13

Basic solid modeling

This chapter is dedicated to discussing the computer representation of mathematical models of physical bodies and some main algorithms on such so-called *solid models.* The attention of "classic" solid modeling is focused on representational issues for computer models of physical bodies. Its core topics concern Boolean operations and domain integration: the former allow the union, intersection or difference of solid models; the latter is used to compute volumes and inertial properties. Simulating numerically controlled (NC) machining [MV92] is another use of solid modeling technology. It also supports offsetting, reverse engineering from both physical and engineering models to computer models, and dimensional tolerancing analysis [RR86, VMJ97, Voe98]. We start this chapter by outlining Voelcker's and Requicha's theoretical approach, and their standard taxonomy of representation schemes. Then some main issues about boundary representations are introduced. Finally, we discuss two simple approaches to Boolean operations and volume integration over polyhedral objects by using a boundary triangulation. Our aim is to present a basic introduction to some classic notions in solid modeling. For an in-depth comprehensive review of both theoretical foundations and research directions, the interested reader is referred to the stimulating survey by Vadim Shapiro [Sha01].

13.1 Representation scheme

The concept of a *representation scheme*, located at the foundations of solid modeling, originates in the theoretical and applied research in design and manufacturing automation carried out by Project Automation Project (PAP) at the University of Rochester in late 1970s. In particular, the idea of a representation scheme was formalized by Aristides Requicha into his milestone paper [Req80]. A new interesting framework for both geometric and solid modeling and its practical implications are discussed by Rappoport in [Rap95].

Geometric Programming for Computer-Aided Design Alberto Paoluzzi
 ISBN 0-471-89942-9

13.1.1 Definition and properties

A *representation scheme* is defined as a mapping $s : M \to R$ between a set M of mathematical *models* of solid objects and a set R of computer *representations*, see Figure 13.1. In particular:

1. The set M contains appropriate mathematical models of the class of solid object that the scheme is designed to represent.
2. The set R contains symbolic representations, i.e. suitable data structures, built according to some appropriate computer grammar.

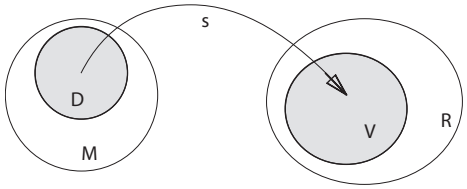


Figure 13.1 The representation scheme s is a mapping between a set M of mathematical models and a set R of computer representations

Domain A representation scheme s is usually defined only on a subset $D \subset M$ of mathematical models, that is called the *domain* of the scheme. In other words, not every element of a certain class of models might be represented in a given scheme, but this one could be designed to represent only a subset of the model collection. E.g., a scheme used to represent plane polygons should be able to filter *a priori* non-simple polygons (i.e. with self-intersecting boundary), because they are not interesting in most modeling problems.

Validity Analogously, the codomain $V = s(D) \subseteq R$ of a scheme s does not necessarily coincide with the target set R of syntactically well-formed representations, i.e. built according to the given grammar. The set V of representations which are not only syntactically correct, but also semantically significant, because they correspond to some model in the domain of the scheme, is called the *validity set* of the scheme.

When designing a representation scheme for a solid modeler it is important to device some *validity test* to check if a given representation really corresponds to some object in the domain of the scheme. As a matter of fact, syntactically correct representations may not be semantically significant.

Let us consider, e.g., the linked lists of vertices as representations of plane polygons. If the domain of the scheme is restricted to simple (non self-intersecting) polygons, then the statement appears false that every sequence of 2D points corresponds to some simple polygon. A typical validity test in this case should look for pair-wise intersection of boundary edges, given by couples of adjacent points.

Completeness A valid representation $r \in V$ is said to be *complete* when it is associated by the scheme s with just one model in the domain D of the scheme. In formal terms, r is said to be complete when

$$|s^{-1}(r)| = 1.$$

The scheme s is also said to be *complete* when every representation in V is complete. In other words, the scheme s is complete when the inverse mapping s^{-1} is a *function*.

The concept of completeness should be considered from an informational point of view. A representation which corresponds to more than one model is said non complete, because it captures some information which is common to all the associated models, and therefore is not sufficient to completely specify any one of them. A non complete representation is, in some sense, too general and not sufficiently specific. Conversely, a complete representation completely specifies the model that it represents.

Unicity A representation $r \in s(m)$ is said to be *unique* when the set $s(m) \subset V$ contains just one element. Analogously, a representation scheme s is complete when $|s(m)| = 1$, for each model $m \in D$.

When a scheme s is both complete and unique, it establishes a one-to-one mapping between the domain and the validity set of the scheme. Such a kind of scheme is called *canonical*, and the associated representations are called canonical representations. A canonical scheme is quite unusual in solid modeling, depending on the high computational costs needed to guarantee the unicity of such kind of schemes.

With a canonical scheme the identity test between two representations $r, q \in V$ could be solved in a purely syntactical way, while conversely it requires serious semantical comparisons of the kind:

$$(s^{-1}(r) \cup s^{-1}(q)) - (s^{-1}(r) \cap s^{-1}(q)) \stackrel{?}{=} \emptyset.$$

Most representation schemes for geometric objects are not unique, because the same model may correspond to several representations. Typical reasons of non-uniqueness are of a *permutational* and *positional* nature.

An example of permutational non-uniqueness is given by the representation of a plane polygon as the linked list of its n vertices. In this case, each one of the n cyclic permutations of vertices gives one equivalent representation of the same polygon. A positional non-uniqueness is introduced when the representation is defined modulus a rigid transformation.

r-sets In some sense, the usefulness of a scheme s is measured by the size of its domain, i.e. by the number of mathematical models it is able to represent. The larger is the domain, the greater are the expressive power of the scheme and its usefulness for the purposes of a modeling system.

The Rochester group pointed out as a useful domain for a solid modeling representation scheme the collection of so-called *r-sets*, i.e. the collection of *regular, closed, bounded and semi-analytic subsets* of $\mathbb{E}^3$.

It may be useful to remember that a subset $S \subset \mathbb{E}^n$ is called:

1. *regular* when it is homogeneously n-dimensional, i.e. when each point $\boldsymbol{x} \in S$ has a neighborhood $N(\boldsymbol{x}) \subset S$ of dimension n;

2. *closed* when it contains its boundary. In other words, when

$$S = \text{clos}(S) = S \cup \partial S;$$

3. *bounded* when each point $\boldsymbol{x} \in S$ has bounded coordinates. In other words, when the distance between every pair of points in S is bounded;
4. *semi-analytic* when the set can be expressed as a Boolean combination of sets of type $\{\boldsymbol{x} \in \mathbb{E}^n | f_i(\boldsymbol{x}) \leq 0\}$, where each f_i is an analytic function.

A function $f : \mathbb{E}^n \to \mathbb{R}$ is said to be *analytic* when it can be expanded into a convergent power series in the neighborhood of each point $\boldsymbol{x}_0$ of its domain:

$$f(\boldsymbol{x}_0) = \sum_{n=0}^{\infty} a_n (\boldsymbol{x} - \boldsymbol{x}_0)^n, \qquad \text{with } |\boldsymbol{x} - \boldsymbol{x}_0| \leq r, r > 0.$$

The sum, difference and product of analytic functions are again analytic. If $f(\boldsymbol{x})$ is analytic at $\boldsymbol{x}_0$, then it is differentiable arbitrarily often in some neighborhood of $\boldsymbol{x}_0$, and:

$$f(\boldsymbol{x}_0) = \sum_{n=0}^{\infty} \frac{f^{(n)}(\boldsymbol{x}_0)}{n!} (\boldsymbol{x} - \boldsymbol{x}_0)^n, \qquad (\textit{Taylor series}).$$

The concept of r-set played a very important role in the development of solid modeling as a discipline, since it seems to capture the more important aspects of the object's *solidity*.

Manifolds A *manifold* object of dimension n is defined as a point set where each point has some neighborhood homeomorphic to the n-disc. This means that there is some neighborhood of the point which may be bicontinuously transformed to such a disc.

Let us note that an n-dimensional r-set S may have a *non-manifold* boundary (see Section 13.1), where some points in ∂S do not have a neighborhood in ∂S homeomorphic to the $(n-1)$-dimensional disc. See, e.g. Figure 13.2.

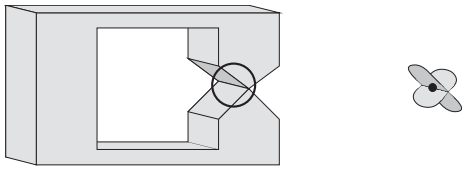


Figure 13.2 Non-manifold boundary points: they do not have a neighborhood homeomorphic to the 2-dimensional disc

It is very important to notice that a set of manifold models is not closed with respect to the property of manifoldness when considering Boolean set operations. For example, the union of two manifolds is not necessarily a manifold. Let us imagine, e.g., the union of two cubes touching along one edge or vertex. Thus, a representation scheme needs to take this point into careful account; in other words, it must be able to represent both manifold and non-manifold objects.

13.1.2 Taxonomy of representation schemes

A taxonomy of the more important representation schemes used in solid modeling and CAD systems developed in the 1970s was introduced by Ari Requicha in his milestone paper [Req80]. Such a classification continues to be valid more than 20 years later, and is very useful both in comparing systems existing today and in designing new ones. It is succinctly summarized in the following paragraphs. Several reviews of the state of art in solid modeling have been published in the last two decades. They include [Req80, RV83, Req88, RR92, Ros94, RR99, Sha01].

Primitive instancing This is a sort of procedural approach. Each representation is a *tuple* with the name of a generating procedure or function as the first element, followed by the actual values of its parameters. Both the domain and the codomain of the scheme may be partitioned into non-intersecting classes. There is a one-to-one mapping between classes of models and classes of representations. This kind of approach was given new life with current object-oriented software development techniques, and is largely used with parametric and feature-oriented modeling systems. Some theoretically oriented recent approaches belong to this class of representations [Rag00, RS02]. It actually needs some underlying more abstract representation scheme where methods of general utility (like Boolean operations and integration operators) are really implemented.

Enumerative schemes A solid model is described by enumerating the set of "full" cells in some partitioning of the embedding space. It is possible to distinguish between schemes using sparse Boolean matrices and schemes based upon hierarchical space decompositions, say *quadtrees* and *octrees*, in 2D and 3D case, respectively. In most cases such representations are approximations of the object's space occupancy, even for linear polyhedra.

Decompositive schemes The object is represented as a set of cells, usually of a given topology. Unlike enumerative schemes, in this case the cell partition is induced by the represented object itself. The BSP (Boundary Space Partition) trees belong to this type of representation [NAT90]. As we have already seen, BSP trees are binary trees where each internal node is associated with one of the boundary hyperplanes of the object, and each leaf node represents a convex cell, either full or empty, of the space partition induced by such a set of hyperplanes. Decompositive schemes also support more abstract and generalized representations of topology [Lie89] and unifying approaches to both geometry representation and physical simulations [JS00].

Boundary schemes These are the more common representation schemes used in computer graphics and in most modeling kernels. The interior of a body is represented by means of its external boundary, and in particular by a partition of it into shells, faces, edges and vertices. Each boundary representation stores in some way some of the binary adjacency relations between the various boundary entities. There is a notable difference between the boundary representations of *manifolds* and those of *non-manifold* objects.

CSG schemes The acronym stands for *Constructive Solid Geometry*. In this scheme the representations are binary trees where internal nodes represent either regularized set operations or affine transformations, whereas leaf nodes represent either primitive solids or implicit halfspaces (usually linear or quadratic). An explicit representation of the boundary is obtained by suitably traversing the tree.

Sweeping schemes In mechanical or civil applications, several solids may be defined as the result of properly moving a surface along an assigned profile curve. The representation in this case contains both the generating surface and curve. A sweeping scheme is particularly useful and simple with parametric representation of surfaces and curves. There are also approaches belonging to this class of schemes which can be reduced to some generalized version of the Cartesian product.

