Chapter 14

3D Models and Matching

Models of 3D objects are used heavily both in computer vision and in computer graphics. In graphics, the object must be represented in a structure suitable for rendering and display. The most common such structure is the 3D mesh, a collection of polygons consisting of 3D points and the edges that join them. Graphics hardware usually supports this representation. For smoother and/or simpler surfaces, other graphics representations include quadric surfaces, B-spline surfaces, and subdivision surfaces. In addition to 3D shape information, graphics representations can contain color/texture information which is then "texture-mapped" onto the rendered object by the graphics hardware. Figure 14.1 shows a rough 3D mesh of a toy dog and a texture-mapped rendered image from the same viewpoint.

In computer vision, the object representation must be suitable for use in object recognition, which means that there must be some potential correspondence between the representation and the features that can be extracted from an image. However, there are several different types of images commonly used in 3D object recognition, in particular: gray-scale images, color images, and range images. Furthermore, it is now common to have either gray-scale or color images registered to range data, providing recognition algorithms with a richer set of features. Most 3D object algorithms are not general enough to handle such a variety of features, but instead were designed for a particular representation. Thus it

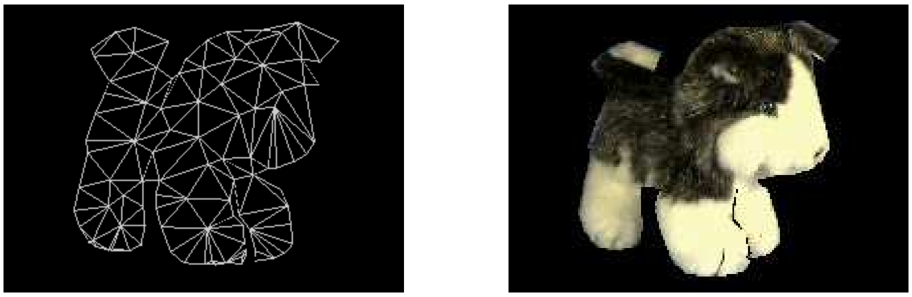


Figure 14.1: 3D mesh of a toy dog and texture-mapped rendered image.

is important to look at the common representations before discussing 3D object recognition. In general categories, there are geometric representations in terms of points, lines, and surfaces; symbolic representations in terms of primitive components and their spatial relationships; and functional representations in terms of functional parts and their functional relationships. We will begin with a survey of the most common methods for representing 3D objects and then proceed to the representations required by the most common types of object recognition algorithms.

14.1 Survey of Common Representation Methods

Computer vision began with the work of Roberts in 1965 on recognition of polyhedral objects, using simple wire-frame models and matching to straight line segments extracted from images. Line-segment-based models have remained popular even today, but there are also a number of alternatives that attempt to more closely represent the data from objects that can have curved and even free-form surfaces. In this section, we will look at mesh models, surface-edge-vertex models, voxel and octree models, generalized-cylinder models, superquadric models, and deformable models. We will also look at the distinction between true 3D models and characteristic-view models that represent a 3D object by a set of 2D views.

14.1.1 3D Mesh Models

A 3D mesh is a very simple geometric representation that describes an object by a set of vertices and edges that together form polygons in 3D-space. An arbitrary mesh may have arbitrary polygons. A *regular mesh* is composed of polygons all of one type. One commonly used mesh is a *triangular mesh*, which is composed entirely of triangles; the mesh shown in Figure 14.1 is a triangular mesh. Meshes can represent an object at various different levels of resolution, from a coarse estimate of the object to a very fine level of detail. Figure 14.2 shows three different meshes representing different levels of resolution of the dog. They can be used both for graphics rendering or for object recognition via range data. When used for recognition, feature extraction operators must be defined to extract features from the range data that can be used in matching. Such features will be discussed later in this chapter.

14.1.2 Surface-Edge-Vertex Models

Since many of the early 3D vision systems worked with polygonal objects, edges have been the main local feature used for recognition or pose estimation. A three–dimensional object model that consists of only the edges and vertices of the object is called a *wire-frame* model. The wire-frame representation assumes that the surfaces of the object are planar and that the object has only straight edges.

A useful generalization of the wire-frame model that has been heavily used in computer vision is the *surface–edge–vertex* representation. The representation is a data structure containing the vertices of the object, the surfaces of the object, the edge segments of the object, and, usually, the topological relationships that specify the surfaces on either side of an edge and the vertices on either end of an edge segment. When the object is polygonal, the surfaces are planar and the edge segments are straight line segments. However, the

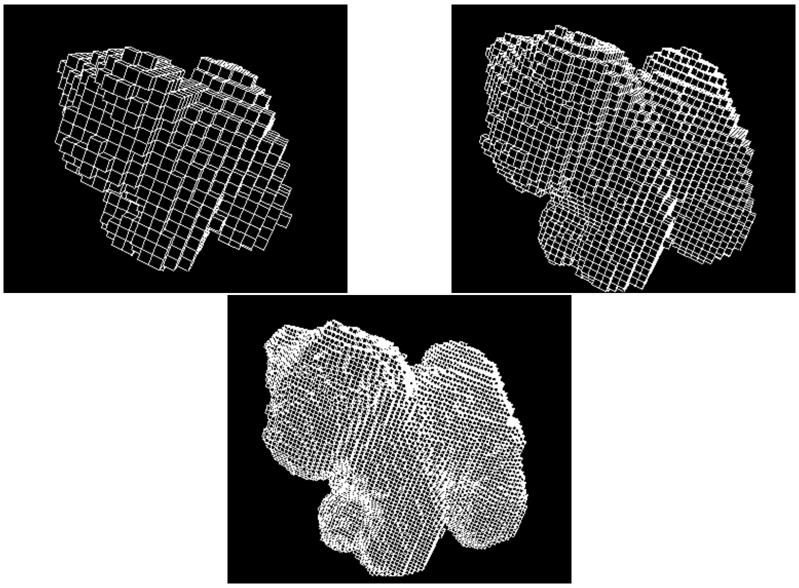


Figure 14.2: Three meshes of the dog at different resolutions.

model generalizes to include curved edge segments and/or curved surfaces.

Figure 14.3 illustrates a sample surface-edge-vertex data structure used for representing a database of object models in a 3D object recognition system. The data structure is hierarchical, beginning with the world at the top level and continuing down to the surfaces and arcs at the lowest level. In Figure 14.3 the boxes with fields labeled [name, type, <entity>, transf] indicate the elements of a set of class <entity>. Each element of the set has a name, a type, a pointer to an <entity>, and a 3D transformation that is applied to the <entity> to obtain a potentially rotated and translated instance. For example, the world has a set called *objects*. In that set are named instances of various 3D object models. Any given object model is defined in its own coordinate system. The transformation allows each instance to be independently positioned in the world.

The object models each have three sets: their edges, their vertices, and their faces. A vertex has an associated 3D point and a set of edges that meet at that point. An edge has a start point, an end point, a face to its left, a face to its right, and an arc that defines its form, if it is not a straight line. A face has a surface that defines its shape and a set of boundaries including its outer boundaries and hole boundaries. A boundary has an associated face and a set of edges. The lowest level entities–arcs, surfaces, and points–are not defined here. Representations for surfaces and arcs will depend on the application and on the accuracy and smoothness required. They might be represented by equations or further broken down into surface patches and arc segments. Points are merely vectors of (x,y,z) coordinates.

Figure 14.4 shows a simple 3D object that can be represented in this manner. To

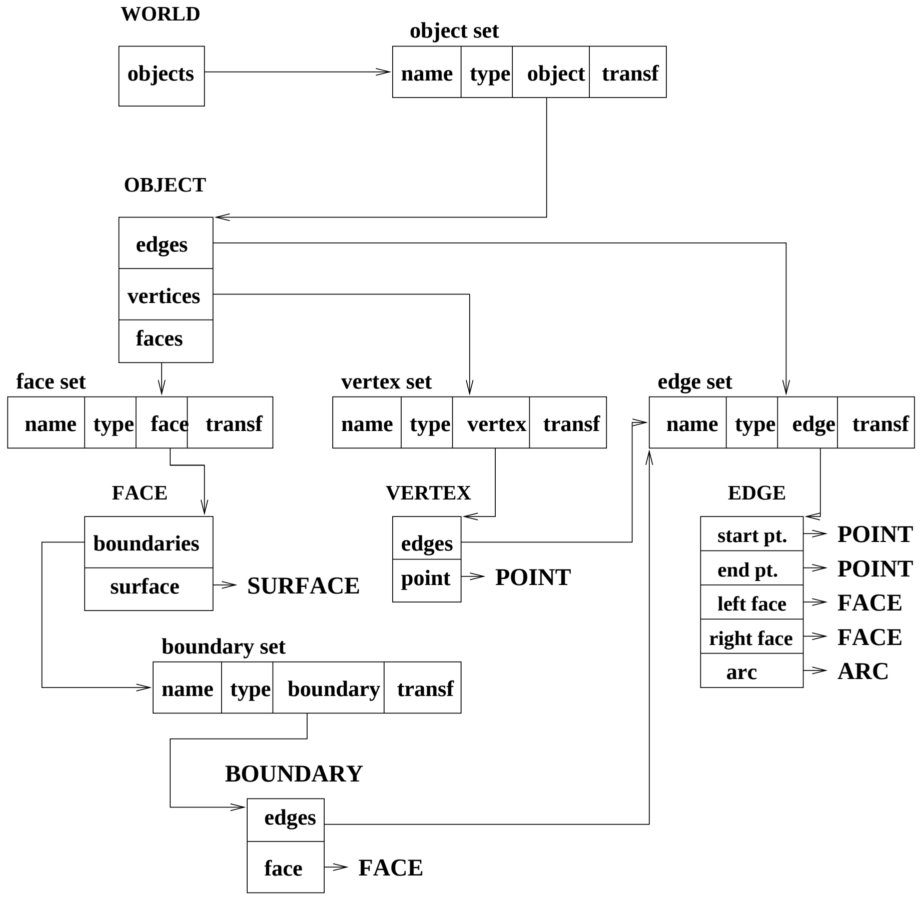


Figure 14.3: Surface-edge-vertex data structure.

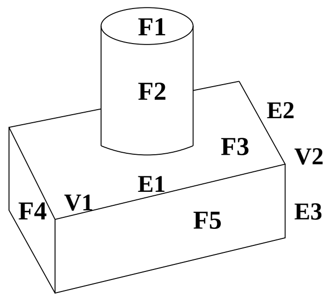


Figure 14.4: Sample 3D object with planar and cylindrical surfaces.

simplify the illustration, only a few visible surfaces and edges are discussed. The visible surfaces are F1, F2, F3, F4, and F5. F1, F3, F4, and F5 are planar surfaces, while F2 is a cylindrical surface. F1 is bounded by a single boundary composed of a single edge that can be represented by a circular arc. F2 is bounded by two such boundaries. F3 is bounded by an outer boundary composed of four straight edges and a hole boundary composed of a single circular arc. F4, and F5 are each bounded by a single boundary composed of four straight edges. Edge E1 separates faces F3 and F5. If we take vertex V1 to be its start point and V2 to be its end point, then F3 is its left face and F5 is its right face. Vertex V2 has three associated edges E1, E2, and E3.

Exercise 1 Surface-edge-vertex structure

Using the representation of Figure 14.3, create a model of the entire object shown in Figure 14.4, naming each face, edge, and vertex in the full 3D object, and using these names in the structure.

14.1.3 Generalized-Cylinder Models

A *generalized cylinder* is a volumetric primitive defined by a space curve axis and a cross section function at each point of the axis. The cross section is swept along the axis creating a solid of revolution. For example, a common circular cylinder is a generalized cylinder whose axis is a straight line segment and whole cross section is a circle of constant radius. A cone is a generalized cylinder whose axis is a straight line segment and whose cross section is a circle whose radius starts out zero at one end point of the axis and grows to its maximum at the other end point. A rectangular solid is a generalized cylinder whose axis is a straight line segment and whose cross section is a constant rectangle. A torus is a generalized cylinder whose axis is a circle and whose cross section is a constant circle.

A generalized cylinder model of an object includes descriptions of the generalized cylinders and the spatial relationships among them, plus global properties of the object. The cylinders can be described by length of axis, average cross–section width, ratio of the two, and cone angle. Connectivity is the most common spatial relationship. In addition to end-point connectivity, cylinders may be connected so that the end points of one connect to an interior point of another. In this case, the parameters of the connection, such as the position at which the cylinders touch, the inclination angle, and the girdle angle describing the rotation of one about the other may be used to describe the connection. Global properties of an object may include number of pieces (cylinders), number of elongated pieces, and symmetry of the connections. Hierarchical generalized cylinder models, in which different levels of detail are given at different levels of the hierarchy, are also possible. For example, a person might be modeled very roughly as a stick figure (as shown in Figure 14.5) consisting of cylinders for the head, torso, arms, and legs. At the next level of the hierarchy, the torso might be divided into a neck and lower torso, the arms into three cylinders for upper arm, lower arm, and hand, and the legs similarly. At the next level, the hands might be broken into a main piece and five fingers, and one level deeper, the fingers might be broken into three pieces and the thumb into two.

A three–dimensional generalized cylinder can project to two different kinds of two–dimensional regions on an image: ribbons and ellipses. A *ribbon* is the projection of the

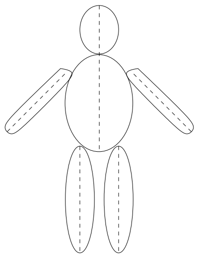


Figure 14.5: Rough generalized cylinder model of a person. The dotted lines represent the axes of the cylinders.

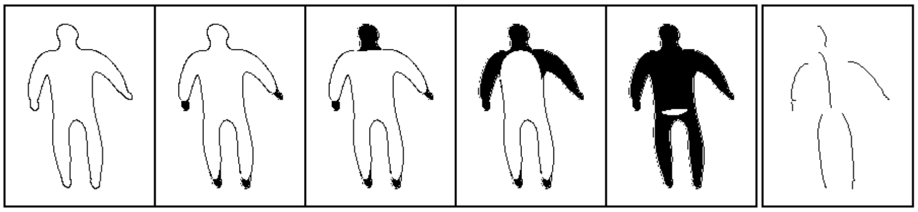


Figure 14.6: The process of constructing a generalized cylinder approximation from a 2D shape. (Example courtesy of Gerard Medioni.)

long portion of the cylinder, while an *ellipse* is the projection of the cross section. Of course, the cross section is not always circular, so its projection is not always elliptical, and some generalized cylinders are completely symmetric, so they have no longer or shorter parts. For those that do, algorithms have been developed to find the ribbons in images of the modelled objects. These algorithms generally look for long regions that can support the notion of an axis. Figure 14.6 shows the process of determining potential axes of generalized cylinders from a 2D shape.

Figure 14.7 shows steps in creating a detailed model of a particular human body for the purpose of making well-fitting clothing. A special sensing environment combines input from twelve cameras. Six cameras view the human at equal intervals of a 2m cylindrical room: there is a low set and high set so that a 2m tall person can be viewed. As shown in Figure 14.7, silhouettes from six cameras are used to fit elliptical cross sections to obtain a cylindrical model. A light grid is also used so that triangulation can be used to compute 3D surface points in addition to points on the silhouettes. Concavities are developed using the structured light data, and ultimately a detailed mesh of triangles is computed.

