

Inferring 3D models from freehand sketches and constraints

Lynn Egli, Ching-yao Hsu, Beat D Brüderlin* and Gershon Elbert

This paper describes 'Quick-sketch', a 2D and 3D modelling tool for pen-based computers. Users of this system define a model by simple pen strokes, drawn directly on the screen of a pen-based PC. Exact shapes and geometric relationships are interpreted from the sketch. The system can also be used to sketch 3D solid objects and B-spline surfaces. These objects may be refined by defining 2D and 3D geometric constraints. A novel graph-based constraint solver is used to establish the geometric relationships, or to maintain them when manipulating the objects interactively. The approach presented here is a first step towards a conceptual design system. Quick-sketch can be used as a hand sketching front-end to more sophisticated modelling, rendering or animation systems. Copyright © 1996 Elsevier Science Ltd

Keywords: geometric constraints, conceptual design, free-hand sketch interpretation

INTRODUCTION

Much effort has gone into making graphical design tools more efficient, and easier to use, mainly by providing mouse controlled object-oriented graphical user interfaces. In recent years, this effort succeeded in making computer-aided design available to drafts people who are not computer specialists, and, at the same time, improved their productivity. Nonetheless, these CAD systems are still not very useful in the initial design phase. It takes many hours or even days of concentrated effort to create the 3D models for new designs, illustration purposes, or for animation. This is in contrast to the seconds or minutes it takes to quickly sketch an idea on paper which is often good enough to convey the essentials of a new concept.

With the work presented in this paper, we extend some of our earlier work^{5,6}, by adding a 3D constraint solver. We combine some of the sketching techniques engineers are already familiar with, with the power of solid

modelling, to build a design tool for 2D and 3D objects. After a brief survey of related work, we describe the basic functions of interpreting pen strokes as 2D shapes, such as lines, circles, arcs or B-spline curves, and how geometric relationships are recognized from the sketch. This information is used to clean up the drawing, and to establish constraints. We describe the criteria applied in interpreting the information, and also the type of feedback the system provides to the user, when editing the input interactively. Then we describe a number of sketch-based 3D techniques developed for Quick-sketch. The graph-based approach to constraint solving developed for this system is used to solve 2D and 3D geometric constraints within the same framework. We give a brief description of the software components employed in the prototype implementation and draw conclusions and develop the plan for future research.

RELATED WORK

This section describes related research in sketch interpretation and the use of constraints in geometric design.

Sketch interpretation

The paper 'Design capture system: capturing back-of-the-envelope sketches' by Hwang and Ullman¹³ is interesting both for the system implemented and for the background research behind that system. The authors performed an extensive study of mechanical engineers in action. They videotaped mechanical engineers solving ill-defined problems for over 46 hours. One of their chief observations was the central role that sketching plays in the design process. It was hypothesized that sketching is an extension of visual memory. It was also noted that these professionals possess considerable expertise in sketching. This talent is generally unexploited in contemporary CAD systems. The study concluded that, in order to be effective, a CAD system must allow sketched input, have a variety of interfaces, recognize features, and manage constraints. Exploiting these criteria, they built a system they felt would be useful to engineers. Their system has two phases. The first is a 2D stroke recognition system. Sketched strokes are interpreted as lines, arcs, circles, ellipses, etc. These

Computer Science Department, University of Utah, Salt Lake City, UT 84112, USA

* To whom correspondence should be addressed: Technical University of Ilmenau, D-98693 Ilmenau, Germany

† Computer Science Department, Technion IIT, Haifa 32000, Israel

Paper accepted: 23 April 1996



Figure 1 Example of sketching primitive shapes with a pen

primitives are accumulated until they can be recognized as a 3D feature. The features are then placed in a coherent 3D topology. New features can be built upon previous ones¹⁴. The limitation with this system is, that it will only recognize a limited set of features (blocks, cylinders and spheres) sketched from a fixed perspective. Also, no constraint system was implemented in the described system.

The system described in ‘Designing solid objects using interactive sketch interpretation’¹⁸ lets the user sketch a whole line drawing of an object in 2D, and then attempts to interpret it into 3D. The system uses hints, such as shading of non-visible lines, and previous interpretations of the object, to guide the interpretation. Additionally, the user can specify constraints on the objects. Sketched segments are automatically aligned, where appropriate, making input easier. The object can be viewed from any view point, and the user can make modifications to any side of the object, later. There are possibly a few limitations with this system: forcing the user to shade non-visible line segments seems tedious and error-prone. Furthermore, it seems questionable that the system could successfully interpret a large complicated drawing all at once.

Constraint solving methods

Probably the first constraint-based graphics system was ‘Sketchpad’²³, by Ivan Sutherland. This system uses constraint propagation as one of its basic mechanisms. The solving of constraints is done by finding an order of evaluation to satisfy all the constraints from some initial values, progressively. However, cyclical dependencies cannot be resolved with the propagation method, and the system resorts to numerical methods for most interesting cases. Despite its promise and the many research efforts made in the past 30 years, since the inception of Sketchpad, constraint solving has remained a very difficult problem. There have been several different approaches developed, but there are still none that are general, efficient, and robust enough for commercial use.

Numerical techniques such as relation and Newton–Raphson iteration can theoretically solve problems even if they do not have a closed form algebraic or geometric solution. In this approach, constraints are translated into a system of algebraic equations and then solved using



Figure 2 The program interprets the strokes as circle, cubic Bézier, line, horizontal and vertical lines



Figure 3 Sketching a sequence of strokes

iterative methods. Numerical approaches are described in References 10 and 15. While they are quite powerful and general, numerical techniques have convergence problems that make them very unpredictable—especially in 3D.

Constructive geometric constraint solvers are the latest trend in CAD. They are especially successful in 2D. These solvers satisfy the constraints by finding a sequence of geometric constructions (mostly ruler and compass constructions) which are capable of resolving some of the cyclical dependencies (although not in all cases). There are roughly two subcategories of constructive solvers, (a) rule-based approaches^{1,4,24}, and (b) graph-based approaches^{2,9,17,20,22}. In this paper we generalize our graph-based constraint solving method to the third dimension, and integrate it with the sketching system.

INTERPRETING PEN STROKES AS GEOMETRIC SHAPES

A user of the Quick-sketch system draws directly on the pressure sensitive LCD screen of a laptop PC, with a pen (Figure 1).

Two-dimensional lines, circular arcs, full circles, and B-spline curves can be sketched. The stroke is sampled as a sequence of points from which the program interprets the type of shape, using some mode dependent preference function (see Figure 2). Once the type is decided, the closest fit to the stroke is determined, using different numerical techniques (a least-square fit approach is taken for lines and B-spline curves; circles and circular arcs are determined using a gridding technique).

Once the closest fitting primitives have been found, the system tries to interpret certain relationships between them (e.g. whether two curves are adjacent and whether two adjacent lines are at a right angle). If such a relationship is found within tolerance, the parameters of the primitives are altered to establish an exact relationship. Figures 3 and 4 show the effect of the interpretation. The program automatically recognizes adjacencies and the right angles, and cleans up the drawing. Two exact rectangles are created (the second one is axis-aligned). Angle constraints (marked with an A) and slope constraints (marked as S) are associated with the points and lines, keeping these relations intact when the geometry is changed by dragging. More examples are shown in Figures 9 and 10.

