Available online at www.sciencedirect.com
ScienceDirect
Computer-Aided Design 40 (2008) 537–553
COMPUTER-AIDED DESIGN
www.elsevier.com/locate/cad

# An extension on robust directed projection of points onto point clouds

Ming-Cui Du, Yu-Shen Liu*

*Tsinghua University, Beijing 100084, PR China*

Received 3 August 2007; accepted 21 January 2008

**Abstract**

Azariadis and Sapidis [Azariadis PN, Sapidis NS. Drawing curves onto a cloud of points for point-based modelling. Computer-Aided Design 2005;37(1):109–22] introduced a novel method of point *directed projection* (DP) onto a point cloud along an associated projection vector. This method is essentially based on an idea of least sum of squares by making use of a weight function for bounding the influence of noise. One problem with their method is the lack of robustness for outliers. Here, we present a simple, robust, and efficient algorithm: *robust directed projection* (RDP) to guide the DP computation. Our algorithm is based on a robust statistical method for outlier detection: least median of squares (LMS). In order to effectively approximate the LMS optimization, the forward search technique is utilized. The algorithm presented here is better suited to detect outliers than the DP approach and thus finds better projection points onto the point cloud. One of the advantages of our algorithm is that it automatically ignores outliers during the directed projection phase.

*Keywords:* Directed projection; Robust statistics; Least median of squares; Forward search; Point clouds

## 1. Introduction

Point projection is one of basic problems in many point-based modelling techniques [1–3]. It can be used in the *moving least squares* (MLS) technique [4], the point cloud collision detection [5], the ICP (*iterative closest point*) algorithm for shape registration [6], parameterization of unstructured point clouds [1], drawing curves onto point clouds [2], and others. The reader may consult Refs. [1–3] for a detailed review of the related works.

Traditionally, there are many methods for projecting a point onto a parametric or implicit surface [7]. A review of many available methods for computing the projection point onto the parametric or implicit surface is beyond the scope of this paper. The reader may consult Ref. [7] for detailed expositions. The existing methods 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 and then use common techniques to project points onto the reconstructed surface. There are several limitations on indirect methods based on explicit or implicit surface reconstruction. First, it is a non-trivial task to build a surface representation that is faithful to the point sets, and an error 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.

Based on the MLS method, Alexa et al. [8,9] define a point set surface approximated locally for a certain neighborhood by a polynomial, and then project the point near the point set onto this surface. Some improved MLS methods are also proposed [4]. Their methods are essentially reconstruction-based methods and need a certain neighborhood of the test point. In addition, since it is common for a point set acquired by 3D scanning devices to include some under-sampled areas [2], it becomes difficult to find a suitable neighborhood for performing the MLS projection operation in those 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

* Corresponding author. Tel.: +86 10 62795455; fax: +86 10 62795460.
*E-mail addresses:* dmc02@mails.tsinghua.edu.cn (M.-C. Du), liuyushen00@gmail.com, liu28@purdue.edu (Y.-S. Liu).

0010-4485/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2008.01.010

samples, resulting in larger approximation errors. Another possible solution is to find $k$-nearest neighborhoods of the test point, and then compute the average of the $k$ neighborhood points. 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, Azariadis and Sapidis [2] introduced a novel algorithm of point *directed projection* (DP) onto a point cloud along an associated projection vector in a point-based CAD/CAM system. The algorithm in Ref. [2] is simple and fast due to operating directly on the point cloud without any explicit or implicit surface reconstruction procedure. Furthermore, no triangulation and complex data structure is used in their algorithm. Their algorithm has two advantages due to an iterative property with weight functions [2]. The first advantage is that no particular neighborhood is fixed for the test point, so it is suitable for point sets with noise and irregular samples. The second advantage is that an appropriate point cloud error function is given to solve the problem of directed projection onto a point cloud. The DP algorithm is essentially based on an idea of least sum of squares by making use of a weight function for bounding the influence of noise. However, one main problem with their algorithm is a lack of robustness for outliers. In this paper, we present an extension of the DP algorithm for outlier detection by combining robust statistics methods.

### 1.1. The review of the DP algorithm

We review the DP algorithm by considering the following problem. Let $\mathcal{C}_N = \{\mathbf{p}_i | i = 1, \ldots, N\}$ be a set of unorganized data points, where $\mathbf{p}_i = (x_i, y_i, z_i)^T$ is a 3D vector. 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 a *point cloud* or *scattered data points* in the literature, may be dense and have non-uniform distribution with considerable noise. Such data point sets are common in reverse engineering processes, where the surface of a sculptured object is measured by a 3D scanner or by a coordinate measurement machine. Let $\mathbf{p}$ be an arbitrary 3D point and $\mathbf{n}$ an associated projection vector. The DP problem is to find the projection point of $\mathbf{p}$ onto the point-sampled surface $S$, which consists of the point cloud $\mathcal{C}_N$, in the direction $\mathbf{n}$ by minimizing an error function.

Azariadis and Sapidis [1,2] proposed an error function and used it to solve the DP problem. Their DP algorithm can be summarized two phases [1,2]: (1) defining an error function for measuring the distance between the point to be projected and the point cloud; (2) projecting the point of interest onto the point cloud by minimizing the above error function. We first review the detail of the error function. Consider a point cloud $\mathcal{C}_N$ and a test point $\mathbf{p} = (x, y, z)^T$ with an associated projection vector $\mathbf{n} = (n_x, n_y, n_z)^T$. Each $\mathbf{p}_i$ in $\mathcal{C}_N$ is associated to a positive weight $\alpha_i$. Let $\mathbf{p}^* = (x^*, y^*, z^*)^T$ be the projection point of $\mathbf{p}$ onto $\mathcal{C}_N$ for the DP problem. $\mathbf{p}^*$ is computed 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 the given weights $\{\alpha_i\}$, which will be referred to in Section 1.2.1, $\mathbf{p}^*$ can be written as

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

Then, by substituting Eq. (2) into Eq. (1), the solution of minimizing Eq. (1) is [1]

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

where $\cdot$ denotes dot product and

$$\lambda = \frac{c_1 n_x + c_2 n_y + c_3 n_z}{c_0}, \quad \text{and}$$

$$c_0 = \sum_{i=1}^{N} \alpha_i,\ c_1 = \sum_{i=1}^{N} \alpha_i x_i,\ c_2 = \sum_{i=1}^{N} \alpha_i y_i,\ c_3 = \sum_{i=1}^{N} \alpha_i z_i. \tag{4}$$

Intuitively, the projection process defined by Eqs. (2) and (3) can be regarded as intersecting a given point cloud with the semi-infinite line defined by $\mathbf{p}$ and $\mathbf{n}$ [2]. Independently, Liu et al. [10] have given a simple result for the similar problem of intersecting a line with a point cloud.

### 1.2. Strengths and weaknesses of the DP algorithm

In the DP algorithm, the weights $\alpha_i$ play a dominant role in the computation of $\mathbf{p}^*$, so they should be chosen carefully [2]. If the weights $\alpha_i \equiv 1$ in Eq. (1), the DP method corresponds to the method of *least squares*. In spite of mathematical beauty and computational simplicity of the method of least squares, this estimator is now being criticized more and more for its dramatic lack of robustness [11]. Indeed, a single sample with a large error, an *outlier*, can have an arbitrarily large effectiveness on the estimate. To overcome the lack of robustness using the method of least squares, some robust methods might be used for improving it. The commonly used techniques for excluding outliers are to make use of some weight functions for bounding the influence of outliers. Jones et al. [12] and Fleishman et al. [13] have applied the bilateral filter to mesh denoising, which gives low weight values to outliers using the Gaussian weight function.

#### 1.2.1. Analyzing the influence of two weight functions

The weight $\alpha_i$ of $\mathbf{p}_i \in \mathcal{C}_N$ should be correlative with the distance between $\mathbf{p}_i$ and the projection $\mathbf{p}^*$, such as $\alpha_i = 1/\|\mathbf{p}_i - \mathbf{p}^*\|^4$. However, this results in a complex nonlinear optimization problem for Eq. (1) due to the unknown $\mathbf{p}^*$ [3]. Azariadis and Sapidis [1,2] assumed that all weights $\alpha_i$ are pre-computed, so the DP problem is simplified as a linear optimization problem of least squares. In general, the weight $\alpha_i$ can be considered to take a larger value when $\mathbf{p}_i$ is closer to the test point $\mathbf{p}$ and a decreasing value as the distance from $\mathbf{p}_i$ to $\mathbf{p}$ increases in Eq. (1). Azariadis and Sapidis [1,2] gave two weight functions for bounding the influence of noise. One weight function, which only takes into account the distance between $\mathbf{p}_i$ and $\mathbf{p}$ [1], is

$$\alpha_i = \frac{1}{\|\mathbf{p}_i - \mathbf{p}\|^4}, \quad \alpha_i \in [0, \infty). \tag{5}$$

The other weight function, which also considers the direction $\mathbf{n}$ associated to the given point $\mathbf{p}$ [2], is

$$\alpha_i = \frac{1}{1 + \|\mathbf{p}_i - \mathbf{p}\|^2 \|(\mathbf{p}_i - \mathbf{p}) \times \mathbf{n}\|^2}, \quad \alpha_i \in [0, 1]. \tag{6}$$

The former takes a maximum value for points in the vicinity of $\mathbf{p}$ while the latter is maximized at points near the projection axis defined by $\mathbf{p}$ and $\mathbf{n}$. When noise and outliers are close to $\mathbf{p}$, the experiments have led to the conclusion that Eq. (6) can obtain better projection than Eq. (5) for point-based modelling [2]. The main reason is that Eq. (6) considers not only the distance between $\mathbf{p}$ and $\mathbf{p}_i$, i.e. $\|\mathbf{p}_i - \mathbf{p}\|$, but the quantity $\|(\mathbf{p}_i - \mathbf{p}) \times \mathbf{n}\|$, measuring the distance between $\mathbf{p}_i$ and the projection axis, as well. However, in the case where noise and outliers are near both $\mathbf{p}$ and the projection axis, two weight functions both make the erroneous estimation.

Fig. 1(b) and (c) give the difference between Eqs. (5) and (6) for a simple 2D example. In the figures, we show the points with the large weights, which are used for the final computation of $\mathbf{p}^*$, using red. In this example, two outlier points are close to the test point $\mathbf{p}$ and also the projection axis (see the input point cloud in Fig. 1). Therefore, based on Eq. (5) or Eq. (6), the larger weights are erroneously assigned to the two outliers. Furthermore, for this case where the projection axis goes through the point cloud multiple times (see Fig. 1(c)), Eq. (6) will also take large weights for some points near the projection axis on the farther side, while the points should be regarded as outliers.

#### 1.2.2. *Forward vs. backward methods*

The DP algorithm is 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, the cardinality of $c_n$ is gradually reduced by removing some points from its point cloud using the weight function in Eq. (6). In this way, one is able to reduce processing time and also increase the accuracy of the projection procedure.

