Computer-Aided Design 75–76 (2016) 1–12

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Variational geometric modeling with black box constraints and DAGs

Gilles Gouaty [a,b,c], Lincong Fang [c,d], Dominique Michelucci [c,*], Marc Daniel [b], Jean-Philippe Pernot [a], Romain Raffin [b], Sandrine Lanquetin [c], Marc Neveu [c]

[a] Arts et Métiers ParisTech, LSIS Laboratory UMR CNRS 7296, France
[b] Aix-Marseille University, LSIS Laboratory UMR CNRS 7296, France
[c] LE2I UMR6306, CNRS, Arts et Métiers, Bourgogne Franche-Comté University, Dijon, France
[d] School of Information Technology, Zhejiang University of Finance & Economics, Hangzhou, China

ARTICLE INFO

Article history:
Received 19 July 2015
Accepted 5 February 2016

Keywords:
Black box constraints
Variational geometric modeling
Direct search methods
First order methods
Constraints
DAG

ABSTRACT

CAD modelers enable designers to construct complex 3D shapes with high-level B-Rep operators. This avoids the burden of low level geometric manipulations. However a gap still exists between the shape that the designers have in mind and the way they have to decompose it into a sequence of modeling steps. To bridge this gap, Variational Modeling enables designers to specify constraints the shape must respect. The constraints are converted into an explicit system of mathematical equations (potentially with some inequalities) which the modeler numerically solves. However, most of available programs are 2D sketchers, basically because in higher dimension some constraints may have complex mathematical expressions. This paper introduces a new approach to sketch constrained 3D shapes. The main idea is to replace explicit systems of mathematical equations with (mainly) Computer Graphics routines considered as Black Box Constraints. The obvious difficulty is that the arguments of all routines must have known numerical values. The paper shows how to solve this issue, *i.e.,* how to solve and optimize without equations. The feasibility and promises of this approach are illustrated with the developed DECO (Deformation by Constraints) prototype.

1. Introduction

Industrial CAD software rely on an incremental B-Rep (Boundary Representation) modeling paradigm where volume modeling is performed iteratively through high-level operators [1]. At a lower description level, those modeling operators are based on Euler operators acting directly on the faces, edges and vertices of B-Rep models. In this way, designers do not manipulate low-level geometric entities, but rather manipulate so-called structural and detail features to shape directly the CAD models.

However, even if CAD modelers provide operators (*e.g.,* pad, pocket, shaft, groove, hole, fillet) to get rid of the direct use and manipulation of canonical surfaces and NURBS [2], working with a CAD modeler is almost procedural and requires a lot of intermediate operations to obtain the desired shape of an object. Using such a procedural approach, designers have to make a mental gymnastic to break down the object body into several basic shapes linked to the different operators of the CAD software. Thus, even if a feature-based approach is used [3], modeling a complex shape still requires a lot of operations. This is even truer when dealing with free form objects for which the notion of free form features does not correspond to current industrial practices.

Clearly, an approach closer to the designers' way of thinking is missing and there is still a gap between the shapes designers have in mind and the tools and operators provided to model them. Various approaches have been introduced to bridge this gap: parametric modeling, feature-based modeling and variational modeling approaches.

In parametric modeling, or parametric and feature-based modeling [4], a shape is defined as a function $F(U)$, where F is some function, and U are its parameters. Designers specify with some interactive graphical interface the function F as well as the values of U parameters. Parameters U are geometric variables (e.g. lengths, angles, tolerances, Cartesian coordinates) or material properties (e.g. density, strength, cost). When the values of some parameters in U are modified during the design process, the shape is automatically updated while re-computing $F(U)$. Most of the time, only a part of $F(U)$ is re-computed, using a dependence analysis.

* Corresponding author.
E-mail address: dominique.michelucci@u-bourgogne.fr (D. Michelucci).

http://dx.doi.org/10.1016/j.cad.2016.02.002
0010-4485/

Variational geometric modeling [4] goes a step further. Designers specify constraints the shape $F(U)$ must satisfy as well as the unknown parameters. Then, a numerical solver tries to satisfy the constraints while computing values of unknown parameters of U. When defining a 3D shape, the constraints are often geometric constraints, which relate to different geometric primitives or features. For example, they can be distances or angles between (special points or axes of) geometric primitives or features, incidence or tangency relations between parts of two geometric primitives or features. In this case, their formal expression is simple and can be easily computed. It leads to a system of equations, most of the time algebraic. Numerous combinatorial or numerical methods [5] were proposed first to detect the under-, over-, and well-constrained parts of this kind of systems, and second, for well-constrained systems (which have as many independent equations as unknowns, and have a finite number of solutions) to decompose them into irreducible subsystems and to assemble their solutions.

In practice, numerical methods like Newton iterations, damped Newton or homotopy are used to solve irreducible subsystems and to assemble the partial solutions. The numerical solver typically starts from the previous values of U, read on some interactively provided sketch, or on the previous state of an iteratively edited shape.

Sometimes, some objective functions $G(U)$ must be optimized: for instance a cost, a weight or an energy should be minimized.

- On one hand, if the constraints system is well-constrained, there is a finite set of solutions and the best one, or a good enough one for the sake of computability, must be selected. This discrete problem is combinatorial and can be hard to solve (*e.g.,* Travelling Salesman Problem).
- On the other hand, if the system is under-constrained, it admits a continuum of feasible solutions and, under the usual mild assumptions, a finite number of parameters values U which satisfy both the constraints and some KKT (Karush–Kuhn–Tucker) or FJ (Fritz-John) conditions for local optimality.

The key feature of current variational geometric modeling approaches is that equations are available, and can be represented with tree-like data structures called DAGs (Directed Acyclic Graphs) in numerical analysis, computational geometry and computer algebra, or SLPs (Straight Line Programs) in dynamic geometry. The main advantage of DAGs is to permit to automatically compute the derivatives and Hessians. It is also possible to substitute parameters at the leaves of a DAG with other DAGs, to convert a given DAG into the corresponding polynomial (a list of monomials) or rational function. This is used to numerically evaluate a given DAG with many arithmetics (floating-point, intervals, exact arithmetics) for given numerical values of U. Like this it is possible to study which nodes in a given DAG depend on which parameters, and thus to update efficiently the value of a DAG when some parameters values in U are changed. To summarize this feature, we say that these DAGs are white box DAGs, or white DAGs.

On the contrary, a DAG is called a black box DAG or a black DAG, and a constraint is called a black box constraint or a black constraint, when the corresponding equation, or system of equations, is not available, or is not computable in practice. In this case, it is only possible to evaluate the corresponding DAG for given numerical values of parameters U, and to approximate the gradient with finite differences. There is no guarantee that the underlying function is continuous or smooth everywhere.

To illustrate differences between white and black DAGs, imagine we need a point x inside a given shape s and closest to a given point p. Then x is the solution of the constrained optimization problem $x = \text{argmin}_{x \in s} \|x - p\|^2$, where, with classical Variational Modeling, the condition $x \in s$ must be expressed as a system of mathematical equations. Clearly, if s is a car or a building, it is just infeasible. Actually it is also infeasible for simple shapes, as soon as they involve nested geometric operations (*e.g.,* rounding, blending, Boolean operations, optimizations). On the other hand, computer graphics methods routinely solve this problem $x = \text{argmin}_{x \in s} \|x - p\|^2$ using the routine closestPt(p, s) which does not rely on systems of equations.

In this paper, we propose to use black box DAGs instead of white box DAGs for Variational Geometric Modeling of free form surfaces and subdivision surfaces. We present a prototype, called DECO (Deformation by Constraints), to show the feasibility and promises of this approach. Our research is devoted to free form parametric surfaces as well as to subdivision surfaces due to the gap which currently exists between variational design and free form surface modeling and because subdivision surfaces are largely used in Computer Graphics and animation movies. Moreover subdivision surfaces do not have implicit or parametric equations and are generally manipulated as meshes approximating the limit surfaces. In addition to the specificity of modeling these types of surfaces, the interest of this novel approach is twofold. First, we no longer have to translate, when it is possible, the geometric constraints and the cost function into equations. Second, to express geometric constraints F and cost function G, we can use existing geometric procedures available in Computational Geometry, CAD/CAM and mathematical or numerical software. Assuming interoperability, functions or macros available in a geometric modeler software could be called. Thus this approach permits to easily extend the set of possible constraints. Certainly, with black DAGs, we can no longer use tools of Computer Algebra (for symbolic and exact computations of Jacobians, Hessians, resultants, Gröbner bases) since no equation is available. But we think that the advantages of our approach far outweigh its disadvantages. Additionally, it must be noticed that we aim at obtaining easily a first draft respecting given constraints in a preliminary stage of a design process and not necessarily final objects. The received models can then be exported to any CAD software for further developments.