Composite schemes Some commonly used schemes use multiple types of representations. Typically, in a composite scheme, a boundary representation is maintained together with another representation, often a CSG or decompositive or primitive instancing. There are some very interesting composite octree/boundary representations. The simplicial based representation [PBCF93], discussed in a following section, can be considered as a composite boundary/decompositive/sweeping representation.

The following sections are dedicated to discussing the more important types of representation schemes.

13.2 Enumerative schemes

In enumerative schemes each *model* is embedded in a space partition made with space cells of fixed topology and geometry, usually of either hexahedral or simplicial type. The enumerative *representation* is some kind of enumeration of material cells of the model, i.e. of full cells, as opposite to empty cells. Most enumerative representations are *approximate* representations, but they may have variable resolution, depending on the needs of the supported applications. This approach is particularly suitable to easy implementation of Boolean operations and computation of volume integrals.

Two different classes of enumerative schemes can be devised, which are respectively classified as:

1. *flat* enumerative schemes;
2. *hierarchical* enumerative schemes.

13.2.1 Flat enumerative schemes

A parallelepiped subset of the containment space of the model is subdivided into hexahedral (cubic) cells of size either constant or variable by rows, columns or planes. Such representation usually consists of a binary array with three indices in the 3D case, or with two indices in the 2D case. The resolution of this approach cannot be very small when representing objects of large size. Consider in fact, e.g., that a mechanical model with containment box of $1\,m^3$, represented with linear resolution

of $10^{-4}m$, would require an array with 10^{12} cells. It is hence customary to store such a representation using sparse matrices, where only the boundary cells are explicitly memorized, relying on the fact that the transitions between empty and full cells are not frequent in a "solid".

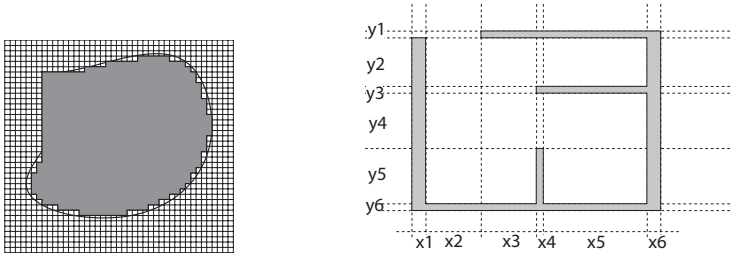


Figure 13.3 Enumerative representations: (a) with constant resolution (b) with variable resolution

Constant resolution In Figure 13.3 we give an example of flat enumerative representation of a curved 2D solid. The cells of plane partition may be labeled full when completely enclosed in the solid, and empty otherwise. A different choice, say with cells labeled empty when completely outside and labeled full otherwise, is equally acceptable, but clearly gives a different approximation of the solid. It is easy to see that such an approach works much better when the model is consistent with the cell grid, i.e. has orthogonal faces and their size is a multiple of the cell step. As we already pointed out, in order to get a good resolution, this approach must be implemented with sparse matrices, that are encoded by using some appropriate compression technique.

Variable resolution In this kind of scheme the *bounding box* of the model is partitioned by three sets of orthogonal planes, normally coplanar to the model faces. The resulting space decomposition into parallelepiped cells is encoded by using some appropriate array. Since the distances between pairs of adjacent cutting planes is generally non-constant, the partition is non-uniform, as shown in Figure 13.3b for the 2D case. An example of this representation, introduced in [Mit77], is given by the primitive `QUADARRAY` defined in Script 7.4.1.

The representation used in the 2D and 3D case, is respectively given by a pair (triplet) of real arrays, that contain the ordered distances between adjacent cutting lines (planes), and by a Boolean array with two (three) indices, which is used to encode the labels (empty/full) of the space partition cells. Such a representation can be encoded as follows, in the 2D and 3D case, respectively:

$$<< \texttt{Xarray}[i_1], \texttt{Yarray}[i_2] >, \texttt{BoolArray}[i_1, i_2] >$$
$$<< \texttt{Xarray}[i_1], \texttt{Yarray}[i_2], \texttt{Zarray}[i_3] >, \texttt{BoolArray}[i_1, i_2, i_3] >$$

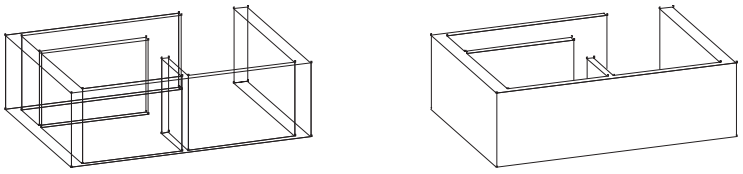


Figure 13.4 3D enumerative representation

13.2.2 Hierarchical enumerative schemes

The *hierarchical enumerative schemes* use a number of trees, which encode different partitioning schemes of the embedding space into cells of different type and size. The most important schemes of this type are 2^n-trees and *bin*-trees, that are briefly discussed in the following subsections.

2^n-Trees

The 2^n*-trees* are *ordered trees* [AU92] characterized by the property that each non-leaf node has exactly 2^n son nodes, respectively denoted as first, second, etc., and as 2^n-th son. When $n = 2$ and $n = 3$ such trees are called *quadtrees* and *octrees*, which are used to represent hierarchical decompositions of the 2D or 3D space, respectively [Sam88, Sam90b, Sam90a].

Quadtrees When the model is embedded in 2D, i.e. when $n = 2$, the 2^n-tree representation is called *quadtree*. Some useful properties of quadtrees follow:

1. a quadtree is a *quaternary* tree (i.e. each non-leaf node has exactly 4 sons);
2. the leafs are either *white* or *black* nodes (i.e. either empty or full);
3. the non-leafs are *gray* nodes (i.e. neither empty nor full);
4. the maximal *depth* of the quadtree is related to its *resolution*.

The number of arcs on the path from the root to a node is called *distance* of the node from the root. *Depth* of a tree is the maximal distance of its nodes from the root. The *resolution* of the quadtree with squared bounding box of size L and depth m is clearly equal to

$$r = L/2^m$$

Example 13.2.1 (Quadtree encoding)
In Figure 13.5 we show both a 2D object given as a hierarchical decompositive representation using the quadtree scheme, and its actual encoding as a labeled tree, where *black*, *white* and *gray* nodes respectively represent full and empty cells, and cells which are neither full nor empty. Note that the sons of a gray node are clockwise ordered, according to the scheme shown in the quadruple of small boxes in the middle top of Figure 13.5.

Octrees When the model is embedded in 3D, i.e. when $n = 3$, the 2^n-tree representation is called an *octree*. In this case the embedding space is subdivided by three pair-wise orthogonal planes, so giving 2^3 cells (either white, black or gray) at each step (see Figure 13.6).

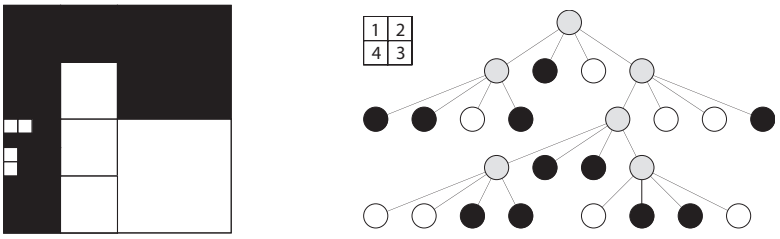


Figure 13.5 Quadtree encoding scheme: (a) 2D object (b) full cells (black), empty cells (white) and decomposed cells (gray)

Bin-trees

The so-called *bin*-trees are *binary* trees representing solids in $\mathbb{E}^n$. In this case tree nodes contain hyperplanes equations of the kind $x_{(d_k \bmod n)} = c_k$, where d_k is the (integer) distance of the node k from the root. Hence, in 2D, *bin*-tree nodes at increasing distance from the root alternatively contain equations such as $x = a_i$ and $y = b_i$. In 3D, node equations will contain either $x = a_i$ or $y = b_i$ or $z = c_i$, alternatively. As in the case of quadtrees and octrees, leaf nodes are labeled either black or white, whereas non-leaf nodes can be considered as gray.

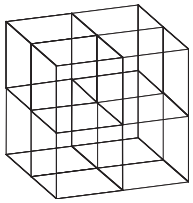


Figure 13.6 Octree: partition of a 3D cell into 8 sub-cells generated by three orthogonal planes

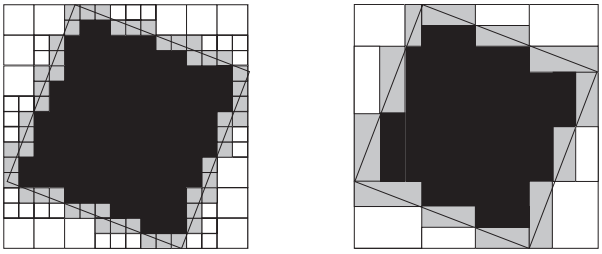


Figure 13.7 Quadtree vs bintree

13.3 Decompositive schemes

In a decompositive representation scheme, a solid object is represented as a set of cells, usually of a given topology. Unlike enumerative schemes, in this case the cell partition is induced by the represented object itself. Also in this case it may be useful

to distinguish between non-hierarchical schemes and hierarchical schemes. Two very useful representations in such classes are the simplicial representations and the BSP-trees, respectively. Both are implemented within the `PLaSM` geometric kernel. They are used by the `MAP` primitive and by the Boolean operations, respectively.

The space of models of decompositive schemes is constituted by the so-called *cellular models*. They have been of increasing interest in recent years [BdB98, ES00]. In particular, the stratification of a solid model into cells provides a common theoretical framework to unify all representations [Sha01]. Furthermore, it also provides a basis for designing a representation-free standard API that is both formal and general [ABC$^+$00].

13.3.1 Simplicial schemes

As we already know from Section 4.5, a *d-simplex* is a sort of *generalized triangle*, defined by the convex combination of a set of $d + 1$ affinely independent points. In particular, a 0-simplex is a point, a 1-simplex is a segment, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, and so on. Analogously, a *simplicial complex* is a sort of *well-formed* generalized *triangulation*, where every two simplices either do not intersect or intersect along a common face, which is in any case a simplex of lower dimension, so that intersecting simplices are well-sticked together. It is possible to show that: (1) every solid may be triangulated by a simplicial complex; (2) every simplicial complex is transformed by a map of its vertices into another simplicial complex. For both such reasons, a representation scheme based on simplicial complexes may be very useful in a geometric modeling environment [PBCF93].

13.3.2 Convex-cell partitioning and covering

In recent years, several geometric environments for solid modeling have been offering the choice of working with either boundary or decompositive representations, by using some kind of convex cells. In particular, a very successful kind of decomposition uses BSP trees. Such trees were adopted as spatial indices for speed acceleration by several computer games, including DOOM and its successors.

When `PLaSM` was designed about one decade ago, a hierarchical scheme called *Hierarchical Polyhedral Complex* (HPC) was devised for it, with only a partial storage of topology of the cell complex. A HPC representation allows efficient representation of dimension-independent polyhedral complexes by using either a *covering* or a *partition* into convex cells of the associated point set, as discussed in Chapter 5.

Currently, `PLaSM` uses a composite scheme which combines both BSP-trees, mainly used for Boolean operations, the simplicial $\mathcal{W}$ representation, used with maps generating polyhedral approximations of curved objects, and the HPC representation, used for product and skeleton operations as well as to implement the hierarchical graph underlying a complex scene or assembly.

BSP-trees

A definition of BSP-trees has already been introduced in Section 10.3.5, when discussing the *hidden-surface removal* (HSR) of 3D scenes. The reader is in this case

referred to that definition and discussion. The only difference introduced when using BSP-trees as a representation of solids is a labeling of leaf nodes. In fact, each leaf of such a tree is associated with a cell of the space partition induced by the hyperplanes stored with the non-leaf nodes of the tree. A label from the set $\{\mathsf{IN}, \mathsf{OUT}\}$ is hence associated with each cell. The solid representation is given in this case by the *labeled* BSP-tree.

Example 13.3.1 (BSP example)
A two-dimensional solid object is generated in Script 13.3.1 as the difference `object` between a `ground` rectangle and a polygonal `hole`. Both the `object` and its 1D skeleton are shown in Figure 13.8b, with the aim of displaying the underlying BSP representation, used by `PLaSM` to perform dimension-independent Boolean operations. The input `hole` is displayed in Figure 13.8a.

