INTERACTION WITH CONSTRAINTS IN 3D MODELING

Wolfgang Sohrt, Beat D. Brüderlin
Department of Computer Science
University of Utah
Email: bdb@cs.utah.edu

ABSTRACT

The purpose of our research is to simplify and improve the effectiveness of the interactive definition of geometric objects in computer aided geometric modeling. To achieve this goal, two ways of defining geometric objects are combined and interfaced: 1) the definition of objects by graphical interaction and 2) the specification of objects by geometric constraints.

To demonstrate the practicability of the proposed approach, a geometric modeling system was implemented. In this system, interactive modeling operations automatically generate constraints to maintain the properties intended by their invocation, and constraints, in turn, determine the degrees of freedom for further interactive modeling operations. A symbolic geometric constraint solver is employed for solving constraint systems of constraints. Group hierarchies are utilized for representing dependencies and for localizing systems of constraints.

1 INTRODUCTION

Computer aided geometric modeling plays an important role in the industrial product design process. However, the definition of geometric shapes with today’s interactive CAD systems is still difficult and not always as natural as one would like. Most modeling systems are based on an operational paradigm, i.e., geometric shapes are defined by a sequence of construction operations (explicit modeling). These systems do not automatically ensure that the generated shape meets some specification, nor do they check whether an operation violates the intended properties of previous operations.

To overcome these problems, some modeling systems provide a way of specifying shapes by geometric constraints (implicit modeling). Since Ivan Sutherland developed the first constraint-based drafting system SketchPad [Sut63], various approaches have been taken to utilize constraints for computer aided design [Bar87] [Bor81] [Bru85] [Bru86] [Bru87] [Ea89] [Fuq87] [Gos83] [LG82] [LGL81] [Nel85] [Ros86] [SS80] [WFB87]. Constraints give users the ability to specify geometric properties of the models, for example, the length of a side, the volume of a solid, or the parallelism of two surfaces. Whenever a part of the model is modified, the system adjusts the rest of the model such that all the user-imposed constraints remain valid.

However, there are problems with purely constraint-based systems. Creating complete and consistent constraint specifications is difficult. Defining a complex model by constraints can be a cumbersome task, while applying some modeling operations would produce the same model in a simpler and more natural way.

The approach presented in this paper is to combine interactive manipulation (explicit modeling) with definition by constraints (implicit modeling) such that they supplement each other. Interactive modeling operations automatically generate constraints to maintain the properties intended by their invocation, and constraints, in turn, determine the degrees of freedom for further interactive mouse-controlled modeling operations.

Wherever possible and useful, constraints are treated as dependencies within a group hierarchy. Where this is not possible, a symbolic constraint solver is used. It returns the results in the form of geometric expressions, which are evaluated in a second phase to calculate the new shape of the specified objects.

2 INTERACTIVE DRAFTING OF 3D OBJECTS

In this section, an overview is given of how graphical, object-oriented user interaction can be applied and extended to become a useful tool for 3D drafting. The user interaction of a geometric modeling system should be as simple and intuitive as possible. In our system, geometric objects are therefore treated as physical objects that can be directly manipulated by drag actions in 3D (using a mouse), similarly to known 2D illustrations programs, as they exist for the Macintosh. However, the requirements for 3D drafting systems are more demanding:

- Drafting applications require exact data input (much more than generally necessary in illustration applications).
- 3D operations on 3D data need to be carried out on a 2D display and with a 2D input device (mouse). Solutions to this problem are suggested, for instance, in [NO86].

To overcome these problems, we provide the following solutions:

- 3D objects provide handles for interactive transformations (rotation, translation and scaling) in certain directions. Thus, object manipulations are achieved conveniently through mouse dragging.
- Unfortunately, dragging operations generally yield inaccurate translation, rotation and scaling parameters and therefore only approximate the intended manipulation (more like a sketch). One way of obtaining exact values is to provide some discretization (e.g., a grid) for these values.
- Another way of achieving exact interactive transformations is by defining them relative to other already existing objects that serve as a reference. Previous work describing this
approach in an interactive environment (snap-dragging by Eric Bier [Bie90]) demonstrates its power and usefulness.

- Yet another way of obtaining exact shapes is by assigning geometric dimensions (angles, distances, etc.) and relations (parallelism, incidence, etc.) to objects. We can use interactive mouse manipulations to first approximately shape and position the geometric objects (sketch), and then specify the exact dimensionings and relations (constraints). A constraint solver is employed for solving systems of simultaneous constraints.

