FIG

XIX CONGRESS

INTERNATIONAL FEDERATION OF SURVEYORS
FÉDÉRATION INTERNATIONALE DES GÉOMÈTRES
INTERNATIONALE VEREINIGUNG DER VERMESSUNGSINGENIEURE

10.–19.6.1990
HELSINKI FINLAND

PAPER. EXPOSÉ. BERICHT.

Land Information Systems
Systèmes d'Information du Territoire
Landinformationssysteme

3
COMMISSION

FIG XIX INTERNATIONAL CONGRESS
HELSINKI, FINLAND, 1990

HUMAN INTERACTION WITH GIS/LIS: EDITING GEOMETRIC MODELS

INTERAKTION MIT GIS/LIS: EDITIEREN GEOMETRISCHER MODELLE

INTERACTION ENTRE L'UTILISATEUR ET LES SIT: EDITER DES MODELES GEOMETRIQUES

Werner Kuhn (Switzerland) and Andrew U. Frank (U.S.A.)

SUMMARY

The acquisition and interactive manipulation of geometric data are fundamental tasks in the use of geographic and land information systems (GIS/LIS). Most systems offer methods for these tasks which are derived from manual geometric constructions. They tend to have restricted and complex user interfaces. The paper explains these shortcomings and proposes an alternative approach, based on the idea of sketching and declaring geometric constraints.

ZUSAMMENFASSUNG

Das Erfassen und interaktive Verändern von geometrischen Daten sind grundlegende Tätigkeiten bei der Benützung von Geographischen und Land-Informationssystemen (GIS/LIS). Die meisten Systeme bieten dafür Methoden an, die vom manuellen geometrischen Konstruieren abgeleitet sind. Sie haben meist einschränkende und komplexe Benützerschnittstellen. Der Beitrag erläutert diese Nachteile und schlägt einen alternativen Ansatz vor, der von der Idee des Skizzierens und der Angabe geometrischer Bedingungen ausgeht.

RESUME

L'acquisition et le traitement interactif de données géométriques sont des tâches fondamentales dans l'utilisation des systèmes d'information du territoire (SIT). La plupart des systèmes offrent des méthodes pour ces tâches qui sont dérivées des constructions géométriques manuelles. Ils ont souvent des interfaces restreints et complexes. L'article discute ces problèmes et présente une approche alternative qui est fondée sur l'idée d'esquisser et de déclarer de contraintes géométriques.

1. Introduction

User interfaces for word processing and administrative tasks like accounting or inventory have developed considerably in the last decade. Also, attractive graphics software allows to prepare illustrations and other graphical products in much easier and more intuitive ways than ever before. So called desktop (or electronic) publishing environments integrate these powerful tools.

When dealing with spatial information, however, we observe a lower quality and a slower evolution of user interfaces. Common problems are that

- commands are difficult to learn and use
- system capabilities are not evident from the user interface
- parts of systems have inconsistent interfaces
- users have to worry about making (fatal) errors
- it is difficult to get an overall picture of how a system behaves
- systems react in unexpected ways
- documentation is often a nightmare by itself, instead of a help.

A problem at a more fundamental level is that systems often don't serve the needs of their users for certain tasks. They may be powerful, offering a wide functionality, but lacking *usability*. Or they may emulate traditional techniques of problem solving and thereby miss many opportunities of the new technology. In both cases, users often wish they could return to the old ways of getting things done.

This paper focusses on user interfaces for plane *geometric construction tasks*. The goals of the reported work were to determine a 'division of labor' between user and system that supports the user's needs and to achieve an effective communication between user and system. In order to reach these goals, it was necessary to analyze first the nature of geometric construction tasks. This section summarizes some of the important characteristics.

Geometric construction tasks are characterized by *constraints*, i.e. conditions to be satisfied by a solution. Examples include constraints on parallelity, orthogonality, distances, radii, angles, areas etc. The concept of constraints plays an important role in *design tasks*: An object being designed has to 'function', i.e. to satisfy certain requirements which can be stated as constraints. Every design process involves a specification of the desired artifact by constraints [Simon 1981, Fischer and Böcker 1983]. Geometric construction tasks can in fact be regarded as design tasks: An engineer, architect or draftsman designs geometric models by specifying a set of constraints. We will refer to geometric construction tasks as design tasks throughout this paper.

The constraints are usually expressed by refering to a *sketch* of the situation, annotating it or separately stating the conditions. The sketch represents more than a simple illustration of the task at hand. It takes care of the ambiguities in task descriptions and supports the cognitive process of finding a solution. Neither the graphic depiction by the sketch nor a description by words alone could generally contain all the information necessary to find a solution.

Designs evolve in stages from sketchy information to more precise data to situations which are commonly overconstrained and require some form of optimization to find a satisfactory solution. At the beginning, a task description is usually *incomplete*: The designer might have to look up values in tables, consult maps and registers, make assumptions on some properties, etc. Once all the available information is collected, it normally overdetermines the situation and contains *inconsistent* requirements.

