Submitted to 1996 IEEE Int'l Conf. on Robotics and Automation

A Generic Algorithm for Constructing Hierarchical Representations of Geometric Objects*

Patrick G. Xavier
Sandia National Laboratories, Albuquerque NM 87185-0951

Abstract:

For a number of years, robotics researchers have exploited hierarchical representations of geometrical objects and scenes in motion-planning, collision-avoidance, and simulation. However, few general techniques exist for automatically constructing them. We present a generic, bottom-up algorithm that uses a heuristic clustering technique to produced balanced, coherent hierarchies. Its worst-case running time is $O(N^2 log N)$, but for non-pathological cases it is $O(N log N)$, where N is the number of input primitives.

We have completed a preliminary C++ implementation for input collections of 3D convex polygons and 3D convex polyhedra and conducted simple experiments with scenes of up to 12,000 polygons, which take only a few minutes to process. We present examples using spheres and convex hulls as hierarchy primitives.

1 Introduction

1.1 Overview

For a number of years, the robotics community has utilized hierarchical geometric representations to avoid *all pairs comparisons* in distance computations and collision detection, with motion planning being the main application. (For example, see [FT87, Qui94].) The automatic generation of hierarchical geometric representations, however, is still a research area, especially if one wants to control properties such as spatial balance, structural balance, coherence, and tightness.

*This research was supported by DOE Contract DE-AC04-94AL85000, and by the Laboratory Directed Research and Development Office of Sandia National Laboratories.

We present a practical generic algorithm for generating hierarchical representations. Our algorithm is generic in that it can be applied to collections of any convex primitive, e.g., a scene of composed of convex polygons or polyhedra. The core of the algorithm is a greedy bottom-up clustering technique that constructs a tree of sets. Tuning parameters allow control of the balance of desirable properties. A hierarchy of convex solids is then constructed by walking the tree. Our algorithm is practical in that is runs in time roughly $O(N \log N)$; our preliminary C++ implementation takes less than five minutes to construct hierarchical groupings from inputs of over 10,000 polygons. Building a hierarchy of solids takes seconds to minutes, depending on the solids and the desired tightness of the geometrical approximation.

1.2 Context

Geometric query algorithms (e.g., distance) frequently take tree-structured representations as input. A typical representation is a tree whose nodes each contain a *geometric primitive*, such as a sphere, convex polygon, or convex polyhedron. The subtree rooted at a node represents the union of the primitives at its leaves. We require that each node of a hierarchical geometric representation contain a conservative approximation, or *wrapper*, of the object represented by its subtree.

A typical branch-and-bound algorithm to answer a query about the geometric object represented by such a tree does a partial traversal, making a local query about the wrapper or primitive at each node it visits. Thus, a optimal tree would minimize

$$\sum_{\alpha \in \{\text{nodes}\}} c_l(\alpha) p(\alpha), \qquad (1)$$

where $c_l(\alpha)$ is the cost of answering the local query to

α and $p(\alpha)$ is the probability that a local query at α will be made in answering a global query. The catch is that even the relative probabilities are generally not known.

There are, however, several applicable heuristic properties. First, a *structurally balanced* tree is suited for the divide-and-conquer nature of distance computation. Second, it is intuitive that the probability that a node is "relevant" to a query would grow with its volume. This leads to the principle that a tree should be *spatially balanced.* Third, we desire *spatially coherent* trees, in which represented geometric objects that are spatially close are also structurally close. Not only are spatially coherent trees intuitively good according to the expression above, but because they agree with human intuition, algorithms that make use of the them would be more likely to perform as expected.

1.3 Our Results

We present a generic algorithm for the following problem. Given a collection of N geometric primitives for which convex wrappers can be efficiently computed, construct a hierarchy of convex geometric primitives such that (a) its leaves cover the input primitives, and (b) each node contains a wrapper for the objects at the leaves of the subtree rooted at it. We assume that the diameter of the largest primitive in the input is at most N orders of magnitude greater than that of the smallest.

