13.472J/1.128J/2.158J/16.940J
COMPUTATIONAL GEOMETRY

Lectures 14 and 15

Prof. N. M. Patrikalakis

Massachusetts Institute of Technology
Cambridge, MA 02139-4307, USA

Contents

Constructive Solid Geometry (CSG)

14.1 Introduction

CSG is a method in which an object is constructed from the standard primitives using regularized Boolean operations. The model is represented in the data structure as a CSG tree, see Figure 14.1, whose terminal nodes are primitives and non-terminal nodes are Boolean operators (intersection, union and difference). Primitives are sized, positioned and oriented first.

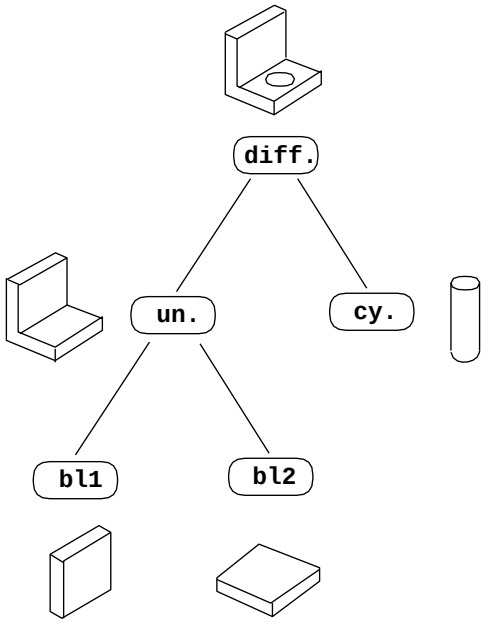


Figure 14.1: An example of CSG tree

14.2 Primitives of CSG

Typical primitives are the rectangular box, the circular cylinder of finite height, the sphere, the cone of finite height and the torus. Figure 14.2 shows two collections of CSG primitives. These primitives may be defined as intersections of halfspaces (defined by algebraic inequalities of the form $f(x, y, z) \geq 0$).

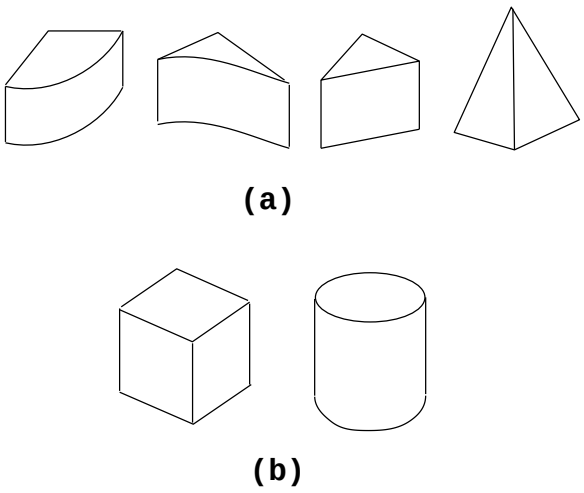


Figure 14.2: Two collections of CSG primitives.

14.3 Boolean operators

Definition: A set S is regular if $S = (int(S))^{cl}$, where int means the interior, and cl means closure. Taking the interior of S means constructing a subset of S where all points in the subset have an ε-neighborhood homeomorphic to a ball. Closure means adding the limit points (or boundary) to the interior points to produce a new set.

Let A, B denote regular sets in R^3, then

$$\begin{aligned} b(A \cup B) &= (bA \cap cB) \cup (bB \cap cA) \\ b(A \cap B) &= (bA \cap iB) \cup (bB \cap iA) \\ b(A - B) &= (bA \cap cB) \cup (bB \cap iA) \end{aligned} \tag{14.1}$$

where bX is the boundary of the set X, iX is its interior and cX is its complement. See Figure 14.3. Obviously, Boolean operators require the execution of surface intersections, and postprocessing to evaluate the correct boundary of the resulting set using (14.1).

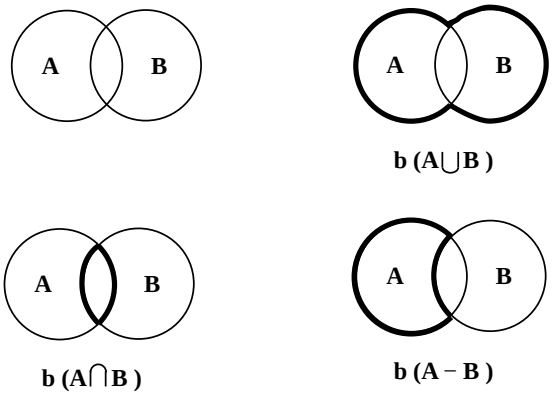


Figure 14.3: Boolean operators

14.3.1 Regularized Boolean operators

Manifold objects are not closed under Boolean operations. Regularized set operations make it so, see Figure 14.4 for example.

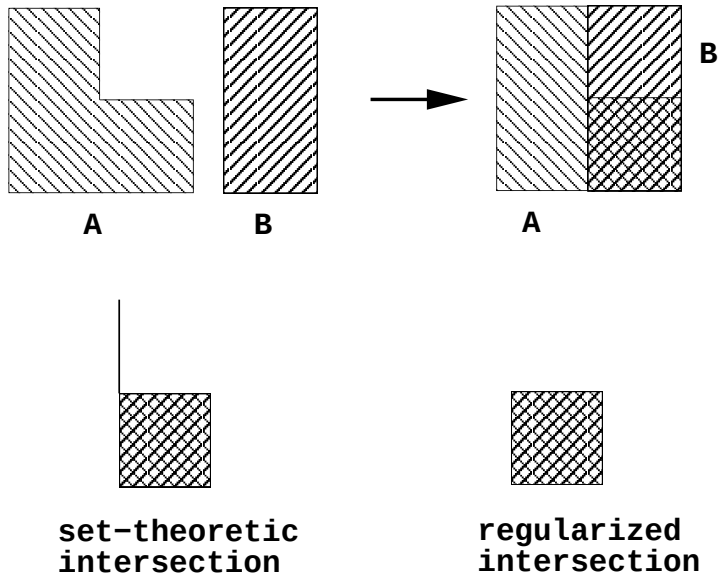


Figure 14.4: Regularized set operations.

Let $\cap^*$ be a regularized set operation, and $C^* = A \cap^* B$, then to compute C^*, we proceed conceptually as follows:

1. $C = A \cap B$
2. $C_i = \text{interior } C$
3. $C^* = \text{closure } C_i$

Figure 14.5 shows the procedure with an example.

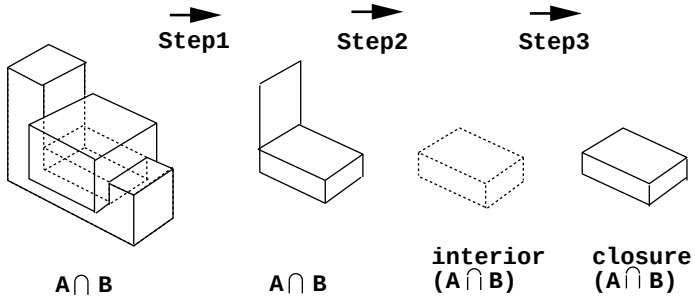


Figure 14.5: Procedure for regularized intersection.

14.4 Set membership classification

Given a CSG model M and an object X in the scene, the function $Classify(M, X)$ will indicate if (see Figure 14.6):

- X is in M, or
- X is on M, or
- X is out of M.

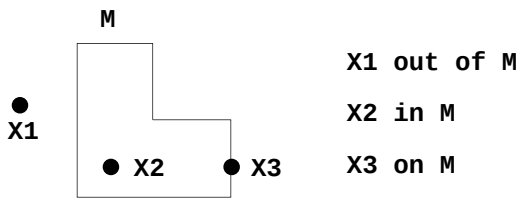


Figure 14.6: Membership classification.

Divide-and-conquer paradigm

CLASSIFY(M,X)

```
1    if   M is a primitive
2         then PRIM-CLASSIFY(M,X)
3         else COMBINE(CLASSIFY(left-subtree(M),X),
                       CLASSIFY(right-subtree(M),X), operation of M)
```

For the example shown in Figure 14.7,

$$\begin{aligned} \text{CLASSIFY(M,X)} &= \text{COMBINE(PRIM-CLASSIFY(A,X), PRIM-CLASSIFY(B,X),} \\ &\qquad \text{intersection)} \\ &= \text{COMBINE(X-in-A, X-in-B, intersection)} \\ &= \text{X-in-M} \end{aligned}$$

Similarly, classifying a line with respect a CSG model $M = A \bigcap B$ typically involves intersections of the line with A and B, and then checking to see if the intersections which are line segments have a common segment, see Figure 14.8.

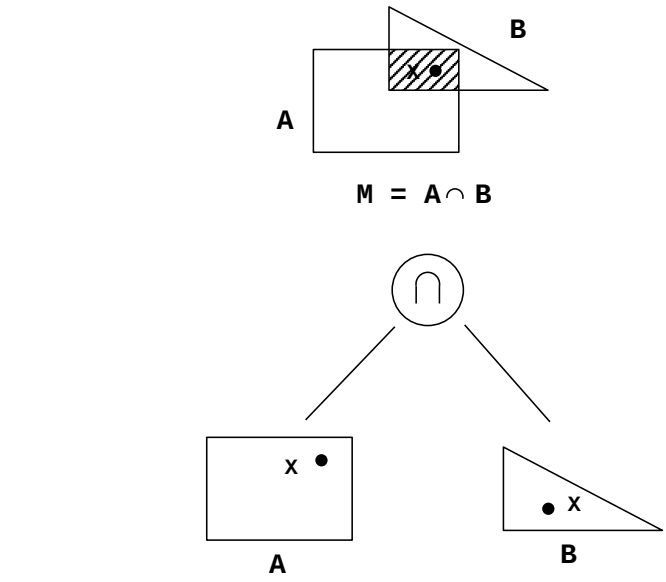


Figure 14.7: Example of membership classification for point

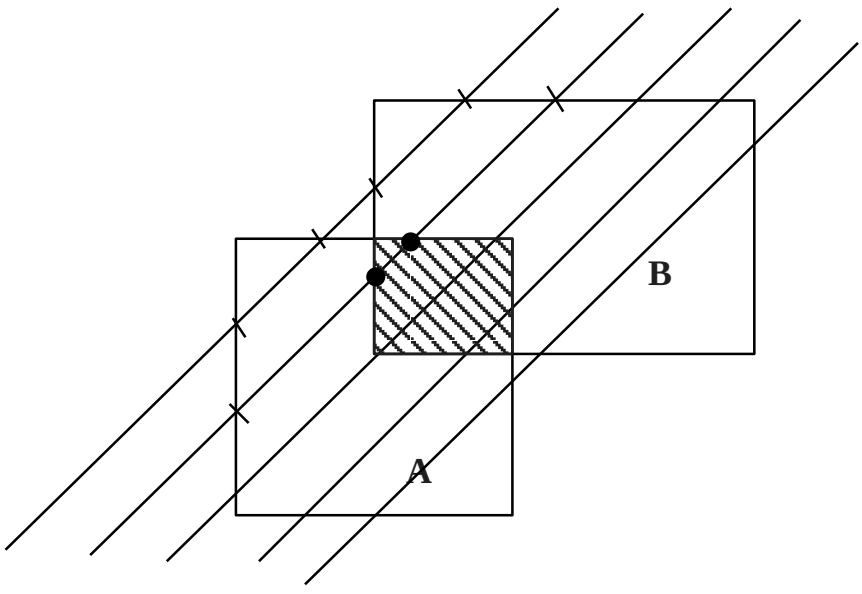


Figure 14.8: Example of membership classification for line

14.5 Properties of CSG

Here are the basic properties of CSG models

- *Advantages*:
  - **validity:** CSG model is always valid;
  - **conciseness:** CSG tree is in principle concise;
  - **computational ease:** primitives are easy to handle;
  - **unambiguity:** every CSG tree unambiguously models a rigid solid (maybe more than one).
- *Disadvantages*:
  - **non-uniqueness:** a solid could have more than one CSG representation,
  - **limit on primitives:** free-form surfaces are excluded, and primitives are typically bounded by a number of simple low order algebraic surfaces.
  - **redundancy of CSG tree:** it may have redundant primitives that do not contribute to final solid.
  - **no explicit boundary information:** CSG tree needs to be evaluated (eg. rendered to evaluate surface, such as ray trace it, etc.)