Geometric constraints are generally not of equal importance and they contain some degree of *uncertainty*. The reason for this uncertainty lies in the information sources involved: Measurements are inherently subject to errors and design constraints are not always strict or precise [Schenk 1986]. A solution method for design tasks should be able to cope with these uncertainties.

The remainder of this paper discusses the traditional 'division of labor' between a designer and a system and proposes an alternative (section 2). A constraint solving method, as required by the alternative approach, is outlined in section 3. Section 4 shows an interface design for the approach. The paper ends with conclusions, discussing related work, the feasibility of the approach (based on an existing implementation) and open problems.

2. Interaction paradigms for design tasks

User interfaces for design tasks are commonly built in analogy to traditional drafting techniques. They offer a set of elementary ("ruler and compass") construction steps to construct geometric figures. We discuss this 'constructive' interaction paradigm and suggest a 'declarative' alternative which is based on the notion of geometric constraints rather than on geometric construction steps.

2.1. Constructive interaction

The constructive interaction paradigm is based on an analogy to the familiar *construction steps* in euclidian geometry. System designers assume that users are experienced with drafting tools like rulers and compasses. They capitalize on that experience by simulating these tools electronically. The resulting user interfaces offer an extended notation of elementary construction steps. To solve design tasks, users can choose and combine commands from a menu (see figure 1) or in a command language.

Such a simulation of manual construction methods allows for familiar operations and promises easy learning of a system. With a sufficiently rich collection of construction steps, many common tasks can be solved in just one step or by a short sequence of operations. However, the analogy carries some limitations with it as well. In complex design tasks, users have to derive sequences of operations all by themselves, without help from the system. Because the sought geometric elements (points and lines) can not be visualized by the system until they are completely determined, the users often need to work with sketches outside the system to develop solution plans. Essentially, they have to solve the tasks themselves, by combining appropriate construction steps from the selection offered by the system.

More importantly, most design tasks are inherently *under- or overconstrained.* A constructive system, however, can only handle exactly determined situations. For example, circles have to be defined by three points, or two tangents and a point, or corresponding data, but not more (e.g. two points and two tangents) and not less (e.g. preliminary information that the endpoints of two lines are linked by a circular arc, without knowing the arc's size yet). Additional difficulties are caused by *uncertainty*in the data. A constructive interface forces users to supply complete and precise data for every construction step.

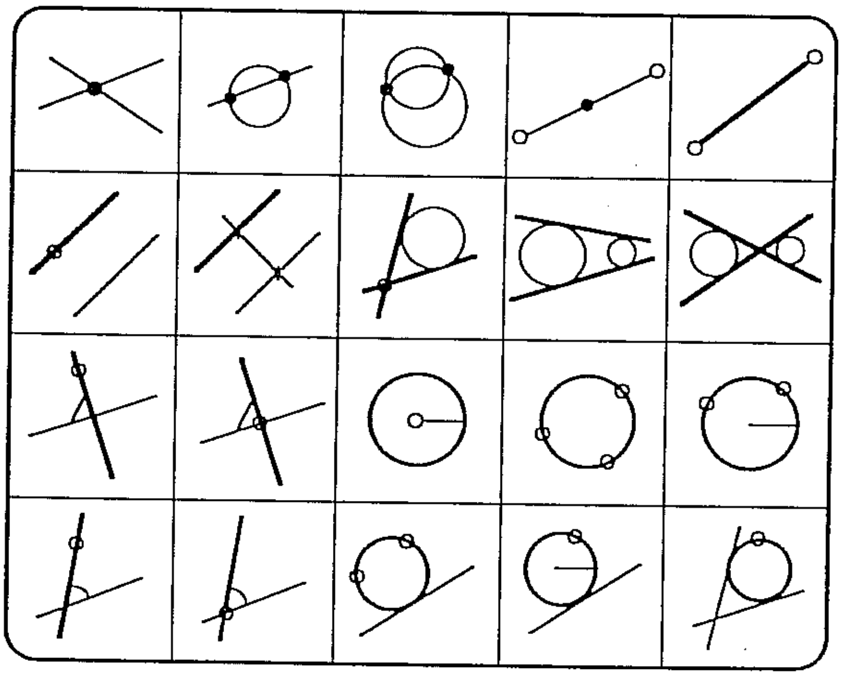


Figure 1: A typical menu with construction steps

Design tasks, particularly in engineering, often involve constraints on *arc lengths* or *polygon areas*. Examples are road designs and parcel subdivisions. Since euclidian constructions contain neither arcs nor polygons as concepts, constructive systems have to provide extra commands for such occasions. Because of the unlimited possibilities for variations in constraints, these commands can only cover special cases, e.g. splitting a polygon with a line which is parallel (but maybe not orthogonal or having a general angle) to an existing boundary.

The number of commands and, consequently, the *complexity* of the user interface are growing, due to the need for special case solutions and high-level construction steps like particular combinations of points, radii and tangents to define circles. Overloaded command menus are commonly observed with constructive user interfaces in practice.