Our approach has two parts. First, in the *Grouping Phase*, we construct a binary tree of sets, with edges corresponding to the subset relationship. This tree is called the *Subset Tree.* Second, in the *Geometrical Construction Phase*, we construct a structurally identical *Wrapper Tree*, in which each node contains a convex wrapper that spatially covers the union of the set elements at the corresponding node in the Subset Tree. Each Wrapper Tree leaf contains the geometric primitive from the corresponding Subset Tree leaf.

We first describe a generic algorithm for the Grouping Phase. This algorithm represents the spatial extent of a set of geometric primitives with a convex *p-set wrapper*,which can be any solid geometric primitive, and uses a heuristic clustering technique to pair up nodes to be siblings. Our description treats creating and merging of p-set wrappers as bottom-level operations. Parameters to a cost function can be used to control the relative emphases on structural balance, spatial balance, and spatial coherence. We present an informal analysis for the case in which the p-set wrappers are axis-aligned bounding-boxes. The analysis yields a worst case running time is $O(N^2 \log N)$, with an $O(N \log N)$ expected case under certain assumptions. We also show that using convex hulls for p-set wrappers yields an $O(N^2 \log^2 N)$ worst case, with $O(N^2 \log N)$ expected.

Recall that any convex primitive can be used for the wrapper type in constructing the Wrapper Tree. We describe three straightforward techniques for the Construction Phase: an $O(N^2 \log N)$ method using convex hulls as primitives, an $O(N)$ bounding sphere technique, and an $O(N \log N)$ bounding sphere technique that produces much tighter hierarchies.

We have implemented the grouping algorithm and both the sphere and convex-hull Wrapper Tree constructions in C++. A 12,000-polygon scene takes less than five minutes to process from scratch if the input primitives are not too packed. This time can be cut by orders of magnitude if instead of starting from scratch (1) component objects are pre-grouped and processed separately first, and (2) repeated component groups are processed separately and then copy-moved. Constructing the Wrapper Tree from the Subset Tree takes seconds to minutes, depending on the wrapper-type used. We provide illustrative examples.

1.4 Previous and Related Work

There is much previous work in hierarchical representations of geometrical objects. A good starting point reference on the use of hierarchical representations in collision avoidance is Faverjon [FT87, Fav89]. Work on the construction of the hierarchical representations falls into three categories: progressive refinement hierarchies, space subdivision, and constructive solid geometry (CSG). While CSG [Req80] is not related to our work, the other two are.

Notable early work in progressive refinement hierarchies includes that of Clark [Cla76], who describes no constructions but introduces a recursive-descent algorithmic paradigm and other important concepts. Rubin and Whitted [RW80] suggest a sort of clustering for automating the hierarchy construction from primitives. Weghorst, Hooper, and Greenberg [WHG84] describe an optimality measure for these hierarchies for ray-tracing, but require construction by the user. Goldsmith and Salmon [GS87] present an implemented method for automatically generating hierarchies of convex bound-

ing volumes. Their algorithm is similar to a "top-down" version of ours, but it has less control over the structural and spatial balance and depends on randomization for protection against "badly ordered data". One motivation for the presented work was to ensure desirable tree properties.

To speed collision detection, Quinlan [Qui94] constructs hierarchies of spheres using a divide-and-conquer method off-line. Earlier work by DelPobil, *et al* [dPSL92] constructs a representation with a given number of spheres, but restricts the input. Hubbard [Hub94] directly optimizes the tightness of the approximate representation at each level and includes bounds on the distance between each sphere and the object. His use of a medial-axis computation slows the method considerably, but it appears useful for off-line processing.

