Constraints in Constructive Solid Geometry

Jaroslaw R. Rossignac

Design Automation Science Project
Manufacturing Research Department
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598.

ABSTRACT

The success of solid modelling in industrial design depends on facilities for specifying and editing parameterized models of solids through user-friendly interaction with a graphical front-end. Systems based on a dual representation, which combines Constructive Solid Geometry (CSG) and Boundary representation (BRep), seem most suitable for modelling mechanical parts. Typically they accept a CSG-compatible input (Boolean combinations of solid primitives) and offer facilities for parameterizing and editing part definitions. The user need not specify the topology of the boundary, but often has to solve three-dimensional trigonometric problems to compute the parameters of rigid motions that specify the positions of primitive solids.

A front-end that automatically converts graphical input into rigid motions may be easily combined with boundary-oriented input, but its integration in dual systems usually complicates the editing process and limits the possibilities of parameterizing solid definitions. This report proposes a solution based on three main ideas: (1) enhance the semantics of CSG representations with rigid motions that operate on arbitrary collections of sub-solids regardless of their position in the CSG tree, (2) store rigid motions in terms of unevaluated constraints on graphically selected boundary features, (3) evaluate constraints independently, one at a time, in user-specified order. The third idea offers an alternative to known approaches, which convert all constraints into a large system of simultaneous equations to be solved by iterative numerical methods.

The resulting front-end is inadequate for solving problems where multiple constraints must be met simultaneously, but provides a powerful tool for specifying and interactively editing parameterized models of mechanical parts and mechanisms. The order in which constraints are evaluated may also be used as a language for specifying the sequence of assembly and set-up operations.

An implementation under way is based on the interpreter of a new object oriented programming language, enhanced with geometric classes. Constraint evaluation results in the activation of methods which compute rigid motions from surface information. The set of available methods may be extended by the users, and methods may be integrated in higher level functions whose algorithmic nature simplifies the treatment of degenerate cases. Graphic interaction is provided through a geometrical engine which lets the user manipulate shaded images produced efficiently from the CSG representation of solid models.

Keywords: Solid modelling, quadric surfaces, constraints, rigid motions, computer graphics.

INTRODUCTION

Different architectures of solid modellers allow to cope with a large scope of geometric domains which extend from cell-based approximations [1] to solids bounded by free-form surfaces [2]. The highly popular Boundary representations and Constructive Solid Geometry representations are often combined in a Dual representation [3,4] which offers a better ground for simple, robust, and efficient application algorithms. As solid modellers migrate from research and educational environments to designers desks, efficient and flexible specification schemes become essential.

Solid modellers based on boundary representations (BRep's) can be easily enhanced with interactive front-ends which support user-friendly specification and provide a "tool-box" of construction routines that would, for instance, position a line tangent to two circles [5], or automatically infer exact geometry from a rough sketch annotated with dimensions [6]. To eliminate the necessity for repeating the same construction each time, the model is edited, the system must store the construction, and not its effect. Attempts to capture user's intentions into a graph of constraints proved limited in domain, and generally result in large systems of simultaneous equations, solved by slow, iterative methods, which are not guaranteed to converge, and therefore are impractical for industrial solid modelling.

On the other hand, Constructive Solid Geometry (CSG) representations offer conveniences for editing and archiving solid models, for defining always valid parameterized generic parts, and for implementing application algorithms that are simpler and often more robust than their counterparts that work from boundary representations. Facilities for converting boundary specifications to CSG representations are currently restricted to sweeps of two dimensional areas bounded by lines and circles [7]. Such a domain is too limited for practical applications. Therefore, to achieve a consistent dual representation, solids must be specified in terms that can be automatically converted to CSG. A BRep is derived algorithmically. Unfortunately, specification becomes tedious when complex constructions, involving tangency conditions and dimensions, must be converted by the user into primary CSG parameters (positions and dimensions of primitives).

We propose to use graphic input for specifying CSG representations in terms of constraints between boundary elements. These specifications, and not their effects, are stored in the CSG graph in an unevaluated form, and can be archived together with the CSG representation. Interactive editing is particularly simple because changes to primitives or sub-trees do not require that constraints higher in the tree be specified again. Constraints are evaluated sequentially and converted into rigid motions, which apply to graphically selected boundary features. The sequential evaluation of constraints leads to a new approach to dimensioning and tolerancing problems, and provides a language for specifying assembly and set-up operations.

The report begins with a brief review of various alternatives in solid modelling and arguments the choice of a dual representation scheme based on CSG-compatible input and on natural quadric surfaces.

The second section extends the semantics of rigid motions in CSG trees. Because geometric specifications may be separated from Boolean expressions, rigid motions may apply to boundary features that need not be localized in the CSG hierarchical structure, but represent collections of primitives that have to be moved as a rigid body. The section discusses methods for graphical selection of boundary features, and indicates the advantages of specifying and storing rigid motions in terms of constraints on surfaces that bound primitive solids.

The third section discusses existing techniques for evaluating the final shape of the solid, and proposes a new approach based on sequential evaluation. Specific methods for converting tangency constraints between natural quadrics into rigid motions are indicated.

Section four describes the overall architecture and discusses current implementation efforts which are based on geometric classes developed in an object-oriented programming language, and on a new display technique for solids defined in CSG.

The final section addresses the current limitations and desirable extensions of the proposed scheme. It also describes possible applications to the specification of dimensions and assembly operations, and to the use artificial intelligence techniques for reasoning on geometric relations.

ALTERNATIVES IN SOLID MODELLING

Solid modellers are meant to produce pictures and compute mathematically well defined properties of real solids. The adequacy and mathematical precision of the results is impeded by several difficulties:

- Models represented in currently available systems correspond to ideal solids that are bounded by simple but perfect surfaces. They do not reflect the surface finish and tolerances of imperfect solids in the physical world [8].
- Robust algorithms for computing intersections of complex surfaces are not available [2], therefore complex curved faces in real world solids are often approximated by collections of simpler (often planar) faces.
- Closed-form solutions for certain well defined integral properties are not available or are too complex to be implemented; therefore they must be approximated [9].
- Symbol manipulation techniques are slow and limited [10], consequently most computation is carried numerically with floating-point approximations of real numbers and yields round-off errors that may lead to approximate and even inconsistent results.

Designers of geometric modellers seek a compromise between the accuracy of the approximations and the complexity of corresponding algorithms. For instance, the use of curved surfaces to exactly model fillets and blends of real parts makes it very difficult to compute integral properties accurately and prevents the use of symbol manipulation techniques to overcome round-off error problems. On the other hand, approximating fillets and blends by piecewise planar faces results in a serious lack of accuracy in volumetric calculations or stress analysis.

The geometric coverage of a solid modeller falls generally under the following categories:

- **Polyhedra:** Boundaries of solids may be approximated by piecewise planar facets. It follows that surface intersection problems are simple, but the number of faces is typically large, and the approximation may lead to results inconsistent with user specifications. For example, a cylindrical pin might not fit in a cylindrical hole of a larger radius, because the facetting of the two cylinders is not properly aligned.
- **Natural quadrics:** Most mechanical parts are bounded by subsets of planar, cylindrical, conical, or spherical surfaces, collectively called natural quadric surfaces. Closed form solution are available for parametric representation of curves of intersection between any two natural quadrics [11,10]. Vertices may be computed by efficient numeric methods [12]. This domain is the most popular for mechanical design and analysis and is often equated with the geometric coverage of CSG-based systems.
- **Canal surfaces:** Survey data [13] indicate that 99 percent of mechanical parts can be modelled exactly if one combines natural quadrics with the possibility of representing fillets and blends. Constant radius blends [14] are the most popular. They are bounded by canal surfaces, which are subsets of envelopes of spheres of constant radius, swept along a space curve [15]. For simple curves, canal surfaces reduce to cylinders and tori but are in general much more complex. Computing their intersections exactly is beyond the scope of current modelling technology. However, canal surfaces can be efficiently approximated with smooth, piecewise-toroidal surfaces, if the trajectory of the sphere is approximated by smooth, piecewise-circular curves [16].
- **Free-form surfaces:** In artistic applications and when modelled shapes are not tightly toleranced, sculptured surfaces (often piecewise bi-cubic) offer a large flexibility [17] and are well suited for interactive specification. The computation of their intersections remains a research issue [2]. On the other hand, several modelling systems are based on a specification of smooth complex shapes that do not require surface/surface intersection computation [18].
- **Procedurally-defined solids:** Great flexibility and generality may be achieved by providing the user with the possibility of writing his own functions or procedures that define the components of the desired solid model [19,20].
- **Cell decomposition:** When crude approximations are sufficient, solids may be modelled by collections of simple quasi-disjoint cells (often cubes). Algorithms that operate on such approximations are of a discrete nature and do not require the computation of surface/surface intersections [1].

