8

Spatial Reasoning Using Modular Inference Engines

PRASANNA G. MULGAONKAR
SRI International, Menlo Park, CA

LINDA G. SHAPIRO
ROBERT M. HARALICK
University of Washington

Knowledge about three-dimensional (3-D) spatial relationships can be encoded as modular inference engines, each of which is an "expert" in analyzing some facet of a single spatial relationship. A collection of such experts codifying the rules of perspective geometry, for example, can then be used to generate hypothetical descriptions of 3-D scenes observed in a perspective image. We describe a spatial reasoning system based on this paradigm that can analyze digitized line drawings of planar and conic 3-D objects and can generate consistent 3-D descriptions from single perspective views. The inference engines operate under distributed control in an organization we call the "network" model. This model is superior to traditional blackboard-based systems and is easier to maintain than an agenda-based system. It can be implemented using associative memory.

1. INTRODUCTION

A key part of the complex visual processing performed by humans is the mapping from the 2-D visual patterns sensed by the retina to 3-D structures in space. This spatial-reasoning ability involves analysis of a large number of diverse visual cues and brings to bear a large unquantifiable body of knowledge culled from prior experience. In order to develop computer systems that will perform vision tasks of complexity similar to those encountered by the human visual system, it is necessary to investigate all potential sources of information used in this spatial reasoning, as well as techniques for efficiently organizing the information.

We contend that one aspect of this spatial reasoning can be viewed as hypothesis-based reasoning. The physics of the imaging process that produces the retinal image provides strong constraints on the possible 3-D interpretations that can be ascribed to the elements of the scene. Knowing the nature of the 3-D elements that are likely to be part of the

scene, and given the "measurements" that can be made on the image, one can generate and test hypotheses describing the scene contents. Any hypothesis that does not violate the constraints imposed by the physics of image formation is a consistent or plausible description of the world.

In reality, the world consists of many types of 3-D elements, which have common properties and which interact with the imaging process in different ways. For example, polyhedra produce polygons of almost constant intensity in the image plane (ignoring surface markings and reflectance changes), and specular objects produce strong highlights that change with the viewpoint. The computer vision literature is replete with techniques that exploit a large range of such phenomena. The "shape-from" techniques estimate 3-D surface properties based on various features such as observed intensity changes (Horn, 1975, 1977) texture gradients (Witkin, 1981), and symmetries (Kanade, 1981). Model-based matching assumes detailed geometric knowledge about a small set of 3-D objects (Horaud & Bolles, 1986; Ikeuchi, 1987). Knowledge-based systems predict what they expect to see in complex scenes and use feedback from the image features to constrain their expectations (Brooks, 1981; McKeown, 1984).

All these techniques have one common underlying theme: Although the projection from 3-D to 2-D is not directly invertible, only a few possible spatial configurations can give rise to the observed spatial patterns in the image. For example, under the assumption that surface markings are evenly distributed, only one particular surface slope would give rise to an observed texture gradient. The interplay of the relationships between 3-D entities and the observed spatial relationships between their images is the key to inverting the projection. At first glance, such a technique may appear to produce only local descriptions of single entities. What allows the technique to work, however, is that these local inferences can then be propagated to neighboring regions, using hypothesized spatial relationships between the entities in the world.

In the remainder of this chapter, we describe a system set up to explore this technique of producing and testing hypotheses about the spatial configurations in the scene. We describe the origin of the knowledge encoded in the system, specifically that of perspective geometry pertaining to the projection of lines, planes, and circular arcs. Next, we describe how the knowledge is encoded and organized, comparing the network model used in our system to the two classical techniques in the AI literature. Finally, we discuss the experiments run with our prototype system.

2. CONSTRAINTS OF PERSPECTIVE GEOMETRY

The human visual system uses a large amount of diverse knowledge sources in concert to analyze its visual input. Information, such as stereo disparity, focal adaptation, and occlusion due to motion, contribute important cues which provide distance information in absolute or relative terms. In addition, we have the capacity to infer three-dimensional (3-D) structure from monocular gray-tone pictures. The process of forming such images from scene structures is one of central projection. Central projection is not an information preserving mapping. Points in the real world are three-dimensional. Points in the perspective or central projection are two-dimensional. Lines viewed on their ends will appear as points in the perspective projection. Planes viewed on their sides will appear as lines in the perspective projection. Therefore, it is not possible to compute a unique 3-D interpretation for any given 2-D image.

During the process of image interpretation, some additional knowledge has to be inserted into the computation in order to compensate for the lost information. Researchers in the past have used various techniques to supply this missing information. The pioneering work by Roberts (1965) on model-based interpretation of images made up for the missing information by supplying exact 3-D models of expected objects. Later, researchers concentrated on various different ways of defining the models. Brooks (1981) worked with a symbolic theorem prover to constrain the free parameters of the image formation process using known models and projective invariants to guide the search. Mulgaonkar, Shapiro, and Haralick (1984) utilized spatial relationships between 3-D primitives of rough object models to control the computation of the free parameters. Recent work on the back projection problem by Barnard (1982) has examined some conditions under which meaningful inferences can be made about 3-D structures without the use of an object model. However, the inferences that can be made by back projection of individual elements in an image are very few, precisely because of the multiplicity of interpretation that each primitive element may have.

In this chapter we show that we can utilize equations of perspective geometry in a cooperative sense to rule out a large number of the multiple interpretations and arrive at a plausible structure or structures that could give rise to the image. Haralick (1980) has an excellent compilation of relevant equations for the perspective projection of points, lines, and planes. Mulgaonkar (1984) contains additional equations dealing

with the perspective projection of conic sections in a common framework.

The domain from which we draw the examples in this chapter consist of solid objects made up of planar and cylindrical faces. We are given an image consisting of edges and arcs corresponding to edges between surfaces in the scene. We do not assume any a priori knowledge about the objects other than the class of surfaces defined above and possible spatial relationships that can be used to define their arrangement in space. For example, we know that straight lines can be parallel to each other, they can lie in a plane, they can be perpendicular to planes, and so on. We show how we can use this knowledge along with the mathematics which transforms the generic class of 3-D primitives into their corresponding images to infer the 3-D structure given a single 2-D image. We assume a nonsingular or general viewpoint which means that straight lines in the image are projections of straight-line segments in the world and no lines in three-space project onto points in the image.

As an example, consider the type of reasoning that would be involved in interpreting the line drawing shown in Figure 8.1. In this example, as in subsequent ones, we follow the convention that image primitives are labeled using lower-case letters, and their corresponding 3-D counterparts are labeled with the equivalent upper-case letters. The image consists of nine visible straight line segments which, when taken four at a time, bound three surfaces. We do not know how these surfaces are arranged in space. However, we do know that any possible arrangement must be such that from some camera position it produces the observed arrangement of lines and regions in the image. Not all possible arrangements of these lines would satisfy this condition. For example, if lines A and B (corresponding to line segments a and b) were parallel and lines C and D (corresponding to line segments c and d) were parallel, then the lengths of the line segments C and D must be equal. In addition, the plane ABCD corresponding to the region abcd would have to be horizontal because the intersection of the images of the parallel pairs lie on the horizontal line through the center of the image.

