Information Sciences 182 (2012) 56–76

Contents lists available at ScienceDirect

Information Sciences

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

Iterative two-step genetic-algorithm-based method for efficient polynomial B-spline surface reconstruction

Akemi Gálvez, Andrés Iglesias *, Jaime Puig-Pey

Dept. of Applied Mathematics and Computational Sciences, University of Cantabria, Avda. de los Castros s/n, E-39005 Santander, Spain

ARTICLE INFO

Article history:
Available online 8 October 2010

Keywords:
Evolutionary computation
Genetic algorithms
Reverse engineering
Surface reconstruction
B-spline surfaces

ABSTRACT

Surface reconstruction is a very challenging problem arising in a wide variety of applications such as CAD design, data visualization, virtual reality, medical imaging, computer animation, reverse engineering and so on. Given partial information about an unknown surface, its goal is to construct, to the extent possible, a compact representation of the surface model. In most cases, available information about the surface consists of a dense set of (either organized or scattered) 3D data points obtained by using scanner devices, a today's prevalent technology in many reverse engineering applications. In such a case, surface reconstruction consists of two main stages: (1) surface parameterization and (2) surface fitting. Both tasks are critical in order to recover surface geometry and topology and to obtain a proper fitting to data points. They are also pretty troublesome, leading to a high-dimensional nonlinear optimization problem. In this context, present paper introduces a new method for surface reconstruction from clouds of noisy 3D data points. Our method applies the genetic algorithm paradigm iteratively to fit a given cloud of data points by using strictly polynomial B-spline surfaces. Genetic algorithms are applied in two steps: the first one determines the parametric values of data points; the later computes surface knot vectors. Then, the fitting surface is calculated by least-squares through either SVD (singular value decomposition) or LU methods. The method yields very accurate results even for surfaces with singularities, concavities, complicated shapes or nonzero genus. Six examples including open, semi-closed and closed surfaces with singular points illustrate the good performance of our approach. Our experiments show that our proposal outperforms all previous approaches in terms of accuracy and flexibility.

1. Introduction

One of the most visible effects of globalization is the growing global competition among manufacturers and industrial corporations to deliver more competitive products with better quality and lower prices. As a consequence, manufacturers are constantly challenged to optimize processes in order to reduce the product development time cycle. For instance, digital data must be available for efficient storage and transmission between different providers and manufacturers. Reverse engineering is a crucial technology in this process. While conventional engineering transforms engineering concepts and models into real parts, in reverse engineering real parts are transformed backward into engineering models and concepts. Advantages of this process are obvious: digital models can be used even although the real objects are lost or become unavailable or unusable. Further, digital models are also often easier and cheaper to modify and analyze than the physical objects themselves.

* Corresponding author.
E-mail address: iglesias@unican.es (A. Iglesias).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.09.031

Currently, laser scanner is the most common method used in reverse engineering applications because it is fast and robust relative to other methods. Laser scanner systems yield a (possibly massive) cloud of 3D data points from which a 3D model is to be reconstructed. This latter process, called surface reconstruction, has many remarkable advantages. By using a mathematical surface model, the same complex shape can be economically represented by as few as 50–500 parameters. This issue becomes increasingly important with the rapid development of powerful communication technologies, making it possible to send those models all over the world in seconds. Furthermore, parametric mathematical models such as B-splines are the preferred representations in engineering design applications because they can be readily modified by changing only a small set of parameters, such as control points or knot vectors.

To obtain a reliable surface reconstruction method for parametric surfaces is however not an easy task. After decades of intensive research, parametric surface reconstruction is still a challenging problem and many data sets cannot be properly reconstructed. The motivation of this paper is to develop an efficient method for surface reconstruction by using polynomial B-splines and with the ability to reconstruct a wide variety of complicated shapes while achieving the highest degree of accuracy.

In this context, the present paper introduces a new method for surface reconstruction from clouds of noisy 3D data points by using polynomial B-spline surfaces as fitting surfaces. The originality of our approach lies in the fact that genetic algorithms are applied iteratively in two steps: the first one determines the parametric values of data points; the later computes surface knot vectors. Then, the fitting surface is calculated by least-squares through either SVD (singular value decomposition) or LU methods. The method yields very accurate results even for surfaces with singularities, concavities, complicated shapes or nonzero genus. Further, it outperforms all previous approaches in terms of flexibility, accuracy and computation times. All these claims will be adequately justified throughout the paper.

1.1. Evolutionary computation

Evolutionary computation (EC) is a subfield of artificial intelligence aimed at solving combinatorial optimization problems by using processes and methods inspired by biological mechanisms of evolution, such as selection, reproduction and survival of the fittest. The field encompasses computer-based problem solving systems involving metaheuristic optimization algorithms such as evolutionary algorithms (genetic algorithms, evolutionary programming, evolutionary strategies, genetic programming and so on) and swarm intelligence (ant colony optimization, particle swam optimization) along with many other evolutionary techniques (artificial immune systems, self-organization systems, cultural algorithms, differential evolution, etc.). Among them, *genetic algorithms* (GA) have been recognized as one of the most powerful computational paradigms for optimization and global search problems. Originated from the seminal work of John Holland in the 70s [29], they combine bio-inspired processes emulating genetic evolution (namely, natural selection, mutation and crossover) in order to describe the growth and development of populations associated with the target problem. In GA populations are formulated as abstract representations (called *chromosomes*) of candidate solutions (called *individuals* or *phenotypes*) to an optimization problem. Typically, evolution starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness (a measure of the quality of the represented solution) of every individual in the population is evaluated. Multiple individuals are then stochastically selected based on their fitness and modified under the action of the genetic operators to form a new population to be subsequently used in the next iteration of the algorithm. Ideally, the algorithm is expected to evolve over time toward better solutions, although convergence to global optima cannot be generally assured.

In spite of their surprising simplicity, GA are seen nowadays as a powerful tool to solve complex optimization problems in various fields of applications. Examples include pattern-classification problems [33], association of visual textures with human perceptions [24], manufacturing process modeling [8], forensic identification [35], optimization of fuzzy-logic controllers for perturbed autonomous wheeled mobile robots [51], multimodal biometry [27], tuning of PID controllers [83], companies failure prediction [64], control charts size determination [39], spectral estimation of time series [34], path planning [72] and multi-objective optimization [65]. In order to improve performance and/or overcome certain drawbacks (such as premature and slow convergence to the global minimum), genetic algorithms have been either modified [48,66] or combined with other approaches, such as evolutionary [21,82], rough sets [9] and fuzzy logic [4,32] techniques. More details on GA will be given in Section 4. The interested reader is referred to the nice books in [1,20,53] for an in-depth overview about genetic algorithms, their main features, variants and applications.

1.2. Surface reconstruction

The problem of finding a mathematical description of the 3D shape of a physical surface, usually referred to as *surface reconstruction*, has been a hot topic of research for the last two decades. Given partial information about an unknown surface U, the goal of surface reconstruction methods is to construct, to the extent possible, a compact representation of a surface model S that approximates U. This problem appears ubiquitously in *reverse engineering*, a field that aims at transforming real parts into engineering models and concepts for further use in manufacturing and rapid prototyping in the automotive, aerospace, ship building, shoe last, etc. industries. Depending on the available information about the surface (2D sections, clouds of points, grid of curves, interactive surface sketching, etc.), qualitatively different approaches can be formulated in order to tackle this issue (see Section 2 for details).

In this paper we consider the case of an initial input comprised of a (possibly massive) set of (noisy) 3D data points. In other words, we assume that no additional*a priori*information about the surface is available and therefore we must rely completely on the sampled set of data points. This has become the most common input in surface reconstruction, because of the wide availability of new affordable optical devices for 3D scanning and measurement. Our aim is to reconstruct the surface from such data points by using a (strictly polynomial) tensor-product B-spline surface (by far the most common parametric surface representation in CAD/CAM-based industries and computer graphics) as fitting surface. In this case, surface reconstruction comprises two main stages: (1) surface parameterization and (2) surface fitting. Both are critical in order to recover surface geometry and topology and to obtain a good fitting to data points. A proper parameterization is essential in order to determine the relationships among neighbor points in the surface parametric domain, and hence surface's topology and boundaries. Furthermore, suitable surface fitting requires a good parameterization of data points. But as soon as the parametric values of data points are considered as variables, surface reconstruction becomes a complicated high-dimensional nonlinear optimization problem. On the other hand, even with a good parameterization, there is no guarantee that the resulting fitting surface will be smooth enough. For instance, improperly chosen knot vectors might lead to a poor matching of data points. As it will be shown later on, our proposed method overcomes these problems at full extent.

1.3. Overview of the method

In this paper, we introduce a new method to address the surface reconstruction problem by using a genetic algorithm approach. Our proposal is briefly summarized in Fig. 1. A set of 3D data points obtained from a laser scanning process are fitted with an unknown tensor-product B-spline surface of a certain order. In order to do so, two genetic algorithm methods are applied: the first one determines the parametric values of data points; the later computes surface knot vectors. Then, the fitting surface is calculated by *LSQ* (least-squares) through either *SVD* (singular value decomposition) or a modification of the LU method (called *LU-mod* onwards). Reconstructed data points are then compared with the original ones. This pipeline is applied iteratively until a prescribed threshold error for data points is achieved.

1.4. Main contributions and structure of this paper

Main contributions of our method are:

- *Generality*: Our method is quite general: input data consists exclusively of clouds of 3D points. We do not impose any constraint (such as continuity, differentiability or others) on the original surface data points come from. The method performs well no matter data points come from surfaces given in polynomial, rational, trigonometric, free-form or any other family of functions. In addition, the method can reconstruct surfaces regardless they are open, closed or semi-closed. This is not a trivial question, since we restrict ourselves to strictly polynomial B-spline surfaces, as opposed to other methods which make use of NURBS surfaces, which are very well suited to represent quadrics and closed surfaces. Our method can

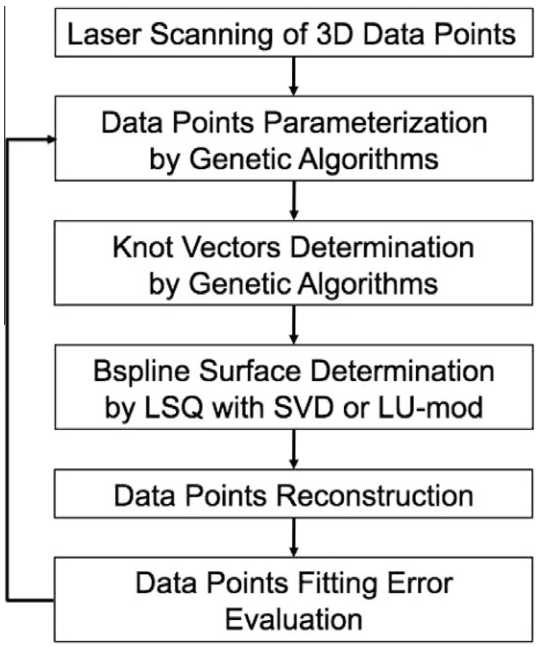

Fig. 1. Workflow of the proposed method.

also deal with surfaces exhibiting singularities (see Examples 2, 3, 4 and 6 below) or concavities (see Examples 2, 3, 5 and 6). Furthermore, as shown in Example 5, surfaces must not necessarily be of genus zero as it happens in many surface reconstruction methods. These are outstanding result since (as it will shown in Section 6.3) all previous methods fail to reconstruct surfaces in Examples 2–6 as meshless analytical surfaces.

