AI Magazine Volume 9 Number 2 (1988) (© AAAI)

A Framework for Representing and Reasoning about Three-Dimensional Objects for Vision

Ellen L. Walker, Martin Herman, Takeo Kanade

The capabilities for representing and reasoning about three-dimensional (3-D) objects are essential for knowledge-based, 3-D photointerpretation systems that combine domain knowledge with image processing, as demonstrated by 3-D Mosaic and ACRONYM. Three-dimensional representation of objects is necessary for many additional applications, such as robot navigation and 3-D change detection. Geometric reasoning is especially important because geometric relationships between object parts are a rich source of domain knowledge. A practical framework for geometric representation and reasoning must incorporate projections between a two-dimensional (2-D) image and a 3-D scene, shape and surface properties of objects, and geometric and topological relationships between objects. In addition, it should allow easy modification and extension of the system's domain knowledge and be flexible enough to organize its reasoning efficiently to take advantage of the current available knowledge. We are developing such a framework—the Frame-based Object Recognition and Modeling (3-D FORM) System. This system uses frames to represent objects such as buildings and walls, geometric features such as lines and planes, and geometric relationships such as parallel lines. Active procedures attached to the frames dynamically compute values as needed. Because the order of processing is controlled largely by the order of slot access, the system performs both top-down and bottom-up reasoning, depending on the current available knowledge. The FORM system is being implemented with the Carnegie-Mellon University-built Framekit tool in Common Lisp (Carbonell and Joseph 1986). To date, it has been applied to two types of geometric reasoning problems: interpreting 3-D wire frame data and solving sets of geometric constraints.

We are developing the Frame-based Object Recognition and Modeling (3-D FORM) System, a framework for representing and reasoning about three-dimensional (3-D) objects. This framework incorporates representations of objects, representations of the relationships between them, and a geometric reasoning capability. Such a representation and reasoning capability is essential for knowledge-based, 3-D photointerpretation systems that combine domain knowledge with image processing, as demonstrated by 3-D Mosaic (Herman, Kanade, and Kuroe 1984; Herman and Kanade 1986) and ACRONYM (Brooks 1981). It is also required for other applications, such as robot navigation, 3-D change detection, and the simulation of a scene's appearance from arbitrary viewpoints. The 3-D FORM system uses frames to represent objects such as buildings and walls, geometric features such as lines and planes, and geometric relationships such as parallel lines. Active procedures attached to the frames dynamically compute values as needed. Because processing is controlled by slot access, the system can hypothesize new objects or compute and verify relationships between existing objects, depending on the currently available knowledge.

The 3-D FORM system can include knowledge about model and data objects organized into IS-A and PART hierarchies, along with relationships between object features, and projections used to convert between model objects and data objects. The knowledge includes both generic object models and specific instances of objects. Once the generic objects and the relationships between them are defined, the reasoning portion of the 3-D FORM system can apply the model to real data, recognizing instances of the defined objects and hypothesizing their missing parts.

Figure 1 shows a portion of the knowledge that might be represented about a building. A BUILDING is a 3D-OBJECT; and a WALL, which is a 2D-OBJECT, is one of the building's parts. The specific building, BLDG1, has parts WALL1 and WALL2, whose geometric features are the primitive geometric objects PLANE1 and PLANE2, respectively. In addition to its geometric feature, WALL1 has the photometric feature COLOR1. The knowledge that a building's walls are mutually perpendicular is represented for BLDG1 by the instance PRPW12 of the PERPENDICULAR-PLANES relationship. The arguments to PRPW12 are PLANE1 and PLANE2, the geometric features of BLDG1's walls. The geometric feature PLANE1 is supported by the data object REGION1, which came from an image whose projection between 2-D and 3-D is CAM1.

In the current implementation, only 3-D objects, geometric features, and geometric relationships are represented. The 3-D FORM system was applied to model-based interpretation of 3-D data and to solutions of sets of geometric constraints. This article discusses some of the issues in geometric reasoning for knowledge-based vision systems and how some existing systems have addressed them. Next, it describes the representations of primitive geometric objects, geometric relationships, and composite objects in the 3-D FORM system. Finally, it presents examples of the system's application to geometric reasoning tasks.

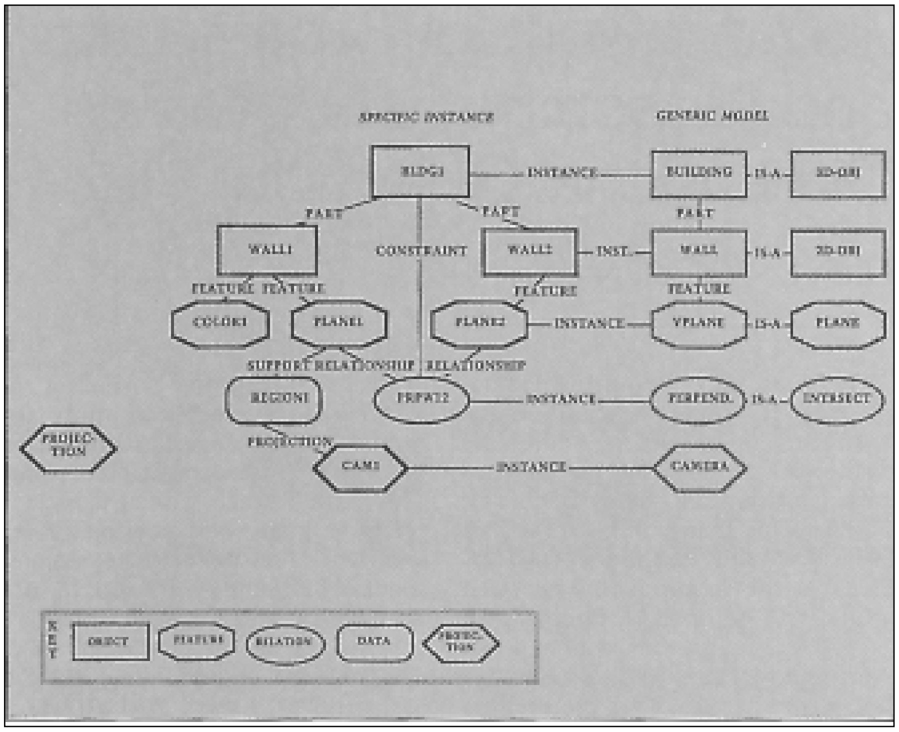


Figure 1. Portion of Knowledge Base for Buildings.

Geometric Reasoning in Knowledge-Based Vision Systems

Domain knowledge was used by earlier vision systems to compensate for the inadequacies of low-level image processing as well as generate reasonable assumptions to make it possible to recover 3-D shape from 2-D data. Because shape is one of the most important cues for object recognition, a knowledge-based vision system must be able to represent and reason about geometric objects. Geometric reasoning helps both data acquisition (bottom-up reasoning) and model matching (top-down reasoning). The overall control of the system should be flexible enough to allow these two processes to be combined to achieve the best results based on the current state of the knowledge base. In addition, the system itself should be domain independent, with the domain-dependent portions collected into a separate replaceable module so that the domain knowledge can easily be modified or extended. Each of the systems described in this section met some of these goals, but no system adequately addressed all of them.

The 3-D Mosaic system (Herman, Kanade, and Kuroe 1984; Herman and Kanade 1986) used 3-D geometric reasoning in the domain of polyhedral building aerial images to acquire a scene description from images from multiple points of view. Using a polyhedral boundary geometric representation, the system hypothesized missing parts of objects in the first view according to a weak model of the urban domain encoded in the program. With the domain models implicit in the system's code and no explicit representation of generic objects, it would be difficult to modify or extend 3-D Mosaic's domain knowledge. For example, it would be a major programming effort to extend 3-D Mosaic to make use of the colors of buildings or the textures of their surfaces. Because it was designed as a model-acquisition system, the 3-D Mosaic system was limited to bottom-up reasoning. When new information invalidated one of the hypotheses generated for missing parts, a complicated network of back-pointers was followed to eliminate the effect of the failed hypothesis.

