Bézier Curve and Surface Fitting of 3D Point Clouds Through Genetic Algorithms, Functional Networks and Least-Squares Approximation

Akemi Gálvez[1], Andrés Iglesias[1], Angel Cobo[1], Jaime Puig-Pey[1], and Jesús Espinola[2]

[1] Department of Applied Mathematics and Computational Sciences, University of Cantabria, Avda. de los Castros, s/n, E-39005, Santander, Spain
akemi.galvez@postgrado.unican.es, {iglesias,acobo,puigpeyj}@unican.es
[2] Faculty of Sciences, National University Santiago Antúnez de Mayolo, Perú
espinolj@gmail.com

Abstract. This work concerns the problem of curve and surface fitting. In particular, we focus on the case of 3D point clouds fitted with Bézier curves and surfaces. Because these curves and surfaces are parametric, we are confronted with the problem of obtaining an appropriate parameterization of the data points. On the other hand, the addition of functional constraints introduces new elements that classical fitting methods do not account for. To tackle these issues, two Artificial Intelligence (AI) techniques are considered in this paper: (1) for the curve/surface parameterization, the use of genetic algorithms is proposed; (2) for the functional constraints problem, the functional networks scheme is applied. Both approaches are combined with the least-squares approximation method in order to yield suitable methods for Bézier curve and surface fitting. To illustrate the performance of those methods, some examples of their application on 3D point clouds are given.

1 Introduction

Fitting curves and surfaces to measurement data plays an important role in real problems such as manufacturing of car bodies, ship hulls, airplane fuselage, and other free-form objects. One typical geometric problem in Reverse Engineering is the process of converting dense data points captured from the surface of an object into a boundary representation CAD model [17,19]. Most of the usual models for fitting in Computer Aided Geometric Design (CAGD) are free-form parametric curves and surfaces, such as Bézier, Bspline and NURBS.

Curve/surface fitting methods are mainly based on the least-squares approximation scheme, a classical optimization technique that (given a series of measured data) attempts to find a function which closely approximates the data (a "best fit"). Suppose that we are given a set of n_p data points $\{(x_i, y_i)\}_{i=1,\ldots,n_p}$, and we want to find a function f such that $f(x_i) \approx y_i, \forall i = 1, \ldots, n_p$. The typical approach is to assume that f has a particular functional structure which depends on some parameters that need to be calculated. The procedure is to

O. Gervasi and M. Gavrilova (Eds.): ICCSA 2007, LNCS 4706, Part II, pp. 680–693, 2007.
© Springer-Verlag Berlin Heidelberg 2007

find the parameter values minimizing the sum S of squares of the ordinate differences (called *residuals*) between the points generated by the function and corresponding points in the data:

$$S = \sum_{i=1}^{n_p} (y_i - f(x_i))^2. \tag{1}$$

It is well-known that if the function f is linear, the problem simplifies considerably as it essentially reduces to a system of linear equations. By contrast, the problem becomes more difficult if such function is not linear, since we then need to solve a general (unconstrained) optimization problem.

In this work, we consider the case of f being a free-form parametric curve or surface. In the former case, we have a curve $\mathbf{C}(t)$ given by:

$$\mathbf{C}(t) = \sum_{j=0}^{M} \mathbf{P}_j B_j(t) \tag{2}$$

where $\mathbf{P}_j = (P_j^x, P_j^y, P_j^z)$ are the vector coefficients (usually called *control points*), $\{B_j(t)\}_{j=0,\ldots,M}$ are the *basis functions* (or *blending functions*) of the parametric curve $\mathbf{C}(t)$ and t is the parameter, usually defined on a finite interval $[\alpha, \beta]$. Note that in this paper vectors are denoted in bold. Now we can compute, for each of the cartesian components (x, y, z), the minimization of the sum of squared errors referred to the data points according to (1), but we need a parameter value t_i to be associated with each data point (x_i, y_i, z_i), $i = 1, \ldots, n_p$:

$$Err_\mu = \sum_{i=1}^{n_p} \left(\mu_i - \sum_{j=0}^{M} P_j^\mu B_j(t_i) \right)^2 \quad ; \quad \mu = x, y, z \tag{3}$$

