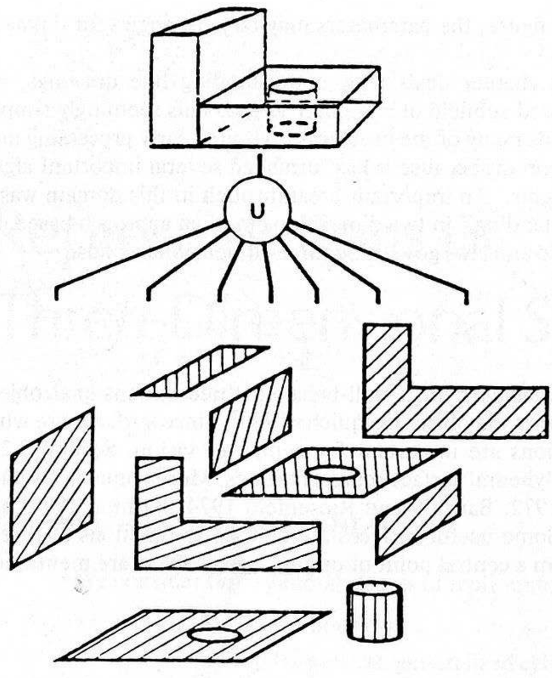


Fig. 9.1 A volume and the faces of a boundary representation.

opinion (Fig. 9.2). In short, any single definition of face is likely to be inadequate for some important application.

The availability of explicit representations of edges, faces, and vertices makes boundary representations quite useful in computer vision and graphics. The computational advantages of polyhedral surfaces are so great that they are often pressed into service as approximate representations of nonpolyhedra (Fig. 9.3).

An influential system for using face-based representations for planar polyhedral objects is the "winged edge" representation [Baumgart 1972]. Included in the system is an editor for creating complex polyhedral objects (such as that of Fig. 9.3) interactively. The system uses rules for construction based on the theorem of Euler that if V is the number of vertices in a polyhedron, E the number of edges, and F the number of faces, then $V - E + F = 2$. In fact, the formula can be extended to deal with non-simply connected bodies. The extended relation is $V - E + F = 2(B - H)$, with B being the number of bodies and H being the

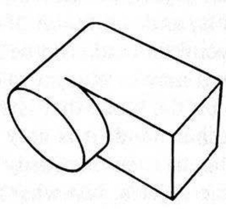


Fig. 9.2 What are the faces?

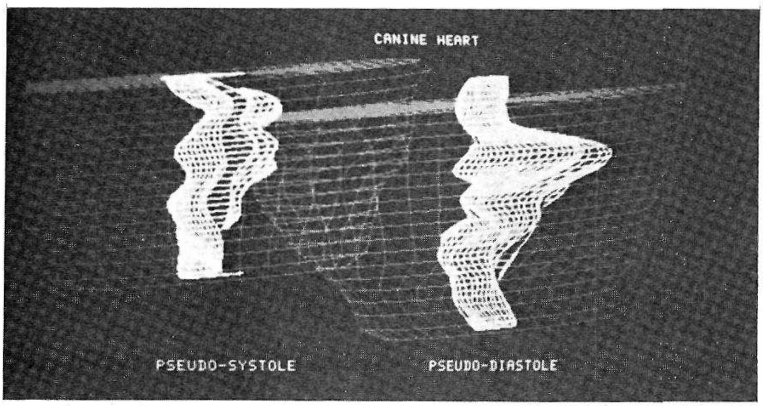


Fig. 9.3 A polyhedral approximation to a portion of a canine heart at systole and diastole. Both exterior (coarse grid) and interior surfaces (fine grid) are shown.

number of holes, or "handles," each resulting from a hole through a body [Lakatos 1976]. Baumgart's system uses these rules to oversee and check certain validity conditions on the constructions made by the editor.

The "winged edge" polyhedron representation achieves many desiderata for boundary representations in an elegant way. This representation is presented below to give a flavor of the features that have been traditionally found useful. Given as primitives the vertices, edges, faces, and polyhedra themselves, and given various relations between these primitives, one is naturally thinks of a record and pointer (relational) structure in which the pointers capture the binary relations and the records represent primitives and contain data about their locations or parameters.

In the winged edge representation, there are data structure records, or nodes, which contain fields holding data or links (pointers) to other nodes. An example using this structure to describe a tetrahedron is shown in Fig. 9.4. There are four kinds of nodes: vertices, edges, faces, and bodies. To allow convenient access to these nodes, they are arranged in a circular doubly linked list. The body nodes are actually the heads of circular structures for the faces, edges, and vertices of the body. Each face points to one of its perimeter edges, and each vertex points to one of the edges impinging on it. Each edge node has links to the faces on each side of it, and the vertices at either end.

Figure 9.4 shows only the last-mentioned links associated with each edge node. The reader may notice the similarity of this data structure with the data structure for region merging in Section 5.4. They are topologically equivalent. Each edge also has associated four links which give the name "winged edge" to the representation. These links specify neighboring edges in order around the two faces which are associated with the edge. The complete link set for an edge is shown in Fig. 9.5, together with the link information for bodies, vertices, and faces. To allow unambiguous traversal around faces, and to preserve the notion of

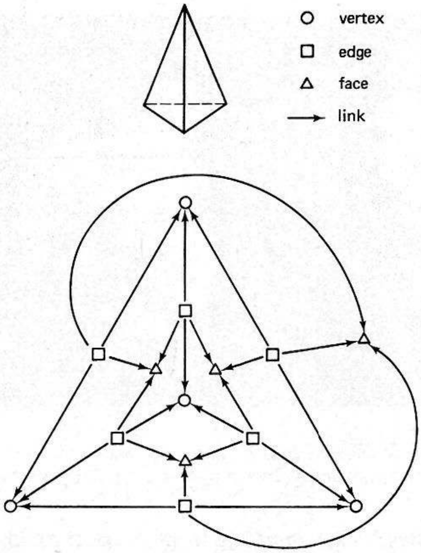


Fig. 9.4 A subset of edge links for a tetrahedron using the "winged edge" representation.

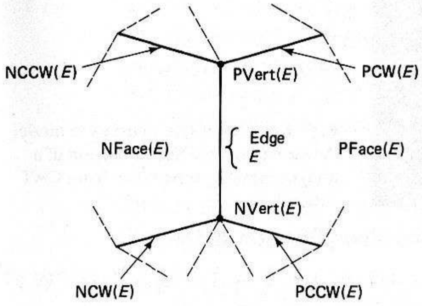
interior and exterior of a polyhedron, a preferential ordering of vertices and lines is picked (counterclockwise, say, as seen from outside the polyhedron).

Data fields in each vertex allow storage of three-dimensional world coordinates, and also of three-dimensional perspective coordinates for display. Each node has fields specifying its node type, hidden line elimination information, and other general information. Faces have fields for surface normal vector information, surface reflectance, and color characteristics. Body nodes carry links to relate them to a tree structure of bodies in a scene, allowing for hierarchical arrangement of subbodies into complex bodies. Thus body node data describe the scene structure; face node data describe surface characteristics; edge node data give the topological information needed to relate faces, edges, and vertices; and vertex node data describe the three-dimensional vertex location.

This rich and redundant structure lends itself to efficient calculation of useful functions involving these bodies. For instance, one can easily follow pointers to extract the list of points around a face, faces around a point, or lines around a face. Winged edges are not a universal boundary representation for polyhedra, but they do give an idea of the components to a representation that are likely to be useful. Such a representation can be made efficient for accessing all faces, edges, or vertices; for accessing vertex or edge perimeters; for polyhedron building; and for splitting edges and faces (useful in construction and hidden-line picture production, for instance).

9.2.2 Surfaces Based on Splines

The natural extension of polyhedral surfaces is to allow the surfaces to be curved. However, with an arbitrary number of edges for the surface, the interpolation of



(a)

Boundary Representation Node Accessing Functions

1. To enter and traverse Face ring of a body:
   NextFace, PreviousFace: Body or Face → Face
2. To enter and traverse Edge ring of a body:
   NextEdge, PreviousEdge: Body or Edge → Edge
3. To enter and traverse Vertex ring of a body:
   NextVert, PreviousVert: Body or Vertex → Vertex
4. First Edge of a Face:
   FirstEdge: Face → Edge
5. FirstEdge of a Vertex:
   FirstEdge: Vertex → Edge
6. Faces of an Edge: [see diagram in (a)]
   N(ext) Face, P(revious) Face: Edge → Face
7. Vertices of an Edge: [see diagram in (a)]
   N(extVert, P(revious) Vert: Edge → Vertex
8. Neighboring Wing Edges of an Edge: [see diagram in (a)]
   NCW, NCCW: Edge → Edge (NFace Edge Clockwise, NFace Edge Counterclockwise)
   PCW, PCCW: Edge → Edge (PFace Edge Clockwise, PFace Edge Counterclockwise

(b)

Fig. 9.5 (a) Node accessing functions. (b) Semantics of winged edge functions.

interior face points becomes impractically complex. For that reason, the number of edges for a curved face is usually restricted to three or four.

A general technique for approximating surfaces with four-sided surface *patches* is that of Coons [Coons 1974]. Coons specifies the four sides of the patch with polynomials. These polynomials are used to interpolate interior points. Although this is appropriate for synthesis, it is not so easy to use for analysis. This is because of the difficulty of registering the patch edges with image data. A given surface will admit to many patch decompositions.

An attractive representation for patches is splines (Fig. 9.6). In general, two-dimensional spline interpolation is complex: For two parameters u and v interpolate with

$$\mathbf{x}(u, v) = \sum_i \sum_j V_{ij} B_{ij}(u, v) \tag{9.1}$$

similar to Eq. (8.4). However, for certain applications a further simplification can be made. In a manner analogous to (8.9) define a grid of knot points $\mathbf{v}_{ij}$ corresponding to $\mathbf{x}_{ij}$ and related by

$$\mathbf{x}_{ij} = M\mathbf{v}_{ij} \tag{9.2}$$

Now rather than interpolating in two dimensions simultaneously, interpolate in one direction, say t, to obtain

$$\mathbf{x}_{ij}(t) = [t^3 \quad t^2 \quad t \quad 1][C][\mathbf{v}_{i-1,j_0}, \mathbf{v}_{i,j_0}, \mathbf{v}_{i+1,j_0}, \mathbf{v}_{i+2,j_0}]^T \tag{9.3}$$

for each value of j. Now compute $\mathbf{v}_{ij}(t)$ by solving

$$\mathbf{x}_{ij}(t) = M\mathbf{v}_{ij}(t) \tag{9.4}$$

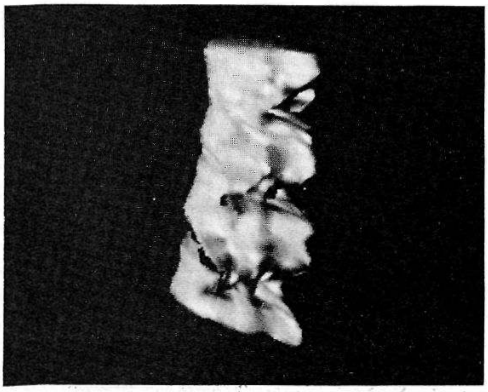


Fig. 9.6 Using spline curves to model the surface of an object: a portion of a human spinal column taken from CAT data.

for each value of t. Finally, interpolate in the other direction and solve:

$$\mathbf{x}_{ij}(s, t) = [s^3 \quad s^2 \quad s \quad 1][C][\mathbf{v}_{i-1,j}(t), \mathbf{v}_{i,j}(t), \mathbf{v}_{i+1,j}(t), v_{i+2,j}(t)] \quad (9.5)$$

This is the basis for the spline filtering algorithm discussed in Section 3.2.3.

Some advantages of spline surfaces for vision are the following.

1. The spline representation is economical: the space curves are represented as a sparse set of knot points from which the underlying curves can be interpolated.
2. It is easy to define splines interactively by giving the knot points; reference representations may be built up easily.
3. It is often useful to search the image in a direction perpendicular to the model reference surface. This direction is a simple function of the local knot points.

9.2.3 Surfaces That Are Functions on the Sphere

Some surfaces can be expressed as functions on the "Gaussian sphere." (the distance from the origin to a point on the surface is a function of the direction of the point, or of its longitude and latitude if it were radially projected on a sphere with the center at the origin.) This class of surfaces, although restricted, is useful in some application areas [Schudy and Ballard 1978, 1979]. This section explores briefly two schemes for representation of these surfaces. The first specifies explicitly the distance of the surface from the origin for a set of vector directions from the origin. The second is akin to Fourier descriptors; an economically specified set of coefficients characterizes the surface with greater accuracy as the number of coefficients increases.

Direction–Magnitude Sets

One approximation to a spherical function is to specify a number of three-dimensional direction vectors from the origin and for each a magnitude. This is equivalent to specifying a set of (θ, ϕ, ρ) points in a spherical coordinate system (Appendix 1). These points are on the surface to be represented; connecting them yields an approximation.

It is often convenient to represent directions as points on the unit (Gaussian) sphere centered on the origin. The points may be connected by straight lines to form a polyhedron with triangular, hexagonal or rhomboidal faces. Moving the points on the sphere out (or in) by their associated magnitude distorts this polyhedron, moving its vertices radically out or in.

The spherical function determines the distance of face vertices from the origin. Resolution at the surface increases with the number of faces. An approximately isotropic distribution of directions over the surface may be obtained by placing the face vertices (directions) in accordance with "geodesic dome"–like calculations which make the faces approximately equilateral triangles [Clinton 1971].

Although the geodesic tesselation of the sphere's surface is more complex than a straightforward (latitude and longitude, say) division, its pleasant properties of isotropy and display [Brown 1979a; 1979b; Schudy and Ballard 1978] sometimes recommend it. Some example shapes indicating the range of representable surfaces are given in Fig. 9.7. Methods for tesselating the sphere are given in Appendix 1.

Spherical Harmonic Surfaces

In two dimensions, Fourier coefficients can give approximations to certain curved boundaries (Section 8.3.4). Analogously in three dimensions, a set of orthogonal functions may be used to express a closed boundary as a set of coefficients when the boundary is a function on the sphere. One such decomposition is *spherical harmonics.* Low order coefficients capture gross shape characteristics; higher order coefficients represent surface shape variations of higher spatial frequency. The function with $m = 0$ is a sphere, the three with $m = 1$ represent translation about the origin, the five with $m = 2$ are similar to prolate and oblate spheroids, and so forth, the lobedness of the surfaces increasing with m. A sample three dimensional shape and its "description" is shown in Fig. 9.8.

Spherical harmonics are analogs on the sphere of Fourier functions on the plane; like Fourier functions, they are smooth and continuous to every order. They may be parameterized by two numbers, m and n; thus they are a doubly infinite set of functions which are continuous, orthogonal, single-valued, and complete on the

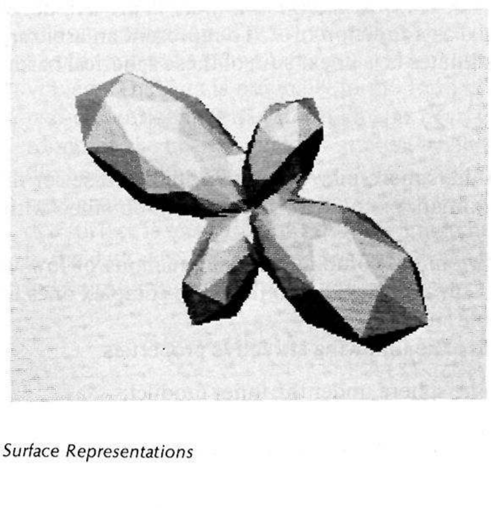


Fig. 9.7 Sample surfaces described by some 320 triangular facets in a geodesic tesselation.

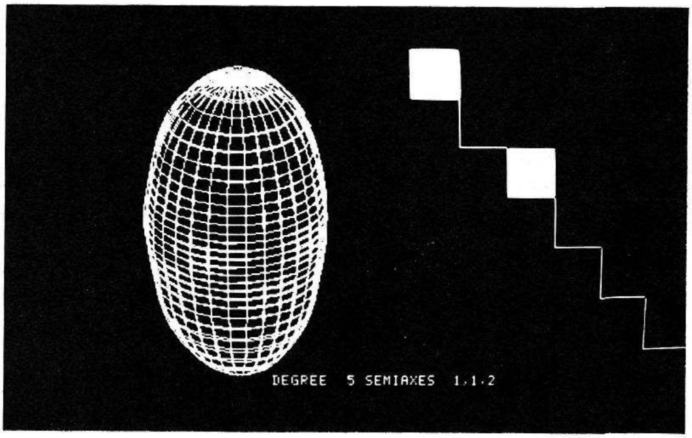


Fig. 9.8 A spherical harmonic function description of an ellipsoid. Coefficients are displayed on the right as grey levels in the matrix format

$$
\begin{array}{lllll}
u_{00} & & & & \\
 & u_{01} & v_{11} & & \\
 & u_{11} & u_{02} & v_{12} & v_{22} \\
 & & u_{12} & & \\
 & & u_{21} & &
\end{array}
$$

sphere. In combination, the harmonics can thus produce all "well-behaved" spherical functions.

The spherical harmonic functions U_{mn} (θ, ϕ) and V_{mn} (θ, ϕ) are defined in polar coordinates by:

$$U_{mn}(\theta, \phi) = \cos(n\theta) \sin^n(\phi) P(m, n, \cos(\phi)) \tag{9.6}$$

$$V_{mn}(\theta, \phi) = \sin(n\theta) \sin^n(\phi) P(m, n, \cos(\phi)) \tag{9.7}$$

with $m = 0, 1, 2, ..., M$; $n = 0, 1, ..., m$. Here $P(m, n, x)$ is the nth derivative of the mth Legendre polynomial as a function of x. To represent an arbitrary shape, let the radius R in polar coordinates be a linear sum of these spherical harmonics:

$$R(\theta, \phi) = \sum_{m=0}^{M} \sum_{n=0}^{m} A_{mn} U_{mn}(\theta, \phi) + B_{mn} V_{mn}(\theta, \phi) \tag{9.8}$$

Any continuous surface on the sphere may be represented by a set of these real constants; reasonable approximations to heart volumes are obtained with $m \leqslant 5$ [Schudy and Ballard 1979].

Figure 9.9 shows a few simple combinations of functions of low values of (m, n). The sphere, or $(0, 0)$ surface, is added to the more complex ones to ensure positive volumes and drawable surfaces.

Spherical harmonics have the following attractive properties.

1. They are orthogonal on the sphere under the inner product;

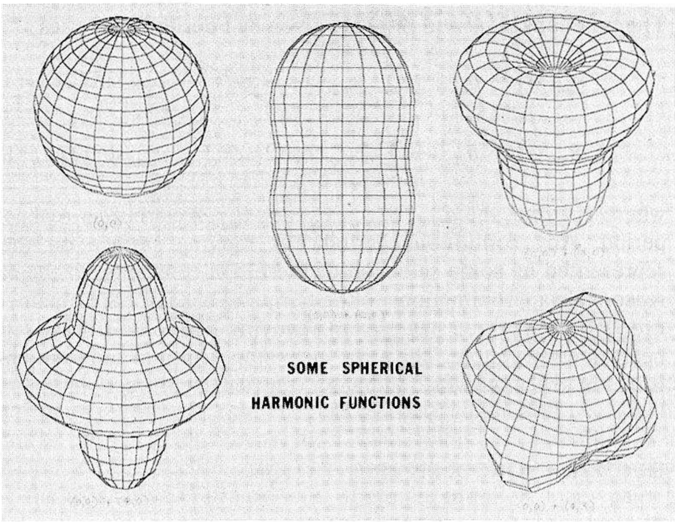


Fig. 9.9 Simple combinations of functions.

$$(u, v) = \int uv \sin\phi \; d\theta \; d\phi$$

2. The functions are arranged in increasing order of spatial complexity.
3. The whole set is complete; any twice-differentiable function on the sphere can be approximated arbitrarily closely.

Spherical harmonics can provide compact, nonredundant descriptions of surfaces that are useful for analysis of shape, but are less useful for synthesis. The principal disadvantages are that the primitive functions are not necessarily related to the desired final shape in an intuitive way, and changing a single coefficient affects the entire resulting surface.

An example of the use of spherical harmonics as a volume representation is the representation of heart volume [Schudy and Ballard 1978, 1979]. In extracting a volume associated with the heart from ultrasound data, a large mass of data is involved. The data is originally in the form of echo measurements taken in a set of two-dimensional planes through the heart. The task is to choose a surface surrounding the heart volume of interest by optimization techniques that will fit three dimensional time-varying data. The optimization involved is to find the best coefficients for the spherical harmonics that define the surface. The goodness of fit of a surface is measured by how well it matches the edge of the volume as it appears in the data slices. To extend spherical harmonics to time-varying periodic data, let the radius R in polar coordinates be a linear sum of these spherical harmonics:

$$R(\theta, \phi, t) = \sum_{m=0}^{M} \sum_{n=0}^{m} A_{mn}(t)\, U_{mn}(\theta, \phi) + B_{mn}(t)\, V_{mn}(\theta, \phi) \tag{9.9}$$

The functions $A(t)$ and $B(t)$ are given by Fourier time series:

$$A_{mn}(t) = a_{mno} + \sum_{i=1}^{I} a_{mni} \cos (2\pi t/\tau) + b_{mni} \sin (2\pi t/\tau) \tag{9.10}$$

$$B_{mn}(t) = b_{mno} + \sum_{i=1}^{I} c_{mni} \cos (2\pi t/\tau) + d_{mni} \sin (2\pi t/\tau) \tag{9.11}$$

where t is time, the a_{mni}, b_{mni}, c_{mni}, and d_{mni} are arbitrary real constants, and τ the period. Any continuous periodically moving surface on the sphere may be represented by some selection of these real constants; in the cardiac application, reasonable approximations to the temporal behavior are obtained with $t \leqslant 3$. Figure 9.10 shows three stages from a moving-harmonic-surface representation of the heart in early systole. The atria, at the top, contract and pump blood into the ventricles below, after which there is a ventricular contraction.

9.3 GENERALIZED CYLINDER REPRESENTATIONS

The volume of many biological and manufactured objects is naturally described as the "swept volume" of a two-dimensional set moved along some three-space curve. Figure 9.11 shows a "translational sweep" wherein a solid is represented as the volume swept by a two-dimensional set when it is translated along a line. A "rotational sweep" is similarly defined by rotating the two-dimensional set around an axis. In "three-dimensional sweeps," volumes are swept. In a "general" sweep scheme, the two-dimensional set or volume is swept along an arbitrary space curve, and the set may vary parametrically along the curve [Binford 1971; Soroka and Bajcsy 1976; Soroka 1979a; 1979b; Shani 1980]. General sweeps are quite a popular representation in computer vision, where they go by the name *generalized cylinders* (sometimes "generalized cones").

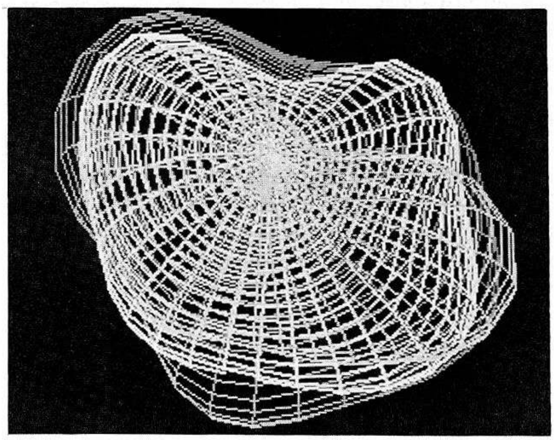


Fig. 9.10 Three stages from a moving harmonic surface (*see text and color insert*).

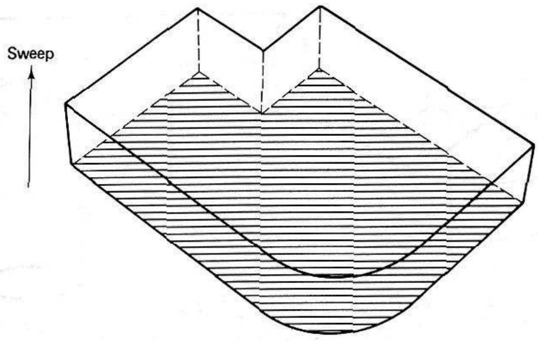


Fig. 9.11 A translational sweep.

A *generalized cylinder (GC)* is a solid whose axis is a 3-D space curve (Fig. 9.12a). At any point on the axis a closed cross section is defined. A usual restriction is that the axis be normal to the cross section. Usually it is easiest to think of an axis space curve and a cross section point set function, both parameterized by arc length along the axis curve. For any solid, there are infinitely many pairs of axis and cross section functions that can define it.

Generalized cylinders present certain technical subtleties in their definition. For instance, can it be determined whether any two cross sections intersect, as they would if the axis of a circular cylinder were sharply bent (Fig. 9.12b)? If the solid is defined as the volume swept by the cross section, there is no conceptual or computational problem. A problem might occur when computing the surface of such an object. If the surface is expressed in terms of the axis and cross-section functions (as below), the domain of objects must be limited so that the boundary formula indeed gives only points on the boundary.

Generalized cylinders are intuitive and appealing. Let us grant that "pathological" cases are barred, so that relatively simple mathematics is adequate for representing them. There are still technical decisions to make about the representation. The axis curve presents no difficulties, but a usable representation for the cross-section set is often not so straightforward. The main problem is to choose a usable coordinate system in which to express the cross section.

9.3.1 Generalized Cylinder Coordinate Systems and Properties

Two mathematical functions defining axis and cross section for each point define a unique solid with the "sweeping" semantics described above. In a fixed Cartesian coordinate system x, y, z, the axis may be represented parametrically as a function of arc length s:

$$\mathbf{a}(s) = (x(s), y(s), z(s)) \tag{9.12}$$

It is convenient to have a local coordinate system defined with origin at each point of $\mathbf{a}(s)$. It is in this coordinate system that the cross section is defined. This system may change in orientation as the axis winds through space, or it may be most natural for it not to be tied to the local behavior of the axis. For instance, imagine tying a knot in a solid rubber bar of square cross section. The cross section

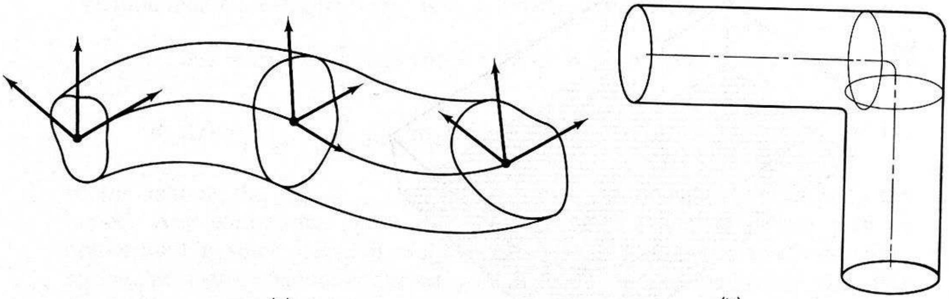


Fig. 9.12 (a) A generalized cylinder and some cross-sectional coordinate systems. (b) A possibly "pathological" situation. Cross sections may be simply described as circles centered on the axis, but then their intersection makes volume calculations (for instance) less straightforward.

will stay approximately a square, and (this is the point) will remain approximately fixed in a coordinate system that twists and turns through space with the axis of the bar. On the other hand, imagine bolt threads. They can be described by a single cross section that stays fixed in a coordinate system that rotates as it moves along the straight axis of the bolt. There is no a priori reason to suppose that such a useful local coordinate system should twist along the GC axis.

