Chapter 6
Space

He bound him onto a swift camel and brought him into the desert. Three days they rode, and then the captor said, "O king of time and crown of the century! In Babylon you lured me into a labyrinth of brass cluttered with many stairways, doors, and walls; now the Almighty has brought it to pass that I show you mine, which has neither stairways to climb, nor doors to force, nor unending galleries to wear one down, nor walls to block one's way."

Jorge Luis Borges, "The Two Kings and their Two Labyrinths"

In the animal kingdom as a whole, spatial reasoning is probably the most common and basic form of intelligence. Nearly all animals have some control over their movements, and any but the simplest local criteria for choosing a motion requires some spatial knowledge of the environment. Flatworms can be taught to turn right or left; honey bees find their way around large areas and communicate their knowledge; many migratory creatures, such as salmon, navigate their ways across oceans. The common human habit of converting problems of all kinds into spatial terms (drawing a diagram or graph) suggests that, for people, spatial reasoning is a particularly powerful and accessible mode of cognition.

In everyday cognition, spatial reasoning serves three primary functions:

- *High-level vision*: The process of interpreting vision draws on a large body of knowledge about the shapes, positions, and motions of objects.
- *Physical reasoning*: The behavior of most physical systems depends strongly on their spatial layout. Changes to spatial layout form a

large part of the behavior of physical systems. General commonsense physical reasoning requires a rich geometric vocabulary and a strong spatial reasoner.

- *Route planning*: The problem of getting from one place to another, or of moving another object from one place to another, is critical for any mobile creature. A large part of this problem is the spatial reasoning involved in retrieving a path leading from the source to the destination. (It should be noted that there are other aspects to the route-planning problem. The hard part of planning to get to the top of Mount Everest or to get supplies to the eastern front is not the spatial reasoning.)

Other cognitive tasks draw on spatial reasoning to a lesser degree. Natural-language processing must use spatial reasoning in dealing with scene descriptions and route instructions. Spatial analogies are used for problem solving of all kinds.

A major part of all these applications, especially route planning, is the construction and maintenance of a *cognitive map*. A cognitive map is a knowledge structure that describes the spatial layout of an environment; it keeps track of what things are where. Thus, a cognitive map encodes the same type of information as a cartographical map. Typical kinds of information recorded might include "The red block is on the blue block," "Looking south from New Haven you can see Long Island," "Oklahoma has a long thin panhandle on the west," "Land elevation increases steadily going west through Nebraska," and "There is no salt at this end of the table." There are, however, two key differences between cartographical and cognitive maps. On the one hand, a cartographical map is constrained to represent most of its information pictorially, so that people can read it easily, while a cognitive map may use any data structure that supports efficient routines. On the other hand, a cognitive map must in general deal much more deeply with the problems of approximation and of partial knowledge. Someone who is drawing a cartographical map generally chooses a certain uniform level of accuracy and completeness for his map, gathers his information to that level, and makes sure that his map reflects that information. Occasionally, cartographical maps indicate uncertainty, by marking an area "Terra incognita" or by marking uncertainty tolerances on the positions of objects, but these are exceptions, rather than the general rule. A cognitive map, by contrast, must record information gathered catch-as-catch-can by a creature whose primary interest is probably not the gathering of spatial information, through a variety of modes: direct perception, particularly vision; natural language; and physical inference. Such information will tend to vary widely in its precision and its completeness; some regions will be known well,

others only sketchily. The design of a cognitive map must therefore reflect both the kinds of information to be retrieved from the map and the kinds of information available in constructing the map.

In many applications, a cognitive map must be combined with a temporal knowledge base to record facts about spatial relations over time and about motion, such as "Jane used to be only four feet tall," "There were no rabbits in Australia before 1800," "Eric crossed the border from Spain to France in October," "The bus is coming down the street at 10 miles an hour," and "I will reach the corner before the bus."

The state of the art in spatial reasoning, like that in quantitative reasoning, is at a rather different level than in most of the other domains we study in this book. Almost any particular sound commonsense inference in spatial reasoning can be expressed and proven as a theorem of Euclidean geometry using well-known mathematical terminology and axioms. (This does not apply to plausible spatial inference. Characterizing these is an open problem.) Thus, there are few ontological problems in spatial reasoning. The space of commonsense reasoning may almost always be taken to be Euclidean space,[1] and the sorts of entities needed to be standard geometric sorts such as points, vectors, mappings, and regions. Likewise, there are essentially no representational or axiomatic problems, in the sense of new concepts that need a formal definition, or axiomatic systems that need to be formulated or evaluated.

Nonetheless, choosing a language for a particular type of spatial reasoning can be trickier than it appears at first glance. Precisely because spatial representations appear so straightforward, a variety of ambiguities can be hidden under the rug in a representation, to make trouble at a later date. We give two examples:

1. *Shape*: Many cognitive maps approximate the complex shapes of real-world features in terms of an idealized simple geometry. In such cases, it is often possible to find two quite different legitimate representations for the same actual shape (Figure 6.1). It is therefore important to define the sense of approximation involved, so that sound rules for matching can be found. Consider, for example, the following approximation criteria for two-dimensional shapes:

    i. The approximation boundary is everywhere close to the real boundary. ("Close" and "small" in these criteria are to be interpreted relative to the diameter of the approximating shape.)

[1] There has been some interesting research on the use of nonstandard geometries in commonsense reasoning [Fleck 1987].

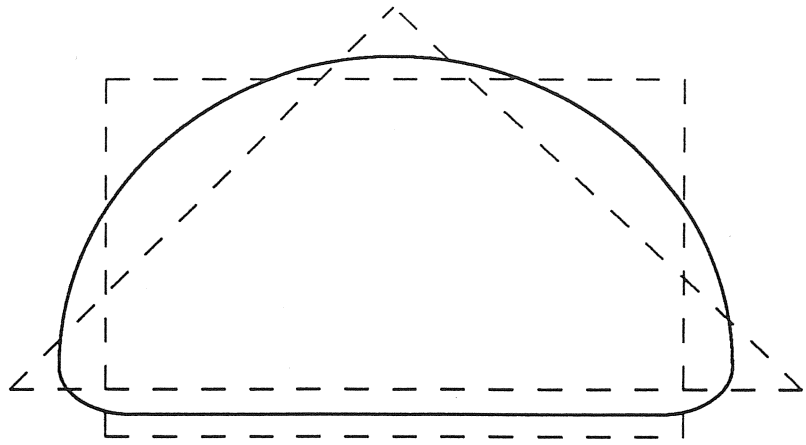


Figure 6.1 Two representations for a single shape

ii. The real boundary is everywhere close to the approximation.
iii. There is a continuous one-to-one function from the approximation interior onto the real interior that moves points only a small amount.
iv. The area of the symmetric difference between the two regions is small.
v. The tangent to the approximation is close to the tangent to the real boundary at some nearby point.

Different approximation criteria lead to different evaluations of correctness. For example, in Figure 6.2, example A satisfies criteria (i), (iv), and (v); B satisfies (ii); C satisfies (i), (ii), (iii), and (v); D satisfies (iv); E satisfies (i), (ii), (iii), and (iv); and F satisfies (i), (ii), (iv), and (v).

2. *Individuation of objects*: In a cognitive map that enumerates discrete objects, two questions arise about the significance of the enumeration. The first question is essentially the validity of a closed-world assumption on objects. Can it be assumed that any object that is in the area shown in the map and that can be detected by the sensors is represented in the map? If so, precisely how do we delimit the area within which the closed-world assumption applies, and the class of objects to which it applies? If the assumption does

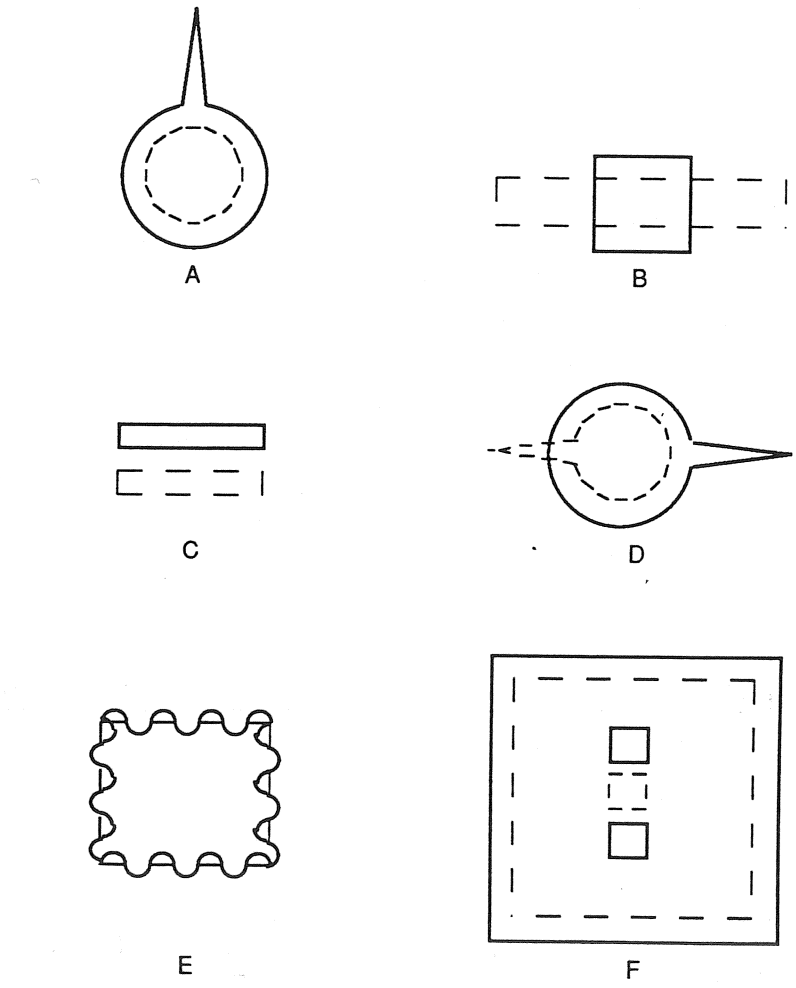


Figure 6.2 Criteria of approximation

not apply in a blanket way, is there any particular way to represent the fact that an area is clear of all objects, or of all objects of a given type? Is it ever possible to infer soundly from a map that a given area is clear of objects of a given type?

The second question is essentially the validity of the unique-names assumption. Can we assume that two separate object descriptions in the map correspond to two different real-world objects? If only separated parts of an object have been perceived, must their representations in the map be identified as being definitely the same object? or possibly the same object?

How the semantics of a map should decide these issues of shape approximation and object individuation depends on the type of information available and the use being made of the information. For example, if the smooth motion of an object is important in some application, then its surface properties will likewise be important, and the semantics of the representation must constrain them. Another example: if the robot often sees only parts of objects due to occlusion, the map should allow the separate representation of two separated parts of an object.

Even when all problems of ontology and representation have been addressed for a given class of spatial problems, it is generally very difficult to design a useful spatial inference module. The very richness of geometric theory makes it essentially hopeless to expect useful results from applying a general-purpose geometric theorem prover to arbitrarily constructed sentences in a geometric language. (Programs that do geometrical theorem proving such as [Gelernter 1963], [Wing and Arbab 1985], [Chou 1986] are not, in general, suitable for commonsense reasoning applications, such as cognitive map maintenance, particularly in view of the typically large size of the knowledge base involved.) It is generally necessary to restrict very tightly the kind of information allowed in a knowledge base and the kinds of inferences to be made, and then to devise special-purpose algorithms to perform these inferences. Even so restricted, many simple geometric problems are computationally intractable, and must be addressed by approximate algorithms or heuristics.

In view of the state of the field, this chapter will focus on illustrating specific representations and methods of inference rather than giving general principles. In Section 6.1, we will illustrate how a number of specific commonsense inferences can be represented and justified. In Section 6.2, we will look at the knowledge structures used in a number of actual spatial reasoning programs.

Table 6.1 Sorts of Geometric Entities

Sort	Notation	Example
Points	Bold face	$\mathbf{P}$
Lengths	Tildes	$\tilde{L}$
Directions	Hats	$\hat{D}$
Coordinate systems	Script letters	$\mathcal{C}$
Mappings	Greek capitals	Φ
Regions	Double letters in bold	$\mathbf{PP}$

We will use a number of geometric sorts in this chapter:

- *Points*: These are the fundamental components of our ontology. A point is an atomic location in space. The space as a whole is a set of points.
- *Measures*: Lengths, areas, and volumes. Each of these is a differential measure space. Area is length squared; volume is length cubed.
- *Directions*: A direction can be viewed as a point on the unit sphere.
- *Coordinate systems*: A right-handed orthogonal coordinate system consists of an origin (a point), a unit length, and a triple of mutually orthogonal axis directions. Let $\mathbf{P}$ be a point, and let $\mathcal{C}$ be a three-dimensional coordinate system, with origin $\mathbf{O}$, unit length $\tilde{L}$, and axis directions $\hat{E}_1$, $\hat{E}_2$, and $\hat{E}_3$. Then the function "coordinates(P,$\mathcal{C}$)" gives the triple of real numbers $< p_1, p_2, p_3 >$ satisfying the equation
$$\mathbf{P} = \mathbf{O} + p_1 \cdot \tilde{L} \cdot \hat{E}_1 + p_2 \cdot \tilde{L} \cdot \hat{E}_2 + p_3 \cdot \tilde{L} \cdot \hat{E}_3$$
- *Mappings*: A mapping is a function from the space to itself.
- *Regions*: A region is a set of points.

We distinguish the sort of variable and constant symbols by conventions of typography and diacritical marks as shown in Table 6.1.

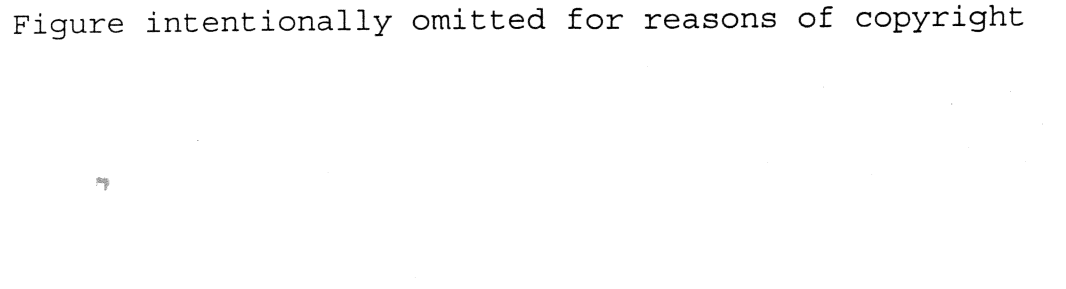


Figure 6.3 Example Scenario: Calvin and his socks

6.1 Spatial Inferences: Examples

In this section, we will show how a number of commonsense spatial inferences can be stated and justified in terms of Euclidean geometry. Our purpose is to illustrate both the range of geometric issues that arise even in relatively simple scenarios, and also the ontological, representational, and inferential adequacy of Euclidean geometry in dealing with these issues.

Our examples all relate to the scenario illustrated in Figure 6.3. Calvin is downstairs, barefoot; his socks and shoes are in a closed bureau drawer upstairs.

6.1.1 Set Operations on Regions

Given that Calvin is inside the living room, and that the living room is disjoint from the bedroom, infer that Calvin is not in the bedroom.

This involves only Boolean operations on point sets. Let **cc** be the region occupied by Calvin; let **ll** be the living room; let **bb** be the bedroom. Then the givens are represented $\mathbf{cc} \subset \mathbf{ll}$ (Calvin is in the living room) and $\mathbf{ll} \cap \mathbf{bb} = \emptyset$ (The living room is disjoint from the bedroom). From these, together with the implicit constraint $\mathbf{cc} \neq \emptyset$ (Calvin is nonempty), the conclusion $\mathbf{cc} \not\subset \mathbf{bb}$ (Calvin is not in the bedroom) follows directly.

6.1.2 Distance

Given that Calvin is less than 100 feet from the nearest point of the bedroom, that the bedroom is less than 40 feet in diameter, and that the socks are inside the bedroom, deduce that Calvin is less than 140 feet from the socks.