The reviewed DP algorithm [2] can be regarded as a *backward method*. The strategy of backward methods for projecting a point onto a noisy data first estimates the projection to the entire sample set and then tries to delete bad samples [4]. A backward method identifies the outliers with respect to the initial guess, thus points with large weight values computed using Eq. (5) or Eq. (6) will not be removed. The main disadvantage for the DP algorithm presented by Azariadis and Sapidis [1,2] is that outliers with large weight values can not be detected due to the limitation of the backward method. Fig. 1(b) and (c) also show the projection results of the DP algorithm using Eqs. (5) and (6), respectively. Note that the computed projection points $\mathbf{p}^*$ using two weight functions both are far from the given point cloud.

Our work is based on a powerful, relatively recent robust statistic technique called the *forward search paradigm* [4,14]. The basic idea in forward search is to start from a small set of robustly chosen samples of the data that excludes outliers; then to move forward through the data adding observations to the subset while monitoring certain statistical estimates [4]. In this paper, we use the *forward method* to guide the DP computation for outlier detection instead of the backward method. In Fig. 1(d), we show that our method can ignore both noise and outliers, and thus produces the expected result.

## 2. Robust directed projection

In this section, we present a simple, robust, and efficient algorithm: *robust directed projection* (RDP) to guide the DP computation. The proposed algorithm uses the forward search method [14] for outlier identification.

### 2.1. *Least median of squares*

The *least median of squares* (LMS) is a robust regression method that estimates the parameters of the model by minimizing the median of the absolute *residuals*. In other words, LMS replaces the sum of least squares by a median. LMS satisfies a 50% *breakdown* point [11], where a breakdown point might be loosely defined as the smallest percentage of outliers that can cause the estimator to take an arbitrarily large aberrant values [4,11]. The resulting estimator using LMS can resist the effect of nearly 50% of contamination in the input data, which is a larger breakdown point than least squares. In our work, we use LMS to replace the least squares used in the DP algorithm. Similar to Eq. (1), we also define the absolute residual as the distance between $\mathbf{p}_i$ and $\mathbf{p}^*$: for the $i$th point the residual $r_i = \|\mathbf{p}^* - \mathbf{p}_i\| = \|(\mathbf{p} + t\mathbf{n}) - \mathbf{p}_i\|$, where $\mathbf{p}^*$ is defined in Eq. (2). This definition of residuals is different from Ref. [4], in which the residuals are defined as the difference between two scale values, i.e. the measurement and estimation, by locally fitting a number of polynomials to points. We define the DP problem by searching for a best solution $t$ along the projection direction $\mathbf{n}$ that minimizes the median of the residuals:

$$\min_t \operatorname*{median}_i \|(\mathbf{p} + t\mathbf{n}) - \mathbf{p}_i\|, \quad t \in \mathbb{R}. \tag{7}$$

Rousseeuw [11] has also pointed out there always exists a solution for LMS.

In order to consider the effective of the distance between $\mathbf{p}$ and $\mathbf{p}_i$, as well as the distance between $\mathbf{p}_i$ and the projection axis defined by $\mathbf{p}$ and $\mathbf{n}$, we improve the optimization in Eq. (7) by also using the weight function in Eq. (6)

$$\min_t \operatorname*{median}_i \alpha_i \|(\mathbf{p} + t\mathbf{n}) - \mathbf{p}_i\|, \quad t \in \mathbb{R}. \tag{8}$$

Eq. (8) is essentially a weighted LMS optimization. Note that the $i$th point's absolute residual is redefined as the weighted distance between $\mathbf{p}_i$ and $\mathbf{p}^*$, i.e. $r_i = \alpha_i \|\mathbf{p}^* - \mathbf{p}_i\| = \alpha_i \|(\mathbf{p} + t\mathbf{n}) - \mathbf{p}_i\|$, for optimization in Eq. (8).

The above LMS optimization can be solved using the following *random sampling* algorithm, which is similar to

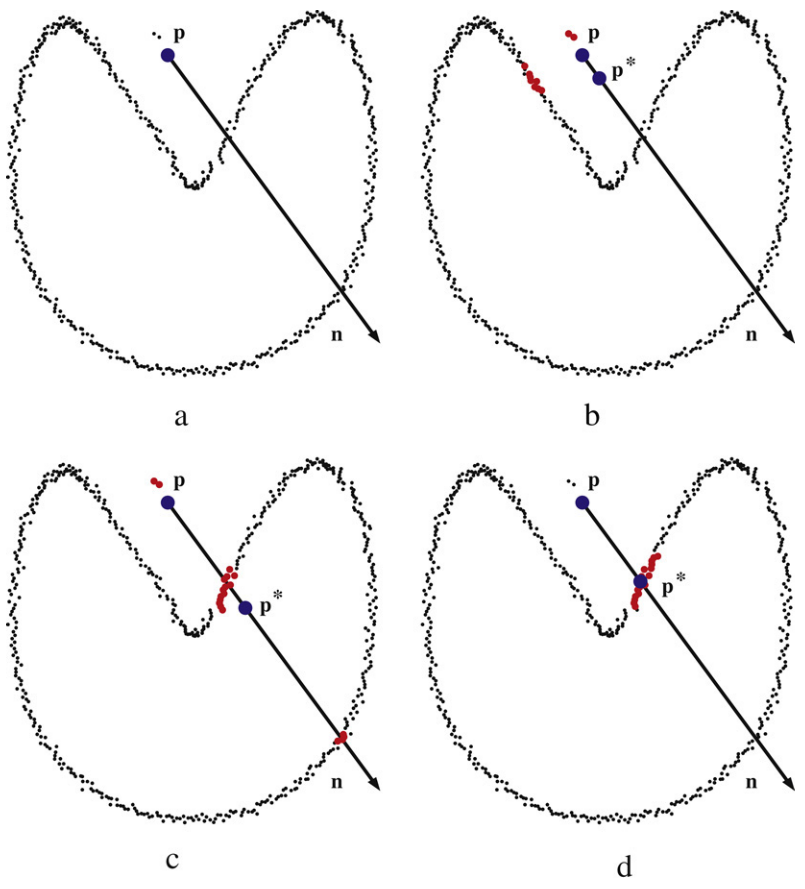


Fig. 1. Comparison between the DP algorithm and our algorithm. Here **p** is the test point to be projected (the blue point), **n** is the projection direction vector, $\mathbf{p}^*$ is the computed projection point (the other blue point), and the set of red points with large weights is the final working point cloud. In (a) we show the original point set with two outlier points (the top-left part of **p**). The results of the DP algorithm using Eqs. (5) and (6) are shown in (b) and (c), respectively. The DP algorithm can not detect the two outliers and it makes the erroneous projection points. In (d) our projection ignores the outliers and thus produces the expected result. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Fleishman et al.'s strategy [4]. First, $k$ points are selected at random, and the projection point is computed using the standard DP algorithm to the points. Next the median of the residuals of the remaining $N-k$ points is computed. The process is repeated $T$ times to generate $T$ candidate solutions. The solution with the minimal median is selected as the final solution $t$. A small value of $k$ does not use all of the available points to projection computation, while a larger value of $k$ requires more iterations. If $k$ is too large, the algorithm becomes sensitive to outliers [4].

## 2.2. *The forward search algorithm*

The forward search algorithm [14] is a robust method that avoids the need to fix $k$. Fleishman et al. [4] applied this technique to reconstruct surfaces from point clouds. Being different to the backward algorithms, which first fit a model to the entire data points and then delete bad samples, the forward algorithm first searches a small outlier-free subset and then iteratively refines the subset by adding one sample at a time. Fleishman et al. [4] show that a single outlier might cause a fitting to fail based on the backward algorithms, whereas the forward algorithm will give satisfactory results. First, the initial subset is computed for Eq. (8) using the LMS method with a small $k$ value, typically $k = p$ for a model with $p$ parameters ($p = 3$ in the 3D case).

During the forward search, a number of parameters can be monitored to detect the influential points. Typically, the forward search will add the good-samples first and only when these are exhausted, will outliers be added. Atkinson et al. [14] suggested several statistics, including the residual-plot, Cook's distance and others, were monitored. For their purposes, these are plotted on a graph and inspected visually. The maximal residual $r_t$ is monitored by Fleishman et al. [4], where $r_t$ is the threshold of maximal tolerated residual. They use a strategy from Ref. [13], in which the users interactively mark a small smooth region on the surface and then the system fits a polynomial to that region and measures the largest residual to set the value $r_t$. The above monitoring techniques are essential to determine the termination conditions for the forward search iteration. In our application, we monitor the maximal residual, and we will discuss it in the next section.

Using the forward search technique for solving Eq. (8), we present the main procedure of computing the projection point $\mathbf{p}^*$ of $\mathbf{p}$ onto a point cloud $\mathcal{C}_N$ as follows:

1. Choose a small outlier-free subset $Q$ using LMS.

2. The solution is computed using the original DP algorithm to the current subset $Q$.
3. The point with the lowest residual in the remaining points is added into $Q$.
4. Repeat steps 2 and 3 until the error is larger than a predefined threshold and identify the points in $\mathcal{C}_N - Q$ as outliers. Finally, recompute the projection position for the final $Q$ using the original DP algorithm.

## *2.3. Initial robust estimator*

In the first step of the forward search algorithm, the initial subset is computed using the LMS algorithm with a small $k$ value. In this procedure, the used random sampling algorithm requires a large iteration number $T$ to achieve a high probability of finding a good estimator. The LMS, as a statistical method, assumes that the samples (points) are independent. If $g$ is the probability of selecting a single good sample at random from the original point set $\mathcal{C}_N$, then the probability $P$ of successfully finding $k$ good samples after $T$ iterations can be computed by $P = 1 - (1 - g^k)^T$ [4]. Furthermore, for every iteration, LMS requires a sort of the residuals of the remaining $N - k$ points to find their median, so this will also require the expenditure of large amounts of time and space for sorting for a large $T$ if $N$ is very large.

In order to reduce processing time and also increase the accuracy of the projection procedure, we first filter the original point cloud $\mathcal{C}_N$ using the weight function in Eq. (6) and get a small working point cloud $c_n \subseteq \mathcal{C}_N$. This is particularly useful for large data sets with tens or hundreds of thousands of points, improving considerably the efficiency of the algorithm. We have tried two ways that reduce $\mathcal{C}_N$ for obtaining a working point cloud $c_n$. One is Azariadis and Sapidis's strategy [2] by setting a constant $K$ as follows. All weights $\{\alpha_i\}$ of points in $\mathcal{C}_N$ are first computed and another local variable named $\alpha_{\text{limit}}$ is defined

$$\alpha_{\text{limit}} = \begin{cases} \alpha_{\text{mean}} + \dfrac{\alpha_{\max} - \alpha_{\text{mean}}}{10 - K}, & K < 9 \\ \alpha_{\text{mean}} + \dfrac{\alpha_{\max} - \alpha_{\text{mean}}}{2}, & \text{otherwise} \end{cases} \tag{9}$$

where the real parameters $\alpha_{\max}$ and $\alpha_{\text{mean}}$ correspond to the maximum and mean value of the computed weights $\{\alpha_i\}$. Eq. (9) is defined in such a way that cloud points with a small influence (i.e. small $\alpha_i < \alpha_{\text{limit}}$) are discarded in the given $K$. In our application, we found that it is not easy to control $K$ for point clouds with different sizes. In our implementation, we sort the weights $\{\alpha_i\}$ in a decreasing order and then choose the $n$th weight as $\alpha_{\text{limit}}$. In this way, $c_n$ may be set to a fixed size, such as $n = 300$ for large point clouds or $n = N/100$.

