1990

BRep-Index: A Multi-Dimensional Space Partitioning Tree

George Vaněček

Report Number:
90-1051

BRep-INDEX: A MULTI-DIMENSIONAL SPACE PARTITIONING TREE

George Vanecek, Jr.

Computer Sciences Department
Purdue University

Technical Report CSD-TR-1051
CAPO Report CER-90-47
December, 1990

BRep-Index: A Multi-dimensional Space Partitioning Tree

George Vaněček, Jr.
Department of Computer Science
Purdue University
West Lafayette, IN 47907
E-Mail: vanecek@cs.purdue.edu

Abstract

In this paper we introduce the brep-index, a new multi-dimensional space partitioning data structure that provides quick access to the vertices, edges and faces of a boundary representation (brep), thus yielding a single unified representation for polyhedral solids. We describe the construction of the brep-index and show that its size is $\Omega(v + e + f)$, where v, e, and f are the number of vertices, edges, and faces of the brep. The lower bound can be achieved for some breps by compressing the structure using simple rewrite rules. We apply the brep-index to solve the point/solid and the line/solid classification problems, and outline an efficient collision detection algorithm as an example of its use.

1 Introduction

Most data structures that exist for representing polyhedral solids can be categorized either as boundary-based or as volume-based. Each category has certain benefits not found in the other and therefore, many geometric modeling systems use both. Typically, the boundary representation (brep) is used as the primary representation with a separate volume-based directory aiding the retrieval of vertices, edges and faces from the brep. A spatial access directory is needed since the vertices, edges and faces of the brep are commonly kept as lists, imposing a sequential search of the lists with complicated tests for volume-based operations such as the point/solid and the line/solid classifications.

While much of the emphasis is placed on correctly and efficiently creating breps, an efficient spatial analysis of a correctly created brep is equally important for applications that repeatadly reason about solids. Collision detection and analysis of moving objects is one such application. It was with such an application in mind, that a volume-based representation that can be integrated with a boundary representation is introduced to provide a single unified representation that permits efficient spatial queries and analyses. We call the new representation a *brep-index.*

The brep-index is an extension of the binary space partition (BSP) tree [13]. A d-dimensional BSP tree hierarchically decomposes space into convex regions. The root node of a BSP tree represents the entire space while the leaves represent regions that are either completely inside or completely outside the solid. As a unique representation for solids, aside from other advantages, the BSP tree has the advantage that it is easy to perform spatial queries, and that it is dimension-independent.

However, it nevertheless partitions only the given d-dimensional space and does not give an access to the lower dimensional boundary elements of the solid. The brep-index extends the BSP tree into a multi-dimensional structure in that the hyperplanes separating the open regions are recursively decomposed as lower-dimensional BSP trees. The brep-index is a ternary tree that partitions the space containing a solid into three, two, one and zero-dimensional regions. In a recent survey, Naylor proposes a similar idea but does not work out important details [14].

The multi-dimensional partitioning has several benefits. First, it allows us to label each region with attributes, in our case the topological elements of the brep. In Section 2 we specify the construction of the brep-index and the labeling of the regions. Second, unlike the BSP tree, the brep-index allows an algebraic approach to compressing and rebalancing the tree, as described in Section 3. Third, the point/solid and the line/solid classification problems can be solved both robustly and efficiently, and with the additional advantage of knowing exactly which topological elements of the brep the point and line contains, as described in Section 4. Section 5 presents the performance data of the brep-index in the collision detection and analysis problem.

2 Construction of the BRep-Index

The brep-index can be constructed for all major breps proposed in the literature, including Baumgart's winged-edge data structure [2], Braid's modified winged-edge data structure [3], Yamaguchi and Tokieda's bridge-edge data structure [22], Vaněček's fedge-based data structures [18], Karasick's star-edge data structure [9], and Mäntylä's half-edge data structure [12]. The construction of the brep-index requires simple Euler operators [12] to perform local topological changes, and also requires the operation to split a face by a cutting plane robustly.

An oriented plane, the halfspace above it, and the halfspace below it are denoted by P, $P^>$, and $P^<$, respectively. A *support region* is a convex set of zero, one, two or three dimensions defined as the intersection of planes and halfspaces. With the superscript denoting the dimension of the set, support regions are defined in general as follows

$$\begin{aligned} r^0 &= P_1 \cap P_2 \cap P_3, \\ r^1 &= P_1 \cap P_2 \cap P_3^< \cap P_4^<, \\ r^2 &= P_0 \cap \bigcap_{i=1}^{n} P_i^<, \end{aligned}$$

$$r^3 = \bigcap_{i=1}^{n} P_i^{<}.$$

Note that for $d > 0$, support regions are (relatively) open sets. We will decompose the solid and its boundary elements into suitable support regions. Each support region is a leaf of the (uncompressed) brep-index tree and, in the case of a dimension less than three, it points to a corresponding boundary element.