Boundary Representation

14.6 Two-manifold B-rep

- **Definition:**
  Boundary models describe solids in terms of their bounding entities, such as faces, loops, edges and vertices.

- **Examples:**
  polygon $\Longrightarrow$ bounding edges
  solid volume $\Longrightarrow$ bounding faces

- The bounding relations shown in Figure 14.9 are:
  cube $\Longrightarrow$ 6 faces (squares)
  square $\Longrightarrow$ 4 edges (line segment)
  line segment $\Longrightarrow$ 2 vertices (points in 3-D space)

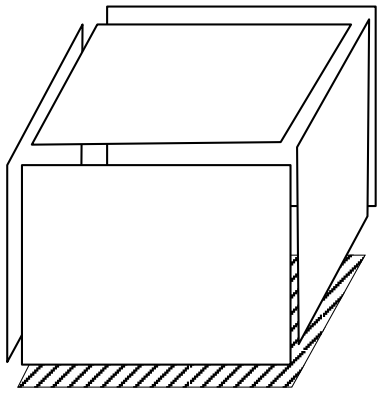


Figure 14.9: Boundary of a cube.

A two-manifold B-rep is an explicit representation of a single object (volume) by its boundary which is assumed to be:

- Compact (closed and bounded),
- Orientable, and
- Two-manifold.

Definitions

1. A surface is *closed* if it is bounded and has no boundary (eg. a plane is unbounded, a patch is bounded but has boundary and a sphere is closed as its surface is bounded and has no boundary).

2. A surface is *orientable* if it is two sided (not like Möbius strip and Klein bottle).

3. A two-manifold surface is topologically *two dimensional connected* surface where each point on the surface has a neighborhood which is topologically equivalent to an open disk.

    - Orientable and closed surfaces are required to distinguish inside and outside.
    - Counter examples of two-manifold surfaces are shown in Figure 14.11.

4. *Topological equivalence (Homeomorphism)*: A homeomorphism is a one-to-one topological transformation which is continuous and has a continuous inverse (intuitively, elastic deformations which preserve adjacency properties). Or, more strictly, if there is a one-to-one correspondence between the points of a surface and those of another surface, so that the topological properties of any figure in one of the surfaces are shared by its image in the other, the two surfaces are said to be homeomorphic to each other, and the mapping from one surface to the other established by the one-to-one correspondence is a homeomorphism.

5. *Open disk*, portion of a 2-D space (surface) which is within a circle of positive radius, excluding circle. See Figure 14.12.

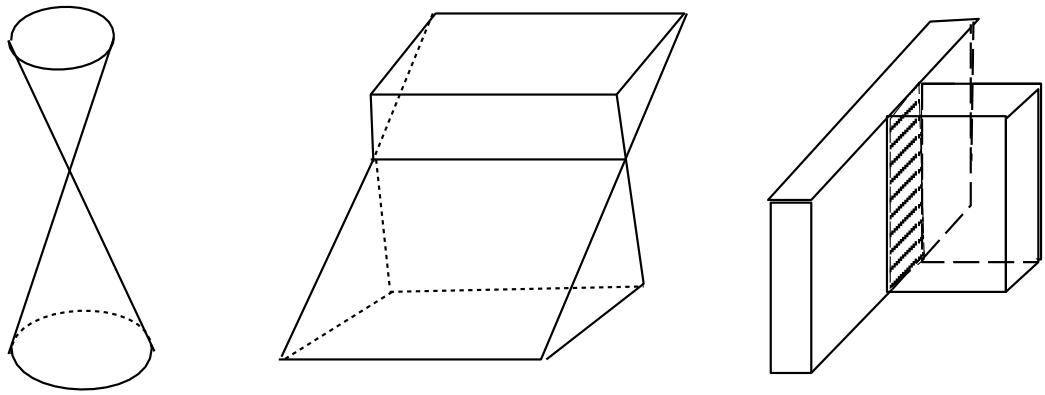


Figure 14.11: Non two-manifold surfaces.

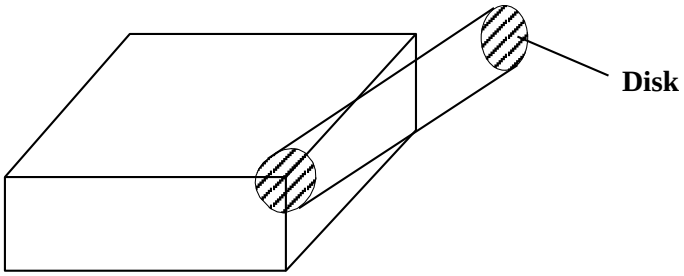


Figure 14.12: Neighborhood of point on two-manifold object is a disk.

14.6.1 Information contained in a B-rep

There are two different kinds of information necessary in a B-rep, geometrical information and topological information, see Figure 14.13. Geometrical information provides a complete specification of the object and topological information is an abstraction, which provides a "fuzzy" definition of the object correct within "genus" specification (number of through holes) and subdivision into faces together with their adjacency.

A geometrical entity S_1 is **incident** to another geometrical entity S_2, if S_1 has dimensionality one higher than S_2, and S_2 is a bounding entity of S_1. Two geometrical entities S_1 and S_2 are **adjacent**, if they have the same dimensionality and share a common bounding entity.

Geometrical information

Complete geometry can be considered to represent all information about the geometric shape of an object including where it lies in space and the precise location of all aspects of its various elements:

- points,
- curves: eg. line segments, circular arcs, B-spline, and Bézier curves, NURBS curves and
- surfaces: e.g. bounded planes, quadrics, B-spline and Bézier surfaces, NURBS patches,

i.e. geometry deals with the relationships between surfaces, curves, points and the coordinate space.

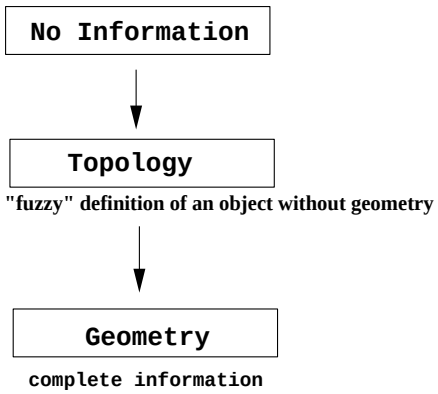


Figure 14.13: B-rep model idealization

Topological information

Topology is an abstraction, and it contains the incidence information of various elements. It is incomplete information which can "ideally" be derived from the complete geometric specification. Topology deals with the adjacency relationships between corresponding entities, namely physical proximity or order of a group of topological elements of one type (such as vertices, edges or faces) around some other specific single topological elements. Adjacency relationships are illustrated in Figure 14.14.

The typical topological elements are:

- **Vertex**: A unique point in space. A vertex lies in one or more faces.
- **Edge**: A finite, non-self-intersecting curve bounded by two not necessarily distinct vertices. An edge lies on the boundaries of exactly two faces of a two-manifold object.
- **Loop**: An ordered alternating sequence of vertices and edges defining a unique point or directed non-self-intersecting, closed space curve.
- **Face**: A finite connected non-self-intersecting oriented piece of a surface bounded by one or more loops. A loop lies in a single face and forms a bound of the face. The number of faces is equal to the number of peripheral loops.
- **Shell**: The collection of consistently oriented faces forming the boundary of a single, connected, closed volume (region).
- **Region**: Unique, identifiable volume in space. There is one region with infinite extent, all others are finite.
- **Model**: 3-D modeling space, consisting of one or more regions.

In a two-manifold representation there is a one-to-one correspondence between a region and its bounding shell. Therefore it is sufficient to have just one of them represented explicitly. In general regions are not represented explicitly in most existing B-rep data structures.

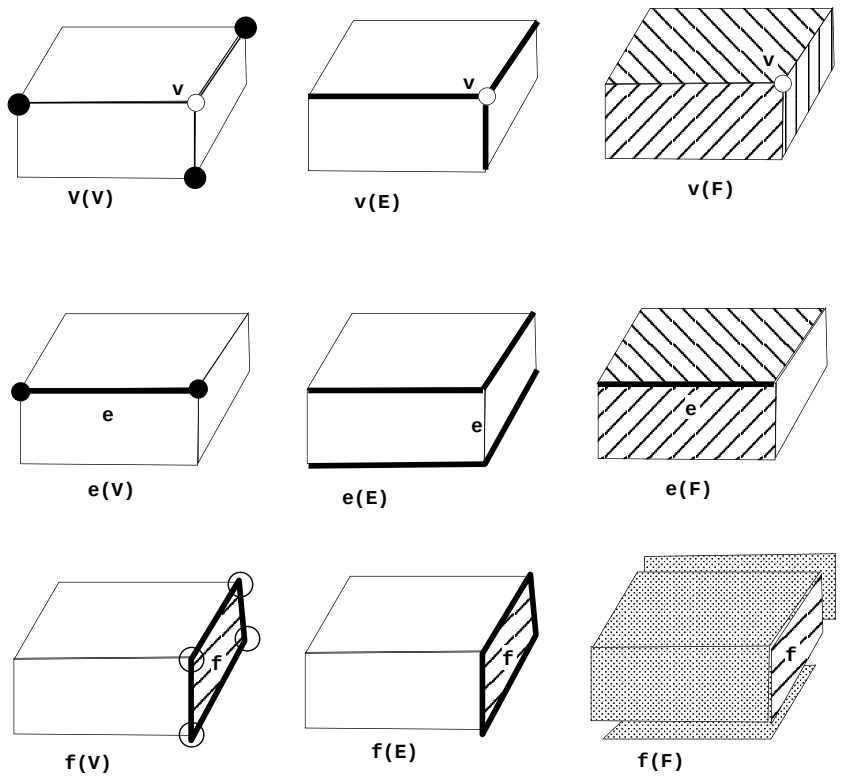


Figure 14.14: Adjacency relationships.

14.6.2 Characteristics of domain for two-manifold solid object representations

- Surfaces: compact, orientable, two-manifold embedded in the 3-D Euclidean space.
- Faces: no self-intersection is permitted but they are allowed to intersect with each other at edges or vertices.

Remarks:

- Adjacency topology explicitly carries all surface intersection information through adjacency information.
- No non two-manifold situations are allowed. Therefore, in a traversal of edges bounding faces, every edge is traversed exactly twice.
- Orientability guarantees that the interior of a solid volume is distinguishable from its exterior (See Figure 14.15). The orientability guarantees that the interior of a solid volume is distinguishable from its exterior.

