Binary Space Partition Tree and Constructive Solid Geometry Representations for Objects Bounded by Curved Surfaces

Suzanne F. Buchele Angela C. Roles

Department of Mathematics and Computer Science
Southwestern University

Abstract

Binary Space Partition (BSP) tree and Constructive Solid Geometry (CSG) tree representations are both set-theoretic binary tree representations of solid objects used in solid modeling and computer graphics. Recently, an extension of the traditional BSP tree definition has been presented, in which surfaces used in the binary partition include curved surfaces in addition to planar surfaces. We examine the relationship between this extended definition of BSP trees and halfspace CSG trees, including conversion between the two representations, in light of this new BSP tree definition for solid objects bounded by curved surfaces.

1 Introduction

Binary Space Partition (BSP) tree and Constructive Solid Geometry (CSG) tree representations are widely used to represent solid objects. Until recently, BSP tree representations were restricted to objects bounded by linear surfaces (lines in 2-D, planes in 3-D). Thus BSP trees were applicable only to polyhedral objects in 3-D. The conversion to and from BSP tree representations and CSG tree representations of 2-D and 3-D solid objects has not been succinctly studied in the past, largely because BSP trees represented a far more limited class of objects.

The conversion of a boundary representation (b-rep) of a solid object to a BSP tree representation has been presented for both polyhedral [6] objects and objects bounded by quadric surfaces [1]. Boundary representation to CSG conversion has also been presented [1, 5].

We present a discussion of the relationship between BSP tree and CSG tree representations of solid objects in 2-D and 3-D. While the theory of BSP and CSG tree representations are applicable to d-dimensional Euclidean space, we focus on 2-D and 3-D applications for practical purposes. The current b-rep to BSP and CSG tree implementations [1, 5] function in 3-D.

2 Background Material

2.1 Regular Sets and Halfspaces

The use of regular sets and regularized set operations prevents the occurrence of "dangling" edges or surfaces (subsets with empty interior) in set-theoretic representations of solids. A set X is said to be regular if X is equal to the closure of the interior of X; that is, $X = cl(int(X))$. Thus, the regularization of a set X is the closure of the interior of X. Regularized set operations operate on regular sets, where the defined binary set operations are regularized union ($\cup^*$), regularized intersection ($\cap^*$), and regularized set difference ($-^*$). The unary set operation regularized complement ($^{-*}$) is also defined. For two regular sets X and Y and a binary set operation $< op >$, $X < op >^* Y = cl(int(X < op > Y))$. The regularized complement of X, $\overline{X}^*$, is the closure of the interior of the complement of X [4].

In 3-D, a halfspace is a set of the form $h = \{(x, y, z) : g(x, y, z) \geq 0\}$ for some function g that maps $\mathbb{R}^3$ into $\mathbb{R}$. We refine the types of halfspaces that we use for both BSP trees and halfspace CSG trees to include only halfspaces h that are non-empty regular sets, and such that the regularized complement halfspace $\overline{h}^*$ is also a non-empty set (and is regular by definition). We refer to zeroes of the function g defining the halfspace as the surface of the halfspace, $S_h = \{(x, y, z) : g(x, y, z) = 0\}$, and we say the surface S_h induces the two halfspaces $h = \{(x, y, z) : g(x, y, z) \geq 0\}$ and $\overline{h}^* = \{(x, y, z) : g(x, y, z) \leq 0\}$. Buchele [1] points out that non-degenerate quadric surfaces of a single sheet induce two non-empty, regular halfspaces, as do many other types of polynomial and non-polynomial functions.