## *2.4. The implementation for the RDP algorithm*

The outline of an algorithm for projecting an arbitrary point onto a cloud of points along a projection vector, called **RobustDP**, is given in Algorithm 1. The algorithm takes as input a point cloud $\mathcal{C}_N$, a test point $\mathbf{p}$ and the projection vector $\mathbf{n}$, and computes the projection $\mathbf{p}^*$ of $\mathbf{p}$ onto $\mathcal{C}_N$ along $\mathbf{n}$. This is achieved through the forward search with the aid of a local variable $Q$ which is a subset of $\mathcal{C}_N$. The procedure starts from a small set $Q$ of robustly chosen samples of the data that excludes outliers using the LMS algorithm. During the projection calculation, the cardinality of $Q$ is gradually increased by adding a point with the lowest residual from its cloud points at each iteration.

According to Algorithm 1, a small working point cloud $c_n$ is first obtained from the input $\mathcal{C}_N$ using **FilterPointCloud**, as shown in Algorithm 3. In the **FilterPointCloud** procedure, the significance of each point in $\mathcal{C}_N$ is determined by the corresponding weight, calculated using Eq. (6); the $n$ points with the largest weights are collected as $c_n$. In this way, one is able to reduce processing time and also increase the accuracy of the projection procedure.

Then an initial subset $Q$ is computed by passing $c_n$ into the **LMS** algorithm (see Algorithm 2), and in the meantime the projection $\mathbf{p}^*$ computed using the DP algorithm for the output $Q$ is also returned. According to Algorithm 2, a loop of $T$ times begins. At each time, $k$ points are selected at random and stored in a temporary set $c_{\text{temp}}$. Then the projection point $\mathbf{p}^*$ is computed using the standard DP algorithm to $c_{\text{temp}}$. Next the median $r_{\text{half}}$ of the residuals of the remaining $n - k$ points, stored in $c_{\text{remain}}$ ($c_{\text{remain}} \Leftarrow c_n - Q$), is computed. The process is repeated $T$ times to generate $T$ candidate solutions. $Q$ and $\mathbf{p}^*$ with the minimal median $r_{\text{half}}$ are put back into Algorithm 1, i.e. **RobustDP**.

In Algorithm 1, the forward search is implemented using an iteration procedure. At each iteration, the point $\tilde{\mathbf{p}}$ with the lowest residual in the remaining points in $c_{\text{remain}}$ ($c_{\text{remain}} \Leftarrow c_n - Q$) is added into $Q$. If the Euclidean distance between the current projection estimation $\mathbf{p}^*$ and $\mathbf{p}$ is larger than a threshold $r_{\text{t}}$, the procedure is terminated. In the opposite case, $\mathbf{p}$ is moved to the current $\mathbf{p}^*$ ($\mathbf{p} \Leftarrow \mathbf{p}^*$) and a new iteration commences with a redefined $Q$. The maximal iteration number MAX_ITERS is usually a predefined integer to ensure that the number of points in $Q$ is more than 50% in $c_n$ (generally $[0.5n] \leq$ MAX_ITERS $\leq n$).

Fig. 2 shows an illustration of the RDP procedure.

*Note* 1. In Algorithm 1, $\mathbf{p}^*$ is computed using the DP algorithm for the current updated $\mathbf{p}$, so the weights of points in $c_{\text{temp}}$ need also be recomputed at each iteration.

*Note* 2. In Algorithm 1, $r_{\text{t}}$ is the threshold of maximal tolerated difference between the current projection estimation $\mathbf{p}^*$ and the updated point $\mathbf{p}$. In our experiments, we have found that $r_{\text{t}} = 0.01\varepsilon$ gives good results, where $\varepsilon$ is the distance between the input $\mathbf{p}$ and $\mathbf{p}^*$ computed using the initial subset with $k$ points, i.e. the output of Algorithm 2. A small value of $r_{\text{t}}$ does not use all of the available samples to compute the projection point, while a larger value for $r_{\text{t}}$ requires more iterations and the algorithm becomes sensitive to outliers. If $r_{\text{t}}$ is too large, so that the final $Q$ is equal to $c_n$, i.e. no outlier is detected, the projection using the proposed RDP algorithm is similar to one using the DP algorithm. In some sense, the DP algorithm is only one special case of our algorithm.

*Note* 3. The computation costs of the RDP algorithm mainly depend on two factors. One factor is the initial subset selection

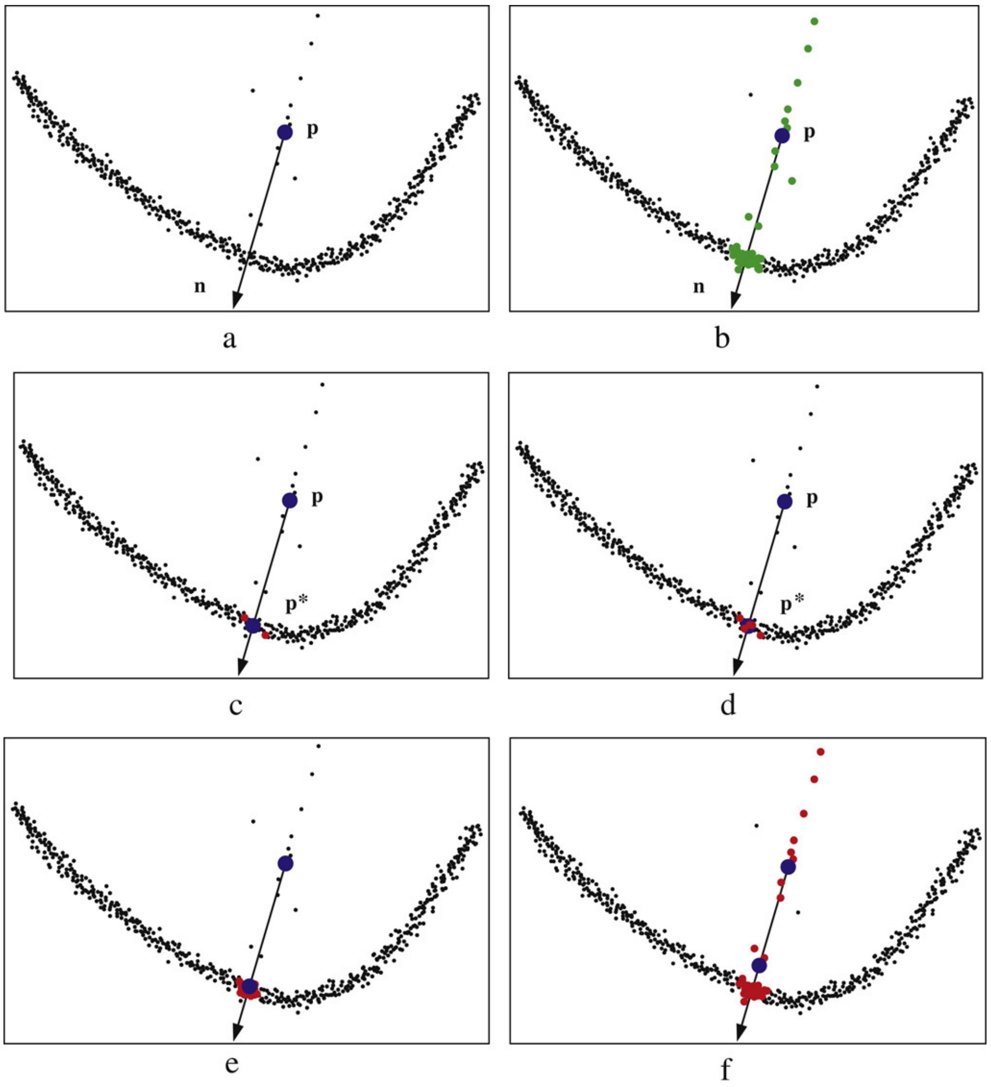


Fig. 2. The illustration of the RDP procedure. The input point cloud, a test point **p** (blue), and the projection direction **n** are shown in (a). First, a small working point set $c_n$ (green points in (b)) is extracted from the input point cloud using the weight function of Eq. (6). Then, an initial subset (red points in (c)) is selected by passing $c_n$ using the LMS algorithm, and the initial projection point $\mathbf{p}^*$ (another blue point in (c)) is computed using the DP algorithm. Next, we iteratively add points with the smallest residual and recompute the projection point $\mathbf{p}^*$ (blue) to the updated subset (red points in (d)). The final projection point calculated using the forward search is shown in (e). The remaining points are regarded as outliers to the DP procedure, and these are not used for computation of the final projection point. In contrast, the result using the original DP algorithm is given in (f), where some outliers affect computation of the projection point. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

using the LMS algorithm. The process is repeated $T$ times, and the DP problem is solved for the current $k$ points selected randomly at each time. Since we limit the LMS process to a small working point cloud $c_n$ by filtering the original point cloud $C_N$, a small $T$ is sufficient for our implementation. In Algorithm 2, we typically choose $T = 1000$ that can obtain both small errors and little computation time. The other factor is that the forward search algorithm described in Algorithm 1 adds a single sample at each iteration and solves a DP problem at each iteration. In the tests that are presented below, we allow the adding of multiple points at each iteration as long as their residuals are within the allowed tolerance and the maximal number of points that we add is not more than 40% of the size of the working point cloud $c_n$. In this way, we can approximate the projection point through the forward search after a few iterations.

An in-depth experimental evaluation of the accuracy and robustness of the proposed RDP method is given in the following section.

## 3. Experimental results

We have applied the proposed **RobustDP** algorithm to some point clouds, which are sampled from two B-spline surfaces (see Figs. 3 and 4). The algorithms described above are 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).

