

Fig. 13.1 A simple block stacking task.

INITIAL STATE: ON(C,A), ON(A, Floor), ON(B, Floor),
 CLEAR(C), CLEAR(B), CLEAR(Floor)

The goal state is one in which the following two assertions are true.

GOAL ASSERTIONS: ON(A,B), ON(B,C)

With only these rules, the formalization of the block stacking world yields a very “loose” semantics. (The task easily translates to sorting integers with some restrictions on operations, or to the “seriation” task of arranging blocks horizontally in order of size, or a host of others.)

Actions transform the set of assertions describing the world. For problems of realistic scale, the representation of the tree of world states is a practical problem. The issue is one of maintaining several coexisting “hypothetical worlds” and reasoning about them. This is another version of the frame problem discussed in sec. 12.1.6. One way to solve this problem is to give each assertion an extra argument, naming the hypothetical world (usually called a situation [Nilsson 1980; McCarthy and Hayes 1969]) in which the assertion holds. Then actions map situations to situations as well as introducing and changing assertions.

An equivalent way to think about (and implement) multiple, dependent, hypothetical worlds is with a tree-structured *context-oriented data base*. This idea is a general one that is useful in many artificial intelligence applications, not just symbolic planning. Such data bases are included in many artificial intelligence languages and appear in other more traditional environments as well. A context-oriented data base *acts* like a tree of data bases; at any node of the tree is a set of assertions that makes up the data base. A new data base (context) may be spawned from any context (data base) in the tree. All assertions that are true in the spawning (ancestor) context are initially true in the spawned (descendant) context. However, new assertions added in any context or deleted from it do not affect its ancestor. Thus by going back to the ancestor, all data base changes performed in the descendent context disappear.

Implementing such a data base is an interesting exercise. Copying all assertions to each new context is possible, but very wasteful if only a few changes are made in each context. The following mechanism is much more efficient. The root or initial context has some set of assertions in it, and each descendant context is merely an *add list* of assertions to add to the data base and a *delete list* of assertions to delete. Then to see if an assertion is true in a context, do the following.

1. If the context is the root context, look up “as usual.”
2. Otherwise, if the assertion is on the *add list* of this context, return *true*. If the assertion is on the *delete list* of this context, return *false*.

3. Otherwise, recursively apply this procedure to the ancestor of this context.

In a general programming environment, contexts have names, and there is the facility of executing procedures “in” particular contexts, moving around the context tree, and so forth. However, in what follows, only the ability to look up assertions in contexts is relevant.

13.1.2 Representing Actions

Represent an action as a triple.

ACTION ::= [PATTERN, PRECONDITIONS, POSTCONDITIONS].

Here the pattern gives the name of the action and names for the objects with which it deals—its “formal parameters.” Preconditions and postconditions may use the formal variables of the pattern. In a sense, the preconditions and postconditions are the “body” of the action, with subroutine-like “variable bindings” taking place when the action is to be performed. The preconditions give the world states in which the action may be applied. Here the preconditions are assumed simply to be a list of assertions all of which must be true. The postconditions describe the world state that results from performing the action. The context-oriented data base of hypothetical worlds can be used to implement the postconditions.

POSTCONDITIONS ::= [ADD-LIST, DELETE-LIST].

An action is then performed as follows.

1. Bind the pattern variables to entities in the world, thus binding the associated variables in the preconditions and postconditions.
2. If the preconditions are met (the bound assertions exist in the data base), do the next step, else exit reporting failure.
3. Delete the assertions in the delete list, add those in the add list, and exit reporting success.

Here is the *Move* action for our block-stacking example.

<i>Move Object X from Y to Z</i>			
<i>PATTERN</i>	<i>PRECONDITIONS</i>	<i>DELETE-LIST</i>	<i>ADD-LIST</i>
Move(X,Y,Z)	CLEAR(X) CLEAR(Z) ON(X,Y)	ON(X,Y) CLEAR(Z)	ON(X,Z) CLEAR(Y)

Here *X*, *Y*, and *Z* are all variables bound to world entities. In the initial state of Fig. 13.1, *Move(C,A,Floor)* binds *X* to *C*, *Y* to *A*, *Z* to *Floor*, and the preconditions are satisfied; the action may proceed.

However, notice two things.

1. The action given above deletes the $\text{CLEAR}(\text{Floor})$ assertion that always should be true. One must fix this somehow; putting $\text{CLEAR}(\text{Floor})$ in the add-list does the job, but is a little inelegant.
2. What about an action like $\text{Move}(C,A,C)$? It meets the preconditions, but causes trouble when the add and delete lists are applied. One fix here is to keep in the data base (“world model”) a set of assertions such as $\text{Different}(A,B)$, $\text{Different}(A,\text{Floor})$, . . . , and to add assertions such as $\text{Different}(X,Z)$ to the preconditions of Move .

Such housekeeping chores and details of axiomatization are inherent in applying basically syntactic, formal solution methods to problem solving. For now, let us assume that $\text{CLEAR}(\text{Floor})$ is never deleted, and that $\text{Move}(X,Y,Z)$ is applied only if Z is different from X and Y .

13.1.3 Stacking Blocks

In the block-stacking example, the goal is two simultaneous assertions, $\text{ON}(A,B)$ and $\text{ON}(B,C)$. One solution method proceeds by repeatedly picking a goal to work on, finding an operator that moves closer to the goal, and applying it. In this case of only one action the question is how to apply it—what to move where. This is answered by looking at the postconditions of the action in the light of the goal. The reasoning might go like this: $\text{ON}(B,C)$ can be made true if X is B and Z is C . That is possible in this state if Y is A ; all preconditions are satisfied, and the goal $\text{ON}(B,C)$ can be achieved with one action.

Part of the world state (or context) tree the planner must search is shown in Fig. 13.2, where states are shown diagrammatically instead of through sets of assertions. Notice the following things in Fig. 13.2.

1. Trying to achieve $\text{ON}(B,C)$ first is a mistake (Branch 1).
2. Trying to achieve $\text{ON}(A,B)$ first is also a mistake for less obvious reasons (Branch 2).
3. Branches 1 and 2 show “subgoal interaction.” The goals as stated are not independent. Branch 3 must be generated somehow, either through backtracking or some intelligent way of coping with interaction. It will never be found by the single-minded approach of (1) and (2). However, if $\text{ON}(C,\text{Floor})$ were one of the goal assertions, Branch 3 could be found.

Clearly, representing world and actions is not the whole story in planning. Intelligent search of the context is also necessary. This search involves subgoal selection, action selection, and action argument selection. Bad choices anywhere can mean inefficient or looping action sequences, or the generation of impossible subgoals. “Intelligent” search implies a meta-level capability: the ability of a program to reason about its own plans. “Plan critics” are often a part of sophisticated planners; one of their main jobs is to isolate and rectify unwanted subgoal interaction [Sussman 1975].

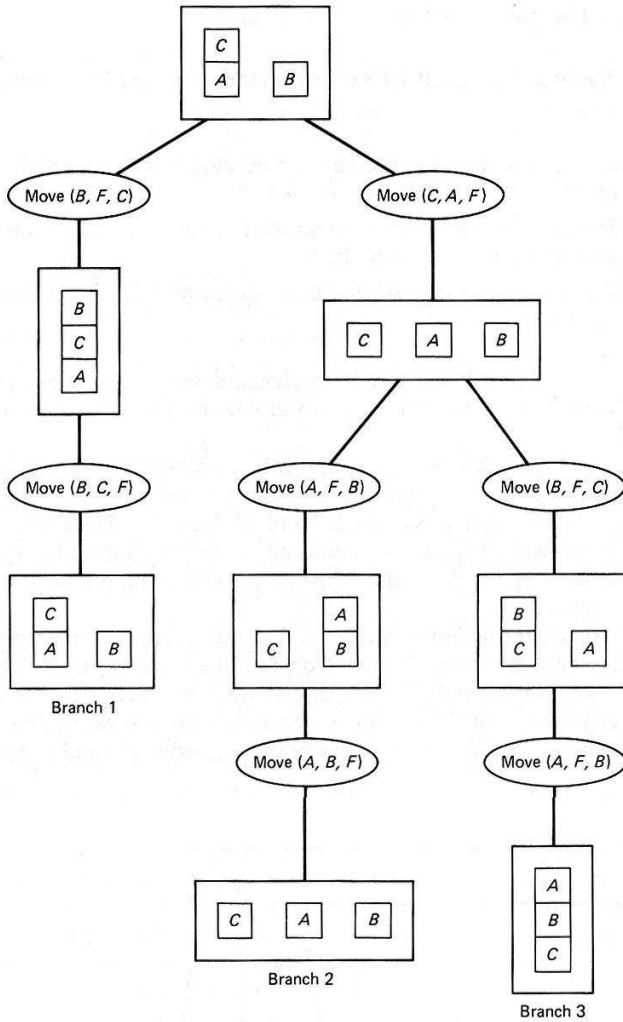


Fig. 13.2 A state tree generated in planning how to stack three blocks.

Intelligent choice of actions is the crux of planning, and is a major research issue. Several avenues have been and are being tried. Perhaps subgoals may be ordered by difficulty and achieved in that order. Perhaps planning should proceed at various levels of detail (like multiresolution image understanding), where the strategic skeleton of a plan is derived without details, then the details are filled in by applying the planner in more detail to the subgoals in the low-resolution plan.

13.1.4 The Frame Problem

All planning is plagued by aspects of the *frame problem* (introduced in Section 12.1.6).

1. It is impractical (and boring) to write down in an action all the things that stay the same when an action is applied.
2. Similarly, it is impractical to reassert in the data base all the things that remain true when an action is implied.
3. Often an action has effects that cannot be represented with simple add and delete lists.

The add and delete list mechanism and the context-oriented data base mechanism addressed the first two problems. The last problem is more troublesome.

Add and delete lists are simple ideas, whereas the world is a complex place. In many interesting cases, the add and delete lists depend on the current state of the world when the action is applied. Think of actions *TURNBY*(*X*) and *MOVEBY*(*Z*) in a world where orientation and location are important. The orientation and location after an action depend not just on the action but on the state of the world just before the action.

Again, the action may have very complex effects if there are complex dependencies between world objects. Consider the problem of the “monkey and bananas,” where the monkey plans to push the box under the bananas and climb on it to reach them (Fig. 13.3). Implementation of realistically powerful add and delete lists may in fact require arbitrary amounts of deduction and computation.

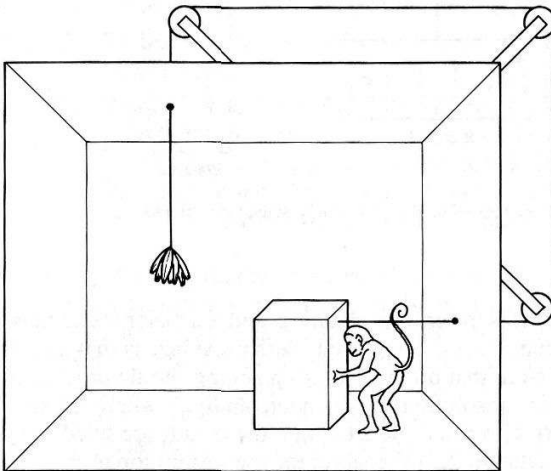


Fig. 13.3 Actions may have complex effects.

This quick précis of symbolic planning does not address many “classical” topics, such as learning or remembering useful plans. Also not discussed are: planning at varying levels of abstraction, plans with uncertain information, or plans with costs. The interested reader should consult the References for more information. The next section addresses plans with costs since they are particularly relevant to vision; some of the other issues appear in the Exercises.

13.2 PLANNING WITH COSTS

Decision making under uncertainty is an important topic in its own right, being of interest to policymakers and managers [Raiffa 1968]. Analytic techniques that can derive the strategy with the “optimal expected outcome” or “maximal expected utility” can be based on Bayesian models of probability.

In [Feldman and Sproull 1977] these techniques are explored in the context of action planning for real-world actions and vision. As an example of the techniques, they are used to model an extended version of the “monkey and bananas” problem of the last section, with multiple boxes but without the maddening pulley arrangement. In the extended problem, there are boxes of different weights which may or may not support the monkey, and he can apply tests (e.g., vision) at some cost to determine whether they are usable. Pushing weighted boxes costs some effort, and the gratification of eating the bananas is “worth” only some finite amount of effort. This extended set of considerations is more like everyday decision making in the number of factors that need balancing, in the uncertainty inherent in the universe, and in the richness of applicable tests. In fact, one might make the claim that human beings always “maximize their expected utility,” and if one knew a person’s utility functions, his behavior would become predictable. The more intuitive claim that humans beings plan only as far as “sufficient expected utility” can be cast as a maximization operation with nonzero “cost of planning.”

The sequential decision-making model of planning with the goal of maximizing the goodness of the expected outcome was used in a travel planner [Sproull 1977]. Knowledge of schedules and costs of various modes of transportation and the attendant risks could be combined with personal prejudices and preferences to produce an itinerary with the maximum expected utility. If unexpected situations (canceled flights, say) arose *en route*, replanning could be initiated; this incremental plan ramification is a natural extension of sequential decision making.

This section is concerned with measuring the expected performance of plans using a single number. Although one might expect one number to be inadequate, the central theorem of decision theory [DeGroot 1970] shows essentially that one number is enough. Using a numerical measure of goodness allows comparisons between normally incomparable concepts to be made easily. Quite frequently numerical scores are directly relevant to the issues at stake in planning, so they are not obnoxiously reductionistic. Decision theory can also help in the process of applying a plan—the basic plan may be simple, but its application to the world may be complex, in terms of when to declare a result established or an action unsuccessful. The decision-theoretic approach has been used in several artificial intelligence and

vision programs [Feldman and Yakimovsky 1974; Bolles 1977; Garvey 1976; Ballard 1978; Sproull 1977].

13.2.1 Planning, Scoring, and Their Interaction

For didactic purposes, the processes of plan generation and plan scoring are considered separately. In fact, these processes may cooperate more or less intimately. The planner produces “sequences” of *actions* for evaluation by the scorer. Each action (computation, information gathering, performing a real-world action) has a *cost*, expressing expenditure of resources, or associated unhappiness. An action has a set of possible *outcomes*, of which only one will really occur when the action is performed. A *goal* is a state of the world with an associated “happiness” or *utility*. For the purposes of uniformity and formal manipulation, goals are treated as (null) actions with no outcomes, and negative utilities are used to express costs. Then the plan has only actions in it; they may be arranged in a strict sequence, or be in loops, be conditional on outcomes of other actions, and so forth.

The *scoring* process evaluates the *expected utility* of a plan. In an uncertain world, a plan prior to execution has only an expected goodness—something might go wrong. Such a scoring process typically is not of interest to those who would use planners to solve puzzles or do proofs; what is interesting is the result, not the effort. But plans that are “optimal” in some sense are decidedly of interest in real-world decision making. In a vision context, plans are usually useful only if they can be evaluated for efficiency and efficacy.

Scoring can take place on “complete” plans, but it can also be used to guide plan generation. The usual artificial intelligence problem-solving techniques of progressive deepening search and branch-and-bound pruning may be applied to planning if scoring happens as the plan is generated [Nilsson 1980]. Scoring can be used to assess the cost of planning and to monitor planning horizons (how far ahead to look and how detailed to make the plan). Scoring will penalize plans that loop without producing results. Plan improvements, such as replanning upon failure, can be assessed with scores, and the contribution of additional steps (say for extra information gathering) can be assessed dynamically by scoring. Scoring can be arbitrarily complex utility functions, thus reflecting such concepts as “risk aversion” and nonlinear value of resources [Raiffa 1968].

13.2.2 Scoring Simple Plans

Scoring and an Example

A *simple plan* is a tree of nodes (there are no loops). The nodes represent actions (and goals). Outcomes are represented by labeled arcs in the tree. A probability of occurrence is associated with each possible outcome; since exactly one outcome actually occurs per action, the probabilities for the possible outcomes of any action sum to unity.

The *score* of a plan is its *expected utility*. The expected utility of any node is recursively defined as its utility times the probability of reaching that node in the

plan, plus the expected utilities of the actions at its (possible) outcomes. The probability of reaching any “goal state” in the plan is the product of probabilities of outcomes forming a path from the root of the plan to the goal state.

As an example, consider the plan shown in Fig. 13.4. If the plan of Fig. 13.4

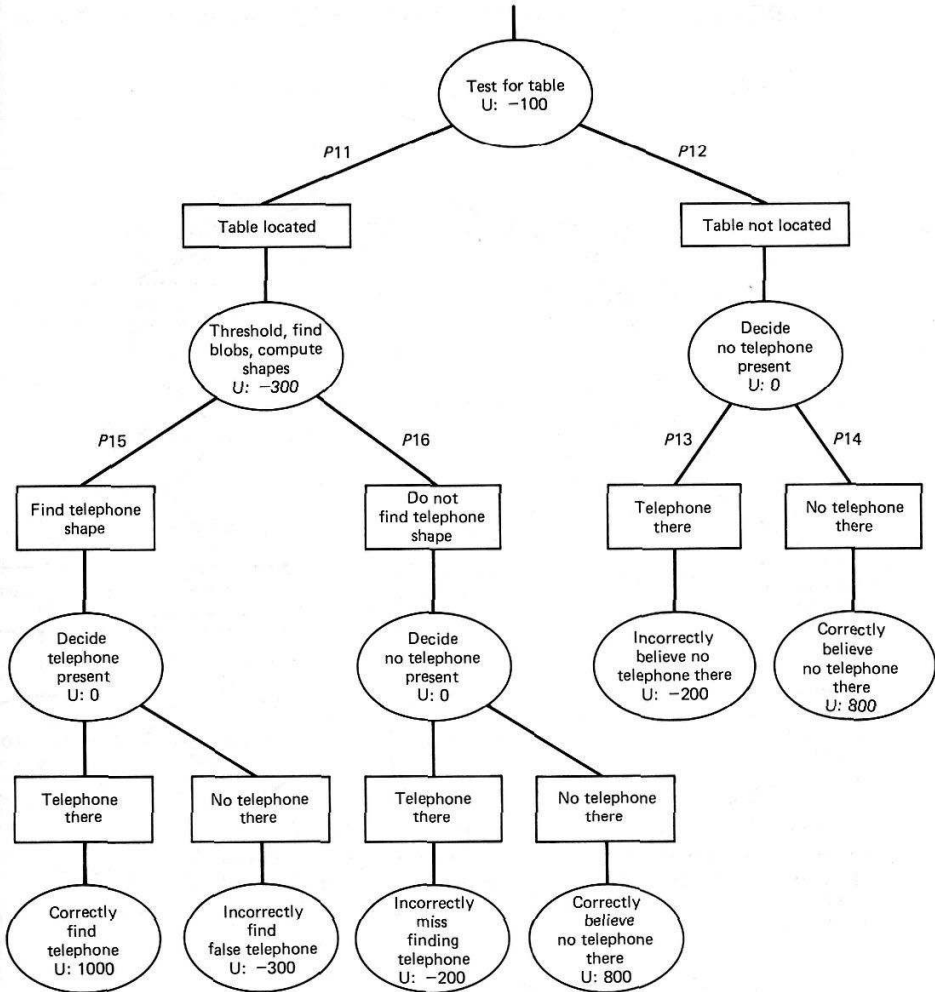


Fig. 13.4 This plan to find a telephone in an office scene involves finding a table first and looking there in more detail. The actions and outcomes are shown. The probabilities of outcomes are assigned symbols (P_{10} , etc.). Utilities (denoted by U ;) are given for the individual actions. Note that negative utilities may be considered costs. In this example, decision-making takes no effort, image processing costs vary, and there are various penalties and rewards for correct and incorrect finding of the telephone.

has probabilities assigned to its outcomes, we may compute its expected utility. Figure 13.5 shows the calculation. The probability of correctly finding the telephone is 0.34, and the expected utility of the plan is 433.

Although the generation of a plan may not be easy, scoring a plan is a trivial exercise once the probabilities and utilities are known. In practice, the assignment of probabilities is usually a source of difficulty. The following is an example using

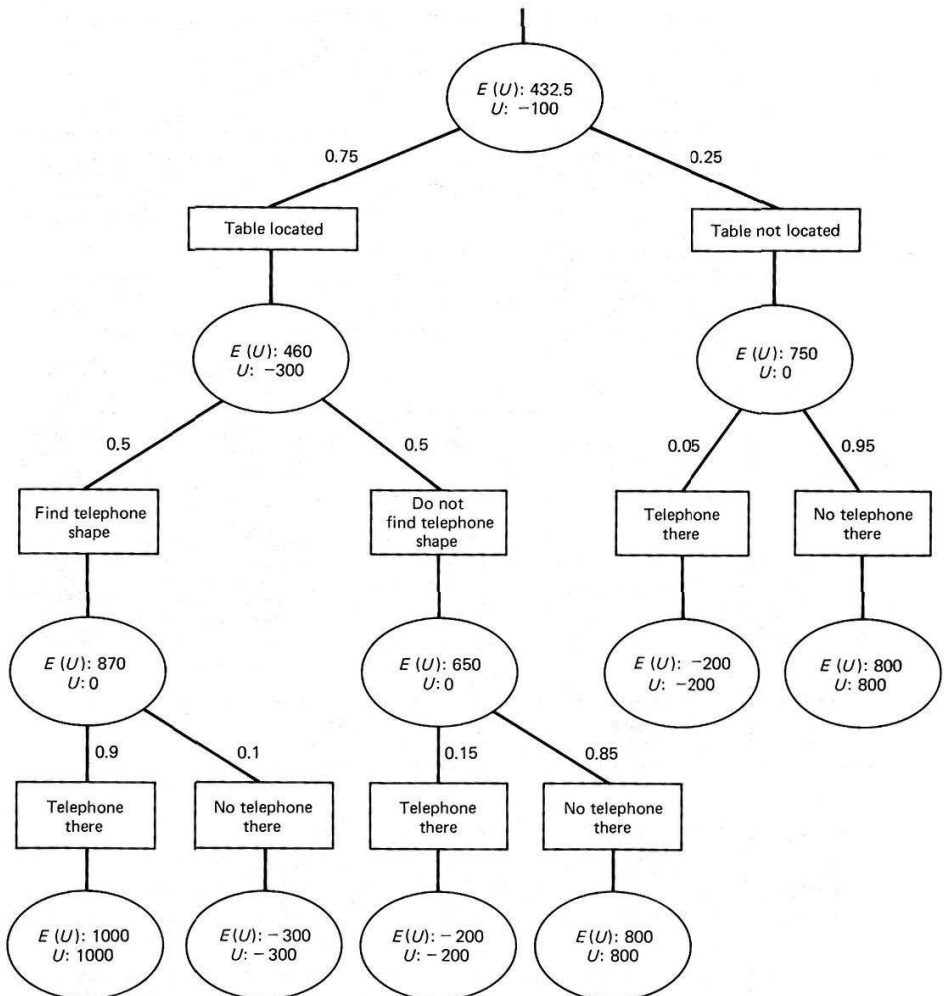


Fig. 13.5 As for Fig. 13.4. U: gives the utility of each action. E(U): gives the expected utility of the action, which depends on the outcomes below it. Values for outcome probabilities are given on the outcome arcs.

the telephone-finding plan and some assumptions about the tests. Different assumptions yield different scores.

Computing Outcome Probabilities: An Example

This example relies heavily on Bayes' rule:

$$P(B|A)P(A) = P(A \wedge B) = P(A|B)P(B). \quad (13.1)$$

Let us assume a specific a priori probability that the scene contains a telephone.

$$P_1 = \text{a priori probability of Telephone} \quad (13.2)$$

Also assume that something is known about the behavior of the various tests in the presence of what they are looking for. This knowledge may accrue from experiments to see how often the table test found tables when telephones (or tables) were and were not present. Let us assume that the following are known probabilities.

$$P_3 = P(\text{table located}|\text{telephone in scene}) \quad (13.3)$$

$$P_5 = P(\text{table located}|\text{no telephone in scene}) \quad (13.4)$$

Either there is a telephone or there is not, and a table is located or it is not, so

$$P_2 = \text{a priori probability of no telephone} = 1 - P_1 \quad (13.5)$$

$$P_4 = P(\text{no table located}|\text{telephone in scene}) = 1 - P_3 \quad (13.6)$$

$$P_6 = P(\text{no table located}|\text{no telephone in scene}) = 1 - P_5 \quad (13.7)$$

Similarly with the "shape test" for telephones: assume probabilities

$$P_7 = P(\text{telephone shape located}|\text{telephone}) \quad (13.8)$$

$$P_9 = P(\text{telephone shape located}|\text{no telephone}) \quad (13.9)$$

with

$$P_8 = 1 - P_7, \quad P_{10} = 1 - P_9 \quad (13.10)$$

as above.

There are a few points to make: First, it is not necessary to know exactly these probabilities in order to score the plan; one could use related probabilities and Bayes' rule. Other useful probabilities are of the form

$$P(\text{telephone}|\text{telephone shape located}).$$

In some systems [Garvey 1976] these are assumed to be available directly. This section shows how to derive them from known conditional probabilities that describe the behavior of detectors given certain scene phenomena.

Second, notice the assumption that although both the outcome of the table test and the shape test depend on the presence of telephones, they are taken to be independent of each other. That is, having found a table tells us nothing about the likelihood of finding a telephone shape. Independence assumptions such as this are

useful to limit computations and data gathering, but can be somewhat unrealistic. To account for the dependence, one would have to measure such quantities as

$$P(\text{telephone shape found} | \text{table located}).$$

Now to compute some outcome probabilities: Consider the probability

$$P_{11} = P(\text{table located}) \quad (13.11)$$

Let us write

TL for Table Located

TNL for Table Not Located.