We express this inference using the Euclidean distance function "dist(**A**,**B**)," mapping two points **A** and **B** to a length. The distance function obeys the metric axioms:

dist(**A**,**A**) = 0.
dist(**A**,**B**) = dist(**B**,**A**).
dist(**A**,**B**) $\leq$ dist(**A**,**C**) + dist(**C**,**B**). (Triangle inequality)

We define the distance between two regions as the distance between their closest points.

$$\text{dist}(\mathbf{PP},\mathbf{QQ}) = \text{glb}\{\text{dist}(\mathbf{P},\mathbf{Q}) \mid \mathbf{P} \in \mathbf{PP}, \mathbf{Q} \in \mathbf{QQ}\}$$

We also introduce the function "diameter(**PP**)," mapping a region **PP** to the maximum distance between two points in **PP**.

$$\text{diameter}(\mathbf{PP},\mathbf{QQ}) = \text{lub}\{\ \text{dist}(\mathbf{P1},\mathbf{P2}) \mid \mathbf{P1},\mathbf{P2} \in \mathbf{PP}\ \}$$

Returning to our example, let **ss** be the region occupied by the socks. We can formalize the constraints as follows:

Calvin is less than 100 feet from the bedroom.	$\text{dist}(\mathbf{cc},\mathbf{bb}) \leq 100$
The bedroom is less than 40 feet in diameter.	$\text{diameter}(\mathbf{bb}) \leq 40$
The socks are in the bedroom.	$\mathbf{ss} \subset \mathbf{bb}$
(Implicit.) The socks are nonempty.	$\mathbf{ss} \neq \emptyset$

Applying the triangle inequality and the above definitions to these constraints, it follows directly that Calvin is less than 140 feet from the socks.

$$\text{dist}(\mathbf{cc},\mathbf{ss}) \leq 140$$

6.1.3 Relative Position

Consider the situation shown in Figure 6.4. Calvin is standing at the doorway of the bedroom. We model the information provided him by his sensors as constraints on the distance from objects to his visual reference point, the angle intercepted by objects at the reference point, and the relative orientations of objects. Calvin also knows from previous experience that the bureau is a 30- by 12-inch rectangle with one-inch sides, and that the socks form a two-inch-radius circular ball

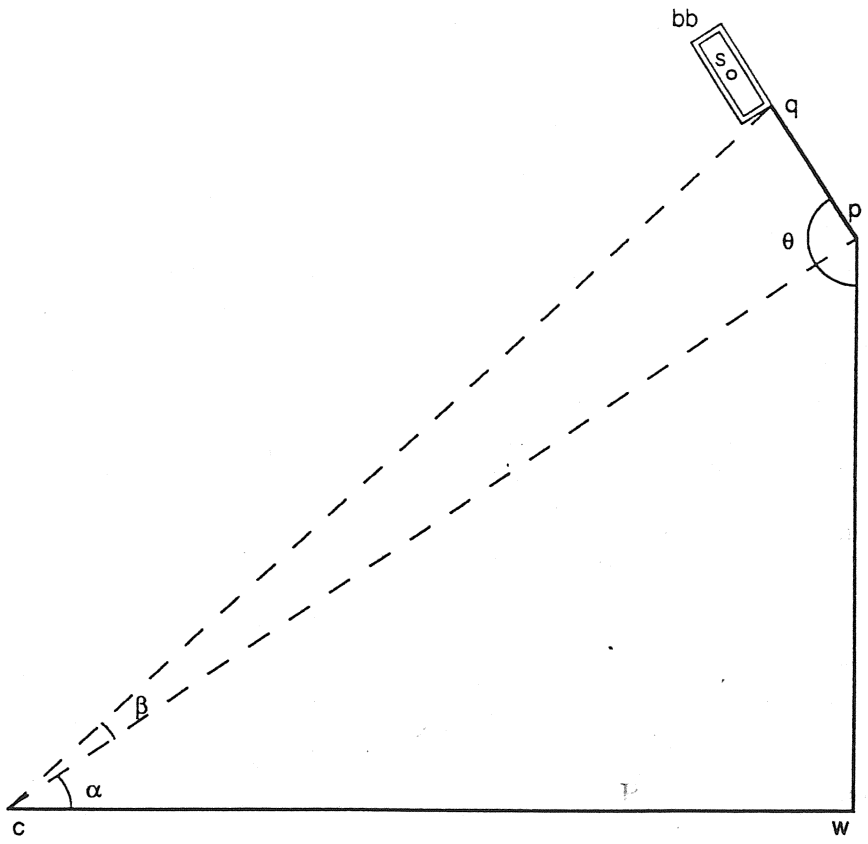


Figure 6.4 Geometry of bedroom

inside the bureau. (For this example, we will assume that the bureau drawer and socks are exactly these ideal shapes. We also restrict this problem to two dimensions. We will loosen these assumptions in Section 6.1.4.) What can Calvin deduce about the distance from his reference point to the socks?

Expressing this problem requires a number of standard geometric primitives to describe angles, coordinate systems, and shapes. These are illustrated in Table 6.2. We also need topological primitives to describe the relations between the socks and the bureau drawer: The socks are *inside* the drawer, but they do not *overlap* the material of

Table 6.2 Primitives for Relative Position Example

Angles:		
angle(**X,Y,Z**)	—	Function. Angle from ray **X–Y** to **Z–Y**.
colinear(**X,Y,Z**)	—	Predicate. **X,Y,Z** are on the same line in that order.
Coordinate systems:		
origin($\mathcal{C}$)	—	Function. Origin of coordinate system $\mathcal{C}$.
unit_length($\mathcal{C}$)	—	Function. Unit length of $\mathcal{C}$
x_axis($\mathcal{C}$)	—	Function. Positive x direction in $\mathcal{C}$.
direction(**X,Y**)	—	Function. Direction from **X** to **Y**.
Shapes:		
rectangle($\mathcal{C}, IX, IY$)	—	Function. The rectangular region of every point whose x coordinate in $\mathcal{C}$ is in IX and whose y coordinate is in IY.
circle(**O**,$\tilde{L}$)	—	Function. The circle of center **O** and radius $\tilde{L}$.
Topological:		
inside(**II,RR**)	—	Predicate. **II** is an inside of closed box **RR**.
is_inside(**AA,BB**)	—	Predicate. **AA** is inside **BB**. if **AA** is a subset of some inside of **BB**.
overlap_reg(**AA,BB**)	—	Predicate. Regions **AA** and **BB** overlap.

the drawer. We may define these as follows. A bounded region **RR** is a *closed box* if the complement of **RR** has more than one connected component. Necessarily, one of these components will be unbounded; the rest will be bounded. The unbounded component is the *outside* of **RR**; any bounded component is an *inside*. Two regions **AA** and **BB** overlap if their interiors intersect.

Table 6.2 enumerates the new primitives needed in this problem. Table 6.3 shows the constants in this problem. Table 6.4 shows the input constraints.

Table 6.3 Constants for Relative-Position Example

c,w,p,q,s	—	points as in Figure 6.4.
bb	—	the region occupied by the material of the bureau.
ss	—	the region occupied by the socks.
α	=	angle(**w,c,p**).
β	=	angle(**p,c,q**).
θ	=	angle(**q,p,w**).
$\tilde{d}$	=	dist(**w,c**).
$\tilde{ft}$	—	standard foot.
$\tilde{in}$	—	standard inch.
$\mathcal{B}$	—	frame of reference aligned with bureau with origin at **q**.

The calculation involves four parts. The first step is to determine that the bureau drawer has a unique inside, rectangle(β, [1, 29], [1, 11]). (β is a coordinate system with an origin **q**, a unit length of an inch, and an x axis oriented along the direction from **p** to **q**.) This can be established from four observations: (a) The region **bb** includes the four edges from $< 1, 1 >$ to $< 29, 1 >$ to $< 29, 11 >$ to $< 1, 11 >$ back to $< 1, 1 >$. Therefore any point in the polygon enclosed by these edges is either an element of **bb** or inside **bb**. The interior of this polygon is just rectangle(β, [1, 29], [1, 11]). (b) The region **bb** does not include any points inside rectangle(β, [1, 29], [1, 11]). (c) The region **bb** does not include any points not in rectangle(β, [0, 30], [0, 12]). Hence, any such point is in the outside of **bb**. (d) Any point that is not outside rectangle(β, [0, 30], [0, 12]) and not inside rectangle(β, [1, 29], [1, 11]) is in **bb**. The desired result follows directly from these facts.

The second inference in our calculation is that the center of the socks **s** is located somewhere in rectangle(β, [3, 27], [3, 9]). From the first step, we know that **s** is somewhere in rectangle(β, [1, 29], [1, 11]). From the shape description of the drawer and the socks, it is easily determined that, if the x coordinate of **s** were less than 3 or greater than 27, or if the y coordinate were less than 3 or greater than 9, then the socks and the drawer would overlap. Since they cannot overlap, the result follows.

The third step is to use the constraints on lengths and angles to fix bounds on the distance between the point **s** and the point **c**. We will do this using as reference a coordinate system $\mathcal{C}$ with origin **c**, unit

Table 6.4 Constraints in Relative-Positions Example

$\tilde{d}/\tilde{ft} \in [20, 25]$.
$\tilde{ft} = 12 \cdot \tilde{in}$.

angle(**p**,**w**,**c**) = $\pi/2$.
$\alpha \in [30°, 40°]$.
$\beta \in [5°, 10°]$.
$\theta \in [145°, 160°]$.

origin($\mathcal{B}$) = **q**.
unit_length($\mathcal{B}$) = $\tilde{in}$.
x_axis($\mathcal{B}$) = direction(**p**,**q**).

bb = rectangle($\mathcal{B}$, [0,30],[0,1]) ∪ rectangle($\mathcal{B}$, [0,1],[0,12]) ∪
rectangle($\mathcal{B}$, [0,30],[11,12]) ∪ rectangle($\mathcal{B}$, [29,30],[0,12]).
ss = circle(**s**,$2 \cdot \tilde{in}$).
is_inside(**ss**,**bb**).
¬overlap_reg(**ss**,**bb**).

length of a foot, and x axis aligned along the line **c**–**w**. For any point **a** and coordinate system $\mathcal{F}$ let $\mathbf{a}_{x,\mathcal{F}}$ be the x coordinate of **a** relative to $\mathcal{F}$ and $\mathbf{a}_{y,\mathcal{F}}$ be the y coordinate. The equations in Table 6.5 can then be derived from standard trigonometric rules.

The problem now is to find bounds on the expression in Table 6.5, given the constraints on the parameters. As it happens, in this example, the distance attains its maximum and minimum values over this region at extremes of the parameters. (Since these functions are nonlinear, this is not true in general.) The minimum value of the distance is 22.3 feet, attained when $\tilde{d} = 20$, $\alpha = 30°$, $\beta = 5°$, $\theta = 145°$, $\mathbf{s}_{x,\mathcal{B}} = 3$, and $\mathbf{s}_{y,\mathcal{B}} = 9$. The maximum value of the distance is 36.4 feet, attained when $\tilde{d} = 25$, $\alpha = 40°$, $\beta = 10°$, $\theta = 160°$, $\mathbf{s}_{x,\mathcal{B}} = 27$, and $\mathbf{s}_{y,\mathcal{B}} = 3$.

The final step of this calculation is to determine that the distance from Calvin to the socks is 2 inches less than the distance from Calvin to the center **s**. This follows directly from the definition of a circle as the locus of all points within the radius of the center. The final answer,

Table 6.5 Equations for Relative-Positions Example

$$\text{dist}(\mathbf{p},\mathbf{c}) = \frac{\tilde{d}}{\cos}(\alpha)$$

$$\text{angle}(\mathbf{c},\mathbf{p},\mathbf{q}) = \theta + \alpha - \pi/2$$

$$\text{angle}(\mathbf{c},\mathbf{q},\mathbf{p}) = 3\pi/2 - (\theta + \alpha + \beta)$$

$$\text{dist}(\mathbf{c},\mathbf{q}) = \text{dist}(\mathbf{p},\mathbf{c}) \cdot \sin(\text{angle}(\mathbf{c},\mathbf{p},\mathbf{q}))/\sin(\text{angle}(\mathbf{c},\mathbf{q},\mathbf{p})) =$$

$$\frac{\tilde{d}\cos(\theta + \alpha)}{\cos(\alpha)\cos(\theta + \alpha + \beta)}$$

$$\mathbf{q}_{x,\mathcal{C}} = \text{dist}(c, q)\cos(\alpha + \beta) = \frac{\tilde{d}\cos(\theta + \alpha)\cos(\alpha + \beta)}{\cos(\alpha)\cos(\theta + \alpha + \beta)}$$

$$\mathbf{q}_{y,\mathcal{C}} = \text{dist}(\mathbf{c},\mathbf{q})\sin(\alpha + \beta) = \frac{\tilde{d}\cos(\theta + \alpha)\sin(\alpha + \beta)}{\cos(\alpha)\cos(\theta + \alpha + \beta)}$$

$$\mathbf{s}_{x,\mathcal{C}} = \mathbf{q}_{x,\mathcal{C}} - \frac{\tilde{in}}{\tilde{ft}} \cdot (\mathbf{s}_{x,\mathcal{B}} \cdot \sin(\theta) - \mathbf{s}_{y,\mathcal{B}} \cdot \cos(\theta)) =$$

$$\frac{\tilde{d}\cos(\theta + \alpha)\cos(\alpha + \beta)}{\cos(\alpha)\cos(\theta + \alpha + \beta)} - (1/12 \cdot \mathbf{s}_{x,\mathcal{B}} \cdot \sin(\theta) - \mathbf{s}_{y,\mathcal{B}} \cdot \cos(\theta))$$

$$\mathbf{s}_{y,\mathcal{C}} = \mathbf{q}_{y,\mathcal{C}} - \frac{\tilde{in}}{\tilde{ft}} \cdot (\mathbf{s}_{x,\mathcal{B}} \cdot \cos(\theta) + \mathbf{s}_{y,\mathcal{B}} \cdot \sin(\theta)) =$$

$$\frac{\tilde{d}\cos(\theta + \alpha)\sin(\alpha + \beta)}{\cos(\alpha)\cos(\theta + \alpha + \beta)} - 1/12 \cdot (\mathbf{s}_{x,\mathcal{B}} \cdot \cos(\theta) + \mathbf{s}_{y,\mathcal{B}} \cdot \sin(\theta))$$

$$\text{dist}(\mathbf{c},\mathbf{s}) = \sqrt{\mathbf{s}_{x,\mathcal{C}}^2 + \mathbf{s}_{y,\mathcal{C}}^2} \cdot \tilde{ft}$$

therefore, is that the distance from the reference point **c** to the socks is between 22.1 and 36.2 feet.

6.1.4 Containment and Fitting

Assume that a shoe is 10 inches long, 4 inches wide, and 3 inches high, and that the bureau drawer is 12 inches deep, 8 inches high, 30 inches wide, and 1 inch thick. Infer that a shoe can fit in the drawer. In this example, by contrast with the previous one, we will work with three dimensions. Further, we will not assume that the shapes of the objects involved correspond exactly to an ideal; rather we give only approximate bounds of the regions that they fill.

First, we must give a precise interpretation to the two concepts of the inside of a bureau drawer, and of an object fitting in a space. "Inside" here means something different than in the previous section, since the drawer does not topologically separate its inside from its outside. In this case of a container with an opening on top, the meaning is something like the following: Region **RR** is inside container **CC** if one could put a horizontal "lid" onto **CC**, and **RR** would be enclosed by the lidded container.

Formally, we define a region **XX** to be an open box with opening **SS** and inside **II** if **SS** is a horizontal planar surface; **XX** $\cup$ **SS** is a closed box; and **II** is the inside of **XX** $\cup$ **SS** but not of **XX** by itself. (Figure 6.5). The formal definitions below use the predicates "planar(**PP**)" and "horizontal(**PP**)" (of a planar surface **PP**) with their standard interpretations.

open_box(**XX**,**PP**,**II**) $\Leftrightarrow$
[planar(**PP**) $\wedge$ horizontal(**PP**) $\wedge$ inside(**II**, **XX** $\cup$ **PP**)].

inside_open_box(**II**,**XX**) $\Leftrightarrow \exists_{PP}$ open_box(**XX**,**PP**,**II**).

Object O fits in region **RR** if there is a physically possible placement of O that is a subset of **RR**. In the case of a rigid object like a shoe, the physically possible placements of O are all congruent to one another (without reflection). We can therefore characterize the regions that can potentially be occupied by the shoe by characterizing the region the shoe occupies in some standard position, and then stating that it may occupy any congruent shape. The statement that the shoe fits inside the drawer is then interpreted as "There is some subset of the inside of the drawer that is congruent to the standard shape of the shoe." To represent this, we introduce the predicate "congruent(**AA**,**BB**)," with a narrowed interpretation that excludes reflections.

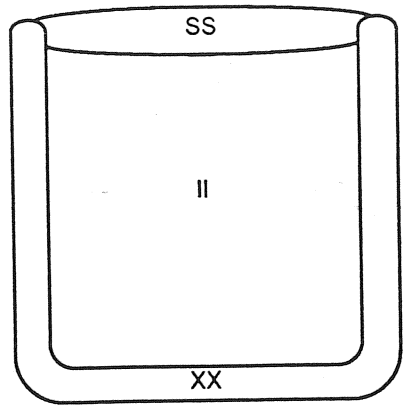


Figure 6.5 Inside of an open box

Next, we formalize the information given about the shoes and the drawer. Let **hh** and **dd** be the regions occupied by the shoes and the drawer in some arbitrary configuration. The statement about the dimensions of the shoe can be stated by asserting that the shoe lies within the box < [0,10],[0,4],[0,3] > in some coordinate frame with inch unit. Formally,

$$\exists_{\mathcal{C}}\ \text{unit\_length}(\mathcal{C}) = \tilde{in} \wedge \mathbf{hh} \subset \text{rectangle}(\mathcal{C}, [0,10], [0,4], [0,3])$$

The description of the drawer is more complicated, since we have to specify more than just its outer limits. There are several ways of formalizing the description above; our formalization will assert that the material of the drawer lies entirely within an inch on the inside of the horizontal sides and bottom of a shell of the specified dimensions. Specifically, let $\mathcal{D}$ be a reference frame for the drawer in a standard position, with opening on top. Let **ddb0** be a 30-by-12-by-8-inch box containing the shelf. Let **ddb1** be the sides and bottom of this box; that is, the boundary of the box minus the top. Then we assert that (i) the shelf is contained within **ddb0**; (ii) all of the shelf is within an inch of **ddb1**; (iii) every point of **ddb1** is within an inch of some point of the shelf; and (iv) the shelf is simply connected; that is, there are no holes through it (Figure 6.6). Introducing the functions "z_coor(**P**,$\mathcal{C}$)" and "simply_connected(**RR**)" with their natural interpretations, we can formalize the constraints as follows:

$$\text{unit\_length}(\mathcal{D}) = \tilde{in}.$$

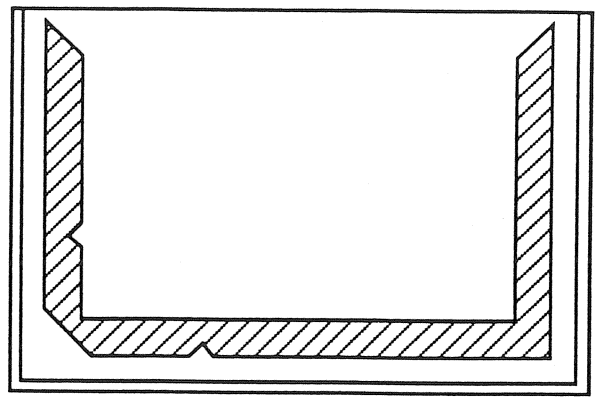


Figure 6.6 Shape of shelf

ddb0 = rectangle($\mathcal{D}$, [0,30], [0,12], [0,8]).
ddb1 = boundary(**ddb0**) − { **P** | z_coor(**P**, $\mathcal{D}$) = 8 }.
dd ⊂ **ddb0**.
$\forall_{\mathbf{P} \in \mathbf{dd}}$ dist(**P**,**ddb1**) < $i\tilde{n}$.
$\forall_{\mathbf{P} \in \mathbf{ddb1}}$ dist(**P**,**dd**) < $i\tilde{n}$.
simply_connected(**dd**).

