An ATMS-Based Geometric Constraint Solver for 3D CAD

Shuichi SHIMIZU  Keisuke INOUE  Masayuki NUMAO
IBM Research, Tokyo Research Laboratory
5-19 Sambancho, Chiyoda-ku, Tokyo 102, Japan

Abstract

Since 3D models have become essential for the integration of CAD/CAM systems, there is a strong need for a new modeling technology for creating and modifying 3D shapes. The feature-based approach is promising, because features such as “step” and “ribs,” which are well known to designers, are used as primitives of Constructive Solid Geometry (CSG) trees. To increase the flexibility of feature operations, the authors have developed a constraint-based method of describing features. Every feature is controlled by geometric constraints, which naturally specify the dimensioning information that reflects the designer’s intent. This paper proposes an ATMS-based geometric reasoning system that efficiently evaluates the constraints in order to determine geometric attributes, and detects over-constrained situations in order to resolve conflicts among the constraints.

1 Introduction

3D models have now become essential for the integration of CAD/CAM systems. As a result, there is a strong need for a new modeling technology for creating and modifying 3D shapes, since the existing technology is immature in comparison with that for 2D models.

In general, a designer spends most of his time creating shapes that satisfy design constraints. If some of these constraints, such as geometric constraints, can be automatically satisfied by the system, he will have more time to concentrate on creative aspects of the design, and will no longer need to consider all the geometric data on products. Therefore, a constraint-based approach is suitable for representing a designer’s intent. Some constraint-based 2D systems have already been developed [1, 2]. However, it is difficult to apply these 2D methods naively to 3D models, mainly because their reasoning time increases in proportion to the polynomial order of the number of the constraints, and because 3D shapes are much more complicated and require many more constraints.

We previously proposed a methodology for representing 3D models by means of constraints and geometric reasoning, using a rule-based system [3]. That system had some serious drawbacks related to efficiency and over-constrained cases. Here, we propose a new method of geometric reasoning based on a definite geometric constraint solver and ATMS [7], which eliminates these drawbacks. In Section 2, we introduce the concept of geometric constraint-based representation. In Section 3, we discuss the problems involved in conventional approaches and describe the new method. In Section 4, we give some examples based on our approach.

2 Constraint-based 3D model

2.1 Hierarchical representation

We adopt a hierarchical representation of 3D objects, including form-features [4] in the intermediate level, because complicated 3D shapes can be easily constructed from simple form features. The levels of hierarchical representation are as follows:

Object represents a product component itself.
Form-feature is defined by a primitive and its relations to other primitives.
Primitive is a simple shape such as a prism, cylinder, or cone.
Element is a geometric component of a primitive, such as a face, edge, or vertex.

To establish a complete set of shapes, it is necessary to determine the following attributes for geometric elements, faces, edges, and vertices:

Face : coordinates of a point on it and a normal
Edge : coordinates of a point on it and a direction
Vertex : spatial coordinates.
Figure 1 shows an example of an object that consists of four form features: box, step, rib, and chamfer. The first three features are represented as four-sided prisms and the last as a three-sided prism. Each primitive is attached to the base object or subtracted from it, according to the properties of the associated form-features.

2.2 Geometric constraints

The geometric constraints among geometric elements in primitives are given, and each is classified into one of the following categories:

Dimensional constraints: measurable relations between elements, such as distances and angles.

Topological constraints: topological relations among neighboring elements.

Since structural constraints such as "on" and "coincident" can be defined as distances whose value is zero, they are included in the category of dimensional constraints. To determine geometric elements uniquely as possible, the following additional flags are attached to distance constraints:

- \( \text{distance}(f, f) \): inside/outside, same/opposite
- \( \text{distance}(e, f) \): inside/outside
- \( \text{distance}(v, f) \): inside/outside
- \( \text{distance}(e, e) \): same/opposite direction

where \( f, e, \) and \( v \) mean face, edge, and vertex, respectively. These flags disambiguate the sign of the direction and of the position. For example, an "inside" flag means that the first argument is inside the second argument.

