

Near-Optimal Construction of Partitioning Trees by Evolutionary Techniques

T. Cassen K. R. Subramanian Z. Michalewicz

Department of Computer Science
The University of North Carolina at Charlotte
Charlotte, NC 28223
{tjc,krs,zbyszcz}@unc.edu

Abstract

We present a technique to construct space partitioning trees that can be used to efficiently represent, operate and manipulate geometric models. Our method is based on casting the tree construction problem as an optimization problem and using *evolutionary techniques* [13, 10, 19] to arrive at a near-optimal solution. Different metrics and cost models are used to evaluate the constructed trees. The metrics are aimed at optimizing criteria that are related to fast rendering, spatial operations such as point location and ray tracing. Extensions to other applications such as multi-resolution representation and compression are straightforward.

1 Introduction

Partitioning trees have been extensively used in representing geometric sets in various forms: octrees and quadtrees [27, 26], k-d trees [3], binary space partitioning trees [22, 8], to mention a few. Many of these representations also act as efficient structures for searching, most notably in rendering applications [9, 1, 15], solid modeling [23], shadow computation [2, 5] and global illumination algorithms [4]. In these applications, the search structure provides efficient access to the underlying model. A classic example of this is the use of hierarchical structures to accelerate ray tracing algorithms, where significant benefits are accrued by replacing needless intersection calculations with simpler tests with the hierarchy's partitioning hyperplanes.

Many partitioning trees can represent the same geometric set; in fact, each distinct ordering of partitioning hyperplanes that make up a tree will be a representation of the same geometric set, thus there are $n!$ different trees representing the same set, assuming n to be the number of partitioning hyperplanes. Examining all of them to optimize certain criteria is impractical, except for extremely small values of n . Thus, various heuristics are typically used to make

local decisions on the choice of hyperplanes, and this has been the widespread choice in constructing hierarchies [24, 11, 18, 15].

In this work, we are concerned with the problem of constructing partitioning trees, specifically, *binary space partitioning (BSP) trees*, which are generalizations of binary search trees to two or more dimensions. We are interested in constructing trees that are near-optimal for certain applications, such as rendering, and spatial operations relevant to geometric modeling. We will explore the use *evolutionary techniques* for representing and constructing BSP trees. Evolutionary techniques are based on *genetic algorithms*, and will be reviewed briefly in the following sections.

We demonstrate results from an initial implementation of a particular evolutionary technique for partitioning tree construction that has been used to minimize the total number of nodes in a BSP tree; additionally, we discuss an automatic way to terminate the tree construction (and the evolution) process, in a manner similar to [17], used for ray tracing hierarchies. We also show results of using this technique to construct hierarchies appropriate for ray tracing, employing a cost function that was developed in [17]. Comparisons to a particular hill climbing technique and a hierarchy construction algorithm very similar to [18] are also described.

2 Partitioning Trees and Evolutionary Algorithms

In this section we provide basic definitions, discuss various cost models, and briefly survey evolutionary algorithms in computer graphics.

2.1 Definition

A partitioning tree can be defined by a set of hyperplanes used to recursively partition an initially open region, and the regions (with their associated domain



specific attributes) that are generated by the partitioning. A method is needed to choose a hyperplane at each step, as well as a representation for the attributes of each (unpartitioned) region.

Binary space partitioning trees result when each partitioning step subdivides the initial region into two halfspaces, followed by recursive subdivision of the halfspaces until a termination criterion is satisfied. Regions that are unpartitioned are termed *cells* and become leaf nodes of the BSP tree, while the internal nodes contain the partitioning hyperplanes. Fig. 1 illustrates an example. Partitioning planes are indicated by letters and the subdivided (homogeneous) regions by numbers.

