International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

GPDOF: A Fast Algorithm to Decompose Under-constrained Geometric Constraint Systems: Application to 3D Modeling

Gilles Trombettoni[1] Marta Wilczkowiak[2]
[1] *COPRIN Project, I3S-INRIA, University of Nice–Sophia Antipolis*
2004 route des lucioles, 06902 Sophia Antipolis cedex, B.P. 93, France
trombe@sophia.inria.fr

[2] *MOVI Project, INRIA Rhône-Alpes,*
655 avenue de l'Europe, Montbonnot, 38334 Saint Ismier cedex, France
Marta.Wilczkowiak@gmail.com[*]

Received (received date)
Revised (revised date)
Communicated by (Name)

Our approach exploits a general-purpose decomposition algorithm, called GPDOF, and a dictionary of very efficient solving procedures, called r-methods, based on theorems of geometry. GPDOF decomposes an equation system into a sequence of small subsystems solved by r-methods, and produces a set of input parameters.[1]

Recursive assembly methods (decomposition-recombination), maximum matching based algorithms, and other famous propagation schema are not well-suited or cannot be easily extended to tackle geometric constraint systems that are under-constrained. In this paper, we show experimentally that, provided that redundant constraints have been removed from the system, GPDOF can quickly decompose large under-constrained systems of geometrical constraints.

We have validated our approach by reconstructing, from images, 3D models of buildings using interactively introduced geometrical constraints. Models satisfying the set of linear, bilinear and quadratic geometric constraints are optimized to fit the image information. Our models contain several hundreds of equations. The constraint system is decomposed in a few seconds, and can then be solved in hundredths of second.

Keywords: geometric constraints; decomposition; computer vision

1. Introduction

Solving a set of geometric constraints is a challenging problem in parametric Computer Aided Design. Because of the intrinsic complexity of the corresponding equation systems, several researchers have followed the "divide and conquer" paradigm. Several rule-based or graph-based decomposition algorithms have thus been designed to split the constraint system into several subsystems that can be solved more efficiently.[2,3,4,5,6,7,8,9]

[*]M. Wilczkowiak currently works at the Mathworks Ltd, Cowley Road, Cambridge, CB4 0HH, UK

Although some difficulties still need to be overcome, these methods behave rather well on well-constrained or rigid problems[a]. It turns out that only little has been done to treat under-constrained problems that have an infinite set of solutions (even modulo the group of direct isometries). However, it would allow a designer to build its mechanism or to draw its figure in an incremental way by adding geometric objects and constraints one by one.

In this paper, we are interested in specific geometric constraint systems for which an initial value is known for all (or most of) the variables involved in the objects. The values are often known approximately and may of course not satisfy "exactly" the constraints. Three main types of problems fall in this category:

- *Solution maintenance:* a figure is corrected and completed incrementally on the screen, but all the already drawn objects have a current, and not definite, position and orientation.
- *Free-hand drawing using a sketch:* the values given by the sketch are used by the constraint solver to draw the final figure.
- *3D model reconstruction from images using geometric constraints* (2D images along with additional constraints on the 3D scene are given to the software): before the constraints are solved, initial point values are computed by a standard optimization algorithm based on images.

When an initial value is available for the variables, the decomposition problem can be tackled by:

- Selecting a set of variables and transforming them into *input parameters* (i.e., constants): selecting them makes the remaining system well-constrained. *The values given to the input parameters, combined with the geometric constraints, fully fix all the degrees of freedom and completely describe the figure.*
- Decomposing the constraint system into subsystems and solving the decomposed system in a given order. In the end, new values are computed for all the variables (except the input parameters) so that all the constraints are satisfied.

Note that this approach can be viewed as a specific way to add non-user defined constraints in order to make the system well-constrained. Indeed, it amounts to adding a unary constraint on every input parameter in order to fix its value (e.g., using the sketch).

It seems that no existing decomposition method is really well-suited to handle under-constrained systems.

Existing propagation mechanisms cannot guarantee finding a correct order between subsystems without a backtracking step, i.e., in a polynomial time. The propagation of the *known states* (see Ref. [10]) and the *reactive* propagation algo-

[a]A rigid system is well-constrained modulo the group of direct isometries.

rithm (see Refs. [11,12]) become exponential when the system is under-constrained (see Ref. [13], pages 64 and 70, Ref. [14], page 172, and Ref. [15]).

Geometric decomposition methods

Bottom-up decomposition methods, also known as recursive assembly methods, reduction analysis or decomposition-recombination methods, use recursively a procedure to identify a rigid subsystem in the whole system.[6,7,8,9] The different corresponding rule-based or graph-based identification procedures have a common point: they are designed for identifying a well-constrained subsystem (modulo the group of direct isometries). As pointed in Ref. [6], when the system is globally under-rigid, they are intrinsically unable to assemble the different rigid subparts together.

Top-down methods, also known as decomposition analysis or recursive division methods, exhibit recursively certain vertices in the constraint graph for splitting a given system in (potentially) rigid subparts.[6,16] These methods can tackle under-rigid systems and "complete" them to make them well-constrained, by adding automatically non user-defined constraints. However, the vertex selection, as well as the constraint completion, are limited to geometric constraint systems in 2D made of specific sets of objects and constraints.

Equational decomposition methods

Other approaches apply the famous Maximum-matching graph algorithm on a dependency graph between equations and variables.[17,18,19] When the system is well-constrained (i.e, no degree of freedom must be fixed), there is a unique decomposition into subsystems, i.e., into strongly connected components.[20] When the system is rigid in 2D, it is yet tractable to fix the 3 remaining degrees of freedom in a coordinate system (all the possible coordinate systems must be tried in a combinatorial way). In this case, the approach is similar to certain graph-based recursive rigidification methods.[7]

When the system is under-rigid, the combinatorial process above is not tractable. Instead, following any maximum matching of the system, Dulmage and Mendelsohn's decomposition allows us to determine a structurally under-constrained part in which input parameters can be extracted.[21,19,22] Unfortunately, this structural analysis has serious drawbacks:

- Different decompositions can be obtained according to the computed matching. The decompositions differ in the set of input parameters (and thus in the actual equation system that is handled), and in the computed subsystems.
- Certain decompositions may contain large subsystems (in terms of number of equations), while others contain smaller subsystems[b]. It appears that

[b]For instance, all the equations of the didactic example below could be put into a unique subsystem if the variables c_{la}, c_{la} and y_{pb} are chosen as input parameters.

finding the decomposition of a structurally under-constrained system minimizing the size of the largest subsystem is the dual of the *minimum dense* problem that has been proven to be NP-hard.[23]

- Certain decompositions are geometrically correct while others are incorrect, that is, contain subsystems with redundant or contradictory equations.[1,14]
- No specific solving procedure exists for solving the subsystems, so that a generic solver must be called to solve the involved equations.

To sum up, faced to an under-constrained system, Maximum-matching may select subsystems that are very large or geometrically incorrect, and no fast procedure is known for solving the subsystems.

Using GPDOF

On the contrary, our `GPDOF` algorithm is particularly well-suited to select input parameters and to compute a decomposition of an under-constrained system. Indeed, the identified subsystems belong to a set of predefined patterns for which a fast solving procedure is known. Also, the identified subsystems are geometrically correct: they contain neither redundant nor contradictory equations in the generic case.

The procedures used to solve the subsystems are called *r-methods* (resolution methods). They are based on theorems of geometry and incorporated into a *dictionary*. The dictionary allows the general-purpose equational algorithm `GPDOF` to make a bridge with the geometric level. Provided that the system contains no redundant equation, `GPDOF` produces a set of input parameters and a sequence of r-methods present in the dictionary in polynomial time (if any).

`GPDOF` (see Ref. [1]) is a generalization of the PDOF local propagation algorithm.[24,10,25] In those approaches, constraints are selected and solved one by one (i.e., the subsystems contain only one equation). Extensions to geometric constraints select subsystems that include several equations but place exactly one object.[26,27,28] `General-PDOF` (`GPDOF`) subsumes all these approaches by selecting r-methods of any type: r-methods solving one equation (like in the standard PDOF), r-methods solving several equations used to compute one object and, potentially, r-methods computing several objects simultaneously.

We have validated `GPDOF` in 3D model reconstruction, an important field in computer vision. Constraints are incorporated into the 3D model acquisition system. The user can select in images objects such as points, lines or planes and can define geometrical dependencies between them, such as parallelism, orthogonality, and distance constraints. Then a model satisfying those constraints, conforming to the image information (by minimization of the reprojection error) is computed. Our system has been applied to two architectural models including several hundreds of constraints. Most of them are bilinear, some of them are linear or quadratic (distance constraints). In both models, the preprocessing step is performed by `GPDOF` in a few seconds, and then all the constraints are solved in hundredths of second.

Contribution and limits

As explained above, GPDOF is of no particular interest for tackling well-constrained or well-rigid geometric constraint systems because Maximum-matching can be used instead. On the contrary, GPDOF is well-suited for under-rigid constraint systems because it selects input parameters and quickly solves a sequence of geometrically correct subsystems.

GPDOF overcomes the main known drawback of local propagation algorithms: the presence of loops in the constraint graph. Indeed, GPDOF selects "general" r-methods that solve any type of equation subsystem that may include cycles.

Also, GPDOF works at two different levels and takes the best of both. Indeed, some r-methods may correspond to theorems of geometry while others may be defined at the variable/equation level. However, in the end, all the r-methods are translated into hyper-edges in the equation graph, that is, at the variable/equation level. Thus, constraint and equation graphs are both taken into account while losing no semantic (i.e., geometric) information. This allows GPDOF to tackle systems made of geometric and non geometric constraints.

Finally, the work presented in this article is the first realistic application of GPDOF. In particular, it describes for the first time a pre-processing phase (called automatic r-method addition phase in the article) which is needed by GPDOF in practice. The implementation experimentally shows that GPDOF and the pre-processing phase constitute a rule-based decomposition method with impressive performance on large under-constrained systems.

Section 6.5 underlines the main limit of GPDOF: the redundant constraints and over-constrained subparts must be removed before running the algorithm.

Also, it is important to understand that geometric (in particular, top-down) decomposition methods and equational decomposition approaches behave differently on under-constrained geometric systems, and in a sense do not tackle the same types of applications. Indeed:

- Top-down approaches must "complete" the system with additional constraints such that the whole construction is rigid (i.e., with 6 degrees of freedom in 3D).
- Equational approaches, such as Maximum-matching and GPDOF, ignore this rigidity property and extract a set of input parameters. That is why they are well-suited to tackle the applications mentioned above: solution maintenance, free-hand drawing using a sketch, and the 3D model reconstruction described in this article.