The proposed approach is modular. It defines a formalism and framework regardless of the resolution method. For example, we use the GNU Scientific Library with the BFGS method. Furthermore, our approach is generic in the sense that one can consider later to treat other types of surfaces. Actually, we simply need to identify variables and define black boxes to make calculations on these surfaces.

The paper is organized as follows. Section 2 studies the related works comparing white and black DAGs. Our new modeler, allowing the specification of a set of constraints as well as an objective function to be minimized is introduced in Section 3. The associated solvers are presented in Section 4. Section 5 is devoted to examples. Finally, Section 6 concludes this paper and exhibits general issues raised by this approach.

2. Related work

Today's industrial CAD modelers are built on top of the well-known B-splines and NURBS paradigms to model free form surfaces [6,2]. Since the expected shapes are generally complex, the designer often has to decompose them into elementary shapes themselves subdivided into several surfaces. Each elementary surface is defined by means of a network of control points, weights and knot sequences. Most of the time, these surfaces must be trimmed to overcome the topological constraints of the mathematical models. Finally, the elementary surfaces are assembled together to produce a manifold solid, *i.e.,* a B-Rep representation expressing the relationships between the vertices, the edges and the faces of the topological model. Nevertheless, interacting at this low level is restricted to experts. Several attempts have been made to try to overcome the limits inherent to

the manipulation of low-level geometric entities and shapes and are discussed below.

Feature-based modeling introduced in [3] and detailed in [7] falls into this category of higher-level approaches. By using features to build their CAD models, designers do not anymore act at a low level but rather on shape primitives that can be parametrized and pre-defined. Unfortunately, the features used to design a part do not necessarily represent the best way to manufacture it. Therefore, it is the designer's responsibility to evaluate all methods that can produce the target shape.

Subdivision surfaces have become popular in Computer Graphics [8,9]. Here, the surface is defined by recursively subdividing to the infinity an initial coarse mesh. This property is useful in a first sketching step, as the user manipulates vertices of a rough mesh which can visually result in a similar but smooth surface. The definition of shapes over a subdivision surface goes through the same procedure as for B-splines and NURBS, *i.e.,* modification of the initial coarse mesh using constraints specified by the users in a more or less intuitive way.

Variational Modeling has been introduced in CAD [10]. It has been used for the design of light reflectors in inverse rendering problems [11], for the design of blend surfaces [12], for the resolution of complex 2D geometric constraints [13] (already with BFGS, and with Levenberg–Marquardt's method), and for the modification of NURBS surface with geometric constraints [14]. However, in all these works, equations were available. The research in variational modeling quickly concentrates on the important problem of constraints modeling, how to represent and organize them with DAGs, and finally how to solve them [15].

For both NURBS and subdivision surfaces, it is not always possible to define the constraints and operators in an explicit manner, *i.e.,* with equations. Thus, the use of black DAGs becomes crucial.

The remaining of this section presents properties of DAGs, how they are used in dynamic and in computational geometry and try to analyze the respective advantages of white and black DAGs.

2.1. DAGs for dynamic geometry

There is a strong analogy of our approach with dynamic geometry. Parametric Modeling is used in dynamic geometry software like Cabri Geometry [16] (Cabri Géomètre is likely the first software of this kind), Cinderella [17,18], or GeoGebra [19,20]. In dynamic geometry, U is a set of 2D points, and the object $F(U)$ is a geometric figure illustrating some geometric theorems, like the alignment of three points in Pappos or Pascal's theorems. Users are typically students. When they drag a point (a parameter) of the figure (possibly constrained to lie on a curve like a line, a circle or a conic), the figure is updated and refreshed, and students can see that the geometric property of a theorem still holds.

In dynamic geometry, the leaves of the DAG (also called SLP, Straight Line Program) are 2D points, called base points. Each base point p_i is represented with two symbols (x_i, y_i) but also with two numeric coordinates the user interactively provides with the sketch. Actually each geometric entity has both a symbolic description, stored in the DAG, and a numerical definition. Each non terminal node of the DAG records the name of a geometric function and pointers to the sons of the node, which define arguments of the function. When the coordinates of some base points are interactively modified by a user, this function is called to update the coordinates of the related geometric entities. For instance, a node Line(p_1, p_2) represents the line passing through two points p_1 and p_2. This node has two sons, the first for p_1 and the second for p_2. A node Point(o_1, o_2) is used for the intersection point of two geometric curves o_1 and o_2, which are lines, circles or conics recursively defined in the same way. In this last example, one can notice that the two curves o_1 and o_2 may intersect at more than one point. For example, a line and a circle, or two circles, generically intersect at 2 points, so an ambiguity arises. When updating the intersection point $o_1 \cap o_2$ between times t_i and t_{i+1}, the ambiguity is typically solved with a continuity argument, *i.e.,* selecting at time t_{i+1} the solution which is the closest to the intersection point at time t_i. A well-known problem in dynamic geometry occurs when two curves no more intersect [21–23]. This problem also occurs in our approach and is related to the Persistent Naming Problem discussed in Section 6.

We now mention the main differences between dynamic geometry and our approach.

In dynamic geometry, the DAGs or SLPs of a figure $F(U)$ are easily converted into a set of mathematical formulas, typically a polynomial triangularized system of equations:

$$\begin{cases} v_1 = f_1(U) \\ v_2 = f_2(U, v_1) \\ \dots \\ v_n = f_n(U, v_1, \dots, v_{n-1}) \end{cases} \qquad \begin{cases} f_1(U, v_1) = 0 \\ f_2(U, v_1, v_2) = 0 \\ \dots \\ f_n(U, v_1, \dots, v_n) = 0 \end{cases}$$

either (on the left) in an explicit form possibly involving some square roots $\pm\sqrt{.}$, or (on the right) in an implicit form when some degree 3 or 4 equations (intersection between conics) are involved. The user cannot specify constraints: the latter would give non triangularized system of equations. For example, when users want to create a circle tangent to three lines or circles, they have to provide the geometric construction by ruler and compass. Sometimes, users can define some macro construction function, and add some button to the graphical interface, but dynamic geometry and more generally parametric modeling do not provide constraints, stricto sensu. This is for pedagogical reasons: users are students learning geometry.

Thus, there is no DAG to specify constraints, or an objective function. Consequently, dynamic geometry does not provide solvers or optimizers and is therefore not adapted to our needs.

Finally, dynamic geometry uses white DAGs unlike our DECO prototype software based on black DAGs.

2.2. DAGs in computational geometry

DAGs are used in Computational Geometry (CG) programs. It is well known that CG methods do not withstand inaccuracy of the floating-point arithmetic as well as the inconsistencies it introduces. So, CG software like CGal [24,25] uses exact decisions for geometric predicates, like the orientation (a, b, c) predicate for three points a, b and c (are a, b, c aligned, or do they turn left, or do they turn right?), or the inCircle(a, b, c, d) predicate (does the point d lie inside, on, or outside the circle passing through the three points a, b, c?). The test is first performed in floating point arithmetic with some error bound (computed at compile time or at run time, and depending on the input range), or with some interval arithmetic. If the accuracy is not sufficient, the test is performed again using some exact arithmetic, typically in $\mathbb{Z}$ or in $\mathbb{Q}$.

2.3. Features of white and black box DAGs

The main feature of DAGs, white or black, is that they can be interactively defined by users. This is done in dynamic geometry. Actually, users of these software do not only define the DAG which represents a function F building a geometric figure $F(U)$, they also provide values for the arguments U of the function F. This is also done in Parametric Modeling, sometimes called history-based modeling. Here, the history of an interactive session is stored in some DAG, and the session can be replayed when the value of some parameters is modified. This is also done in our prototype.

We list now the properties of white box DAGs, used in dynamic geometry, and computational geometry. We also indicate whether black box DAGs share these properties.

1. DAGs can be interactively defined. DAGs are programs, which can be created by users who are not computer scientists. It also holds for black box DAGs.
2. It is possible to automatically compute the DAG of derivatives and Hessians from the given white DAG of a function. This is not possible for black box DAGs. We usually approximate the derivatives with first-order one-sided finite differences. More accurate results could probably be obtained by more complex formulae, but they require more computing time and the current accuracy is sufficient for our needs, as our goal is to get sketches satisfying approximative criteria.
3. It is possible to substitute parameters, at the leaves of a DAG, with other DAGs. It also holds for black box DAGs.
4. It is easy to convert a given white DAG into explicit mathematical formulas, say polynomials or rational functions in the simplest case. This conversion is not practicable for black DAGs, even though it is possible in theory [26]. Black DAGs are programs, programs are Turing machines, and Turing machines are representable with polynomials. But the only use of this equivalence is for Matiyasevich's proof of the impossibility of solving Diophantine equations (Hilbert's tenth problem).
5. It is possible to numerically evaluate a given DAG with many arithmetics (floating-point, intervals, exact arithmetics in $\mathbb{Z}$, $\mathbb{Q}$, etc.) given numerical values for U. For black box DAGs, only floating-point evaluation is possible, thus interval solvers cannot be used to solve $\min G(U)$ with constraints $F(U) = 0$.
6. It is possible to study which nodes in a given DAG depend on which parameters, and thus it is possible to update efficiently the value of a DAG when some parameters values in U are changed. This holds both for white and black box DAGs (assuming the signature of functions underlying DAGs is known).
7. It is possible to compile DAGs, *i.e.,* to generate automatically an equivalent C or C++ program, to compile it and to dynamically link it with the current process. The latter holds both for white and black box DAGs.