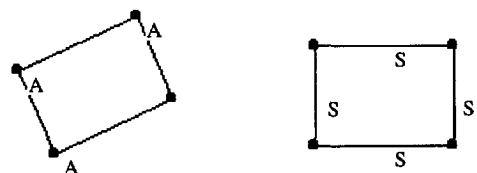


Figure 4 The program automatically recognizes relationships

SYSTEM BEHAVIOR AND FEEDBACK

Quick-sketch applies a number of criteria to interpret the type of shape and the relationships between objects. Since the sketched shapes are not usually exact, the system has to apply tolerance in interpretation. If the interpretation does not correspond to the user's intention, there needs to be an easy way of recognizing and changing it. This section describes the concepts in some detail.

Tolerance-based interpretation

The interpretation of pen strokes takes into account the closeness ϵ to the exact shape, the speed of the input ν , a user settable tolerance τ which is associated with the user's skill, the length of the stroke (significance σ), and the user preference mode μ , which represents the type of the task.

Modes

Examples for user preference modes are, for instance:

- *Technical drawing mode*—preferring lines and circular arcs, right angles and tangencies, parallel lines, and concentric circles.
- *Symbol mode*—preferring horizontal and vertical lines, symmetric shapes, right angles, parallelism, semi- and quarter-circles.
- *Free form mode*—preferring B-spline curves and tangencies.

In each of the modes, a higher tolerance is applied to the preferred attributes, making them a more likely choice.

Significance

The longer a curve is, the more accurate its relative global features will be drawn. This is based on the assumption that the user will have a chance to do some midcourse correction. For instance, a short horizontal line, in the average, will deviate by a larger angle from an exact horizontal line than a long one. The system divides the tolerance applied to angles by $\sqrt{\text{length-of-line}}$, within certain limits. Similarly, other properties are derived using a length dependent tolerance function.

Speed

Sketching an object very quickly will be less accurate, in general, than drawing it more carefully and slowly. The speed of a stroke is taken into consideration by multiplying all the tolerances with the speed. Therefore, the faster an object is drawn, the larger is the tolerance used in the interpretation, and thus it is more likely that a correction is applied. This approach is assuming that the faster strokes are more sloppy, and in reality should represent some more accurate shape (for instance a rectangle). This behaviour can also be exploited by the user. For instance, to draw a line at a small angle (which should not be interpreted as horizontal) one would draw such a line more carefully and deliberately.

Skill

The speed factor is compensated somewhat by a user settable skill factor. The more skilled a designer is, the less his or her drawing should be revised, even if drawn

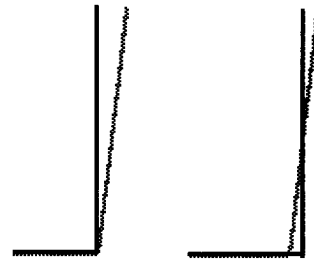


Figure 5 Snapping to a right angle constraint (2 possibilities are shown)

fast. This is achieved by setting the tolerance τ to be inversely proportional to the skill.

Cleaning up the drawing and establishing constraints

The system tries to satisfy the relationships interpreted into the sketch, with the least amount of local change. This paradigm is applied on a case by case basis, using heuristic rules.

For example, *Figure 5* shows two ways of snapping two lines in a right angle. The choice on the left side results in a larger change on one end. The system tries to distribute the change over both ends. This, however, requires a change in the adjacent line, at the bottom, but it is less surprising to the user than the first choice. More examples for this approach can be found in Reference 5.

Correction of automatically made interpretations

The criteria applied in making automatic interpretation work together naturally, and make the behaviour of the system quite predictable. Nevertheless, there will always be cases where the system makes a wrong interpretation of the user's intent. It is therefore essential to provide the designer with clearly understandable feedback about the interpretations made, and to allow for a simple way of correcting them, if they are wrong.

Each interpretation made by the system will result in highlighting of corresponding icons that easily identify the choice (see, for instance, *Figure 6*. The last stroke was interpreted as a line with a right angle constraint). The user can quickly unselect the highlighted constraint on the menu, or select others instead, and the program instantly reinterprets the stroke.

Using a context sensitive menu for corrections after-the-fact is a very efficient way of interaction, especially for pen-based computers. With mouse-based interaction, on the other hand, it is usually necessary first to tell the system each time whether we want to draw a line, B-spline curve, or a circle, and then one has a few limited

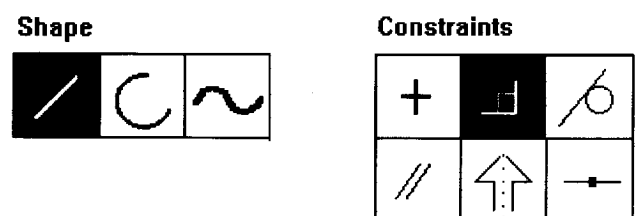


Figure 6 Highlighting the system's choice

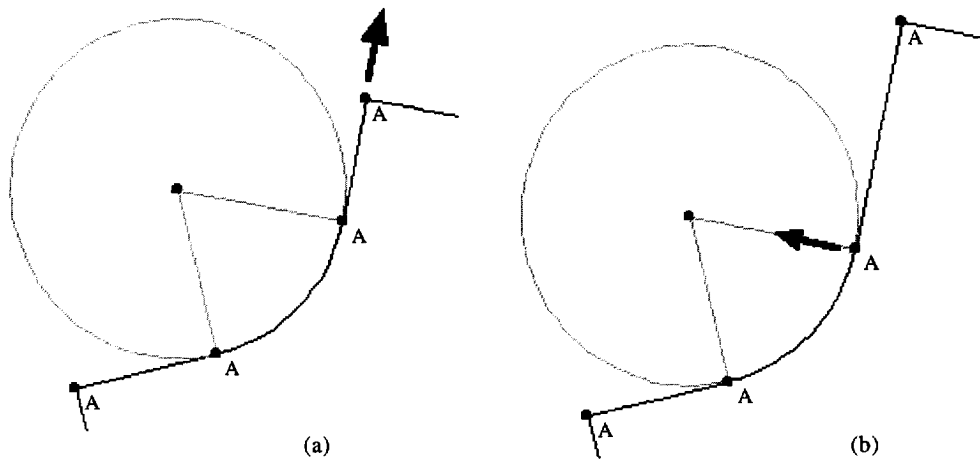


Figure 7 (a) Dragging the point at the top upward (in direction of arrow) will stretch the upper portion of the profile (see *Figure 7b*). Dragging a point on the periphery of the circular arc inward causes the circle to shrink to size (see *Figure 8a*)

options for determining the shape interactively (by dragging the provided handles into place). Since a pen provides a lot more dexterity, we just draw the object and let the system figure out the type of object, the parameters, and the relationships to other objects. In the majority of cases, the automatic interpretations are correct and easy to predict. Only the menus/icons relevant to the objects drawn will then be displayed, so the user can make possible corrections (e.g. to undo snapping to right angles, or to reinterpret a stroke as a low curvature arc, instead of a line, etc.). Together, these approaches speed up the design significantly, as the examples show.

EDITING WITH GESTURES AND SOFT CONSTRAINTS

Quick-sketch allows for interactive manipulation of sketches, by dragging the control points displayed in the drawing. The previously established constraints are maintained during dragging. ‘Gestural manipulation’ is used to disambiguate the interaction for underconstrained drawings, taking the direction of the stroke into account.

Implicit (soft) constraints have also been introduced to

achieve even better predictable behaviour when manipulating underconstrained drawings. The way the soft constraints are treated in the constraint solver is similar to the hierarchical constraint solver described in Reference 8. The solving mechanism used in our approach, however, is quite different. The details of our constraint solver are described in a later section.