Our work is focused on natural quadrics because they offer a good compromise between robust methods for intersection calculation and a geometric coverage that is adequate for industrial parts.

Application algorithms operate on one or several internal representations, which can be explicit or constructive. **Explicit** representations contain a list of solid or boundary elements.

- **Boundary representation:** Solids may be unambiguously represented by their boundary, which may be decomposed into faces. Each face may be represented by its supporting surface, and by its two-dimensional boundary in that surface. Such a 2-D boundary may be represented by edges and vertices. Since an edge is shared by several faces, and a vertex bounds several edges and lies on several faces, boundary representations usually eliminate information redundancy by separating the geometry of surfaces, curves, and vertices from the topology, represented by a graph, which indicates adjacency relations between faces, edges, and vertices [21]. In the case of polyhedral modellers, the geometric information may be limited to vertex coordinates, and topological information may be used to derive geometry of edges and faces. This representation is well suited for graphic applications [22].
- **Spatial enumeration and cell decomposition:** For approximate models, the world is partitioned into regular cells, and the representation is merely an indication of which cells are contained in the solid. Such an indication may be efficiently encoded in run-length vectors or octrees [1].
- **Primitive instancing:** Solids that correspond to unions of disjoint parameterized primitives at arbitrary positions and orientations may be represented explicitly by enumerating the types, parameter values, and locations of primitives. Solids' properties are generally obtained by combining properties of each primitive computed separately using parameterized formulae. Unfortunately, the scope of such solids is limited, even with elaborate primitives.

On the other hand, **constructive** representations provide a sequence of operations for transforming or combining primitive elements, but do not contain explicit descriptions of the solid they represent.

Paradoxically, a boundary representation may be stored in a constructive manner as a sequence of primitive Euler operations [23], which indicate how the topology of the final object may be constructed. The coordinates of vertices must also be provided. Euler operations [24,25] ensure a valid topology, but checking geometric information for validity and for compatibility with the topology is an expensive task.

CSG represents solids without carrying any information about their topology, which is derived algorithmically. The representation uses higher level operations to transform or combine primitives such as:

- **Half-spaces,** which are defined by algebraic inequalities, and provide equations for the bounding surfaces and a simple test for determining whether a point is inside or outside a half-space [26],
- **Primitive solids,** for instance parameterized blocks, spheres, and cylindrical or conical sections, which are usually defined as intersections of half-spaces, or by algorithmic procedures that compute a penalty function [20],
- **Sweeps of planar cross-sections,** which are usually limited to extrusions (translations) or rotations. If the cross-section is bounded by linear segments and circular arcs, the sweep can be expressed as a Boolean combination of half-spaces that are bounded by natural quadrics and tori, and may be converted to CSG [7].

Primitives mentioned above may be combined into complex solids by **Boolean operations** (union, intersection, and difference) which are also very powerful for modelling certain manufacturing processes such as machining. A regularized version of Boolean operators was introduced [26] to ensure that the result is a valid solid (is closed and contains no dangling faces, edges, or vertices), and that it may be used as argument of other Boolean operations. A Boolean expression may be encoded in a binary tree, where internal nodes correspond to Boolean operations, and leaves correspond to solid primitives. Such a representation is usually equated with Constructive Solid Geometry.

Individual subsolids may be transformed by rigid motions, and also by the following operations:

- **Blending and filleting** is usually viewed as a boundary modification (replacing an edge by a smooth transition between adjacent faces) [27]. However, blending may be elegantly incorporated in CSG by introducing a blending operator [14], which globally rounds all concave or convex edges of solids that correspond to intermediate nodes of the tree.
- **Offsetting operations** were introduced to enhance the power of CSG [28]. Sub-solids may be expanded or shrunk by arbitrary distances, and the results combined through Boolean operations or offset again. Offsetting operations are useful to model various manufacturing operations (etching, coating) and to model global blending and filleting.

- **Sweeps** provide a powerful operation for the analysis of mechanisms and for modelling machining operations. As a rigid motion operator takes a solid and a rigid motion and defines another solid, the sweeping operator takes a solid and a trajectory in six dimensions and defines another solid that corresponds to the region swept by the moving solid. Although the representation is simple to implement, existing algorithms for computing the boundaries (or other properties) of such swept solids are currently limited to trivial cases [29].

Certain application algorithms are simpler to design and implement in a robust manner with a CSG representation than with a BRep. For instance, a basic tool for most applications is a point membership classification algorithm which determines whether a point is inside or outside a solid [30]. Such algorithms may be very simply implemented in CSG by a divide and conquer paradigm. The point is classified with respect to primitive solids, and classification results are recursively combined up the tree. A similar algorithm for BRep's is more complex and typically requires casting a ray from the point to infinity, computing intersections of the ray with surfaces, classifying the intersection points with respect to faces by a 2-D version of the same algorithm, and inferring the result from local neighborhood information or by counting the parity of the number of intersections along the ray. Both CSG- and BRep-based algorithms need to deal with possible singularities, but reasonably robust CSG-based algorithms are simpler to design [31].

On the other hand, many application algorithms are based on explicit boundary information, therefore a combination of BRep and CSG is becoming increasingly popular [4]. Such a combination is called a **dual** representation. It allows application algorithms to work from the BRep, from CSG, or from both. However, to obtain consistent results, the two representations must be equivalent (specifically, the BRep may not be an approximation of a perfect-form CSG). A typical implementation of a dual representation scheme is limited to natural quadrics because intersections of more complex surfaces cannot be represented exactly in the BRep. The integration of offsetting and blending operators in dual representation modellers [32] is impeded by the difficulty of supporting exact intersections of canal surfaces. Therefore, we shall restrict our work to providing an interface for the specification of geometry in systems based on a dual representation, where the CSG contains only Boolean and rigid motion operators.

To maintain consistency in a dual modeller, one must be able to convert user input into both representations. Boundary evaluation algorithms for converting a CSG representation in terms of natural quadrics into a Brep are available [31]. However, Brep-to-CSG conversion algorithms are currently limited to 2D areas bounded by line segments and circular arcs [7]. Consequently all user input for Dual systems must be in a CSG-compatible form. This method has certain drawbacks, especially for user interaction, but offers serious advantages for industrial design, where definitions of parts must be parameterized and frequently edited. Constructive specifications combine or transform valid primitive solids by operations that preserve their validity. Only the geometry is specified. The connectivity is algorithmically derived by boundary evaluation.

The PADL-2 language [33] offers a typical example of a CSG input. The language can handle variables, arithmetic expressions, and some algorithmic statements, but is primarily descriptive and consists of a sequence of assignments that define nodes of a CSG tree in terms of Boolean combinations of other nodes, which may be transformed by rigid motions. Primitive solids (blocks, cylinders, half-cones, spheres, and tori) as well as rigid motions (rotations around the three axis of the global coordinate system, and translations along these axis) may be parameterized. The language offers facilities for storing parametric definitions in the form of generics that may be invoked in other definitions. Any sub-solid may be used at different places in the tree, and each instance may correspond to a different position and orientation.

The choice of primitives in a CSG specification of a given solid is, most of the time, clearly indicated by the nature of bounding surfaces. Boolean expressions may not be trivial, but usually can be produced after a small number of trials even for complex topologies. Specifications may involve redundancy of primitives or subtrees, but such redundancy is not a problem, and if necessary, can even be detected algorithmically [34]. The major difficulty remains the specification of the exact geometry and is the main thrust of this report.

In traditional CSG-oriented specification, solid primitives or subsolids can be arbitrarily positioned and oriented before they are combined by Boolean operations. Such positions and orientations are usually specified by a sequence of simple rigid motions (translations and rotations around the axes of the global coordinate system). In CSG, the rigid motions are stored as part of the tree, therefore editing is trivial because any rigid motion may be redefined by a simple statement. To show the effect of such a modification, a new picture is usually generated, either by expensive boundary evaluation, or by shading techniques that operate directly from CSG [35]. This trial-and-error process is the current technique for achieving a correct specification of the desired solid model. During this editing phase, a boundary feature (pocket, hole, or boss) may have to be moved as a whole. Correcting the situation in a traditional CSG structure may require the alteration of many internal nodes and rigid motions because the primitives that contribute to the feature may be arbitrarily distributed throughout the CSG tree. It may be impossible - without moving other primitives - to transform all selected primitives in one step, by applying a rigid motion to a single node of the tree. To overcome this limitation, we propose an extension of the semantics of the internal structure of CSG.

In a natural CSG representation a rigid motion M that applies to some internal node S may be stored as the right son of a node F, of which S is the left son. This configuration may be specified by a simple declarative statement of the form: $F = S$ **at** M. In the proposed extension, several rigid motions may be associated with the same internal node S, and each motion may operate on any set of sub-nodes of the tree spanned by S. For example, in the solid defined by $S = A + (B - C)$, the same motion M may be simultaneously applied to A and C, without altering B. This is represented by storing, with S, the motion M, and by indicating that it operates on the list of sub-nodes (A,C). Throughout the rest of this report, the term **boundary feature** will denote a collection of primitives instances or sub-nodes, selected by the user, and transformed collectively by a rigid motion.