Coefficients $\mathbf{P}_j$, $j = 0, \ldots, M$, have to be determined from the information given by the data points (x_i, y_i, z_i), $i = 1, \ldots, n_p$. Note that performing the component-wise minimization of these errors is equivalent to minimizing the sum, over the set of data, of the Euclidean distances between data points and corresponding points given by the model in 3D space. This problem is far from being trivial: because our curves and surfaces are parametric, we are confronted with the problem of obtaining a suitable parameterization of the data points. As remarked in [1] the selection of an appropriate parameterization is essential for topology reconstruction and surface fitness. Many current methods have topological problems leading to undesired surface fitting results, such as noisy self-intersecting surfaces. In general, algorithms for automated surface fitting [2,13] require knowledge of the connectivity between sampled points prior to parametric surface fitting. This task becomes increasingly difficult if the capture of the coordinate data is unorganized or scattered. Most of the techniques used to compute connectivity require a dense data set to prevent gaps and holes, which can significantly change the topology of the generated surface. Therefore, in addition to the coefficients of the basis functions, $\mathbf{P}_j$, the parameter values,

t_i, $i = 1, \ldots, n_p$, corresponding to the data points also appear as unknowns in our formulation. Due to the fact that the blending functions $B_j(t)$ are nonlinear in t, the least-squares minimization of the errors becomes a strongly nonlinear problem [20], with a high number of unknowns for large sets of data points, a case that happens very often in practice.

Some recent papers have shown that the application of Artificial Intelligence (AI) techniques can achieve remarkable results regarding this problem [7,10,11,14,15,16]. Most of these methods rely on some kind of neural networks, either standard neural networks [10], Kohonen's SOM (Self-Organizing Maps) nets [1,11], or the Bernstein Basis Function (BBF) network [16]. In some cases, the network is used exclusively to order the data and create a grid of control vertices with quadrilateral topology. After this preprocessing step, any standard surface reconstruction method (such as those referenced above) has to be applied. In other cases, the neural network approach is combined with partial differential equations [1] or other approaches.

Our strategy for tackling the problem also belongs to this group of AI techniques. As it will be described later on, we propose a hybrid method combining genetic algorithms for searching the parameter values t_i for the data points, and computing the best least-squares fitting coefficients $\mathbf{P}_j$ and the corresponding error for the set of parameter values provided by the genetic algorithm. The process is performed iteratively until a certain termination condition is reached. Details about this method will be given in Sections 2 to 5.

Although the genetic algorithm approach works well for curve/surface fitting (particularly, with the problem of curve/surface parameterization), neither this method nor the standard neural networks account for additional functional constraints that are described by mathematical equations rather than by discrete sets of points. However, this case is quite usual in the realm of computer graphics and CAGD. For instance, biomedical images are often generated from a sequence of cross-sections, three-dimensional volumes can be generated from the intersection of the objects with parallel planes, etc. Even if the user is provided with a large set of data points, any additional information (such as cross-sections or any other kind of curve on the interpolating/approximating surface) might greatly simplify the problem. This kind of functional information, while being quite useful for the given problem, is rarely used in applied domains. Perhaps the main reason is the difficulty to solve the functional equations arising from such constraints (the reader is referred to [6] for a gentle introduction to functional equations). In this paper, a new formalism, the *functional networks*, are applied to solve the Bézier surface fitting when some functional constraints are also considered (see Sections 6 and 7 for details).

2 Genetic Algorithms

Genetic Algorithms (GA) [9] 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 coded solutions as

finite-length strings. They were developed by John Holland at the University of Michigan [12] and are categorized as global search heuristics or meta-heuristics [3], a group of techniques that encompasses trajectory methods such as Tabu Search, Simulated Annealing or Iterated Local Search, and population-based methods such as Genetic Algorithms, Ant Colonies and Particle Swarm Optimization, to mention just a few methods.

Genetic Algorithms handle populations consisting of a set of potential solutions, i.e. the algorithm maintains a population of n individuals $Pop(iter) = \{x_1(iter), \ldots, x_n(iter)\}$ for each iteration *iter*, 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 can be either constant or variable. A study about the optimal population size can be found in [8]. Each individual in the population, i.e. 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 as a biased roulette wheel with slots weighted in proportion to individual adaptation values. The selection process is repeated n 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 a GA the crossover operator is randomly applied with a specific probability, a good GA performance requires the choice of a high crossover probability. *Mutation* is a unitary transformation which creates, with low probability, 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.

In spite of their surprising simplicity, GAs have been recognized as a powerful tool to solve optimization problems in various fields of applications; examples of such problems can be found in a great variety of domains such as transportation problems, wire routing, travelling salesman problem [9]. The CAD (Computer-Aided Design) journal devoted the 35 special issue of 2003 to genetic algorithms

Table 1. 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
Offspring 1	0.123	0.178	**0.271**	**0.274**	0.367	0.521	0.679	0.781	0.812	0.912
				chromosomes sorting						
Offspring 2	0.086	0.167	0.197	**0.456**	0.571	0.701	0.789	0.843	0.921	0.950

[18], and included one paper addressing data fitting with B-splines polynomials in explicit form [21]. In this work we consider parametric models instead, which are by far more relevant for CAGD than the explicit ones.

