Interaction with Constraints in 3D Modeling‡

Wolfgang Sohrt, Beat Brüderlin
Computer Science Department
University of Utah
Salt Lake City, UT 84112,
bruderlin@cs.utah.edu
Technical Report Number: UUCS-90-024

November, 1990

‡This work was supported in part by NSF grant # DDM-8910229

Interaction with Constraints in 3D Modeling

Wolfgang Sohrt, Beat Brüderlin

Department of Computer Science, University of Utah

Interactive geometric modeling is an important part of the industrial product design process. This paper describes how constraints can be used to facilitate the interactive definition of geometric objects and assemblies. We have implemented a geometric modeling system that combines the definition of objects by interactive construction operations and specification of geometric constraints. The 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. A symbolic geometry constraint solver is employed for solving systems of constraints.

1 Introduction

Today's modeling system are mostly based on an operational paradigm, i.e. geometric shapes are defined by a sequence of construction operations (*explicit* modeling). Such systems do not automatically ensure that the generated shape meets some specification, nor does it check whether an operation violates the intended properties of previous operations.

A much smaller number of modelers are based on constraint solving systems [SUT63, BOR81, LIN81, LIG82, GOS83, NEL85, BRÜ86, BRÜ87, BAR87, FUQ87, BOR88, EAT89, etc.]. Here, a geometric shape is defined by geometric dimensions, such as distance, angles, parallelism of lines and surfaces, etc. (*implicit* modeling). This way, a definition can be checked for consistency. Unfortunately, as soon as one tries to construct anything more than trivial examples, constraint specification and solving becomes very complicated. Even if the constraint solver is capable of handling arbitrarily complex specifications, it is still difficult for the user to keep track of what is already defined, why some specifications are contradictory, and

how to resolve the inconsistencies that are likely to occur. In most cases it is much easier to just apply a powerful modeling operation where it would be difficult to come up with a complete specification for the same object.

In this research we investigated how the two seemingly opposite approaches of explicit and implicit modeling can be integrated effectively in an interactive environment.

2 Interactive Drafting of 3D Objects

In this section we give an overview of how graphical, object-oriented user interaction can be applied and extended to become a useful tool for 3 drafting. The user interaction of a geometric modeling system should be as simple and intuitive as possible. In our system, geometric operations are therefore not defined by entering commands and parameters in some command language, but by treating the objects on the screen as physical objects that can be directly manipulated by drag actions in 3-D (using a mouse), similarly to known 2-D illustration programs, as they exist for the Macintosh. However, the requirements for 3-D drafting systems are more demanding:

- Drafting applications require exact data input (much more than generally necessary in illustration applications).
- 3-D operations on 3-D data need to be carried out on a 2-D display and with a 2-D input device (mouse). Solutions to this problem are suggested, for instance, in [NIE86].

To overcome these problems, we provide the following solutions:

- 3-D objects provide handles for interactive transformations (rotation, translation and scaling) in certain directions (see Example 1 & 2 below). 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 a (user definable) 3D 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 [BIE90] demonstrates its power usefulness.

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

A previous approach that combined interactive modeling and constraints is described in [ROS86]. One of the major goals of our research was to investigate further ways of making all the 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 Interfacing Interactive Manipulations with Constraint Specification.

Whenever a geometric operation is carried out, we wish to keep the intended resulting geometric relationships, and these should not be violated by subsequent operations. E.g. when we position an object on a table, one of its faces becomes coplanar to the table surface. Future operations should maintain the coplanarity (i.e. we can translate the object within that plane or rotate it about an axis perpendicular to the plane). Unfortunately, conventional modeling systems don't remember such properties of previous operations. The approach taken here is to determine the properties of the modeling operations and to maintain them during further manipulations. Not only does this avoid possible errors, but also, it will make the interaction more efficient. If done properly, the system does what the user means without him explicitly stating his intentions all the time. The idea is to keep the postconditions of previous manipulations as constraints. When constraints are imposed on objects or groups of objects (assemblies), they restrict the degrees of freedom for manipulations. These constraints will be visualized and enforced during interactive, direct manipulations. It must be intuitively clear which operations are possible, and which are not (A rigid object, for instance, must look rigid; if the rotational degrees of freedom are restricted to an axis, this axis can be displayed, etc.).

Example 1

This example demonstrates how the principles introduced above can be used to assemble objects. A T-shaped object is to be fit into a U-shaped object with a slanted side (Fig. 1). First, we select one point on the T that shall be matched with a second point on the U (as indicated by the arrow in Fig. 2).