A coordinate system that mirrors the local behavior of the GC axis space curve is the "Frenet frame," defined at each point on the GC axis. This frame provides much information about the GC-axis behavior. The GC axis point forms the origin, and the three orthogonal directions are given by the vectors $(\boldsymbol{\xi}, \boldsymbol{\nu}, \boldsymbol{\zeta})$, where

$\boldsymbol{\xi}$ = unit vector tangent axis

$\boldsymbol{\nu}$ = unit vector direction of center of curvature of axis

normal curve

$\boldsymbol{\zeta}$ = unit vector direction of center of torsion of axis

Consider the curve to be produced by a point moving at constant speed through space; the distance the point travels is the parameter of the space curve [O'Neill 1966]. Since ξ is of constant length, its derivative measures the way the GC axis turns in space. Its derivative ξ' is orthogonal to ξ and the length of ξ' measures the curvature κ of the axis at that point. The unit vector in the direction of ξ' is ν. Where the curvature is not zero, a binormal vector ζ orthogonal to ξ and ν is defined. This binormal ζ is used to define the torsion τ of the curve. The vectors ξ, ν, ζ obey Frenet's formulae:

$$\begin{aligned} \boldsymbol{\xi}' &= \kappa\boldsymbol{\nu} \\ \boldsymbol{\nu}' &= -\kappa\boldsymbol{\xi} + \tau\boldsymbol{\zeta} \\ \boldsymbol{\zeta}' &= -\tau\boldsymbol{\nu} \end{aligned} \tag{9.13}$$

where

$$\kappa = \text{curvature} = -\boldsymbol{\nu}' \cdot \boldsymbol{\xi} = \boldsymbol{\nu} \cdot \boldsymbol{\xi}' \tag{9.14}$$

$$\tau = \text{torsion} = \boldsymbol{\nu}' \cdot \boldsymbol{\zeta} = -\boldsymbol{\nu} \cdot \boldsymbol{\zeta}' \tag{9.15}$$

The Frenet frame gives good information about the axis of the GC, but it has certain problems. First, it is not well defined when the curvature of the GC axis is zero. Second, it may not reflect known underlying physical principles that generate the cross sections (as in the bolt thread example). A solution, adopted in [Agin 1972, Shani 1980], is to introduce an additional parameter that allows the cross section to rotate about the local axis by an arbitrary amount. With this additional degree of freedom comes an additional problem: How are successive cross sections registered? Figure 9.13 shows two solutions in addition to the Frenet frame solution.

The cross sectional curve is usually defined to be in the $\boldsymbol{\nu}$–$\boldsymbol{\zeta}$ plane, normal to $\boldsymbol{\xi}$, the local GC axis direction. The cross section may be described as a point set in this plane, using inequalities expressed in the $\boldsymbol{\nu}$–$\boldsymbol{\zeta}$ coordinate system. The cross section boundary (outline curve) may be used instead, parameterized by another parameter r. Let this curve be given by

$$\text{cross section boundary} = (x(r, s),\ y(r, s))$$

The dependence on s reflects the fact that the cross section shape may vary along the GC axis. The expression above is in world coordinates, but should be moved to

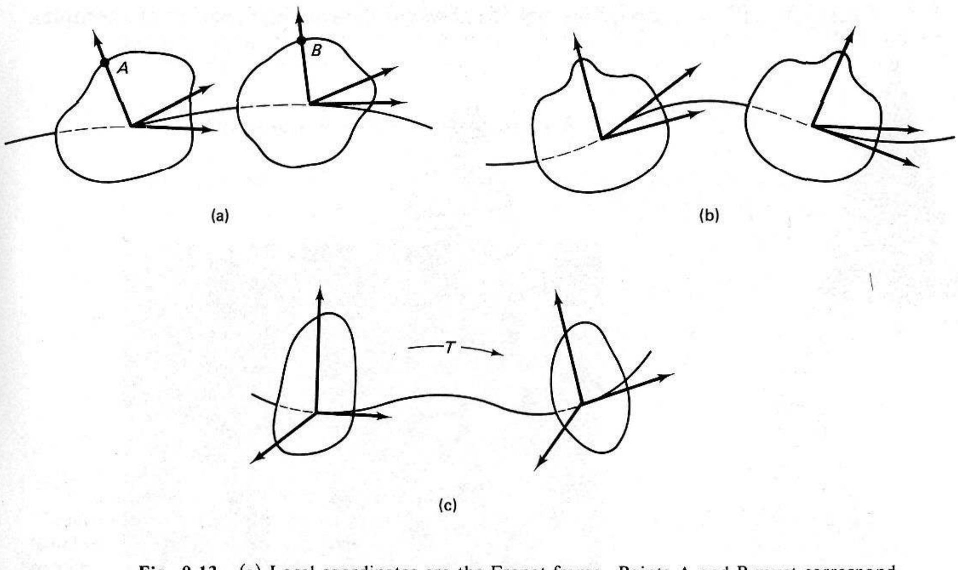


Fig. 9.13 (a) Local coordinates are the Frenet frame. Points A and B must correspond. (b) Local coordinates are determined by the cross sectional shape. (c) Local coordinates are determined by a heuristic transformation from world coordinates.

the local coordinates on the GC axis. A transformation of coordinates allows the GC boundary to be expressed (if the GC is well behaved) as

$$B(r, s) = \mathbf{a}(s) + x(r, s)\,\boldsymbol{\nu}(s) + y(r, s)\,\boldsymbol{\zeta}(s) \tag{9.16}$$

One of the advantages of the generalized cylinder representation is that it allows many parameters of the solid to be easily calculated.

- In matching the GC to image data it is often necessary to search perpendicular to a cross section. This direction is given from $x(r, s)$, $y(r, s)$ by $((dy/ds)\boldsymbol{\nu}, -(dx/ds)\boldsymbol{\zeta})$.
- The area of a cross section may be calculated from Eq. (8.16).
- The volume of a GC is given by the integral of: the area as a function of the axis parameter multipled by the incremental path length of the GC axis, i.e.,

$$\text{volume} = \int_0^L \text{area}(s)\; ds$$

9.3.2 Extracting Generalized Cylinders

Early work in biological form analysis provides an example of the process of fitting a GC to real data and producing a description [Agin 1972]. One of the goals of this work was to infer the stick figure skeleton of biological forms for use in matching models also represented as skeletons. In Fig. 9.14 the process of inferring the axis from the original stripe three-dimensional data is shown; the process iterates toward a satisfactory fit, using only circular cross sections (a common constraint with "generalized" cylinders). Figure 9.15 shows the data and the analysis of a complex

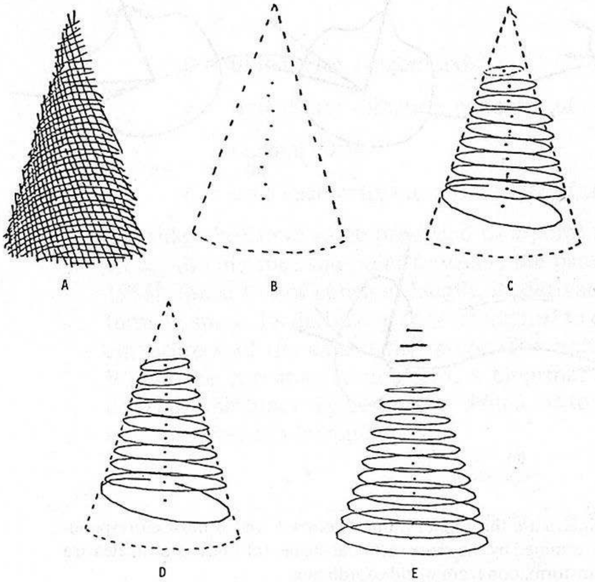
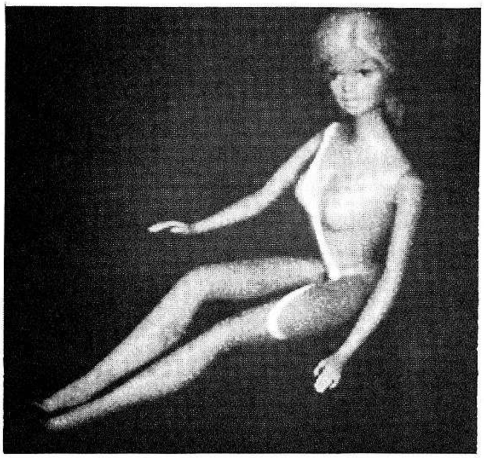
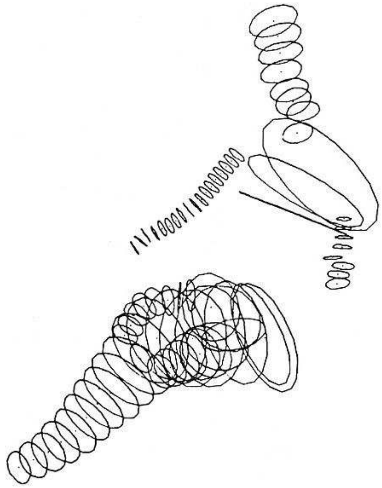


Fig. 9.14 Stages in extracting a generalized cylinder description for a circular cone. (a) Front view. (b) Initial axis estimate. (c) Preliminary center and axis estimate. (d) Cone with smoothed radius function. (e) Completed analysis.



(a)



(b)

Fig. 9.15 (a) TV image of a doll. (b) Completed analysis of doll.

biological form. In real data, complexly interrelated GCs are hard to decompose into satisfactory subparts. Without that, the ability to form a satisfactory articulated skeleton is severely restricted.

In later work, GCs with spline-based axes and cross sections were used to model organs of the human abdomen [Shani 1980]. Figure 9.16 shows a rendition of a GC fit to a human kidney.

9.3.3 A Discrete Volumetric Version of the Skeleton

An approximate volume representation that can be quite useful is based on an articulated wire frame skeleton along which spheres (not cross sections) are placed.

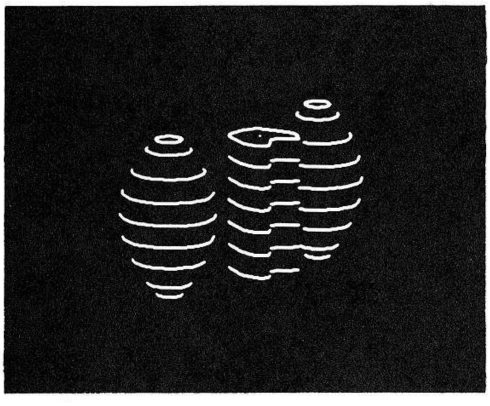


Fig. 9.16 Generalized cylinder representation of two kidneys and a spinal column. This coarse, nominal model is refined during examination of CAT data (see Fig. 9.6).

This representation has some of the flavor of an approximate sweep representation. An example of the use of such a representation and a figure are given in Section 7.3.4. This representation was originally conceived for graphics applications (the spheres look the same from any viewpoint) [Badler and Bajcsy 1978]. Collision detection is easy, and three-dimensional objects can be decomposed into spheres automatically [O'Rourke and Badler 1979]. From the spheres, the skeleton may be derived, and so may the surface of the solid. This representation is especially apt for many computer vision applications involving nonrigid bodies if strict surface and volumetric accuracy is not necessary [Badler and O'Rourke 1979].

9.4 VOLUMETRIC REPRESENTATIONS

Most world objects are solids, although usually only their surfaces are visible. A representation of the objects in terms of more primitive solids is often useful and can have pleasant properties of terseness, validity, and sometimes ease of computation. The representations given here are presented in order of increasing generality; constructive solid geometry includes cell decomposition, which in turn includes spatial occupancy arrays.

Algorithms for processing volume-based representations are often of a different flavor than surface-based algorithms. We give some examples in Section 9.4.4. Objects represented volumetrically can be depicted on raster graphics devices by a "ray-casting" approach in which a line of sight is constructed through the viewing plane for a set of raster points. The surface of the solid at its intersection with the line of sight determines the value of the display at the raster point. Ray casting can produce hidden-line and shaded displays; graphics is only one of its applications (Section 9.4.4).

9.4.1 Spatial Occupancy

Figure 9.17 shows that three-dimensional spatial occupancy representations are the three-dimensional equivalent of the two-dimensional spatial occupancy representations of Chapter 8. Volumes are represented as a three-dimensional array of cells which may be marked as filled with matter or not. Spatial occupancy arrays can require much storage if resolution is high, since space requirements increase as the cube of linear resolution. In low-resolution work with irregular objects, such as arise in computer-aided tomography, spatial occupancy arrays are very common. It is sometimes useful to convert an exact representation into an approximate spatial occupancy representation. Slices or sections through objects may be easily produced. The spatial occupancy array may be run-length encoded (in one dimension), or coded as blocks of different sizes; such schemes are actually cell-decomposition schemes (Section 9.4.2).

With the declining cost of computer memory, explicit spatial occupancy arrays may become increasingly common. The improvement of hardware facilities for parallel computation will encourage the development of parallel algorithms to compute properties of solids from these representations.

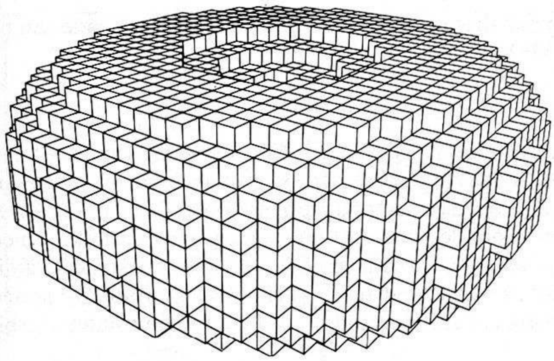


Fig. 9.17 A solid (the shape of a human red blood cell) approximated by a volume occupancy array.

9.4.2 Cell Decomposition

In cell decomposition, cells are more complex in shape but still "quasi-disjoint" (do not share volumes), so the only combining operation is "glue" (Fig. 9.18). Cells are usually restricted to have no holes (they are "simply connected"). Cell decompositions are not particularly concise; their construction (especially for curved cells) is best left to programs. It seems difficult to convert other representations exactly into cell decompositions. Two useful cell decompositions are the "oct-tree" [Jackins and Tanimoto 1980] and the kd-tree [Bentley 1975]. They both can be produced by recursive subdivision of volume; these schemes are the three-dimensional analogs of pyramid data structures for two dimensional binary images.

The quasi-disjointness of cell-decomposition and spatial-occupancy primitives may be helpful in some algorithms. Mass properties (Section 9.4.4) may be computed on the components and summed. It is possible to tell whether a solid is connected and whether it has voids. Inhomogeneous objects (such as human anatomy inside the thorax) can be represented easily with cell decomposition and spa-

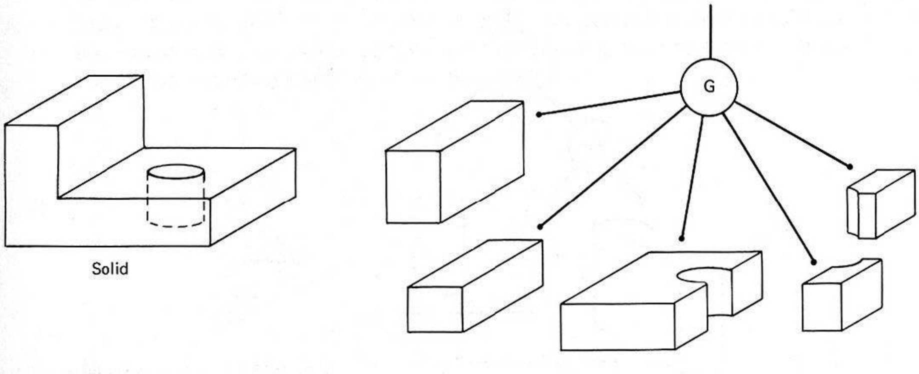


Fig. 9.18 A volume and its cell decomposition.

tial occupancy. The CT number (transparency to x-rays) or a material code can be kept in a cell instead of a single bit indication of "solid or space."

9.4.3 Constructive Solid Geometry

Figure 9.19 shows one constructive solid geometry (CSG) scheme [Voelcker and Requicha 1977; Boyse 1979]. Solids are represented as compositions, via set operations, of other solids which may have undergone rigid motions. At the lowest level are primitive solids, which are bounded intersections of closed half-spaces defined by some $F(x, y, z) \geqslant 0$, where F is well-behaved (e.g., analytic). Usually, primitives are entities such as arbitrarily scaled rectangular blocks, arbitrarily scaled cylinders and cones, and spheres of arbitrary radius. They may be positioned arbitrarily in space.

Figure 9.20 shows a parameterized representation [Marr and Nishihara 1978; Nishihara 1979] based on shapes (here cylinders) that might be extracted from an image.

A CSG representation is an expression involving primitive solid and set operators for combination and motion.

<CSG*Rep*> ::= <primitive solid> |
MOVE <CSG Rep> BY <Motion Params> |
<CSG Rep> <Combine Op> <CSG Rep>

The combining operators are best taken to be *regularized* versions of set union, intersection, and difference (the complement is a possible operator, but it allows unbounded solids from bounded primitives).

Regularity is a fundamental property of any set of points that models a solid. In a given space, a set X is regular if $X = kiX$, where k and i denote the *closure* and *interior* operators. Intuitively, a regular set has no isolated or dangling boundary points. The regularization r of a set X is defined by $rX = kiX$. Regularization informally amounts to taking what is inside a set and covering that with a tight skin. Regular sets are not closed under conventional set operations, but *regularized*

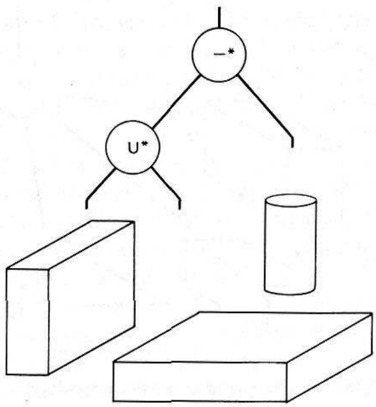


Fig. 9.19 Constructive solid geometry for the volume of Fig. 9.18.

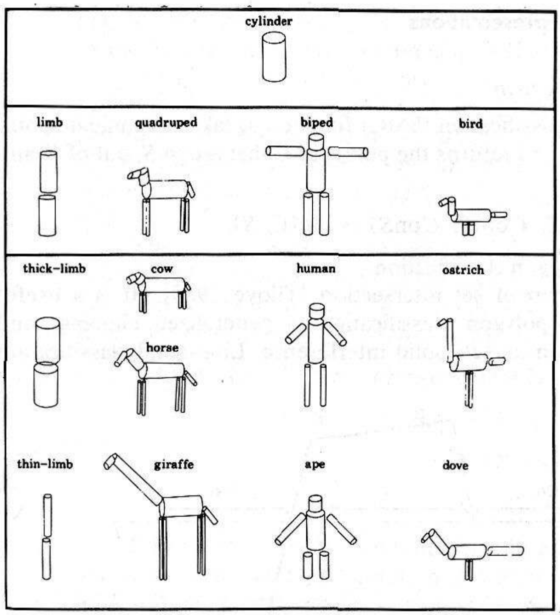


Fig. 9.20 A parameterized constructive representation for animal shapes.

operators do preserve regularity. Regularized operators are defined by

$$X \langle \mathrm{OP} \rangle * Y = r(X \langle \mathrm{OP} \rangle Y)$$

Regularity and regularized set operators provide a natural formalization of the dimension-preserving property exhibited by many geometric algorithms, thus obviating the need to enumerate many annoying "special cases." Figure 9.21 illustrates conventional versus regularized intersection of two sets that are regular in the plane.

If the primitives are unbounded, checking for boundedness of an object can be difficult. If they are bounded, any CSG representation is a valid volume representation. CSG can be inefficient for some geometric applications, such as a line drawing display. (Converting the CSG representation to a boundary representation is the one way to proceed; see Section 9.4.4.)

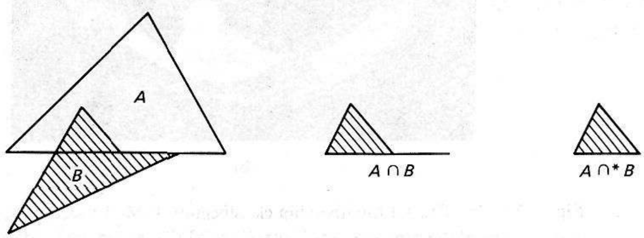


Fig. 9.21 Conventional ($\cap$) and regularized ($\cap^*$) polygon intersection.

9.4.4 Algorithms for Solid Representations

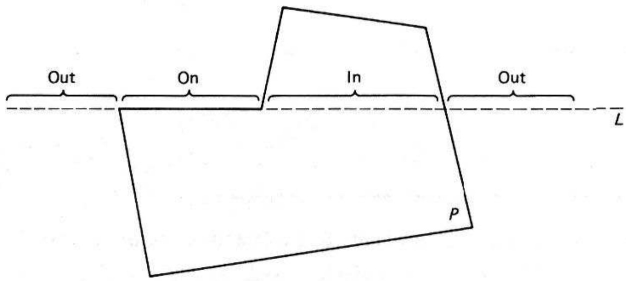
Set Membership Classification

The set membership classification (SMC) function M takes a candidate point set C and a reference set S, and returns the points of C that are in S, out of S, and on the boundary of S.

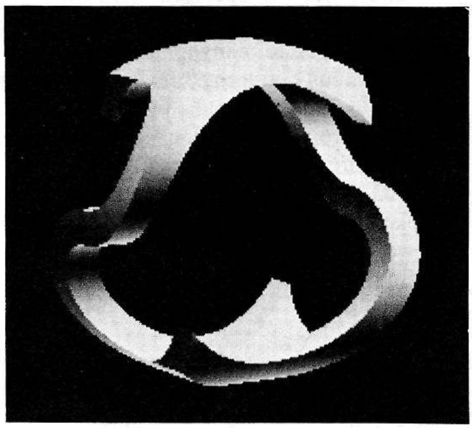
$$(C\text{in}S,\ C\text{out}S,\ C\text{on}S) := M(C, S)$$

Figure 9.22a shows line–polygon classification.

SMC is a generalization of set intersection [Tilove 1980]. It is a useful geometric utility; polygon–polygon classification is generalized clipping, and volume–volume classification detects solid interference. Line–solid classification



(a)



(b)

Fig. 9.22 (a) The set membership classification (SMC) function M(L, P) finds the portions of the candidate set L (here a line) that are in, on, and out of a reference set (here a polygon) P. (b) Image produced by ray casting, a special case of SMC.

may be used for ray casting visualization techniques to generate images of a known three-dimensional representation (Fig. 9.22b).

An algorithm for SMC illustrates a "divide and conquer" approach to computing on CSG. Recall that CSG is like a tree of set operations, whose leaves are primitive sets which usually are simple solids such as cylinders, spheres, and blocks. Presumably classification can be more easily computed with these simple sets as reference than with complex unions, intersections, and differences as reference.

The idea is that the classification of a set C with respect to a complex object S defined in CSG may be determined recursively. Any internal node S in the CSG tree is an operation node. It has left and right arguments and an operation OpofS. Each subtree is itself a CSG subtree or a primitive.

$$
\begin{aligned}
M(X, S) = \text{IF } S \text{ is a primitive THEN prim}-M(X, S) \\
\text{ELSE Combine}(M(X, \text{ left}-\text{subtree}(S)), \\
M(X, \text{ right}-\text{subtree}(S)), \\
\text{OPof}S);
\end{aligned}
$$

Prim-M is the easily computed classification with respect to a simple primitive solid. The Combine operation is a nontrivial calculation that combines the subresults to produce a more complex classification. It is illustrated in two dimensions for line classification in Fig. 9.23. Having classified the line L against the polygon $P1$ and $P2$, the classifications can be combined to produce the classification for $P1 \cap P2$. Precise rules for combine may be written for (regularized) union, intersection, and set difference. An important point is that when a point is in the "on" set of S_1 and in the "on" set of S_2, the result of the combination depends on extra information. In Fig. 9.23, segments X and Y both result from this ON–ON case of combine, but segment X is OUT of the boundary of the intersection and Y is IN the intersection. The ambiguity must be resolved by keeping "neighborhood information" (local geometry) attached to point sets, and combining the neighborhoods along with the classifications. The technical problems surrounding combine can be solved, and SMC is basic in several solid geometric modeling systems [Boyse 1979; Voelcker et al. 1978; Brown et al. 1978].

Mass Properties

The analog of many two-dimensional geometric properties is to be found in "mass properties," which are defined by volume integrals over a solid. The four types of mass properties commonly of interest are:

Volume: $V = \int_s du$

Centroid: e.g. $GC_x = \dfrac{\int_s x \, du}{V}$

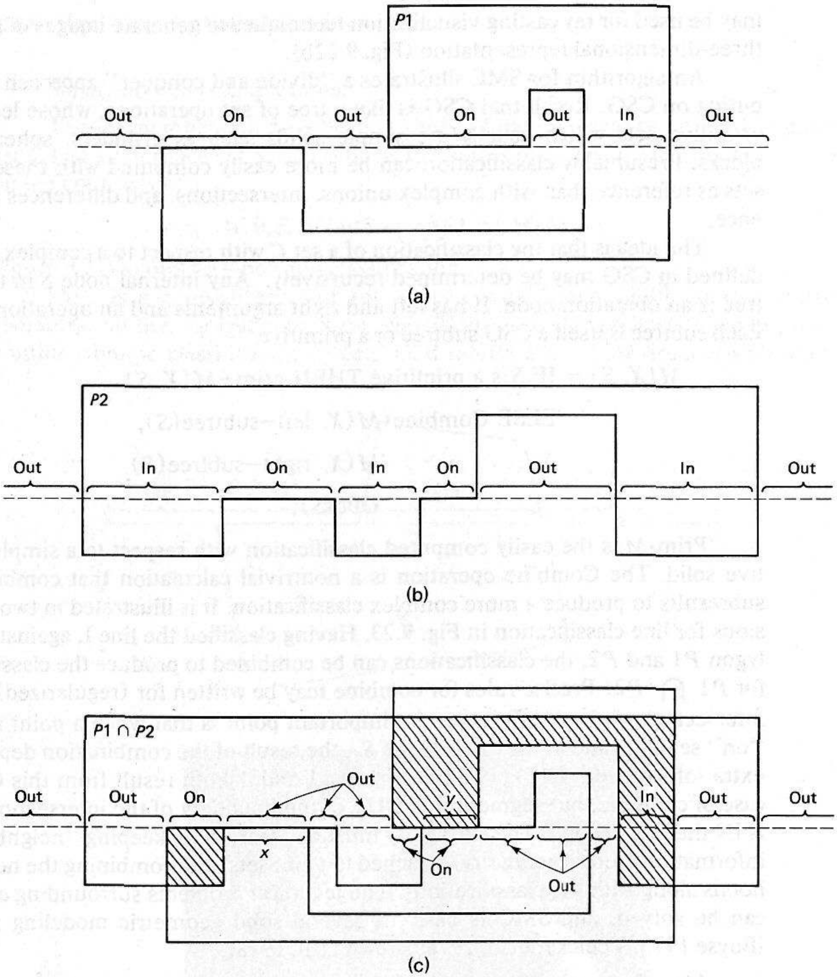


Fig. 9.23 Combining line-polygon classifications (a) and (b) must produce the classification (c).

$$\text{Moment of Inertia: e.g. } I_{xx} = m \int_s (y^2 + z^2)\, du \tag{9.17}$$

$$\text{Product of Inertia: e.g. } P_{xy} = m \int_s xy\, du$$

where m is a density measure, du the volume differential, and integrals are taken over the volume.