2.2 BSP Trees

A binary space partition tree, or BSP tree, is a binary tree that recursively partitions d-dimensional Euclidean space. The primary application of BSP trees is to represent a subset of $\mathbb{R}^d$. In the traditional definition of BSP trees, the space is split into two parts by a hyperplane, and each of the subspaces created is recursively split until the entire space is split into homogeneous regions which are either completely inside or outside the total region to be represented. In our development, we assume that the left child of an internal node represents the "positive" side of the partitioning surface, e.g. $\{(x, y, z) : g(x, y, z) \geq 0\}$. The right child of an internal node represents the "negative" side of the partitioning surface, $\{(x, y, z) : g(x, y, z) \leq 0\}$. Homogeneous regions (leaves) in a BSP tree are denoted by "in" or "out" cells, depending on whether the region represented by that leaf is completely inside or outside of the total region to be represented.

In the traditional definition of BSP trees, internal nodes are hyperplanes: lines in 2-D and planes in 3-D. Recently, Buchele [1] proposed an extended definition of BSP trees, in which curved surfaces are allowed as partitioning surfaces. Equations of the partitioning surfaces are stored in internal nodes. For a curved surface in 3-D defined by $\{(x, y, z) : g(x, y, z) = 0\}$, the left subtree represents the halfspace $h = \{(x, y, z) : g(x, y, z) \geq 0\}$, and the right subtree represents the halfspace $\overline{h}^* = \{(x, y, z) : g(x, y, z) \leq 0\}$. We use this extended definition when we refer to BSP trees in this paper. Note that the extended definition of BSP trees includes the traditional definition. An example of a two-dimensional solid and a BSP tree representation of it is given in Figures 2a and 2b.

For a given object in $\mathbb{R}^3$, we define the faces of the object to be maximally connected components of surfaces bounding the object. Surfaces containing faces bounding the object are called natural surfaces of the object [1]. A natural halfspace of the object is a halfspace associated with a natural surface of the object [5]. An autopartition of the object is defined to be a binary partition of $\mathbb{R}^3$, separating the object from the rest of $\mathbb{R}^3$ such that only natural surfaces of the object are used in the partition [3].

2.3 Halfspace CSG Tree Representations

A CSG tree is a binary tree that has regularized set operations as internal nodes and regular sets as leaves. CSG trees are also used to represent subsets of $\mathbb{R}^d$; the represented region is the result of applying the regularized set operations on the regular sets. An advantage of the use of CSG trees to represent solid objects is the fact that regularized set operations applied to regular sets always produces a regular set.

Halfspace CSG trees perform regularized set operations on regular sets that are halfspaces. We restrict our attention to halfspace CSG trees, whose leaves are non-empty regular halfspaces in $\mathbb{R}^3$, and such that the

```
BSP_to_CSG (BSP tree B)

Input: BSP tree B
Output: CSG tree C
Methods used:
   Compl(): returns the complement of the halfspace parameter

If B.right is "out"
   If B.left is "in"
      C.root : = B.root
   Else
      C.root, C.left, C.right : = ∩*, BSP_to_CSG(B.left), B.root
Else If B.left is "out"
   If B.right is "in"
      C.root : = Compl(B.root)
   Else
      C.root, C.left, C.right : = ∩*, BSP_to_CSG(B.right), Compl(B.root)
Else If B.left is "in"
   C.root, C.left, C.right : = ∪*, B.root, BSP_to_CSG(B.right)
Else If B.right is "in"
   C.root, C.left, C.right : = ∪*, Compl(B.root), BSP_to_CSG(B.left)
Else /* B.root has both a left subtree and a right subtree */
   C.root, C.left, C.right : = ∪*, ∩*, ∩*
   C.left.left : = BSP_to_CSG(B.left)
   C.left.right : = B.root
   C.right.left : = Compl(B.root)
   C.right.right : = BSP_to_CSG(B.right)
Return C
```

Figure 1: Algorithm BSP_to_CSG

corresponding regularized complement halfspace exists and is non-empty.