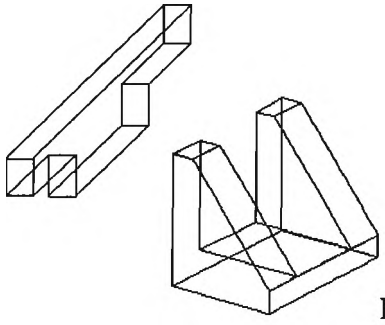

Fig. 1

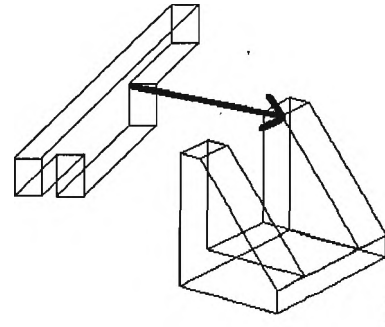

Fig. 2

Since so far the T is not constrained, a translation is possible. The system moves the T towards the U such that the two selected points match. By carrying out this translate-to-match transformation, a constraint is automatically created which ensures that the point selected on the T stays fixed to the point on the U (indicated by a black dot in Fig. 3).

Now, the T cannot be translated relative to the U any longer (but it can still be rotated about the fixed point). Hence, when we select the next two points as shown by the arrow in Fig. 3, the system will, by default, apply a rotation transformation 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.

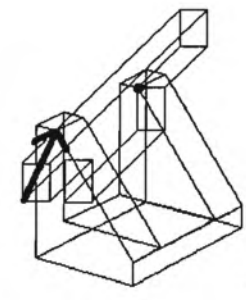

Fig. 3

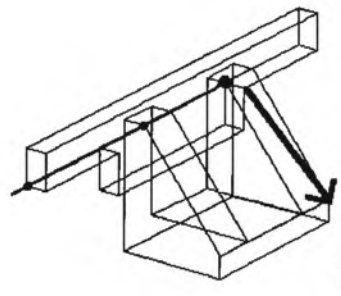

Fig. 4

A new constraint is automatically created that requires the T to remain fixed with respect to the merged half lines (indicated by a thick black line in Fig. 4). T can still be rotated, but only about this line. In order to align the T with the slanted sides of the U, we select the two points indicated by the arrow in Fig. 4, 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 grayed surfaces in Fig. 5).

Now, the position as well as the orientation of the T are fixed. Only its size can still be changed (unless the user explicitly specified size constraints before). We select two points on the fixed line, as indicated by the arrow in Fig. 5. The line is collinear with a body axis of the T. Hence, the system carries out a one-dimensional scaling along this axis with the fixed point determining the center of scaling. The result is shown in Fig. 6.

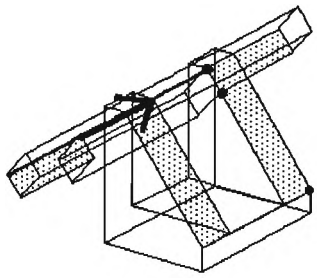

Fig. 5

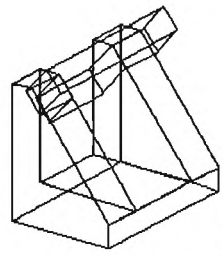

Fig. 6

The size of the 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 two handles for drag-scaling with the fixed point as the common center (Fig. 7).

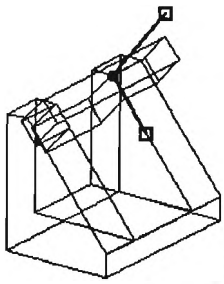

Fig. 7

To scale the object interactively, the user places the mouse cursor on the appropriate handle and drags it towards or away from the fixed point, until the object has reached the desired size in that direction. As an aid to obtain exact

values, the scaling can be done in discrete intervals (grid), and dimension numbers are displayed while dragging. To obtain a finer resolution than it is possible with the mouse input, the user can correct the displayed dimension by typing in intermediate values. After the scaling has been carried out, the new dimension can be turned into a constraint by clicking on a corresponding check-box.

4 Constraints, Dependencies and Object Hierarchies.

Geometric objects in modeling applications are often logically grouped and structured in hierarchies (assemblies). Operations can be defined for a single object, as well as for a group of objects. In the following we show how constraints between objects can also lead to hierarchical structures in the cases where we can derive a clear dependency of one object from another.