Unlike the 3-D Mosaic system, ACRONYM (Brooks 1981) used explicit representations of generic objects, representing its geometric objects in a hierarchy of frames. Geometric relationships between objects were represented as quantified algebraic inequalities, and interpretation was done by an external graph matching procedure. To perform the matching, ACRONYM needed strong domain models. The graph matching procedure was primarily top-down, with special low-level objects (ribbons and ellipses) for its generalized cylinder representation of objects. Because the matching procedure was independent of the data, ACRONYM could not organize its search to match the most certain or most complete data first and restrict the search for the remainder of the data. The use of quantifiers removed the constraints one level from the data, making them difficult to read, modify, and extend.

Mundy and others (Barry et al. 1986; Kapur et al. 1985; Mundy 1985) are developing a system that combines algebraic methods for geometric reasoning with a hierarchical organization of knowledge (both object knowledge and knowledge about geometric reasoning). Algebraic constraints from the perspective projection are combined with additional constraints from the model to derive equations describing families of object interpretations. Inequalities from line labeling (Huffman 1971) are then used to constrain the solutions to these equations. Because the relationships, as well as the objects, are represented in a concept hierarchy, the geometric reasoning component should be flexible and extensible. The disadvantage of using algebraic methods is their inefficiency for handling inequalities, an important component of real-world geometric relationships.

Although the SIGMA system (Hwang 1984; Davis and Hwang 1985) used only 2-D geometric reasoning, its method for representing relationships was unique. Each relationship was represented as two procedures attached to its arguments: one for top-down hypothesizing and the other for bottom-up verification. Representing the relationships as active components of the object representation allowed both top-down and bottom-up reasoning, although not at the same time. Only a restricted class of binary relationships was implemented.

The 3-D FORM system uses frames to represent objects such as buildings and walls, geometric features such as lines and planes, and geometric relationships such as parallel lines. Because shape is one of the most important cues for object recognition, a knowledge-based vision system must be able to represent and reason about geometric objects

Like ACRONYM and SIGMA, the 3-D FORM system uses frames to represent its objects. The frames are organized into a hierarchy within which parts and relationships can be inherited. The hierarchical representation also simplifies matching new objects because only objects of the same type can match. The reasoning process used by the 3-D FORM system is an extension and generalization of SIGMA's use of relationships. Frames are used to represent relationships between objects and have active procedures (demons) attached to their arguments so that they are hypothesized or computed as needed. Primitive objects also have demons to compute missing information from known object information. For example, a line has a demon to compute its vector from the known points on the line. Because both object and relationship knowledge are explicitly represented, extending the system to additional domains involves adding new frames but not modifying the code that manipulates the frames. The reasoning process is controlled largely by accessing objects, which are computed as needed; therefore, the representation is equally amenable to top-down and bottom-up processing. In addition, no need exists for an external ordering mechanism such as a focus of attention. Instead, the dynamically computed COMPLETENESS value for each object is used to next select the most complete object to match or relationship to evaluate.

Representing Primitive Geometric Objects

Geometric representation in the 3-D FORM system has three parts: (1) representing primitive geometric objects such as points, lines, and planes; (2) representing primitive geometric relationships between these objects such as parallel lines and perpendicular planes; and (3) combining this information with a part hierarchy to represent composite objects such as faces and buildings. All objects and relationships are represented using frames. The slots of the frames are used to store parameters of the object or relationship. Each slot can have demons associated with it to compute or recompute the slot when necessary. In addition, some slots have facets, which contain constraints on the values that can fill these slots. Frames representing generic objects are arranged in an IS-A hierarchy, and each specific object has an INSTANCE slot pointing back to its generic object. Slots left empty in a particular instance of an object are inherited from the generic object across the INSTANCE link and by means of the IS-A hierarchy.

The primitive geometric objects represented in the current system are points, lines, and planes. For example, the generic line frame shown in figure 2 has slots for points on the line, the line's vector, vectors of lines perpendicular to the line, and the error in fitting a line to the points. The slots PT1, VEC, ERR, and COMPLETENESS have if-needed demons (designated by N in the figure) to determine the value from other slots, as needed. In addition, the slots PT1 and PTS have if-added demons to propagate the new information to other slots in the frame. For example, when additional points are added to a line, the line's old vector and error values are invalidated; so, they are deleted. When one of these values is needed later, it is recomputed by fitting a line to the set of points. Often, the value of an object's slot can be computed in more than one way from other slots of this object. For example, the vector of a line can be computed either by fitting a line to its points or taking the cross-product of its normal vectors. The if-needed demons take into account the available information in choosing a method to compute their results. The use of redundant slots, such as PT1 in the line frame, allows each relationship to choose the most convenient parameterization of the object to work with.

In addition to the slots for the parameters of the object, every geometric object has slots to represent knowledge used in computing its relationships and matching its instances. Currently, we have defined three slots for this purpose: ERR, CONSTRAINTS, and COMPLETENESS.

ERR contains the error in applying the geometric primitive to the given constraints (for example, the error in fitting a line to a set of points). This value is used to constrain matches.

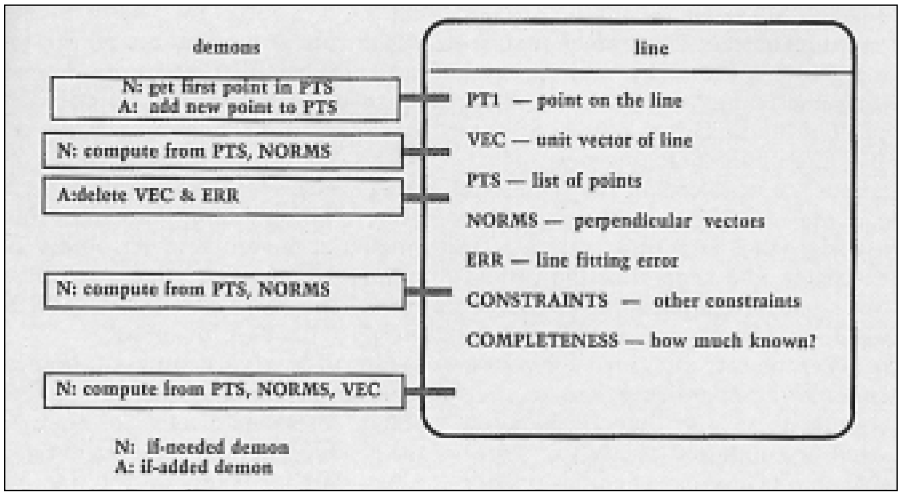

demons
N: get first point in PTS
A: add new point to PTS
N: compute from PTS, NORMS
A:delete VEC & ERR
N: compute from PTS, NORMS
N: compute from PTS, NORMS, VEC
N: if-needed demon
A: if-added demon
line
PT1 — point on the line
VEC — unit vector of line
PTS — list of points
NORMS — perpendicular vectors
ERR — line fitting error
CONSTRAINTS — other constraints
COMPLETENESS — how much known?

Figure 2. Representation of a Line.

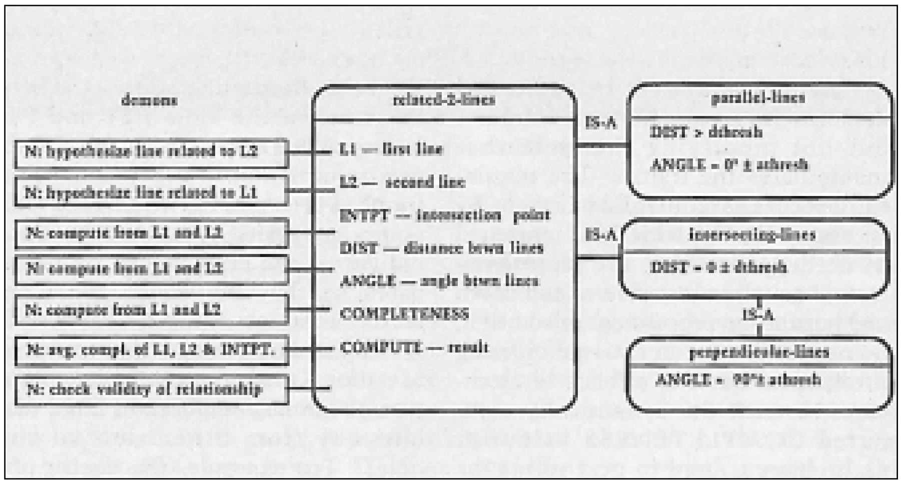

demons
N: hypothesize line related to L2
N: hypothesize line related to L1
N: compute from L1 and L2
N: compute from L1 and L2
N: compute from L1 and L2
N: avg. compl. of L1, L2 & INTPT.
N: check validity of relationship
related-2-lines
L1 — first line
L2 — second line
INTPT — intersection point
DIST — distance btwn lines
ANGLE — angle btwn lines
COMPLETENESS
COMPUTE — result
IS-A
parallel-lines
DIST > dthresh
ANGLE = 0° ± athresh
IS-A
intersecting-lines
DIST = 0 ± dthresh
IS-A
perpendicular-lines
ANGLE = 90°± athresh

Figure 3. Relationships Between Two Lines.