Boundary features may not always be specified by simply providing the user-defined name of an internal node of the tree because certain internal nodes do not have user-defined names, and because, on the other hand, instances of sub-solids with the same name may appear at different places in the tree. To overcome the naming process, pathnames are used [36]. A pathname for a node or primitive instance N in a tree T is an efficient encoding (in terms of left or right branches) of the path from the root T to N. We use an enhanced version of pathnames, which allow to distinguish between the different faces of a primitive instance. Pathnames are a very unnatural way of selecting a node or a primitive. Interactive graphic input is much more attractive. Picking primitives graphically may be done on a picture of the evaluated solid or on a display of the individual primitives. Each one of these two methods has its own advantages and limitations, therefore both methods should be simultaneously available in a user-friendly system.

Given a picture (wire-frame or shaded image) of an object produced on a screen, the user may select faces graphically by positioning a cursor on top of the chosen face. Face determination may be done by casting a ray [37] from the eye through the cursor, and computing what faces of the solid are hit by the ray. With wire-frame displays, hidden faces may also be displayed and selected. Ambiguities occurring when several faces are hit by the ray are removed by letting the user circulate through all picked faces and select one. The point where the ray penetrates the selected face may lie on several primitives. Some of the primitives may be **inactive** in the region where the point is located, i.e., modifying the primitive in that region will not alter the resulting solid. Such configurations may be detected by classifying the intersection point with respect to the **active zone** [34] of each potentially selected primitive.

Geometrical engines let the user rotate the picture of an object in 3-D and translate the view point to penetrate inside an object, thus exposing its hidden faces. Using such a device, the user can access all faces of the solid without ambiguity. Picking faces on an evaluated model has the advantage that the user sees what is wrong and what needs to be modified. Geometrical engines with depth-buffer hardware offer such possibilities if they are provided with 3-D data that describe the boundary of the solid. Boundary evaluation is too expensive for interactive design. We use a technique, described in section five, to generate the required 3-D data directly from CSG.

The impossibility of selecting primitives that do not contribute to the boundary of the defined solid is a major drawback of a scheme where faces are picked from evaluated models. To allow the graphic selection of all primitives, whether their boundary contributes to the boundary of the solid or not, all primitive instances must be displayed. If the primitives are solid primitives, they can be displayed in a wire-frame manner, i.e. their edges and profile (or silhouette) curves are displayed in their entirety, regardless of their presence in the final solid. Such a display mode is usually available in most modellers. Picking primitives by ray casting is cumbersome, because usually too many primitives are hit by a single ray. Better results are obtained if for each primitive, one displays characteristic points and simple curves, which symbolically indicate the nature and orientation of the associated primitive. For example, to display a cylinder, Anderson [38] uses a line segment to indicate the axis, and circles to indicate the radius and extent of the cylinder. A characteristic point on the axis may be used to select the cylinder. In a polyhedral modeller, vertices of facetted approximations of primitives may be used for picking purposes. The above technique, based on characteristic points, seems even more appropriate if the primitives are half-spaces or sweeps. For example a plane has no profile edges and is intersected by most rays, therefore it cannot be displayed, nor distinguished by ray casting, from most other planes. A plane may be symbolized by a point and a normal, a cylinder by a circular cross-section and possibly the axis, a cone by its tip, axis and several linear generators, and finally a sphere by its center and three orthogonal great circles. A cross-sectional sweep can be symbolically displayed by its spine and a few cross-sections. The symbolic display of primitives offers a significant advantage, because it does not require the full power of a solid modeller, and therefore may be implemented in a front-end program that is independent of the modeller it interfaces. Anderson describes such a local implementation on a Macintosh [38].

We have discussed techniques for graphically selecting the primitives that contribute to a boundary feature. Let us now address the issue of how to specify the rigid motion that will operate on such a boundary feature. The traditional method of specifying translation coordinates and angles of rotation around the axis of the global coordinate system is very cumbersome and error-prone. A more user-friendly solution uses boundary elements and some intrinsic features of primitives (such as axis of cylinders) to specify a rigid motion. For instance the system should be able to align automatically the axis of two cylinders or compute the angle of a rotation that would bring one surface in contact with another. Anderson [38] describes an implementation based on previously mentioned characteristic points. Its functionality is restricted to aligning axes or edges and to making vertices coincident, but these functions seem nicely integrated with a graphical input.

Rigid motions, specified in terms of boundary elements, are usually stored in an evaluated form: i.e. the parameters of the motion (or the corresponding matrix) are derived from boundary information, and the specification itself is discarded. This approach leads to a painful editing process, because changes to boundary elements used for deriving a rigid motion will invalidate that motion. Usually, in such schemes, editing is done by a selective replay of a log file that contains the whole user-modeller interaction. Since boundary features are often specified graphically, a slight change of the geometry, produced at the beginning of a log file, will alter the meaning of the specification stored in the rest of the file. Very often the user must redefine most rigid motions. This process is even more difficult if the user does not remember how constraints were specified. The problem becomes critical in industrial environment, where the complexity of the models is large and time delays between changes to a design are important.

Similarly, storing evaluated rigid motions makes parameterization very difficult. Some efforts are underway at the University of Bologna [39], to convert boundary-defined constraints into parameterizations expressed in PADL-2. The remaining of this report describes a different approach.

Instead of storing the rigid motion computed from a constraint-based specification, one can store the specification expressed in terms of unevaluated constraints. This approach offers important advantages: editing is greatly simplified because constraints are explicitly available to the user, who can insert new constraints and alter or delete existing ones. Further more, altering a constraint does not require that any previously defined constraint be issued again. Parameterization of models is possible, because all constraints can be evaluated automatically each time a new set of parameters is provided.

EVALUATION OF CONSTRAINTS

The idea of using a graph of constraints to represent geometry has received much attention. Several publications, briefly discussed below, are related to the work described in this report. Most of them are based on boundary representation and attempt to solve all constraints simultaneously, which results in a large system of non-linear equations.

The general idea of using a graph, whose nodes represent solids, and whose branches represent constraints and relations that correspond to rigid motions, is described by Liberman, Wesley, et al. [40,41] as the architecture of their solid modelling system AUTOPASS. They do not, however, address the problem of specification and interactive editing of geometry. Eastman [42] proposed a similar hierarchical location graph, where links correspond to relative positions between bodies and are stored as rigid motions. Lee and Gossard [43] extended this idea and described a data structure, which stores relations between components of an assembly, and can be created interactively. It is based on a boundary representation and uses graphic input to select features that define relations.

Sutherland seems to be responsible for the popularity of constraint-based specification for geometry. His system SKETCHPAD [44] uses constraints between coordinates. Constraints are described by error expressions and solved by propagation of degrees of freedom, and by relaxation methods which are extremely slow. Constraints on coordinates were also used by Hillyard and Braid [45], who developed a general theory in which dimensions are used to specify algebraic constraints on the coordinates of vertices of a wire-frame representation. The constraint-based graphic system JUNO, developed by Nelson [46], uses Newton-Raphson iterations to solve a system of constraints on coordinates of 2-D points, expressed in terms of distances between vertices, and parallelism relations between line segments. It requires a good starting point to converge. Instead of vertices, Lin, Gossard, and Light [47] propose to use characteristic points of simple primitives bound by planes, cylinders, spheres, and cones. The geometry of a part is specified by a fixed topology and a set of constraints (dimensions) used to position the characteristic points.

More complex constraints cannot always be specified in terms of vertices. Kimura, Suzuki, and Wingard [48] use first-order predicate logic to express implicit constraints that allow specifying that the axis of a cylinder be aligned with the intersection of two planes. The topology of the part is assumed constant and the geometry is constructed by a sequence of transformations. Artificial intelligence techniques based on production rules are invoked to determine the evaluation sequence. Similarly, Managaki and Kawagoe [49] achieve a parameterization of geometric models by storing relations between parts. Relations are specified implicitly by topological information and by limits on the values of variables. Their model was also implemented by Krishnan and Patnaik [23] in the GEODERM system which uses an Entity-Relationship database model and a sequence of Euler operations to represent the boundary of designed parts.

Geometric constraints on curved surfaces can also be expressed explicitly without using topological information. The studies of Ambler and Popplestone [50], and of Lee and Andrews [51] address specifically the problem of converting constraints on surfaces into systems of equations. In both cases, the scope is limited to two constraint types: **against,** which matches two planes or achieves tangency between a cylinder and a plane [50], and **fit,** which aligns the axes of two cylinders. In Lee and Andrew's work, redundant equations are eliminated using a Jacobian matrix, and the remaining equations are solved by Newton-Raphson iterations, which may require a good guess for initial conditions in order to converge. Each constraint is converted into 16 or 18 equations and a typical 3-body configuration yields 100 equations with 84 variables. To improve the resolution process, Ambler and Popplestone developed techniques for separating the rotational and translational parts of rigid motions. They generate algebraic expressions for rigid motion matrices that correspond to compositions of a small number of constraints. Cycles in the graph are translated into two ways of writing the same motion. The resulting system of equations is manipulated algebraically to eliminate linearly-occurring variables.