Constraints are internally represented as constraints between an object and the group in which the object belongs (or relative to the world coordinate system group, which is the implicit group to which all objects belong by default). Objects of a subgroup are the dependents of their super group. Whenever a constraint is defined between two unrelated objects, a group hierarchy is established automatically, as shown in the following example.

Example 2

The robot finger in fig. 8 has been constructed by matching the leftmost edge of part 2 to the rightmost edge of part 1 and then matching the leftmost edge of part 3 to the rightmost edge of part 2. These matching operations generate constraints between objects automatically, as described in the previous section. The resulting degrees of freedom between adjacent parts are rotation axes which are visually indicated by the rotation handles in fig. 8.

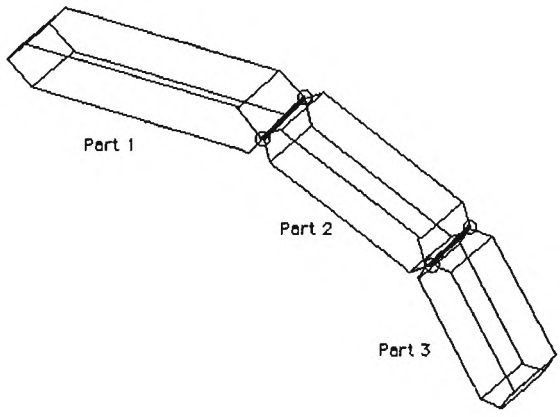

Fig. 8

Induced by the matching operations also a group hierarchy is generated automatically. The rule applied to generate the hierarchy is that the object being moved will become the dependent of the reference object relative to which it has been moved. Logically, part 2 is a subgroup of part 1, part 3 is a subgroup of part 2. Internally, a representation as depicted in fig. 9 was chosen. Part 1 is completely constrained relative to the new group 'Finger', part 2 is completely constrained to the new group 'Part 2 & Part 3' which can be rotated about an axis relative to group 'Finger", but not translated. Part 3 can be rotated about the axis between part 2 and part 3.

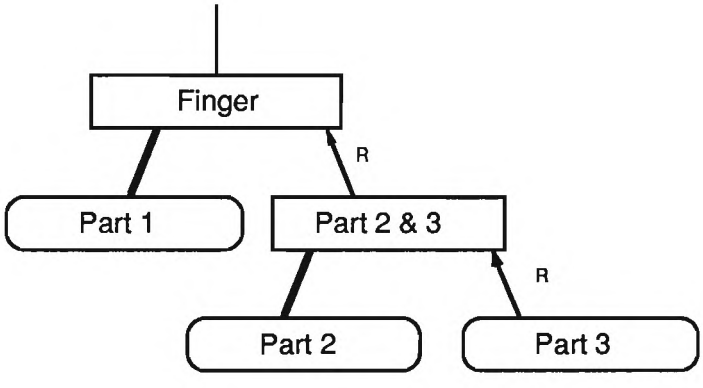

Fig. 9

The circular handles between the parts in Fig. 8 indicate the rotational degrees of freedom. If the user rotates part 2 about the joint between part 1 and part 2 by dragging the appropriate handle, part 3 will follow. If the joint between part 2 and part 3 is rotated, only part 3 will rotate. Part 1 can be freely translated and rotated, and the other parts will follow.

This simple model of object dependencies is very useful for many construction operations and for simple mechanical systems, such as manipulators. Mechanical systems like linkages, however, can get very complex, and a more sophisticated approach, such as it is described in the next section, is necessary.

5 Explicitly Defined Constraints

So far we described how constraints can defined implicitly as properties (postconditions) of a modeling operation. These constraints are always satisfied, since they are generated after an operation has been carried out. Degrees of freedom limit the possible manipulations, and thus constraints guarantee that future operation will not violate previously defined constraints. Our system also provides the possibility of explicitly specifying, relaxing or changing constraints any time. For instance, we may want to dimension distances, angles, etc. of a not yet completely constrained shape or assembly. A symbolic constraint solver is employed for the following tasks:

- Automatically derive the transformation on the objects that are necessary to meet the constraints.
- Compute the degrees of freedom for future interactive manipulations.
- Detect if a subset of the constraints cannot be satisfied simultaneously, i.e., is contradictory.

The symbolic constraint solver simplifies a given set of constraints by applying geometric rewrite rules, derived from axioms of Euclidean geometry. The result is a symbolic form of geometric operations. A short description of the constraint solver is given in the next section; for a more comprehensive treatment of the subject we refer to [BRÜ88] and [BRÜ90].