The following example shows how the system reacts when dragging points of a profile (with circular arc tangent to two lines, and two right angles at both ends; *Figures 7 and 8*).

The idea behind gestural manipulation is to use the direction of the pen stroke (gesture) in determining which effect a manipulation will have. The direction of the stroke is compared against the direction of each line adjacent to the point picked. In the case of circles or B-spline curves the control polygons are used as a reference (drawn dimly on the screen). If the stroke is (within tolerance) along such an existing line, the constraint solver tries to achieve a 1D degree-of-freedom motion in this direction. Gestural manipulation for 3D interaction was proposed in Reference 16. Due to the increased dexterity of a pen over a mouse, this type of interaction gains new importance.

The possible reaction to manipulations, even with gestures, may still be ambiguous, especially if only a few

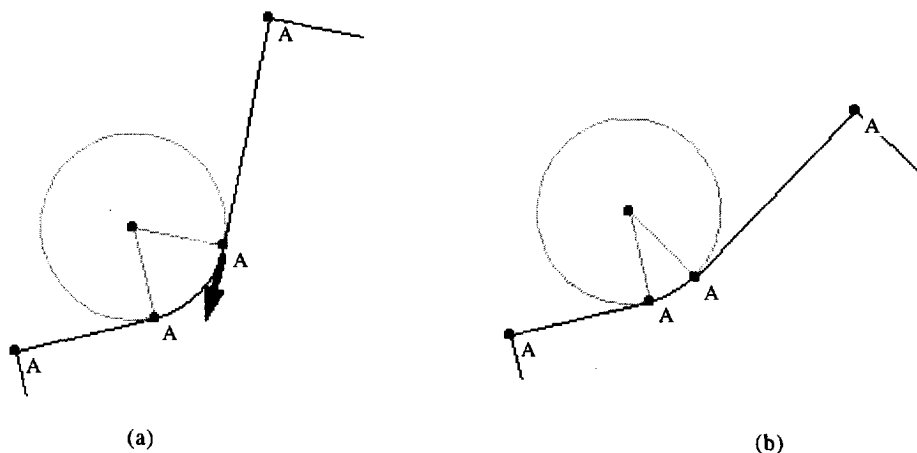


Figure 8 (a) Dragging that same point tangential to the circle causes a change in the opening angle. (b) The change causes the upper half of the profile to be rotated about the circle centre due to the constraints. Notice that all the previously imposed constraints, such as right angles and tangencies, are maintained during all manipulations

Table 1 Mode dependent weights for soft constraints

Mode	Position	Slope	Distance	Angle
Rigid	0	0	10	10
Bend	5	0	10	0
Stretch	5	10	0	10
Free	5	0	0	0

constraints are defined. To disambiguate the system's behaviour further, we introduced so-called soft constraints. Soft constraints are only observed if they do not contradict any explicit (hard) constraints. Each type of soft constraint has a mode dependent weight associated with it. The weight is interpreted as a 'penalty' for violating that constraint. For hard constraints the penalty is infinity. When manipulating an object, the constraint solver, in its planning phase, tries different ways of transforming the objects while dragging. For any given plan, some of the soft constraints will have to be violated. The weight of each violated constraint is summed up. Several plans are analysed, and in the end the one with the least penalty is used. Giving different weights to different types of constraints will effect different behaviours in interactive situations. In our system, we do not put the burden to assign weights to the individual constraints on the user, but rather provide predefined sets that can be associated intuitively with some geometric behaviour. The weights are summarized in *Table 1*.

In *rigid* mode, a high penalty is associated with violation of distance and slope constraints but no penalty is associated with position and slope constraints, causing concatenated primitives (e.g. in a profile) to be translated or rotated as a rigid object.

In *stretch* mode, no penalty is associated with distance violations but slopes and angles carry a high penalty. The penalty on position constraints keeps the transformation more local which causes the objects to be stretched locally.

In *bend* mode, angle constraints have no penalty, but the system tries to maintain distances causing a kind of a bending or shearing transformation.

In *free* mode, only position constraints carry a penalty,

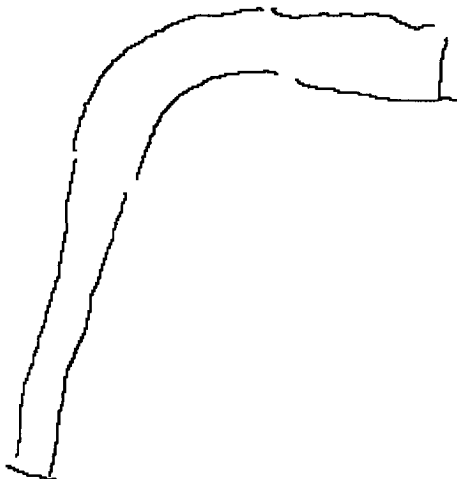


Figure 9 A 2D sketch of a profile of a mechanical part

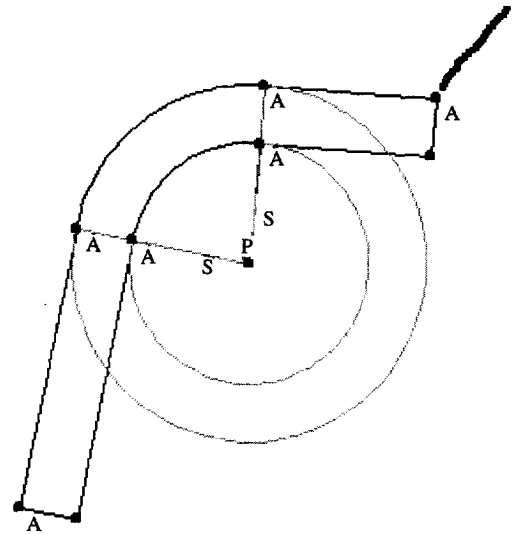


Figure 10 Pen strokes of *Figure 9* recognized as circular arcs and lines

causing free local deformation, maintaining neither angles nor distances.

More details on this subject can also be found in Reference 5.

SKETCHING IN 3D

To model 3D objects by sketching, the system currently allows the following techniques: extrusion surfaces can be generated by sweeping a 2D profile along a straight line. Ruled surfaces can be defined between two sketched curves. A sketched cross-section can be swept along a sketched curve (*Figure 17*), creating a sweep surface. A surface of revolution can be created by simply sketching two approximately symmetric silhouette lines.

Also, lines and curves can be sketched on planar faces of existing objects. This way, features can be quickly added to objects.