- *Global method*: Most of surface reconstruction approaches require the combination of several methods or techniques in order to fully determine suitable values for all variables of the problem. Even worse, they fail to fully accomplish all steps of the reconstructon process. For instance, surface parameterization is not computed, but assumed (see Section 6.3 for details). Previous methods rely on the projection of data points onto a base flat surface (usually a plane) and usually skip the determination of knot vectors, which are also assumed. All these assumptions prevent those methods from effectively reconstructing non-trivial shapes. As opposed, our method is global and self-contained in the sense that it returns all data necessary to recover the surface without the addition of extra methods or further any pre-/post-processing.
- *Good performance*: To test our method, we considered several point clouds from known surfaces of different types including open, closed and semi-closed surfaces with singular points, nonzero genus and complicated shapes. Notably, our method has been able to replicate the original surfaces (written in terms of strictly polynomial B-spline surfaces) in all cases. This is a remarkable result, as polynomial B-spline basis functions lack the flexibility inherent to rational basis functions such as NURBS. This good performance does not come at the cost of expensive computation times. Back to front, our results show that our CPU times are much better than those of previous approaches (see Section 6.4 for details).
- *Accuracy*: In the reported examples, mean error of data points is about 10^{-4} for the worst cases, but generally as good as 10^{-6}–10^{-15}. In other words, this method outperforms previous methods by orders of magnitude in terms of numerical accuracy of data points. Remarkably, these excellent results are obtained for examples that are qualitatively much more complicated than those usually reported in the literature.
- *Robustness against noise and GA parameters*: Since the method is based on approximating the data points rather than interpolating them, it is robust against noise in data points. To test this issue, we introduced small random perturbations on some samples extracted from data points. As expected, global results were very similar, meaning that the method is not globally affected by noisy data points provided that noise intensity is kept below a certain threshold. Also, our results show that the method is robust against variations of GA parameters such as crossover and mutation probability values within a prescribed range (see Section 6.5 for details).

The structure of this paper is as follows: firstly, previous literature regarding relevant surface reconstruction methods is reported in Section 2. Then, some basic concepts about B-spline surfaces and genetic algorithms are given in Sections 3 and 4, respectively. The core of the paper is in Section 5: it begins with a detailed explanation about the fitting surface problem. Then, main steps of our approach along with the numerical procedures used in the process are discussed in detail. Section 6 reports our experimental results. Some implementation issues, a detailed comparison with previous approaches and a careful analysis about the computation times and the influence of GA parameters on the method are also given in this section. The paper closes with the main conclusions and our future work.

2. Previous work

Surface reconstruction has received a lot of attention from the scientific community during the last 20 years, with several papers reporting outstanding applications in approximation theory [10], numerical analysis [14,59], geometric modeling [16,76], computer-aided geometric design (CAGD) [57], Computer-aided manufacturing (CAM) [56,60], data visualization [61], cultural heritage preservation [46], virtual reality [45] and many others.

Depending on the initial input available, surface reconstruction methods can be roughly classified into three groups. The first group addresses the problem by obtaining a surface model exclusively from a set of given cross-sections [2,15,38,40,52,54]. This is a classical problem in medical science, biomedical engineering and CAD/CAM in which a volumetric object (such as a body inner organ, for instance) is typically defined by a sequence of 2D cross-sections or thin layers (acquired from computer tomography, magnetic resonance imaging, ultrasound imaging, etc.). The second group makes use of iso-parametric curves on the surface [22] and even mixed information, such as scattered points and contours [50,70] or iso-parametric curves and data points [36,37].

In most cases, however, available information about the surface is typically a dense set of (either organized or scattered) 3D data points, leading to the third group of surface reconstruction methods. Such points are usually obtained by using scanner devices, the most common data point acquisition technique in reverse engineering applications (see, e.g., [25,26,30,47,49,71]). This is the case we focus on in this paper. In particular, our problem can be stated as follows: *given a set of sample points* $\mathbf{Q}$ *assumed to lie on an unknown surface* $\mathbf{U}$, *construct, to the extent possible, a B-spline representation of a surface model* $\mathbf{S}$ *that approximates* $\mathbf{U}$.

Depending on the nature of data points, two different approaches are employed: interpolation and approximation. In interpolation, the parametric surfaces are constrained to pass through all the given set of data points. This approach is generally suitable when the shape of data points describing the physical object is sufficiently smooth and accurate. In contrast, approximating surfaces should pass near (but not necessarily through) the given data points. Approximation techniques are specially recommended when data are not exact, but subjected to measurement errors. Another important reason to choose approximation could be the great computational effort required to obtain surfaces by interpolating an infinite number of

data, such as curve forms. An example is given in surface skinning problems, where we look for a smooth surface passing through a set of cross-sectional curves. In [58] it has been mentioned that, using the NURBS representation, interpolation of only a few cross-sections of various types may require as high a number as hundreds of thousands of surface control points. Furthermore, because industrial parts can easily contain hundreds of surfaces, interpolation of these parts becomes unrealizable in practice. Finally, in many applications, data consist of a very large number of measurements, causing the number of basis functions to be very large as well. In addition, new measurement points may change the structure of the solution.

The obvious solution to these problems is to consider an approximation scheme that allows us significant saving of time and memory. This is also the approach taken in this paper. This problem has been analyzed from several points of view, such as parametric methods [5,71] or implicit surfaces [47,62]. The interested reader is referred to [76] for a comprehensive introduction to the field and many interesting references. See also [80].

Some papers have shown that the application of artificial intelligence techniques can achieve remarkable results regarding this problem [25,28,41,43,36,37,73]. Most of these methods rely on some kind of neural networks, either standard neural networks [25,74], Kohonen's SOM (self-organizing maps) nets [3,28,43], or the Bernstein basis function (BBF) network [41]. In some cases, the network is used exclusively to order the data and create a grid of control vertices with quadrilateral topology [28]. After this pre-processing step, any other classical surface reconstruction method operating on organized points is subsequently applied. A work using a combination of neural networks and partial differential equation (PDE) techniques for the parameterization and reconstruction of surfaces from 3D scattered points can be found in [3]. Other sophisticated techniques include functional networks [37], a generalization of neural networks based on functional equations [6,7]. Finally, other techniques use evolutionary computation principles to solve the problem, such as genetic algorithms [16,42], particle swarm optimization [17], artificial inmune systems [73,75], simulated evolution [67,78] or dominant point solution [55]. However, these works restrict to the case of curves and little attention has been placed upon the case of surfaces.

Reconstruction of free-form objects consists typically of two main stages: (1) surface parameterization and (2) surface fitting. Two basic approaches have been considered to solve these stages [76,80]: the first one uses an iterative process in which the region is approximated by an initial surface which is then smoothed iteratively [31,49]. The initial surface, usually called *base surface*, is chosen so that it reflects the distribution of points in the 3D space. Potential base surfaces are planes, cylinders or Coons patches. Sample points are projected onto this base surface in order to extract boundary features as well as a coarse 2D parameterization of the surface. These approaches fail in case data points cannot be projected in an unambiguous way. An alternative is to consider a 3D triangulation and then derive a topologically equivalent 2D triangulation [11,13]. These methods are, however, slow and computational expensive for large numbers of data points and find difficulties when the point cloud contains concavities or holes. The second approach uses a variational scheme that seeks to minimize a hybrid functional accounting simultaneously for good approximation and smoothness [11,23]. They tend to fail in case of uneven distributions of points over the region of interest, and present the problem that additional coefficients (such as the smoothness weighted factor) must be determined. Another problem is the correct determination of the length of knot vectors, as it affects the surface quality and the computational efficiency of the process. In general, a trade-off must be achieved between increasing the number of knots to increase the degrees of freedom and to reduce such a number in order to alleviate the computation time. All these methods also tend to fail for surfaces of non-zero genus. Our proposed method overcomes such limitations, as we will show later on.

3. Basic definitions and notation

3.1. B-spline surfaces

Let $\mathcal{M} = \{\mu_0, \mu_1, \mu_2, \ldots, \mu_{M-1}, \mu_M\}$ be a nondecreasing sequence of real numbers called *knots*. $\mathcal{M}$ is called the *knot vector*. The *ith* B-spline basis function $N_{i,k}(u)$ of order k (or equivalently, degree $k-1$) is defined by the recurrence relations

$$N_{i,1}(u) = \begin{cases} 1 & \text{if } \mu_i \leqslant u < \mu_{i+1}, \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

with $i = 0,\ldots,M-1$ and

$$N_{i,k}(u) = \frac{u-\mu_i}{\mu_{i+k-1}-\mu_i} N_{i,k-1}(u) + \frac{\mu_{i+k}-u}{\mu_{i+k}-\mu_{i+1}} N_{i+1,k-1}(u) \tag{2}$$

for $k > 1$. Note that ith B-spline basis function of order 1, $N_{i,1}(u)$, is a piecewise constant function with value 1 on the interval $[\mu_i, \mu_{i+1})$, called the *support* of $N_{i,1}(u)$, and zero elsewhere. This support can be either an interval or reduce to a point, as knots μ_i and μ_{i+1} must not necessarily be different. If necessary, the convention $\frac{0}{0} = 0$ in Eq. (2) is applied. Any basis function of order $k > 1$, $N_{i,k}(u)$, is a linear combination of two consecutive functions of order $k-1$, where the coefficients are linear polinomials in u, such that its order (and hence its degree) increases by 1. Simultaneously, its support is the union of the (partially overlapping) supports of the former basis functions of order $k-1$ and, consequently, it usually enlarges.

The number of times a knot appears in the knot vector is called the *multiplicity* of the knot and has an important effect on the shape and properties of the associated basis functions. Basically, knot vectors can be classified into two groups. The first

one is the *uniform knot vector*: each knot appears only once and the distance between consecutive knots is always the same. As a consequence, each basis function $N_{i,k}(u)$ is plotted similarly to the previous one, $N_{i-1,k}(u)$, but shifted to the right according to such a distance. A qualitatively different behavior is obtained when any of the knots appears more than once or knots are not equally spaced (such cases are usually referred to as *non-uniform knot vector*). The most common case of non-uniform knot vectors consists of repeating the end knots as many times as the order while interior knots appear only once (such a knot vector is called *non-periodic knot vector*).

With the same notation, given a set of three-dimensional points (called *control points* as they roughly determine the shape of the surface) $\{\mathbf{P}_{ij}\}_{i=0,\ldots,m;j=0,\ldots,n}$ (note that in this paper vectors are denoted in bold) in a bidirectional net and two knot vectors $\mathcal{M} = \{\mu_0, \mu_1, \ldots, \mu_{M-1}, \mu_M\}$ and $\mathcal{N} = \{v_0, v_1, \ldots, v_{N-1}, v_N\}$, a *B-spline surface* $\mathbf{S}(u, v)$ *of order* (k, l) is a parametric surface given by:

$$\mathbf{S}(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{P}_{i,j} N_{i,k}(u) N_{j,l}(v), \tag{3}$$

where the $\{N_{i,k}(u)\}_{i=0,\ldots,m}$ and $\{N_{j,l}(v)\}_{j=0,\ldots,n}$ are the B-spline basis functions of order k and l, respectively, defined following Eqs. (1) and (2). Without loss of generality parameters u, v can be assumed to take values on the interval $[0,1]$ [57]. For a proper definition of a B-spline surface in Eq. (3), the following relationships must hold (see [57] for further explanation): $M = m + k$, $N = n + l$. In general, a B-spline surface does not interpolate any of its control points; interpolation only occurs for non-periodic knot vectors (in that case, the B-spline surface does interpolate the corner control points). Since they are the most common in computer graphics and industrial domains, in this work we will restrict ourselves to non-periodic knot vectors. Note however that our method does not preclude any other kind of knot vectors to be used at all.

