Winter 2018

Adaptive Methods for Point Cloud and Mesh Processing

Zinat Afrose
Old Dominion University, zinatafrose@gmail.com

ADAPTIVE METHODS FOR POINT CLOUD AND MESH PROCESSING

by

Zinat Afrose
B.S. May 2009, Jahangirnagar University, Bangladesh
M.S. January 2012, Jahangirnagar University, Bangladesh

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
December 2018

Approved by:

Dr. Yuzhong Shen (Director)

Dr. Rick Mckenzie (Member)

Dr. Jiang Li (Member)

Dr. Ahmed K. Noor (Member)

ABSTRACT

ADAPTIVE METHODS FOR POINT CLOUD AND MESH PROCESSING

Zinat Afrose
Old Dominion University, 2018
Director: Dr. Yuzhong Shen

Point clouds and 3D meshes are widely used in numerous applications ranging from games to virtual reality to autonomous vehicles. This dissertation proposes several approaches for noise removal and calibration of noisy point cloud data and 3D mesh sharpening methods. Order statistic filters have been proven to be very successful in image processing and other domains as well. Different variations of order statistics filters originally proposed for image processing are extended to point cloud filtering in this dissertation. A brand-new adaptive vector median is proposed in this dissertation for removing noise and outliers from noisy point cloud data.

The major contributions of this research lie in four aspects: 1) Four order statistic algorithms are extended, and one adaptive filtering method is proposed for the noisy point cloud with improved results such as preserving significant features. These methods are applied to standard models as well as synthetic models, and real scenes, 2) A hardware acceleration of the proposed method using Microsoft parallel pattern library for filtering point clouds is implemented using multicore processors, 3) A new method for aerial LIDAR data filtering is proposed. The objective is to develop a method to enable automatic extraction of ground points from aerial LIDAR data with minimal human intervention, and 4) A novel method for mesh color sharpening using the discrete Laplace-Beltrami operator is proposed.

Median and order statistics-based filters are widely used in signal processing and image processing because they can easily remove outlier noise and preserve important features. This

dissertation demonstrates a wide range of results with median filter, vector median filter, fuzzy vector median filter, adaptive mean, adaptive median, and adaptive vector median filter on point cloud data. The experiments show that large-scale noise is removed while preserving important features of the point cloud with reasonable computation time. Quantitative criteria (e.g., complexity, Hausdorff distance, and the root mean squared error (RMSE)), as well as qualitative criteria (e.g., the perceived visual quality of the processed point cloud), are employed to assess the performance of the filters in various cases corrupted by different noisy models. The adaptive vector median is further optimized for denoising or ground filtering aerial LIDAR data point cloud. The adaptive vector median is also accelerated on multi-core CPUs using Microsoft Parallel Patterns Library. In addition, this dissertation presents a new method for mesh color sharpening using the discrete Laplace-Beltrami operator, which is an approximation of second order derivatives on irregular 3D meshes. The one-ring neighborhood is utilized to compute the Laplace-Beltrami operator. The color for each vertex is updated by adding the Laplace-Beltrami operator of the vertex color weighted by a factor to its original value. Different discretizations of the Laplace-Beltrami operator have been proposed for geometrical processing of 3D meshes. This work utilizes several discretizations of the Laplace-Beltrami operator for sharpening 3D mesh colors and compares their performance. Experimental results demonstrated the effectiveness of the proposed algorithms.

This dissertation is dedicated to my family for their endless support and encouragement.

ACKNOWLEDGMENTS

This dissertation is the culmination of my journey of Ph.D. research which was just like climbing a high peak step by step accompanied by encouragement, hardship, trust, and frustration. This dissertation would not be possible to finish without the invaluable contribution of a great number of people to whom I would like to convey my gratitude in this acknowledgment.

First, I would like to express my sincere gratitude to my advisor Dr. Yuzhong Shen for his enormous support, insightful guidance, and inspiring attitude. Under his guidance, I successfully overcame many difficulties and learned a lot. His zeal for perfection, passion, unflinching courage, and conviction has always inspired me to do more. He has supported me not only academically but also emotionally through the rough road to finish this dissertation. For all these reasons, I sincerely thank him from the bottom of my heart and will be truly indebted to him throughout my lifetime.

Besides my advisor, I would like to thank the rest of my dissertation committee members: Dr. Rick Mckenzie, Dr. Jiang Li, and Dr. Ahmed K. Noor for their great support and invaluable advice.

Importantly, my spiritual leader, through whom I get inspiration to go beyond the known and know the unknown, I am so humble and grateful to him.

Where would I be without my family? My parents and my in-laws deserve special thanks for their devoted support and prayers. I am deeply grateful to my dear mother who worked with undying support for me throughout my whole study life. Moreover, my father...I know he is watching me from heaven and feels proud to see his child fulfilling his dream.

Thanks to my siblings for their continued patience, and endless support. Also, thanks to my friend Hasib for his countless endurance and help during this journey.

I would like to express my deepest appreciation to my husband, Mahbubul Alam, for his continued and constant support and understanding during my pursuit of Ph.D. degree that made the completion of this dissertation possible. He was always by my side when I thought that it was impossible to continue and encouraged me to work harder. Finally, I would like to mention my baby, my little bundle of joy Zaaib, who is an integral part of this journey. I am grateful to both of them for bringing pure joy and blessings to my life.

Lastly, thanks to the Almighty Allah for giving me the strength to take challenges and for His countless blessings in my life.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

This chapter briefly defines the motivation behind the work of this dissertation. It then discusses the objectives of this dissertation and concludes with the dissertation structure[1].

1.1 Motivation

Three-dimensional (3D) models are widely used in a variety of applications, such as game development, computer animation, movies, preservation of historical heritage and mechanical devices, and virtual reality walkthroughs. Most of these applications demand an accurate and usable computer model of an object which best suits the underlying application, be it to render improved and noise free presentation of the object from arbitrary viewpoints under different lighting conditions, or for accurate computations and simulations. There are two common approaches to create 3D models: either the model is designed from scratch using interactive modeling software, or the model is digitized from a physical object using acquisition hardware and algorithms to reconstruct a 3D model from the acquired 3D data. For the latter approach, point clouds are a natural way to represent 3D sensor output, and no available connectivity information can be assumed from the underlying topology in point clouds. Working directly with raw point clouds in the input 3D space offers several advantages, such as better suited for applications requiring data addition and deformation. Although point clouds can be rendered directly using points or textures, more common use of point cloud data is to generate 3D surface meshes for graphics rendering and other purposes such as modeling and simulation,

[1] IEEE Transactions and Journals style is used in this thesis for formatting figures, tables, and references.

as 3D meshes possess topological information for easy handling of the neighborhood. Point clouds are used in a wide range of applications. Fig. 1 [1] shows the point cloud generated by scanning an industrial facility and the corresponding 3D mesh generated from the point cloud. Fig. 2 shows the point clouds generated by using LIDAR (Light Detection and And Ranging) laser scanners to scan terrain, and the final digital elevation model (DEM) generated after processing the point cloud data. LIDAR scanning can cover large areas uniformly and rapidly, and DEM data are widely used in forest planning and management, environmental assessment, defense, and gaming, to mention just a few.

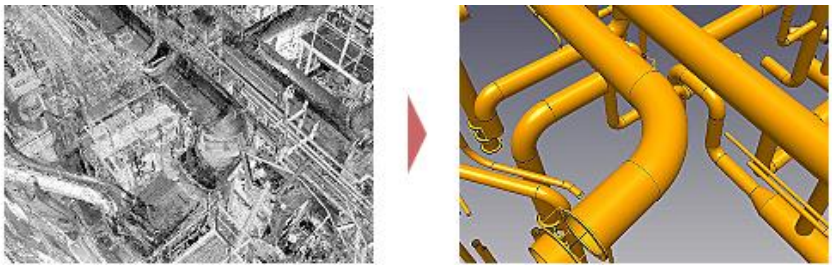


Fig. 1. Point Cloud Data in CAD Model Generation [1].

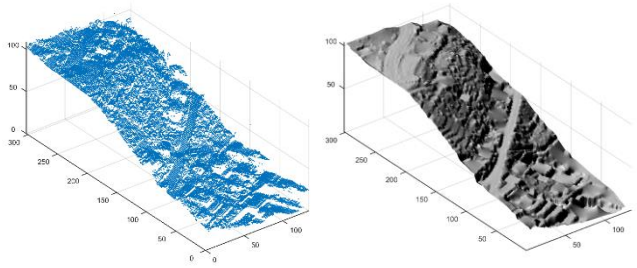


Fig. 2. LIDAR data in DEM generation.

In addition to scanning large-scale figures (plant and terrain) as discussed above, point clouds are heavily used individual objects as well. Fig. 3 shows the point cloud by scanning a small statue and the corresponding 3D mesh generated from the point cloud. More and more such point clouds are used in medical modeling and simulation, manufacturing, architecture, 3D printing, gaming, and various virtual reality (VR) applications.

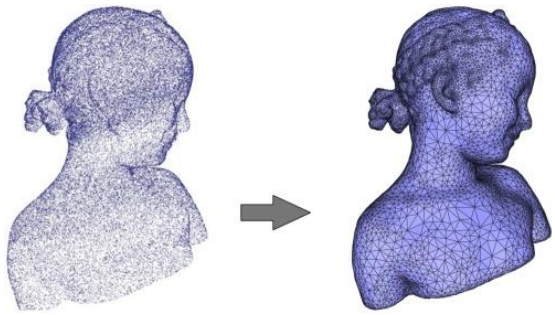


Fig. 3. Point cloud to 3D mesh model generation.

While 3D data acquisition hardware has advanced tremendously, various types of noise are still introduced in the acquisition process, caused by the limitation of device precision, influence, and reflection of light, shadows, low contrast, etc. The noise present in the point clouds cause distortion of the 3D surfaces reconstructed from the point clouds. Also, a 3D scan of the environment includes all objects in the environment, some of which might not be the data or information needed. For example, the initial LIDAR scan of the terrain includes vegetation and other objects, while the purpose of the scan is the terrain elevation. In this case, vegetation and other undesired objects should be removed to extract the true and accurate terrain information. This dissertation will focus on how to improve the quality of the noisy data models with structural and visual improvement. Various filtering for point clouds will be proposed and developed, as well as a method for improving the quality of 3D color meshes.

1.2 Objectives

There are four objectives to be achieved by the research in this dissertation. The first objective is to develop and implement point cloud filtering algorithms to automatically reduce the amount of noise and outliers in small-scale point cloud datasets. Outliers are undesired noise that introduces errors in applications using point cloud data. Hence, trimming them out of the point cloud will produce point clouds of better quality that facilitate further usage of point clouds. Several filtering methods originally developed for image processing are extended to point cloud filtering, including vector median, fuzzy vector median, adaptive mean, and adaptive vector median filters. A completely new method, namely adaptive vector median (AVM) filter, is proposed in this dissertation and utilized for point cloud filtering. The AVM is able to preserve detail while eliminating or reducing the impulse noise and outliers in the point cloud. The second objective is to implement parallel processing of the AVM filter to accelerate processing of huge datasets. Despite the efficiency of the AVM filter, additional efforts are required that increase the computational time. Furthermore, the availability of desktop multicore and multithreaded processors offers new opportunities to speed up point cloud filtering. The proposed solution achieves a computational time gain close to the number of physical cores. The third objective is to optimize the AVM filter for processing of aerial LIDAR data. The non-ground objects are eliminated by applying a threshold value based on elevation differences and terrain slope, and the remaining noisy points are removed by using the AVM filter. The fourth and final objective of this dissertation is to develop and implement color mesh sharpening method to improve the quality of the 3D mesh. This method extended traditional image sharpening techniques for 2D regular images to irregular 3D meshes. In particular, this method utilized several discretizations

of the Laplace-Beltrami operator, including Pinkall, Meyer, Mayer, Desbrun-1, and Desbrun-2 discretizations and was applied to various kinds of 3D models for color mesh sharpening.

1.3 Dissertation Structure

The dissertation is organized in four parts: noisy point cloud filtering, parallel implementation of point cloud filtering, ground filtering of aerial LIDAR data, and 3D color mesh sharpening. Chapter 1 presents the motivation and objectives for the work in this dissertation and the overall dissertation structure. Chapter 2 gives a literature review on point cloud processing, parallel processing, LIDAR data filtering, and 3D meshes. The first contribution of this dissertation is introduced in Chapter 3, including five different filtering methods for the noisy point cloud, namely vector median filter, fuzzy vector median filter, adaptive mean, adaptive median, and adaptive vector median filter. Parallel implementation of the adaptive vector median filter is presented in Chapter 4. Chapter 5 describes the method for aerial LIDAR data ground filtering. Chapter 6 presents the algorithm for color mesh sharpening. Chapter 7 concludes the dissertation and discusses directions for future research.

CHAPTER 2

LITERATURE REVIEW

This chapter presents a review of the literature that is relevant to the work in this dissertation, including point cloud processing, multi-core and GPU-based parallel computing, aerial LIDAR data processing, and mesh processing.

2.1 Point Cloud Processing

Point clouds are a natural way to represent 3D sensor outputs with no assumption of connectivity information or underlying topology. In early efforts of digitization of several cultural heritage sites and statues, processing a huge amount of noisy 3D data in a reasonable time presented a big challenge [2], [3], [4]. Many techniques have been introduced to remove noise and outliers from the scanned point cloud using smoothing filters [5], [6], [7], [8], [9], [10]. However, this approach could not preserve sharp features, which were undesirable for some cases. Wang et al. [11] combined fuzzy c-means and bilateral filtering and produced good results but also partly smoothed the sharp features while clustering. Data clustering is robust for removing noise [12], [13] though it requires prior knowledge about the input objects. Mederos et al. [14] introduced a smoothing operator that could preserve the significant edges of the surface, which was inspired by the moving least square method and robust statistics theory. However, this approach only removed a small range of noise, but the elimination of outliers remained mostly a manual procedure. Lea [15] presented a GPU-based implementation of moving least squares and Liu and Zayer [16] proposed Bundle Adjustment for the multiview point cloud.

Both approaches [15], [16] were intended for smoothing the point cloud data. However, both smoothing and sharp feature preservation become a challenge and computationally expensive for huge datasets. Majority voting method [17], an improved approach to k-means clustering [18], [19], cluster analysis and segmentation [20], and K-NN algorithm with clustering [21] all required high computational cost for large datasets. A new approach of SVM was introduced for noise reduction in the point cloud based on density and distance [22] for a small-scale data. Several studies [17], [18], [23], [24], [25] were intended for small data sets and consumed much execution time for large datasets. An image processing technique namely Wiener filter was used with patch collaborative spectral analysis [26]. This image-based technique was also computationally demanding. Deschaud and Goulette [27] presented an approach to handle large datasets by filtering normal and voxel growing for plane detection in the presence of noise, but this approach failed in the border with small variation. In a different paper [28], they identified and removed outliers by utilizing a dissimilarity measure based on point positions and normal, but the quality decreases if the voxel size is large. Digne [29] introduced low/high-frequency decomposition by comparing the neighborhoods of the points. This method, however, required reasonable point cloud density. Estimation of threshold in the high-density point cloud was also considered [30], [31]. However, this may lead to holes in the regions where noise and outliers are concentrated. These methods tend to eliminate critical features such as sharp edges or corners. Various approaches were attempted for extracting sharp features in point clouds. Delaunay tetrahedralization ([32], [33]) produced surface meshes from noisy point cloud while preserving important features. Zheng et al. [34] discoursed this issue with multiple normals according to the feature type. In this case, large datasets were not taken

into account. A wide range of comparisons on point cloud denoising algorithms and evaluation of the subjective performance of the well-known quality metrics were presented in [35] and [36].

2.2 Multi-core and GPU-based Parallel Computing

With advances in hardware design and VLSI technologies, a single processor VLSI chip now contains multiple cores, called multi-core or many-core processors. For example, an Intel Xeon processor can have as many as 24 cores on a single chip. Therefore, computations can be partitioned into multiple subtasks and then allocated to multiple cores on the same CPU chip for parallel processing. Multi-core processor architecture contains several execution cores within a single processor package. Multi-core processors now are a standard configuration on desktop and laptop computers and even smartphones. Graphics Processing Units (GPUs) are another category of computing hardware that is now widely used for parallel computing on personal computers, workstations, and clusters. GPU contains multiple Streaming Multiprocessors (SMs), and each SM consists of many CUDA cores (also called Stream Processors, or SPs). The latest NVidia GPUs contain thousands of CUDA cores and thus can execute thousands of threads concurrently. GPUs are optimized for computations used by computer graphics, such as affine transformations, lighting, and texture mapping. In recent years, GPUs have been utilized for general purpose computing (GPGPU) such as image processing, computational fluid dynamics, and machine learning. Yang et al. [37] calculated integral images on GPU to accelerate the whole cost-volume filtering process whereas Aitali et al. [38] proposed a SIMD architecture for bias field estimation and image segmentation. Three different GPUs have been utilized to accelerate compute-intensive portions of the original sequential code. The speedup depends on the model of the GPU, image size, and number of clusters. Galliani et al. [39] presented a multiview variant of

Patchmatch Stereo with a new, highly parallel proliferation scheme that delivered dense multiview correspondence over ten 1.9-Megapixel images in 3 seconds on a consumer-grade GPU. It achieved an accurate and complete reconstruction with the low runtime. Based on NVIDIA CUDA, Anderson et al. [40] and Li et al. [41] proposed an improved classic Fuzzy C-Means clustering algorithm which adaptively updated membership values and the update criterion of cluster centers. Their methods produced better visual effects and segmentation efficiency. A 2-level parallel computing framework to accelerate the SVM was proposed in [42] by utilizing CUDA and OpenMP. Wu et al. [43] presented a computationally efficient parallel implementation of a spectral-spatial classification method based on adaptive Markov random fields. It was more accurate and 70 times faster than the original sequential code. El-Nashar [44] discussed the issue of speedup gained from parallelization using MPI and proposed a way to predict the speedup of MPI application. Parallelization was also utilized in 3D point cloud matching and filtering [45], [29]. Jorge et al.'s approach [45] attained computational gain, which was close to the number of cores. Digne et al. [29] analyzed a parallel implementation of the bilateral filter for the point cloud. The registration problem for 3D scans was addressed with GPGPU [46], and the nearest neighbor search algorithm was used for 3D point cloud registration. The registration of a large dataset is computationally expensive. Their method was able to achieve a speedup of 88 over the sequential algorithm. Kun et al. [16] presented a parallel surface reconstruction algorithm that ran entirely on GPU. This approach produced high-quality surfaces through global optimization. GPU was also used to speed up the process of filtering LIDAR data significantly [47].

2.3 Aerial LIDAR Data Processing

Aerial Light detection and ranging (LIDAR) integrates the Global Navigation Satellite System and Inertial Navigation System with laser scanning and ranging technologies. It enables direct measurement of the 3-D coordinates of points on ground objects for the efficient creation of digital surface models (DSMs). The large volume of scanned data that are manipulated when processing a LIDAR point cloud has been one of the major challenges in data processing. For example, one strip of a scanned area can easily produce tens of millions of points. Efficient algorithms are, therefore, important in practical applications. For some critical fields, such as emergency response, very short data processing time is required. For example, after an earthquake, terrain maps are required urgently for damage estimation and rescue plans. The filtering of the LIDAR point cloud is an important step in LIDAR data processing. It classifies the LIDAR points into ground points and nonground points, which are objects such as buildings, trees, and low vegetation. Filtering is one of the most important steps in producing the digital elevation model (DEM) and terrain information.

The diversity of the terrain, the complexity of features, and the irregular distribution of the points bring significant difficulties to the filtering process [48]. For many years, researchers showed filtering LIDAR data is an extremely problematic task and is still currently actively under investigation [49], [50], [51], [52], [53], [54], [55]. Researchers have proposed different types of filtering methods. These methods can be grouped into several categories based on the filter strategies, such as iterative interpolation, morphology, slope, segmentation or clustering, region, machine learning, statistical analysis, as discussed below.

Interpolation based filtering: For interpolation-based methods, the initial ground points are selected and then densified iteratively to create a provisional surface that gradually approaches the final ground surface [49], [50], [56], [57], [58].

Morphology-based filtering: This algorithm originated in mathematical morphology theory, which uses morphological operations, such as opening operation [59], to approximate terrain surface or building detection [60, 61]. Kilian et al. [62] proposed a progressive morphological filter based on a series of opening operations applied to a gridded surface model to remove the objects with different size. The progressive morphological filtering method proposed by Keqi et al. [59] used the increased radius of the structuring element to remove non-ground points. However, these methods generally assume the terrain has a constant slope. Chen proposed a morphological algorithm with varying slope [63]. The biggest challenge for these methods was how to maintain the terrain features when the size of the filter window changes. Silva et al. [64] evaluated four ground filtering algorithm and showed progressive morphological filters achieved less accuracy than the other three algorithms.

Slope-based filtering: The common assumption of slope-based algorithms is that the change in the slope of terrain is usually gradual in a neighborhood, while the change in slope between buildings or trees and the ground is very large. Based on this assumption, Vosselman [65] developed a slope-based filtering algorithm by comparing slopes between a LIDAR point and its neighbors. To improve the calculation efficiency, Shan and Sampath [66] calculated the slopes between neighbor points along a scan line in a specified direction, which was extended to multidirectional scan lines by Meng et al. [61]. Acquiring an optimal slope threshold that can be applied to terrain with different topographic features is difficult with these methods.

Segmentation/ Clustering based filtering: The motivation behind such procedures is that any points that cluster must belong to an object if their cluster is above its neighborhood. For such a concept to work the clusters/segments must delineate objects and not facets of objects. There are various ways in which cluster boundaries or segments can be obtained. Clustering methods have been proposed by Filin [67] and Roggero [68]. These clustering methods work by projecting and separating the data into a feature space. Segmentation algorithms have been proposed by Lee and Schenk [69], Hosseini [70], Liu et al. [71] and Sithole [72]. Another way of obtaining cluster boundaries is to contour the point-cloud. An object is then suspected to exist where the length (or internal area) of a contour does not grow significantly from a lower contour. This idea is employed by Zhan et al. [73] and Elmqvist [74, 75].

Machine learning based filtering: Machine learning has been used in pattern recognition, classification, regression, and clustering for a long time. The deep convolutional neural networks (CNN) are inspired by biological vision systems; these networks have recently shown their ability to extract high-level representations through compositions of low-level features. Hu and Yuan [47] proposed ground filtering based on CNN. Classification of individual trees [76] and above ground object classification [77], [78] utilized deep learning. Backpropagation neural network [79], Support Vector Machine [80], [81], [82], [83] and random forest [84] are widely used in classification technique for LIDAR data. The key benefit of using this type of methodology is the simplicity and clarity of the resulting model. On the contrary, they also have some drawbacks: they provide a set of highly correlated predictors with little physical justification and require long times to train the model.

Many experiments and projects have been applied various filtering algorithms to range images [85], [86], [87], [88], [89], [90], [12]. Several papers [60], [64], [91], [92], evaluated and

analyzed different ground filtering approach and concluded that not all of the algorithms were capable of producing reliable results but adaptive filtering algorithms have more promising results.

2.4 Mesh Processing

Various methods have been proposed to recover the quality of the meshes generated by 3D scanners, such as surface smoothing [93], which removes geometrical noise in the mesh using Laplacian smoothing. However, local Laplacian smoothing leads to a variety of artifacts such as geometric distortion and shrinkage due to the irregular connectivity of the mesh. Several techniques were proposed to eliminate this shrinkage problem and topological effects of smoothing [94], [95], [96]. Wang [97] proposed a sharpening method using bilateral filtering followed by iteratively modifying the mesh's connectivity to form single wide, sharp edges that were detected by their dihedral angles. A distance measure was defined based on normal tensor analysis [98]. This algorithm consisted of two stages that require much computation time and worked only around the edge features of the model. Particular focus was also on edge sharpening. Attene et al. [99] applied a filtering approach that required subdivision of Chamfer triangles. Ohtake et al. [100] proposed polyhedral surface smoothing that was a combination of Laplacian smoothing flow and discrete mean curvature flow. Another approach for smoothing surfaces was introduced in [101] using fuzzy vector median filters for surface normal filtering in a two-step procedure. Anisotropic geometric diffusion was proposed for surface fairing in [102]. The multiscale method combined the image processing methodology based on nonlinear diffusion equations and the theory of geometric evolution problems for surface processing. This method smoothed the surface by enhancing the edges and corners of the surface. Surface fairing

or removing rough features was also conducted by [103] and [104]. Hildebrandt and Polthier [103] proposed an algorithm based on a constraint that controls the spatial deviation of the surface. Shen et al. [105] applied normal filtering to improve the quality of the mesh surface and remove the noise. This geometric approach consumed much computational cost because of its feature detection stage. Since Laplacian cannot be applied to the irregular meshes due to the irregular topology of meshes, Laplace-Beltrami operator was introduced in different applications, such as computational fluid dynamics [106], [107] and shape segmentation [108]. Petronetto et al. [108] introduced a mesh-free discrete Laplace-Beltrami operator that is defined on point-based surfaces for filtering and shape segmentation. Belkin et al. [109] proposed an algorithm to approximate the Laplace operator of a surface with point-wise convergence that is applicable to arbitrary meshed surfaces. Scale-dependent Laplacian operator was utilized in [110] to improve the smoothness of surface with volume preservation. Gu et al. [111] applied discrete Laplace-Beltrami operator to determine the discrete Riemannian metric. To solve the convergence problem for numerical simulations over the surfaces, Wu et al. [112] and Xu [113] introduced a convergent algorithm of Laplace-Beltrami operator. Xiong et al. utilized this convergent property of Laplace-Beltrami operator for mesh surface smoothing in [114]. Wetzler et al. [115] applied the Laplace-Beltrami operator as a diffusion filter and an invariant metric to obtain geometric shape matching. All of these methods used different approaches for geometric processing. The Laplace-Beltrami operator has been used only for geometrical processing, not for color processing.