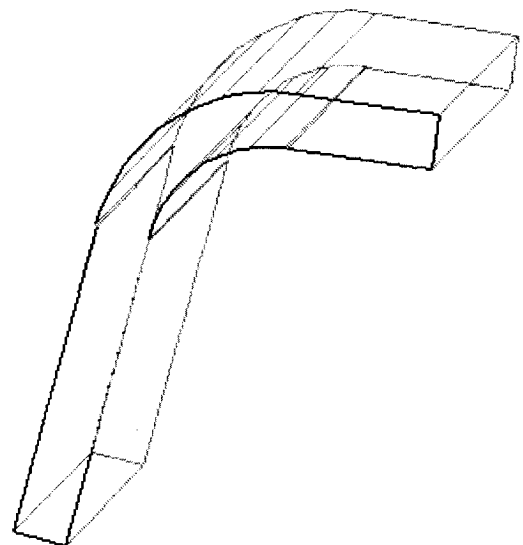


Figure 11 The resulting 3D part

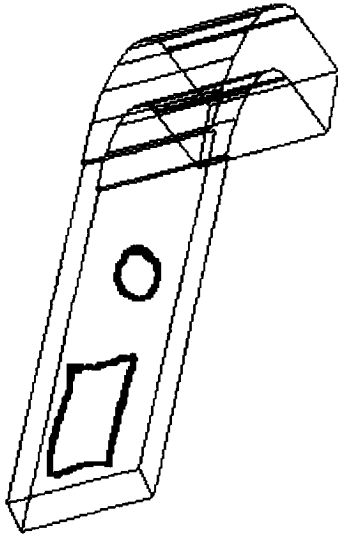


Figure 12 Sketching features on the solid

Sketching of a solid model

An example sequence for sketching a 3D object from scratch is shown in *Figures 9–13*.

We start by sketching a 2D cross-section (profile) of a mechanical part (*Figure 9*). The pen strokes of *Figure 9* are recognized as circular arcs and lines. Also, the lines are recognized to be tangent to the circles. Parallel lines, concentric circles, and right angles are interpreted. The drawing is cleaned up instantly, and constraints are established (*Figure 10*). The 2D profile can then be extruded

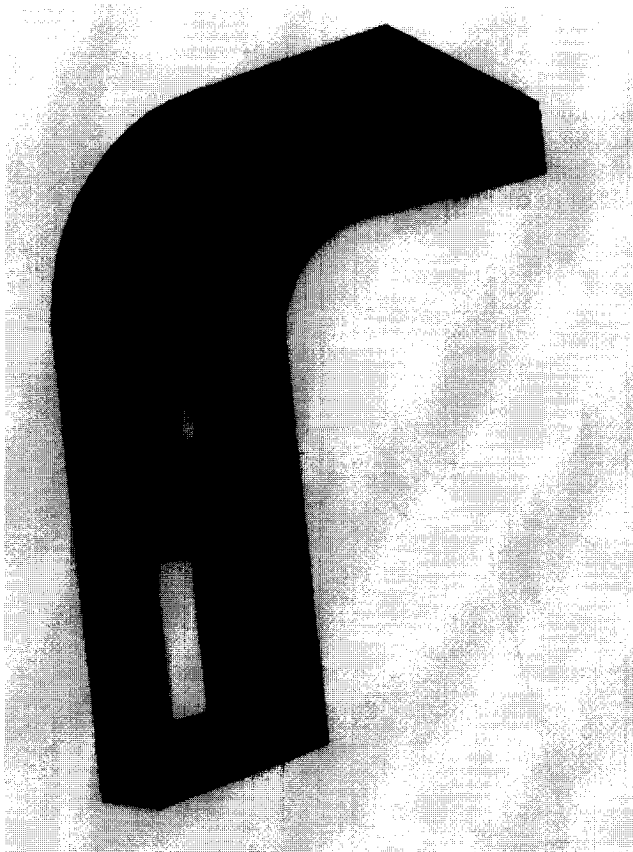


Figure 13 A shaded rendering of the part in *Figure 12*

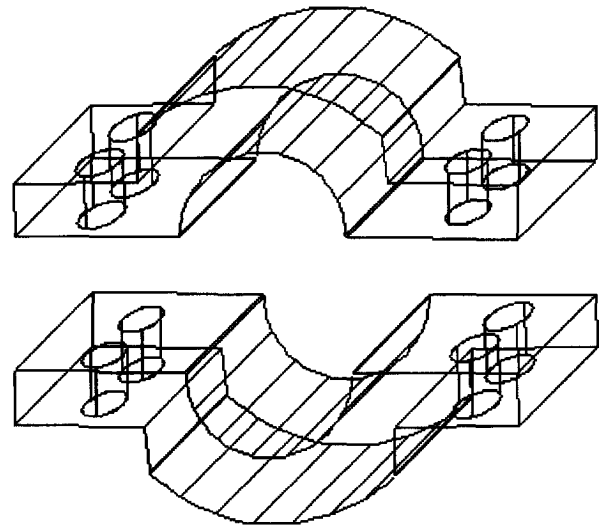


Figure 14 Sketching parts for assembly

into the third dimension (indicated by a pen stroke near the top right), which results in a solid object (see *Figure 11*).

Features can be sketched onto any flat surface facing the viewer in any perspective (*Figure 12*). These features are also cleaned up (by establishing right angles, and aligning them with the boundary lines), and then can be extruded with a pen stroke. All the previously introduced 2D techniques also work for profiles sketched onto a face (e.g. detection of parallelism, snapping to right angles, tangent circles, constrained manipulation, etc.). However, within the local coordinate system of the face they are drawn on. The resulting solid object was rendered with the IRIT library (*Figure 13*). The data can be exported to other solid modelling systems, for postprocessing.

The interactive modelling of this solid from scratch took about 35 s. This is even slightly faster than trying to sketch the same object with pencil on paper, and it has the additional advantage of generating a full 3D CAD model that can be edited, dimensioned, and rendered, etc. Another example for parts sketched with the system is shown in *Figure 14*.

Sketching B-spline surfaces

The following examples show a few sketching techniques for the creation of B-spline surfaces, in Quick-sketch. Ruled surfaces can be created by interpolating between two curves (*Figures 15 and 16*). The two curves are first interpreted in the x/y plane. The stroke between the two curves is then used to determine the depth. It is first projected onto the x/z plane, in the view direction; and

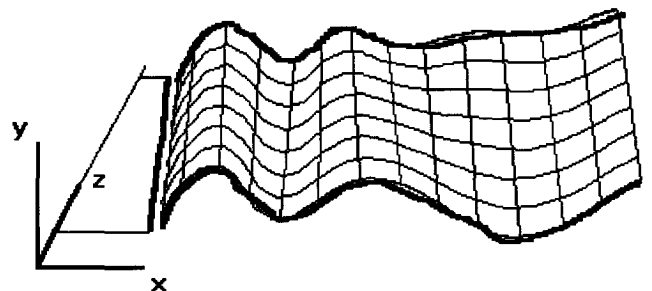


Figure 15 Ruled surface construction

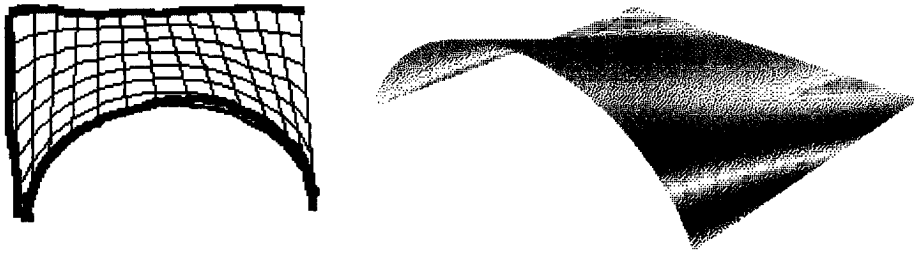


Figure 16 Ruled surface between a line and a semi-circle

then an orthogonal projection within the x/z plane onto the z -axis determines the offset between the two interpolated curves in the z -direction, assuming that the stroke is parallel to the x/z plane.

Sweeping a circular arc along a B-spline curve creates a sweep surface, as shown in Figure 17a. The stroke for the cross-section curve is interpreted as a projection onto the x/y plane, and the sweep curve is projected onto a plane parallel to the x/z plane. In Figure 17b a circular arc is swept along another circular arc, creating a portion of a torus.

