BINTREES, CSG TREES, AND TIME

Hanan Samet
Computer Science Department
University of Maryland, College Park, MD 20742

Markku Tamminen
Laboratory for Information Processing Science
Helsinki University of Technology, 02150 Espoo 15, Finland

ABSTRACT

A discussion is presented of the relationship between two solid representation schemes: constructive solid geometry (CSG trees) and recursive spatial subdivision exemplified by the bintree, a generalization of the quadtree and octree. Detailed algorithms are developed and analyzed for evaluating CSG trees by bintree conversion, i.e., by determining explicitly which parts of space are solid and which empty. These techniques enable the addition of the time dimension and motion to the approximate analysis of CSG trees in a simple manner to solve problems such as dynamic interference detection. For "well-behaved" CSG trees, the execution time of the conversion algorithm is directly related to the spatial complexity of the object represented by the CSG tree (i.e., asymptotically it is proportional to the number of bintree nodes as the resolution increases). The set of well-behaved CSG trees includes all trees that define multidimensional polyhedra in a manner that does not give rise to tangential intersections at CSG tree nodes.

CR Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling – solid and object representations; geometric algorithms and systems; I.3.3 [**Computer Graphics**]: Picture/Image Generation – display algorithms; viewing algorithms

General Terms: Algorithms, Data Structures

Additional Key Words and Phrases: constructive solid geometry (CSG), time, motion, conversion, image processing, hierarchical data structures, bintrees, quadtrees, octrees, interference detection, solid modeling

1. INTRODUCTION

Constructive solid geometry (CSG) uses trees (CSG trees) of building block primitives (parallelepipeds, spheres, cylinders, ...), combined by geometric transformations and Boolean set operations as a representation of three-dimensional solid objects [13]. Each primitive solid can be decomposed into a subtree whose leaves are halfspaces, each described by an equation of the form:

$$f(x,y,z) \geq 0.$$

Substituting this subtree for every occurrence of that primitive in the original CSG tree gives rise to an expanded tree having only halfspaces as leaves. In the present article we shall assume this simple halfspace formulation of CSG (see also [12,21]). Clearly, the CSG approach can be used to describe objects of any dimensionality and many interesting solid modelers have been based on it [14].

It has been known for some time that octree-like recursive subdivision can facilitate the evaluation of CSG trees; e.g., the analysis [8,20] and display [21] of solid objects modeled by them. A hardware processor with such a capability is described by Meagher [10]. A bintree represents discrete solid objects of arbitrary dimensionality (e.g., binary images in two dimensions) by a binary tree defining a recursive subdivision of space and recording which parts are empty (WHITE) and which are solid (BLACK). The bintree is a dimension-independent variant of the more familiar quadtree and octree representations. For example, Figure 1c is a bintree corresponding to a 2-dimensional binary image (Figure 1a) consisting of two regions. See the survey of Samet [15] for a comprehensive survey and bibliography of quadtree related methods. Mentions of arbitrary dimensionality are found in the literature [4,5,9,22] but few concrete applications have been demonstrated.

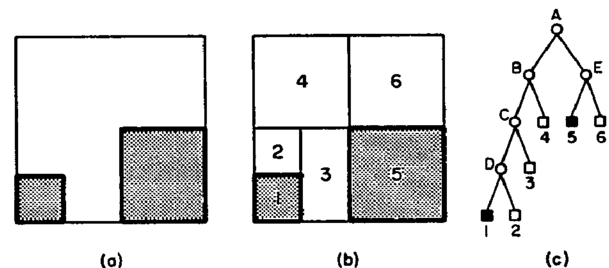


Figure 1. Sample image and its bintree. (a) Image (b) Block decomposition (c) Bintree.

Time and motion are important elements of advanced solid modeling. In particular, given a moving object we may wish to determine whether it intersects a stationary object *(static interference detection)* or whether it intersects another moving object *(dynamic interference detection)*. Even though it appears that the time dimension can be added to CSG trees in a conceptually simple fashion, rather little attention has

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1985 ACM 0-89791-166-0/85/007/0121 $00.75

been focussed on CSG trees in a dynamic situation. Perhaps this is due to the difficulty of evaluation in the now four-dimensional space.

Static interference detection is discussed by Boyse [3] but only boundary representations are considered. Tilove [19] has provided a good analysis of the equivalent "NULL object detection" problem in the CSG setting. Our work seems to complement that of Tilove. We do not repeat his formal analysis of the "pruning" of CSG trees but show in detail how a CSG tree can be efficiently pruned against an adaptable grid (i.e., bintree), even in the case of non-bounded halfspace primitives. We also show that for "well-behaved" CSG trees, the amount of work involved in pruning the CSG tree against all the cells of such an adaptable grid is asymptotically proportional to the number of cells and does not depend on the number of nodes in the CSG tree representation.

In the rest of this paper we show how bintree conversion provides an efficient and dimension-independent tool for evaluating CSG trees. The time dimension is handled without extra conceptual overhead. Our emphasis is on CSG trees defined by linear halfspaces and on motion along a piecewise linear trajectory but the techniques are shown to extend to the non-linear case. We present and analyze the evaluation (conversion) algorithm and show that asymptotically, as resolution is increased, the amount of work it involves is directly related to the spatial complexity of the object represented by the CSG tree. Thus, despite the added dimension, dynamic interference detection by bintree conversion is often efficient (because the object sought is the null object). Finally, we present some experimental results obtained by using the discrete solid modeler described in [18]. The simplicity of the algorithms is striking when compared with algorithms for converting boundary representations to bintrees [17].

2. DEFINITIONS

In our algorithms we use a linear tree representation that is based on the preorder traversal of a bintree. The traversal yields a string over the alphabet "(", "B", "W" corresponding respectively to internal nodes, BLACK leaves, and WHITE leaves. This string is called a *DF-expression* [6]. For the image of Figure 1 it becomes ((((BWWW(BW. Note that bintrees are size-independent: i.e., a given tree can define an object in a universe of any size. However, we usually portray each bintree as embedded in the d-dimensional unit cube. Let us say that the *resolution* of a bintree, say M, is the maximal number of units into which each side of the d-dimensional universe of a d-dimensional bintree can be divided. A cube of side length 1/M is called a *voxel*.

A point x in a d-dimensional universe (d-space) is represented by a row vector $x_0, x_1, \cdots,$and x_d of $d+1$ homogeneous coordinates with x_0 denoting the scale factor. We shall only consider the case with $x_0=1$. In the general case, the d ordinary Euclidean coordinates are obtained by dividing $x_1, \cdots, x_d$ by x_0. Note that usually the scale factor is taken to be the last component of x. With our choice, the scale factor retains its original index when the time dimension is added.