3.2. Some vector definitions

The *vectorization* of a matrix $\mathbf{C}$, denoted by $\mathbf{vec}(\mathbf{C})$, is a linear transformation which converts the matrix into a column vector by stacking its columns on top of one another. For instance, for:

$$\mathbf{C} = \begin{pmatrix} c_{1,1} & \cdots & c_{1,n} \\ c_{2,1} & \cdots & c_{2,n} \\ \vdots & \ddots & \vdots \\ c_{m,1} & \cdots & c_{m,n} \end{pmatrix}, \quad \mathbf{vec}(\mathbf{C}) = (c_{1,1}, \ldots, c_{m,1}, \ldots, c_{1,n}, \ldots, c_{m,n})^T.$$

Given two vectors $\mathbf{X} = (x_1, x_2, \ldots, x_m)$ and $\mathbf{Y} = (y_1, y_2, \ldots, y_n)$ of lengths m and n, respectively, its *outer product*, denoted by $\mathbf{X} \otimes \mathbf{Y}$, is a matrix of size $m \times n$ given by:

$$\mathbf{X} \otimes \mathbf{Y} = \begin{pmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{pmatrix}.$$

4. Genetic algorithms

Genetic algorithms (GA) are search procedures based on principles of evolution and natural selection. They can be used in optimization problems where the search of optimal solutions is carried out in a space of solutions coded as finite-length strings called chromosomes. To do so, GA handle populations consisting of sets of potential solutions, i.e. the algorithm maintains a population of p individuals $\mathbf{Pop}(g) = \{x_1(g), \ldots, x_p(g)\}$ for each iteration g (also called generation), where each individual represents a potential solution of the problem. Normally the initial population is randomly selected, but some knowledge about the specific problem can be used to include in the initial population special potential solutions in order to improve the convergence speed. The size of this initial population is one of the most important aspects to be considered and may be critical in many applications. If the size is too small, the algorithm may converge too quickly, and if it is too large the algorithm may waste computational resources. The population size is usually chosen to be constant although GA with varying population size are also possible. A study about the optimal population size can be found in [19]. Each individual in the population, i.e. each potential solution, must be represented using a genetic representation. Commonly, a binary representation is used, however other approaches are possible. Each one of the potential solutions must be evaluated by means of a fitness function; the result of this evaluation is a measure of individual adaptation.

The algorithm is an iterative process in which new populations are obtained using a selection process (reproduction) based on individual adaptation and some "genetic" operators (crossover and mutation). The individuals with the best adaptation measure have more chance of reproducing and generating new individuals by crossing and muting. The reproduction operator can be implemented in several ways, such as tournament, roulette wheel, rank-based, hall of fame and others (see

Table 1
General structure of the genetic algorithm.

```
begin
  Let g = 0 be the generation counter
  Create and initialize a population, Pop(0)
  repeat
    Evaluate the fitness, f(x_i(g)), of each individual x_i of Pop(g)
    Select individuals from Pop(g)
    Apply crossover with probability p_c to produce offspring
    Apply mutation with probability p_m on offspring
    Set population of new generation Pop(g + 1)
    Advance to the new generation g = g + 1
    until stopping condition is true
end
```

[20,53]). The selection process is repeated d times and the selected individuals form a tentative new population for further genetic operator actions.

After reproduction some of the members of the new tentative population undergo transformations. A *crossover* operator creates two new individuals (*offsprings*) by combining parts from two randomly selected individuals of the population. In GA the crossover operator is randomly applied with a specific probability, p_c. A good GA performance requires the choice of a high crossover probability. *Mutation* is a unitary transformation which creates, with certain probability, p_m, a new individual by a small change in a single individual. In this case, a good algorithm performance requires the choice of a low mutation probability (inversely proportional to the population size). The mutation operator guarantees that all the search space has a nonzero probability of being explored. Using these genetic operators, the general structure of the algorithm is described in Table 1.

This procedure is repeated several times (thus yielding successive generations) until a termination condition has been reached. Common terminating criteria are that a solution is found that satisfies a lower threshold value, or that a fixed number of generations has been reached, or that successive iterations no longer produce better results.

5. Our proposal

5.1. Surface fitting problem

Let us suppose that we are provided with a set of three-dimensional data points arranged in a matrix as: $\mathbf{Q} = \{\mathbf{Q}_{r,s} = (x_{r,s}, y_{r,s}, z_{r,s})\}_{r=0,\ldots,R;\, s=0,\ldots,S}$. We want now to fit such points with a B-spline surface $\mathbf{S}(u, v)$ of order (k, l) defined as above (with $M < R$, $N < S$) such that:

$$\mathbf{Q}_{r,s} = \sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{P}_{i,j}N_{i,k}(u_r)N_{j,l}(v_s) \quad \forall r = 0,\ldots,R;\ s = 0,\ldots,S. \tag{4}$$

To do so, we must first make an association (parameterization) of parametric values (u_r, v_s) to each of our data points $\mathbf{Q}_{r,s}$. Given this association, surface fitting can be formulated from (3) and (4) as finding the surface which minimizes the following least-squares expression:

$$E_{lsq} = \sum_{r=0}^{R}\sum_{s=0}^{S}\left(\mathbf{Q}_{r,s} - \mathbf{S}(u_r, v_s)\right)^2 = \sum_{r=0}^{R}\sum_{s=0}^{S}\left(\mathbf{Q}_{r,s} - \sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{P}_{i,j}N_{i,k}(u_r)N_{j,l}(v_s)\right)^2. \tag{5}$$

The case of unorganized points can be handled in a quite similar way by replacing matrix $\mathbf{Q}$ by a vector. In other words, data points are now arranged along a vector $\{\mathbf{Q}_t = (x_t, y_t, z_t)\}_{t=0,\ldots,T}$, where T is the number of data points. In this case, we must associate suitable parametric values (u_t, v_t) to each of our data points $\mathbf{Q}_t$ so the surface fitting problem is expressed as:

$$E_{lsq} = \sum_{t=0}^{T}(\mathbf{Q}_t - \mathbf{S}(u_t, v_t))^2 = \sum_{t=0}^{T}\left(\mathbf{Q}_t - \sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{P}_t N_{i,k}(u_t)N_{j,l}(v_t)\right)^2. \tag{6}$$

As a result, an over-constrained system of linear equations can be generated from either (5) or (6) where control points $\mathbf{P}_{i,j}$ or $\mathbf{P}_t$ are the unknowns, respectively. Therefore, a least-squares solution can be performed in order to compute such control points. The problem is that in real applications, we are not given any parameterization of data points. In fact, even the number of control points and the number of knots are also unknown. Because the B-spline basis functions are nonlinear functions, the problem requires to solve a nonlinear optimization process, with a high number of unknowns for large sets of data points, a case that happens very often in practice.

As a conclusion, obtaining a feasible solution of the B-spline surface fitting problem requires to perform next steps:

(1) Firstly, a careful choice of suitable parameters is needed in order to construct the B-spline basis functions in Eq. (3). Such parameters include the order of B-spline basis functions and the number of control points and knots. In practical applications these parameters are usually chosen by end-users and so are in our method.
(2) Secondly, a proper parameterization of data points must be achieved.
(3) We should then obtain surface knot vectors.
(4) Next, coefficients of the least-squares problem are to be determined.
(5) Lastly, reconstructed data points are generated and compared with the original ones. Previous steps are repeated if error is larger than a prescribed threshold.

All these sub-problems are solved in next section by applying a genetic algorithm approach in order to determine all relevant parameters.

5.2. The proposed method

The proposed method has been briefly sketched in Section 1.3. In this section, we describe all steps in detail. The initial input of our problem is given by:

- The collection of 3D data points, $\mathbf{Q}$;
- The order of the B-spline surface (k,l);
- The number of control points, (m,n), and
- Possibly, a threshold tolerance error, ϵ.

5.2.1. Step 1: Data points parameterization

First step consists of obtaining an adequate parameterization of data points. We perform this task by using a genetic algorithm. In order to do so, some aspects must be previously considered. First of all, a typical GA requires two elements to be defined prior to its use: (1) a genetic representation of each potential solution of the problem and (2) a measure of the quality of the solution (the fitness function). Since in this step we are interested in the assignment process of parameter values to data points, we propose the use of a real-coded genetic algorithm in which the genetic representation (chromosome) of an individual will be a real σ-dimensional vector, where each coordinate represents the parameter value assigned to a data point. Although we have $(R+1)\times(S+1)$ data points, the tensor-product structure of B-spline surfaces implies that we only need two vectors of length $(R+1)$ and $(S+1)$, respectively to store the parametric values in u and v along the interval $[0,1]$. As initial population we consider a randomly generated set of parameter vectors. To widen the search area of the algorithm it is desirable that the population size be large; however the computation time increases as this parameter rises, so a trade-off between both considerations is actually required. In addition, some standard parameterizations can be included in order to speed up the convergence of the process [57], such as the chord length method:

$$\alpha_0 = 0, \quad \alpha_\Psi = 1, \quad \alpha_\xi = \alpha_{\xi-1} + \frac{|\mathbf{Q}_\xi - \mathbf{Q}_{\xi-1}|}{\sum_{l=1}^{\Psi}|\mathbf{Q}_l - \mathbf{Q}_{l-1}|}$$

or the centripetal method:

$$\alpha_0 = 0, \quad \alpha_\Psi = 1, \quad \alpha_\xi = \alpha_{\xi-1} + \frac{\sqrt{|\mathbf{Q}_\xi - \mathbf{Q}_{\xi-1}|}}{\sum_{l=1}^{\Psi}\sqrt{|\mathbf{Q}_l - \mathbf{Q}_{l-1}|}}$$

for $\xi = 1,\ldots,\Psi-1$, where $\alpha = u$ (resp. v) and $\Psi = R$ (resp. S).

Regarding the fitness function to measure the quality of our assignment, it is the error function of the fitting process, i.e. Eq. (5) or (6). Note that they require to specify an initial knot vector for each u and v. Initially, they are chosen randomly and then optimized in the second step of our process.

The algorithm then uses three genetic operators to obtain new populations of individuals: selection, crossover and mutation. In our case, the selection operator is implemented as the classical biased roulette wheel with slots weighted in proportion to individual fitness values. For crossover we use a one-point operator that randomly selects a crossover point within an individual, then swaps the two parent chromosomes to the right from this point and eventually sorts the obtained vectors to produce two new offsprings. This process is illustrated in Table 2. Finally, as a mutation method we propose to select the position φ with worst fit error in the parameter vector of the solution and change the value of the selected parameter by the arithmetic mean of the previous and next parameters in the vector, that is, $\alpha_\varphi = \frac{\alpha_{\varphi-1}+\alpha_{\varphi+1}}{2}$. Note that $\alpha_{\varphi-1} < \alpha_\varphi < \alpha_{\varphi+1}$, hence no sorting method is required.

5.2.2. Step 2: Knot vectors computation

In second step of our method we compute the knot vectors by using another genetic algorithm. In this case, individuals are encoded as two real vectors of lengths $M+1$ and $N+1$ valued onto the interval $[0,1]$, initialized with random values. In

Table 2
Crossover operator.

Parent 1									
0.123	0.178	0.274		**0.456**	0.571	0.701	0.789	0.843	0.921	0.950
				↑↓					
Parent 2									
0.086	0.167	0.197		**0.271**	0.367	0.521	0.679	0.781	0.812	0.912
				Cross point					
Offspring 1									
0.123	0.178	**0.274**		**0.271**	0.367	0.521	0.679	0.781	0.812	0.912
Chromosomes sorting									
Offspring 1									
0.123	0.178	**0.271**		**0.274**	0.367	0.521	0.679	0.781	0.812	0.912
Offspring 2									
0.086	0.167	0.197		**0.456**	0.571	0.701	0.789	0.843	0.921	0.950

this paper we consider nonperiodic knot vectors exclusively, so we have respectively, k and l end knots fixed as 0s and 1s and only $m-k+1$ and $n-l+1$ knots to be calculated. This initial population is enriched with the knot vectors calculated by the averaging method:

$$\beta_0 = \cdots = \beta_{\zeta-1} = 0, \quad \beta_{\Theta-(\zeta-1)} = \cdots = \beta_{\Theta} = 1$$
$$\beta_{j+\zeta-1} = \frac{1}{\zeta-1}\sum_{h=j}^{j+\zeta-2}\beta_h \quad (j = 1,\ldots,\Theta-\zeta+1),$$

where $\beta = \mu$ (resp. ν), $\zeta = k$ (resp. l) and $\Theta = M$ (resp. N). Both the fitness function and the genetic operators are similar to those in previous step. Genetic algorithms in steps 1 and 2 of our method also share some parameter values: probability of crossover $p_c = 0.9$ and probability of mutation $p_m = 0.2$, while population size changes from example to example, ranging from $p = 50$ to $p = 500$. In our experiments, the order of the fitting surface is chosen based on the visual complexity of data points; simple problems are assigned values between 3 and 8, and this value is increased for intricate shapes. However, B-splines are piecewise polynomial functions, so this choice is not critical at all. Finally, our termination criterion is that of not improving the solution after 10 consecutive generations.