Often, support regions are defined redundantly with more halfspaces than necessary. The redundancy allows us to organize the planes and halfspaces of all the support regions into a ternary tree hierarchically. Each internal tree node n represents an open convex region $R(n)$, and contains a plane, $P(n)$, that intersects $R(n)$. The three children of the node represent the subregions of $R(n)$ that lie above, on, and below $P(n)$, and are referred to as $\text{ABOVE}(n)$, $\text{ON}(n)$, and $\text{BELOW}(n)$, respectively. Thus,

$$\begin{aligned} R(\text{ABOVE}(n)) &= R(n) \cap P(n)^{>}; \\ R(\text{ON}(n)) &= R(n) \cap P(n); \\ R(\text{BELOW}(n)) &= R(n) \cap P(n)^{<}. \end{aligned}$$

Here, *above* is in the direction of the normal vector of $P(n)$. If the dimension, $\dim(n)$, of $R(n)$ is d, then the regions $R(\text{ABOVE}(n))$ and $R(\text{BELOW}(n))$ are also d-dimensional, but the region $R(\text{ON}(n))$ is $(d-1)$-dimensional.

Consider the example of Figure 1. A nonconvex solid is partitioned and its top face along with the corresponding subtree of the brep-index is shown. The subtree has seventeen internal nodes. In the figure, four similar subtrees are overlayed to improve readability, as indicated by the parallel edges.

2.1 Main Algorithm

The algorithm for building the brep-index consists of the following steps:

Algorithm CBI. Construct the brep-index T for a given brep B.

1. Fragment B obtaining B', create the brep-index T' (see Section 2.2).
2. Attach T' to B' (see Section 2.3).
3. Reduce B' reconstructing the brep B (see Section 2.4), and simultaneously update T' to yield the brep-index T.
4. Optionally compress T (see Section 3).
5. Return T.

2.2 Building the Tree

Step 1 of Algorithm CBI builds the ternary tree in three phases. In the first phase, all faces are processed and the support planes of the faces in B are used to recursively partition the solid and the surrounding space. This phase is similar to the BSP tree construction but does less internal bookkeeping in anticipation of adding the middle

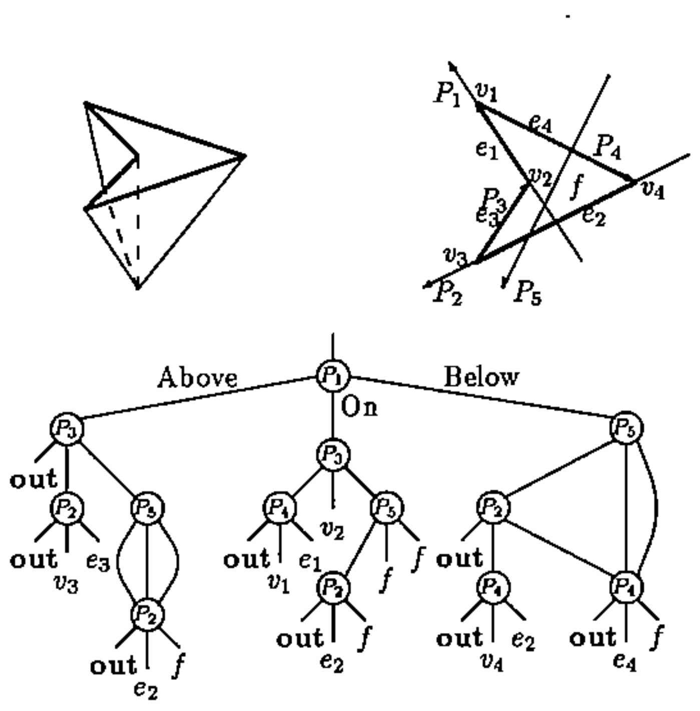

Figure 1: The partitioning of the top face of a solid, shown top left, is shown on the top right. The tree has seventeen nodes; some are overlayed for readability.

descendants at each node. At the end of this phase, the planar regions between the constructed 3D support regions are not yet subdivided. The algorithm is as follows; note that it creates a ternary tree with three branches labeled **out**, **on**, and **in**, respectively, and that the middle branches are just place holders at this time:

Algorithm PF. Given the set F of faces of B, process F recursively as follows:

1. Select a face f from F.
2. Remove from F the face f and all other faces coplanar with f. Fragment all of the removed faces that are not convex into convex faces.
3. Partition the reduced set F into sets F_{below} and F_{above} by the oriented support plane of f.
4. Create node n, let $P(n)$ be the support plane of f, and $\text{ON}(n)$ the value **on**.
5. When F_{above} is empty, assign $\text{ABOVE}(n)$ the value **out**, otherwise assign it the subtree created recursively with F_{above}.
6. When F_{below} is empty, assign $\text{BELOW}(n)$ the value **in** , otherwise assign it the subtree created recursively with F_{below}.

In the second phase the edges of the brep are processed. The corresponding Algorithm PE is the 2D version of Algorithm PF. It constructs the corresponding tree subdividing each hyperplane region that descends from some middle branch of the tree constructed by Algorithm PF. Again, the middle branches in the trees constructed by Algorithm PE are place holders for lines that are not yet subdivided into their proper supporting regions. At the end of this phase, all 3D and 2D supporting regions are known.