CHAPTER 3

POINT CLOUD PROCESSING

This chapter starts with an introduction to point clouds and various types of noise present in point clouds. It then describes the details of five filtering methods for point clouds, including vector median, fuzzy vector median, adaptive mean, adaptive median, and adaptive vector median, and their experimental results. The chapter ends with an implementation of a variant of the adaptive vector median.

3.1 What is Point Cloud

A point cloud (Fig. 4) is a data structure used to denote a group of multi-dimensional points and is commonly used to represent three-dimensional data [116]. In a three-dimensional coordinate system, these points are usually defined by *X*, *Y*, and *Z* coordinates, and often are intended to represent the peripheral surface of an entity.

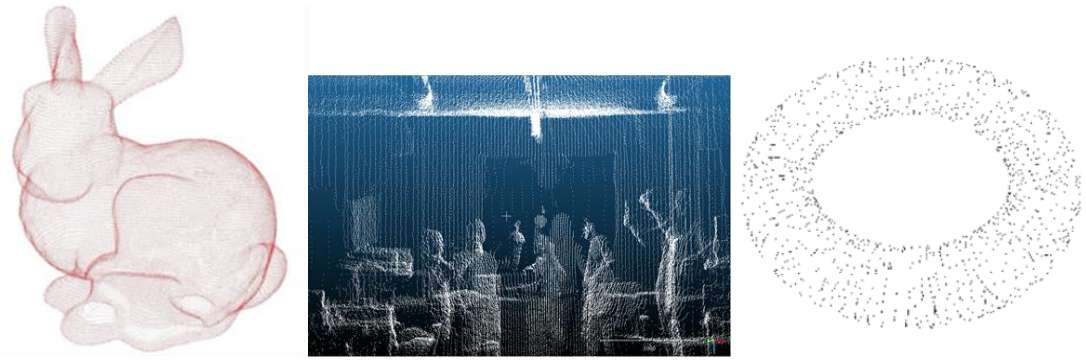


Fig. 4. Examples of point clouds.

3.2 Categories of Point Clouds

Unorganized point clouds are captured from varied inputs like RGB-D cameras, stereo cameras, 3D laser scanners, time-of-flight cameras (Fig. 5), photogrammetric image measurements, motion sensors, or synthetically from software. They pose a tough problem of reconstruction, especially challenging in case of incomplete, noisy, and sparse data. Despite the fact that in practice the sample points produced by a 3D scanner are measured with some regularity, the points in a point cloud are typically not assumed to have any particular structure. The reason for this is to make the algorithms operating on point clouds as general as possible, not depending on the scanner or the way the object was scanned. Obviously, efficient processing of such unorganized point clouds is a central issue in all 3D scanning applications. Depending on the size of the object, its geometry, and the required precision of the scan, different approaches are used. Many technologies exist today to acquire 3D point clouds from various environments. Range-based technologies include 3D laser scanners (also known as terrestrial laser scanners) and time-of-flight (ToF) cameras. Accuracies of laser scanners at the present time are generally within 1 to 5 millimeters. The accuracy of Leica HDS2500 laser scanner is 5mm at 100m [117], and for Leica TC2003 Total Station the accuracy is 1 mm over the range of 2.5 to 3.5 km [118]. 3D laser scanning has become a relatively matured technology, and many commercialized systems are available such as Faro, Leica, Riegel, Topcon, Trimble, Zoller & Frohlich, and others. Laser triangulation-based 3D scanners are less accurate, but significantly faster, which project a laser beam on the object and use the triangulation principle to derive the distance to the object. Structured light scanners project an entire 2D pattern onto the object and calculate the 3D surface points by analyzing the deformation of the pattern. The advantage of structured light scanners is their fast speed so that they can be used to scan moving and deforming objects.

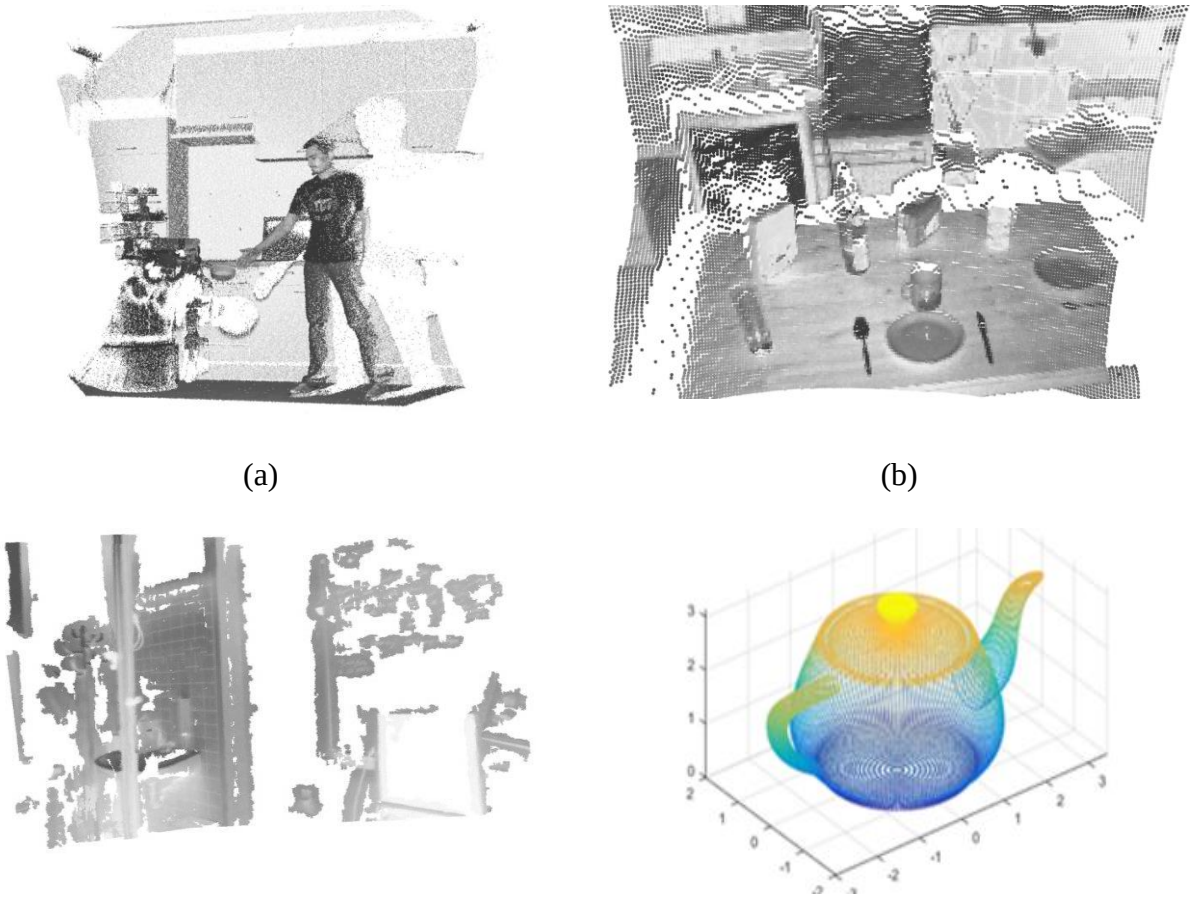
A time-of-flight camera has several benefits; for example, it can measure 3D depth maps at video rate, and as a result, it can be employed as a fast object scanner. The travel time of infrared light is one of the measurement technique of ToF cameras and thus it does not interfere with the visual field. A passive stereo technique is another alternative to point sample an object or scene. However, stereo processing algorithms depend on the presence of the texture in the image, and they have a few parameters that can be altered to generate a better result such as disparity range or correlation window size.

(a) (b)

(c) (d)

Fig. 5. Point cloud capture devices (a) Kinect, (b) Creative Senz3D scanner, (c) Trimble scanner, (d) NextEngine 3D.

The use of different sensor types (e.g., digital cameras, thermal cameras, multispectral cameras, range cameras, laser scanners, etc.) typically results in data in the form of 2D imagery or 3D point cloud data. Some examples [119] are shown in Fig. 6.



(a) (b)

(c) (d)

Fig. 6. Example of point cloud generation (a) using 2D laser sensor, (b) using Time-of-flight camera, (c) using Stereo camera [120], (d) Synthetically.

3.3 Applications of 3D Scanning

Three-Dimensional point clouds are widely used in various applications such as modeling, rendering, and CAD model generation. These point clouds are mainly generated using

3D scanners. Other applications include inspection and quality control, where a manufactured part is compared with its intended design CAD model. Numerical simulation using finite elements can be performed on the scanned models, e.g., the simulation of the aerodynamic flows inside and outside of an object. Also, scanned models are used in computer graphics to render realistic scenes and in the film animation industry.

However, although the scanning technology has improved and offered better features, it still has some problems such as distortion, reflections, shadows, low contrast, etc. Limitation of device precision, the influence of light, and reflection may cause the addition of noise in the original data, which damages the original representation of the model and also hampers the accuracy of the surface reconstruction.

3.4 Types of Noise in Point Clouds

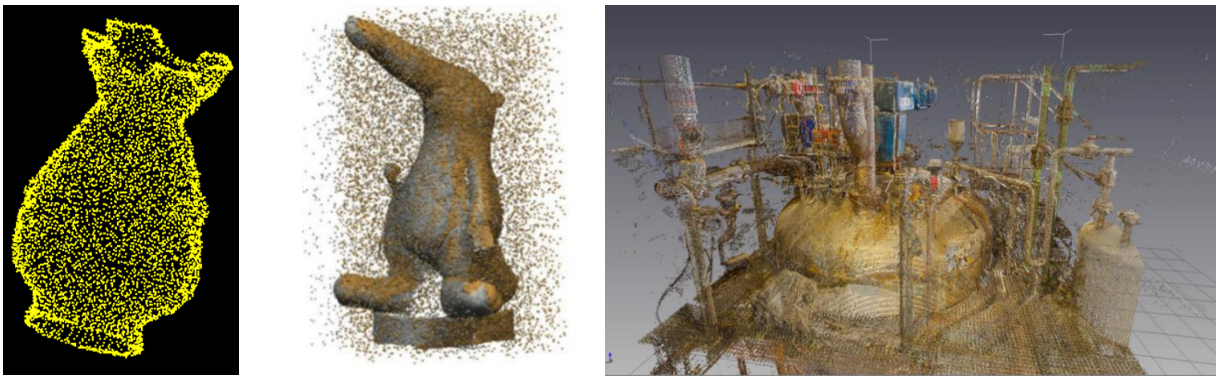
Point clouds generated from the scanners are not clean. Most of the data are incomplete, unclean, or contaminated by noise and missing important features. Several factors can cause noise to the original point cloud such as sensor noise, depth quantization, distance in relation to the scanner, etc. The noise can be of different types such as Gaussian, outlier, and shot (Fig. 7).

Gaussian noise: This type of noise is generated due to sensor imperfections during procurement. Generally, when the same scene is taken from different viewpoints or more than one camera is used in image acquisition, Gaussian noise is presumably added to the original point cloud. This noise affects the position of all points of the point cloud, and the level can be modeled by a standard deviation.

Outlier: This type of noise is generated due to structural artifacts in the acquisition process. Mostly, it happens during multiview stereo acquisition where view dependent

reflectivity of a surface can result in false correspondence. Sometimes, outliers are randomly distributed in the volume, and the density is much less than the sample density of the overall points.

Shot noise: This type of noise is produced because of the misjudgment of the scanner when it is scanning the boundary of the object. The scanner cannot locate the depth boundary of the model, and thus it generates some tail along the depth of the object. In most of the cases, the density is high and can't be separated from the original shape of the object using density estimation. Some individual points have extreme values.



(a) (b) (c)

Fig. 7. Types of noise in point cloud. (a) Gaussian noise, (b) outlier, (c) shot noise.

A point cloud has to be processed before generating a 3D mesh surface. A well-prepared point cloud leads to strong time saving in the further surface editing or modeling processes. The importance of noise removal in point cloud data is to generate a cleaner and smoother exterior of the original data with minimum topological error. For this reason, both noise removal and structural improvement are the objectives of this dissertation research.

3.5 Proposed Methods

In this section, five different filtering methods are proposed for point cloud processing. Among these five filtering methods, vector median, fuzzy vector median, adaptive mean, and adaptive median filter have been effectively used in image processing and mesh processing. These filters are extended and implemented for point cloud processing in this dissertation. The adaptive vector median filter is a new filtering method that is first proposed in this dissertation for the point cloud processing.

3.5.1 Vector Median Filter

The vector median filter is the extension of median filter [121]. Given an input point cloud $P = \{p_i \epsilon R^3\}$, and an observation window $\Omega = \{p_1, p_{2,\ldots..}, p_N \epsilon \boldsymbol{R}^m\}$, the output of the vector median filter is defined as [101]:

$$P_{VM} = arg \min_{p \in \Omega} \sum_{i=1}^{N} \|p - p_i\|_{L_p}, \quad (1)$$

where, p_1, $p_{2\ldots}$, p_N are input points, N is the window size and $\|.\|_{L_p}$ denotes the L_p norm. The sum of L_p is the total distance from each point to all other points. The vector median is a suboptimal estimate, in the maximum likelihood sense, of the location parameter of a multivariate Laplacian distribution. To find the vector median, the sum of L_p distances from each sample to all other samples is computed, $d(p_j) = \sum_{i=1}^{N} \|xp_j - xp_i\|_{L_p}$, $j = 1,2, \ldots\ldots\ldots N$, then, the vector median is set as $P_{VM} = \arg min_{p_j}(dp_j)$. Although this computation has a complexity of O(N^2), it performs well in practice and is not generally computationally prohibitive as the window size N is usually a small number. In addition, fast vector median methods are available [122].

3.5.2 Fuzzy Vector Median Filter

The concepts of fuzzy relations and fuzzy median filters to the vector data case are extended in this method [101]. The FVM filter is applied to the smoothing of surface normals and yields results that minimize the effect of noise while simultaneously preserving the fine structure, edges, and other visually important cues. Following the FVM-based smoothing of surface normals, the point positions are updated based on a system of linear equations structured on the smoothed normals using the least square error (LSE) method.

To utilize fuzzy membership functions on vector-valued data, an appropriate distance metric $D \in \mathbb{R}^m$ for vectors u and $v \in \mathbb{R}^m$ must be established. This metric must satisfy the following conditions:

1. $D(u,v) \geq 0$, and $D(u,v) = 0$ iff $u = v$,

2. $D(u,v) = D(v,u)$,

3. $D(u,v) + D(v,w) \geq D(u,w)$.

The distance metric $D(\cdot, \cdot)$ may be application dependent. For example, if the directions that vectors u and v represent are the main features of concern, then the angle between u and v is a good distance metric. Conversely, if the physical distance between u and v defines a feature, then the Lp norm is the appropriate metric. Although other metrics can be adopted, we restrict our focus to the commonly used angle and Lp norm metrics. The specific metric utilized will be clear from the context. The angle metric and Lp norm metric can be written as:

$$D(u,v) = \begin{cases} A(u,v) = \angle(u,v), \\ L_p(u,v) = \|u - v\|_p. \end{cases} \quad (2)$$

Given a vector distance metric, we now define a vector-based fuzzy membership function, denoted $\mu_{\tilde{R}}(u,v)$: $\mathbb{R}^m \times \mathbb{R}^m \mapsto [0,1]$, where the constraints are:

1. $lim_{D(u,v)\to 0}\mu_{\tilde{R}}(u,v) = 1$,

2. $lim_{D(u,v)\to M}\mu_{\tilde{R}}(u,v) = 0, where\ M = \sup D(u,v)$,

3. $D(u_1,v_1) \le D(u_2,v_2) \Rightarrow \mu_{\tilde{R}}(u_1,v_1) \ge \mu_{\tilde{R}}(u_2,v_2)$. .

The metric $D(\cdot,\cdot)$ can also be used to extend the Gaussian membership function to vector-valued data:

$$\mu_G(u,v) = e^{-D(u,v)^2/2\sigma^2}, \tag{3}$$

where σ is the spread parameter.

The vector median is based on the vector distance metric. The vector median, represented by $p_{(\delta)}$, is the sample that minimizes the distance metric D (. , .) between itself and all other samples:

$$P_{(\delta)} = \arg\min_{p\in\Omega} \sum_{i=1}^{N} D(P,P_i). \tag{4}$$

If more than $\frac{N+1}{2}$ samples have the same value p_m, then the vector median filter selects p_m as its output.

The fuzzy vector median filter is implemented in the following way: given an input point cloud $P = \{p_i \epsilon R^3\}$, a K-d tree is formed to represent the neighborhood information. A K-d tree, or K-dimensional tree, is a data structure used for organizing some number of points in a space with k dimensions [120]. Since point cloud is three-dimensional, K-d trees used here are also three-dimensional. K-d tree uses partition method to organize the number of points in a space. The final outcome is the weighted sum of input point sets, where the weights are determined by the fuzzy relation between each input vector and the vector median. The vector median is determined based on a distance metric. The output of the FVM is defined as [101]:

$$P_{FVM} = \frac{\sum_{i=1}^{N} P_i \tilde{R}_{i,(\delta)}}{\sum_{i=1}^{N} \tilde{R}_{i,(\delta)}}, \tag{5}$$

where $\tilde{R}_{i,(\delta)} = \mu_{\tilde{R}}(P_i, P_{(\delta)})$ is the fuzzy relation between P_i and $P_{(\delta)}$. P_i is the input point data set

and $P_{(\delta),}$ is the median. The relation function can be any shape that reflects the most relevant information between samples. Two identical samples should have relation 1, while the relation of two infinitely distant samples should be 0. Moreover, the relation between samples should increase as the distance between them decreases.

3.5.3 Adaptive Filters

Adaptive filters are widely used in the image-processing domain for their capability to enhance the eminence of the images and remove the unwanted pixels that cause the degradation of the image. The most important characteristic of these filters is that the filter can self-adjust some of its property during the filtering process based on some criteria. Adaptive filters perform better than mean and median filters. The adaptive filters exhibit significant improved performance in image processing if the image contains outliers, shot, or Gaussian noise. Since the filtering operation is performed based on the local characteristics of the image, it can keep the small details and enhance the edges of the image.

3.5.3.1 Adaptive Mean Filter

The adaptive mean filter changes its behavior according to the statistical characteristics of the point cloud inside the filter window S_{xyz} with a specified radius [123]. Given an input point cloud $P = \{p_i \epsilon R^3\}$, a K-d tree is formed to represent the neighborhood information. For this filtering approach, the algorithm is applied to the neighborhood of a point. The neighborhood is defined by the window $S_{xyz} = \{p_{ij} \epsilon P\}$. Four parameters are considered here: the depth value of the noisy point p_i, the variance of the noise σ^2_n corrupting the original points, the local mean of the points m_L in the region S_{xyz} and local variance of the points σ^2_L in the region. The behavior of

the filter is as follows:

First, in a specific window compute the local mean, local variance of the depth values of that region and the variance of overall noise.

i) If $\sigma^2_n = 0$, the filter should return the value of the centered point. This happens when there is no noise present.

ii) If σ^2_L is high relative to σ^2_n, the filter should return values close to the point under consideration. A high local variance means it is related to the edges, and these should be preserved.

iii) If $\sigma^2_L = \sigma^2_n$, the filter should return the arithmetic mean value of the points in the region S_{xyz}. The local noise is reduced by simple average.

According to the preceding assumptions the filter response can be modeled as:

$$P_{AM} = Z_i - \frac{\sigma_\eta^2}{\sigma_L^2}[Z_i - m_L] \quad (6)$$

If $\sigma^2_\eta > \sigma^2_L$ then the ratio is set to one. Here, $Z_i$represents the depth value of a point, σ_η^2 represents the variance of overall noise, σ^2_L represents the local variance of the local region and m_L represents the local mean.

3.5.3.2 Adaptive Median Filter

As an adaptive filter, adaptive median filter also changes its behavior based on the statistical characteristics of the point cloud inside. Given an input point cloud $P = \{p_i \epsilon R^3\}$, a K-d tree is formed to represent the neighborhood information. The neighborhood is defined by the window $S_{xyz} = \{p_{ij} \epsilon P\}$. However, it changes the size of S_{xyz} during filter operation, depending on the following conditions. The filter works in two stages, denoted stage S_1 and stage S_2:

Stage S_1: $S_{11} = Z_{med} - Z_{min}$

$S_{12} = Z_{med} - Z_{max}$

if $S_{11} > 0$ and $S_{12} < 0$, go to stage S_2

else increase the window size

if window size $\leq S_{max}$ repeat stage S_1

else output Z_{med}

Stage S_2: $S_{21} = Z_i - Z_{min}$

$S_{22} = Z_i - Z_{max}$

If $S_{21} > 0$ and $S_{22} < 0$, output Z_i

else output Z_{med}

Here, Z_{med} = median of depth value in S_{xyz}, Z_{min} = minimum depth value in S_{xyz}, Z_{max} = maximum depth value in S_{xyz}, Z_i = depth value of point P_i, S_{max}= maximum allowed size of S_{xyz}.

3.5.4 Adaptive Vector Median Filter

The resulting 3D point cloud of a real object often contains noise-induced artifacts, which are typically located around the ends and border of the model. These noise-induced artifacts are unwanted and feature in the point cloud as clusters of neighboring points, which are not actually part of the original model surface. In other words, the outliers are the product of the sensor's inaccuracy, which registers measurements where there should not be any. The adaptive median filter attempts to preserve detail while smoothing the impulse noise and outliers in point cloud. The adaptive vector median filter is based on the spatial processing of the point cloud. The first step is to find a neighborhood for each point. An adaptive structure of the filter ensures that most noisy points are detected even at a high noise level.

Given an input point cloud $P = \{p_i \epsilon R^3\}$, a local neighborhood $S_{xyz} = \{p_{ij} \epsilon P\}$ for each point p_i is determined by the KNN (K-Nearest Neighbor) where p_{ij} is the jth neighbors around p_i and a 3D kd-tree representation is constructed for P_c. The point containing the vector median (vector median is calculated based on distance) in S_{xyz} is defined as p_j. This filter detects the noisy candidate p_i and replaces the noisy candidate with the vector median of the points in a local window. However, it changes the size of S_{xyz} during filter operation, depending on the following conditions.

The algorithm checks both the point of interest and the point containing the vector median. Four different situations may arise in detecting noise in the point cloud.

1. The point of interest p_i is noisy,

2. The point containing the vector median p_j is noisy,

3. Both the p_i and p_j is noisy,

4. None of them are noisy.

Given a noisy point cloud and an initial window size, the adaptive vector median filter performs several steps.

Stage 1: First, for each specified window, it calculates the vector median. Next, it checks if the point containing the vector median value p_j is noisy based on the depth value (z component) using the following formula:

$$Z_{min} \leq Z_{med} \leq Z_{max} \tag{7}$$

where Z_{min} is the minimum of depth value in S_{xyz}, Z_{med} is median of depth value in S_{xyz} and Z_{max} is the maximum of depth value in S_{xyz}. If p_j is not noisy (eq. 7 is satisfied), then it continues to stage 2. Otherwise it expands the window size and repeats stage 1.

Stage 2: Check if the center point p_i is noisy by the following formula:

$$Z_i - Z_{min} > 0 \; and \; Z_i - Z_{max} < 0 \tag{8}$$

where Z_i is the depth value of p_i, Z_{min} is the minimum of depth value in S_{xyz} and Z_{max} is the maximum of depth value in S_{xyz}. If the condition satisfies then p_i is not noisy, the filter output is the original center point, and it continues to the next point, otherwise p_i is replaced by the vector median p_j; If both the vector median p_j and the center point p_i are noisy, the filter window is expanded, and the above process is repeated.

A flowchart of the proposed filtering method is illustrated in Fig. 8.

Fig. 9(a) shows a noisy model with several outliers and Gaussian noise with σ = 0.001 (m) and Fig. 9(b) illustrates the result of the filtering. If the noise candidates are detected, the denoising performance is expected to be improved. The adaptive vector median filter is considered to remove the difficulties faced by the standard vector median filter. The basic difference between vector median and adaptive vector median filter is that, in the adaptive vector median filter, the size of the window adjacent to each point is adjustable. This change of window size depends on the vector median of the points in the present window. If the vector median value is between the max and min value, then the size of the window is extended. Otherwise, further processing is ended on the part of the data within the current window specifications. So far, we only used a fixed maximum radius to compute the local neighborhoods for detecting the window size. Fig. 9 shows a 3D point cloud of a sphere with artificially added Gaussian noise and outlier. This point cloud has some non-isolated outliers that are not separable using simple distance criterion.

The AVM filters out the outlier based on the window-based technique and successfully retrieve the original shape of the sphere. Fig. 10 represents an artificial iron model in 3D where Fig. 10(a) shows a noisy point cloud with Gaussian noise with σ = 0.001 (m) and several outliers

and Fig. 10(b) illustrates the result of our filtering. The result shows the proposed method maintains the better quality of the original model with precise features. We represent this comparison to illustrate the effectiveness of the proposed method for removing outliers and noisy points and improving the prominent features of this complex dataset. The vector attribute of the point cloud and the adaptive nature of the window size handles the noise efficiently. The points in the point cloud are not removed permanently; rather the positions are updated according to the algorithm.