5.2.3. Step 3: Control points calculation

Once parametric values of data points and knot vectors are obtained, the method computes the control points of the B-spline surface by using least-squares method. Before explaining the procedure in detail, let us firstly introduce the following notation:

$$\mathbf{N}(\tilde{u}) = (N_{0,k}(\tilde{u}), N_{1,k}(\tilde{u}), \ldots, N_{m,k}(\tilde{u})),$$
$$\mathbf{N}(\tilde{v}) = (N_{0,l}(\tilde{v}), N_{1,l}(\tilde{v}), \ldots, N_{n,l}(\tilde{v})),$$
$$\mathbf{N_{i,k}} = (N_{i,k}(u_0), N_{i,k}(u_1), \ldots, N_{i,k}(u_R)),$$
$$\mathbf{N_{j,l}} = (N_{j,l}(v_0), N_{j,l}(v_1), \ldots, N_{j,l}(v_S)),$$
$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_{0,0} & \mathbf{Q}_{0,1} & \cdots & \mathbf{Q}_{0,S} \\ \mathbf{Q}_{1,0} & \mathbf{Q}_{1,1} & \cdots & \mathbf{Q}_{1,S} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Q}_{R,0} & \mathbf{Q}_{R,1} & \cdots & \mathbf{Q}_{R,S} \end{pmatrix}; \quad \mathbf{P} = \begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} & \cdots & \mathbf{P}_{0,n} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} & \cdots & \mathbf{P}_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{m,0} & \mathbf{P}_{m,1} & \cdots & \mathbf{P}_{m,n} \end{pmatrix}.$$

Expression (4) can now be rewritten as:

$$\mathbf{Q}_{r,s} = \mathbf{N}(u_r)\cdot\mathbf{P}\cdot\mathbf{N}^T(v_s) \quad \forall r = 0,\ldots,R; \quad s = 0,\ldots,S, \tag{7}$$

where $(\cdot)^T$ means the transpose of the matrix. Extending Eq. (7) to all indices (r,s) and applying the vectorization operator to matrices $\mathbf{Q}$ and $\mathbf{P}$ leads to the general expression:

$$\mathbf{vec}(\mathbf{Q}) = \begin{pmatrix} \mathbf{N_{0,k}}\otimes\mathbf{N}(v_0) & \mathbf{N_{1,k}}\otimes\mathbf{N}(v_0) & \cdots & \mathbf{N_{m,k}}\otimes\mathbf{N}(v_0) \\ \mathbf{N_{0,k}}\otimes\mathbf{N}(v_1) & \mathbf{N_{1,k}}\otimes\mathbf{N}(v_1) & \cdots & \mathbf{N_{m,k}}\otimes\mathbf{N}(v_1) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{N_{0,k}}\otimes\mathbf{N}(v_S) & \mathbf{N_{1,k}}\otimes\mathbf{N}(v_S) & \cdots & \mathbf{N_{m,k}}\otimes\mathbf{N}(v_S) \end{pmatrix}\cdot\mathbf{vec}(\mathbf{P}), \tag{8}$$

where each element $\mathbf{N_{i,k}} \otimes \mathbf{N}(v_s)$ of the expanded matrix in the right-hand side of Eq. (8) (denoted as $\mathbf{M}$ henceforth) has size $(R+1) \times (n+1)$. Note that $\mathbf{vec}(\mathbf{Q})$ is a column vector of length $(R+1) \times (S+1)$ while $\mathbf{vec}(\mathbf{P})$ is a column vector of length $(m+1) \times (n+1)$ so system (8) is overdetermined. Pre-multiplication of both sides by $\mathbf{M}^T$ yields:

$$\mathbf{M}^T \cdot \mathbf{vec}(\mathbf{Q}) = \mathbf{M}^T \cdot \mathbf{M} \cdot \mathbf{vec}(\mathbf{P}). \tag{9}$$

By this way, the problem becomes a classical linear least-squares minimization. Computation of coefficients $\mathbf{P}_{i,j}$ will give the best fit in the discrete least-squares sense to the data points, thus actually reconstructing the fitting B-spline surface. To solve Eq. (9) a modification of two standard numerical routines for the least squares problem have been used in this paper: the LU decomposition method and a non-iterative least squares fitting method accomplished by the singular value decomposition (SVD) method [63]. Method LU, primarily chosen because of its theoretical simplicity, can be computed by several algorithms (Doolittle, Crout, etc.) at a cost of $O(2p^3/3)$ excluding pivoting. We also used the SVD method, mostly because of its robustness and because it provides the best numerical answer in the sense of least-squares for those cases in which the exact solution is not possible [63]. To this aim, matrix $\mathbf{M}$ in Eq. (8) is decomposed as the matrix product $\mathbf{M} = \mathbf{U} \cdot \mathbf{\Phi} \cdot \mathbf{V}^T$ where $\mathbf{U}$ is a column-orthogonal matrix, $\mathbf{\Phi}$ is a diagonal matrix with positive or zero elements ϕ_k called the singular values and $\mathbf{V}$ is a square orthogonal matrix. Similar decomposition can be performed on $\mathbf{M}^T \cdot \mathbf{M}$ in case Eq. (9) is alternatively used. Furthermore, since $\mathbf{M}^T \cdot \mathbf{M}$ is square, its inverse can readily be obtained as: $\mathbf{V} \cdot \left[diag\left(\frac{1}{\phi_k}\right)\right] \cdot \mathbf{U}^T$. SVD can be computed by a two-step procedure with a computational cost of $O(p \cdot q^2)$ for a problem of size $p \times q$. In general, SVD performs well and yields very accurate results, although special care is needed in case of very small (or even zero) singular values [63]. Because of this reason, in this paper we applied optimized *Matlab* routines (mostly based on LAPACK numerical procedures) that handle this issue automatically (see Section 6.2 for details).

5.2.4. Step 4: Data points reconstruction

In this step we collect all output of previous steps in order to perform surface reconstruction. Reconstructed data points, denoted as $\widehat{\mathbf{Q}}_{r,s}$ (organized points) or $\widehat{\mathbf{Q}}_t$ (scattered points), are then compared to original data points according to some error measure, such as the point-wise Euclidean distance, the mean squared error or the root mean squared error. In this paper we consider the maximum error value and the mean error value (**ME**), given by:

$$\mathbf{ME} = \begin{cases} \frac{\sum_{r=0}^{R}\sum_{s=0}^{S}|\mathbf{Q}_{r,s}-\widehat{\mathbf{Q}}_{r,s}|}{(R+1)(S+1)} & \text{(organized points)}, \\ \frac{\sum_{t=0}^{T}|\mathbf{Q}_t-\widehat{\mathbf{Q}}_t|}{T} & \text{(scattered points)}. \end{cases}$$

Steps 1–4 can be iteratively applied until the computed error does not longer improve. In this paper, the workflow shown in Fig. 1 is repeated until that the difference between the mean error of two consecutive iterations becomes smaller than a given tolerance threshold value η.

6. Experimental results

In this section we report our experimental results. Some implementation issues are also briefly outlined. Finally, a comparison with some previous approaches and the issues of computation times and choice of GA parameters are discussed.

6.1. Illustrative examples

We have tested our algorithm on several examples from different families of functions. In this paper we present six of them: a genuine free-form B-spline surface and five surfaces defined parametrically. Examples have been primarily chosen to reflect the diversity of situations our algorithm can be applied to, and hence they provide a potential benchmark for future experiments. Our catalogue includes open (Example 1), semi-closed (Examples 2 and 3) and closed surfaces (Examples 4–6). To the aim of checking the performance of our proposal, examples also contain pretty challenging features, such as intricate shapes (Examples 1, 2, 5 and 6), singularities (Examples 2, 3, 4 and 6), concavities (Examples 1, 2, 3, 5 and 6) and non-zero genus (Example 5). In all cases, we were able to yield a B-spline surface with extremely small fitting errors, showing the robustness, flexibility and excellent performance of our approach.

6.1.1. Example 1: A B-spline surface

As first example we consider a set of 14,400 data points from a (5,4)-order B-spline surface with 8×6 control points. Data points have been fitted to a B-spline surface with orders varying from 3 to 9 for both u and v and with a number of control points ranging from 5 to 20 in both directions. Fig. 2 shows the fitting B-spline surface of order (6,6) and 14×14 control points. In this case, mean errors for data points coordinates are 3.97×10^{-6}, 9.56×10^{-6} and 7.76×10^{-6}, respectively, showing that the reconstructed surface matches original surface very well. In fact, maximum error for data points coordinates is 2.41×10^{-5}, 4.82×10^{-5} and 4.35×10^{-5}, showing that no large discrepancies with mean errors occur. In other words, neither spurious results nor artifacts are generated for the fitting surface.

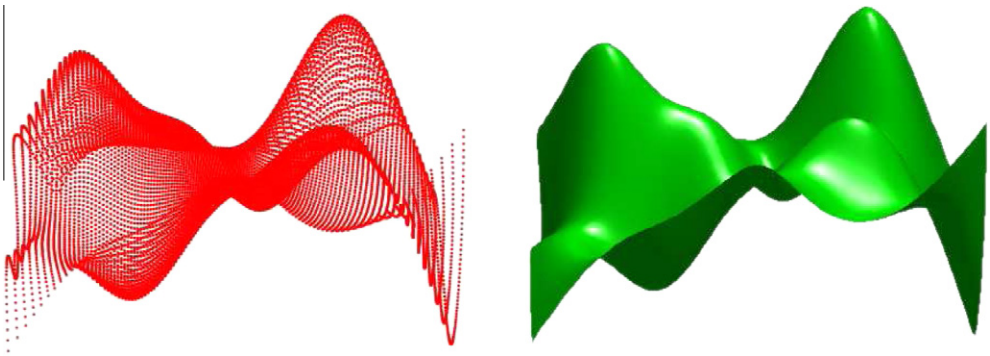

Fig. 2. Reconstruction of a (5,4)-order B-spline surface: (left) cloud of 3D data points; (right) (6,6)-order fitting B-spline surface.

In general, we could reconstruct the original surface very well although, depending on our choice of surface parameters (order and number of control points), fitting errors vary from 10^{-4} to 10^{-6}. For instance, for a (3,3)-order B-spline surface with 8×8 control points, mean errors were 9.7×10^{-4}, 4.2×10^{-4} and 2.6×10^{-4} for x, y and z, respectively. For the genetic algorithms of our method we used several initial population sizes ranging from 50 to 500 individuals, but reported errors were quite similar in all cases.

We also tested the robustness of our approach against noise, by perturbing some sampled data points by a real uniform random variable of mean 0 and variance $0.05 \times \delta$, where δ is a parameter accounting for surface size. Our trials showed that the method is robust against low-intensity noise, since our results are not quantitatively different than those obtained in absence of noise. We applied such perturbations to all other examples and obtained very similar results in all cases.