3 Using Genetic Algorithms for Data Fitting

In order to use GA for fitting curves/surfaces to data points, several aspects must be previously considered. First of all, a typical GA requires two elements to be defined prior to its use: the genetic representation of each potential solution of the problem and a measure of the quality of the solution (usually referred to as the *fitness function*). In our problem, we are interested on the assignment process of parameter values to data points, so we propose the use of a real-coded genetic algorithm in which the genetic representation of an individual will be a real n_p-dimensional vector, where each coordinate represents the parameter value assigned to a data point. The fitness function that allows measuring the quality of an assignment will be based on the error function of the fitting process.

As initial population we will consider a randomly generated set of parameter vectors (individuals). 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. 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. We use an one-point crossover operator that randomly selects a crossover point within an individual, then swaps the two parent chromosomes to the left and to the right from this point and eventually sorts the obtained vectors to produce two new offsprings. This process is illustrated in Table 1.

As a mutation method we propose to select the position k with worst fit error in the vector parameter 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, $t_k = \dfrac{t_{k-1} + t_{k+1}}{2}$. Note that $t_{k-1} < t_k < t_{k+1}$, and hence no sorting method is required. Using these genetic operators, the general structure of the algorithm is described in Table 2.

Table 2. General structure of the genetic algorithm

```
begin
    iter=0
    random initialization of Pop(iter)
    fitness evaluation of Pop(iter)
    while (not termination condition) do
        Select individuals from Pop(iter)
        Apply crossover and mutation operator with probabilities p_c and p_m
        Set Pop(iter + 1)
        iter = iter + 1
    end
end
```

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.

4 Best Least-Squares Approximation

Let us consider a set of 3D data points $\mathbf{D}_i = (x_i, y_i, z_i)$, $i = 1, \ldots, n_p$. We describe our procedure in more detail for the x's coordinates, (the extension to y's and z's is immediate). The goal is to calculate the coefficients P_j^x, $j = 0, \ldots, M$, which give the best fit in the discrete least-squares sense to the column vector $\mathbf{X} = (x_1, \ldots, x_{n_p})^T$ where $(.)^T$ means transposition, by using the model $x(t) = \sum_{j=0}^{M} P_j^x B_j(t)$, supposing that t_i $(i = 1, \ldots, n_p)$ are parameter values assigned to the data points and the $B_j(t)$ are the known blending functions of the model. Considering the column vectors $\mathbf{B}_j = (B_j(t_1), \ldots, B_j(t_{n_p}))^T$, $j = 0, \ldots, M$ and solving the following system gives the coefficients P_j^x:

$$\begin{pmatrix} \mathbf{B}_0^T.\mathbf{B}_0 & \ldots & \mathbf{B}_M^T.\mathbf{B}_0 \\ \vdots & \vdots & \vdots \\ \mathbf{B}_0^T.\mathbf{B}_M & \ldots & \mathbf{B}_M^T.\mathbf{B}_M \end{pmatrix} \begin{pmatrix} P_0^x \\ \vdots \\ P_M^x \end{pmatrix} = \begin{pmatrix} \mathbf{X}^T.\mathbf{B}_0 \\ \vdots \\ \mathbf{X}^T.\mathbf{B}_M \end{pmatrix}. \tag{4}$$

The elements of the coefficient matrix and the independent terms are calculated by performing a standard Euclidean scalar product between finite-dimensional vectors. This system (4) results from minimizing the sum of squared errors referred to the x_i coordinates of the data points, as indicated in Section 1. Considering all the x_i, y_i, z_i coordinates, the solution of the three linear systems with the same coefficient matrix provides the best least-squares approximation for the curve $\mathbf{C}(t) = \sum_{j=0}^{M} \mathbf{P}_j B_j(t)$. For surfaces in parametric form, one uses the

tensor-product surface $\mathbf{S}(u,v) = \sum_{i=0}^{N}\sum_{j=0}^{M}\mathbf{P}_{i,j}B_i(u)B_j(v)$, a very common model in CAGD. The coefficients $\mathbf{P}_{i,j}$ are the control points in 3D, arranged in a quadrilateral topology, and functions $B_i(u)$ and$B_j(v)$ are the same basis functions used for representing curves, for example Bernstein polynomials or B-splines piecewise polynomials. The parameters u and v are valued on a rectangular domain $[u_m, u_M] \times [v_m, v_M]$, a Cartesian product of the respective domains for u and v. If $B_i(u)$ and $B_j(v)$ are Bézier basis functions, the $(M+1).(N+1)$ bivariate polynomials $B_{i,j}(u,v) = B_i(u).B_j(v)$, $i = 0,\ldots,N$, $j = 0,\ldots,M$ constitute a vector basis for a linear vector space of polynomials in u and v on the squared domain $[0,1] \times [0,1]$. Given a cloud of points $(x_{l,k}, y_{l,k}, z_{l,k})$, in 3D, with a quadrilateral structure, $l = 1,\ldots,n_{p_u}$, $k = 1,\ldots,n_{p_v}$, and a set of parameter values (u_l, v_k) associated one-to-one with the data points in the cloud such that these points form a cartesian set in the parameter domain, a discrete formulation similar to that for fitting points to a curve can be made. The best least-squares tensor product surface fitting the points can be obtained using the system (4), in which the role of the $\mathbf{B}$'s is now assumed by the $B_{i,j}(u,v)$ described earlier.