Section 6.6 illustrates this difference and highlights why geometric decomposition methods can generally better decompose a geometric system while GPDOF is a general-purpose algorithm able to handle geometric and non geometric equations.

Outline

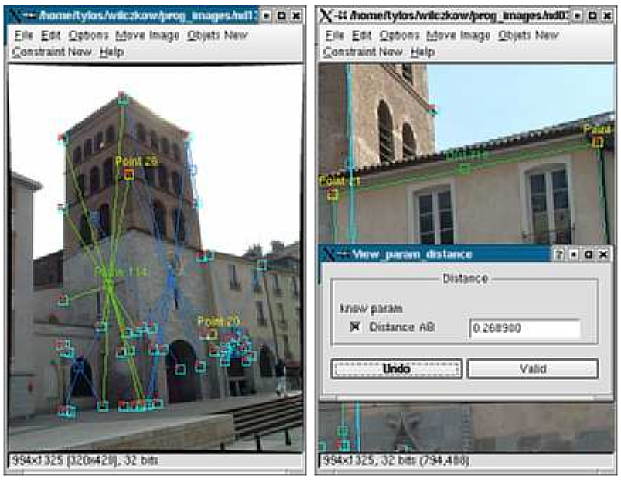
Section 2 presents an overview of our method. Section 3 introduces a scene in 2D illustrating the main algorithms used by our system. Section 4 details how objects and constraints are modeled, and includes the background necessary to understand the constraint satisfaction part. Section 5 details the automatic r-method addition phase which is a preliminary step for the use of `GPDOF` described in Section 6. The optimization phase is described in Section 7. Section 8 shows the experiments we have performed on two models of a church. Section 9 describes the related work in computer vision.

2. Overview of the approach used in 3D model reconstruction

Our 3D model acquisition makes use of geometric constraints. It is divided into three main phases: initialization, constraint planning and optimization.

Initialization

In addition to 2D images and feature projections matched between images, geometric objects and constraints between them must be defined as input to create the 3D model. The model is represented here by *points*, *lines* and *planes*. They are subject to linear, bilinear and quadratic constraints such as *distance*, *incidence*, *parallelism* and *orthogonality*. All the objects and constraints are introduced using a graphical user interface (see Figure 1). The cameras are then calibrated using the


File Edit Options Move Image Objets New
Constraint New Help
Point 26
Point 20
View_param_distance
Distance
know param
Undo
Valid
994x1325 (320x428), 32 bits
994x1325, 32 bits (794,488)

☒ Distance AB 0.268980

(a) (b)

Fig. 1. Interface used for the scene definition. (a) Definition of points and planes in one of the images belonging to the Notre Dame sequence. (b) Definition of a distance constraint between two points.

linear method described in Ref. [29]. An initial reconstruction is provided by a multilinear approach exploiting projections and geometric constraints combined with an unconstrained bundle adjustment.[30] However, any other approach for calibration and reconstruction could be used here. *Note that after this phase all the variables (camera and model parameters) have an initial value.*

Constraint planning

The goal of the constraint planning phase is to transform a model defined by constraints among objects into a parametric model. Thus, the output of this step is composed of:

- *Input parameters*: ideally, the number of input parameters is equal to the number of degrees of freedom of the model $n_m = n_o - n_c$, where n_o is the sum of degrees of freedom of the model objects (i.e., the number of involved variables) and n_c is the sum of degrees of freedom of the model constraints (i.e., the number of independent equations).
- *A plan*: a sequence of routines (called *r-methods*) which, given values assigned to the input parameters, computes the coordinates of all the scene objects in a way that all the inter-object relations are satisfied.

The model reconstruction system we propose requires an input set of *r-methods* which allow us to decompose the whole equation system into small subsystems. An r-method is a hard-coded procedure used to solve here a subset of geometric constraints.[1,13] An r-method computes the coordinates of *output objects* based on the current value of *input object* coordinates with respect to the underlying constraints between input and output objects.

An example is an r-method that computes the parameters of a line based on the current position of two points incident to this line. Another example is an r-method that computes the position of a 3D point A located at known distances from three other points B, C, D. This r-method computes the (at most) two possible positions for A by intersecting the three spheres centered respectively in B, C, and D. Sixty r-method patterns have been incorporated in a dictionary used by our system. They correspond to ruler-and-compass routines used in geometry or, more generally, to standard theorems of geometry.

The geometric constraint system and the corresponding algebraic equations are represented by graphs. The *constraint graph* yields the dependencies between constraints and objects. The *equation graph* yields the dependencies between equations and variables. Based on these graphs, the constraint planning uses two algorithms:

(1) *R-method addition phase:* Add *automatically* in the equation graph all the r-methods corresponding to r-method patterns present in the dictionary. For instance, the r-method pattern “line incident to two known points” may occur a lot of times in the system. Every time two points incident to a line are recognized in the constraint graph, a corresponding r-method is added to the equation graph (see example below).
This phase thus produces an equation graph “enriched” with r-methods.

(2) *Planning phase:* Perform GPDOF on the enriched equation graph. GPDOF produces:
   - a set of input parameters, that is, a subset of the variables describing the scene such that, when a value is given to them, there exists a finite set of solutions for the rest of the system satisfying the constraints;
   - a sequence of r-methods (called *plan*) to be executed one by one.

Model optimization

The model is refined by an unconstrained optimization process run over the input parameters only. The optimization produces values for all the model variables so that all the constraints are satisfied (exactly) and the sum of reprojection errors is minimal. At each iteration of the optimization algorithm, the input parameter values are modified and the plan is executed, resulting in new coordinate values for all the other scene objects. The cost function is then computed as the reprojection error of all the points (and possibly lines).

3. Example of constraint planning

To illustrate the algorithms presented in this article, we will take a small example describing a parallelogram in 2D in terms of lines, points, incidence constraints and parallelism constraints (see Figure 2). Of course, the scenes we handle with our tool are in 3D, and this example is just presented for didactic reasons. Figure 2 also shows the bipartite constraint graph containing four points P_a,...,P_d with coordinates $(x_{pa}, y_{pa}), ..., (x_{pd}, y_{pd})$, four lines L_a,...,L_d with coordinates $(a_{la}, b_{la}, c_{la}), ..., (a_{ld}, b_{ld}, c_{ld})$, eight incidence constraints C_1,...,C_8 and two parallelism constraints C_9, C_{10}.

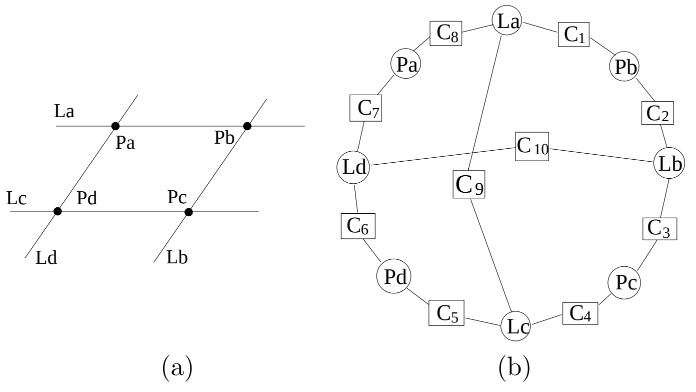
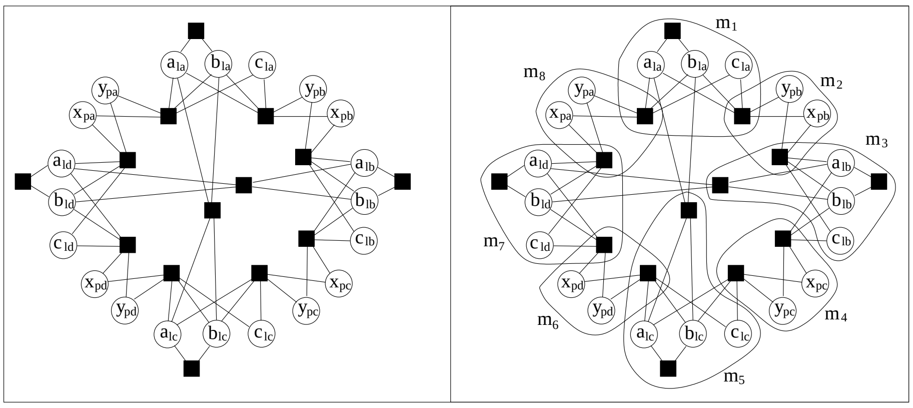


Fig. 2. A didactic example of a 2D scene (a) and the corresponding constraint graph (b).

The equation graph corresponding to the 2D scene is shown in Figure 3-left. The right side of Figure 3 shows the equation graph enriched with a set of r-methods that can be used to solve subparts of the system. An r-method is represented by a hyper-arc including its equations and its output variables. These r-methods belong to one of the three following categories (these patterns could appear in a dictionary of 2D r-methods):

- line incident to two points (e.g., r-methods m_1 and m_7);
- point at the intersection of two known lines (e.g., m_2, m_4, m_6, m_8);
- line passing through a known point and parallel to another line (e.g., m_3, m_5).



(a) (b)

Fig. 3. (a) The equation graph of the 2D scene. Variables are represented by circles and equations are represented by black rectangles. An equation involving only the a and b coordinates of a line has the form $a^2 + b^2 = 1$ to allow a unique representation of a line. (b) The enriched equation graph. Only 8 of the 16 possible r-methods are depicted for the sake of clarity.

The GPDOF algorithm, presented in Section 6, works on the enriched equation graph. It is able to select for example the coordinates of P_a, P_b and P_d as a set of input parameters. It also produces a plan, e.g., the sequence of r-methods (m_1, m_7, m_3, m_5, m_4), whose execution results in new coordinate values for objects L_a, L_d, L_b, L_c, P_c respectively. Figure 4 illustrates an execution of this plan.

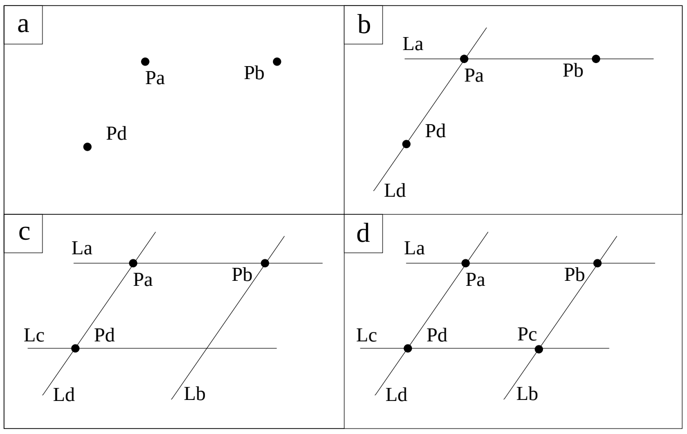


Fig. 4. Execution of r-methods in the plan (m_1, m_7, m_3, m_5, m_4). (a) The input parameters (i.e., coordinates of P_a, P_b and P_d) are replaced by their current value. (b) m_1 and m_7 place lines L_a and L_d resp. (c) m_3 and m_5 place L_b and L_c resp. (d) Finally, m_4 places point P_c.

