An Interactive Constraint-Based Graphics System with Partially Constrained Form-Features

Borut Zalik
University of Maribor, Faculty of Electrical Engineering and Computer Science,
Smetanova 17, SI-2000 Maribor, Slovenia.
Phone: +386-62-25-461 – Fax: +386-62-225-013
E-mail: zalik@uni-mb.si
WWW: http://www.uni-mb.si/~uelgng03f/index.html

Abstract

The paper considers a 2D constraint-based geometric modelling system which distinguishes between an auxiliary and a visual geometry. The former consists of points, lines, and circles, and the later includes line segments, arcs, circles and cubic Bézier curve segments. Geometric constraints are applied only upon the auxiliary geometry. Some constraints and the majority of auxiliary geometry are extracted by the system automatically what liberates the user from giving self-understandable facts. This separation of the geometry enables developing a simple but efficient approach to constrain the cubic Bézier curves. The system can work with under-constrained parts of geometric objects what makes it interactive entirely. The paper includes a practical example of the design in the constraint-based environment and gives an insight into a user interface used..

Keywords

Geometric constraints, interactive constraining process, constraint solving, user interface.

Introduction

In the paper, a 2D constraint-based geometric modelling system is described. In such a system, it has been required frequently that exactly all needed constraints are inserted before constraint solver is made active. Two main drawbacks can be identified quickly within such an approach. First, the inability to support an interactive work and second, a requirement for specifying a correct set of constraints at once. The presented approach offers a new, more flexible way of inserting constraints and enables the user to follow their effects. The system automatically extracts self-evident constraints and in this way the number of constraints which have to be inserted manually by the user is reduced.

Geometric modelling systems usually do not support the work with so called auxiliary geometry. The user can only use guiding lines or grids, but the rest of the aux-

iliary geometry cannot be represented and stored. Because of this, the huge number of construction techniques which have been developed by engineers cannot be used. Our system efficiently supports auxiliary geometry which is needed during the construction, and stores it together with so called visible geometry.

1 Background

In geometric modelling, geometric entities such as line segments, circles, arcs, and splines describe the shape of designed artefacts. Because of the speed, reliability and offered construction possibilities, computer programs for drawing have already become unavoidable tools of a design environment. To further increase the designer's productivity, a reusability of already constructed parts of artefacts is a new desired property [Zalik93].

Today, the geometric entities which logically fit together can be grouped, uniquely named, and independently stored to be reused again in different situations. In general, the grouping is done in a very simple way just by selection of all elements of a group. Unfortunately, the reusability of such defined groups of geometric entities is very limited. Users can perform only basic geometric transformations (translation, rotation, and scaling), but they cannot change the key geometric parameters of the group, like distances or angles (figure 1).

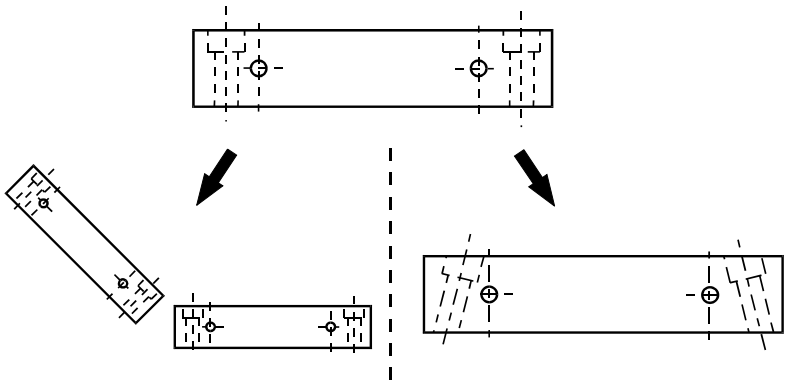

Figure 1. Instances generated by geometric transformations (on the left) and from the parametric representation of the generic object (on the right)

The reason for this is the primitive way of describing the groups of geometric entities. To further increase the reusability of designed geometric objects, a corresponding parametric representation has to be performed. If a mechanism for combining the groups of parametricaly represented geometric entities is available, then the meaningful collection of geometric and topological entities is named form- or geometric-feature [Rossignac90]. The parametrisation of the geometric features can be achieved in different ways [McMahon92]. One of the possibilities is to introduce geometric constraints. Groups of geometric entities, which beside topological and

geometrical information also include constraints, show much higher level of reusability.

1.1 Constraints

A constraint describes a relation that should be satisfied [Freeman-Benson90]. Examples of constraints in geometric modelling are: line is vertical, point lies in the middle of two other points, two parallel planes are at distance d, two faces are perpendicular, etc.

Constraints are usually expressed in a declarative way by predicates. Following Aldefeld's division, the predicates defining constraints in our system are divided in three main groups [Aldefeld91]:

- Structural constraints (see table 1) determine spatial relationships between geometric entities which will not change. As will be shown in the rest of the paper, some of them can be determined by the system automatically.

Predicate	Meaning - effect
HLine (l_i)	line l_i is horizontal
VLine (l_i)	line l_i is vertical
Through (l_i, p_j)	line l_i passes through point p_j
On (p_i, l_j)	point p_i lies on line l_j
Perpendicular (li, lj)	lines li and lj are perpendicular
Parallel (l_i, l_j)	lines l_i and l_j are parallel
Middle (p_i, p_j, p_k)	point p_j is in the middle of points p_j and p_k
Symmetric (l_i/p_i, l_j, l_k/p_k)	lines l_i and l_k (or points p_i and p_k) are symmetric regarding the reference line l_j
Coincidence (l_i/p_i, l_j/p_j)	lines l_i and l_j (or points p_i and p_j) coincide

Table 1. Predicates of structural constraints

- Dimensional constraints determine positions, distances, coordinates, and angles. They include variables which are parameters of geometric objects. Examples of dimensional constraints implemented in our system are shown in table 2.

