Functional Surface Reconstruction from Unorganized Noisy Point Clouds

Xueshu Liu

Dalian University of Technology, liuxs@dlut.edu.cn

ABSTRACT

As point clouds have become an important representation of objects in reverse engineering, surface reconstruction from point clouds has consequently been an active research topic over many years. In this paper a new procedure for surface reconstruction directly from point clouds is proposed. Similar to many reported works, boundary points of functional surfaces are firstly detected by the extended difference of normals operator. After the points belonging to the same functional surface are grouped by a simple algorithm, a B-spline surface is fitted to these points so as to generate an editable NURBS surface. Experimental results shown in this literature demonstrate the feasibility of the proposed method to reconstruct curved surface with high curvature.

Keywords: surface reconstruction, point cloud processing, difference of normals.

1. INTRODUCTION

Nowadays, it becomes easily to obtain large and complex object models composed of point clouds sampled from real-world objects with the help of high precision 3D scanners. Reconstructing the geometry from the raw point clouds has consequently been an active research topic in reverse engineering over the last decades. A more recently review can be found in [10, 14]. Generally speaking, reported methods can be divided into two groups regarding whether feature curves are explicitly extracted from point clouds.

In the first group, feature curves are explicitly extracted and surface reconstruction via feature curve extraction has become a strategy. Among the reported efforts Salman et al. [12] find points close to features by the approach of principal component analysis (PCA) and feature curves are subsequently recovered by clustering feature points and connecting a subset of them. However, due to the use of implicit surfaces during surface reconstruction, their approach cannot handle non-manifolds or manifolds with boundaries. Gauss map clustering is adopted in [15, 16] for feature point detection. The authors first use a global threshold to detect feature points that are close to sharp features and then a more precise iterative selection process is performed so as to obtain the exact feature points. In [3] feature points are detected by Gaussian-weighted graph Laplacian and feature curves are constructed by Reeb graph. In [4] the authors first detect feature points by voronoi-based method. The feature direction information is then determined by PCA and corner points are simultaneously distinguished. A filter is then applied to remove noisy points that are detected as feature points before feature curves are constructed. In [2], an algorithm to extract closed sharp feature lines is proposed which applies a first order segmentation to extract candidates feature points and process them as a graph to recover the sharp feature lines. In [8] moving least-squares approximation is used to estimate the local curvatures and their derivatives at a point by means of an approximating surface. The neighbor information is computed by a Delaunay tessellation and ridge and valley points are detected as zero-crossings.

Works focusing on feature preserving surface reconstruction without feature curves rebuilding have also been studied. Reported works include [9, 17] and so on. However, volume based reconstruction method expects the surface to be input as a point cloud with oriented normal vectors [9]. Voxel-based surface reconstruction algorithm can handle non-manifolds and boundaries but not sharp features [17].

Although various reconstruction methods have been reported, many problems still remain to be addressed due to geometry shape complexity. Moreover, surface reconstruction becomes harder in the presence of noise. For example, Delaunay tetrahedralization based methods work well for smooth surface reconstruction. Unfortunately, they may fail when dealing with noisy point clouds. In addition, the level set method is proved powerful and attractive when it comes to surface reconstruction. Nevertheless,

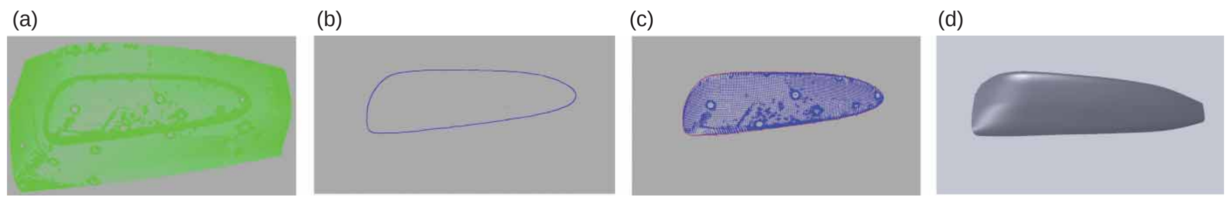

Fig. 1: Proposed method: (a) Input noisy point cloud, (b) Detected boundary of a functional surface by EDN, (c) Points corresponding to the same functional surface, and (d) Reconstructed functional surface.

finding a set of functions to form an implicit surface is difficult, especially for a free form surface.