Previous item (4) may be misunderstood as a limitation or a drawback of black DAGs. On the contrary, black nodes can harness the power of very efficient Computer Graphics or Computer Geometry routines, *e.g.,* ClosestPt(p, s) which computes the point on the shape s which is the closest to a given point p. These routines numerically solve complex (possibly nested) constrained optimization problems, without need of their mathematical expressions, which are not available, or would be of exponential size.

To conclude this section, our aim in this paper is to create a modeler starting from a description and transformed in a set of geometric constraints. Note that our approach is to generate drafts of shapes from more or less approximate descriptions. It is not necessary for us to seek very accurate solutions of the constraints system. Therefore, numerical accuracy problems inherent in our black DAGs approach are unimportant for us. Numerical methods generate a sequence that tends towards the exact solution. Thus, we can stop these iterations before reaching a more accurate but time consuming result.

As a conclusion, the complexity of the set of constraints we will encounter leads us to base our modeler on a black DAG approach described in the following. Some white DAGs are also included when equations are available.

3. DECO, a modeler based on black DAGs

The developed modeler allows the user to define an object by means of a description. This description consists in declaring a set of objects and specifying a set of properties. These properties may be related to particular objects or to a set of constraints linking various objects. The descriptions given by the user are represented in a data structure embedding all necessary information for the modeler. This section introduces the data structure used to represent the descriptions. It uses intensively black DAGs.

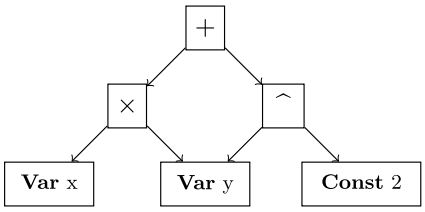

Fig. 1. Example of a white DAG composed of scalar operators, variables and constants, representing the expression $xy + y^2$.

At the global level, the data structure consists in the following information:

- a list of objects (parametric surfaces or subdivision surfaces) defined by the subset X of unknown parameters in U;
- a list of constraints $F_i(X) = 0$, where F_i functions are not necessarily expressed by equations;
- a function to minimize $G(X)$.

In the proposed approach, DAGs are used to represent both F_i and G functions. Each node of a DAG represents an operator and the list of the sons represents the list of arguments of this operator. A first set of nodes has been defined and is presented below. These operators constitute a basic vocabulary of geometric computations. They can be dynamically composed to represent and solve a large variety of problems.

A node has a certain type corresponding to the type of the value returned by the operator. This type can be either a scalar value, a point, a parametric surface (denoted pSurf) or a subdivision surface (denoted sSurf). In the proposed approach, we use Catmull–Clark subdivision surfaces evaluated at a fixed level. We do not use an exact evaluation of points on the limit surface [27] as it is time-consuming and complex to obtain the local frame according to each point. Moreover, one important goal of the DECO project is to be able to consider all together constraints on meshes and parametric surfaces so as to define a draft model to be refined later on.

Nodes with no argument constitute the leaves of the DAGs. These nodes can be either constant values or variables. They both return a scalar value. These variables are the unknowns of the system.

We define a set of scalar nodes for the basic operations $+, -, \times, \div, \widehat{}$ and for the usual functions. Unlike other nodes, they can be considered as white box operators.

An example of DAG representing the scalar expression $xy + y^2$ is shown in Fig. 1.

Other scalar operators can take non-scalar data as argument. It is the case for the operators called GetX, GetY and GetZ. They take one argument of type point and return the corresponding coordinate. Reciprocally, we define a node Point taking three scalar arguments and returning the point with the given coordinates.

The nodes PSurf and SSurf take a list of control points as arguments and return a surface of type pSurf or sSurf. Other parameters, such as the degrees and the knot vectors for parametric surfaces and the mesh topology for subdivision surfaces, are not stored in DAGs. In the current implementation, we consider these parameters as internal to the nodes, even if the proposed approach could be extended while considering them as unknowns in a future version of DECO. They are chosen at the creation time of every nodes and are not changed anymore. Reciprocally, the node GetCtrlPt extracts a control point from a surface at given indices. These indices are given by a scalar parameter but we assume their values are integer.

The node CalcPt computes a point on a parametric surface for two scalar coordinates (u, v) in the parametric domain $[0, 1]^2$. In

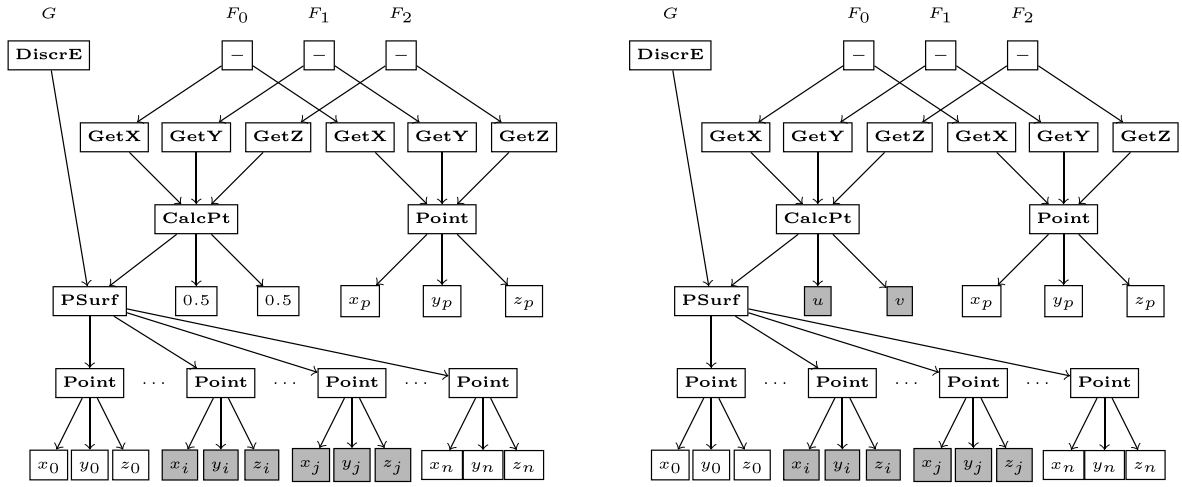

Fig. 2. Example 1: DAGs representing the optimization problems to be solved, when (u, v) is constant (left) and variable (right).

the same way, the node CalcNorm computes the normal at this point.

The node ClosestPt computes from a given point the closest point on a surface or a list of surfaces. These surfaces can be indifferently parametric surfaces or subdivision surfaces. In the same way, the node ClosestNorm computes the unit normal vector at this point.

Finally, the node DiscrE computes the discrete strain energy of a surface. It takes one argument of type pSurf or sSurf and returns a scalar value. This operator is mainly used to define the function to be minimized. The system of equations is generally under-constrained. So the minimization of the discrete energy allows to choose the smoothest solution, *i.e.,* the one which is obtained with less energy. The discrete strain energy is quickly computed and is a good approximation of the strain energy [28] that fits our needs. It is computed from the discrete Gaussian curvature K_v, and the discrete mean curvature H_v, for each vertex v. The discrete Gaussian curvature K_v of a vertex v can be given by:

$$K_v = \frac{2\pi - \sum_i \alpha_i}{\frac{1}{3}A_v}$$

where α_i is the angle between two consecutive edges incident at v and A_v is the area of the star (*i.e.,* the area of the polygons around the vertex v). The discrete mean curvature H_v of a vertex v can be given by:

$$H_v = \frac{1}{4}\sum_i \ell_i\,(\pi - \beta_i)$$

where β_i is the internal dihedral angle between two consecutive faces around the vertex v and ℓ_i the length of their common edge. By convention, as the mean curvature is non zero along the edge, it is distributed equally on its two vertices. Finally, the discrete energy of the whole mesh is the sum of the contributions of all vertices v:

$$E = \sum_v 4H_v^2 - 2K_v.$$

Providing efficient second order estimators for discrete curvatures is often studied. The reader can obtain valuable information in [29] and [30]. However, the above chosen formulas were sufficient for the objective of this paper.

Fig. 2 presents two graphs illustrating the use of these different nodes.

In practice, the user creates dynamically these DAGs by providing a description in entry of the software. In a first version, this description is a text written in Python language. Thus, a library of Python functions has been defined to automatically generate the different DAGs. These functions cover all the levels of descriptions and generate more or less complex DAGs. For example, two points or vectors can be constrained to be equal using a single function that generates three equations for each coordinate x, y and z. At a higher level, one can for example establish G^0 or G^1 continuity with a function that creates automatically a list of equalities between points and vectors distributed along two edges of two connected surfaces. The DAGs are then converted into C++ wherein the DECO prototype is implemented and can be evaluated by a numerical solver.

