Constructing Good Partitioning Trees

Bruce Naylor
AT&T Bell Laboratories
Murray Hill, NJ

ABSTRACT

Partitioning trees, a multi-dimensional generalization of binary search trees, is alone among the principal methods for representing geometry in combining the representation of a set with the geometric search structure required for efficient spatial operations such as set operations and visibility. Since a partitioning tree may be interpreted as specifying a program for exploring the structure induced on a space (e.g. by objects), there are many trees which represent the same spatial structure but provide different searches of the space. The issue of generating a good program to determine spatial relations between sets is then transformed into the issue of constructing good partitioning trees. The metric we choose for characterizing goodness is the expected cost of various elementary operations calculated using simple probability models. However, choosing the optimal from at least n! different trees by enumeration is not viable. Consequently, we employ heuristics that make local decisions based on the expected cost models. In addition to this quantitative methodology, we develop a qualitative understanding of what constitutes a good representation. This leads us to the notion of a good tree as one that provides a sequence or set of approximations, each obtained by various prunings of a single tree.

INTRODUCTION

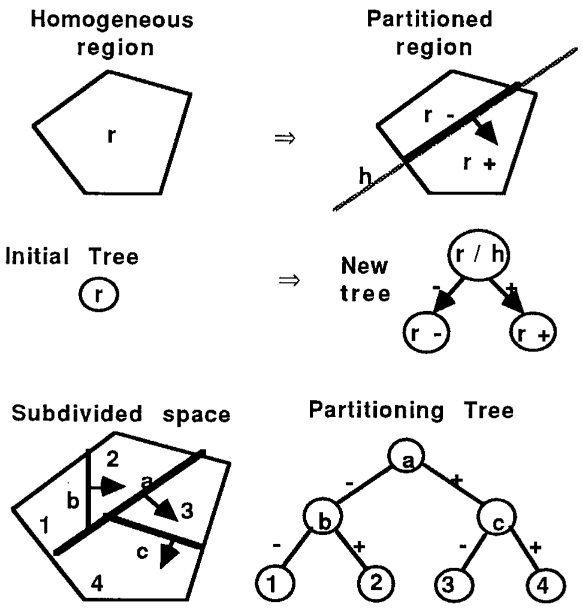
Of fundamental importance in any computational domain is the representation of the values in that domain, i.e. the data structures. For the domain of geometric computation, a few predominate representations have emerged. These include discrete space representations (arrays of lattice points), topological representations (boundary-reps, parametric surfaces), simplicial decompositions, and partitioning trees. Binary Space Partitioning Trees generalize binary search trees to dimensions > 1 and so combine a search structure with the representation of a set. Consequently, many partitioning trees can represent the same set and the classic issue of constructing a good search structure becomes a central issue for partitioning tree theory. It is this issue of constructing good partitioning trees that we wish to address in this paper.

A solution to this issue is well known for one particular case: point classification with 1-D trees. For this the methods of constructing optimal binary search trees using dynamic programming provide an optimal solution in O($\mathbf{n}^3$)**.** What is interesting about our problem is not this 1-dimensional case, however, in which the total ordering of the Reals permits the use, of dynamic programming, but instead the higher dimensional problem whose optimal solution appears intractable. We also wish to account for a wider variety of operations in which, unlike point classification, the operand being inserted into the tree has extent, such as is the case with set operations or ray tracing. The main technique we bring to this problem is utilizing expected case models to guide tree construction. Cost models for classifying points, lines and planes give us a measure of the goodness of any tree and provide a basis for improved heuristics for constructing trees.

The incipient versions of partitioning trees originated independently in computer graphics ([Schumacker et al 69] [Sutherland 73] and [Fuchs, Kedem and Naylor 80]) called *binary space partitioning trees (bsp trees)* for solving the visible surface problem, and in theoretical computer science (e.g. [Rabin 72]) as *linear decision trees* for proving lower bounds. The same structure has also been used to represent finite sets of points, where it has been called *partition trees* [Willard 82]. We will refer to the structure simply as partitioning trees.

Partitioning trees are most easily understood by considering the process that constructs them. Figure 1 illustrates this. Given a homogeneous open region **r**, a hyperplane **h** that intersects **r** is chosen using some criteria. Then **h** is used to induce a binary partitioning on **r** that generates two new d-dimensional regions, **r+ = r <intersect> h+** and **r- = r <intersect> h-,** where **h+** and **h-** are the positive and negative open halfspaces of **h** respectively. Also, generated is a (d-1)-dimensional region $\mathbf{r}^0$ = **r <intersect> h,** called *a sub-hyperplane.*
Thus
$\mathbf{r} = \mathbf{r}^+ \cup \mathbf{r}^- \cup \mathbf{r}^0 = (\mathbf{r} \cap \mathbf{h}^+) \cup (\mathbf{r} \cap \mathbf{h}^-) \cup (\mathbf{r} \cap \mathbf{h})$
Any of these new unpartitioned homogeneous regions can similarly be partitioned, and so on recursively to produce a binary tree of regions. When the process is terminated, the remaining unpartitioned regions, called *cells,* together with the

sub-hyperplanes forms a partitioning of the initial region. In figure 1, the cells are labeled with numbers and the sub-hyperplanes with letters.



Constructing a partitioning tree
Figure 1

Partitioning trees can be used as the basis for creating a computational object, or abstract data-type, for the semantic domain of geometric sets. These are subsets of a d-dimensional space which are the domain of a function of the form **f: d-space => attributes.** Specifically, we use it to create an isomorphism between geometric entities and a combinatorial structure manipulated by algorithms. A polytope can be represented by associating with each leaf a membership attribute = **{ in, out },** dividing the cells into <u>in-cells</u> and <u>out-cells</u>. A point can then be classified as **in** or **out** by following the path in the tree to the cell $\mathbf{c_i}$ that contains the point [Thibault and Naylor 87].

