GEOMETRIC MODELING: A First Course

3. Representations

Representations were defined in the Introduction as symbol structures that correspond to (mathematical models of) physical entites. Our focus in this course is on mathematical models and computer representations that capture the shape of physical objects. This chapter discusses fundamental properties of representation schemes, and outlines the known approaches for representing geometric entities.

3.1 Representation Schemes

This section introduces basic notions and properties of representations through some simple examples. First we need a few definitions. A polygon is a 2-D region of the plane bounded by line segments called *edges.* Adjacent edges cannot be collinear. A *vertex* of a polygon is a point of the polygon's boundary where two adjacent edges meet. Two edges may intersect only at a common vertex. That is, self-intersecting figures are not considered polygons in this course. A *simple* polygon is a polygon without holes. Figure 3.1.1 shows examples.

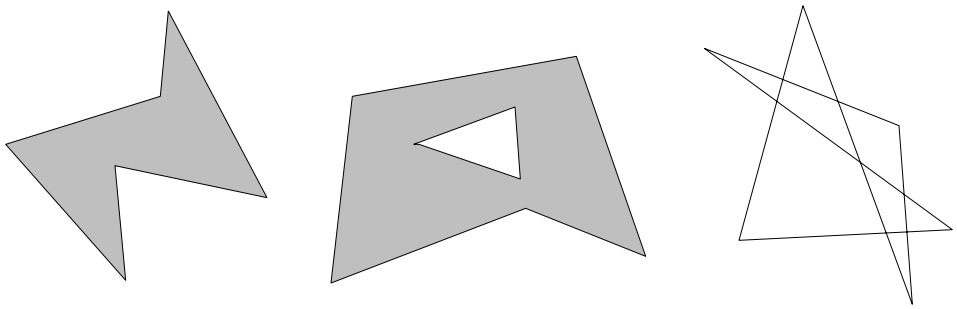

Figure 3.1.1 – A simple polygon with six vertices and six edges (left),
a polygon that is not simple because it has a hole (center),
and a figure that is not a polygon because it self-intersects (right).

A set X is *convex* if the line segment $\mathbf{pq}$ lies entirely within X for every pair of points $\mathbf{p}$, $\mathbf{q}$ of X. Figure 3.1.2 shows a set that is not convex because it does not contain entirely a line segment that connects two of the points of the set.

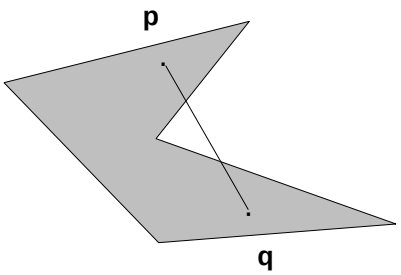

Figure 3.1.2 – A polygon that is not convex

Given any set X, one can always construct infinitely many other sets that enclose X and are convex. The smallest such set is called the *convex hull* of X. Figure 3.1.3 shows a set of discrete points in the plane and its convex hull. Note that the convex hull of a set of discrete points is a convex polygon whose vertices are a subset of the given points.

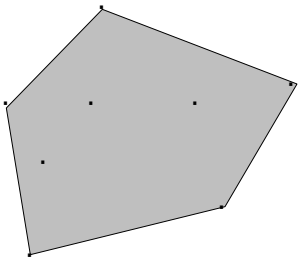

Figure 3.1.3 – A set of discrete points and its convex hull

Suppose now that we want to represent in a computer simple polygons. The set of objects to be represented—in our example, the set of all the simple polygons—is called the *domain* of the representation scheme. We propose a scheme, which we call Scheme 1, defined as follows.

1. For each polygon, construct the set of its vertices, *in arbitrary order*.
2. For each vertex, construct a pair of real numbers with the coordinates of the vertex in some agreed frame.
3. Make a list (i.e, a sequence of elements) containing all these pairs of reals.

Therefore the symbol structure used to represent a polygon in Scheme 1 is simply a list of pairs of real numbers:

$$\left(\left(x_1,y_1\right)\ \ \left(x_2,y_2\right)\ \ \ldots\ \ \left(x_n,y_n\right)\right).$$

This defines the format or *syntax* of the representation. Any representation obeying this syntax is called *syntactically correct*.

The meaning or *semantics* of a representation must also be defined, by means of a mathematical rule that associates the symbol structures to geometric entities. Each pair of reals corresponds to a point. Since points are the lowest-level geometric entities in this

representation, we refer to them as *primitives*. A specific point, also called a *primitive instance*, is represented by a pair of parameters, which are its coordinates.

The list of pairs is a *structured representation*, constructed with representations of primitive instances. Now we need to provide a rule for associating the list to a polygon. But this cannot be done unambiguously, as shown in Figure 3.1.4.

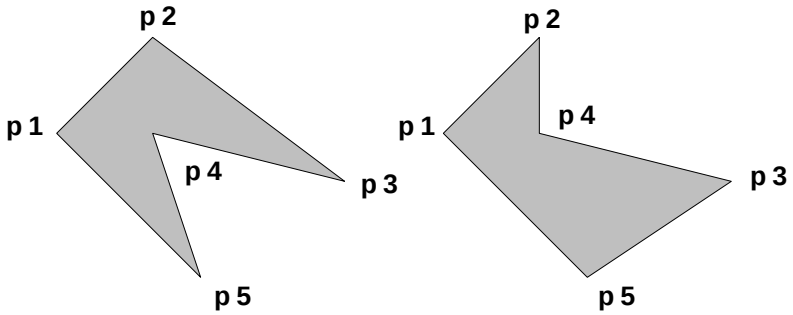

Figure 3.1.4 – Two distinct simple polygons with the same vertices.

