# An Approach to Geometric Reasoning in Robotics

**ROBERT F. WOODBURY**
**IRVING J. OPPENHEIM**
Carnegie-Mellon University

**Anticipating the use of knowledge-based expert systems for robot task planning, a geometric reasoner is envisioned for the representation and manipulation of geometric information in a manner uniform with other knowledge. The general requirements for such a reasoner are outlined and a proposed architecture is described, built upon four concepts: a class structure of spatial sets, features as subsets of those spatial sets, geometric abstractions as partial representations of features, and constraints as a mechanism for reasoning. An example implementation is built in an object-oriented programming environment and uses an underlying solids modeler, transparent to the end user.**

Manuscript received April 13, 1987; revised November 4, 1987.

IEEE Log No. 23001

This paper was presented at the Workshop on Space Telerobotics, Jet Propulsion Laboratories, Pasadena, Calif., Jan. 1987.

This work was supported by the Electric Power Research Institute under Contract RP2515-1.

Author's address: Dep't. of Architecture (R.F. Woodbury and I.J. Oppenheim) and Dep't. of Civil Engineering (I.J. Oppenheim), Carnegie-Mellon University, Pittsburgh, PA 15213.



## I. GEOMETRIC KNOWLEDGE AND ROBOT TASK PLANNING

Cognition in robotics will be required when the work environment is uncontrolled, when contingencies are prevalent, or when task complexity is large. Major missions such as space telerobotics, autonomous vehicles, and nuclear plant operations are lead application domains for intelligent robotics. One particularly distinctive feature of cognitive robotics emphasized here is the role of geometric knowledge (its representation and manipulation) and its use in task planning. Consider the following steps in problem solving for a robotic manipulation problem: identification of the environemnt and objects within it, identification of physical relationships between objects, generation of plans to accomplish robotic task objectives; decisions at different levels of abstraction, and execution of any manipulator motion.

Cognition on geometric objects is reflected in every step. This article describes an approach which would permit computer-based reasoning on geometry, utilizing constructs that human designers and users of robotic systems can employ. The particular objective is the provision of an environment in which future knowledge-based expert systems (KBES) can be implemented treating geometric knowledge with powerful high-level concepts, and doing so uniformly with other knowledge in the KBES.

Geometric reasoning spans issues in the representation and manipulation of models describing geometric objects. It is of general applicability over the entire range of disciplines which address real-world physical objects. While differences in the information requirements of various disciplines exist, the commonalities with respect to geometric information are marked. This approach takes as its point of departure these commonalities and seeks an architecture for geometric reasoning. The research methodology has two related aspects. The first is a search for concepts around which an architecture may be organized, and the second is the development of a prototype system as a medium for exploration and as a demonstration of concept.

## II. BACKGROUND

The origins of geometric reasoning for robot task planning or for related demands are found in a number of disciplines. Work that may be directly labelled as geometric reasoning comprises a particular, limited literature and is briefly reviewed here. Kuipers [1] created a theoretical model of human spatial cognition using concepts of representation developed by Lynch [2]. These concepts are related together in a network, through which propagation is used to perform inference; an implicit hierarchical organization is provided by using containment relations on spatial entities. Brooks [3] developed a constraint-based geometric reasoner as part

of the ACRONYM vision system. The reasoner maintains a class hierarchy (Brooks names it a restriction graph) of geometric representations based upon generalized cylinders. The parameters which determine a generalized cylinder are restricted through the use of algebraic constraints. Geometric reasoning is accomplished by algebraic and numerical methods to determine bounds on satisfiability of constraint sets and extremum of objective functions. Davis [4] extended the procedure of metric spatial inference, a technique to make inferences about the relative locations of objects based on simple descriptions, earlier described by McDermott [5]. Constraints on the relative locations of coordinate systems are used to define fuzzy maps which capture the constraint information in a form amenable to computation of queries. Davis [6] then expanded on this earlier work to develop a theory of cognitive mapping which particularly addressed issues of representation, retrieval, and assimilation. The seminal contribution of this work is the formalization of a means of building a map incrementally and of performing inferences on incomplete maps. As an example of another approach, Akiner [7] built a geometric reasoner based on resolution theorem proving in the domain of right rectangular orthogonally oriented cuboids. Geometric objects are expressed as assertions in Prolog and reasoning is accomplished by application of the backtracking search in Prolog using a set of rules about geometry. The concept of features is exemplified by Dixon [8, 9], using that concept to build various specialized representations for design problems; features are groupings of boundary elements of a solid that provide units of concept to application programs. Wing and Arbab [10] outlined a deductive geometric reasoning system. The main contribution of their proposal is recognition of the need for a clear language for geometric reasoning. Shape grammars [11, 12] provide means to describe a class of geometric objects as a set of shape rules, symbols, shapes, and an initial shape. Together, these four constructs define a language of shapes. Shape grammars have a natural relationship to rule-based systems. With the addition of a control structure a shape grammar becomes a production system on a special representation. Kapur and Mundy [13] provide an overview of a recently proposed method for the algebraic proof of geometry theorems [14]. The method represents both hypothesis and conjectures as polynomials with integer coefficients. It attempts to show that the common zeros of the hypotheses are also zeros of the conjecture.

It is appropriate to mention the background literature found on constraints as the basis for computations to maintain constant certain characteristics of an object under modification. Constraints are intimately related to dimensioning, tolerancing, and variational geometry, and the literature is large and growing. Sutherland [15] created SKETCHPAD, recognized as the seminal work in the area. The contributions of SKETCHPAD include propagation (named the one pass method by Sutherland), numerical solution through relaxation, and a method (pins) for linking constrained objects. Steele and Sussman [16] present a language for the representation of almost hierarchical constraint networks, a method of explanation of constraint calculations, and a simple solution technique named local propagation. They also discuss algebraic transformations of constraint networks. In his dissertation Steele [17] examines the implementation of constraint systems at considerable depth. Borning [18] developed a programming language dominated by constraints. His contributions include a strong use of the object-oriented programming paradigm, a local and fast propagation algorithm, and the use of planning for propagation. Light and Gossard [19], demonstrate variational geometry, an instance of the relaxation algorithm used to dynamically manipulate parametric primitives. Gosling [20] contributed several new techniques for constraint satisfaction. These include: the use of breadth-first search to plan propagation, the use of automatic transformation of constraint networks, and a fast graph isomorphism algorithm to make frequent use of transformation feasible.

It is also appropriate to cite examples from the broader field of artificial intelligence which illustrate key concepts of search, hierarchical knowledge representation, inheritance, theorem proving, and deduction within the approximate context of geometric reasoning. Specific results in robot task planning [21, 22], geologic map inerpretation [23] and real-world simulation [24] provide useful precedents. STRIPS [21] demonstrated robot task planning in a circumscribed domain, while Fahlman [22] developed a planning system which was highly dependent on a powerful underlying model. The resulting simplification of the actual planner demonstrates the effectiveness of isolating geometric issues. In two papers [25, 26] researchers at Edinburgh present a comprehensive capability for the representation and computation of high-level spatial relationships between objects. The primary contribution is in the realm of the algebraic solution of spatial constraints. Agin [27] presents a formalism for a hierarchical modeler based on spatial relationships between generalized cylinders. Assemblies are defined hierarchically using any of a number of ways of specifying attachment between primitive objects, represented as generalized cylinders. Simmons [23] used a diagramming scheme separate from the rest of his program to compute adjacencies, positions, and orientation of parts of geologic formations. Klahr [24] discussed various approaches to geometric relationships in the development of an object-oriented battle simulator. His conclusions concerning the use of so-called auxiliary objects strongly support an independent treatment of geometric information. Numerous other applications exist for robot task planning; one example is a domain model for robot task planning in a nuclear power plant environment, implemented in an object-oriented programming environment [28].

Finally, a major element of the authors' approach is found in geometric modeling itself, which accompanied

early applications of computers to design and manufacturing [29, 30]. Geometric modeling is concerned with the representation and manipulation of subsets of three-dimensional Euclidean space and with the construction of computer systems to support these tasks. Several significant and distinct methods of representation have derived from geometric modeling. These methods are: pure primitive instancing, spatial occupancy enumeration, tree decomposition, sweep representation, cell decomposition, constructive solid geometry, and boundary representation. Of these, tree decomposition, constructive solid geometry, and boundary representation are of most significance to this work and in recent years a great number of results have emerged. Additionally there is the topic of computational geometry which according to Preparata and Shamos [31] is an attempt to "reshape—whenever necessary—the classical discipline (of geometry) into its computational incarnation." This field is concerned with the design and analysis of problems in the construction and computation of properties of geometric objects. Preparata classifies problems in computational geometry into five categories, each describing problems in one of the following areas: convex hull, intersection, geometric search, proximity, and geometric optimization. The results of such an approach may provide useful algorithms for these problems.