B-REP TO TREE CONVERSION

Since representing objects as trees is not how humans experience objects, it is natural to ask how could such a tree be constructed from something which is intuitive. The most common method is to convert from a boundary representation, which corresponds to how humans see the world, to a partitioning tree. Since all discontinuities/boundary-points must lie in partitioning hyperplanes, we can convert from a b-rep to a tree simply by using all of the face hyperplanes as partitioning hyperplanes (see Figure 2). The face hyperplanes can be chosen in any order and the resulting tree will always generate a convex decomposition of the interior and the exterior.

This conversion immediately raises the question as to the relative size of the two representations, and of course tree size is the simplest measure of goodness. If the only operations on the tree entails visiting the entire tree, as with visibility priority, then this is the only measure needed. Some worst case results on size were proven in [Naylor 81] where it was shown that partitioning trees can represent arrangements of hyperplanes and that the size of arrangements gives a tight bound of $q(\mathbf{n}^d)$ on the size of the largest possible partitioning tree formed using **n** hyperplanes in d-space. It was also shown that a set of disjoint **(d-l)-faces** could result in a tree of size $W(\mathbf{n}^{d-1})$. In [Paterson and Yao 89] algorithms were presented for converting a 2D b-rep of size **n** to a partitioning tree of size $q(\mathbf{n\ log\ n})$, and in 3D to a tree of size $q(\mathbf{n}^2)$.

However, empirical results first appearing in [Naylor 81] indicated that partitioning tree representation of 3D polytopes resulted in trees much closer to **0(n log n).** Much subsequent experience with constructing partitioning trees has been consistent with these initial data points. Thus, the worst case results must be interpreted with some care. We believe that the reason for the difference between the worst case complexity and the empirical results can be captured by a *principal of locality :* in most all circumstances, the individual geometric elements are bounded sets (e.g. faces, line segments, points) whose size is small relative to the region of space containing the set of elements. That is, they are purely local features with respect to that region of space (I am a local feature with respect to my house, city, planet, etc.).

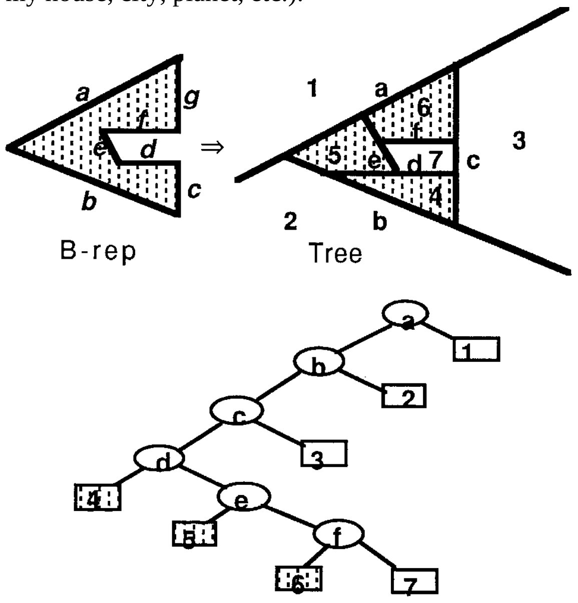


Figure 2

In contrast, for all constructions of worst case behavior discovered so far, the principal of locality is violated. This fact is obvious with arrangements of hyperplanes since by definition hyperplanes span all of d-space. The lower bound quoted above in [Naylor 81] of $\Omega(\mathbf{n}^{\mathbf{d-1}})$ was produced by arranging (d-I)-polytopes in d-space so that their projection was essentially equivalent to an arrangement of hyperplanes in (d-l)-space; consequently, there were no local features. The examples in [Paterson and Yao 89] are of a similar nature, and there is no example yet of a single manifold comprised of **n** faces generating worst case behavior. The crucial difference between arrangements of hyperplanes and partitioning trees is the ordering induced on the set of hyperplanes by the tree. This results in representing the elements using sub-hyperplanes which almost always span a local region of space and it is this property that allows partitioning trees to model local features efficiently.

Once spatial search operations are introduced, such as ray tracing [Naylor and Thibault 86] or set operations [Naylor, Amanatides and Thibault 90] (which includes clipping and collision detection [Naylor 92]), in which only a subset of the tree need be visited, the shape of the tree becomes at least as important as size, and probably more important, in determining the efficiency of partitioning tree algorithms (this observation is re-enforced by interpreting the partitioning tree as a computation graph, so the tree is the computation). With this fundamental change in the operations on a tree, it becomes misleading to compare tree size directly to the size of a brep until one includes in the comparison the size of an additional spatial search structure in which to embed the brep. When this is included for the worst case examples in [Paterson and Yao 891, say using a regular gird with a constant number of b-rep faces per grid cell, then the size of the grid has the same complexity as the size of a tree.

The essential property of partitioning trees upon which this paper is based is that there are many trees that can represent the same set or function. This suggests the need for a quantitative measure of "goodness" and methods to generate good trees. Below we introduce such a measure (actually a class of measures) and examine certain methods which we have implemented for constructing good trees that exploit directly the definition of this measure. However, before considering these ideas, we wish to first explore a qualitative characterization of good trees that is intuitive and leads to a more general understanding of efficient geometric computation.

