Lecture XIII
SOLID MODELING

Modeling is a major theme in all of the sciences. Examples include modeling the weather pattern over a geographic region or combustion inside a jet engine. In many examples, the models will have a significant geometric character (for instance, the topography of the region or the interior chambers of the jet engine). In this chapter, we focus in those physical models that are "purely geometric". Such models are commonly encountered in engineering and architecture: computer-aided design (CAD); computer-aided manufacturing (CAM); automobile, ship or aircraft design; robotics; virtual reality systems; and electronic prototyping. Geometric models are needed in the computational sciences for simulations, computational fluid dynamics, etc. The subject usually called **geometric design** or **solid modeling**. addresses the problems of representing, manipulating and visualizing such models. Representation issues requires topological tools, while the manipulation of nonlinear representations requires numerical and algebraic computation. Visualization implies that this subject is intimately related to computer graphics. Two references are the books by Martti Mäntylä's [2] and Chris Hoffmann [1].

§1. Solid Modeling

The models we address are typically 3-dimensional solids. We may define a **solid** to be closed subset of $\mathbb{R}^3$. This is too general as real solids has many other additional properties. The prototypical solid is the convex polytope. More generally, we have the general polyhedron which is just a finite union of convex polytopes There are four basic approaches to solid modeling:

Spatial Decomposition This method decomposes the target solid into simpler objects (cells), together with their adjacency or incidence relations. This is basically a 3-D cell complex. Mesh generation is one way to create this cell complex.

Constructive or Operational Representation This is sometimes called "constructive solid geometry" (CSG) but more generally, it represents a solid by an expression whose leaves are primitive objects. The internal nodes represent operations such as the Boolean operations. The objects in each node of the expression are typically solids, but this need not be so. For instance, we could have a node represening a planar region, which is swept to generate a volume.

Boundary Representation Here, we only represent the boundary of the solid. Invariably, the boundary is the 2-D cell complex. For short, we call this the B-Rep approach. The simplest B-Rep models are polyhedral and the most flexible are the free-form surface designs. A major computational bottleneck of free-form design is computing intersections.

Wire Frames This is inherently a 1-D representation (just as B-Reps are inherently 2-D), The merit of wire frames is its simplicity and compactness in representing solids, but it is clearly an incomplete representation. It is limited to polyhedral solids. This method was important in the early days of computer graphics and geometric modeling, but is now a curiosity.

Perhaps a fifth approach is the representation of solids by planar, sectional and perspective views. This is the traditional drafting approach, but we decline to call a collection of such views a model of the solid. We can also combine the above approaches – for instance, a CSG system might also maintain the surface representation.

In considering the relative merits of the first three approaches above, it might appear that the first two approaches are superior to the boundary representations since they directly model the interior of the solid.

Indeed, for a class of applications where the interior of the solid is critical (e.g., modeling soft tissues of the brain, or the atmosphere), a surface representation is worthless.

Visualization is a critical part of the modeling function. Indeed, visualization is sometimes the sole object of the modeling. There are two basic kinds of objects in visualization: opaque versus translucent. For the former, a direct representation of the solid surface is more useful (e.g., it is hard to visualize an opaque solid directly from its CSG representation). Traditional computer graphics has developed many tools for this problem. To visualize translucent objects, we need the tools of volume graphics. This is an increasingly important form of visualization. However, our main interest here is opaque solids.

The solid representation to modeling (via spatial decomposition or CSG) has the advantage of being unambigous. Perhaps the most important advantage is this: a single CSG expression can represent an entire parametrized family of solids. For instance, we can parametrize the position of a subsolid by the amount of its translation. See figure 1 which shows the picture of a bracket, but this is supposed to be only one possibility of a parametrized family. Unfortunately, it is easy to generate undesirable and unexpected models in the parametrized family (see the next item below).

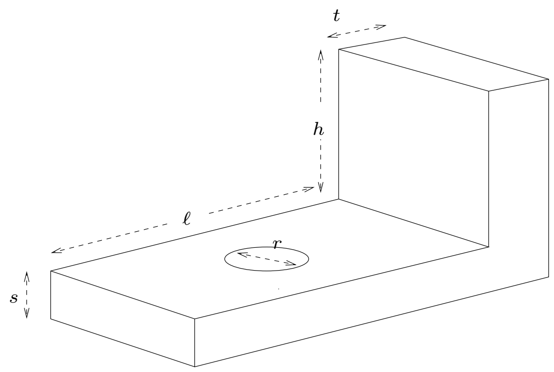

Figure 1: A parametrized family of brackets using CSG.

Nice Representations. A representation system must maintain a balance among several desirable properties, including efficiency (e.g., wire frames), modeling power (e.g., spatial representation can represent internal properties of solids), and flexibility (e.g., parametrized models in CSG). Over flexible or over powerful representations may allow us to represent objects that are not "nice". This is an informal notion, but here are some examples of such objects.

