M. A. Wesley
T. Lozano-Pérez
L. I. Lieberman
M. A. Lavin
D. D. Grossman

A Geometric Modeling System for Automated Mechanical Assembly

Very high level languages for describing mechanical assembly require a representation of the geometric and physical properties of 3-D objects including parts, tools, and the assembler itself. This paper describes a geometric modeling system that generates a data base in which objects and assemblies are represented by nodes in a graph structure. The edges of the graph represent relationships among objects such as part-of, attachment, constraint, *and* assembly. *The nodes also store positional relationships between objects and physical properties such as material type. The user designs objects by combining positive and negative parameterized primitive volumes, for example, cubes and cones, which are represented internally as polyhedra. The data base is built by invoking a procedural representation of the primitive volumes, which generates vertex, edge, and surface lists of instances of the volumes. Several applications in the automatic assembly domain have been implemented using the geometric modeling system as a basis.*

1. Introduction

This paper describes a computer-based system for modeling three-dimensional (3-D) objects. The initial motivation for this research was to create models for a very high level programming language for mechanical assemblers. The result has evolved into a general purpose system for describing parts both in terms of their three-dimensional geometry and their mechanical properties.

• *The task domain: mechanical assembly*

A general purpose manipulator consists of a gripper mechanism whose position and orientation are under computer control. When such a device is coupled with sensory feedback, such as force and touch sensing, it can perform complex assembly tasks. However, the usefulness of manipulators may be limited by the fact that programming them in terms of joint angles (motor positions), motor torques, raw sensor values, etc., is difficult. It would be preferable to specify assembly operations in terms of their intended effect on the parts and the tools in the workspace. These high level descriptions must then be translated into machine level manipulator commands. However, to do this, the user must be able to describe the parts, tools, and desired assembly relationships.

A design for a high level, assembly directed programming system, AUTOPASS (AUTOmated Parts ASsembly System) is described in [1]. The assembly world of AUTOPASS is quite complex. It involves both objects and operations that may be intricate and, when interpreted by humans, may require extensive experiential knowledge (*e.g.*, use of tools, or how parts fit together). The AUTOPASS system is intended to have a compiler that transforms task level statements into programs that can be executed by a mechanical assembler. AUTOPASS, as it compiles an assembly program, uses a 3-D *world model* to simulate the state of the world. Thus, the model records the changing 3-D locations and orientations of parts and of the mechanical assembler itself. In order to perform certain operations, such as determining how to grip an object or planning a collision-free movement, the compiler requires the 3-D physical characteristics of the world, such as the shape and weight of parts.

The AUTOPASS user refers to "assembly operands" by symbolic names, such as *bracket* or *fixture*, which are encoded in the 3-D model. Names may be associated with objects and sub-objects, and also with geometric elements of objects—edges, surfaces, and points. In order to determine, for example, whether an object can be moved and which other objects will move with it, the model includes various relationships among objects (such as "A *is attached to* B").

• *Related work in computer geometric modeling*

The following discussion, although not comprehensive, is intended to illustrate the scope of work in this area. Early applications of computer geometric processing arose with the introduction of numerically controlled (NC) machine tools. Several computer languages exist for the production of instruction sequences for NC machine tools. One of the most widely used languages for this purpose is APT (Automatically Programmed Tools) [2], which allows the programmer to describe continuous 3-D tool paths for machining the part. Thus, the APT model does not explicitly contain any geometric structure for parts. A number of modeling systems based on polyhedra have been developed. Typical of these systems is GEOMED [3], with which the user creates an object by describing a cross-section and moving this "wire frame" through space. Algorithms for merging and intersecting the polyhedra are also provided. Another form of modeling program is based on the combination of volume primitives, as exemplified by BUILD [4]. BUILD's six primitive volume types may be merged using a constructive geometry approach to generate representations of greater complexity, and merging can be performed on the results of previous merges.

The Part and Assembly Description Language (PADL [5]) is an example of a constructive geometry programming language which provides procedures for defining parts in terms of a small number of primitives. The PADL interpreter translates these procedural object definitions into object representations for later processing. PADL provides set operations on volumes and thus allows the user to create hierarchically structured descriptions of objects. The current version of PADL has restrictions on component orientation and type, thus limiting the repertoire of the resulting objects. A more general procedural part description system is the method originated by Grossman [6]. In this system generalized semantics may be attached to object descriptions, *e.g.*, all objects that are in the class *part* will have their mass calculated and stored. Object descriptions are hierarchical calls to procedures that are trapped by an interpretive system, which then applies the appropriate semantic routines. A procedural representation based on Grossman's work is used in the system described here to create an initial geometric description.

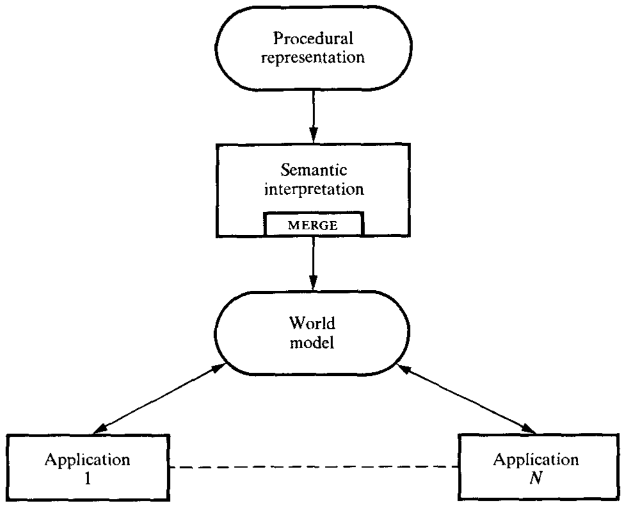

Figure 1 GDP system structure.

• *Overview of current work*

To address the needs of the mechanical assembly task domain, the geometric modeling system described here was developed. The components of the system, called GDP (for Geometric Design Processor), are illustrated in Fig. 1.