It is easily shown from these constraints that the drawer is an open box with an inside of at least 28 by 10 by 7, and that, therefore the shoe fits inside the drawer.

$\exists_{II,HH1}$ inside_open_box(**II**, **dd**) ∧ **HH1** ⊂ **II** ∧ congruent(**HH1**,**hh**)

6.1.5 Abutment and Overlapping

When Calvin is wearing his socks, they cover the whole surface of his foot.[2] Therefore, his feet do not directly touch his shoes while he is wearing his socks.

In order to describe two objects meeting in space, we introduce the function "boundary(**AA**)", giving the boundary of region **AA**. We also use the function "abut(**AA,BB,FF**)", meaning that **AA** and **BB** abut on surface **FF**. Formally, abut(**AA,BB,FF**) holds if **AA** and **BB** do not overlap, but their boundaries have a non-null intersection, and **FF** is that intersection.

$$\begin{aligned}&\text{abut}(\mathbf{AA},\mathbf{BB},\mathbf{FF}) \Leftrightarrow \\ &[\neg\text{overlap\_reg}(\mathbf{AA},\mathbf{BB}) \wedge \\ &\text{boundary}(\mathbf{AA}) \cap \text{boundary}(\mathbf{BB}) = \mathbf{FF} \neq \emptyset]\end{aligned}$$

To prevent the sock from being a planar surface, which would allow the shoe to touch the foot right through the sock, we require that all the physical objects involved be *regular*. A regular region is "three-dimensional" throughout; it does not reduce to a two-dimensional surface or a one-dimensional curve anywhere. (Formally, we will define a region **RR** as normal if **RR** is closed, and every point in **RR** is on the boundary of an arbitrarily small connected open subset of **RR**. This is slightly stronger than the usual definition, that **RR** is equal to the closure of its interior.)

One might think that the constraint that the sock abuts the whole foot on its entire outer surfaces, together with the physical constraints that the shoe and sock cannot overlap and must be regular, would be sufficient to support the conclusion that the shoe cannot abut the foot. However, such a conclusion would not be valid, as Figure 6.7 illustrates.

Therefore, we must weaken the conclusion to state that the shoe cannot abut the foot in an extended region. (An alternative is to strengthen the premises to assert that the sock has no such thin points.) To express this, we introduce the predicate "two_d(**FF**)", which means that **FF** is the union of separated two-dimensional surfaces. (One possible formal definition is that **FF** has zero volume but nonzero area.) Our conclusion is now that the shoe does not abut the foot in any face.

Tables 6.6 and 6.7 show the formalization of the problem and its solution.

[2] This is not exactly realistic, since socks do not directly cover the inner surfaces of the toes. The assumption more accurately describes a skin-tight glove.

Table 6.6 Axioms for the Shoe-Sock Example

Geometric Axioms

- The boundary of a regular shape is two dimensional

  regular(**XX**) ⇒ two_d(boundary(**XX**))

- If the union of two regions **BB** and **CC** is two dimensional, then either **BB** or **CC** is two dimensional. (We assume that all sets involved are measurable.)

  two_d(**BB** ∪ **CC**) ⇒ [two_d(**BB**) ∨ two_d(**CC**)]

- Let **AA**, **BB** and **CC** be regular; let **BB** abut **AA** in **FF** and let **CC** abut **AA** in **GG**. If **FF** ∩ **GG** is two dimensional, then **BB** overlaps **CC**. (Figure 6.8)

  [regular(**AA**) ∧ regular(**BB**) ∧ regular(**CC**) ∧
  abut(**BB**,**AA**,**FF**) ∧ abut(**CC**,**AA**,**GG**) ∧ two_d(**FF** ∩ **GG**)]
  ⇒ overlap_reg(**BB**,**CC**)

Physical Axioms

- A solid object is regular.

  solid(X) ⇒ regular(value_in(S,place(X)))

- Two solid objects do not overlap.

  solid(X) ∧ solid(Y) ∧ $X \neq Y$ ⇒
  ¬overlap_reg(value_in(S,place(X)),value_in(S,place(Y)))

Table 6.7 Specifications for the Shoe-Sock Example

Constants

s1	—	A situation where Calvin is wearing his sock
foot	—	the foot
sock	—	the sock
shoe	—	the shoe
ankle	—	the ankle
ff1	—	the region occupied by the foot
ss1	—	the region occupied by the sock
hh1	—	the region occupied by the shoe
aa1	—	the region occupied by the ankle
oo1	—	the outer surface of the foot

Problem Specifications

- Definitions of the point sets.

  ff1 = value_in(s1,place(foot)).
  ss1 = value_in(s1,place(sock)).
  hh1 = value_in(s1,place(shoe)).
  aa1 = value_in(s1,place(ankle)).

- Unique names: The sock, shoe, foot, and ankle are distinct.

  distinct(sock,shoe,foot,ankle).

- The sock, shoe, foot, and ankle are solid objects.

  solid(sock) $\wedge$ solid(shoe) $\wedge$ solid(foot) $\wedge$ solid(ankle).

- The sock covers the entire outer surface of the foot.

  oo1 $\subset$ boundary(**ss1**).

- The boundary of the foot consists of its outer surface and the face where it abuts with the ankle.

  $\exists_{AA}$ abut(**ff1**,**aa1**,**AA**) $\wedge$ **AA** $\cup$ **oo1** = boundary(**ff1**).

Conclusion

- The shoe does not abut the foot in a face.

  abut(**hh1**,**ff1**,**FF**) $\Rightarrow$ $\neg$two_d(**FF**).

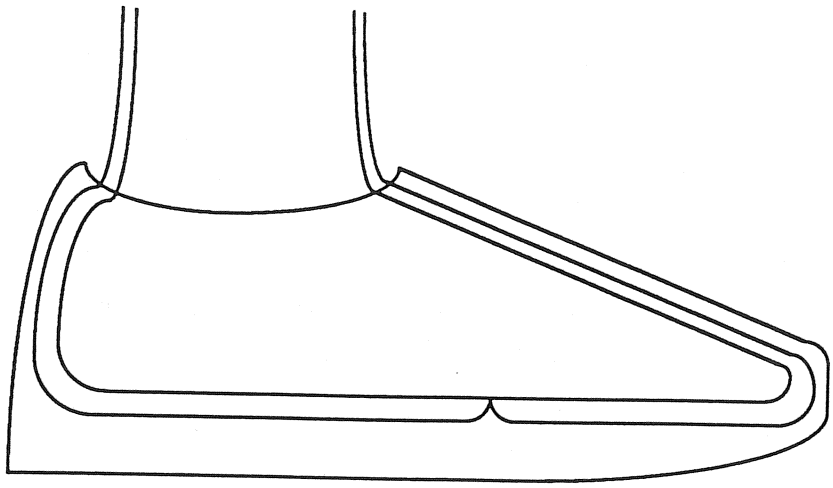


Figure 6.7 Contact through a thin point

6.1.6 Motion

(Note: This section depends on Chapter 5.) The only opening of the bedroom is the doorway. Infer that Calvin must go through the doorway to get from the living room to the bedroom.

The major concepts to be represented here are that of the opening of a region, and that of going through a region. The first is straightforward, given the concepts we have already developed. A region **OO** is an opening of the barrier **BB** into the interior region **II** iff **II** is an inside of the closed box **BB** ∪ **OO**, but **II** is not an inside of **BB** by itself.

$$\text{opening}(\mathbf{OO},\mathbf{BB},\mathbf{II}) \Leftrightarrow \text{inside}(\mathbf{II},\mathbf{BB} \cup \mathbf{OO}) \wedge \neg\text{inside}(\mathbf{II},\mathbf{BB})$$

The constraint in our problem is then that the doorway is the only opening into the bedroom through a solid barrier. Formally, let **ww** be the doorway. Our constraint is that there is a region **XX**, corresponding to the walls, ceiling, and floor of the bedroom, such that the doorway is an opening through **XX** into the bedroom.

$$\exists_{XX}\ \text{regular}(\mathbf{XX}) \wedge$$
$$\forall_S\ \mathbf{XX} \subset \text{value\_in}(S,\text{place}(\text{house})) \wedge \text{opening}(\mathbf{ww},\mathbf{XX},\mathbf{bb})$$

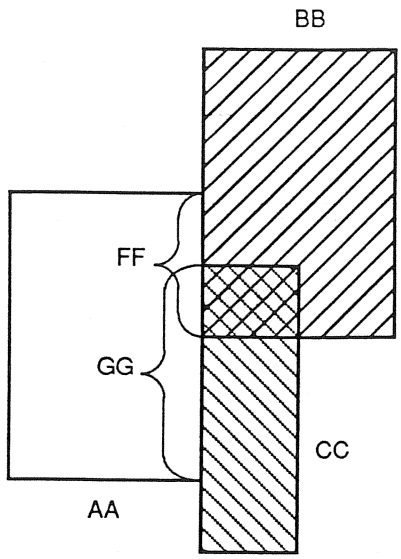


Figure 6.8 A common extended abutment implies an overlap

The second concept we need is that of an object going through a region. This is not (as far as I know) a standard geometrical term, and it is technically somewhat tricky. The formal definition is given in Appendix B of this chapter. Here, we will just introduce the event "goes_through(**FF**, **PP**)" meaning that fluent **FF** goes through region **PP**. For this example, we need the following axiom on this predicate: Let **FF** be a continuous fluent whose values are regular regions (such as "place(O)"); let **XX** be a regular region; and let **OO** be an opening through barrier **XX** into interior space **II**. If **FF** goes from outside **XX** into **II** without ever overlapping **XX**, then **FF** must go through **OO**.

[continuous(**FF**) $\wedge$ opening(**OO**,**XX**,**II**) $\wedge$ regular(**XX**) $\wedge$
value_in(start(I),**FF**) $\subset$ outside(**XX** $\cup$ **OO**) $\wedge$
value_in(end(I),**FF**) $\subset$ **II** $\wedge$
$\forall_{S \in I}$ regular(value_in(S,**FF**)) $\wedge$
$\neg$overlap_reg(value_in(S,**FF**),**XX**)] $\Rightarrow$
goes_through(**FF**,**OO**)

Now, given that Calvin and the house are two distinct solid objects, it can be shown that Calvin must go through the doorway to go from outside the room to inside it.

6.1.7 Surface Differential

Infer that Calvin cannot open the bureau drawer without touching the handle.

The inference here involves the following steps: Calvin can move the drawer out of the bureau only by exerting a net force on it with a positive component in the outward direction. The force exerted by Calvin on the drawer at any point of contact between him and the drawer is normal to the surface of the drawer and directed into the material of the drawer. Therefore, to push the drawer out of the bureau, Calvin must make contact with the drawer at some point where the surface normal into the material of the drawer is directed away from the bureau. Though there are a number of such areas on the surface of the drawer, the only such area that is accessible to Calvin's hand when the drawer is closed is the inner surface of the handle. Therefore, Calvin must touch the inner surface of the handle in order to open the drawer.

We will not give a detailed formalization of this inference. The bulk of the inference process is showing that the other surfaces of the drawer with the proper orientation are not accessible to Calvin, an inference which is involved but not deep. The major new primitive needed is the predicate "surf_norm(PP,X)," which gives the surface normal pointing outward from a solid region PP at point X. X must be on the boundary of PP, and must be a point where the boundary is smooth.

6.1.8 Other Predicates

Other situations require the use of still further types of spatial predicates. The curvature of two-dimensional curve PP at point X will be important in the characterization of a roller-coaster track discussed in Section 6.2.6. The volume of a three-dimensional region, denoted "volume(RR)," will be used in Section 7.4 to characterize the quantity of a liquid. A spatial description of a textured object such as a sieve, a sponge, a golf ball, or a fur coat must be in terms of the density of a certain kind of feature. Many natural objects such as clouds, mountain tops, and rivers have characteristic irregularities of shape, which can be represented in terms of fractals. A representation of a shape of a tree should express the fact that its upper part is tree structured in the graph-theory sense. (Excuse the nonpun.) The arboreal trunk is the graphical root; the arboreal branches are the graphical intermediate nodes, getting smaller as one gets further in the graph from

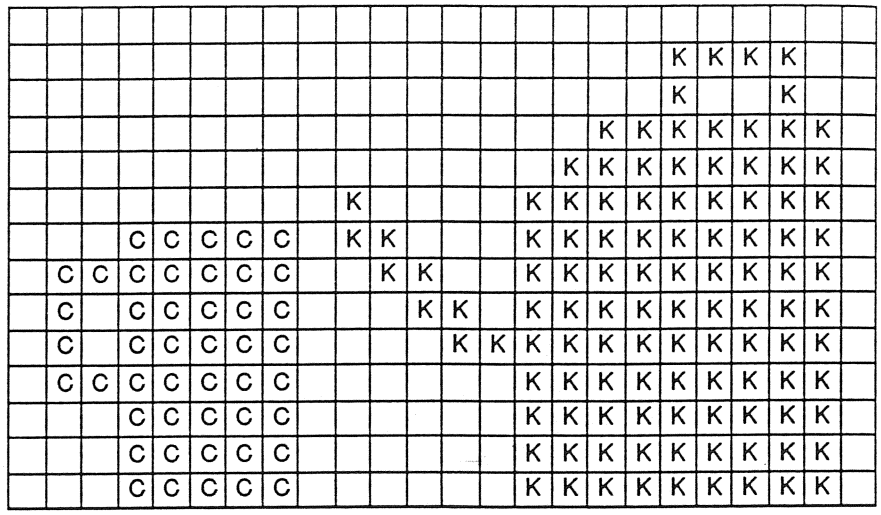


Figure 6.9 An occupancy array

the root, and connected to their parent nodes by biological branching; and the arboreal leaves are the graphical leaves.

6.2 Knowledge Structures

In this section we will discuss the spatial representations used in a number of actual AI programs. We have, in some cases, simplified or modified the actual representation for the sake of clarifying the presentation.

6.2.1 Occupancy

One of the simplest of all spatial representation schemes is the spatial occupancy array. In an occupancy array A, the element $A[I, J]$ corresponds to the square $[I, I+1] \times [J, J+1]$ in a fixed coordinate system. The array element holds a list of the objects that occupy its square. (Figure 6.9). (For efficiency of retrieval, it may be efficient to associate a list of the squares occupied with the representation of the object.)

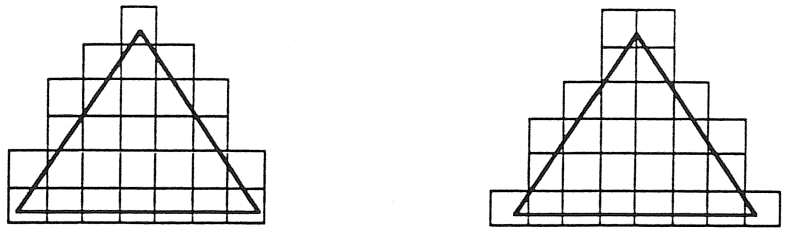


Figure 6.10 Matching shapes in a simple occupancy array

Occupancy arrays have a number of attractive features. They are easy for a programmer to visualize, to interpret, and to interface to graphics systems. They are similar to the pixel representation of an image that forms the input to most vision systems. Certain computations, such as checking intersection or performing translations are easily performed and easily adapted to parallel implementations. Furthermore, there is a large body of psychological experiments on visualization whose results have been interpreted as indicating that two-dimensional occupancy arrays are used in human spatial reasoning. However, as we shall see, in their simplest form, they are rather inflexible and inexpressive. In the following, we will consider a number of limitations and propose schemes for extending the representation to remove them; unfortunately, these extensions also tend to complicate the representation and make it harder to use.

Almost all actual implementations of occupancy arrays use the Boolean labeling indicated above; a square is either occupied by an object or unoccupied. They do not, in general, distinguish between fully and partially occupied squares. Rather, the functions that operate on the representation generally use an implicit assumption that cells on the boundary of the region may be fully or partially occupied, but those in the interior must be fully occupied. For example, a routine designed to match a given shape against a typical template would be tolerant of changes of one tile more or less on the exterior, since these can arise just as a result of the exact position of the shape with respect to the grid, but would be intolerant of changes in the interior. (Figure 6.10)

This assumption, however, means that the system is incapable of accurately representing shapes where interior squares are not wholly filled, or that if such shapes are represented, the functions may give incorrect answers. For example, if the system were asked whether the two shapes represented in Figure 6.11A intersect, it would an-

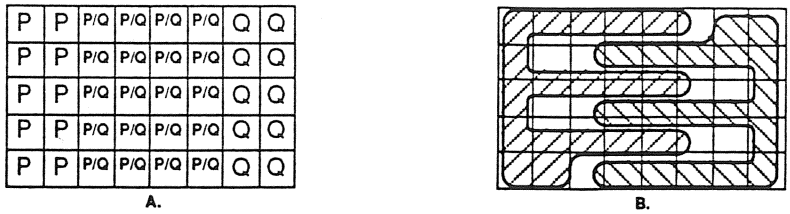


Figure 6.11 False evaluation of intersection

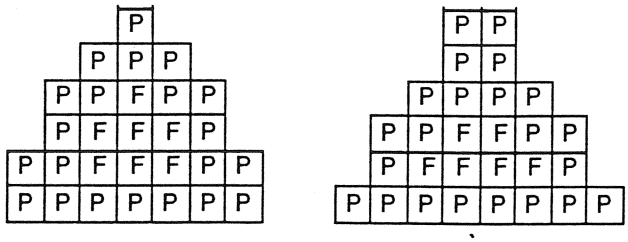


Figure 6.12 Occupancy array with partial/full occupancy

swer "Yes"; as Figure 6.11B shows, however, this need not be correct. We have therefore three options: (i) to exclude by fiat objects such as those in Figure 6.11B that have complex behavior on a scale comparable to the grid size; (ii) to refuse to accept queries such as "Do two objects intersect?" requiring that the query be posed instead as "Do the two objects come within the grid size of one another?"; and (iii) to distinguish between squares that are partially occupied from those that are fully occupied. Since the first two are necessarily very restrictive, we will here explore only the third. Figure 6.12 shows this representation applied to the sample figures of Figure 6.10.