A surface of revolution can be created by sketching two approximately symmetric silhouette lines in the x/y plane. The system determines the symmetry axis to be parallel to either the x -axis or the y -axis (Figure 18).

GRAPH-BASED GEOMETRIC CONSTRAINT SOLVING

This section introduces a graph-based approach to geometric constraint solving. A *geometric model* in our approach is defined by a set of geometric objects $O = \{o_1, o_2, \dots, o_n\}$ and the set of geometric relations (constraints) $C = c_1, c_2, \dots, c_m$ between the elements of O .

Geometric objects such as points, lines, and circles listed in Table 2 own degrees of freedom, which allow them to vary in shape, position, size and orientation. The set of degrees of freedom $DOF = \{dof_1, dof_2, \dots, dof_1\}$ owned by the objects in O represent different types of domains. Each domain represents a set of allowable

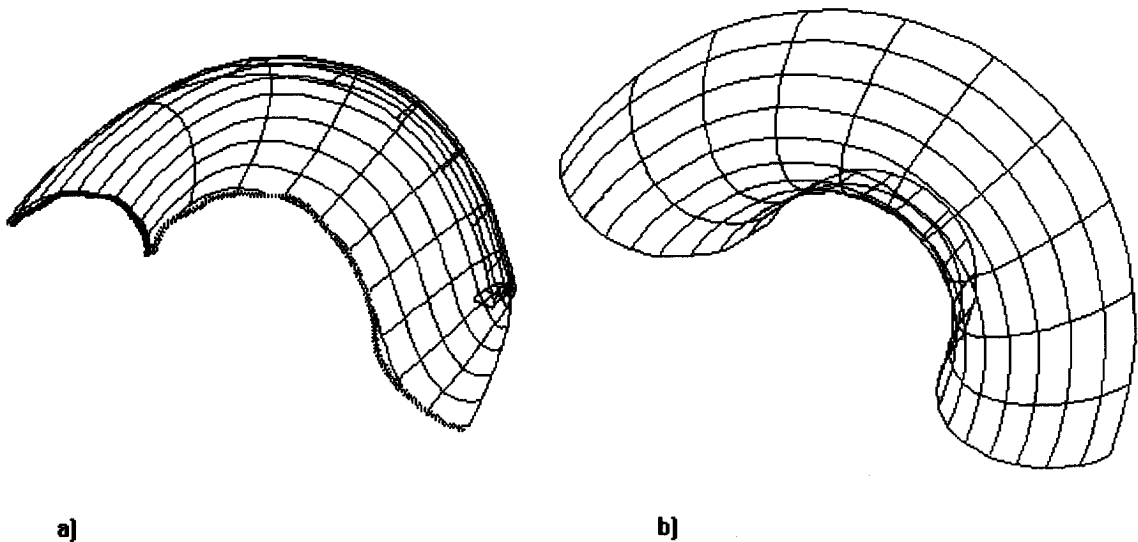


Figure 17 Sweep surfaces

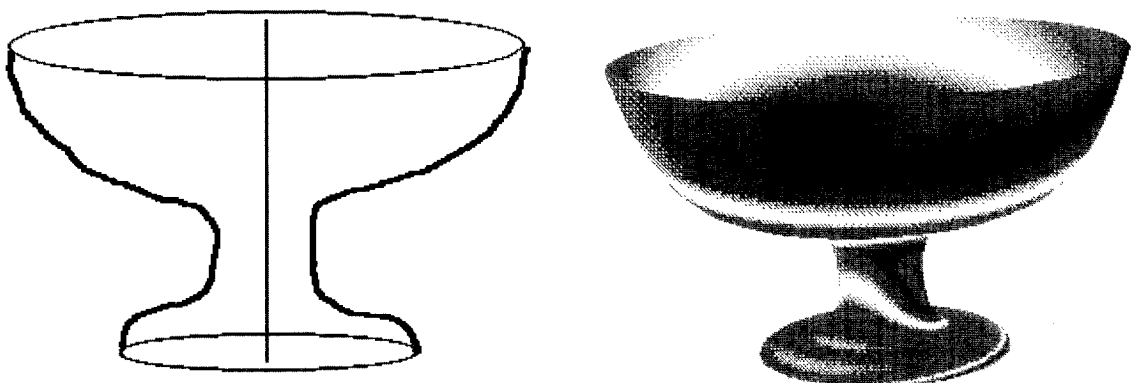


Figure 18 Sketching surface of revolution and its rendering with IRIT

Table 2 2D object types and their degrees of freedom

Object type	Degree of freedom
Point	2
Line	2
Circle with fixed radius	2
Circle with variable radius	3

Table 3 Constraint types and their attributes

Constraint type	Arguments	Valency
Distance	pt,pt	1
Incidence	pt,ln	1
Incidence	pt,cl	1
Angle	ln,ln	1
Tangency	ln,cl	1
Tangency	cl,cl	1
Mid-point	pt,pt,pt	2

values. The cross product of these domains contains the possible states of the geometric model. The number of degrees of freedoms l is the dimension of the model (also called its total degree of freedom).

Geometric constraints such as those in Table 3 define an n -ary geometric relation among a set of n objects, $c_i = c_i(o_{i_1}, o_{i_2}, \dots, o_{i_n}, \lambda)$, where λ is the parameter of the constraint. Depending on the constraint type, the parameter may be a scalar, a vector, or empty. A constraint reduces the degree of freedom of the model by a certain number (called the *valency* of the constraint, as defined in Reference 1).

We also introduce a special class of constraints, so-called *local (unary) constraints*, which may consume all or part of the degrees of freedom owned by an object. For example, a constant x -coordinate constraint on a 2D point fixes its x -coordinate in space (we say one degree of freedom owned by the point is consumed by that constraint).

In the following, we will introduce constraint networks as the graph representation of geometric models. A *constraint network* is an undirected graph which consists of a finite set of nodes and arcs. A *node* in the network represents a geometric object and is depicted as a rectangular box annotated with $DOF_{owned}/DOF_{consumed}$. An *arc* represents an n -ary relation and is therefore a general undirected arc with fan-out equal to n . We label

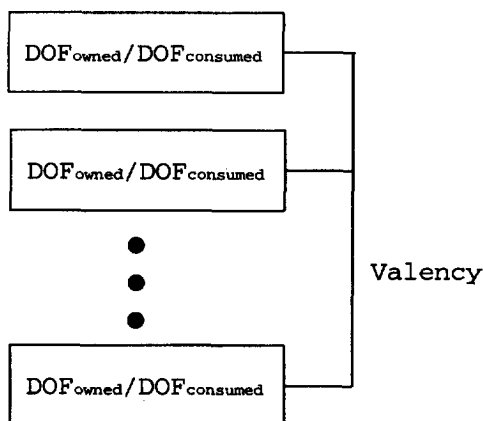
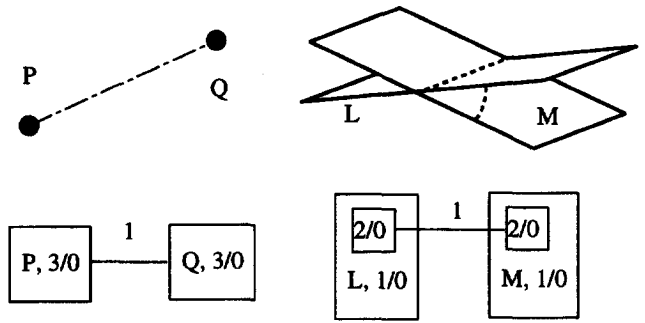


Figure 19 The nodes and arcs of a constraint network



(a) A distance constraint. (b) An angle constraint.