Related spatial-subdivision algorithms fall into two categories: those based on binary space partitioning (BSP) trees [FKN80] and those based on octrees. BSP trees and variants such as Vanecek's B-Rep Index have been used effectively in clash detection and modeling [NAT90, Nay92, Van91]. However, these structures have not yet been shown to be efficient for distance computation or the creation of general progressive-refinement hierarchies. Octrees for representing 3D geometric objects have been in the literature for at least a decade and a half; see [Man88] for a review. Extended octrees (see [BN90], for example) are more economical and permit exact representations. Octrees have also been used to create initial structures in generating other hierarchies [Hub93]. Non-uniform hierarchical space subdivisions based on balancing the number of primitives on each side of a cutting plane have also been used to create hierarchies, as reported by Zachmann and Felger [ZF95]. Although this type of partitioning is very fast, our previous experiments with a similar algorithm showed that the hierarchical representations were more prone to grouping artifacts than what we present here.

2 Building Hierarchies Via A Clustering Technique

2.1 Basic Idea: A Hierarchical Clustering Technique

We now describe the Grouping Phase of our algorithm. Recall that the general idea is to use a heuristic clustering technique to build a Subset Tree whose leaves contain the input primitives. Because of the need to handle inputs of size 10^3 to size 10^5, we must rule out any algorithm with an expected runtime of $O(N^2)$ or more. While we can't minimize cost (1) because of unknown distributions, we still want to have some control over spatial balance, structural balance, and coherence. Intuitively, this points to some sort of bottom-up construction.

Specifically, we had the following basic idea:

- Maintain a collection of nodes without parents.
- Until this collection contains just one node (the root of the Subset Tree), do the following: consider the convex hulls of the contents of pairs of nodes, and make a parent for the pair whose convex-hull diameter is minimal.

We call making two nodes the children of a new node *merging* them. This also entails conceptually combining the sets that corresponding to the node and creating a new p-set wrapper for the resulting set.

One problem with this approach is it appears to be at least $O(N^2 \log N)$ because of the number of node pairs that need to be considered. In addition, it lacks control over coherence; it ignores the heuristic that it is qualitatively better to merge two nodes if they are spatially contiguous or close than if they are slightly smaller but not contiguous.

Our solution to both problems is based on the observation that the diameter of new parent grows monotonically. Let us assume that the primitives are not long and skinny (or flat) and not closely packed like uncooked spaghetti. Then, suppose we set a diameter limit, and only consider pairs of nodes with p-set wrappers that have smaller diameters and centers that are closer than this diameter; when the queue is empty, we simply double the diameter limit. If the limit is set appropriately, this greatly cuts the number of nodes any given node could merge with. In fact, if we bound the aspect ratios of the primitives, then each node can merge with at most a constant number of others. This cuts the length of the queue of pairs of parent-less nodes to $O(N)$. Furthermore, by spatially hashing the parent-less nodes, we make constant the time it takes to determine which nodes another node can merge with.

2.2 The Clustering Algorithm

There are other factors we might want to consider in determining the priority of merging a pair of nodes. We define the *Balance* of a node to be the greater ratio of diameters of its children. We define the *Fill* of a node to be the ratio of its diameter to the sum of the diameters of its children. These measures can be used in assigning the *Cost* of merging a pair of nodes. The cost is associated with the p-set wrapper of the would-be new node. We have been experimenting with a *Cost* of the form:

$$Cost = diameter^A \times (B \times Fill + C \times Balance), \quad (2)$$

where A, B, and C, are non-negative constants. Roughly, increasing A improves spatial balance of the hierarchy. B and C are used to tune the emphases on spatial coherence and structural balance.

To speed the algorithm, instead of using convex hulls for the geometric extent of nodes and L_2 distance to measure gaps, we can use axis-aligned bounding boxes and L_∞ distance. We will use the term *body* to refer to a node, its p-set wrapper, and the set contained by that node. We now sketch the algorithm.

1. Let *GLimit* be the maximum gap allowed between two bodies to be merged. This must be positive if the data may contain more than one connected component. Let *DLimit* be the maximum diameter of bodies eligible for merging.

