Computer Aided Geometric Design 29 (2012) 448–462

Contents lists available at SciVerse ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

Fast B-spline curve fitting by L-BFGS

Wenni Zheng [a,*], Pengbo Bo [a,b], Yang Liu [c], Wenping Wang [a]

[a] The University of Hong Kong, China
[b] Harbin Institute of Technology at Weihai, China
[c] Microsoft Research Asia, China

ARTICLE INFO

Article history:
Available online 10 April 2012

Keywords:
L-BFGS
Curve fitting
B-spline curve
Point cloud
Quasi-Newton method

ABSTRACT

We propose a fast method for fitting planar B-spline curves to unorganized data points. In traditional methods, optimization of control points and foot points are performed in two alternating time-consuming steps in every iteration: 1) control points are updated by setting up and solving a linear system of equations; and 2) foot points are computed by projecting each data point onto a B-spline curve. Our method uses the L-BFGS optimization method to optimize control points and foot points simultaneously and therefore it does not need to solve a linear system of equations or performing foot point projection in every iteration. As a result, the proposed method is much faster than existing methods.

1. Introduction

Curve fitting is a fundamental problem in many fields, such as computer graphics, image processing, shape modeling and data mining. Depending on applications, different types of curves such as parametric curves, implicit curves and subdivision curves are used for fitting. In this paper, we study the problem of fitting planar B-spline curves to unorganized data points.

Given a set of unorganized data points $\{\mathbf{X}_i\}_{i=1}^N \subset \mathbb{R}^2$ sampled from the outline of a planar shape, the aim of curve fitting is to find a B-spline curve $\mathbf{P}(t) = \sum_{i=1}^n \mathbf{P}_i N_i(t)$ that best approximates the shape's outline. The outline is called a *target shape*, and the B-spline curve is called a *fitting curve*. Here, $\mathcal{P} := \{\mathbf{P}_i\}_{i=1}^n \subset \mathbb{R}^2$ is the set of B-spline control points, $\{N_i(t)\}_{i=1}^n$ are B-spline basis functions. We suppose that knots of the B-spline curve are fixed and therefore not subject to optimization, and all the basis functions are thus defined on fixed uniform knot vectors throughout the curve fitting process.

For a data point $\mathbf{X}_k$, let $\mathbf{P}(t_k)$ denote the nearest point of $\mathbf{X}_k$ on the fitting curve. Then, the distance between data point $\mathbf{X}_k$ and the fitting curve is $\|\mathbf{P}(t_k) - \mathbf{X}_k\|$. Here, t_k is called the *location parameter* of $\mathbf{X}_k$, $\mathbf{P}(t_k)$ is called the *foot point* corresponding to $\mathbf{X}_k$. Denote $\mathcal{T} := \{t_1, \ldots, t_N\}$, i.e. the collection of the location parameters of all the data points. The fitting problem is then formulated as:

$$\min_{\mathcal{P},\mathcal{T}} \frac{1}{2} \sum_{k=1}^{N} \left\| \mathbf{P}(\mathcal{P}; t_k) - \mathbf{X}_k \right\|^2 + F_{fairing}, \tag{1}$$

* Corresponding author.
E-mail address: wnzheng@cs.hku.hk (W. Zheng).

0167-8396/$ – see front matter
http://dx.doi.org/10.1016/j.cagd.2012.03.004

where $F_{fairing}$ is a fairing term which defines the fairness of a curve. $F_{fairing}$ is commonly defined as follows (Wang et al., 2006):

$$F_{fairing} = \alpha \int_0^1 \left\| \mathbf{P}'(t) \right\|^2 dt + \beta \int_0^1 \left\| \mathbf{P}''(t) \right\|^2 dt. \tag{2}$$

Since the objective function in Eq. (1) is nonlinear, it is natural to apply iterative minimization methods to solve it. Most prevailing methods for solving this problem in CAGD are not standard optimization methods in the sense that they separately optimize location parameters $\mathcal{T}$ and control points $\mathcal{P}$, making the problem much simpler to handle. However, these methods are time-consuming because they need to compute foot points on the fitting curve and to formulate and solve a linear system of equations in every iteration. Our contribution is the observation that these time-consuming operations can be avoided by employing a L-BFGS optimization method that solves $\mathcal{T}$ and $\mathcal{P}$ simultaneously. We show that the resulting algorithm is very efficient because in every iteration it does not need to perform foot point projection or solve a linear system of equations.

The remainder of this paper is organized as follows. In Section 2, we review some previous work. Section 3 introduces the standard L-BFGS optimization method. Section 4 presents our new algorithm. Section 5 shows experimental results and comparisons with existing methods. Then we conclude the paper in Section 6 with discussions of future work.

2. Related work

Problem (1) can also be formulated as a nonlinear constrained minimization problem with unknown variables $\mathcal{P}$:

$$\min_{\mathcal{P}} \frac{1}{2} \sum_{k=1}^{N} \left\| \mathbf{P}(t_k) - \mathbf{X}_k \right\|^2 + F_{fairing}, \tag{3}$$

where each t_k is chosen such that $\mathbf{P}(t_k)$ is the foot point of $\mathbf{X}_k$ and thus satisfies:

$$\left(\mathbf{P}(t_k) - \mathbf{X}_k\right)^T \mathbf{P}'(t_k) = 0, \quad k = 1, \ldots, N. \tag{4}$$

By representing $\{t_k\}_{k=1}^N$ as functions of $\mathcal{P}$ and putting them into the objective function in Eq. (3), we obtain a nonlinear unconstrained minimization problem of variables $\mathcal{P}$:

$$\min_{\mathcal{P}} \frac{1}{2} \sum_{k=1}^{N} \left\| \mathbf{P}\big(t_k(\mathcal{P})\big) - \mathbf{X}_k \right\|^2 + F_{fairing}. \tag{5}$$

This is the viewpoint taken in Liu and Wang (2008) that reveals inherent relationship between some traditional methods and standard optimization techniques. Most methods for solving problem (5) deal with control points and foot points separately in an alternating manner (Hoschek et al., 1989; Liu and Wang, 2008). Every iteration of these methods consists of the following two steps:

Step 1: *Foot point projection*: Fixing the control points of the current fitting curve, compute the location parameters $\mathcal{T} = \{t_k\}_{k=1}^N$ for the data points $\{\mathbf{X}_k\}_{k=1}^N$ such that $\{\mathbf{P}(t_k)\}_{k=1}^N$ are the foot points of $\{\mathbf{X}_k\}_{k=1}^N$ on the current fitting curve. This step preserves the orthogonality constraint in Eq. (4).