To support more general constraints, Borning developed THINGLAB [52], where constraints are defined by rules to test them and by methods (described algorithmically) to satisfy them. They can be incomplete, circular, or contradictory, and are solved in two steps. First, methods of propagating degrees of freedom or known states yield a plan for constraint evaluation and compile the result of a plan for incremental satisfaction of constraints into Smalltalk code. The code is used to solve, whenever possible, individual constraints in one path and relies on relaxation methods to deal with circularity. Constraints are not restricted to geometry. User-defined objects can be organized in a hierarchical manner, similar to CSG, allowing two objects to share common sub-parts.

To avoid the problem of converting the constraints into a large system of simultaneous equations, we propose to evaluate constraints independently, in a sequential manner. The sequence of evaluation is defined by the user, rather than computed automatically as in [52].

Clearly, the execution of one constraint may invalidate a relation that was established by a previously executed constraint. For instance, if constraint 1 positions a surface *A* tangent to a cylinder *C*, and constraint 2 positions the same surface *A* tangent to a third surface *B*, the execution of constraint 2 after constraint 1 may result in situations where *A* is no longer tangent to *C*. Two techniques might be helpful to overcome such problems:

- indicate that the second constraint should be met by rotating *A* around the axis of *C*, or by translating it along that same axis,
- select *A* and *C* as the boundary feature of constraint 2 and move them both as a rigid body, while preserving their tangency relation.

These techniques, sufficient for many simple constructions, are inadequate to solve problems that may be easily formulated by several simultaneous constraints, but for which no simple sequential process is available.

Specifying the intrinsic dimensions of primitives through constraints may be addressed by constraint-based scaling transformations. However, in this report, we shall focus on rigid motions only.

To emphasize the techniques mentioned above, we restrict constraint-based motions to pure rotations and pure translations. This restriction is easily overcome by combining rigid motions, and leads to simpler specifications. We also introduce the notion of a current axis which may be set by the user and may be automatically changed during the evaluation of a constraint. Certain constraints[1] are independent of the current axis, but others will use the current axis and produce rotations around it or translations along it.

For instance, in our previous example, constraint 1 may result in a translation of *A* along the shortest path that will make it tangent to *C*. To preserve the tangency relation between *A* and *C*, constraint 1 may align the current axis with the axis of the cylinder *C*. If the user specifies that constraint 2 should not alter the current axis, the resulting motion will be a rotation around the axis of a cylinder or a translation along that axis and will preserve the tangency relation between *A* and *C*. This technique uses the natural symmetry of the cylinder and may be applied successfully to other natural quadrics.

This technique is illustrated in Figure 1 on page 11 and Figure 2 on page 12 where three non-parallel cylinders are automatically translated in such a way that each cylinder has a tangency contact point with the two others. This rearrangement is specified by three constraint-based motions which are independent of the dimensions of the cylinders.

Our representation is **descriptive**, because it is defined in terms of constraints, and also **imperative,** because the sequence and methods for evaluating the specification are provided by the user. Constraint-based rigid motions may be specified algorithmically. They are stored in methods written in AML/X [53], which typically contain calls to primitive geometric methods that compute measures defined by constraints between boundaries of primitives. Such basic methods typically take, as arguments, two geometric objects (points, curves, or surfaces) that are associated with primitives, and specified by pathnames. The methods return a measure, which can be an angle, a distance, a vector, or a rigid motion. Techniques for computing measures based on translational or rotational distances between two quadric surfaces are outlined below, and can be extended by the user, and integrated in higher level functions. This approach offers more flexibility for dealing with situations where a constraint may not be satisfied. The user may define his own exception handlers by using conditional statements, available in AML/X, to test the outcome of a measure computation and provide an alternative.

The remainder of this section describes techniques for computing measures and rigid motions from surface data. Computing measures from intersection data (edges and vertices) will not be addressed here because most intersections come from Boolean operations and are not directly available to the front-end system, and also because, unless the intersection edges are lines or circles, measures, such as the minimum distance between two three-dimensional curve segments, are very expensive to compute and rarely used to dimension mechanical parts. Typical methods compute parameters for rigid motions that establish a specific relation between two surfaces, for instance bring one surface in contact with another. We distinguish four cases:

[1] Throughout this report, we shall often use the term "constraint" to denote a constraint-based rigid motion.

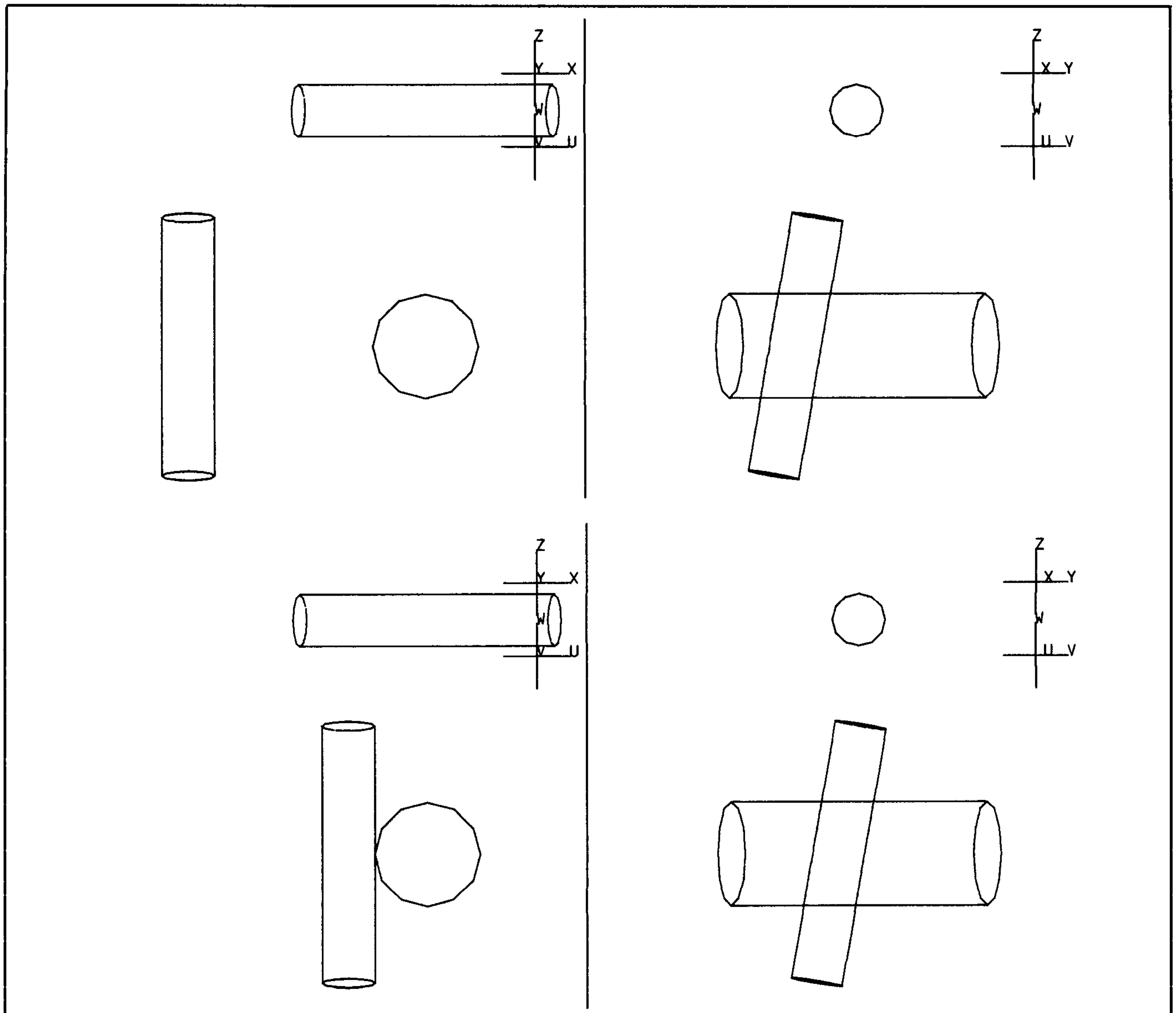

Figure 1. Three pairwise tangent cylinders: Three non-parallel cylinders are shown in a front and a side view (top). A first constraint-based transformation translates the left cylinder B1 until it touches the central cylinder A (bottom). The translation is done along the shortest path between both cylinders.

