An approach to applying constraints in geometric modelling

Borut Žalik and Nikola Guid
University of Maribor, Faculty of Electrical Engineering and Computer Science, Slovenia

A 2D constraint-based geometric modelling system is considered in the paper. Constraints are solved by local propagation of known states. The weakness of local propagation (it cannot solve cyclic constraints) is explained by an example. A biconnected constraint description graph is used to support local propagation. As a result of constraint solving an acyclic constraint description graph is obtained and it can be observed as a parametric description of a geometric object. The acyclic constraint description graph stores the sequence of applying constraints for generation of instances. It is shown that the time complexity of the algorithm for generation of an instance is $O(n^2)$, where n is the number of basic geometrical elements included. In the last part of the paper we demonstrate how geometrical relations among geometrical objects can be established by the introduced geometrical constraints.

Keywords: geometric modelling, constraints, local propagation, constraint description graphs

1. Introduction

Design is a process of determining the properties of an emerging artefact. The design consists of a designer's idea and his/her skill to describe this idea (A. BIJL 1987). In geometric modelling, the designer is primarily concerned with the shape of a new geometric object. Therefore, sketches and figures have for a long time been the most natural way of describing geometry. To support the designer to express his/her ideas, geometric modelling systems have been developed and they are at the heart of current CAD/CAM systems. These systems are amazing in terms of their speed, visual effects, and capabilities to analyse designed objects. But, they also have some serious drawbacks such as:

- Current geometric modelling systems are built on the base of the theory of rigid solids. Therefore it is very hard to describe tolerances in a natural way.
- For solving geometrical problems, analytic geometry has been used exclusively. Many of these problems can be solved more efficiently using theorems of Euclidean geometry.
- User interfaces are inefficient. The main problems occur with the input to geometric modelling systems which are not able to accept the rich and concise engineering vocabulary. Instead, they force the user to think with and to use primitive low-level geometric modelling operators. An important step to rise the level of geometric modelling operators has been made with the introduction of features to design.
- Reusability of already created objects is rarely supported. In many cases only small modifications are needed to already constructed geometric objects to obtain a new object. If reuse of geometrical objects is not supported by the geometric modelling system, modifying an old object becomes awkward and time consuming.
- It is not possible to handle different geometrical meaning in various phases of the design and manufacturing process. The geometric modelling system should support the user from the beginning to the end of these processes.

The most serious drawback of current geometric modelling systems is that they essentially differ from the designers — they have no intelligence. Because of this, the design process can still be observed as the most primitive approach of trial and error. The process of searching for an artefact that satisfies given requirements is still exclusively a task of the designer. Current geometric modelling systems are usually not

able to support even this approach effectively. For example, *are they able to position a hole in the centre of an already existing slot?* Or *are they able to move a hole that lies in the centre of a slot if the slot is moved?* In many cases the answer is NO. The hole should be placed in the middle of the slot very carefully by the designer, taking into account the actual geometry of the slot. Or, if the designer wants to move the slot containing the hole, he/she has to mark all geometric elements describing the slot and the hole, then carefully change the adjacent geometrical elements of the slot and place the slot with a hole at a new position. These tasks are not creative and they just burden the designer. Because of this, geometric modelling systems should be able to capture somehow the designer's intention (to place a slot in the middle of an object, or to move a hole together with a slot). As Rossignac said, a geometric modelling system with such capability can be observed as an intelligent assistant for designers (J. R. ROSSIGNAC et al 1989). It should be able to perform automatically all well defined, but repetitive, awkward, and time consuming tasks instead of them. One of the possibilities to achieve this aim is to introduce geometric constraints into geometric modelling.

2. Related works

From the authors' point of view constraint-based geometric modelling systems are divided into two groups:

1. **Systems that use numerical methods for constraint solving exclusively**

   A constraint in such a system is immediately converted into an equation. In this way a system of (usually non-linear) equations is obtained. Unknowns in the system of equations are characteristic points of a geometric object. The position and the shape of a geometrical object is determined after the system of equations is solved. Representatives of geometric modelling systems from this group have been described, for example, by Fitzgerald (W. J. FITZGERALD 1981), Lin (V. C. LIN 1981), Light (R. A. LIGHT AND D. GOSSARD, 1982), and Nelson (G. NELSON 1985).

2. **Systems which use geometric reasoning for constraint solving**

Constraints are normally described in the form of predicates. The methods for constraint solving are very different. Some of the systems explore the ability of the designer to break the problem into independent or weakly connected subproblems, which can be solved independently using known procedural or fast numerical methods. Such systems enable interactive work. Let us take a brief summary of some systems from this group. Sunde constructed a very attractive 2D interactive system, where the user can get a picture of an object although it is not completely described (G. SUNDE 1987). The system solves the constraints by the help of two sets: the set of constrained angles (CA — set) and the set of constrained distances (CD — set). In a 2D system presented by Aldefeld, constraints are described by means of n-placed predicates (B. ALDEFELD 1988) (B. ALDEFELD et al 1991). The solution is reached in two steps: in the first step a construction plan is generated by a forward reasoning method. It defines the order of calculations. In the second step, the construction plan is used to calculate characteristic points of an object. Ando's system is intended for construction of iron-sheet elements (H. ANDO et al 1989). The most important result of his work is the introduction of predefined constraints which are used in the case of underdimensioning. Further, in his system the constraints have the priority against topology which is not the case in other systems. Kimura presented a 3D system where tolerances can be added to the constraints easily and naturally (F. KIMURA et al 1987). Tolerances are divided into three groups: tolerances of position, size, and shape. Constraints are solved by a geometric reasoning system written in Lisp. Suzuki presented a 2D system, where constraints are applied for defining topological structure of a geometric object, its geometrical structure, and relations among an object's geometrical and topological elements (H. SUZUKI et al 1990). Kondo presented PIGMOD, a 3D geometric modelling system (K. KONDO 1990). He introduced a history graph which stores the geometrical relations among geometrical elements and defines propagation of constraints. WRIGHT — a constraint based spatial layout system has been represented

by Baykan (C. A. BAYKAN AND M. S. FOX 1990). Constraints are used to represent arbitrary amounts of expertise in a uniform and principled manner. The system solves the constraints by constrained heuristic search. Verroust followed Aldefeld's and Sunde's approach (A. VERROUST et al 1992). The method is based on an expert-system shell which uses constraints and points as facts to evaluate the location of the model points. Kurlander noticed the difficulty of specifying constraints by traditional methods. In order to make constraints easier to declare, a method based on multiple examples — snapshots — has been suggested (D. KURLANDER AND S. FEINER 1993).

3. Determination of the world of observation

An experimental 2D system, based on geometric constraints, will be presented below. For input, it accepts a sketch — the correct dimensions of a geometric object are achieved interactively by inserting constraints. The constraints should be solved as rapidly as possible to enable the shape of the object to be updated. In this way the user, while still using a trial and error approach, can follow the effects of given constraints. The order of inserting constraints should not be important — the system should store the way of applying the given constraints, and a tool for their automatic execution should be available.

3.1. Determination of the sets of geometric elements

Basic elements in our system are points, lines, and circles. Let us define:

1. A finite non empty set of point names:

   $\mathbf{P} = \{p_1, p_2, \ldots, p_n\}$.

   For each point with the name $p_i \in \mathbf{P}$ $(1 \leq i \leq n)$, an ordered pair of real numbers exists (p_{ix}, p_{iy}): $p_{ix}, p_{iy} \in \mathrm{Re}$, where p_{ix} and p_{iy} are the coordinates of the point p_i.

2. A finite set of line names:

   $\mathbf{L} = \{l_1, l_2, \ldots, l_m\}$.

   Each line with the name $l_i \in \mathbf{L}$ $(1 \leq i \leq m)$ is determined by an ordered triple of real numbers $(l_{ix}, l_{iy}, l_{i\alpha})$: $l_{ix}, l_{iy}, l_{i\alpha} \in \mathrm{Re}$, $l_{i\alpha} \in [0, \pi)$, where (l_{ix}, l_{iy}) are the coordinates of a point through which the line l_i passes and $\tan(l_{i\alpha})$ determines the slope of the line l_i.

3. A finite set of circle names:

   $\mathbf{C} = \{c_1, c_2, \ldots, c_p\}$.