APPROXIMATIONS

A central concept for understanding efficient geometric computation is that of the precision or resolution of a computed value; and the quantity of computation used to generate such a value at a particular resolution should be, in some sense, proportional to the resolution. In computer graphics, the most familiar example of this arises from the limits of the human visual system. We can quantize the image space and the color space into pixels, and the image need be computed only to the resolution of the pixels.

This leads us into the general milieu of approximations. In computer graphics, curved surfaces are often approximated by piecewise linear surfaces. A specific example of this is the recursive subdivision of Bezier curves. In this case, the set of successive control polygons generated during the process provides an ordered set of linear approximations of the curve. A classic example of an ordered set of approximations is the Taylor series representation of functions. Here, the kth element in an approximating sequence is the sum of the first k terms of the Taylor series.

One may also interpret an ordered set of approximations as providing a multi-resolution representation of a value. With finite, discrete space representations, such as for images as 2D arrays of pixels, the idea of a sequence of approximations has manifested in the form of a *pyramid* of images, where each successive member of the sequence has a 1D resolution that differs by a factor of 2. In the case of 3D continuous space, a version of this idea has been used for some time in real-time graphics systems as *levels of detail.* An object has two or three different representations each corresponding to a different level of detail. When an object is distant, a low level of detail is used, and when close, a higher level. Each level is typically constructed manually. Recently, work has appeared that provides automation of the process of creating a multi-resolution version of a b-rep, e.g. [Turk 92].

Partitioning trees as an ordered set of approximations

Recall that a partitioning tree generates a hierarchical decomposition of space, and the regions along any path of the tree are strictly decreasing in volume. This sequence of regions could be interpreted as a sequence of approximations that converges to the region corresponding to the leaf node (the last term in the sequence). This then suggests the potential of exploiting the hierarchy in order to realize the idea of a set of approximations of a set.

We begin our development by constructing a simple probability model for *approximate* point classification with respect to an arbitrary polytope **P.** For any finite region **r,** some percentage **p** of **r** lies in the interior of P and **q = 1-p** percent lies in the exterior. Thus if a point **x** lying in **r** is selected randomly from a uniform distribution, the probability that it lies in the interior of **P** is exactly **p** and in the exterior **q.** If we initially select a point **x** from some finite region **R,** say one that completely contains **P,** and we compute this classification probability for **R** and associate it with the root region of a partitioning tree, then this could constitute a first approximation: we would know the probability of any point being classified as **in** or **out**. Now we could equally well apply this to every node of the tree; that is, for each region **r** of the tree determine the following conditional probability: if a point **x** in **R** also lies in **r,** chosen from a distribution that is uniform over **R,** what is the probability that **x** is in the interior of the set (we will for now ignore the boundary, as it is a 0 measure set). The cells of the tree would, of course, have **p =** 1 or 0, reflecting whether the cell is **in** or **out** of **P** respectively.

Given a tree with such probabilities at each region, a simple way to treat it as an ordered set of approximations would be to threshold the expected values at each node to 0 or 1, which would also produce an expected error.

(classification, error)
if p < .5 then (0, p) else **(1,1-p)**

We could then classify **x** in **R** approximately by descending a path until the expected error was below some desired threshold, including no error, obtained by descending to a leaf. Figure 3 gives the simplest of examples. The tuples of numbers at internal nodes is *(expected value : threshold value, probability of being at the node, the expected error caused by thresholding).* The expected value for classification using only the root node of the tree is 1/3, and thresholding produces 0 resulting in an expected error of 1/3. We can improve this by classifying a point with respect to the first hyperplane. This gives us an expected error for a point in the left half (graphically) of 0 since this is a leaf, and in the right of (I- 2/3) * 1/2 = 1/6. So the total expected error for a tree comprised of the top three nodes is 1/6. This is an improvement over 1/3 and so represents a better approximation.

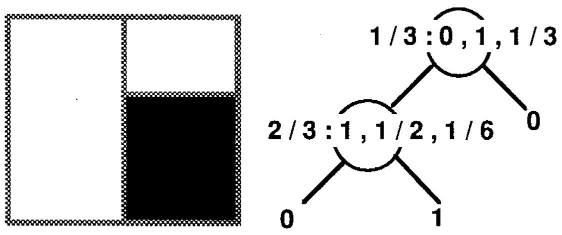


Figure 3

One may tend to think of a set of approximations only in terms of forming a sequence, since this is the simplest to understand and the most familiar. Thus one might envision each level in a partitioning tree as corresponding to an element in this sequence. But employing partitioning trees gives us not just a sequence but rather a *tree of approximations :* for a tree **T,** every connected subgraph of **T** which contains the root of **T** can serve as an approximation (although not necessarily a good one). A top-down view of this is that one can terminate any path at any node. A bottom-up view says that the subtree rooted at any node can be pruned away, being replaced with some approximation of that subtree. Those readers familiar with machine learning methodologies based on decision trees (the linear variety being syntactically the same as partitioning trees) will recognize this tree pruning strategy as a method for making generalizations.

EXPECTED CASE MODELS

The foregoing has been in the main a qualitative discussion, which while essential for our understanding does not directly give an algorithmic method of constructing good trees. A classic quantitative measure of goodness is worst case complexity, a measure that can typically be arrived at deductively (analytically). This is certainly an important measure, but it often provides only a first approximation to what is needed. It has long been thought that expected case complexity constitutes a better measure, but the required *a priori* analysis is often too difficult. However, this does not preclude writing a program for measuring the expected case complexity of a given tree, nor does it preclude utilizing expected case models to inform the tree construction process. And this is exactly what we have done.

