Towards an homogeneous handling of under-constrained and well-constrained systems of geometric constraints

Simon E.B. Thierry
Ph.D. student
LSIIT, UMR CNRS 7005
Pôle Technologique,
BP10413, 67412, Illkirch,
France
thierry@lsiit.u-strasbg.fr

Pascal Mathis
Associate Professor
LSIIT, UMR CNRS 7005
Pôle Technologique,
BP10413, 67412, Illkirch,
France
mathis@lsiit.u-strasbg.fr

Pascal Schreck
Professor
LSIIT, UMR CNRS 7005
Pôle Technologique,
BP10413, 67412, Illkirch,
France
schreck@lsiit.u-strasbg.fr

11 March 2007

Abstract

Most of the geometric constraints solvers consider systems of constraints well-constrained *modulo* the rigid motions group, and either halt on error when they encounter under-constrained sub-systems, or attempt to add parameterized constraints so as to get rid of the under-constriction.

We studied transformations groups making well-constrained some problems that are usually considered as under-constrained. This leads to new algorithms which allow an homogeneous handling of systems of geometric constraints and thus a better adaptation to the needs of the user.

Geometric constraints solving, Under-constraint, Invariance under transformations groups

1 Introduction

Geometric constraints solving in Computer-Aided Design (CAD) aims at yielding a figure which meets some metric requirements (*e.g.* distances between points or angles between lines), usually specified under graphical form. Solutions are returned as the coordinates of the geometric entities. >From the designer's point of view, there must be one and only one solution to a constraints system, since it is used to represent a physical object. Yet, a constraints system may have no solutions due to numerical inconsistancies. It is then said over-constrained. On the other hand, a system may have an infinity of solutions when it is under-constrained, that is to say when there are not enough constraints to compute the position of every geometric entity. In both cases, geometric constraints solvers usually halt on error, either because it is impossible to find a solution, or because they cannot choose one in particular. Solvers are generally intended to work on well-constrained systems, for which there is a finite – but possibly large – number of solutions. For further details about the field of geometric constraints solving and the most efficient current methods, the reader may refer to [4, 7].
The notion of well-constriction, though, is ambiguous. Rigid systems, *i.e.* systems where all distances and angles are known, are usually considered as well-constrained. The geometric entities of these systems have no fixed coordinates. For the designer, moving an object does not change it, but in terms of coordinates, there is an infinity of solutions.
Explicitly expressing an invariance group can solve this ambiguity: if there are a finite number of solutions from which, using transformations of a given group G, one can generate all the solutions of the system, these solutions are invariant under the group G. The system itself is said well-constrained *modulo* the invariance group. A rigid object is then well-constrained *modulo* the rigid motions.

Using classical transformations groups (rotations, translations, scaling operations, and their compositions) solves the ambiguity explained above. In this paper, we present a way to extend this approach so as to get an homogeneous way of handling well-constrained and under-constrained

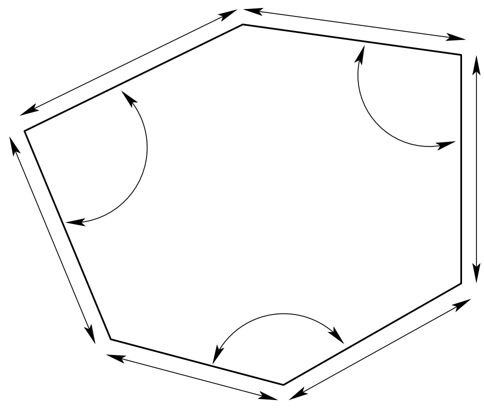
Figure 1: Rigid GCS

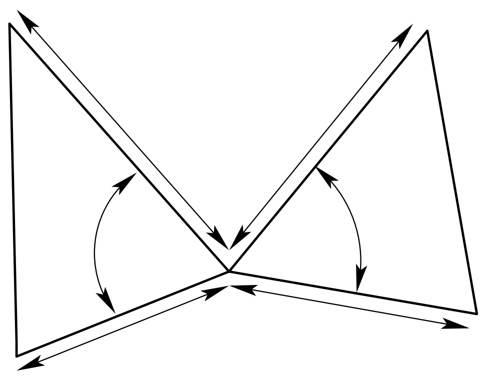
Figure 2: Non rigid GCS

systems: we expose transformations groups that are specific to the system to be solved, acting only on parts of it. This enables us to build algorithms working on any kind of geometric constraints systems, without prior hypothesis on its level of constriction.