5 Examples

In this section two examples (a Bézier curve and a Bézier surface) aimed at showing the performance of our method are discussed.

5.1 Fitting a Bézier Curve

As a first example we consider a Bézier curve of degree d whose parametric representation is given by Eq. (2) where the basis functions of degree d are defined as: $B_i^d(t) = \binom{d}{i}\ t^i\ (1-t)^{d-i}$, $i = 0,\ldots,d$. In this example, we have chosen a set of eigth 3D points to be fitted to a Bézier curve of degree $d = 4$. The unknowns are 23 scalar values: a vector of 8 parameter values associated with the 8 data points, plus 15 coefficients (3 for the coordinates of each of the 5 control points of the curve of degree 4). The data for the genetic algorithm are set as follows: we select an initial population of 100 parameter vectors, each having 8 elements generated randomly from a Uniform$[0,1]$ distribution and sorted in increasing order. Then, we apply the procedure shown in Table 2 to produce successive generations. In this example, the crossover and mutation operators are applied with probabilities $p_c = 0.90$ and $p_m = 0.20$, respectively. A typical output for a couple of parent chromosomes is shown in Table 1, yielding two new offsprings in the next generation. Regarding our termination condition, these steps are repeated until the results no longer change for 20 successive iterations. In this example, the optimal fit is attained at the $76th$ generation with the following results: the error in the population (the maximum point error in the best fitted parameter vector) is 1.8774, while the mean error in the population is 2.0875. The number of crossings and mutations for the last generation are 46 and 24, respectively. The optimum

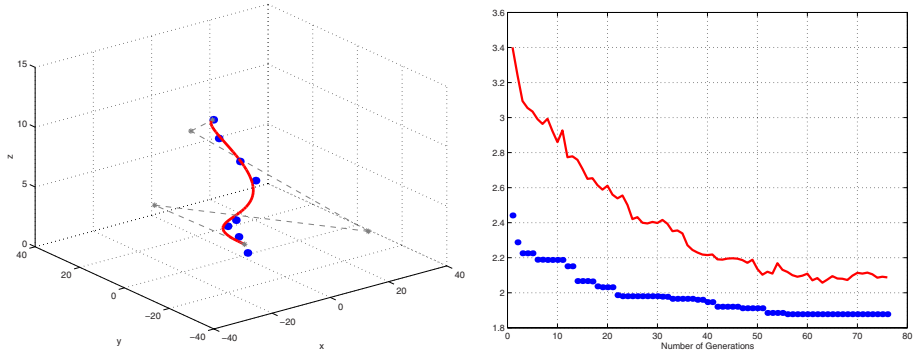

Fig. 1. Example of a Bézier curve fitting: (left) the Bézier curve along with its control points (stars) and the data points (spheres); (right) evolution of the mean (solid line) and the best (dotted line) Euclidean errors along the generations

parameter vector obtained is $[0, 0.0131, 0.0583, 0.3556, 0.5384, 0.7138, 0.7899, 1]$. The computation time for this example (in Matlab, running on a Pentium IV, 3 GHz) has been 4.16 sec.

Fig. 1(left) shows the data points (represented as spheres), the fourth-degree 3D Bézier fitting curve and its 5 control points (represented as stars). Fig 1(right) shows the evolution of the mean (solid line) and the best (dotted line) Euclidean errors of the parameter vectors for each generation along the successive generations. Note that the best error becomes small very quickly at the beginning, the reduction rate getting slower for later generations.

5.2 Fitting a Bézier Surface

We consider now a parametric Bézier surface of degree N in u and M in v whose representation is given by:

$$\mathbf{S}(u,v) = \sum_{i=0}^{N} \sum_{j=0}^{M} \mathbf{P}_{i,j} B_i^N(u) B_j^M(v) \tag{5}$$

where the basis functions (the Bernstein polynomials) are defined as above and the coefficients $\mathbf{P}_{i,j}$ are the surface control points. For this example we take 256 data points from a bicubic Bézier surface, generated with the following parameter values for u and v: for the u's of data points, we choose two groups of 8 equidistant parameter values in the intervals $[0, 0.2]$ and $[0.8, 1]$ and similarly for the v parameter. The unknowns are $3 \times 16 = 48$ scalar coefficients (3 coordinates for each of 16 control points) and two parameter vectors $\mathcal{U}$ and $\mathcal{V}$ (each of size 16), which combined by product of sets, are associated with the 256 data points. That makes a total of 80 scalar unknowns.

