Towards a better integration of modelers and black box constraint solvers within the product design process

Jean-Philippe Pernot[1] · **Dominique Michelucci**[2] · **Marc Daniel**[3] · **Sebti Foufou**[2,4]

Abstract
This paper presents a new way of interaction between modelers and solvers to support the Product Development Process (PDP). The proposed approach extends the functionalities and the power of the solvers by taking into account procedural constraints. A procedural constraint requires calling a procedure or a function of the modeler. This procedure performs a series of actions and geometric computations in a certain order. The modeler calls the solver for solving a main problem, the solver calls the modeler's procedures, and similarly procedures of the modeler can call the solver for solving sub-problems. The features, specificities, advantages and drawbacks of the proposed approach are presented and discussed. Several examples are also provided to illustrate this approach.

Keywords Geometric modeling · Constraints · Procedural constraints · Solver · Modeler

1 Introduction

Product design is a cyclic and iterative process, a kind of systematic problem solving, which aims at finding a solution integrating the procedural aspects of design with the structural

✉ Sebti Foufou
sfoufou@u-bourgogne.fr

Jean-Philippe Pernot
Jean-Philippe.Pernot@ensam.eu

Dominique Michelucci
dmichel@u-bourgogne.fr

Marc Daniel
Marc.Daniel@univ-amu.fr

1 Arts et Métiers, LISPEN EA 7515, HeSam, Aix-en-Provence, France

2 LE2I, CNRS UMR 5158, University of Burgundy Franche-Comté, Dijon 21000, France

3 LIS Laboratory, UMR CNRS 7020, Aix-Marseille University, Marseille, France

4 Computer Science, New York University, Abu Dhabi, United Arab Emirates

and feasibility aspects of design problems [22]. Design is also a heuristic interactive process that can be adapted to the particular requirements of a task [14]. It is therefore crucial that computer-aided tools adopted during the industrial Product Development Process (PDP) can support users in dealing with these paradigms and associated requirements. Clearly, the PDP is not unique and may vary deeply from company to company, depending on several factors such as the complexity of the product to be designed, the available financial resources, the equipment used, the team of specialists involved and so on. The variability of the PDP is also directly linked to the adopted scenarios and the needs are different if the product has to be designed starting from scratch, or using an existing product to be reverse engineered, or if a version already exists and can be used as a starting point for the development of a new product. Anyhow, structuring a PDP is always more and more critical as product complexity increases.

Despite this variability, a generic structure of a PDP can be devised as a reference scheme that can be adapted to specific companies, scenarios and classes of products. Here, even if alternative paradigms can be imagined [13], it is assumed that a digital model of the product stands as a reference product model. Figure 1 summarizes the structure of such a reference PDP [22]. The arrows represent the flow of information and/or digital models that can be communicated from one activity to another as soon as the first activity has been carried out. Double-headed arrows are not prescribing systematic communications between two or more activities. They can be reduced to single-way communications for some specific scenarios [22].

The PDP involves many actors, tasks, and activities, currently supported by dedicated Information Technology (IT) tools such as: computer-aided design (CAD), computer-aided styling (CAS) [21], computer-aided engineering (CAE), and computer-aided manufacturing (CAM). This process often relies on digital mock-ups (DMUs) that integrates multi-representation and multi-resolution geometric models to shape complex components and products possibly incorporating free-form surfaces [53]. In any case, dedicated adaptation processes have to be developed to answer the needs for multi-representations and multi-resolutions at the different stages of a PDP. Such adaptations also have to take into account

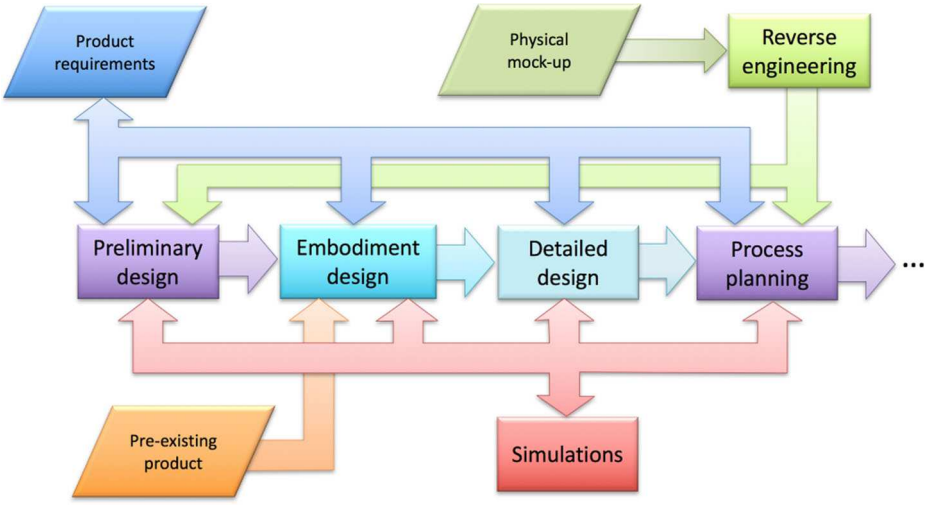

Fig. 1 Generic structure of a PDP and its associated flows of information and/or digital models [22]

interoperability conditions that are key issues when setting up a PDP within a competitive industrial environment. Unfortunately, the ability of the existing tools to interoperate is not yet fully supported and some adaptation steps can for instance take up to several thousands of hours for the preparation of specific simulations in the aeronautic domain [15].

At the end of the PDP, the complete product definition can be seen as the result of an optimization process where various requirements (e.g. functional, aesthetic, economical, feasibility) have to be satisfied so as to get the best admissible and optimized solution. Most of the time, all the requirements cannot be fulfilled and a compromise has to be found. Sometimes, requirements are even conflicting. Moreover, any optimization process calls for more structure between the design and simulation tasks to tightly connect simulation and design updates when component shapes are evolving. Of course, this also refers to the previously identified lack of interoperability. It is illustrated in Fig. 1 with double-headed arrows between design processes and simulation to enforce that CAD and FE models must interact with each other during a shape optimization process.

Consequently, shape modifications and dimension adjustments are common practices that must be supported by the digital models of a product and its components. This statement justifies the need for a modeler to interact with the product characteristics and the need for a solver to find a solution which satisfies the requirements specified by the actors of the PDP. Historically, the first modelers were mainly used to define CAD models so that CNC machines can produce the different parts. Now, CAD models are regarded as product reference data which can be used and adapted all along the PDP and not only to prepare the manufacturing phases. Thus, the modeler has become mainstream in the PDP. Today, even if they can still be improved and notably for the design of free-form shapes, modelers are quite robust and are widespread in the industry. They can support different modeling strategies more or less adapted for design reusability [11]. They can support product models composed of several hundreds of thousands parts. Most of the time, the modeler also encapsulates a solver used to play with the parameters of the CAD models so as to find a solution which best satisfies the requirements. Such an encapsulation is clearly a limitation as it prevents a proper integration of the numerous requirements associated with the different steps of the PDP. It will be further discussed in this paper.

Regarding the solvers, the needs have also evolved. The first attempts have been to solve problems having rather simple geometrical constraints [3, 34]. But the needs for more complex constraints have raised and new taxonomies have been suggested [12]. In the recent years, semantic-based design, which includes declarative modeling, has become an important field of research [2]. The idea is to avoid low-level manipulations of geometric entities, and to be closer to the way the actors of the PDP are thinking and working, i.e. closer to the semantics they manipulate. Today, one major issue is to be able to manage requirements which cannot be expressed as a set of equations. For examples, the maximum of the Von Mises stress should be smaller than 200 MPa, the final product should cost less than $150, the mass of the object should be smaller than 1 kg, there should not be collisions between the parts. These are examples of requirements which cannot be transformed in a set of equations. Thus, the solvers also have to take into account procedural requirements and constraints, i.e. requirements and constraints requiring the call to a procedure or a function. This refers to the notion of black box constraints, which is discussed in this paper.

Thus, there exists a need for developing a new generation of modelers able to handle a broad type of constraints resulting from more or less complex procedures (black box constraints or procedural constraints), that are specified at the different steps of the PDP. Such a modeler must be able to delegate the optimization loop to a solver which might have to call procedures possibly available in different software or libraries used all along the PDP.

Thus, the communication between the modeler and the solver is two-ways and the interoperability is improved. In this article, the modeler plays a central role in the sense that it is used to generate and/or update the CAD models according to geometric commands directly sent by the user, or by the solver. It also supports the call to procedures used to evaluate specific properties of the CAD models. In the context of this work the modeler has to be understood, at a broad sense, as the tool which performs the geometric modifications when changes are requested. It is in charge of maintaining the consistency of the current geometric configuration. A solver finds the values of unknown variables linked by constraints and submitted to optional energy function to be minimized.

This paper contributes to the analysis and understanding of the existing technics and proposes a new approach for a better integration of modelers and solvers within the PDP. The modeler and the solver are separated and can interact, meaning that the solver can call procedures provided by the modeler. However, this separation has to be balanced. Sometimes the modeler and the solver could be two modules of the same software (for instance, in Section 4.4 the CATIA solver is used through plugins on CATIA, while in Sections 4.1 and 4.2 solvers are seen as independent modules but linked to the application), or they could be decoupled (in Section 4.3, UNITY calls an external solver). The contributions of this paper could be summarized as: (i) a framework to better interoperate and to support the interactions between a solver and the numerous tools used along the PDP; (ii) the framework supports both equations and black box constraints. It is compatible with today's tools for detecting conflicts and redundancies between constraints, it can still compute derivatives and exploit the sparsity of the systems; (iii) several examples are presented to show possible implementation architectures for the framework.

This paper is organized as follows. The next section analyzes the current approaches and their limitations. Section 3 details the new proposed approach and discusses the advantages and disadvantages. Section 4 illustrates some of its features through several examples, before the conclusion in Section 5.

2 Limitations of today's approaches

Nowadays, CAD modelers provide their solvers of geometric constraints and usually the solver has its own constraints editor. In some cases the solver may even have one editor for 2D and another for 3D constraints. Basically, the constraints concern vertices of interest, straight lines, planes, circles, spheres, and cylinders whose parameters are the unknown variables. These classical solvers use various types of constraints (such as incidence, tangency, alignment, parallelism, orthogonality, angles and distances) to sketch and constrain the shape of mechanical parts. These solvers translate the constraints into equations. For example, the 2D constraint of distance d between two points (x, y) and (x', y') is translated to the equation $(x - x')^2 + (y - y')^2 - d^2 = 0$. Here, the variable d is either a parameter in which case its value is known to the solver, or unknown in which case the solver will have to compute it.

