Particle Swarm Optimization for Bézier Surface Reconstruction

Akemi Gálvez, Angel Cobo, Jaime Puig-Pey, and Andrés Iglesias

Department of Applied Mathematics and Computational Sciences,
University of Cantabria, Avda. de los Castros, s/n, E-39005, Santander, Spain
{galveza,acobo,puigpeyj,iglesias}@unican.es

Abstract. This work concerns the issue of surface reconstruction, that is, the generation of a surface from a given cloud of data points. Our approach is based on a metaheuristic algorithm, the so-called Particle Swarm Optimization. The paper describes its application to the case of Bézier surface reconstruction, for which the problem of obtaining a suitable parameterization of the data points has to be properly addressed. A simple but illustrative example is used to discuss the performance of the proposed method. An empirical discussion about the choice of the social and cognitive parameters for the PSO algorithm is also given.

1 Introduction

A major challenge in Computer Graphics nowadays is that of *surface reconstruction*. This problem can be formulated in many different ways, depending on the given input, the kind of surface involved and other additional constraints. The most common version in the literature consists of obtaining a smooth surface that approximates a given cloud of 3D data points accurately. This issue plays an important role in real problems such as construction of car bodies, ship hulls, airplane fuselages and other free-form objects. A typical example comes from Reverse Engineering where free-form curves and surfaces are extracted from clouds of points obtained through 3D laser scanning [5,11,12,17,18].

The usual models for surface reconstruction in Computer Aided Geometric Design (CAGD) are free-form parametric curves and surfaces, such as Bézier, Bspline and NURBS. This is also the approach followed in this paper. In particular, we consider the case of Bézier surfaces. In this case, the goal is to obtain the control points of the surface. This problem is far from being trivial: because the surface is 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 [2,10] 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.

Some recent papers have shown that the application of Artificial Intelligence (AI) techniques can achieve remarkable results regarding this parameterization problem [5,8,9,11,12,16]. Most of these methods rely on some kind of neural networks, either standard neural networks [8], Kohonen's SOM (Self-Organizing Maps) nets [1,9], 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 [9]. 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. The generalization to functional networks is also analyzed in [5,11,12]. A previous paper in [7] describes the application of genetic algorithms and functional networks yielding pretty good results for both curves and surfaces.

Our strategy for tackling the problem also belongs to this group of AI techniques. In this paper we address the application of the Particle Swarm Optimization method for Bézier surface reconstruction. Particle Swarm Optimization (PSO) is a popular metaheuristic technique with biological inspiration, used in CAM (Computer-Aided Manufacturing) for dealing with optimization of milling processes [6]. The original PSO algorithm was first reported in 1995 by James Kennedy and Russell C. Eberhart in [3,13]. In [4] some developments are presented. These authors integrate their contributions in [15]. See also [19].

The structure of this paper is as follows: the problem of surface reconstruction is briefly described in Section 2. Then, Section 3 describes the PSO procedure in detail. A simple yet illustrative example of its application is reported in Section 4. This section also discuss the problem of the adequate choice of the social and cognitive parameters of the PSO method by following an empirical approach. The main conclusions and further remarks in Section 5 close the paper.

2 Surface Reconstruction Problem

The problem of surface reconstruction can be stated as follows: *given a set of sample points X assumed to lie on an unknown surface U, construct a surface model S that approximates U.* This problem is generally addressed by means of the least-squares approximation scheme, a classical optimization technique that (given a series of measured data) attempts to find a function (the *fitness function*) which closely approximates the data. The typical approach is to assume that f has a particular functional structure which depends on some parameters that need to be calculated. In this work, we consider the case of f being a Bézier parametric surface $\mathbf{S}(u,v)$ of degree (M,N) whose representation is given by:

$$\mathbf{S}(u,v) = \sum_{i=0}^{M} \sum_{j=0}^{N} \mathbf{P}_{i,j} B_i^M(u) B_j^N(v) \tag{1}$$

where $B_i^M(u)$ and $B_j^N(v)$ are the classical Bernstein polynomials and the coefficients $\mathbf{P}_{i,j}$ are the surface control points. Given a set of 3D data points $\{\mathbf{D}_k\}_{k=1,\dots,n_k}$, we can compute, for each of the cartesian components, (x_k, y_k, z_k) of $\mathbf{D}_k$, the minimization of the sum of squared errors referred to the data points:

$$Err_\mu = \sum_{k=1}^{n_k} \left(\mu_k - \sum_{i=0}^{M} \sum_{j=0}^{N} P_{ij}^\mu B_i^M(u_k) B_j^N(v_k) \right)^2 \quad ; \quad \mu = x, y, z \qquad (2)$$