All these shortcomings restrict the usability of constructive systems for practical design tasks in fields like engineering and architecture. The goal of 'user friendliness' which motivated the analogy with familiar drafting operations, is defeated by the limited practical usability of the resulting systems. These are often electronic drafting boards rather than design systems. While this represents already a significant improvement over manual drafting technology, computers can and should be used in more effective ways than just to emulate the technology they replace.

2.2. Declarative interaction

Adequate computer support for design tasks [Greenberg 1984] should deal with the complex and ill-defined nature of the tasks as well as with the fact that design information is collected in a continual process and necessarily contains uncertainty . How can these requirements be satisfied without sacrificing the advantages of an analogy with familiar concepts?

While analogies can be powerful means to facilitate learning and operating of systems [Smith et al. 1982], we believe that it is crucial to choose them at the right level. In the case of design tasks, an analogy can not only be established at the level of task *execution* (by construction steps) but also at the level of task *description.* This is the idea behind the declarative interaction paradigm.

Users should be able to describe their knowledge about a design task and let the system propose solutions [Kuhn 1986]. The familiar means for such a task description are a *sketch* of the desired situation and an indication of *constraints* which have to be satisfied. This kind of sketch-based communication has been successful between engineers and draftsmen over centuries. It can now be applied to human-computer interaction, given today's hard- and software environments.

In the declarative interaction paradigm, sketching has two main purposes: First, it serves to specify the *topology* of a situation, e.g. which points and lines exist and what incidences they have. Not only is topological information normally represented correctly in a sketch of a situation (as opposed to the distorted metric information), but sketching seeems also to be the most natural way for humans to express it.

Second, a sketch is used to define an *approximate solution* to a design task. The declarative interaction paradigm presupposes that a sketch determines a situation completely and uniquely, though not precisely. This ensures that every task will be computationally overdetermined: The explicitly stated constraints are improving on an already determined approximate solution.

A declarative user interface must allow to describe all geometric constraints possibly occurring in design tasks. This requires some thought about what kinds of constraints have to be dealt with. A theoretical study of geometric constraints in two dimensions has been done in [Kuhn 1989]. Based on differential geometry, a complete set of primitive functions and relations has been identified and a logic-based language for their combination to simple and complex constraints has been defined. With this language (which will not be discussed here), two-dimensional geometric models of arbitrary complexity can be described.

With the declarative interaction paradigm, users don't have to worry about construction steps necessary to satisfy the constraints. They can fully concentrate on describing their knowledge of a situation. Over- and underdeterminations are normal and do not have to concern a user at the time of the task description. Users can, however, ask the system where more information is needed or where conflicting information has been stated.

This approach allows to describe as much as is known at a particular time about a design task. When additional information becomes available, it can be entered, or the information already described can be edited. The sequence of problem description, solution planning, plan execution and result assessment [Polya 1945] is broken up: Since each constraint improves on a previous solution, it can immediately be evaluated and its effect can be displayed. This supports the interaction style of *direct manipulation* [Hutchins et al. 1986]: users see the results of their operations immediately and can decide to proceed or to undo them.

Finally, an essential difference between the constructive and declarative interaction paradigms is the following: In a constructive system, the constraints of a task are not part of the system's geometric model. They are "forgotten" after the user has translated them into construction steps. With the declarative paradigm, on the other hand, constraints are recognized as central to a task and constitute the fundamental entities of a geometric model. They represent an *explicit description of task knowledge* in a system and can be inspected, modified or deleted at any time.

Section 4 will illustrate a possible declarative interface for design tasks. Before getting to it, however, we need to show how a system can actually solve a task described by constraints.

3. Satisfying constraints by least squares adjustments

An automatic satisfaction of geometric constraints requires a method to solve a system of simultaneous equations representing the constraints. This method has to cope with overdetermined systems of equations and with uncertainty in the constraints. A least squares adjustment fulfills these requirements and was adopted as constraint solving method in our approach. This section briefly explains how constraints can be satisfied by a least squares solution along the lines of [Schmid and Schmid 1965].

3.1. Describing constraints analytically

Geometric constraints are assertions about topological and metrical properties and relations of geometric objects. These assertions can analytically be expressed by equations. The equations contain

- constants (real numbers), standing for observed values and the like;
- variables, standing for point coordinates and for parameters describing curves;
- function terms, expressing metric properties of points and curves.

Summarizing all variables in a vector **u** and all functions (including constants) in a vector function f, we can represent the set of geometric constraints in a design task by

$$f(u) = 0$$

Since the sketch represents an approximate solution, we can regard it as defining a set of *initial constraints*. For each variable u_k, the sketch defines an approximate value u_k^o, leading to a constraint

$$u_k - u_k^o = 0$$

Including these initial constraints, the system $f(u) = 0$ will always be overdetermined, except in the trivial case where no constraints have been added to the sketch.