At the center of the system is the *world model*, a graph-structured data base which describes the structure and relationships of 3-D objects. The data base is initially created by invoking system- and user-supplied *procedural descriptions* in which instances of parts and sub-parts are represented by invocations of *object procedures*. The interpretation of object procedures by system-supplied semantic routines results in the construction of the world model. The MERGE algorithm, which computes the point set operations of union, intersection, and difference for a pair of arbitrary polyhedra, plays an important role in interpreting the object-defining procedures to produce the world model. Various applications have been developed which access the world model.

2. The world model

The world model is represented as a graph-structure in which each vertex represents a volumetric entity—a *part*, *sub-part*, or *assembly*, and the edges are directed and labeled to indicate four kinds of relationships: *part-of*, *at-*

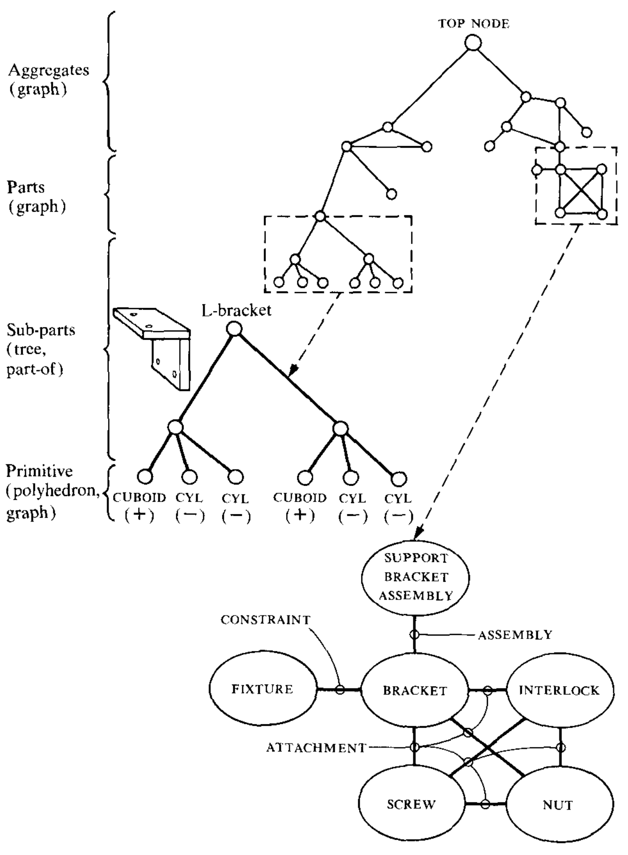

Figure 2 World model data structure.

tachment, *constraint*, and *assembly-component*. Figure 2 shows an example of a graph-structured world model.

An *object vertex* may represent one of three types of entities:

- A three-dimensional *part* (*e.g.*, a machine screw or L-bracket), whose shape is fixed but whose position and orientation may be changed dynamically.
- A *sub-part* (*e.g.*, a screw-head or bore-hole), a volumetric entity out of which parts are composed.
- An *aggregate* (*e.g.*, a piston and crankshaft or a model of a manipulator), whose constituent assemblies or parts bear a fixed functional relationship but whose internal geometry can change.

The role of a particular object vertex is determined by its context in the world model structure and the value of its volume class attribute (described below). The attributes of a node are as follows:

Name A symbolic name provided by the user.

Class A list of classes to which the object belongs. These classes provide semantic information to the various planning components of the manipulator language compiler. Examples of object classes are: general object, fastener, assembly, hole, rotor, and locking device. In the example, the *L-bracket* is a nonprimitive, general object (see below for a discussion of primitive and nonprimitive). Some of these classes are further qualified by type and may have associated geometric information. An example is *fastener*, which may have type qualifiers such as *clip, bolt, nut*, etc.

Volume class Defines the interpretation of the object vertex: A value of PART means this vertex is a part, in the sense described above. A value of AGGREGATE means this vertex is an assembly or aggregate. Values of SOLID or HOLE indicate that this vertex is a sub-part which adds or subtracts a volumetric component to or from the part or sub-part to which it belongs. (See below for a description of how positive and negative volumes are combined.)

Geometric description The object's shape, represented as a polyhedron. The polyhedral description is a set of point, edge, and surface list structures (Fig. 3) accessed by a pointer at the object vertex. The individual elements of the polyhedron may be accessed, thus allowing the user to refer to *spatial features*. The choice of representing shapes in terms of polyhedra results in closure under the set operations of union, intersection, and difference. This property provides a uniform geometric representation of primitive volumes, complex objects, and assemblies. The polyhedra are automatically generated by the MERGE algorithm, described below.

Coordinate transformation A 4 × 4 homogeneous transformation matrix, which, when applied to a point in the object's coordinate frame, yields the point's coordinates in a world reference frame.

Physical properties For example, material and weight, used in such calculations as center of gravity and object stability.

Primitive parameters If the object vertex represents an instance of one of the primitive volumes (see below), the arguments passed to the primitive object procedure are stored in the object vertex. For example, for a CUBOID, the X, Y, and Z side-lengths would be stored. This is particularly useful in cases where the polyhedral representation only approximates the "ideal" primitive volume (*e.g.*, CYLINDER).

An object vertex also contains a list of pointers to relational descriptors. These descriptors define physical relationships that the object has with other objects in the world model. The relationships provided are:

Part-of The part-of relation represents the logical containment of one object in another; it has several interpretations:

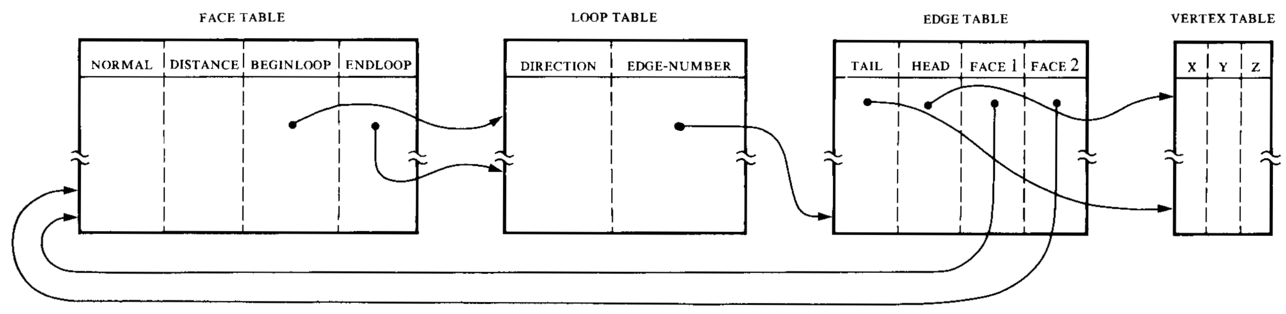

Figure 3 Data structure of a GDP polyhedron.