- Impossible objects. The artist Escher popularized many "impossible objects" in the graphic arts. An over flexibe system may also let us to create impossible objects.
- Ambiguous objects. The wire frame cannot easily distinguish between the surface of a cubic or the surface of the cube with one face omitted.
- Irregular solids. Given a set $S \subseteq \mathbb{R}^3$, we define $S^* = \text{closure}(\text{interior}(S))$. We say S is **regular** if $S = S^*$. Our system may be able to represent irregular solids. Traditional CSG considers this a demerit, but this may not be a universal view today.
- Solids with internal holes. This solid is not "manufacturable" by conventional means; perhaps we do not want to model such objects.
- Nonmanifold or nonorientable surfaces. Some B-Rep model allow nonmanifold and/or, nonorientable surfaces.

FIGURE of IMPOSSIBLE OBJECT (Mantyla)

FIGURE OF AMBIGUIOUS OBJECT

User Level Issues. What is a **feature**? Parametrized features are even harder to understand: When is a "corner" no longer a corner? E.g., A "slot" in a square black is moved to the boundary; is it still a slot? How can we introduce **constraints** into the system? We are interested in **design by functionality**.

Some Algorithmic Issues. Generally, robustness and efficiency is an issue. The following are some basic problems:
Compute the intersection of 2 solids;
Given two intersecting surfaces, find a blending surface;
Given a point and a surface, does the point lie on the surface?
Compute the envelop of sweeping an piecewise algebraic curve along a space curve.

Beyond Geometry. Although we focus on geometric issues only, there will be other demands on a geometric modeler in the real world. Issues such as manufacturability, mechanical properties, etc, may have to be answered from the model.

§2. Planar Modelers

Before we consider a 3-dimensional modeling system, it is instructive to consider a simple 2-dimensional system. It will serve to expose some of the basic issues of any geometric modeler. Even in a 2-dimensional system, we can introduce advanced features that are beyond the capabilities of most current modelers. For instance, we can introduce physical attributes such as mass, friction and dynamics into our model.

Geometric Objects. Geometric objects are classified as **primitive** or **nonprimitive**. The distinction is purely based on their representational size: primitive objects have constant size, and non-primitive ones have unbounded size. Primitive objects include **(real) numbers**, **points**, **generalized line segments** and **triangles**. By generalized line segments, we mean both finite line segments as well as lines and rays. Nonprimitive objects include **polylines**, **polygons** (simple or otherwise) and **plane graphs**. The special value is considered an instance of any of these objects.

It is useful to introduce nonstandard real numbers (these can be infinitely large or infinitely close to an actual real number). We use ω for the infinitely large number and ε for the infinitely small number, where $\omega = 1/\varepsilon$. Thus nonstandard numbers are just elements of the function field $\mathbb{R}(\omega)$. Henceforth, numbers refers to elements of $\mathbb{R}(\omega)$. Numbers in the set $\mathbb{R}$ is called **standard** and otherwise **nonstandard**.

We represent a point P as a pair (x, y) of numbers or as a non-zero triple (x, y, w) in homogeneous coordinates (with w as the homogenizing coordinate). The advantages of the latter is that any affine transformation (in particular, translation, rotation and scaling) can be written as a matrix multiplication. A point with nonstandard numbers as coordinates is called a nonstandard point. Similarly for nonstandard segments, etc.

For simplicity, generalized line segments are simply called **segments**. A segment can be represented by a pair of non-standard points. A segment is classified as finite if it has two finite endpoints, and infinite otherwise. A triangle can be represented by a triple of points. A polyline is a finite sequence of points, with an associated sequence of edges connecting successive points. A polygon can be represented by a polyline with the implicit rule that the last point is connected to the first by an edge. A plane graph is represented by a planar graph G together with coordinates for each of its vertices.

We will adopt the object-oriented programming language terminology and call points, segments, etc, as **(object) classes**. Individual elements of these classes are called **instances** or **members**. Special functions that create members are called **constructors**. Let us give some simple examples of the constructors.

Object	Constructor	Example	Remarks
Number	Number(Expression)	A = Number(1.2 + 3/4), B = Number(sqrt(2))	Expression can be null or Omega or Epsilon.
Point	Point(Number, Number)	P = Point(2, 0), Q = Point(1,1)	
Segment	Segment(Point, Point), Segment(Point, Point, type)	Segment(P, Q), Segment(P, Point(0,1), ray)	 type=ray or line
Triangle	Triangle(Point, Point, Point)	Triangle(P, Q, R)	

Attributed Objects. As a design decision, we will try not to proliferate the class of objects. Let us illustrate this: for instance, segments can be given an orientation. Instead of saying that oriented and unoriented segments are two classes of objects, we say they are the same class of objects but with different orientation attributes. Similarly, polygons can be oriented or not.

In order to make the system as flexible as possible, we allow object to have *as many or as few* attributes as it wishes. It is nonobvious in our "attributed design" is that we intend to make the coordinates x, y of point, or the equation $aX + bY + c = 0$ of a line to be attributes as well! Consider the alternative (which is quite reasonable): suppose we require that certain **defining attributes** for a particular class of objects to always have a value. For instance, we may insist that the coordinates of a point is always defined. Unfortunately, there are situations where even this is undesirable! Hence, we give up trying to impose any requirements. In any case, the possibility of null values means that the enforcement of defining attributes may be meaningless (or hard to achieve).