The figure shows that a representation in this scheme may correspond to several distinct objects. We say that the scheme is *ambiguous* or *incomplete*. Representational ambiguity has unpleasant consequences. Suppose that we wanted to write an algorithm for computing the area of a polygon represented by the vertices shown in Figure 3.1.4. Should the algorithm return the area of the polygon on the left of the figure, or the area of the polygon on the right? We cannot write an algorithm to reliably compute the area, because we do not know which polygon is being represented. An incomplete representation does not contain enough information to designate a unique geometric object, and cannot support the automatic computation of all the geometric properties of the object.

Now let us define Scheme 2 by modifying slightly Scheme 1. We keep the same rule for constructing vertex-list representations, but restrict the domain to *convex* polygons. Because the convex hull of a set of points is uniquely defined, a vertex list corresponds to a single convex polygon. Therefore Scheme 2 is unambiguous.

Suppose that we construct a syntactically correct representation in Scheme 2, i.e. a list of real-number pairs, that corresponds to the points shown in Figure 3.1.3. This list violates the representation construction rules given above, because some of the points are not vertices of the convex-hull polygon, and only vertices of a polygon should appear in its representation. The representation is semantically incorrect, or *invalid*. The validity of representations in Scheme 2 is not easy to establish. In essence, it requires that we compute the convex hull of the set of points, and then verify that all the given points are vertices of the hull.

We consider again the entire domain of simple polygons, but change the rules for constructing representations, and define Scheme 3 as follows. We now require that the vertices be listed in consecutive order, as one follows the boundary of the polygon. The two polygons of Figure 3.1.4 have distinct representations in Scheme 3. The left polygon is represented by the list

$$(\mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_3 \quad \mathbf{p}_4 \quad \mathbf{p}_5),$$

whereas the representation for the right polygon is

$$(\mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_4 \quad \mathbf{p}_3 \quad \mathbf{p}_5).$$

A list of consecutive vertices is equivalent to an edge list. For example, the vertex list for the left polygon of Figure 3.1.4 is equivalent to

$$(\overline{\mathbf{p}_1\mathbf{p}_2} \quad \overline{\mathbf{p}_2\mathbf{p}_3} \quad \overline{\mathbf{p}_3\mathbf{p}_4} \quad \overline{\mathbf{p}_4\mathbf{p}_5} \quad \overline{\mathbf{p}_5\mathbf{p}_1}).$$

A planar polygon is completely determined by its edges, and therefore Scheme 3 is unambiguous.

These examples show that minor modifications in the domain or in the representation construction rules may change drastically the properties of a representation scheme.

We have seen several schemes for representing polygons in terms of their vertices. Are there other significantly different schemes? The answer is yes. Scheme 4 provides an example. Its domain is the set of convex polygons. But now we represent a polygon by a list of planar half spaces. A half space is the region of a plane that lies on one side of an infinite straight line. Figure 3.1.5 shows a half space primitive and its representation by a three-tuple (d, θ, s). Here d is the (perpendicular) distance between the origin and the line, or, equivalently, the length of the normal vector $\boldsymbol{v}$, shown in the figure. The second parameter is the angle the normal vector makes with the x axis. And s is a sign that indicates the side of the line on which the half space lies. A positive s implies that the "material" side is in the direction of $\boldsymbol{v}$; a negative sign denotes material on the opposite direction to $\boldsymbol{v}$.

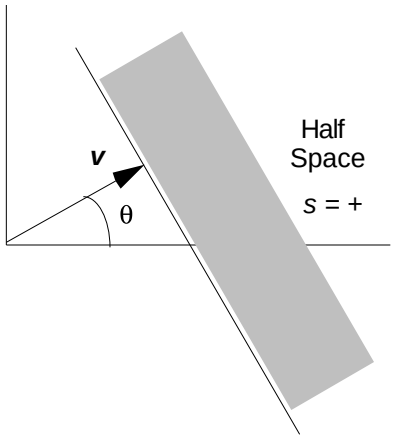

Figure 3.1.5 – Representation for a primitive half space.

The polygon that corresponds to a list of half spaces is simply the (set-theoretic) intersection of the half spaces. Figure 3.1.6 provides an example. The result is always a convex polygon because each half space is a convex set, and it can be shown that the intersection of convex sets also is convex. Scheme 4 is unambiguous, because the intersection defines a unique set.

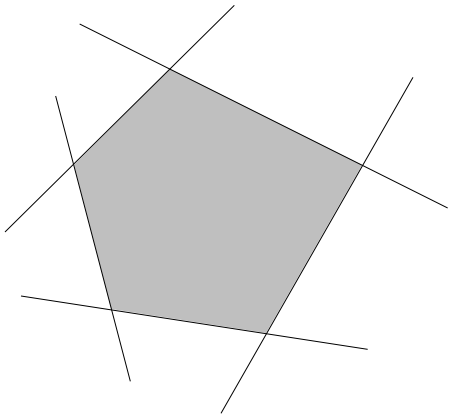

Figure 3.1.6 – A convex polygon defined as the intersection of five half spaces.