- minimum-distance translation,
- minimum-twist rotations,
- translations along the current axis,
- rotations around the current axis.

A **minimum-distance translation** makes two solids come in contact, so that their boundaries are tangent at the contact points. If the solids are initially disjoint, the simplest approach is to compute the minimum distance between the two solids and produce a pair of points (one on each solid), where the minimum distance is achieved. Clearly a translation of one of the two solids along the vector that separates the two points will achieve the desired contact. When the solids are disjoint polyhedra, an efficient algorithm developed by Orlowski [54] may be used. However, in the initial position, the two solids may interpenetrate. To correct the situation, one must bring the solids to a position, where their boundaries touch, but their interiors are disjoint. Cameron and Culley [55] developed algorithms that compute such a translation for convex polyhedra. For solids bound by subsets of curved surfaces, the above problem is more difficult and requires the computation of points, where

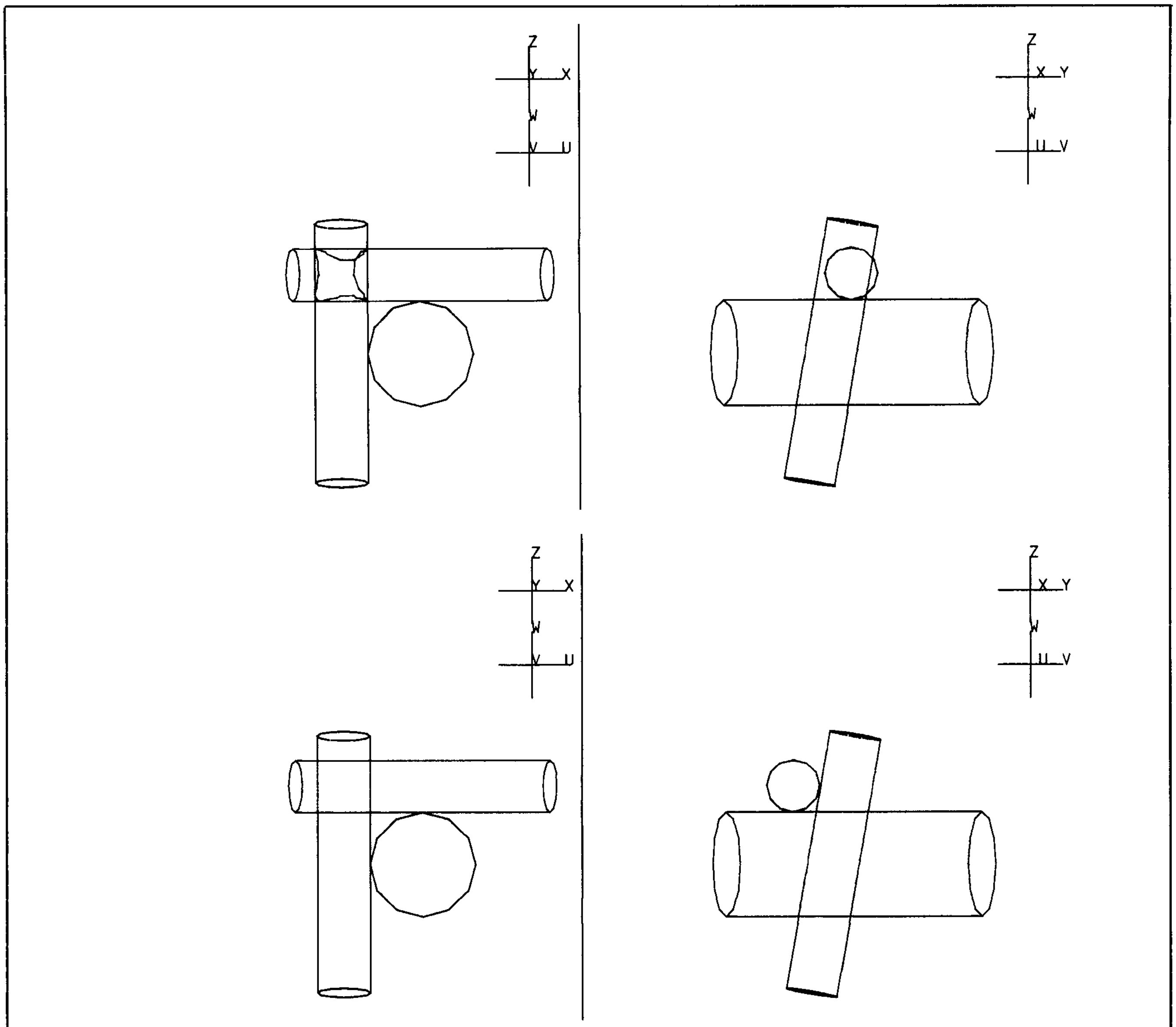

Figure 2. Three pairwise tangent cylinders (continued): A second transformation translates the top cylinder B2 until it also touches A. At this point B1 and B2 inter-penetrate (top). A third transformation translates B2 along the axis of A until B2 is tangent to B1. The tangency constraint between B2 and A is preserved (bottom).

extrema of distances between two sets are achieved. Pairs of such corresponding points are called *extremum-distance points.* Solving the general problem requires computing extremum-distance points between two surfaces, between a curve and a surface, between a point and a surface, between two curves, and so on.

We restrict our work to the surface-surface case, which is solved by finding pairs of distinct extremum-distance points P_A on surface A, and P_B on surface B, such that the line passing through P_A and through P_B is orthogonal to A at P_A, and to B at P_B. Clearly, for certain configurations of surfaces (such as two intersecting planes) this problem has no solution, and for other configurations (such as disjoint parallel planes) it has infinitely many. There is often more than one pair of such points; using any of these pairs to define a translation vector achieves a tangency relation between the two surfaces. Which pair should one use? The simplest is to pick the pair that defines the shortest translation vector, however this choice might lead to undesired configurations. To illustrate the problem, consider the tangent plane to both surfaces at the contact point after the tangency relation was achieved. If the two half-spaces bounded by the tangent surfaces are convex, they will lie either on the same side of the tangent plane or on opposite sides. The user might wish to distinguish between these cases. Furthermore, in the case of two quasi-disjoint tori, two topologically different tangency situations may arise: the two tori may or may not be linked. An elegant solution is to design procedures that will return all possible solutions, and to provide a syntax that will let the user specify unambiguously which solution should be used.

To simplify the description of the methods used to compute extremum-distance points, we use the notion of a *normal projection* of a point P on a surface S. This projection is the set of points Q_i such that the vector PQ_i is orthogonal to S at Q_i. Normal projections on natural quadrics are simple to compute [32].

For cases where a discrete number of solutions exist, the computation of such pairs of points is relatively simple. Instead of describing in detail the derivation for each one of the ten combinations of surface types, we solve only the most difficult case and briefly indicate methods used to solve the others. For two planes, the computation has a meaning only if the planes are parallel, in which case, we use any point on one plane and its normal projection on the other plane. If one of the surfaces is a sphere, minimal distance points Q_i on the other surface may be obtained as normal projections of the center of the sphere on the other surface. Corresponding points P_i on the sphere are in turn computed as the normal projections of the Q_i the sphere. If one of the surfaces is a plane and the other is a cone, extreme-distance points exist only for configurations where the axis of the cone forms a specific angle with the normal to the surface. In such cases, the computation reduces to computing the distance between the apex of the cone and the plane. The cylinder-plane case is even simpler. Cylinder-cylinder minimum distance is computed from the minimum distance between two lines. The cone-cylinder case is a simplification of the cone-cone case, which can be solved analytically as follows. Let cone 1 have for apex the point P_1, and for axis direction the unit vector D_1. Let θ_1 be the half-angle at the apex of cone 1. A point I_1 on the axis of cone 1 may be defined parametrically by $I_1 = P_1 + t_1D_1$. We use an equivalent notation for cone 2. The line passing through I_1 and I_2 will intersect the surfaces of both cones at right angles if it forms an angle $\pi/2 - \theta_1$ with the axis of cone 1 and an angle $\pi/2 - \theta_2$ with the axis of cone 2. This reduces to four systems of two simultaneous equations:

$$\begin{cases} I_1I_2 \cdot D_1 = \pm\lambda_1 \| I_1I_2 \| \\ I_1I_2 \cdot D_2 = \pm\lambda_2 \| I_1I_2 \| \end{cases}$$

with $\lambda_1 = \cos(\pi/2 - \theta_1)$ and $\lambda_2 = \cos(\pi/2 - \theta_2)$. Only two of these systems are needed to compute all solutions. These equations can be converted to:

$$\begin{cases} (I_1I_2 \cdot D_1)^2 = \lambda_1^2 \| I_1I_2 \|^2 \\ I_1I_2 \cdot (\lambda_2 D_1 \pm \lambda_1 D_2) = 0 \end{cases}$$

Substituting I_1 and I_2 by their parametric form yields two systems of two equations, which can be simply solved for t_1 and t_2, because the second equation in each system is linear and the first equation is quadratic. For each system, we obtain two pairs of values for t_1 and t_2, which yield two pairs of points I_1 and I_2. Intersections of the line through these points with both cones generate pairs of points that define the desired translation vectors. The four solutions correspond to four ways of translating a cone to achieve a tangency relation with another cone. Degenerate cases, where the above derivation is invalid, may easily be detected and solved by simpler methods.