In this paper, a new procedure for functional surface reconstruction directly from unorganized point cloud is introduced. The input to our algorithm is unorganized noisy point cloud captured from a piece of functional surface (Fig. 1 (a)), which suffers data missing and noise. A multi-scale operator named extended difference of normals (EDN) combining with a simple filter is applied to the input for obtaining boundary of the functional surface (Fig. 1 (b)). The input is then clustered according to the detected boundary by a simple algorithm (Fig. 1 (c)) and each cluster constructs a functional surface by fitting a B-spline surface to itself (Fig. 1 (d)).

2. PROPOSED METHOD

2.1. Boundary Points Detection

Given a point cloud $P = \{p_i \in R^3\}$, where $i = 1, 2, \cdots, m$ and m indicates the number of points, the normal $n(p)$ for each point $p \in P$ is often estimated ahead of any point processing technique. When $n(p)$ is computed, some neighbors $N \subseteq P$ of p are always needed. A common way to define N is by $N = \{q \in P | \mathrm{dis}(p, q) < r\}$, where r indicates a user defined threshold and dis() represents the Euclidean distance. To select a reasonable threshold r, the average distance $\bar{d}$ between points of P is computed by the following equation

$$\bar{d} = \frac{\sum_1^m \operatorname{argmin}_{q \in P, q \neq p_i} \mathrm{dis}(p_i, q)}{m} \tag{2.1}$$

where $p_i \in P$.

The estimated normal $\check{n}(p)$ at p is usually not the exact $n(p)$, that is, $\check{n}(p) \approx n(p)$. For one thing, the noisy points contained in P may affect the estimation. For the other thing, even for a clean point cloud the neighbors $N \subseteq P$ used for normal estimation may also affect the computation, which heavily depends on the location of p. An example is given in Fig. 2. The red dots indicate point p whose normal is going to be estimated and the blue ones represent the neighbors used for $n(p)$ estimation. The green arrows show the estimated normal $\check{n}(p)$ and the pink arrow represents the difference between the estimated normals obtained by using different number of neighbors. From this example we can see for the points close to local ridges and valleys their estimated normals are different if different number of neighbors are used for normal estimation. Evaluating the difference between the estimated normals brings us a multi-scale operator n_d^1 (difference of normals (DN)), which was first reported in [5] and used for feature point detection

$$n_d^1 = |\check{n}(p, r_s) - \check{n}(p, r_l)| \tag{2.2}$$

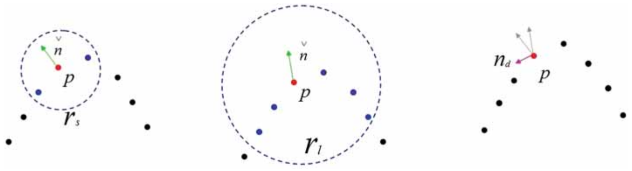

Fig. 2: Difference of normals: estimated normals may be different due to the different number of neighbors used for normal estimation.

where $r_s, r_l \in R, r_s < r_l$ signify the radii and $\check{n}(p, r)$ presents the estimated normal at p with radius r.

Provided a threshold δ, point p is regarded as a feature point if and only if $n_d^1(p, r_s, r_l) > \delta$ holds. Although the result depends on the selection of r_s and r_l, the points close to local ridges and valleys have more chance to be regarded as feature points than the others. In addition, to some extent, r_s determines how close the points from local ridges and valleys will be disregarded during feature point detection and r_l determines how far away the points from local ridges and valleys have the chance to be detected as feature points.

From the above discussion we know the exact feature points are likely to be neglected by DN operator because they are usually on the local ridges and valleys. To solve this problem, a variant of DN (VDN) is given as

$$n_d^2 = \max|\check{n}(p, r, N_i) - \check{n}(p, r, N_j)| \tag{2.3}$$

where $i, j = 1, 2, \cdots g$, $i \neq j$, $N_i \subset N(p)$, $N_j \subset N(p)$, $N_i \cap N_j = \emptyset$, $\sum_1^g N_i = N(p)$ and g is a user defined threshold.