CONSTRAINTS contains geometric constraints that cannot be represented by filling in any other slot. These constraints allow relationships to affect later matching. Each value consists of a function to compute the constraint and a pointer to the relationship that caused it. An object's constraints are evaluated whenever sufficient information is added to the object. For example, if two lines are supposed to intersect, but neither has any points specified yet, a constraint is placed on each line consisting of a function to compute the distance between the lines and a pointer back to the intersection relationship. When one line is further specified, the distance function is executed. If the distance is small enough, the constraint of intersection is satisfied and therefore, removed. If the distance is too large, an error is returned to the process that changed the line. If the other line is not yet specified, the original relationship is reevaluated to put a new constraint on the other line.

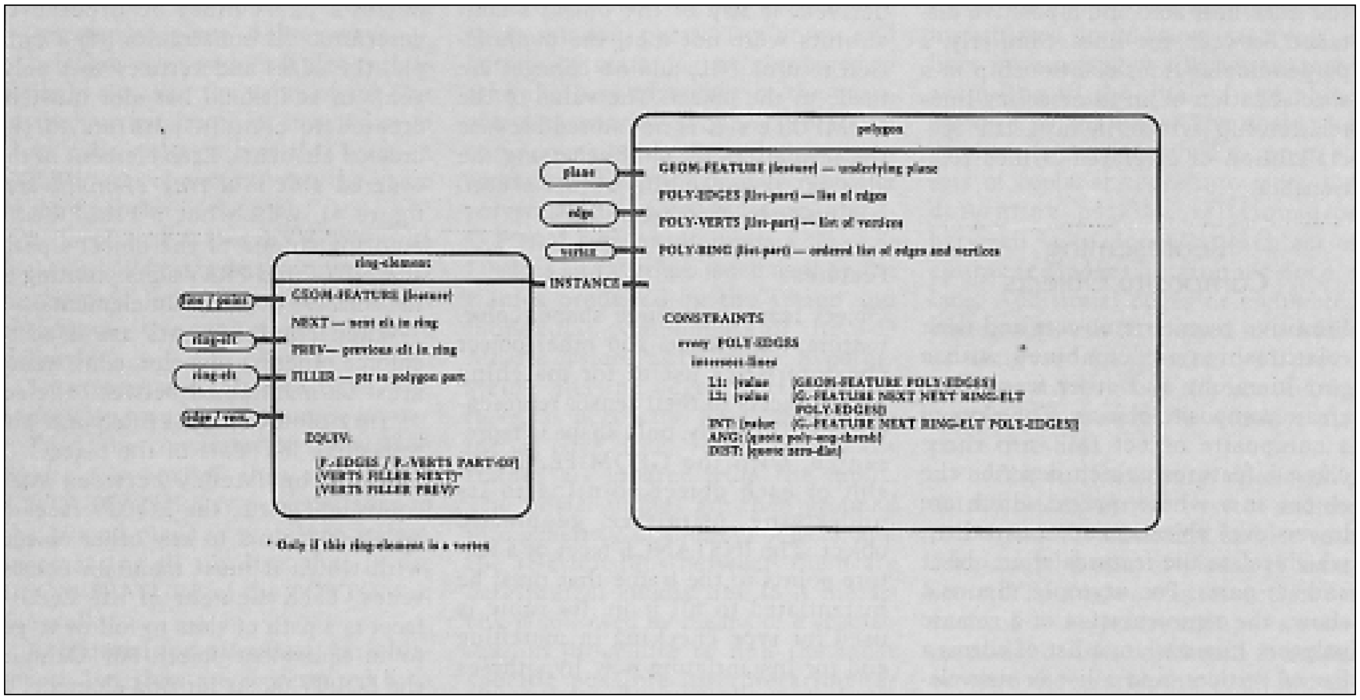
COMPLETENESS contains a user-defined measure of the information stored in the object. This value is used for sorting relationship computation and matching operations so that the most complete items are tried first. For example, the completeness of a line is greatest if two or more points are known, but it is greater if one point and the vector are known than if only one point or the vector is known.

Representing Primitive Geometric Relationships

Like geometric objects, primitive geometric relationships are represented by frames. The system currently considers relationships between pairs of lines, pairs of planes, lines and the planes they lie in, and points and the lines they lie on. Each frame representing a primitive geometric relationship has slots for two or more geometric objects for which the relationship is defined, one or more numeric ranges for parameters of the relationship, a COMPLETENESS slot, and a COMPUTE slot. In the related-2-lines relationship shown in figure 3, the slots L1, L2, and INTPT contain objects, and slots DIST and ANGLE contain parameters of the relationship between the objects.

The COMPLETENESS slot and the COMPUTE slot are computed only when needed. A demon attached to the COMPLETENESS slot of each relationship computes the average completeness value of the geometric object arguments of the relationship. A demon attached to the COMPUTE slot of each relationship evaluates the relationship. The evaluation function first attempts to fill in any missing slots by hypothesizing geometric objects or computing numeric ranges. When objects are hypothesized, only the slot values that are known are filled in. After attempting to hypothesize each missing argument, the evaluation function adds constraints derived from the relationship to each geometric object. For example, the perpendicular-lines relationship adds the vector of L1 to the norms of L2, the vector of L2 to the norms of L1, and the coordinates of INTPT to both lines. If the geometric arguments of the relationship are not fully specified, as much constraint as possible is applied to the remaining geometric objects. Finally, the evaluation function computes the true values for the numeric arguments of the relationship and determines whether they fall within the specified ranges.

The related-2-lines relationship has several specializations, also shown in figure 3. The parallel-lines relationship is a related-2-lines relationship specialized to have the angle between



polygon
plane
GEOM-FEATURE [feature] — underlying plane
edge
POLY-EDGES [list-part] — list of edges
POLY-VERTS [list-part] — list of vertices
vertex
POLY-RING [list-part] — ordered list of edges and vertices
INSTANCE
ring-element
line / point
GEOM-FEATURE [feature]
NEXT — next elt in ring
ring-elt
PREV— previous elt in ring
ring-elt
FILLER — ptr to polygon part
edge / vert.
EQUIV:
[P.-EDGES / P.-VERTS PART-OF]
(VERTS FILLER NEXT)*
(VERTS FILLER PREV)*
* Only if this ring-element is a vertex
CONSTRAINTS
every POLY-EDGES
intersect-lines
L1: [value [GEOM-FEATURE POLY-EDGES]]
L2: [value [G.-FEATURE NEXT NEXT RING-ELT POLY-EDGES]]
INT: [value [G.-FEATURE NEXT RING-ELT POLY-EDGES]]
ANG: [quote poly-ang-thresh]
DIST: [quote zero-dist]

Figure 4. Representation of a Generic Polygon.