The key feature of the reasoning is that we measured relationships between the lower-case entities to determine the possible relationships between the corresponding upper-case entities. We define an *interpretation* of an image to be a set of inferred relationships between the 3-D entities which, when transformed by the rules of perspective geometry, agree with the measurements made in the image. Of all possible interpretations, we look for those that are in some sense *maximal*

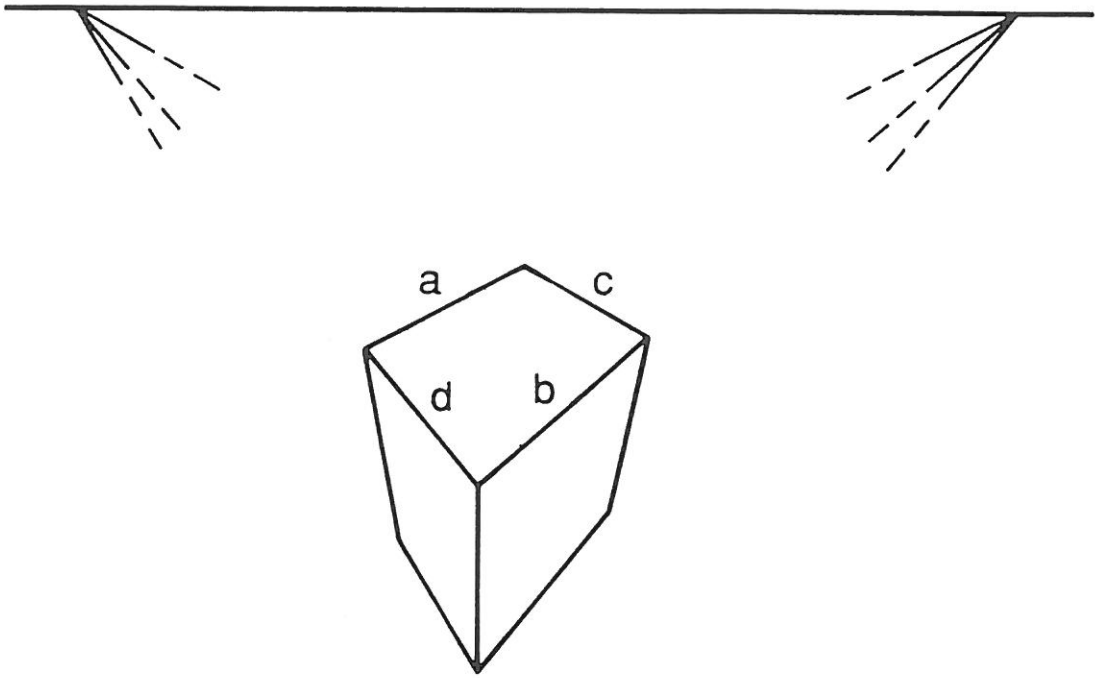


Figure 8.1. A Typical Line Drawing Requiring Interpretation.

or *best*. In this chapter, we define an *optimal interpretation* to be that interpretation (or interpretations) that constrains the largest number of 3-D entities.

The search may be performed by hypothesizing possible relationships between groups of 3-D entities and verifying their correspondence with measurements by applying the perspective equations. In reality, this problem is a lot more complicated. Spatial relationships between 3-D entities have numerical valued attributes whose values are drawn from the infinite domain of real numbers. For example, parallel lines are attributed by the normal distance between them; surfaces have the direction cosines of their normals as attributes. The values of these attributes control the appearance of the entities in the projection. Therefore, a hypothesis cannot be verified until all relevant attributes of the spatial relationships have numerical values. The domain of the attribute values is infinite and therefore we must compute them rather than search for them.

The equations of perspective geometry can be inverted and values for the attributes computed based on the spatial relationships present in the hypothesis. For example, if a pair of lines in the image is hypothesized parallel in three-space, then the direction cosines of the lines can be computed based on the measured location of their vanishing point in the image. Perspective geometry provides a large repertoire of such inferences and, therefore, a large number of attributes relate to

more than one image level measurement. Thus, there are often multiple computation paths by which we can compute the values of most attributes. The definition of consistency in this framework then becomes:

> A hypothesis consisting of proposed spatial relationships between 3-D entities is *inconsistent* if the measurements from the image imply that some attribute of some relational tuple in the hypothesis, simultaneously, must have more than one distinct value. A hypothesis that is not inconsistent is *consistent*.

The vision system described in this chapter uses this definition of consistency to find the largest consistent hypothesis relating the 3-D entities corresponding to the entities visible in a given perspective image.

In the next section, we define the notion of an *inference engine* used for applying a single rule of perspective geometry to a hypothesis. We develop the concept of a string of inference steps consisting of sequential applications of inference engines to compute numerical values for attributes. We then prove the stability of such a distributed computation scheme for determining the consistency of a given hypothesis.

3. INFERENCE ENGINES

Inference engines are modular computation units which accept as input a specific set of attributed relational tuples made up of possible relationships between world entities and a set of measurements taken from the image. Based on the measurements and on the previously computed values for the attributes of the relational tuples, they compute values for other attributes of tuples in the input set.

The mode of operation of these inference engines is as follows: The initial input consists of a hypothesis whose validity is to be determined. All the attributes of all the relational tuples in the hypothesis are initially valueless. Inference engines are triggered based on their input requirements and compute values for some attributes. For example, if the hypothesis contains a relational tuple of the form (*parallel lineA lineB*), the vanishing point inference engine would be triggered, since all parallel lines have the same vanishing point, and it would compute a value for the vanishing point attribute of *lineA* and *lineB*. The processing involved in this case is to compute the intersection of the corresponding image lines *linea* and *lineb*. Subsequent inference engines whose computations use vanishing points may then be triggered. For example, one inference engine may compute the focal length

of the camera based on vanishing points for two nonparallel, planar pairs of lines. This inference engine would look for the following relational tuples in the hypothesis:

```
(3D-line lineA *a *b *c *d)
(3D-line lineB *a *b *c *d)
(parallel lineA lineB *upx *upz)
(parallel lineC lineD *upx *upz)
(in-a-plane lineA lineC *normala *normalb *normalc *offsetd)
(not-parallel lineA lineC *angle)
```

The terms with asterisks are place holders for the attributes of the relational tuples. For example, the attributes of the 3-D line-relational tuples define the equation of the line in the form

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

the attributes of the *parallel* relational tuples encode the vanishing point in screen coordinates for the pair of lines participating in the relation; and the attributes of a plane define the plane equation. The inference engine would examine the vanishing point attributes of the two *parallel* tuples to see if they had values assigned to them. To summarize, inference engines are independent computational modules which compute values for attributes of an input set of relational tuples, based on measurements from the image and possibly some previously computed attributes of some tuples in the input. A hypothesis is inconsistent if applications of these engines lead to distinctly different values for any attribute.

The questions that arise in this context deal with the stability of termination of the computations. Suppose a set of inference engine applications determine a hypothesis to be valid. Is it possible to apply a different set of engines or to change the order of application and arrive at the conclusion that the same hypothesis is inconsistent? Is it possible that changing the order of engine applications changes the final set of values for the attributes in a hypothesis and, if so, does the application process involve a search for the *correct* sequence of applications?

These questions arise in every *blackboard* type distributed computation system where a group of modules can independently update and change information in a common data store. Independent modules can be shown to be correct. However, since the inference engines interact with each other and use the results of each other's calculations, the question of order dependence and uniqueness of result must be proved.

In predicate calculus, consistency of a set of predicates may be

viewed as a conjunction of conditions that the set of predicates must jointly satisfy. Since conjunction is a commutative operation, the order in which the terms of the predicate are tested is irrelevant. Intuitively, it may seem that similar results should hold for applications of inference engines because they, too, check consistency with respect to individual rules of perspective geometry. However, inference engines cannot be applied in any arbitrary order. An engine can *only* be applied if the information it requires to execute is already present. That is, if an inference engine requires some attributes of some tuples in the input hypothesis set to have previously computed values, it cannot operate until some other module computes the required values. Thus, there is a partial ordering which describes all the legal sequences in which the engines may be applied. Other complications result from the fact that inference engines compute values for attributes in addition to providing a *consistent/inconsistent* response. These values cause the state of the hypothesis to incrementally change. Therefore, some thought is required to show that the intuitive result does indeed hold and that inference engine application is a stable process whose result does not depend on the particular sequence chosen.