The representation must be further extended in order to compute motions of an object. Suppose that we start with a shape such as is illustrated in Figure 6.13, and we specify that the object translates one-half a grid length to the right. How, then, shall we label squares 1 and 7 in the result? Square 1 may either be partially occupied or empty; 7 may be partially occupied or full. It therefore seems expedient to add these labels to the representation, and also the label "Don't know," meaning that the square may be full, partial, or empty.

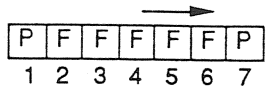


Figure 6.13 Loss of knowledge due to motion

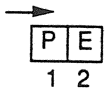
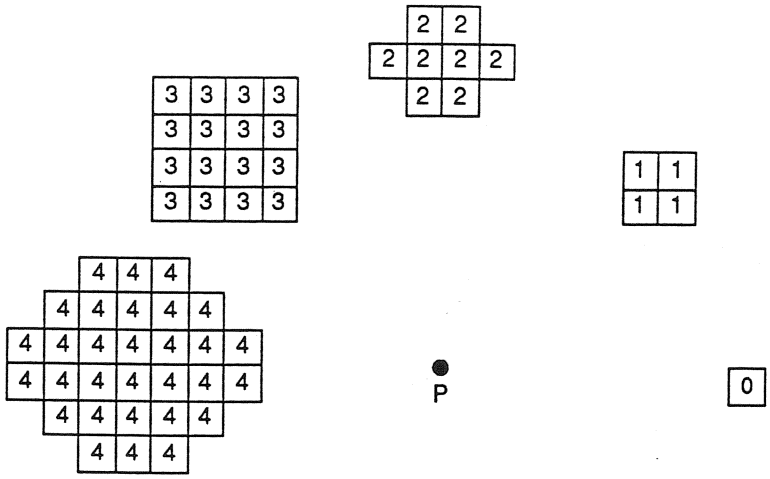


Figure 6.14 Disjunction in occupancy arrays

Our representation now has seven possible labels for each square and object; full, partial, empty, full or partial, partial or empty, or don't know. The operations on the representations, however, are qualitatively very much the same as in the initial simple representation. Functions like the intersection of two objects can still be computed cell by cell; all that is needed is a table specifying how to compute the combinations of labels on the cell. (Exercise 1.)

Motion raises another issue that is more troublesome. Suppose, as in Figure 6.14, we start with an object that partially occupies cell 1, and we move it half a cell length to the right. Now, cell 1 may be either empty or partial, and cell 2 may be either empty or partial. However, they cannot both be empty. Thus, we should add a disjunction, "Either cell 1 is partial or cell 2 is partial," to our individual cell labels. However, the cost of computing with such constraints is so large (problems become NP-hard) and the information expressed is in general so weak, that these are not generally worth including.

With or without the use of disjunctive constraints, the motion of a shape is almost always accompanied by a substantial loss of information. If a series of k rotations is applied incrementally to a single cell, in the end there will be on the order of k^2 cells that cannot be labeled "Empty." (Figure 6.15). One method to avoid this problem in computing the results of motions is to represent a region as a pair of an ideal, time-invariant shape, represented as an occupancy array, and a rigid mapping (Figure 6.16). The mapping corresponds to the change in position of the object; it is recomputed each time the object moves. This



Incremental rotations of 45° applied to square 0 around point **P**.

Figure 6.15 Rotation applied to a Occupancy Array

representation relies on the fact that the compositions of two rigid mappings is a rigid mapping and is easily computed. Appendix A of this chapter discusses effective representations for rigid mappings. To perform operations that require an occupancy-array representation of the actual spatial region occupied at a given time, the rigid mapping is applied to the time-invariant shape; since this is a single motion, the loss of information is as small as possible.

Another serious problem with occupancy arrays is that they can be very space inefficient. If the features of interest are spread out over a region of diameter D and you need to represent details of size δ, then you need an array of size D^2/δ^2 in two dimensions and of size D^3/δ^3 in three dimensions. This space requirement can often be overcome by the use of quad trees or oct trees. A quad tree is a tree of squares, structured by containment. Squares that lie entirely inside or entirely outside an object are leaves of the tree; they are not further decomposed. A square is decomposed to smaller squares only if the object partly overlaps the square (Figure 6.17). Analogously, an oct tree decomposes three-dimensional space into a hierarchy of labeled cubes.

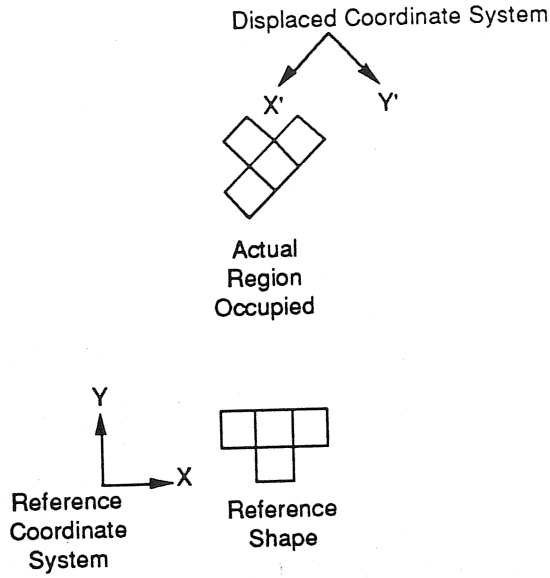
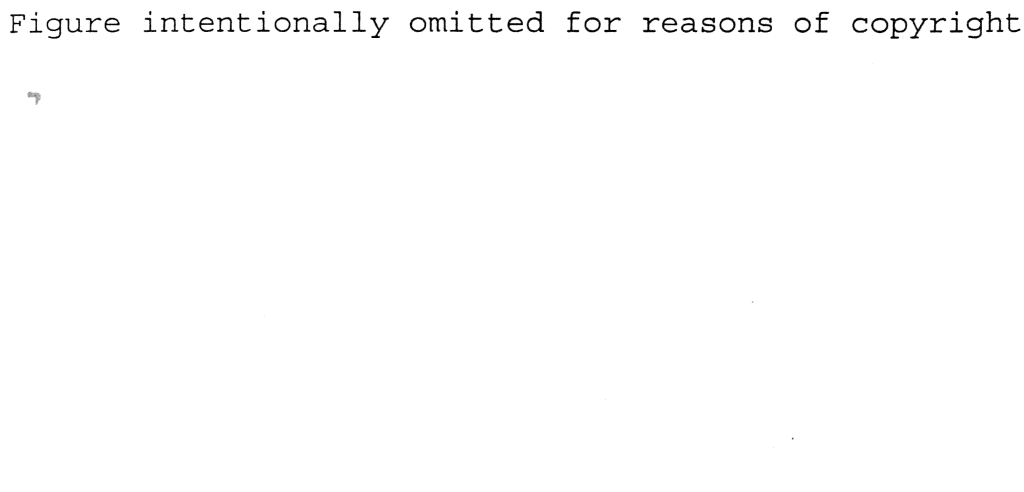


Figure 6.16 Occupancy Array and Rigid Mapping

The most serious limitation of occupancy arrays is their clumsiness at expressing partial knowledge. Frequently, an intelligent creature may know two separate areas in detail, while knowing only roughly the relative positions of the two objects. For example, suppose that Frederick knows the position of his socks in his bureau drawer to within three inches, and the position of his stapler on his office desk to within three inches, but knows the relative position of his office from his home only to within a quarter of a mile and knows nothing at all about their relative orientation. Then Frederick's knowledge cannot be represented in a single occupancy array or quad tree. If the positions of the office and the house are represented with the appropriate uncertainty, all knowledge about the position of the stapler will be washed out. Instead, we will need a number of occupancy arrays, one for each area and scale of information, and these arrays will be related by partial constraints. At this point, however, any query that involves combining information across different maps will involve primarily computing with the constraints connecting the maps, which vitiates most of the advantages of the representation.



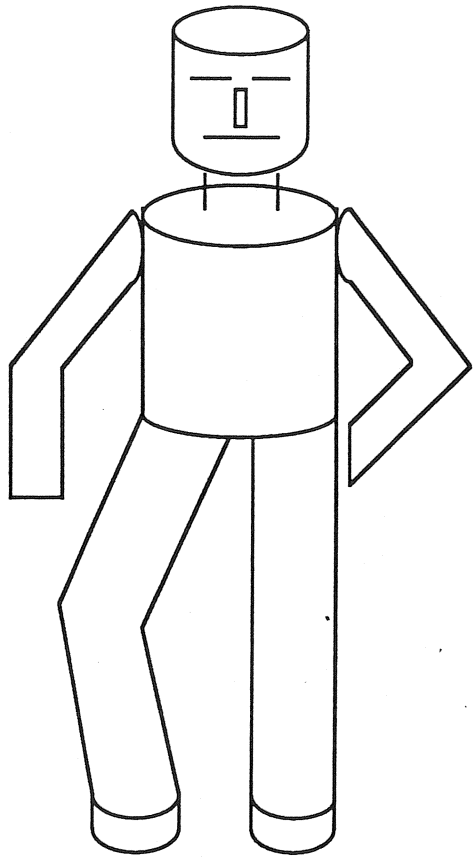
(From [Ballard and Brown 1982].)

Figure 6.17 Quad tree

In short, occupancy arrays can be an effective representation when the level of precision of information is more or less constant, and global knowledge is not much less precise than local information. In other circumstances they are inadequate, and must be augmented with other knowledge structures.

6.2.2 Constructive Solid Geometry

Constructive solid geometry (CSG) is another basic system for representing shapes. In CSG, a complex shape is described as the combination of shapes in different positions. For example, Figure 6.18 illus-

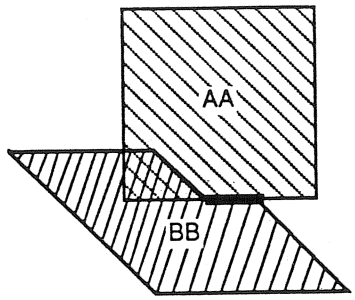


(From [Davis 1986a].)

Figure 6.18 A human as the union of cylinders

trates the representation of a human figure as the union of a collection of cylinders. CSG is a typical example of a *volumetric* representation.

A CSG language contains a number of composition operators, a vocabulary of primitive shapes, and a system for describing relative position. In simple CSG systems, the only composition operator may be forming the union of two regions. More complex systems may include the intersection and set difference operations as well. (Technically, since intersection and set difference can lead to regions that are not normal, the operations used in CSG are generally intersection or set



The intersection of the closed regions AA and BB includes both the cross-hatched region and the heavy-lined edge. Normalization drops the dangling edge.

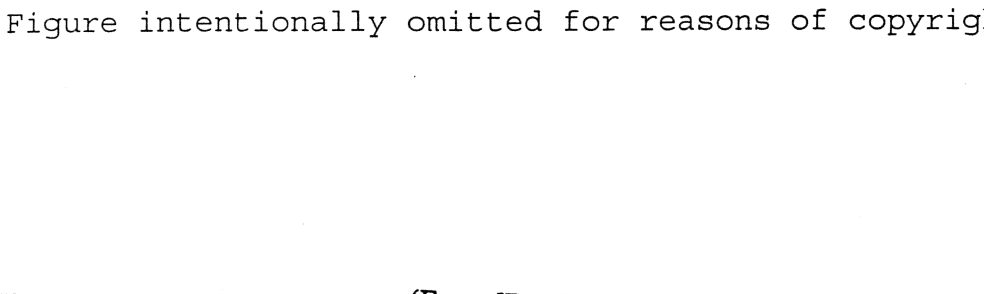
Figure 6.19 Normalized set operations

difference followed by a normalization procedure (Figure 6.19). The normalization of region **RR** is the closure of the interior of **RR**.)

What primitive shapes are used in a CSG system depends on which shapes occur naturally in the domain in question. In dealing with tools, the natural primitives are shapes that are easy to manufacture, such as prisms and cylinders. In dealing with natural objects, a popular choice as primitive is the *generalized cylinder,* the volume generated by sweeping a shape along a central axis. Each generalized cylinder is described in terms of its cross section, its axis, and its sweeping rule. Each of these may be chosen from a fixed vocabulary of primitive values with numerical parameters. For instance, the cross section may be a circle or a polygon; the axis may be a straight line or a circular arc; and the sweeping rule may be constant, linear contraction, or two different linear contractions in orthogonal directions. With each generalized cylinder is associated a coordinate system; the cylinder occupies a standard position and orientation within its own coordinate system. Figure 6.20 shows a variety of generalized cylinders.

The position of a given shape in a region can be described using the strategies for expressing rigid mappings given in Appendix A of this chapter.

The quantities used in the shape or position description may be represented as exact quantities or as constrained parameters. Such constraints can be used either to represent partial knowledge of the



(From [Brooks 1981]).

Figure 6.20 Generalized Cylinders

exact parameter values or to represent generic shapes that may vary within the constraints. Table 6.8 shows part of a representation for the human body of Figure 6.18.

A disadvantage of this method for relating the positions of regions is that the available information often relates to the positions of the boundaries of objects, rather than the positions of their reference points. For example, the distance or angle between the two closest boundaries of two objects is often perceptually evident or inferrable from physical behavior. Expressing this information about object boundaries in terms of constraints on rigid mappings may lead to ugly formulas that are difficult to use.

For a volumetric representation to be used as an approximation for shapes that do not conform exactly to its ideals (i.e., that are not perfect combinations of primitive shapes), a definition of the type and degree of approximation should be provided in the semantics of the representation. Otherwise, there is no way to specify that any given volumetric representation does not correspond to any arbitrary other shape; it might just be a very bad approximation. In practice, this sense of approximation is not generally specified, but is defined only operationally.

Table 6.8 CSG Generic Shape

human = head ∪ neck ∪ torso ∪
ru_arm (right upper) ∪ rl_arm ∪ lu_arm ∪ ll_arm ∪
right_thigh ∪ right_shin ∪ left_thigh ∪ left_shin.

/* cylinder($\tilde{L}, \tilde{R}, \mathcal{F}$) is the right circular cylinder of length $\tilde{L}$ and radius $\tilde{R}$, positioned so that the axis is aligned on the z axis of $\mathcal{F}$ and the center of the bottom face is the origin of $\mathcal{F}$ */

torso = cylinder(torso_height,torso_radius,torso_frame).
torso_height ∈ overall_height · [.25, .4].
torso_radius ∈ torso_height · [.25, .5].

ru_arm = cylinder(upper_arm_length, upper_arm_radius,
ru_arm_frame).

upper_arm_length ∈ torso_height · [.6,.9].
upper_arm_radius ∈ upper_arm_length · [.1,.2].

/* Position of arm relative to torso. The origin of the arm frame is placed slightly inside the torso, to avoid unsightly gaps when the arm bends. See Appendix A of this chapter for an explanation of the Z-Y-Z Euler angles */

coordinates(origin(ru_arm_frame), torso_frame) =
1/scale(torso_frame) · < torso_radius − upper_arm_radius, 0,
torso_height − upper_arm_radius >.

z_y_z_euler(ru_arm_frame, torso_frame) = $< \theta_1, \theta_2, \theta_3 >$.
$\theta_1 \in [0, \pi/4]$.
$\theta_2 \in [0, 4\pi/3]$.
$\theta_3 \in [-2\pi/3, 4\pi/3]$.

6.2.3 Boundary Representation

Information about the relations between the boundaries of objects, such as discussed above, can be expressed most easily in a boundary-based representation. In this section, we discuss a two-dimensional boundary representation used in the MERCATOR program [Davis 1986a]. MERCATOR is designed to simulate how a robot could put together a cognitive map in two dimensions by traveling through a region and using a vision system.

The most notable characteristic of the MERCATOR representation is its use of three well-defined types of partial knowledge:

- Partial knowledge of dimension, direction, and angle, both in the shapes of single objects and in the relative positions of objects.
- Partial knowledge of the shapes of objects. The fine detail of the shape of an object may be unknown if the object has been seen only from a considerable distance.
- Partial knowledge of the extent of an object. If an object has been partially occluded, then the knowledge base must record what has been seen without overconstraining the unseen part.

In order to deal with these three issues, MERCATOR uses a two-level representation. The first level consists of a grid of points connected by edges. Partial knowledge of dimensions and directions is expressed through constraints on the lengths and directions of the edges. MERCATOR uses a particularly simple form of constraint: interval bounds on the measurements of length and direction relative to a fixed coordinate system. The second level relates the shape of objects to the grid of edges. The known part of the boundary of an object is approximated by a chain of edges; the approximation is a tag indicating the maximum deviation of the real boundary from its approximation. If parts of an object have not been seen, then the chain of approximating edges does not form a closed cycle.

Formally, the second level of representation, relating the approximation of the boundary to its real position, is expressed using the predicate

$$\text{tolerance}(\mathbf{CC}, << \mathbf{EE1}, \tilde{D}_1 >, \ldots, < \mathbf{EEk}, \tilde{D}_k >>),$$

The predicate "tolerance" takes as arguments a directed curve **CC** and a tuple of pairs $< \mathbf{EEi}, \tilde{D}_i >$, where **EEi** is a directed edge and $\tilde{D}_i$ is a distance. The predicate is true if there is a direction-preserving, continuous, one-to-one function from the union of the **EEi** to **CC** such that no point in **EEi** is moved by more than $\tilde{D}_i$.

Table 6.9 and Figure 6.21 illustrate the MERCATOR representation.

MERCATOR uses this same representation both for the simulated visual input and for the permanent cognitive map. MERCATOR must therefore address three problems: first, to match the objects in the visual input with objects in the known map; second, to incorporate the new information in the visual input into the cognitive map; and third, to use the cognitive map to retrieve spatial information. Due to the richness of the representation and the inherent complexity of two-dimensional space, it is demonstrably computationally infeasible to develop complete algorithms for these tasks. Even reasonable heuristic algorithms tend to be quite involved.

Table 6.9 MERCATOR Representation

Grid of edges:

dist(**a,b**) ∈ [10,15] · foot.
dist(**b,c**) ∈ [20,30] · foot.
dist(**b,e**) ∈ [10,15] · foot.
dist(**c,d**) ∈ [11,17] · foot.
dist(**c,f**) ∈ [12,18] · foot.
dist(**e,f**) ∈ [12,16] · foot.
dist(**e,j**) ∈ [8,12] · foot.
dist(**f,g**) ∈ [12,18] · foot.
dist(**g,h**) ∈ [9,15] · foot.
dist(**h,i**) ∈ [13,19] · foot.
dist(**i,j**) ∈ [12,16] · foot.