The experiments focus on the efficiency of our algorithm for point clouds, with outliers, by comparing the true results of projecting some test points onto one surface and the approximate results of projecting the same test points onto point

**Algorithm 1** : **RobustDP**($\mathcal{C}_N$, **p**, **n**, $\mathbf{p}^*$, $Q$)

```
Input:
  $\mathcal{C}_N \in \mathbb{R}^{3\times N}$: the given point cloud with $N$ points
  $\mathbf{p}$: the test point
  $\mathbf{n}$: the projection direction vector
Output:
  $\mathbf{p}^*$: the projection point of $\mathbf{p}$
  $Q \subseteq \mathcal{C}_N$: the subset of points for computing $\mathbf{p}^*$
Local variables:
  $k$: the number of random samples
  $r_t$: the maximal tolerated residual
  $I$: the current iteration
  $c_n \subseteq \mathcal{C}_N$: the working point cloud with $n$ points
  $c_{temp} \subseteq c_n$: a point set used for temporary storge
  $c_{remain} \subseteq c_n$: the set of the remaining points, i.e. $c_n \backslash Q$
  MAX_ITERS: the maximal number of iterations
begin
 1: FilterPointCloud($\mathcal{C}_N$, $\mathbf{p}$, $\mathbf{n}$, $c_n$);
 2: $Q \Leftarrow \emptyset$;
 3: LMS($c_n$, $\mathbf{p}$, $\mathbf{n}$, $k$, $Q$, $\mathbf{p}^*$);
 4: $I \Leftarrow 0$;
 5: $c_{temp} \Leftarrow Q$;
 6: while ($I++ <$ MAX_ITERS) do
 7:    $c_{remain} \Leftarrow c_n - Q$;
 8:    Compute the residuals to $\mathbf{p}^*$ for all points in $c_{remain}$;
 9:    Get the point $\tilde{\mathbf{p}}$ with the lowest residual from $c_{remain}$;
10:    $c_{temp} \Leftarrow c_{temp} + \tilde{\mathbf{p}}$;
11:    Compute $\mathbf{p}^*$ using $c_{temp}$ based on the DP algorithm
       through Eqs. (2), (3) and (6);
12:    if ($I > 1$ and $\|\mathbf{p} - \mathbf{p}^*\| > r_t$) then
13:       return
14:    end if
15:    $\mathbf{p} \Leftarrow \mathbf{p}^*$;
16:    $Q \Leftarrow c_{temp}$;
17: end while
end
```

**Algorithm 2** : **LMS**($c_n$, **p**, **n**, $k$, $Q$, $\mathbf{p}^*$)

```
Input:
  $c_n \in \mathbb{R}^{3\times n}$: the working point cloud with $n$ points
  $\mathbf{p}$: the test point
  $\mathbf{n}$: the projection direction vector
  $k$: the number of random samples
Output:
  $Q \subseteq c_n$: the initial subset
  $\mathbf{p}^*$: the projection point of $\mathbf{p}$
Local variables:
  $T$: the number of iterations
  $c_{temp} \subseteq c_n$: the subset selected randomly
  $c_{remain} \subseteq c_n$: the set of the remaining points, i.e. $c_n \backslash c_{temp}$
  $\mathbf{r} \in \mathbb{R}^{n-k}$: the vector of residuals
  $r_{half}$: the median of residuals
  $r_{min}$: the minimal residual
begin
 1: $r_{min} \Leftarrow \infty$;
 2: for ($j = 0; j < T; j++$) do
 3:    Select randomly a subset $c_{temp}$ with $k$ points;
 4:    Compute $\mathbf{p}^*$ using $c_{temp}$ based on the DP algorithm
       through Eqs. (2), (3) and (6);
 5:    $c_{remain} \Leftarrow c_n - c_{temp}$;
 6:    /* Compute $\mathbf{r}$ as the residuals of points in $c_{remain}$ */
 7:    for ($i = 0; i < |c_{remain}|; i++$) do
 8:       $\mathbf{p}_i \Leftarrow c_{remain}(i)$;
 9:       $\mathbf{r}(i) \Leftarrow \|\mathbf{p}_i - \mathbf{p}^*\|$;
10:    end for
11:    Compute the median $r_{half}$ by sorting $\mathbf{r}$;
12:    if ($r_{half} < r_{min}$) then
13:       $r_{min} \Leftarrow r_{half}$;
14:       $Q \Leftarrow c_{temp}$;
15:    end if
16: end for
end
```

clouds sampled from the surface. Furthermore, in all results shown in this section paper, we use $T = 1000$ iterations in Algorithm 2 for obtaining both small errors and short computation time.

### 3.1. Example 1

Fig. 3(a) shows an example of projecting a set of 20 points, sampled from a line segment, onto a B-spline surface. First we use a modified Newton–Raphson method [7] to pre-compute the closest point for orthogonal projection onto the B-spline surface. Suppose $\mathbf{p}_i$ ($i = 1, \ldots, l$) are the test points sampled from the line segment, where $l = 20$ is the number of test points sampled. We obtain 20 *true* projection points $\mathbf{p}_i^S$ by projecting $\mathbf{p}_i$ onto the given B-spline surface using the modified Newton–Raphson method. For each $\mathbf{p}_i$, the vector

$$\mathbf{n}_i = (\mathbf{p}_i^S - \mathbf{p}_i)/\|\mathbf{p}_i^S - \mathbf{p}_i\|, \quad i = 1, \ldots, l \tag{10}$$

are assumed as the input projection direction vector for testing the **RobustDP** algorithm.

For this experiment, an initial point cloud with 60,000 points is generated by sampling from the B-spline surface in Fig. 4 (a). Then, a series of noisy point clouds are produced by randomly adding Gaussian noise to each point cloud along the positive normal directions with increasing variances $\rho$, where $\rho$ is the point cloud's "thickness factor" [2] with respect to the surface bounding box. For all results in the example, we have used five scales $\rho \in \{\gamma, 5\gamma, 10\gamma, 15\gamma, 20\gamma\}$, where $\gamma$ is 1.0% of the length of the diagonal of the bounding box of the model. In addition, the working point cloud $c_n$ is chosen as 300 points for Algorithm 3 for this example.

Table 1 lists the accuracy and execution time for the point clouds with outliers, where "Relative error" is defined by the projection error relative to the corresponding $\mathbf{p}_i^S$. All data are the average of projecting equally 20 spaced points onto the given B-spline surface and the corresponding $\mathcal{C}_N$. The accuracy, i.e. relative error, is the average error, which is computed by

$$\frac{\sum_{i=1}^{l} \frac{\|\mathbf{p}_i^* - \mathbf{p}_i^S\|}{\|\mathbf{p}_i - \mathbf{p}_i^S\|}}{l}. \tag{11}$$

Table 1
Results of point projection onto noisy point clouds sampled from the B-spline surface in Fig. 3 with increasing density

| $\rho$ | Time (s) | Relative error RDP | Relative error DP1[a] | Relative error DP2[b] |
|---|---|---|---|---|
| $1\gamma$ | 0.175753 | 0.00704593 | 0.00711199 | 0.0151443 |
| $5\gamma$ | 0.176754 | 0.00711019 | 0.0888141 | 0.0127748 |
| $10\gamma$ | 0.173750 | 0.00754417 | 0.367748 | 0.0131383 |
| $15\gamma$ | 0.178757 | 0.00831023 | 0.661613 | 0.0143648 |
| $20\gamma$ | 0.174251 | 0.00903053 | 0.671681 | 0.0145268 |

[a] DP1 is the DP algorithm using Eq. (5).
[b] DP2 is the DP algorithm using Eq. (6).

```
Algorithm 3 : FilterPointCloud($\mathcal{C}_N$, p, n, $c_n$)
Input:
  $\mathcal{C}_N \in \mathbb{R}^{3\times N}$: the given point cloud with $N$ points
  p: the test point
  n: the projection direction vector
Output:
  $c_n \in \mathbb{R}^{3\times n}$: the working point cloud with $n$ points
Local variables:
  $\mathbf{a} \in \mathbb{R}^n$: the weights vector
  $\alpha_{\text{limit}} \in \mathbb{R}$: the $n$th weight of a sorted in a decreasing order
begin
  1: $c_n \Leftarrow \emptyset$;
  2: Compute weights a through Eq. (6);
  3: Sort weights a in a decreasing order;
  4: $\alpha_{\text{limit}} \Leftarrow \mathbf{a}(n)$;
  5: for $(i = 0; i < N; i++)$ do
  6:     if $(\alpha_i > \alpha_{\text{limit}})$ then
  7:         $c_n \Leftarrow c_n + \mathcal{C}_N(i)$;
  8:     end if
  9: end for
end
```

The experimental data shows that the new method has good approximation.

### 3.2. Example 2

Fig. 4(a) shows an example of projecting a set of 20 points $\mathbf{p}_i$ $(i = 1, \ldots, l)$, sampled from a NURBS curve, onto another B-spline surface. First we also use the modified Newton–Raphson method to pre-compute the closest point for orthogonal projection onto the B-spline surface, and obtain 20 *true* projection points $\mathbf{p}_i^S$ by projecting $\mathbf{p}_i$ onto the given B-spline surface. Next we use Eq. (10) for computing the input projection direction vector for each $\mathbf{p}_i$ to test the **RobustDP** algorithm. For this experiment, an initial point cloud with 60,000 points is generated by sampling from the B-spline surface in Fig. 4. Then, a series of noisy point clouds are produced by adding Gaussian noise to each point cloud along the positive normal directions with increasing variances $\rho$ with respect to the surface bounding box. The accuracy and execution time for different point clouds are given in Table 2. The experimental data shows that the new method has good approximation.

### 3.3. Example 3

Fig. 5(a) gives an example of projecting a set of 20 points $\mathbf{p}_i$, sampled from a NURBS curve, onto a point cloud of the mans head model. First we use the method proposed in [3] to pre-compute the closest point for orthogonal projection onto the model, and obtain 20 *true* projection points $\mathbf{p}_i^S$. Next we use Eq. (10) for computing the input projection direction vector for each $\mathbf{p}_i$ for testing the **RobustDP** algorithm. For this experiment, the model has added Gaussian noise along the normals with $\rho = 20\gamma$. Note that the results in Fig. 5(b) and (c) using the DP algorithm make erroneous projection points, whereas our projection ignores outliers and thus produces the expected result.

## 4. Discussion

In this section, we will discuss the differences with Fleishman et al.'s works, applications, and limitations on the RDP algorithm.

