Available online at www.sciencedirect.com

ScienceDirect

Computer-Aided Design 38 (2006) 1251–1263

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Automatic least-squares projection of points onto point clouds with applications in reverse engineering

Yu-Shen Liu[a,b,*], Jean-Claude Paul[a], Jun-Hai Yong[a], Pi-Qiang Yu[c], Hui Zhang[a], Jia-Guang Sun[a], Karthik Ramani[b]

[a] *School of Software, Tsinghua University, Beijing 100084, PR China*
[b] *Purdue Research and Education Center for Information Systems in Engineering (PRECISE), Purdue University, West Lafayette, IN 47907-2024, USA*
[c] *School of Computer and Information Technology, Beijing Jiaotong University, PR China*

Received 19 January 2006; accepted 1 September 2006

Abstract

A novel method for projecting points onto a point cloud, possibly with noise, is presented based on the point *directed projection* (DP) algorithm proposed by Azariadis P., Sapidis N. [Drawing curves onto a cloud of points for point-based modelling. Computer-Aided Design 2005; 37(1): 109–22]. The new method operates directly on the point cloud without any explicit or implicit surface reconstruction procedure. The presented method uses a simple, robust, and efficient algorithm: *least-squares projection* (LSP), which projects points onto the point cloud in a least-squares sense without any specification of the projection vector. The main contribution of this novel method is the automatic computation of the projection vector. Furthermore, we demonstrate the effectiveness of this approach through a number of application examples including thinning a point cloud, point normal estimation, projecting curves onto a point cloud and others.

Keywords: Point clouds; Projection; Least-squares; Reverse engineering

1. Introduction

With 3D scanners becoming the standard source for geometric data acquirement, point clouds have received a growing amount of attention as an alternative surface representation [16,26]. Point-based techniques such as point rendering [16], parameterization [7], simplification [27], thinning [3], area computation [21], registration [11,23], and reconstruction [18], also become an active research area in computer aided design (CAD) and computer graphics (CG). In this paper, we focus on the projection of points onto point clouds, which is a common problem for many point-based techniques.

Projecting a point onto a parametric or implicit surface in order to find the closest point (footpoint) is an important problem in geometric modeling, computer graphics, and computer vision (e.g. Refs. [13,22,29]). Similarly, projecting a point onto a point cloud is also a key issue in many point-based techniques. It can be used in the *moving least squares* (MLS) technique [18,20], the point cloud collision detection [15], and the ICP (*iterative closest point*) algorithm for shape registration [9,23]. Based on a novel point *directed projection* algorithm, Azariadis [7] develops the Dynamic Base Surfaces (DBS) method in order to solve the parameterization problem for unstructured point clouds. The resulting parameterization is then utilized for high-accuracy surface fitting. By combining with the point directed projection algorithm, Azariadis and Sapidis [8] also design a new curve-drawing technique for producing curves lying onto point clouds.

1.1. Related work

Traditionally, there are many methods for projecting a point onto a parametric or implicit surface [13,22,29]. A review of the many available methods for computing the projection point onto the parametric or implicit surface is beyond the scope

* Corresponding author at: School of Software, Tsinghua University, Beijing 100084, PR China. Tel.: +86 10 62795455; fax: +86 10 62795460.
E-mail addresses: liuyushen00@gmail.com (Y.-S. Liu), paul@tsinghua.edu.cn (J.-C. Paul), yongjh@tsinghua.edu.cn (J.-H. Yong), yupiqiang@gmail.com (P.-Q. Yu), huizhang@tsinghua.edu.cn (H. Zhang), ramani@purdue.edu (K. Ramani).

0010-4485/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2006.09.001

of this paper. The reader may consult Ref. [13] for detailed expositions. The existing methods in Refs. [13,22,29] require a parametric or implicit representation for the surface. However, an unorganized point set does not contain any extra information of the surface, except for the geometric position. One possible solution is to reconstruct globally or locally surfaces from the point set [5,12,24] and then use the common techniques to project points onto the reconstructed surface. There are several limitations on the indirect method based on explicit or implicit surface reconstruction. First, it is a non-trivial task to build a surface representation that is faithful to point sets, and the errors of the projection approximation would be introduced by various surface reconstruction methods. Second, it often fails to reconstruct surfaces from large and complex point sets acquired by 3D scanning devices such that the projection operation is unsuccessful. Furthermore, the explicit or implicit surface reconstruction requires the expenditure of large amounts of time and space if the number of points is gigantic.

Recently, some point-based rendering techniques [1,31] are also presented. Adamson and Alexa [1] use implicit surface models for ray tracing point set surfaces. Wald and Siedel [31] improve their works in order to allow for interactively ray tracing even highly complex models of several million points. The core of ray tracing of point-based models is an efficient intersection algorithm between a ray and the point set surface. Although a very efficient intersection implementation is performed, it looks more relative to the point directed projection onto the point cloud along an associated projection vector [7, 8]. It needs to know the projection direction determined by the ray. In contrast, our paper focuses on the problem of point projection onto point clouds without any specification of the projection vector.

Based on the MLS method [20], Alexa et al. [2,3] define a point set surface approximated locally for a certain neighborhood by polynomial, and then project the point near the point set onto the surface. Some improved MLS methods are also proposed in Refs. [6,10,18]. Their methods are essentially reconstruction-based methods and need a certain neighborhood of the test point. In addition, since it is not rare for a point set acquired by 3D scanning devices to include under-sampled areas [8], it becomes difficult to find a suitable neighborhood for performing the MLS projection operation in under-sampled areas. Though the closest point can also be simply computed as the one with the minimal Euclidean distance of the test point to all points of the point cloud, point sets acquired by 3D scanning devices typically contain noise and irregular samples, resulting in larger approximation errors. Another possible solution is to find the k-nearest neighborhoods of the test point, and then compute the average of the k neighborhood points [15]. This method is not related to a measurement error analysis, and it is also faced with the same problem how to choose a suitable neighborhood. Recently, Klein and Zachmann [15] describe a method for collision detection of point clouds. Although the authors do not compute a k-nearest neighborhood (they utilize the simple implicit surface definition and a hierarchical sphere covering for automatically defining a neighborhood), the performance and space can be increased during the increase of the point hierarchy and the hierarchical sphere covering. The current literature does not offer a satisfactory solution to the problem of projecting an arbitrary point onto a point cloud when no explicit or implicit surface reconstruction procedure is used.

The work most related to ours is Ref. [7], in which a new method for finding a parameterization of an unorganized point cloud using the point directed projection onto the point cloud along an associated projection vector is presented by Azariadis. His method operates directly on the point cloud without any explicit or implicit surface reconstruction procedure. The projection method is essentially a least-squares method along associated projection vectors. It has also been applied to curve-drawing onto point clouds for point-based modeling [8], where the projection vectors are specified through a graphics interface tool. Their directed projection algorithm has two advantages due to the iterative property with weight functions [8]. The first advantage is that no certain neighborhood is fixed for the test point, so it is suitable for the point set with noise and irregular samples. The second advantage is that an appropriate point cloud error function is given, which is used to solve the problem of directed projection onto a point cloud. However, since the projection vector is unknown for a test point and an unorganized point cloud, their method cannot be applied directly to compute point projection without any projection direction. The main difficulty is how to determine automatically the projection direction.

1.2. Contributions

To overcome the aforementioned difficulties, we present a novel algorithm of projecting points onto a point cloud. This algorithm is not using any explicit or implicit surface reconstruction procedure to the given point cloud. In some sense, our algorithm can be considered an extension of the *directed projection* (DP) algorithm proposed by Azariadis and Sapidis [7,8]. Essentially, the proposed algorithm is also a least-squares method like Azariadis's method, so we call the new method a *least-squares projection* (LSP). First, the projection direction of LSP is determined by optimizing the solutions of all projection directions. Then the test point is projected onto the point cloud along the determined projection direction, where the projection position of the test point is computed by minimizing an error function defined for measuring the distance between the projected point and the point cloud [8]. In order to investigate the accuracy and robustness of the LSP algorithm, several experiments have been conducted. Furthermore, we demonstrate the effectiveness of this algorithm through a number of application examples including thinning a point cloud, point normal estimation, projecting curves onto a point cloud and others. The major contributions of our work are as follows.

– Propose the formula of the projection direction and extend the directed projection algorithm [7,8] to the LSP algorithm. Especially, the new method does not use any explicit or implicit surface reconstruction procedure and can be implemented easily.

– Investigate the effect of the LSP algorithm on the discontinuous problem. Our experimental data shows the LSP algorithm has a good approximation.
– Apply the LSP algorithm to some point-based techniques, including thinning a point cloud, point normal estimation, projecting curves onto a point cloud and others. The proposed algorithm has given us some very promising results in reverse engineering.

The remainder of this paper is organized as follows. In Section 2, the procedure of the LSP method is described. In Section 3, some experimental results are introduced. In Section 4, some applications are presented. The comparison with the MLS method is provided in Section 5. Finally, we give conclusions in Section 6.