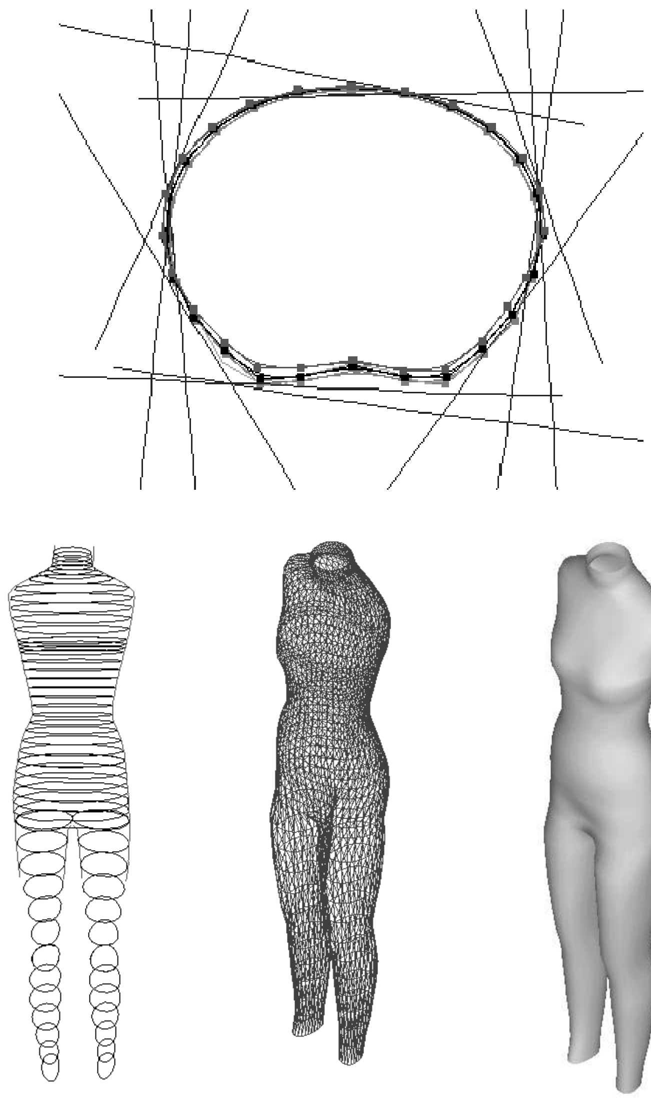


Figure 14.7: Steps in making a model of a human body for fitting clothing. (Top) Three cross section curves along with cross section silhouettes from six cameras viewing the body (the straight lines project the silhouette toward the cameras). Structured light features allow 3D points to be computed in concavities. (Bottom) Generalized cylinder model created by fitting elliptical cross sections to the six silhouettes, resulting triangular mesh, and shaded image. (Courtesy of Helen Shen and colleagues at the Dept. of Computer Science, Hong Kong University of Science and Technology: project supported by grant AF/183/97 from the Industry and Technology Development Council of Hong Kong, SAR of China in 1997.)

Exercise 2 Generalized cylinder models

Construct a generalized cylinder model of an airplane. The airplane should have a fuselage, wings, and a tail. The wings should each have an attached motor. Try to describe the connectivity relationships between pairs of generalized cylinders.

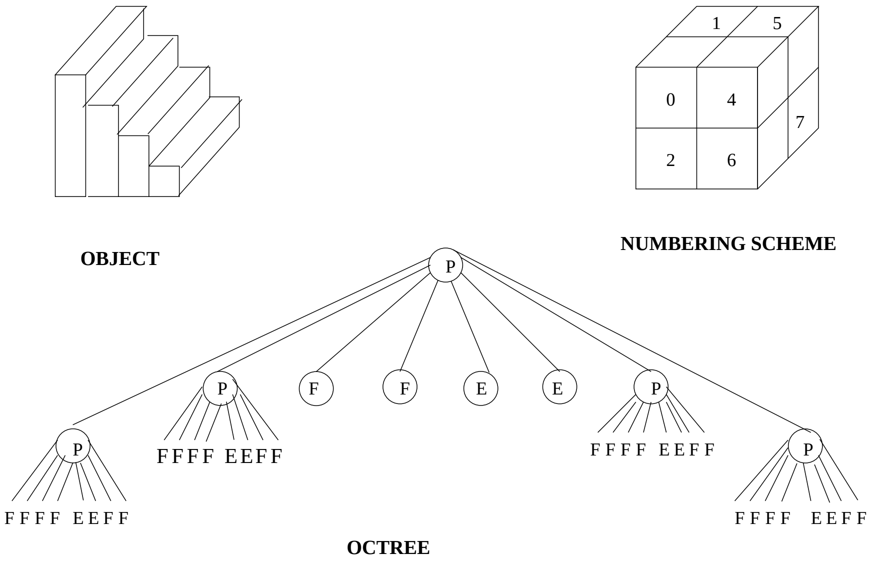


Figure 14.8: A simple three–dimensional object and its octree encoding.

14.1.4 Octrees

An *octree* is a hierarchical 8–ary tree structure. Each node in the tree corresponds to a cubic region of the universe. The *label* of a node is either *full*, if the cube is completely enclosed by the three–dimensional object, *empty* if the cube contains no part of the object, or *partial*, if the cube partly intersects the object. A node with label *full* or *empty* has no children. A node with label *partial* has eight children representing the partition of the cube into octants.

A three–dimensional object can be represented by a $2^n \times 2^n \times 2^n$ three–dimensional array for some integer n. The elements of the array are called *voxels* and have a value of 1 (full) or 0 (empty), indicating the presence or absence of the object. The octree encoding of the object is equivalent to the three–dimensional array representation, but will generally require much less space. Figure 14.8 gives a simple example of an object and its octree encoding, using the octant numbering scheme of Jackins and Tanimoto.

Exercise 3 Octrees

Figure 14.11 shows two views of a simple chair. Construct an octree model of the chair. Assume that the seat and back are both 4 voxels by 4 voxels by 1 voxel and that each of the legs is 3 voxels by 1 voxel by 1 voxel.

14.1.5 Superquadrics

Superquadrics are models originally developed for computer graphics and proposed for use in computer vision by Pentland. Superquadrics can intuitively be thought of as lumps of clay that can be deformed and glued together into object models. Mathematically superquadrics form a parameterized family of shapes. A superquadric surface is defined by a vector S whose x, y, and z components are specified as functions of the angles η and ω via the equation

$$S(\eta,\omega) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 \cos^{\epsilon_1}(\eta)\cos^{\epsilon_2}(\omega) \\ a_2 \cos^{\epsilon_1}(\eta)\sin^{\epsilon_2}(\omega) \\ a_3 \sin^{\epsilon_1}(\eta) \end{bmatrix} \tag{14.1}$$

for $-\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2}$ and $-\pi \leq \omega < \pi$. The parameters a_1, a_2, and a_3 specify the size of the superquadric in the x, y and z directions, respectively. The parameters ϵ_1 and ϵ_2 represent the squareness in the latitude and longitude planes.

Superquadrics can model a set of useful building blocks such as spheres, ellipsoids, cylinders, parallelepipeds, and in-between shapes. When ϵ_1 and ϵ_2 are both 1, the generated surface is an ellipsoid, and if $a_1 = a_2 = a_3$, a sphere. When $\epsilon_1 << 1$ and $\epsilon_2 = 1$, the surface looks like a cylinder.

The power of the superquadric representation lies not in its ability to model perfect geometric shapes, but in its ability to model deformed geometric shapes through deformations such as *tapering* and *bending.* Linear tapering along the z–axis is given by the transformation

$$\begin{aligned} x' &= \left(\frac{k_x}{a_3}z + 1\right)x \\ y' &= \left(\frac{k_y}{a_3}z + 1\right)y \\ z' &= z \end{aligned}$$

where k_x and k_y $(-1 \leq k_x, k_y \leq 1)$ are the tapering parameters with respect to the x and y planes, respectively, relative to the z direction. The bending deformation is defined by the transformation

$$\begin{aligned} x' &= x + \cos(\alpha)(R - r), \\ y' &= y + \sin(\alpha)(R - r), \\ z' &= \sin(\gamma)(\tfrac{1}{k} - r) \end{aligned}$$

where k is the curvature, r is the projection of the x and y components onto the bending plane $z - r$ given by

$$r = \cos\left(\alpha - \tan^{-1}\left(\frac{y}{x}\right)\right)\sqrt{x^2 + y^2},$$

R is the transformation of r given by

$$R = k^{-1} - \cos(\gamma)\left(k^{-1} - r\right),$$

and γ is the bending angle

$$\gamma = zk^{-1}.$$

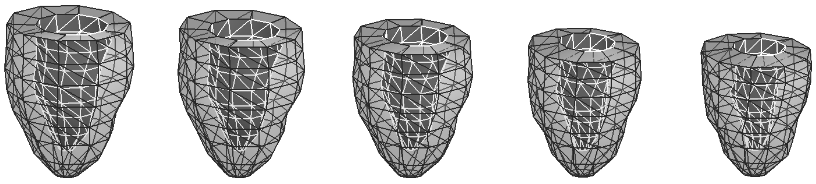


Figure 14.9: Fitted left-ventricle models at five time points during systole, using extended superquadrics with parameter functions (courtesy of Jinah Park and Dmitris Metaxas).

Superquadric models are mainly for use with range data and several procedures for recovering the parameters of a superquadric fit to a surface have been proposed. Figure 14.9 illustrates superquadric fits to 3D data from the left ventricle of a heart at five time points. These are extended superquadrics with parameter functions, where the parameters are not constants but functions.

14.2 True 3D Models versus View-Class Models

All of the above object representations emphasize the three–dimensional nature of the objects, but ignore the problem of recognizing an object from a two–dimensional image taken from an arbitrary viewpoint. Most objects look different when viewed from different viewpoints. A cylinder that projects to a ribbon (see above) in one set of viewpoints also projects to an ellipse in another set of viewpoints. In general, we can partition the space of viewpoints into a finite set of *view classes* [1], each view class representing a set of viewpoints that share some property. The property may be that the same surfaces of the object are visible in an image taken from that set of viewpoints, the same line segments are visible, or the relational distance (see Chapter 11) between relational structures extracted from line drawings at each of the viewpoints is small enough. Figure 14.10 shows the view classes of a cube defined by grouping together those viewpoints which produce line drawings that are topologically isomorphic. Figure 14.11 shows two views of a chair in which most, but not all, of the same surfaces are visible. These views could be grouped together via a clustering algorithm using their relational distance defined over region primitives and the region adjacency relation as a basis for closeness. They are among many different similar views that together form a view class; the number of views is potentially infinite. The main point is that once the correct view class has been determined for an object, the matching to determine the correspondences necessary for pose determination is a highly–constrained, two–dimensional kind of matching.

The use of view classes was proposed by Koenderink and van Doorn (1979). The structure they proposed is called an *aspect graph.* An *aspect* is defined as a qualitatively distinct view of an object as seen from a set of connected viewpoints. The nodes of an aspect graph represent aspects and the arcs connect adjacent aspects. The change in appearance at the boundary between two aspects is called a *visual event.* Algorithms for automatic construction of aspect graphs were developed in the late 1980s, but because the structures are very

[1] Also called characteristic views.

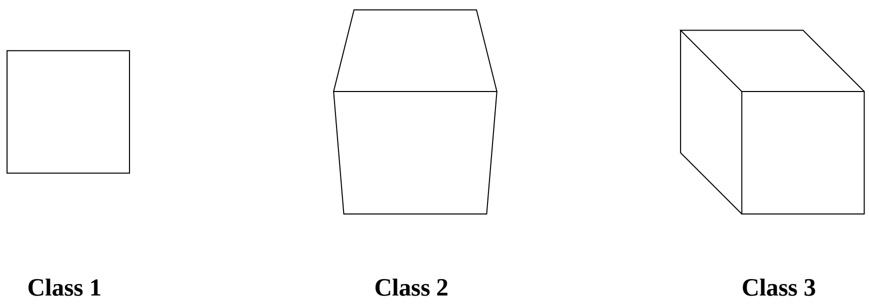


Figure 14.10: The three view classes of a cube defined by grouping together viewpoints that produce topologically isomorphic line drawings.

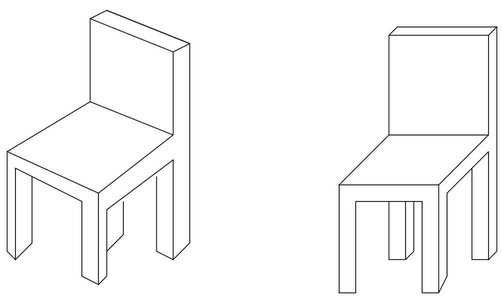


Figure 14.11: Two chair views that belong to the same view class based on their low relational distance.

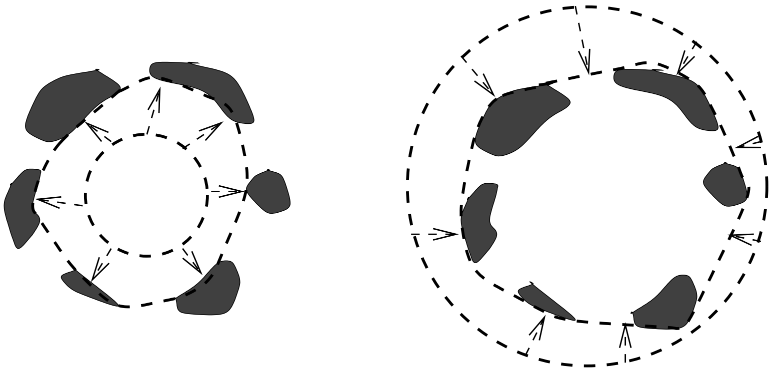


Figure 14.12: (Left) 2D "balloon" or *active contour* being blown up to fit the 2D points. (Right) 2D "rubber band" stretched over the outside of the 2D points.

large for realistic objects, they have not been used much in object recognition. The view class or characteristic view concept has been heavily used instead.

Exercise 4 View-class models

The two chairs of Figure 14.11 both belong to a single view class of a three-dimensional object. Draw pictures of three more common view-classes of the chair.

14.3 Physics-based and Deformable Models

Physics-based models can be used to model the appearance and behavior of an actual physical object being imaged. An example is given below where the object is the human heart. Often, the principles of physics are not used to model an actual physical system, but instead are used in analogy to simulate some image analysis task. An example is given below, where a mesh of triangles is *blown up* as a balloon to fit 3D data points taken from a telephone handset. In the modeling of the heart, the objective is to model the changing shape and behavior of an object over time so that its functioning can be understood. In the modeling of the telephone handset, the objective is to obtain a good mesh model of the static measurements.

A term that is strongly related to the term *physics-based model* is *deformable model.* The latter term emphasizes that the change in object shape is to be modeled.

There has been good progress recently in physics-based and deformable modeling. Theory and applications are rich and more complex than the other areas covered in this text. The brief coverage given here is only for the purpose of introducing the topic and motivating the student to do outside reading in the rapidly developing literature.

14.3.1 Snakes: Active Contour Models

Most of us have placed a rubber band around our outstretched fingers. Our fingers are analogous to five points in 2D and the rubber band is a closed contour "through" the five points.

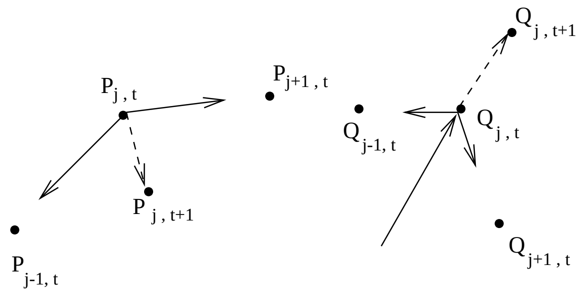


Figure 14.13: (Left) Forces on a point of a stretched rubber band tend to move that point inward. (Right) An inflationary force on a point of a balloon tends to move it outward, provided the elastic forces from neighboring points are exceeded.

The action of the rubber band can be simulated by an *active contour* which moves in an image toward a minimum energy state. A rubber band tends to contract to reduce its stored energy, at least until it has met with supporting forces (the fingers). Figure 14.12(right) illustrates this concept. The small dark regions are analogous to our fingers: the contraction of the rubber band is stopped by these regions. Another aspect of the analogy is that the rubber band will not have infinite curvature; even if stopped by a single point (or wire) the rubber band will smoothly wrap around that point. Preventing or punishing high curvature can be done during simulation. Figure 14.12(left) shows what would happen if a balloon were blown up inside our hand with fingers posed as if grasping a ball. Analogously, we could simulate a virtual balloon being blown up inside some image regions or points.

Figure 14.12 illustrates a critical advantage of active contours: the contour is a complete structure even though the data being fit may be badly fragmented. Moreover, other characteristics can be enforced top-down, such as smoothness, perimeter limits, and the property of being a simple curve. We now give a brief sketch of how the behavior of an active contour can be simulated by a computer algorithm.

To simulate the behavior of an active contour object, we first need a memory state to define the structure and location of the contour. Consider the simple case where we have a fixed set of N points at time t, each located at $P_{j,t}$ and each circularly related to two neighbors $P_{j-1,t}$ and $P_{j+1,t}$. In the case of a virtual rubber band, each point will be subject to a pulling force from the two neighbors, the result of which will accelerate the point $P_{j,t}$ to a new position $P_{j,t+1}$. Figure 14.13(left) illustrates this. We usually consider each point to have unit mass so we can easily compute acceleration in terms of force. Using acceleration we compute velocity, and using velocity we compute position. Thus, our memory state at time t should also contain the acceleration and velocity of each point: moreover, these need not be zero at the start of the simulation. One more data member is needed: we use a Boolean variable to indicate whether or not the point has stopped moving due to running into some data point (called a *hard constraint*). In addition to our active contour object, of course, we need to store the data to be modeled: this could be a greyscale image, a set of 2D edge points, a set of 3D surface points, etc. represented in some manner as described in this chapter or in Chapters 2 or 10.