### 4.1. Orthogonal projection vs. directed projection

We first focus on the differences with Fleishman et al.'s works. The problem of point projection onto point clouds is classified as two types: *orthogonal projection* and *directed projection*. The standard moving least squares (MLS) algorithm [4,8,9] deals with the former, whereas our algorithm, presented in this work, focuses on the latter. The goal of orthogonal projection is to find the closest point (footpoint) on a point cloud surface, where there is no constraint on the projection direction that is determined by the input test point and the final projection point computed. One main application of MLS is to smooth a point cloud by orthogonally projecting each point of the point cloud onto the fitted surfaces [4,8,9], which will move noisy points onto the virtual surfaces and provide a clean point cloud surface for point-based modelling. In contrast, the goal of the directed projection or DP problem discussed in our paper is to find the projection point of a test point along a constraint projection direction. The DP problem is actually equivalent to the intersection problem of a ray and a point cloud surface, where the ray is determined by the test point and the given projection direction. The projection point of the DP algorithm approximates the first intersection point between the ray and the point cloud surface. Two constraint conditions in the DP optimization are that the computed projection point is on the ray

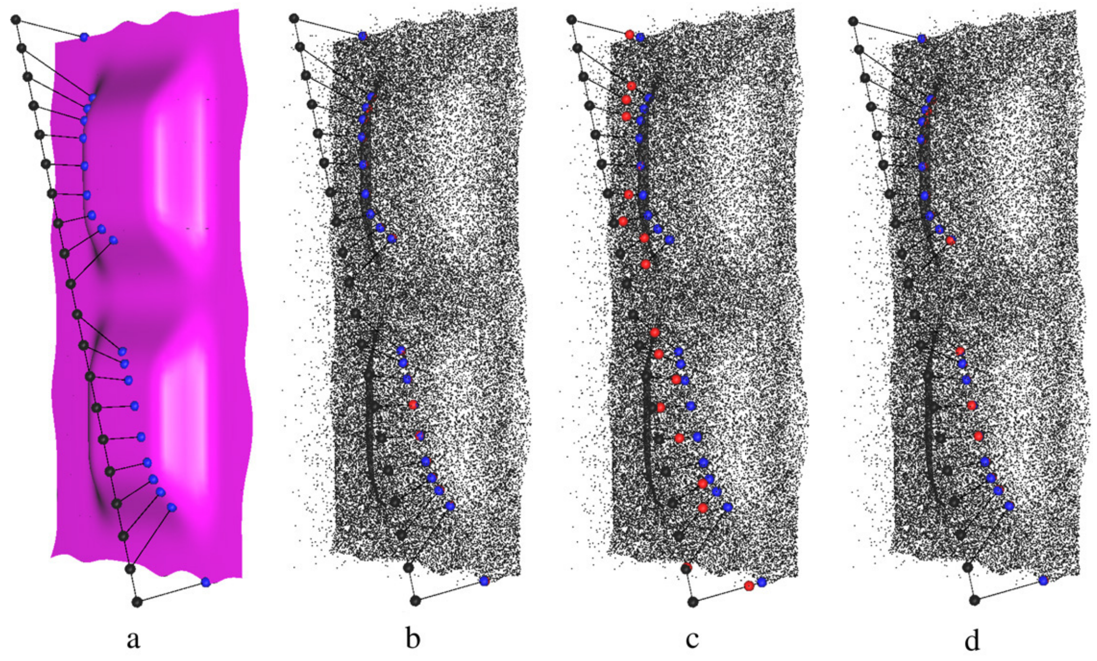


Fig. 3. Comparing the results of projecting points onto a B-spline surface and onto noisy point clouds consisting of 60,000 points sampled from the surface. The projection points are blue and red, respectively. Here the test points (the black points) are sampled from a line segment. (a) The pre-computed closest point for orthogonal projection onto the B-spline. (b) The result of our algorithm. (c) The result of the DP algorithm using Eq. (5). (d) The result of the DP algorithm using Eq. (6). Note that the DP algorithm using Eq. (5) can not detect the outliers and makes erroneous projection points, and the DP algorithm using Eq. (6) can get a good projection result for most test point due to the outliers being far from the direction vector. In contrast, our projection ignores outliers and thus produces the expected result. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Results of point projection onto noisy point clouds sampled from the B-spline surface in Fig. 4 with increasing noise

| $\rho$ | Time (s) | Relative error RDP | Relative error DP1[a] | Relative error DP2[b] |
|---|---|---|---|---|
| $1\gamma$ | 0.175252 | 0.000912825 | 0.00265534 | 3.45114 |
| $5\gamma$ | 0.178256 | 0.00127872 | 0.0065534 | 3.46366 |
| $10\gamma$ | 0.177255 | 0.00216594 | 0.0862853 | 3.47652 |
| $15\gamma$ | 0.177255 | 0.00344006 | 0.0468185 | 3.48209 |
| $20\gamma$ | 0.177255 | 0.0035093 | 0.442069 | 3.48902 |

[a] DP1 is the DP algorithm using Eq. (5).
[b] DP2 is the DP algorithm using Eq. (6).

(see Eq. (1)) and should be close to the point cloud surface (see Eq. (2)). Some applications of the DP problem will be discussed in Section 4.3.

In spite of of the fact that the mathematical tools used in our RDP algorithm and robust moving least squares (RMLS) [4] are same, i.e. replacing “least squares” by “least median of squares” and approximating the LMS optimization with the forward search, the defined optimization equations are different between RDP and RMLS as follows. The original DP algorithm needs to solve a minimal error function (i.e. Eq. (1)) with a projection direction constraint (i.e. Eq. (2)). In contrast, the standard MLS algorithm only solves the minimal error function similar to Eq. (1) without any extra constraint. In some sense, the DP optimization contains one more constraint than MLS.

The common motivation between our RDP algorithm and Fleishman et al.’s RMLS algorithm [4] is to replace “least squares” by “least median of squares”, such that the improved algorithms are robust. At each iteration of the forward search used for approximating the LMS optimization, our algorithm can obtain a faster convergence than RMLS. This improves on the original DP algorithm which gives a closed form solution Eq. (3) in a linear optimization. In contrast, RMLS is based on the standard MLS algorithm whose solution is a non-linear optimization problem [8]. In our algorithm, we assume that the weights are pre-computed at each iteration such that the optimization function is simple. In the future we plan to optimize the projection computation together with the weight function. This is an area for the future (and challenging) research for the DP problem.

## 4.2. *Disturbing points*

The DP problem has three input conditions: the test point, the projection direction, and point clouds. Actually, an further post-processing based on smoothing or de-noising could clear up the point cloud [15], but it can not ensure that the projection

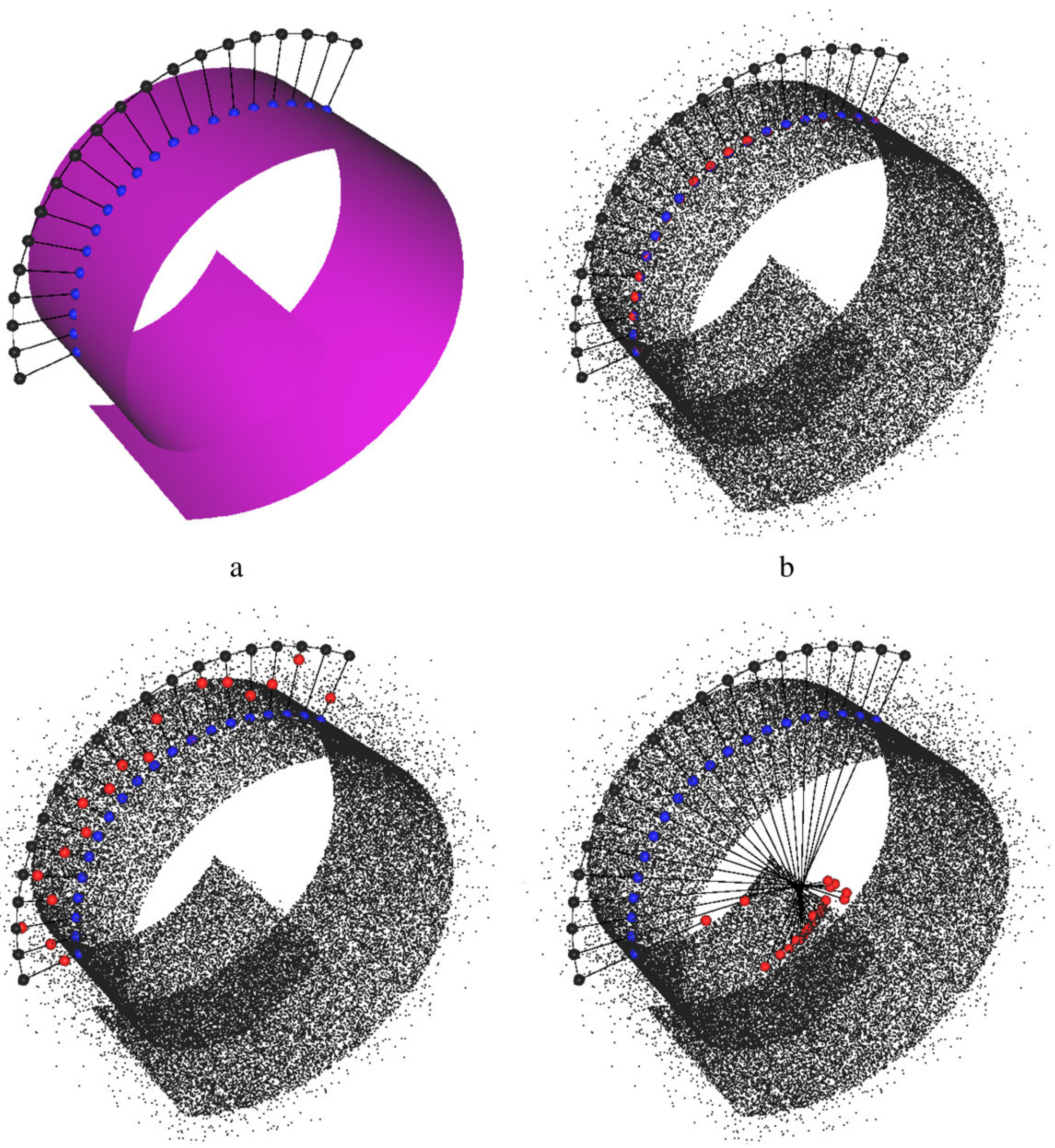


Fig. 4. Comparing the results of projecting points onto a B-spline surface, and onto noisy point clouds consisting of 60,000 points sampled from the corresponding surface. The projection points are blue and red, respectively. Here the test points (the black points) are sampled from a NURBS curve. (a) The pre-computed closest point for orthogonal projection onto the the B-spline. (b) The result of our algorithm. (c) The result of the DP algorithm using Eq. (5). (d) The result of the DP algorithm using Eq. (6). Note that the DP algorithm using Eq. (6) can not detect the outliers due to the folded surface and makes erroneous projection points, whereas our projection ignores outliers and thus produces the expected result. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

works well for the DP problem. The main reason is that both the test point position and the projection direction also affect the final projection computation. In fact, not only noisy points of point cloud but also some other points can result in projection error. In Fig. 6, a simple example shows the weakness of the DP algorithm on a noiseless point cloud. In Fig. 6(b), some red non-noisy points, which are far from the input test point but close to the projection axis, disturb the final projection computation. We call these points of a point cloud, which are not noisy points of point cloud itself but disturb the position of the final projection point, *disturbing points* of the point cloud surface relative to the given test point and the projection direction. In this paper, we classify the outliers defined in this paper as two types: noisy points and *disturbing points* of the point cloud surface. Noisy points could be removed by some existing smoothing or denoising algorithms. In contrast, the disturbing points can not be detected by smoothing techniques, since they are correct points relative to the given point cloud itself but they are, however, wrong points relative to the test point and projection axis in the DP problem. Disturbing points are usually non-noisy points close to the test point or the projection axis.

