SRI International

HIERARCHICAL REPRESENTATION OF THREE-DIMENSIONAL OBJECTS USING VERBAL MODELS

Technical Note 182

March 1979

By: Gerald J. Agin, Senior Computer Scientist

Artificial Intelligence Center

SRI Project 1187

The work reported herein was supported by the Office of Naval Research under Contract N00014-71-C-0294.

Paper to be presented at a Workshop on Representation of Three-Dimensional Objects, Philadelphia, Pa., May 2, 1979

SRI International

333 Ravenswood Ave. • Menlo Park, California 94025
(415) 326-6200 • Cable: STANRES, Menlo Park • TWX: 910-373-1246

ABSTRACT

We present a formalism for the computer representation of three-dimensional shapes, that has as its goal to facilitate man-machine communication using verbal, graphic, and visual means. With this method, pieces may be assembled hierarchically using any of several ways of specifying attachment. The primitives of the representation are generalized cylinders, and the creating of assemblies may make use of the axes inherent in the primitives. Generic models may be described that may leave some parameters or dimensions unspecified, so that when a specific instance of the model is described, those parameters may either be explicitly specified or take on default values. The axes of local coordinate frames may be given symbolic names. A set of computer programs translates descriptions of objects into polyhedral models and line drawings.

I INTRODUCTION

The computer representation of three-dimensional shape has occupied the attention of a number of researchers in the past several years. The interest arises at least in part because a useful theory of shape would have applicability to a wide variety of fields, such as design automation, manufacturing automation, terrain mapping, vehicle guidance, archaeology, restoration of works of art, surveillance, and intelligent robots in general. But aside from any practical applications, the problem has a great deal of inherent scientific and mathematical interest.

Finding a useful and general method for representing shape is not a simple problem. Methods that are primarily numerical are usually limited in generality. They have great precision, but they are usually restricted to a specific domain. Humans, on the other hand, can communicate and understand a wide variety of shapes.

Various systems of computer representation have evolved or have been invented for specific tasks. Generally, they fall into two categories: surface representations and volume representations.

Surface may be approximated by triangular patches [1] with various interpolation schemes. Rectangular patches have been used successfully with spline interpolation for the precise specification of airfoils and automobile bodies [2]. Contour maps are a surface representation widely used for a number of applications.

In volumetric modeling, the usual approach is to represent objects as intersections and unions of simpler objects. Combining polyhedra and cylinders in this way has been shown to be useful in the description of machined metal parts for design automation [3-7]. Generalized cylinders have been used for modeling many everyday shapes for computer vision [8,9]. A more specialized use of generalized cylinders has been used in classification of pottery shapes [10].

The facet of the problem we are most interested in is the representation of shape in ways that can be easily communicated and understood by humans. This rules out methods that involve equations or large arrays of numbers. Natural media of communication include (1) words, (2) pictures, and (3) examples. We use the terms verbal communication, graphical communication, and visual communication to denote these three modes.

Informal studies show that notions of similarity and difference are important to human communication of shape concepts. The descriptions used frequently begin with a familiar object, then mention significant differences between the object being described and the familiar one. Many alternate descriptions of the same object are possible.

This report describes one approach to describing shapes in natural, human terms. The method is hierarchical, using generalized cylinders to characterize primitive elements and their assemblies into higher level subparts and parts.

The primitives of the representation are generalized cylinders, comprised of a central axis or spine, and a cross section function defined on that axis. The primitives may be combined in ways that make use of the axes inherent in the primitives. In many cases, however, spine/cross-section representation is not appropriate, and more general but less intuitive methods are needed. We provide the capability for representing arbitrary spatial relationships for these cases.

Examples of some of the objects that may be usefully represented by our methods are shown in Figures 1, 2, and 3. Figure 1 shows a screwdriver, for which spine/cross-section methods are completely sufficient. Figure 2 is a representation of an airplane. Spine/cross-section methods are used to define most of its structure and shape, but some auxiliary positioning methods are needed for some of its component parts. For the chair of Figure 3, only general structural relationships will conveniently work.