3 BSP to CSG Conversion

A simplistic and verbose approach to the BSP to halfspace CSG conversion problem is to note that an object represented by a BSP tree can also be represented by the union of the "in" cells in the tree. An "in" cell in a BSP tree can be represented as the intersection of all halfspaces on the path from the root node of the BSP tree to the "in" cell. Buchele [1] presented and proved the correctness of a more efficient algorithm, called BBHC, to take a BSP tree representation of an object and convert it to a halfspace CSG tree representation of the object. Algorithm BSP_to_CSG (Figure 1) is an algorithmic restatement the BBHC algorithm, and takes as input a BSP tree representation of a 3-D object and recursively processes each node in order to form a CSG representation of the object. We assume that the input BSP tree is in a simplified form, in that no internal node has two "in" cells or two "out" cells as child nodes.

Algorithm BSP_to_CSG begins with the root node of the BSP tree, and processes each internal node in the tree through recursive calls to the BSP_to_CSG algorithm. Let halfspace h (initially the root node) be the internal node in the BSP tree under consideration in the BSP_to_CSG algorithm. If h has an "in" cell and an internal node as children, the corresponding CSG representation will consist of the union of h or $\overline{h}^*$ with the CSG representation of the subtree rooted at the internal node (formed by a recursive call to the BSP_to_CSG algorithm). If the "in" cell is the left child of h, then the corresponding CSG representation of the BSP tree rooted at h will be the union of h and the

CSG-conversion of the right subtree of h. If the "in" cell is the right child of h, then the CSG representation of the BSP tree rooted at h will consist of the union of $\overline{h}^*$ and the CSG-conversion of the left subtree of h. The CSG representation of the child subtree is formed by a recursive call to the BSP_to_CSG algorithm.

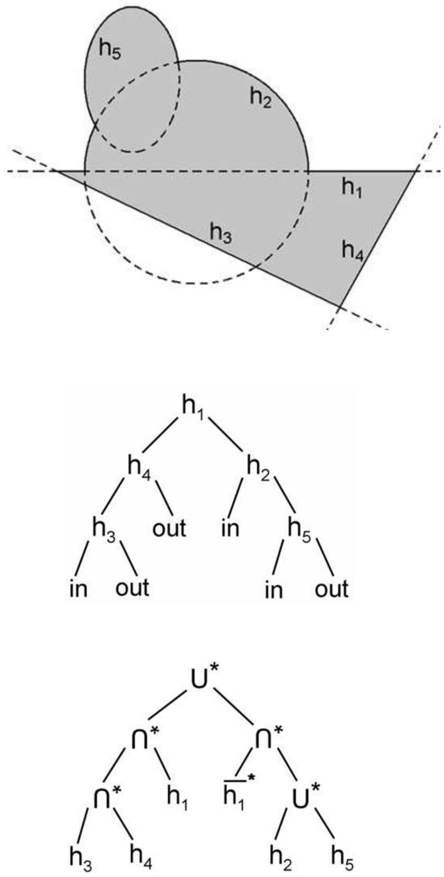
If h has an "out" cell and an internal node as children, the CSG representation will be the intersection of h or $\overline{h}^*$ with the CSG representation of the subtree rooted at the internal node. If the "out" cell is the right child of h, then the corresponding CSG representation of the BSP tree rooted at h will be the intersection of h and the CSG-conversion of the left subtree of h. If the "out" cell is the left child of h, then the CSG representation of the BSP tree rooted at h will consist of the intersection of $\overline{h}^*$ and the CSG-conversion of the right subtree of h.

If h has as children an "in" cell and an "out" cell, then the CSG representation will simply be h or $\overline{h}^*$. If the "in" cell is the left child of h, h is used in the CSG tree. Similarly, if the "in" cell is the right child of h, $\overline{h}^*$ will be used in the CSG tree.