4. Scene Modeling and Background

4.1. *Geometric objects*

In the current implementation of the system, available objects are points, lines and planes. Their representation has a significant impact on the performance of the system. It is necessary to use a representation that corresponds to the number of degrees of freedom characterizing the objects and allows an efficient computation. We briefly detail how the objects are represented in the system.

Points: the 3 degrees of freedom (dof) of a point are represented by 3 variables x, y, z.

Lines: the 4 degrees of freedom of a line are represented by 6 variables (called Plücker coordinates) and 2 internal relations. The coordinates correspond to the line direction vector $\mathbf{d}$ and the vector $\mathbf{n}$ normal to the plane passing through the line and the coordinate system origin. The line coordinates are related by two conditions: the directional vector is constrained to be of unit length and vectors $\mathbf{n}$ and $\mathbf{d}$ are constrained to be perpendicular. See Ref. [31] for details about the Plücker line representation.

Planes: the 3 degrees of freedom of a plane are represented by 4 variables and 1 internal relation. The coordinates correspond to the plane normal $\mathbf{n}$ and its distance d from the origin. The normal vector $\mathbf{n}$ is constrained to be of unit length.

Note that the solving process described in this article can also support other parameterizations and other types of primitives. Also note that, in the computer vision application, the final model is represented only by a set of points, that is, lines and planes are viewed as collinearity and coplanarity constraints between points.

4.2. *Geometric constraints*

Constraints are also characterized by a number of degrees of freedom they fix on the involved objects. In the current implementation, we consider the following constraints:

- *distance:* point-point (1 dof), point-line (1 dof), point-plane (1 dof),
- *incidence:* point-line (2 dofs), point-plane (1 dof), line-plane (2 dofs),
- *parallelism:* line-line (2 dofs), line-plane (1 dof), plane-plane (2 dofs),
- *orthogonality:* line-line (1 dof), line-plane (2 dofs), plane-plane (1 dof).

Any other constraint which can be expressed as equations in terms of objects coordinates, like angles, distance and angle ratios can be incorporated.

4.3. *R-methods*

Our model reconstruction system is based on a dictionary containing an input set M of *r-methods.*

An r-method is a routine executed to satisfy a subset E_m of equations in E by calculating values for its *output variables* as a function of the other variables implied in the equations. Two examples of r-methods are shown in Figure 5.

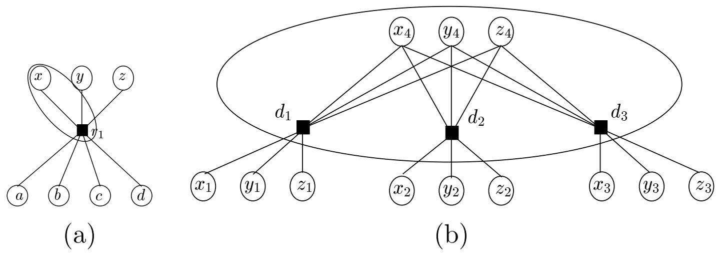


Fig. 5. Representation of r-methods by hyper-arcs that include the satisfied equations and the output variables. (a) An r-method fixing the remaining degrees of freedom of a point incident to a plane. (b) An r-method computing the position of a point using 3 point-point distance constraints.

Definition 1. An **r-method** m in M is a function over a set of **input variables** I. The variables involved in the equations $E_m \subset E$ are divided into a non-empty set of **output variables** $O \subset V$, and a set of input variables $I \subset V$.

Given a set of values $\overline{I}$, the r-method m yields the set of solutions for O satisfying E_m. This operation is called **execution** of the r-method m.

The r-method m is **free** if no variable v in O is involved in a constraint in $E \setminus E_m$. Thus, executing a free method cannot violate other equations in $E \setminus E_m$.

Note that a given equation can generally be solved by several r-methods. For instance, 3 r-methods (computing a value for x, y, *or* z) could be used to solve the point-plan incidence shown in Fig. 5(a)). This makes combinatorial the problem of computing a sequence of r-methods to solve all the equations.

The current dictionary of our system contains 60 r-methods. These r-methods use only 45 different execution procedures. For instance, plane-plane parallelisms and line-line parallelisms are solved by the same procedure.

The dictionary includes all the r-methods that solve constraints by computing (output) parameters of *one* object. More complicated r-methods computing more than one object at a time can be envisaged. The algorithms used in our system can deal with any type of r-method, although the time complexity will grow with the number of constraints involved in r-methods (see Section 8.2). Details on the design and implementation of r-methods are given in Ref. [14].

An **r-method pattern** present in the dictionary is a generic constraint graph corresponding to the equations solved by the r-method. We design this constraint graph by a pattern because a similar constraint graph (pattern) may occur several times in the actual constraint graph corresponding to the model, thus leading to the creation of several similar r-methods. For instance, the r-method pattern “line incident to two known points” is a graph made of 4 vertices (3 objects and 1 constraint). Every time two points incident to a line are recognized in the constraint graph (as a subgraph), a corresponding r-method is added to the equation graph. Thus, an r-method in the dictionary is defined at both levels: geometric one (i.e., a pattern made of objects and constraints) and equational one (i.e., a hard-coded procedure solving the corresponding equations). Finally, it is important for our application to distinguish so-called *linear r-methods* giving one solution and *non-linear r-methods* giving generally several solutions.

Definition 2. Let m be an r-method, E_m be the set of equations solved by m, and O (resp. I) be the set of output (resp. input) variables of m.

The r-method m is a **linear r-method** iff all the equations in E_m are linear in terms of variables in O (i.e., E_m in which the variables in I are replaced by a constant become linear). m is a **non-linear r-method** iff at least one equation in E_m (in terms of O) is non-linear: m may produce several solutions.

For instance, an r-method that computes the position of some 3D point A located at known distances from three other points B, C, D is a non-linear r-method. This r-method generally produces two possible positions for A.

An r-method that computes the parameters of a line based on the current position of two points incident to this line is a linear r-method. Even though an incidence constraint is bilinear, when the coordinates of the involved points are known, this gives linear equations relating line coordinates.

Hypotheses on r-methods

R-methods must compute a finite set of solutions for the output variables. In other words, the dimension of the variety of the solutions is 0. Therefore r-methods have generally as many equations as output variables. This is the case for the r-methods in our dictionary.

In addition, an r-method, especially a non-linear r-method, must be able to compute *all* the solutions satisfying the involved equations. Indeed, this allows the backtracking phase described in Section 7.1 to combine the solutions computed by the different r-methods in the plan, without losing any solution.

The remark above highlights that numerical local minimization methods cannot be used for executing an r-method because they cannot obtain *all* the solutions for the output variables.

Interests of r-methods

Using r-methods for decomposing the constraint system of the model has a lot of advantages:

- The code of an r-method allows a very good performance. Executions of r-methods in our dictionary run in several microseconds.
- A lot of r-methods in our dictionary are linear while the involved constraints are bilinear (e.g., incidence, parallelism). This highlights that using r-methods to decompose a system of equations is an interesting way to lower the complexity of the equations and thus to improve the performance.

- The semantics behind a given r-method (i.e., the fact that it is a theorem of geometry) ensures that the implied geometric constraints can be solved and helps to detect singular configurations. On the contrary, the subsystems of equations created by pure graph-based decomposition methods, such as the Maximum-matching, are arbitrary (see Refs. [1,14]).
- As said above, r-methods yield all the solutions to the implied equations.

4.4. *Graph representation of the model and data structures*

The algorithms used by our system require a structural view of the entities in the scene. The geometric constraint system and the equation system are respectively represented by a *constraint graph* and an *equation graph* (see Figures 2 and 3). An equation graph indicates the dependencies between equations and variables in the scene.

Definition 3. A **constraint graph** is a bipartite graph where vertices are constraints and objects, represented by rectangles and circles respectively. Each constraint is connected to its objects.
An **equation graph** is a bipartite graph (V, E, A) where vertices are equations in E and variables in V, represented by rectangles and circles respectively. Each equation is connected to its variables by an edge in A.
An **enriched equation graph** (V, E, A, M) is an equation graph (V, E, A) enriched with a set of hyper-edges corresponding to r-methods in M. A hyper-edge of a given r-method $m \in M$ is a subgraph induced by the variables and equations of m.

Our system is implemented in `C++`. The different entities (constraints, geometric objects, equations, variables and r-methods) are represented by structured objects. Several fields have been added to allow a direct access to the entities. For example, for a given variable v, we can know in constant time the set of equations involving v, of which r-method v is an output variable, to which geometric object v belongs, and so on. In this implementation, the constraint graph, the equation graph and the enriched equation graph share the same data structures.

The dictionary of r-methods is implemented as a hash table in order to make the automatic r-method addition phase quicker. Details about this hash table are given in Section 5.

The next two sections detail the algorithms used by the constraint planning: the automatic addition of r-methods to the equation graph (based on the dictionary), and the computation of a set of input parameters and a sequence of r-methods (based on the enriched equation graph).

5. Automatic R-method Addition Phase

This phase enriches the equation graph with r-methods found in the dictionary. It considers as input:

- the constraint graph corresponding to the scene;
- the dictionary of r-method patterns.

This phase works on the constraint graph. It performs a matching between subgraphs (made of constraints and objects) in the constraint graph and r-method patterns present in the dictionary. More precisely, we handle a *subgraph isomorphism* problem. All the connected subgraphs of the constraint graph with a "small" size are explored. When a subgraph corresponds to an entry in the dictionary, the corresponding r-methods are added to the equation graph.

For instance, on the 2D scene, a certain iteration of the r-method addition phase considers the subgraph made of nodes P_a, C_8, L_a, C_1, P_b (see Figure 2). A corresponding subgraph pattern (i.e, 2 points incident to a line) is found in the dictionary, so that the r-method m_1 (i.e., line L_a passing through two known points P_a and P_b) is created and added to the equation graph.

The algorithm explores all the connected subgraphs of size less than or equal to a small value k (see next paragraph). For every found subgraph, the procedure `Subgraph_recognition` compares it with the subgraph patterns in our dictionary. If the subgraph matches, the corresponding r-methods are added to the equation graph[c].