A table may be located whether or not a telephone is in the scene. In terms of known probabilities, Bayes' rule yields

$$P_{11} = P_3 P_1 + P_5 P_2 \quad (13.12)$$

Then

$$P_{12} = P(\text{TNL}) = 1 - P_{11} \quad (13.13)$$

Calculating P_{13} shows a neat trick using Bayes' Rule:

$$P_{13} = P(\text{telephone} | \text{TNL}) \quad (13.14)$$

That is, P_{13} is the probability that there is a telephone in the scene given that search for a table was unsuccessful. This probability is not known directly, but

$$\begin{aligned} P_{13} &= \frac{P(\text{telephone and TNL})}{P(\text{TNL})} \\ &= \frac{P(\text{TNL and telephone})}{P_{12}} \\ &= \frac{[P(\text{TNL} | \text{telephone})P(\text{telephone})]}{P_{12}} \\ &= \frac{[P_4 P_1]}{P_{12}} \end{aligned} \quad (13.15)$$

Then, of course

$$P_{14} = 1 - P_{13} \quad (13.16)$$

Reasoning in this way using the conditional probabilities and assumptions about their independence allows the completion of the calculation of outcome probabilities (see the Exercises). One possibly confusing point occurs in calculation of P_{15} , which is

$$P_{15} = P(\text{telephone shape found} | \text{table located}) \quad (13.17)$$

By assumption, these events are only indirectly related. By the simplifying assumptions of independence, the shape operator and the table operator are independent in their operation. (Such assumptions might be false if they used common image processing subroutines, for example.) Of course, the probability of success of each

depends on the presence of a telephone in the scene. Therefore their performance is linked in the following way (see the Exercises). (Write TSL for Telephone Shape Located.)

$$P_{15} = P(\text{TSL}|\text{TL})P(\text{TSL}|\text{telephone})P(\text{telephone}|\text{TL}) \quad (13.18) \\ + P(\text{TSL}|\text{no telephone})P(\text{no telephone}|\text{TL})$$

13.2.3 Scoring Enhanced Plans

The plans of Section 13.2.2 were called “simple” because of their tree structure, complete ordering of actions, and the simple actions of their nodes. With a richer output from the symbolic planner, the plans may have different structure. For example, there may be *OR* nodes, any one of whose sons will achieve the action at the node; *AND* nodes, all of which must be satisfied (in any order) for the action to be satisfactorily completed; *SEQUENCE* nodes, which specify a set of actions and a particular order in which to achieve them. The plan may have loops, shared subgoal structure, or goals that depend on each other. How enhanced plans are interpreted and executed depends on the scoring algorithms, the possibilities of parallel execution, whether execution and scoring are interleaved, and so forth. This treatment ignores parallelism and limits discussion to expanding enhanced plans into simple ones.

It should be clear how to go about converting many of these enhanced plans to simple plans. For instance, sequence nodes simply go to a unique path of actions. Alternatively, depending on assumptions about outcomes of such actions (say whether they can fail), they may be coalesced into one action, as was the “threshold, find blobs, and compute shapes” action in the telephone-finding plan.

Rather more interesting are the *OR* and *AND* nodes, the order of whose subgoals is unspecified. Each such node yields many simple plans, depending on the order in which the subgoals are attacked. One way to score such a plan is to generate all possible simple plans and score each one, but perhaps it is possible to do better. For example, loops and mutual dependencies in plans can be dealt with in various ways. A loop can be analyzed to make sure that it contains an exit (such as a branch of an *OR* node that can be executed). One can make ad hoc assumptions that the cost of execution is always more than the cost of planning [Garvey 1976], and score the loop by its executable branch. Another idea is to plan incrementally with a finite horizon, expanding the plan through some progressive deepening, heuristic search, or pruning strategy. The accumulated cost of going around a loop will soon remove it from further consideration.

Recall (Figs. 13.4 and 13.5) that the expected utility of a plan was defined as the sum of the utility of each leaf node times the probability of reaching that node. However, the utilities need not combine linearly in scoring. Different monotonic functions of utility express such different conceptions as “aversion to risk” or “gambling addiction.” These considerations are real ones, and nonlinear utilities are the rule rather than the exception. For instance, the value of money is notoriously nonlinear. Many people would pay \$5 for an even chance to win \$15; not so many people would pay \$5,000 for an even chance to win \$15,000.

One common way to compute scores based on utilities is the “cost/benefit” ratio. This, in the form “cost/confidence” ratio, is used by Garvey in his planning vision system. This measure is examined in Section 13.2.5; roughly, his “cost” was the effort in machine cycles to achieve goals, and his “confidence” approximated the probability of a goal achieving the correct outcome. The utility of correct outcomes was not explicitly encoded in his planner.

Sequential plan elaboration or partial plan elaboration can be interleaved with execution and scoring. Most practical planning is done in interaction with the world, and the plan scoring approach lends itself well to assessing such interactions. In Section 13.2.5 considers a planning vision system that uses enhanced plans and a limited replanning capability.

A thorny problem for decision making is to assess the cost of planning itself. The planning process is given its own utility (cost), and is carried only out as far as is indicated. Of course, the problem is in general infinitely recursive, since there is also the cost of assessing the cost of planning, etc. If, however, there is a known upper bound on the utility of the best achievable plan, then it is known that infinite planning could not improve it. This sort of reasoning is weaker than that needed to give the expected benefits of planning; it measures only the cost and maximum value of planning.

Another more advanced consideration is that the results of actions can be continuous and multidimensional, and discrete probabilities can be extended to probability distribution functions. Such techniques can reflect the precision of measurements.

An obviously desirable extension to a planner is a “learner,” that can abstract rules for action applicability and remember successful plans. One approach would be to derive and remember ranges of planning parameters arising during execution; a range could be associated with a rule specifying appropriate action. This problem is difficult and the subject of current research.

13.2.4 Practical Simplifications

The expected utility calculations allow plans to be evaluated in a more or less “realistic” manner. However, in order to complete the calculations certain probabilities are necessary, and many of these reflect detailed knowledge about the interaction of phenomena in the world. It is thus often impractical to go about a full-blown treatment of scoring in the style of Section 13.2.2. This section presents some possible simplifications.

Of course, in many planning problems, such as those whose costs are nil or irrelevant, or all of whose goals are equally valuable, there is no need to address utility of plans at all. Such plans are typically not concerned with expenditure of real-world or planning resources.

Independence of various probabilities is one of the most helpful and pervasive assumptions in the calculation of probabilities. An example appeared in Section 13.2.2 with the table and telephone shape detectors.

Certain information can be ignored. Garvcy [Garvcy 1976] ignores failure information. His planning parameters include the “cost” of an action (strictly nega-

tive utilities reflecting effort), the probability of the action “succeeding,” and the conditional probability that the state of the world is correctly indicated, given success. Related to ignoring some information is the assumption that certain outcomes are more reliable than others. For instance, the decision not to plan past “failure” reports means that they are assumed reliable.

Non-Bayesian rules of inference abound in planners [Shortliffe 1976]; the idea of assigning a single numerical utility score to plans is by no means the only way to make decisions.

13.2.5 A Vision System Based on Planning

Overview

This section outlines some features of a working vision system whose actions are controlled by the planning paradigm [Garvey 1976]. As with all large vision systems, more issues are addressed in this work than with the planning paradigm as a control mechanism. For one thing, the system uses multisensory input, including range and color information. An interactive facility aids in developing and testing low-level operators and “strategies” for object location. The machine-usable representation of knowledge about the objects in the scene domains and how they could be located is of course a central component.

The domain is office scenes (Fig. 13.6). For the task of locating different objects in such scenes, a “uniform strategy” is adopted. That is, the vision task is always broken down into a sequence of major goals to be performed in order. Such uniform strategies, if they are imposed on a system at all, tend to vary with different tasks, with different sensors or domain, or with different research goals.

Garvey’s uniform strategy consists of the following steps.

1. *Acquire* some pixels thought to be in the desired region (the area of scene making up the image of the desired object).

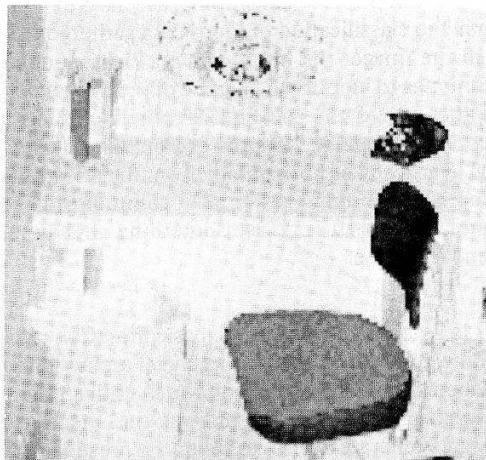


Fig. 13.6 The planning vision system uses input scenes such as these, imaged in different wavelengths and with a rangefinder.

2. *Verify* to some confidence that indeed the region was the desired one.
3. *Bound* the region accurately.

The outline the plan generation, scoring, and execution used in the system are described in the following paragraphs. The plans generated by the system are typically enhanced versions of plans like the telephone finder. Plan scoring proceeds as expected for such plans; allowances are made for the enhanced semantics of plan nodes. A “cost/confidence” scoring function is used, and various practical simplifications are made that do not affect the planning paradigm itself.

An Example Plan and Its Execution

The system’s plans are enhanced plans, in the sense of Section 13.2.3. Actions can be *AND*, *OR* or *SEQUENCE* actions, and shared plan structure and loops are permitted. Loops that contain only internal, planning actions would never terminate. However, a loop with an OR node can terminate (has an exit) if one of the subactions of the OR is executable. A plan for locating a chair in an office scene is shown in Fig. 13.7. In Fig. 13.7, the acquire-validate-bound strategy is evident in the two SEQUENCE subgoals of the Find Chair main goal, which is an AND goal. The loop in the plan is evident, and makes sense here because often planning is done for information gathering, not for real world actions.

As noted in Section 13.2.3, an enhanced plan may not be completely specified. If it is to be executed one subgoal at a time (no parallelism is allowed), sequences of subactions must be determined for its AND and OR actions. In Garvey’s planner, these sequences are determined initially on the basis of apriori information, but the partial results of actions are “fed back,” so that dynamic rescoring and hence dynamic reordering of goal sequences is possible. For example, if one subgoal of an AND action fails, the AND action is abandoned. Thus this planner is to some degree incremental.

In execution, Fig. 13.7 might result in the sequence of actions depicted in Fig. 13.8. The acquisition phase of object location has the most alternatives, so plan generation effort is mainly spent there. Acquisition proceeds either directly or indirectly. Direct acquisition is the classification of input data gathered from a random sampling of a window in the image; the input data are rich enough to allow basic pattern recognition techniques to identify the source of individual pixels.

Indirect acquisition is the use of the location of other “objects” (really identified regions) in the scene to locate the desired region. The desired region might be found by “scanning” vertically or horizontally from the already identified region, for instance. The idea is a planning version of a common one (e.g., the geometric location networks of Section 10.3.2): use something already located to limit and direct search for something else.

Plan Generation

A plan such as Fig. 13.7 is “elaborated” from the basic Find Chair goal by recursively expanding goals. Some goals (such as to find a chair) are not directly executable; they need further elaboration. Elaboration continues until all the subgoals are executable. Executable subgoals are those that analyze the image, run filters and detectors over parts of it, and generate decisions about the presence or absence

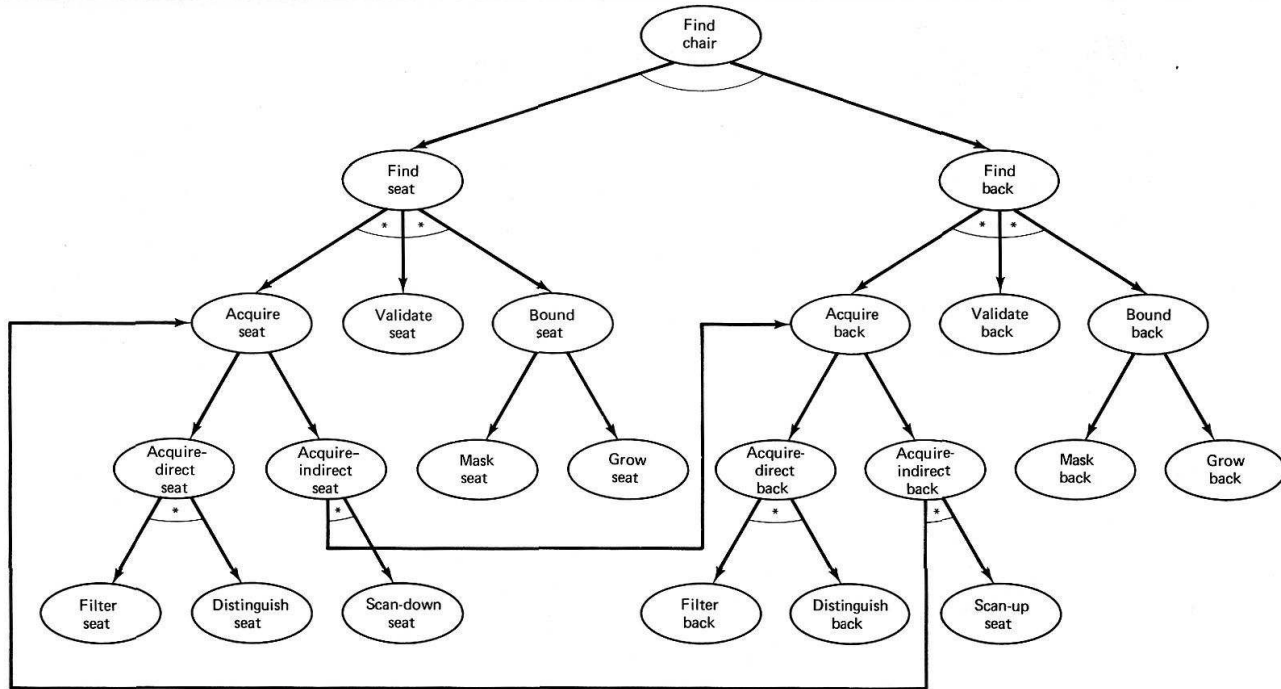
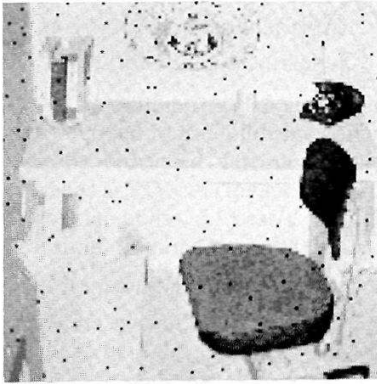
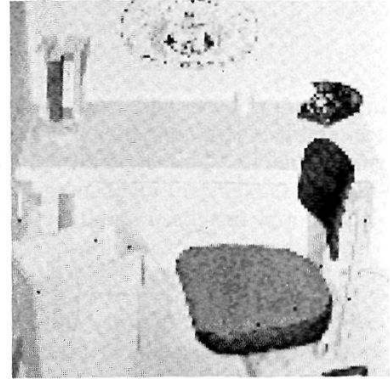


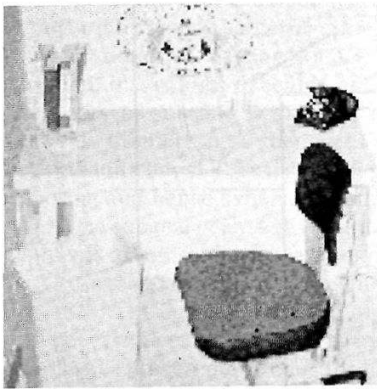
Fig. 13.7 An enhanced plan to locate a chair in an office scene. Untied multiple arcs denote OR actions, arcs tied together denote AND actions, those with '*'s denote SEQUENCE actions. The loop in the plan has executable exits.



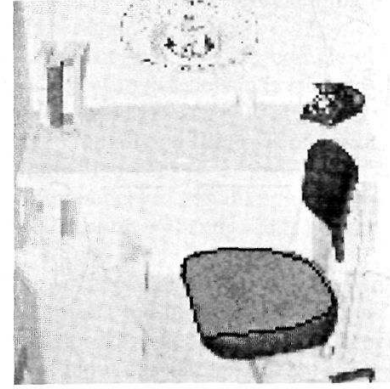
(a)



(b)

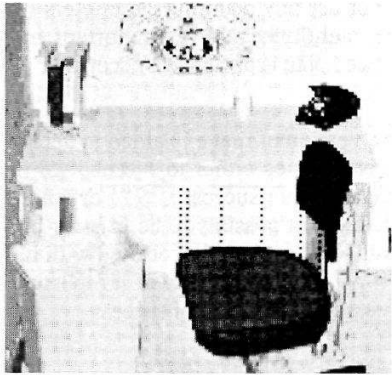


(c)

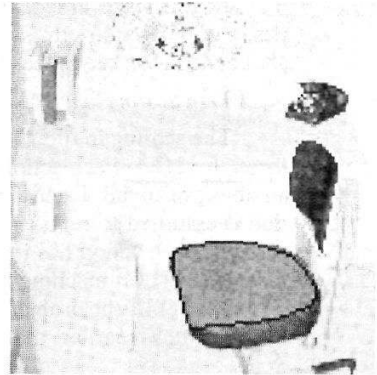


(d)

Fig. 13.8 The plan of Fig. 13.7 finds the most promising execution sequence for finding the chair in the scene of Fig. 13.6: find the seat first, then scan upwards from the seat looking for the back. Acquisition of the seat proceeds by sampling (a), followed by classification (b). The Validation procedure eliminates non-chair points (c), and the Bounding procedure produces the seat region (d). To find the back, scanning proceeds in the manner indicated by (e) (actually fewer points are examined in each scan). The back is acquired and bounded, leading to the final location of the chair regions (f).



(e)



(f)

Fig. 13.8 (cont.)

of image phenomena. This straightforward elaboration is akin to macro expansion, and is not a very sophisticated planning mechanism (the program cannot criticize and manipulate the plan, only score it). A fully elaborated plan is presented for scoring and execution.

The elaboration process, or planner, has at its disposal several sorts of knowledge embodied as modules that can generate subgoals for a goal. Some are general (to find something, find all its parts); some are less general (a chair has a back and a seat); some are quite specific, being perhaps programs arising from an earlier interactive method-generation phase. The elaborator is guided by information stored about objects, for instance this about a tabletop:

OBJECT	PROPERTIES	RELATIONS
Table TOP	Hue:26-58 Sat.: 0.23-0.32 Bright.: 18-26 Height: 26-28 Orient.: -7-7	Supports Telephone 0.6 Supports Book 0.4 Occludes Wall 1

Here the orientation information indicates a vertical surface normal. The planner knows that it has a method of locating horizontal surfaces, and the plan elaborator can thus create a goal of direct acquisition by first locating a horizontal plane. The relational information allows for indirect acquisition plans. The elaborator puts direct and indirect alternatives under an *OR* node in the plan. Information not used for acquisition (height, color) may be used for validation.

Loops may occur in an elaborated plan because each newly generated goal is checked against goals already existing. Should it or an equivalent goal already exist, the existing goal is substituted for the newly generated one. Goals may thus have more than one ancestor, and may depend on one another.

At this stage, the planner does not use any planning parameters (cost, utilities, etc.); it is strictly symbolic. As mentioned above, important information about execution sequences in an enhanced plan is provided by scoring.

Plan Scoring and Execution

The scoring in the vision plan is a version of that explained in Sections 13.2.2 through 13.2.4. Each action in a plan is assumed either to succeed (S) in locating an object or to fail. Each action may report either success (" S ") or failure. An action is assumed to report failure correctly, but possibly to be in error in reporting success. Each action has three "planning parameters" associated with it. They are C , its "cost" (in machine cycles), $P("S")$ the probability of it reporting success, and $P(S|"S")$, the probability of success given a report of success.

As shown earlier, the product

$$P(S|"S")P("S") \quad (13.19)$$

is the probability that the action has correctly located an object and reported success. This product is called the "confidence" of the action. An action has structure as shown in Fig. 13.9.

The score of an action is computed as

$$\text{score} = \frac{\text{cost}}{\text{confidence}} \quad (13.20)$$

The planner thus must minimize the score.

The initial planning parameters of an executable action typically are determined by experimentation. The parameters of internal (AND, OR, SEQUENCE) actions by scoring methods alluded to in Sections 13.2.2, 13.2.3, and the Exercises (there are a few idiosyncratic ad hoc adjustments.)

It may bear repeating that planning, scoring, and execution are not separated temporally in this system. Scoring is used after the enhanced plan is generated to derive a simple plan (with ordered subgoals). Execution can affect the scores of nodes, and so execution can alternate with "replanning" (really rescoreing resulting in a reordering). Recall the example of failure of an *AND* or *SEQUENCE* subgoal, which can immediately fail the entire goal. More generally, the entire goal and ultimately the plan may be rescored. For instance, the parameters of a successful action are modified by setting the cost of the executed action to 0 and its confidence to its second parameter, $P(S|"S")$.

Given a scored plan, execution is then easy; the execution program starts at the top goal of the plan, working its way down the best path as defined by the scores of nodes it encounters. When an executable subgoal is found (e.g. "look for a green region"), it is passed to an evaluation function that "runs" the action associated with the subgoal.

The subgoal is either achieved or not; in either case, information about its outcome is propagated back up the plan. Failure is easy; a failed subgoal of an AND or SEQUENCE goal fails the goal, and this failure is propagated. A failed subgoal of an OR goal is removed from the plan. The use of success information is more complex, involving the adjustment of confidences and planning parameters illustrated above.

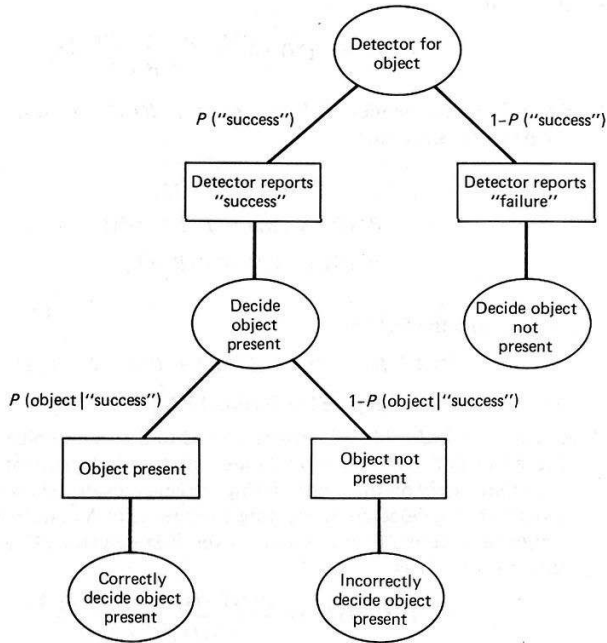


Fig. 13.9 This is the microstructure of a node ("action") of Garvey's planning system in terms of simple plans. Think of actions as being object detectors which announce "Found" or "Not Found." Garvey's planning parameters are $P(\text{"Found"})$ and $P(\text{Object is there}|\text{"Found"})$. Confidence in the action is their product; it is the probability of correctly detecting the object. All other outcomes are lumped together and not used for planning.

After the outcome of a goal is used to adjust the parameters of other goals, the plan is rescored and another cycle of execution performed. The execution can use knowledge about the image picked up along the way by prior execution. This is how results (such as acquired pixels) are passed to later processing stages (such as the validation process). Such a mechanism can even be used to remember successful subplans for later use.

EXERCISES

- 13.1 Complete the computation of outcome probabilities in the style of Section 13.2.2, using the assumptions given there. Check your work by showing (symbolically) that the probabilities of getting to the terminal actions ("goal states") of the plan sum to 1.
- 13.2 Assume in Section 13.2.2 that the results of the "table" and "telephone shape" detectors are not independent. Formulate your assumptions and compute the new outcome probabilities for Fig. 13.4.

13.3 Show that

$$P(A|(B \wedge C)) = \frac{P(B|(A \wedge C))P(A|C)}{P(B|C)}$$

13.4 B and C are independent if $P(B \wedge C) = P(B)P(C)$. Assuming that B and C are independent, show that

$$\begin{aligned} P(B|C) &= P(B) \\ P((B \wedge C)|A) &= P(B|A)P(C|A) \\ P(B|(A \wedge C)) &= P(B|A) \end{aligned}$$

13.5 Starting from the fact that

$$P(A \wedge B) = P(A \wedge B \wedge C) + P(A \wedge B \wedge (\neg C))$$

show how P_{15} was computed in Section 13.2.2.

13.6 A sequence $D(N)$ of N detectors is used to detect an object; the detectors either succeed or fail. Detector outputs are assumed independent of each other, being conditioned only on the object. Using previous results, show that the probability of an object being detected by applying a sequence of N detectors $D(N)$ is recursively rewritable in terms of the output of the first detector D_1 and the remaining sequence $D(N-1)$ as

$$P(O|D(N)) = \frac{P(D_1|O)P(O|D(N-1))}{P(D_1|D(N-1))}$$

13.7 Consider scoring a plan containing an OR node (action). Presumably, each subgoal of the OR has an expected utility. The OR action is achieved as soon as one of the subgoals is achieved. Is it possible, to order the subgoals for trial so as to maximize the expected utility of the plan? (This amounts to a unique “best” rewriting of the plan to make it a simple plan.)

13.8 Answer question 13.7 for an AND node; remember that the AND will fail as soon as any of its subgoals fails.

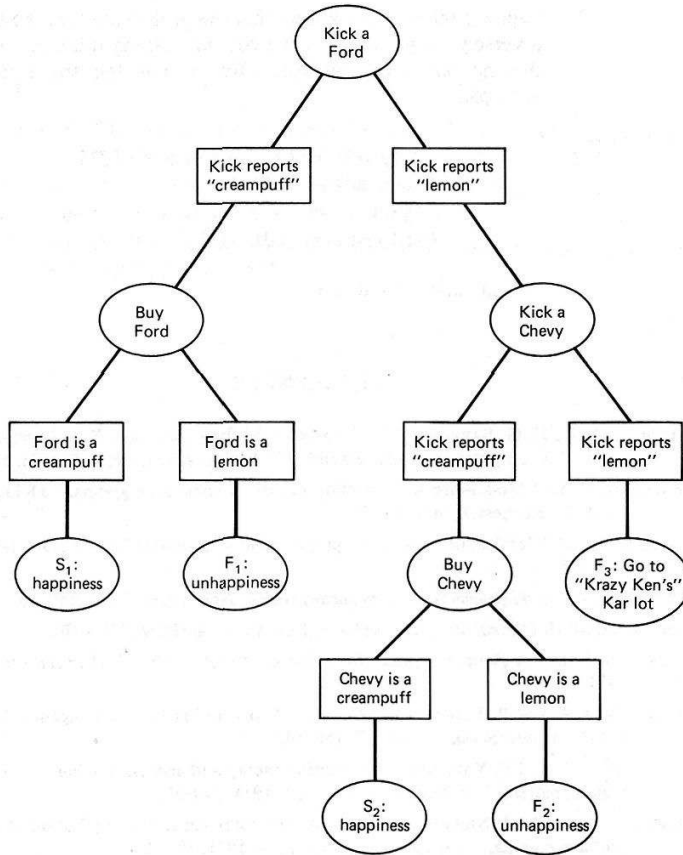
13.9 What can you say about how the cost/confidence ratio of Garvey’s planner is related to the expected utility calculations of Section 13.2.2?