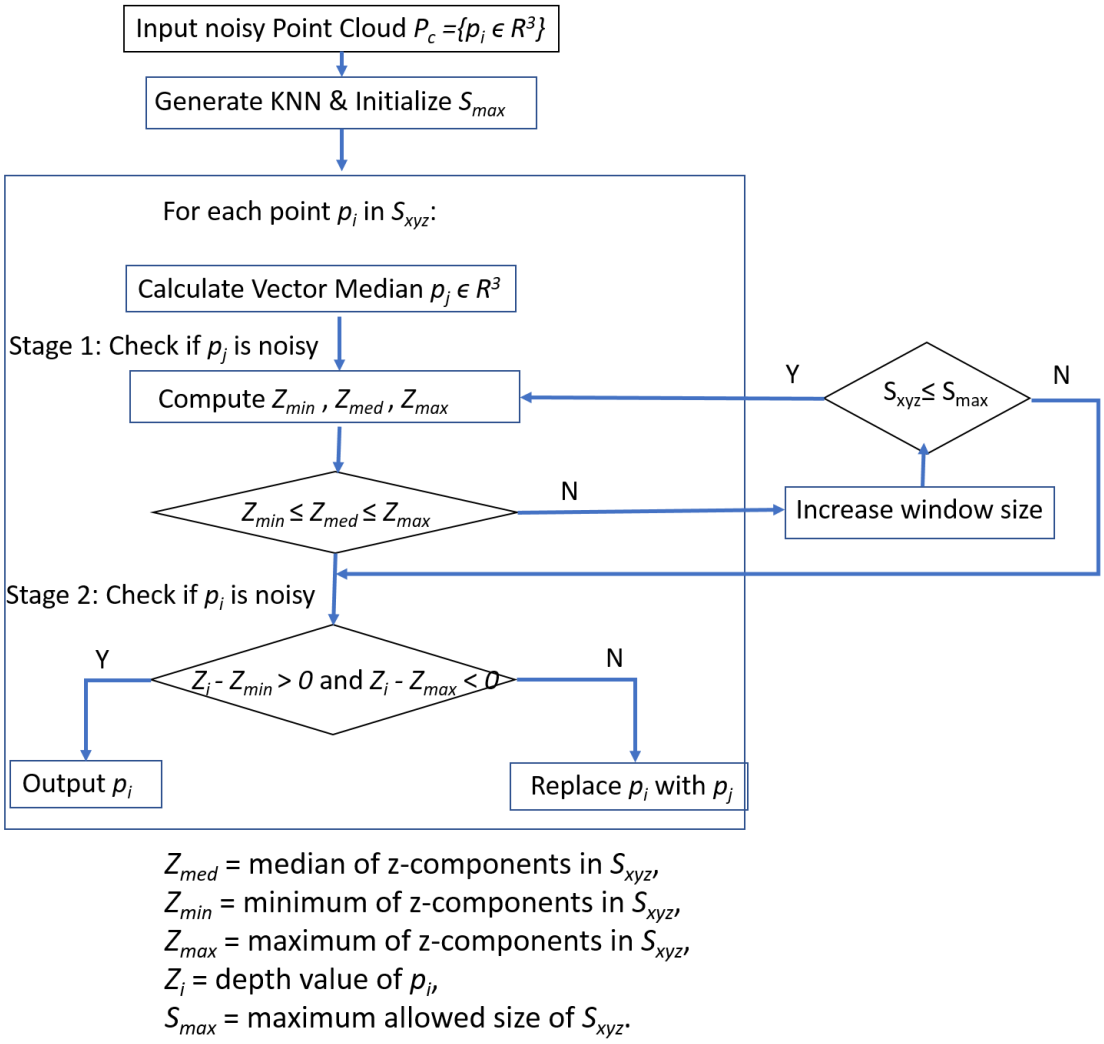


Fig. 8. Flowchart of Adaptive vector median filter.

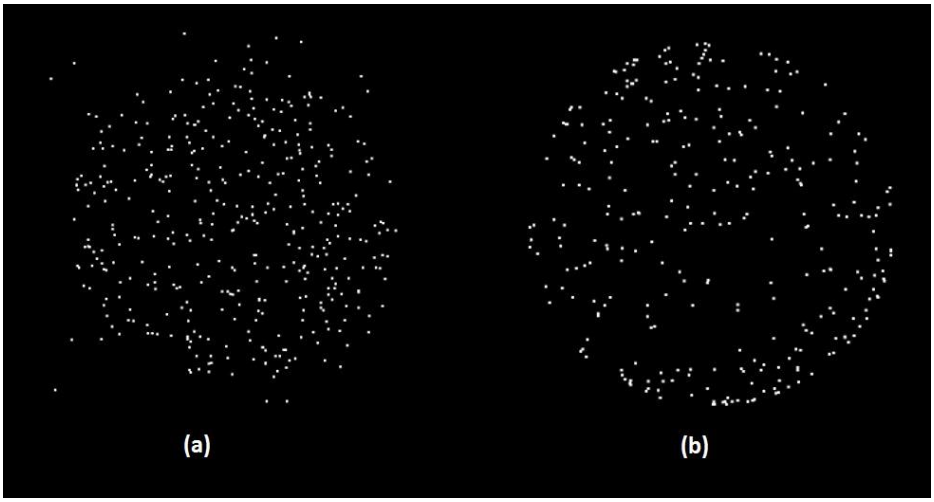


Fig. 9. Point cloud processing. (a) Point cloud of an artificial sphere with a high density of noise (b) Result after noise removal using AVM.

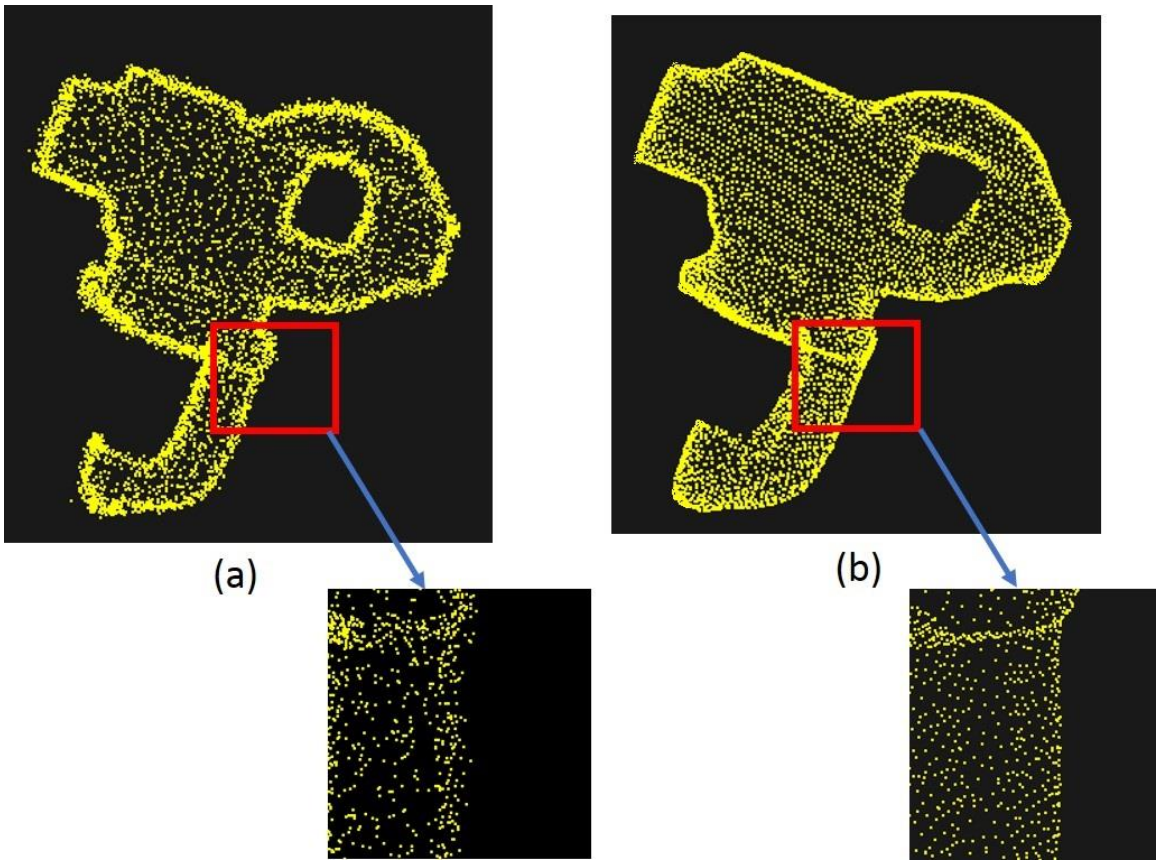


Fig. 10. Denoising point cloud (a) Input noisy model, (b) Filtered result using AVM.

3.6 Software Implementation and Experimental Results

This section presents the implementation of the various filtering methods for point cloud processing as well as their results for different data sets.

3.6.1 Software Implementation

The Point Cloud Library (PCL) is an open-source software library [124] for 2D/3D image and point cloud processing, developed by contributors from many different academic and commercial organizations. It contains a large collection of software library modules for various tasks in point cloud processing, such as filters, features, registration, kdtree, octree, segmentation, recognition, and visualization. PCL is open source software written in C++ and released under BSD license; it is free for commercial and research use. PCL is utilized as a basic framework to implement the proposed filtering methods for point cloud processing in this dissertation.

In addition, an application with a graphical user interface (GUI) was developed for the selection of filters, setting filter parameters, file input/output, and visualization of point clouds. The GUI was developed using the QT library, which is a cross-platform application framework and widget toolkit for generating graphical user interfaces. Fig. 11 shows the available functionality of the application and its GUI. MATLAB was used to add Speckle noises to the models. Kinect, Kinect 1.0, NextEngine 3D scanner, LIDAR data and other sources of scanned data were used to generate the models for results. A Dell Precision M6600 of Intel Core i7 processor with 16GB RAM has been used to execute the methods.

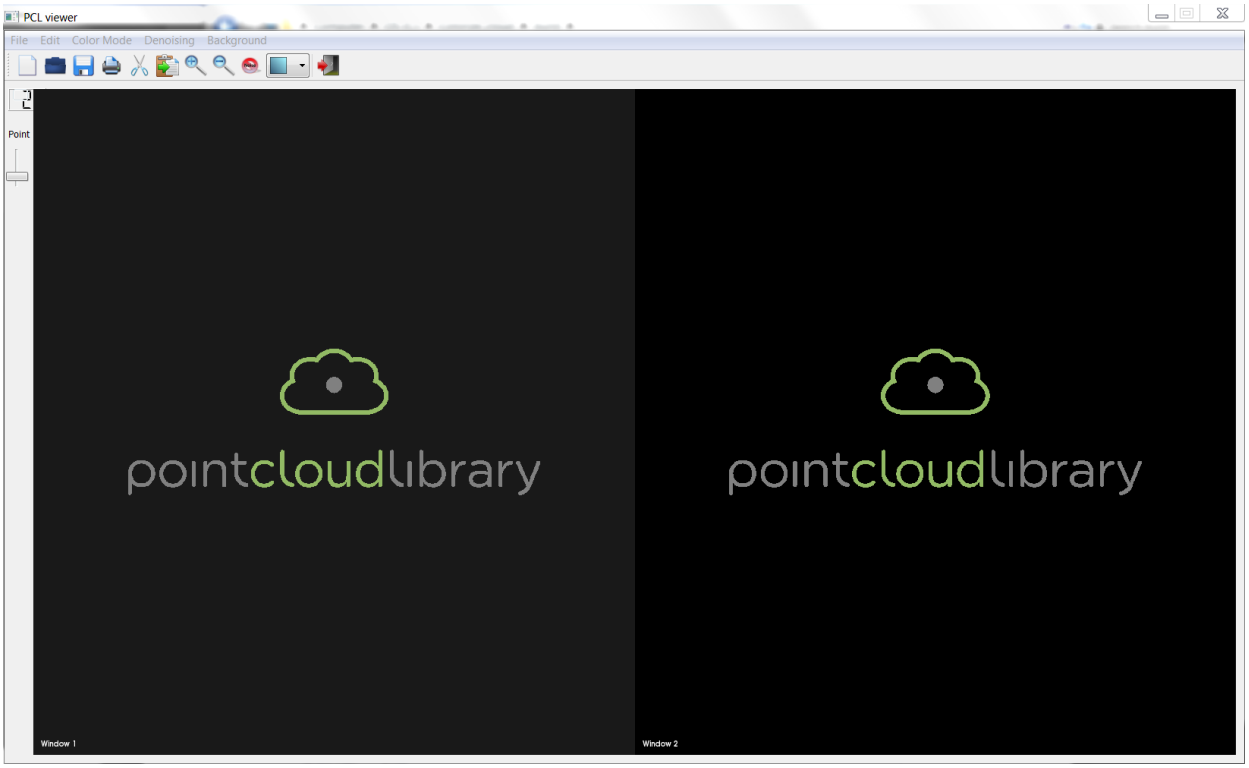


Fig. 11. Graphical User Interface for point cloud visualization.

A close view of the menu buttons of the interface is shown in Fig. 12. The menu has several options such as new workspace, open a file, save a file, print, cut, paste, zoom in, zoom out, denoising filter options (median, vector median, fvm, adaptive mean, adaptive median, avm, sor, ror), different view mode, and exit. Also, the size of the point cloud can be adjusted using the sliding bar. The available size is from 1 to 5. 1 represents the smallest, while 5 presents the largest in point size. Two viewports are provided to compare the input and output point clouds. The pan, rotation or move can be done by the mouse. The background color can be changed black, white, pink and cyan.

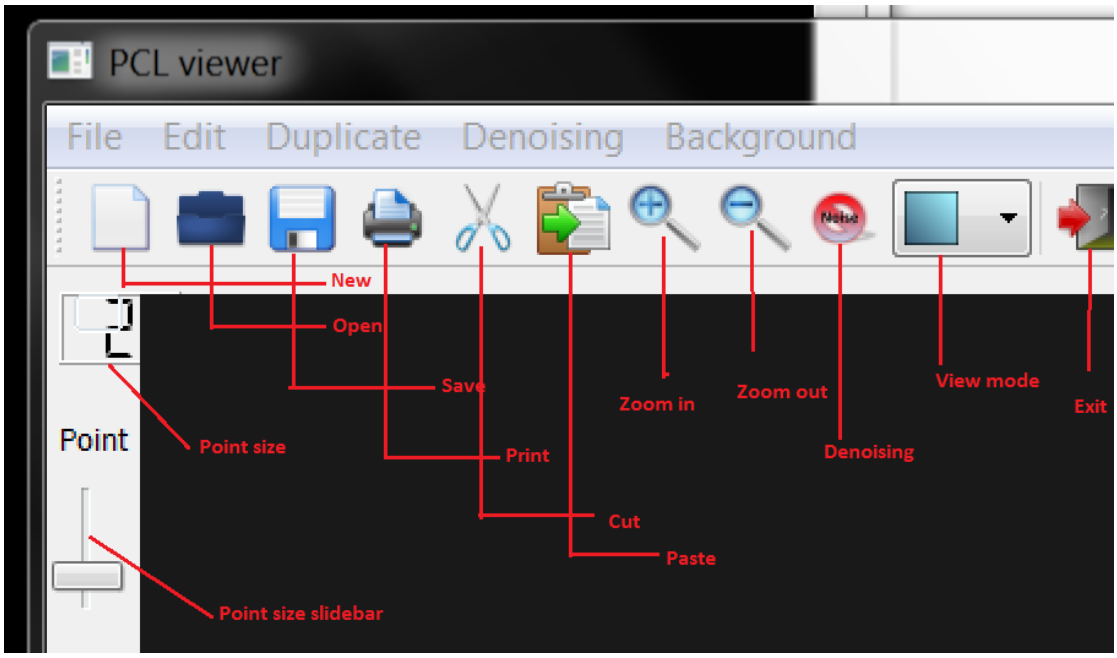


Fig. 12. Menu Items in the interface.

We have tested our approaches with several models generated from different sources. Several artificial models were generated using PCL such as Sphere, Cube, etc. Also, some existing point cloud models of known objects are used here. Different types of noises such as Gaussian, Shot and Salt and Pepper noise are added to the synthetic models for the evaluation.

3.6.4 Experimental Results

The various point cloud-processing methods developed in this dissertation have been applied to a variety of real-world and synthetic data sets. The real-world data sets have been captured as multiple depth maps with Microsoft Kinect (Fig. 5(b)), created with photogrammetric reconstruction from multiple images, or acquired with a laser scanner (Fig. 5(c)). Several artificial models were generated using PCL such as Sphere, Cube, etc. Also, some existing point cloud models of known objects are used here. Different types of noises, such as

Gaussian, Shot, and Salt and Pepper noise are added to the Synthetic models for the evaluation. Two categories of models are evaluated and included in this dissertation, namely, synthetic models and real scenes model captured by Kinect or Cyberware 3030 MS scanner. For the real scene model, several well-known high-density point sets from the literature, such as the Stanford Bunny and Happy Buddha [125] are included. To validate that the proposed algorithms can perform well even in the presence of high density of noise, we have added additional noise to the real scene models.

Category 1: Synthetic Models

Fig. 13 shows an artificial sphere with added outlier and Gaussian noise with standard deviation σ of 0.003 (m). The median filter removes most of the noise except few outliers. Vector median and FVM performs almost the same. On the other hand, adaptive mean, adaptive median and adaptive vector median shows a similar result for this model. Most of the noise is eliminated perfectly with these methods. It is worth noting that the outlier is detected first by the KNN (distance-based/nearest neighbor/radius-based approach). For the distance-based approach, the points that are significantly far away from the center of the dataset are considered outlier and removed from the dataset. In the nearest neighbor-based approach, points that don't meet the criteria of having specific numbers of neighbors are removed. Finally, for the radius-based approach, the user specifies the radius and the points that don't belong to the radius are not considered as the dataset. The proposed methods are applied after this step. This step is effective whenever outlier is present.

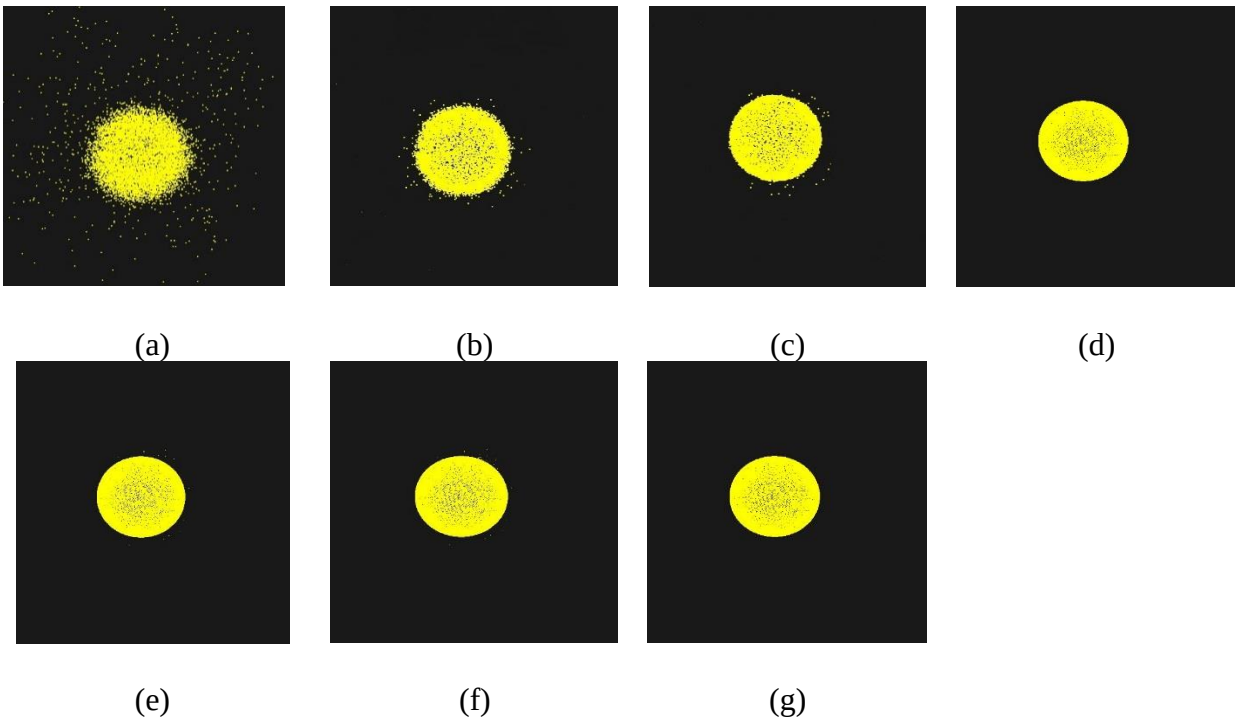
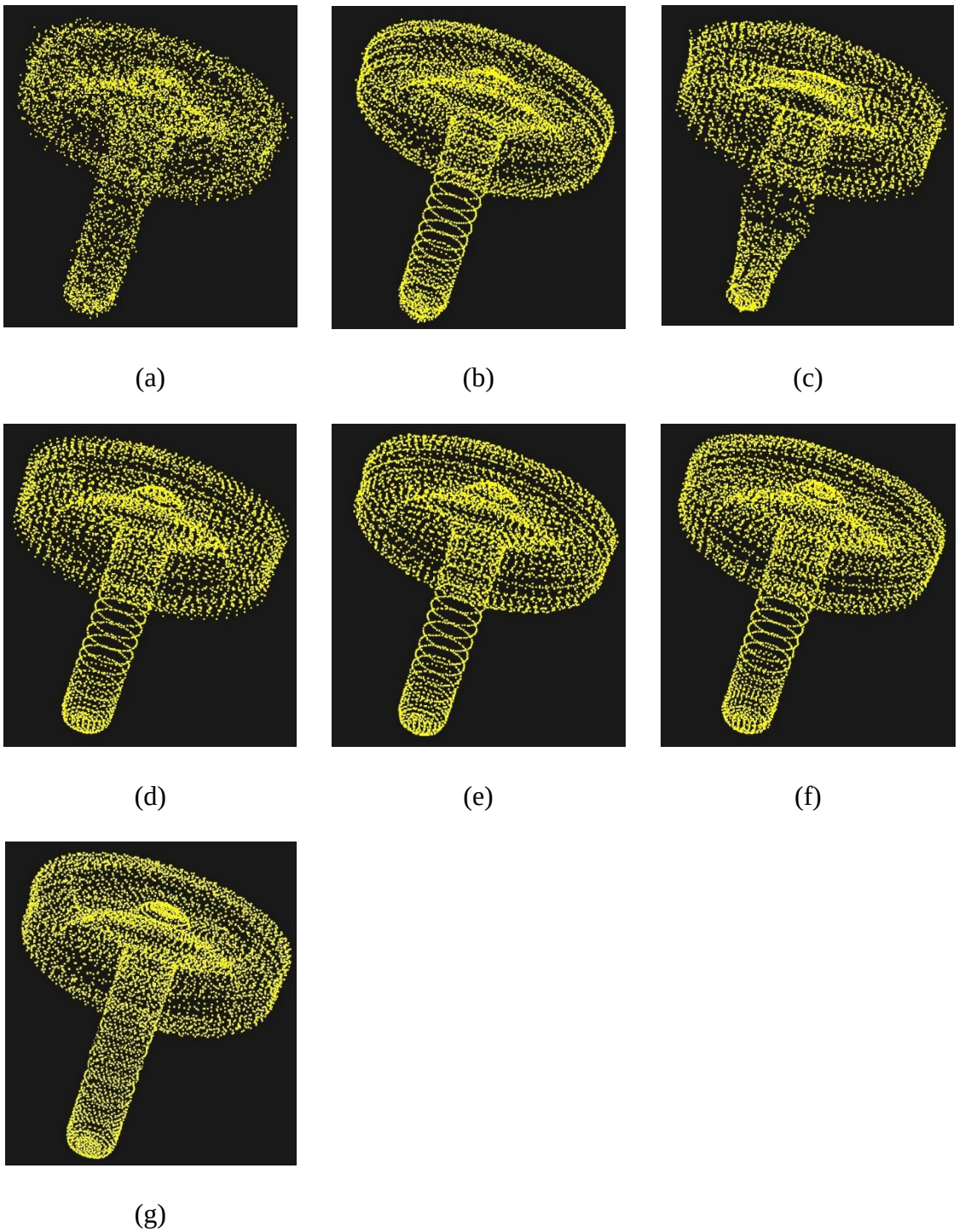


Fig. 13. Artificial model (Sphere). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

A representation of a synthetic gear model is shown in Fig. 14. Gaussian noise with 0.003 (m) standard deviation was added to the original model. All of the methods except vector median improved the noisy model to some extent. For the vector median filter, the structure of the gear model is distorted, and the handle lost its proper shape. However, median filter, FVM, adaptive mean, adaptive median, and adaptive vector median removed the noise and kept the structure intact.