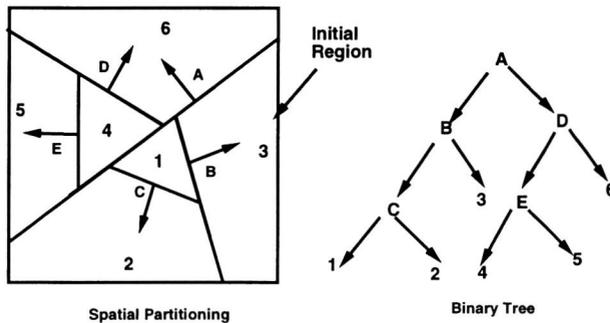


Fig. 1: Constructing a Partitioning Tree

Partitioning trees can represent continuous functions by subdivision of an initial domain (possibly infinite) into a hierarchical collection of sub-domains; points within each sub-domain i are defined by a value-continuous function f_i . Points at which f_i is discontinuous is typically contained within the partitioning hyperplanes. The function can be evaluated at any point x by simply following a path from the root of the tree to the cell containing the point and evaluating $f_i(x)$. The most common use of partitioning trees in this form has been in the representation and manipulation of polyhedral objects in interactive solid modeling [23] and game applications¹.

2.2 Cost Models

The overwhelming strategy in building good partitioning trees has been through the use of heuristics; a significant amount of work has appeared in building good search structures for ray tracing. The work of Goldsmith and Salmon [11] was the first method that had the capability of automatically constructing hierarchies of bounding extents. Equally important

was the use of a model that could evaluate the cost of the hierarchy during construction. The heuristic used can be stated as follows: for rays with an endpoint at a fixed distance from bounding extent, the probability that a ray will penetrate the bounding extent is proportional to the solid angle subtended by the surface of a bounding volume. At large distances and for convex volumes, the solid angle can be approximated by the surface area of the extent [31]. A comprehensive set of experiments [16] has demonstrated the quality of hierarchies constructed using this heuristic. Macdonald and Booth [18] used a similar heuristic to building partitioning trees with axis-aligned partitioners. In this case, the heuristic was to minimize the sum of the surface areas on either side of the partitioning hyperplane. In [15], it was shown that significant improvements in performance can be obtained if the two regions on either side of the binary partitioner are enclosed with bounding extents and the sum of the surface areas of the bounding extents throughout the hierarchy was minimized. A generalized cost model was also developed that was applicable to both hierarchies of bounding extents (with overlapping regions) as well as those employing partitioning hyperplanes (that produce non-overlapping regions), with or without bounding extents around objects.

In [24], a cost model was presented to work with binary space partitioning trees that minimizes expected cost; probability models were constructed for operations such as point location, inserting a line or plane into a partitioning tree, and, ray tracing. The model can be described as follows (reproduced from [24]):

$$E_{cost}[T] = \begin{cases} 0, & T \text{ is a leaf} \\ 1 + p^- * E_{cost}[T^-] \\ \quad + p^+ * E_{cost}[T^+], & \text{else} \end{cases}$$

where T is a partitioning tree at T , T^+ and T^- are the positive and negative subtrees, and p^+ and p^- are the probability models for the particular operation whose expected cost is being minimized. For instance, the model for point lying within a child region given that it is within the region of its parent is given by the ratio of sizes of the two regions:

$$p^+ = vol(r^+)/vol(r) + p^0 \\ p^- = vol(r^-)/vol(r) + p^0$$

which is not unlike Goldsmith's cost model, except that area was being used to calculate the conditional

¹DOOM uses a simplified version of a partitioning tree for visible surface generation.



probabilities. p^0 is the probability of a point lying on the partitioning hyperplane.

Good results were reported from using this model to generate multi-resolution representations of objects [21]. In [32], this model was applied to partitioning tree representations of medical images and the resulting trees exhibited good multi-resolution capability, demonstrated by interactively pruning the tree by continuous variation of a global error threshold.

2.3 Evolutionary Techniques in Computer Graphics

The idea behind evolutionary techniques (whether genetic algorithms, evolutionary strategies, evolutionary programming, genetic programming, to name a few) is to mimic the natural principle of genetic inheritance and survival of the fittest members of a population. Although there are a few independently developed and distinct approaches originated from such an idea [25, 29, 7, 13, 14], all evolutionary computation systems maintain a population of individuals representing potential solutions to the problem to be solved, incorporate a selection process based on an extrinsically imposed fitness measure of each individual, and include methods for creating randomly perturbed variants of current individuals, which are hopefully fitter new individuals.