What are attributes? (1) Each attribute has a unique name (e.g. "orientation", "x-coordinate", "color"). (2) Each attribute has a value type (e.g., string value, number value, point class) (3) There might be dependencies among certain attributes: for instance, a line may have two points that lie on the line as attributes, but it may also have a line equation attribute. These are not independent.

What are the attributes of interest? Perhaps the most important attributes are display attributes: we can attach **visibility** or transparency coefficients and **texture maps** to each object. These attributes will be used by display functions. Objects such as points and lines can have **size** attributes. Another useful attribute is **label** and **name**. Both of these are string values, but names must be unique among objects of the same class.

When we consider physical modeling, we can introduce more attributes such as mass, velocity, acceleration, coefficient of friction, etc. Edges may have spring constants, etc.

Predicates and Operators. Operators are functions on geometric objects. For instance, we have an operator for computing the distance between a point and another geometric object.

Predicates are those operators whose result is a Boolean value. For instance, we may want to check if a triangle is degenerate, that is, its three vertices are collinear. We may have the intersection operator that takes two line segments and define their intersection which is a point, possibly null.

Thus constructors above are just a very special kind of operator. A general operator is the **copy** operator, which recreates an unshared copy of an object.

Grouping. A collection of geometric objects forms a **grouping**. A grouping can have a **transform** attribute, which is a general affine transformation. It can also have visibility attributes, which is independent of the visibility attributes of the individual objects in the grouping.

Parametric Grouping and Constraints. A grouping can have **parameters** which are number variables. Each parameter may be given a number as its **value**. Objects in the grouping may use these parameters in place of numbers. E.g., a scaling parameter allows us to change the size of all the objects in a grouping.

Constraints are special attributes associated to a grouping, and is only possible when we have parameters. Examples of constraints:

- a point must lie on another line
- 2 points at a fixed distance
- two lines are parallel
- three points are collinear
- three lines are coincident

Constraints introduce a new level of sophistication. We will focus only on **polynomial constraints**. Thus we introduce a class called **polynomials** and constraints are just inequalities for polynomials.

In an initial implementation, we will dispense with this generality, but only have very simple constraints: equality or inequality among parameters. As a next step up, we may introduce linear constraints.

Ruler and Compass Constructions and Theorems We illustrate the use of the constraint system in the design of a ruler-and-compass constructor system.

2+ϵ-dimensional System. Examples: suppose we want to represent a polygonal knot. Then we want each intersection of line segments to have a specified "above/below" relationship. We call this a (2+ϵ)-dimensional system.

Physical Mechanisms. rigid rods, springs, strings, etc

Pseudo-triangles and rigidity

Cinderella system

Dynamic Systems and Animation. How do we construct animated mechanisms?

Exercises

Exercise 2.1: Design and program a system in which the user can construct the above ◊

End Exercises

§3. CSG Models

A CSG model is basically a system to construct an expression that represents a solid object. Let us briefly look at a "core" CSG model, that has primitive solids, affine transformations and regularized Boolean operations. Figure **??** illustrates a core CSG tree that represents a bracket.

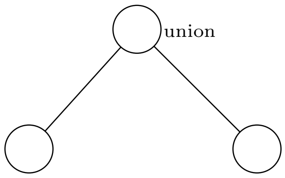

Figure 2: x

A CSG tree representing a Bracketbracketxp

The leaves of an expression are populated by **primitive solids**. Typical primitive solids are:

half-spaces, cube, prisms, sphere, cylinder, cone, torus.

These solids are **generic** in that the user can instantiate each with parameters, E.g. cube(1), cylinder(1, 3). These solids are located in standard positions. E.g., the cube is centered at the origin, with axes-parallel faces.

The simplest affine operators are translation, scaling and rotation. But more generally, we can descibe them by a non-singular transformation matrix. Thus, we may write "transform(S, scale(x, 3))" to change the scale in the x-direction by a factor of 3. They allow us to create important variations of the primitive solids. Using scaling, we can transform a cube into any rectangular block. Alternatively, we may want to introduce the "block primitive": block(1, 3, 4) instead of saying "transform(transform(cube(1), scale(y, 3)), scale(z, 4))".

Regularized Boolean Operations. The regularized Boolean operations are variations on the set-theoretic Boolean operations of union, intersection and complement:

$$\cup^*, \quad \cap^*, \quad \setminus^*.$$

Regularization simply ensures that the result is always a regularized solid. That is, we perform the usual operations, take the interior and then the closure. It is interesting to note that every core CSG expression can be put in a **canonical form**, comprising of a finite union of intersections of transformed primitive solids.

Interrogation We want to interrogate the CSG object. Most basic is: is a given point p on/in/out of an object? Called the **point/solid classification** problem. We outline the basic algorithm in the context of a core CSG system. We begin at the root of the expression.

- We pass p down to the children. Before passing p, if the current node is a rigid motion, then apply the inverse transformation to p before passing it on.
- At the leaves, we assume the classification problem can be solved directly.
- Now we pass the answers (in/out/on) up the tree. If the answers arrives an a Boolean node, we combine the answers of the children according to the table below.

