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 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. 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 implicitly 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. 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].

2.2 Halfspace CSG Tree Representations

A CSG tree is a binary tree that has regularized set operations as internal nodes (regularized union ($\cup^*$), regularized intersection ($\cap^*$), and regularized set dif-

```
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
```

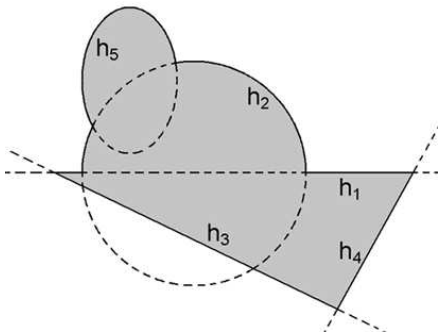
Figure 1: Algorithm BSP_to_CSG

ference ($-^*$)) and regular sets as leaves (A set X is said to be regular if X is equal to the closure of the interior of X; that is, $X = cl(int(X))$) [4]. 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.

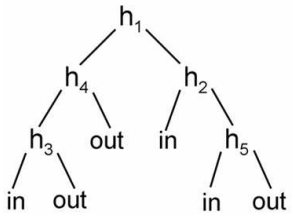
Halfspace CSG trees perform regularized set operations on regular sets that are halfspaces. We restrict our attention to halfspace CSG trees, whose leaves are non-empty regular halfspaces in $\mathbb{R}^3$, and such that the corresponding regularized complement halfspace exists and is non-empty.

3 BSP to CSG Conversion

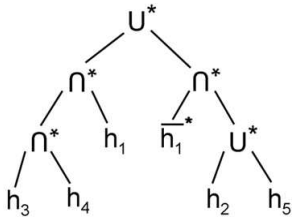
A simplistic and verbose approach to the BSP to halfspace CSG conversion problem is to note that an object represented by a BSP tree can also be represented by the union of the "in" cells in the tree. An "in" cell in a BSP tree can be represented as the intersection of all halfspaces on the path from 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 of 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.



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.

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.

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.

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 (traditional) 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 subtraction) 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.

```
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

For example, figure 4a denotes a 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.

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

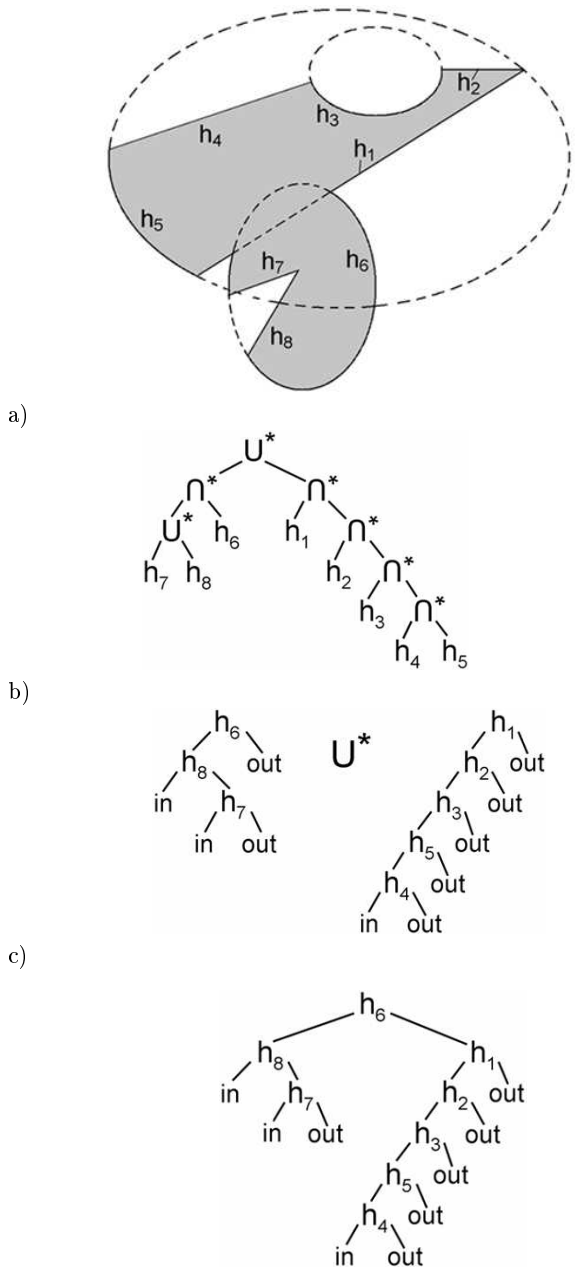


Figure 4: (a) Two-dimensional object (shaded region). (b) A halfspace CSG tree representation of the object, consisting only of unions and intersections. (c) An intermediate step in the CSG_to_BSP algorithm (d) BSP tree representation of the object, obtained using algorithm CSG_to_BSP

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.