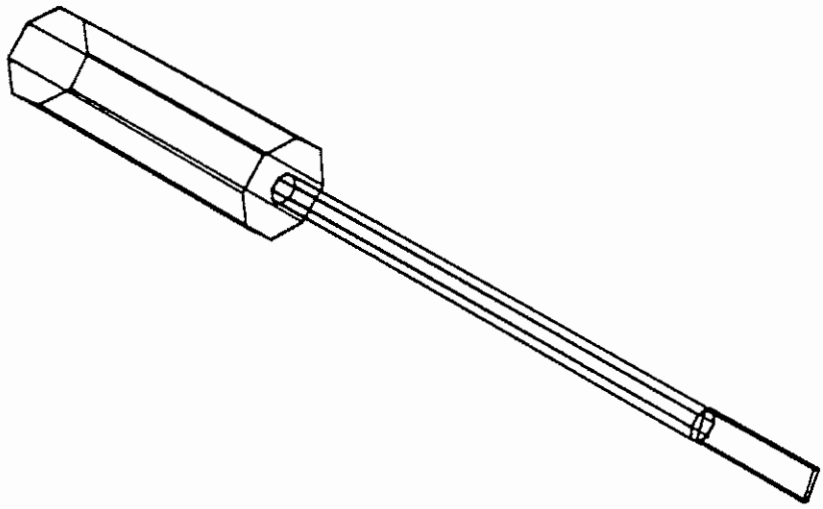

Figure 1 Model of a Screwdriver

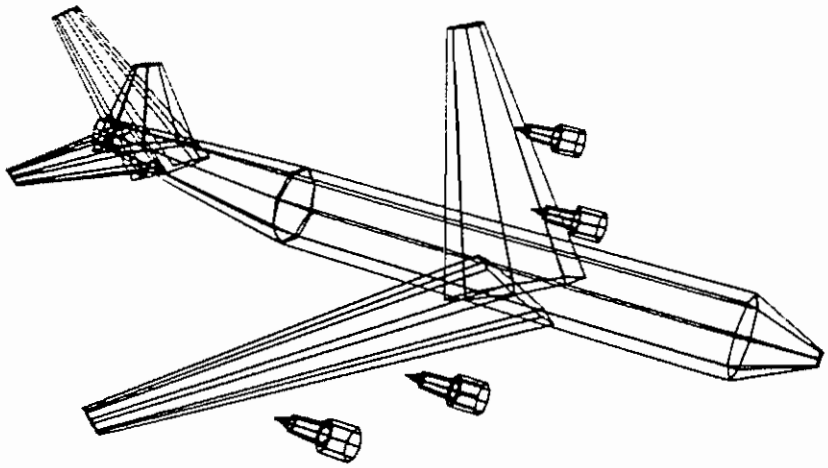

Figure 2 Model of an Airplane

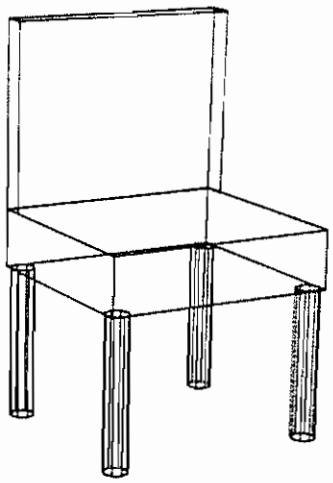

Figure 3 Model of a Chair

The representation was developed without a specific application in mind. One of the scenarios we used is similar in many ways to computer-aided design: A user of the system describes an object to the computer using LISP S-expressions, and the system then draws a picture of the object to demonstrate that it understands the description. In another scenario, stored descriptions are used to guide the segmentation of data from a time-of-flight range finder.

The following sections present the representation in more detail.

II REPRESENTATION BY SPINE/CROSS-SECTION METHODS

Representation by spine/cross-section methods uses a space curve, or axis, and a cross-section function defined on this axis. The primitives of this representation have been called generalized cylinders [8]. Given a simple object, we may determine its description by locating an axis such that the object's cross section (normal to the axis) varies in a uniform manner along the axis. A description of a complex object may be built up by "cutting and pasting" the descriptions of its constituent parts.

These concepts provide a natural, intuitive way of representing certain solid objects. The model represents volumes, instead of surfaces or appearances. The method allows segmentation of a complex object into parts easily represented, and a hierarchical approach to the description of objects.

Representation by generalized cylinders is synthetic in nature; that is, given a model, one can uniquely synthesize the contours of the object. As an analytic representation (for generating a model to describe a given real object), only objects that have a well-defined axis are easily described. In general, analysis in terms of an axis and cross section does not yield a unique answer; heuristic or interactive techniques are needed to select natural or useful axes.

Segmentation is a critical issue here. For analysis or for synthesis it is necessary to find parts of an object that may be described simply and to paste these segments together.

In our representation, objects may be composed of an assemblage of smaller subobjects, or they may be primitive. Only primitive objects have an explicit cross-section description; for all other objects the shape is described by the shapes of, and relationships among, its component parts. Hierarchical representations of complex objects may be built up by independently describing subparts, then describing the structural relationships among the subparts.