   Each circle with the name $c_i \in \mathbf{C}$ $(1 \leq i \leq p)$ is determined by an ordered triple of real numbers (c_{ix}, c_{iy}, c_{ir}): $c_{ix}, c_{iy}, c_{ir} \in \mathrm{Re}$, $c_{ir} > 0$, where (c_{ix}, c_{iy}) is the centre of the circle c_i and c_{ir} is its radius.

The common set $\mathbf{G}$ of all names of presented geometrical elements is defined as a union

$\mathbf{G} = \mathbf{P} \cup \mathbf{L} \cup \mathbf{C}$.

3.2. Predicates

To present geometrical constraints in an unambiguous way, predicates have been introduced. Each predicate consists of a name and a list of arguments written in parentheses. Arguments can be either constants or variables. Constants are elements of the set $\mathbf{G}$ and are always written in lowercase. In the formal description they are sometimes divided by "/" which is read as "or". In this case, at an actual position in the predicate one of the mentioned elements of a different subset of the set $\mathbf{G}$ can occur. Variables are marked as optional combinations of alphanumeric characters. Predicates are divided into three groups: dimensional, structural, and combined predicates. Similar divisions can be found in many authors (B. ALDEFELD 1988) (B. ALDEFELD et al 1991) (H. SUZUKI et al 1990). In the following sections some typical predicates from each group will be defined.

3.2.1. Predicates of dimensional constraints

These determine coordinates, distances, and angles. Constituent parts of their formal description are variables (or numbers) which can take their values from prescribed intervals. Variables presented in predicates of dimensional constraints represent parameters of a generic

geometrical object. It is natural to add the description of a tolerance into predicates of geometrical constraints. A tolerance is denoted by δ. The value δ determines the maximal tolerance deviation. If a predicate is written without the tolerance, it is assumed that the tolerance takes the default value δ_d. Predicates from this group are:

• *Point*(p_i, x, y, δ) — absolute coordinates of the point p_i are (x, y) within tolerance δ.

• *AngleValue*(l_i, α, δ) — absolute value of the slope of the line l_i is $\tan(\alpha)$ within tolerance δ.

• *Distance*(p_i, p_j, d, δ) — the distance between points p_i and p_j is d within tolerance δ.

• *PosXY*(p_i, p_j, xr, yr, δ_x, δ_y) — relative distance between points p_i and p_j in the x and y directions is xr and yr respectively within tolerances δ_x and δ_y.

• *Angle*(l_i, l_j, α, δ) — the angle between lines l_i and l_j is α within tolerance δ.

3.2.2. Predicates of structural constraints

Structural constraints refer to those spatial relations among geometrical elements which cannot be changed continuously. In a formal description they contain only the elements of the set **G**. Attributes of structural constraints are invariants of a generic object. They are set during the process of creation of an object and they are not changed subsequently. For example, if we set n parallel guiding lines, they should remain parallel permanently. The predicates of structural constraints do not contain the description of tolerances. For example, if the constraint *Perpendicular* is established between two lines, then these two lines are perpendicular without deviation. Of course, to express the tolerance of the angle between the lines, the user can employ the metric constraint *Angle*. Predicates from this group are:

• *Through*(p_i, l_j/c_k) — the line l_j (or circle c_k) passes through the point p_i.

• *On*(l_i/c_j, p_k) — the point p_k lies on the line l_i (or circle c_j).

• *Perpendicular*(l_i, l_j) — lines l_i and l_j are perpendicular.

• *Parallel*(l_i, l_j) — lines l_i and l_j are parallel.

• *Middle*(p_i, p_j, p_k, l_w) — the point p_j is in the middle of the points p_i and p_k; all three points lie on the line l_w. Predicate *Middle* can be expressed as a conjunction of the following predicates:

Middle(p_i, p_j, p_k, l_w) = *On*(l_w, p_i) & *On*(l_w, p_j) & *On*(l_w, p_k) & *Middle_c*(p_j) & *Middle_b*(p_i) & *Middle_b*(p_k).

The additional predicates *Middle_c* and *Middle_b* determine the role of each individual point as follows:

Middle_b(p_i) — the point p_i is the border point in the constraint expressed by the predicate *Middle*.

Middle_c(p_j) — the point p_j is the centre point in the constraint expressed by the predicate *Middle*.

3.3. Predicates of combined constraints

These constraints include both dimensional and structural information and can therefore be expressed as conjunctions of metric and structural predicates. For example:

• *Line*(l_i, p_j, x, y, α, δ_i, δ_j) — the line l_i has the slope $\tan(\alpha)$ within tolerance δ_i and passes through the point p_j whose position (p_{jx} = x, p_{jy} = y) is within tolerance δ_j.

Line(l_i, p_j, x, y, α, δ_i, δ_j) = *Point*(p_j, x, y, δ_j) & *AngleValue*(l_i, α, δ_i) & *Through*(p_j, l_i).

• *Distance*(l_i, l_j, d, δ) — the lines l_i and l_j are parallel and are at a distance d within tolerance δ.

Distance(l_i, l_j, d, δ) = *PosXY*(p_i, p_j, x, y, δ_x, δ_y) & *Parallel*(l_i, l_j) & *Through*(p_j, l_j).

4. Constraint description graphs

The work in our geometric modelling system begins with a sketch. Exact geometry is determined later in an interactive way by applying constraints. This is why one should not expect that constraints are inserted in an order which is the most suitable for the solution. To support an interactive design (or a design which is as close to it as possible) constraints have to be solved

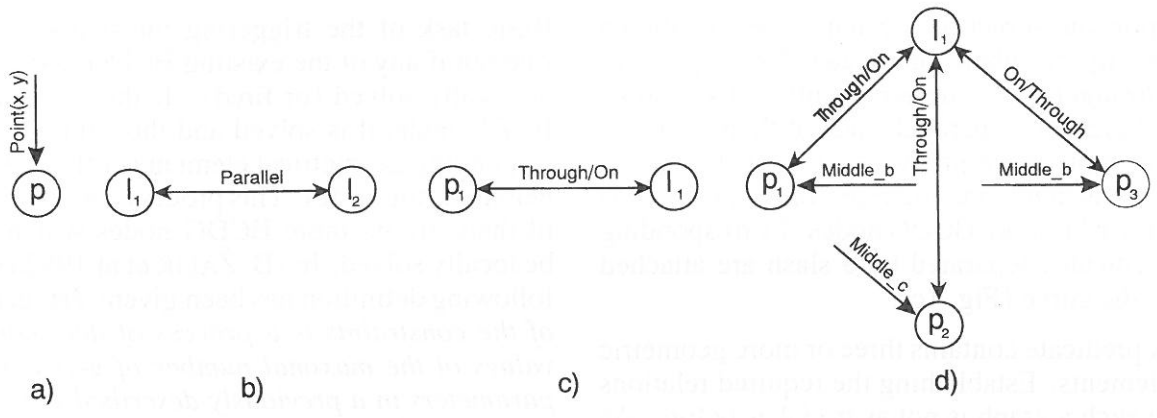

Fig. 1. Labelling of Biconnected Constraint Description Graph

as soon as possible. Because of this, local propagation of known states (called local propagation for short) has been chosen (W. LELER 1988). It is not the most powerful method for constraint solving, but it is quick, simple, and does not require that the whole problem is completely constrained. The well-known algorithm for constraint solving called DeltaBlue uses this method, too (B. N. FREEMAN–BENSON et al 1990) (M. SANNELLA et al 1993). In general, with case of cyclic constraints the local propagation does not find a solution. An example where local propagation fails is shown in section 4.3.

4.1. Biconnected constraint description graphs

In most cases a graph is used to represent how local propagation works. The known states propagate through the arcs of the graph. When a node gets sufficient information to solve itself, it fires and offers its data to the neighbouring nodes. This process can be represented as a chain-like reaction and it continues until there is no node which can be fired. To support local propagation and to make our system insensitive to the order of inserted constraints, a structure called a biconnected constraint description graph (BCDG) has been developed (B. ŽALIK et al 1992a). It is defined as a triple:

$$\mathrm{BCDG} = (\mathbf{N}, \mathbf{A}, \mathbf{D})$$

where:

- **N** is non-empty finite set of BCDG nodes,
- **A** is finite set of arcs between BCDG nodes, and
- **D** is non-empty finite set of real numbers (geometrical parameters) which control the actual appearance of the designed geometrical object.

The set **N** of BCDG nodes stores the elements of the set **G**. Each node contains exactly one element.