The examples presented above show that geometric objects may represented in many ways, and that the following are important properties of representations and schemes.

- *Domain* – Which objects can be represented in the scheme? The domain is the geometric coverage of a scheme.
- *Validity* – Does a representation correspond to (at least) one object in the domain? Computation on invalid data is meaningless, and often causes system crashes.
- *Non-ambiguity* – Does a valid representation correspond to only one object in the domain? Non-ambiguity, or completeness, is crucial for the automatic computation of properties of the represented objects.

The following additional properties also help to characterize representation schemes.

- *Uniqueness* – Does an object in the domain have a unique representation in the scheme? Uniqueness greatly facilitates testing objects for equality.
- *Conciseness* – How large are the representations? Representations in verbose schemes consume large amounts of memory, and are difficult to transmit rapidly in distributed environments.
- *Ease of construction* – How hard is it to construct a valid representation? Representations in verbose schemes with complex validity conditions are difficult to construct, especially by humans.
- *Suitability for applications* – Are there good application algorithms that operate on the representations of the scheme? Experience shows that representation schemes are not uniformly suitable for all applications. Many modeling systems use multiple representation schemes, and *convert* between them as needed, depending on the specific computations they support.

Representations often contain *redundant* data. For example, we could represent simple polygons in a Scheme 5 by listing the polygon's edges, with each edge represented by two vertices. This representation contains two copies of each vertex, and therefore is redundant. Redundancies are often introduced in representations to facilitate certain computations. They can be considered as a manifestation of the trade-off between storing and recomputing, which is ubiquitous in Computer Science. If we find that certain data are often needed by our algorithms, we may decide to attach these data to the representations rather than to recompute them as needed. Redundancy implies relationships or constraints

between the data stored, and complicates the analysis of the validity of a representation, because one has to ensure that the implied constraints are satisfied.

Storing several representations for each object, in different schemes, is an extreme form of redundancy. This is sometimes done in geometric modeling systems that support a variety of applications, to ensure that each application has access to the most suitable representation. Two unambiguous representations, in different schemes, are *consistent* if they correspond to the same object. In a multiple representation system it is crucial that representational consistency be maintained. Often this is achieved by invoking *representation conversion* algorithms. A conversion algorithm is called *consistent* if it guarantees that its output representation is consistent with its input representation. Consistency maintenance in systems that offer powerful object editing facilities is a major problem.

3.2 Methods for Representing Geometric Entities

Here we discuss known aproaches for constructing representation schemes for geometric objects. It is useful to classify them into several categories that correspond to the subsections below, although there is some overlap between categories.

3.2.1 Primitive Instancing

Every scheme has primitives, which must be instantiated to construct structured representations. Primitives may be low level, e.g. the points used to represent polygons in the previous section. But they also may be high level. For example, many modeling system have solid primitives such as blocks and cylinders.

A primitive is a parameterized geometric entity, typically represented by a tuple containing a type code (e.g. a string `'point'` or `'block'`) followed by several real numbers that correspond to specific parameter values. For example, a solid block aligned with the principal axes can be represented by the 4-tuple

```
('block', XSize, YSize, ZSize),
```

where the last three parameters are real numbers that define the dimensions of the block, as shown in Figure 3.2.1.1.

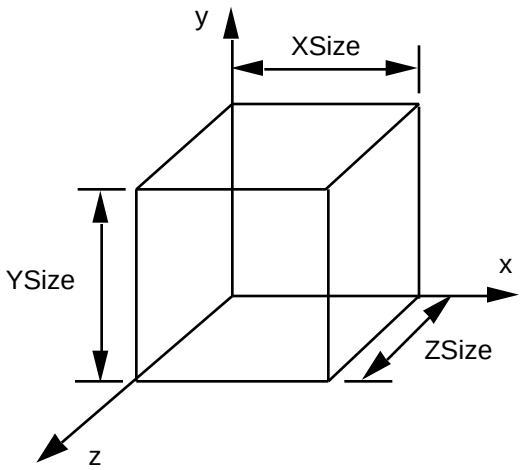

Figure 3.2.1.1 – A primitive block in standard pose.

To represent a block in a general pose one can add a fourth parameter, which is a transformation that takes the block from its standard pose, shown in Figure 3.2.1.1, to its actual pose.

Pure primitive instancing schemes, which have no structured representations, are not very attractive. They are akin to a language that has words but no sentences. They tend to have a small domain. In addition, each primitive type requires special-case algorithms for evaluating its properties. Primitive instancing is important primarily in the context of other schemes that not only instantiate primitives but also combine them into higher-level structures.

3.2.2 Spatial Decomposition

Here the idea is to partition space into regions called *cells*, and to enumerate those cells which are filled with material and therefore constitute the object being represented. Decompositions into regular, fixed-size cells are called *spatial enumerations*. In 2-D, the primitive square cells are sometimes called *pixels* (an abbreviation for “picture elements”), and in 3-D, cubical cells are called *voxels* (an abbreviation for “volume elements”). Regular decompositions usually represent only “staircase” approximations of the desired objects. For reasonable accuracy, the decompositions become very large.

Hierarchical decompositions with cells of varying sizes are smaller than their regular counterparts, because small cells are used only where required for an accurate approximation. 2-D *quadtrees* are used extensively in image processing. Figure 3.2.2.2 shows a simple 2-D polygon with sides parallel to the principal axes, and its quadtree.