13.10 You are at Dandy Dan’s used car lot. *Consumer Reports* says that the a priori probability that any car at Dandy Dan’s is a lemon is high. You know, though, that to test a car you kick its tire. In fact, with probability:

- $P(“C”|C)$: a kick correctly announces “creampuff” when the car actually is a creampuff
- $P(“C”|L)$: a kick incorrectly announces “creampuff” when the car is actually a lemon
- $P(L)$: the a priori probability that the car is a lemon

Your plan for dealing with Dandy Dan is shown below; give expressions for the probabilities of arriving at the nodes labeled S_1 , S_2 , F_1 , F_2 , and F_3 . Give numeric answers using the following values

$$P(“C”|C) = 0.5, P(“C”|L) = 0.5, P(L) = 0.75$$



Ex. 13.10

13.11 Two bunches of bananas are in a room with a monkey and a box. One of the bunches is lying on the floor, the other is hanging from the ceiling. One of the bunches is made of wax. The box may be made of flimsy cardboard. Given that:

- $P(WH) = 0.2$: probability that the hanging bananas are wax
- $P(WL) = 0.8$: probability that the lying bananas are wax
- $P(C) = 0.5$: probability that the box is cardboard
- $U(\text{eat}) = 200$: utility of eating a bunch of bananas
- $C(\text{walk}) = -10$: cost of walking a unit distance
- $C(\text{push}) = -20$: cost of pushing the box a unit distance
- $C(\text{climb}) = -20$: cost of climbing up on box

- (a) Analyze two different plans for the monkey, showing all paths and calculations. Give criteria (based upon extra information not given here) that would allow the monkey to choose between these plans.

- (b) Suppose the monkey knows that the probability that the box will collapse is inversely proportional to the cost of pushing the box a unit distance (and that he can sense this cost after pushing the box 1 unit distance). For example,

$$P(C) = 1.0 - [C(\text{push}) \times 0.01]$$

$$P(C(\text{push}) = 10) = 0.1$$

$$P(C(\text{push}) = 20) = 0.1$$

$$P(C(\text{push}) = 100) = 0.1$$

Repeat part (a) (in detail).

REFERENCES

- AMBLER, A. P., H. G. BARROW, C. M. BROWN, R. M. BURSTALL, and R. J. POPPLESTONE. "A versatile system for computer controlled assembly." *Artificial Intelligence* 6, 2, 1975, 129-156.
- BALLARD, D. H. "Model-directed detection of ribs in chest radiographs." TR11, Computer Science Dept., U. Rochester, March 1978.
- BOLLES, R. C. "Verification vision for programmable assembly." *Proc.*, 5th IJCAI, August 1977, 569-575.
- BUNDY, A. *Artificial Intelligence: An introductory course*. New York: North Holland, 1978.
- DEGROOT, M. H. *Optimal Statistical Decisions*. New York: McGraw-Hill, 1970.
- FAHLMAN, S. E. "A planning system for robot construction tasks." *Artificial Intelligence* 5, 1, Spring 1974, 1-49.
- FELDMAN, J. A. and R. F. SPROULL. "Decision theory and artificial intelligence: II. The hungry monkey." *Cognitive Science* 1, 2, 1977, 158-192.
- FELDMAN, J. A. and Y. YAKIMOVSKY. "Decision theory and artificial intelligence: I. A semantics-based region analyser." *Artificial Intelligence* 5, 4, 1974, 349-371.
- FIKES, R. E. and N. J. NILSSON. "STRIPS: A new approach to the application of theorem proving to problem solving." *Artificial Intelligence* 2, 3/4, 1971, 189-208.
- FIKES, R. E., P. E. HART, and N. J. NILSSON. "New Directions in robot problem solving." In *M17*, 1972a.
- FIKES, R. E., P. E. HART, and N. J. NILSSON. "Learning and executing generalized robot plans." *Artificial Intelligence* 3, 4, 1972b, 251-288.
- GARVEY, J. D. "Perceptual strategies for purposive vision." Technical Note 117, AI Center, SRI Int'l, 1976.
- MACKWORTH, A. K. "Vision research strategy: Black magic, metaphors, mechanisms, mini-worlds, and maps." In *CVS*, 1978.
- MCCARTHY, J. and P. J. HAYES. "Some philosophical problems from the standpoint of artificial intelligence." In *M14*, 1969.
- NILSSON, N. J. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Company, 1980.
- RAIFFA, H. *Decision Analysis*. Reading, MA: Addison-Wesley, 1968.
- SACERDOTI, E. D. "Planning in a hierarchy of abstraction spaces." *Artificial Intelligence* 5, 2, 1974, 115-135.
- SACERDOTI, E. D. *A Structure for Plans and Behavior*. New York: Elsevier, 1977.
- SHIRAI, Y. "Analyzing intensity arrays using knowledge about scenes." In *PCV*, 1975.

- SHORTLIFFE, E. H. *Computer-Based Medical Consultations: MYCIN*. New York: American Elsevier, 1976.
- SPROULL, R. F. "Strategy construction using a synthesis of heuristic and decision-theoretic methods." Ph.D. dissertation, Dept. of Computer Science, Stanford U., May 1977.
- SUSSMAN, G. J. *A Computer Model of Skill Acquisition*. New York: American Elsevier, 1975.
- TATE, A. "Generating project networks." *Proc.*, 5th IJCAI, August 1977, 888-983.
- WARREN, D. H. D. "WARPLAN: A system for generating plans." Memo 76, Dept. of Computational Logic, U. Edinburgh, June 1974.

Some Mathematical Tools

Appendix 1

A1.1 COORDINATE SYSTEMS

A1.1.1 Cartesian

The familiar two- and three-dimensional rectangular (Cartesian) coordinate systems are the most generally useful ones in describing geometry for computer vision. Most common is a right-handed three-dimensional system (Fig. A1.1.). The coordinates of a point are the perpendicular projections of its location onto the coordinate axes. The two-dimensional coordinate system divides two-dimensional space into quadrants, the three-dimensional system divides three-space into octants.

A1.1.2 Polar and Polar Space

Coordinate systems that measure locations partially in terms of angles are in many cases more natural than Cartesian coordinates. For instance, locations with respect

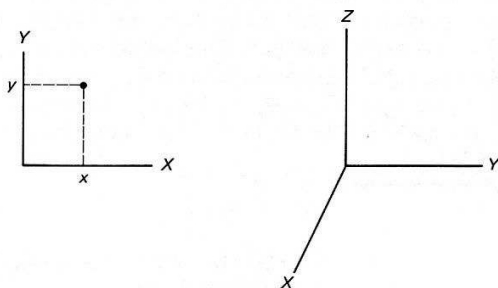


Fig. A1.1 Cartesian coordinate systems.

to the pan-tilt head of a camera or a robot arm may most naturally be described using angles. Two- and three-dimensional polar coordinate systems are shown in Fig. A1.2.

<i>Cartesian Coordinates</i>	<i>Polar Coordinates</i>
x	$\rho \cos \theta$
y	$\rho \sin \theta$
$(x^2 + y^2)^{1/2}$	ρ
$\tan^{-1} \left(\frac{y}{x} \right)$	θ

<i>Cartesian Coordinates</i>	<i>Polar Space Coordinates</i>
(x, y, z)	$(\rho \cos \xi, \rho \cos \eta, \rho \cos \zeta)$
$(x^2 + y^2 + z^2)^{1/2}$	ρ
$\cos^{-1} \left(\frac{x}{\rho} \right)$	ξ
$\cos^{-1} \left(\frac{y}{\rho} \right)$	η
$\cos^{-1} \left(\frac{z}{\rho} \right)$	ζ

In these coordinate systems, the Cartesian quadrants or octants in which points fall are often of interest because many trigonometric functions determine only an angle modulo $\pi/2$ or π (one or two quadrants) and more information is necessary to determine the quadrant. Familiar examples are the inverse angle functions (such as arctangent), whose results are ambiguous between two angles.

A1.1.3 Spherical and Cylindrical

The spherical and cylindrical systems are shown in Fig. A1.3.

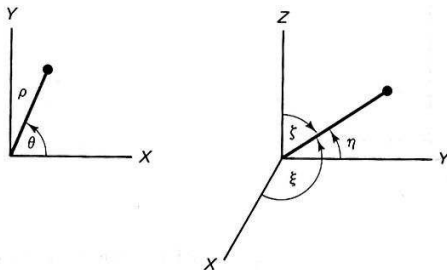


Fig. A1.2 Polar and polar space coordinate systems.

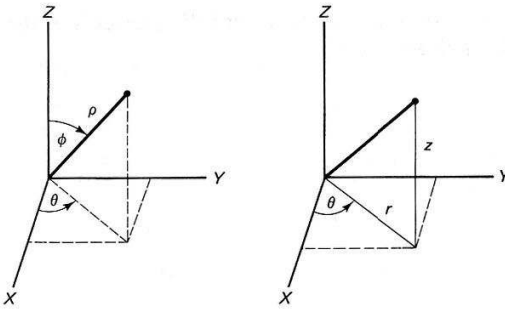


Fig. A1.3 Spherical and cylindrical coordinate systems.

<i>Cartesian Coordinates</i>	<i>Spherical Coordinates</i>
x	$\rho \sin \phi \cos \theta$
y	$\rho \sin \phi \sin \theta = x \tan \theta$
z	$\rho \cos \theta$
$(x^2 + y^2 + z^2)^{1/2}$	ρ
$\tan^{-1} \left(\frac{y}{x} \right)$	θ
$\cos^{-1} \left(\frac{z}{\rho} \right)$	ϕ

<i>Cartesian Coordinates</i>	<i>Cylindrical Coordinates</i>
x	$r \cos \theta$
y	$r \sin \theta$
z	z
$(x^2 + y^2)^{1/2}$	r
$\tan^{-1} \left(\frac{y}{x} \right)$	θ

A1.1.4 Homogeneous Coordinates

Homogeneous coordinates are a very useful tool in computer vision (and computer graphics) because they allow many important geometric transformations to be represented uniformly and elegantly (see Section A1.7). Homogeneous coordinates are redundant: a point in Cartesian n -space is represented by a line in homogeneous $(n + 1)$ -space. Thus each (unique) Cartesian coordinate point corresponds to infinitely many homogeneous coordinates.

<i>Cartesian Coordinates</i>	<i>Homogeneous Coordinates</i>
(x, y, z)	(wx, wy, wz, w)
$\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right)$	(x, y, z, w)

Here $x, y, z,$ and w are real numbers, $wx, wy,$ and wz are the products of the two reals, and x/w and so on are the indicated quotients.

A1.2. TRIGONOMETRY

A1.2.1 Plane Trigonometry

Referring to Fig. A1.4, define

$$\text{sine:} \quad \sin(A) \text{ (sometimes } \sin A) = \frac{a}{c}$$

$$\text{cosine:} \quad \cos(A) \text{ (or } \cos A) = \frac{b}{c}$$

$$\text{tangent:} \quad \tan(A) \text{ (or } \tan A) = \frac{a}{b}$$

The inverse functions arcsin, arccos, and arctan (also written $\sin^{-1}, \cos^{-1}, \tan^{-1}$) map a value into an angle. There are many useful trigonometric identities; some of the most common are the following.

$$\tan(x) = \frac{\sin(x)}{\cos(x)} = -\tan(-x)$$

$$\sin(x + y) = \sin(x) \cos(y) + \cos(x) \sin(y)$$

$$\cos(x + y) = \cos(x) \cos(y) - \sin(x) \sin(y)$$

$$\tan(x \pm y) = \frac{\tan(x) \mp \tan(y)}{1 \mp \tan(x) \tan(y)}$$

In any triangle with angles A, B, C opposite sides $a, b, c,$ the Law of Sines holds:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

as does the Law of Cosines:

$$a^2 = b^2 + c^2 - 2bc \cos A$$

$$a = b \cos C + c \cos B$$

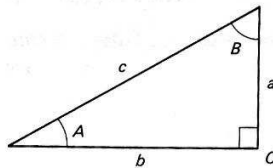


Fig. A1.4 Plane right triangle.

A1.2.2. Spherical Trigonometry

The sides of a spherical triangle (Fig. A1.5) are measured by the angle they subtend at the sphere center; its angles by the angle they subtend on the face of the sphere.

Some useful spherical trigonometric identities are the following.

$$\frac{\sin A}{\sin a} = \frac{\sin B}{\sin b} = \frac{\sin C}{\sin c}$$

$$\cos a = \cos b \cos c + \sin b \sin c \cos A = \frac{\cos b \cos (c \pm \theta)}{\cos \theta}$$

$$\text{Where } \tan \theta = \tan b \cos A,$$

$$\cos A = -\cos B \cos C + \sin B \sin C \cos a$$

A1.3. VECTORS

Vectors are both a notational convenience and a representation of a geometric concept. The familiar interpretation of a vector \mathbf{v} as a directed line segment allows for a geometrical interpretation of many useful vector operations and properties. A more general notion of an n -dimensional vector $\mathbf{v} = (v_1, v_2, \dots, v_n)$ is that of an n -tuple abiding by mathematical laws of composition and transformation. A vector may be written horizontally (a row vector) or vertically (a column vector).

A point in n -space is characterized by its n coordinates, which are often written as a vector. A point at X, Y, Z coordinates $x, y,$ and z is written as a vector \mathbf{x} whose three components are (x, y, z) . Such a vector may be visualized as a directed line segment, or arrow, with its tail at the origin of coordinates and its head at the point at (x, y, z) . The same vector may represent instead the direction in which it points—toward the point (x, y, z) starting from the origin. An important type of direction vector is the normal vector, which is a vector in a direction perpendicular to a surface, plane, or line.

Vectors of equal dimension are equal if they are equal componentwise. Vectors may be multiplied by scalars. This corresponds to stretching or shrinking the vector arrow along its original direction.

$$\lambda \mathbf{x} = (\lambda x_1, \lambda x_2, \dots, \lambda x_n)$$

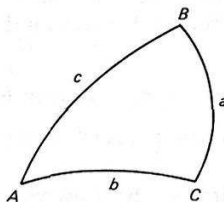


Fig. A1.5 Spherical triangle.

Vector addition and subtraction is defined componentwise, only between vectors of equal dimension. Geometrically, to add two vectors \mathbf{x} and \mathbf{y} , put \mathbf{y} 's tail at \mathbf{x} 's head and the sum is the vector from \mathbf{x} 's tail to \mathbf{y} 's head. To subtract \mathbf{y} from \mathbf{x} , put \mathbf{y} 's head at \mathbf{x} 's head; the difference is the vector from \mathbf{x} 's tail to \mathbf{y} 's tail.

$$\mathbf{x} \pm \mathbf{y} = (x_1 \pm y_1, x_2 \pm y_2, \dots, x_n \pm y_n)$$

The length (or magnitude) of a vector is computed by an n -dimensional version of Euclidean distance.

$$|\mathbf{x}| = (x_1^2 + x_2^2 + \dots + x_n^2)^{1/2}$$

A vector of unit length is a unit vector. The unit vectors in the three usual Cartesian coordinate directions have special names.

$$\mathbf{i} = (1, 0, 0)$$

$$\mathbf{j} = (0, 1, 0)$$

$$\mathbf{k} = (0, 0, 1)$$

The inner (or scalar, or dot) product of two vectors is defined as follows.

$$\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}||\mathbf{y}| \cos \theta = x_1y_1 + x_2y_2 + \dots + x_ny_n$$

Here θ is the angle between the two vectors. The dot product of two nonzero numbers is 0 if and only if they are orthogonal (perpendicular). The projection of \mathbf{x} onto \mathbf{y} (the component of vector \mathbf{x} in the direction \mathbf{y}) is

$$|\mathbf{x}| \cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{y}|}$$

Other identities of interest:

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$$

$$\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) = \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}$$

$$\lambda (\mathbf{x} \cdot \mathbf{y}) = (\lambda \mathbf{x}) \cdot \mathbf{y} = \mathbf{x} \cdot (\lambda \mathbf{y})$$

$$\mathbf{x} \cdot \mathbf{x} = |\mathbf{x}|^2$$

The cross (or vector) product of two three-dimensional vectors is defined as follows.

$$\mathbf{x} \times \mathbf{y} = (x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, x_1y_2 - x_2y_1)$$

Generally, the cross product of \mathbf{x} and \mathbf{y} is a vector perpendicular to both \mathbf{x} and \mathbf{y} . The magnitude of the cross product depends on the angle θ between the two vectors.

$$|\mathbf{x} \times \mathbf{y}| = |\mathbf{x}||\mathbf{y}| \sin \theta$$

Thus the magnitude of the product is zero for two nonzero vectors if and only if they are parallel.

Vectors and matrices allow for the short formal expression of many symbolic

expressions. One such example is the formal determinant (Section A1.4) which expresses the definition of the cross product given above in a more easily remembered form.

$$\mathbf{x} \times \mathbf{y} = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$$

Also,

$$\begin{aligned} \mathbf{x} \times \mathbf{y} &= -\mathbf{y} \times \mathbf{x} \\ \mathbf{x} \times (\mathbf{y} \pm \mathbf{z}) &= \mathbf{x} \times \mathbf{y} \pm \mathbf{x} \times \mathbf{z} \\ \lambda(\mathbf{x} \times \mathbf{y}) &= \lambda \mathbf{x} \times \mathbf{y} = \mathbf{x} \times \lambda \mathbf{y} \\ \mathbf{i} \times \mathbf{j} &= \mathbf{k} \\ \mathbf{j} \times \mathbf{k} &= \mathbf{i} \\ \mathbf{k} \times \mathbf{i} &= \mathbf{j} \end{aligned}$$

The triple scalar product is $\mathbf{x} \cdot (\mathbf{y} \times \mathbf{z})$, and is equivalent to the value of the determinant

$$\det \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix}$$

The triple vector product is

$$\mathbf{x} \times (\mathbf{y} \times \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})\mathbf{y} - (\mathbf{x} \cdot \mathbf{y})\mathbf{z}$$

A1.4. MATRICES

A matrix A is a two-dimensional array of elements; if it has m rows and n columns it is of dimension $m \times n$, and the element in the i th row and j th column may be named a_{ij} . If m or $n = 1$, a row matrix or column matrix results, which is often called a vector. There is considerable punning among scalar, vector and matrix representations and operations when the same dimensionality is involved (the 1×1 matrix may sometimes be treated as a scalar, for instance). Usually, this practice is harmless, but occasionally the difference is important.

A matrix is sometimes most naturally treated as a collection of vectors, and sometimes an $m \times n$ matrix M is written as

$$M = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_n]$$

or

$$M = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_m \end{bmatrix}$$

where the \mathbf{a} 's are column vectors and the \mathbf{b} 's are row vectors.

Two matrices A and B are equal if their dimensionality is the same and they are equal elementwise. Like a vector, a matrix may be multiplied (elementwise) by a scalar. Matrix addition and subtraction proceeds elementwise between matrices of like dimensionality. For a scalar k and matrices A , B , and C of like dimensionality the following is true.

$$A = B \pm C \quad \text{if } a_{ij} = b_{ij} \pm c_{ij} \quad 1 \leq i \leq m, \quad 1 \leq j \leq n$$

Two matrices A and B are conformable for multiplication if the number of columns of A equals the number of rows of B . The product is defined as

$$C = AB \quad \text{where an element } c_{ij} \text{ is defined by } c_{ij} = \sum_k a_{ik} b_{kj}$$

Thus each element of C is computed as an inner product of a row of A with a column of B . Matrix multiplication is associative but not commutative in general. The multiplicative identity in matrix algebra is called the identity matrix I . I is all zeros except that all elements in its main diagonal have value 1 ($a_{ij} = 1$ if $i = j$, else $a_{ij} = 0$). Sometimes the $n \times n$ identity matrix is written I_n .

The transpose of an $m \times n$ matrix A is the $n \times m$ matrix A^T such that the i, j th element of A is the j, i th element of A^T . If $A^T = A$, A is symmetric.

The inverse matrix of an $n \times n$ matrix A is written A^{-1} . If it exists, then

$$AA^{-1} = A^{-1}A = I$$

If its inverse does not exist, an $n \times n$ matrix is called singular.

With k and p scalars, and A , B , and C $m \times n$ matrices, the following are some laws of matrix algebra (operations are matrix operations):

$$\begin{aligned} A + B &= B + A \\ (A + B) + C &= A + (B + C) \\ k(A + B) &= kA + kB \\ (k + p)A &= kA + pA \\ AB &\neq BA \quad \text{in general} \\ (AB)C &= A(BC) \\ A(B + C) &= AB + AC \\ (A + B)C &= AC + BC \end{aligned}$$

$$\begin{aligned}
 A(kB) &= k(AB) = (kA)B \\
 I_m A &= A I_n = A \\
 (A + B)^T &= A^T + B^T \\
 (AB)^T &= B^T A^T \\
 (AB)^{-1} &= B^{-1} A^{-1}
 \end{aligned}$$

The determinant of an $n \times n$ matrix is an important quantity; among other things, a matrix with zero determinant is singular. Let A_{ij} be the $(n-1) \times (n-1)$ matrix resulting from deleting the i th row and j th column from an $n \times n$ matrix A . The determinant of a 1×1 matrix is the value of its single element. For $n > 1$,

$$\det A = \sum_{i=1}^n a_{ij} (-1)^{i+j} \det A_{ij}$$

for any j between 1 and n . Given the definition of determinant, the inverse of a matrix may be defined as

$$(a^{-1})_{ij} = \frac{(-1)^{i+j} \det A_{ji}}{\det A}$$

In practice, matrix inversion may be a difficult computational problem, but this important algorithm has received much attention, and robust and efficient methods exist in the literature, many of which may also be used to compute the determinant. Many of the matrices arising in computer vision have to do with geometric transformations, and have well-behaved inverses corresponding to the inverse transformations. Matrices of small dimensionality are usually quite computationally tractable.

Matrices are often used to denote linear transformations; if a row (column) matrix X of dimension n is post (pre) multiplied by an $n \times n$ matrix A , the result $X' = XA$ ($X' = AX$) is another row (column) matrix, each of whose elements is a linear combination of the elements of X , the weights being supplied by the values of A . By employing the common pun between row matrices and vectors, $\mathbf{x}' = \mathbf{x}A$ ($\mathbf{x}' = A\mathbf{x}$) is often written for a linear transformation of a vector \mathbf{x} .

An eigenvector of an $n \times n$ matrix A is a vector \mathbf{v} such that for some scalar λ (called an eigenvalue),

$$\mathbf{v}A = \lambda \mathbf{v}$$

That is, the linear transformation A operates on \mathbf{v} just as a scaling operation. A matrix has n eigenvalues, but in general they may be complex and of repeated values. The computation of eigenvalues and eigenvectors of matrices is another computational problem of major importance, with good algorithms for general matrices being complicated. The n eigenvalues are roots of the so-called characteristic polynomial resulting from setting a formal determinant to zero:

$$\det(A - \lambda I) = 0.$$

Eigenvalues of matrices up to 4×4 may be found in closed form by solving the characteristic equation exactly. Often, the matrices whose eigenvalues are of interest are symmetric, and luckily in this case the eigenvalues are all real. Many algorithms exist in the literature which compute eigenvalues and eigenvectors both for symmetric and general matrices.

A1.5. LINES

An infinite line may be represented by several methods, each with its own advantages and limitations. An example of a representation which is not often very useful is two planes that intersect to form the line. The representations below have proven generally useful.

A1.5.1 Two Points

A two-dimensional or three-dimensional line (throughout Appendix 1 this shorthand is used for “line in two-space” and “line in three-space”; similarly for “two (three) dimensional point”) is determined by two points on it, \mathbf{x}_1 and \mathbf{x}_2 . This representation can serve as well for a half-line or a line segment. The two points can be kept as the rows of a $(2 \times n)$ matrix.

A1.5.2 Point and Direction

A two-dimensional or three-dimensional line (or half-line) is determined by a point \mathbf{x} on it (its endpoint) and a direction vector \mathbf{v} along it. This representation is essentially the same as that of Section A1.5.1, but the interpretation of the vectors is different.

A1.5.3 Slope and Intercept

A two-dimensional line can often be represented by the Y value b where the line intersects the Y axis, and the slope m of the line (the tangent of its inclination with the x axis). This representation fails for vertical lines (those with infinite slope). The representation is in the form of an equation making explicit the dependence of y on x :

$$y = mx + b$$

A similar representation may of course be based on the X intercept.

A1.5.4 Ratios

A two-dimensional or three-dimensional line may be represented as an equation of ratios arising from two points $\mathbf{x}_1 = (x_1, y_1, z_1)$ and $\mathbf{x}_2 = (x_2, y_2, z_2)$ on the line.

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1}$$

A1.5.5 Normal and Distance from Origin (Line Equation)

This representation for two-dimensional lines is elegant in that its parts have useful geometric significance which extends to planes (not to three-dimensional lines). The coefficients of the general two-dimensional linear equation represent a two-dimensional line and incidentally give its normal (perpendicular) vector and its (perpendicular) distance from the origin (Fig. A1.6).