$\cup^*$	in	on	out
in	in	in	in
on	in	on/in	on
out	in	on	out

$\cap^*$	in	on	out
in	in	on	out
on	on	on/out	out
out	out	out	out

The question is how to resolve the middle entry? E.g. consider two identical boxes A, B with A stacked on top of B. Let p be a point in the middle of the top face of B. Then p is on A and on B. But p is in $A \cup^* B$ and out of $A \cap^* B$. We cannot decide this except by a "neighborhood analysis".

We do as suggested by Hoffmann (note: we only work with polyhedral neighborhoods).

Let us illustrate this for one case: suppose p is on a face of A and a face of B and we want to classify a point relative to $A \cup^* B$. Then:
(a) If the 2 faces are NOT coplanar, then p is ON the union of A and B.
(b) If the two faces are coplanar, then if the outward normals are equal then p is ON the union of A, B, else, it is IN.

It is not hard to extend the above table for the complement operation.

Extensions. The above "core CSG" system can be extended in various ways. For instance, we need not assume that the nodes of a CSG expression represent solids – it is useful to introduce nonsolids (geometric objects of dimensions 0, 1 or 2), and other operations that can create solids out of them. For instance, if we have a planar polygon, we can sweep it along a finite trajectory to generate a cylider/prism like object.

§4. Theory of 2-Manifolds

The theory of 2-manifolds is very well-understood, and fairly intuitive. This can be found in any introductory book on algebraic topology. The computational aspects of this theory is addressed more recently. E.g., [3, **?**, **?**].

A **k-manifold** (without boundary) is a topological space that is locally homeomorphic to an open k-dimensional ball. We are interested in the case $k = 2$, so the k-dimensional ball is an open disk. Also, we simply say "manifold" to mean 2-manifold.

For instance, the surface of a three-dimensional ball is a manifold with boundary, usually called the 2-sphere and denoted S^2. Another interesting manifold without boundary is the torus, denoted T^2. However, the theory also non-orientable mobius strip, torus, Klein bottle

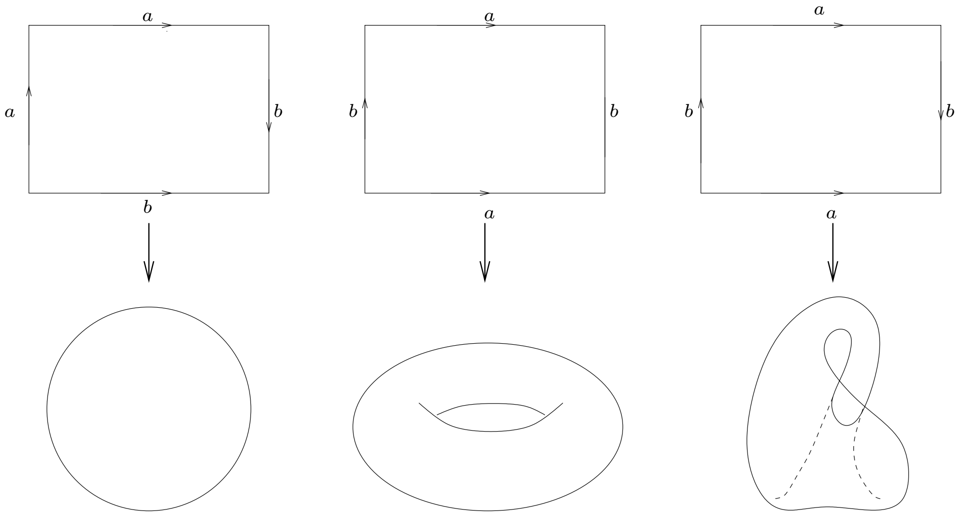

Figure 3: 2-Manifolds without boundary: S^2, T^2 and Klein bottle.

[BACKGROUND: topological space]

Classification of 2-manifolds: orientable/non-orientable.

NOTE: An **oriented** a planar polygon is a polygon together with an arbitrary orientation (cw or ccw). Each edge of the polygon has an induced direction. The induced direction is said to be **positive** with respect to the orientation of the polygon; the other is said to be **negative**

Polygonal or Plane Model of 2-manifolds:
a collection of oriented disjoint polygon with a total of $2n$ edges, labeled with the symbols a_i ($i = 1, \ldots, n$) where each a_i appears exactly twice and each appearance is associated with an arbitrary sign $\pm$.

The signs give each edge a direction (positive if the label is positive, negative else) with respect to the orientation of its polygon. We form a 2-manifold by identifying each pair of directed edges with the same label (ignoring their signs).

NOTE: it is easy to see that we can reduce to just one polygon. In the case of one polygon, we need not orient the polygon since the fundamental properties of the surface is unchanged if we flip every sign.

Möbius' Rule: the manifold is orientable iff such that for each pair of identified edges, one edge occurs is positive and the other is negative.

EXAMPLES:
(a) Torus: $ab\overline{a}\overline{b}$
(b) Klein bottle: aa

Brahana-Dehn-Heegaard Classification: Every 2-manifold is topologically equivalent to one of the following:

$$(a) a_1\overline{a_1}, \quad (b) \prod_{i=1}^{m} a_i b_i \overline{a_i}\overline{b_i}, \quad (c) \prod_{i=1}^{m} a_i a_i,$$

for some $m \geq 1$.