A simple algorithm for moving active contour points is sketched below. Contour points move until they meet hard constraints or until the resultant force on them is zero. Or, perhaps the algorithm never stops, as in the case where the active contour is tracking a pair of speaking lips! Note that an initial position for the contour is required.

Move contour (snake) points $P_{j,t}$ to next position $P_{j,t+1}$

Input: N data points at time t; each $P_{j,t}$ has velocity $V_{j,t}$ and acceleration $A_{j,t}$.

Output: N data points at time $t+1$; each $P_{j,t+1}$ has velocity $V_{j,t+1}$ and acceleration $A_{j,t+1}$.

Using time step Δt, do at each point $P_{j,t}$ that has not stopped at a hard constraint:

1. Compute the resultant force on $P_{j,t}$ using its neighbors.
2. Using the force, compute the acceleration vector $A_{j,t+1}$.
3. Compute velocity $V_{j,t+1} = V_{j,t} + A_{j,t}\Delta t$
4. Compute new position $P_{j,t+1} = P_{j,t} + V_{j,t}\Delta t$
5. If $P_{j,t+1}$ is within tolerance of some data point, freeze this position.

Algorithm 1: Single update stage for an active contour.

Algorithm 1 sketches a simple stage of an Euler algorithm that computes, for a small time step, acceleration from force, velocity from acceleration and position from velocity. A point's position is frozen when it collides with a data point, edge, or surface patch. In general, this can be a costly computation that requires search through the data structure or image to find such a point.

Hooke's Law models a spring, which is a common element of physics-based models. Suppose a spring of natural length L connects points P_j and P_k. The force F acting on P_j is proportional to how the spring is stretched (or compressed) relative to its natural length.

$$F \;=\; -k_L(||P_j - P_k|| \;-\; L)\frac{P_j - P_k}{||P_j - P_k||} \tag{14.2}$$

This should suffice to simulate our rubber band. A damping force should be added if it is possible that the spring system could oscillate indefinitely. A remaining problem is to determine a good length L. If we are modeling a known object, such as talking lips, we should be able to determine practical values for N, L and k_L. k_L is a "stiffness" parameter that relates force to deformation.

* An energy minimizing formulation

Although the notion of an active contour had been used previously by others, a 1987 paper by Kass, Witkin and Terzopoulos seemed to ignite the interest of the computer vision community. Much of the discussion above was motivated by their view of "snakes", as they were called. Fitting a snake to data was defined as an optimization problem that sought a minimum energy boundary subject to some hard constraints. As useful formulation is to consider that total energy is a sum of three components; (1) *internal contour energy* characterized by the stretching and bending of the contour itself, (2) *image energy* that characterizes how the contour fits to the image intensity or gradient, and (3) *external energy* due to constraint forces. Constraints are used to apply information from an interactive user or a higher level CV process.

A contour parameterized by $s \in [0,1]$ is $\mathbf{v}(s) = [x(s), y(s)]$, which is just a function of real variable s. The problem is to find the function that minimizes the energy defined as follows.

$$E_{contour} = \int_0^1 (E_{internal} + E_{image} + E_{constraints})\, ds. \tag{14.3}$$

$$E_{internal} = \alpha(s)|\mathbf{v}'(s)|^2 + \beta(s)|\mathbf{v}''(s)|^2 \tag{14.4}$$

The snake can be controlled to pass near some designated points by adding in the squared distance between each point and the snake using $E_{constraints}$. E_{image} might just be the sum of the squared distances between a snake point and the closest edge point. The internal energy definition is perhaps more interesting. The first part of $E_{internal}$ punishes higher variance in the lengths of small contour segments — lower energy means small variance in their lengths. The second part punishes curvature. The weighting functions $\alpha(s)$ and $\beta(s)$ are used for blending and can also allow the process to form a sharp corner where a corner detector has found one or to take a long leap over bland texture.

The fitting of active contours to images can be related to the making of airfoils or canoes. Figure 14.14 shows what happens when a strip of wood is nailed at regular intervals to cross sections held in place by a "strongback". The wood bends to smoothly fit the (air) space in between the cross sections making a smooth but possibly complex curve. Contact with the cross sections enforces a hard constraint. High curvature is reduced as the wood distributes the bending energy over many points. Computer algorithms can easily produce such *spline* curves — in fact, Figure 14.14 was produced by such an algorithm in the xfig tool!

The approaches to minimizing the energy of a contour are beyond our scope here. Careful numerical programming is needed to obtain good control of an active contour. Finite elements packages can be used. After 1987, several new works appeared that used dynamic programming, instead of the scale space approach proposed by Kass *et al.* The interested reader can find many interesting works in the literature.

14.3.2 Balloon Models for 3D

A balloon model can be a mesh made by approximating a sphere. Most soccer balls are made from 12 pentagonal and 20 hexagonal pieces, each of which can be divided into triangles. Edges of the triangles can be modeled by springs so that the shape of the entire system can deform, either by expanding or contracting. Figure 14.15 shows such a spherical model expanding inside a cloud of 3D data points taken by sensing a telephone handset. The algorithm freezes the position of a vertex when it makes contact with sensed data. An inflating force is applied to each vertex in the direction of the surface normal interpolated at that vertex. To detect contact with sensed data, the data are searched only in a direction along

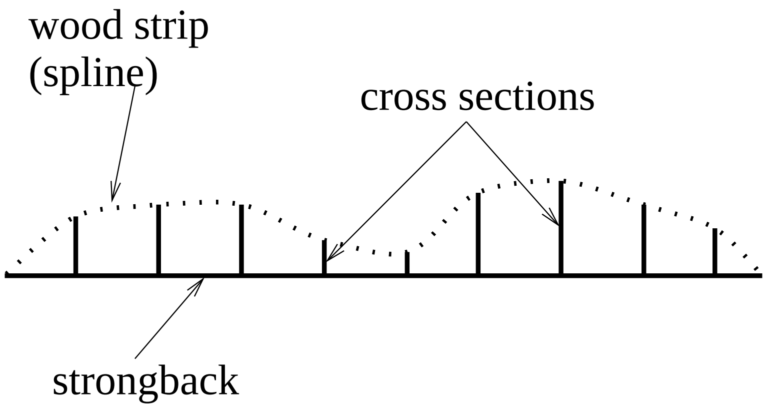


Figure 14.14: A wood strip attached to cross sections makes a "low energy" contour. (Smooth spline courtesy of xfig.)

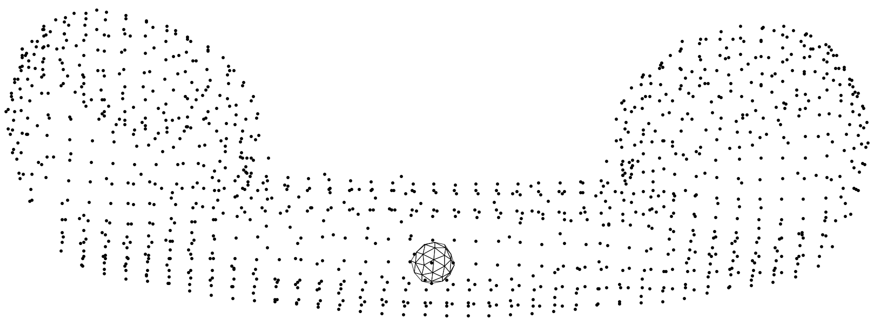
this normal. When an inflating triangle becomes large, the algorithm subdivides it into four triangles. In this manner, the sphere can inflate to the elongated shape of the data as shown in Figure 14.15(b),(c). The 3D data points were obtained from several different views of the object surface using a range scanner, each rigidly transformed to a global coordinate system. Imagine the difficulty of "sewing together" the several different surface meshes that could have been obtained independently from the different views. The balloon model retains the correct topology and approximate uniformity of triangles as it deforms to fit the data. It is difficult to start with just a set of the 3D points and build up a good surface model.

14.3.3 Modeling Motion of the Human Heart

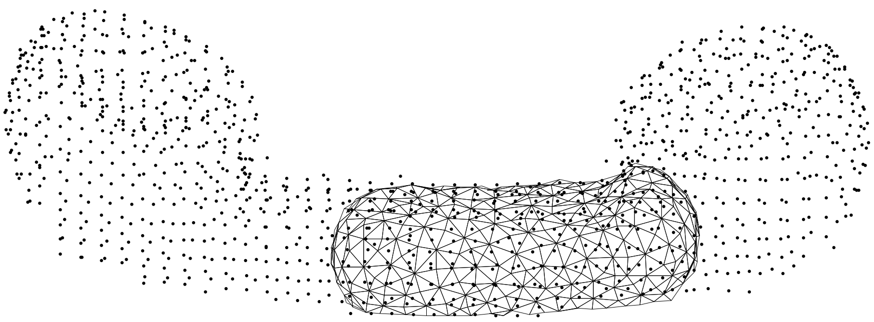
While triangular mesh elements are commonly used for modeling surfaces, 3D volume models can be constructed using tetrahedral elements. Each tetrahedral element has four vertices, four faces, and six edges. *Stiffness* values can be assigned to the edges based on the characteristics of the material being modeled. When forces are applied to various points of the model, the structure will deform. Figure 14.16 shows two states of a beating heart computed from tagged magnetic resonance imagery. The sensor can tag certain parts of living tissue so that they may be sensed in motion in 3D. The heart model fit to the data is meant to model real physics and heart function. The motion of the points of the fitted model can be used to understand how the heart is functioning. The deformation of a tetrahedral element of the model is related to the forces on it and the stiffness of the tissue that it models.

14.4 3D Object Recognition Paradigms

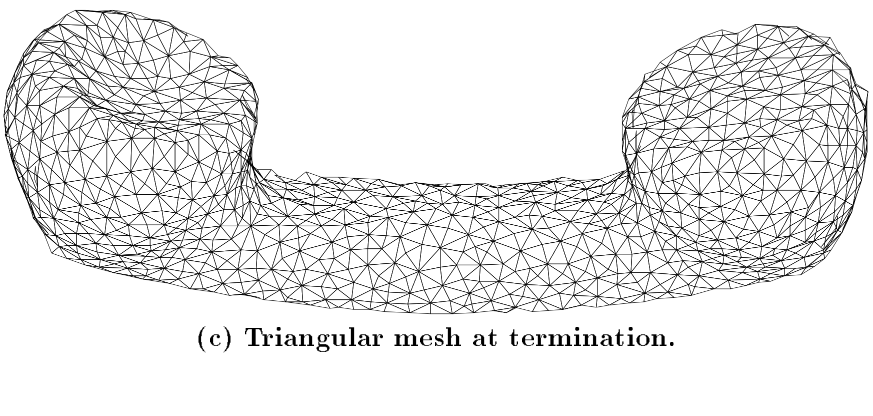
Having surveyed different models for 3D objects, we now discuss the most commonly used general paradigms for 3D object recognition. This is actually difficult, because the method used depends heavily on the application, the type of data, and the requirements of the recognition task. There are many dimensions along which one can classify or constrain an



(a) Initialization with balloon entirely within the 3D point cloud.



(b) Balloon inflated so that some triangles contact data.



(c) Triangular mesh at termination.

Figure 14.15: Three snapshots of the physics-based process of inflating a mesh of triangles to fit a cloud of 3D data. (Courtesy of Yang Chen and Gerard Medioni.)

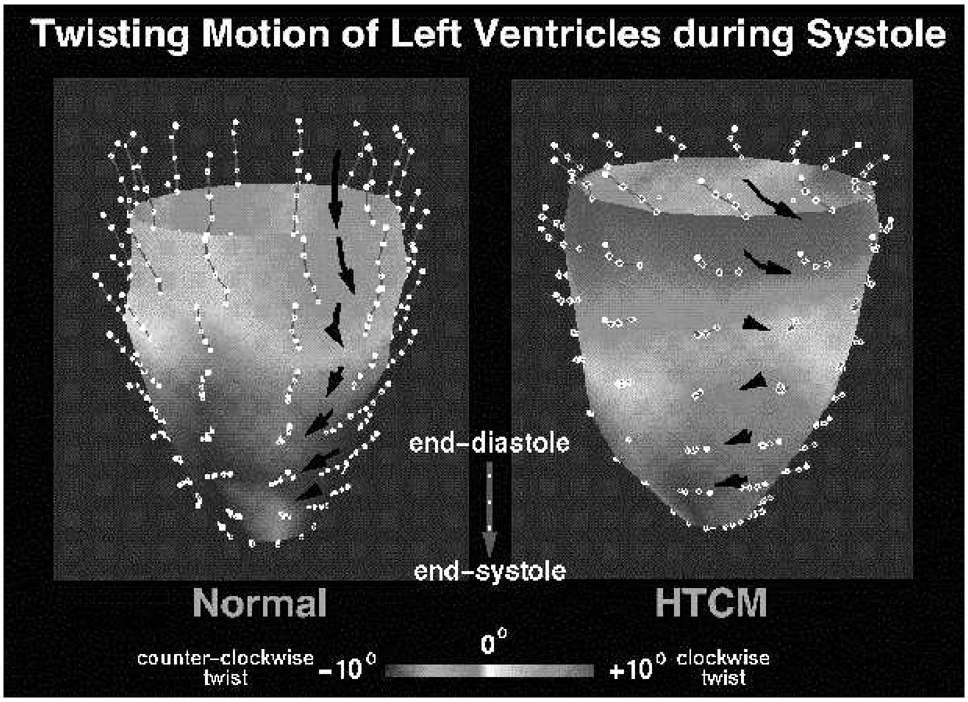


Figure 14.16: Motion of two beating hearts as computed from tagged MRI data. The sensor can tag living tissue and sense its motion in 3D. The heart model fit to the data is meant to model real physics and heart function. The motion vectors shown are different for the two hearts. (Courtesy of Jinah Park and Dimitris Metaxas.)

object recognition problem. These include

- **Is the interest engineering or cognitive science?** In case we want to engineer a solution to an immediate practical problem, our problem may be specific enough to be simple. For example, we may have to grab a steel cylinder from a jumble of many of them. On the other hand, our interest may be in understanding human object recognition. This implies development of a very general theory which is consistent with multifarious psychological data–a much more difficult problem.
- **Does the task involve natural or manufactured objects?** Manufactured objects are usually more regular than natural ones. Moreover, there are rigid iconic prototypes for many man-made objects, making several known matching paradigms applicable. Natural objects are created by processes (geological, biological, etc.) which produce a great deal of variety that may be difficult to model. Furthermore, the context of natural objects may be less contrained and less predictable than the context in which manufactured objects are found. For example, the problem of object recognition for autonomous navigation in outdoor environments seems to be much more difficult than the problem of recognizing and determining the pose of objects for factory automation.
- **Are the object surfaces polyhedral, quadric, or free-form?** Many recognition projects have dealt only with polyhedra, which makes modeling particularly simple. Recently, researchers have turned toward use of quadric surfaces, which it is claimed can model about 85% of manufactured objects. The major convenience is that the modeling and the sensed data are readily described by the same primitives, possibly with some fitting of parameters. It is not clear how best to model sculpted, free-form objects even when they are rigid objects. A sculpted object, such as a sports car, turbin blade, or iceberg, may have many different smoothly blending surface features which are not easily segmented into simple primitives.
- **Is there one object in the scene or are there many?** Some object recognition schemes assume that objects to be recognized are presented in isolation. This may or may not be possible to engineer in the task domain. Multiple object environments are typically harder because object features will be both masked and intermixed. Global feature methods work well only for single objects. The segmentation problem can be acute in multiple object environments.
- **What is the goal of the recognition?** We might need to recognize an object for inspection, grasping, or object avoidance. For inspection, we would look at the small details of at least part of the object –modeling and measurement precision must be good. Grasping an object has different requirements. Not only does the task require some rough geometrical knowledge, but it must also consider balance and strength and accessibility of the object in the workspace. A robot recognizing that an object in its path must be avoided must only have a rough idea of the size, shape, and location of that object.
- **Is the sensed data 2D or 3D data?** Humans can operate quite well with the image from only one eye. Many researchers have designed systems that use only 2D intensity images as input. 2D features from the object image are related to a 3D model via the view transformation; usually the matching process has to discover this transformation as well as the object identity. Matching is often easier if 3D data is directly available

and this is the reason why many current researchers are working with range data. The belief is that the surface shape of objects and their positions can be directly sensed; this in turn provides a direct index into possible object models and also reduces the amount of ambiguity in computing the registration transformation.