- A *part* may be *part-of* an *aggregate*. An *aggregate* can also be *part-of* another *aggregate*.
- A *sub-part* (represented by an object vertex with volume class SOLID or HOLE) may be *part-of* a *part* or of another *sub-part*.

The part-of relation induces a tree structure on the world model, as illustrated in Fig. 2. The system-supplied aggregate vertex TOP_NODE serves as the root of the part-of tree.

Other relations in the world model are used to represent mechanical and functional links among aggregates, parts, and sub-parts and are also shown in Fig. 2.

Attachment There are three types of *attachment* relationships: *rigid, nonrigid*, and *conditional*. Each descriptor indicates the type of attachment, the two objects related, and relative coordinate transforms between the two objects. In addition, nonrigid and conditional attachment descriptors contain information to qualify the particular relation.

Rigid attachment occurs when no relative motion is possible between the frames of objects; the relationship may have a force qualifier that defines ranges of thrusts and torques over which the rigid attachment holds.

Nonrigid attachment occurs when objects cannot be separated by an arbitrarily large distance but relative motion between their frames is possible. Nonrigid attachments are the basis of the system's model of mechanisms and are classified on the basis of the type of mechanical joint they provide. The representation of nonrigid attachment enables the system to analyze the kinematics of mechanisms. At present two joint types, linear and rotational, are permitted, though this may be extended later to include other types such as ball joint; both types may be qualified with force and spatial limits.

Conditional attachment is the representation of objects being supported by gravity (but not strictly attached). The relationship enables the system to move a part supporting another part. Qualifiers in the command define the range of orientations over which the support relationships will hold, so that, for example, the supported part is not allowed to fall off. These qualifiers form a condition list, each element of which is a spatial position, orientation, or velocity condition expressed parametrically as a linear inequality. The intent is that the attachment holds only as long as the constraints are met.

Constraint relationships Represent physical constraints of one object on another. A constraint is described by a type, a direction vector, a force threshold, and a pointer to the constraining object. A *translational constraint* specifies that the object cannot move in the given direction unless the force threshold is exceeded. A *rotational constraint* describes limits of rotation of an object about a given vector. Constraint relationships need to be updated whenever an object is moved to reflect newly created or removed physical constraints resulting from that movement.

Assembly component A type of object called an *assembly* has special meaning in the world model. It may be dynamically created during high level language compilation and represents a set of objects which is closed under *attachment* and *assembly*. An assembly vertex is created when an assembly naming statement is encountered in an AUTOPASS program [1]. This type of statement assigns a name and a coordinate frame to a vertex and establishes the *assembly-component* relationship between that vertex and an object (specified in the statement). The programmer is thus dynamically declaring a new structure that can then be used like any other object in subsequent statements. Such an object does not have a polyhedral representation of its own but implicitly acquires the descriptions of the (transi-

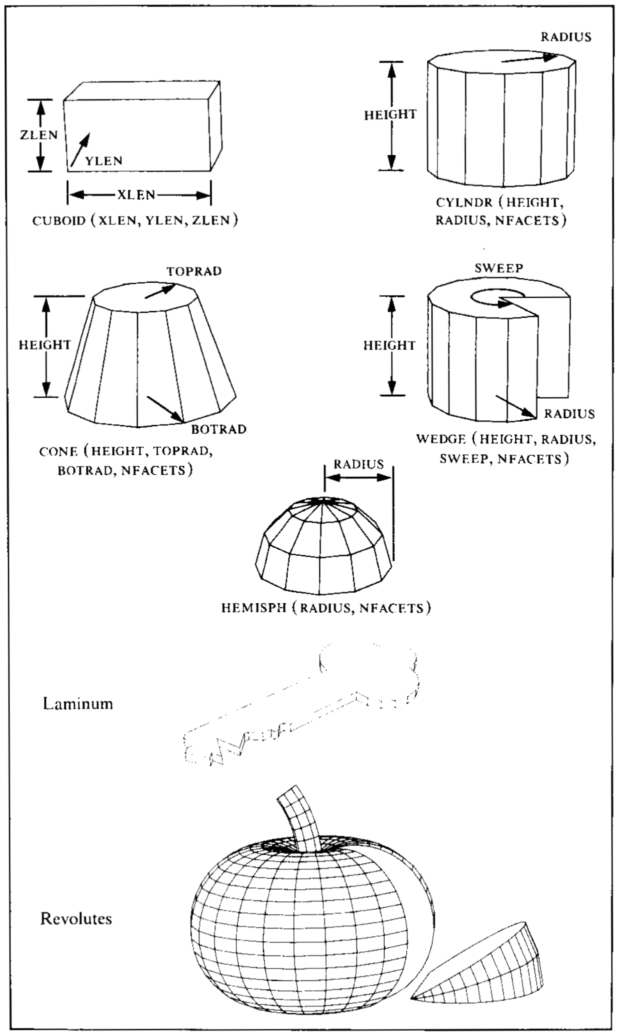
RADIUS
HEIGHT
ZLEN
YLEN
XLEN
CUBOID (XLEN, YLEN, ZLEN)
CYLNDR (HEIGHT, RADIUS, NFACETS)
TOPRAD
SWEEP
HEIGHT
HEIGHT
BOTRAD
RADIUS
CONE (HEIGHT, TOPRAD, BOTRAD, NFACETS)
WEDGE (HEIGHT, RADIUS, SWEEP, NFACETS)
RADIUS
HEMISPH (RADIUS, NFACETS)
Laminum
Revolutes

Figure 4 Primitive objects.

tively) attached chain of objects. Thus, an *assembly* allows grouping of assembled parts under one name for future reference.

