33

A Ruler-and-Compass Geometric Constraint Solver

R. Joan-Arinyo, A. Soto
Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Av. Diagonal 647, 8a, E-08028 Barcelona
email: [robert, toni]@lsi.upc.es

Abstract

This work reports on the implementation of a two-dimensional, variational geometric constraint solver based on a constructive approach. The solver computes a solution in two phases. First, using rewrite rules, the solver builds a sequence of construction steps. Then the construction steps are carried out to generate an instance of the geometric object for the current dimension values. We discuss some issues concerning the data representation and the rules used. Then a simple example illustrates how the solver works. Finally we give a correctness proof of the solver.

Keywords

Computer aided design, geometric constraint solving, rewrite systems

1 INTRODUCTION

Constraints are an important enabling technology for interactive graphics applications. One of such applications where the use of constraints is growing more rapidly is in geometric modelling. The goal is to develop sketching systems based on geometric constraint solvers in which the user defines a rough sketch, annotated with dimensions and constraints, and the system builds an instance that satisfies all constraints.

Several approaches to the geometric constraint solving problem have been reported in the literature. Among them, the constructive technique is one of the most promising approaches. In this class of constraint solvers the constraints are satisfied constructively, by placing geometric elements in some order. They are based on the fact that most configurations in an engineering drawing are solvable by using a rather small set of tools like ruler, compass and protractor.

A geometric constraint problem is *well-constrained* or *consistently defined* if it has a finite number of solutions. If the problem has an infinite number of solutions then it is *underconstrained*. Finally a problem is *overconstrained* if after deleting one constraint the problem still has a finite number of solutions. In general, overconstrained problems have

M. J. Pratt et al. (eds.), *Product Modeling for Computer Integrated Design and Manufacture*

no solution, but a given overconstrained problem may have a solution if the additional constraints are consistens with previous constraints.

This paper reports on the work developed while building a prototype of a Geometric Constraint Solving System (GCSS). We consider only well-constrained, two-dimensional sketches formed from points, lines, circles and arcs of circle.

The GCSS has two major components, the *analyzer* and the *constructor*. The analyzer deals with the problem of determining symbolically whether or not a geometric sketch is solvable. It is based on a constructive approach which exhibits properties of both rule- and graph-constructive approaches. The analyzer is fed with a dimensioning scheme; i.e., a topologically correct sketch properly annotated with a set of constraints. Then, if the set of constraints consistently defines the object, the analyzer generates a sequence of constructive steps that determine each geometric element such that the constraints are satisfied. The constructor responds to the problem of building an instance of the geometric object. The instantiation is carried out by applying the sequence of construction steps generated by the analyzer to the actual parameter values. Whenever no numerical incompatibilities arise in the computation, an instance of the geometric object is generated. By clearly separating in this way the symbolic computation from numerical computation, we can use tools specifically tailored for each commitment. As a result, the interactive performance of the whole GCSS is improved.

Given a dimensioning scheme which consistently defines a geometric object, the analyzer, considered as a rewrite system, is canonical; i.e., the system has a unique normal form that is obtained by finitely many reduction steps.

2 A FORMALIZATION OF THE 2D PROBLEM

It is well known that the relative position of n given points $\{p_1, p_2, \ldots, p_n\}$ in the bidimensional Euclidian space, are determined by $2n - 3$ independent relationships defined between the points. In what follows we will use the words *relation* and *constraint* indistinguishably. We shall refer to a given set of points and a set of constraints defined between them as a *dimensioning scheme*.

We assume that a given set of n points on which a dimensioning scheme has been defined, can be split into two nonempty disjoint subsets. One subset, $\vec{p'} = \{p'_1, p'_2, \ldots, p'_k\}$, contains all those given points with fixed position. The other subset, $\vec{p} = \{p_1, p_2, \ldots, p_l\}$, contains all those points with unknown position. Note that this decomposition is always possible because $2n - 3$ independent relationships between n given points define a rigid body with three remaining degrees of freedom, two of them corresponding to a translation and the third one corresponding to a rotation. Hence, the absolute position for at least one given point should be specified.

As we shall see in Section 4, our approach to geometric constraint solving is such that given the set of points $\{p_1, p_2, \ldots, p_n\}$ and a dimensioning scheme $\varphi(\vec{p'}, p_1, \ldots, p_l)$ defined on them, we search a sequence of *construction* steps such that if the constraints define the geometric object in space consistently they will be able to figure out the relative position of each point.