2. Least-squares projection of a point onto a point cloud

2.1. Problem statement

We consider the following problem. Let $\mathcal{C}_N = \{\mathbf{p}_i | i = 1, \ldots, N\}$ be a set of unorganized data points. The set $\mathcal{C}_N$ of data points is assumed to be a sampling of an unknown surface S, called the point-sampled surface, with or without boundary. We suppose that the unorganized data points, often referred to as a *point cloud* or *scattered data points* in the literature, may have non-uniform distribution with considerable noise. Such data point sets are common in reverse engineering process, where the surface of a sculptured object is measured by a 3D scanner or by a coordinate measurement machine. Suppose that $\mathbf{p} = (x, y, z)$ be an arbitrary 3D point. Our goal is to find the projection point of $\mathbf{p}$ onto the point-sampled surface S consisting of the point cloud $\mathcal{C}_N$ by minimizing an error function.

2.2. The review of the directed projection algorithm

Azariadis [7] proposes an appropriate point cloud error function and uses it to solve the problem of point projection onto a point cloud along a projection direction. We call Azariadis's method the *directed projection* (DP) algorithm. First, we summarize the DP algorithm as follows [7,8].

1. Define an error function for measuring the distance between the point to be projected and the point cloud.
2. Project the point of interest onto the point cloud by minimizing the above error function.

Following the above DP algorithm, we review the detail of the error function. Consider a point cloud $\mathcal{C}_N$ and a test point $\mathbf{p} = (x, y, z)$ with an associated projection vector $\mathbf{n} = (n_x, n_y, n_z)$. Each $\mathbf{p}_i$ is associated to a positive weight α_i. Let $\mathbf{p}^*$ be the projection point of $\mathbf{p}$ for the DP problem. Find $\mathbf{p}^*$ by minimizing the following weighted sum of the squared distances:

$$E(\mathbf{p}^*) = \sum_{i=1}^{N} \alpha_i \|\mathbf{p}^* - \mathbf{p}_i\|^2. \tag{1}$$

For given weights $\{\alpha_i\}$, Azariadis writes $\mathbf{p}^* = (x^*, y^*, z^*)$ as

$$\mathbf{p}^* = \mathbf{p}^*(t) = \mathbf{p} + t\mathbf{n}, \quad t \in \mathbb{R}, \tag{2}$$

where $\mathbf{n}$ is the projection vector. Then, by substituting Eq. (2) into Eq. (1), the solution of minimizing Eq. (1) is [7]

$$t = \frac{\beta - \mathbf{p} \cdot \mathbf{n}}{\|\mathbf{n}\|^2}, \tag{3}$$

where $\cdot$ denotes the dot product and

$$\beta = \frac{c_1 n_x + c_2 n_y + c_3 n_z}{c_0}, \quad \text{and}$$
$$c_0 = \sum_{i=1}^{N} \alpha_i, \; c_1 = \sum_{i=1}^{N} \alpha_i x_i,$$
$$c_2 = \sum_{i=1}^{N} \alpha_i y_i, \; c_3 = \sum_{i=1}^{N} \alpha_i z_i. \tag{4}$$

Intuitively, the projection process defined by Eqs. (2) and (3) can be regarded as a method for intersecting a given point cloud with the semi-infinite line defined by $\mathbf{p}$ and $\mathbf{n}$. Independently, Liu et al. [21] and Schaufler and Jensen [30] have given a simple result for the similar problem of intersecting a line with a point cloud.

2.3. Least-squares projection of a point onto a point cloud

Although the DP algorithm on the basis of Eqs. (2) and (3) can produce reasonable results along an projection direction vector $\mathbf{n}$, we would like to find the projection point of $\mathbf{p}$ under the condition of an unknown projection direction. The main difficulty is how to determine automatically the projection direction. A possible approach would be to consider projecting a point $\mathbf{p}$ onto the point cloud $\mathcal{C}_N$ along different directions on the basis of Eq. (3), and then finding an "appropriate" projection direction. "Appropriate" means that the sum of the squared distances, which is computed through Eq. (1) using the DP algorithm along the projection direction, is not more than one along any other projection direction.

Consider a variable projection vector $\mathbf{n}$ for Eq. (3). Then Eq. (3) can be rewritten as

$$t(\mathbf{n}) = \frac{\beta - \mathbf{p} \cdot \mathbf{n}}{\|\mathbf{n}\|^2} = \frac{\frac{1}{c_0}\mathbf{c} \cdot \mathbf{n} - \mathbf{p} \cdot \mathbf{n}}{\|\mathbf{n}\|^2}, \tag{5}$$

where $\mathbf{c} = (c_1, c_2, c_3)$ and $c_i (i = 0, 1, 2, 3)$ are given by Eq. (4). We denote

$$\mathbf{m} = \frac{1}{c_0}\mathbf{c} - \mathbf{p},$$

where

$$\mathbf{m} = (m_1, m_2, m_3), \quad \text{and}$$
$$m_1 = \frac{c_1}{c_0} - x, \; m_2 = \frac{c_2}{c_0} - y, \; m_3 = \frac{c_3}{c_0} - z. \tag{6}$$

Thus, $t(\mathbf{n})$ in Eq. (5) can also be written by

$$t(\mathbf{n}) = \frac{\mathbf{m}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{n}}.$$

2.3.1. Nonlinear optimization

How to define an optimization function for approximating the projection vector? By substituting $t(\mathbf{n})$ into Eq. (2), it could be a choice to minimize the objective function equation (1) as follows:

$$\begin{aligned} E(\mathbf{n}) &= \sum_{i=1}^{N} \alpha_i \|\mathbf{p}^*(\mathbf{n}) - \mathbf{p}_i\|^2 \\ &= \sum_{i=1}^{N} \alpha_i \|\mathbf{p} - \mathbf{p}_i + t(\mathbf{n})\mathbf{n}\|^2 \\ &= \sum_{i=1}^{N} \alpha_i \left\|\Delta\mathbf{p}_i + \frac{\mathbf{m}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{n}}\mathbf{n}\right\|^2, \end{aligned} \tag{7}$$

where $\Delta\mathbf{p}_i = \mathbf{p} - \mathbf{p}_i$.

Eq. (7) expresses the energy function $E(\mathbf{n})$ with respect to the variable projection direction $\mathbf{n}$. Thus, $E(\mathbf{n})$ is optimized when its derivative with respect to $\mathbf{n}$ is zero (this implies that the partial derivatives of $E(n_x, n_y, n_z)$ with respect to n_x, n_y, and n_z should be zero). For this problem, the constraints of $\mathbf{n}\cdot\mathbf{n} = 1$ and $\mathbf{m}\cdot\mathbf{n} >= 0$ can be enforced, then

$$\begin{cases} \text{Min}\left\{E(\mathbf{n}) = \sum_{i=1}^{N} \alpha_i \|\Delta\mathbf{p}_i + (\mathbf{m}\cdot\mathbf{n})\mathbf{n}\|^2\right\} \\ \mathbf{n}\cdot\mathbf{n} = 1. \end{cases} \tag{8}$$

However, it is not easy to solve the nonlinear optimization problem. In the next section, we propose a linear optimization method for approximating a projection vector of Eq. (5), and use the solved projection vector for computing the projection point.

2.3.2. Linear optimization

Being different from the above nonlinear optimization method, in this section we first optimize directly the function of the solution $t(\mathbf{n})$ with respect to the variable projection direction $\mathbf{n}$, which can be regarded as a method for finding the optimization solution when projecting a point onto a given point cloud surface along different directions $\mathbf{n}$. Then, we will give Propositions 2.1 and 2.2 for proving our conclusion, in which the weighted mean point that minimizes Eq. (1) lies on the line defined by the test point $\mathbf{p}$ and the optimization projection vector $\mathbf{n}$.

$t(\mathbf{n})$ is optimized when its derivative with respect to $\mathbf{n}$ is zero (this implies that the partial derivatives of $t(n_x, n_y, n_z)$ with respect to n_x, n_y, and n_z should be zero). For this problem, the constraint of $\|\mathbf{n}\|^2 = 1$ (i.e. $\mathbf{n}\cdot\mathbf{n} = 1$) can also be enforced, then

$$\begin{cases} \dfrac{\partial t(\mathbf{n})}{\partial \mathbf{n}} = 0 \\ \mathbf{n}\cdot\mathbf{n} = 1. \end{cases} \tag{9}$$

Since $\mathbf{n}\cdot\mathbf{n} = 1$, using a lagrange multiplier for Eq. (9), we obtain

$$\begin{aligned} L(\mathbf{n}) &= t(\mathbf{n}) + \lambda(\mathbf{n}\cdot\mathbf{n} - 1) \\ &= \mathbf{m}\cdot\mathbf{n} + \lambda(\mathbf{n}\cdot\mathbf{n} - 1) \\ &= (m_1 n_x + m_2 n_y + m_3 n_z) + \lambda(n_x^2 + n_y^2 + n_z^2 - 1). \end{aligned} \tag{10}$$

In order to simplify the optimization function $t(\mathbf{n})$, we assume that the weights α_i are pre-computed and are not correlative with the projection direction $\mathbf{n}$. That is, that the weights are constant in Eq. (5) when the test point and point cloud are given.

By setting

$$\begin{cases} \dfrac{\partial L}{\partial n_x} = m_1 + 2\lambda n_x = 0 \\ \dfrac{\partial L}{\partial n_y} = m_2 + 2\lambda n_y = 0 \\ \dfrac{\partial L}{\partial n_z} = m_3 + 2\lambda n_z = 0, \end{cases} \tag{11}$$

we obtain

$$\mathbf{n} = -\frac{1}{2\lambda}\mathbf{m}. \tag{12}$$

Next, the following proposition points out that the direction vector $\mathbf{n}$ defined by Eq. (12) maximizes or minimizes $t(\mathbf{n})$ with respect to $\lambda < 0$ or $\lambda > 0$.