(a) (b) (c)

(d) (e) (f)

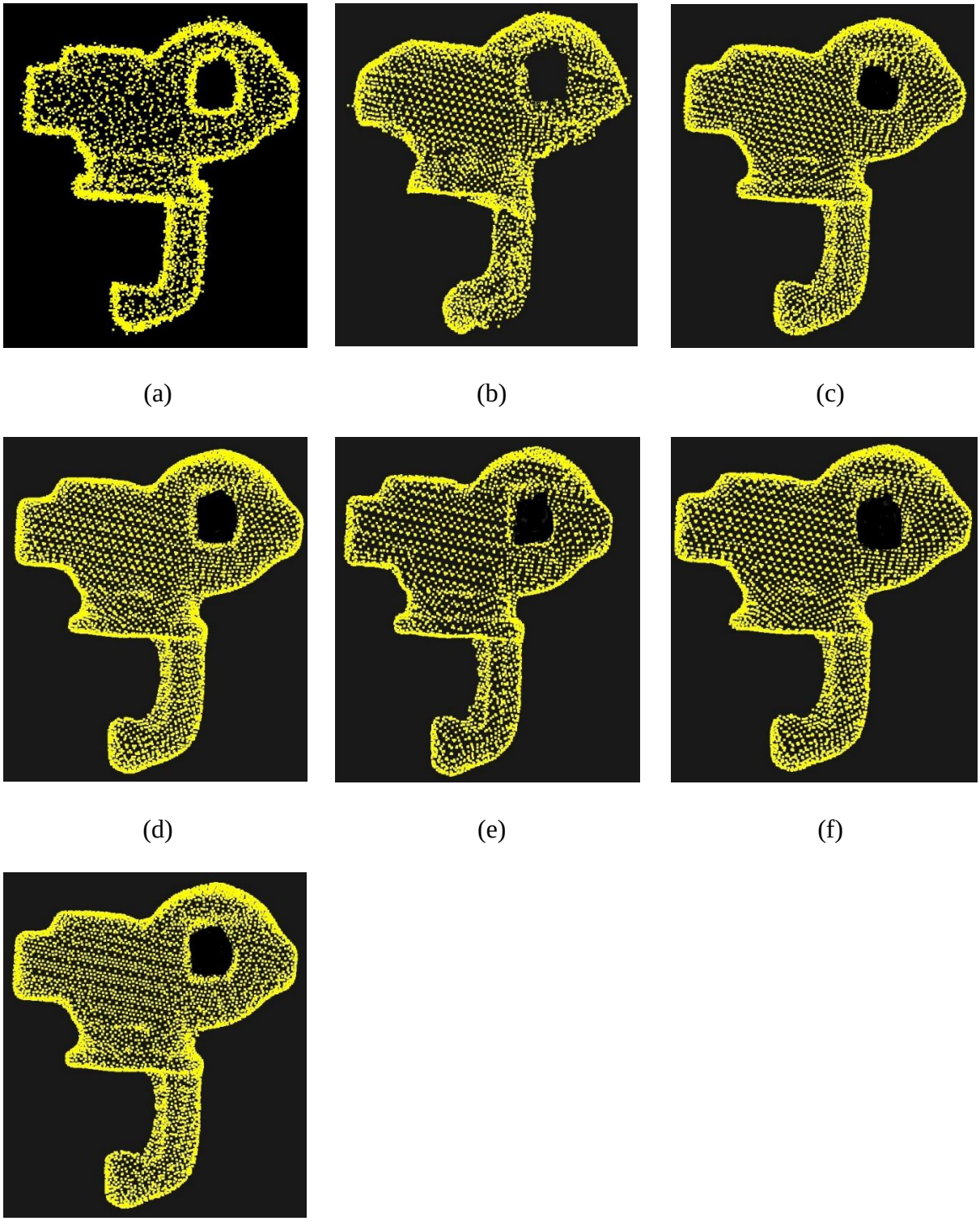
(g)

Fig. 14. Standard model (Gear). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

The third example is an iron structure (Fig. 15) with a hole in it. The original model was contaminated with Gaussian noise with standard deviation $\sigma = 0.002$ (m) in this figure. For this example, median, vector median, adaptive mean, adaptive median, and adaptive vector median performs almost similar. The noise is removed, the points look sharper, and the edges look prominent. The median filter produced results worse than other filters.

Another artificial model of a modified torus was used (Fig. 16). As can be seen, vector median distorted the model a little bit while median and adaptive median filter worked better but smoothed the sharp edge of the model. On the other hand, FVM, adaptive mean and adaptive vector median removed the noise and also kept the important feature (edge) intact.

An artificial model Sharp Sphere is illustrated in Fig. 17. The structure of the model has a complex construction. Some point inside the model has a lower density than the outer portion of the model. Additionally, a number of outliers are added to the model to demonstrate the effectiveness of the methods. Due to the outlier, the overall orientation of the model was distorted. Median filter and vector median filter could not denoise the noisy points properly. FVM, adaptive mean and AVM showed almost similar results. Adaptive median could not denoise the model properly but showed a better result than median or vector median filter.



(a) (b) (c)

(d) (e) (f)

(g)

Fig. 15. Standard model (Iron). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

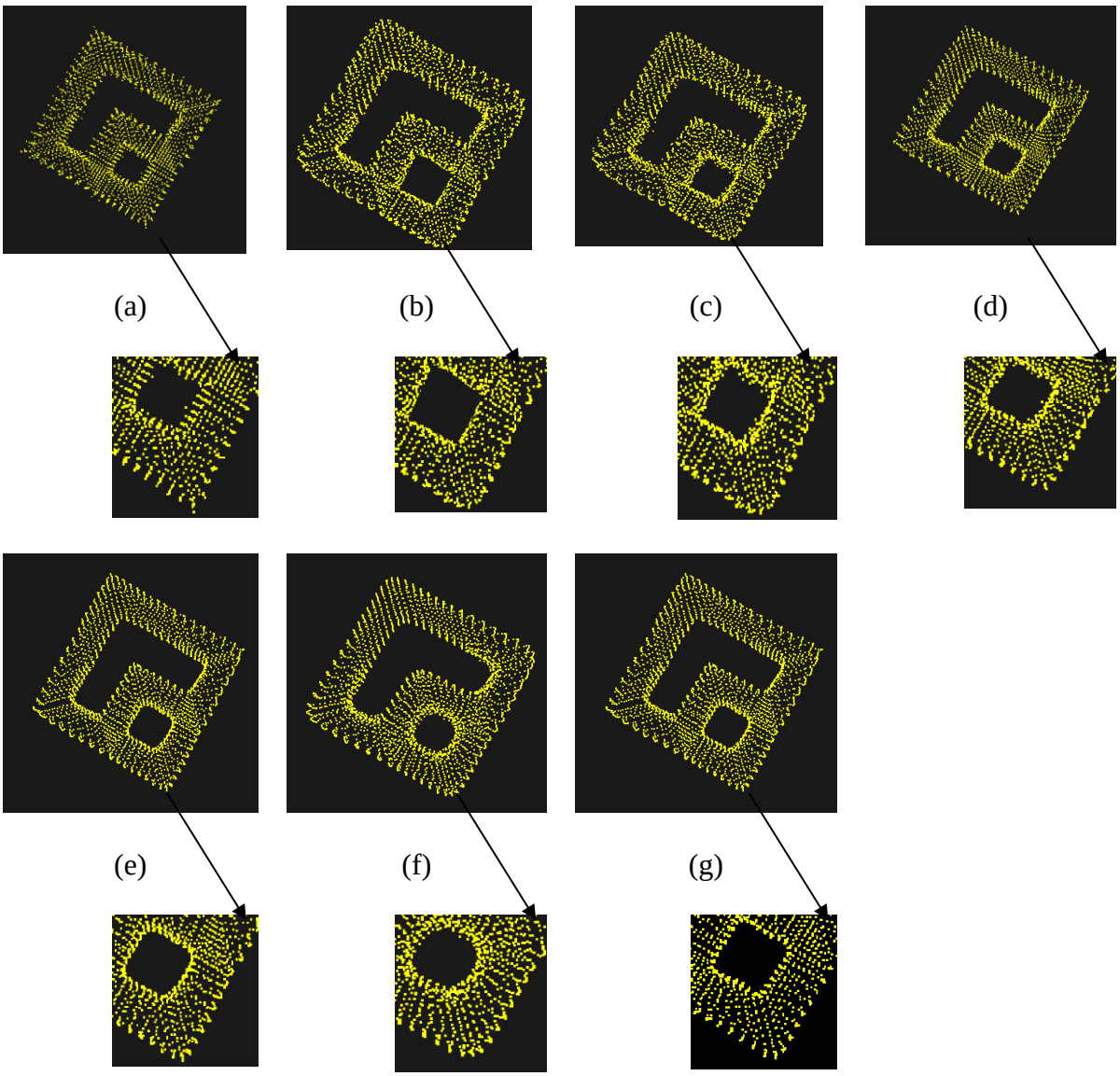
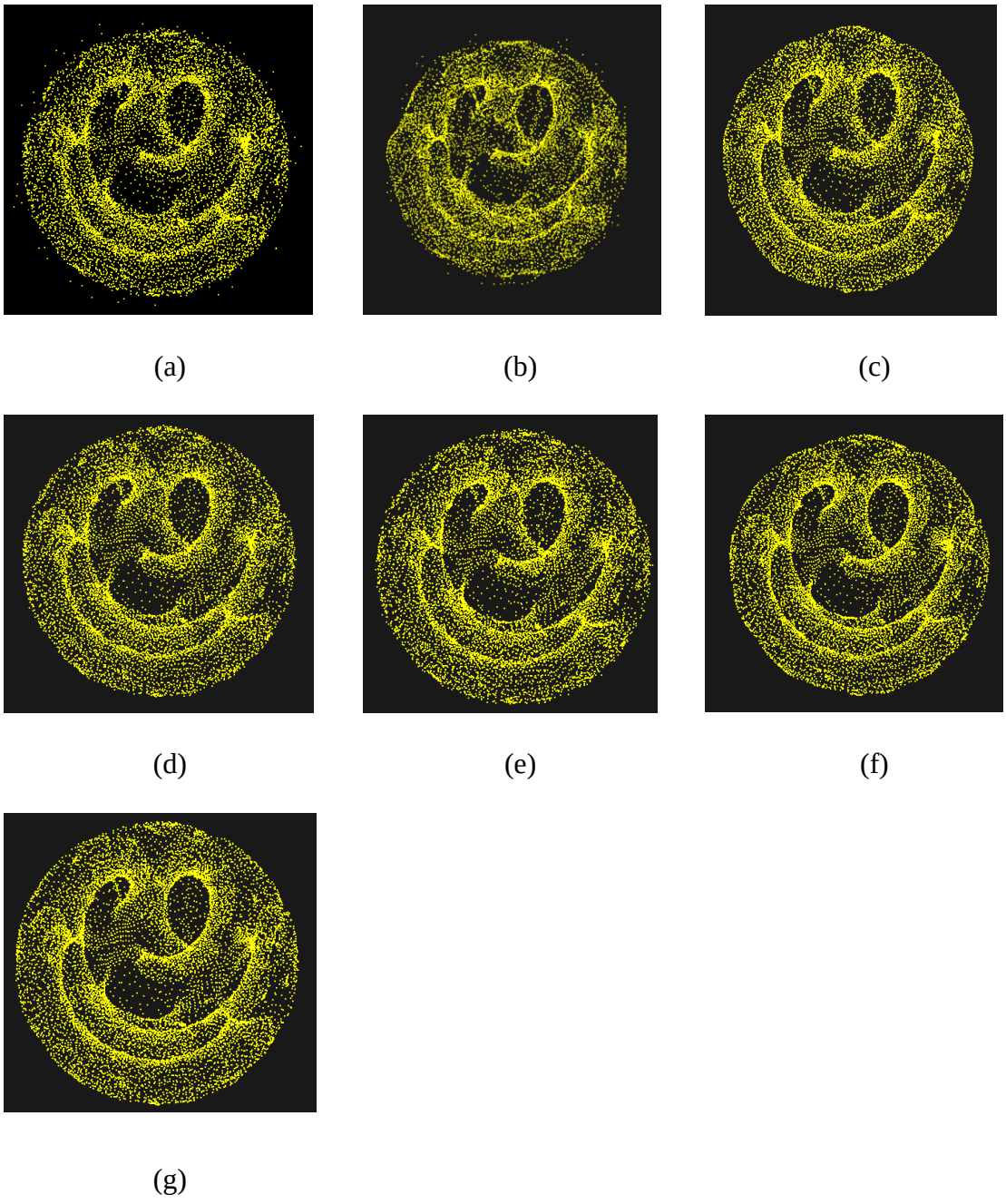


Fig. 16. Artificial model (Torus). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.



(a) (b) (c)

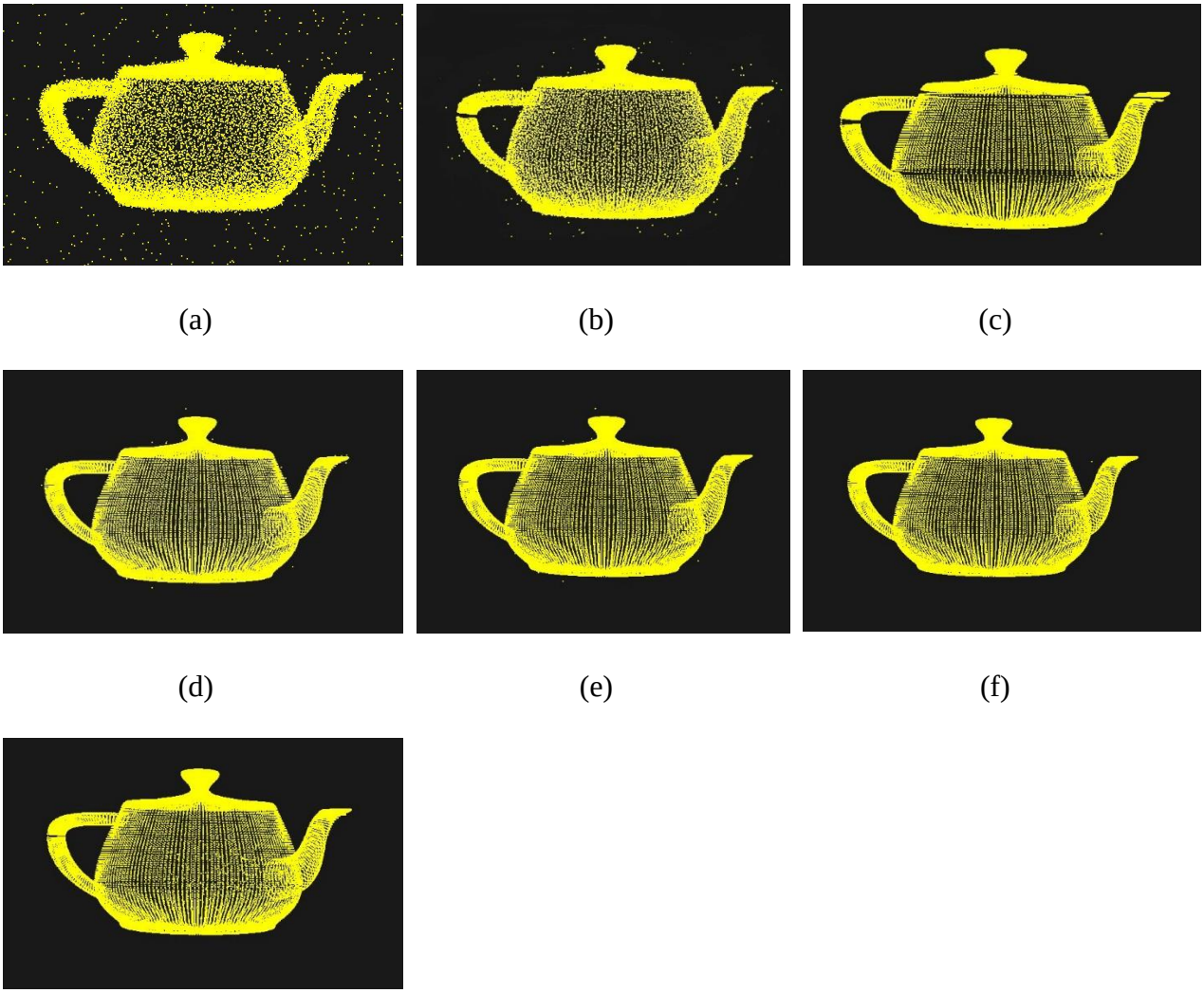
(d) (e) (f)

(g)

Fig. 17. Artificial model (Sharp Sphere). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

Some 3D models are very widely used in the computer graphics community, such as the teapot, fandisk, and bunny. Gaussian noise with $\sigma = 0.001$(m) was added in the following

examples. From Fig. 18 it is easily seen that the noise was not properly removed by the median filter. Also, the position of the upper part of the teapot was slightly distorted. The middle portion and the handle of the teapot lost some of the points after applying the vector median filter. However, FVM, adaptive mean, adaptive median, and adaptive vector median removed most of the noise and kept the structure of the teapot unchanged.



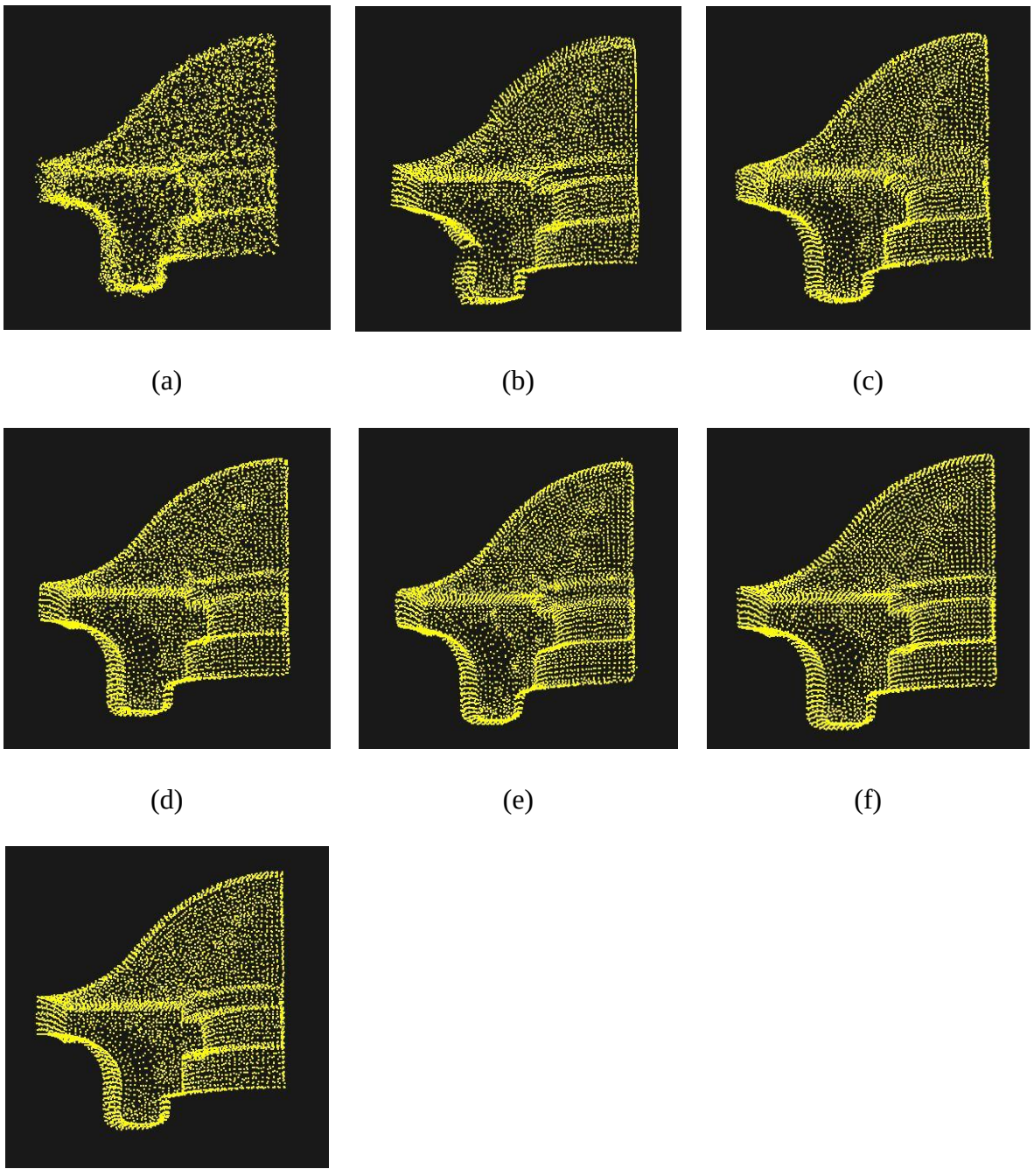
(a) (b) (c)

(d) (e) (f)

(g)

Fig. 18. Standard model (Teapot). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

Another model included here is a fandisk (Fig. 19). For this model, median filter removed the noise but additionally removed some of the points from the original model also. On the other hand, vector median. FVM, adaptive mean, adaptive median and adaptive vector median performs well. Here, the noise level is Gaussian with standard deviation $\sigma = 0.003$ m.



(a) (b) (c)

(d) (e) (f)

(g)

Fig. 19. Standard model (Fandisk). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

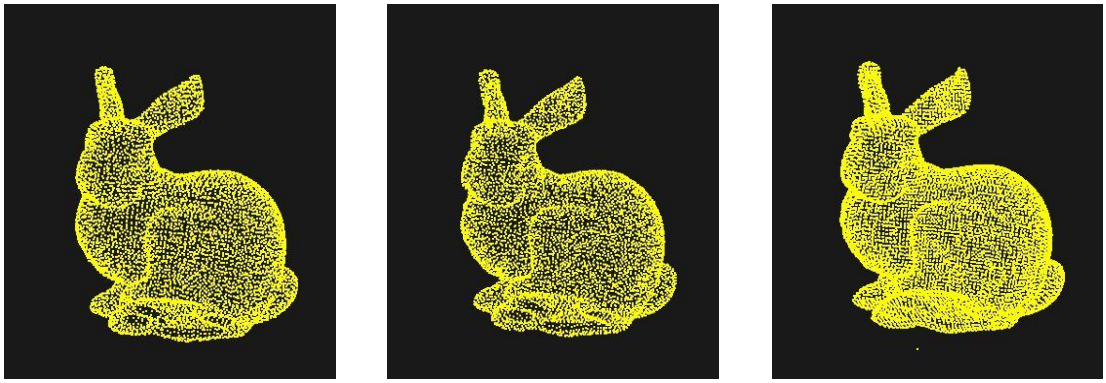
Category 2: Real Scene Models

The first example is the famous Stanford bunny (Fig. 20), which might be the most widely used model in the computer graphics community. The median filter distorted some of the points and removed a portion of the back of the bunny. Also, some outliers on the edge of the bunny were not removed properly. The vector median removed most of the outlier and Gaussian noise, but few noises remain at the border of the edge of the ear and feet. FVM, adaptive mean, adaptive median, and adaptive vector median performed pretty well. These methods removed noise as well as kept the important features of the bunny. Here, the noise level is Gaussian with standard deviation $\sigma = 0.003$ (m) and contaminated by thousands of outliers.

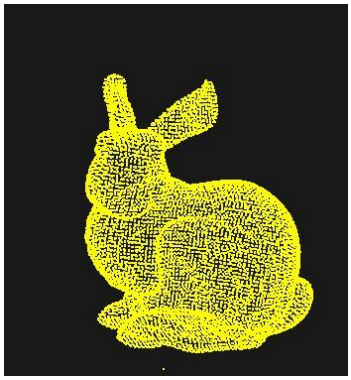
Fig. 21 is a scanned version of an angel. As can be seen, all of the methods perform well to some extent. The noise is removed and the sharp features especially the edges of the model are well preserved.

Fig. 22 is a scanned model of a happy Buddha. Gaussian noise with standard deviation $\sigma = 0.001$(m) was added to the original model. Median filter removed most of the noise but could not preserve details of the model. Vector median, adaptive mean and adaptive median performed almost similar in removing noise. Adaptive vector median filter removed the noise and preserved the small details to some extent. The edges are more prominent in this filtering approach.