Predicate	Meaning - effect
Point (p_i, x, y)	point p_i gets absolute coordinates (x, y)
AngleValue (l_i, α)	line l_i gets absolute value of its slope
Distance (p_i, p_j, d)	distance between points p_i and p_j is d
Distance (l_i, l_j, d)	distance between parallel lines l_i and l_j is d
Angle (l_i, l_j, α)	angle between lines l_i and l_j is α

Table 2. Predicates of dimensional constraints

- Predicates of numerical constraints provide a mechanism for connecting parameters of dimensional constraints. Examples of these are depicted in table 3.

Predicate	Meaning - effect
Product(a, b, c)	$c = a * b;\ a = c * b^{-1};\ b = c * a^{-1};$
Sum(a, b, c)	c = a + b; a = c - b; b = c - a;
SameValue(a, b)	a = b; b = a;

Table 3. Predicates of numerical constraints

1.2 Problems at Constraint Description

The declarative approach of expressing geometry and geometric relations (constraints) should be closer to the human way of thinking than the procedural approach. Unfortunately, it turns out that a lot of serious problems are associated with such an approach. For example:

- The user is required to insert exact number of constraints. We distinguish among following cases [Fudos93]:
  - geometric object is well-constrained if it has a finite number of solutions;
  - geometric object is under-constrained if it has an infinitive number of solutions;
  - geometric object is over-constrained if it has no solutions.

  This requirement is very hard and therefore different authors have already suggested different approaches how to avoid it. Borning and his group introduced the hierarchy of constraints [Borning87], Ando et al. suggested to use default constraints [Ando89], Hel-Or introduced probabilistic constraints [Hel-Or94].
- In real applications a huge number of constraints have to be specified. The manual inserting of constraints is an awkward, tedious, and error-prone work. Therefore, an automatic constraining procedure based on multiply snapshots has been proposed by Kurlander [Kurlander93]. However, this approach requires the object of interest to be already constructed.
- Ordinarily, current constraint-based geometric modellers include the basic geometric entities such as points, lines, and circles. For the majority of them the complexity of free-form entities (as Bézier cubics or NURBS) seems to be too difficult. However, very recently Fudos and Hoffmann integrated successfully parametric conics into a constrained-based environment [Fudos96].

1.3 Form-Features

In geometric modelling, there are many definitions of what the word form-feature means (see for example [Falcidieno89, Rossignac90]). It depends on an application how to choose, define and name form-features. We choose fonts (in fact the font outlines) as the geometric objects of interest and they will be used in the continuation. If the characters of a certain font family (like Times Roman) are observed, a lot of identical parts can be noticed quickly (see figure 2). For Times Roman font family, a stem, a bar, a slant, a bow, and a serif have been suggested [Karow90].

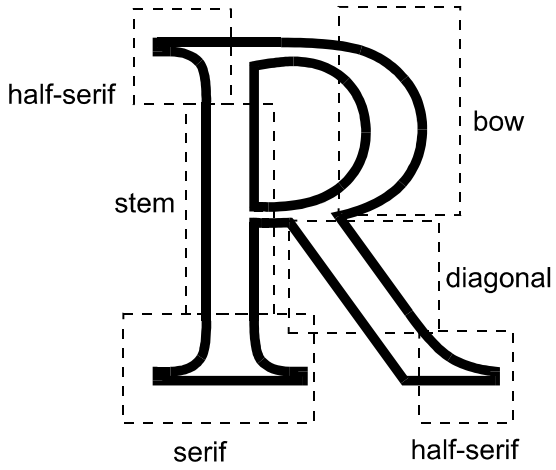

Figure 2. Character outlines consist from font-features

Although everything seems perfect with the fonts inside modern computerised environment, some problems still exist. Today's applications require a huge number of different fonts of different sizes and orientations. This requires a vector representation of individual characters upon which required geometric transformations can be applied easily.

Unfortunately, the majority of output devices use raster data, and work within the finite resolution. Because of this, the vector representation of fonts has to be rasterised quickly and accurately before any character is displayed. However, due the rounding errors caused by the finite arithmetic of digital computers, the result of rasterising algorithms can lead to unpleasant visual effects [Karow90]. To reduce them, the vector representation of font outlines is equipped by an additional information how to perform the rasterisation also visually correct. This information is usually called a hint or an instruction. If the hints are associated with individual form-features, then specifying them for the whole character set is easier and quicker. This is particularly important in large character sets (for example Kanji [Duerst93]).

Herz and Hersch developed an auto-hinting system for typographic shapes [Herz-94]. They limited their discussion on stems and bars, that is on form-features, where only line segments are presented. The procedure is based on successive steps beginning with extraction of straight line segments within prescribed deviation, their merging, classification, and at least production of the hints. However, we feel that in our system the generation of hints can be done much easier by using the existed information about topology, geometry, and described constraints. The introduced constraints directly carry the information which is needed for the visual correct rasterisation process as for example: a line is vertical, or two Bézier curves are symmetric regarding a reference line.

However, the focus of the paper is entirely on the interactive techniques employed to constrain the form-features and to achieve an appropriate level of their reusability, and does not touch the font design problems.

2 Constraint Solving

The heart of any constraint-based geometric modelling system is a constraint-solving engine which has to solve given constraints without help of the designer. There exist different approaches for constraint solving. Our system uses the local propagation of known states. Although this method has some remarkable drawbacks (it cannot solve cyclic constraints), its ability to support interactive design was a good reason for its selection. A geometric example where the local propagation of known states fails is given in [Zalik95a].