Proposition 2.1. *The maximum (or minimum) of function $t(\mathbf{n})$ in Eq. (5) with respect to $\mathbf{n}$, is defined by*

$$\mathbf{n} = -\frac{1}{2\lambda}\mathbf{m},$$

i.e. Eq. (12), where $\lambda < 0$ (or $\lambda > 0$).

Proof. The necessary condition for the maximization (or minimization) of Eq. (5) with respect to $\mathbf{n}$ is

$$t'(\mathbf{n}) = 0 \Rightarrow \mathbf{n} = -\frac{1}{2\lambda}\mathbf{m} \tag{13}$$

and that the Hesse matrix of the lagrange function $L(\mathbf{n})$ (Eq. (10)) is the negative (or positive) definite matrix. The Hesse matrix of $L(\mathbf{n})$ is defined by

$$H_L = \begin{bmatrix} \dfrac{\partial^2 L}{\partial n_x^2} & \dfrac{\partial^2 L}{\partial n_x \partial n_y} & \dfrac{\partial^2 L}{\partial n_x \partial n_z} \\ \dfrac{\partial^2 L}{\partial n_y \partial n_x} & \dfrac{\partial^2 L}{\partial n_y^2} & \dfrac{\partial^2 L}{\partial n_y \partial n_z} \\ \dfrac{\partial^2 L}{\partial n_z \partial n_x} & \dfrac{\partial^2 L}{\partial n_z \partial n_y} & \dfrac{\partial^2 L}{\partial n_z \partial n_z} \end{bmatrix} = \begin{bmatrix} 2\lambda & 0 & 0 \\ 0 & 2\lambda & 0 \\ 0 & 0 & 2\lambda \end{bmatrix}.$$

If $\lambda < 0$, H_L is the negative definite matrix; otherwise if $\lambda > 0$, H_L is the positive definite matrix. Therefore, the unit vector defined by Eq. (12) with $\lambda < 0$ (or $\lambda > 0$) corresponds to the maximum (or minimum) of Eq. (5). □

Here, $\mathbf{n}$ is the enforced constraint that $\|\mathbf{n}\| = 1$. In practice, a simple way to maximize the solution is to solve Eq. (12) by setting $\lambda = -1$, and then rescale the direction $\mathbf{n}$ so that its length is one (which yields the same result). Especially, if $\mathbf{m}$ is equal to zero, t in Eq. (5) is also enforced zero, i.e. $\mathbf{p}^* = \mathbf{p}$.

Consider $\lambda = \lambda_0 < 0$ so that $\mathbf{n} = -\frac{1}{2\lambda_0}\mathbf{m}$ is the unit vector. Substituting $\mathbf{n} = -\frac{1}{2\lambda_0}\mathbf{m}$ into Eq. (5), then $t(\mathbf{n}) = -\frac{1}{2\lambda_0}\mathbf{m}^2 > 0$. Since $t = t(\mathbf{n}) > 0$, the projection point $\mathbf{p}^*$ of $\mathbf{p}$ in Eq. (2) is always on the positive direction of the projection vector $\mathbf{n}$. Correspondingly, if $\lambda = \lambda_0 > 0$, we obtain $t = t(\mathbf{n}) < 0$ (this means that $\mathbf{p}^*$ is always on the negative direction of $\mathbf{n}$). Along the projection direction $\mathbf{n} = -\frac{1}{2\lambda}\mathbf{m}$ with respect to $\lambda < 0$ or $\lambda > 0$, we can get the same projection point $\mathbf{p}^*$ by using the DP algorithm.

Now, we check the projection vector $\mathbf{n}$ in Eq. (12) through the following proposition.

Proposition 2.2. *The weighted mean point that minimizes Eq. (1) lies on the line defined by the test point* $\mathbf{p}$ *and the projection vector* $\mathbf{n}$ *in Eq. (12).*

Proof. We begin by considering the problem of finding a point $\mathbf{p}^{**}$ such that the sum of the squared distances between $\mathbf{p}^{**}$ and $\mathbf{p}_i$ is as small as possible. We use the squared-error criterion function in Eq. (1) and seek the value of $\mathbf{p}^{**}$ that minimizes Eq. (1). Partially being differing from the DP problem, $\mathbf{p}^{**}$ is independent of the test point $\mathbf{p}$ and the projection vector $\mathbf{n}$. Then $\mathbf{p}^{**}$ is the solution of the following problem:

$$\text{Find } \mathbf{p}^{**} \text{ minimizing } E(\mathbf{p}^{**}) = \sum_{i=1}^{N} \alpha_i \|\mathbf{p}^{**} - \mathbf{p}_i\|^2.$$

The point $\mathbf{p}^{**}$ is also called the *weighted mean* point of the point cloud $\mathcal{C}_N$. Then the minimization with respect to $\mathbf{p}^{**}$ is

$$E'(\mathbf{p}^{**}) = 0 \Rightarrow \mathbf{p}^{**} = \frac{\sum_{i=1}^{N} \alpha_i \mathbf{p}_i}{\sum_{i=1}^{N} \alpha_i} = \frac{\mathbf{c}}{c_0},$$

where c_0 and $\mathbf{c}$ are defined by Eqs. (4) and (5), respectively. Thus,

$$\mathbf{p}^{**} - \mathbf{p} = \frac{\mathbf{c}}{c_0} - \mathbf{p} = \mathbf{m},$$

where $\mathbf{m}$ is defined by Eq. (6). By substituting $(\mathbf{p}^{**} - \mathbf{p})$ into Eq. (12), we get

$$\mathbf{n} = -\frac{1}{2\lambda}\mathbf{m} = -\frac{1}{2\lambda}(\mathbf{p}^{**} - \mathbf{p}).$$

Clearly, $\mathbf{p}^{**}$ lies on the line defined by $\mathbf{p}$ and the projection vector $\mathbf{n}$ in Eq. (12). □

From Proposition 2.2, we can obtain an interesting conclusion that the projection vector $\mathbf{n}$ defined by Eq. (12) is co-linear with the line through the test point $\mathbf{p}$ and the *mean* point $\mathbf{p}^{**}$ of the point cloud $\mathcal{C}_N$. Geometrically, the projection vector defined by Eq. (12) can also be regarded as the vector from the mean point of the point cloud relative to the test point. Fig. 1(b) shows an example of projecting a test point $\mathbf{p}$ onto the whole point cloud, where $\mathbf{n}$ and $\mathbf{p}^*$ are the projection direction vector computed by Eq. (12) and the corresponding projection point using the DP method, respectively.

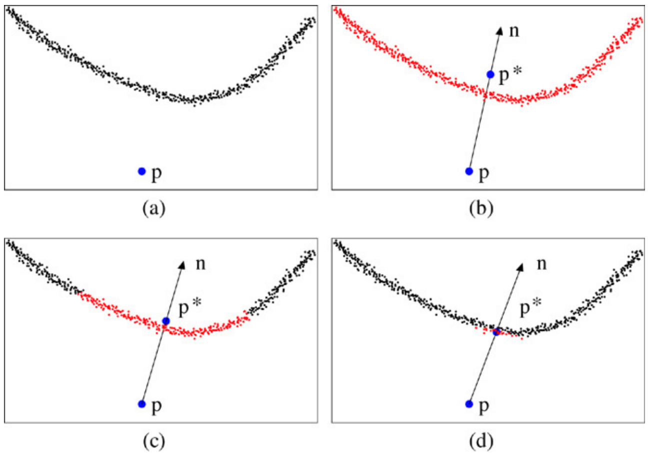


Fig. 1. The illustration of the LSP procedure. First, the projection direction vector $\mathbf{n}$ of projecting the test point $\mathbf{p}$ (the blue point) onto the point cloud is computed through Eq. (12), where the set of red points is the current working point cloud c_n after several iterations (K). Then the projection point $\mathbf{p}^*$ (the other blue point) is computed by the DP algorithm along $\mathbf{n}$. (a) The original point set. (b) $K = 1$. (c) $K = 2$. (d) $K = 5$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

For the DP method, Azariadis and Sapidis [8] give the error analysis which is related to the measurement error in the point cloud with the implied error in the solution of the DP problem. Their conclusion is that the error in the solution of the DP problem is always bounded and depends linearly on the error in the measured point cloud. Therefore, the error of our algorithm is also bounded by the measured point cloud after we calculate the projection direction $\mathbf{n}$ through Eq. (12).

2.4. Selecting an appropriate weight function

The weights α_i play a dominant role in the computation of $\mathbf{p}^*$, so they should be chosen carefully. In general, the weights α_i of $\mathbf{p}_i \in \mathcal{C}_N$ should take a larger value when $\mathbf{p}_i$ is closer to the point $\mathbf{p}$, to be projected, and a descending value as the distance from $\mathbf{p}_i$ to $\mathbf{p}$ increases. Azariadis and Sapidis [7,8] give two weight functions. One weight function, which only takes into account the distance between $\mathbf{p}_i$ and $\mathbf{p}$ [7], is

$$\alpha_i = \frac{1}{\|\mathbf{p} - \mathbf{p}_i\|^4}, \quad \alpha \in [0, \infty). \tag{14}$$

The other weight function, which also considers the direction $\mathbf{n}$, is associated to the given point $\mathbf{p}$ [8]. Since this weight function is related to the direction $\mathbf{n}$, it is inapplicable to our case. In our case, $\mathbf{n}$ is unknown before projecting, and its computation utilizes weights α_i.

