

Robust and Parallel Mesh Reconstruction from Unoriented Noisy Points



SHEUNG, Hoi

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Philosophy

in the

Mechanical and Automation Engineering

The Chinese University of Hong Kong

July, 2009



Thesis/ Assessment Committee

Professor Hui, Kin-chuen (Chair)

Professor Charlie C. L. Wang (Thesis Supervisor)

Professor Chung, Chi-kit Ronald (Committee Member)

Professor Ma, Weiyin (External Examiner)

Abstract of thesis entitled

In this work, we present a robust method to generate mesh surface from unoriented noisy points. The whole procedure consists of three steps. Firstly, the normal vectors at points are evaluated by a highly robust estimator, which can fit surface corresponding to less than half of the data points and fit data with multi-structures. This benefits us with the ability to well reconstruct the normal vectors around sharp edges and corners. Meanwhile, clean point cloud equipped with normal vectors is obtained by projecting points according to the robust fitting. Secondly, an error-minimized subsampling is applied to generate a well-sampled point cloud. Lastly, a triangular mesh is reconstructed by local triangulation for fast computation or Tight Cocone algorithm for better triangulation quality. A polygonal mesh which preserves sharp features can be constructed by the dual-graph of triangular mesh. The algorithms exploited here are highly parallelized to take advantage of the single-instruction-multiple-data (SIMD) parallelism that is available on consumer level graphics hardware. Our approach has been applied and succeeded in reconstructing several piecewise-smooth surfaces with sharp features preserved from noisy point clouds.

Submitted by Hoi Sheung

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in June, 2009

摘要

本論文提出了一個魯棒的方法以從沒有法向信息並且受噪聲影響的點雲中，重構出網格曲面。整個算法由三個步驟組成：首先，我們利用一個高度魯棒的估計函數求出點的法向。這個估計函數能夠僅用不到一半的點雲數據或是多結構的點雲數據進行曲面擬合。得益於上述性質，位於尖銳特徵周圍的法向都可以正確地重建。同時我們利用一種魯棒的擬合方法將點雲投影，就可以得到消除了噪音並配備法向的點雲。然後，我們對點雲進行子採樣，將採樣誤差最小化以得到良好的結果。最後，我們可以使用局部三角化方法快速重構三角網格，或採用 Tight Cocone 算法重構出三角片質量較好的網格。通過找出三角網格的對偶圖 (dual-graph)，構建出保留模型尖銳特徵的多邊形網格。本論文提出的算法能夠利用一般消費級圖形硬件中的單指令多數據流 (SIMD) 進行高度並行計算。我們已將此方法應用到有噪音的點雲數據上，並成功重構出分段光滑且保留完好尖銳特徵的曲面。

Acknowledgements

I offer my heartfelt thanks to my supervisor Dr. Charlie C. L. Wang. His invaluable advices and guidances give me a clear view on my research direction. I admire him for both of his enthusiasm and scientific knowledge. I also thank my co-examiners for their suggestions of improvements. I had a great time at the Shape Modeling Group at the Department of Mechanical and Automation Engineering in CUHK and I hope that our cooperation will continue in the future. I am deeply appreciative of the help of my colleagues, Yunbo Zhang, Debbie Yuen-Shan Leung, Shengjun Liu, Hanli Zhao, Jiayi Xu, Wei-Lun Tsai, Ya-Tien Tsai, Juncong Lin, Penelope Watkins, Tom Tsz-Ho Kwok, Pu Huang and Samuel Sai-Man Li. I have gained a lot of new knowledge from the discussions with these talents.

Last but not least, a very special thank to Siu Ping Mok for proofreading my papers, reports and this thesis.

This research work is partially supported by CUHK Direct Grant CUHK/2050400, Hong Kong RGC CERG Grant CUHK/417508 and CUHK/416307, and ITF TS/026/07.

Contents

Abstract	v
Acknowledgements	ix
List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Main Contributions	3
1.2 Outline	3
2 Related Work	5
2.1 Volumetric reconstruction	5
2.2 Combinatorial approaches	6
2.3 Robust statistics in surface reconstruction	6
2.4 Down-sampling of massive points	7
2.5 Streaming and parallel computing	7
3 Robust Normal Estimation and Point Projection	9
3.1 Robust Estimator	9
3.2 Mean Shift Method	11
3.3 Normal Estimation and Projection	11
3.4 Moving Least Squares Surfaces	14
3.4.1 Step 1: local reference domain	14
3.4.2 Step 2: local bivariate polynomial	14
3.4.3 Simpler Implementation	15
3.5 Robust Moving Least Squares by Forward Search	16
3.6 Comparison with RMLS	17
3.7 K-Nearest Neighborhoods	18
3.7.1 Octree	18
3.7.2 Kd-Tree	19
3.7.3 Other Techniques	19
3.8 Principal Component Analysis	19
3.9 Polynomial Fitting	21
3.10 Highly Parallel Implementation	22

4	Error Controlled Subsampling	23
4.1	Centroidal Voronoi Diagram	23
4.2	Energy Function	24
4.2.1	Distance Energy	24
4.2.2	Shape Prior Energy	24
4.2.3	Global Energy	25
4.3	Lloyd's Algorithm	26
4.4	Clustering Optimization and Subsampling	27
5	Mesh Generation	29
5.1	Tight Cocone Triangulation	29
5.2	Clustering Based Local Triangulation	30
5.2.1	Initial Surface Reconstruction	30
5.2.2	Cleaning Process	32
5.2.3	Comparisons	33
5.3	Computing Dual Graph	34
6	Results and Discussion	37
6.1	Results of Mesh Reconstruction form Noisy Point Cloud	37
6.2	Results of Clustering Based Local Triangulation	47
7	Conclusions	55
7.1	Key Contributions	55
7.2	Factors Affecting Our Algorithm	55
7.3	Future Work	56
A	Building Neighborhood Table	59
A.1	Building Neighborhood Table in Streaming	59
B	Publications	63
	Bibliography	65

List of Figures

1.1	Overview of our robust mesh reconstruction pipeline	2
3.1	A single outlier can greatly distort a least squares fit	10
3.2	Robust estimator	13
3.3	The procedure of MLS projection	15
3.4	The first two projection steps of [ARAA04]	16
3.5	Performance comparison of different estimators at a highly noisy corner region	17
3.6	A miss-projected point	18
3.7	Local neighborhood and covariance analysis	20
4.1	Example showing zero energy by $E_P^S(x)$	25
4.2	Clustering with different weight ratios	26
4.3	Geometric meanings of E_{shape} and E_{dist}	26
4.4	Example showing clustering steps	27
5.1	An example showing that duplicate triangles are avoided during local triangulation	31
5.2	An example showing that the non-manifold parts are cleaned and the holes are filled.	33
5.3	Close view of reconstructed surfaces in region of close structures	33
5.4	Dual graph computation	34
6.1	Reconstruction of the Octa-flower model	39
6.2	Geometric deviation of Octa-flower	39
6.3	Reconstruction of the Gear model	40
6.4	Geometric deviation of Gear	40
6.5	Reconstruction of the Hub model	41
6.6	Geometric deviation of Hub	41
6.7	Reconstruction of the Dragon model	42
6.8	Projection test of RMLS versus our approach on an edge	43
6.9	The mesh reconstruction of real scanned data <i>Carter</i>	44
6.10	The change of energy E_{global} during clustering optimizations	45
6.11	Users can directly control the desirous number of vertex in resultant mesh by specifying parameter N_C	49
6.12	Reconstruction of <i>Filigree</i> and a mechanical model <i>Engine</i>	50
6.13	Two more examples reconstructed from scattered point data	51
6.14	Comparison of processing time of our approach with Ohtake's	53
6.15	The energy graph of clustering	54

A.1	Uniform partitioning of the point data and streaming to process one layer of cells with neighboring two layers	60
A.2	Data structure of <i>CellBuffer</i>	60

List of Tables

6.1	Computational Statistics	46
6.2	Computational statistics of processing time in each step, and the geometry approximation errors measured by Metro	52
6.3	Details of testing Ohtake's program in [OBS05b].	53

Chapter 1

Introduction

The reverse engineering problem for reconstructing three dimensional models from un-oriented noisy point clouds has been problematic for more than decades in applications of computer graphics, virtual reality and CAD/CAM. At present, many 3D surface scanning devices (using different sorts of methods like structured light, stereo vision based scanners) are available. These devices always generate unstructured clouds of measurement points in \mathbb{R}^3 . With no surprise, measurement noises embedded in these points cannot be avoided, which make the downstream mesh reconstruction very troublesome. In addition, as such acquisition devices become more and more accurate, the number of points employed to capture the shape of an industrial model has increased significantly. Nowadays, several hundred thousands of points are usually involved. This greatly enhances the possibility of reconstructing accurate models in a computer system. However, it also makes the reconstruction procedure very time-consuming. Therefore, it is impractical to use those algorithms which are designed for processing tens of thousands of points (e.g., [HDD*92, BBX95, DG03]) to process a massive number of points. Furthermore, the preservation of sharp features is an important issue especially in reconstructing mechanical models such as gears. It is a big challenge to preserve those sharp features in the presence of noises as simple smoothing techniques like [Lev03] will probably smooth them out. To solve all these problems, we develop a highly parallel meshing pipeline to reconstruct mesh surface from a massive number of unoriented noisy points by using the strong computational power which is available on consumer graphics hardware with graphics processing unit (GPU).

The existing work in literature can be classified into two major groups: 1) computational geometry approaches and 2) volumetric reconstruction techniques. The computational geometry approaches are usually based on the Voronoi diagram of a given point cloud



Figure 1.1: Overview of our robust mesh reconstruction pipeline: (leftmost) the given fandisk model with 18% Gaussian noises randomly distributed in the range of 0.5% of the bounding box’s diagonal length, (left) the fandisk model with normals estimated and outliers removed, (middle) the clustering result of subsampling and the points in different colors belong to different clusters, (right) the triangulation result on down-sampled points, and (rightmost) the final resultant mesh model with sharp features preserved.

and reconstruct a mesh surface by directly linking the input samples. Normal information is not required. However, it is generally difficult to avoid including noises in the final reconstructed surface. Moreover, as both the memory and the time cost to compute Voronoi diagram are expensive, these approaches are always applied to a small or medium size of point clouds (e.g., input with less than 50K points).

The volumetric reconstruction techniques attempt to build a signed implicit function that interpolates or approximates the point cloud samples, and then reconstruct its isosurface using, e.g., the Marching Cubes algorithm [LC87]. Nevertheless, the computation of such a signed implicit function requires the point cloud samples to be equipped with normal vectors, which can hardly be obtained directly from scanning devices. The estimation of normals on given cloudy points is actually one of the most critical steps in the reconstruction pipeline – especially when the points are in the presence of noise, sharp features, or thin structure. Although the input samples can be denoised slightly by applying the approximation scheme of implicit function reconstruction (e.g., [OBS05a]), the sharp features are always blurred together with noise. Furthermore, the approximation techniques like least-square fitting in general cannot satisfactorily handle the outliers.

We present a robust mesh reconstruction method from noisy points in this thesis. In contrast to existing approaches, we evaluate normal vectors on noisy point cloud by a highly robust estimator which allows us to reconstruct normals which well preserve sharp features. The points are projected to the robustly fitted surface. The massive clean points resulted are further down-sampled into user specified number of points. The subsampling is based on an iterative clustering algorithm and with the shape-approximation-error minimized. Note that, since only the position and the tangent

plane at a sample are required for the clustering, the direction of normals determined in the first step is unnecessary to be consistently pointing outwards (or inwards) which is very difficult in surface reconstruction pipeline. Lastly, the down-sampled points are connected into a triangular mesh and its dual-graph, a polygonal mesh preserving sharp features, is computed. Figure 1.1 gives an overview of the steps in our approach. In order to borrow the advanced computational power available on consumer PCs, all the steps are parallelized using the architecture of GPU.

1.1 Main Contributions

Our aim is to reconstruct surface from unoriented points destroyed by a large amount of noises. Parallelism approach is exploited in order to speed up the whole process greatly. The main contributions of this thesis fall in the following aspects:

- A highly parallel and robust normal estimation and point projection method to deal with noisy point cloud while preserving the shape of sharp features;
- A parallel subsampling method of points with shape-approximation-error minimized;
- A new, efficient and effective mesh reconstruction pipeline based on the two techniques stated above.

Thanks to the most advanced parallel computation power which has been available on consumer level PCs, the efficiency of our approach becomes much outstanding.

1.2 Outline

This thesis is organized as follows:

- In Chapter 2, we first review the work of volumetric reconstruction which is the foremost approach in reconstructing surface from unoriented points. After introducing another approach from the computational geometry point of view, we focus on the robust statistics based methods as these kinds of approaches have gained much more attention and much works have been published recently. Techniques in subsampling of massive points are then described. Lastly, the work in streaming and parallel computing are pointed out concisely.

- Chapter 3 explains the main concepts of robust normal estimation and point projection. The robust estimator is introduced first and then the detailed algorithms for estimating normals and projecting points are described. To speed up this most time-consuming step in our surface reconstruction pipeline, we also present the idea of parallel implementation on GPU.
- Chapter 4 describes how to down-sample the point cloud into a smaller number of points. The shape-approximation-error of subsampling is minimized. Unlike other flooding based resampling approaches, our algorithm based on local update can be easily parallelized.
- In Chapter 5, we explain the local triangulation based on the clustering result for a fast generation of mesh. We also describe the use of Tight Cocone algorithm to obtain a triangular mesh of better quality. Then the method of generating a sharp feature preserved mesh surface from the initial surface is presented.
- In Chapter 6, we apply our reconstruction pipeline to a set of models to examine its performance in practice. The results obtained have demonstrated the advantage of our approach in preserving sharp features. Statistical analysis has also shown the small shape errors on our reconstructed models.
- Lastly in Chapter 7, we summarize the work conducted in this research. Some possible enhancements are also suggested as a future work.
- The appendix supplements the implementation of generating the neighborhood table of points on GPU.

Chapter 2

Related Work

The related work in the aspects of volumetric reconstruction, combinatorial approaches, robust statistics in surface reconstruction, down-sampling of massive points, and streaming and parallel computing are reviewed in this chapter.

2.1 Volumetric reconstruction

Pioneered by the work of Hoppe et al. [HDD*92], the approaches in this category always start from estimating normals by a local principal component analysis (PCA), followed by a graph search to unify their inside/outside direction. Then, the samples equipped with normals are used to construct an implicit function in the forms of signed distance field [HDD*92, HDD*94], piecewise algebraic surfaces combined with α -shape [BBX95], globally supported radial basis functions (RBF) [TO99, CBC*01], compactly supported RBF [SAAY06], blended quadratic functions [OBA*03, OBS05a, XMQ04], or 3D indicator functions [Kaz05, KBH06, BKBH07, ACSTD07]. Afterwards, a Marching Cubes (MC) algorithm [LC87] is employed to reconstruct the surface at the zero level-set of the implicit function. All these approaches rely on the input of oriented points, which is however difficult to be obtained from scanning devices. Recently, Hornung and Kobbelt developed a method in [HK06] to reconstruct watertight 3D models from point clouds without normal information. They converted the surface reconstruction into a minimum cut problem of a weighted spatial graph structure. A mesh template fitting based method was proposed in [SLS*06] to realize a similar function. However, there is no direct extension of these methods to generate mesh surface preserving sharp features.

2.2 Combinatorial approaches

The problem of mesh reconstruction was also approached from the computational geometric point of view. Amenta et al. in [ABK98, ACK01] gave a provable guarantee of reconstructing a correct model given a minimum sampling density dependent on the local feature size. Recently, the approach was extended to be able to handle noisy input in [MAVdF05]. However, as they did not remove outliers, the quality of resultant meshes was not good. Several variations of [ABK98] are available in [ACDL00, DGH01, DG03]. Recently, Kuo and Yau in [KY06] proposed a combinatorial algorithm to triangulate a given point cloud with sharp features. When applying their algorithm to practical data sets, there are two difficulties: 1) similar to other combinatorial approaches, it is sensitive to noise and 2) the measured points rarely locate along the sharp features. Furthermore, it is uneasy to apply this sort of method to massive points.

2.3 Robust statistics in surface reconstruction

The computer graphics community pays more attention to the robust statistics based methods recently. Ivriissimtzis et al. [IJS03] employed neural network as a triangular mesh to connect sample points and updated the mesh respectively. A method to quantify uncertainty in point cloud data by analyzing how far a point agrees with locally weighted planes has been proposed in [MP04]. The authors in [SSB05] used support vector machine for reconstruction, hole filling and morphing between data sets. Schall et al. in [SBS05] employed locally defined kernels to analyze the point neighborhood, and then computed a global surface probability distribution. However, all these approaches did not solve reconstruction problems of a surface fitting corresponding to less than 50% of the data points or a surface fitting to multi-structure. Such a problem was first addressed in [FCOS05] by a forward search approach, but they projected points onto moving least squares (MLS) surfaces instead of reconstructing explicit meshes. The techniques employed in [DTB06] and [JWB*06] are quite similar, where both used a Gaussian error model in conjunction with surface priors and performed numerical optimization to maximize the posterior probability of the model. The work in [DTB06] focuses on a given triangular mesh, and Jenke et al. in [JWB*06] processed point clouds into well-sampled ones with noise removed. Nevertheless, since they are based on region growing, their computations are very time-consuming and can hardly be parallelized. Unlike [FCOS05] and ours, the method in [DTB06] relies on the sharp edge identification and it may fail if the noise is less than the selected curvature threshold. Moreover, as shown in Fig.3.2, ours outperforms the method of [FCOS05] on the point set with

a very low signal-to-noise ratio. Recently, Schnabel et al. in [SDK09] presented a different approach to reconstruct the mesh guided by a set of primitive shapes (planes, spheres, cylinders, cones and tori) that are fitted to the input point cloud. With this guidance, a closed, sharp features preserved and noise free mesh can be reconstructed from the input which contains large holes and noises. However, the small size and variety of the set of primitive shapes limit the reconstruction in many cases especially the free-form objects. On the contrary, our approach does not have such a limitation as demonstrated in Fig. 6.7.

2.4 Down-sampling of massive points

Given a point set, the decimation process in [ABCO*03] repeatedly removes the point that contributes the smallest amount of information to the shape. Kalaiah and Varshney [KV03] represented surfaces by a sampled collection of differential points and offered a novel point-based simplification technique that factored in the complexity of local geometry. Song and Feng [SF08] studied the problem of point cloud simplification by searching for a subset of the original input data set according to a user-specified number of points. Liu et al. [LWL*08] applied quasi-Newton methods to compute Centroidal Voronoi Diagram (CVD) and demonstrated a faster convergence than Lloyd's method [Llo82]. The time cost of computation in these approaches however is very expensive, and it is difficult for them to be parallelized. A technique very similar to ours was [VCP08] where Valette et al. proposed a local update scheme, but not K-means [CSAD04] or Lloyd relaxation, to compute CVD on a given mesh surface, and then remeshed the given surface according to CVD. Although [LWL*08] can generate better remeshing quality in terms of the regularity of vertex degrees and shape of the triangle faces, [VCP08] performs much faster due to the local boundary edge updating method. The most significant difference between [VCP08] and our work is that they tried to locate the seeds of Voronoi diagram along sharp features, which however in general cannot be guaranteed (e.g., the result in the last figure of [VCP08]). Here, we approximate the given point clouds with a set of proxies (i.e., Voronoi diagram). No matter the seeds of Voronoi diagram are along the sharp features or not, we can still reconstruct sharp features by using the points coupled with the normals.