14.6.3 Euler-Poincaré equation

This equation is a relationship between topological elements for a single two-manifold shell:

$$V - E + F - L_i = 2(1 - G) \tag{14.2}$$

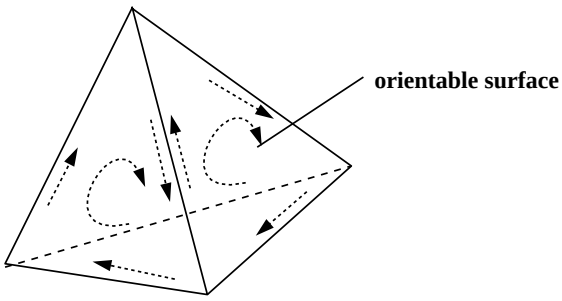


Figure 14.15: Orientable surface.

where,
V: Number of vertices.
E: Number of edges.
F: Number of faces.
L_i: Number of interior loops.
G: Genus, the number of closed paths on a surface which do not separate the surface into more than one region. Or, genus is the number of handles to be added to a sphere to make it homeomorphic to the object.

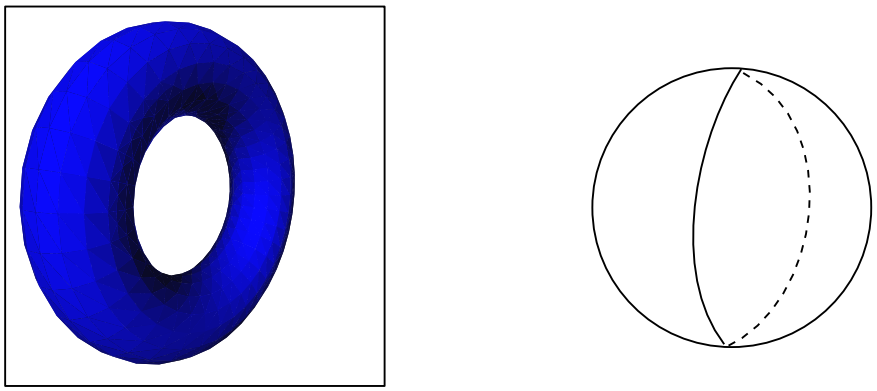


Figure 14.16: Torus and sphere.

Another form of the Euler equation is

$$V - E + 2F - L = 2(1 - G) \tag{14.3}$$

(using the relations $L = L_p + L_i$ and $L_p = F$, L_p: number of peripheral loops)
For multiple shelled objects (objects with cavities), the Euler equation becomes

$$V - E + F - L_i = 2(S - G) \tag{14.4}$$

S: number of shells.
Euler equation is a necessary but not sufficient condition for validity of a B-rep.

Conditions for topological validity

1. $V, E, F, L_i, S, G \geq 0$,
2. If $V = E = F = L_i = 0 \Longrightarrow S = 0, G = 0$,
3. If $S \geq 0 \Longrightarrow V \geq S$ and $F \geq S$, and
4. For a shell to exist, there must be at least one vertex and one face on the shell.

Examples

- *Example 1*: A tetrahedron (see Figure 14.17)

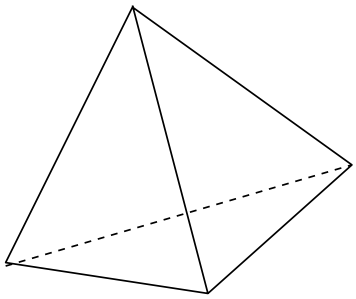


Figure 14.17: Tetrahedron.

$V = 4, E = 6, F = 4, L_i = G = 0$
$4 - 6 + 4 - 0 = 2 = 2(1 - 0)$

- *Example 2*: A cube with or without a hole (see Figure 14.18)

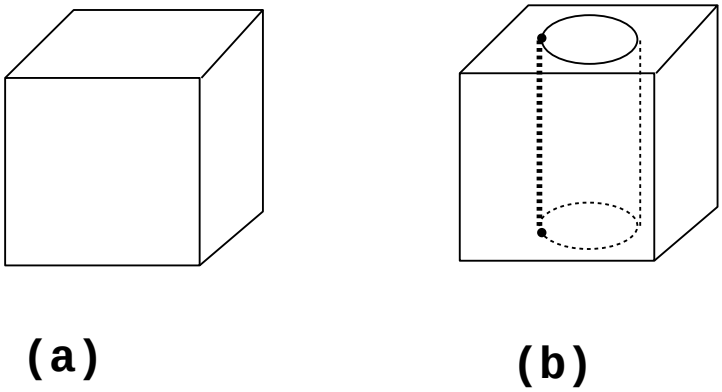


Figure 14.18: (a) Cube; (b) Cube with a hole.

(a) Without hole $V = 8, E = 12, F = 6, L_i = G = 0$
$8 - 12 + 6 - 0 = 2$

(b) With a hole $V = 10, E = 15, F = 7, L_i = 2, G = 1$
$10 - 15 + 7 - 2 = 2(1 - 1) = 0$

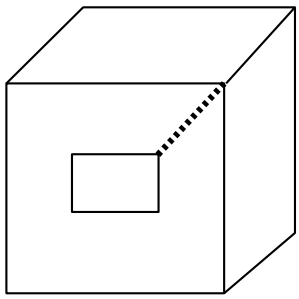


Figure 14.19: Cube with a loop

- *Example 3*: A cube with a loop, (see Figure 14.19)
  - Original case
    $V = 12, E = 16, F = 7, L_i = 1, S = 1, G = 0$
    $12 - 16 + 7 - 1 = 2 = 2(1 - 0)$
  - Connecting the interior loop with one corner vertex of the cube
    $V = 12, E = 16 + 1 = 17, F = 7, L_i = 1 - 1 = 0, S = 1, G = 0$
    $12 - 17 + 7 = 2 = 2(1 - 0)$
- *Example 4*: A sphere with a handle, (see Figure 14.20)

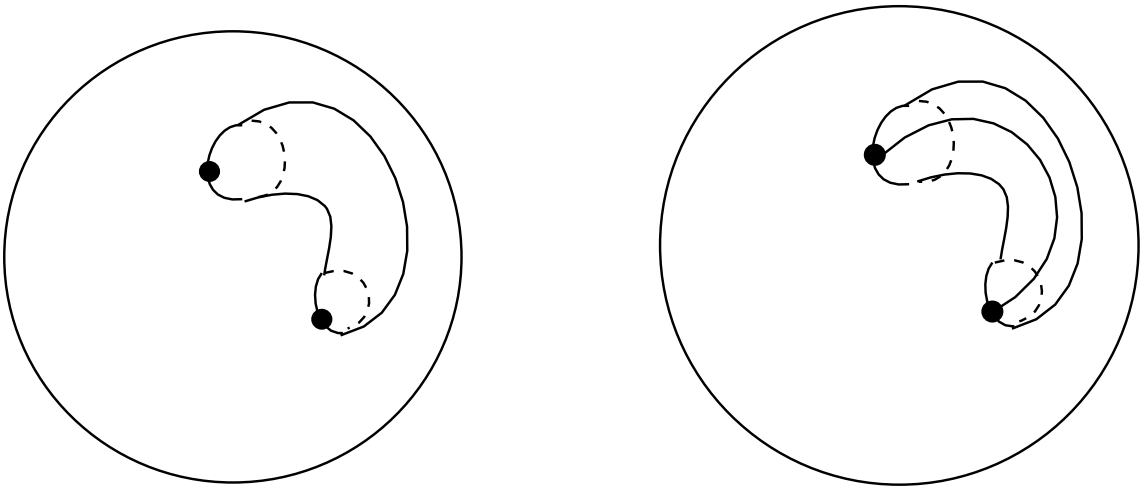


Figure 14.20: Sphere with a handle.

  - Original case
    $V = 2, E = 2, F = 2, L_i = 2, S = 1, G = 1$
    $2 - 2 + 2 - 2 = 0 = 2(1 - 1)$
  - Add one more edge on the handle (namely connect the two loops)
    $V = 2, E = 3, F = 2, L_i = 1, S = 1, G = 1$
    $2 - 3 + 2 - 1 = 0 = 2(1 - 1)$

14.6.4 Sufficiency of a geometric modeling representation

Sufficiency of a geometric modeling representation is the ability to completely and unambiguously represent all adjacency topological relationships of elements.

Theoretical sufficiency is the absolute minimum information required to reproduce unambiguously a complete adjacency topology.

In general, adjacency relationship informations V(E) (cyclicly ordered edges around vertices) and F(E) (cyclicly ordered edges around faces) are individually sufficient to represent all adjacency relationships.

Some combinations of single insufficient adjacency relationships are sufficient under certain conditions.

14.6.5 Boundary representation model

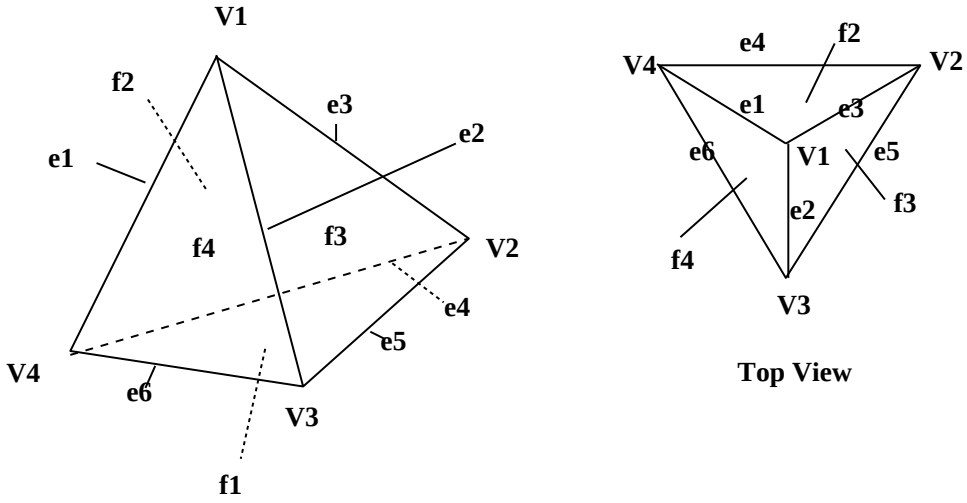


Figure 14.21: Boundary representation model.

Figure 14.21 shows an example of a tetrahedron. The boundary representation model for this example is shown in Figure 14.22. See also Table 14.1.

14.7 Data structures for manifold representations

Three different edge-based data structures for representing manifold topologies useful in solid modeling are:

1. the winged-edge data structure
2. the vertex-edge data structure
3. the face-edge data structure

The winged edge data structure keeps the edge information as a single unit while the face-edge and vertex-edge structures split the information related to each edge into two parts based on the specific usage of the edge in the adjacency relationships.

