Selecting a figure using S-homotopy in a CAD system

C. Essert-Villard, P. Schreck, J.-F. Dufourd

LSIIT, UPRES-A CNRS 7005
Université Louis Pasteur de Strasbourg, France
essert,schreck,jfd@dpt-info.u-strasbg.fr

Abstract: In CAD systems, formal geometric solvers allow to scan the entire space of the solutions found for a constrained sketch drawn by the designer. We introduce a sketch-based method that enables to easily eliminate most of the solutions and to keep the only, or at the worst the few solutions that have the best likeness with the original drawing.

Keywords: formal geometric constructions; constraint solving; tree pruning; computer-aided design

1 Introduction

In Computer-Aided Design (CAD), drawing constrained figures and automatically solving them are subjects that have been studied by many authors, but that still remain topical. In geometric modeling, constraints bring a precise description of geometric properties that must be respected by the object. Two main classes of solving constraints can be distinguished: numerical approaches and formal methods. The first ones are often chosen, and consist in solving numerically the equation system related to the dimensions [2, 6, 8, 9]. A formal resolution of the symbolic constraint system offers the advantage of efficiently manipulating the defined figure by varying parameters values [1, 3]. Unfortunately, algebraic tools for formal calculus are much too general to be efficient for that purpose. We gave priority to a both formal and geometric approach, that was materialized by a software called YAMS [4, 7].

Whatever the approach, a constraint system doesn't usually define a single figure. Indeed, the existence of polynomial equations whose degree is higher or equal to 2, on an algebraic point of view, or of multiple intersections, on a geometric point of view, quickly leads to a combinatorial explosion of the number of solutions. In most cases, CAD users only want one solution figure when they design an object. That's why an important matter of geometric solvers is identifying the solution that is most consistent with the user's expectations, as we can see in [2] and [6]. The most common response to this problem is the use of heuristics to filter the results. When using a numerical method, the constrained figure is compared with each of the numerical solutions. This is generally characterized by slow runtimes, and there is often more than one solution left. Our formal approach allows us to take advantage of the construction program to compare the sketch with a solution.

The rest of the paper is organized as follows. Section 2 outlines our formal approach of constraints solving, and the relating notions. Section 3 exposes our vision of likeness between figures, and a structuring of the solutions space. Section 4 presents a method to choose the intended figure amongst many solutions. Section 5 deals with restrictions that apply to it, and the way to solve particular cases. Section 6 shows some results provided by our technique, and Section 7 concludes.

2 Geometric constraints system solving

We now present the solver part of YAMS, which acts in two stages, a symbolic one and an interpretative one.

2.1 Symbolic resolution

In the first stage, given a dimensioned sketch, the solver associates the geometric objects with some identifiers, then turns the constraints into formal parameters to form the geometric constraint system $S = (X, A, C)$, where X is a set of unknowns, A a set of parameters, and C a set of constraints of the form $C = \{p_1(X, A), \ldots, p_m(X, A)\}$ where each $p_i(X, A)$ is a predicative term, namely a constraint, whose variables are in X or in A.

Then, according to the constraints, YAMS produces *definitions* that ensure correspondence between functional terms and identifiers. The definitions are brought together forming a general *construction plan*, that is the result of the formal phase. This plan indicates how and in what order the geometric objects must be built to produce the figure. More formally, a plan T is a result of a symbolic resolution of a constraint system S if T is a triangular solved system, and $S \equiv T$, that is S and T have the same solutions. Using instantiation of parameter symbols in A with values of any tuple u, we obtain $S_u \equiv T_u$.

2.2 Interpretative stage

In the second stage, the original dimensions are used as parameters for the numerical interpretation of the construction plan. Since used functional terms may provide multiple results, each functional symbol is associated with a numerical *multifunction*. For example, the intersection between a line and a circle, symbolized by *interlc*, may produce two points.

Once values are assigned to the parameters, we can consider the interpretation as the building of a tree labeled with numerical values. Each interpreted definition produces a branching. At the end, the tree called *solutions tree* represents the solution space, and one solution corresponds to the labels of one branch. Even if, obviously, this tree is not really built in practise but explored by backtracking [5], it may increase fast and be very wide. Thus, it is useful to precisely number the solution.

Actually, the computed construction plan enables to construct all the solutions as well as other figures which are "false solutions". The false solutions can quickly be eliminated thanks to a simple test, as they are not consistent with the constraints. But that may be insufficient. On the example presented on Fig.1, there are 32768 different solutions for a geometric object made of 15 equilateral triangles figure (each segment corresponds to an equally constrained distance), but the solution space can't be reduced because all of the figures are consistent with the constraints. Some of them are presented on Fig.2 (some triangles may be superimposed). Other heuristics are necessary to drastically prune the tree of solutions, eliminating the figures that doesn't look like the sketch.