Clearly, many evolutionary algorithms can be formulated for a given problem. Such algorithms may differ in many ways; they can use different data structures for implementing a single individual, genetic operators for transforming individuals, methods for creating an initial population, methods for handling constraints of the problem, and parameters (population size, probabilities of applying different operators, etc.). However, they share a common principle: a population of individuals undergoes some transformations, and during this evolution process the individuals strive for survival.

The recent work of Karl Sims [30] is perhaps the only direct application of evolutionary techniques to computer graphics. In this work the author used a genetic programming method [14] combined with interactive control of genetic parameters; this resulted in successful application of the system to evolving plants, images and textures (both 2D and 3D).

To evolve images, functions such as *abs*, *mod*, *and*, *bw-noise*, *color-noise*, *warped-color noise*, *blur*, *band-pass*, *grad-mag* were applied on an initial set of images. Images that looked ‘interesting’ were selected by a user for further application of different functions.

To generate 3D volume textures, another variable to incorporate the third dimension is added, together with 3D solid noise generating functions. Solid textures can be visualized by mapping them onto 3D surfaces such as spheres or moving planes, but unless powerful hardware is available, previewing them in real time is almost impossible.

By incorporating time as a variable into this process, animations can be evolved in a similar manner.

3 Evolving Partitioning Trees

In this section we provide a statement of the problem and discuss details of our evolutionary approach (representation, initial population, selection method, operators used, evaluation function).

3.1 Problem Specification

Our objective is to construct a ‘good’ partitioning tree that satisfies a particular evaluation criterion or cost function in a best way. To cast the tree construction problem as an evolution programming problem, we first recognize that the search space for this problem is the domain of all possible partitioning trees that are defined by a set of partitioning hyperplanes. Assuming there are n such partitioners, there are $n!$ different trees that define the partitioning of the same domain. The key ingredient that distinguishes one tree from another is the *order* in which the partitioners are chosen during construction. For instance, the tree in Fig. 1 has 5 partitioners and hence 120 possible trees.

Thus, our strategy will be to select a random subset of these trees as the chromosomes of our initial population and then evolve this population based on different criteria.

3.2 Chromosome Representation and Initial Population

As the ordering of the partitioners distinguishes one chromosome from another, each chromosome can be represented as a permutation of n integer numbers from 1 to n . For example, a permutation

$$(3, 23, 11, 17, \dots, 5)$$

uniquely defines a partitioning tree.

We create *pop_size* (population size) of random permutations; each permutation corresponds to a chromosome.



3.3 Selection Process

We have used a tournament selection in our system: pairs of chromosomes are randomly selected from the population and their fitness is compared. The more fit chromosome from each pair is chosen for the new population. For a population of size *pop_size*, *pop_size* pairs of chromosomes are examined. Since pairs are selected at random, duplicate chromosomes could appear in the new population. Also, we have used the elitist model, where the best (most fit) chromosome in the population is always included in the next generation.

3.4 Genetic Operators

Crossover and mutation are used to alter the chromosome's genetic material; we discuss them in turn.

3.4.1 Crossover

The crossover is similar to the order crossover defined by Davis [6] except for the difference in the way the operator maintains relative order of the partitioners.

For a pair of parent chromosomes we select a random crossover point C. All components of the first parent P1, from position 1 to C (inclusive), are copied to offspring O1. The remainder of O1 is populated by copying components from P2, from the beginning (i.e., from position 1); this copying is done in such a way as to avoid duplicate entries in the offspring.

Offspring O2 is formed in a similar manner, first copying from P2, then from P1.

The following example illustrates this procedure; for the following pair of parents

Position	1	2	3	4	5	6	7	8	9
Parent P1	A	B	C	D	E	F	G	H	I
Parent P2	H	F	I	A	C	E	B	G	D

and for crossover point at 4, we get

Position	1	2	3	4	5	6	7	8	9
Offspring O1	A	B	C	D	H	F	I	E	G
Offspring O2	H	F	I	A	B	C	D	E	G

The parents, P1 and P2, are replaced by O1 and O2, respectively.