4. The DECO solvers

4.1. Solving without equations

The objects being described, we have to solve the problem: $\min G(X)$ with $F(X) = 0$. Here, X stands for the subset of parameters U which are unknown. Solvers and optimizers must be compatible with the fact that the functions G and F are represented with black DAGs. Therefore, it is only possible to evaluate F and G for given floating-point values of X. For example, interval solvers are not compatible.

The main idea is to reduce this problem to an unconstrained optimization problem as explained in Section 4.2. To solve this optimization problem, we can mention first order methods and direct search methods.

First order methods need gradient vectors. We approximate the gradient vectors with finite differences (this is always numerically a great challenge). We use first order Taylor expansions due to computing time limitation even if more accurate solutions can be developed. Well-known first order methods are for instance the steepest gradient descent, BFGS (Broyden–Fletcher–Goldfarb–Shanno) method, and Nesterov's method [31]. Several libraries provide BFGS: GSL (GNU Scientific Library), Octave or MATLAB.

Apart the BFGS, other Newton or quasi-Newton methods are not convenient due to the size of the Hessian. However we did not try to exploit the sparsity of the Hessian and we did not even study its sparsity. For this first investigation, we also did not experiment JFNK (Jacobian-free Newton–Krylov) methods [32].

Direct search methods [33], also known as pattern search methods, do not use gradient vectors. In this class, we tried Nelder–Mead, Torczon and the Hooke–Jeeves algorithms. They are very close to each other. In comparison with first order methods,

direct methods require a much larger number of iterations. However, each iteration is much quicker because it requires only the evaluation of the function at a given point. The performance of direct and first order methods is generally quite similar, both for running time and accuracy.

We assume that $G(X)$ is bounded below, for the problem to make sense. In addition, $G(X)$ has no reason to be convex, and likely can have many local minima, which these solvers can be trapped in. There are two answers:

- If we have no idea for values of X, we may use meta-heuristics: simulated annealing, tabu search, swarm optimization, genetic algorithms, evolutionary algorithms, etc.
- Otherwise, we expect the solver to converge to a solution close to the initial value $X^{(0)}$. We assumed that we are in such a situation. We could use meta-heuristics, but probably they would give solutions too far from $X^{(0)}$.

We implemented different solvers which can be launched by the user to solve his/her problem. We expect that the first order solver or the direct search solver likely converges to the solution intuitively closest to the initial solution. To increase the probability to find the closest solution, we propose an homotopy inspired method presented in Section 4.2. By default, BFGS solver is used but the user can change the solver and/or the parameters if he/she expects a better accuracy.

4.2. Reductions

Let $\mathcal{P}$ be the problem to solve: $\min G(X)$ with $F(X) = 0$. If there is no objective function $G(X)$ to minimize, but only constraints $F(X) = 0$, we minimize $H(X) = \sum_i (F_i(X))^2$. If there is an objective function $G(X)$ to minimize, and no constraint, then the optimization problem is unconstrained, and we just minimize $G(X)$.

For constrained minimization, the problem is reduced to an unconstrained minimization problem. The naive way is to minimize $H(X) = G(X) + w^2 \sum_i (F_i(X))^2$, where w^2 is a great positive weight which penalizes the violation of constraints $F(X) = 0$. The method is simple and often efficient, and can be used in the first iterations of the solver or if an approximate solution (*i.e.,* a draft) is expected. But the latter does not converge to an exact (local or global) solution of $\mathcal{P}$.

Instead, we can consider the Lagrangian of $\mathcal{P}$:

$$\mathcal{L}(X, L) = G(X) + \sum_i L_i F_i(X)$$

where $L = (L_i)$ are Lagrangian multipliers. Solutions to the problem $\mathcal{P}$: $\min G(X)$ with constraints $F(X) = 0$ are solutions of the well-constrained system:

$$\begin{cases} 0 = \dfrac{\partial \mathcal{L}}{\partial X_k}(X, L) = \dfrac{\partial G}{\partial X_k}(X) + \displaystyle\sum_i L_i \frac{\partial F_i}{\partial X_k}(X) \\ 0 = \dfrac{\partial \mathcal{L}}{\partial L_i}(X, L) = F_i(X) \end{cases}$$

where derivatives are in practice approximated with finite differences. This system of equations is reduced to the unconstrained minimization of:

$$H(X, L) = \|\nabla \mathcal{L}\|^2 = \sum_k \left(\frac{\partial \mathcal{L}}{\partial X_k}(X, L) \right)^2 + \sum_i \left(\frac{\partial \mathcal{L}}{\partial L_i}(X, L) \right)^2 .$$

In our framework, we often know an initial value $X^{(0)}$ for X and we expect the solver to converge to a solution close to $X^{(0)}$. Unfortunately, we do not have starting values for the Lagrange multipliers. Inspired by the homotopy method, we define a continuum of problems $P^{(t)}$, depending on a real parameter t in $[0, 1]$:

$$P^{(t)} : \quad \min G^{(t)}(X) = tG(X) + (1-t)G^{(0)}(X), \quad t \in [0, 1] \\ \text{with } F^{(t)}(X) = tF(X) + (1-t)F^{(0)}(X) = 0.$$

Clearly $G^{(1)}(X) = G(X)$, and $F^{(1)}(X) = F(X)$, thus $P^{(1)}$ is $\mathcal{P}$. It remains to define $F^{(0)}(X)$ and $G^{(0)}(X)$, so that $X^{(0)}$ is a solution. Classically, we use:

$$F^{(0)}(X) = F(X) - F(X^{(0)}).$$

So $X = X^{(0)}$ is indeed a root of $F^{(0)}$. But $X = X^{(0)}$ must also minimize $G^{(0)}$. Thus, we choose:

$$G^{(0)}(X) = C \quad \text{or} \quad G^{(0)}(X) = \|X - X^{(0)}\|^2$$

where C is any non zero constant.

In both cases, null Lagrange multipliers are solution for $P^{(0)}$. Thus a solution for $P^{(0)}$ is known. Then problems $P^{(t)}$ are solved for $t > 0$, increasing t step by step, from 0 to 1, with increment $1/n$ in the simplest implementation. For each value of $t > 0$, the solver starts from the previous solution.

As already mentioned at the beginning of this section, the naive method is usually adequate to achieve the required accuracy while being faster than Lagrangian and homotopy. However, we use these two other methods as a last resort, in order to refine the results.

4.3. Optimizations

Classical methods of qualitative analysis and decompositions of systems of constraints, either combinatorial (based on flows or maximum matching) or witness-based, may significantly speed up the solving procedure. Unfortunately, they assume white boxes, and we did not try to adapt these methods to our framework.

Nevertheless, some optimizations are possible and have been implemented. Let e_i be the vector of the canonical base, *i.e.,* the ith line of the identity matrix. We often need to compute f at a point X and at a point $X + \lambda_i e_i$, for approximating the ith derivative with finite differences at point X with first order methods, or for direct search methods like Nelder–Mead, Torczon, Hooke–Jeeves. If the function f does not depend on the ith unknown X_i, then $f(X) = f(X + \lambda_i e_i)$ and computing both $f(X)$ and $f(X + \lambda_i e_i)$ is useless. Even when f indeed depends on X_i, f likely calls many functions which do not depend on X_i. As a consequence, specific labels and time stamps are associated to each variable to exploit that point.

These labels also avoid to recompute continuously a shared expression, *e.g.,* when GetX(E), GetY(E), GetZ(E) are called with the same expression E, especially when E is costly to compute. The same holds when ClosestPt(E) and closestNorm(E) are called with the same expression E. The main idea is maximal sharing: there is only one node which is associated to an expression E which is present several times in the description. This method is classical and stems back to LISP. Indeed, it is called hash consing. Labeling nodes with time stamps permits to evaluate a node only one time, i.e. the first time its value is needed. It is sometimes called memoization or memorization.

We did not try to generate a C program from DAGs, compile it using optimization options, link it dynamically with the DECO modeler and run it. Such an approach should take profit of powerful optimizing compilers like gcc, the GNU C Compiler.

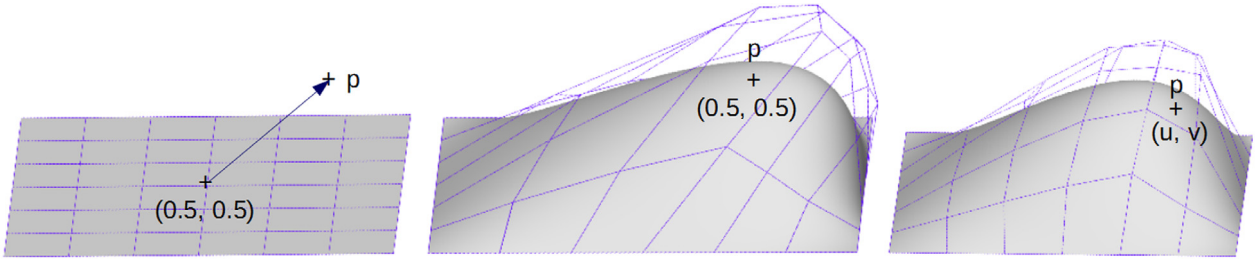

Fig. 3. Example 1: Initial problem (left), final position satisfying the constraints when (u, v) is constant (center) or variable (right).