2.5 Streaming and parallel computing

The Streaming technique has been employed in surface reconstruction from massive points for a long time, where the most recent work was [BKBH07]. In [ZGHG08], a

GPU-based implementation of [KBH06] has been developed using NVIDIA's CUDA. Nevertheless, the reconstruction with sharp features preserved and the reconstruction on noisy input have not been addressed. Regarding interpolating surfaces, it is rather common to use the *advancing front techniques* (AFT). AFT begins with a minimal subset of final reconstruction and iteratively expands its boundary until every part of the surface is covered. Therefore, its degree of parallelization is rather low. Bernardini et al. [BMR*99] used a ball which is pivoted around the edges of mesh boundaries to add subsequent triangles to the seed triangle. Besides, Crossno & Angel [CA99] introduced local triangulations which mark a point as finished only if it is completely surrounded by triangles. Recently, a GPU interpolating reconstruction method which makes use of local Delaunay triangulation presented by Gopi et al. [GKS00] has been put forward by Buchart et al. [BBA08]. The method starts by identifying the k -nearest neighbors to each point on the CPU. It proceeds onward by sorting the neighbors of each point on the GPU in the order of their angles surrounding it and performing local Delaunay triangulation. However, the noisy input has not been considered.

Chapter 3

Robust Normal Estimation and Point Projection



To robustly estimate normal, we fit a surface to the local shape around a sample point in \mathbb{R}^3 . Then the point is projected onto the fitted surface and the normal vector of the projected point is estimated. In this chapter, the basic concepts of robust estimator are described. The related work of moving least squares (MLS) surfaces is also reviewed, as well as its extension, robust moving least squares with sharp features. The techniques in estimating local surface properties such as normals, neighborhoods and fitting polynomials are explained in detail. Lastly, we present the highly parallel implementation which can utilize the power of GPU. In our work, we choose a very robust one – Maximum Density Power Estimator (MDPE) and adopt it in a highly parallel algorithm of normal estimation and projection.

3.1 Robust Estimator

The most classical method for fitting a model to data is linear regression using least-squares. However, as carefully discussed in [FCOS05], a single sample with a large error,

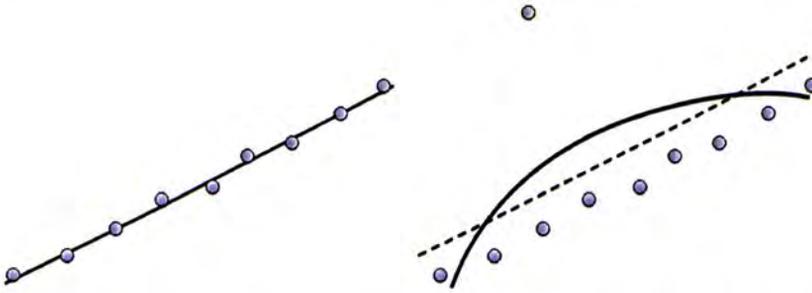


Figure 3.1: A single outlier can greatly distort a least squares fit: (left) no outlier and (right) one outlier only. Increasing the order of fitted model does not help either.

called *outlier*, can change the fitted model arbitrarily. More specifically, as shown in Fig.3.1, a single outlier can fail a least-squares fit. Robust estimation techniques try to fit a model to data that contain outliers. A robust estimator of local shape is very useful and important when the given point sets are in the presence of noises. Generally speaking, when a model is correctly fitted, the following two criteria should be satisfied:

- There are as many as possible data points on or near the model;
- The residuals of inliers should be as small as possible.

The least squares method uses the second criterion as its objective function to minimize the residuals without distinguishing the inliers from outliers. MUSE [MS96], the technique employed in [FCOS05] tries to minimize the scale estimate provided by the k th ordered absolute residual instead of minimizing the residual of inliers. Wang and Suter presented an estimator, MDPE, in [WS04] which considers both of these two criteria in its objective function. In comparison, it outperforms other estimators (RESC, ALKS, LMedS, RANSAC and Hough Transform) by tolerating more than 85% of outliers. MDPE is based on the strategy of random sampling [FB81] to choose p points (called a p -subset) and then determine the parameters of a model for this p -subset, where for example $p = 2$ for a line, $p = 3$ for a circle or plane, and $p = 6$ for a quadratic curve. It finally outputs the parameters determined by a p -subset with the minimum or maximum of the respective objective function.

Briefly, if the model to fit has been correctly estimated in MDPE, the data points on or near the fitted structure should have a high score in the following probability density power function

$$DP = \frac{\sum_{X_i \in W_c} \hat{f}(X_i)}{\exp(|X_c|)} \quad (3.1)$$

where X_c is the center of the converged window W_c obtained by applying the mean-shift procedure, and $\hat{f}(X_i)$ is the multivariate kernel density estimator defined on a set of

points $\{X_i\}_{i=1,\dots,n}$ in a d -dimensional Euclidean space \mathfrak{R}^d as

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{x - X_i}{h}\right) \quad (3.2)$$

with the window band-width h and the Epanechnikov kernel K yielding minimum-mean integrated square-error. The kernel is defined as

$$K(X) = \begin{cases} \frac{1}{2}c_d^{-1}(d+2)(1 - X^T X) & X^T X < 1 \\ 0 & \text{else} \end{cases} \quad (3.3)$$

where c_d is the volume of a unit d -dimensional sphere, e.g. $c_1 = 2$, $c_2 = \pi$, and $c_3 = 4\pi/3$. $d = 1$ is employed in our following normal estimation and point projection.

3.2 Mean Shift Method

To compute the center X_c of the converged window W_c on a given set of n data points $\{X_i\}_{i=1,\dots,n}$, the mean-shift update vector can be derived from the gradient of the kernel density estimate in Eq.(3.2). In short, the mean-shift update vector is defined as

$$M_h(x) = \frac{1}{n_x} \sum_{X_i \in S_h(x)} X_i - x \quad (3.4)$$

where the region $S_h(x)$ is a hypersphere with radius h containing n_x data points. The mean-shift procedure is as follows:

Algorithm 1: Mean Shift

Choose the radius h of the search window;

Initialize the location X_c of the window with zero;

repeat

 Compute the mean shift vector $M_h(x)$ by Eq.(3.4);

$X_c \leftarrow X_c + M_h(x)$;

until reach the terminal condition ;

We stop the iteration when either $|\|M_h^{last}(x)\| - \|M_h(x)\|| < 0.01\|M_h^{last}(x)\|$ or the loop has run more than 300 times.

3.3 Normal Estimation and Projection

The above estimator is conducted to find a quadratic surface best fitting the local shape around a sample x . The basic idea is that, p points are randomly selected from

the neighbors $N(x)$ of the given sample x to fit a quadratic surface S , and then the probability density power DP according to this fit S is evaluated by the residuals of points in $N(x)$ to S . The estimation will be repeated for m times, and among the m fits, the surface with the maximal score in DP is utilized as the robust fitting result.

More specifically, the robust estimation starts from choosing a search window radius h for MDPE and a repetition count m . The value of h greatly affects the robustness, the smaller h is used, the more sensitive to noises the estimator is. However, some inliers may be ignored if h is too small. By experiences, we choose $h = 2\bar{L}$ in all our examples (except the model with nonuniform point density), where \bar{L} is the average of point distances on the given model. Theoretically, the value of m relates to the probability P that at least one clean p -subset is chosen from m p -subsets as

$$m = \frac{\log(1 - P)}{\log[1 - (1 - \varepsilon)^p]}, \quad (3.5)$$

where ε is the fraction of outliers.

After randomly selecting p -subset, the points are used to form a quadratic surface S . Here, we first compute the centroid of the p points, and employ the principal component analysis (PCA) to form a local coordinate-frame at the average position (ref. [Pet02]). Then, the surface S

$$S(s, t) = as^2 + bt^2 + cst + ds + et \quad (3.6)$$

is fitted by the mapped coordinates of these p points at this local coordinate-frame. Fleishman et al. in [FCOS05] suggested to let p equal the number of parameters in a quadratic surface to fit. Here we do not let $p = 5$ although there are only five parameters (a, b, c, d, e) to be determined in Eq.(3.6). Instead, we use $p = 6$ and then determine $S(s, t)$ by computing the least-square solution with singular value decomposition (SVD), which makes the model fitting numerically more stable.

By a fit, the residuals of all points in $N(x)$ to the determined surface $S(s, t)$ are used as input to the mean-shift procedure to compute a converged window. Note that, the mean shift is conducted in one-dimensional space – signed residual space. Lastly, the value of the probability density power function, DP in Eq.(3.1), is scored for this fit, S . The surface fitting will be applied for m times. Among all m fits, the fitted surface with maximum DP is regarded as the best surface S^* . Then the projected position x' of x is the closest point $x_c \in S^*$ to x which is searched by Newton's method. The normal of surface S^* at x_c is employed as the normal vector to equip x' .

Using the value of m defined in Eq.(3.5) as the number of repetition is impractical. There are two reasons for this. Firstly, we do not know the value of ε , the fraction

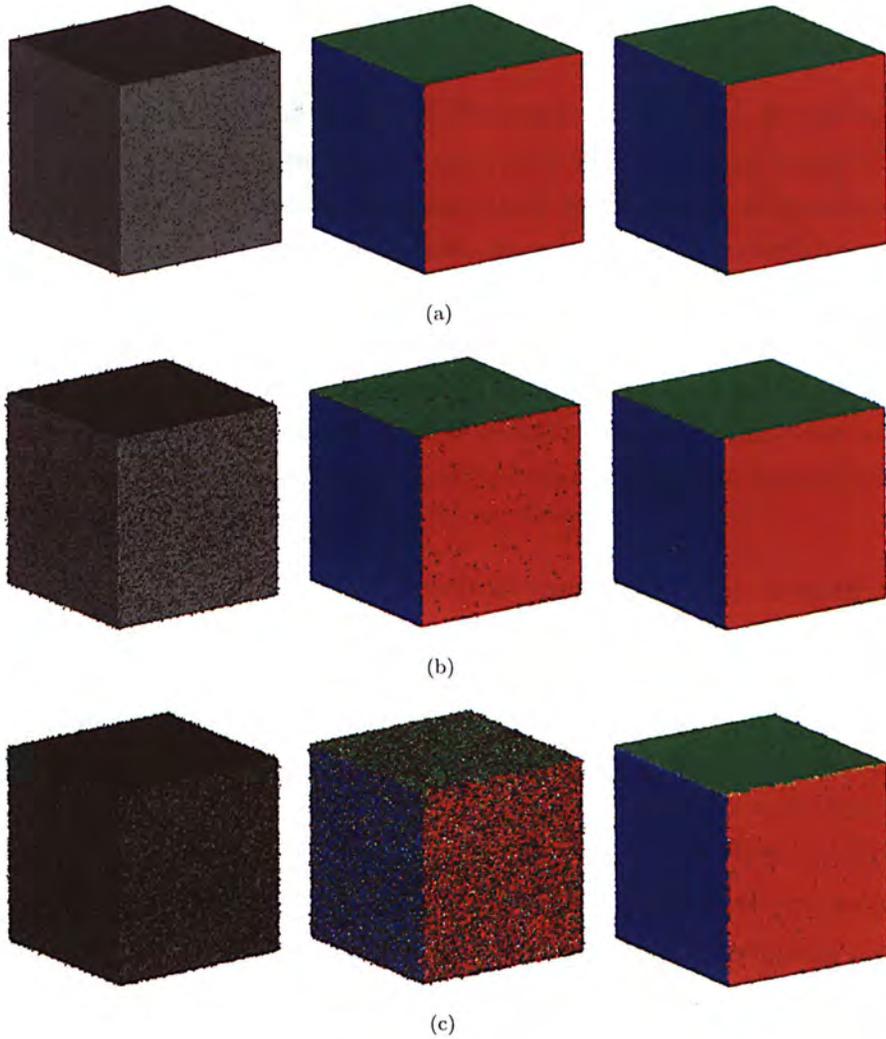


Figure 3.2: Robust estimator. (left column) The point set with 240K samples of a cube model embedding (a) 6% of noises, (b) 25% of noises and (c) 70% of noises. (middle column) RMLS starts to fail at 25% of noises. (right column) Our approach with the points successfully projected even at 70% noises. Points are displayed with color-coded normals.

of outliers. Secondly, using a value of m computed by Eq.(3.5) cannot guarantee to find a good fit among random selections, and it can be much higher as discussed in [TM02]. Therefore, we use a more practical solution in our algorithm. After assigning a fixed number for m (e.g., $m = 300$), we can obtain a relatively clean point cloud with singular normal on very few isolated samples. Taking the cloudy points of the cubes in Fig.3.2 with different percentages of noises as an example, we can successfully project the points onto the cubes while [FCOS05]’s RMLS starts to fail at 25% of noise. The ability to fit surface corresponding to less than 50% of the data points is very important to the correct normal estimation on samples near sharp features where there are multi-structures.

3.4 Moving Least Squares Surfaces

The related work, moving least squares (MLS) surface, defines an implicit surface which approximates or interpolates the given point samples using a projection operator. This method represents the surface by projecting all the points onto the estimated smooth surface. It is different from reconstructing surface which converts the points into another representation of surface, e.g. triangular mesh or polygonal mesh. The major work of projection operator introduced by Levin [Lev03] consists of two basic steps with the first stage of defining a local reference domain and the second stage of fitting a local bivariate polynomial over the reference domain. Many researchers investigated the extension of Levin's projection operator, such as [ARAA04, AA04, FCOS05].

3.4.1 Step 1: local reference domain

Given the point data set $P = \{p_i | i = 1, \dots, N\}$ and let x be the point to be projected, the local reference domain (or plane), $H = \{n \cdot x - d = 0, x \in \mathbb{R}^3\}$ where $n \in \mathbb{R}^3$ and $\|n\| = 1$, is computed by minimizing the weighted sum of squared distances from p_i to H

$$\sum_{i=1}^N (n \cdot p_i - d)^2 \theta(\|p_i - q\|) \quad (3.7)$$

among all normal directions \mathbf{n} , θ is a smooth, positive and monotonically decreasing function and q is the orthogonal projection of x on H . The local reference domain is given by an orthonormal coordinate system on H with q as the origin.

3.4.2 Step 2: local bivariate polynomial

The reference domain of x is then used to find a local bivariate polynomial approximation to fit the points in the neighborhood of x . Let q_i be the orthogonal projection of p_i onto H , and $f_i = \mathbf{n} \cdot (p_i - q)$ be the height of p_i over H . The polynomial approximation g is computed by minimizing the weighted least squares error

$$\sum_{i=1}^N (g(x_i, y_i) - f_i)^2 \theta(\|p_i - q\|) \quad (3.8)$$

where (x_i, y_i) is the local 2D coordinates of q_i in the local reference domain H . The projection of x is then given by the polynomial value at the origin

$$\Psi(x) = q + g(0, 0)\mathbf{n} \quad (3.9)$$

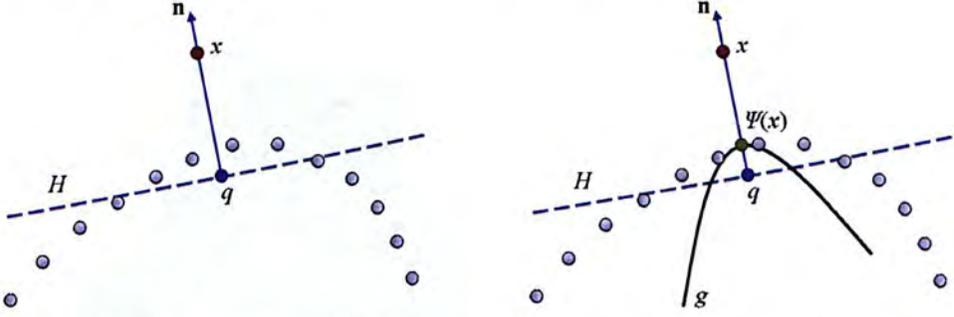


Figure 3.3: The procedure of MLS projection: (left) a local reference domain H is determined, (right) then a local bivariate polynomial g over H is computed. $\Psi(x)$ (green point) is the MLS projection result.

Figure 3.3 illustrates the whole MLS projection steps.

3.4.3 Simpler Implementation

In practice, the MLS surface is computationally expensive due to the non-linear optimization of local reference domain computation. Inspired by MLS surfaces, Alexa and Adamson [ARAA04] presented a much simpler and efficient projection operator which also allows the orthogonal projections by computing the exact surface normals. Their projection iteratively computes the locally weighted average position and projects the point along the normal direction yielding a new position until it converges.

Given a point x , the locally weighted average is defined as

$$a(x) = \frac{\sum_{i=1}^N \theta(\|x - p_i\|) p_i}{\sum_{i=1}^N \theta(\|x - p_i\|)} \quad (3.10)$$

where θ is a weight function which specifies the influence of the point. This weight function is smooth, positive and monotonically decreasing. Typically, a Gaussian-shaped function is used such that

$$\theta(d) = e^{-d^2/h^2} \quad (3.11)$$

where h is the anticipated feature size which is a global scale factor that determines the Gaussian kernel width. Features would be smoothed out if their feature sizes are smaller than h . Choosing a suitable h is difficult for non-uniformly sampled point set. [AA04] computed h as the average Euclidean k -nearest neighborhood distance with $k = 6$, which gives a practically adaptive approximation of local sampling density.

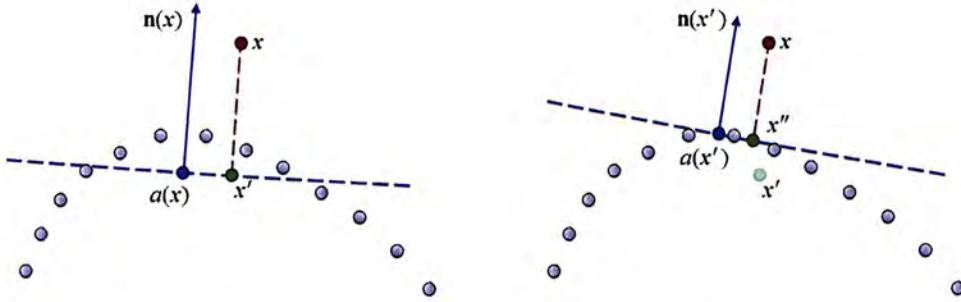


Figure 3.4: The first two projection steps of [ARAA04]. In each step, the current updated position x' is used to compute the orthogonal tangent frame by $a(x')$ and $\mathbf{n}(x')$. Then x is projected onto the local frame to compute the new approximation.