Algorithm PE. Given the set E of edges of B and the root node n of the tree created by Algorithm PF, process E recursively as follows:

1. Split the edges of E so that no edge crosses the plane $P(n)$ and partition the edges into the set E_{above}, E_{on}, and E_{below}.
2. Process the edge sets E_{above} and E_{below} recursively in the subtrees ABOVE(n) and BELOW(n), respectively.
3. When E_{on} is nonempty, create a new subtree ON(n) as follows:
   (a) Select an edge e from E and remove from E the edge e and all other edges collinear with e.
   (b) Create node m and associate with it the support plane of an adjacent face of e that is not $P(n)$. However, if the two adjacent faces of e are coplanar, compute instead the plane that contains e and that is perpendicular to $P(n)$.
   (c) Split all edges crossing $P(m)$ into two edges and partition the set into the sets E_{above}, E_{on} and E_{below}, and ignore the set E_{on}.
   (d) When E_{above} is empty assign ABOVE(m) the value **out**, otherwise assign it the subtree created recursively with E_{above}.
   (e) When E_{below} is empty assign BELOW(m) the value **in**, otherwise assign it the subtree created recursively with E_{below}.

In the third phase the vertices are processed by Algorithm PV. Algorithm PV is the one-dimensional version of Algorithm PE and subdivides the lines at the unspecified middle branches in the trees constructed by Algorithm PE. Here, a ternary tree results whose middle branches index points. A new 1D node is created with plane chosen that is the support plane of an adjacent face to the vertex such that the face is not coplanar with either of the two planes already defining the vertex in the tree. When a vertex has more than three adjacent faces, the third plane is chosen to maximizes the determinant formed from the three support-plane normal vectors selected; that is, the choice maximizes the numerical accuracy of determining the vertex coordinates by intersecting the three planes. Algorithm PV for processing the vertices is omitted.

2.3 Attaching the BRep-Index to the BRep

The ternary tree just created is now attached to the brep data structure by classifying the brep vertices, edges and faces. Since the brep has been fragmented, each element of B' corresponds to exactly one leaf of the tree. So, the following algorithm connects each leaf of T' to the element of B' it represents.

Algorithm ATB. Given the ternary tree rooted that was created by Algorithm PF, PE, and PV, and the fragmented brep B', attach the two by processing each topological entity x (i.e., faces, edges, and vertices) of B' as follows:

1. Descend the tree starting from the root node down to some fringe node n' by comparing x against $P(m)$ of each internal node m along the path and moving down the ABOVE(m), ON(m), or BELOW(m) subtrees as appropriate.
2. Replace the appropriate empty child of node n' with x depending on whether x is above, on, or below $P(n')$.

The comparison of x against an oriented plane P requires one point/plane comparisons for a vertex, two for an edge, and several for a face (i.e., one for each vertex incident to the face). In the case of a face, since the face is known to lie completely above, on, or below the planes of the tree nodes, the first incident vertex that is not on the plane determines the side.

2.4 Reducing the BRep

Recall that the original brep B has been fragmented during the construction of tree T'. In the third step of creating the brep-index, the fragmented brep B' is reduced to B by combining elements of B' that are on the same element of B. This is done incrementally, as sketched next.

Consider merging two adjacent coplanar faces f_1 and f_2 of B' that are adjacent in the edge e of B'. We remove e, f_1, and f_2 from B' and replace them with a new face f that covers all three, thus obtaining a brep B_1. Simultaneously we merge the three corresponding leaves in T' obtaining a directed acyclic graph (dag) T_1, where the new leaf points to f in B_1. Iterating this process, we merge all coplanar, adjacent faces and, by an analogous computation, merge all adjacent collinear edges. The result is a minimal brep $\overline{B}$ and an indexing dag $\overline{T}$. Assuming that the original brep B is minimal, we have $\overline{B} = B$. The dag $\overline{T}$ is the brep index T.

Figure 1 shows the updated brep-index attached to the reduced brep. In the figure, the top face of B is fragmented into four adjacent, coplanar faces of B'.

3 Compressing the Brep-index

After reduction, the final brep is minimal. Moreover, the associated brep-index is complete and can be used for solving the point/solid and the line/solid classification problems described in Section 4. However, unlike the brep, the brep-index can usually be compressed further. That is, by restructuring the tree so that the cut planes are queried in a different order, other cut planes may become redundant and the corresponding nodes eliminated. In this section, we propose a way for restructuring and compressing the brep-index.

We write the brep-index in a prefix notation. If n is a node of the tree, the subtree rooted at n is written

$$P(a, b, c)$$

where P is the cut plane at n, and a, b, and c denote the left, middle, and right subtree of n, respectively.

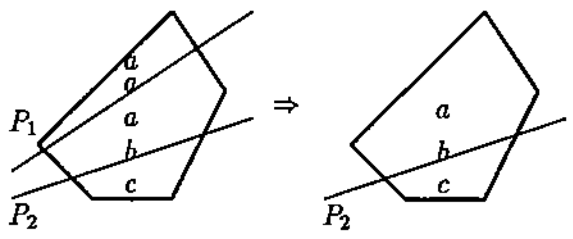

Figure 2: Example of where Rule 6 applies.

We develop tree rewrite rules that effect the compression of T. At this time, we do not have a proof that the rules always find the minimum size brep-index.

The first rule states that any nonsingular region can be cut by a plane. The plane must be transversal with the region, and so it cannot contain the region.