5.1. *Exploring all connected subgraphs of size at most k*

The value k is the maximum number of nodes (objects+constraints, or constraints only in the current implementation) implied in any r-method of the dictionary (e.g., 7 in our system; 4 in the last version - see Section 8). Starting from a single node (S is a singleton), the subgraphs are built by incrementally adding a neighbor node to the current connected subgraph S until the size k is reached. This depth-first search algorithm detailed in Algorithm `All_connected` is a simplification of the algorithmic scheme presented in Ref. [32].

```
algorithm All_connected (S: set of nodes; d: current depth; k: max size; G: constraint graph):
    Subgraph_recognition ($S$)
    if $d < k$ then
        $N' \leftarrow$ Selected_neighbors ($S$, $d$, $k$, $G$)
        for every neighbor $n$ in $N'$ do
            All_connected ($S \cup \{n\}$, $d+1$, $k$, $G$)
        end
    end
end.
```

The time complexity of this algorithm[d] is $O(N \times a \times k^4)$, where N is the actual number of connected subgraphs of size k or less and a is the maximum degree of

[c]Remember that several r-methods may exist for the same set of constraints.
[d]The call to `Subgraph_recognition` is not taken into account.

nodes in the graph. N is $O(n^k)$, where n is the number of vertices in the constraint graph.

5.2. *Subgraph recognition*

The function `Subgraph_recognition` compares every subgraph S found in the constraint graph with the subgraph patterns in our dictionary. However, the problem of deciding whether two graphs are *isomorphic* is still an open problem for which no polynomial algorithm is known.[33] In order to quickly know whether a subgraph S is isomorphic with a subgraph S' in the dictionary, we proceed as follows:

(1) We first compute a string e corresponding to the number of nodes in S in every category: the number of points in S, its number of lines, number of planes, number of point-line incidence constraints, and so on. The dictionary is implemented as a hash table, and such strings are the keys for indexing into the hash table (hash functions). If the key e corresponds to no entry in the dictionary, the second step below must not be performed and no r-method will be created based on the subgraph S.

    Otherwise, this means that an entry in the hash table contains one set ES' of subgraphs (having the same number of nodes in every category). The step 2 below checks whether one subgraph pattern $S' \in ES'$ is isomorphic with S.

(2) To know whether two graphs S and S' (with the same number of nodes in every category) are isomorphic, we use a combinatorial process inspired by the solving process of Constraint Satisfaction Problems (chronological backtracking). In short, objects in the subgraph S are reordered to be matched with objects in the pattern S'. Two objects at the same rank in the order must have the same type and also the same types of constraints with objects placed before.

(3) If S and S' match, then the r-methods associated to S' are added to the equation graph.

In our dictionary, the 60 r-method patterns are generated by 45 different subgraph patterns. The hash table contains 45 entries, which means that all the subgraph patterns are discriminated by their number of nodes in every category (i.e., the size of ES' is always 1 in our current version). The example in Figure 6 highlights why the combinatorial process (step 2) remains necessary.

5.3. *Practical time complexity*

In practice, as detailed in Section 8, the time complexity of the r-method addition phase is negligible (one or two seconds for our 3D models). Two reasons explain this good behavior. First, in our current dictionary, the subgraph patterns are small, so that the size k is small. Second, constraint graphs corresponding to scenes are rather sparse: the number of equations is equal to about half of the number of variables.

The situation would worsen if the designer wanted to add in the dictionary more complicated r-methods, especially r-methods with more than one object as output.

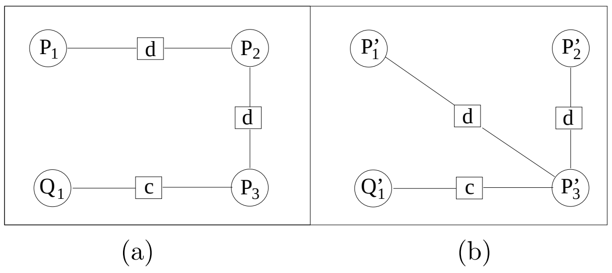


Fig. 6. Two subgraphs with the same entry in the hash table, i.e., characterized by the same number of points, planes, distance constraints and incidences. (a) a “bad” subgraph found in the constraint graph with no corresponding r-method; (b) a subgraph pattern in our dictionary made of 3 points P'_1, P'_2, P'_3, a plane Q'_1, two distance constraints and one incidence.

In this case, we think that more sophisticated subgraph isomorphism algorithms should be envisaged.[34,35,36]

6. Computing a Plan and a Set of Input Parameters

These computations are obtained by the GPDOF algorithm.[1] GPDOF[e] computes a sequence of r-methods to be executed for satisfying all the equations (the plan). GPDOF solves this combinatorial problem in polynomial time. The main advantages of GPDOF are the following:

- GPDOF is very fast (quasi-linear in practice).
- GPDOF can find a sequence of r-methods present in a dictionary if such a plan exists.
- A set of input parameters can be immediately deduced from the plan. Thus, GPDOF is also a procedure to determine a set of input parameters in polynomial time.

These attractive properties come under the assumption that the constraint system contains no redundant constraints, that is, the system must include only independent equations. Section 6.5 details this point and explains the first procedures used by our tool for removing redundant constraints before the use of GPDOF.

6.1. *Description of GPDOF*

GPDOF works on an enriched equation graph (see Section 5). GPDOF runs the three following steps until no more equation remains in the equation graph G (success) or no more free r-method is available (failure):

(1) select a **free** r-method [f] m,
(2) remove from G the equations and the output variables of m,
(3) call the Connect procedure: create all the *submethods* of every r-method m_i that share equations or output variables with m (see Section 6.2).

[e]GPDOF stands for General Propagation of Degrees of Freedom.
[f]Recall that output variables of a *free r-method* appear in no “external” equations (see Definition 1).

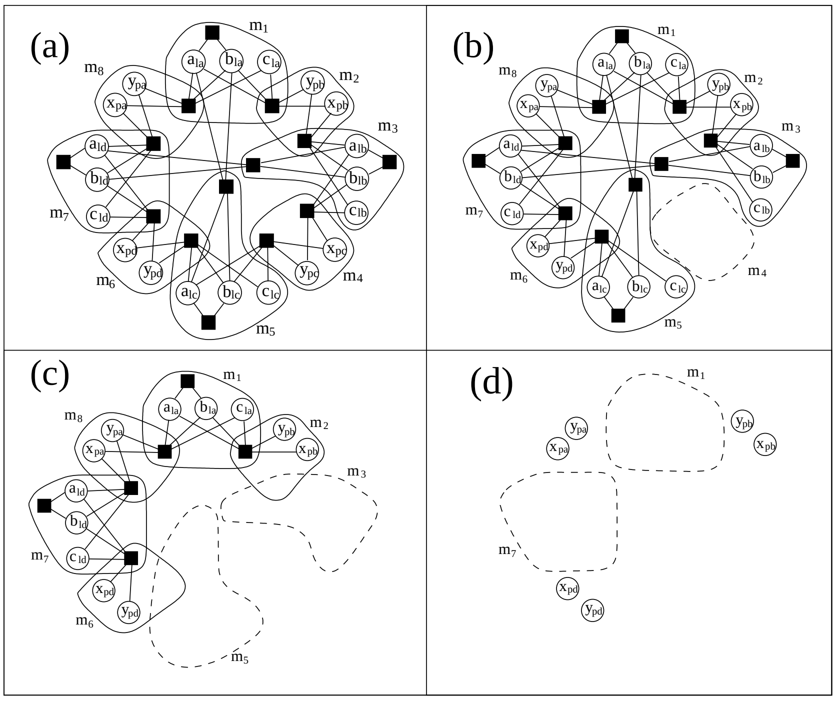


Fig. 7. A constraint planning phase performed by GPDOF on the 2D scene. (a) In the beginning, r-methods m_2, m_4, m_6, m_8 are free, so that one of them is selected, e.g., m_4. (b) This selection implies the removal of the equations and the output variables of m_4 from the equation graph. (c) This frees r-methods m_3 and m_5 which are selected and removed next in any order. (d) R-methods m_1 and m_7 become free and can be selected. The process ends since no more constraint remains in the equation graph. The obtained plan is the sequence (m_1, m_7, m_3, m_5, m_4) and the input parameters are the remaining variables.

A plan can be obtained by reversing the selection order: the first selected r-method will be executed last. The work of GPDOF is illustrated in Figure 7.

The first two steps above define the standard PDOF algorithm on which GPDOF is based (PDOF accepts only r-methods solving one equation).[24] Iteratively selecting free r-methods ensures that no loop is created in the plan.

GPDOF may fail when no more free r-method is available. In this case, it is ensured that no complete plan can be computed. One obtains an incomplete plan which solves only a subset of the equations (i.e., the equations removed by step 2 of GPDOF) and contains thus more input parameters. This incomplete plan can nevertheless be executed, although the equations not removed in steps 2 will not be satisfied.

6.2. *Overlap of r-methods and submethods*

It appears that, when r-methods solve several equations, there is no guarantee that the standard PDOF finds a plan, even if one exists. This means that straightforward extensions of PDOF to geometry (see Refs. [27,28,26]) may be unable to compute a sequence of subsystems corresponding to patterns in the dictionary, even if one such sequence exists. A simple example is described in Figure 6 of Ref. [1], and the case does occur in real applications. To overcome this drawback, one needs to be able to select r-methods that "overlap", that is, sharing equations and variables. A plan that contains overlapping r-methods means that some constraints will be solved several times during one plan execution. This analysis has led to the design of GPDOF that introduces the notion of *submethod* and adds the third step above.

That step, called Connect in Ref. [1], maintains a connectivity property on the hyper-edges (r-methods). The procedure is called once a free r-method m has been selected in step 1 and removed in step 2, which removes some equations and/or output variables from m_i. The obtained subgraph of m_i is called *submethod* m'_i. The modified hyper-edge m'_i is maintained in the set of candidate r-methods. This means that a "partially" removed r-method m_i is still candidate for a future selection. If m_i becomes free during the process and is selected, it will overlap the r-method m. It appears that the subgraph corresponding to a submethod must be a connected graph, so that a given r-method m_i may be theoretically split in several connected submethods (m'_{i1}, m'_{i2}...) . Refs. [1,13] detail how submethods are precisely constructed and why submethods are needed to ensure that a sequence of r-methods will be found (if any).

Examples of submethods are depicted in Figure 8. R-methods m'_5, m'_7 are the submethods of resp. m_5, m_7 due to the selection of m_6 and the removal of equations solved by m_6 (black rectangles).