Measures such as these are not necessarily easy to compute from a given representation. The calculation of mass properties of solids from various representations is discussed in [Lee and Requicha 1980]. The approaches suggested by the representations are shown in Fig. 9.24.

One method is based on decomposing the solid into quasi-disjoint cells. An integral property of the cell decomposition is just the sum of the property for each of the cells. Hence if computing the property for the cells is easy, the calculation is easy for the whole volume. One is invited to decompose the body into simple cells, such as columns or cubes, as shown in Fig. 9.25. The resulting calculations, performed to reasonable error bounds on fairly complex volumes, take unacceptably long for the pure spatial occupancy enumeration, but are acceptable for the column and block decompositions. (The column decomposition corresponds to a ray casting approach.) The block decomposition method can be programmed using oct-trees or kd-trees in a manner reminiscent of the Warnock hidden-line algorithm [Warnock 1969], in which the blocks are found automatically, and their size diminishes as increased resolution is needed in the solid. In calculating from a constructive solid geometry representation, the same divide-and-conquer strategy that is useful for SMC may be applied. Again, it recursively solves subproblems induced by the set operators (Fig. 9.26). The strategy is less appealing here since the number of subproblems can grow exponentially in the worst case.

In boundary representations, one can perhaps directly integrate over the boundary in a three-dimensional version of the polygon area calculation given in Chapter 8. This method is often impossible for curved surfaces, which, however, may be approximated by planar faces. An alternative is to use the divergence

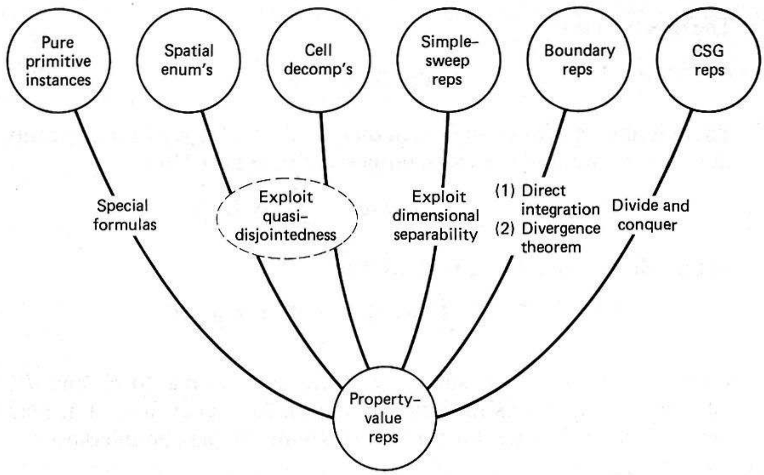


Fig. 9.24 "Natural" approaches to computing mass properties from several representations.

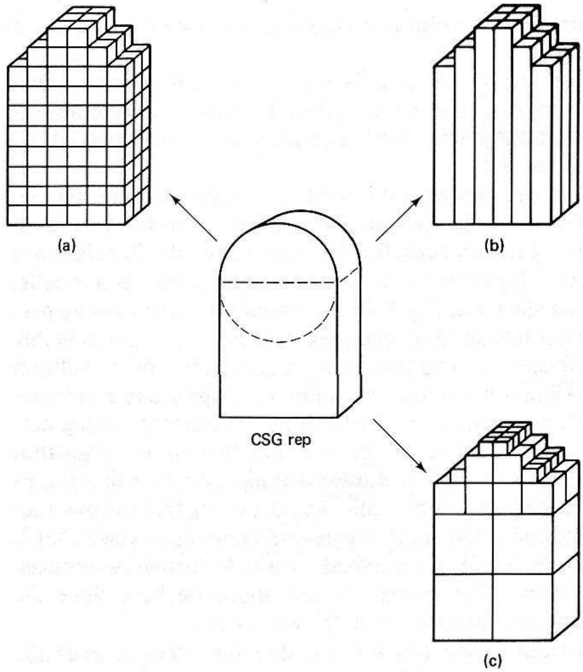


Fig. 9.25 Cell decompositions for mass properties.

theorem (Gauss's theorem). The *divergence* is a scalar quantity defined at any point in a vector field by writing the vector function as

$$G(x, y, z) = P(x, y, z)\mathbf{i} + Q(x, y, z)\mathbf{j} + R(x, y, z)\mathbf{k}. \tag{9.18}$$

The divergence is

$$\text{div } \mathbf{G} = \frac{P}{x} + \frac{Q}{y} + \frac{R}{z} \tag{9.19}$$

There is always a function **G** such that div **G** $= f(x, y, z)$ for any continuous function f (f computes the integral property of interest.) Thus

$$\int_s f \, dv = \int_s \text{div } \mathbf{G} \, dv \tag{9.20}$$

But the divergence theorem states that

$$\int_s \text{div } \mathbf{G} \, dv = \sum_i \int_{F_i} \mathbf{G}\mathbf{n}_i \, dF_i \tag{9.21}$$

where F_i is a face of the solid S, $\mathbf{n}_i$ is the unit normal to F_i, and dF_i the surface differential. Again this formula works well for planar faces, but may require approximation techniques for curved faces with complex boundaries.

Boundary Evaluation

The calculation of a face-based surface (boundary) representation from a

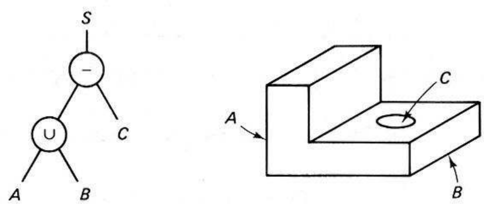
• Divide and conquer

Reduction formula

$$\int_{A \cup B} = \int_{A} + \int_{B} - \int_{A \cap B}$$

$$\int_{A - B} = \int_{A} - \int_{A \cap B}$$

Example



$$I_S = I_A + I_B - I_{A \cap B} - \underbrace{I_{A \cap C}}_{\emptyset} - I_{B \cap C} + \underbrace{I_{A \cap B \cap C}}_{\emptyset}$$

Fig. 9.26 Recursive problem decomposition for mass property calculation.

CSG representation is called *boundary evaluation.* It is an example of *representation conversion.* Both the CSG and boundary are usually unambiguous representations of a volume; a CSG expression (a solid) has just one boundary, but a boundary (representing a solid) usually has many CSG expressions. Since a solid may be put together from primitives in many ways, the mapping back from boundary to CSG is not usually attempted (but see [Markovsky and Wesley 1980, Wesley and Markovsky 1981]).

One style of boundary evaluation is based on the following observations [Voelcker and Requicha 1980; Boyse 1979].

- Boundaries of composite objects may be computed from certain set-theoretic formulae. For (regularized) intersection of two objects S and T, the formula is

$$b(S \cap^* T) = (bS \cap^* iT) \cup^* (iS \cap^* bT) \cup^* (bS \cap^* bT \cap^* ki(S \cap^* T)) \tag{9.22}$$

  where $\cap^*$ and $\cup^*$ are regularized intersection and union: b, i, and k are the boundary, interior, and closure operators. (Recall that ki is r, the regularization operator).
- Faces of composite objects can arise only from faces of primitives.
- Faces are either bounded by edges or are self-closing (as is the sphere).

These observations and the existence of the classification operation motivate the grand strategy that follows (ignoring several important details and concentrating on the core of the algorithm.)

1. Find all possible ("tentative") edges for each face of each primitive in the composite.
2. Classify each tentative edge with respect to the composite solid.
3. The ON portions of those edges must be enough to define the boundary.

Given the grand strategy, several algorithms of varying sophistication are possible, depending on what edges should be classified (how to generate tentative edges), in what order they should be classified, and how classification is done. The following algorithm is very simple (but very inefficient); useful algorithms are rather more complex.

Algorithm 9.1: CSG to Boundary Conversion (top-level control loop)

Input: Solid defined by CSG expression of regularized set operations applied to primitive solids.

Output: "Bfaces" in the object boundary. Bfaces are represented by their bounding edges. They may have little relation to the "intuitive faces" of the boundary; they may overlap each other, and a Bface may be disconnected (specify more than one region). Edges may appear many times. The Bface-oriented boundary may be processed to remove repetition and merge Bfaces into more intuitively appealing boundary faces.

BEGIN

Form a list PFaces of all ("intuitive") faces of primitive solids involved in the CSG expression, and an initially empty list BFaces to hold the output faces.

For every PFace $F1$ in PFaces:

Create a B-Face called ThisBFace, initially with no edges in it.

For every PFace $F2$ after $F1$ in the PFaces list (this generates all distinct pairs of PFaces just once):

Intersect $F1$ and $F2$ to get TEdges, a set of edges tentatively on the boundary of the solid. If $F1$ and $F2$ do not intersect or intersect only in a point, TEdges is empty. If they intersect in a line, TEdges is the single resulting edge. If they intersect in a two-dimensional region, TEdges contains the bounding edges of the intersection region.

Classify every TEdge in TEdges with respect to the whole solid (the CSG expression). Put TEdges that are ON the solid boundary into ThisBFace.

If ThisBFace is not empty, put it into BFaces.

End Inner Loop

End Outer Loop

END

Algorithms such as this involve many technical issues, such as merging coplanar faces, stitching edges together into faces, regularization of faces, removing multiple versions of edges. Boundary evaluation is inherently rather complex, and depends on such things as the definition and representation of faces as well as the geometric utilities taken as basic [Voelcker and Requicha 1981]. Boundary evaluation is an example of exact conversion between significantly different representations. Such conversions are useful, since no single representation seems convenient for all geometric calculations.

9.5 UNDERSTANDING LINE DRAWINGS

"Engineering" line drawings have been (and to a great extent are still) the main medium of communication between human beings about quantitative aspects of three-dimensional objects. The line drawings of this section are only those which are meant to represent a simple domain of polyhedral or simply curved objects. Interpretation of "naturalistic" drawings (such as a sketchmap [Mackworth 1977]) is another matter altogether.

Line drawings (even in a restricted domain) are often ambiguous; interpreting them sometimes takes knowledge of everyday physics, and can require training. Such informed interpretation means that even drawings that are strictly nonsense can be understood and interpreted as they were meant. Missing lines in drawings of polyhedra are often so easy to supply as to pass unnoticed, or be "automatically supplied" by our model-driven perception.

Generalizing the line drawing to three dimensions as a list of lines or points is not enough to make an unambiguous representation, as is shown by Fig. 9.27,

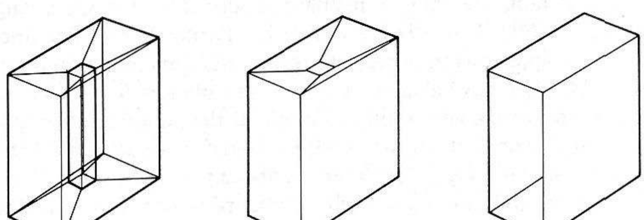


Fig. 9.27 An ambiguous (wireframe) representations of a solid with two of three possible interpretations.

which illustrates that a set of vertices or edges can define many different solids. (It is possible, however, to determine algorithmically all possible polyhedral boundaries described by a three-dimensional wireframe [Markowsky and Wesley 1980].). A line drawing nevertheless does convey three-dimensional information. For any set of N projection specifications (e.g., viewpoint and camera transform), a wire-frame object may be constructed that is ambiguous given the N projections. However, for a given object, there is a maximum number of projections that can determine the object unambiguously. The number depends on the number of edges in the object [Shapira 1974]. Reconstruction of all solids represented by projections is possible [Wesley and Markowsky 1981].

Line drawings were a natural early target for computer vision for the following reasons:

1. They are related closely to surface features of polyhedral scenes.
2. They may be represented exactly; the noise and incomplete visual processing that may have affected the "line drawing extraction" can be modelled at will or completely eliminated.
3. They present an interpretation problem that is significant but seems approachable.

The understanding of simple engineering (3-view) drawings was the first stage in a versatile robot assembly system [Ejiri et al. 1971]. This application underlined the fact that heuristics and conventions are indispensible in engineering drawing understanding. This section deals with the problem of "understanding" a single-view line drawing representation of scenes containing polyhedral and simple curved objects like those in Fig. 9.28.

Our exposition follows a historical path, to show how early heuristic programs in the middle 1960s evolved into more theoretical insights in the early 1970s.

The first real computer vision program with representations of a three-dimensional domain appeared around 1963 [Roberts 1965]. This system, ambitious even by today's standards, was to accept a digitized image of a polyhedral scene and produce a line drawing of the scene as it would appear when viewed from any requested viewpoint. This work addressed basic issues of imaging geometry, feature finding, object representation, matching, and computer graphics.

Since then, several systems have appeared for accomplishing either the same or similar results [Falk 1972; Shirai 1975; Turner 1974]. The line drawings of this section can appear as intermediate representations in a working polyhedral vision system, but they have also been studied in isolation. This topic took on a life of its own and provides a very pretty example of the general idea of going to the three-dimensional world of physics and geometry to understand the appearance of a two-dimensional image. The later results can be used to understand more clearly the successes and failures of early polyhedral vision systems. One form of understanding (line labelling) provided one of the first and most convincing demonstrations of parallel constraint propagation as a control structure for a computer vision process.

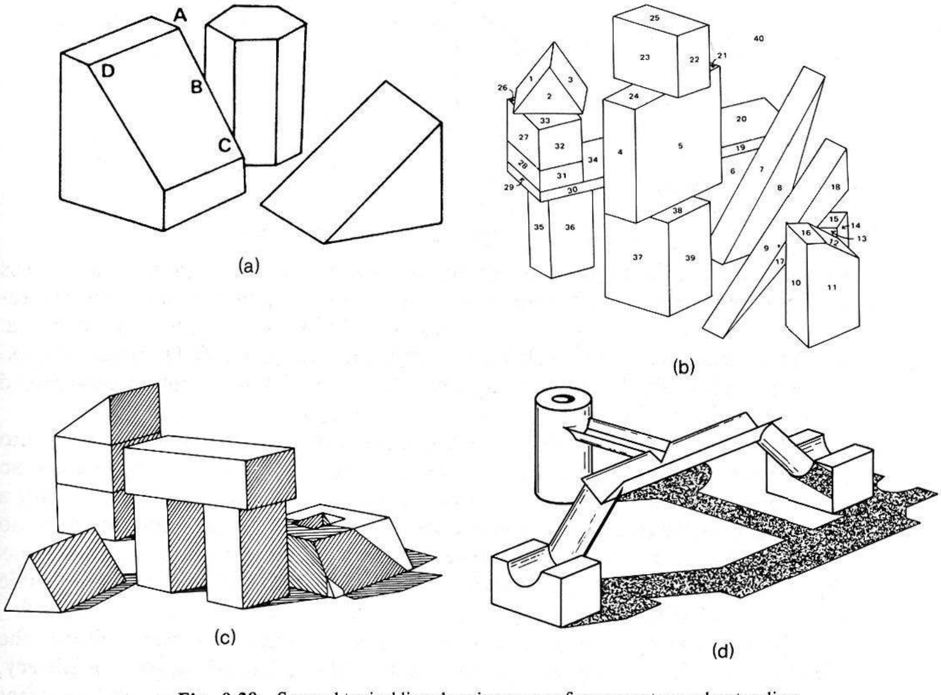


Fig. 9.28 Several typical line drawing scenes for computer understanding.

9.5.1 Matching Line Drawings to Three-dimensional Primitives

Roberts desires to interpret a line drawing such as Fig. 9.28a in terms of a small set of three polyhedral primitives, shown in Fig. 9.29. A simple polyhedron in a scene is regarded as an instance of a transformed primitive, where a transform may involve scaling along the three coordinate axes, translation, and rotation. Compound polyhedra, such as Fig. 9.28a, are regarded as simple polyhedra "glued together." (A cell-decomposition representation is thus used for compound polyhedra.) The program is first to derive from the scene the identity of the primitive objects used to construct it (including details of the construction of compound polyhedra). Next, it is to discover the transformations applied to the primitives to obtain the particular incarnations making up the scene. Finally, to demonstrate its understanding, it should be able to construct a line drawing of the scene from any viewpoint, using its derived description.

To understand a part of the scene, the program first decides which primitive it comes from, and then derives the transformation the primitive underwent to appear as it does in the scene. Identifying primitives is done by matching "topological" features of the line drawing (configurations of faces, lines, and vertices) with those of the model primitives; matching features induce a match between scene and model points. At least four noncoplanar matching points are needed to derive

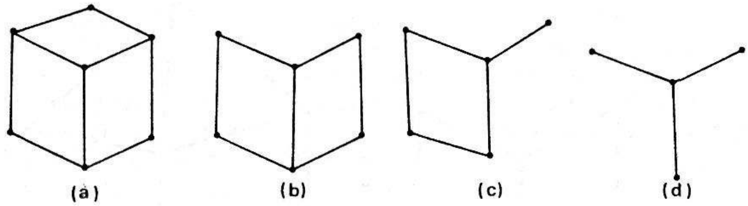


Fig. 9.31 Topological match structures of Roberts.

The idea once again is to accumulate local evidence from the scene, and then to group polygons on the basis of this evidence. The evidence takes the form of "links" which link two regions if they may belong to the same body; links are planted around vertices, which are classified into types, each type always planting the same links (Fig. 9.32). No links are made with the background region.

Scenes are interpreted by grouping according to regions/links, using fairly complex rules, including "inhibitory links" that preclude two neighboring regions from being in the same body.

The final form of the program performs reasonably well on scenes without accidents of visual alignment, but it is a maze of special cases and exceptions, and seems to shed little light on what is going on in known polyhedral line-drawing perception. One might well ask where the links come from; no justification of why they are correct is given. Further ([Mackworth 1973]), Guzman can accept as one body the two regions in Fig. 9.33a. Finally, one feels a little dissatisfied with a scheme that just answers "one body" to a scene like Fig. 9.33b, instead of answering "pyramid on cube" or "two wedges," for example.

Guzman's method is correct for a world of convex isolated trihedral polyhedra: it is extended by ad hoc adjustments based on various potentially conflicting items of evidence from the line drawing. Ultimately it performs adequately with a much increased range of scenes, albeit not very elegantly. Further progress in the line drawing domain came about when attention was directed at the three-dimensional causes of the different vertex types.

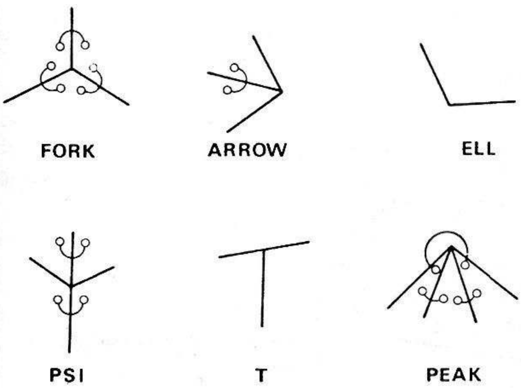


Fig. 9.32 Links around vertices.

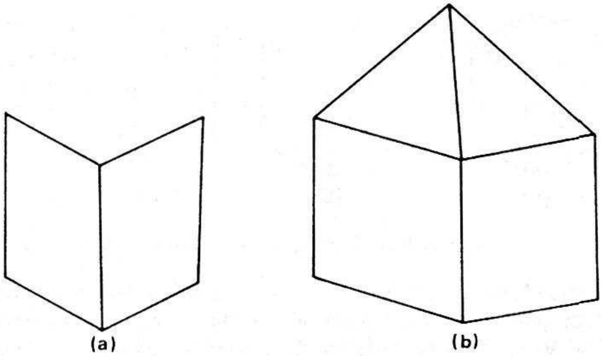


Fig. 9.33 (a) Non-polyhedral scene. (b) Two wedges or a pyramid on cube.

9.5.3 Labeling Lines

Huffman and Clowes independently concerned themselves with scenes similar to Guzman's, not excluding non-simply connected polyhedra, but excluding accidents of alignment [Huffman 1971; Clowes 1971]. They desired to say more about the scene than just which regions arose from single bodies; they wanted to ascribe interpretations to the lines. Figure 9.34 shows a cube resting on the floor; lines labeled with a + are caused by a convex edge, those labeled with a − are caused by a concave edge, and those labeled with a > are caused by matter occluding a surface behind it. The occluding matter is to the right of the line looking in the direction of the >, the occluded surface is to the left. If the cube were floating, one would label the lowest lines with < instead of with −. The shadow line labels (arrows) were not used by Huffman.

A systematic investigation can find the types of lines possibly seen around a trihedral corner; such corners can be classified by how many octants of space are filled by matter around them (one for the corner of a cube, seven for the inside corner of a room, etc.). By considering all possible trihedral corners as seen from

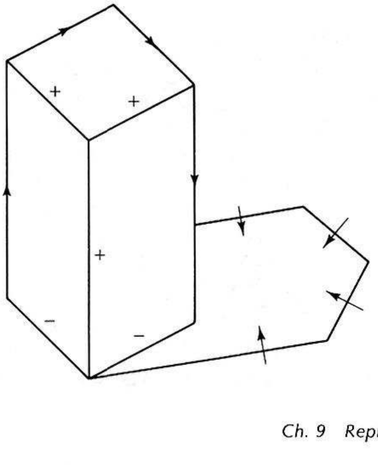


Fig. 9.34 A block resting on its bottom surface.

all possible viewpoints, Huffman and Clowes found that without occlusion, just four vertex types and only a few of the possible labelings of lines meeting at a vertex can occur. Figure 9.35 shows views of one- and three-octant corners which give rise to all possible vertices for these corner types. The vertices appear in the first two rows of Table 9.1, which is a catalogue of all possible vertices, including those arising from occlusion, in this restricted world of trihedral polyhedra. It is easy to imagine extending the catalog to include vertices for other corner types.

It is important to note that there are four possible labels for each line ($+ - > <$), and thus $4^3 = 64$ possible labels for the fork, arrow, and T and 16 possible labels for the ell. In the catalog, however, only 3/64, 3/64, 4/64, and 6/16, respectively, of the possible labels actually occur. Thus only a small fraction of possible labels can occur in a scene.

The main observation that lets line-labeling analysis work is the coherence rule: In a real polyhedral scene, *no line may change its interpretation (label) between vertices.* For example, what is wrong with scenes like Fig. 9.36 is that they cannot be coherently labeled; lines change their interpretation within the impossible object. Perhaps the lines in drawings of real scenes can be interpreted quickly because the small percentage of meaningful labelings interacts with the coherence rule to reduce drastically the number of explanations for the scene.

How does line labeling relate to Guzman? A labeled-line description clearly indicates the grouping of regions into bodies, and also rejects scenes like Fig. 9.33a, which cannot be coherently labeled with labels from the catalog. The origin of Guzman's links can be explained this way: consider again the world of convex polyhedra; the only labels from the catalog that are possible are shown in Fig. 9.37a. Further, it is clear that a convex edge has two faces of the same body on either side of it, and an occluding edge has faces from two different bodies on either side of it. A convex label means the regions on either side of it should be linked; this is Guzman's link-planting rule (Fig. 9.37b). The inhibition rules are a further corollary of the labels; they are to suppress links across an edge if evidence that it

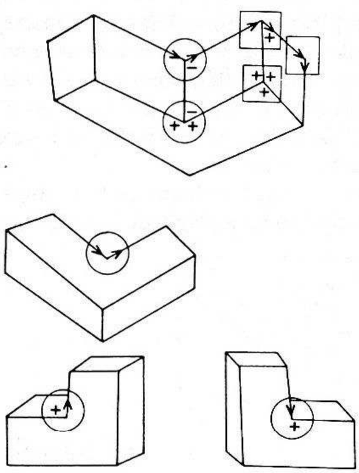
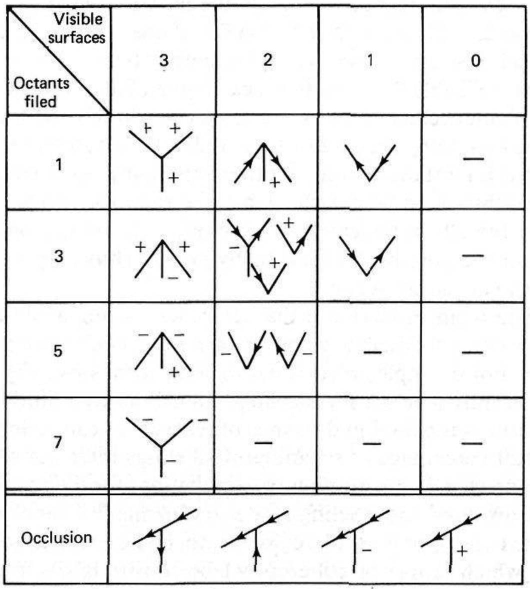


Fig. 9.35 Different views of various corner types.

Table 9.1
VERTEX CATALOGUE



must be occluding is supplied by the vertex at its other end (Fig. 9.37c). When vertices at both ends of a line agree that the line is convex, Guzman would have planted two links; this is in fact the strongest evidence that the regions are part of the same body. If just one vertex gives evidence that the edge has a link, a decision based on heuristics is made; the coherence rule is being used implicitly by Guzman. The same physical and geometric reality is driving both his scheme and that of Huffman.

The labeling scheme explained here still has problems: syntactically nonsensical scenes are coherently labeled (Fig. 9.38a); scenes are given geometrically impossible labels (Fig. 9.38b); and scenes that cannot arise from polyhedra are easily labelled (Fig. 9.38c). It is very hard to see how a labeling scheme can detect the illegality of scenes like (Fig. 9.38c); the problem is not that the edges are incorrectly labeled, but that the faces cannot be planar.

Concern with this last-mentioned problem led to a program (see the next section) that can obtain information about a polyhedral scene equivalent to labeling it,

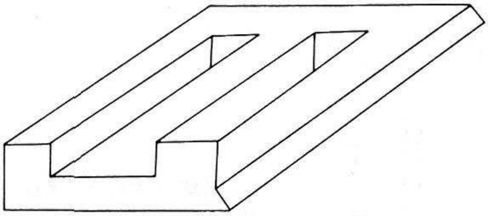


Fig. 9.36 An impossible object.