($\hat{x}$ below is a standard reference direction.)

angle(**b-a**,$\hat{x}$) ∈ [20°, 35°].
angle(**c-b**,$\hat{x}$) ∈ [0°, 5°].
angle(**e-b**,$\hat{x}$) ∈ [80°, 90°].
angle(**d-c**,$\hat{x}$) ∈ [15°, 40°].
angle(**f-c**,$\hat{x}$) ∈ [120°, 130°].
angle(**f-e**,$\hat{x}$) ∈ [−10°, 0°].
angle(**j-e**,$\hat{x}$) ∈ [80°, 90°].
angle(**g-f**,$\hat{x}$) ∈ [95°, 110°].
angle(**h-g**,$\hat{x}$) ∈ [70°, 85°].
angle(**i-h**,$\hat{x}$) ∈ [175°, 185°].
angle(**j-i**,$\hat{x}$) ∈ [280°, 305°].

(dboundary(**RR**) below is the boundary of region **RR**, directed counterclockwise around **RR**.
dedge(**X,Y**) is the directed edge from **X** to **Y**.)

tolerance(dboundary(road),
<< dedge(**a,b**), 1.5 · foot >, < dedge(**b,c**), 1.2 · foot >,
< dedge(**c,d**), 0.6 · foot >>).

tolerance(dboundary(road),
<< dedge(**d,c**), 1.2 · foot >, < dedge(**c,b**), 1.2 · foot >,
< dedge(**b,a**), 0.5 · foot >>).

tolerance(dboundary(lake),
<< dedge(**e,f**), 2.5 · foot >, < dedge(**f,g**), 1.8 · foot >,
< dedge(**g,h**), 1.8 · foot >, < dedge(**h,i**), 2.5 · foot >,
< dedge(**i,j**), 2.5 · foot >, < dedge(**j,e**), 1.5 · foot >>).

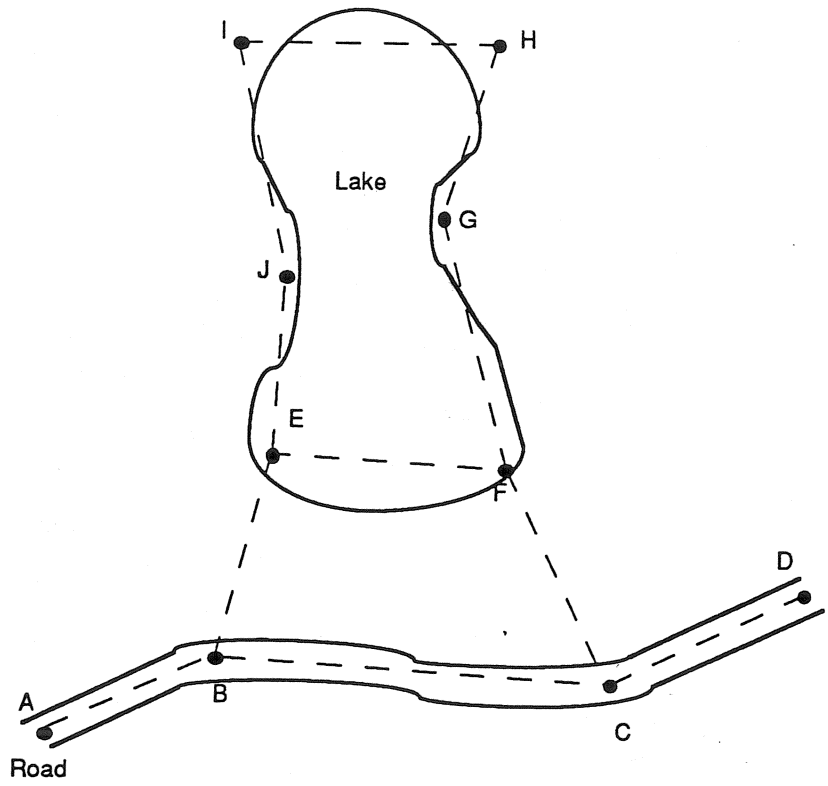


Figure 6.21 MERCATOR representation

For example, to compute tight bounds on the distance between two points **U** and **V** in the first-level grid, given the constraints on edges in the graph, is demonstrably NP-hard. MERCATOR uses the following heuristic: Find a path of edges from **U** to **V**, and evaluate the distance from **U** to **V** under randomly chosen values for the lengths and orientations of the edge satisfying the constraints. To find the distance between two polygons in the grid, MERCATOR uses a heuristic search to pick out likely candidates for the closest pairs of vertices and edges, and then uses a similar Monte Carlo search to evaluate the distance between each candidate vertex-edge pair. To find bounds on the distance between two objects (or, more exactly, between the known parts of two objects), MERCATOR finds the distance between their approx-

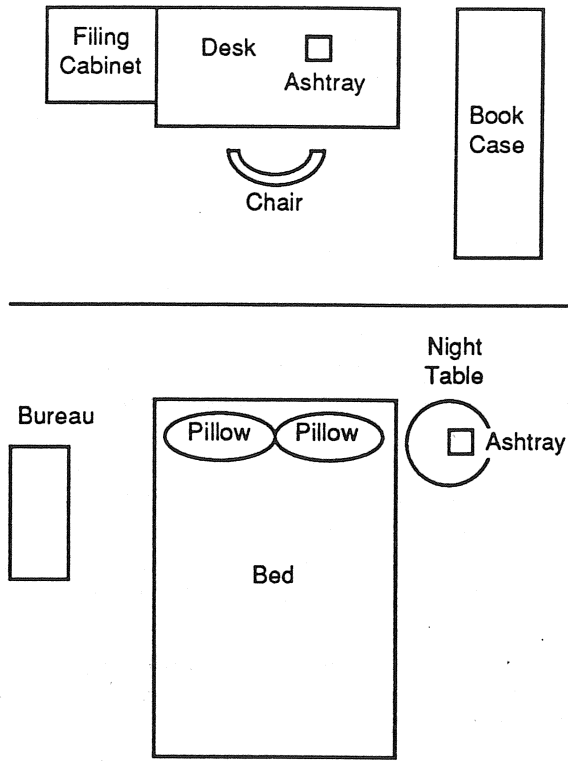
imating polygons, then adds in an uncertainty corresponding to the sum of the accuracies of the two approximations.

Two particular omissions in the MERCATOR representation should be noted. First, the language of constraints used — interval bounds on lengths and directions of edges — is not powerful enough to express many common states of partial knowledge. There is no way in such a representation to express the statement that a given quadrilateral is a square of unknown dimensions, or of unknown orientation, since each side of the square must be allowed to vary independently within the specified bounds. This defect could be remedied, in either of two ways. One way would be just to allow symbolic constraints of a more complex form, such as a constraint "length(**EE**) = length(**FF**)" for two edges **EE** and **FF**. The other would be to use multiple coordinate frames. We could then describe a square of unknown size and orientation by saying that it is a square of size 1 in some coordinate frame, but that the unit length and orientation of that frame are only partially known. (This approach is used in [McDermott and Davis 1984].) Of course, either of these extensions would make the problems of computing values from a map even more difficult.

A second omission in the MERCATOR representation is that it offers no way to express absence information. A cognitive map in MERCATOR never rules out, explicitly or implicitly, the possibility of any number of other objects being anywhere they choose. MERCATOR always allows the possibility that it may have overlooked objects of any kind, and it is not aware of restrictions on the positions of objects, such as the rule that solid objects may not overlap. Therefore, if the matcher of MERCATOR is given two maps, such as in Figure 6.22, which disagree entirely except for two matching objects, the matcher will not conclude that they represent two different places. Rather, it will conclude that the matching objects are in fact the same, and that the remaining objects were just overlooked on one occasion or the other.

6.2.4 Topological Route Maps

Most land travel over extended distances is carried out on well-defined roads, which are very long and thin. The primary knowledge needed for planning routes for such travel is the incidence relations between roads and places: which roads meet, where they meet, and which places lie on which roads. It is considerably less important to know distance or angle relations; one should distinguish between a ten-minute ride and a three-hour ride, or between a gentle left turn and a hard



Absent other information, MERCATOR will identify the two ashtrays, and will create a composite containing all the objects from both rooms, freely overlapping one another.

Figure 6.22 Maps matched by MERCATOR

right, but precise measurements are not generally needed. Global spatial knowledge is often entirely superfluous; it is notorious that people can travel from point a to point b for years, and still have only vague or mistaken notions of their relative positions. The TOUR program [Kuipers 1978] was designed for performing assimilation and inference on such information about incidence relations.

TOUR uses three ontological sorts of entities: places, which are points; paths, which are directed curves; and regions. There is also a single mobile robot; the robot is the only thing that moves over time.

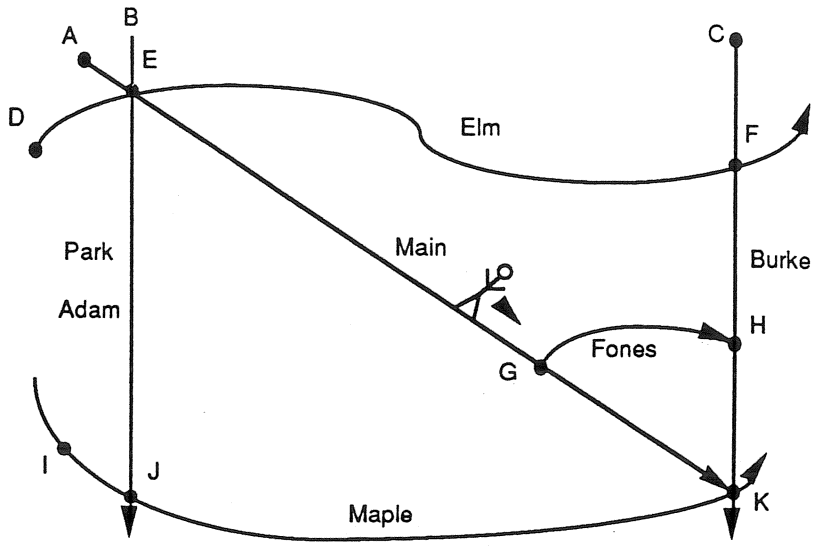


Figure 6.23 World state in TOUR

The state of the robot is defined by its position, which is a place, and its orientation, which is a path and a direction on the path. Figure 6.23 shows a typical world state. A TOUR cognitive map is a collection of atomic, ground formulas, together with a state function that tracks the current position of the robot. Table 6.10 lists the primitives used by TOUR; Table 6.11 gives the TOUR representation of the world state of Figure 6.23.

The TOUR program simulates the assimilation of a cognitive map through the execution of these actions. The robot starts out knowing nothing about the relations in the world. It is given a series of executable actions to carry out. When it executes a "go" command, it can add its destination on the path traveled, in the proper direction from its source. When it executes a "turn" command, it finds itself on a new path, which it can record as meeting the star of its current place. TOUR does not support the assimilation of region information.

Table 6.10 Primitives in TOUR

Static, spatial predicates:

on_path($\mathbf{PP}, \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_k$) —
Places $\mathbf{X}_1$ through $\mathbf{X}_k$ occur in that order on path **PP**.

star($\mathbf{X}, < \mathbf{PP}_1, S_1 >, < \mathbf{PP}_2, S_2 > \ldots < \mathbf{PP}_k, S_k >$) —
Paths $\mathbf{PP}_1$ through $\mathbf{PP}_k$ all meet at place **X**; moreover, the directed paths $\mathbf{PP}_i$ with sense S_i occur in counter-clockwise order around **X**. S_i is a Boolean, indicating forward or backward direction.[3]

border(**RR** , **PP** , S) —
Path **PP** is on the border of region **RR**. S is a Boolean, indicating whether the forward direction on **PP** goes clockwise or counter-clockwise around **RR**.

$\mathbf{X} \in \mathbf{RR}$

$\mathbf{RR1} \subset \mathbf{RR2}$

State function:

robot(**X**,**PP**, S) —
The robot is now at place **X** oriented along path **PP** in the direction indicated by Boolean S.

Events:

go_forward — The robot moves to the next place along current path.

turn(R) — Turn to next path at current place. R is a Boolean, indicating clockwise or counter-clockwise rotation.

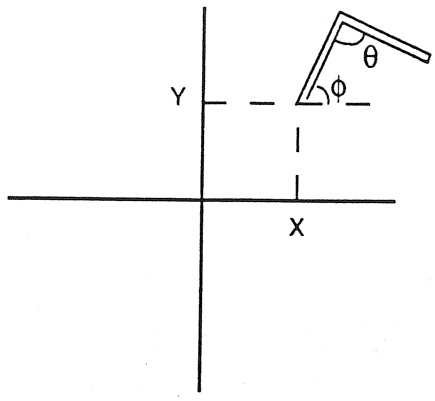
[3]The actual TOUR program indicated the angle of the various paths, and the angle of rotation in the "turn" action described below.

Table 6.11 World State in TOUR

```
path(main, a, e, g, k).
path(elm, d,e,f).
path(maple, i,j,k).
path(adam, b,e,j).
path(burke,c,f,h,k).
path(fones,g,h).

star(e, < elm, + >, < adam, − >, < main, − >,
        < elm, − >, < adam, + >, < main, + >).
star(f, < elm, + >, < burke, − >, < elm, − >, < burke, + >).
star(g, < main, + >, < fones, + >, < main, − >).
star(f, < burke, + >, < burke, − >, < fones, − >).
star(j, < maple, + >, < adam, − >, < maple, − >, < adam, + >).
star(k, < maple, + >, < burke, − >, < main, − >,
        < maple, − >, < burke, + >).

border(maple, park, +).
border(adam, park, −).
border(elm, park, −).

robot(g, main,+).
```

6.2.5 Configuration Spaces

In reasoning about the motion of hard objects among hard obstacles, it is sometimes helpful to view the problem in terms of a *configuration space*. A *configuration* is a positioning of all the mobile objects in the problem, and a configuration space is the space of all such configurations. The dimensionality of the configuration space is the total number of degrees of freedom of the objects. k separate objects in two space determine a $3k$-dimensional configuration space (two dimensions of translation and one of rotation for each object), while k objects in three space determine a $6k$-dimensional configuration space (three of translation and three of rotation). Two objects connected by a pin joint in two dimensions have, together, four degrees of freedom: three determining the translation and rotation of the first, and one determining the angle of the joint (Figure 6.24). They are thus described in a four-dimensional configuration space.



The configuration space of the pair of jointed rods has four dimensions: x, y, , ϕ and θ

Figure 6.24 Configuration of a joint

Using a configuration space thus reduces a complex state description to a single point, at the cost of imposing a complex, high-dimensional space. The space is rather complex because some of its dimensions correspond to rotations and are therefore cyclical.

The configuration space is divided into physically attainable regions, where the actual objects do not overlap, and physically unattainable regions, where they do. All physical behavior must take place in the physically attainable regions. For example, a feasible motion of the objects from one positioning to another corresponds to a continuous path in configuration space from the starting point to the ending point through the physically attainable region.

If **RR** is the region occupied by an object O in some standard position, and C is a configuration of O, then we will denote the region occupied by O in C as "image(C, **RR**)." In this book, we will use this only in the case where O is a rigid object, so that the configuration C is a rigid mapping in space.

Configuration spaces are easiest to use and most effective when the problem is structured so that the configuration space has relatively few dimensions. This can happen if the motion of the mobile objects is restricted by some external constraint to a few degrees of freedom. An example is a pair of meshed gears that are each pinned so that they

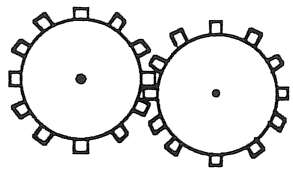


Figure 6.25 Meshed gears

can only rotate around their axes. The relative position of the gears is completely determined by their two orientations, and the relevant configuration space describes the pairs of orientations that do not cause them to overlap (Figure 6.25). Another example is in the analysis of jointed objects such as robots: A configuration space analysis focuses attention on the actual configurations that the robot can attain, rather than all conceivable combinations of positions of its parts.

The dimension of the configuration space can also be reduced if the mobile object has some kind of continuous symmetry so that movements in the symmetry group can be ignored: An example is a circular disk moving in two space around obstacles. The configuration space of the disk is two dimensional, since it is invariant under rotations. This configuration space is the set of positions of the center of the disk. The unattainable regions have the shape of the obstacles, swollen by the radius of the disk (Figure 6.26).

Configuration spaces are particularly useful in reasoning about rigid objects in circumstances where their physical behavior depends on their shapes and positions, and is independent of such issues as velocity and mass. Consider, for example, a collection of blocks on a table that move only when they are pushed. Suppose that the motion of one block is controlled by an external force, and we wish to determine the motions of the remaining blocks. This behavior can be calculated from the configuration space as follows: If the motion of the controlled block lies in the interior of the space, then the other blocks remain motionless. If the motion of the controlled block will bring it into the area of the space forbidden by the other blocks, then the configuration as a whole moves along the boundary of the forbidden area, in a way that the control block executes its specified motion (Figure 6.27).