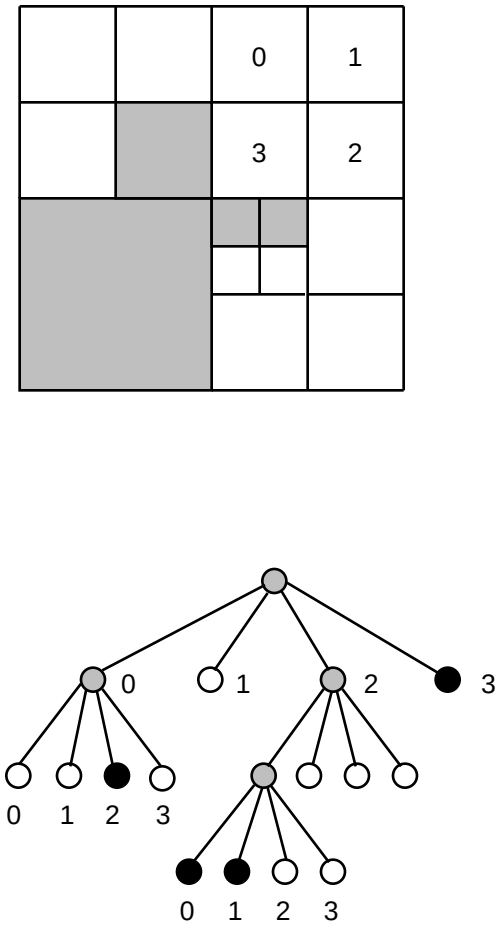

Figure 3.2.2.1 – A 2-D object and its quadtree.

In the figure, quadrants are numbered 0, 1, 2, 3 in clockwise order, as shown at the top. The quadtree has three types of nodes. Grey nodes correspond to cells that are neither completely full nor completely empty. Such cells must be subdivided. Black nodes are full and white nodes are empty. The root node of the quadtree corresponds to the entire space, i.e., to a square box that encloses the object. The quadtree may be constructed by the following (conceptual) procedure. If a cell is full or empty, mark it black or white, respectively; otherwise mark it grey, subdivide it, and recurse.

Quadtrees are the 2-D analogs of binary search trees, and can be used for spatial search. There is a large body of literature on algorithms for processing quadtrees [Samet 199?]. The 3-D analogs of quadtrees are called *octrees*. Both quadtrees and octrees are special cases of *k-d trees*, which are studied in the theoretical computational geometry literature. These are trees with *k* branches per grey node, and with cells of dimension *d*.

Issues of completeness and validity are trivial for most spatial decompositions. For example, a spatial enumeration is always valid and unambiguous.

Decompositions into cells with curved boundaries that follow closely the object's shape also are useful, for example for finite-element analysis. (Finite element analysis, or FEA, is a numerical technique for solving partial differential equations over complicated geometric domains by subdividing the domain into cells, and approximating the solution within each

cell by a polynomial function.) A set of cells that decomposes an object is called a *mesh* in the FEA field.

3.2.3 Constructive Methods

A constructive representation defines an object by a sequence of operations for constructing the object. The most common constructive representations are called CSG (for Constructive Solid Geometry), and use Boolean (set-theoretic) operations. The operation sequence is typically stored as a tree. Figure 3.2.3.1 shows a simple 2-D object and its CSG tree, built upon 2-D solid primitive rectangles and disks. The object in the figure is constructed as a union of two rectangles, from which a disk is subtracted.

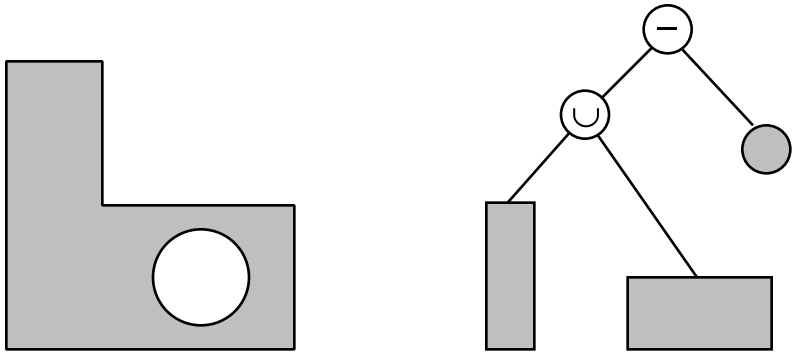

Figure 3.2.3.1 – CSG in 2-D.

The Boolean operations used in CSG are slightly modified set-theoretic union, intersection and difference. To see why the modifications are needed, consider the example of Figure 3.2.3.2. The L-shaped object *A* is intersected with the rectangle *B*. Although both *A* and *B* are 2-D solids, their set-theoretic intersection shown on the right is not. It has a dangling edge.

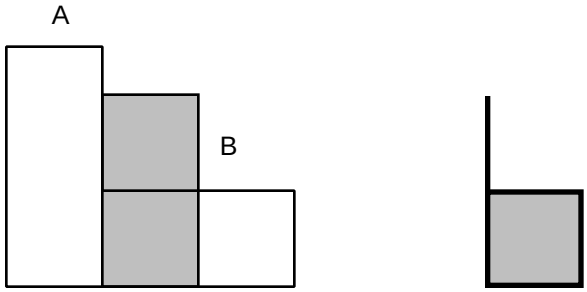

Figure 3.2.3.2 – The intersection of two solids need not be a solid.