The meaning of this operator is that $N(p)$ is divided into g groups and each group $N_i \subset N(p)$ is used to estimate $n(p)$. The difference of $\check{n}(p)$ is then evaluated and used to determine whether p is a feature point. A 2D illustrator is given in Fig. 3, where red dots indicate p and blue ones signify the neighbors of p used for normal estimation. In Fig. 3 (a) the left two neighbors are used to compute $\check{n}_1(p)$ and the right two ones are used for $\check{n}_2(p)$ in Fig. 3 (b). The red arrow in Fig. 3 (c) indicates the difference between the two estimated normals shown in Fig. 3 (a) and Fig. 3 (b), respectively.

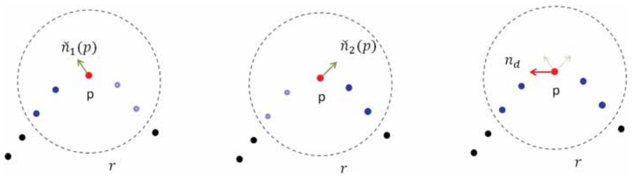

Fig. 3: Variant of difference of normals operator: (a) the left two blue dots are used for normal estimation, (b) the right two blue dots are used for normal estimation, (c) the difference between the two estimated normals.

An example is given in Fig. 4 in which the red dot enclosed by black circle indicates *p*. The neighbors *N(p)* of *p* are divided into 4 groups and each group is represented by a color in this example.

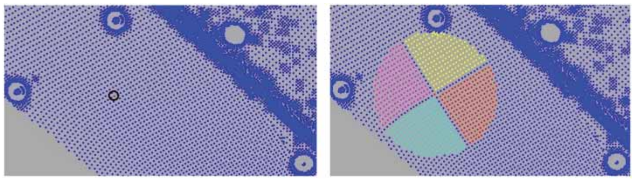

Fig. 4: Neighbors grouping. A point *p* is given as a red dot enclosed by black circle and its neighbors used for normal estimation are divided into 4 groups, which are represented by difference colors.

The grouping procedure used in this study is achieved by Algorithm 1, in which bold character ***p*** indicates the vector of coordinates of *p*.

The thinking behind this algorithm is that we use two perpendicular planes $\gamma_1(p, n_1)$ through p with normal n_1 and $\gamma_2(p, n_2)$ through p with normal n_2 to divide *N(p)* and 4 sections are generated by γ_1 and γ_2. Points in each section compose of a group. Note that n_1 is obtained by applying PCA to *V*.

Combining the two operators DN and VDN, we get an extend difference of normals (EDN) operator n_d, which inherits the advantage of multi-scale operator and avoids ignoring the points on local ridges and valleys.

$$n_d(p, r_s, r_l) = \alpha n_d^1(p, r_s, r_l) + \beta \max(n_d^2(p, r_s), n_d^2(p, r_l)) \tag{2.4}$$

where α and β are two user defined coefficients and $0 \le \alpha \le 1$, $0 \le \beta \le 1$, $\alpha + \beta = 1$.

For point $p \in P$, it is going to be regarded as a feature point when $n_d(p, r_s, r_l) > \delta$ holds. By this means the boundary points $B = \{p \in P | n_d(p, r_s, r_l)\rangle > \delta\}$ are

Algorithm 1 : Neighbor points grouping

```
Input:
  $V := N(p)$
  $N_1 = \{\}, N_2 = \{\}, N_3 = \{\}, N_4 = \{\}$
Do:
  $n_0$ :=estimated normal at $p$ by using $V$
  $n_1$ := principal vector of $V$
  $n_2 := n_0 \times n_1$
  while $V$ is not empty do
    $x$:=pop(V)
    $d_1 := (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n_1}$
    $d_2 := (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n_2}$
    if $d_1 > 0$ and $d_2 > 0$ then
      $N_1 := N_1 \cup \{x\}$
    else if $d_1 > 0$ and $d_2 < 0$ then
      $N_2 := N_2 \cup \{x\}$
    else if $d_1 < 0$ and $d_2 < 0$ then
      $N_3 := N_3 \cup \{x\}$
    else if $d_1 < 0$ and $d_2 > 0$ then
      $N_4 := N_4 \cup \{x\}$
    end if
  end while
Output:
  $N_1, N_2, N_3, N_4$
```

obtained. Note that *B* indicates a broad bound of the functional surface with noisy points (Fig. 5 (a)). To get the exact boundary, a simple filter is first adopted to remove the outliers by checking the point density, which was introduced in [11] (Fig. 5 (b)). The exact boundary points are then approximated by using Algorithm 2. Depending on the user defined threshold r_u, the approximation always needs several times computation. Fig. 5 (c) and (d) show the results after one and two times computation, respectively. After several times computation, we get the approximation boundary as shown in Fig. 1 (b).