Now one may ask what is the connection between the idea of a set of approximations and utilizing expected case models. If our thesis is true, that representing a geometric set as an ordered set of approximations is a good representation, then this goodness should manifest itself quantitatively by yielding good expected case complexity. Conversely, a method that constructs representations with good expected case complexity should produce a representation that is in the neighborhood of a set of approximations. The preliminary empirical evidence we have generated does in fact support such a connection. When we examine the trees produced by trying to minimize the expected cost of several elementary operations, the results appear to correspond to something like a set of approximations.

Let us now construct a cost model. Consider first a single operation involving a tree **T** and some other operand, such as a point, a ray or another tree, referred to as the *input* and denoted as **I**. Expected case analysis requires that one have a probability distribution of the input and the cost of performing the operation. By weighting the cost by the probability, the expected cost is defined as the weighted sum of all inputs:

$$\mathbf{E}_{\mathbf{cost}}[\,\mathbf{T}, \mathbf{I}\,] = \Sigma_j\, \mathbf{I[j].prob} * \mathbf{I[j].cost},\ \ 0 < j \leq |\mathbf{I}|.$$

If **I** is uncountable, then we would need to integrate instead of forming the sum.

This can be extended easily to encompass a set of operations **O**, such as rendering or set operations, where we associate a distinct input set **O[I].I** with each operation along with the probability of that operation being selected (the relative frequency with which a given operation is executed).

$$E_{cost}[\, T, 0 \,] = \Sigma_i\ 0[I].prob * E_{cost}[\, T, 0[i].I \,],\ 0 < i \leq |O|.$$

In practice, each **0[I].prob** can be estimated by modeling the context in which the tree will be used and applying any sound sampling methodology to determine the relative frequency of the operations.

Now to compute the expected cost for a particular operation for a given tree **T,** we will in effect insert **I**, treated as a random variable, into the tree. To do this we need, as always, to know how to "partition" **I** at an internal region **r,** and in this case this means we need to know the probability of **I** lying in **r+** and **r-**. If we assign a unit cost to the partitioning operation then we have:

```
E_cost[ T, I ] =
        IF T is a cell
        THEN 0
        ELSE $1 + p- * E_{cost} [T^-] + p+ * E_{cost} [T^+]$
```

This formula as stated does not directly express any dependency upon a particular operation; those characteristics are encoded in the two probabilities **p-** and **p+**. Once a model for these is specified, the expected cost for a particular operation can be computed for any tree.

Now consider point classification where **I** is a random point chosen from a uniform distribution over some initial region **R** partitioned by **T.** We need to know at each node of the tree corresponding to a region **r,** the conditional probability of the point lying in **r+** and **r-** given that it lies in **r.** For a uniform distribution this is determined simply by the relative sizes of the two child-regions to their parent region:

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

where **vol(r)** is the d-dimensional volume of **r,** and p^0 is the probability of a point being on the hyperplane. When a point is **on,** both subtrees are traversed to find all cells in its ε-neighborhood, which is essential if boundary points are to be identified. If on the other hand the boundary is treated as a 0 measure set, then p^0 will be 0; otherwise it is a function of the width of the subhyperplane, treated as being non-zero, and thus we would have $p+ + p- > 1$.

For classifying a line segment which spans the bounding volume of an object, we can model the segment by its two endpoints, x_1 and x_2, and assume that each endpoint is uniformly distributed on the surface of **r.** Then **T+** will be visited any time x_1 or x_2 is in **r+.** If we let p_i^+ denote the probability of endpoint x_i being in **r+,** and similarly p_I^- for being in **r-,** we have then

$$p_i^+ = area(\, r+ \,)\ /\ area(\, r \,) + p^0,\ \ i = 1,2$$
$$p_i^- = area(\, r- \,)\ /\ area(\, r \,) + p^0,\ \ i = 1,2$$

$$p^+ = p_1^+ * p_2^+ + p_1^+ * p_2^- + p_1^- * p_2^+$$
$$= 1 - (\ p_1^- * p_2^-\ )$$
$$p^- = p_1^+ * p_2^- + p_1^- * p_2^+ + p_1^+ * p_2^-$$
$$= 1 - (\ p_1^+ * p_2^+\ )$$

An alternative developed for ray classification in [Goldsmith and Salmon 87] models the probability of a line intersecting any region as being proportional to its area. We then have

$$p^+ = area(\, r+ \,)\ /\ area(\, r \,)$$
$$p^- = area(\, r- \,)\ /\ area(\, r \,)$$

In either case, $p^+ + p^-$ is always > 1. Thus, unlike point classification in which only a single path in the tree is typically visited, classifying a line requires traversing every path to a cell that contains the line, and so connected subgraphs of **T** are visited that are equivalent to the union of all such paths. Thus the above cost function calculates the cost of traversing these subgraphs weighted by their probability of being selected.

Those readers familiar with optimal binary search trees and Huffman codes will recall that probabilities are associated with each leaf node rather than with the decisions at internal nodes. If one thinks of the two probabilities p^- and p^+ being associated with the two edges of a given node, then the leaf probabilities are simply the product of the edge probabilities on the path from the root to a leaf. By using decision probabilities, we are able to easily extend our model from points to encompass extended input such as lines and planes.