Lemma 1 (Cut) *For* $P \cap R(a) \subset R(a)$ *and* $\dim(a) > 0$,

$$a \Rightarrow P(a,a,a). \tag{R1}$$

Conversely, a plane cutting through a homogeneous region is redundant and can be removed.

Lemma 2 (Eliminate)

$$P_1(a,a,a) \Rightarrow a. \tag{R2}$$

Flipping the orientation of a plane causes the order of the children to reverse.

Lemma 3 (Flip)

$$P(a,b,c) \Leftrightarrow -P(c,b,a). \tag{R3}$$

Two transversal cuts passing through a given region can be ordered in two ways.

Lemma 4 (Swap)

$$P_1\left(P_2(a,b,c), P_2(d,e,f), P_2(g,h,i)\right) \Leftrightarrow P_2\left(P_1(a,d,g), P_1(b,e,h), P_1(c,f,i)\right). \tag{R4}$$

Three nonintersecting cuts in a given region with a redundant middle plane allow the removal of the middle plane. This rule has four variations, only one of which is given here. For each, the consequent has two equal forms.

Lemma 5

$$P_1\left(P_2(a,b,c), c, P_3(d,e,c)\right) \Rightarrow \left\{ \begin{array}{l} P_2\left(a,b,P_3(d,e,c)\right) \\ P_3\left(d,e,P_2(a,b,c)\right). \end{array} \right. \tag{R5}$$

Similar to the above rule, the following rule handles a region with two nonintersecting cuts, one of which is redundant. This rule also has four variations, only one of which is given.

Lemma 6

$$P_1\,(a,a,P_2(b,c,a)) \Rightarrow P_2(b,c,a). \tag{R6}$$

The next rule works only for 1D regions, and is the analogue of a left rotation in a binary search tree. The corresponding right rotation has similar form.

Lemma 7 (Left Rotation) *For* $\dim(d)=0$,

$$P_2(P_1(a,b,c),d,P_3(e,f,g)) \Rightarrow \begin{cases} P_3(P_2(P_1(a,b,c),d,e),f,g) & \textit{if } d \textit{ is above } P_3 \\ P_3(e,f,P_2(P_1(a,b,d),d,g)) & \textit{if } d \textit{ is below } P_3. \end{cases} \tag{R7}$$

To illustrate the use of the rewrite rules, we compress the brep-index of Figure 1. In each tree, the subtrees to be changed are superscripted with the number of rule that is being applied. We begin the rewritting by noting that P_5 is redundant. To remove P_5, we propagate it to the bottom of the tree where it will disappear by Rules (R2) and (R7).

$$\begin{aligned}
& P_1(P_3(\mathbf{out},P_2(\mathbf{out},v_3,e_3),\alpha),\\
& \quad P_3(P_4(\mathbf{out},v_2,e_1),v_2,\beta),\\
& \quad \gamma)\\
& \quad \alpha \;=\; P_5(P_2(\mathbf{out},e_2,f),P_2(\mathbf{out},e_2,f),P_2(\mathbf{out},e_2,f))^{\mathrm{R4}}\\
& \qquad =\; P_2(P_5(\mathbf{out},\mathbf{out},\mathbf{out})^{\mathrm{R2}},P_5(e_2,e_2,e_2)^{\mathrm{R2}},P_5(f,f,f)^{\mathrm{R2}})\\
& \qquad =\; P_2(\mathbf{out},e_2,f)\\
& \quad \beta \;=\; P_5(P_2(\mathbf{out},e_2,f),f,f)^{\mathrm{R6}}\\
& \qquad =\; P_2(\mathbf{out},e_2,f)\\
& \quad \gamma \;=\; P_5(P_2(\mathbf{out}^{\mathrm{R1}},P_4(\mathbf{out},v_4,e_2),P_4(\mathbf{out},e_4,f))^{\mathrm{R4}},\\
& \qquad\qquad P_4(\mathbf{out},e_4,f),P_4(\mathbf{out},e_4,f))\\
& \qquad =\; \ldots\\
& \qquad =\; P_4(\mathbf{out},P_2(\mathbf{out},v_4,e_4),P_2(\mathbf{out},e_2,f))\\
= \;& P_1(P_3(\mathbf{out},P_2(\mathbf{out},v_3,e_3),P_2(\mathbf{out},e_2,f)),\\
& \quad P_3(P_4(\mathbf{out},v_1,e_1),v_2,P_2(\mathbf{out},e_2,f)),\\
& \quad P_4(\mathbf{out},P_2(\mathbf{out},v_4,e_4),P_2(\mathbf{out},e_2,f))).
\end{aligned}$$

Now we note that $P_2(\mathbf{out},e_2,f)$ appears multiple times as a cut of P_1. We conclude that we should have cut with P_2 before cutting with P_1, and proceed to exchange the order of the cuts by moving P_2 up and exchanging with Rule R4.