5. Examples

5.1. Example 1: passing point

In this first and trivial example, we start from a flat B-spline surface S with 7×7 control points. We choose a biquadratic surface and take

$(0, 0, 0, 1/5, 2/5, 3/5, 4/5, 1, 1, 1)$

as knot vectors. We set a constraint to force the surface to pass through a 3D given point p. In the first case, we impose the constraint $S(0.5, 0.5) = p$. In the second case, we do not impose the parametric values so that the constraint is $S(u, v) = p$ with (u, v) unknown. We just initialize (u, v) to $(0.5, 0.5)$.

The control points on the borders of the surface are fixed and the 25 inner points are free to move. This distinction is done by specifying that the coordinates of the control points are either of type *Const* or type *Var*. The middle point of surface S corresponds to the expression $CalcPt(s, u, v)$. The constraint consists to equalize this point with the given point p.

$$CalcPt(s, u, v) = p.$$

The equalization of these two points implies three equations, *i.e.,* one for each coordinate of the points. These equations are described by three functions F_0, F_1 and F_2 which have to be equal to 0 at the solution.

$$F_0(X) = GetX(CalcPt(s, u, v)) - GetX(p) = 0$$
$$F_1(X) = GetY(CalcPt(s, u, v)) - GetY(p) = 0$$
$$F_2(X) = GetZ(CalcPt(s, u, v)) - GetZ(p) = 0.$$

As explained in Section 4.3, the function $CalcPt(s, u, v)$ is not computed three times but only once.

This system has a large set of solutions. We can choose a smooth solution by minimizing the discrete energy of the surface.

$$G(X) = DiscrE(s)$$

Fig. 2 illustrates the DAG representation of the two different systems. The grayed nodes represent scalar variables.

The solver evaluates these functions and modifies the values of the variables in order to satisfy the constraints and minimize the objective function (Fig. 3). In the first case, the system has $3 \times 5 \times 5 = 75$ unknowns corresponding to the coordinates of the inner points and 3 equations. In the second case, the solver has two additional degrees of freedom to solve the optimization problem. The system has now $3 \times 5 \times 5 + 2 = 77$ unknowns for the 3 same equations. The modification of u and v allows us to find a solution satisfying the constraints and having a lower discrete energy. In this example, the (u, v) coordinates, starting from $(0.5, 0.5)$ converge to $(0.748, 0.611)$. In the first situation, the discrete energy calculated was approximately 25, against 20 for the second one, which represents a decrease of 20%.

5.2. Example 2: contact area between two parametric surfaces

In this example, we have two cylindric-like shapes defined by several parametric B-spline surfaces constrained to be in contact along an area. Each pseudo closed cylinder is composed of 6 B-spline patches of degrees 2 with 5×5 control points. They are constrained to satisfy G^0 (for the covers) and G^1 continuity along their edges (Fig. 4), with equalities of points and unitary normal vectors. For each edge, the continuity is established on a discretized set of 9 points.

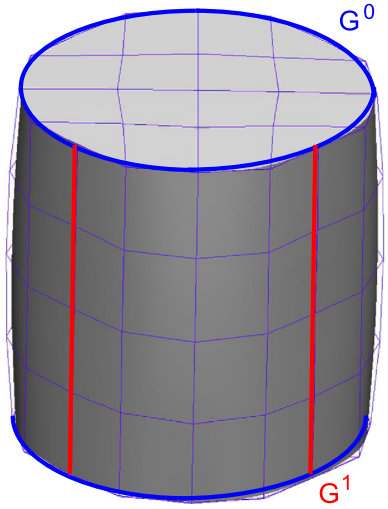

Fig. 4. Example 2: Cylindric-like shape composed by 6 surfaces with G^0 and G^1 blending conditions.

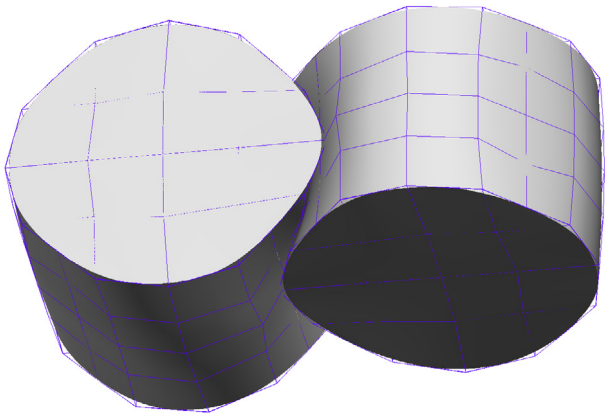

Fig. 5. Example 2: Contact area between two parametric objects.

For each cylinder, the contact area is inside one single patch. Join conditions with other patches must be ensured. The result is shown in Fig. 5.

The constraint is discretized into a finite set of punctual contact constraints. A contact area is given by a center and a radius in the parametric space. It is converted into a finite list of 2D points. We have chosen 17 points, as shown in Fig. 6.

Each surface s_1 and s_2 has its own contact area, represented as a point list z_1 and z_2. The contact is done for each point in z_1 with its

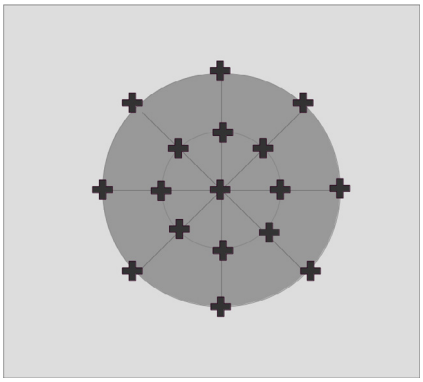

Fig. 6. Discretization of a contact area into a point list in the parametric space.

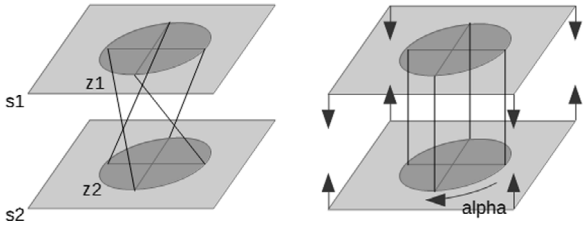

Fig. 7. Points mapping between two areas with a varying angle.

corresponding point in z_2, by equalizing points and normals vectors on the surfaces. If (u_{1_i}, v_{1_i}) and (u_{2_i}, v_{2_i}) are corresponding points belonging respectively to zones z_1 and z_2, the contact constraint is established by requiring:

$$CalcPt(s_1, u_{1_i}, v_{1_i}) = CalcPt(s_2, u_{2_i}, v_{2_i})$$
$$CalcNorm(s_1, u_{1_i}, v_{1_i}) = -CalcNorm(s_2, u_{2_i}, v_{2_i}).$$

The method must not be dependent on the orientation of the surfaces. To achieve this, we give a degree of freedom to one of the contact areas to spin round. The coordinates in z_1 are constant values, and the coordinates in z_2 are the image of z_1 after a rotation of an unknown angle *alpha*, as shown in Fig. 7. These coordinates are represented by compound scalar DAGs depending on constant values and the variable *alpha*.

Due to the computation model, the contact is completed using approximations (in an error range). This cannot guarantee limited intersections of the two objects.

5.3. Example 3: contact area between a parametric surface and a subdivision surface

This example is similar to the previous description except that one of the shapes to put in contact is a subdivision surface. This second surface is the result of an irregular subdivision process after a given number of iterations applied on an initial mesh [34]. The initial mesh is a cube. Since the edges are weighted, the subdivision process generates an irregular surface mesh (Fig. 8) with more or less rounded areas. The result is shown in Fig. 9.

As the second surface is a subdivision surface, it is not possible to use the functions *CalcPt* and *CalcNorm*. Instead, the *ClosestPt* and *ClosestNorm* functions are used. Each point on s_1 belonging to the chosen area is constrained to be in contact with its closest point on s_2.

$$CalcPt(s_1, u, v) = ClosestPt(CalcPt(s_1, u, v), s_2)$$
$$CalcNorm(s_1, u, v) = -ClosestNorm(CalcPt(s_1, u, v), s_2).$$

Note that this last formulation using $ClosestPt(\ldots, s_2)$ instead of $CalcPt(s_2, \ldots)$ is more generic, and could also be used in the previous examples.

