To appear in Proceedings of the AAAI Workshop on Spatial Reasoning and Multisensor Fusion, Oct. 1987.

A FRAMEWORK FOR REPRESENTING AND REASONING ABOUT THREE-DIMENSIONAL OBJECTS FOR VISION*

Ellen Lowenfeld Walker
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

Martin Herman
Robot Systems Division
National Bureau of Standards
Gaithersburg, MD 20899

Takeo Kanade
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

The capabilities for representing and reasoning about three-dimensional objects are essential for knowledge-based, 3D photo-interpretation systems that combine domain knowledge with image processing, as demonstrated by such systems as 3D Mosaic and Acronym. Three-dimensional representation of objects is necessary for many additional applications such as robot navigation and 3D change detection. Geometric reasoning is especially important, since 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 2D image and a 3D 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, called the 3D FORM (Frame-based Object Recognition and Modelling) 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. Since the order of processing is controlled largely by accessing objects' slots, the system performs both top-down and bottom-up reasoning, depending on the current available knowledge. The FORM system is being implemented using the CMU-built Framekit tool in Common Lisp [3]. Examples of interpretation using a simple model of building as rectangular prism are presented.

* This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, monitored by the Air Force Avionics Laboratory under Contract F33615-84-K-1520. 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 the Defense Advanced Research Projects Agency or of the U. S. Government.

1. Introduction

We are developing the 3D FORM (Frame-based Object Recognition and Modelling) System, a framework for representing and reasoning about three-dimensional objects. This framework incorporates projections between a 2D image and a 3D scene, representations of shape and surface properties of objects and geometric and topological relationships between objects, and a geometric reasoning capability. Such a representation and reasoning capability is essential for knowledge-based, 3D photo-interpretation systems which combine domain knowledge with image processing, as demonstrated by such systems as 3D Mosaic [4, 5] and ACRONYM [2], and for many applications such as robot navigation, 3D change detection, and simulating the appearance of a scene from arbitrary viewpoints. The 3D 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. The order of processing is controlled largely by accessing objects' slots, so the system performs both top-down and bottom-up reasoning, depending on the current available knowledge.

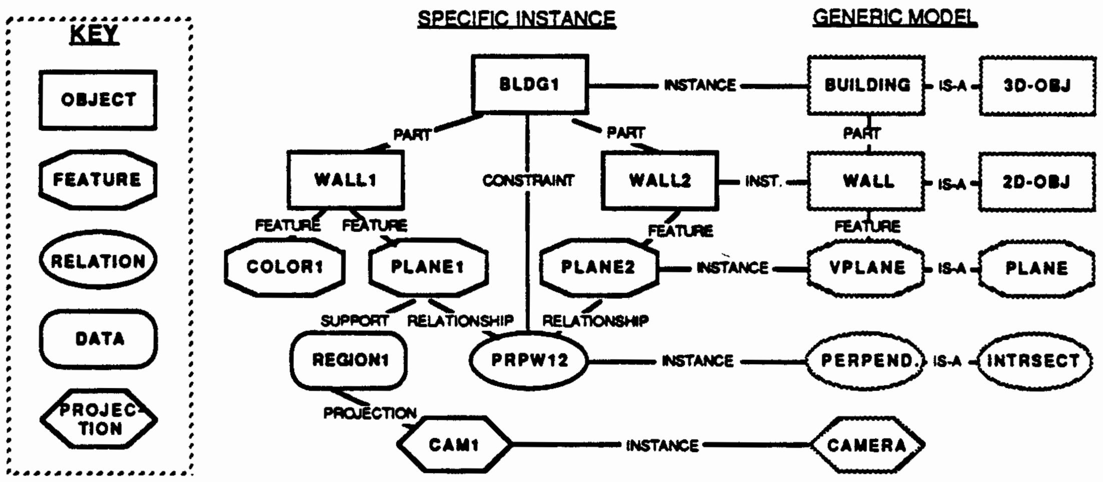

Figure 1: Portion of knowledge base for buildings

The 3D 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. Figure 1 shows a portion of the knowledge that might be represented about a building. A BUILDING is a 3D-OBJECT, with a WALL, which is a 2D-OBJECT as one of its 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 2D and 3D is CAM1.

In the current implementation, only 3D objects, geometric features, and geometric relationships are represented. The next section will discuss some of the issues in geometric reasoning for knowledge-based vision systems and how some existing systems have addressed them. The following sections will discuss the representations of primitive geometric objects, geometric relationships, and composite objects in the 3D FORM system. The final section will discuss the application of the representation to 3D data and present examples.