The rest of this paper is organized as follows. Section 2 is an overview of existing work about under-constriction. Section 3 explains the philosophy and necessary definitions of our approach. Section 4 describes algorithms aiming at obtaining an invariance group of a system. Section 5 discusses the application fields and the advantages of our approach.

2 Related work

Detection and eradication of under-constriction are major issues in the field of geometric constraints systems, since most of the resolution methods make the hypothesis that the system is well-constrained. The possibility to determine the well-constriction of a subsystem even is mandatory for decomposition methods [7].

Most of the methods to detect under-constrained parts of a system are based on flows, following the approach of Latham and Middleditch [10]. In [6], Jermann *et al.* proposed a flow-based method extending the one of Hoffmann *et al.* [5] and considering a more generalized notion of rigidity which includes numerical consistancy.
In [14], Sitharam and Zhou introduced an approximate rigidity characterization in 3D distance constraint graphs, called module-rigidity.
The witness configuration method of Michelucci and Foufou [12] is a numerical method that also allows the detection of non-rigid parts by computing Cayley-Menger determinants.

Since detecting an under-constrained system only allows the software to produce a warning to the user, automatically transforming it into a well-constrained system is a major issue in CAD.
An important work on this domain was done by Joan-Arinyo *et al.* [8], introducing the notion of *deficit* and explaining a decomposition method, based on a modification of Owen's algorithm, that extracts under-constrained subsystems and finds which parameterized constraints should be added to deal with the under-constriction.
Based on this method, Gao *et al.* [3] proposed efficient completion algorithms. Another successful attempt is the one of Li *et al.* [11], which deals with under- and over-constrained systems in 3 dimensions, but has a high complexity.
All these methods need to rerun the algorithm from scratch when the user adds a new constraint.

In the field of mechanisms, a lot of work has been done on the synthesis of articulated systems and the computation of motion abilities of a given solution under kinematic constraints. To our knowledge, few papers are about methods to find an initial position to a mechanism designed with constraints. For more details, see [1] and references therein.

The works by Kramer [9] are based on an incremental assembly of geometric objects during which the algorithm reckons the degrees of freedom up at linking points. The degrees are removed with the consideration of the incidence constraints and using a simple loci method. If some degrees still remain though all objects and constraints have been consumed, a point of articulation is found. This approach may fail with some systems, such as the one on figure 1, where the lengths of the "inner triangle" are not computed.

In [13], we solved the ambiguity evoked in the introduction by considering as well-constrained *modulo* a transformations group a system with a finite number of solutions from which, using the transformations, all solutions can be obtained.

3 Towards a new definition of well-constriction

As seen in section 2, most current applications try to add parameterized constraints so as to get rid of the under-constriction, but this approach is somehow limited by a few problems. First of all, not all under-constrained systems can currently be detected, and there is no complete algorithm to add parameterized constraints to any known under-constrained system. Even if such an algorithm existed, one still would not be assured to satisfy the user's will, since there often are several ways to add constraints, but not all have a "meaning" for the user.

Moreover, a user may *want* to sketch an under-constrained system. For instance, a pair of scissors has a remaining degree of freedom due to the rotating possibility around the linking point of the two scissors.

3.1 Motivation

Our approach intends to avoid these drawbacks by generalizing the notion of well-constriction to under-constrained systems, and finding ways to handle them the same way we handle classical well-constrained systems. We consider a system as well-constrained *modulo* an invariance group.

Definition 1: Well constriction *modulo* an invariance group
A geometric constraints system $\mathcal{S}$ is well-constrained *modulo* a group G if there exists a finite number of particular solutions $S_1 ... S_n$ of $\mathcal{S}$ such that any solution of $\mathcal{S}$ can be obtained by the application of a transformation of G to one of the n particular solutions. $\square$

In [13], we considered classical geometrical transformations groups, such as the *euclidean* group, the rigid motions or the similarities. Considering only these groups does not help here, since it still implies many prior conditions on the system to be solved.