(a) (b) (c)

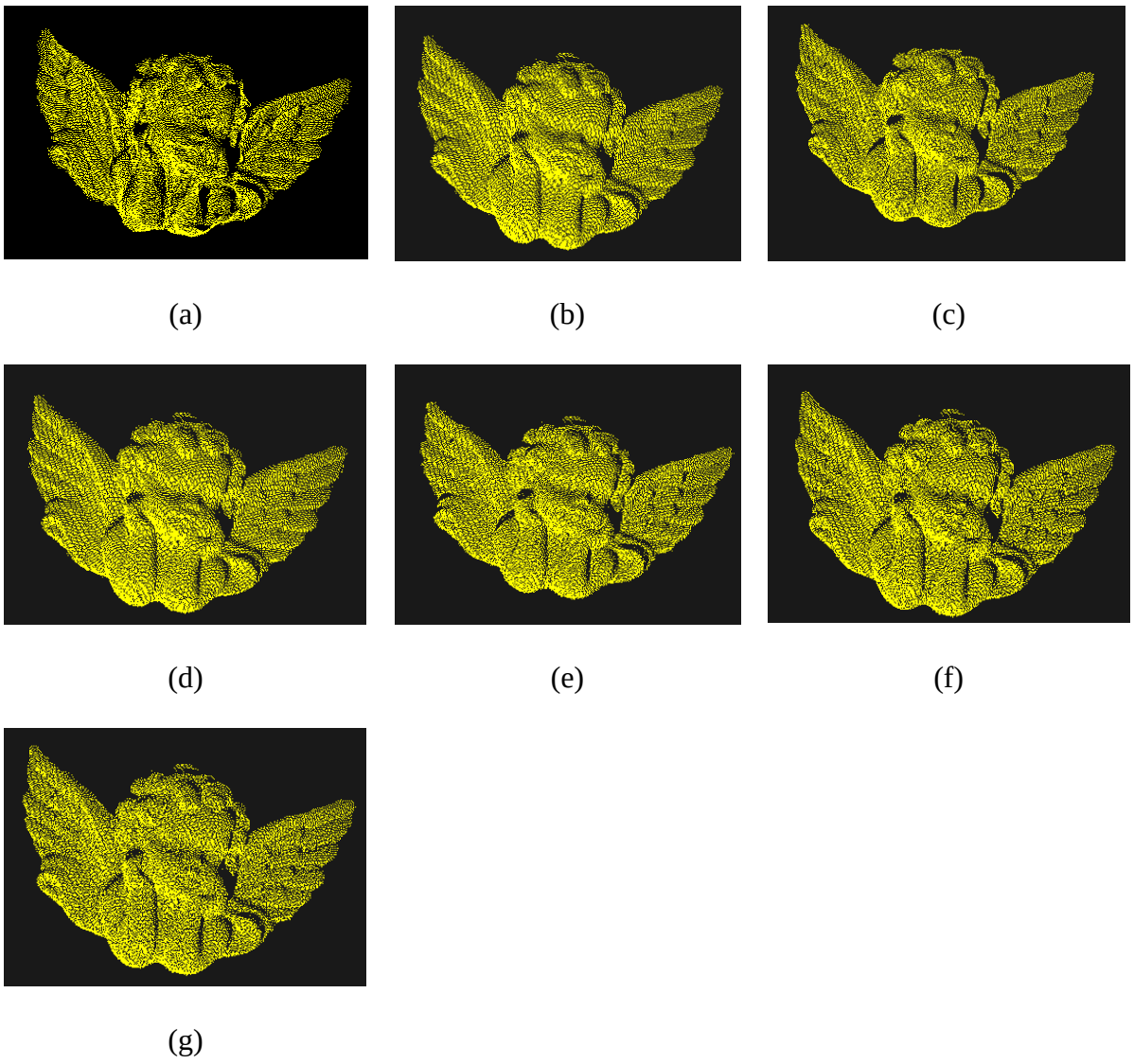


(d) (e) (f)



(g)

Fig. 20. Standard model (Bunny). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

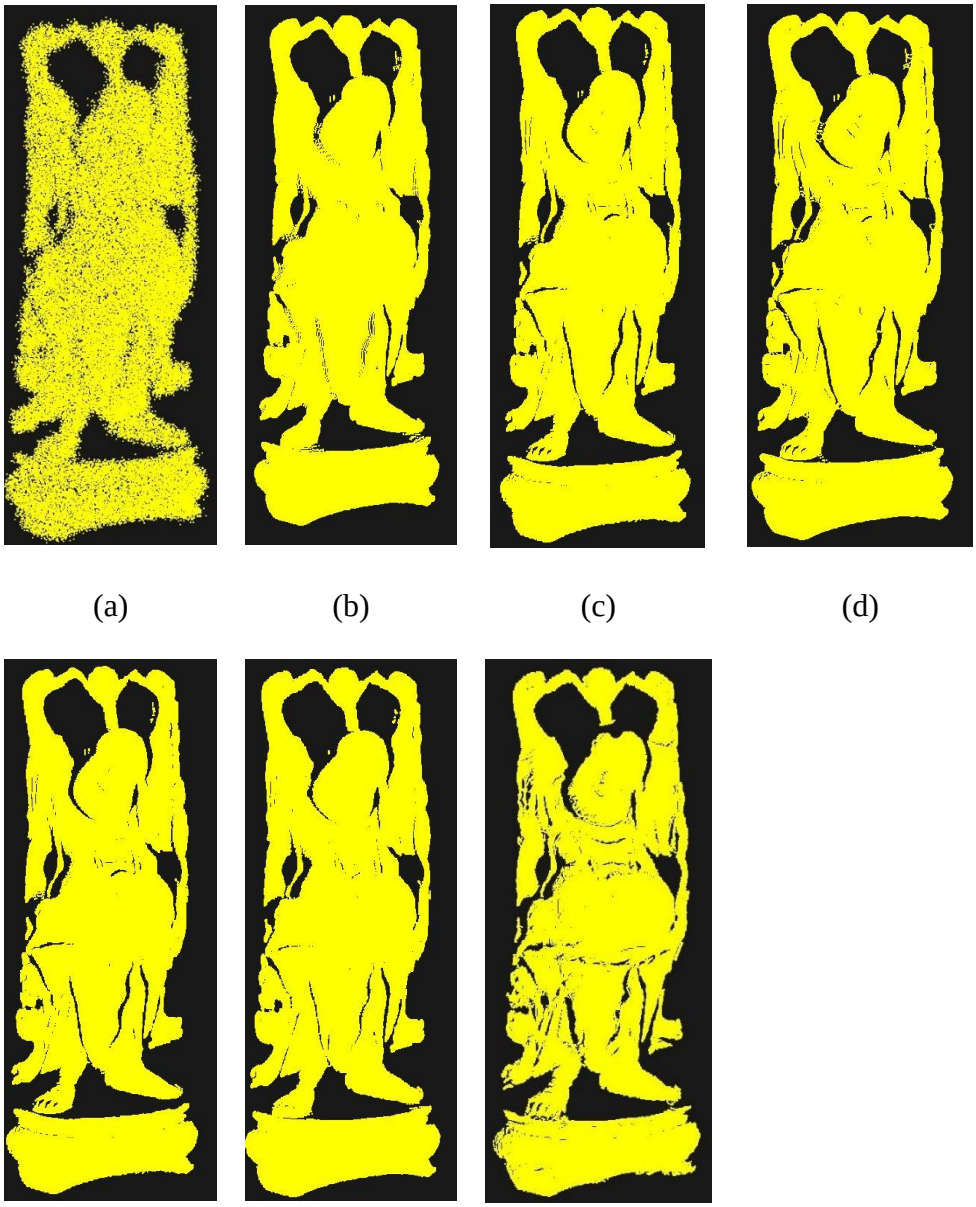


(a) (b) (c)

(d) (e) (f)

(g)

Fig. 21. Real Scene model (Angel). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.



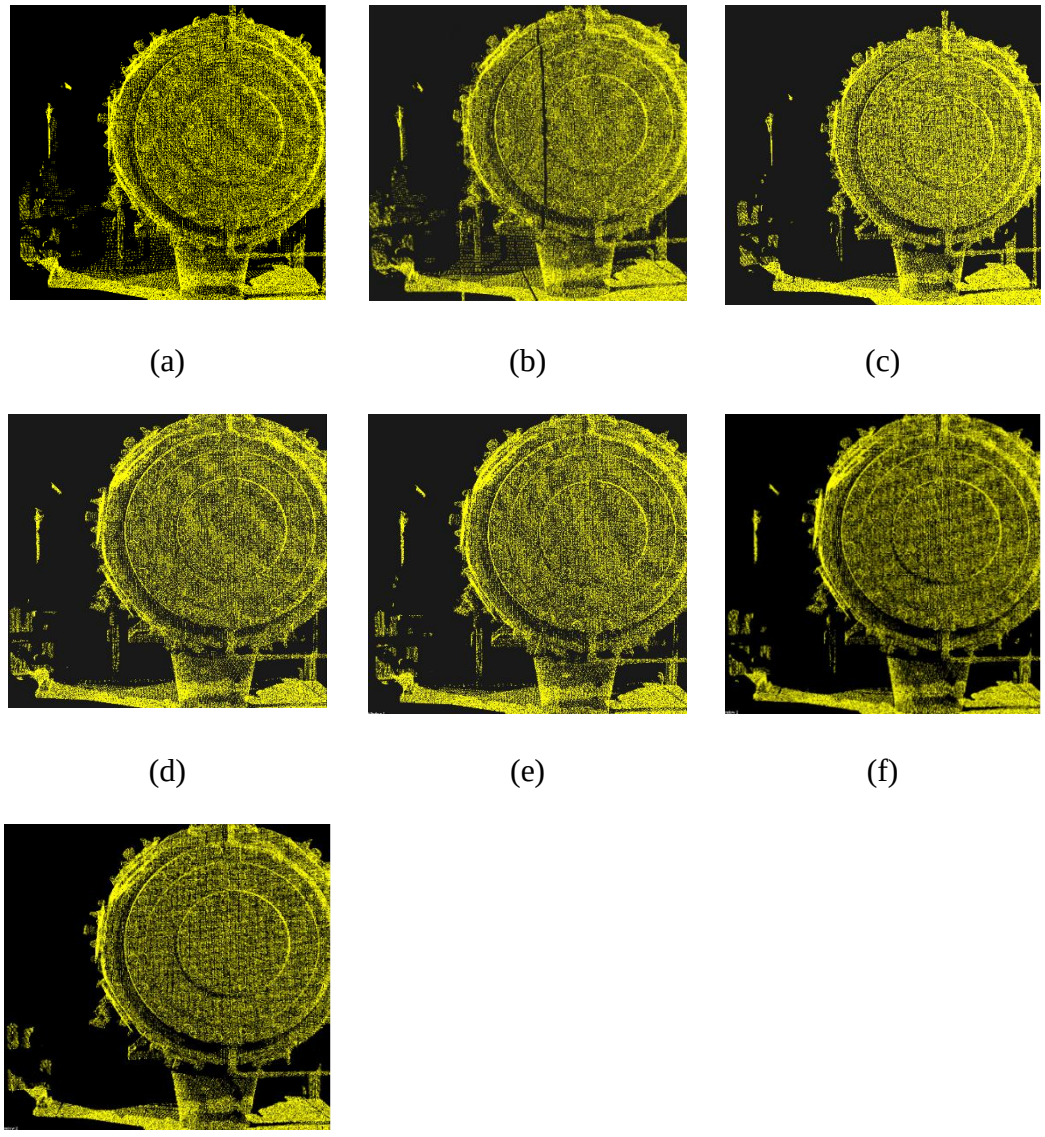
(a) (b) (c) (d)

(e) (f) (g)

Fig. 22. Real Scene model (Happy Buddha). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

The next example is a compressor head. The scanned version is very noisy itself. There are lots of unwanted points in the data, and it is difficult to generate a 3D mesh from the point cloud. The six methods were applied to the point cloud, and the results can be seen in Fig. 23.

Median filter removes a portion of the original parts of the model. Vector median, FVM, and adaptive mean removed some prominent noise with some noise still visible. On the other hand, adaptive median and adaptive vector median performed almost similarly in removing noise. Most of the unwanted points were removed, and the bolts of the compressor looked sharper.



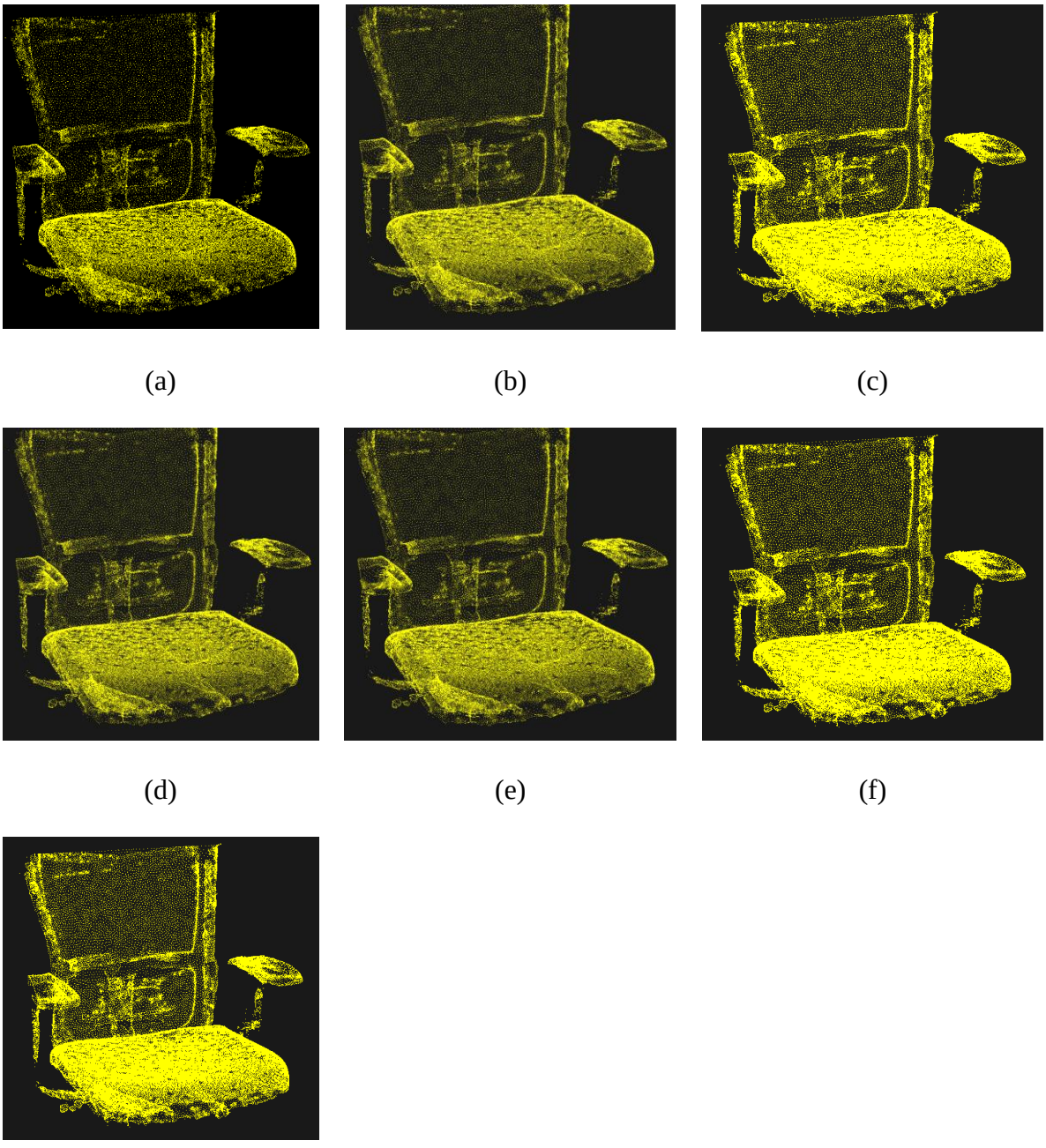
(a) (b) (c)

(d) (e) (f)

(g)

Fig. 23. Real Scene model (Compressor). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

A scanned model of a chair is shown in Fig. 24. The results are almost similar for all of the methods in this research. The difference is barely visible among all the methods.



(a) (b) (c)

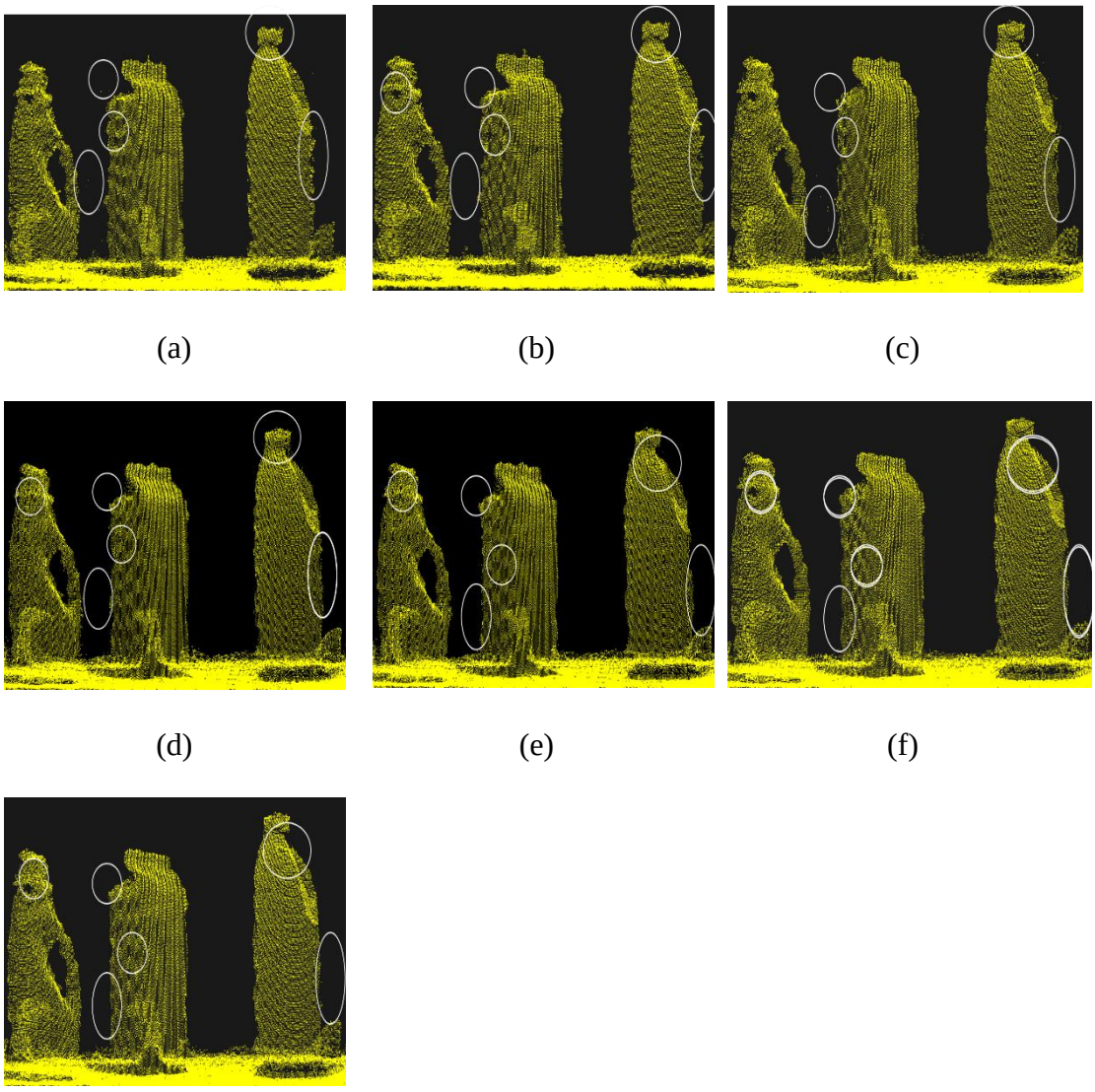
(d) (e) (f)

(g)

Fig. 24. Real Scene model (Chair). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

Fig. 25 illustrates an example of a real scene, which is a point cloud representation of milk cartons. The scanned point cloud is really noisy with some outlier noise, and the original structure of the cartons is distorted due to scanning error. The noise and the areas of improvements after filtering are highlighted with circles. Some areas are recovered by the median filter, but most of the noises are kept unchanged while the vector median removed much of the noises with some visible outliers. Fuzzy vector median adjusted the uneven point clouds and also removed the outliers from the point cloud. Adaptive mean, adaptive median, and adaptive vector median performed almost similarly. Most of the noise is removed, and the edges of the bottles look sharper, and the edges are prominent with these methods.

Fig. 26 is a laser scan of a table scene. The original model has a lot of noise and outliers by the edge of the table. Median, vector median, and adaptive mean filters removed some of the outliers but also removed some points from the edge, mistaken as outliers. On the other hand, FVM, adaptive median, and adaptive vector median performed well. These methods removed the outlier perfectly and kept the edges of the table intact and made it sharper.

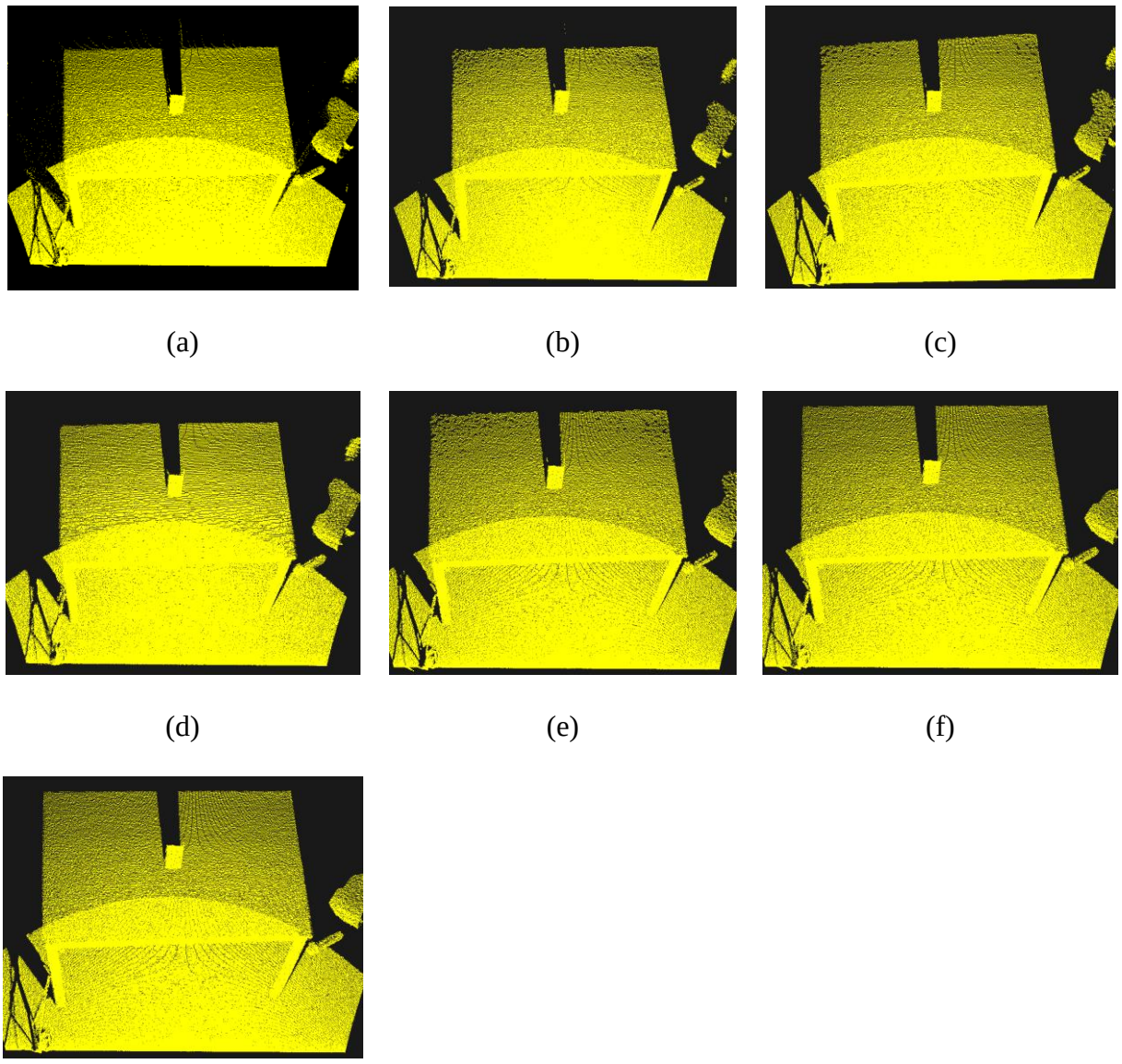


(a) (b) (c)

(d) (e) (f)

(g)

Fig. 25. Real Scene model (Milk Bottle). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.



(a) (b) (c)

(d) (e) (f)

(g)

Fig. 26. Real Scene model (Table). Results of (a) Noisy, (b) Median, (c) Vector Median, (d) FVM, (e) Adaptive Mean, (f) Adaptive Median, (g) AVM.

Fig. 27 shows the RMSE (Root Mean Square Error) of the models (teapot, sphere, Buddha, gear) for the methods with various noise densities. The larger the RMSE is, the poorer the denoising effect is. Large RMSE values indicate that the denoised point cloud data were

seriously deviated from the original point could data. The plot shows the larger noise density tends to result in higher RMSE values. For computing RMSE, three options are available such as (1) if the compared clouds have same number of points, compute using equal indices correspondence heuristic, (2) if the compared clouds do not have same number of points, either compute using the nearest neighbor correspondence heuristic or (3) compute using the nearest neighbor plane projection heuristic. For the plane projection option, the target cloud needs to contain normals.

The equation to derive the RMSE is as follows:

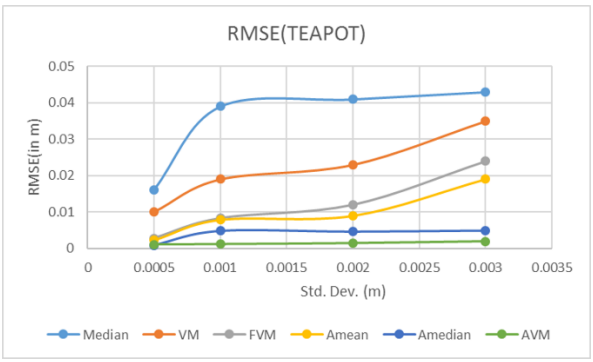
$$\text{RMSE (X, Y)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \tag{9}$$

where *X* and *Y* are two point clouds, *x* and *y* are subsets of *X* and *Y* respectively. It has an identical unit of measurement as the original quantity.