The model for ray-tracing is similar to that for line classification, but since ray-tracing is an ordered search for the first intersection point, we have extended the model to account for this termination. Assuming that the ray is ordered from x_1 to x_2, then if x_1 is in **r-** and x_2 is in **r+** of node **v**, then the probability of traversing **T+** is reduced by the probability t^- that the search has terminated in **T-.** We must also account for the probability t^0 that the search terminates from intersection with any faces at **v**. This is modeled as **area(faces) / area(sub-hp).** Thus, using p_I^- and p_i^+ as defined above for the line model, we have:

$$p^+ = p_1^+ * p_2^+ + p_1^+ * p_2^- + p_1^- * p_2^+ *$$
$$(\ 1 - (\ t^- + t^0\ )\ )$$

$$p^- = p_1^- * p_2^- + p_1^- * p_2^+ + p_1^+ * p_2^- * (1 - (t^+ + t^0))$$

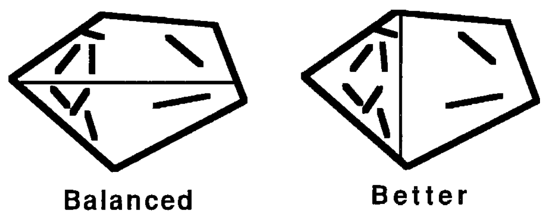
We would also like to directly model set operations, but this is too complicated. A related quantity is the expected cost of classifying a hyperplane. The intersection of a hyperplane with any region is a convex (d-I)-polytope. We approximate this with an extension of the idea used above for classifying a line segment. A sub-hp is approximated by a (d-l)-simplex whose vertices are uniformly distributed on the boundary of the region. For 3D we would have:

$$p^+ = 1 - (p_1^- * p_2^- * p_2^-)$$
$$p^- = 1 - (p_1^+ * p_2^+ * p_2^+)$$

where p_I^- and p_I^+ are defined as for the line model. While we would prefer a more accurate model, this nonetheless seems to capture the essential properties of the expected cost for this operation, for we have validated the model empirically on around a dozen examples, and the predicted value was typically within 5% of the measured value (the sample planes were generated by selecting 3 points randomly on the surface of the bounding box of an object).

If one accepts the above quantitative models, what then do they tell us qualitatively about good trees? The simplest characterization is that we want *short paths to high probability cells* (i.e. large cells), and symmetrically, we will accept *long paths to low probability cells* (small cells corresponding to "detail"). This is the same characterization that has been made of optimal binary search trees and Huffman codes, but by substituting "size of a region" for probability we obtain a geometric rather than a numeric interpretation.

By using decision probabilities rather than leaf probabilities, we see that locally the best situation at an internal region is one in which we have a partitioning in which most of the information is in a small sub-region (a low probability region), and as little as possible is in a large region (see Figure 4). Thus, *balanced is not optimal* in general, where balanced means equal sized trees with equal probability of being selected. Balanced trees are optimal only if the data is uniformly distributed, which of course is the same distribution for which uniform grids provide the optimal search structure. However, 3D geometric data is usually distributed very non-uniformly (consider a pine tree in the middle of a parking lot). Instead of balanced, the locally optimal decision for non-uniformly distributed data is a partitioning that is as asymmetric as possible. (Of course, no sequence of locally optimal decisions will necessarily lead to the global optimum.)



Balanced is not necessarily optimal

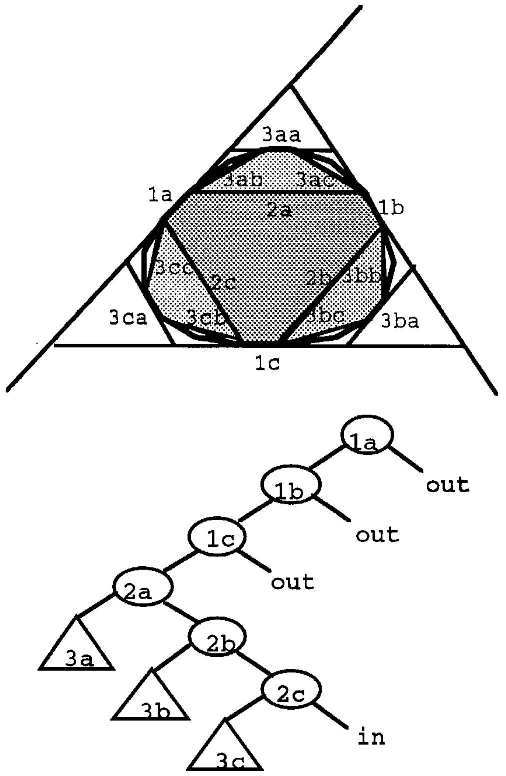
Figure 4

Representation of convex polygons

Let us now apply some of these ideas to constructing a good expected case representation of convex polygons. In figure 5, we illustrate our proposal. The first three hyperplanes create a simple approximation of the exterior (a.k.a. a bounding volume) and the next three similarly form an approximation to the interior. This is consistent with our rule of thumb that says we should construct short paths to large homogeneous regions. This partitioning scheme also divides the polygon into three segments each contained within a triangular region. For each, we construct the subtree recursively by choosing the hyperplane of the median face to improve the approximation of the exterior, followed by two hyperplanes incident with this face which improve the interior's representation. This process yields a nearly balanced tree whose depth is **log n** for an n-sided polygon. More importantly, it provides a sequence of approximations of the interior and exterior which converge to the boundary of the polygon.