From the ratio representation above, it is easy to derive (in two dimensions) that

$$(x - x_1) \sin \theta - (y - y_1) \cos \theta = 0$$

so for

$$d = -(x_1 \sin \theta - y_1 \cos \theta),$$

$$x \sin \theta - y \cos \theta + d = 0$$

This equation has the form of a dot product with a formal homogeneous vector $(x, y, 1)$:

$$(x, y, 1) \cdot (\sin \theta, -\cos \theta, d) = 0$$

Here the two-dimensional vector $(\sin \theta, -\cos \theta)$ is perpendicular to the line (it is a unit normal vector, in fact), and d is the signed distance in the direction of the normal vector from the line to the origin. Multiplying both sides of the equation by a constant leaves the line invariant, but destroys the interpretation of d as the distance to the origin.

This form of line representation has several advantages besides the interpretations of its parameters. The parameters never go to infinity (this is useful in the Hough algorithm described in Chapter 4). The representation extends naturally to representing n -dimensional planes. Least squared error line fitting (Section A1.9) with this form of line equation (as opposed to slope-intercept) minimizes errors perpendicular to the line (as opposed to those perpendicular to one of the coordinate axes).

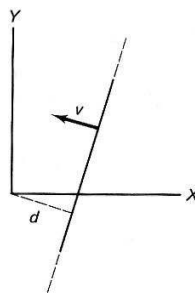


Fig. A1.6 Two-dimensional line with normal vector and distance to origin.

A1.5.6 Parametric

It is sometimes useful to be able mathematically to “walk along” a line by varying some parameter t . The basic parametric representation here follows from the two-point representation. If \mathbf{x}_1 and \mathbf{x}_2 are two particular points on the line, a general point on the line may be written as

$$\mathbf{x} = \mathbf{x}_1 + t(\mathbf{x}_2 - \mathbf{x}_1)$$

In matrix terms this is

$$\mathbf{x} = [t \ 1]L$$

where L is the $2 \times n$ matrix whose first row is $(\mathbf{x}_2 - \mathbf{x}_1)$ and whose second is \mathbf{x}_1 . Parametric representations based on points on the lines may be transformed by the geometric point transformations (Section A1.7).

A1.6. PLANES

The most common representation of planes is to use the coordinates of the plane equation. This representation is an extension of the line-equation representation of Section A1.5.5. The plane equation may be written

$$ax + by + cz + d = 0$$

which is in the form of a dot product $\mathbf{x} \cdot \mathbf{p} = 0$. Four numbers given by $\mathbf{p} = (a, b, c, d)$ characterize a plane, and any homogeneous point $\mathbf{x} = (x, y, z, w)$ satisfying the foregoing equation lies in the plane. In \mathbf{p} , the first three numbers (a, b, c) form a normal vector to the plane. If this normal vector is made to be a unit vector by scaling \mathbf{p} , then d is the signed distance to the origin from the plane. Thus the dot product of the plane coefficient vector and any point (in homogeneous coordinates) gives the distance of the point to the plane (Fig. A1.7).

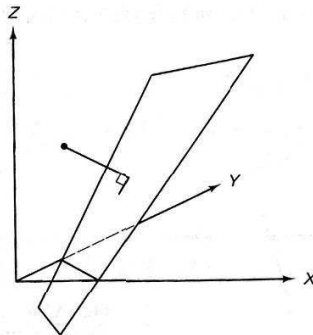


Fig. A1.7 Distance from a point to a plane.

Three noncollinear points x_1, x_2, x_3 determine a plane p . To find it, write

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} p = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

If the matrix containing the point vectors can be inverted, the desired vector p is thus proportional to the fourth column of the inverse.

Three planes p_1, p_2, p_3 may intersect in a point x . To find it, write

$$x \begin{bmatrix} p_1 & p_2 & p_3 & 0 \\ & & & 0 \\ & & & 0 \\ & & & 1 \end{bmatrix} = [0 \quad 0 \quad 0 \quad 1]$$

If the matrix containing the plane vectors can be inverted, the desired point p is given by the fourth row of the inverse. If the planes do not intersect in a point, the inverse does not exist.

A1.7 GEOMETRIC TRANSFORMATIONS

This section contains some results that are well known through their central place in the computer graphics literature, and illustrated in greater detail there. The idea is to use homogeneous coordinates to allow the writing of important transformations (including affine and projective) as linear transformations. The transformations of interest here map points or point sets onto other points or point sets. They include rotation, scaling, skewing, translation, and perspective distortion (point projection) (Fig. A1.8).

A point x in three-space is written as the homogeneous row four-vector (x, y, z, w) , and postmultiplication by the following transformation matrices accomplishes point transformation. A set of m points may be represented as an $m \times 4$ matrix of row point vectors, and the matrix multiplication transforms all points at once.

A1.7.1 Rotation

Rotation is measured clockwise about the named axis while looking along the axis toward the origin.

Rotation by θ about the X axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

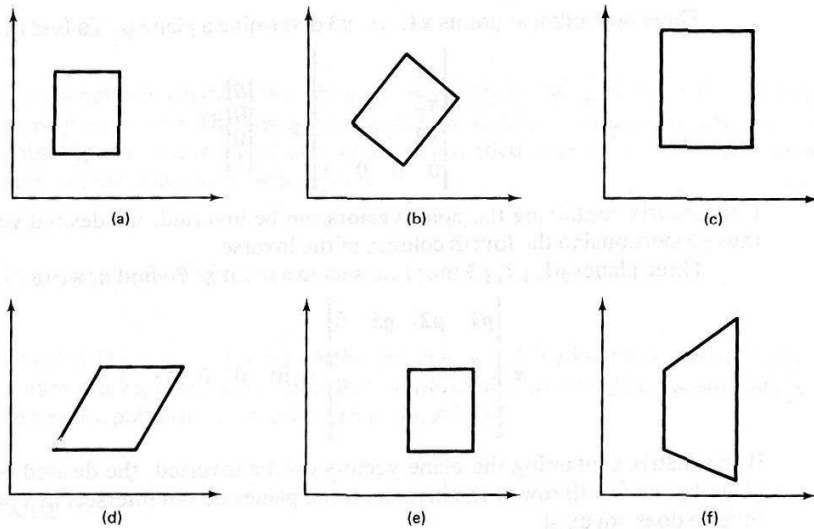


Fig. A1.8 Transformations: (a) original, (b) rotation, (c) scaling, (d) skewing, (e) translation, and (f) perspective.

Rotation by θ about the Y axis:

$$\begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation by θ about the Z axis:

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A1.7.2 Scaling

Scaling is stretching points out along the coordinate directions. Scaling can transform a cube to an arbitrary rectangular parallelepiped.

Scale by S_x , S_y , and S_z in the X , Y , and Z directions:

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A1.7.3 Skewing

Skewing is a linear change in the coordinates of a point based on certain of its other coordinates. Skewing can transform a square into a parallelogram in a simple case:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ d & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In general, skewing is quite powerful:

$$\begin{bmatrix} 1 & k & n & 0 \\ d & 1 & p & 0 \\ e & m & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation is a composition of scaling and skewing (Section A1.7.7).

A1.7.4 Translation

Translate a point by (t, u, v) :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t & u & v & 1 \end{bmatrix}$$

With a three-dimensional Cartesian point representation, this transformation is accomplished through vector addition, not matrix multiplication.

A1.7.5 Perspective

The properties of point projection, which model perspective distortion, were derived in Chapter 2. In this formulation the viewpoint is on the positive Z axis at $(0, 0, f, 1)$ looking toward the origin: f acts like a “focal length”. The visible world is projected through the viewpoint onto the $Z = 0$ image plane (Fig. A1.9).

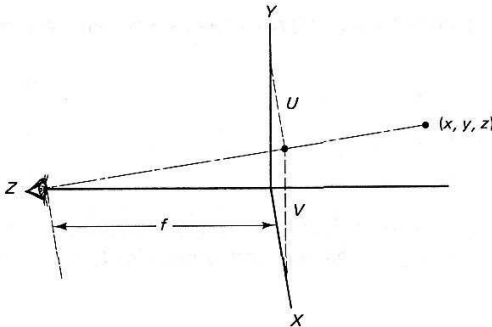


Fig. A1.9 Geometry of image formation.

Similar triangles arguments show that the image plane point for any world point (x, y, z) is given by

$$(U, V) = \left(\frac{fx}{f-z}, \frac{fy}{f-z} \right)$$

Using homogeneous coordinates, a “perspective distortion” transformation can be written which distorts three-dimensional space so that after orthographic projection onto the image plane, the result looks like that required above for perspective distortion. Roughly, the transformation shrinks the size of things as they get more distant in Z . Although the transformation is of course linear in homogeneous coordinates, the final step of changing to Cartesian coordinates by dividing through by the fourth vector element accomplishes the nonlinear shrinking necessary.

Perspective distortion (situation of Fig. A1.9):

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective from a general viewpoint has nonzero elements in the entire fourth column, but this is just equivalent to a rotated coordinate system and the perspective distortion above (Section A1.7).

A1.7.6 Transforming Lines and Planes

Line and plane equations may be operated on by linear transformations, just as points can. Point-based parametric representations of lines and planes transform as do points, but the line and plane equation representations act differently. They have an elegant relation to the point transformation. If T is a transformation matrix (3×3 for two dimensions, 4×4 for three dimensions) as defined in Sections A1.7.1 to A1.7.5, then a point represented as a row vector is transformed as

$$\mathbf{x}' = \mathbf{x}T$$

and the linear equation (line or plane) when represented as a column vector \mathbf{v} is transformed by

$$\mathbf{v}' = T^{-1}\mathbf{v}$$

A1.7.7 Summary

The 4×4 matrix formulation is a way to unify the representation and calculation of useful geometric transformations, rigid (rotation and translation), and nonrigid

(scaling and skewing), including the projective. The semantics of the matrix are summarized in Fig. A1.10.

Since the results of applying a transformation to a row vector is another row vector, transformations may be concatenated by repeated matrix multiplication. Such composition of transformations follows the rules of matrix algebra (it is associative but not commutative, for instance). The semantics of

$$\mathbf{x}' = \mathbf{x}ABC$$

is that \mathbf{x}' is the vector resulting from applying transformation A to \mathbf{x} , then B to the transformed \mathbf{x} , then C to the twice-transformed \mathbf{x} . The single 4×4 matrix $D = ABC$ would do the same job. The inverses of geometric transformation matrices are just the matrices expressing the inverse transformations, and are easy to derive.

A1.8. CAMERA CALIBRATION AND INVERSE PERSPECTIVE

The aim of this section is to explore the correspondence between world and image points. A (half) line of sight in the world corresponds to each image point. Camera calibration permits prediction of where in the image a world point will appear. Inverse perspective transformation determines the line of sight corresponding to an image point. Given an inverse perspective transform and the knowledge that a visible point lies on a particular world plane (say the floor, or in a planar beam of light), then its precise three-dimensional coordinates may be found, since the line of sight generally intersects the world plane in just one point.

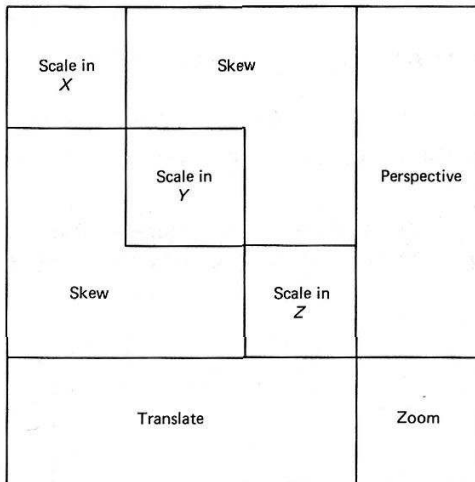


Fig. A1.10 The 4×4 homogeneous transformation matrix.

A1.8.1 Camera Calibration

This section is concerned with the “camera model”; the model takes the form of a 4×3 matrix mapping three-dimensional world points to two-dimensional image points. There are many ways to derive a camera model. The one given here is easy to state mathematically; in practice, a more general optimization technique such as hill climbing can be most effective in finding the camera parameters, since it can take advantage of any that are already known and can reflect dependencies between them.

Let the image plane coordinates be U and V ; in homogeneous coordinates an image plane point is (u, v, t) . Thus

$$U = \frac{u}{t}$$

$$V = \frac{v}{t}$$

Call the desired camera model matrix C , with elements C_{ij} and column four-vectors C_j . Then for any world point (x, y, z) a C is needed such that

$$(x, y, z, 1)C = (u, v, t)$$

So

$$u = (x, y, z, 1)C_1$$

$$v = (x, y, z, 1)C_2$$

$$t = (x, y, z, 1)C_3$$

Expanding the inner products and rewriting $u - Ut = 0$ and $v - Vt = 0$,

$$xC_{11} + yC_{21} + zC_{31} + C_{41} - UxC_{13} - UyC_{23} - UzC_{33} - UC_{43} = 0$$

$$xC_{12} + yC_{22} + zC_{32} + C_{42} - VxC_{13} - VyC_{23} - VzC_{33} - VC_{43} = 0$$

The overall scaling of C is irrelevant, thanks to the homogeneous formulation, so C_{43} may be arbitrarily set to 1. Then equations such as those above can be written in matrix form:

$$\begin{bmatrix} x^1 & y^1 & z^1 & 1 & 0 & 0 & 0 & 0 & -U^1x^1 & -U^1y^1 & -U^1z^1 \\ 0 & 0 & 0 & 0 & x^1 & y^1 & z^1 & 1 & -V^1x^1 & -V^1y^1 & -V^1z^1 \\ x^2 & y^2 & z^2 & 1 & \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & & & & & & & & & & \cdot \\ \cdot & & & & & & & & & & \cdot \\ \cdot & & & & & & & & & & \cdot \\ 0 & 0 & 0 & 0 & x^n & y^n & z^n & 1 & -V^n x^n & -V^n y^n & -V^n z^n \end{bmatrix} \begin{bmatrix} C_{11} \\ C_{21} \\ \cdot \\ \cdot \\ \cdot \\ C_{34} \end{bmatrix} = \begin{bmatrix} U^1 \\ V^1 \\ \cdot \\ \cdot \\ U^n \\ V^n \end{bmatrix}$$

Eleven such equations allow a solution for C . Two equations result for every association of an (x, y, z) point with a (U, V) point. Such an association must be established using visible objects of known location (often placed for the purpose). If more than $5\frac{1}{2}$ such observations are used, a least-squared-error solution to the overdetermined system may be obtained by using a pseudo-inverse to solve the resulting matrix equation (Section A1.9).

A1.8.2 Inverse Perspective

Finding the world line corresponding to an image point relies on the fact that the perspective transformation matrix also affects the z component of a world point. This information is lost when the z component is projected away orthographically, but it encodes the relation between the focal point and the z position of the point. Varying this third component references points whose world positions vary in z but which project onto the same position in the image. The line can be parameterized by a variable p that formally occupies the position of that z coordinate in three-space that has no physical meaning in imaging.

Write the inverse perspective transform P^{-1} as

$$(x', y', p, 1)P^{-1} = (x', y', p, 1 + \frac{p}{f})$$

Rewriting this in the usual way gives these relations between the (x, y, z) points on the line.

$$(x, y, z, 1) = \left(\frac{fx'}{f+p'}, \frac{fy'}{f+p'}, \frac{fp'}{f+p'}, 1 \right)$$

Eliminating the parameter p between the expressions for z and x and those for z and y leaves

$$x = \frac{x'}{y'} y = \frac{-x'}{f} (z - f)$$

Thus x , y , and z are linearly related; as expected, all points on the inverse perspective transform of an image point lie in a line, and unsurprisingly both the viewpoint $(0, 0, f)$ and the image point $(x', y', 0)$ lie on it.

A camera matrix C determines the three-dimensional line that is the inverse perspective transform of any image point. Scale C so that $C_{43} = 1$, and let world points be written $\mathbf{x} = (x, y, z, 1)$ and image points $\mathbf{u} = (u, v, t)$. The actual image points are then

$$U = \frac{u}{t}, \quad V = \frac{v}{t}, \quad \text{so } u = Ut, \quad v = Vt$$

Since

$$\mathbf{u} = \mathbf{x}C,$$

$$u = Ut = \mathbf{x}C_1$$

$$v = Vt = \mathbf{x}C_2$$

$$t = \mathbf{x}C_3$$

Substituting the expression for t into that for u and v gives

$$U\mathbf{x}C_3 = \mathbf{x}C_1$$

$$V\mathbf{x}C_3 = \mathbf{x}C_2$$

which may be written

$$\mathbf{x}(C_1 - UC_3) = 0$$

$$\mathbf{x}(C_2 - VC_3) = 0$$

These two equations are in the form of plane equations. For any U, V in the image and camera model C , there are determined two planes whose intersection gives the desired line. Writing the plane equations as

$$a_1x + b_1y + c_1z + d_1 = 0$$

$$a_2x + b_2y + c_2z + d_2 = 0$$

then

$$a_1 = C_{11} - C_{13}U \quad a_2 = C_{12} - C_{13}V$$

and so on. The direction (λ, μ, ν) of the intersection of two planes is given by the cross product of their normal vectors, which may now be written as

$$\begin{aligned} (\lambda, \mu, \nu) &= (a_1, b_1, c_1) \times (a_2, b_2, c_2) \\ &= (b_1c_2 - b_2c_1, c_1a_2 - c_2a_1, a_1b_2 - a_2b_1) \end{aligned}$$

Then if $\nu \neq 0$, for any particular z_0 ,

$$x_0 = \frac{b_1(c_2z_0 + d_2) - b_2(c_1z_0 - d_1)}{a_1b_2 - b_1a_2}$$

$$y_0 = \frac{a_2(c_1z_0 + d_1) - a_1(c_2z_0 - d_2)}{a_1b_2 - b_1a_2}$$

and the line may be written

$$\frac{x - x_0}{\lambda} = \frac{y - y_0}{\mu} = \frac{z - z_0}{\nu}$$

A1.9. LEAST-SQUARED-ERROR FITTING

The problem of fitting a simple functional model to a set of data points is a common one, and is the concern of this section. The subproblem of fitting a straight line to a set of (x, y) points ("linear regression") is the first topic. In computer vision, this line-fitting problem is encountered relatively often. Model-fitting methods try to find the "best" fit; that is, they minimize some error. Methods which yield closed-form, analytical solutions for such best fits are at issue here.

The relevant “error” to minimize is determined partly by assumptions of dependence between variables. If x is independent, the line may be represented as $y = mx + b$ and the error defined as the vertical displacement of a point from the line. Symmetrically, if x is dependent, horizontal error should be minimized. If neither variable is dependent, a reasonable error to minimize is the perpendicular distance from points to the line. In this case the line equation $ax + by + 1 = 0$ can be used with the method shown here, or the eigenvector approach of Section A1.9.2 may be used.

A1.9.1 Pseudo-Inverse Method

In fitting an $n \times 1$ observations matrix y by some linear model of p parameters, the prediction is that the linear model will approximate the actual data. Then

$$Y = XB + E$$

where X is an $n \times p$ formal independent variable matrix, B is a $p \times 1$ parameter matrix whose values are to be determined, and E represents the difference between the prediction and the actuality: it is an $n \times 1$ error matrix.

For example, to fit a straight line $y = mx + b$ to some data (x_i, y_i) points, form Y as a column matrix of the y_i .

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ & \cdot \\ & \cdot \\ & \cdot \end{bmatrix}$$

$$B = \begin{bmatrix} b \\ m \end{bmatrix}$$

Now the task is to find the parameter B (above, the b and m that determine the straight line) that minimizes the error. The error is the sum of squared difference from the prediction, or the sum of the elements of E squared, or $E^T E$ (if we do not mind conflating the one-element matrix with a scalar). The mathematically attractive properties of the squared-error definition are almost universally taken to compensate for whatever disadvantages it has over what is really meant by error (the absolute value is much harder to calculate with, for example).

To minimize the error, simply differentiate it with respect to the elements of B and set the derivative to 0. The second derivative is positive: this is indeed a minimum. These elementwise derivatives are written tersely in matrix form. First rewrite the error terms:

$$\begin{aligned} E^T E &= (Y - XB)^T (Y - XB) \\ &= Y^T Y - B^T X^T Y - Y^T X B + B^T X^T X B \\ &= Y^T Y - 2B^T X^T Y + B^T X^T X B \end{aligned}$$

(here, the combined terms were 1×1 matrices.) Now differentiate: setting the derivative to 0 yields

$$0 = X^T X B - X^T Y$$

and thus

$$B = (X^T X)^{-1} X^T Y = X^+ Y$$

where X^+ is called the pseudo-inverse of X .

The pseudo-inverse method generalizes to fitting any parametrized model to data (Section A1.9.3). The model should be chosen with some care. For example, Fig. A1.11 shows a disturbing case in which the model above (minimize vertical errors) is used to fit a relatively vertical swarm of points. The “best fit” line in this case is not the intuitive one.

A1.9.2 Principal Axis Method

The principal axes and moments of a swarm of points determine the direction and amount of its dispersion in space. These concepts are familiar in physics as the principal axes and moments of inertia. If a swarm of (possibly weighted) points is translated so that its center of mass (average location) is at the origin, a symmetric matrix M may be easily calculated whose eigenvectors determine the best-fit line or plane in a least-squared-perpendicular-error sense, and whose eigenvalues tell how good the resulting fit is.

Given a set $\{\mathbf{x}^i\}$ row of vectors with weights w^i , define their “scatter matrix” to be the symmetric matrix M , where $\mathbf{x}^i = (x_1^i, x_2^i, x_3^i)$:

$$M = \sum_i \mathbf{x}^i{}^T \mathbf{x}^i$$

$$M_{kp} = \sum_i x_k^i x_p^i \quad 1 \leq k, p \leq 3$$

Define the dispersion of the \mathbf{x}^i in a direction \mathbf{v} (i.e., “dispersion around the plane whose normal is \mathbf{v} ”) to be the sum of weighted squared lengths of the \mathbf{x}^i in the direction \mathbf{v} . This squared error E^2 is

$$E^2 = \sum_i w^i (\mathbf{x}^i \cdot \mathbf{v})^2 = \mathbf{v} \left(\sum_i w^i \mathbf{x}^i{}^T \mathbf{x}^i \right) \mathbf{v}^T = \mathbf{v} M \mathbf{v}^T$$

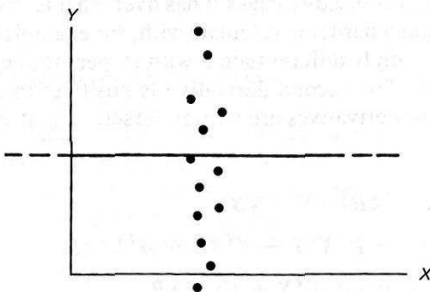


Fig. A1.11 A set of points and the “best fit” line minimizing error in Y .

To find the direction of minimum dispersion (the normal to the best-fit line or plane), note that the minimum of $\mathbf{v}M\mathbf{v}^T$ over all unit vectors \mathbf{v} is the minimum eigenvalue λ_1 of M . If \mathbf{v}_1 is the corresponding eigenvector, the minimum dispersion is attained at $\mathbf{v} = \mathbf{v}_1$. The best fit line or plane of the points goes through the center of mass, which is at the origin; inverting the translation that brought the centroid to the origin yields the best fit line or plane for the original point swarm.

The eigenvectors correspond to dispersions in orthogonal directions, and the eigenvalues tell how much dispersion there is. Thus with a three-dimensional point swarm, two large eigenvalues and one small one indicate a planar swarm whose normal is the smallest eigenvector. Two small eigenvalues and one large one indicate a line in the direction of the normal to the "worst fit plane", or eigenvector of largest eigenvalue. (It can be proved that in fact this is the best-fit line in a least squared perpendicular error sense). Three equal eigenvalues indicate a "spherical" swarm.

A1.9.3 Fitting Curves by the Pseudo-Inverse Method

Given a function $f(\mathbf{x})$ whose value is known on n points $\mathbf{x}_1, \dots, \mathbf{x}_n$, it may be useful to fit it with a function $g(\mathbf{x})$ of m parameters (b_1, \dots, b_m). If the squared error at a point x_i is defined as

$$(e_i)^2 = [f(\mathbf{x}_i) - g(\mathbf{x}_i)]^2$$

a sequence of steps similar to that of Section A1.9.1 leads to setting a derivative to zero and obtaining

$$0 = G^T G \mathbf{b} - G^T \mathbf{f}$$

where \mathbf{b} is the vector of parameters, \mathbf{f} the vector of n values of $f(\mathbf{x})$, and

$$G = \frac{\partial \mathbf{g}}{\partial \mathbf{b}} = \begin{bmatrix} \frac{\partial g(x_1)}{\partial b_1} & \frac{\partial g(x_2)}{\partial b_2} & \dots \\ \vdots & \vdots & \vdots \\ \vdots & \dots & \frac{\partial g(x_n)}{\partial b_m} \end{bmatrix}$$

As before, this yields

$$\mathbf{b} = (G^T G)^{-1} G^T \mathbf{f}$$

Explicit least-squares solutions for curves can have nonintuitive behavior. In particular, say that a general circle is represented

$$\mathcal{G}(x, y) = x^2 + y^2 + 2Dx + 2Ey + F$$

this yields values of D , E , and F which minimize

$$e^2 = \sum_{i=1}^n \mathcal{G}(x_i, y_i)^2$$

for n input points. The error term being minimized does not turn out to accord with our intuitive one. It gives the intuitive distance of a point to the curve, but weighted by a factor roughly proportional to the radius of the curve (probably not desirable). The best fit criterion thus favors curves with high average curvature, resulting in smaller circles than expected. In fitting ellipses, this error criterion favors more eccentric ones.

The most successful conic fitters abandon the luxury of a closed-form solution and go to iterative minimization techniques, in which the error measure is adjusted to compensate for the unwanted weighting, as follows.

$$e^2 = \sum_{i=1}^n \left(\frac{f(x_i, y_i)}{|\nabla f(x_i, y_i)|} \right)^2$$

A1.10 CONICS

The conic sections are useful because they provide closed two-dimensional curves, they occur in many images, and they are well-behaved and familiar polynomials of low degree. This section gives their equations in standard form, illustrates how the general conic equation may be put into standard form, and presents some sample specific results for characterizing ellipses.