3. Creating world models

In addition to supporting the world model data base described above, GDP provides a facility to allow the user to create such world models by executing *procedural representations* of aggregates, parts, and sub-parts.

• *Hierarchical object descriptions*

The specification which is used to create a world model reflects the hierarchical structure defined by the *part-of* relation in the model itself. The user defines aggregates by listing the parts and aggregates from which they are composed. Similarly, parts (and sub-parts) are defined by grouping together collections of sub-parts.

Several other features are required to produce a complete description of the world model: First, the user must be able to define the geometrical relationships among a set of entities which are *part-of* another entity. This amounts to being able to define the positions and orientations of those sub-entities in some suitable coordinate frame. Second, it is necessary to specify more exactly the relation between an entity and another entity which it is *part-of.* Specifically, the user must be able to indicate that one entity is a *part* and is *part-of* an *aggregate*. Also the user must be able to indicate that an entity is a *sub-part* which is *part-of* a *part* (or another *sub-part*) and, in addition, whether the *sub-part* contributes a positive (SOLID) or negative (HOLE) volume to its super-part. Complex shapes can be formed by "gluing together" simpler shapes (formally, taking the union of polyhedral point sets). Conversely, one can use negative sub-parts (HOLES) as "machine tools" that can cut and drill sections out of parts (taking the set difference of polyhedra). This ability greatly reduces the burden of describing all the points, lines, and surfaces which define a complex part's polyhedral representation; it is possible because of the MERGE algorithm for combining general polyhedra (see below).

The final feature needed to complete the hierarchical description process is a basis set of objects from which others can be defined. GDP supplies a set of seven *primitive objects* (see Fig. 4). The primitive objects are "parameterized" in the sense that the user may invoke instances which vary in size, aspect ratio, fineness of polyhedral approximation (for curved objects), etc. Because the user-supplied object descriptions use the same procedural representation as the system-supplied primitive objects, the user can effectively "customize" and extend the set of primitives.

• *Defining objects with procedures*

Following the work of Grossman [2], GDP implements the hierarchical object descriptions in the form of a procedural representation, in which aggregates, parts, and subparts are represented by *object procedures*. An object procedure represents a "template" for an object. An activation of an object procedure corresponds to creating an *instance* of the object in 3-D space. An object procedure may have parameters, which change the form of the *instance* of the object resulting from an activation of that procedure. Such a procedure may represent a generic object; for example:

```
SCREW: PROCEDURE (LENGTH,DIAMETER,
  THREADSPERINCH,NSECTORS);
  /* OBJECT PROCEDURE DEFINING A MACHINE SCREW.
       NSECTORS IS THE NUMBER OF FACETS IN THE POLY-
       HEDRON APPROXIMATION OF THE CYLINDER THAT
       MODELS THE SHAFT OF THE SCREW.  */
  DECLARE (DIAMETER,HEADRADIUS,HEADTHICKNESS,
    LENGTH,NSECTORS SLOTDEPTH,
    SLOTTHICKNESS,THREADSPERINCH) FLOAT;
  CALL SOLID(SHAFT,'SHAFT',LENGTH,DIAMETER,
    THREADSPERINCH,NSECTORS);
  CALL ZTRAN(LENGTH);
  HEADTHICKNESS=DIAMETER;
  HEADRADIUS=DIAMETER;
  CALL SOLID(CYLNDR,'HEAD',HEADTHICKNESS,
    HEADRADIUS,NSECTORS);
  SLOTDEPTH=0.4*HEADTHICKNESS;
  SLOTTHICKNESS=0.5*SLOTDEPTH;
  CALL XYZTRAN(-HEADRADIUS,-SLOTTHICKNESS/2.0,
    HEADTHICKNESS-SLOTDEPTH);
  CALL HOLE(CUBOID,'SLOT',2.0*HEADRADIUS,
    SLOTTHICKNESS,SLOTDEPTH);
END SCREW;
```

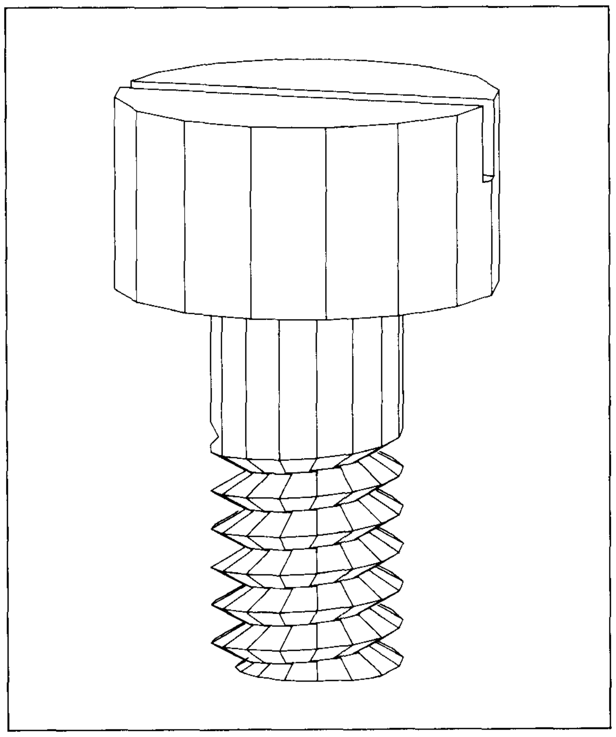
Figure 5 Result of procedure SCREW.

```
      ⎧   HOLE    ⎫  ⎧ (CUBOID,``name``,XLEN,YLEN,ZLEN);
      ⎪           ⎪  ⎪ (CYLNDR,``name``,HEIGHT,RADIUS,NFACETS);
      ⎪   SOLID   ⎪  ⎪ (WEDGE,``name``,HEIGHT,RADIUS,SWEEP,NFACETS);
CALL  ⎨           ⎬  ⎨ (CONE,``name``,HEIGHT,TOPRAD,BOTRAD,NFACETS);
      ⎪   PART    ⎪  ⎪ (HEMISPH,``name``,RADIUS,NFACETS);
      ⎪           ⎪  ⎪ (LAMINUM,``name``,DATA-ARRAY);
      ⎩ AGGREGATE ⎭  ⎪ (REVOLUTE,``name``,DATA-ARRAY);
                     ⎩ (user-supplied object procedure,parameter-list);
```

Figure 6 Primitive object procedures.