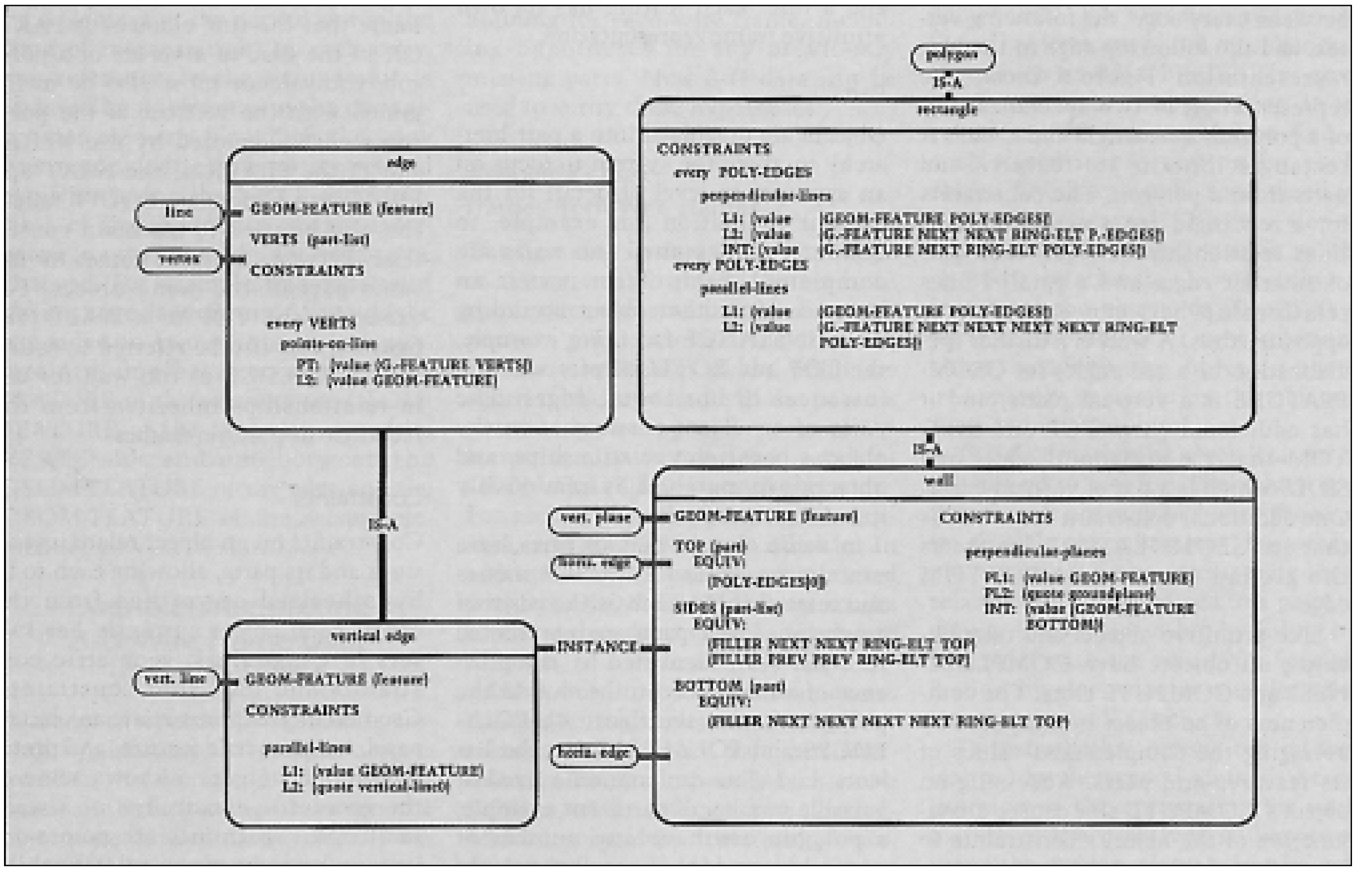


Figure 5. Representations of a Wall, a Rectangle, a Vertical Edge, and an Edge.

the lines near zero and a positive distance between the lines. Similarly, a perpendicular-lines relationship is a specialization of an intersecting-lines relationship, which, in turn, is a specialization of a related-2-lines relationship.

Representing Composite Objects

Primitive geometric objects and their relationships are combined with a part hierarchy and other features to create composite objects. The slots of a composite object fall into three classes: features, which describe the object as a whole; parts, which are lower-level objects; and constraints, which relate the features of an object and its parts. For example, figure 4 shows the representation of a generic polygon. Its parts are a list of edges; a list of vertices; and a list of ring elements, which link the polygon's edges and vertices in order. Its geometric feature is a plane. Its constraints are an intersecting-lines relationship between every edge, the following vertex, and the following edge in the ring representation. Figure 5 shows the representation of two specializations of a polygon: a rectangle and a wall. A rectangle inherits its features and parts from a polygon. The constraints for a rectangle are a perpendicular-lines relationship between each pair of adjacent edges and a parallel-lines relationship between each pair of opposite edges. A wall is a further specialization of a rectangle; its GEOM-FEATURE is a vertical plane, and it has additional parts TOP and BOTTOM that are horizontal edges and SIDES which is a list of vertical edges. One additional constraint for a wall is that its GEOM-FEATURE intersects the ground plane at its BOTTOM edge.

Like primitive objects and relationships, all objects have COMPLETENESS and COMPUTE slots. The completeness of an object is computed by averaging the completeness values of its features and parts. Accessing an object's COMPUTE slot causes a conjunction of the object's constraints to be evaluated. As a side effect of computing an object, hypotheses for the object's parts and features can be derived. If any of the object's constraints were not met, the conjunction returns NIL, and no changes are made to the object. The value of the COMPUTE slot is not stored because the primary purpose of accessing the slot is to cause the object's constraints to be evaluated.

Features

Object features include shape, color, texture, reflectance, and other object characteristics useful for matching world objects to their sensor representations. Currently, only shape is represented, with the GEOM-FEATURE slot of each object pointing to its underlying primitive geometric object. The INSTANCE facet of a feature points to the frame that must be instantiated to fill it in. Its value is used for type checking in matching and for instantiating new hypotheses for features. In figure 5, for example, the geometric features of a wall, a vertical edge, and an edge, respectively, are a vertical plane, a vertical line, and a line. Each feature has its own primitive frame representation.

Parts

Objects are organized into a part hierarchy to allow the system to focus on an appropriate level of detail for the current evaluation (for example, to ignore windows until the walls are completed). Each object part is an instance of another object according to its INSTANCE facet. For example, the TOP and BOTTOM of a wall are instances of horizontal edges. The parts of an object participate in the object's constraint relationships, and objects are matched by recursively matching their parts.

In some objects, sets of parts have exactly the same INSTANCE values and relationships, such as the edges of a polygon. These parts are represented in a list slot, identified by the presence of a LIST facet in the slot. In the polygon frame (see figure 4), POLY-EDGES and POLY-VERTS are the list slots. List slots can contain a fixed or variable number of parts. For example, a polygon can have any number of edges, but an edge always has exactly two vertices. The order of elements in list slots does not matter. When an object's parts must be ordered to determine its constraints (for example, the edges and vertices of a polygon), an additional list slot must be created to contain pointers to the ordered elements. Each element of the ordered slot is a *ring element* (see figure 4), which has a FILLER slot pointing to one of the object's parts and NEXT and PREV slots pointing to the adjacent ordered slot elements.

When ring elements are used to enforce ordering in a slot, consistency must be maintained between the set of ring-element FILLERs, and the respective list slots of the object. To enforce consistency between such equivalent parts, the EQUIV facet of an object points to any other objects with which it must maintain consistency. Each element of the EQUIV facet is a path of slots to follow to get to an equivalent object. For example, the EQUIV facets for ring elements of a polygon are shown in figure 4. The FILLER of a ring element is equivalent to a member of either the POLY-EDGES or POLY-VERTS slot of the frame that the ring element is PART-OF. In the case of a vertex of a polygon, equivalence must also be maintained with the vertices of the polygon's edges, denoted by the VERTS slot of the FILLER of the NEXT and PREV ring elements. EQUIV facets can also be used to maintain consistency between different names for the same part of the same object. For example, the TOP of a WALL (see figure 5) can also be referred to as the first POLY-EDGE of the wall for use in relationships inherited from the rectangle or polygon frames.

Constraints

Constraints on an object relate its features and its parts, allowing each to be hypothesized or verified from the other. Each object currently has two sets of constraints: geometric constraints and inclusion constraints. *Geometric constraints* relate an object's parts, its geometric feature, and prototype frames. Figure 5 shows some of the geometric constraints of a wall. *Inclusion constraints* are points-on-line or lines-in-plane relationships between the object's geometric feature and its parts. Each constraint is a template for a relationship specifying its arguments in one of three ways.

1. (**value** <slot-path>): If slot-path is a single slot, use the value of the slot in the current frame. If it is a list of slots, follow the path as if it were an EQUIV facet element, using the value of the final slot in the path.

2. (**local** <var>): Use the value of the local variable var, initializing to NIL if necessary. Local variables persist throughout the conjunction they are defined in.

3. <expression>: Evaluate the expression, usually a prototype frame.