All the standard form conics may be subjected to rotation, translation, and scaling to move them around on the plane. These operations on points affect the conic equation in a predictable way.

Circle: $r = \text{radius}$ $x^2 + y^2 = r^2$

Ellipse: $a, b = \text{major, minor axes}$ $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

Parabola: $(p, 0) = \text{focus}, p = \text{directrix}$ $y^2 = 4px$

Hyperbola: vertices $(\pm a, 0)$, asymptotes $y = \pm \left(\frac{b}{a}\right)x$ $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$

The general conic equation is

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0$$

This equation may be written formally as

$$(x \ y \ 1) \begin{bmatrix} A & B & D \\ B & C & E \\ D & E & F \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{xMx}^T = 0$$

Putting the general conic equation into one of the standard forms is a common analytic geometry exercise. The symmetric 3×3 matrix M may be diagonalized, thus eliminating the coefficients B , D , and E from the equation and reducing it to be close to standard form. The diagonalization amounts to a rigid motion that puts the conic in a symmetric position at the origin. The transformation is in fact the 3×3 matrix E whose rows are eigenvectors of M . Recall that if \mathbf{v} is an eigenvector of M ,

$$\mathbf{vM} = \lambda \mathbf{v}$$

Then if D is a diagonal matrix of the three eigenvalues, $\lambda_1, \lambda_2, \lambda_3$,

$$EM = DE$$

but then

$$EME^{-1} = DEE^{-1} = D$$

and M has been transformed by a similarity transformation into a diagonal matrix such that

$$\mathbf{x}D\mathbf{x}^T = 0$$

This general idea is of course related to the principal axis calculation given in Section A1.9.2, and extends to three-dimensional quadric surfaces such as the ellipsoid, cone, hyperbolic paraboloid, and so forth. The general result given above has particular consequences illustrated by the following facts about the ellipse. Given a general conic equation representing an ellipse, its center (x_c, y_c) is given by

$$x_c = \frac{BE - 2CD}{B^2 - 4AC}$$

$$y_c = \frac{2EA - BD}{B^2 - 4AC}$$

The orientation is

$$\theta = \frac{1}{2} \tan^{-1} \left(\frac{B}{A - C} \right)$$

The major and minor axes are

$$\frac{-2G}{(A + C) \pm [B^2 + (A - C)^2]^{1/2}}$$

where

$$G = F - (Ax_c^2 + Bx_c y_c + Cy_c^2)$$

A1.11 INTERPOLATION

Interpolation fits data by giving values between known data points. Usually, the interpolating function passes through each given data point. Many interpolation methods are known; one of the simplest is Lagrangean interpolation.

A1.11.1 One-Dimensional

Given $n + 1$ points (x_j, y_j) , $x_0 < x_1 < \dots < x_n$, the idea is to produce an n th-degree polynomial involving $n + 1$ so-called Lagrangean coefficients. It is

$$f(x) = \sum_{j=0}^n L_j(x) y_j$$

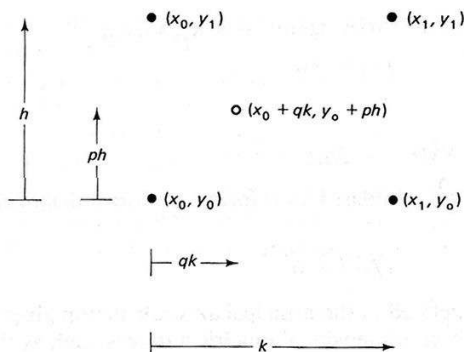


Fig. A1.12 Four point lagrangean interpolation on rectangular grid.

where $L_j(x)$ is the j th coefficient;

$$L_j(x) = \frac{(x - x_0) (x - x_1) \cdots (x - x_{j-1}) (x - x_{j+1}) \cdots (x - x_n)}{(x_j - x_0) (x_j - x_1) \cdots (x_j - x_{j-1}) (x_j - x_{j+1}) \cdots (x_j - x_n)}$$

Other interpolative schemes include divided differences, Hermite interpolation for use when function derivatives are also known, and splines. The use of a polynomial interpolation rule can always produce surprising results if the function being interpolated does not behave locally like a polynomial.

A1.11.2 Two-Dimensional

The four-point Lagrangean method is for the situation shown in Fig. A1.12. Let $f_{ij} = f(x_i, y_j)$. Then

$$f(x_0 + qk, y_0 + ph) = (1 - p)(1 - q)f_{00} + q(1 - p)f_{10} + p(1 - q)f_{01} + pqf_{11}$$

A1.12 THE FAST FOURIER TRANSFORM

The following routine computes the discrete Fourier transform of a one-dimensional complex array XIn of length $N = 2^{\log N}$ and produces the one-dimensional complex array XOut. It uses an array W of the N complex Nth roots of unity, computed as shown, and an array Bits containing a bit-reversal table of length N. N, LogN, W, and Bits are all global to the subroutine as written. If the logical variable Forward is TRUE, the FFT is performed; if Forward is FALSE, the inverse FFT is performed.

```

SUBROUTINE FFT(XIn, KOut, Forward)
GLOBAL W, Bits, N, LogN
LOGICAL Forward
COMPLEX XIn, Xout, W, A, B
INTEGER Bits
ARRAY(0:N) W, Bits, XIn, XOut

```

```

DO (I = 0, N - 1) XOut(I) = XIn(Bits(I))
JOff = N/2
JPnt = N/2
JBk = 2
IOFF = 1
DO (I = 1, LogN)
.   DO (IStart = 0, N - 1, JBk)
.   .   JWPnt = 0
.   .   DO (K = IStart, IStart + IOff - 1)
.   .   .   WHEN (Forward)
.   .   .   .   A = XOut(K + IOff) * W(JWPnt) + XOut(K)
.   .   .   .   B = XOut(K + IOff) * W(JWPnt + JOff) + XOut(K)
.   .   .   .   ... FIN
.   .   .   ELSE
.   .   .   .   A = XOut(K + IOff) * CONJG(W(JWPnt)) + XOut(K)
.   .   .   .   B = XOut(K + IOff) * CONJG(W(JWPnt + JOff)) + XOut(K)
.   .   .   .   ... FIN
.   .   .   XOut(K) = A
.   .   .   XOut(K + IOff) = B
.   .   .   JWPnt = JWPnt + JPnt
.   .   .   ... FIN
.   .   ... FIN
.   JPnt = JPnt/2
.   IOff = JBk
.   JBk = JBk * 2
.   ... FIN
UNLESS (Forward)
.   DO (I = 0, N - 1) XOut(I) = XOut(I)/N
.   ... FIN
END

```

TO INIT-W

```

.   Pi = 3.14159265
.   DO (K = 0, N - 1)
.   .   Theta = 2 * Pi/N
.   .   W(K) = CMPLX(COS(Theta * K), SIN(Theta * K))
.   .   ... FIN
.   ... FIN

```

TO BIT-REV

```

.   Bits(0) = 0
.   M = 1
.   DO (I = 0, LogN - 1)
.   .   DO (J = 0, M - 1)
.   .   .   Bits(J) = Bits(J) * 2

```

```

. . . Bits(J + M) = Bits(J) + 1
. . . ... FIN
. . . M = M * 2
. . . ... FIN
... FIN

```

A1.13 THE ICOSAHEDRON

Geodesic dome constructions provide a useful way to partition the sphere (hence the three-dimensional directions) into relatively uniform patches. The resulting polyhedra look like those of Fig. A1.13.

The icosahedron has 12 vertices, 20 faces, and 30 edges. Let its center be at the origin of Cartesian coordinates and let each vertex be a unit distance from the center. Define

$$t, \text{ the golden ratio} = \frac{1 + \sqrt{5}}{2}$$

$$a = \frac{\sqrt{t}}{5^{1/4}}$$

$$b = \frac{1}{(\sqrt{t} 5^{1/4})}$$

$$c = a + 2b = \frac{1}{b}$$

$$d = a + b = \frac{t^{3/2}}{5^{1/4}}$$

$$A = \text{angle subtended by edge at origin} = \arccos\left(\frac{\sqrt{5}}{5}\right)$$

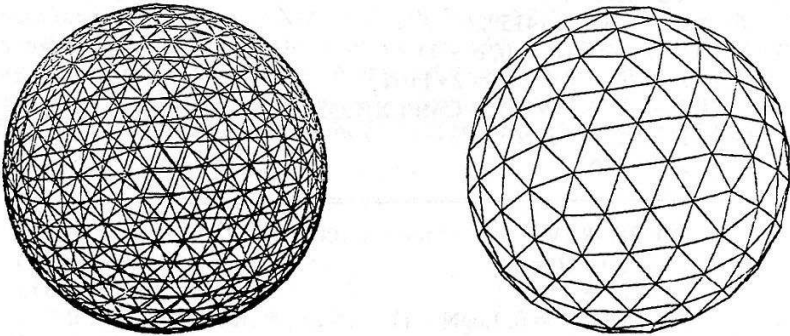


Fig. A1.13 Multifaceted polyhedra from the icosahedron.

Then

angle between radius and an edge = $b = \arccos(b)$

edge length = $2b$

distance from origin to center of edge = a

distance from origin to center of face = $\frac{ta}{\sqrt{3}}$

The 12 vertices may be placed at

$$(0, \pm a, \pm b)$$

$$(\pm b, 0, \pm a)$$

$$(\pm a, \pm b, 0)$$

Then midpoints of the 20 faces are given by

$$\frac{1}{3}(\pm d, \pm d, \pm d)$$

$$\frac{1}{3}(0, \pm a, \pm c)$$

$$\frac{1}{3}(\pm c, 0, \pm a)$$

$$\frac{1}{3}(\pm a, \pm c, 0)$$

To subdivide icosahedral faces further, several methods suggest themselves, the simplest being to divide each edge into n equal lengths and then construct n^2 congruent equilateral triangles on each face, pushing them out to the radius of the sphere for their final position. (There are better methods than this if more uniform face sizes are desired.)

A1.14 ROOT FINDING

Since polynomials of fifth and higher degree are not soluble in closed form, numerical (approximate) solutions are useful for them as well as for nonpolynomial functions. The Newton–Raphson method produces successive approximations to a real root of a differentiable function of one variable.

$$x^{i+1} = x^i - \frac{f(x^i)}{f'(x^i)}$$

Here x^i is the i th approximation to the root, and $f(x^i)$ and $f'(x^i)$ are the function and its derivative evaluated at x^i . The new approximation to the root is x^{i+1} . The successive generation of approximations can stop when they converge to a single value. The convergence to a root is governed by the choice of initial approximation to the root and by the behavior of the function in the vicinity of the root. For instance, several roots close together can cause problems.

The one-dimensional form of this method extends in a natural way to solving systems of simultaneous nonlinear equations. Given n functions F_i , each of n parameters, the problem is to find the set of parameters that drives all the functions to zero. Write the parameter vector \mathbf{x} .

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Form the function column vector \mathbf{F} such that

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{bmatrix}$$

The Jacobean matrix J is defined as

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \dots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \dots & \dots & \frac{\partial F_n}{\partial x_n} \end{bmatrix}$$

Then the extension of the Newton–Raphson formula is

$$\mathbf{x}^{i+1} = \mathbf{x}^i - J^{-1}(\mathbf{x}^i)\mathbf{F}(\mathbf{x}^i)$$

which requires one matrix inversion per iteration.

EXERCISES

A1.1 \mathbf{x} and \mathbf{y} are two two-dimensional vectors placed tail to tail. Prove that the area of the triangle they define is $|\mathbf{x} \times \mathbf{y}|/2$.

A1.2 Show that points \mathbf{q} in a plane defined by the three points \mathbf{x} , \mathbf{y} , and \mathbf{z} are given by

$$\mathbf{q} \cdot \left[(\mathbf{y} - \mathbf{x}) \times (\mathbf{z} - \mathbf{x}) \right] = \mathbf{x} \cdot (\mathbf{y} \times \mathbf{z})$$

A1.3 Verify that the vector triple product may be written as claimed in its definition.

A1.4 Given an arctangent routine, write an arcsine routine.

A1.5 Show that the closed form for the inverse of a 2×2 A matrix is

$$\frac{1}{\det A} \begin{bmatrix} a_{22} & -a_{21} \\ -a_{12} & a_{11} \end{bmatrix}$$

A1.6 Prove by trigonometry that the matrix transformations for rotation are correct.

- A1.7 What geometric transformation is accomplished when a_{44} of a geometric transformation matrix A varies from unity?
- A1.8 Establish conversions between the given line representations.
- A1.9 Write a geometric transform to mirror points about a given plane.
- A1.10 What is the line-equation representation of a line $L1$ through a point \mathbf{x} and perpendicular to a line $L2$ (similarly represented)? Parallel to $L2$?
- A1.11 Derive the ellipse results given in Section A1.10.
- A1.12 Explicitly derive the values of D , E , and F minimizing the error term

$$\sum_{i=1}^n [f(x_i, y_i)]^2$$

in the general equation for a circle

$$x^2 + y^2 + 2Dx + 2Ey + F = 0$$

- A1.13 Show that if points and lines are transformed as shown in Section A1.7.6, the transformed points indeed lie on the transformed lines.
- A1.14 Explicitly derive the least-squared-error solution for lines represented as $ax + by + 1 = 0$.
- A1.15 If three planes intersect in a point, is the inverse of

$$\begin{bmatrix} p1 & p2 & p3 & 0 \\ & & & 0 \\ & & & 0 \\ & & & 1 \end{bmatrix}$$

guaranteed to exist?

- A1.16 What is the angle between two three-space lines?
- A1.17 In two dimensions, show that two lines \mathbf{u} and \mathbf{v} intersect at a point \mathbf{x} given by $\mathbf{x} = \mathbf{u} \times \mathbf{v}$.
- A1.18 How can you tell if two line segments (defined by their end points) intersect in the plane?
- A1.19 Find a 4×4 matrix that transforms an arbitrary direction (or point) to lie on the Z axis.
- A1.20 Derive a parametric representation for planes based on three points lying in the plane.
- A1.21 Devise a scheme for interpolation on a triangular grid.
- A1.22 What does the homogeneous point $(x, y, z, 0)$ represent?

REFERENCES AND FURTHER READING

Computer Graphics

1. NEWMAN, W. M., and R. F. SPROULL. *Principles of Interactive Computer Graphics*, 2nd Ed. New York: McGraw-Hill, 1979.
2. CHASEN, S. H. *Geometric Principles and Procedures for Computer Graphic Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

3. FAUX, I. D., and M. J. PRATT. *Computational Geometry for Design and Manufacture*, Chichester, UK: Ellis Horwood Ltd, 1979.
4. ROGERS, D. F., and J. A. ADAMS. *Mathematical Elements for Computer Graphics*. New York: McGraw-Hill 1976.

Computer Vision

5. HORN, B. K. P. "VISMEM: Vision Flash 34." AI Lab, MIT, December 1972.
6. SOBEL, I. "Camera models and machine perception." AIM-21, Stanford AI Lab, May 1970.
7. DUDA, R. O. and P. E. HART. *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
8. ROSENFELD, A., and A. C. KAK. *Digital Picture Processing*. New York: Academic Press, 1976.
9. PAVLIDIS, T. *Structural Pattern Recognition*. New York: Springer-Verlag, 1977.

Geometry, Calculus, Numerical Analysis

10. WEXLER, C. *Analytic Geometry: A Vector Approach*. Reading, MA: Addison Wesley, 1962.
11. APOSTOL, T. M. *Calculus*, Vol. 2, Waltham, MA: Blaisdell, 1962.
12. CONTE, S. D. and C. DEBOOR. *Elementary Numerical Analysis: An Algorithmic Approach*. New York: McGraw-Hill, 1972.
13. RALSTON, A. *A First Course in Numerical Analysis*. New York: McGraw-Hill, 1965.
14. ABRAMOWITZ, M, and I. A. STEGUN. *Handbook of Mathematical Functions*. New York: Dover, 1964.
15. HODGEMAN, C. D. (Ed.). *CRC Standard Mathematical Tables*. West Palm Beach, FL: CRC Press.

Geodesic Tessellations

16. BROWN, C. "Fast display of well-tesselated surfaces." *Computers and Graphics* 4, 1979, 77-85.
17. CLINTON, J. D. "Advanced structural geometry studies, part I: polyhedral subdivision concepts for structural applications," NASA CR-1734/35, September 1971.

Fast Transformations

18. PRATT, W. K. *Digital Image Processing*. New York: Wiley-Interscience, 1978.
19. ANDREWS, H. C., and J. KANE. "Kronecker matrices, computer implementation, and generalized spectra." *J. ACM*, April 1970.

Advanced Control Mechanisms

Appendix 2

This appendix is concerned with specific control mechanisms that are provided by programming languages or that may be implemented on top of existing languages as aids to doing computer vision. The treatment here is brief; our aim is to expose the reader to several ideas for control of computer programs that have been developed in the artificial intelligence context, and to indicate how they relate to the main computational goals of computer vision.

A2.1 STANDARD CONTROL STRUCTURES

For completeness, we mention the control mechanisms that are provided as a matter of course by conventional research programming languages, such as Pascal, Algol, POP-2, SAIL, and PL/I. The influential language LISP, which provides a base language for many of the most advanced control mechanisms in computer vision, ironically is itself missing (in its pure form) a substantial number of these more standard constructs. Another common language missing some standard control mechanisms is SNOBOL. These standard constructions are so basic to the current conception of a serial von Neumann computer that they are often realized in the instruction set of the machine. In this sense we are almost talking here of computer hardware.

The standard mechanisms are the following:

1. *Sequence.* Advance the program counter to the next instruction.
2. *Branch instruction.* Go to a specific address.
3. *Conditional branch.* Go to a specific address if a condition is true, otherwise, go to the next instruction.
4. *Iteration.* Repeat a sequence of instructions until a condition is met.

5. *Subroutines.* Go to a certain location; execute a set of instructions using a set of supplied parameters; then return to the next instruction after the subroutine call.

All the standard control structures should be in the toolkit of a programmer. They will be used, together with the data structures and data types supplied in the working language, to implement other control mechanisms. The remainder of this appendix deals with “nonstandard” control mechanisms; those not typically provided in commercial programming languages and which have no close correlates in primitive machine instructions. Nonstandard control mechanisms, although not at all domain-specific, have developed to meet needs that are not the “lowest common denominator” of computer programming. They impose their own view of problem decomposition just as do the standard structures.

Less standard mechanisms are *recursion* and *co-routining*. Co-routining can be thought of as a form of recursion.

A2.1.1 Recursion

Recursion obeys all the constraints of subroutining, except that a routine may call upon “itself.” The user sees no difference between recursive and nonrecursive subroutines, but internally recursion requires slightly more bookkeeping to be performed in the language software, since typically the hardware of a computer does not extend to managing recursion (although some machines have instructions that are quite useful here).

A typical use of a recursive control paradigm in computer vision might be:

```
To Understand-Scene (X);
(
  If Immediately-Apparent(X)
  then Report-Understanding-Of(X);
  else
    (SimplerParts ← Decompose(X);
     ForEach Part in SimplerParts
       Understand-Scene(Part);
    )
  )
[ )
```

Recursion is an elegant way to specify many important algorithms (such as tree traversals), but in a way it has no conceptual differences from subroutining. A routine is broken up into subroutines (some of which may involve smaller versions of the original task); these are attacked sequentially, and they must finish before they return control to the routine that invokes them.

A2.1.2 Co-Routining

Co-routines are simply programs that can call (invoke) each other. Most high-level languages do not directly provide co-routines, and thus they are a nonstandard control structure. However, co-routining is a fundamental concept [Knuth 1973]

and serves here as a bridge between standard and nonstandard control mechanisms.

Subroutines and their calling programs have a “slave–master” aspect: control is always returned to the master calling program after the subroutine has carried out its job. This mechanism not only leads to efficiencies by reducing the amount of executable code, but is considered to be so useful that it is built into the instruction set of most computers. The pervasiveness of subroutines has subtle effects on the approach to problem decomposition, encouraging a hierarchical subproblem structure. The co-routine relationship is more egalitarian than the subroutine relationship. If co-routine *A* needs the services of co-routine *B*, it can call *B*, and (here is the difference) conversely, *B* can call *A* if *B* needs *A*’s services.

Here is a simple (sounding) problem [Floyd 1979]: “Read lines of text, until a completely blank line is found. Eliminate redundant blanks between the words. Print the text, 30 characters to a line, without breaking words between lines.” This problem is hard to program elegantly in most languages because the iterations involved do not nest well (try it!). However, an elegant solution exists if the job is decomposed into three co-routines, calling each other to perform input, formatting, and output of a character stream.

A useful paradigm for problem solving, besides the strictly hierarchical, is that of a “heterarchical” community of experts, each performing a job and when necessary calling on other experts. A heterarchy can be implemented by co-routines. Many of the nonstandard mechanisms discussed below are in the spirit of co-routines.

A2.2 INHERENTLY SEQUENTIAL MECHANISMS

A2.2.1 Automatic Backtracking

The PLANNER language [Hewitt 1972] implicitly implemented the feature of “automatic backtracking.” The advisability of uniformly using this technique, which is equivalent to depth-first search, was questioned by those who wished to give the programmer greater freedom to choose which task to activate next [Sussman and McDermott 1972].

A basic backtracking discipline may be provided by recursive calls, in which a return to a higher level is a “backtrack.” The features of automatic backtracking are predicated on an ability to save and reinstate the computational state of a process automatically, without explicit specification by the programmer.

Automatic backtracking has its problems. One basic problem occurs in systems that perform inferences while following a particular line of reasoning which may ultimately be unsuccessful. The problem is that along the way, perhaps many perfectly valid and useful computations were performed and many facts were added to the internal model. Mixed in with these, of course, are wrong deductions which ultimately cause the line of reasoning to fail. The problem: After having restored control to a higher decision point after a failure is noticed, how is the system

to know which deductions were valid and which invalid? One expensive way suggested by automatic backtracking is to keep track of all hypotheses that contributed to deriving each fact. Then one can remove all results of failed deduction paths. This is generally the wrong thing to do; modern trends have abandoned the automatic backtracking idea and allow the programmer some control over what is restored upon failure-driven backtracking. Typically, a compromise is implemented in which the programmer may mark certain hypotheses for deletion upon backtracking.

A2.2.2 Context Switching

Context switching is a general term that is used to mean switching of general process state (a control primitive) or switching a data base context (a data access primitive). The two ideas are not independent, because it could be confusing for a process to put itself to sleep and be reawakened in a totally different data context.

Backtracking is one use of general control context switching. The most general capability is a "general GO TO." A regular GO TO allows one to go only to a particular location defined in a static program. After the GO TO, all bindings and returnpoints are still determined by the current state of processing. In contrast, a general GO TO allows a transfer not only across program "space," but through program "time" as well. Just as a regular GO TO can go to a predefined program label, a general GO TO can go to a "tag" which is created to save the entire state of a process. To GO TO such a tag is to go back in time and recreate the local binding, access, control, and process state of the process that made the tag.

A good example of the use of such power is given in a problem-solving program that constructs complex structures of blocks [Fahlman 1974].

A2.3 SEQUENTIAL OR PARALLEL MECHANISMS

Some language constructs explicitly designate parallel computing. They may actually reflect a parallel computing environment, but more often they control a simulated version in which several control paths are maintained and multi-processed under system control. Examples here are module and message primitives given below and statements such as the CO-BEGIN, CO-END pairs which can bracket notionally parallel blocks of code in some Algol-like language extensions.

A2.3.1 Modules and Messages

Modules and messages form a useful, versatile control paradigm that is relatively noncommittal. That is, it forces no particular problem decomposition or methodological style on its user, as does a pure subroutine paradigm, for example. Message passing is a general and elegant model of control which can be used to subsume others, such as subroutines, recursion, co-routines, and parallelism [Feldman 1979].

There are many antecedents to the mechanism of modules communicating by messages described here. They include [Feldman and Sproull 1971; Hewitt and

Smith 1975; Goldberg and Kay 1976; Birtwhistle et al. 1973]. In the formulation presented by Hewitt, the message-passing paradigm can be extended down into the lowest level of machine architecture. The construction outlined here [Feldman 1979] is more moderate, since in it the base programming language may be used with its full power, and itself is not module and message based.

A program is made up of *modules*. A module is a piece of code with associated local data. The crucial point is that the internal state of a module (e.g. its data) is not accessible to other modules. Within a module, the base programming language, such as Algol, may be used to its full power (subroutine calls, recursion, iteration, and so forth are allowed). However, modules may not in any sense “call upon” each other. Modules communicate only by means of *messages*. A module may send a message to another module; the message may be a request for service, an informational message, a signal, or whatever. The module to whom the message is sent may, when it is ready, receive the message and process it, and may then itself send messages either to the original module, or indeed to any combination of other modules.

The module–message paradigm has several advantages over subroutine (or co-routine) calls.

1. If subroutines are in different languages, the subroutine call mechanisms must be made compatible.
2. Any sophisticated lockout mechanism for resource access requires the internal coding of queues equivalent to that which a message switcher provides.
3. A subroutine that tries to execute a locked subroutine is unable to proceed with other computation.
4. Having a resource always allocated by a single controlling module greatly simplifies all the common exclusion problems.
5. For inherently distributed resources, message communication is natural. Module-valued slots provide a very flexible but safe discipline for control transfers.

Another view of messages is as a generalization of parameter lists in subroutine or coroutine calls. The idea of explicitly naming parameters is common in assembly languages, where the total number of parameters to a routine may be very large. More important, the message discipline presents to a module a collection of suggested parameters rather than automatically filling in the values of parameters. This leads naturally to the use of semantic checks on the consistency of parameters and to the use of default values for unspecified ones, which can be a substantial improvement on type checking. The use of return messages allows multiple-valued functions; an answer message may have several slots. Messages solve the so-called “uniform reference problem”—one need not be concerned with whether an answer (say an array element) is computed by a procedure or a table.