As we discussed earlier, geometric constraints are subject to uncertainty. This means that the vector equations representing them have to include a vector of (unknown) *corrections*:

$$f(u) = v$$

For example, if a distance between two points is constrained to be 10 meters, allowing for a slight derivation, the corresponding equation is

$$\sqrt{[(x_2-x_1)^2 + (y_2-y_1)^2]} - 10 = v$$

If we regard the corrections v as random variables with an expected value of zero, they have an associated variance σ_v^2. Tchebyscheff's inequality [van der Waerden 1957] shows that the variance determines how much a random variable can deviate from its expected value:

$$\Pr(|v| \geq c) \leq \sigma_v^2/c^2$$

By indicating a variance for every constraint, the corresponding correction can therefore be limited. The smaller a tolerated correction is, the smaller the variance has to be. Variances can therefore be used as a measure for decribing the uncertainty of constraints.

3.2. Satisfying a system of constraints

The vector equation $f(u) = v$ is an underdetermined non-linear system. Its linearized form shall be

$$A\Delta - l = v$$

where A is the Jacobian matrix of $f(u)$, Δ represents the differences between the parameters and their approximations $u^{(o)}$, and l is $f(u^{(o)})$.

Applying the least squares principle to the corrections v:

$$\|v\| = v^T P v \rightarrow \text{Min.}$$

leads to the familiar normal equations

$$A^T P A \Delta = A^T P l$$

and from them we obtain the least squares estimates

$$\hat{\Delta} = (A^T P A)^{-1} A^T P l$$
$$\hat{u} = u^{(o)} + \hat{\Delta}$$

The solution of this linearized system has of course to be iterated to satisfy the non-linear constraints. The iteration can be stopped when the change in the parameters falls below a certain threshold.

So far, we have not made any assumptions about the matrix P (except that it should be positive definite in order to guarantee a positive length of the vector v). It can be used to represent the uncertainties in the constraints: If the variances representing the uncertainties are summarized in a covariance matrix K_v, the matrix P can be selected as its inverse. Since P only needs to be known up to a constant factor (as can be seen from the normal equations), we get:

$$P = \sigma_o^2 K_v^{-1}$$

Note that no assumption about normal distribution is necessary to solve the constraint system. The least squares principle is not dependent on such an assumption and minimizes the corrections independently of their probability distribution. A normal distribution is only required for a statistical analysis of the results, e.g. to test whether the corrections obtained are compatible with the chosen variances.

More details on using least squares adjustments for solving constraint systems can be found in [Kuhn 1989]. This reference also contains an extension of the method to constraints involving inequalities, and an algorithm for a sequential adjustment, allowing to compute an updated situation for every new constraint.

4. A geometry editor

A declarative user interface for a design system serves to create, view and edit geometric models and can therefore be regarded as an *editor for geometric models*. This section shows a possible design of such an editor, based on the interaction style of direct manipulation. The screens shown are intended to illustrate the idea of declarative interaction and are *not* taken from an actual implementation. The presupposed interaction devices (a bitmap screen and a mouse or another pointing and drawing device) as well as software support for windows, icons, and menus are readily available for low-cost workstations and personal computers.

The proposed editor allows to describe geometric models by sketching and by building expressions in a constraint language. The system represents the models graphically (figure 2), displaying the geometric elements as precisely as they have been described at any given instant. This graphic view of a model also provides the context for sketching: Users sketch new elements directly in this view, relating them to existing elements.

By selecting one of three different sketching tools for points, lines and arcs, users describe the existence, dimension, and shape of elements. Then, by positioning a point or sketching the path of a line or arc, the approximate position, orientation, and size of the element are described. The relative positions of points, lines, and arcs define their incidences: When a point is positioned on a line or arc or when a sketched path crosses an existing element, the system infers an incidence between these elements.