Otherwise, h has internal nodes as both of its children. The CSG representation of the left subtree of h is intersected with halfspace h in order to obtain the region defined on the positive side of h. Similarly, the CSG representation of the right subtree of h is intersected with halfspace $\overline{h}^*$ in order to obtain the region defined on $\overline{h}^*$. The CSG representation of each subtree is obtained by recursive calls to the BSP_to_CSG algorithm. The CSG representation of the BSP tree rooted at h then consists of the union of these two intersections.

For example, Figure 2a denotes an object in two dimensions. Figure 2b shows an extended BSP tree representation for this object, and Figure 2c shows the corresponding CSG tree representation generated by the BSP_to_CSG algorithm. At the root of the input BSP tree, h_1 has children which are both internal nodes. Therefore, the CSG tree in 2c has a root which is a union node, with both children intersection nodes. The left intersection node, as the algorithm specifies, has as children the CSG conversion of h_1's left subtree (that which is rooted by h_4 in the BSP tree), and the positive halfspace h_1. The right intersection node has as children CSG conversion of h_1's right subtree (that which is rooted by h_2 in the BSP tree) and the negative halfspace $\overline{h}_1^*$.

In the subtree rooted by h_4 in the BSP tree, the right child of h_4 is an "out" cell, and the left child is an internal node (the subtree rooted at h_3). Therefore, the root of the CSG representation for this region is an intersection node, the left child is the CSG conversion of the subtree rooted at h_3, and the right child is h_4. For the subtree rooted at h_3, h_3 has both an "in" cell and an "out" cell, so h_3 is used in the corresponding CSG tree of figure 2c.



a)
b)
c)

Figure 2: (a) Two-dimensional object (shaded region). (b) A BSP tree representation of the object. (c) A halfspace CSG tree representation of the object, obtained using the BSP_to_CSG algorithm.

In the subtree rooted at h_2 (the left subtree of the BSP root node), the left child of h_2 is an "in" cell, and the right child is an internal node (the subtree rooted at h_5). Therefore, the root of the CSG representation for this region is a union node, the union node's left child is h_2, and the right child is the CSG conversion of the subtree rooted at h_5, which is just h_5.

Due to the straightforward nature of the BSP_to_CSG algorithm, and the proof of correctness by Buchele [1], we arrive at the following Lemma:

Lemma: An arbitrary BSP tree of size n internal nodes can be converted to a CSG tree representation of $O(n)$ size in $O(n)$ time.

Proof: We assume that the input BSP tree is simplified in that no internal node can be replaced with an "in" or "out" cell; if so, an $O(n)$ pre-processing step may be applied to perform any necessary replacement. Let B be a BSP tree of size n internal nodes, and let C be the CSG tree resulting from the BSP_to_CSG algorithm. Each of the n internal nodes of B translates to at most 3 internal nodes in C. However, at least one internal node in B must have both an "in" cell and an "out" cell (two leaves) as children, and thus translates to a single leaf node in C. So, $|C| < 3n$ where n is the size of B. Thus the size of C is $O(n)$.

Clearly, conversion is completed in $O(n)$ time since each internal node of B is visited exactly once. □

It is not always the case that a BSP tree which uses only natural surfaces in an object (surfaces in the autopartition of the object) will exist. Due to the presence of curved surfaces, partitioning surfaces may be necessary to represent the object [1, 5]. However, we have shown that if a BSP tree of size n internal nodes of the object does exist, using either natural surfaces of the object or natural surfaces and additional partitioning surfaces, then the object can be represented by an $O(n)$ sized halfspace CSG tree as well.

```
CSG_to_BSP (CSG tree C)

Input: CSG tree C
Output: BSP tree B
Methods used:
  CSG_Union_Isect(): converts CSG tree to contain only ∪* and ∩*
        operations
  Merge_Bspts(): Naylor's Merge_Bspts(), with added parameter
        to specify ∪* or ∩*

C : = CSG_Union_Isect(C)
If C.root a halfspace
  B.root, B.left, B.right : = C.root, "in", "out"
Else
  If C.root an operation AND (C.left OR C.right a halfspace)
    If C.root is ∩*
      If C.right a halfspace
        B.root : = C.right
        B.left, B.right : = CSG_to_BSP (C.left), "out"
      Else If C.left a halfspace
        B.root : = C.left
        B.left, B.right : = CSG_to_BSP (C.right), "out"
    Else If C.root is ∪*
      If C.right a halfspace
        B.root : = C.right
        B.left, B.right : = "in", CSG_to_BSP (C.left)
      Else If C.left a halfspace
        B.root : = C.left
        B.left, B.right : = "in", CSG_to_BSP (C.right)
  Else
    B.left, B.right : = CSG_to_BSP (C.left), CSG_to_BSP (C.right)
    If C.root is ∩*
      B : = Merge_Bspts (B.left, B.right, ∩*)
    Else If C.root is ∪*
      B : = Merge_Bspts (B.left, B.right, ∪* )
Return B
```