The input parameters for the procedure are as follows: Population size: 200; $p_c = 0.95$; $p_m = 0.20$; Termination criteria =no improvement after 30 consecutive

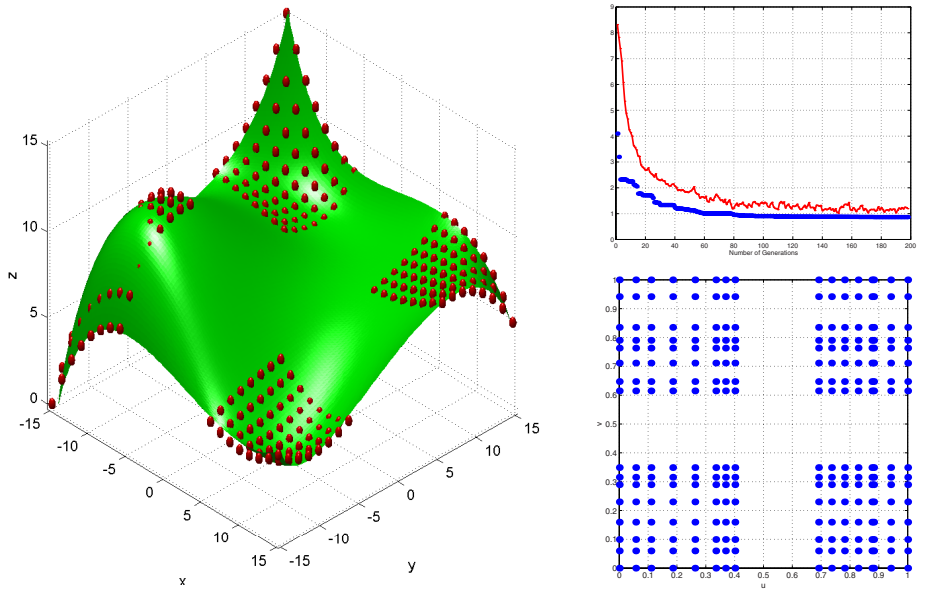

Fig. 2. Example of Bézier surface fitting: (left) bicubic Bézier surface and data points; (right-top): evolution of the mean (solid line) and the best (dotted line) Euclidean errors along the generations; (right-bottom): optimum parameter values on the parametric domain

generations. Initially, we have a population of 200 $\mathcal{U}$-vectors and 200 $\mathcal{V}$-vectors, each one constructed by assigning random parameter values with Uniform$[0, 1]$ distribution, and sorting them within each vector. The best solution is attained at generation 198 with the following results: Least error in the fit: 0.9891; Mean error: 1.2314; Number of crossings (resp. mutations) for the last generation: 103 (resp. 59); cpu time (Pentium IV, 3 GHz running Matlab): 33.81 sec.

Fig. 2(left) shows the data points and the bicubic Bézier fitting surface. In Fig. 2(right-top) we display the evolution of mean error (solid line) and best (dotted line) distance error for each generation along the iterations. The optimum parameter values for u and v are depicted in Fig 2(right-bottom) where one can see how the fitting process grasps the distribution of parameters values assigned to the data points. It is worthwhile to mention the tendency of the obtained parameter values to concentrate at the corners of the unit square parameter domain, thus adjusting quite well the input information.

6 Functional Networks

An alternative approach to solving the problem of scattered data parameterization is to assume a predefined 2D-mesh with the desired connectivity between neighbouring vertices or nodes. An interpolation algorithm is then used to iteratively adjust the nodes in the mesh in order to match the coordinate data

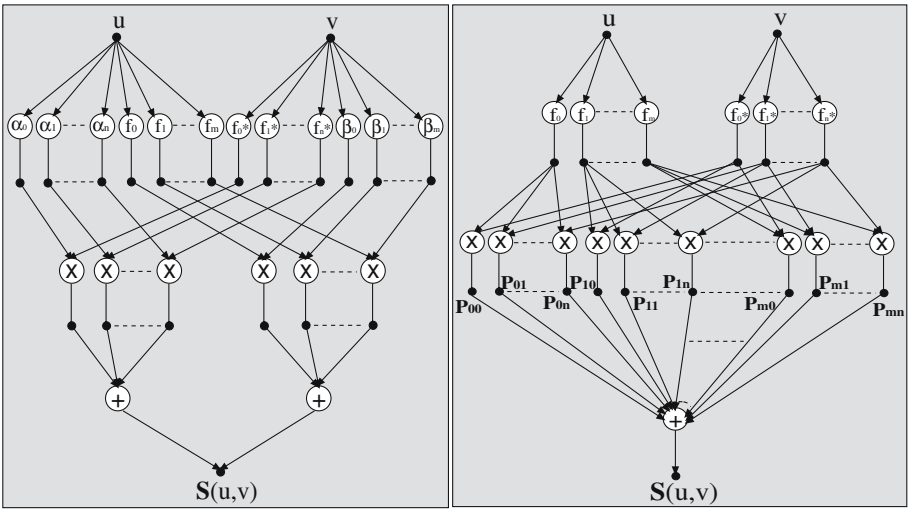

Fig. 3. (left) Representation of the functional network for the parametric surface in eq. (6); (right) Equivalent functional network