Notice that set *difference* between the geometric values stored in `ground` and `hole` symbols is denoted by the symbol "`-`" (minus). Analogously, set *union*, *intersection* and *symmetric difference* (also called *exclusive or*, i.e. XOR) are denoted by "`+`", "`&`" and "∧", respectively.

Script 13.3.1 (BSP example)

```
DEF ground = CUBOID:<19,22>;

DEF hole = MKPOL:<
   <<9.8,4.2>,<3.2,5.8>,<1.6,12.5>,<12.4,18.2>,
   <16.5,8.6>,<9.8,12.5>,<6.2,10.4>>,
   <<1,2,3,7>,<3,4,6,7>,<4,5,6>>,<<1,2,3>>>;

DEF object = ground - hole;
DEF out = (STRUCT ~ [ID, @1]):object;
VRML:out:'out.wrl'
```

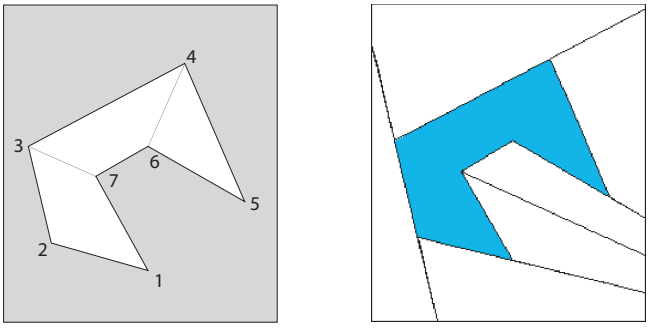


Figure 13.8 (a) Convex cells of the input (b) BSP representation of the difference `object` produced by Script 13.3.1

Hierarchical polyhedral complexes (HPC)

The HPC representation scheme [PFP96] is the main representation used by the geometrical kernel underlying `PLaSM`, and is based on a hierarchical description of the object structure. In particular, the HPC representation scheme describes the geometric shapes as hierarchical collections of polyhedra, where each elementary polyhedron is decomposed in a set of convex cells. Each convex cell is in turn represented as a collection of either vertex vectors or facet covectors.

In such an approach an object is represented as a multilevel hierarchical structure. In particular, each object is represented as a decomposition in a set of objects, which are, in turn, either hierarchical decompositions or elementary polyhedra. For example, the plan of the building floor in Figure 13.9a is represented by the multigraph in Figure 13.9b. Each node with an outgoing arc is called *polyhedral complex*, and is decomposed in a set of disjoint elements. The outgoing arcs that compose the complex are called *polyhedral instances*, and relocate (affinely map) the pointed nodes in the proper position and orientation. The basic objects (associated with leaf nodes) are called *elementary polyhedra*, and are represented in a local coordinate frame as full dimensional complexes of convex cells.

So, a decompositive representation in convex cells is used in the representation of each elementary polyhedron. Along with each cell a rich symbolic description is maintained, and thus the use of numerical information is kept to a minimum. In this way the emphasis is shifted from numeric to symbolic information, with the first being considered less entrusting than the second, thereby increasing the robustness of the supported algorithms.

In PLASM and in several approaches to solid modeling [Bri89, RO90, Mau91, ES00, ABC$^+$00] the reference representation is a cell decomposition. To represent the cells as an intersection of halfspaces has several advantages. In particular, with this representation the extrusion operation, which is of great importance in a dimension-independent approach, is simply a linear operator upon the space of linear functions (covectors) associated with the cell faces. Also, an affine transformation may be applied to this representation by just multiplying the face covectors by the inverse matrix of the transformation. Also, the computation of geometry and topology of the result of the generalized product described in Section 14.4 is very simple.

Progressive difference A "complete" representation of a polyhedral complex must satisfy a quite difficult geometric constraint: the interiors of elementary polyhedra cannot intersect each other. This constraint is needed to guarantee unambiguous representations, but reduces the flexibility of the representation scheme, which cannot represent any design configuration where the design components are actually overlapping. Hence it becomes useful both to extend the domain of the scheme to the class of overlapping sets of polyhedra, and to define an operator `PD` (called the *progressive difference*) to automatically remove the intersections between elementary polyhedra, where they exist. In this way, i.e. by relaxation of the non-intersection constraint, a *weak* representation of hierarchical objects is defined, that we call the *polyhedral sequence*.

Complete and weak representation A *complete representation* of an object is a partitioning of its point set with an unordered set of (hierarchical) polyhedra, while a *weak representation* is simply a covering of the point set. An application of the `PD` operator to a weak representation returns a complete representation, since it subtracts each element of the polyhedral sequence from all the elements with higher position in the sequence. In such a way the ordering on the component instances is translated into a precedence rule on their point sets: points belonging to more than one component object are assigned to the first one in the polyhedral sequence which they belong to.

According to [Req80], we can distinguish between the representation of an object and its abstract model, and define the latter as an element of the set $\mathcal{M}$, which is the domain of both the complete and weak representation schemes. The set $\mathcal{R}$, which contains the complete representations, is the validity set of the scheme. Conversely, the set $\mathcal{R}^*$ of all weak representations is the range set of the scheme, with $\mathcal{R} \subset \mathcal{R}^*$. A more formal and detailed definition of the HPC scheme used in the `PLaSM` language can be found in [PFP96].

Example 13.3.2 (Floor layout)
Consider, e.g., the example in Figure 13.9, where the multigraph structure of the polyhedral complex (b) is represented. An equivalent polyhedral sequence (c) is obtained by replacing the `kitchen`, `living` and `bed2` rooms by their bounding boxes. Their sequence is a weak representation of the object (a) if `bed2` follows the `bath`, and both `kitchen` and `living` follow the `balcony`. An application of the `PD` operator to the weak representation (c) produces the complete representation (b).

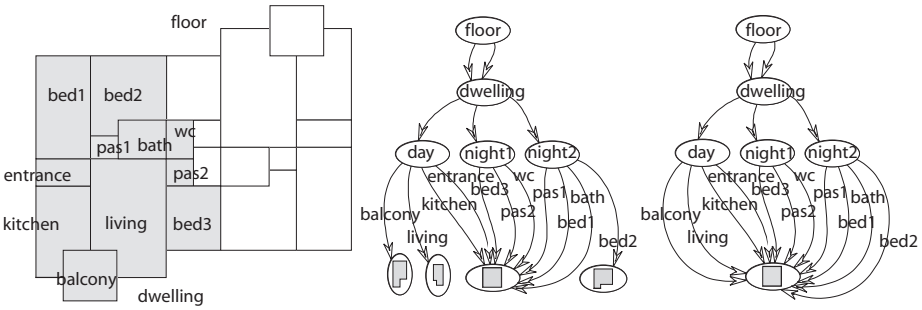


Figure 13.9 Two representations: (a) building floor (b) *complete* representation (partitioning) (c) *weak* representation (covering)

In Script 13.3.2 we give the `PLaSM` definition of the `layout` of building floor shown in Figure 13.10, whose HPC representation is pictorially displayed in Figure 13.9.

The `PLaSM` definition of symbols `day`, `night1` and `night2`, which are the components of the 2D polyhedral complex `dwelling`, is given in Script 13.3.3. Notice that `floor_layout` is an assembly of two `dwelling` instances, where the second is flipped and translated both horizontally and vertically.

Notice also that the model embedded in 3D displayed in Figure 13.10 is generated by the last expression of the Script. This model is an assembly of the embedded `layout` object and of an extrusion of its 1D skeleton, generated by Cartesian product times a 1D interval.

Script 13.3.2 (Layout as assembly)

```
DEF floor_layout =
   STRUCT:< dwelling, T:<1,2>:<41,34>, S:<1,2>:<-1,-1>, dwelling >;
DEF dwelling = STRUCT:< day, T:<1,2>:<17,4>:night1, T:2:16:night2 >;

(STRUCT ~ [EMBED:1, @1 * K:(QUOTE:<6>)]):floor_layout;
```

All the sizes and the measures of length are given as multiples of a modular unit $1M = 30cm$. In particular, in Script 13.3.3 a `room` alias is given for the predefined `PLaSM` operator `CUBOID`. Then the three `dwelling` components called `day`, `night1` and `night2` are respectively defined.

Script 13.3.3 (Layout components)

```
DEF room = CUBOID;
DEF PD = STRUCT ~ PDIFFERENCE;

DEF day =
   PD:< T:<1,2>:<3.5,-3>:balcony, kitchen, T:1:7:living, T:2:12:entrance >
WHERE
   kitchen = room:< 7, 12 >,
   living = room:< 10,16 >,
   balcony = room:< 7, 7 >,
   entrance = room:< 7, 4 >
END;

DEF night1 = STRUCT:< bed3, T:2:8, pas2, T:2:4, wc >
WHERE
   bed3 = room:< 7, 8 >,
   pas2 = room:< 3.5, 4 >,
   wc = room:< 3.5, 6 >
END;

DEF night2 = PD:< bed1, T:1:7:pas1, T:1:10.5:bath, T:1:7:bed2 >
WHERE
   bed1 = room:< 7, 14 >,
   pas1 = room:< 3.5, 4 >,
   bath = room:< 6.5, 6 >,
   bed2 = room:< 10, 14 >
END;
```

Note In Script 13.3.3 a definition is also given for the `PD` operator, discussed in this section, which may transform a weak HPC representation into a valid complete representation, by executing a sequence of difference operations, called the *progressive difference*, upon the ordered sequence of polyhedral complexes the weak representation is composed of. Notice that `PDIFFERENCE` is a primitive `PLaSM` operator which maps a *sequence* of polyhedral complexes into a *complex* of (possibly empty) polyhedral complexes.

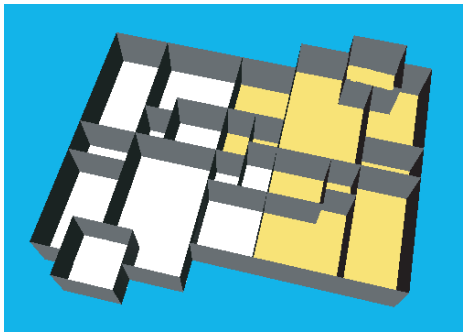


Figure 13.10 Assembly with `layout` embedded in 3D and the extrusion of its 1D skeleton

A full implementation of the `PD` operator, with

$$\texttt{PD} : \mathcal{R}^* \rightarrow \mathcal{R}$$

would actually require a recursive implementation, so that it would be possible to apply the operator to any pair of hierarchical polyhedral sequences of whatever complexity. Such an approach would require, in practical cases, an intolerable amount of computation. So, we decided for a basic non-recursive implementation, with usage under direct user control, as shown by Script 13.3.3.

Notice also that, from a user viewpoint, the given `PD` implementation must apply to sequences of polyhedral complexes, and not to sequences of polyhedral complexes and affine transformations, as the `STRUCT` operator is allowed to do.

13.4 Constructive Solid Geometry

A *Constructive Solid Geometry* (CSG) representation scheme [RV77, Req80] has the set of 3D r-sets as domain and a set of binary trees as the validity set. In the standard definition, a *CSG tree* is a binary tree where non-leaf nodes are either regularized set operations (see below) or affine transformations, and where leaf nodes are either primitive solids or implicit halfspaces.

This kind of representation has a very useful semantics. It often models exactly the process of shape creation in the designer intentions, as well as in the manufacturing process. Complex objects are often assembled/manufactured by applying Boolean set operation on subassemblies or on elementary parts.

In most current systems CSG trees are used to capture the design intention, and are not generally used as the primary internal representation of the solid. In `PLaSM` a CSG representation is naturally expressed as a language expression involving polyhedral arguments, Boolean operations and affine transformations, as shown in Example 13.4.1. Efficient algorithms exist for converting a boudary representation of a solid into a CSG representation [SV93]. A CSG representation can be *optimized* by using algebraic rewriting rules [SV91]. An extention of the Constructive Geometry to representing *non-regularized* objects is discussed in [RR91].

Example 13.4.1 (Boolean expression)
The solid `object` shown in Figures 13.11 and 13.12 is generated by Script 13.4.1.