Coefficients $\mathbf{P}_{ij} = (P_{ij}^x, P_{ij}^y, P_{ij}^z)$, $i = 0, \dots, M$, $j = 0, \dots, N$, are to be determined from the information given by the data points (x_k, y_k, z_k), $k = 1, \dots, n_k$. 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. Note that, in addition to the coefficients of the basis functions, $\mathbf{P}_{ij}$, the parameter values, (u_k, v_k), $k = 1, \dots, n_k$, associated with the data points also appear as unknowns in our formulation. Due to the fact that the blending functions $B_i^M(u)$ and $B_j^N(v)$ are nonlinear in u and v respectively, 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.

3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic algorithm based on the evolution of populations for problem solving. PSO is a kind of *swarm intelligence*, a field in which systems are comprised by a collection of individuals exhibiting decentralized or collective behavior such that simple agents interact locally with one another and with their environment. Instead of a central behavior determining the evolution of the population, are these local interactions between agents which lead to the emergence of a global behavior for the swarm. A typical example of PSO is the behavior of a flock of birds when moving all together following a common tendency in their displacements. Other examples from nature include ant colonies, animal herding, and fish schooling.

In PSO the particle swarm simulates the social optimization commonly found in communities with a high degree of organization. For a given problem, some fitness function such as (2) is needed to evaluate the proposed solution. In order to get a good one, PSO methods incorporate both a global tendency for the movement of the set of individuals and local influences from neighbors [3,13]. PSO procedures start by choosing a population (swarm) of random candidate solutions in a multidimensional space, called *particles*. Then they are displaced throughout their domain looking for an optimum taking into account global and local influences, the latest coming form the neighborhood of each particle. To this purpose, all particles have a position and a velocity. These particles evolve all through the hyperspace according to two essential reasoning capabilities: a

Table 1. General structure of the particle swarm optimization algorithm

```
begin
    k=0
    random initialization of individual positions $P_i$ and velocities $V_i$ in Pop(k)
    fitness evaluation of Pop(k)
    while (not termination condition) do
        Calculate best fitness particle $P_g^b$
        for each particle $i$ in Pop(k) do
            Calculate particle position $P_i^b$ with best fitness
            Calculate velocity $V_i$ for particle $i$ according to (3)
            while not feasible $P_i + V_i$ do
                Apply scale factor to $V_i$
            end
            Update position $P_i$ according to (4)
        end
        $k = k + 1$
    end
end
```

memory of their own best position and knowledge of the global or their neighborhood's best. The meaning of the "best" must be understood in the context of the problem to be solved. In a minimization problem (like in this paper) that means the position with the smallest value for the target function.

The dynamics of the particle swarm is considered along successive iterations, like time instances. Each particle modifies its position P_i along the iterations, keeping track of its best position in the variables domain implied in the problem. This is made by storing for each particle the coordinates P_i^b associated with the best solution (fitness) it has achieved so far along with the corresponding fitness value, f_i^b. These values account for the *memory* of the best particle position. In addition, members of a swarm can communicate good positions to each other, so they can adjust their own position and velocity according to this information. To this purpose, we also collect the best fitness value among all the particles in the population, f_g^b, and its position P_g^b from the initial iteration. This is a global information for modifying the position of each particle. Finally, the evolution for each particle i is given by:

$$V_i(k+1) = w\,V_i(k) + \alpha R_1[P_g^b(k) - P_i(k)] + \beta R_2[P_i^b(k) - P_i(k)] \tag{3}$$

$$P_i(k+1) = P_i(k) + V_i(k) \tag{4}$$

where $P_i(k)$ and $V_i(k)$ are the position and the velocity of particle i at time k respectively, w is called *inertia weight* and decide how much the old velocity will affect the new one and coefficients α and β are constant values called *learning factors*, which decide the degree of affection of P_g^b and P_i^b. In particular, α is a weight that accounts for the "social" component, while β represents the "cognitive" component, accounting for the memory of an individual particle