The weight function used in Eq. (14) is only related to the distance between $\mathbf{p}_i$ and $\mathbf{p}$, so it is suitable for our case. However, when $\mathbf{p} \in \mathcal{C}_N$, i.e. $\mathbf{p}$ coincides with one of the point cloud, α_i does not ensure numerical stability of the computation. We slightly improve Eq. (14) by defining the following weight function

$$\alpha_i = \frac{1}{1 + \|\mathbf{p} - \mathbf{p}_i\|^4}, \quad \alpha \in (0, 1]. \tag{15}$$

In addition, some more complex functions, such as Gaussian function [7,20], can also be considered. One of the future works is to search for better weight functions.

2.5. The implementation details for our LSP algorithm

The implementation of the LSP algorithm is given in Listing (1). The algorithm takes as input a point $\mathbf{p}$ and a point cloud $\mathcal{C}_N$ and computes the projection point $\mathbf{p}^*$ of $\mathbf{p}$ onto $\mathcal{C}_N$. In Ref. [8], the DP problem and Eqs. (2)–(4) require the specification of the projection direction $\mathbf{n}$. Azariadis et al. propose an iterative approach, which relies on the user to define $\mathbf{n}$. In our case, the projection direction $\mathbf{n}$ is determined by optimizing the solutions of all projection directions in Eq. (12). Our algorithm is also achieved through an iterative procedure with the aid of a local variable c_n which is a working sub-cloud of $\mathcal{C}_N$; initially, $c_n \equiv \mathcal{C}_N$. During the projection calculation, c_n is gradually reduced by removing some points from the cloud point $\mathcal{C}_N$. In this way, we are able to reduce the processing time and also increase the accuracy of the projection procedure. The significance of each point in the current c_n is determined by the corresponding weights α_i, calculated using Eq. (15); all weights are stored in a single vector $\mathbf{a} \in \mathbb{R}^n$.

The following parameter selected is similar to the strategy in Ref. [8]. In order to decide which point of the point cloud should be ignored in the K iteration, another local variable named α_{limit} is defined by

$$\alpha_{\text{limit}} = \begin{cases} \alpha_{\text{mean}} + \dfrac{\alpha_{\max} - \alpha_{\text{mean}}}{12 - K}, & K < 11 \\ \alpha_{\text{mean}} + \dfrac{\alpha_{\max} - \alpha_{\text{mean}}}{2}, & \text{otherwise} \end{cases} \tag{16}$$

where two real parameters: $\alpha_{\max}$ and α_{mean}, correspond to the maximum and mean value of the computed weights α_i, respectively.

The main steps of the LSP algorithm are as follows.

1. Compute the projection direction $\mathbf{n}$ through Eq. (12).
2. Compute the projection point $\mathbf{p}^*$ of $\mathbf{p}$ along the computed projection direction $\mathbf{n}$ by using the DP algorithm.
3. Repeat steps 1 and 2 until the projection error is less than a predefined threshold.

In addition, the detailed pseudocode of the above LSP algorithm is given in Listings 1 and 2.

Listing 1: The fundamental **PointLSP** algorithm for implementing our LSP method.

```
Procedure PointLSP($\mathbf{p}, \mathcal{C}_N, \mathbf{n}, \mathbf{p}^*, t$);
  Input:
    $\mathbf{p}$ the test point
    $\mathcal{C}_N \in \mathbb{R}^{N\times 3}$ the given point cloud with
      $N$ points
  Output:
    $\mathbf{n}$ the projection direction vector
      (Eq. (12))
    $\mathbf{p}^*$ the projection point of $\mathbf{p}$
    $t \in \mathbb{R}$ the corresponding solution (Eq. (5))
  Local variables:
    $K$ the current iteration
    $c_n \subseteq \mathcal{C}_N$ the current working point cloud
    $n \in N$ the number of points of $c_n$
    $c_{\text{temp}} \subseteq \mathcal{C}_N$ a point set used for temporary
     storage
    $t^* \in \mathbb{R}$ the current solution computed
     through Eq. (5)
    $\mathbf{a} \in \mathbb{R}^n$ the weight vector
    $\alpha_{\text{mean}} \in \mathbb{R}$ the mean value of $\mathbf{a}$
    $\alpha_{\max} \in \mathbb{R}$ the maximum value of $\mathbf{a}$
    $\alpha_{\text{limit}} \in \mathbb{R}$ computed through Eq. (16)
Begin
  $K := 0$;
  $c_n := \mathcal{C}_N$;
  $t := 0$;
  while($K{+}{+} <$ MAX_ITERS)
  Begin
    OptimProjectToCloud($\mathbf{p}, c_n, \mathbf{n}, \mathbf{p}^*, t^*, \mathbf{a}$);
    if($\|t - t^*\| < \varepsilon$) then return;
    Compute local variables $\alpha_{\max}$, $\alpha_{\text{mean}}$, $\alpha_{\text{limit}}$;
    $c_{\text{temp}} := \emptyset$;
    for($i = 0; i < n; i{+}{+}$) do
        if($\alpha_i >= \alpha_{\text{limit}}$) then $c_{\text{temp}} := c_{\text{temp}} + c_n(i)$;
    $t := t^*$;
    $c_n := c_{\text{temp}}$;
    Compute the number $n$ of points of $c_n$;
    if($n == 0$) then return;
  End;
End
```

Listing 2: The algorithm for estimating the optimal projection of $\mathbf{p}$ onto c_n for each iteration.

```
Procedure OptimProjectToCloud($\mathbf{p}, c_n, \mathbf{n}, \mathbf{p}^*, t, \mathbf{a}$);
  Input:
    $\mathbf{p}$ the current test point
    $c_n \subseteq \mathcal{C}_N$ the current working point cloud
  Output:
    $\mathbf{n}$ the projection direction vector
    (Eq. (12))
    $\mathbf{p}^*$ the optimum projection of $\mathbf{p}$ along
     direction $\mathbf{n}$
    $t \in \mathbb{R}$ the corresponding solution (Eq. (3))
    $\mathbf{a} \in \mathbb{R}^n$ the weight vector
Begin
  Compute weights $\mathbf{a}$ through Eq. (15);
  Compute the projection direction $\mathbf{n}$
     through Eq. (12);
  Rescale the direction $\mathbf{n}$ so that $\mathbf{n}$ is a
     unit vector;
  Compute $\mathbf{p}$ and $t$ using Eq. (5) and Eq. (2);
End
```

According to Listing 1, the proposed **PointLSP** algorithm is initialized by setting the working sub-cloud equal to the initial one ($c_n := \mathcal{C}_N$) and the parameter t equal to 0 ($t := 0$).

Then a loop begins and the given point **p** with the current working cloud c_n is passed into the **OptimProjectToCloud** procedure (Listing 2) producing **n**, **p*** and t^*, which are the current estimation for the projection of **p** onto c_n. **OptimProjectToCloud** also returns the weight vector **a** with respect to the input point **p**. If the difference between the current solution t^* and the previous solution t is less than a predefined threshold ε, the procedure is terminated. The other termination conditions can also be used in the **PointLSP** algorithm. For instance, if the distance between the current projection point and the previous projection point is less than a threshold.

In the opposite case, a new iteration redefines points of c_n, where those weights that are smaller than the current α_{limit} are removed from c_n. Finally, if the number of points of c_n is equal to 0, the algorithm terminates.

Fig. 1 shows an illustration of the LSP procedure.

3. Experimental results

We have applied the LSP algorithm to some point clouds, which are sampled from the B-spline surface (see Fig. 2). The algorithm described above is implemented in C++. The execution time is given in seconds on a Pentium IV 1.70 GHz processor with 512M RAM excluding the time of loading point clouds.

3.1. Testing the effectiveness for noiseless point clouds

This section investigates the numerical performance of the proposed **PointLSP** algorithm by comparing the true result of projecting some test points onto one surface and the approximating result of projecting the same test points onto point clouds sampled from the surface.

The Newton–Raphson method is used to improve the accuracy of the closest point for orthogonal projection onto curves and surfaces. Piegl and Tiller [28] use the Newton–Raphson method to minimize the distance between the test point and the whole NURBS surface. Recently, Piegl and Tiller [29] provide another method to solve the point projection problem for the NURBS surface by decomposing the NURBS surface into quadrilaterals. Ma and Hewitt [22] also present a practical algorithm for computing a good initial value for the Newton–Raphson method. Hu and Wallner [13] propose a second order algorithm for orthogonal projection onto curves and surfaces. In this paper, we use a modified Newton–Raphson method [28] to pre-compute the projection points onto the given B-spline surface. We first show one example for testing the effectiveness of the **PointLSP** algorithm for noiseless point clouds.

Example 1. Fig. 2 shows an example of projecting a set of 20 points sampled from a NURBS curve onto a B-spline surface and point clouds sampled from the surface. Several point clouds $\mathcal{C}_N$, with a varying density, are randomly sampled from the B-spline surface. The number of points is specified from 10,000 to 300,000. The accuracy and execution time for different $\mathcal{C}_N$ are given in Table 1. The experimental data shows that the new method has good approximation.