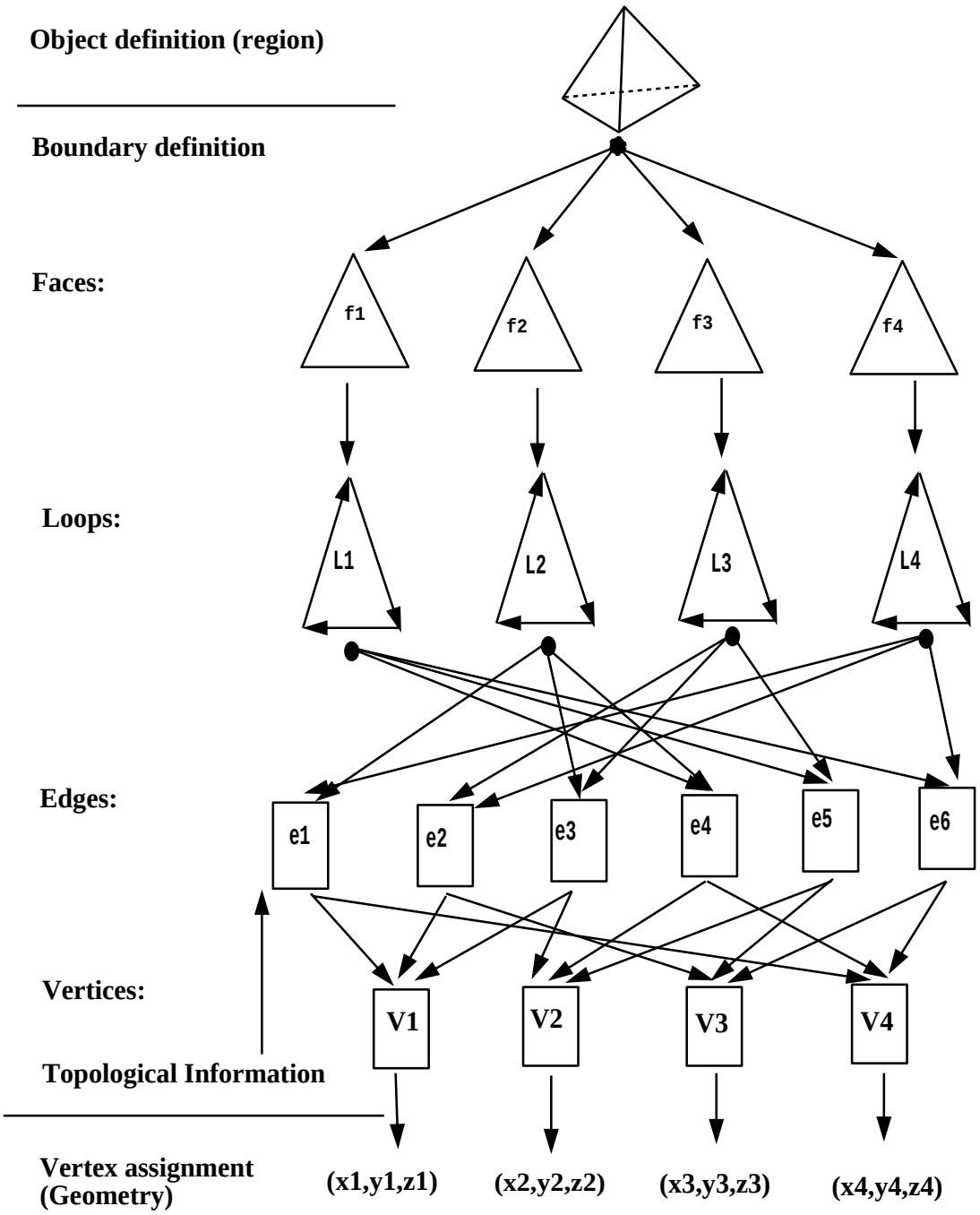


Figure 14.22: Boundary representation model for a tetrahedron

V(V)	V(E)	V(F)
$V_1(V) = (V_2\ V_3\ V_4)$	$V_1(E) = (e_1\ e_3\ e_2)$	$V_1(F) = (f_2\ f_3\ f_4)$
$V_2(V) = (V_3\ V_1\ V_4)$	$V_2(E) = (e_5\ e_3\ e_4)$	$V_2(F) = (f_1\ f_3\ f_2)$
$V_3(V) = (V_1\ V_2\ V_4)$	$V_3(E) = (e_6\ e_2\ e_5)$	$V_3(F) = (f_1\ f_4\ f_3)$
$V_4(V) = (V_2\ V_1\ V_3)$	$V_4(E) = (e_4\ e_1\ e_6)$	$V_4(F) = (f_1\ f_2\ f_4)$
E(V)	E(E)	E(F)
$e_1(V) = (V_1\ V_4)$	$e_1(E) = [(e_3\ e_2)(e_6\ e_4)]$	$e_1(F) = (f_2\ f_4)$
$e_2(V) = (V_1\ V_3)$	$e_2(E) = [(e_1\ e_3)(e_5\ e_6)]$	$e_2(F) = (f_3\ f_4)$
$e_3(V) = (V_1\ V_2)$	$e_3(E) = [(e_2\ e_1)(e_4\ e_5)]$	$e_3(F) = (f_2\ f_3)$
$e_4(V) = (V_2\ V_4)$	$e_4(E) = [(e_5\ e_3)(e_1\ e_6)]$	$e_4(F) = (f_1\ f_2)$
$e_5(V) = (V_2\ V_3)$	$e_5(E) = [(e_3\ e_4)(e_6\ e_2)]$	$e_5(F) = (f_1\ f_3)$
$e_6(V) = (V_3\ V_4)$	$e_6(E) = [(e_2\ e_5)(e_4\ e_1)]$	$e_6(F) = (f_1\ f_4)$
F(v)	F(E)	F(F)
$f_1(V) = (V_2\ V_4\ V_3)$	$f_1(E) = (e_4\ e_6\ e_5)$	$f_1(F) = (f_3\ f_2\ f_4)$
$f_2(V) = (V_4\ V_2\ V_1)$	$f_2(E) = (e_4\ e_3\ e_1)$	$f_2(F) = (f_1\ f_3\ f_4)$
$f_3(V) = (V_1\ V_2\ V_3)$	$f_3(E) = (e_3\ e_5\ e_2)$	$f_3(F) = (f_2\ f_1\ f_4)$
$f_4(V) = (V_1\ V_3\ V_4)$	$f_4(E) = (e_2\ e_6\ e_1)$	$f_4(F) = (f_2\ f_3\ f_1)$

Table 14.1: Adjacency relationships for boundary representation of tetrahedron.

ev_ptr [1]	ev_ptr [2]
ee_cw_ptr [1]	ee_cw_ptr [2]
ee_ccw_ptr [1]	ee_ccw_ptr [2]
ef_ptr [1]	ef_ptr [2]
e_ptr (edge attribute)	

Table 14.2: Winged-edge data structure.

14.7.1 Winged-edge data structure

(1) Data structure

Topological information stored for each edge is composed of the adjacencies of that edge with four other edges, two faces, and two vertices, see Figure 14.23.

(2) Application of winged-edge data structure, see Figure 14.24 and Table 14.3.

Question: adjacency relationship $f_4(E)$?
Start with edge "e_1" and traverse data structure
$e_1(CCW[2]) \longrightarrow e_6$, check $f_4 = f[2]$ and
$e_6(CCW[2]) \longrightarrow e_2$, check $f_4 = f[2]$ and
$e_2(CCW[2]) \longrightarrow e_1$

(3) Supporting data structure (see Figure 14.25)

- "Shell"

shell_attribute_ptr
face_ptr (doubly linked list of faces)

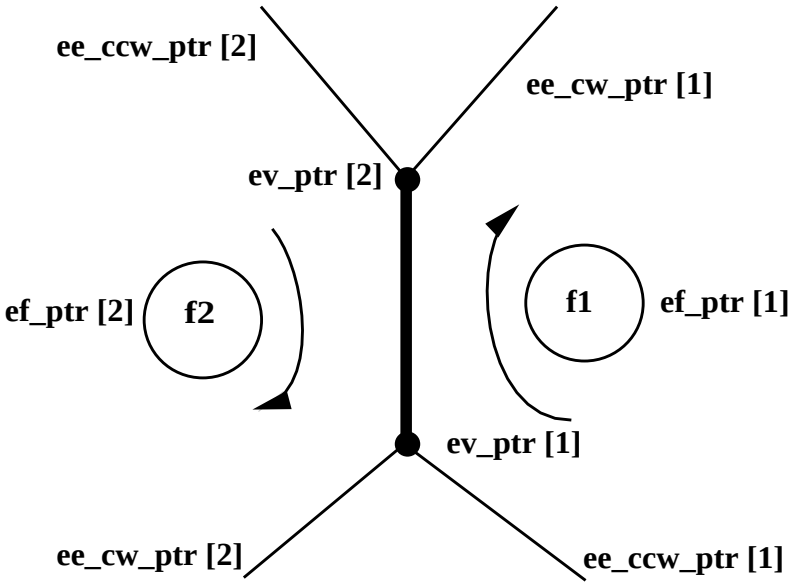


Figure 14.23: Winged-edge data structure.

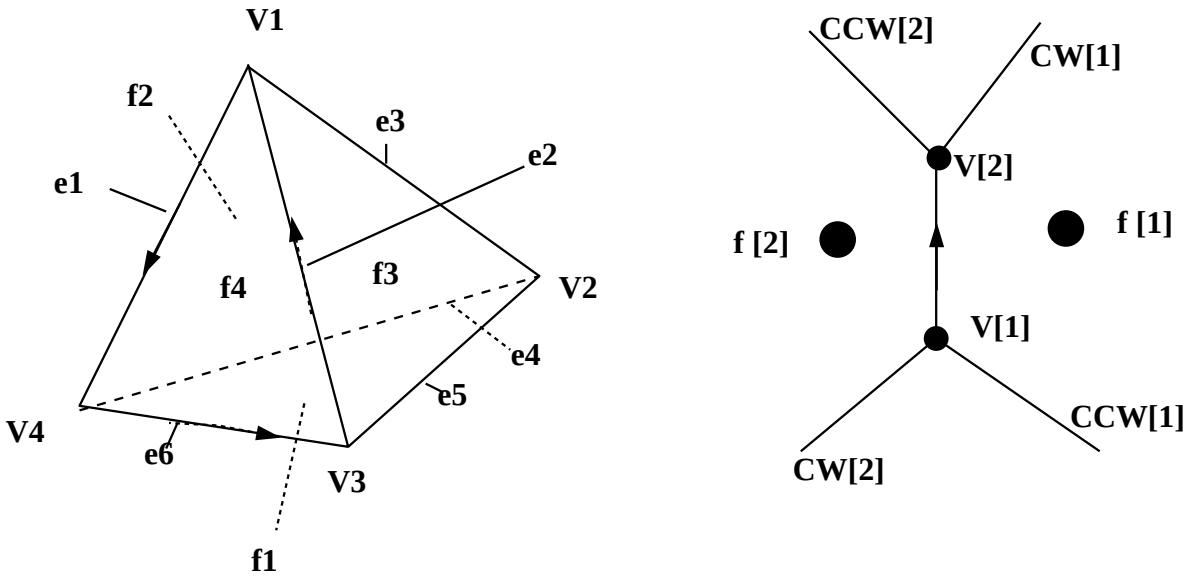


Figure 14.24: Application of winged-edge data structure.

edge	$V[1]$	$V[2]$	$f[1]$	$f[2]$	$CW[1]$	$CCW[1]$	$CW[2]$	$CCW[2]$
e_1	V_1	V_4	f_2	f_4	e_4	e_3	e_2	e_6
e_2	V_3	V_1	f_3	f_4	e_3	e_5	e_6	e_1
e_3	V_1	V_2	f_3	f_2	e_5	e_2	e_1	e_4
e_4	V_2	V_4	f_1	f_2	e_6	e_5	e_3	e_1
e_5	V_2	V_3	f_3	f_1	e_2	e_3	e_4	e_6
e_6	V_4	V_3	f_1	f_4	e_5	e_4	e_1	e_2

Table 14.3: Application of winged-edge data structure.

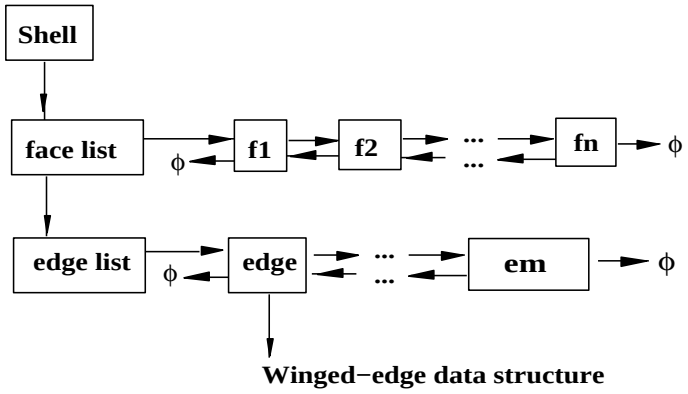


Figure 14.25: Supporting data structure.