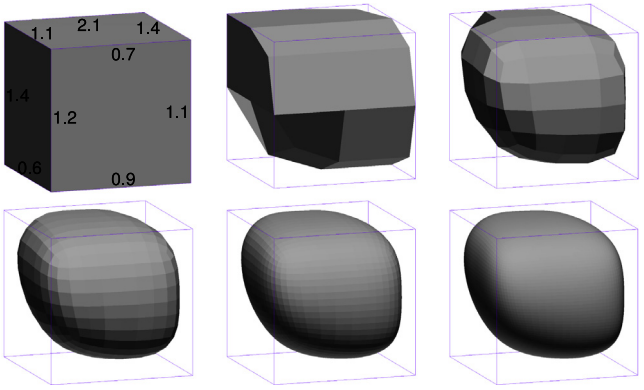

Fig. 8. First iterations of an irregular subdivision surface with weighted edges.

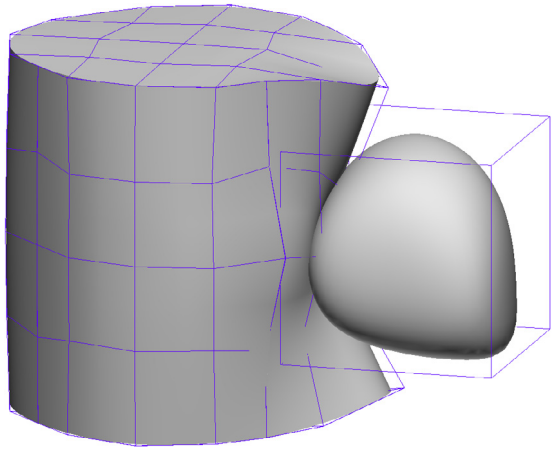

Fig. 9. Example 3: Contact area between a parametric surface (left) and a subdivision surface (right).

5.4. Example 4: gearing

The proposed solver offers other possibilities as demonstrated in this example of gearing. We focus here on the creation of a complete constraints set but the exact modeling of a gear is not really part of the intended application area. The result obtained is only an outline, not taking into account common criteria used in the engineering industry, *e.g.,* clearance. We consider two gears having given numbers of teeth z_1 and z_2. The teeth should have a particular shape so that the gear assembly is working. Each gear rotates at a speed inversely proportional to the number of teeth. Throughout this movement, there must be only contact point between a tooth and the opposite tooth. This property may be reflected in our formalism by contact constraints such as described in the previous examples. However, this constraint is not sufficient to determine the profile of the teeth. The gears have a known pressure angle characteristic that determines the direction of the applied force between the gears, or equivalently the normal to the surface at the contact point, but also the direction of the path of contact. We consider this pressure angle, denoted α as a parameter of the system. To determine the profile of the teeth, we first study each gear independently, by setting passing point constraints between the tooth (after rotation) and a point on the path of contact (Fig. 11).

Initially, we define the set of surfaces which compose the gear, and the set of variables that determine them. The gear contains z_1 patterns identical by rotation. Each pattern is composed of two cylindrical parts A and B, and two surfaces S and T (see Fig. 10). We consider that S and T are symmetrical to one another. The surface S itself determines the entire object. This surface is a B-spline surface

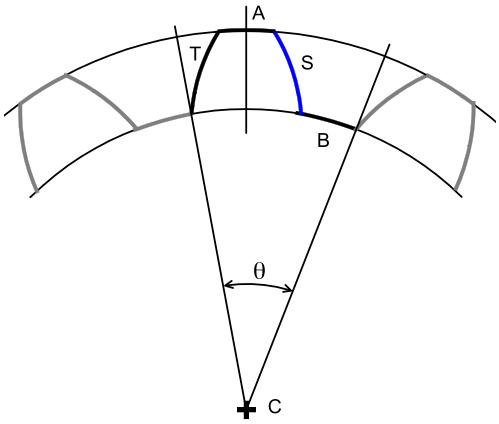

Fig. 10. Example 4. Decomposition into surfaces patches and identification of the geometrical parameters.

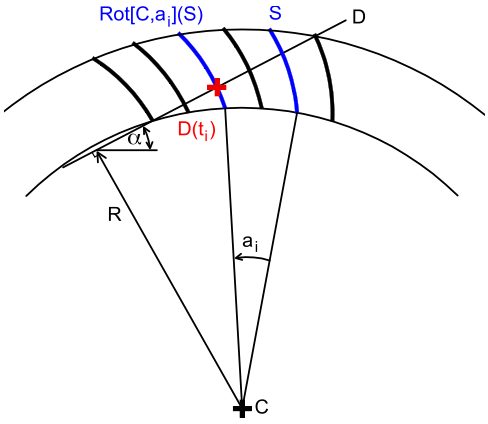

Fig. 11. Example 4. Geometrical representation of the constraints.

of degree 2×1 with $n_p\times2$ control points. The two rows of n_p control points are in a plane and identical by translation. The variables are the coordinates of one of this row. Then, the surface T is deduced by symmetry. A and B are cylindric parts with 3×2 control points and degrees 2×1; their boundaries and radii are determined in function of S and T. The z_1-1 other patterns are deducted by rotation of angle $k\theta$ where $\theta=2\pi/z_1$.

We consider that the path of contact is expressed in a parametric form $D(t)$. We also assume that the function $Rot[c,a]$ computes the image of a point or a surface by a rotation of center c and angle a. For a discretized set of values $\{t_1,\ldots,t_{n_c}\}$ and angles $\{a_1,\ldots,a_{n_c}\}$, we set the following constraints (as shown in Fig. 11):

$$\forall i\in\{1,\ldots,n_c\},\quad CalcPt(Rot[C,a_i](S),u_i,v_i)=D(t_i).$$

The list of u_i coordinates can be unknown or predefined constant values from 1 to 0, and the v coordinate can be the constant 0 because the problem is 2D. The discretized sets of t_i and a_i have a constant step and must have a speed in accordance one with the other :

$$\forall i\in\{1,\ldots,n_c-1\},\quad \overrightarrow{D(t_i)D(t_{i+1})}=R(a_{i+1}-a_i)\,\vec{d}\;=\;\vec{\Delta}$$

where R is the distance from D to C, $\vec{d}$ is the unit direction vector of D and $\vec{\Delta}$ is the displacement of the contact point for each step.

Note that computing a rotation on a surface is equivalent to compute the rotation for each control point. An equivalent way to set the constraint is to compute the rotation after calculating the point on the surface, which is less time consuming for the solver. Indeed, we have:

$$CalcPt(Rot[C,a_i](S),u_i,v)=Rot[C,a_i](CalcPt(S,u_i,v)).$$

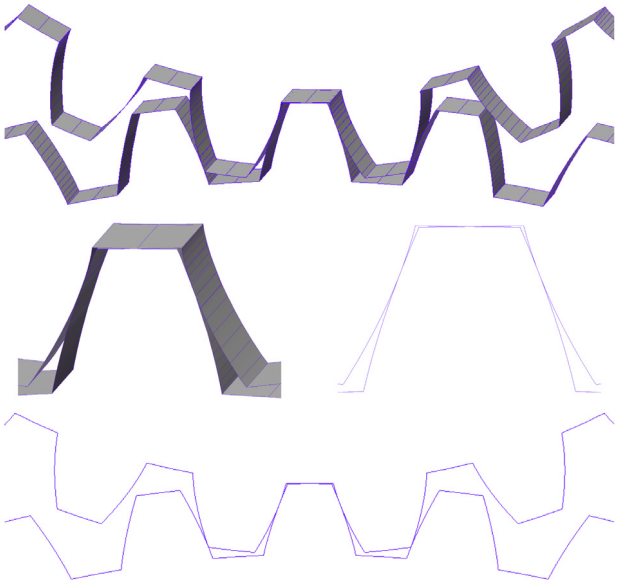

Fig. 12. Example 4: Positioning the two gears satisfying the constraints. Theoretical result without clearance.

We could add other constraints on the normal vector:

$$\forall i\in\{1,\ldots,n_c\},\quad Rot[C,a_i](CalcNorm(S,u_i,v_i))=\vec{d}\,.$$

However, it is not necessary to include them in the system because they are implicitly satisfied.

As said previously, the geometry is completely determined by n_p control points of S belonging to a given plane, which represents $2n_p$ degrees of freedom. Each of the n_c constraints sets a contact between two points of the plane, which theoretically decreases the total number of degrees of freedom by $2n_c$. We choose $n_c>n_p$ in order to ensure that the solution is determined without ambiguity. The problem is over-constrained but consistent. The theoretical solution is an involute of circle. A B-spline cannot exactly represent this curve but we can get a good approximation by choosing a relatively large number of control points. Although there is no solution satisfying exactly all the constraints at the same time, the solver will give a solution whose constraints are the closest to being met.