III STRUCTURAL RELATIONSHIPS AMONG PARTS

There are three fundamental ways the relationships among parts may be specified in our representation: (1) Structural relationships involving snakes, which describe objects displaced along a single axis, like beads on a string. The axes of the individual parts combine to form the axis of the assembly. (2) Relationships involving attachment points, which are like Tinkertoys: parts have predefined points at which other pieces may be attached. (3) Displacement by arbitrary transforms, which is the most general of the three methods; the other two can be considered special cases of arbitrary displacements. But snakes and attachment points are closer to intuitive notions of structure, and usually assume a more compact form. Usually the description of any object of significant complexity will use all three of these methods.

For ease in exposition, we will consider general relationships first, then return later to the easier-to-use structures.

A. General Positional Relations

Consider the task of specifying a structure in which a one-inch cube rests on top of a two-inch cube, as shown in Figure 4. Suppose the two subparts have been defined as will be shown in Section V and given the names CUBE1 and CUBE2. Then the structure may be described by the following S-expression:*

(CUBE2 (TRANSLATE +Z 2) CUBE1)

We start the interpretation of this description by placing the two-inch cube at the origin of the local coordinate system. The coordinate system is then translated two inches upward, and the one-inch cube is placed at the relocated origin of coordinates.

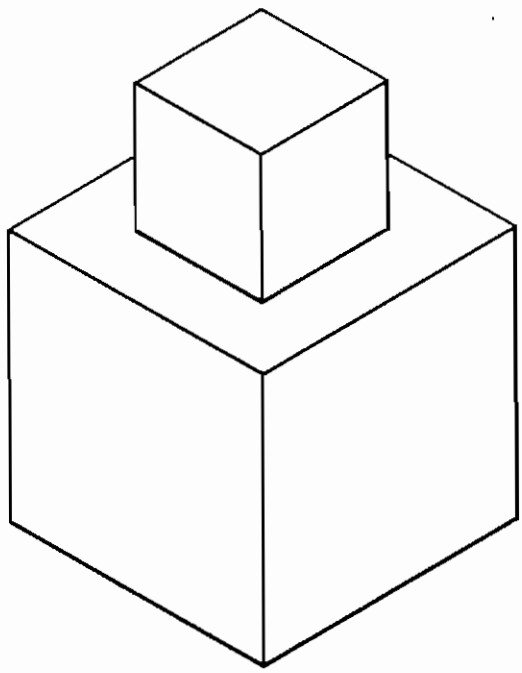

Figure 4 A One-Inch Cube on Top of a Two-Inch Cube

Syntactically, a general structural relationship is represented by a list of pieces and transforms. A piece may be a symbolically named subpart (see Section V), or it may be another structure. This

---

* See any standard text on LISP [12 or 13] for a description of lists and S-expressions.

allows a hierarchy of structures within the hierarchy of parts and subparts.

A transform is an operation that changes one coordinate system into another. The two most frequently used primitive transforms are ROTATE and TRANSLATE, which provide rotary and linear motion, respectively. Several additional transformation primitives give additional descriptive power to the representation, such as SCALE, MIRROR, and SKEW.

A complete description of the syntax of general structural relationships is given in Appendix A.

A general structure descriptor should be interpreted from head to tail. The position of any piece with respect to the coordinate system in which the overall structure is described is the product of all the transforms that precede the piece in the list. The structure descriptor may be thought of as instructions to maneuver a "bus" that drops pieces at its local origin of coordinates between maneuvers.

The chair of Figure 3 was assembled from primitive elements using only translational and rotational transforms.

B. Attachment Points

A difficulty in the preceding example is that we must know the height of CUBE2 in order to say where to place CUBE1. The situation can be made easier by the use of symbolic names instead of numbers, but there still is the problem of ascertaining that the names match. It is much more useful to say, in effect, "place CUBE1 on top of CUBE2," regardless of the size of either cube.

Attachment points provide one way to accomplish this. The following S-expression describes the two cubes of Figure 4 in terms of attachment points:

```
( CUBE2 (ATTACH CUBE2 TOP) CUBE1 )
```

An attachment point reference is syntactically equivalent to a transform. It must contain the symbolic name of another subpart at the appropriate level of hierarchy--i.e., the name of a piece occurring

previously in the structure description. The part name must be followed by the name of an attachment point, which must be defined elsewhere.

For all primitive objects the symbolic attachment-point names BASE, TOP, SIDE, and BACK are predeclared. For nonprimitive objects, attachment points may be declared as described in Section V. Declaring a new attachment point involves specifying a transform or relative displacement with respect to the BASE of an object--i.e., the origin of the coordinate system in which the object is described.