\[
\begin{align*}
(f_1, f_2, f_3, v) & : v = f_1, f_2, f_3 \text{ and } [f_1, f_2, f_3] \geq 0 \\
(e_1, e_2, v) & : v = e_1, e_2 \\
(e, f, v) & : v = e, f \text{ and } (e, f) \geq 0 \\
(v_1, v_2, e) & : e = v_1, v_2 \\
(f_1, f_2, e) & : e = f_1, f_2 \\
(e_1, e_2, f) & : f = e_1, e_2 \\
(v, e, f) & : f = v, e \\
(v_1, v_2, v_3, f) & : f = v_1, v_2, v_3
\end{align*}
\]

Table 1: Topological constraints

The topological constraints shown in Table 1 refer to the topology within a primitive, and consist of derivation rules and directional limits. Figure 2(a) shows an example of three faces, \( f_1, f_2, \) and \( f_3 \), that share a vertex \( v \) in a convex prism. The derivation rule of the topological constraint "topological\( (f_1, f_2, f_3, v) \)" indicates that the vertex \( v \) is derived from the three faces. Its directional limit indicates that their normal vectors have the following limitation:

\[
[n_{r1}, n_{r2}, n_{r3}] \geq 0,
\]

where \( n_{r1}, n_{r2}, \) and \( n_{r3} \) are the normal vectors of the faces, and \( [a, b, c] \) represents a scalar triple product of the vectors \( a, b, \) and \( c \). As well as the flags of the dimensional constraints, the directional limits are used to choose a correct attribute value from several ambiguous ones. Either a positive or a negative flag is attached to each of the directional limits. The constraint "topological\( (f_1, f_2, e) \)" indicates that the edge \( e \) is derived from the faces, \( f_1 \) and \( f_2 \). The order of the faces is important because the direction of \( e \) should be uniquely determined, as shown in Figure 2(b).

![Figure 2: Examples of topological constraints](image-url)
3 Geometric reasoning

3.1 Shortcomings of previous work

Some previous studies have presented constraint-based approaches to handling geometry. Aldefeld [1] has proposed geometric constraint propagation with a rule-based system for 2D drawings. Figure 3 shows an example of his rule, which is arranged for 3D input. When all the conjoined parts of the condition are satisfied, the constraints are evaluated so that the attribute of the conclusion part is assigned by the numeric calculation procedure attached to the rule. This assignment may then trigger the firing of the other rules that include the conclusion of the rule in the condition. Thus, constraint evaluation and propagation is performed as a forward inference mechanism of a rule-based system.

condition:

\[
\begin{align*}
\text{(Attribute vertex1)} & - \\
\text{(Attribute vertex2)} & - \\
\text{(Attribute face1)} & - \\
\text{(Distance vertex1 face2)} & - \\
\text{(Distance vertex2 face2)} & - \\
\text{(Angle face2 face1)} & - \\
\end{align*}
\]

closure:

\[
\begin{align*}
\text{(Attribute face2)} & * \\
\end{align*}
\]

Figure 3: Example of constraint evaluation by rule

The rule-based evaluation and propagation approach has two major problems, namely, poor handling of temporary conflicts and inefficiency. To avoid infinite loops, the forward inference mechanism does not allow multiple values to be assigned to a single geometric element. This automatically excludes over-constrained situations, but it causes another problem: no temporary conflicts can be handled. Over-constrained situations, especially conflicts of constraints, should be resolved in the final stage of design, so that a product has a consistent shape. However, they should be temporarily allowed in some design stages, since they usually occur when designers modify the attribute values of some constraints or replace some constraints with others because of changes in a product's specification. Otherwise, the designers are obliged to find and remove any constraints that conflict with the new ones before they make any changes.

To resolve the first problem, Ando et al. [2] introduced the Assumption-based Truth Maintenance System (ATMS) [7] for dependency management with a rule-based system. The dependency information in the ATMS represents the history of determination processes. Using this information, the system can distinguish over-constrained situations from cyclic dependency, which causes infinite loops. If conflicts occur, their sources are recorded as nogoods, and the reasoning continues in multiple consistent contexts each of which contain no nogoods. However, the inefficiency problem still remains, because of the evaluation and propagation by the forward inference mechanism, as described below, and also because of multiple context reasoning.