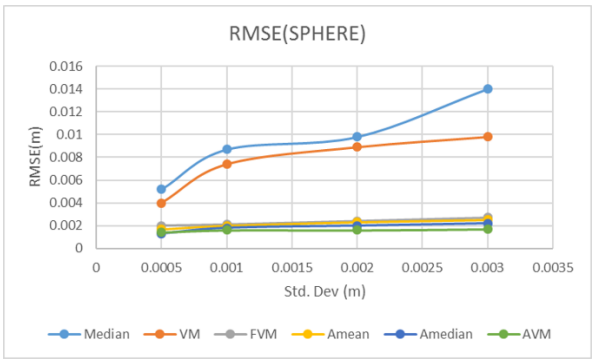
Also, Hausdorff distance is shown in Fig. 28. Hausdorff distance measures the distance of two subsets of a metric space. In other words, two sets are close in the Hausdorff distance if every point of either set is close to nearly some point of the other set. It is defined as follows:

$$H(X,Y) = \max_{x \in X}\{\min_{y \in Y}\{d(x,y)\}\} \tag{10}$$

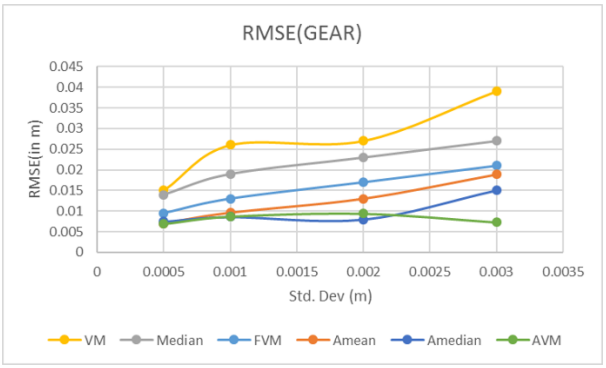
where *x* and *y* are points of sets *X* and *Y* respectively, and *d(x,y)* is the Euclidean distance between *x* and *y*. For each point x on X it searches the closest point y on the other point cloud Y. Here, we have compared with the original data with the filtered data. The lower the distance value, the best is the filtered data. The filtered data is matched with the original data before applying noise.



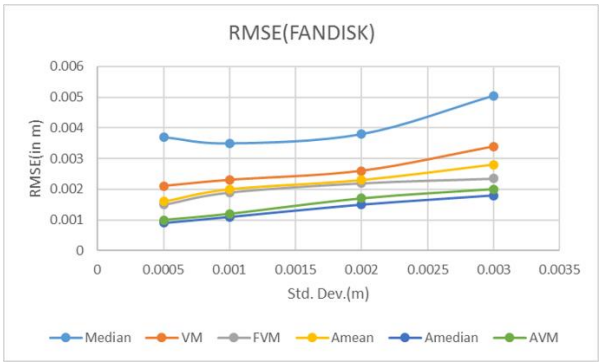
(a)



(b)



(c)



(d)

Fig. 27. RMSE of (a) Teapot, (b) Sphere, (c) Gear, (d) Fandisk.

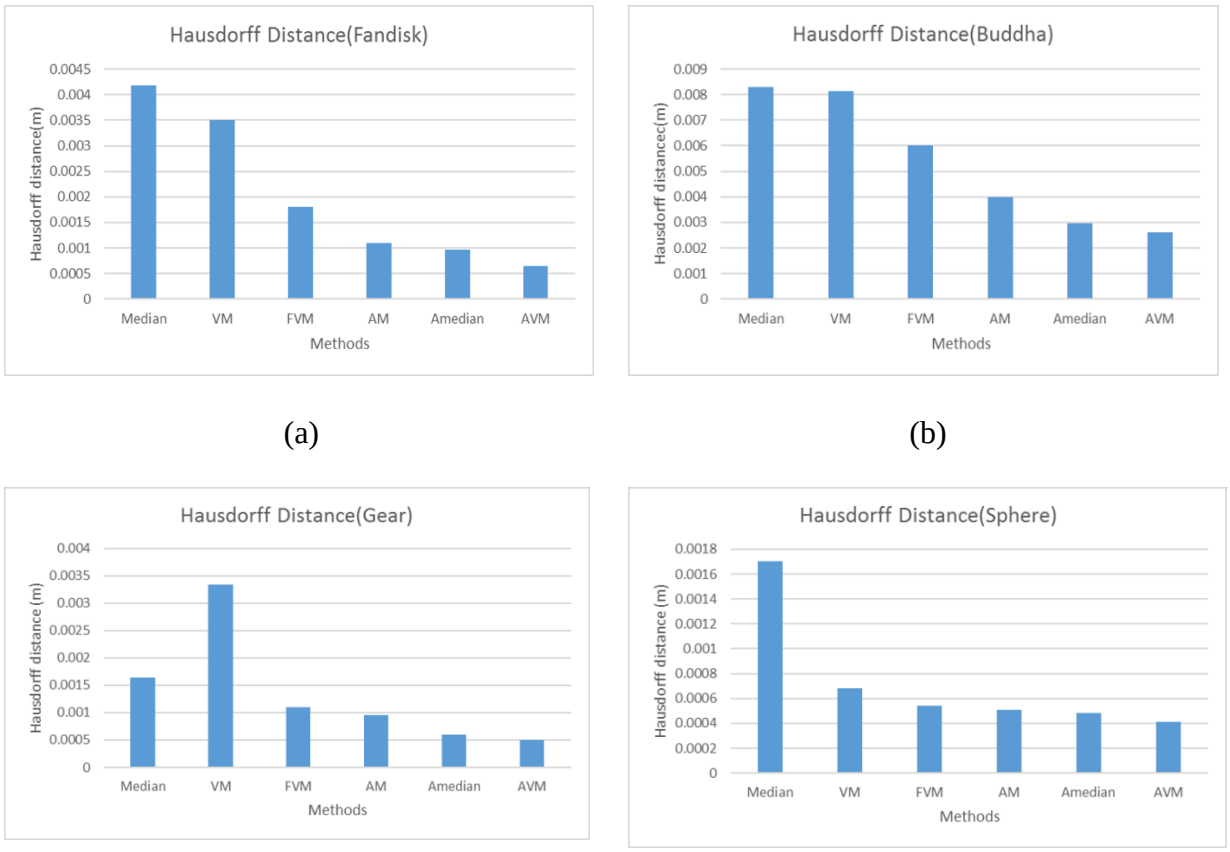


Fig. 28. Hausdorff Distance of (a) Fandisk, (b) Buddha, (c) Gear, (d) Sphere.

Comparison

In this dissertation, we also implemented other widely used point cloud filtering and processing methods, which were Statistical Outlier (SO), Conditional Removal (CR), Radius Removal (RR) and Bilateral Filter (BF). Different point cloud models with corrupted Gaussian noise were used to evaluate the corresponding performance of the algorithms.

Fig. 29 shows the filtering results of these five methods applied to a table scene model. It can be seen that the model is still noisy after filtering by CR and RR, while SOR, BF, and AVM produced better visual results. Overall AVM yielded the best result compared to other in terms of

denoising and feature preserving. TABLE I represents the computation time of the proposed methods, and TABLE III illustrates different methods on different point cloud data with different numbers of points. CR and SOR performed similarly, but BF was relatively time-consuming. The performance of the RR depends on the radius. Larger radius tends to take a longer time to process. Since the number of points affects the computational time, several point cloud data sets were used to evaluate the computational time of the methods.

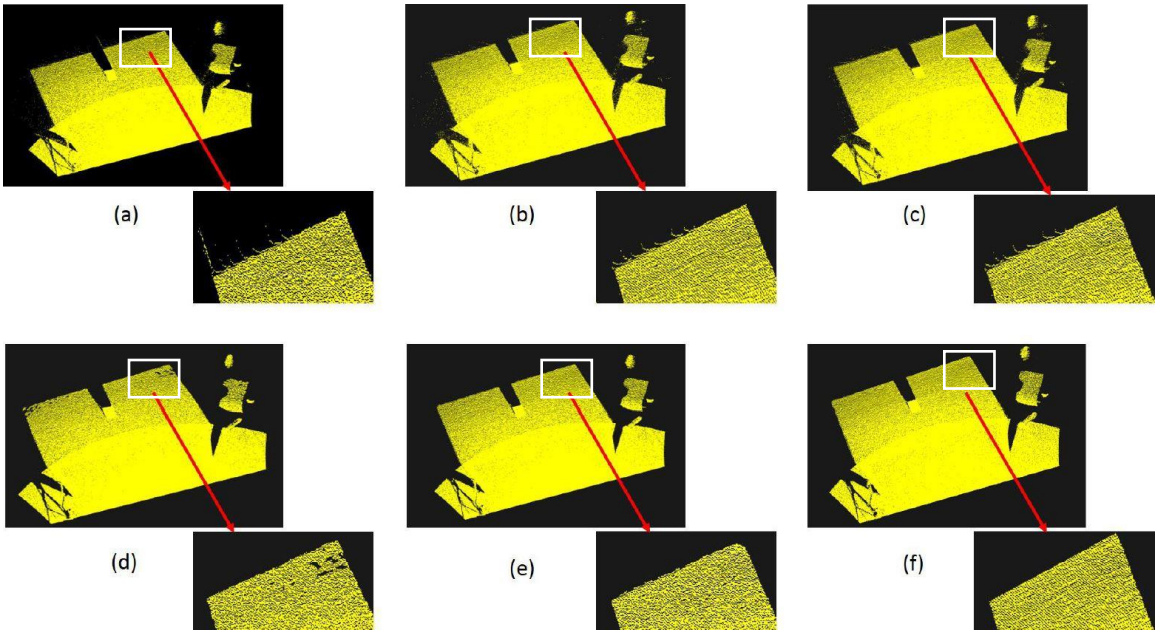


Fig. 29. Comparisons of different filtering methods. (a) Noisy Table model; filtering result with (b) Conditional removal, (c) Radius removal, (d) Statistical outlier removal, (e) Bilateral filter, and (f) Adaptive vector median filter.

TABLE I

COMPUTATIONAL TIME

Dataset (No. of Points)	MED (in sec)	VM (in sec)	FVM (in sec)	AMEAN (in sec)	AMED (in sec)	AVMED (in sec)
Teapot (41784)	2.113	2.571	4.320	3.108	3.672	3.675
Mechanical structure (361043)	10.572	11.489	16.744	15.497	16.133	16.241
Gear (6268)	0.425	0.512	0.647	0.568	0.599	0.610

The time complexity of the algorithms (median, vector median, FVM, adaptive mean, adaptive median, and adaptive vector median) are O(N), O(N^2), O(MN^2), O(MN), O(MN) and O(MN), respectively, where M is the number of points in the cloud, and N is the window size of the methods. Although the methods could extract most of the expected outliers in the models, there are still some noisy points that are not detected due to the similarity of the point density. These methods are expected to behave well when dealing with reasonable point densities.

3.7 Normal based point cloud processing

The AVM method can also be applied to the normal. The application of AVM filters to the normal of each point yields improved results but requires an extensive amount of time. The proposed method works as follows:

1. Given an input point cloud $P_c = \{p_i \epsilon R^3\}$, a local neighborhood $S_{xyz} = \{p_{ij} \epsilon P_c\}$ for each point p_i is determined by the KNN (K-Nearest Neighbor) where p_{ij} is the jth neighbors around p_i and a 3D kd-tree representation is constructed for P_c. Not all point cloud has the normal information. In that case, we estimate the initial normal vector using Principal

Component Analysis (PCA)[126]. The eigenvalues and the eigenvectors of a covariance matrix are created from the nearest neighboring points of the centered point. For each point p_i, the covariance matrix C is defined as:

$$C = \frac{1}{K}\sum_{i=1}^{K} \xi_i \cdot (p_i - \bar{p}) \cdot (p_i - \bar{p})^T, C \cdot \vec{v}_j = \lambda_j \cdot \vec{v}_j, j \in \{0,1,2\} \quad (11)$$

where K is the number of neighborhoods of p_i, ξ_i is a possible weight for p_i. $\bar{p}$ is the 3D centroid of the nearest neighbors, λ_j is the j-th eigen value, $\vec{v}_j$ is the j-th eigenvector of the covariance matrix. For each $p_i \epsilon C$ the normal is denoted as n_i.

2. For each specified window, it calculates the vector median based on a direction metric. The point p_j is the vector median whose angular distance is minimum than all other points in a specific neighborhood. Next, it checks if p_j is noisy based on depth value (z component) (as described in section 3.5.4). If it is not noisy, and the center point p_i is noisy, then replace p_i with p_j (both position and normal information). If the center point p_i is also not noisy, the filter window is expanded, and the above process is repeated. The main idea behind this approach is that two data points p_i and p_j belong to the same surface and none of the points is noisy, they need to have their normal closely oriented, and they should be geometrically close, i.e., $\vec{n}_i \cdot \vec{n}_j \approx 1$.

We test the effectiveness of our method on synthetic datasets containing both sharp and soft features, using the well-known Stanford bunny (Fig. 30), a cylinder (Fig. 31), and a fandisk (Fig. 32). Each dimension of the vertex positions in the bunny model is corrupted by independent zero-mean additive Gaussian noise with a standard deviation of 0.005 and several outliers.

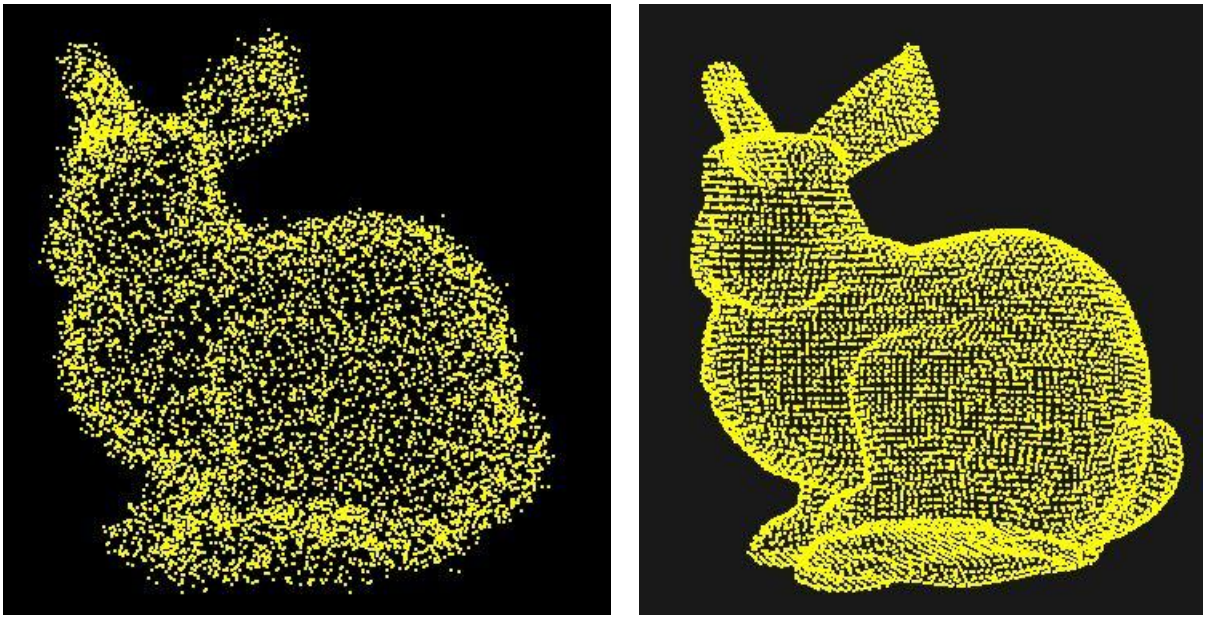


Fig. 30. Normal based AVM filtering of bunny (a) Noisy (0.005), (b) Filtered.

Fig. 31 illustrates a simple cylinder where several outliers have been added to the original model and Fig. 31(b) shows the filtered result of the cylinder. We use the fandisk model (Fig. 32) to demonstrate the capability of normal-based AVM to handle noisy input data with sharp features. The result shows that AVM not only smooths out noise in point positions but also effectively preserves the edges and important features of the fandisk. TABLE II shows the computation time using the distance-based AVM and the normal based AVM. Normal based AVM produced slower computation time as we performed both the depth and normal based computation. However, for models with fewer points, this approach is acceptable. Since this approach keeps both the positional and normal information of the points, it can properly handle the sharp features of the point cloud.

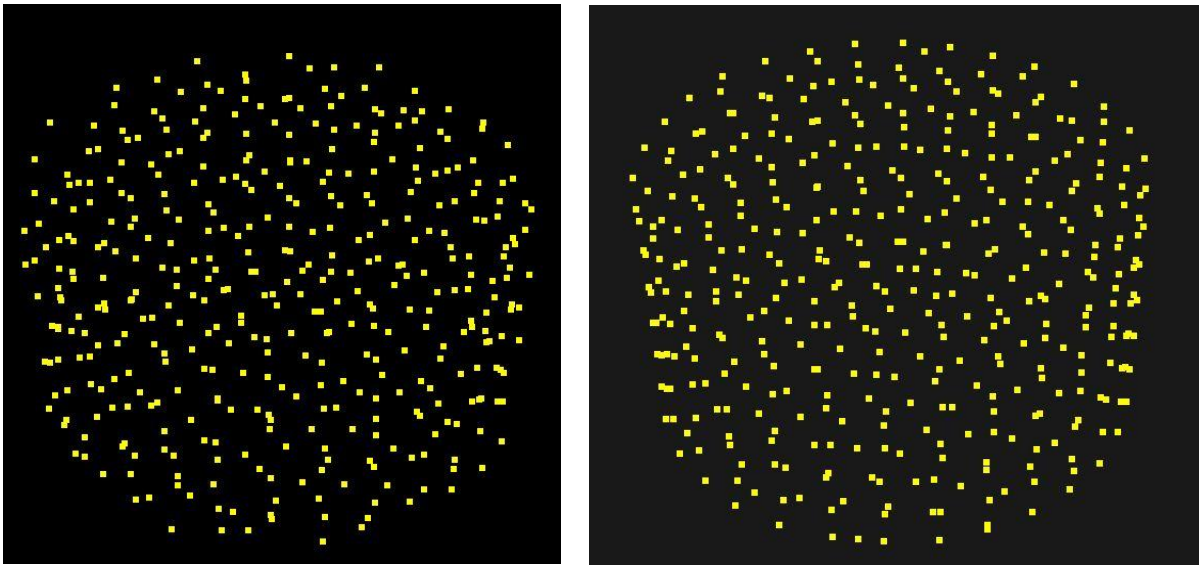


Fig. 31. Normal based AVM filtering of cylinder (a) Noisy, (b) Filtered.

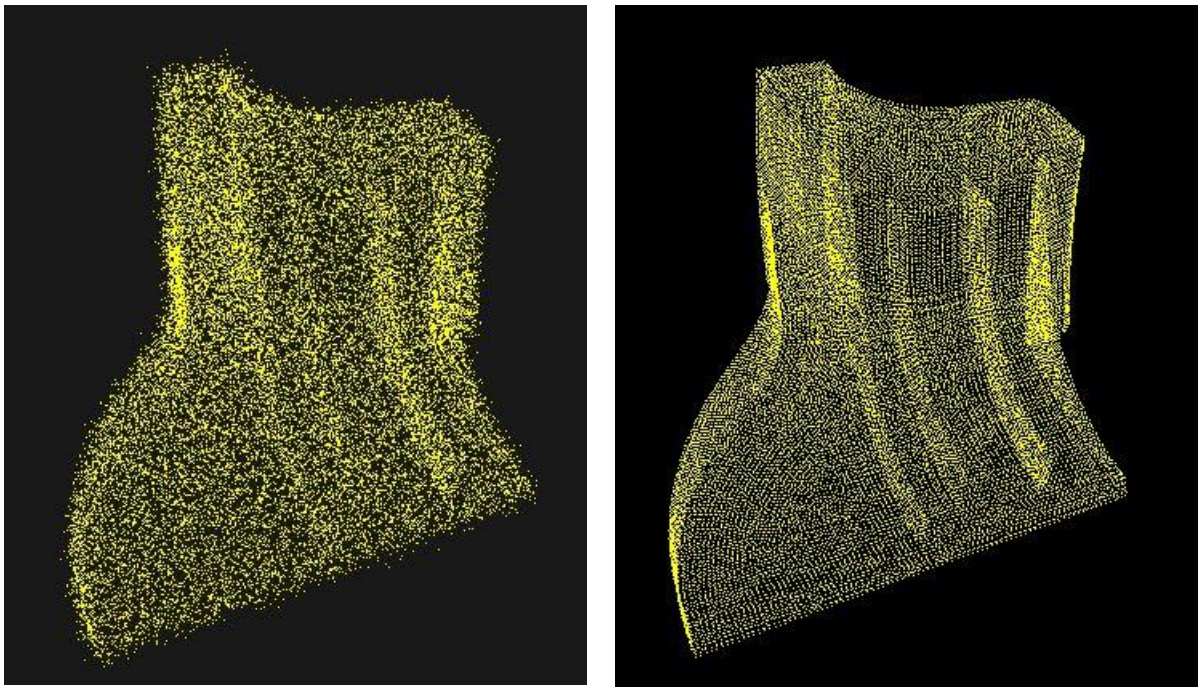


Fig. 32. Normal based AVM filtering of fandisk (a) Noisy (0.002), (b) Filtered.

TABLE II

COMPUTATIONAL TIME

Dataset	AVM (in sec)	NormalBasedAVM (in sec)
Bunny	0.128	0.399
Cylinder	0.101	0.286
Fandisk	0.439	1.537

The next chapter describes another research goal of this dissertation: Parallel implementation of the proposed algorithm adaptive vector median filter using Microsoft's Parallel Pattern Library.

CHAPTER 4

PARALLEL IMPLEMENTATION OF ADAPTIVE VECTOR MEDIAN FILTER

In this dissertation, we use Microsoft's Parallel Pattern Library (PPL) to accelerate the AVM (Adaptive Vector Median) algorithm. This section describes the parallel technologies briefly and the efficiency of our algorithm utilizing this approach.

4.1 Multi-core Architecture

With advances in hardware design and VLSI technologies, a single processor VLSI chip now contains multiple cores, called multi-core or many-core processors. A multicore processor is a single computing module that contains multiple independent core processing units. For example, an Intel Xeon processor can have as many as 24 cores on a single chip. Therefore, computations can be divided into several subtasks, and these subtasks can be allocated to multiple cores on the same CPU chip for parallel processing. The single processor can execute multiple instructions (add, move, branch, etc.) on separate cores at the same time, thus increasing overall speed for the programs. Since the multicore processor can run multiple applications concurrently; it can increase CPU performance. However, the rate of the performance increase depends on the number of cores, the use of shared resources, and the level of real concurrency in the actual software. Traditional, single-core processors are being replaced by the multicore processors so that less single-core processors are being produced and maintained. Therefore single-core processors are becoming technologically outdated. Multi-core processors now are a standard configuration on desktop and laptop computers and even smartphones.

Fig. 33 shows the execution mechanism of single core and multi-core processors. Multiple threads will end up sharing single core (left side of the figure). Two threads are sharing the single core. Switching back and forth to a single thread generate overhead. On the other hand, in the multicore scenario, multiple tasks can run simultaneously in parallel.

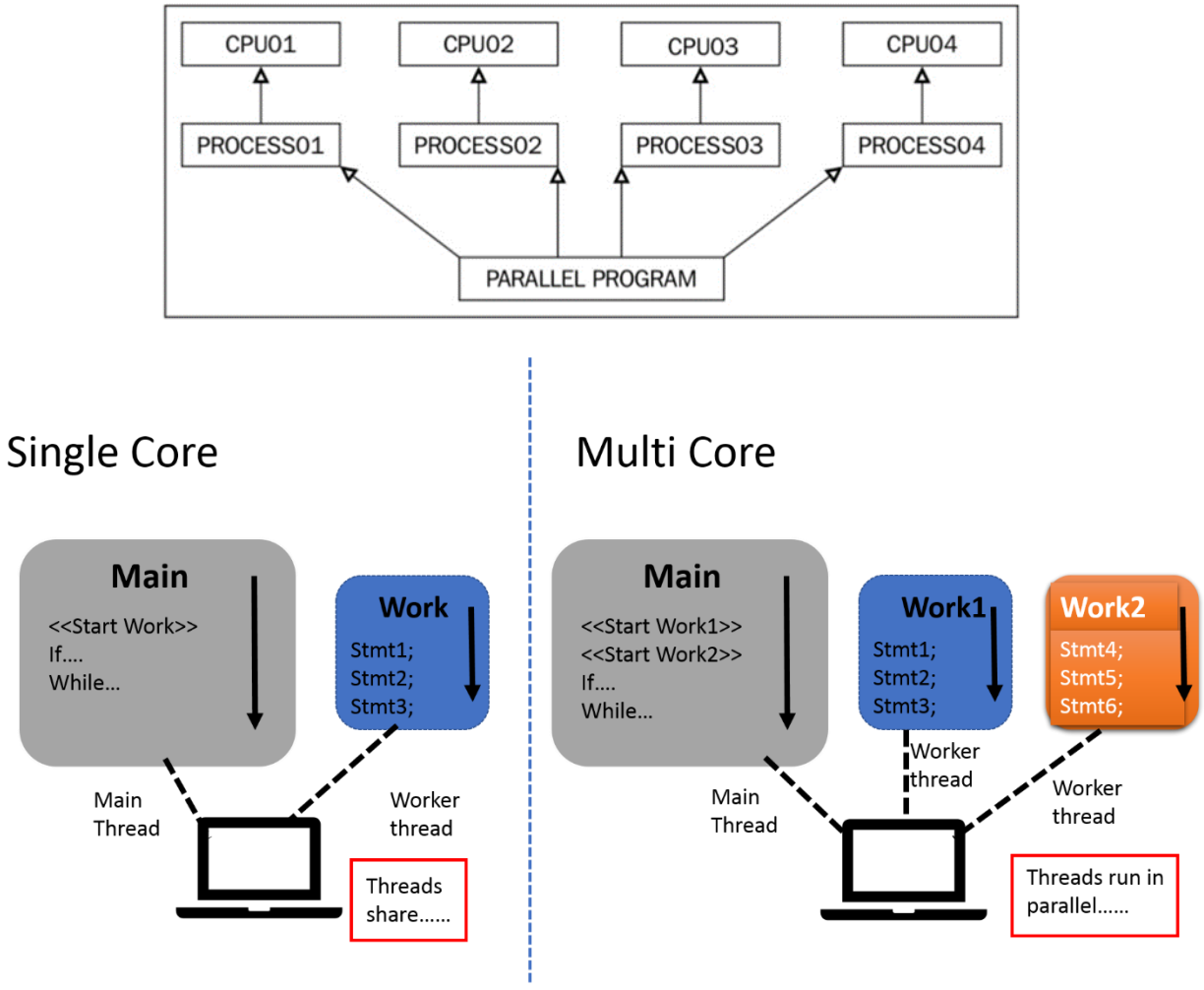


Fig. 33. Execution model of parallel processing.