2. If there is only one parent-less body, return it; it is the root of the Subset Tree. Otherwise, let the set of active bodies *ABods* be the set of parent-less bodies that obey *DLimit*. Let rest of the parent-less bodies form the set of dormant bodies *DBods*.

3. Let $CSize = DLimit + GLimit$. Subdivide the bounding box of the input into voxels of edge-length *CSize* to create a body-center occupancy array. Then scan in the centers of the active bodies to determine possible eligible pairs, and use the gap limit *GLimit* to filter these to obtain the queue of eligible pairs *MPairs* sorted by increasing *Cost*.

4. Until *MPairs* is empty do the following:

   (a) Pop off the first pair in *MPairs*, and remove other pairs that share an element with this pair. Create a merged body from this pair.

   (b) If this body meets the *DLimit* criterion, then add it to the set *ABods* and determine which other members of *ABods* it is eligible to merge with, using and updating the array from step 3. Enqueue the new eligible pairs in *MPairs*.

   (c) Otherwise, add the new body to *DBods*.

5. Double *DLimit* and *GLimit*. Go to 2.

2.3 Subset Tree Analysis

We now present an informal analysis of the algorithm.

Since the hierarchy is a binary tree with N leaves, exactly $N-1$ interior nodes are created, and at most N bodies (nodes) are active at any time. Let us assume that during the construction no body is eligible to merge with more than k other bodies for some constant k that depends on how "pathological" the input is. We justify the assumption by considering the case in which input is uniformly sized and uniformly distributed and in which (2) is simplified to be *diameter*. The key observation is that because *DLimit* doubles in step 4 and the ordering of *MPairs* favors the creation of roughly cuboid bodies, doubling the *DLimit* and *GLimit* in step four does not, on average, increase the number of bodies any one body is eligible to merge with.

The assumption justified, it follows that the maximum length of the queue *MPairs* is kN, and at most kN queue operations are performed. These operations take total time $O(kN(\log k + \log N))$ time. Furthermore, consulting the array of lists in steps 3 and 4(b) also takes a constant amount of time dependent on k for each body. We note that the array can be implemented virtually with a spatial hash table, using array coordinates as keys and cell contents as data; thus, the storage for the array only really need be size $O(N)$. The other operations are constructing the geometric extents of the bodies, merging them, and determining the diameters and gaps. Assuming that we use axis-aligned bounding boxes, each takes constant time. Since all other individual operations are constant time, the total running time is then $O(kN(\log K + \log N)$, or $O(N \log N)$ for a given quality of input. In the worst case, with "pathological" inputs such as a box of spaghetti, the constant k is replaced by N. Thus the worst case running time is $O(N^2 \log N)$. Finally, ordering of *MPairs* also balances the tree to be of $O(\log N)$ depth, with the constant depending on the *Cost* function and the quality of the in-

put; the assumption about the ratio between greatest and least diameters among the input primitives also comes into play.

We note that if convex hulls are used to model geometric extent, then each merge takes time $O(n \log n)$ for a pair with $O(n)$ vertices. However, if we assume the input primitives each have a constant-bounded number of vertices, then the overall running time would be $O(N \log^2 N)$ because of the balance of the tree.

2.4 Building a Geometric Hierarchy

To construct the Wrapper Tree for the output of our algorithm, we do a depth-first traversal of the Subset Tree, assigning a wrapper to each node once we're done with its descendents. There are two ways we can assign a wrapper to a parent node. The first option is to geometrically merge the wrappers at its children. The second is to create a wrapper that geometrically covers the primitives at the leaves of its subtrees — i.e., the members of the set at the corresponding node in the Subset Tree. If we use convex hulls for the wrapper type, then these options are geometrically identical, costing $O(N \log^2 N)$ overall. However, if we use bounding spheres, bounding boxes, bounding ellipsoids, or any other sort of convex object for wrappers, then the second option results in geometric hierarchies that cover the collection of input primitive much more tightly.