A (linear) halfspace in d-space is defined by an inequality on the $d+1$ homogeneous coordinates:

$$\sum_{i=0}^{d} a_i \cdot x_i \geq 0 \qquad (1)$$

The halfspace is represented by a column vector a. In vector notation (1) is written as $a \cdot x \geq 0$, with column vector a representing the halfspace. Figure 2 shows the halfspace represented by $4x-2y-1 \geq 0$. The point set satisfying this relation lies to the right of the line. (partially shaded). Given a point x, the value of the left side of (1) at x is called the *value* of halfspace a at x.

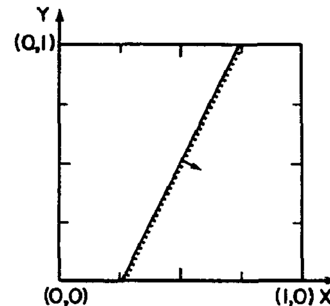


Figure 2. Halfspace corresponding to $4x-2y-1 \geq 0$.

We shall concentrate on CSG trees in the simplest of settings, that of halfspaces defined by hyperplanes (linear halfspaces). Arbitrary CSG tree can be approximated in this way [2,21]. Also, qualitatively, our methods extend to the general case.

In this article we define a data structure for CSG trees as follows. A CSG tree is a binary tree in which internal nodes correspond to geometric transformations and Boolean set operations while leaves correspond to halfspaces.* A node of a CSG tree is described by a record of type *csgnode* with six fields. The first two fields, LEFT and RIGHT, contain pointers to the node's left and right sons respectively. The TYP field indicates the node's type. There are five node types - UNION, INTERSECTION, BLACK, WHITE, and HALFSPACE. Types UNION and INTERSECTION correspond to the Boolean set operations. HALFSPACE corresponds to a leaf (i.e., halfspace). The field HSP contains an identifier for the halfspace. It is an index to a table, HS, of $d+1$ element coefficient vectors of the different halfspaces involved in the CSG tree. The remaining two fields, MIN and MAX, are used for auxiliary data in our algorithms. They record the minimum and maximum values, respectively, of a halfspace in a given bintree block.

Note that our definition of a CSG tree allows for leaves that are completely BLACK or WHITE as required in our algorithm. In addition, in contrast to the conventional use of CSG, we only use the Boolean set operations UNION and INTERSECTION, as the effect of the third one, MINUS, can be achieved by application of De Morgan's laws to all nodes of type UNION and INTERSECTION. The complement of a halfspace is obtained by changing the signs of all the coefficients (i.e., the direction of its normal). Note that our universe is finite, as required by the bintree representation.

3. CONVERTING CSG TREES TO BINTREES

Our algorithms traverse the universe in a depth-first manner and evaluate each successive subuniverse against the CSG tree. This enables pruning areas of no interest. Whenever

*Our discussion assumes that the transformations have been propagated to the leaves. We also assume a bounded universe, for simplicity in the form of the unit cube. Actually, so-called regularized versions of the set operations must be used [13]. However, we shall not repeat the term regularized.

the hyperplane of a halfspace, say H, passes through a subuniverse (i.e., a bintree node), say S, then we say that H is *active* in S (i.e., there exists a point in S such that $a \cdot x = 0$). A CSG tree node is said to be *active* in S if both of its sons are active in S. As an example of the use of these terms, consider the conversion of the triangle given in Figure 3 whose CSG tree is given in Figure 4. It is composed of the intersection of the three halfspaces $2x-1 \geq 0$, $2y-1 \geq 0$, and $-2x-2y+3 \geq 0$ labeled A, B, and C respectively. Conversion starts with the unit square universe. In this case halfspaces A, B, and C are all active and so is the CSG tree (in other words, it is not totally BLACK or WHITE). Thus we first have to split the unit square into two halves (split along the x coordinate). Now, evaluating the CSG tree against the left half of the bintree, say L, we find that A is WHITE and thus not active. Therefore, L will have to be WHITE since A is combined with the rest of the CSG tree by an INTERSECTION node and the intersection of WHITE with anything is WHITE.

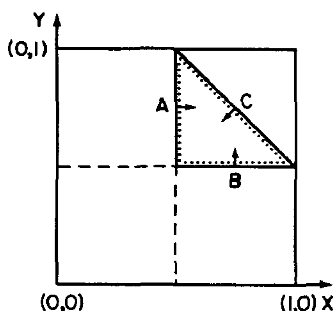


Figure 3. Sample triangle image.

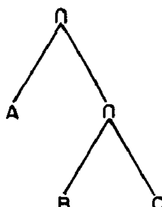


Figure 4. CSG tree corresponding to Figure 3.

First, let us examine the construction of a bintree corresponding to a halfspace as given by (1). This is achieved by traversing the universe in the DF-order and determining the range of the left side of (1) in each subuniverse. Each node in the bintree in which the halfspace is active is decomposed into two sons and the process is recursively applied to them. The process stops when the halfspace is not active in a bintree node or if the bintree node corresponds to a voxel. All voxels in which the halfspace is active are labeled BLACK in the version of the algorithm presented here. The resulting bintree is represented using a DF-expression.

Determining whether a halfspace is active in a bintree node is facilitated by keeping track of the minimum and maximum values of $a \cdot x$ for each bintree node. Whenever the maximum is ≤ 0 the bintree node is WHITE and when the minimum is ≥ 0 it is BLACK. Otherwise the halfspace is active and subdivision is required. Initially, for the unit cube, the minimum value of $a \cdot x$ is the constant factor of a plus the sum of all negative coefficients in a. The maximum value is the constant factor of a plus the sum of all positive coefficients in a. For example, for Figure 2, the initial minimum value is -3 and the initial maximum value is 3.

Whenever a bintree node is subdivided, either the maximum or minimum (never both at the same time) of $a \cdot x$ for each son node changes with respect to that of the father. Let the subdivision be performed on a hyperplane (e.g., a line in two dimensions) perpendicular to the axis corresponding to coordinate i ($1 \leq i \leq d$) and let w_i be the width of the side along coordinate i of the block resulting from the subdivision. The amount of change is $\delta_i = a_i \cdot w_i$. For the left son, if $\delta_i > 0$, then δ_i is subtracted from the maximum; otherwise, δ_i is subtracted from the minimum. For the right son, if $\delta_i > 0$, then the minimum is incremented by δ_i; otherwise, δ_i is added to the maximum.