An arc from the set **A** is oriented. We represent it as a curve with an arrow. The arc carries a predicate which determines the appropriate geometrical relation. Depending upon the number of presented geometrical elements from the set **G** in a predicate, the following situations are possible:

1. A predicate contains exactly one geometric element. In this case the arc attaches exactly one BCDG node supplying it with information which is self sufficient. Such predicates are, for example, *Point* (which is shown in Fig. 1a) and *AngleValue*.
2. A predicate contains exactly two geometric elements. The arc connects exactly two BCDG nodes, establishing the required relation between them. We distinguish between two cases:
   - a relation with a symmetric effect (e.g. *Parallel*(l_1, l_2) shown in Fig. 1b). In this case there is no need to orient the BCDG arc. For practical reasons (to unify the algorithm) there are two oriented arcs and each of them carries the same predicate. To make our graphical presentation clearer, only one predicate is drawn on a curve representing the BCDG arc. The curve is equipped by two arrows indicating that there are physically two oriented arcs.
   - a relation between two nodes depends

upon the direction. Such a case is shown in Fig. 1c where predicates *On*(l_1, p_1) and *Through*(p_1, l_1) are represented. Two physical arcs are generated, each of them carrying the appropriate predicate. For the same reasons as in the previous case only one curve is drawn between BCDG nodes. Corresponding predicates separated by a slash are attached to the curve (Fig. 1c).

3. A predicate contains three or more geometric elements. Establishing the required relations in such a graph is not as trivial as before. As an example, the predicate *Middle*(p_1, p_2, p_3, l_1) is shown in Fig. 1d. The points p_1 and p_3 are boundary points denoted by the predicate *Middle_b*, the point p_2 is the point placed in the middle (centre) of the previous two and it is denoted by the predicate *Middle_c*. All points are placed on the line l_1.

Elements of the set **D** are parameters of metric constraints. By changing these parameters new instances of a generic geometric object are generated automatically.

A proper data structure allows us to access each BCDG node in linear polynomial time. Behind the main data structure representing the BCDG, one-way connected lists have been introduced for each set of geometrical elements. In our case, lists of points, lines, and circles are needed. Each record in the lists contains an identifier of the appropriate geometric element and a pointer into BCDG. Consider the case having n points, m lines, and p circles already included into BCDG (n, m, $p \geq 0$). The total of all included geometric elements is then $k = n + m + p$. To access any point p_i which is already a member of the BCDG, only $O(n) \leq O(k)$ checks have to be made.

4.2. Triggering mechanisms and the triggering table

Building a BCDG is very simple. Each time a new predicate is inserted by the user the BCDG is expanded. First, geometric elements presented in the predicate are checked to see whether they exist in the set **N**. If they do not, new BCDG nodes are created. Then new BCDG arcs with corresponding predicates are added. When all required changes in BCDG are performed, a triggering mechanism is made active.

Basic task of the triggering mechanism is to find out if any of the existing BCDG nodes can be locally solved (or fired). If there is such a BCDG node, it is solved and the value of corresponding geometrical element is offered to its neighbouring nodes. This process continues until there are no more BCDG nodes which can be locally solved. In (B. ŽALIK et al 1992a) the following definition has been given: *Triggering of the constraints is a process of determining values of the maximal number of geometrical parameters in a previously described part of a geometrical object, after an explicit parameter value has been set or a new geometric relation established.* Conditions which have to be fulfilled to fire a BCDG node are stored in a triggering table. The triggering table consists of:

- a predicate list; this stores the predicates which must be presented at the input of a BCDG node to fire this node,

- a condition predicate list; this contains the predicates which must exist in the immediately neighbouring BCDG nodes. To highlight this, let us observe the predicate *Distance*(p_i, p_j, d, δ). We would like to determine a point p_j if p_i is known. Both points have to lie on the same line l_k which is not in the list of parameters of the predicate. To determine the position of the point p_j, once the point p_i and the line l_k are known, the point p_j should be a member of the line l_k, too. Because of this, the BCDG node storing either the line l_k or the point p_i has to be visited. From this node an arc carrying the predicate *On*(l_k, p_i) (or *Through*(p_i, l_k)) should exist. Of course, the additional checking is not always necessary and the condition predicate list is empty.

- an exact data list; this stores exact values of geometrical elements which have to be known in neighbouring nodes. Namely, some geometrical elements can still have approximate values obtained from a sketch and such data cannot be used during local propagation. In addition, in some cases only a part of the graphical information is sufficient for solving a constraint. For example, if two lines are constrained by the predicate *Parallel*, there is no need to know the exact position of any line, it is enough to know only the slope.

- a calculated data item; this informs us which part of a geometric element is calculated. For

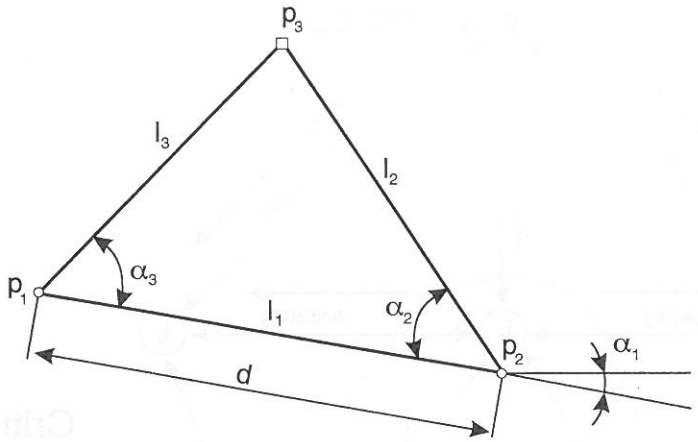

Fig. 2. Completely determined triangle

example, in the case of a circle, only its radius (or its centre point) can be determined.

If local propagation of known states is used for constraint solving, only well-determined and usually very simple calculations should be performed (such as finding the intersection of two lines). However, ambiguities can still occur in some cases. Let us observe again the predicate *Distance*(p_i, p_j, *d*, δ). An unknown point p_j can lie "on the left" or "on the right" side of the point p_i at the required distance d. To reduce such problems, the principle of minimal disturbance is explored. All solutions (in our case, two) are compared with old (or approximate) values and the one which differs the least is presented to the user first. If the user does not want this solution, the system may be utilised to get other solutions obtained. Of course, only one solution is then accepted by the system.

4.3. Weakness of local propagation; an example

The main weakness of local propagation of known states as a method for constraint solving is that it can use only the information which is local to the observed graph node. Therefore this some well-defined problems cannot be solved (W. LELER 1988). Let us demonstrate this with a simple geometrical problem. Consider a triangle shown in Fig. 2. The following data are known for it:

- exact position of the point p_3 (denoted by a squared marker),
- the slope $\tan(\alpha_1)$ of the line l_1,
- inner triangle angles α_2 and α_3, and
- the distance between the points p_1 and p_2.

These data completely determine the triangle and they can be formally expressed by using the predicates introduced in Chapter 3. In Table 1 they are listed without stating tolerances.

The resulting BCDG is shown in Fig. 3, where the propagation of known states is represented by thicker arrows. By the predicate *Point* exact position of the point p_3 is determined and its position is passed to the BCDG nodes storing lines l_2 and l_3. The slopes of these two lines are calculated from the known slope of the line l_1 using the *Angle* predicates and these two lines become completely known. By means of l_2 and l_3, neither the position of the line l_1 (only its slope is known), nor the points p_1 or p_2 can be

1.	Point(p_3, x, y)	2.	Through(p_3, l_2)
3.	Through(p_3, l_3)	4.	Angle(l_3, l_1, α_3)
5.	Angle(l_2, l_1, α_2)	6.	AngleValue(l_1, α_1)
7.	Distance(p_1, p_2, d)	8.	On(l_1, p_1)
9.	On(l_3, p_1)	10.	On(l_2, p_2)
11.	On(l_1, p_2)		