The apparent advantage of the first option is that for bounding spheres, bounding boxes, and bounding ellipsoids, the cost of merging a pair is constant, so that the overall time for this phase would be $O(N)$. However, for axis-aligned bounding boxes, the constructions are identical. In addition, we note that computing an approximately minimum bounding sphere or rectangular prism for n points takes time $O(n)$ [Wu92]. Furthermore, a bounding sphere or bounding prism of the vertices of a collection of convex polygons and convex polyhedra bounds the collection itself. Recalling that we assume a constant-bounded number of vertices on each input primitive and that the Subset Tree will be of depth $O(\log N)$, we see that the cost of computing a Wrapper Tree of bounding-spheres or rectangular prisms directly from the corresponding nodes in the Subset Tree is $O(N \log N)$, which is perfectly acceptable since the constant is small and this complexity matches the cost of constructing the Subset Tree with axis-aligned bounding boxes as p-set wrappers.

3 Implementation and Examples

3.1 Implementation

We have completed a preliminary implementation of our algorithm in C++. By "preliminary" we mean that we have not yet attempted to optimize our code, e.g., by using single-floats wherever possible or using malloc-frugal memory management. For computing bounding spheres we use axis-aligned bounding-box centers. We use the Quickhull code from the University of Minnesota for computing convex hulls. We began our implementation on a Sun SPARCStation 10/51; we are currently developing the code on an SGI Indigo2 R4400/200.

3.2 Examples

We have tested with this implementation on a variety of inputs of up to 12,000 polygons. In this section we present some illustrative examples. In all cases we used axis-aligned bounding boxes for p-set wrappers when building the Subset Tree.

Figure 1 shows four levels of a tree generated with a 276-polygon robot base model as input, which yields a 12- or 13-level tree. (There is some randomization when ties occur.) The first frame shows the input. The second, third, and fourth frames show different levels of the bounding-sphere Wrapper Tree, progressing to the root. Leaves that are at or below the traversal level are represented as polygons, while interior nodes are represented as spheres. The three frames show representations of 251, 104, and, 8 nodes, respectively. Note that our clustering method does not guarantee a bound on structural balance; however, the implementation has mostly generated trees no more than $2 \log_2 N$ deep.

Figure 2 demonstrates the difference between the Subset Tree and the Wrapper Tree. The first frame shows a 626-polygon model of a lamp. For this input, the implementation produced hierarchies 16 or 17 levels deep. The next three frames again show different tree levels, with 414, 167, and 4 nodes respectively. The last two frames show 167- and 4-node levels of the corresponding convex-hull Wrapper Tree constructed from the same Subset Tree. The quality of the convex-hull Wrapper Tree suggests that our algorithm would produce very good hierarchies of bounding ellipses or rectangular prisms.

Our next example (Figure 3) illustrates a convex-hull

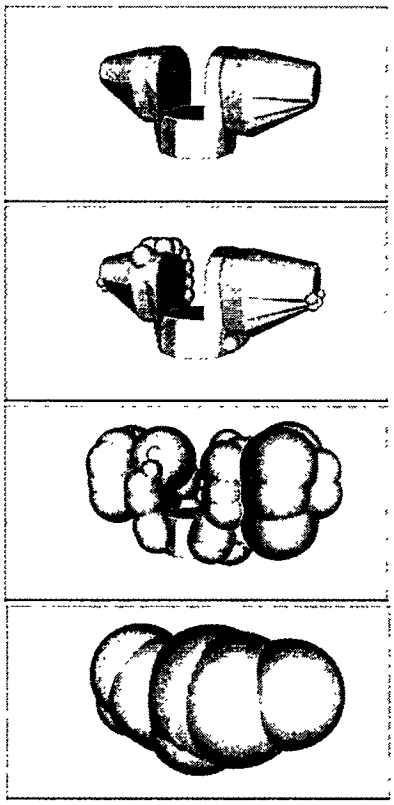

Figure 1: Four levels of the sphere hierarchy constructed from the 276-polygon model of a robot base part.