As an example, consider again the halfspace given by $4x-2y-1 \geq 0$ as shown in Figure 2. Assume that the universe is the unit square. The maximum and minimum values 3 and -3 are attained respectively at (1,0) and at (0,1). Subdividing along the x axis yields two sons. The maximum value of $a \cdot x$ in the left son has decreased by 1/2 times the coefficient of the x coordinate (i.e., 2) to 1 and is attained at (0.5,0), while the minimum value remains the same. The minimum value of $a \cdot x$ in the right son has increased by 1/2 times the coefficient of the x coordinate (i.e., 2) to -1 and is attained at (0.5,1) while the maximum value remains the same.

A CSG tree is evaluated, i.e., converted, to a bintree by traversing the universe in depth-first order and evaluating each subuniverse against the CSG tree. Leaf nodes (halfspaces) are evaluated using the method described above and their results are combined by *pruning* the CSG tree to the subuniverse. Pruning means that only that part of the CSG tree which is active within the subuniverse is retained [19,21]. Once pruning has reduced the CSG tree to a leaf node (i.e., a halfspace), the conversion procedure becomes identical to that described above for converting a halfspace to a bintree. Each node in the bintree, say B, in which the CSG tree is active, is decomposed into two sons and they are in turn intersected with only that part of the CSG tree that is active in B. This process stops when the CSG tree is not active in a bintree node or if the bintree node corresponds to a voxel. All voxels in which a CSG tree is active are labeled by procedure CLASSIFY_VOXEL which we do not describe here. At its simplest (as used in the experiments described in Section 6), it treats all such voxels as BLACK (or WHITE). At its most complex, CLASSIFY_VOXEL corresponds to Tilove's NULL object algorithm applied to the active CSG subtree at the voxel [19].

The conversion of a CSG tree to a bintree is performed by procedures CSG_TO_BINTREE, INIT_HALFSPACES, CSG_TRAVERSE, PRUNE, and HSPEVAL given below. They make use of BLACK_CSG_NODE and WHITE_CSG_NODE which are global pointers to BLACK and WHITE CSG tree nodes. In these and all other procedures, we shall use the following global constants: D is the dimensionality of the space, VOXEL_LEVEL is the level of the bintree corresponding to voxels, and EPSILON is the tolerance for deciding when a bintree node is WHITE (i.e., at most a proportion EPSILON of its "critical" diagonal is contained in the halfspace) or BLACK.

CSG_TO_BINTREE serves to initialize the traversal process. First, it invokes INIT_HALFSPACES to traverse the CSG tree to compute the minimum and maximum values of each halfspace in the whole universe (i.e., the unit cube). These

values are stored in the MIN and MAX fields of the CSG tree node corresponding to each halfspace. Next, it calls on CSG_TRAVERSE to perform the actual conversion. CSG_TRAVERSE traverses the universe by recursively subdividing it corresponding to the depth-first traversal order of the resulting bintree. At each subdivision step, PRUNE is called to attempt to reduce the size of the CSG tree which will be evaluated in the bintree block. PRUNE traverses the CSG tree in depth-first order and removes inactive CSG nodes with the aid of HSPEVAL which determines if a halfspace is active within a given bintree block. Assuming that T is CSG node, PRUNE applies the following four rules to the CSG tree.

(1) BLACK UNION T = BLACK
(2) WHITE UNION T = T
(3) BLACK INTERSECTION T = T
(4) WHITE INTERSECTION T = WHITE.

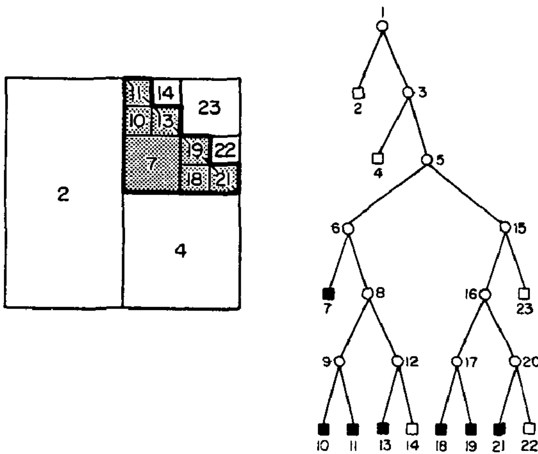


Figure 5. Bintree corresponding to the triangle in Figure 3 before collapsing.

As an example, consider the triangle of Figure 3 whose CSG tree is given in Figure 4. Figure 5 is the corresponding bintree, with VOXEL_LEVEL=6. The nodes have been numbered in the order in which they were output. Initially, the entire CSG tree (i.e., Figure 4) is assumed to be active in the whole universe (i.e., the unit square). Node 1 is output as a NON-LEAF and we process its left son next. First, we attempt to prune the CSG tree with respect to the left half of the universe. Since halfspace A is inactive here (i.e., it is WHITE), we can apply pruning rule (4) and there is no need to further process the remainder of the CSG tree in Figure 4. We output node 2 as WHITE and process the right son of node 1 next. Pruning the CSG tree shows that halfspace A is again inactive but this time it is BLACK. Since both halfspaces B and C are active here, pruning rule (3) leaves us with the CSG tree given by Figure 6. We now output node 3 as NON-LEAF and process its left son next. Pruning the CSG tree results in halfspace B being inactive (i.e., it is WHITE) and pruning rule (4) means that there is no need to further process this CSG tree. We output node 4 as WHITE and process the right son of node 3 next. This time pruning the CSG tree results in halfspace B being inactive again but now it is BLACK. Pruning rule (3) leaves us with just halfspace C. Node 5 is output as NON-LEAF and the conversion process is applied to its two sons next.

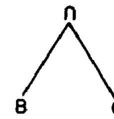


Figure 6. Result of pruning the CSG tree of Figure 4 in the right half of the root of its bintree.