Thus, the constraint of the wall in figure 5 specifies that the wall's GEOM-FEATURE is perpendicular to the ground plane (a prototype frame), intersecting in the line that is the GEOM-FEATURE of the BOTTOM of the wall.

Relationships affecting variable-length list slots are represented by a template preceded by the word **every** and the name of the slot. When such a template is instantiated, a distinct copy of the relationship is made for each element of the named slot at the time of instantiation. In each copy, any reference to the named slot is replaced by a reference to the particular slot element for which the copy was made. List elements are related explicitly using the NEXT and PREV slots of ring elements so that references to other slots do not need to be changed. For example, the constraint on an edge (see figure 5) would be expanded to two points-on-line constraints, one between the GEOM-FEATURE of the edge and the GEOM-FEATURE of the first element of its VERTS slot and one between the GEOM-FEATURE of the edge and the GEOM-FEATURE of the second element of its VERTS slot.

Example: Interpreting 3-D Wire Frame Data with 3-D FORM

A model of a particular domain is created by defining the generic objects found in this domain. In addition, features and relationships between the objects are defined using the facilities described in the previous three sections. The 3-D FORM system applies the domain model to real-world data to recognize objects and hypothesize their missing parts. Top-down and bottom-up reasoning are combined to take the best advantage of the available data, controlled by the procedural component of the knowledge representation. Given the domain model of polygonal, flat-roofed buildings, the 3-D FORM system interprets a set of 3-D edges and vertices (such as the wire frames produced by the stereo and monocular components of the 3-D Mosaic system) as buildings, hypothesizing missing edges, vertices, and faces, as necessary. First, appropriate initial edge, line, vertex, and point frames are created from the input. The initial frames are then grouped into generic 2-D and 3-D objects, and the relationships between them are determined. Finally, the IS-A hierarchy is followed by means of a specialization procedure to find the most specific possible interpretation for each object and to fill in its slots. Once all input features are placed into object slots, the top-level objects are computed. The result is a completed building for each wire frame, including hypotheses for any previously missing parts. New 3-D data can be used to verify these hypotheses.

Acquiring Object Frames from Wire Frames

The first step in data interpretation is to create initial object frames from the input points and lines. For each point, a point frame is instantiated, and the coordinates of the point are added to the new frame. If the point is a vertex between two lines, a vertex frame is also instantiated, and its GEOM-FEATURE is set to the point. For each line, a line frame is instantiated, and its PTS slot is filled in. In addition, an edge frame is instantiated with its GEOM-FEATURE set to the line, and the edge's vertices, if any, are filled in.

Next, the initial object frames are grouped into complex objects, and the relationships between them are determined. The dual space (Mackworth 1973) is used to efficiently find parallel and coincident lines and planes. Each new line is added to a dual-space database. For each pair of parallel or coincident lines found in the dual space, an appropriate relationship is instantiated. In addition, each pair of lines intersecting at a vertex is stored according to the dual of the plane spanned by the lines. The dual-space database is then searched to group all sets of coplanar edges into faces and determine parallel relationships between faces. Initially, each set of connected edges is grouped into a face. Additional edges or connected edge groups can be merged into a face as long as the greatest distance between any point on the face and the centroid of the face is within a distance threshold.

Finally, an intersection relationship is instantiated for each vertex and the pair of lines it intersects and for each edge and the pair of planes it intersects. The angle of intersection for each of these relationships is automatically computed when it is needed or when the relationship itself is computed.

Specializing Objects and Relationships

After initial object creation and grouping, all objects are of the most general type, such as 3D-OBJECT. The next process in data interpretation is to search the IS-A hierarchy to find a specialized interpretation for each object. An object can be specialized in one of three ways: (1) fill in a slot of the object with a more specific value, (2) add a relationship constraining a feature of the object, or (3) add new parts to the object or new relationships between the object's parts (recursively specializing the object's parts and relationships).

The first method of specialization is the easiest to test for. The values in a frame's slot are matched with those of its possible specializations. For example, in figure 3, the intersecting-lines relationship is specialized to a perpendicular-lines relationship by filling its angle slot with the value 90°. Specialization by slot value is used for relationships as well as objects.

To test whether objects can be specialized by the second method, a conjunction of the constraints for each feature of the new type is computed using the current object's feature values. If this conjunction computes successfully for all features of the new

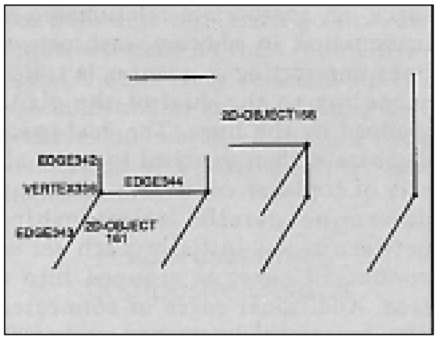
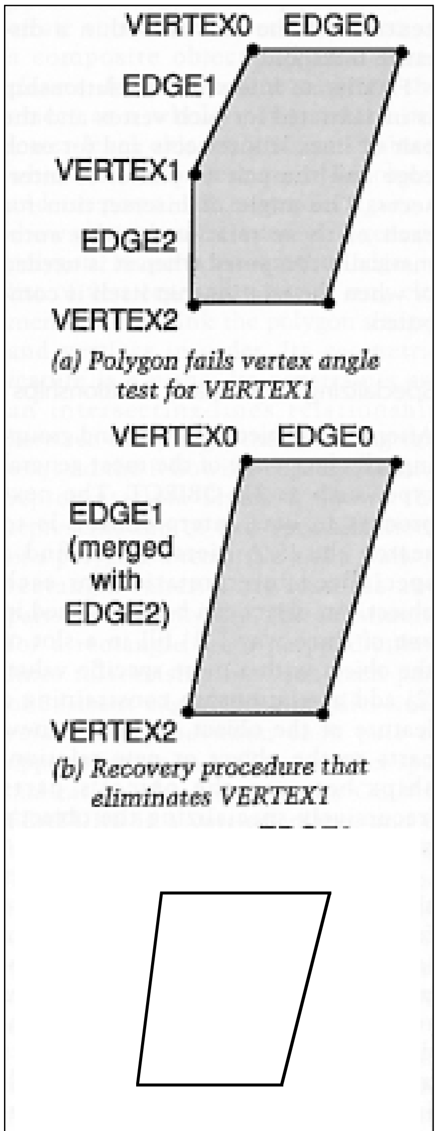


Figure 6. Object to be Specialized.



(a) Polygon fails vertex angle test for VERTEX1

(b) Recovery procedure that eliminates VERTEX1

(c) Recovery procedure that eliminates EDGE1 and its vertices.

Figure 7. Two Recovery Procedures for Polygons.

type, the object can be specialized. Thus, in figure 5, an edge is specialized to a vertical edge by adding a relationship constraining its GEOM-FEATURE to be parallel to the prototype vertical line.

To specialize an object by the third method, a correspondence between the parts of the candidate object and the parts of the specialized object is determined so that each part is an instance of the right object, and all constraints are satisfied. For example, to specialize a rectangle to a wall, the slots TOP, BOTTOM, and SIDES and the relationships between them are added. The correspondence of parts to slots is done in two phases. First, a list of matches using only local considerations is made; then, this list is pruned by propagating relationship information. The considerations for local matching are (1) For each part, which slots have the right INSTANCE value? (2) If a slot is filled in, can the part be successfully matched with the slot's current value?

For example, when specializing the face 2D-OBJECT161 in figure 6 to a wall, the local match for EDGE343 is SIDES (see figure 5) because EDGE343 is a vertical edge. Although EDGE344 (a horizontal edge) has the right INSTANCE value for both TOP and BOTTOM of the wall, the BOTTOM slot is already filled with the intersection of the wall plane and the ground plane. Because EDGE344 lies above the ground, its only possible local match is TOP.