3.4.2 Mutation

The purpose of mutation is to introduce a small random change in a chromosome. We have implemented this as a simple swap of two randomly chosen members of the chromosome being mutated.

3.5 Chromosome Evaluation

Evaluation of a chromosome is accomplished by applying a cost function to the constructed partitioning tree implied by this chromosome. We are currently experimenting with two different cost functions, targeted at tree construction for rendering applications. Our task is to minimize these functions.

Our first evaluation function returns the tree size, in terms of the total number of nodes in the partitioning tree. Operations that entail traversing large parts of the tree or the entire tree will clearly perform better when there are fewer nodes. A classic example is when the partitioning tree is used to generate objects in priority order, back to front [8]; when used for rendering objects in this order, all nodes are visited. It is assumed that all objects (or discontinuities) are contained within the partitioning hyperplanes. For instance, in [8], polygonal objects are rendered in this manner and partitioning hyperplanes are determined by the planes incident on each (polygonal) face that make up the object. This does not preclude using non-face partitioners in the tree, though.

Our second evaluation function is targeted towards constructing trees suitable for accelerating ray tracing algorithms. In [17], a cost model was developed for hierarchies appropriate for ray tracing. The cost function can be described as follows:

$$C_{tot} = C_{sc}(h, s) + C_{tr}(h, s)$$

where

$$\begin{aligned} C_{sc}(h, s) &= C_{pr} R(h, s) n_{pr}(h, s) \\ C_{tr}(h, s) &= R(h, s) n_{pl}(h, s) C_{pl} \end{aligned}$$

The total cost, C_{tot} , is the sum of $C_{sc}(h, s)$, the cost of examining the scene being rendered, and $C_{tr}(h, s)$, the cost involved in traversing the search structure of average height h . C_{pr} is the average cost of an intersection test with an object primitive, $R(h, s)$ is the expected number of regions in the search structure examined by each ray and $n_{pr}(h, s)$ is the expected number of primitives within each region. $n_{pl}(h, s)$ is the average path length of the tree and C_{pl} is the cost of performing a ray-hyperplane intersection. All of these quantities are computed as described in [17]. The main difference is the way the cost function is being used; in [17], it was used to predict the termination point of the search structure construction, whereas here the cost function will be used to evaluate a population of search structures for selection and optimization.



4 Experiments

An evolutionary system has been developed on UNIX workstations for constructing partitioning trees. The system works in conjunction with *SCULPT* [23], a solid modeling system under development by Dr. Bruce Naylor, AT&T Bell Laboratories, based on BSP Trees. All experiments were performed on a SGI Indigo-2 running IRIX 5.2.

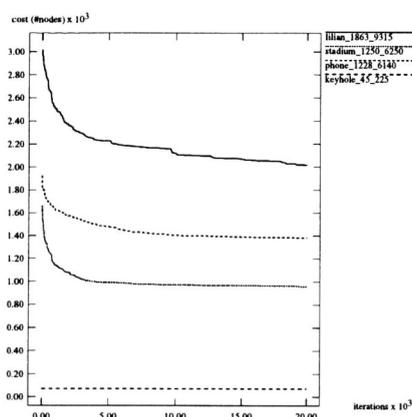


Fig. 2. Large Populations

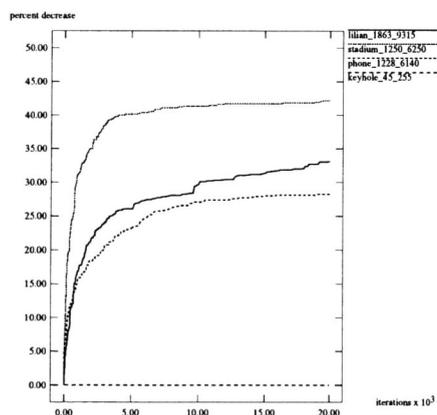


Fig. 3. Percent Decrease in Cost

The current implementation allows user control over chromosome population, mutation and crossover probabilities. Only polygonal objects are currently supported by this implementation, although this restriction will be overcome in the near future. While most of the experiments were performed with trees comprised of faces incident with hyperplanes, we have recently introduced the capability of allowing partitioning hyperplanes that do not necessarily have to be incident with any face of the input object. This would allow the system to construct better trees in