Attachment points are particularly useful in describing biological (animal, human) shapes and a wide variety of manufactured objects. A computer model of an air compressor [11] was built almost entirely with attachment-point structures. To represent the airplane of Figure 2, we used attachment points to designate the attachment of the wings and tail assembly to the fuselage.

C. Snakes and Stacks

A snake is a group of objects sharing a single extended axis. It is the snake relationship that gives our representation its spine/cross-section capabilities. All primitive objects are defined to have an axis and a length. Nonprimitive objects may also have axes and lengths, either explicitly specified or implicitly derived. To assemble parts in a snake structure we place the components end-to-end with their axes touching, like stringing beads on a necklace.

Continuing our example of Figure 4, we have the following S-expression as a representation of the structure using a stack:

```
( STACK CUBE2 CUBE1 )
```

As a LISP S-expression, a stack is a list containing the atom STACK followed by an arbitrary number of pieces. Each piece must have an axis and length defined. The structure created by interpreting this expression will have its axis vertical, with pieces stacked from bottom to top. If the ultimate orientation of the structure is to be other than vertical, additional transforms will be needed. Axes with bends or

corners in them may be created by inserting rotation transforms between the component parts.

Each primitive object has a predefined axis and length. When parts are assembled in a stack, the combined axis and length are assigned to the structure. For nonprimitive objects assembled by other means, an axis may be defined as a length plus a transform to define the axis direction and location.

The screwdriver of Figure 1 was modeled exclusively with the STACK construct. The fuselage of the airplane of Figure 2 was modeled with a stack.

D. Summary

For the example of the two blocks stacked on top of one another, the three methods of specifying their relative positions achieved the same result by using the same interpretive machinery. But the compactness of the description varied, as did the amount of variation allowable in the description. Description by means of stacks is the most compact and most intuitive; it requires that parts to be mated have their axes lined up. Description of structure by means of attachment points removes the restriction of coincident axes, but requires an explicit description of the attachment point in cases where the attachment point is not predefined. Description by means of arbitrary transforms is the most general of all, but does not easily allow displacements to be a function of the base object.

IV PRIMITIVES

At the lowest level of the hierarchy of parts and subparts are the primitives--pieces that are not further divisible and about which the drawing routines have specific knowledge.

The most basic primitive we provide is the cylinder. Specifically, we mean a right circular cylinder; the ends are square and the cross

section is circular. It is described in its canonical orientation: with its axis extending upward (in the +z direction) from the origin of coordinates. The two specifiable dimensions are LENGTH and DIAMETER.

The other primitive we provide is the brick. A brick is similar to a cylinder in all respects except the shape of its cross section. Specifiable dimensions are LENGTH (parallel to the z-axis in the canonical orientation), WIDTH (parallel to the x-axis), and DEPTH (parallel to the y-axis).

It would be easy to create other primitives with new cross section shapes. But the cylinder and the brick have been adequate so far for our purposes. Clearly a general cross section description facility would be useful for describing such things as rulers or fluted screwdriver handles. But we have not given much thought as to how to implement such a capability.

Note that some shapes are derivable from the circle and the rectangle. For example, an elliptical cross section can be generated by transforming a cylinder with a nonuniform SCALE transform.

Cross section dimensions are allowed to vary linearly from one end of the segment to the other. This allows the generation of conical or pyramidal shapes. DIAMETER, DEPTH, and WIDTH apply to the base of the primitive. The dimensions at the top are determined by the specifiable parameter TAPER. A TAPER of zero (the default) implies a uniform cross section; otherwise the top dimensions are (TAPER+1) times the bottom dimensions. A TAPER of -1 yields a cone or pyramid with a point at the top.

The only type of axis segment we currently permit is a straight one. Curved axes may be approximated by piecewise linear axes by including rotation transforms in a STACK construct. The obvious first-order improvement would be to allow axes to have a curvature (numerically, the reciprocal of the radius of curvature). The plane of curvature would have to be fixed with respect to the canonical position. This first-order extension should be able to handle 95% of the usual cases of objects modeled with curved axes.

V CREATING OBJECTS FROM PRIMITIVES

There are three classes of object in our representation: prototypes, descriptions, and instances. A prototype contains information regarding a class of objects, including nominal or typical dimensions, and allowable variation. A prototype is usually created by a programmer, who decides how to represent a particular object or piece. (An interesting question for future research is how to write a computer program that creates prototype descriptions from visual or range data, possibly with interactive help from a human teacher.) The prototype SCREWDRIVER in Appendix B contains all information the system knows about screwdrivers in general.