A previous approach that combined interactive modeling and constraints is described in [Roe86]. One of the major goals of our research was to investigate further ways of making all the before-mentioned alternative approaches work together and supplement each other, rather than being separate, isolated tools. Each of the approaches is already very powerful by itself; through an appropriate integration we obtain an even more powerful tool.

3 INTERACTION WITH CONSTRAINTS

In current geometric modeling systems, the modeling operations operate on geometric objects and yield new (transformed) geometric objects as results. However, much of the interaction of those operations is not represented in the resulting objects, but only implicitly in the operation.

For example, when a phone is positioned on a table, its bottom face becomes coplanar to the table surface. This relationship is probably intentional and should be maintained by future operations. This means the phone can be translated only within that plane or rotated about an axis perpendicular to the plane. Even more so, when the table is moved, the phone should be moved with it. Unfortunately, conventional modeling systems do not remember such relations intended by geometric operations.

3.1 Implicitly Defined Constraints

The approach taken in this paper is to determine the intended properties of modeling operations and to maintain them during future manipulations. Not only does this avoid possible errors, but also, it will make the interaction more efficient. If done properly, the system operates according to the users' intentions without them explicitly stating these intentions all the time.

The idea is to keep the postconditions of geometric operations as constraints. Since they are not established explicitly by the user, but implicitly as a side effect of modeling operations, they are called implicit constraints. When constraints are imposed on objects or groups of objects (assemblies), they restrict the degrees of freedom for interactive manipulations. Constraints are visualized to make it intuitively clear to the user which operations are possible, and which are not. For example, if the rotational degrees of freedom of an object are restricted to an axis, this axis is displayed. The following Example 1 shows how constraints can be used to easily fit two geometric objects together.

Example 1. A T-shaped object (T) is to be assembled with a U-shaped object (U) with a slanted side (Figure 1) by positioning, orienting and resizing the T. First, to move the T towards the U, a point on T is selected that shall be matched with a second point selected on U (as indicated by the arrow in Figure 1).

So far, both objects are unconstrained. The system translates T towards U such that the two selected points match. After carrying out this transformation, a constraint is automatically created which ensures that the point selected on T stays fixed to the point on U (indicated by a black dot in Figure 2).

Now, T cannot be translated relative to U any longer, but it can still be rotated in all three directions about the fixed point. The three rotation handles displayed are centered at this point. To orient T, the user selects two points, as shown by the arrow in Figure 2. The system will apply a rotation about the fixed point on T such that the two half lines originating at the fixed point and each going through one of the two selected points will coincide afterwards.

A new constraint is automatically created which requires T to remain fixed with respect to the merged half lines (indicated by a thick black line in Figure 3).

Now, T can be rotated only about this line. In order to align T with the slanted sides of the U, another two points are selected, as indicated by the arrow in Figure 3, and the system will carry out a rotation about the fixed line such that the two selected points lie in the same half-plane originating at the fixed line (indicated by the dotted surfaces in Figure 4).

Now, the position as well as the orientation of T are determined and fixed. Only its size can still be changed (unless the user explicitly specified size constraints before; see Subsection 3.2). If two points on the fixed line are selected (as indicated by the arrow in Figure 4), the line is collinear with a body axis of T. Hence, the system carries out a 1D scaling along this axis with the fixed point determining the center of scaling. The result is shown in Figure 5.
Figure 3: \( T \) has been rotated about the fixed point. Further rotation is possible only about the fixed line.

Figure 4: \( T \) has been rotated about the fixed line. Scaling in all 3 directions is still possible.

The size of \( T \) is now constrained in the direction of the axis, but it can still be scaled in the other two directions with the fixed point remaining unchanged. The modeler displays handles for drag-scaling with the fixed point as the common center to indicate the remaining degrees of freedom.

Figure 5: \( T \) has been scaled along the fixed line. Further scaling is possible only orthogonal to the fixed line.

Figure 6: An object with fixed point, line and plane.

3.2 Representation and Specification of Constraints

Objects can be constrained such that a point, a line or a plane or a combination of them are fixed. In the following, the internal and visual representation of constraints and their specification is discussed.