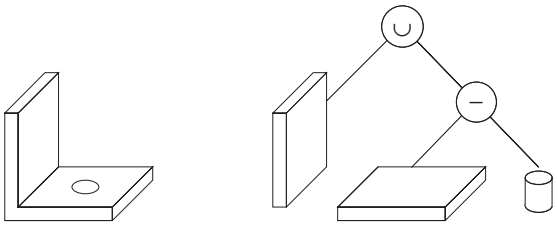


Figure 13.11 (a) Solid object (b) Constructive Solid Geometry (CSG) representation

Notice that the `PLaSM` denotation for *union* and *difference* set operations is "`+`" and "`-`", respectively. Set *intersection* and *symmetric difference* (`XOR`) are denoted as "`&`" and "`^`", respectively. No special symbol is available for `complement` operation. Since infix operators have the same precedence and are evaluated from left to right, it may often be necessary to enclose some sub-expression between parentheses, in order to correctly encode the desired CSG tree.

Script 13.4.1

```
DEF object = CUBOID:<2,10,10> +
   (CUBOID:<12,10,2> - T:<1,2>:<5,5>:(CYLINDER:<1,2>:24))
```

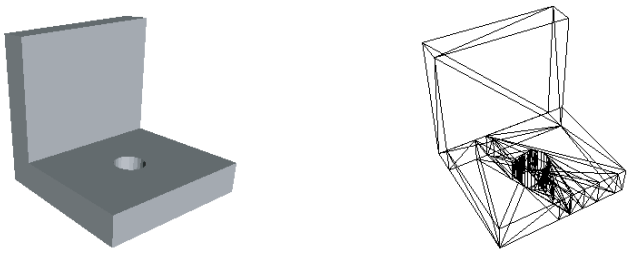


Figure 13.12 (a) Resulting solid object (b) Triangulation of the boundary of cells of the generated space partition

13.5 Boundary schemes

Boundary representations are not used by `PLaSM`. Anyway, they are quite well discussed in this section because they are used by most commercial solid modeling kernels and systems. Boundary representations, at least as the output generated by some kind of boundary evaluation algorithm, are also needed when graphically rendering any solid model, with the unique important exception of volume rendering techniques.

Definition A *boundary* representation scheme, often called *b-rep*, represents a d-dimensional solid model through some representation of its $(d-1)$–dimensional boundary [Bra75]. Such an approach is complete, i.e. unambiguous, because the

boundary of an orientable solid determines unambiguously its interior. In other words, two different solids cannot have the same boundary. A boundary representation is often defined inductively:

1. the boundary of a 3D solid is represented by a partition into bounded 2D pieces called *faces*;
2. each face is in turn represented by a partition of its 1D boundary into connected pieces called *edges*;
3. each edge is represented by its 0D boundary elements, i.e. by its extreme points, called *vertices*.

More in general, a boundary representation of a d-dimensional solid is given by the inductive subdivision of the elements of its $(d-1)$-dimensional boundary, thus generating a partition into cells of dimensions $d-1, d-2, \ldots, 1, 0$.

Topology and geometry In solid modeling literature the terms *topology* and *geometry* of a model usually refer to the set of incidence and adjacency relations between boundary elements, described below, and to the set of parametric equations of faces and edges, respectively.

When representing a linear d-polyhedron, all its geometry can be recovered by storing only the 0-dimensional elements, i.e. the vertices. Each p-dimensional boundary cell is in fact contained into the affine hull generated by $p+1$ affinely independent points.

For example, in 3D an edge is supported by the straight line generated by 2 vertices, and a face is supported by the plane generated by three non-collinear vertices. In the case of curved solids, the geometric information given by vertices is not sufficient, and the equations (either parametric or implicit) of the manifolds (say, curves and surfaces) supporting the boundary cells must be explicitly stored in computer's memory. A discussion of boundary representation with implicit algebraic surfaces can be found in [Hof89].

An important distinction concerns manifold and non-manifold b-reps. The kernel geometric libraries nowadays used in design environments with industrial strength are largely based on non-manifold boundary representations, whose study started with Weiler's thesis [Wei86].

Boundary representations as hierarchical descriptions As we have already seen, three main entities, i.e. *faces* (F), *edges* (E) and *vertices* (V), are normally used in b-reps of 3D solids. Other useful entities, shown in Figure 13.13b, are often adopted in practical b-reps. In particular, it is customary to introduce also the following entities:

1. the set B of connected components of the solid, called *bodies*;
2. the set S of connected components of the body boundary, called *shells*;
3. the set L of connected components of the face boundary, called *loops*.

It may be interesting to note that B, S and L are sets of geometric objects of dimension 3, 2 and 1, respectively. It may be also useful to remember that the arrow in a Bachman diagram stands for the *is-made-of* relationship. So, a solid is made of a subset of bodies, a body is made of a subset of shells, and so on.

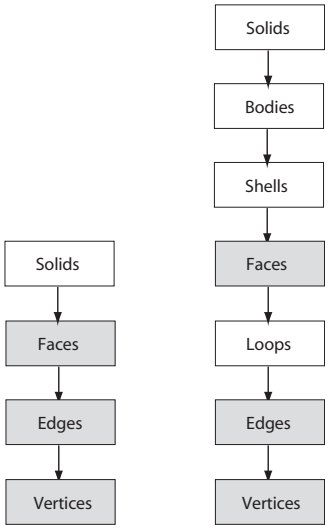


Figure 13.13 (a) Bachman diagram of fundamental 3D boundary elements (b) Extended hierarchy of 3D boundary elements

13.5.1 Adjacency and incidence relations

Let us consider the three main boundary entities F, E and V. Clearly there will exist $3 \times 3 = 9$ binary relationships between, displayed in Table 13.1a. The diagonal elements of Table 13.1a are called *adjacency* relations; the others are called *incidence* relations. So we speak, e.g., about the adjacency of faces and about the incidence of faces and vertices.

Every relation in Table 13.1a is a subset of the Cartesian set product of the involved entities. For example:

$$FE \subset F \times E$$

In the remainder of this chapter we use a lower case letter to denote the cardinality, i.e. the number of elements, of a set. So, we set

$$f = |F|, \quad e = |E| \quad \text{and} \quad v = |V|.$$

It is possible to show quite easily [Woo85] that every binary relation, except EE, contains $2e$ elements. Conversely, it is $|EE| \geq 4e$, where the equality holds for solids with exactly 3 edges incident on each vertex, like, e.g., a cube, a tetrahedron or a polyhedral approximation of a circular cylinder.

Example 13.5.1 (Cardinality of *VE*)
It is easy to verify, looking at Figure 13.14, that the pairs of incident vertices and edges in a double pyramid are exactly

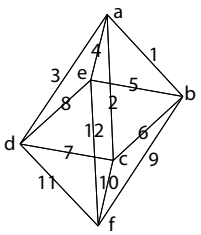
$$|VE| = 24 = 2e.$$

as shown by Table 13.1b.

Table 13.1 (a) Binary relations between boundary entities (b) Cardinalities of boundary relations

	F	E	V
F	FF	FE	FV
E	EF	EE	EV
V	VF	VE	VV

	F	E	V
F	$2e$	$2e$	$2e$
E	$2e$	$\geq 4e$	$2e$
V	$2e$	$2e$	$2e$



$$\begin{aligned} VE \;=\; & (\{a\} \times \{1,2,3,4\}) \cup \\ & (\{b\} \times \{1,5,6,9\}) \cup \\ & (\{c\} \times \{2,6,10,7\}) \cup \\ & (\{d\} \times \{3,7,8,11\}) \cup \\ & (\{e\} \times \{4,5,12,8\}) \cup \\ & (\{f\} \times \{9,10,11,12\}) \end{aligned}$$

Figure 13.14 Incidence relation VE between vertices and edges

13.5.2 Euler equation

A fundamental equation due to Leonhard Euler (1707–1783) holds between the numbers of faces, edges and vertices of 3D polyhedra which are *homeomorphic* to the sphere, i.e. that can be bicontinuously transformed to the sphere. Such solids, like e.g. the cube or the tetrahedron, are said to have topological genus $g = 0$, and also to be *topologically equivalent* to the sphere. In this case it is

$$v - e + f = 2 \tag{13.1}$$

Topological genus The number g of handles of a "sphere with handles" topologically equivalent to a solid is called the *topological genus* of the solid. Such a topological invariant is used to classify solids into equivalence classes. Solids in the same class, i.e. with same genus, are said to be topologically equivalent. This statement actually means that such solids may bicontinuously transform into each other, i.e. that each can be continuously deformed into the other, and vice versa.

Example 13.5.2 (Euler equation)
In the case of a 3D cube, we have $v = 8$, $e = 12$, $f = 6$, so that

$$8 - 12 + 6 = 2.$$

Analogously, for a 3D tetrahedron, or 3-simplex, we have $v = 4$, $e = 6$, $f = 4$, and hence we have, as expected

$$4 - 6 + 4 = 2.$$

This demonstrates that the cube and the tetrahedron are in the same equivalence class with respect to the topological genus.

Euler–Poincaré equation A strong generalization of the Euler equation (13.1) is due to Poincaré, French mathematician who started the study of algebraic topology at the end of the nineteenth and beginning of the twentieth centuries.

The so-called Euler–Poincaré equation holds between numbers of vertices, edges and faces of a connected polyhedral solid in 3D:

$$v - e + f = 2(s - g) + h, \tag{13.2}$$

where

1. s is the number of connected components of the boundary of the solid body. In other words $s = |S|$, where S is the set of shells of the body;
2. g is the topological *genus* of the body;
3. h is the number of *holes* (or internal loops) in body faces.

Example 13.5.3 (Euler–Poincaré equation)
We compute here the topological genus of the body in Figure 13.15, generated by subtracting two orthogonal parallelepipeds from a central cube. The g genus is computed by counting the numbers of vertices, edges, faces and so on, and by substituting such values into equation (13.2), thus getting the unknown value of g. In particular, by substituting $v = 32$, $e = 48$, $f = 15$, $s = 1$ e $h = 4$ in (13.2) we get

$$32 - 48 + 16 = 2(1 - g) + 4$$

and hence $g = 3$. The solid is topologically equivalent to the sphere with 3 handles, as shown by Figure 13.15.

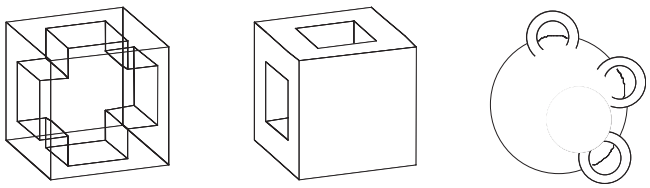


Figure 13.15 The Euler–Poincaré equation $v - e + f = 2(s - g) + h$ is specialized into $32 - 48 + 16 = 2(1 - g) + 4$, so that $g = 3$

13.5.3 Edge-based schemes

Boundary representation schemes can be classified in two main classes, that we call *edge-based* and *face-based*. The first class contains most of known representations, and is discussed in this section. In particular, we illustrate here the so-called *minimal* boundary representation, as well as Baumgart's *winged-edge* [Bau72] and Mäntylä's *half-edge* [Män88] representations.

Minimal b-rep

It has been shown [Woo85] that the data structures which may implement every b-rep of 3D models are connected subgraphs of the complete oriented graph, shown in Figure 13.16a, defined by the sets F, E, V of boundary entities.

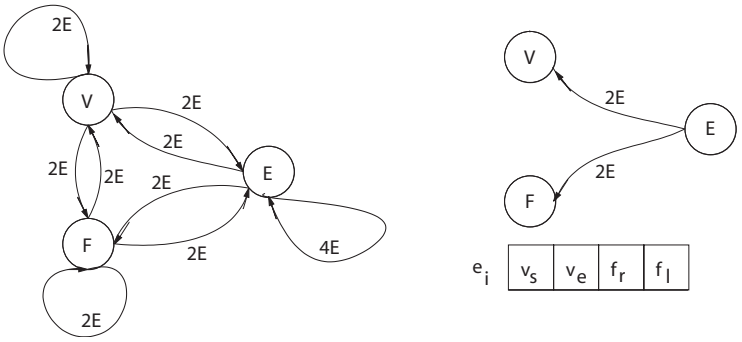


Figure 13.16 (a) Complete graph of topological relations between boundary entities (b) Minimal b-rep, given by two relations in normal form (Codd)

The smallest boundary representation, that stores two incidence relations in first normal form [Dat97], i.e. where each tuple has constant length, is given by the pair (EV, EF).