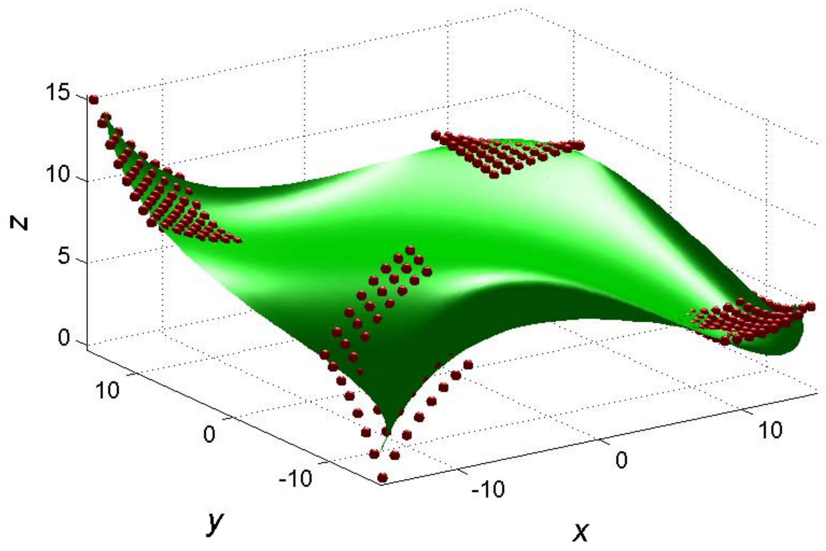

Fig. 1. Example of surface reconstruction through particle swarm optimization: reconstructed bicubic Bézier surface and data points

along the time. Two random numbers, R_1 and R_2, with uniform distribution on $[0, 1]$ are included to enrich the searching space. Finally, a fitness function must be given to evaluate the quality of a position. This procedure is repeated several times (thus yielding successive generations) until a termination condition is 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. The final PSO procedure is briefly sketched in Table 1.

4 An Illustrative Example

In this section we analyze a simple yet illustrative example aimed at showing the performance of the presented method. To this purpose, we consider an input of 256 data points generated from a Bézier surface as follows: for the u's and v's of data points, we choose two groups of 8 equidistant parameter values in the intervals $[0, 0.2]$ and $[0.8, 1]$. This gives a set of 256 3D data points. Our goal is to reconstruct the surface which such points come from. To do so, we consider a bicubic Bézier surface, so the unknowns are $3 \times 16 = 48$ scalar coefficients (3 coordinates for each of 16 control points) and two parameter vectors for u and v (each of size 16) associated with the 256 data points. That makes a total of 80 scalar unknowns. An exact solution (i.e. with zero value error) exists for this problem.

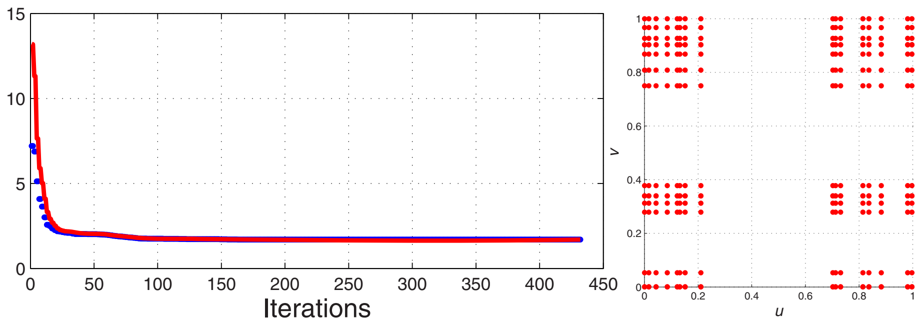

Fig. 2. Example of surface reconstruction through particle swarm optimization: (left) evolution of the mean (solid line) and the best (dotted line) Euclidean errors along the generations; (right): optimum parameter values for u and v on the parametric domain

The input parameter values for the PSO algorithm are: population size: 200 individuals or particles, where each particle is represented by two vectors, U and V, each with 16 components initialized with random uniform values on [0,1] sorted in increasing order; inertia coefficient $w = 1$. The termination criteria is that of not improving the solution after 30 consecutive iterations.