Tab. 1. Predicates describing a triangle

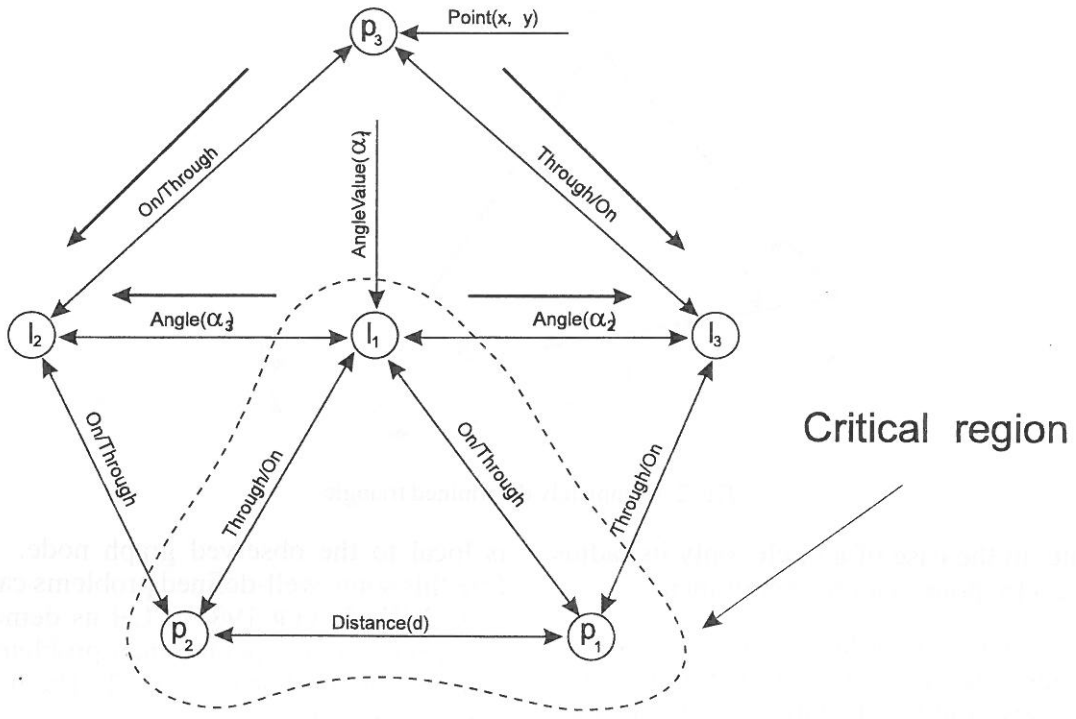

Fig. 3. Biconnected Constraint Description Graph of a triangle

determined. Exact position of the line l_1 can be calculated by knowing the point p_1 (or p_2) and the exact position of the point p_1 (or p_2) can be determined using the line l_1. This situation can be observed as a cycle. We call such parts in the graph BCDG critical regions. The process of constraint solving is stopped and the system waits for an additional constraint. If the user inserts it, overdimensioning occurs. For example, the user inserts a new constraint *Distance*(p_1, p_3, d_3), the position of the point p_1 can be calculated now. From it, BCDG nodes carrying the line l_1 and the point p_2 are reachable. Position of the point p_2 can be now determined in two ways:

- directly from the point p_1 using the *Distance*(p_1, p_2, d) predicate, or
- as an intersection between the lines l_1 and l_2.

In general, two different positions of the point p_2 can be obtained. Because of this, inserting redundant (and possibly conflicting) additional constraints is not a good solution.

What we need is a tool able to determine that one part of a geometrical object is already completely correctly described. After that, critical regions have to be extracted and solved by other, more powerful, techniques for constraint solving. A typical representative is relaxation, a classical iterative numerical method (W. LELER 1988). Of course, it needs adequate starting values to find a solution. Approximate values of geometrical elements provide a good hint. Critical regions are subgraphs of a BCDG and the relaxation can be performed in a reasonable time.

4.4. Acyclic constraint description graph

All arcs in BCDG, which have not been used during constraint solving, are marked with a flag unused and these arcs can be removed from the BCDG. An acyclic constraint description graph (ACDG) is obtained. ACDG = (**N**, **A**, **D**) has the same number of nodes as the BCDG from which ACDG has been extracted. If two ACDG nodes are connected, there is exactly one arc connecting them. Each arc carries a predicate describing the required relation between nodes. The elements of the set **D** have to be accessible to the user and they must represent the entries into ACDG. Each geometric object determined by the ACDG can be observed as a parametrically described geometrical object.

The following work gives a simple example of design in our system. Suppose the user has

1.	Point(p_1, x, y)	2.	Through(p_1, l_1)
3.	AngleValue(l_1, α_1)	4.	On(l_1, p_2)
5.	Distance(p_1, p_2, d_1)	6.	Perpendicular(l_1, l_2)
7.	Through(p_1, l_2)	8.	Parallel(l_2, l_3)
9.	Through(p_2, l_3)	10.	On(l_2, p_8)
11.	Distance(l_1, l_6, d_2)	12.	On(l_6, p_8)
13.	On(l_6, p_3)	14.	Parallel(l_3, l_4)
15.	On(l_3, p_3)	16.	On(l_6, p_7)
17.	Distance(p_8, p_7, d_5)	18.	On(l_6, p_4)
19.	Distance(p_3, p_4, d_3)	20.	Through(p_7, l_4)
21.	Parallel(l_4, l_5)	22.	Through(p_4, l_5)
23.	On(l_4, p_6)	24.	On(l_5, p_5)
25.	Distance(l_7, l_1, d_4)	26.	On(l_7, p_6)
27.	On(l_7, p_5)		

Tab. 2. Predicates describing the object containing a slot

already inserted the sketch of an object containing a slot. For determining of the correct dimensions and for establishing the required geometrical relations shown in Fig. 4 the constraints given by means of predicates are inserted. These are shown in Table 2.

Constraints are solved by the BCDG and, as a result, the corresponding ACDG is obtained. A step-by-step example of constraint solving by using BCDG can be found in (B. ŽALIK et al 1992b). Suppose that the object from Fig. 4 has already been completely determined and that its ACDG (shown in Fig. 5) has been successfully generated. The user has generated the instance shown in Fig. 4a using the following values of metric constraints: $p_{1x} = 10$, $p_{1y} = 10$, $\alpha_1 = 0°$, $d_1 = 100$, $d_2 = 50$, $d_3 = 50$, $d_4 = 30$, and $d_5 = 15$ (the values can be set, for example, in millimetres). If the user is not satisfied with the appearance of the generated object, it can be changed very simply. The user perhaps wants to increase the distance d_5 to 25 and decrease the distance d_3 to 40 (Fig. 4b). In this case there is no need to recalculate the whole object completely. In fact, only positions of the points p_4, p_5, p_6, and p_7 have to be changed. This information is effectively captured by the ACDG shown in Fig. 5. Only two new entries into ACDG occur in this case (the distances d_3 and d_5). At first, the ACDG nodes containing geometrical elements presented in the constraints *Distance(p_3, p_4, d_3)* and *Distance(p_7, p_8, d_5)* are observed. Let us observe the constraint *Distance(p_3, p_4, d_3)* for a while. At first, the algorithm should discover, which point (p_3 or p_4) should be changed to satisfy the given constraint. As the point p_4 is derived from the point p_3 position of the point p_4 should be recalculated. For the same reasons position of the point p_7 is corrected in the constraint *Distance(p_7, p_8, d_5)*. Now, all ACDG nodes accessible from p_4 and p_7 are visited (these nodes store l_4, l_5, p_5,