- "face"

next _face_ptr
previous_face_ptr
edge_ptr or vertex_ptr

- vertex

vertex_attribute (geometry)

14.7.2 Vertex-edge data structure (V-E)

V-E data structure represents the adjacency information of the edge by splitting it into two structures, each of which is related to one of the two edge *end vertices*, see Figure 14.26 and Table 14.4. Each edge is used exactly twice in opposite directions by two adjacent faces. This results in the concept of "edge-use"

ev_ptr
ee_cw_ptr
ee_ccw_ptr
ef_ptr
ee_mate_ptr (other end of edge)
e_ptr (edge attribute)

Table 14.4: Vertex-edge data structure.

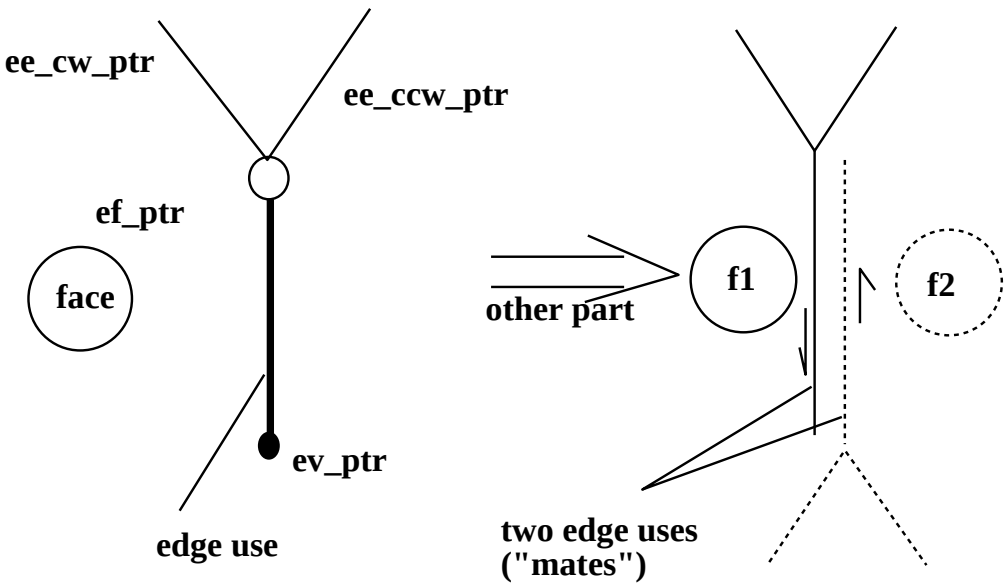


Figure 14.26: Vertex-edge data structure.

ev_ptr
ee_cwe_ptr
ee_ccwe_ptr
ef_ptr
ee_mate_ptr (other half of the edge)
e_ptr (edge attribute)

Table 14.5: Face-edge data structure.

14.7.3 Face-edge data structure (FE)

The F-E structure represents the adjacency information of the edge by splitting it into two structures, each of which is related to one of the two *edge sides* as found around the periphery of faces, see Figure 14.27 and Table 14.5. This results in the concept of "edge-use" .

14.8 Operators for manipulating manifold topologies

The Euler operators are a set of operators which can manipulate manifold boundary based topology representations in a low level, incremental and systematic fashion, constructing a topology primarily edge by edge. They can be used with any of the previously described edge based data structures.

They are, relatively speaking, low level operators since they act on topological primitive elements (vertices, edges and faces). We can also see them as high level operators because they allow us to construct a manifold adjacency topology without getting into the details of the underlying data structures. Indeed while implementation of the Euler operators is specific to the data structure actually used, the external interface to the operators can remain the same, and the implementation of all higher level operations can be identical regardless of the data structure chosen, see Figure 14.28.

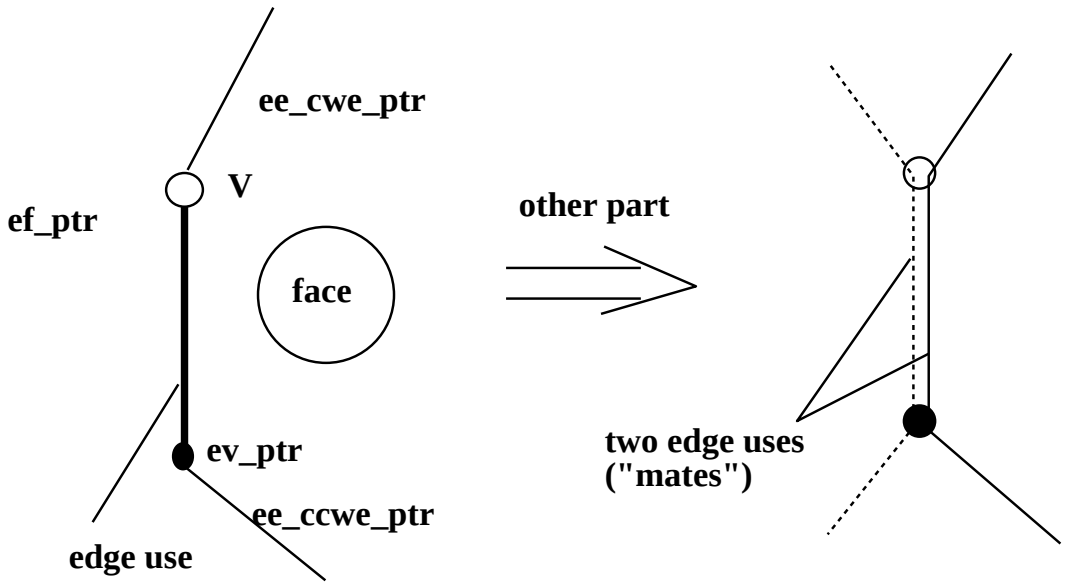


Figure 14.27: Face-edge data structure.

Five basic Euler operators presented below are *sufficient* to create any topology but others are also defined to add convenience and flexibility to the surface construction process.

- MSFLV: make-shell-face-loop-vertex
- MEV : make-edge-vertex
- ME : make-edge
- GLUE : glue faces (merge two simple loop faces together)
- KE : kill-edge

Examples of Euler operators are shown in Figure 14.29 and Figure 14.30. Figure 14.31 shows an application of Euler operators in the construction of a box. At every step, $V - E + F - L_i = 2(S - G)$, or in this case $V - E + F = 2$

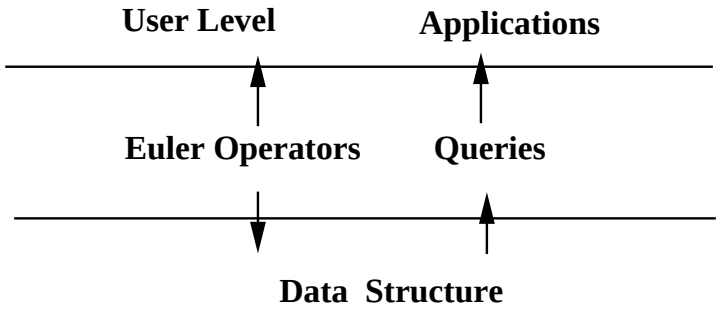


Figure 14.28: Abstraction levels using Euler operators.

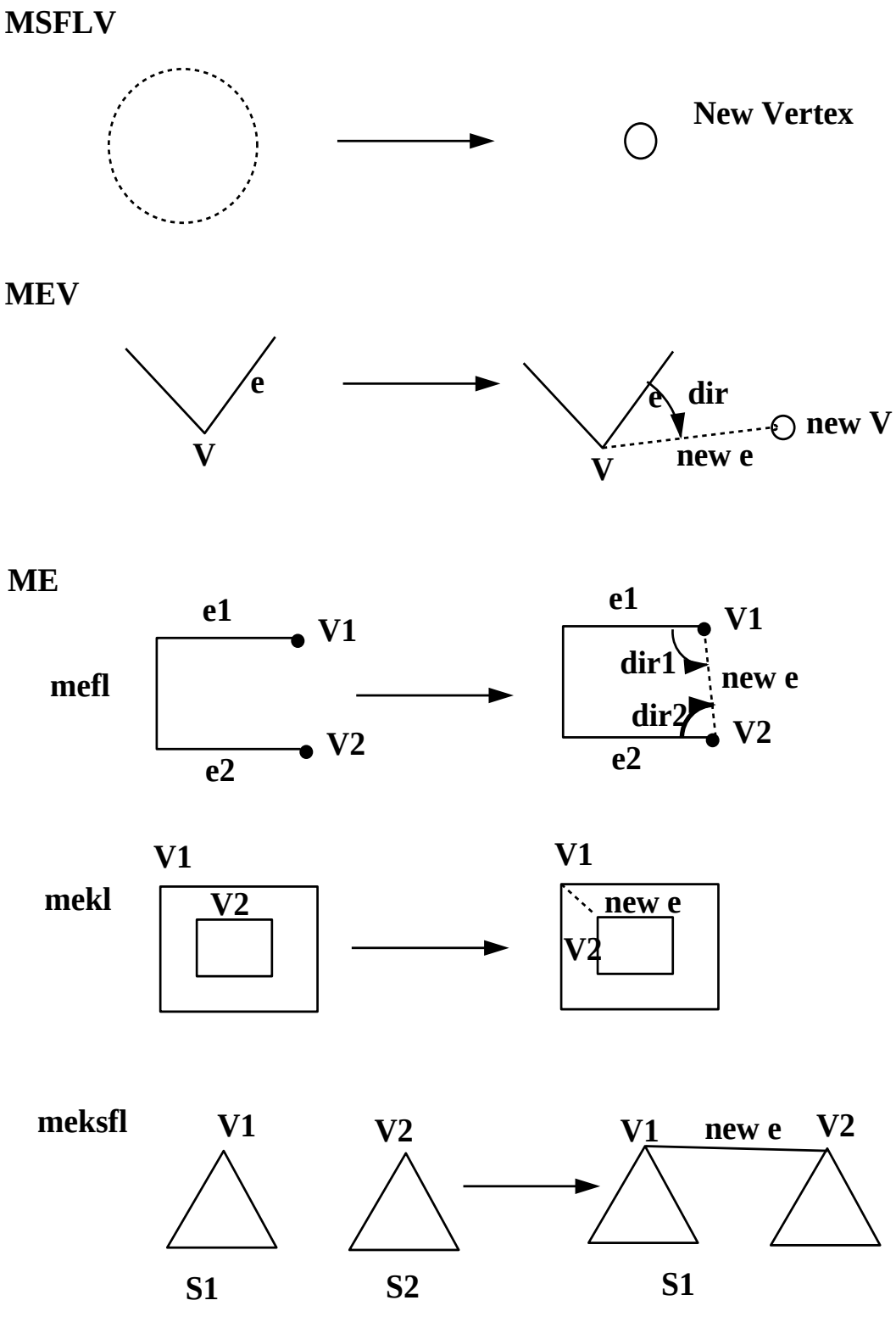
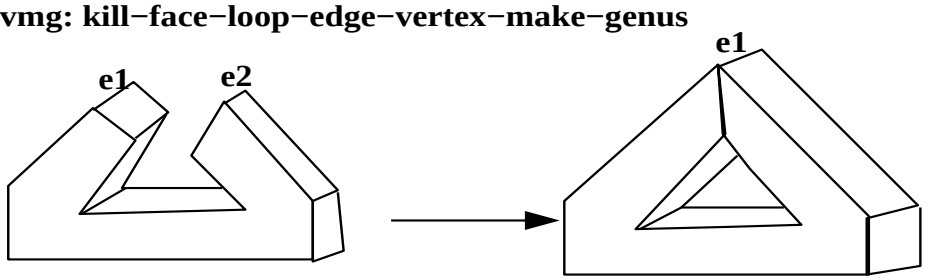