Solidity is preserved if we define a *regularized* intersection operation as follows. First we perform the conventional intersection. Then we take its *interior* and find the *closure* of the interior. Interior and closure are defined rigorously in a branch of mathematics called *topology*. Here an intuitive understanding suffices. The sequence of operations which consists of interior followed by closure is called *regularization*. Figure 3.2.3.3 illustrates

the regularization procedure for the intersection of Figure 3.2.3.2. The dangling edge has no 2-D interior and therefore disappears, as shown at the center. The interior here is simply a square that does not include its borders. Closure adds the boundary to a set, and therefore produces the closed square shown on the right, which is a solid.

Figure 3.2.3.3 – Regularized intersection.

Regularized union, difference and complement are defined similarly. A set that equals the closure of its interior is called regular. Regular sets are used to model solids because they have no dangling edges or faces, or other non-solid characteristics. We see that regularized operations applied to regular sets produce other regular sets. In other words, regular sets are algebraically closed under regularized Boolean operations. This is important, because it implies that CSG representations for regular sets are always valid, since a sequence of regularized Boolean operations on regular primitives produces another regular set. Algebraic closure under the construction operations ensures that the result of an operation in a modeling system is valid, and can serve as input for further operations. Thus complex objects may be defined by successive operations on simpler ones.

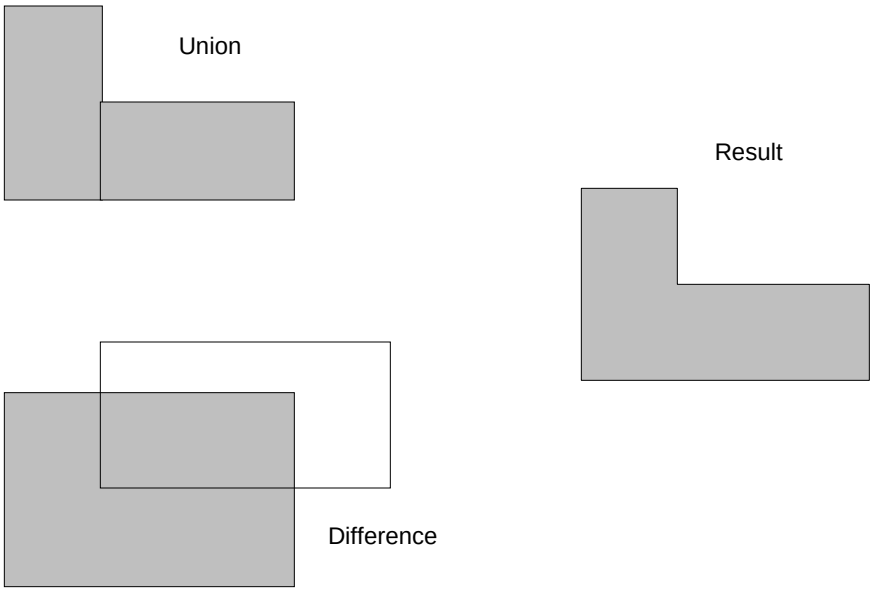

Figure 3.2.3.4 – Non-uniqueness of CSG

CSG representations are non-unique. For example, the L-shaped object on the right in Figure 3.2.3.4 may be defined as a union or as difference of two primitive rectangles, as shown on the left. Deciding whether two representations correspond to the same object—sometimes called the *same-object detection* problem—is not easy. In particular, the

null-object detection problem, which consists of checking if a representation corresponds to the null object, is non trivial. Note that to detect a collision or interference between two objects *A* and *B* it suffices to check if the intersection of *A* and *B* is null. In CSG it is very easy to represent the intersection, but hard to determine if it is empty.

3.2.4 Sweeping

Sweeping could be considered a constructive method, but we will treat it separately because it is sufficiently distinctive. The Minkowski sum of two sets *A* and *B*, denoted $A + B$, is the region swept by the set *B* when its reference point takes all possible positions within *A*. The orientation of *B* is fixed during the sweep. Figure 3.2.4.1 illustrates the definition. Here *A* is a rectangle, and *B* is a disk with its reference point at the center. The Minkowski sum shown on the right is a larger rectangle with rounded corners. The additional region generated by the sweep is highlighted by darker shading. The center of the figure shows the disk *B* at several of the positions it takes during the sweep.

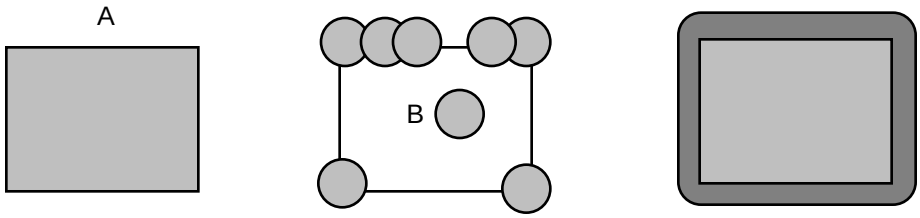

Figure 3.2.4.1 – Minkowski sum of a rectangle with a disk.

A Minkowski sum with a disk in 2-D or a ball in 3-D is also called a *solid offsetting* operation. Offsetting is useful for rounding and filletting objects, and for various other applications [Rossignac & Requicha 1984].