Whatever the numbers $n_c>n_p$ we choose, the system has three remaining degrees of freedom. The discretized intervals t_i and a_i can be offsetted or simultaneously scaled. These degrees of freedom act geometrically on the internal and external diameters and position of S up to a rotation around C. We set a value to the internal and external diameters. The third degree of freedom is reduced by setting the tooth thickness equal to the tooth space, which means that we consider the offset equals zero. An example of result is shown in Fig. 12 with parameters $z_1=51$, $z_2=29$, $\alpha=20°$, $n_p=15$ and $n_c=29$. The unknowns of the system are the $2n_p$ coordinates of the 2D control points, the $2n_c$ values of t_i and a_i and the 2 coordinates of $\vec{\Delta}$, making a total of $2n_p+2n_c+2=90$ unknowns. Given the fact that the problem is 2D, there are $2n_c+4(n_c-1)+3=173$ equations.

5.5. Example 5: bump/hollow

This last example corresponds to the action of creating a hollow in a car door in order to let room for inserting the hand behind the handle. For a simpler illustration, we start from a flat surface and we apply a distortion to create a hollow. As we study it from behind, it becomes a bump. The surface is constrained to reach a target curve (a piece of ellipse) while remaining fixed along an outline curve (an other ellipse). The outline is discretized into $n_1=$

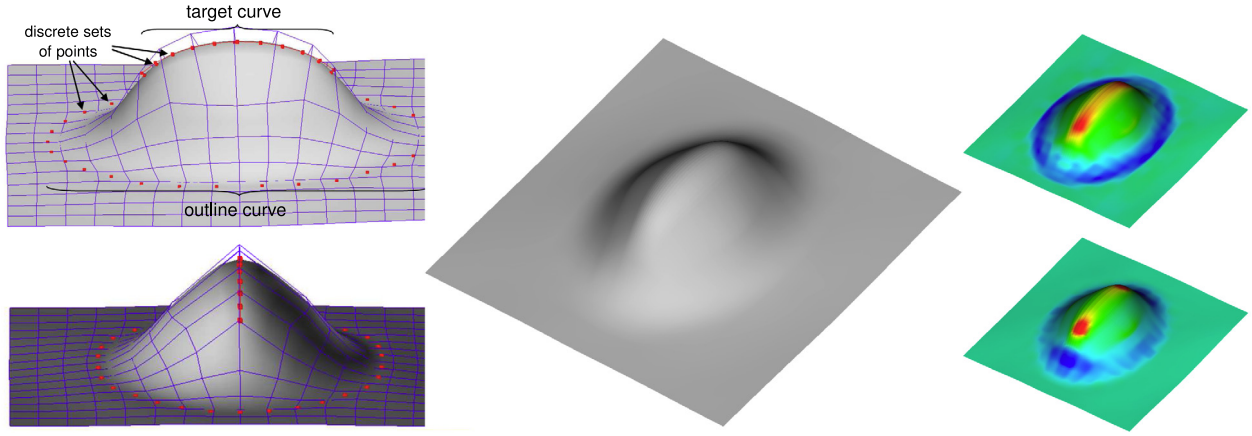

Fig. 13. Example 5. Simulation of a hollow in a car door. Final position satisfying the constraints. Outline and target curves discretization represented by red points (left), visualization of the mean (top right) and Gaussian (bottom right) curvatures. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Example 5. Evolution of the errors on outline and target curves depending on the number of iterations and the computation time.

iter	0	50	100	150	200	250	300
time(s)	0	45	86	126	159	192	224
max1	0	0.127	0.059	0.015	0.0027	0.0029	0.00085
rmse1	0	0.086	0.031	0.007	0.0017	0.0015	0.00063
max2	0.229	0.115	0.017	0.010	0.0056	0.0053	0.00415
rmse2	0.212	0.127	0.010	0.006	0.0037	0.0021	0.00166

30 3D points $(P1_1, \ldots, P1_{n_1})$ and the target curve into $n_2 = 11$ points $(P2_1, \ldots, P2_{n_2})$, drawn with red dots in Fig. 13(left).

We label $S0$ the initial surface with constant given control points and $S1$ its final position. These surfaces are biquadratic B-splines and have 17×17 control points. Control points of $S1$ are variables initialized with the same values as $S0$. For each outline point $P1_i$ projected on $S0$, the point on S at variable coordinates $(u1_i, v1_i)$ is constrained to have the same position and normal as $S0$:

$$\forall i \in \{1, \ldots, n_1\},$$
$$\begin{cases} CalcPt(S, u1_i, v1_i) = ClosestPt(S0, P1_i) \\ CalcNorm(S, u1_i, v1_i) = ClosestNorm(S0, P1_i). \end{cases}$$

The surface is also forced to go through the target curve for each point $P2_i$ at a point of variable coordinates $(u2_i, v2_i)$:

$$\forall i \in \{1, \ldots, n_2\}, CalcPt(S, u2_i, v2_i) = P2_i.$$

Surface $S1$ has $17 \times 17 \times 3 = 867$ variable 3D coordinates and $n_1 + n_2 = 41$ variable 2D coordinates, making a total of 908 variables. There are $n_1 \times 6 + n_2 \times 3 = 213$ equations. Fig. 13 shows the result after solving the constraints.

Some quantitative results are given in Table 1. It shows the accuracy related to the number of iterations or the computing time in seconds. The values *max*1 and *rmse*1 correspond to the maximum and root mean square error on the outline ellipsis (the side length of the surface is 1). The values *max*2 and *rmse*2 are the errors to the target curve. We used the BFGS method, without Lagrangian and homotopy. The hardware configuration is Intel® Core™ i7-4800MQ CPU 2.70 GHz × 8 with 16 GB RAM, but running on one core.

At the initialization, the constraints on the outline are already satisfied since the surface S has the same control points as $S0$. During a first phase, the algorithm seeks to improve constraints on the target curve, even if it deteriorates the outline constraints. Reaching about 50 iterations, the algorithm improves all the constraints and makes them tend towards zero. The algorithm stops at about 300 iterations because of numerical accuracy limits due to numerical gradient estimations based on a first order Taylor expansion. Fig. 13 represents graphically the mean (top right) and Gaussian (bottom right) curvatures of the final surface. One can notice that the curvature distributions are regular, despite we have chosen only a biquadratic surface. The error values (Table 1) correspond to our expectations for what is considered as initial drafting stage.

Such descriptions can be embedded in high level operators. We are currently developing an operator that takes intuitive parameters in "natural" language and creates automatically a DAG. Those parameters can be of different types (quantifier, localization, etc.) and can take different values defined in dictionaries ("normally", "very", "few", "center", "right", "up", etc.). The parameters of the operator and the values we gave for this example are position: center, incline: horizontal, long: very, wide: normally, deep: normally and direction: down. Those parameters are automatically converted into a mathematical description by defining the outline and target curves.

6. Conclusion

The DECO prototype described in this paper showed the feasibility and the promises of variational geometric modeling with black box DAGs, *i.e.,* without equations. The proposed 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 depending on the user's goal. The user can also create simple constraints and use them to create his/her more elaborate ones. This work raises several issues, which are actually scientific locks. We mention some of them in the following.

Constant subdivision levels. Up to now, subdivision levels for subdivision surfaces are fixed at initialization and remain constant, for simplicity.

Vocabulary. What are the most convenient high level black nodes? DECO suggests: CalcPt, CalcNorm, ClosestPt, ClosestNorm, but other routines may be interesting: volume or other moments (inertia), intersection with a line, etc. In Computer Graphics, especially in Ray Casting, only three geometric functions are needed: the intersection of an object with a given ray, the normal at a given point (maybe the osculating quadric or an approximation), and the bounding box or sphere of an object. Of course, the color of an object at a given point is also needed, but it is not a geometric function. These three simple functions are sufficient to ray trace all possible objects and scenes. What is the corresponding set of functions for geometric modeling? Hoffmann et al. [35] address a similar issue: which queries can assure the interoperability between different geometric modelers? They propose a basic set of queries, including point membership test and distance computation.

Granularity. What is the convenient granularity level for DAGs? Of course, we may achieve Turing completeness if we introduce arithmetic nodes for $+, -, \times, \div$, cos, exp, etc., a node for the IF-THEN-ELSE instruction, and nodes for defining and calling functions. But such a fined-grained level is not user friendly. Designers and engineers may not feel comfortable using these tools. For computer scientists too, it is not convenient since they prefer their favorite programming language and IDE (integrated development environment). A node EXPR (similar to the Unix tool expr in the Shell languages), for evaluating simple mathematical expressions, seems a good tradeoff.

Persistent naming. When parameters values are modified, some part or feature (*e.g.,* a chamfered edge) may disappear or be disconnected. Naming geometric parts in an unambiguous and robust way is known as the Persistent Naming problem [36] in CAD–CAM. Our examples are simple enough to avoid this problem for now. In the future, we have to use methods for solving the Persistent Naming problem.