The representation, shown in Figure 13.16b can be given as a table indexed on edges, where the tuple indexed by e_i contains the two vertices and the two faces incident the edge e_i. This *minimal* boundary representation has size $4e$, because each edge-tuple contains 4 data items.

The minimal b-rep can only be used with manifold solids, where it is guaranteed that exactly two faces meet on each edge. The number of faces incident on an edge is, more in general, even for non-manifold solids. Clearly, the domain of minimal b-rep scheme is restricted to the subset of manifold models.

Winged-edge representation

The so-called *winged-edge* representation, developed by B. G. Baumgart at the AI Laboratory of MIT in early 1970s [Bau72], is probably the more well-known and historically important boundary representation.

In this case we have a relational representation in normal form, based on two tables of indices and on a primary table indexed on edges. In particular, the tuple of the primary table indexed by e_j contains 8 data items, including:

1. references to 2 incident vertices,
2. references to 2 incident faces and
3. references to 4 adjacent edges which also meet on the two adjacent faces.

The winged representation WE is shown in Figure 13.17. The WE b-rep contains the minimal one, since it contains the two relations EV and EF. Also it contains a subset of EE with cardinality $4e$.

The tuple of the primary WE table, indexed by the e_j edge, accommodates respectively the following fields:

$$(v_i, v_f, f_r, f_l, e_{ir}, e_{il}, e_{fr}, e_{fl})$$

where v_i, v_f denote the references to the *initial* and *final* vertex of the edge e_j, f_r, f_l are references to *right* and *left* incident faces on e_j, and $e_{ir}, e_{il}, e_{fr}, e_{fl}$ are references to:

1. the edge incident on the initial vertex and on the right face of e_j,
2. the edge incident on initial vertex and left face,
3. the edge incident on final vertex and right face,
4. the edge incident on final vertex and left face.

The name of this data structure probably derives from the picture of the tuple, that looks like the shape of a butterfly.

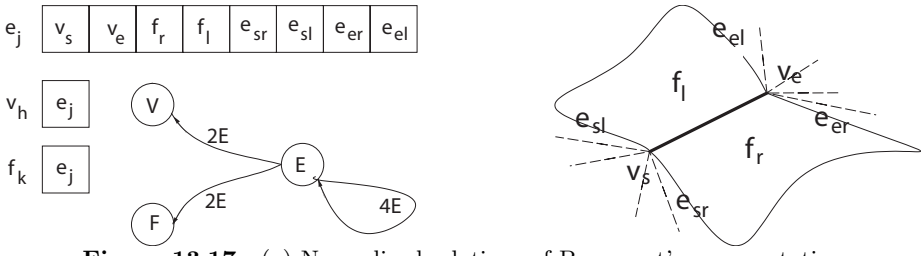


Figure 13.17 (a) Normalized relations of Baumgart's representation (b) Incidence/adjacency relations stored in the primary tuple

Optimality of *winged-edge* b-reps The WE representation is very interesting, because it allows an answer to every topological query in time proportional to the size of the query output. This is generally impossible with the minimal b-rep previously discussed.

For example, in order to return the subset of vertices on the boundary of a face, it is possible to access, through the table of face indices, the tuple of one of incident edges, and then to navigate the only tuples of edges incident the face, thus accumulating, e.g., the first vertex of each tuple, until the starting tuple is accessed again. This approach results in a number of storage accesses equal to the number of boundary edges (and vertices). Conversely, an access to every tuple of the table would be needed when using the minimal b-rep.

Multiply connected faces Like the minimal b-rep, the winged-edge scheme may only represent manifold solids with simply connected faces, i.e. with single-loop face boundaries. Two well-known variations of the WE scheme, allowing representation of multiply connected faces, require either an explicit representation of the *face-loop* relation [Bra79] or the introduction of so-called *bridge-edges* [YT85] to reduce multiply connected face to simply connected ones, as shown by Figure 13.18. Bridge-edges are easily recognized in the primary WE table, because they have $f_r = f_l$.

Half-edge representation

The *half-edge* [Män86] representation by Martï Mäntylä is described by the Bachman diagram shown in Figure 13.19. In this representation the `Edge` entity is needed to implement the FF relation. In particular, each instance of the `Edge` entity is linked to two instances of the `HalfEdge` entity. Clearly enough, a `Loop` entity is a linked list of `HalfEdge` instances.

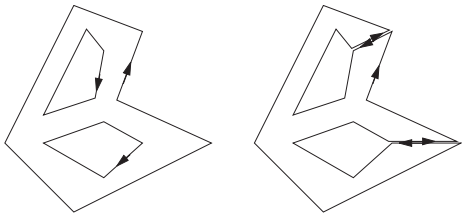


Figure 13.18 (a) Multiply connected face (b) Simply connected face with inserted *bridge-edges*

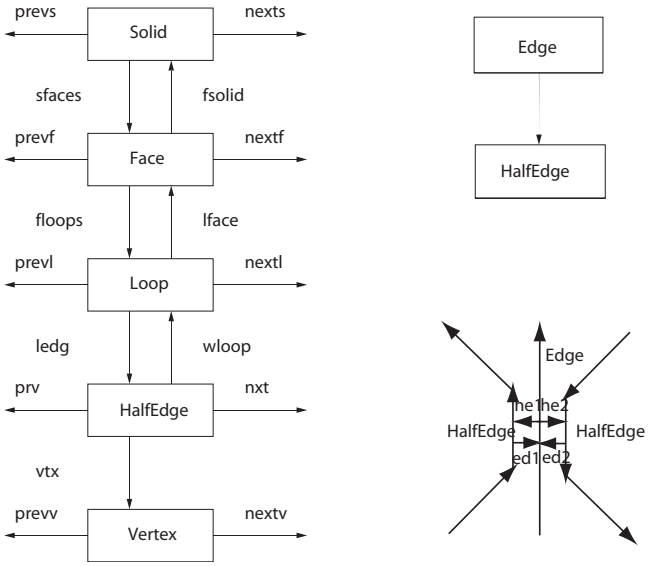


Figure 13.19 (a) Mäntylä's *Half-Edge* representation (b) Diagram of topological relations between edges and halfEdges

A typical characteristic of both this representation and of several other b-reps, mainly when using very complex data structures, is the extensive use of so-called *Euler operators* [Män88], used as middle-level access operations to the representation, in particular when implementing very complex algorithms, like Boolean operations. Such middle-level operations are used to step-wise modify the representation in a consistent way, i.e. by always satisfying the Euler–Poincaré equation.

13.5.4 Facet-based schemes

Facet-based schemes are b-rep schemes where the main role is assigned to the 2D elements of the partition of the object's boundary. We may classify here both the *face-adjacency hypergraph* by L. De Floriani and B. Falcidieno [ADF85], and *the winged-triangle* b-rep [PRS89] by A. Paoluzzi *et al.*, both discussed in this section. We also discuss in this section, despite its name, the *quad-edge* representation [GS85] by L. Guibas and J. Stolfi, because it fully exploits the duality relation between graphs embedded on surfaces.

Face Adjacency Graph

An interesting facet-based boundary representation was proposed by S. Ansaldi, L. De Floriani and B. Falcidieno [ADF85], and called *Face Adjacency Graph*, making use of some graph theoretical concepts described below, and in particular of the duality between embedded graphs. Their approach may be found very useful when describing and classifying geometric features in object design and manufacturing.

Graph embedding The boundary of a 3D polyhedron of genus 0 can be represented as a *planar graph*, i.e. as a graph embedded — or "drawn" — on either the plane or the sphere, with an intersection of edges located only at vertices. More generally, the boundary of a solid of g genus can be embedded as a planar graph on a surface of g genus.

For example, the boundary of a 3D cube can be drawn on the plane as shown in Figure 13.20. Such *embedding* of the boundary graph contains 8 vertices, 12 edges and 6 faces. A *face* is here defined as a connected region of the plane partition generated when subtracting vertices and edges (as point sets) from the embedding surface. One of such regions, called the *external face*, is unbounded. Clearly, an external face does not exist when the graph is drawn on the sphere.

Duality between planar graphs Let us consider an *abstract planar graph* as a triplet $G = (V, E, F)$, where V is the set of vertices, E is the set of edges and F is the set of faces. It can be seen that *cycles* of edges constitute a group (the *cycle group*) with respect to an operation of cycle addition. All the cycles can be generated by a subset of independent generators, which constitute a basis for the group. There exists a bijective mapping between the set of *internal* faces of the graph embedding and a basis of cycles.

Given an abstract planar graph $G = (V, E, F)$ it is always possible to uniquely associate another planar graph, called the *dual planar graph*, $G' = (V', E', F')$ defined by three bijective maps f, g and h, that associate vertices of primal with faces of dual, faces of primal with vertices of dual, and edges of primal with edges of dual, respectively:

$$f : V \to F', \qquad g : F \to V', \qquad h : E \to E'.$$

Clearly, the duality relation is idempotent, i.e. the dual of the dual is the primal.

The two edges (v_i, v_j) and $h(v_i, v_j) = (v'_a, v'_b)$, associated by duality, must satisfy a mutual intersection constraint. In particular, both must be obtained by intersection of (the cycles of edges associated to) dual faces of the extreme vertices of the other. In formal terms:

$$(v_i, v_j) = g^{-1}(v'_a) \cap g^{-1}(v'_b),$$

and

$$(v'_a, v'_b) = f(v_i) \cap f(v_j),$$

so that

$$h(g^{-1}(v'_a) \cap g^{-1}(v'_b)) = f(v_i) \cap f(v_j).$$

Example 13.5.4 (Boundary graphs)
In Figure 13.20 we show two plane embeddings of the primal and dual graphs associated to the boundary of a 3D cube.

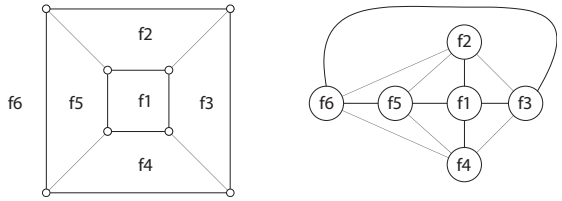


Figure 13.20 Primal and dual boundary graph of the cube

Face-Adjacency Graph The *Face-Adjacency Graph* (FAG) representation [ADF85] by Ansaldi, De Floriani and Falcidieno, represents a 3D solid through the non-directed dual of the boundary graph, where the object faces are used as vertices and the pairs of adjacent faces are used as arcs. The connectivity properties of this graph are representative of the topological characteristics of the solid boundary. Variations of this structure are often used to discover and reason with the geometric features (pockets, slots, lumps, etc.) of the object.

Quad-edge representation

A particularly elegant data structure for polyhedra is the *quad-edge* data structure, invented by Guibas and Stolfi [GS85]. It is limited to closed manifolds, where edges are always shared by two faces. In the quad-edge data structure, there are records for vertices, edges, and faces, but the *Edge* record play the leading role. The edges store complete topological information; all of the topological information stored by the faces and vertices is redundant with respect to information in the edges.

Given a *directed* Edge, it is possible to find the immediately adjacent vertices, faces, and edges, and the "symmetric" edge that points in the opposite direction. In particular, for each edge, there are pointers to *next* edge around right face, with same right face, *next* edge around left face, with same left face, *next* edge around origin, with same origin, *next* edge around dest, with same destination. Similar pointers are stored for *previous* elements.

Winged-triangle representation

The so-called *winged-triangle* representation WT, by Paoluzzi, Ramella and Santarelli [PRS89], is a b-rep based on vertices and triangles (0- and 2-simplices) of a boundary triangulation, i.e. of a simplicial complex associated to the object boundary. In particular, the WT representation is a table in first normal form [Dat97] where each tuple, indexed by the t_j triangle, contains:

1. 3 references to incident vertices;
2. 3 references to adjacent triangles.