Activating this procedure with parameters (0.5, 0.25, 20, 16) will create an instance of a screw with length 0.5, diameter 0.25, 20 threads per inch, and, in the internal representation, will approximate the curved portions of the screw by 16 facets (Fig. 5). Object procedures may call other object procedures, which corresponds to building an object up out of sub-objects, as described above. In the machine screw example the threaded shaft of the screw is defined and generated by another procedure SHAFT (not shown here). Object procedures are always called through one of four "generic" procedures: Calls to PART and AGGREGATE generate PART and AGGREGATE object vertices; calls to HOLE or SOLID generate object vertices corresponding to positive and negative sub-parts. In the example the slot in the head of the screw is formed by subtraction of the primitive CUBOID, in the statement that begins "CALL HOLE(CUBOID . . . ." The actual parameters used in creating a primitive are always saved in the data structure generated by the semantic processor. By use of addition and subtraction operators on pairs of volumes (primitive or not), very complex objects may be realized. While not all possible shapes can be realized, the domain of parts for mechanical assembly and most other tasks can be adequately approximated. (The set of primitive object procedures is listed in Fig. 6.)

So far, objects have been described only by their shape; their locations and orientations in 3-D space must also be established. Associated with the object procedure interpreter is a *position cursor*, a current reference frame having a 3-D location and orientation. Any activation of an object procedure causes the reference frame to be passed to it. The procedure defines the object with respect to the reference frame. The user can manipulate, store, and retrieve the position cursor through a number of PL/I procedures supplied with the system. In the example the call to ZTRAN moves the reference frame from one end of SHAFT to the end where the HEAD is to be positioned. (The frame-manipulation and other auxiliary procedures are listed in Fig. 7.)

In addition to calling other object- or frame-manipulation procedures, an object procedure can perform arbitrary computations to the limit of the base language (PL/I). This is illustrated by the line

SLOTDEPTH=0.4*HEADTHICKNESS;

CALL $\left\{\begin{matrix}\text{X}\\ \text{Y}\\ \text{Z}\\ \text{XYZ}\end{matrix}\right\}$TRAN (DIST); /*Translate position cursor*/

CALL $\left\{\begin{matrix}\text{X}\\ \text{Y}\\ \text{Z}\end{matrix}\right\}$ROT (ANGLE); /*Rotate position cursor*/

CALL $\left\{\begin{matrix}\text{X}\\ \text{Y}\\ \text{Z}\end{matrix}\right\}$MIRROR; /*Reflect position cursor about axis*/

CALL $\left\{\begin{matrix}\text{STORE}\\ \text{RECALL}\end{matrix}\right\}$(COORDINATE-FRAME);

Figure 7 Auxiliary world model definition procedures.

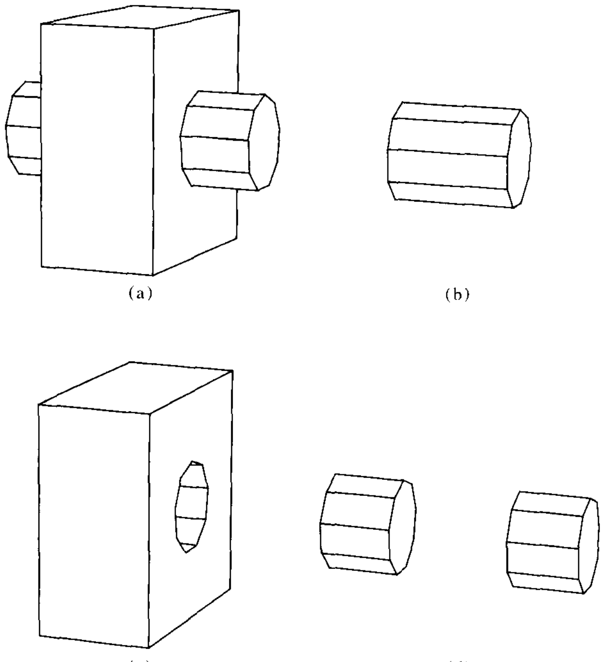

Figure 8 MERGE operations: (a) union; (b) intersection; (c) and (d) difference.

This, together with the ability to pass parameters between object procedures, greatly increases the power and flexibility of object definitions.

• *Interpreting procedural descriptions*
The procedural representation is converted into a world model by compiling and executing the object procedures in a context where the GDP primitives are defined. Executing the object procedures builds the world model data structure as follows:

Any invocation of one of the generic object procedures (HOLE, SOLID, PART, or AGGREGATE) results in the allocation of an object vertex in the world model. The volume-class for that vertex is set as specified by the called procedure (HOLE, SOLID, etc.). If a name for the object vertex was supplied, it is stored in the vertex. If the "called object" is one of the GDP-supplied primitives, a polyhedral representation of the primitive object, based on the object parameters and the position cursor, is bound to the object vertex.

In the case of nonprimitives, the user-supplied object procedure (*e.g.*, SCREW) is then invoked; usually it will invoke other user-supplied or primitive object procedures. Except for the case of *aggregates*, the next step is to create a polyhedral representation of the nonprimitive part or sub-part. To do this, it is necessary to combine the polyhedral representations of the sub-parts according to their specified polarity (HOLE or SOLID). This task is performed by the MERGE algorithm, which realizes the complete range of set operations on arbitrary polyhedra. The algorithm takes two polyhedra, described by lists of points, lines, and surfaces, and yields a new polyhedron which is either the union, intersection, or difference of its arguments (see Fig. 8 for definitions of these operations).

As the procedural representations are interpreted, successive applications of the MERGE algorithm are used to build up quite complex shapes, as illustrated in Fig. 9. The MERGE algorithm can also be invoked directly in GDP, allowing the world model to be altered and subsequently re-merged.