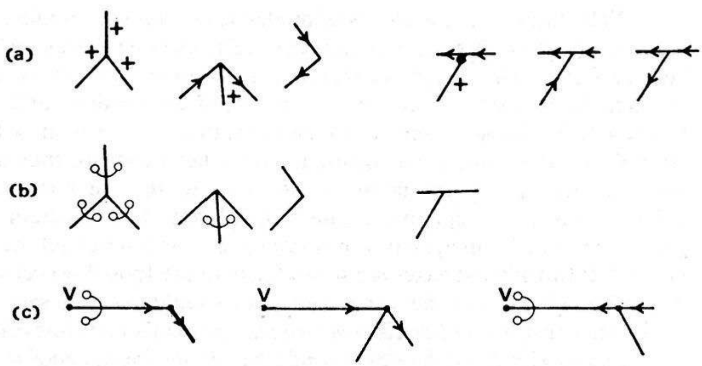


Fig. 9.37 The relation of links to labels. (a) Line labels. (b) Link planting vertices. (c) Inhibitory links.

and also can reject non-polyhedra as impossible. There has also been an exciting denoument to the line-labeling idea [Waltz 1975; Turner 1974].

Waltz extends the line labels to include shadows, three illumination codes for each face on the side of an edge, and the separability of bodies in the scene at cracks and concave edges; this brings the number of line labels possible up to just below 100. He also extends the possible vertex types, so that many vertices of four lines occur. He can deal with scenes such as the one shown in Fig. 9.28c.

The combinatorial consequence of these extensions is clear; the possible vertex labelings multiply enormously. The first interesting thing Waltz discovered was that despite the combinatorics, as more information is coded into the lines, the smaller becomes the percentage of geometrically meaningful labels for a vertex. In his final version, only approximately 0.03 percent of the possible arrow labels can occur, and for some vertices the percentage is approximately 0.000001.

The second interesting thing Waltz did was to use a constraint-propagating labeling algorithm which very quickly eliminates labels for a vertex that is impossible given the neighboring vertices and the coherence rule, which places *constraints* on labelings. The small number of meaningful labels for a vertex imposes severe constraints on the labeling of neighboring vertices. By the coherence rule, the constraints may be passed around the scene from each vertex to its neighbors; eliminating a label for a vertex may render neighboring labels illegal as well, and so on recursively.

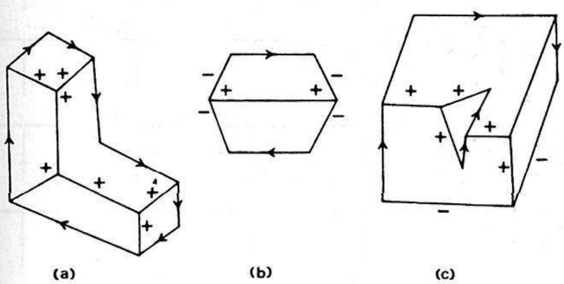


Fig. 9.38 Nonsense labelings and nonpolyhedra.

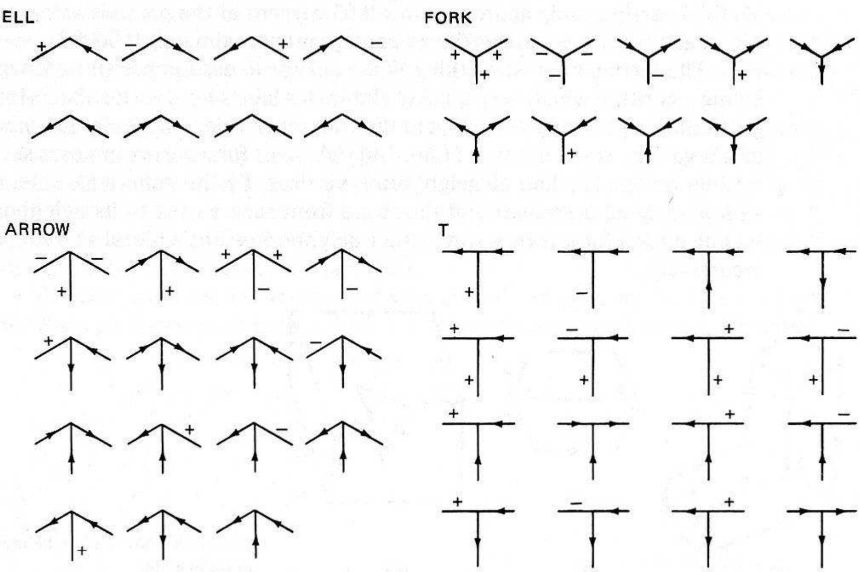
Waltz found that for scenes of moderate complexity, eliminating all impossible labelings left only one, the correct one. The labeling process, which might have been expected to involve much search, usually involved none. This constraint propagation is an example of parallel constraint satisfaction, and is discussed in Chapter 12 in a broader context. In the event that a vertex is left with several labels after all junction coherence constraints have been applied, they all participate in *some* legal labeling. At this point one can resort to tree search to find the explicit labelings, or one can apply more constraints. Many such constraints, heuristic and geometric, may be imagined. For instance, a constraint could involve color edge profiles. If two aligned edges are separated by some (possibly occluding) structure, but still divide faces of the same color, they should have the same label. Another important constraint concerns how face planarity constrains line orientations.

Scenes with missing lines may be labeled; one merely adds to the legal vertex catalog the vertices that result if lines are missing from legal vertices. This idea has the drawbacks of increasing the vertex catalog and widening the notion of consistency, but can be useful.

Another extension to line labeling is that of [Kanade 1978]. This extension considers not only solid polyhedra but objects (including nonclosed "shells") made up of planar faces. This extension has been called *origami world* after the art of making objects from folded (mostly planar) paper. An example from origami world is the box in Fig. 9.39a. A quick check shows that this cannot be labeled with the Huffman-Clowes label set. It can be labeled using the origami world label set (Table 9.2) and its interpretation is shown in Fig. 9.39b.

Table 9.2

EXPANDED JUNCTION TABLE



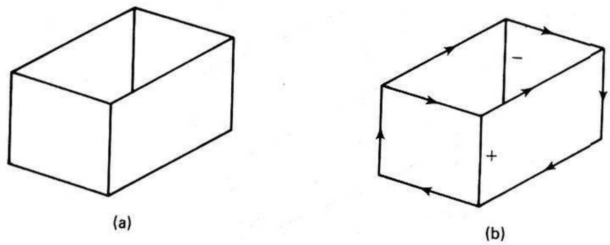


Fig. 9.39 (a) Box. (b) Labeled edges according to origami world label set.

The vertex labels may be extended to include scenes with cylinders, cones, spheres, tori, and other simple curves. In expanded domains the notion of "legal line drawing" becomes very imprecise. In any event the number of vertex types and labels grow explosively, and the coherence rule must be modified to cope with the fact that lines can change their interpretation between vertices and can tail off into nothing, and that one region can attain all three of Waltz's illumination types [Turner 1974, Chakravarty 1979]. The domain is of scenes such as appear in Fig. 9.28d.

9.5.4 Reasoning About Planes

The deficiencies in the scene line-labeling algorithms prompted a consideration of the geometrical foundations of the junction labels [Mackworth 1973, Sugihara 1981]. This work seeks to answer the same sorts of questions as do labeling programs, but also to take account of objects that cannot possibly be planar polyhedra, such as those of Fig. 9.40. Neither approach uses a catalog of junction labels, but relies instead on ideas of geometric coherence. The basis is a plane-oriented formulation rather than a line-oriented one.

Gradient Space

Mackworth's program relies heavily on the relation of polyhedral surface gradients to the lines in the image (recall section 3.5.2). Image information from orthographic projections of planar polyhedral scenes may be related to gradient information in a useful way. An image line L is the projection of a three-space line M arising from the intersection of two faces lying in distinct planes Π_1 and Π_2 of gradients (p_1, q_1) and (p_2, q_2). With the (p, q) coordinate system superimposed on the image (x, y) coordinate system, there is the following constraint. The orientation of L constrains the gradients of Π_1 and Π_2; specifically, the line L is perpendicular to the line G between (p_1, q_1) and (p_2, q_2) (Fig. 9.41).

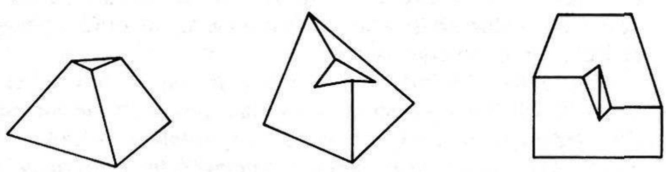


Fig. 9.40 Labelable but not planar polyhedra.

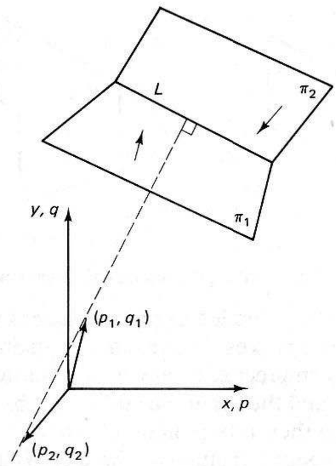


Fig. 9.41 Gradient space constraint.

The result is easily shown. With orthographic projection, the origin may be moved of the image plane to be in L without loss of generality. Then L is defined by its direction vector $(\lambda, \mu) = (\cos\theta, \sin\theta)$. The three-space point on Π_1 corresponding to $(0,0)$ may be expressed as $(0, 0, k_1)$, and at (λ, μ) the corresponding point is $(\lambda, \mu, \lambda p_1 + \mu q_1 + k_1)$. Thus moving along M (which is in Π_1) from $(x, y) = (0, 0)$ to $(x, y) - (\lambda, \mu)$ moves along $-z$ by $\lambda p_1 + \mu q_1$. The coordinates of a unit vector on L can then be expressed as $(\lambda, \mu, \lambda p_1 + \mu q_1)$. But L is also in Π_2, and this argument may be repeated for Π_2, using p_2 and q_2. Thus

$$\lambda p_1 + \mu q_1 = \lambda p_2 + \mu q_2 \tag{9.23}$$

or

$$(\lambda, \mu) \cdot (p_2 - p_1,\ q_2 - q_1) = 0 \tag{9.24}$$

Equation (9.24) is a dot product set equal to zero, showing that its two vector operands are orthogonal, which was to be shown.

Every picture line results from the intersection of two planes, and so it has a line associated with it in gradient space which is perpendicular to it. Furthermore, if the gradients of the surfaces are on the same side of the picture line as their surfaces, the edge was convex; if the gradients are on opposite sides of the line from their causing surfaces, the edge was concave (Fig. 9.42). For every junction in the image there are just two ways the gradients can be arranged to satisfy the perpendicularity requirement (Fig. 9.43). In the first, all edges are convex, in the second, concave. Switching interpretations from one to the other by negating gradients is the psychological "Necker reversal."

Notice that if an image junction is a three-space polyhedral vertex, each edge of the vertex is the intersection of two face planes. If the corresponding gradients are connected, a "dual" (p, q) space representation of the (x, y) space junction is formed. The connected (p, q) gradient points form a polygon whose edges are perpendicular to the junction lines in (x, y) space. The polygon is larger if the three-

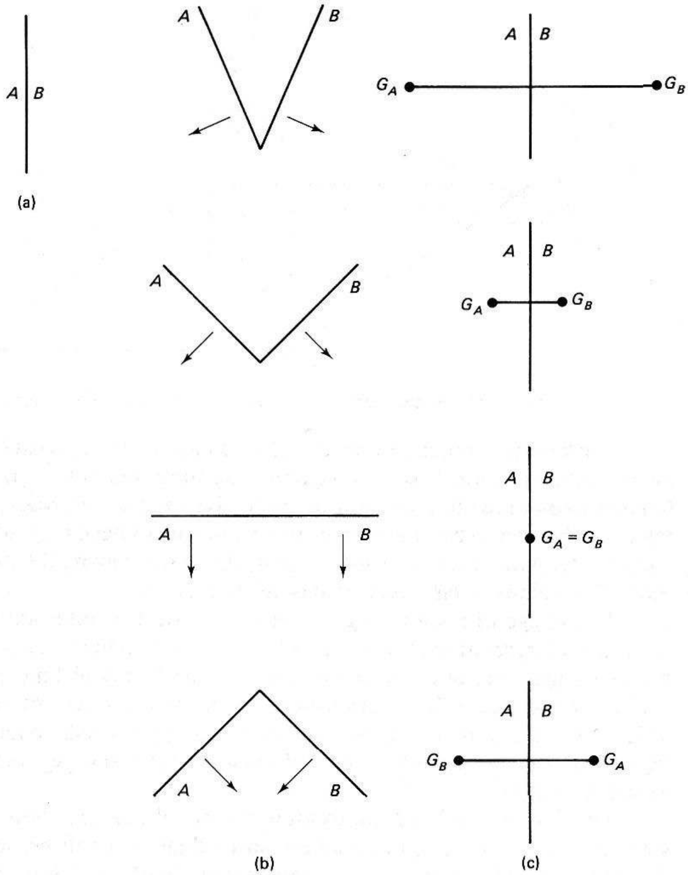


Fig. 9.42 Relation of gradients, image and world structures. (a) Image. (b) World. (c) Gradients.

dimensional corner is sharper, and shrinks toward the junction point as the corner gets blunter.

Interpreting Drawings

It is possible to use these geometric results to interpret the lines in orthogonally projected polyhedral scenes as being "connect" (i.e., as being between two connected faces) or occluding. It can also be determined if connect edges are convex or concave, and for occluding edges which surface is in front. Hidden parts of the scene may sometimes be reconstructed. The orientation of each surface and edge in the scene may be found. Thus a program can determine that input such as Fig. 9.40 is not a planar-faced polyhedron [Mackworth 1973]. Sugihara's work generalizes Mackworth's; it does not use gradient space and does not rely on orthographic projection.

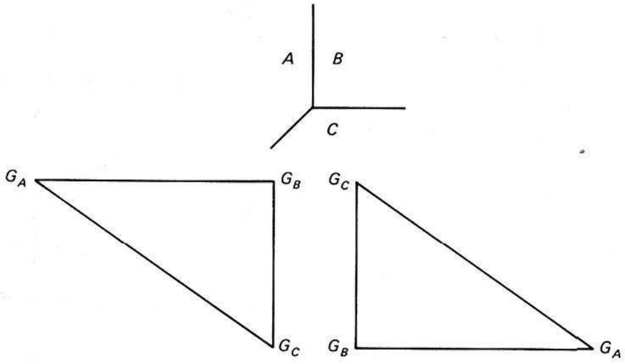


Fig. 9.43 A scene junction and two resulting triangles in gradient space.

Mackworth's procedure to establish connect edges produces the most connected interpretation first (a nonconnected interpretation is just a collection of floating faces which line up by accident to give the line drawing). The background region is the first to be interpreted; that is, means to have its gradient fixed in gradient space. After a region is interpreted, the region having the most lines in common with regions so far interpreted is interpreted next.

The image of a scene is given in Fig. 9.44a; it is interpreted as follows. No coherent interpretation is possible with five or four connect edges. Trying for three connect edges, the program interprets A by arbitrarily picking a gradient for the surface A represents (the background). It picks the origin of gradient space. In order to be able to reason about lines in the image, it needs to have an interpreted region on either side of the line, so it must interpret another region. It picks B (C would be as good).

The lines bounding B are examined to see if they are connect. Line 1 is considered. If it is connect, the gradient space dual of it will be perpendicular to it through the gradient space point representing surface A (i.e., the origin). Now another arbitrary choice: The gradient corresponding to surface B is placed at unit distance from the origin, thus "imagining" the second gradient in a row. From now on, the gradients are more strongly located. The arbitrary scaling and point of origin imposed by these first two choices can be changed later if that is important.

In gradient space, the situation is now shown in Fig. 9.44b. Now consider line 2; to establish it as a connect edge, $\boldsymbol{G}_B = (p_B, 1_B)$ (the gradient space point corresponding to the surface B) must lie on a line perpendicular to 2 through $\boldsymbol{G}_A$ (Fig. 9.44c). This cannot happen; the situation with 1 and 2 both connect is incoherent. Thus, with a line 1 connect edge, 2 must be occluding. This sort of incoherency result was what kept the program from finding four or five edges connect. Further interpretation involves assigning gradients and vertices into the developing diagram in a noncontradictory, maximally connected manner (Fig. 9.44d).

The next part of the program determines convexity or concavity of the lines. The final part of the program looks at occlusion. It also suggests hidden surfaces

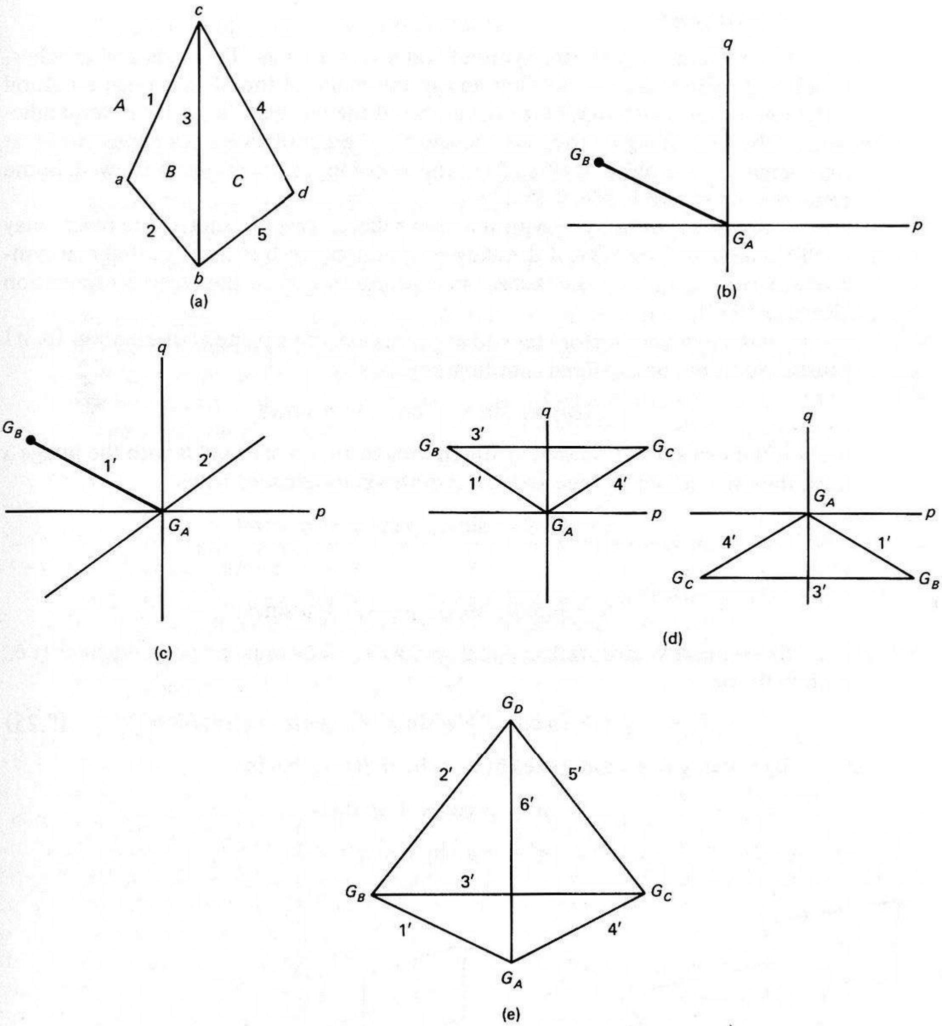


Fig. 9.44 (a) Polyhedral scene considered by Mackworth. (b) Partial interpretation. (c) Continued interpretation. (d) Occluding and connect interpretations. (e) Final interpretation.

and thus hidden lines that are consistent with the interpretation (Fig. 9.44e). This figure in gradient space resembles a tetrahedron, as well it might; it is formed in the same way as the graph-theoretic dual (point per face, edge per edge, face per point) which defines dual graphs and dual polyhedra; the tetrahedron is self-dual. The arbitrary choices of gradient reflect degrees of freedom in the drawing that are also identified by Sugihara.

Skewed Symmetry

Many planar objects are symmetrical about an axis. This axis and another, which is perpendicular to the first and in the plane of the object, form a natural orthogonal coordinate system for the object. If the plane of the object is perpendicular to the line of sight from the viewpoint, the coordinate axes appear to be at right angles. If the object is tilted from this position, the axes appear skewed. Some examples are shown in Fig. 9.45.

A skewed symmetry may or may not reflect a real symmetry; the object may itself be skewed. However, if the skewed symmetry results from a tilted real symmetry, a constraint in gradient space may be developed for the object's orientation [Kanade 1979].

An imaged unit vector inclined at α inscribed on a plane at orientation (p, q) must have three-dimensional coordinates given by

$$(\cos\alpha,\ \sin\alpha,\ p\cos\alpha + q\sin\alpha)$$

Thus if the two axes of skewed symmetry make angles of α and β with the image x axis, the two vectors in three-space a and b must have coordinates

$$\mathbf{a} = (\cos\alpha,\ \sin\alpha,\ p\cos\alpha + q\sin\alpha)$$

and

$$\mathbf{b} = (\cos\beta,\ \sin\beta,\ p\cos\beta + q\sin\beta)$$

Since these vectors reflect a real symmetry, they must be perpendicular (i.e., $\mathbf{a}\cdot\mathbf{b} = 0$), or

$$\cos(\alpha - \beta) + (p\cos\alpha + q\sin\alpha)(p\cos\beta + q\sin\beta) = 0 \qquad (9.25)$$

By rotating the p and q axes by $\lambda = (\alpha + \beta)/2$, that is

$$p' = p\cos\lambda + q\sin\lambda$$

$$q' = -p\sin\lambda + q\cos\lambda$$

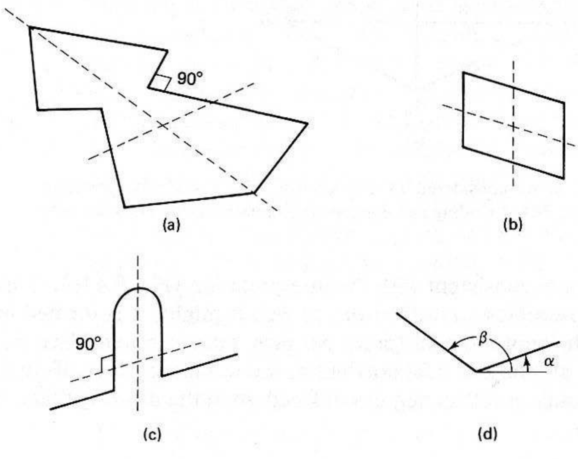


Fig. 9.45 Skewed symmetries. (a,b,c) are examples. (d) Each skewed symmetry defines two axes.

Equation (9.25) can be put into the form

$$p'^2 \cos^2\left(\frac{\gamma}{2}\right) - q'^2 \sin^2\left(\frac{\gamma}{2}\right) = -\cos\,(\gamma)$$

where $\gamma = \alpha - \beta$. Thus the gradient of the object must lie on a hyperbola with axis tilted λ from the x axis, and with asymptotes perpendicular to the directions of α and β. This constraint is shown in Fig. 9.46.

To show how skewed symmetry can be exploited to interpret objects with planar faces, reconsider the example of Fig. 9.43. In that example the three convex edges constrained the gradients of the corresponding faces to be at the vertices of a triangle, but the size or position of the triangle in gradient space was unknown. However, skewed symmetry applied to each face introduces three hyperbola upon which the gradients must lie. The only way that both the skewed symmetry constraint and triangle constraint can be satisfied simultaneously is shown in Fig. 9.47—the combined constraints have uniquely determined the face orientations.

EXERCISES

9.1 Derive an expression for the volume of an object represented by spherical harmonics of order $M = 1$.

9.2 Derive an expression for the perpendicular to the surface of an object represented by spherical harmonics in terms of the appropriate derivatives.

9.3 Derive an expression for the angle centroid of each of the spherical harmonic functions for $M \leqslant 2$.

9.4 Label the lines in the objects of Fig. 9.48.

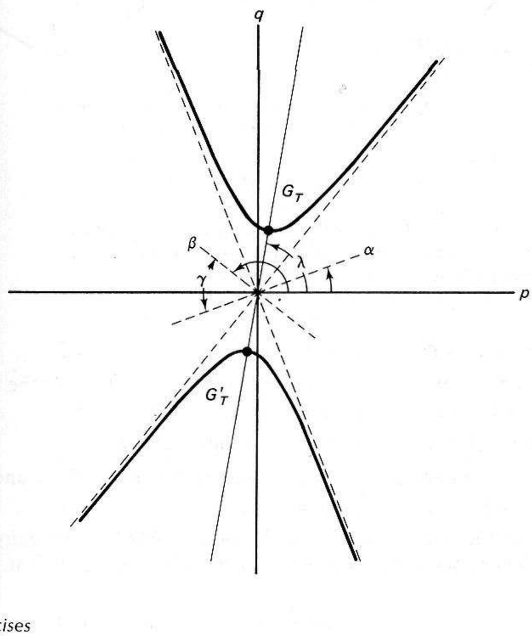


Fig. 9.46 Skewed symmetry constraint in gradient space.

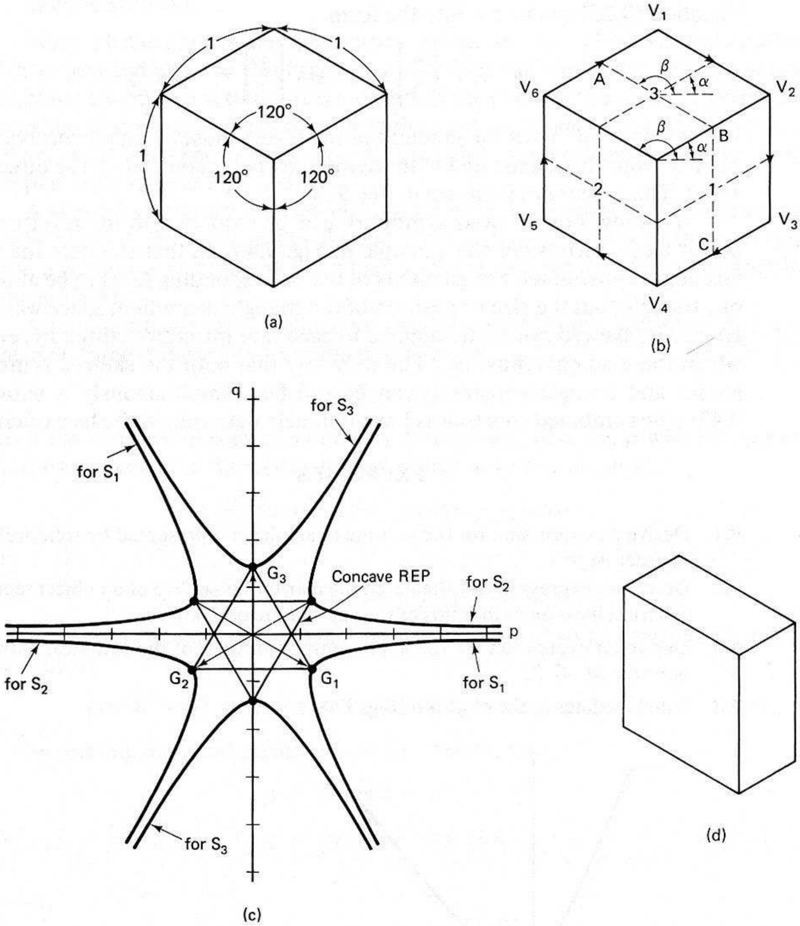


Fig. 9.47 Using skewed symmetry to orient the faces of a cube. (a) The cube. (b) Skewed symmetries. (c) skewed symmetries and junction constraint plotted in gradient space. (d) another possible object obeying the constraints.