2. Geometric reasoning in knowledge-based vision systems

Domain knowledge has been used by previous vision systems to compensate for the inadequacies of low level image processing, as well as to generate reasonable assumptions to make it possible to recover 3D shape from 2D data. Shape, one of the most important cues for object recognition, must be included in any domain knowledge representation for a vision system. Therefore, a knowledge based vision system must be able to represent and reason about geometric objects. Geometric reasoning assists in 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 the domain knowledge can be easily modified or extended. Each of the systems described in this section has met some of these goals, but no system has adequately addressed all of them.

The 3D Mosaic system [4, 5] used 3D geometric reasoning in the domain of aerial images of polyhedral buildings to acquire a scene description from images from multiple points of view. Using a polyhedral boundary geometric representation, it hypothesized missing parts of objects in the first view according to a weak model of the urban domain encoded into the program itself. 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 3D Mosaic's domain knowledge. For example, it would be a major programming effort to extend the 3D Mosaic system to make use of the colors of buildings or the textures of their surfaces. Since it was designed as a model acquisition system, the 3D Mosaic system was limited to bottom-up reasoning. When new information invalidated one of the hypotheses generated for missing parts, a complicated network of backpointers was followed to eliminate the effect of the failed hypothesis.

Unlike the 3D Mosaic system, ACRONYM [2] used explicit representation 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. Since 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 more difficult to read, modify, and extend.

Mundy and others [1, 8, 10] are developing a system which 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 a family of interpretations for each object. Inequalities from line labeling [6] are then used to constrain the solutions to these equations. Since the relationships as well as the objects are represented in a concept hierarchy, the geometric reasoning component should be both flexible and extensible. The disadvantage of using algebraic methods is their inefficiency for handling inequalities, an important component of real-world geometric relationships.

Although Hwang's thesis [7] used only two-dimensional 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.

Like ACRONYM and Hwang's system, the 3D FORM system uses frames to represent its objects. Frames are also used by the system 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 parts of their descriptions from other parts. For example, a line has a demon to compute its vector from the known points on the line. Since 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, so the representation is equally amenable to top-down and bottom-up processing. In addition, there is no need for an external ordering mechanism such as a focus of attention. Instead, the dynamically computed COMPLETENESS value for each object is used to select the most complete object to match or relationship to evaluate next.

3. Representing primitive geometric objects

Geometric representation in the 3D 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 may 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 those 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 with 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 may be computed in more than one way from other slots of that object. For example, the vector of a line may be computed either by fitting a line to its points, or by 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.

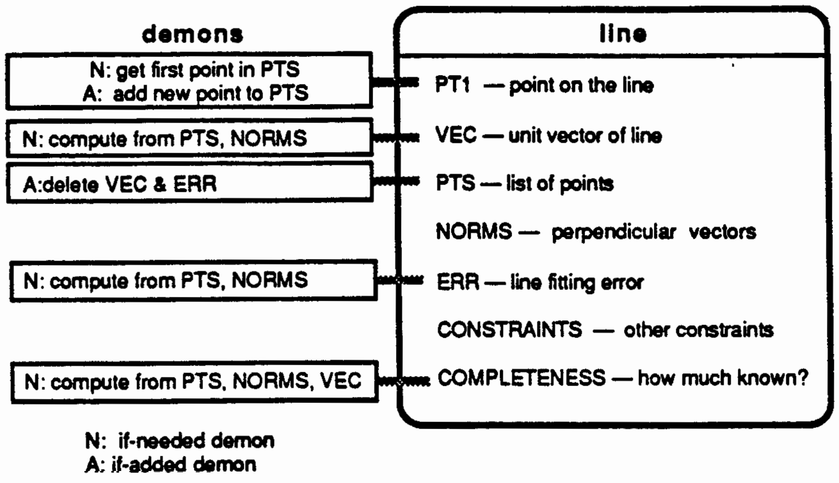
demons
N: get first point in PTS
A: add new point to PTS
N: compute from PTS, NORMS
A:delete VEC & ERR
N: compute from PTS, NORMS
N: compute from PTS, NORMS, VEC
N: if-needed demon
A: if-added demon
line
PT1 — point on the line
VEC — unit vector of line
PTS — list of points
NORMS — perpendicular vectors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

Figure 2: Representation of a line

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 the following slots for this purpose:

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

- 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, and is evaluated when sufficient information is added to the constrained 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 has been satisfied and is removed, and 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 re-evaluated 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 the most complete items are tried first. For example, the completeness of a line is greatest if two or more points are known, but greater if one point and the vector are known than if only one point or the vector is known.