Discontinuity. A sticking point is to handle discontinuities. Homotopy assumes continuity and may likely fail with discontinuities. For instance, ClosestPt(s, p) is continuous almost everywhere, but not everywhere: imagine that p is moving in a hollow sphere between two points on the same diameter. By the way the related function Distance(s, p) is continuous everywhere. Dynamic Geometry already faced the discontinuity problem.

Failures and loops. For now, we assume that the evaluation of DAGs always succeeds. But, with a more complex environment, the evaluation of a DAG may abort, raise an exception, or loop. The top level program which evaluates DAGs should catch exceptions, and abort computations after some delay.

Labels on nodes. It may be convenient to label DAG nodes with properties of the underlying function: convexity, continuity, differentiability, polynomiality, degree, linearity, etc., so that the procedure evaluating $F(U)$ or $G(U)$ may exploit these information.

We did not try to optimize the computations but they can obviously be reduced in time. For example, parallel and cloud computing will make this approach scalable. The functions $F(U)$ and $G(U)$ are evaluated many times, and this can be done in parallel. Each call to $F(U)$ can also be computed in parallel. Notice in addition that most of the computing time is due to the final steps of the solving process. If only an approximate solution is expected (a draft model), the solver can be stopped before the convergence, thus saving computing time. The potential failure of convergence must also be studied. Starting with a starting point not too far from the solution reduces the risks.

Since any black box can easily be added and contains any type of algorithm, this approach widens the scope of geometric design with constraints. In the future, our approach may apply to many other applications: for CAD, the computation of bars lengths for the synthesis of mechanisms; for animation and special effects in movies, the computation of the initial state and impulse in order to reach a given state after a physical and kinematic simulation; etc.

Evidently, topics for future works are multimodal interfaces, and an higher-level language for specifying shapes, constraints and goals. Ideally users express their needs in a language close to a natural one and the DECO software translates the sentences into a list of instructions. We are currently working on this higher level approach.

Finally, DECO permits to easily produce draft models. After exportation in some standard exchange format, the later can be edited with any traditional CAD modeler. Unfortunately, during this exportation, the semantic information in the initial description is lost. Keeping this semantic would be valuable to optimize forthcoming processings on the object, but it is not compatible with the current generation of CAD modelers. Keeping the semantic is also essential for "semantic PLM", *i.e.,* ontology-based standards [37].

Acknowledgments

The authors would like to thank the two French Institutes Carnot ARTS and Carnot STAR for their support to this research project. Lincong Fang thanks for their support the National Natural Science Foundation of China (No. 61272300), the Zhejiang Provincial Natural Science Foundation of China (LQ13F020003) and the China Scholarship Council.

They also did appreciate the valuable comments provided by the reviewers.

References

[1] Hoffmann CM. Geometric and aolid modeling: An introduction. Morgan Kaufman; 1989.
[2] Piegl L, Tiller W. The NURBS book. Springer Verlag; 1995.
[3] Shah J, Rogers M. Expert form feature modeling shell. Comput Aided Des 1988; 20(9):515–24.
[4] Bettig B, Hoffmann CM. Geometric constraint solving in parametric computer-Aided Design. J Comput Inf Sci Eng 2011;11(2):021001.
[5] Jermann C, Trombettoni G, Neveu B, Mathis P. Decomposition of geometric constraint systems: a survey. Int J Comput Geom Appl (IJCGA) 2006; 16(05n06):379–414.
[6] Farin G. Curves and surfaces for computer aided geometric design: A practical guide. 4th ed. Academic Press; 1997.
[7] Shah J, Mäntylä MM. Parametric and feature-based CAD/CAM. Wiley Sons Inc.; 1995.
[8] Warren JD, Weimer H. Subdivision methods for geometric design: A constructive approach. San Francisco: Morgan Kaufmann; 2002.
[9] Cashman TJ. Beyond Catmull-Clark? A survey of advances in subdivision surface methods. Comput Graph Forum 2012;31(1):42–61.
[10] Lin V, Gossard D, Light R. Variational geometry in computer aided design. ACM Comput Graph 1981;15(3):171–7.
[11] Anson O, Seron FJ, Gutierrez D. NURBS-based inverse reflector design. In: Matey L, Torres JC, editors. CEIG 08—congreso espanol de informatica grafica. The eurographics association; 2008. http://dx.doi.org/10.2312/LocalChapterEvents/CEIG/CEIG08/065-074.
[12] Bloor MIG, Wilson MJ. Generating blend surfaces using partial differential equations. Comput Aided Des 1989;21(3):165–71.
[13] Ge J-X, Chou S-C, Gao X-S. Geometric constraint satisfaction using optimization methods. Comput Aided Des 1999;31:867–79.
[14] Hu S-M, Li Y, Ju T, Zhu X. Modifying the shape of nurbs surfaces with geometric constraints. Comput Aided Des 2001;33:903–12.
[15] Hoffmann CM, Juan R. Erep: an editable, high-level representation for geometric design and analysis. In: IFIP TC5/WG5.2 working conference on geometric modeling for product realization. North-Holland Publishing; 1992. p. 129–64.
[16] Laborde J-M, Strässer R. Cabri-géomètre: A microworld of geometry for guided discovery learning. Zentralblatt didakt math 1990;90(5):171–7.
[17] Kortenkamp U. Foundations of dynamic geometry [Ph.D. thesis]. Swiss Federal Institute of Technology Zurich; 1999.
[18] Richter-Gebert J, Kortenkamp UH. User manual for the interactive geometry software cinderella. Springer Science & Business Media; 2000.
[19] Hohenwarter M, Preiner J. Dynamic mathematics with GeoGebra, J Online Math Appl; 7.
[20] Sangwin C. A brief review of geoGebra: dynamic mathematics. MSOR Connect 2007;7(2):36–8.

[21] Kortenkamp U, Richter-Gebert J. Decision complexity in dynamic geometry. In: Automated deduction in geometry. Springer; 2001. p. 193–8.
[22] Denner-Broser B. About tracing problems in dynamic geometry. Discrete Comput Geom 2013;49(2):221–46.
[23] Hidalgo MR, Joan-Arinyo R. The reachability problem in constructive geometric constraint solving based dynamic geometry. J Autom Reasoning 2014;52(1):99–122.
[24] The CGAL Project, CGAL User and Reference Manual, 4.5.2 Edition, CGAL Editorial Board, 2015. URL http://doc.cgal.org/4.5.2/Manual/packages.html.
[25] CGAL: Computational geometry algorithms library, http://www.cgal.org.
[26] Chan A, Mourad K. Theoretical computers and diophantine equations. Math Medley 1990;18(2):66–77.
[27] Stam J. Exact evaluation of Catmull–Clark subdivision surfaces at arbitrary parameter values. In: Proceedings of SIGGRAPH. 1998. p. 395–404.
[28] Bousquet J, M D. Flaw removal on surfaces. In: Curves and surfaces with applications in CAGD. Vanderbilt University Press; 1997. p. 43–52.
[29] Borrelli V, Cazals F, Morvan J-M. On the angular defect of triangulations and the pointwise approximation of curvatures. Comput Aided Geom Des 2003; 20:319–41.
[30] Meyer M, Desbrun M, Schröder P, Barr AH. Discrete differential-geometry operators for triangulated 2-manifolds. In: Hege H-C, Polthier K, editors. Visualization and mathematics III. Heidelberg: Springer-Verlag; 2003. p. 35–57.
[31] Nesterov Y. Introductory lectures on convex optimization, vol. 87. Springer Science & Business Media; 2004.
[32] Knoll DA, Keyes DE. Jacobian-free Newton–krylov methods: a survey of approaches and applications. J Computat Phys 2004;193(2):357–97. URL http://www.cs.odu.edu/~keyes/papers/jfnk.pdf.
[33] Kolda TG, Lewis RM, Torczon V. Optimization by direct search: New perspectives on some classical and modern methods. SIAM Rev 2003;45(3): 385–482.
[34] Reusche L. Conversion of trimmed NURBS surfaces to subdivision surfaces [M.S. thesis]. Germany: Technical University Braunschweig; 2005.
[35] Hoffmann C, Shapiro V, Srinivasan V. Geometric interoperability via queries. Comput Aided Des 2014;46:148–59. 2013 SIAM Conference on Geometric and Physical Modeling. http://dx.doi.org/10.1016/j.cad.2013.08.027. URL http://www.sciencedirect.com/science/article/pii/S001044851300167X.
[36] Marcheix D, Pierra G. A survey of the persistent naming problem. In: Symposium on solid modeling and applications. 2002. p. 13–22. http://dx.doi.org/10.1145/566282.566288. URL http://doi.acm.org/10.1145/566282.566288.
[37] Barbau R, Krima S, Rachuri S, Narayanan A, Fiorentini X, Foufou S, Sriram RD. Ontostep: Enriching product model data using ontologies. Comput Aided Des 2012;44(6):575–90. http://dx.doi.org/10.1016/j.cad.2012.01.008. URL http://www.sciencedirect.com/science/article/pii/S001044851200022X.