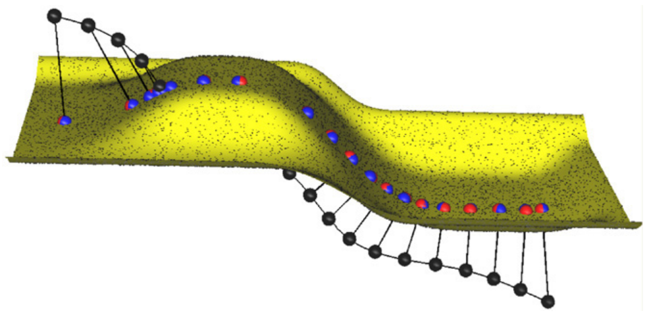


Fig. 2. Comparing the results of projecting points onto the B-spline, and onto point clouds consisting of 10,000 points sampled from the corresponding surface. The projection points are blue and red, respectively. Here the test points (the black points) are sampled from a NURBS curve. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Results of point projection onto noiseless point clouds sampled from the B-spline surface in Fig. 2 with increasing density

N	Iteration	Final sub-cloud	Time (s)	Relative error
10,000	11	4	0.0148437	0.0339159
30,000	10	19	0.04375	0.0274106
60,000	10	26	0.084375	0.0188686
100,000	11	45	0.146094	0.0121798
140,000	11	62	0.213281	0.0102712
200,000	10	127	0.295312	0.00835032
300,000	10	149	0.482031	0.00673906

Table 1 lists the size N of each point cloud $\mathcal{C}_N$, the number of iterations, the size of the final sub-cloud, the execution time, and the accuracy. All data are the average of projecting 20 equally spaced points onto the given B-spline surface and the corresponding $\mathcal{C}_N$. Suppose $\mathbf{q}_i$ $(i = 1, \dots, l)$ are the test points sampled from the NURBS curve, where l is the number of some test points sampled from the curve. We obtain 20 *true* projection points $\mathbf{q}_i^S$ by projecting $\mathbf{q}_i$ onto the given B-spline surface using the modified Newton–Raphson method, and 20 points $\mathbf{q}_i^*$ by projecting $\mathbf{q}_i$ per point cloud $\mathcal{C}_N$ using our **PointLSP** algorithm. The accuracy for each $\mathbf{q}_i$ is the average error, which is computed by

$$\frac{\sum_{i=1}^{l} \frac{\|\mathbf{q}_i^* - \mathbf{q}_i^S\|}{\|\mathbf{q}_i - \mathbf{q}_i^S\|}}{l}. \tag{17}$$

3.2. Testing the effectiveness for noisy point clouds

Models created from 3D scanners usually contain noise [14]. Noise tends to increase point-error (or “point cloud thickness”) [8]. Next, we give one experimental result demonstrating the approximation influences of the **PointLSP** algorithm on noisy point clouds.

Example 2. For this experiment, a series of initial point clouds are generated by sampling from the B-spline surface in Fig. 2. Then, noisy point clouds are produced by adding Gaussian

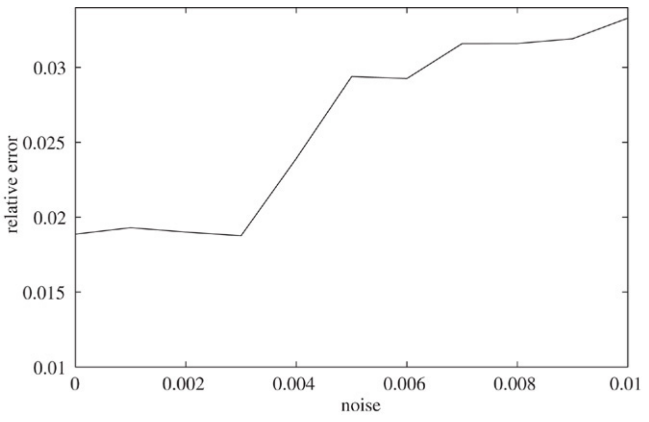


Fig. 3. Error curve of the **PointLSP** algorithm for a series of point clouds with increasing noise ρ. The solid line corresponds to the error curve of projecting points onto point clouds generated through the surface in Fig. 2.

noise to each point cloud along the normals with increasing variances ρ, as proposed by Azariadis and Sapidis [8]. $\rho \in [0, 0.01]$ is the point cloud's "thickness factor" with respect to the surface bounding box, where $\rho = 0$ corresponds to the initial point cloud sampled from the corresponding surface. The number N of points for each point cloud is equal to 60,000. Fig. 3 shows the error curve, which is relative to the thickness factor ρ of the produced noisy point clouds, respectively in Fig. 2. In Fig. 3, the curve shows that the projection error is increased accordingly when the thickness ρ is increased. This test yields incorrect results if the noise is larger than the sampling density of a point cloud.

4. Applications in reverse engineering

4.1. Thinning a point cloud

A given point cloud might have erroneous point locations (i.e. noise). The problem of noise can be handled by projecting the points onto the point cloud surface themselves [3,18]. The result of the projection procedure is a thin point cloud [3], and the procedure can be called the *thinning* operation [18]. The thinning operation is necessary for reconstructing or fitting curves and surfaces from unorganized and noisy points [18,32].

The algorithm of thinning a point cloud $\mathcal{C}_N$ consists of two steps.

1. Project each point in $\mathcal{C}_N$ onto $\mathcal{C}_N$ and obtain the projection position through the **PointLSP** algorithm.
2. Move each point in $\mathcal{C}_N$ to its projection position.

Fig. 4 shows an example of thinning a 2D point cloud sampled from a closed B-spline curve. In Fig. 5, another example of thinning a 3D point cloud sampled from an open helix curve is illustrated. Fig. 6 shows a complex example of thinning a dense point cloud, which is acquired by 3D scanning devices and contains additional noise.

From an de-noising view, the presented application in point-cloud thinning is similar to "noise filtering". Our goal is also to remove noise and thin the point cloud. However, our thinning method begins by considering the problem of finding a point

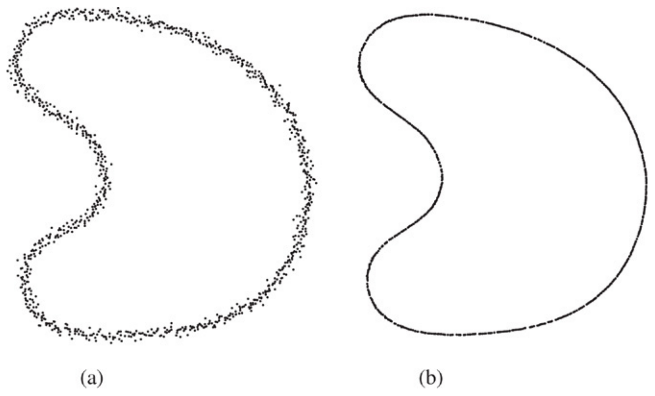


Fig. 4. Thinning a 2D point cloud using our LSP algorithm. (a) 1000 points (with noise) sampled from a closed B-spline curve. (b) The result of thinning the point cloud.

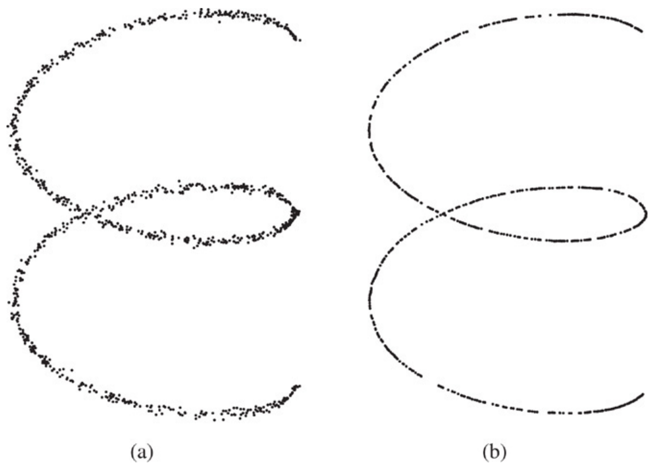


Fig. 5. Thinning a 3D point cloud using our LSP algorithm. (a) 1000 points (with noisy) sampled from an open helix curve. (b) The result of thinning the point cloud.

$\mathbf{p}^*$ along a projection direction $\mathbf{n}$ such that the sum of the squared distances between $\mathbf{p}^*$ and points of the point cloud is as small as possible. Therefore, the new method is an optimization method in a least-squares sense under the squared distances. However, most "noise filtering" methods mainly derive from image processing techniques [14] or differential geometry [17].

4.2. Point normal estimation

The estimation of the normal vector at a discrete data point in a scanned point data set is important for the CAD technologies when the continuous CAD model representation is not available [25]. It is a fundamental problem in many CAD and CG applications, such as smoothing [17] and surface reconstruction [12]. Many researchers have attempted to estimate the normal vectors of discrete points by locally fitting parametric or implicit surfaces [4,12,17,25]. In general, the estimation procedure is as follows [25].