## III. REQUIREMENTS FOR A GEOMETRIC REASONING SYSTEM

In this approach an architecture for geometric reasoning was defined by postulating and examining a set of requirements. The system for geometric reasoning must meet the following criteria.

1) It must represent a wide domain of geometric entities. A geometric entity is a subset of three-dimensional Euclidean space ($R^3$). Combinations of restrictions based upon finite describability, compactness, and regularity can be used to specify a large and useful set of classes of geometric entities.

2) It must represent and compute on a wide domain of spatial relationships. An object in space is located with reference to some other object or to some coordinate frame. Such a reference establishes a relation between the located object and its referent. These relations are the basic means of composition of assemblies.

3) The system must support a wide variety of useful queries on both single geometric entities and on assemblies. Queries take as input a description of subsets of $R^3$ and produce as results another subset of $R^3$, a vector, a scalar or some textual description. The results of queries may be considered as partial models of the original subset of $R^3$.

4) It must support frame-based representation, especially the definition of classes of geometric objects and the use of inheritance as a means of definition and reasoning. Classes are the basis of one of the powerful means of frame-based reasoning; it must be possible to define prototypical descriptions (classes) based upon existing descriptions.

5) The system must provide a means to generate complete models of objects from incomplete models. An example is the creation of a routing for piping (a complete model), given only the topological interconnections of the piping (an incomplete model). Generation is an information adding operation on a model; many possible complete models may exist for a given incomplete model.[1]

6) It must provide a wide variety of modification techniques on geometric data. A modification of geometric data is an operator which changes some aspect of the data while leaving other portions unaffected. Both object definition and object altering operators are modification techniques. A prime consideration in modification operators is the maintenance of consistency.

7) The system must support representation at multiple levels of abstraction. The modeling abilities of a geometric reasoner must be able to represent what is known about an object without having to completely specify it. It must be possible to incrementally add information to a model; each incremental addition creating a less abstract, more complete model.

8) It must make its data directly accessible to reasoning mechanisms within a knowledge-based system. Such reasoning mechanisms include those of frame-based systems (defaults, procedural attachments and inheritance) as well as others associated with knowledge-based programming (constraint satisfaction, search). This implies an ability to decompose representations down to an atomic level.

9) It must be extensible to include new algorithms for geometric computation, without structural change to the system. Both the number of applications which use geometry and the number of algorithms on geometric represenations are growing explosively. A geometric reasoner must be able to gracefully accept the addition of new algorithms and representations if it is to continue to be useful.

A tenth requirement that a geometric reasoner must be numerically robust within its domain was considered. Geometric modelers are generally plagued with problems of numerical robustness as geometric properties approach the hardware precision. Recent research [32] indicates that solutions to this problem lie at a level significantly below that addressed here, so numeric robustness was excluded as a requirement on this effort.

[1]Generation is the reverse of querying. Every type of query poses a problem in generation. For example, the query center-line on a straight homogeneous generalized cylinder produces a line segment. This line segment is a partial description of any member of an infinite set of possible cylinders. A generation algorithm provides the ability to systematically enumerate this set.

## IV. CONCEPTS FOR THE SYSTEM ARCHITECTURE

The requirements stated in Section III constitute objectives for system functionality, but do not in themselves prescribe means of achieving such goals. We have designed a system archictecture to satisfy those requirements, based upon four component concepts: classes of spatial sets, geometric abstractions, constraints, and features. The following subsections describe each concept. Section V then relates the concepts together to form the overall architecture.

### A. Classes of Spatial Sets

The concept of class is fundamental to frame-based knowledge representation and need only be discussed summarily here. A class is a template, from which individual instances of the class may be created. These instances fill in the template structure defined by the class to describe a particular object, be it a physical object, a process, an idea and or some other entity. Classes are usually organized into an inheritance hierarchy. This permits the omission of information from a particular class specification; the information is instead acquired from classes superior in the inheritance hierarchy. Inherited information may be either data or functions; the latter are usually called methods.

A spatial set is a region of three-dimensional space. In the boundary-based modeling scheme, it is described by: 1) a topology, a graph representation of the adjacencies between primitive elements, 2) a geometry, the numeric information which imbeds the topology in space, and 3) a location, describing the location of the model with respect to some datum.[2]

Classes support the incremental specification of these properties by specialization. For example, consider the class specification of a cuboid as shown in Fig. 1. Class cuboid inherits the templates for slots body and geom from class polyhedron which in turn inherits the body slot from class topology. It incrementally adds a definition, used by all members of class cuboid, for the topology of the cuboid. Specific instances of the cuboid will add a geometry definition to create a complete model of a cuboid polyhedron. Note that the class definition for cuboid actually fills in the body slot. This slot value is essential to the definition of the class, and is not a default. It provides a template of the topology of a cube; this template may see multiple uses to define specific cubes. Other characteristics of a model may be defined in classes and inherited. They are addressed in Section V.

Classes of spatial sets address four of the requirements on geometric reasoning outlined in Section III: 1) They facilitate the definition and specification of a wide domain of spatial sets (requirement 1); 2) they provide a framework for the integration of queries on geometric entities

**Class definition for *topology***

```
(ndefstruct (topology (:class class) (:conc-name nil))
            (body nil))
```

**Class definition for *polyhedron***

```
(ndefstruct (polyhedron (:include topology)
                        (:class class) (:conc-name nil))
            (geom nil))
```

**Class description for *cuboid***

```
(ndefstruct (cuboid (:include polyhedron)
                    (:class class) (:conc-name nil))
            (body #.(extrusion 4 '*tope* '*bote*)
                  :allocation :class)
            (bottom-edge #.*bote* :allocation :class)
            (top-edge #.*tope* :allocation :class)
            (num-sides 4 :allocation :class))
```

**Call to function to create a cuboid instance**

```
(make-cuboid :geom (cuboid (body *general-cuboid*) 10.0 20.0 30.0))
```

**Instance of a cuboid**

```
CUBOID CLASS
        CLASS VARIABLES:
                BODY            297792
                NUM-SIDES       4
                BOTTOM-EDGE     296204
                TOP-EDGE        296576
        INSTANCE VARIABLES:
                GEOM            298408
```

Fig. 1. Class hierarchy for class cuboid.

(requirement 3); 3) they directly support frame-based representation and reasoning (requirement 4); and 4) they establish a clear framework for extensibility (requirement 9).

A wide domain of spatial sets is naturally supported by the class concept. Structuring geometry into classes allows the transparent definition of useful representations and algorithms for different types of objects. Each class of objects, for example, polygons, may have a specialized data structure or structures and a set of algorithms for computing on those structures.

Classes of spatial sets provide the appropriate places for the integration of queries on specific classes. These take the form of methods, defined on a specific class. For example, a large number of different functions is required to query a boundary representation topology. These are defined as methods which can operate only on instances of class topology, or of classes which inherit class topology. Similarly methods which require geometric definitions are defined on class polyhedron.

One of the characteristics of frame-like descriptions of objects is their reduction of a representation to a small number of apparently independent pieces of information (slots). This presentation of an object description allows the use of local reasoning algorithms (inheritance, defaults, procedural attachment, constraints) that base their specific actions on current values in slots, seemingly independent of other characteristics; further discussion of this reasoning occurs in Section IV D. This feature of frames is fundamental, it is central to the capability of frames for prototypical representation. When frame-like descriptions are used to represent geometry, it is very convenient to employ a geometric representation scheme which is also based on prototypes.[3] Use of a scheme less dependent upon prototypes, such as boundary representation, implies a mismatch between frames and

[2]We use the usual geometric modeling terms for description here. It is understood that the terms topology and geometry are not precise.

[3]Generalized cones are an example of such a scheme.

the geometry scheme. The use of the class concept presented here attempts to resolve this dilemma. Essentially, classes of spatial sets denote rather than represent. They are a layer of abstraction upon a geometry representation proper rather than a complete geometry representation in themselves. Consider the description of class cuboid in Fig. 1. An instance of this class contains five slots, all but the GEOM slot actually allocated in the class. The four slots which have as their value 6-digit numbers contain references to specific geometric models. The body slot refers to a topology representation for the entire cuboid, the top-edge and bottom-edge slots to specific edges in the cube and the geom slot to a representation for the numeric information which imbeds the topology representation in space.[4] Any operations on these models are performed by the algorithms which operate on the model, not the frame representation. This relation between frames and underlying models is shown in Fig. 2.