Those mathematical equations are usually represented using Directed Acyclic Graphs (DAGs). In such a representation, a DAG is a tree with some shared vertices. The leaves of the tree are either variables (i.e. parameters or unknowns) or numerical coefficients. The internal nodes of the tree are either elementary arithmetic operations or functions such as exp, sin, cos, tan. As the mathematical equations associated to geometric constraints are available, it is possible to compute the expressions of the derivatives, and use formal calculus. Indeed, if all the constraints are algebraic it is, theoretically, possible to resort to formal

calculus using standard or Grobner basis. It might also be possible to triangulate the system of equations and put it in the form $f_1(U, x_1) = f_2(U, x_1, x_2) = \ldots = 0$ where U is the vector of parameters and x_i are the unknown variables. When the constraints are expressed in 2D and the problem can be solved by hand using ruler and compass, it is possible to compute the construction plan of the solution figures. The construction plan can be computed using geometric algorithms [3, 4, 36, 39, 51, 59], Artificial Intelligence [17, 18, 20], or even formal calculus such as some variants of Lebesgue algorithm. In practice, geometry learning software applications, for instance GeoGebra [30], resort to formal calculus more than CAD solvers [42, 54]. Finally, DAGs can be translated into C or C++ programs, which are compiled and linked dynamically to numerical solvers and modelers.

Strictly geometric constraints are not sufficient for CAD/CAM. Thus, in late nineties, Hoffmann et al. [35], and Joan-Arinyo [40] proposed an hybrid solving method. The idea is to combine a geometric solver and an equational solver. This hybrid approach also relies on equations. So the solver is still unable to call procedures of the modeler.

Clearly, such an architecture is limiting and is not likely to improve the level of interoperability within the PDP. Often, the product shape results from an optimization problem where the various requirements are specified by the actors involved in the PDP. Those requirements can be of different types and their computation may require the need of external tools or libraries. For example, the shape of a turbine blade is the result of a complex optimization process which aims at finding the best compromise between notably its aerodynamic and mechanical performances. During such an optimization loop, several complex simulations are required and cannot be performed neither by the modeler nor by the solver. Thus, several calls to external software or libraries are performed.

Moreover, the requirements may not always be represented with equations. This is true when specifying constraints on mechanical quantities such as the Von Mises stress which results from a complex Finite Element simulation. But this can also happen when dealing with free-form surfaces which are often obtained tediously from fairly sophisticated modeling functions (e.g. sweep, loft, blend). Thus, solvers have to take into account constraints expressed by procedures or functions. Of course, these so-called black box constraints cannot be manipulated in the same way as if some equations were available. It is for example not possible to compute symbolic expressions of derivatives. However derivatives can still be approximated with finite differences or any other approximation scheme, and exactly evaluated with dual numbers (Section 3.8).

Furthermore, it can be observed that free-form surface models do not benefit from construction tree structures that ease modifications and associate a process with a digital representation of an object [12]. Industrial CAD software rely on an incremental modeling paradigm where a complex free-form shape is incrementally and interactively generated through a sequence of simple shape modeling operations. The chronology of these operations is at the basis of a history tree describing the construction process of an object. Consequently, without a real construction tree, free-form shape modifications are generally tedious and frequently result in update failures. This is even true when dealing with surfaces exchanged through the STEP or IGES standard formats for which the construction tree is not transferred. As a result, it is always difficult to incorporate free-form surface modifications in a shape optimization loop since the impact of parameter modifications are much less predictable on free-form surfaces than on CAD models composed of simple analytical surfaces. Such a limitation reduces drastically the capacity to interoperate and to really integrate the requirements coming from the entire PDP. Sometimes, the modification of some parameters require interactive manipulations which reduce the efficiency of a PDP with a significant cost [62].

The uncertainties the actors have when defining their requirements are also not fully handled by today's modelers and solvers. Product requirements can be subdivided into two complementary categories, qualitative and quantitative ones. Quantitative requirements like a power or a velocity can be subjected to tolerances, which in some sense give a flexibility that can be used to find the compromise. Qualitative requirements like aesthetics cannot be associated with tolerances, and compromises are more subjective, which in some sense can be a good mean to absorb the uncertainties. Even if those examples suggest some possible ideas to better take into account the uncertainties, specific developments need to be undertaken.

Finally, because modifying a shape is also comparable to exploring a solution domain, letting the user know that some shape modifications are just not possible is as important as providing solutions. User-specified requirements may not always be consistent and the overall set can be over-constrained. It is up to the solver to detect those inconsistencies and to give feedbacks on how to remove them. If this is well done in today's 2D sketchers [44], such a level of understanding is not yet fully supported within commercial software. Here again, such a limitation restricts the integration of 3D shapes in the product optimization process since the solver can play with some parameters and can generate non acceptable and not detected configurations which can, at the extreme, cause a crash of the system. We show that all low- or high-level existing methods for detecting conflicts and providing feedbacks to users still apply with the proposed framework. The latter extends their scope.

3 Next generation of modeler-solver interactions

3.1 Towards a new approach

To overcome the limits of today's approaches, a new modeling framework, with a solver handling constraints associated with the different steps of the PDP, is proposed. Figure 2 illustrates this idea of solver-modeler interactions with a solver at the center and other tools and software surrounding it. The modeler calls the solver for solving a main problem, the solver calls the modeler's procedures, and similarly procedures of the modeler can call the

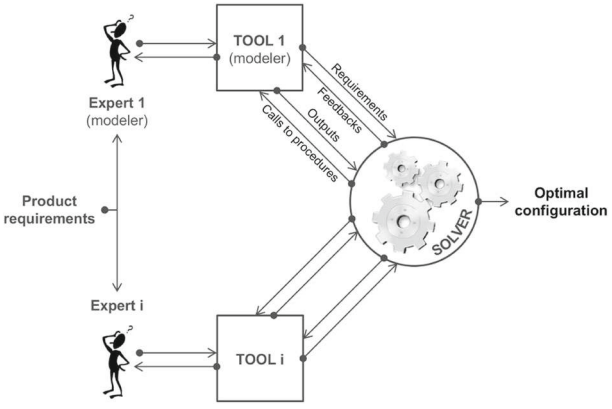

Fig. 2 Framework of the envisioned approach with a solver handling black box constraints associated to the different steps of the PDP

solver for solving sub-problems. Similarly, other tools (e.g. CAE, CAS, CAM software) can also be integrated and do interact with the solver as the modeler does. Thus, the solver interacts with the different tools, and notably with the modeler, using geometric procedures to evaluate the constraints arising from the product and process requirements. The solver sends feedbacks to the different tools. Considering the interactions with the modeler, the solver calls two kinds of procedures: constructive procedures which create shapes, and query procedures which interrogate existing shapes and measure their properties. As a result, the solver receives outputs which can be used during the solving process. All procedures input numerical values, and return numerical values.

Thus constraints are no more represented with equations $F(X) = 0$, for a function F from $\mathbb{R}^m$ to $\mathbb{R}^n$, but only with procedures that compute $F(V)$ for given numerical values $V \in \mathbb{R}^m$. These constraints are called black box constraints [32] or procedural constraints, to emphasize that equations are no more available, contrarily to what happen with classical geometrical solvers. The solver has to find values V such that $F(V)$ returns 0 in $\mathbb{R}^n$. The solver is aware of the signature of the underlying function F: $\mathbb{R}^m \rightarrow \mathbb{R}^n$. The communication protocol between the solver and the modeler is detailed below. However, the user-specified requirements may be inconsistent and the overall set can be over-constrained. Thus, the solver has to detect those inconsistencies and give feedbacks to the experts on how to remove them. The way over-constrained configurations can be detected and treated is discussed in this paper.

This new approach makes sense only if systems of procedural constraints can be efficiently solved. Fortunately, classical fast methods in numerical analysis can be used to solve systems of procedural constraints. Moreover, they permit to exploit the sparsity of systems of procedural constraints.

Integrating a procedural constraints solver into an existing parametric modeler, such as FreeCAD [26] and FreeSHIP [27] for CAD, or Blender [6] for computer graphics, brings several low-cost advantages such as: (i) improving the functionalities of the modeler, (ii) opening more possibilities for the modeler in terms of constraint formulation and solving, (iii) simplifying the solver as well as the modeler in terms of functionalities and usage. Actually, the main features of the proposed approach are discussed below.

The implementation of the proposed approach does not require the development of a new type of modelers, contrarily to the DECO project [32]. Moreover, the solver can use a large set of very sophisticated and efficient geometric procedures available in the modeler or in the PDP software. Few examples of such procedures are the computations of distances, furthest distances, interpenetration depths, bounding boxes, volumes, intersections, Boolean operations between solids, blending, filleting, meshing, reconstructing, etc. In the classical approach, the distance can only be the distance between simple elements (points, lines or planes), with procedural constraints, it can be the distance between complex shapes like assemblies. Figure 3 shows the generalization to three arbitrary objects of Appolonius problem which consists in finding the circle tangent to three given circles. Procedural constraints permit to generalize to three arbitrary objects, solving with BFGS: $d_A(x, y) - R = d_B(x, y) - R = d_C(x, y) - R = 0$. The procedure $d_A(x, y)$ computes the smallest distance from point (x, y) to the object A, whereas x, y and R are the unknowns. It is possible to generalize using greatest distances D_A, D_B, D_C and solving: $(d_A(x, y) - R)(D_A(x, y) - R) = (d_B(x, y) - R)(D_B(x, y) - R) = (d_C(x, y) - R)(D_C(x, y) - R) = 0$.

In the proposed approach, the solver delegates the details and the complexities of the calculations to the procedures of the modeler. Therefore, the solver remains simple and easy to maintain. Similarly, the modeler no longer has to manage the resolution of some constraints: it delegates them to the solver. Thus, both the solver and the modeler are simplified.

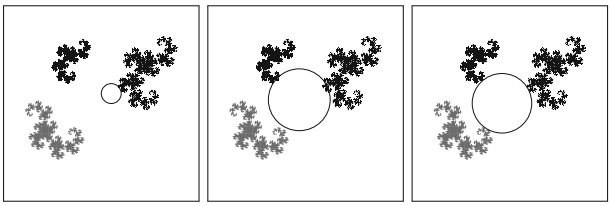

Fig. 3 Generalization of the Appolonius problem to three arbitrary objects. Some iterations of the L-BFGS solver are sufficient to visually converge to a solution

The use of constructive and query procedures permits to dramatically extend the scope of expressible constraints, and the power of the modeler.

This approach helps combining heterogeneous geometric representations such as meshes, implicit analytical surfaces (e.g. quadrics, torii, cyclides), and parametric surfaces such as Bézier or NURBS [32]. When dealing with such heterogeneity, the solver only needs to call the appropriate procedures from the modeler. It is impossible with classical solvers.