4.2 Microsoft Parallel Patterns Library

The Microsoft Parallel Patterns Library (PPL) [127] offers a programming model that promotes scalability. This model also provides easy to use platform to develop concurrent applications. The scheduling and resource management components of the Concurrency Runtime are enhanced in PPL. It increases the level of abstraction between the application code and the fundamental threading mechanism. PPL provides generic, type-safe algorithms and containers that act on data in parallel.

The following features are provided by PPL [127]:

- *Task Parallelism/Concurrency runtime*: a mechanism that works on top of the Windows ThreadPool. It works to execute several work items (tasks) in parallel. In the Concurrency Runtime, a task is a unit of work that accomplishes a specific job and typically executes in parallel with other tasks. A task can be broken down into extra, and more fine-grained tasks that are ordered into a task group. Tasks can be used during an asynchronous code, and some operation needs to occur after the asynchronous operation completes. On the other hand, tasks groups can be used to decompose parallel work into reduced pieces.
- *Parallel algorithms*: generic algorithms that work on top of the Concurrency Runtime. It acts on collections of data in parallel. The parallel algorithms are collected from present functionality in the Concurrency Runtime. Parallel pattern library provides loop parallelization with a parallel for loop. Several things were considered in parallel for implementation such as load balancing, nested parallelism, cancellation, exception handling, cooperative blocking, and arbitrary types. The parallel_for algorithm divides tasks in an optimal way for parallel execution, and also it supports *nested parallelism.*

The parallel_for algorithm has two possible loaded versions. The first version inputs a start value, an end value, and a work function. The second version has a start value, an end value, a value by which to step, and a work function. PPL also provides a parallelized version of for_each. The parallel_for_each algorithm performs the tasks simultaneously and performs better with random access iterators, though it works on both forward iterators and random-access iterators. The parallel_for_each is also designed with similar considerations like parallel_for algorithm such as effective load balancing, nested parallelism, cancellation, exception handling, and cooperative blocking. PPL provides another algorithm (parallel_invoke) which is suitable when several independent tasks are needed to execute at the same time. The parallel_invoke algorithm takes a series of work functions (lambda functions, function objects, or function pointers) as its parameters. Two more parallel algorithms are available in PPL namely parallel_reduce and parallel_transform. These algorithms can be used when the code uses a large set, and the performance and scalabilities are benefited if it is converted to parallel version.

- *Parallel containers and objects*: generic container types that offer safe concurrent access to their elements. A *concurrent container* offers concurrency-safe access to the most significant processes. The concurrency::concurrent_vector class is similar to the std::vector class, except that the concurrent_vector class appends elements in parallel. If the parallel code requires both read and write access to the same container, then concurrent containers can be utilized. A *concurrent object* is shared synchronously between components. A procedure that computes the state of a concurrent object in parallel produces the same outcome as another process that calculates the same state serially. The concurrency::combinable class is one instance of a concurrent object type.

The combinable class allows to perform computations in parallel, and then associate those computations into a final result.

4.3 Implementation

Microsoft's Parallel Patterns Library (PPL) provides features for multicore programming. Multicore programming is becoming popular for the applications to speed up executions. Point cloud datasets can contain millions or even billions of points, which can lead to a huge amount of time for processing.

In this dissertation work, PPL is utilized in AVM for several reasons:

- PPL allows to write parallel code without having to manage the formation and break down of the threads by the developer.
- PPL allows serial algorithms to be spread across several cores without having to re-design the algorithm significantly.

The overall method of AVM in parallel implementation can be summarized as below:

1) Read the point coordinates in a single pass and arrange.

2) For a specified window, calculate vector median. For the vector median calculation, the algorithm needs to calculate Euclidean distance between the center point and the neighboring points in a specific window. Then sort the values based on their distances and find the minimum distance among them. PPL's parallel radix sort improved the computation time:

```
parallel_radixsort(begin(distances), end(distances),
[center] (const Point& p)-> size_t {
return euclidean_distance (p, center);
});
```

```
//After sorting the distances
Parallel_for_each(begin(distances), end(distances),
[center](const Point& p){
euclidean_distance(p,center)
}
//Find the minimum distance
compare the distances and update minimum_distance;
```

3) Again in that specific window, check the depth values of the center point and the neighboring points. Find the minimum, maximum and the median in depth value in that window.

```
parallel_for( 0, depth, 1, [&](int y) {
compute_minimum();
compute_maximum ();
store the minimum and maximum depth values in array;
//use parallel_sort for finding the median
parallel_sort(begin(values), end(values));
store the median value in array;
}
```

4) To find if the vector median point is noisy, a condition must satisfied. Use parallel_for to compare minimum, maximum and median value in that speific window.

```
parallel_for( 0, depth, 1, [&](int y) {
check the condition
}
```

If the vector median is not noisy, it goes to next stage otherwise increases widow size and repeat the previous steps.

5) To find if the center point is noisy, a condition must be satisfied. Use parallel_for to compare minimum, maximum and the depth value in that specific window.

```
parallel_for( 0, depth, 1, [&](int y) {
compute_minimum();
      compute_maximum ();
store the minimum and maximum depth values in array;
parallel_for( 0, depth, 1, [&](int y) {
 check the condition;
}
```

6) If the condition satisfies the center point is not noisy, the filter outputs the original value otherwise it is replaced by the vector median value.

To estimate the efficiency of the proposed method, we extensively experiment with both serial and parallel version of the algorithm and presented speed up performance analysis and execution time. Speed up results were carried out on 2 devices: Intel(R) Xeon(R) CPU E5-2687Wv3 10 cores, 3.10 GHz and Intel(R) Core (TM) i7-2760QM CPU 4 cores, 2.39GHz. The operating system for both of the devices was 64 bits Windows 7. In this experiment, the execution time and speed up ratio was collected using the adaptive vector median filter with different sets of point cloud data. Fig. 34 shows the speedup ratio of the proposed algorithm using 4 physical cores and 10 physical cores respectively. The red line indicates the estimated curve line for speedup using Amdahl's law [128] and the blue line indicates the resulted speedup using the proposed method. Amdahl's law can be defined in simple form as below:

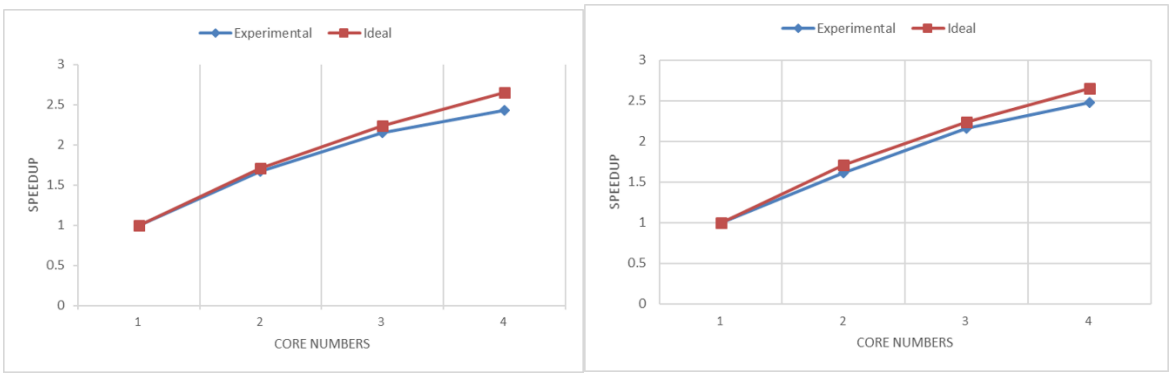
$$\Psi(N) = \frac{t_{seq}}{t_{par}} \leq \frac{T}{(1-\alpha)T+\alpha\frac{T}{N}} = \frac{1}{(1-\alpha)+\frac{\alpha}{N}}, \quad (12)$$

Here, *T* is the time needed for a program to perform on a single CPU, α is the part of the computation that can be done in parallel so that 1- α is the section that must be carried out on a single CPU and *N* is the number of cores. α is determined based on measuring the elapsed execution time of the program. In this application, about 83% of the total code can be parallelized. So, theoretically, the parallel version of the program can run 2.6 times faster (in a 4-core processor) than the serial execution time. However, some intrinsic sequential part of the algorithm, communication cost, load balancing, etc. can limit the achievable speedup.

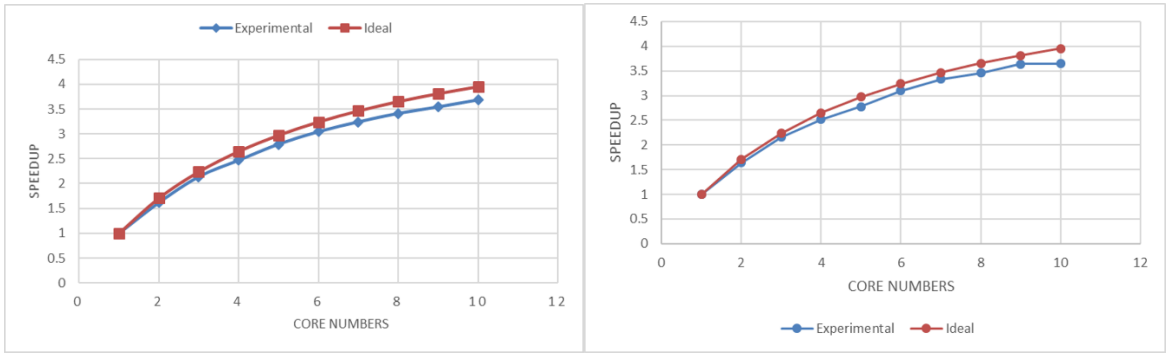
4.3 Results

Experimental results show the behavior is linear for two different processors. The speedup performance clearly depends on the configurations of the processors. TABLE III shows the computation time both in serial and parallel for different point cloud data in a 4 cores device.

The next chapter describes the background, methodology and several results on aerial LIDAR data filtering using AVM.



(a)



.

(b)

Fig. 34. Experimental speedup for a dataset using AVM (a) with 4 logical processors and (b) with 10 logical processors.

TABLE III

COMPUTATION TIME (IN SEC)

Model	No. of points	BF	SO	CO	RO	Serial time (AVM)	Parallel time (AVM)
Bunny	15726	5.428	0.467	0.556	74.69	0.128	0.041
Compressor	361043	12.468	3.887	11.953	34.518	3.117	0.989
Gear	6268	0.2478	0.098	0.227	1.658	0.056	0.017
Milk_carton	307200	34.018	15.102	9.04	65.147	11.872	3.829
Table_scene	460400	13.851	4.012	6.35	87.214	3.694	1.055
Happy_buddha	79087	8.574	0.889	2.703	97.245	0.631	0.208
Thai_statue	4999996	122.045	52.148	98.37	852.145	40.733	13.577
Armadillo	172974	20.225	1.125	3.59	703.03	0.904	0.262
Julius	36201	11.516	0.457	1.19	374.35	0.297	0.086
Iron	85574	24.144	0.789	2.914	98.21	0.676	0.191

CHAPTER 5

AERIAL LIDAR DATA PROCESSING

Aerial LIDAR is a special type of LIDAR that is important for many applications. In this chapter, the adaptive vector median is further optimized for effective processing of aerial LIDAR data.

5.1 Aerial LIDAR

Aerial LIDAR is a special type of LIDAR that is mounted to an aircraft equipped with a Global Positioning System (GPS) sensor and Inertial Measurement Unit (IMU) sensor. The point cloud captured by aerial LIDAR is geo-referenced, with x, y usually representing latitude and longitude positions and z representing the elevation of the ground or features on the ground, such as vegetations and buildings. Such georeferenced data are utilized for the purpose of mapping, recognition, and classification. Aerial LIDAR is usually mounted on an airplane or helicopter, called airborne laser scanning (ALS) systems. Recently, Unmanned Aerial Vehicle (UAV), an airplane without a humanoid pilot onboard, is becoming the most promising platform for a laser scanner for economic reasons. However, the data processing techniques needed to produce a point cloud from raw data acquired by the UAV system are not fully established. The UAV system requires more calibration and computation to produce a point cloud completely on geometric quality because the UAV is more delicate to the platform fluctuation and vibration than the ALS. Thus, the ALS system has more benefits in data quality, collection speed, and scanning coverage compared with other LIDAR systems. Hence, we have used the ALS system-based data in this dissertation research.

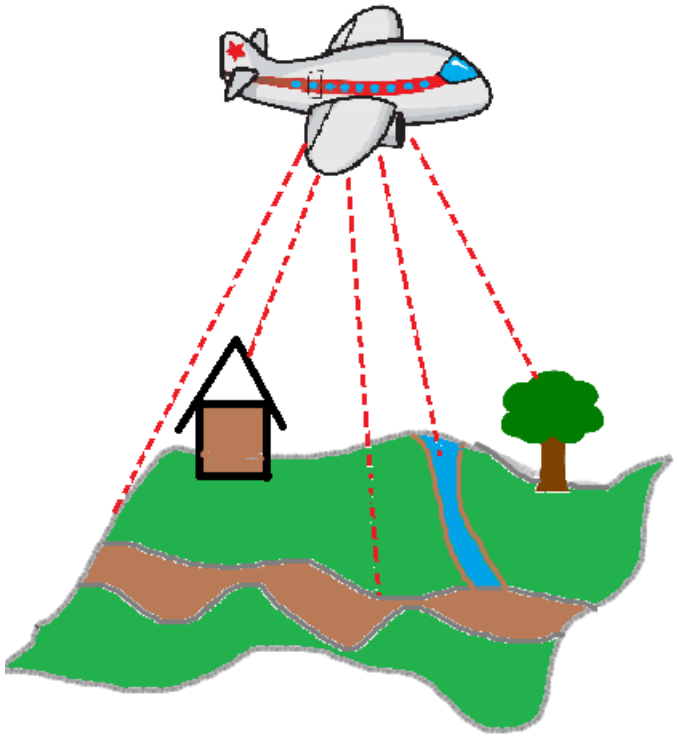


Fig. 35. Aerial LIDAR technique.

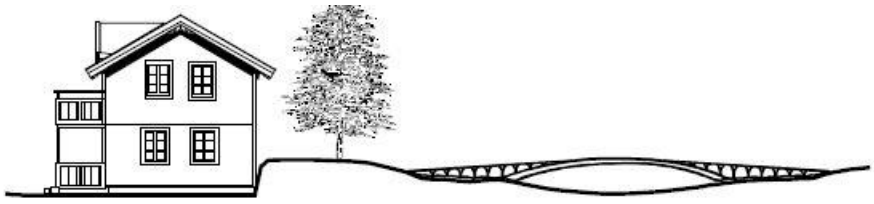
The generation of Digital Terrain Model (DTM) or bare earth surface elevation has been one of the most elementary applications of aerial LIDAR technology in recent years. The DTM generation needs filtering out the ground (or terrain) from raw LIDAR data so that the bare earth surface elevation can be computed. Researchers have been working on this research field for several years. Some of the research areas have been discussed elaborately in the literature review in chapter 2. A few more examples are presented here to highlight the trends in this area. One of the widely used methods is *simple filtering*, which allocates a point with the lowest elevation in a local area to ground; *morphological filtering* extends ground points if they are within a distance threshold to a seeded ground point [65]; *recursive filtering* recursively updates a reference terrain surface by adding ground points obtained from topological analysis [129]; *surface-based filtering* removes above ground points from a surface model that is primarily created using all the

points [130], [57]; *segment-based filtering* identifies ground segments (points with a similarity are grouped as a segment) by comparing surface normal between ground-assigned segments and others [67] and [57], [48]. Mostly, these classified ground points are converted into one of the formats, TIN (Triangulated Irregular Network), grid, mesh, and quad-tree to generate DTM.

5.2 Basic Definitions

The following basic definitions were presented by Sithole and Vosselman [48]. The associated illustrations are also taken from [48].

Landscape: The geography. A scene consisting of the earth and any other features (buildings, trees, power lines, etc.,) residing on it.



Bare Earth: Earth or any thin layering (asphalt, pavement, etc.) covering it. Haugerud and Harding [131] defined bare earth as "the continuous and smooth surface that has nothing visible below it."

Object: Vegetation and other non-natural features that have been constructed by a human.

Detached object: Objects that rise vertically (on all sides) above the bare earth or other Objects.

Attached object: Objects that rise vertically above the bare earth only on some sides but not all (e.g., bridges, gangways, ramps, etc.,).

Filtering: Generalization of the bare earth from point clouds.

Outlier: Point(s) in a point-cloud that are not from the landscape (e.g., birds, gross errors from the ALS system, etc.) and resides far away from the ground points.

5.3 Ground Characteristics Used for LIDAR Ground Filtering

LIDAR point measurements are influenced by three components: bare ground, above-ground objects, and noise [132].

$$M_s = H_g + H_{non\text{-}g} + M_n \quad (13)$$

where M_s is the measurements from the LIDAR sensor, H_g is the elevation of the ground, $H_{non\text{-}g}$ is the elevation of the non-ground, and M_n is the undesired measurements (the noise from sensors, airplanes, or birds).

Generally, the ground points are the bare earth points that represent the low-level surface of an area. Trees (tall or small), buildings, bridges, electric poles, shrubs, etc. are the non-ground points that exist above the bare earth.

However, sometimes non-ground points can be confusing to identify and appear to be ground points. So, some specific characteristics should be taken into account to understand and identify or recognize ground points that differentiate them from non-ground points.

Four categories of characteristics can be defined of the ground or bare earth surface based on their physical features:

1. *Category 1- Lowest elevation*: Ground surface or the bare earth usually has the lowest height in a local neighborhood. Several existing methods use this feature to adjust the ground filtering process [61], [133], [134], [135].

2. *Category 2 – Steepness*: The slope of the surface is considered here. Generally, two neighboring bare earth points have lower slope than that between bare earth and a non-ground object [134]. Several ground filtering approaches [136],[75] define a point with slope larger than the maximum ground slope as the non-ground points. However, this steepness of slope may differ for erratic surface types. An even urban area may have a lower slope value than a

mountain area. So, complex surfaces such as uneven mountain area or high-density forest canopy may have steeper slopes and may require a larger threshold to effectively recognize ground from non- ground objects.

3. *Category 3 - Elevation difference*: This category is also based on height. Since most ground areas have inadequate sharp changes in height, the elevation difference from bare earth to surrounding bare earth is usually less than the difference between ground and neighboring non-ground points. Hence, trees (tall or small), buildings, electric poles, etc. are indeed non-ground points as they have a higher elevation than a location-specific threshold [65].

4. *Category 4 - Similarity in features*: In most of the cases, bare earth is discreetly continuous and smooth; on the other hand, non-ground objects have different heights and textures. Trees and buildings have different features. Trees and shrubs generally are less smooth than bare earth and buildings So, they can be removed based on morphological characteristics [137].

These four categories are frequently used for filtering aerial LIDAR data. However, in some cases, the bare earth may not have these common characteristics, and the assumptions may fail and misinterpret ground points as non-ground points or vice versa. For example, cliffs have higher elevation difference, and many filtering methods misclassified them as non-ground points.

5.4 Methodology

The aerial LIDAR data filtering presented in this chapter is mainly based on the adaptive vector median filter proposed for point cloud mentioned in Chapter 3. The concept of point cloud filtering is extended and modified for the aerial LIDAR data ground filtering purpose. Several reasons are considered for the LIDAR data filtering. The size of the LIDAR data is tremendously

enormous. Abrupt changes in terrain heights such as cliffs, mountain ridges, and peaks can be likely to be removed. However, LIDAR data may have varied landscapes with complex objects or abrupt changes in terrain heights. These difficult situations can make the filtering task challenging. This method of ground filtering also faces several challenges. Firstly, the raw LIDAR data are in las format that is converted to pcd file format for our filtering purpose. Secondly, huge datasets require large computation time and effort.

For the ground detection, the filtering process works in two steps. The first step removes the outliers and the objects that are far from the ground such as large buildings, tall trees, electric poles, etc. based on the threshold value. The second step filters out the noise that is left behind during the first step and the non-ground points that are close to the ground using adaptive vector median filter. The steps are discussed briefly:

Step 1: A Kd tree is constructed for the nearest neighbor search for the point cloud. This step of the algorithm needs a few more parameters in addition to *x, y,* and *z* coordinates of the points in the original LIDAR data. For the given point cloud this method identifies the non-ground points in a local window.

Within a specific window radius, one height threshold (minimum elevation) for each window is defined. Another filtering parameter is the height difference threshold, which is the minimum of the height of the object in each window.

Then, all points with elevations greater than a threshold above the minimum are discarded. A point P_{ij} in a specific region is removed if:

$$Z_{i,j} - Z_{i,min} > H_{i,T} \tag{14}$$

where $Z_{i,j}$ is the elevation of P_{ij}, $Z_{i,min}$ is the minimum elevation inside the window, $H_{i,T}$ is the height difference threshold.

Step 2: The method is applied to the rest of the points that remain after Step 1 filtering. For each specified window S_{xyz}, it calculates the vector median based on distance. The point containing the vector median in S_{xyz} is defined as p_j. This filter detects the noisy candidate p_i and replaces the noisy candidate with the vector median of the points in a local window. However, it changes the size of S_{xyz} during filter operation, depending on the following conditions.

The algorithm checks both the point of interest and the point containing the vector median. Four different situations may arise in detecting noise in the point cloud.

1. The point of interest p_i is noisy,

2. The point containing the vector median p_j is noisy,

3. Both the p_i and p_j is noisy, and

4. None of them are noisy.

Given a noisy point cloud and an initial window size, the adaptive vector median filter performs several steps.

Stage 1: First, for each specified window, it calculates the vector median. Next, it checks if the point containing the vector median value p_j is noisy based on the elevation (z component) using the following formula:

$$Z_{min} \leq Z_{med} \leq Z_{max} \tag{15}$$

where Z_{min} is the minimum of elevation in S_{xyz}, Z_{med} is median of elevation in S_{xyz} and Z_{max} is the maximum of elevation in S_{xyz}. If p_j is not noisy (Eq. 7 is satisfied) , then go to Stage 2; otherwise, expand the window and repeat Stage 1.

Stage 2: Check if the center point p_i is noisy by the following formula:

$$Z_j - Z_{min} > 0 \; and \; Z_j - Z_{max} < 0 \tag{16}$$

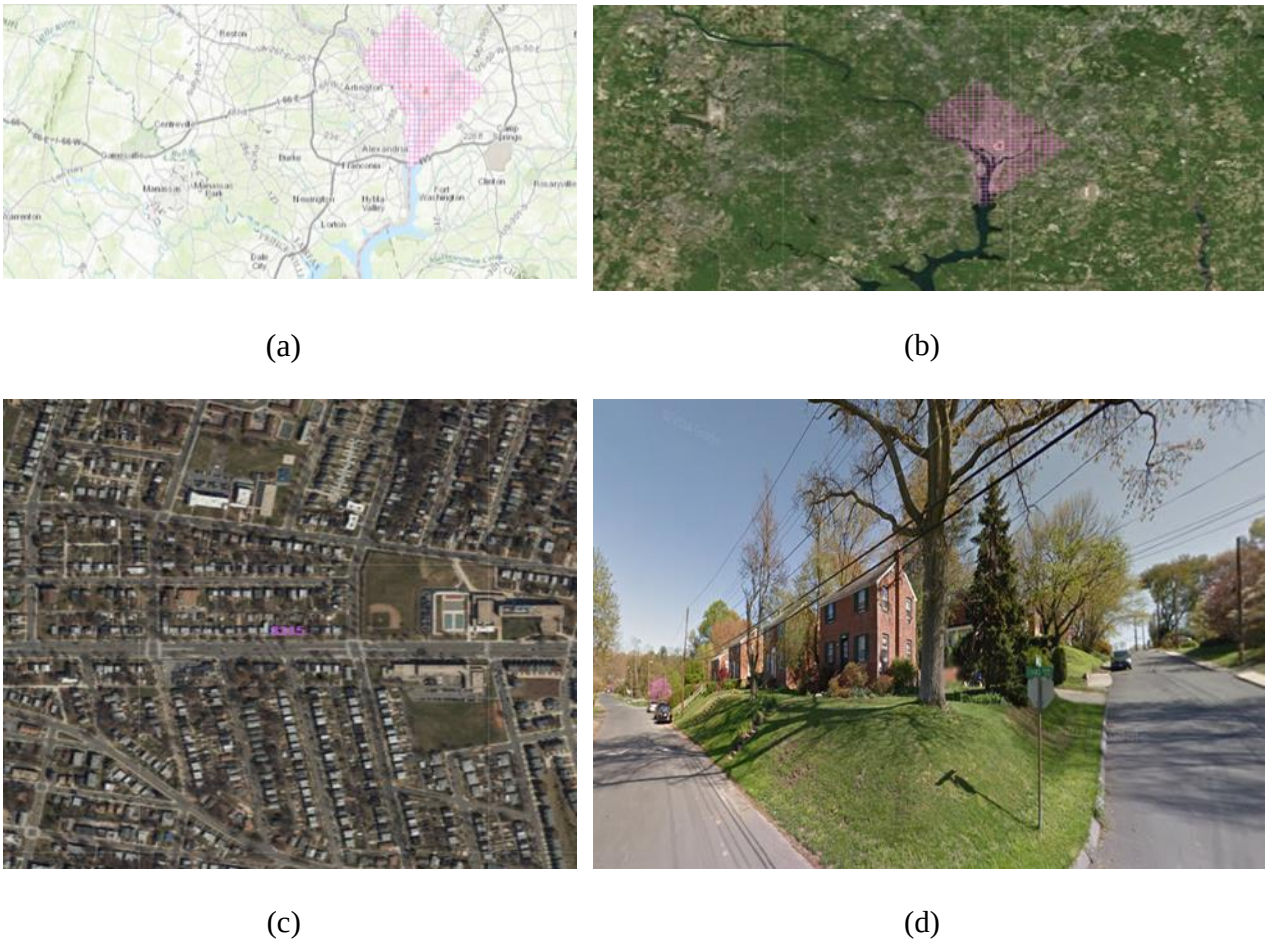
where Z_i is the elevation of p_i. Z_{min} is the minimum of elevation in S_{xyz} and Z_{max} is the maximum of elevation in S_{xyz}. If the condition satisfies, then p_i is not noisy, the filter output is the original center point, and the algorithm continues to the next point; otherwise, p_i is replaced by the vector median p_j; If both the vector median p_j and the center point p_i are noisy, the filter window is expanded, and the above process is repeated.