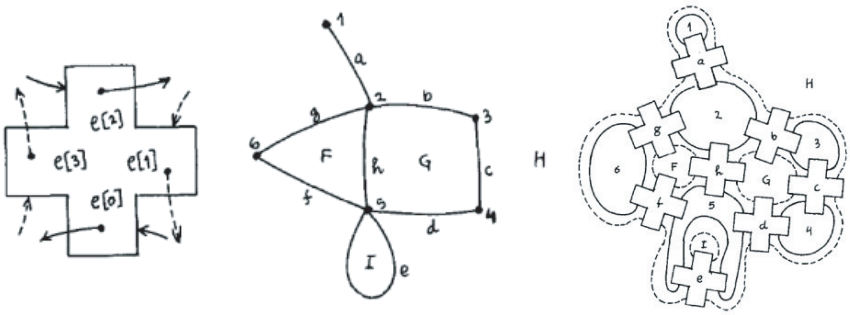


Figure 13.21 (a) `Edge` record showing `next` links (b) A subdivision of the sphere (c) The data structure for the subdivision

This scheme is characterized by the design choice of making no use of Euler operators. This choice is allowed by the extreme simplicity of the data structure representing the triangulation of the boundary, and by the implementation of the Boolean algorithms, discussed in the following subsections, where the consistency of the boundary simplicial complex is easily maintained.

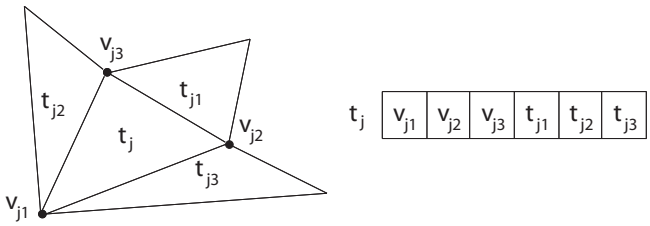


Figure 13.22 (a) Triangle t_j, with triplets of adjacent triangles and of incident vertices (b) Tuple associated to t_j

Domain of the scheme The space of mathematical models represented by the WT scheme is quite extensive. It coincides with the set of linear regular 3-polyhedra which are possibly:

1. unconnected;
2. unbounded (but with bounded boundary);
3. non-manifold ;
4. multishell;
5. with multiply connected faces.

Range of the scheme A representation in this scheme is a quadruple $WT(\Sigma) = (V, T, \nu, \tau)$, where V and T are respectively the sets of 0-simplices and 2-simplices of a boundary complex Σ, and where

$$\begin{aligned} \nu &: T \to V \times V \times V, \\ \tau &: T \to T \times T \times T. \end{aligned}$$

Hence, for the boundary triangle t_j, as displayed by the tuple in Figure 13.22, we have:

$$\begin{aligned} \nu(t_j) &= (v_{j_1}, v_{j_2}, v_{j_3}), \\ \tau(t_j) &= (t_{j_1}, t_{j_2}, t_{j_3}). \end{aligned}$$

Validity set A $WT(\Sigma)$ representation is *valid* if and only if the associated boundary simplicial complex Σ is:

1. finite;
2. bounded;
3. closed (i.e. without boundary);
4. orientable.

The aims of such requirements are quite simple to explain. An infinite complex is intractable, because it cannot be stored in a computer memory. Also, an unbounded boundary complex is beyond the concept of a "solid" in the common experience. An open (i.e. with boundary) boundary complex does not satisfy the fundamental property which says that, for every body B, "the boundary of the boundary is empty":

$$\partial\partial B = \emptyset.$$

Finally, a boundary complex must be orientable in order to separate the interior from the exterior space.

When the above properties are satisfied, the support space $[\Sigma]$, i.e. the point set union of simplices in Σ, coincides with the boundary of some orientable 3-polyhedron. It may be also useful to note that such representation scheme is *complete*, but *not unique*. As a matter of fact, the boundary of a solid may support several different triangulations.

Example 13.5.5 (Closed surfaces)
In Figure 13.23 we show the solid resulting from union of two translated parallelepiped solids. We remark that in every *valid* representation the boundary complex is closed, i.e. that every boundary triangle is always adjacent to *three* other triangles.

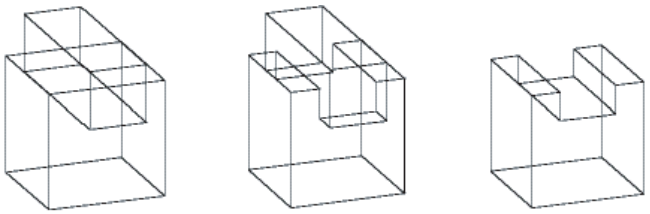


Figure 13.23 (a) Aggregation of solids A and B (b) $A \cup B$ (c) $A - B$

Example 13.5.6 (Open surfaces)
What was noted above is not true for the winged-triangle representation of open (say with-boundary) polyhedral surfaces. Let consider the surface in Figure 13.24. Notice that the triangles which are adjacent to the boundary curves have only *two* adjacent

triangles. Anyway, a WT representation, which is not valid as a solid representation, can be usefully exploited by adopting a special symbol, say either -1 or $\perp$, to read as "undefined", to denote the empty adjacencies in triangle tuples.

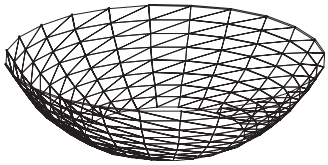


Figure 13.24 Open triangulated surface

Non-manifolds as pseudo-manifolds Theoretically, the domain of the scheme should only embrace manifold solids. But, in order to make the domain closed with respect to Boolean operations, it is useful to represent non-manifold models in the same scheme. To give a pseudo-manifold WT representation of a non-manifold solid is very easy and "natural". Let us consider for this purpose Figure 13.25. A *pseudo-manifold* complex can be defined as a simplicial complex with some *duplicated* simplices, and having a non-manifold support space.

No problems arise with duplicated *non-manifold edges*, if we choose from the various possible representations the one with the minimum number of shells. In this case the connection across the non-manifold edge is correctly preserved. Slightly more complex is the case of *non-manifold vertices*, where some auxiliary data structure, storing the shell-vertex incidence, is needed to track correctly the shell connection across such vertices.

Anyway, no management of this very special case is needed either for Boolean operations or for integration algorithms, as discussed in the following sections.

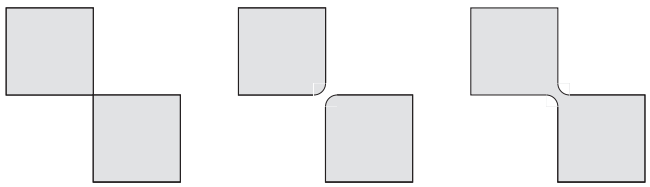


Figure 13.25 Pseudo-manifold representations of a non-manifold model

Storage space of WT representation

For each surface triangulated by a simplicial complex the following useful properties hold. They have direct implications on the storage of the WT representation in the computer memory.

Theorem (Storage space) The following relations between the numbers of triangles, edges and vertices of a boundary triangulation hold:

1. (Triangles-edges)
   The number $t = |T|$ of triangles and the number $e = |E|$ of edges satisfy the equation

$$t = \frac{2}{3}e. \tag{13.3}$$

2. (Triangles-vertices)
   In a polyhedron with a number $v = |V|$ of vertices, a number $s = |S|$ of boundary shells and g genus, a minimal set of triangles T which triangulates the boundary has size

$$t = 2v - 4(s - g). \tag{13.4}$$

3. (Storage respect to triangulation edges)
   The storage space *mem(WT)* needed by a triangulated surface, where $e = |E|$ is the number of edges of the triangulation, is:

$$mem(WT) = 4e. \tag{13.5}$$

4. (Storage respect to polyhedron edges)
   For a 3D polyhedron, with $\hat{e}$ original edges, say non induced by the triangulation, $\hat{f}$ polygonal faces and h holes in the faces (rings), we have:

$$mem(WT) = 12(\hat{e} - \hat{f} + h). \tag{13.6}$$

5. (Lower and upper bounds)
   The storage space *mem(WT)*, for any 3D polyhedron, is included between $\frac{1}{2}$ and $\frac{3}{2}$ of the storage required by *WE* representation:

$$\frac{1}{2}mem(WE) \leq mem(WT) \leq \frac{3}{2}mem(WE). \tag{13.7}$$

Proof For point 1, since each triangle t_i is incident to three vertices, all the subsets t_iV of the incidence relation $TV \subset T \times V$ have the same number 3 of elements, so that $|t_iV| = 3$ for each $t_i \in T$, and hence

$$|TV| = \sum_{t_i \in T} |t_iV| = \sum_{t_i \in T} 3 = 3t.$$

Also we know that the incidence relation VF has cardinality $2e$. For the WT representation the boundary faces coincide with the triangles of the boundary complex, so that:

$$|VT| = 2e.$$

By the symmetry of incidence relation, we have $|TV| = |VT|$, so that equation (13.3) is proved. For point 2, the Euler–Poincaré equation [Poi53] (see Section 13.5.2) can be specialized for a triangulation as

$$v - e + t = 2(s - g),$$

since holes in the faces (rings) are not allowed ($h = 0$), so that

$$v - \frac{3}{2}t + t = 2(s - g)$$

and hence equation (13.4) is proved. For point 3, we can write for the primary table of the representation:

$$mem(WT) = 6t = 6\left(\frac{2}{3}e\right)$$

For point 4, from Theorem (13.4) we have

$$t = 2v - 4(s - g), \tag{13.8}$$

and, from Euler–Poincaré equation

$$v = 2(s - g) + h + \hat{e} - \hat{f}. \tag{13.9}$$

So, by substituting the v expression (13.9) into the t expression (13.8) we get

$$t = 2(\hat{e} - \hat{f} + h)$$

and hence the statement is proved. The above properties can be combined, to get lower and upper bounds for the storage space of the WT representation, as stated by point 5.

Storage bounds Such results are quite interesting, because they give an upper bound for the WT storage, depending either on the number of triangles or on the number of vertices. Notice in particular that the number of boundary triangles, and hence the size of the WT primary table, is $O(v)$ and that, in particular, is bounded by $2v$. The lower bound of point 5 is obtained when all the original faces are triangular. This happens, e.g., when polyhedrally approximating solids with double curvature external surfaces. In this case the WT representation is space-optimal, since it requires exactly the same space than the minimal b-rep discussed in Section 13.5.3. Conversely, the upper bound is approached when the original faces of the solid have an asyntotically increasing number of edges. In normal usage with linear polyhedra, the average size of a WT representation is about 6 times the number of original edges of the object.

Example 13.5.7
A minimal triangulation of cube boundary is shown in Figure 13.26. The number of edges of such triangulation is $e = 12 + 6$, given by the original edges of cube, plus one edge for each face. From equation (13.3) we get, for the number of triangles, $t = \frac{2}{3}18 = 12$.

Boolean operations on b-reps

Several approaches to the computation of Boolean operators using a boundary representation can be found in the solid modeling literature (see, e.g. [Bra79, HHK88, LTH86, Män88]). The very simple Boolean algorithms defined with the WT b-rep, that do not require the use of Euler operators, can be found in [PRS89].

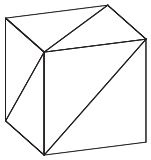


Figure 13.26 *Winged-triangle* representation of cube boundary

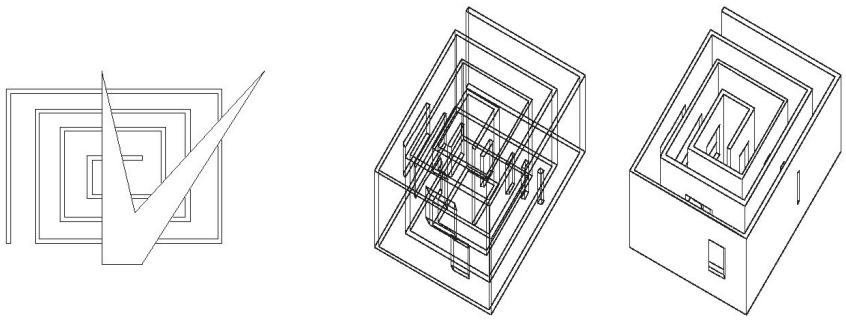


Figure 13.27 (a) 2D polygons (b) Boolean difference between extruded solids

Regularized operations Set operations of union, intersection and difference, as defined in solid modelers, are usually "regularized". Such modified operations are in fact closed on the set of regular solids. In other words, the combination of regular solids by means of a *regularized* set operation, always returns a regular solid. This property is generally not true for standard set operations. In Figure 13.28 we show that a standard set operation between regular arguments may produce a non-regular result.