Descriptions and instances refer to particular objects. A description is a concise recipe for copying a prototype and assigning specific values to its variables to form an instance. The description

```
(SCREWDRIVER (LENGTH 14)
             (BLADE-THICKNESS (TIMES DEFAULT .5) ) )
```

refers to a screwdriver 14 inches long and with a narrow blade.

An instance is a copy of a prototype, with all of its parameters, dimensions, and options given specific values. The process of converting a description to an instance is called instantiation. Upon instantiation, the description above might produce an instance whose name is SCREWDRIVER0023.

Prototypes and instances are carried in our representation as LISP atoms with property lists. The principal entries on the property list of a prototype are as follows:

* DIMENSIONS--Names the sizes, angles, and other parameters that may vary or be specified in a description, and their defaults.
* COORDINATE-FRAME--May define a local coordinate system.
* PARTS--Specifies the pieces to be assembled to produce this object.
* STRUCTURE--Gives the spatial relationships of the parts to each other and to the whole, as described in Section III.

* ATTACHMENTPOINTS--Can provide named locations for assembly with other objects.
* AXIS--Specifies the direction and length of the central axis of the object.

Instantiating an object involves creating a context in which symbolic expressions may be evaluated. (The spaghetti stack capability of INTERLISP [12] permits the preserving of contexts and their variable bindings.) A tree of contexts parallels the tree of subparts.

When an instance is created from a prototype, a new context is created. The DIMENSIONS and the COORDINATE-FRAME properties of a part provide a list of names that will be bound in the new context. For instance, if the identifier LENGTH heads an entry in the DIMENSIONS property, a new variable of that name will be created as a part of the new context. Usually each variable name will have a default value or expression for the variable to take. The defaults may be overridden by modifiers in a description (as in the case of the 14-inch screwdriver, above).

A large number of dimensions are specified in the SCREWDRIVER description of Appendix B. The large number of dimensions is desirable so that we can precisely specify the shape for drawing routines, but the programmer who makes use of the prototype description need not concern himself with the ones he is not explicitly modifying.

The DIMENSIONS property of the chair in Figure 3 contains the dimension names SEAT-WIDTH, SEAT-DEPTH, and LEG-DIAMETER (among others). The numbers corresponding to these names are used to calculate the relative positions of the legs with respect the the chair's seat.

At some time or another, each dimension must be evaluated--i.e., a number must be calculated and assigned. At instantiation time, we have the choice of transferring the unevaluated symbolic expressions from the property list of the prototype to the variable bindings of the new context, or evaluating each expression and storing the number as the value of the variable. If expressions are stored, evaluation will take place only when a number is needed (for instance, by the drawing

program). This keeps useful information available for question answering purposes, and will permit changing top-level parameters and positions without the necessity of modifying the data structure. On the other hand, the storing of numbers is vastly more efficient if semantic capabilities are not needed. In our implementation of the representation, provision is made to carry out instantiation in either mode: numeric or symbolic.

A unique feature of our representation is the ability to specify local coordinate directions. The need for this becomes apparent when parts are being assembled in a major assembly. It is frequently advantageous to describe a part in one orientation when its eventual mounting will be in another direction. Consider, for example, what it means to refer to the "top" of a bolt. If the bolt were considered in isolation, the top would be assumed to refer to the head end, but if the screw were inserted in a horizontal position the "top" would be meaningless. We permit specifying direction names such as TOWARD-HEAD or TOWARD-TIP that are "embedded" in the object regardless of the object's eventual orientation. Some more useful applications include establishing a FRONT and BACK to assemblies, or a BOW, STERN, PORT, and STARBOARD for ships, aircraft, and vehicles. The screwdriver prototype of Appendix B defines the direction names HANDLE and TIP.

The entries in the list of COORDINATE-FRAME definitions are treated identically to the variables of the DIMENSIONS property. That is, for the screwdriver example, the names HANDLE and TIP are bound on the stack and given the default values associated with each.

Every nonprimitive object should have a PARTS property. Each part is given a symbolic name and is described by a part description with appropriate modifiers. The STRUCTURE property specifies how the parts are to be placed in relation to each other. Instantiating any object causes its parts to be instantiated also, so that a hierarchy of instances will parallel the hierarchy of prototypes.

To describe the airplane shown in Figure 2 required prototypes with up to five levels of hierarchy. The screwdriver example in the appendix

has only a limited hierarchy. The prototype SCREWDRIVER-HANDLE is also a component of other parts; the parts lists of other objects point to it. The modifiers in the descriptions in the PARTS property of SCREWDRIVER refer to the variables that will be bound in accordance with the DIMENSIONS property of SCREWDRIVER-HANDLE.