To represent the local propagation of known states a graph is used in the most cases [Leler88]. The local states propagate through arcs of the graph. When a graph node gets enough local information to be solved, it fires and offers its data (the result of solving of the local constraints) to the neighbouring graph nodes. These nodes then check, if they obtained enough information to be fired. The process works as a chain-like reaction while there are any graph nodes to be solved. The solver stops and waits for a new constraint to be inserted or informs the user that the considered geometric object is completely constrained already.

For the local propagation of known states in geometric environment, a special data structure, called biconnected constraint description graph has been developed by Zalik [Zalik95a]. To enable a fast constraint solving, the scope of supported types of geometric entities is usually limited. As it has been already mentioned in the introduction, geometric entities in our system have been divided into two groups:

- The auxiliary geometry consists of three types of geometric entities: points, lines, and circles. The constraints refer only to the entities of this group. In this way the constraint solver is not burdened with too many different types of geometric entities and its implementation is easier. This, however, does not limit the power of the constraint solver significantly, because the majority of engineering constructions involve only lines and circles. Of course, the auxiliary geometry should be switched off when the final shape of a geometric object has to be displayed.
- Upon the auxiliary geometry, the visible geometry is built. It consists of line segments, arcs, and cubic Bézier curves. If the auxiliary geometry is constrained, then the visible geometry is constrained, too (vice versa is not necessary true). Each entity of the visible geometry has its topological counterpart, while the geometric entity of auxiliary geometry has not.

In figure 3a, line l which is a member of the auxiliary geometry, carries two line segments (ls_1 and ls_2). If, for example, the slope of line l is changed, the slopes of

both line segments have to be changed, too. In this way, the constraints which force both line segments to lie on line l are satisfied.

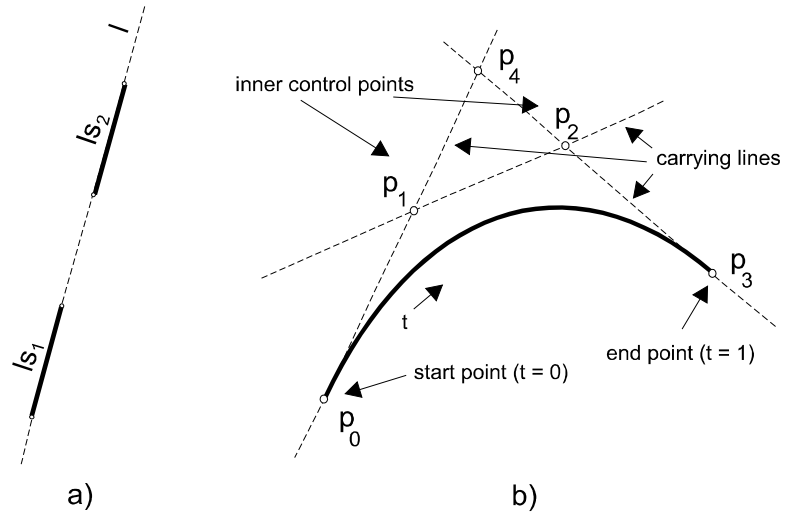

Figure 3. Two examples of the visible and the auxiliary geometry a) two line segments carried by a line b) Bézier cubic and associated auxiliary geometry

In a similar way, the constraining of the Bézier cubic is done. The approach is very simple but efficient. Instead of constraining the curve directly, we constrain its control points which are treated as entities of the auxiliary geometry.

Constraining of the anchor Bézier control points (p_0 and p_3 in figure 3b) is obvious. It is the same as constraining the end points of the line segment. For the inner Bézier control points the following procedure is applied:

a) At the Bézier anchor points, tangent vectors are calculated and they are placed on carrying lines passing through the anchor control points (see figure 3b). This implies that the control points lie on these lines.
b) Distances between respective anchor and inner control points (figure 3b) are then determined.
c) An intersection point between carrying lines is calculated (denoted as point p_4 in figure 3b).
d) The distances between the anchor points and the intersection point p_4 are calculated.
e) The rates of the distances (denoted qd_1 and qd_2) are calculated and stored beside the constraining scheme. The rates are evaluated as follows:

$$qd_1 = \frac{|p_0, p_4|}{|p_0, p_1|} \qquad \text{and} \qquad qd_2 = \frac{|p_3, p_4|}{|p_2, p_3|}.$$

To aid interaction, the user has the opportunity to change the position of the Bézier control points, and hence the curve shape. The user can move the inner con-

trol points along the carrying lines. This causes the rates of qd_1 and qd_2 are changed accordingly.

2.1 Interactive Constraining Process

The serif as one of the form-features shown in figure 2 are used to highlight the facilities of our constraint-based geometric modelling system. The serif is described probably by an untrained user as a feature having three line segments and two curves (suppose they are Bézier cubic). According to our previous discussion, these entities determine the visible geometry. This description, however, can be treated as being very desultory because it does not include any relations between used geometric entities.

Since users do not think about spatial relations between geometric entities at this stage of design, a good system should not force users to express these relations unless this is necessary (and therefore more natural). In our approach, the actions of the designer are just observed by the system and it generates automatically all self-understandable spatial relations - constraints. Of course, it is a question which constraints are self-understandable and always expected by the user. In our system, an automatic extraction of constraints which would be determined on the base of "a small number ε" is avoided. Beside a question how small the ε should be, it could happen easily that the system "becomes to clever" and just confuses the user.

A typical scenario would be a line drawn (almost) horizontal. It would be easy to detect such case and to constrain the line within the tolerance ε with the HLine constraint. However, we cannot be sure that the user really wants this constraint.

Perhaps he/she needs a line with a very small slope indeed or intents to establish the Parallel relation with some other line which is not horizontal. It could happen that the user would even like to vary the slope of this line. In such cases, the user should abandon the HLine constraint manually. To abandon a constraint which he/she does not inserted explicitly may only confuse the user. Because of these reasons, automatic detection of constraints based on the "small number ε" are excluded in our implementation. Instead, in such cases it is required that the user explicitly expresses his/her design intent.