[4]The numbers are actually the integer renditions of pointers. Geometry models are created and managed in all these examples by a solids modeler, VEGA, which is written in the C Language. Pointers to VEGA models are used to designate geometric representations.

This frame-like representation of geometry is still not sufficient for use by most reasoning algorithms. The values in the slot are not atomic. Further selective reduction to more atomic descriptions is required. Concepts for these reductions are discussed in Sections IV B and IV C.

A final use of classes of spatial sets is that of extensibility. The support provided to incremental system extension by inheritance is well documented; the use of classes presented here retains all of these advantages. Essentially, program development using classes proceeds by the incremental addition of restrictions on class descriptions and the definition of new methods. All of the superclass slots and methods not changed by a specific class are available, untouched, for ready use.

## B. Features

The origin of the term features arises from its general use in computer-aided design as a means of specification of portions of a model that are of interest. In this work feature is simiarly, but more precisely used, to define a subset of the set of primitive objects which compose a

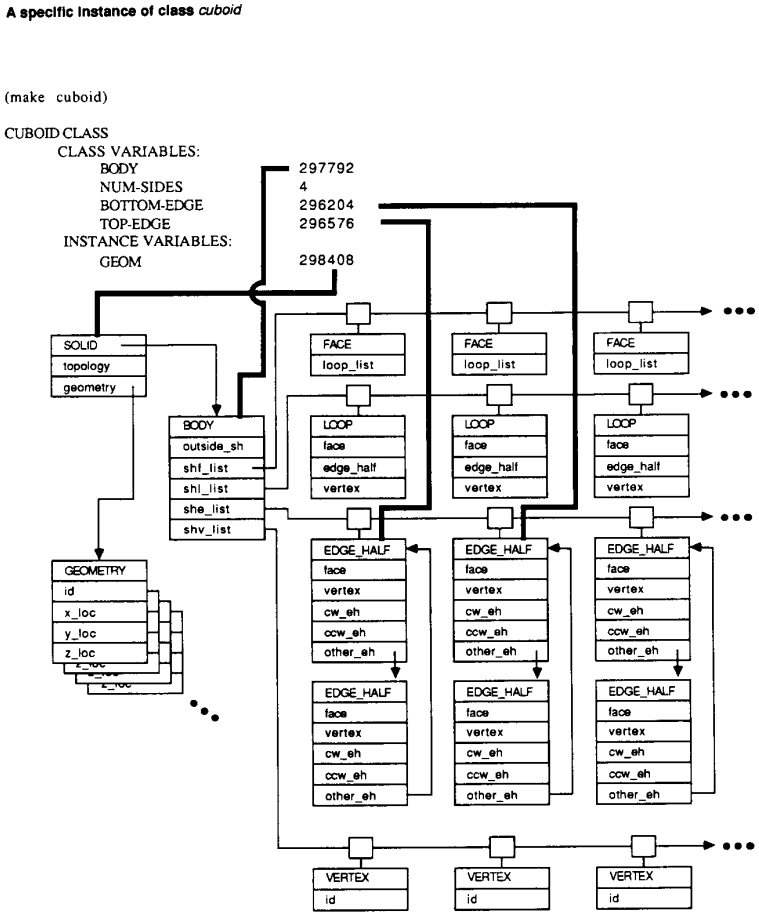


Fig. 2. Slot values in classes denote rather than represent.

geometric representation. Consider any modeling scheme. In all schemes but the most simple, objects are composed of other objects. For example, in a boundary-based solids modeling scheme, the primitive parts of a solid object are vertices, edges, loops, faces and shells. As another example, in a CSG-based solids modeling scheme the primitives are simple volumetric objects, at specified locations in space. A feature of such an object can be any subset of these primitives. Features are used to identify objects of interest, be they assemblies, individual solids, collections of faces, individual faces, or other parts of a geometric model. In the following discussion of features, the boundary-based modeling scheme is used as the underlying representation upon which features are defined. If another scheme was used, the types of entities the features would refer to would be different, but the fundamental structure of the feature concept would remain invariant.

Features are a data structuring concept which lies above a modeling scheme. Fig. 3 diagrams this relationship for the simple example of a cube. The modeling scheme for the cube completely describes its geometry, including both the topological information which describes connectedness between faces, edges and vertices and the numeric or geometric information which describes the embedding of the topology in three-dimensional space. The feature top-face identifies a portion of the cube model, an actual face description. Similarly, the feature cube identifies the description of the entire cube.

Features are hierarchical, in that a given feature may contain subfeatures. For example, in Fig. 3 the feature cube has as subfeatures the features top-face, bottom-face and side-face. At the very highest level, all features currently active are considered subfeatures of a root feature.

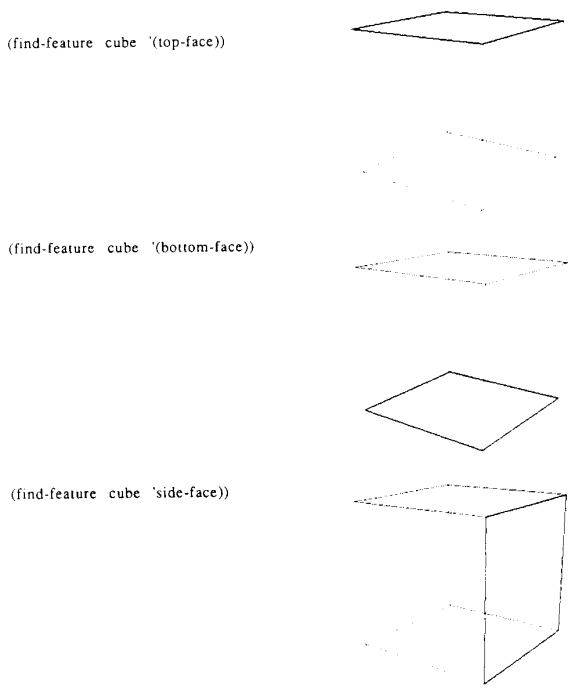


Fig. 3. Features access geometric models.

In their least restricted form, feature hierarchies place almost no restrictions on the contents of subfeatures of a feature. The single exception, is a restriction to non-self-referentiality, that is, feature hierarchies must be acyclic. Such an unrestricted data structure is useful in some contexts, for example for representing as a single feature all faces in an assembly which compose holes for fasteners. The addition of further restrictions on the allowable contents of subfeatures adds significant representational power. For example, consider a feature which is known to denote a single polyhedron. Restricting the allowable subfeatures of such a feature to only those faces, loops, edges, and vertices which compose the polyhedron transforms features in this application into something very like a part-whole hierarchy on solids models. As another example, take a feature which is known to contain either only individual polyhedra or other features like itself. Such a feature is equivalent to the data structure known in CAD as an instance tree.

Feature hierarchies need not exhaustively label every entity in a model. The model is independent of the feature structures which point to it and it alone actually represents the geometry of physical objects. Features merely identify parts of those objects which are of interest to a particular application. This relief from the burden of being an actual representation scheme for geometric modeling allows features to only partially specify the form of an entity, by only specifying those parts of its boundary which are pertinent or known.

As is implicitly shown in Fig. 3, features are given names, by which they are known to external applications. The names provide access to feature structures which identify a particular part. Names for features need be unique only within an individual branch of the feature hierarchy. Later, in Section V, this nonunique naming will be exploited to allow the inheritance of feature information as part of a class description for a geometric object.

Features alone act to meet four of the nine requirements on a geometric reasoning system[5]: 1) By acting as aggregation hierarchies, features add the notion of an assembly to the domain of representation (requirement 1). 2) By acting as a focus of attention mechanism, features support the expression of queries and spatial relationships on geometric objects (requirements 2 and 3). 3) By providing a uniform interface to any part of geometric object, features provide application-oriented access to geometric information (requirement 8).

Aggregation is one of the fundamental mechanisms for abstraction. Informally, an aggregation abstraction

[5]When combined with classes of spatial sets, they provide, in addition, a means of prototypical description. See Section V.

allows the specification of an object by describing the parts of which it is composed. Many data structures commonly used in computer-aided design (CAD) and artificial intelligence (AI) are aggregation hierarchies, for example, instance trees in drawing and solid modeling systems and part-whole hierarchies in their numerous instantiations in AI systems. The use of the term features in the CAD literature in general is that of an aggregation abstraction (e.g., Dixon's design-with-features [9, 8]). The features of this work find a primary use as a means of aggregation abstraction. Consider a window regulator, a simple mechanical device used to raise and lower automobile windows. It is composed of several parts, each of which contains subparts and so on. A feature hierarchy of a regulator is used to identify parts of the regulator that will be used in specific computations. Fig. 4 shows a drawing of a window regulator and an associated feature hierarchy for that regulator.