4. Representing primitive geometric relationships

Like geometric objects, primitive geometric relationships are also represented by frames. The system currently considers relationships between pairs of lines, between pairs of planes, between lines and the planes they lie in, and between 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.

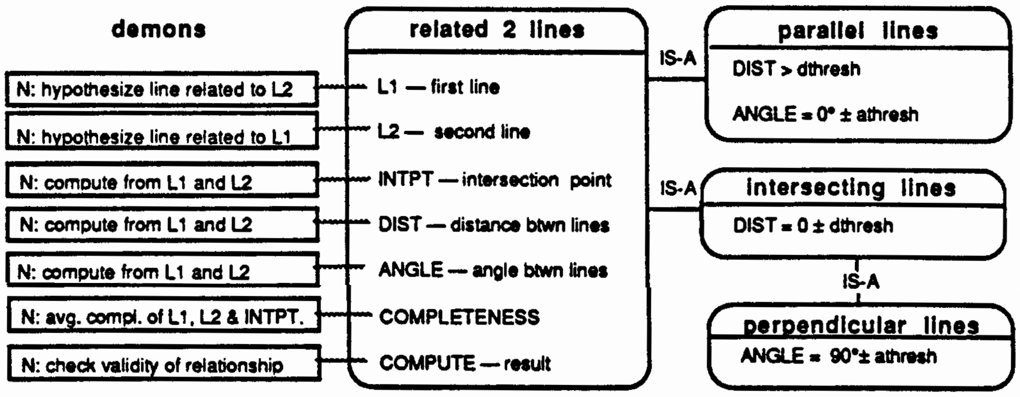
demons
N: hypothesize line related to L2
N: hypothesize line related to L1
N: compute from L1 and L2
N: compute from L1 and L2
N: compute from L1 and L2
N: avg. compl. of L1, L2 & INTPT.
N: check validity of relationship
related 2 lines
L1 — first line
L2 — second line
INTPT — intersection point
DIST — 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 2 lines

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

5. 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 wall. Its parts are two vertical edges, two horizontal edges, and four vertices, and its geometric feature is a vertical plane. Among the constraints of the wall are a **perpendicular-planes** relationship between its geometric feature and the ground and **perpendicular-lines** relationships between the top edge and each of the two vertical edges.

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 may be derived.