The remainder of the conversion is equivalent to that described earlier for the conversion of a halfspace as the CSG tree has been reduced to one halfspace. The result is given in Figure 5. Note that the sequence of nodes that is output is not minimal in the sense that collapsing may yet have to be performed (i.e., when two terminal brother nodes are BLACK). Application of collapsing results in merging nodes 10 and 11, and nodes 18 and 19.

```
procedure CSG_TO_BINTREE(P,N,HS);
/* Convert the D-dimensional CSG tree pointed at by P to a
   bintree. HS contains N halfspaces. */
begin
 value pointer csgnode P;
 global value integer N;
 global value real array HS[1:N,0:D];
 INIT_HALFSPACES(P);
 CSG_TRAVERSE(P,0,1.0);
end;
```

```
procedure INIT_HALFSPACES(P);
/* Compute the minimum and maximum values of each of the
   halfspaces of the CSG tree P in the D-dimensional unit
   universe. */
begin
 value pointer csgnode P;
 integer I,J;
 if TYP(P)='HALFSPACE' then
  begin
   I←HSP(P);
   MIN(P)←MAX(P)←HS[I,0];
   for J←1 step 1 until D do
    begin
     if HS[I,J]>0.0 then MAX(P)←MAX(P)+HS[I,J]
     else MIN(P)←MIN(P)+HS[I,J];
    end;
  end
 else
  begin
   INIT_HALFSPACES(LEFT(P));
   INIT_HALFSPACES(RIGHT(P));
  end;
end;
```

```
procedure CSG_TRAVERSE(P,LEV,W);
/* Convert the portion of the CSG tree P that overlaps the
   D-dimensional subuniverse of volume 2^-LEV whose smallest
   side has width W. The bintree is constructed by evaluat-
   ing the CSG tree in the subuniverse. The evaluation pro-
   cess consists of pruning the nodes of the CSG tree that are
   outside of the subuniverse. A new copy of the relevant
   part of the CSG tree is created as each level is descended
   in the bintree. This storage is reclaimed once a
   subuniverse at a given level has been processed. */
```

```
begin
 value pointer csgnode P;
 value integer LEV;
 value real W;
 pointer csgnode FS; /* Pointer to stack of free nodes */
 if TYP(P)='BLACK' or TYP(P)='WHITE' then
  output(TYP(P))
 else if LEV=VOXEL_LEVEL then
  output(CLASSIFY_VOXEL(P));
 else /* Subdivide and prune the CSG trees */
  begin
   FS←first_free(csgnode);
    /* Save pointer to free storage stack */
   output('NON-LEAF');
   if LEV mod D=0 then W←W/2;
   CSG_TRAVERSE(PRUNE(P,LEV+1,W,'LEFT'),LEV+1,W);
   free(FS); /* Release storage for CSG tree nodes */
   CSG_TRAVERSE(PRUNE(P,LEV+1,W,'RIGHT'),LEV+1,W);
   free(FS);
  end;
end;
```

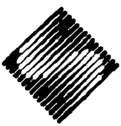
```
pointer csgnode procedure PRUNE(P,LEV,W,DIR);
/* Evaluate the portion of the CSG tree P that overlaps the
   D-dimensional subuniverse of volume 2^-LEV whose smallest
   side has width W. The subuniverse corresponds to the DIR
   (LEFT or RIGHT) subtree of its father bintree node. */
begin
 value pointer csgnode P;
 value integer LEV;
 value real W;
 value direction DIR;
 pointer csgnode T,Q; /* Auxiliary variables */
 pointer csgnode L,R;
  /* Auxiliary pointers to left and right pruned subtrees */
 if TYP(P)='HALFSPACE' then
  return(HSPEVAL(P,LEV,W,DIR))
 else
  begin
   T←if TYP(P)='UNION' then BLACK_CSG_NODE
      else if TYP(P)='INTERSECTION' then
       WHITE_CSG_NODE
      else <error>; /* Enable a quick application of prun-
      ing rules (1) and (4) */
   L←PRUNE(LEFT(P),LEV,W,DIR);
   if L=T then return(T)
   else
    begin
     R←PRUNE(RIGHT(P),LEV,W,DIR);
     if R=T then return(T)
     else if TYP(L)=OPPOSITE(TYP(T)) then return(R)
      /* OPPOSITE of BLACK is WHITE and vice
         versa */
     else if TYP(R)=OPPOSITE(TYP(T)) then return(L)
     else    /* Evaluation has not resulted in eliminating */
      begin /* one of P's sons */
       Q←create(csgnode);
       TYP(Q)←TYP(P);
       LEFT(Q)←L;
       RIGHT(Q)←R;
       return(Q);
      end;
    end;
  end;
end;
```

```
pointer csgnode procedure HSPEVAL(P,LEV,W,DIR);
/* Determine if the D-dimensional subuniverse of volume
   2^-LEV and smallest side of width W intersects halfspace P
   or corresponds to a BLACK or WHITE region. The
   subuniverse is the DIR (LEFT or RIGHT) subtree of its
   father. If the halfspace intersects the subuniverse, then the
   subuniverse will have to be subdivided again and a new
   CSG tree node is allocated for the halfspace to record the
   new minimum and maximum values of the halfspace. */
begin
 value pointer csgnode P;
 value integer LEV;
 value real W;
 value direction DIR;
 integer I,J;
 real DELTA;
 pointer csgnode Q;
 Q←create_and_copy(P);
 J←HSP(P);
 I←LEV mod D;
 DELTA←HS[J,I+1]*W;
 if DIR='LEFT' then
  begin
   if DELTA≤0 then MIN(Q)←MIN(Q)-DELTA
   else MAX(Q)←MAX(Q)-DELTA;
  end
 else
  begin
   if DELTA>0 then MIN(Q)←MIN(Q)+DELTA
   else MAX(Q)←MAX(Q)+DELTA;
  end;
 if MIN(Q)>-EPSILON then return(BLACK_CSG_NODE)
 else if MAX(Q)<EPSILON then
  return(WHITE_CSG_NODE)
 else return(Q); /* The subuniverse is intersected */
end;
```

4. TIME AND MOTION

Often a geometric representation, such as CSG, is not convenient for a desired computation. The solution that is frequently adopted is to transform the object into another representation - i.e., one in which the computation is simpler. In the previous section we saw how a CSG representation can be converted to a bintree. In this section we show how the time dimension can be added to a CSG representation so that motion can be analyzed using the algorithm of the previous section.

Let T be a solid model described by a CSG tree and assume that it is defined in some model-specific coordinate system. We describe the motion of T in some common world coordinate system by a time-varying geometric transformation matrix $A(t)$. Each value of $A(t)$ is a matrix defining a rigid motion from the local coordinates of T to its position and orientation in world coordinates at time t. Note that if our world coordinate system is the unit cube, then we may also have to include a scaling in $A(t)$. We call $A(t)$ the *trajectory* of T. Let $A(t)$ be piecewise linear, meaning that it can be broken down into a series of segments defined by time points $(t_0, t_1, \cdots)$ so that $A_{i+1}=A_i B_i$, where B_i is a transformation matrix corresponding to a translation describing the motion during that time segment. In the following we discuss the motion accomplished in one time segment in a more concrete setting.