A **minimum twist rotation** is used to align surfaces or axes. To align the axes of two cylindrical or conical surfaces, we must find a rotation that transforms the direction unit vector of one axis into the direction of the other. An example of an AML/X method that computes such a rotation is provided in the next section. Rotations that make axes parallel to planes are computed by similar methods. A variation of such methods may also be used to position a cone, so that its axis lies at a specified angle from a given plane, or from the axis of another cone or cylinder. These tools allow the positioning of a cone so that its minimum distance to a plane, a cylinder, or another cone is achieved along a whole line. Such a rotation, when combined with a previously described translation, will result in a tangency relation achieved along a line on the cone. For example, such a relation is obtained when a half-cone lies on a flat surface.

In certain configurations, two surfaces may be brought in contact, by **translation along the current axis.** We explain below how to compute the corresponding translational distance. For simplicity, we omit again the discussion of special cases. For two parallel planes, the translation distance is obtained by dividing the distance between the two planes by the dot product between the normal to the planes and the direction of the current axis. When one of the surfaces is a sphere, we compute the intersection of a line passing through the center of the sphere and parallel to the current axis with a pair of surfaces obtained by n-offsetting the second surface by the radius of the sphere. The n-offset of a surface is composed of a pair of surfaces, parallel to the original one, and positioned at a given distance on each side of it. N-offsets of natural quadrics are composed of natural quadrics and are simple to compute [32]. Similarly, when one of the surfaces is a cylinder, the problem reduces to finding a translation distance that achieves a tangency condition between the axis of the cylinder and the surfaces produced by n-offsetting the second surface by the radius of the cylinder. This problem is solved by considering the intersections of a line with a natural quadric surface. When the discriminant of the resulting quadratic equation is made equal to zero, we obtain a formula, which corresponds to a double intersection, i.e., a tangency condition. The line is described in this equation by a point P and a direction D. To indicate that the line may be translated along the current axis, we express P as $P_0 + tT$, where P_0 is the original position of a point on the line, and where T is the direction of the current axis. Solving for t yields the desired translation distance. Computing the distance between a cone and another cone or a cylinder, requires a more complicated approach. Let us consider the case of two cones A and B. We wish to find the length of a translation vector V that is parallel to the direction D of the current axis, given the constraint that a translation by V of the cone B will achieve a tangency condition between A and B. An efficient and simple solution comes from the following considerations. Let B' be a cone obtained by translating B by the vector that separates its apex from the apex of A, and let P be a plane that is tangent to both A and B'. Unless one of these two cones is "inside" the other, such planes exist (typically two or four) and may be easily computed from the descriptions of A and B. Any translation of B' in a direction parallel to P will achieve a tangency contact between A and B'. Let I be the intersection point of P with the line that passes through the apex of B and is parallel to the current axis. (If D is not parallel to P the point I exists and may be easily computed.) A translation of B that takes its apex to the point I will achieve the desired tangency condition. The above description leads to a closed form solution.

Similarly, in certain configurations, two surfaces may be brought in tangency contact by a **rotation around the current axis.** We briefly describe methods to compute corresponding angles. The intersections of a circle with natural quadrics requires solving a polynomial of degree four or less. Methods for automatically deriving the coefficients of such polynomials are described in [16]. When one of the surfaces is a sphere, we use again the technique of n-offsetting the other surface by the radius of the sphere. The problem reduces to the computation of the intersections of a circle, obtained by rotating the center of the sphere around the current axis, with the n-offset surfaces. When one of the surfaces is a plane and the other is a plane, a cylinder, or a cone, we compute the angle of a rotation around the current axis that will bring one of the two surfaces in a position where the angle formed by the normals or axes of both surfaces has a user-defined value. When both surfaces are cylinders or cones, the computation of the rotation angle that achieves tangency between them requires solving polynomials of high degree.

When a constraint may not be achieved by a motion of a specified type, the methods return a NIL symbol, which can be tested for by the calling statement. Therefore, the semantics of low-level methods is unambiguous, and the treatment of validity problems in the low level methods is avoided.

The system described here is meant to be used as a front-end to a full-fledged solid modeller. The front-end is based on the integration of an interpreter for AML/X [53] with a program for interactive graphics running on a geometrical engine.

The AML/X interpreter lets the user define and edit a CSG specification graph by creating solid primitives, and composing them through Boolean expressions that correspond to nodes of the tree. A typical input sequence follows:

```
X: NEW csg.cylinder(1,3);
Y: NEW csg.block(1, 1, 3);
Z: NEW csg.sphere(1);
V: NEW csg();
W: NEW csg();
W = X - ( Y + Z ) + V + V;
V = Y * Z - X;
```

AML/X offers possibilities for *overloading* operators, i.e. giving them a semantics that is dependent on the type of the operands. Here, *X*, *Y*, and *Z* are primitives, and *V* and *W* are nodes. The operators + , − , and * symbolize Boolean union, difference, and intersection. Their evaluation result in the execution of methods, which construct the binary CSG specification graph. Conventional operator precedence rules apply. The resulting tree has four additional internal nodes with no user-defined names. Each one of the three primitives appears three times in the graph. Editing the specification graph is simply done by providing new definitions. Rigid motions specifications might look like:

```
V.move(<X>, MDTrans, p1, p2);
T: NEW v3d(1,1,3);
W.move(<p3,p4,p5,p6>, T));
```

The first one applies to the primitive *X* in all instances of *V*, and is defined as a minimum-distance translation that will bring in contact two surfaces, represented by the pathnames *p*1 and *p*2. It is expected that the values of these pathnames be automatically derived by the system from graphically selected faces on a picture of *V*, which is produced by the evaluation sequence described below. The above specification would also be valid if *X* was a sub-tree of *V*. The second rigid motion is a simple translation by the vector *T*, and it applies to four graphically selected primitive instances.

The user may scroll through the list of constraints attached to each node, and select a position where a constraint should be modified, inserted or deleted. The general syntax for specifying a constraint-based rigid motion takes the form of a method call with four parameters: an aggregate of pathnames to primitive instances that constitute the boundary feature, the pathnames of a handle-surface and of a target-surface, and the name of a geometric method provided by the user and probably based on previously described procedures. The method will produce a rigid motion which will position the handle-surface so that, together with the target-surface, they satisfy the desired constraint.

Let us illustrate the implementation of these procedures in AML/X by providing, as an example, a simplified version of the method "**to**", extracted from the low level geometric extensions to AML/X, described in [56]. The method is associated with a class of 3-D vectors and may be invoked by **U.to(V),** where both **U** and **V** are unit vectors. It returns a rigid motion that transforms **U** into **V** by rotating around an axis that is orthogonal to both vectors. For simplicity, we omit the treatment of degenerate cases.

```
to: METHOD(V)
  W: NEW SELF ** V;
  M: NEW m3d.rztow(W);
  c: NEW SELF.dot(V);
  RETURN ((-M)*m3d.rz(ACOS(c))*M);
  END;
```

Several internal variables are declared and initialized at the same time. **W** is the cross product of both vectors. **M** is a rotation that aligns the Z-axis with **W** and is computed by the class-method **m3d.rztow.** The variable **c** represents the dot product of **U** and **V.** When the two vectors are not parallel, the resulting rigid motion is composed[2] of the inverse of *M*, followed by a rotation around the Z-axis, followed by *M*.

Graph evaluation is initiated by the user. First, the specification graph is expanded into a tree by duplicating all instances of nodes that appear more than once in the graph. A rigid motion, initialized to identity, is associated with each instance of a primitive. Then, rigid motion specifications are evaluated one by one, starting from the bottom of the specification graph. The evaluation of constraint-based motions is done in three steps:

1. use the pathnames of the handle and target surfaces to extract their surface type, parameters, and associated rigid motions,

[2] When its arguments are rigid motions, the "*" operator was overloaded with a new semantics which returns the combination of both rigid motions (the motion defined by the left argument is applied first). Similarly, the unary "−" operator, when applied to a rigid motion returns its inverse.

2. call the geometric method using the information obtained in 1 as actual arguments,
3. use the pathnames in the boundary feature aggregate to locate corresponding primitive instances in the tree, and combine their rigid motions with the rigid motion returned by 2.

When the evaluation process is over, the final positions of all primitives are described by the associated rigid motions in the expanded CSG tree. The tree may be passed to the display program described below, or sent to a full-fledged modeller.