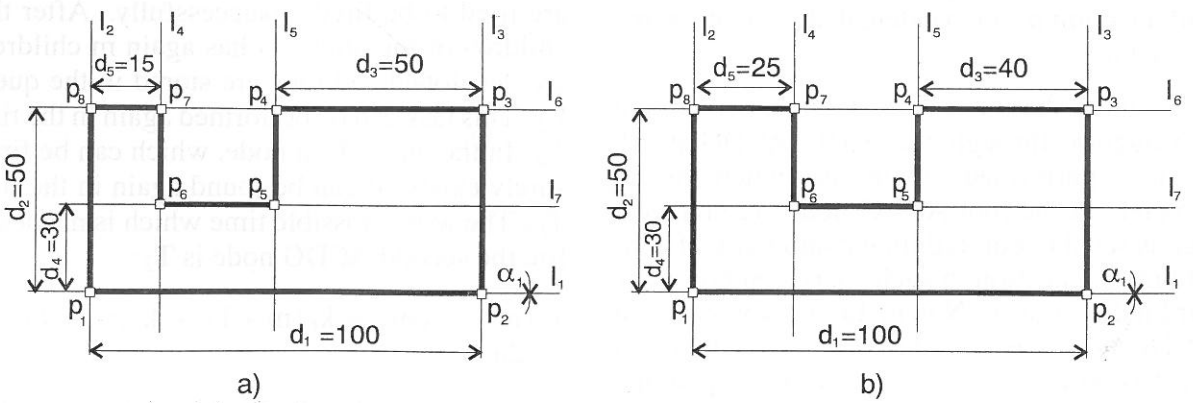

Fig. 4. An object containing a slot

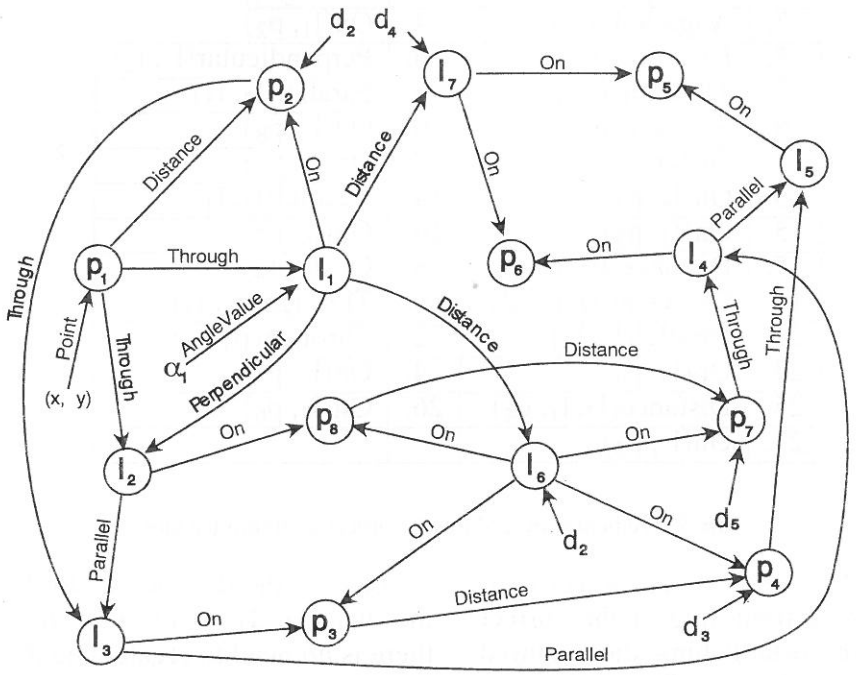

Fig. 5. Acyclic Constraint Description Graph of the object containing a slot

and p_6) and they are marked as ACDG nodes storing only approximate values of geometrical elements. After that, the determination of new geometrical values in the newly marked ACDG nodes is started. At first, new positions of the points p_4 and p_7 are calculated. The line l_4 is reachable from the point p_7. Its new geometrical values can be calculated (the position is determined by means of the constraint *Through*(p_7, l_4) and the slope by means of the constraint *Parallel*(l_3, l_4)). From the line l_4, the point p_6 and the line l_5 can be reached (see Fig. 5). New position of the point p_6 is calculated as the intersection between the lines l_4 and l_7. The line l_5 and the point p_5 are recalculated using the same procedure.

From this example it is obvious that constraint propagation through the whole ACDG graph has to be performed only in cases where the parameters in the root ACDG node are changed. Let us see the required time complexity of constraint propagation in such a case. Suppose the cardinality of a set **N** is n. Let us say that each ACDG node has m children $0 \leq m \leq n$. From the root node, m nodes can be developed and they are stored in a queue F_1. For this task time $T_d(m) = k_d m$ is needed, where k_d is the time needed to develop one ACDG node. With the help of the triggering table, the first node which can be fired is determined — let us denote it as i ($i \in [1, m]$). This can be done in time $T_f = k_f m$, where k_f is an average time needed for checking if an ACDG node can be fired. To fire the node i, the time T_1 is needed which is determined as:

$$T_1(m) = k_d m + k_f m.$$

In the next step, a node j from the queue F_1 is tried to be fired ($j \in ([1, m] - i)$). In the worst case all m-1 remaining nodes from the queue F_1 are tried to be fired unsuccessfully. After that children of the node i (i has again m children) are developed and they are stored in the queue F_2. This task can be performed again in the time T_d. In the queue F_2 a node, which can be fired, surely exists. It can be found again in the time T_f. The worst possible time which is needed to fire the second ACDG node is T_2:

$$T_2(m) = k_d m + k_f (m - 1) + k_f m = k_d m + k_f (2m - 1).$$

The time needed to develop the last n-th ACDG node is then:

$T_n(m) = k_d m + k_f\ (nm - (n - 1)) + k_f m = k_d m + k_f (nm - n + 1)$.

The time needed to fire all ACDG nodes can be easily obtained as:

$$\begin{aligned} T(m) &= T_1 + T_2 + T_3 + \ldots + T_n \\ &= (k_d m + k_f m) + (k_d m + k_f (2m - 1)) \\ &\quad + (k_d m + k_f (3m - 2)) + \ldots \\ &\quad + (k_d m + k_f (nm - n + 1)) \\ &= n k_d m + k_f (m + 2m + \ldots + nm \\ &\quad - (1 + 2 + 3 + \ldots + (n - 1))) \\ &= n k_d m + \tfrac{1}{2} k_f (mn(n + 1) - n(n - 1)) \\ &= n(k_d m + \tfrac{1}{2} k_f (m(n + 1) - (n - 1)). \end{aligned}$$

However, if n is large enough, many ACDG nodes are without children. In this case $m = 0$, to obtain an average time needed to solve all constraints, we can assert $m \ll n$ and m can be observed as a constant. Consider k_m is the average number of children of an ACDG node. The average time complexity is then:

$$T_A = n(k_d k_m + \tfrac{1}{2} k_f (k_m(n+1) - (n-1))) = O(n^2).$$

5. Combining geometrical objects represented by ACDG's

In geometric modelling, users usually combine already-made geometrical objects to obtain more complicated geometrical shapes. For this purpose in many cases Boolean regularised operators are offered to the users. In spite of certain disadvantages they have been widely accepted as the operators for simulating manufacturing operations; e.g. removing or adding portions of material. Quick and robust algorithms for regularised Boolean operations have been proposed even in B-rep. Nowadays feature-based operators have been extensively introduced in geometric modelling (J. R. ROSSIGNAC 1990). They capture the whole richness of engineering vocabulary and represent the most natural way for integrating CAD, CAPP, and CAM. Using either more primitive Boolean operators or high-level feature-based operators, users still have to do a lot of work without the support of a geometric modelling system. Namely, they have to place geometrical objects at the required positions manually and then perform the desired operations. In a 2D world, the users have to maintain 3 degrees of freedom and in 3D space, 6 degrees of freedom.

These difficulties appear because expressing spatial relationships in actual geometric modelling systems is not supported. As was mentioned in the introduction, expressions like *the hole is in the centre of the slot* or *the slot is in the top face of the object* are not supported. Previous sections have explained how geometric constraints can be used to establish spatial relationships among geometrical elements representing a geometric object. In this section, it will be shown how geometrical constraints can be used to express spatial relationships among geometrical objects, and the problems associated with this task will be highlighted.

5.1. Expressing Geometrical relationships among geometrical objects represented by Acyclic Constraint Description Graph

In Fig. 6a, a simple geometric object named A is shown. Suppose the user wants to place an object B (Fig. 6b) on the object A. We would like to obtain a new geometric object C (Fig. 7a) using the existing objects A and B. Consider the objects A and B are represented by appropriate ACDGs. Only the parameters of each ACDG are accessible, as shown schematically in Fig. 6a and 6b. To have complete control over the new object C, an appropriate ACDG representing it should be generated.

At this point we are interested mainly in the process of establishing appropriate space relations between geometrical objects A and B. Because of this, necessary changes in the topology of the new object C will not be observed. To put object B on the object A we require:

- p_{b1} lies **on** the line l_a,
- the **distance** between vertices p_{b1} and p_{a1} is c, and
- the lines l_b and l_a are **perpendicular**.

All expressions written in bold type are already known to us. They are exactly the constraints

1.	On(l_a, p_{b1})
2.	Distance(p_{a1}, p_{b1}, c, δ)
3.	Perpendicular(l_a, l_b)

Tab. 3. Predicates needed to express spatial relationship between objects A and B

introduced in section 3 and they can be easily transformed into corresponding predicates shown in Table 3. They are stored in the so-called "constraint interface" CI.