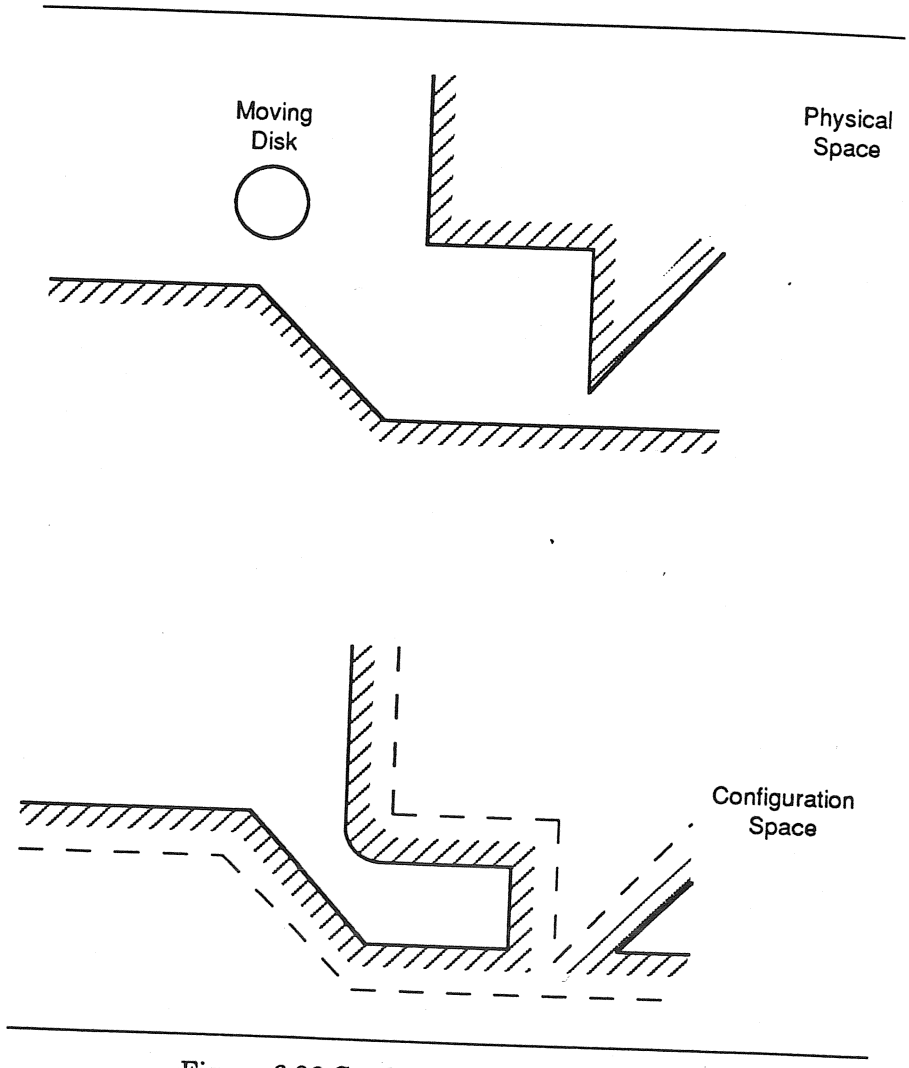
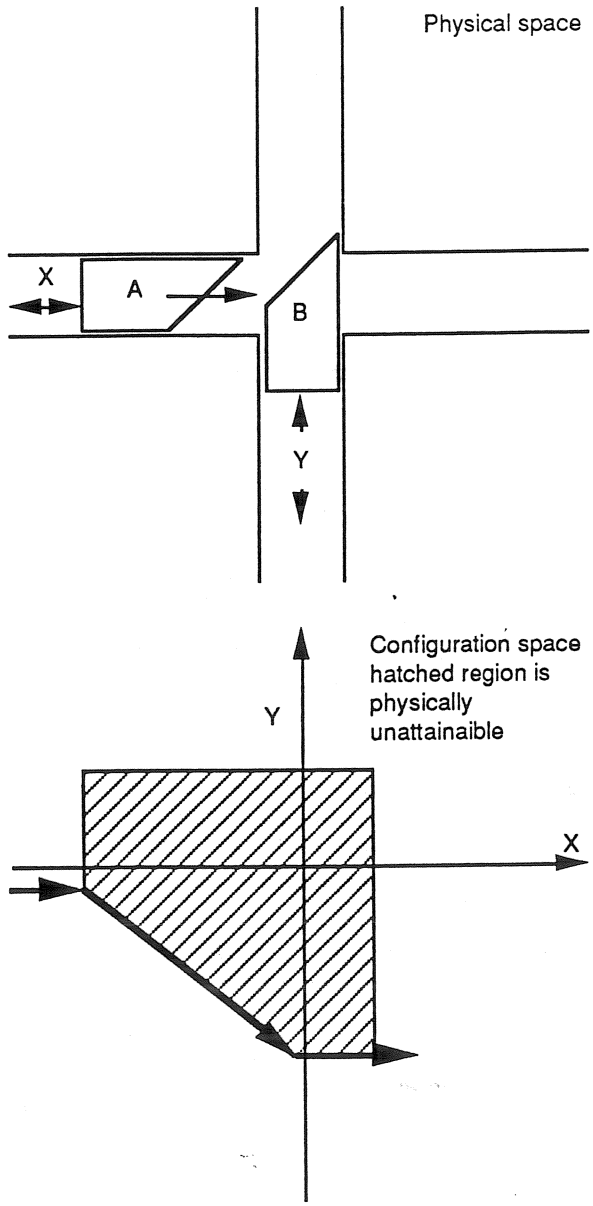


Figure 6.26 Configuration space of a disk



If block A is pushed to the right, the configuration follows the marked path through configuration space.

Figure 6.27 Configuration space in quasi-static environments

6.2.6 The Roller Coaster

NEWTON [de Kleer 1975] is a program that predicts the behavior of a cart on a roller-coaster track from qualitative information about the shape of the track. The cart is assumed to move without friction under the force of gravity. The problem is to predict how the velocity and position of the cart will change over time. The cart is not securely held to the track, so it is possible that the cart may fall or fly off the track. NEWTON will predict that the cart comes off the track, but does not attempt to guess its further fate.

In a complete model, we would have to consider the cart and the track as three-dimensional objects, that may be in contact over some extended region. (Figure 6.28). However, for the purposes of qualitative reasoning, we may make some radical simplifications. We assume that we can ignore the third dimension, and describe the entire system in terms of a world with one horizontal and one vertical dimension. We assume that the interactions between the cart and the track depend only on the boundary of the track; hence, we can model the track as a curve in two dimensions, with a distinguished clockwise direction around it. (The two-dimensional representation of the track may cross itself; the two corresponding parts of the three-dimensional track would be slightly displaced in the third dimension at the cross point.) The cart is assumed to be small enough (compared to any shape features of the track) that the shape of the cart is irrelevant and the contact between the cart and the track can be approximated as a single point. Therefore, we can describe the state of the cart simply by specifying the location of the contact point and requiring that the cart lies on the outside of the track rather than inside it. Whether these approximations can be validly made for a particular cart and track is outside the scope of NEWTON's theory.

Two forces affect the behavior of the cart relative to the track: the gravitational force and the centrifugal force. The gravitational force is always constant, and directed vertically downward. The centrifugal force is always directed along the normal to the track. Where the curvature is positive (the track curves up toward the cart), the centrifugal force points inward to the track; where it is negative, the centrifugal force points outward, away from the track. The magnitude of the centrifugal force is proportional to the square of the velocity of the cart times the curvature of the track.

We can summarize the effect of these two forces in the following two rules:

1. The acceleration of the cart along the track is proportional to the component of gravity in the direction tangent to the track. (The

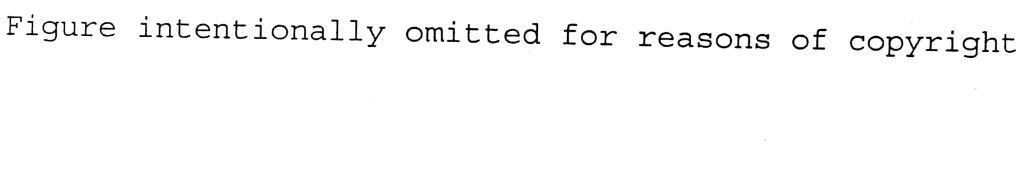


Figure 6.28 Cart on a roller coaster

centrifugal force is always normal to the track.) If the forward direction of the track is uphill, then the cart will accelerate backward; if it is downhill, the cart will accelerate forward.

2. The force holding the cart on the track is the sum of the centrifugal force plus the component of gravity in the direction normal to the track. If this sum is positive or zero, the cart will stay on the track; if it becomes negative, the cart will fall or fly off the track.

Reasoning about the behavior of the cart therefore requires some specification of the shape of the track, including its tangent and curvature at each point, and the history of position and velocity of the cart along the track over time. In NEWTON, the shape of the track is represented as an ordered sequence of segments. For each segment, we record the quadrant of the forward tangent direction, the sign of the curvature, and whether the segment is an isolated point or is a curve of finite length. (See Figure 6.29 and Table 6.12.) The segments are chosen so that the quadrant of the tangent direction and the sign of the curvature have a single value over the segment.[4]

Directions are divided into eight quadrant values: Up, Right, Down, Left, Up-Right, Up-Left, Down-Right, Down-Left (abbreviated U, R, D, L, UR, UL, DR, DL). We will call the first four "point directions," and the last four "range directions." (Figure 6.30)

This representation for curves is analogous to the representation of temporal parameters discussed in Section 4.8. Here, the independent variable, corresponding to time, is arc length; the dependent variables, corresponding to parameters, are curvature and direction.

We specify the position of the cart at an instant in terms of the segment of the curve where it is located. We specify its velocity as going forward on the curve, going backward, or standing still. A state of the cart is specified by giving the position of the cart and its velocity. We also label each state with an indication of whether it persists for finite time or whether it occurs only for an instant. For example, the starting state of the cart illustrated in Figure 6.29 is specified as being in SEG4, with a positive velocity; the state lasts for finite time. The special state "FELL_OFF" is terminal.

The task of the NEWTON program is thus to characterize the possible sequences of states of the cart, given the shape description of the track and some starting state of the cart. The output of the program is expressed in terms of a *mode transition graph,* which shows which states can follow other states. As discussed in Section 4.9, any pos-

[4]The actual NEWTON program supported a richer representation with metric information about height, which allowed some ambiguities in behavior to be resolved.

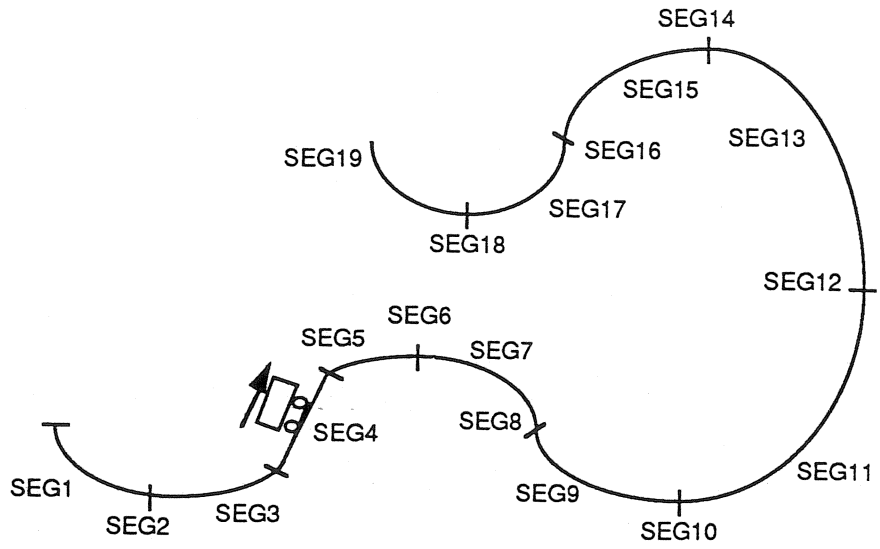


Figure 6.29 Track in NEWTON

sible behavior of the system must correspond to a path through the graph; the converse is not true.

There are two problems to be solved: (1) Determine whether a shape description of the track is consistent; and (2) Find the behavior of the cart. We will consider each of these in turn.

The shape description of the track must satisfy the following geometrical constraints:

A.1. Continuity. The quadrant of the tangent must change continuously, going from one range in the circle of quadrants to a neighboring range (Figure 6.30). The sign of the curvature must change continuously; to go from negative to positive curvature or vice versa, it must go through zero.

A.2. Mean-value theorem. The curvature is the derivative of the tangent angle; hence, changes in the tangent must be in the direction indicated by the curvature. Specifically:

   i. If the tangent direction changes from a point direction to the neighboring range in the counterclockwise (clockwise) direction, then the curvature in the second segment must be positive (negative).

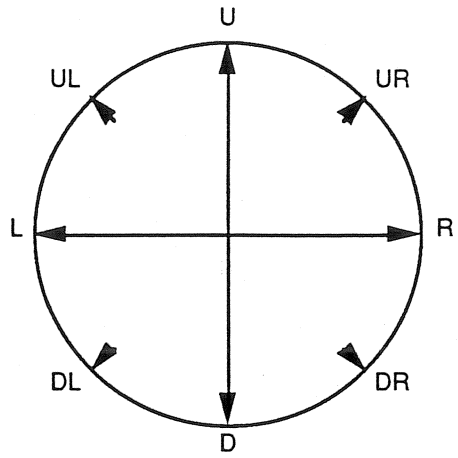


Figure 6.30 Quadrants of directions

ii. If the tangent direction changes from a range direction to the neighboring point in the counterclockwise (clockwise) direction, then the curvature in the first segment must be positive (negative).

A.3. A segment with a point direction and nonzero curvature must occupy only a point and it must border a segment with a range direction on both sides. A segment with a range direction and a nonzero curvature must have finite length.

A.4. (Topology of change.) It is impossible that one segment should have nonzero curvature and a point tangent direction and that an adjacent segment have zero curvature and a range tangent direction.

Any sequence of segments obeying these constraints is a valid shape description. Note that, if two segments appear consecutively, at least one must be of finite length.

We can now describe the dynamics of the system using qualitative differential equations, as described in Section 4.9. Let **PP** be the direction curve of the track; let $\mathbf{X}(T)$ be the arc length coordinate of the cart along the track; and let $V(T)$ be the derivative of $\mathbf{X}(T)$. Then the acceleration of the cart along the track is proportional to the

Table 6.12 Qualitative Representation of the Track in Figure 6.29

Segment	Tangent	Inward Normal	Curvature	Length
SEG1	DR	DL	POS	Finite
SEG2	R	D	POS	Point
SEG3	UR	DR	POS	Finite
SEG4	UR	DR	ZERO	Finite
SEG5	UR	DR	NEG	Finite
SEG6	R	D	NEG	Point
SEG7	DR	DL	NEG	Finite
SEG8	DR	DL	ZERO	Point
SEG9	DR	DL	POS	Finite
SEG10	R	D	POS	Point
SEG11	UR	DR	POS	Finite
SEG12	U	R	POS	Point
SEG13	UL	UR	POS	Finite
SEG14	L	U	POS	Point
SEG15	LD	UL	POS	Finite
SEG16	LD	UL	ZERO	Point
SEG17	LD	UL	NEG	Finite
SEG18	L	U	NEG	Point
SEG19	UL	UR	NEG	Finite

component of the forward tangent in the downward direction. This can be stated in the following pair of qualitative differential equations:

$$\partial X = V$$
$$\partial V = -[\ \text{direction}(\text{tangent}(\mathbf{PP}\ ,\ \mathbf{X})) \cdot \hat{k}]$$

where tangent(**PP**, **X**) is the forward tangent to the curve **PP** at **X**, and $\hat{k}$ is the vertical direction.

These equations are adequate as long as the cart stays on the track. If the cart were attached to the track, like a bead on a wire, they would be a full specification of the behavior. We can extend the system to consider the possibility that the cart may fall off the track as follows: As stated above, the cart will fall off if the sum of the centrifugal force with the component of gravitational force in the direction normal into the track becomes negative. If both of these terms are negative, or one is negative and the other zero, then the sum must be negative; if one is negative and the other positive, then the sum could be negative,

zero, or positive. The sign of the centrifugal force is equal to the sign of the curvature. Therefore, we can formulate the following rule:

Falling off: The cart may fall off at point **X** if and only if either normal(**PP** , **X**) has a component vertically down or if curvature(**PP** , **X**) is negative and V is nonzero. The cart must fall off if normal(**PP** , **X**) has a component vertically down, and either V is zero or curvature (**PP** , **X**) is not positive.

By combining these special falling-off transitions with the transitions generated in solving the above QDE's, we can find the behavior of the cart.

For example, the cart in Figure 6.29 starts in a state, which we call ST4a, in which it is located in SEG4 with a positive velocity. Since the tangent to SEG4 is upward and to the right, the acceleration must be negative. Since the inward normal to the track is downward and to the right, the force of gravity pushes it onto the track. Since the curvature of the track is zero, the centrifugal force is zero. Therefore, the cart cannot fly off the track. Two transitions out of ST4a are possible. The velocity may carry the position to the next segment. Then the cart will be on SEG5 with positive velocity; this is state ST5a. Alternatively, the acceleration may bring the velocity to zero. Then the cart will still be on segment SEG4 with zero velocity; this is state ST4b.

In state ST5a, the acceleration of the cart on the track is negative, since the forward tangent to SEG5 is upward to the right. Since the inward normal is downward and to the right, the gravitational force tends to hold the cart on the track. However, since the curvature of the track is negative, the centrifugal force tends to push the cart off the track. If the velocity of the cart is large, so that the centrifugal force is large, or if the slope of the track is large, so that the component of gravity pushing toward the track is small, then the centrifugal force may overcome gravity, and the cart will fly off the track. Otherwise, there are three possible transitions: either the velocity will carry the cart onto segment SEG6, or the acceleration will bring the velocity to zero, or both will happen simultaneously (Figure 6.31).

By continuing this type of analysis, one can create the complete envisionment graph for the behavior. (Exercise 3). The transition graph allows many different possible behaviors: the cart may oscillate in the first valley, in the second valley, or between the two valleys; it may fly off on the hill, or fall off in the overhang (SEG11, SEG12, and SEG13). If it makes it to the inverted hill (segment SEG17), then it must fall off.