3.2.1 Representation of Constraints

The constraints are represented by a so-called constraint coordinate system and a number of Boolean constraint flags. The constraint coordinate system is defined by a point (the origin), a line (the \( x \)-axis) and a plane (the \( xy \)-plane of the constraint coordinate system). The constraint flags determine whether the point, line or plane are constrained to be fixed for translation, rotation or scaling, and in which axis directions of the group coordinate system scaling is possible.

Initially, the constraint flags are all switched off, and the constraint coordinate system is aligned with the local coordinate system of the group. Both the constraint flags and the constraint coordinate system may be changed when constraints are imposed on the group.

The constraints imposed on objects are displayed graphically. A fixed point is shown as a thick dot, a fixed line as a thick line and a fixed plane as a rectangle (Figure 6). Additional visual information about the degrees of freedom of objects is provided by displaying the corresponding drag-handles (see Subsection 3.3).

3.2.2 Specification of Constraints

Constraints are installed either automatically by transform-to-match operations or through explicit editing by the user. Drag-operations do not automatically create constraints, since they are usually used for rough sketching, and the results are only temporary.

Transform-to-match operations result in the implicit definition of constraints on the transformed object that fix a point, line or plane. If a point has been fixed, this point becomes the new origin of the constraint coordinate system. If a line has been fixed, it becomes the \( x \)-axis of the constraint coordinate system. If a plane has been fixed, it becomes the \( xy \)-plane of the constraint coordinate system.

Example 1 shows how the various transform-to-match operations lead to new constraints.

The modeler provides an option for manually switching (on or off) the point, line and plane constraint flags of an object for trans-
lation, rotation and scaling. It is also possible for the user to impose additional restrictions on scaling.Scaling along the local group coordinate axes can be disabled, and the ratio between any two or all three directions can be restricted to remain constant. The dialog form for editing constraints is shown in Figure 7.

![Figure 7: The dialog form for editing constraints](Image)

### 3.3 How Constraints Are Enforced

Point, line, plane and scaling ration constraints on an object restrict the degrees of freedom for translation, rotation and scaling. The system displays drag-handles only in the directions of the object's degrees of freedom to ensure that constraints cannot be violated by drag-transformations. If the user defines a transformation by numerical values that would violate a constraint, the operation is not carried out, and an error message is issued to the user.

Constraints cannot only be used to prevent the user from accidently undoing the results of previous operations. Since constraints determine the degrees of freedom of objects, they restrict possible transformations and thus can also be exploited to reduce the amount of user input that is necessary.

To transform an object, the user indicates the origin and destination of the transformation by selecting two points. The system chooses a default operation (translation, rotation or scaling) from the object's degrees of freedom. For instance, if translation is not possible because of a point constraint, and if rotation is constrained by a fixed line, rotation about that line is chosen. The user simply confirms the action chosen by the system or specifies another transformation type, for example scaling. Using this automatic mechanism for choosing a transform-to-match operation under constraints, complex models can be assembled easily with a few mouse clicks, as Example 1 demonstrates.

The algorithms for enforcing constraints and using constraints to automatically determine the type of an operation are described in detail in [Soh91].

### 4 SIMULTANEOUS SYSTEMS OF CONSTRAINTS

#### 4.1 Overview and Example

The constraints described in Section 3 were defined implicitly as properties (postconditions) of modelling operations. These constraints are immediately satisfied, since they are generated after an operation has been carried out. They result in degrees of freedom and dependencies between objects that limit the possible manipulations and thus guarantee that future operations will not violate the constraints later on.

However, there are situations where constraints cannot be represented by a simple dependency hierarchy. Our modeler also facilitates explicit specifying, relaxing or changing additional constraint types, such as distances, angles, etc. between objects. The symbolic constraint solver described in section 4.3.1 is employed for the following tasks:

- automatic derivation of the transformations that are necessary to meet the constraints and
- detection of inconsistent constraint specification, i.e., subsets of constraints that cannot be satisfied simultaneously.

It turns out that for many applications, the constraints divide into independent subsets that are contained within some 2D hyperplane, which is called a constraint plane in this paper. Hence, the authors equipped the modeler with a 2D constraint solver.