The inference engine of a rule-based system must observe every datum that is put into a working memory, and apply it to every rule. This global processing of all input is a major reason for the inefficiency problem: the time required for pattern matching between all the data and rules increases in a polynomial order whose base is the number of geometric elements and whose power depends on the number of condition patterns in each rule [5]. An explicit control of reasoning [6] does not help much, since every geometric rule must be applied. Aldefeld [1] reported that the reasoning time increases approximately in proportion to the fourth power of the number of geometric elements in his system. Our previous approach [3] has the same problem of inefficiency.

In the next section, we describe our new geometric constraint solver, which localizes the constraint evaluation in each geometric element, and leads to efficient reasoning. We then describe the geometric constraint propagation realized by using ATMS consumer architecture [9]. We also discuss the computational complexity of the ATMS-based constraint propagation in our approach.

3.2 Using intermediate states

First we introduce the term intermediate states for geometric elements. These states make geometric reasoning much more efficient, because they remove the need for complicated pattern matching to find complete conditions that determine the geometric attributes of elements. As there are many combinations of constraints that determine geometric attributes, it may be thought that the system has to prepare many rules, since each combination requires one rule unit.

Figure 3 shows as an example a rule that determines a face if a vertex on it, the face's distance from another vertex, and the angle the face makes with another face are given. However, such rules can be avoided by introducing intermediate states and using a small set of reduction rules, as described below.
This idea comes from the following observation: The face in the conclusion of Figure 3 cannot be determined until all the constraints in the condition part have been collected. When not all of the constraints are satisfied, the rule is not fired. In this case, however, the face is constrained in terms of freedom. For instance, when the distance constraint between a vertex and a face is given, and the vertex has been determined, the face is constrained to be tangent to a sphere whose center is the vertex and whose radius is the distance value, as shown in Figure 4(a). If, in addition, another vertex on the face is given, the face is constrained to be tangent to a cone, as shown in Figure 4(b). If such intermediate states are manipulated, it is unnecessary to wait until all of the conditions are satisfied, and the attribute determination processes can be localized in each element so that the reasoning becomes more efficient.

![Diagram](image)

Figure 4: Examples of intermediate states for faces

The geometric constraints given by the designers are fed into the constraint solver in terms of constraint symbols. A face accepts the following constraint symbols:

- **pnt** (1): the face includes a point on it
- **sph** (1): the face is tangent to a sphere
- **lin** (2): the face includes a line on it
- **cyl** (2): the face is tangent to a cylinder
- **agl** (1): an angle from some vector is given
- **nrm** (2): a normal vector of the face is given
- **con** (2): the face is tangent to a cone

because every geometric constraint can be converted to a constraint symbol. The numbers following the symbols indicate how many degrees of freedom they remove. Table 2 shows conversions from constraints to symbols for a face. For instance, when the distance from an edge to a face is given, and the edge is already determined, the face is constrained by the term “lin” if the distance value is zero; otherwise it is constrained by “cyl,” which means that it is tangent to a cylinder. Every conversion can be immediately performed whenever any constraint is given.

<table>
<thead>
<tr>
<th>constraint</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>distance(face, face)</td>
<td>face</td>
</tr>
<tr>
<td>distance(edge, face)</td>
<td>cyl or lin</td>
</tr>
<tr>
<td>distance(vert, face)</td>
<td>sph or pnt</td>
</tr>
<tr>
<td>angle(face, face)</td>
<td>agl or nrm</td>
</tr>
<tr>
<td>angle(edge, face)</td>
<td>agl or nrm</td>
</tr>
</tbody>
</table>

Table 2: Mapping of constraints to symbols for a face

<table>
<thead>
<tr>
<th>pnt</th>
<th>sph</th>
<th>lin</th>
<th>cyl</th>
<th>agl</th>
<th>nrm</th>
<th>con</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each intermediate state is represented by one or more symbols: the examples in Figures 4(a) and (b) indicate the “sph” state and the “con” state, respectively. Some pairs in the set can be reduced to other symbols. Table 3 shows a matrix whose contents indicate the reduction results of two constraint symbols. A circle sign in the table means a determined state. For instance, the symbols “pnt” and “nrm” are reduced to “face.” A minus sign means that no reduction is necessary, and a plus sign means the same, except that a conflict check is necessary.