Two of the properties of features, denotation and nonexhaustiveness extend features beyond most uses of aggregation. Features denote; the resulting relief from the burden of representation allows the selective structuring of information. Features need not exhaustively describe; by omission only those parts of a model of interest need to be structured. Features thus provide a means for the focusing of attention upon specific objects or parts of objects in a model. Consider an example, the use of fasteners through holes to position parts of a window regulator. The relative position of parts of a regulator is determined only by the mating faces and aligned holes of the regulator. All other information about the regulator is superfluous to this task. Features are used to denote just these objects of interest. Fig. 5 shows the features required to express the mating conditions.

The objects seen by a programmer using the geometric reasoning architecture are features and abstractions of those features.[6] This permits the information in a geometric model to be structured with reference to the reasoning methods useful for an application rather than in terms of the modeling scheme itself.

## C. Geometric Abstractions

In the following discussion we use the term geometric abstraction, and when clear in context just the word abstraction, to describe a particular form of abstraction of information. We regret that this conflicts with other established usage of the word abstraction, but prefer this usage to any of the opaque alternatives we considered.

Geometric abstractions represent only a portion of the properties of a spatial set. In this respect they are like features, which also abstract from a spatial set. The

[6]To be discussed in Section IV C.

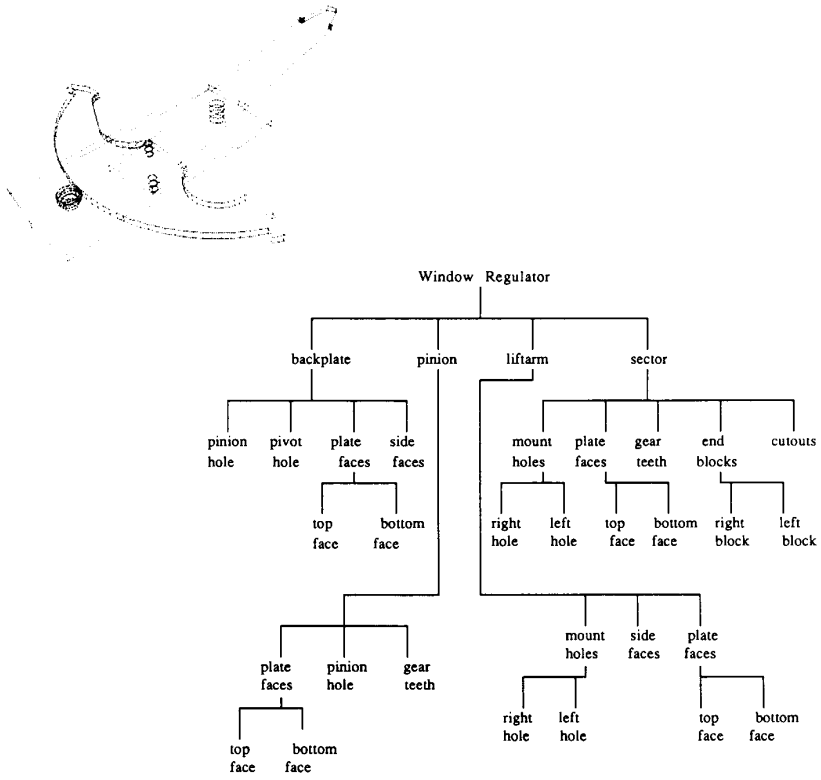


Fig. 4. Aggregation abstraction in window regulator.

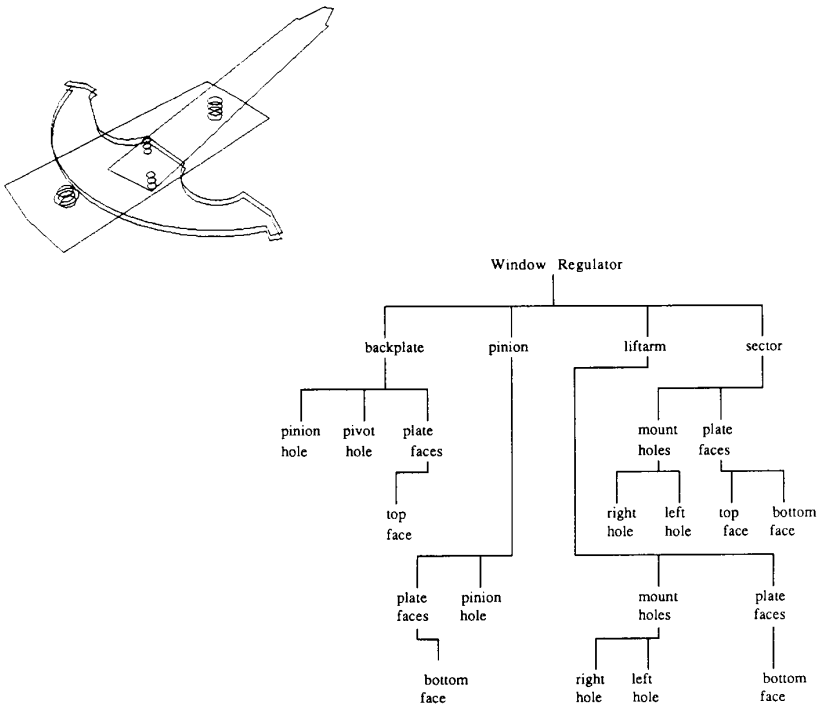


Fig. 5. Use of features to focus attention to particular part of model.

essential difference between the two concepts can be shown by discussion of both in reference to the modeling scheme of the geometric modeler which underlies a geometric reasoner. A modeling scheme is a means, used pervasively in a modeler, of organizing primitive components into a whole. For example, the boundary representation scheme organizes geometric information into related collections of shells, faces, loops, edges and vertices. The components used in a modeling scheme are considered the evaluated primitives of the scheme. The evaluated primitives do the following: 1) They impose an organization and view on the data in the model. For example boundary modelers impose the view that an object is defined by its boundary. The primitives of shell, face, loop, edge, and vertex are those through which an object is conceived. 2) They are the only elements which are immediately available in the modeler. All others must be computed by some algorithm. For example, in Constructive Solid Geometry (CSG) schemes, edges are not primitives. They are computed by a process known as boundary evaluation.

Features denote sets of evaluated primitives. Geometric abstractions compute, from those sets, other representations which express some of the information carried in a set and suppress all else.

Consider Fig. 6, which shows two cylindrical holes through a plate in a boundary representation modeler. Directly accessible from the modeler are the adjacencies of faces and usually the equations of the cylindrical face and the adjacent top and bottom faces. This information may be accessed; anything else must be computed. Of

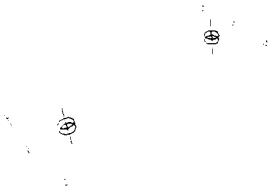

Fig. 6. Geometric abstractions of two holes.

frequent interest for such holes is information about the location of the center-line of the hole. This may be computed as a line segment; the face equation of the hole provides sufficient information to compute the line equation, and the intersection of the resulting line and the top and bottom faces of the hole determine its extent. The result is a line segment, described by six values. This line segment is an abstract representation of the hole; it preserves location, orientation and length, while suppressing diameter and interaction with other geometric elements in the model. It is stored as a geometric abstraction, named center-line of the cylindrical hole.

A geometric abstraction is composed of three parts; some data which constitute the model of the abstraction, a label indicating the semantic meaning of the abstraction, and a referent which points to the object which the abstraction partially describes. For example, the center-line abstraction described above requires a line segment, a label which indicates what the line segment actually represents, and a pointer back to the specific hole which it partly represents.

The concept of geometric abstractions contributes to six of the nine requirements for geometric reasoning. 1) It

provides a framework for the definition and use of queries on geometric objects (requirement 3). 2) It provides a framework for the formulation of generation problems (requirement 5). 3) It allows the representation of entities within the modeling domain prior to having complete knowledge of their form (requirement 7). 4) It supports the maintenance of consistency of geometric information under modification (requirement 6). 5) It makes geometric information accessible to frame-based reasoning processes (requirement 8). 6) It provides a framework within which to extend a geometric reasoning system (requirement 9).