If at least one possible match exists for each part, then relationship information is propagated by assigning one part to one of its possible slots and pruning the possibilities for the other parts according to its relationships. This pruning is done by matching the relationships of the current part with the relationship templates of its assigned slot. For example, when specializing 2D-OBJECT158 to a polygonal roof, any of the edges of 2D-OBJECT158 can match any of the edge slots of the roof after local matching. However, once EDGE344 is assigned to an edge of the roof, EDGE342, which intersects it at VERTEX336, must be assigned either to the NEXT of the NEXT of the edge or to the PREV of the PREV of the edge, with VERTEX336 as the NEXT or the PREV, respectively. From the remaining possibilities, a new assignment is chosen, and the propagation process is repeated until all parts are matched, all slots are filled, or a part cannot be matched. If there is an unmatched part, and not all the slots were filled, the specialization fails. Otherwise, hypotheses for missing parts of the object can be generated by accessing its COMPUTE slot. Two facets in the PARTS slot of the object are used to store information in case computing the object fails: The MATCHES-TRIED facet contains matches already tried, and the LOCAL-MATCH facet contains the original set of local matches for each part.

Controlling the Matching Process

Because matching is expensive, it is advantageous to limit the number of object pairs to be matched. One way the 3-D FORM system limits the number is to specialize each data object as much as possible before any matching is attempted. Because only instances of the same generic objects can possibly match, specializing an object limits its possible matches. A second method of limiting the number of object pairs to match is to consider the relationships between the parts being matched, eliminating object pairs with conflicting relationships. These two methods of eliminating matches correspond to the two conditions for local matching used in specialization. However, even after local matching, multiple possibilities often remain.

Once it is determined that general matching must be done, processing is limited by ensuring that if a match eventually fails, it fails as early as possible, cutting off the recursion tree near the top. Because empty objects match anything, the more complete an object is, the less likely it is to match a given object. Therefore, whenever a list of possible matches to be tried exists, the matches are sorted by the object's COMPLETENESS values, and the pair with the greatest average completeness is tried first. In the case of parts being matched to slots, these heuristics are applied by doing the local matches first; choosing the object with the smallest number of possibilities; and finally, choosing the possibility with the greatest average completeness.

Recovering from Constraint Failures in Object Completion

Ordinarily, failure of a constraint during the completion of an object causes the completion to fail. However, some objects can be dynamically reconfigured during completion to satisfy the failed constraint. Recovery procedures for these objects are stored in the RECOVERY facet of their GEOM-CONSTRAINT slot. Each recovery procedure determines the reason for constraint failure and then applies the appropriate reconfiguration technique. For example, in the POLYGON frame, a constraint prevents the angle of intersection at each vertex from being too near 180°. A polygon that fails this constraint is shown in figure 7a.

Two recovery procedures for polygons were defined. One procedure eliminates the vertex at which the constraint fails, merging the edges that intersected to form the vertex. This method is illustrated in figure 7b. The second recovery method eliminates one of the edges at the vertex, forming a new vertex at the intersection of the preceding and following edges. In figure 7c, EDGE1 is eliminated this way. Either procedure can create an acceptable polygon from one that fails the vertex angle constraint. The recovery procedure that eliminates an edge is also appropriate when a polygon edge fails the length constraint by being too short.

Examples of 3-D Wire Frame Interpretation

This section describes experiments in which 3-D wire frames, generated from image edges by hand, were read into the frame database, specialized, and evaluated to generate hypotheses for missing edges. In the first experiment, a new edge was then entered manually, and the system matched it to one of the hypothesized object edges. The initial wire frame for the first experiment consisted of 12 edges, 7 horizontal and 5 vertical (see figure 8a). During the initial processing, the horizontal edges were combined into one face (2D-OBJECT158), and six vertical faces, including 2D-OBJECT161, were created from the vertical edges and the horizontal edges that they intersected. Because each face intersected at least one of the other faces at an edge, a 3D-OBJECT was created with all seven faces as its parts.

In the specialization process, the edges were divided into horizontal and vertical edges, and 2D-OBJECT161 was found to be a wall because it was a vertical face. Because 2D-OBJECT158 is above the ground plane, it was found to be the roof rather than the floor of the building. The parts of each of these objects were assigned to slots, as discussed in Specializing Objects and Relationships. The final result of specialization was an incomplete polygonal building with 2D-OBJECT158 as its roof.

Next, the building's constraints were evaluated, providing hypotheses for its missing slots. The result of this evaluation is shown in figure 8b. Some details of the completion of 2D-OBJECT158 as a polygonal roof are shown in figure 9.

Figure 9a shows the roof after specialization. Because it was found to be a general polygon, a vertex was hypothesized for each free endpoint. In figure 9b, EDGE576 is hypothesized to extend from VERTEX531 to VERTEX532, using the points-on-line relationships from the EDGE frame. The intersecting-lines relationship at VERTEX531 failed; so, the polygon recovery procedure deleted VERTEX531, merging EDGE576 with EDGE341, as seen in figure 9c. The final result of completion is shown in figure 9d. Notice that VERTEX528, which also failed the angle constraint, was eliminated, as was EDGE661, which was too short.

Once the building's constraints were evaluated, the capability of the system to revise its knowledge based on new data was tested. A new 3-D edge, EDGE865, was entered and matched to the current building hypothesis. The algorithm involved taking the new data, specializing it as much as possible, and attempting to match the top-level object to all other instances of the same object until a

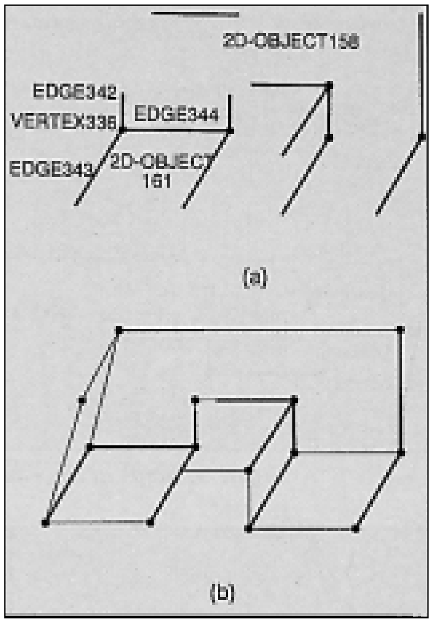


Figure 8. (a) Initial Wire Frame; (b) Visible Faces of Completed Building.

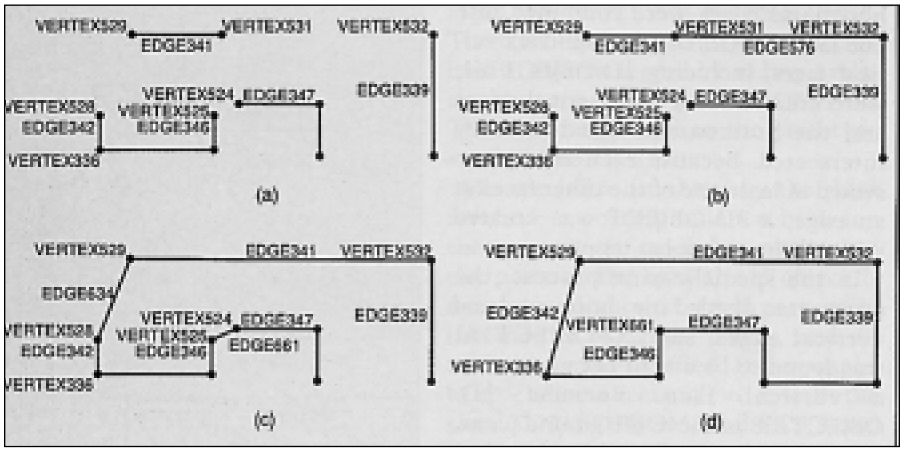


Figure 9. Steps in the Completion of a Polygonal Roof.

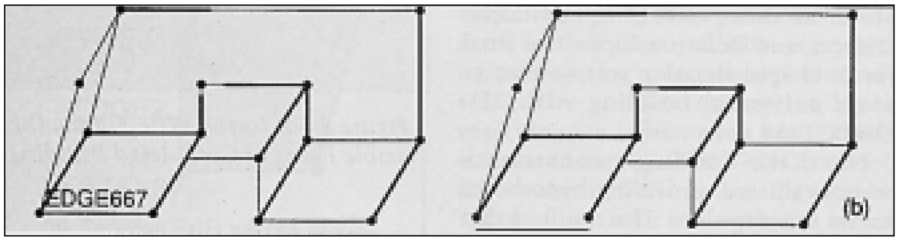


Figure 10. (a) Building from Figure 8 with New Edge; (b) Revised Building after Merging New Edge.