The screwdriver has no attachment points defined. However, examples of attachment-point definitions may be seen on the property lists of BRICK and CYLINDER in Appendix B.

VI THE DISPLAY OF WIRE MODELS

We have a set of techniques and computer programs that will draw simple "wire frame" displays of parts described in our formal representation. We do this not only to demonstrate that graphical interaction with the models is possible, but also to ensure that our modeling techniques are correct and unambiguous. Unless we are capable of displaying a number of parts in correct relationship to one another, we cannot be sure the relationship has been adequately defined. A graphical capability is necessary for debugging the individual models, and for assessing the adequacy of the descriptive method itself.

It is not our intention to invent new techniques for the display of polyhedral models. Many other systems exist for display of polyhedra [3-7]; these systems offer advanced capabilities such as hidden-line elimination, shading, and calculation of interpenetration. There is sufficient information available in our models to interface with any of these programs.

Once an object has been instantiated, the instance may be passed to the drawing routines for display. This process occurs in three stages. First, the position and orientation of every primitive object in the hierarchy must be computed. Next, the spine/cross-section representations are converted to approximating polyhedra, and the corners and edges are stored in a buffer array. Lastly, the picture is

drawn on the display console, using a perspective transformation whose parameters are controllable by an operator at the terminal console.

The computation of position and orientation involves a fairly complicated set of actions. The position of a top-level object is specified at the time it is instantiated; if no position is specified the canonical orientation at the origin of coordinates defaults. The position of each subpart of that object is the product of two transforms--the position of its parent part (the next higher level in the hierarchy), and its relative position within the parent assembly. The position within the parent may be computed from the STRUCTURE property, and it is stored in the property list under the property POSITIONINPARENT. The position of any low-level subpart instance is the product of a sequence of relative positions up the hierarchy, times the position of the top-level assembly.

The routines to interpret the structure are a large portion of the entire software package. They must be cognizant of the dimensions of all the parts to be assembled, the correct order in which to evaluate positions, in which context to evaluate each datum, and numerous other factors.

Transforms (positions and orientation descriptors) are carried in symbolic form in instance prototypes, as S-expressions of the forms (TRANSLATE ...) and (ROTATE ...). A certain amount of symbolic computation can be done to multiply the transforms together with symbolic results. The computation makes use of rules such as following:

(TRANSLATE A B C) (TRANSLATE D E F) = (TRANSLATE A+D B+E C+F)

(ROTATE <any axis> A) (ROTATE <same axis> B)
= (ROTATE <same axis> A+B)

(ROTATE -<any axis> A) = (ROTATE +<same axis> -A)

(ROTATE +X 90) (TRANSLATE A B C)
= (TRANSLATE A -C B) (ROTATE +X 90)

In many cases the product of a large number of relative displacements and rotations can be expressed as a single translation times a single rotation.

Numerical information will be needed for the actual display, so numerical evaluation routines generate a 4 x 4 homogeneous transform matrix [14] according to the "directions" in the symbolic transform. The dimensions and the transform matrix are now ready to be passed to the actual drawing routine.

Each type of primitive object has its own drawing routine; the cylinder drawing routine is typical. To draw a cylinder requires knowing its dimensions (LENGTH and DIAMETER) and its position in space. The atoms LENGTH and DIAMETER are evaluated in the context of the variable bindings of this cylinder. If numeric information has been bound in the context of the part, the evaluation will yield a direct answer. Otherwise, repetitive evaluation must occur until a numeric answer is found.

To display any polyhedron, the three-dimensional coordinates of each of its edges are computed and stored in a separate display data structure. (Cylinders are approximated by octagonal prisms for display purposes.) When the edges are plotted in perspective on the display screen, the polyhedra appear transparent, with wire edges. No attempt at hidden line elimination is made.

The edges are drawn according to a perspective transformation representing an imaginary camera. The position, orientation, and internal parameters of this camera are controllable by a keyboard interpreter to produce an arbitrary view of the object or assembly. Keyboard commands can simulate the translation or rotation of the scene or the camera with respect to a variety of coordinate systems. Split-screen stereo may also be produced.

When a polyhedron is placed in the display data structure, the indices of the first and last points in that data structure are stored in its property list so that if changes are made to the part's description (i.e., if the part is moved) the display data structure may be updated without redrawing the entire scene.

The wire-model drawings of Figures 1 through 3 were produced from the prototypes in Appendix B and similar prototypes by the routines described in this section.

VII CONCLUSIONS

A formal representation has been presented that permits description of solid objects in natural, intuitive terms. A special feature of this representation is the ability to define prototype objects and to create instances of the prototypes with selected dimensions or parameters altered. Another special feature is the variety of means for specifying the relationships of objects and parts to each other, methods that exploit inherent axes in the objects. Information is stored semantically rather than numerically, making it available for a variety of purposes.