E.g., we can eliminate the pair $a\overline{a}$ in any sequence $\ldots a\overline{a}\ldots$.

Euler-Poincaré Formula: for an orientable solid (??)

$$v - e + f = 2(s - h)$$

where v is the number of vertices, e the number of edges, f the number of faces, s the number of *shells* (connected surfaces) and h the number of holes.

E.g., for a donut, $s = 1, h = 1$.

E.g., suppose we have a solid whose boundary has two shells: a cube inside a cube. Then $s = 2, h = 0$. Now $v = 16, e = 24, f = 12$ and so $v - e + f = 4$, as desired.

[NOTE: Actually $2(s - h)$ is equal to $h_0 - h_1 + h_2$ where h_i are the Betti Numbers. See Mantyla, p.45.]

Extension to manifolds with boundaries. This is not much different from the manifold case. In terms of the polygonal model, we now allow symbols that appears at most once (in which case, its sign is irrelevant).

Examples: the closed disc, Möbius strip.

Extension to non-simply connected faces. If there are r *rings* (i.e., loops that represent holes in a face), we can clearly remove the ring by introducing an edge to connect the ring to the outer bounding loop. Thus e should be replaced by $e + r$ to make the Euler-Poincaré formula hold. The modified Euler-Poincaré formula becomes:

$$v - e + f = 2(s - h) + r.$$

TOPICS:

- IDENTIFICATION along edges: Let P be a set of polygons. Let $E = \{e_i\}$ be some set of edges of the polygons, and each e_i is given an orientation. We identify parametrize these edges by the unit interval $[0, 1]$. Then points $e_i[t] = e_j[t]$ for all t and i, j. The neigborhoods of a point p in this quotient P/E is the union of the half-disks at this p.
- IDENTIFICATION at vertices. Similarly, if V are some set of vertices of P, we can form P/V.
- A **plane model** is a planar digraph (P, V, E) where P are polygons bounded by V and by E. The edges and vertices are labeled. Edges are oriented. A collection of edges (resp.vertices) are identified if they have the same label.
- EXAMPLE: (figure 41) plane model of a pyramid.
- A plane model is a **subdivision** (SHOULD BE CALLED **manifold subdivision**) if the following is true of its identification:

  (1) each edge is identified with exactly one other edge. (2) each identified point is locally a disk.
- Orientability. Relative to a orientation of a polygon, every edge orientation can be $+/-$.

  Möbius Rule: if the polygons are oriented in such a way that for every pair of identified edges, one is $+$ and the other $-$, then we say (P, V, E) is **orientable**.

- Euler Characteristics. If S is a sphere given by a plane model (P, V, E) then $v - e + f = 2$. We call $2 = \chi(S)$ the **Euler Characteristic** of S.
- Betti Numbers for any general closed orientable surface S:

  homology theory stats that $\chi = h_0 - h_1 + h_2$ where h_i are the Betti numbers.

  h_0 is the number of connected components.

  h_1 is the **connectivity number** of S – it is the largest number of closed curve that can be drawn on the surface without disconnecting it. E.g., for a Torus, $h_1 = 2$. In general, for a k-hold torus, $h_1 = 2k$.

  h_2, for us, is always h_0.

  GENERALLY:
  $$v - e + f = 2(s - h)$$
  where s is the number fo connected surfaces (shells) (so $s = h_0$) and $h = h_1/2$ is the **genus** of the surface.

  This h is the sum of the number of holes in each connected surface.
- Manifold with Boundary: $\chi(S) = 2(s - h) - b$ where b is the number of "blown holes" in the surface.
- DUALITY: All topological properties of a plane model are preserved in its dual. In particular, orientability is preserved.

§5. Nonmanifold Surfaces

This is a major extension. In terms of the plane model, we now require each symbol to appear an even number of times. If we also allow non-manifold with boundaries, then there is simply on restriction on the number of occurrences of a symbol. The book of Hoffmann takes this extension.

One the nice things about non-manifold is that it is closed under intersection. Let us assume that a solid need not be connected. Then the intersection to two manifold solids need not be a manifold solid. A simple example is:

FIGURE (generalize fig 14.8 of Mantyla)

Similar remarks hold if we restrict considerations to connected solids, provided we take the connected components of the intersection.

The neighborhood structure of each point or edge becomes more complicated: for an edge, this becomes a circular list of sectors. For vertices, this becomes a collection of 'planar patches'.

§6. Boundary polyhedral representation: manifold case

This is shortened to B-rep, and and contains (1) topological and (2) geometric information.

There are three types of basic objects: vertices, edges and faces.