1. Identify the applicable neighborhood points for estimating the normal vector;

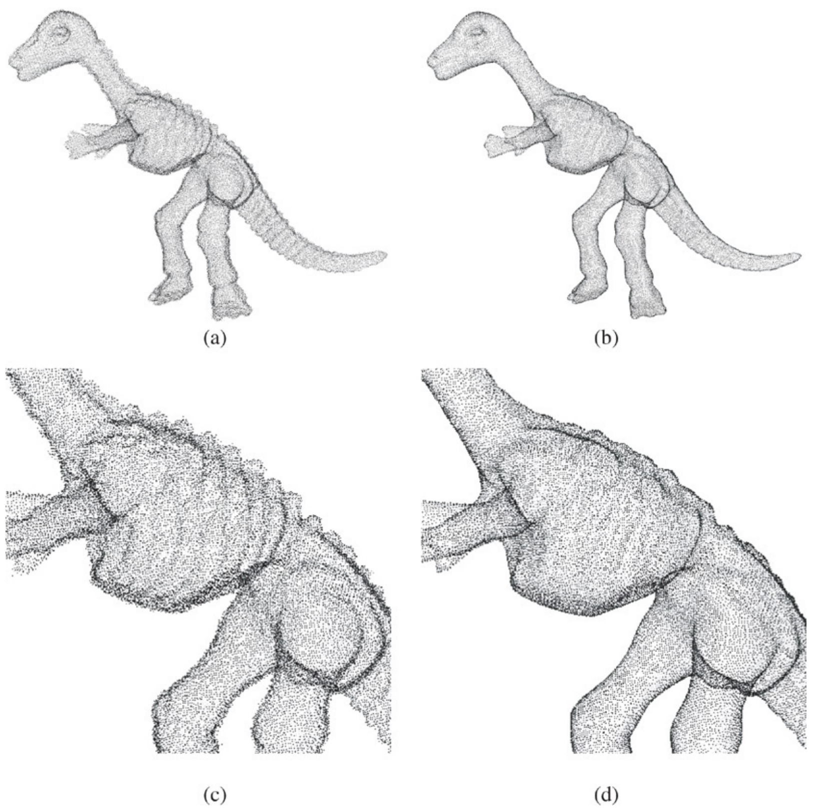


Fig. 6. Thinning a point cloud representing the dinosaur, which is acquired by 3D scanning devices and contains additional noise. (a) A set of $70K$ points. (b) The result of thinning the point cloud. (c) A magnified view of (a). (d) A magnified view of (b).

2. Estimate the normal vector based on points in the local neighborhood;
3. Establish the inner and outer directions of the normal vector.

Many researchers have used a fixed number of Euclidean nearest neighborhood data points, i.e. k-nearest neighborhoods, to estimate the normal vector at a given data point [12,17]. Those k-nearest neighborhoods were used to fit the tangent plane or the local quadric surface at the point [12]. The intended k neighborhood points may not be suitable to estimate the normal vector when the data is not in the fixed grid format of range images [17]. Another class of approaches is based on the global or local Delaunay triangulation to obtain the reconstructed mesh [5]. Then, the point normal of the mesh can be approximated instead of the normal of the point cloud. However, the reconstruction-based method is not suitable for noisy and dense point clouds because the true surface representation is unknown and the construction of a valid global or local Delaunay triangulation mesh is not always possible.

The LSP method can be directly used to estimate the point normal of an unorganized point cloud. The new method does not need to specify the size of the neighborhood and it is robust for dense point clouds. The normal estimation procedure of the whole point cloud consists of three steps.

1. Obtain the new position $\mathbf{p}_i^*$ for each point $\mathbf{p}_i$ in the point cloud $\mathcal{C}_N$ by our projection method.
2. Rescale the new vector $\mathbf{n}_i = \mathbf{p}_i^* - \mathbf{p}_i$ so that $\mathbf{n}_i$ is an unit vector, and then the unit vector $\mathbf{n}_i$ is chosen as the initial normal vector for $\mathbf{p}_i$ (see Fig. 7(a)). If $\mathbf{p}_i^*$ is equal to $\mathbf{p}_i$, $\mathbf{n}_i$ only can be computed by other methods, such as Ref. [12].
3. Establish the inner and outer directions of the normal vector (see Fig. 7(b)).

Fig. 7 shows an example of point normal estimation for a dense point cloud, where the red lines denote the unit normal vectors. The original point cloud can be found in Fig. 11(a). In Fig. 7(b), most red lines look good to approximate normal vectors of the point cloud though several red lines are imperfect.

In addition, one of the most difficult problems in normal vector estimation is the establishment of a consistent inner and outer orientation (i.e. Step 3), and the presented application does not contribute in this critical issue. OuYang and Feng [25] compare several algorithms for establishing the inner and outer directions. In our implement, we only use a simple strategy similar to Hoppe et al.'s method [12], in which the normal direction is propagated from an initial normal. The propagation method requires that all pairs of sampled points are sufficiently close and the sampled surface is smooth. Suppose two points $\mathbf{p}_i$ and $\mathbf{p}_j$ in a point cloud $\mathcal{C}_N$ are close, so the corresponding normals $\mathbf{n}_i$ and $\mathbf{n}_j$ are nearly parallel, i.e. $\mathbf{n}_i \cdot \mathbf{n}_j \approx 1$; otherwise, either $\mathbf{n}_i$ or $\mathbf{n}_j$ should be flipped. To assign an initial normal, the unit normal of the point whose center has the largest z coordinate is forced to point toward the $+z$ axis.

4.3. Projecting curves onto a point cloud

Designing curves onto a point cloud is an important problem for many applications in reverse engineering (RE), such as

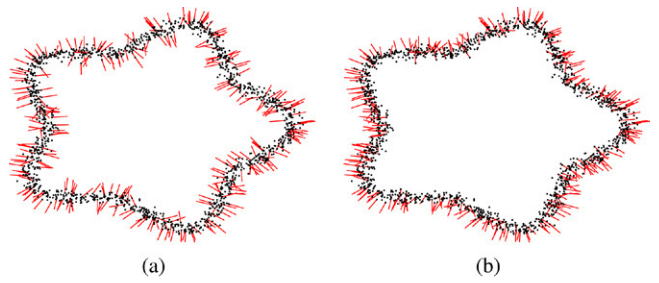


Fig. 7. An example of point normal estimation. (a) Projecting each point onto the point cloud and obtaining normal vectors (red lines). (b) Adjusting normal vectors. Here, those normal vectors of random points on the point cloud are chosen for display. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

segmentation, parameterization [7], reconstruction, and point-based modeling [8]. A generic approach of reconstructing surfaces from an unorganized point cloud first needs to divide the point cloud into subsets which correspond to certain regions of the object surface where a single surface can be fitted [7]. In some RE software, users usually finish the segmentation through designing interactively curves onto a point cloud, but this step is not easy. Azariadis and Sapidis [8] develop a new technique, called drawing curves onto a cloud of points, for point-based modeling. This technique is based on the DP method of projecting points onto a point cloud, whereas the projection vectors must be specified through a graphics interface tool.

In this section, we improve Azariadis et al.'s method [8] by using the LSP method instead of the DP method. The improved algorithm of curve projection consists of four steps as follows.

1. Discretize the test curve into some line segments through sampling the curve.
2. Project those line segments onto the point cloud and obtain a polyline. This step is similar to Ref. [8], and the only difference is that our LSP method replaces their DP method.
3. Smooth the projection polyline. This step follows Azariadis's algorithm about smoothing projected segments [8].
4. Designing the B-spline curve onto the point cloud by interpolating the endpoints of the smoothed polyline.

Fig. 8 shows an example of projecting two curves onto a point cloud sampled from a cylinder model. Fig. 9 shows the other example of projecting a B-spline curve onto a point cloud which is acquired from a man model by a 3D scanning device.

4.4. Other applications

Besides those applications proposed in previous several sections, the LSP method can also be applied in some high-grade domains.

One application is to reconstruct or approximate curves from a point cloud [18]. First, the noisy point cloud is thinned by our LSP algorithm. Then the projection point set is approximated with a curve. Here, we do not repeat the application of curve reconstruction from a point cloud [18]. Based on the method of projecting curves onto a point cloud, we give a simple and

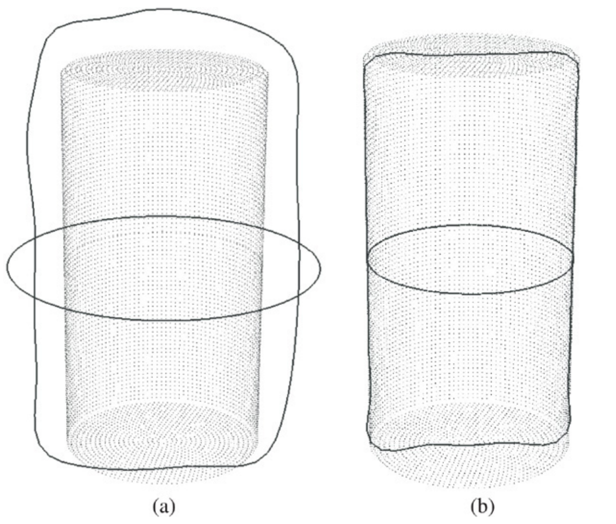


Fig. 8. Projecting two curves onto a point cloud of a cylinder. (a) 12,516 points sampled from a cylinder surface. (b) The result of projecting two curves onto the point cloud.