Another advantage of this approach is that procedures are definitely much more convenient and powerful than equations. Geometric modelers on the market provide efficient procedures for generating shapes, and for interrogating shapes and sets of shapes (called assemblies, or scenes). For instance, an interrogating procedure computes the point of a given shape S the closest (or the furthest) to a given point p. This procedure may use GPU or accelerating data structures such as octrees, BSP trees, and kd-trees. If S is an unknown shape, it may seem that this procedure can no more be used. But, when a numerical iterative solver is used, it provides (approximate) numerical values for unknown variables (actually modifiable) X at each step. At initialization, components of X^0 are read from the sketch. Thus, after all, it is possible to use procedures of the modeler even when S is unknown. This advantage of procedures over equations also holds when the shape S in the previous example is as simple as a segment (a part of a line), a triangle (part of a plane), or a square (part of a plane), which are the simplest examples of composite figures ubiquitous in CAD/CAM. Procedures are also much more convenient than equations for the constraint: $|M(X)| = 0$, where the determinant of the matrix M must vanish. Entries of M can be piecewise polynomials or rational functions. Unfolding the mathematical expression for $|M(X)|$ has exponential cost, when X are unknowns. But all Linear Algebra packages provide procedures to compute in polynomial time $|M(U)|$ for given numerical values $U \in \mathbb{R}^n$.

One inconvenience of the proposed approach is that the solver can no more use Interval Analysis and interval solvers [38], subdivision methods [19, 24, 28, 50], or Computer Algebra. However, procedures still can use all these methods because their parameter values are known.

Another inconvenience is due to non analytic functions (splines, NURBS, min, max, |.|, functions using if-then-else instructions). Non analytic functions must be taken into account. Procedure compute them in a straightforward way, contrarily to equations. But using non analytic functions has two consequences. Only the second is indeed an inconvenience.

First consequence, deciding dependences of non analytic functions on variables (does $F_l(X)$ depends on X_c?) is non decidable. It is decidable in a probabilistic way for analytical functions, but slow. Thus the best solution is that the modeler informs the solver of the dependences, in other words which entries in the Jacobian are not structurally zero. This information is called the sparsity data or the sparsity graph. It is equivalent to the classical bipartite graph linking equations (now procedures) and unknowns (now modifiable

variables). This graph is used to detect the structurally over-, under- and well-constrained part, and the irreducible well-constrained subsystems in the well-constrained part. Methods in sparse linear algebra also rely on this graph [41]. Today, the interface of some non linear solvers or optimizers does not account for the sparsity data. Assuming that the sparsity data is available, it is still possible (Section 3.6) to exploit the sparsity of systems of constraints in CAD/CAM, i.e., the fact that the results of procedures do not depend on all unknown (actually modifiable) variables. For sure, when considering more requirements, it will be interesting to try to decompose the problem in subproblems which can be treated and analyzed step by step [37].

Second consequence, homotopy solvers which find all roots of polynomial systems [61] are no more usable, for two reasons: first, equations are no more available, and second, even if they were (with DAG), functions computed by procedures are non analytical, thus non polynomial, so homotopy theory no more applies. For example, the fundamental theorem of algebra (a degree d polynomial has d complex roots, i.e., $\mathbb{C}$ is an algebraically closed field) does not apply to piecewise polynomials.

3.2 Technical and theoretical issues

In this section, we mention some technical or theoretical issues of our approach.

First, when solving procedural constraints of type $F(X) = 0$, the solver has no more access to the mathematical expression for the underlying function F (if one exists). The solver can no more compute the mathematical expression of derivatives of F. Yet it can still approximate the values of derivatives with finite differences but this is not accurate enough for the witness method which computes the rank of the Jacobian minors. A significant result is that automatic differentiation (at running time), with the arithmetic of dual numbers [23, 25], computes precise (with the accuracy of floating-point arithmetic) values of derivatives at a given point ($f'(3) = 6$ if $f(x) = x^2$). For instance, if an algorithmic shape depends on, say, a small number $n = 10$ of parameters $U = (u_1, \ldots u_n)$, a FEM simulation using the dual numbers arithmetic and computing a performance $p(u_1, u_2, \ldots u_n)$ will automatically compute in the same time $p(U)$ and all derivatives, i.e., the gradient $\nabla p = (\partial p/\partial u_i(U))$, making possible to search the optimal value U^* which maximizes the performance $p(U)$. Even if the procedure computing the performance $p(U)$ generates for some physical FEM simulation some temporary and huge mesh with $N \gg n$ 3D vertices, e.g., $N = 10^5$, or any other huge data structure, only n dual numbers or infinitesimals are needed, and not $n+3N$.

Second, some procedures compute several functions, like $CalcPt(s, u, v)$ which computes three coordinates $x_{u,v}, y_{u,v}, z_{u,v}$ of the point with parameters u, v on a surface with control net s. Some book-keeping is needed to avoid multiple evaluations. Either some memorization (called memoization in functional programming) is used, or the solver calls some handle function update(X) to enable the modeler to update its internal data structures, before the solver evaluates procedural constraints.

Third, some functions may have an incomplete definition domain, for example $CalcPt(s, u, v)$ which assumes $(u, v) \in [0, 1]^2$. This problem already occurred and is solved in previous examples like DECO [32]. A solution is as follows: the procedure $CalcPt(s, u, v)$ clamps its arguments u and v, but it does not modify the values of variables u and v (only the solver can do that). The inequality constraints on u and v can be either reduced to equations using slack variables ($(u - 1/2)^2 + u_s^2 - 1 = 0$ with u_s the slack variable for u), or are bounds constraints [10] in some constrained optimization problem. Byrd et al [10] propose a variant of L-BFGS for solving constrained optimization problems like: min $G(X)$ with $L \leq F(X) \leq U$.

Fourth, discontinuity issues might be another difficulty for the proposed approach. As an example, let's consider the function ClosestPt(S, p), which returns the closest point to $p \in \mathbb{R}^3$ in or on the shape S, is not continuous everywhere, e.g., when S is not convex. In practice, the consequence of such discontinuity is that the solver must start from an initial guess close enough to the expected solution. Other source of discontinuity may include staircases, toothed wheels and frameworks, which are typical parametric or algorithmic shapes. The number of parts of parametric or algorithmic shapes depends on parameter values. For example, the number of steps, and the number of intermediate bearings of a staircase depend on the height of the floor; the number of toothed wheels depends on the size of the gear; the number of bars and nodes of frameworks or lattices depends on their spans; the number of vertices of a control net of a NURBS may depend on the resolution; etc. Every algorithmic shape (or technical feature) could be specified in a Unified Technical Document (or a standardized format e.g. STEP) to be implemented using a procedure. Toothed wheels are an example of algorithmic shapes that have been implemented in a procedure and constrained in the DECO project [32].

Fifth, the interface, or the communication protocol, between the modeler and the solver must permit to benefit from the sparsity of procedural constraints. There are mainly two kinds of interfaces. The first interface is used in the DECO project: the modeler and the solver use black DAGs (Directed Acyclic Graphs). This first interface permits to exploit the sparsity of procedural constraints. However, many libraries such as GNU GSL or Scipy, which provide solvers or minimizers, use the second kind of interface, as follows. To solve a system $F(U, X) = 0$ with $U = U_T$, the value of U (for target T). The solver typically receives three arrays F, U, X: F is an array of pointers to procedures computing $F(U, X)$ (procedural equations are: $F(U, X) = 0$), U is the array of floating-point values for parameters, X is the array of initial values for the unknowns X. The solver can modify X, but not U. Sometimes the solver accepts an array (of pointers to procedures) F' for computing the gradient vectors ∇F_i. It can also use finite differences to approximate derivatives. The solver also receives technical information such as array sizes, threshold values for termination tests, a maximum number of iterations, etc. The same kind of interface applies for constrained or unconstrained minimization problems. Finally, the solver receives some handle or callback functions, e.g., for drawing pictures of the current figure in our context; it can permit users to monitor and drive the resolution process. Clearly, this second interface is not sufficient for exploiting sparsity. Another array D is needed for specifying dependences: $D[c]$ is the list of all (index of) variables X_k on which the constraint F_c depends. D can be seen as a sparse matrice, and its transpose T can be used instead: $T[k]$ is the list of (index of) constraints depending on X_k. These two arrays are clearly equivalent. They are sufficient to exploit sparsity. They are already used for that purpose in the interface of many Sparse Linear Algebra libraries, for instance to compute fill-reducing ordering of unknowns [28, 56]. These arrays represent the bipartite graph equations-unknowns used in matching theory (Dulmage-Mendelsohn decomposition).

3.3 Solving without equations

This section discusses the scalability of our approach through exploiting sparsity. The solver must solve without equations the underlying systems $F(X) = 0$, starting from an initial guess close enough to the expected root. This proximity was already needed with classical geometric constraints, i.e. when equations are available. In practice, users provide this initial guess with a sketch.

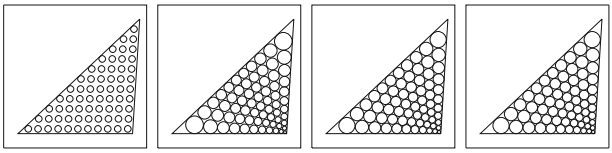

Fig. 4 Solving with Newton (11 rows).

We implemented and tested several solving methods to be used in the proposed approach: Newton, Levenberg-Marquardt [29], L-BFGS (Limited memory Broyden-Fletcher-Goldfarb-Shanno) [7], Hooke-Jeeves, stochastic descent, and the Jaya heuristic [55]. Some methods need gradients. They are computed with centered finite difference (not with dual numbers).

The methods have been tested on this geometric problem: let ABC be a given triangle; let r be an integer; place r rows of circles in ABC, with one circle in the first row, k circles in row k, so that circles are outwards tangent to their neighbors or to the sides of ABC. See Figs. 4 and 5. For r rows, there are $r(r+1)/2$ circles and 3 times more unknowns: $n = 3r(r+1)/2$ as each circle k has unknown center coordinates (x_k, y_k) and unknown radius r_k. This is a well-constrained problem as the number of constraints is equal to the number of unknowns $n = 3r(r+1)/2$. It is also irreducible: it has no well-constrained (nor rigid) subsystem. Though equations (circle-circle tangencies, or line-circle tangencies) are available, they are not accessible to the solver: the solver is only aware of an array of n (pointers to) procedures $F[0, \ldots n-1]$, and an array of n unknowns V which are the concatenation of tuples (x_k, y_k, r_k) for $k = 0, \ldots n-1$. The initial values for (x_k, y_k, r_k) are obtained as follows: this problem is easy to solve when the triangle ABC is equilateral, which gives barycentric coordinates a_k, b_k, c_k (with $a_k + b_k + c_k = 1$) for the center of circle k: $(x_k, y_k) = a_k A + b_k B + c_k C$. They give an initial guess for X.