$$\begin{aligned}
& P_1(P_3(\mathbf{out}^{\mathrm{R1}},P_2(\mathbf{out},v_3,e_3),P_2(\mathbf{out},e_2,f))^{\mathrm{R4}},\\
& \quad P_3(P_4(\mathbf{out},v_1,e_1),v_2,P_2(\mathbf{out},e_2,f))^{\mathrm{R7}},\\
& \quad P_4(\mathbf{out},P_2(\mathbf{out},v_4,e_4),P_2(\mathbf{out},e_2,f)))\\
= \;& P_1(P_2(\mathbf{out},P_3(\mathbf{out},v_3,e_2),P_3(\mathbf{out},e_3,f)),\\
& \quad P_2(\mathbf{out},e_2,P_3(P_4(\mathbf{out},v_1,e_1),v_2,f)),\\
& \quad P_2(\mathbf{out},P_4(\mathbf{out},v_4,e_2),P_4(\mathbf{out},e_4,f)))^{\mathrm{R4}}\\
= \;& P_2(P_1(\mathbf{out},\mathbf{out},\mathbf{out})^{\mathrm{R2}},\\
& \quad P_1(P_3(\mathbf{out},v_3,e_2),e_2,P_4(\mathbf{out},v_4,e_2))^{\mathrm{R5}},\\
& \quad P_1(P_3(\mathbf{out},e_3,f),P_3(P_4(\mathbf{out},v_1,e_1),v_2,f),P_4(\mathbf{out},e_4,f)))\\
= \;& P_2(\mathbf{out},P_3(\mathbf{out},v_3,P_4(\mathbf{out},v_4,e_2)),\\
& \quad P_1(P_3(\mathbf{out},e_3,f),P_3(P_4(\mathbf{out},v_1,e_1),v_2,f),P_4(\mathbf{out},e_4,f))).
\end{aligned}$$

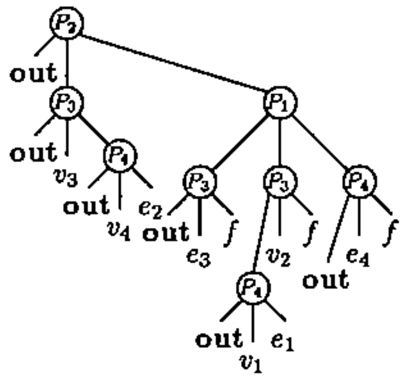

Figure 3: The compressed brep-index of the initial brep-index of Figure 1.

The original brep-index tree consisted of 17 internal nodes; the final tree has only eight nodes, as show graphically in Figure 3. In this simple example, we achieved a reduction in the size by more than fifty percent. This brep-index is minimal for this example.

Theorem 1 *A brep-index for a brep B in which no two edges are collinear and no two faces are coplanar partitions space into at least*

$$1+2\sum_{k=0}^{d-1} n^k,$$

regions, where n^k is the number of k-faces (i.e., vertices, edges and faces) of B.

Proof: Consider Step 1 of Algorithm CBI. Since we are interested in the lower bound assume w.l.o.g. that P can be partitioned by correctly choosing cut planes that do not cause any fragmentation. Initially there is one region, $\mathbf{E}^3$. Since any one hyperplane (i.e., plane, line, point) contains at most one k-face, each k-face partitions the enclosing $(k+1)$-dimensional region into three subregions. That is, one region is replaced with three. Therefore, each k-face contributes two regions to the original one. □

The theorem shows that the brep-index of a 2D polygon with v vertices, and e edges partitions space into at least $1+2v+2e$ regions. Since a ternary tree with n leaves has $n-1/2$ internal nodes, we get $v+e$ as the number of internal nodes. In the example of Figure 1 with four vertices and four edges, this implies the existence of a brep-index with at least eight internal nodes and 17 regions. For 3D, the theorem states that the number of regions and the number of internal nodes of the brep-index is at least $1+2(v+e+f)$ and $(v+e+f)$, respectively, where v, e, and f is the number of vertices, edges and faces, respectively. When colinear edges and coplanar faces are allowed, we have $1+2(v+e'+f')$ for the minimum number of regions, where e' and f' is the number of maximal faces and edges in B. A maximal face consists of all the

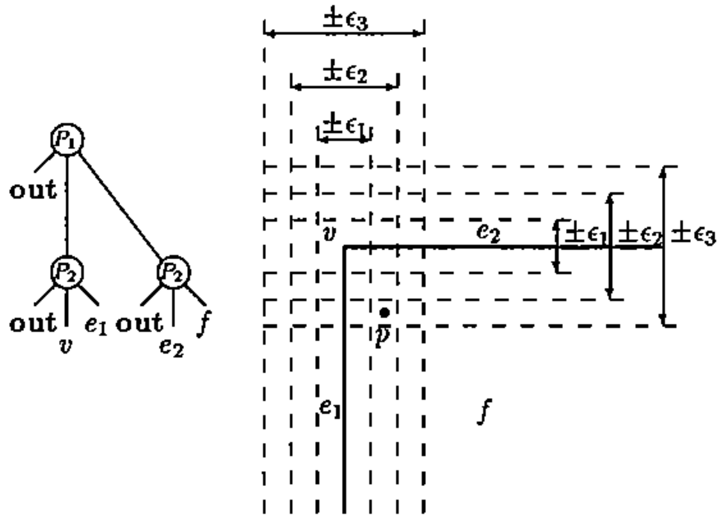

Figure 4: Example of classifying point p under three different tolerances.

connected faces lying in a given plane. A maximal edge is defined similarly. So, a compressed brep-index has size linear in the number of vertices, edges and faces.