Many practical 3D construction examples have in common that the constraints are defined in some two dimensional hyperplane. In our current implementation we therefore restricted ourselves to apply a two-dimensional constraint solver. (Some simple generalizations to 3D constraint solving have been proposed in [BRÜ86] and [BRÜ87]). The mentioned hyperplane usually doesn't have to be specified explicitly by the user, but is already defined by the degrees of freedom of the objects involved. For instance, if a rotational degree of freedom is defined for an object, future constraints will automatically be interpreted as being defined relative to a plane perpendicular to the rotation axis. In other examples the plane for constraint solving is defined by

constraining a face of an object to be coplanar or parallel to some face of an object higher in the hierarchy. Before solving the constraint system, the program has to make sure that the constraints are in a common plane (or in parallel planes). All points that are to be constrained will be projected onto that plane, the constraints will be solved within the plane, and the 2D solution coordinates will be mapped back into 3-space.

We provided the following 2D predicates for specifying constraints:

- position of a point,
- distance between two points,
- directed slope of the line between two points, and
- angle between two lines.

This set of predicates is sufficient for the specification of a wide range of constraints. For example, to make a line vertical, a slope constraint can be imposed on its end points.

The constraints are defined by choosing the appropriate constraint type, selecting the points for which the constraint shall be imposed, and specifying the constraint value. If no value is specified, the constraint will assume the value that will keep the current state. For instance, if a distance constraint is to be specified between two points, and the user selects the two points, per default their current distance (projected onto the constraint plane) will be constrained, unless the user types in a new value.

We decided that explicitly defined constraints between two objects shall not affect the shape of either of them. The objects are assumed to be rigid. This fact must be made known to the constraint solver. Thus, if two points of the same object are constrained, the modeler automatically generates a distance constraint between them.

6 The Constraint Solver

Constraints are relations between points which may be expressed by predicates on these points. The predicates may be stored in the database of a Prolog interpreter as so called 'facts'. Here is a list of some geometric predicates on points, as used in our approach:

```
p(P, Pos).
d(P1, P2, D).
s(P1, P2, Alpha).
```

The semantics of the predicates is defined as follows:

To express that the position of a point *P* is known we use the predicate *p(P, Pos)*. This means that the position of *P* is defined by the value of *Pos* to which a symbolic expression may be assigned. Possible expressions are, e.g. a pair of coordinates *[10, 0.5]*, or a geometric function, such as *intersection(line(p(P2), 90), circle(p(P3), 20))*, meaning that the point is located at the intersection of a line through another point *P2* with a slope of 90 degrees toward the x-axis, and a circle with center point *P3* and a radius of 20.

The predicate *d(P1, P2 Dist)*, expresses that we know the distance between two points. For the value of *Dist* we may again write a constant number, a parameter, or a function such as *length(vector(P3, P4))*, i.e. the distance of two points with already specified position.

s(P1, P2, Alpha) constrains the slope of the line connecting *P1* and *P2* by the angle *Alpha* to the x-axis (measured counterclockwise).

To derive the position of points that are not explicitly specified by a corresponding predicate, we may apply rules known from constructing with compass and ruler. An example: given the positions of two points *P1* and *P2*, the distance between *P1* and a third point *P3* and the distance between *P2* and *P3*, we may construct *P3* by intersecting two circles. We first write the precondition of the rule by a conjunction of predicates:

p(P1, [Pos1]) ∧ p(P2, [Pos2]) ∧ d(P1, P3, R1) ∧ d(P2, P3, R2)

The position of the third point *P3* is found by intersecting two circles with centers *P1* and *P2*. This is expressed symbolically by:

p(P3, intersection(circle(p(P1), R1), circle(p(P2), R2))).

We formalize this construction rule by introducing the following notation for rewrite-rules: Conjunctions of predicates are expressed as lists of predicates '[ ]', and the symbol '->' indicates the direction in which the rewrite-rule is applied. The above rule is represented by the following rewrite-rule:

```
[p(P1, [Pos1]), p(P2, [Pos2]), d(P1, P3, R1), d(P2, P3, R2)]
-> [p(P1, [Pos1]), p(P2, [Pos2]),
    p(P3, intersection(circle(p(P1), R1), circle(p(P2), R2)))].
```

A a more comprehensive description of the constraint solver can be found in [BRÜ87], [BRÜ88] and [BRÜ90b].