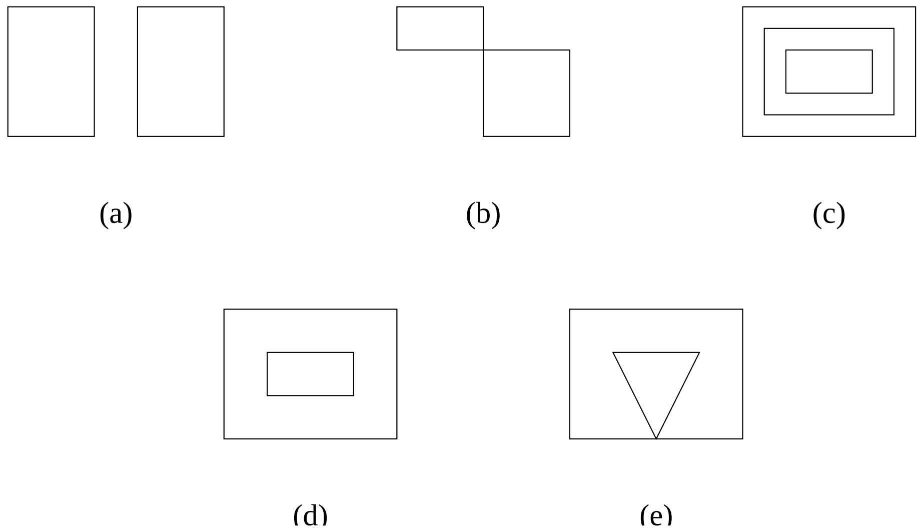

Figure 4: Possible Faces

What is a face? See figure 4 for some possibilities.

We now list the possibilities, with increasing generality.

Triangles: we need a triangulation algorithm.

Convex Polygons: need a convex decomposition algorithm.

Simple polygons: rather natural.

Simply-connected interior: a face is defined by one polygon whose edges can touch or even overlap. We "cuts" into the interior. If the exterior of the face is the "sea" then the cuts are "rivers". We allow "lakes" provided that they are connected by one river to the sea. Edges corresponding to rivers will be traversed twice. We do not allow an isthmus of land jutting into the sea. We will call such faces **INTERIOR TYPES**. There is a complementary notion called **EXTERIOR TYPES** whose exterior is simply connected – we allow isthmus of land but not rivers. Note a line-segment is accepted as an exterior type (it is an isthmus of land with no island to connect to!). See figure **??**.

Connected interior: we allow lakes and rivers but no islands inside lakes. This can be represented by a collection of loops: one loop is of **interior type** and there are zero or more pairwise-disjoint **exterior types** loops (these are called rings.

Planar regions: there is no requirement on connectivity, only all the regions are coplanar. This is not very convenient.

Hoffmann works with the connected interior case. The reason will be clear shortly. In some sense, the simply-connected is not really more restrictive than the connected interior case – we just have to add an edge that is traversed twice, a **canal**. It is not really too expensive. In displays, we may want to omit such bridges, but this is easy to detect using the fact that the fcw and $fccw$ pointers point to the same face (see below).

A **shell** is a connected component of the boundary of the solid. A solid can be defined be its forest of shells.

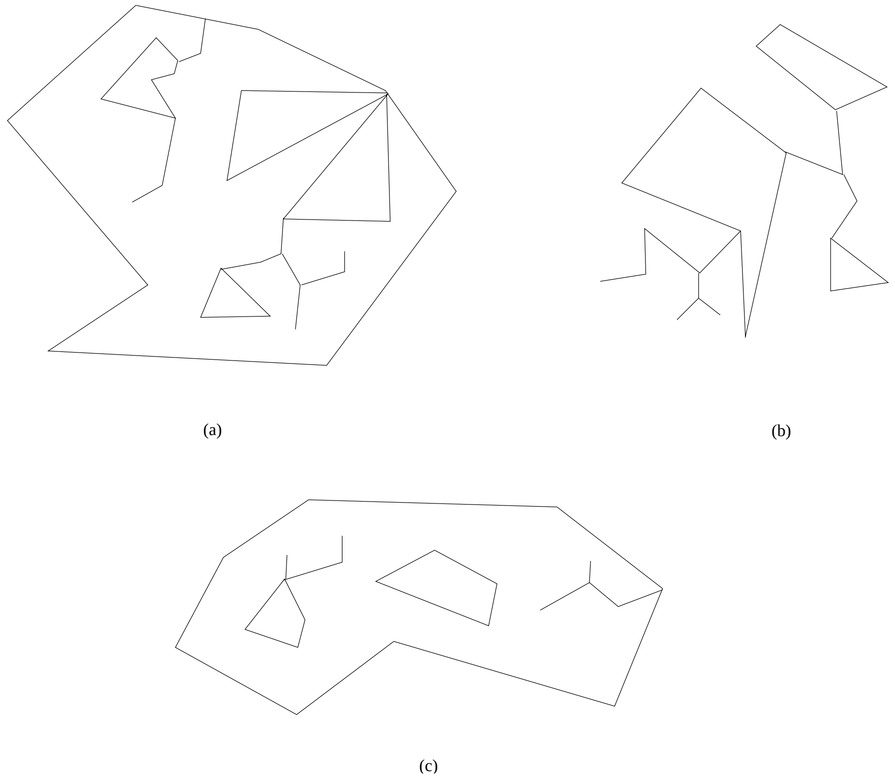

Figure 5: Simply-connected faces: interior and exterior types

Baumgart's winged-edge datastructure. Most of the information is associated with the edge. Hence this is sometimes called an **edge-based B-rep**. Every edge e is associated with the following data:

vtail, vhead: The leading (vhead) and trailing (vtail) vertices of an edge. This head/tail terminology fits nicely with the view of edges as arrows (winged lines). The direction from *vtail* to *vhead* gives the orientation of the edge.

ncw, nccw: The next cw and next ccw edges. This assumes that the incident edges at any vertex have a natural cw ordering.