The Minkowski difference $A - B$ is defined as

$$A - B = c(cA + B) \ .$$

(Some texts use a different definition, which amounts to adding to *A* the reflection of *B* about the origin.) The Minkowski sum is a growing operation, and often called *dilation*, whereas the difference is a shrinking operation (because we grow the complement of *A*), often called *erosion.* These two operations are studied at length in a field called "mathematical morphology" [Matheron 1975, Serra 1982], and applied extensively in image processing [Haralick 19??].

When *A* is a 2-D planar set and *B* is a line segment normal to the plane of *A* their Minkowski sum is called a *translational sweep* or *extrusion.* (Oblique sweeps can also be defined.) Figure 3.2.4.2 shows and example. Extrusions are use extensively in mechanical CAD systems.

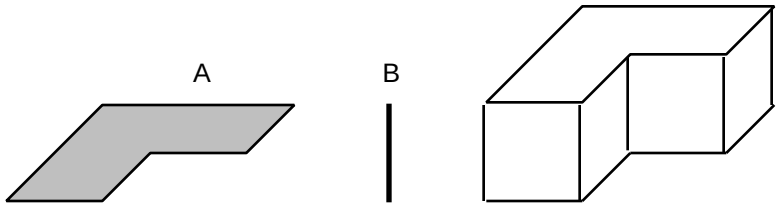

Figure 3.2.4.2 – An extrusion operation.

A Minkowski sum is not the most general form of sweep, because the moving set must maintain a fixed orientation, and cannot change its size or shape. An interesting generalization involves a planar set, or cross-section, that translates along a possibly curved trajectory, or *spine*, while undergoing changes. The resulting sets are (usually) 3-D solids called *generalized cylinders* or *generalized cones*, and are popular in computer vision [Ballard & Brown 19??].

A planar set may also be described by sweeping a disk of continuously-varying radius along a planar spine. The spine plus a function that defines the changes in radius constitute the *medial axis transform* or *skeleton* of the set. The notion extends to 3-D, by using solid balls instead of disks. Medial axis transforms have a variety of applications, from image processing [????] to finite element mesh generation [Srinivasan et al 19??].

3.2.5 Interpolation and Approximation

Interpolation and approximation methods typically are used to define higher-dimensional entities from lower-dimensional primitives. For example, a set of points may define a curve segment or a subset of a surface (often called a *patch* in the modeling jargon). We will discuss specific schemes later in this course, when we study curve and surface modeling. Here we present some fundamental notions through a simple example.

Suppose, for concreteness, that we want to represent a planar, 2nd. degree parametric polynomial curve whose generic point $\mathbf{p}(u)$ has coordinates

$$x(u) = au^2 + bu + c$$
$$y(u) = du^2 + eu + f$$

where u is the parameter, and the 6 coefficients a, b, c, d, e, f are initially unknown. If we require, for example, that

$$\mathbf{p}(0.0) = \mathbf{p}_1$$
$$\mathbf{p}(0.5) = \mathbf{p}_2$$
$$\mathbf{p}(1.0) = \mathbf{p}_3$$

the curve will pass through 3 given points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. We say that the curve *interpolates* the points. Writing the equations above in terms of point coordinates we obtain 6 linear equations in the 6 unknown coefficients, which, barring singularities, can be solved uniquely. Therefore the 3 points represent unambiguously the curve.

An interpolating curve passes through the given points but may have large oscillations between the points. An alternative approach is to require that the curve approximate the points, i.e., pass near the points without necessarily containing them. If we select a specific approximation method that ensures a unique result, the points will represent the curve unambiguously. Approximation methods tend to produce smoother, better-behaved curves than interpolation, and are widely used.

In a similar vein, points or curves may represent parametric surface or solid patches through interpolation and approximation schemes.

3.2.6 Boundary Methods

An object may be represented by its boundary plus the *host* space (e.g., an unbounded cylindrical surface) in which it lies. We already saw an example of a boundary representation scheme. It was Scheme 3 for representing polygons by their edges, discussed in Section 2.1. Knowledge of the boundary of a set does not always determine uniquely the set. However, for the objects we normally encounter in geometric modeling, boundary information does suffice to determine the "inside" unambiguously.

Boundary representations lower the dimensionality of the entities we must represent. For example, for a 3-D solid we need only represent its 2-D faces. And for each face, we need only represent its 1-D edges (plus host surface information). Thus, one can "recurse in dimension", eventually ending out with 0-D point primitives.

Additional data may be required for entities that lie in certain host spaces. For example, a closed contour of edges on a spherical surface cuts the sphere into two bounded subsets—see Figure 3.2.6.1. Therefore, the edge-set is an ambiguous representation. It must be augmented with *neighborhood* information that indicates which of the two possible faces is to be selected.

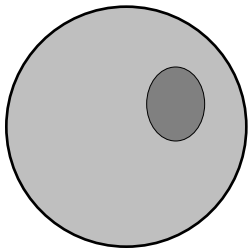

Figure 3.2.6.1 – Two faces lying in a spherical surface and having the same boundary.

The neighborhood of a point **p** with respect to a set S in Euclidean 3-space is the intersection of S with a solid ball centered on **p** and with a radius R, as R approaches zero. Figure 3.2.6.2 shows the neighborhoods of (i) a midpoint of an edge of a polygon and (ii) a vertex of the polygon. There are many ways of representing neighborhoods computationally. For polygons and other objects represented by edge-lists, neighborhood information is often encoded by ordering the edges, and orienting them consistently. For example, we may require that an observer moving along the edges in the specified direction see the material to his or her left. This convention is shown by the arrows in Figure 3.2.6.2.