The definition and solving of constraints is done in the following order:

1. A group of objects is selected among which a constraint network is to be defined or modified.
2. The system tries to determine a constraint plane from the degrees of freedom of the objects of the selected group. If this is inconclusive, the user is asked to interactively define the constraint plane.
3. For each object, the modeler determines its degrees of freedom within the chosen constraint plane and from that derives a set of predetermined 2D constraints.
4. Additional constraints may be specified and added using the input form shown in Figure 8. The constraints are defined between points of the objects projected onto the constraint plane.

![Figure 8: The form for editing simultaneously defined constraints](Image)

5. Once the constraint network is complete, the constraint solver is started. It operates in two phases:

- The first phase finds a symbolic solution for each point involved in the constraint network. The symbolic solution is a prescription of how to construct the point geometrically. For example, if point A is defined by the distances \(d_{AB}\) and \(d_{AC}\) to point B and point C, resp., for which symbolic solutions have been found earlier, the symbolic solution for A would describe the intersection of two circles:
  \[ A = \text{intersection}(\text{circle}(A, d_{AB}), \text{circle}(C, d_{AC})). \]
If the constraint network is overconstrained, a warning is issued to the user that not all constraints can be satisfied. This first phase is run only if the constraint network has been structurally altered by adding or removing constraints.

- The second phase evaluates the symbolic solutions found in the first phase to calculate a numerical solution. Each point involved is projected onto the constraint plane, the numerical solution within the 2D constraint plane is calculated, and the point is projected back into 3-space, using the previous distance from the plane. Then, the objects to which the points belong are adjusted by translation and rotation in order to match the new point positions. Scaling is not used for satisfying simultaneous system of constraints; objects are treated as rigid.

In the case of ambiguities (for example in the case of two solutions for the intersection of two circles), the solution that is closest to the previous position and orientation is chosen. The second phase is run every time a numerical value either in the constraint definitions or in the dimensions of the objects has changed, or after the first phase was run.

Example 2 demonstrates the use of a simultaneous system of constraints.

Example 2 A toy device, the so-called Connecticut “do-nothing” machine or “treadmill,” is to be constructed. It consists of a board with two notches orthogonal to each other, two sliders that run in these notches, and a handle whose one end is attached to one slider and whose mid subsection is attached to the other slider. If the handle is turned, the sliders move back and forth along the notches.

The parts of the treadmill are shown in Figure 9. The board with the notches is constructed by generating five blocks, dimensioning them to their proper size, and assembling them, using matching operations. A Boolean operation (union) is carried out to merge the blocks into one solid.

For the sliders, cubes are instantiated, translated, rotated and scaled, such that they fit into the notches. To position them, they are attached to the corners of the notches. Afterwards, their translation constraints in the directions of their notches are relaxed. The handle bar is created by instantiating another cube, scaling it to the appropriate dimensions, placing a cylinder on one end as the handle, and creating two cylinders as bolts and placing them on the opposite end and in the middle of the bar. Finally, reference points are installed on the sliders and the bolts.

When the two sliders are selected, the modeler automatically determines the constraint plane that contains their translation degree of freedom.

For each slider, the modeler automatically creates a slope constraint between the reference point on the slider and a point outside the model. These slope constraints restrict translation of the sliders to their linear degrees of freedom. The distance between the two bolts is also automatically constrained to the current distance value to ensure that the handle bar remains rigid.

To attach the handle bar to the sliders, the user needs to define only a few additional constraints: The reference point of each bolt needs to be merged with the reference point of one slider. Another constraint that has to be specified explicitly is the slope of the handle bar. It is defined between the reference points of the bolts and given the symbolic parameter name alpha. For the numerical evaluation, alpha must be given a value later.

Finally, the user starts the solver. Phase 1 finds a symbolic solution, and phase 2 uses it to calculate the new 2D coordinates of the constrained points. The points are projected back into 3-space, and the objects are adjusted to the new point positions. Figure 10 shows the treadmill after this adjustment.

Figure 9: The treadmill parts and the constraints defined on them. The two "merge points" constraints and the slope constraint on the handle bar's bolts were defined by the user. The distance constraint between the bolts, the reference points outside of the model and the slope constraints between the reference points and the sliders were derived automatically from the degrees of freedom.

Figure 10: The treadmill after the solving process

If the numerical values of the constraints are changed, for example by assigning a new value to the parameter alpha, or if the interior shape of one of the involved objects is modified, only the second phase of the solver has to be started again, using the symbolic solution from the previous solving process.