This cycle is repeated during the specification and editing processes. If the user wishes to attach a constraint-based motion to a node, he selects that node and asks for a display of the solid currently represented by that node. The specification graph spanned by the selected node is evaluated, and the final tree is passed to the display program. The user can graphically select boundary features, handle-surfaces, and target-surfaces. He also indicates the name of the geometric procedure to be used to compute the rigid motion. Then this specification is stored in the list associated with the selected node. The user may order its immediate evaluation. A consistency check mechanism could speed up the evaluation by remembering whether the rigid motions of the primitives are compatible with the constraint that precedes the one currently added.

Modification of the Boolean structure of the specification graph may produce inconsistencies between pathnames and primitive instances. A solution, described in [36], allows to detect changes that produce such inconsistencies, and indicate them to the user.

The effect of constraints evaluation is displayed via a shaded picture which can be interactively manipulated by the user. The picture is produced directly from CSG by a method partly described in [35]. The basic principle is to generate a regular grid of points on the boundary of the primitives. The points are classified with respect to the CSG tree. A typical point membership classification procedures computes whether a point is in, out, or on the boundary of a given solid [30]. Our procedure, instead, classifies points with respect to the I-zones of the corresponding primitives [34], and is more efficient because it needs to classify points with respect to fewer primitives. It returns GOOD, if the point is inside or on the boundary of the solid, and BAD otherwise. Problems of coincident faces are solved by offsetting the points by a small distance, in a specific direction, away from the boundary of the primitives. Points are organized into sets of four neighbors, which define the corners of cells on the faces of primitives. If the four points of a cell are GOOD, their coordinates and associated intensities are sent to the geometrical engine. If the four points are BAD, the cell is not considered any further. If some, but not all points are GOOD, the cell is recursively subdivided into four subcells by generating intermediate points on the surface. When a maximum level of subdivision is reached, BAD corners of the cells are cut to better approximate intersection edges, and what remains is sent to the display device. At the end of the process, the geometrical engine contains a set of polygons with associated color and intensity information, and can fill them using Gourauld shading, while performing a depth buffer test for each pixel, in order to eliminate hidden faces. The user can move the view point interactively and obtain a new image per second.

The interactive shading program is fully operational, and AML/X methods for computing rigid motions from simple constraints, and for specifying, editing, and processing the CSG graph are available. The system was integrated as a front-end to the GDP solid modeller. However, the integration of these tools in a user-friendly system that offers higher level functions with intuitive semantics is still under progress.

DISCUSSION AND CONCLUSION

Our goal was not to develop a problem-solving system, but to produce an interface that will simplify the specification of geometry. Therefore, our approach does not provide a tool for solving problems where many constraints must be satisfied simultaneously. For example, our method cannot be used to specify a triangle by the length of its three sides. Auxiliary geometric entities, such as intersections of curves and surfaces, could be used to solve certain problems. For instance, the above triangle may be constructed from the intersection of two circles. A more general approach is to use constraint-type statements in AML/X, and have the system compute a solution. Such an extension to programming languages was proposed by Wilkes [57]. Statements, in form of equations, are differentiated from assignments and imply relations between variables.

The problem of specifying the dimensions of primitives is not addressed here. It may be elegantly integrated in our scheme by adding to the rigid motions a scaling operation, and by providing geometric methods that compute values of a scaling factor, for which specific tangency constraints are satisfied.

A more delicate problem is to enlarge the geometric domain. Toroidal surfaces are important in geometric modelling of mechanical parts. Unfortunately, computation on tori is much more difficult than on quadrics. For example, a solution to a fourth-degree equation is required to compute the minimum distance between a torus and a cylinder.

These problems need to be addressed in the future. However, the current design already offers some possibilities for important applications.

Traditionally, dimensions and tolerances were specified in blue prints as a set of simultaneous constraints, which can be viewed as an indirect parameterization [8]. Sequential constraint-based specification of geometry offers an alternative to the difficult problem of inverting indirect parameterization into CSG representation. This approach seems to be following the trend established by modern tolerancing practice. That is, replace a purely descriptive semantics of dimensioning, with a more procedural approach based on ordered sequences of dimension specifications. The procedural approach is better suited for capturing the designers' intentions of the functional meaning of dimensions, and suggests new ways of attacking the problem of tolerance analysis.

A sequence of rotations and translations, around or along local axes, may be viewed as a description of mechanisms or assembly processes when applied to sub-solids. For example, a constraint-based rigid motion that aligns the axes of two cylinders can be used to specify a prismatic or revolute joint. Using the common axis as the current axis, another constraint-based rigid motion may be used to compute extreme positions of the mechanism, defined by a contact between the moving part and some obstacle. Similarly, a sequence of assembly or set-up operations may be defined by constrained rotations and translations. A typical set-up sequence may position a part on a table, then push it against a fixture or clamp, then slide it along the table and the clamp until it touches a thrust. Such a sequence may be easily captured by simple rigid motions that align surfaces and translate solids until contact constraints are met. This description is preferable to computing the rigid motions that aligns a coordinate system with another because it offers a user-controlled decomposition of the set-up process into a sequence of operations that have good potential for automation [58].

If, from a constraint-based specification, one could deduce tangency relations between surfaces, the problem of unstable topologies [10] could be solved exactly by pure logic and would not require expensive symbolic manipulation. Constraint propagation in hierarchical networks was studied by Sussman and Steele [59], and the applications of inference mechanisms to geometric reasoning are currently under investigation [60]. Unfortunately, all constraints specified in the graph of our model are not necessarily met by the primitives in their final configuration because a rigid motion may destroy a relation established by a previously evaluated constraint. However, if a constraint-based motion establishes a relation between two primitives, it is simple to test if any further motion destroys this relation, because a relation may be destroyed only if a motion is applied to a boundary feature that contains one of the two primitive, but not both. Therefore, it appears possible to construct and maintain a graph of valid relations between primitive instances, as the specification graph is evaluated.

In conclusion, the proposed approach seems a viable alternative to purely descriptive schemes, which must deal with large systems of simultaneous equations. Its sequential and algorithmic nature offers a large flexibility and may lead to extensions important for manufacturing automation.

REFERENCES

[1] D. Meagher, "Geometric modeling using octree encoding", *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129-147, June 1982.

[2] R.F. Sarraga and W.C. Waters, "Free-form surfaces in GMSOLID: Goals and issues", in M. S. Pickett and J. W. Boyse, Eds., *Solid Modeling by Computers.* New York: Plenum Press, pp. 187-209, 1984.

[3] A.A.G. Requicha and H.B. Voelcker, "Solid modelling: Current status and research directions", *IEEE Computer Graphics and Applications*, vol. 3, no. 7, pp. 25-37, October 1983.

[4] J.W. Boyse and J.E. Glichrist, "GMSolid: Intersactive modelling for design and analysis of solids", *IEEE Computer Graphics and Applications*, vol. 2, no. 2, pp. 27-40, March 1982.

[5] D.D. Freund, "An interactive procedure for constructing lines and circle tangencies", *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 59-63, April 1986.

[6] W.F. Fitzgerald, "Using actual dimensions to determine the proportions of line drawings in computer graphics", *Computer Aided Design*, vol. 13, no. 6, pp. 377-382, November 1981.

[7] D.L. Vossler, "Sweep-to-CSG conversion using pattern recognition techniques", *IEEE Computer Graphics and Applications*, vol. 5, no. 8, pp. 61-68, August 1985.

[8] A.A.G. Requicha, "Representation of tolerances in solid modeling: Issues and alternative approaches", in M.S. Pickett and J.W. Boyse, Eds., *Solid Modelling by Computers*, New York: Plenum Press, pp. 3-22, 1984.

[9] Y.T. Lee and A.A.G. Requicha, "Algorithms for computing the volume and other integral properties of solids: I - Known methods and open issues, II - A family of algorithms based on representation conversion and cellular approximation ", *Comm. ACM*, vol. 25, no. 9, September 1982.

[10] M.A. O'Connor, "Projection of natural quadrics", IBM Research Report RC 11188, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, June 1985.

[11] R.F. Sarraga, "Algebraic methods for intersections of quadric surfaces in GMSOLID", *Computer Vision, Graphics and Image Processing*, vol. 22, no. 2, pp. 222-238, May 1983.

[12] A.P. Morgan and R.F. Sarraga "A method for computing three surface intersection points in GMSOLID", Report GMR-3964, General Motors Research Labs., Warren, MI, February 1982.

[13] N.M. Samuel, A.A.G. Requicha and S.A. Elkind, "Methodology and results of an industrial part survey", Tech. Memo. No. 21, Production Automation Project, Univ. of Rochester, July 1976.

[14] J.R. Rossignac and A.A.G. Requicha, "Constant radius blending in Solid Modelling", Computers in Mechanical Engineering, vol. 3, no. 1, pp. 65-73, July 1984.