That the detected feature points are on the two sides of the exact boundary guarantees the feasibility of Algorithm 2.

2.2. Point Cloud Clustering

After obtaining the approximated boundary points $\hat{B}$ of a functional surface, we cluster the input points according to $\hat{B}$, which is achieved by the following algorithm.

The meaning of this algorithm is that we divide *P*-$\hat{B}$ into two groups according to $\hat{B}$, that is, inside and outside. And the points enclosed by $\hat{B}$ are used for later surface reconstruction. The example given in Fig. 1 (c) shows the inside points $T \subset P$ (blue dots) and boundary points $\hat{B}$ (red dots).

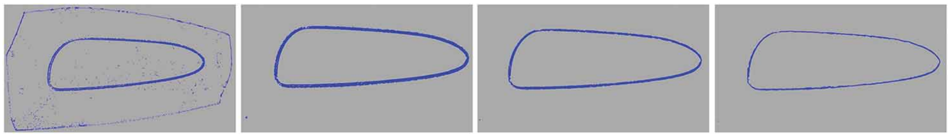
Fig. 5: Exact boundary approximation: (a) detected boundary points by EDN, (b) boundary points after outlier removal, (c) one time exact boundary approximation, (d) two times exact boundary approximation.

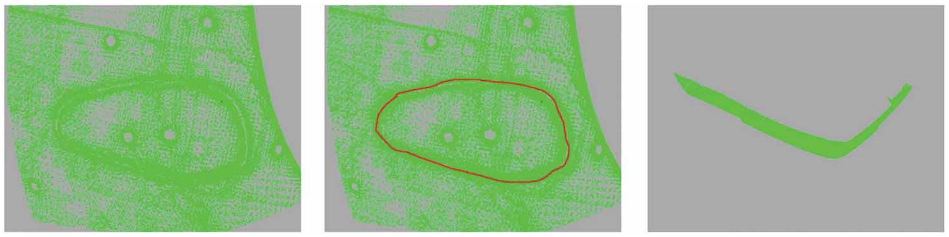
Fig. 6: Example for boundary point detection: (a) front view, (b) ideal boundary, (c) top view.

```
Algorithm 2 : Exact boundary point approximation
Input:
  $B$, $r_u$
  $\hat{B} = \{\}$
Do:
  while $B$ is not empty do
    $x$:=pop(B)
    $N(x)$:= Neighbors of $x$ defined by raidus $r_u$
    $c$:= the centroid of $N(x)$
    $\hat{B}$:=$\hat{B} \cup \{c\}$
    $B$:=$B - N(x)$
  end while
Output:
  $\hat{B}$
```

```
Algorithm 3 : Points clustering
Input:
  $V := P - \hat{B}$, $c := \frac{\sum_{p \in \hat{B}} p}{\#\{p \in \hat{B}\}}$
  $T = \{\}$
Do:
  while $V$ is not empty do
    $x$:=pop(V)
    $y$:=argmin$_{p \in \hat{B}}$ dis$(x, p)$
    if $\vec{xy} \cdot \vec{xc} < 0$ and dis$(y, c)$ > dis$(x, c)$ then
      $T := T \cup \{x\}$
    end if
  end while
Output:
  $T$
```

2.3. Functional Surface Reconstruction

For surface reconstruction, fitting method plays an important role. So far quantity of works have been reported which include [1, 6, 7, 13]. Here we fit a B-spline surface to point set T using point-distance-minimization strategy and finally we can get the desired functional surface as shown in Fig. 1 (d).

3. EXPERIMENTAL RESULTS

Note that the operators given in this literature can be performed iteratively. Given a point cloud P, a set of boundary points B can be obtained by applying an operator to P. After that, the operator is applied to B for refined feature point detection. Repeat this procedure and we can get a better result in many cases.

An example for boundary point detection of a functional surface is given in Fig. 6 (a), which contains 26975 points. The ideal result is to divide this point cloud into two groups and the boundary is given in Fig. 6 (b). However, Fig. 6 (c) shows the challenge for any feature point detection algorithm.