4. Applications of GDP
This section briefly describes several of the applications that have been developed based on the world models supported by GDP.

• *Interactive scene composition and analysis*
In addition to the execution of procedural descriptions of objects to produce world models, an extension to GDP allows interactive scene composition and analysis. Scene composition enables separately generated world models to be retrieved from a file system and edited into a single world model; objects and sub-objects may be moved between world model vertices, may be translated and rotated, and the MERGE algorithm may be re-applied. Such interactively generated composite scenes may be displayed in perspective views with hidden lines suppressed [7] on a storage-tube display. The edited models may be saved in the file system for subsequent use.

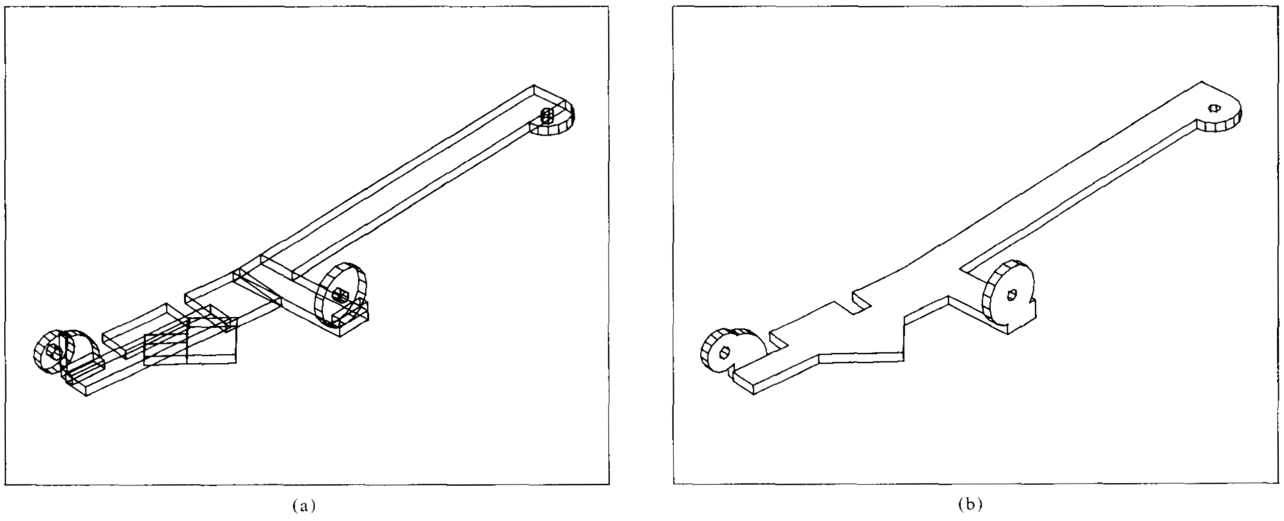

Figure 9 MERGE example: (a) components before merging; (b) resulting object.

In the world model, names can be supplied not only for aggregates, parts, and sub-parts, but also to *spatial features* (surfaces, edges, and points). The interactive scene composer allows the user to identify a feature by pointing to it with the graphics console cursor. The program then matches the 2-D coordinates to the nearest corresponding feature in the 3-D polyhedron. A name is then supplied and associated with the feature in the polyhedral descriptor. The features to which the user has access are only those of the internal polyhedral representation, *i.e.*, the vertices, straight lines, and flat surfaces which in many cases are the approximations of curved entities.

• *Path planning*

One of the fundamental problems in manipulator programming is how to find collision-free paths for a manipulator and the object it is holding through a cluttered environment. A new method for doing this is described in [8]. The technique involves modeling the objects in the workspace as polyhedra (using GDP), "shrinking" the manipulator model to a point in its articulation or motor space while concurrently "growing" the obstacles, and then performing a search through the visibility graph of the grown obstacles' vertices.

Figure 10 shows side and top views of a manipulator workspace with several obstacles. In the side view the manipulator gripper is shown, and in both views the computed trajectory of the tip of the gripper fingers is also shown. The particular problem involves moving a part from its "pallet" position at the left to a new position and orientation in a fixture at the right. The wall-like obstacles are arranged to exercise the algorithm; the larger blocks are high enough so that the manipulator cannot go directly over them. Note also that the object being carried is a complex part model itself and that its geometry is accounted for in the path calculation.

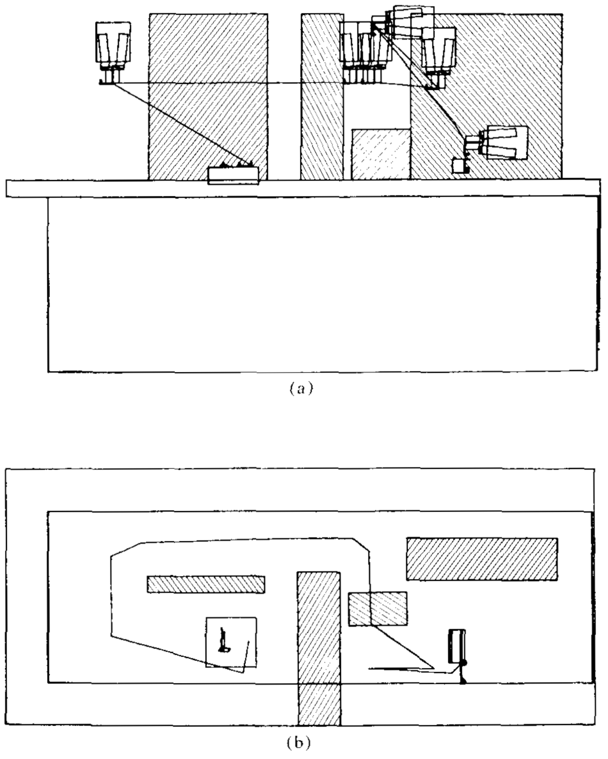

Figure 10 Path-finding application: (a) side view of calculated trajectory; (b) top view of trajectory (gripper not shown).