<table>
<thead>
<tr>
<th>distance(face, face)</th>
<th>distance(edge, face)</th>
<th>distance(vert, face)</th>
<th>angle(face, face)</th>
<th>angle(edge, face)</th>
</tr>
</thead>
<tbody>
<tr>
<td>face</td>
<td>cyl</td>
<td>sph or pnt</td>
<td>agl or nrm</td>
<td>agl or nrm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Reduction matrix for a face's symbols

Let us now give an example to show how a reduction matrix is used: When a face constrained in a “sph” state (Figure 4(a)) is given a “pnt” symbol, then the two symbols are integrated into “con” (Figure 4(b)), and as a result, the face becomes tangent to a cone. When, in addition, a new “agl” symbol is given, the “con” and “agl” are reduced to “face,” which indicates a determined state.

Note that, in some cases, the reduction leads to a determined state, even when the simple accumulation of degrees of freedom is less than three, which is the number of degrees of freedom of a face. For instance, the reduction of “pnt” and “sph” may be the face itself if the “pnt” is exactly on the “sph.” The reduction processes thus decrease the degrees of freedom to zero. However, this does not always mean that the value is uniquely determined, but only that the domain of the possible values becomes finite. For instance, the two “agl”s are reduced to “nrm.”

1 Like faces, edges and vertices have reduction matrices and conversion rules, but we do not show them here.
but the normal vector may have two different values. In this case, two “urg”’s are produced, and thus they make the face ambiguous. In order to get a unique value, we need disambiguation. In Section 3.4, we will describe a disambiguation process.

3.3 Propagation using ATMS

Every constraint can be represented as an assumption, which may be canceled later. On the other hand, every element is considered indirectly by one or more such constraints. The ATMS maintains such dependency information: an ATMS node has various sets of assumptions under which it is believed. Each set of assumptions is called an environment, and the environments in which a node is believed are called that node’s label [7]. Consequently, geometric elements and constraints with specific attribute values can be represented as ATMS nodes, while those that are variables can be represented as ATMS classes, which are sets of nodes.

An ATMS justification refers to direct dependency between nodes: a node depends on a conjunction of other nodes. The labels are calculated according to given justifications. As the environments in the labels are guaranteed to be minimal, cyclic dependency is automatically rejected in the ATMS. In other words, the direction of the label propagation is automatically determined by the label’s minimality, even though some dual justification pairs, such as “a ⇒ b” and “b ⇒ a,” are given.

When a constraint between two elements is given, the system should determine which element is constrained by the other and what values are given to the state of the appropriate element. The ATMS label and ATMS consumer architecture [9] makes this determination straightforward. For instance, a distance constraint between an edge e1 and a face f2 is described by conjunctive class consumers as follows:

\[
\text{distance}(e_1, f_2), e_1 \rightarrow [f_2; \text{cyl or lin}], \\
\text{distance}(e_1, f_2), f_2 \rightarrow [e_1; \text{plan}],
\]

where the right-hand side indicates the consumer body that calculates values and declares appropriate justifications, and the left-hand side consists of trigger conditions for invoking the consumer. As the conditions consist of classes, the consumer is run once every node tuple when all nodes in the tuple are believed, that is, when they all have nonempty labels. In this example, if the class e1 has a value node, the following justifications are invoked:

\[
\langle e_1 = (P, D) \rangle, \langle \text{distance}(e_1, f_2) = (L, \text{inside}) \rangle \\
\Rightarrow \langle f_2; \text{cyl} = (P, +L, D) \rangle,
\]

where P and D are a point on the edge and the direction of the edge, and L is the distance value.

Note that these two consumers show cyclic dependency between elements. However, as mentioned above, the sequences of the label update always terminate, because of label minimality.