Without prior hypothesis on the system, there always exists at least one invariance group: the group of the permutations of the solutions. This group is specific to the system, and the corresponding transformations exist only for its solutions. Obviously, this group cannot be useful because it involves the prior knowledge of all the solutions, which is precisely the goal of geometric constraints solving.

Our approach is an intermediary approach: by studying the system, we build transformations groups, specific to the system to be solved. These transformations are based on classical groups (*i.e.* translations, rotations and scaling operations) but may apply only to some parts of the solution figures. For instance, a GCS describing a pair of scissors is well-constrained *modulo* a group, which we call a *rotoid* transformations group, consisting in a translation of the linking point of the two scissors, and of two rotations around this point, each applying only to one of the scissors.

3.2 Definitions and terminology

In this paper, we assume that the considered systems are 2D systems with distances and angles between points and lines.

Definition 2: Solutions of a GCS
A geometric constraints system is a triple $\mathcal{S} = (C, X, A)$ with C (the constraints) a set of propositional terms on $X \cup A$, X the set of unknowns, generally geometric entities, and A the set of parameters, *i.e.* metric values of the constraints or entities with given coordinates.

A solution of $\mathcal{S}$ is a valuation of the unknowns in the considered geometric universe. $\square$

3.2.1 Assembly transformations

Based on our geometric universe, we built three families of transformations groups: rotoid transformations, sliding transformations and positioning transformations. We call transformations of one of these families *assembly transformations.* Let us first define on which kind of systems these transformations apply.

Definition 3: Non rigid assemblies
Let $\mathcal{S}$ be the union of two non self-included GCS $\mathcal{S}_1$ and $\mathcal{S}_2$, such that $\mathcal{S}_1$ and $\mathcal{S}_2$ share one and only one point p, and such that there is no constraint in $\mathcal{S}$ concerning both one geometric entity of $\mathcal{S}_1$ and one of $\mathcal{S}_2$, except p. $\mathcal{S}$ is called a rotoid assembly of $\mathcal{S}_1$ and $\mathcal{S}_2$. We similarly define a sliding

(resp. positioning) assembly as a the union of two subsystems with only a line (resp. nothing) in common. □

Definition 4: Simple and complex assemblies
Let $\mathcal{S}$ be a rotoid, sliding or positioning assembly of two subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$. If both $\mathcal{S}_1$ and $\mathcal{S}_2$ are well-constrained *modulo* rigid motions, $\mathcal{S}$ is called a simple assembly. Otherwise, it is called a complex assembly. □

The assembly transformations are defined as follows (the notation $f|_{\mathcal{S}}$ stands for the restriction of figure f to the subsystem $\mathcal{S}$).

Definition 5: Assembly transformations
Let $\mathcal{S}$ be a rotoid assembly of two subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$ with p their common point, and let f_a and f_b be two solutions of $\mathcal{S}$. A rotoid transformation of f_a into f_b consists in the composition of

- a rigid motion of f_a with result f'_a such that values of point p in f'_a and f_b are the same
- a rotation of $f'_a|_{\mathcal{S}_1}$ around the common point p such that $f'_a|_{\mathcal{S}_1}$ and $f_b|_{\mathcal{S}_1}$ coincide
- a rotation of $f'_a|_{\mathcal{S}_2}$ around the common point p such that $f'_a|_{\mathcal{S}_2}$ and $f_b|_{\mathcal{S}_2}$ coincide

We similarly define a sliding transformation as a rigid motion and two translations along the common line, and a positioning transformation as two rigid motions. □

This last definition implies that $f_a|_{\mathcal{S}_i}$ and $f_b|_{\mathcal{S}_i}$, $i \in [1,2]$, are superposable, thus requiring prior assembly transformations on $\mathcal{S}_1$ or $\mathcal{S}_2$ if those subsystems are themselves simple or complex assemblies. Notice that the existence of complex assemblies implies that there is not always a single invariance group for an assembly.