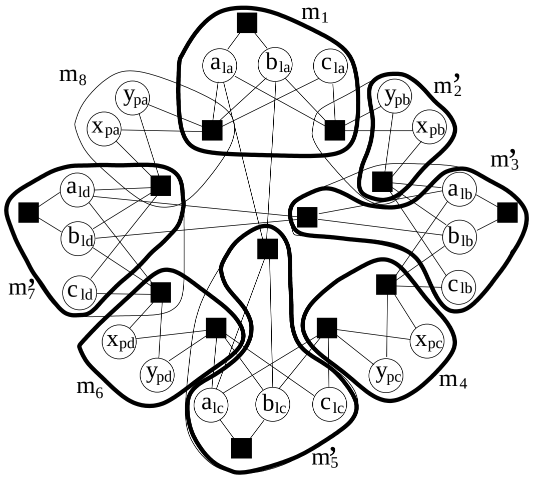


Fig. 8. GPDOF may first select m_6 that is free. Step 3 of GPDOF then creates the submethod m'_5 of m_5 and the submethod m'_7 of m_7. The process continues and selects m_4, m'_5, m_1 (creation of submethod m'_2), m'_2 (creation of submethod m'_3), m'_3, and finally m'_7. The obtained plan is the sequence (m_7, m_3, m_2, m_1, m_5, m_4, m_6). Selected r-methods (m_1, m_4, m_6) and submethods (m'_2, m'_3, m'_5, m'_7) are represented by thick hyper-arcs.

It is important to understand that the notion of submethod is only used during the work of GPDOF which is graph-based, but no submethod appears in the final plan: every time GPDOF selects a submethod, the corresponding r-method is added to the plan in order to be executed later.

The example also shows that, due to the selection of overlapping r-methods, some constraints are solved several times by different r-methods in the plan. For instance the two r-methods m_2 and m_3 belong to the same plan so that the incidence constraint between point P_b and line L_b will be solved twice.

Solving several times a same equation has no significant impact on the plan execution. In particular, all the constraints are solved in the end. The only drawback is that a plan with overlapping r-methods contains generally more r-methods (e.g., the plan shown in Fig. 7 contains 5 r-methods, while the plan in Fig. 8 contains 7 r-methods). As a result, we could expect a loss in performance.

However, r-methods are executed in several microseconds (see Section 8). Moreover, the phenomenon of selecting overlapping r-methods can be easily limited by heuristics: when GPDOF has a choice between several free r-methods at a given iteration (step 1), GPDOF selects one r-method that is not a submethod (if any).

6.3. *Properties of GPDOF*

Due to the notion of submethod, it is proven that GPDOF guarantees to compute a sequence of r-methods present in the dictionary, if one such sequence exists.[1]

In addition, GPDOF solves this combinatorial problem in polynomial time. Its worst-case time complexity is $O(n \times dc \times dv \times m \times (g \times dc + g^2))$,[1] where n is the number of equations, m is the maximum number of r-methods per equation, dc and dv are the maximum degrees of respectively equations and variables in the equation graph, and g is the maximum number of equations and output variables involved in an r-method [g].

GPDOF (and the r-method addition phase) run in a few seconds on our two 3D models with several hundreds equations, showing that it should be acceptable for real applications.

6.4. *Computing the input parameters*

Since GPDOF computes the plan in a reverse order, obtaining the input parameters is a side-effect of GPDOF. When no r-method in the plan corresponds to a submethod selection, the input parameters simply consist of the variables that are output of none of the r-methods in the plan. This yields the 6 coordinates of points P_a, P_b, P_d for the plan illustrated in Fig. 7. The general case is a little bit more complicated and produces two disjoint subsets of input parameters:

- The set $\mathcal{P}_1$ contains the variables which are output by no r-method in the plan (as above).

[g]Theoretically, m is $O(n^g)$, rendering the approach practicable for small patterns only. A reviewer has built a scalable 2D example (made of points and distances between points) with a quadratic number of r-methods, due to a pair of points implied in all the constraints (i.e., with an unbounded value of dv). We know no pathological example when the number dv of constraints per variable is limited...

- The set $\mathcal{P}_2$ comes from the overlap phenomenon. $\mathcal{P}_2$ contains variables v such that:
  - $v \notin \mathcal{P}_1$,
  - v is an input variable of an r-method m_j in the plan,
  - v is not an output variable of any r-method m_i placed before m_j in the sequence,
  - v is an output variable of an r-method m_k placed after m_j in the sequence.

The values of variables in $\mathcal{P}_1$ must be known before the plan is executed and will not be modified by this execution. On the opposite, the initial value of a variable v in $\mathcal{P}_2$ is used when an r-method m_j is executed, but v will be modified later by another r-method m_k in the plan.

In the plan illustrated in Fig. 8, the subset $\mathcal{P}_1$ contains the coordinates of point P_a. The subset $\mathcal{P}_2$ contains the parameters of points P_b, P_c, P_d and lines L_a, L_b.

Number of input parameters in the system

The example has been chosen to illustrate the two subsets of input parameters and contains a large set $\mathcal{P}_2$. In practice however, due to the heuristics avoiding the selection of submethods by GPDOF, $\mathcal{P}_2$ is small. This will be underlined in the experiments made on the two realistic models presented below. Anyway, a natural question arises: what is the number of input parameters?

Assume that all the r-methods are *square*, that is, they have as many output variables as equations[h]. If $\mathcal{P}_2$ is empty (because no submethod has been selected by GPDOF), it is straightforward to prove that $|\mathcal{P}_1| = n - e$ (n is the number of variables in the equation system, and e is the number of equations). In the general case however:

- $|\mathcal{P}_1| \leq n - e$ and
- $|\mathcal{P}_1| + |\mathcal{P}_2| \geq n - e$

Roughly, this means that the more GPDOF must select submethods to calculate a plan, the larger will be the set of input parameters.

It appears however that the plan produced by GPDOF can be used to yield a set of $n - e$ input parameters. Indeed, a set of parameters $\mathcal{P}'_2$ can be computed by a maximum matching such that $|\mathcal{P}_1| + |\mathcal{P}'_2| = n - e$. A maximum matching must be applied on every subgraph corresponding to the selected submethods. The variables that are not matched constitute the set of input parameters $\mathcal{P}'_2$ (see Ref. [14], page 151). Although interesting in theory, considerations about performance let us think that this variant is not promising in practice. Indeed, the transformed submethods cannot be solved by a hard-coded procedure anymore, so that we need to resort to a generic solver.

[h]This the case for the 60 r-methods in our dictionary.

6.5. *Dealing with singularities and redundant constraints*

Singularities have been the major cause of occasional divergence of the optimization process. For instance, a singularity may occur during an r-method execution that computes a plane based on 3 almost collinear points. Such singularities necessarily occur inside an r-method subsystem and can often be easily detected. In particular, before a linear r-method is added to the enriched graph, the corresponding subsystem of equations (in which the initial values of the input variables are used) is checked against a singularity using a Singular Value Decomposition (SVD).[37] Details are discussed in Ref. [14].

Another problem is that our graph-based algorithms may be misled by redundant constraints. However, this can often be fixed in practice by making use of geometric information.

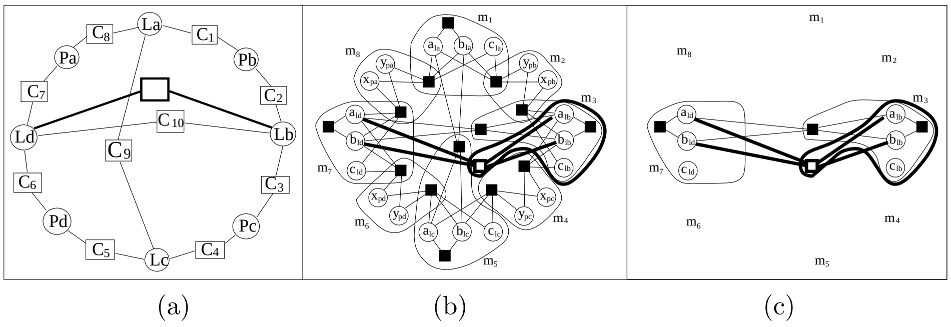


Fig. 9. Failure of GPDOF in presence of a redundant parallelism constraint. (a) The parallelism constraint is redundant to the parallelism C_{10} (in the constraint graph). (b) The enriched equation graph with the corresponding additional equation (white rectangle) and two additional r-methods (only one is represented). (c) After having removed all possible free r-methods and submethods, GPDOF is stuck because the redundant equation prevents r-method m_3 from being free.

Redundant constraints involve non-independent equations. Because they correspond to theorems of geometry, the r-methods selected by GPDOF in the plan correspond necessarily to non-contradictory and independent systems of equations. However, GPDOF may fail in presence of redundant constraints because the selection of free r-methods is purely structural. As an example, consider Figure 9 where an additional parallelism constraint has been added between lines L_b and L_d. This constraint is redundant with the existing parallelism constraint and prevents GPDOF from finding a plan. Since the selection step of GPDOF is structural, all occurs as if all equations were independent.

It is of course not acceptable to rely on the user to not introduce redundant constraints. Dealing with constraint redundancy has been a subject of research in the CAD community for a long time and it is still an open problem in the general case. Straightforward procedures have been introduced in our tool to remove very common causes of redundancy. For example, one type of redundancy occurs if the user adds an incidence c_1 between a point P and a line L, an incidence c_2 between the line L and a plane A, and an incidence c_3 between the point P

and the plane A. In this case, our procedure removes from the whole system all redundant constraints such as c_3. We found these procedures helpful in practice although we know that many occurring redundancies cannot be handled this way. More complicated procedures developed in the geometry/CAD community should be used to remove more redundancies. For example, a lot of redundant parallelisms and orthogonalities could be removed by using a straightforward routines closing the Desargues theorem.[38,39] Also, a generalization of the numerical technique used to tackle singularities could be used to detect redundant constraints.

6.6. *Comparison between geometric and equational decomposition techniques*

In the introduction, we made a comparison between both main equational decomposition techniques. On one hand, as opposed to Maximum-matching, `GPDOF` can produce "small" and "correct" subsystems of equations that specific hard-coded procedures can solve quickly. On the other hand, Maximum-matching is not limited by predefined types of subsystems present in a dictionary. In fact, both algorithms can complete each other. The reader interested in this subject will find in Ref. [1] a description of a `MM-PDOF` hybrid algorithm, in which a maximum matching is incrementally updated and used to find a free subsystem when `PDOF` has failed to find one. This section compares equational (MM and PDOF based) and geometric (top-down and bottom-up) decomposition techniques. Four differences between both approaches are underlined below.

First, equational techniques are intrinsically not limited to pure geometric systems. They can also take into account constraints about thermodynamics, electricity, costs, and so on.