Step 2: *Control point update*: In this step, $\mathcal{T}$ is fixed and a quadratic function e_k in terms of the control points $\mathcal{P}$ is used to approximate the nonlinear squared distance from a data point $\mathbf{X}_k$ to the fitting curve. Then the control points $\mathcal{P}$ are computed by minimizing the quadratic objective function $Q(\mathcal{P}) = \sum_k e_k(\mathcal{P}) + F_{fairing}$. Since both e_k and $F_{fairing}$ are quadratic functions of $\mathcal{P}$, this step entails solving the linear equations $\nabla Q(\mathcal{P}) = 0$.

Depending on different quadratic approximations chosen for e_k, there are mainly three kinds of existing optimization methods for curve fitting. The first one is the *Point Distance Minimization* method, or PDM. This method is widely used because of its simplicity. References on PDM include (but are not limited to) Pavlidis (1983), Plass and Stone (1983), Hoschek (1988) and Saux and Daniel (2003) on curve fitting as well as Hoppe et al. (1994), Forsey and Bartels (1995) and Haber et al. (2001) on surface fitting. The error term used in PDM is defined by

$$e_{PD,k} = \left\| \mathbf{P}(\mathcal{P}; t_k) - \mathbf{X}_k \right\|^2. \tag{6}$$

Geometrically, this function defines the distance between a data point and a point on the fitting curve at a particular parameter t_k. Considering the fact that $t_k(\mathcal{P})$ is set to a constant, this definition is a poor approximation of the nonlinear distance in Eq. (5). As pointed out in Björck (1996), from the viewpoint of optimization, PDM is an alternating method and exhibits linear convergence rate. We will see in our experiments that PDM is the slowest among all the methods we have tested.

The second method is called the *tangent distance minimization* method (TDM) (Blake and Isard, 1998) which uses the error term

$$e_{TD,k} = \left[\left(\mathbf{P}(\mathcal{P}; t_k) - \mathbf{X}_k\right)^T \cdot \mathbf{N}_k\right]^2, \tag{7}$$

where $\mathbf{N}_k$ is the unit normal vector at point $\mathbf{P}(t_k)$ on the curve.

The term $e_{TD,k}$ defines the distance between a data point $\mathbf{X}_k$ and the tangent line at $\mathbf{P}(t_k)$. Although this is a fair approximation to the true squared distance near a flat part of curve, it is not accurate near high curvature regions since no curvature information is considered. As a result, TDM does not show stable performance near high curvature regions (Wang et al., 2006). In fact, it has been pointed out in Wang et al. (2006) that TDM is essentially Gauss–Newton minimization without step-size control, and regularization should be used to improve the stability of TDM.

Applying the Levenberg–Marquardt regularization to TDM leads to a method called TDMLM (Wang et al., 2006). Suppose the linear system for control points updating in TDM is $A_{TDM} \cdot \mathcal{P} = \mathbf{b}_{TDM}$, where A_{TDM} is a matrix and $\mathbf{b}_{TDM}$ is a vector. In TDMLM, the control points $\mathcal{P}$ are computed by solving

$$(A_{TDM} + \mu I) \cdot \mathcal{P} = \mathbf{b}_{TDM}.$$

Empirically, μ is set as $\mu = \frac{tr(A_{TDM})}{80n}$, where $tr(A_{TDM})$ is the trace of A_{TDM}, n the number of control points, and I the identity matrix.

The third method, called the *Squared Distance Minimization* method or SDM (Wang et al., 2006), uses a curvature-based error term, which is a variant of the second order approximation to the true squared distance introduced in Pottmann et al. (2002), Pottmann and Hofer (2003). This error term, called the *SD error term*, is defined by

$$e_{SD,k} = \begin{cases} \frac{d}{d-\rho}[(\mathbf{P}(\mathcal{P}; t_k) - \mathbf{X}_k)^T \cdot T_k]^2 + [(\mathbf{P}(\mathcal{P}; t_k) - \mathbf{X}_k)^T \cdot N_k]^2, & \text{if } d < 0, \\ [(\mathbf{P}(\mathcal{P}; t_k) - \mathbf{X}_k)^T \cdot N_k]^2, & \text{if } 0 \leqslant d < \rho, \end{cases} \tag{8}$$

where ρ is the curvature radius at $\mathbf{P}(t_k)$ and d is the positive distance between $\mathbf{X}_k$ and $\mathbf{P}(t_k)$. The SD error term contains some second order derivative information and is therefore a better approximation to the true squared distance function than those used in TDM and PDM. From the viewpoint of optimization, SDM is a Newton-like optimization method that employs a modified Hessian matrix of the original nonlinear distance function. This modification discards some complicated parts in the true Hessian matrix and keeps other parts with intuitive geometric meanings (Wang et al., 2006). The semi-definite positive property of the modified Hessian matrix is also guaranteed. It has been demonstrated in Wang et al. (2006) that SDM exhibits better performance in terms of convergence rate and stability than PDM and TDM.

Since the curve fitting problem is formulated as a nonlinear least squares minimization problem in Eq. (1), it is natural to study how to solve it using standard optimization methods. The authors of Liu and Wang (2008) apply the Gauss–Newton method to Eq. (1) and derive new error terms using simplified partial derivatives of the objective function in Eq. (1). These methods are observed to have similar performances as SDM.

All the above methods update control points $\mathcal{P}$ and location parameters $\{t_k\}_{k=1}^N$ in two interleaving steps. The main difference of our new method with these existing methods is that in every iteration we update $\mathcal{P}$ and $\{t_k\}_{k=1}^N$ simultaneously. In this sense the most closely related work is (Speer et al., 1998) which also optimizes control points and location parameters simultaneously in every iteration. However, that method uses the Gauss–Newton optimization and therefore still needs to valuate and store the Jacobian matrices of the objective function, whose size depends on the number of data points and control points (Speer et al., 1998), as well as to solve a linear system of equations. In contrast, our approach based on L-BFGS does not need to formulate and solve any linear equations and is therefore faster than the method in Speer et al. (1998), as we are going to demonstrate in later experiments.

Other optimization techniques have been explored for surface and curve fitting problems in literature. The authors of Xie and Qin (2001) proposed a method for NURBS curve and surface fitting which optimizes control points, parameters and knots by a conjugate gradient method. Genetic Algorithms and optimal control methods have also been tried in curve fitting (Sarfraz et al., 2006; Alhanaty and Bercovier, 2001). These methods are generally slow and have only been applied to simple examples.

3. Limited memory BFGS – L-BFGS

Limited Memory BFGS, or L-BFGS, is a quasi-Newton method for solving unconstrained nonlinear minimization problems (Nocedal and Wright, 1999). L-BFGS approximates the inverse Hessian matrix of the objective function by a sequence of gradient vectors from previous iterations. Suppose we want to solve an unconstrained optimization problem

$$\min_x f(x),$$

where $f(x)$ is a nonlinear function to minimize and x a set of unknown variables. In the k-th iteration of L-BFGS, the variables x_{k+1} are updated by

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k),$$

where H_k is an approximation to the inverse Hessian matrix of $f(x)$ at x_k. Here, $-H_k \nabla f(x_k)$ is a search direction, and α_k a scalar variable controlling the step-size of search direction (Nocedal and Wright, 1999).