- **Are object models geometric or symbolic?** Geometric models describe the exact 3D shape of an object, while symbolic models describe an entire class of objects. Geometric models are heavily used in industrial machine vision, where the objects to be recognized come from a small prespecified set. CAD data is becoming more available and will usually have all the necessary geometric detail. Symbolic models are required in tasks where there are many different varieties of the object class to be recognized. For example, in medical imaging, each organ provides a new object class and every person provides a unique variation. Many objects in our environment, for example, chairs, have many variations and require more than a geometric approach.
- **Are object models to be learned or preprogrammed?** Object models may contain a large amount of precise data which is very difficult for humans to provide. CAD data alone may not be enough; some additional organization of that data, such as emphasizing features, is often necessary. Having a system learn object geometry by presenting the object to its sensors is an attractive possibility.

14.4.1 Matching Geometric Models via Alignment

Recognition by alignment employs the same principles in 3D object recognition as in 2D matching. (See Chapter 11 for basic definitions.) The main idea is expressed in Algorithm 2.

Determine if a set of image data points matches a 3D object model.

1. hypothesize a correspondence between a set of model points and a set of image data points,
2. use that correspondence to determine a transformation from the model to the data,
3. apply the transformation to the model points to produce a set of transformed model points, and
4. compare the transformed model points to the data points to verify or disprove the hypothesis.

Algorithm 2: The Basic Alignment Algorithm

We will look at both the 3D-3D and 2D-3D cases.

3D-3D Alignment Let us assume that the 3D models are, or can be converted to, collections of 3D model-point features. If the data is range data, then corresponding 3D data-point features are required for matching. The alignment procedure finds the correspondences from three chosen model-point features to three corresponding data-point features. This correspondence determines a 3D transformation consisting of a 3D rotation and a 3D translation

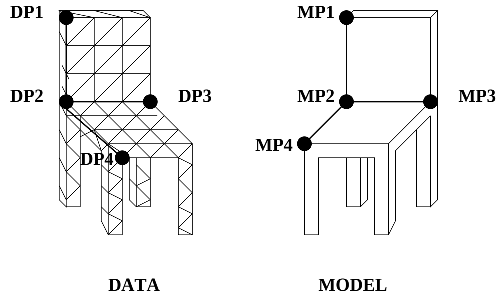


Figure 14.17: Correspondences between 3D model points and 3D mesh data points that can be used to compute the transformation from the model to the data in the 3D-3D alignment procedure.

whose application to the three model points produces the three data points. An algorithm to do this appeared in Chapter 13. If the point correspondence is correct and there is no noise, the correct 3D transformation can be found from the three matches. Since this is rarely the case, more robust procedures that typically use on the order of ten correspondences have been developed. In any case, once the potential transformation has been computed, it is applied to all of the model points to produce a set of transformed model points that can be directly compared to the full set of data points. As in the 2D case, a verification procedure decides how well the transformed model points line up with the data points and either declares a match or tries another possible correspondence. As in the 2D case, there are intelligent variations that select the corresponding points by the local-feature-focus method or by some other perceptual grouping technique. Figure 14.17 illustrates the 3D-3D correspondences that come about from matching a three-segment 3D junction of a chair model to a 3D mesh dataset.

Exercise 5 3D-3D feature alignment

Junctions of line segments are common in polyhedral objects. Consider a 3D cup object having a cylindrical part with a cylindrical cavity for the liquid and a semicircular handle. What features of the cup might be detected in the 3D data and used for finding correspondences in matching?

Feature extraction is an important issue here. If the class of objects is such that distinguished points, such as corners points, peaks, and pits, can be easily found, then the above procedure should work well. If surfaces are smooth and distinguished points are rare or nonexistant, then a better method for finding correspondences is needed. Johnson and Hebert at CMU have developed a very robust method for exactly this problem. Their 3D object representation consists of 1) a 3D mesh model of the object and 2) a set of *spin images* constructed from the mesh that characterize local shape features of the object.

Given a mesh model of a 3D object, the surface normal can be estimated at each vertex of the mesh. Then the relationship between any oriented point in 3D-space and a surface normal at a particular vertex can be represented by two distance parameters, α and β, where α is its perpendicular distance to the surface normal, and β is its signed perpendicular distance to the tangent plane at that vertex. Rotational angles are omitted from this description, because they are ambiguous.

A *spin image* is a kind-of histogram that can be computed at a selected vertex of the mesh. Each spin image will have a set of *contributing points* used in its construction. The size of the volume of contributing points depends on two spin-image parameters: D, the maximum distance from a contributing point to the selected vertex, and A, the angle allowed between the normal of the contributing point and the normal of the vertex. A spin image is constructed about a specified oriented point o with respect to a set of contributing points C that have been selected based on specified spin-image parameters A and D. An array of accumulators $S(\alpha, \beta)$ represents the spin image and is initially set to zero. Then for each point $c \in C$, its distance parameters α and β are computed with respect to the selected mesh vertex o, and the accumulator bin corresponding to this α and β is incremented. Note that the size of a bin in the accumulator array is on the order of the median distance between vertices in the 3D mesh. Figure 14.18 gives some examples of spin images.

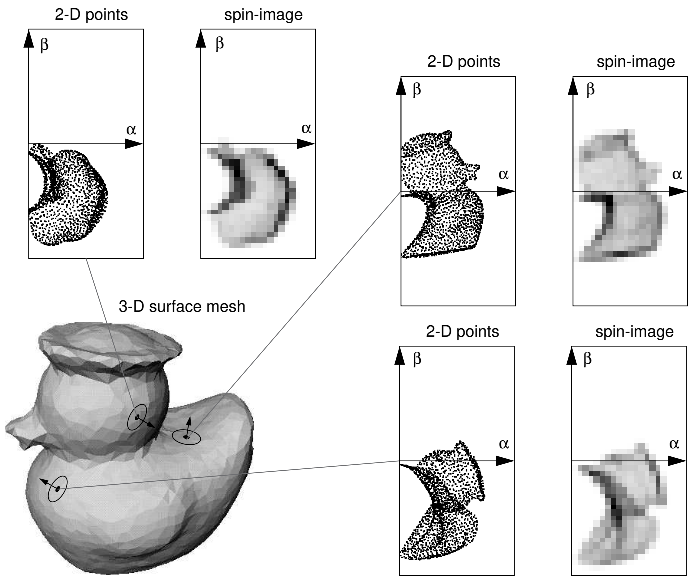


Figure 14.18: Examples of spin images. (Figure courtesy of Andrew Johnson with permission of IEEE. Reprinted from "Efficient Multiple Model Recognition in Cluttered 3-D Scenes," by A. E. Johnson and M. Hebert, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 1998. © 1998 IEEE)

Spin images are constructed at each vertex of the mesh model. This gives information on the local shape at every point of the mesh. To match two objects, the two sets of spin images are used. The spin image at each point of the first object is compared to the spin image at each point of the second object by computing the correlation coefficient of the pair. Those point pairs with high correlations form the 3D point correspondences needed for object matching. The point correspondences are grouped and outliers eliminated using geometric consistency. Then, as in alignment in general, a rigid transformation is computed and used to either verify the match or rule it out. Figure 14.19 shows the operation of the spin-image recognition method on a difficult, cluttered image containing six different objects that correspond to models in a database of twenty object models.

2D-3D Alignment Alignment can also be applied to 2D-3D matching in which the object model is three-dimensional, but the data comes from a 2D image. In this case, the transformation from model points to data points is more complex. In addition to the 3D rotation and 3D translation, there is a perspective projection component. The full transformation can be estimated from a set of point correspondences, a set of line segment correspondences, a 2D ellipse to 3D circle plus single point correspondence, or combinations of all three types of features. This gives us a very powerful tool for matching. Correspondences are hypothesized either blindly or through relational matching (see below) and used to determine a potential transformation. The transformation is applied to the 3D model features to produce 2D data features. Here a new problem comes up that did not appear in 2D-2D or 3D-3D alignment. In any 2D perspective view of a 3D object, some of the transformed features appear on surfaces that do not face the camera and are occluded by other surfaces that are closer to the viewer. Thus in order to accurately produce a set of transformed features to compare to image features, a hidden feature algorithm must be applied. Hidden feature algorithms are related to graphics rendering algorithms and, if applied in software, can be prohibitively slow. If appropriate mesh models and graphics hardware is available, then the full rendering is possible. Otherwise, it is common to either ignore the hidden feature problem or use an approximate algorithm that is not guaranteed to be accurate, but may be good enough for verification.

The TRIBORS object recognition system uses view-class models of polyhedral objects and finds correspondences between triplets of model line segments and triplets of 2D image line segments. Model triplets are ranked in a training phase so that triplets with high probabilities of detection are selected first in the matching phase and those with low probabilities are not considered at all. Triplets are described by a vector of nine parameters that describe the appearance of that triplet in the view class being matched. Figure 14.20 shows the parametrization of a line segment triplet. A model triplet is matched to an image triplet that has similar parameters. Once a match is hypothesized, the line junction points from the data triplet are paired with the hypothesized corresponding 3D vertices from the model, and an iterative point-to-point correspondence-based exterior orientation algorithm (as given in Chapter 13) is used to determine the transformation. The transformation is then applied to a wireframe model of the 3D object and the visible edges are determined through a hidden line detection algorithm. For each predicted edge, the closest image line segment is determined and verification is performed based on how similar each predicted edge is to its closest image segment. Figure 14.21 shows the operation of the TRIBORS system.

Smooth Object Alignment We have talked about the alignment of a 3D mesh model to a 3D range image and a 3D polyhedra model to a 2D intensity image. We consider here the problem of computing the identity and pose of a free-form 3D object from a single 2D

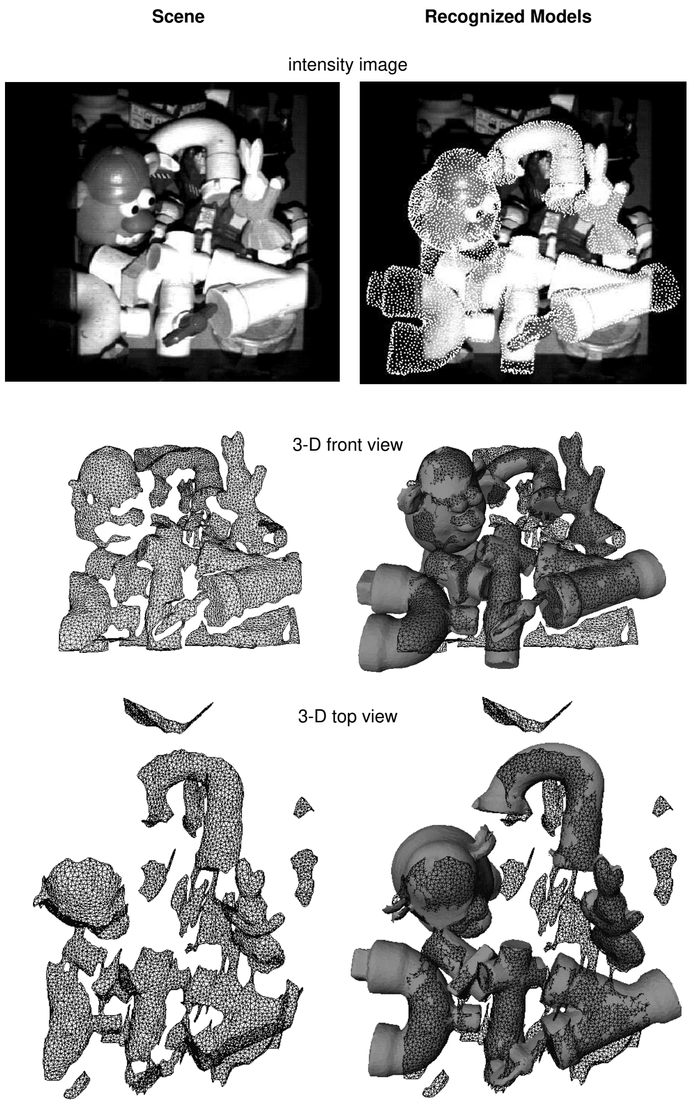


Figure 14.19: Operation of the spin-image recognition system. (Figure courtesy of Andrew Johnson with permission of IEEE. Reprinted from "Efficient Multiple Model Recognition in Cluttered 3-D Scenes," by A. E. Johnson and M. Hebert, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 1998. © 1998 IEEE)

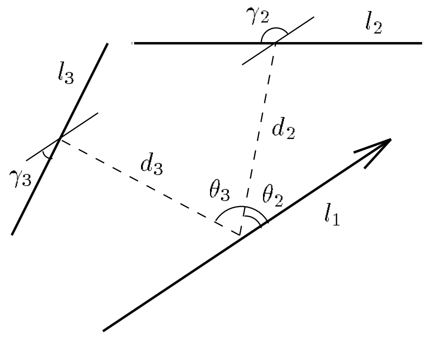


Figure 14.20: The parametrization of a line triplet in the TRIBORS system. The quantities d_2 and d_3 are the distances from the midpoint of line l_1 to the midpoints of lines l_2 and l_3, respectively. Angles γ_2 and γ_3 are the angles that line segments l_2 and l_3 make with line segment l_1. Angles θ_2 and θ_3 are the angles between line segment l_1 and the connecting lines shown to l_2 and l_3, respectively.

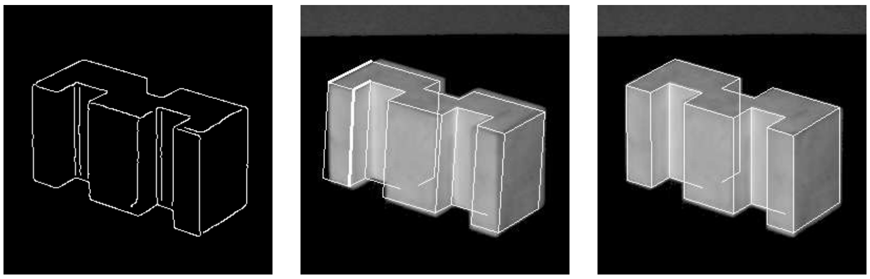


Figure 14.21: (a) The edges extracted from a real image. (b) The matched triplet of line segments (thick lines) and the initial pose estimate. (c) The final match and pose.

Exercise 6 Matching in TRIBORS

TRIBORS uses the 9 parameters associated with a triple of line segments to identify potential matches between a model triple and an image triple. Generate several different views of a single view class of a 3D polyhedral object, such as the chair object of Figure 14.11. Identify three major line segments that appear in all your views and compute the 9 parameters shown in Figure 14.20. How similar are the different parameter vectors that you computed? Compare the 9 parameters of these three line segments to those of a completely different set of three line segments. How well do the 9 parameters discriminate among triples of line segments?

intensity image. The solution we will look at employs a view-class type of model, but the representation of a view class is quite different from the sets of triplets of line segments that TRIBORS used and matching is performed at the lowest level on edge-images.

The algorithm discussed here is based on the work of Chen and Stockman, who created a system to determine the pose of 3D objects with smooth surfaces. In this system, a 3D object is modeled by a collection of $2\frac{1}{2}$D views (called model aspects) each constructed from five images taken by rotating the viewpoint up, down, left and right of a central viewpoint. Input for construction of one model aspect of a car is shown in Figure 14.22. The silhouette of the central edgemap is extracted and segmented into curve segments whose invariant features are derived for indexing to this model aspect during recognition.

Stereo-like computations are performed to compute points [x,y,z] on the 3D rim for each 2D silhouette point [u,v] in the central image. The up and down images are used to compute the curvature of the object rim in the y direction and the left and right images are used to compute curvature in the x direction. Similarly, stereo-like computations are used to compute the 3D locations of crease and mark points in the central edgemap. Thus, the $2\frac{1}{2}$D model aspects consist of the 3D rim, crease and mark points corresponding to the central edge image plus the x and y curvature at each of these points. Using this information, a mathematical formula can then produce an edge map for any other viewpoint in that view class, given the view parameters. A view class is described by 1) the set of 3D points and curvatures described above and 2) a set of invariant features to be used for indexing. The 3D points are derived from stereo-like correspondences between the central and adjacent edgemaps as described in Chapter 13. The invariant features are derived from the 2D edgemap of the central image as described in Chapter 10.

An image to be analyzed is processed to produce its edge map and a set of curve segments. The curve segments are used to index the database of model views to produce object-view hypotheses. The matching scheme tests hypotheses generated from the indexing scheme; each hypothesis includes both object identity and approximate pose. Verification is carried out by fitting the $2\frac{1}{2}$D aspect of each candidate model to the observed edgemap. Initially, the pose of the object is set to the pose that would have produced the central aspect that has been hypothesized to match. The projected edge image of that model aspect is compared to the observed edge map. In most cases, they will not align well-enough. Thus the matching will proceed by refining pose parameters $\vec{\omega}$ in order to diminish the 2D distance between the projected model edgemap and the observed edgemap. Figure 14.23 illustrates the matching step. The edgemap derived from the input image is shown in (a), and a model pose hypothesized from the indexing step is shown in (b). Several iterations of model boundary generation are shown in (c), and the first acceptable match is shown in (d).