6.1.2. Example 2: Shell surface

Next example is a parametric surface known as *shell* and given by:

$$\begin{cases} x = \frac{1}{5}\left(1 - \frac{v}{2\pi}\right)\cos(2v)[1 + \cos(u)] + \frac{1}{10}\cos(2v), \\ y = \frac{1}{5}\left(1 - \frac{v}{2\pi}\right)\sin(2v)[1 + \cos(u)] + \frac{1}{10}\sin(2v), \quad u, v \in [0, 2\pi], \\ z = \frac{v}{2\pi} + \frac{1}{5}\left(1 - \frac{v}{2\pi}\right)\sin(u). \end{cases}$$

We applied our method on a given set of 7500 data points with a B-spline surface of order varying from 3 to 9. In our trials, error improved as the order increased up to 7, with no further improvement for larger values. Genetic algorithms in steps 1 and 2 of our approach have been applied on a population of 100 individuals. Fig. 3 shows the fitting surface obtained by our method for a (7,7)-order B-spline surface with a net of 20×20 control points. Mean errors in this example for data points coordinates are 8.84×10^{-7}, 7.34×10^{-7} and 1.47×10^{-9}, respectively. Although this surface does neither exhibit self-intersections nor holes, it still has a complicated, twisted and concave, shape. Furthermore, it exhibits a singularity at the end point and it is a closed surface in one parametric direction while it is open in the other one, so knots determination is

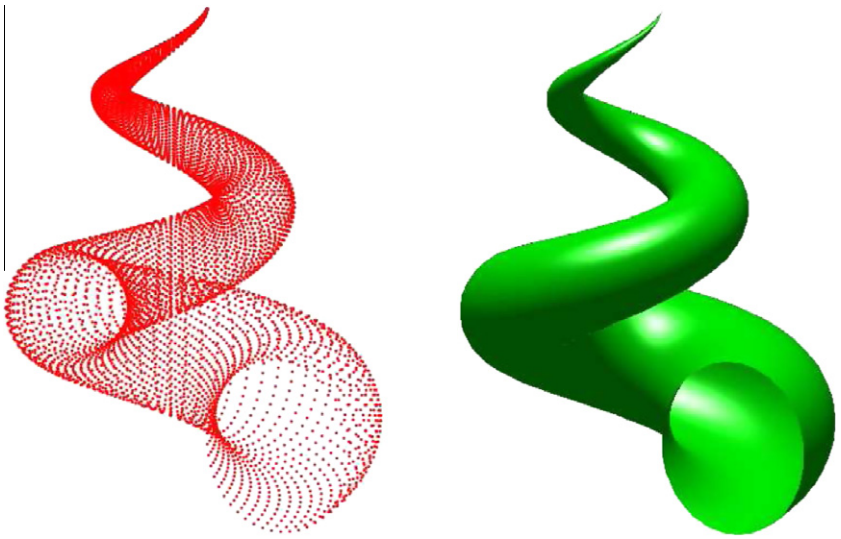

Fig. 3. Reconstruction of shell surface: (left) cloud of 3D data points; (right) (7,7)-order fitting B-spline surface.

not trivial. In fact, we remark that current surface reconstruction methods usually fail to reconstruct it accurately through polynomial B-spline surfaces.

Since our method is iterative, it might be helpful to display the evolution of the reconstructed surface from generation to generation. This is done in Fig. 4 for the shell surface. To keep the paper at reasonable length, only three generations are shown: $g = 3$ (left), $g = 7$ (middle) and $g = 11$ (right). For each, two different views - high-angle view (top) and cenital view (bottom) – are used for better visualization. This figure illustrates the convergence process from the initial surface to an extremely accurate fitting surface.

6.1.3. Example 3: Horn surface

In this example we consider a parametric surface called *horn* described mathematically by:

$$\begin{cases} x = [2 + u\cos(v)]\sin(2\pi u), \\ y = [2 + u\cos(v)]\cos(2\pi u) + 2u, \quad 0 \leqslant u \leqslant 1,\ 0 \leqslant v \leqslant \pi, \\ z = u\sin(v). \end{cases}$$

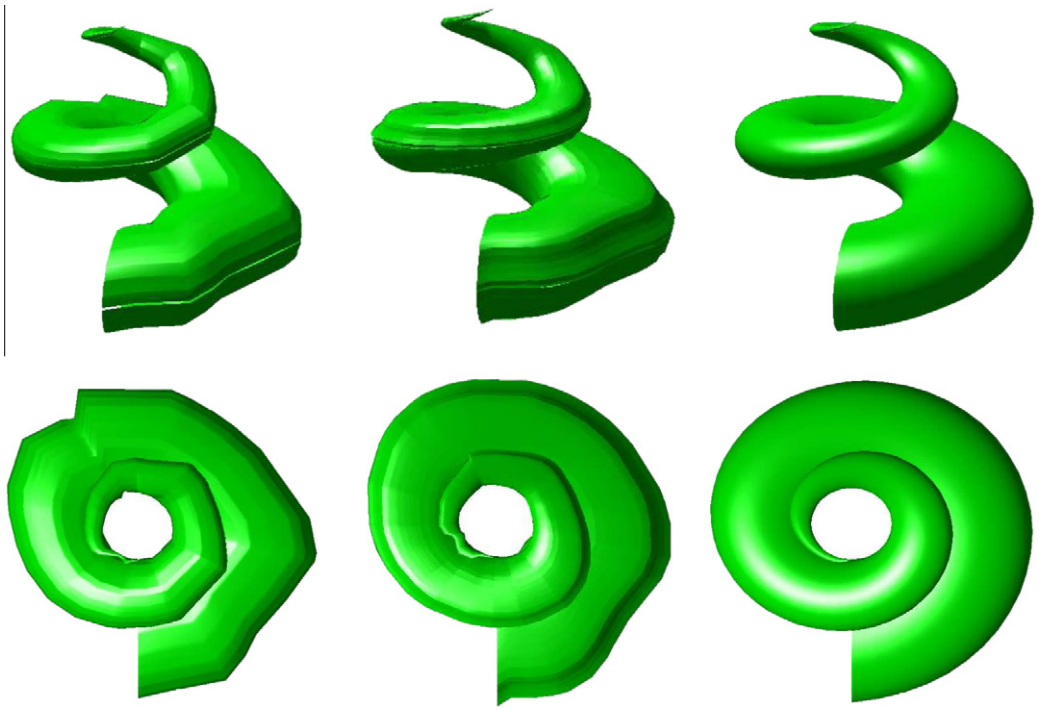

Fig. 4. High-angle view (top) and top view (bottom) of the evolution of the reconstructed shell surface for: (left) $g = 3$, (middle) $g = 7$ and (right) $g = 11$ generations.

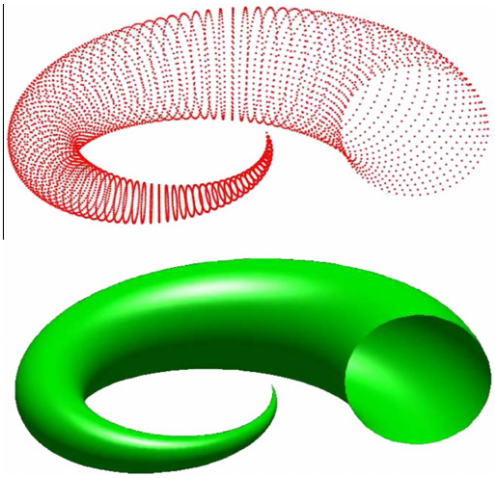

Fig. 5. Reconstruction of horn surface: (up) cloud of 3D data points; (down) (4,4)-order fitting B-spline surface.

Similarly to previous example, this surface is closed in one parametric direction and open in the other one, and also has a singularity point. Given a set of 6000 data points, the best fitting surface we got (depicted in Fig. 5) is a (4,4)-order B-spline surface with 12×12 control points and a population of 100 individuals. Mean error for data points coordinates are respectively, 3.24×10^{-6}, 3.04×10^{-6} and 7.84×10^{-7}, while maximum errors in data points are 1.78×10^{-5}, 2.16×10^{-5} and 9.63×10^{-6}. Results do not improve significantly when increasing the order and/or the number of control points from those in Fig. 5. Good errors we obtained reflect the ability of our method to reconstruct complicated surfaces through a strictly polynomial low-order B-spline surface with a limited amount of control points. To our knowledge, no previous method has achieved this goal, as they usually rely on NURBS surfaces which are very well suited to fitting quadrics and closed surfaces, whilst in this work we fit shapes by using polynomial surfaces exclusively.

6.1.4. Example 4: Teardrop surface

This surface, called *teardrop* because of its droplet shape, is an example of a closed surface with a singular point. It is expressed parametrically by:

$$\begin{cases} x = \frac{1}{2}[1-\cos(u)]\sin(u)\cos(v), \\ y = \frac{1}{2}[1-\cos(u)]\sin(u)\sin(v), \quad 0 \leqslant u \leqslant \pi,\ 0 \leqslant v \leqslant 2\pi, \\ z = \cos(u). \end{cases}$$

For a set of 6000 data points, the best fitting surface in this case (shown in Fig. 6) is a (3,3)-order B-spline surface with 16×16 control points for a population of 200 individuals, with mean errors 1.23×10^{-7}, 1.21×10^{-7} and 3.77×10^{-8} for x, y and z, respectively. Once again, this surface has been quite challenging since only polynomial basis functions are allowed in our approach.

6.1.5. Example 5: Pisot triaxial surface

In this example, we consider a complicated non-zero genus surface known as *Pisot triaxial*, given in parametric form by:

$$\begin{cases} x = A\cos(B+u)(2+\cos(v)), \\ y = C\cos(D-u)(2+E\cos(F+v)), \quad 0 \leqslant u, v \leqslant 2\pi, \\ z = E\cos(F+u)(2+G\cos(H-v)), \end{cases}$$

In this paper the following parameter values are used: $A = 0.655866$, $B = 1.03002$, $C = 0.754878$, $D = 1.40772$, $E = 0.868837$, $F = 2.43773$, $G = 0.495098$ and $H = 0.377696$. We applied our method to a set of 10,500 data points from this surface and obtained extremely accurate results. Best fitting surface for this example was a (20,20)-order B-spline surface (displayed in Fig. 7) with a net of 20×20 control points. Our trials show that decreasing B-spline order or number of control points yields worse (although still very accurate) fitting errors. In surface of Fig. 7, mean errors for the data points coordinates are 2.90×10^{-15}, 3.92×10^{-15} and 4.85×10^{-15}, respectively. This example shows that the method performs extremely well even for non-zero genus surfaces, which are traditionally troublesome for many surface reconstruction methods.

6.1.6. Example 6: Cresent surface

Last example corresponds to a surface called *Cresent* defined parametrically by:

$$\begin{cases} x = [2+\sin(2\pi u)\sin(2\pi v)]\sin(3\pi v) \\ y = [2+\sin(2\pi u)\sin(2\pi v)]\cos(3\pi v) \\ z = \cos(2\pi u)\sin(2\pi v) + 4v - 2 \end{cases}$$

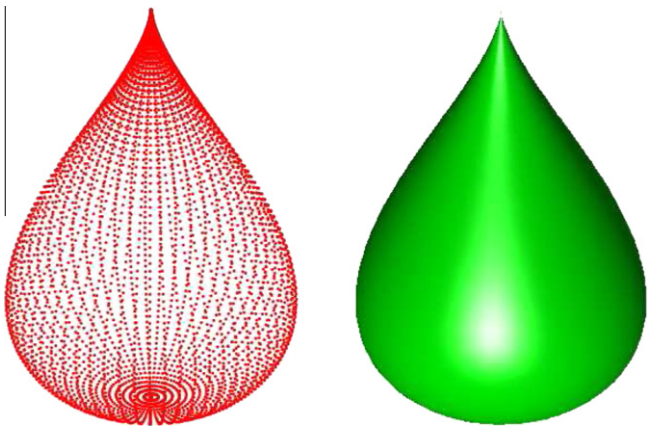

Fig. 6. Reconstruction of teardrop surface: (left) cloud of 3D data points; (right) (3,3)-order fitting B-spline surface.

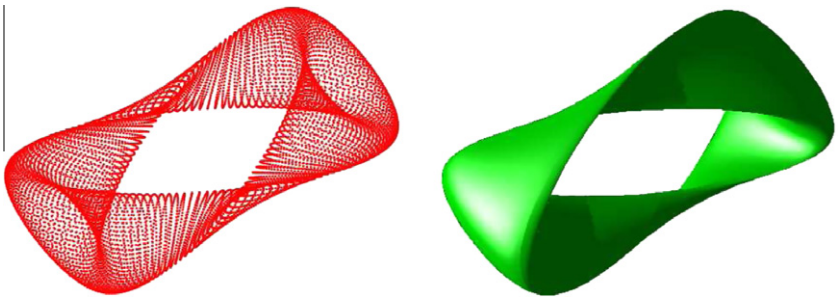

Fig. 7. Reconstruction of Pisot triaxial surface: (left) cloud of 3D data points; (right) (20,20)-order fitting B-spline surface.

with $0 \leqslant u, v \leqslant 1$. In this case, the best fitting surface (shown in Fig. 8) to a set of 14,200 data points is a (8,8)-order B-spline surface with a net of 30×30 control points. Mean error for data points coordinates is 1.01×10^{-10}, 1.07×10^{-10} and 5.32×10^{-11}, respectively. We remark the excellent results obtained by the method in this example and previous one; to the best of our knowledge, they improve by far those found in the literature on the subject.