In Figure 11, the bolt in the middle of the handle bar was translated along the bar. Thus, the interior shape of the handle bar is changed such that the distance between the bolts is increased. Phase 2 of the constraint solver is run again to compute the new configuration.

Figure 12 shows four positions out of an animated sequence created by letting alpha run from 10 to 360 degrees in 10-degree increments. Only phase 2 of the solver is run for each parameter change.
4.2 Generating Symbolic Constraints Automatically from Degrees of Freedom

The degrees of freedom of the objects that are to be constrained by a simultaneous system of constraints are exploited by the modeler for two purposes: automatically determining the 2D constraint plane within which the system of constraints is specified, and deriving the 2D constraints imposed on the objects have within that constraint plane.

Table 1: Predicates and their order for the 2D solver

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Order</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>d(P1,P2,dst)</td>
<td>6</td>
<td>distance between P1 and P2 is dst</td>
</tr>
<tr>
<td>a(P1,P2,P3,ang)</td>
<td>5</td>
<td>angle (P1,P2,P3) is ang</td>
</tr>
<tr>
<td>s(P1,P2,slp)</td>
<td>4</td>
<td>directed slope of vector (P1, P2) is slp</td>
</tr>
<tr>
<td>tr(P1,P2,P3,trg)</td>
<td>3</td>
<td>unsized triangle (P1, P2, P3) is trg</td>
</tr>
<tr>
<td>v(P1,P2,vec)</td>
<td>2</td>
<td>vector (P1, P2) is vec</td>
</tr>
<tr>
<td>p(P1,pos)</td>
<td>1</td>
<td>position of point P1 is pos</td>
</tr>
<tr>
<td>false</td>
<td>0</td>
<td>contradiction detected</td>
</tr>
</tbody>
</table>

Figure 11: A bolt has been translated

Figure 12: The treadmill with four different values for alpha
The order of the left hand side is the sum of the individual orders of its predicates, \(4+6 = 10\), and the order of the right hand side is 2. Hence, we obtain the rewrite rule
\[ s(A, B), d(A, B) \rightarrow v(A, B). \]

The rewrite rule is extended with a prescription on how to derive the geometric expression of the right side from the left side:
\[ s(A, B, S), \ d(A, B, D) \rightarrow v(A, B, sd2v(S, D)). \]

sd2v is the symbol for a function that computes an offset vector from its slope and length.

Constraints are expressed in terms of geometric predicates. They are solved by repeatedly finding predicates in the constraint network that are matched by the left side of a rule and replacing them with the predicates of the right side. Since the rule was gained from an equation, the new constraint network is equivalent to the old one, but with a simpler representation. Each rule is a simplification rule in the sense that it lowers the summed order of the constraint network. If the algorithm succeeds, only position constraints on points remain, that is, the coordinates of all points have been determined.

In addition to the rules, a set of equivalences is needed. An example for such an equivalence is that \(d(A, B)\) is equivalent to \(d(B, A)\). Equivalences cause a slowdown in terms of execution speed, because they require extensive backtracking. For example, if no rule's left side matches with \(d(A, B)\), the solver also tries again with \(d(B, A)\), and that, of course, takes extra time. The number of equivalent combinations of predicates increases polynomially with the number of constraints.

The advantages of the rewrite rule constraint solver are:
- The solving itself is purely geometrical; numerical values are only used to display the results.
- Rules that detect inconsistencies due to overdetermined systems can be expressed in the same mechanism [Brau87].
- Since it returns symbolic results, all solutions are found, and the program can pick the "best" one by some criteria.
- It finds degrees of freedom and rigid parts (that is, parts with points completely constrained relative to each other).

Disadvantages are:
- The predicate matching requires backtracking and therefore becomes the slower, the larger the constraint network is.
- For each set of rules implemented in the constraint solver, constraint graphs exist which cannot be solved by these rules, even if a solution exists. Non-constructive, numerical solvers might find such solutions.
- Equivalences must be introduced. They cause polynomial increase of searching time in the number of predicates when trying to match predicates to the left side of a rule.

### 4.3.2 Phase 1: Finding a Symbolic Solution

The constraints are sent to the constraint solver in the form of Prolog predicates. The constraint solver subsequently applies the rewrite rules to these predicates in order to replace them by equivalent predicates of a lower order, until no further application of rules is possible. At this, the last argument of each newly asserted predicate is a symbolic expression in terms of the expressions associated with the replaced predicates. This expression describes the geometric construction by which the point can be computed. The geometric relation represented by the predicate can be computed. Common subexpressions are extracted and just referenced, which speeds up the solving and the numerical evaluation, and which is important in the case of ambiguous solutions (see Section 4.5).