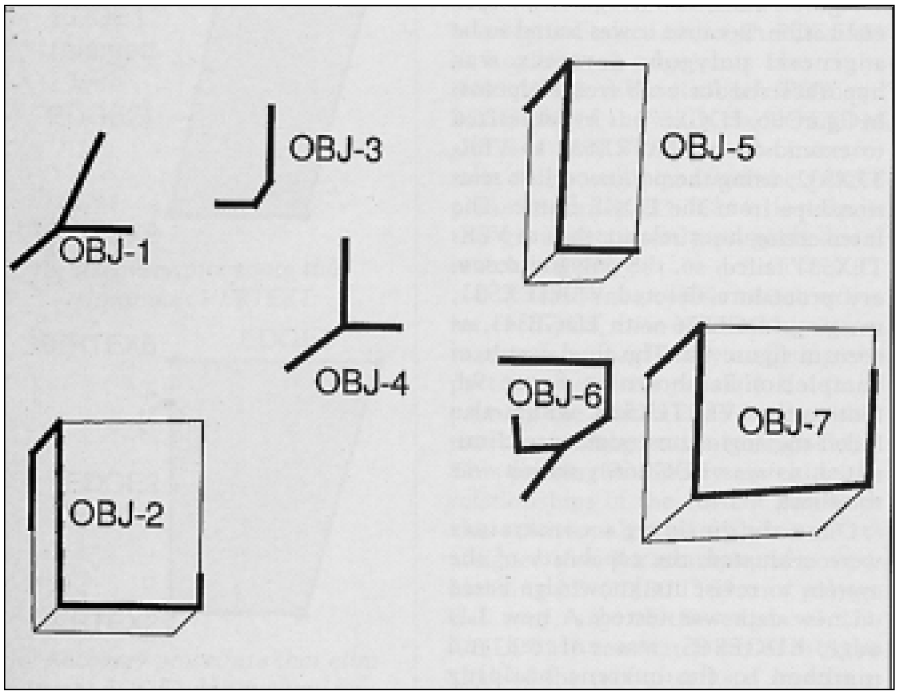


Figure 11. Wire Frames and Hypotheses for a Group of Buildings.

match was found. The new edge was found to match the hypothesized edge, EDGE667. Figure 10a shows the building and the new edge, and figure 10b shows the result of merging the new edge with the old using this algorithm. Because the vertices of the new edge were close enough to it by the tolerances given to the system, the vertices were not moved.

The second experiment evaluated a realistic set of wire frames using a rectangular building model. Three buildings were found in the image, and two objects were rejected because they did not fit the rectangular building model. Figure 11 shows the results of evaluating these wire frames. The bold lines are the initial wire frames; the remaining lines were hypothesized when the buildings were evaluated. Objects 1 and 6 were rejected as buildings because each had a nonperpendicular vertex. Objects 3 and 4 were considered too small. The remaining objects were successfully completed.

Example: Solving Sets of Geometric Constraints with 3-D FORM

The model-based reasoning done by the 3-D FORM system is not limited to physical geometric models. Any set of geometric constraints can be modeled as a single object whose constraints are given and whose parts are the variables for the constraints. The constraints are applied to a given situation by instantiating the model, filling in the known parts, and computing the object. The known parts can be partially specified objects. Missing information in the resulting object is computed when needed by evaluating the constraints.

For example, consider the basic shadow problem, as defined by Shafer (1983), illustrated in figure 12a. A single intersecting pair of edges (Eo1 and Eo2, intersecting at Io12) on an object surface (So) casts corresponding shadow edges (Es1 and Es2, intersecting at Is12) on a shadow surface (Ss). Two illumination surfaces (Si1 and Si2, shown in figure 12b) are defined by the planes of light through corresponding object and shadow edge pairs. The illumination surfaces inter-

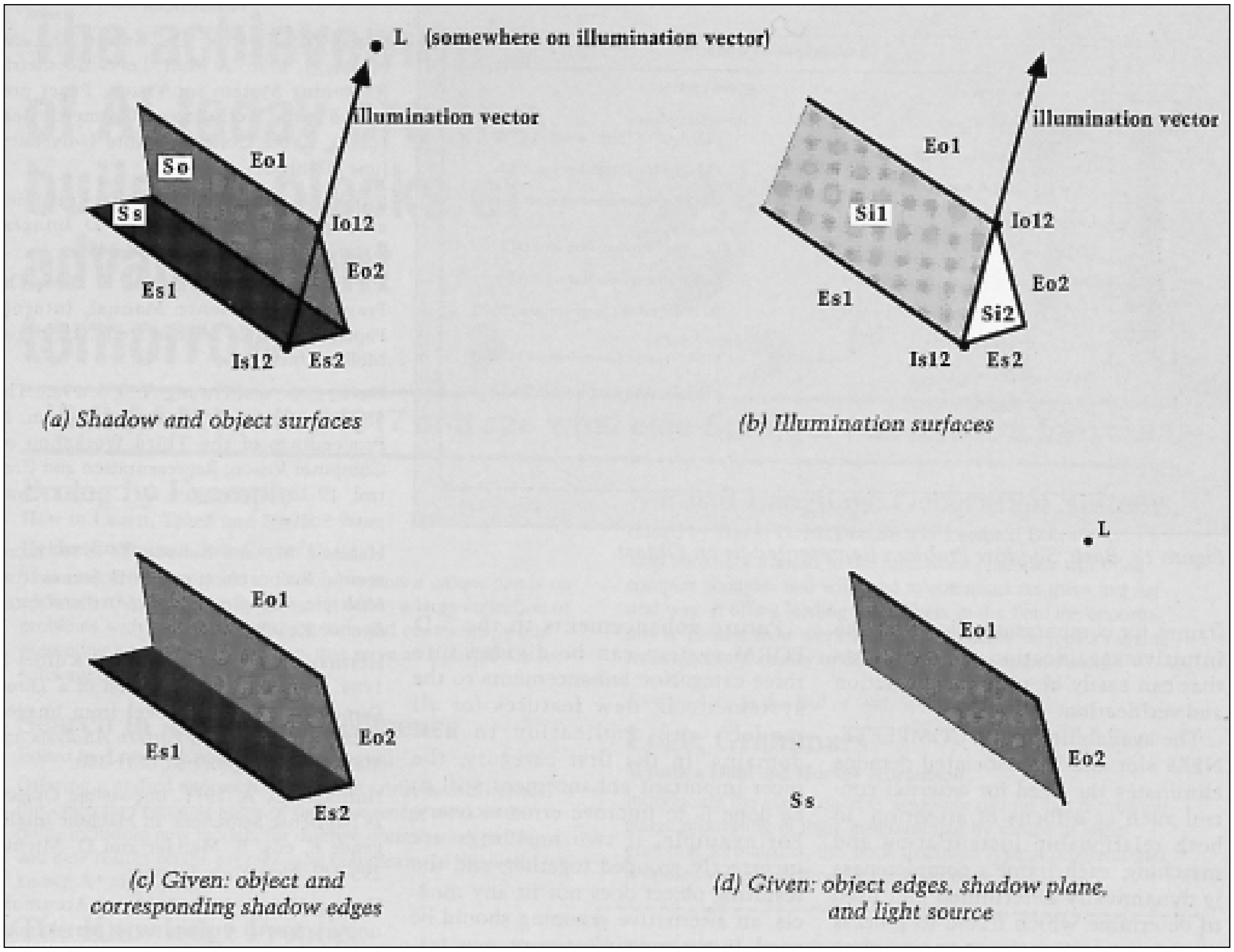

L (somewhere on illumination vector)
illumination vector
So
Eo1
Ss
Io12
Eo2
Es1
Is12
Es2
(a) Shadow and object surfaces
illumination vector
Eo1
Si1
Io12
Es1
Eo2
Si2
Is12
Es2
(b) Illumination surfaces
Eo1
Es1
Eo2
Es2
(c) Given: object and corresponding shadow edges
L
Eo1
Eo2
Ss
(d) Given: object edges, shadow plane, and light source

Figure 12. Solving Instances of the Basic Shadow Problem.

sect in the line through Io12 and Is12, which is the illumination vector from the light source L. This set of constraints is represented as the frame BASIC-SHADOW-PROBLEM, with slots EO1, EO2, IO12, SO, ES1, ES2, IS12, SS, SI1, SI2, ILLUM, and L (see figure 13). Some of the constraints of the basic shadow problem are intersect-lines between EO1 and EO2 intersecting in IO12, intersect-lines between ES1 and ES2 intersecting in IS12, and points-on-line with L on ILLUM.