ADDITIONAL INFORMATION The following are sometimes omitted.

eqn: The equation of the line. This can be implied from the end vertices.

lface, rface: The two adjacent faces. Note that we are assuming faces have been oriented according to Möbius' rule above.

pcw, pccw: The previous cw and previous ccw edges.

Each face f is associated with:

external-loop: an arbitrary edge in the external loop

internal-loops: a linked list of edges, one from each internal loop.

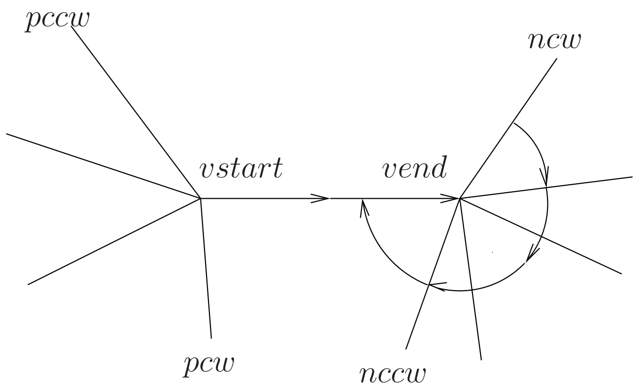

Figure 6: Baumgart's winged-edge structure

ADDITIONAL INFORMATION These are sometimes omitted.

eqn: plane equation, which can indicate the outward normal (i.e., points away from the solid). By convention, ccw is the orientation when viewing the face from outside.

Each vertex v is associated with:

x,y,z: coordinates.

Using the above structure, we can easily traverse the loops of a face.

Validity We shall define a winged structure **(topologically) valid** if it "topologically defines a solid" and the equation

$$v - e + f = 2(s - h) + r$$

holds.

Consistency Notice that the numerical values of the vertices are the only ones that must be represented explicitly. The others are implied when missing. We can now define what it means for the structure to be **consistent**: it means that
(i) the structure is valid,
(ii) each connected component of solid is bounded by the indicated faces,
(iii) There is no inter-penetration of faces.
(iv) All the local data are consistent.

APPARENTLY sugihara and Iri has proved that consistency can be reduced to classifying points w.r.t. edges and faces only?

§7. Euler Operators

These were introduced by Baumgart and Braid et al. They are defined to preserve the Euler-Poincaré formula. In principle, every and only valid solids can be constructed using these operators. Notice that we do not guarantee consistency (i.e., the numerical data may be inconsistent with the combinatorial structure).

Let us begin with an example:

MVFS, KVFS. MVFS stands for "make vertex, face, solid". It creates a solid (a half-space) with one face (an infinite plane) with one isolated vertex. Thus this face has one trivial loop, with no edges. MVFS is the inverse, for "kill vertex, face and solid".

This is the starting point for creating a winged structure. Let us also note the changes do not affect the validity of the Euler-Poincaré formula:

OPERATOR	v	e	f	h	r	s
MVFS:	+1	0	+1	0	0	+1
KVFS:	-1	0	-1	0	0	-1

The example illustrates our conventions for naming the operations:

Type	Symbol	Name or function
Operation	M	make
	K	kill
	S	split
	J	join
Basis Feature	V	vertex
	E	edge
	F	face
	S	solid
Complex Feature	H	hole
	R	ring

Note that S denotes split and solid – but these can be distinguished by their position in the name. We normally order the symbols in the order "Make Feature(s) [Kill Features".

MEV, KEV "Make edge, vertex" and "kill edge, vertex" are basic.

KEMR, MEKR

§8. Boundary polyhedral representation: non-manifold case

Hoffmann [p.39] argues that non-manifold models with finite surface area (but volume can be infinite!) are natural to represent in a modeling system. Figure 7 shows some non-manifold surfaces that we may want to model.

(1) Each vertex v: adjacent faces and edges are given.

Figure 7: Non-manifold objects

(2) Each edge e: the 2 incident vertices and all incident faces are given. Faces are cyclically ordered. Pairs of adjacent faces in this cyclic order that encloses volume are noted.

(3) Each face f: bounding vertices and edges are noted, in cyclic order.

LINE/FACE CLASSIFICATION. Basically we want to decide for every point p on a line L, its status (IN/ON/OUT) with respect to a face f. Of course, this can be made a finite classification by dividing L into a finite number of linesegments and we only classify the interior of these line segments (the boundary of the linesegments are always ON f).

E.G.

First we compute all the connected components of

$$\partial f \cap L = C_1 \cup C_2 \cup \cdots \cup C_m,$$

which can be ordered along L. Each component $C = C_i$ is either a point or an edge of f:
C IS A POINT: we must classify BOTH the left and right neighborhoods of C whether each lies IN f or OUT of f. If exactly one side of C is inside f, we call C a SIMPLE intersection, else a DOUBLE intersection. So a DOUBLE intersection is an isolated point that separates two OUT segments or two IN segments.