By carefully formalizing the concept of an inference engine, we can prove a series of interesting theorems which show the desired properties of the inferencing process. These theorems were originally proved in Mulgaonkar (1984) and Mulgaonkar, Shapiro, and Haralick (1986).

We start with some definitions of the terms involved.

We say that an inference engine E is *applicable* to a subset K of the input hypothesis H if the following conditions are met:

1. K satisfies the input requirements of E. That is, K contains the relational tuples that E uses as the basis of its computation and the required attributes of tuples in K have previously been assigned values.
2. No proper subset of K satisfies the input requirements of E.

An *application* of E to K is denoted $E(K)$ where E is applicable to K. An application may succeed or fail. An application fails if the value computed by E for some attribute of K is inconsistent with a previously computed value for the same attribute. If the application succeeds, the set K is changed to a new set K' which differs from the original in that, at most, one attribute of K, which was previously undefined, now has a numeric value associated with it.

An *inference step* consists of one application of E to a subset K of the hypothesis and the replacement of the subset K by the new subset K' in the set of input hypotheses.

A *sequence of inferences* on a hypothesis H is a sequence $E_j(E_{j-1} \ldots (H) \ldots)$ of inference steps where each engine is applied to the output of the previous step. We stipulate that no inference engine may apply more than once to the same subset K of H.

A sequence of inferences $E_j(E_{j-1} \ldots (H) \ldots)$ is called *terminating* if either $E_j(.)$ is a failed inference step, or there does not exist any other inference engine E_{j+1} which is applicable to the result of the sequence.

Theorem 1: Consider a sequence of applications of the inference engines to a hypothesis H. If at the i_{th} step, engine E_i is applicable, then E_i will remain applicable at all inference steps $j > i$. That is, we can defer the application of an applicable inference engine.

Theorem 2: If, at Step i, application of inference engine E_i would fail, then E_i would fail even if its application is deferred.

Theorem 3: Changing the order of application of inference engines does not cause a successful sequence to terminate in failure.

Theorem 4: Any sequence is either a terminating sequence or can be extended by further applications of inference engines to form a terminating sequence.

Theorem 5: If any one sequence of applications of a set of inference engines to a hypothesis terminates in failure, then all possible sequences of applications terminate in failure.

Theorem 6: If any one sequence of applications terminates successfully, then all possible sequences terminate successfully.

Theorem 7: Ignoring permutations, there is at most one successful sequence of applications of a set of inference engines to a hypothesis.

We use the concept of terminating sequences to define consistency of hypotheses as follows: A given hypothesis is *inconsistent* with respect to the knowledge encoded in a given set of inference engines and the measurements from a given image if it has an associated sequence of applications which terminate in failure. If, on the other hand, the associated sequence of applications (which may be of zero length) ends successfully, then the hypothesis is *consistent*.

As examples, we list a few of the inference engines that are implemented in the experimental system. The system currently has 100 inference engines. Any vision system that goes all the way from low-level image operations to a final interpretation of the scene would require a much greater variety of knowledge sources. The system reported here has the specific task of verifying our hypothesis that perspective geometry provides a strong enough set of constraints to produce reasonable hypotheses about scene structure. In addition, these constraints can be

implemented to use closed form inverse projection equations instead of a theorem prover search over the infinite real number set.

Inference Engine 1: Given the hypothesis that two lines are parallel and the fact that their images are not parallel, determine the vanishing point of the lines as the intersection of their images.

Inference Engine 2: Given the hypothesis that two lines are coplanar, and the fact that they do not have a common vanishing point, compute the vanishing trace of their common plane.

Inference Engine 3: Given a plane with a known vanishing trace, compute the vanishing points of all lines hypothesized as lying in that plane.

Note that Engines 1 and 2 both compute values for the same attribute—the vanishing point of a line. This is the basis of consistency checking. The method relies on the fact that perspective geometry has a large number of such mutually constraining equations.

In summary, we have shown that the problem of checking the consistency of hypotheses against the constraints of perspective geometry can be solved efficiently. Thus, such a process can be usefully incorporated as one component of an image-interpretation system on an equal footing with other shape-from processes. It could be used to verify the consistency of hypotheses generated using, say, shape-from-shading or shape-from-symmetry. In addition, by incorporating processes for hypotheses generation, it can be used to generate all consistent interpretations for the structures in the scene.

4. DISTRIBUTED PROCESSING

Each inference engine is a complete module. As its input, it takes the entire hypothesis that has been constructed up to that point and examines it for specific conditions under which it can operate. For example, consider an inference engine which computes the vanishing point of parallel lines. Given the data from the image and a hypothesis about the lines in the 3-D world, this inference engine computes the vanishing point of the lines which are hypothesized as being parallel. What should the hypothesis contain that enables this engine to "fire"? It must contain a prediction that there exist two lines (say, line a and line b) which are parallel in the world. Only when such a prediction is present in the hypothesis will the inference engine be able to perform its function.

Once the inference engine detects that such a prediction is present in the hypothesis, it obtains the relevant parameters from the image mea-

surements (say, the equations of the projections of these lines), and computes the vanishing point. Now there are three choices open. First, it may be the case that the vanishing points of the two lines were previously unknown. In this case, the action taken by the inference engine is simply to assert that the vanishing point of the two lines is the one that it just computed. Second, based on inference engines that were activated before the current one, it may happen that the vanishing point of one or both of these lines may have previously been assigned some values. If the newly computed vanishing point matches the previously asserted values, everything is fine. Otherwise, the hypothesis is inconsistent because the two different reasoning paths, which lead to values for the same physical attribute, do not yield the same result. In this case, the inference engine must indicate that an inconsistency has been detected.

To sum up, inference engines:

1. Check the hypothesis for relevant predictions.
2. Use the predictions combined with measurements from the image to compute the values for attributes.
3. Check these values for consistency with previous predictions for the same attributes.
4. If the hypothesis is inconsistent, indicate that backtracking must occur in the search for the interpretation.

5. REQUIREMENTS FOR CONTROL

In this section, we are concerned with the question of how such a group of modules can be controlled. Since inference engines accept as input hypotheses modified by other engines, and in turn modify the attributes of the hypotheses themselves, we need to have a technique for sharing all available information in a global sense.

Secondly, note that if an inference engine works on some subset of the hypothesis, and if its conclusion is that the hypothesis is consistent, the hypothesis may remain unchanged. That is, all the predictions that formed a part of the hypothesis before the inference engine was invoked are all still present at the termination of the inference process. Therefore, some care has to be taken to ensure that the same inference engine does not claim that it is still applicable and repeat its calculations over and over again. Effective control of the inference process demands that an effective technique be developed for marking the fact that a certain subset of predictions has been examined by some particular inference engine at some state in the process of interpreta-

tion. We will first examine the question of data sharing and sequencing the inference engines, and then look at the use of associative memory for saving the history of computations done by each engine.

6. BLACKBOARD-BASED SYSTEMS

It should be clear from the previous section that the inference engines can run in an entirely distributed mode of operation as long as there is some common area where the hypothesis can be stored for access and updated by each module. A common data area for such purposes is called a *blackboard* in the AI literature. The first use of this term in a large expert system was in the HEARSAY II expert system for the speech analysis (Erman, Hayes-Roth, Lesser, & Reddy, 1980). A blackboard can be defined as a structured global database which can be queried and updated by independent processes which use the data. Access mechanisms have to be provided for the access and update procedures to maintain consistency of the database. Computational processes in HEARSAY II were organized as independent units (demons) which would be triggered by specific processing conditions noted in the blackboard. For example, if the sentence currently being parsed had a noun phrase isolated, the adjective expert would recognize that fact by examining the blackboard and get triggered.