6.2. Implementation issues

All computations in this paper have been performed on a 2.4 GHz. Intel Core 2 Duo processor with 4 GB of RAM. The source code has been implemented by the authors in the native programming language of the popular scientific program *Matlab*, version 2009a. In our opinion, *Matlab* is a very suitable tool for this task: it is fast and provides reliable, well-tested routines for efficient matrix manipulations. It also contains a bulk of resources regarding the solving of systems of equations. This feature proved to be very valuable in case of ill-conditioned matrices, i.e., with too large (or even infinite) condition number. This is a situation that can actually happen in practice, for instance, when one or several singular values in SVD decomposition are null or very near to zero. Advisable answer to this problem is to set reciprocals of such singular values to zero. *Matlab* command `svd` handles this situation for us. Similarly, when LU decomposition is used instead, *Matlab* command `lu` returns a suitable matrix factorization regardless the sparsity of the matrix, although different (mostly LAPACK and UMFPACK) routines are invoked in each case. Also, *Matlab* provides us with the command `mldivide` to solve the equation $\mathbf{A} \cdot \mathbf{X} = \mathbf{B}$ for both squared and non-squared systems (by using Gaussian elimination with partial pivoting and least-squares techniques, respectively). Depending on the general structure of matrix **A**, this command applies specialized LAPACK and BLAS routines to get the best possible solution to this system. Besides, *Matlab* provides excellent graphical options and optimized code for input/output interaction and high performance computations.

6.3. Comparison with other approaches

As shown earlier, our GA-based surface reconstruction method performs very well for the problems analyzed above and others not included here to keep the paper at reasonable length. Compared with other surface reconstruction approaches found in the literature, our method outperforms them in terms of accuracy and flexibility. To support this claim, a careful comparison with recent alternative surface reconstruction methods that use smooth piecewise surfaces and are based on

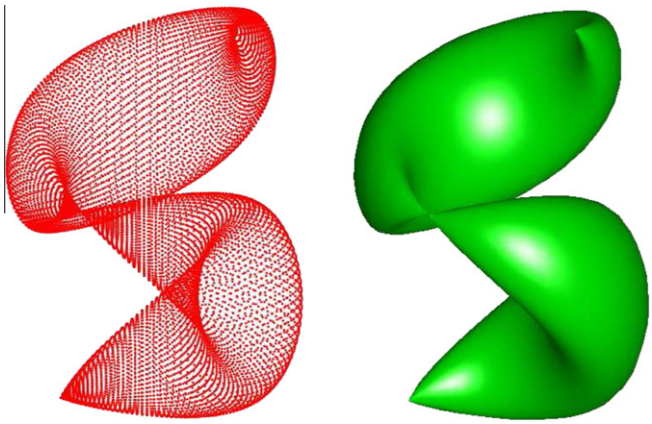

Fig. 8. Reconstruction of cresent surface: (left) cloud of 3D data points; (right) (8,8)-order fitting B-spline surface.

evolutionary techniques has been carried out. This gives four methods [68,77,79,81]. Another powerful method for piecewise surface reconstruction but not based on evolutionary techniques has been included to enrich the discussion [3]. Methods considering polygonal meshes fall out of the scope of this paper and therefore should not be considered for comparison. However, for illustrative purposes we decided to include a recent powerful method for polygonal meshes [18].

Table 4 summarizes our main results. Compared methods are arranged in rows and sorted so that methods not based on evolutionary algorithms appear first [3]; then, evolutionary-based methods for polygonal meshes [18] and piecewise surfaces [68,77,81,79] follow; finally, our method is reported. For each method, the following issues are discussed: technique employed in the method, number of data points in examples reported in the corresponding entries, population size (when applicable), output of the method and number of parameters involved, runtime, reported error, behavior of the method with respect to our examples 1–6 and some additional comments.

First method compared is that by Barhak and Fisher [3] where two different schemes, SOM (self-organizing map) neural network and PDEs (partial differential equations), are used to cope with the surface parameterization issue, and GDA (gradient descent algorithm) is used to create a 3D base surface that is iteratively modified to get the reconstructed surface. Such a surface is described in terms of cubic polynomial B-splines with errors of about 10^{-1}–10^{-2} for sets of about 10^4 data points. However, this method requires data points to be projected onto the base surface and therefore, only examples of neither self-intersecting nor high-genus surfaces are properly handled. In particular, the method fails for our examples 2–6 because data points cannot unambiguously projected.

In next method [18] evolutionary algorithms are used to recover the shape of tessellated surfaces such as a sphere, a fractal surface and a head (the Igea model). For a population size of 30–50 particles, a polygonal mesh is obtained. The method is able to reconstruct examples 1–4, but fails for examples 5 and 6, where a proper triangulation cannot be obtained. Moreover, since the surface is only linearly approximated, points on the reconstructed surface different from data points exhibit very large errors, several orders of magnitude larger than ours. However, its most important limitation does not concern the errors but the fact that it does not return an analytical surface, just merely an approximating polygonal mesh.

The method by Weinert et al. [79] combines NURBS with Constructive Solid Geometry in a hybrid evolutionary algorithm/genetic programming approach. In practice, however, the method is much simpler and less powerful than ours since no parameterization of data points is actually computed. Instead, it is assumed that the NURBS patch represents a surface of a solid which is infinite in the negative direction of the z-axis. This constraint, designed to avoid the parameterization process, has proved to be very limiting: examples 2–6 of this paper cannot be reconstructed with this method. In fact, the method is limited to very simple examples in which a planar base surface can be used (thus preventing closed or semi-closed surfaces to be used). And even in this case, reported errors for data points are quite large, of order 10^{-2} for an almost trivial example.

A very recent approach is given in [77], where a multi-objective evolutionary algorithm (MOEA) approach is applied to reconstruct a simple smooth surface with different sets of data points: the initial set of 17,307 data points was reduced to a set of 823 data points in order to decrease the runtime from several days to 9 h for errors of magnitude 10^{-2}. The method is also limited since it is assumed that sampled points are uniformly spaced in the (u, v) domain, so clouds of scattered data points cannot be reconstructed. Finally, data points should also be projected in order to perform the evaluation within the MOEA. As a consequence, the method is restricted to rather simple examples. In fact, the only reported example is by far much simpler than ours. Also, examples 2–6 of this paper cannot be reconstructed with this method.

Next method by Wen [81] applies simulated annealing for optimizing NURBS ship hull fitting. This work performs surface reconstruction from a set of cross-sections (called NURBS skinning) rather than from clouds of points. Given a set of data points, they are used to generate a set of cross-sections for surface fitting by making all those cross-sections compatible and joining them with one another. Then, surface optimization is accomplished on either Y or Z coordinate using simulated annealing (SA). In other words, SA is only applied to achieve surface optimization (not surface reconstruction), while non-evolutionary techniques are used for data points parameterization and surface fitting. Errors with this method are quite large, and the method is limited to very specific surfaces for which suitable cross-sections can readily be obtained. Once again, examples 2–6 of this paper cannot be reconstructed with this method.

In [68] an evolutionary heuristic technique known as simulated evolution (SimE) is applied to curve and surface fitting problems using NURBS. This method is very limited in nature, since instead of computing the knot vectors and the parameterization of data points, their values are assumed *a priori*. Similarly, it is assumed that the number of control points is equal to the order of the NURBS, then control points are calculated by least-squares method directly. In fact, all relevant NURBS parameters but the weights are either assumed or computed in standard way and SimE is only applied to determine such weights. Furthermore, curves and surfaces are assumed not self-intersecting. Compared with our approach, reported errors are very large and the reconstructed surface is of low quality. Moreover, this method fails for non-height-map surfaces so examples 2–6 of this paper cannot be reconstructed.

It is worthwhile to mention that in this paper we use strictly polynomial B-spline surfaces as fitting surfaces, while several approaches make use of NURBS surfaces, which are more flexible and versatile, and therefore they should yield smaller fitting errors. Even in this case, none of previous methods reported fitting errors as low as those achieved in this work. Besides, some examples analyzed here are, by far, much more complex and challenging that those usually found in previous approaches. Our method is conceptually simple, and can be implemented in a straightforward manner. It is also very versatile and exhibits a remarkable flexibility, being able to adapt to a wide range of situations. On the contrary, previous methods usually focus on specific surface types, derived from particular (although interesting) problems. In addition, previous methods discussed

above introduce some assumptions to skip the calculation of some relevant parameters, mostly knot vectors and parametric values of data points. This is the reason why none of previous methods was able to reconstruct our examples 2–6 with B-spline surfaces.

6.4. Computation times

Another important issue concerns the computation time. For this work we carried out several thousands of executions of our method on a number of different examples and got some interesting conclusions. The most important one is that the reported method is not as very-time consuming as it might appear at first sight. A crude version of the method can be computed in "reasonable" time for many applications, although our approach is not well suited for real-time applications; the emphasis is on accuracy. A typical run may take from a few seconds to several minutes (with the most complicated examples tested so far demanding about 2–3 h). CPU times depend on many factors such as surface complexity, number of initial data points, initial population, numerical procedure employed, required accuracy, and GA parameters. Therefore, it is hard to obtain a general rule for determining how long does it take for a given example to be solved.

However, some hints can still be given. Our trials show that the most important factors are the number of initial data points, the required accuracy and the initial population for the GA (in general, the larger, the longer the CPU time), while surface parameters have smaller influence on the overall computation times. Table 3 shows the CPU times and order of accuracy we obtained for the examples in this paper. The role of GA parameters in the computation times is analyzed in next section.

On the other hand, the crude version of our method can be improved for better performance. A feasible way to do it is to include any extra information you can know from your problem, as we actually do in our paper. For instance, in step 1 of our approach, GA are applied to compute a parameterization of data points. Initial population for such GA is randomly generated, but some standard parameterizations have been added to speed up the convergence of the process. A similar idea is applied in step 2 to the determination of knot vectors (in this case, by using the averaging method, as explained in Section 5.2.2). This strategy paid off: even although these individuals were not usually the best, computation times in our experiments were significantly reduced. Another key factor to reduce computation times concerns the code optimization. Using Matlab and its core of highly-optimized routines has reduced CPU times significantly with respect to a coarse, brute-force implementation.

We also noticed that our approach is much faster than previous methods even although our examples are much more complicated. The paper in [3] reported times of about 200–350 min, while in [12] the fitting of tens of thousands of points took hundreds of minutes on a HP 735 workstation. However, this comparison is not totally fair, since those papers were written time ago and hardware and software have improved dramatically since then. A more pertinent comparison of runtimes with recent entries [18,68,77,79] is done in Table 4. The method in [18] reported runtimes from minutes to hours even although the output is a polygonal mesh, while times reported in [68,81] are not comparable, since they correspond to the optimization process only, just a small step of the whole procedure. A more adequate comparison can be done with [77,79]; they reported times of one to several days for sets of data points comparable to our simplest examples (that take only from tens of seconds to 2–3 min with our approach), while our most complicated examples take about 2–3 h. It must also be noticed that all those methods reported much worse fitting errors than our method. This observation is important, since most of previous methods include an optimization stage in addition to the fitting [3,44,69]. Not by chance, such optimization stage is typically the time bottleneck [3] meaning that fitting error enhancement with those methods may demand significantly longer even for modest improvement factors.

6.5. Analysis of GA parameters

In this section we discuss the influence of GA parameters on the overall performance of the method and the computation time. In general, a good GA performance requires the choice of a high crossover probability p_c and a low mutation probability p_m. Accordingly, we applied our method to examples 1–6 of our paper for different values of the probabilities of crossover p_c from 0.8 to 1.0 and mutation p_m from 0 to 0.2 and step-size 0.02 for both parameters. This generates a matrix of 11×11 couples (p_c, p_m). For each, we carried out 10 executions and computed the mean value of the results. Such results are depicted in Fig. 9 (from top to bottom) for the horn, shell and cresent surfaces, where p_c and p_m are represented in the horizontal and

Table 3
Computation times and accuracy for the examples reported in this paper.

Example	CPU time	Accuracy
B-spline surface	8–15 s	10^{-6}
Shell surface	11–19 min	10^{-7}–10^{-8}
Horn surface	1–2 min	10^{-7}
Teardrop surface	20–65 s	10^{-7}
Pisot triaxial surface	2–3 h	10^{-15}
Cresent surface	14–20 min	10^{-11}

Table 4
Comparison of the proposed method with other alternative surface reconstruction methods.