An example of reconstructed surface along with the data points is shown in Figure 1. This example does not correspond to the best solution (note that some points do not actually lie on the surface); it is just an average solution instead. It has been attained from eqs. (3)-(4) with $\alpha = \beta = 0.5$ at generation 432 with the following results[1]: best error in the fit: 1.6935426; mean error: 1.6935856; computation time: 56.73 seconds. All computations in this paper have been performed on a 2.4 GHz. Intel Core 2 Duo processor with 2 GB. of RAM. The source code has been implemented in the popular scientific program *Matlab*, version 7.0.

Fig. 2(left) displays 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, v) are depicted in Fig. 2(right) where one can see how the fitting process grasps the distribution of parameter values assigned to the data points. It is worthwhile to mention the tendency of the obtained parameter values, initially uniformly distributed on the unit square, to concentrate at the corners of such unit square parameter domain, thus adjusting well the input information. However, neither the couples (u, v) are uniformly distributed at the corners nor they fall within the intervals $[0, 0.2]$ and $[0.8, 1]$ for u and v, meaning that current results might actually be improved.

A critical issue in this method is the choice of the coefficients α and β accounting for the global (or social) and the local (or cognitive) influences, respectively. In order to determine their role in our approach and how their choice affects method's performance, we considered values for α and β ranging from 0.1 to 0.9

[1] For the sake of easier comparison, this case has been boldfaced in Table 2.

Table 2. Executions of PSO algorithm for our Bézier surface reconstruction problem. Cases (left-right, top-bottom): α from 0.9 to 0.4 with step 0.1, $\beta = 1 - \alpha$ in all cases.

$\alpha = 0.9, \beta = 0.1$

Best error	Mean error	# iter.	CPU time
1.1586464	1.1586805	338	48.49
1.6367439	1.6368298	624	85.03
2.3784306	2.3784327	736	93.27
1.8768595	1.8778174	350	46.26
2.1174907	2.1174907	917	131.72
1.1742370	1.1785017	145	20.97
2.0640768	2.0640795	503	60.19
2.2379692	2.2380810	302	46.43
2.1448603	2.1450512	443	56.22
2.0545017	2.0547538	408	52.48

$\alpha = 0.8, \beta = 0.2$

Best error	Mean error	# iter.	CPU time
2.1516721	2.1727290	820	151.74
2.5577003	2.5578876	308	46.45
2.0212768	2.0212868	431	69.06
1.8898777	1.8899836	455	69.23
2.0019422	2.003234	456	57.26
1.6520482	1.6520508	815	106.01
2.1574432	2.1574436	1540	205.35
2.4201197	2.4202493	822	100.09
2.0328183	2.0328913	587	72.24
2.2947584	2.2949567	3144	402.61

$\alpha = 0.7, \beta = 0.3$

Best error	Mean error	# iter.	CPU time
1.8639684	1.8653115	162	21.19
1.5992922	1.5993440	192	24.27
2.2059607	2.2059608	1340	149.33
2.3826529	2.3836276	185	22.42
2.0860826	2.0862052	400	49.38
1.5692097	1.5692186	967	174.42
1.6049119	1.6049300	470	58.12
1.3689993	1.3690215	292	39.17
1.6388899	1.6395055	458	50.81
1.7290016	1.7291297	389	41.83

$\alpha = 0.6, \beta = 0.4$

Best error	Mean error	# iter.	CPU time
1.9953687	1.9959765	379	44.73
2.1917047	2.1921891	289	39.14
1.2152328	1.2152527	382	49.42
1.9617652	1.9623143	303	43.27
1.2548267	1.2548387	1222	143.48
1.8748061	1.8752081	407	52.23
1.9507635	1.9509406	363	40.67
2.0454719	2.0464246	692	82.73
1.3580824	1.3583463	212	25.62
1.9017035	1.9017552	791	92.15