Prototypes have been written to describe three disparate objects: a screwdriver, an airplane, and a chair. The complete description of one of these, the screwdriver, is presented in Appendix B. Instances have been generated from these prototypes, and wire-model displays have been drawn.

The models have been used in a research effort to analyze data from a time-of-flight laser range finder [15]. Briefly, a sequence of interactive techniques segments the array of range data into portions corresponding to the primitive elements described in the preceding sections. Surface-fitting routines find the sizes, positions, and orientations for the primitives that best fit the measured data. These results are put into correspondence with the hierarchical models.

Thus with the aid of our models, the range-finder has been used as a crude interactive modeling system. The techniques of shape modeling and of range data analysis can form the basis for a future system for computer vision. To achieve such a capability would be a truly significant achievement.

Appendix A

THE SYNTAX OF STRUCTURAL RELATIONSHIPS

Structural relationships are described by LISP S-expressions. Words in upper case denote specific literal atoms--i.e., themselves. Words in lower case denote S-expressions that are defined or described elsewhere. For a description of dotted pairs and lists see Weissman's introduction to LISP [13].

```
assembly ::= gen-structure
         ::= stack

gen-structure ::= NIL
              ::= ( transform . gen-structure )
              ::= ( piece . gen-structure )

stack ::= ( STACK . stacktail )

stacktail ::= NIL
          ::= ( piece . stacktail )
          ::= ( rotation-transform . stacktail )

transform ::= attachment
          ::= translation-transform
          ::= rotation-transform
          ::= skew-transform
          ::= scale-transform
          ::= reflection-transform

attachment ::= ( ATTACH part-name attachment-point-name )

translation-transform ::= ( TRANSLATE direction number )
                      ::= ( TRANSLATE number number number )
     (The first form transform generates a motion parallel
     to a single axis.  For the second form the three numbers
     indicate simultaneous motions in x, y, and z, respectively.)

rotation-transform ::= ( ROTATE direction number )
     (The ROTATE transform generates a rotation about one axis.)

skew-transform ::= ( SKEW direction number )

scale-transform ::= ( SCALE direction number )
                ::= ( SCALE number )
     (The first form denotes a stretching or compression in a
     single dimension only.  The second form denotes a uniform
     change in scale.)
```

```
reflection-transform ::= ( MIRROR direction )

direction ::= X | +X | -X | Y | +Y | -Y | Z | +Z | -Z

piece ::= part-name
      ::= assembly

part-name ::= identifier

attachment-point-name ::= identifier

number ::= identifier
       ::= numerical constant
```

Appendix B

SCREWDRIVER PROTOTYPE

What follows is the complete description of a generic screwdriver, as property lists attached to the LISP atoms SCREWDRIVER, SCREWDRIVER-HANDLE, BRICK, and CYLINDER.