[15] G. Monge, *Applications de l'analyse à la géometrie.* Paris: Bachelier, 5th ed., 1849.

[16] J.R. Rossignac and A.A.G. Requicha, "Piecewise Constant Curvature Approximations for Geometric Modelling", IBM Research Report RC 12171, IBM, T.J. Watson Research Center, Yorktown Heights, NY, October 1986.

[17] W. Bohm, G. Farin, and J. Kahmann, "A survey of curve and surface methods in CAGD", *Computer Aided Geometric Design*, vol. 1, no. 1, pp. 1-60, July 1884.

[18] H. Chiyokura and F. Kimura, "Design of solids with free-form surfaces", *Proc. ACM SigGraph '83*, Detroit, MI, pp. 73-82, July 25-29, 1983.

[19] D.D. Grossman, "Procedural representation of three-dimensional objects", IBM Journal of Research and Development, vol. 20, pp. 582-589, November 1976.

[20] N. Okino *et al.*, "TIPS-1", Institute of Precision Engineering, Hokkaido University, Sapporo, Japan, 1978.

[21] K.J. Weiler, "Edge-based data structure for solid modelling in curved surface environments", *IEEE Computer Graphics and Applications*, vol. 5, no. 1, pp. 21-40, January 1985.

[22] W.M. Newman and R.F. Sprull, *Principles of interactive computer graphics*, New York: McGraw-Hill, 2nd edition, 1979.

[23] D. Krishnan and L.M. Patnaik, "Design system using an entity-relationship model", *Computer Aided Design*, vol. 18, no. 4, pp. 207-218, May 1986.

[24] C.M. Eastman and K. Weiler, "Geometric modelling using the Euler operators", *Proc. First Annual Conf. on Computer Graphics in CAD/CAM Systems*, MIT Cambridge, MA, pp. 248-259, April 9-11, 1979.

[25] M. Mantyla and R. Sulonen, "GWB: a solid modeller with Euler operators", *IEEE Computer Graphics and Applications*, vol. 2, no. 7, pp. 17-31, September 1982.

[26] A.A.G. Requicha, "Mathematical models of rigid solid objects", Tech. Memo. No. 28, Production Automation Project, Univ. of Rochester, November 1977.

[27] C. Hoffmann and J. Hopcroft, "Automatic surface generation in computer aided design", TR 85-661, Computer Science Dept., Cornell Univ., Ithaca, NY, January 1985.

[28] J.R. Rossignac and A.A.G. Requicha, "Offsetting operations in Solid Modelling", *Computer Aided Geometric Design*, vol. 3, no. 2, pp. 129-148, August 1986.

[29] A.A.G. Requicha, "Representations for rigid solids: Theory, methods, and systems", *ACM Computing Surveys*, vol. 12, no. 4, pp. 437-464, December 1980.

[30] R.B. Tilove, "Set membership classification: A unified approach to geometric intersection problems", *IEEE Trans. on Computers*, vol. C-29, no. 10, pp. 874-883, October 1980.

[31] A.A.G. Requicha and H.B. Voelcker, "Boolean Operations in Solid Modelling: Boundary Evaluation and Merging Algorithms", *Proceedings of the IEEE*, Vol. 73, No 1, January 1985.

[32] J.R. Rossignac, "Blending and Offsetting Solid Models", Tech. Memo. No. 54 (Ph.D. Dissertation), Production Automation Project, Univ. of Rochester, June 1985.

[33] C.M. Brown, "PADL-2: A technical summary", *IEEE Computer Graphics and Applications*, vol. 2, no. 2, pp. 69-84, March 1982.

[34] J.R. Rossignac and H.B.Voelcker, "Active Zones in Constructive Solid Geometry for Redundancy and Interference Detection", IBM Research Report, RC 11991, IBM, T.J. Watson Research Center, Yorktown Heights, NY, June 1986.

[35] J.R. Rossignac and A.A.G. Requicha, "Depth buffering display techniques for constructive solid geometry", *IEEE, Computer Graphics and Applications,* vol. 6, no. 9, pp. 29-39, September 1986.

[36] A.A.G. Requicha and S.C. Chan, "Representation of geometric features, tolerances and attributes in solid modellers based on constructive geometry", *IEEE Journal of Robotics and Automation*, vol. 2, no. 3, September 1986.

[37] S.D. Roth, "Ray casting for modeling solids", *Computer Graphics and Image Processing*, vol. 18, no. 2, pp. 109-144, February 1982.

[38] D.C. Anderson, "Closing the gap: a workstation-mainframe connection", *Computers in Mechanical Engineering*, vol. 4, no. 6, pp. 16-24, May 1986.

[39] P.G. Molari, (Private communication), Facolta di ingegneria, Instituto di Meccanica Applicata alle macchine, University of Bologna, Italy, August 1986.

[40] L.I. Liberman and M.A. Wesley, "An automatic programming system for computer controlled mechanical assembly", *IBM Journal of Research and Development*, vol. 21, no. 4, pp. 321-333, July 1977.

[41] M.A. Wesley, T. Lozano-Perez, L.I. Lieberman, M.A. Lavin, and D.D.Grossman, "A geometric modelling system for automated mechanical assembly", *IBM Journal of Research and Development*, vol. 24, no. 1, pp. 64-74, January 1980.

[42] C.M. Eastman, *The design of assemblies*, SAE Technical Paper Series, Society of Automotive Engineers, Inc., USA, 1981.

[43] K. Lee and D.C. Gossard, "A hierarchical data structure for representing assemblies: Part 1" *Computer Aided Design*, vol. 17, no. 1, pp. 15-19, January/February 1985.

[44] I. Sutherland, "Sketchpad, a man-machine graphical communication system", PhD thesis, MIT, January 1963.

[45] R.C. Hillyard and I.C. Braid, "Characterizing non-ideal shapes in terms of dimensions and tolerances" *ACM Computer Graphics*, vol. 12, no. 3, pp. 234-238, August 1978.

[46] G. Nelson, "Juno, a constraint-based graphics system", *SIGGRAPH'85*, vol. 19, no. 3, pp. 235-243, San Francisco, July 22-26, 1985.

[47] V.C. Lin, D.C. Gossard, and R.A. Light, "Variational geometry in computer aided design", *ACM Computer Graphics*, vol. 15, no. 3, pp. 171-177, August 1981.

[48] F. Kimura, H. Suzuki, and L. Wingard, "A uniform approach to dimensioning and tolerancing in product modelling", *Proc. CAPE'86*, 1986.

[49] M. Managaki and K. Kawago, "Parametric man/machine interactions with semantic data", *Computer Graphics*, vol. 7, no. 3-4, pp. 233-242, 1983.

[50] A.P. Ambler and R.J. Poppelstone, "Inferring the positions of bodies from specified spatial relationships" *Artificial Intelligence vol. 6, pp. 157-174, 1975.*

[51] K. Lee and G. Andrews, "Inference of the positions of components in an assembly: Part 2." *Computer Aided Design*, vol. 17, no. 1, pp. 20-24, January/February 1985.

[52] A. Borning, "The programming language aspects of Thinglab, a constraint-oriented simulation laboratory", *ACM Transactions on Programming Languages and Systems*, vol. 3, no. 4, pp. 353-387, October 1981.

[53] L.R. Nackman, M.A. Lavin, R.H. Taylor, W.C. Dietrich, and D.D. Grossman, "AML/X: a programming language for design and manufacturing", IBM Research Report RC 11992, July 1986.

[54] M. Orlowski, "The computation of the distance between polyhedra in 3-space", presented at *SIAM Conference on Geometric Modeling and Robotics*, Albany, NY, July 1985.

[55] S.A. Cameron and R.K. Culley, "Determining the minimum translational distance between two convex polyhedra", *Proc. 1986 IEEE Int'l Conf. on Robotics and Automation*, San Francisco, CA, pp. 591-596, April 7-10, 1986.

[56] J.R. Rossignac, "Simple geometry in AML/X: points, vectors, coordinate frames, and linear transformations", Internal report, Design Automation Group, IBM, T.J. Watson Research Center, Yorktown Heights, NY, May 1986.

[57] M.V. Wilkes, "Constraint-type statements in programming languages", *Communications of the ACM*, vol. 7, no. 10, pp. 87-88, October 1964.

[58] S.C. Chan and H.B. Voelcker, "An introduction to MPL - A new machining process/programming language", *Proc. IEEE Internat. Conf. on Robotics and Automation", San Francisco, CA, pp. 333-344, April 7-10, 1985.*

[59] G.J. Sussman and G.L. Steele Jr. "CONSTRAINTS - A language for expressing almost-hierarchical descriptions", *Artificial Intelligence*, vol. 14, pp. 1-39, 1980.

[60] F. Arbab and J.M. Wing, "Geometric reasoning: A new paradigm for processing geometric information", Report TR-85-333, Computer Science Department, University of Southern California, Los Angeles, July 1985.