general, both for representation and manipulation of polyhedral objects, as well as making it useful for applications such as ray tracing. Our implementation also has the capability to construct k-d trees [3]; in a k-d tree, the partitioning hyperplanes are axis-aligned (with any of the 3 dimensions), however there is no restriction of the location or the particular dimension used at any stage of the partitioning (a detailed study of using k-d trees for ray tracing is described in [16]). A user can control the maximum number of hyperplanes that can potentially be used along each dimension (equally spaced within the domain of interest).

The first evaluation function that has been tested on several objects of varying size and characteristics returns the total number of nodes in the tree. Fig. 2 shows a plot of the cost (#nodes) vs. the number of iterations for four different objects. In each case, the population size was fixed at five times the number of polygonal faces of the object. Each characteristic is identified by the name of the object being tested, followed by the number of faces of the object and the population size at which the experiment was performed. The smallest sized objects converge to a minimum within a few hundred iterations, whereas the larger sized objects take longer to converge. Fig. 3 shows a plot of the percent decrease in the cost as a function of the number of iterations. The percent decrease is measured by calculating cost differences between consecutive points in the plot of Fig. 2 and normalizing with respect to first cost value, i.e. the cost after the first iteration. In each case, the cost decreases by about 25%, except for the 'keyhole' object exhibits only a small decrease, because of its high convexity.

Our goal is to be able to use this characteristic as a means to terminate the evolution process, when a plateau is reached. Some form of a threshold measure will need to be used in this regard; for instance, if the percent decrease is within $p\%$, over some k iterations, then the process can automatically terminate and output the best chromosome. Since there is very little change in cost in the flat portions of the characteristics, slight differences in the actual termination point should have minimal effect on the goodness of the tree.

In Fig. 4, we have plotted each object's characteristic at both a small population and a relatively large population. The small population size is equal to the number of polygonal faces of the object while the large population is fixed at five times the small population size. It is clear from the plots that increasing



the population size leads to faster convergence. This is especially true with the larger objects, where the minimum is not reached within 20,000 iterations.

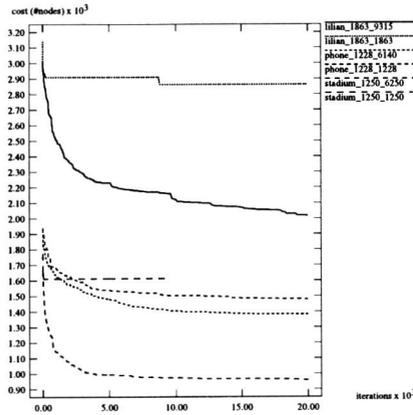


Fig. 4. Large vs. Small Populations

Object	faces	Pop. Size	Trees Const.	Time (Hrs.)	Tree Nodes
bird	231	231	19238	4.0	281
bookshelf	378	1890	22116	6.0	685
brush	279	279	19245	3.2	309
clutch	420	420	19534	2.5	267
hyperboloid	314	1570	21651	7.5	1315
keyhole	45	45	18841	0.9	73
lamp_stand	1185	5925	29178	18.0	1105
lilian	1863	1863	35160	19.8	2863
phone	1228	1228	21013	13.0	1479
stadium	1250	6250	29921	15.2	959

Table 1: Minimizing #Tree Nodes).

Table 1 shows the results for all of the objects on which experiments were performed and the minimum cost obtained. Fig. 5 shows some of the objects used in the experiments. In all cases, the crossover probability p_c was set at 0.85 and the mutation probability p_m was 0.25. The running times for the system are for a maximum of 20,000 iterations (and for construction of over 20,000 trees in most cases). When used in a real application with an automatic termination threshold, these will be lower, depending on the complexity of the object.