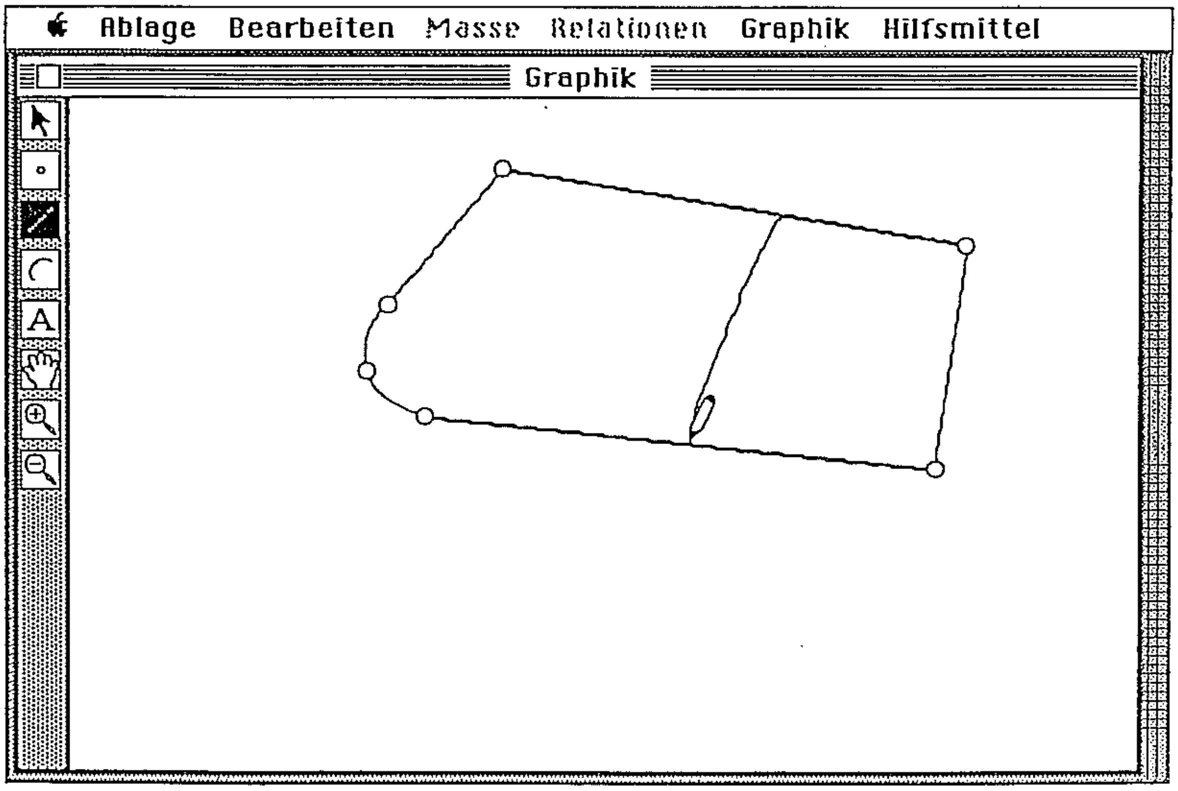


Figure 2: Sketching a new boundary line which splits an existing parcel. Whenever the sketched path intersects an existing line, the system highlights the line and records an incidence. The sketched path implicitly defines approximate positions for the new boundary points.

To state explicit constraints, users select the geometric elements involved and choose the appropriate functions or relations from menus. For example, to state that two lines should be orthogonal, the two lines and the relation 'orthogonal' are selected (figure 3). Note that in the menus, only those functions and relations can be chosen which are applicable to the currently selected geometric elements.