If the solving is successful, the rewrite rules convert all predicates into point position predicates. However, if the constraint network was underdetermined, there won't be a position predicate for each point, and some other predicates may not have been replaced. If the network was overdetermined, there are always unused predicates in addition to the position constraints. These unused predicates are displayed to the user, since the corresponding constraints were not satisfied during the solving.

### 4.3.3 Phase 2: Evaluating the Symbolic Solution

Phase 2 of the constraint solver uses the symbolic solution produced by phase 1 and numerically evaluates it to compute the position and orientation of the constrained objects. It is run in the following cases:
- After Phase 1 was run, changing the symbolic solutions.
- The interior of an object has changed, for example, if it is a group and a subgroup has been scaled.
- A constraint is defined by a parameter, and its value has changed, for example during an animated display of a model of a mechanical device.

For every point constrained by simultaneous constraints, the corresponding entry in a database of evaluated geometric relations is looked up. Three cases are possible:
- The point has a position constraint. In this case, the value (point coordinates) of the constraint is assigned to it.
- The point has already been evaluated as an intermediate step in another computation. In this case, the database entry contains the numerical value of the point.
- The point has not been evaluated yet. In this case, the relation contains a symbolic description of the geometric construction by which the point can be computed. This geometric construction involves other points, vectors, slopes, etc., which may either be given by constraints, evaluated earlier, or unevaluated, in which case they have to be evaluated as intermediate results first.

Some of the geometric constructions do not necessarily yield a unique solution. For example, the intersection of two circles may yield zero, one or two points. If the intersection yields zero points, the user is given a warning that the numerical evaluation failed. If the intersection contains one point (i.e., the circles are tangent), the solution is unique. If it contains two points, topological information from the original model is used to choose the solution that changes the model as little as possible.

A more detailed description of the two phases of the constraint solver is given in [So91].

### 4.4 Adjusting Objects to the Results of the Constraint Solver

Each time the numerical evaluation of the constraint system (Phase 2) is rerun, the involved objects have to be adjusted to the solution afterwards. The new 3D coordinates of all constrained points are computed by projecting their new position from the constraint plane back into 3-space, such that they have the same distance from the plane as in the original model.

If only one point of an object was constrained and its position has changed, the object is translated so that the new position of the point is matched or (if the object's translation is fixed by an implicitly defined construction constraint) rotated about the fixed point so that the new orientation is matched. (If both rotation
and translation were fixed, the point would not have been changed during the solving.)

If two points of the object were constrained, their new positions uniquely determine the object's new position and orientation. The automatic compliance to implicitly defined constraints and the rigidity of objects ensures that the object is transformed only within its degrees of freedom, and that any further constrained points of the same object are transformed into their correct positions by adjusting the object with respect to the first two points.

To ensure that the result of the solution process is not undone by future interactive operations, restrictions on the degrees of freedom, represented by the constraint coordinate system, are imposed on the objects (the same way as for transform-to-match operations). However, a flag is set with the new implicit constraints that they are the result of symbolic constraint solving and are already represented by symbolic constraints. If the symbolic constraint solver is run again later (because the constraint network has been modified structurally), they will not be converted back into symbolic constraints.

4.5 Ambiguous Solutions

In some cases, multiple (finitely many) possible evaluations to a symbolic solution exist. In this case, Phase 2 of the constraint solver maintains the topological relations in the model as it was before Phase 2 was started.

Example 4: In the linkage shown in Figure 13, the bars, the board and the slider are connected by the bolts, which act as rotational joints. Once the position of the slider is determined, there still exist two solutions for the bars (as shown in Figure 13 and Figure 14). Phase 2 of the solver determines whether the bolts were in clockwise or counterclockwise order before Phase 2 was started, and it chooses the solution that maintains this order.

Figure 13: The linkage of Example 4

For dealing with ambiguities, it is important that all intermediate results are evaluated only once, so that all subsequent computations use the same solution. Another advantage of this approach is an improvement in speed by avoiding redundant computation.

5 CONSTRAINTS AND GROUPS

The group hierarchy plays an important role in establishing and enforcing constraints. It represents dependencies and localizes networks of simultaneous constraints.