Figure 12 shows the solution of two instances of the shadow problem. The first (figure 12c) instance has the object and shadow edges given and must determine the surfaces, the illumination vector, and the location of the light source. All are determined exactly except for the light source, which is represented as a point with unknown coordinates but constrained to lie on the illumination vector. Figure 12d shows the second instance, with the object edges and shadow surface given. In this case, the shadow edges, the object and illumination surfaces, and the illumination vector are determined. The final result for both instances of the problem was the same as the initial definition shown in figures 12a and 12b.

Although basic-shadow-problem objects do not correspond to any objects in the real world, they could be created and evaluated along with real objects to add constraint to an image interpretation. For example, the low-level procedures that now provide wire frames could also provide marked shadow edges. The shadow edges would be used as the seeds in a grouping procedure to create basic-shadow-problem objects. The basic-shadow-problem objects, when evaluated, would constrain their parts, which, in turn, would constrain real objects, and so on.

Conclusions

Experiments to date with the 3-D FORM system have shown that frame-based modeling with constraints provides a powerful and flexible geometric reasoning system. The system was demonstrated using two problems: the interpretation of 3-D wire frames as polygonal or rectangular flat-roofed buildings and the solution of the constraints in the basic shadow problem. In each case, both the prediction and the verification of information were automatically done, as needed, by

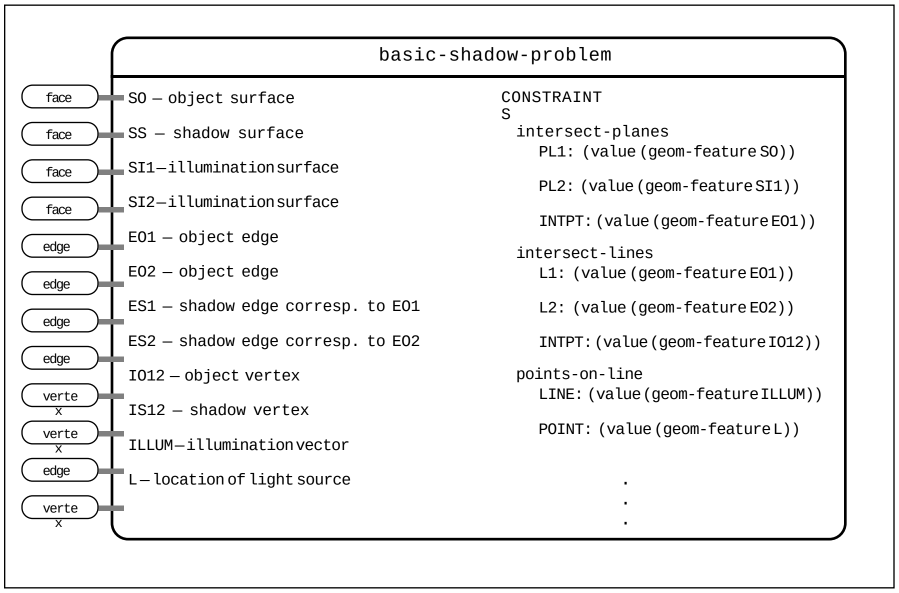

basic-shadow-problem
face
face
face
face
edge
edge
edge
edge
vertex
vertex
edge
vertex
SO – object surface
SS – shadow surface
SI1–illumination surface
SI2–illumination surface
EO1 – object edge
EO2 – object edge
ES1 – shadow edge corresp. to EO1
ES2 – shadow edge corresp. to EO2
IO12 – object vertex
IS12 – shadow vertex
ILLUM–illumination vector
L – location of light source
CONSTRAINTS
intersect-planes
PL1: (value (geom-feature SO))
PL2: (value (geom-feature SI1))
INTPT: (value (geom-feature EO1))
intersect-lines
L1: (value (geom-feature EO1))
L2: (value (geom-feature EO2))
INTPT: (value (geom-feature IO12))
points-on-line
LINE: (value (geom-feature ILLUM))
POINT: (value (geom-feature L))
.
.
.

Figure 13. Basic Shadow Problem Represented as an Object.

evaluating the object frames. The use of demons attached to relationship frames for computation allows for the intuitive specification of constraints that can easily be used for prediction and verification.

The availability of the COMPLETENESS slot and its associated demons eliminates the need for external control such as a focus of attention. In both relationship instantiation and matching, each frame's completeness is dynamically determined and used to determine which frame to process next. Instantiating the most complete relationship at each time forces the maximum amount of knowledge to be used in making each hypothesis. Matching the most complete objects first enhances the possibility of either a quick failure or a complete success for each match.

Because of the hierarchical nature of the model, objects to be matched could be specialized first, then matched to a smaller number of possible objects of the more specialized type. The hierarchy also allows a form of incremental interpretation. For example, because a wall is a specialization of a rectangle, no object will even be tried as a wall unless it has already passed the qualifications for a rectangle. The inheritance allowed by the IS-A slot simplifies the process of creating new frames because only the differences from an existing frame must be specified.

Future enhancements to the 3-D FORM system can be divided into three categories: enhancements to the system itself, new features for all models, and application to new domains. In the first category, the most important enhancement still to be done is to improve error recovery. For example, if two buildings are incorrectly grouped together, and the resulting object does not fit any models, an alternative grouping should be tried. In the second category, new features that could be added include descriptions of viewpoints, projections, and photometric features. The models defined for the current system could be expanded by adding objects that contain multiple buildings (such as city blocks) or creating representations for curved surfaces.

Acknowledgments

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA), ARPA Order No. 4976, and monitored by the Air Force Avionics Laboratory, under contract F33615-84-K-1520. Ellen Walker is also supported by a fellowship from the AT&T Bell Laboratories Graduate Research Program for Women. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. government.

References

Barry, M.; Cyrluk, D.; Kapur, D.; and Mundy, J. 1986. A Multi-Level Geometric Reasoning System for Vision. Paper presented at the Workshop on Geometric Reasoning, Keble College, Oxford University, June 30-July 3.

Brooks, R. A. 1981. Symbolic Reasoning among 3-D Models and 2-D Images. *Artificial Intelligence* 17:285-348.

Carbonell, J. G., and Joseph, R. 1986. The FrameKit+ Reference Manual, Internal Paper, Computer Science Dept., Carnegie-Mellon Univ.

Davis, L. S., and Hwang, V. S. S. 1985. The SIGMA Image Understanding System. In Proceedings of the Third Workshop on Computer Vision: Representation and Control, 19-26. Washington D. C.: IEEE Computer Society.

Herman, M., and Kanade, T. 1986. Incremental Reconstruction of 3-D Scenes from Multiple, Complex Images. *Artificial Intelligence* 30:289-341.

Herman, M.; Kanade, T.; and Kuroe, S. 1984. Incremental Acquisition of a Three-Dimensional Scene Model from Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6(3):331-340.

Huffman, D. A. 1971. Impossible Objects as Nonsense Sentences. In Machine Intelligence 6, eds. B. Meltzer and D. Michie, 295-323. New York: Elsevier.

Hwang, V. S. S. 1984. Evidence Accumulation for Spatial Reasoning in Aerial Image Understanding. Ph.D diss., Dept. of Computer Science, Univ. of Maryland.

Kapur, D.; Mundy, J.; Musser, D.; and Narendran, P. 1985. Reasoning about Three-Dimensional Space. In Proceedings of the 1985 IEEE International Conference on Robotics and Automation, 405-410. Washington D. C.: IEEE Computer Society.

Mackworth, A. K. 1973. Interpreting Pictures of Polyhedral Scenes. *Artificial Intelligence* 4: 121-137.

Mundy, J. L. 1985. Image Understanding Research at General Electric. In Proceedings of the Image Understanding Workshop, 83-88. Washington D. C.: Science Applications International Corporation.

Shafer, S. A. 1983. Shadow Geometry and Occluding Contours of Generalized Cylinders. Ph.D. diss., Dept. of Computer Science, Carnegie-Mellon Univ. Also Technical Report CMU-CS-83-131