Figure 3: Algorithm CSG_to_BSP

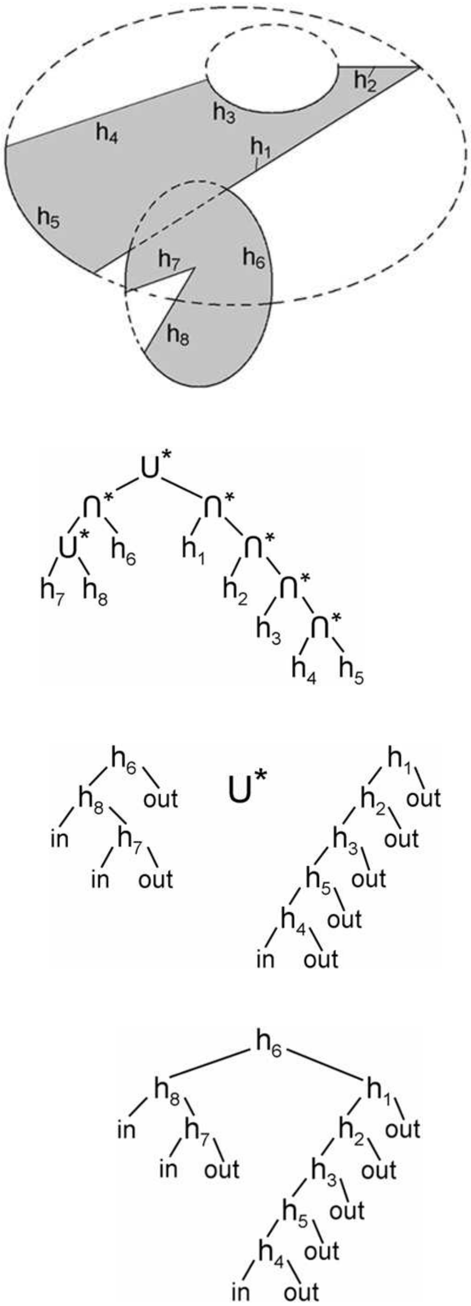
4 CSG to BSP Conversion

Thibault and Naylor [6] presented work on the use of (traditional) BSP trees as a method for representing polyhedra, and the use of BSP tree algorithms to implement set operations on polyhedra. As part of this work, they presented an algorithm to convert a CSG tree to a (traditional) BSP tree. Later, Naylor et. al. [2] presented an algorithm called Merge_Bspts which takes as input two BSP trees that partition the same subspace and merges them together. This is accomplished by systematically partitioning one tree by the binary partitioner at the root of the other, and recursively overlaying the partitionings from the two trees.