set. This interpolation can be replaced by a coarser approximation, provided that some additional (functional) information is available. For instance, in some practical cases we can determine either the boundary curves or some 3D cross-sections of the surface. Sometimes, such curves are described by their mathematical functions rather than by a set of data points (this situation is usually referred to as *transfinite interpolation*), which cannot be embedded into the neural network paradigm [14]. In [5] the authors suggested a powerful extension of the classical neural networks, the so-called *functional networks*, able to deal with this kind of functional constraints. The functional network (FN) formalism has been successfully applied to surface reconstruction problems [7,15].

In this paper we consider the case of a Bézier surface subjected to some functional constraints. In particular, let us assume that we look for a Bézier surface $\mathbf{S}(u, v)$ such that its isoparametric curves $u = u_0$ and $v = v_0$ are linear combinations of the sets of functions (not necessarily belonging to the same family of functions; see example below) $\mathbf{f}(u) = \{f_0(u), f_1(u), \ldots, f_m(u)\}$ and $\mathbf{f}^*(v) = \{f_0^*(v), f_1^*(v) \ldots, f_n^*(v)\}$ respectively. In other words, we look for surfaces $\mathbf{S}(u, v)$ such that they satisfy the system of functional equations

$$\mathbf{S}(u,v) \equiv \sum_{j=0}^{n} \boldsymbol{\alpha}_j(u) f_j^*(v) = \sum_{i=0}^{m} \boldsymbol{\beta}_i(v) f_i(u) \tag{6}$$

where the sets of coefficients $\{\boldsymbol{\alpha}_j(u); j = 0, 1, \ldots, n\}$ and $\{\boldsymbol{\beta}_i(v); i = 0, 1, \ldots, m\}$ can be assumed, without loss of generality, as sets of linearly independent functions. This problem admits the graphical representation given in Figure 3(left)

which, at first sight, looks like a neural network. However, the previous description in terms of neural networks presents the following problems:

1. Neural functions in neural networks are identical, whereas neural functions in our example are different. For instance, we may find product and sum operators (symbols '$\times$' and '$+$' in Figure 3(left) respectively).
2. In neural networks there are weights, which must be learned. These weights do not appear in functional networks; neural functions are learned instead.
3. The neuron outputs of neural networks are different; however, in our scheme, some neuron outputs in the example are coincident (this is the case of the output $\mathbf{S}(u, v)$ in Figure 3(left)). This fact leads to a set of functional equations, which have to be solved.

These and other disadvantages suggest that the neural networks paradigm should be improved. In this paper, we will do so by using *the functional networks*. In particular, the solution of this problem is given by (see [4]):

$$\mathbf{S}(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{P}_{ij} f_i(u) f_j^*(v) = \mathbf{f}(u).\mathbf{P}.(\mathbf{f}^*(v))^T \tag{7}$$

where $\mathbf{P}_{ij}$ are elements of an arbitrary vector matrix $\mathbf{P}$, that is, $\mathbf{S}(u, v)$ is a tensor product surface. The resulting functional network is depicted in Fig. 3(right).

7 Using Functional Networks for Bézier Surface Fitting

In this section the functional networks approach is combined with the least-squares method to solve the surface fitting problem. In this paper we focus on Bézier surfaces, so the families of functions $\{f_i(u)\}_i$ and $\{f_j^*(v)\}_j$ are both Bernstein polynomials. Figure 4 shows a typical example of the problems the functional networks can solve. We look for the Bézier surface that *interpolates* some given curves (transfinite interpolation) and *best approximates* (in the sense of least-squares) a given set of 3D data points (displayed as red spheres in Fig. 4(right)). The functional constraints in this example are given in the form of cross-sections[1], shown in Fig. 4(left-top and middle): 5 section curves in u direction, described as polynomials of different degrees, and 6 section curves in v direction, described as Bézier curves. We remark that the data points do not have a rectangular structure, as shown in Fig. 4(left-bottom), where the (u, v) coordinates of the given points are displayed on the parametric domain. Note that in general we cannot expect data points to belong to the target surface (see Fig. 4(right)), which is forced to fulfill the functional constraints. Those constraints are described by systems of functional equations in our formulation. Note also that several arbitrary constraints might become incompatible in practice. In those cases, the resulting functional equations have no solution, so the system of functional equations play the role of the *compatibility conditions* [15].