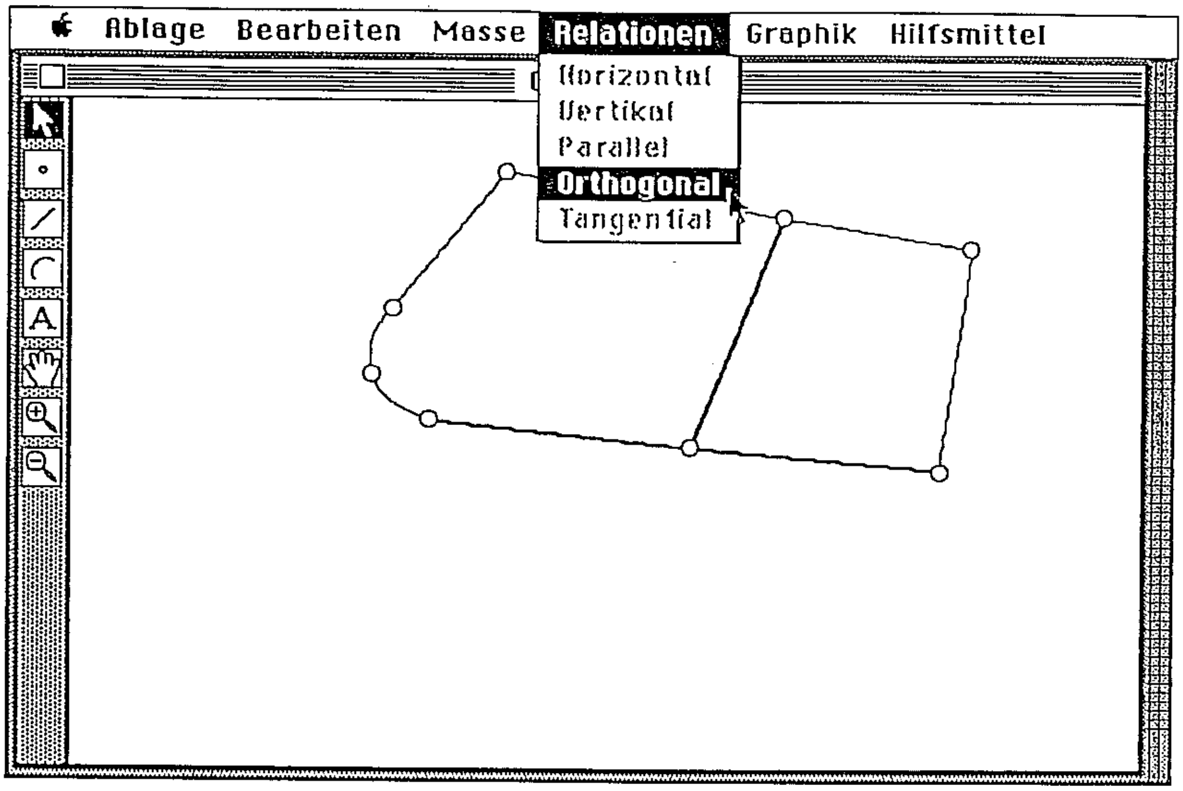
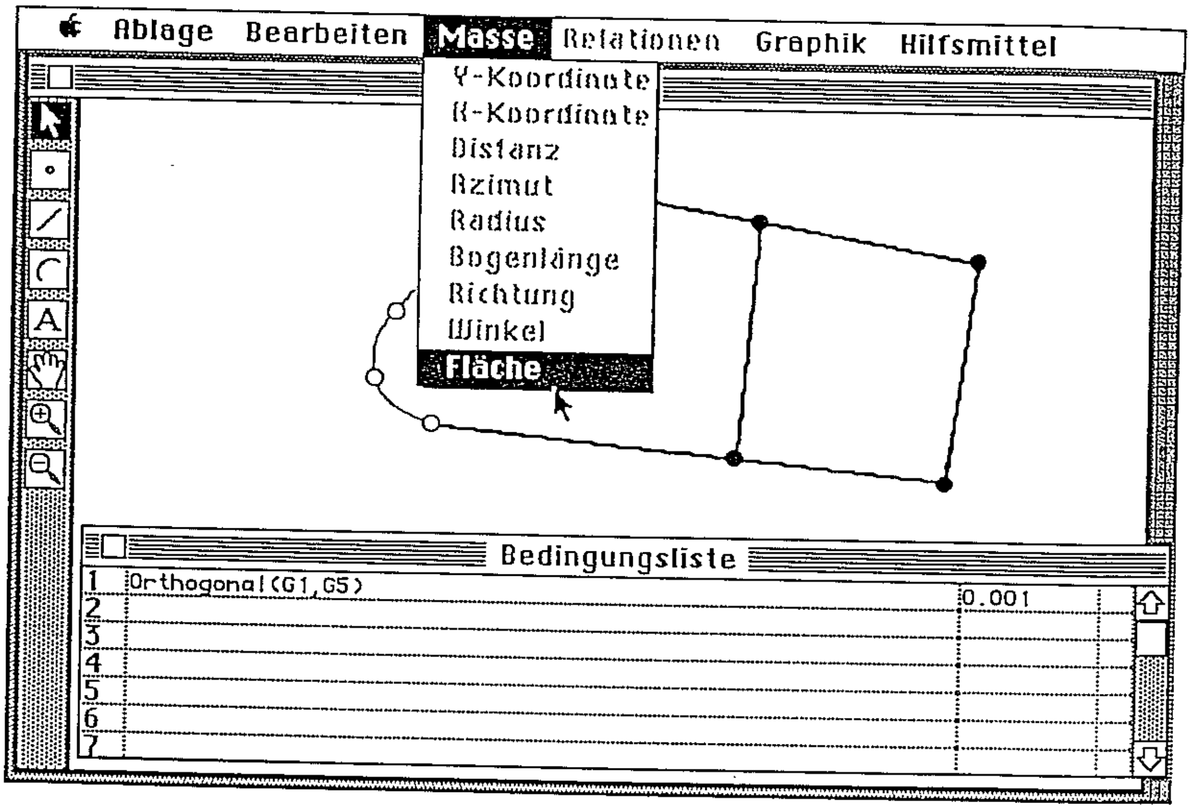


Figure 3: The system has evaluated the sketched information and displayed the new boundary line and points as far as they have been determined. The user has selected the new and an existing boundary to state the constraint that they be orthogonal.