Define $s_k := x_{k+1} - x_k$, $y_k := \nabla f_{k+1} - \nabla f_k$, $\rho_k := \frac{1}{y_k^T s_k}$, $V_k := I - \rho_k y_k s_k^T$. L-BFGS uses the values of the objective function and its gradient in the $(k-m)$-th iteration through $(k-1)$-th iteration to compute H_k (Nocedal and Wright, 1999):

$$\begin{aligned} H_k = {} & \left(V_{k-1}^T \cdots V_{k-m}^T\right) H_k^0 (V_{k-m} \cdots V_{k-1}) \\ & + \rho_{k-m}\left(V_{k-1}^T \cdots V_{k-m+1}^T\right) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_{k-1}) \\ & + \rho_{k-m+1}\left(V_{k-1}^T \cdots V_{k-m+2}^T\right) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_{k-1}) + \cdots \\ & + \rho_{k-1} s_{k-1} s_{k-1}^T, \end{aligned} \tag{9}$$

where H_k^0 is a diagonal matrix defined by $H_k^0 := \gamma_k I$, where $\gamma_k := \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$ (Nocedal and Wright, 1999).

In practice, we do not need to compute and store the matrix H_k. Instead, we compute the search direction $-H_k \nabla f_k$ directly by a *L-BFGS two-loop recursion* algorithm (Algorithm 1) (Nocedal and Wright, 1999):

Algorithm 1 L-BFGS two-loop recursion

$q = \nabla f(x_k)$;
for $i = k-1, k-2, \ldots, k-m$ **do**
  $\alpha_i = \rho_i s_i^T q$;
  $q = q - \alpha_i y_i$;
end for
$z = H_k^0 q$;
for $i = k-m, k-m+1, \ldots, k-1$ **do**
  $\beta_i = \rho_i y_i^T z$;
  $z = z + s_i(\alpha_i - \beta_i)$;
end for
Output z to be $H_k \nabla f_k$.

The L-BFGS optimization procedure is described in Algorithm 2 (Nocedal and Wright, 1999).

Algorithm 2 the L-BFGS algorithm

Choose a starting point x_0 and a positive integer m;
$k = 0$;
repeat
  Choose H_k^0;
  Compute a descent direction p_k by the two-loop recursion algorithm;
  Compute $x_{k+1} = x_k + \alpha_k p_k$, α_k is chosen to satisfy the Wolfe conditions;
  if $k > m$ **then**
    Discard s_{k-m} and y_{k-m};
  end if
  Compute the values of s_k, y_k and store them;
until convergence

L-BFGS stops when the norm of the gradient of the objective function is smaller than a specified tolerance value ϵ, i.e. $\|\nabla f\| < \epsilon$.

In Algorithm 2, once the descent direction p_k is obtained, the variables x_{k+1} should be updated by $x_{k+1} = x_k + \alpha_k p_k$. Here α_k is chosen to guarantee the decreasing of the value of the objective function. This is usually solved by a linesearch algorithm. Basically, a linesearch algorithm starts with $\alpha_k = 1$ and decreases the value of α_k by some strategy until the following *Wolfe conditions* are satisfied (Nocedal and Wright, 1999):

$$\begin{cases} f(x_k + \alpha_k p_k) \leqslant f(x_k) + c_1 \alpha_k \nabla f_k^T p_k, \\ \nabla f(x_k + \alpha_k p_k)^T p_k \geqslant c_2 \nabla f_k^T p_k. \end{cases}$$

Here c_1 and c_2 are constants which satisfy $0 < c_1 < c_2 < 1$. In our algorithm we use $c_1 = 10^{-4}$ and $c_2 = 0.9$ throughout optimization. We have found from our experience that $\alpha_k = 1$ is often good enough and the computational time of linesearch only takes a small partition of the total computational time. We will show these timing data in later sections.

4. Curve fitting with L-BFGS

4.1. Algorithm outline

We employ L-BFGS directly to solve the nonlinear least squares minimization problem defined in Eq. (1). Our algorithm is described in the following steps.

1. Specify an initial curve $\mathbf{P}(t)$.
2. Find the foot point $\mathbf{P}(t_k)$ on $\mathbf{P}(t)$ for every data point $\mathbf{X}_k$. This gives the initial position of location parameters $\{t_k\}_{k=1}^N$.
3. Run Algorithm 2: the L-BFGS algorithm until convergence.

It should be noticed that $\mathcal{P}$ and $\mathcal{T}$ are independent variables in an unconstrained problem defined in Eq. (1), and will be solved simultaneously by the L-BFGS algorithm. We will refer to this algorithm as the *L-BFGS fitting method* in the following sections. It is easy to see that, when the L-BFGS algorithm converges, it indeed solves the constraint minimization problem defined in Eq. (3), since at a local minimum of the objective function in Eq. (1), the conditions in Eq. (4) are satisfied.

4.2. Selection of m

The L-BFGS algorithm (Algorithm 2) uses m gradient vectors in a sequence of iterations to approximate the inverse Hessian matrix. A larger m can result in a more accurate approximation but at the same time, it will take more computational time. Therefore, it is important to select a proper value of m which balances between the competing objectives, i.e., accuracy and efficiency. In literature, the value of m is often chosen between 3 and 20 (Nocedal and Wright, 1999). However there are also papers reporting that a very large value of m is necessary for generating satisfactory results. One example is (Jiang et al., 2004) in which m is set to 240.

To find a proper value of m in our curve fitting problem, we have tested many examples with $m = 3$, 5, 7, 20, 50 and 120 to understand the behavior of our algorithm. Our conclusion is, for simple examples, using $m = 3$ and $m = 5$ may lead to a slightly faster fitting error descending speed. On the other hand, using $m = 20$, 50 and 120 will contribute to a faster gradient norm convergence speed. For more complicated examples (examples with more data points and heavier noise), the behaviors of error descending speed and gradient norm descending speed using different m tend to be indistinguishable. Considering all these factors, we suggest using $m = 20$ in experiments.

4.3. Foot point projection

Computing foot points needs to be performed in every iteration of traditional curve fitting methods. Foot point projection is itself an optimization problem which is investigated in Hoschek et al. (1989), Saux and Daniel (2003), Aigner and Jüttler (2005) and Hu and Wallner (2005):

$$\min_{t_k} \left\| \mathbf{P}(t_k) - \mathbf{X}_k \right\|^2.$$

When implementing the traditional methods we compute foot points in every iteration using the Gauss–Newton method outlined below.

1. *Initialization*: In this step location parameters corresponding to data points are roughly estimated. The estimation will be used as initialization for further optimization in the next step. A straightforward method is: Sampling dense enough points on the fitting curve and finding the closest sample to each data point as the estimation of foot points.