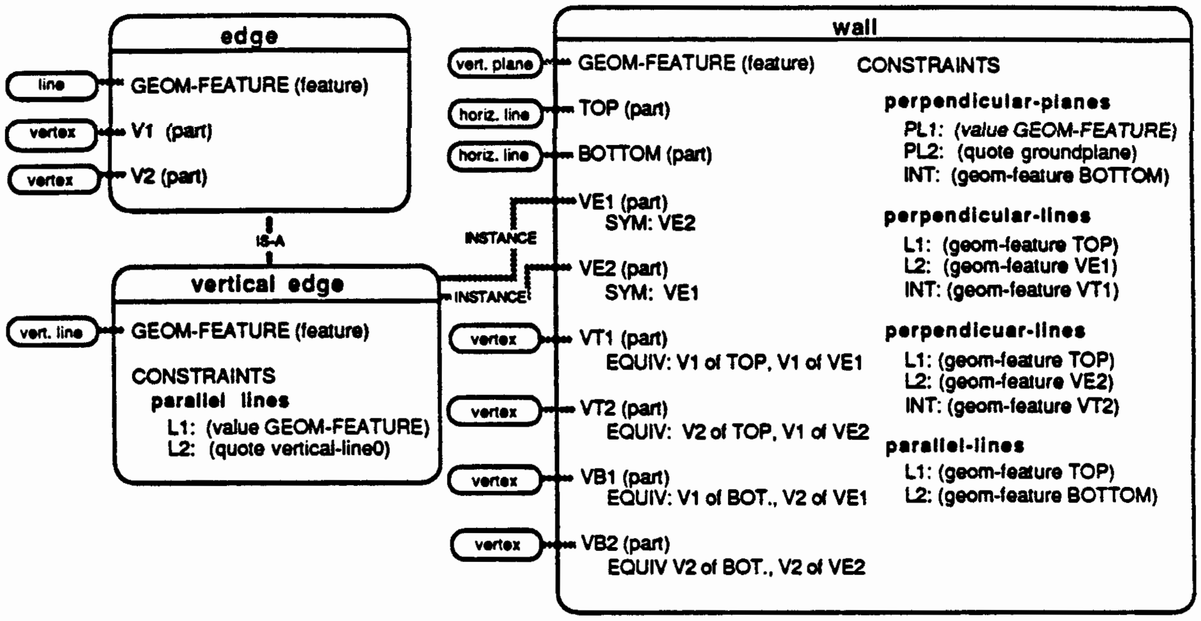

Figure 4: Representations of wall, vertical edge, and edge

5.1. 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 to instantiate new hypotheses for features. In Figure 4, 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.

5.2. 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 parts VE1 and VE2 of a wall are instances of vertical edges. The parts of an object participate in the object's constraint relationships, and objects are matched by recursively matching their parts.

Since an intersection relationship between two objects also refers to the object of intersection, the object of intersection is part of the parent object as well as both of the intersecting parts. For example, the vertex at the intersection of the top edge and one vertical edge of a wall is part of the wall, and also part of the top edge and the vertical edge. To enforce consistency between such equivalent parts, the EQUIV facet of each part of a parent object contains pointers to equivalent parts of the child objects, and demons check all equivalent parts whenever a part slot is accessed. If the values of the equivalent parts cannot be reconciled, an error is signalled to the process that accessed the slot. The EQUIV facet is defined for each of the vertices of the wall frame shown in Figure 4.

In some objects, there are pairs of parts that have exactly the same INSTANCE values and relationships, such as the vertical edges VE1 and VE2 of the wall in Figure 4. Each is perpendicular to the top and bottom edges and parallel to the other vertical edge. These slots are called *symmetric* , and are identified by their SYM facets. Until feature knowledge is added by filling in one of these slots, any data object that matches one will match both and could be assigned to either one. Therefore, matching must take into consideration all possible combinations of symmetric slots.

5.3. 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 4 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 four ways:

(**value** *slot*):	Use the value of *slot* in the current frame.
(**geom-feature** *slot*):	Use the GEOM-FEATURE of the value of *slot* in the current frame.
(**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.
expression:	Evaluate the expression, usually a prototype frame.

Thus, the first constraint of the wall in Figure 4 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.

The evaluation function for an object computes a conjunction of the relationships specified by its constraints. A side effect of evaluating an object's relationships is to generate hypotheses for missing parts of the objects and fill in partially specified objects when possible.

6. Applying the representation to 3D wire frame data

A model of a particular domain is created by defining the generic objects found in that domain. In addition, features and relationships between the objects are defined using the facilities described in Sections 3-5. The 3D 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 best advantage of the available data, controlled by the procedural component of the knowledge representation. Given a simple domain model of rectangular prism buildings, the 3D FORM system interprets a set of 3D edges and vertices (such as the wire frames produced by the stereo and monocular components of the 3D 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 2D and 3D 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 fill in its slots. Once all input features have been 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 3D data may be used to verify these hypotheses.

6.1. 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 more complex objects, and the relationships between them are determined. The dual space [9] 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 to determine parallel relationships between faces. 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 whenever the relationship itself is computed.

6.2. 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 more 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
3. Add new parts to the object and/or relationships between its 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 type, the object may be specialized. Thus, in Figure 4, 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 vertical face to a wall, the slots TOP, BOTTOM, VE1 and VE2 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:

- For each part, which slots have the right INSTANCE value?
- 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 5b to a wall, the local matches for EDGE133 are VE1 and VE2 (see Figure 4), since EDGE133 is a vertical edge. Although EDGE134 (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. Since EDGE134 lies above the ground, its only possible local match is TOP.

If there is at least one possible match 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 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 roof, after local matching, any of the edges of 2D-OBJECT158 can match any of the edge slots of the roof. However, once EDGE134 is assigned to the roof's edge 4, EDGE132, which is perpendicular to it, can no longer be assigned to the roof's edge 2, which is parallel to edge 4. 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 all slots were not filled, the specialization fails. Otherwise, hypotheses for missing parts of the object may 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.