Then, the next updated position x' of x is computed by

$$x' = x - n(x)^T(x - a(x))n(x) \quad (3.12)$$

where $n(x)$ is computed by a weighted least squares fit

$$\mathbf{n}(x) = \arg \min \sum_{i=1}^N \|\mathbf{n}^T(x - p_i)\|^2 \theta(\|x - p_i\|) \quad (3.13)$$

or a weighted averaging of input normals \mathbf{n}_i

$$\mathbf{n}(x) = \frac{\sum_{i=1}^N \theta(\|x - p_i\|) \mathbf{n}_i}{\|\sum_{i=1}^N \theta(\|x - p_i\|) \mathbf{n}_i\|} \quad (3.14)$$

The projection procedure is performed by repetitively computing the updated position x^{i+1} with $a(x^i)$ and $\mathbf{n}(x^i)$. The convergence of the iteration yields a point on the surface. Figure 3.4 gives an illustration of the first two projection procedures of [ARAA04].

3.5 Robust Moving Least Squares by Forward Search

The conventional MLS surface defines a surface that is smooth everywhere, thus it cannot preserve sharp features. The sharp features are however important in reconstructing the mechanical parts such as gears and engines. The presence of noises in the data smoothes out the sharp features, or even distorts the real surface. Fleishman et al. [FCOS05] introduced a robust method, forward search algorithm, to identify the outliers. Starting from a small outlier-free region estimated by an initial robust estimator, one good sample is added iteratively to re-fit the polynomial until the largest residual is greater than a certain threshold. One surface is then classified and the whole process is repeated until the sample set is empty. Finally, a number of surfaces are classified.

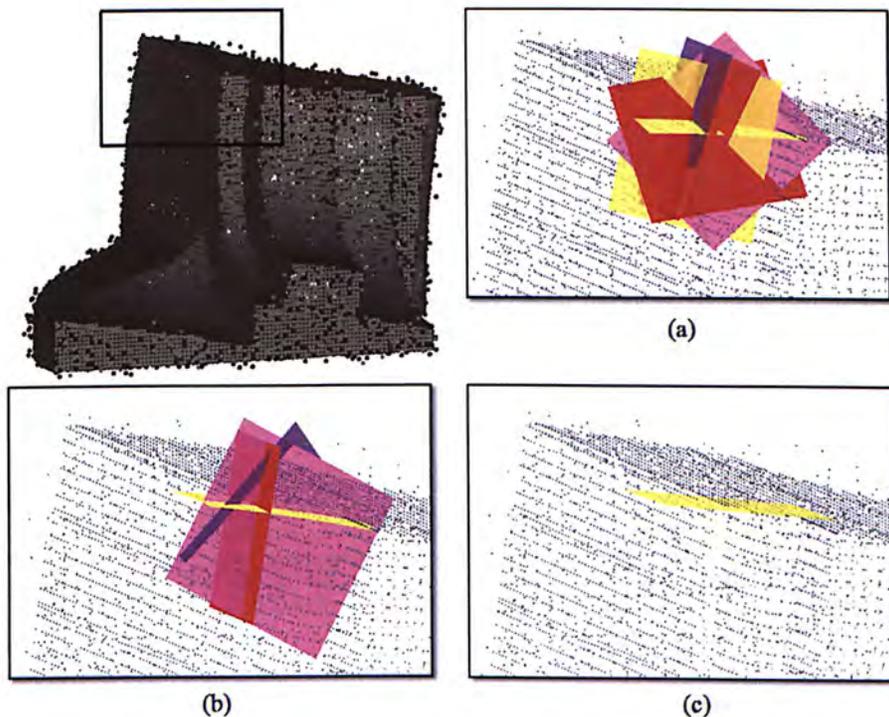


Figure 3.5: Performance comparison of different estimators at a highly noisy corner region: (a) the forward search misclassifies the regions as LMS fails to obtain a good initial fit, (b) the k th order estimator overclassifies the regions, thus a surface region is mistakenly recognized on outliers, and (c) our MDPE based estimator does not have these problems – only the best fitted surface is estimated.

A new projection operator is exploited to define a piecewise smooth surface using this classification.

3.6 Comparison with RMLS

The approach in [FCOS05] requires a robust initial estimator to start the forward search algorithm. It is essential for the initial estimator to fit an outlier-free surface as the forward search is carried out based on this initial guess. They adopted k th ordered-statistics [MS96] to grade the fitted surfaces instead of using least median of squares (LMS). In Fig.3.5, we compare the influence of LMS and k th order to the forward search with the MDPE at a corner of noisy region. We can clearly notice from Fig.3.5(a) that the forward search misclassifies the region as LMS fails to obtain a good initial fit. With k th order statistics, a good initial fit can be obtained but the region is over classified to four surfaces at a corner which actually contains three surfaces only as shown in Fig.3.5(b). This is because one forward search is conducted on outliers. It is however difficult to determine whether the regions classified by forward search belong to outliers or real surfaces. Hence, the point would be projected to a wrong position. In contrast,

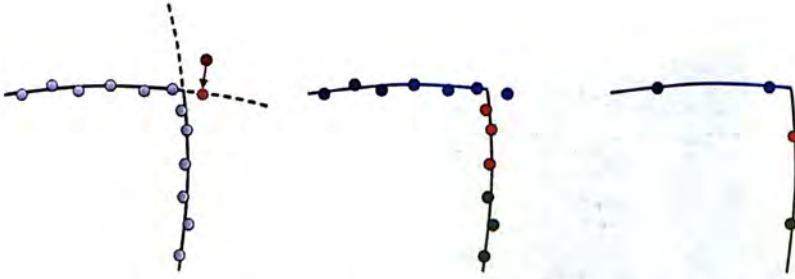


Figure 3.6: A miss-projected point outside the model (left) can be eliminated by the clustering (middle) and the subsequent subsampling (right).

our approach only estimates the best fitted surface among the noisy region with MDPE as demonstrated in Fig.3.5(c). This ensures that the point is projected to the correct surface. Note that if the point is outside of the actual model, there is a chance for the point to be projected on an invalid position. Nevertheless, such points are eliminated in the subsequent subsampling step as illustrated in Fig.3.6.

3.7 K-Nearest Neighborhoods

In local approaches, only the points that are spatially close together (usually in terms of Euclidean distance) are involved in the computation. Hence, a fast and efficient method to compute the k -nearest neighborhoods (k NN) is important when we deal with a massive number of points. Finding k NN can be split into two steps, construction of an efficient data structure to store the points and a fast access method to query the k NN. Conventionally, octree and k d-tree are the most common spatial data structures to decompose the space into different cells. Recently, with the advance of parallel computing technologies (multi-core CPU and many-core GPU), a number of publications presented the parallel implementation of these space partitioning algorithms [SSK07, ZGHG08, ZHWG08] and other techniques [GDB08] for fast computation of k NN.

3.7.1 Octree

Octree is a hierarchical data structure based on recursively decomposition of space. Given a data set, a bounding box is first computed and then recursively subdivided into internal cells which contain data. Each cell can be subdivided into a maximum number of eight nonempty octants and hence such a data structure is called octree. Recently, Zhou et al. [ZGHG08] implemented a parallel construction of Octree data

structure on GPU, which is over two orders of magnitude faster than CPU octree construction algorithm.

3.7.2 Kd-Tree

In general, 3D data points cannot be subdivided evenly into eight octants, which may lead to an unbalanced and suboptimal data structure. Kd-tree however can ensure the generation of a fully balanced binary search tree and the partitioning of the space into a minimal number of subdivisions. Zhou et al. [ZHWG08] presented an algorithm to construct kd-tree on GPU in real-time. In terms of speed, their algorithm performs 4 to 7 times faster than the well-optimized single-core CPU algorithm by [HMS06] and is competitive with the multi-core CPU algorithms by [SSK07]. It also outperforms the kd-tree algorithm in the open source ANN library [MA06] by being 20 times faster.

3.7.3 Other Techniques

Besides the space partitioning techniques, Garcia et al. [GDB08] proposed a “brute force” k NN search which is implemented on GPU by CUDA. Their BF algorithm is actually an exhaustive search. First, all the Euclidean distances between the query point q_i and the data points $P = \{p_i | i = 1, \dots, N\}$ are computed, which are then sorted by insertion sorting algorithm. The k reference points corresponding to the k smallest distances are the result of k NN. By utilizing the highly parallel computing of GPU, their approach is up to 148 times faster than ANN kd-tree [MA06]. However, such an enhancement can only be achieved when the data dimension is very large. Besides, the number of data points is also limited due to the restriction of GPU hardware. In surface reconstruction, we deal with a massive number of data points in 3-dimensional space. The “brute force” algorithm is obviously unsuitable for this application.

In appendix, we introduce a GPU k NN computation based on space partitioning method using CUDA. Different from [ZGHG08, ZHWG08], our approach is simple and easy to be implemented. It can also be run in a streaming mode so that it is able to process a massive number of points.

3.8 Principal Component Analysis

The z -axis (upward direction) of the local coordinate-frame can be estimated through local principal component analysis for the local surface normal. This approach in fact is

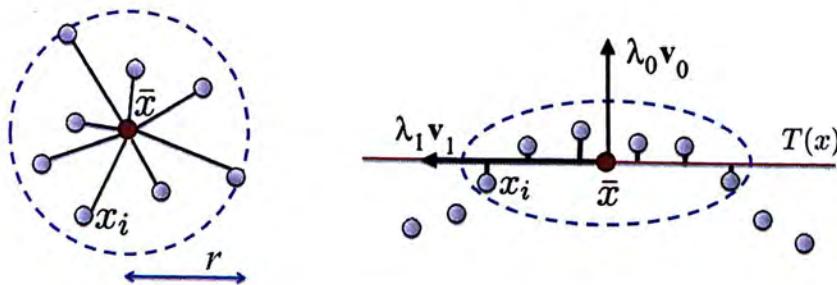


Figure 3.7: Local neighborhood (left) and covariance analysis (right).

an eigenanalysis of the covariance matrix of a local neighborhood [PGK02]. Within k -nearest neighborhoods, the covariance matrix is computed and then the surface normal direction can be deduced from the eigenvector associated to the smallest eigenvalue. The 3×3 covariance matrix \mathbf{C} for a sample point x is given by

$$\mathbf{C} = \begin{bmatrix} x_1 - \bar{x} \\ \dots \\ x_k - \bar{x} \end{bmatrix}^T \cdot \begin{bmatrix} x_1 - \bar{x} \\ \dots \\ x_k - \bar{x} \end{bmatrix}, x_i \in N(x) \quad (3.15)$$

where \bar{x} is the centroid of the neighbors x_i of x (see Figure 3.7). Consider the eigenvector problem

$$\mathbf{C} \cdot \mathbf{v}_j = \lambda_j \cdot \mathbf{v}_j, j \in 0, 1, 2 \quad (3.16)$$

Since \mathbf{C} is symmetric and positive semi-definite, all the eigenvalues λ_j are real-valued and the eigenvectors \mathbf{v}_j form an orthogonal frame corresponding to the principal components of the point set defined by $N(x)$. The λ_j measures the variation of the $x_i \in N(x)$ along the direction of the corresponding eigenvectors. The total variation (sum of squared distances) of x_i from their center of gravity is

$$\sum_{i \in N(x)} |x_i - \bar{x}|^2 = \lambda_0 + \lambda_1 + \lambda_2 \quad (3.17)$$

Assume $\lambda_0 \leq \lambda_1 \leq \lambda_2$, it follows that the plane

$$T(x) : (x - \bar{x}) \cdot \mathbf{v}_0 = 0 \quad (3.18)$$

passing through \bar{x} minimizes the sum of squared distances to the neighbors of x . Thus \mathbf{v}_0 approximates the surface normal at x .

3.9 Polynomial Fitting

Fitting surface S in Eq.(3.6) means solving all the coefficients in the polynomial. In our case, we need to fit the surface with some p points. As stated in [Pet02], the general approach is started by computing the average position \bar{p} of the p points, and these p points will then be mapped into a rotated principal frame, i.e. local coordinate frame, by a transformation. Let p_i be those p points with coordinates expressed in global coordinate frame and p'_i be the mapped points that are expressed in local coordinate frame. Then

$$p'_i = R(p_i - \bar{p}) \quad (3.19)$$

where R is a rotation matrix called the attitude matrix. One useful local principal frame is defined by choosing $R = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)^T$ as follows:

$$\mathbf{r}_3 = \mathbf{n}, \quad \mathbf{r}_1 = \frac{(I - \mathbf{n}\mathbf{n}^T)\mathbf{i}}{\|(I - \mathbf{n}\mathbf{n}^T)\mathbf{i}\|}, \quad \mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \quad (3.20)$$

where \mathbf{i} is along the first axis in the global coordinate frame, I is the identity matrix and \mathbf{n} is the normal vector at \bar{p} computed as mentioned in the previous session. In other words, rotation R aligns p'_i with the projection of p_i onto the tangent plane defined by \mathbf{n} . The coefficients of the rotated principal quadric can be obtained by solving the following over-determined system of linear equations:

$$\begin{bmatrix} s_1^2 & t_1^2 & s_1 t_1 & s_1 & t_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_p^2 & t_p^2 & s_p t_p & s_p & t_p \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} S(s_1, t_1) \\ \vdots \\ S(s_p, t_p) \end{bmatrix}, i_j \in N(x) \quad (3.21)$$

with

$$\begin{aligned} & \begin{bmatrix} s_1^2 & t_1^2 & s_1 t_1 & s_1 & t_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_p^2 & t_p^2 & s_p t_p & s_p & t_p \end{bmatrix}^T \begin{bmatrix} s_1^2 & t_1^2 & s_1 t_1 & s_1 & t_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_p^2 & t_p^2 & s_p t_p & s_p & t_p \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} \\ & = \begin{bmatrix} s_1^2 & t_1^2 & s_1 t_1 & s_1 & t_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_p^2 & t_p^2 & s_p t_p & s_p & t_p \end{bmatrix}^T \begin{bmatrix} S(s_1, t_1) \\ \vdots \\ S(s_p, t_p) \end{bmatrix} \end{aligned} \quad (3.22)$$

Algorithm 2: Normal Estimation and Point Projection

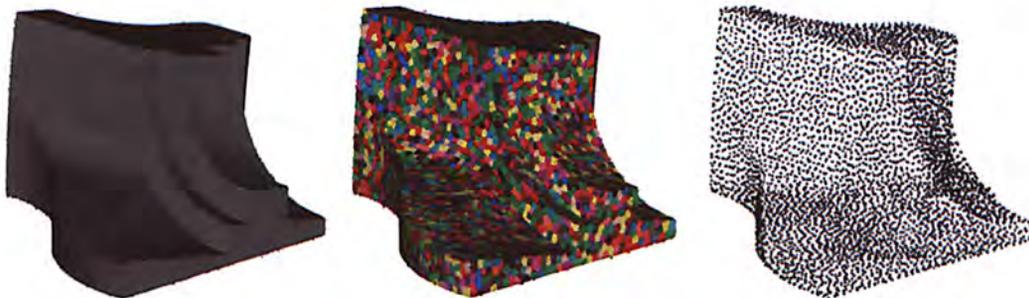
-
- 1: Initialize DP_{\max} of a given sample x by zero;
 - 2: **for** $i = 1$ to m **do**
 - 3: Randomly choose p points to form a p -subset, P_i ;
 - 4: Fit a quadratic surface S to P_i ;
 - 5: Compute the signed residuals for all neighbors of x ;
 - 6: Use the mean shift procedure to determine the center X_c of converged window on the residuals;
 - 7: Evaluate DP_i by Eq.(3.1);
 - 8: **if** $DP_i > DP_{\max}$ **then**
 - 9: $DP_{\max} \leftarrow DP_i$ and $P_{\max} \leftarrow P_i$;
 - 10: **end if**
 - 11: **end for**
 - 12: Fit a quadratic surface S^* to P_{\max} ;
 - 13: Find the closest point x_c on S^* to x ;
 - 14: $x' \leftarrow x_c$;
 - 15: Let the surface normal of S^* at x_c be the normal of x' ;
-

3.10 Highly Parallel Implementation

Similar to all other random sampling techniques, the computation of robust estimators is very costly in time. Running the above algorithm on an advanced PC with 250K points takes about one and a half hour. Different from the techniques employed in [FCOS05, DTB06, JWB*06], the proposed estimation method in this work can be parallelized using the single-instruction-multiple-data (SIMD) parallelism and the architecture that is available on the consumer graphics hardware with the graphics processing unit (GPU). We first store all samples in the given point cloud in the device memory of graphics card. Then, the ANN KD-tree [MA06] is adopted to find k nearest neighborhoods of every sample. Following the suggestion of using $k = 6p$ neighbors gives good results in our tests. Since the KD-tree does not allow multiple access at the same time, we store the query result in a neighborhood information table for later use. Afterwards, records of the table are passed to Algorithm 2 to evaluate the normal vectors and project points in a streaming manner. Because of the texture memory limitation on a graphics card, we process 2048 samples at each pass. The parallelism is easy to be implemented by NVIDIA's CUDA library.

Chapter 4

Error Controlled Subsampling



It is impractical to generate a mesh connecting the cleaned point cloud immediately after the first step. As the data points are projected onto their fitted surface, the total number of points remained is the same as that of the original point cloud which contains massive number of samples. Therefore, we down-sample the point set into user specified number of points to be further triangulated. However, it is unavoidable that reducing the complexity of the point cloud will create some approximation errors. In order to control the quality of subsampling, we developed an energy minimization based method that groups the given massive points into optimal clusters. The shape of points in a cluster is then approximated by a proxy represented by a site point, which is the average position of all points in this cluster.

4.1 Centroidal Voronoi Diagram

Voronoi Diagram is a kind of decomposition of the space into a discrete set of cells. Clustering is performed by applying the Centroidal Voronoi Diagram (CVD) on the surface. CVD is a special Voronoi Diagram where the site point (seed) of each Voronoi cell is located at the centroid position. Given a connected compact region $\Omega \subset \mathbb{R}^a$, let

$S = \{s_i | i = 0, n - 1\}$ be the set of n distinct site points, and the CVD is defined as n Voronoi cells C_i as

$$C_i = \{x \in \Omega | d(x, s_i) < d(x, s_j), \forall j \neq i\} \quad (4.1)$$

where d is the Euclidean distance and the site point is

$$s_i = \frac{\int_{C_i} x \rho(x) dx}{\int_{C_i} \rho(x) dx} \quad (4.2)$$

where $\rho(x)$ is a density function.