The test problem described in the previous paragraph is artificial but very convenient for measuring performances, and controlling the behavior and output of algorithms, due to its visual and intuitive nature. Figure 4 shows Newton iterations for 11 rows, and Fig. 5 some Hooke-Jeeves iterations. Table 1 shows the figures with more rows, and Table 2 gathers together the empirical complexities of the tested algorithms. The empirical complexity for an algorithm is the slope of the curve (very close to a line) of the log-log diagram $(\log n_i, \log T(n_i))$, where n_i is the number of unknowns, $T(n_i)$ is the running time of the algorithm. We tested with 50, 100, 150, 200, and 400 rows. L-BFGS has the best empirical complexity: $O(n^{1.33})$. Then Newton and Levenberg-Marquardt have complexity $O(n^{1.4})$. Hooke-Jeeves has complexity $O(n^{1.9})$. Neither parallelism nor GPU acceleration is used. All these complexities are less than $O(n^2)$, the size of a dense Jacobian. To reach them, it is essential to exploit the sparsity of systems of procedural constraints. Remember that, for multiplying two dense matrices, or solving a dense linear system, the complexity of the

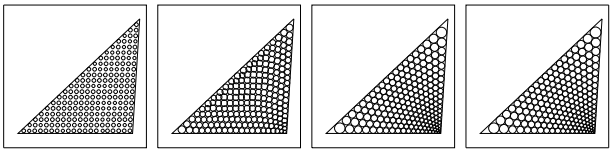

Fig. 5 Solving with Hooke-Jeeves (20 rows).

Table 1 Number of circles and unknowns for the test problem

nb of rows	50	100	150	200	300	400
nb of circles	1275	5050	11325	20100	45150	80200
nb of unknowns	3825	15150	33975	60300	135450	240600

famous Strassen method is $O(n^{2.807})$, and the complexity of the Coppersmith-Winograd algorithm (the best method known so far, but unused in practice) is $O(n^{2.376})$. Even the Hooke-Jeeves method is much better, and lower than $O(n^2)$. Other methods: stochastic gradient, and Jaya either diverge, or converge with damping but are too slow, so we will not comment their complexity.

In our experiments, classical and heuristics enhanced solvers such as Jaya and stochastic descent did not perform well with the test problem. However, turning to meta-heuristics could be interesting when there is a huge set of solutions for which users have no a priori preference, but they can reject a bad solution, even if it is difficult for them to say why; or it is too time consuming to explicit all constraints and preferences. A similar problem is met in forensic: some witnesses can not describe faces or persons, though they can recognize them. To solve this forensic issue [60], a software (like EvoFIT) generates a set of 20 pictures of random faces; the witness chooses the faces which resemble the most the target face. The software combines parts of selected faces, proposing a new set of 20 faces to the witness. After some rounds, a resembler portrait is reached. One may imagine to use the same method for choosing a solution amongst a huge set. The modeler generates 20 solutions, for instance starting from 20 random seeds, improved with some Newton iterations or some minimization; users reject too bad solutions; maybe they can say which part is good in some solution and should be kept. The modeler combines selected solutions, as in genetic algorithms, and improves each new combination to satisfy the procedural constraints, with Newton iterations or a Minimizer. One may expect to find a good solution after some rounds.

3.4 Existing tools for qualitative study

It is essential to provide users with qualitative study tools for debugging constraints, detecting errors and fixing them, for decomposing systems into sub-systems, and for exploiting sparsity to speed-up the solving process. We mention hereunder three kinds of tools.

Methods in the first kind call the solver. We call them protocols. Many protocols have been suggested to detect contradicting, or conflicting subsystems. Remember that an over-constrained system can be contradicting or redundant. Here is an example of such a protocol: start from a figure close to the expected solution, and solve constraints incrementally. Let

Table 2 n is the number of circles or of unknowns and constraints (and not the number of rows)

This table gives the empirical complexity (the slope of log-log diagram) of algorithms for the test problem, exploiting sparsity

Algorithm	Complexity
L-BFGS ($m = 10$)	$O(n^{1.33})$
Newton	$O(n^{1.4})$
Levenberg-Marquardt	$O(n^{1.4})$
Hooke-Jeeves	$O(n^{1.9})$

F_k be the first non-satisfied constraint, then $C = \{F_1, \ldots F_k\}$ is contradicting. Then remove from C every constraint F_i such that $C - F_i$ is still non-satisfied. This protocol gives the smallest contradicting subsystem and can be used when equations are not available.

The two other kinds of tools do not call the solver.

The second kind is a set of combinatorial (or structural) methods [4, 33, 39, 51, 59] which consider various graphs. These methods do not call the solver. For simplicity, this article considers only the sparsity graph [33, 59]. It is a bipartite graph linking equations (now procedural constraints) and unknowns (now modifiable variables). Combinatorial methods rely on Matching Theory [47]. They compute maximum matching in the sparsity graph. They detect structurally under, over and well-constrained parts. They also detect structurally under, over and well-constrained parts modulo isometries, e.g., a triangle is well-constrained modulo isometries by three constraints. In both cases, they decompose into structurally irreducible parts. They still can be used when constraints are represented with procedures rather than equations, as explained below (Section 3.5).

But combinatorial methods are limited: they detect only structural conflicts. Moreover a combinatorial characterization of rigidity (well-constrainedness modulo isometries) is still unknown for general geometric constraints and its existence is questionable. The combinatorial characterization of rigidity is the topic of Rigidity Theory. The latter considers only generic point-point distances, which is insufficient for CAD/CAM.

The third and last kind of tools is the witness method [48] and its variants [37, 49, 63]. The principle of these methods is as follows: suppose we want to solve $F(U, X) = U - U_T = 0$, where U are names of parameters (unmodifiable variables), U_T is the value of U (T for target), X are names of unknown (modifiable) variables. A witness is a couple U_W, X_W such that U_W and X_W are vectors of numerical values and such that $F(U_W, X_W) = 0$. Moreover it is assumed that the witness is typical of (or even very close [37] to) the target, so that the witness and the target share the same combinatorial properties. More precisely, the ranks of each minor in the known witness Jacobian $F'(U_W, X_W)$ and in the unknown target Jacobian $F'(U_T, X_T)$ ($F'(U_T, X_T)$ is unknown because exact values of X_T are unknown) are equal. Under mild assumptions of typicality and exactness (consider that for the sake of simplicity, there is no inaccuracy), the witness method detects all dependences between constraints: it is more powerful than structural methods. In practice, users are in charge of deciding the typicality: for instance, a flat triangle (or a flat polyhedron) is not typical of a triangle (of a polyhedron), and this assumption is not an issue. The simplifying exactness assumption is not practicable and is more problematic, but Hao Hu et al. [37] recently proposed tools to account for the unavoidable numerical inaccuracy, when using the witness method for modelling free-form curves and surfaces. The witness method can still be used when constraints are represented with procedures rather than equations, as explained below in Section 3.6.

3.5 Building the sparsity graph

We explain here how the sparsity graph, equivalent to the standard bipartite graph equation-unknown, can be constructed.

Let $F(x, y) = b_1$, $G(x, y) = b_2$, $H(y, z) = b_3$ be a sparse and linear or linearized system with $F(x, y) = ax + by$, $G(x, y) = cx + dy$, $H(y, z) = ey + fz$. Figure 6 illustrates the bijection between the two perfect matchings adf and bcf of the bipartite graph and the two terms in the determinant $adf - bcf$. The latter does not depend on e, because none of the perfect matchings contains edge e. Matching theory [4, 44, 47, 59] also permit to detect that $(F, G)(x, y)$ is a subsystem.

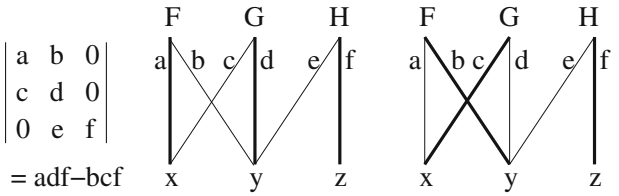

Fig. 6 Determinant matching with bipartite graph. Edges in the perfect matchings are bold

Consider a procedure computing a function $F : X \in \mathbb{R}^n \rightarrow Y = F(X) \in \mathbb{R}^m$. Combinatorially, it is equivalent to m equations f_i: $Y_i - F_i(X) = 0, i = 1, \ldots m$. As usual, there is a vertex for each f_i, for each $Y_i, i = 1, \ldots m$, and for each $X_k, k = 1, \ldots n$. One edge links f_i to Y_i, and n edges link f_i to $X_1, \ldots X_n$. Then structural methods apply.

3.6 Interface for using the witness method

For the witness method to apply [43, 48, 63], the modeler (or any procedure calling the solver) must provide a witness: either a previous release of the product, or a computed witness [43]. It is the users who decide if the witness is typical of the expected solution. Also, to enable the witness method to detect rigid subsystems, variables must be tagged so that the witness method can compute an a priori basis for infinitesimal rigid body motions; again, infinitesimals, or dual numbers, appear [23, 48]. This a priori basis is independent of the constraints, it depends only on the kind of variables, more precisely on the kind of unknown coordinates. Tags give the kind of coordinates of the tagged variable when it is a coordinate: values of coordinates depend on the used Cartesian frame. A tag, i.e, a coordinate can be:

- the x, the y or the z of a point. The x, y, z of the same point must be linked together in some way, e.g., the variable X_3 is the x of the point (X_3, X_4, X_5).
- the x, the y or the z of a vector. Indeed, vectors and points are different because they do not behave the same under translations. The x, y, z of the same vector must be linked in some way.
- the a or b or c of a normal vector. Vectors and normal vectors are different: a vector is the difference between two points. A normal vector is associated to a linear form: $(x, y, z)^t \rightarrow (a, b, c)(x, y, z)^t$. Actually, the a, b, c is the vectorial part of the equation of a plane (see below). The a, b, c of the same normal vector must be linked in some way.
- the a or b or c or d of the equation of a plane: $ax + by + cz + d = 0$. The a, b, c, d of the same plane must be linked together.
- a geometric variable which is independent on the Cartesian frame: radius or length, area, volume, scalar product.
- finally a variable can be a non geometric variable, thus independent on the Cartesian frame: energy, force, cost, etc.

The last two tags can be merged. Tags permit the witness method to decompose systems into rigid (i.e., well-constrained modulo isometry) sub-systems. It is also possible to tag variables to account for similitudes but this decomposition is more rare [57, 58].

3.7 Mathematical necessary conditions

This paragraph summarizes necessary mathematical conditions for the solver to find a solution, or to detect conflicts and help users to fix them. The necessary conditions are classical and do not change with our approach.

It is assumed that procedures are correct and consistent, for instance parameters of each procedure are independent, and, better, orthogonal in some sense. Of course, it is assumed that procedures are not malevolent, to avoid the Byzantine problem.

Functions computed by procedures must be smooth (like a distance function) or smooth almost everywhere (like the closest point function, i.e., the orthogonal projection of a given point on a given shape). This mathematical condition can be relaxed for algorithmic shapes, e.g., gearwheels [32], as already mentioned. Also, the sketch should be close enough to the expected root.