Let us examine how such a completely distributed system would actually work in a programming environment which is sequential. Each self-contained expert would need to have the capability of analyzing the contents of the blackboard to determine if it could use the available data and contribute new information to the system. On determining that it was in fact capable of performing its specified task, it would then need to compute its results and place them in the appropriate place in the global database.

Such a control strategy is conceptually simple. However, in sequential computing machines, it needs modification. The usual modification is the addition of a supervisory task whose function it is to poll each module to determine if it could run and, if so, to allow it to complete its task. The simplest supervisory strategy is to sequentially poll each module in turn until one full sweep of all the modules yields no new information on the blackboard.

In addition, each module needs to remember the particular set of inputs that it has examined so that when it is polled again, it will not repeat calculations that it has already performed. Otherwise the system will loop forever, constantly repeating the same task over and over again.

To summarize the distributed processing control strategy with processes communicating over a common shared database:

1. A supervisory process sequentially invokes each computational module.
2. Each module examines the available data and determines if it can work.
3. If it can perform its duties, it remembers the exact condition under which it was triggered and updates the database with its conclusions.
4. The process terminates when no module can add any new information to the blackboard.

Although this control strategy works well in many situations, it is easy to see that it has shortcomings. It takes a certain amount of effort for each module to determine if its conditions have been met by previous computations and if it can compute new results using input it had not used before. Sequentially invoking each module is computationally very expensive. Most of the time the majority of the modules will simply not be applicable. As the number of modules increase, the time spent in the supervisory process rises and becomes comparable to the time spent by the modules actually computing new results.

The solution to this problem is to recognize the fact that the modules are not actually independent. Most modules depend on others for their input. Consider the example given before; the adjective detection module depends on the fact that a noun phrase has been detected by a previous module. Consequently, it is possible to effectively disregard the existence of the adjective module until an appropriate noun phrase has been detected. This can be done by providing the supervisory process with the appropriate "meta" knowledge about the task and, instead of a simple sequential scan of the modules, it performs the task of invoking the computations in a more intelligent fashion.

However, if the supervisory process needs to perform checks of complexity similar to that of the modules themselves, nothing is gained. On the other hand, recognizing the fact that the modules are related, the task of checking the preconditions of a module can be delegated to the other modules which are related. In the simplest case, if Module A uses the output of B as its input, Module B can check its output at the time it is generated and inform Module A when its output is of the correct form.

Systems using this strategy are termed *agenda-based* systems in the AI literature.

7. AGENDA-BASED CONTROL STRATEGY

The most complex agenda-based system in the AI literature is the AM expert system (Lenat & Davis, 1982) whose field of expertise is in the area of arithmetic proofs. In essence, agenda-based systems also consist of independent processes communicating over a common shared information source. However, each module knows about the other modules in the system, and knows the conditions which are required by the other modules to perform their tasks. Once a module computes a particular piece of information, it knows which of the other modules could use what it has just computed. It then checks to see if the preconditions of the other modules are satisfied by the state of the blackboard updated with the information it just computed. If it detects that some module has all the information it needs, that module is triggered.

On sequential systems, the process of triggering a module is implemented by placing the module to be executed in a first-in, first-out queue. The task of the supervisory process is then reduced to removing the first task on the queue and executing the appropriate module. The process terminates when the queue is emptied. This simple strategy may be further enhanced by treating the queue as a set and allowing the supervisory process the freedom to choose the next module to be tried. This allows some meta-level knowledge about the problem domain to be placed in the control process.

Under such a control paradigm, it is not necessary for each module to check the blackboard for its preconditions. The fact that someone placed it on the agenda queue is sufficient information that its preconditions are satisfied. From the standpoint of execution speed, this situation is excellent. Unless some module can potentially run, its preconditions are never checked and, for all practical purposes, does not exist. Thus, unlike the distributed processing case described earlier, there is no overhead involved with checking if all the modules can execute.

In summary, the agenda-based systems are characterized by the following:

1. Each module knows which other modules depend on it for input.
2. Each module checks all the preconditions for those modules which depend on it.
3. If some other module's preconditions are satisfied, the module is placed on a processing queue for future execution.

However, the drawback with this scheme is that the checking of preconditions is far too distributed. Consider what happens if a new module must be added to the system. If the new module takes n pieces of data

as its input, and if each of these pieces is updated by m possible modules, in the worst case mn modules must be changed to recognize the existence of the new module. In each of those mn cases, code corresponding to the full check on the preconditions of the new module must be added and checked. This becomes worse as more and more modules are added. So although we gain execution speed, it is at the expense of readability and maintainability of the system as a whole. Contrast this situation with the demon-based control strategy. Demons are totally uncoupled and, consequently, whenever a new module must be added to the system, the only change that must be made is to inform the supervisory process that the number of modules to be polled has increased.

In the next section, we present a hybrid technique which retains the best of both worlds. It has the advantage of the ease of maintenance of the distributed control strategy, along with the speed of the agenda system. Although it is not as fast as the agenda system or as uncoupled as the distributed system, we find it to be a suitable blend of the two.

8. THE NETWORK MODEL FOR MODULE CONTROL

All inference engines take a particular set of tuples and their attributes from the hypothesis as their input and produce new values for some of the attributes as their output. These new values update the hypothesis and are, in turn, accepted as inputs by other inference engines in the system. This fact can be compactly denoted by representing the engines as nodes in a network. The nodes in a network may have some arcs leading into the node and some arcs leading out of the node. Every node leading in corresponds to an input attribute of some tuple which must have a previously computed value, while outgoing arcs represent values for attributes that are computed by the inference engine located at the node.

In addition to these data paths, the inference engines have access to the measurements that are computed from the image. All the inference engines have equal access to the entire set of image information. Since the image information is given a priori, there is no need to explicitly denote it as inputs to the inference engines.

In addition, the tuples in the hypothesis must be of the right form. For example, consider the input requirements of an inference engine that computes the focal length of the camera. We know that the vanishing points of two perpendicular lines contain enough information to compute the focal length. Thus, a module which performs this calculation needs to have two lines in the image which are mutually perpen-

dicular. Besides, it needs the computed vanishing points for each of the lines. The vanishing points must be nonsingular. Only under these conditions can the inference engine perform its computation. If this engine were to be controlled using an agenda-based paradigm, all these checks would need to be performed by each inference engine which computes vanishing points of a line. Thus, the inference engines, which compute vanishing points for a line, need to know not only all the information about the line, but also all information about other lines in the image and their current status to be able to determine if the focal length expert should be invoked.

From the point of view of modularity, it is desirable that the amount of information that any given module needs to know should be restricted to the minimum necessary to achieve the desired efficiency. If the focal length computing module were to be updated to require some additional information as input, the amount of updating necessary to keep the system consistent would be minimized.

In the network approach, the only information that is provided to the modules that compute values for attributes is the names of the inference engines which could potentially use the computed value. Thus, all that the vanishing point calculation engines need to know is that there exists a module for the computation of focal length which uses vanishing point that it calculates.

Once an inference engine has computed its values and updated the global database to reflect the changes, it then triggers the appropriate inference engine which could use the results of its computation. It is the job of the triggered module to check the conditions and determine if it indeed has all the required information. If a module gets triggered and figures out that it cannot do its task, it turns itself off. It then stays off until some other module determines that it may be able to use its output.

The key to the success of such a scheme hinges on the fact that once values are computed for any attribute, these values remain in the global database and are not lost. Thus, if at a later time some module can use them, they are still available.

In the case of the focal length module, consider what happens when some module computes the vanishing point of some line. Assume for the moment that this is the only line for which a vanishing point has been calculated. The vanishing point module turns on the focal length module. However, all the input conditions for the focal length module have not yet been satisfied. Consequently, the focal length module turns itself off. It will stay off until a vanishing point gets calculated for some other line in the image. Once again it will get invoked, and will examine the available information to see if it can perform its task.