4.2 Energy Function

Our clustering is driven by minimizing the discrete energy terms. Our formulation of energy function in clustering is based on two criteria:

- The distribution of clusters should enable their proxies to best approximate the shape of the given model.
- Clusters should maintain a disk-like shape.

To satisfy them, we define two energy terms to score clusters.

4.2.1 Distance Energy

To control the disk-like shape of clusters, we introduce an energy term based on distance according to the site point p_i of a cluster C_i as

$$E_{dist}(x) = \|x - p_i\|^2. \quad (4.3)$$

4.2.2 Shape Prior Energy

Our shape prior energy is

$$E_{shape}(x) = \|(x - p_i) \cdot n_x\|^2 \quad (4.4)$$

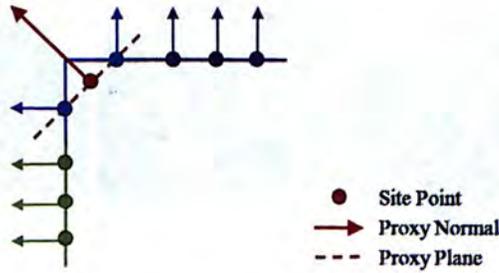


Figure 4.1: Using the shape error measurement obtained by the proxy plane which passes through the site point cannot avoid moving site point away from the surface near sharp features, which however gives large error using our shape prior energy term $E_{shape}(x)$. The samples in different colors belong to different clusters.

Note that, different from [CSAD04, SF08], we employ the normal vector n_x at a sample x but not using the normal vector of a proxy as $E_P^S(x) = \|(x - p_i) \cdot n_{p_i}\|^2$ with n_{p_i} , the average normal of all samples in the proxy. This is because using a metric with the proxy normal does not sensitively reflect the shape-approximation-error when the cluster is crossing a sharp edge. For example, in Figure 4.1, the site point is far from the original surface but shows zero energy by E_P^S . Locating a site point far from the original surface will introduce large shape error on later generated mesh surfaces. Such error can hardly be recovered, and should be prevented. On the contrary, using E_{shape} defined in Eq.(4.4) will make the resultant site points of optimization near to the given point cloud, and thus mesh surfaces with less shape error will be generated.

4.2.3 Global Energy

The energy on a cluster C_i is weighted to control the influence of each criterion and so it is defined as

$$E(C_i) = w_1 \sum_{x \in C_i} E_{dist}(x) + w_2 \sum_{x \in C_i} E_{shape}(x), \quad (4.5)$$

and the global energy of a clustering is defined by adding the energy terms of all clusters together as

$$E_{global} = \sum_i E(C_i). \quad (4.6)$$

Figure 4.2 demonstrates the clustering results contributed with different weight ratios of the energy terms. With $E(C_i)$ totally contributed by E_{dist} , it actually performs uniform clustering, but the clusters are likely to be located across the edges. Besides, if only E_{shape} is taken into account, the clusters will be formed in an arbitrary shape. Hence, to let the clusters better approximate the curvature while maintaining a disk-like



Figure 4.2: Clustering with different weight ratios: (left) $w_1 = 1, w_2 = 0$, where the clusters are formed uniformly; (middle) $w_1 = 0, w_2 = 1$, where the clusters formed are arbitrary in shape in the regions of similar curvatures; and (right) $w_1 = 0.1, w_2 = 0.9$, where the clusters best approximate the surface with a disk-like shape.

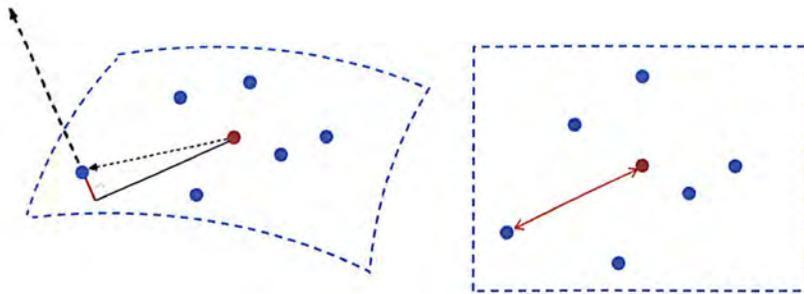


Figure 4.3: Geometric meanings of E_{shape} (left) and E_{dist} (right). Note that the value of E_{shape} is much smaller than that of E_{dist} , so a scale term is necessary to balance them.

shape, we choose $w_1 = 0.1$ and $w_2 = 0.9\varsigma$ with

$$\varsigma = \frac{\text{avg}\{E_{dist}(x)\}}{\text{avg}\{E_{shape}(x)\}} \quad (4.7)$$

defined according to the initial clustering to balance the weight between E_{dist} and E_{shape} in Eq.(4.5) (see the illustration of Fig.4.3).

4.3 Lloyd's Algorithm

Lloyd's algorithm starts from an initial partitioning step which groups the points into n cluster sets. This initial partitioning step is performed randomly for uniform clustering result. However, if adaptive clustering is desired, the random approach slows down the convergence speed as the clusters in low density regions move slowly towards the high density regions. An initial distribution according to density function is suggested by [VCP08] to produce an adaptive initial sampling in order to speed up the convergence. The algorithm is then iteratively performed by the following two steps:

1. Compute the centroid of each cluster as the representative point of cluster.

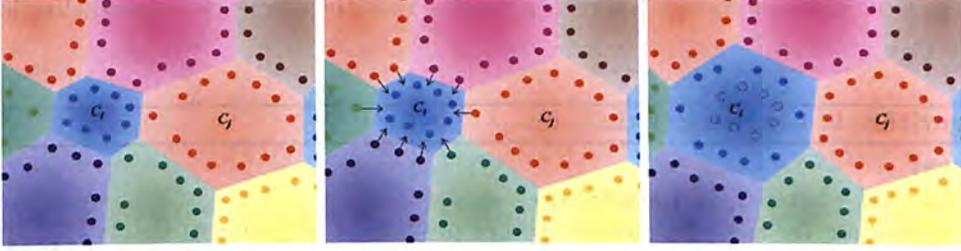


Figure 4.4: Example showing clustering steps: After the random initial seed selection (left), boundary points will be shifted to another cluster if such a change reduces global energy (middle). Within several iterations, the clustering is optimized (right).

2. Form the new partition by assigning each data point to its closest representative point in Euclidean space.

The algorithm stops when the points no longer shift to other clusters, or the centroids of clusters remain unchanged.

4.4 Clustering Optimization and Subsampling

Similar to Lloyd's algorithm, we first perform an initial clustering by partitioning the cleaned point cloud into n , a user specified number, clusters. n points are randomly selected from the point cloud as site points of clusters, and are inserted into a KD-tree. Then, the cluster ID of every rest point is assigned by querying the closest site point through the KD-tree. This initialization step in general can be completed very fast.

Second, the clustering is optimized by minimizing the global energy in Eq.(4.6) with an iterative algorithm. A clustering can be optimized just according to the tests on the boundary points between different clusters. Given a boundary point $x_b \in C_i$ adjacent to another cluster $C_j (i \neq j)$, shifting p from the cluster C_i to the cluster C_j will change the energy locally on $E(C_i)$ and $E(C_j)$. Therefore, the cluster ID of x_b is changed from i to j if such shifting reduces the global energy. Figure 4.4 illustrates the process.

Due to the local manner of clustering update, the optimization step can be parallelized to employ the computing power of multi-core CPU or GPU. The k -nearest neighbors of each sample are pre-computed by the spatial partition techniques (e.g., KD-tree) and stored in a neighboring information table. Choosing $k = 10$ can balance both the quality and the speed. Through this table, we can in parallel detect every sample if it is on the boundary by comparing its cluster ID with the cluster ID of its neighbors. The cluster shifting of a boundary point by comparing the local energy change can also be performed in parallel. We have implemented the Algorithm 3 using CUDA on GPU.

There is no data confliction between threads - thus the efficiency of parallelization is very high.

Algorithm 3: Clustering Optimization

```

repeat
  for each cluster  $C_i$  in parallel do
    | Update the site point to the average position of all points in  $C_i$ ;
    | Find the boundary points in  $C_i$ ;
  end
  for each cluster  $C_i$  in parallel do
    | for each boundary point  $x_b \in C_i$  do
    |   | for neighbors  $x_j \in C_j$  of  $x_b$  do
    |   |   | if moving  $x_b$  to  $C_j$  reduces the energy then
    |   |   |   | Update the cluster ID of  $x_b$ ;
    |   |   | end
    |   | end
    | end
  end
end
until the change of  $E_{global}$  is less than 1% ;

```

Lastly, there are two options to generate the subsampling by the result of clustering optimization:

- Using site points as the down-sampled points;
- Or using the closest point in a cluster to the site point as the down-sampled point in this cluster.

Normals are also assigned to down-sampled points as they are important to the sharp feature reconstruction. There are also two options to generate normals:

- Using the average normal of all points in a cluster;
- Or using the normal at the closest sample point to the site point.

By our experiences, using site points and the normals at their closest points gives better results than other options.

Chapter 5

Mesh Generation



Resultant meshes are generated by first triangulating the down-sampled points into a surface M using the Tight Cocone algorithm [DG03]. We also developed a local triangulation method based on the clustering result. However, the sharp features are chamfered due to the positions of the down-sampled points. To recover sharp features, we generate a dual-graph \tilde{M} of M by converting each vertex in M to a polygon and each triangle $T_i \in M$ into a vertex $v_i \in \tilde{M}$. As every vertex in M generated from the down-sampled point cloud is equipped with a normal vector, we locate vertex $v_i \in \tilde{M}$ by the position which minimizes the Quadratic Energy Function (QEF) defined by the three vertices in T_i and their normals. To be robust, the position is computed using the singular value decomposition (SVD). The resultant polygonal mesh \tilde{M} thus preserves sharp features quite well.

5.1 Tight Cocone Triangulation

Tight Cocone reconstructs water-tight surfaces by computing an initial surface using Voronoi based approach followed by a subsequent marking and peeling step for filling all holes to complete the mesh reconstruction. The initial surface is obtained by the

modified Cocone algorithm in [DGH01] which removes the needless triangles near the undersampled regions that produce holes and other artifacts. Before the peeling process, there is a marking step to identify the poor tetrahedra to be peeled.

5.2 Clustering Based Local Triangulation

Our meshing step can also be performed based on the clustering result to connect site points for triangulation. Similar to [OBS05b], we first reconstruct an initial surface which contains non-manifold parts due to the local triangulation approach. A cleaning process is then performed to clean out those redundant triangles as well as filling the holes generated.

5.2.1 Initial Surface Reconstruction

Every site point s_i forms triangular faces with its neighboring site points $s \in Nbhd(s_i)$. To obtain the connectivity information of site points $S = \{s_i | i = 1, \dots, N_C\}$, we first build a neighboring cluster table which stores all neighboring cluster information. Again, the first element in each row is used to store the number of neighboring cluster. The *ClusterID* of each point is compared with its neighboring points to see if they belong to different clusters. If so, their clusters are adjacent. For instance, if the *ClusterID* of $s_i \neq ClusterID$ of s_j but s_i and s_j are neighbors, then $C_j \in Nbhd(C_i)$. This further implies that $s_j \in Nbhd(s_i)$ and hence should be connected. As only the boundary points are involved in the comparisons, the computational cost is very low.

Similar to [BBA08], the neighboring site points are then projected onto the tangent plane of s and sorted radially according to the angles to reference vector \vec{v}_r . Denote s_j to be the j -th neighboring site point of s and $\vec{s}_j = s_j - s$ be the vector pointing to s_j , then the projected vector \vec{t}_j is computed by

$$\vec{t}_j = \frac{\vec{s}_j - (\vec{s}_j \cdot \vec{n}_s)\vec{n}_s}{\|\vec{s}_j - (\vec{s}_j \cdot \vec{n}_s)\vec{n}_s\|}. \quad (5.1)$$

Taking \vec{t}_0 as the reference vector \vec{v}_r , the angle between \vec{t}_j and \vec{v}_r is

$$\theta_j = \begin{cases} \arccos(\vec{t}_j \cdot \vec{v}_r) & \text{if } (\vec{v}_r \times \vec{n}_s) \cdot \vec{t}_j \geq 0 \\ 2\pi - \arccos(\vec{t}_j \cdot \vec{v}_r) & \text{if } (\vec{v}_r \times \vec{n}_s) \cdot \vec{t}_j < 0 \end{cases}. \quad (5.2)$$

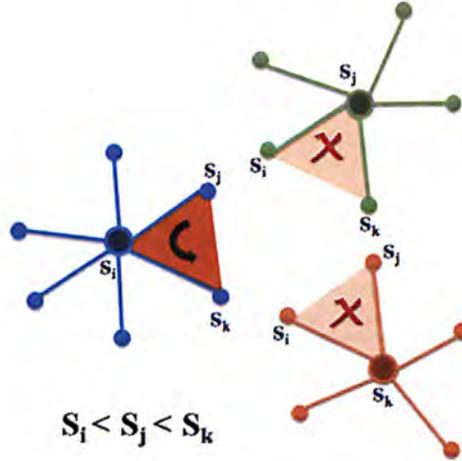


Figure 5.1: An example showing that duplicate triangles are avoided during local triangulation. Triangle is created (left) as the index of s_i is the smallest. In contrast, triangles are discarded (right) for s_j and s_k .

Note that it is unnecessary to compute the actual value of angles for sorting. We can simply multiply the dot values by -1 which gives the same order as that resulted using actual angles, so the computations of arccos are eliminated.

Algorithm 4: Local Triangulation

```

1: for each site point  $s_i$  in parallel do
2:   for each neighboring site point  $s_j$  do
3:     Project points to tangent plane forming  $\vec{t}_j$  by Eq.(5.1);
4:   end for
5:    $\vec{v}_r \leftarrow \vec{t}_o$ ;
6:    $\theta_0 \leftarrow -1$ ;
7:   for  $j = 1$  to  $(neighNum-1)$  do
8:     Compute  $\theta_j$  by Eq.(5.2);
9:   end for
10:  Sort  $s_j$  according to  $\theta_j$ ;
11:  for each pair of consecutive neighboring points  $s_j, s_k$  do
12:    if index of  $s_i$  is the smallest then
13:      Create triangle  $\Delta s_j s_i s_k$ ;
14:    end if
15:  end for
16: end for

```

Triangulation is performed locally for each site point by connecting itself with its neighboring site points as described in Algorithm 4. As the construction of triangles is implemented on GPU in the single-instruction-multiple-data (SIMD) manner, duplicate triangles are likely to be created. For example, if s_i , s_j and s_k are neighboring to each other, $\Delta s_i s_j s_k$ might be constructed three times by the threads running on s_i , s_j and s_k respectively. Therefore, to avoid this kind of redundant triangles, we only connect

the points to form one triangle in the thread of s_i if the index of s_i is the smallest among the three. Figure 5.1 gives an illustration.

5.2.2 Cleaning Process

Similar to all other local triangulation based surface reconstruction approaches, the triangulation itself cannot guarantee to generate closed mesh surfaces with two-manifold topology. To correct the non-manifold conductivities and fill the holes, we use the cleaning procedure as stated in [OBS05b].

Given a mesh vertex v , the curvature of its 1-ring neighborhood D is defined as

$$k(D) = \max \theta_{j,j+1} \quad (5.3)$$

where $\theta_{j,j+1}$ is the dihedral angle between T_j and T_{j+1} which are the triangles of D sorted clockwise or counter-clockwise. Then the curvature measure of v is

$$k(v) = \min_i k(D_i) \quad (5.4)$$

where D_i is all the disk-shaped 1-ring neighborhood of v . The detailed algorithm of cleaning process is given in Algorithm 5. Holes are filled by Liepa's method [Lie03]. Figure 5.2 shows an example after performing the cleaning process.

Algorithm 5: Cleaning

```

1: Sort all  $v$  according to  $k(v)$ ;
2:  $v_0 \leftarrow \min v$ ;
3: Mark  $v_0$  as visited;
4:  $Q \leftarrow push(v_0)$ ;
5: while !isEmpty(Q) do
6:    $v \leftarrow pop(Q)$ ;
7:   Find  $D(v)$ , the minimal curvature disk-shaped 1-ring neighborhood of  $v$ ;
8:   if  $D(v)$  exists then
9:     Remove all the triangles around  $v$  and  $\notin D(v)$ ;
10:    for all boundary vertices  $v_j$  of  $D(v)$  do
11:      if  $v_j$  is not visited then
12:         $Q \leftarrow push(v_j)$ ;
13:        Mark  $v_j$  as visited;
14:      end if
15:    end for
16:  end if
17: end while

```

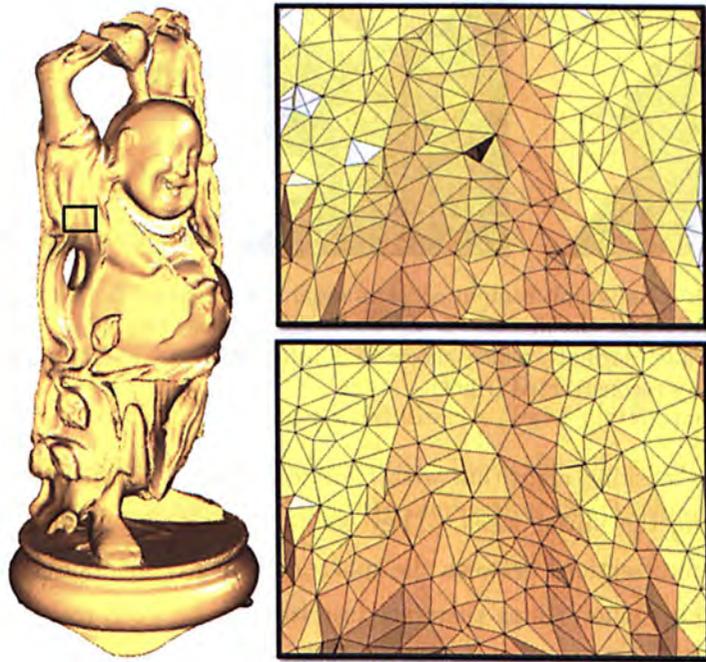


Figure 5.2: An example showing that the non-manifold parts are cleaned and the holes are filled.

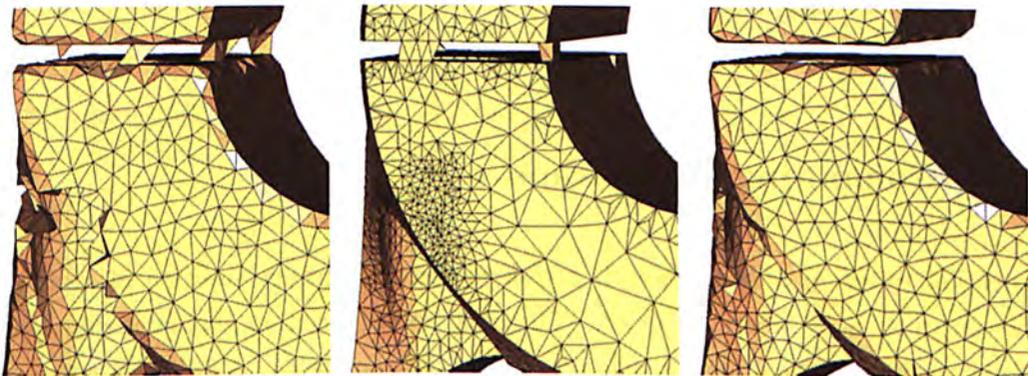


Figure 5.3: Close view of reconstructed surfaces in region of close structures: (left) Tight Cocone algorithm, (middle) Ohtake's method and (right) our clustering based approach.