14.4.2 Matching Relational Models

As in two-dimensional matching, relational models can be used in 3D object recognition to move away from the geometric and toward the symbolic. Algorithm 3 summarizes the basic relational distance matching technique that was described in Chapter 11, simplified to single relations. The exact models and methods used depend on whether the image data is 3D or 2D.

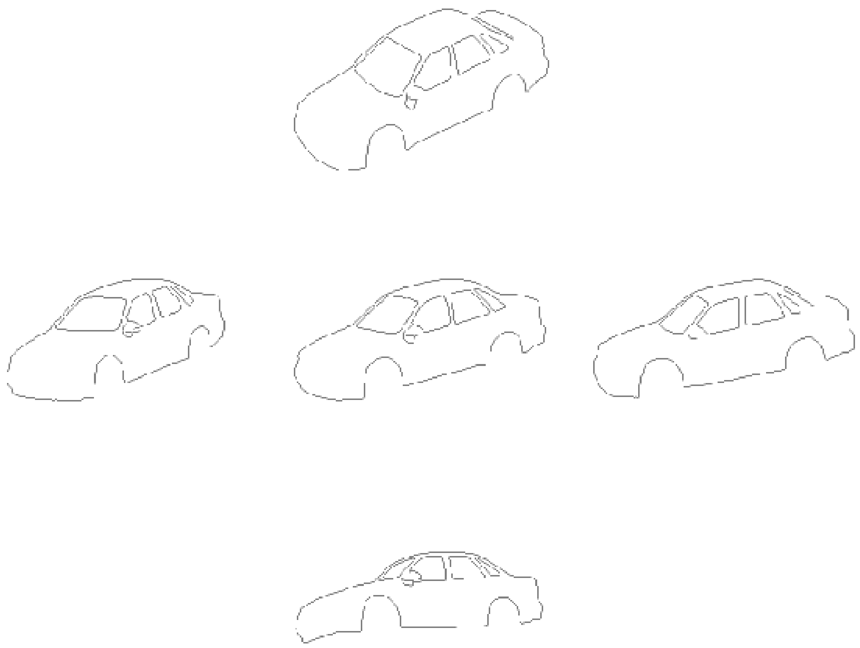
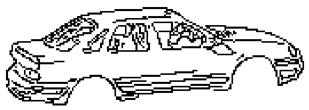


Figure 14.22: Five input images used for constructing one model aspect of a car.

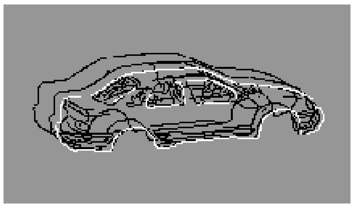
3D Relational Models Three-dimensional relational models are composed of 3D primitives and 3D spatial relationships. Primitives may be volumes, surface patches, or line/curve features in 3D space. Generalized cylinders are commonly used as volumetric primitives along with some kind of 3D connection relationship. *Geons, or geometric ions*, are volumetric primitives hypothesized to be used in human vision, have also been used in 3D object recognition. Industrial objects may be represented by their planar and cylindrical surfaces and the surface adjacency relationship. Three-dimensional line and curve segments can be used with different kinds of spatial relationships, such as connections, parallel pairs, and collinear pairs.

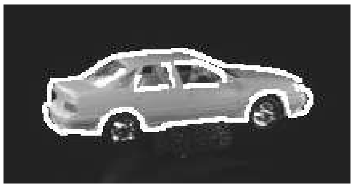
The *sticks, plates, and blobs* models were designed to describe very rough models of 3D objects and are intended for the description and recognition of complex man-made objects, which are made up of many parts. The parts can have flat or curved surfaces, and they exist in a large variety. Instead of trying to describe each part precisely, as in the surface–edge–vertex models, for rough matching each part can be classified as a *stick*, a *plate*, or a *blob*. Sticks are long, thin parts that have only one significant dimension. Plates are flatish, wide parts with two nearly flat surfaces connected by a thin edge between them. Plates have two significant dimensions. Blobs are parts that have all three significant dimensions. All three kinds of parts are near–convex, so a stick cannot bend very much, the surfaces of a plate cannot fold very much, and a blob can be bumpy, but cannot have large concavities. Figure



(a) detected edges

(b) model aspect





(c) fitting steps **(d) final fit**

Figure 14.23: Matching edgemaps using 1:2 scale s; (a) observed edgemap; (b) the model edgemap; (c) evolution of convergence in the alignment algorithm; (d) fitted edgemap shown superimposed on the original image. (Example courtesy of Jin-Long Chen.)

Exercise 7

In each case below explain how many object models and how many model aspects would be needed for the application. Also, explain what accuracy of pose might be needed. (a) In a carwash, the automatic machinery needs to reset itself according to the model of car that has entered. (b) In a parking garage, a monitoring and inventory system needs to recognize each car model that enters and exits and record the time of the event. (c) A computer vision system needs to scan an automobile graveyard to record how many wrecks are present and what kind they are.

Exercise 8

Make a curvature model aspect of some object and show that it can generate silhouettes of that object under small rotations. For example, consider a torus of major outside diameter 10 and minor diameter of 1. The model aspect is centered at the view perpendicular to the circle of the major diameter. Define a set of 3D points along the model silhouette together with their x and y curvatures. Then show how this silhouette changes under small rotations by creating synthetic images.

Determine if two relational descriptions are similar enough to match.
P is a set of model parts.
L is a set of possible labels for the parts.
$\mathbf{R_P}$ is a relation over the parts.
$\mathbf{R_L}$ is a relation over the labels.

Find a mapping f from P to L that minimizes the error given by $E_S(f) = \mid R_P \circ f - R_L \mid + \mid R_L \circ f^{-1} - R_P \mid$ using an interpretation tree, discrete or probabilistic relaxation, or other methods described in Chapter 11.

Algorithm 3: The Basic Relational Distance Matching Technique

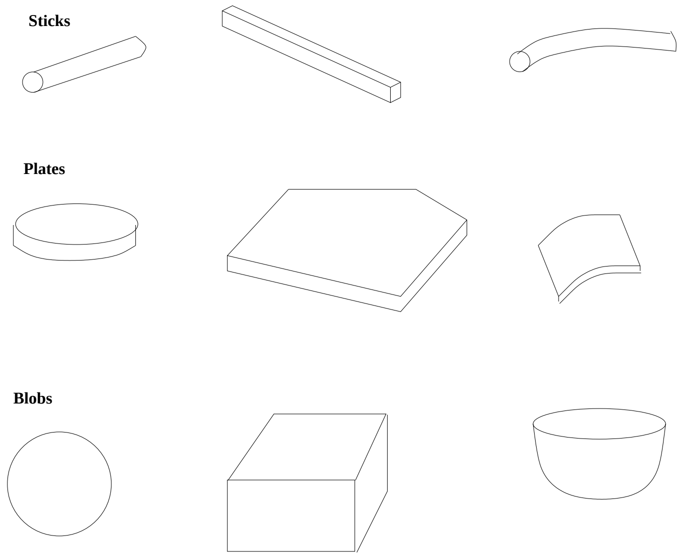


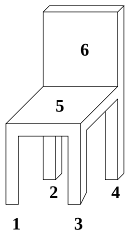
Figure 14.24: Several examples each of sticks, plates, and blobs.

14.24 shows several examples of sticks, plates, and blobs.

A sticks–plates–and–blobs model describes how the sticks, plates, and blobs are put together to form an object. These descriptions are also rough; they cannot specify the physical points where two parts join. A stick has two logical end points, a logical set of interior points, and a logical center of mass that can be specified as connection points. A plate has a set of edge points, a set of surface points, and a center of mass. A blob has a set of surface points and a center of mass. Only this minimal information can be used in the object models.

The relational model for a sticks–plates–and–blobs model is a good example of a fairly detailed symbolic object model that has been used successfully in symbolic object recognition. It consists of five relations. The unary SIMPLE PARTS relation is a list of the parts of the object. Each part has several descriptive attributes including its type (stick, plate or blob) and may also include numeric information pertaining to the size and/or shape of the part. The CONNECTS/SUPPORTS relation contains some of the most important information on the structure of the object. It consists of 6-tuples of the form $(s_1, s_2, SUPPORTS, HOW)$. The components s_1 and s_2 are simple parts, SUPPORTS is true if s_1 supports s_2 and false otherwise, and HOW describes the connection type of s_1 and s_2.

The other four relations express constraints. The TRIPLE CONSTRAINT relation has 4–tuples of the form $(s_1, s_2, s_3, SAME)$ where simple part s_2 touches both s_1 and s_3, and SAME is true if s_1 and s_3 touch s_2 on the same end (or surface) of s_2 and false otherwise. The PARALLEL relation and the PERPENDICULAR relation have pairs of the form (s_1, s_2) where simple parts s_1 and s_2 are parallel (or perpendicular) in the model. Figure 14.25 illustrates the sticks–plates–and–blobs model of a prototype chair object. All chairs with similar relations should match this model, regardless of the exact shapes of the parts.



SIMPLE-PARTS

PART#	TYPE
1	Stick
2	Stick
3	Stick
4	Stick
5	Plate
6	Plate

CONNECTS-SUPPORTS

SP1	SP2	SUPPORTS	HOW
1	5	True	end-edge
2	5	True	end-edge
3	5	True	end-edge
4	5	True	end-edge
5	6	True	edge-edge

TRIPLES

SP1	SP2	SP3	SAME
1	5	2	True
1	5	3	True
1	5	4	True
1	5	6	False
2	5	3	True
2	5	4	True
2	5	6	False
3	5	4	True
3	5	6	False
4	5	6	False

PARALLEL

SP1	SP2
1	2
1	3
1	4
2	3
2	4
3	4

PERPENDICULAR

SP1	SP2
1	5
2	5
3	5
4	5
5	6

Figure 14.25: The full relational structure of the sticks–plates–and–blobs model of a chair object.

Exercise 9 Sticks–plates–and–blobs models

Draw a picture of a simple polyhedral desk object and construct a full relational sticks–plates–and–blobs model for the object.

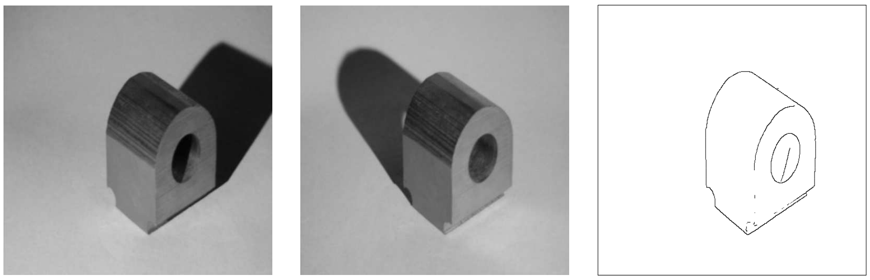


Figure 14.26: The left and right images of an industrial object and the edges extracted through an image processing step that removes shadows and most highlights.

View-Class Relational Models When the data consists of 2D images, view class models can be used instead of full 3D object models. Training data derived either synthetically or from a set of real images of the object can be used in the construction of these models. Depending on the class of objects, a set of useful 2D features are extracted from images of the object. The features extracted from each training image are used to produce a relational description of that view of the object. The relational descriptions are then clustered to form the view-classes of the object. Each view class is represented by a combined relational description that includes all the features that have been detected in all views of that view class. The combined relational description is the relational model of the view class. Typically, an object will have on the order of five view classes, each with its own relational description. The view-class models can be used for full relational matching, which is expensive when there are many different models in the database or for relational indexing, as introduced in Chapter 11. For our example here, we take the relational indexing approach.

The RIO object recognition system recognizes 3D objects in multi-object scenes from 2D images. Images are taken in pairs, with the camera fixed for the pair. One image has the light source at the left and the other has the light source at the right. The two images are used to determine which regions are shadows and highlights, so that a high-quality edge image of just the objects can be obtained. The edge image is used to obtain straight-line and circular-arc segments from which the recognition features are constructed. Figure 14.26 shows a sample left and right image pair and the extracted edge image obtained. Figure 14.27 shows the straight lines and circular arcs extracted from the edge image.

RIO objects can have planar, cylindrical, and threaded surfaces. This leads to a number of useful high-level features. The ten features employed by RIO are: ellipses, coaxial arcs (two, three, and multiple), parallel pairs of line segments (both close and far), triples of line segments (U-shaped and Z-shaped), L-junctions, Y-junctions, and V-junctions. Figure 14.28 shows some of the features constructed from the line segments and arcs of Figure 14.27. The line features include two L-junctions and a pair of parallel lines. The arc cluster shows three coaxial arcs. Note that not every line segment or arc segment becomes a part of the final features used in matching. Figure 14.29 illustrates the entire set of RIO features.

In addition to the labeled features, RIO uses labeled binary relations over the features

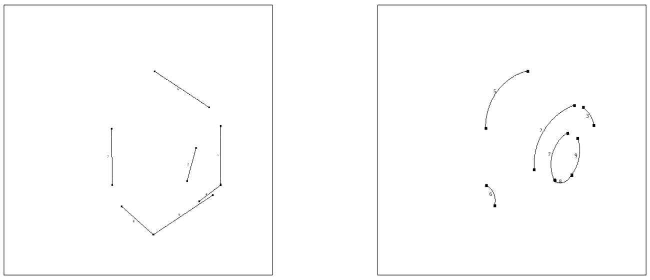


Figure 14.27: The straight line segments and circular arcs extracted from the edge image of Figure 14.26.