Figure 20 Two examples of the constraint representation

the arc with the valency of the constraint, as shown in Figure 19. In the following, we will use the terms ‘object’ and ‘node’, as well as ‘constraint’ and ‘arc’ interchangeably.

In 3D we typically have objects, such as planes, lines, vertices, spheres, etc. A plane has two rotational and one translational degrees of freedom. Note that (depending on the representation) there are cases of constraints that only affect specific degrees of freedom of an object. For example, if a plane is represented by one translational degree of freedom and two rotational degrees of freedom, an angle constraint between two planes would only affect the rotational degrees of freedom. We will therefore sometimes represent the different types of degrees of freedom in an object node by *sub-nodes* (depicted by small boxes inside the *master node*). The arcs representing the constraints may connect these small boxes directly as shown in Figure 20b. This type of classification of degrees of freedom often helps in detecting degenerate cases of constraints (examples are described in Reference 11). In other cases, the degrees of freedom are affected indistinguishably (e.g. the x -/ y -/ z -coordinates of a point are equally affected by a distance constraint in Figure 20a).

Solid objects, in boundary representation (B-rep), can be translated into a constraint representation as follows. An edge in a B-rep may be derived by intersecting two planes. We can also represent the incidence relation between edges and planes by constraints, as shown in Figure 21a. A vertex can be defined as incident on three intersecting planes, or as incident on one edge and on one plane, as shown in Figure 21b and c.

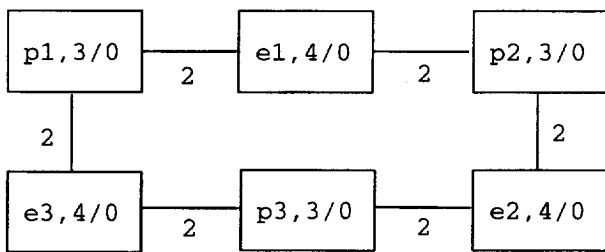
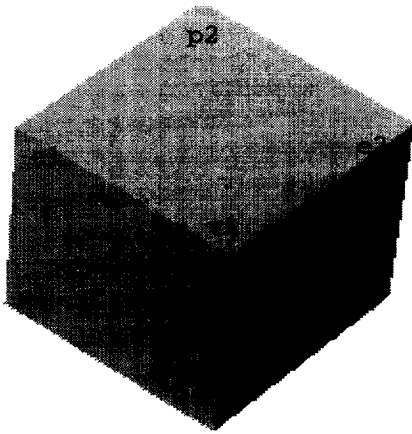
Representing B-reps as constraint networks allows us to handle constrained 3D B-reps consistently within the same framework as for 2D constraints. The advantage of this approach is shown in subsequent examples.

Solving constraints by geometric constructions

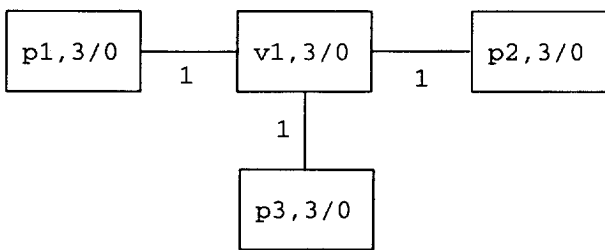
A *geometric construction* may be applied when there is a definitive way to construct a geometric object from other, already constructed objects. In other words, the geometric object in question becomes fully constrained with respect to its related objects. We will explain the concept in terms of the nodes and arcs in the constraint graph.

When all but one node connected to an arc become fully constrained, we say that the arc is ready to *fire*

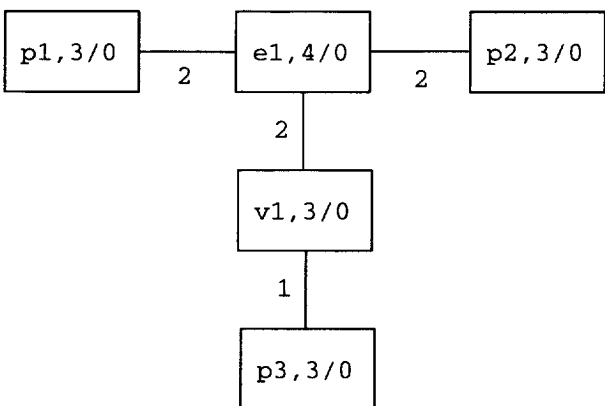
(firing arc). It is depicted as a bold arc with an arrow pointing to the not yet fully constrained node, as shown in Figure 22a. Furthermore, a node becomes ready to fire when the sum of the valencies of the connected arcs that are ready to fire is equal to the available degrees of freedom of the node ($DOF_{\text{owned}} - DOF_{\text{consumed}}$). We call this node a *firing node* (depicted as a bold-line box as shown in Figure 22b).



(a)



(b)



(c)

Figure 21 Incidence relation between planes, edges, and vertices

Depending on the types of the firing arcs, different object constructors may be used to construct the firing node. Here, we use geometric constructions, such as intersecting planes, spheres, lines, or constructing planes from three points, etc. The definition of a geometric constructor implies the construction for general configurations, as well as constructions for degenerate configurations. Also, for each construction, conditions indicating whether the construction is defined for given parameters, and if so, the conditions specifying which of several possible solutions, are chosen.

Constructions represented by firing nodes and arcs introduce a direction into the previously undirected constraint graph, by eliminating cyclical dependencies. In some way, this approach can be seen as a graph representation of the geometric rewrite rule system introduced in References 3 and 4, but it has the additional advantage that the relations are explicitly represented by arcs in the graph (which eliminates unnecessary search for an applicable rewrite rule). Also, the notion of degrees of freedom and valencies is applied here. Together, those notions make it possible to apply a number of efficient graph algorithms to analyse the geometric problem at a very high level. One such approach is clustering²; another graph algorithm for degree of freedom analysis was introduced in Reference 11. The goal of the degree of freedom analysis is to extract from a constraint network a connected portion which possesses a specified degree of freedom. We represent this subgraph as a so-called dependency graph. After this analysis, we can determine, for instance, whether we can manipulate the portion of the constraint network within the degree of freedom, and what the dependencies are between objects.

The example in Figure 23 shows an object defined by 3D constraints. The object on the left, a block with a slot feature, is an object which can easily be generated by the sketch system. By defining additional constraints the object's shape can be modified in various ways. For instance, we want to enforce that the slot feature maintains the right angles between its planes, and it has a given width, while we change the orientation of the feature (model on the right side). The incidence relations between planes and vertices (which are automatically derived from the boundary representation) ensure that the previously defined planar sides remain consistently planar with these modifications. It is therefore not possible to drag a vertex out of any of the planes defined by the other vertices in adjacent faces. The example on the right side shows the creation of a solid object from sketching and 3D constraints, in a very simple way. Note that the object could not have been easily defined from a 2D profile with a sweep alone (the edges in the profile do not form right angles, and their length is not equal to the desired dimensions of the slot), but actually it required the use of the 3D constraint solver.