Wrapper Tree and shows again that using axis-aligned bounding boxes for p-set wrappers produces a good Subset Tree. The first frame shows a 3130-polygon example generated by replicating and randomly moving the lamp model four times. The second frame shows the 8-node level of the 21-level tree produced by our implementation in 40 seconds. Thus, three levels from the root, eight convex hulls cover the five lamps. Using boundary connectivity analysis to pre-group the input for separate (pre)processing would result in reduced overall running time and probably a lower-volume covering at this tree level.

Our final example (Figure 4) was generated by replicating and randomly moving the 43 times. The input to the algorithm was the resulting 12144-polygon list; this is shown in the first frame. Execution time was about 250 seconds. Increasing the average spacing so that fewer objects intersect decreases the running time. The second and third frames show two Wrapper Tree levels of 1604 and 126 nodes, respectively.

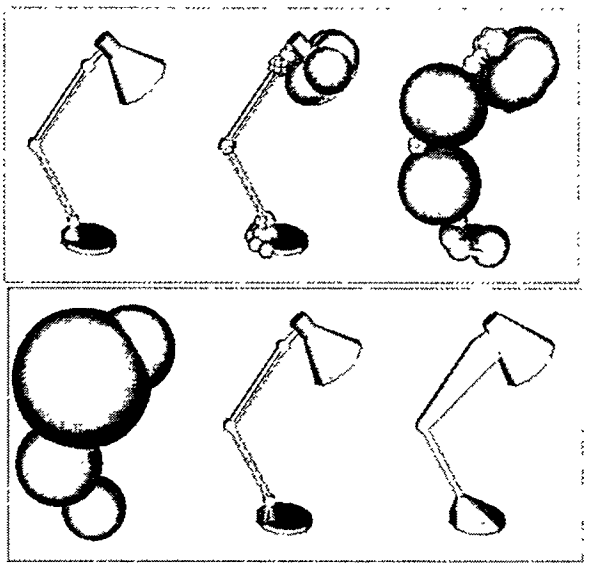

Figure 2: Four levels of the sphere hierarchy constructed from the 626-polygon model of a lamp; two levels from the convex-hull Wrapper tree.

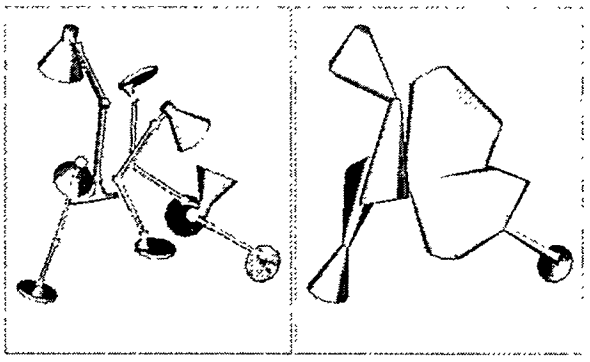

Figure 3: Multiple lamps, 3130 polygons; convex-hull Wrapper Tree level of 8 nodes.

4 Future Work

There are many directions for future work — in applications, in improving our algorithm, and in combining hierarchical techniques with other methods for distance computation and collision detection. We are currently implementing a distance computation algorithm similar to that of [Qui94] for a motion-planning application, and this application is certain to drive further research.