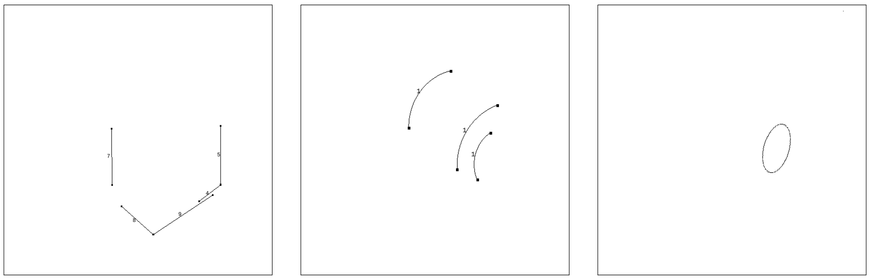
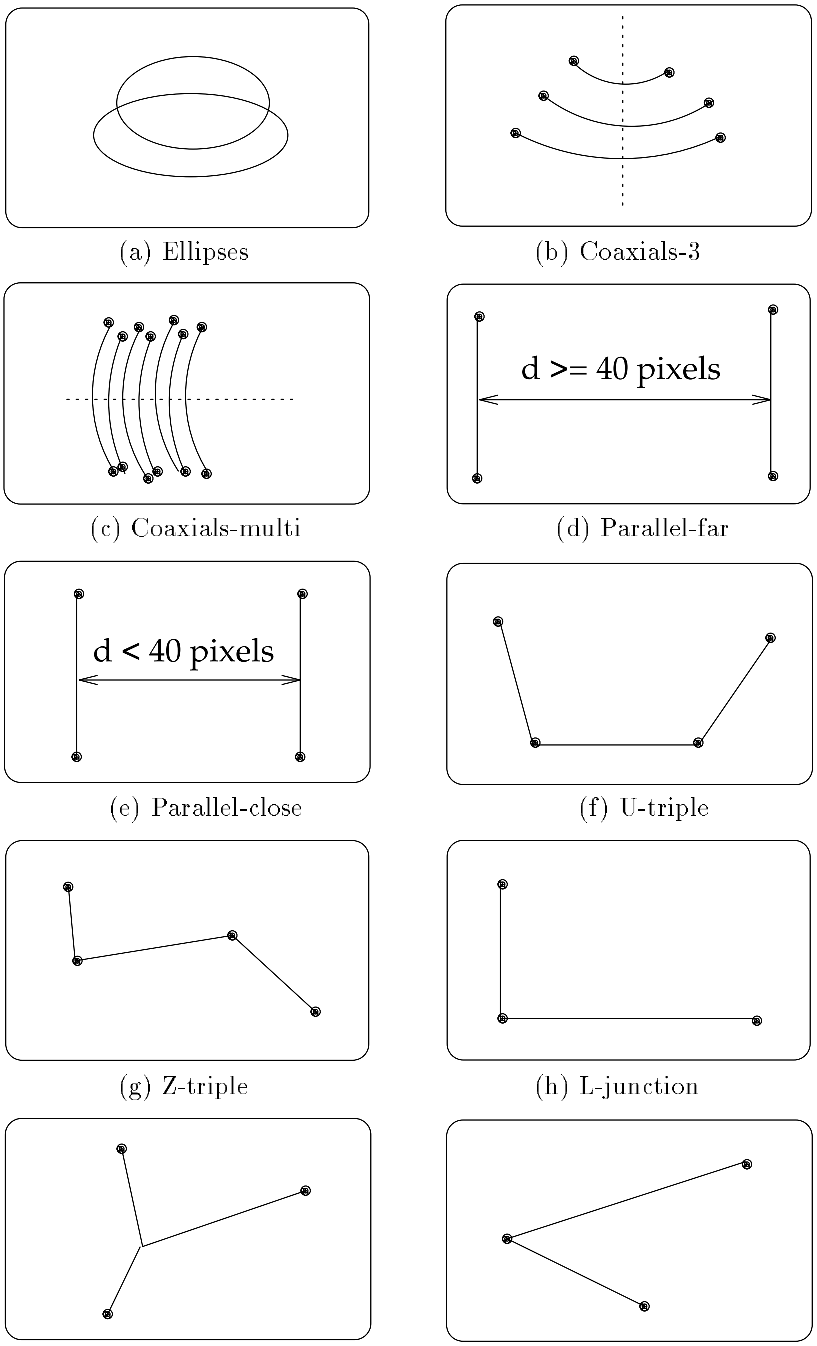
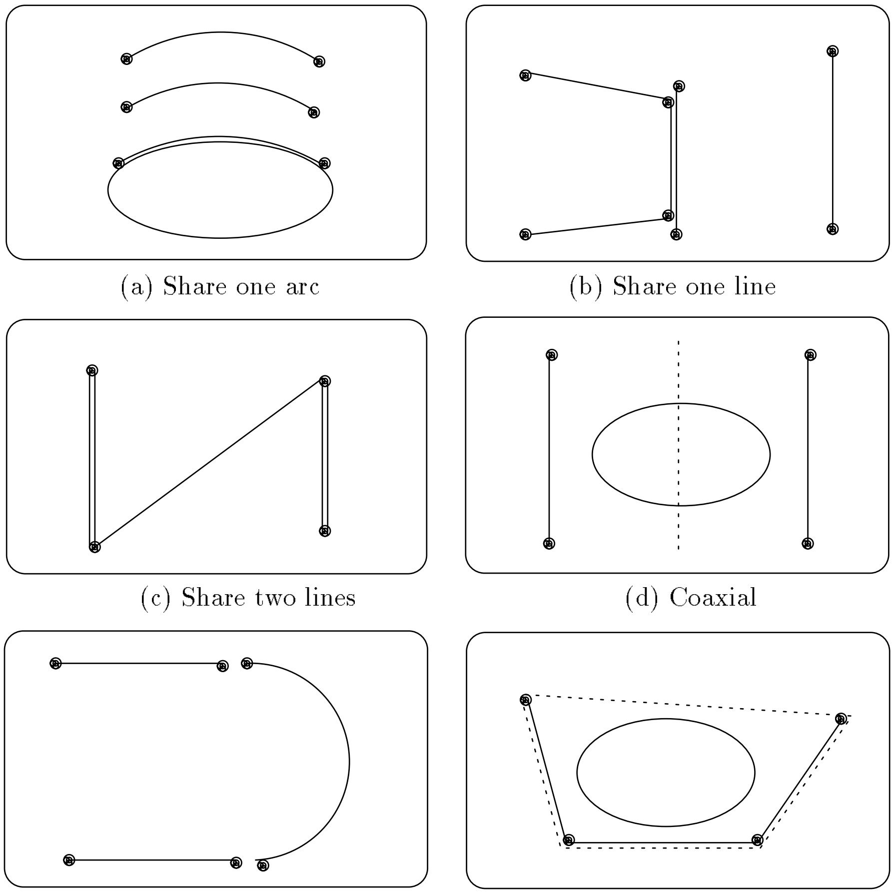


Figure 14.28: The line features, arc features and ellipse features constructed from the lines and arcs of Figure 14.27.



(a) Ellipses
(b) Coaxials-3
(c) Coaxials-multi
(d) Parallel-far
(e) Parallel-close
(f) U-triple
(g) Z-triple
(h) L-junction
(i) Y-junction
(j) V-junction

Figure 14.29: Features used in the RIO system.



(a) Share one arc

(b) Share one line

(c) Share two lines

(d) Coaxial

(e) Close at extremal points

(f) Bounding box encloses/
is enclosed by bounding box

Figure 14.30: Relations between sample pairs of features.

to recognize objects. The relationships employed by RIO are: sharing one arc, sharing one line, sharing two lines, coaxiality, proximity at extremal points, and encloses/enclosed-by, as shown in 14.30.

The structural description of each model-view is a graph structure whose nodes are the feature types and whose edges are the relationship types. For use in the relational indexing procedure, the graph is decomposed into a set of 2-graphs (graphs of two nodes), each having two nodes and a relationship between them. Figure 14.31 shows one model-view of a hexnut object, a partial full-graph structure representing three of its features and their relationships, and the 2-graph decomposition.

Relational indexing in a procedure that matches an unknown image to a potentially large database of object-view models, producing a small set of hypotheses as to which objects are present in the image. There is an off-line preprocessing phase to set up the data structures and an on-line matching phase. The off-line phase constructs a hash table that is used by

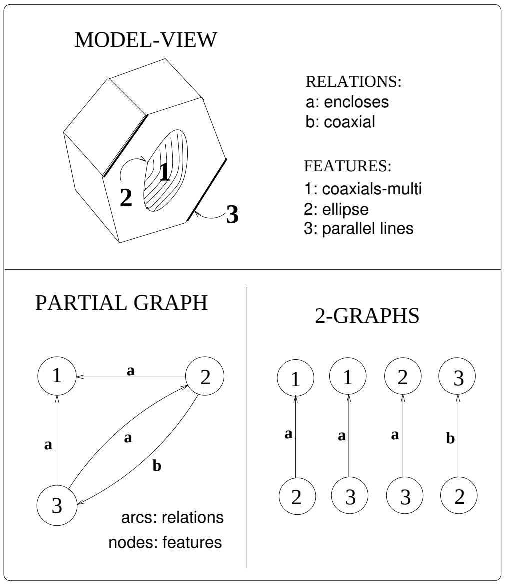


Figure 14.31: Sample graph and corresponding 2-graphs for the "hexnut" object.

the on-line phase. The indices to the hash table are 4-tuples representing 2-graphs of a model-view of an object. The components of the 4-tuple are the types of the two nodes and the types of the two relationships. For example, the 4-tuple (ellipse, far parallel pair, enclosed-by, encloses) means that the 2-graph represents an ellipse feature and a far parallel pair feature, where the ellipse is enclosed by the parallel pair of line segments and the parallel pair thus encloses the ellipse. Since most of the RIO relations are symmetric, the two relationships are often the same. For instance, the 4-tuple (ellipse, coaxial arc cluster, share an arc,share an arc) describes a relationship where an ellipse and a coaxial arc cluster share an arc segment. The symbolic components of the 4-tuples are converted to numbers for hashing. The preprocessing stage goes through each model-view in the database, encodes each of its 2-graphs to produce a 4-tuple index, and stores the name of the model-view and associated information in a list in the selected bin of the hash table.

Once the hash-table is constructed, it is used in on-line recognition. Also used is a set of accumulators for voting, one for each possible model-view in the database. When a scene is analyzed, its features are extracted and a relational description in the form of a set of 2-graphs is constructed. Then, each 2-graph in the description is encoded to produce an index with which to access the hash table. The list associated with the selected bin is retrieved; it consists of all model-views that have this particular 2-graph. A vote is then cast for each model-view in the list. This is performed for all the 2-graphs of the image. At the end of the procedure, the model-views with the highest votes are selected as hypotheses. Figure 14.32 illustrates the on-line recognition process. The 2-graph shown in the figure is converted to the numeric 4-tuple (1,2,9,9) which selects a bin in the hash table. That bin is accessed to retrieve a list of four models: M_1, M_5, M_{23}, and M_{81}. The accumulators of

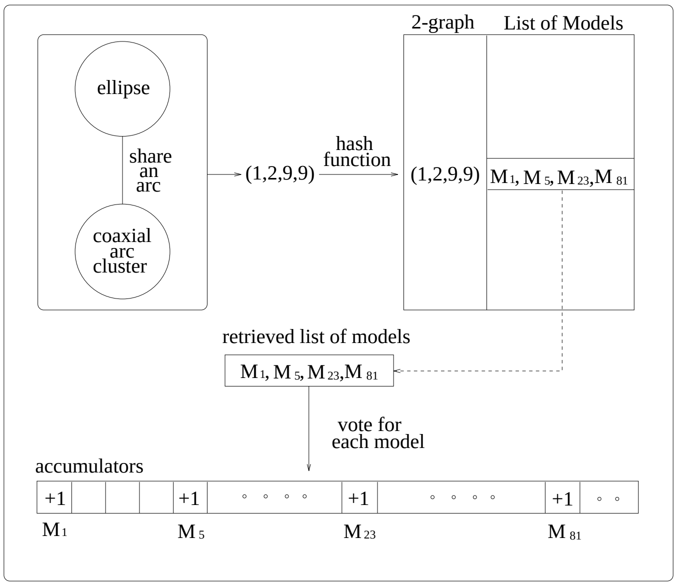


Figure 14.32: Voting scheme for relational indexing.

each of these model-views are incremented.

After hypotheses are generated, verification must be performed. The relational indexing step provides correspondences from 2D image features to 2D model features in a model-view. These 2D model features are linked with the full 3D model features of the hypothesized object. The RIO system performs verification by using corresponding 2D-3D point pairs, 2D-3D line segment pairs, and 2D ellipse-3D circle pairs to compute an estimate of the transformation from the 3D model of the hypothesized object to the image. Line and arc segments are projected to the image plane and a distance is computed that determines if the verification is successful or if the hypothesis is incorrect. Figures 14.33 and 14.34 show a sample run of the RIO system. Figure 14.33 shows the edge image from a multi-object scene and the line features, circular arc features and ellipses detected. Figure 14.34 shows an incorrect hypothesis produced by the system, which was ruled out by the verification procedure and three correct hypotheses, which were correctly verified. The RIO pose estimation procedure was given in Chapter 13. Figure 14.35 shows a block diagram of the whole RIO system.

Exercise 10 Relational indexing

Write a program that implements relational indexing for object matching. The program should use a stored library of object models, each represented by a set of 2-graphs. The input to the recognition phase is a representation of a multi-object image, also in terms of a set of 2-graphs. The program should return a list of each model in the database that has at least 50% of its 2-graphs in the image.

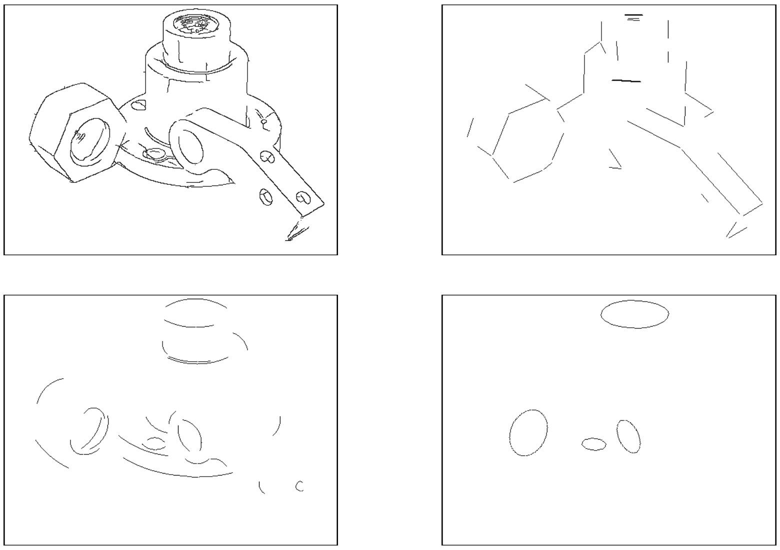


Figure 14.33: A test image and its line features, circular arc features, and ellipse features.

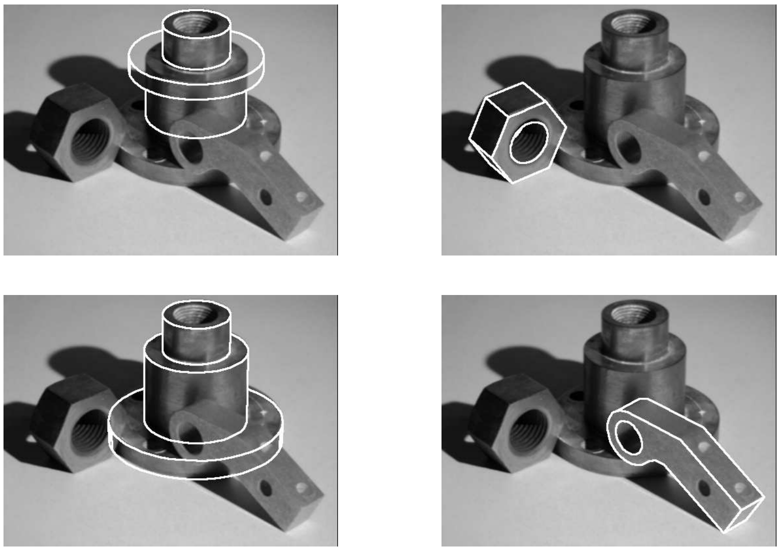


Figure 14.34: An incorrect hypothesis (upper left) and three correct hypotheses.

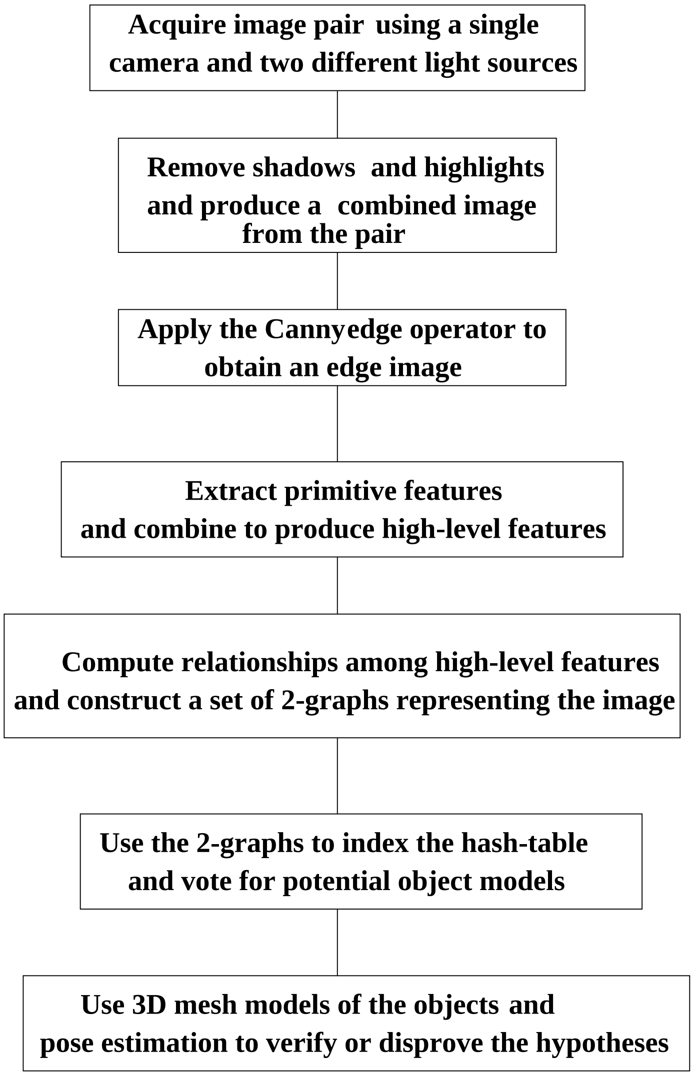


Figure 14.35: Flow diagram of the RIO object recognition system.

14.4.3 Matching Functional Models

Geometric models give precise definitions of specific objects; a CAD model describes a single object with all critical points and dimensions spelled out. Relational models are more general in that they describe a class of objects, but each element of that class must have the same relational structure. For example, a chair might be described as having a back, a seat, and four legs attached underneath and at the corners of the seat. Another chair that has a pedestal and base instead of four legs would not match this description. The function-based approach to object recognition takes this a step further. It attempts to define classes of objects through their function. Thus a chair is something that a human would sit on and may have many different relational structures, which all satisfy a set of functional constraints.