5.2.3 Comparisons

In our case, we reconstruct the surface from the simplified point cloud which is much sparser than the raw input data. This causes the Tight Cocone algorithm to fail in regions of thin and close structures as demonstrated in Fig. 5.3 (left). Our triangulation method however can handle such cases because it is performed based on the clustering result. Neighborhood information of clusters can be obtained accurately from the data points as they remain dense after projection. This ensures that the triangles will not be created across distinct surfaces in thin and close regions.



Figure 5.4: Dual graph computation: original chamfered mesh (left) can be sharpened by minimizing the distance to all the tangent planes (middle). Note that there are two parallel normals, which shows that using SVD can handle such cases with sharp features reconstructed (right).

Comparing our meshing algorithm with Ohtake's, the main difference is in the stage of reconstructing the initial mesh. They create triangles by generating spheres and inspecting the intersections between the spheres. Wrong connections are likely to happen near the thin structures as there are intersections between the spheres formed in separate surfaces as shown in Fig. 5.3 (middle). In contrast, we create triangles by examining the neighborhood information of clusters, which leads to a much better approximation of the surfaces. Figure 5.3 (right) shows an example of triangulation result using our clustering based approach. More detailed results are discussed in the next chapter.

5.3 Computing Dual Graph

Locating vertex $v_i \in \tilde{M}$ actually is a task to find the optimal position in collapsing the edges $e_i \in T_i$ where $T_i \in M$. The ideal vertex position should minimize the deviation from all the tangent planes corresponding to each $v_i \in T_i$. In other words, such position should have a minimum distance to the tangent planes (see figure 5.4). It is more practical to compute the solution in a least squares sense as the tangent planes approximated may not intersect each other at certain single point.

Consider one vertex $v_i \in T_i$, the quadratic distance of an arbitrary point x from the supporting plane of that vertex is defined as

$$(n_i^T x - d_i)^2 \quad (5.5)$$

where n_i is the normal vector of v_i and

$$d_i = (v_i - \bar{v}) \cdot n_i \quad (5.6)$$

where \bar{v} is the average of $v_i \in T_i$. Thus, the objective function for minimization is given by

$$E(x) = \sum_i (n_i^T x - d_i)^2 \quad (5.7)$$

The iso-contours of this error function are ellipsoids and hence this function is called QEF. The minimized position is given by the solution of

$$\left(\sum_i n_i n_i^T\right)x = \left(\sum_i n_i d_i\right) \quad (5.8)$$

The above system of equations can be solved directly if the matrix has full rank, i.e. the normal vectors n_i are not parallel. In order to make the solution robust and avoid handling of special cases, a Singular Value Decomposition (SVD) should be used here as illustrated in Figure 5.4 (middle).

Chapter 6

Results and Discussion

6.1 Results of Mesh Reconstruction from Noisy Point Cloud

We have implemented the pipeline of mesh reconstruction from unoriented noisy points into a prototype system by C++. NVIDIA's CUDA library is utilized to implement the parallel algorithms on GPU. Our test platform is a PC with two Intel Xeon 2.8GHz quad-core CPU and 4GB main memory. The PC also has a graphics card with NVIDIA GeForce 9800 GTX GPU and 512MB memory, and runs Windows Vista operating system.

To test the robustness of our approach in handling the noises and sharp features, we applied Gaussian noise to the original point cloud by shifting the points in a range of 0.5% of the bounding box's diagonal. The Gaussian noise is randomly distributed with the specified percentage of amount. In addition, we compare our results with RMLS [FCOS05] by embedding RMLS into our mesh reconstruction pipeline for the estimation of normals and projection of points. The first example we tested is the Octa-flower with 18% of noises as shown in Fig.6.1. The thin and sharp features have been successfully reconstructed by our approach but damaged by RMLS. A mechanical model Gear which contains a huge number of points is distorted with 11% of noises and the reconstructed surfaces using RMLS and our approach are given in Fig.6.3. A much more complicated mechanical model Hub is distorted with a large number of noises and the results are demonstrated in Fig.6.5. In order to test the performance of our method on sculpture objects, we choose the Dragon model in Fig.6.7 with 18% of noises. The computational statistics of all these examples are listed in Table 6.1. Benefited from the parallelization of our algorithms, the speed of our surface reconstruction outperforms other methods using robust estimators. Model of half a million points only needs about five minutes to process. Another interesting study is to measure the shape error of

reconstructed models to original models. We adopt the publicly available Metro tool [CRS98] to measure the Hausdorff distances between the reconstructed meshes and the original meshes. The distance errors E_{mean} and E_{max} are given with respect to the diagonal lengths of bounding boxes. To visualize these errors, we use another mesh comparison tool, PolyMeCo [SMS05], to illustrate the geometric differences between the reconstructed meshes and the original ones. Figures 6.2, 6.3 and 6.5 show the comparison results of the corresponding models using a common color scale. It can be concluded from the statistics in Table 6.1 and the visualizations that our method generates very small errors on reconstructed models.

In Fig.6.8, we further compare the performance of normal estimation and point projection between our approach and RMLS on an example of edge. When the noise level increases, our approach can still project the points onto the surface and estimate the normals correctly. On the contrary, RMLS starts to produce errors at 10% noise and totally fails at 60% noise as demonstrated by the points that are color-coded with normals in Fig.6.8(b). In short, our approach is much more robust with the use of MDPE.

For real raw data, noises usually are not present everywhere. It is therefore impractical to perform the robust estimation for all points. It would waste a lot of time as the results of our projection and conventional Levin's projection [Lev03] are nearly the same for inliers. Hence, we can first check the region around the point as described in [FCOS05]. If it is identified as smooth, we simply project the point by Levin's projection. Otherwise, the point is projected using our proposed approach.

One of the difficulties in dealing with the real scanned data is the presence of structural noises which exist in a larger number and are distributed uniformly as shown in Fig.6.9(a). To deal with this, we increase the neighborhood size and process iteratively so that the signal-to-noise ratio is large enough for our robust estimation. Figure 6.9(c) shows the process dealing with structural noise in three iterations.

Lastly, to check whether the clustering based subsampling algorithm converges, we draw the bar chart of shape approximation error energy, E_{global} , and study its variation during the optimization. From Fig.6.10, we can easily observe that the energy, E_{global} , drops very fast using our local update based clustering algorithm. Usually, the iteration stops within ten steps.

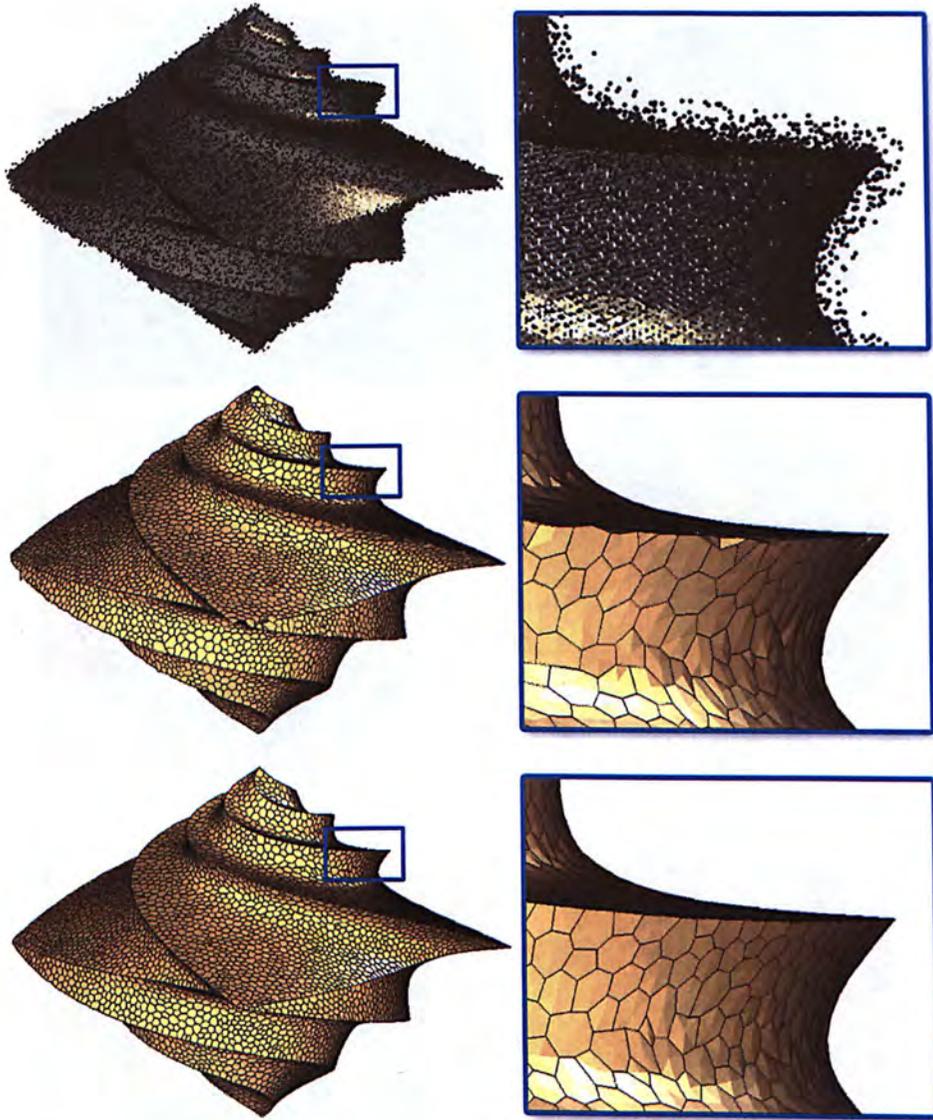


Figure 6.1: Reconstruction of the Octa-flower model: (top) input points with 18% of noises, (middle) reconstruction using RMLS for normal estimation and point projection, and (bottom) our approach can successfully reconstruct the sharp features that are damaged when using RMLS to estimate normals and project points.

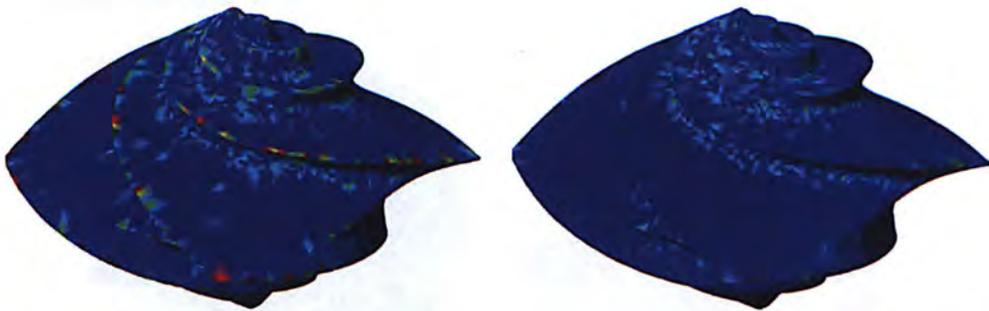


Figure 6.2: Geometric deviation of Octa-flower: (left) RMLS, and (right) our approach.

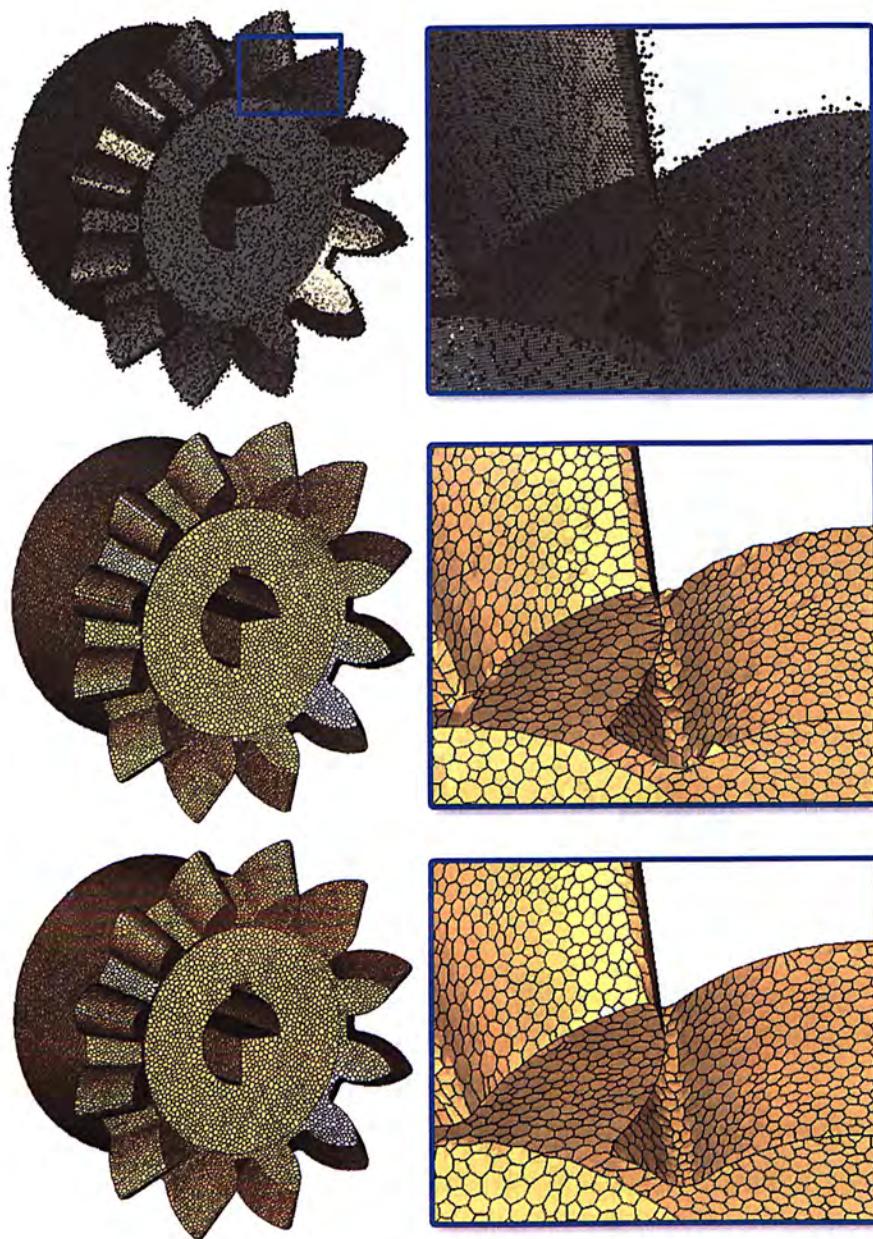


Figure 6.3: Reconstruction of the Gear model: (top) input points with 11% of noises, (middle) reconstruction using RMLS for normal estimation and point projection, and (bottom) our approach can successfully reconstruct the surface with sharp features from large amount of points.

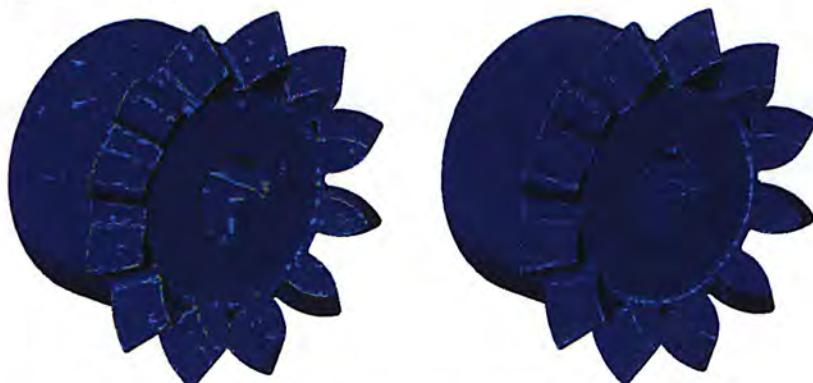


Figure 6.4: Geometric deviation of Gear: (left) RMLS, and (right) our approach.

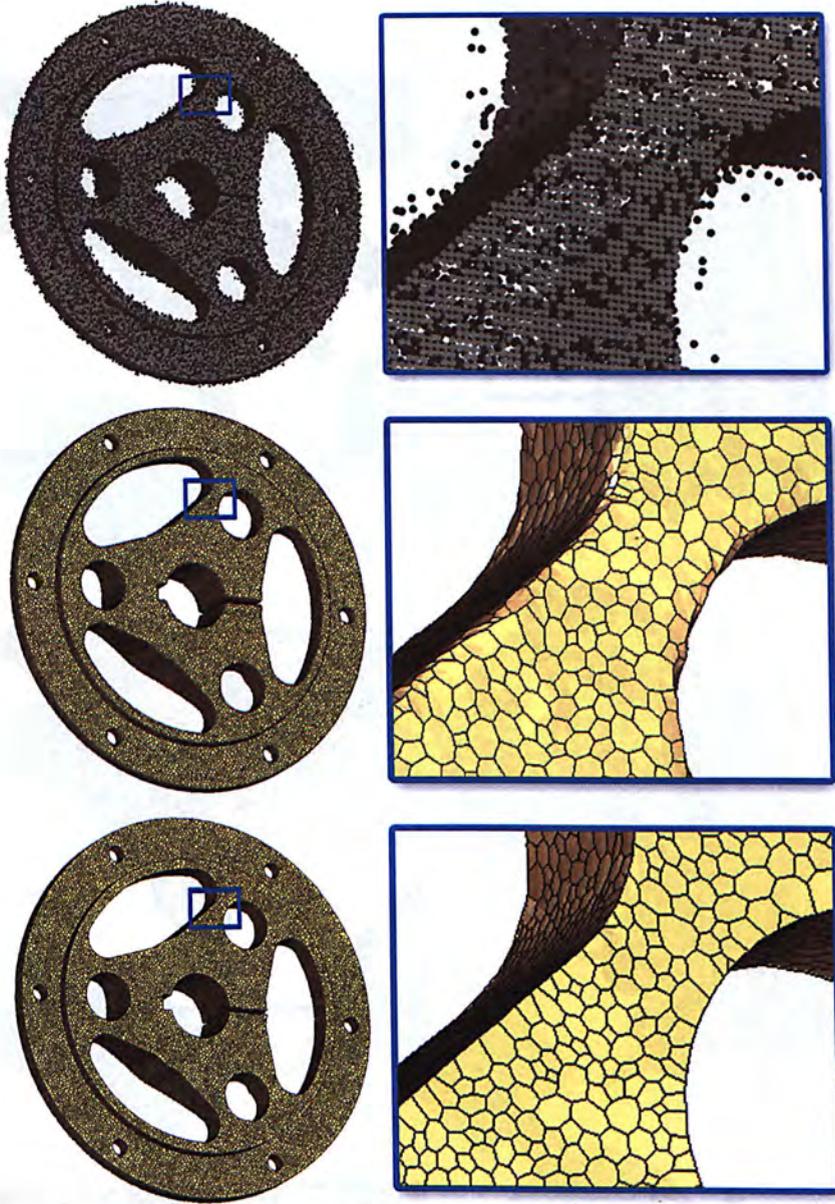


Figure 6.5: Reconstruction of the Hub model: (top) input points with 21% of noises, (middle) reconstruction using RMLS for normal estimation and point projection, and (bottom) our approach can reconstruct the mesh surface from the input with high amount of noises.