We propose a new algorithm, CSG_to_BSP, to convert a CSG tree representation of an object to a BSP tree representation (Figure 3). The Merge_Bspts algorithm [2] is used to implement the union or intersection of two subtrees that have already been converted to BSP trees. The fundamental operation of the Merge_Bspts algorithm, Partition_Bspt [2], is used to partition a subtree by a halfspace, and it is at this point that either union, intersection, or set difference can be carried out to implement the operation between the two trees. Code to implement this fundamental operation of partitioning an existing BSP tree by a single halfspace, for a halfspace defined by a quadric surface of a single sheet and a BSP tree whose node halfspaces are defined by quadric surfaces of a single sheet, is implemented in the BB method presented by Buchele [1]. Thus, the CSG_to_BSP algorithm is implementable for CSG trees representing objects bounded by quadric surfaces of a single sheet.

The CSG_to_BSP algorithm requires that the CSG tree representation of the region be in terms of regularized unions and regularized intersections only. This can be accomplished easily, since for regular sets X and Y, $X -^* Y = X \cap^* \overline{Y}^*$. Therefore, expressing a CSG tree in terms of regularized unions and intersections only requires that any $-^*$ (regularized set subtractions) in the tree be converted to $\cap^*$ (regularized set intersection), with the right child of the regularized intersection node complemented. DeMorgan's Laws can be used to complement subtrees properly. In the CSG_to_BSP algorithm, the method CSG_Union_Isect carries out this preliminary conversion.

In the CSG to BSP conversion itself, the root node of the tree is first checked. If this node is already a halfspace (a leaf), the BSP tree returned contains that node as the root, with an "in" cell and an "out" cell as its children. Otherwise, the tree is more than just a leaf and there is at least one set operation that must be handled. If the root operation is $\cup^*$ and at least one of the children is a halfspace, then that child which is a halfspace (or the right halfspace, if both children are halfspaces) becomes the root of the output BSP tree. The right child of this halfspace in the BSP tree then becomes an "out" cell, and the left child will be the CSG to BSP conversion of the intersection node's other subtree, through a recursive call to the algorithm. Similarly, if the root operation is $\cap^*$ and at least one of the children is a halfspace, then that child which is a halfspace (or, again, the right halfspace, if both children are halfspaces) becomes the root of the out-



a)

b)

c)

d)

Figure 4: (a) Two-dimensional object (shaded region). (b) A halfspace CSG tree representation of the object, consisting only of unions and intersections. (c) An intermediate step in the CSG_to_BSP algorithm (d) BSP tree representation of the object, obtained using algorithm CSG_to_BSP

put BSP tree. The left child in the BSP tree, in this case, then becomes an "in" cell. The right child will be the CSG to BSP conversion of the union node's other subtree, once again through a recursive call. If none of the above cases are true, then the root node of the CSG tree has as children two set operation nodes. In this case, both children are therefore roots of subtrees. Both of these subtrees are converted to BSP trees using recursive calls to the CSG_to_BSP algorithm. These two BSP trees are then merged using a version of the Merge_Bspts algorithm presented by Naylor et. al. [2]. The only difference in form is an added parameter to specify whether a union or an intersection should be carried out at the operation between the two input trees.

For example, figure 4a denotes a two-dimensional solid which is represented in figure 4b by a CSG tree consisting only of unions and intersections. Figure 4c shows an intermediate step, and figure 4d shows the BSP tree representation of the figure which results from the CSG_to_BSP algorithm. The root of the CSG tree is a union, with intersections as both children. Therefore, the left and right subtrees of the root union node will be converted to BSP trees separately and then merged. At the lowest levels, the subtrees will be merged accordingly with "in" or "out" cells. In the left subtree, rooted by an intersection, the intersection's right child is a halfspace, h_6. Therefore, h_6 becomes the root of the resulting BSP tree, the left child of h_6 will be the BSP conversion of the intersection's other child, and the right child of h_6 will be an "out" cell. The intersection's left child is the union of two halfspaces, so h_8 becomes the root of the left child of h_6 in the BSP tree. The left child of h_8 becomes an "in" cell (since the root of the corresponding CSG tree was a union node), and the right child of h_8 is h_7, which has an "in" cell and an "out" cell as children. In figure 4c, the BSP conversion of the left subtree of the root of the CSG tree is shown on the left.