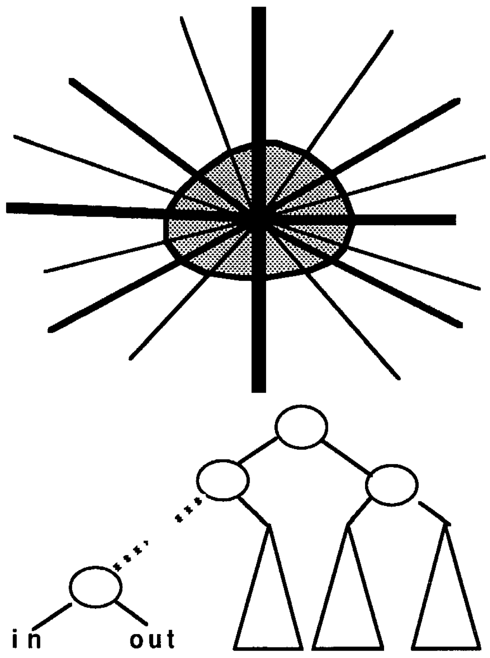
Now we are not claiming this representation is necessarily the optimal expected case representation, for in fact we do not know precisely what the optimal is. But we do claim, given the foregoing qualitative and quantitative characterization of goodness, that one should expect it to be near optimal.

Since the tree in Figure 5 has **log n** depth, it is worst case optimal for point classification [Preparata and Sharnos 85]. But so is the tree in Figure 6 which is a balanced binary tree with faces at the nodes lying at the end of each path and the hierarchy of non-face hyperplanes bisecting the set of faces lying in its region. However, it does not appear to provide a sequence of approximations. How does this difference appear quantitatively? For point classification, the expected case differs markedly. For figure 6, all cells are at depth **log n,** so the expected case is the same as the worst case regardless of the sample space. However, in figure 5, the top 3 out-cells would typically constitute most of the sample space, and so the expected case would converge to **O(l)** as the ratio of polygon-area/sample-area went to 0.



A good tree representation of a convex polygon
Figure 5

For line classification the two differ significantly both in the worst and expected cases. For figure 5, the work required to classify a line is the same as that required to classify the two points of intersection between the line and the polygon. Inserting the line partitions it into two nested sequences of intervals converging to the intersection points. Thus, the worst case is **O(log n)** while the expected case converges to **O(l**) as before. However, for figure 6, almost any line would intersect all of the radiating lines forming the partitioning, and so the worst and expected case is **0(n).** From these bounds on line classification, we can easily derive bounds for tree merging which is required for set operations. Tree merging can be accomplished by inserting the sub-hyperplanes of one tree into the second tree. For two trees/polygons of size **n,** if we treat the sub-hyperplanes as lines, **this** can take at most **O(n log n)** for figure 5 but **O(n^2)** for figure 6. Thus, what is good for point classification's expected case, is also good for line classification and set operations, viz. a set of approximations.



A poor tree representation of a convex polygon
Figure 6

General Tree Construction

To generate trees for arbitrary polytopes, we need a much more general method than the one just presented for convex polygons. Given a polytope represented by its faces (i.e. as a b-rep), we can construct a partitioning tree representation of that polytope using a top down recursive algorithm. At each stage in the algorithm, we have a region **r** containing the subset **s** of the boundary lying in **r.** If the subset is non-empty, we choose some hyperplane which intersects **r** and partition **s** into **s+** and **s-**, and then recurse with each of these new subsets. Otherwise we create a cell.

Brep to Bspt : Brep b => Bspt T

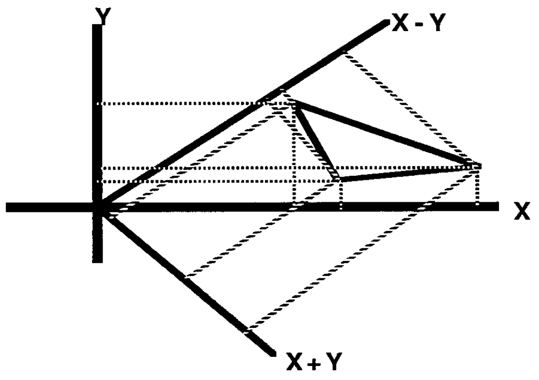
```
IF b = NULL
THEN
    T = a cell
ELSE
    h = Choose-Hyperplane( b )
    { $b^+$, $b^-$, $b^0$ } = Partition_Brep( b, h )
    T.faces = $b^0$
    T.pos_subtree = Brep_to_Bspt( $b^+$ )
    T.neg_subtree = Brep_to_Bspt( $b^-$ )
END
```

The central issue in constructing good trees is hyperplane selection. Generating the optimal tree, like most combinatorial optimization problems, appears intractable. Our problem can in fact be considered to be harder than those problems which require enumeration of the power set since we have ordering; thus, for **n** faces lying in **n** unique hyperplanes, there are at least **n!** different trees. Consequently, we have always employed heuristic methods (beginning with [Fuchs, Kedem and Naylor 80]). These have been in the form of a sum of positively weighted good attributes, such as degree of balancing, and negatively weighted bad attributes such as amount of face partitioning. In addition to these, we now employ directly the expected case models which measure the goodness of the tree. At each call to Brep_To_Bspt, we must chose a single partitioning hyperplane. Since we typically have no way of knowing the optimal hyperplane, our approach is to first generate a number of candidates that we would expect to be good ones, determine the location of a subset of the faces with respect to this hyperplane (locations are: negative halfspace, positive halfspace, both halfspaces, and on the hyperplane), and then select a hyperplane by estimating the goodness of each candidate using the expected case models. The candidates are generated using three different methods.

As stated earlier, all faces must lie in partitioning hyperplanes, and so the first method is to use face hyperplanes as in the original work of [Fuchs, Kedem and Naylor 80]. However, considering every possible candidate at each recursive call to the construction process is often too time consuming and usually not necessary for generating good trees. Instead, the number of such candidates is determined by a user specified cost factor. This cost factor is a mechanism for attempting to balance the cost of constructing the tree vs. the expected pay-back obtained during subsequent operations on the tree from having built a hopefully better tree. The cost factor determines directly the initial number of candidates, which varies from **log n** to **n.** The number of candidates **c** selected at each call as a percentage of the number of faces **f** lying in the current region is increased as a linear function of **f** until a predetermined threshold is reached, after which all face hyperplanes are chosen (currently 20). In addition, since we are interested in generating a multi-resolution representation, we bias the selection process by first sorting the face hyperplanes by area (each hyperplane is represented only once, and has with it a list of coincident faces). The candidates are then the first **c** on this sorted list.