In order to compare this method of tree construction to competing methods based on local heuristics, a greedy method was implemented based on a "sliding window", which defines a local neighborhood. The method chooses a local neighborhood of hyperplane candidates, permutes them within the neighborhood, evaluates each of the permutations (the candidates outside the neighborhood remain

constant), and chooses the best among them. The candidates under the window that correspond to the lowest cost replace the corresponding candidates in the original permutation, the window slides over by a fixed amount (currently set at 1) and the process is repeated. The tree with the lowest cost is always retained until the end of the process.



Fig. 4. Sliding Window (Greedy Method)

Fig. 4 shows an example. Here the window is now in its 3rd position (with a width of 3). With three candidates, six different permutations can be generated, defining 6 different sequences of hyperplane ids and hence, six different trees. All six are typically evaluated and the best among these determine the order among the candidates 0, 4 and 1. The window is next is moved one position so that it now includes hyperplane 5.

Table 2 illustrates the results of using both methods on five different objects. In each case, the evolution method results in a tree with fewer nodes; the difference is more significant on the larger objects. A window size of 4 is used for the greedy method, but only 30% of the generated permutations (picked at random) are evaluated, due to computational constraints. In order to make a fair comparison to the evolution technique, the total number of evaluations is made the same for both methods and the evaluations are done from the same population of chromosomes. For the greedy method, each chromosome is evaluated in sequence until the maximum number of evaluations has been performed. The evolution method is also terminated when the maximum number of evaluations has been performed.

Object	Faces	Max. Trees	Min. #Nodes	
			Evolution	Greedy
keyhole	45	5000	71	77
bird	231	15000	299	369
brush	279	5000	353	385
hyperboloid	314	15000	1585	1699
phone	1228	15000	1769	2215

Table 2: Evolution vs. Greedy Method.

The cost function described in Section 3.5 was implemented and used with a k-d tree. In our experiments, the maximum number of axis aligned planes (a user controlled parameter) was fixed at 50 along each of the three dimensions. The planes are equally



spaced along each dimension. Each of these hyperplanes is a candidate hyperplane for tree construction and has a unique id. As before, a population of chromosomes is created by generating ids at random for each chromosome. The sequence of ids of each chromosome determines the order in which the hyperplanes are chosen. The hyperplane corresponding to the first id in the sequence becomes the root of the tree that partitions the object set into two groups (objects that intersect the hyperplane belong to both regions). In addition, the partitioning candidates are also subdivided into two groups. If for instance, a hyperplane aligned with the X axis is the chosen binary partitioner, the remaining hyperplanes aligned with X will be divided into two groups, located on either side of the partitioning hyperplane. The remaining candidates, aligned with the Y and Z axes, are all orthogonal to the chosen partitioning hyperplane and will be replicated on either side of the partition. If a region is determined to be a leaf cell (see below), then the remaining candidate hyperplanes are discarded and the objects become part of the cell.

Not all of the hyperplane candidates will necessarily be used in the constructed tree. This is determined by what constitutes a leaf cell of the tree. In our implementation a region becomes a leaf cell if (a) the number of objects within a region falls below a threshold, (b) the partitioning hyperplane has an intersection with all of the objects in the region or (c) there are no partitioners left to partition the region. For cases (a) and (b), the hyperplane is discarded as there is no benefit to partitioning a region with this hyperplane (as far as ray tracing algorithms are concerned). Case (c) can be avoided by choosing a larger value for the maximum number of hyperplane candidates.

Object	$R(h, s)$	$n_{pr}(h, s)$	$n_{pl}(h, s)$	C_{tot}
keyhole	1.78	3.07	11.43	156.17
bird	1.70	3.23	12.48	159.98
brush	1.72	3.08	12.13	155.42
hyperboloid	1.75	3.17	11.58	157.33
phone	1.80	2.99	12.23	160.94

Table 3: Minimizing C_{tot} using Evolution Method.

Table 3 shows the application of this cost function to five different objects. In each case, a maximum of 10,000 iterations was performed and the population is set to be three times the total number of hyperplane candidates. Upto 50 hyperplane candidates are located in each dimension. In order to better interpret these results, a heuristic was used to build the k-d tree

using the same set of hyperplane candidates (as used in the evolution method). The heuristic [18, 16], minimizes the sum of the surface areas of the bounding extents of objects across each side of the partitioning hyperplane. This was empirically shown in [16] to perform well on a number of data sets (many of which are from Eric Haines SPD package[12]). The hyperplane that minimizes this quantity (among the hyperplane candidates for this region) is chosen as the partitioning hyperplane.