The symbol reduction processes for a face in Table 3 can also described as follows by using consumers:

\[
\text{put, put} \rightarrow \text{[lin]}, \\
\text{put, sph} \rightarrow \text{[con, face, or NG]}, \\
\text{sph, sph} \rightarrow \text{[cyl, con, face or NG]}, \\
\text{agl, agl} \rightarrow \text{[nrm or NG]},
\]

where “NG” means that conflicts occur and that their sources should be recorded in the nogood database [7], so that reasoning in inconsistent environments can be avoided. Consumers that include an identical class in the conditions are evaluated when a new node is added to the class. Note that consumers corresponding to the circle and plus sign contents in Table 3 are also necessary, but are not shown here. Their bodies are “[face or NG]” and “[NOP or NG],” respectively, where “NOP” means no operation.

Multiple environments in a node’s label indicate that some constraints are redundant. On the other hand, multiple nodes in a class indicate that some constraints conflict, or that reduction processes are required. For instance, two “urg” states always cause conflicts, since a face should have a single normal vector, while two “put” states always produce a new “lin” state. We call these two types of class a single-node class and a multiple-node class, respectively. Face, edge, and vertex classes are single-node classes.

Constraint propagation is achieved by consumers and by the incremental label update algorithm of the ATMS [10]. Once an element has a determined attribute, it may impose constraints on other elements. Then the reduction processes begin for them, and they may be determined and in their turn become sources of further geometric elements.

3.4 Handling ambiguity

When a symbol is produced with ambiguous attribute values, it constructs multiple contexts by using “choose” and “nogood” forms of the ATMS as follows:

\[
\text{choose}\{A, B\} \quad \text{and} \quad \text{nogood}\{A, B\},
\]

which mean that the disjunction “AVB” must hold, and that the assumptions A and B cannot hold simultaneously. The nodes of the new symbol are then justified with these assumptions as follows: 286
is given in Section 4.2. The reasoning is guaranteed to be performed in a consistent environment, which is one of the multiple contexts resulting from division by the nogood database in the ATMS. However, as the designer needs only one consistent shape for a product, which means one consistent context, dependency-directed backtracking [11] is necessary in order to cancel all contexts except the desired one.

In order to reduce multiple contexts to a single context, the appropriate single-term nogood should be asserted, that is, some assumptions should individually be asserted as nogood. The diagnostic reasoning for this has been discussed in terms of minimal candidates [12] and runs as follows: Suppose that the nogood database is \( \{E_1, E_2, \ldots\} \). The disjunctive form,

\[
E_1 + E_2 + \ldots = 0,
\]

shows the necessary and sufficient condition for the desired single context in which no multiple-term nogood is allowed. For instance, the disjunctive form of nogoods \( \{\{A, B, C\}, \{A, B, D\}, \{B, C, D\}\} \) is as follows:

\[
ABC + ABD + BCD = 0.
\]

As any disjunctive form can be transformed into a conjunctive form, this form is transformed into

\[
(A + C)(A + D)B(C + D) = 0,
\]

which means that the conjunction \( A V C, A V D, B, \) and \( C V D \) should be false, and thus that each of the terms is a cancellation candidate. For instance, if assumptions \( A \) and \( C \) are asserted as nogood, no multiple-term non-controlled nogood exists, and a single potential interpretation exists.

Note that while canceling each of the candidates is sufficient to resolve conflicts, canceling some of them may also change uniquely determined elements to undetermined ones. Thus, the candidates are divided into two types: those that make all nodes in some single-node classes empty-labeled, and those that keep only one node in each single-node class nonempty-labeled. It is usually desirable to cancel the latter type. A detailed example is given in Section 4.3.

3.6 Complexity consideration

Our approach has the following characteristics with regard to computational complexity.

No interpretation construction required. As mentioned above, our geometric constraint solver is based on an incremental label update algorithm in the ATMS. Interpretation construction, which is NP-complete [13], is not required.

No heavy hyperresolution is required. In most cases, the number of ambiguous candidates is two. Consequently, disambiguation can be performed by the rule for binary disjunctions, which is very fast [8].

Each primitive can bound chosen assumptions. Ambiguous candidates may cause an exponential increase in the size of the ATMS when they are sources of other ambiguous candidates. However, this can be avoided, if chosen assumptions are propagated only inside a primitive. We set a restriction that forbids propagation from a primitive's elements to other primitives unless the elements are uniquely determined. It leads to no serious defect, because most ambiguities are resolved by the topology inside each primitive.