4 Point and Line Classification

Most classification problems can be reduced to a point/solid and a line/solid classification problem. This includes problems such as boundary merging, collision detection, and ray casting. The solution of a point/solid classification can return whether the point is inside, outside, or on the boundary of the solid. Additionally, when the point is on the boundary of the solid, the solution could include a pointer to the topological entity of the brep that the point lies on. The brep-index gives this additional information. Likewise, the solution for the line/solid classification can simply return what portions of the line lie inside, outside or on the solid. In addition, the solution could also include pointers to the topological entities that the line crosses or lies in, and exactly at what points. Again, the brep index delivers this additional information. Moreover, the line/solid classification routine can be implemented with a high degree of robustness, as explained below.

Since the exact answer to "a lying on b" cannot be given precisely in floating point arithmetic, the classifications depend on a tolerance $\epsilon > 0$. We assume that ϵ is less than half the minimum distance between any two nonincident topological entities in the brep. That is, the smallest distance between any two nonincident pairs of any combination of vertices, edges or faces.

Consider first the point/solid classification problem.

Algorithm PSC. Given the root node n of a brep-index, a tolerance ϵ, and a point p, return the classification of p.

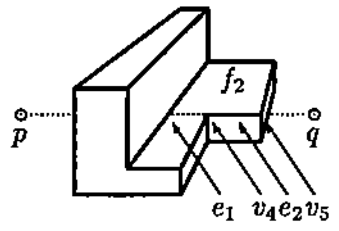

Figure 5: Line-segment/solid classification example.

1. If n is not an internal node, then return n and skip the remaining steps.
2. Let d be the signed distance of p to $P(n)$. Then reassign

$$n = \begin{cases} \text{ABOVE}(n) & \text{if } d > \epsilon \\ \text{BELOW}(n) & \text{if } d < -\epsilon \\ \text{ON}(n) & \text{otherwise} \end{cases}$$

3. Repeat from Step 1.

We use Figure 4 as an example, to show the result of classifying a point p lying in a face f for three different values of ϵ, where $0 < \epsilon_1 < \epsilon_2 < \epsilon_3$. With ϵ_1, p is found to lie on face f since p is below both P_1 and P_2; with ϵ_2 it is found to lie on edge e_1 since p is on P_1 and below P_2; with ϵ_3 it is on the vertex v since it lies on both P_1 and P_2.

Consider now the line/solid classification problem and refer to Figure 5 for an example. The line segment $\overline{pq}$ when classified, results in the list

$$[\mathbf{out}, (f_1, p), \mathbf{in}, (e_1, q), f_2, (v_4, p_4), e_2, (v_5, p_5), \mathbf{out}], \tag{1}$$

where p, q, p_4, p_5 are the computed points at which the line penetrates the corresponding entities, and the v's, e's and f's are the vertex, edge and face nodes of the brep.

An intuitive way to classify the line is to pass the line segment down the tree, split the line segment whenever it crosses $P(n)$ at a node n, and classify the portions above, on, and below recursively. This, however, gives incorrect results when the line segment forms a very small angle with a cutting plane. Figure 6 shows a line segment close to a vertex v. Although the line segment can be arbitrarily close to v, the line segment crosses plane P_1 arbitrarily far away from v, because of the small angle between P_1 and the line. So the computed point on P_1 above P_2 and the two created segment to fall in the regions labeled **out** (refer to Figure 4 for the tree). This results in an incorrect classification, partly because the thickness of the $\pm\epsilon$ region around the planes has not been well accounted for in the above strategy.

A correct classification is obtained when the interval of the line passing through the $\pm\epsilon$ region around each $P(n)$ is accounted for. We represent the line segment to be classified parametrically, as

$$q = p_0 + \mathbf{v} \cdot t, \tag{2}$$

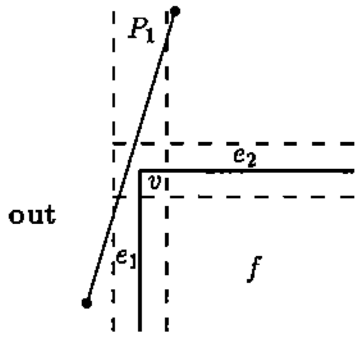

Figure 6: A cross section of a block showing only the top left edge, and its two indicent faces, and a line segment lying arbitrarily close to the edge.

		t_u Below	On	Above
		$[t_l, t_u]$	$[t_l, t_-]$	$[t_l, t_-]$
	Below	n/a	$[t_-, t_u]$	$[t_-, t_+]$
		n/a	n/a	$[t_+, t_u]$
		$[t_-, t_u]^*$	n/a	n/a
t_l	**On**	$[t_l, t_-]^*$	$[t_l, t_u]$	$[t_l, t_+]$
		n/a	n/a	$[t_+, t_u]$
		$[t_-, t_u]^*$	n/a	n/a
	Above	$[t_+, t_-]^*$	$[t_+, t_u]^*$	n/a
		$[t_l, t_+]^*$	$[t_l, t_+]^*$	$[t_l, t_u]^*$

$*$ — reverse the list
n/a — not applicable

Table 1: Intervals below $P - \epsilon$, in $P \pm \epsilon$, and above $P + \epsilon$ for Algorithm LSC.

for $0 \leq t \leq 1$, and keep an ordered list of parameters

$$[t_0 = 0, t_1, \ldots, t_k = 1],$$

with $t_i < t_{i+1}$ to represent the line intervals in the various support regions.

