves, PCTs demonstrate significantly faster initialization times and faster query times over GSBTs as the number of curves increases (see Figure 5). Additionally, PCTs do not require implementing supporting algorithms (e.g., an algorithm to construct a minimum spanning tree) as required by GSBTs, thereby simplifying implementations.

The better query times of PCTs over GSBTs can be understood by considering the structure produced by the two methods (see Figure 6). PCTs initially group objects into variable-sized clusters using global information,

Font	Method	Initialization Time (ms)	Query Time (ms)	Glyphs per Second
Arial	brute force	0.0000	7.00	142.82
	bounding boxes	0.0151	1.39	714.10
	bounding triangles	0.0210	1.97	501.19
	bounding circles	0.0194	1.66	596.86
	PCTs	0.0167	0.79	1244.06
	GSBTs	0.0602	0.87	1081.14
Times New Roman	brute force	0.0000	14.45	69.20
	bounding boxes	0.0106	2.50	397.67
	bounding triangles	0.0134	3.66	272.08
	bounding circles	0.0192	2.97	334.64
	PCTs	0.0236	1.37	719.54
	GSBTs	0.1021	1.45	645.45

Table 1. ADF generation times are compared using brute force, simple bounding regions, PCTs, and GSBTs [García et al. 99] to prune the quadratic Bézier curves of glyph outlines when computing minimum distances for antialiasing TrueType fonts. Results are measured based on the number of ADF glyphs generated per second using each method. Average times required to construct the particular data structure and to query the minimum distance for each glyph are also reported. In this example, PCTs were constructed using `MAX_APPS` = 4 and `MAX_CHILDREN` = 6.

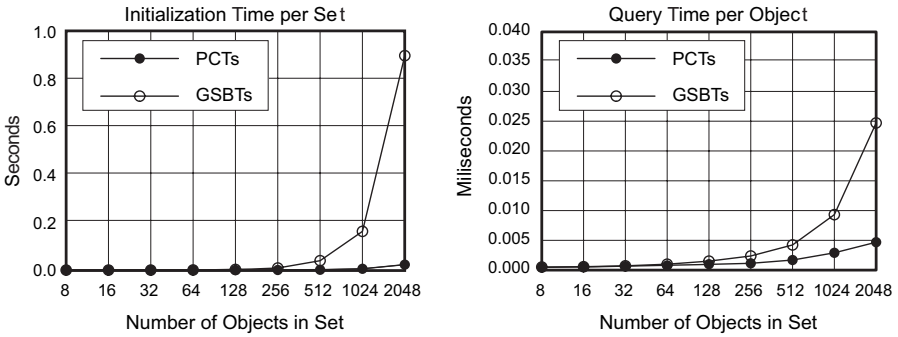
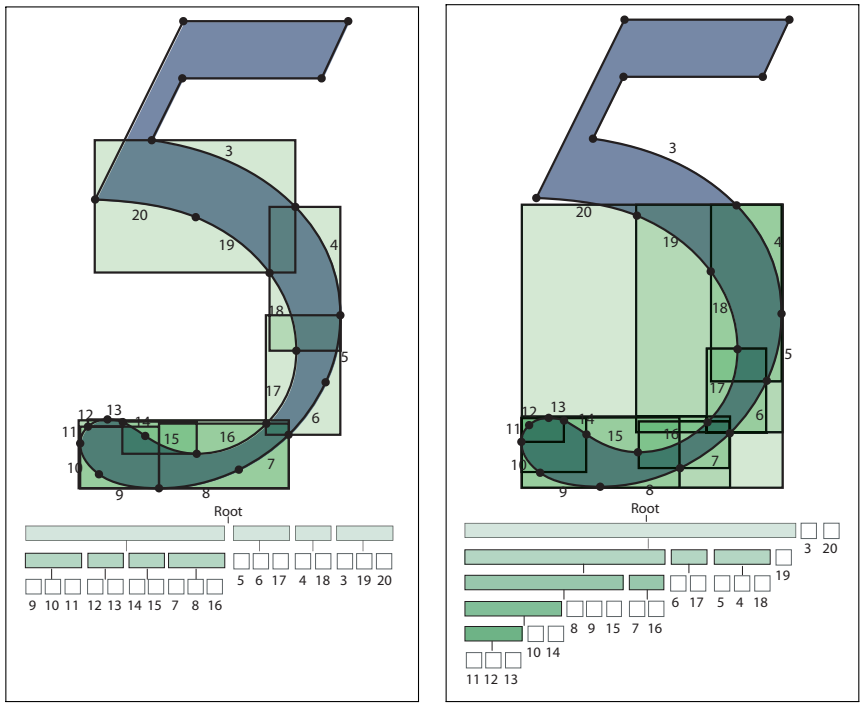