The next example (Figure 24) is a block with two intersecting slot features. The user defines additional constraints in the form of angle congruence between the walls and the bottom of the feature. Also, the bottom plane of the feature is constrained to be parallel (at a fixed distance) to the bottom plane of the object. In addition to the 9 user-defined constraints, the system automatically derives 96 incidence constraints (between each vertex and the intersecting planes). The fact that the

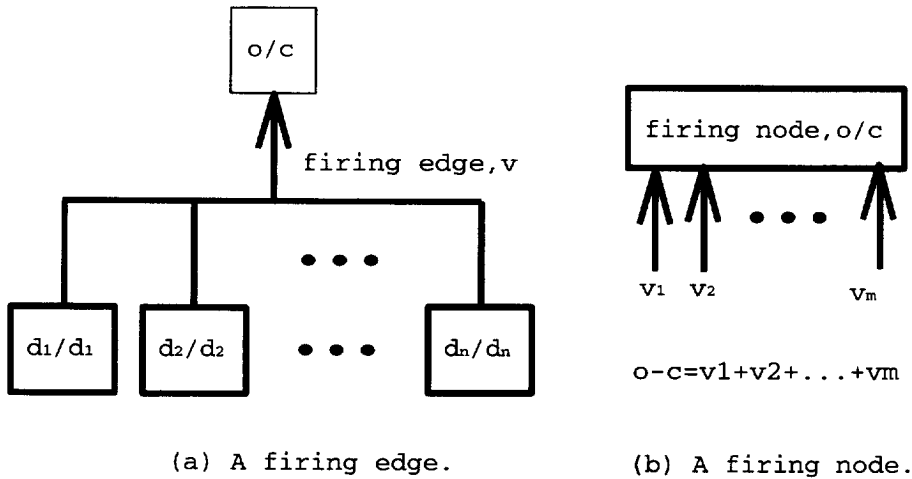


Figure 22 Firing conditions for arcs and nodes

four top faces are coplanar and the feature planes are pair-wise coplanar is implicit in the representation, and does not need to be explicitly defined as a constraint. Nevertheless, a total of 105 constraints is defined in this relatively simple object. With this system we can interactively explore alternatives of the model within its degrees of freedom. The degree of freedom analysis and the recalculation of the geometry can be done in real time with this system, by manipulating the planes and vertices with the mouse. Some snapshots are shown in *Figures 24 and 25*.

Due to the geometric nature of the constraint solver, it can be integrated well with other geometric operations. All geometric constructions including boundary representations derived from sweeps or from Boolean set operations are represented as bi-directional relationships, instead of functional dependencies. For instance,

constructing a vertex by intersecting three planes will result in an incidence relationship of this vertex with three planes. Later, if additional constraints are defined between that same vertex and other objects, the system may decide to reverse the order of construction (expressed by a differently oriented graph), and, for instance, construct one of the three planes through a given vertex, parallel to another plane. In an earlier publication²¹, we showed how snap-dragging can be integrated with constraint solving, and we are currently working on implementing this idea with the new 3D solver.

In general, not all cyclic dependencies can be resolved by a sequence of geometric construction operations. To some extent, this is due to the theoretic impossibility of solving any geometric problem (which is expressible by a constraint graph, and for which a solution exists in principle) with a finite set of construction operations. This general impossibility is formulated and proven by the Galois theory. In some cases a solution in the desired geometric form might still exist, but it may only be derived via a complicated problem transformation which cannot be achieved by an efficient algorithm, at this time. As a practical alternative we therefore introduced so-called iterative geometric constructions¹². This is a geometric variation of the multivariate secant method, and it can be used in combination with the direct object constructions described above. This way, a powerful hybrid approach is created, which combines the robustness and efficiency of a symbolic, geometric solution with the generality of iterative constraint solvers.

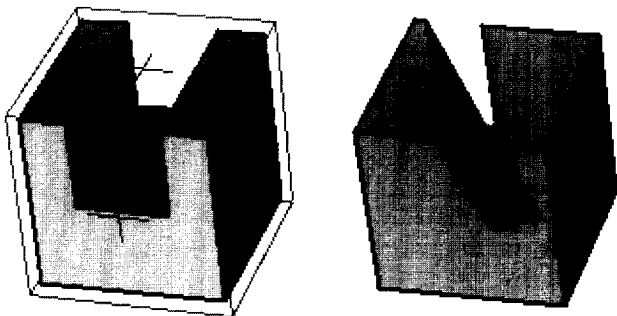


Figure 23 Slot feature with variation

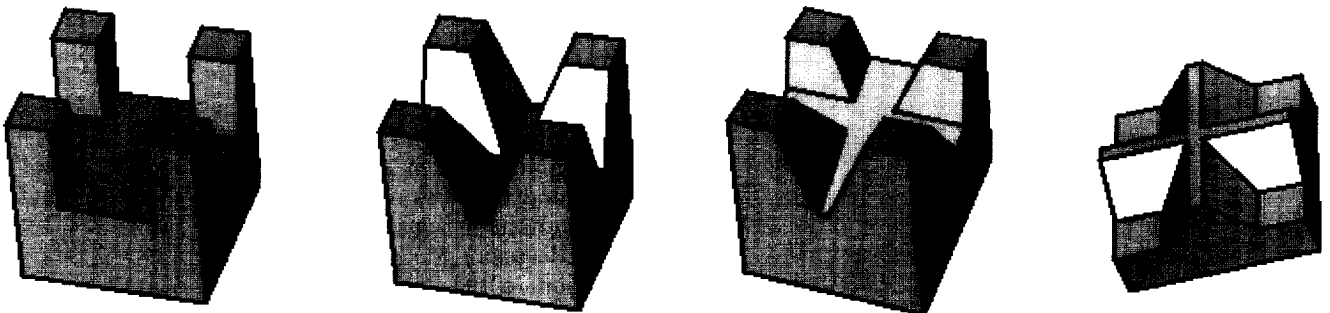


Figure 24 Example for 3D constraints

Figure 25 Variations of example in Figure 24

IMPLEMENTATION

The prototype implementation of Quick-sketch was realized with the Dialog Interface (DI) software, a portable graphical user interface toolkit, and the 'IRIT' computer-aided geometric design package. The program runs, among others, on a 'Compaq Concerto' which is a pen-based laptop PC, running with an Intel 486 Processor, at 33 MHz. The response times for all the examples shown in this paper are unnoticeably short. The only operation that is not done in real time on this machine is the redrawing of the sculpted surfaces.

The 'IRIT' libraries⁷ provide the necessary mathematical tools, such as least-square fit, calculating B-spline curves of arbitrary orders and degrees of freedom through the sampled points of a pen stroke. Functions for creating surfaces of revolution from two silhouette curves, for constructing sweep surfaces from cross-section and axis curves, and for Boolean sum surfaces or Coons patches are also available in IRIT.

The DI library¹⁹ provides the graphical user interface to the system. DI consists of 2D gadgets and 3D interactors and display functions have been implemented as C++ classes in a portable way. An interactive geometric modelling system has been implemented, using DI. The portability of the library allows this modeller to run on PCs under Windows 3.1, as well as Unix, with X-windows (with or without Motif) virtually unmodified.

CONCLUSION AND OUTLOOK

The experience drawn from the development of Quick-sketch is that in 2D the automatic interpretation of pen strokes is a powerful technique. 'Reading the user's mind' can be done reliably, since the input device (pen) is also 2D. The extra control and dexterity provided by a pen allows for these new techniques that would have been impractical with a mouse. The automatic interpretation is generally correct, and, in the few cases where it is wrong, it can be corrected with the press of a button. The way of designing with sketches feels very natural and is also very efficient.