E.g., C is a regular vertex of f, we think of this as a SIMPLE point if the the two incident edges are on different side of L. BUT this can be on a river! Now, this slightly complicated.
C IS AN EDGE: THEN C is classified as ON. (This is the only type of ON segment). Again, we must classifications the the left and right neighborhoods of C. NOTE that C maybe a river edge, sea edge or a lake edge.

Containment test of 2 faces in the same plane. Suppose f, g are 2 faces in the same plane and their boundary are disjoint. Let us test to see if one contains the other, or they are disjoint: pick a point p on the boundary of f and q on the boundary of g and consider the line segment (p, q). This will intersect the boundary of f and g in a sequence of points $p < u_1 < u_2 < \cdots < u_k < q$ that we can order. There must be a first pair of consecutive points u_i, u_{i+1} where u_i is an intersection with ∂f and u_{i+1} is an intersection with ∂g. (NOTE: by assumption, we cannot have u_i representing intersection with f AND with g.) Then there are four cases for the segment $[u_i, u_{i+1}]$:

- In f, In g: This cannot occur if $\partial f, \partial g$ are disjoint.

- In f, Out g: $g \subseteq f$.
- Out f, In g: $f \subseteq g$.
- Out f, Out g: f and g are disjoint.

LINE/SOLID CLASSIFICATION. Let L be a line and S a solid. AGAIN, the basic goal is to output a partition of L into segments that are classified as IN/OUT/ON S.

If L intersects ∂S in some connected component C.
(1) CASE C is a point.
(1.1) If C is in the interior of a face, then C is a simple point and it is easy to decide which side of C belongs to $L \cap S$.
(1.2) If C is in the interior of an edge, then it is possible that both sides of C are inside or both outside $L \cap S$. This calls for neighborhood analysis. Also, L could partially lie inside a face of S.
(1.3) If C is a vertex of S, we can pick any plane containing L and which intersects some solid angle incident at C. This reduces the classification of the L-neighborhoods of C to the planar problem.

Containment test for 2 shells. Let A, B be disjoint shells. Again, we join a point $p \in \partial A$ with a point $q \in \partial B$.

§9. Polyhedral Intersection

Intersection of 2 shells. OUTLINE for intersecting shells A and B:
1. find all pairs of faces $f \in A$ and $g \in B$ such that $f \cap g$. If there are no intersecting pairs, then TEST CONTAINMENT and stop.
2. For each face $f \in A$, construct $f \cap B$, and keep the regularized part. (NOTE THE ASYMMETRY HERE)
3. Transfer the relevant line segments in step 2 to faces of B to (1) find out how the faces $g \in B$ must be subdivided and discarded (if they are outside the intersection).
4. Assemble all these faces into $A \cap^* B$.

STEP 1. An important heuristic: enclose each face by a bounding box and only consider those pairs $f \in A, g \in B$ whose bounding boxes intersect. If there are J such intersecting pairs of boxes, the algorithm runs in

$$O(n \log^2 n + J).$$

We return to this below.

Suppose there is no intersecting pairs of faces (f, g). Then we can use the containment test described above to decide if $A \cap^* B$ is empty, A or B.

STEP 2. This is conceptually not hard, but the details are involved because we get INTERMEDIATE (INCOMPLETE) edge and face subdivisions. Basically, we do the following for each intersecting pair (f, g):

2.1. Construct the points and line segments of their intersection.
2.1.1. Assume the planes of f and g are different and let L be the intersection of these 2 planes. We classify L with f and g, respectively. Then we "merge" the classification results from f and g. Thus:

```
OUT*X = X*OUT = OUT
IN*IN = IN
ON*IN = IN*ON = ON
ON*ON = ON/OUT
```

Again, the ON*ON case needs analysis.
2.1.2. Suppose the planes are equal. Then if their outward normals are different, then there can be no surface in the regularized result. If the same orientation, then we intersect 2 polygons f, g and the result is a face of $A \cap^* B$.

RESUME: at this point, we have placed segments and points on faces, placed points on edges, segmented edges, found adjacencies between points and segments.

2.2. Transfer the results to all adjacent faces of f and g.

2.3. Link up the intersection elements into complete faces and edge subdivisions.

After this, we know the surface of $A \cap^* B$ that lies on the surface of A and B (but not those that lie COMPLETELY INSIDE A or B).

E.G. Two intersecting boxes A, B where B intrudes through the interior of one face of A. After step 2, we found 5 of the 6 faces of the intersection. (See HOFFMANN, p.84).

§10. Bounding Box intersection

A good heuristic is to put each face in a bounding box and to a preliminary elimination of pairs whose bounding box are disjoint. We show how to do this in $O(J + n\log^2 n)$ where $J = O(n^2)$ is the number of pairwise intersecting boxes.

References

[1] C. M. Hoffmann. *Geometric and Solid Modeling: an Introduction.* Morgan Kaufmann Publishers, Inc., San Mateo, California 94403, 1989.

[2] M. Mäntyä. *An Introduction to Solid Modeling.* Computer Science Press, 1988.

[3] G. Vegter and C. K. Yap. Complexity of combinatorial surfaces. *ACM Symp. on Computational Geometry*, 6:102–111, 1990.