[1] This example is discussed here for illustrative purposes. Other (more awkward) kind of functional constraints might also be considered.

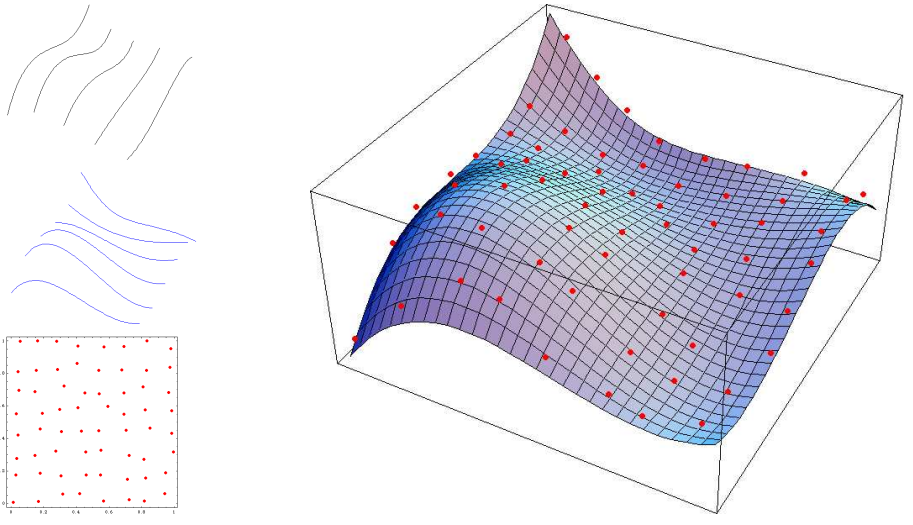

Fig. 4. Example of Bézier surface fitting: (left, top and middle) curve constraints; (left-bottom) coordinates of the data points on the parametric domain; (right) fitted Bézier surface along with the approximated data points

The least-squares method is now applied during the learning process, in which the neural functions are estimated (learned). To do so, each neural function f_i is approximated by a linear combination of functions in a given family $\{\phi_{i0}, \ldots, \phi_{im_i}\}$. Thus, the approximated neural function $\hat{f}_i(\mathbf{x})$ becomes $\hat{f}_i(\mathbf{x}) = \sum_{j=0}^{m_i} a_{ij}\phi_{ij}(\mathbf{x})$ where $\mathbf{x}$ are the inputs associated with the i-th neuron. In our case, this step reduces to estimating the neuron functions $x(u, v)$, $y(u, v)$ and $z(u, v)$ from a given sequence of triplets $\{(x_k, y_k, z_k)\}_{k=1,\ldots,K}$ which depend on u and v so that $x(u_k, v_k) = x_k$ and so on. So we build the sum of squared errors function (see Eq. (3)):

$$Q_\mu = \sum_{k=1}^{K} \left(\mu_k - \sum_{i=0}^{I} \sum_{j=0}^{J} a_{ij}\phi_i(u_k)\psi_j(v_k) \right)^2 \tag{8}$$

where we must consider an error function for each variable x, y and z. This is assumed by μ in the previous expression, so (8) must be interpreted as three different equations, for $\mu = x$, y and z. The optimum value is obtained for

$$\frac{\partial Q_\mu}{2\partial a_{rs}} = \sum_{k=1}^{K} \left(\mu_k - \sum_{i=0}^{I} \sum_{j=0}^{J} a_{ij}\phi_i(u_k)\psi_j(v_k) \right) \phi_r(u_k)\psi_s(v_k) = 0 \tag{9}$$

In this example, we consider two sets of data points: a first set of 64 data points (called the *training points* as they will be used to learn the neuron functions) with parametric coordinates uniformly distributed on the square $[0, 1] \times [0, 1]$

and a larger set of 256 data points generated in a similar way (called the *testing points*). To fit the training points, we have used Bernstein polynomials of degree I (resp. J) in u (resp. v) for the functions $\{\phi_i(u)\}_{i=0,1,\ldots,I}$ and $\{\psi_j(v)\}_{j=0,1,\ldots,J}$ in (8). Of course, every different choice for I and J yields a new system (9), which must be solved. In particular, we have taken values for I and J from 2 to 6. Solving the compatibility equations along with system (9) for all cases, we can compute the Bézier surface of degree (I, J) interpolating the given curves and that best approximate the data points. To test the quality of the model we have calculated the mean, maximum, minimum and the root mean squared (RMS) Euclidean errors for I and J from 2 to 6 for the 64 training data points. The best choice is for $I = J = 3$ that corresponds to the bicubic Bézier surface displayed in Fig. 4 (errors are as follows: maximum: 2.1457; minimum: 0.0052; mean: 0.3564; RMS: 0.0795). We also performed a cross validation of the model to check for overfitting. We have calculated the mean, the maximum and the root mean squared (RMS) errors of the 256 testing points. The final values are comparable, showing that no overfitting occurs. All computations have been carried out on a Pentium IV, 3 GHz. running Mathematica v5.2.