Initially all the modules are deactivated. The process that generates

the hypotheses turns on appropriate modules depending on the tuples being hypothesized. Not all these modules may actually be applicable. Each module then gets a chance to see the input and check its own conditions. Those that are not applicable, turn themselves off. Others compute new values and, in turn, mark other modules for activation.

Thus, this scheme is decoupled in the sense that the conditions for activating each module is concentrated in the module itself. It has the flavor of an agenda-based system. Modules know of the existence of other modules to the point that they know which modules could, under some circumstances, use the output that they generate. The advantage of this system over the pure demon-based system is that most of the time the majority of modules are deactivated and, therefore, do not have to be considered by the supervisory process. Thus, there is no overhead involved in trying to invoke modules which are obviously inapplicable. The advantage over the pure agenda-based system is that the amount of information that other modules need to have about any given module is absolutely minimal.

The overall control algorithm consists of a supervisor, which sequentially polls every module that is not deactivated. If the module determines that it is not applicable, it deactivates itself. If, on the other hand, the module is applicable, it performs its actions and, if it computes any new values for attributes, it in turn activates other inference engines. Note that if an active engine determines that it can perform its task, it does not turn itself off, because there may be more than one subset of the hypothesis to which it may be applicable. It turns itself off only after it has performed its task on all the applicable subsets of the hypothesis.

We have shown in Mulgaonkar, Haralick, & Shapiro (1984) that the order in which the engines are actually invoked does not affect the result. This means that the simple sequential scheduling policy described above is appropriate and no erroneous conclusions will be reached.

9. CONTROL OF INFERENCE ENGINES USING ASSOCIATIVE TABLES

As was shown in the previous sections, one aspect of effective control is to be able to selectively enable and disable the inference modules. In addition to this, the inference engines need the capability to determine, with minimum computational overhead, whether they are in the "on" state or the "off" state, and to find the subset of the data on the blackboard which they need to analyze if they are "on." If they find a subset of the data to which they are applicable, they must then determine if

that subset has been examined in the past. In this section we show that the use of associative tables greatly simplifies this task.

Consider the actions that an inference module may be called upon to perform. Since these actions all affect the global database, we may consider the generic actions that can be used on any data structure. These actions are:

1. Addition of new information
2. Deletion of old information
3. Updating and changing existing information.

In addition, an inference engine may determine that any information it could add or update already existed in the database and, consequently, would take no action. In general, these actions would be data-dependent. That is, some combination of input conditions would be satisfied by the global data; based on this, the inference actions would be performed.

If the action of deleting information or updating existing information changed the preconditions for the inference engine, there would be no problem. Otherwise, the input conditions would still hold after the application of the inference task. In this case, if the module were to be polled again, it would still find the same conditions as before, and recompute the previously computed values.

There are several ways in which this problem could be tackled. One technique could be to construct all engines such that values never get recomputed. This could be done by making sure that one of the input conditions determines whether the output variables of the engine have already been computed. Thus, the same engine, if called on to test the blackboard, would not work with the same data. However, this is an undesirable situation in the context in which this model of inference engines was developed.

By our definition of consistency, we declare a hypothesis inconsistent precisely when an inference engine recomputes the value for a variable which was given a value by another engine, and finds that the two values do not agree. It hinges on the fact that values get computed more than once and, obviously, testing the output variable of the inference engine is not the solution we seek.

Another possible solution could be to define some kind of a search pattern in which an inference engine searches for the data on the blackboard for applicable input sets. Thus, it would never return to a particular input without having scanned the entire possible sets of inputs on which it could perform its task.

This, too, is an undesirable situation, because for it to work successfully, we must have some way to ensure that once an engine has passed

a particular point in its scanning pattern, no changes are made to the database which cause a new possible input set to be generated, which lies in the part of the search pattern already completed by the inference engine. Allowing the engines to repeat their search patterns just brings us back to the previous problem.

The solution that we seek is one in which each engine could remember the exact input conditions of the sets of data that it had used in its previous computations, and the next time it encountered the same input set, it would ignore it. Note that in this technique the inference engine may encounter the same input set each time it is invoked, so we must rely on a rapid way in which the determination of whether it had previously seen the input could be made.

Our solution to this problem involves the use of associative tables. Associative tables, which associate with a key a prespecified value, allow the convenience of storing arbitrary keys and rapidly accessing the stored information indexed by the keys. Our implementation of such tables makes use of hash functions which can take any data structures, such as lists, character strings, and indeed even other tables as the key, and allow accesses with near linear expected time performance. Once an engine has examined a particular input set, it stores information in the table that "ENGINE x" has worked on "input data i, input data j . . . ." Whenever the same data is encountered again, it can determine the fact instantly.

The same tables are also used to store information about the current status of the engine, namely, whether it has been signaled as a possible candidate to be tried by some other inference engine, or if it is in a quiescent state. This strategy allows the network model to work successfully, because all the time-consuming bookkeeping tasks get reduced to rapid accesses into the associative memory. The supervisory task does not even have to check if an engine is in the "on" or "off" states, because now that task can be relegated to the inference engine itself. The first task of the engine is to determine its state. If it is "off," it exits and allows the next engine in line to start off. If it is on, it searches the blackboard for candidate input sets, and examines each one till it finds one it has not seen. It then performs the relevant actions on the input.

10. SEARCHING FOR THE MAXIMAL CONSISTENT HYPOTHESIS

In the previous sections, we discussed a technique for encoding the knowledge of perspective geometry and applied that knowledge to determine the consistency of a hypothesis. We showed that the problem of

determining consistency of a hypothesis is a linear complexity task which can be performed efficiently. Thus, the practicality of using shape-from-perspective processes to infer scene structure is contingent on the efficiency of algorithms for generating hypotheses to be tested.

The efficiency of any processing depends on the amount of world knowledge that is brought to bear. Model-based vision systems, which have geometric or structural models of the three-dimensional world, have an advantage over nonmodel-based systems because the models constrain the space of interpretations. Vision-based, object-identification systems can test the more likely hypotheses before the unlikely ones. For example, to identify objects in an office scene, the hypothesis generator could check for "telephone" before "refrigerator." However, in such cases, it is hard to separate the contribution of the hypothesis generation process from that of the hypothesis-testing process towards the overall interpretive power of the system. For example, if some hypothesis was not produced in the list of "consistent interpretations" output by the system, it may not be clear if the knowledge embodied in the consistency checking was powerful enough to reject the hypothesis, or if the hypothesis was never generated.

To study the interpretive power of perspective geometry, we restrict our attention in this paper to a hypothesis generation system that does not use object models. Another motivation for studying such a system is that there may be cases where object-level knowledge is simply not available. Such bottom-up hypothesis generation techniques could also be used to augment and extend partial top-down hypotheses which may be model-driven. We show that although the search space of hypotheses is extremely large, it has a structure which can be utilized in order to quickly converge to the desired point.

The space over which the search for the *best* hypotheses is performed is the set of all possible subsets of relational tuples that can be constructed from a set of 3-D entities. Recall that although the consistency of hypotheses is determined by numeric values assigned to the attributes of the tuples, the search is not performed over the infinite set of real numbers. Instead, by using the inverse perspective equations encoded as closed form inference engines, we *compute* the numeric values based on the finite set of relational tuples. This reduces the overall complexity of the search to an NP-complete problem. However, a blind search through this exponential space is still prohibitively expensive.