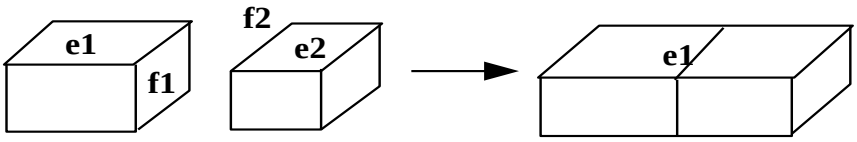
Figure 14.29: Euler operators (I)

GLUE

kflevmg: kill−face−loop−edge−vertex−make−genus

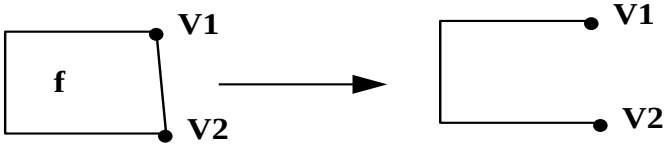


kflevs

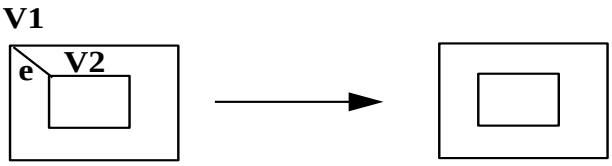


KE

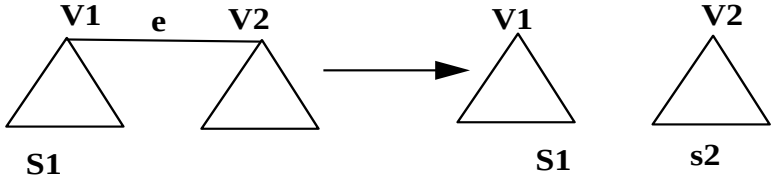
kefl



keml



kemsfl



KSFLEV: delete an object from model

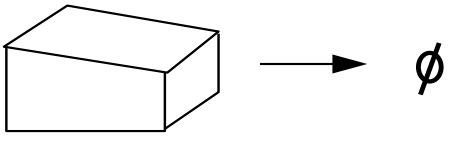


Figure 14.30: Euler operators (II)

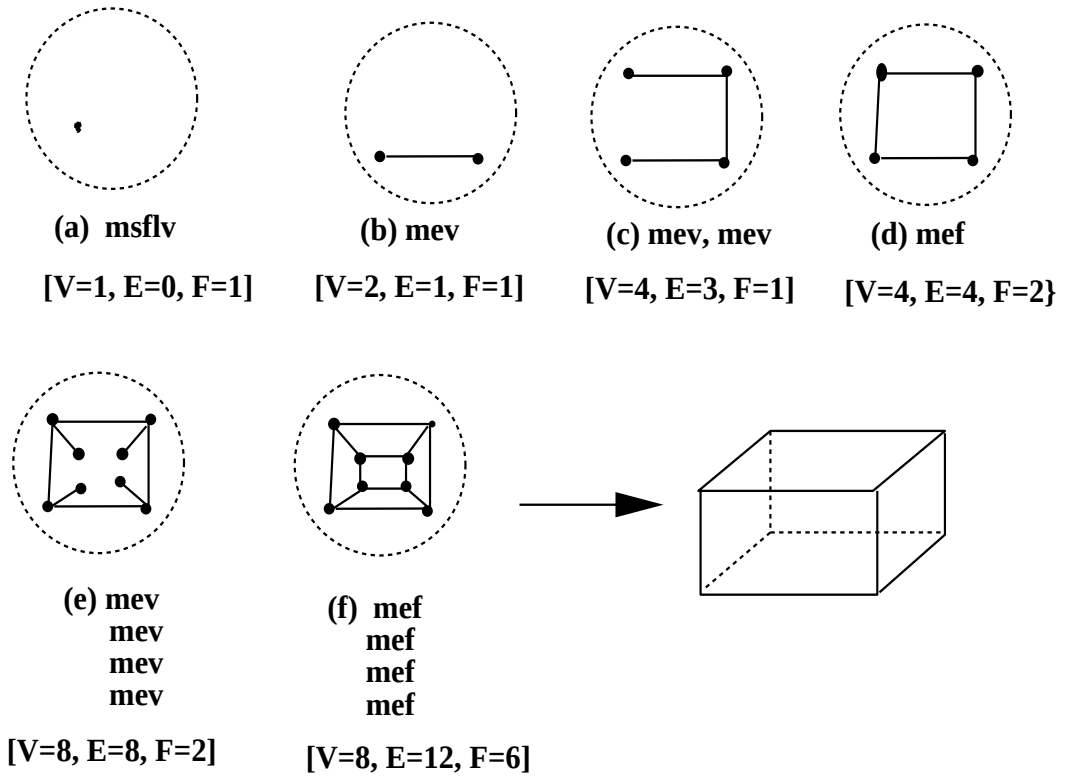


Figure 14.31: Making a box using Euler operators

14.8.1 Basic operators

- *Constructive*
  - MSFLV
  - MEV
  - ME
    - mefl
    - mekl
    - meksfl
  - GLUE
    - kflevmg
    - kflevs
- *Destructive*
  - KSFLEV
  - KEV
  - KE
    - kefl
    - keml

    - kemsfl
  - UNGLUE
    - mflevkg
    - mflevs

A brief description of these operators follows:

1. **MSFLV** (new-face, new-loop, new-vertex)
   "Make shell, face, loop, vertex" creates a new manifold surface in the topology and it is the *first* operator used in any topology construction. MSFLV creates a new shell, the face "new-face", the loop "new-loop", and the vertex "new-vertex". The single vertex created, "new-vertex" can be used as a starting point for subsequent construction of additional topological features on the manifold surface.

2. **MEV** (vertex, edge, direction, new-edge, new-vertex)
   "Make edge, vertex" creates a new edge and vertex. The new edge, "new-edge", starts at the existing vertex "vertex", and ends at the new vertex "new-vertex". If the optional placement arguments "edge" and "direction" are specified, "new-edge" will be positioned in direction "direction" from "edge" about "vertex" as seen when looking towards the manifold surface from outside above "vertex".

3. **ME** (ver1, edge1, dir1, ver2, edge2, dir2, new-edge, new-face, new-loop)
   "Make edge" creates an edge between existing vertices ver1 and ver2. If optional placement is specified, the new edge, "new-edge", will be direction "dir1" from "edge1" about vertex "ver1" and direction "dir2" from "edge2" about vertex "ver2".

   - mefl: "make edge, face, loop" occurs when the new edge will close off one portion of the face it is on from the rest of the face. In this case, the new face "new-face" and loop "new-loop" will lie to the "dir1" side of "new-edge" about "ver1".
   - mekl: "make edge, kill loop" occurs when the new edge will not close off one portion of the face it is on from the rest of the face. In this case, the vertices "ver1" and "ver2" were on different loops of the same face, but afterwards will be located on the same loop. The surviving loop is the loop associated with "ver1".
   - meksfl: "make edge, kill shell, face, loop" occurs when the two specified vertices are on different shells. The new edge links together the two shells into a single shell. The shell of "ver1" is the surviving shell.

4. **GLUE** (face1, edge1, face2, edge2)
   "Glue faces" merges two single loop faces (simply connected faces) together, deleting both faces and vertices, with the effect of joining together the volumes which the two faces are bounding. Both loops must have the same number of edges and vertices, and must have no edges in common. The merge is performed so that edge1 of face1 and edge2 of face2 are merged into the same edge. The surviving set of edges and vertices are those of face1.

   - kflevmg: "Kill face, loop, edge, vertex, make genus" occurs when both faces exist on the same shell. The glue operation increases the genus of the shell by one.

- kflevs: "kill face, loop, edge, vertex, shell" occurs when the two faces exist on different shells. The glue operation merges the two shells together into a single shell, with the shell of face1 being the survivor.

5. **KE** (edge, vertex, new-loop)
   "Kill edge", deletes the specified edge "edge".

   - kefl: "kill edge, face, loop" occurs when the edge to be deleted separates two different faces. In this case, the edges of the two loops using the deleted edge are merged and one face and loop are deleted. The surviving face and loop are those found to the right of the edge to be deleted, when transversing the edge from the optionally specified vertex "vertex" to the other vertex. Any other loops of the deleted face are moved to the surviving face.
   - keml: "kill edge, make loop" occurs when the edge to be deleted occurs twice on a loop of a single face. In this case a new loop, "new loop" will be generated on the same face.
   - kemsfl (edge, vertex, new-face, new-loop): "kill edge, make shell, face, and loop" deletes the specified edge, "edge", which is required to have the same face on both sides. The two disconnected graph components that result are treated as separate shells.
   - ksflev (vertex): "kill shell, face, loop, edge, vertex" determines the shell of the specified vertex and deletes the shell and all its topological elements.
   - kev (edge, vertex, vsurvivor): "kill edge, vertex" *squeezes* the ends of the specified edge "edge" together, deleting the edge and a vertex "vertex" while preserving adjacencies. The topological parameter " vertex", if specified, designates which vertex of the edge will survive. In any case, the surviving vertex is indicated by the "vsurvivor" return parameter.

6. **UNGLUE** (edge, face, newf1, newf2, loop, newl1, newl2)
   "Unglue faces" takes a single loop of edges starting with edge "edge", separates the model along the loop. The process creates two new faces "newf1" and "newf2" and their respective loops "newl1" and "newl2". The loop marked for the UNGLUE must be complete and must not cross itself.

   - mflevkg: "make face, loop, edge, vertex, kill genus" occurs when the separation induced by the operation leaves the object still connected. In this case the specified loop lies on a handle of the shell which has a genus of one or more. The handle is removed and the single shell with genus reduced by one in the result.
   - mflevs: "make face, loop, edge, vertex, shell" occurs when the separation induced by the operation creates disconnected shells. Each component of the result is treated as a separate shell. Thus two separate volumes are created.

The effects of the basic Euler operators on topological elements are summarized in Table 14.6.

Operator	Change in number of topological elements					
	Shells	Faces	Loops	Edges	Vertices	Genus
MSFLV	+1	+1	+1		+1	
MEV				+1	+1	
ME						
mefl		+1	+1	+1		
mekl			-1	+1		
meksfl	-1	-1	-1	+1		
GLUE						
kflevmg		-2	-2	$-n_e$	$-n_v$	+1
kflevs	-1	-2	-2	$-n_e$	$-n_v$	
KSFLEV	-1	$-n_f$	$-n_l$	$-n_e$	$-n_v$	$-n_g$
KEV				-1	-1	
KE						
kefl		-1	-1	-1		
kekl			+1	-1		
kemsfl	+1	+1	+1	-1		
UNGLUE						
mflevkg		+2	+2	$+n_e$	$+n_v$	-1
mflevs	+1	+2	+2	$+n_e$	$+n_v$	

Table 14.6: Effects of basic Euler operators on topological elements

14.8.2 Building high level functions on the Euler operators

Euler operators provide a flexible basis for higher level operators while insulating those new operators from the details and complexities of the actual data structures used. They are flexible, because they are fairly low level operators which systematically manipulate the model on an edge by edge basis. They also provide automatic topological integrity checking. Almost any other kind of commonly found modeling operator or procedure can be built on top of the Euler operators, including parametric primitives and sweeps.