7 Interfacing Between Implicitly and Explicitly Defined Constraints

Solving explicitly defined constraints should, of course, not violate the constraints derived as a postcondition of modeling operations (implicitly defined constraints), and modeling operations should not violate explicitly defined constraints. Whenever the symbolic constraint solver is invoked, the implicit constraints on the objects are turned into temporary explicit constraints on their constrained points. Then, the constraint solver is started.

For the constraint solver all objects are regarded as being rigid objects. This simplifies the user interaction and constraint solving dramatically. It means, for example, that implicitly the distance between two points of the same object is automatically constrained. Also, points of an object higher in the hierarchy are completely constrained relative to the points of an object lower in the hierarchy.

Figure 10 gives an overview of how the various parts of the system are interrelated.

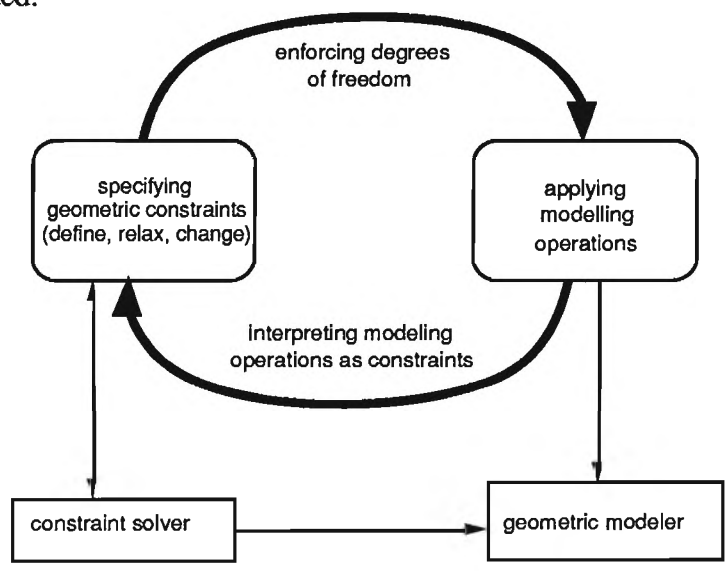

Fig. 10

The constraint solver finds symbolic expressions for the positions of the points specified by the constraints. These expressions are the numerically evaluated and directly translated into transformations on the objects involved. If several points of the same object are constrained, the transformations are interpreted as follows: The computed new position of the first point leads to a translation that maps the old position into the new one. The position of the second point in the same object yields a rotation. In the case where the degrees of freedom are two dimensional, two points is all we can specify for one object. For three dimensional degrees of freedom, one additional rotation may be specified. If more points shall be specified by constraints, an additional scaling operation may have to be carried out to satisfy the constraints. Since we limit ourselves to rigid objects, the constraint solver would report an overconstrained configuration. in this case.

Example 3

As an example for demonstrating the interaction between the constrain solver and the interactive part, we construct a toy device, called "treadmill" (fig. 11). It consists of a board with two notches orthogonal to each other, two sliders that run in those notches, and a handle whose one end is attached to one slider and whose mid section is attached to the other slider. If we turn the handle, the sliders move back and forth along the notches.