5.1 Implicitly Defined Constraints and Groups

Implicitly defined constraints are always defined between the constrained group and its immediate supergroup (which is possibly the worldgroup). They restrict the transformations of the group relative to the supergroup it is dependent on. Group hierarchies are therefore established automatically along with constraints during a construction operation.

If a so far unconstrained group is constrained relative to another group (the so-called reference group), it becomes logically dependent on that group. This dependency is represented in the hierarchy by making the transformed group a subgroup of the reference group.

Single objects cannot have subgroups. If such an object is used as a reference, the transformed group is made a dependent of a new common supergroup that is rigidly connected to the reference object. The other subgroups of the new supergroup thus behave like dependents of the reference object.

Example 5: The robot finger shown in Figure 15 is modeled using transform-to-match operations. The group hierarchy shown in Figure 16 is established automatically.

Figure 15: A model of a robot finger

Initially, the three parts Part 1, Part 2 and Part 3 are unrelated subgroups of the worldgroup. To join Part 1 and Part 2, Part 2 is translated such that one of the corners of the joint matches a corner of the corresponding joint of Part 1 (Figure 17). Logically, Part 2 thus becomes a dependent of Part 1. However, since Part 1 is an atomic object and hence cannot have subgroups, internally
5.2 Simultaneous Constraint Systems and Groups

Whereas implicitly defined constraints directly express dependencies that are naturally represented by a group hierarchy, simultaneous systems of constraints can be utilized to express relations between subgroups of the same supergroup.

A great advantage of the group hierarchy is that it localizes the constraint networks. Every group may feature a constraint network between its subgroups, but on the average, these local networks will be small. Since the time needed for solving a constraint network can grow substantially with the number of constraints specified, localizing the constraint solving improves the speed enormously. Localized, small constraint systems are also easier to understand for the user. In addition, for each such constraint network a different constraint plane may be used.

The numerical evaluation of constraints is done by traversing the group hierarchy bottom-up (in post order traversal). Whenever the shape of a group is changed, the solutions of all its intermediate supergroups must be reevaluated. (The symbolic solutions themselves, however, do not need to be deduced again.) Thus, at each level in the hierarchy, subgroups of a group are treated as rigid objects.

Example 6 Figure 19 shows the group hierarchy of the linkage of Example 4. Assume there is a distance constraint defined between Bolt 2b, which can slide along Bar 2, and Bolt 2a, which is constrained completely to Bar 2. This distance constraint is defined within the interior of Bar 2 (constraint network 1).

If the distance is increased, first the interior of Bar 2 is adjusted by reevaluating the symbolic solution of the constraint system defined on Bolt 2a and Bolt 2b (i.e., Bolt 2b is translated along Bar 2). Then, the group hierarchy is traversed upwards, and Bar 1 and Bar 2 are adjusted by reevaluating the symbolic solution of their constraint system (constraint network 2). Figure 20 shows the configuration of the linkage before and after the distance constraint is increased.

6 CONCLUSION

By integrating modeling operations with graphical user interaction, constraint solving and group hierarchies, our research presents a step forward towards an easy-to-use interactive 3D geometric modeling system. Objects can be created, translated, rotated and scaled according to exact specifications with a few mouse-clicks and dragging operations. The intermediate results of such operations are automatically fixed by constraints, so that future modifications will not violate previously fulfilled design requirements. Dependencies between objects are represented by a
Figure 20: The distance between the bolts of Bar 2 is increased. The dotted lines show the configuration after the change.

Constraints are not only helpful for constructing models, but they are seen as an integral part of the model. This is particularly useful for models of assemblies with moving parts. Also, by integrating constraints into the model, design teams can communicate incomplete specifications for further work on the model. Constraints ensure that earlier design decisions are not violated by modifications in later design stages. Recipients of model data can not only look at the components of the model, but also see how these components are attached to one another. They can modify the model within its degrees of freedom and even view an animated display of the working model.

7 ACKNOWLEDGMENTS

We would like to thank Shiaofen Fang for implementing a first version of the symbolic constraint solver.

This work was supported in part by the National Science Foundation (Grant DDM-89 10229). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

“Macintosh” is a trademark of Apple Computers, Inc.

References


[Bor81] A. Borning. The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. ACM Toplas, 3(4), October 1981.