Geometric abstractions are created by query, generation, or assertion. When created by query, a model which contains sufficient information to compute the abstraction is used as input to a query function. This function returns as a geometric abstraction object, a partial model of the original object. These query functions define a geometric abstraction hierarchy in which the nodes are geometric abstraction classes and the arcs are the query functions. Actual computation of an abstraction instantiates part of this abstraction hierarchy for a particular geometric object. For example, consider the query function for creating a center-line abstraction. This function takes as input a boundary representation of a hole, and produces as output an abstraction for the center-line of that hole. Another level in the hierarchy is produced from the center-line by the query function midpoint. This function takes the abstraction center-line as argument and produces as result the abstraction midpoint of that center-line. The (trivial) abstraction hierarchy which describes these relations is shown in Fig. 7.

This hierarchy reveals a relationship between querying and generation. A query on an object produces an abstraction of that object. If computation were to proceed in the other direction, from the abstraction to the original object, information would have to be added to the model. Generation would occur. For every abstraction producing query on an object, there can be posed a converse

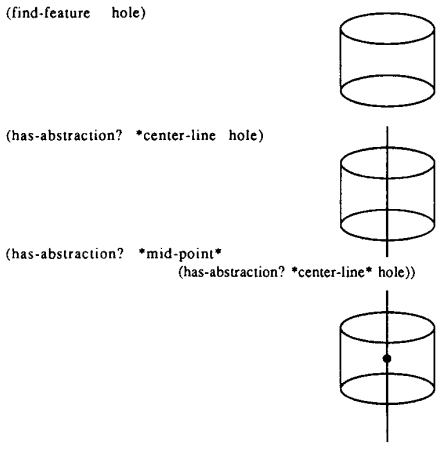


Fig. 7. Simple geometric abstraction hierarchy.

problem in generation. Generation of form from incomplete descriptions is a difficult problem, which admits no simple solutions and which is a very open and current research issue [33]. What is provided by abstraction hierarchies is a framework for stating generation problems and for integrating generation programs in a straightforward manner.

The third method by which abstractions can be generated is assertion. Some agent external to the reasoner asserts the description of a certain abstraction. If this is not inconsistent with information known in the model, it becomes part of the model maintained by the reasoner. Asserting abstractions provides a means to represent an object partially, in a manner uniform with more complete representations.

The maintenance of abstractions under modification to the model is automatic, with the default being removal. If a model of an object is altered then some of the abstractions of the model will become inconsistent unless updated. Given the existence of query algorithms for creating abstractions from higher level representations in the abstraction hierarchy, a default solution for maintaining this consistency is recomputation, i.e., throw away the abstraction and recompute it from the changed data. With certain classes of abstractions it is possible to easily define algorithms which maintain consistency without recourse to such drastic measures. For example, consider once again the center-line abstraction. Movement of the object from which the center-line is defined requires an identical movement by the center-line. Instead of recomputing the center-line, a simple three-dimensional transformation may be applied.

Abstractions are used to build frame-like descriptions of geometric models for specific applications. A frame-based representation of a geometric object used in a particular context or application, would be structured using concepts relevant to that application. For example, a structural design system might specify elements in terms of the planes that they occupy in space. Information of this type is only implicitly stored in solid modeling data structures. Consider a boundary-based representation of solid models. The primitives available are shells, faces, loops, edges, and vertices. Specification of a frame template using these elements directly could not result in a structure reasonable for specific problem domains. General solid modeling data structures do not lend themselves to the application orientation of frames.[7] Abstractions, by selecting and restructuring information implicity held in a geometric model, support the use of this information in an application-oriented manner. They

[7]Boundary, CSG, spatial occupancy enumeration and tree decomposition representations are general schemes. One more specific scheme, that of generalized cones has seen wide use in AI applications. Generalized cones are inherently prototypical; they are defined by values on parameters. The advantages of prototypical specification outweigh the lack of generality of generalized cones for many applications and particularly for research efforts which are focused on applications not representation.

accomplish this by reduction of modeling data to prototypical sets of data. For example, in Fig. 6, the center-line abstraction expresses information about a hole in a model using only six numbers, and a template which structures those six numbers as a line segment. Once these frame-like structures are defined, they can be used in the accustomed computational style of knowledge-based systems.

The concept of geometric abstractions and the associated abstraction hierarchy provides a means to incrementally extend the functionality of a geometric reasoning system. By defining new classes of abstractions and their relation to other classes already in the abstraction hierarchy, functionality can be added to a geometric reasoner in a strictly local fashion.

## D. Constraints

One view of a reasoning mechanism is that it is an algorithm which acts upon a knowledge structure to make explicit some of the knowledge implicit in the structure. Many reasoning mechanisms have been developed in artificial intelligence; those applicable to frames include inheritance reasoning, procedural attachment, defaults and constraints. All of these reasoning mechanisms capitalize on the quasi-independent slots of which frames are composed. The reasoning mechanism (or more properly, the reasoning paradigm) of constraints, is particularly appropriate to geometric reasoning due to its easy accomodation of algebra.[8] Many geometry problems admit to algebraic specification and solution. Much of the research on geometric reasoning has been focused on solutions of algebraic systems [3, 13, 19, 26]. The constraint paradigm is discussed at length in a number of places [15, 18, 16, 20, 34, 35] and will not be reviewed here. The discussion instead focuses on the uses of constraints in the context of this research.

Constraints address several of the requirements discussed in Section III. Specifically, constraints provide: 1) a representation for certain classes of spatial relationships (requirement 2); 2) a means for the maintenance of consistency under modification (requirement 6), and 3) extensibility (requirement 9).

Certain classes of spatial relationships between objects may be conveniently represented through constraints. These include relationships such as parallelism, orthogonality, general angle relationships and adjacency between planes and lines, equality, verticality and horizontality between points, and intersection between lines. Other classes of spatial relationships, for example containment in general polyhedra either do not have convenient constraint formulations or have specific and efficient solution techniques. For some problems, the metaphor of constraint satisfaction is not appropriate. The exhaustive enumeration of layouts based on orthogonal relationships [33] is such a case. Nonetheless, the representation of constraints and particularly of algebraic constraints provides a useful means to express the meaning of many types of spatial relationships.

It is clearly shown in several results in the literature [3, 18, 20, 35, 36] that the actual methods of constraint solution are problem dependent. For example, numerical relaxation, while a very general technique, is usually quite slow. An application which is interactive requires faster methods, even if they are more specialized. Another failure of the relaxation method is its lack of capability to support explanation. Propagation methods are superior here, as an explanation can be constructed in terms of the chain of firings leading to a result. Fortunately, solution techniques for constraints are independent of their high-level specification. Even if widely varying methods of constraint solution are used in different applications the means of expressing constraints described here can be used. The only change required to accomodate different constraint representations is in the methods for instantiation of primitive constraints. These methods write into a data structure the appropriate constraint instantiations.

Constraints can be used to express integrity relations in a geometric model. Once expressed, a general constraint solver is responsible for satisfaction of the relations. As an example, take the plate from Fig. 6. A required distance between the two holes can be expressed as a constraint on the center-lines of the two holes. When one hole is moved the other will have to move so that the distance constraint is maintained.

Constraint representations are easily extensible to include new types of constraints. This is achieved through a set of language constructs or forms which define composite constraint values (the defconval form), new primitive constraints between constraint values (the defcon form) and composite constraints from other constraints (the defmaccon form). These forms automatically generate all necessary datatypes and algorithms for new classes of values and constraints. Consider the construction [20, pp. 17 and 21] of a constraint on a point to be at the midpoint of a line segment. The mathematical conditions are that the coordinate values of the midpoint be the average of the respective coordinate values of the endpoints. This may be expressed as two equations for each of the coordinates of the points. Let $A$ and $C$ be endpoints of the line segment. Let $B$ be the midpoint. Then:

$$\forall\, i \in \{x, y, z\}\ (iC) = (iB) + (iT) \wedge (iB) = (iA) + (iT).$$

$T$ is a variable, local to this expression, which will solve, for each of the respective coordinates, to be half the distance between the two endpoints.