On the other hand, the constraints which are always desired are the connectivity constraints represented by the predicates On and Through (see table 1). Let us consider a simple example. If the user draws a circular arc as an element of the visible geometry, there are the following self-evident facts:

- the arc lies on the circle which becomes a member of the auxiliary geometry;
- the arc is bounded with two end points (members of auxiliary geometry), which lie on the circle (two predicates On are determined automatically);
- through the centre of the circle and through both end points of the arc pass auxiliary lines and the angle between them determines which part of the auxiliary circle is part of the arc. Thus, beside two auxiliary lines the constraints con-

necting line segments with arc end points and the centre of the circle are generated automatically, too.

All these facts are generated by the system without a fear that some of them could be redundant or wrong. To constrain the arc completely, the user must constrain the circle (its centre point and the radius) and the end points of the arc. This, however, can be done in different ways depending on construction.

It has been just noticed that the same geometric object can be well-constrained in different ways. Of course, a question appears which constraining scheme to employ. In the case of constraining of form-features, this depends on the way, how individual form-features are bound together. In the final object (a character in our case), the constraints should propagate from one form-feature to another. At first sight it seems there exist two possibilities to solve this problem:

- more than one constraining schemes of the same form-feature are prepared and the user then chooses the most suitable one;
- only one constraining scheme exists and by using a transformation algorithm one can derive a desired constraint description.

Both solutions are difficult to realised in practice. The first one could increase enormously the number of prepared form-features and it would be difficult to find the appropriate one. The problem of the second proposal is that universal and reliable transformation algorithms do not exist. They depend on the internal representation of the form-features, an implementation of the constraint solver, and they have to be guided by the user in many cases [Zalik93].

Therefore, we employ a new strategy: The form-features need not be constrained completely. Instead, the majority of structural and just some geometric constraints are included in such a constraining scheme. The complete constraining of the form-feature can be done during the process of combining of individual form-features. In this way the constraining process is more natural and therefore less demanding.

3 Demonstration of our system

Let us demonstrate how the serif in our system is generated. The construction begins, for example, with drawing the bottom line segment. The first point of the line segment is marked by clicking the mouse, the mouse is then moved to the end point of the line segment and clicked again.

When the line segment is drawn, the program automatically creates required auxiliary geometric entities. These are: the start and the end points, and a line on which the drawn line segment lies. In addition, the system creates automatically a set of self-understandable constraints which define the spatial relationships between the line segment and the carrying line, and adds them to the description of the form-feature.

Figure 4 shows the situation after adding two line segments. The main window contains a menu, two toolbars, and sub-windows. The vertical toolbar stores drawing tools while the horizontal toolbar is used for a file and window management. The system offers the following sub-windows:

- The drawing sub-window is the main interactive working area of the system. At the moment, it contains two connected line segments with associated auxiliary lines. The user can decide whether to see the generated auxiliary geometry during the drawing or not. By default the auxiliary geometry is turned-off. The drawing window is the only one which is opened automatically; all other sub-windows are opened only on the user's request.