The modeler must provide the sparsity data or sparsity graph to the solver, and a typical witness, so that the solver can use classical qualitative study methods. Users are in charge of deciding the typicality. Remember that a witness is a sketch, but the converse is not mandatory.

When some cost or energy function $G(X)$ has to be minimized, it is assumed that G is as smooth as possible, that $G(X)$ becomes infinite when $\|X\|$ becomes infinite and that $G(X)$ has a finite lower bound [7].

3.8 Dual numbers

Derivatives are key components in many geometric computations. Dual numbers have been implicitly used in the witness method [25] and in the dual quaternions. This section introduces dual numbers and shows how they can be used to compute derivatives exactly [23], with the accuracy of floating-point arithmetic, even when equations are not available. This accuracy is needed by the witness method [25]. Dual numbers are not new, but to the best of our knowledge, the way we use them in geometric modeling has never been done before. Within the proposed paradigm, dual numbers allows the computation of the derivatives even when the equations are not available.

Dual numbers are best understood with an analogy with complex numbers ($\mathbb{C}$). For a computer scientist, writing a C++ library for an arithmetic for dual numbers and for complex numbers is almost the same.

A complex number z is a pair of two real numbers ($x \in \mathbb{R}, y \in \mathbb{R}$). The pair $(0, 1)$ is called i. The part x of z is its real part, and y its imaginary part. The addition of two complex numbers $z = (x, y)$ and $z' = (x', y')$ is defined by

$$(x, y) + (x', y') = (x + x', y + y')$$

Thus it is consistent to write the pair $z = (x, y)$ as $(x, 0) + (0, y) = x(1, 0) + y(0, 1) = x1 + yi$. The product of z and z' is defined by

$$(x, y) \times (x', y') = (xx' - yy', xy' + yx')$$

thus $i^2 = (0, 1) \times (0, 1) = (-1, 0) = -1$. Actually, reducing i^2 to -1 gives another path to the product rule :

$$\begin{aligned}(x + yi) \times (x' + y'i) &= xx' + yy'i^2 + (xy' + yx')i \\ &= (xx' - yy') + (xy' + yx')i\end{aligned}$$

There is a remarkable isomorphism $\phi_{\mathbb{C}}$ between $z \in \mathbb{C}$ and the 2×2 real matrice

$$\phi_{\mathbb{C}}(z) = \begin{pmatrix} x & -y \\ y & x \end{pmatrix}$$

Indeed, $\phi_{\mathbb{C}}(z + z') = \phi_{\mathbb{C}}(z) + \phi_{\mathbb{C}}(z')$ and $\phi_{\mathbb{C}}(z \times z') = \phi_{\mathbb{C}}(z) \times \phi_{\mathbb{C}}(z')$.

A dual number resembles a complex number. It is a pair of two real numbers ($x \in \mathbb{R}$, $y \in \mathbb{R}$). The pair (0, 1) is called ϵ and can be thought as an infinitesimal number. x is the standard part of the pair (x, y), and y its infinitesimal or non standard part. The addition of two dual numbers (x, y) and (x', y') is defined by

$$(x, y) + (x', y') = (x + x', y + y')$$

Thus it is consistent to write the pair $z = (x, y)$ as $(x, 0) + (0, y) = x(1, 0) + y(0, 1) = x1 + y\epsilon$. The product of $z = x + y\epsilon$ and $z' = x' + y'\epsilon$ is defined by

$$(x, y) \times (x', y') = (xx', xy' + yx')$$

thus $\epsilon^2 = (0, 1) \times (0, 1) = (0, 0)$. Actually, reducing ϵ^2 to 0 gives another path to the product rule :

$$\begin{aligned} (x + y\epsilon) \times (x' + y'\epsilon) &= xx' + (xy' + yx')\epsilon + yy'\epsilon^2 \\ &= xx' + (xy' + yx')\epsilon \end{aligned}$$

This time, the isomorphism ϕ between the dual number $z = x + y\epsilon$ and the 2×2 real matrix $\phi(x + y\epsilon)$ is defined by:

$$\phi(z) = \begin{pmatrix} x & 0 \\ y & x \end{pmatrix}$$

Indeed, $\phi(z + z') = \phi(z) + \phi(z')$, $\phi(z \times z') = \phi(z) \times \phi(z')$, thus $\phi(1/z) = \phi(z)^{-1}$ and $\phi(z^k) = \phi(z)^k$. Let us detail the product:

$$\begin{array}{ccccc} (a + b\,\epsilon) & \times & (a' + b'\,\epsilon) & = & aa' + (ab' + ba')\,\epsilon \\ \downarrow & & \downarrow & & \downarrow \\ \begin{pmatrix} a & 0 \\ b & a \end{pmatrix} & \times & \begin{pmatrix} a' & 0 \\ b' & a' \end{pmatrix} & = & \begin{pmatrix} aa' & 0 \\ ba' + ab' & aa' \end{pmatrix} \end{array}$$

From this isomorphism, we deduce that:

$$\frac{1}{a + b\,\epsilon} = \frac{1}{a} - \frac{b}{a^2}\,\epsilon \text{ when } a \neq 0$$

thus $b\epsilon$ has no inverse (the associated matrix is not invertible). This rule is a special case of:

$$(a + b\epsilon)^k = a^k + ka^{k-1}b\,\epsilon$$

If P is a polynomial, then $P(x_v + \epsilon)$ where x_v is a floating-point number, gives $P(x_v)$ and the derivative $P'(x_v)$:

$$\begin{aligned} P(x_v + \epsilon) &= a(x_v + \epsilon)^3 + b(x_v + \epsilon)^2 + c(x_v + \epsilon) + d \\ &= a(x_v^3 + 3x_v^2\,\epsilon) + b(x_v^2 + 2x_v\,\epsilon) + c(x_v + \epsilon) + d \\ &= (ax_v^3 + bx_v^2 + cx_v + d) + (3ax_v^2 + 2bx_v + c)\,\epsilon \\ &= P(x_v) + P'(x_v)\,\epsilon \end{aligned}$$

It extends to multivariate polynomials: either we have only one ϵ and two evaluations are needed:

$$\begin{aligned} Q(x_v + \epsilon, y_v) &= Q(x_v, y_v) + Q'_x(x_v, y_v)\epsilon \\ Q(x_v, y_v + \epsilon) &= Q(x_v, y_v) + Q'_y(x_v, y_v)\epsilon \end{aligned}$$

or each variable is attached its own ϵ and one evaluation suffices:

$$\begin{aligned} Q(x_v + \epsilon_x, y_v + \epsilon_y) \;=\; & Q(x_v, y_v) + Q'_x(x_v, y_v)\epsilon_x + \\ & Q'_y(x_v, y_v)\epsilon_y \end{aligned}$$

Actually, this feature (computing $(P(x), P'(x))$ together) extends to non polynomial functions. The arithmetic of dual numbers is used to compute $f(t)$ and its derivative $f'(t)$ in the same time: here f is a continuous and derivable function (at least at value t), and $f'(t)$ is the value of the derivative of f at $t \in \mathbb{R}$. The idea is to represent the pair $(f(t), f'(t))$ with a dual number $f(t) + f'(t)\epsilon$. It is possible because the arithmetic of dual numbers mimics the rules for derivatives of sums and products (caution: the primes in f', g' denote the derivatives):

$$(f, f') + (g, g') = (f + g, f' + g')$$

and

$$(f, f') \times (g, g') = (f \times g, f \times g' + f' \times g)$$

In a library for complex numbers, we have to define exp, cos, etc for complex numbers. Idem for dual numbers. This definition is straightforward: for any f, $f(a + b\epsilon)$ equals by definition $f(a) + b\epsilon f'(a)$, thus:

$$\begin{aligned} \exp(a + b\epsilon) &= e^a + be^a\,\epsilon \\ \cos(a + b\epsilon) &= \cos(a) - b\sin(a)\,\epsilon \\ \sin(a + b\epsilon) &= \sin(a) + b\cos(a)\,\epsilon \\ \tan(a + b\epsilon) &= \tan(a) + b(1 + \tan^2(a))\,\epsilon \end{aligned}$$

The definition of the function $\mathrm{sgn}(a + b\epsilon)$ is easy:

$$\mathrm{sign}(a + b\epsilon) = \mathrm{sign}(a) + (1 - \mathrm{sign}(a)^2)\mathrm{sign}(b)$$

where $\mathrm{sgn}(v)$ is the sign of $v \in \mathbb{R}$; It is -1 if $v < 0$, 1 if $v > 0$ and 0 if v is zero. Then the definition of the absolute value follows:

$$|a + b\epsilon| = \mathrm{sign}(a + b\epsilon) \times (a + b\epsilon)$$

The definitions of the min and max of two dual numbers could be formulated in a similar way. In passing, symbolic differentiation (at compile time) can not deal so nicely with functions $|.|$, min, max and if-then-else constructs.

Dual numbers permit to compute the derivative of $D(X) = \det(M(X))$, for square matrices $M(X)$, even if entries of M are piecewise polynomials, or algorithms: just replace floating point numbers with dual numbers and then use any standard numerical method (Gauss pivot, LUP). There are also formulas.

Lemma *$det(I + \epsilon M) = 1 + Trace(M)\,\epsilon$, where M is standard:*

$$\begin{aligned} det(I + \epsilon M) \;&=\; (1 + M_{11}\epsilon)(1 + M_{22}\epsilon)\ldots(1 + M_{nn}\epsilon) + R \\ &=\; 1 + Trace(M)\,\epsilon + R \end{aligned}$$

where R represents the other perfect matchings in $I + \epsilon M$. But other perfect matchings use at least two off-diagonal entries in $I + \epsilon M$, thus are multiples of ϵ^2, thus are zero.

When A is invertible, $det(M(x+\epsilon)) = det(A+\epsilon B)$ is:

$$\begin{aligned} det(A+\epsilon B) &= det(A(I+\epsilon A^{-1}B)) \\ &= det(A)\det(I+\epsilon A^{-1}B) \\ &= det(A)(1+Trace(A^{-1}B)\,\epsilon) \end{aligned}$$

When A is not invertible, we use its SVD : $A = U\Sigma V^t$ (with Σ diagonal and U, V unitary):

$$\begin{aligned} det(A+\epsilon B) &= det(U\Sigma V^t + \epsilon B) \\ &= det(U(\Sigma V^t + \epsilon U^t B)) \\ &= det(U(\Sigma + \epsilon U^t B V)V^t) \\ &= det(\Sigma + \epsilon U^t B V) \end{aligned}$$

equals the product of diagonal entries of $\Sigma + \epsilon U^t B V$. It is 0 when there are at least two null singular values in Σ. Otherwise it is

$$(\sigma_1 + k_1\epsilon)\ldots(\sigma_{n-1} + k_{n-1}\epsilon)(0 + k_n\epsilon) = 0 + \sigma_1 \ldots \sigma_{n-1}k_n\,\epsilon$$

The main mathematical difference between complex numbers and dual numbers is that complex numbers form a (commutative) field, while dual numbers only form a commutative ring: they are not a field because there are non zero divisors of zero (ϵ and all $b\epsilon$ for $b \in \mathbb{R}$).