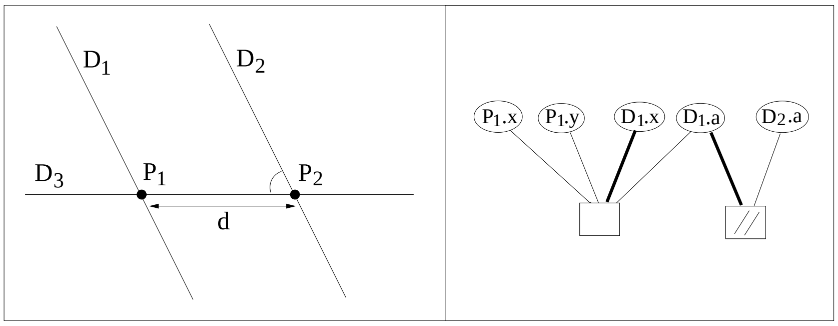


Fig. 10. **Left:** A rigid geometric system in 2D made of 2 points, 3 lines, 1 distance, 1 angle, 1 parallelism, and 4 point-line incidence constraints. **Right:** Equation subgraph including the incidence constraint between point P_1 and line D_1, and the parallelism between lines D_1 and D_2. An equational decomposition places line D_1 by solving the 2 equations in sequence: the angle of D_1 and then its distance to origin.

Second, equational techniques work at the variable/equation level and can then produce sometimes smaller subsystems than bottom-up or top-down techniques by distinguishing the different degrees of freedom of a same object. Figure 10-left shows a simple example in 2D where the two variables of line D_1 (for instance, its angle $D_1.a$ and its distance to origin $D_1.x$) are not distinguished by a geometric solver. Without detailing, $D_1.x$ and $D_1.a$ are both put into a subsystem of size 2. An equational decomposition technique would produce a finer decomposition with two subsystems of size 1, computing first $D_1.a$ and then $D_1.x$ (see Figure 10-right).

We do not believe that this observation gives a significant advantage to equational techniques. Indeed, to our knowledge, there exists no geometric rigid system that could be decomposed by an equational technique, but not by a geometric technique. If our intuition is true, for obtaining a finer decomposition and better handling systems like the one in Fig. 10-left, one has to perform the following steps:

(1) apply a geometric decomposition into clusters,
(2) apply a final equational decomposition inside every cluster.

This principle has been followed for obtaining the decomposed systems studied in Ref. [40]. Note that rule-based geometric decomposition algorithms generally perform this type of fine decomposition implicitly.

Third, geometric techniques can generally better decompose a rigid system because their recursive use of the rigidity concept transforms (and, in a sense, simplifies) the system of equations. Figure 11-left shows a small example in 2D made of points and distances between points.

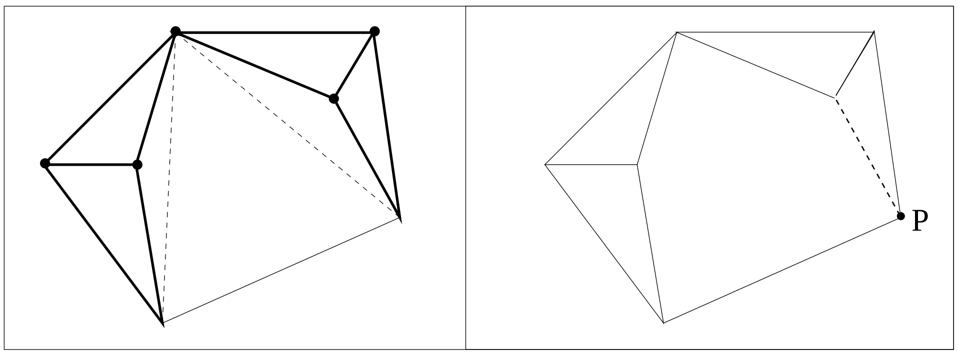


Fig. 11. **Left:** A rigid geometric system in 2D made of points and distance constraints. A geometric technique can decompose it into 2×2 subsystems while an equational approach computes one 10×10 subsystem (the 5 black points and the 10 bold-faced edges - distances). **Right:** An under-rigid geometric system tackled differently by equational and geometric decomposition techniques.

This system can be decomposed by geometric techniques, but not by `GPDOF` (or Maximum-matching). For instance, several bottom-up solvers identify the two rigid "rhombuses", replace them by a representative (the distance constraints in dotted lines) and finally handle the last triangle. `GPDOF` identifies no free 2×2 r-method, and Maximum-matching (or `GPDOF` with a rich dictionary!) creates a large 10×10 subsystem including the point shared by the two rhombuses, the four neighbouring points and the 10 (bold-faced) distance constraints.

Fourth, as underlined at the end of the introduction, geometric and equational techniques do not tackle under-constrained (i.e., under-rigid) systems in the same way. Due to its inherent mechanism, a geometric decomposition method must complete an under-rigid figure so as to maintain the obtained figure invariant by displacements. GPDOF must not fulfill this property.

This point is illustrated in Figure 11-right by a variant of the previous system from which a distance constraint (in dotted line) has been removed. A geometric top-down decomposition technique, such as the ones described in Refs. [6,16], completes the figure by adding, among other possibilities, the dotted edge, while GPDOF works differently. It first selects and removes two subsystems in the "incomplete rhombus". Next, GPDOF selects and removes for example point P with the corresponding distance. This means that one coordinate of P (let us say $P.x$) is chosen as input parameter while the other ($P.y$) is computed by the selected 1×1 free r-method.

Finally, the last example also illustrates that the more under-constrained the system is, the better GPDOF works, because *it is easier to find a free r-method in the whole system.* Of course, an under-constrained system is not a sufficient condition to make it (well) decomposable by equational techniques[i].

7. Optimization Phase

As said in the overview, the optimization process produces values for the variables such that all the constraints are satisfied *exactly* and the reprojection error is minimized.

It is first important to understand that executing the plan computed by the constraint planning is very fast in our system because the r-methods are hard-coded. To give an idea, solving a plan made of 100 equations needs about 12 milliseconds. This explains why the plan execution step is called numerous times in our optimization phase, performed as follows:

(1) Based on the plan computed by GPDOF and the variable values calculated by the initialization phase, a *backtracking phase* chooses which solution to select for every r-method.

    Indeed, if the execution of a non-linear r-method in the plan produces at most k different solutions, the number of total solutions given by the plan execution can potentially be multiplied by k. Thus, the total number of solutions is majored by k^r, where r is the number of non-linear r-methods in the plan[j].

    The backtracking phase is a preprocessing step called only once before the numerical optimization. Based on this selection, the following plan executions will always select the same solution to every non-linear r-method in the plan.

[i]For instance, the under-rigid graph of Figure 12 in Ref. [6] cannot be decomposed into 2×2 subsystems by GPDOF (or by Maximum-matching).

[j]Our dictionary contains 7 non-linear r-methods (among 60), each yielding (at most) $k = 2$ solutions.

(2) Then, a standard *numerical optimization algorithm* (a Levenberg-Marquardt algorithm in our system) interleaves input variable modifications and constraint satisfaction phases. The constraint satisfaction is obtained by executing the plan selected by the backtracking phase.

The numerical algorithm only modifies the values of the input parameters ($\mathcal{P}_1$ and $\mathcal{P}_2$). Every time it does so, the plan is executed, resulting in new coordinate values for all the other scene objects. The cost function is then computed as the reprojection error of all the points.

7.1. *Backtracking phase*

This phase computes one solution with a lowest cost. This problem is called *root identification problem* in Ref. [6]. While some geometric constraint solvers use heuristics to select the desired solution, we resort here to a combinatorial process because the (non-linear) r-methods generally yield several (sub)solutions to the corresponding subsystems.[41,40]

The precise cost to be minimized is $C = R + \alpha C_S$ (Experiments have led us to choose $\alpha = 1$.) The same cost function is taken into account in the backtracking phase and in the optimization. This multi-criteria cost function C has two components: the well-known reprojection error R, but also a constraint violation cost C_S. The latter is the sum of the constraint violation costs induced by all non-linear r-methods in the plan. The constraint violation cost associated to an r-method m_i is 0 if the r-method succeeds and yields one or more solutions. Otherwise, the constraint violation cost of m_i is a “smooth” measure of the error related to the semantics of constraints. For instance, the error of a point-point distance is the square difference between the distance value in the equation and the actual distance between the points. The error of an incidence constraint is the square of the actual distance between the two related objects.

The backtracking phase is a *branch and bound* algorithm executing all the r-methods in the plan in order. When a non-linear r-method m_i produces several solutions for its output variables, all the solutions are tried, which leads to a combinatorial process. However, this process is not so costly in practice because, among the k solutions given by an r-method, the algorithm tries first the solution that minimizes the reprojection error. In addition, branches of this tree search with a cost greater than the best current cost are cut. Ref. [14] explains how singularities are tackled during this phase.

The backtracking results depends on the initial values of the model variables. It may happen that the solution is not visually correct, especially when the initial reconstruction does not satisfy well the geometrical constraints. However, our experiments have shown that the reprojection error criterion is quite reliable.

7.2. *Exact constraint satisfaction*

The optimization process described above satisfies exactly the equations involved in linear r-methods. This is also the case when the execution of non-linear r-methods succeeds. However, the execution of a non-linear r-method m_i may fail. In this case, the latest computed values for the corresponding variables are reused, and a measure of the constraint violations is added to the cost function.

However, after several optimization iterations, the model quality increases and the case occurs that a given non-linear r-method m_i succeeds for the first time. That is, m_i had always given 0 solution during the backtracking phase and in the previous plan execution steps as well. Consider for example an r-method computing the position of a point using distance constraints from 3 other points. It may happen that the initial positions of these points are inconsistent with the distance contraints, so that the r-method yields no solution. However, with the increasing quality of the reconstruction, the point positions can be moved to positions allowing the distance constraints to be satisfied, and the r-method to give the two possible positions for the output point. When m_i succeeds for the first time, among the k possible solutions yielded by m_i, our optimization process selects the one leading to the lowest reprojection error.

This means that the number of unsatisfied non-linear equations decreases as long as the model quality increases. This great behavior highlights the interest of our fast plan execution step included inside the numerical algorithm.

8. Experimental Results

We have used our approach to build a model of the *Place Notre-Dame* in Grenoble. A set of images have been used, together with architectural plans from which several distance measurements have been extracted. We have first built a medium-size model constructed from 5 images, called `ND1` hereafter. A larger model, called `ND2`, including peripheric walls and additional details, has then been built from 15 images. The characteristics of these models are reported in Table 1. Three of the images used for reconstructing the models are shown on Fig. 12.