The results got by DN and VDN are given in Fig. 7. Fig. 7 (a), Fig. 7 (b) and Fig. 7 (c) show the results got by using DN operator on model given in Fig. 7 (a) with 1, 2 and 3 times computation respectively. Fig. 7 (d), Fig. 7 (e) and Fig. 7 (f) show the results got by using VDN operator on the same model with 1, 2 and 3 times computation respectively. It is clear that VDN is better than DN for this example. The parameters used for this computation are $r_s = 1.5\bar{d}$, $r_l = 10\bar{d}$ and $\delta = 20°$.

A better result can be got by sequentially applying DN and VDN operators to the model, which contains less undesired points (Fig. 8). The result is got by applying 1 time DN operator on model shown in Fig. 7 (f). However, it is difficult to judge when DN or VDN should be applied during feature point detection.

A substitution is to combine DN and VDN, namely, EDN. Fig. 9 shows the results of boundary detection by EDN with different coefficients of α and β. For the example given in Fig. 1 (a) DN brings us a more acceptable result while VDN for that given in Fig. 6.

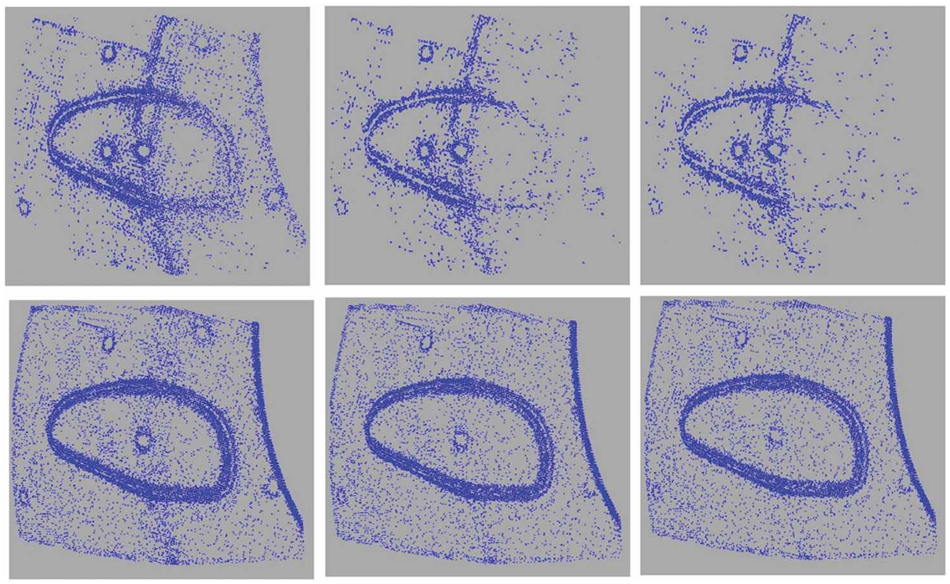
Fig. 7: Boundary detection with operators DN and VDN. Top row: results got by DN operator; bottom row: results got by VDN operator. From left to right: 1, 2 and 3 times iterative computation.

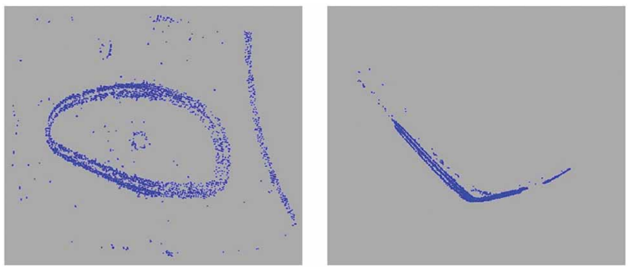
Fig. 8: Boundary point detection by two operators: (a) front view, (b) top view.

The parameters used for computation is the same as the above example. The parameters α and β used in this example are $\alpha = 1.0, \beta = 0.0$ for the first column and $\alpha = 0.4, \beta = 0.6$; $\alpha = 0.6, \beta = 0.4$ and $\alpha = 0.0, \beta = 1.0$ for the second, third and fourth column respectively. The parameters used for this test are $r_s = 1.5\bar{d}$, $r_l = 20\bar{d}$ and $\delta = 20°$. Compared with those shown in Fig. 7 and Fig. 8, although the results got by EDN are not so much desirable, they can be used for exact boundary point approximation.