Figure 5. The initialization and query times of PCTs were compared to GSBTs [Garcá et al. 99] for sets of randomly generated quadratic Beziér curves and randomly generated query points. In this example, PCTs were constructed using `MAX_APPS` = 16 and `MAX_CHILDREN` = 6

providing a well-formed spatial hierarchy. In contrast, GSBTs tend to be unbalanced, thus making queries less efficient. GSBTs form clusters using local information. Although an object may be most similar to a given cluster from a global perspective, it may seem to differ locally, thereby forcing the creation of a new cluster resulting in a spatial hierarchy that poorly represents the underlying data (see Figure 6(b)).



(a) (b)

Figure 6. TrueType glyph outlines consist of line segments and quadratic Bézier curves. It is time-intensive to compute the distance to a quadratic Bézier curve. Organizing Bézier curves into a hierarchical spatial data structure expedites finding the closest curve. (a) PCTs better represent the layout of the underlying data in a well-formed structure. (b) GSBTs [García et al. 99] form clusters using local information that can force globally similar objects into different clusters, resulting in an unbalanced structure.

Acknowledgments. This research was funded by Mitsubishi Electric Research Laboratories.

References

[Frisken and Perry 06] Sarah F. Frisken and Ronald N. Perry. "Method for Antialiasing an Object Represented as a Two-Dimensional Distance Field in Object-Order." Patent no. 6982724, 2006.

[Frisken et al. 00] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics." In *Proceedings of SIGGRAPH*

2000, Computer Graphics Proceedings, Annual Conference Series, edited by Kurt Akeley, pp. 249–254. Reading, MA: Addison Wesley, 2000.

[García et al. 99] Miguel Angel García, Angel Domingo Sappa, and Luis Basañez. "Efficient Generation of Object Hierarchies from 3D Scenes." In *Proc. 1999 IEEE International Conference on Robotics and Automation*, pp. 1359–1364. Los Alamitos, CA: IEEE Press, 1999.

[Gottschalk et al. 96] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection." In *Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series*, edited by Holly Rushmeier, pp. 171–180. Reading, MA: Addison Wesley, 1996.

[Guttman 84] Antonin Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching." In *Proc. 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47–57. New York: ACM Press, 1984.

[Hjaltason and Samet 99] Gísli R. Hjaltason and Hanan Samet. "Distance Browsing in Spatial Databases." *ACM Transactions on Database Systems* 24:2 (1999), 265–318.

[Perry and Frisken 05] Ronald N. Perry and Sarah F. Frisken. "Rendering Cell-Based Distance Fields Using Texture Mapping." Patent no. 6917369, 2005.

[Roussopoulos et al. 95] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. "Nearest Neighbor Queries." In *Proc. 1995 ACM SIGMOD International Conference on Management of Data*, pp. 71–79. New York: ACM Press, 1995.

[Sappa and García 04] Angel Domingo Sappa and Miguel Angel García. "Hierarchical Clustering of 3D Objects and its Application to Minimum Distance Computation." In *Proc. 2004 IEEE International Conference on Robotics and Automation*, pp. 5287–5287. Los Alamitos, CA: IEEE Press, 2004.

[van den Bergen 97] Gino van den Bergen. "Efficient Collision Detection of Complex Deformable Models Using AABB Trees." *journal of graphics tools* 2:4 (1997), 1–13.

[Xavier 96] Patrick G. Xavier. "A Generic Algorithm for Constructing Hierarchical Representations of Geometric Objects." In *Proc. 1996 IEEE International Conference on Robotics and Automation*, pp. 3644–3651. Los Alamitos, CA: IEEE Press, 1996.

Web Information:

http://jgt.akpeters.com/papers/HutchinsonEtAl08/

Elena Jakubiak, Tufts University Department of Computer Science, 161 College Avenue, Medford, MA 02155 (jakubiak@cs.tufts.edu)

Sarah Frisken, Tufts University Department of Computer Science, 161 College Avenue, Medford, MA 02155 (frisken@cs.tufts.edu)

Ronald Perry, Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA 02139 (perry@merl.com)

Received March 15, 2007; accepted in revised form March 7, 2008.