2. *Iterative Update*: Update the location parameter iteratively (Wang et al., 2006): in the i-th iteration, the location parameter of $\mathbf{X}_k$ is updated by $t_{k,i+1} = t_{k,i} + a\delta t$, where

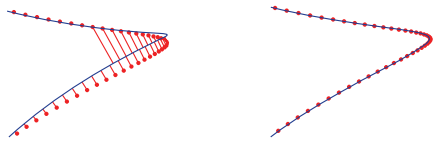
$$\delta t = \frac{(\mathbf{X}_k - \mathbf{P}(t_{k,i})) \cdot \mathbf{P}'(t_{k,i})}{\|\mathbf{P}'(t_{k,i})\|^2}$$

is the suggested update in the descent direction. The value of a is decided by a simple linesearch method to guarantee the decreasing of the orthogonal distance, i.e., $\|\mathbf{P}(t_{k,i+1}) - \mathbf{X}_k\| < \|\mathbf{P}(t_{k,i}) - \mathbf{X}_k\|$. The optimization process stops when $\|(\mathbf{X}_k - \mathbf{P}(t_{k,i})) \cdot \mathbf{P}'(t_{k,i})\| < 10^{-10}$.

The *Initialization* step is generally time consuming. In practice, after several iterations in the beginning of curve fitting, the shape of the fitting curve will not change a lot in later optimization. In this case, we can use foot points computed in the i-th iteration (i.e., $\{t_{k,i}\}$) as initialization foot points for the $(i+1)$-th iteration. To determine whether it is safe to directly use the foot points from the last iteration as initialization, we use a criterion based on variation of fitting error: We suggest to re-initialize foot points in the $(i+1)$-th iteration when $\frac{|E_{i+1} - E_i|}{E_{i+1}} > 0.2$ where E_i and E_{i+1} are the fitting error in the i-th iteration and $(i+1)$-th iteration respectively, as defined in Eq. (10). In our experiments, with this criterion, for all traditional methods, the number of foot point initializations needed is from 1 to 4 in all our tested examples.

4.4. Foot point correction and restart of L-BFGS

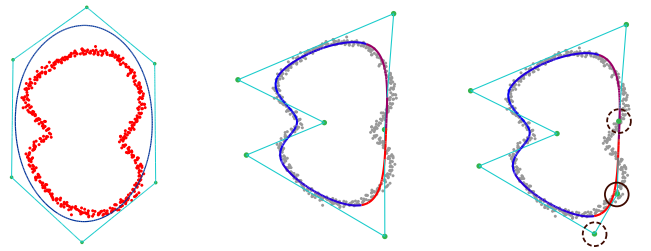
The initialization of the L-BFGS fitting method also needs foot point projection to determine the initial set $\mathcal{T}$ of location parameters. Then, in the subsequent iterations the L-BFGS fitting method in general does not need to perform foot point projection any more but rather optimizes location parameters $\{t_k\}_{k=1}^N$ and control points $\mathcal{P}$ simultaneously. In some rare cases, especially when the initial fitting curve specified is not good enough, it is possible that $\mathbf{P}(t_k)$ is far from the closet point on the curve to $\mathbf{X}_k$, even if $\mathbf{P}(t_k) - \mathbf{X}_k$ is orthogonal to $\mathbf{P}'(t_k)$. An example is shown in Fig. 1(a): it is part of a fitting



(a) Without foot point correction.

(b) With foot point correction.

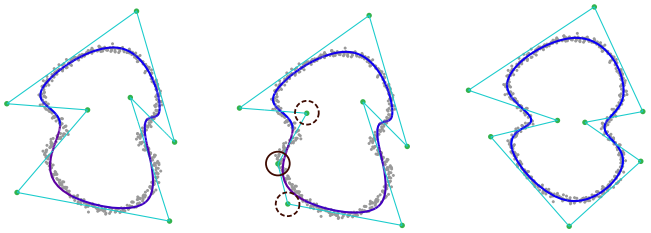
Fig. 1. Foot point correction.



(a) Initialization.

(b) L-BFGS algorithm converged.

(c) Adding a control point.



(d) L-BFGS algorithm restarted and converged.

(e) Adding another control point.

(f) Final result.

Fig. 2. Addition of control points.

curve on convergence, but there are points $\mathbf{P}(t_k)$ which are not the foot points of $\mathbf{X}_k$, because the L-BFGS fitting method gets stuck in a poor local minimum. In this case the following remedy can be used.

To rectify the incorrect foot point projections, we just perform foot point computation after the termination of current L-BFGS algorithm, and start a new L-BFGS algorithm taking initial control points from the previous run of L-BFGS and initial location parameters from the output of foot point computation. Fig. 1(b) shows the result fitting curve. To detect a local minimum automatically, we measure the fitting error E after the L-BFGS algorithm and the fitting error E_+ after the additional operation of foot point computation. If the error changing $\|E - E_+\|$ is bigger than a tolerance (we use 10^{-6} in our experiments), we decide that the foot points of some data points need to be corrected and an additional run of the L-BFGS algorithm is needed.

4.5. The initial curve and insertion of control points

An automatic approach to generating an initial fitting curve with good quality is beyond the scope of the present paper. In the experiments for comparing the convergence rates of different curve fitting algorithms, we generate the initial curve by hand. This is enough for our purpose as long as the initial curves are the same for different methods in one example. In practice, an automatic curve fitting approach may start with an initial curve with a small number of control points and insert new control points at the regions with big fitting error. We give such an example in Fig. 2. When a new control point is inserted, we need to restart the L-BFGS fitting algorithm.

5. Results and discussions

In this section, we first present some experiments comparing the L-BFGS fitting method with existing methods, then we give explanation on the fast speed of the L-BFGS fitting method.

5.1. Experiments

The fitting error of the i-th iteration is measured by:

$$E_i := \left(\sum_{k=1}^{N} \frac{1}{N} \left\| \mathbf{P}(t_k) - \mathbf{X}_k \right\|^2 \right)^{\frac{1}{2}}. \tag{10}$$

The parameter domains of B-spline curves in these examples are set to $[0, 1]$. All data points are scaled into a unit box: $[0, 1] \times [0, 1]$.

Due to different complexities and the set up of initial curves in the examples in our experiments, we use different coefficients of fairing terms α and β based on experience in different examples to obtain satisfactory fitting curves. In each example, the same values of faring term coefficients are used for all tested methods. The values of coefficients are noted in the captions of figures.

Comparison with traditional methods. Three data sets are given in Figs. 3, 4 and 5 for comparisons with three traditional methods: PDM, TDMLM and SDM. For each data set, we show data points, the initial fitting curve and the final fitting curve of the L-BFGS fitting method. Three charts are also shown for each data set. The first two charts show the fitting error versus the iteration number and computational time respectively. The third chart shows the decreasing of gradient norm versus computational time.