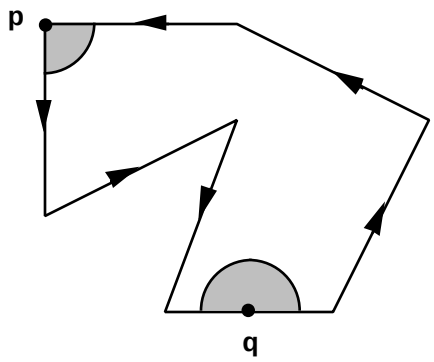

Figure 3.2.6.2 – Neighborhoods for points on a polygon's boundary.

The validity of boundary representations is a relatively complex issue. The validity conditions depend on the domain of objects to be represented, and on the details of the specific boundary representation scheme. Validity will be discussed later in this course, when we study the various geometric entities of interest such as curves or solids.

3.2.7 Hybrid Methods

It is possible to construct *hybrid* schemes, which combine several of the methods described earlier. For example, one can define extrusions by means of translational sweeps, and then combine the extruded objects by using Boolean operations, thus defining a sweep/constructive hybrid.

The main problem with hybrid schemes is the difficulty of writing algorithms for processing such representations. One approach is to implement separate algorithms to deal with each of the individual schemes, and then combine the results. An often simpler approach consists of converting a hybrid representation into a single-scheme representation, and then writing algorithms for this scheme. For example, the sweep/constructive hybrid could be converted to a BRep, and this representation used for developing the required algorithms.

3.3 Mathematical Underpinnings

3.3.1 Mathematical Models and Computational Representations

The framework introduced in Chapter 1 may be formalized as follows [Requicha 1980]. Let *M* be a space whose elements are abstract entities called *mathematical models*. In geometric computation, mathematical models usually are point sets in Euclidean space. As we will see later in this course, sharper characterizations of models exist for the various entities we will consider, e.g., curves or solids. *M* is called a *mathematical modeling space*.

The set of all syntactically correct representations constitutes another space called *representation space*. A *representation scheme* is formally defined as a mathematical relation *s* from *M* to *R*, as shown in Figure 3.3.1.1. The relation has a domain *D*, called the *domain* of the scheme, and a range *V*. A representation is *valid* if it belongs to *V*. A valid representation is both syntactically and semantically correct.

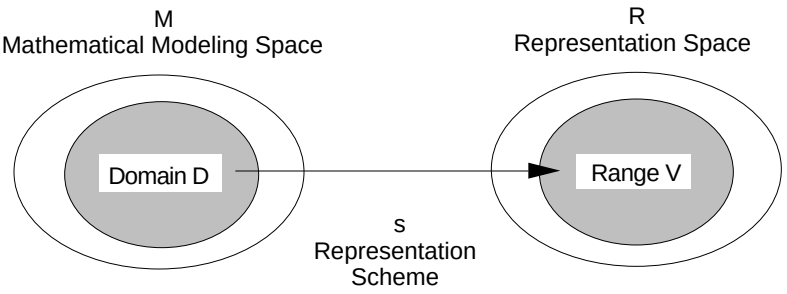

Figure 3.3.1.1 – Representation schemes as relations

A *representation r* in *V* is *unambiguous* or *complete* if it corresponds to a single model *m* in *M*, i.e., if the inverse image of *r* is a single-element set: $s^{-1}(r) = \{m\}$. It is unique if $s[s^{-1}(r)] = \{r\}$. A *representation scheme* is *unambigous* or *complete* if the inverse relation s^{-1} is a function. It is unique if *s* is a function.

3.3.2 General Topology and Regular Sets

The notions of interior, boundary, and so on, are formally defined in a branch of mathematics called point-set, or *general topology* [Mendelson 1975]. The main concepts of general topology that are of interest in geometric modeling are summarized below.

Let *W* be a set of abstract elements (the "world") and $d : W \times W \rightarrow$, where denotes the set of real numbers, a function called a *metric* or *distance*, that satisfies the following axioms, for all *x*, *y*, *z* of *W*.

1. $d(x,y) \geq 0$
2. $d(x,y) = 0 \Leftrightarrow x = y$
3. $d(x,y) = d(y,x)$
4. $d(x,z) \leq d(x,y) + d(y,z)$

A pair (*W*, *d*) with these properties is called a *metric space*. An important example of a metric space is the Euclidean space with its usual distance.

An *open ball* of radius $R > 0$ and centered at a point *x* of *W* is the set

$$B(x;R) = \{y \quad W : d(x,y) < R\}.$$

A subset *X* of a metric space *W* is *open* if it contains an open ball about each of its points. Let *W* be the real line with its usual distance; an open ball is an open interval. Similarly, an open ball in E^2 is a disk (without its bounding circle), and in E^3 it is a solid sphere (also without its bounding surface). Open sets in a metric space have the following properties:

1. The empty set Ø and the universe *W* are open.
2. The intersection of a finite number of open sets is open.
3. The union of any collection of open sets is open.

The intersection of an infinite collection of open sets need not be open. For example the intersection of the open intervals (–1,1), (–1/2, 1/2), (-1/3, 1/3), ..., is a set consisting of the single point {0} and therefore not open.