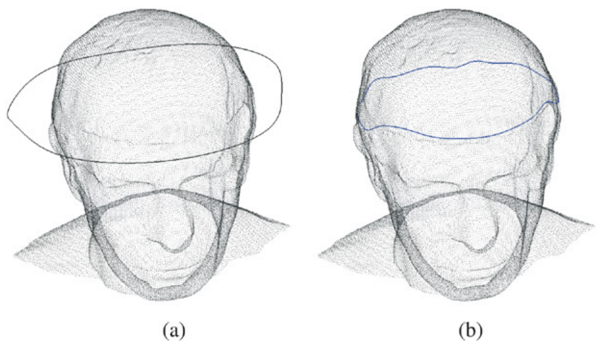


Fig. 9. Projecting a B-spline curve onto a point cloud of a man model. (a) The initial 3D curve and point cloud (62,202 points). (b) The result of projecting the curve onto the point cloud.

interesting algorithm for approximating a digital point cloud extracted from a 2D CT image.

1. Extract a digital point cloud from a 2D CT image, where the point cloud is unordered and possibly dense.
2. Specify a proper initial curve of a B-spline approximating curve.
3. Project the curve onto the point cloud.
4. Repeat step 3 until a pre-specified error threshold is satisfied.

In the first step, the digital point cloud can be extracted by some image processing techniques, such as choosing some points whose gray values are greater than or equal to a threshold. The second step, i.e. the choice of the initial shape specification, is very important for obtaining a good approximation. Wang et al. [32] use a quad-tree cell partition to obtain a collection of connected cells of possibly different sizes that cover the data points. Then a sequence of feature points of these covering cells are extracted as the control points of an initial B-spline curve. For a closed and relatively simple target shape, we specify a boundary circle as the initial approximating curve. Fig. 10 shows an example of approximating the contour of a CT image with a B-spline curve after one iteration, where the image can

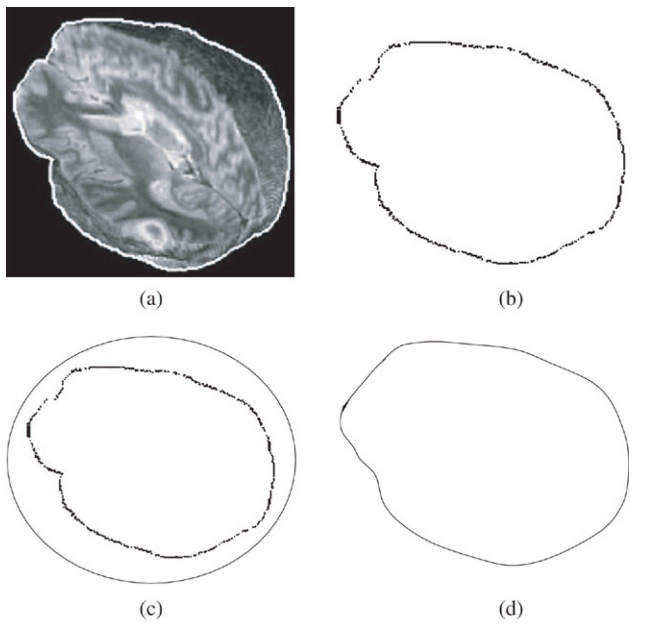


Fig. 10. Approximating the contour of a CT image with a B-spline curve. (a) The target CT image. (b) A target point cloud obtained from the CT image using image processing techniques. (c) The initial approximating curve (circle). (d) The approximated result generated by projecting the initial curve onto the target point cloud after one iteration.

be found in Ref. [33] and the white contour in Fig. 10(a) is drawn manually with the aid of image processing software. The number of control points of an approximation curve may be inadequate or redundant. As an alternative to our approach, one could use a method of inserting and deleting control point for the resulting B-spline curve [33].

Another class of applications is parameterization of point clouds [7]. Dynamic base surfaces (DBS) [7] are developed to solve the parameterization problem for unstructured point clouds. Then, the resulting parameterization is utilized for high-accuracy surface fitting [7]. According to the original DBS methodology, an initial DBS is constructed by interpolating a given closed boundary topologically equivalent to a rectangle with a bi-cubic Coons surface. A grid of points is derived along the two parametric directions of the DBS and then it is projected onto the point cloud surface by minimizing a set of distance metrics. The projection direction of each grid point is set equal to the corresponding DBS normal direction. However, the defined projection direction might result in the slow convergence for the complex initial DBS. Unlike Azariadis's method [7], we replace the DP method as our LSP algorithm, which can avoid any specification of the projection vector.

The LSP algorithm can also be applied in the ICP algorithm for shape registration of point clouds [11,23]. It seems reasonable for shape registration of dense point clouds in order to ensure the convergence of the ICP algorithm.

5. Comparison with the moving least-squares (MLS) method

The moving least-squares (MLS) method can also be used to thin a point cloud [3,18–20]. The thinning procedure based on the MLS method is reviewed as follows [18]. For each data point $\mathbf{p}_i$ in a point set $\mathcal{C}_N$, a simple curve or surface that fits some neighborhood points of the point $\mathbf{p}_i$ is first computed using a weighted regression scheme. Then, the point $\mathbf{p}_i$ is moved to a new position on this approximated curve or surface. However, the MLS method is sensitive to the size of the chosen neighborhood of each point $\mathbf{p}_i$ for a local polynomial approximation [3,18]. Let H be the radius of the circle with the center $\mathbf{p}_i$. If H is too small, the local regression does not reflect the thickness of the point cloud, resulting in the thinned points are scattered (see Fig. 11(b)). If H is too large, the local regression may contain some unwanted points, which may occur the failure of the MLS algorithm [18]. To prevent the effects, Lee [18] has used the Euclidean minimum spanning tree (EMST) to make the connectivity of the point elements. Lee's improved MLS method needs the Delaunay triangulation of the point set, and it is difficult to extend his method to 3D point sets. Since MLS is based on local fitting, it is also a reconstruction-based approach. The reconstruction-based drawback for the projection operation has been mentioned in Section 1. Our LSP method can be applied to thin 2D and 3D point sets without any explicit or implicit surface reconstruction procedure, and can get good results (see Section 4.1). Furthermore, no triangulation and complex data structures are used in the new method.

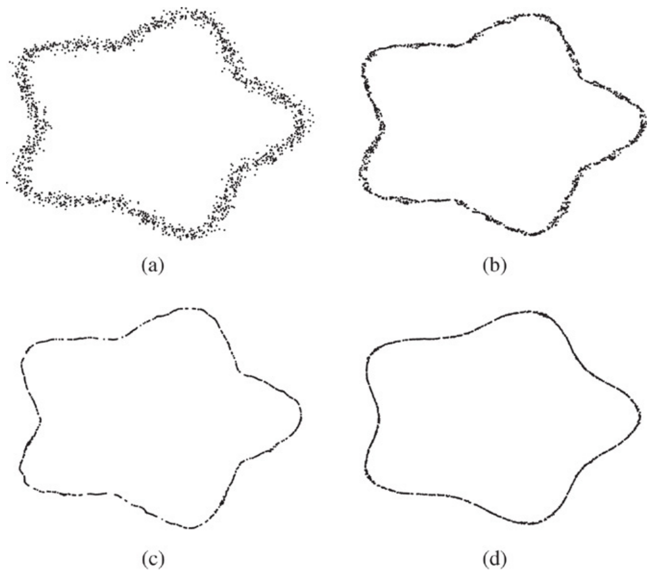


Fig. 11. Comparison with the MLS method. (a) The original dense 2D point set (1511 points). (b) The result of the MLS method by choosing 25 neighborhood points after one iteration without EMST. (c) The result of the MLS method after two iterations. (d) The result of the new method.

In order to compare the new method with the MLS method, Fig. 11 shows a thinned example for a 2D point set. The thinned point set using the MLS method is shown in Fig. 11(b), whereas it still contains some noisy points. An iteration scheme [18] for refining the point set is also presented to improve the result, but there still remain some difficulties (see Fig. 11(c)). The thinning procedure based on the LSP method can solve those problems caused by the MLS method (see Fig. 11(d)), because the size of the current neighborhood used for computing the projection position is controlled by the iterative error ε (see Section 2.5).

In some sense, the new method still has the same drawback with the MLS method. That is that the thinned point set does not represent the shapes of the original point set near the end points of open curves or surfaces.

6. Conclusions

We have presented a simple and efficient algorithm, the so-called LSP, for projecting points onto a point cloud. The new method is based on the directed projection algorithm [7, 8] and does not need any complex data structure. The major advantage in the LSP method is that the projection vector is computed automatically. Our experiments show that the LSP method is fast, robust and obtains high accuracy without requiring an explicit or implicit reconstruction of the underlying surface from the point cloud. We have also shown the efficiency of the proposed method for some applications in thinning, normal estimation, drawing curve, reconstruction and others. We believe that there are numerous other applications that can benefit from the LSP method.

Many areas of future works and potential applications are envisaged. In our algorithm, we assume that the weight function is not correlative with the projection direction so that the optimization function $t(\mathbf{n})$ is simple. In the future we plan to optimize the projection direction together with the corresponding weight function. This is an area for future (and challenging) research for projection problems.

In addition, our application on projecting a curve onto a point cloud is based on the method proposed in Ref. [8]. Our application looks probably very useful for automatic segmentation of point clouds for reconstruction. There is still one open problem in this area: how to discretize effectively a continuous curve in such a way that its projection onto the point cloud is a “digital curve that lies on that cloud surface”?