Now we analyze the reason for the failure of the original DP algorithm in the example in Fig. 6. In Fig. 6(a), a point cloud is randomly sampled from a curve, where the ray determined by

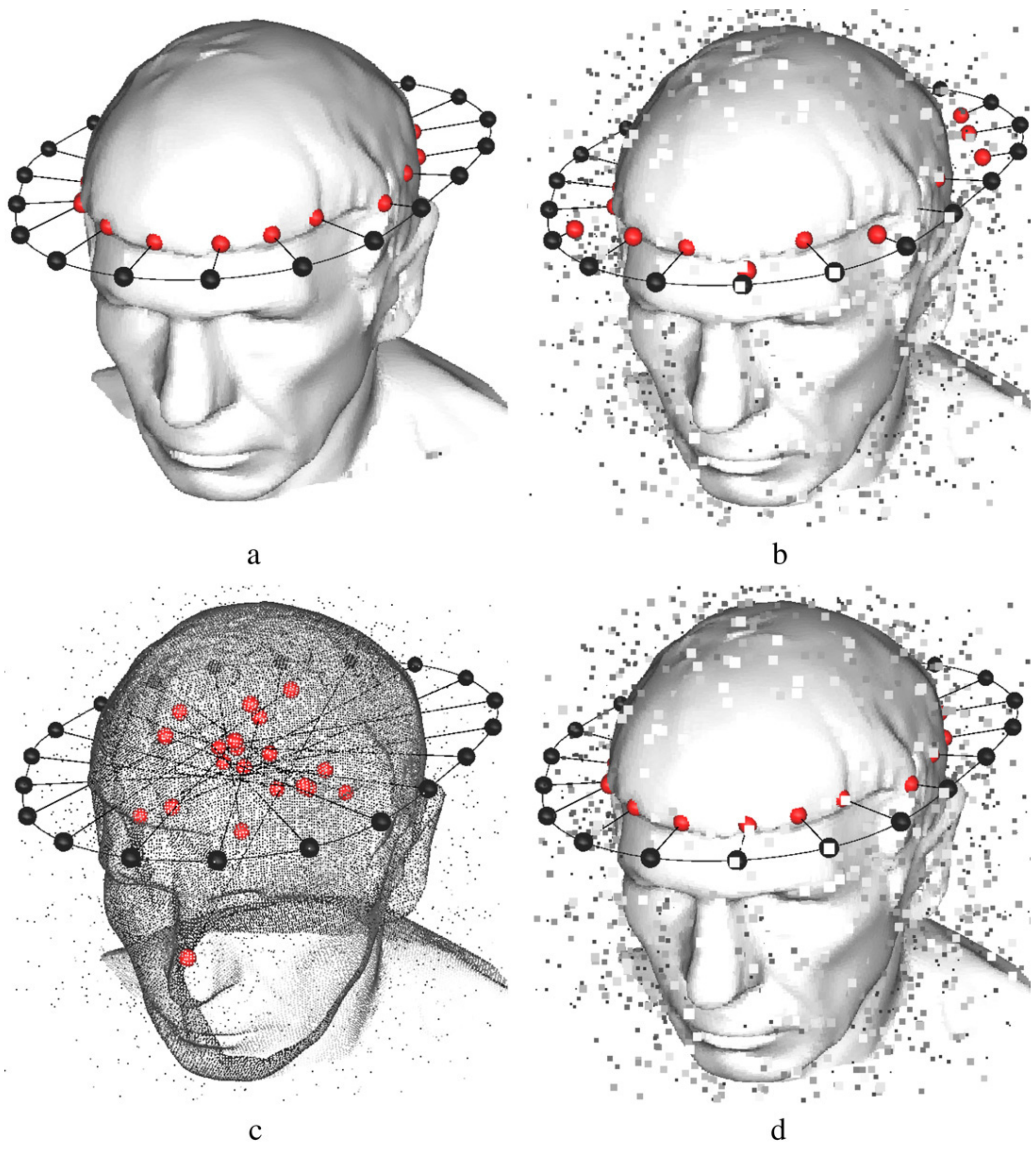


Fig. 5. Comparing the results of projecting points onto a mans head model consisting of 62,202 points. The projection points are red. Here the test points (the black points) are sampled from a NURBS curve. (a) The pre-computed closest point for orthogonal projection onto the point cloud and the projection direction vectors. (b) The result of the DP algorithm using Eq. (5). (c) The result of the DP algorithm using Eq. (6). (d) The result of our algorithm. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the test point and the projection direction goes through the point cloud four times. In Fig. 6(b), Eq. (6) also takes large weights for points near the projection axis on the farther side, while the points on the farther side are disturbing points and should be ignored for the final projection computation. The disturbing points in Fig. 6(b) result in computed projection points that are far from the given point cloud. Fig. 6(c) demonstrates the result using the RDP algorithm, which produces a reasonable approximation.

Another example is shown in Fig. 7 to illustrate the effect of the disturbing points. Eq. (5) tends to find the average position of the nearest points of a test point as its projection point. In this figure, the left hand of dinosaur is close to several test points, which leads to a large projection error using Eq. (5) (see Fig. 7(e)). In addition, Eq. (6) is usually invalid for the closed model. When there are two or more intersection points between the point-based model and the ray determined by a test point and the projection direction, Eq. (6) actually tends to find the average position between all intersection points. Fig. 7(f) shows that most of the projection points computed using Eq. (6) are inside this model. Fig. 7(d) shows the result of our algorithm. Although our method is based on Eq. (6), our method can ignore the effect of disturbing points and approximate the first intersection point as the final projection point by combining LMS and the forward search.

Although the currently existing algorithms for the DP problem can deal with noise by combining some smoothing techniques, they can not sufficiently solve the case of disturbing points. In contrast, we treat the noise and disturbing points together as outliers, and detect them by using the LMS optimization and forward search. Our method does not depend on the smoothing and denoising procedure.

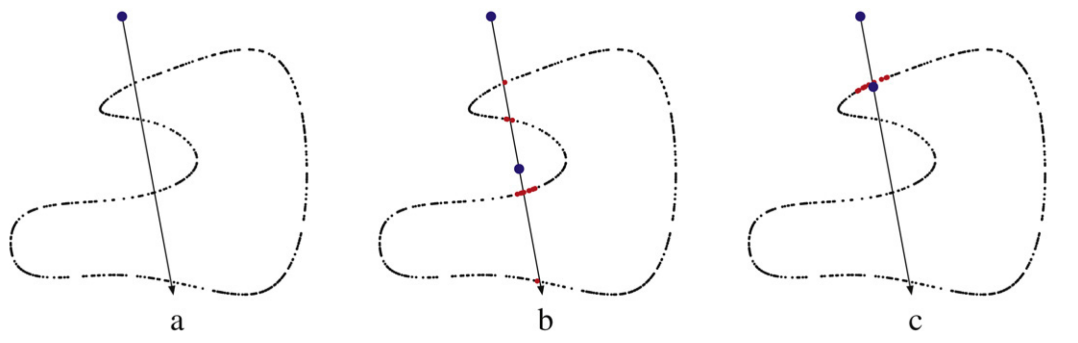


Fig. 6. Comparing the results of directed projection of points onto a noiseless point cloud. (a) An input point cloud, a test point (blue) and the projection direction, where points of the point cloud are randomly sampled from a B-spline curve without additional noise added. (b) The projection result of the DP algorithm using Eq. (6), where the projection point is another blue point. (d) The result of our algorithm. Here the red points make up of the final working point set that is used for computing the projection point. Note that the original point cloud has no noise, but the projection result computed using the original DP algorithm is still affected by the disturbing points of point cloud (some red points far from the test point in (b)), whereas our algorithm can give a reasonable approximation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

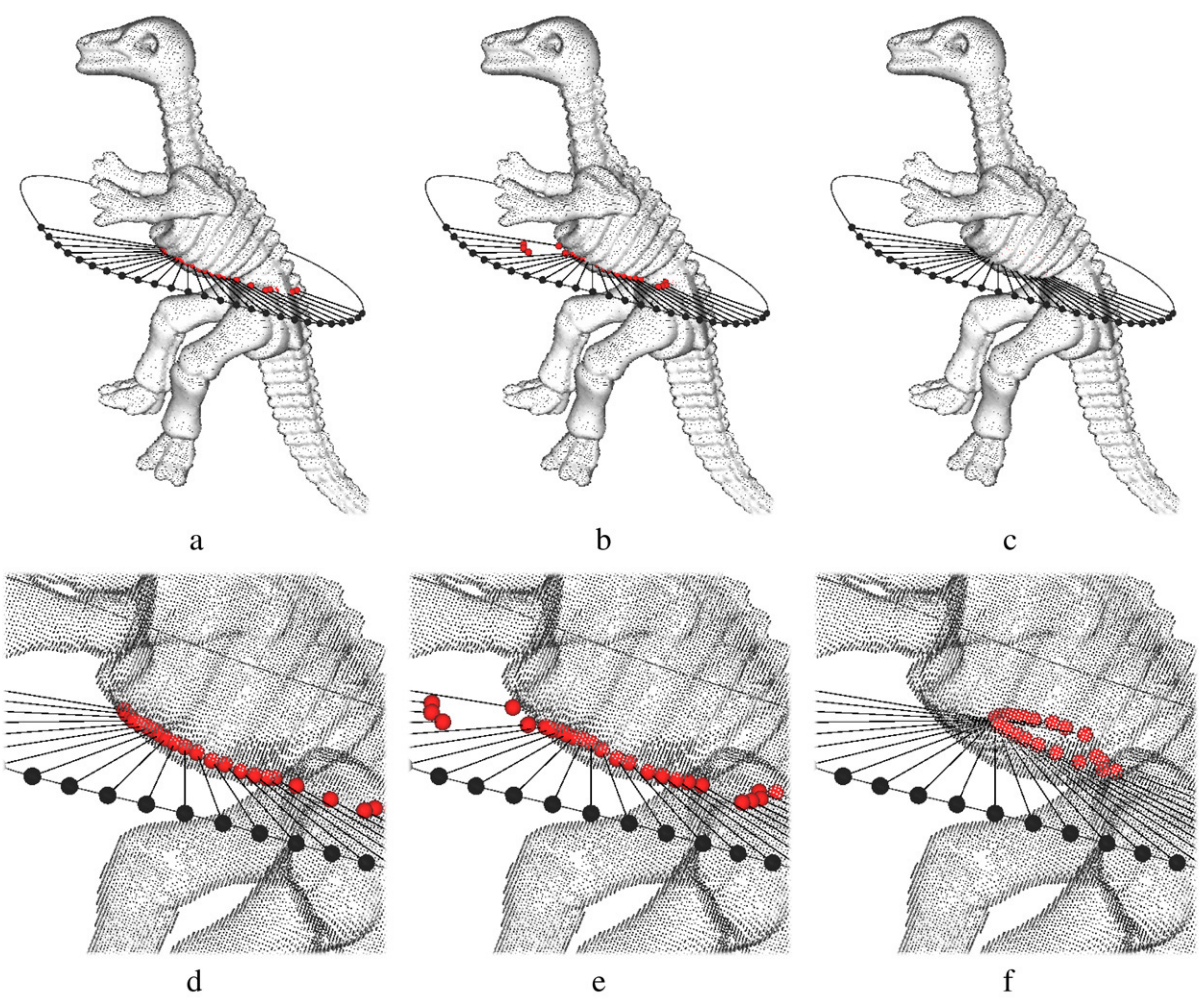


Fig. 7. Comparing the results of projecting points onto a noiseless point cloud representing the dinosaur model with 70,526 points. Here the test points (black points) are sampled from a B-spline curve around this model, and the projection direction of each test point is chosen by the vector from the center of the B-spline to the test point itself. The computed projection points are red. (a) The result of our algorithm. (b) The result of the DP algorithm using Eq. (5). (c) The result of the DP algorithm using Eq. (6). The magnified views of (a), (b) and (c) are shown in (d), (e) and (f), respectively. Although the input point cloud does not contain additional noise, the projection results computed using the DP algorithm can not give satisfactory results for part of test points. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

### 4.3. Application on computing intersections of ray/point cloud surface

As stated in Section 4.1, the DP problem is equivalent to the problem of computing the first intersection point between a ray and a point cloud surface. There are some applications that can benefit from the RDP algorithm. Designing curves onto a point cloud is an important problem for many applications in reverse engineering (RE), such as segmentation, parameterization [1], reconstruction, and point-based modelling [2]. A general 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 [1]. In some RE software, users usually finish the segmentation through interactively designing curves onto a point cloud, but this step is not easy. Azariadis et al. [2] developed a new technique, called drawing curves onto a cloud of points, for point-based modelling. Their technique is based on the DP method of projecting points onto a point cloud, where the projection vectors are usually specified through a graphics interface tool. Although the work in [1,2] can also handle noisy points in point clouds in some sense, their work can not sufficiently solve the case of disturbing points. For example, when the ray determined by the test point and the given projection direction goes through the point cloud surface many times, the original DP algorithm mistakenly computes the average position between all intersection points as the projection point.