A *topological space* is a pair (W, T) where W is a set (the universe) and T is a collection of subsets of W, called *open sets*, that satisfy properties 1-3 above (by definition). T is called a *topology*. It is clear that any metric space with its open sets defined via open balls is also a topological space. Its (metrically-defined) open sets constitute the space's *natural topology*, which is sometimes also called the *topology induced by the metric*. However, the axiomatic definition of topological space given just above is more general, does not require any notion of distance, and includes spaces in which no metric can be defined. Usually, several topologies can be associated with a single set W, resulting in several different topological spaces.

A *neighborhood* $N(x)$ of a point x in a topological space (W,T) is any subset of W that contains an open set that contains x. For example, the closed interval [0,2] contains the open ball (0.9, 1.1) and therefore is a neighborhood of the point 1. Note that in the geometric modeling jargon "neighborhood" often has a more restricted meaning, explained in Section 3.2.6. Topology provides a notion of "nearness" that does not depend on distance. Thus, a point y is near a point x if y belongs to a neighborhood of x. General topology can be developed in terms of this notion of nearness—see [Henle 19??].

A subset X of a topological space (W, T) is *closed* if its complement cX is open. Note that closed sets are not the opposite of open sets. Some sets may be both closed and open, e.g., $\emptyset$ and W, whereas others may be neither closed nor open, e.g. the interval [0,1), which includes 0 but not 1.

A point x is a *limit point* of a subset X of a topological space (W, T) if each neighborhood of x contains at least another point of X different from x. Limit points of X need not belong to X. For example, 0 and 1 are limit points of the open interval (0,1) but do not belong to the interval.

The *closure* of a subset X of a topological space (W, T), denoted kX (this is not standard notation), is the union of X with all its limit points. It can be shown that kX is the smallest closed set that includes X, and that X is closed if and only if $X = kX$.

A point x of a topological space (W, T) is an interior point of a subset X of W if X is a neighborhood of x, i.e., if X contains an open set that contains x. The set of all interior points of X is called its *interior*, and denoted iX in this course. It can be shown that iX is the largest open set that is included in X. Thus, a set is always bracketed by an open and a closed set, which are the set's interior and closure.

A point x of a topological space (W, T) is a *boundary* point of a subset X of W if every neighborhood of x intersects both X and its complement cX. The boundary of X, denoted ∂X, is the set of all boundary points of X. It can be shown that ∂X is a closed set, and that the boundaries of a set and of its complement are the same, i.e., $\partial X = \partial cX$. Also, any subset X decomposes W into three pair-wise disjoint subsets:

$$W = iX \quad \partial X \quad icX \ .$$

It can be shown that the collection of sets of the form

$$X' = X \cap W' \ ,$$

where W' is a subset of a topological space (W, T) and X is an open set in the topology of (W, T), is a topology T' for W'. T' is called the *relative* or *induced* topology, and (W', T') is called a *topological subspace* of (W, T). The sets that constitute the relative topology T' are called *relatively open*, or open–W'. It is important to note that the definitions of closure, interior and boundary depend on the topology being considered. For example, let W be Euclidean 3-space and W' a 2-D Euclidean plane. A solid disk X lying in W' is closed both in W and W'; however, the relative boundary of X in the 2-D topology of W' is a circle, whereas its boundary in the 3-D topology of W is the disk itself.

The geometric modeling notion of neighborhood of a point $\mathbf{p}$ with respect to a set X corresponds to an open ball centered at $\mathbf{p}$ in the relative topology induced in X by the underlying Euclidean space.

A function f from a topological space (W, T) to another topological space (V, S) is *continuous* if, for every open set X of V, the inverse image $f^{-1}(X)$ is an open set of T. It is a *homeomorphism* if it is continuous and has a continuous inverse. One can show that a homeomorphism establishes a one-to-one correspondence between the elements of the two sets W and V, and also between the open sets of the topologies T and S. Two sets related by a homemorphism are called *homeomorphic* or *topologically equivalen*t. Properties that are invariant under homeomorphisms are called *topological properties*.

A subset X of Euclidean space is *bounded* if it can be enclosed by a ball of finite radius. It is *compact* if it is both closed and bounded. Boundedness is not a topological property, but compactness is.

A space is *connected* if it is not the union of two disjoint non-empty open sets. Connectedness also is a topological property. A disconnected set can always be decomposed into a union of disjoint, connected open sets called its *connected components*. A space is *path-connected* if every pair of points of the space can be joined by a curve that lies entirely within the space. Path-connectedness and connectedness are equivalent for the sets we normally consider in geometric modeling, but not for general sets.

A subset X of a topological space (W, T) is a closed regular set [Kuratowski & Mostowski 1976], or simply a *regular set*, it it equals the closure of its interior:

$$X = kiX \ .$$

The operation that consists of taking the closure of the interior of a set is called *regularization*, and denoted by rX. The regularized set operators are defined by regularization of their standard counterparts:

$$X \ {}^{*} Y = r(X \ Y)$$
$$X -^{*} Y = r(X - Y)$$
$$X \cap^{*} Y = r(X \cap Y)$$
$$c^{*} X = r(cX)$$

It can be shown [Kuratowski & Mostowski 1976] that the regular sets with the regularized set operations are a Boolean algebra, and therefore have the same algebraic properties as general sets with the standard set operators.