An ACDG the representing the object C can be easily obtained by connecting the ACDGs of the objects A and B by the predicates mentioned above. Fig. 7b shows established spatial relationships between the objects A and B. Constraints used for this task are stored in the constraint interface CI_1. A very clear hierarchical structure between the objects A and B has been obtained. If any parameter of the object A is changed, all necessary changes in the position of the object B are made automatically, as explained in section 5. The whole struc-

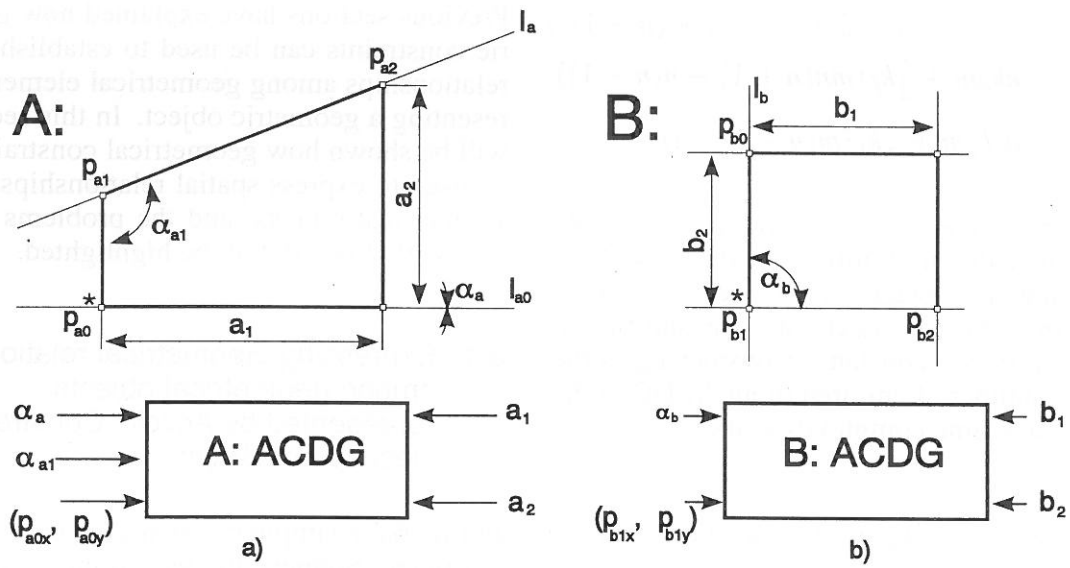

Fig. 6. Basic geometric objects A and B

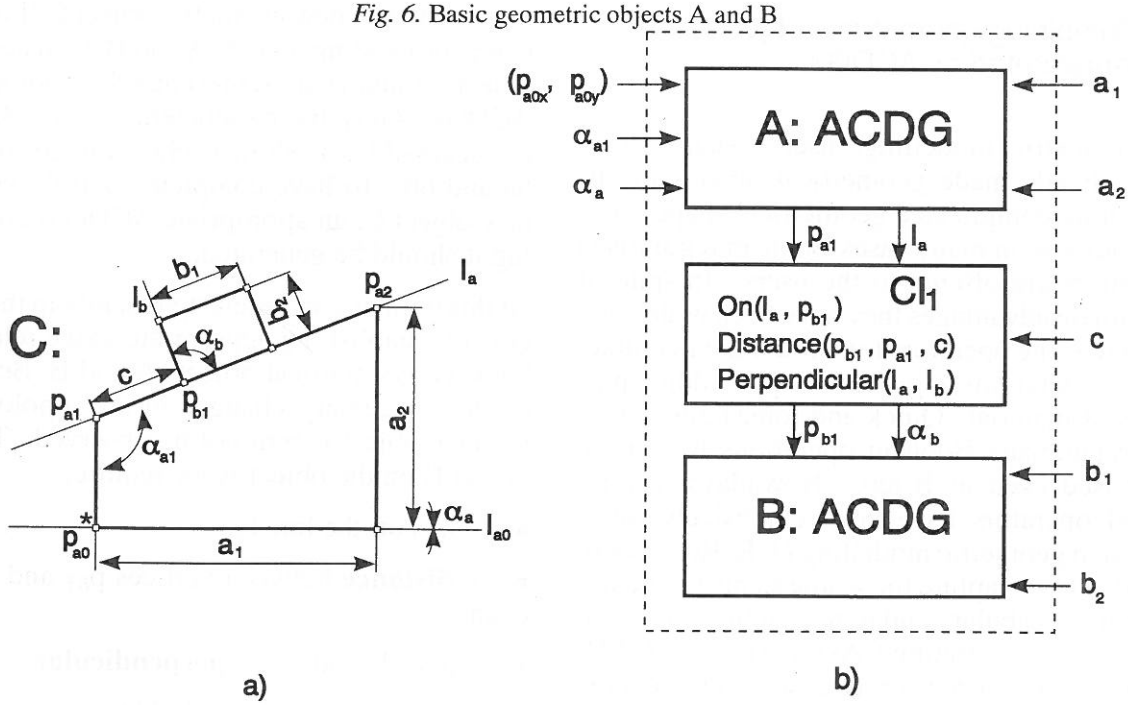

Fig. 7. a) Target geometrical object C b) Connecting geometrical objects A and B

ture can be observed as one common ACDG (it is schematically presented in Fig. 8) and one could completely eliminate the borders between its constituent parts. However, this is not practical for at least two reasons:

- individual parts may be supplemented by additional local information (such as colour, material, etc.) and
- removing individual parts from an object would no longer be trivial.

The case when all geometrical elements of hierarchically inferior geometrical objects, which are needed to establish spatial relationships among geometrical objects (in our case the point p_{b1} and the slope of the line l_b), are directly accessible by the user is called a "well-adopted graph". Serious problems occur in other cases, named "ill-adopted graphs".

5.2. Establishing spatial relationships in case of an ill-adopted graph

Reconsider establishment of the spatial relationship between geometrical objects A and B. This time the user does not have explicit access to the point p_{b1} and the slope of the line l_b. The position of the point p_{b1} is calculated from the point p_{b0} which is in this case, the root node of our ACDG. Similarly, the line l_b slope is obtained from the line l_{b0} (see Fig. 9a). Of course, to establish required spatial relationships between objects A and B, the same list of predicates is needed as in the previous case (see Table 3). To put the object B at the required position the following steps have to be performed:

1. Using the constraints from the constraint interface CI_1 (Table 3) and known values of the point p_{a1} and the line l_a determine the required values for the point p_{b1} and the line l_b.

2. Propagate constraints through ACDG of the object B using old (or even approximate) values for the point p_{b0} and the line l_{b0}. The values $l'_{b\alpha}$ and p'_{b1} are found — these values usually differ from required values obtained in the step 1. The calculated values $l'_{b\alpha}$ and p'_{b1} are used to correct the values of the point p_{b0} and the slope of the line l_{b0}. This is done by using the constraints stored in the con-