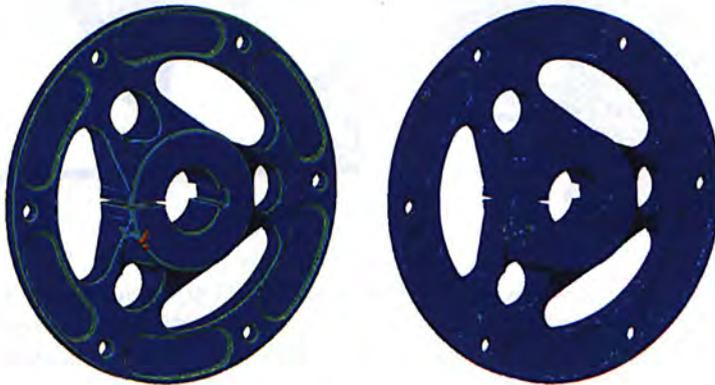


Figure 6.6: Geometric deviation of Hub: (left) RMLS, and (right) our approach.

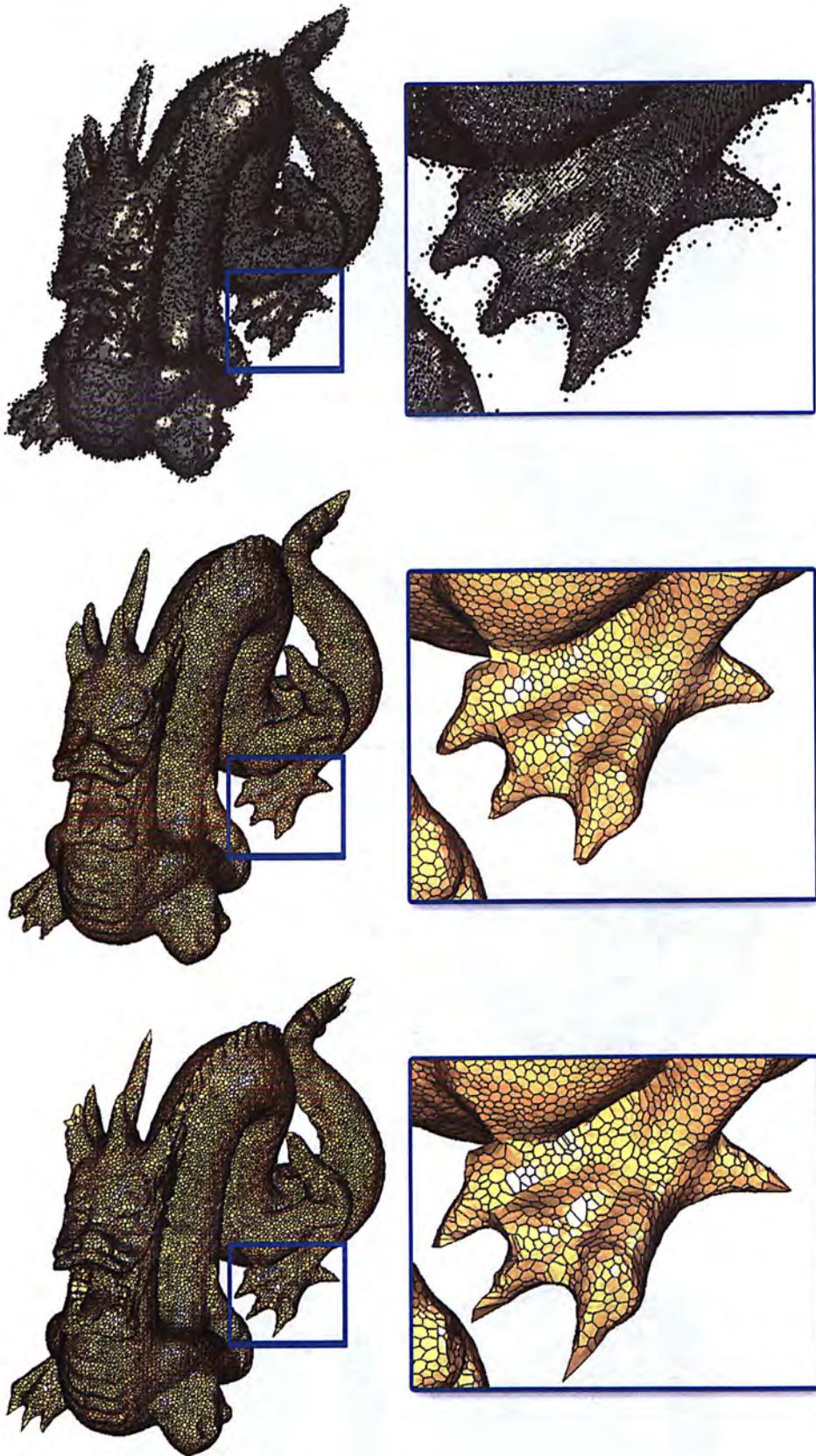


Figure 6.7: Reconstruction of the Dragon model: (top) input points with 18% of noises, (middle) reconstruction using RMLS for normal estimation and point projection, and (bottom) our approach can successfully reconstruct the sculpture model.

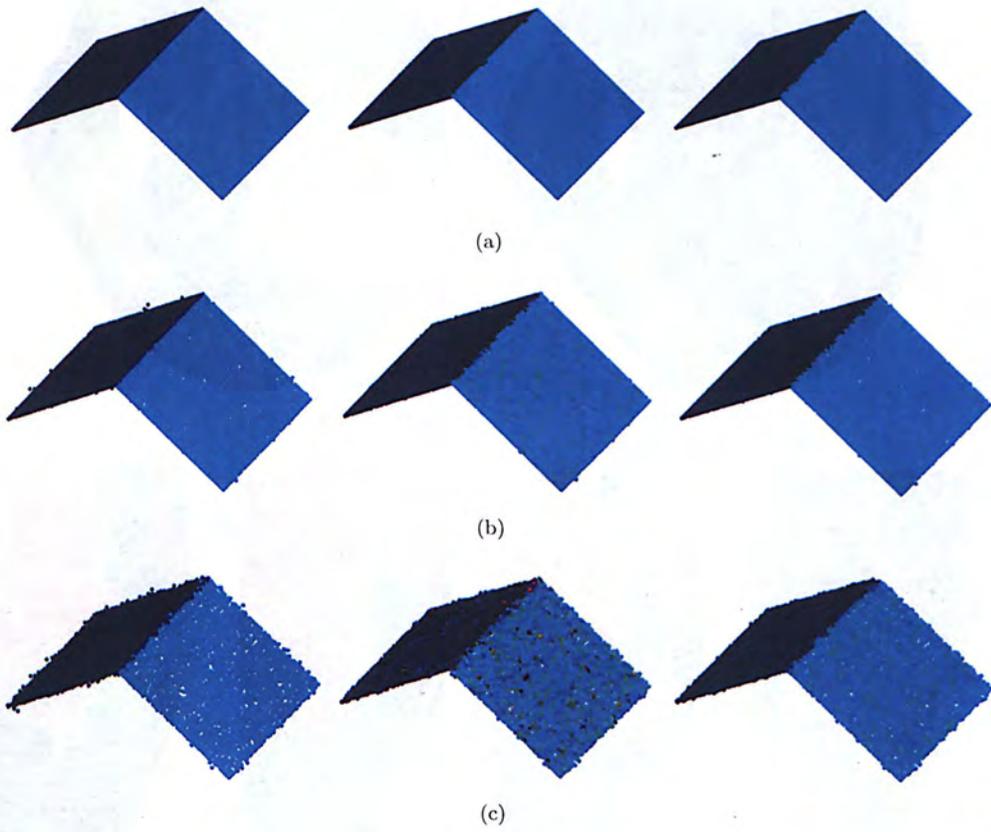


Figure 6.8: Projection test of RMLS versus our approach on an edge: (left column) original points, (middle column) RMLS result and (right column) our approach by MDPE. (a) Clean data. (b) Distorted with 10% Gaussian noises in range of 1% of diagonal. (c) Distorted with 60% Gaussian noises in range of 1% of diagonal.

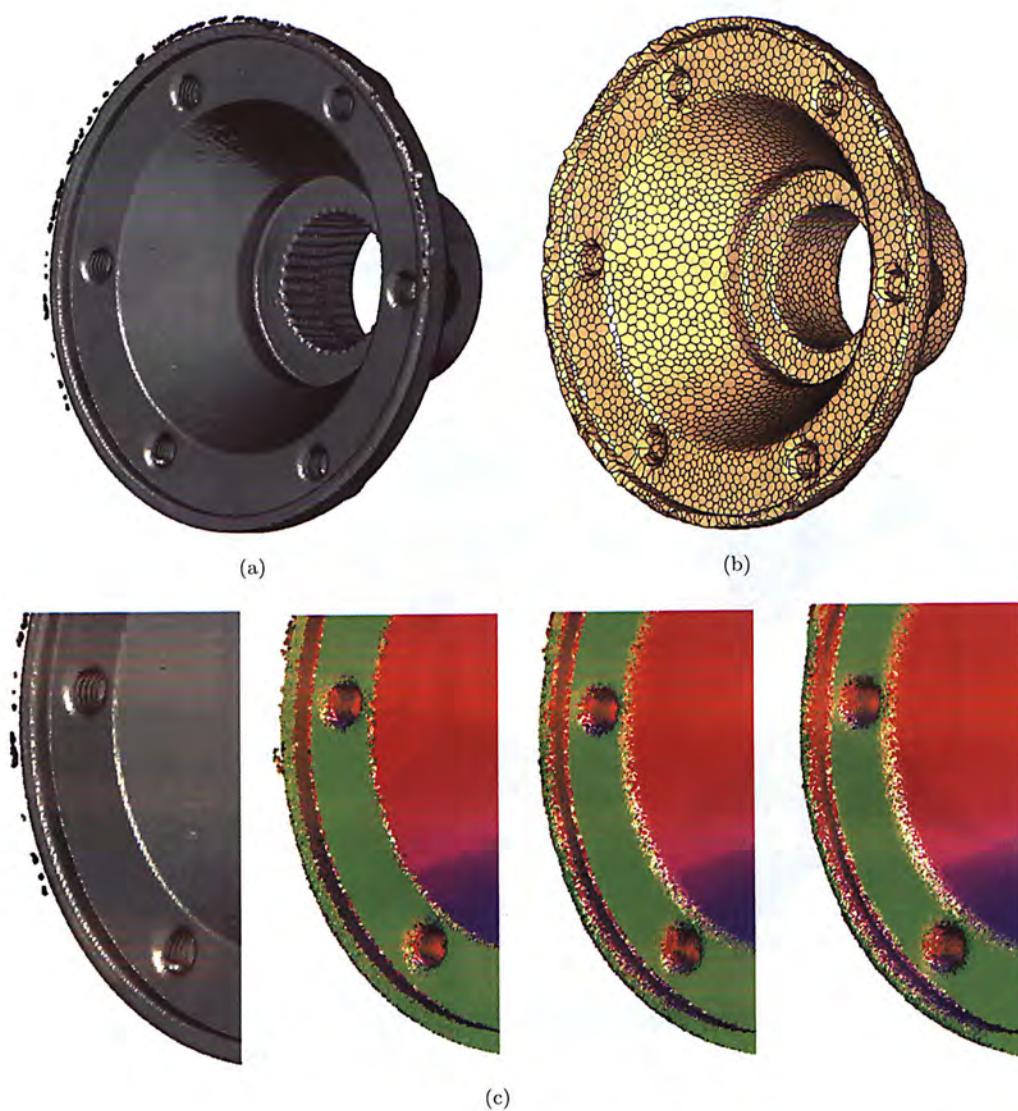


Figure 6.9: The mesh reconstruction of real scanned data *Carter*: (a) raw input with 546K points, (b) resultant mesh generated from 10K down-sampled points, and (c) the robust estimation step is performed three times in order to project the structural noise.

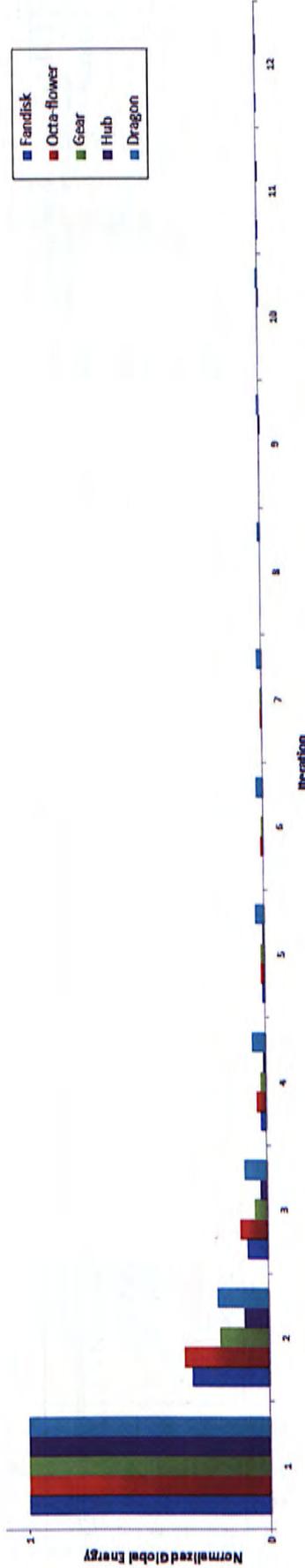


Figure 6.10: The change of energy E_{global} during clustering optimizations. The energy is normalized for display.

Model	Noise Level	Given	Down-sampled	Robust Estimation	Subsampling	Mesh Generation	E_{mean}	E_{max}
Fandisk	18%	550K	10K	4.7 min	4.6 sec	5.4 sec	1.8×10^{-5}	2.3×10^{-3}
Octa-flower	18%	556K	10K	5.0 min	5.2 sec	5.5 sec	5.5×10^{-5}	3.8×10^{-3}
Gear	11%	884K	30K	8.2 min	7.0 sec	18.5 sec	5.2×10^{-5}	1.6×10^{-3}
Hub	21%	474K	50K	4.0 min	4.3 sec	35.6 sec	2.9×10^{-5}	2.5×10^{-2}
Dragon	18%	550K	50K	4.4 min	5.0 sec	34.0 sec	1.2×10^{-4}	2.6×10^{-2}

Table 6.1: Computational Statistics

6.2 Results of Clustering Based Local Triangulation

Besides, the proposed algorithm of clustering based local triangulation is implemented and tested on a PC with Intel Core2 Quad 2.4GHz CPU and a NVIDIA GeForce GTX 295 1792MB graphic card. We used NVIDIA CUDA to implement the algorithms on GPU which is a highly parallel processing unit that can do works simultaneously in many threads. The number of thread blocks and threads per block are needed to be specified for device functions. We use 192 threads per block as suggested in [NVI08] which also balances the speed and device memory usage. The number of thread blocks is then computed by dividing the number of single-instruction by the thread number per block (e.g. $N_C/192$ for the clustering algorithm). Note that if it cannot be fully divided, the number of thread blocks should be added by one. Furthermore, we set $k = 8$ for the k NN which is sufficient for the whole processes.

We performed tests on various types of massive point data. Figure 6.11 shows the meshing of a dragon consisting of 549K points with different user input N_C which is the only parameter required to be specified. The example given in Fig.6.12 is used to demonstrate that our approach can deal with thin and close structures very well. Figure 6.13 demonstrates two more examples of surface reconstruction.

Table 6.2 lists the computational time for all the data sets tested. Note that only the building of *Cellbuffer* is performed on CPU and all the other steps are processed on GPU in parallel. Hence, the speed of our method outperforms some conventional methods. We compared the computational time of our approach with Ohtake's approach [OBS05b], which claimed to be faster than the MPU method [OBA*03], the RBF-based approach of [CBC*01] and the greedy Delaunay-based surface approach of [CSF02]. The comparison of time results on various models shown in Fig.6.14 indicates that our approach is around 3-4 times faster. As Ohtake's method cannot precisely control the vertex number of the output mesh, we adjusted its parameter T_{err} to give a close quality of output for correct comparison – Table 6.3 lists details of the parameters used in the comparison. The last column of Table 6.2 is the geometry approximation errors E_{mean} measured by the publicly available Metro tool [CRS98]. E_{mean} gives the mean distance with reference to the bounding box diagonal between the original meshes and the reconstructed meshes, which concludes that our approach generates very small errors on the reconstructed meshes.

In order to check whether the clustering algorithm converges, we draw the curve of the shape approximation energy E_{global} and study its variation during the optimization in Fig.6.15. From it, we can easily observe that E_{global} drops rapidly in the first few

iterations and then remains nearly unchanged. The iteration of our local update based clustering algorithm usually stops within ten steps.