	ND1	ND2		ND1	ND2
#images	5	15	#point projections	286	546
#variables	436	819	#equations	273	452
#objects	120	238	#constraints	151	279
#points	90	189	#incidences	124	234
#lines	23	28	#angles	17	34
#planes	7	21	#distances	10	11

Table 1. The two scenes: Notre Dame (`ND1`) and Extended Notre Dame (`ND2`).

8.1. *Reconstruction results*

The interest of our method is especially well illustrated in Figure 13. The reconstruction results highlight that models are visually and geometrically correct. The first column contains the top and side views of the initial model, where constraints are respected approximately. The second column contains the top and side views of the model obtained using a standard unconstrained optimization method. One of the points is visible only in two images with a very small baseline and its position is false due to divergence of the optimization process. Other parts of the model also suffer from several artifacts such as an unsatisfied coplanarity. By imposing appropriate constraints, we have overcome these problems. The third column of Figure 13 contains the top and side views of the model produced by our method. We show how the parts of the model mentioned above have been corrected, leading to a visually correct model.

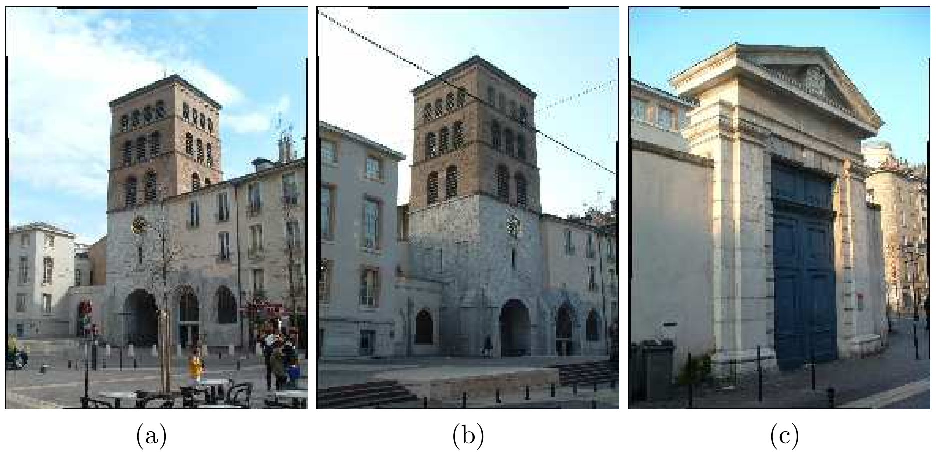


Fig. 12. Three images from the sequence *Notre Dame*. Image (c) has been included only into the ND2 sequence.

Several artifacts have been corrected after several optimization steps, which highlights the interest of our optimization phase and of our fast plan execution (due to r-methods). Also, when in the initial reconstruction the constraints are not sufficiently satisfied, it may happen that a plan execution causes artifacts in the scene, such as in the center of Figure 14–(a). The optimization corrects the errors created this way (see Figure 14–(b)).

Table 2 (top) reports statistics about the number of r-methods added automatically to the equation graph (#r-methods), the number of r-methods selected in the plan (plan size), the number of bundle adjustment iterations (#iterations) and plan executions (#executions) performed by the optimization. Each optimization iteration calls several plan executions (on average, $185 = \frac{2040}{11}$ for ND1; 462 for ND2), and chooses one of them according to the criterion. The reprojection error (reproj. error) and the number of violated constraints (#violated) is given before the optimization (i.e., after a single plan execution), and also after the optimization.

8.2. *Performance tests*

All the times reported below have been obtained with a `Linux` operating system on a `Pentium IV 2 Ghz` processor.

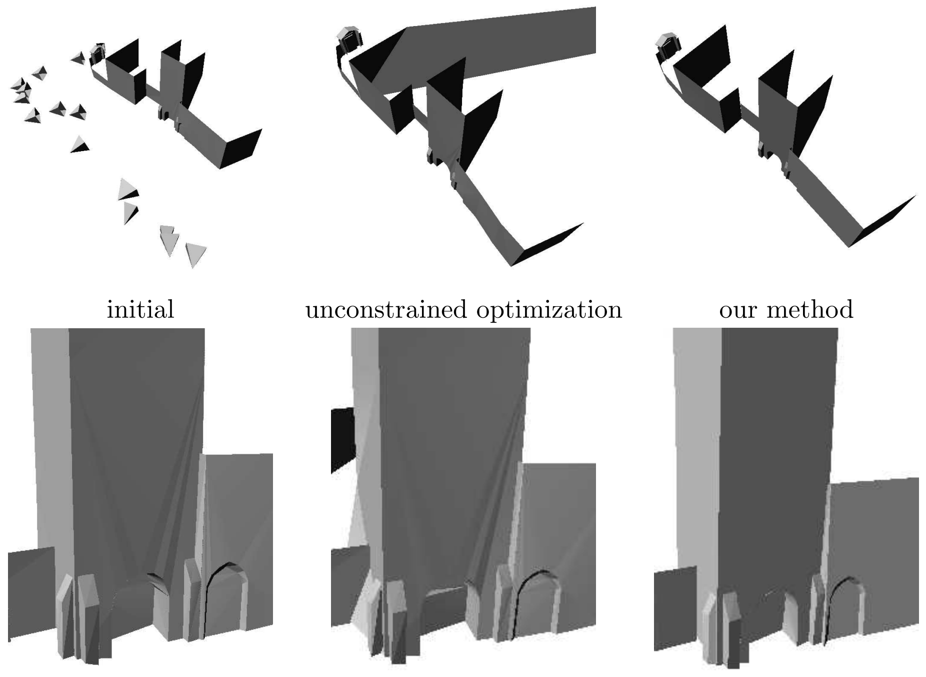


Fig. 13. Reconstruction of the ND2 scene. Top (1st row) and side (2nd row) views of the initial model (1st column); the model obtained using unconstrained optimization (2nd column); the model obtained using our method (3rd column).

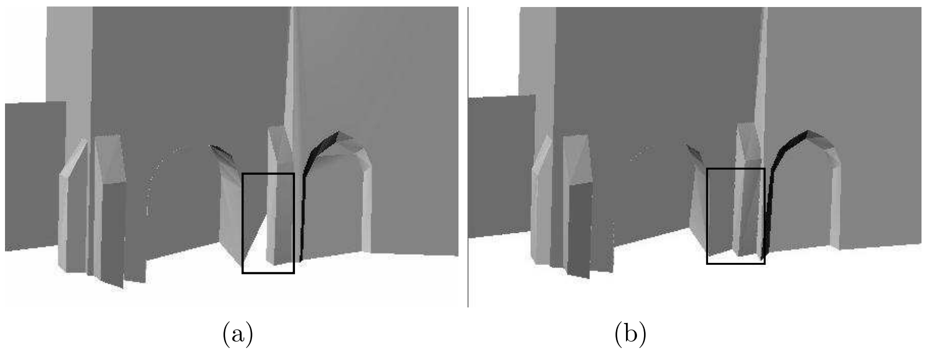


Fig. 14. Part of the ND2 scene after (a) one step of GPDOF; (b) optimization: initial artifacts have been corrected.

Recall that our r-method dictionary contains 60 r-methods. The most complex r-methods solve 3 geometric constraints (6 equations) and involve 4 geometric objects (1 as output and 3 as input).

We have first evaluated the time required to execute one r-method. This time varies from 20μs to 90μs. The execution time of non-linear r-methods is shorter ($\sim 28\mu$s on average) because the corresponding procedures are hard-coded, while linear r-method routines use a generic SVD (Singular Value Decomposition). Table 2 (bottom) details the times spent in the different phases of our model reconstruction system.

	ND1	ND2		ND1	ND2
# r-methods	3017	6695	Plan size	118	228
# iterations	11	17	$\mathcal{P}_1$	158	352
# executions	2040	7866	$\mathcal{P}_2$	10	25
Reproj. error before	20.42	23.01	Reproj. error after	4.038	4.16
# violated before	2	2	# violated after	2	1

Phase	ND1	ND2
Initialization	21	331
R-method addition	0.7	1.7
GPDOF	1.4	3.9
Backtracking	0.2	0.2
Optimization	53	467

Table 2. Statistics on our GPDOF-based model reconstruction (top), and performance of the different phases in seconds (bottom).

The time for the initialization phase is dominated by the non-linear unconstrained optimisation process ($\sim 80\%$ of the total time) which is executed to refine the initial, parallelepiped-based calibration. A very interesting characteristic of our system is that the constraint planning phase (automatic r-method addition, GPDOF and backtracking) requires only a few seconds. Note also that the time required for executing a plan is really impressive. With ND2, the 451 equations can be solved 7866 times in 467 s (only one solution per subsystem is chosen). This means that the hard-coded r-methods allow us to solve 100 equations in 13 milliseconds on average!

Remark

Table 2 (bottom) clearly shows that the exploration of all the connected subgraphs of size at most k is fast (0.72 s for ND1). As mentioned in Section 5.1, the time complexity of this phase is highly dominated by the number of connected subgraphs. This number strongly depends on the size k of the largest subgraph. In the first version of our tool,[42] this phase was even more time-consuming (253 s on ND1). The exploration of the constraint graph considered connected subgraphs in terms of objects *and* constraints. The value of k was 7 because the largest r-methods in the dictionary include 3 geometric constraints and 4 objects. In the new version, $k = 3$ because the connected subgraphs are built in terms of constraints only. Two constraints are neighbors in the constraint graph iff they share an object.

This means that our simple automatic r-method addition algorithm can handle in practice any type of r-method that outputs a single object (point, line or plan), even if other types of constraints are added, such as angles, distance ratios.[14]

8.3. *Comparison with a penalty function method*

We have compared our model reconstruction system with a classical constrained optimization method where the constraints introduce a penalty to the cost function.[43,44] No free tool was rapidly available so that we have developed a simple version in our architecture.[14] Based on `ND1` and `ND2`, we have created two models respecting all the geometrical and projection constraints perfectly. We then have introduced an increasing Gaussian noise on 2D coordinates on one hand and on 3D coordinates on the other. The comparisons between the two approaches are the following:

- The penalty-based approach is general, although it is difficult to compare penalties related to different types of constraints, and to tune the coefficient K increasing the constraint violation part in the cost function (as compared to the reprojection error) if one wants to avoid that the penalty-based method diverges.
- The visual aspect of the `GPDOF`-based method is very good; the visual aspect of the penalty-based method is correct, although we can notice that the constraints are not exactly solved.
- The time required by the penalty-based method is about 2 or 3 times higher than the time spent by the `GPDOF`-based method.
- The constraint error with our tool is null, but is low with the penalty-based method, provided that a right value has been selected for the coefficient K.
- With very noisy initial data, the `GPDOF`-based method yields a significant reprojection error.