A second and very important method of generating candidates is similar to techniques for constructing k-d trees [Bentley 75] which we have been using in various forms since the time of [Thibault and Naylor 87] and which were explored in various forms in [Toffes 90]. For each of a predefined number of directions, we project all of the vertices onto that direction and then sort them. We then consider hyperplanes orthogonal to this direction which contain vertices at certain positions in the ordering. The percentage of positions tested is treated similarly to that for choosing the number of face hyperplanes. The directions we are currently using correspond to the k-faces of a hypercube, whose number in 3D is 13 = 26/2 (see figure 7 for the 2D case). This is very similar to the schema for hyperplane generation used in [Kay and Kajiya 86], the principal difference being that we are not constructing bounding volumes; most of the hyperplanes intersect the polytope. This is a very important difference, with two consequences. First, using, for example, only the median and no face hyperplanes, we can generate a reasonable partitioning hyperplane in **0(n)** time which can be used to partition an initially large number of faces into smaller subsets with which we can apply more expensive and presumably better selection methods of complexity > **0(n).** Secondly, we are able to generate candidates that partition the interior of many-sided convex polytopes so that the tree is comparatively balanced. Using only face hyperplanes, any tree would be no more than a list of the faces, and so would not be an effective search structure. With our method, we can generate trees similar to the one in figure 5 automatically without having any specific knowledge that the set is convex.

The third method is similar to the second, but uses least squares fit to generate a direction. In particular, we compute the least squares fit of the set of vertices lying in the current region, and then use the normal of the resulting hyperplane as a new direction when applying the same techniques as used with the pre-defined directions.



Generating candidate hyperplanes by projection of vertices

Figure 7

Given the candidate set, we try to predict how effective each one will be based on the expected case models. In particular, given the region **r** which we are going to partition by a candidate, we can compute exactly p $\mathbf{p}^+$ and $\mathbf{p}^-$ for the given operation. However, we can only estimate $\mathbf{E_{cost}}$ $[\mathbf{T}^+]$ and $\mathbf{E_{cost}}$ $[\mathbf{T}^-]$**.** The estimators for these values can depend only upon a few simple properties of **n+** and **n-** such as number of faces and total surface area. We have chosen very simple estimators using as our model $\mathbf{n}^{a}$, where **a** is a value that depends upon the operation. Using a (currently small) sample set to determine these values, we have measured them to be (points = .4, line = .6, plane = .8). The number **n** is determined by computing the location of a subset of the faces, as mentioned earlier. The subset is chosen by sampling in a manner analogous to the sampling of face hyperplanes for use as candidates. Faces that lie in both halfspaces are added to the count for sides weighted by a penalty factor for splitting (the factor various between 1.5 and 2). We would anticipate being able to improve the quality of these estimators as we gain more understanding, but they seem even now to be reasonably effective.

Examples

We now show a few examples to illustrate the above ideas. The schema of representing convex polygons has been applied to constructing trees of 3D objects defined as surfaces of revolution [Ihm and Naylor 91]. The user first generates a "surface curve" which is to be revolved and a convex "path of revolution", both of which are given as a dense set of sample points (e.g. from digitization). We then construct good linear approximations using a sparse subset of the sample points. If the axis of revolution is thought of as being vertical, then we construct a balanced partitioning tree out of horizontal planes so that each segment of the surface curve lies between a pair of horizontal planes. The curve of revolution is represented as in figure 5, and the "revolution" of each segment is achieved by using this structure to represent the set between each pair of horizontal planes. See Picture 1 for an example of a goblet represented in this manner where the edges reveal all the intersections between sub-hyperplanes and the boundary.

In Picture 2, we show a geometric model of a phone handset, also with edges illustrating the partitioning. This was generated using constructive solid geometry starting from a few convex primitives each represented by a tree generated using the general b-rep to bspt conversion just described. The resulting tree was produced by using tree merging to provide set operations. By recognizing that the sound holes represent smaller features than the ear or mouth piece, the partitioning due to these holes is limited to the vicinity of their boundary.

In Picture 3, we illustrate the sequence of approximations idea. The original b-rep data was generated by triangulating between successive contours of CAT-scan sample points. A tree then was generated using the b-rep to tree conversion. The full resolution tree is shown as the largest head on the far right. The other three heads were created by pruning this tree, and are shown scaled to the approximate relative size that would be selected by a resolution sensitive renderer. Each of the successively smaller heads has a tree of about 1/2 the size of the next larger one. To decide how to prune a tree, we maintain for each region **r** an error estimate and classification defined as:

error = Var(membership over r) * Vol(r)

classification = E[membership] > 1/2 ? IN : OUT

Given an error threshold (manually selected in this example), a region was considered to be a cell if **error < threshold.** What we are showing in Picture 3 should not be considered as the images that would be generated by a resolution dependent rendering, since the simple thresholding to IN or OUT introduces too much high frequencies. Nor are we claiming our pruning strategy is the best one. Rather it is only meant to illustrate our contention that there is a correlation between expected case and sets of approximations.