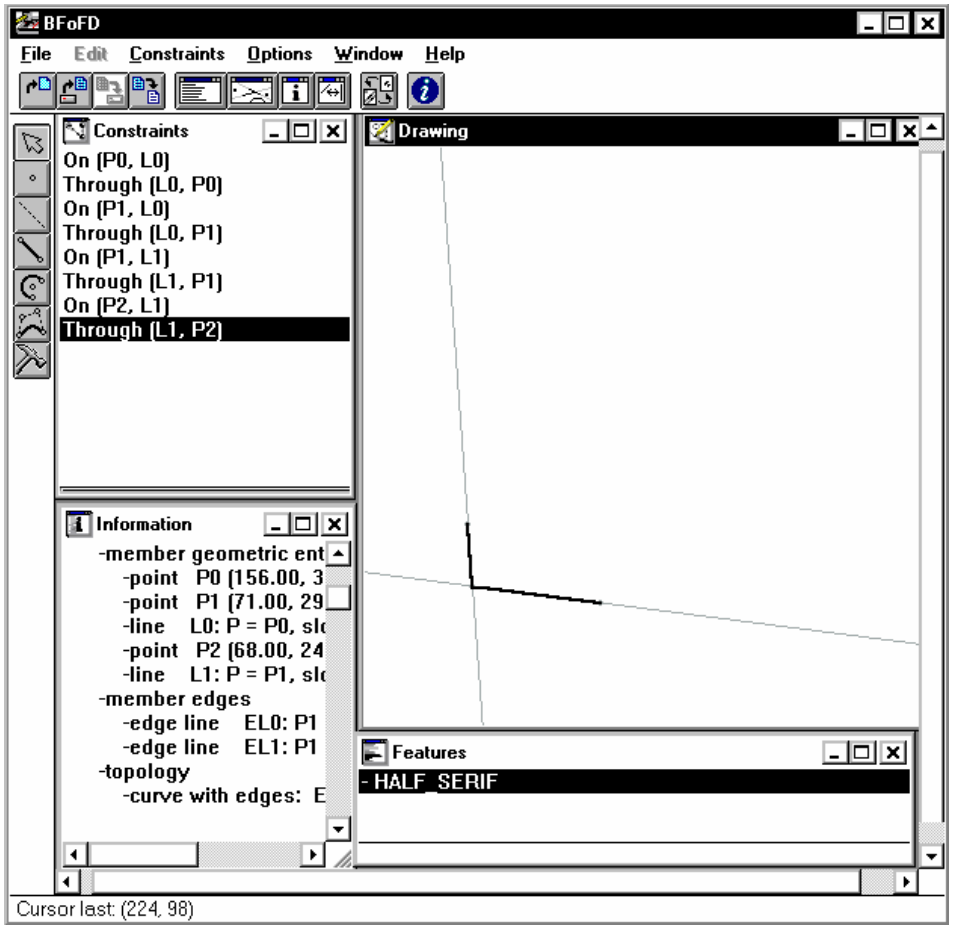

Figure 4. Beginning of the construction of the serif

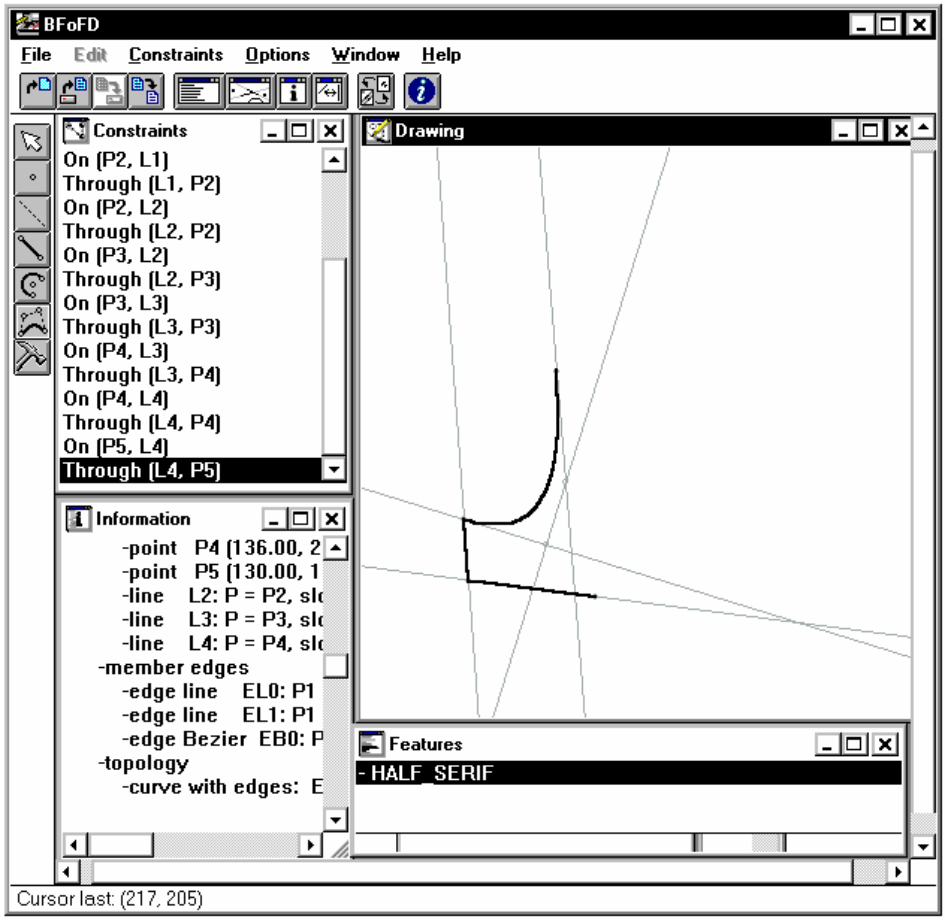

Figure 5. The sketch of the half-serif

- The information sub-window is on the left side of the screen. It displays the geometric and topological information of inserted geometry in a textual form. The description begins with the auxiliary geometric entities. Five elements of the auxiliary geometry are presented: the start point, the common point, the end point of both line segments, and the lines on which the line segments lie.

  Points are defined with coordinates and the lines with points they pass through and their slopes. All these values have initial approximate values which will be fixed by adding more constraints. These approximate values are used until the geometric entities are not constrained and they have also an active role during the constraint solving process [Zalik95a]. Each geometric entity in the information sub-window has a unique name within the form-feature (for example point P0). Next group are the elements of the visible geometry. At the moment it includes two line segments, both determined by two points. The last group contains the topological information.

- The constraint sub-window on the top left side of figure 4 lists all constraints which have been determined automatically by the system. The constraints are described with predicates and a list of associated parameters. The corresponding biconnected constrained description graph has been generated, too, but it is not shown to the user. For a simple identification of constraints a visual link between the constraint sub-window and the drawing sub-window should be established.
  For example, if the user would pick at a constraint in the constraint sub-windows this constraint would be displayed in the drawing sub-window. For this purpose a set of visual symbols for each type of constraints would be introduced. However, at the moment of writing this feature has not been implemented in the system, yet.

Figure 5 shows the situation after a Bézier cubic segment has been inserted[1]. Again, the needed auxiliary geometry has been added automatically (compare it with figure 3), structural constraints have been extracted, the topology has been changed, and both windows with textual information have been updated.

Till now, the user has not been aware that he/she has worked inside the constraint-based environment because the system has behaved as a classical drawing program. Because the initial sketch does not look "nice", the user can correct it easily by manually giving a few constraints. This is done in a very simple and natural way. The user opens the new window (figure 6) where he/she chooses a desired constraint and then simply picks required geometric entities.

For example, if we would like to tell the system, that lines l_i and l_j are parallel, then we must choose the predicate Parallel and pick both lines by the mouse. This new fact is accepted immediately by the system and it tries to solve the given constraint. Therefore, the biconnected constraint description graph is extended by the new information and then the constraint solver is activated. The solver must be prepared to handle the following three cases:

1. The slope of one of the picked lines is already determined (constrained) and the slope of the second is not. Of course, the slope of the second line is changed and becomes the same as the slope of the first line.
2. None of the involved line segments have the slopes already determined (constrained); they both contain only approximate data. In this case, the slope of the second picked line is set to be equal to the slope of the first picked line. In this way the system efficiently uses the approximate data and modifies them progressively according to the given constraints.
3. The slopes of both lines are already constrained. A message is generated to the user because an over-constrained case occurred. A list of involved constraints is

[1] The inner Bézier control points are not visible well because of reduction of the taken picture. They can be seen better in figure 6.

shown to him/her. If the user still wants to establish this relation then he/ she must abandon one of the constraints which constrain the slopes of the lines.