Algorithm LSC. Given an interval $[t_l, t_u]$ on a parameterized line (Eq. (2)) and the root node n of a brep-index, classify the interval, and return the ordered list of classifications.

1. Classify the interval, and obtain the list L, as follows:
   (a) If n is a leaf, then form the list $[n]$, return it, and skip the remaining steps.
   (b) Let t_+ and t_- be the parametric values at which the line segment intersects the positive ϵ plane and the negative ϵ plane, and let b, o, a be the intervals from Table 1.

(c) Referring to Table 1, classify the intervals b, o, a, in the appropriate subtrees of node n from Step 1a obtaining the classification lists B,O,A. Note that some intervals are not classified and are dropped. They correspond to the n/a entries in the table.

(d) Append the indicated lists, possibly after reversal, and return the result.

2. Replace each adjacent pair $[x_i, x_{i+1}]$ of L for which $x_i = x_{i+1}$, with $[x_i]$.

3. Replace each sequence $[x_i, \ldots, x_j, \ldots, x_k]$ of L for which $\dim(x_i) > \dim(x_j) < \dim(x_k)$, with the triple $[x_i, (x_j, p_j), x_k]$, where p_j is the closest point on the line to the entity x_i.

Algorithm LSC performs several tasks. It classifies the line segment obtaining the list L of classifications, it reduces L, and it computes the points on the boundary of the solid at which the line segment penetrates.

The list of classifications consists of the labels **out**, and **in** and the elements which are the vertices, edges and faces from B. Due to the line segment crossing many partitions of space, L consists of equal adjacent classifications. As an example, the classification of Eq. (1) after Step 1 of Algorithm LSC may have been the sequence,

$$[\mathbf{out}, f_1, \mathbf{in}, \mathbf{in}, e_1, f_2, f_2, v_4, e_2, e_2, e_2, v_5, \mathbf{out}].$$

Since adjacent and equal elements in L signify the same classification between which the line is not penetrating a boundary of the solid, the duplicate elements are merged in Step 2.

In Step 3, the support regions on the boundary where the line segment penetrates the solid are determined, and adjacent regions grazed by the line segment are removed. To understand this, consider again the example of Figure 6. List L before applying Step 3 is

$$[\mathbf{out}, e_1, v, \mathbf{out}].$$

We recognize that edge e_1 appearing in the list must be a result of the imposed around the vertex and should be omitted from the list, yielding the correct classification

$$[\mathbf{out}, (v, p), \mathbf{out}],$$

for some p on the line segment that is closest to v. The reasoning goes like this. If e_1 appears in the list correctly, it is either being crossed transversly by the line segment, or it is collinear with it. Assume that it crosses transversly, then v should not appear in the list. Now assume that it is colinear with it, then either the line segment starts in the middle of e_1, and the first **out** should not appear, or else it starts outside the edge and there should also appear the other vertex of the edge.

5 Conclusion

The brep-index has been implemented in Common Lisp. The code was added to ProtoSolid [18], a solid modeler written in Common Lisp that runs on our Symbolics 36xx lisp machine, with a hardware speed of approximately 0.75 MIPS.

Size N	Index Nodes	Index Height	Average Height	Time (sec)
Sphere				
58	58	16	7.8	1.7
134	135	19	9.4	4.5
242	245	18	9.9	7.7
382	463	18	10.4	15.6
1562	2309	24	13.0	104.0
Torus				
64	110	9	6.1	1.9
144	299	11	7.3	8.0
256	435	14	8.6	12.5
400	1019	19	10.2	39.0
1600	4721	27	12.6	233.0

Table 2: A sphere and a torus of various size N, where $N = V + E + F$.

To illustrate the sizes of the brep-index trees, spheres and torii with various number of faces have been created. Table 2 shows the number of vertices, edges and faces, the number of internal nodes in the uncompressed brep-index, the height of the brep-index, the average height of the brep-index, and the time in seconds to create the brep-index.

A small average height of the tree is important to achieve efficiency, because the average classification cost of a point with respect to a solid is linear in the height of the brep-index. Thus, the classification of n points by a brep-index of average height H requires $O(nH)$ steps. The cost of a line segment classification depends on the number of internal nodes that split the line segment, and is proportional to the length of the final classification.

The brep-index was integrated into ProtoSolid and provide the solid modeling support to Project Newton [20, 7, 8]. Project Newton is a dynamic simulation system that uses rigid body dynamics to simulate the motion of objects. With the use of the brep-index, we can simulate systems in which collisions between objects of any shape must be detected and analyzed. One example is the *tumble rings puzzle* [6]. Two chains of rings are interlinked in a pattern shown in Figure 7, with one chain beginning with the topmost ring, and the second chain beginning with the shaded ring just below. When the shaded ring is held fixed, the first chain drops giving the impression that the top ring tumbles to the bottom.

The rings are modeled as breps with 128 vertices, 256 edges and 128 faces. The uncompressed brep-index consists of 1059 internal nodes, and has a maximum depth of 16 and an average depth of 9. A trivial compression heuristic using only Rules 2, 5 and 6 repeatadly, resulted in a tree with 1003 internal nodes which is clearly short of the 512 nodes expected.