14.9 Non Two-Manifold B-rep

Non two-manifold (NTM) representations are geometric modeling representations which allow volume, both manifold and non-manifold surface, curve and point elements in a single uniform environment. This allows topological surfaces which are not constrained to be homeomorphic to a two-dimensional topological disk at every point, such as the objects in Figure 14.32.

A NTM representation therefore allows a general wire mesh with surfaces and volumes embedded in space and can be a superset of wireframe, surface and traditional manifold solid modeling forms.

This section will discuss a representation of NTM topologies and introduce the Radial Edge Data Structure, a data structure developed for NTM topologies.

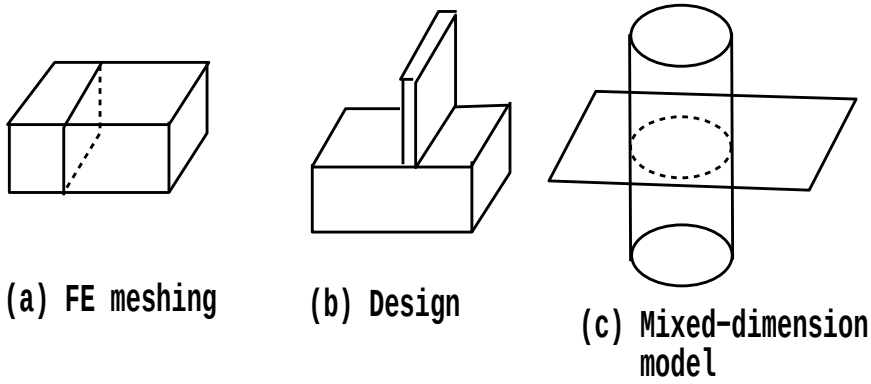


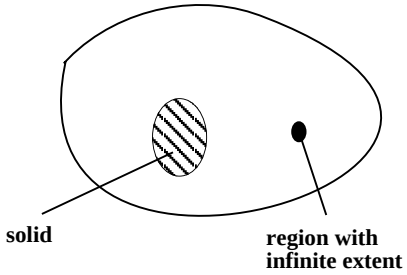
Figure 14.32: Examples of non-two manifold models.

14.9.1 Topological elements in NTM topologies

At least seven distinct element types, including six basic topological element types are involved in a NTM topology representation.

A *model* is a single 3D topological modeling space, consisting of one or more distinct regions of space. A model is not strictly a topological element but acts as a repository for all topological elements contained in a geometric model, allowing the naming and manipulation of multiple models by a geometric modeling system.

A *region* is a volume of space. There is always at least one in a model. Only one region in a model may have infinite extent; all others have a finite extent, and when more than one region exists in a model, all regions have a boundary.



A *shell* is an oriented boundary surface of a region. A single region may have more than one shell (such as a solid object with a void contained within the solid). A shell may consist of a connected set of faces which form a closed volume or may be an open set of adjacent faces, a wire frame or a combination of these or even a single point.

A *face* is a bounded portion of a shell. It is orientable, though not oriented, as two region boundaries (shells) may use different sides of the same face. Thus only the "use of a face" by a shell is oriented.

A *loop* is a connected boundary of a single face. A face may have one or more loops. Loops normally consist of an alternating sequence of edges and vertices in a complete circuit, but many consist of only a single vertex. Loops are also orientable but not oriented as they bound a face which may be used by up to two different shells. Thus the "use of a loop" is oriented.

An *edge* is a portion of a loop boundary between two vertices. Topologically an edge is a bounding curve segment which may serve as part of a loop boundary for one or more faces

which meet at that edge. Every edge is bounded by a vertex at each end (possibly the same one). An edge is orientable, though not oriented. The "use of an edge" is oriented.

A *vertex* is simply a topologically unique point in space. Single vertices may also serve as boundaries of faces and as *complete shell boundaries*.

At least four additional types of topological element adjacency "uses" associated with the *face*, *loop*, *edge* and *vertex* elements may be defined.

A *face-use* is one of two sides of a face. Face-uses, ie. the uses of a face by a shell, are oriented with respect to the face geometry.

A *loop-use* is one of the uses of a loop associated with one of the two uses of a face. It is oriented with respect to the associated face use.

An *edge-use* is an oriented bounding curve segment on a loop-use of a face-use and represents the use of an edge by that loop-use or if a wire frame edge by the point vertices. Orientation is specified with respect to edge geometry. There may be many uses of a single edge in a model, but there will always be an even number of edge-uses (since each use by a face produces two edge uses with one for each side). A wireframe edge produces two edge-uses, one for each end of the edge.

A *vertex use* is a structure representing the adjacency use of a vertex by an edge as an end point, by a loop in the case of a single vertex loop or by a shell in the case of a single vertex shell.

14.9.2 Topological sufficiency

Thirty six topological element adjacency relationships are possible in a NTM boundary representation with respect to six basic topological elements (vertex, edge, loop, face, shell and region). There is no complete theory available yet concerning the identification and proof of the theoretical minimal amount of topological information required to reconstruct a NTM topology. The radial edge data structure developed for NTM topologies has been proven to be sufficient and complete.

14.10 Radial edge data structure

The radial edge data structure explicitly represents eleven topological element types. A hierarchical representation of topological elements in the radial edge structure is shown in Figure 14.33. Key ideas in design of the Radial Edge data structure are:

- Top-down and bottom-up hierarchical relationships are represented.
- Face-use, loop-use, edge-use and vertex-use are utilized to represent the adjacencies of topological elements.
- Radial ordering of faces around common edges is represented directly (this information allows representation of a NTM condition along an edge, see Figure 14.34).
- Adjacency information pointing from vertex level to higher levels is represented (this information allows representation of a NTM condition at a point, see Figure 14.34).

The radial edge data structure can be implemented in terms of a set of doubly linked lists (DLL) and supporting data structures, see Figures 14.35- 14.38. The Euler-Poincare formula is

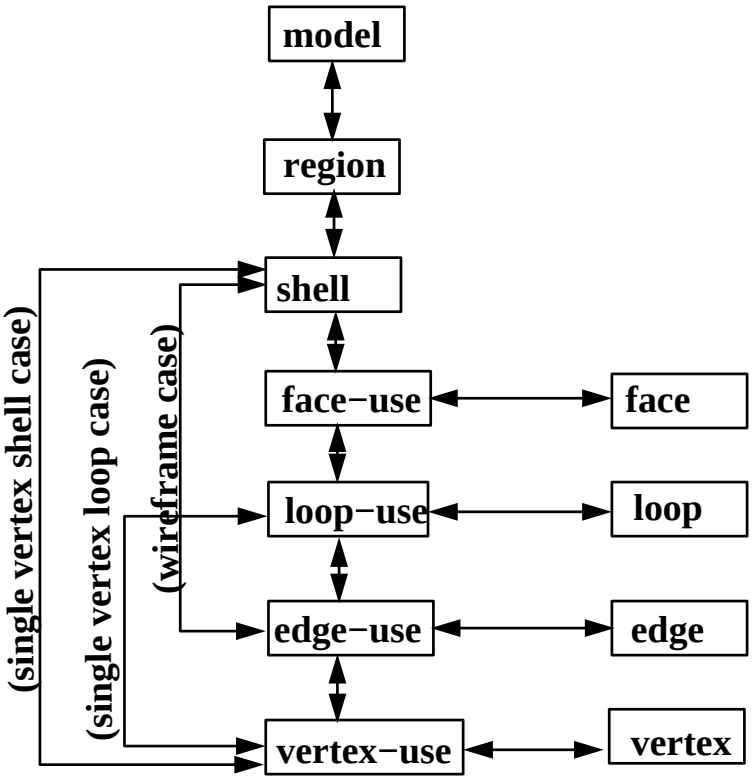
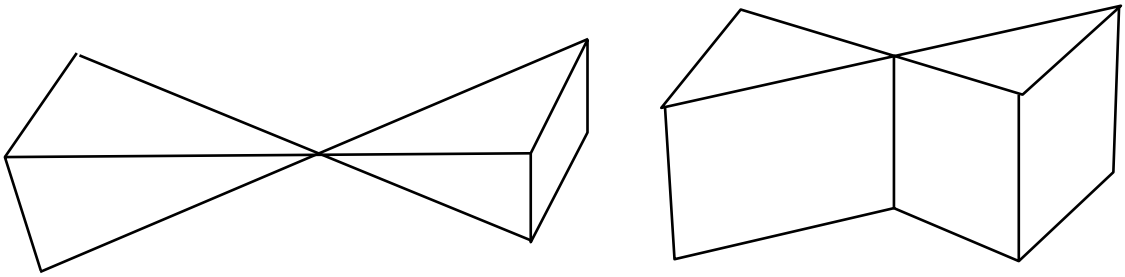


Figure 14.33: Radial edge data structure (up-down point for boundary and ownership relationships; left-right pointer for definition and attributes of an element).



(a) NTM condition at a point

(b) NTM condition along an edge

Figure 14.34: NTM conditions.

no longer valid for NTM topologies. NTM topology operators similar to Euler operators have been specified and defined by Weiler.A *minimal* sufficient set of NTM operators to construct any model are:

- M-MR = create the model and initial region
- M-SV = make shell vertex (for every vertex)
- M-E = make edge
- M-F = make face
- K-E = kill edge.

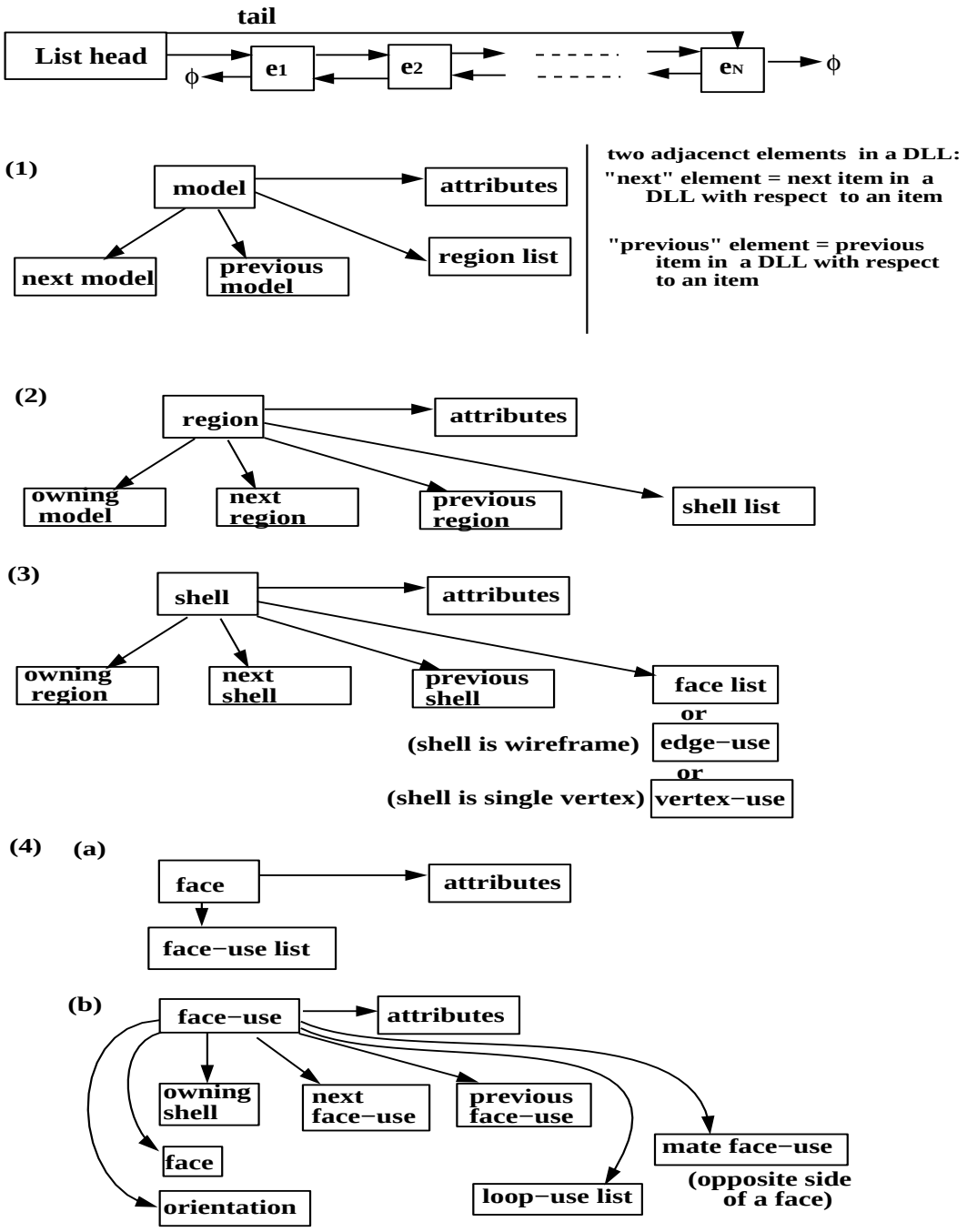


Figure 14.35: Implementation of radial edge structure.