Using definitions 2 and 3 is not sufficient to describe all GCS built on our geometric universe because of the *closed chains*. For simplification purpose, we consider a closed chain as a simple or complex assembly with one or more *closure conditions*. This allows us to consider only binary assemblies, and considerably simplify the data structures.

Definition 6: Closure condition
Let $\mathcal{S}$ be a simple or a complex assembly and $c \notin \mathcal{S}$ a constraint linking an entity of each of the two assembled systems. c is a closure condition of $\mathcal{S}$ if $\mathcal{S}+c$ is not rigid and cannot be decomposed in two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}+c$ is a simple or a complex assembly of $\mathcal{S}_1$ and $\mathcal{S}_2$. □

3.2.2 G-references

A *reference* for a G-system $\mathcal{S}$ (*i.e.* well-constrained *modulo* G) is a GCS R, whose entities are all in $\mathcal{S}$, such that $\mathcal{S} \cup R$ has one and only one solution (resp. a finite number) for the transformation to act simply (resp. finitely) transitively. In the case of a GCS well-constrained *modulo* the rigid motions, a reference could for instance be a point and a line to get a finite number (maximum two) solutions, together with an orientation to get a single solution.

For simple rotoid assemblies, a reference could be the common point of the two subsystems, and a line in each subsystem. For simple sliding assemblies, a reference could be two points on the common line, one in each of the two subsystems. For positioning assemblies, a reference consists in a reference for each of the two subsystems.

The notion of reference becomes recursive with complex assemblies: for instance, a reference for a complex rotoid assembly would be the common point, a line in each of the subsystems, and to this one adds the entities of the subsystems that lack to have references for the subsystems.

We introduce the notion of *relative reference.*

Definition 7: Relative reference
Let G_1 and G_2 be two transformations groups and $\mathcal{S}$ a G_1-system. A GCS generated by a constraint c is a reference for G_1 relatively to G_2 if $\mathcal{S}+c$ is a G_2-system. □

For instance, a distance between the two extremities of the scissors is a reference for the rotoid transformations group of the pair of scissors relatively to the rigid motions group.

3.2.3 Border of an assembly

The usual definition of the border is the following:

Definition 8: Border of a rigid system
Let $\mathcal{S}$ be a rigid GCS and $\mathcal{H}$ any GCS. The border of $\mathcal{S}$ relatively to $\mathcal{H}$ is the system $B = (C, X, A)$ with $X = X_{\mathcal{S}} \cap X_{\mathcal{H}}$, C the set of all distances between two points and angles between two lines in X and A the set of parameters appearing in C, with values being computed from $\mathcal{S}$. □

This definition does not hold for assemblies. We extended it with a recursion ending at the rigid subsystems.

Definition 9: Border of an assembly
Let $\mathcal{S}$ be an assembly of two subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$ and $\mathcal{H}$ any system. The border of $\mathcal{S}$ relatively to $\mathcal{H}$ is the union of the borders of $\mathcal{S}_1$ and $\mathcal{S}_2$ relatively to $\mathcal{H}$. $\square$

4 An incremental assembly algorithm

Based on the definitions discussed in section 3, we established algorithms to find an invariance group of a GCS and deduce a construction plan. We detail them in this section. These algorithms were made with a will of generality, enabling an extension of the geometric universe and of the considered invariance groups without too much adaptation.

4.1 Obtaining the describing terms

The basic idea is to consider the constraints of the GCS one by one, and compute an invariance group of the system after the addition of each of them. We must have a list of the terms (in the sense of term logic) describing the system generated by one constraint

The global algorithm consumes a constraint, finds the corresponding term, and assembles the generated system with the already constructed system. The functions used in algorithm 1 are the followings:

Alg. 1 Term describing a GCS

```
Require: $\mathcal{S}$: a GCS
  Term $\mathcal{T}$ := EmptyTerm()
  for all $c \in$ Constraints($\mathcal{S}$) do
    $\mathcal{T}$ := Assemble($\mathcal{T}$, Catalog($c$))
  end for
  $\mathcal{T}$ := AddEntities($\mathcal{T}$,$\mathcal{S}$)
  return $\mathcal{T}$
```