Similar prototypes have been created to represent a chair and an airplane. Space restrictions do not allow their reproduction here.

```
(PUTPROPS SCREWDRIVER
   DIMENSIONS ((LENGTH (BETWEEN 6 24))
               (HANDLE-LENGTH (TIMES (BETWEEN .2 .5)
                                     LENGTH))
               (SHAFT-LENGTH (DIFFERENCE LENGTH HANDLE-LENGTH))
               (SHAFT-DIAMETER (TIMES LENGTH (BETWEEN .01 .05)))
               (TIP-LENGTH (TIMES SHAFT-DIAMETER 4))
               (HANDLE-DIAMETER (TIMES LENGTH (ABOUT .1)))
               (SLOT-THICKNESS (TIMES SHAFT-DIAMETER (BETWEEN .1 .3)))
               (SLOT-LENGTH SHAFT-DIAMETER))
   PARTS ((HANDLE (SCREWDRIVER-HANDLE (LENGTH HANDLE-LENGTH)
                                      (DIAMETER HANDLE-DIAMETER)))
          (SHAFT (CYLINDER (LENGTH (DIFFERENCE SHAFT-LENGTH
                                               TIP-LENGTH))
                           (DIAMETER SHAFT-DIAMETER)))
          (BLADE (BRICK (LENGTH SHAFT-DIAMETER)
                        (WIDTH SLOT-THICKNESS)
                        (HEIGHT TIP-LENGTH))))
   STRUCTURE ((ROTATE -Y)
              (TRANSLATE 0 0 (MINUS (QUOTIENT LENGTH 2)))
              (STACK HANDLE SHAFT BLADE))
   COORDINATE-FRAME ((HANDLE X)
                     (TIP -X))
```

```
   ATTACHMENTPOINTS NIL)

(PUTPROPS SCREWDRIVER-HANDLE
   DIMENSIONS ((LENGTH (ABOUT 6))
               (DIAMETER (TIMES LENGTH .25))
               (XSEC (QUOTE HEXAGON))
               (!LENGTH LENGTH)
               (!DIAMETER DIAMETER)
               (CYL-LENGTH LENGTH))
   PARTS ((HANDLE (CYLINDER (LENGTH !LENGTH) (DIAMETER !DIAMETER))))
   STRUCTURE ((STACK HANDLE))
   COORDINATE-FRAME ((BUTT X)
                     (BLADE -X)))

(PUTPROPS BRICK
   DIMENSIONS ((LENGTH)
               (DEPTH WIDTH)
               (HEIGHT)
               (WIDTH DEPTH)
               (CYL-LENGTH HEIGHT))
   ATTACHMENTPOINTS ((BASE NIL)
                     (TOP (TRANSLATE +Z HEIGHT))
                     (SIDE (TRANSLATE (TIMES HEIGHT .5)
                                      0
                                      (TIMES HEIGHT .5))
                           (ROTATE Y 90))
                     (BACK (TRANSLATE 0
                                      (TIMES HEIGHT .5)
                                      (TIMES HEIGHT .5))
                           (ROTATE X 90))))

(PUTPROPS CYLINDER
   DIMENSIONS ((HEIGHT LENGTH)
               (DIAMETER)
               (BOTTOM-DIAMETER DIAMETER)
               (TOP-DIAMETER DIAMETER)
               (CYL-LENGTH HEIGHT)
               (LENGTH HEIGHT))
   ATTACHMENTPOINTS ((BASE NIL)
                     (TOP (TRANSLATE +Z LENGTH))
                     (SIDE (TRANSLATE (TIMES DIAMETER .5)
                                      0
                                      (TIMES LENGTH .5))
                           (ROTATE Y 90))
                     (BACK (TRANSLATE 0
                                      (TIMES DIAMETER .5)
                                      (TIMES LENGTH .5))
                           (ROTATE X 90))))
```

REFERENCES

1. R. E. Barnhill et al., "Smooth Interpolation in Triangles," Journal of Approximation Theory, Vol. 8, pp. 114-128 (1973).

2. S. A. Coons, "Surfaces for Computer-Aided Design of Space Forms," Project MAC Report MAC-TR-41, Massachusetts Institute of Technology, Cambridge, Mass. (June 1967).

3. I. C. Braid, Designing with Volumes (Cantab Press, Cambridge, England, 1973).

4. B. G. Baumgart, "GEOMED--A Geometric Editor," Stanford Artificial Intelligence Project Memo AIM-232, Stanford University, Stanford, Calif. (May 1974).

5. A. A. G. Requicha, "Part and Assembly Description Languages -- I, Dimensioning and Tolerancing Facilities in PADL," Production Automation Project Report TM-19, University of Rochester, Rochester, New York (1976).

6. A. A. G. Requicha et al., "Part and Assembly Description Languages -- II, Proposed Specifications for Definitional Facilities in PADL-1.n and Tentative Specifications for Command Facilities," Production Automation Project Report TM-20a, University of Rochester, Rochester, New York (1974).

7. N. Okino et al., "TIPS-1: Technical Information Processing System for Computer-Aided Design," in Computer Languages for Numerical Control (American Elsevier, New York, 1973).

8. G. J. Agin and T. O. Binford, "Computer Descriptions of Curved Objects," IEEE Transactions on Computers, Vol. 25, No. 4 (April 1976).

9. R. K. Nevatia and T. O. Binford, "Structured Descriptions of Complex Objects," Proc. Third International Joint Conference on Artificial Intelligence, Stanford, Calif. (1973).

10. J. M. Hollerbach, "Hierarchical Shape Description of Objects by Selection and Modification of Prototypes," Master's Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass. (1975).

11. Nils J. Nilsson, ed., "Artificial Intelligence--Research and Applications," Progress Report, Project 3805, Stanford Research Institute, Menlo Park, Calif. (December 1975).

12. W. Teitelman, Interlisp Reference Manual,Xerox Palo Alto Research Center Palo Alto, Calif. (October 1978).

13. C. Weissman, LISP 1.5 Primer (Dickenson Publishing Company, Belmont, Calif., 1967).

14. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis (John Wiley and Sons, New York, N. Y., 1973).

15. G. J. Agin, "Hierarchical Representation of Three-Dimensional Objects," Project 1187, Stanford Research Institute, Menlo Park, Calif. (March 1977).