This arithmetic can be generalized in two directions. First it is possible to manage many infinitesimals. The idea is to attach an infinitesimal number ϵ_i to each unknown x_i with the rule $\epsilon_i^2 = \epsilon_i\epsilon_j = 0$. Note that one ϵ is sufficient for computing directional derivatives, needed for example in line-search for BFGS. Second, it can be interesting for some applications to replace the rule $\epsilon^2 = 0$ with the rule $\epsilon^3 = 0$ or $\epsilon^4 = 0$.

We conclude this section by showing the relevance of dual numbers for geometric computations. An algebraic construction ϕ starting from $\mathbb{R}$ gives the quaternions, which represents 3D rotations. If ϕ is applied to $\mathbb{R} + \epsilon\mathbb{R}$, it gives biquaternions also called dual quaternions, which represents both 3D rotations and translations.

4 Examples highlighting the new approach

The previous section presented the new approach and discussed its specificities. It showed that the solver can still perform the required computations even when equations are unavailable or irrelevant. This section introduces four illustrating examples. In the three first examples, the solver has to solve the following constrained minimization [5] problem $\mathcal{P}$:

$$\begin{cases} F(X) = 0 \\ \min G(X) \end{cases}$$

where X is the vector of unknowns, $F(X)$ is the vector of constraints, and $G(X)$ is an objective function required when the constraints do not necessarily yield a unique solution. Depending on the examples, the unknowns, the constraints, and the objective functions will be different. The last example is slightly different since it corresponds to an attempt to create a declarative modeler on top of an existing modeler. In our prototyping and experiments, we have implemented simple classical algorithms [5, 7] (typically, minimizing Augmented Lagrangian Merit Functions) and sometimes used Matlab or Mathematica. A professional implementation could use WORHP ("We Optimize Really Huge Problems"), a recent library [9, 46] solving such problems.

Table 3 Characteristics of the four proposed examples with respect to criteria on: the mixing of geometric models, the use of DAG with black boxes, the type of interaction modelersolver, the possibility to use any solver, the level of the constraints

Example	Mixed	Black boxes	Integration	Solvers	Level
#1	yes	yes	P	yes	high
#2	no	no	P	yes	high
#3	yes	no	E	yes	high
#4	no	no	I	no	high

As explained in Section 3, the tools can interact with the solver in multiple ways. This is summarized in Table 3 wherein each example is characterized with respect to four criteria. The first column indicates if the application mixes different geometric models (e.g. NURBS, subdivision surfaces, canonical surfaces). The second column indicates the full use of DAGs with black boxes. The third column deals with how the modeler and the solver are interconnected. Letter P means that both are connected in the application for convenience purposes (Package). Letter E corresponds to a solver which is completely outside the modeler (External). Letter I corresponds to a full integration of the modeler and the solver (Integrated). The next column indicates the possibility to freely apply different solvers. The last column proposes a (subjective) judgment of the abstraction level of the given constraints. Here, high means that the user does not have to manipulate low-level constraints or equations, but he/she can specify higher level requirements.

4.1 On the use of black box constraints and DAGs

The first example is given by the DECO project [32]. The designer enters in Python (still at a rather low level of abstraction) a description of a shape that is converted into a DAG used to solve the problem and to generate the desired shape. The proposed approach has been validated with a modeler and solver totally implemented from scratch [32]. The system accepts usual DAGs (i.e. the so-called white DAGs) as illustrated in Fig. 7 but also more complex DAGs (i.e. the so-called black DAGs) where some nodes can be procedures (e.g. written in C++).

To illustrate this approach, a very simple example and its associated black DAG are given on Figs. 8 and 9. A B-spline surface defined by a network of control points (bottom of Fig. 8) has to pass through a user-specified point p of coordinates (x_p, y_p, z_p). Some control points can freely move and their coordinates are the unknowns X of the optimization problem (grey nodes on Fig. 8). The others are fixed and correspond to the surrounding

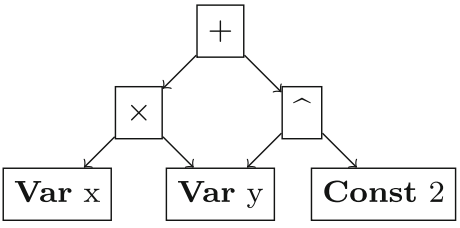

Fig. 7 Example of a white DAG composed of scalar operators, variables and constants, representing the expression $xy + y^2$ [32]

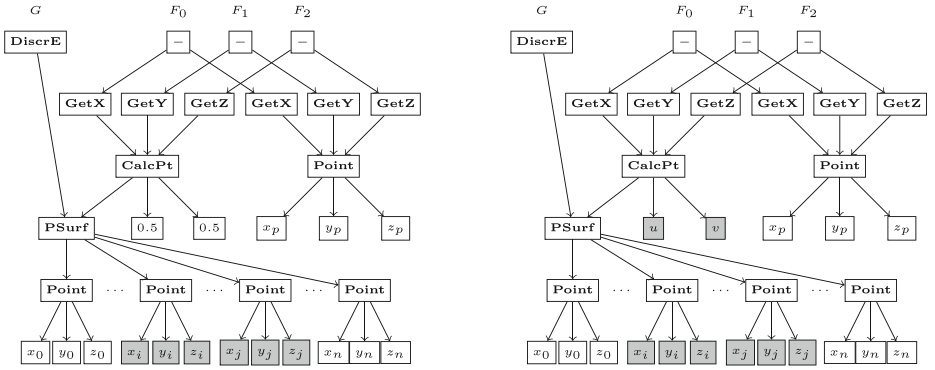

Fig. 8 Black DAGs representing the optimization problems to be solved, when (u, v) is constant (left) and variable (right) [32]

control points visible in Fig. 9. This specification can be described by three functions F_0, F_1 and F_2 which have to be equal to 0.

$$\begin{aligned} F_0(X) &= \text{GetX}(CalcPt(s, u, v)) - \text{GetX}(p) = 0 \\ F_1(X) &= \text{GetY}(CalcPt(s, u, v)) - \text{GetY}(p) = 0 \\ F_2(X) &= \text{GetZ}(CalcPt(s, u, v)) - \text{GetZ}(p) = 0 \end{aligned}$$

The function $CalcPt(s, u, v)$ is written in C++. It returns the 3D coordinates of a point of parametric coordinates (u, v) of a B-spline surface s.

As the solution is not unique, a function $G(X)$ has to be minimized. Here, it corresponds to the discrete energy of the surface computed on its control polyhedra [32]. This function is also written in C++. This specification gives rise to the black DAGs of Fig. 8. Two distinct cases have been tested. On the left part, the parametric coordinates of the point are fixed and equal to $(0.5, 0.5)$. On the right part, the parametric coordinates (u, v) are not known and correspond to the grey nodes. In both cases, the solver must work without equations. The results are proposed in Fig. 9.

Other examples [32] in DECO combine heterogeneous geometries: NURBS, subdivision surfaces and algorithmic shapes (toothed gears), and involve hundreds of unknowns.

The DECO project shows the feasibility and the promises of variational geometric modeling with black DAGs, i.e. without equations. DECO examples do not cover all the potential cases in geometric design but the reader can imagine that different or more complex examples can be treated with the same approach. The energy to minimize can be easily changed

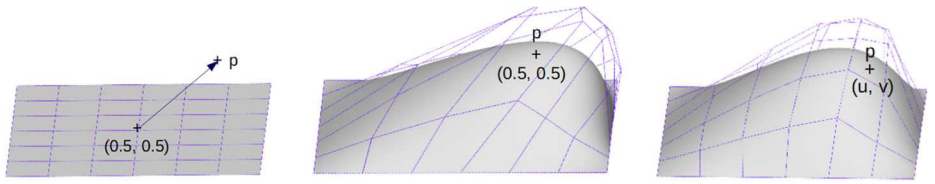

Fig. 9 Results corresponding to the problems of Fig. 8: starting point (left), final position when (u, v) is constant (center) or variable (right) [32]

depending on the user's goal. The user can also create simple constraints and use them to create his/her more elaborated ones thus extending drastically the constraints toolbox. Ideally, the proposed solver could benefit from higher-level procedures available in commercial software packages. This will be further illustrated in the example of Section 4.4.

4.2 Aesthetic-oriented modifications of free form curves

Curves are widely used within the PDP and many approaches try to develop manipulation technics closer to the way designers and stylists think and work [1, 8, 64]. The idea is to try to get rid of the underlying mathematical models while focusing on high level shape description procedures which transform a description in a set of procedures and/or equations. In this context, deformation operators enabling direct aesthetic modifications have been developed [31]. The deformation results from the resolution of the previously introduced optimization problem $\mathcal{P}$ where X are unknown coordinates of the control points. The constraints are defined by aesthetic properties which can be translated to a vector of equations $F(X)$. For example, the designer can change the straightness S (or non-straightness NS) of the curve that is directly linked to the unknowns X by the following scalar equations:

$$S = \frac{1}{1+NS} \quad \text{and} \quad NS = \frac{A.C.L}{\ell^2} \in [0, +\infty[$$

where A is the area under the curve, C is the integral of the curvature, L is the curve length and ℓ is the chord length [31]. The user can also interact directly with the curvature or with the position of some particular points as it will be illustrated in the proposed example. The system of equations resulting from the specification is often under-constrained and an objective function $G(X)$ has to be minimized. Here, the objective function is a quadratic function which results from the coupling of a mechanical model directly connected to the control points of the curve to be deformed [52].

The example of Fig. 10 illustrates how the user can deform an initial curve while imposing a position constraint, a curvature constraint and an overall straightness constraint. In this example, the initial straightness $S_i = 0.6556$ is increased by 0.15 to reach the final value of $S_f = 0.8056$ in the deformed configuration. In this case, since there are more unknowns

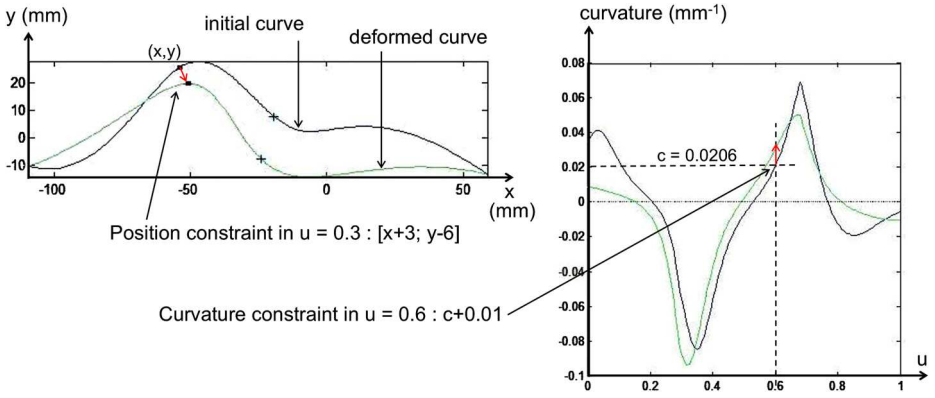

Fig. 10 Aesthetic modification of curve using black box constraints [31]

than equations, the minimization of the variation of the external forces has been used to preserve the shape of the initial curve [52]. Of course, the user does not see the values of the straightness and curvature but can interact with them through a set of high-level parameters to modify the values.