3 APPROACHES TO GEOMETRIC CONSTRAINT SOLVING

In general *Numerical Constraint Solvers*, the constraints are translated into a system of algebraic equations and are solved using an iterative method like the Newton–Raphson method. *Constructive Constraints Solvers* are based on the fact that most configurations in an engineering drawing are solvable by using a rather small set of tools like ruler, compass and protractor. The constraints are satisfied constructively by placing geometric elements in some order. *Rule-constructive* and *graph-constructive* solvers are two different versions of the constructive approach. In *Propagation Methods* the constraints are first translated into a system of equations involving variables and constants. Then an undirected graph is created whose nodes are the equations, variables and constants, and whose edges represent the occurrence of a variable or a constant in an equation. The method then attempts to direct the graph edges so that every equation can be solved in turn, initially only from the constants. *Symbolic Constraint Solvers* transform the constraints into a system of algebraic equations; the system is solved with symbolic algebraic methods such as Gröbner bases. For a thorough review of approaches to geometric constraint solving see Bouma *et al.* (1993) and Fudos (1995).

4 THE SOLVER

The solver handles bidimensional geometric configurations composed from points, segments, arcs and circles. The constraints that can be defined on those objects include distance between two points, perpendicular distance between a point and a segment, angle between two segments, incidence, perpendicularity, parallelism, tangency and concentricity.

The solver is *variational*, i.e., the solver processes the constraints without the need of arranging them in a predefined ordering sequence. Furthermore, the solver can deal with dimensioning schemes with circular constraints.

Our basic method for solving geometric constraints is a constructive approach and exhibits properties of both rule- and graph-constructive approaches, Bouma *et al.* (1993). In a first phase, our solver uses rewrite rules to build a sequence of construction steps which are able to figure out the relative position of each point. In a second phase, the construction steps are carried out to generate an instance of the geometric object for the current dimension values. Let us see now how the data is represented and which rules are available.

4.1 Data Representation

All the constraints above mentioned can be represented by means of distance between two points, distance between a point and a straight segment and angle between two straigth segments. We use a notation derived from that reported by Verroust (1992). The distance between points constraint is represented by means of a **CD** set, the point-segment distance constraint is represented by a **CH** set, and the angle between two segments is represented by a **CA** set. We define these sets as follows.

A **CD** set is a set of points with mutually constrained distances. A frame of reference is attached to each **CD** set to which the points in the set are refered to. When a **CD**

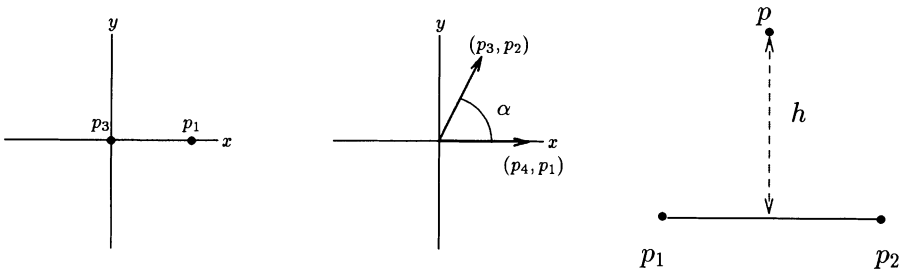

Figure 1 From left to right: Creation of elementary CD, CA and CH sets.

set contains just two points, we will refer to it as an *elementary* **CD** set. It is worth to note that a sketch is solved when all the points in the sketch belong to the same **CD** set. This **CD** can be seen conceptually as the result of applying the constructive formula in Section 2 to the initial set of constraints.

A **CH** set is a point and a segment constrained by the perpendicular distance from the point to the segment.

A **CA** set is a pair of oriented segments which are mutually constrained in angle.

In what follows, we will refer generically to the **CD**, **CA** and **CH** sets as *constraint sets.*

4.2 Rules

Depending on the functionality of the rules, we classify them as belonging to one of the following types: creation rules, merging rules or construction rules.

Creation rules

Creation rules create elementary **CD** sets, **CA** sets and **CH** sets as an interpretation of the dimensioning scheme defined by the user. The sign of the distances and angles are defined based on what the user has sketched. The following is illustrated in Figure 1. When an angle constraint between two directed segments is placed, a **CA** set is created. When a distance constraint between two points is given, a **CD** set is created. The position of the points in the associated frame of reference are $(0,0)$ and $(d,0)$. Whenever a point, a segment and the perpendicular distance from the point to the segment are given, a **CH** set is created.