Apart from some applications [1,2] in a point-based CAD/CAM system, some papers have referred to some additional applications for this intersection problem. For instance, some point-based rendering techniques using ray tracing algorithms need to compute the intersection points between the ray from a viewpoint and point clouds [16–18]. In addition, Liu et al. [10] presented to use Cauchy–Crofton formula for computing areas of point-sampled surfaces, without any surface reconstruction procedure, where a key technique is to count the number of intersection points between the lines created randomly and the point-sampled surfaces. Another application might be interactive selection for a point cloud surface, which usually needs to compute the intersection.

Traditionally, there are two approaches to compute the intersection of a ray and the point cloud surface. One indirect approach is based on surface reconstruction techniques, and then uses general ray/surface intersection methods. To speed up the intersection computation, Adamson et al. [16] define a point set surface approximated locally by polynomial, and then compute the intersection of a ray and the point set surface. The other approach described by Schaufler et al. [17] computes directly the intersection between a ray and the point set by placing a disk at each point of the point set without reconstructing any surface. Their method first intersects a cylinder around the ray with those disks. Then, the intersection is computed as a weighted average of disks whose centers are inside the cylinder. However their method can only find the average position of all intersection points for this case with multiple intersection points. Liu et al. [10] extended Schaufler et al.'s approach for finding all intersection points of a ray with the point cloud surface based on a clustering technique. However, one major drawback of this approach is that the results of classifying intersecting points are dependent on the orientation of the normal vector of original points of the point cloud surface. The original DP algorithm also belongs to the latter, which can not find the first intersection point at all. Although several intersection algorithms for the DP problem have been implemented [10,16–18], all are backward methods like the original DP algorithm and might be not robust even for the noiseless point cloud surface (or after smoothing).

Our method can be directly applied to compute the first intersection point by projecting the star point from the ray onto the point cloud surface. In Fig. 8, an example is given for showing the application of computing intersections of multiple rays and a point cloud sampled from a curved surface (see Fig. 8(a)). Here 20 test points (black points in Fig. 8(b)) are sampled from a B-spline curve around this surface, and the projection direction of each test point is chosen by the vector from the center of the B-spline curve to the test point itself. Every ray is determined by one test point and the corresponding projection direction. Some rays intersect the point cloud surface for five times at most, and our algorithm can approximate the first intersection point for each ray well(see Fig. 8(b)). However, the original DP algorithm can not give satisfactory results for most of the rays. In Fig. 8(c), note that the intersection points between the left seven rays and the point cloud surface are inside. The main reason is that the original DP algorithm is achieved through an iterative procedure. It initially computes a projection point for the whole point cloud, and then the current projection point is regarded as the new test point for the next iteration computation on a shrinking working point cloud. Since Eq. (5) gives the first projection points as inside for the left seven rays, the final projection points approximate the inside part of point cloud. In Fig. 8(d), the DP algorithm using Eq. (5) apparently finds the average position of multiple intersection points as the final intersection point for every ray.

To determine the positions of multiple intersection points between a ray and a point cloud $\mathcal{C}_N$, we might use an iterative algorithm as follows. Like the RDP algorithm, we first filter the original point cloud $\mathcal{C}_N$ using the weight function in Eq. (6) and get a small working point cloud $c_n \subseteq \mathcal{C}_N$. Then, we search a subset $Q \subseteq c_n$ to compute the projection point by the RDP algorithm and identify the rest $Q_1$ of $c_n$ as outliers (i.e. $Q_1 = c_n - Q$). The computed projection point is regarded as the first intersection point. Next, we remove the samples $Q$ from $c_n$ and update $c_n$ as $c_n = c_n - Q$. Then we repeat the RDP process and obtain the second projection point (as the second intersection point) for the updated $c_n$. The iteration process is terminated when $c_n$ is empty or enough small.

### 4.4. Limitations

Although the RDP algorithm has been presented for strengthening the robustness of the original DP algorithm, there are still some limitations in real computation and applications. This section will discuss them.

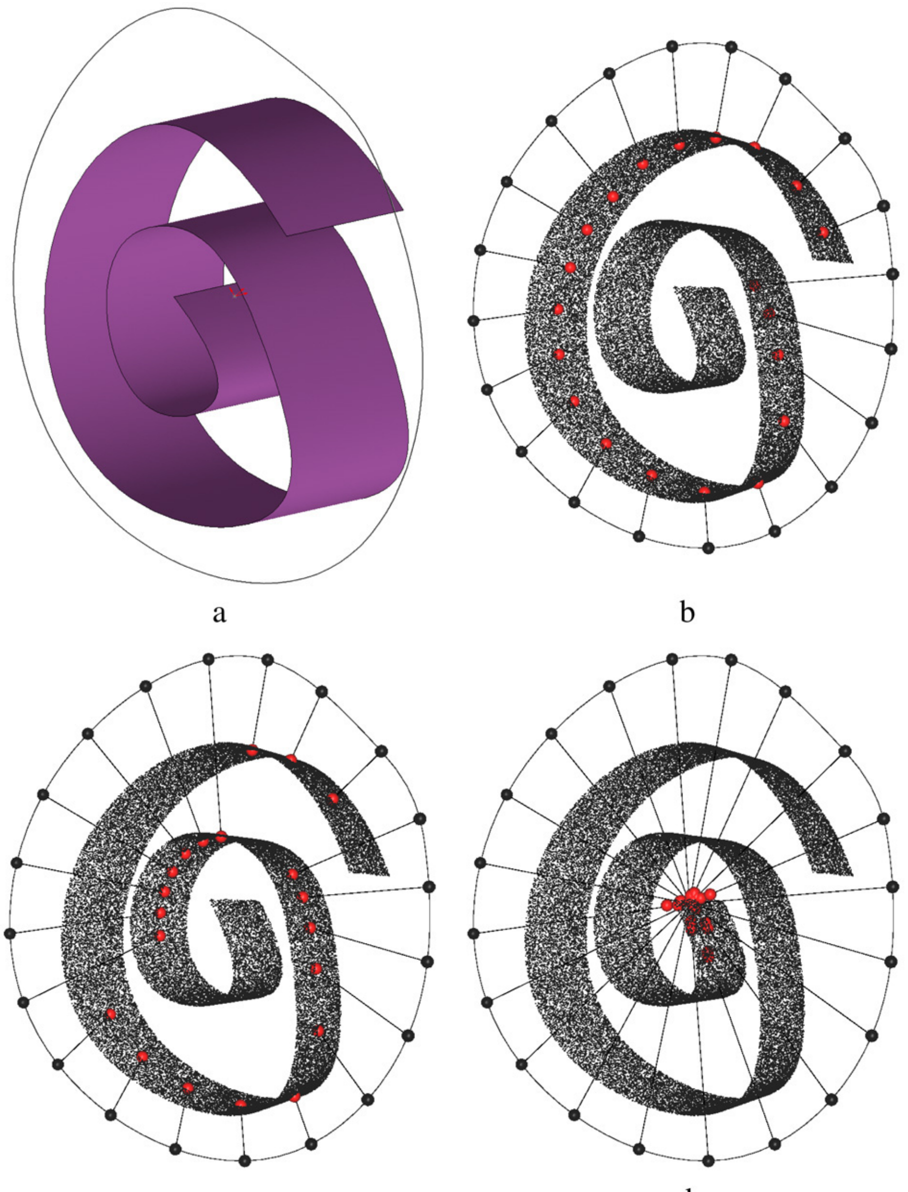
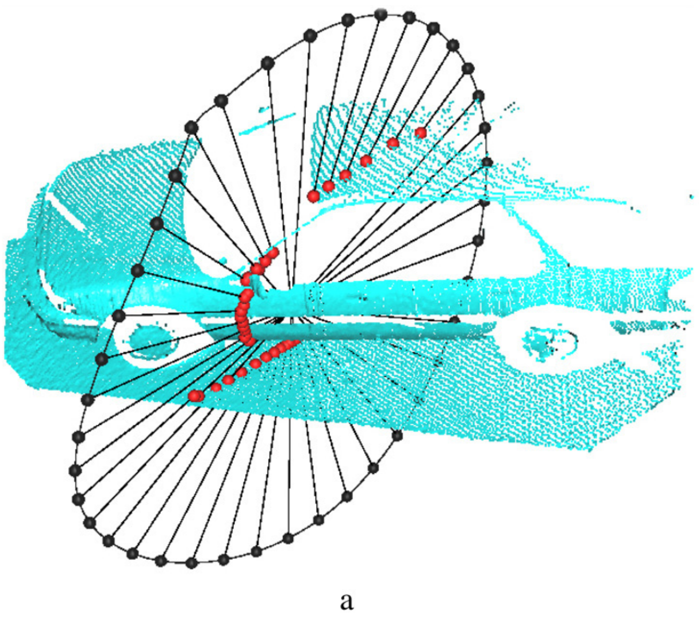


Fig. 8. One application on computing the intersection points between rays and point cloud. A point cloud consists of 80,000 points sampled from a curved B-spline surface in (a), and no additional noise is added. The computed projection points are red. (b) The result of our algorithm. (c) The result of the DP algorithm using Eq. (5). (d) The result of the DP algorithm using Eq. (6). Note that there are multiple intersection points between each ray and the curved surface. Our algorithm can find the first intersection point, whereas the original DP algorithm can not give satisfactory results for most of rays. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