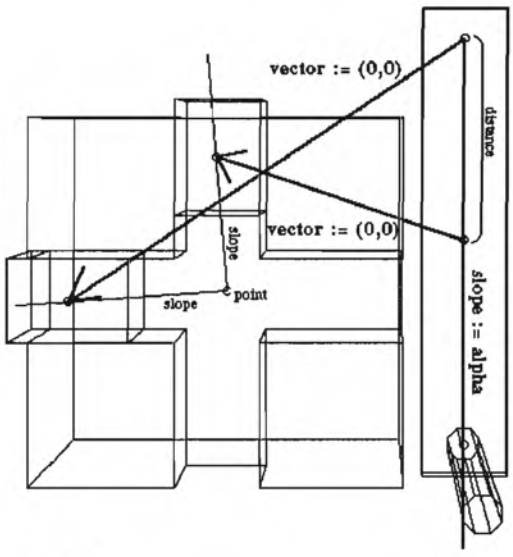

Fig. 11

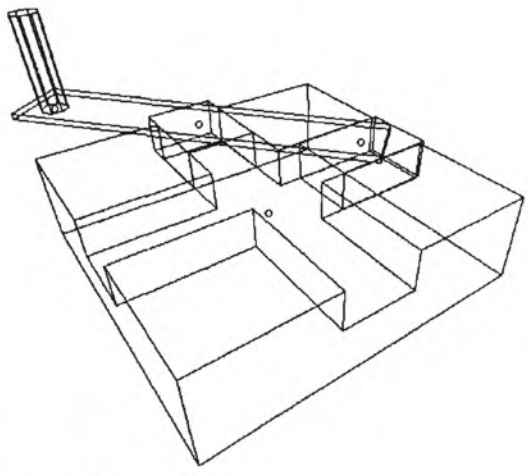

Fig. 12

First we construct the board with the notches. We generate five blocks dimensioned to their proper size and assemble them, using matching operations. A boolean operation (union) is carried out to merge the blocks into one solid. For the sliders, we instantiate cuboids, translate, rotate and scale them, such that they fit into the notches. To position them, we first attach them to the corners of the notches. Afterwards, we relax their translation constraints in the direction of their notches. Finally, we create the handle bar by instantiating another cuboid, scale it to the appropriate dimensions, and place a cylinder on one end. Then we install reference points on the board, the sliders, and the handle bar (indicated by circles in fig. 11).

To position the parts of the treadmill correctly we have to add explicit constraints to the ones we already specified implicitly; we constrain the vectors between the reference points on the handle and those on the sliders to be of length zero to make the points coincide (fig. 11).

Some explicit constraints are derived from the implicit constraints automatically: Either slider can only move in one direction relative to the board, so a slope constraint is imposed on the lines between their reference points and a point on the board (which happens to be the reference point of the board in this case). The distance between the reference points on the sliders is constrained, because they are matched with the reference points of the handle bar, which is treated as being rigid.

The constraints need to be defined in the plane that is defined by the two directions in which the sliders can move. This means that all constraints are specified and solved between projections of the constrained points onto that plane.

There is only one more constraint left that has to be specified to constrain the assembly completely: the slope of the handle. We constrain it by giving it a symbolic parameter name "alpha". For the numerical evaluation we need to specify a numeric value for "alpha". The result of the constraint solving is shown in fig. 12. (For the demonstration we let alpha run from 0 to 360 degrees in 10-degree-increments. This will give an animated display of the assembly (see fig. 13).