Towards improving our algorithm, the first question that arises is how to better handle large and flat or long and thin input primitives, such as the polygons that

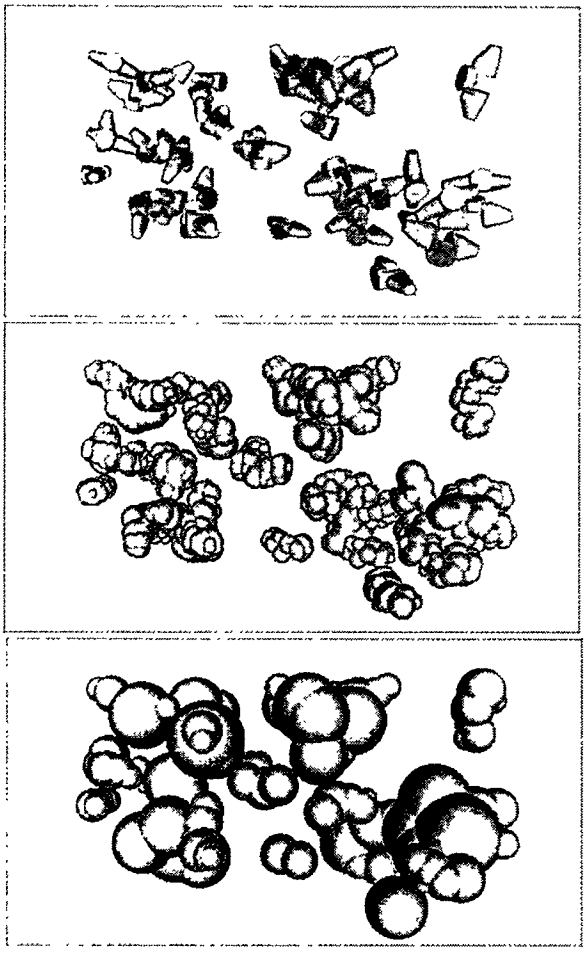

Figure 4: Three levels from the hierarchy constructed from a 12,144-polygon input.

model the surface of the lamp arm. A simple approach would be to preprocess the input to obtain polygons or polyhedra with limited aspect (primary-axis:secondary-axis) ratios, but this is only a partial answer. In addition, we would like to study the structure of the trees produced, in order to improve our control over structural and spatial balance. It would be also useful for the algorithm to compute bounds tightness of the approximation.

Another issue that arises is how to adapt the algorithm or use its output to produce a "near-optimal" collection of wrappers that cover the input, given a limit on the size of this output. We see this as a step to combining hierarchical representations with Baraff's [Bar90] and Lin's [LMC94] technique of updating bounding-box relationships. Their "Sweep and Prune" technique is faster than hierarchical techniques for contact-set detection when more than a certain fraction of primitives are close, and it is also advantageous when there are multiple moving bodies or the bodies are not rigid. However, there is the disadvantage that without some hierarchical organization, all the primitives must be updated. (This is somewhat unavoidable when the objects are flexible.) Furthermore, since Canny's and Lin's method [CL91] for computing the distance between collections of polygons that form the boundaries of two convex objects is a clear winner over tree-walking, a complete framework would include an algorithm for lopping off maximum convex boundary sections.

Finally, we would like to pursue a more efficient algorithm. The current algorithm is a suitable tool for use with input size up to 10^3, and it appears quite fast for off-line use with input sizes into the 10^4 range. While models and scenes are often broken up into individual object models each well within this range, in certain applications we encounter raw model data that is orders of magnitude larger. We are currently pursuing a top-down algorithm based on Goldsmith and Salmon [GS87] that promises a smaller time-constant than that of our current algorithm; key extensions will improve control over coherence and spatial and structural balance.

5 Conclusions

We have presented a general, bottom up clustering technique for automatically building hierarchical representations of objects. The first phase of the technique employs a heuristic clustering technique to construct a binary Subset Tree in time $O(N^2 logN)$ worst-case and $O(NlogN)$ expected. The second phase then uses this hierarchy to construct a Wrapper Tree of geometric objects using any of variety of primitives.

Our preliminary C++ implementation builds a sphere hierarchy from a 12,000-polygon input in less than five minutes. Most of this is spent in the grouping algorithm. Future work includes incorporating the technique into general distance computation and collision-detection schemes and investigating whether Goldsmith and Salmon's faster "top-down" grouping technique can be used while maintaining adequate control over characteristics of the hierarchies.