- `EmptyTerm`: returns the term describing an empty GCS
- `Constraints`: returns the set of constraints of a GCS
- `Catalog`: returns the term describing the system generated by a constraint
- `Assemble`: returns the term describing the union of any two systems
- `AddEntities`: creates positioning assemblies for geometric entities that were not linked by any constraint

We so have an incremental strategy. Notice that the algorithm is extremely general, and may apply to any geometric universe and any set of invariance groups, as long as one has the `Catalog` and `Assemble` functions.

The output of algorithm 1 is a term describing the assembly. If it determines that the system is an assembly of two systems $\mathcal{S}_1$ and $\mathcal{S}_2$, the output will be $\mathcal{A}(\mathcal{S}_1, \mathcal{S}_2)$, $\mathcal{A}$ being the kind of assembly: A_R for a rotoid assembly, A_S and A_P for a sliding and a positioning assembly. Knowing the kind of assembly means knowing an invariance group.

4.2 Assembling two systems

The function `Propagation`($\mathcal{T}_1, \mathcal{T}_2$) is the core function of the method: it computes the border of $\mathcal{T}_2$ relatively to $\mathcal{T}_1$ and computes which changes are made to $\mathcal{T}_1$ by this border. That is, it finds the parts of $\mathcal{T}_1$ that become rigid by the adjunction of $\mathcal{T}_2$.

Function `DetermineAssembly` determines the assembly to construct: after the propagation, the new rigid parts are incorporated to the first system. One just has to see what entities the first system now has in common with the rest of the second system: if they have one point in common, a rotoid assembly is constructed, for instance.

For means of generality and extendability, the calls to `Propagation` are made in a loop, so as to ensure that the modifications made to one of the two systems are taken into account in the second system. Function `Assemble` then works as follows:

In our geometric universe, the loop is not useful, but one can easily imagine more complex constraints leading to the necessity of several propagations and back-propagations. Moreover, this

Alg. 2 Term describing the assembly of two systems

```
Require: $\mathcal{T}_1, \mathcal{T}_2$: terms describing the systems to assemble
  Term $\text{buf}_1 := \mathcal{T}_1$, $\text{buf}_2 := \mathcal{T}_2$
  while $\text{buf}_1 = \mathcal{T}_1$ or $\text{buf}_2 = \mathcal{T}_2$ do
    $\text{buf}_1 := \mathcal{T}_1, \text{buf}_2 := \mathcal{T}_2$
    $\mathcal{T}_1 :=$ Propagation($\mathcal{T}_2, \mathcal{T}_1$)
    $\mathcal{T}_2 :=$ Propagation($\mathcal{T}_1, \mathcal{T}_2$)
  end while
  return DetermineAssembly($\mathcal{T}_1, \mathcal{T}_2$)
```

approach makes algorithm 2 usable with any two systems, enabling us to give up the incremental strategy if needed.

4.3 Closed chains

The propagation method we have been using so far does not always manage to determine the rigidity of a system after adding a constraint. The method actually works when there exists a subsystem which is a simple assembly and when the added constraint is a relative reference for this simple assembly relatively to the rigid motions group.

Otherwise, the propagation method does only consider the new constraint as a closure condition. There are different ways to solve this problem: one can compute the *deficit* [8] of the closed chain, and if it is zero, rewrite the term to show that the closed chain actually is rigid; one may also use a more complex geometric reasoning, such as a *loci* method [2].

4.4 Retrieval of a construction plan

The algorithms explained above are used to obtain a term describing a GCS, letting the user know an invariance group of the system. The next step is to find the particular solutions from which all other solutions can be obtained by application of the transformations.

We keep the history of the term construction and of the rewritings due to the detection of relative references. Computing the assembly transformation one should apply to a simple assembly so that it satisfies a relative reference is trivial. Knowing the history of the term construction allows us to build a construction plan based on these transformations.

This works as long as the `Propagation` function manages to determine the rigidity of a closed chain. Otherwise, we need to use an external solver to get the valid solutions of the rigid subsystem.