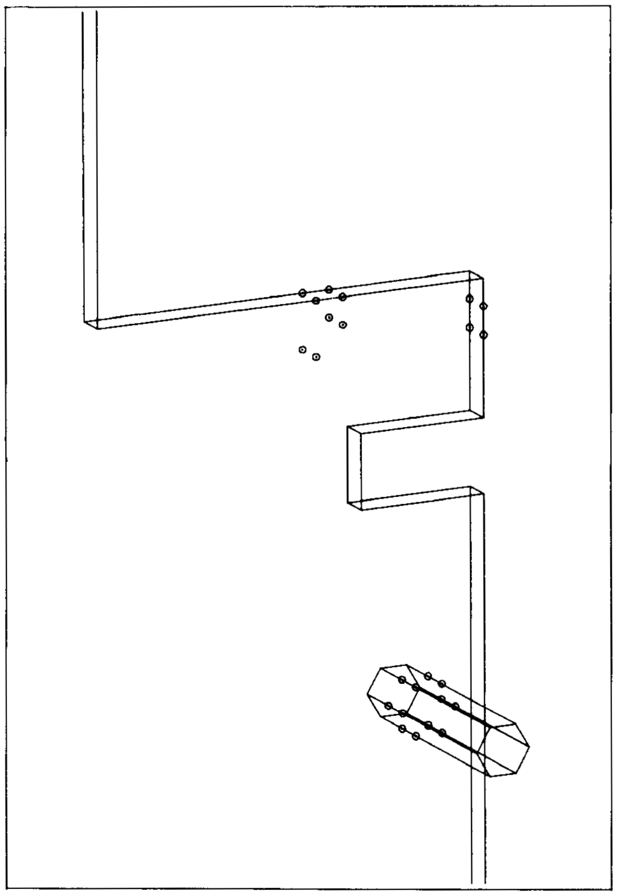

Figure 11 Interference-checking application: detail of interferences at one corner of mounting plate (also showing one of the studs as an object of interference).

• *Interference checking*
Another application, that of mechanical design verification, makes use of an interference checking facility of GDP. Intersections of object edges with other objects can easily be detected and shown graphically. (Note that such edge-face intersection reporting is one of the basic calculations done in the polyhedron merging routine.) Figure 11 shows one corner of a mounting plate to be attached to a frame (not shown) with mounting studs that are intended to fit into slots. In this GDP drawing interferences between objects are shown by small circles; a mounting stud is shown out of place (it should be positioned in the slot), and the locations of other interferences with the frame around the plate are indicated above the slot. In addition to such static interference checking, dynamic interference checking has been implemented using the algorithm of Boyse [9].

• *Applications to machine vision*
As an example of the use of polyhedral models in vision, programs have been written to determine the stable orientations of parts on a plane and then to generate the 2-D outlines and extract features from them [10]. The program STABLE takes as input a polyhedron and returns a list of support planes defining "stable" orientations of that polyhedron. The method used first calculates the convex hull of the object by applying a "gift-wrapping" algorithm to the polyhedron's vertices. Figure 12(b) shows the convex hull computed for the part shown in Fig. 12(a). Next, the object's center of mass is found by integrating the volume formed by the prism under each polyhedral face edgeloop, and appropriately combining the volumes. The center of mass is projected onto each of the convex hull faces, thus hypothesizing that the face is a stable support plane. If the projected point lies within the bounds of the face polygon, then it is tentatively declared a stable plane. A test is made for the degree of stability by calculating the energy needed to "tip" the object by rotating over the edge in the support plane nearest the projected center of mass. If the energy is large enough, *i.e.*, exceeds a user-set threshold, then the convex face is a support plane. All the faces are checked and those passing the tests are ranked in decreasing order of stability (*i.e.*, "tipping" energy) and returned in a list to the calling program. Figure 13 shows the stable positions found for the part.

The object model can be manipulated so that it rests on any of the stable support faces of the convex hull [11]. An imaginary camera viewpoint is set to simulate the real camera's placement in the workstation, and a program then calculates the outer boundary of the object's projection into the display plane. The vector list may be displayed or passed to application programs for feature analysis. Figure 14 shows the outlines for each of the three stable orientations of the interlock as viewed from directly above a block platform.

5. Summary
The geometric modeling system GDP presented above satisfies the functional and user requirements of the Automated Parts Assembly System (AUTOPASS) language. It provides a structure for representing arbitrarily complex objects in terms of primitive objects and for representing assemblies as relationships among objects. A means of creating object models using procedural descriptions of parts and sub-parts has been developed. The original direction of this project, and the form of the world model produced by GDP, were suggested by research in programming mechanical assemblers. Several applications in this domain have been implemented, based on GDP; however, the authors believe that GDP has evolved into a general system for modeling 3-D objects. The significant features of GDP are:

1. The structure of the world model.

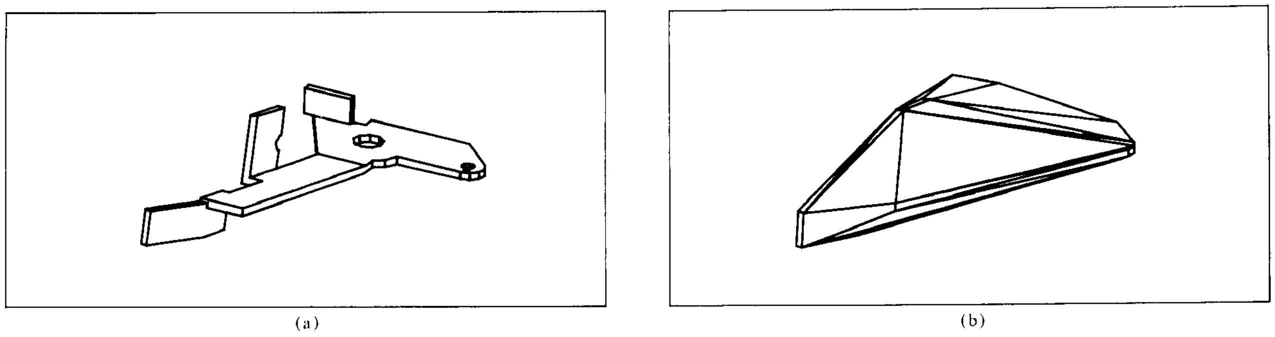

Figure 12 Machine vision application: (a) model of a typical part; (b) convex hull of the part shown in (a).