In this work, the solver is decoupled from the modeler as suggested in Section 3.1. Even if the equation of the straightness was available, for simplicity raison, it was decided to consider it as a black box and only the evaluation output by the black box was used. Anyhow, for more complex relationships one could imagine that there is not explicit translation between the aesthetic properties and geometric parameters, and that the constraints result from a procedure which can be considered as a black box.

4.3 Mixing heterogenous models within a deformation process

Multi-representations can be used at different stages of the PDP and a given object can be seen as a combination of components linked with relationships specifying the constraints and spatial transformations. In this context, a new shape description model together with its associated constraints toolbox was proposed in [45] to enable the description of complex shapes from multimodal data. Not only rigid transformations are considered but also scale modifications according to the specified context of the constraint setting. The heterogeneous virtual objects (i.e. composed by scalable multimodal components) then result from the resolution of a constraint satisfaction problem through an optimization approach. Here unknowns X are the positions, the orientations and the scaling factors of each heterogeneous components. So, there are exactly 9 unknowns per component involved in the virtual object definition. The constraints $F(X)$ are taken from a toolbox and can either be expressed by equations or by procedures. The objective function $G(X)$ to be minimized is obtained by combining the energies required to translate (P), rotate (R) and scale (S) the components between their initial and final configurations [45] :

$$G(X) = \sum_i \left(\mu_{pi} \, ||\Delta P_i||^2 + \mu_{ri} \, ||\Delta R_i||^2 + \mu_{si} \, ||\Delta S_i||^2 \right)$$

The example of Fig. 11 shows the result of the combination of two images and two meshes to define a so-called crazy chair. Here, there are 36 unknowns for the 4 components and 18 equations corresponding to 6 position constraints. The scaling factors are set up so that the initial components are not scaled too much : $\mu_{pi} = \mu_{ri} = 500$ and $\mu_{si} = 10^5$.

This approach has been implemented in Unity for the Virtual Environment and makes use of Mathematica for the solving. Thus, as suggested in Section 3.1, the solver is decoupled from the modeler and it can make use of procedures and black box constraints.

4.4 Declarative modeling approach built on top of a CAD modeler

This example explores the idea to construct a plugin above a modeler, taking advantages of its already existing powerful functions and set of procedures [16]. Starting from a mental image, the designer uses a *semantic description* module to express his/her intent through a specific vocabulary and grammar based on the same idea of the example proposed in Section 4.2. The description combines shapes with location attributes and relative quantifiers. The second step of the process is to transform this first description into a set of *generic shape modeling* functions and operators. This second description is closer to the usual

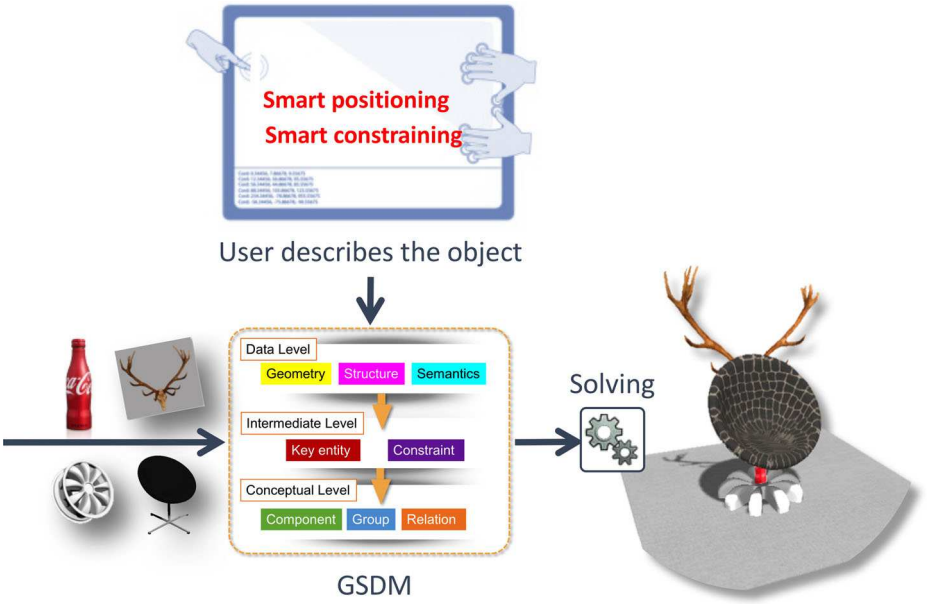

Fig. 11 Heterogeneous model modification using procedures [45]

operators available in CAD modelers but it remains independent of any CAD modelers. The third step consists in interpreting the last description into *specific CAD modeling* functions so that this modeler can finally be launched to generate the desired shape. Here, the proposed approach has been implemented and validated using CATIA as a CAD modeler.

This entire process is illustrated on the example of Fig. 12. Starting from the description below (restricted to its first items) directly specified by the user through a dedicated graphic user interface, the resulting CAD model is obtained together with its building tree. One can notice that this description is based on the creation of simple objects subjected to generic shape modeling operations: localization with [above] and bending with [bend]. Different generic operations were implemented like slicing an object or bumping a face each of them having a set of control parameters which values are directly linked to the adjectives [few], [moderately] and so on.

- [Start with] [cylinder 1] [moderately] [wide], [few] [high]
- [Above] [cylinder 1], [add] [cylinder 2] [moderately] [wide], [extremely-few] [high]
- [Bend] [extremely-few] [cylinder 2]
- [Above] [bending 1], [add] [cylinder 3], [moderately] [wide], [very-few] [high]

The advantage of this approach is the possibility to completely exploit the plugged modeler since the model is created by this modeler. This is particularly interesting to sketch a draft CAD model in the early design phases. This draft CAD model can then be seen as an initial configuration, or sketch, X^0 for the optimization problem to be solved by the solver. More complex requirements must still be added to be able to create complex initial models.

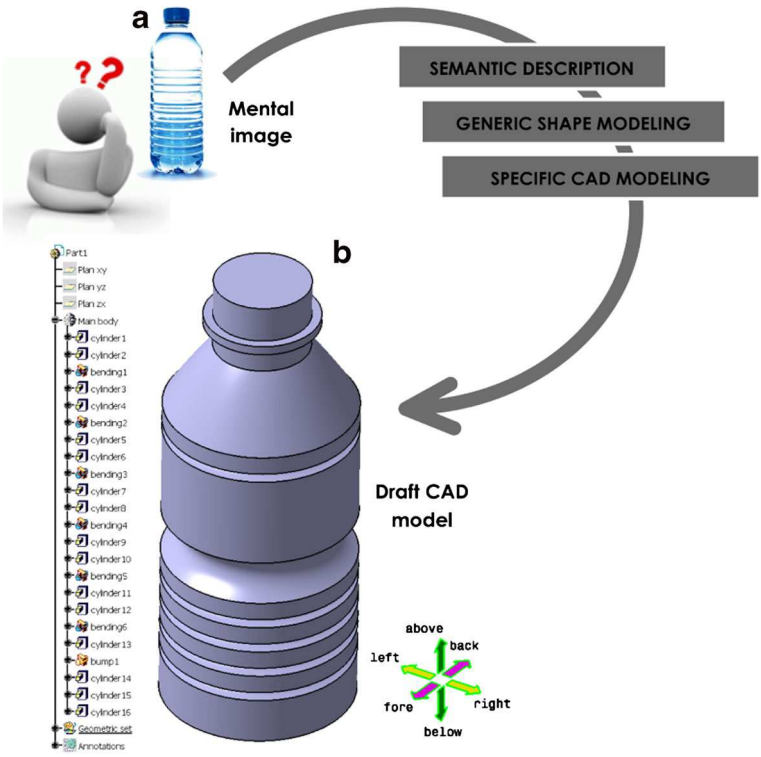

Fig. 12 A bottle of water (a) and the draft model obtained with the declarative modeler (b) [16]

5 Conclusion

This paper considers a new approach of the design process by proposing a framework to interconnect a modeler with a solver, considering that the PDP is a large optimization process. The idea is not to replace existing solvers but rather to better integrate them within an open framework. In some sense, our approach can inherit the advantages of the existing solutions.

Procedures are more general and more convenient than equations, which are not available in some cases. It is always possible to use the best algorithms for numerical solving and qualitative study of systems. It is always possible to compute, with floating point arithmetic precision, the values of the derivatives of functions computed with procedures despite the absence of their mathematical expressions.

Assuming an adequate interface between the modeler and the solver, it is always possible to exploit the sparsity of constraint systems being reducible or not reducible. This ensures the scalability of the approach. At a higher level, this approach allows to combine heterogeneous geometric representations. It doesn't require the redesign of the geometric modeler, and can even use the procedures of different geometric modelers.

Limitations of using this approach include the fact that the solver cannot resort to formal calculus, and interval analysis. It can no more compute the analytic expression of derivatives. However, the procedures called by the solver still can use formal calculus, or interval analysis.

The proposed approach has been illustrated with several promising examples. This is the first step towards the next generation of modelers and solvers for a better interoperability within the PDP.