Function-based object recognition was pioneered by Stark and Bowyer in their GRUFF system. GRUFF contains three levels of knowledge:

1. the category hierarchy of all objects in the knowledge base
2. the definition of each category in terms of functional properties
3. the knowledge primitives upon which the functional definitions are based

Knowledge Primitives Each knowledge primitive is a parametrized procedure that implements a basic concept about geometry, physics or causation. A knowledge primitive takes a portion of a 3D shape description as input and returns a value that indicates how well it satisfies specific requirements. The six GRUFF knowledge primitives define the concepts of:

- relative orientation
- dimensions
- proximity
- stability
- clearance
- enclosure

The *relative orientation* primitive is used to determine how well the relative orientation of two surfaces satisfies some desired relationship. For example, the top surface of the seat of a chair should be approximately perpendicular to the adjacent surface of the back. The *dimensions* primitive performs dimensions tests for six possible dimension types: width, depth, height, area, contiguous_surface, and volume. In most objects the dimensions of one part of the object constrain the dimensions of the other parts. The *proximity* primitive checks for qualitative spatial relationships between elements of an object's shape. For example, the handle of a pitcher must be situated above the average center of mass of an object to make it easy to lift.

The stability primitive checks that a given shape is stable when placed on a supporting plane in a given orientation and with a possible force applied. The *clearance* primitive checks whether a specified volume of space between parts of the object is clear of obstructions. For example, a rectangular volume above the seat must be clear for a person to sit on it. Finally, the *enclosure* primitive tests for required concavities of the object. A wine goblet, for instance, must have a concavity to hold the wine.

Functional Properties The definition of a functional object class specifies the functional properties it must have in terms of the Knowledge Primitives. The GRUFF functional categories that have been used for objects in the classes *furniture*, *dishes*, and *handtools* are defined by four possible templates:

- provides stable X
- provides X surface
- provides X containment
- provides X handle

where X is a parameter of the template. For example, a chair must provide stable support and a sittable surface for the person who will sit on it. A soup bowl must provide stable containment for the soup it will contain. A cup must contain a suitable handle that allows it to be picked up and fits the dimensions of its body.

The Category Hierarchy GRUFF groups all categories of objects into a category tree that lists all the categories the system can currently recognize. At the top level of the tree are very generic categories such as furniture and dishes. Each succeeding level goes into more detail; for example, furniture has specific object classes chair, table, bench, bookshelf, and bed. Even these object classes can be divided further; chairs can be conventional chairs, lounge chairs, balans chairs, and highchairs, etc. Figure 14.36 shows portions of the GRUFF category definition tree.

Rather than "recognizing an object," GRUFF uses the function-based definition of an object category to reason about whether an observed object instance (in range data) can function as a member of that category. There are two main stages of this function-based analysis process: the pre-processing stage and the recognition stage. The preprocessing stage is category independent; all objects are processed in the same manner. In this stage the 3D data is analyzed and all potential functional elements are enumerated. The recognition stage uses these elements to construct indexes that are used to rank order the object categories. An index consists of a functional element plus its area and volume. Those categories that would be impossible to match based on the index information are pruned from the search. The others are rank ordered for further evaluation. For each class hypothesis, first each of its knowledge primitives is invoked to measure how well a functional element from the data fits its requirements. Each knowledge primitive returns an evaluation measure. These are then combined to form a final association measure that describes how well the whole set of function elements from the data matches the hypothesized object category. Figure 14.37 shows a typical input to the GRUFF system, and Figure 14.38 shows a portion of the functional reasoning in the analysis of that data.

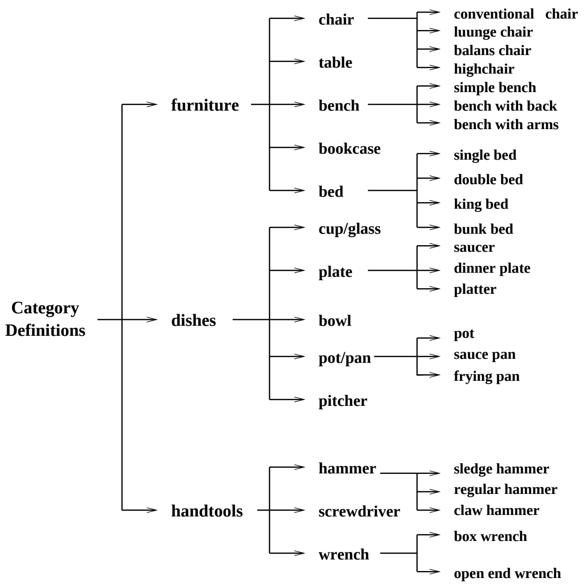
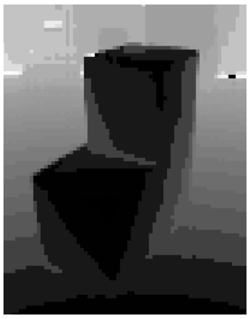
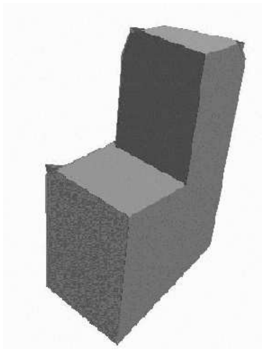


Figure 14.36: Portions of the GRUFF Category Definition Tree. (Courtesy of Louise Stark and Kevin Bowyer.)



range image



segmented range image

Figure 14.37: Input data for the GRUFF system. (Courtesy of Louise Stark and Kevin Bowyer.)

Exercise 11 Functional object recognition

Consider two tables: one with 4 legs at the 4 corners and the other having a pedestal. What similarities between these two tables would a functional object recognition system use to classify them both as the same object?

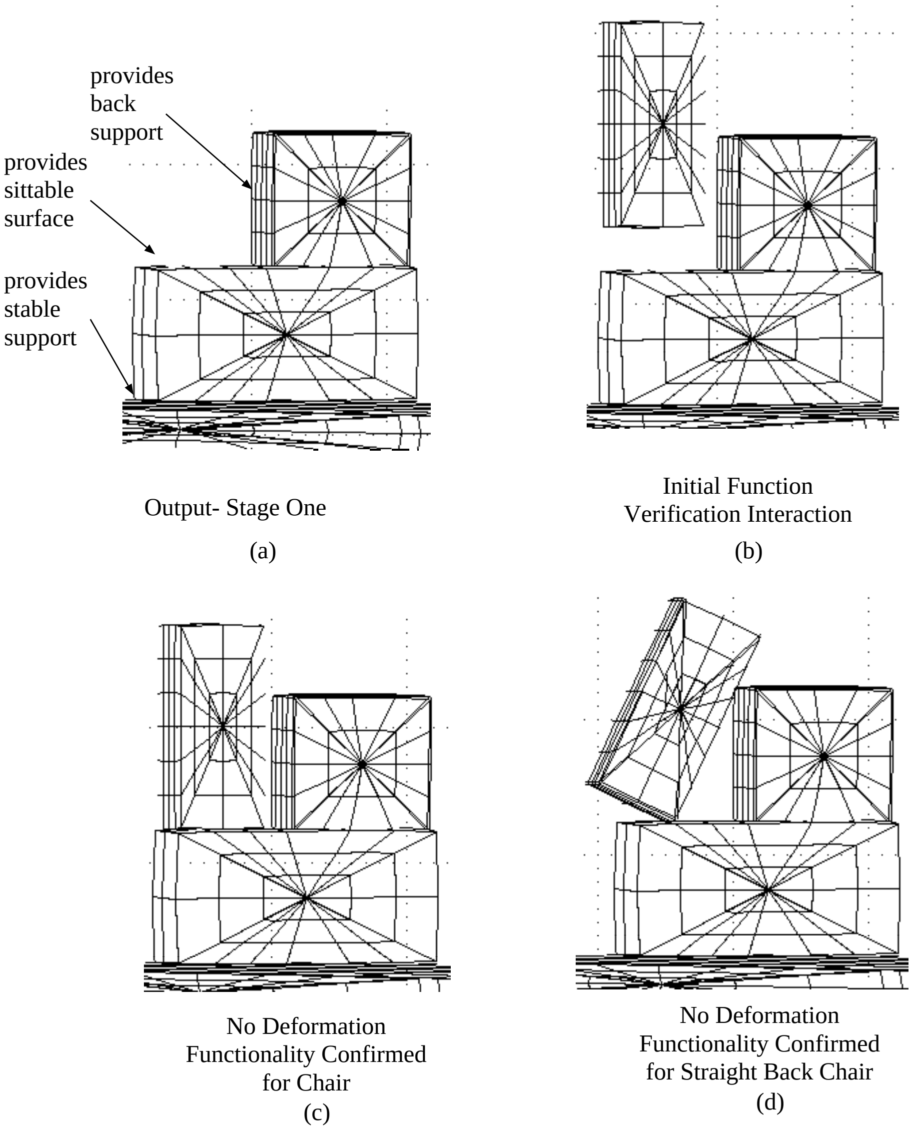


Figure 14.38: Processing by the GRUFF system. (Courtesy of Louise Stark and Kevin Bowyer.)

14.4.4 Recognition by Appearance

In most 3D object recognition schemes, the model is a separate entity from the 2D images of the object. Here we examine the idea that an object can be learned by memorizing a number of 2D images of it: recognition is performed by matching the sensed image of an unknown object to images in memory. Object representation is kept at the *signal level*; matching is done by directly comparing intensity images. Higher level features, possibly from parts extraction, are not used, and thus possibly time-consuming and complex programming that is difficult to test is not needed. Several problems with this signal level approach are addressed below. The simplicity of appearance-based recognition methods have allowed them to be trained and tested on large sets of images and some impressive results have been obtained. Perhaps, the most important results have been obtained with human face recognition, which we will use here as a clarifying example.

A coarse description of recognition-by-appearance is as follows.

- During a training, or learning, phase a database of labeled images is created. $\mathbf{DB} = \{< I_j[\,], L_j >_{j=1,k}\}$ where I_j is the j-th training image and L_j is its label.
- An unknown object is recognized by comparing its image I_u to those in the database and assigning the object the label L_j of the closest training image I_j. The closest training image I_j can be defined by the minimum Euclidean distance $\| I_u[\,] - I_j[\,] \|$ or by the maximum dot product $I_u \circ I_j$, both of which were defined in Chapter 5.

There are, of course, complications in each step that must be addressed.

- Training images must be representative of the instances of objects that are to be recognized. In the case of human faces (and most other objects), training must include changes of expression, variation in lighting, and small rotations of the head in 2D and 3D.
- The object must be well-framed; the position and size of all faces must be roughly the same. Otherwise, a search of size and position parameters is needed.
- Since the method does not separate object from background, background will be included in the decisions and this must be carefully considered in training.
- Even if our images are as small as 100×100, which is sufficient for face recognition, the dimension of the space of all images is $10,000$. It is likely that the number of training samples is much smaller than this; thus, some method of *dimensionality reduction* should be used.

While continuing, the reader should consider the case of discriminating between two classes of faces — those with glasses and those without them; or, between cars with radio antennas and those without them. Can these differences be detected among all the other variations that are irrelevant?

We now focus on the important problem of reducing the number of signal features used to represent our objects. For face recognition, it has been shown that dimensionality can be reduced from 100×100 to as little as 15 yet still supporting 97% recognition rates. Chapter 5 discussed using different bases for the space of $R \times C$ images, and showed how an image could be represented as a sum of meaningful images, such as step edges, ripple, etc. It was also shown that image energy was just the sum of the squares of the coefficients when the image was represented as a linear combination of orthonormal basis images.

Basis Images for the Set of Training Images

Suppose for the present that a set of orthonormal basis images **B** can be found with the following properties.

1. $\mathbf{B} = \{F_1, F_2, \ldots, F_m\}$ with m much smaller than $N = R \times C$.

2. The average quality of representing the image set using this basis is satisfactory in the following sense. Over all the M images I_j in the training set, we have
   $I_j^m = a_{j1}F_1 + a_{j2}F2 + \ldots + a_{jm}F_m$ and
   $\sum_{j=1}^{M} (\| I_j^m - I_j \|^2 / \| I_j \|)^2 > P\%$.

I_j^m is the approximation of original image I_j using a linear combination of just the m basis images.

The top row of Figure 14.39 shows six training images from one of many individuals in the database made available by the Weizmann Institute. The middle row of the figure shows four basis images that have been derived for representing this set of faces; the leftmost is the mean of all training samples. The bottom row of the figure shows how the original six face images would appear when represented as a linear combination of just the four basis vectors. Several different research projects have shown that perhaps $m = 15$ or $m = 20$ basis images are sufficient to represent a database of face images (e.g. 3000 face images in one of Pentland's studies), so that the average approximation of I_j^m is within 5% of I_j. Therefore, matching using the approximation will yield almost the same results as matching using the original image. It is important to emphasize that for the database illustrated by Figure 14.39, **each training image can be represented in memory by only four numbers**, which enables efficent comparison with unknown images. Provided that the four basis vectors are saved in memory, a close approximation to the original face image can be regenerated when needed. (Note that the first basis vector is the mean of the original face set and not actually one of the orthonormal set.)

Computing the Basis Images

Existence of the basis set **B** allows great compression in memory and speedup in computations because m is much smaller than N, the number of pixels in the original image. The basis images F_i are called *principal components* of the set of training samples. Algorithm 4 below sketches recognition-by-appearance using these principal components. It has two parts: an offline training phase and an online recognition phase. The first step in the training phase is to compute the mean of the training images and use them to produce a set Φ of difference images, each being the difference of a training image from the mean image. If we think of each difference image Φ_i as a vector of N elements, Φ becomes an array of N rows and M columns. The next step is to compute the covariance matrix Σ_Φ of the training images. By definition, $\Sigma_\Phi[i,i]$ is the variance of the ith pixel, while $\Sigma_\Phi[i,j]$ is the covariance of the ith and jth pixels, over all the training images. Since we have already computed the

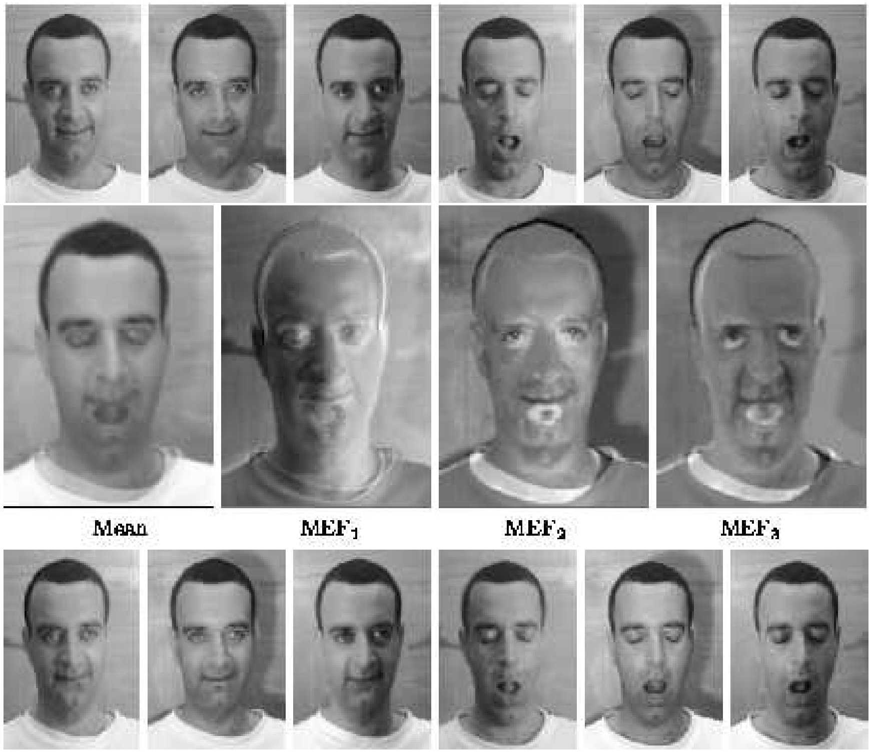


Figure 14.39: (Top row) Six training images from one of many individuals in a face image database; (middle row) average training image and three most significant eigenvectors derived from the scatter matrix; (bottom row) images of the top row represented as a linear combination of only the four images in the middle row. (Database of images courtesy of The Weizmann Institute; processed images courtesy of John Weng.)

mean and difference images, the covariance matrix is defined by

$$\Sigma_\Phi = \Phi^T \Phi \tag{14.5}$$

The size of this covariance matrix is very large, $N \times N$, where N is the number of pixels in an image, typically 256×256 or even 512×512. So the computation of eigenvectors and eigenvalues in the next step of the algorithm would be extremely time-consuming if Σ_Φ were used directly. (See *Numerical Recipes in C* for the principal components algorithm.) Instead, we can compute a related matrix Σ'_Φ given by

$$\Sigma'_\Phi = \Phi\Phi^T \tag{14.6}$$

which is much smaller ($M \times$M). The eigenvectors and eigenvalues of Σ'_Φ are related to those of Σ_Φ as follows:

$$\Sigma_\Phi F = \lambda F \tag{14.7}$$

$$\Sigma'_\Phi F' = \lambda F' \tag{14.8}$$

$$F = \Phi^T F' \tag{14.9}$$

where λ is the vector of eigenvalues of Σ_Φ, F is the vector of eigenvectors of Σ_Φ, and F' is the vector of eigenvectors of Σ'_Φ.