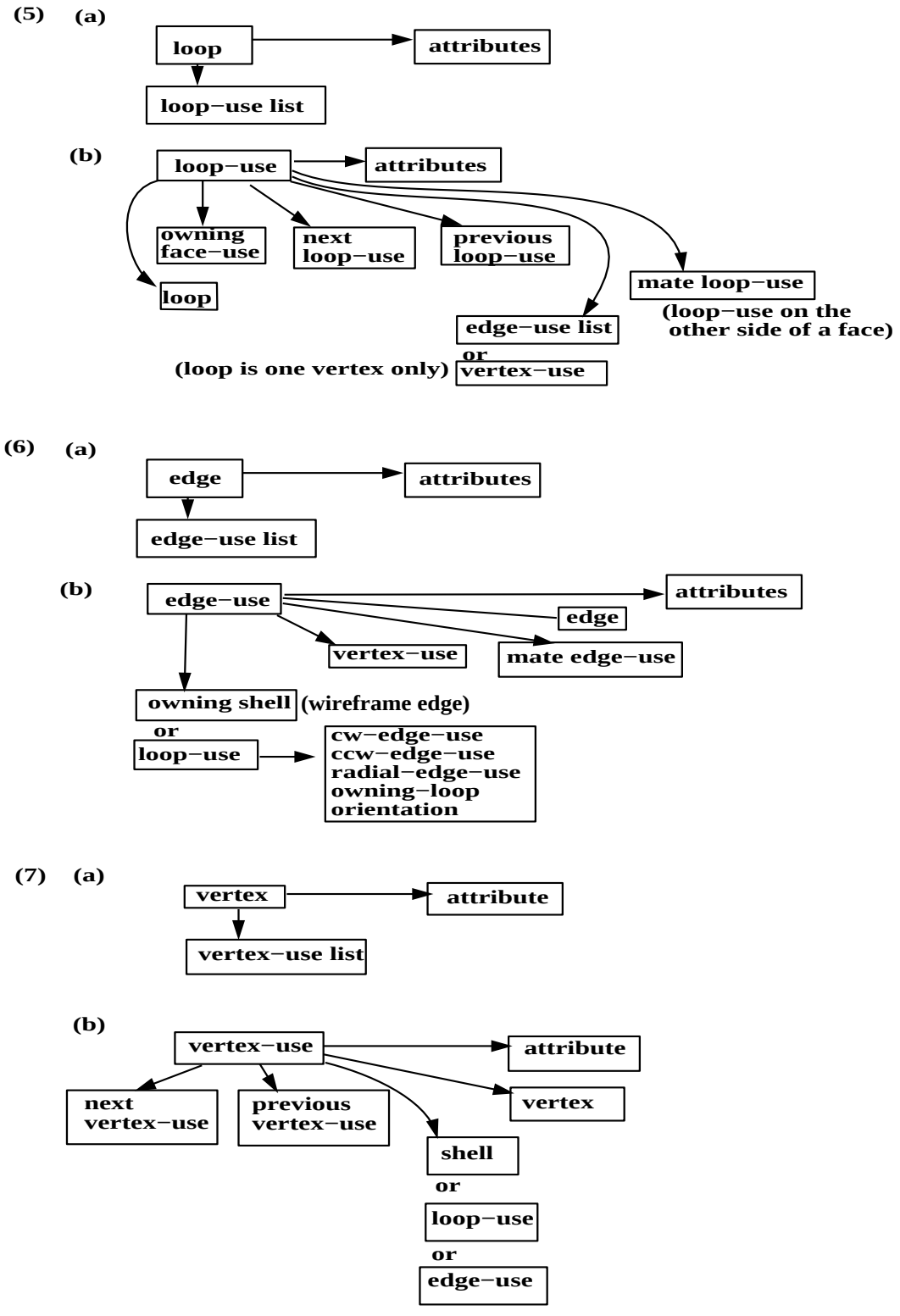


Figure 14.36: Implementation of radial edge structure (continued).

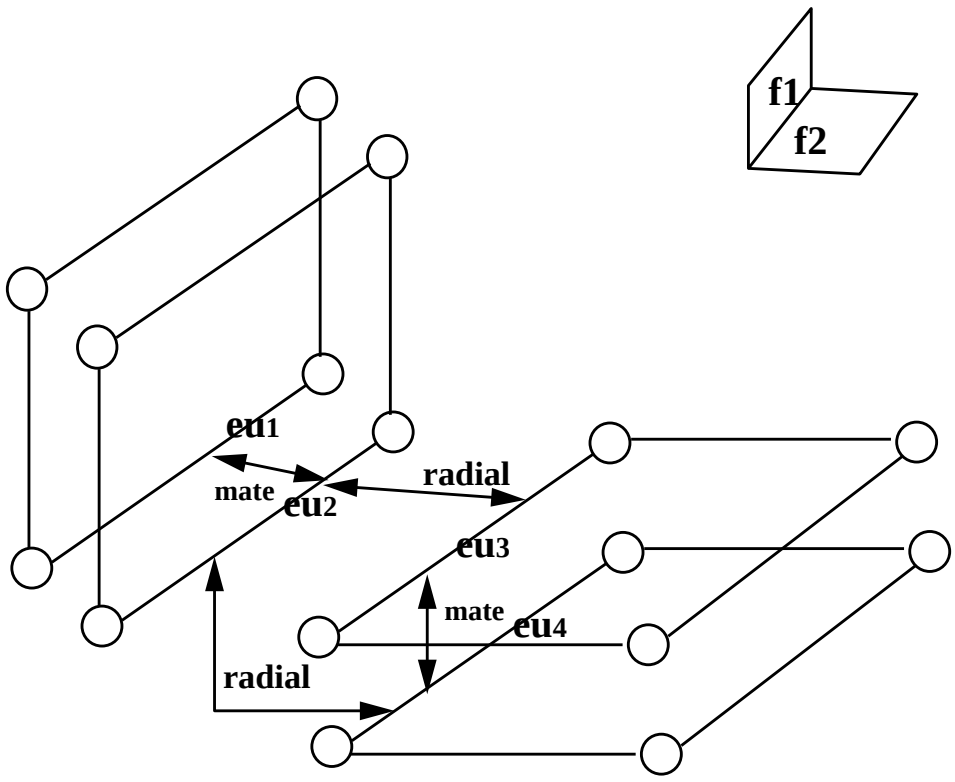


Figure 14.37: Radial edge representation of two faces joining along a common edge showing how the four edge uses of the common edge (each side of each face uses the edges) are connected.

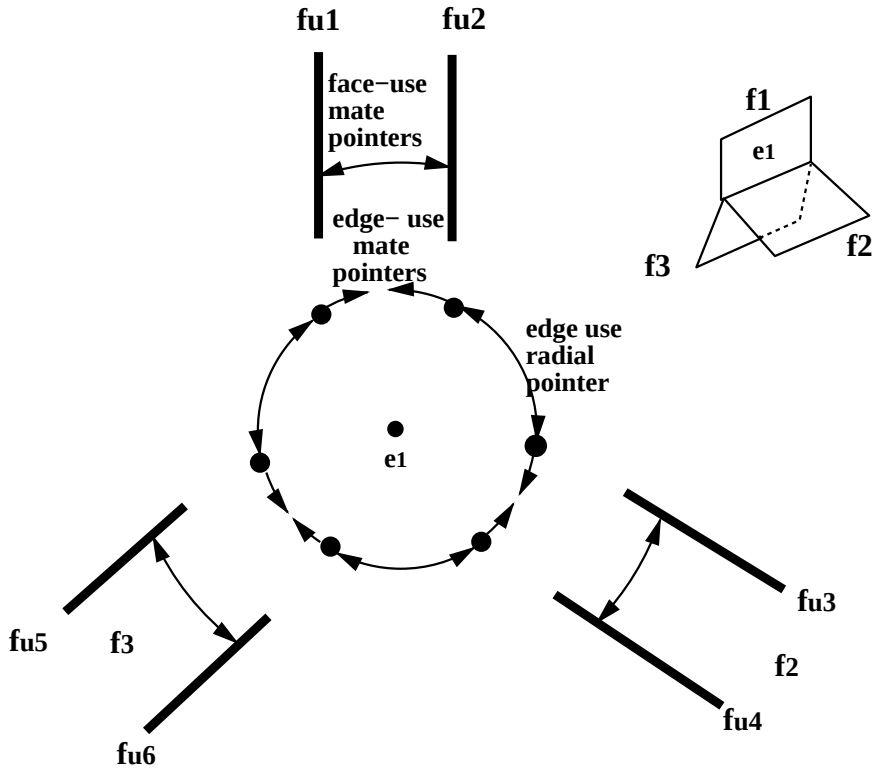


Figure 14.38: Cross-section of three faces sharing a common edge in the radial edge representation

Bibliography

[1] L. Bardis and N. M. Patrikalakis. Topological structures for generalized boundary representations. MITSG 94-22, MIT Sea Grant College Program, Cambridge, Massachusetts, September 1994.

[2] B. G. Baumgart. Winged edge polyhedron representation. Technical Report STAN-CS-320, Computer Science Department, Stanford University, 1972.

[3] E. Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete and Computational Geometry*, 9:387–426, 1993.

[4] E. L. Gürsöz and F. B. Prinz. Node-based representation of non-manifold surface boundaries in geometric modeling. In M. Wozny, J. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*. North-Holland, 1989.

[5] C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1989.

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

[7] M. E. Mortenson. *Geometric Modeling*. John Wiley and Sons, New York, 1985.

[8] J. Rossignac and M. O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wosny, J. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 145–180. North-Holland, 1989.

[9] V. Shapiro. Solid modeling. *Invited paper in Handbook of Computer Aided Geometric Design, Chapter 20*, pages 473–518, July 2002.

[10] K. Weiler. Edge-based data structures for solid modeling in curved surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.

[11] K. Weiler. *Topological Structures for Geometric Modeling*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 1986.

[12] P. R. Wilson. Euler formulas and geometric modeling. *IEEE Computer Graphics and Applications*, 5(8):45–60, August 1985.