We observe that in the first several iterations, the fitting error of the L-BFGS fitting method does not decrease as fast as SDM and TDMLM. That is because that in the L-BFGS algorithm (Algorithm 2), the approximation of inverse Hessian matrix needs to be accumulated by using information from a sequence of sufficiently many iterations. Therefore, at the first several iterations, the approximant matrix is not accurate enough and this slows down the performance of the L-BFGS fitting method. However, an iteration of the L-BFGS fitting method is much faster than those of PDM, TDMLM and SDM. As a result, the L-BFGS fitting method converges much faster than the other three methods in terms of computational time, as shown in Figs. 3(d), 4(d) and 5(d).

Convergence. The convergence behaviors of the four methods can be observed from the third chart in the above three examples, showing the decrease of gradient norm against computational time. The termination criterion for all these examples is $\|\nabla f\|_\infty < 10^{-8}$, where f is the objective function. From Figs. 3(e), 4(e) and 5(e), we observe that the L-BFGS fitting method is the only method that always meets this criterion, i.e. the gradient norm of its objective function reaches the threshold of 10^{-8}. This is not surprising since the L-BFGS fitting method implements a well-studied optimization method (the L-BFGS algorithm, Algorithm 2) that has demonstrated superior convergence behavior close to the superlinear rate possessed by the BFGS method (Nocedal and Wright, 1999).

Comparison with the method of Speer et al. In Speer et al. (1998), Speer et al proposed to use the Gauss–Newton method to solve the least squares problem (1). This method also optimizes $\mathcal{T}$ and $\mathcal{P}$ simultaneously. However, in every iteration it still needs to formulate and solve a linear system of equations which involve both location parameters and control points as variables. Therefore, this method is inefficient for large data sets. Fig. 6 shows an example with noise containing 2500 data points. We use this example to compare the L-BFGS fitting method with SDM and Speer's method. From Fig. 6 we can see that the L-BFGS fitting method is capable of producing a satisfactory curve about 9 times faster than SDM; SDM in turn is about 9 times faster than Speer's method.

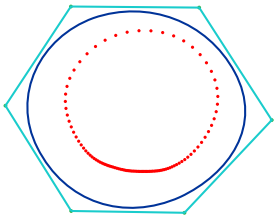
More examples. We present more examples in Figs. 9, 10 and 11.

5.2. Analysis and discussions

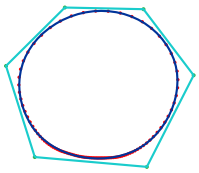
It is difficult to provide a theoretical proof on the superior efficiency of the L-BFGS fitting method over existing methods. As an alternative, in this section we shall conduct an empirical study on the efficiency of PDM, TDMLM, SDM and the L-BFGS fitting method, in order to gain a better understanding of their relative performances. The traditional methods (PDM, TDMLM, SDM) that update control points and location parameters separately mainly include the following tasks: linear system formulation and solving for control points and foot points computation for location parameters. The timing data for different parts of the methods for the examples in this paper are presented in Table 1. The L-BFGS fitting method consists of two parts: the two-loop algorithm for computing a descent direction and a line-search algorithm for deciding step-size. Timings for these two parts on the same examples as in Table 1 are listed in Table 2.

We have the following observations on these timing data.

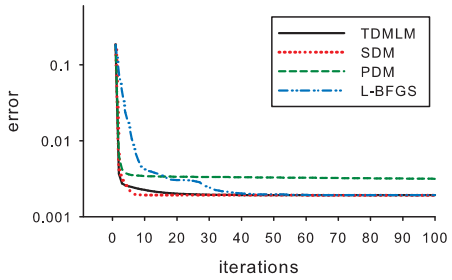
- Although the number of control points is generally much fewer than the number of data points, traditional methods (e.g. PDM, TDMLM and SDM) still consume more than half of the total time on updating control points, because of the need to fill the matrix and solving the linear system in every iteration. The L-BFGS algorithm instead does not solve any linear equations, so is much faster.
- Foot point computation is very time consuming. If we re-compute the initialization of foot points in every iteration, the overall time for foot point computation would be more than 90% of the total time of the algorithm, as observed in Wang et al. (2006). In our implementation of the traditional methods used for comparison in this paper, we use as much as possible the foot points from the previous iteration as initialization for the current iteration, thus having saved



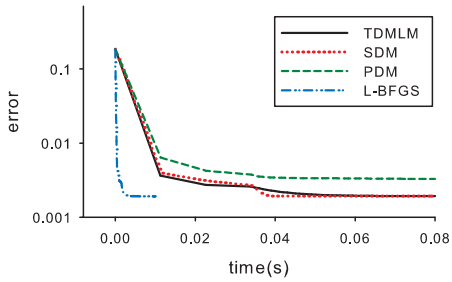
(a) Initialization.



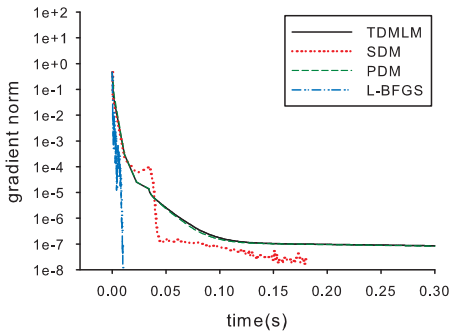
(b) Fitting curve.



(c) Error vs iteration.



(d) Error vs time.



(e) Gradient norm vs time.

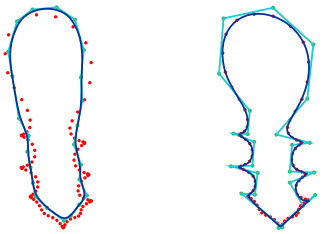
L-BFGS	PDM	TDMLM	SDM
$3.2 \cdot 10^{-3}$	0.95	$4.5 \cdot 10^{-2}$	$4.0 \cdot 10^{-2}$

(f) Time to attain minimal error (in seconds).

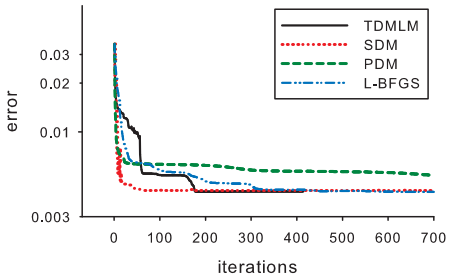
L-BFGS	PDM	TDMLM	SDM
$5.94 \cdot 10^{-5}$	$9.61 \cdot 10^{-4}$	$1.04 \cdot 10^{-3}$	$1.41 \cdot 10^{-3}$

(g) Time cost for an iteration (in seconds).