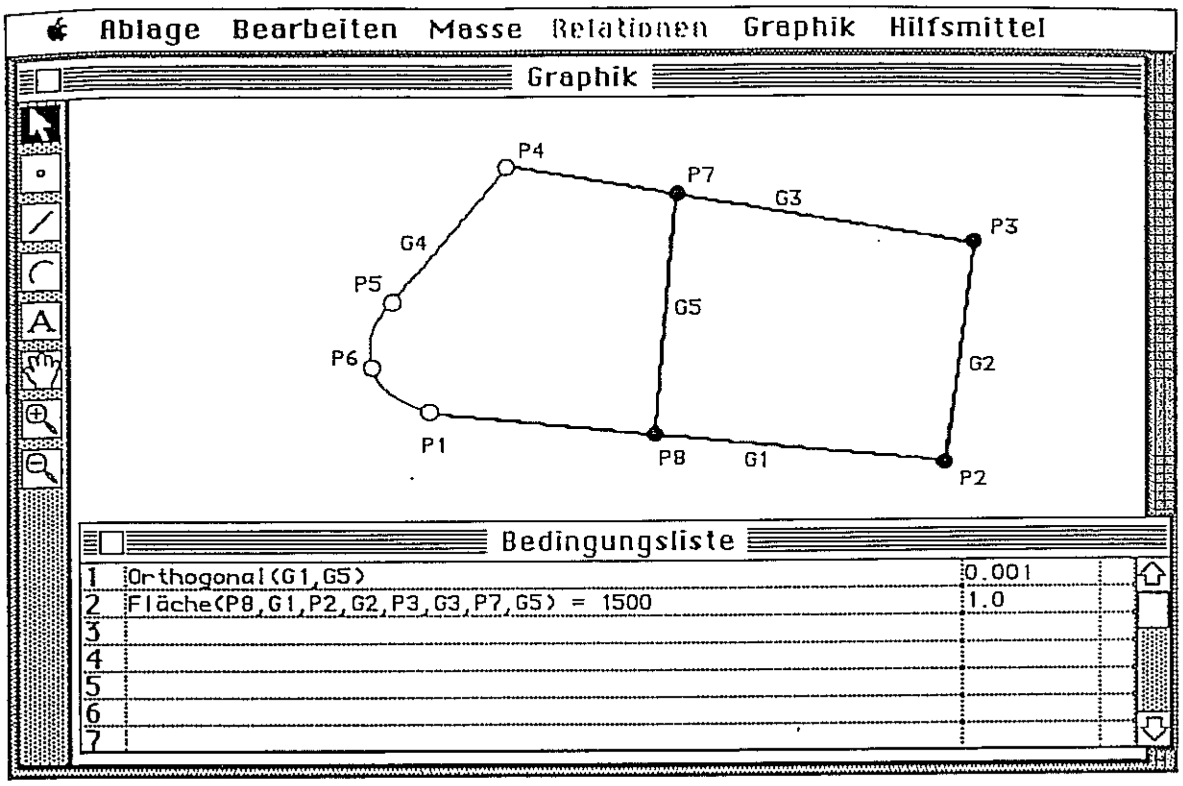
Functions are used to state constraints involving scalar values. For example, a constraint on the area of a polygon is stated by selecting the area (represented by the bounding polygon) and choosing the area-function from a menu (figure 4). The system then displays the current value of the area and the user can type it over with the desired value (figure 5). Thus, the method to define constraints serves at the same time as a *query language* for metric properties.

The system displays constraint language expressions in a separate window. In this *constraint editor* window, users can subsequently modify or delete constraints. Experienced users can also state constraints by directly typing the corresponding expressions in this window, instead of selecting elements, functions, and relations from the graphic view and the menus. The last column in the constraint editor window shows the variances associated with the constraints. The system sets default values for them which can be modified by the users.



1	Orthogonal(G1,G5)	0.001	
2			
3			
4			
5			
6			
7			

Figure 4: The system has displayed the orthogonality constraint in the constraint editor window ('Bedingungsliste') and updated the geometry to reflect its effect. The user has selected one of the new parcels (indicated by the highlighted boundary lines) to state an area constraint.



1	Orthogonal(G1,G5)	0.001	
2	Fläche(P8,G1,P2,G2,P3,G3,P7,G5) = 1500	1.0	
3			
4			
5			
6			
7			

Figure 5: After the user has stated the area constraint, the system displays the updated geometry. To link the constraint expressions to the graphic view, the system displays labels for points and lines which can be edited by the user to reflect actual names of elements in an application.

The constraint language furthermore provides built-in extensibility, i.e. it can be used to define higher-level constraints. In a constraint definition window (not shown here), users compose formulae which define higher-level relations by combining functions and relations which are offered by the system or have already been defined. For example, based on the length function, a constraint on the length of a polygon could be defined.

5. Conclusion

Systems which solve geometric problems based on constraints have been proposed since the early days of computer graphics. With the pioneering Sketchpad system [Sutherland 1963], geometric figures could be drawn and topological and metrical constraints could be formulated. Constraints were solved (among other, more restricted methods) by relaxation. This comes close to a least squares adjustment but does not allow for over- or underdetermination.

ThingLab [Borning 1979] was a direct descendant from Sketchpad, using the constraint idea to build a general-purpose, object-oriented simulation system. Another interesting approach, combining a declarative interaction with a constructive solution of geometric construction tasks was proposed in [Brüderlin 1987]. One of the very few commercial applications of the constraint-based approach is a CAD system developed from [Light and Gossard 1982]. However, all of these systems, as well as others [e.g. van Wyk 1982, Nelson 1985], could at most *detect* under- and overdeterminations but not accommodate them.

Some geographic and land information systems are incorporating constraint-based ideas, primarily for rectifying digitized data. A generalization of this approach towards providing constraining and editing functionality for geometric models (not just for digitized data) has yet to reach the user community.