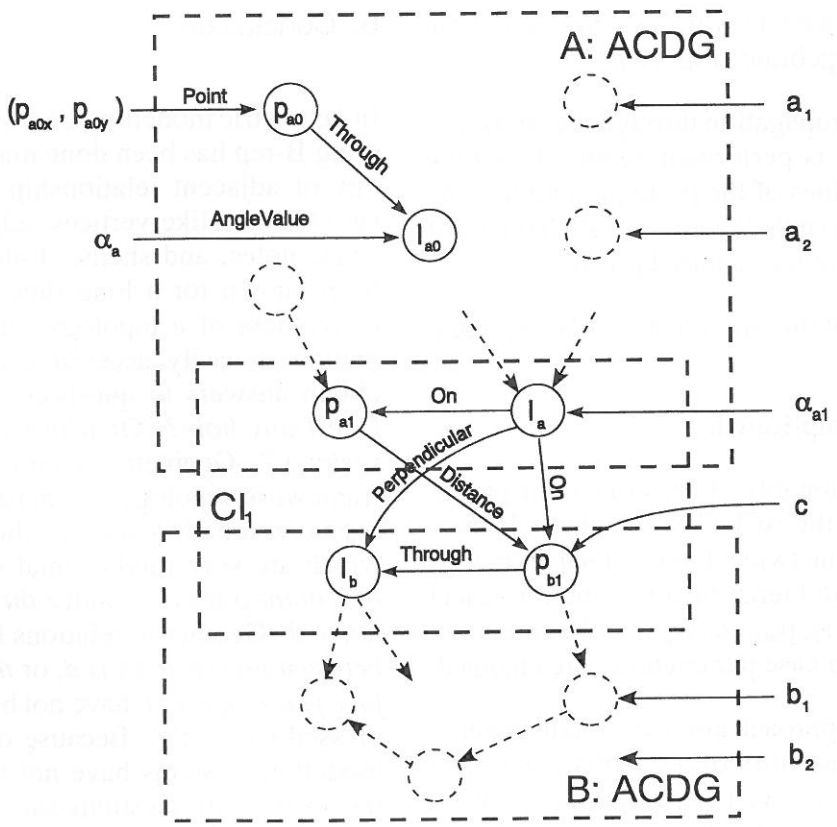

Fig. 8. ACDG of the object C

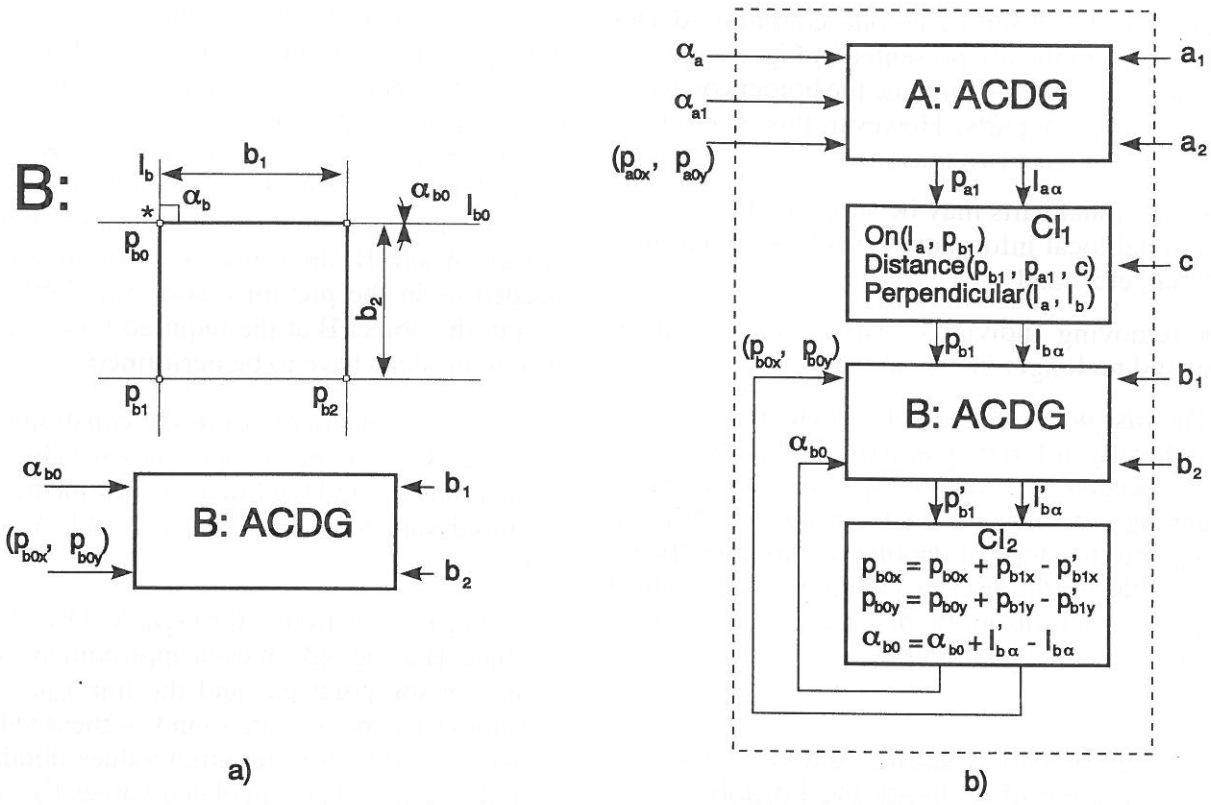

Fig. 9. a) Geometric object B with point p_{b0} as a root ACDG node b) ACDG of the object C in the case of ill-adopted graph

straint interface Cl_2 which are not geometric but purely algebraic (Fig. 9b).

3. Constraint propagation through the ACDG of the object B is performed again. This time corrected values of the point p_{b0} and the line l_{b0} are used to put all geometrical elements of the object B at the required position.

At the first sight this approach is not convenient because:

- it is fairly complicated,

- it is unquestionably slow; constraint propagation through the ACDG of the object B must always to be done twice if any of the parameters belonging to the hierarchically superior object (in our case a_1, a_2, p_{a0}, α_a, α_{a1}) or the constraint interface (in our case parameter c) are changed.

However, this approach also has certain benefits because the reusability of a geometrical object is completely preserved (B. ŽALIK et al 1993). Of course, our aim should be to combine geometrical objects with "well-adopted graphs".

6. Conclusion

In geometric modelling a lot of research considering B-rep has been done mainly on the topology of adjacent relationship among topological elements like vertices, edges, faces, loops, rings, holes, and shells. Euler operators have been known for a long time and these assure correctness of a topology. Topological information is easily accessible and the user can obtain answers to questions such as: *has the face f any hole?* Or *which edges ends in the vertex v?* Geometry is built directly into the framework topology has defined so far. It has been controlled by the so-called local operators which are very fundamental such as: *position of a point p is (x, y), move the edge e for a distance d.* Geometric relations like: *the distance between two vertices is d*, or *the face f_1 and the face f_2 are parallel*, have not been able to be expressed explicitly. Because of this, geometric modelling systems have not been able to give the answers to questions such as: *how large is the distance between points p_1 and p_2?* Or *are the faces f_1 and f_2 parallel?* without additional