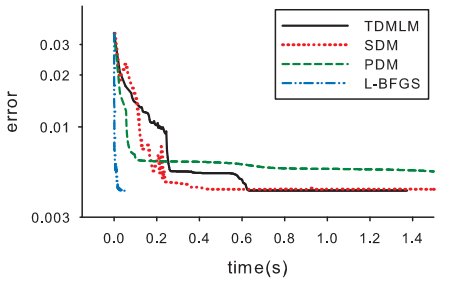
Fig. 3. The target shape is a set of 100 points on a circle. A B-spline curve with 6 control points is used to fit it. No fairing term is used in this example.



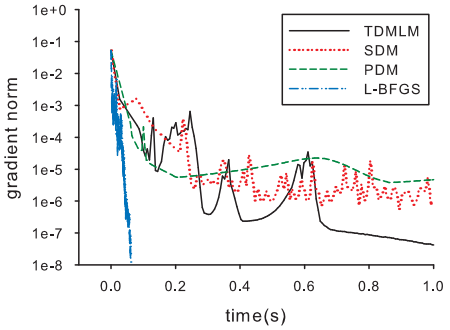
(a) Initialization. (b) Fitting curve.



(c) Error vs iteration.



(d) Error vs time.



(e) Gradient norm vs time.

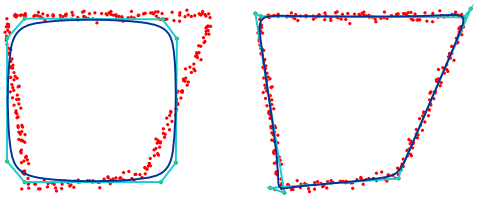
L-BFGS	PDM	TDMLM	SDM
$3.5 \cdot 10^{-3}$	3.3	0.63	0.43

(f) Time to attain minimal error (in seconds).

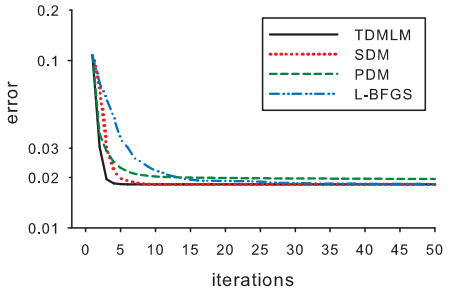
L-BFGS	PDM	TDMLM	SDM
$5.39 \cdot 10^{-5}$	$2.22 \cdot 10^{-3}$	$3.32 \cdot 10^{-3}$	$4.64 \cdot 10^{-3}$

(g) Time cost for an iteration (in seconds).

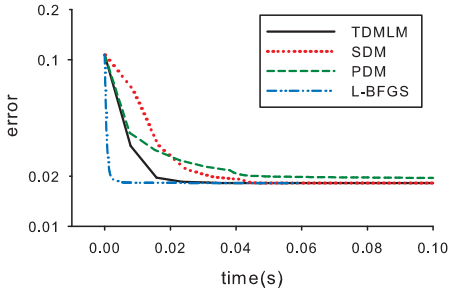
Fig. 4. An example with sharp features. The fitting curve has 24 control points and the data set contains 90 points. The coefficients of the fairing term are set to $\alpha = 0$ and $\beta = 5 \cdot 10^{-4}$.



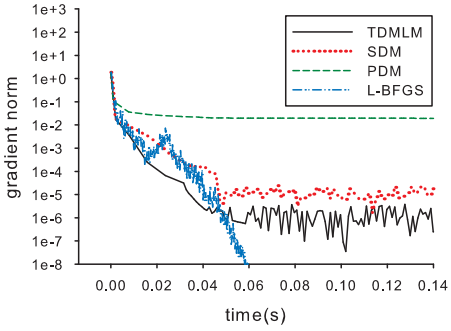
(a) Initialization. (b) Fitting curve.



(c) Error vs iteration.



(d) Error vs time.



(e) Gradient norm vs time.

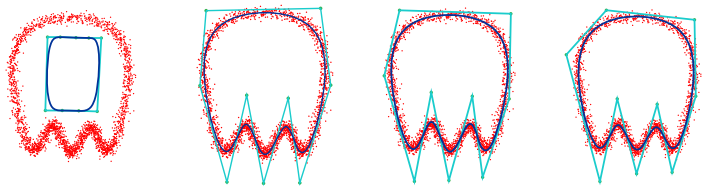
L-BFGS	PDM	TDMLM	SDM
$5.2 \cdot 10^{-3}$	0.82	$3.2 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$

(f) Time to attain minimal error (in seconds).

L-BFGS	PDM	TDMLM	SDM
$1.52 \cdot 10^{-4}$	$9.07 \cdot 10^{-4}$	$1.11 \cdot 10^{-3}$	$1.14 \cdot 10^{-3}$

(g) Time cost for an iteration (in seconds).

Fig. 5. An example with noisy data points. The fitting curve has 8 control points and the data set has 300 points. No fairing term is used in this example.



(a) Initialization. (b) Result of L-BFGS. (c) Result of SDM. (d) Result of Speer's method.

	L-BFGS	SDM	Speer's
time(s)	1.0202	9.2754	87.5810
error	0.01796	0.01805	0.01947

(e) Timing and fitting errors.

Fig. 6. Comparisons with L-BFGS, SDM and Speer's method. The coefficients of the fairing term are $\alpha = 0$ and $\beta = 10^{-3}$.

Table 1
Computational time for different parts of the PDM, TDMLM and SDM methods.

		Matrix filling	Matrix solving	Foot point projection
Fig. 3	PDM	23.8%	15.5%	60.6%
(ctrl pts: 6	TDMLM	22.6%	35.3%	41.9%
data pts: 100)	SDM	31.5%	41.4%	26.8%
Fig. 4	PDM	10.7%	55.6%	33.8%
(ctrl pts: 24	TDMLM	7.1%	67.3%	26.0%
data pts: 90)	SDM	8.8%	70.0%	21.1%
Fig. 5	PDM	25.5%	26.6%	48.3%
(ctrl pts: 8	TDMLM	25.5%	29.6%	45.2%
data pts: 300)	SDM	33.9%	26.6%	39.8%
Fig. 9	PDM	3.4%	73.6%	23.0%
(ctrl pts: 32	TDMLM	14.3%	63.6%	22.0%
data pts: 96)	SDM	18.9%	60.4%	20.6%
Fig. 10	PDM	21.2%	34.2%	45.0%
(ctrl pts: 30	TDMLM	8.1%	78.6%	13.0%
data pts: 600)	SDM	23.1%	50.2%	27.2%
Fig. 11	PDM	9.9%	15.1%	73.3%
(ctrl pts: 66	TDMLM	15.6%	31.8%	50.0%
data pts: 2000)	SDM	29.0%	26.9%	41.9%

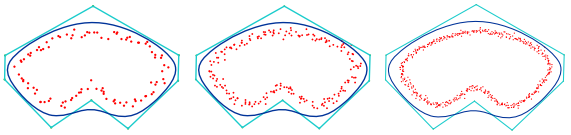
Table 2
Computational time for different parts of the L-BFGS algorithm, $m = 20$.