Fig. 10 shows the procedure of functional surface reconstruction. Starting from Fig. 9 (f), exact boundary points are first approximated by Algorithm 2. Fig. 10 (a), (b) and (c) show the approximation results after performing one, two and three times Algorithm 2 respectively. Fig. 10 (d) shows the boundary points extracted from Fig. 10 (c), which are used to construct the boundary curve of the functional surface. After manually removing the undesired boundary points, the approximated boundary of functional surface is constructed (Fig. 10 (e)) and used for point cloud clustering by Algorithm 3. Finally, the functional surface is reconstructed (Fig. 10 (f)).

Fig. 11 gives another example of boundary detection by EDN. The original point cloud is given in Fig. 11 (a), which contains 1678267 points. Fig. 11 (b), (c) and (d) are the results by EDN for boundary points

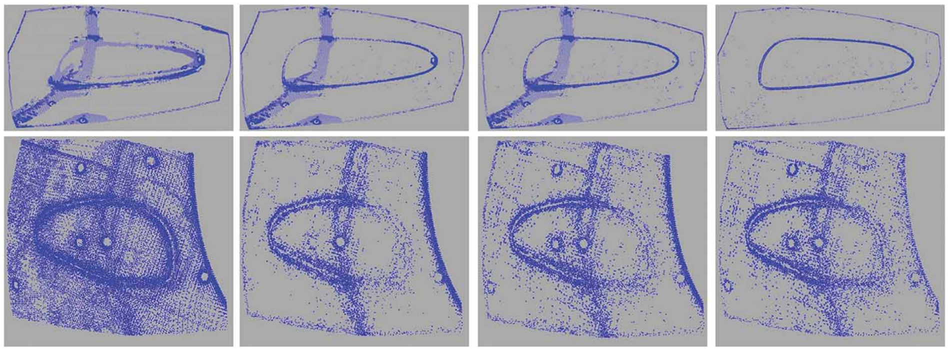
Fig. 9: Boundary detection by EDN with different coefficients.

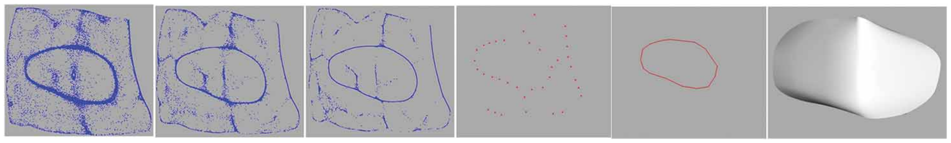

Fig. 10: Procedure of functional surface reconstruction.

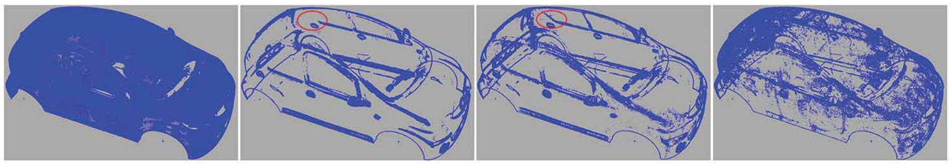

Fig. 11: Boundary detection by EDN: (a) original point cloud, (b), (c) and (d) detected boundary points with different coefficients.

detection with different coefficients. The parameters used for this test are $r_s = 1.5\bar{d}$, $r_l = 30\bar{d}$ and $\delta = 20^\circ$. $\alpha = 0.0, \beta = 1.0$; $\alpha = 0.3, \beta = 0.7$ and $\alpha = 0.6, \beta = 0.4$ are used for Fig. 11 (b), (c) and (d) respectively. For Fig. 11 (b), it is indeed the result got by DN operator. Although it looks fine, lots of details are lost as that shown in this example, where is highlighted by red curves. Currently, the computation takes almost 5 minutes on a computer with I7 3770 CPU and 32G RAM.

4. CONCLUSION AND DISCUSSION

In this paper, a new procedure for functional surface reconstruction directly from an unorganized point cloud is proposed. First, a multi-scale operator by extending the difference of normals operator is used to detect the boundary points of functional surface. The proposed method makes it possible to detect the boundary points of functional surfaces without losing too many details. The points corresponding to the same functional surface are then grouped and used to reconstruct the curved functional surface. Although the feasibility of the proposed method is proved by experimental results, modifications and improvements should be made in the future to make the proposed method more robust and efficient.