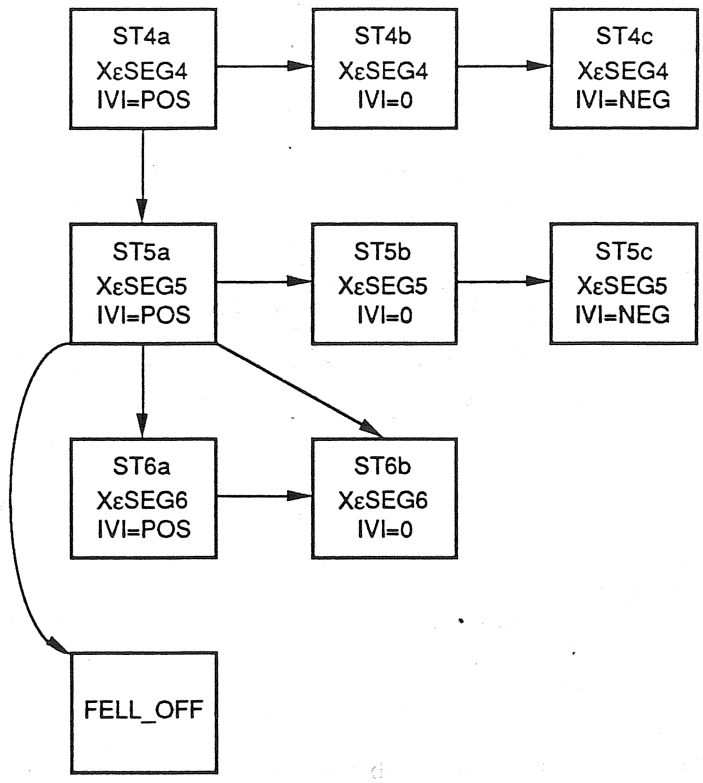


Figure 6.31 State transitions in NEWTON

6.3 Appendix A: Coordinate Transformations

As we have seen in Sections 6.1.3, 6.2.1, and 6.2.2, it is often convenient to describe spatial information in terms of coordinates in a specially chosen coordinate system. There are a number of purposes that may be served by such a representational strategy:

- It may simplify description and calculation. For example, the inequalities that describe a solid right cylinder are very much simpler in a coordinate system aligned with the axis of the cylinder than in other coordinate systems.
- If the shape moves rigidly through space over time, then the motion of the object can be described by having its coordinate system move

and the shape of the object remain constant relative to its own coordinate system. The motions of a coordinate system are much simpler and easier to describe than the transformations of a region.

- The information available to an intelligent creature is often precise on a local scale and much less precise on a global scale. In such a case, it may be effective to use a collection of local coordinate systems.

In order to use this strategy, we must have a language for expressing the relations between one coordinate system and another, and for computing with these relations. This appendix presents a number of ways to do this.

Formally, a *coordinate system* in n-space is a triple $\mathcal{C} =< \mathbf{O}, \tilde{L}, \hat{DD} >$, where $\mathbf{O}$ is the origin (a point), $\tilde{L}$ is the unit length, and $\hat{DD}$ is the frame of axis directions, an n-tuple $< \hat{D}_1 \ldots \hat{D}_n >$ of mutually perpendicular directions, ordered with the "right-hand" orientation. Fixing a coordinate system gives a standard way of naming points, lengths, and directions:

Let $\mathcal{C} =< \mathbf{O}, \tilde{L}, < \hat{D}_1 \ldots \hat{D}_n >>$ be a coordinate system in n-space. We define the measure of length $\tilde{M}$ in $\mathcal{C}$, "measure($\tilde{M}, \mathcal{C}$)," to be the real number $\tilde{M}/\tilde{L}$. The product $\tilde{L} \cdot \hat{D}$ is defined as the vector with length $\tilde{L}$ and direction $\hat{D}$. Finally, we define the coordinates of a vector $\vec{V}$ in $\mathcal{C}$ to be the unique n-tuple of real numbers $c_1 \ldots c_n$ such that

$$\vec{V} = c_1 \cdot \tilde{L} \cdot \hat{D}_1 + \ldots + c_n \cdot \tilde{L} \cdot \hat{D}_n$$

We define the coordinates of a point $\mathbf{A}$ in $\mathcal{C}$, "coordinates($\mathbf{A}, \mathcal{C}$)," to be equal to the coordinates of the vector $\mathbf{A} - \mathbf{O}$. In this representation, addition and subtraction of vectors and points can be carried out place by place.

There are a number of ways of using coordinate systems to specify a direction $\hat{E}$. One approach is to use the directional cosines: that is, the projections of $\hat{E}$ on the coordinate axes. Formally, we define the direction cosines of direction $\hat{E}$, written "dir_cosines($\hat{E}, \mathcal{C}$)," to be the coordinates of $\tilde{L} \cdot \hat{E}$. Another approach is to represent $\hat{E}$ in terms of the angle between $\hat{E}$ and the coordinate axes. In two dimensions, only one angle is needed; we define "angle($\hat{E}, \mathcal{C}$)" to be the angle between $\hat{E}$ and the x axis of $\mathcal{C}$. In three dimensions, two angles are needed. There are a number of ways to choose these angles. One typical approach is to use the co-latitude of $\hat{E}$, which is the angle between $\hat{E}$ and the z axis, and the longitude of $\hat{E}$, which is the angle between the x axis and the x–y projection of $\hat{E}$ (Figure 6.32).

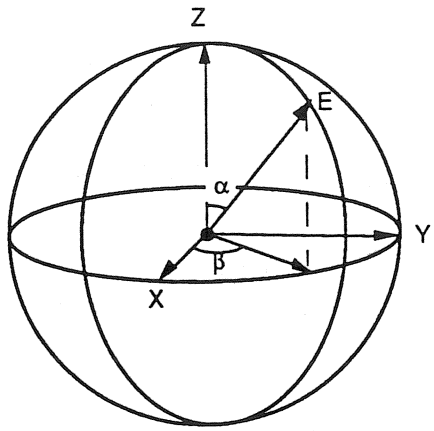


Figure 6.32 Angle representations for a direction

The relation between two frames $\mathcal{C}$ and $\mathcal{F}$ can be represented by characterizing the elements of $\mathcal{F}$ — its origin, unit length, and frame of axis directions — in terms of $\mathcal{C}$. The unit and origin are straightforward; the unit is given by its measure and the origin by its coordinates. There are, however, a number of ways of representing the axis directions, each with strengths and weaknesses.

The first method is to represent the frame of axis directions of $\mathcal{F}$ in terms of the directional cosines of each direction. Let $\mathcal{C}$ and $\mathcal{F} = <\mathbf{O}, \tilde{L}, < \hat{D}_1, \hat{D}_2 >>$ be two two-dimensional coordinate frames. Let coordinates($\mathbf{O}$,$\mathcal{C}$) = $< O_1, O_2 >$; let measure($\tilde{L}, \mathcal{C}$) = L; let dir_cosines ($\hat{D}_1, \mathcal{C}$) = $< D_{11}, D_{12} >$; and let dir_cosines($\hat{D}_2, \mathcal{C}$) = $< D_{21}, D_{22} >$. The transformations from $\mathcal{F}$ to $\mathcal{C}$ obey the following rules:

For any length $\tilde{M}$, measure($\tilde{M}, \mathcal{C}$) = L·measure($\tilde{M}, \mathcal{F}$).

For any direction $\hat{E}$, let dir_cosines($\hat{E}, \mathcal{F}$) = $< E_1, E_2 >$. Then
dir_cosines($\hat{E}, \mathcal{C}$) = $< D_{11}E_1 + D_{21}E_2, D_{12}E_1 + D_{22}E_2 >$

For any point $\mathbf{P}$, let coordinates($\mathbf{P}, \mathcal{F}$) = $< P_1, P_2 >$. Then

$$\text{coordinates}(\mathbf{P}, \mathcal{C}) = < O_1 + L(D_{11}P_1 + D_{21}P_2), \\ O_2 + L(D_{12}P_1 + D_{22}P_2) >$$

The last two transformations above can be written more concisely and clearly in matrix notation. Let us identify any pair of numbers with the corresponding column array.

$$< X_1, X_2 >= \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

Let dir_cosines($\hat{DD}, \mathcal{C}$) be the square array of direction cosines

$$\text{dir\_cosines}(\hat{DD}, \mathcal{C}) = \begin{bmatrix} D_{11} & D_{21} \\ D_{12} & D_{22} \end{bmatrix}$$

Then the above formulas may be expressed as follows:

$$\text{dir\_cosines}(\hat{E}, \mathcal{C}) = \text{dir\_cosines}(\hat{DD}, \mathcal{C}) \cdot \text{dir\_cosines}(\hat{E}, \mathcal{F})$$

$$\text{coordinates}(\mathbf{P}, \mathcal{C}) = \text{coordinates}(\mathbf{O}, \mathcal{C}) + \text{measure}(\tilde{L}, \mathcal{C}) \cdot \text{dir\_cosines}(\hat{DD}, \mathcal{C}) \cdot \text{coordinates}(\mathbf{P}, \mathcal{F})$$

The rule for coordinate transformations above uses a matrix multiplication and a vector addition. These two can be collapsed into a single matrix multiplication. This reduction involves adding an additional fictional dimension. We represent a two-dimensional point $\mathbf{P}$ in coordinate frame $\mathcal{C}$ by a column array of three elements: the two coordinates of $\mathbf{P}$ in $\mathcal{C}$ and the number 1.

$$\text{coor1}(\mathbf{P}, \mathcal{C}) =< \text{coordinates}(\mathbf{P}, \mathcal{C}), 1 >$$

We represent the coordinate transformation from $\mathcal{C}$ to $\mathcal{F}$ by the three-by-three array:

$$K = \begin{bmatrix} LD_{11} & LD_{21} & O_1 \\ LD_{12} & LD_{22} & O_2 \\ 0 & 0 & 1 \end{bmatrix}$$

The transformation from coordinate system $\mathcal{F}$ to $\mathcal{C}$ can now be represented just as multiplying by the matrix K.

$$\text{coor1}(\mathbf{P}, \mathcal{C}) = K \cdot \text{coor1}(\mathbf{P}, \mathcal{F})$$

If the matrix above is considered as a representation for the transformation from $\mathcal{F}$ to $\mathcal{C}$, then the composition of transformations can be computed as the product of matrices, and the inversion of a transformation can be computed as the inverse of the matrix.

Three-dimensional coordinate transformations can be handled analogously.

The advantage of this representation is the simplicity of the above formulas. The disadvantages arise from the fact that it is highly redundant. The representation uses four numbers to represent a

two-dimensional rotation, which is a one-parameter space, and nine numbers to represent a three-dimensional rotation, which is a three-parameter space. This redundancy means that the numbers used here are mutually constrained. If exact values are known, then there is no problem. However, if only partial information is available, then reasoning with this information will involve incorporating these built-in constraints, either explicitly or implicitly. This complicates both the representation of partial information and calculations with it. It is therefore worth considering alternative representations for rotations, based on angles. These simplify the representation of rotations, at the cost of complicating the transformation equations. Here, we must consider the two-dimensional case, which is quite easy, separately from the three-dimensional case, which is difficult.

In two dimensions, the rotation from coordinate frame $\mathcal{C}$ to $\mathcal{F}$ can be represented by the angle between their x directions, which we denote "angle($\mathcal{F}, \mathcal{C}$)". Let "angle($\hat{E}, \mathcal{C}$)" be the angle between the direction $\hat{E}$ and the x axis of $\mathcal{C}$. We then have the following simple equations:

$$\text{angle}(\hat{E}, \mathcal{F}) + \text{angle}(\mathcal{F}, \mathcal{C}) = \text{angle}(\hat{E}, \mathcal{C}) \bmod 2\pi.$$
$$\text{angle}(\mathcal{G}, \mathcal{F}) + \text{angle}(\mathcal{F}, \mathcal{C}) = \text{angle}(\mathcal{G}, \mathcal{C}) \bmod 2\pi.$$
$$\text{angle}(\mathcal{F}, \mathcal{C}) = -\ \text{angle}(\mathcal{C}, \mathcal{F}) \bmod 2\pi$$

Moreover, uncertainty in the amount of a rotation can be represented in simple interval constraints, such as $\text{angle}(\mathcal{F}, \mathcal{C}) \in [20°, 30°]$.

Overall, the coordinate transformation from $\mathcal{F}$ to $\mathcal{C}$ is represented in terms of four real numbers: The coordinates $< O_1, O_2 >$ of the origin of $\mathcal{F}$ in $\mathcal{C}$; the angle α between $\mathcal{F}$ and $\mathcal{C}$; and the scale change L. The transformation formula is as follows: Let $< P_1, P_2 >$ be the coordinates of point $\mathbf{P}$ in $\mathcal{F}$. Then

$$\begin{aligned}\text{coordinates}(\mathbf{P}, \mathcal{C}) = < & O_1 + L \cdot P_1 \cos(\alpha) - L \cdot P_2 \sin(\alpha), \\ & O_2 + L \cdot P_1 \sin(\alpha) + L \cdot P_2 \cos(\alpha) >\end{aligned}$$

Thus, computing coordinate transformation requires computing trigonometric functions. Similarly, computing angles given coordinate or distance information requires computing inverse trigonometric functions. By contrast, all calculations with directional cosines are algebraic.

The elegance of angles as representations for two-dimensional directions and rotations relies on a few particular properties of these spaces. None of these properties hold in three dimensions; hence, the representation of three-dimensional rotations and directions is necessarily clumsier and more complicated.

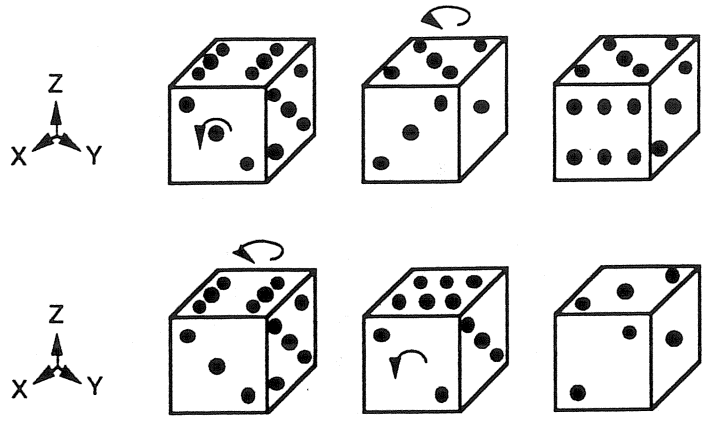


Figure 6.33 Noncommutativity of three-dimensional rotations

- Rotations do not commute. The result of a 90° rotation around the x axis followed by a 90° rotation around the z axis is not equal to the result of the same rotations in the opposite order (Figure 6.33).
- The three-parameter space of rotations is not isomorphic to the two-parameter space of directions. In particular, there is not a unique rotation that maps one direction $\hat{D}$ into another $\hat{E}$; there is a class of such rotations.
- There is no mapping that preserves congruence, either from the plane to the space of three-dimensional directions, or from three space to the space of three-dimensional rotations. Therefore, no technique for naming directions in terms of two real parameters, or for naming rotations in terms of three real parameters, can treat all points uniformly; any naming will treat some points and some areas differently from others. (This last point can be made more strongly in topological terms: Any continuous mapping from two space to the space of three-dimensional directions, or from three space to the space of three-dimensional rotations, must have topological singularities.)

These properties suggest strongly that any representation for three-dimensional directions and rotations will have some inelegancies. There are a number of ways of choosing three angles associated with a rotation as the representation of the rotation. For example, any

three-dimensional rotation has a fixed axis that remains unchanged. A rotation can therefore be represented by giving the colatitude and longitude of the direction of the axis of rotation, and the angular amount of the rotation. Similar representations can be derived by writing a rotation as the composition of three rotations of variable angles around fixed axes. For example, let $\Psi_Z(\alpha)$ be the rotation by angle α around the z axis, and let $\Psi_Y(\beta)$ be the rotation by angle β around the y axis. It is then a fact that any rotation can be decomposed in the form $\Psi_Z(\alpha) \cdot \Psi_Y(\beta) \cdot \Psi_Z(\gamma)$ for some angles α, β, γ. The triple $< \alpha, \beta, \gamma >$ are known as the Z-Y-Z Euler angles of the rotation. (Figure 6.34)

Such angle-based systems have the advantage that, with practice, they can be easily visualized (at least as compared to the nine directional cosines). Moreover, they lend themselves to certain types of physical calculations; the Euler angles, for example, are useful in calculating the energy of objects that are radially symmetric. However, for pure geometric calculations, they are mostly horrible. To determine the image of an arbitrary direction under a rotation described in terms of a three-axis system, or to compose two rotations described in a three-axis system, it is generally necessary to compute many trigonometric and inverse trigonometric functions; in effect, to translate from the axis-system to the directional cosines and then translate back.

Another technique for representing three-dimensional rotations is to use *quaternions*. Quaternions are less compact than the above angle representations, having one redundant parameter, but they are much easier to compute with. A quaternion is a triple of four numbers, $< a, b, c, d >$, generally written $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$. (The symbols **i,j,k** here are particular quaternion constants, rather than geometrical points.) Quaternions may be added and multiplied according to the following rules: Let $P = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$. Let $Q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$. Then

$$\begin{aligned} P + Q &= (a + w) + (b + x)\mathbf{i} + (c + y)\mathbf{j} + (d + z)\mathbf{k} \\ PQ &= (aw - bx - cy - dz) + \\ &\quad (ax + bw + cz - dy)\mathbf{i} + (ay - bz + cw + dx)\mathbf{j} + \\ &\quad (az + by - cx + dw)\mathbf{k} \end{aligned}$$

This complicated rule for multiplication can be derived from the following simpler axioms:

i. Multiplication is associative. $P(QW) = (PQ)W$.
ii. Multiplication distributes over addition on either side.
$(P + Q)W = PW + QW$.
$W(P + Q) = WP + WQ$.