The editor for geometric models proposed in this paper goes beyond related proposals by combining the following features:

- Geometric constraints are treated as first-class objects of geometric models. They can be viewed and edited at any time.
- An underlying constraint language allows to express a complete set of constraints and to define extensions.
- Taking a sketch as a first approximation allows to deal with incomplete knowledge in design tasks.
- A least squares adjustment is used to satisfy constraints. This allows for dealing with under- and overdetermined situations as well as uncertainty.
- Users have full control over the influence of individual constraints.

The *feasibility* of the approach has been shown by a prototype implementation on the Apple Macintosh™. This program, HILS (Human Interface to Least Squares), developed by Michael White [White 1987], has evolved into a standalone application for surveying and civil engineering design tasks. It offers the functionality of the proposed editor for points and straight lines.

Some *practical problems* remain to be dealt with in our approach. The convergence of the adjustment depends crucially on the quality of the sketch and on the uncertainty of the constraints. While the accuracy of the sketch turned out to be less critical than expected (as long as the topology is correct), the relative size of the variances is crucial. It is also not clear how default variances should be chosen realistically and what other measures of uncertainty the users should be able to use. An extension to curves other than circular arcs will probably require other constraint solving methods than least squares adjustments, due to problems in numerical stability [Krasznai 1988].

Many interesting *theoretical issues* should be addressed. Primarily, a graphic representation of the constraints, or at least a closer link with the graphic view of the model, would be helpful. One of the questions we are currently researching is how spatial relations can be visualized for this purpose. Another issue is how to describe tasks with incomplete or uncertain topological information, i.e. situations that cannot be sketched. Finally, a generalization of the approach to three dimensional models is desirable. This seems straightforward for the constraint language and the solution procedure, while the interaction would become more complex.

References

Borning, A.	1979	THINGLAB - A Constraint-Oriented Simulation; Laboratory; Ph.D. Thesis, Stanford University
Brüderlin, B.D.	1987	Rule-Based Geometric Modelling; ETH Zürich, Informatik-Dissertationen Nr.7
Fischer, G. Böcker, H-D.	1983	The Nature of Design Processes and How Computer Systems can support them; in: Integrated Interactive Computing Systems (Degano, P. and Sandewall, E.; Eds.)
Greenberg, D.	1984	The Coming Breakthrough of Computers as a true Design Tool; Architectural Record, September 1984
Hutchins, E.L. Hollan, J.D. Norman, D.A.	1986	Direct Manipulation Interfaces; in: User Centered System Design - New Perspectives in Human-Machine Interaction (Norman, D.A. and Draper, S.W.; Eds)
Krasznai, P.	1988	Die Klotoide in der Horizontalen Trassierungslinie; Dissertation, ETH Zürich
Kuhn, W.	1986	LIS and the User - Looking for Easier Ways to do Geometric Constructions; FIG XVIII Congress, Toronto, Canada 1986, P305.2
Kuhn, W.	1989	Interaktion mit raumbezogenen Informationssystemen - Vom Konstruieren zum Editieren geometrischer Modelle; Dissertation Nr. 8897, ETH Zürich
Light, R. Gossard, D.	1982	Modification of geometric models through variational geometry; computer-aided design, vol. 14, No. 4, July 1982
Nelson, G.	1985	Juno, a constraint-based graphics system; ACM SIGGRAPH'85, Computer Graphics, vol. 19, No. 3, July 1985
Polya, G.	1945	How to solve it; Princeton University Press
Schenk, T.	1986	Ausgleichung von Rechtwinkelzügen; Bildmessung und Luftbildwesen, vol. 54, No. 4
Schmid, H.H. Schmid, E.	1965	A Generalized Least Squares Solution for Hybrid Measuring Systems; Canadian Surveyor, vol. 19, No. 1
Simon, H.A.	1981	The Sciences of the Artificial; MIT Press

Smith, D.C.S. et al. 1982 Designing the Star User Interface;
BYTE, vol. 7, No. 4, April 1982

Sutherland, I.E. 1963 SKETCHPAD: A Man-Machine Graphical Communication;
System; Spring Joint Computer Conference 1963

v.d. Waerden, B.L. 1957 Mathematische Statistik;
Springer-Verlag

van Wyk, C.J. 1982 A High-Level Language for Specifying Pictures;
ACM Transactions on Graphics, vol. 1, No. 2

White, R.M. 1987 HILS - Development of a Human Interfaced Least Squares
Adjustment; Master Thesis, University of Maine

Acknowledgements:
The research reported here was conducted while the first author was at the Swiss Federal Institute of Technology (ETH), Institute of Geodesy and Photogrammetry, CH-8093 Zurich. Support from ETH under grant No. 0.330.079.80/7 and, for continuing research, from NCGIA under grant No. SES 88-10917 are gratefully acknowledged. Thanks also to Dr. Peter Krasznai for many stimulating discussions.

Authors' present address:
Werner Kuhn, Andrew U. Frank
National Center for Geographic Information and Analysis
and Department of Surveying Engineering
University of Maine
Orono, Maine 04469
U.S.A.
kuhn@mecan1.bitnet, frank@mecan1.bitnet