Expressing this midpoint as a constraint requires in addition to primitive constraint-values and a primitive sum constraint upon them, definitions of the complex value point, the constraint sum on points, and the

[8]The usage of constraints in this thesis is that of the SKETCHPAD [15] lineage. The related paradigms of constraints in line labellings and constraint satisfaction in search are not explicitly addressed here.

constraint midpoint. The complex value for a point is called *c*-point-3d and is defined with a defconval form.

```
(defconval c-point-3d :supers-list (point-3d)
                      :alter-list  ((x constraint-value)
                                    (y constraint-value)
                                    (z constraint-value)))
```

The primitive constraint sum between atomic values is called sum-constraint and is defined with a defcon form. This form defines a class for the sum-constraint, a set of methods for its solution by propagation, and a method for instantiation of sum-constraints. The form consists of a declaration of the classes of allowable arguments, a rule which provides a test for constraint satisfaction and a set of clauses for propagation.

```
(defcon sum-constraint: ((result constraint-value)
                         (op1 constraint-value)
                         (op2 constraint-value))
  :rule ((op1 op2 result) (equal result (plus op1 op2)))
  :clauses
    (((result) (op1 op2) (((op1 op2) (plus op1 op2))))
     ((op1) (result op2) (((result op2) (minus result op2))))
     ((op2) (result op1) (((result op1) (minus result op1))))))
```

The compound constraint sum between points is called sum-constraint and is defined with a defmaccon form. The defmaccon form defines a method named instantiate-sum-constraint, which due to the classes of its arguments is distinguished from existing methods of the same name.

```
(defmaccon sum-constraint ((result c-point-3d)
                           (p1 c-point-3d)
                           (p2 c-point-3d)
  :code ((constrain sum-constraint (x result) (x p1) (x p2))
         (constrain sum-constraint (y result) (y p1) (y p2))
         (constrain sum-constraint (z result) (z p1) (z p2))))
```

The constraint midpoint is called midpoint-constraint and is defined with another defmaccon form. Note the use of the temporary variable temp, declared in the locals part of the form to be of the same class as the argument p1. This definition itself can be used, via the constrain macro, as part of the definition of new constraints.

```
(defmaccon midpoint-constraint ((p1 c-point-3d)
                                (mid c-point-3d)
                                (p2 c-point-3d))
          :locals ((temp p1))
          :code ((constrain sum-constraint mid p1 temp)
                 (constrain sum-constraint p2 mid temp)))
```

## V. THE CONCEPTS TOGETHER

### A. The Architecture

The four concepts, classes of spatial sets, features, abstractions, and constraints, interact to enable specific representation and reasoning capabilities. Out of these interactions is developed the overall system architecture. Four pairwise interactions are significant to the architecture as it is discussed here: 1) classes of spatial sets and features, 2) features and abstractions, 3) classes of spatial sets and abstractions, and 4) abstractions and constraints.

In this Section each of the four interactions are reviewed with the aid of diagrams and their composition into a whole is shown. For each interaction, a succinct description is given, followed by an explanation, a list of the useful characteristics added to the reasoner by the interaction, and a diagram.

Spatial sets and features may be combined in a class. Consider Fig. 4 in which a feature breakdown of a window regulator is presented. This breakdown may be part of the class definition of a window regulator. Every window regulator instance that is created from such a class will have the feature structure already defined upon it. Fig. 8 shows how this is achieved using inheritance. Three significant characteristics emerge from this relation.

1) Features remain the fundamental programming interface to the geometric reasoner. An instance of the class of window regulator described above would be a feature. Any queries or operations on that instance will produce either other features or abstractions of features. This simplifies greatly writing the method protocols which act as a programmers interface.

2) Features may be inherited as part of the definition of a class of a geometric object. Classes specify prototypes, and knowledge of the special identity of certain components of a model, i.e., features, is an important part of the knowledge which needs to be inherited.

3) The contents of features can be restricted to be part of the geometric object which is described by a class. Attempts to add a subfeature to the window regulator class which is not a subcomponent of the topology of the body slot of the class results in refusal. This restriction on allowable contents transforms feature hierarchies into a close approximation of part-whole hierarchies.

Only features can be abstracted. Consider Fig. 4, the window regulator device. Within this device is a feature

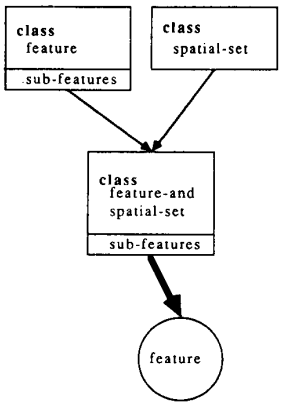


Fig. 8. Classes of spatial sets and features.

sector which in turn contains a number of subfeatures. These features and subfeatures are the only objects in the sector upon which abstractions may be computed. Fig. 10 diagrams this relation. Two implications for the system architecture emerge.

1) Abstractions are in a subordinate relationship to features in the architecture. An abstraction is a partial model of something; in this architecture that something is always a feature. For example, to access the center-line abstractions discussed above it is necessary to first access the feature which possesses the abstraction.

2) Features store abstractions at the appropriate level. Every abstraction which is computed on a feature is stored in that feature. Consider Fig. 9 which shows five abstractions: center-lines of the two mounting holes, face-equation of the top face of the sector (shown as the nested rectangles), base-location of the entire sector and bounding box of the sector. Each of these are stored in different places in the feature hierarchy which denotes the sector.

Classes of spatial sets and abstractions relate in two distinct ways.

Spatial sets can represent abstractions of features. When the center-line abstraction is computed its result is a line-segment, an instance of the class center-line which is in turn a subclass of classes abstraction and line-segment. Class abstraction endows the center-line with certain properties, notably: 1) a slot for storing a back reference to the object to which the abstraction pertains, 2) a slot which marks the abstraction as either computed or declared, and 3) methods to access and modify these slots. From class line-segment are inherited all of the slots and methods which support representation and computation on line-segments. These same slot templates and methods would be used for features which represent line segments. Fig. 11 diagrams this example to show how abstractions use classes of spatial sets. The geometric reasoning architecture benefits from this interaction in a single way. Geometry algorithms are defined in only one place in the reasoning system: in the class definitions for the various types of geometric objects. This greatly simplifies implementation from a software engineering point of view.

Abstractions are virtually inherited. Abstractions are not explicitly inherited from class definitions; rather the methods for creating those abstractions are inherited. The actual abstractions are created, local to the instances and only upon request. This lazy evaluation of abstractions means that 1) only those abstractions that are needed for a particular application are actually computed, and 2) since abstraction instantiations are persistent, abstractions are not repeatedly computed.

Abstractions and constraints interact in two distinct ways.

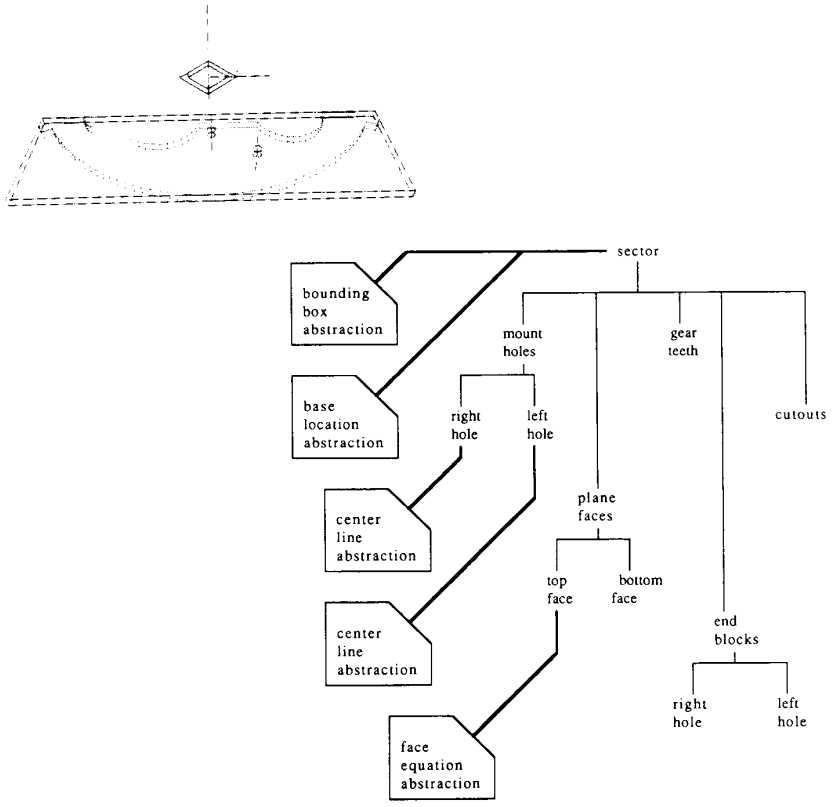


Fig. 9. Abstractions stored at appropriate level in feature hierarchies.