This study would suggest the interest of a hybrid optimization method where a call to the `GPDOF`-based method would follow a call to the penalty-based one to impose the constraints exactly. Also, another variant would use a final single execution of the plan produced by `GPDOF`.

9. Related Work in Computer Vision

(A more detailed version of this section can be found in Wilczkowiak's PhD thesis.[14])

In Computer Vision, geometrical constraints are traditionally expressed as a set of algebraic equations among real-valued variables and are solved by numerical methods. Depending on the complexity of the considered problem and the variety of considered objects and constraints, different approaches have been used.

Many systems limit the set of available constraints to collinearity, coplanarity and parallelism, so that linear approaches allow for fast scene reconstruction.[45,46]

When dealing with uncalibrated cameras, and more complicated constraints, it is necessary to use non-linear methods. Constrained minimization techniques, such as Lagrange multipliers (see Ref. [47]) or the penalty-based method described above,[43,44] do not guarantee however that the final model respects *exactly* the constraints, and lead to convergence problems because it is difficult to choose the weights associated to different types of constraints.

Ref. [48] describes an elegant solution to compute a set of input parameters for systems of bilinear constraints. The approach is based on a symbolic method, but the computational cost turns out to increase very quickly with the number of variables in the system, making the use of this method difficult for large systems.

In Ref. [49], the scene is represented by points, segments, and parallelograms. The constraints between them are used to reduce the number of parameters representing the scene objects by applying rewrite rules. However, the local application of the rewrite rules does not guarantee that a solution satisfying all the constraints in the scene will be found.

10. Conclusion

We have presented a solution to the problem of decomposing and solving large under-constrained systems of geometric constraints in the 3D space. An application to 3D modeling from images has shown that plunging the dictionary of r-method patterns in the actual system and decomposing the enriched equation graph with GPDOF run in several seconds. The obtained plan can be executed in hundredths of second.

We should highlight that the approach is dedicated to, but not limited to, geometric constraints. The general-purpose GPDOF works at the equational level and can thus easily take into account systems including geometric and non geometric components.

Several simple numeric and symbolic solutions related to singularities and redundant constraints have been implemented. Further developments should be performed to detect other cases of redundant constraints.

In conclusion, we hope that the geometric constraint community will pay more attention to the GPDOF decomposition method which appears to be very useful for tackling under-constrained systems when approximate values are known for the variables.

Acknowledgments

Thanks to Edmond Boyer, Christophe Jermann and Peter Sturm for the collaboration on previous stages of this work. Thanks to David Daney, Bertrand Neveu and Théodore Papadopoulo for useful discussions. Thanks to Gilles Chabert for comments. Thanks to Mr. Dominique Chancel for the architectural drawings.

References

1. G. Trombettoni. A Polynomial Time Local Propagation Algorithm for General Dataflow Constraint Problems. In *Proc. Constraint Programming CP'98, LNCS 1520*, pages 432–446, 1998.
2. G. Sunde. Specification of shapes by dimensions and other geometric constraints. In *IFIP WG Geometric Modeling*, 1986.
3. J. Owen. Algebraic solution for geometry from dimensional constraints. In *Proceedings of the* 1^{st} *ACM Symposium on Solid Modeling and CAD/CAM Applications*, pages 397–407. ACM Press, 1991.

4. G. Kramer. *Solving Geometric Constraint Systems.* MIT Press, 1992.
5. S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of constraint systems. In *Compugraphic*, 1993.
6. I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.
7. C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Finding solvable subsets of constraint graphs. In *Constraint Programming CP'97*, pages 463–477, 1997.
8. X.S. Gao and G.F. Zhang. Geometric constraint solving via C-tree decomposition. In *SM'03: Proc. of the ACM symposium on Solid Modeling and Applications*, pages 45–55, New York, NY, USA, 2003. ACM Press.
9. C. Jermann, B. Neveu, and G. Trombettoni. Algorithms for Identifying Rigid Subsystems in Geometric Constraint Systems. In *Proc.of IJCAI'03, International Joint Conference on Artificial Intelligence*, pages 233–238, 2003.
10. A. Borning. THINGLAB*: A Constraint–Oriented Simulation Laboratory.* PhD thesis, Stanford University, 1979.
11. D. Bondyfalat, B.Mourrain, and T.Papadopoulo. An application of automatic theorem proving in computer vision. In *Aut. Deduction in Geometry*, pages 207–231, 1999.
12. S. Ducasse, M. Blay-Fornarino, and A.M. Pinna-Déry. A reflective model for first class dependencies. In *Int. Conf. on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA'95*, pages 265–280, 1995.
13. G. Trombettoni. *Algorithmes de maintien de solution par propagation locale pour les systèmes de contraintes.* Thèse de doctorat, Université de Nice–Sophia Antipolis, 1997.
14. M. Wilczkowiak. *3D Modelling from Images Using Geometric Constraints.* PhD thesis, Institut National polytechnique de Grenoble, France, 2004.
15. S. Hudson. Incremental attribute evaluation: a flexible algorithm for lazy update. *ACM Transactions on Programming Languages and Systems*, 13(3):315–341, 1991.
16. R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pasto. Transforming an under-constrained geometric constraint problem into a well-constrained one. In *SM'03: Proc. of the ACM symposium on Solid Modeling and Applications*, pages 33–44, New York, NY, USA, 2003.
17. D. Serrano. *Constraint Management in Conceptual Design.* PhD thesis, MIT, 1987.
18. M. Gangnet and B. Rosenberg. Constraint programming and graph algorithms. In *Int. Symposium on Artificial Intelligence and Mathematics*, 1992.
19. H. Lamure and D. Michelucci. Qualitative study of geometric constraints. In B. Bruderlin and D. Roller, editors, *Geometric Constraint Solving and Applications*, pages 234–258. Springer, 1998.
20. D. König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. In *Math Ann 77*, pages 453–465, 1916.
21. A.L. Dulmage and N.S. Mendelsohn. Covering of bipartite graphs. *Canad. J. Math.*, 10:517–534, 1958.
22. C. Bliek, B. Neveu, and G. Trombettoni. Using Graph Decomposition for Solving Continuous CSPs. In *Proc. Constraint Programming, CP'98, LNCS 1520*, pages 102–116, 1998.
23. A. Lomonosov. *Graph and combinatorial algorithms for geometric constraint solving.* PhD thesis, University of Florida, 2004.
24. I. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System.* PhD thesis, Department of Electrical Engineering, MIT, 1963.
25. B. Vander Zanden. An incremental algorithm for satisfying hierarchies of multi-way, dataflow constraints. *ACM TOPLAS*, 18(1):30–72, 1996.
26. S. Ait Aoudia. *Modélisation géométrique par contraintes : quelques méthodes de résolution.* Ph.d. thesis, Ecoles des Mines de Saint Etienne, 1994.

27. P. Massan Kuzo. *Des contraintes projectives en modélisation tridimensionnelle interactive.* Ph.d. thesis, Ecole des Mines de Nantes, 1999.
28. E. Eremchenko and A. Ershov. Two new decomposition techniques in geometric constraint solving. Research report Preprint number 11, LEDAS Company, 2004.
29. M. Wilczkowiak, E. Boyer, and P. Sturm. 3D Modelling Using Geometric Constraints: A Parallelepiped Based Approach. In *ECCV'02, LNCS 2353*, pages 221–236, 2002.
30. M. Wilczkowiak, P. Sturm, and E. Boyer. The Analysis of Ambiguous Solutions in Linear Systems and its Application to Computer Vision. In *Proc. of the British Machine Vision Conference, BMVC*, pages 53–62, 2003.
31. R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, June 2000.
32. D. Avis and K. Fukuda. Reverse Search Enumeration. *Discrete Applied Mathematics*, 6:21–46, 1996.
33. C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
34. J.R. Ullman. An Algorithm for Subgraph Isomorphism. *JACM*, 23(1):31–42, 1976.
35. J.C. Régin. *Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique.* PhD thesis, LIRMM, Montpellier, France, 1995.
36. S. Sorlin and C. Solnon. A global constraint for graph isomorphism problems. In *Proc of the first conference CP-AI-OR'2004, LNCS 3011*, pages 287–301, 2004.
37. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C.* Cambridge Univ. Press, 1988.
38. A. Sosnov and P. Macé. Rapid algebraic resolution of 3D geometric contraints and control of their consistency. In *Worksh. on Aut. Deduction in Geometry, ADG*, 2002.
39. A. Sosnov. *Modélisation Géométrique par Séparation de Contraintes.* Thèse de doctorat, Université de Nantes, 2003.
40. C. Jermann, B. Neveu, and G. Trombettoni. Inter-Block Backtracking: Exploiting the Structure in Continuous CSPs. In *Selected papers of the* 2^{nd} *Int. Worksh. on Global Optimization and Constraint Satisfaction, COCOS, LNCS 3478*, 2005.
41. C. Essert, P. Schreck, and J.F. Dufourd. Sketch-based pruning of a solution space within a formal geometric constraint solver. *Artificial Intelligence*, 124:139–159, 2000.
42. M. Wilczkowiak, G. Trombettoni, C. Jermann, P. Sturm, and E. Boyer. Scene Modeling Based on Constraint System Decomposition Techniques. In *Proc. International Conference on Computer Vision, ICCV*, pages 1004–1010, 2003.
43. B.T. Polyak. *Introduction to Optimization.* Optimizationn Software, 1987.
44. C. McGlone. Bundle adjustment with object space geometric constraints for site modeling. In *Proc.* SPIE *Conf. on Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision II*, volume 2486, pages 25–36, 1995.
45. P. Sturm and S.J. Maybank. A method for interactive 3D reconstruction of piecewise planar objects from single images. In *British Machine Vision Conference, BMVC*, pages 265–274, 1999.
46. E. Grossmann and J. S. Victor. Single and multi-view reconstruction of structured scenes. In *Proceedings of the Fifth Asian Conference on Computer Vision, Melbourne, Australia*, pages 93–104, January 2002.
47. P. McLauchlan, X. Shen, A. Manessis, P. Palmer, and A. Hilton. Surface-based structure-from-motion using feature groupings. In *Proceedings of the Fourth Asian Conference on Computer Vision*, pages 699–705, 2000.
48. D. Bondyfalat and S. Bougnoux. Imposing euclidean constraints during self-calibration processes. In *SMILE workshop on 3D structure from multiple images of large-scale environments, LNCS 1506*, pages 224–235, 1998.
49. P.-L. Bazin. A parametric scene reduction algorithm from geometric relations. In *Vision Geometry IX, SPIE*, 2000.