First of all, the proposed method was preliminarily tested on model with single functional surface. In case a complex model is handled (Fig. 11), an algorithm to automatically group the detected boundary points should be studied, with which the proposed method can be made more user friendly. In addition, the proposed method is currently time consuming. How to make it more efficient is another research topic. Moreover, even though an iterative computation can refine the detected boundary points, some undesired points are still inevitable included as shown in the given examples. Improving the proposed method by introducing an algorithm for undesired noisy point removal is another topic for our future study.

ACKNOWLEDGMENTS

This work is partially supported by The Education Department of Liaoning Province (Grant No. L2012012). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] Carlson, N.; Gulliksson, M.: Surface fitting with NURBS- a gauss newton with trust region approach, In Proceedings of 13th WSEAS International Conference on Applied Mathematics, 2008.

[2] Demarsin, K.; Vanderstraeten, D.; Volodine, T.; Roose, D.: Detection of closed sharp edges in point clouds using normal estimation and graph theory, Computer Aided Design, 39, 2007, 276–283, http://dx.doi.org/10.1016/j.cad.2006.12.005.

[3] Dey, T.-K.; Ge, X.; Que, Q.; Safa, I.; Wang, L.; Wang, Y.: Feature preserving reconstruction of singular surfaces, Computer Graphics Forum, 31(5), 2012, 1787–1796, http://dx.doi.org/10.1111/j.1467-8659.2012.03183.x.

[4] Dey, T.-K.; Wang, L.: Voronoi-based feature curves extraction for sampled singular surfaces, Computer and Graphics, 37, 2013, 659–668, http://dx.doi.org/10.1016/j.cag.2013.05.014.

[5] Ioannou, Y.; Taati, B.; Harrp, R.; Greenspan, M.: Difference of normal as a multi-scale operator in unorganized point clouds, ArXiv e-prints, 2012.

[6] Jiang, D.; Wang, L.: An algorithm of NURBS surface fitting for reverse engineering, The International Journal of Advanced Manufacturing Technology, 31, 2006, 92–97, http://dx.doi.org/10.1007/s00170-005-0161-3.

[7] JOO, Y.: Three-dimensional surface reconstruction of human bone using a B-spline based interpolation approach, Computer-Aided

Design, 43, 2011, 934–947, http://dx.doi.org/10.1016/j.cad.2011.03.002.

[8] Kim, S.: Extraction of ridge and valley lines from unorganized points, Multimed. Tools. Appl, 63, 2013, 265–279, http://dx.doi.org/10.1007/s11042-012-0999-y.

[9] Kobbelt, L.; Botsch, M.; Schwanecke, U.; Seidel, H.: Feature sensitive surface extraction from volume data, In SIGGRAPH, 2001, 57–66, http://dx.doi.org/10.1145/383259.383265.

[10] Lim, S.-P.; Haron, H.: Surface reconstruction techniques: a review, Artificial Intelligence Review, 2012, http://dx.doi.org/10.1007/s10462-012-9329-z.

[11] Liu, X.; Jin, C.: Feature line extraction from unorganized noisy point clouds, Journal of Computational Information System, in press.

[12] Salman, N.; Yvinec, M.; Merigot, Q.: Feature preserving mesh generation from 3D point clouds, In Symposium on Geometry Processing, 2010, 1623–1632, http://dx.doi.org/10.1111/j.1467-8659.2010.01771.x

[13] Sun, Y.; Guo, D.; Jia, Z.; Liu, W.: B-spline surface reconstruction and direct slicing from point clouds, The International Journal of Advanced Manufacturing Technology, 27, 2006, 918–924, http://dx.doi.org/10.1007/s00170-004-2281-6.

[14] Tang, R.; Halim, S.; Zulkepli, M.: Surface reconstruction algorithms: review and comparisons, In ISDE2013, 2013.

[15] Weber, C.; Hahmann, S.; Hagen, H.: Sharp feature detection in point clouds, In IEEE International Conference on Shape modeling and Applications, 2010, 175–186, http://dx.doi.org/10.1109/SMI.2010.32

[16] Weber, C.; Hahmann, S.; Hagen, H.; Bonneau, G.: Sharp feature preserving MLS surface reconstruction based on local feature line approximations, Graphical Models, 74(6), 2012, 335–345, http://dx.doi.org/10.1016/j.gmod.2012.04.012

[17] Wang, J.; Oliveira, M.; Kaufman, A.: Reconstructing manifold and non-manifold surface from point clouds, In IEEE Visualization, 2005, 415–422.