$\alpha = 0.5, \beta = 0.5$

Best error	Mean error	# iter.	CPU time
1.0799138	1.0805757	408	52.47
1.7608284	1.7608294	808	102.93
1.8697185	1.8697928	809	104.86
1.6935426	**1.6935856**	**432**	**56.73**
1.2815625	1.2815865	495	61.72
2.1078771	2.1079752	401	61.96
1.7415515	1.7415516	574	78.96
1.6556435	1.6556464	1083	143.02
2.0329562	2.0329594	1286	172.42
1.0632575	1.0632593	413	54.67

$\alpha = 0.4, \beta = 0.6$

Best error	Mean error	# iter.	CPU time
0.9491048	0.9492409	382	45.20
1.7165179	1.7165739	1573	178.12
1.3993802	1.3993921	466	58.42
1.1001050	1.1001427	720	104.10
1.2968360	1.2968360	1236	157.12
0.9909381	0.9909412	575	73.24
1.4642326	1.4642397	781	89.32
1.6312540	1.6312592	619	74.42
1.4394767	1.4394768	665	83.01
1.4422279	1.4422380	784	91.03

with step 0.1 in the way of a convex combination, i.e., $\alpha + \beta = 1$. This choice allows us to associate the values of the couple (α, β) with a probability, so that their interplay can be better analyzed and understood. Note that the limit values 0 and 1 for any parameter α or β automatically discards the other one so they are not considered in our study. Furthermore, in [14] some experiments for the two extreme cases, social-only model and cognitive-only model, were accomplished

Table 3. Executions of PSO algorithm for our Bézier surface reconstruction problem. Cases: $\alpha = 0.3$ (top-left), $\alpha = 0.2$ (top-right), $\alpha = 0.1$ (bottom) $\beta = 1 - \alpha$ in all cases.

$\alpha = 0.3, \beta = 0.7$				$\alpha = 0.2, \beta = 0.8$			
Best error	Mean error	# iter.	CPU time	Best error	Mean error	# iter.	CPU time
2.1435684	2.1435684	974	109.00	1.866938	1.867126	461	61.99
1.8060138	1.8060836	531	57.48	0.9551159	0.9557403	290	37.26
2.2339992	2.2339993	1159	129.07	2.0777061	2.0777782	940	113.08
2.0832623	2.0832632	3781	387.39	2.0558802	2.0559779	807	96.07
1.6257242	1.6257283	748	83.71	1.6975429	1.6975450	1330	163.06
1.6742073	1.6742091	831	104.53	1.9514682	1.9514725	1405	177.13
2.2355623	2.2356626	1262	134.60	1.8397214	1.8397337	898	107.12
2.0420308	2.0420326	510	64.98	1.8298951	1.8298951	1297	165.25
2.2102381	2.2102381	741	82.48	2.0990000	2.0990008	905	100.92
2.2140913	2.2141552	2168	243.69	1.5363575	1.5363722	766	92.41

$\alpha = 0.1, \beta = 0.9$							
Best error	Mean error	# iter.	CPU time	Best error	Mean error	# iter.	CPU time
1.6872942	1.6873196	1885	197.43	1.6657973	1.6658058	1377	156.11
1.8395049	1.8395169	1793	194.67	1.8360591	1.8360592	3918	412.28
1.3436841	1.3436841	1887	214.16	1.4325258	1.4325283	751	99.35
1.7886651	1.7889720	561	67.38	0.9387540	0.9387899	814	89.28
1.0500875	1.0501339	1442	154.02	0.9643215	0.9642876	798	84.32

and the author found out that both parts are essential to the success of PSO. On the other hand, to overcome the randomness inherent in our method, we carried out 25 executions for each choice of these parameters. The 15 worst results were then removed to prevent the appearance of spurious solutions leading to local minima. The remaining 10 executions are collected in Tables 2 and 3.

Columns of these tables show the best and mean errors, the number of iterations and the computation time (in seconds) respectively. Note that the best and mean errors take extremely close (although slightly different) values, with differences of order 10^{-5} in most cases. Note also that the number of iterations (and hence the computation time) varies a lot among different executions. However, larger number of iterations do not imply, in general, lower errors.