The simulation requires 21 pairs of rings to be checked for contact at every time step. The contacts are determined by classifying every vertex and edge of each ring

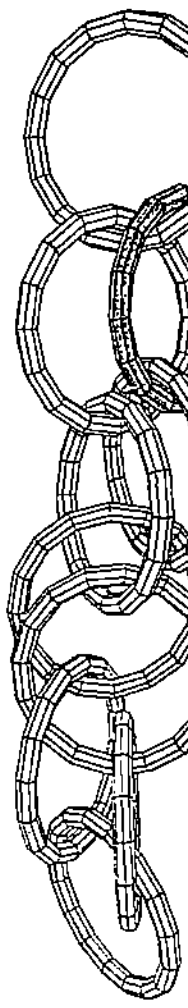

Figure 7: Ten link chain simulated by Newton.

with the brep-index of the other ring. Since both rings of each pair are the same shape, the classification of all 10 rings uses only a single brep and its brep-index by transforming the edges and vertices to account for relative position. With that, the simulation requires approximately 7 seconds of simulated time to tumble the rings. At an average simulated time step of 0.00025 seconds, the simulation takes approximately 28000 time steps and therefore evaluates 560000 pairs of rings for contact. Thus, roughly 3×10^9 vertices and 6×10^9 edges are classified.

This example illustrates that even seemingly simple looking simulations can require an enormous computing effort. With the brep-index such simulations can be performed robustly and with reasonable efficiency. Currently we are perfecting the rules and an algorithm for compressing the trees.

Acknowledgements

This work has been supported by NSF Grant CCR-86-19817.

References

[1] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics (Proc. of SIGGRAPH '88)*, 23(3):223–231, July 1989.

[2] B. Baumgart. Winged–Edge Polyhedron Representation. CS–320, Stanford University, Stanford, California, 1972.

[3] I. C. Braid. The synthesis of solids bounded by many faces. *Communications of the ACM.* 18(4):209–216, April 1975.

[4] D. P. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *Journal of Algorithms*, 8:348–361, 1987.

[5] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *Conf. Proc. of SIGGRAPH '80*, 14(3):124–133, July 1980.

[6] M. Gardner. "Mathematical Games," *Scientific American*, 207:120–126, August 1962.

[7] C. M. Hoffmann and J. E. Hopcroft. Simulation of physical systems from geometric models. *IEEE Journal of Robotics and Automation*, RA–3(3):194–206, June 1987.

[8] C. M. Hoffmann and J. E. Hopcroft. Model generation and modification for dynamic systems from geometric data. *CAD Based Programming for Sensory Robots*, F50:481–492, 1988.

[9] M. Karasick. *On the Representation and Manipulation of Rigid Solids.* PhD thesis, McGill University, 1988.

[10] D. T. Lee and F. P. Preparata, "Location of a point in a planar subdivision and its applications," *SIAM Journal on Computing*, 6(3):594–606, Sept, 1977.

[11] M. Mäntylä. Localized Set Operations for Solid Modeling. Computer Graphics, 17(3):279–288, July 1983.

[12] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, Rockville, Maryland, 1988.

[13] W. C. Thibault and B. F. Naylor. Set Operations on Polyhedra Using Binary Space partitioning Trees. *ACM Computer Graphics SIGGRAPH '87.* 21(4):153–162, July 1987.

[14] B. Naylor. Binary Space Partitioning Trees as an Alternative Representation of Polytopes. *Computer-Aided Design*, 22(4):250–252, May 1990.

[15] A. Paoluzzi. Motion Planning+Solid Modeling=Motion Modeling. Dip. di Informatica e Sistemistica, Università "La Sapienza", Techn. Rep. 17–89, Rome, November 1989.

[PS85] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction.* Springer-Verlang, 1985.

[16] A. A. G. Requicha and R. Tilove. Mathematical Foundations of Constructive Solid Geometry: General Topology of Regular Closed Sets. Technical Memorandum TM–27a. Production Automation Project, University of Rochester, Rochester NY, March 1978.

[17] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *ACM Computer Graphics SIGGRAPH '87*, 21(4):153–162, July 1987.

[18] G. Vaněček Jr. Protosolid: An inside look. CAPO Report CER-89-26, Purdue University, Department of Computer Science, West Lafayette, IN 47907, November 1989.

[19] G. Vaněček Jr. *Set Operations on Polyhedra using Decomposition Methods.* PhD thesis, University of Maryland, College Park, Maryland, June 1989.

[20] G. Vaněček Jr. *A Data Structure for Analyzing Collisions of Moving Objects.* The Hawaii International Conference on System Sciences, HICSS–24, Kailua Hawaii, January 1991.

[21] G. Vaněček Jr. and V. Ferrucci. *A Spatial Index for Simplicial Complexes in d Dimensions*, to appear as a Leonardo Fibonacci Institute Report, Trento Italy, December 1990.

[22] F. Yamaguchi and T. Tokieda. Bridge Edge and Triangulation Approach in Solid Modeling. "Frontiers in Computer Graphics '84." Springer–Verlag, 44-65, 1985.