Translating a halfspace, say given by (1), along a vector v gives rise to the translated halfspace

$$\sum_{i=0}^{d} a_i \cdot x_i - \sum_{i=0}^{d} a_i \cdot v_i \geq 0 \tag{2}$$

If point x satisfies (1), then the transformed (translated) point $x+v$ satisfies (2). In order to be dimensionally-consistent with the d-dimensional unit cube, our discussion always assumes a unit time interval. Motion in a unit time interval, and at a fixed speed defined by vector s with $s_0=0$, is described by a vector v such that for all i, $v_i=s_i \cdot t$. Thus using v to translate halfspace (1) we find that at each instant, say t, it corresponds to the halfspace given by (3), below. Letting t vary, we obtain a linear halfspace with an additional variable t.

$$\sum_{i=0}^{d} a_i \cdot x_i - (\sum_{i=0}^{d} a_i \cdot s_i) \cdot t \geq 0 \tag{3}$$

When we have a CSG tree in motion, transformation (3) can be applied to each halfspace separately and the tree of Boolean set operations applied to the resulting $(d+1)$-dimensional halfspaces to define a set of points in (location,time)-space satisfying the CSG tree.

For dynamic interference detection, we must determine whether the intersection of two (location,time) objects is empty while for static interference detection, we must check whether two stationary objects intersect or whether a moving object intersects a stationary one. The intersection of two different (location,time) CSG trees, (each derived from a separate motion but with a common "time axis") is obtained by attaching them as sons to a newly created CSG node of type INTERSECTION. The actual evaluation of the intersection can be performed by applying the bintree conversion algorithm of Section 3 in the $(d+1)$-dimensional space with time included. For static interference detection there is no need to add time as an extra dimension if we can otherwise solve for the swept area of the moving object. Note also that in the case of interference detection there is often little motivation for storing the entire resulting tree. Instead, a variable can be included in the tree traversal algorithm to indicate the minimal t value of a BLACK node encountered so far in the traversal. Any subtree whose minimum value of t is greater than this value need not be inspected.

Usually primary interest is not in motion along a straight vector but in more complicated trajectories. Assume that such trajectories can be approximated by a sequence of segments, each with motion corresponding to a linear translation at a fixed speed. For example, suppose that we wish to determine whether two motions, defined by piecewise linear trajectories A_{1i} and A_{2i} of objects T_1 and T_2, respectively, intersect in the unit time interval. Let the time intervals defining the n_1 and n_2 linear pieces of the trajectories be $(t_{10},t_{11},\cdots)$ and $(t_{20},t_{21},\cdots)$, respectively. Now, during the time interval $(0,\min(t_{10},t_{20}))$ both motions are linear and their intersection can be determined as discussed above. This same procedure is applied to the remaining intervals (a maximum of n_1+n_2-1 intervals) each preceded by an application of the appropriate transformations A_{ij} to the halfspaces of T_1 and T_2.

In the general case, a CSG tree can contain non-linear halfspaces or the motion itself cannot be described as a series of translations. Nevertheless, we can still use methods similar to the ones described above. In particular, each bintree node corresponds to a $(d+1)$-dimensional *interval* such that $x_0 \leq x < x_1$, $y_0 \leq y < y_1$, $\cdots$, $t_0 \leq t < t_1$. *Interval arithmetic* is a method of evaluating functions $f(x,y,\cdots)$ in cases where the arguments are not exact values but intervals. The value of the interval function corresponding to function f is also an interval - i.e., a range covering any values that f can obtain given as arguments any values in the argument intervals. It should be clear that interval arithmetic is appropriate for the CSG tree to bintree conversion process since for an arbitrary function the value of the corresponding interval function covers the function's possible values in the bintree node. If zero does not belong to this range, then the bintree block need not be subdivided.

For example, let us apply the above to determine the (location,time) bintree of a linear halfspace a subjected to arbitrary motion defined by the matrix function $A(t)$. Denoting intervals by capital letters, the interval function that must be evaluated at each node of the bintree is $F(X,T)=(A^{-1}(T)a)\cdot X$ where X is an interval in d-dimensional space and T is a time interval. Remember that at each time instant t, $A(t)$ is a linear transformation and the image of a halfspace, say a, is obtained by multiplying a by the inverse of $A(t)$. When $A(t)$ contains a rotation, the interval function will be a linear composition of sine and cosine functions with respect to T. In any sub-interval of T, where the composite function is monotonic, interval arithmetic can be applied in a fashion to provide tight bounds for the resulting interval.

Interval arithmetic is easy to incorporate in our CSG tree to bintree conversion since we merely need to recast procedure HSPEVAL in terms of interval evaluations. Procedure INT_HSPEVAL given below achieves this and can be substituted for procedure HSPEVAL of Section 3 in procedure PRUNE. Notice the use of INTERVAL_EVALUATE to determine the range of the function corresponding to the non-linear halfspace. Its value is a pointer to a record of type *interval* with two fields MIN and MAX corresponding to an interval covering the function values in the node.

```
pointer csgnode procedure INT_HSPEVAL(P,LEV,W,DIR);
/* Use interval arithmetic to determine if the D-dimensional
   subuniverse of volume 2^-LEV intersects non-linear halfspace
   P or is BLACK or WHITE. W is its smallest side. The
   subuniverse is the DIR subtree of its father. */
begin
 value pointer csgnode P; /* A leaf of the CSG tree */
 value integer LEV;
 value real W;
 value direction DIR;
 interval I;
 I←INTERVAL_EVALUATE(P,LEV,W,DIR);
 return(if MAX(I)≤EPSILON then WHITE_CSG_NODE
        else if MIN(I)≥-EPSILON then
          BLACK_CSG_NODE
        else 'HALFSPACE');
end;
```

Interval arithmetic has been applied to the somewhat similar task of evaluating curved surfaces by recursive subdivision [1,11]. Nevertheless, this technique should be used with caution. In particular, interval arithmetic does not necessarily yield the minimal range covering the function's values given the domains of the arguments; instead, it may be a wider interval guaranteed to cover the function's values. This

estimate may sometimes be very poor, and the poorer the substitute for the true range of function values, the more unnecessary subdivision we must perform in the bintree conversion. There are many function transformations that can be applied to tighten the ranges [1].