There is yet another useful view of messages. One can view a message as a partially specified relation (or pattern), with some slot values filled in and some unbound. This is common in relational data bases [Astrahan et al. 1976] and artificial intelligence languages [Bobrow and Raphael 1974]. In this view, a mes-

sage is a task specification with some recipient and some complaint departments to talk to about it. Various modules can attempt to satisfy or contract out parts of the task of filling in the remaining slots. A module may handle messages containing slots unknown to it. This allows several modules to work together on a task while maintaining locality. For example, an executive module could route messages (on the basis of a few slots that it understands) to modules that deal with special aspects of a problem using different slots in the message.

There is no apparent conflict between these varying views of messages. It is too early in their development to be sure, but the combined power of these paradigms seems to provide a qualitative improvement in our ability to develop vision programs.

A2.3.2 Priority Job Queue

In any system of independent processes on a serial computer, there must be a mechanism for scheduling activation. One general mechanism for accomplishing scheduling is the priority job queue. Priority queues are a well-known abstraction [Aho et al. 1974]. Informally, a priority job queue is just an ordered list of processes to be activated. A monitor program is responsible for dequeuing processes and executing them; processes do not give control directly to other processes, but only to the monitor. The only way for a process to initiate another is to enqueue it in the job queue. It is easiest to implement a priority job queue if processes are definable entities in the programming language being used; in other words, programs should be manipulable datatypes. This is possible in LISP and POP-2, for example.

If a process needs another job performed by another process, it enqueues the sub job on the job queue and *suspends* itself (it is *deactivated*, or put to sleep). The sub job, when it is dequeued and executed by the monitor, must explicitly enqueue the “calling” process if a subroutining effect is desired. Thus along with usual arguments telling a job what data to work on, a job queue discipline implies passing of control information.

Job queues are a general implementational technique useful for simulating other types of control mechanisms, such as active knowledge (Chapter 12). Also, a job queue can be used to switch between jobs which are notionally executing in parallel, as is common in multiprocessing systems. In this case sufficient information must be maintained to start the job at arbitrary points in its execution.

An example of a priority job queue is a program [Ballard 1978] that locates ribs in chest radiographs. The program maintains a relational model of the ribcage including geometric and procedural knowledge. Uninstantiated model nodes corresponding to ribs might be called hypotheses that those ribs exist. Associated with each hypothesis is a set of procedures that may, under various conditions, be used to verify it (i.e., to find a rib). Procedures carry information about preconditions that must be true in order that they may be executed, and about how to compute estimates of their utility once executed. These descriptive components allow an executive program to rank the procedures by expected usefulness at a given time.

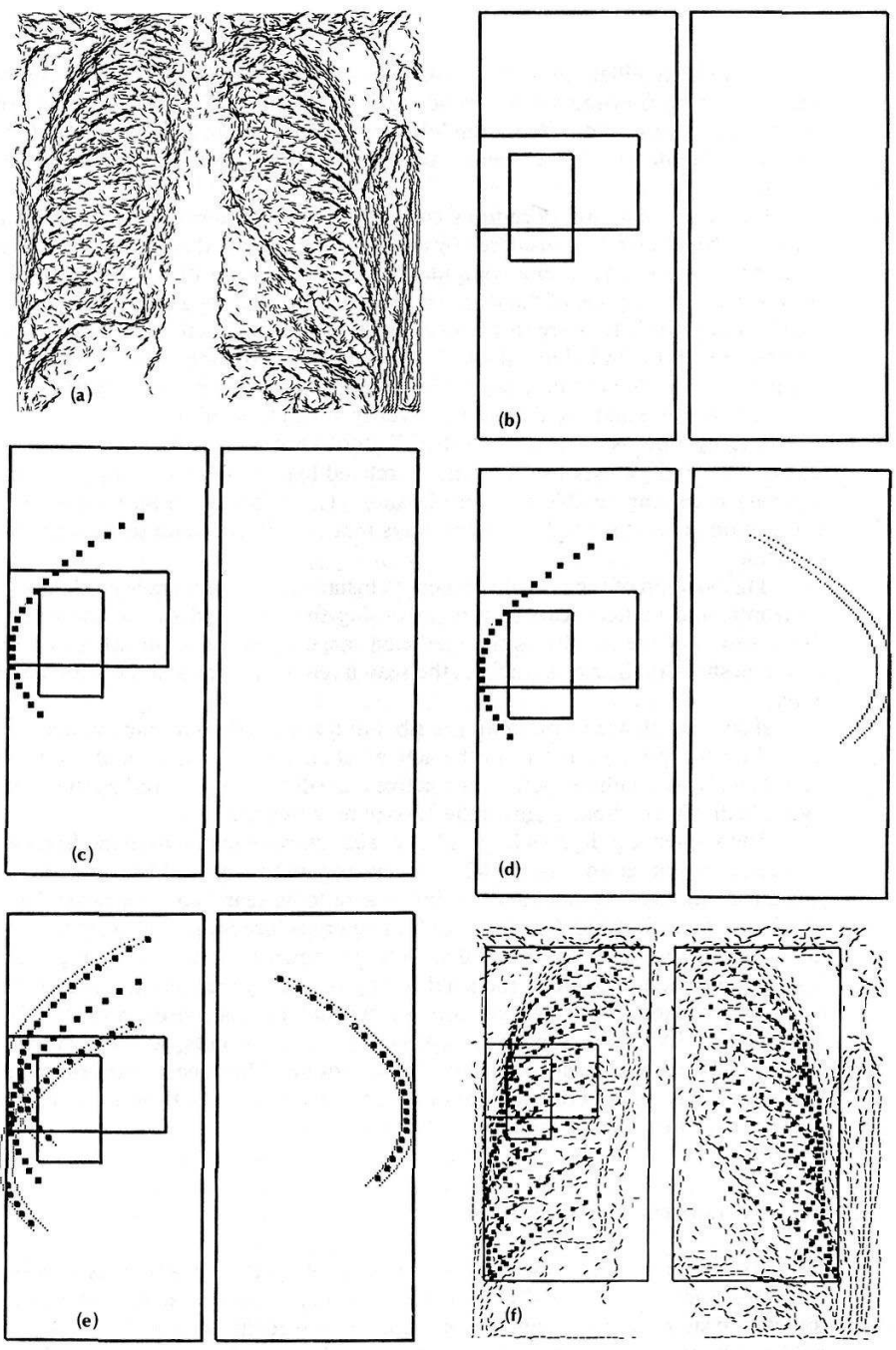


Fig. A2.1 The rib-finding process in action (see text).

There is an initial action that is likely to succeed (locating a particular rib that is usually obvious in the x-ray). In heterarchical fashion, further actions use the results of previous actions. Once the initial rib has been found, its neighbors (both above and below and directly across the body midline) become eligible for consideration.

Eligible rib-finding procedures correspond to short-term plans; they are all put on a job queue to be considered by an *executive* program that must compute the expected utility of expending computational energy on verifying one of the hypotheses by running one of the jobs. The executive computes a priority on the jobs based on how likely they are to succeed, using the utility functions and parameters associated with the individual nodes in the rib model (the individual hypotheses) and the current state of knowledge. The executive not only picks a hypothesis but also the procedure that should be able to verify it with least effort.

The hypothesis is either “verified,” “not-verified,” or “some evidence is found.” Verifying a hypothesis results in related hypotheses (about the neighboring ribs) becoming eligible for consideration. The information found during the verification process is used in several ways that can affect the utility of other procedures.

The position of the rib with respect to instantiated neighbors is used to adjust horizontal and vertical scale factors governing the predicted size of the ribcage. The position of the rib affects the predicted range of locations for other unfound ribs. The shape of the rib also affects the search region for uninstantiated rib neighbors.

If some evidence is found for the rib, but not enough to warrant an instantiation, the rib hypothesis is left on the active list and the rib model node is not instantiated. Rib hypotheses left on the active list will be reconsidered by the executive, which may try them again on the basis of new evidence.

The sequence of figures (Fig. A2.1, p. 503) shows a few steps in the finding of ribs using this program. Figure A2.1a shows the input data. A2.1b shows rectangles enclosing the lung field and the initial area to be searched for a particular rib which is usually findable. Only one rib-finding procedure is applicable for ribs with no neighbors found, so it is invoked and the rib shown by dark boxes in Fig. A2.1b is found. Predicted locations for neighboring ribs are generated and are used in order by the executive which invokes the rib-finding procedures in order of expected utility (A2.1c-e). Predicted locations are shown by dots, actual locations by crosses; in Fig. A2.1f, all modelled ribs are found. The type of procedure that found the rib is denoted by the symbol used to draw in the rib. Figure A2.1f shows the final rib borders superimposed on the data.

A2.3.3 Pattern Directed Invocation

Considerable attention has been focused recently on pattern directed systems (see, e.g., [Waterman and Hayes-Roth 1978]). Another common example of a pattern directed system is the production system, discussed in Section 12.3. The idea behind a pattern directed system is that a procedure will be activated not when its

name is invoked, but when a key situation occurs. These systems have in common that their activity is guided by the appearance of “patterns” of data in either input or memory. Broadly construed, all data forms patterns, and hence patterns guide any computation. This section is concerned with a definition of patterns as something very much smaller than the entire data set, together with the specification of control mechanisms that make use of them.

Pattern directed systems have three components.

1. A data structure or data base containing modifiable items whose structure may be defined in terms of patterns
2. Pattern-directed modules that match patterns in the data structure
3. A controlling executive that selects modules that match patterns and activates them

A popular name for a pattern-directed procedure is a *demon*. Demons were named originally by Selfridge [Selfridge 1959]. They are used successfully in many AI programs, notably in a natural language understanding system [Charniak 1972]. Generally, a demon is a program which is associated with a *pattern* that describes part of the knowledge base (usually the pattern is closely related to the form of “items” in a data base). When a part of the knowledge base matching the pattern is added, modified, or deleted, the demon runs “automatically.” It is as if the demon were constantly watching the data base waiting for information associated with certain patterns to change. Of course, in most implementations on conventional computers, demons are not always actively watching. Equivalent behavior is simulated by having the demons register their interests with the system routines that access the data base. Then upon access, the system can check for demon activation conditions and arrange for the interested demons to be run when the data base changes.

Advanced languages that support a sophisticated data base often provide demon facilities, which are variously known as if-added and if-removed procedures, antecedent theorems, traps, or triggers.

A2.3.4 Blackboard Systems

In artificial intelligence literature, a “blackboard” is a special kind of globally accessible data base. The term first became prominent in the context of a large pattern directed system to understand human speech [Erman and Lesser 1975; Erman et al. 1980]. More recently, blackboards have been used as a vision control system [Hanson and Riseman 1978]. Blackboards often have mechanisms associated with them for invoking demons and synchronizing their activities. One can appreciate that programming with demons can be difficult. Since general patterns are being used, one can never be sure exactly when a pattern directed procedure will be activated; often they can be activated in incorrect or bizarre sequences not anticipated by their designer. Blackboards attempt to alleviate this uncertainty by controlling the matching process in two ways:

1. Blackboards represent the current part of the model that is being associated with image data;

- Blackboards incorporate rules that determine which specialized subsystems of demons are likely to be needed for the current job. This structuring of the data base of procedures increases efficiency and loosely corresponds to a “mental set.”

These two ideas are illustrated by Figs. A2.2 and A2.3 [Hanson and Riseman 1978]. Figure A2.2 shows the concept of a blackboard as a repository for only model-image bindings. Figure A2.3 shows transformations between model entities that are used to select appropriate groups of demons.

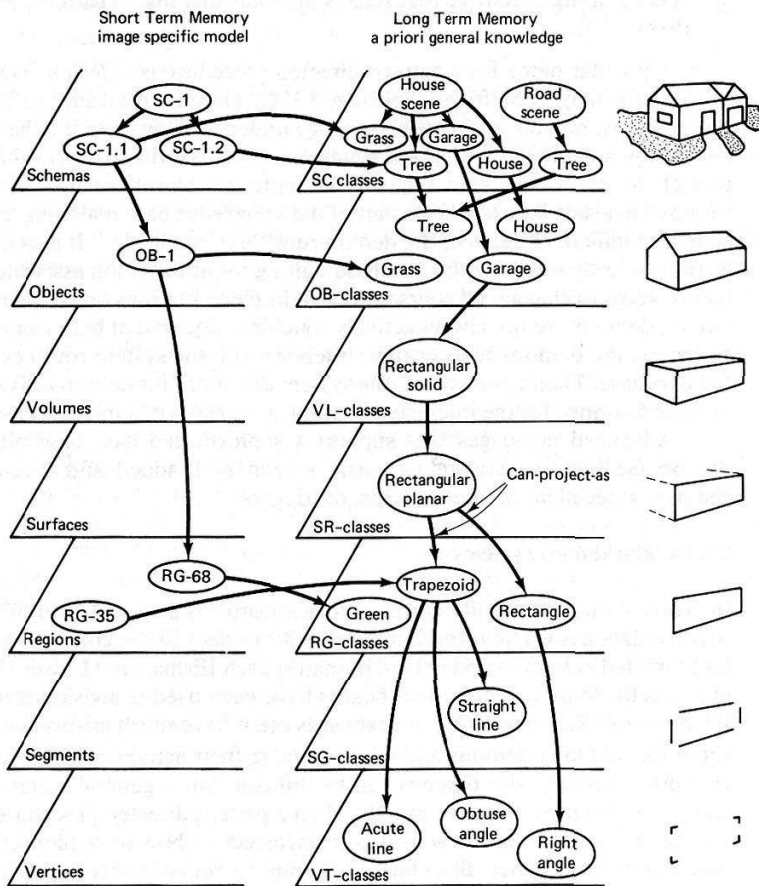


Fig. A2.2 An implementation of the blackboard concept. Here the blackboard is called Short Term Memory; it holds a partial interpretation of a specific image.

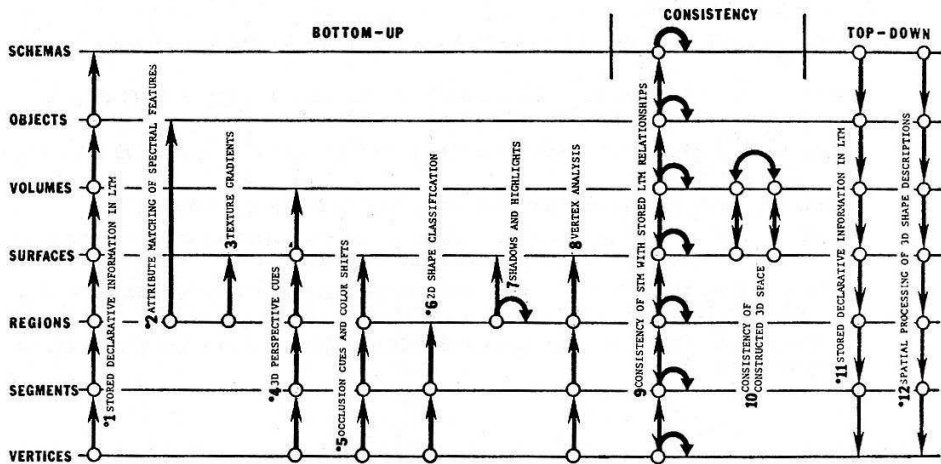


Fig. A2.3 Paths for hypothesis flow, showing transformations between model entities and the sorts of knowledge needed for the transformations.

REFERENCES

- AHO, A. V., J. E. HOPCROFT and J. D. ULLMAN. *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.
- ASTRAHAN, M. M. et al. "System R: A relational approach to data base management." IBM Research Lab, February 1976.
- BALLARD, D. H. "Model-directed detection of ribs in chest radiographs." Proceedings, Fourth IJCP, Kyoto, Japan, 1978.
- BIRTWHISTLE, G. et al. *Simula Begin*. Philadelphia: Auerbach, 1973.
- BOBROW, D. G. and B. RAPHAEL. "New programming languages for artificial intelligence." *Computing Surveys* 6, 3, September 1974, 155-174.
- CHARNIAK, E. "Towards a model of children's story comprehension." AI-TR-266, AI Lab, MIT, 1972.
- ERMAN, L. D. and V. R. LESSER. "A multi-level organization for problem solving using many diverse cooperating sources of knowledge." *Proc.*, 4th IJCAI, September 1975, 483-490.
- ERMAN, L. D., F. HAYES-ROTH, V. R. LESSER, and D. R. REDDY. "The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty." *Computing Surveys* 12, 2, June 1980, 213-253.
- FAHLMAN, S. E. "A planning system for robot construction tasks." *Artificial Intelligence* 5, 1, Spring 1974, 1-49.
- FELDMAN, J. A. "High-level programming for distributed computing." *Comm. ACM* 22, 6, July 1979, 363-368.
- FELDMAN, J. A. and R. F. SPROULL. "System support for the Stanford hand-eye system." *Proc.*, 2nd IJCAI, September 1971, 183-189.
- FLOYD, R. W. "The paradigms of programming." *Comm. ACM* 22, 8, August 1979, 455-460.

- GOLDBERG, A. and A. KAY (Eds). "SMALLTALK-72 Instruction Manual." SSL 76-6, Xerox PARC, Palo Alto, CA, 1976.
- HANSON, A. R. and E. M. RISEMAN. "Visions: A computer system for interpreting scenes." In *CVS*, 1978.
- HEWITT, C. "Description and theoretical analysis (using schemata) of PLANNER" (Ph.D. dissertation). AI-TR-258, AI Lab, MIT, 1972.
- HEWITT, C. and B. SMITH. "Towards a programming apprentice." *IEEE Trans. Software Engineering*, 1, 1, March 1975, 26-45.
- KNUTH, D. E. *The Art of Computer Programming*, Vol. 1. Reading, MA: Addison-Wesley, 1973.
- SELFRIDGE, O. "Pandemonium, a paradigm for learning." In *Proc., Symp. on the Mechanisation of Thought Processes*, National Physical Laboratory, Teddington, England, 1959.
- SUSSMAN, G. J. and D. McDERMOTT. "Why conniving is better than planning." AI Memo 255A, AI Lab, MIT, 1972.
- WATERMAN, D. A. and F. HAYES-ROTH (Eds.). *Pattern-Directed Inference Systems*. New York: Academic Press, 1978.

Author Index

- Abdou, I. E., 76, 77, 84
Abelson, R. P., 322, 334
Abramowitz, M., 496
Adams, J. A., 496
Aggarwal, J. K., 208, 216, 220
Agin, G. J., 52, 54, 277, 278
Aho, A. V., 359, 502
Aikens, J. S., 407
Akatsuka, T., 79
Ambler, A. P., 237, 346, 359, 366, 370, 375, 439
Anderson, J. R., 320
Andrews, H. C., 40, 65, 496
Apostol, T. M., 496
Ashkar, G. P., 133, 137
Astrahan, M. M., 501
Attneave, F., 75
- Badler, N. I.**, 216, 217, 219, 220, 280
Bajcsy, R., 184, 188, 400, 401, 274, 280
Ballard, D. H., 125, 128, 136, 139, 141, 244, 270, 271, 272, 273, 321, 335, 344, 355, 439, 446, 502
Barnard, S. T., 208
Barnhill, R. E., 239, 242, 265
Barrow, H. G., 63, 141, 161, 238, 318, 354, 362, 376, 410, 413
Bartlett, F. C., 343
Baumgart, B. G., 266
Bellman, R., 137
Bentley, J. L., 281
Berge, C., 357
Berziss, A. T., 358
Binford, T. O., 89, 274, 335
Birney, 34
- Birtwhistle, G., 501
Bittner, J. R., 364
Blinn, J., 95
Blum, H., 252
Bobrow, D. G., 334, 501
Boggess, L., 320
Bolles, R., 121, 319, 343, 446
Bower, G. H., 320
Boyse, J. W., 282, 285, 289
Brachman, R. J., 323, 326, 331, 396
Braunstein, M. L., 206
Bribiesca, E., 259
Brice, C., 157, 158, 236
Brodatz, P., 166, 186
Bron, C., 359
Brooks, R. A., 335
Brown, C., 255, 271, 285, 430, 496
Buchanan, B. G., 407
Buckhout, R., 343
- Chakravarty, I.**, 301
Charniak, E., 505
Chasen, S. H., 495
Chen, C. C., 80
Chien, Y. P., 136, 142
Chow, C. K., 152
Clinton, J. D., 271, 496
Clocksin, W. F., 196, 202, 206
Clowes, M. B., 296
Collins, A., 384
Connors, R., 170
Conte, S. D., 496
Coons, S. A., 269

Corneil, D. G., 358, 359
Cover, T. M., 182
Crowther, R. A., 59

Davis, L. S., 378, 408
Davis, R., 398
deBoor, C., 239, 242, 496
DeGroot, M. H., 445
DeRosier, D. J., 57, 59
Deliyanni, A., 390
Doyle, J., 347
Dreyfus, S., 137
Duda, R. O., 30, 123, 144, 208, 220, 234, 496

Eisenbeis, S. A., 155
Ejiri, J., 292
Elliott, G. L., 365
Elschlager, R. A., 142, 360
Erman, L. D., 400, 505

Fahlman, S. E., 322, 439, 500
Falk, G., 292
Faux, I. D., 496
Feigenbaum, E. A., 407
Feldman, J. A., 157, 161, 163, 412, 439, 445, 446, 500, 501
Fennema, C. L., 157, 158, 236
Fikes, R. E., 395, 396, 397, 439
Findler, N. V., 323
Fischler, M. A., 141, 360
Floyd, R. W., 499
Fodor, J. D., 320
Forrest, A. R., 265
Freeman, H., 211, 235, 236
Frei, W., 80
Freuder, E. C., 322, 413, 430
Fu, K. S., 136, 142, 172, 173, 176, 181, 238
Fukunaga, K., 181
Funt, B. V., 322

Gallus, G., 236
Garfinkel, R. S., 429
Garvey, T. D., 54, 319, 439, 446, 449, 452, 453
Gelernter, H., 322
Geschwind, N., 342
Gibson, J. J., 168, 189, 196
Gips, J., 173
Goldberg, A., 501
Goldstein, I. P., 334
Gombrich, E. H., 343
Gomory, R. E., 429
Gonzalez, R. C., 25, 65, 181
Gordon, R., 56
Gordon, W. J., 243
Gotlieb, C. C., 358, 359

Graliam, M., 206
Gramiak, R., 54
Granrath, D. J., 31
Greenberg, D., 33
Gregory, R. L., 343
Griffith, A. K., 81
Guzman, A., 259, 294

Hall, E. L., 186
Hannah, M. J., 68, 89
Hanson, A. R., 111, 150, 153, 161, 505, 506
Haralick, R. M., 184, 186, 360, 365, 408, 410
Harary, F., 357
Harlow, C. A., 155
Hart, P. E., 30, 123, 144, 208, 234, 496
Hayes, P. J., 334, 384, 393, 396, 440
Hayes, Philip J., 331, 334
Hayes-Roth, F., 397, 399, 504
Helmholtz, H. von, 196, 319, 348
Hendrix, G. G., 323, 331, 334, 396
Henschen, L., 434
Herbrand, J., 389
Herman, G. T., 56, 146
Hewitt, C., 322, 396, 397, 499, 500
Hinton, G. E., 335, 420, 425
Hodgeman, C. D., 496
Hopcroft, J. E., 172, 359
Horn, B. K. P., 22, 23, 74, 93, 95, 104, 196, 496
Horowitz, S. L., 157, 233
Hough, P. V. C., 124
Hubel, D. H., 80
Hueckel, M. H., 76
Huffman, D. A., 296
Hummel, R. A., 82, 420, 430
Hunt, B. R., 65, 40
Hurvich, L. M., 34

Ikeuchi, K., 93, 99, 100

Jackins, C. L., 281
Jain, A. K., 18
Jain, R., 222
Jameson, D., 34
Jayaramamurthy, S. N., 179
Joblove, G. H., 33
Johansson, G., 210, 215
Johnson-Laird, P. N., 319, 320, 321
Joshi, A. K., 400
Julesz, B., 169

Kak, A. C., 17, 25, 39, 76, 144, 153, 252, 496
Kanade, T., 300, 306
Kane, J., 496
Kaneko, T., 152

Kartus, J. S., 410
Kay, A., 501
Kelly, M. D., 121
Kender, J. R., 33, 169, 191
Kerbosch, J., 359
Kibler, D. P., 233
Kimme, C., 124, 125
King, J., 398
Kirsch, R. A., 79
Klug, A., 59
Knodel, W., 359
Knuth, D. E., 108, 498
Kosslyn, S. M., 320
Kowalski, R. A., 388, 390, 394, 396
Kruger, R. P., 186, 258

Lakatos, I., 267
Land, E. H., 34
Lantz, K. A., 125
Laws, K. I., 166, 185, 186
Lawton, D. T., 196, 199, 213, 214
Lee, D. N., 196, 206
Lee, Y. T., 287
Lehnert, W., 334
Lesser, V. R., 400, 505
Lester, J. M., 133, 136
Levine, M. D., 111
Lieberman, L., 184, 188
Lindsay, R. K., 407
Lishman, J. R., 196, 206
Liu, H. K., 82, 146
Loomis, J. M., 196
Loveland, D., 388, 390
Lozano-Perez, T., 319
Lu, S. Y., 173, 176

Mackworth, A. K., 291, 295, 301, 303, 439
Maleson, J. T., 188
Markovsky, G., 211, 289, 292
Marr, D., 90, 91, 93, 119, 252, 282
Martelli, A., 132, 143, 145
Martin, W. N., 220
McCarthy, J., 384, 395, 396, 440
McCulloch, W. S., 344
McDermott, D., 322, 325, 394, 396, 499
Mendelson, E., 383
Mero, L., 76
Merrill, R. E., 248
Milgram, D. L., 178
Minsky, M. L., 334, 335, 400
Mitchell, T. M., 407
Modestino, J. W., 133, 137
Montanari, U., 139
Moore, J., 334
Moore, R. C., 330