The methods of principal components analysis discussed here have produced some impressive results in face recognition (consult the references by Kirby and Sirovitch, Turk and Pentland, and Swets and Weng). Skeptics might argue that this method is unlikely to work for images with significant high frequency variations because autocorrelation will drop fast with small shifts in the image thus stressing the object framing requirement. Picture functions for faces do not face this problem. Swets and Weng have shown good results with many (untextured) objects other than faces as have Murase and Nayar, who were actually able to interpolate 3D object pose to an accuracy of two degrees using a training base of images taken in steps of ten degrees.

Turk and Pentland gave solutions to two of the problems noted above. First, they used motion techniques, as in Chapter 9, to segment the head from a video sequence — this enabled them to frame the face and also to normalize image size. Secondly, they reweighted the image pixels by filtering with a broad Gaussian that dropped the peripheral background pixels to near zero while preserving the important center face intensities.

Offline Training Phase:
Input a set I of M labeled training images and produce a basis set B and a vector of coefficents for each image.

$I = \{I_1, I_2, \ldots, I_M\}$ is the set of training images. (input)
$B = \{F_1, F_2, \ldots, F_m\}$ is the set of basis vectors. (output)
$A_j = [a_{j1}, a_{j2}, \ldots, a_{jm}]$ is the vector of coefficients for image I_j. (output)

1. $I_{mean} = mean(I)$.
2. $\Phi = \{\Phi_i | \Phi_i = I_i - I_{mean}\}$, the set of difference images
3. Σ_Φ = the covariance matrix obtained from Φ.
4. Use the principal components method to compute eigenvectors and eigenvalues of Σ_Φ. (see text)
5. Construct the vector **B** as the basis set by selecting the most significant m eigenvectors; start from the largest eigenvalue and continue in decreasing order of the eigenvalues to select the corresponding eigenvectors.
6. Represent each training image I_j by a linear combination of the basis vectors: $I_j^m = a_{j1}F_1 + a_{j2}F_2 + \ldots + a_{jm}F_m$

Online Recogniton Phase:
Input the set of basis vectors B, the database of coefficient sets $\{A_j\}$, and a test image I_u. Output the class label of I_u.

1. Compute vector of coefficients $A_u = [a_{u1}, a_{u2}, \ldots, a_{um}]$ for I_u;
2. Find the h nearest neighbors of vector A_u in the set $\{A_j\}$;
3. Decide the class of I_u from the labels of the h nearest neighbors (possibly reject in case neighbors are far or inconsistent in labels);

Algorithm 4: Recognition-by-Appearance using a Basis of Principal Components.

Exercise 12

Obtain a set of 10 face images and 10 landscapes such that all images have the same dimensions $R \times C$. Compute the Euclidean distance between all pairs of images and display the distances in a 20×20 upper triangular matrix. Do the faces cluster close together? The landscapes? What is the ratio of the closest to farthest distance? Is the Euclidean distance promising for retrieval from an image database? Explain.

Exercise 13

Let I_u be an image of an unknown object and let $\mathbf{B} = \{< I_j, L_j >\}$ be a set of labeled training images. Assume that all images are normalized so that $\| I_j[\,] \| = 1$. (a) Show that $\| I_u - I_j \|$ is minimized when $I_u \circ I_j$ is maximized. (b) Explain why the result is not true without the assumption that all images have unit size.

Better Discrimination and Faster Search of Memory

The methods of prinipal components analysis allow a subspace of training patterns to be represented in a compact form. The basis that best represents the training data, computed as above, has been called the set of *most expressive features (MEFs)*. The work of John Weng has demonstrated that while the most expressive features *represent* the subspace of training images optimally, they need not represent well the *differences between images* in different classes. Weng introduced the use of the *most discriminating features (MDFs)*, which can be derived from discriminant analysis. MDFs focus on image variance that can differentiate objects in different classes. Figure 14.40 contrasts MEFs with MDFs. The original data coordinates are (x_1, x_2). y_1 is the direction of maximum variance and y_2 is orthogonal to y_1; thus, coordinates y_1, y_2 are MEFs. The original classes of vectors are represented by the ellipses with major and minor axes aligned with y_1, y_2. (Recall that an algorithm for finding these axes in the 2D case was first presented in Chapter 3.) Thresholds on either y_1 or y_2 do not discriminate well between the two classes. MDF axes z_1, z_2, computed from discriminant analysis, allow perfect separation of the training samples based on a threshold on z_1.

A second improvement made by Weng and his colleagues to the "eigenspace recognition-by-appearance" approach is the development of a search tree construction procedure that provides $O(log_2 S)$ search time for finding the nearest neighbors in a database of S training samples. Recall that decision trees for object classification were introduced in Chapter 4. At each decision point in the tree, an unknown image is projected onto the most discriminating subspace needed to make a decision on which branch or branches to take next. The MDFs used at different nodes in the decision tree vary with the training samples from which they were derived and are tuned to the particular splitting decisions that are needed. It is best for the interested reader to consult the references to obtain more details on this recently developed theory.

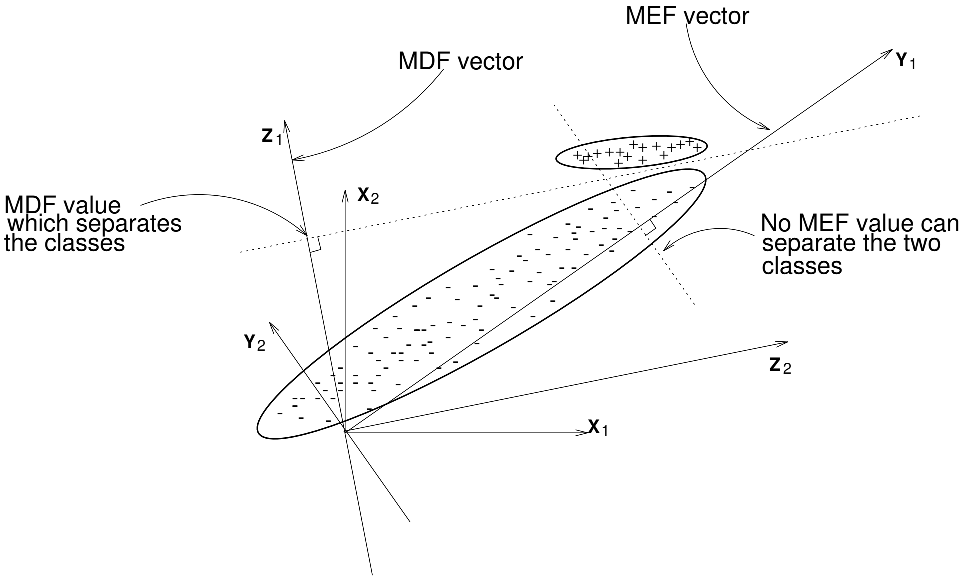


Figure 14.40: The most expressive features determined by the eignevectors of the scatter matrix represent the data well, but may not represent the differences between classes well. Discriminant analysis can be used to find subspaces that emphasize differences between classes. (Figure contributed by J. Swets and J. Weng.)

Exercise 14

(a) Obtain a set of 300 human face images and treat each one as a single vector of $R \times C$ coordinates. (b) Compute the scatter matrix and mean image from these 300 samples. (c) From the results in (b) compute the m largest eigenvalues of the scatter matrix and the corresponding m eigenvectors so that 95% of the energy in the scatter matix is represented. (d) Select 5 of the original faces at random; represent each as a linear combination of the m best eigenvectors from (c). (e) Display each of the 5 approximations derived in (d) and compare them to the original image.

14.5 References

Mesh models come from computer graphics, where they are usually called polygon meshes. The Foley *et al.* graphics text is a good reference for this subject. The surface-edge-vertex representation was introduced in the VISIONS system at University of Massachusetts in the 1970s. The structure shown in this text comes from the more recent work of Camps (1992). Generalized cylinder models were first proposed by Binford and utilized by Nevatia and Binford (1977) who worked with range data. The more recent article by Rom and Medioni discusses the computation of cylinders from 2D data. Octrees were originally proposed by Hunter (1978) and developed further by Jackins and Tanimoto (1980). They are discussed in detail in Samet's book (1990). Our discussion of superquadrics comes mostly from the work of Gupta, Bogoni, and Bajcsy (1989), and the left vetricle illustrations are from the more recent work of Park, Metaxas, and Axel (1996), which is also discussed in the section on deformable models.

The introduction of the view-class concept is generally credited to Koenderink and Van Doorn (1979). Camps (1992), Pulli (1996), and Costa (1995) used view-class models to recognize three-dimensional objects. Matching by alignment was introduced by Lowe (1987) and thoroughly analyzed by Huntenlocher and Ullmann (1990). The 3D-3D alignment discussed here comes from the work of Johnson and Hebert (1998), while the 2D-3D discussion comes from the work of Pulli (1996). The treatment of recognition-by-alignment of smooth objects was taken from the work of Jin-Long Chen (1996), and is related to the original work of Basri and Ullman (1988). Matching sticks-plates-and-blobs models was described in the work of Shapiro *et al.* (1984). Relational matching in general was discussed in Shapiro and Haralick (1981, 1985). Relational indexing can be found in the work of Costa (1995). Our discussion on functional object recognition comes from the work of Stark and Bowyer (1996).

Kirby and Sirovich (1990) approached the problem of face image compression, which Turk and Pentland (1991) then adopted for more efficient recognition of faces. Swets and Weng (1996) developed a general learning system, called SHOSLIF, which improved upon the principal components approach by using MDFs and by constructing a tree-structured database in order to search for nearest neighbors in $log_2 N$ time. Murase and Nayar (1994) also produced an efficient search method and showed that 3D object pose might be estimated to within $2°$ by interpolating training views taken at $10°$ intervals; moreover, while working with several objects other than faces, they also found that an eigenspace of dimension 20 or less was sufficient for good performance. The coverage of recognition-by-appearance in this chapter drew heavily from the work of Weng and the frequently referenced work of Turk and Pentland.

Energy minimization was used in the 70's for smoothing contours. However, the 1987 paper by Kass, Witkin and Terzopoulos in which the term *snake* was introduced, seemed to freshly ignite the research interest of many other workers. Applications to fitting and tracking surfaces and volumes quickly followed. Amini *et al* (1988) proposed dynamic programming to fit active contours to images. One of many examples of it use in medical images is Yue *et al* (1995). The works by Chen and Medioni (1995) and Park, Metaxas and Axel (1996) are two good examples of rapidly developing research and applications in physics-based and deformable modeling.

1. A. Amini, S. Tehrani and T. Weymouth (1988), *Using dynamic programming for*

minimizing the energy of active contours in the presence of hard constraints, **Proc. IEEE Int. Conf. on Computer Vision** (1988)95-99.

2. R. Basri and S. Ullman, "The Alignment of Objects with Smooth Surfaces," *Proc. of 2nd Intern. Conf. on Computer Vision*, 1988, pp. 482-488.

3. I. Biederman, "Human Image Understanding: Recent Research and Theory," *Computer Vision, Graphics, and Image Processing*, Vol. 32, No. 1, October, 1985, pp. 29-73.

4. O. I. Camps, L. G. Shapiro, and R. M. Haralick, "Image Prediction for Computer Vision," *Three-dimensional Object Recognition Systems*, A. Jain and P. Flynn (eds.), Elsevier Science Publishers BV, 1992.

5. Y. Chen and G. Medioni (1995) *Description of Complex Objects from Multiple Range Images Using an Inflating Balloon Model*, Computer Vision and Image Understanding, Vol. 61, No. 3, (May 1995)325-334.

6. J.L. Chen and G. Stockman, "Determining Pose of 3D Objects with Curved Surfaces," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 18 No. 1, 1996, pp. 57-62.

7. M. S. Costa and L. G. Shapiro, "Scene Analysis Using Appearance-Based Models and Relational Indexing," *IEEE Symposium on Computer Vision*, November, 1995, pp. 103-108.

8. J. Foley, A. van Dam, S. Feiner, J. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, MA, 1996.

9. A. Gupta, L. Bogoni, and R. Bajcsy, "Quantitative and Qualitative Measures for the Evaluation of the Superquadric Model," *Proceedings of the IEEE Workshop on Interpretation of 3D Scenes*, 1989, pp. 162-169.

10. Hunter, G. M., *Efficient Computation and Data Structures for Graphics*, Ph.D. Dissertation, Princeton University, Princeton, NJ, 1978.

11. D. P. Huttenlocher and S. Ullman, "Recognizing Solid Objects by Alignment with an Image," *International Journal of Computer Vision*, Vol. 5, No. 2, 1990, pp. 195-212.

12. C. L. Jackins and S. L. Tanimoto, "Oct-trees and Their Use in Representing Three-Dimensional Objects," *Computer Graphics and Image Processing*, Vol. 14, 1980, pp. 249-270.

13. A. E. Johnson and M. Hebert, "Efficient Multiple Model Recognition in Cluttered 3-D Scenes," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1998, pages 671-677.

14. M. Kass, A. Witkin, and D. Terzopoulos (1987) *Snakes: Active Contour Models*, Proc. First Int. Conf. on Computer Vision, Lonond, UK (1987)259-269.

15. M. Kirby and L. Sirovich, " Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 1, 1990, pp. 103-108.

16. J. J. Koenderink and A. J. Van Doorn, “The Internal Representation of Solid Shape with Respect to Vision,” *Biological Cybernetics*, Vol. 32, 1979, pp. 211-216.

17. D. G. Lowe, “The Viewpoint Consistency Constraint,” *International Journal of Computer Vision*, Vol. 1, 1987, pp. 57072.

18. H. Murase and S. Nayar (1995), “Parametric Appearance Representation,” in *3D Object Representations in Computer Vision*, J. Ponce and M. Herbert(eds), Springer-Verlag, 1995.

19. R. Nevatia and T. O. Binford, “Description and Recognition of Curved Objects,” *Artificial Intelligence*, Vol. 8, 1977, pp. 77-98.

20. J. Park, D. Metaxas and L. Axel (1996) *Analysis of Left Ventricular Wall Motion Based on Volumetric Deformable Models and MRI-SPAMM*, Medical Image Analysis Journal, Vol. 1, No. 1 (1996)pp. 53-71

21. Pulli, K. and L. G. Shapiro, “Triplet-Based Object Recognition Using Synthetic and Real Probability Models,” *Proceedings of ICPR96*, 1996, Vol. IV, pp. 75-79.

22. H. Rom and G. Medioni, “Hierarchical Decomposition and Axial Shape Description,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 10, 1993, pp. 973-981.

23. H. Samet, *Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

24. L. G. Shapiro, J. D. Moriarty, R. M. Haralick, and P. G. Mulgaonkar, ”Matching Three-Dimensional Objects Using a Relational Paradigm,” *Pattern Recognition,* Vol. 17, No. 4, 1984, pp. 385-405.

25. L. G. Shapiro and R. M. Haralick, “Structural Descriptions and Inexact Matching”, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-3, No. 5, Sept. 1981, pp. 504-519.

26. L. G. Shapiro, and R. M. Haralick, ” A Metric for Comparing Relational Descriptions”, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-7, No. 1, Jan. 1985, pp. 90-94.

27. L. Stark and K. Bowyer, *Generic Object Recognition Using Form and Function*, World Scientific Publishing Co. Pte Ltd, Singapore, 1996.

28. D. Swets and J. Weng “Using Discriminant Eigenfeatures for Image Retrieval,” *IEEE Trans. Pattern Analysis and Machien Intelligence*, Vol. 18, 1996, pp. 831-836.

29. M. Turk and A. Pentland “Eigenfaces for Recognition,” *Journal of Cognitive Neuroscience*, Vol. 3, No. 1, 1991, pp. 71-86.

30. Z. Yue, A. Goshtasby and L. Ackerman (1995), *Automatic Detection of Rib Borders in Chest Radiographs*, **IEEE Trans. MEdical Imaging**, Vol. 14, No. 3, (1995)525,536.