# Construction and manipulation of geometric figures

R. Allen, Denis Bouhineau, Laurent Trilling

# Construction and manipulation of geometric figures

Richard Allen, allen@stolaf.edu
Denis Bouhineau, bouhino@imag.fr
Laurent Trilling, trilling@imag.fr

January 12, 1993

Our problem domain is automatic construction of geometric figures which conform to a logical specification. The specification contains geometric objects and geometric constraints among the objects. The objects are points, lines, and circles, and the constraints represent geometric properties such as belonging, perpendicular, parallel. Our main objective is to provide a system for teaching geometry which constructs figures efficiently and which, for a given specification, is capable of constructing all possible figures conforming to the specification, that is animation. It is also important to provide the system with a natural notion of completeness.

In this extended abstract, we outline our approach: the introduction of geometric constraints as linear algebraic constraints; the use of the linear equation solver of PrologIII to resolve these algebraic constraints; and our approach for handling nonlinear constraints and for providing a notion of completness

**Introduction :** Currently available software for learning and teaching geometry (for example, the microworlds, Sketchpad and Cabri) permits construction and animation of geometric figures using the equivalent of an imperative geometric programming language. The user first exhibits a procedural construction of the figure and afterwards can drag certain initial objects of the figure, producing a new figure, which is a displacement of the original one still possessing the same objects and properties as before. The software constructs the new figure according to the procedure given in the first step. The consequences of imperative geometric programming are simple: all displacements of the original figure are not possible, only those starting from initially free objects are. Consequently, the order in which the figure has been constructed is important, and, more importantly, the exploration of the figure through the addition or supression of certain constraints is delicate.

These observations reinforce the value of defining and implementing a declarative geometric programming language and we have found that Constraint Logic

Programming is the most adequate tool for achieving it.

**Resolution :** The first idea was to transform all the geometric constraints into algebraic constraints and then to give them to PrologIII. This simple solution gives very good results when the equations introduced are linear (this being the case for geometric constraints such as belonging, perpendicular, parallel) since PrologIII computes the exact solution of linear systems over rationals. Moreover, the overconstraints in the specification are verified. In such a system, we would like to say that all ruler and square constructions are solved by linear algebra; yet this is not the case, even for ruler only constructions. In fact, it is actually hard to find a notion of completness which corresponds to this particular restricted case.

For the first approach, problems appear with the introduction of distances and circles since they correspond to second degree equations. Such equations introduce floating point numbers and the correctness of the resolution of linear systems is no longer assured since an overconstraint may be not numerically satisfied because of accumulated error in the processing of real numbers. Since overconstraints arise naturally in geometry, it is important that they be treated satisfactorily. In the present design, overconstraints are assumed to be always satisfied. In the next implementation we are considering verifying them symbolically with a geometry theorem prover like the one proposed by Wu, or at least verifying that there are satisfied with interval arithmetic.

Technically, each geometric constraint is associated with an algebraic equation, and each equation is associated with a process whose purpose is to solve the equation; that is, every linear equation is introduced into the PrologIII linear equation solver and every second degree equation like $(x-x_0)^2+(y-y_0)^2-r^2=0$ are solved in a classical way when all variables appearing in it are linearly dependant of an unique one $X$, that is when the equation can be rewrite in $aX^2+bX+c=0$. The program consists of two parts. The first part analyses the specification and creates all the processes by means of the PrologIII built-in predicate freeze(s,B) which delays the evaluation of the predicate B until the instanciation of s. The second part is the control part which activates the processes by means of previously created triggers. There are two triggers, one for each class of equations. The control part starts the activation of first degree equations. First degree equations are only considered for solution when they are truly linear. Note that an equation like $Y-a*X+b=0$ is not always linear, since a and X may be both not known; such an equation is said to be pseudo-linear. If an awakened equation be not linear, then its introduction into the solver is delayed again. On the other hand, when all variables are instanciated, the equation is said to be an overconstraint (the same is said of completely instantiated second degree equations), and values are assumed to be exact. Second degree equations are considered only after all possible insertions of linear equations into the solver have been accomplished; in order to activate

second degree equations the control part verifies that the activation of first degree equations has not lead to results. If a first degree equation is introduced, then control activates the remaining first degree equations one more time. If an awakened second degree equation can be solved, then it is effectively solved and the control part tries again to introduce linear equations. The resolution stops when every geometric element is computed or when all processes are blocked.

Concerning completeness of this system, we can say that, under the unique name assumption for the specification, if a construction with ruler and compass exists using only the elements occuring in the specification, then a construction is provided by the system and all possible animations can thus be obtained. Otherwise nothing is guaranteed. It is important to recall that from our educational perspective, the essential goal is to provide a system with increased power over imperative geometric programming.

**Development and perspectives :** Certain problems have led us to improve our resolver. Many specifications require supplementary constructions to permit overall construction of the figure. Consequently, we systematically add the common chord of two circles when some point of the specification belongs to it. Likewise, we could have added other objects which can be considered as overconstraints or as help for the resolution of the problem. But it is hard to guarantee that the unique name assumption will be retained with those new objects (note that this overconstraint would be always verified).

The general resolution of the intersection of a line and a circle requires a predicate which indicates whether two variables are in linear dependance or not. At the present moment, PrologIII only indicates if a variable has been instantiated or not. It doesn't say if it is linearly dependant with another one. So we had to built such a predicate which is not straightforward and uses side-effecting. We think it would be of general interest if PrologIII would offer a predicate ResLin$(X, Y, a, b, c)$ which succedeed if $aX+bY+c = 0$ at the moment of the evaluation, with some restrictions for $a, b, c$.

The representation of lines have been chosen considering the priority given to first degree equations: the line $\delta : aX + bY + c = 0$ is represented by $\delta : (a, b, c)$ and the triple $(a, b, c)$ is defined by (1,b,c) or (0,1,c). It is quite easy to do with Prolog, and exotic in algebra. Normally, in algebra, a line is uniquely described by $(a, b, c)$ with $a^2 + b^2 = 1$, which introduces second degree equation. The disadvantage of such a definition is in the possible introduction of a exponential explosion of the complexity of the Prolog program. The existence of the predicate ResLin$(X, Y, a, b, c)$ may lead to improvments in this domain too.

Existing implementations execute in reasonable time and we anticipate future implementations to handle more complicated specifications as well as improved execution times.