In the first two cases, the result - the lines become parallel - is shown to the user immediately. The system performs the propagation of constraints from those bi-connected graph nodes which have been changed. In this way the whole shape is recalculated and updated regarding the new facts. It is easy to show that the worse time complexity of the constraint propagation is $O(N^2)$ where N is the number of involved geometric entities of auxiliary geometry [Zalik95a] and this means that it can be done quickly enough.

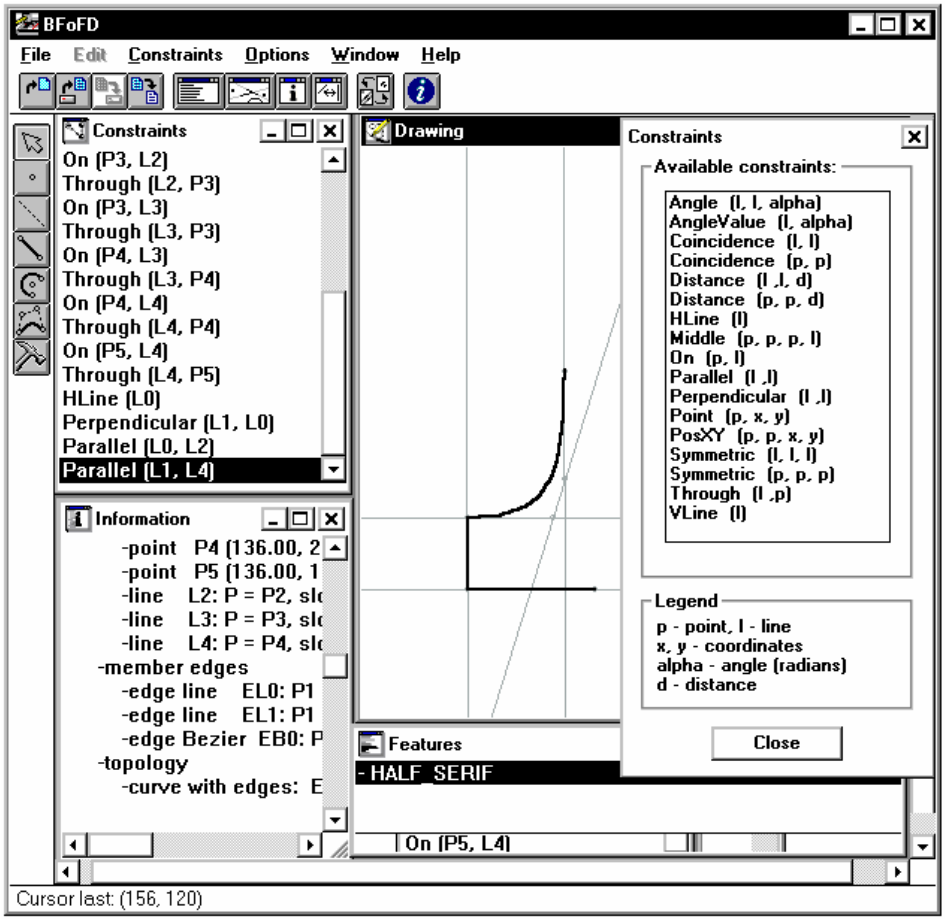

Figure 6. The situation after adding manually the last four constraints

To obtain the situation shown in figure 6, four constraints have been inserted:

- At first, the bottom line of the half-serif is said to be horizontal (constraint HLine(l_0) in figure 6).

- The perpendicular relation between the bottom half-serif line and the left half-serif line is established (constraint Perpendicular(l_0, l_1)).
- After that, the Bézier auxiliary lines are aligned to be parallel with lines l_0 and l_1 (constraints Parallel(l_0, l_1) and Parallel(l_1, l_4)).

Till now, we have not given any metric constraint and because of this, of course, our object is under-constrained.

However, we have been able to work with it entirely interactive and we could follow the effect of given constraints. The only difference between well-constrained and under-constrained parts of a geometric object visible to the user is the colour of geometric entities. The geometric entities drawn in green are already constrained and those drawn in black are not.