**Non-uniform sampling.** Point sets acquired by 3D scanning devices usually contain irregular samples. When a point set $\mathcal{C}_N$ sampled from an underlying surface $S$ is insufficient, it is impossible to approximate point projection onto $S$ with high accuracy. The original DP algorithm is actually to interpolate a projection point by some candidate points of the point cloud by bounding a weight function. Our main improvement for the DP algorithm is the choice of candidate points based on the LMS optimization that rejects outliers including noise and disturbing points. If no outlier is detected in our algorithm, the projection result is same as the one using the original DP algorithm. The effectiveness of the original DP algorithm for point clouds with a varying density has been investigated. Azariadis and Sapidis [2] concluded that when the density is increased the corresponding projection error is reduced. A similar conclusion has been given in Ref. [10]. Fig. 9 shows an example of directed projection of points onto a point cloud with high nonuniform sampling. In this example, our algorithm can perform better than the original DP algorithm for most of test points. However, since the sampling on the roof of the car is very sparse, there are two projection points going through the roof. Highly irregularly sampled point clouds are uncommon in the CAD systems and scanned data-sets. If the sampled points are not uniformly distributed over the underlying surface, our method may lead to large approximation errors like the DP algorithm. In fact, the users do not directly operate with the original uncompleted



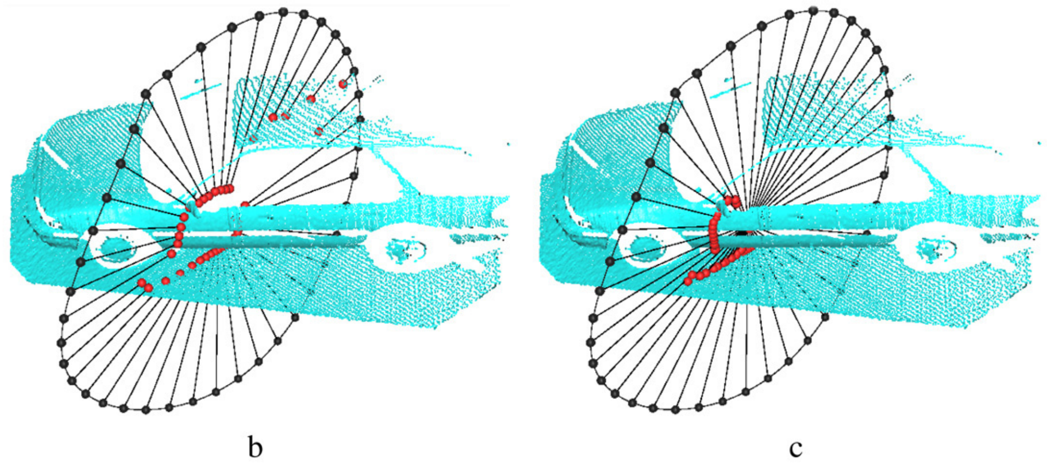


Fig. 9. Comparing the results of projecting points onto a car model with high nonuniform sampling. This model consists of 34,945 points. Here the test points (black points) are sampled from a B-spline curve around this model, and the projection direction of each test point is chosen by the vector from the center of the B-spline to the test points itself. The computed projection points are red. (a) The result of our algorithm. (b) The result of the DP algorithm using Eq. (5). (c) The result of the DP algorithm using Eq. (6). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

point cloud in most practical CAD/CAM systems. In general, it must follow two steps of preprocessing of the point cloud as follows: (1) smoothing the point data and removing the noise; (2) filling in the holes and other uncompleted regions in point cloud. The DP problem can also benefit from preprocessing.

**High curved point cloud surfaces.** In addition, since our method is based on the DP algorithm by interpolating a projection point by some weighted candidate points of the point cloud, there is also the same problem for high curved point cloud surfaces as the DP algorithm. In fact, our method can also be combined with some fitting techniques for the final projection computation for high curved point cloud surfaces. The strategy is as follows: first, the candidate points are obtained using our algorithm; then fit a bivariate polynomial surface of degree two to the candidate points; finally, compute the directed projection point onto the fitted polynomial surface. The main advantage is to accelerate the speed of projection computation. For this DP solution based on fitting, our method can offer a better candidate point set for fitting than the backward methods.

**Multiple intersection points.** The influence of disturbing points on final projection has been weakened by the RDP algorithm, as shown in Figs. 6 and 7, but it has not been solved thoroughly. For the case that the ray determined by the test point and the given projection direction goes through the point cloud surface many times, the RDP algorithm might be invalid. Fig. 10(a) illustrates a 2D invalid example for this problem. The reason for the failure is that the initial subset is selected wrongly. At the phase of initial subset selection, the original LMS uses the random sampling algorithm for selecting $k$ initial points with a small value of $k$. At each iteration, (1) $k$ points are first selected from the current working point set at random; (2) then the median of the residuals of the remaining points is computed; (3) finally, $k$ points with the minimal median are

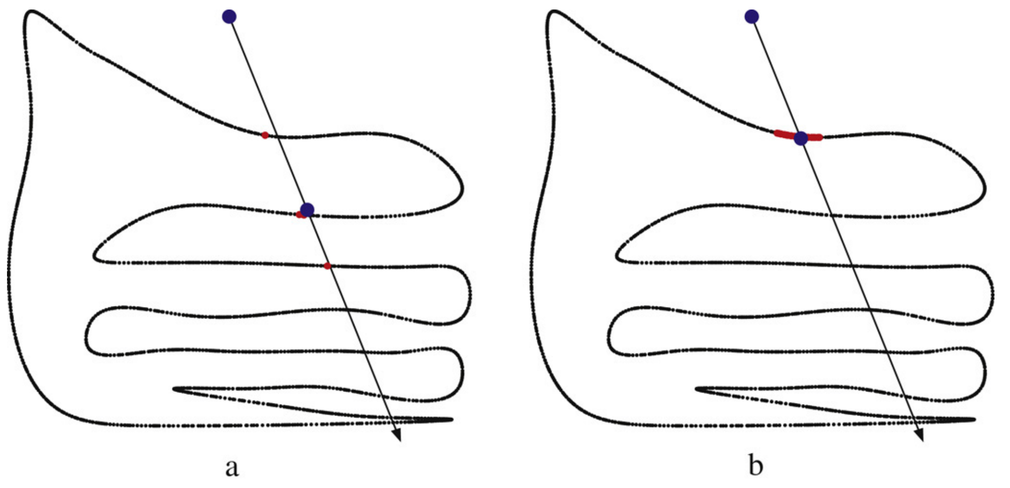


Fig. 10. If the ray determined by the test point and the given projection direction goes through the point cloud many times, the RDP algorithm based on the minimal median measurement might be invalid. (a) The result of using the minimal median measurement. (b) The improved result through changing the parameter of the RDP algorithm, i.e. replacing the median (50%) by the first quartile (25%).

selected as the initial subset for the forward search. Instead of the minimal median measurement, we may use the $m$th smallest value from the residuals of the remaining points for improving the initial subset. In Fig. 10(b), we use the first quartile (25%) instead of the median (50%) for cutting out the largest 75% outliers. The new projection point is reasonable. In fact, we found that the standard median measurement can obtain satisfactory results for most point clouds acquired by 3D scanning devices.

## 5. Conclusions

We have presented a robust, simple and efficient algorithm, so-called RDP, for projecting points onto a point cloud along an associated projection vector. The new method is based on the directed projection (DP) algorithm [1,2] and does not need any complex data structure. Our experiments show that the RDP algorithm obtains high accuracy without requiring an explicit or implicit reconstruction of the underlying surface from the point cloud. The main theoretical/methodological contributions of our work can be summarized as follows:

- We use the least median of squares (LMS) for replacing the least sum of squares used in the DP algorithm. In contrast to the original DP optimization equation, which minimizes the weighted sum of the squared distances between the point to be projected and the point cloud, our algorithm minimizes the median of the residuals. In order to effectively approximate the LMS optimization, the forward search technique is utilized instead of the backward method.
- The major advantage in the RDP algorithm is to automatically detect outliers during the directed projection phase.
- We treat the noise and disturbing points together as outliers. Noisy points could be removed by some existing smoothing algorithms, whereas the disturbing points can not be detected or removed by smoothing techniques. Our method does not depend on the smoothing procedure.
- Our method can be directly applied to approximate the first intersection point of ray/point cloud surface by projecting the star point from the ray onto the point cloud surface.

Furthermore, we also discuss the differences with the existing works, applications, and limitations on the RDP algorithm. We believe that there are numerous other applications that can benefit from the RDP algorithm.

## Acknowledgements

The authors appreciate the comments and suggestions of all anonymous reviewers, whose comments significantly improved this paper.

## References

[1] Azariadis PN. Parameterization of clouds of unorganized points using dynamic base surfaces. Computer-Aided Design 2004;36(7):607–23.
[2] Azariadis PN, Sapidis NS. Drawing curves onto a cloud of points for point-based modelling. Computer-Aided Design 2005;37(1):109–22.
[3] Liu Y-S, Paul J-C, Yong J-H, Yu P-Q, Zhang H, Sun J-G, et al. Automatic least-squares projection of points onto point clouds with applications in reverse engineering. Computer-Aided Design 2006;38(12):1251–63.
[4] Fleishman S, Cohen-Or D, Silva CT. Robust moving least-squares fitting with sharp features. In: Proceedings of SIGGRAPH'05. 2005. pp. 544–52.
[5] Klein J, Zachmann G. Point cloud collision detection. Computer Graphics Forum 2004;23(3):567–76.
[6] Mitra N, 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. 23–2.
[7] Piegl L, Tiller W. The NURBS book. Berlin: Springer; 1995.
[8] Alexa M, Behr J, Cohen-Or D, Fleishman S, Levin D, Silva C. Point set surfaces. In: Proceedings of IEEE visualization'01. 2001. p. 21–8.
[9] Alexa M, Behr J, Cohen-Or D, Fleishman S, Levin D, Silva C. Computing and rendering point set surfaces. IEEE Transactions on Visualization and Computer Graphics 2003;9(1):3–15.
[10] 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.
[11] Rousseeuw PJ. Least median of squares regression. Journal of the American Statistical Association 1984;79(388):871–80.

[12] Jones T, Durand F, Desbrun M. Non-iterative, feature-preserving mesh smoothing. In: Proceedings of SIGGRAPH'03. 2003. p. 943–49.
[13] Fleishman S, Drori I, Cohen-Or D. Bilateral mesh denoising. In: Proceedings of SIGGRAPH'03. 2003. p. 950–53.
[14] Atkinson A, Riani M. Robust diagnostic regression analysis. Springer; 2000.
[15] Weyrich T, Pauly M, Keiser R, Heinzle S, Scandella S, Gross M. Post-processing of scanned 3D surface data. In: Proceedings of eurographics symposium on point-based graphics 2004. 2004. p. 85–94.
[16] Adamson A, Alexa M. Ray tracing point set surfaces. In: Proceedings of shape modeling international 2003. 2003. p. 272–79.
[17] Schaufler G, Jensen H. Ray tracing point sampled geometry. In: Proceedings of the 11th eurographics workshop on rendering. 2000. p. 319–28.
[18] Wald I, Seidel H. Interactive ray tracing of point based models. In: Proceedings of symposium on point based graphics. 2005.