In 3D design, the situation is much more difficult. Interpreting an arbitrary 2D input as a 3D object is too ambiguous, in general. We decided against this idea, and instead, we developed specific drawing techniques that have an unambiguous interpretation in 3D. These techniques are partly adaptations of conventional 3D techniques for a sketch-based environment, in combination with the new 2D sketching and manipulation techniques. In the future, we plan to add many more techniques specifically for free-form surface design, for which sketching can be a powerful technique. Each of these techniques will purposely be limited, and therefore simple. However, we feel that a transparent combination of a few such simple techniques will add up to a very powerful tool.

The use of geometric constraints has proven to be a powerful tool to express design intent, especially in the preliminary design phase, where the exact shape is not generally known. The geometric relationships can be derived from the sketch input in many cases. These constraints become part of the designed object. In the

later design phases the shape of the model can be refined and modified, for instance, by adding more features, adding exact dimensional information (in the form of additional constraints) and by editing the parameters.

It is our goal to make sketching with the computer as natural as using paper and pencil, and even more efficient. It should be possible to jot down an idea in a few minutes or even seconds, avoiding the tediousness of current drafting packages. Sketching makes it worthwhile using the computer in the preliminary design phase, where ideas are still developed. At present, in this stage, designers still use pencil and paper. Only after an idea is almost completely thought out is it transferred to the computer, by hand. With the new sketching technology being developed, a 3D model can be in digital format even during the conceptual phase, and it can be used directly for refinement in the later stages of design.

ACKNOWLEDGMENTS

This work was supported in part by grant No 92-00223 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and an NSF/CISE infrastructural grant. However, opinions, conclusions or recommendations arising out of supported research activities are those of the authors or the grantees and should not be presented as implying that they are the views of the sponsoring agencies.

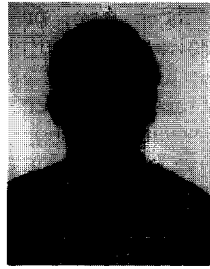
REFERENCES

- Aldefeld, B., Variation of geometries based on a geometric-reasoning method. *Computer-Aided Design*, 1988, **20** (3), 117-126.
- Bouma, W., Fudos, I., Hoffmann, C., Cai, J. and Paige, R., A geometric constraint solver. *Computer-Aided Design*, 1995, **27** (6), 487-501.
- Brüderlin, B., Constructing three-dimensional geometric object defined by constraints. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics, ACM SIGGRAPH*, Chapel Hill, North Carolina, 1986.
- Brüderlin, B., Using geometric rewrite rules for solving geometric problems symbolically. *Theoretical Computer Science*, 1993, **2** (116), 291-303.
- Eggli, D. L., Sketching with constraints. Master's thesis, Department of Computer Science, University of Utah, February 1994.
- Eggli, D. L., Brüderlin, B. and Elber, G., Sketching as a solid modeling tool. In *Proceedings of the 1995 ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Salt Lake City, May 1995, pp. 17-19.
- Elber G., *IRIT 6.0 User's Manual*.
- Freeman-Benson, B. N. and Maloney, J., The DeltaBlue algorithm: An incremental constraint hierarchy solver. Technical report, 88-11-09, Computer Science Department, University of Washington, November 1988.
- Fudos, E., Constraint solving for computer aided design. Ph.D. thesis, Computer Science Department, Purdue University, August 1995.
- Hillyard, R. and Braid, I., Analysis of dimensions and tolerances in computer-aided mechanism design. *Computer-Aided Design*, 1978, **10** (3), 161-166.
- Hsu, C., Graph-based approach for solving constraint problems. Ph.D. thesis, Computer Science Department, University of Utah, May 1996.
- Hsu, C. and Brüderlin, B., A hybrid geometric constraint solver using exact and iterative geometric constructions. In *CAD Tools and Methods for Design System Development* ed. D. Roller and P. Brunet. Springer-Verlag, in press.
- Hwang, T. and Ullman, D., The design capture system: Capturing back-of-the-envelope sketches. *Journal for Engineering Design*, 1990, **1** (4), 339-353.

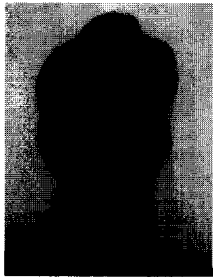
14. Hwang, T. and Ullman, D., Recognizing features from freehand sketches. *Computers in Engineering, ASME*, 1994, 1 67-78.
15. Light, R. and Gossard, D., Modification of geometric models through variational geometry. *Computer-Aided Design*, 1982, 14 (4), 209-214.
16. Nielson, G. and Olsen, Jr D., Direct manipulation techniques for 3D objects using 3D locator devices. In *Proceedings of the 1986 Symposium on Interactive 3D Graphics*, 1987.
17. Owen, J., Algebraic solution for geometry from dimensional constraints. In *Proceedings of the 1991 ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, May 1991.
18. Pugh, D., Designing solid objects using interactive sketch interpretation. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, 1992.
19. Salem, M., Skowronski, S. and Brüderlin, B., GDI reference manual. technical report, UUCS-92-031, Computer Science Department, University of Utah, 1992.
20. Serrano, D., Automatic dimensioning in design for manufacturing. In *Proceedings of the 1991 ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, May 1991.
21. Sohrt, W. and Brüderlin, B., Interaction with constraints in 3D modeling. *International Journal of Computational Geometry and Application*, 1991, 1 (4), 405-425.
22. Solano, L. and Brunet, P., Constructive constraint-based model for parametric CAD systems. *Computer-Aided Design*, 1994, 26 (8), 614-621.
23. Sutherland, I., Sketchpad, a man-machine graphical communication system. Ph.D. thesis, MIT, January 1963.
24. Verroust, A., Schoneck, F. and Roller, D., Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 1992, 24 (10), 531-540.



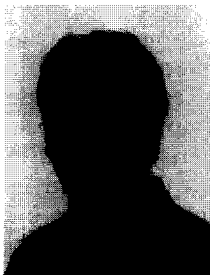
Gershon Elber is a senior lecturer in the Computer Science Department, Technion, Israel. His research interests span computer-aided geometric design and computer graphics. Dr Elber received a BS in computer engineering and an MS in computer science from the Technion, Israel, in 1986 and 1987, respectively, and a Ph.D. in computer science from the University of Utah, USA, in 1992. He is a member of the ACM and IEEE.



Ching-yao Hsu received his Ph.D. in computer science from the University of Utah, in 1996. His research interests are in geometric constraint solving, and geometric tolerances in CAD. Dr Hsu currently works at EDS-Unigraphics.



Beat Brüderlin is a professor of computer science, and chair of the computer graphics program, at the Technical University of Ilmenau, Germany. He received his Ph.D. in computer science from the Swiss Federal Institute of Technology, ETH Zurich, Switzerland, in 1987, and MS in physics, from the University of Basel, Switzerland, in 1981. In 1988 he became an assistant professor at the University of Utah, and was a visiting professor at the University of Bern, Switzerland, in 1991 and 1994. In 1995 he co-founded Conceptual Geometry LLC. Professor Brüderlin's research interests include constraint-based interactive CAD, robust and tolerance-based geometric modelling, and conceptual design approaches.



Lynn Egli received a BS and MS in computer science from the University of Utah. He currently develops modelling tools for flight simulators at Evans and Sutherland.