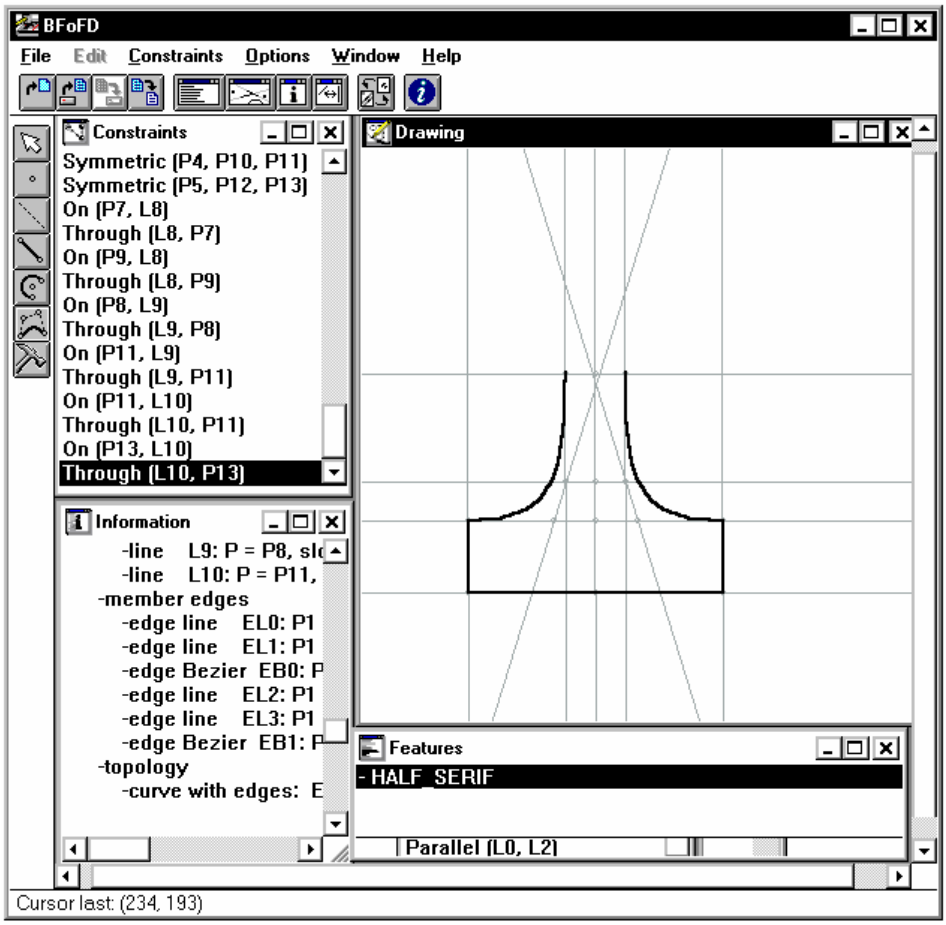

Figure 7. Generated serif and its auxiliary geometry

Now we have two possibilities how to continue with the generation of the serif:

1. The obtained half-serif can be stored into a library. Then we can generate two instances of the stored feature, and glue them into a new form-feature - a serif. The procedure how to perform this can be found in [Zalik95b].
2. We simply continue with the design to generate the whole serif. For the serif, the symmetrical property can be easily identified. Therefore, a line of symmetry is added as shown in figure 7. With a few additional operations, all performed by the mouse, the resulting shape is obtained quickly and easily.

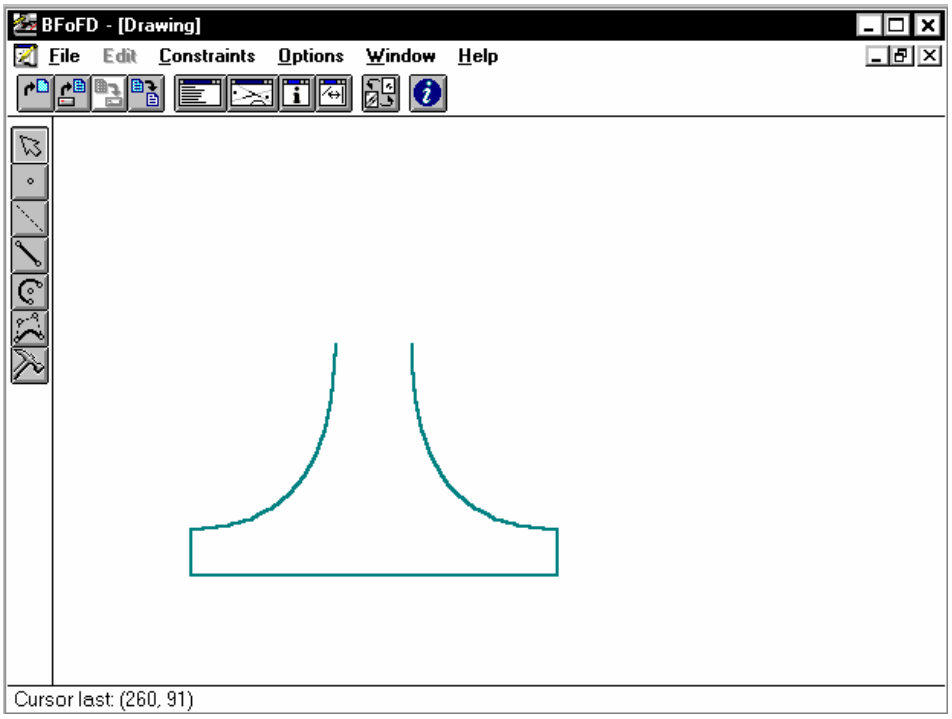

Figure 8. In instance of the serif

Let us suppose now, the user would like to generate an instance of described serif. Depending on how the serif is used, different parameters are needed and they are provided by the use of the metric constraints. The instance of the serif shown in figure 8 (the auxiliary geometry has been switched off) is obtained by giving the following metric constraints:

- the exact position of the left bottom serif point (predicate Point(point shown by mouse, 90, 300));
- the length of the horizontal line (predicate Distance(point shown by mouse, point shown by mouse, 240));
- the distance between end points of both Bézier cubics (predicate Distance(line shown by mouse, line shown by mouse, 50));

- the height of the whole serif (predicate Distance(line shown by mouse, line shown by mouse, 150);
- the height of the vertical line segment (predicate Distance(point shown by mouse, point shown by mouse, 30).

Conclusion

The paper describes an interactive 2D constraint-based geometric modelling system. It bases on distinguishing between two types of geometry. An auxiliary geometry is controlled by geometric constraints, and a desired geometry (we named it a visible geometry) is built in the framework defined by the auxiliary geometry. The auxiliary geometry has been used widely in the past at hand-made blueprints, but it is not handled at all in the present drawing packages.

Therefore, the presented system addresses surely a new approach in computer-based geometric modelling systems and gives directions how to realise an efficient user interface.