Merging rules

Only one rule belongs to this type and it is illustrated in Figure 2. The rule allows to compute the transitive closure of the angle constraint set. When a segment belongs to two different **CA** sets, ca_1 and ca_2, a new **CA** set, ca_3, is created which constrains in angle two segments, one from ca_1 and one from ca_2, both segments being different from the segment shared by ca_1 and ca_2.

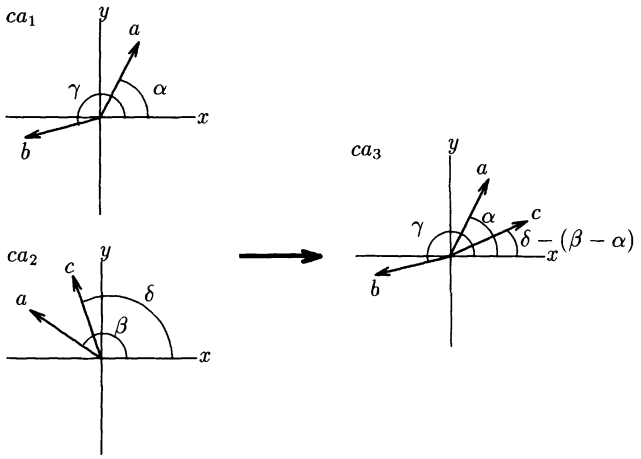

Figure 2 Merging two CA sets.

Construction rules

Construction rules merge **CD** sets, **CH** sets, and **CA** sets into larger **CD** sets. Merging is performed by building triangles and a few quadrilaterals. A complete description of each rule can be found in Juan-Arinyo *et al.* (1995).

4.3 Implementation

Figure 3 shows the architecture of our Geometric Constraint Solving System (GCSS). The geometric constraint solving problem is split into two main components: the *analyzer* and the *constructor*. The analyzer determines whether the dimensioning scheme is or is not symbolically solvable. If it is solvable, the analyzer generates a sequence of construction steps equivalent to the constructive formula we were looking for in Section 2. Then, the constructor instantiates the geometric object by applying the sequence of construction steps generated by the solver to the actual parameter values.

There are several reasons that lead us to this architecture. First, the nature of the computations in each step are quite different. The analyzer requires symbolic computation while the constructor only performs numerical computations. Second, determining whether the problem can be symbolically solved or not is performed in the analysis step and it does not depend either on the actual parameter values nor on the geometric computations. Next, with the proposed decoupling, when computing instances for different parameter values, only the second step needs to be carried out. This allows to skip the analysis step which is computationally the most expensive. Finally, given a symbolically solvable dimensioning scheme and a set of actual values for the parameters, the object

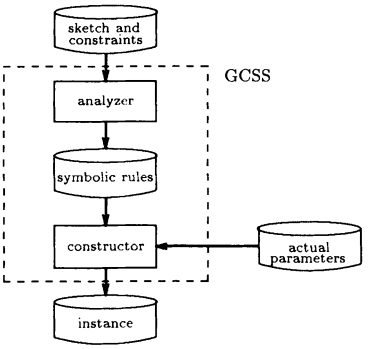

Figure 3 Solver Architecture.

can be instantiated if there are not numerical impossibilities. These impossibilities are detected while carrying out the geometric computations.

The analyzer is an expert system programmed in Prolog; Bratko (1990), Cosmadopoulos (1992). We have chosen this programming language because of the symbolic computation needed and because it is a rapid prototyping language.

The constructor is a virtual machine that executes the sequence of construction steps generated by the analyzer. In general, the position of the geometric elements can be expressed as non linear algebraic equations with the constraints as parameters in the equations. This means that a consistently defined geometric object may have an exponencial number of distinct solutions, dependent on the number of geometric elements. For exemple, Figures 4b and 4c show two possible solutions for the object in Figura 4a. The constructor provides a tool that allows the user to navigate interactively through the space of solutions.

5 CASE STUDY

In order to illustrate how the system works, let us present here a case study. Assume that the user has sketched and annotated a pentagon as shown in Figure 4a. The initial set of constraints derived from the dimensioning scheme defined by the user is

$$distance(p_1, p_2) = d_1 \quad (1)$$
$$distance(p_2, p_3) = d_2 \quad (2)$$
$$distance(p_3, p_4) = d_3 \quad (3)$$
$$angle((p_3, p_4), (p_3, p_2)) = a_1 \quad (4)$$
$$distance(p_4, p_5) = d_4 \quad (5)$$
$$distance(p_5, p_1) = d_5 \quad (6)$$
$$angle((p_1, p_2), (p_1, p_5)) = a_2 \quad (7)$$