Authors	Method	# Data points	Population	Output	Runtime	Error	Examples 1–6	Additional comments
Not evolutionary approach								
Barak and Fisher [3]	Neural network SOM and PDE with GDA (gradient descent algorithm)	10^4	Not applicable here	Cubic polynomial B-spline	3–6 h	Data points: 10^{-1}–10^{-2}	Only ex. 1 works properly	Requires projection of data points onto a base surface
							Ex. 2–6 fail	
Evolutionary approach								
Goinski [18]	Evolutionary algorithms (EA)	Not reported	30 (sphere) 30 (fractal) 50 (head)	Polygonal mesh (triangulation)	Tens of min. ∼ hours	Not reported for data points (in general, high)	Ex. 1–4 reconstructed Ex. 5 and 6 fail	Not actual surface is reconstructed (it is a polygonal mesh)
Weinert et al. [79]	Evolutionary search (ES) and Genetic programming (GP)	192	500	Polynomial B-spline	24 h	Data points: 10^{-2}	Only ex. 1 works properly	No parameterization is computed
							Ex. 2–6 fail	Pre-processing required
Wagner et al. [77]	Multi-objective evolutionary and genetic algorithm (MOEA)	823–17,307	20	Polynomial B-spline	Tens of hours ∼ days	Data points: 10^{-2}	Only ex. 1 works properly	No parameterization is computed
							Ex. 2–6 fail	Pre-processing required
Wen [81]	Simulated annealing (SA)	Not reported (cross-sections)	Not applicable here	NURBS skinning	Not reported	Data points: 10^{-1}–10^{-2}	Only ex. 1 works properly	SA only applied to Y or Z coordinate optimization
							Ex. 2–6 fail	Pre-/post-processing required
Sarfraz [68]	Simulated evolution algorithm (SimE)	441–1024	Not applicable here	Polynomial B-spline	Minutes (just weights optimization)	Data points: 10^{-1}	Only ex. 1 works properly	Neither parameterization nor fitting is computed
							Ex. 2–6 fail	Pre-/post-processing required
Our method	Iterative two-steps genetic algorithm with SVD/LU-mod LSQ fitting	14,400 (ex. 1) 7500 (ex. 2) 6000 (ex. 3) 6000 (ex. 4) 10,500 (ex. 5) 14,200 (ex. 6)	50–500	Polynomial B-spline	Tens of secs. ∼ 2–3 h	Data points: 10^{-6}–10^{-15} (worst cases: 10^{-4})	All work properly	Automatic data points parameterization and knot vector determination
								No pre-/post-processing required

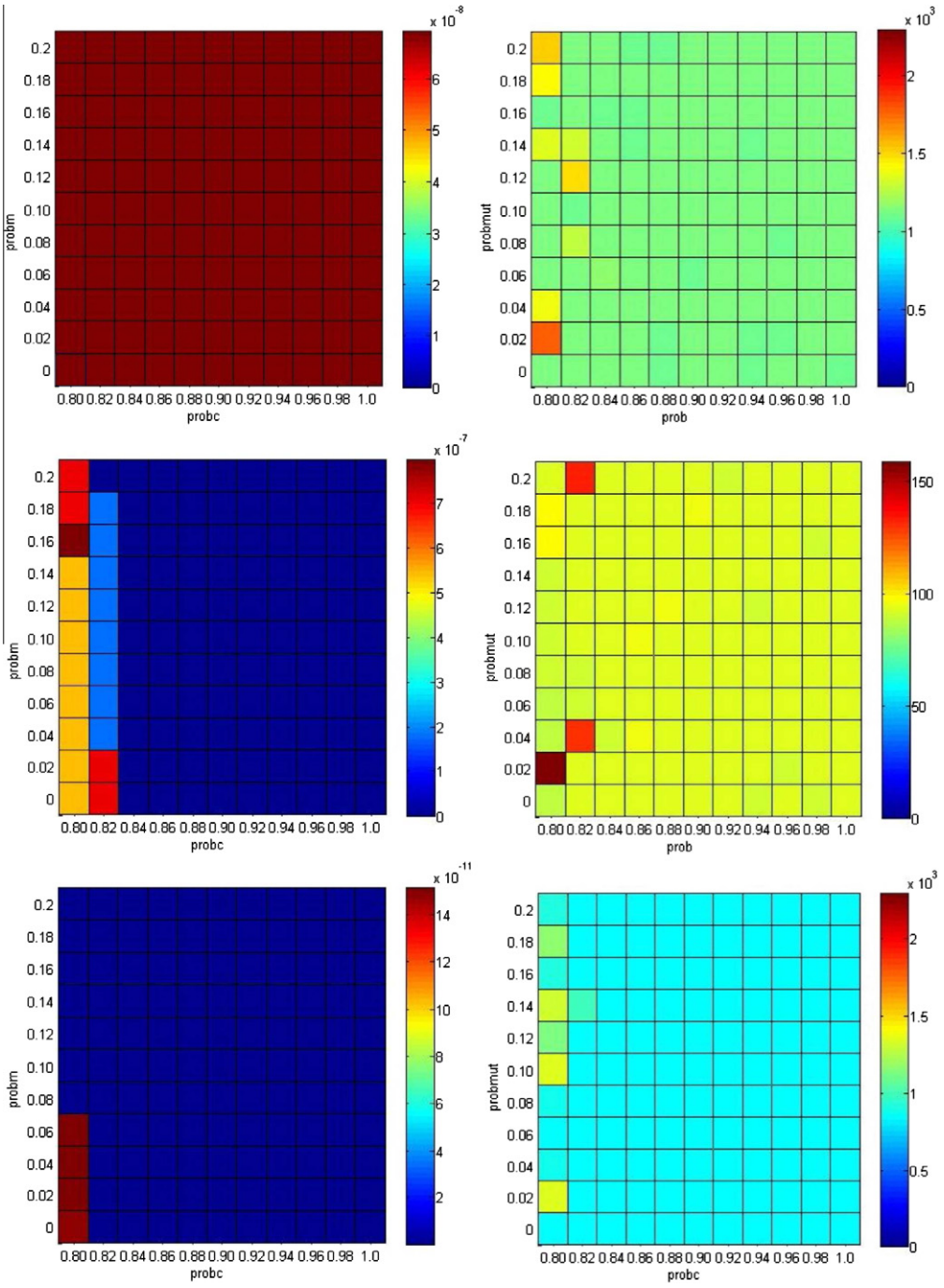

Fig. 9. Mean error of: (left) the method; (right) the CPU time for the examples (from top to bottom) shell, horn and cresent surfaces. For each figure 10 executions have been carried out; then, mean value of the results is displayed.

vertical axis respectively. Results for the other three examples (and others not described in this paper) were very similar, so they are omitted for the sake of brevity. For each example, we display the mean error of the method (left) and the mean value of the CPU time (right). Our results are displayed in color map code where lowest values are colored in dark blue,[1] then light blue, green, yellow, orange, light red and finally dark red for upper values. Corresponding numerical values are indicated in the color bar on the right of each picture.

[1] For interpretation of color in Fig. 9, the reader is referred to the web version of this article.

The main conclusion we obtained is that neither p_c nor p_m play a significant role in the performance or CPU times of the method in the given range. Performance decreases however outside such range, especially for the crossover. In fact, we noticed that values of p_c near to 0.8 degrade the performance noticeably, and this degradation effect increases as the value decreases. In other words, our reported results regarding the accuracy of the method and CPU times are robust against variations of p_c and p_m in the prescribed range.

Finally, in our experiments population size changes from example to example, ranging from $p = 50$ to $p = 500$. Since the population size affects the CPU time (the larger, the longer), our strategy is to keep this value initially as low as possible, then check if an increase of the population size improves the accuracy. This process is repeated until accuracy is no longer improved. Of course, this is at the cost of increasing the PCU times, but as mentioned above, our goal is accuracy rather than speed, so we found this strategy very convenient for our purposes. Moreover, our reported times are still much lower than those of previous works. In general, we found that a population of $p = 100$ individuals is usually enough to get the best accuracy, and larger values do not improve it, only CPU times, but we cannot assure this is a perennial rule for all cases.

7. Conclusions and future work

In this paper, we introduce an efficient GA-based surface reconstruction method from clouds of 3D data points. The method does not assume any input information other than the given set of data points and considers strictly polynomial B-spline surfaces as fitting surfaces. In our approach, GA are applied iteratively to accomplish two tasks: on one hand, to determine the parametric values of data points; on the other hand, to compute surface knot vectors. Then, the fitting surface is calculated by least-squares through either SVD or LU methods. Six examples including open, semi-closed and closed surfaces illustrate the good performance of our approach. Experiments show that the proposed method yields very good results even in presence of problematic features, such as intricate shapes, singularities, non-zero genus, concavities and noisy data. In all examples our fitting errors are astonishingly small when compared to any other previous approach.

In spite of these excellent results, there is still room for further improvement. Future work includes the consideration of NURBS surfaces as fitting surfaces of our method. It is also an open question how to optimize the process in order to reduce the computational load of this approach. On the other hand, our scheme is very general and can therefore be applied to a wide range of practical problems. We expect to apply it to some real-world settings described in the literature. This task will give us a very valuable feedback towards the potential improvement of our current approach.

Acknowledgements

This research has been supported by the Computer Science National Program of the Spanish Ministry of Education and Science, Project Ref. #TIN2006-13615 and the University of Cantabria. We also thank the Editor-in-Chief and the four anonymous reviewers for their encouraging and constructive comments and very helpful feedback that allowed us to improve our paper significantly.

References

[1] T. Back, Evolutionary Algorithms in Theory and Practice, Oxford Univ. Press, NY, 1996.
[2] C. Bajaj, F. Bernardini, G. Xu, Automatic reconstruction of surfaces and scalar fields from 3D stcans, in: Proceedings of SIGGRAPH'95, 1995, pp. 109–118.
[3] J. Barhak, A. Fischer, Parameterization and reconstruction from 3D scattered points based on neural network and PDE techniques, IEEE Transactions on Visualization and Computer Graphics 7 (1) (2001) 1–16.
[4] F.J. Berlanga, A.J. Rivera, M.J. Jesus, F. Herrera, GP-COACH: Genetic Programming-based learning of COmpact and ACcurate fuzzy rule-based classification systems for High-dimensional problems, Information Sciences 180 (8) (2010) 1183–1200.
[5] R.M. Bolle, B.C. Vemuri, On three-dimensional surface reconstruction methods, IEEE Transactions on Pattern Analysis and Machine Intelligence 13 (1) (1991) 1–13.
[6] E. Castillo, A. Iglesias, Some characterizations of families of surfaces using functional equations, ACM Transactions on Graphics 16 (3) (1997) 296–318.
[7] E. Castillo, A. Iglesias, R. Ruiz-Cobo, Functional Equations in Applied Sciences, Elsevier Science, Amsterdam, 2005.
[8] K.Y. Chan, C.K. Kwong, T.C. Fogarty, Modeling manufacturing processes using a genetic programming-based fuzzy regression with detection of outliers, Information Sciences 180 (4) (2010) 506–518.
[9] C.H. Chen, T.L. Cheng, L.Y. Wei, A hybrid model based on rough sets theory and genetic algorithms for stock price forecasting, Information Sciences 180 (9) (2010) 1610–1629.
[10] M.G. Cox, Algorithms for spline curves and surfaces, in: L. Piegl (Ed.), Fundamental Developments of Computer-Aided Geometric Design, Academic Press, London, San Diego, 1993, pp. 51–76.
[11] M. Eck, J. Hadenfeld, Local energy fairing of B-spline curves, in: G. Farin, H. Hagen, H. Noltemeier (Eds.), Computing 10, Springer-Verlag, 1995.
[12] M. Eck, H. Hoppe, Automatic reconstruction of B-spline surfaces of arbitrary topological type, in: Proceedings of SIGGRAPH'96, 1996, pp. 325–334.
[13] M.S. Floater, Parameterization and smooth approximation of surface triangulations, Computer Aided Geometric Design 14 (1997) 231–250.
[14] D.R. Forsey, R.H. Bartels, Surface fitting with hierarchical splines, ACM Transactions on Graphics 14 (1995) 134–161.
[15] H. Fuchs, Z.M. Kedem, S.P. Uselton, Optimal surface reconstruction from planar contours, Communications of the ACM 20 (10) (1977) 693–702.
[16] A. Gálvez, A. Iglesias, A. Cobo, J. Puig-Pey, J. Espinola, Bézier curve and surface fitting of 3D point clouds through genetic algorithms, functional networks and least-squares approximation, Lectures Notes in Computer Science 4706 (2007) 680–693.
[17] A. Gálvez, A. Iglesias, Particle swarm optimization for NURBS surface reconstruction from clouds of 3D data points, submitted for publication.
[18] A. Goinski, Evolutionary surface reconstruction, in: Proceedings of IEEE Conference on Human System Interactions, Krakow, Poland, 2008, pp. 464–469.
[19] D.E. Goldberg, Optimal Initial Population Size for Binary-Coded Genetic Algorithms, TCGA Report No. 85001, University of Alabama, 1985.
[20] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.
[21] M. Gong, L. Jiao, L. Zhang, Baldwinian learning in clonal selection algorithm for optimization, Information Sciences 180 (8) (2010) 1218–1236.
[22] W.J. Gordon, Spline-blended surface interpolation through curve networks, Journal of Mathematics and Mechanics 18 (10) (1969) 931–952.
[23] G. Greiner, Variational design and fairing of spline surfaces, Computer Graphics Forum 13 (3) (1994) 143–154.