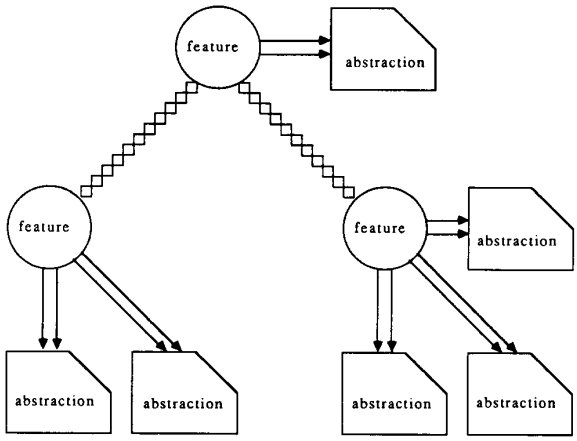


Fig. 10. Features and abstractions.

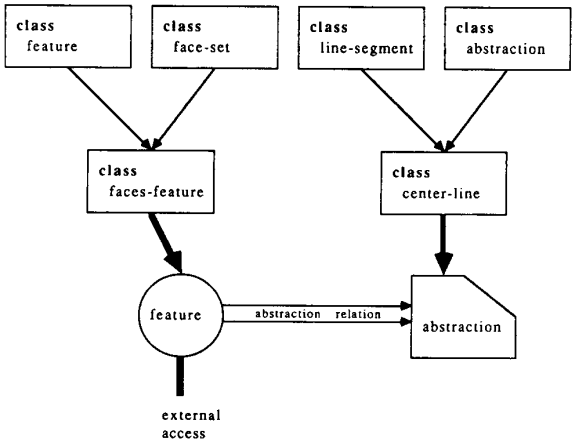


Fig. 11. Classes of spatial sets and abstractions.

Abstractions are the objects upon which constraints act. Abstractions are the units of representation upon which constraints, particularly algebraic constraints, are most conveniently formulated. For example, a center-line is represented as six atomic values. These atomic values are required to state constraints using only simple variables as required by algebraic constraint algorithms. This relation completes a portion of the system architecture. It shows how reasoning on a geometric model may be achieved. The model is accessed through its features, which are then abstracted to partial representations. These partial representations, composed only of simple variables, are used in reasoning processes. Fig. 12 diagrams this process.

Constraints are used to enforce the consistency of abstractions. As discussed in Section IV D, constraints are a vehicle for maintaining the integrity of geometric information. In the context of abstractions, they are used to maintain the consistency of an abstraction as its parent object is changed. For example, the center-line abstraction must update its values if the hole of which it is an abstraction is moved. This is achieved by constraints between the face equation of the hole and the center-line abstraction.

The overall architecture is the conjunction of the all of its subcomponents. Shown in Fig. 13 is such a combination, somewhat simplified in the interests of readability. All of the relations that occur between component concepts in the present implementation are demonstrated. Fig. 14 provides a key to the graphic symbols used in the above diagrams.

## B. Present Implementation

An implementation of this architecture has been achieved and tested in a number of prototype applications. Fundamental, and in retrospect essential, to the implementation is an object-oriented programming environment, particularly the portable common loops (PCL) language.[9] Without the constructs of class, multiple inheritance and multimethods provided by PCL, the implementation would have been much more difficult to achieve.

[9]Developed at XEROX Palo Alto Research Center. The design of PCL was instrumental to the recent specification of an object-oriented standard (CLOS) for Common Lisp. PCL is converging on that standard and stands as its first implementation.

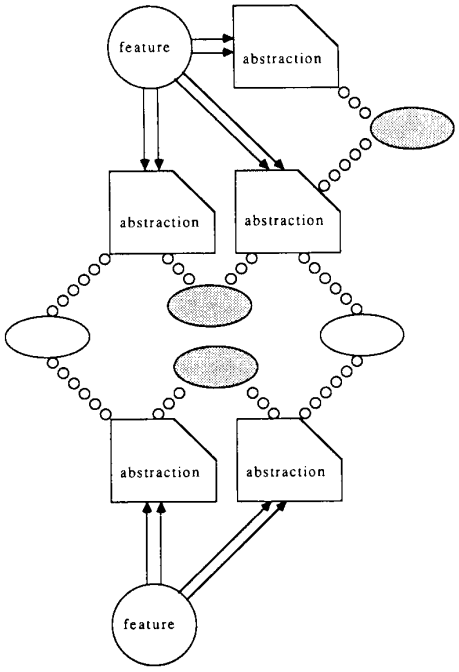


Fig. 12. Abstractions and constraints.

We first implemented a two-dimensional version of a geometric reasoning system which successfully demonstrated the four concepts (spatial sets, features, geometric abstractions, and constraints) constituting the approach. The two-dimensional system operated on polygons. It demonstrated geometric reasoning in this simple realm, and established the validity of the approach. However, it had no particular domain or application context (in no way did it model a robot or robot task environment) nor did it resolve the larger and more challenging implementation problems found in a three-dimensional domain. However, its basic structure was unrestricted and it readily became the foundation for a subsequent full three-dimensional implementation.

The three-dimensional geometric reasoning system required additional system design decisions. Most significant was our decision to use a full geometric modeler as an underlying layer, transparent to system users. The modeler was VEGA [37] and its particular implementation required the development of an interface between the Lisp environment of CommonLoops and the C-language version of VEGA. Another design decision was the isolation of graphics and window manager software from the geometric reasoner. As a result our particular interface environment, the Hewlett-Packard Windows 9000 and Starbase graphics packages, is hidden from the implementation and poses few portability problems.

The implementation is structured as a number of layered components, including the VEGA interface, a spatial set definition capability, a feature definition language, support for the definition of abstraction, a constraint definition language and a constraint solver. All of these have been implemented and the system operates successfully in several prototype applications. However, further demonstration and documentation of results must

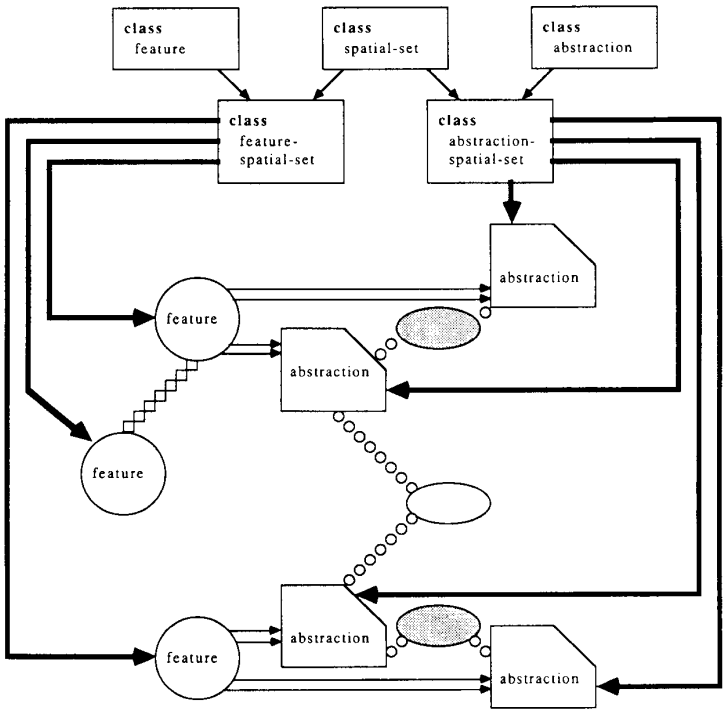


Fig. 13. Overall system architecture.

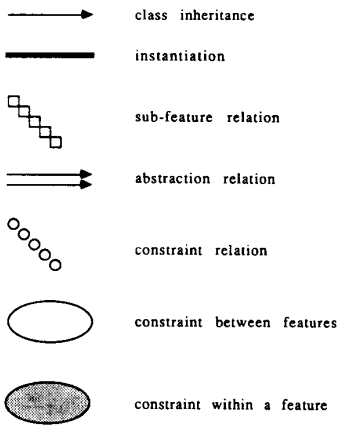


Fig. 14. Graphic symbols for concept interaction diagrams.

await developer experience with major robot domain test cases, which have not at this time been completed.

## VI. CONCLUSIONS

The architecture for geometric reasoning presented here is offered as a promising approach to the representation and manipulation of geometric information for robot task planning, and specifically for use by KBESs. An implementation of the reasoner has been completed for a three-dimensional version sufficiently powerful to test complex robot task planning applications. Such test demonstrations are planned for the domains of nuclear plant repair activities, space telerobotic manipulation, and mechanical parts description for manufacturing.