4.5 Example of use

Let us incrementally build the term describing the GCS of figure 1, with one possible order of constraints consideration. Some steps are shown on figure 3. Step a shows a rotoid assembly constructed after the consideration of two distance constraints. The addition of an angle constraint leads, through the `Propagation` function, to the rewriting of this assembly: the angle is a relative reference, and the resulting system is rigid. At step b, three other distance constraints and one angle have been consumed, resulting in a complex rotoid assembly containing three rigid subsystems. The addition of the distance constraint leads to the creation of a non rigid closed chain: this constraint is a closure condition. When, at step c, the last angle is added, the rigidification of the bottom "triangle" is detected, and the changes are propagated, leading to the rigidification of the whole system.

The fourth image of figure 3 (d) shows a closed chain made of a complex assembly of three rigid bars and a closure condition (the fourth distance). The addition of the angle constraint forces our method to use an external solver, because it is not a relative reference for an assembly.

5 Applications

In this section, we show the application field and the possibilities granted by our approach and by our algorithms.

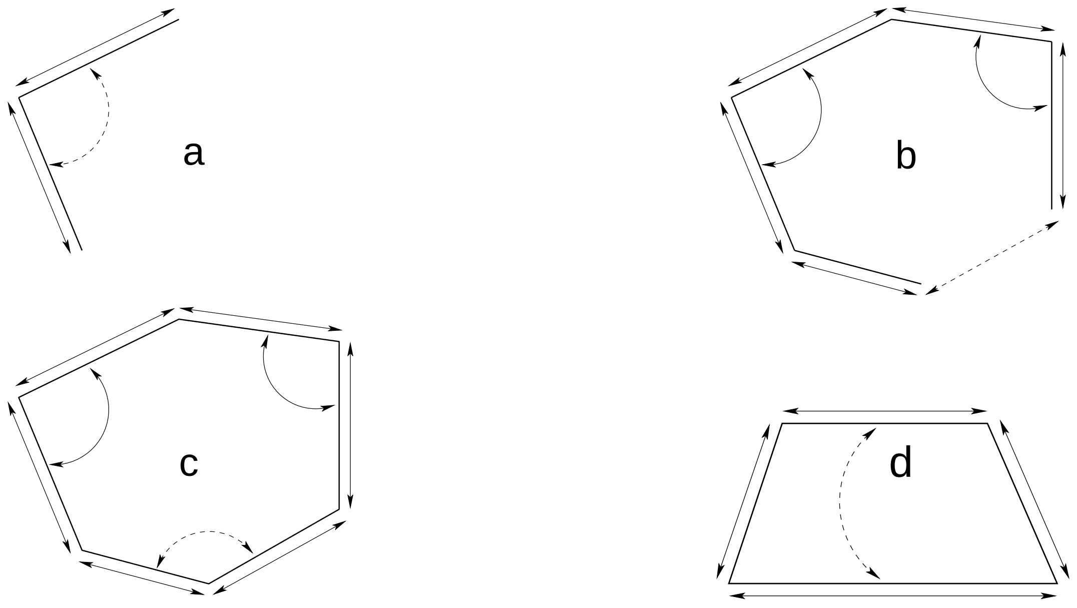

Figure 3: Construction steps of figure 1 (a, b, c) and a problematic rigid GCS (d)

5.1 Homogeneous consideration of GCS

The consideration of invariance groups and the generality of our algorithms leads to an homogoneous capture of the systems to be solved, without a prior sort of under-constrained and well-constrained systems: no hypothesis is made on the GCS. Under-constrained and well-constrained (in the classical sense) systems are handled through the same process, without adding any constraint.

Such an approach opens the way to softwares where the user sketches a GCS and gets the invariance group of the system and the particular solutions from which all other solutions can be obtained by transformations of the invariance group.

The knowledge of the invariance group allows the software to build animations showing the user, to some extent, how the system he sketched can move. The user can then see if his GCS is articulated, scalable, rigid, *etc.*.