Although we can follow the development of constraint-based geometric modelling systems from the early beginning of the computer graphics in the sixties, the first commercial products have arisen very recently. The reason for this delay was the lack of interactivity caused by the problems of constraint solving (section 1.2 gives a brief survey of them). With presented approach we aim to minimise the described pitfalls with the following techniques:

- The local propagation of known states is chosen as a method for constraint solving because it can support an interactive design.
- The system automatically extracts all self-understandable constraints by following the designer's actions. The number of constraints required of the user is importantly reduced in this way.
- We developed a natural way of constraining Bézier cubics (the approach can be easily extended to other types of free-form curves). Again, this could be achieved by splitting the presented geometry in the auxiliary and the visible part.
- The system can handle well-constrained and under-constrained geometrical objects. In the case of an under-constrained object, the initial approximate geometrical data of geometric object entities are used. In this way, the user can see the under-constrained object and he/she can even change parameters which are already associated with partially well-constrained parts of a geometric object. In this way, the reusability of designed form-features is increased importantly. Additional constraining can be done later, when these parts are used to be bound together into the final shape. In this way, the time of giving additional constraints is synchronised with user's needs and, in this way, the whole constraining process is much easier. During the process of combining individual form-features we have to establish spatial relationships again, and therefore the

same set of constraints, the same user interface, and the same constraint solver could be used.

The system has been written in C++ and runs on personal computers in MS Windows environment.

Acknowledgements

This research is funded by Ministry of Science and Technology of Republic Slovenia under grant no. J2-5147-0796-94. The author would like to thank to Simon Kolmanic for his help during programming and the referees for their valuable comments and suggestions for improvements. I would like to thank also to dr. Fiaz Hussain from De Montfort University Milton Keynes, U.K. for his advice to apply presented approach to fonts.

References

[Aldefeld91] Aldefeld, B., Malberg, H., Richter, H., Voss, K., *Rule-Based Variational Geometry in Computer-Aided Design*, in « Artificial Intelligence in Design », D.T. Pham (Ed.), Berlin, Springer-Verlag, 1991, pp. 131-139.

[Ando89] Ando, H., Suzuki, H., Kimura, F., *A Geometric Reasoning System for Mechanical Product Design*, in « Computer Applications in Production and Engineering », F. Kimura, A. Rolstadas (Eds.), Elsevier Science Publishers, Amsterdam, 1989, pp. 131-139.

[Borning87] Borning, A., Duisberg, R., Freeman-Benson, B.N., *Constraint Hierarchies*, in Proceedings of Conference on Object Oriented Programming, Systems, Languages and Applications OOPSLA'87 (Orlando, 4-8 October 1987), pp. 48-60.

[Duerst93] Duerst, M.J., *Coordinate-Independent Font Description using Kanji as an Example*, Electronic Publishing, Vol. 6, No. 3, September 1993, pp 133-143.

[Falcidieno89] Falcidieno, B., Giannini, F., *Automatic Recognition and Representation of Shape-Based Features in a Geometric Modeling System*, Computer Vision, Graphics, and Image Processing, Vol. 48, No. 1, October 1989, pp. 93-123.

[Freeman-Benson90] Freeman-Benson, B.N., Maloney, J., Borning, A., *An Incremental Constraint Solver*, Communications of the ACM, Vol. 33, No. 1, January 1990, pp. 54-63.

[Fudos93] Fudos, I., Hoffmann, C.M., *Correctness Proof of a Geometric Constraint Solver*, Technical Report CSD 93-076, Department of Computer Science, Purdue University, West Lafayette, December 1993, ftp://arthur.cs.purdue.edu/pub/cmh/Reports/EREP/Erep4.ps.Z.

[Fudos96] Fudos, I., Hoffmann, C.M., *Constraint-based parametric conics for CAD*, Computer-Aided Design, Vol. 28, No. 2, February 1996, pp. 91-100.

[Hel-Or94] Hel-Or, Y., Rappoport, A., Werman, M., *Relaxed parametric design with probabilistic constraints*, Computer Aided Design, Vol. 26, No. 6, 1994, pp. 426-434.

[Herz94] Herz, J., Hersch, R.D., *Towards a Universal Auto-hinting System for Typographic Shapes*, Electronic Publishing, Vol. 7, No. 4, 1994, pp. 251-260.

[Karow90] Karow, P., *Font Technology*, Springer-Verlag, Berlin, 1990.

[Kurlander93] Kurlander, D., Feiner, S. *Inferring Constraints from Multiple Snapshots*, ACM Transactions on Graphics, Vol. 12, No. 4, October 1993, pp. 227-304.

[Leler88] Leler, W., *Constraint Programming Language*, Addison-Wesley, Reading, 1988.

[McMahon92] McMahon, C.A., Lehane, J., Williams, H.S., Webber G., *Observations on the Application and Development of Parametric-Programming Techniques*, Computer-Aided Design, Vol. 24, No. 10, October 1992, pp. 541-546.

[Rossignac90] Rossignac, J.R., *Issues on Feature-Based Editing and Interrogation of Solid Models*, Computer & Graphics, Vol. 14, No. 2, 1990, pp.149-172.

[Zalik93] Zalik, B., Guid, N., Vesel, A., *Reusability of Parametrized Geometric Objects*, Programming and Computer Software, Vol. 19, No. 4, July/August 1993, pp. 165 - 176.

[Zalik95a] Zalik, B., Guid, N., *An approach of Applying Constraints in Geometric Modelling*, Journal of Computing and Information Technology, Vol. 3, No.4, 1995, pp. 229-244.

[Zalik95b] Zalik, B., *Font Design with Incompletely Constrained Font-Features*, in « Computer Graphics and Applications », S.Y. Shin, T.L. Kunii (Eds.), World Scientific Publishing, Singapore, 1995, pp. 512-526.