The objective was to supply a system developer with a tool enabling KBESs to perform robot task planning. Development of the expert systems themselves is another demand, one which has not been addressed in this article, and one which is itself some years ahead of us. Nonetheless, the authors are confident that a reasoner as described will play an important role in robotic task planning. One recommendation to other developers is to exert care in choosing the implementation environment. The influence of language capabilities upon software of this complexity can be pronounced, and the linking to supporting modelers and displays could be prohibitively expensive if not properly managed.

### ACKNOWLEDGMENTS

The authors are pleased to acknowledge the support, interest, and guidance of the Electronic Power Research Institute and Dr. Floyd Gelhaus. We are especially grateful for permission to share this approach with the research community prior to its conclusion and delivery to EPRI, and point out that the results have not yet been reviewed or checked for accuracy.

### REFERENCES

[1] Kuipers, B.J. (1977)
Representing knowledge of large-scale space.
Ph.D. dissertation, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, July 1977.

[2] Lynch, K. (1960)
*The Image of the City.*
Cambridge, Mass.: M.I.T. Press, 1960.

[3] Brooks, R.A. (1981)
Symbolic reasoning among 3-D models and 2-D images.
Ph.D. dissertation, Stanford University, Calif., June 1981.

[4] Davis, E. (1981)
Organizing spatial knowledge.
Technical Report 193, Computer Science Dep't., Yale University, New Haven, Conn., Jan. 1981.

[5] McDermott, D. (1980)
A theory of metric spatial inference.
In *Proceedings of the Second National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Palo Alto, Calif., 1980, pp. 246–248.

[6] Davis, E. (1986)
*Representing and Acquiring Geographic Knowledge.*
London: Pitman Publishing Ltd., 1986.

[7] Akiner, V.T. (1985)
Topology-1: a reasoning system for objects and space modelling via knowledge engineering.
Presented at the Design Engineering Division Conference on Mechanical Vibration and Noise, American Society of Mechanical Engineers, Cincinnati, Ohio, Sept. 1985.

[8] Libardi, E.C., Dixon, J.R., and Simmons, M.K. (1986)
Designing with features: design and analysis of extrusions as an example.
Presented at the Spring National Design Engineering Conference, American Society of Mechanical Engineers, Chicago, Ill., Mar. 24–27, 1986.

[9] Neilson, E.H., Dixon, J.R., and Simmons, M.K. (1986)
How shall we represent the geometry of designed objects?.
Technical report, Department of Mechanical Engineering, University of Massachusetts, Amherst, 1986.

[10] Wing, J.M., and Arbab, F. (1985)
Geometric reasoning: a new paradigm for processing geometric information.
Technical report cmu-cs-85-144, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pa., 1985.

[11] Stiny, G. (1980)
Introduction to shape and shape grammars.
*Environemnt and Planning B, 7,* 3 (1980), 343–352.

[12] Gips, J. (1975)
*Shape Grammar and Their Uses.*
Basel, Switzerland: Birkhauser Verlag, 1975.

[13] Kapur, D., and Mundy, J. (1987)
Wu's method: an informal introduction.
To appear in a Special Issue of Artificial Intelligence on Geometric Reasoning, 1987.

[14] Wu, W.T. (1986)
On the decision problem and mechanization of theorem proving.
*Journal of Automated Reasoning, 2* (Sept. 1986), 221–252.

[15] Sutherland, I.E. (1963)
Sketchpad: a man-machine graphical communication system.
Technical report 296, M.I.T. Lincoln Lab., Lexington, Mass., 1963.

[16] Sussman, G., and Steele, G. (1980)
CONSTRAINTS—a language for expressing almost hierarchical descriptions.
*Artificial Intelligence, 14* 1 (Aug. 1980), 1–40.

[17] Steele, G.L., Jr. (1980)
The definition and implementation of a computer programming language based on constraints.
Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Aug. 1980.

[18] Borning, A. (1979)
ThingLab—a constraint oriented simulation laboratory.
Technical report, Palo Alto Reseach Center, Palo Alto, Calif., July 1979.

[19] Light, R., and Gossard, D. (1982)
Modification of geometric models through variational geometry.
*CAD—Computer Aided Design, 14*, 4 (July 1982), 209–214.

[20] Gosling, J. (1983)
Algebraic constraints.
Ph.D. dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pa., May 1983.

[21] Fikes, R.E., and Nilsson, N.J. (1971)
STRIPS: a new approach to the application of theorem proving to problem solving.
*Artificial Intelligence, 2* (1971), 189–208.

[22] Fahlman, S.E. (1974)
A planning system for robot construction tasks.
*Artificial Intelligence, 5*, 1 (1974), 1–49.

[23] Simmons, R.F. (1982)
Spatial and temporal reasoning in geologic map interpretation.
In *Proceedings of the Third National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Carnegie-Mellon University, Pittsburgh, Pa., Aug. 1982, pp. 152–154.

[24] Klahr, P., McArthur, D., and Narain, S. (1982)
Swirl: an object-oriented air battle simulator.
In *Proceedings of the Third National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Carnegie-Mellon University, Pittsburgh, Pa., Aug. 1982, pp. 331–334.

[25] Ambler, A.P., and Popplestone, R.J. (1975)
Inferring the positions of bodies from specified spatial relations.
*Artificial Intelligence, 6* (1975), 175–208.

[26] Popplestone, R.J., Ambler, A.P., and Bellos, I. (1980)
An interpreter for a language for describing assemblies.
*Artificial Intelligence, 14*, 1 (1980), 79–107.

[27] Agin, G.J. (1981)
Hierarchical representation of three-dimensional objects using verbal models.
*IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-3*, 2 (Mar. 1981), 197–204.

[28] Keirouz, W.T., Rehak, D.R., and Oppenheim, I.J. (1986)
Object-oriented domain modeling of constructed facilities for robotic applications.
In *Proceedings of the 1st International Conference of Applications of Artificial Intelligence in Engineering Problems*, Southampton University, UK, Apr. 1986, pp. 141–150.

[29] Requicha, A.A.G. (1980)
Representation for rigid solids: theory, methods and systems.
*Computing Surveys, 12* 4 (Dec. 1980), 437–464.

[30] Eastman, C.M., and Weiler, K. (1979)
Geometric modeling using the Euler operators.
Technical report 78, Institute of Physical Planning, Carnegie-Mellon Univ., Pittsburgh, Pa., Feb. 1979.

[31] Preparata, F.P., and Shamos, M.I. (1985)
*Computational Geometry—An Introduction.*
New York: Springer-Verlag, 1985.

[32] Milenkovic, V. (1987)
Verifiable implementations of geometric algorithms using finite precision arithmetic.
To appear in a Special Issue of Artificial Intelligence on Geometric Reasoning, 1987.

[33] Flemming, U. (1986)
On the representation and generation of loosely-packed arrangements of rectangles.
*Planning and Design* (Dec. 1986), 12:189–205.

[34] Gross, M.D. (1986)
Design as exploring constraints.
Ph.D. dissertation, Department of Architecture, Massachusetts Institute of Technology, Cambridge, Feb. 1986.

[35] Woodbury, R. (1987)
The knowledge based representation and manipulation of geometry.
Ph.D. disseration, Department of Architecture, Carnegie-Mellon University, Pittsburgh, Pa., Oct. 1987.

[36] Kingsley, C. (1981)
Earl: a language for integrated circuit design.
Ph.D. dissertation, California Institute of Technology, Pasadena, 1981.

[37] Woodbury, R. (1986)
VEGA: A geometric modeling system.
Presented at the IBM Advanced Educational Projects Conference, IBM, San Diego, Calif., Apr. 1986.

**Robert Woodbury** received the B.Arch. degree from Carleton University, Ottawa, Canada in 1981, and his M.S. and Ph.D. degrees from Carnegie-Mellon University, Pittsburgh, Pa., both in architecture, in 1982 and 1987.

He is currently an Assistant Professor in the Department of Architecture at Carnegie-Mellon University. His research activities include representations and algorithms for spatial reasoning, computer-based design environments, geometric modeling, and tolerance optimization in mechanical design.

**Irving Oppenheim** received his BCivil degree from The Cooper Union, New York, N.Y. in 1968, and the MS (civil) from Lehigh University, Bethlehem, Pa. in 1970. He received the PhD (Civil Engineering) from Cambridge University, England, in 1972.

He is an Associate Professor, jointly appointed in the Departments of Architecture and Civil Engineering at Carnegie-Mellon University, Pittsburgh, Pa. His recent research activities include dynamic stability in manipulation, control of flexible link manipulators, force influences and spatial influences in task planning, and representations for spatial reasoning.