A *regularized set operation* is defined as the closure of the interior of the result of the standard set operation. Let us denote the operation as op, with op $\in \{\cap, \cup, -\}$. So, a regularized operation, denoted as op*, is defined as

$$A \text{ op}^* \ B = \mathbf{k}\,\mathbf{i}\,(A \text{ op } B),$$

where $\mathbf{k}$ and $\mathbf{i}$ denote the topological operations of *closure* and *interior*, respectively. It may be useful to remember that $\mathbf{k}(S) = S \cup \partial S$.

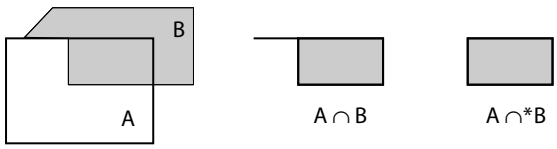


Figure 13.28 Standard set intersection and regularized set intersection

Boolean algebra Several boundary representation schemes have as domain the set of *bounded* manifolds (or non-manifolds). So, they do not allow for a complement operator. Consequently, they must devise appropriate algorithms for each set operation, i.e. for union, intersection and difference.

Conversely, the WT representation scheme allows for a complement, and hence properly implements a Boolean algebra over the set of 3D polyhedra [PRS89], by explicitly enforcing the closure of the validity set of the scheme under the complement of representations. In this way, e.g., *intersection* and *difference* are reduced to the combination of *complement* and *union*, according to De Morgan's theorems of Boolean algebras. In particular, we have:

$$\begin{aligned} A \cap B &= \neg(\neg A \cup \neg B), \\ A - B &= \neg(\neg A \cup B). \end{aligned}$$

Let us note that only two algorithms are needed in this case, to implement the basic operations of complement and union, respectively.

13.5.5 Mass and inertia properties

The evaluation of area, volume, centroid and moments of inertia of rigid homogeneous solids frequently arises in a large number of engineering applications, both in Computer-Aided Design and in Robotics. Hence, quadrature formulae for multiple integrals have always been of great interest in computer applications.

Many papers on integration methods were related to solid modeling, but as Lee and Requicha pointed out in [LR82] most computational studies in multiple integration often deal with calculations over very simple domains, like a cube or a sphere, while the integrating function $f(\mathbf{p})$ is very complicated. Conversely, in most of the engineering applications, the opposite problem usually arises.

Definitions Mass and inertia properties of solid objects are defined as volume integrals of low-degree monomial fields $f(x, y, z)$, the integration being done over the space portion occupied by the object under consideration. If $B \subset \mathbb{E}^3$ is the set of body points, then the *mass* M, the *first moments* M_x, M_y, M_z, the *second moments* M_{xx}, M_{yy}, M_{zz} and the *products of inertia* M_{yz}, M_{xz}, M_{xy} are defined as

$$\iiint_B f(x, y, z)\, dM = \iiint_B f(x, y, z)\rho(x, y, z)\, dV,$$

where $\rho(x, y, z)$ is the local *density* of the body, and the scalar field $f(x, y, z)$ is respectively equal to 1 (mass); x, y and z (first moments); x^2, y^2 and z^2 (second moments); yz, xz and xy (products of inertia).

Centroid The *centroid* $G = (G_x, G_y, G_z)$ of a body, also known as "center of mass", is defined by the ratios $G_x = \frac{M_x}{M}$, $G_y = \frac{M_y}{M}$ and $G_z = \frac{M_z}{M}$ of the first moments to the body mass. In *homogeneous* solids, where the density is constant, the centroid position does not depend on the density, but only on the geometry of the body.

Moments of inertia The *moments of inertia* with respect to some axis are defined as the volume integrals of the squared orthogonal distances from the axis. Moments

of inertia with respect to a coordinate axis are hence given, respectively, by

$$\begin{aligned} M_{r_x^2} &= M_{yy} + M_{zz}, \\ M_{r_y^2} &= M_{xx} + M_{zz}, \\ M_{r_z^2} &= M_{xx} + M_{yy}. \end{aligned}$$

The moment of inertia M_{r^2} of body B with respect to any axis r is computed by first translating r to the origin, then by rotating the translated axis r' to some coordinate axis, say z, and finally by computing $M_{r_z^2}$, i.e.:

$$M_{r^2}(B) = M_{r_z^2}(\boldsymbol{R}_{r' \to z} \boldsymbol{T}_{r \to r'} B).$$

Timmer-Stern's method

The volume integration problem can be stated as follows: *evaluate the volume integral*

$$\iiint_B f(\boldsymbol{p})\, dV, \qquad \boldsymbol{p} \in B \tag{13.10}$$

where $f(\boldsymbol{p})$ is a scalar-valued field over $B \subset \mathbb{E}^3$.

The integration method discussed in the following section may be considered as a specialization of Timmer and Stern's general method [TS80, Mor85], consisting of the transformation of a volume integral into a surface integral and then into a parametric line integral.

Algorithm Timmer-Stern's integration procedure can be summarized as follows:

1. Search for a vector field $\boldsymbol{\Phi}$ such that:
$$\iiint_B f(\boldsymbol{p})\, dV = \iiint_B \nabla \cdot \boldsymbol{\Phi}\, dV. \tag{13.11}$$
2. Use the divergence theorem to transform the right-hand term of equation (13.11) into an integral on the closed boundary S of the B integration domain
$$= \iint_S \boldsymbol{\Phi} \cdot \boldsymbol{n}\, dS. \tag{13.12}$$
3. Use the property of domain-additivity of surface integrals over a partition of the boundary surface S into a set $\{S_i\}$ of faces such that $\cup_i S_i = S$, and $S_i \cap S_j = \emptyset$ for each $i \neq j$:
$$= \sum_i \iint_{S_i} \boldsymbol{\Phi} \cdot \boldsymbol{n}\, dS_i. \tag{13.13}$$
4. Transform each surface integral in a double integral in the parametric domain S_{uv}^i of the face S_i of the scalar field $\boldsymbol{\Psi}(u, v)$:
$$= \sum_i \iint_{S_{uv}^i} \boldsymbol{\Psi}(u, v)\, du\, dv, \tag{13.14}$$

where

$$\boldsymbol{\Psi}(u,v) = \boldsymbol{\Phi}(x(u,v), y(u,v), z(u,v)) \cdot \boldsymbol{n}(x(u,v), y(u,v), z(u,v))|\boldsymbol{p}_u \times \boldsymbol{p}_v|.$$

5. Use again the divergence theorem in 2D to transform these last integrals into line integrals on the closed and simple boundary curves C_i of domains S^i_{uv}. In other words, search for a vector field $\boldsymbol{\chi} = (\chi_1, \chi_2)$ such that:

$$= \sum_i \iint_{S^i_{uv}} \boldsymbol{\Psi}(u,v)\, du\, dv \tag{13.15}$$

$$= \sum_i \iint_{S^i_{uv}} \nabla \cdot \boldsymbol{\chi}\, du\, dv \tag{13.16}$$

$$= \sum_i \oint_{C^i_{uv}} \boldsymbol{\chi} \cdot \hat{\boldsymbol{n}}\, ds. \tag{13.17}$$

6. Then use the domain-additivity of curve integral to integrate over the set of curves associated with the boundary edges C^{ij}_{uv} in parametric domain uv of each boundary face:

$$= \sum_i \sum_j \int_{C^{ii}_{uv}} \boldsymbol{\chi} \cdot \hat{\boldsymbol{n}}\, ds. \tag{13.18}$$

7. Finally, transform each term into a single integral in the parametric domain C^{ij}_t of the trimming edge C^{ii}_{uv}:

$$= \sum_i \sum_j \int_{C^{ii}_t} \boldsymbol{\chi}(t) \cdot \hat{\boldsymbol{n}}(t)\, s'(t)\, dt. \tag{13.19}$$

The Timmer's and Stern's method is specialized in the next subsection for when (a) the field $f(\boldsymbol{p})$ is a polynomial, (b) the integration domain B is a 3D polyhedron and (c) a triangulation of ∂B is available.

Integration of polynomials over polyhedral domains

Here we summarize from [CP90] an exact and symbolic solution both to the surface and volume integration of polynomials, by using a triangulation of the volume boundary. The evaluation of surface and volume integrals is achieved by transforming them into line integrals over the boundary of every 2-simplex of a domain triangulation. A different approach to integration, using a decomposition into volume elements induced by a boundary triangulation is given in [lLK84] where a closed formula for volume integration over polyhedral volumes, by decomposing the solid into a set of solid tetrahedra, but such a method cannot be used for surface integrations.

Problem statement A finite method [CP90] to compute double and triplet integrals of monomials over linear regular polyhedra in $\mathbb{R}^3$ is discussed. In particular, this method enables practical formulae for the exact evaluation of integrals to be achieved:

$$II_S \equiv \iint_S f(\mathbf{p})\, dS, \qquad III_P \equiv \iiint_P f(\mathbf{p})\, dV, \tag{13.20}$$

where S, and P are linear and regular 2- or 3-polyhedra in $\mathbb{R}^3$, dS and dV are the differential surface and the differential volume. The integrating function is a trivariate polynomial

$$f(\mathbf{p}) = \sum_{\alpha=0}^{n} \sum_{\beta=0}^{m} \sum_{\gamma=0}^{p} a_{\alpha\beta\gamma} x^\alpha y^\beta z^\gamma,$$

where α, β, γ are non-negative integers.

Since the extension to $f(\mathbf{p})$ is straightforwardly given by the linearity of integral operator, we may focus on the calculation of integrals of monomials:

$$II_S^{\alpha\beta\gamma} \equiv \int\!\!\int_S x^\alpha y^\beta z^\gamma \, dS, \qquad III_P^{\alpha\beta\gamma} \equiv \int\!\!\int\!\!\int_P x^\alpha y^\beta z^\gamma \, dV. \tag{13.21}$$

Algorithm preview Surface integrals are computed as a summation of integrals over a triangulation of the surface. Any triangle is mapped into the unit triangle in the 2-space of parameters, where integrals of monomials become particularly simple. Then formulae for integrals over polyhedral volumes are given. They are easily derived by transforming volume integrals in surface integrals. It is possible to show that such integrals are computable in polynomial time, and that inertia moments are computable in $O(E)$ time, E being the number of edges of the solid model of the integration domain.

A very important feature of the integration formulae presented here is that they can also be used with a partial model of a polyhedron, consisting of the collection of its face loops. Loops are oriented counter-clockwise if external, clockwise if internal to another loop. Such a model, without explicit storage of face adjacencies, is very frequently adopted in Computer Graphics.

In this case it is sufficient to consider any $n+1$-sided (also unconnected or multiply connected) face as topological sum of $n-1$ oriented triangles t_i, with vertices $\langle v_0, v_i, v_{i+1} \rangle$, where $1 \leq i \leq n-1$. In applying formulae (13.31) or (13.34) to such a set of triangles, any edge that does not belong to the original polygon will be computed twice, in the two opposite directions. These contributions to the whole integral will mutually cancel each other out, as they correspond to pairs of line integrals evaluated along opposite paths.

Surface integration We call *structure product* the integral of a monomial over a simplicial complex. Exact formulae for structure products over n-sided polygons in 2-space, the unit triangle in 2-space, and an arbitrary triangle in 3-space, are derived in the following. Structure products are a generalization of the usual products and moments of inertia, that can be obtained from (13.21) by assuming $\alpha + \beta + \gamma \leq 2$.