Finally, as an illustration of the effects of using different probability models on tree construction, we give a table of values in which we used only one model to construct the tree (indicated by column headers), but then measure the resulting tree using all the models (indicated by row headings). The "old" method did not use expected case models. The values given are the expected number of internal nodes visited by each operation, with the last row being the total number of nodes in the tree. The test object was the space shuttle (Picture 4).

Construction	Point	Line	Plane	Old
Measured				
Point	7.96	8.12	8.56	11.7
Line	16.75	15.7	18.43	25.8
Plane	193	124	113	141
Nodes	1552	1173	1073	767

As one can see from this one example, the specific probability **model** applied effects the resulting tree is a way that yields the least cost tree for the modeled operation when compared to the other models. Also, the use of expected case, when compared to previous methods, tends to produce larger trees in order to achieve better search structures.

CONCLUDING REMARKS

Like many combinatorial optimization problems, direct generation of the optimal is not an alternative, and so the question of how to best arrive somewhere in the vicinity of the optimal using a set of local decisions is always an open question (note how many solutions to the Traveling Salesman Problem have been proposed). We certainly have not closed the topic with this paper. But we do feel that the notion of an ordered set of approximations is the right one and generating trees using expected case models is effective.

As for future work along these lines, the idea which we are most interested in pursuing is seeing what can be done with reorganizing an already constructed tree without having to first convert back to a b-rep. Since the 1D balancing techniques for AVL trees and Red-Black trees do not apply directly in dimensions greater than 1, it is an open question as to what can be done.

REFERENCES

[Bentley 75]
John L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching", CACM Vol. 19, (Sept. 1975).

[Fuchs, Kedem, and Naylor 80]
H. Fuchs, Z. Kedem, and B. Naylor, "On Visible Surface Generation by a Priori Tree Structures," **Computer Graphics** Vol. 14(3), pp. 124-133, (June 1980).

[Goldsmith and Salmon 87]
Jeffery Goldsmith and John Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing", **IEEE** CG&A vol. 7(3), pp. 14-20, (May 1987).

[Ihm and Naylor 91]
Insung Ihm and Bruce Naylor, "Piecewise Linear Approximations of Curves with Applications," Proceeding of Computer Graphics International '91, Springer-Verlag (June 1991).

[Kay and Kajiya 86]
Timothy L. Kay and James T. Kajiya, "Ray Tracing Complex Scenes," **Computer Graphics** Vol. 20(4), pp. 269-278, (June 1986).

[Naylor 81]
Bruce F. Naylor, "A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes," Ph.D. Thesis, University of Texas at Dallas (May 1981).

[Naylor and Thibault 86]
Bruce F. Naylor and William C. Thibault, "Application of BSP Trees to Ray-Tracing and CSG Evaluation", Technical Report GIT-ICS 86/03, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia 30332 (February 1986).

[Naylor, Amanatides and Thibault 90]
Bruce F. Naylor, John Amanatides and William C. Thibault, "Merging BSP Trees Yields Polyhedral Set Operations", **Computer Graphics** Vol. 24(4), pp. 115-124, (August 1990).

[Naylor 92]
Bruce F. Naylor, "Interactive Solid Geometry Via Partitioning Trees", **Graphics Interface,** pp. 11-18 (May 1992).

[Preparata and Shamos 85]
Franco P. Preparata and Michael Ian Shamos, "Computational Geometry: An Introduction", Springer-Verlag, 1989.

[Paterson and Yao 89]
M.S. Paterson and F.F. Yao, "Binary partitions with applications to hidden-surface removal and solid modeling", Proceedings of Fifth Symp. on Computational Geometry, pp. 23-32, 1989.

[Rabin 72]
Michael 0. Rabin, "Proving Simultaneous Positivity of Linear Forms", **Journal of Computer and Systems Science,** Vol. 6, pp. 639-650 (1991).

[Schumacker et al 69]
R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp, "Study for Applying Computer-Generated Images to Visual Simulation," AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory (1969).

[Sutherland 73]
Ivan Sutherland "Polygon Sorting by Subdivision: A Solution to the Hidden-Surface Problem", unpublished manuscript.

[Thibault and Naylor 87]
W. Thibault and B. Naylor, "Set Operations On Polyhedra Using Binary Space Partitioning Trees", **Computer Graphics** Vol. 21(4), pp. 153-162, (July 1987).

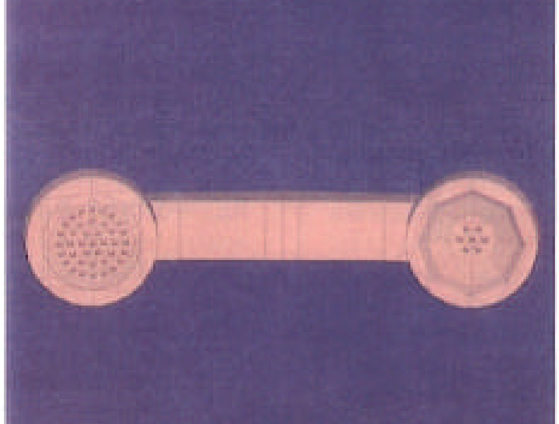
[Torres 90]
Eric Torres, "Optimization of the Binary Space Partition Algorithm (BSP) for the Visualization of Dynamic Scenes", Eurographics, (Sept. 1990).

[Turk 92]
Greg Turk, "Re-Tiling Polygonal Surfaces", **Computer Graphics** Vol. 26(2), pp. 153-64, (July 1992).

[Willard 82]
D.E. Willard, "Polygon Retrieval", **SIAM Journal on Computing,** Vol. 11, pp. 149-165, (July 1987).

Picture 1

Picture 2



Picture 3

Picture 4