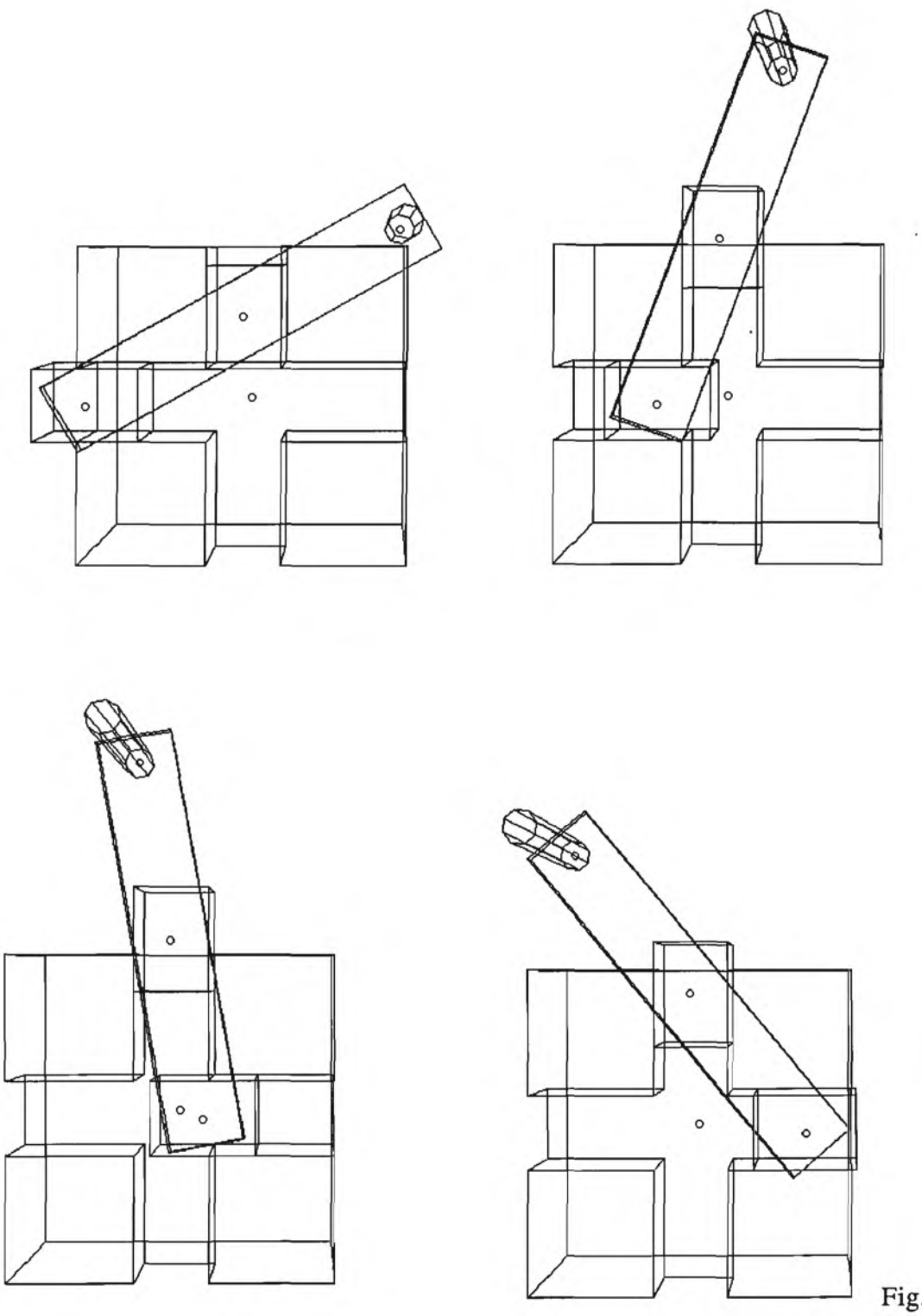

Fig. 13

Once the symbolic constraints have been solved, the results will be turned into degrees of freedom. For the example above, the positions and orientations of the two sliders and the handles are completely constrained (they are functions of the parameter "alpha"). This means we can no longer freely manipulate these objects.

8 Group Hierarchies and Explicitly Defined Constraints

Whereas implicitly defined constraints directly express dependencies that are naturally represented by a group hierarchy (see section 4), explicitly defined constraints can be utilized to express relations between objects within the same group.

The constraint solver solves systems of constraints bottom up. At each level in the hierarchy, objects in a group (and all assemblies) are treated as rigid objects. 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 in the average, these local networks will be fairly small. Since the time needed for solving a constraint network can grow substantially with the number of constraints specified, localizing the the constraint solving improves the speed enormously.

Whenever the shape of a group is changed, all of their super groups need to be reevaluated. The symbolic constraints, however, do not need to be recomputed. For example, if the positions of the axles in a gearbox are constrained such that the cogwheels touch appropriately, and the diameter of one of the cogwheels is modified, the position of the axles have to be re-adjusted by substituing the new values into the old symbolic solutions.

9 Implementation

The system described in this paper comprises an interactive geometric modeler for polyhedral objects, providing among other things, regularized set operations. The symbolic constraint solver is implemented in Prolog, connected to the modeling system through a language interface. The system is implemented on Macintoshes, Sun workstations under the "Suntools" windows systems and for X_windows. We have also implemented an interface to the sculptured surface modeler Alpha_1, using the Alpha_1 network model server. This enables direct manipulations and the use of constraints for Alpha_1, currently on the level of CSG primitives and 3D transfor-

mations. This will also allow us to experiment with more complex models; we are no longer restricted to simple polyhedra.

10 Conclusion

By utilizing constraints, we have done a step towards building an easy-to-use interactive geometric modeling system. Objects can be created, dimensioned relative to other objects, translated, rotated and scaled according to exact specifications with a few mouse-clicks and dragging motions. The intermediate results of such operations are fixed by constraints, so that future modifications will not violate previously fulfilled design requirements. Handles provide the user not only with an interactive transformation facility, but also with feedback about the degrees of freedom that are left for an object. If dependencies are not sufficient, constraints can be specified explicitly and solved with a symbolic constraint solver.

It is important to realize that geometric modeling is only one part of the industrial design process. A design system must support the complete production process, consisting of manufacturing, cost, maintenance, documentation, material and parts, standards, functional and physical properties, etc. Therefore, geometric modeling must be integrated into such a larger design systems (including production planning systems, facility management systems, etc.). Most current CAD systems output mechanical parts as fixed geometric shapes; the geometric design part is almost completely separated from other design aspects. It is difficult to add (or change) information under a different perspective. Changing a part may inadvertently violate previous design decisions. Utilizing constraints prevents the user from doing inappropriate modifications; it is also more flexible than parametric design: Constraints can be associated with geometric objects (or they can be removed) at any time, whereas parametric design usually requires that the user specifies the operations and dependencies in a specific order.