References

1. Fiores II 2000–03, Character preservation and modelling in aesthetic and engineering design. GROWTH Project GRD-CT-2000-0003. http://www.fiores.com
2. Aim@Shape: Engineering Design Methods: Strategies Design Product. Advanced and Innovative Models and Tools for the Development of Semantic-Based Systems for Handling, Acquiring, and Processing Knowledge Embedded in Multi-Dimensional Digital Objects. European Network of Excellence Key Action: 2.3.1.7. Semantic-based Knowledge Systems, VI Framework (2004)
3. Ait-Aoudia, S., Foufou, S.: A 2d geometric constraint solver using a graph reduction method. Adv. Eng. Softw. **41**, 1187–1194 (2010)
4. Ait-Aoudia, S., Jegou, R., Michelucci, D.: Reduction of constraint systems. arXiv:1405.6131. (Third COMPUGRAPHICS 1993, pp. 331–340) (2014)
5. Bierlaire, M.: Optimization: Principles and Algorithms. EPFL Press, Lausanne (2015)
6. Blender.org: Blender for Open source 3D Creation. https://www.blender.org/
7. Bonnans, J.F., Gilbert, J.C., Lemaréchal, C., Sagastizábal, C.A.: Numerical optimization: theoretical and practical aspects. Springer Science & Business Media (2006)
8. Bouchard, C., Aoussat, A., Duchamp, R.: Role of sketching in conceptual design of car styling. Journal of Design Research **5**(1), 116–148 (2006)
9. Büskens, C., Wassel, D.: The ESA NLP Solver WORHP. In: Fasano, G., Pint'er, J.D. (eds.) Modeling and Optimization in Space Engineering, vol. 73, pp. 85–110. Springer, New York (2013)
10. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**, 1190–1208 (1994)
11. Camba, J.D., Contero, M., Company, P.: Parametric cad modeling: an analysis of strategies for design reusability. Comput. Aided Des. **74**, 18–31 (2016)
12. Cheutet, V., Daniel, M., Hahmann, S., La Gréca, R., Léon, J.C., Maculet, R., Ménégaux, D., Sauvage, B.: Constraint modeling for curves and surfaces in CAGD: a survey. Int. J. Shape Model. **13**(2), 159–199 (2007)
13. Contero, M., Company, P., Vila, C., Aleixos, N.: Product data quality and collaborative engineering. IEEE Comput. Graph. Appl. **22**, 32–42 (2002)
14. Cross, N. Engineering Design Methods: Strategies for Design Product, 3rd edn. Wiley, Chichester (2000)
15. Danglade, F., Pernot, J.P., Véron, P.: On the use of machine learning to defeature CAD models for simulation. Comput.-Aided Des. Applic. **11**(3), 358–368 (2014)
16. Decriteau, D., Pernot, J.P., Daniel, M.: Towards declarative CAD modeler built on top of a CAD modeler. In: Proceedings of CAD'15, Computer Aided Design and Applications, pp. 107–112 (2015)
17. Dufourd, J.F., Mathis, P., Schreck, P.: Formal resolution of geometrical constraint systems by assembling. In: Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications, pp. 271–284. ACM (1997)
18. Dufourd, J.F., Mathis, P., Schreck, P.: Geometric construction by assembling solved subfigures. Artif. Intell. **99**(1), 73–119 (1998)
19. Elber, G., Kim, M.S.: Geometric constraint solver using multivariate rational spline functions. In: Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications, pp. 1-10. ACM, New York (2001)
20. Essert-Villard, C., Schreck, P., Dufourd, J.F.: Sketch-based pruning of a solution space within a formal geometric constraint solver. Artif. Intell. **124**(1), 139–159 (2000)
21. Eva Catalano, C., Falcidieno, B., Giannini, F., Monti, M.: A survey of computer-aided modeling tools for aesthetic design. J. Comput. Inf. Sci. Eng. **2**, 11–20 (2002)
22. Falcidieno, B., Giannini, F., Léon, J.-C., Pernot, J.-P.: Processing free form objects within a Product Development Process framework. In: Michopoulos, J.G., Paredis, C.J.J., Rosen, D.W., Vance, J.M. (eds.) Advances in Computers and Information in Engineering Research, vol. 1, pp. 317–344. ASME-Press (2014)
23. Fischer, I.: Dual-Number Methods in Kinematics, Statics and Dynamics. Routledge, Evanston (2017)
24. Foufou, S., Michelucci, D.: Bernstein basis and its application in solving geometric constraint systems. Journal of Reliable Computing **17**, 192–208 (2012)

25. Foufou, S., Michelucci, D.: Interrogating witnesses for geometric constraint solving. Inf. Comput. **216**, 24–38 (2012). Special Issue: 8th Conference on Real Numbers and Computers
26. FreeCAD: FreeCAD: An open-source parametric 3D CAD modeler. https://www.freecadweb.org/
27. FreeSHIP: FreeSHIP: Surface Modeling. https://sourceforge.net/projects/freeship/
28. Fünfzig, C., Michelucci, D., Foufou, S.: Nonlinear systems solver in floating-point arithmetic using lp reduction. In: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, SPM'09, pp. 123–134. ACM, New York (2009)
29. Ge, J.X., Chou, S.C., Gao, X.S.: Geometric constraint satisfaction using optimization methods. Comput.-Aided Des. **31**(14), 867–879 (1999)
30. GeoGebra.org: GeoGebra: Dynamic Mathematics for Learning and Teaching. https://www.geogebra.org/
31. Giannini, F., Montani, E., Monti, M., Pernot, J.P.: Semantic evaluation and deformation of curves based on aesthetic criteria. Comput.-Aided Des. Applic. **8**(3), 449–464 (2011)
32. Gouaty, G., Fang, L., Michelucci, D., Daniel, M., Pernot, J.P., Raffin, R., Lanquetin, S., Neveu, M.: Variational geometric modeling with black box constraints and DAGs. Comput. Aided Des. **75**, 1–12 (2016)
33. Hendrickson, B.: Conditions for unique graph realizations. SIAM J. Comput. **21**(1), 65–84 (1992)
34. Hoffmann, C.M.: Geometric and Solid Modeling: an Introduction. Morgan Kaufman, San Mateo (1989)
35. Hoffmann, C.M., Joan-Arinyo, R.: Symbolic constraints in constructive geometric constraint solving. J. Symb. Comput. **23**(2-3), 287–299 (1997)
36. Hoffmann, C.M., Lomonosov, A., Sitharam, M.: Geometric constraint decomposition. In: Geometric Constraint Solving and Applications, pp. 170–195. Springer (1998)
37. Hu, H., Kleiner, M., Pernot, J.P.: Over-constraints detection and resolution in geometric equation systems. Comput. Aided Des. **90**, 84–94 (2017)
38. Jaulin, L., Kieffer, M., Didrit, O., Walter, É.: Interval Analysis. Springer, London (2001)
39. Jermann, C., Trombettoni, G., Neveu, B., Mathis, P.: Decomposition of geometric constraint systems: a survey. Int. J. Comput. Geom. Appl. **16**(05n06), 379–414 (2006)
40. Joan-Arinyo, R., Soto-Riera, A.: Combining constructive and equational geometric constraint-solving techniques. ACM Trans. Graph. (TOG) **18**(1), 35–55 (1999)
41. Khorramizadeh, M.: An application of the Dulmage-Mendelsohn decomposition to sparse null space bases of full row rank matrices. In: International Mathematical Forum, vol. 52, pp. 2549–2554 (2012)
42. Kondo, K.: Algebraic method for manipulation of dimensional relationships in geometric models. Comput. Aided Des. **24**(3), 141–147 (1992)
43. Kubicki, A., Michelucci, D., Foufou, S.: Witness computation for solving geometric constraint systems. In: Science and Information Conference (SAI), 2014, pp. 759–770. IEEE (2014)
44. Lesage, D., Léon, J.C., Sebah, P., Rivière, A.: A proposal of structure for a variational modeler based on functional specifications. In: Gogu, G., Coutellier, D., Chedmail, P., Ray, P. (eds.) Recent Advances in Integrated Design and Manufacturing in Mechanical Engineering, pp. 73–84. Springer (2003)
45. Li, Z., Giannini, F., Pernot, J.P., Véron, P., Falcidieno, B.: Re-using heterogeneous data for the conceptual design of shapes in virtual environments. Virtual Reality **21**(3), 127–144. Springer (2017)
46. Linke, T., Wassel, D., Büskens, C.: Recent advances in the solution of large nonlinear optimisation. In: Rodrigues, H., Herskovits, J., Soares, C.M., Guedes, J.M. (eds.) Engineering Optimization IV, pp. 141–146. Taylor & Francis, New York (2014)
47. Lovász, L., Plummer, M.D.: Matching Theory. American Mathematical Society Providence, USA. ISBN13 9780821847596 (2009)
48. Michelucci, D., Foufou, S.: Interrogating witnesses for geometric constraint solving. In: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, pp. 343–348. ACM (2009)
49. Moinet, M., Mandil, G., Serre, P.: Defining tools to address over-constrained geometric problems in computer aided design. Comput. Aided Des. **48**, 42–52 (2014)
50. Mourrain, B., Pavone, J.P.: Subdivision methods for solving polynomial equations. J. Symb. Comput. **44**(3), 292–306 (2005)
51. Owen, J.C.: Algebraic solution for geometry from dimensional constraints. In: Proceedings of the First ACM Symposium on Solid modeling Foundations and CAD/CAM Applications, pp. 397–407. ACM (1991)
52. Pernot, J.P., Falcidieno, B., Giannini, F., Léon, J.C.: A hybrid models deformation tool for free-form shapes manipulation. In: 34Th Design Automation Conference (ASME DETC08-DAC 49524), pp. 647-657. ASME, New-York (2008)
53. Pernot, J.P., Falcidieno, B., Giannini, F., Léon, J.C.: Incorporating free-form features in aesthetic and engineering product design: state-of-the-art report. Comput. Ind. **59**(6), 626–637 (2008)
54. Rameau, J.F., Serré, P.: Computing mobility condition using Groebner basis. Mech. Mach. Theory **91**, 21–38 (2015)

55. Rao, R.: Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int. J. Ind. Eng. Comput. **7**(1), 19–34 (2016)
56. Saad, Y.: Iterative methods for sparse linear systems. SIAM, Philadelphia (2003)
57. Schreck, P., Mathis, P.: Geometrical constraint system decomposition: a multi-group approach. Int. J. Comput. Geom. Appl. **16**(05n06), 431–442 (2006)
58. Schreck, P., Schramm, É.: Using invariance under the similarity group to solve geometric constraint systems. Comput. Aided Des. **38**(5), 475–484 (2006)
59. Serrano, D.: Automatic dimensioning in design for manufacturing. In: Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, SMA '91, pp. 379–386. ACM, New York (1991)
60. Solomon, C., Gibson, S.J., Maylin, M.I.S.: A new computational methodology for the construction of forensic, facial composites. In: Proceedings of the 3rd IWCF, Netherlands, 2009. Lecture Notes in Computer Science, vol. 5718, pp. 67–77. Springer (2009)
61. Sommese, A.J., Wampler, C.W. II.: The Numerical Solution of Systems of Polynomials Arising in Engineering and Science. World Scientific, Singapore (2005)
62. Stiteler, M.: Construction History and Parametrics: Improving Affordability through Intelligent CAD Data Exchange. Chaps program final report, Advance Technology Institute (2004)
63. Thierry, S.E., Schreck, P., Michelucci, D., Fünfzig, C., Génevaux, J.D.: Extensions of the witness method to characterize under-, over-and well-constrained geometric constraint systems. Comput. Aided Des. **43**(10), 1234–1249 (2011)
64. Tovey, M.: Intuitive and objective processes in automotive design. Des. Stud. **13**(1), 23–41 (1992)