	Computing descent direction	Linesearch
Fig. 3	95.8%	4.2%
Fig. 4	97.8%	2.1%
Fig. 5	96.8%	3.2%
Fig. 9	96.4%	3.6%
Fig. 10	91.4%	8.6%
Fig. 11	89.7%	10.3%

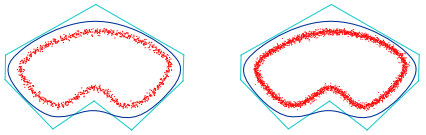
a lot of time for traditional methods. Even so, the L-BFGS fitting methods still outperforms these traditional methods, since there is generally no need to perform foot point projection in the L-BFGS fitting method. In rare cases, foot point computation is needed for the L-BFGS method to jump out of a poor local minimum, as we have explained in Section 4. Note that this re-initialization issue is mostly due to the quality of initialization and it affects the L-BFGS algorithm as well as the other traditional methods.

- The L-BFGS fitting method performs optimization in a much higher dimensional space than those of traditional methods since generally the number of data points is much larger than that of the control points. Therefore, the terrain of the functional is supposed to be much more complicated and the optimization is more difficult. The linesearch algorithm is therefore necessary for the stable convergence of the L-BFGS fitting algorithm. Table 2 shows that the computational time by the linesearch algorithm usually takes a small part of the total time (less than 10% in most cases).

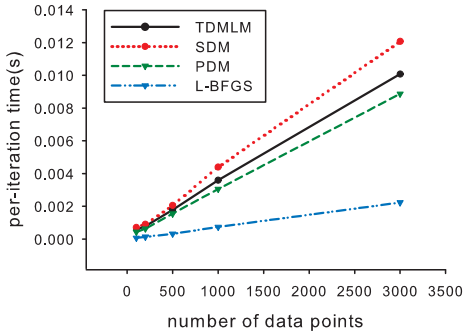
We now study how the computational time depends on the number of control points and the number of data points.



(a) 100 data points. (b) 200 data points. (c) 500 data points.

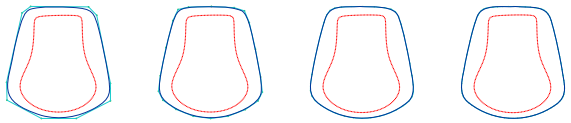


(d) 1000 data points. (e) 3000 data points.

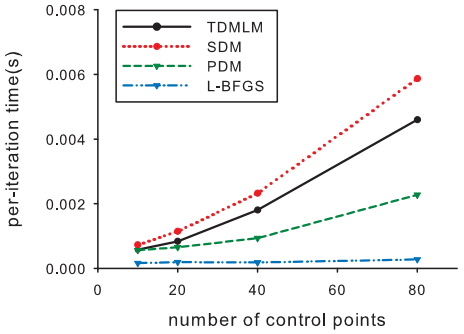


(f) Per-iteration time as the number of data points increases.

Fig. 7. Increasing data points: An example with 8 control points. The coefficients of the fairing term are $\alpha = 5 \cdot 10^{-4}$ and $\beta = 0$.



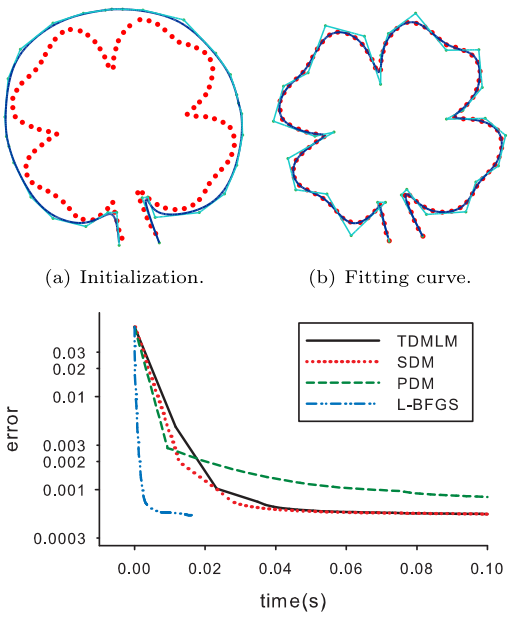
(a) 10 Control points (b) 20 Control points (c) 40 Control points (d) 80 Control points



(e) Per-iteration time as the number of control points increases.

Fig. 8. With increased number of control points. There are 200 data points. No fairing term is used.

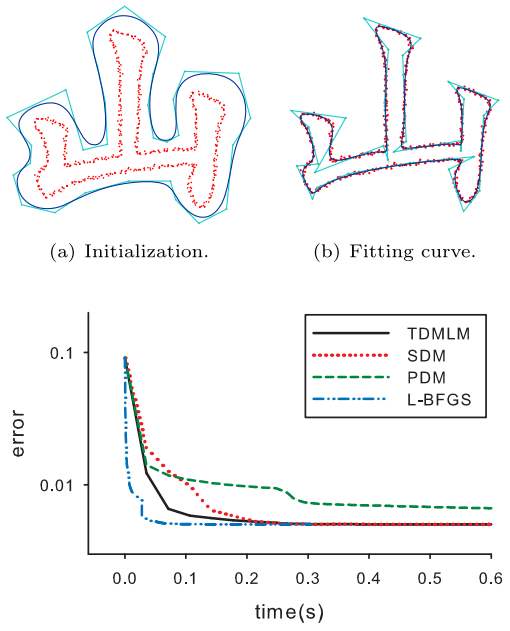
Timing vs # of data points. In Fig. 7, we show computational time with increased number of data points for various methods. The number of data points in these point sets is 100, 200, 500, 1000 and 3000 respectively. The fitting curve has 8 control points. Fig. 7 shows that the computational time for each iteration of all 4 methods depends almost linearly on the number of data points. This can be explained as follows. It is not difficult to see that in the PDM, TDMLM and SDM, the time for matrix building and foot point projection is linear in the number of data points. The time for solving linear system is constant since the number of control points is fixed. Consequently, the total time for these three methods increase



(a) Initialization. (b) Fitting curve.

(c) Fitting error vs time.

Fig. 9. An open curve of clover shape with 32 control points and 96 data points. The coefficients of the fairing term are $\alpha = 10^{-4}$ and $\beta = 0$.



(a) Initialization. (b) Fitting curve.

(c) Fitting error vs time.

Fig. 10. A Chinese character with 30 control points and 600 data points which means "mountain". The coefficients of the fairing term are $\alpha = 5 \cdot 10^{-4}$ and $\beta = 0$.

linearly as the number of data points increases. For the L-BFGS fitting method, computational time is linear in the number of variables ($2 \times$ the number of control points $+$ the number of data points), therefore the computational time of the L-BFGS fitting method also increases linearly as the number of data points increases.