The user can then "ask a question" about the possibilities of the solutions by adding a new constraint (for instance: are there solutions where the distance between points A and B is x ?"). Our incremental strategy allows us to answer the question without starting the solving process from scratch, using the term describing the initial system.

5.2 Multi-solver strategy

Our method alone does not solve more geometric constraints problems than other existing solvers, it mainly brings some under-constrained systems to the field of the solved systems. By integrating it within a multi-solver strategy, we also ensure we don't solve less than other solvers. This approach enables optimal handling of the user's sketch: should he specify from the beginning that he intends his system to be well-constrained *modulo* the rigid motions, our strategy would be to use a classical decomposition algorithm, calling the algorithms of section 4 only if it came to fail because of an under-constriction *modulo* the rigid motions.

Moreover, a multi-solver approach allows us to give up the incremental strategy when it does not seem to fit our needs. One could for instance prefer the use of a classical decomposition method to extract all greatest rigid subsystems, before finally using the `Assemble` function to assemble them if there is more than one. Should the user be unsatisfied and ask for a rigid solution, this approach enables us to add relative references or to use another completion algorithm, like [8].

6 Conclusion and perspectives

Under-constrained systems form a problematic family in the field of geometric constraints solving. In this paper, we presented an approach to homogeneously handle under- and well-constrained systems. Our method is based on a generalization of the notion of well-constriction, considering invariance groups. We exposed the assembly transformations, specific to the system to be solved. A class of problems, usually considered as under-constrained, can now be seen as well-constrained *modulo* assembly transformations. We then built extendable algorithms intended to obtain the description of a geometric constraints system as a term together with one of its invariance groups.

This approach appears promising to us. Indeed, in order to use our method with any under-constrained system, one only has to find the appropriate transformations groups and their relative references, so as to get the particular solutions from which the whole solution space can be generated. Extension of the considered groups and generalization of the transformration groups are the next steps of our work.

Moreover, we think that this approach may lead to a new characterization of rigidity in 3D. The famous double-banana would then be well-constrained *modulo* the rotations around the line passing through the two common points.

References

[1] J. Alba, M. Doblaré, and L. Gracia. A simple method for the synthesis of 2D and 3D mechanisms with kinematic constraints. *Mech. Mach. Theory*, 35(5):645–674, 2000.

[2] X.-S. Gao, C. Hoffmann, and W.-Q. Yang. Solving spatial basic geometric constraint configurations with locus intersection. In *ACM SMA'02*, pages 95–104.

[3] X.-S. Gao and G.-F. Zhang. Well-constrained completion for under-constrained problems. *IJCGA*, 16, 2006.

[4] C. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *CAD&A*, 2005.

[5] C. Hoffmann, A. Lomonosov, and M. Sitharam. Geometric constraint decomposition. In *Geometric Constraint Solving and Applications*, pages 170–195. Springer, 1998.

[6] C. Jermann, B. Neveu, and G. Trombettoni. A new structural rigidity for geometric constraints systems. In *ADG'02*, pages 87–105.

[7] C. Jermann, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: a survey. *IJCGA*, 16, 2006.

[8] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pasto. Revisiting decomposition analysis of geometric constraint graphs. *Computer-Aided Design*, 36:123–140, 2002.

[9] G. Kramer. Using degrees of freedom analysis to solve geometric constraint systems. In *ACM symp. on Solid Modeling CAD/CAM*, pages 371–378, 1991.

[10] R. Latham and A. Middleditch. Connectivity analysis : a tool for processing geometric constraints. *Computer-Aided Design*, 28(11):917–928, 1996.

[11] Y.-T. Li, S.-M. Hu, and J. Sun. A constructive approach to solving 3D geometric constraint systems using dependence analysis. *Computer-Aided Design*, 34:97–108, 2002.

[12] D. Michelucci and S. Foufou. Geometric constraints solving: the witness configuration method. *Computer-Aided Design*, 38:284–299, 2006.

[13] P. Schreck and P. Mathis. Geometrical constraint system decomposition: a multi-group approach. *IJCGA*, 16, 2006.

[14] M. Sitharam and Y. Zhou. A tractable, approximate, combinatorial 3D rigidity characterization. In *ADG'04*.