6.3. Controlling the matching process

Since matching is expensive, it is advantageous to limit the number of pairs of objects to be matched. One way this is done by the 3D FORM system is to specialize each data object as much as possible before any matching is attempted. Since 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 pairs of objects 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 making sure that the most likely matches are tried first, and if a match eventually fails, it fails as early as possible, cutting off the recursion tree near the top. Since empty objects match anything, the more complete an object is, the less likely it is to match a given object. Therefore, whenever there is a list of possible matches to be tried, they 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, then choosing the object with the fewest possibilities, and finally choosing the possibility with the greatest average completeness.

6.4. Examples: from 3D wire frames to complete 3D objects

This section describes experiments in which 3D wire frames, generated from image edges by hand, were read into the frame database, specialized, and evaluated, generating hypotheses for missing edges. In the first example, a new edge was then entered manually, and the system matched it to one of the hypothesized edges of the object. The initial wire frame for the first experiment consisted of four edges, three horizontal and one vertical (see Figure 5a). During the initial processing, the three horizontal edges were combined into one face (2D-OBJECT158), and a second face (2D-OBJECT161) was created from the intersection of the vertical edge with one of the horizontal edges. Since the faces intersected at an edge, a 3D-OBJECT was created with both faces as its parts.

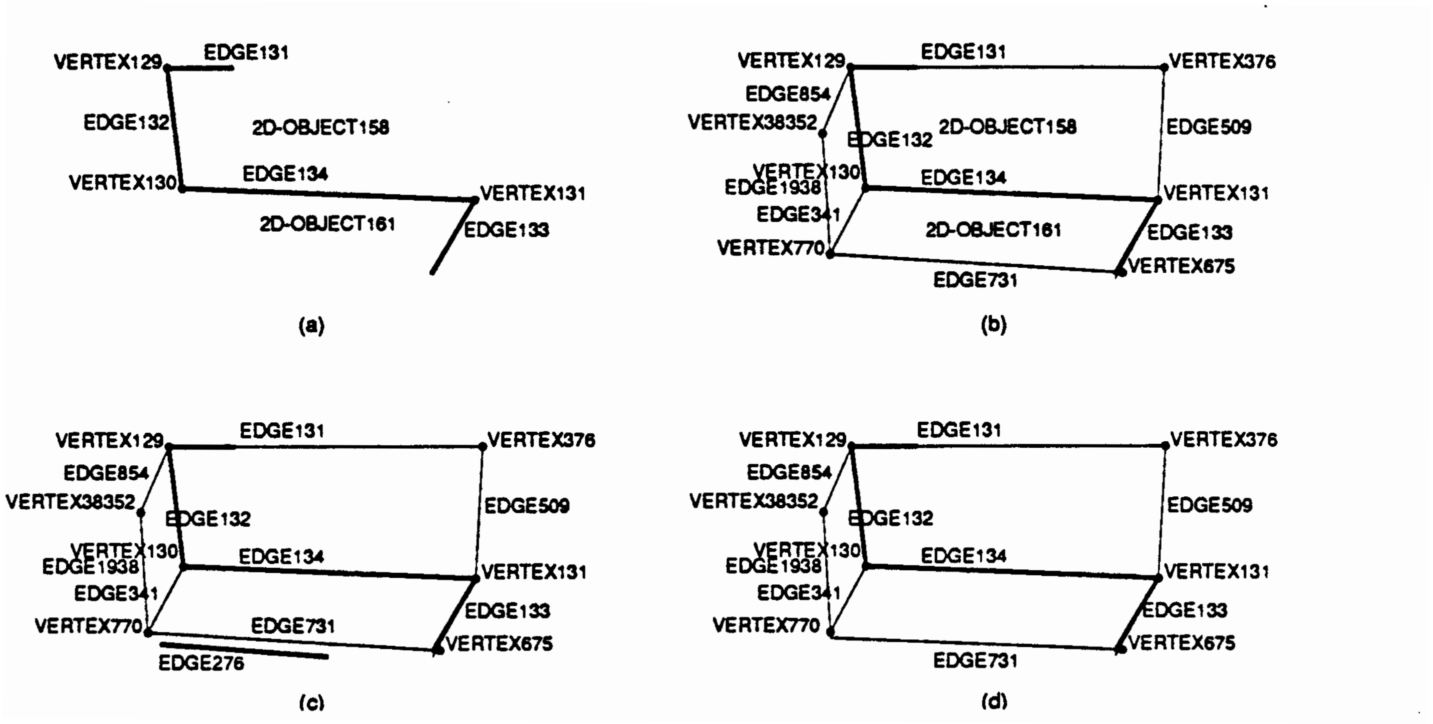

Figure 5: (a) Initial wire frame; (b) visible faces of completed building; (c) completed building with new edge; (d) revised building after merging new edge

In the specialization process, the edges were divided into horizontal and vertical edges, and 2D-OBJECT161 was found to be a wall, since it was a vertical face. Since 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 Section 6.2. The final result of specialization was a building with 2D-OBJECT158 as its roof and 2D-OBJECT161 as its 4th wall.