9.5 Give two sets of CSG primitives with same domain.

9.6 Show that the dual of the plane of interpretation for a line and the duals of the two planes that meet in the edge causing the line are all on the dual of the edge.

9.7 Prove (Section 9.3.1) that in the Frenet frame $\boldsymbol{\xi}'$ is perpendicular to $\boldsymbol{\xi}$.

9.8 Write the precise rules for combining classification results for $\bigcup^*$, $\bigcap^*$, and $-$ operations.

9.9 Find two interpretations of the tetrahedron of Fig. 9.44a that differ in convexity or concavity of lines. (Hint: The concave interpretation has an accident of alignment.)

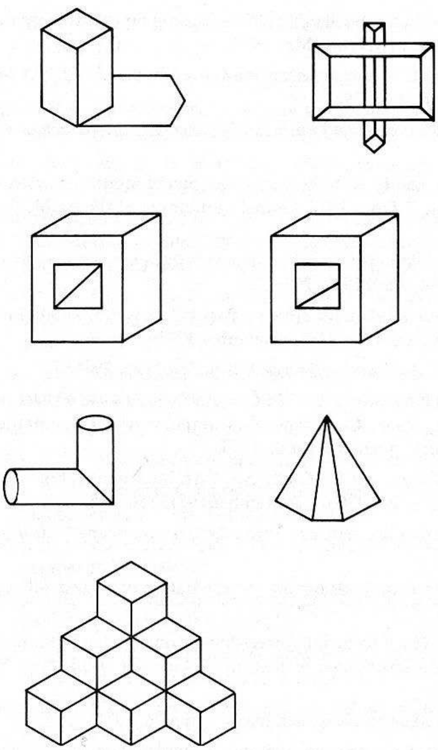


Fig. 9.48 Objects for labeling.

REFERENCES

AGIN, G. J. "Representation and description of curved objects" (Ph.D. dissertation). AIM-173, Stanford AI Lab, October 1972.

BADLER, N. I. and R. K. BAJCSY. "Three-dimensional representations for computer graphics and computer vision." *Computer Graphics 12*, August 1978, 153–160.

BADLER, N. I. and J. O'ROURKE. "Representation of articulable, quasi-rigid, three-dimensional objects." NSF Workshop on the Representation of Three-Dimensional Objects, Univ. Pennsylvania, May 1979.

BARNHILL, R. E. "Representation and approximation of surfaces." In *Mathematical Software III*, J. R. Rice (Ed.). New York: Academic Press, 1977.

BARNHILL, R. E. and R. F. RIESENFELD. *Computer Aided Geometric Design.* New York: Academic Press, 1974.

BAUMGART, B. G. "Winged edge polyhedron representation." STAN-CS-320, AIM-179, Stanford AI Lab, October 1972.

BENTLEY, J. L. Multidimensional search trees used for associative searching, *Comm. ACM 18*, 9, Sept. 1975, 509–517.

BINFORD, T. O. "Visual perception by computer." IEEE Conf. on Systems and Control, Miami, December 1971.

BOYSE, J. W. "Data structure for a solid modeller," NSF Workshop on the Representation of Three-Dimensional Objects, Univ. Pennsylvania, May 1979.

BROWN, C. M. "Two descriptions and a two-sample test for 3-d vector data." TR49, Computer Science Dept., Univ. Rochester, February 1979a.

BROWN, C. M. "Fast display of well-tesselated surfaces." *Computers and Graphics 4*, 2, September 1979b, 77–85.

BROWN C. M., A. A. G. REQUICHA, and H. B. VOELCKER. "Geometric modelling systems for mechanical design and manufacturing." *Proc.*, 1978 Annual Conference of the ACM, Washington, DC, December 1978, 770–778.

CHAKRAVARTY, I. "A generalized line and junction labelling scheme with applications to scene analysis," IEEE Trans. PAMI, April 1979, 202–205.

CLINTON, J. D. "Advanced structural geometry studies, Part I: Polyhedral subdivision concepts for structural applications." NASA CR-1734/35, September 1971.

CLOWES, M. B. "On seeing things." *Artificial Intelligence 2*, 1, Spring 1971, 79–116.

COONS, S. A. "Surface patches and B-spline curves." In *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld (Eds.). (*Proc.*, Conference on Computer Aided Geometric Design, Univ. Utah, March 1974.) New York: Academic Press, 1974.

EJIRI, M., T. UNO, H. YODA, T. GOTO, and K. TAKEYASU. "An intelligent robot with cognition and decision-making ability." *Proc.*, 2nd IJCAI, September 1971, 350–358.

FALK, G. "Interpretation of important line data as a three-dimensional scene." *Artificial Intelligence 3*, 1, Spring 1972, 77–100.

FORREST, A. R. "On cones and other methods for the representation of curved surfaces." *CGIP 1*, 4, December 1972, 341–359.

GUZMAN, A. "Decomposition of a visual scene into three-dimensional bodies" (Ph.D. dissertation). In *Automatic Interpretation and Classification of Images*, A. Grasseli (Ed.). New York: Academic Press, 1969.

HUFFMAN, D. A. "Impossible objects as nonsense sentences." In *MI6*, 1971.

JACKINS, C. L., and S. L. TANIMOTO. Oct-trees and their use in representing three-dimensional objects, *CGIP 14*, 3, Nov. 1980, 249–270.

KANADE, T. "A theory of Origami world." CMU-CS-78-144, Computer Science Dept., Carnegie-Mellon Univ., 1978.

KANADE, T. "Recovery of the three-dimensional shape of an object from a single view." CMU-CS-79-153, Computer Science Dept., Carnegie-Mellon Univ., October 1979.

LAKATOS, I. *Proofs and Refutations.* Cambridge, MA: Cambridge University Press, 1976.

LEE, Y. T. and A. A. G. REQUICHA. "Algorithms for computing the volume and other integral properties of solid objects." Tech. Memo 35, Production Automation Project, Univ. Rochester, Rochester NY, Feb. 1980.

MACKWORTH, A. K. "Interpreting pictures of polyhedral scenes." *Artificial Intelligence 4*, 2, June 1973, 121–137.

MACKWORTH, A. K. "On reading sketch maps." *Proc.*, 5th IJCAI, August 1977, 598–606.

MARKOWSKY, G. and M. A. WESLEY. "Fleshing out wire frames." *IBM J. Res. Devel. 24*, 1 (Jan. 1980) 64–74.

MARR, D. and H. K. NISHIHARA. "Representation and recognition of the spatial organization of three-dimensional shapes." *Proc., Royal Society of London B 200*, 1978, 269–294.

NISHIHARA, H. K. "Intensity, visible surface and volumetric representations." NSF Workshop on the Representation of Three-Dimensional Objects, U. Pennsylvania, May 1979.

O'NEILL, B. *Elementary Differential Geometry.* New York: Academic Press, 1966.

O'ROURKE, J. and N. I. BADLER. "Decomposition of three-dimensional objects into spheres." *IEEE Trans. PAMI 1*, July 1979.

REQUICHA, A. A. G. "Representations of rigid solid objects." *Computer Surveys 12*, 4, December 1980.

ROBERTS, L. G. "Machine perception of three-dimensional solids." In *Optical and Electro-optical Information Processing*, J.P. Tippett et al. (Eds.). Cambridge, MA: MIT Press, 1965.

SCHUDY, R. B. and D. H. BALLARD. "Model-detection of cardiac chambers in ultrasound images." TR12, Computer Science Dept., Univ. Rochester, November 1978.

SCHUDY, R. B. and D. H. BALLARD. "Towards an anatomical model of heart motion as seen in 4-d cardiac ultrasound data." *Proc.*, 6th Conf. on Computer Applications in Radiology and Computer-Aided Analysis of Radiological Images, June 1979.

SHANI, U. "A 3-d model-driven system for the recognition of abdominal anatomy from CT scans." TR77, Computer Science Dept., U. Rochester, May 1980; also in Proc. 5th IJCPR, Miami, December 1980, 585–591.

SHAPIRA, R. "A technique for the reconstruction of a straight-edge, wire-frame object from two or more central projections." *CGIP 3*, 4, December 1974, 318–326.

SHIRAI, Y. "Analyzing intensity arrays using knowledge about scenes." In *PCV*, 1975.

SOROKA, B. I. "Generalised cylinders from parallel slices." *Proc.*, PRIP, 1979a, 421–426.

SOROKA, B. I. "Understanding objects from slices." Ph.D. dissertation, Dept. of Computer and Information Science, Univ. Pennsylvania, 1979b.

SOROKA, B. I. and R. K. BAJCSY. "Generalized cylinders from serial sections." *Proc.*, 3rd IJCPR, November 1976, 734–735.

SUGIHARA, K. "Mathematical structures of line drawings of polyhedra," RNS 81-02, Dept. of Info. Science, Nagoya Univ., May 1981.

TILOVE, R. B. "Set membership classification: a unified approach to geometric intersection problems." *IEEE Trans. Computers 29*, 10, October 1980.

TURNER, K. J. "Computer perception of curved objects using a television camera." Ph.D. dissertation, Univ. Edinburgh, 1974.

VOELCKER, H. B. and A. A. G. REQUICHA, Boundary evaluation procedures for objects defined via constructive solid geometry, Tech. Memo 26, Production Automation Project, Univ. Rochester, 1981.

VOELCKER, H. B. and A. A. G. REQUICHA. "Geometric modeling of mechanical parts and processes." *Computer 10*, December 1977, 48–57.

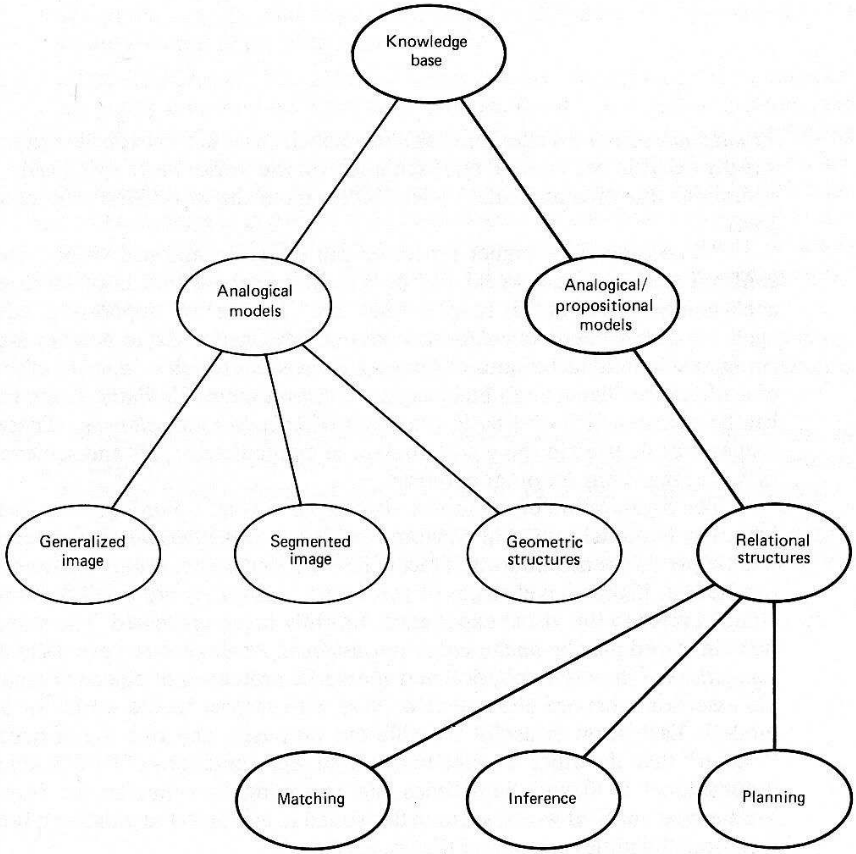
VOELCKER, H. B. and Staff of Production Automation Project, "The PADL-1.0/2 system for defining and displaying solid objects." *Computer Graphics 12*, 3, August 1978, 257–263.

WALTZ, D. I. "Generating semantic descriptions from drawings of scenes with shadows." Ph.D. dissertation, AI Lab, MIT, 1972; also in *PCV*, 1975.

WARNOCK, J. G. "A hidden-surface algorithm for computer-generated halftone pictures." TR 4-15, Computer Science Dept., Univ. Utah, June 1969.

WESLEY, M. A. and S. MARKOWSKY. "Fleshing out projections." *IBM J. Res. Devel. 25*, 6 (Nov. 1981), 934–954.

RELATIONAL STRUCTURES IV



Visual understanding relates input and its implicit structure to explicit structure that already exists in our internal representations of the world. More specifically, vision operations must maintain and update *beliefs* about the world, and achieve specific *goals.*

To consider how higher processes can influence and use vision, one must confront the nonvisual world and powers of reasoning that have more general applicability. The world models that are capable of supporting advanced application-dependent calculations about objects in the visual domain are quite complex. General techniques of *knowledge representation* developed in other fields of artificial intelligence can be brought to bear on them. Similarly, much research has been invested in the basic processes of *inference* and *planning.* These techniques may be used in the visual domain to manipulate beliefs and achieve goals, as well as reasoning for other purposes.

The organization of a complex visual system (Fig. 1.5 or Fig. 10.1), is a loose hierarchy of models of world phenomena. The *relational models* that concern us in this chapter are removed from direct perceptual experience—they are used mainly for the last, highest-level stages of perception. Also, they are used for knowledge attained prior to the visual experience currently being processed. The representations involved may be *analogical* or *propositional.* Analogical representations allow *simulations* of important physical and geometric properties of objects. Propositions are assertions that are either true or false with respect to the world (or a world model). Each form is useful for different purposes, and one is not necessarily "higher" than the other. The techniques and representations of Part IV are mainly propositional in flavor. Sometimes the reasoning they implement (say about geometrical entities) would seem better suited to analogical calculations; however, technical difficulties can render that impossible.

Part IV is concerned with techniques for making the "motivation" and "world view" of a vision system explicit and available. Such explicit models would be interesting from a scientific standpoint even if they were not directly useful. But explicitly available models are decidedly useful. They are useful to the system designer who desires to reconfigure or extend a system. They are useful to the system itself, which can use them to reason about its own actions, flexibly control its own resources in accordance with higher goals, dynamically change its goals, recover from mistakes, and so forth.

We organize the major topics of Part IV as follows.

1. Knowledge representation (Chapter 10). *Semantic nets* are an important technique for structuring complex knowledge, and can be used as a knowledge representation formalism in their own right.
2. Matching (Chapter 11). *Matching* puts a derived representation of an image into correspondence with an existing representation. This style of processing representations is more pronounced as domain-dependent knowledge, idiosyncratic goals, and experience begin to dominate the ultimate use (or understanding) of the visual input.
3. Inference (Chapter 12). Classical *logical inference* (a technique for manipulating purely propositional knowledge representations) is a well-understood and elegant reasoning technique. It has good formal properties, but occasionally seems restricted in its power to duplicate the range of human processing. *Extended inference* techniques such as *production systems* are those in which the inference process as well as the propositions may contribute materially to the derived knowledge. *Labeling* techniques can "infer" consistent or likely interpretations for an input from given rules about the domain. Inference can be used for both problem solving and belief-maintenance activity.
4. Planning (Chapter 13). *Planning* techniques are useful for problem solving, and are especially tailored to integrating vision with real-world *action.* Planning can be used for resource allocation and attentional mechanisms.
5. Control (Chapter 10; Appendix 2). Control *strategies* and *mechanisms* are of vital concern in any complex artificial intelligence system, and are particularly important when the computation is as expensive as that of vision processing.

Learning is missing from the list above. Disappointing as it is, at this writing the problem of learning is so difficult that we can say very little about it in the domain of vision.

Knowledge Representation and Use

10

10.1 REPRESENTATIONS

An internal representation of the world can help an intelligent system plan its actions and foresee their consequences, anticipate dangers, and use knowledge acquired in the past. In Part IV we investigate the creation, maintenance, and use of a *knowledge base*, an abstract representation of the world useful for computer vision. Chapter 1 introduced a layered organization for the knowledge base and divided its contents into "analogical" and "propositional" models. In this section we consider this high-level division more deeply.

The outside world is accessible to a computer vision program through the imaging process. Otherwise, the program is manipulating its internal representations, which should correspond to the world in understood ways. In this sense, the knowledge base of generalized images, segmented images, and geometric entities contains "models" of the phenomena in the world. Another more abstract sense of "model" is high-level, prior expectations about how the world fits together. Such a high-level model is often much more complex than the lower-level representations, often has a large "propositional" component, and is often manipulated by "inference-like" procedures. Explicit knowledge and belief structures are a relatively new phenomenon in computer vision, but are playing an increasingly important role.

The goals of this chapter are three.

1. To develop in more depth some issues of high-level models (Section 10.1).
2. To describe *semantic nets*—an important and general tool for both organizing and representing models (Sections 10.2 and 10.3).
3. To address issues of *control*, at both abstract and implementational levels (Section 10.4 augmented by Appendix 2).

10.1.1 The Knowledge Base—Models and Processes

Figure 10.1 shows the representational layers in the knowledge base as we have developed it through the book, and shows the place of important processes. This organization might be compared with that in [Barrow and Tenenbaum 1981].

The knowledge base organization is mirrored in the organization of the book. Parts I to III dealt with analogical models and their construction; Part IV is concerned with propositional and complex analogical models. In Chapters 11 to 13, the emphasis moves from the structure of models to the processes (matching, inference, and planning) needed to manipulate and use them.

The knowledge base should have the following properties.

- Represent analogical, propositional, and procedural structures
- Allow quick access to information
- Be easily and gracefully extensible
- Support inquiries to the analogical structures
- Associate and convert between structures
- Support belief maintenance, inference, and planning

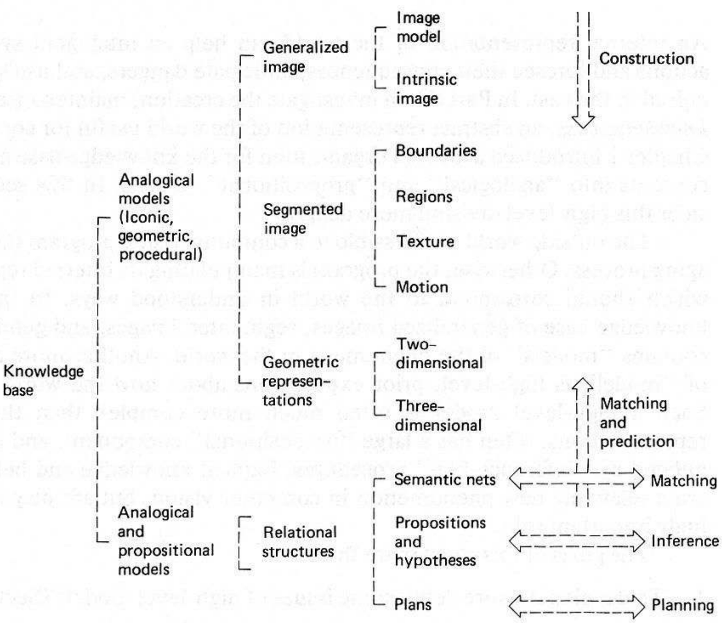


Fig. 10.1 The knowledge base and associated processes in a computer vision system.

The highest levels of the knowledge base contain both *analogical* and *propositional* models. Analogical tools do not exist for many important activities, and when they do exist they are often computationally intensive. A three-dimensional geometric modeling system for automatic manufacturing has very complex data structures and algorithms compared to their elegant and terse counterparts in a propositional model that may be used to plan the highest-level actions. In general it makes sense to do some computation at the analogical level and some at the propositional. This multiple-representation strategy seems more efficient than translating all problems into one representation or the other.

The computations in a vision system should be organized so that information can flow efficiently and unnecessary computation is kept to a minimum. This is the function of the *control* disciplines that allocate effort to different processes. Even the simplest biological vision systems exhibit sophisticated control of processing.

Constructive processes dominate the activity in building lower-level models, and *matching* processes become more important as prior expectations and models are brought into play. Chapter 11 is devoted to the process of matching.

We postulate that an advanced vision system is engaged in two sorts of high-level activity: *belief maintenance* and *goal achievement.* The former is a more or less passive, data-driven, background activity that keeps beliefs consistent and updated. The latter is an active, knowledge-driven, foreground activity that consists of planning future activities. Planning is a problem-solving and simulation activity that anticipates future world states; in computer vision it can determine how the visual environment is expected to change if certain actions are performed. Planning can occur with symbolic, propositional representations (Chapter 13) or in a more analogical vein with such simulations as trajectory planning [Lozano-Perez and Wesley 1979]. Planning is useful as an implementational mechanism even in contexts that are not analogous to human "conscious" problem solving [Garvey 1976]. Helmholtz likened the results of perception to "*unconscious conclusions*" [Helmholtz 1925]. Similarly even "primitive" vision processes (computer or biological) may use planning techniques to accomplish their ends.

Inference and planning are both classical subfields of artificial intelligence. Neither has seen much application in computer vision. Inference seems useful for belief maintenance. Extended inference can deal with inconsistent beliefs and with beliefs that are maintained with various strengths. We treat inference in Chapter 12. Applications of planning to vision [Garvey 1976; Bolles 1977] show good promise. Planning is treated in Chapter 13.

10.1.2 Analogical and Propositional Representations

Our division of the internal knowledge base into "analogical" and "propositional" reflects a similar division in theories of how human beings represent the world [Johnson-Laird 1980]. Psychological data are not compelling toward either pure theory; there are indications that human beings use both forms of representation. We introduce the division in this book because we find it conceptually useful in the following way. Low-level representations and processes tend to be purely analogical; high-level representations and processes tend to be both analogical and propositional.

Analogical representations have the following characteristics [Kosslyn and Pomerantz 1977; Shepard 1978; Sloman 1971; Kosslyn and Schwartz 1977, 1978; Waltz and Boggess 1979].

1. *Coherence.* Each element of a represented situation appears once, with all its relations to other elements accessible.
2. *Continuity.* Analogous with continuity of motion and time in the physical world; these representations permit continuous change.
3. *Analogy.* The structure of the representation mirrors (and may be isomorphic to) the relational structure of the represented situation. The representation is a description of the situation.
4. *Simulation.* Analogical models are interrogated and manipulated by arbitrarily complex computational procedures that often have the flavor of (physical or geometric) simulation.

Propositional representations have the following characteristics [Anderson and Bower 1973; Palmer 1975; Pylyshyn 1973].

1. *Dispersion.* An element of a represented situation can appear in several propositions. However, the propositions can be represented in a coherent manner by using semantic nets.
2. *Discreteness.* Propositions are not usually used to represent continuous change. However, they may be made to approximate continuous values arbitrarily closely. Small changes in the representation can thus be made to correspond to small changes in the represented situation.
3. *Abstraction.* Propositions are true or false. They do not have a geometric resemblance to the situation; their structure is not analogous to that of the situation.
4. *Inference.* Propositional models are manipulated by more or less uniform computations that implement "rules of inference" allowing new propositions to be developed from old ones.

Each sort of model derives its "meaning" differently; the distinctions are interesting, because they can point out weaknesses in each theory [Johnson-Laird 1980; Schank 1975; Fodor, et al. 1975]. Especially in computer implementations, the two representations only differ essentially in the last two points. It is often possible to transform one representation to another without loss of information.

Some examples are in order. A generalized image (Part I) is an analogical model: to find an object above a given object, a procedure can "search upward" in the image. An unambiguous three-dimensional model of a solid (Chapter 9) is analogical. It may be used to calculate many geometric properties of the solid, even those unimagined by the designer of the representation. A set of predicate calculus clauses (Chapter 12) is a propositional model. Closely related models can be used to solve problems and make plans [Nilsson 1971, 1980; Chapter 13].

A short digression: It is interesting that people do not seem to perform syllogistic inference (formal propositional deduction) in a "mechanical" way. Given two clauses such as "Some appliances are telephones" and "All telephones are black," we are much more likely to conclude "Some appliances are black" than the equally valid "Some black things are appliances." There is not a satisfying theory of the mental processes underlying syllogistic inference. An interesting speculation [Johnson-Laird 1980] is that inference is primarily done through analogical mental models (in which, for example, a population of individuals is conjured up and manipulated). Then syllogistic inference techniques may have arisen as a bookkeeping mechanism to assure that analogical reasoning does not "miss any cases."

10.1.3 Procedural Knowledge

Procedures as explicit elements in a model pose problems because they are not readily "understood" by other knowledge base components. It is very hard to tell what a procedure does by looking at its code.

In our taxonomy we think of "procedural" knowledge as being analogical. The sequential nature of a program's steps is analogous to an ordering of actions in time that can only be clumsily expressed in current propositional representations. Knowledge about "how-to" perform a complex activity is most propitiously represented in the form of explicit process descriptions. Descriptions not involving the element of time may be naturally represented as passive (analogical or propositional) structures.

There have been several attempts to organize chunks of procedural knowledge by associating with the procedure a description of what it is to accomplish. For example, procedural knowledge can be stored in the internal model structure (knowledge base) indexed under *patterns* that correspond to the arguments of the procedure. *Pattern-directed invocation* involves going to the knowledge base for a procedure that matches the given pattern, matching pattern elements to bind arguments, and invoking the procedure. Several advantages accrue in pattern-directed invocation, such as not having to know the "proper names" of procedures, only their descriptions (what they claim to do). Also, when several procedures match a pattern, one either gets nondeterminism or a chance to choose the best. Often system facilities include a procedure to run to choose the best procedure dynamically. Similar pattern matching is involved in resolution theorem provers and production systems (Chapter 12).