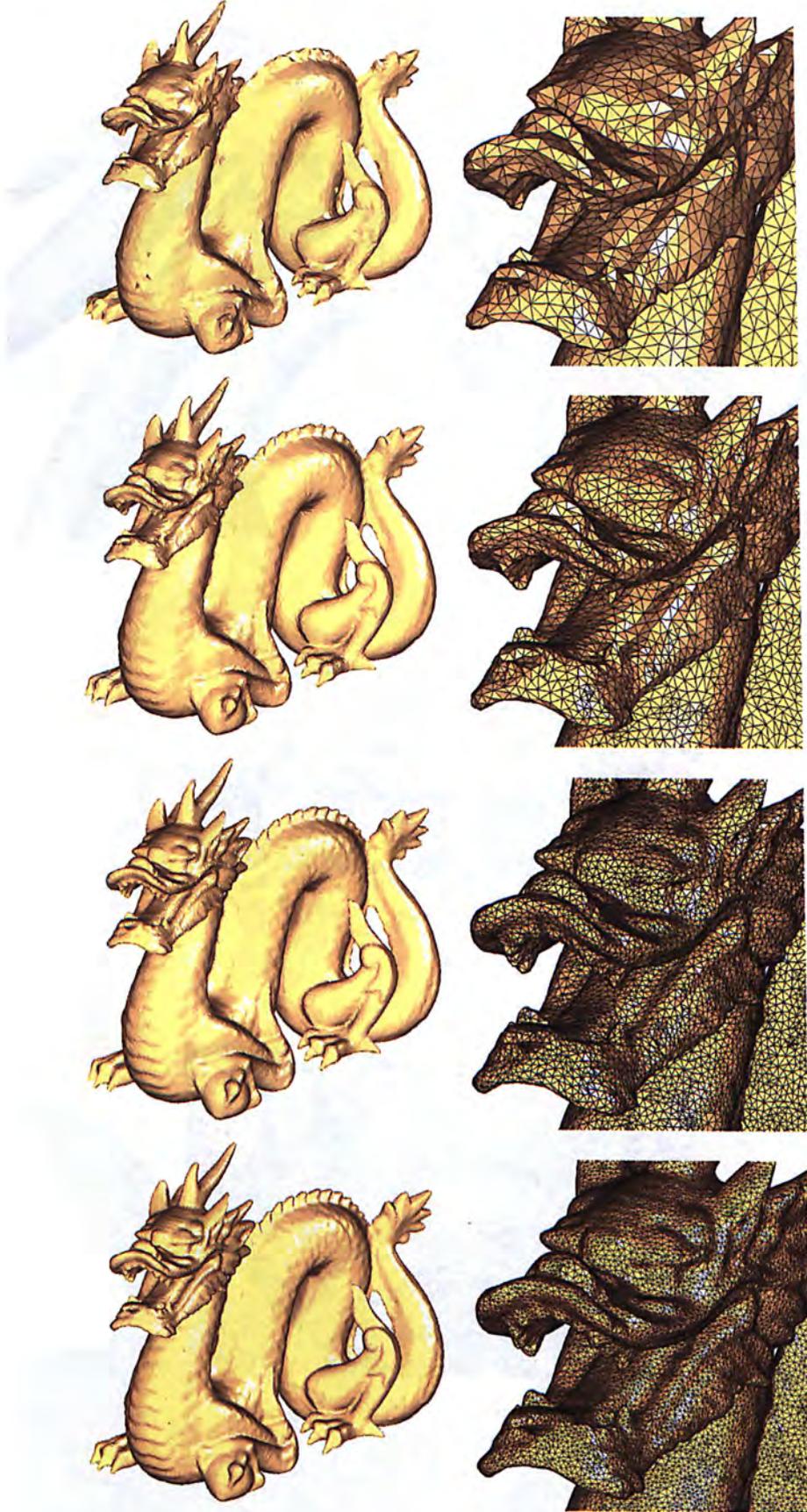


Figure 6.11: Users can directly control the desired number of vertex in resultant mesh by specifying parameter N_C (from top to bottom: 10K, 20K, 40K, 80K respectively). Desirous level of details can be obtained easily and intuitively.

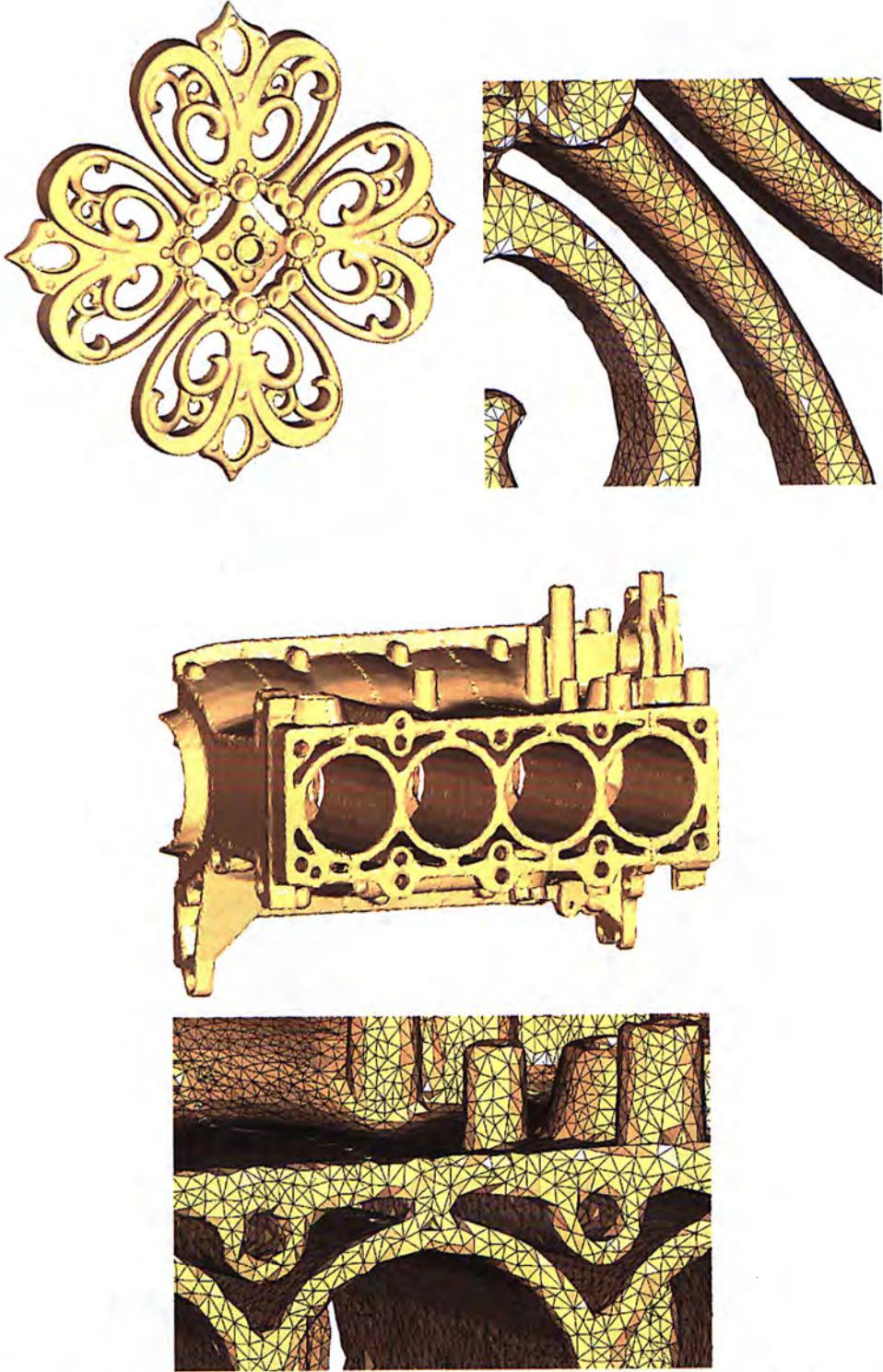


Figure 6.12: Successful reconstruction of *Filigree* and a mechanical model *Engine* which contain many thin and close structures.

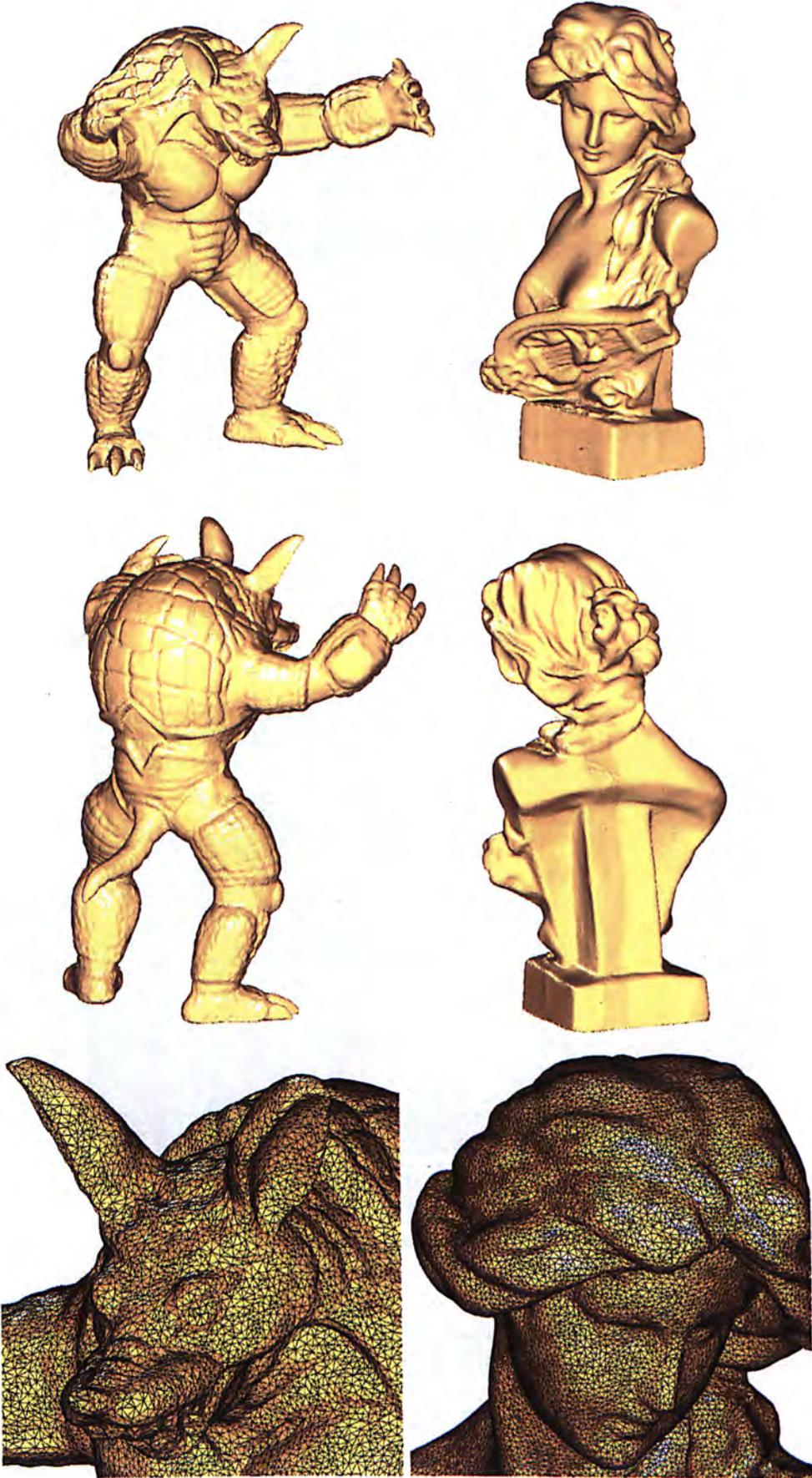


Figure 6.13: Two more examples reconstructed from scattered point data: (left) *Armalido* and (right) *Statue*.

Scattered Data	Point Number	Cluster Number	Preprocessing	Clustering	Meshing	Cleaning	Total Reconstruction Time (*)	E_{mean}
Arnaldo	173K	60K	1.076s	0.827s	0.013s	1.029s	2.945s (2.962s)	1.056×10^{-4}
Statue	428K	70K	1.763s	1.404s	0.016s	1.263s	4.446s (4.523s)	8.178×10^{-5}
Filigree	437K	50K	1.685s	1.435s	0.012s	0.827s	3.959 (4.071s)	1.717×10^{-4}
Buddha	434K	80K	1.669s	1.591s	0.018s	1.669s	4.947s (5.07s)	1.488×10^{-4}
Engine	490K	100K	1.014s	1.888s	0.025s	3.573s	6.5s (6.616s)	3.479×10^{-4}
Dragon	549K	10K	1.904s	1.264s	0.002s	0.156s	3.326s (3.448s)	5.855×10^{-4}
	549K	20K	1.888s	1.404s	0.004s	0.39s	3.686s (3.806s)	3.373×10^{-4}
	549K	40K	1.826s	1.545s	0.009s	0.608s	3.988s (4.118s)	1.932×10^{-4}
	549K	80K	1.732s	1.872s	0.019s	1.388s	5.011s (5.24s)	1.121×10^{-4}
Lion	734K	50K	1.981s	2.131s	0.013s	0.796s	4.921s (5.102s)	3.196×10^{-4}

Table 6.2: Computational statistics of processing time in each step, and the geometry approximation errors measured by Metro. *The time shown in bracket is the total reconstruction time including memory managements (allocation, deallocation, memory transfer between CPU and GPU).

Scattered Data	Output Vertex #	T_{err}	Total Time
Armalido	59276	1.5×10^{-6}	10.683s
Statue	69435	7.5×10^{-7}	14.287s
Filigree	47693	1.4×10^{-6}	13.735s
Buddha	82335	1.0×10^{-6}	18.469s
Engine	92320	4.3×10^{-6}	30.171s
Dragon	79025	7.5×10^{-7}	18.599s
Lion	47654	2.0×10^{-6}	17.876s

Table 6.3: Details of testing Ohtake's program in [OBS05b].

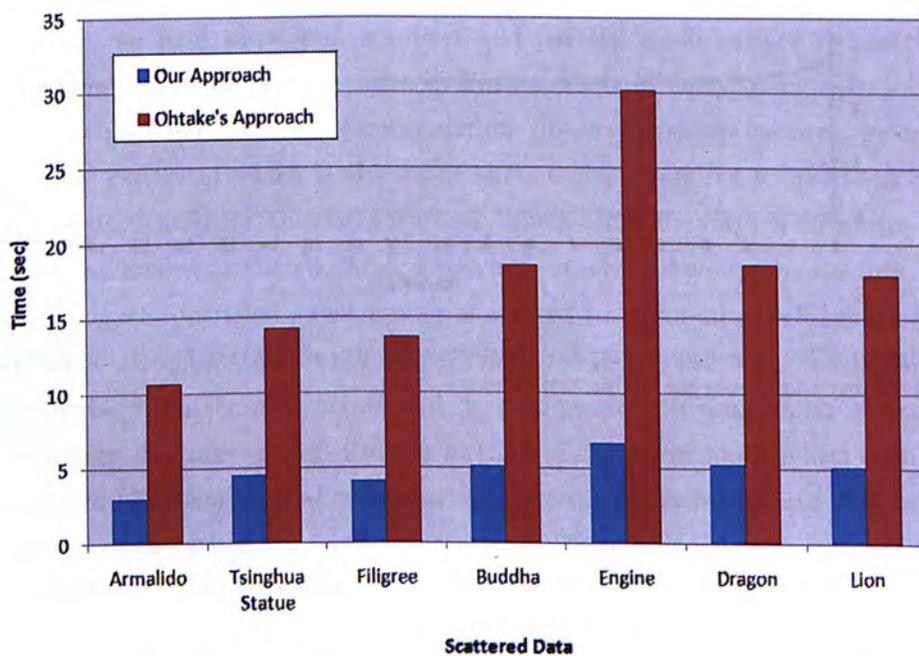


Figure 6.14: Comparison of processing time of our approach with Ohtake's [OBS05b]. Note that the processing time of our approach mainly depends on the number of input data points, while Ohtake's approach depends on the vertex number of output mesh.

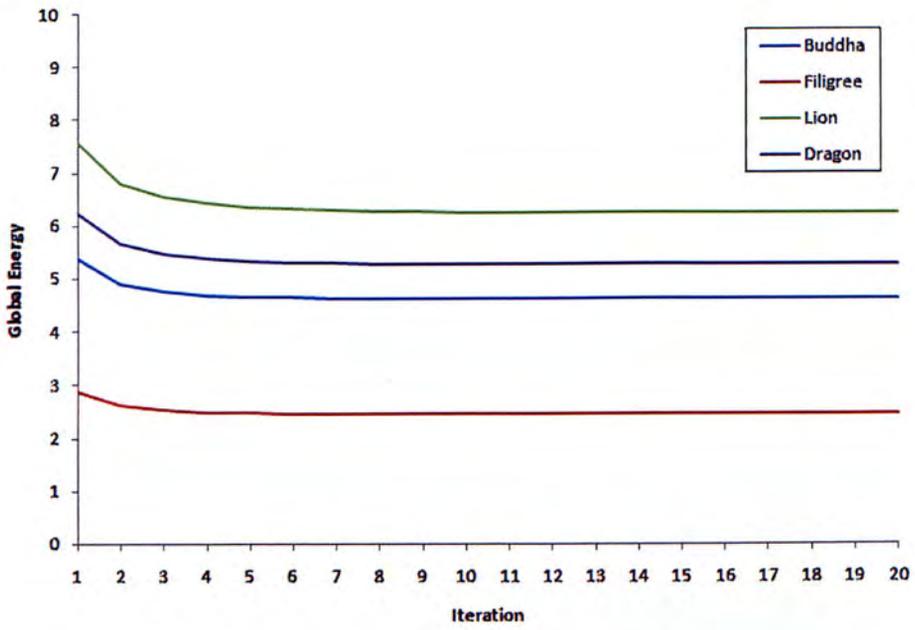


Figure 6.15: The energy graph of clustering. Energy drops very fast in the first few iterations and converges within ten steps.

Chapter 7

Conclusions

This chapter summarizes the key contributions and the limitations of the current implementation. The possible future research directions are pointed out at the end.

7.1 Key Contributions

In this work, we have developed a robust and parallel mesh surface reconstruction technique that reconstructs mesh surfaces from unoriented noisy points. To solve the most difficult problem in surface reconstruction, preserving sharp features, we adopt a highly robust estimator which is able to fit surface corresponding to less than 50% of the data points. Therefore, the sharp features, which are in the form of data with multi-structures, can be reconstructed. After removing outliers and equipping the inliers with normals, an error-controlled subsampling is applied to result in a well-sampled point cloud. Lastly, a combinatorial approach is employed to reconstruct a triangular mesh connecting the down-sampled points and its dual-graph polygonal mesh is computed to recover sharp features. Parallelization method of our algorithm has also been given. The approach has been applied to reconstruct several piecewise-smooth surfaces from noisy point clouds with sharp features preserved. This demonstrates the functionality of our robust mesh reconstruction pipeline.

7.2 Factors Affecting Our Algorithm

The main factor affecting our algorithm is the point density. For example if the point density near the round edges is not high enough, in other words, if the point distances are larger than the radius of round edges, false sharp edges would be introduced. As

we recover the sharp features by locating optimized vertices of dual graph, which are estimated by minimizing QEF, such estimated vertices may not be actually located on the real surfaces. If the downsampling ratio is set too high in our second step, this kind of errors will be more obvious. For instance, as shown in Fig. dragon, sharp claws which are not present in the original model are generated under our approach. This shows that our method does not perform well in handling semi-sharp features namely radius fillets or smooth blends between neighboring surfaces, which appear particularly in the free-form objects that contain many round features. However, dealing with the free-form objects which are smooth everywhere is a much simpler problem as many works have been proposed in literature that performed well. In our approach, we can actually skip the dual graph generation step to give a smooth result, but our goal is to reconstruct the surface with sharp features preserved, which indeed is much more difficult especially with the presence of noise. Our approach can be extended to overcome this situation by performing adaptive clustering in the downsampling step. Smaller clusters are formed near the edges or corners while larger clusters are formed in region with small curvature. This can increase the point density near the sharp features so that the error of the optimized vertices of dual graph is minimized. Hence, our approach can be worked in any kinds of models, even the one contains both sharp features and round features.