In the right subtree of the root union node, the root is again an intersection. The intersection's left child is a halfspace, and the right child is a subtree. Therefore, the root of the BSP conversion of this subtree is a halfspace, h_1. The left child of h_1 is the BSP conversion of the right subtree of the intersection, and the right child of h_1 is an "out" cell. This manner of construction continues similarly, until, at the bottom of the CSG tree, we have an intersection node with right child h_5 and left child h_4. So, h_5 becomes the root of the corresponding BSP subtree, the right child of h_5 will be an "out" cell, and the left child of h_5 is the BSP conversion of CSG node h_4, which is the BSP node h_4 with both an "in" cell and an "out" cell as children. In figure 4c, the BSP conversion of the right subtree of the root of the CSG tree is shown on the right. Now, since both the left and right subtrees of the root of the CSG tree have been converted to BSP trees, merging can be carried out, and results of applying the Merge_Bspts algorithm is shown in figure 4d.

Given the algorithm above, we can conclude that if there exists a CSG tree representation of an object (bounded by second degree polynomials), then there will always exist an extended BSP tree representation of the object.

5 Concluding Remarks

Algorithm BSP_to_CSG will convert an arbitrary BSP tree of n internal nodes into an $O(n)$ halfspace CSG tree representation in $O(n)$ time. The CSG_to_BSP algorithm will convert an arbitrary halfspace CSG tree into a BSP tree. Since we have conversion both ways, our results show that BSP trees and CSG trees can both represent the same objects.

Future work is possible for determining the complexity of the CSG_to_BSP algorithm. Thibault and Naylor do not present a complexity argument for their CSG to (traditional) BSP conversion algorithm [6], although it appears to be $O(2^n)$. It can be shown that the CSG_to_BSP algorithm presented here produces a BSP tree in $O(n^4)$ time for a CSG tree of size n nodes in worst case. This is because the Merge_Bspts algorithm [2] is $O(n^3)$ for non-planar surfaces in the worst case, and will have to be done at most once at each of the $O(n)$ set operation nodes of the CSG tree. If we assume that each face of the represented object is bounded by a constant number of adjoining faces, then an overall CSG_to_BSP worst case complexity of $O(n^3)$ can be shown. As with many complexity results in BSP tree theory, it may be the case that the CSG_to_BSP algorithm operates in $O(n)$ or $O(n^2)$ time and produces an $O(n)$ sized BSP tree in practice, even though the worst case size and time complexity is much larger.

References

[1] S. F. Buchele. *Three-Dimensional Binary Space Partitioning Tree and Constructive Solid Geometry Tree Construction from Algebraic Boundary Representations*, Ph.D. Dissertation, The University of Texas at Austin, 1999.

[2] B. Naylor, J. Amatides, and W. Thibault. *Merging BSP Trees Yields Polyhedral Set Operations*, Computer Graphics, Vol. 24, No. 4, 1990, pp. 115-124.

[3] M. Paterson and F. Yao. *Efficient Binary Space Partitions for Hidden-Surface Removal and Solid Modeling*, Discrete and Computational Geometry, Vol. 5, 1990, pp. 485-503.

[4] A. A. G. Requicha, R. B. Tilove. *Mathematical Foundations of Constructive Solid Geometry: General Topology of Closed Regular Sets*, Technical Memo 27, Production Automation Project, University of Rochester, Rochester, N.Y., 1978.

[5] V. Shapiro, D. L. Vossler. *Construction and Optimization of CSG Representations*, Computer Aided Design, Vol. 23, No. 1, 1991, pp. 4-20.

[6] W. Thibault and B. Naylor. *Set Operations on Polyhedra Using Binary Space Partitioning Trees*, Computer Graphics, Vol. 21, No. 4, 1987, pp. 153-162.