Polygon integrals A structure product over a polygon π in the plane xy is

$$II_\pi^{\alpha\beta} = \int\!\!\int_\pi x^\alpha y^\beta \, dS, \qquad \alpha, \beta \geq 0, integers. \tag{13.22}$$

Such integrals can be exactly expressed, when π is a polygon with n oriented edges, as:

$$II_{\pi}^{\alpha\beta} = \frac{1}{\alpha+1} \sum_{i=1}^{n} \sum_{h=0}^{\alpha+1} \binom{\alpha+1}{h} x_i^{\alpha+1-h} X_i^h \sum_{k=0}^{\beta} \frac{\binom{\beta}{k}}{h+k+1} y_i^{\beta-k} Y_i^{k+1} \tag{13.23}$$

where $\mathbf{p}_i = (x_i, y_i)$, $X_i = x_{i+1} - x_i$, $Y_i = y_{i+1} - y_i$ and $\mathbf{p}_{n+1} = \mathbf{p}_1$. The derivation of the formula (13.23) is based on the application of Green's theorem and on Newton's expression for binomial powers.

Unit triangle integrals The general formula (13.23) can be specialized for the unit triangle $\tau' = \langle \mathbf{w}_o, \mathbf{w}_a, \mathbf{w}_b \rangle$, with vertices

$$\mathbf{w}_o = (0,0), \qquad \mathbf{w}_a = (1,0), \qquad \mathbf{w}_b = (0,1), \tag{13.24}$$

getting a very simplified expression. With some algebraic manipulations, we obtain[1]

$$II^{\alpha\beta} = \frac{1}{\alpha+1} \sum_{h=0}^{\alpha+1} \binom{\alpha+1}{h} \frac{(-1)^h}{h+\beta+1}, \tag{13.25}$$

which reduces, for $\alpha = \beta = 0$, to the area of the triangle (13.24): $II^{00} = 1/2$.

Triangle integrals In the following we derive the general expression for structure products evaluated on an arbitrary triangle $\tau = \langle \mathbf{v}_o, \mathbf{v}_a, \mathbf{v}_b \rangle$ of the 3-space xyz, defined by $\mathbf{v}_o = (x_o, y_o, z_o)$ and by the vectors $\mathbf{a} = \mathbf{v}_a - \mathbf{v}_o$ and $\mathbf{b} = \mathbf{v}_b - \mathbf{v}_o$. The parametric equation of its embedding plane is:

$$\mathbf{p} = \mathbf{v}_o + u\,\mathbf{a} + v\,\mathbf{b}, \tag{13.26}$$

where the area element is

$$d\tau = |J|\,du\,dv = \left| \frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v} \right| du\,dv = |\mathbf{a} \times \mathbf{b}|\,du\,dv. \tag{13.27}$$

A structure product over a triangle τ in 3-space can be transformed by a coordinates transformation, as follows:

$$II_{\tau}^{\alpha\beta\gamma} = \iint_{\tau} x^{\alpha} y^{\beta} z^{\gamma}\,d\tau = |\mathbf{a} \times \mathbf{b}| \iint_{\tau'} x^{\alpha}(u,v) y^{\beta}(u,v) z^{\gamma}(u,v)\,du\,dv, \tag{13.28}$$

where τ' is the uv domain that corresponds to τ under the coordinate transformation (13.26). In this case we have (the proof is given in [CP90]):

$$\begin{aligned} II_{\tau}^{\alpha\beta\gamma} \quad = \quad & |\mathbf{a} \times \mathbf{b}| \sum_{h=0}^{\alpha} \binom{\alpha}{h} x_o^{\alpha-h} \sum_{k=0}^{\beta} \binom{\beta}{k} y_o^{\beta-k} \sum_{m=0}^{\gamma} \binom{\gamma}{m} z_o^{\gamma-m} \cdot \\ & \cdot \sum_{i=0}^{h} \binom{h}{i} a_x^{h-i} b_x^i \sum_{j=0}^{k} \binom{k}{j} a_y^{k-j} b_y^j \sum_{l=0}^{m} \binom{m}{l} a_z^{m-l} b_z^l\, II^{\mu\nu}, \end{aligned} \tag{13.29}$$

[1] $II_{\pi}^{\alpha\beta}$ is substituted, when referred to the unit triangle, by the symbol $II^{\alpha\beta}$.

where $\mu = (h+k+m) - (i+j+l), \nu = (i+j+l)$, and $II^{\mu\nu}$ is a structure product over the triangle (13.24), as given by formula (13.25). Of course the area of a triangle τ is:

$$II_{\tau}^{000} = \iint_{\tau} d\tau = |\mathbf{a} \times \mathbf{b}|\, II^{00} = \frac{|\mathbf{a} \times \mathbf{b}|}{2}. \tag{13.30}$$

Surface integrals In conclusion, a structure product over a polyhedral surface S, open or closed, is a summation of structure products (13.29) over the 2-simplices of a triangulation K_2 of S:

$$II_{S}^{\alpha\beta\gamma} = \iint_{S} x^{\alpha} y^{\beta} z^{\gamma}\, dS = \sum_{\tau \in K_2} II_{\tau}^{\alpha\beta\gamma}. \tag{13.31}$$

Volume integration Let P be a three-dimensional polyhedron bounded by a polyhedral surface ∂P. The regularity of the integration domain and the continuity of the integrating function enable us to apply the divergence theorem, which can be briefly summarized, for a vector field $\mathbf{F} = \mathbf{F}(\mathbf{p})$ as:

$$\iiint_{P} \nabla \cdot \mathbf{F}\, dx\, dy\, dz = \iint_{\partial P} \mathbf{F} \cdot \mathbf{n}\, dS = \sum_{\tau \in K_2} \iint_{\tau} \mathbf{F} \cdot \mathbf{n}_{\tau}\, d\tau, \tag{13.32}$$

where $\mathbf{n}$ is the outward vector normal to the surface portion dS, and hence $\mathbf{n}_{\tau} = \mathbf{a} \times \mathbf{b}/|\mathbf{a} \times \mathbf{b}|$.

As the function $x^{\alpha} y^{\beta} z^{\gamma}$ equates the divergence of the vector field $\mathbf{F} = (x^{\alpha+1} y^{\beta} z^{\gamma}/\alpha + 1, 0, 0)$, an expression for $III_{P}^{\alpha\beta\gamma}$ is easily derived, which depends only on the 1-simplices of a triangulation of the domain boundary and on the structure products over its 2-simplices.

As a matter of fact, we have:

$$\begin{aligned} III_{P}^{\alpha\beta\gamma} &= \iiint_{P} x^{\alpha} y^{\beta} z^{\gamma}\, dx\, dy\, dz \\ &= \iiint_{P} \frac{\partial}{\partial x} \left(\frac{1}{\alpha+1} x^{\alpha+1} y^{\beta} z^{\gamma} \right) dx\, dy\, dz \\ &= \frac{1}{\alpha+1} \sum_{\tau' \in K'_2} (\mathbf{a} \times \mathbf{b})_x \iint_{\tau'} x^{\alpha+1} y^{\beta} z^{\gamma}\, du\, dv. \end{aligned} \tag{13.33}$$

Taking into account equations (13.27) and (13.28), we can substitute the integral in the previous equation, getting finally:

$$III_{P}^{\alpha\beta\gamma} = \frac{1}{\alpha+1} \sum_{\tau \in K_2} \left[\frac{(\mathbf{a} \times \mathbf{b})_x}{|\mathbf{a} \times \mathbf{b}|} \right]_{\tau} II_{\tau}^{\alpha+1,\beta,\gamma} \tag{13.34}$$

where the surface integrals are evaluated by using the formula (13.29).

Computation of inertia has linear complexity Surface and volume integrals over linear polyhedra are computable in linear time. In particular, *surface and volume integrals of a monomial $x^\alpha y^\beta z^\gamma$ over a linear 2-or 3-polyhedron are computable in $O(\alpha^3\beta^2\gamma^2 E)$ time, E being the number of edges of the polyhedron.*

In fact for the surface and volume integrals it is very easy to see, from inspection of the given equations, that both integrals may be evaluated in $O(\alpha^3\beta^2\gamma^2 T)$ time, T being the number of triangles of a minimal triangulation of the domain. It is easy to show that the relation $T = 2E - 2F < 2E$ holds between the number T of triangles of a minimal triangulation of a polyhedron boundary and the numbers E and F of its original edges and faces respectively. When all triangle faces are triangular, the relation reduces to $T = \frac{2}{3}E$.

This property is very important for Computer-Aided Design and Robotics applications: it demonstrates that the inertia tensor of a linear polyhedral solid is easy to compute. It directly implies that *the inertia tensor of a linear polyhedron is computable in $O(E)$ time.* As a matter of fact the elements of the inertia matrix of a homogeneous object B, namely its *mass* M, *first moments* M_x,M_y,M_z, *products of inertia* M_{xy}, M_{yz}, M_{zx}, and *second moments* M_{xx}, M_{yy}, M_{zz}, can be all expressed as

$$\rho \int_B x^\alpha y^\beta z^\gamma \, dV, \tag{13.35}$$

where ρ is the constant density, and where $0 \leq \alpha + \beta + \gamma \leq 2$. Being α, β, γ bounded, the assertion follows from the previous claim.

13.6 Examples

13.6.1 Umbrella modeling (4): solid parts

The 1D subcomplex with handle and rods defined in our last umbrella version with curve segments and surface patches (see Section 12.6.1) is refined with 3D solid parts in Script 13.6.1. In particular, the new handle is generated as a 3-variate mapping of the boundary `Dom` of a translated parallelopided `D1 * D2 * D3` in parametric space, where `D1` is a partition in 8 parts of the interval $[\pi, 2\pi]$, `D2` is a partition in 1 parts of the interval $[0, r/3]$, where r is the handle radius, and `D3 = D2`.

Figure 13.29 Some perspectives of the model generated by `Umbrella:<10,80>`, and `Umbrella:<10,30>`, where `80` and `30` (degrees) are the opening angles.

Finally, the curved tiny rods of the umbrella are generated from their initial wire frame definition, by using the primitive `BezierStripe` operator. Such a function will

Script 13.6.1 (Umbrella modeling (4))

```
DEF Handle (h::Isreal) = T:<1,2>:<Radius*7/6,-:Radius/6>:
   (MAP:[S2*COS ∼ S1, S3, S2*SIN ∼ S1]:dom )
WHERE
   D1 = (T:1:PI ∼ S:1:PI ∼ Domain):8,
   D2 = (S:1:(1/3*Radius) ∼ Domain):1,
   D3 = D2,
   Dom = @2:(T:2:Radius:(D1 * D2 * D3)),
   Radius = h/18
END;

DEF Axis (h::IsReal) = (@2 ∼ STRUCT):<
   Handle2, T:3:(h/10), MetalRing, Rod, T:3:(2*AB), Tip >
WHERE
   Handle2 = T:<1,2>:<-1/6*Radius,-1/6*Radius>:
      (CUBOID:<1/3*Radius,1/3*Radius,h/10>),
   MetalRing = CYLINDER:<h/50, 0.5*h/10>: 12,
   Rod = CYLINDER:<h/90, 2*AB>: 12,
   Tip = Handle2,
   Radius = h/18,
   AB = h*4/10
END;

DEF Rod (len::IsReal) = T:<1,2>:<-2*a,-1/2*a>:((@2 ∼ CUBOID):<2*a,a,len>)
   WHERE a = len/100 END;
```

generate a plane surface stripe depending on the 3 control points of a quadratic Bezier curve. The 3D solid rods are thus generated by the polyhedral product of the surface stripe times a 1D segment of size `len/100`.

Script 13.6.2 (Umbrella modeling (4))

```
DEF RodCurve (len,beta,n::IsReal) =
   (R:<2,3>:(PI/2) ∼ T:1:(2*len/100) ∼ @2):
      ((T:1:(-2*len/(100 - 5)) ∼ BezierStripe):
         <<<0,0>,<0,len/2>,<len/2 * sin:beta,len>>,2*len/100,n>
         * QUOTE:<len/100>);

Umbrella:<10,80>
```

Some projections of the refined model, for different values of the opening angle, are given in Figure 13.29.