[24] W. Groissboeck, E. Lughofer, S. Thumfart, Associating visual textures with human perceptions using genetic algorithms, Information Sciences 180 (11) (2010) 2065–2084.
[25] P. Gu, X. Yan, Neural network approach to the reconstruction of free-form surfaces for reverse engineering, Computer Aided Design 27 (1) (1995) 59–64.
[26] B. Guo, Surface reconstruction from points to splines, Computer-Aided Design 29 (4) (1997) 269–277.
[27] D. Hidalgo, O. Castillo, P. Melin, Type-1 and type-2 fuzzy inference systems as integration methods in modular neural networks for multimodal biometry and its optimization with genetic algorithms, Information Sciences 179 (13) (2009) 2123–2145.
[28] M. Hoffmann, Numerical control of Kohonen neural network for scattered data approximation, Numerical Algorithms 39 (2005) 175–186.
[29] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975.
[30] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, Proceedings of SIGGRAPH'92, Computer Graphics 26 (2) (1992) 71–78.
[31] J. Hoschek, Smoothing of curves and surfaces, Computer Aided Design 2 (1985) 97–105.
[32] T.H. Hsu, T.N. Tsai, P.L. Chiang, Selection of the optimum promotion mix by integrating a fuzzy linguistic decision model with genetic algorithms, Information Sciences 179 (1–2) (2009) 41–52.
[33] Y.C. Hu, Analytic network process for pattern classification problems using genetic algorithms, Information Sciences 180 (13) (2010) 2528–2539.
[34] J.C. Hung, A genetic algorithm approach to the spectral estimation of time series with noise and missed observations, Information Sciences 178 (24) (2008) 4632–4643.
[35] O. Ibáñez, L. Ballerini, O. Cordón, S. Damas, J. Santamaría, An experimental study on the applicability of evolutionary algorithms to craniofacial superimposition in forensic identification, Information Sciences 179 (23) (2009) 3998–4028.
[36] A. Iglesias, A. Gálvez, A new artificial intelligence paradigm for computer aided geometric design, Lectures Notes in Artificial Intelligence 1930 (2001) 200–213.
[37] A. Iglesias, G. Echevarría, A. Gálvez, Functional networks for B-spline surface reconstruction, Future Generation Computer Systems 20 (8) (2004) 1337–1353.
[38] M. Jones, M. Chen, A new approach to the construction of surfaces from contour data, Computer Graphics Forum 13 (3) (1994) 75–84.
[39] I. Kaya, A genetic algorithm approach to determine the sample size for attribute control charts, Information Sciences 179 (10) (2009) 1552–1566.
[40] R. Klein, A. Schilling, W. Strasser, Reconstruction and simplification of surfaces from contours, Graphical Models 62 (6) (2000) 429–443.
[41] G.K. Knopf, J. Kofman, Free-form surface reconstruction using Bernstein basis function networks, in: C.H. Dagli et al. (Eds.), Intelligent Engineering Systems Through Artificial Neural Networks, vol. 9, ASME Press, 1999, pp. 797–802.
[42] S.G. Kumar, P.K. Kalra, S.G. Dhande, Parameter optimization for B-spline curve fitting using genetic algorithms, in: Proceedings of Congress on Evolutionary Computation, vol. 3, 2003, pp. 1871–1878.
[43] S.G. Kumar, P.K. Kalra, S.G. Dhande, Curve and surface reconstruction from points: an approach based on self-organizing maps, Applied Soft Computing 5 (5) (2004) 55–66.
[44] P. Laurent, M. Mekhilef, Optimization of a NURBS representation, Computer Aided Design 25 (11) (1993) 699–710.
[45] M.C. Leu, X. Peng, W. Zhang, Surface reconstruction for interactive modeling of freeform solids by virtual sculpting, CIRP Annals-Manufacturing Technology 54 (1) (2005) 131–134.
[46] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, D. Fulk, The Digital Michelangelo Project: 3D scanning of large statues, in: SIGGRAPH 2000, New Orleans, 2000, pp. 131–144.
[47] C. Lim, G. Turkiyyah, M. Ganter, D. Storti, Implicit reconstruction of solids from cloud point sets, in: Proceedings of 1995 ACM Symposium on Solid Modeling, Salt Lake City, Utah, 1995, pp. 393–402.
[48] M. Lozano, F. Herrera, J.R. Cano, Replacement strategies to preserve useful diversity in steady-state genetic algorithms, Information Sciences 178 (23) (2008) 4421–4433.
[49] W.Y. Ma, J.P. Kruth, Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces, Computer Aided Design 27 (1995) 663–675.
[50] I. Maekawa, K. Ko, Surface construction by fitting unorganized curves, Graphical Models 64 (2002) 316–332.
[51] R. Martínez, O. Castillo, L.T. Aguilar, Optimization of interval type-2 fuzzy logic controllers for a perturbed autonomous wheeled mobile robot using genetic algorithms, Information Sciences 179 (13) (2009) 2158–2174.
[52] D. Meyers, S. Skinnwer, K. Sloan, Surfaces from contours, ACM Transactions on Graphics 11 (3) (1992) 228–258.
[53] M. Mitchell, An Introduction to Genetic Algorithms (Complex Adaptive Systems), MIT Press, 1998.
[54] H. Park, K. Kim, Smooth surface approximation to serial cross-sections, Computer Aided Design 28 (12) (1997) 995–1005.
[55] H. Park, J.H. Lee, Error-bounded B-spline curve approximation based on dominant point selection, in: Proceedings of International Conference on Computer Graphics, Imaging and Visualization (CGIV05), 2005, pp. 437–446.
[56] N.M. Patrikalakis, T. Maekawa, Shape Interrogation for Computer Aided Design and Manufacturing, Springer-Verlag, 2002.
[57] L. Piegl, W. Tiller, The NURBS Book, Springer-Verlag, Berlin, Heidelberg, 1997.
[58] L. Piegl, W. Tiller, Algorithm for approximate NURBS skinning, Computer Aided Design 28 (9) (1997) 699–706.
[59] H. Pottmann, S. Leopoldseder, M. Hofer, Approximation with active B-spline curves and surfaces, in: Proceedings of Pacific Graphics, IEEE Computer Society Press, 2002, pp. 8–25.
[60] H. Pottmann, S. Leopoldseder, M. Hofer, T. Steiner, W. Wang, Industrial geometry: recent advances and applications in CAD, Computer-Aided Design 37 (2005) 751–766.
[61] M. Prasad, A. Zisserman, A.W. Fitzgibbon, Single view reconstruction of curved surfaces, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE Computer Society Press, Los Alamitos, CA, 2006.
[62] V. Pratt, Direct least-squares fitting of algebraic surfaces, Proceedings of SIGGRAPH'87, Computer Graphics 21 (4) (1987) 145–152.
[63] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes, second ed., Cambridge University Press, Cambridge, 1992.
[64] P. Ravisankar, V. Ravi, I. Bose, Failure prediction of dotcom companies using neural network genetic programming hybrids, Information Sciences 180 (8) (2010) 1257–1267.
[65] J. Sanchis, M.A. Marínez, X. Blasco, Integrated multiobjective optimization and a priori preferences using genetic algorithms, Information Sciences 1784 (2008) 931–951.
[66] L.M. San José-Revuelta, A new adaptive genetic algorithm for fixed channel assignment, Information Sciences 177 (13) (2007) 2655–2678.
[67] M. Sarfraz, S.A. Raza, M.H. Baig, Computing optimized curves with NURBS using evolutionary intelligence, Lecture Notes in Computer Science 3480 (2005) 806–815.
[68] M. Sarfraz, Computer-aided reverse engineering using simulated evolution on NURBS, Virtual and Physical Prototyping 1 (4) (2006) 243–257.
[69] B. Sarkar, C.H. Menq, Parameter optimization in approximating curves and surfaces to measurement data, Computer Aided Geometric Design 8 (1991) 267–290.
[70] V. Savchenko, A. Pasko, O. Okunev, T. Kunii, Function representation of solids reconstructed from scattered surface points and contours, Computer Graphics Forum 14 (4) (1995) 181–188.
[71] F. Schmitt, B.A. Barsky, W. Du, An adaptive subdivision method for surface fitting from sampled data, Proceedings of SIGGRAPH'86, Computer Graphics 20 (4) (1986) 179–188.
[72] A. Ugur, Path planning on a cuboid using genetic algorithms, Information Sciences 178 (16) (2008) 3275–3287.
[73] E. Ulker, V. Isler, An artificial immune system approach for B-spline surface approximation problem, Lecture Notes in Computer Science 4488 (2007) 49–56.

[74] E. Ulker, A. Arslan, The calculation of parametric NURBS surface interval values using neural networks, Lecture Notes in Computer Science 3992 (2006) 247–254.
[75] E. Ulker, A. Arslan, Automatic knot adjustment using an artificial immune system for B-spline curve approximation, Information Sciences 179 (2009) 1483–1494.
[76] T. Varady, R. Martin, Reverse engineering, in: G. Farin, J. Hoschek, M. Kim (Eds.), Handbook of Computer Aided Geometric Design, Elsevier Science, 2002.
[77] T. Wagner, T. Michelitsch, A. Sacharow, On the design of optimizers for surface reconstruction, in: Proceedings of the 2007 Genetic and Evolutionary Computation Conference-GECCO2007, London, England, 2007, pp. 2195–2202.
[78] K. Weinert, J. Mehnen, F. Albersmann, P. Dreup, Optimized NURBS ship hull fitting using simulated annealing, in: Proceedings of International Conference on Computer Graphics, Imaging and Visualisation (CGIV'06), Capri, Italy, 1998, pp. 431–438.
[79] K. Weinert, S.M.H. Shamsuddin, Y. Samian, Evolutionary surface reconstruction using CSG-NURBS-hybrids, in: Proceedings of the 2001 Genetic and Evolutionary Computation Conference-GECCO2001, San Francisco, USA, 2001, pp. 1456–1463.
[80] V. Weiss, L. Or, G. Renner, T. Varady, Advanced surface fitting techniques, Computer Aided Geometric Design 19 (2002) 19–42.
[81] A.S. Wen, New solutions for surface reconstruction from discrete point data by means of computational intelligence, in: Proceedings of Intelligent Computation in Manufacturing Engineering-ICME'98, Sydney, Australia, IEECS Press, 2006, pp. 431–438.
[82] Q. Yuan, F. Qian, W. Du, A hybrid genetic algorithm with the Baldwin effect, Information Sciences 180 (5) (2010) 640–652.
[83] J. Zhang, J. Zhuang, S. Du, S. Wang, Self-organizing genetic algorithm based tuning of PID controllers, Information Sciences 179 (7) (2009) 1007–1018.