Results of using this heuristic on the same set of objects is illustrated in Table 4. The costs using the evolution method is roughly half of that obtained using the area heuristic. $R(h, s)$, the expected number of visited regions, is significantly lower using the evolution method. A more detailed study is required to better understand these parameters, including performance measurement from ray tracing these objects using trees generated from both of these techniques. Effects of varying the maximum number of hyperplane candidates also needs to be studied.

Object	$R(h, s)$	$n_{pr}(h, s)$	$n_{pl}(h, s)$	C_{tot}
keyhole	2.76	3.63	16.61	316.70
bird	2.81	3.64	16.72	323.29
brush	2.76	3.59	16.71	315.58
hyperboloid	2.79	3.62	16.59	318.81
phone	2.81	3.61	16.70	321.55

Table 4: Minimizing C_{tot} using Area Heuristic.

5 Conclusions

Partitioning trees provide a good computational representation for a variety of applications, but constructing good trees has always been a hard problem, because of the number of possible trees. We have presented a method using ideas of artificial evolution as a way of providing a near-optimal solution. The major advantage as we see it, is the ease with which the problem can be formulated: most often, what is required is a cost function that can evaluate the generated solutions.

As to future extensions, a look at the computational times in Table 1 illustrates the necessity of speeding up this technique for its wide acceptance. Noting that the tree construction dominates the computation time, we are focusing on improving the construction time using two different approaches. One idea is to exploit the coherence of the chromosomes between successive generations. It is possible to cache frequently occurring subtrees (corresponding to substrings within a chromosome, represented as an array



of partitioner ids) and use tree merging [20] to efficiently construct trees for new chromosomes.

Second, we have noticed that as the trees evolve from generation to generation, the nodes of the tree closer to the root converge more rapidly than those deeper in the tree. This trend can be detected and the top level partitioners 'fixed'; which means a smaller chromosome will need to be considered from then on, making subsequent evaluations less expensive. Similar ideas have been explored in [28].

We plan to test various crossover and mutation operators and study their effects on the tree construction. We also would like to encode partitioning trees and apply evolutionary techniques to minimize the total number of bits needed to represent the geometric set. This is especially important when partitioning trees are used to represent discrete sets, for example, 2D images [32], for storage and transmission applications. It was suggested in [24, 32] that good partitioning trees typically exhibit multi-resolution capabilities and it would be interesting to verify this premise by building trees using the cost model to reduce expected cost in conjunction with artificial evolution. Lastly, it would be a good exercise to go back to the cost models suggested in [15, 11, 18] and apply the evolutionary techniques to study its implications; at the very least, this will prove to be a good benchmarking tool for measuring the effectiveness of cost models for different applications.

6 Acknowledgements

Our thanks to Bruce Naylor, AT&T Bell Laboratories, who has granted us access to *SCULPT*, on which this system is being built. This work was funded in part by funds provided by the University of North Carolina at Charlotte.

References

- [1] James Arvo and David Kirk. Fast ray tracing by ray classification. *Computer Graphics*, 21(4):269-278, July 1987.
- [2] A.T. Campbell, III and Donald S. Fussell. Adaptive mesh generation for global diffuse illumination. *Computer Graphics*, 24(4):155-164, August 1990.
- [3] Jon Louis Bentley. Data structures for range searching. *Computing Surveys*, 11(4), December 1979.
- [4] A.T. Campbell. *Modeling Global Diffuse Illumination for Image Synthesis*. PhD thesis, Dept. of Computer Sciences, The University of Texas at Austin, December 1991.
- [5] Norman Chin and Steven Feiner. Near real-time shadow generation using bsp trees. *Computer Graphics*, 23(3):99-106, July 1989.
- [6] L. Davis. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, 1987.
- [7] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.
- [8] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124-133, July 1980.
- [9] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15-22, October 1984.
- [10] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [11] Jeff Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, pages 14-20, May 1987.
- [12] Eric A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, pages 3-5, November 1987.
- [13] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.
- [14] J. Koza. *Genetic Programming*. MIT Press, 1992.
- [15] K.R.Subramanian. *Adapting Search Structures to Scene Characteristics for Ray Tracing*. PhD thesis, Dept. of Computer Sciences, The University of Texas at Austin, December 1990.
- [16] K.R.Subramanian and Donald S. Fussell. Factors affecting performance of ray tracing hierarchies. Technical Report TR-90-21, Department of Computer Sciences, The University of Texas at Austin, July 1990.