The key structure of the space comes from two sources. The first deals with the nature of the consistency determination. If a hypothesis is consistent as defined by the terminating computations of a group of inference engines, all subsets of the hypothesis are also consistent. We

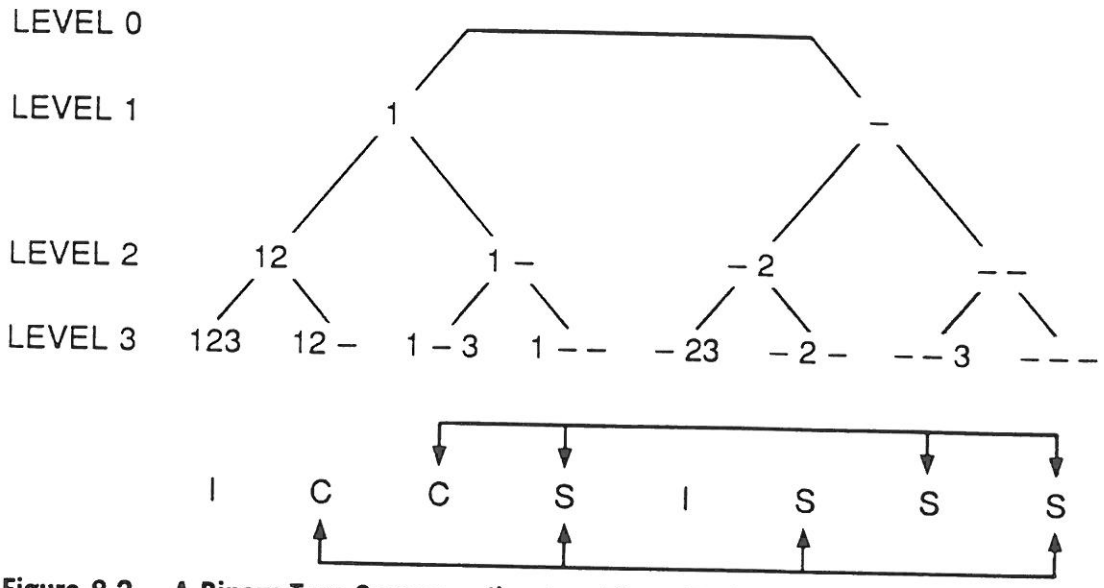


Figure 8.2. A Binary Tree Corresponding to a Three-Tuple Search Space. 123 and 23 are inconsistent; 12 and 13 are maximally consistent; and 1, 2, 3, and NUL are subjects of maximally consistent sets. The subsets of the maximally consistent hypotheses do not follow any regular pattern.

cannot force inconsistency into a set of relational tuples by *removing* any elements from it. In addition, if a hypothesis is inconsistent, it cannot be made consistent by adding new tuples. As long as the inconsistent core remains, the new superset of tuples would still result in a failed terminating sequence of inferences.

The structure can be utilized during the search process by ensuring that the search algorithm never considers parts of the space that cannot contain the desired solution. Consider the search space organized as a binary tree. Suppose there are n possible relational tuples. The tree would then have $n + 1$ levels labeled 0 through n. The 2^n leaf nodes correspond to the possible hypotheses. At each internal node at level i, the left subtree corresponds to the hypotheses in which tuple i is present and the right subtree corresponds to the hypotheses without tuple i. Figure 8.2 shows the binary tree corresponding to a three-tuple search space. It can be seen that the subsets of the consistent nodes and the supersets of the inconsistent nodes are not organized in any simply denotable or regular fashion. Thus, it is not easy to automatically disregard all subsets once a consistent solution is found.

However, it can be proved that if the *leftmost* leaf node in any subtree rooted at an internal node is a subset of some previously generated solution, then *all* nodes in that subtree are subsets of the same solution. Therefore the entire subtree can be pruned and ignored by the search algorithm. Similarly, if the *rightmost* leaf node is *not* a subset of any previous solution, then none of the possible solutions in the tree are

subsets. This tree pruning can be augmented by forward checking (Haralick & Elliott, 1979) applied at internal nodes in the tree.

The second source of knowledge which structures the search space is the knowledge of the semantics of the spatial relations from which the hypotheses are drawn. For example, if one of the spatial relations is the *parallel* relation between pairs of lines, then it is meaningless to construct tuples of the form (*parallel PlaneA PlaneB*)[1] when *PlaneA* and *PlaneB* are planes. Secondly, the relations themselves may be symmetric, reflexive, or transitive. In such cases, hypotheses which do not satisfy these conditions are automatically *inconsistent*. For example, if a hypothesis consists of the following two tuples: {(*parallel lineA lineB*) (*parallel lineB lineC*)}, but does not contain the tuple (*parallel lineA lineC*), then it can be declared inconsistent by virtue of being incomplete. This form of transitivity can be extended to include interrelation relationships. For example, if three lines *A*, *B*, and *C*, lie in a common plane with line *B* perpendicular to both lines *A* and *C*, then lines *A* and *C* must be parallel. In fact, such interrelationships between the various spatial relationships can be used to automatically extend partial hypotheses of the form {(*parallel lineA lineB*) (*parallel lineB lineC*)} to include the implied relational tuple (*parallel lineA lineC*).

Such semantic consistency rules can be used to efficiently reduce the search space by forcing logical completeness at all internal nodes of the tree. For example, if at internal node i in the tree, we take the left branch (include tuple i in the partial hypothesis), we also include all the tuples implied by the selections at levels 0 through $i - 1$ and the new tuple i. This allows us to make decisions about lower levels in the tree well before we reach them, reducing the number of levels that we actually have to search. For example, if we have already included a relationship (*parallel LineB LineC*), we can automatically include the logically implied tuple (*parallel LineA LineC*).

11. IMAGE PROCESSING EXAMPLES

The spatial-reasoning system described in the previous sections has been implemented using a set of approximately 100 inference engines. It can reason about the relationships between 3-D points, lines, circular arcs, and planes. Its input consists of a digitized, perspective line drawing produced with an unknown camera position and with a camera of unknown focal length. The input can be augmented, if necessary, by

[1] Note: For clarity, the place holders for the attributes have been omitted in this and subsequent examples.

supplying any of the unknowns, such as the absolute location of some point in space, or some of the camera parameters. The inference engines are coded in a modified Prolog language, and interpreted at execution time. The modifications to the standard Prolog language consist of extensions for handling new data structures, such as associative tables and images, and to allow deterministic flow of control in areas where a procedural approach (as opposed to a backtracking search) is necessary.

The final output of the system is a listing of the best hypothesis along with the computed values for the relevant numeric attributes, such as coordinates of points, direction cosines of lines, normals to the planes, and others. Absolute coordinates of points cannot be computed unless the true location of some reference point is known. This is because perspective projection involves an unknown scale change, that is, if all dimensions in the scene are scaled by a constant, the image does not change.

The system was tested on a variety of digitized line drawings as reported in Mulgaonkar (1984). The drawings were obtained from a book of perspective etchings by deVries (1968) containing a large number of architectural indoor and outdoor scenes. The drawings contain perspective projects of solids consisting of lines and circular arcs shown in perspective. In addition, several projections were constructed of synthetic 3-D objects of varying complexity (for example, see Figure 8.3) using a graphics system.

To aid in the interpretation of the results, the coordinates of points and equations of the lines output by the reasoning system were fed into a graphics package to generate a synthetic perspective view from a viewpoint different than that of the original image. This involved the manual identification of the 3-D coordinates of one point in the scene. Figure 8.4a shows an input image of moderate complexity, obtained from deVries (1968). The bold lines were selected as input to the reasoning system. Although all the lines could have been input, the number was restricted to keep execution times manageable for the Prolog interpreter in use. The results of processing lines in the input for the image are shown in Figure 8.4b. Similarly, Figure 8.5a shows another 3-D scene from deVries (1968), and its interpretation is shown in Figure 8.5b.