Usually we are not interested in time as such. Often it merely serves as an auxiliary variable for describing motion. For instance, the process of determining the swept area for static interference detection is equivalent to a transformation that eliminates the time dimension. In geometric terms it is a projection parallel to the t-axis. In general, we do not know how to perform such a projection directly in the CSG representation. Given a CSG tree having A OP B as its root, we cannot necessarily distribute the projection operation - i.e.,

$$\text{PROJ(A OP B)} \neq \text{PROJ(A) OP PROJ(B)}$$

For example, suppose we are given two non-intersecting objects as in Figure 7 that are moving at identical speeds in the direction of the x-axis. Clearly, their swept areas intersect whereas the objects themselves do not intersect.

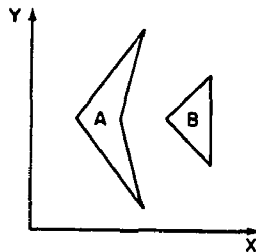


Figure 7. Example of two disjoint objects A and B whose projection on the y-axis is non-empty.

Fortunately, projection in the discrete bintree domain is simple. Approximate evaluation of a CSG tree involving a projection operation is a two-step process. We first generate the $(d+1)$-dimensional bintree and then project it to d dimensions to obtain an evaluation of the projected CSG tree as a d-dimensional bintree (see [16]). Projection consists of eliminating one coordinate and keeping track of all occupied locations in the resulting d-dimensional space. In three dimensions, the projection algorithm is almost identical to that for viewing a three-dimensional bintree in the direction of a coordinate axis [18]. The only difference is that in viewing, some shading information must be recorded at each 2-d pixel that is "covered", whereas the projection discussed here only records whether or not such a pixel is covered.

5. ANALYSIS

A quick perusal of procedure CSG_TO_BINTREE, as given in Section 3, reveals that the amount of work performed in the conversion is proportional to the sum of the sizes of the CSG trees that are active at the bintree nodes (i.e., blocks) that are evaluated. This number can be quite large even though procedure CSG_TRAVERSE attempts to prune the CSG tree each time it descends to a deeper level in the tree. However, in a typical case, as we descend in the bintree, many of the CSG tree nodes are no longer active thereby reducing the number of CSG nodes that must be visited. We are not interested in the absolute worst-case value of the complexity. Instead, we shall focus on the "practical" efficiency of these algorithms. Very poor cases can be attained by constructing a complicated CSG tree that evaluates to the NULL object in such a way that the whole CSG tree is active in a large number of nodes. For example, consider the intersection of a halfspace with its complement. In fact, a pruned CSG tree may be active in a bintree block even though the CSG tree defines a NULL object in the region corresponding to the block. For example, consider the CSG tree given in Figure 8a consisting of the two circles L1 and R and the halfspaces A and B as shown in Figure 8b. The CSG tree of Figure 8a is active in the bintree block represented by the dashed square in Figure 8b even though the object defined by it does not extend so far.

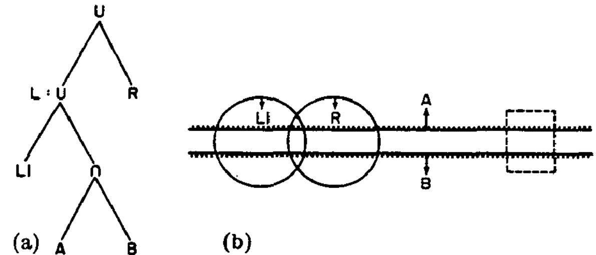


Figure 8. (a) A CSG tree and (b) its corresponding object.

First, let us examine the number of halfspaces that are active at each node of voxel size in a bintree of a polyhedron. This discussion is heuristic in that we speak of voxels as if they were infinitely small. At each vertex, at least three halfspaces are active. Elsewhere at each edge of the polyhedron exactly two halfspaces are active. Elsewhere at each face, only one halfspace is active. We can estimate the total number of active halfspaces by counting the number of voxels that intersect the edges, vertices, and faces of the polyhedron. We know that the total number of voxels intersecting faces is proportional to the surface area [9] while the total number of voxels that intersect edges is proportional to the sum of the edge lengths at the given resolution [4]. The number of voxels containing vertices is always bounded by the number of vertices irrespective of resolution. Assuming a resolution of M, the number of voxels with more than one active halfspace grows only linearly with M, while the total number of voxels grows with M^2. Thus the average number of halfspaces active in a node of voxel size approaches one asymptotically in a CSG tree that corresponds to a polyhedron. A similar result will hold for polyhedron-like objects of arbitrary dimension.

It should be clear that the amount of work necessary in performing the conversion is at least proportional to the number of nodes in the bintree. It has been shown [16] that there exists a class of CSG trees for which the complexity of evaluation is of the same order as the number of nodes in the bintree of the corresponding object. Such CSG trees are said to be "well-behaved" and this concept applies also to CSG trees with non-linear halfspaces. This characteristic is determined solely by the way the objects defined by the pair of brother subtrees of the CSG tree intersect each other. In a well-behaved CSG tree the intersections are not allowed to be "tangential". In two dimensions, this means that the boundaries of the objects corresponding to brother subtrees intersect at only a finite number of points. In three dimensions, for polyhedra, the boundaries should not coincide but are permitted to intersect along one-dimensional edges. In the general case, for d dimensions, the permitted intersection must

similarly be at most (d-2)-dimensional (see [16] for more details). Generalizing Hunter's and Meagher's image complexity results for polygons and polyhedra to d dimensions leads to a complexity of $O(M^{d-1})$ bintree nodes for bintrees of resolution M. This bound is attainable in a manner that does not depend on the number of nodes in the CSG tree. The following theorem, a variant of whose proof can be found in [16], summarizes the above discussion.

Theorem: Let T be a well-behaved CSG tree defining a non-degenerate d-dimensional object. The proportion of bintree nodes where more than one CSG tree node is active approaches zero asymptotically as the resolution increases.

The above results lead us to draw the following unexpected but practical conclusions about the performance of our algorithms for the conversion of CSG trees to bintrees when the CSG trees are well-behaved.

(1) The "practical" complexity of CSG tree evaluation is $O(M^{d-1})$ as resolution M is increased.
(2) The average number of active CSG tree nodes in a bintree block approaches one asymptotically as resolution is increased.
(3) The computational complexity of converting a CSG tree approximation of a given object to a bintree is asymptotically independent of the number of halfspaces used in the approximation.

Result (3) means that the linear approximation of curved halfspaces can be computationally practical even though it leads to a great increase in the size of the CSG tree. Of course, the above results are asymptotical and thus are directly relevant only when the number of halfspaces is not large in comparison to the resolution.