As an example, in a program to locate ribs in a chest radiograph [Ballard 1978], procedures to find ribs under different circumstances are attached to nodes in a mixed analogic and propositional model of the ribcage as shown in Fig. 10.2. Each procedure has an associated description which determines whether it can be run. For example, some programs require instances of neighboring ribs to be located before they can run, whereas others can run given only rudimentary scaling information. When invoked, each procedure tries to find a geometric structure corresponding to the associated rib in a radiograph. Instead of searching for ribs in a mechanical order, descriptors allow a choice of order and procedures and hence a

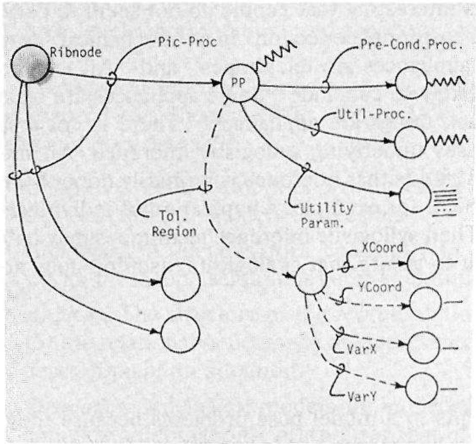


Fig. 10.2 A portion of a ribcage model (see text). Procedural attachment to a model is denoted by jagged lines.

more flexible, efficient and robust program (Appendix 2).

The representation and use of procedural knowledge is an important topic [Schank and Abelson 1977; Winograd 1975; Freuder 1975]. We expect it to be increasingly important for computer vision.

10.1.4 Computer Implementations

A computer implementation can (and often does) obscure the sharp divisions imposed by pure philosophical differences between analogical and propositional models. A propositional representation need not be an unordered set of clauses, but may have a coherent structure; the coherent versus dispersed distinction is thus blurred. A geometry theorem prover or a block-stacking program may manipulate diagrams or simulate physical phenomena such as gravitational stability and wobble in the manipulator [Gelernter 1963; Fahlman 1974; Funt 1977]. "Nonstandard inference" is an important tool that extends classical inference techniques. Although techniques such as production systems and relaxation labeling algorithms (Chapter 11) bear little superficial resemblance to predicate logic, both may be naturally used to manipulate propositional models.

Propositions may be implemented as procedures. If a proposition "evaluates" to true or false, it is perhaps most naturally considered a function from a world (or world model) to a truth value. This is not to say that all such functions exist or are evaluated when the proposition is "brought to mind"; perhaps "understanding a proposition" is like compiling a function and "verifying a proposition" is like evaluating it. The function may be implicit in an evaluation (inference) mechanism or more explicit, as in a "procedural" semantics such as that of the programming languages PLANNER and CONNIVER [Hewitt 1972; Sussman and McDermott 1972; Winograd 1978]. A proposition may thus be encoded as an (analogical!) procedural recipe for establishing the proposition. An example might be this representation of the fact "In California, Grass and Trees produce green regions."

(To-Establish (GreenRegion *x*)
  Establish (AND (InCalifornia())
    (OR (Establish (Grass *x*))
    (Establish (Trees *x*)))))

This might mean: To infer that x is a green region, establish that you are in California and then try to establish that x arose from grass. Should the grass inference fail, try to establish that x arose from trees. Since the full power of the programming language is available to an Establish statement, it can perform general computations to establish the inference.

The important point here: Rather than a set of clauses whose application must be organized by an interpreter, propositions may be represented by an explicit control sequence, including procedure calls to other programs. In the example, (Grass *x*) and (Trees *x*) may be procedures which have their own complicated control structures.

To say that in a computer "everything is propositions" is a truism; any program can be reduced to a Turing machine described by a finite set of "propositions" with a very simple rule of "inference." The issue is at what level the program should be described. A program may be doing propositional resolution theorem proving or analogical trajectory planning with three-dimensional models; it is not helpful to blur this basic functional distinction by appealing to the lowest implementational level.

10.2 SEMANTIC NETS

10.2.1 Semantic Net Basics

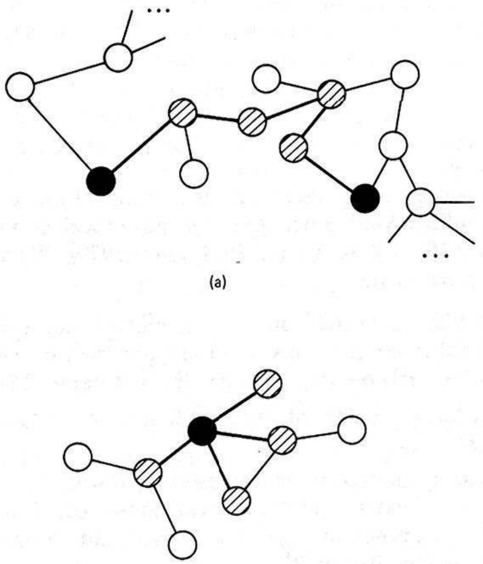
Semantic nets were first introduced under that name as a means of modeling human associative memory [Quillian 1968]. Since then they have received much attention [Nilsson 1980; Woods 1975; Brachman 1976; Findler 1979]. We are concerned with three aspects of semantic nets.

1. Semantic nets can be used as a data structure for conveniently accessing both analogical and propositional representations. For the latter their construction is straightforward and based solely on propositional syntax (Chapter 12).
2. Semantic nets can be used as an analogical structure that mirrors the relevant relations between world entities.
3. Semantic nets can be used as a propositional representation with special rules of inference. Both classical and extended inference can be supported, but it is a challenging enterprise to design net structure that provides the properties of formal logic [Schubert 1976; Hendrix 1979].

A semantic network represents objects and relationships between objects as a graph structure of *nodes* and (labeled) *arcs.* The arcs usually represent relations between nodes and may be "followed" to proceed from node to node. A directed arc with label L between nodes X and Y can signify that the predicate $L(X, Y)$ is true. If, in addition, it has a value V, the arc can signify that some function or relation holds: $L(X, Y) = V$.

The *indexing property* of a network is one of its useful aspects. The network can be constructed so that objects that are often associated in computations, or are especially relevant or conceptually close to each other, may be represented by nodes in the network that are near each other in the network (as measured by number of arcs separating them). Figure 10.3 shows these ideas: (a) nodes can be associated by searching outward along arcs and (b) nodes near a specified node are readily available by following arcs. Semantic networks are especially attractive as analogical representations of spatial states of affairs. If we restrict ourselves to binary spatial relations ("above," and "west of," for example), physical objects or parts of objects may be represented by nodes, and their positions with respect to each other by arcs.

Let us look at a semantic net and make some basic observations. Figure 10.4 is meant to be an analogical representation of an arrangement of chairs around a table. The LEFT-OF and RIGHT-OF relations are directed arcs, the ADJACENT relation is undirected; there can be several such undirected arcs between nodes. Note here that the LEFT-OF and RIGHT-OF relations do not behave in their normal way. If they are transitive, as is normal, then every chair is both LEFT-OF and



(a)

(b)

Fig. 10.3 Semantic networks as structures for associative search. (a) Associating two nodes. (b) Retrieving nearby nodes.

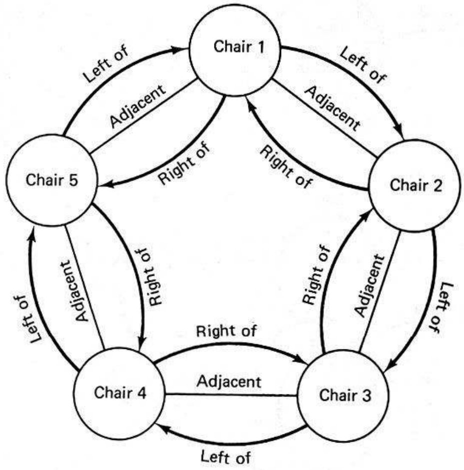


Fig. 10.4 A representation of chairs at a table.

RIGHT-OF every other chair. Flexible treatment of this sort of phenomenon is sometimes difficult in propositional representations.

A simple but basic point: The net of Fig. 10.4 seems to say interesting things about furniture in a scene. But notice that merely by rewriting labels the same net could be "about" modular arithmetic, a string of pearls, or any number of things. There are two morals here. First, a sparsely connected representation (analogical or propositional) may have several equally good interpretations. Second, a net without any interpretation procedures essentially represents nothing [McDermott 1976].

Now consider three neighboring chairs described by the following relations.

1. CHAIR(Armchair), CHAIR(Highchair), CHAIR(Stool)
2. WIDE(Armchair)
3. HIGH(Highchair)
4. LOW(Stool)
5. LEFT-OF(Armchair, Highchair)
6. LEFT-OF(Highchair, Stool)
7. BETWEEN(Highchair, Armchair, Stool)

The relations include four properties (relations with "one argument"), a two-argument and a three-argument relation. One way to encode this information in a net is shown in Fig. 10.5a. Nodes represent individuals, and properties are kept as node contents. The directed arcs represent only binary relations, and "betweenness" is left implicit. Properties can equally well be represented as labeled arcs (Fig. 10.5b).

Relations are encoded as nodes in Fig. 10.6. Here the BETWEEN relation is encoded asymmetrically: it is not possible to tell by arcs emanating from the stool that it is in a "between" relationship.

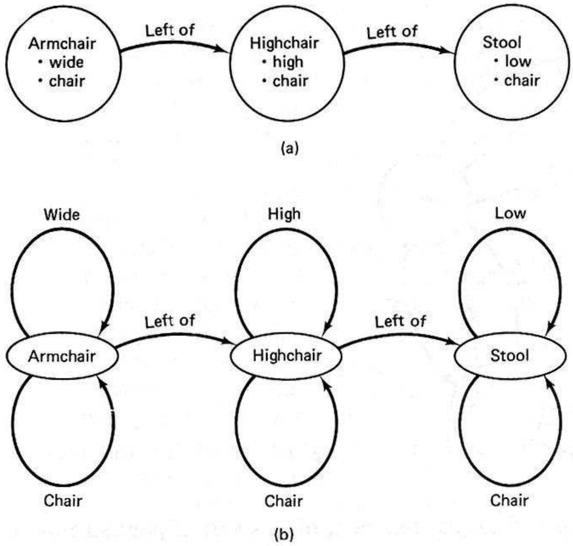


Fig. 10.5 (a) A simple semantic net. (b) An equivalent net.

The three-place relation is treated more symmetrically in Fig. 10.7. In general, *n*-place relations may be "binarized" this way; create a node for the "relation instance" and new (relation) nodes for each distinct argument role in the *n*-ary relation.

An important point: Arcs and nodes had a uniform semantics in Fig. 10.4. This property was lost in the succeeding nets; nodes are either "things" or relations, and arcs leading into relations are not the same as those leading out. For such nets to be useful, the net interpreter (a program that manipulates the net) must keep these things straight. It is possible but not easy to devise a rich and uniform network semantics [Brachman 1979].

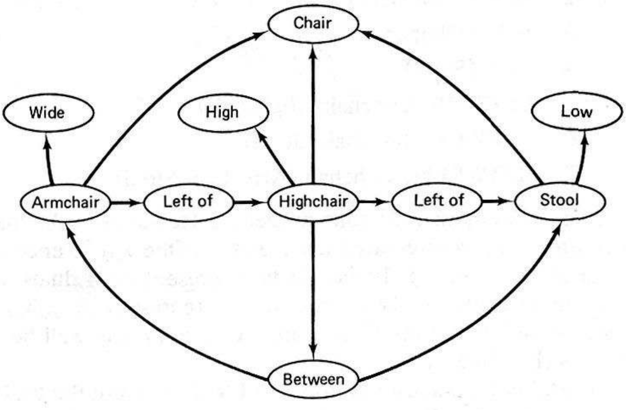


Fig. 10.6 A net with more explicit information.

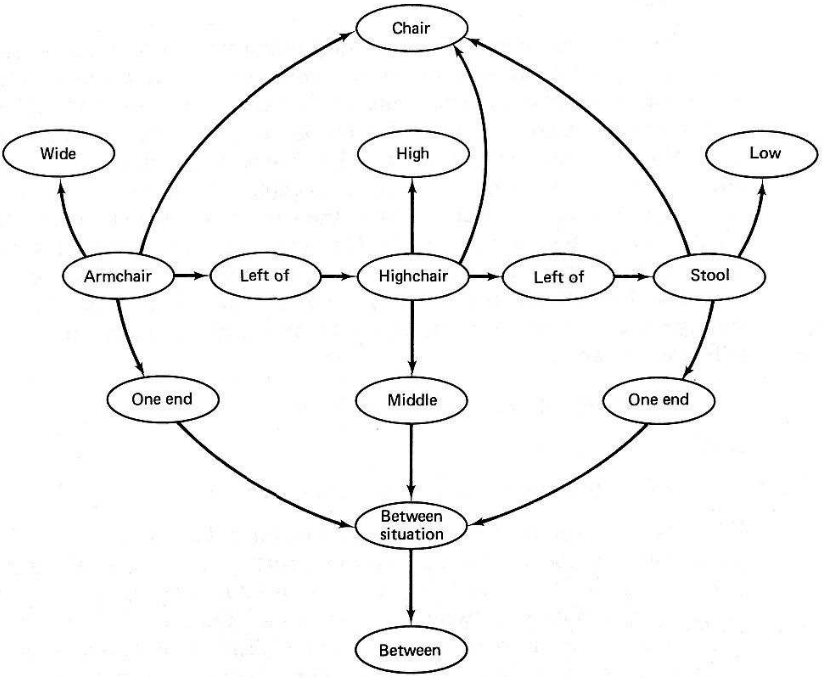


Fig. 10.7 A net with yet more explicit information.

10.2.2 Semantic Nets for Inference

This section explores some further important issues in the semantics of semantic nets. In Chapter 12 semantic nets are used as an indexing mechanism in predicate calculus theorem proving. In some applications an inference system with provably good formal properties may be too restrictive. Some formal properties (such as maintaining consistency by not deducing contradictions) may be considered vital, however. How can "good behavior" be obtained from a representation that may contain "inconsistent" information?

One example of an "inconsistent" representation is the net of Fig. 10.3, with its LEFT-OF and RIGHT-OF problem. Another example is a net version of the propositions "All birds fly," "Penguins are birds," "Penguins do not fly." The generalization is useful "commonsense" knowledge, but the rare exceptions may be important, too. Network interpreters can cope with these sorts of problems by a number of methods, such as only accessing a consistent subnetwork, making deductions from the particular toward the general (this takes care of penguins), and so forth. All these techniques depend on the structure imposed by the net.

Some more subtle aspects of net representations appear below.

Nodes

The basic notation of Fig. 10.4 may tempt us to produce a net such as that shown in Fig. 10.8. Consider the object node sky in Fig. 10.8. Does it stand for the generic sky concept or for a particular sky at a particular time and location? Clearly both meanings cannot be embodied in the same node because they are used in such different ways in reasoning. The standard solution is to use nodes to differentiate between a *type*, or generic concept, and a *token*, or instance of it. Figure 10.9 shows this modification using the *e* (element of) relation to relate the individual to the generic concept. In this simple case, the node sky stands for the *type*, and the empty node stands for a *token*, or instance of the sky concept.

The distinction between type and token is related to the distinction between *intensional* and *extensional* concepts. In analyzing an aerial image there is a difference between

"All bridges span roads or rivers." (10.1)

and

"All bridges (found so far) span roads or rivers." (10.2)

If "bridges" in (10.1) means *any* bridge that might be found, "bridges" is used in an *intensional* sense. If "bridges" means a particular set, it is used it in an *extensional* sense. Normally relations between *type* nodes are used in an intensional sense and relationships between *token* nodes have the extensional sense.

Virtual nodes are objects that are not explicitly represented as object nodes. The need for them arises in expressing complex relations. For example, consider

"The bridge that is at the intersection of road 57
and river 3 is near building 30." (10.3)

which may be represented as shown in Fig. 10.10. The node labeled x is the result of intersecting a particular road with a particular river. It is not represented explicitly as an instance of any generic concept; it is a *virtual node*. Virtual nodes can be eliminated by introducing very complex relations, but this would sacrifice an important property of networks, the ability to build up a very large number of com-

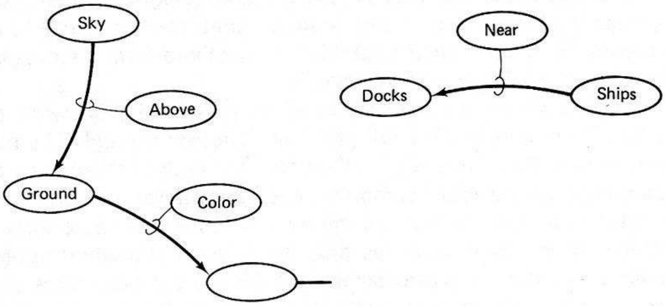


Fig. 10.8 Type or token nodes?

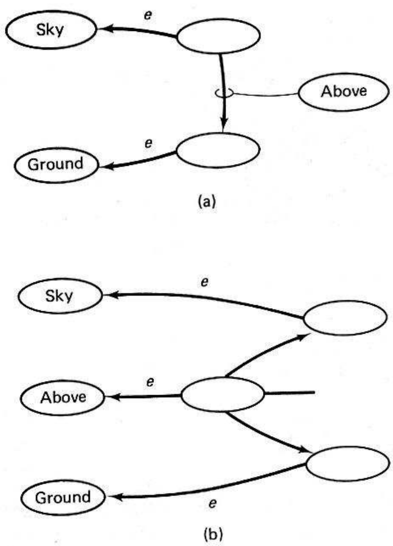


Fig. 10.9 Distinguishing between types and tokens: (a) Tokenizing an instance. (b) Tokenizing an assertion.

plex relations from a small set of primitives. Virtual nodes enhance this ability by referring to portions of complicated relations.

Nodes in the network can also be used as *variables.* These variables can match other nodes which represent constants. In Fig. 10.11, x and y are variables and the rest of the nodes are constants. If node x is matched to the "telephone" node, then x can be regarded as a "telephone" node.

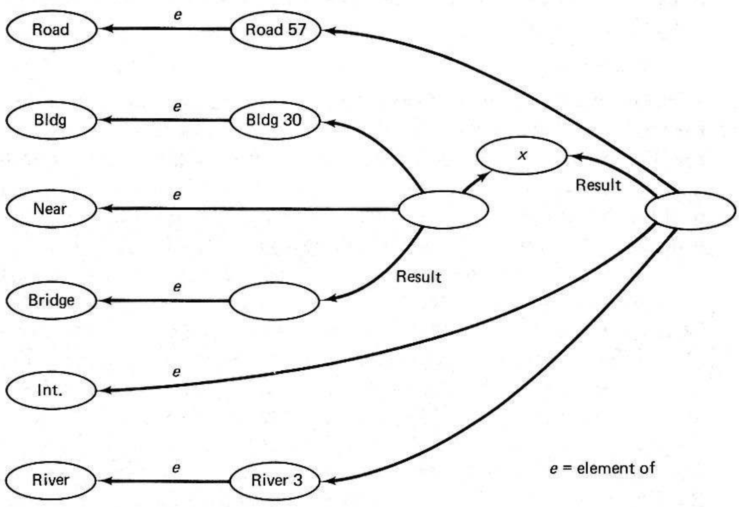


Fig. 10.10 Virtual nodes.

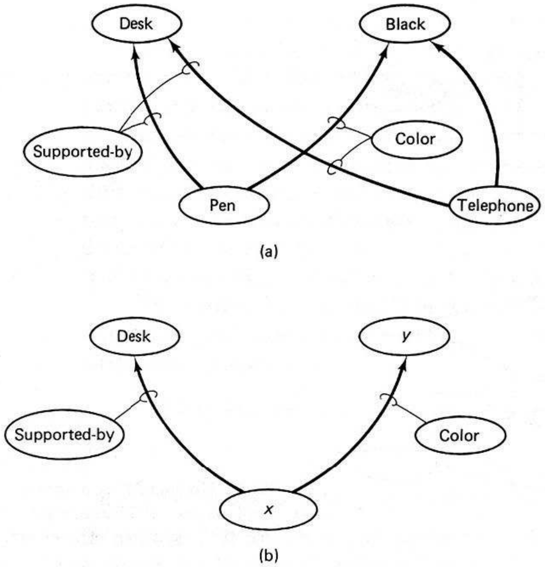


Fig. 10.11 Nodes as variables. (a) Black telephone and pen on desk. (b) Object denoted by variable x with variable color y.

Often, it is useful to have numerical values as node properties. This can extend the discrete representation of nodes and arcs to a continuous one. For example, in addition to "color of x is red37" we may also associate the particular value of red that we mean with node red37. A special kind of value is a *default value.* If a value can be found for the node in the course of matching other nodes with values or by examining image data, then that value is used for the node value. Otherwise, the default is used.

Relations

Complex relations of many arguments are not uncommon in the world, but for the bulk of practical work, relations of only a few arguments seem to suffice. Semantic nets can clearly represent two-argument relations through their nodes and arcs. More complex relations may be dealt with by various devices. The links to multiple arguments may be ordered within a relation node, or new nodes may be introduced to label the roles of multiple arguments (Fig. 10.7).

If inference mechanisms are to manipulate semantic nets, certain important relations deserve special treatment. One such relation is the "IS-A" relation. The basic issue addressed by this relation is *property inheritance* [Moore 1979]. That is, if Fred IS-A Camel and a Camel IS-A Mammal, then presumably Fred has the properties associated with mammals. It often seems necessary to differentiate between various senses of "IS-A." One basic sense of "X IS-A Y" is "X is an element of the set Y"; others are "X denotes Y," "X is a subset of Y," and "Y is an abstraction of X." Notice that each sense depends on differently "typed" arguments; in the first three cases X is, respectively, an individual, a name, and a set. Deeper treatments of these issues are readily available [Brachman 1979; Hayes* 1977; Nilsson 1980].

It is particularly helpful to have a denotion link to keep perceptual structures separate from model structures. Then if mistakes are made by the vision automaton, a correction mechanism can either sever the denotation link completely or create a new denotation link between the correct model and image structures.

When dealing with many spatial relations, it is economical to recognize that many relations are "inverses" of each other. That is, LEFT-OF(x,y) is the "inverse" of RIGHT-OF(x,y);

$$\text{LEFT-OF}(x,y) \iff \text{RIGHT-OF}(y,x)$$

and also

$$\text{ADJACENT}(x,y) \iff \text{ADJACENT}(y,x)$$

Rather than double the number of these kinds of links, one can *normalize* them. That is, only one half of the inverse pair is used, and the interpreter infers the inverse relation when necessary.

Properties have a different semantics depending on the type of object that has the property. An "abstract" node can have a property that gives one aspect or refinement of the represented concept. A property of a "concrete" node presumably means an established and quantified property of the individual.

Partitions

Partitions are a powerful notion in networks. "Partition" is not used in the sense of a mathematical partition, but in the sense of a barrier. Since the network is a graph, it contains no intrinsic method of delimiting subgraphs of nodes and arcs. Such subgraphs are useful for two reasons:

1. *Syntactic.* It is useful to delimit that part of the network which represents the results of specific inferences.
2. *Semantic.* It is useful to delimit that part of the network which represents knowledge about specific objects. Partitions may then be used to impose a hierarchy upon an otherwise "flat" structure of nodes.

The simple way of representing partitions in a net is to create an additional node to represent the partition and introduce additional arcs from that node to every node or arc in the partition. Partitions allow the nodes and relations in them to be manipulated as a unit.

Notationally, it is cleaner to draw a labeled boundary enclosing the relevant nodes (or arcs). An example is shown by Fig. 10.12 where we consider two objects each made up of several parts with one object entirely left of the other. Rather than use a separate LEFT-OF relation for each of the parts, a single relation can be used between the two partitions. Any pair of parts (one from each object) should inherit the LEFT-OF relation. Partitions may be used to implement quantification in semantic net representations of predicate calculus [Hendrix 1975, 1979]. They may be used to implement frames (Section 10.3.1).

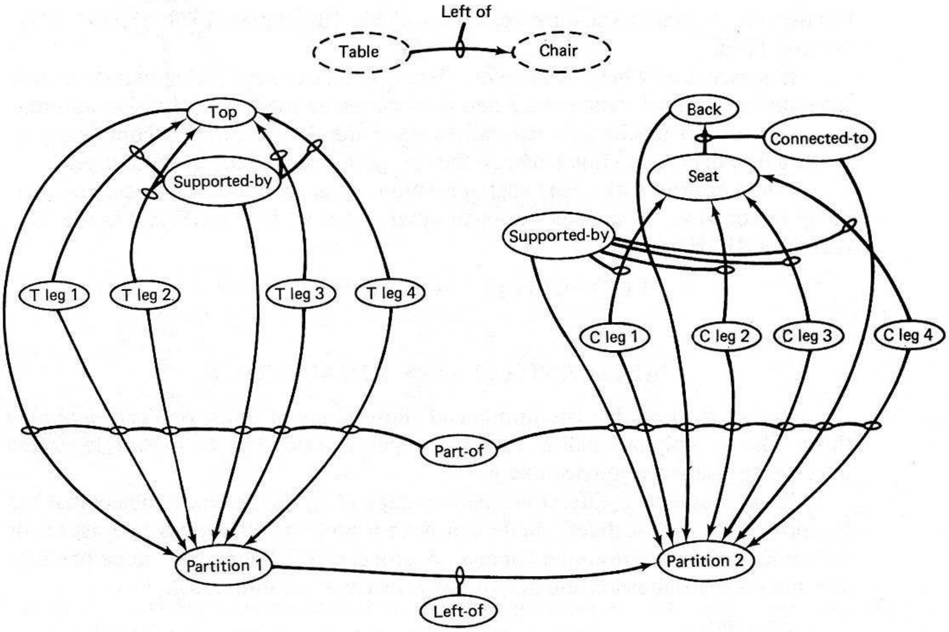


Fig. 10.12 The use of partitions. (a) Construction of a partition. (b) Two objects described by partitions.

Conversions

It is important to be able to transform from geometric (and logical) representations to propositional abstract representations and vice versa. For example, in Fig. 10.13 the problem is to find the exact location of a telephone on a previously located desk. In this case, propositional knowledge that telephones are usually on desktops, together with the desk top location and knowledge about the size of telephones, define a search area in the image.

Converting image data about a particular group of objects into relational form involves the inverse problem. The problem is to perform a level of abstraction to remove the specificity of the geometric knowledge and derive a relation that is appropriate in a larger context. For example, the following program fragment creates the relations ABOVE(A,B), where A and B are world objects.

Comment: assume a world coordinate system where Z is the positive vertical.

> Find $ZA_{\min}$ for Z in A and $ZB_{\max}$ for Z in B.
> If $ZA_{\min} > ZB_{\max}$, then make ABOVE (A,B) true.