Timing vs # of control points. The relationship of computational time to the number of control points of the fitting B-spline curve can be observed in Fig. 8. We insert new control points by knot insertion in each knot interval and get 4 B-spline curves with the number of control points: 10, 20, 40 and 80 respectively. The number of target data points is 200. We see that the per-iteration time for the tested traditional methods increases faster than the L-BFGS fitting method when the

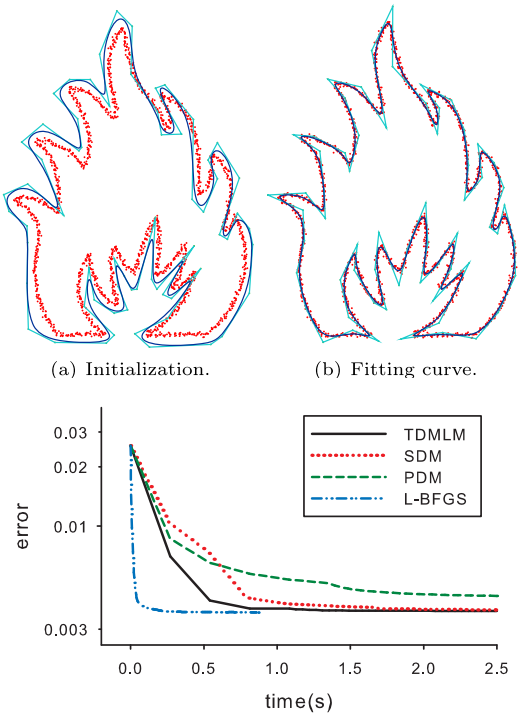


Fig. 11. Flame with 66 control points and 2000 data points. The coefficients of the fairing term are $\alpha = 10^{-3}$ and $\beta = 10^{-2}$.

number B-spline control points increases. That is because in the PDM, TDMLM and SDM, the time to solve the linear system by conjugate gradient method is quadratic in the number of control points, but the computational time of the L-BFGS algorithm (i.e. the two-loop algorithm and the linesearch) depends only linearly on the number of control points.

These experiments show that the L-BFGS fitting method is more suitable for large scale curve fitting problems, especially when the target shape is complicated and a large number of control points are involved.

6. Conclusion and future work

In this paper, we propose a new curve fitting method based on the L-BFGS optimization technique. The unique features of this algorithm are that it does not need to perform the time-consuming foot point projection in every iteration as in traditional approaches and that it does not need to formulate and solve a linear system of equations in every iteration; instead, it uses only efficient vector multiplications. As a result, this new method is much faster than other traditional methods, as demonstrated by a number of experimental results presented. In the future we will extend this method to solving the B-spline surface fitting problem, for which we expect even more significant improvements over the existing methods because usually a large number of data points as well as a large number of control points are used for surface fitting.

Acknowledgements

The work of Pengbo Bo was partially supported by the National Natural Science Foundation of China (No. 61173086 and No. 61179009). The work of Wenping Wang was partially supported by the National Basic Research Program of China (2011CB302400) and the State Key Program of NSFC project (60933008).

References

Aigner, M., Jüttler, B., 2005. Robust computation of foot points on implicitly defined curves. In: Dæhlen, M., Mørken, K., Schumaker, L. (Eds.), Mathematical Methods for Curves and Surfaces: Tromsø 2004. Nashboro Press, Brentwood, pp. 1–10.

Alhanaty, M., Bercovier, M., 2001. Curve and surface fitting and design by optimal control methods. Computer-Aided Design 33, 167–182.

Björck, Å., 1996. Numerical Methods for Least Squares Problems. Society for Industrial Mathematics.

Blake, A., Isard, M., 1998. Active Contours, vol. 2. Springer-Verlag, London.

Forsey, D.R., Bartels, R.H., 1995. Surface fitting with hierarchical splines. ACM Trans. Graph. 14, 134–161.

Haber, J., Zeilfelder, F., Davydov, O., Seidel, H., 2001. Smooth approximation and rendering of large scattered data sets. In: Visualization, VIS'01. Proceedings. IEEE, pp. 341–571.

Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W., 1994. Piecewise smooth surface reconstruction. In: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94. ACM, New York, NY, USA, pp. 295–302.

Hoschek, J., 1988. Intrinsic parametrization for approximation. Comput. Aided Geom. Design 5, 27–31.

Hoschek, J., Schneider, F.-J., Wassum, P., 1989. Optimal approximate conversion of spline surfaces. Comput. Aided Geom. Design 6, 293–306.

Hu, S.-M., Wallner, J., 2005. A second order algorithm for orthogonal projection onto curves and surfaces. Comput. Aided Geom. Design 22, 251–260.

Jiang, L., Byrd, R., Eskow, E., Schnabel, R., 2004. A preconditioned L-BFGS algorithm with application to molecular energy minimization. Technical report. Colorado Univ. at Boulder Dept. of Computer Science.

Liu, Y., Wang, W., 2008. A revisit to least squares orthogonal distance fitting of parametric curves and surfaces. In: Proceedings of the 5th International Conference on Advances in Geometric Modeling and Processing. Springer-Verlag, Hangzhou, China, pp. 384–397.

Nocedal, J., Wright, S.J., 1999. Numerical Optimization. Springer-Verlag.

Pavlidis, T., 1983. Curve fitting with conic splines. ACM Trans. Graph. 2, 1–31.

Plass, M., Stone, M., 1983. Curve-fitting with piecewise parametric cubics. In: Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '83. ACM, New York, NY, USA, pp. 229–239.

Pottmann, H., Hofer, M., 2003. Geometry of the squared distance function to curves and surfaces. In: Visualization and Mathematics, III, pp. 221–242.

Pottmann, H., Leopoldseder, S., Hofer, M., 2002. Approximation with active B-spline curves and surfaces. In: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications 2002, pp. 8–25.

Sarfraz, M., Riyazuddin, M., Baig, M.H., 2006. Capturing planar shapes by approximating their outlines. Journal of Computational and Applied Mathematics 189, 494–512.

Saux, E., Daniel, M., 2003. An improved Hoschek intrinsic parametrization. Comput. Aided Geom. Design 20, 513–521.

Speer, T., Kuppe, M., Hoschek, J., 1998. Global reparametrization for curve approximation. Comput. Aided Geom. Design 15, 869–877.

Wang, W., Pottmann, H., Liu, Y., 2006. Fitting B-spline curves to point clouds by curvature-based squared distance minimization. ACM Trans. Graph. 25, 214–238.

Xie, H., Qin, H., 2001. Automatic knot determination of NURBS for interactive geometric design. In: Proceedings of the International Conference on Shape Modeling & Applications. IEEE Computer Society, pp. 267–276.