Propagation paths are short. In most design stages, the designer adds new features to established ones. As a result, most of the label propagation is performed inside the new feature. That is, most propagation does not depend on the model size, and is done within
a constant time, except when some of the constraints that are specified at an early stage are changed, or when an over-constrained situation occurs.

Therefore, most of the constraint specification into our system should preferably be performed in some constant time, regardless of the number of constraints and elements.

4 Examples

Here, we give an example of a simple "step" feature that shows how the geometric constraint solver works and how conflicts are resolved. The meaning of the "step" feature may be represented by the basic structural constraints and the default dimensional constraints. In Figure 5, the "step" feature is implicitly given by coincident constraints for four faces, \( f_0, f_1, f_2, \) and \( f_3, \) to be located on the four faces of the base object, respectively, and is explicitly given some dimensional constraints to determine its shape. The step is then created by a boolean operation between the base and the step object.

![Figure 5: Example of a "step" feature](image)

4.1 Constraint evaluation

First, we consider the following three geometric constraints, which are necessary and sufficient to determine the shape of the step without conflicts:

\[
\begin{align*}
\text{distance}(f_4, f_3) & \iff \{B_1\}, \\
\text{distance}(e_3, e_9) & \iff \{B_2\}, \\
\text{angle}(f_3, f_4) & \iff \{B_3\},
\end{align*}
\]

where \( B_1, B_2, \) and \( B_3, \) and \( B_3, \) are respective assumptions for each constraint. Suppose that the faces of the base block are determined and have the assumptions, as shown in the figure: then \( f_1 \) and \( f_2 \) are determined by the structural constraints, and the nodes that belong to these face classes have the labels \( \{A_1\} \) and \( \{A_3\} \), respectively. The first constraint determines \( f_1, \) whose label is now \( \{A_3, B_1\}, \) and the second determines \( e_3, \) whose label becomes \( \{A_1, A_3, B_1, B_2\}, \) because \( e_3 \) depends on \( f_1 \) and \( f_2. \)

We focus on the face \( f_3 \) in order to explain how symbol reduction consumers work. This face is constrained by a pair of dimensional constraints, "distance\((e_9, f_3)\)" (that is, on \( e_9 \) and \( f_3 \)) and "angle\((f_3, f_4)\)." These constraints are converted into consumers that produce constraint symbols, such as:

\[
\begin{align*}
\text{distance}(e_9, f_3) & \iff \{f_3: \text{cyl or lin}\}, \\
\text{distance}(e_9, f_3) & \iff \{e_9: \text{pln}\}, \\
\text{angle}(f_3, f_4) & \iff \{f_3: \text{agl}\}.
\end{align*}
\]

Since \( e_9, f_4, \) and the constraints are determined and have nonempty-labeled nodes, these consumers invoke the following justifications:

\[
\begin{align*}
\text{(distance}(e_9, f_3)=0) & \iff \langle e_9=(P_1, V_1) \rangle, \\
\text{(angle}(f_3, f_4)=R_1) & \iff \langle f_3=(P_2, V_2) \rangle
\end{align*}
\]

where \( P_1 \) and \( P_2 \) are points on \( e_9 \) and \( f_4, \) respectively, \( V_1 \) is the direction vector of \( e_3, \) \( V_2 \) is the normal vector of \( f_4, \) and \( R_1 \) is the value of the angle between \( f_3 \) and \( f_4. \) These justifications add new labels for their consequents. The related reduction consumers are then applied, and the following justifications are invoked in order:

\[
\begin{align*}
\text{(f3:agl=(V_2, R_1))} & \iff \langle f_3=(P_1, V_1) \rangle, \\
\text{(f3:agl=(V_2, R_1))} & \iff \langle f_3=(P_1, V_1) \rangle
\end{align*}
\]

4.2 Disambiguation

As the labels of the two nodes in \( f_3 \) are updated, the following derivation rule of the topology is invoked:

\[
e_9 \iff f_3, f_4,
\]

which derives \( e_9 \)'s attribute values from \( f_3 \) and \( f_4. \) In this case, the rule calculates two attribute values for the edge \( e_9, \) and then invokes the justifications in order to propagate the labels as follows:
\( (f_3=(P_1, V_3)), (f_4=(P_3, V_5)) \Rightarrow (e_9=(P_1, V_5)) \)
\( (f_3=(P_1, V_3)), (f_4=(P_3, V_5)) \Rightarrow (e_9=(P_1, V_1)) \)

where "\( V_1 \)" is the reverse of "\( V_1 \)."