Next, the building was evaluated, providing hypotheses for the missing slots. Figure 5b shows the visible faces of the completed building. Notice that an extension to the input edge EDGE131 was hypothesized to complete the rectangular roof.

Finally, a new 3D edge was entered and matched to the existing building hypothesis. The algorithm used was to take the new data, specialize it as much as possible, and attempt to match the top-level object to all other instances of the same object until a match was found. The new edge, EDGE276, was found to match the hypothesized edge, EDGE731. Figure 5c shows the building and the new edge, and Figure 5d shows the result of merging the new edge with the old using this algorithm.

The second experiment evaluated a more realistic set of wire frames, finding three buildings in the image, and rejecting one object because it did not fit any models. Figure 6 shows the results of evaluating these wire frames. The bold lines are the initial wireframes, and the remaining lines were hypothesized when the buildings were evaluated. Objects 1 and 6 were rejected as buildings, since each had a non-perpendicular vertex. Objects 3 and 4 were considered to have too little information with only three edges each. The remaining objects were successfully completed.

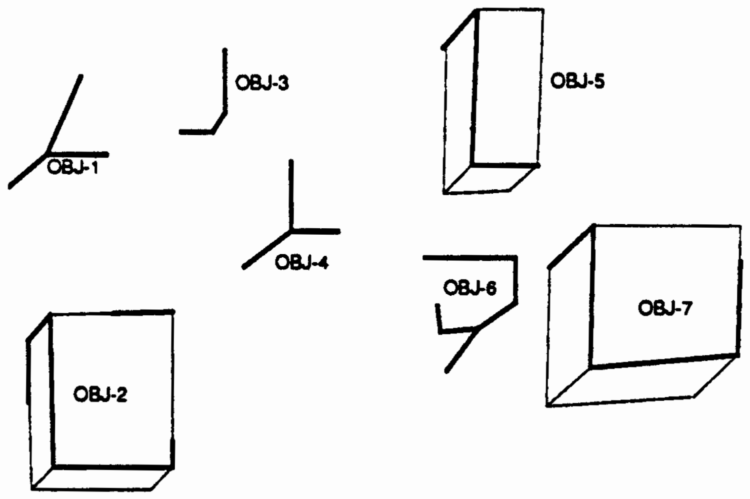

Figure 6: Wire frames and hypotheses for a group of buildings

References

[1] Barry, M., Cyrluk, D. Kapur, D., and Mundy, J. A multi-level geometric reasoning system for vision. In Kapur, D. and Mundy, J. L. (ed.), *Proceedings of a Workshop on Geometric Reasoning.* Keble College - Oxford University, June, 1986. To appear as a special issue of Artificial Intelligence.

[2] Brooks, R. A. Symbolic reasoning among 3-D models and 2-D images. *Artificial Intelligence* 17:285-348, 1981. Special volume on computer vision.

[3] Carbonell, J. G. and Joseph, R. *The FrameKit+ reference manual.* 1986. CMU Computer Science Department internal paper.

[4] Herman, M. Kanade, T. and Kuroe, S. Incremental acquisition of a three-dimensional scene model from images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6(3):331-340, 1984.

[5] Herman, M. and Kanade, T. Incremental reconstruction of 3-D scenes from multiple, complex images. *Artificial Intelligence* 30:289-341, 1986.

[6] Huffman, D. A. Impossible objects as nonsense sentences. *Machine Intelligence*. Elsevier, New York, 1971.

[7] Hwang, V. S. S. *Evidence accumulation for spatial reasoning in aerial image understanding.* PhD thesis, University of Maryland, November, 1984.

[8] Kapur, D., Mundy, J., Musser, D, and Narendran, P. Reasoning about three dimensional space. In *1985 IEEE International Conference on Robotics and Automation.* IEEE, St. Louis, Missouri, March, 1985.

[9] Mackworth, A. K. Interpreting pictures of polyhedral scenes. *Artificial Intelligence* 4:121-137, 1973.

[10] Mundy, J. L. Image understanding research at General Electric. In *Proceedings of Image Understanding Workshop*, pages 83-88. 1985.