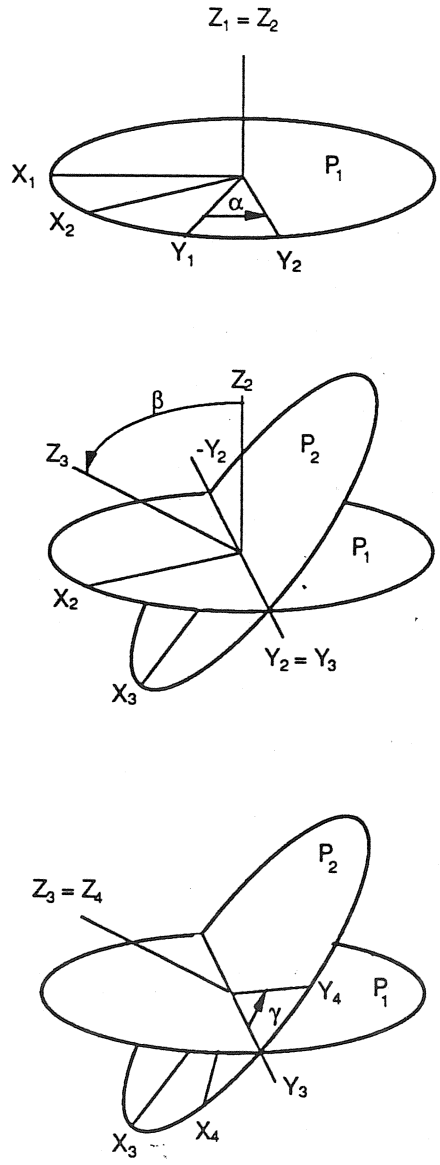


Figure 6.34 Euler angles. $P_1 = X_1 - Y_1$ plane. $P_2 = X_3 - Y_3$ plane.

iii. The quaternion $a+0\mathbf{i}+0\mathbf{j}+0\mathbf{k}$ is identified with the scalar a. Multiplication of a quaternion by a scalar is commutative. $Pa = aP$.

iv. $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$.

v. $\mathbf{ij} = -\mathbf{ji} = \mathbf{k}; \mathbf{jk} = -\mathbf{kj} = \mathbf{i}; \mathbf{ki} = -\mathbf{ik} = \mathbf{j}$.

As part v makes evident, multiplication of quaternions is not commutative.

Let $P = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$. We define conj($P$),the conjugate of P, to be $a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$.

We may now apply quaternion arithmetic to directions as follows. Let $< P_1, P_2, P_3 >$ be the coordinates of point **P** in coordinate frame $\mathcal{F}$. Identify **P** with the quaternion

$$\text{quat}(\mathbf{P}, \mathcal{F}) = 1 + P_1\mathbf{i} + P_2\mathbf{j} + P_3\mathbf{k}$$

Let $\mathcal{C}$ be a coordinate frame with the same origin and unit length as $\mathcal{F}$, so that they differ only by a rotation Φ. Let $\hat{N}$ be the direction that is fixed under Φ; let $< N_1, N_2, N_3 >$ = dir_cosines($\hat{N}, \mathcal{C}$), and let θ be the amount of Φ. We map the rotation to the quaternion

$$\text{quat}(\mathcal{F}, \mathcal{C}) = \cos(\theta/2) + \sin(\theta/2)(N_1\mathbf{i} + N_2\mathbf{j} + N_3\mathbf{k})$$

Then the following identities hold:

a. $\text{quat}(\vec{V}, \mathcal{C}) = \text{quat}(\mathcal{F}, \mathcal{C}) \cdot \text{quat}(\vec{V}, \mathcal{F}) \cdot \text{conj}(\text{quat}(\mathcal{F}, \mathcal{C}))$.
b. $\text{quat}(\mathcal{G}, \mathcal{C}) = \text{quat}(\mathcal{G}, \mathcal{F}) \cdot \text{quat}(\mathcal{F}, \mathcal{C})$.

That is, we can carry out a rotation of a coordinate system in terms of multiplication by quaternions. Proofs of these equations can be found in [Bottema and Roth 1979, chap. 13].

6.4 Appendix B: Going Through

In this appendix, we formally define the event of a fluent **FF** "going through" a surface **PP**. The primary purpose in this discussion is to illustrate the technique and issues that arise in characterizing topological properties of motion.

Let **PP** be a smooth surface homeomorphic to a closed disk. For the purposes of our discussion here, references to the "boundary" and "interior" of **PP** will not be taken relative to the containing three-dimensional space (in which all of **PP** is on the boundary). Rather, by

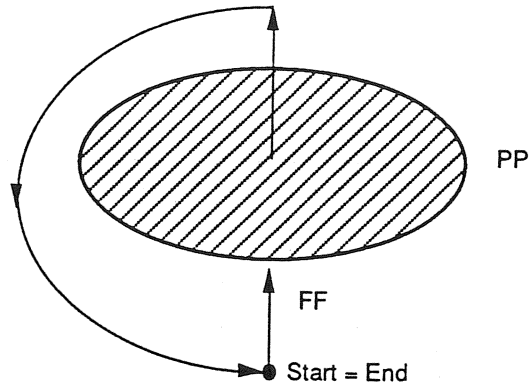


Figure 6.35 Going through and coming back

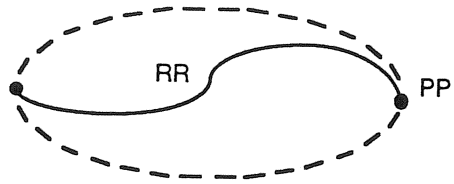
the boundary of **PP**, we mean the curve that forms the edge of **PP**; formally, the image of the boundary of the unit disk in two dimensions, under some homeomorphism between the disk and **PP**. The interior of **PP** is all points not in the boundary. **FF** is restricted to be a continuous fluent whose value at each time instant is a regular region. (We will give the definition for three-dimensional space; the definition for a two-dimensional object going through a curve is analogous.)

Note that whether **FF** goes through **PP** cannot be determined from the starting and ending positions of the fluent. In fact, **FF** may go through **PP** and yet start and end in the same position. (Figure 6.35).

We begin with the following definition: Let **PP** be as above. A region **RR** is a *divided neighborhood* of **PP** if (i) **RR** is a connected open set in the three-dimensional space; (ii) **RR** contains the interior of **PP**, but not the boundary of **PP**; and (iii) the difference **RR** – **PP** consists of two connected components (Figure 6.36). It is easily seen that we can assign labels "positive" and "negative" to the components of all divided neighborhoods of **PP** in a consistent way; that is, so that the positive components of any two divided neighborhoods intersect with each other, as do their negative components.

Let **X** be a point-valued fluent. We say that **X** threads **PP** once positively during the time interval $[T_0, T_1]$ if there is a divided neighborhood **NN** of **PP** satisfying the following conditions.

1. value_in(T_0, **X**) is in the negative component of **NN**.



PP — Solid Curve
RR — Interior of dotted curve

Figure 6.36 Divided neighborhood

2. value_in(T_1, **X**) is in the positive component of **NN**.

3. For all $T \in [T_0, T_1]$, value_in(T, **X**) is in **NN**.

We say that **X** threads **PP** negatively during $[T_0, T_1]$ if, for some divided neighborhood, **X** goes from the positive to the negative components of **NN**, while always staying in **NN**. We say that **X** does not thread **PP** if, for some divided neighborhood **NN**, **X** stays in **NN** during I and starts and ends in the same component of **NN**. (Note that this includes, as a special case, cases where **X** does not intersect **PP** at all during I.)

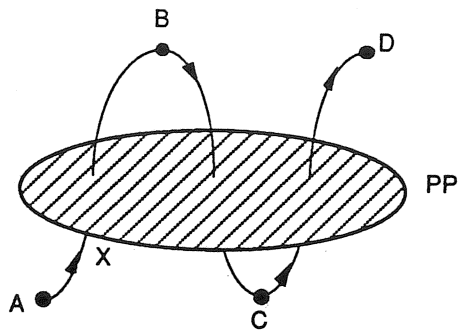
Note that these events are mutually exclusive during a single interval I; that is, if **X** threads **PP** positively in I, then it does not thread **PP** negatively in I, or fail to thread **PP** in I. However, any of these event types may be occur within any of the others; for example, it is possible for **X** to thread **PP** positively in I, but to thread **PP** negatively in some subinterval $I2$ of I. (Figure 6.37).

Now, let **PP** and **X** be as above, and let $[A, B]$ be a time interval such that neither value_in(A, **X**) nor value_in(B, **X**) are in **PP**. Assume further that **X** does not intersect the boundary of **PP** during the interval $[A, B]$. It can be shown that it is possible to break up the interval $[A, B]$ into a finite sequence of subintervals $[A_0 = A, A_1], [A_1, A_2] \ldots [A_{k-1}, A_k = B]$ such that in each interval $[A_i, A_{i+1}]$, exactly one of the following holds:

a. **X** threads **PP** once positively in $[A_i, A_{i+1}]$.

b. **X** threads **PP** once negatively in $[A_i, A_{i+1}]$.

c. **X** does not thread **PP** in $[A_i, A_{i+1}]$.



X threads PP positively between A and D but negatively between B and C

Figure 6.37 Positive and negative threadings

Assign the number 1 to intervals of type (a), -1 to intervals of type (b), and 0 to intervals of type (c). Add up all these numbers to get a total k (which may be positive, zero, or negative). Then we say that **X** threads **PP** positively k times in the interval $[A, B]$. It can be shown that this total k is the same for any subdivision of $[A, B]$ satisfying conditions (a) – (c).

Finally, let **FF** be a region-valued fluent. We say that **FF** goes through **PP** k times positively during I if the following holds: Let **X** be a continuous point-valued fluent. Moreover, let **X** stay in the interior of **FF** during I; that is, for any situation S in I, value_in(S,**X**) is an interior point of value_in(S,**FF**). Then X threads **PP** k times positively.

6.5 References

General: [McDermott 1987b] is a survey of work on commonsense spatial reasoning. [Fleck 1987] presents an alternative spatial topology for use in commonsense reasoning. [Randell and Cohn 1989] discuss and axiomatize a variety of topological and metrical operators.

Cognitive maps: Several research projects have addressed the problem of constructing and using cognitive maps. Kuipers' TOUR program [1977, 1978] and Davis's MERCATOR program [1986a] are

described in the text. Lavin's DYNAVU [1979] constructs a map of a landscape of Gaussian hills rising above a plane from simulated visual input, recording the coordinates and height of the peaks with some measure of uncertainty. Rowat's UTAK [1981] constructs a cognitive map of a two-dimensional layout of objects and uses it to plan the manipulation of objects. It used an occupancy array. McDermott's SPAM [McDermott and Davis 1984] creates a three-dimensional map from a series of symbolic constraints. SPAM uses multiple frames of references, whose relative positions are partially specified by interval bounds on length ratio, origin coordinates, and angles between coordinate axes. Figures are represented as unions and differences of cylinders with spherical endcaps. The NX program [Kuipers and Levitt 1988] constructs a cognitive map consisting of a collection of perceptually distinguishable places, with sensory-motor rules for getting from one to another, from simulated sonar input. Moravec [1988] uses a bit-map representation of space, with cells labeled by probability of occupancy.

Closely related is the problem of reconstructing a spatial arrangement from constraints. Ambler and Popplestone [1975] study combining constraints derived from coplanarity of faces of given polyhedra.

Physical reasoning: Some work on physical reasoning has studied geometric features important in physics. Much of this is discussed further in Chapter 7. De Kleer's NEWTON [1975], discussed in Section 6.2.6, represented curves in two dimensions with a distinguished vertical dimension in terms of the signs of their derivatives and the relative heights of distinguished points. He used this representation to find the behavior of a point object moving on such a curve. [Forbus 1979] considered a similar two-dimensional world where the ground is a polyline whose edges are specified in terms of the sign of the normal and the relative heights of their endpoints. He used this representation to find a physically meaningful decomposition of free space into regions bounded by the ground and horizontal and vertical lines, and to determine the behavior of elastic balls moving in this world. [Hayes 1978] studied representations for the shapes of containers of liquids in a three-dimensional space with a distinguished vertical. Hayes's representations combined physical properties with spatial properties; the spatial aspects of his representations mostly relate to topology and the vertical dimension. [Davis 1988] discussed a variety of spatial concepts necessary for qualitative descriptions of the dynamics of solid objects; in particular, differential surface properties. [Shoham 1985] analyses the differential motions possible for rigid objects constrained by other rigid objects in two dimensions. Recent work on the kinematics of mechanisms ([Gelsey 1987; Joskowicz 1987]) has been

largely concerned with finding the extended motions possible to a piece or collection of pieces in a three-dimensional mechanism. Faltings [1987a,b] discusses the use of configuration space in the analysis of mechanisms and gives a thorough analysis of the configuration space of pairs of polygonal objects, each with one degree of freedom, such as pairs of interlocking gears.

One physical-reasoning problem that has been studied in great depth is the "piano movers" problem: how to move an object around obstacles from a starting to an ending position. Heuristics for various forms of this problem are discussed in [Brooks 1982], [Thorpe 1984] and [Wallace 1984]. The problem has also been studied extensively from the point of view of computational geometry; [Schwartz Sharir, and Hopcroft, 1987] contains many important papers in this area.

Natural language: The interpretation of spatial information in natural-language text has been a central part of a number of AI programs, including [McDermott 1974], [Boggess 1979], [Novak 1977], [Riesbeck 1980], [Waltz 1980], and [Retz-Schmidt 1988].

Vision and robotics: Much of the work in spatial reasoning in AI has been done in the contexts of vision and of robotics. Vision research in this area is mainly concerned with representing figures and shapes so that they can be easily matched against images. [Ballard and Brown 1982] contains a fine survey of this research. The CSG representation discussed in Section 6.2.2 is modeled on that used by Brooks [1981], who applied constructive solid geometry to visual recognition. Robotics has mostly worked on the problem of determining the figure occupied by a manipulator, given joint positions, and of finding a method of getting a manipulator or an entire mobile robot with contents into a desired configuration. This research is surveyed in [Craig 1986].

Other computer science: Spatial reasoning is also central in other areas in computer science, particularly computer-aided design, computer graphics, and computational geometry. Of these, computer-aided design is the closest in its interests to AI. [Requicha 1980] surveys representations of three-dimensional figures, and [Requicha 1983] deals with questions of tolerances. Computational geometry seeks to find very efficient algorithms for geometrical problems; like most work in algorithms, it tends to focus more on issues of efficiency rather than expressiveness. [Hoffmann 1989] surveys shape representations from the point of computational geometry. As mentioned above, [Schwartz, Sharir, and Hopcroft 1987] contains many important papers on the piano-movers problem.

Psychology: Spatial reasoning has been studied extensively by cog-

nitive psychologists, possibly because of the ease of designing experiments. [Piaget and Inhelder 1967] studies the development of spatial reasoning in children. [Downs and Stea 1973a] surveys psychological studies of cognitive mapping. [Downs and Stea 1973b] is a collection of articles on spatial reasoning generally. [Tversky 1981] presents some interesting results on characteristic errors in cognitive maps. The nature of mental imagery has been the subject of substantial debate among cognitive scientists. The "pro-imagery" side is argued in [Kosslyn 1980]; the "anti-imagery" side is argued in [Pylyshyn 1984] and [Hinton 1979].

6.6 Exercises

(Starred problems are more difficult.)

1. Consider an occupancy-array spatial representation in which each region is marked in each cell with one of the following labels: F (full), P (partial), E (empty), FP (full or partial), EP (empty or partial, or DK (don't know).
   (a) Boolean operation on regions can be carried out cell by cell. That is, it is possible to compute functions like the intersection of two regions **AA** and **BB** by looping through each cell X in the array, and computing the intersection of **AA** and **BB** on X. To do this we would use rules like "If **AA** is full in X, and **BB** is full or partial in X, then A intersection B is full or partial in X." Such rules can be displayed in a table of **AA** values by **BB** values; the above rule would place "FP" as the value in the "F" row and "FP" column. Construct such tables for the intersection, union, and complementation operations.
   (b) Do the tables constructed in part a satisfy De Morgan's laws? Do they have the property that union can be distributed over intersection, and vice versa? Do they observe the law of associativity?
   (c) Describe an algorithm that takes as input two regions **AA** and **BB** and determines whether **AA** is a subset of **BB**. (Note: The answer can be "True," "False," or "Maybe.")
   (d) * Describe an algorithm that takes as input two regions **AA** and **BB** and returns upper and lower bounds on the minimal distance between **AA** and **BB**. Your algorithm should return a lower bound that is as tight as possible, given the information, and an upper bound that is within one grid length of the best-possible upper bound, given the information.

(e) *** Give an algorithm that gives a tight upper bound on the minimal distance between two regions. (Note: If you solve this problem, please inform the author of this book.)

2. Let **AA**,**BB**,**CC**,**DD** range over connected, bounded, regular, non-empty regions in the X-Z planes. We wish to define the concept of **AA** being "above" **BB**, and we have a number of desirable properties and proposed definitions. Let x(**P**) and z(**P**) be the x and z coordinates of point **P**.

   Definitions:

   (a) **AA** is above **BB** if, for every point **A** in **AA** and **B** in **BB**, z(**A**) > z(**B**).
   (b) **AA** is above **BB** if, for every point **A** in **AA**, there is a point **B** in **BB** such that z(**A**) > z(**B**).
   (c) **AA** is above **BB** if, for every point **B** in **BB** there is a point **A** in **AA** such that z(**A**) > z(**B**) and x(**A**) = x(**B**).
   (d) **AA** is above **BB** if z(**A**) > z(**B**) for every point **A** in **AA** and **B** in **BB** such that x(**A**) = x(**B**).

   Axioms:

   W. If **AA** is above **BB** then **BB** is not above **AA**.
   X. If **AA** is above **BB** and **CC** is a subset of **AA**, then **CC** is above **BB**.
   Y. If **AA** is above **BB** and **DD** is a subset of **BB**, then **AA** is above **DD**.
   Z. If **AA** is above **BB**, then, if **AA** is "moved downward" without intersecting **BB**, then the result of that motion is above **BB**. Formally, if **AA** is above **BB** and, for all $t \in [0, k]$, **AA** $- t\hat{z}$ is disjoint from **BB**, then **AA** $-k\hat{z}$ is above **BB**.

   (a) State which of these axioms are true under which of these definitions.
   (b) Can you find a definition of "above" that satisfies all these properties? Can you find such a definition that at all corresponds to the standard meaning of "above"?

3. Find the complete transition graph for the attainable states of the cart on the track in Figure 6.29.

4. Extend the CSG representation of a human in Table 6.8 by adding the right upper leg and its position relative to the torso. The constraints you give should have some distant relation to the variation possible for a human being and the range of positions that a human can achieve. Do not worry about achieving any great degree of precision; the representation is too crude to achieve it.

5. Figure 6.1 shows a shape approximated by two different polygons. Show how these two approximations can be expressed in the MERCATOR representation.

6. * Using the information in Table 6.9, find a lower bound on the distance between the lake and the road. (Your demonstration of the lower bound need not be rigorously complete. The lower bound need not be tight, but it should be greater than two feet.)