6. EMPIRICAL RESULTS

In order to test our predictions, we conducted a number of experiments with polyhedron-like objects in several dimensions. Our experiments have been performed with versions of the algorithms of Section 3, as implemented in C in the system [18] and executed on a VAX 11/750 running version 4.2BSD of UNIX. Note that the contribution to CPU time incurred by writing the packed DF-expression into a file is quite noticeable.

At each bintree node a fixed amount of work is performed for each CSG tree node that is active in it. Thus an implementation-independent measure of work is the total number of CSG tree nodes active at all of the bintree nodes. This is reported as the statistic "CSG evaluations" below. The statistic "Halfspace evaluations" forms part of it and denotes the number of halfspace value range computations performed. Note that the algorithm can be implemented in a manner that requires only one addition operation for each such evaluation [16]. From these values we can derive the average number of active CSG tree nodes (or halfspaces) in a bintree node for the purpose of comparison with the theoretical analysis. Note that the program that we instrumented used a pointer-less CSG tree representation, which allows less pruning than the algorithm we have described in Section 3. Thus the number of CSG node evaluations reported below is an upper bound on the true value obtained by the algorithm.

For our first experiment we approximated a circle with an 11-gon and formed its bintree at resolution 4096. This approximation produced a bintree with 80828 nodes and required 87592 CSG tree node and 81823 halfspace evaluations. Thus on the average each bintree node contained less than 1.1 active CSG tree nodes which correlates with our prediction. The CPU time required was 19.2 seconds, including about 6 seconds necessary to output the packed DF-expression. The time required to perform the same task by a program specifically designed to convert convex polyhedra was 17.1 CPU seconds so that the overhead of the general CSG tree representation was not very large.

The second experiment demonstrates that the complexity of CSG tree evaluation is $O(M^{d-1})$ as resolution M is increased. For this experiment we tabulate in Table 1 the CPU conversion times for a series of approximations of a unit circle by 5, 11, and 19 halfspaces at various resolutions. Notice that the execution time doubles with resolution as predicted for $d=2$. The different approximations are not completely comparable as they represent different objects. Thus the bintree at resolution 4096 contains 76294, 80628 and 81410 nodes for 5, 11 and 19 halfspaces respectively.

Table 1. Conversion times (CPU seconds) for different discs.

Number of Halfspaces	Resolution				
	256	512	1024	2048	4096
5	1.2	2.1	4.2	8.5	16.2
11	1.7	2.9	4.7	9.4	17.1
19	2.5	3.8	6.1	10.8	18.6

The third experiment shows that the size of the three-dimensional bintree of a polyhedron is proportional to the square of the resolution and a four-dimensional bintree is proportional to the third power of the resolution. The execution times for large values of resolution exhibited similar behavior. For this we modeled the motion of two identical square blocks situated at opposite corners of the unit square, moving towards each other, so that at time $t=1$ they overlap on an area of size 0.05 by 0.05. We also performed an identical three-dimensional experiment (two moving boxes) resulting in a four-dimensional bintree. In the first case (Figure 9) we have a total of 8 halfspaces and in the second case we have a total of 12 halfspaces. Tables 2 and 3 contain the result of the evaluation of the three-dimensional and four-dimensional trees at varying resolutions. Note that for these examples, the maximum sizes of the universes are 2^{33} and 2^{36} voxels respectively.

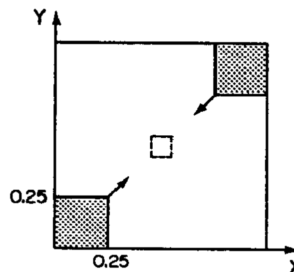


Figure 9. Intersection of two moving objects.

Table 2. Intersecting two moving 2-d blocks.				
Resolution	CPU seconds	BIN nodes	Halfspace evaluations	CSG evaluations
64	0.7	826	1641	3124
128	1.3	2898	4357	7300
256	3.2	10866	13552	19358
512	10.3	42514	47684	59254
1024	37.6	172802	183335	207248
2048	144.1	699362	720631	769768

Table 3. Intersecting two moving 3-d blocks.				
Resolution	CPU seconds	BIN nodes	Halfspace evaluations	CSG evaluations
16	0.8	370	1846	3900
32	1.2	898	3354	7008
64	3.1	4658	10471	21326
128'	12.5	31458	49969	90614
256	67.4	231570	295662	430730
512	428.0	1826466	2071158	2695692

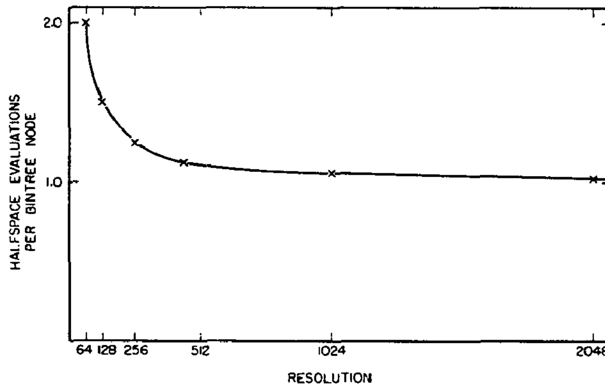


Figure 10. The number of halfspace evaluations per bintree node as a function of resolution for Table 2.

Figure 10 shows the number of halfspace evaluations per bintree node as a function of resolution in the experiment of Table 2. Notice that the asymptotical bound does hold in this case. Nevertheless, inspection of Tables 2 and 3 shows that the convergence to this bound is not necessarily very fast in the high dimensional (location,time)-space. Therefore this method should be used primarily when the CSG tree is expected to evaluate to NULL. Furthermore, it is much more efficient to determine just the first moment of intersection. In order to illustrate this point, we modified the coefficient of time in the experiment reported in Table 3 so that the true intersection was localized within eight three-dimensional voxels. The resulting four-dimensional evaluation at resolution 512 produced 530 bintree nodes (of which 8 were BLACK), evaluated 2765 halfspaces and 5884 CSG tree nodes, and required 1.2 CPU seconds.

The above experiments have only dealt with convex objects (i.e., intersections of halfspaces). To study the performance of our algorithms with a UNION operation, we performed one more simple experiment. We used hexagons to approximate five disks with radii 0.3 and centers at (0.0,0.0), (0.25, 0.25), (0.5,0.5), (0.75,0.75) and (1.0,1.0). The CSG tree that describes the union of these five circles within the unit square was of depth 6 and contained 30 leaves and 29 internal nodes (see Figure 11). Table 4 contains the results on a VAX 11/780.