Moravec, H. P., 69, 89, 107, 208
Munsell, A. H., 33

Nagel, H.-H., 222
Nahin, P. J., 256
Nakayama, K., 196
Neisser, U., 343
Nemhauser, G. L., 429
Neurath, P. W., 236
Nevatia, R., 76, 335, 370, 372
Newell, A., 334, 390
Newman, W. M., 495
Nicodemus, F. E., 23
Nilsson, N. J., 132, 157, 320, 323, 331, 365, 387, 388, 390,
396, 397, 440, 446
Nishihara, H. K., 264, 282
Nitzan, D., 54

O'Connell, D. N., 210
Ohlander, R., 153
O'Neill, B., 276
O'Rourke, J., 216, 217, 280
Osteen, R. E., 359

Palmer, S. E., 320
Papert, S., 144
Paton, K. A., 239
Pavlidis, T., 109, 157, 233, 253, 254, 496
Pennington, K. S., 52
Persoon, E., 136, 238
Phong, B. T., 95
Pingle, K. K., 81
Poggio, T., 90, 91, 93
Pomerantz, J. R., 320
Popplestone, R. J., 52, 238, 362, 376
Potter, J. L., 220
Prager, J. M., 85, 196, 198, 199
Pratt, M. J., 496
Pratt, W. K., 17, 25, 65, 84, 181, 496
Prazdny, K., 196, 206
Prewitt, J. M. S., 77
Price, K. E., 221, 335
Pylyshyn, Z. W., 320

Quam, L., 68, 89
Quillian, M. R., 323

Raiffa, H., 445, 446
Ralston, A., 496
Ramer, U., 133
Raphael, B., 501
Rashid, R. F., 216, 217
Reddy, R., 221

- Reingold, E. M., 359, 364
 Reiter, R., 395, 396
 Requicha, A. A. G., 231, 254, 265, 282, 287, 289, 291
 Riesenfeld, R. F., 239, 242, 265
 Riseman, E. M., 111, 150, 153, 161, 505, 506
 Roache, J. W., 216
 Roberts, L. G., 76, 77, 234, 235, 292
 Roberts, R. B., 334
 Robertson, T. V., 153
 Robinson, J. A., 383, 389
 Rogers, B., 206
 Rogers, D. F., 496
 Rosenfeld, A., 17, 25, 39, 64, 76, 144, 153, 178, 252, 408,
 415, 416, 496
 Rumelhart, D. E., 334
 Russell, D. L., 100
 Russell, D. M., 335
 Rychner, M., 400
- Sacerdoti, E. D.**, 439
 Samet, H., 249, 251
 Schank, R. C., 320, 322, 334
 Schiff, W., 196
 Schneier, M., 251
 Schubert, L. K., 323
 Schudy, R. B., 270, 271, 272, 273, 355
 Schunck, B. G., 104, 196
 Schwartz, S. P., 320
 Selfridge, P. G., 122
 Selfridge, O., 345, 505
 Shani, U., 274, 277, 279
 Shapira, R., 89, 211, 292
 Shapiro, L. G., 360, 408
 Shepard, R. N., 320
 Shirai, Y., 81, 233, 292, 346, 439
 Shortliffe, E. H., 407, 453
 Sjoberg, R. W., 22, 23, 94
 Sklansky, J., 79, 125, 136, 233, 258
 Sloan, K. R., 401
 Sloman, A., 320
 Smith, A. R., 33
 Smith, B., 501
 Smoliar, S. W., 219
 Snyder, W. E., 195
 Sobel, I., 496
 Soroka, B. I., 274
 Sproull, R. F., 439, 445, 446, 495, 500
 Stallman, R. M., 347
 Stefik, M., 334
 Stegun, I. A., 496
 Steiglitz, K., 107
 Stevens, K. A., 168, 191
 Stiny, G., 173
 Stockham, T. J., Jr., 74
- Sugihara, K., 52, 301
 Sussman, G. J., 322, 347, 396, 439, 442, 499
- Tamura, H.**, 188
 Tanimoto, S. L., 109, 281
 Tate, A., 439
 Teevan, 34
 Tenenbaum, J. M., 33, 63, 81, 161, 318, 410, 413
 Thompson, W. B., 207, 208
 Tilove, R. B., 284
 Tomek, I., 233
 Tou, J. T., 181, 359
 Tretiak, O. J., 76
 Tsotsos, J. K., 207
 Turner, K. J., 89, 234, 239, 292, 299, 301, 433
- Ullman, J. D.**, 172, 359
 Ullman, J. R., 358, 359, 360
 Ullman, S., 210, 212
- Vassy, Z.**, 76
 Voelcker, H. B., 254, 282, 285, 289, 291
- Waag, R. B.**, 54
 Wallach, H., 210
 Waltz, D. A., 299, 320
 Warnock, J. G., 287
 Warren, D. H. D., 394, 439
 Waterman, D. A., 397, 399, 504
 Wechsler, H., 79, 125, 136
 Wesley, M. A., 211, 289, 292, 319
 Weszka, J. S., 181
 Wexler, C., 496
 Weyl, S., 33
 Whitted, T., 95
 Wiesel, T. N., 80
 Wilks, Y., 334
 Will, P. M., 52
 Winograd, T., 322, 334, 384, 395, 396
 Winston, P. H., 333, 370
 Wintz, P., 25, 65
 Woodham, R. J., 93, 98
 Woods, W. A., 323
 Wu, S., 243
- Yakimovsky, Y.**, 157, 161, 163, 412, 446
 Young, I. T., 256
- Zadeh, L.**, 422, 423
 Zucker, S. W., 82, 85, 150, 172, 408, 420, 430

Subject Index

- A Algorithm, 132
- A priori probabilities in plan scoring, 449
- Abstraction in knowledge representation, 320, 505
- Acting and planning cycle, 315, 347
- Action:
 - frame problem, 444
 - plans, 441, 446
- Active:
 - imaging, 14
 - knowledge, 384, 430–434
- Algorithm:
 - A, 132
 - boundary evaluation for solids, 288–291
 - correlation by binary search, 108
 - directed graph isomorphism by backtrack search, 364
 - discrete labeling, 410–415
 - edge detection by dynamic programming, 140
 - edge detection, hierarchical, 109
 - edge relaxation, 85
 - ellipse detection with Hough algorithm, 127
 - fast Fourier transform, 490–492
 - generalized Hough algorithm, 129
 - heuristic search, 132
 - line detection with Hough algorithm, 123
 - mass properties of solids, 285–286
 - medial axis transformation, 252
 - multiframe optical flow calculation, 105
 - nondeterministic for graphs, 359
 - optical flow by relaxation, 104
 - piecewise linear curve segmentation, 234
 - quad tree generation, 250
 - region boundary melting, 159
 - region growing, semantic, 162
 - region growing by blob coloring, 151
 - region merging with adjacency graph, 160
 - region splitting, recursive, 153
 - region splitting and merging, 157
 - set membership classification, 284
 - shape from shading, 100
 - shape number calculation, 260
 - solid to surface representation conversion, 290
 - stereopsis, 91
 - strip tree curve-region intersection, 247
 - strip tree generation, 244
 - strip tree intersection, 245
 - tumor detection, 345
- Aliasing, 41
- Ambiguity in grammars, 172
- Analog-digital conversion, 50
- Analogical models (*See* Knowledge representation, analogical and proportional)
- And-or trees as plans, 453–459
- Aperiodic correlation, 67
- Applications of computer vision, 12
- Arcs in semantic nets, 324
- Area:
 - chain code, 235–236
 - cross section of generalized cone, 278
 - in location networks, 336
 - polygon, 235
 - quad trees, 251
 - region, 254
- Array grammar, 178–181
- Association graph, 358, 365–369
- Associative recall, 334
- Asynchronous relaxation, 412

- Atomic formula in logic, 384
- Attention, control of, 340
- Automated inference systems, 396

- B-spline**, 239–243
- Background subtraction:
 - low-pass filtering, 72
 - spline surface, 72
- Backtrack search, 363–365, 372–375
 - automatic, 499
 - variations and improvements, 364–365
- Backward chaining, 342, 399
- Bandlimited signal, 41
- Basis for color space, 33–35
- Bayes' rule, 449 (*See also* Decision, theory and planning)
- Bayesian decisions and region growing, 162
- Belief maintenance, 319, 346
- Bending energy of curve, 256
- Binary search correlation, 108
- Binary tree, 244
- Binocular imaging, 20–22 (*See also* Stereo vision)
- Binormal of space curve, 276
- Blackboard, 505
- Blob finding, 143–146, 151
- Block stacking, 322, 438–443
- Blocks world:
 - vision, 291
 - structure matching, 370–372
- Bottom-up (*See* Control; Inference)
- Boundary, 75, 265
 - conditions for B-splines, 241
 - detection, 119–148
 - in binary images, 143
 - divide and conquer, 122
 - dynamic programming, 137–143
 - Hough algorithm, 123–131
 - evaluation, 288–291
 - as graph, 131
 - representations, 232–247, 265–274
- Branch and bound search:
 - backtracking improvement, 364
 - for boundaries, 136
- Breakpoints in linear segmentation, 232

- Calculus, predicate** (*See* Predicate logic)
- Camera model and calibration, 481–484
- Cartesian coordinate system, 465
- CAT imagery, 1, 56–59
- Cell decomposition volume representation, 281
- Centroid of volume, 285
- Chain code, 235–237, 256, 258
 - area calculation, 236
 - derivative, 236
 - merging, 236
 - normalized, 236
- Chamfer matching, 354
- Charge transfer devices, 49
- Chessboard metric, 39
- Chest radiograph understanding, 321, 344–346
- Chromaticity diagram, 37
- Chunks of knowledge, 334
- Circular arcs, 237
- City block metric, 38
- Classification:
 - in pattern recognition, 181–184
 - set membership, 284
 - tree for regions, 163
- Clause form of predicate logic, 384
- Clique, and use in matching, 358, 366–369, 375
- Closed curves, 246
- Closure operator for sets, 282
- Clustering:
 - motion detection, 217
 - parametric and non-parametric for pattern recognition, 181–183
- Co-routining, 498 (*See also* Control)
- Coherence:
 - of knowledge representation, 320
 - rule for line-drawing interpretation, 297
- Collision detection with optic flow, 201
- Color, 31–35
 - bases, 33–34
 - space histograms, 153–155
- Comb, dirac, 19, 40
- Combining operators for volumes, 282
- Compactness of region, 256
- Completeness of inference system, 389
- Complexity of graph algorithms, 359
- Component, r-connected, 369, 380
- Computer as research tool, 9
- Concave line label, 296
- Concavity tree of region, 258
- Cone, generalized (*See* Generalized, cone)
- Confidence:
 - planning, 415 (*See also* Supposition value in relaxation)
 - region growing, 164
- Conic, 239, 488–489
- Conjunctive normal form for logic, 388
- Connect line label, 303
- Connected:
 - component of graph, 369, 380
 - region, 150, 255
- Connectives of logic, 385
- Connectivity:
 - difference, 375
 - image, 36
 - matching, 372–375
- CONNIVER, 322
- Consolidation 102, (*See also* Pyramid)
- Constraint (*See also* Relaxation)

- inconsistency, 427
- as inequality in linear programming, 423
- labeling, 408–410
- n-ary, 410
- propagation, 299, 413–415
- relaxation, 408–430
- satisfaction for belief maintenance, 347
- semantic, on region-growing, 160–164
- Constructive solid geometry volume
 - representation, 282
- Context:
 - data base, 440
 - switching, 500
- Continuity of knowledge representation, 320
- Contour:
 - following, 143–146
 - occluding, 101
- Contrast enhancement, 71
- Control, 315, 340–350, 497–502
 - bottom-up or data-driven, 341, 344–346
 - hierarchical and heterarchical, 341–346
 - in knowledge representations, 317
 - message passing, 501
 - mixed top-down and bottom-up, 344–346
 - structures, standard and nonstandard, 497–500
 - top-down, 342, 343–346
- Convergence of relaxation algorithms, 414, 418
- Conversions, logic to semantic nets, 332
- Convex:
 - decomposition of region, 253
 - line label, 296
 - region, 258
- Convolution, 25, 68
 - theorem, 30
- Cooperative algorithms, 408–430
- Coordinate systems, definitions and conversions, 465–468
- Correctness of inference system, 389
- Correlation, 25, 30, 66–70
 - binary search, 108
 - coefficient, 419
 - metrics, 362
 - non-linear for edge linking, 121
 - normalized, 68–70
 - periodic and aperiodic, 67
 - texture, 187
- Correspondence problem, 89
- Cost:
 - of planning, 452
 - in plans, 445–459
- Crack edges, 78
- Curvature:
 - boundary, 256
 - in evaluation function, 133
 - space curve, 276
- Curve 231:
 - detection, Hough algorithm, 126
 - fitting, 487
 - intersection, 247
 - segmentation techniques, 233–234
- Cutting planes in linear programming, 428
- Cylinder, generalized, 274–280
- Cylindrical coordinate system, 466
- Data:**
 - base, 398, 431, 440
 - driven control, 341–346
 - fitting, 239, 484–488
 - nodes in location networks, 336, 338
 - structure for boundaries, 158
- Decision:
 - theory and planning, 446–453
 - trees for matching, 370–377
- Decomposition:
 - region, 253
 - solid, 287
- Default values in knowledge representations, 330, 334–335
- Delete list, 440
- Delta function, 18–19, 40
- Demon, 412, 429, 505
- DeMorgan's laws, 387
- Densitometer, 46
- Density of image, 44, 74
- Dependence, gray-level, 186–188
- Depth:
 - first search and variations, 136, 363–365, 372, 412
 - from optic flow, 201
- Determinant, 473
- Difference measurement in motion, 221
- Digital images, 35–42
- Digitizers, image, 45
- Dirac Comb, 19, 40
- Direction-magnitude sets, 270
- Discrete:
 - images, 35–42
 - knowledge representation, 320
 - labeling algorithms, 410–415
- Disparity, 21, 89, 208
- Dispersion of knowledge representation, 320
- Distance:
 - on discrete raster, 36
 - image (*See* Image, range)
- Distortion, perspective (*See* Projection, perspective)
- Divergence theorem for mass properties, 288
- Divide and conquer:
 - algorithms for CSG, 285
 - method for boundary detection, 122
- Domain-dependent and -independent motion
 - understanding, 196–199, 214–219
- Drum scanner, 46

- Dual graph, 159
- Dynamic programming and search, 137-143
- Early processing**, 63-65
- Eccentricity of region, 255
- Edge, 75
 - detection
 - in binary images, 143-146
 - from optic flow, 202-206
 - in pyramids, 109
 - following, 131-146
 - as blob finding, 143-146
 - as dynamic programming, 137-143
 - as graph search, 131-143
 - labels, 296-297
 - linking, 119-131
 - known approximate location, 121-122
 - problems with, 119-120
 - Hough algorithm, 123-131
 - operator, 64, 75-88
 - gradient, 76-80
 - Kirsch, 79
 - Laplacian, 76-79
 - performance, 77, 83-84
 - relaxation, 85-88
 - templates, 79
 - 3-D performance, 81-83
 - profiles, 75
 - representation for surfaces, 266
 - strength in evaluation function, 133
 - thresholding, 80
- Eigenvalues and eigenvectors, 473, 486-487
- Element, texture, 166
- Elongation of region, 255
- Enclosing surface, 265
- Energy, texture, 187
- Engineering:
 - drawings, 291
 - knowledge, 407
- Entropy, texture, 187
- ERTS imagery, 46
- Euclidean metric, 38
- Euler number of region, 255, 266
- Evaluation:
 - function for heuristic search, 133
 - mechanism in semantic networks, 337
- Existential quantifiers, 385
- Extended inference, 315, 319, 322, 383, 395-396
- Extensional concepts in knowledge representation, 328
- Faces**, 271
 - for surface representation, 265, 271
- Feature
 - classification and matching, 376-378
 - texture, 184-186
 - vectors and space, 181
- Field, television, 46
- Figure-ground distinction, 4
- Filtering, 25, 64-75
- First order predicate logic (*See* Predicate logic)
- Fitting data (*See* Data, fitting)
- Flat-bed scanner, 46
- Flying spot scanner, 45
- Focal length, 19, 479
- Focus of expansion in optical flow, 199
- Formal inference system, 390
- Forward chaining, 342, 399
- Fourier
 - descriptors, 238
 - filtering, 65
 - transform, 24-30, 490-492
- Frame:
 - problem, 395, 444
 - system theory, 334-335
- Frenet frame and formulae, 276
- Function:
 - image, 18-19
 - logic, 385
 - Skolem, 387
- G-Hough algorithm**, 128
- Gamma, film, 45
- Gaussian sphere, 101, 270
- Generalized:
 - clipping, 284
 - cone, 274-280
 - matching to data, 278, 372-375
 - cylinder (*See* Generalized, cone)
 - image, 6, 14, 320
- Geodesic tessellation, 271, 493
- Geometric:
 - matching, 354
 - operations in location networks, 336
 - relations and propositions, 332
 - representations, 8, 227-311
 - structures and matching, 354
 - transformations, 477-481
- Geometry theorem prover, 322
- Gestalt psychology, 116
- Goal achievement, 319, 346-347, 438-439
- Goodness of fit, 273
- Gradient:
 - edge operator, 76-80
 - space, 95, 301
 - techniques, 355
 - texture, 168, 189-193
 - use in Hough algorithm, 124
- Grammar:
 - ambiguous, 172

- array, 178–181
 - on pyramid, 179
 - shape, 173–174
 - stochastic, 172
 - texture, 172–181
 - tree, 175–178
- Graph 131:
 - adjacency for regions, 159
 - algorithms, complexity, 359
 - association, 358, 365–369
 - dual, 159
 - isomorphism, 357–359, 364
 - matching, 355
 - r-connected component, 369, 380
- Gray level, 18, 23, 35
 - dependence matrices for texture, 186–188
 - Grazing incidence, 111
- H&D curve**, 44
- Heart volume, 273
- Heterarchical control, 341–346, 499
- Heuristic search:
 - boundaries, 131–133
 - dynamic programming, 143
 - region growing, 157
- Hierarchical (*See also* Pyramid)
 - abstractions, 505
 - control, 341–346
 - textures, 170
- High-level:
 - models, 317
 - motion detection, 196–199
 - vision, 2–6
- Hill-climbing and matching, 355
- Histogram, 70
 - equalization and transformation, 70
 - splitting for thresholds, 152
 - color space, 153–155
- Homogeneous:
 - coordinate system, 467
 - regions, 150
 - texture, 188
- Hough algorithm, 123–131
 - generalized, 128–131
 - refinements, 124
 - vanishing points, 191
- Human body for motion understanding, 214–219
- Hungry monkey planning problem, 445
- Hypotheses, 343, 384, 422
 - active knowledge, 431
- Hypothetical worlds, 432, 440
- Iconic structures and matching**, 353
- Icosahedron, 492
- IHS color basis, 33
- Image (*See also* Imaging)
 - aerial, 1, 335
 - CAT, 1, 56–59
 - connectivity, 36
 - digital, 35–42
 - digitizers, 45
 - distance on raster, 36
 - edges, 75–88
 - ERTS or LANDSAT, 46
 - formation, 17
 - function, 18–19
 - generalized, 6, 14, 320
 - histogram, 70
 - intrinsic, 7, 14, 63
 - irradiance, 23, 73
 - orthicon, 47
 - plane, 19
 - processing, 2, 17, 25
 - range, 52–56, 64, 88
 - sampling, 18, 35
 - segmented, 7
 - sequence understanding, 207–222
 - ultrasound, 54
 - variance, 69
- Imaging:
 - active, 14
 - devices, 42–59
 - geometry, 19–21
 - light stripe, 22
 - model, 17–42
 - monocular and binocular, 19–22
 - stereo, 20–22, 52–54, 88–93, 98
- Inconsistent labeling, 410 (*See also* Labeling)
- Indexing property of semantic nets, 324
- Inequalities in linear programming, 422, 427
- Inference, 314, 319–321, 383
 - bottom-up and top-down, 392
 - extended, 315, 319, 322, 383, 395
 - rules of, 388
 - in semantic nets, 327
 - systems, formal and informal, 390
 - sylogistic, 321
- Infinity, point at, 20
- Informal inference system, 390
- Inheritance of properties in knowledge
 - representation, 330, 335
- Inhibitory local evidence in line drawings, 295
- Intensional concepts in knowledge
 - representation, 328
- Interaction graph for dynamic programming, 143
- Interest operator, 69, 208
- Interior operator, 282
- Interpolation, 489–490
- Interpretation:
 - matching, 352
 - region-growing, 160–164
- Interpreter
 - production system, 398–399
 - semantic net, 326, 339
- Intersection of strip trees, 244
- Interval, sampling, 35
- Intrinsic:
 - image, 7, 14, 63
 - parameters, 63

- Inverse:
 - perspective (*See* Projection)
 - relations, 331
- Irradiance, image, 23, 73
- Isomorphism, graph and subgraph, 357–359, 364
- Iterative region merging with semantics, 163

- Job queue**, 502–504

- Kd-tree**, 281, 287
- Knowledge:
 - base, 317, 318–323
 - chunking, 334
 - engineering, 407
- Knowledge representation, analogical and propositional, 9, 314, 319–322 (*See also* Active knowledge, Predicate logic, Procedural embedding, Production systems, Representation, Semantic nets)

- Labeling**, 296–301, 408–420
 - compatibilities, 415
 - consistent, inconsistent, optimal, 408, 410
 - discrete, 410–415
 - interpretation, 315, 353, 383, 408
 - lines, 296–301
 - scene, 408–430
 - tree search, 412
- Lambertian surfaces, 94
- LANDSAT imagery, 46
- Laplacian operator, 76–79
- Laser rangefinders, 54 (*See also* Image, range)
- Learning, 315
- Least-squared error fitting, 484–488
- Legendre polynomial, 272
- Light:
 - flux, 22
 - stripe imaging, 22
 - structured, 52–54
- Line:
 - detection, Hough algorithm, 123
 - drawing understanding, 265, 291–307
 - drawings for motion understanding, 220–222
 - equation, 475
 - fitting, 484–487
 - labeling, 296–301
 - labeling by relaxation, 416–421, 428
 - representations, 474–476
 - segment, 232
 - transformation, 480
- Linear:
 - structure matching, 378–380
 - transformation, 473
- Linear programming for relaxation, 420–430
- Linking edges (*See* Edge, linking)

- Local evidence in line drawings, 294
- Location networks, 335–340
- Logarithmic filtering, 73
- Logic (*See* Predicate logic)
- Long-term memory, 400
- Low-level:
 - motion detection, 196
 - vision, 3–6

- Manhattan metric**, 36
- Mass properties, 285
- Matched filtering and Hough algorithm, 124
- Matching, 315, 352, 398
 - blocks world structures, 370–372
 - clique-finding, 366–369, 375
 - complexity, 359
 - examples, 369–380
 - expectation-driven, 353
 - generalized cylinders, 372–375
 - geometric structures, 354
 - graphs, 355
 - iconic structures, 353
 - knowledge representation, 319
 - line drawings to primitives, 293
 - linear structures, 378–380
 - metrics, 360–362, 375, 378
 - metrical in line drawing understanding, 294
 - nondeterministic algorithm, 359
 - optic flow, 208
 - optimization, 354
 - pattern (*See* Pattern, matching in production systems)
 - relational structures, 353, 355, 365–372
 - rules in production system, 399–400
 - templates and springs, 360–362
 - topological, for line drawings, 293
- Matrix algebra, 471–474
- Matte reflectivity, 23
- Maximal clique, 366
- Medial axis transform, 252–253
- Membership array for region, 248
- Memory, long-term and short-term, 400
- Merging:
 - branches for backtracking improvement, 364
 - curve segmentation, 233
 - regions, 155–160
- Message-passing (*See* Modules and messages)
- Metric:
 - distance on raster, 36
 - matching, 360–362, 375, 378
- Minimum:
 - cost search for boundaries, 132
 - spanning tree for clustering, 217
- Model:
 - analogical and propositional, 319
 - driven control, 342

- human body for motion, 217–219
 - in knowledge representation, 9, 317
- Modules and messages, 500–502
- Modus Ponens and Modus Tollens, 388
- Moment of inertia, 255, 286, 473, 486
- Monocular imaging (*See* Imaging)
- Motion, 195 (*See also* Optic flow)
 - adjacency and collision detection, 201
 - body model, 214–219
 - common in sequence, 199, 208
 - consistent match, 199
 - continuity, 197
 - depth, 201
 - human body, 214–219
 - image sequences, 207–222
 - maximum velocity, 198
 - moving light displays, 214–217
 - observer, 206
 - rigid bodies, 197, 210–214
 - surface orientation and edge detection, 202
- Multi-
 - dimensional histograms, 153–155
 - modal sensor, 453–459
 - resolution images, 100–110 (*See also* Pyramid)
- Nearest-neighbor clustering**, 183
- Network:
 - interpreter, 326
 - representation, 391 (*See also* Semantic nets)
- Newton-Raphson, 493
- Node types in semantic nets, 324–329
- Noise, 65
- Nonclausal form, 385–387 (*See also* Predicate logic)
- Nondeterministic algorithms, 359
- Nonrigid:
 - body motion understanding, 214–217
 - solids, 264
- Nonstandard:
 - control structures, 499–507 (*See also* Control)
 - inference (*See* Extended inference)
- Normalized correlation, 68–70
- NP-completeness, 359
- NTSC, 34
- Object identification in line drawings**, 294
- Occluding:
 - contour, 101
 - line label, 296
- Oct-tree, 281, 287
- Office scene understanding, 453–459
- Operator (*See* Edge, Interest operator, Interior operator, Closure operator for sets, Planning Relaxation, etc.)
- Opponent processes, 33
- Optic flow, 65, 102–105, 196, 199–206
- Optical system analysis, 23
- Optimal labeling, 410
- Optimization:
 - linear programming, 424–425
 - matching, 354
- Orientation of surface (*See* Surface, orientation calculation)
- Origami world, 300
- Orthicon, image, 47
- Orthographic projection (*See* Projection)
- PANDEMONIUM**, 345
- Parallel:
 - computation, 64, 341, 360
 - iterative refinement in graph matching, 358, 378
 - iterative relaxation, 64, 412
- Parameter:
 - optimization as matching, 354
 - space, 123
- Parametric:
 - clustering, 183
 - edge models, 80–81
 - line representation, 476
- Parseval's theorem, 256
- Partial:
 - knowledge in location networks, 339
 - matches, 360–362
- Partition:
 - feature space in pattern recognition, 181
 - Fourier space, 185
 - semantic nets, 331, 391
 - space, 150
- Pattern:
 - directed invocation, 321, 504
 - matching data base (*See* Data, base)
 - matching in production systems, 399–400
 - recognition, 2, 181–184
 - texture, 166
- Performance, edge operators, 77, 83–84
- Periodic:
 - correlation, 67
 - function, 237
- Perspective (*See* Projection)
- Photography, 44–45
- Photometric stereo vision, 98
- Picture element (pixel), 36
- Piecewise polynomial, 240
- Plane:
 - curves and regions, 231
 - cutting in linear programming, 428
 - representation, 476
 - transformation, 480
- PLANNER, 322