Many other definitions of ABOVE, one of which compares centers of gravity, are possible. In most cases, the conversion from continuous geometric relations to discrete propositional relations involves more or less arbitrary conventions. To appreciate this further, consult Fig. 10.14 and try to determine in which of the cases

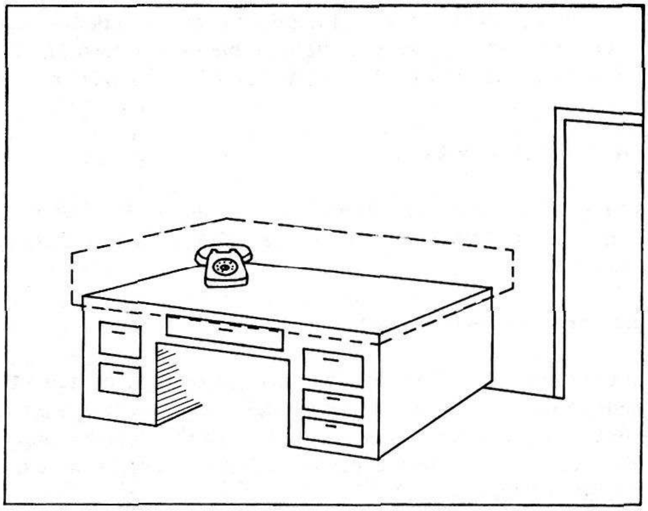


Fig. 10.13 Search area defined by relational bindings.

block A is LEFT-OF block B. Figure 10.14d shows a case where different answers are obtained depending on whether a two-dimensional or three-dimensional interpretation is used. Also, when relations are used to encode what is *usually* true of the world, it is often easy to construct a counterexample. Winston [Winston 1975] used

SUPPORTS (B,A) ABOVE (A,B)

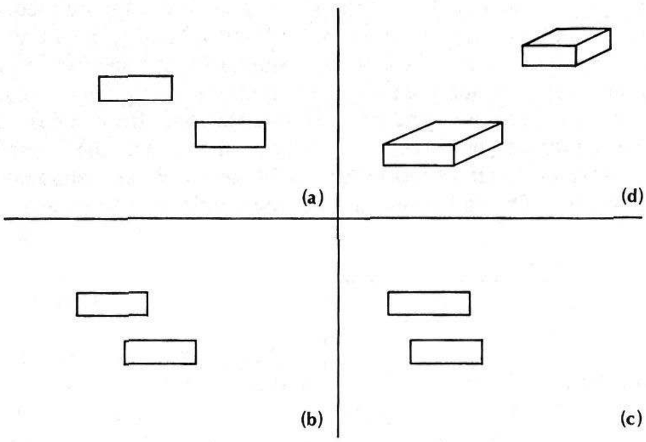


Fig. 10.14 Examples to demonstrate difficulties in encoding spatial relation LEFT-OF (see text).

which is contradicted by Fig. 10.15, given the previous definition of ABOVE.

One common way around these problems is to associate quantitative, "continuous" information with relations (section 10.3.2 and later examples).

10.3 SEMANTIC NET EXAMPLES

Examples of semantic nets abound throughout Part IV. Two more examples illustrate the power of the notions. The first example is described very generally, the second in detail.

10.3.1 Frame Implementations

Frame system theory [Minsky 1975] is a way of explaining our quick access to important aspects of a (perhaps perceptual) situation. It is a provocative and controversial idea, and the reader should consult the References for a full treatment. Implementationally, a frame may be realized by a partition; a frame is a "chunk" of related structure.

Associating related "chunks" of knowledge into manipulable units is a powerful and widespread idea in artificial intelligence [Hayes 1980; Hendrix 1979] as well as psychology. These chunks go by several names: units, frames, partitions, schemata, depictions, scripts, and so forth [Schank and Abelson 1977; Moore and Newell 1973; Roberts and Goldstein 1977; Hayes* 1977; Bobrow and Winograd 1977, 1979; Stefik 1979; Lehnert and Wilks 1979; Rumelhart et al. 1972].

Frames systems incorporate a theory of associative recall in which one selects frames from memory that are relevant to the situation in which one finds oneself. These frames include several kinds of information. Most important, frames have *slots* which contain details of the viewing situation. Frame theory dictates a strictly specific and prototypical structure for frames. That is, the number and type of slots for a particular type of frame are immutable and specified in advance. Further, frames represent specific prototype situations; many slots have default values; this is where expectations and prior knowledge come from. These default values may be disconfirmed by perceptual evidence; if they are, the frame can contain information about what actions to take to fill the slot. Some slots are to be filled in by investigation. Thus a frame is a set of expectations to be confirmed or disconfirmed

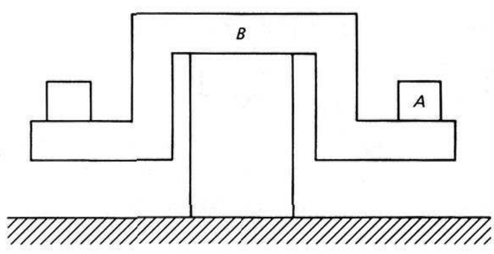


Fig. 10.15 A counterexample to SUPPORTS(B, A) => ABOVE(A, B).

and actions to pursue in various contingencies. One common action is to "bring in another frame."

The theory is that based on a partial match of a frame's defining slots, a frame can be "brought to mind." The retrieval is much like jumping to a conclusion based on partial evidence. Once the frame is proposed, its slots must be matched up with reality; thus we have the initial major hypothesis that the frame represents, which itself consists of a number of minor subhypotheses to be verified. A frame may have other frames in its slots, and so frames may be linked into "frame systems" that are themselves associatively related. (Consider, for example, the linked perceptual frames for being just outside a theater and for being just inside.) Transformations between frames correspond to the effects of relevant actions. Thus the hypotheses can suggest one another. "Thinking always begins with suggestive but imperfect plans and images; these are progressively replaced by better—but usually still imperfect—ideas" [Minsky 1975].

Frame theory is controversial and has its share of technical problems [Hinton 1977]. The most important of these are the following.

1. Multiple instances of concepts seem to call for copying frames (since the instances may have different slotfillers). Hence, one loses the economy of a preexisting structure.
2. Often, objects have variable numbers of components (wheels on a truck, runways in an airport). The natural representation seems to be a rule for constructing examples, not some specific example.
3. Default values seem inadequate to express legal ranges of slot-filling values or dependencies between their properties.
4. Property inheritance is an important capability that semantic nets can implement with "is a" or "element-of" hierarchies. However, such hierarchies raise the question of which frame to copy when a particular individual is being perceived. Should one copy the generic Mammal frame or the more specific Camel frame, for instance. Surely, it is redundant for the Camel frame to duplicate all the slots in the Mammal frame. Yet our perceptual task may call for a particular slot to be filled, and it is painful not to be able to tell where any particular slot resides.

Nevertheless, where these disadvantages can be circumvented or are irrelevant, frames are seeing increasing use. They are a natural organizing tool for complex data.

10.3.2 Location Networks

This section describes a system for associating geometric analogical data with a semantic net structure which is sometimes like a frame with special "evaluation" rules. The system is a geometrical inference mechanism that computes (or infers) two-dimensional search areas in an image [Russell 1979]. Such networks have found use in both aerial image applications [Brooks and Binford 1980; Nevatia and Price 1978] and medical image applications [Ballard et al. 1979].

The Network

A *location network* is a network representation of geometric point sets related by set-theoretic and geometric operations such as set intersection and union, distance calculation, and so forth. The operations correspond to restrictions on the location of objects in the world. These restrictions, or rules, are dictated by cultural or physical facts.

Each internal node of the location network contains a geometric *operation*, a list of *arguments* for the operation, and a *result* of the operation. For instance, a node might represent the set-theoretic union of two argument point sets, and the result would be a point set. Inference is performed by *evaluating* the net; evaluating all its operations to derive a point set for the top (root) operation.

The network thus has a hierarchy of ancestors and descendents imposed on it through the argument links. At the bottom of this hierarchy are *data nodes* which contain no operation or arguments, only geometric data. Each node is in one of three states: A node is *up-to-date* if the data attached to it are currently considered to be accurate. It is *out-of-date* if the data in it are known to be incomplete, inaccurate, or missing. It is *hypothesized* if its contents have been created by net evaluation but not verified in the image.

In a common application, the expected relative locations of features in a scene are encoded in a network, which thus models the expected structure of the image. The primitive set of geometric relations between objects is made up of four different types of operations.

1. *Directional* operations (left, reflect, north, up, down, and so on) specify a point set with the obvious locations and orientations to another.
2. *Area* operations (close-to, in-quadrilateral, in-circle and so on) create a point set with a non-directional relation to another.
3. *Set* operations (union, difference and intersection) perform the obvious set operations.
4. *Predicates* on areas allow point sets to be filtered out of consideration by measuring some characteristic of the data. For example, a predicate testing width, length, or area against some value restricts the size of sets to be those within a permissible range.

The location of the aeration tank in a sewage treatment plant provides a specific example. The aeration tank is often a rectangular tank surrounded on either end by circular sludge and sedimentation tanks (Fig. 10.16). As a general rule, sewage flows from the sedimentation tanks to aeration tanks and finally through to the sludge tanks. This design permits the use of the following types of restrictions on the location of the aeration tanks.

Rule 1: "Aeration tanks are located somewhere close to both the sludge tanks and the sedimentation tanks."

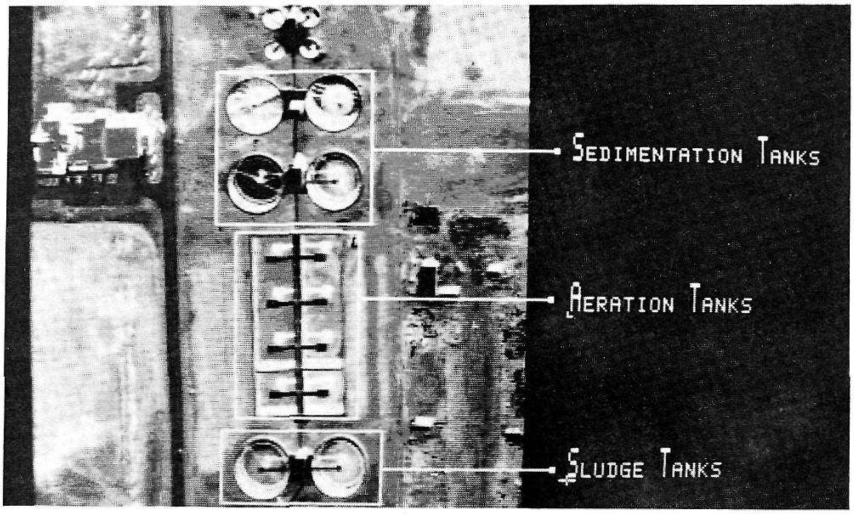


Fig. 10.16 Aerial image of a sewage plant.

The various tanks cannot occupy the same space, so:

> *Rule 2*: "Aeration tanks must not be too close to either the sludge or sedimentation tanks."

Rule 1 is translated to the following network relations.

CLOSE-TO(Union (LocSludgeTanks, LocSedTanks), Distance X)

Rule 2 is translated to

NOT-IN(Union (LocSludgeTanks,LocSedTanks), Distance Y)

The network describing the probable location of the aeration tanks embodies both of these rules. Rule 1 determines an area that is close to both groupings of tanks and Rule 2 eliminates a portion of that area. Thinking of the image as a point set, a set difference operation can remove the area given by Rule 2 from that specified by Rule 1. Figure 10.17 shows the final network that incorporates both rules.

Of course, there could be places where the aeration tanks might be located very far away or perhaps violate some other rule. It is important to note that, like the frames of Section 10.3.1, location networks give prototypical, likely locations for an object. They can work very well for stereotyped scenes, and might fail to perform in novel situations.

The Evaluation Mechanism

The network is interpreted (evaluated) by a program that works top-down in a recursive fashion, storing the partial results of each rule at the topmost node as-

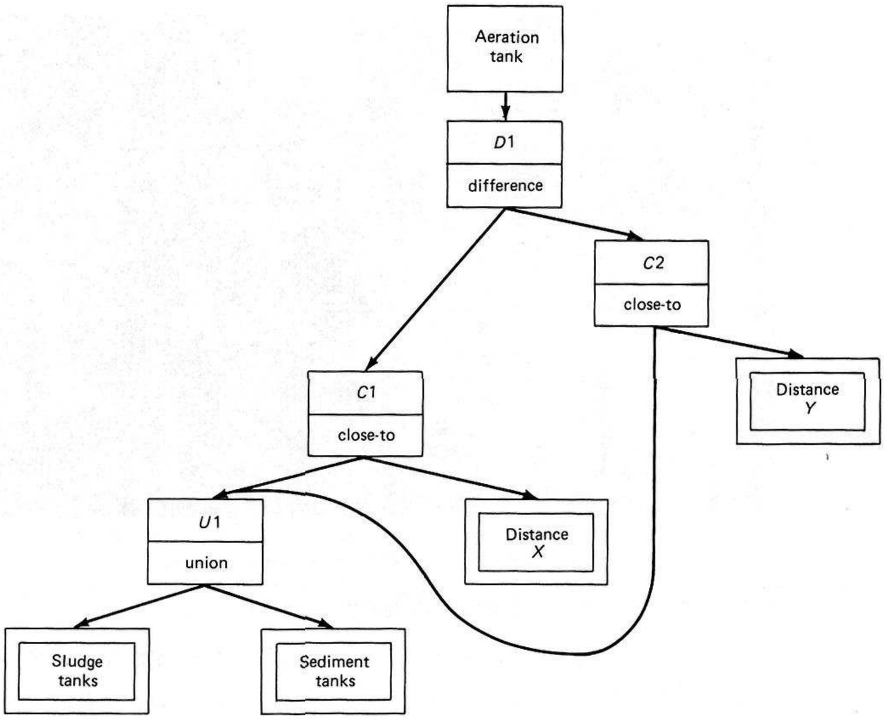


Fig. 10.17 Constraint network for aeration tank.

sociated with that rule (with a few exceptions). Evaluation starts with the root node. In most networks, this node is an operation node. An operation node is evaluated by first evaluating all its arguments, and then applying its operation to those results. Its own result is then available to the node of the network that called for its evaluation.

Data nodes may already contain results which might come from a map or from the previous application of vision operators. At some point in the course of the evaluation, the evaluator may reach a node that has already been evaluated and is marked up-to-date or hypothesized (such a node contains the results of evaluation below that point). The results of this node are returned and used exactly as if it were a data node. Out-of-date nodes cause the evaluation mechanism to execute a low-level procedure to establish the location of the feature. If the procedure is unable to establish the status of the object firmly within its resource limits, the status will remain out-of-date. At any time, out-of-date nodes may be processed without having to recompute any up-to-date nodes. A node marked hypothesized has a value, usually supplied by an inference process, and not verified by low-level image analysis. Hypothesized data may be used in inferences: the results of all inferences based on hypothesized data are marked hypothesized as well.

If a data node ever has its value changed (say, by an independent process that adds new information), all its ancestors are marked out-of-date. Thus the root node will indicate an out-of-date status, but only those nodes on the out-of-date path must be reevaluated to bring the network up to date. Figure 10.18 shows the operation of the aeration tank network of Fig. 10.17 on the input of Fig. 10.16. In this case the initial feature data were a single sludge tank and a single sedimentation tank. Suppose additional work is done to find the location of the remaining sludge and sediment tanks in the image. This causes a reevaluation of the network, and the new result more accurately reflects the actual location of the aeration tanks.

Properties of Location Networks

The location network provides a very general example of use of semantic nets in computer vision.

1. It serves as a data base of point sets and geometric information. The truth status of items in the network is explicitly maintained and depends on incoming information and operations performed on the net.
2. It is an expansion of a geometric expression into a tree, which makes the order of evaluation explicit and in which the partial results are kept for each geometric calculation. Thus it provides efficient updating when some but not all the partial results change in a reevaluation.
3. It provides a way to make geometrical inferences without losing track of the hypothetical nature of assumptions. The tree structure records dependencies among hypotheses and geometrical results, and so upon invalidation of a geometric hypothesis the consequences (here, what other nodes have their values affected) are explicit. The record of dependencies solves a major problem in automated inference systems.
4. It reflects implicit universal quantification. The network claims to represent true relations whose explicit arguments must be filled in as the network is "instantiated" with real data.
5. It has a "flat" semantics. There are no element-of hierarchies or partitions.
6. The concept of "individual" is flexible. A point set can contain multiple disconnected components corresponding to different world objects. In set operations, such an assemblage acts like an explicit set union of the components. An "individual" in the network may thus correspond to multiple individual point (sub)sets in the world.
7. The network allows use of partial knowledge. A set-theoretic semantics of existence and location allows modeling of an unknown location by the set-theoretic universe (the possible location is totally unconstrained). If something is known not to exist in a particular image, its "location" is the null set. Generally, a location is a point set.
8. The set-theoretic semantics allows useful punning on set union and the OR operation, and set intersection and the AND operation. If a dock is on the

shoreline AND near a town, the search for docks need only be carried out in the intersection of the locations.

10.4 CONTROL ISSUES IN COMPLEX VISION SYSTEMS

Computer vision involves the control of large, complex information-processing tasks. Intelligent biological systems solve this control problem. They seem to have complicated control strategies, allowing dynamic allocation of computational resources, parallelism, interrupt-driven shifts of attention, and incremental behavior modification. This section explores different strategies for controlling the complex information processing involved in vision. Appendix 2 contains specific

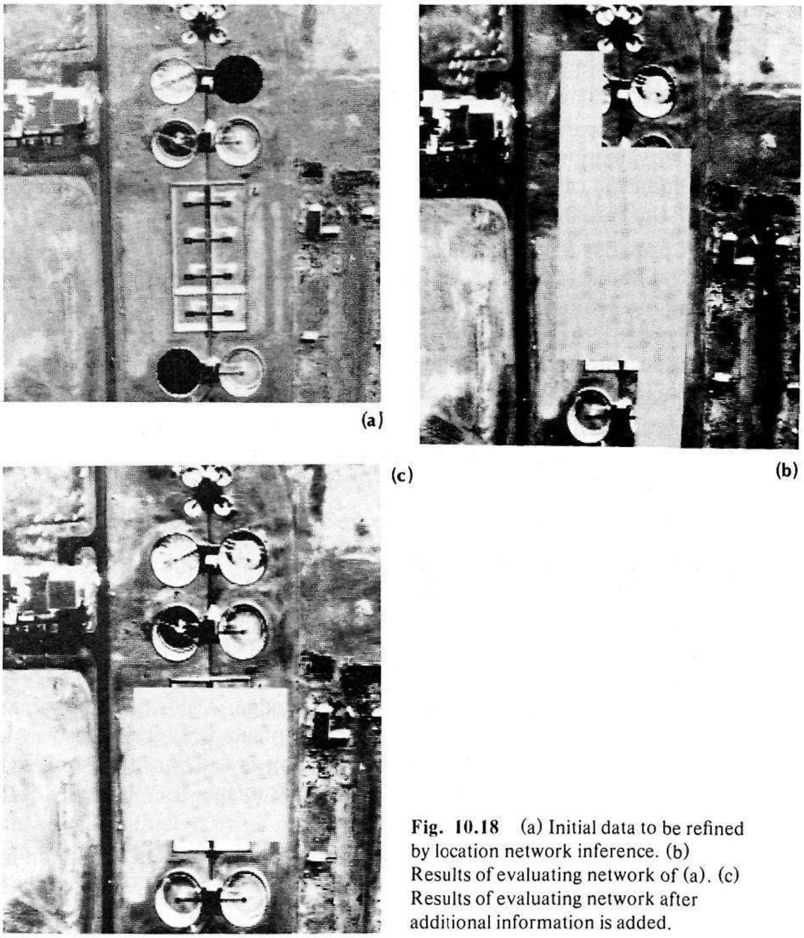

(a) (b) (c)

Fig. 10.18 (a) Initial data to be refined by location network inference. (b) Results of evaluating network of (a). (c) Results of evaluating network after additional information is added.

techniques and programming language constructs that have proven to be useful tools in implementing control strategies for artificial intelligence and computer vision.

10.4.1 Parallel and Serial Computation

In *parallel computation*, several computations are done at the same time. For example, different parts of an image may be processed simultaneously. One issue in parallel processing is synchronization: Is the computation such that the different parts can be done at different rates, or must they be kept in step with each other? Usually, the answer is that synchronization is important. Another issue in parallel processing is its implementation. Animal vision systems have the architecture to do parallel processing, whereas most computer systems are serial (although developing computer technologies may allow the practical realization of some parallel processing). On a serial computer parallelism must be simulated—this is not always straightforward.

In *serial computation*, operations are performed sequentially in time whether or not they depend on one another. The implied sequential control mechanism is more closely matched to a (traditional) serial computer than is a parallel mechanism. Sequential algorithms must be stingy with their resources. This fact has had many effects in computer vision. It has led to mechanisms for efficient data access, such as multiple-resolution representations. It has also led some to emphasize cognitive alternatives for low-level visual processing, in the hope that the massive parallel computations performed in biological vision systems could be circumvented. However, this trend is reversing; cheaper computation and more pervasive parallel hardware should increase the commitment of resources to low-level computations. Parallel and serial control mechanisms have both appeared in algorithms in earlier chapters. It seems clear that many low-level operations (correlation, intrinsic image computations) can be implemented with parallel algorithms. High-level operations, such as "planning" (Chapter 13) have inherently serial components. In general, in the low levels of visual processing control is predominately parallel, whereas at the more abstract levels some useful computations are necessarily serial in nature.

10.4.2 Hierarchical and Heterarchical Control

Visual control strategies dictate the flow of information and activity through the representational layers. What triggers processing: a low level input like a color patch on the retina, or a high level expectation (say, expecting to see a red car)? Different emphasis on these extremes is a basic control issue. The two extremes may be characterized as follows.

1. Image data driven. Here the control proceeds from the construction of the generalized image to segmented structures and finally to descriptions. This is also called *bottom-up* control.

2. Internal model driven. Here high-level models in the knowledge base generate expectations or predictions of geometric, segment, or generalized image structure in the input. Image understanding is the verification of these predictions. This is also called *top-down* control.

Top-down and bottom-up control are distinguished not by what they do but rather by the order in which they do it and how much of it they do. Both approaches can utilize all the basic representations—intrinsic images, features, geometric structures, and propositional representations—but the processing within these representations is done in different orders.

The division of control strategies into top-down and bottom-up is a rather simplistic one. There is evidence that attentional mechanisms may be some of the most complicated brain functions that human beings have [Geschwind 1980]. The different representational subsystems in a complex vision system influence each other in sophisticated and intricate ways; whether control flows "up" or "down" is only a broad characterization of local influence in the (loosely ordered) layers of the system.

The term "bottom-up" was originally applied to parsing algorithms for formal languages that worked their way up the parse tree, assembling the input into structures as they did so. "Top-down" parsers, on the other hand, notionally started at the top of the parse tree and worked downward, effectively generating expectations or predictions about the input based on the possibilities allowed by the grammar; the verification of these predictions confirmed a particular parsing.

These two paradigms are still basic in artificial intelligence, and provide powerful analogies and methods for reasoning about and performing many information-processing tasks. The bottom-up paradigm is comparable in spirit with "forward chaining," which derives further consequences from established results. The top-down paradigm is reflected in "backward chaining," which breaks problems up into subproblems to be solved.

These control organizations can be used not only "tactically" to accomplish specific tasks, but they can dictate the whole "strategy" of the vision campaign. We shall discover that in their pure forms the extreme strategies (top-down and bottom-up) appear inadequate to explain or implement vision. More flexible organizations which incorporate both top-down and bottom-up components seem more suited to a broad spectrum of ambitious vision tasks.

Bottom-Up Control

The general outline for bottom-up vision processing is:

1. *PREPROCESS.* Convert raw data into more usable intrinsic forms, to be interpreted by next level. This processing is automatic and domain-independent.
2. *SEGMENT.* Find visually meaningful image objects perhaps corresponding to world objects or their parts. This process is often but not always broken up into (a) the extraction of meaningful visual primitives, such as lines or regions of homogeneous composition (based on their local characteristics); and (b) the agglomeration of local image features into larger segments.

3. *UNDERSTAND.* Relate the image objects to the domain from which the image arose. For instance, identify or classify the objects. As a step in this process, or indeed as the final step in the computer vision program, the image objects and the relations between them may be described.

In pure bottom-up organization each stage yields data for the next. The progression from raw data to interpreted scene may actually proceed in many steps; the different representations at each step allow us to separate the process into the main steps mentioned above.

Bottom-up control is practical if potentially useful "domain-independent" processing is cheap. It is also practical if the input data are accurate and yield reliable and unambiguous information for the higher-level visual processes. For example, the binary images that result from careful illumination engineering and input thresholding can often be processed quite reliably and quickly in a bottom-up mode. If the data are less reliable, bottom-up styles may still work if they make only tolerably few errors on each pass.

Top-Down Control

A bottom-up, hierarchical model of perception is at first glance appealing on neurological and computational grounds, and has influenced much classical philosophical thought and psychological theory. The "classical" explanation of perception has relatively recently been augmented by a more cognition-based one involving (for instance) interaction of knowledge and expectations with the perceptual process in a more top-down manner [Neisser 1967; Bartlett 1932]. A similar evolution of the control of computer vision processing has accounted for the augmentation of the pure "pattern recognition" paradigm with more "cognitive" paradigms. The evidence seems overwhelming that there are vision processes which do not "run bottom-up," and it is one of the major themes of this book that internal models, goals, and cognitive processes must play major roles in computer vision [Gregory 1970; Buckhout 1974; Gombrich 1972]. Of course, there must be a substantial component of biological vision systems which can perform in a noncognitive mode.

There are probably no versions of top-down organization for computer vision that are as pure as the bottom-up ones. The model to keep in mind in top-down perception is that of goal-directed processing. A high-level goal spawns subgoals which are attacked, again perhaps yielding sub-subgoals, and so on, until the goals are simple enough to solve directly. A common top-down technique is "hypothesize-and-verify"; here an internal modeling process makes predictions about the way objects will act and appear. Perception becomes the verifying of predictions or hypotheses that flow from the model, and the updating of the model based on such probes into the perceptual environment [Bolles 1977]. Of course, our goal-driven processes may be interrupted and resources diverted to respond to the interrupt (as when movement in the visual periphery causes us to look toward the moving object). Normally, however, the hypothesis verification paradigm requires relatively little information from the lower levels and in principle it can control the low-level computations.

The desire to circumvent unnecessary low-level processing in computer vision is understandable. Our low-level vision system performs prodigious amounts of information processing in several cascaded parallel layers. With serial computation technology, it is very expensive to duplicate the power of our low-level visual system. Current technological developments are pointing toward making parallel, low-level processing feasible and thus lowering this price. In the past, however, the price has been so heavy that much research has been devoted to avoiding it, often by using domain knowledge to drive a more or less top-down perception paradigm. Thus there are two reasons to use a top-down control mechanism. First, it seems to be something that human beings do and to be of interest in its own right. Second, it seems to offer a chance to accomplish visual tasks without impractical expenditure of resources.

Mixed Top-Down and Bottom-Up Control

In actual computer vision practice, a judicious mixture of data-driven analysis and model-driven prediction often seems to perform better than either style in isolation. This meld of control styles can sometimes be implemented in a complex hierarchy with a simple pass-oriented control structure. An example of mixed organization is provided by a tumor-detection program which locates small nodular tumors in chest radiographs [Ballard 1976]. The data-driven component is needed because it is not known precisely where nodular tumors may be expected in the input radiograph; there is no effective model-driven location-hypothesizing scheme. On the other hand, a distinctly top-down flavor arises from the exploitation of what little is known about lung tumor location (they are found in lungs) and tumor size. The variable-resolution method using pyramids, in which data are examined in increasingly fine detail, also seems top-down. In the example, work done at 1/16 resolution in a consolidated array guides further processing at 1/4 resolution. Only when small windows of the input array are isolated for attention are they considered at full resolution.