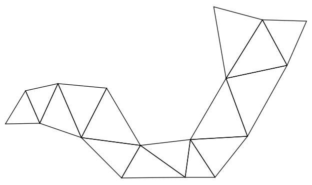
Figure 1: 15 triangles configuration : the sketch

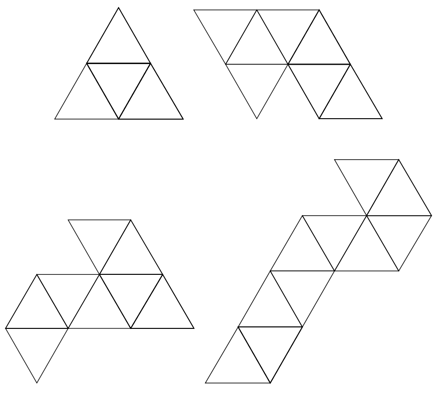
Figure 2: Four solutions among 32768 to "15 triangles"

3 Using the sketch's data to prune the tree of solutions

Our purpose is to obtain a single solution figure that bears the best resemblance to the original drawing. First, let us define what we mean by saying a figure looks like another.

3.1 Usual criteria of likeness

Two figures are often said to look like each other if they have some geometric properties in common, such as relative placing of points, lines and circles, angles acuteness, and convexity of some parts of the sketch. We only notice that most of these properties comparisons can be held in check by some simple examples : on Fig.3 b), all angles are acute and all points have the same relative placing so we can't decide ; on Fig.4, the sketch has a convexity flow with respect to the solution.

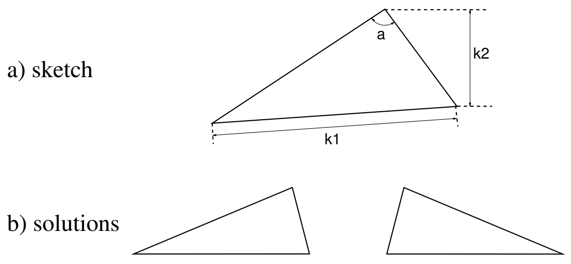

Figure 3: Lack of discrimination criterion

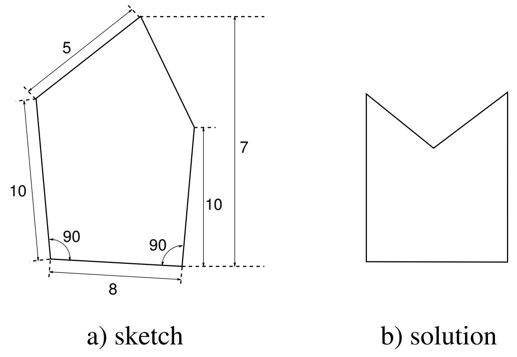

Figure 4: Convexity flaw

3.2 Homotopy as a notion of likeness

When considering only topological properties, the continuous deformation between two parametric curves called *homotopy* is an usual likeness concept. Of course, this definition is much too general for us because it doesn't take into consideration elementary geometry properties of the objects, particularly their type. For example, a circle is homotopic to a triangle, and this is meaningless for CAD users. So, we have to refine this characterization in a geometric framework.

With the help of a coordinates system, we can define a metric topology, from which a notion of proximity follows. Now we can give our definition of *geometric homotopy*, that is a continuous transformation that preserves incidence relationships and geometric types (such as points, lines, etc.).

However, geometric homotopy does not take into account the constraint systems. We have to refine the notion in this sense.

3.3 Constrained deformation

If we want to study continuous deformations of a dimensioned figure, not only have we to define the continuous deformation of a figure, but also of its constraint system. So we introduce the definition of *continuous deformation of a constraint system* S as the continuous deformation preserving the number of solutions of S and of every well-constrained sub-system of S.

Thus, we can link this notion with the continuous geometric deformation of a figure. If there is a continuous deformation ψ between two different instances S_u and S_v of a constraint system, and a continuous deformation φ between two figures e and f that are respectively solutions of S_u and S_v, and if $\forall t \in [0,1]$, $\varphi(t)$ is a solution of $S_{\psi(t)}$, then we say that there is a geometric homotopy between e and f with respect to the constraint system S, in short a *S-homotopy*.

Note that, the deformation of the constraint system must not reach any degenerate case, because if that occurs we can swap to another solution instead of always following the same one. This leads us to give a numbering for multifunctions values, compatible with the continuous deformation of the constraint system, which we define as *continuous numbering*.

Now that we exposed our vision of likeness and numbering, we will be able to make the link between these notions, by saying that two figures e and f are S-homotopic if and only if they have the same number in their solutions tree. In addition, given a sketch e, there is at most a unique f such that e and f are S-homotopic.