Now, \( e_9 \) has two conflicting nodes whose labels are \( \{\{A_1, A_3, B_1, B_2\}\} \) and \( \{\{A_1, A_3, B_1, B_2, B_3, C_2\}\} \), respectively. Since multiple nodes for any element always cause conflicts, the appropriate consumers give the following nogood to the ATMS:

\[
\text{nogood}\{A_1, A_3, B_1, B_2, B_3, C_2\}.
\]

Since this nogood triggers the hyperresolution rule, the label of "\( C_1 \)" is updated as described in Section 3.4:

\[
\{\{C_1\}, \{A_1, A_3, B_1, B_2, B_3\}\}.
\]

Consequently, the first node in the face \( f_3 \) is updated with its label from \( \{\{A_1, A_3, B_1, B_2, B_3, C_1\}\} \) to \( \{\{A_1, A_3, B_1, B_2, B_3\}\} \), and becomes active. Thus, the disambiguation is performed, and \( f_3 \) now has only one active node that represents a unique attribute value for \( f_3 \).

### 4.3 Over-constraint resolution

Next, we consider the over-constrained situation in which a new constraint is added to an established shape. This situation often occurs when the specification of a product is changed. Suppose that a new dimensional constraint is added between edges \( e_5 \) and \( e_{11} \):

\[
\text{distance}(e_5, e_{11}) = \{B_4\},
\]

where \( B_4 \) is the assumption automatically attached by the system. This constraint causes a new constraint-solving process and propagation, and then the following nogood may be reported:

\[
\text{nogood}\{A_1, A_3, B_1, B_2, B_3, B_4\},
\]

because three elements, \( f_3, e_9, e_{11} \), may have multiple attributes, as shown in Figure 6(a).

If the designer wishes to resolve the conflicts, he can find the following cancellation candidates:

\[
\{A_1\}, \{A_3\}, \{B_1\}, \{B_2\}, \{B_3\}, \text{and} \{B_4\},
\]

which are obtained from the above nogood. As mentioned in Section 3.5, there are two types of candidate: if \( A_1, A_2, \) or \( B_3 \) is canceled, then not only conflicting elements but also consistent ones have no nodes with nonempty labels. On the other hand, if \( B_2, B_3, \) or \( B_4 \) is canceled, then each conflicting element has only one nonempty-labeled node and some empty-labeled ones. Consequently, the system can show only the latter candidates to the designer. If he cancels \( B_3 \), the product will have a consistent shape, as shown in Figure 6(b).

![Figure 6: Example of conflict resolution](image)

The solution seems naive in this example, but if the shape is complicated, it is difficult for the designer to determine which constraints cause conflict. Therefore, the function described above is very important for an interactive modeling system.

### 5 Conclusion

We have proposed a new methodology for an ATMS-based geometric constraint solver in order to resolve two serious problems in conventional rule-based approaches, namely, poor handling of temporary conflicts and inefficiency. The straightforward application of the ATMS overcomes the first problem. Now, the system can, by minimal diagnosis, detect an over-constrained situation and advise the designers on which constraints should be canceled in order to resolve it. With regard to the second problem, we have shown that in our approach, the complexity becomes almost linear with respect to the number of constraints.

We have also shown how modification of the symbolic constraints affects the shape. We expect to apply our technology to an intelligent CAD system that can understand symbolic specification and can generate an object that satisfies the specification.

### Acknowledgments

We would like to express our gratitude to Messrs. H. Masuda and A. Okano for their technical help with the 3D modeler, and to Mr. H. Maruyama for his useful comments on this paper. We would also like to thank Dr. M. Koda for his management support of this research project.
References