12. CONCLUSION

In this chapter we have demonstrated a technique for organizing and controlling inference engines so that the overhead consumed for per-

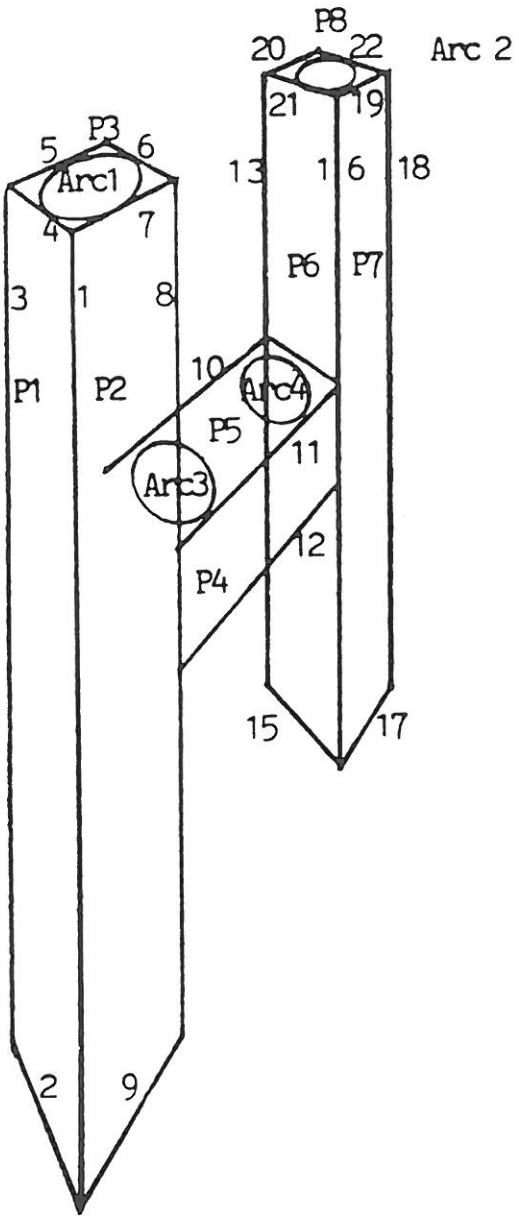
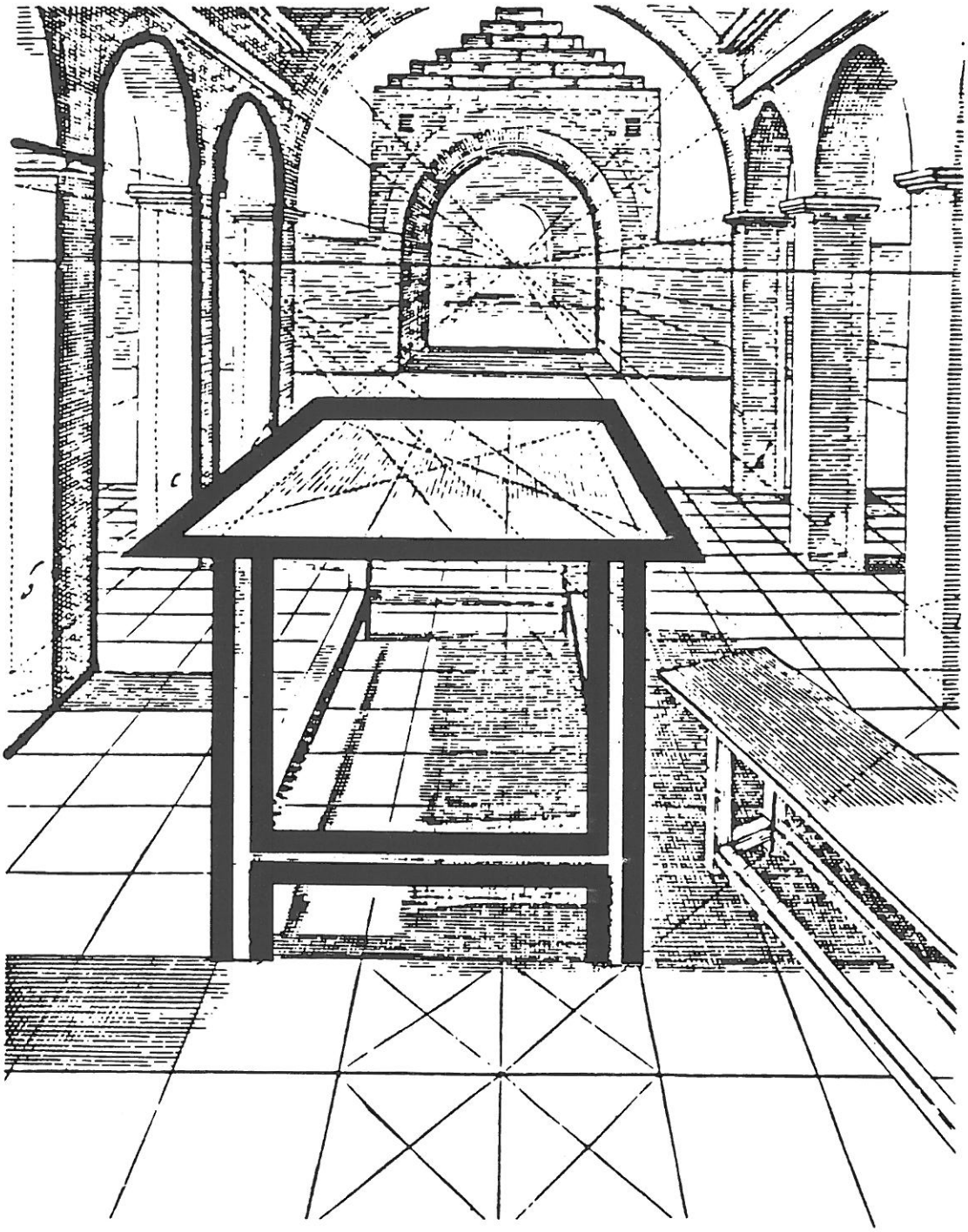


Figure 8.3. An Artificial 3-D Scene Generated Using a Graphics Package.

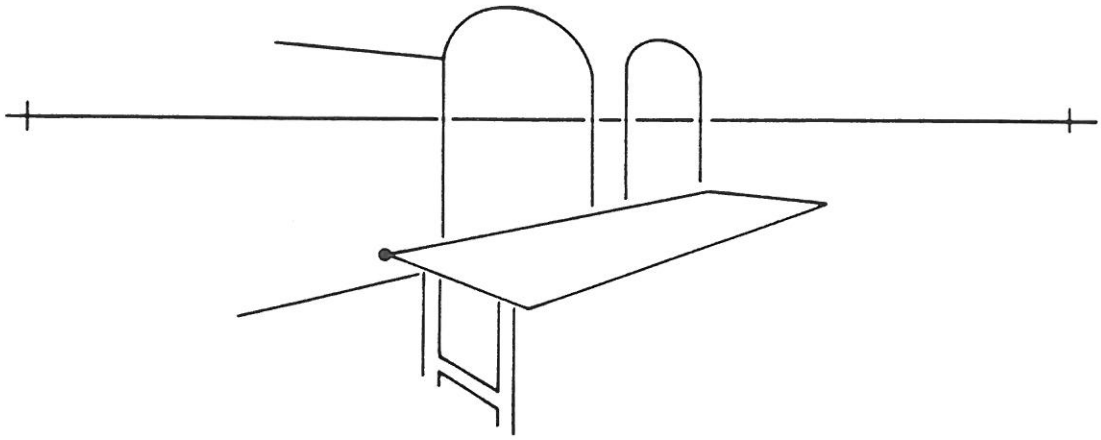
forming their work is minimized. We showed that the network organization permits the engines to run in a weakly coupled mode where the major part of the computational effort required for determining the applicability of engines is left to each engine itself. It obtains a little help from the other modules which determine if it "may" be applicable. This allows easy updating of the knowledge contained in the system, and yet retains the efficiency of a coupled system.