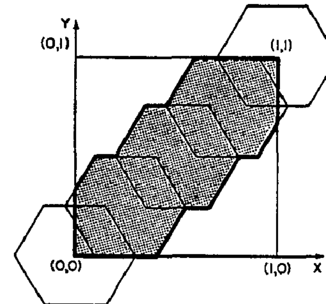


Figure 11. Union of five hexagons within the unit cube.

Table 4. Union of five disks approximated as hexagons.			
Resolution	CPU seconds	BIN nodes	Halfspace evaluations
1024	2.1	14418	17748
2048	3.9	28828	29489
4096	7.1	57654	58353

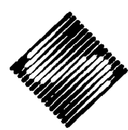
7. CONCLUDING REMARKS

The analysis of the execution time of CSG to bintree conversion was based on our definition of well-behaved which eliminated certain objects. If we expect such objects (beside the extensions reported below), the more complicated CSG tree redundancy checking algorithms of Tilove [19] should be used once the pruned tree has reached a certain (small) size. Note, however, that if we apply as CLASSIFY_VOXEL a CSG evaluation at the center of the voxel, no incorrect "false positive" bintree nodes result (i.e., nodes that should be completely WHITE but are classified as BLACK).

As presented, the algorithm of Section 3 does not handle the case that both a halfspace and its complement are leaves of the CSG tree. This case is actually quite common when a CSG tree is composed of a union of convex components (e.g., a triangulation). Nevertheless, the performance of our algorithms, as well as the analytical results of Section 5, remain valid by adding the following rule to the CSG evaluation in procedure PRUNE:

> If in a bintree node only a halfspace and its complement are active, then the node is BLACK if the root of the active CSG tree is UNION, and WHITE otherwise.

Our algorithms have several useful applications aside from volume-like computations and interference checking. Viewing three-dimensional CSG models is a prime application [7]. In this case bintree conversion would be performed solely for the sake of generating shaded output. The method of view-



ing three-dimensional bintrees in the direction of a coordinate axis described in [18] can be used in this case because the eyepoint dependent operations (e.g., perspective transformation of halfspaces, etc.) can precede bintree conversion. Shading would be generated from the normals of the halfspaces active at each visible node at voxel level. Evaluation would proceed from front to back and be combined with projection so that the nodes known to be covered would not be generated. The efficiency of our conversion algorithm is such that this might be a practical alternative as a system for viewing faceted three-dimensional CSG trees [2].

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under Grant DCR-8302118 and in part by the Finnish Academy. We thank Jarmo Alander, Olli Karonen, Petri Koistinen, Walter Kropatsch, Martti Mantyla, Reijo Sulonen, and Robert E. Webber for comments.

REFERENCES

[1] J. Alander, Interval arithmetic methods in the processing of curves and sculptured surfaces, *Proceedings of the Sixth International Symposium on CAD/CAM*, Zagreb, Yugoslavia (1984).

[2] P.R. Atherton, A scan-line hidden surface removal procedure for constructive solid geometry, *Computer Graphics 17*, 3(1983), 73-82.

[3] J.W. Boyse, Interference detection among solids and surfaces, *Communications of the ACM 22*, 1(January 1979), 3-9.

[4] G.M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

[5] C. Jackins and S.L. Tanimoto, Quad-trees, oct-trees, and k-trees - a generalized approach to recursive decomposition of Euclidean space, *IEEE Transactions on Pattern Analysis and Machine Intelligence 5*, 5(September 1983), 533-539.

[6] E. Kawaguchi and T. Endo, On a method of binary picture representation and its application to data compression, *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*, 1(January 1980), 27-35.

[7] P. Koistinen, M. Tamminen, and H. Samet, Viewing solid models by bintree conversion, to appear in *Proceedings of the EUROGRAPHICS '85 Conference*, Nice, September 1985.

[8] Y.T. Lee and A.A.G. Requicha, Algorithms for computing the volume and other integral properties of solids. I and II, *Communications of the ACM 25*, 9(September 1982), 635-650.

[9] D. Meagher, Octree encoding: a new technique for the representation, manipulation and display of arbitrary 3-D objects by computer, Report IPL-TR-80-111, Rensselaer Polytechnic Institute, Troy, New York, 1980.

[10] D. Meagher, The Solids engine: a processor for interactive solid modeling, *Proceedings of the NICOGRAPH '84 Conference*, Tokyo, November 1984.

[11] S.P. Mudur and P.A. Koparkar, Interval methods for processing geometric objects, *IEEE Computer Graphics and Applications 4*, 2(February 1984), 7-17.

[12] N. Okino, Y. Kakazu, and H. Kubo, TIPS-1: Technical information processing system for computer aided design, drawing and manufacturing, in *Computer Languages for Numerical Control*,, J. Hatvany, Ed., North Holland, Amsterdam, 1973, 141-150.

[13] A.A.G. Requicha, Representations of rigid solids: theory, methods, and systems, *ACM Computing Surveys 12*, 4(December 1980), 437-464.

[14] A.A.G. Requicha and H.B. Voelcker, Solid modeling: current status and research directions, *IEEE Computer Graphics and Applications 3*, 7(1983), 25-37.

[15] H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys 16*, 2(June 1984), 187-260.

[16] H. Samet and M. Tamminen, Approximating CSG trees of moving objects, Computer Science TR-1472, University of Maryland, College Park, MD, January 1985.

[17] M. Tamminen and H. Samet, Efficient octree conversion by connectivity labeling, *Computer Graphics 18*, 3(July 1984), pp. 43-51 (also *Proceedings of the SIGGRPAH '84 Conference*, Minneapolis, July 1984).

[18] M. Tamminen, P. Koistinen, J. Hamalainen, O. Karonen, P. Korhonen, R. Raunio, and P. Rekola, Bintree: a dimension independent image processing system, Report-HTKK-TKO-C9, Helsinki University of Technology, 1984.

[19] R.B. Tilove, A null-object detection algorithm for constructive solid geometry, *Communications of the ACM 27*, 7(July 1984), 684-694.

[20] A.F. Wallis and J.R. Woodwark, Creating large solid models for NC toolpath verification, *Proceedings of CAD 84*, 1984.

[21] J.R. Woodwark and K.M. Quinlan, Reducing the effect of complexity on volume model evaluation, *Computer-aided Design 14*, 2(1982), 89-95.

[22] M. Yau and S.N. Srihari, A hierarchical data structure for multidimensional digital images, *Communications of the ACM 26*, 7(July 1983), 504-515.