8 Conclusions and Future Work

In this paper we propose the use of two Artificial Intelligence techniques: (1) the curve/surface parameterization problem is performed by applying genetic algorithms; (2) the functional constraints problem is addressed by using functional networks. Both approaches are combined with the least-squares approximation method to yield suitable methods for curve and surface fitting. Some examples of their applications on 3D point clouds through Bézier curves and surfaces (including the case of cross-section constraints) are also given.

Our future work include the consideration of piecewise polynomials models like B-spline or NURBS for fitting the data, with some changes in the computational process for dealing with the knot vectors, which are other parameters in these models. The distance error function for fitting data to the different models seems to present a behaviour with multiple relative minima. This makes more difficult attaining a global optimum. Some ideas on how to improve globally the search process are also part of our future work.

The authors would like to thank the financial support from the SistIng-Alfa project, Ref: ALFA II-0321-FA of the European Union and the Spanish Ministry of Education and Science, Project Ref. #TIN2006-13615.

References

1. Barhak, J., Fischer, A.: Parameterization and reconstruction from 3D scattered points based on neural network and PDE techniques. IEEE Trans. on Visualization and Computer Graphics 7(1), 1–16 (2001)
2. Bradley, C., Vickers, G.W.: Free-form surface reconstruction for machine vision rapid prototyping. Optical Engineering 32(9), 2191–2200 (1993)

3. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 35(3), 268–308 (2003)
4. Castillo, E., Iglesias, A.: Some characterizations of families of surfaces using functional equations. ACM Transactions on Graphics 16(3), 296–318 (1997)
5. Castillo, E., Cobo, A., Gomez-Nesterkin, R., Hadi, A.S.: A general framework for functional networks. Networks 35(1), 70–82 (2000)
6. Castillo, E., Iglesias, A., Ruiz-Cobo, R.: Functional Equations in Applied Sciences. Elsevier Pub., Amsterdam (2005)
7. Echevarría, G., Iglesias, A., Gálvez, A.: Extending neural networks for B-spline surface reconstruction. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J.J., Hoekstra, A.G. (eds.) Computational Science - ICCS 2002. LNCS, vol. 2330, pp. 305–314. Springer, Heidelberg (2002)
8. Goldberg, D.E.: Optimal Initial Population Size for Binary-Coded Genetic Algorithms, TCGA Report No.85001. University of Alabama (1985)
9. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
10. Gu, P., Yan, X.: Neural network approach to the reconstruction of free-form surfaces for reverse engineering. Computer Aided Design 27(1), 59–64 (1995)
11. Hoffmann, M., Varady, L.: Free-form surfaces for scattered data by neural networks. J. Geometry and Graphics 2, 1–6 (1998)
12. Holland, J.H.: Adaptation in Natural and Artificial Systems. Michigan Press, Ann Arbor (1975)
13. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Surface reconstruction from unorganized points. In: Proc. of SIGGRAPH92, vol. 26(2), pp. 71–78 (1992)
14. Iglesias, A., Gálvez, A.: A New Artificial Intelligence Paradigm for Computer-Aided Geometric Design. In: Campbell, J.A., Roanes-Lozano, E. (eds.) AISC 2000. LNCS (LNAI), vol. 1930, pp. 200–213. Springer, Heidelberg (2001)
15. Iglesias, A., Echevarría, G., Gálvez, A.: Functional networks for B-spline surface reconstruction. Future Generation Computer Systems 20(8), 1337–1353 (2004)
16. Knopf, G.K., Kofman, J.: Free-form surface reconstruction using Bernstein basis function networks. In: Dagli, C.H., et al. (eds.) Intelligent Engineering Systems Through Artificial Neural Networks, vol. 9, pp. 797–802. ASME Press (1999)
17. Pottmann, H., et al.: Industrial geometry: recent advances and applications in CAD. Computer-Aided Design 37, 751–766 (2005)
18. Renner, G., Ekrt, A.: Genetic algorithms in computer aided design. Computer-Aided Design 35, 709–726 (2003)
19. Varady, T., Martin, R.: Reverse Engineering. In: Farin, G., Hoschek, J., Kim, M. (eds.) Handbook of Computer Aided Geometric Design. Elsevier Science, Amsterdam (2002)
20. Weiss, V., Andor, L., Renner, G., Varady, T.: Advanced surface fitting techniques. Computer Aided Geometric Design 19, 19–42 (2002)
21. Yoshimoto, F., Harada, T., Yoshimoto, Y.: Data fiting with a spline using a real-coded algorithm. Computer-Aided Design 35, 751–760 (2003)