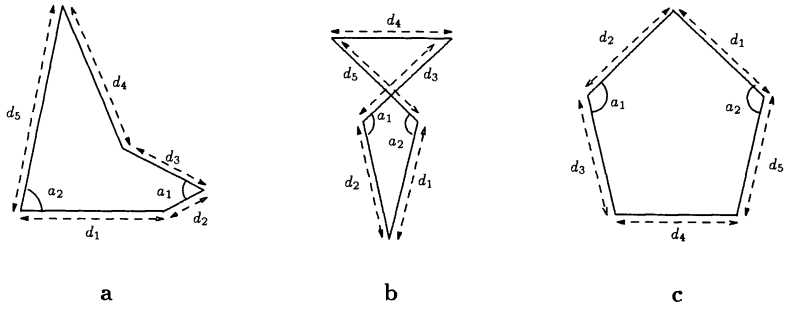

Figure 4 Case Study. a) Sketch and dimensionig scheme. b) and c) Instances generated with $d_1 = d_2 = d_3 = d_4 = d_5 = 100$ and $a_1 = a_2 = 120°$.

Starting with this set of constraints the analyzer writes it until either all the points belong to the same **CD** set or no rule applies. The data representation evolves as follows.

1. The **CD** set creation rule is applied to constraints 1, 2, 3, 5, and 6 giving the **CD** constraint sets $cd_1 = \{p_1, p_2\}, cd_2 = \{p_2, p_3\}, cd_3 = \{p_3, p_4\}, cd_4 = \{p_4, p_5\}$ and $cd_5 = \{p_5, p_1\}$
2. The **CA** set creation rule is applied to constraints 4 and 7 giving the constraint sets $ca_1 = \{(p_3, p_4), (p_3, p_2), a_1\}$ and $ca_2 = \{(p_1, p_2), (p_1, p_5), a_2\}$. Note that, in this case, the set of angle constraints and its transitive closure coincide.
3. Let us denote by `DDA1` the constructive rule that builds a triangle given by the lengths of two sides and the angle between them. Rule `DDA1` merges two **CD** sets by building a triangle on cd_1, cd_2 and ca_1. Then constraint sets cd_1, cd_2 and ca_1 are replaced with the **CD** set $cd_6 = \{p_2, p_3, p_4\}$.
4. Now rule `DDA1` can be triggered on cd_1, cd_5 and ca_2. These constraint sets are replaced with the **CD** set $cd_7 = \{p_1, p_2, p_5\}$.
5. Finally, if `DDD` is a constructive rule which builds a triangle given three **CD** sets with three pairwise common points, it can be triggered with the constraint sets cd_4, cd_6 and cd_7. These **CD** sets are replaced with $cd_8 = \{p_1, p_2, p_3, p_4, p_5\}$.

The last step yields a **CD** set to which all the points in the original sketch belong to. This fact tells the analyzer that the solving process is over.

The constructor now asks for actual values for the parameters and generates object instances by calling procedures in the underlaying geometric engine according to the construction steps.

6 SOLVER CORRECTNESS

After Brüderlin (1988) and Dershowitz (1990), the idea behind solvers based on geometric rewrite rules is to replace some facts in the database by *simpler* ones. In our solver,

initially, the **CD** and **CH** constraint sets, represent the sets of point-point and point-segment distance constraints derived from the dimensioning scheme while the **CA** sets are the transitive closure of the angle constraints defined by the user. The solver starts by applying the rules to these initial sets. Then the rules are repeatedly applied to the resulting constraint sets until either there is only one **CD** set which contains all the points in the sketch or no rule applies. In the first case the resulting **CD** set is a solution whereas in the second case the dimensioning scheme either does not define the geometric object consistently or is not solvable with the available set of rules. Every time a rule is triggered we will say that a reduction step has been performed.

Assuming that the dimensioning scheme does define the geometric object consistently, two things should be proved in order to prove the solver correctness: 1) The algorithm applying the rules stops in any case, and 2) The sequence in which the rules are applied does not matter for the result. In next section we recall the general form of the rules available in our solver as rewrite rules. Then we shall state the solver correctness.

6.1 Rewriting Rules

In rewriting theory, a rule over a set of terms $\mathcal{T}$ is an ordered pair $< l, r >$ of terms, which are usually written as $l \rightarrow r$. It is said that l rewrites or reduces to r; Bachmair (1991), Dershowitz (1990), Rosen (1973). When no rule applies to a term it is said that the term is *irreducible.*

In our system, all the construction rules can be expressed as rewrite rules where the terms on the left are constraint sets and the term on the right is always a **CD** set. The whole set of rewrite rules in our system can be found in Juan-Arinyo *et al.* (1995) and will be denoted by $\rightarrow_R$. In particular, all the construction rules $\mathbf{CR}_l \rightarrow \mathbf{CR}_r$, are such that $\mathbf{CR}_l = \{\mathbf{CX}_1, \mathbf{CX}_2, \mathbf{CX}_3\}$ and $\mathbf{CR}_r$ is a **CD** set.