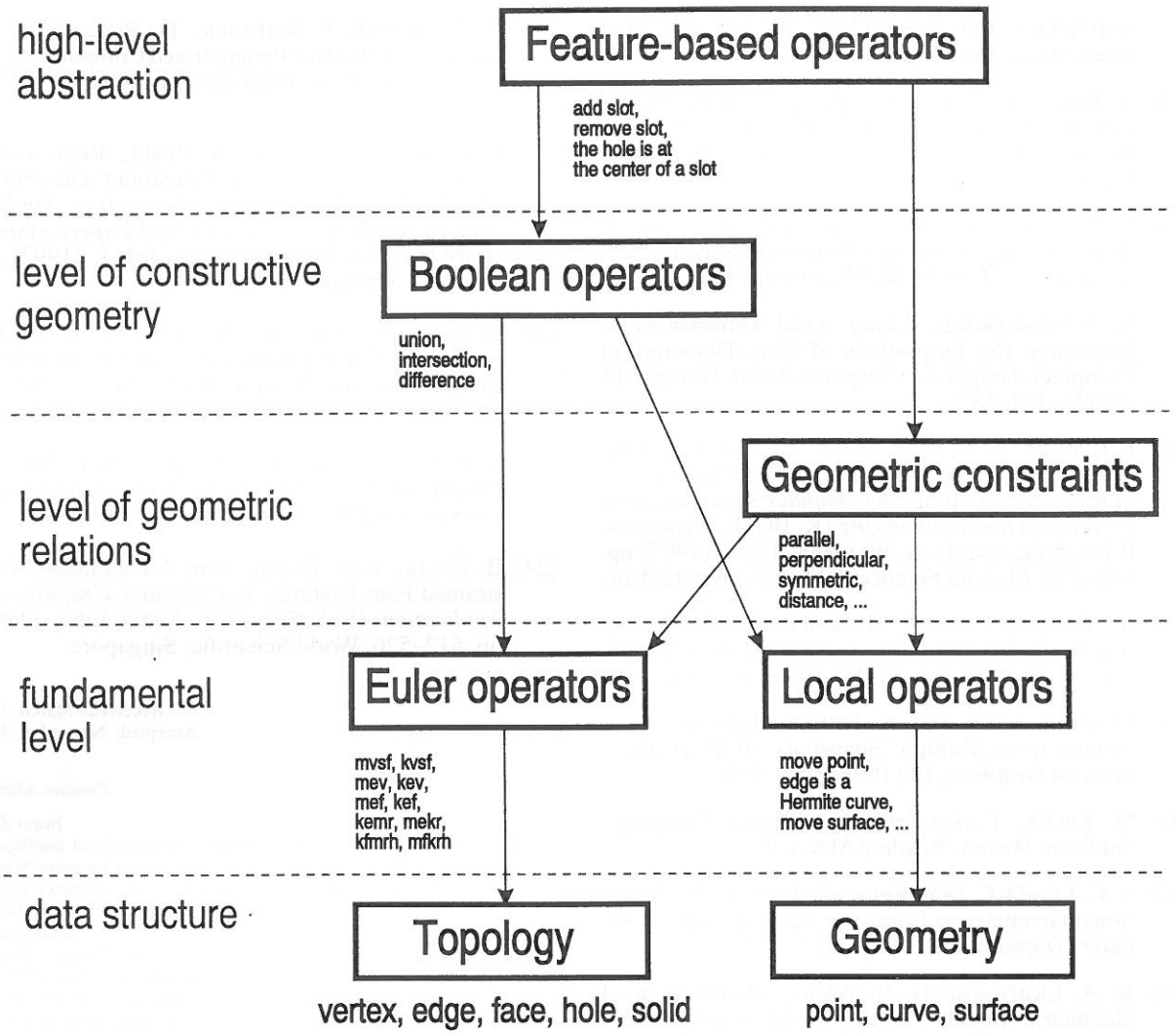

Fig. 10. Hierarchy of operators in geometric modelling

calculations. Because of the problem of finite arithmetic, inserted constraints have been lost many times. Fig. 10 shows the authors' view of the hierarchy of geometric modelling operators in B-rep. Geometric constraints can be observed as a missing level in describing geometrical objects. They are able to capture geometrical relations and offer the user operators which have been used in engineering for many years.

The presented constraint-based approach has been used as a base for a system for describing scaleable oulines of fonts (B. Žalik 1995).

Acknowledgements

The authors thank the referees for their valuable comments and suggestions for improvements.

References

[1] B. Aldefeld, Variation of Geometries Based on a Geometric-Reasoning Method. *Computer-Aided Design*, **20**, (1988) 117–126.

[2] B. Aldefeld, H. Malberg, H. Richter, and K. Voss, Rule-Based Variational Geometry in Computer-Aided Design. In *Artificial Intelligence in Design* (D. T. Pham, Ed.), (1991) pp. 27–46. Springer Verlag, Berlin.

[3] H. Ando, H. Suzuki, F. Kimura, A Geometric Reasoning System for Mechanical Product Design. In *Computer Applications in Production and Engineering* (F. Kimura and A. Rolstadås, Eds.), (1989) pp. 131–139. Elsevier Science Publishers, Amsterdam.

[4] C. A. Baykan and M. S. Fox Wright, A Constraint Based Spatial Layout System. In *Artifical Intelligence in Engineering Design – Design Representation and Models of Routine Design* (C. Tong

AND S. DUVVURU, Eds.), (1990) pp. 395–431. Academic Press, San Diego.

[5] A. BIJL, An Approach to Design Theory. In *Design theory for CAD* (H. YOSHIKAWA AND E. A. WARMAN, Eds.), (1987) pp. 3–25. Elsevier Science Publishers, Amsterdam.

[6] B. N. FREEMAN–BENSON, J. MALONEY AND A. BORNING, An Incremental Constraint Solver. *Communications of the ACM*, **33** (1990), 54–63.

[7] W. J. FITZGERALD, Using Axial Dimensions to Determine the Proportions of Line Drawings in Computer Graphics. *Computer-Aided Design*, **13** (1981), 377–382.

[8] F. KIMURA, H. SUZUKI, AND L. A. WINGARD, Uniform Approach to Dimensioning and Tolerancing in Product Modelling. In *Computer Applications in Production and Engineering* (K. BØ. L. ESTENSEN, P. FALSTER, AND E. A. WARMAN, Eds.), (1987) pp. 165–178. Elsevier Science Publishers, Amsterdam.

[9] K. KONDO, PIGMOD, Parametric and Interactive Geometric Modeller for Mechanical Design. *Computer-Aided Design*, **22** (1990), 633–644.

[10] D. KURLANDER AND S. FEINER, Inferring Constraints from Multiple Snapshots. *ACM Transactions on Graphics*, **12** (1993), 227–304.

[11] W. LELER, *Constraint Programming Language*. Addison–Wesley, Reading MA, 1988.

[12] V. C. LIN, D. C. GOSSARD, AND R. A. LIGHT, Variational Geometry in Computer Aided Design. *Computer Graphics*, **15**, (1981), 171–177.

[13] R. A. LIGHT AND D. GOSSARD, Modification of Geometric Models Through Variational Geometry. *Computer-Aided Design*, **14**, (1982), 209–214.

[14] G. NELSON, Juno, a Constraint-Based Graphics System. *Computer Graphics*, **19**, (1985), 235–243.

[15] J. R. ROSSIGNAC, P. BORREL, AND L. R. NACKMAN, Interactive Design with Sequences of Parametrized Transformations. In *Intelligent CAD Systems II* (V. AKMAN, P. J. W. TEN HAGEN, P. J. VEERKAMP, Eds.), (1989) pp. 93–125. Springer Verlag, Berlin.

[16] J. R. ROSSIGNAC, Issues on Feature-Based editing and interrogation of solid models. *Computer & Graphics*, **14** (1990), 149–172.

[17] M. SANNELLA, J. MALONEY, B. N. FREEMAN–BENSON, AND A. BORNING, Multi–Way Versus One–way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm. *Software-Practice and Experience*, **23** (1993), 529–566.

[18] G. SUNDE, CAD System with Declarative Specification of Shape. Presented at the *Eurographics Workshop on Intelligent CAD Systems*, (1987), Noordwijkerhout, The Netherlands.

[19] H. SUZUKI, H. ANDO, AND F. KIMURA, Geometric Constraints and Reasoning for Geometrical CAD Systems. *Computer & Graphics*, **14** (1990), 211–224.

[20] A. VERROUST, F. SCHONEK, D. ROLLER, Rule-Oriented Method for Parametrized Computer-Aided Design. *Computer-Aided Design*, **24** (1992), 531–540.

[21] B. ŽALIK, N. GUID, AND A. VESEL, Representing Geometric Objects Using Constraint Description Graphs. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (F. BELLI, F. J. RADERMACHER, Eds.), (1992) pp. 505–514. Springer Verlag, Berlin.

[22] B. ŽALIK, N. GUID, AND A. VESEL FLEXI, An Experimental Constraint-Based Modeling System. In *CG International '92* (T. L. KUNII, Ed.), (1992) pp. 697–710. Springer Verlag, Tokyo.

[23] B. ŽALIK, N. GUID, AND A. VESEL, Reusability of Parametrized Geometric Objects. *Programming and Computer Software*, **19** (1993), 165–176.

[24] B. ŽALIK, Font Design with Incompletely Constrained Font Features. In *Computer Graphics and Applications* (S. Y. SHIN, T. L. KUNII, Eds.), (1995) pp. 512–526. World Scientific, Singapore.

Received: March, 1995
Accepted: November, 1995

Contact address:

Borut Žalik
Laboratory for Computer Graphics and Artificial Intelligence
Department of Computer Science
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova 17
SI-2000 Maribor, Slovenia
phone: ++ 386 62 25–461
fax: ++ 386 62 225–013
e-mail: zalik@uni-mb.si

BORUT ŽALIK is an Assistant professor of Computer Science at the Univeristy of Maribor. Currently, he is a head of a Centre for Geometric Modelling at the Faculty of Electrical Engineering and Computer Science Maribor and a member of a Laboratory for Computer Graphics and Artificial Intelligence. His research interests include computer graphics, geometric modelling, CAD, computational geometry, multimedia, and computer pheripheral devices. Žalik received a Ph.D. in Computer Science from the University of Maribor, Slovenia in 1993.

NIKOLA GUID is presently a Full Professor at the Faculty of Electrical Engineering and Computer Science in the Univeristy of Maribor, Slovenia, and a head of Laboratory for Computer Graphics and Artificial Intelligence. His current research interests are computer graphics, computer aided geometric design, geometric modelling, computer simulation, search strategies, and knowledge representation. He wrote two textbooks on computer graphics. Guid received both a B.Sc. and M.Sc. in Electrical Engineering from the University of Ljubljana in 1974 and 1977, respectively, while a Ph.D. from the University of Maribor in 1984. He is a member of the IEEE and Eurographics.