Acknowledgements

We would like to thank Dr. Heng Liang from Department of Mathematics at Tsinghua University for many helpful discussions, and Wenke Wang for some valuable comments during our work. The authors appreciate the comments and suggestions of three anonymous reviewers. The research was supported by Chinese 973 Program (2004CB719400), and the National Science Foundation of China (60403047, 60533070). The third author was supported by the project sponsored by a Foundation for the Author of National Excellent Doctoral Dissertation of PR China (200342), and a Program for New Century Excellent Talents in University (NCET-04-0088).

References

[1] Adamson A, Alexa M. Ray tracing point set surfaces. In: Proceedings of shape modeling international 2003. 2003. p. 272–79.

[2] Alexa M, Behr J, Cohen-Or D, Fleishman S, Levin D, Silva CT. Point set surfaces. In: Proceedings of IEEE visualization’01. 2001. p. 21–8.

[3] Alexa M, Behr J, Cohen-Or D, Fleishman S, Levin D, Silva CT. Computing and rendering point set surfaces. IEEE Transactions on Visualization and Computer Graphics 2003;9(1):3–15.

[4] Alexa M, Adamson A. On normals and projection operators for surfaces defined by point sets. In: Proceedings of eurographics symposium on point-based graphics. 2004. p 149–56.

[5] Amenta N, Bern M, Kamvysselis M. A new Voronoi-based surface reconstruction algorithm. In: Proceedings of SIGGRAPH’98. 1998. p. 415–21.

[6] Amenta N, Kil Y. Defining point-set surfaces. In: Proceedings of SIGGRAPH’04. 2004. p. 264–70.

[7] Azariadis P. Parameterization of clouds of unorganized points using dynamic base surfaces. Computer-Aided Design 2004;36(7):607–23.

[8] Azariadis P, Sapidis N. Drawing curves onto a cloud of points for point-based modelling. Computer-Aided Design 2005;37(1):109–22.

[9] Besl PJ, McKay ND. A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 1992;2(6): 239–56.

[10] Fleishman S, Cohen-Or D, Silva CT. Robust moving least-squares fitting with sharp features. In: Proceedings of SIGGRAPH’05. 2005. p. 544–52.

[11] Gelfand N, Mitra NJ, Guibas L, Helmut P. Robust global registration. In: Proceedings of eurographics symposium on geometry processing. 2005. p. 197–206.

[12] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Surface reconstruction from unorganized point. In: Proceedings of SIGGRAPH’92. 1992. p. 71–8.

[13] Hu SM, Wallner J. A second order algorithm for orthogonal projection onto curves and surfaces. Computer Aided Geometric Design 2005;22(3): 251–60.

[14] Jones T, Durand F, Desbrun M. Non-iterative, feature-preserving mesh smoothing. In: Proceedings of SIGGRAPH’03. 2003. p. 943–9.

[15] Klein J, Zachmann G. Point cloud collision detection. Computer Graphics Forum 2004;23(3):567–76.

[16] Kobbelt L, Botsch M. A survey of point-based techniques in computer graphics. Computers and Graphics 2004;28(6):801–14.

[17] Lange C, Polthier K. Anisotropic smoothing of point sets. Computer Aided Geometric Design 2005;22(7):680–92.

[18] Lee IK. Curve reconstruction from unorganized points. Computer Aided Geometric Design 2000;17(2):161–77.

[19] Levin D. The approximation power of moving least-squares. Mathematics of Computation 1998;67:1517–31.

[20] Levin D. Mesh-independent surface interpolation. In: Brunnett, Hamann, Mueller, editors. Geometric modeling for scientific visualization. Springer-Verlag; 2003. p. 37–49.

[21] Liu Y-S, Yong J-H, Zhang H, Yan D-M, Sun J-G. A quasi-Monte Carlo method for computing areas of point-sampled surfaces. Computer-Aided Design 2006;38(1):55–68.

[22] Ma YL, Hewitt WT. Point inversion and projection for NURBS curve and surface: Control polygon approach. Computer Aided Geometric Design 2003;20(2):79–99.

[23] Mitra NJ, Gelfand N, Pottmann H, Guibas L. Registration of point cloud data from a geometric optimization perspective. In: Proceedings of eurographics symposium on geometry processing. 2004. p. 23–32.

[24] Ohtake Y, Belyaev A, Alexa M, Turk G, Seidel HP. Multi-level partition of unity implicits. In: Proceedings of SIGGRAPH’03. 2003. p. 463–70.

[25] OuYang D, Feng HY. On the normal vector estimation for point cloud data from smooth surfaces. Computer-Aided Design 2005;37(10):1071–9.

[26] Pauly M, Kobbelt L, Gross M. Multiresolution modeling of point-sampled geometry. ETH Zurich technical report, #378. September 16, 2002.

[27] Pauly M, Gross M, Kobbelt L. Efficient simplification of point-sampled surfaces. In: Proceedings of IEEE visualization’02. 2002. p. 163–70.

[28] Piegl LA, Tiller W. The NURBS book. Berlin: Springer; 1995.

[29] Piegl LA, Tiller W. Parameterization for surface fitting in reverse engineering. Computer-Aided Design 2001;33(8):593–603.

[30] Schaufler G, Jensen HW. Ray tracing point sampled geometry. In: Proceedings of the 11th eurographics workshop on rendering. 2000. p. 319–28.

[31] Wald I, Seidel H-P. Interactive ray tracing of point based models. In: Proceedings of 2005 symposium on point based graphics. 2005.

[32] Wang W, Pottmann H, Liu Y. Fitting B-spline curves to point clouds by squared distance minimization. ACM Transactions on Graphics 2006; 25(2):214–38.

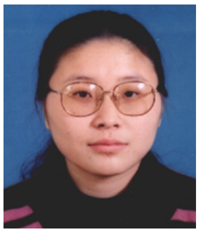
[33] Yang HP, Wang W, Sun JG. Control point adjustment for B-spline curve approximation. Computer-Aided Design 2004;36(7):639–52.

Yu-Shen Liu is currently a postdoctoral scholar at PRECISE (the Purdue Research and Education Center for Information System in Engineering), Purdue University. He received his Ph.D. in the Department of Computer Science and Technology at Tsinghua University of China in 2006. He received his B.S. in Mathematics from Jilin University of China in 2000. His research interests are computer-aided design and computer graphics.

Jean-Claude Paul is a senior researcher at CNRS and INRIA (France), and currently visiting professor at Tsinghua University. He received his Ph.D. in Mathematics from Paris University in 1976. His research interests include numerical analysis, physics-based modeling and computer-aided design.

Jun-Hai Yong is an associate professor in School of Software at Tsinghua University, China. He received his B.Sc. and Ph.D. in Computer Science from the Tsinghua University, China, in 1996 and 2001, respectively. He held a visiting researcher position in the Department of Computer Science at Hong Kong University of Science and Technology in 2000. He was a postdoctoral fellow in the Department of Computer Science at the University of Kentucky from 2000 to 2002. His research interests include computer-aided design, computer graphics, computer animation, and software engineering.

Pi-Qiang Yu is an assistant professor in School of Computer and Information Technology at Beijing Jiaotong University, China. He received his B.Sc. and Ph.D. degrees in Computational Mathematics in 1997 and 2002 both from Dalian University of Technology of China, and he finished his post-doctor research at Tsinghua University in 2004. His current research interests are computer graphics and computer-aided geometric design.



Hui Zhang is an assistant professor in School of Software at Tsinghua University, China. She received her B.Sc. and Ph.D. in Computer Science from the Tsinghua University of China in 1997 and 2003, respectively. Her research interests are computer-aided design and computer graphics.

Jia-Guang Sun is a professor in the Department of Computer Science and Technology at Tsinghua University, China. His research interests are computer graphics, computer aided design, computer-aided manufacturing, product data management and software engineering.

Karthik Ramani is a professor in the School of Mechanical Engineering at Purdue University. He earned his B.Tech. from the Indian Institute of Technology, Madras, in 1985, an M.S. from The Ohio State University, in 1987, and a Ph.D. from Stanford University in 1991, all in Mechanical Engineering. He has worked as a summer intern in Delco Products, Advanced Composites, and as a summer faculty intern in Dow Plastics, Advanced Materials. He was awarded the Dupont Young Faculty Award, the National Science Foundation Research Initiation Award, the National Science Foundation CAREER Award, the Ralph Teetor Educational Award from the Society of Automotive Engineers, Outstanding Young Manufacturing Engineer Award from the Society of Manufacturing Engineers, and the Ruth and Joel Spira Award for Outstanding contributions to the Mechanical Engineering Curriculum. In 2002, he was recognized by Purdue University through a University Faculty Scholars Award. In 2005 he won the Discovery in Mechanical Engineering Award for his work in shape search. He has developed many successful new courses Computer-Aided Design and Prototyping, Product and Process Design and codeveloped an Intellectual Property course. He founded the Purdue Research and Education Center for Information Systems in Engineering (PRECISE) and ToolingNET, a collaborative 21st century project funded by the State of Indiana. A major area of emphasis in his group is shape representations for search and configuration in both engineering and biology. His research is funded by the NSF, DLA/Army, and the National Institute of Health. He also chairs an ASME Computers and Information in Engineering Committee and is on the editorial board of Computer-Aided Design.