The process proceeds in three passes which move from less to greater detail (Fig. 10.19), zooming in on interesting areas of image, and ultimately finding objects of interest (nodules). Two later passes (not shown) "understand" the nodules by classifying them as "ghosts," tumors or nontumors. Within pass II, there is a distinct data-driven (bottom-up) organization, but passes I and III have a model-directed (top-down) philosophy.

This example shows that a relatively simple, pass-oriented control structure may implement a mixture of top-down and bottom-up components which focus attention efficiently and make the computation practical. It also shows a few places where the ordering of steps is not inherently sequential, but could logically proceed in parallel. Two examples are the overlapping of high-pass filtering of pass II with pass I, and parallel exploration of candidate nodule sites in pass III.

Heterarchical Control

The word "heterarchy" seems to be due to McCulloch, who used it to describe the nonhierarchical (i.e., not partially ordered in rank) nature of neural responses implied by their connectivity in the brain. It was used in the early 1970s to characterize a particular style of nonhierarchical, non-pass-structured control

	PREPROCESS	SEGMENT	CONTROL
Pass 0			
(Digitize radiograph)	The digitizer has a hardware attachment which produces the optical density.		
Pass I			
(Find lung boundaries)	In 64 X 56 consolidated array, apply gradient at proper resolution	In 64 X 56 array, find rough lung outline; in 256 X 224 array, refine lung outline	TOP–DOWN
Pass II			
(Find candidate nodule sites and large tumors)	In 256 X 224 array, apply high-pass filter to enhance edges, then inside lung boundaries; apply gradient at proper resolution	In 256 X 224 array use gradient-directed, circular Hough method to find candidate sites; also detect large tumors	BOTTOM–UP
Pass III			
(Find nodule boundaries)	From 1024 X 896 array, extract 64 X 64 window about each candidate nodule site, then in window apply high-pass filter for edge enhancement; then apply gradient at proper resolution	In 64 X 64 full-resolution, pre-processed window, apply dynamic programming technique to find accurate nodule boundaries	TOP–DOWN

Fig. 10.19 A hierarchical tumor-detection algorithm. Technical details of the methods are found elsewhere in this volume. The processing proceeds in passes from top to bottom, and within each pass from left to right. The processing exhibits both top-down and bottom-up characteristics.

organization. Rather than a hierarchical structure (such as the military), one should imagine a community of cooperating and competing experts. They may be organized in their effort by a single executive, by a universal set of rules governing their behavior, or by an a priori system of ranking. If one can think of a task as consisting of many smaller subtasks, each requiring some expertise, and not necessarily performed globally in a fixed order, then the task could be suitable for heterarchical-like control structure.

The idea is to use, at any given time, the expert who can help *most* toward final task solution. The expert may be the most efficient, or reliable, or may give the most information; it is selected because according to some criterion its subtask is the best thing to do at that time. The criteria for selection are wide and varied, and several ideas have been tried. the experts may compute their own relevance, and the decision made on the basis of those individual local evaluations (as in PANDEMONIUM [Selfridge 1959]). They may be assigned a priori immutable rank, so that the highest-ranking expert that is applicable is always run (as in [Shirai 1975; Ambler et al. 1975]). A combination of empirically predetermined and dynamically situation-driven information can be combined to decide which expert applies.

The actual control structure of heterarchical programming can be quite simple; it can be a single iterative loop in which the best action to take is chosen, applied, and interpreted (Fig. 10.20).

10.4.3 Belief Maintenance and Goal Achievement

Belief maintenance and goal achievement are high-level processes that imply differing control styles. The former is concerned with maintaining a current state, the latter with a set of future states. Belief maintenance is an ongoing activity which can ensure that perceptions fit together in a coherent way. Goal achievement is the integration of vision into goal-directed activities such as searching for objects and navigation. There may be "unconscious" use of goal-seeking techniques (e.g., eye-movement control).

Belief Maintenance

An organism is presented with a rich visual input to interpret. Typically, it all makes sense: chairs and tables are supported by floors, objects have expected shapes and colors, objects appear to flow past as the organism moves, nearer objects obscure farther ones, and so on. However, every now and then something

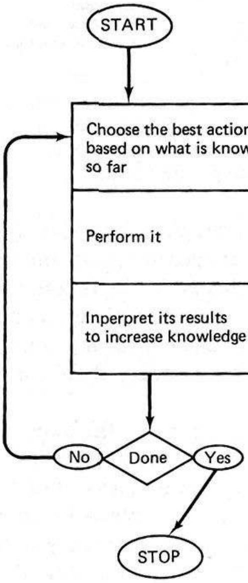


Fig. 10.20 A main executive control loop for heterarchical vision.

enters the visual field that does not meet expectations. An unfamiliar object in a familiar environment or a sudden movement in the visual periphery can be "surprises" that do not fit in with our existing beliefs and thus have to be reckoned with.

It is sometimes impossible to ignore movements in our visual periphery, but if we are preoccupied it is easily possible to stay unconscious of small changes in our environment. How is it possible to notice some things and not others? The belief maintenance mechanism seems to be resource-limited. A certain amount of "computing resource" is allocated for the job. With this resource, only a limited amount of checking can be done. Checks to be made are ranked (somehow—responses to events in the periphery are like reflexes, or high-priority hard-wired interrupts) and those that cannot be done within the resource limit are omitted. Changes in our beliefs are often initiated in a *bottom-up* way, through unexpected inputs.

A second characteristic of belief maintenance is the almost total absence of sequential, simulation-based or "symbolic" planning or problem-solving activity. Our beliefs are "in the present"; manipulation of hypothetical worlds is not belief maintenance. "Truth maintenance" schemes have been discussed in various contexts [Doyle 1979; Stallman and Sussman 1977].

We conjecture that constraint-satisfaction (relaxation) mechanisms (Chapters 3, 7, and 12) are computationally suited to maintaining belief structures. They can operate in parallel, they seek to minimize inconsistency, they can tolerate "noise" in either input or axioms. Relaxation techniques are usually applied to low-level visual input where locally noisy parameters are combined into globally consistent intrinsic images. Chapter 12 is concerned with inference, in which constraint relaxation is applied to higher-level entities.

Characteristics of Goal Achievement

Goal achievement involves two related activities: planning and acting. Planning is a simulation of the world designed to generate a plan. A *plan* is a sequence of actions that, if carried out, should achieve a goal. *Actions* are the primitives that can modify the world. The motivation for planning is survival. By being able to simulate the effects of various actions, a human being is able to avoid dangerous situations. In an analogous fashion, planning can help machines with vision. For example, a Mars rover can plan its route so as to avoid steep inclines where it might topple over. The incline measurement is made by processing visual input. Since planning involves a sequence of actions, each of which if carried out could potentially change the world, and since planning does not involve actually making those changes, the difficult task of the planner is to keep track of all the different world states that could result from different action sequences.

Vision can clearly serve as an important information-gathering step in planning actions. Can planning techniques be of use directly to the vision process? Clearly so in "skilled vision," such as photointerpretation. Also, planning is a useful computational mechanism that need not be accompanied by conscious, cognitive behavior.

These inductive conclusions leading to the formation of our sense perceptions certainly do lack the purifying and scrutinizing work of conscious thinking. Nevertheless, in my opinion, by their particular nature they may be classed as *conclusions*, inductive conclusions unconsciously formed. [Helmholtz 1925]

The character of computations in goal achievement is related to the inference mechanisms studied in Chapter 11, only planning is distinguished by being dynamic through time. Inference (Chapter 12) is concerned with the knowledge base and deducing relations that logically follow from it. The primitives are *propositions*. In planning (Chapter 13) the primitives are *actions*, which are inherently more complex than propositions. Also, planning need not be a purely deductive mechanism; instead it can be integrated with visual "acting", or the interpretation of visual input. Often, a long deductive sequence may be obviated by using direct visual inspection. This raises a crucial point: Given the existence of plans, how does one choose between them? The solution is to have a method of scoring plans based on some measure of their effectiveness.

EXERCISES

10.1 (a) Diagram some networks for a simple dial telephone, at various levels of detail and with various complexities of relations.

(b) Now include in your network dial and pushbutton types.

(c) Embed the telephone frame into an office frame, describing where the telephone should be found.

10.2 Is a LISP vision program an analogical or propositional representation of knowledge?

10.3 Write a semantic net for the concept "leg," and use it to model human beings, tables, and spiders. Represent the fact "all tables have four legs." Can your "leg" model be shared between tables and spiders? Shared within spiders?

REFERENCES

AMBLER, A. P., H. G. BARROW, C. M. BROWN, R. M. BURSTALL, and R. J. POPPLESTONE. "A versatile system for computer controlled assembly." *Artificial Intelligence* 6, 2, 1975, 129–156.

ANDERSON, J. R. and G. H. BOWER. *Human Associative Memory.* New York: V. H. Winston & Sons, 1973.

BALLARD, D. H. *Hierarchic Recognition of Tumors in Chest Radiographs.* Basel: Birkhauser-Verlag (ISR-16), 1976.

BALLARD, D. H. "Model-directed detection of ribs in chest radiographs." TR11, Computer Science Dept., Univ. Rochester, March 1978.

BALLARD, D. H., U. SHANI, and R. B. SCHUDY. "Anatomical models for medical images." *Proc.*, 3rd COMPSAC, November 1979, 565-570.

BARROW, H. G. and J. M. TENENBAUM. "Computational vision." *Proc. IEEE* 69, 5, May 1981, 572-595.

BARTLETT, F. C. *Remembering: A Study in Experimental and Social Psychology.* Cambridge: Cambridge University Press, 1932.

BOBROW, D. G. and T. WINOGRAD. "An overview of KRL-0, a knowledge representation language." *Cognitive Science 1*, 1, 1977, 3–46.

BOBROW, D. G. and T. WINOGRAD. "KRL: another perspective." *Cognitive Science 3*, 1, 1979, 29–42.

BOLLES, R. C. "Verification vision for programmable assembly." *Proc.*, 5th IJCAI, August 1977, 569–575.

BRACHMAN, R. J. "What's in a concept? Structural foundations for semantic networks." Report 3433, Bolt, Beranek and Newman, October 1976.

BRACHMAN, R. J. "On the epistemological status of semantic networks." *In Associative Networks: Representation and Use of Knowledge by Computers*, N.V. Findler (Ed.). New York: Academic Press, 1979, 3–50.

BROOKS, R. A. and T. O. BINFORD. "Representing and reasoning about specified scenes." *Proc.*, DARPA IU Workshop, April 1980, 95–103.

BUCKHOUT, R. "Eyewitness testimony." *Scientific American*, December 1974, 23–31.

DOYLE, J. "A truth maintenance system." *Artificial Intelligence 12*, 3, 1979.

FAHLMAN, S. E. "A planning system for robot construction tasks." *Artificial Intelligence 5*, 1, 1974, 1–49.

FINDLER, N. V. (Ed.). *Associative Networks: Representation and Use of Knowledge by Computers.* New York: Academic Press, 1979.

FODOR, J. D., J. A. FODOR, and M. F. GARRETT. "The psychological unreality of semantic representations." *Linguistic Inquiry 4*, 1975, 515–531.

FREUDER, E. C. "A computer system for visual recognition using active knowledge." Ph.D. dissertation, MIT, 1975.

FUNT, B. V. "WHISPER: a problem-solving system utilizing diagrams." *Proc.*, 5th IJCAI, August 1977, 459–464.

GARVEY, J. D. "Perceptual strategies for purposive vision." Technical Note 117, AI Center, SRI International, 1976.

GELERNTER, H. "Realization of a geometry-theorem proving machine." In *Computers and Thought*, E. Feigenbaum and J. Feldman (Eds.). New York: McGraw-Hill, 1963.

GESCHWIND, N. "Neurological knowledge and complex behaviors." *Cognitive Science 4*, 2, April 1980, 185–193.

GOMBRICH, E. H. *Art and Illusion.* Princeton, NJ: Princeton University Press, 1972.

GREGORY, R. L. *The Intelligent Eye.* New York: McGraw-Hill, 1970.

HAYES, Patrick J. "The logic of frames." *In The Frame Reader.* Berlin: DeGruyter, 1980.

HAYES*, Philip J. "Some association-based techniques for lexical disambiguation by machine." Ph.D. dissertation, École polytechnique fédérale de Lausanne, 1977; also TR25, Computer Science Dept., Univ. Rochester, June 1977.

HELMHOLTZ, H. von. *Treatise on Physiological Optics* (translated by J. P. T. Sauthall). New York: Dover Publications, 1925.

HENDRIX, G. G. "Expanding the utility of semantic networks through partitions." *Proc.*, 4th IJCAI, September 1975, 115–121.

HENDRIX, G. G. "Encoding knowledge in partitioned networks." In *Associative Networks: Representation and Use of Knowledge by Computers*, N.V. Findler (Ed.). New York: Academic Press, 1979, 51–92.

HEWITT, C. "Description and theoretical analysis (using schemata) of PLANNER" (Ph.D. dissertation). AI-TR-258, AI Lab, MIT, 1972.

HINTON, G. E. "Relaxation and its role in vision." Ph.D. dissertation, Univ. Edinburgh, December 1977.

JOHNSON-LAIRD, P. N. "Mental models in cognitive science." *Cognitive Science 4*, 1, January–March 1980, 71–115.

KOSSLYN, S. M. and J. R. POMERANTZ. "Imagery, propositions and the form of internal representations." *Cognitive Psychology 9*, 1977, 52–76.

KOSSLYN, S. M. and S. P. SCHWARTZ. "A simulation of visual imagery." *Cognitive Science 1*, 3, July 1977, 265–295.

KOSSLYN, S. M. and S. P. SCHWARTZ. "Visual images as spatial representations in active memory." In *CVS*, 1978.

LEHNERT, W. and Y. WILKS. "A critical perspective on KRL." *Cognitive Science 3*, 1, 1979, 1–28.

LOZANO-PEREZ, T. and M. A. WESLEY. "An algorithm for planning collision-free paths among polyhedral obstacles." *Comm. ACM 22*, 10, October 1979, 560–570.

MCDERMOTT, D. "Artificial intelligence meets natural stupidity." *SIGART Newsletter 57*, April 1976, 4–9.

MINSKY, M. L. "A framework for representing knowledge." In *PCV*, 1975.

MOORE J. and A. NEWELL. "How can MERLIN understand?" In *Knowledge and Cognition*, L. Gregg (Ed.). Hillsdale, NJ: Lawrence Erlbaum Assoc., 1973.

MOORE, R. C. "Reasoning about knowledge and action." Techical Note 191, AI Center, SRI International, 1979.

NEISSER, U. *Cognitive Psychology*. New York: Appleton-Century-Crofts, 1967.

NEVATIA, R. and K.E. PRICE. "Locating structures in aerial images." USCIPI Report 800, Image Processing Institute, Univ. Southern California, March 1978, 41–58.

NILSSON, N. J. *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw-Hill, 1971.

NILSSON, N. J. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.

PALMER, S. E. "Visual perception and world knowledge: notes on a model of sensory-cognitive interaction." In *Explorations in Cognition*, D.A. Norman, D.E. Rumelhart, and the LNR Research Group (Eds.). San Francisco: W.H. Freeman, 1975.

PYLYSHYN, Z. W. "What the mind's eye tells the mind's brain; a critique of mental imagery." *Psychological Bulletin 80*, 1973, 1–24.

QUILLIAN, M. R. "Semantic memory." In *Semantic Information Processing*, M. Minsky (Ed.). Cambridge, MA: MIT Press, 1968.

ROBERTS, R. B. and I. P. GOLDSTEIN. "The FRL primer." AI Memo 408, AI Lab, MIT, 1977.

RUMELHART, D. E., P. H. LINDSAY, and D. A. NORMAN. "A process model for long-term memory." In *Organization of Memory*, E. Tulving and J. Donaldson (Eds.). New York: Academic Press, 1972.

RUSSELL, D. M. "Where do I look now?" *Proc.*, PRIP, August 1979, 175–183.

SCHANK, R. C. *Conceptual Information Processing*. Amsterdam: North-Holland, 1975.

SCHANK, R. C. and R. P. ABELSON. *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Lawrence Erlbaum Assoc., 1977.

SCHUBERT, L. K. "Extending the expressive power of semantic networks." *Artificial Intelligence 7*, 2, 1976, 163–198.

SELFRIDGE, O. "Pandemonium, a paradigm for learning." In *Proc.*, Symp. on the Mechanisation of Thought Processes, National Physical Laboratory, Teddington, England, 1959.

SHEPARD, R. N. "The mental image." *American Psychologist 33*, 1978, 125–137.

SHIRAI, Y. "Analyzing intensity arrays using knowledge about scenes." In *PCV*, 1975.

SLOMAN, A. "Interactions between philosophy and artificial intelligence: the role of intuition and non-logical reasoning in intelligence." *Artificial Intelligence 2*, 3/4, 1971, 209–225.

STALLMAN, R. M. and G. J. SUSSMAN. "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis." *Artificial Intelligence 9*, 2, 1977, 135–196.

STEFIK, M. "An examination of a frame-structured representation system." *Proc.*, 6th IJCAI, August 1979, 845–852.

SUSSMAN, G. J. and D. MCDERMOTT. "Why conniving is better than planning." AI Memo 255A, AI Lab, MIT, 1972.

WALTZ, D. and L. BOGGESS. "Visual analog representations for natural language understanding." *Proc.*, 6th IJCAI, August 1979, 226–234.

WINOGRAD, T. "Extended inference modes in reasoning by computer systems." *Proc.*, Conf. on Inductive Logic, Oxford Univ., August 1978.

WINOGRAD, T. "Frame representations and the declarative/procedural controversy." In *Representation and Understanding*, D. G. Bobrow and A. M. Collins (Eds.). New York: Academic Press, 1975, 185–210.

WINSTON, P. H. "Learning structural descriptions from examples." In *PCV*, 1975.

WINSTON, P. H. *Artificial Intelligence.* Reading, MA: Addison-Wesley, 1977.

WOODS, W. A. "What's in a link? Foundations for semantic networks." In *Representation and Understanding*, D. G. Bobrow and A. M. Collins (Eds.). New York: Academic Press, 1975.

Matching 11

11.1 ASPECTS OF MATCHING

11.1.1 Interpretation: Construction, Matching, and Labeling

Figure 10.1 shows a vision system organization in which there are several representations for visual entities. A complex vision system will at any time have several coexisting representations for visual inputs and other knowledge. Perception is the process of integrating the visual input with the preexisting representations, for whatever purpose. Recognition, belief maintenance, goalseeking, or building complex descriptions—all involve forming or finding relations between internal representations. These correspondences *match* ("model," "re-represent," "abstract," "label") entities at one level with those at another level.

Ultimately, matching "establishes an interpretation" of input data, where an interpretation is the correspondence between models represented in a computer and the external world of phenomena and objects. To do this, matching associates different representations, hence establishing a connection between their interpretations in the world. Figure 11.1 illustrates this point. Matching associates TOKNODE, a token for a linear geometric structure derived from image segmentation efforts with a model token NODE101 for a particular road. The token TOKNODE has the interpretation of an image entity; NODE101 has the interpretation of a particular road.

One way to relate representations is to *construct* one from the other. An example is the construction of an intrinsic image from raw visual input. Bottom-up construction in a complex visual system is for reliably useful, domain-independent, goal-independent processing steps. Such steps rely only on "compiled-in" ("hard-wired," "innate") knowledge supplied by the designer of the system. Matching becomes more important as the needed processing becomes more diverse and idiosyncratic to an individual's experience, goals, and

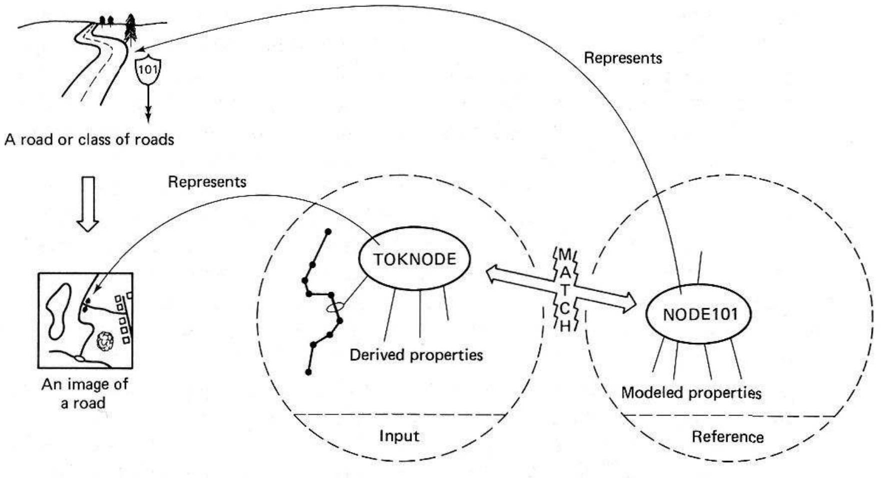


Fig. 11.1 Matching and interpretation.

knowledge. Thus as processing moves from "early" to "late," control shifts from bottom-up toward top-down, and existing knowledge begins to dominate perception.

This chapter deals with some aspects of matching, in which two already existing representations are put into correspondence. When the two representations are similar (both are images or relational structures, say), "matching" can be used in its familiar sense. When the representations are different (one image and one geometric structure, say), we use "matching" in an extended sense; perhaps "fitting" would be better. This second sort of matching usually has a top-down or expectation-driven flavor; a representation is being related to a preexisting one.

As a final extension to the meaning of matching, matching might include the process of checking a structure with a set of rules describing structural legality, consistency, or likelihood. In this sense a scene can be matched against rules to see if it is nonsense or to assign an interpretation. One such interpretation process (called *labeling*) assigns consistent or optimally likely interpretations (labels) at one level to entities of another level. Labeling is like matching a given structure with a possibly infinite set of acceptable structures to find the best fit. However, we (fairly arbitrarily) treat labeling in Chapter 12 as extended inference rather than here as extended matching.

11.1.2 Matching Iconic, Geometric, and Relational Structures

Chapter 3 presented various correlation techniques for matching *iconic* (image-like) structures with each other. The bulk of this chapter, starting in Section 11.2, deals with matching *relational* (semantic net) structures. Another important sort of matching between two dissimilar representations fits data to parameterized models (usually geometric). This kind of matching is an important part of computer vision. A typical example is shown in Fig. 11.2. A preexisting representation (here a straight line) is to be used to interpret a set of input data. The line that best "explains" the data is (by definition) the line of "best fit." Notice that the decision to use a line (rather than a cubic, or a piecewise linear template) is made at a higher level. Given the model, the fitting or matching means determining the *parameters* of the model that tailor it into a useful abstraction of the data.

Sometimes there is no parameterized mathematical model to fit, but rather a given geometric structure, such as a piecewise linear curve representing a shoreline in a map which is to be matched to a piece of shoreline in an image, or to another piecewise linear structure derived from such a shoreline. These geometric matching problems are not traditional mathematical applications, but they are similar in that the best match is defined as the one minimizing a measure of disagreement.

Often, the computational solutions to such geometric matching problems exhibit considerable ingenuity. For example, the shore-matching example above may proceed by finding that position for the segment of shore to be matched that minimizes some function (perhaps the square) of a distance metric (perhaps Euclidean) between input points on the iconic image shoreline and the nearest point on the reference geometric map shoreline. To compute the smallest distance between an arbitrary point and a piecewise linear point set is not a trivial task, and this calculation may have to be performed often to find the best match. The computation may be reduced to a simple table lookup by precomputing the metric in a "chamfer array," that contains the metric of disagreement for any point around the geometric reference shoreline [Barrow et al. 1978]. The array may be computed efficiently by symmetric axis transform techniques (Chapter 8) that "grow" the linear structure outward in contours of equal disagreement (distance) until a value has been computed for each point of the chamfer array.

Parameter optimization techniques can relate geometrical structures to lower-level representations and to each other through the use of a merit function measuring how well the relations match. The models are described by a vector of parameters $\mathbf{a} = (a_1, \ldots, a_n)$. The merit function M must rate each set of those parameters in terms of a real number. For example, M could be a function of both a, the parameters, and $f(x)$, the image. The problem is to find a such that

$$M(\mathbf{a}, f(\mathbf{x}))$$

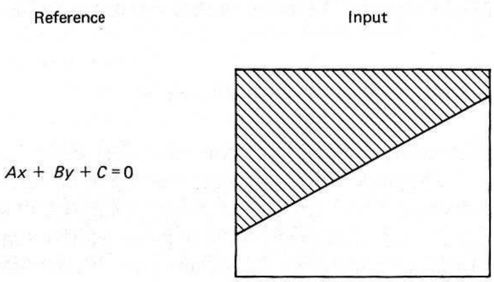


Fig. 11.2 Matching or fitting a straight line model to data.

is maximized. Note that if **a** were some form of template function rather than a vector of parameters, the problem statement would encompass the iconic correlation techniques just covered. There is a vast literature on optimization techniques and we cannot do more than provide a cursory discussion of a few cases with examples.

Formally, the different techniques have to do with the form of the merit function M. A fundamental result from calculus is that if M is sufficiently well behaved (i.e., has continuous derivatives), then a condition for a local maximum (or minimum) is that

$$M_{a_j} = \frac{\partial M}{\partial a_j} = 0 \qquad \text{for } j = 1,\ldots,n \tag{11.1}$$

This condition can be exploited in many different ways.

- Sometimes Eqs. (11.1) are sufficiently simple so that the a can be determined analytically, as in the least squares fitting, described in Appendix 1.
- An approximate solution a^0 can be iteratively adjusted by moving in the gradient direction or direction of maximum improvement:

  $$a_j^k = a_j^{k-1} + cM_{a_j} \tag{11.2}$$

  where c is a constant. This is the most elementary of several kinds of *gradient (hill-climbing) techniques.* Here the gradient is defined with respect to M and does not mean edge strength.
- If the partial derivatives are expensive to calculate, the coefficients can be perturbed (either randomly or in a structured way) and the perturbations kept if they improve M:

  (1) $\mathbf{a}' := \mathbf{a} + \Delta\mathbf{a}$

  (2) $\mathbf{a} := \mathbf{a}'$ if $M(\mathbf{a}') > M(\mathbf{a})$

A program to fit three-dimensional image data with shapes described by spherical harmonics used these techniques [Schudy and Ballard 1978]. The details of the spherical harmonics shape representation appear in Chapter 9. The fitting proceeded by the third method above. A nominal expected shape was matched to boundaries in image data. If a subsequent perturbation in one of its parameters results in an improvement in fit it was kept; otherwise, a different perturbation was made. Figure 11.3 shows this fitting process for a cross section of the shape.

Though parameter optimization is an important aspect of matching, we shall not pursue it further here in view of the extensive literature on the subject.

11.2 GRAPH-THEORETIC ALGORITHMS

The remainder of this chapter deals with methods of matching relational structures. Chapter 10 showed how to represent a relational structure containing n-ary relations as a graph with labeled arcs. Recall that the labels can have values from a