- [17] K.R.Subramanian and Donald S. Fussell. Automatic termination criteria for ray tracing hierarchies. In *Proceedings of Graphics Interface '91*, Calgary, Alberta, Oct. 3-7, 1991.
- [18] J. David Macdonald and Kellog S. Booth. Heuristics for ray tracing using space subdivision. *Visual Computer*, 6(3), June 1990.
- [19] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.
- [20] Bruce Naylor, William Thibault, and Bruce Naylor. Merging bsp trees yields polyhedral set operations. *Computer Graphics*, 24(4):115-124, August 1990.
- [21] Bruce F. Naylor. Personal Communication.
- [22] Bruce F. Naylor. *A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes*. PhD thesis, The University of Texas at Dallas, May 1981.
- [23] Bruce F. Naylor. Interactive solid modeling using partitioning trees. In *Proceedings of Graphics Interface '92*, Vancouver, CA, May, 1992.
- [24] Bruce F. Naylor. Constructing good partitioning trees. In *Proceedings of Graphics Interface '93*, Toronto, CA, May, 1993.
- [25] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, 1973.
- [26] Hanan Samet. *Applications of Spatial Data Structures*. Addison Wesley, 1990.
- [27] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, 1990.
- [28] N. Schraudolph and R. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9(1), June 1992.
- [29] H.-P. Schwefel. *Numerical Optimization for Computer Models*. Wiley, 1981.
- [30] Karl Sims. Artificial evolution for computer graphics. *Computer Graphics*, 25(4), July 1991.
- [31] L. Stone. *Theory of Optimal Search*, pages 27-28. Academic Press, New York, 1975.
- [32] K.R. Subramanian and Bruce F. Naylor. Representing medical images with partitioning trees. In *Proceedings of Visualization '92*, Boston, MA, Oct. 19-23, 1992.

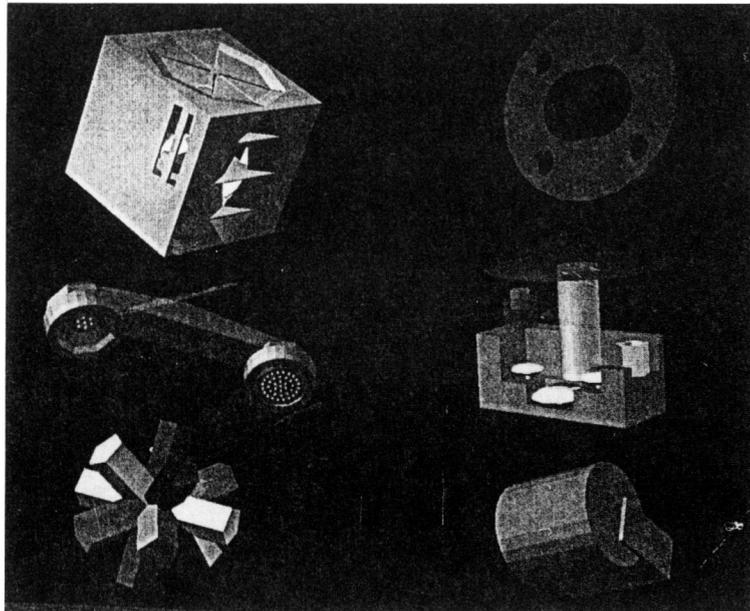


Fig. 5. (Clockwise from Top Left) (1) Lilian (2) Clutch (3) Lamp Stand (4) Keyhole (5) Brush (6) Phone