Another factor influencing our algorithm is the system parameters. There are two major parameters to be set in our method. The first one is the repetition count m for the robust estimation. The larger the number of iterations is, the better the estimation result will be. However, the computational time will be increased as well. The second one is the number of clusters in the downsampling step. It directly controls the number of points in the downsampled point cloud. More detailed features can be reconstructed if larger number of clusters is set. Similarly, longer time is required for the whole process to be performed.

7.3 Future Work

As discussed in the previous chapter, the major limitation of our approach is our inability to deal with structural noises well. As our approach is performed in a local manner, it is hard to distinguish such kind of noise from the real surface. [SYSC08] introduced a new approach to remove non-isolated surface outlier clusters. In our future work, we will consider this as a preprocessing step followed by the method proposed in this thesis so that it is capable of dealing with the measurement errors generated from scanning devices. A desirous result can then be achieved.

And currently, the cleaning procedure is carried out sequentially on CPU, which is inefficient and slows down the whole process. Redesigning the cleaning algorithm with divide-and-conquer scheme is a way to utilize the parallel computing power of multi-core CPU or many-core GPU. In addition, processing extremely massive point sets requires a large amount of memory. To overcome the memory limitation problem, the whole process can be performed using out-of-core techniques like [WK03]. Other improvements in the quality of the reconstructed meshes are also considered as our future work. For instance, to generate anisotropic meshes, we can include a density mapping according to the local curvatures [VCP08] in the energy function for clustering.

Appendix A

Building Neighborhood Table

This chapter describes how to build the neighborhood table with k -nearest neighbors to each point. We first explain the region subdivision based method for parallel and streaming computation of k NN. Then, a novel approach is introduced which effectively and efficiently constructs the neighborhood table in parallel through a buffer.

A.1 Building Neighborhood Table in Streaming

The neighborhood information is vital in local operations as only some of the neighborhoods are contributed to the computations. We first find out the k NN for each point and store their indices in a table (so-called neighborhood table) which is therefore $k \times N$ (where N is the total number of points) in size and is stored in the global memory of graphics hardware.

The simplest way to find k NN is computing the Euclidean distances between all the data points and the query point first. Then sort the distances in ascending order and the first k points with respect to the distances are the result. It is however very computationally expensive. The authors of [GDB08] implemented this approach on GPU which resulted in a faster computing compared with other traditional k NN algorithms on CPU[MA06]. Nevertheless, the enhancement is only conspicuous when the dimension of data is high. Besides, the size of data set is also limited due to the restriction of GPU. In comparison, our algorithm for finding k NN is based on partition of the region of space that is occupied by the input data so that only a small portion of data is involved in calculation at each time. This enables our algorithm to perform in a streaming manner, thus our approach does not restrict to massive points. Currently, uniform partition is adapted to divide the data into cells of dimension d^3 which is varied according to the number of points.

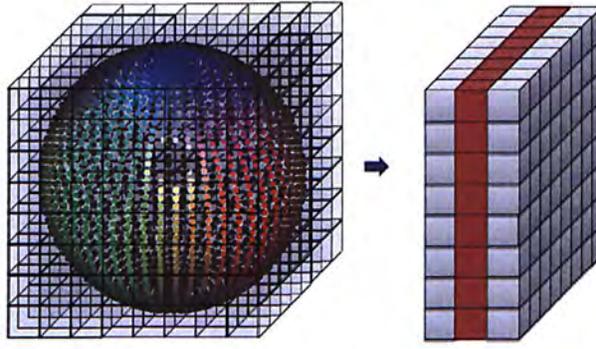


Figure A.1: Uniform partitioning of the point data (left) and streaming to process one layer of cells (red in right) with neighboring two layers.

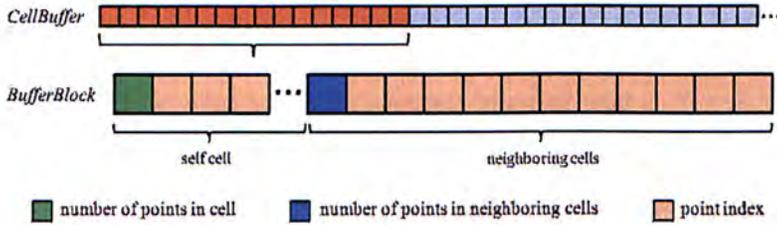


Figure A.2: Data structure of *CellBuffer*.

Given a scattered point data set $P = \{p_i | i = 1, \dots, N\}$, the actual cell size c_{size} is determined as the greatest margin to dimension ratio l/d of the bounding box. Then each point p_i is assigned to its corresponding cell with a *CellID* in parallel and it is given by

$$CellID(x) = (x - x_{min}) / c_{size}. \quad (\text{A.1})$$

After that, a coordinate work is conducted on CPU to form a set of cells $C = \{c_i | i = 1, \dots, d^3\}$ to build the neighborhood table cell by cell (see the illustration in Fig.A.1). As for each point $p \in c_i$, only the points in $Nbhd(c_i)$ are involved in computation. Hence, at each stream, we process one layer of cells in parallel while passing three layers of cell information as illustrated in Fig. A.1.

Such approach however is not very efficient as there can be a relatively high number of empty cells at each stream, which then idle many of the threads. To overcome this problem, we selectively pass the required cells through the *CellBuffer* to GPU. The *CellBuffer* consists of two parts: 1) self cell point indices; 2) neighbor cell point indices (see Fig.A.2). Note that the size of each block $Block_{size}$ must be fixed in order to fetch the correct portion of memory in each thread. We simply set it as the maximum number of points in the $3 \times 3 \times 3$ cells (i.e., $Block_{size} = 27 \times Cell_{size}$). Due to the fixed size of blocks, the first element of each part is used to store the number of points contained in the corresponding part, as for valid visit across the points. Currently, the construction

Algorithm 6: Building Neighborhood Table

```

1: for each point  $p_i$  in parallel do
2:   Compute the CellID by Eq.(A.1);
3: end for
4: Build the CellBuffer;
5: for each BufferBlock  $B_i$  in parallel do
6:    $selfCellPtNum \leftarrow B_i[0]$ ;
7:    $neighborCellsPtNum \leftarrow B_i[i * B_{size} + c_{size}]$ ;
8:    $tempArray \leftarrow$  new array;
9:   for  $i = 0$  to  $selfCellPtNum$  do
10:    for  $j = 0$  to  $selfCellPtNum + neighborCellsPtNum$  do
11:      Compute the square distance  $d$  between  $p_i$  and  $p_j$ ;
12:      if  $d > 0$  and  $d < radius$  then
13:        Store  $p_j$  to  $tempArray$ ;
14:      end if
15:    end for
16:    Sort  $tempArray$  by insertion sort  $k$  times;
17:    Store the result as the first  $k$  elements in  $tempArray$ ;
18:  end for
19: end for

```

of the *CellBuffer* is carried out on the CPU. To further speed up, we ignore the points outside the sphere centered at p with radius = 0.01% of the diagonal of bounding box during sorting. Detail pseudo-code is given in Algorithm 6.

Appendix B

Publications

- Hoi Sheung and Charlie C. L. Wang, “Robust mesh reconstruction from unoriented noisy points”, SIAM/ACM Joint Conference on Geometric and Physical Modeling, October 5-8, 2009, San Francisco, California.
- Hoi Sheung, Siu Ping Mok and Charlie C. L. Wang, “A highly parallel approach to meshing scattered point data”, ASME 2009 International Design Engineering Technical Conferences (IDETC) & Computers and Information in Engineering Conference (CIE), August 30-September 2, 2009, San Diego, California.

Bibliography

- [AA04] ADAMSON A., ALEXA M.: Approximating bounded, non-orientable surfaces from points. In *In Shape Modeling International* (2004), pp. 243–252.
- [ABCO*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics* 9, 1 (January 2003), 3–15.
- [ABK98] AMENTA N., BERN M., KAMVYSSELIS M.: A new voronoi-based surface reconstruction algorithm. In *SIGGRAPH '98* (1998), pp. 415–421.
- [ACDL00] AMENTA N., CHOI S., DEY T. K., LEEKHA N.: A simple algorithm for homeomorphic surface reconstruction. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry* (2000), pp. 213–222.
- [ACK01] AMENTA N., CHOI S., KOLLURI R. K.: The power crust. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications* (2001), pp. 249–266.
- [ACSTD07] ALLIEZ P., COHEN-STEINER D., TONG Y., DESBRUN M.: Voronoi-based variational reconstruction of unoriented point sets. In *SGP '07* (2007), pp. 39–48.
- [ARAA04] ALEXA M., RUSINKIEWICZ S., ALEXA M., ADAMSON A.: On normals and projection operators for surfaces defined by point sets. In *In Eurographics Symp. on Point-Based Graphics* (2004), pp. 149–155.
- [BBA08] BUCHART C., BORRO D., AMUNDARAIN A.: Gpu local triangulation: an interpolating surface reconstruction algorithm. *Comput. Graph. Forum* 27, 3 (2008), 807–814.
- [BBX95] BAJAJ C. L., BERNARDINI F., XU G.: Automatic reconstruction of surfaces and scalar fields from 3d scans. In *SIGGRAPH '95* (1995), pp. 109–118.

- [BKBH07] BOLITHO M., KAZHDAN M., BURNS R., HOPPE H.: Multilevel streaming for out-of-core surface reconstruction. In *SGP '07* (2007), pp. 69–78.
- [BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H. E., SILVA C. T., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. Vis. Comput. Graph.* 5, 4 (1999), 349–359.
- [CA99] CROSSNO P., ANGEL E.: Spiraling edge: Fast surface reconstruction from partially organized sample points. In *IEEE Visualization* (1999), pp. 317–324.
- [CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01* (2001), pp. 67–76.
- [CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: measuring error on simplified surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174.
- [CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *SIGGRAPH '04* (2004), pp. 905–914.
- [CSF02] COHEN-STEINER D., F. D.: *A greedy Delaunay-based surface reconstruction algorithm*. Tech. rep., 2002.
- [DG03] DEY T. K., GOSWAMI S.: Tight cocone: a water-tight surface reconstructor. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), pp. 127–134.
- [DGH01] DEY T. K., GIESEN J., HUDSON J.: Delaunay based shape reconstruction from large data. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics* (2001), pp. 19–27.
- [DTB06] DIEBEL J. R., THRUN S., BRÜNIG M.: A bayesian method for probable surface reconstruction and decimation. *ACM Trans. Graph.* 25, 1 (2006), 39–59.
- [FB81] FISCHLER M. A., BOLLES R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.
- [FCOS05] FLEISHMAN S., COHEN-OR D., SILVA C. T.: Robust moving least-squares fitting with sharp features. In *SIGGRAPH '05* (2005), pp. 544–552.

- [GDB08] GARCIA V., DEBREUVE E., BARLAUD M.: Fast k nearest neighbor search using gpu. *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on* (June 2008), 1–6.
- [GKS00] GOPI M., KRISHNAN S., SILVA C. T.: Surface reconstruction based on lower dimensional localized delaunay triangulation. *Comput. Graph. Forum* 19, 3 (2000).
- [HDD*92] HOPPE H., DEROSE T., DUCHAMP T., McDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *SIGGRAPH '92* (1992), pp. 71–78.
- [HDD*94] HOPPE H., DEROSE T., DUCHAMP T., HALSTEAD M., JIN H., McDONALD J., SCHWEITZER J., STUETZLE W.: Piecewise smooth surface reconstruction. In *SIGGRAPH '94* (1994), pp. 295–302.
- [HK06] HORNUNG A., KOBELT L.: Robust reconstruction of watertight 3d models from non-uniformly sampled point clouds without normal information. In *SGP '06* (2006), pp. 41–50.
- [HMS06] HUNT W., MARK W. R., STOLL G.: Fast kd-tree construction with an adaptive error-bounded heuristic. In *2006 IEEE Symposium on Interactive Ray Tracing* (Sept. 2006), IEEE.
- [IJS03] IVRISIMTZIS I. P., JEONG W.-K., SEIDEL H.-P.: Using growing cell structures for surface reconstruction. In *SMI '03: Proceedings of the Shape Modeling International 2003* (Washington, DC, USA, 2003), p. 78.
- [JWB*06] JENKE P., WAND M., BOKELOH M., SCHILLING A., STRASSER W.: Bayesian point cloud reconstruction. *Comput. Graph. Forum* 25, 3 (2006), 379–388.
- [Kaz05] KAZHDAN M.: Reconstruction of solid models from oriented point sets. In *SGP '05* (2005), p. 73.
- [KBH06] KAZHDAN M., BOLITHO M., HÖPPE H.: Poisson surface reconstruction. In *SGP '06* (2006), pp. 61–70.
- [KV03] KALAI AH A., VARSHNEY A.: Modeling and rendering of points with local geometry. *IEEE Transactions on Visualization and Computer Graphics* 9, 1 (2003), 30–42.
- [KY06] KUO C.-C., YAU H.-T.: A new combinatorial approach to surface reconstruction with sharp features. *IEEE Trans. Vis. Comput. Graph.* 12, 1 (2006), 73–82.

- [LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.* 21, 4 (1987), 163–169.
- [Lev03] LEVIN D.: Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization* (2003), Springer-Verlag, pp. 37–49.
- [Lie03] LIEPA P.: Filling holes in meshes. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 200–205.
- [Llo82] LLOYD S. P.: Least squares quantization in pcm. *IEEE Transactions on Information Theory* 28 (1982), 129–136.
- [LWL*08] LIU Y., WANG W., LVY B., SUN F., YAN D. M., LU L., YANG C.: *On Centroidal Voronoi Tessellation - Energy Smoothness and Fast Computation*. Tech. rep., Hong-Kong University and INRIA - ALICE Project Team, 2008. Accepted pending revisions.
- [MA06] MOUNT D. M., ARYA S.: Ann: A library for approximate nearest neighbor searching. <http://www.cs.umd.edu/mount/ANN/>.
- [MAVdF05] MEDEROS B., AMENTA N., VELHO L., DE FIGUEIREDO L. H.: Surface reconstruction from noisy point clouds. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing* (2005), p. 53.
- [MP04] MARK PAULY NILOY MITRA L. G.: Uncertainty and variability in point cloud surface data. In *Symposium on Point-Based Graphics 2004* (2004).
- [MS96] MILLER J. V., STEWART C. V.: Muse: Robust surface fitting using unbiased scale estimates. In *CVPR* (1996), pp. 300–.
- [NVI08] NVIDIA: *CUDA programming guide 2.0*. Tech. rep., July 2008.
- [OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Trans. Graph.* 22, 3 (2003), 463–470.
- [OBS05a] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: 3d scattered data interpolation and approximation with multilevel compactly supported rbfs. *Graph. Models* 67, 3 (2005), 150–165.
- [OBS05b] OHTAKE Y., BELYAEV A. G., SEIDEL H.-P.: An integrating approach to meshing scattered point data. In *Symposium on Solid and Physical Modeling* (2005), pp. 61–69.

- [Pet02] PETITJEAN S.: A survey of methods for recovering quadrics in triangle meshes. *ACM Comput. Surv.* 34, 2 (2002), 211–262.
- [PGK02] PAULY M., GROSS M. H., KOBBELT L.: Efficient simplification of point-sampled surfaces. In *IEEE Visualization* (2002).
- [SAAY06] SAMOZINO M., ALEXA M., ALLIEZ P., YVINEC M.: Reconstruction with voronoi centered radial basis functions. In *SGP '06* (2006), pp. 51–60.
- [SBS05] SCHALL O., BELYAEV A., SEIDEL H.-P.: Robust filtering of noisy scattered point data. In *IEEE/Eurographics Symposium on Point-Based Graphics* (Stony Brook, New York, USA, 2005), Pauly M., Zwicker M., (Eds.), pp. 71–77.
- [SDK09] SCHNABEL R., DEGENER P., KLEIN R.: Completion and reconstruction with primitive shapes. *Computer Graphics Forum* 28, 2 (2009), 503–512.
- [SF08] SONG H., FENG H.-Y.: A global clustering approach to point cloud simplification with a specified data reduction ratio. *Computer Aided Design* 40, 3 (2008), 281–292.
- [SLS*06] SHARF A., LEWINER T., SHAMIR A., KOBBELT L., COHEN-OR D.: Competing fronts for coarse-to-fine surface reconstruction. *Computer Graphics Forum* 25, 3 (2006), 389–398.
- [SMS05] SILVA S., MADEIRA J., SANTOS B. S.: Polymeco ” a polygonal mesh comparison tool. In *IV '05: Proceedings of the Ninth International Conference on Information Visualisation* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 842–847.
- [SSB05] STEINKE F., SCHÖLKOPF B., BLANZ V.: Support vector machines for 3d shape processing. *Comput. Graph. Forum* 24, 3 (2005), 285–294.
- [SSK07] SHEVTSOV M., SOUPIKOV A., KAPUSTIN A.: Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Comput. Graph. Forum* 26, 3 (2007), 395–404.
- [SYSC08] SHEN J., YOON D., SHEHU D., CHANG S.-Y.: Spectral moving removal of non-isolated surface outlier clusters. *Computer-Aided Design In Press, Corrected Proof* (2008), –.
- [TM02] TORDOFF B., MURRAY D. W.: Guided sampling and consensus for motion estimation. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision – Part I* (London, UK, 2002), Springer-Verlag, pp. 82–98.

- [TO99] TURK G., O'BRIEN J. F.: Shape transformation using variational implicit functions. In *SIGGRAPH '99* (1999), pp. 335–342.
- [VCP08] VALETTE S., CHASSERY J. M., PROST R.: Generic remeshing of 3d triangular meshes with metric-dependent discrete voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics* 14, 2 (2008), 369–381.
- [WK03] WU J., KOBELT L.: A stream algorithm for the decimation of massive meshes. In *Graphics Interface* (2003), pp. 185–192.
- [WS04] WANG H., SUTER D.: MDPE: A very robust estimator for model fitting and range image segmentation. *Int. J. Comput. Vision* 59, 2 (2004), 139–166.
- [XMQ04] XIE H., McDONNELL K. T., QIN H.: Surface reconstruction of noisy and defective data sets. In *VIS '04: Proceedings of the conference on Visualization '04* (2004), pp. 259–266.
- [ZGHG08] ZHOU K., GONG M., HUANG X., GUO B.: Highly parallel surface reconstruction. Microsoft Technical Report, MSR-TR-2008-53, <http://www.kunzhou.net/2008/MSR-TR-2008-53.pdf>.
- [ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time kd-tree construction on graphics hardware. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (New York, NY, USA, 2008), ACM, pp. 1–11.

CUHK Libraries



004660027