3.4 Practical numbering

We also notice that some geometric properties characterize the discrimination between the values of a multifunction. Then, for every multifunction we currently use in our solver, we described such a geometric characteristic. For example, the results of *mkcir4*, that yields a circle c_2 tangent to a given circle c_1, are differentiated by the placement of c_2 with respect to c_1, inside or outside. The degenerate case is reached when c_1 has a radius equal to zero.

These geometric properties are preserved through a continuous deformation, so they allow us to define a continuous numbering.

4 Freezing of a branch

We put forward the hypothesis that all the expectations of the designer are in his constrained sketch.

We first point out the fact that the sketch can be seen as a particular solution of the constraint system instantiated by the values read on the sketch. So, it corresponds to a particular branch of the tree of solutions, that we call the *sketch branch*. Then, we suppose that the user gave a sketch and some constraints that look like his expectations, that is, in the sense we defined earlier, there is a continuous deformation between the sketch and the solution. Thus, if we can find a solution having the same number than the sketch, then we advance the idea that it is the intended solution.

The operation consisting in storing the number of the sketch branch is called *freezing* of a branch. We this number is known, we can launch an interpretation with the given dimensions, guided by the number of the *frozen* branch.

One of the advantages of this method is its speed. Indeed, instead of potentially comparing some geometric properties of all the objects of the figure with each other, we only compare, at each junction of the tree, the objects that are brought into play in the concerned multifunction. Since the treatment is made as the interpretation goes along, this method reduces significantly the processing time in comparison with a systematic method. An example of a result provided by this method can be seen on Fig.5. It shows the single solution found by using the freezing of a branch on the constrained sketch given on Fig.1.

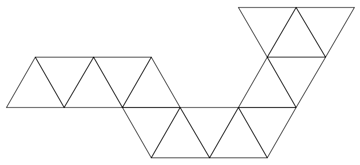

Figure 5: The required solution of the sketch given on Fig.1

Let us note here that, as geometric criteria on multifunctions are dependent on lines orientation. This orientation of all objects is computed from the sketch and the user's constraints.

5 Discussion

The method we exposed in previous section works fine when all constraints are metric. But another type of constraints is also used in YAMS: boolean constraints. As examples, we can cite tangency, or equality of objects. In the case of boolean constraints, some information is missing to find the intended solution. Actually, unlike metric constraints that don't affect topology, these constraints are not always respected on the sketch, as shown on Fig.6 and 7: on the sketch, the circle is actually not tangent to the line contrary to the constraints given by the user.

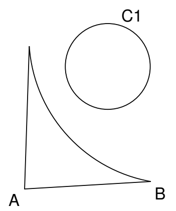

Figure 6: Tangency problem : the sketch

In these situations, two general approaches can be considered, those correcting the sketch to fit the previous conditions, and those producing several branches giving a small subtree to

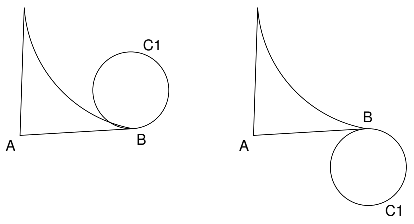

Figure 7: Two possibilities for tangency

explore. The first one has not yet been studied, but it may consist in interpreting the construction plan with the data read on the sketch by correcting the inconsistent objects as they go along. For example, on Fig.6, the sketch could be modified in such a way that the circle becomes tangent to the line. Among the two possible circles given by the construction plan, we choose the one that is closest to the circle on the sketch, in the sense of Euclidean distance over the coordinates.

The second approach consists in making a maximum freezing, that means keeping all the possible solutions if it is not possible to decide. The result is a frozen subtree that can be explored thanks to some tools provided by the software. Among these tools, we can cite classical geometric heuristics to prune and/or classify the branches of the remaining subtree, algorithmic tools to make faster the complete or a partial exploration of the solution space (hash table, intelligent backtracking, reduction of the complexity of the tree, etc.), and an user friendly interface.

A first tool of this user friendly interface is inspired by *debug tools* provided by most of the development systems in software engineering. This is possible because our approach is formal and we have a construction plan. So it is easy to do a step by step evaluation, allowing the user to choose, at each fork, a value among the available results. This simple mechanism can be enhanced with several kinds of breakpoint tools, like in *Prolog*. Moreover, it is possible to offer the opportunity to freeze a part of a solution tree between two breakpoints, and then to skip this part that has become a big step.

We also intend to implement a second class of tools, that is based on the idea of a *magnetic grid*. It allows a more intuitive approach of the selection problem. On the basis of a solution that doesn't fit the user's expectations, he can drag the misplaced element of the figure until one of the positions allowed by the tree of solutions.