We also showed how the use of associative tables controls the structure of the inference engines and simplifies the task of determining an input set of data for the engines to use.



(a) Input Scene
Bold lines were supplied as input to the system

Figure 8.4. Interpretation of an Indoor Scene.



(b) Interpretation Generated by Spatial Reasoning
Coordinates of indicated point were supplied as a-priori information

REFERENCES

Barnard, S. T. (1982). *Interpreting perspective images* (Tech. Rep. No. 271). Menlo Park, CA: A.I. Center, SRI International.

Brooks, R. A. (1981). Symbolic reasoning among three dimensional models and two dimensional images. *Artificial Intelligence, 17,* 285–348.

deVries, J. V. (1968). *Perspective.* New York: Dover Publications.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980). The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys, 12*(2), 213–254.

Haralick, R. M. (1980). Using perspective transformations in scene analysis. *CGIP, 13,* 191–221.

Haralick, R. M., & Elliott, G. (1979). Increasing tree search efficiency for constraint satisfaction problems. *6th International Joint Conference on Artificial Intelligence,* Tokyo, Japan.

Horaud, P., & Bolles, R. C. (1986). *3DPO: A System for matching 3-D objects in range data* (pp. 359–370). Norwood, NJ: Ablex.

Horn, B. P. K. (1975). *Shape from shading.* New York: McGraw-Hill.

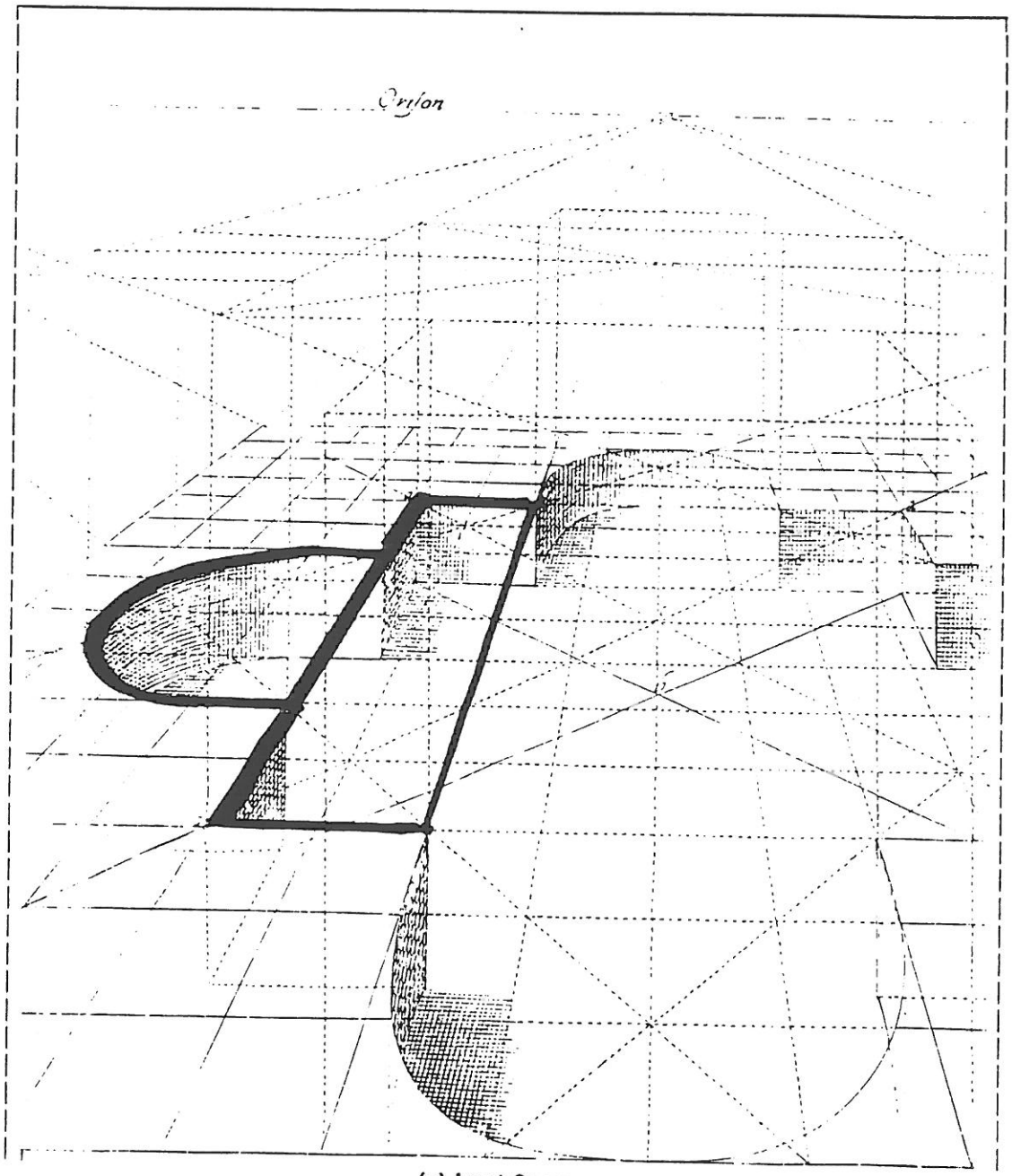
Horn, B. P. K. (1977). Understanding image intensities. *Artificial Intelligence, 8,* 201–231.

Ikeuchi, K. (1987). Generating an interpretation tree from a CAD model for 3D-Object Recognition in bin-picking tasks. *International Journal of Computer Vision, 1*(2), 145–165.

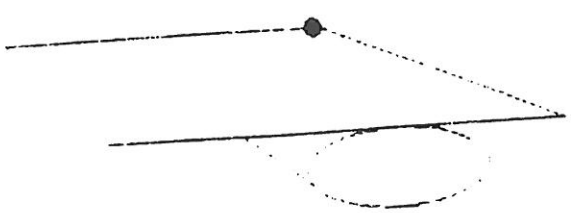
Kanade, T. (1981). Recovery of three-dimensional shape of an object from a single view. *Artificial Intelligence, 17,* 409–460.

Lenat, D. B., & Davis, R. (1982). *Knowledge based systems in artificial intelligence.* New York: McGraw Hill.

McKeown, D. M., Harvey, W. A., & McDermott, J. (1984). Rule based interpretation of aerial imagery. *Proceedings of the IEEE Workshop on Principles of Knowledge Based Systems* (pp. 145–157). Silver Springs, MD: IEEE Computer Society Press.



(a) Input Scene
Bold lines were supplied as input to the system



(b) Interpretation Generated by Spatial Reasoning
Coordinates of indicated point were supplied as a priori information

Figure 8.5. Interpretation of an Outdoor Scene.

Mulgaonkar, P. G. (1984). *Analyzing perspective line drawings using hypothesis based reasoning.* PhD Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Mulgaonkar, P. G., Haralick, R. M., & Shapiro, L. G. (1984). A computational framework for hypothesis based reasoning and its application to computer vision. *First Conference on Artificial Intelligence Applications,* Denver, CO.

Mulgaonkar, P. G., Shapiro, L. G., & Haralick, R. M. (1986). Shape from perspective: A rule-based approach. *Computer Vision, Graphics, and Image Processing, 36,* 298–320.

Mulgaonkar, P. G., Shapiro, L. G., & Haralick, R. M. (1984, March). Matching sticks plates and blobs objects using geometric and relational constraints. *Image and Vision Computing, 1,* 85–98.

Roberts, L. G. (1965). *Machine perception of three-dimensional solids.* Cambridge, MA: MIT Press.

Witkin, A. P. (1981). Recovering surface shape and orientation from texture. *Artificial Intelligence, 17,* 17–46.