Acknowledgements

The author thanks Peter Watterberg (Sandia) and Philip Hubbard (Cornell) for providing the geometric

models used in our experiments. The author also thanks Russell Brown and Pang Chen (Sandia) for their comments and suggestions.

References

[Bar90] D. Baraff. Curved surfaces and coherence for nonpenetrating rigid body simulation. *Computer Graphics* (Proc. SIGGRAPH), 24(4):19–28, August 1990.

[BN90] P. Brunet and I. Navazo. Solid representationa dn operation using extended octrees. *ACM Transactions on Graphics*, 9(2):170–197, April 1990.

[CL91] J. Canny and M. C. Lin. A fast algorithm for incremental distance calculation. In *Proc. IEEE ICRA 1991*, pages 1008–1014, Sacremento, California, 1991.

[Cla76] J. H. Clark. Hierarchical geometri models for visible surface algorithms. *Communications of the ACM*, 19(10):547–555, 1976.

[dPSL92] A. del Pobil, M. Serna, and J. Llovet. A new representation for collision avoidance and detection. In *Proc. IEEE ICRA 1992*, pages 246–251, 1992.

[Fav89] B. Faverjon. Hierarchical object models for efficient anti-collision algorithms. In *Proc. IEEE ICRA 1989*, pages 333–340, Scottsdale, AZ, 1989.

[FKN80] H. Fuchs, Z. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. In *Proc. of ACM SIGGRAPH*, pages 124–133, 1980.

[FT87] B. Faverjon and P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. In *Proc. IEEE ICRA 1987*, pages 1152–1159, Raleigh, North Carolina, 1987.

[GS87] J. Goldsmith and J. Salmon. Automatic creation fo object heirarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[Hub93] P. Hubbard. Interactive collision detection. In *Proc. IEEE Sumposium on Research Frontiers in Virtual Reality*, pages 24–31, October 1993.

[Hub94] P. Hubbard. *Collision Detection for Interactive Graphics Applications*. PhD thesis, Brown University, Providence, RI, October 1994.

[LMC94] M. Lin, D. Manoucha, and J. Canny. Fast contact determination in dynamic environments. In *Proc. IEEE ICRA 1994*, pages 602–607, San Diego, CA, May 1994.

[Man88] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, MD, 1988.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. *Computer Graphics (Proc. SIGGRAPH 1990)*, 24, August 1990.

[Nay92] B. Naylor. Interactive solid geometry via partitioning trees. In *Proc. of Graphics Interface*, pages 11–18, May 1992.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. 1994 IEEE Int'l Conference on Robotics and Automation*, San Diego, CA, 1994.

[Req80] A. A. G. Requicha. Representations of solid objects — theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.

[RW80] S. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex scenes. *Computer Graphics (Proc. SIGGRAPH)*, pages 110–116, July 1980.

[Van91] G. Vanecek. Brep index, a multi-dimensional space partitioning tree. In *Proc. 1st ACM-SIGGRAPH Symp. on Solid Modeling Foundations and CAD/CAM Applications*, pages 35–44, Austin,TX, June 1991.

[WHG84] H. Weghorst, G. Hooper, and D. Greenberg. Improved computational methods for ray tracing. *ACM Trans. on Graphics*, 3(1):52–69, January 1984.

[Wu92] Xiaolin Wu. A linear-time simple bounding volume algorithm. In D. Kirk, editor, *Graphics Gems III*, pages 301–306. Academic Press, San Diego, CA, 1992.

[ZF95] G. Zachmann and W. Felger. The box tree: Enabling real time and exact collision detection of arbitrary polyhedra. In *Proc. 1st Workshop on Simulation and Interaction in Virtual Environments*, pages 104–113, Iowa City, IA, July 1995.

DISCLAIMER