6.2 Termination Properties

Let us define a *term* as a set whose members are constraint sets. Let $\mathbf{C}_0$ be the initial term with

$$\mathbf{C}_0 = \cup_i \mathbf{CD}_i \cup_j \mathbf{CH}_j \cup_k \mathbf{CA}_k$$

$\mathbf{CD}_i$ and $\mathbf{CH}_j$ are, respectively, the sets of point-point and point-segment distance constraints derived from the dimensioning scheme defined by the user. $\cup_k \mathbf{CA}_k$ is the transitive closure of the angle constraint sets defined by the user.

Let us denote by $\mathbf{C}_i$ the term whose members are the constraint sets after having applied the i-th reduction step. Then, the analyzer can be represented by the pair $(\mathbf{C}_i, \rightarrow_R)$ which is a *reduction system*; Rosen (1973). The proofs of the following statements can be found in Juan-Arinyo *et al.* (1995).

Theorem 1 *If the set of constraints consistently defines the geometric object, the reduction system* $(\mathbf{C}_i, \rightarrow_R)$ *has a uniform termination; that is, the analyzer stops after a finite number of reduction steps.*

Theorem 2 *If the set of constraints consistently defines the geometric object, different reduction processes starting from the same dimensioning scheme always terminate at the*

same irreducible term; that is, the sequence in which the rules are applied does not matter for the result.

7 CONCLUSION AND FUTURE WORK

We have presented the work developed while building a prototype of a variational geometric constraint solver based on the constructive approach which exhibits properties of both rule- and graph-constructive approaches. Given a wellconstrained dimensioning scheme, the solver, considered as a rewrite system, is canonical; i.e., the system has a unique normal form that is obtained by finitely many reduction steps.

Currently, our group is working on a number of topics that are of capital importance in order to build useful Computer Aided Design systems based on geometric constraint solving. One issue concerns detecting dimensioning schemes which do not consistenly define a geometric object because the scheme is either underconstrained or overconstrained. In both cases the system should provide appropiate hints to help the user. The other concern refers to extend the logic capabilities of the solver following two concurrent directions: 1) Given a subset of parameters with fixed values, figure out ranges of acceptable values for the remaining parameters, and 2) To extend the way of expressing constraints by including relations between variables.

8 ACKNOWLEDGEMENTS

This work has been supported in part by the CICYT grant TIC95-0630-C05-04.

REFERENCES

Bachmair, L. (1991) Proof methods for equational theories. Technical report, Department of Computer Science, SUNY at Stony Brook, Stony Brook, New York 11794.

Bouma, W., Fudos, I., Hoffmann, C., Cai, J. and Paige., R. (1995) A geometric constraint solver. *Computer Aided Design* 27(6):487–501, June.

Bratko, I. (1990) *PROLOG. Programming for Artificial Intelligence. Second Edition.* Addison Wesley.

Brüderlin, D.D. (1988) *Rule-Based Geometric Modelling.* PhD thesis, Institut für Informatik der ETH Zürich.

Cosmadopoulos, Y. and Chu, D. (1992) *IC Prolog II.* Deprt. of Computing, Imperial College, London, November. Version 0.94, for Sun Workstations.

Dershowitz, N. and Jouannaud, J.-P. (1990) *Handbook of Theoretical Computer Science*, chapter Rewrite Systems, pages 244–320. Elsevier Science Publishers B.V.

Fudos, I. and Hoffmann, C.M. (1993) Correctness proof of a geometric constraint solver. Technical Report CSD 93-076, Department of Computer Sciences, Purdue University, December.

Fudos, I. (1995) *Constraint Solving for Computer Aided Design.* PhD thesis, Purdue University, Department of Computer Sciences.

Juan-Arinyo, R. and Soto, A. (1995) A set of rules for a constructive geometric constraint solver. Technical Report LSI-95-19-R, Department LiSI, Universitat Politècnica de Catalunya.

Rosen, B.K. (1973) Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, January.

Verroust, A., Schonek, F. and Roller, D. (1992) Rule-oriented method for parameterized computer-aided design. *Computer Aided Design*, 24(10):531–540, October.