- Planning, 314–319, 347, 438–445
 - cost of, 452
 - costs, 445–459
 - edge linking, 121–122
 - example, 453–459
 - extended and-or graphs, 451
 - problem reduction, 394
 - symbolic, 439–445
- Point:
 - at infinity, 20
 - membership, 246
 - projection (*See* Projection, perspective)
 - spread function, 28
- Polar and polar space coordinate systems, 465
- Polygon:
 - area calculation, 235
 - images for motion, 220–222
 - regions, 294
- Polyhedra, 291
- Polylines, 232–235
- Pre- and postconditions in plans, 441
- Predicate logic, 383–395
 - clauses and semantic nets, 390–392
 - decidability, 388
 - extensions (*See* Extended inference)
 - inference, 315
 - knowledge representation, 392–395
 - proof, 388
 - strengths and weaknesses, 393–394
 - syntax and semantics, 384–387
 - truth table, 386
- Predicates in location networks, 336
- Primitive:
 - solids in volume representation, 280, 282
 - volumes for line drawing understanding, 293
- Principal axes:
 - for fitting data, 473, 486
 - of inertia of region, 255
- Procedural embedding of knowledge, 321, 322, 406, 430–434 (*See also* Active, knowledge)
- Processing, image, 17
- Product of inertia of solid, 286
- Production systems, 315, 383, 396–408
 - example, 401–406
 - rule matching, 399
 - strengths and weaknesses, 406–408
- Projection:
 - inverse perspective, 481–484
 - orthographic, 20, 212
 - perspective, 19–20, 214, 479
- Proof of logic, 388–390
- Propagation of constraints (*See* Constraint)
- Property inheritance (*See* Inheritance of properties in knowledge representation)
- Propositional model (*See* Representation)
- Prototype situations, 334
- Pruning for backtracking improvement, 364
- Pseudo-inverse for fitting, 485, 487
- Psi-s curve, 237, 238, 256
- Pyramid, 15 (*See also* Quad tree; Strip trees)
 - edge detection, 109
 - grammars, 179
 - multi-resolution, 65, 106, 249, 281
 - thresholding, 155
- Quad tree**, 249–252
 - area, 251
 - generation, 250
- Quantifier, logical, 385
- Radiance and gray levels**, 22–23
- Radiograph understanding, 321, 344–346, 502–504
- Range image (*see* Image)
- Ray casting, 280
- Reciprocity failure of film, 44
- Reconstruction, two-dimensional image, 56–59
- Recursive procedure, 244, 498
- Reflectance, 93–95
 - calculation, 73–75
 - functions, common, 23, 94
 - map, 96–98
 - models, 22–24
- Region, 149–150, 231
 - data structures for, 158
 - finding, by thresholding, 152–155
 - finding, local techniques, 151
 - growing and heuristic search, 157
 - growing and semantics, 160–164
 - homogeneous, 150
 - partition, 150
 - properties, 254–261, 376–377
 - representation, 232–254
 - by boundary, 232–247
 - decompositions, 253
 - medial axis transform, 252
 - non-boundary, 247–254
 - quad trees, 249
 - spatial occupancy array, 247
 - y-axis, 248
 - splitting and merging, 155–160
- Regular:
 - sets and set operators, 231, 282–283
 - tesselation, 170
- Relational:
 - functions and dynamic programming, 142
 - models, 9, 314, 317
 - semantic nets, 325, 330, 408
 - structures and matching, 353, 355, 365–372
- Relaxation, 64 (*See also* Constraint)

- algorithms, convergence properties, 414
- asynchronous, 412
- convergence, 414
- edge operators, 85–88
- for optic flow, 208
- labelling, 408–430
- line labeling, 416–421, 428
- linear and non-linear operator example, 415–420
- as linear programming, 420–430
- optical flow, 103
- serial and parallel iterative, 412
- shape from shading, 99–102
- stereo, 89
- Representation:
 - actions in symbolic planning, 441
 - analogical and propositional, 9, 314, 319–322
 - conversion, 289
 - knowledge, 317–347 (*See* Knowledge)
 - matching, 352–355
 - predicate logic, 392–395
 - range of, 6–9
 - solids, 264
 - world in symbolic planning, 439
- Resolution
 - gray levels, 35
 - pyramids, 15, 106–110
 - multiple in thresholding, 155
 - rule of inference, 389
 - spatial, 36–37
 - texture, 169
 - theorem proving, 388–390
- Response, human spectral, 31
- Rewriting rule (*See* Rule, Grammar, Production system)
- Rib-finding, 321, 502–504
- Rigidity assumption in motion understanding, 210
- Root finding, 493
- Rotation rigid transformation, 477
- Rotational sweep, 274
- Rule:
 - based systems, 397 (*See also* Production systems)
 - inference, 388
 - production system, 398
 - rewriting, 172, 383, 398
 - for texture generation, 172–181
- Sampling:
 - image, 18, 35
 - tessellation, 35
 - theorem, 39
- Scaling matrix, 478
- Scanner, digitizing, 25
- Scene:
 - irradiance, 73
 - labeling, 408–430
- Scope of quantifier, 385
- Scoring plans, 445–459
- Search:
 - backtrack, 363–365, 372–375
 - depth-first, 412
 - graph, and region growing, 157
 - heuristic and variations, 132–136
 - tree, for labeling, 412
- Segmentation, 7, 116 (*See also* Edge; Line; Region)
- Semantic nets, 315–317, 323–340, 390, 396
 - arcs, 324
 - conversion with other representations, 332
 - examples, 334
 - indexing property, 324
 - inference, 327
 - nodes, 324, 328
 - partitions, 331
 - predicate logic, 390–392
 - relations in, 325, 330
 - semantics and partitions, 331
- Semantics:
 - of images and region growing, 160–164
 - of logic, 385
- Semi-
 - decidability of logic, 388
 - regular tessellation, 171
- Sentence of logic, 384
- Serial:
 - computation, 341
 - relaxation, 412
- Set
 - membership classification, 284
 - operations in location networks, 336
 - operations in three dimensions, 284
- Shadow line label, 296
- Shape, 228 (*See also* Three dimensional)
 - detection, Hough algorithm, 128–131
 - grammar, 173–174
 - properties of region, 254–261, 376–377
 - recognition, 228–229
 - from shading, 65, 93, 99–102
 - from texture, 189–193
- Shift theorem, 30
- Short-term memory, 400
- Signature of region, 257
- Silhouette detection, Hough algorithm, 128–131
- Similarity:
 - analysis in motion, 221
 - tree for regions, 261
- Simplex algorithm, 423
- Simulation with knowledge representation, 320
- Skeleton (*See* Medial axis transform)
- Skew:
 - symmetry, 306
 - transformation, 479

- Skilled vision, 347
- Skolem function, 387
- Slope density function of boundary, 256
- Slots in frames, 334
- Smoothing image (*See Consolidation*)
- Solid (*See Three dimensional*)
- Spaces, color (*See Color*)
- Spatial:
 - representations (*See Three dimensional*)
 - resolution, 36, 37
- Spectral response, human eye, 31
- Specular reflectivity, 23 (*See also Reflectance, functions, common*)
- Spherical:
 - coordinate system, 270, 466
 - function, 270
 - harmonic surfaces, 271–274, 355
 - trigonometry, 469
 - volume representation, 279
- Splines (*See B-spline*)
- Splitting:
 - curve segmentation, 233–234
 - regions, 155–160
- Statistical:
 - pattern recognition, 2, 181–184
 - texture model, 168
- Stereo vision, 20–22, 52–54, 88–93, 98
- Stochastic grammars, 172 (*See also Grammar*)
- Streaks and strokes, 134
- Strip trees, 244–247
- STRIPS, 396
- Structural:
 - matching, 355, 365–372
 - models of texture, 170–181
- Structured light, 52–54
- Subgoals, 343, 438 (*See also Goal achievement; Planning*)
- Subgraph isomorphism, 357–359, 375 (*See also Graph*)
- Supposition value in relaxation, 415, 419
- Surface:
 - direction-magnitude set, 270
 - functions on Gaussian sphere, 270
 - geometry and line-drawings, 301–307
 - orientation calculation, 64, 93–102, 189–193, 202–206
 - patches, 269
 - representations, 265–274
 - set of faces, 265
 - spherical harmonic, 271–274
 - from volume calculation, 288–291
- Sweep representations of solids, 274–280 (*See also Three dimensional*)
- Syllogistic inference, 321
- Symbolic planning, (*See Planning*)
- Symmetry, skew, 306
- Synchronization, 341
- T vertices**, 294
- Tables, dynamic programming, 137–139
- Tangent to space curve, 276
- Television cameras, 46–52
- Template:
 - matching, 65
 - and Springs for matching, 360–362
- Term in logic, 384
- Tessellation:
 - geodesic, 492
 - regular and semiregular for texture, 170–172
 - sampling, 35
- Texture, 166–168, 404
 - correlation, 187
 - element (texel), 166, 169, 188
 - element placement tessellations, 170–172
 - energy, frequency and spatial domain, 184–185
 - features, 187–188
 - gradient and surface orientation, 168, 189–193
 - grammars, 172–181
 - homogeneity, 188
 - pattern recognition, 181–184
 - resolution issues, 169
 - shape and vanishing point from, 189–193
 - statistical and structural models, 168, 170–181
- Theorem:
 - convolution, 25
 - divergence, 288
 - Parseval's, 26, 256
 - sampling, 39
 - shift, 30
- Theorem proving by resolution, 388–390 (*See also Predicate logic*)
- Thinning algorithms, 253 (*See also Medial axis transform*)
- Three dimensional:
 - contour following in, 146
 - decomposition, 287
 - edge operators, 81–83
 - image, 88–93
 - model, 320
 - objects, several representation, 264, 274–283
 - primitives for line drawing understanding, 293
 - shapes, 228
 - structure from image sequence, 210–214
 - volume algorithms, 284
- Threshold:
 - determination from histogram, 152
 - multi-dimensional space, 153
 - multiple resolution, 155
 - for region finding, 152–155
- Token-type distinction, 328
- Top-down (*See Control and Inference*)
- Topological connectivity and matching, 293, 372–375

Torsion of space curve, 276
 Transformation:
 geometric, 477–481
 lines and planes, 480
 Translation rigid transformation, 479
 Translational sweep, 274
 Tree:
 grammars, 175–178
 quad, 249–252
 rearrangement for backtracking improvement, 364
 search for labeling, 412
 strip, 244–247
 Triangulation (*See* Stereo vision)
 Trigonometry, plane and spherical, 468–469
 Truth table, 386
 Tumor detection, 344–346
 Turtle algorithm for blob finding, 144
 TV (*See* Television cameras)
 Two-dimensional shape (*See* Shape)
 Type-token distinction, 328

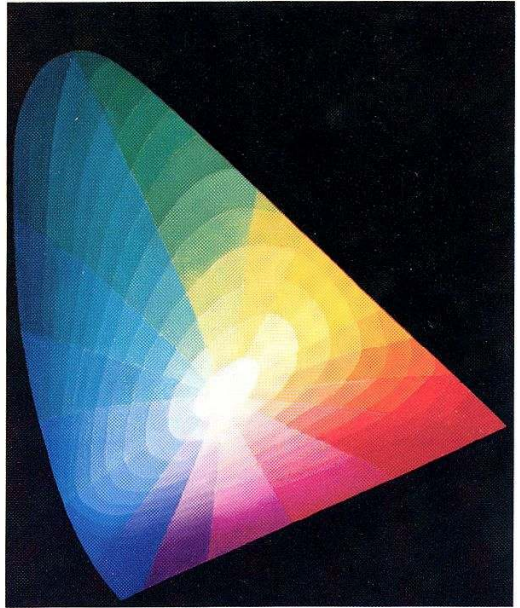
Ultrasound, 54, 273
 Unambiguous representation, 231
 Undecidability of logic, 388
 Units, meaningful, 116 (*See* Segmentation)
 Universal quantifiers, 385
 Unsatisfiability in logic, 388–389
 Utility theory and planning, 446, 453

Vanishing point from texture, 191
 Variable nodes in semantic nets, 329
 Variance, image, 69, 208
 Vector algebra, 469–471
 Velocity (*See* Motion, Optic flow)
 Vergeance, 21
 Vertex:
 catalogues, 298, 300
 types, 295
 Vidicon, 48
 Viewpoint, (*see* Projection)
 Virtual nodes in semantic nets, 328
 Vision:
 high- and low-level, 2–6
 as planning, 6–9
 system organization, 352
 Volume (*See* Three dimensional)

Waltz filtering, 299 (*See also* Constraint; Labeling)
 Well-formed formulae of logic, 385
 Winged edge for surface representation, 266–267
 Wire frame objects, 291
 from projections, 211
 World states in planning, 439

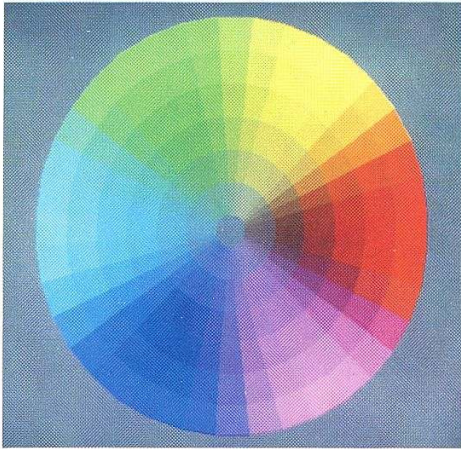
Y-axis region representation, 248
 YIQ color basis, 34

FIG. 2-7a



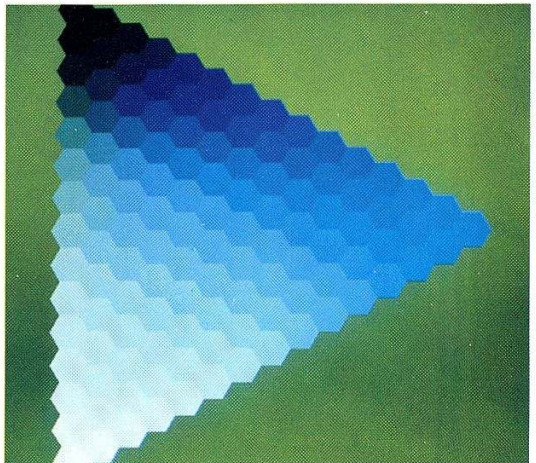
A painting by Louis Condax; courtesy of Eastman Kodak Company and the Optical Society of America.

FIG. 2-8a



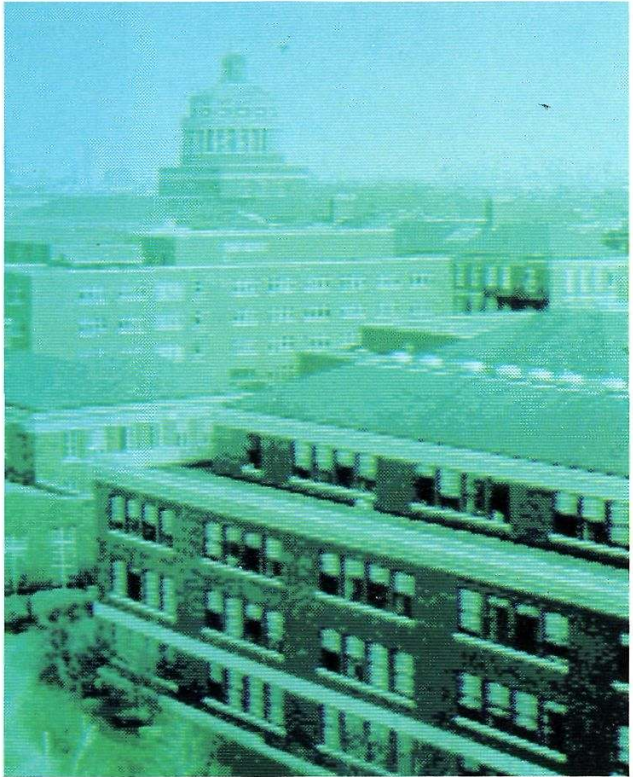
Courtesy of D. Greenberg and G. Joblove, Cornell Program of Computer Graphics.

FIG. 2-8b



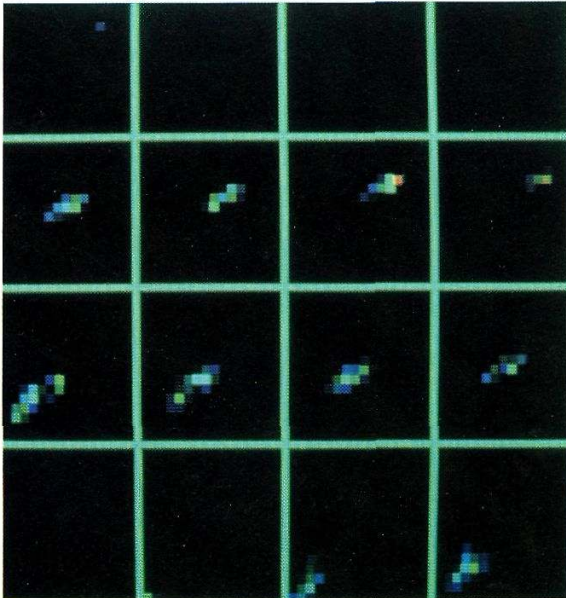
Courtesy of Tom Check.

FIG. 5-4a



Courtesy of Sam Kapilivsky.

FIG. 5-4b



Courtesy of Sam Kapilivsky.

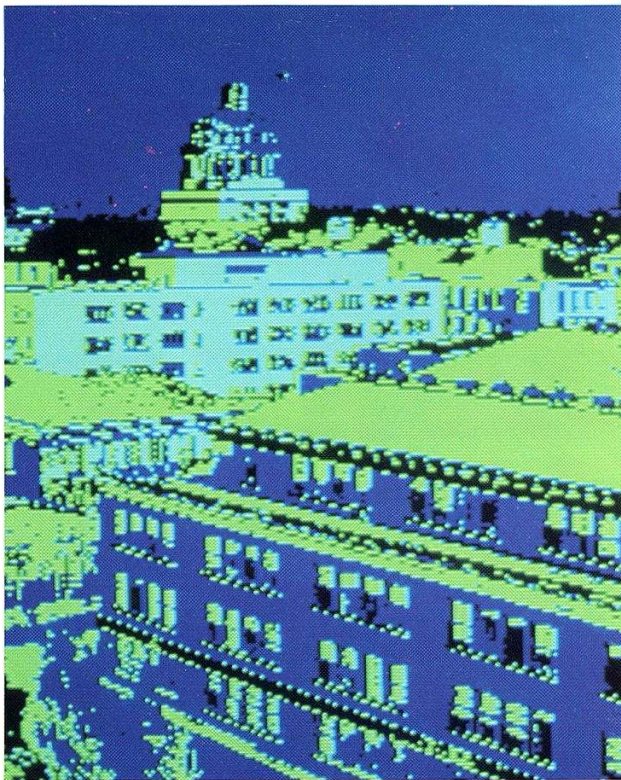
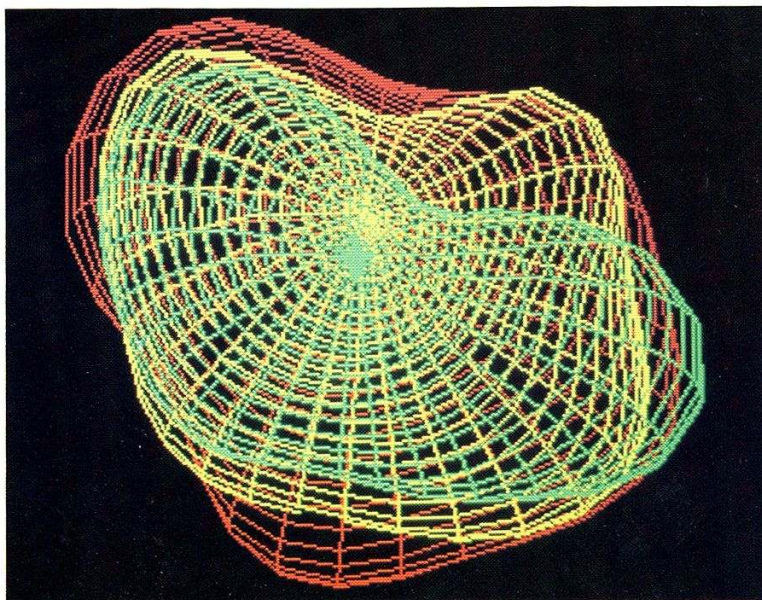


FIG. 5-4c

Courtesy of Sam Kapilivsky.

FIG. 9-10



Courtesy of Robert Schudy.

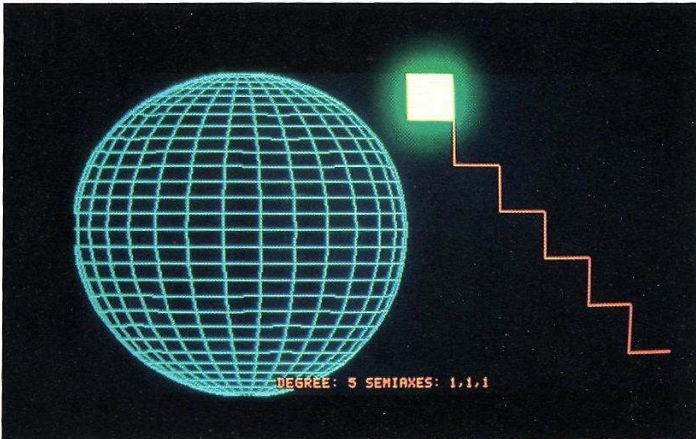
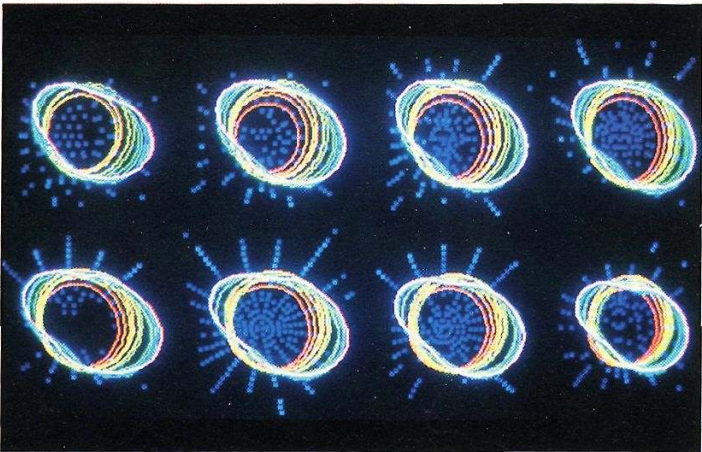


FIG. 11-3a

Courtesy of Robert Schudy.

FIG. 11-3b



Courtesy of Robert Schudy.

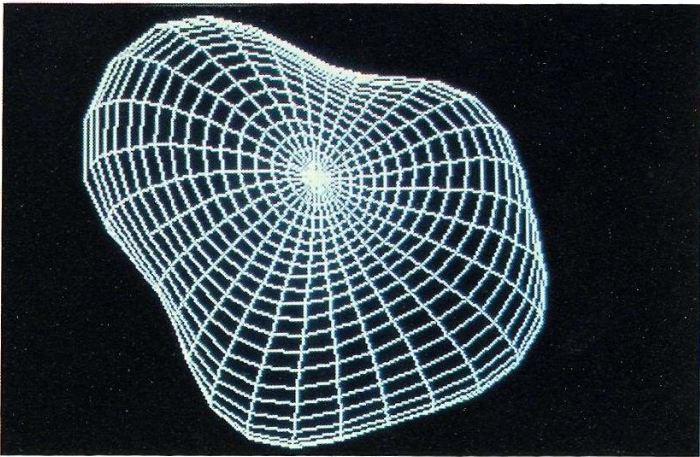


FIG. 11-3c

Courtesy of Robert Schudy.

COMPUTER VISION

DANA H. BALLARD • CHRISTOPHER M. BROWN

What information about scenes can be extracted from an image using only basic assumptions about physics and optics?

How are images segmented into meaningful parts?

At what stage must domain-dependent, prior knowledge about the world be incorporated into the understanding process?

How are world models and conceptual knowledge represented and used?

These and many other questions, inherent in this relatively new and fast-growing field, are explored and answered in **Computer Vision** by **Dana H. Ballard** and **Christopher M. Brown**. The authors assemble crucial material from many disciplines including artificial intelligence, psychology, computer graphics, and image processing to form a practical text and reference for anyone involved in building vision systems.

Ballard and Brown write in their preface, "**Computer Vision** has a strong artificial intelligence flavor, and we hope this will provoke thought. The text shows how both intrinsic image information and internal models of the world are important in successful vision systems."

Divided into four parts, **Computer Vision** offers descriptions of objects at four levels of abstraction:

- Generalized images—images and image-like entities;
- Segmented images—images organized into sub-images that are likely to correspond to "interesting objects";
- Geometrical structures—quantitative models of image and world structures;
- Relational structures—complex symbolic descriptions of image and world structures.

PRENTICE-HALL, INC., Englewood Cliffs, N.J. 07632

ISBN 0-13-165316-4

IPR2022-00092 - LGE
Ex. 1015 - Page 539