Our goal is to develop a flexible framework to communicate ideas between different departments of a design office during the early design process. Constraints can be used to communicate design specifications together with the geometric objects, so that modifications in later design steps don't violate the original requirements. Therefore, constraints should be an integral part of the information stored in a CAD database.

Acknowledgements

We would like to thank Shiaofen Fang, Kwansik Kim and Susan Skowronski for their help with the implementation of the modeler.
This work was supported in part by the National Science Foundation,
grant #DDM 8910229.

References

[BIE 90] Bier, Eric A., Snap dragging in 3 Dimensions, Proceedings of the 1990 Symposium on Interactive 3D Graphics, March 1990, pp. 193-204

[BAR87] Barford, L.A., A Graphical, Language-Based Editor For Generic Solid Models Represented By Constraints, PhD Thesis, Cornell University, May 1987

[BOR81] Borning, A., The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory, ACM Toplas, Vol 3, No 4, October 1981

[BRÜ85] Brüderlin, B.D., Using Prolog for Constructing Geometric Objects Defined by Constraints, Springer LNCS, No 204, pp. 488-459, Proceedings of Eurocal '85, Springer Verlag Berlin, New York 1985

[BRÜ86] Brüderlin, B.D., Constructing Three-Dimensional Geometric Objects Defined By Constraints, 1986 Workshop on Interactive 3D Graphics, Conference Proceedings, Chapel Hill, North Carolina, ACM Siggraph 1986

[BRÜ87] Brüderlin, B.D., Rule-Based Geometric Modeling, PhD Thesis, ETH Zürich, Switzerland, Verlag der Fachvereine, vdf-Verlag, Zürich 1987

[BRÜ88] Brüderlin, B.D., Automatizing geometric proofs and constructions, Computational Geometry and its Applications, H. Noltemeier (Ed), Springer Lecture Notes in Computer Science No 333, pp. 253-373, Springer Verlag 1988

[BRÜ90] Brüderlin, B.D., Symbolic Computer Geometry for Computer Aided Geometric Design, Proceedings of the NSF Design and Manufacturing Systems Conference, January 1990, pp. 177-181

[BRÜ90b] Brüderlin, B.D., An Axiomatic Approach for Solving Geometric Problems Symbolically, Tech Report UUCS 90-023, Computer Science, University of Utah, November 1990.

[EAT89] Eaton, R.L., Explicit Geometric Constraints, MS Thesis, Cornell University 1989

[FUQ87] Fuqua, T.W., Constraint Kernels: Constraints and Dependencies in a Geometric Modeling System, MS Thesis, University of Utah, August 1987

[GOS83] Gosling, J.A., Algebraic Constraints, PhD Thesis, Carnegie-Mellon University, May 1983, published as CMU Computer Science Department tech report CMU-CS-83-132

[HOP84] Hopcroft, J., Joseph, D., Whitesides, S., Movement Problems for 2-Dimensional Linkages, SIAM Journal of Computation, Vol. 13, No. 3, August 1984

[LIG82] Light, R., Gossard, D., Modification of geometric models through variational geometry, Computer Aided Design, Vol. 14 No. 4, July 1982, pp. 209-214

[LIN81] Lin, V.C., Gossard, D.C., Light, R.A., Variational geometry in computer-aided design, Proceedings of SIGGRAPH, 1981

[NEL85] Nelson, G., Juno, a constraint-based graphics system, ACM Siggraph 1985, pp. 235-243

[NIE86] Nielson, G.A., Olsen, D.R.Jr., Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices, Workshop on Interactive 3D Graphics, October 23/24 1986, Dept. of Computer Science, University of North Carolina at Chapel Hill

[ROS86] Rossignac, J.R., Constraints in Constructive Solid Geometry, Interactive 3D Graphics, October 23-24, 1986, Chapel Hill

[STE80] Steele, G.L.Jr., Sussman, G.J., CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions, Artificial Intelligence, 14(1):1-39, January 1980

[SUT63] Sutherland, I., Sketchpad, A Man-Machine Graphical Communication System, PhD Thesis, MIT, January 1963

[WIT87] Witkin, A., Fleischer, K., Barr, A., Energy Constraints on Parameterized Models, Siggraph 87 Conference Proceedings, ACM Siggraph, August 1987, pp. 225-232