5 Conclusions and Future Work

In this paper we consider the problem of the reconstruction of a Bézier surface from a set of 3D data points. The major problem here is to obtain a suitable parameterization of the data points. To this aim, we propose the use of the PSO algorithm that is briefly described in this paper. The performance of this method is discussed by means of a simple example.

In general, the PSO performs well for the given problem. Errors typically fall within the interval $[0.9, 2.6]$ in our executions, although upper (and possibly lower) values can also be obtained. This means that the present method compares well with the genetic algorithms approach reported in [7], although the PSO seems to yield more scattered output throughout the output domain. Other remarkable feature is that the best and mean errors are very close each other for all cases, as opposed to the genetic algorithms case, where the differences are generally larger. On the other hand, there is not correlation between the number of iterations and the quality of the results. This might mean that our way of exploring the space domain of the problem is not optimal yet, and consequently there is room for further improvement. This can be achieved by a smart choice of the PSO parameters. As a first step, we performed an empirical analysis about the choice of the (α, β) parameters of the PSO (although other parameters such as the initial population and the number of neighbors for each particle are also relevant). The results show that there is no significant differences when changing the (α, β) values in the way of a convex combination, although the condition $\beta \geq \alpha$ seems to achieve slightly better results. However, further research is still needed in order to determine the role of the parameter values at full extent.

Our future work include further analysis about the influence of the PSO parameters on the quality of the results. Some modifications of the original PSO scheme might lead to better results. Other future work is the consideration of piecewise polynomials models like B-spline or NURBS, which introduce some changes in the computational process for dealing with the knot vectors (that are other parameters to be taken into account in these models). Some ideas on how to improve globally the search process are also part of our future work.

Acknowledgments. The authors 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, National Program of Computer Science, Project Ref. TIN2006-13615 and National Program of Mathematics, Project Ref. MTM2005-00287.

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. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, pp. 39–43 (1995)
4. Eberhart, R.C., Shi, Y.: Particle swarm optimization: developments, applications and resources. In: Proceedings of the 2001 Congress on Evolutionary Computation, pp. 81–86 (2001)
5. 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., Hoekstra, A.G. (eds.) ICCS-ComputSci 2002. LNCS, vol. 2330, pp. 305–314. Springer, Heidelberg (2002)

6. El-Mounayri, H., Kishawy, H., Tandon, V.: Optimized CNC end-milling: a practical approach. International Journal of Computer Integrated Manufacturing 15(5), 453–470 (2002)
7. Gálvez, A., Iglesias, A., Cobo, A., Puig-Pey, J., Espinola, J.: Bézier curve and surface fitting of 3D point clouds through genetic algorithms, functional networks and least-squares approximation. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007, Part II. LNCS, vol. 4706, pp. 680–693. Springer, Heidelberg (2007)
8. 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)
9. Hoffmann, M., Varady, L.: Free-form surfaces for scattered data by neural networks. J. Geometry and Graphics 2, 1–6 (1998)
10. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Surface reconstruction from unorganized points. In: Proc. of SIGGRAPH 1992, vol. 26(2), pp. 71–78 (1992)
11. 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)
12. Iglesias, A., Echevarría, G., Gálvez, A.: Functional networks for B-spline surface reconstruction. Future Generation Computer Systems 20(8), 1337–1353 (2004)
13. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942–1948 (1995)
14. Kennedy, J.: The particle swarm: social adaptation of knowledge. In: IEEE International Conference on Evolutionary Computation, Indianapolis, Indiana, USA, pp. 303–308 (1997)
15. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco (2001)
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., Leopoldseder, S., Hofer, M., Steiner, T., Wang, W.: Industrial geometry: recent advances and applications in CAD. Computer-Aided Design 37, 751–766 (2005)
18. Varady, T., Martin, R.: Reverse Engineering. In: Farin, G., Hoschek, J., Kim, M. (eds.) Handbook of Computer Aided Geometric Design. Elsevier, Amsterdam (2002)
19. Vaz, I.F., Vicente, L.N.: A particle swarm pattern search method for bound constrained global optimization. Journal of Global Optimization 39, 197–219 (2007)
20. Weiss, V., Andor, L., Renner, G., Varady, T.: Advanced surface fitting techniques. Computer Aided Geometric Design 19, 19–42 (2002)