6 Results

In order to illustrate our method, we expose here a quite representative example. The sketch on Fig.8, showing a lever, comes with 106 constraints including 2 tangency constraints, $tgcl(c1, l1)$ and $tgcl(c3, l2)$. Note that to lighten the figure, we avoided to represent the constraints by arrows.

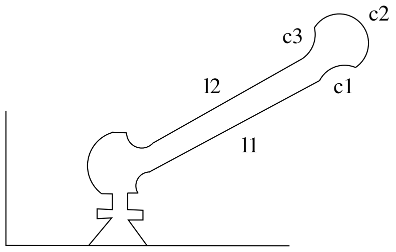

Figure 8: Sketch of the lever

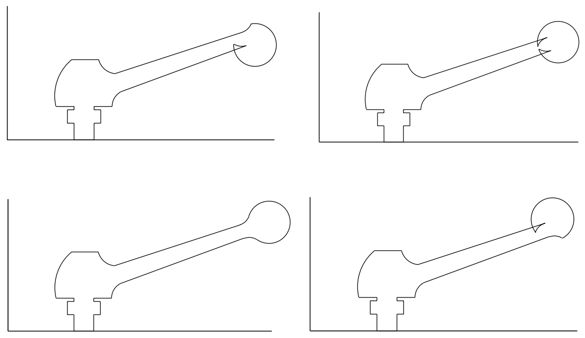

Figure 9: Solutions for the lever

Using these constraints, the solver produces a construction plan, containing 252 definitions. As among these definitions, 29 have an arity of 2, the tree of possibilities of this constraint system has $2^{29} = 536870912$ branches. Actually, some branches lead to an interpretation failure, and others can be eliminated with a simple constraints verification, so the tree of solutions provided by the interpretation stage only has 160 branches.

Our method allows to prune significantly the tree of solutions, providing a four-branched subtree. The remaining solutions can be seen on Fig.9. Among the four figures, the part that remains unchanged corresponds to the metric constraints, and the uncertain part corresponds to the two tangency constraints. It is not possible to have only one solution since the tangency constraints are not respected on the sketch. To prune the subtree, we have implemented classic heuristics such as relative placing of circles and lines. With these heuristics, we found the intended solution, that is the bottom-left part of Fig.9.

7 Conclusion

In this paper, we exposed our formal approach of geometric constructions, that yields a construction plan from a dimensioned sketch. Then, we defined a notion of likeness coming from the topological homotopy notion called *S-homotopy*. This allows us to define in some way what is the structuring of the solution space of a constraint system. We also proposed a method to select *one* solution amongst many, by *freezing* a branch of the tree of solutions with the help of the sketch.

This method works fine while all constraints are metric, as we shown on some simple examples. However, as the limits of our method were clearly identified, we studied some tools to explore the solution tree, and to help the user to find the intended solution.

In previous papers [4, 5], we exposed that a symbolic solving has many advantages. Here we showed that it is also useful for solutions selection or exploration. Various debugging tools will soon be implemented, and we plan to develop an intuitive graphic interface to deal with them.

Further work is needed to analyze more in detail the structuring of solutions space. For example, in the case of articulated systems animation, one of the problems is the crossing of dead points. This problem is linked with some degenerate cases, and with the transition from one branch to another in the tree of solutions.

References

[1] B. Aldefeld. Variations of geometries based on a geometric-reasoning method. *Computer-Aided Design*, 20(3):117-126, 1988.

[2] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer-Aided Design*, 27(6):487-501, 1995.

[3] B. Brüderlin. Constructing three-dimensional geometric objects defined by constraints. *Proceedings of the ACM Siggraph Workshop in Interactive 3D Graphics*, p.111-129, 1986.

[4] J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Journal of Artificial Intelligence*, 99(1):73-119, 1998.

[5] C. Essert, P. Schreck, and J.-F. Dufourd. Interprétation d'un programme avec multifonctions géométriques. *Proceedings of 6èmes journées de l'AFIG, Dunkerque, France*, p.223-232, 1998.

[6] H. Lamure and D. Michelucci. Solving constraints by homotopy. *Proceedings of the ACM-Siggraph Solid Modelling Conference*, p.134-145, 1995, ACM Press.

[7] P. Mathis. Un système de résolution de contraintes par assemblage en modélisation géométrique, Ph.D. Thesis, Université de Strasbourg, 1997.

[8] J. Owen. Algebraic solution for geometry from dimensional constraints. *Proceedings of the 1st ACM Symposium of Solid Modelling and CAD/CAM Applications*, p.397-407, 1991, ACM Press.

[9] G. Sunde. Specification of shape by dimensions and other geometric constraints. *Proceedings of the Eurographics Workshop on Intelligent CAD systems*, Noordwisjkerout, 1987.