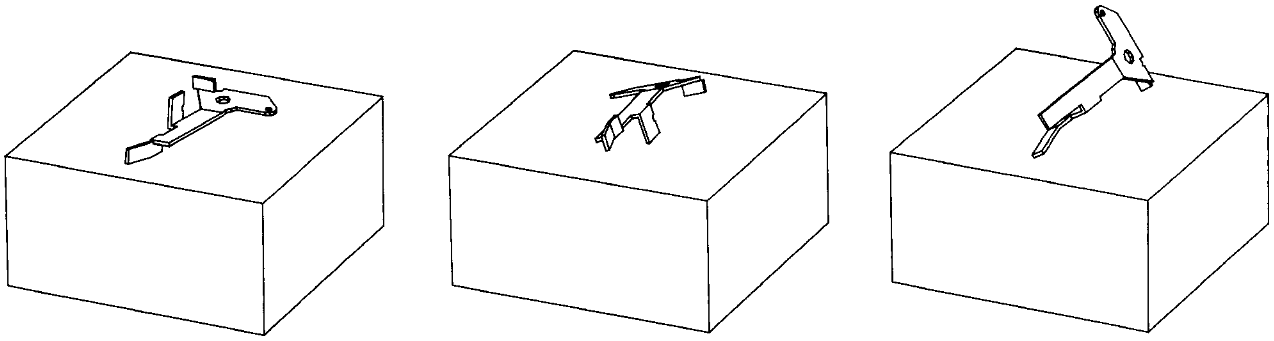

Figure 13 Machine vision example: computed stable orientations of the part.

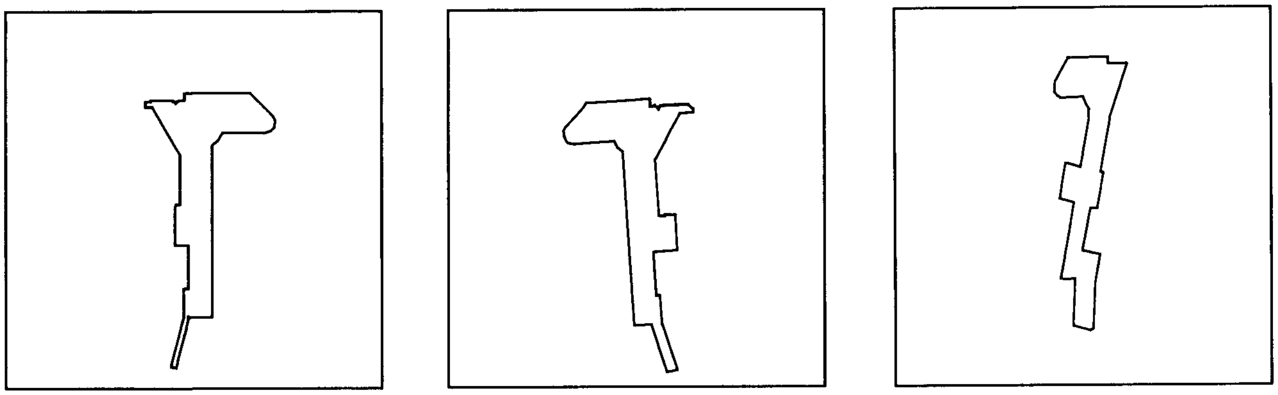

Figure 14 Machine vision example: computed outlines for each of the orientations of Fig. 13, viewed from directly "overhead."

2. A procedural representation for initial specification of object shapes and locations.
3. A semantic interpretation phase that produces polyhedral representations of described objects.
4. A procedure for merging general polyhedra.

GDP has been implemented as a set of PL/I, FORTRAN, and BAL programs that run on a System 370/168 under the VM/CMS operating system; interactive graphics facilities are provided by a storage tube graphics terminal.

Acknowledgments
The authors wish to thank Peter Will for inspiring this work, Roger Evans for his collaboration on the merging algorithm, and Anna Bruss for generating the revolute model of Fig. 3.

References

1. L. I. Lieberman and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM J. Res. Develop.* **21,** 321-333 (1977).
2. W. H. P. Leslie, Ed., *Numerical Control Users' Handbook*, McGraw-Hill Book Co., Inc., New York, 1970.
3. B. G. Baumgart, "Geometric Modeling for Computer Vision," *Stanford Artificial Intelligence Laboratory Memo AIM-249*, Stanford University, Stanford, CA, October 1974.
4. I. C. Braid, *Designing With Volumes*, Cantab Press, Cambridge, England, 1974.
5. "An Introduction to PADL," *Production Automation Project Technical Memorandum 22*, University of Rochester, Rochester, NY, December 1974.
6. D. D. Grossman, "Procedural Representation of Three-dimensional Objects," *IBM J. Res. Develop.* **20,** 582-589 (1976).
7. A. Appel, "The Notion of Quantitative Invisibility and the Machine Rendering of Solids," *Proc. 22nd ACM Nat. Conf.*, 1967, pp. 387-393.
8. T. Lozano-Pérez and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Commun. ACM* **22**, 560-570 (1979).
9. J. W. Boyse, "Interference Detection Among Solids and Surfaces," *Commun. ACM* **22,** 3-9 (1979).
10. L. I. Lieberman, "Model Driven Vision for Industrial Automation," presented at the IBM International Symposium on Advances in Digital Image Processing, Bad Neuenahr, Germany, Sept. 26-28, 1978.
11. L. I. Lieberman, D. D. Grossman, M. A. Lavin, T. Lozano-Pérez, and M. A. Wesley, "Three Dimensional Modeling for Automated Mechanical Assembly," presented at the NSF Workshop on 3-Dimensional Object Representation and Description, Philadelphia, PA, May 1979.

Received November 10, 1978; revised July 2, 1979

M. A. Wesley, L. I. Lieberman, M. A. Lavin, and D. D. Grossman are located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598; T. Lozano-Pérez is located at the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.