5.5 Experimental Results

The AVM filter was tested on two types of datasets. One is publicly available LIDAR data points of the Washington DC area, and the other one is reference data provided by the ISPRS that is widely used to validate the efficiency of the LIDAR filtering methods.

5.5.1 Washington DC area

The study area (Fig. 36) is located in the District of Columbia, Arlington County, in Washington DC covering 80 square miles. The single "LAS" file (LASer file) contains approximately 2,000,000~50,000,000 points. The ground sampling distance is greater than 0.35 meter. It is worth mentioning that the LAS dataset contains point cloud with no filtering applied.



(a) (b)

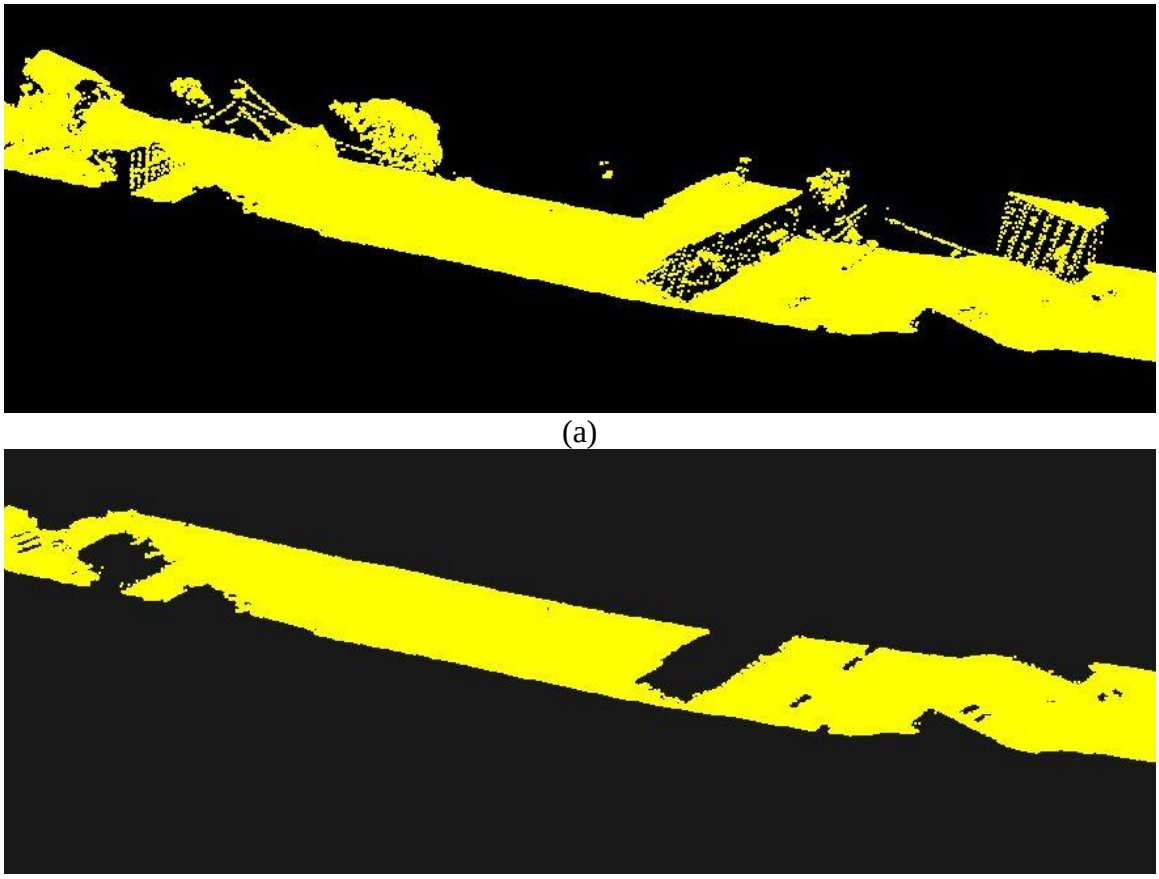
(c) (d)

Fig. 36. Original Study area (Washington DC). (a), (b), (c) Map view, (d) Street view.

The LIDAR data of Washington DC area was delivered in RAW flight line and created Classified LAS 1.2 Files with individual 1500m x 1500m tiles. The LIDAR data was collected in winter season in the year 2014. The ground contained no snow and rivers were at or below normal levels.

To see the efficiency of AVM on LIDAR data, we presented three sample areas below. Fig. 37(a) shows the original point cloud of the District of Columbia, Arlington County, in Washington DC. The data consists of lots of noise, an outlier, electric wires and polls, trees, house,

and buildings, etc. Fig. 37(b) demonstrates the filtered result using the AVM. The proposed method successfully removed the noise and other non- ground objects.

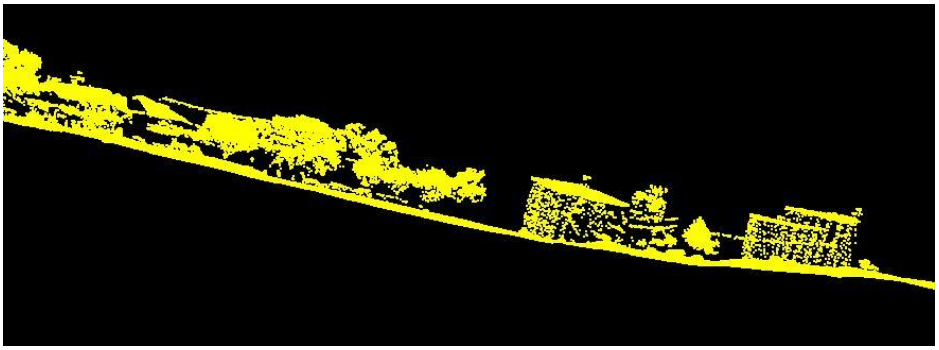


(a)

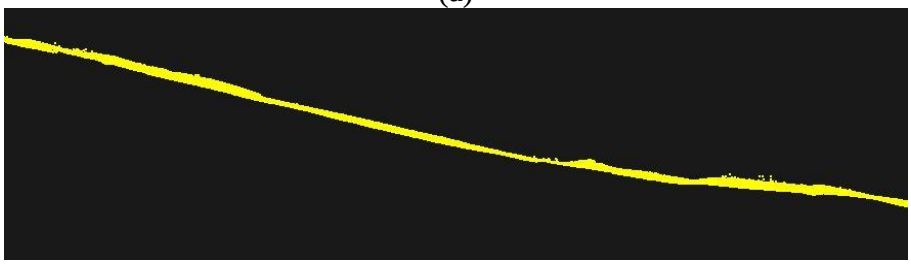
(b)

Fig. 37. Study area-I. (a) Original (b) Filtered.

A side view of another part of the previous location is shown in Fig. 38. AVM successfully removed most of the non-ground objects. Some non-ground points that are close to the ground can be misclassified as ground points. The original data of this location is in LAS format. For processing purpose, we convert it to PCD file format. Fig. 39(a) illustrates another area where small buildings, trees, shrubs, and electric poles and wires exist in the original data. AVM successfully removes the non-ground points, and the final result is shown in Fig. 39(b).

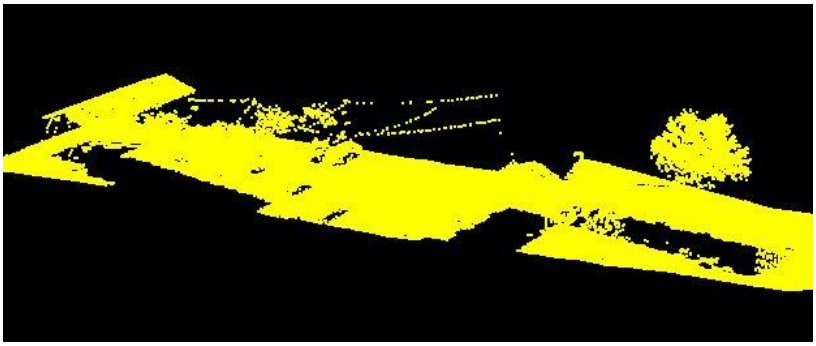


(a)

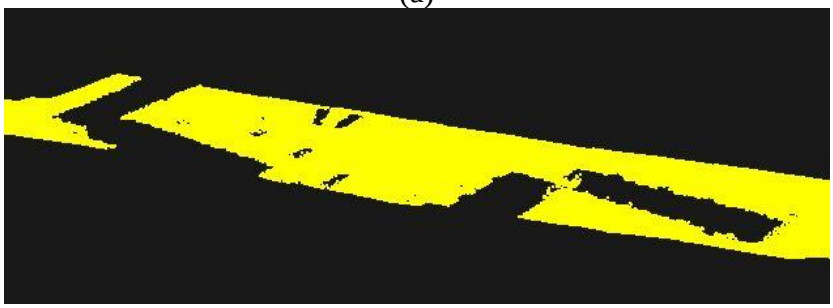


(b)

Fig. 38. Study area-II. (a) Original (b) Filtered.



(a)



(b)

Fig. 39. Study area-III. (a) Original (b) Filtered.

5.5.2 Reference Data

The following are 15 reference datasets that are widely used for comparing aerial LIDAR filtering results generated for their efficiency and accuracy. These areas were chosen mainly because of their feature content in an assorted way (open fields, vegetation, buildings, roads, railroads, rivers, bridges, power lines, water surfaces, etc.,). However, the areas can be divided into two groupings, urban and rural. The sites denote four regions with urban characteristics and another three with rural characteristics. Some characteristics of the test-sites are provided in TABLE IV.

TABLE IV

CHARACTERISTICS OF THE REFERENCE DATA

Environment	Point spacing	Site	Sample	Features
Urban	1.0-1.5 m	1	11	Mixture of vegetation and buildings on hillside
			12	Buildings on hillside
		2	21	Large buildings and bridge
			22	Irregularly shaped buildings
			23	Large, irregularly shaped buildings
			24	Steep slopes
		3	31	Complex buildings
		4	41	Data gaps
			42	Railway station with trains
Rural	2.0-3.5m	5	51	Mixture of vegetation and buildings on hillside
			52	Buildings on hillside
			53	Large buildings and bridge
			54	Irregularly shaped buildings
		6	61	Large, irregularly shaped buildings
		7	71	Steep slopes

A total of seven test sites (four urban and three rural) were chosen because they contained a variety of features that were anticipated to be challenging for automatic filtering. The datasets comprise terrain with steep slopes, dense vegetation, densely packed buildings with vegetation in between, large buildings (a railway station), multi-level buildings with courtyards, ramps, tunnels, tunnel entrances, bridges, a mine, and data gaps. The urban sites were recorded with a point spacing of 1–1.5 m, and the rural sites had a point spacing of 2–3.5 m.

The reference data were produced by filtering the datasets manually. All points in the datasets were labeled either ‘‘ground’’ or ‘‘non-ground’’. For the purpose of this test, ground or the bare-earth was defined using the definition presented in the previous section (earth or any thin layering (asphalt, pavement, etc.) covering it). According to this definition bridges, gangways, etc., were treated as objects. Ramps leading towards bridges, however, were classified as bare earth.

Furthermore, the bare-earth was treated as a continuous surface. From the seven datasets, 15 samples were abstracted. These 15 samples were representative of different backgrounds.

Samp11

Samp 11 (Fig. 40) is a LIDAR scan of an area with a combination of trees and buildings. Steep slopes and complex scenes are present in sample 11. In the lower portion of the slopes of sample 11, there are many buildings, and several difficult objects or resources are present with which the filters may have difficulties to identify them properly. However, the filtering result effectively identified the ground. Several non-ground points are identified as ground points in this sample area due to the close features to the ground points.

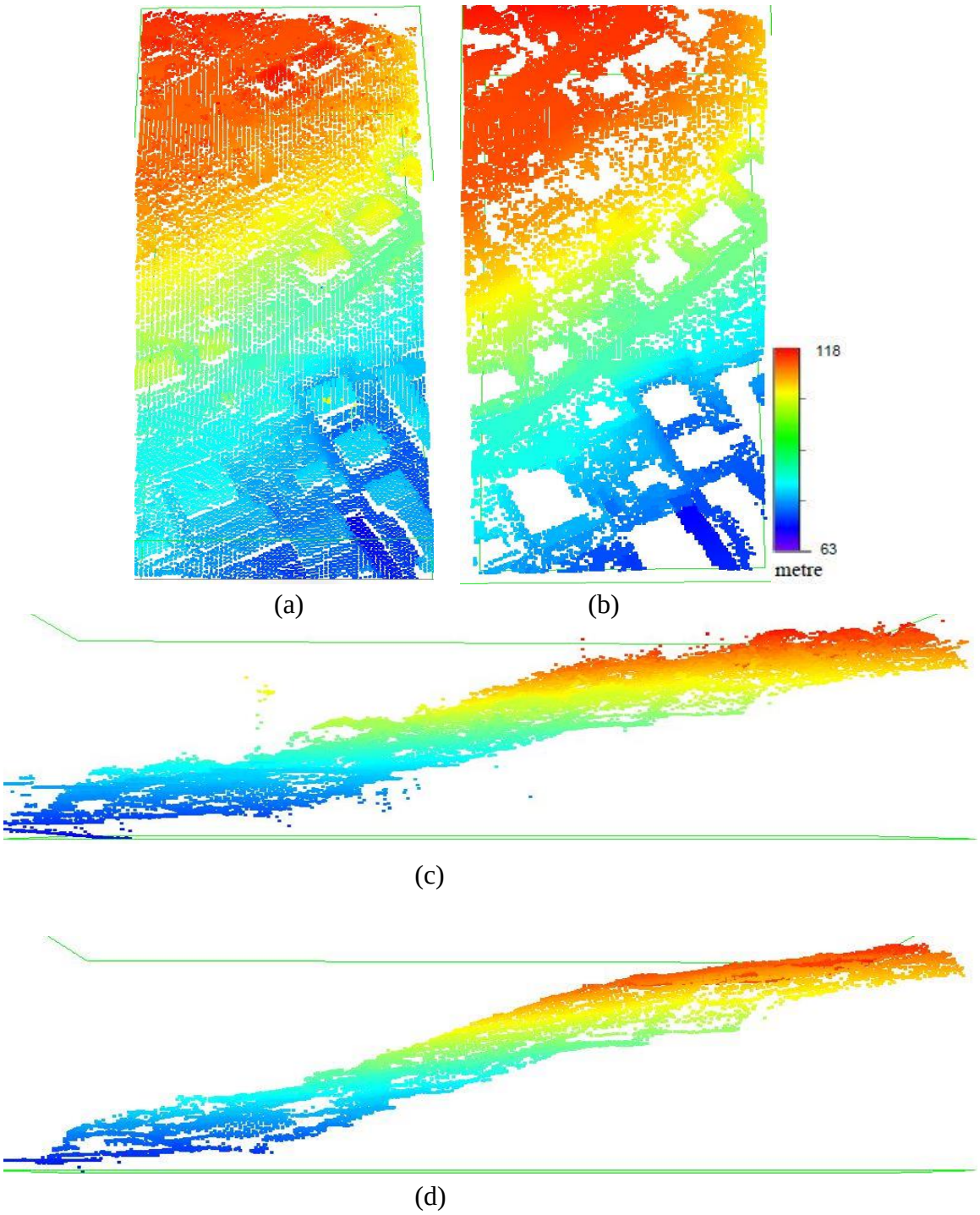


Fig. 40. Sample 11 (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

Samp12

Sample 12 (Fig. 41) has buildings on the hillside. The proposed filter can effectively remove diverse buildings of different sizes and complex shapes. Small objects such as cars and shrubs are mostly eliminated.

Samp 21

This region (Fig. 42) contains rooftops, houses, several scattered non-ground objects, etc. The filtering method successfully identified most of the ground and the non-ground objects.

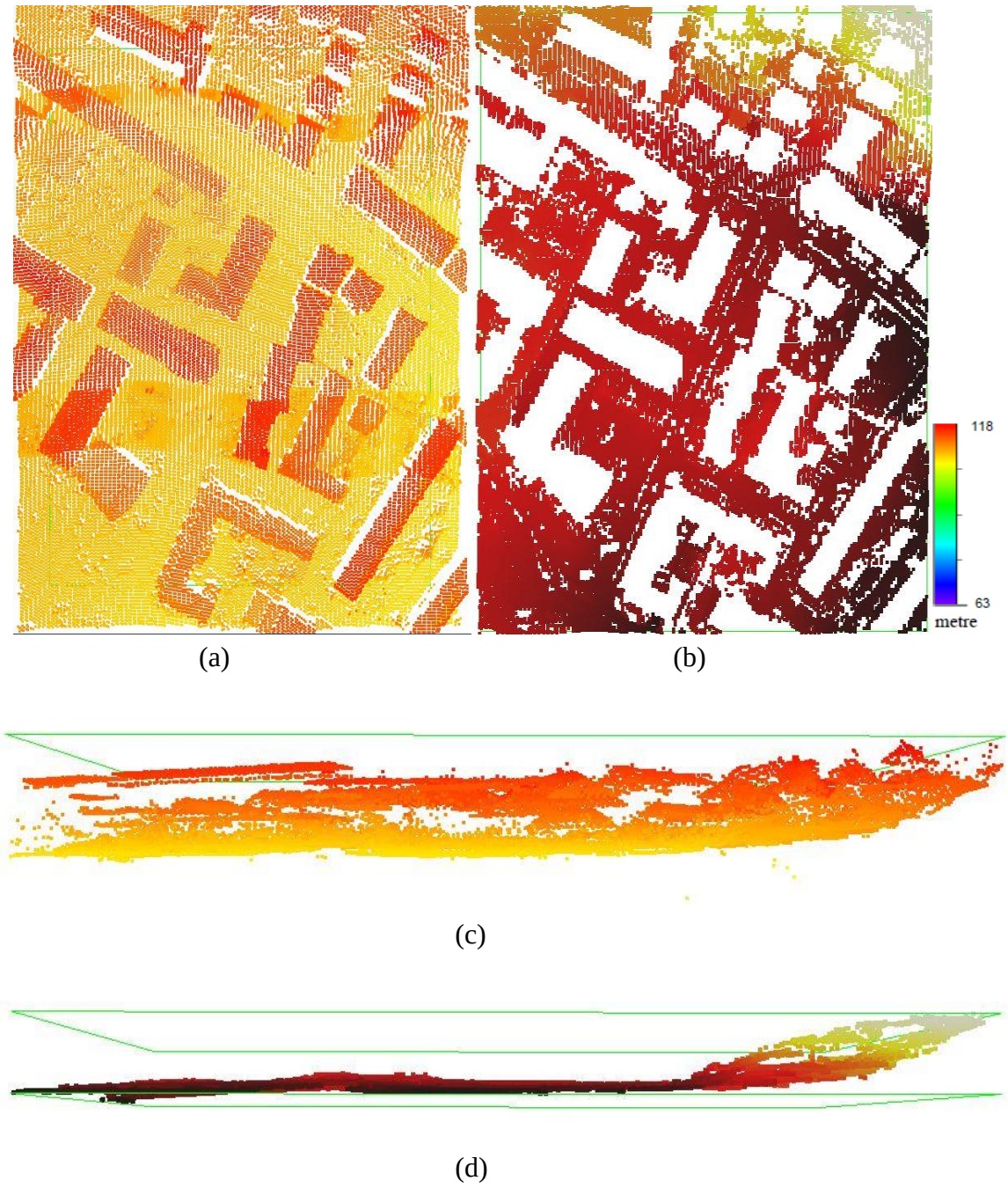


Fig. 41. Sample 12. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

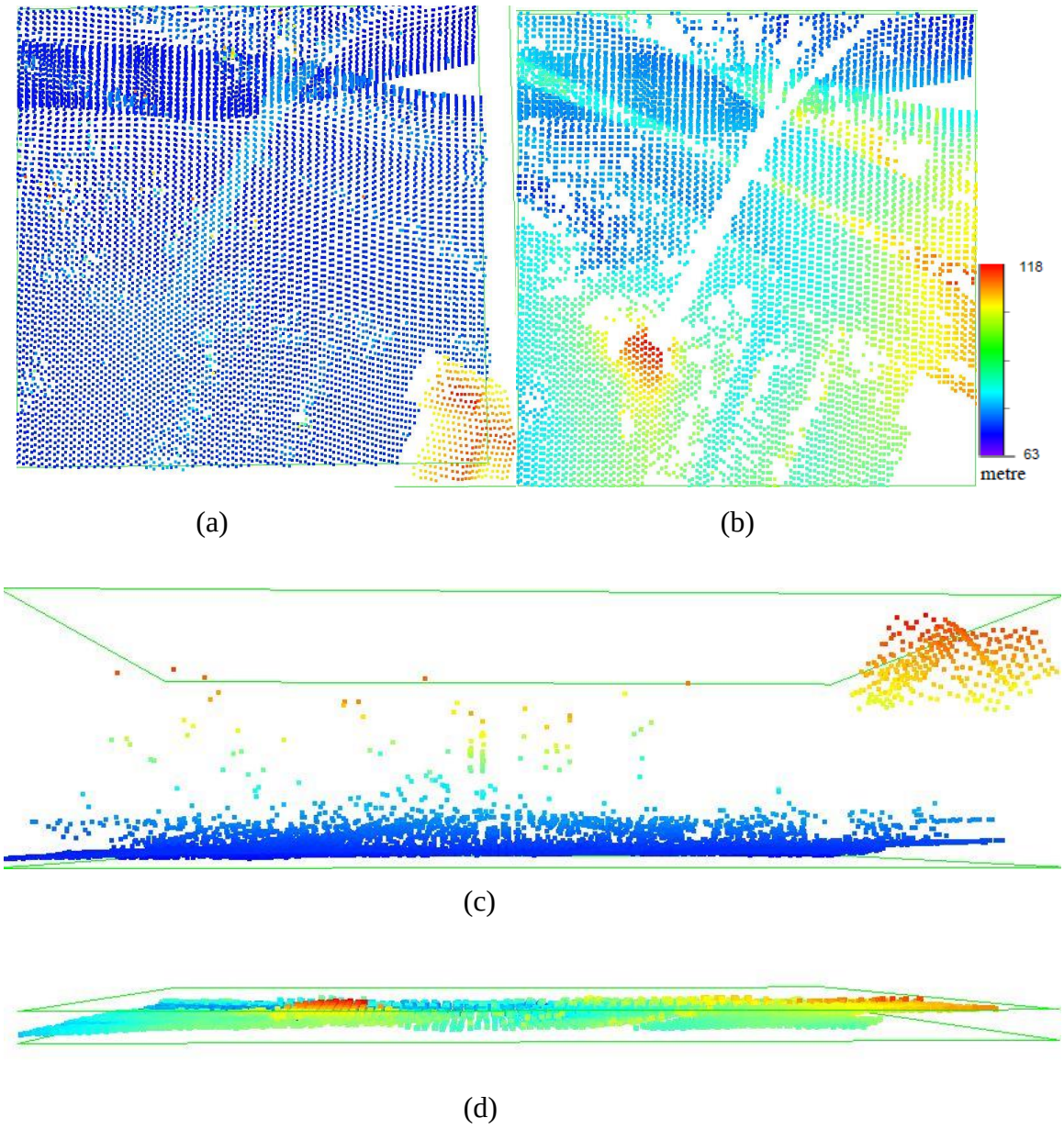


Fig. 42. Sample 21 (a) Original data, (b) Filtered data,(c) Side view (original), (d) Side view (filtered).

Samp 22

Sample 22 (Fig. 43) has a gap in the ground surface with several houses and buildings. Since the proposed method utilizes the height difference and the neighborhood information, these types of problems are easily solved.

Samp 23

Sample 23 (Fig. 44) presents the most difficult challenge. The scene has a plaza and several blocks of buildings. There is a lower walkway in the center of the plaza also. In this test, the plaza and walkway were presumed to be the ground point since it is possible to walk without any hindrance from the plaza to the roads. The difficult part of this scene was well maintained and filtered by the proposed method.

Samp 24

This sample (Fig. 45) also has quite a lot of vegetation and hillside buildings. These are effectively filtered by the proposed method. Since vegetation has abrupt height differences, the proposed method can easily perform the filtering process.

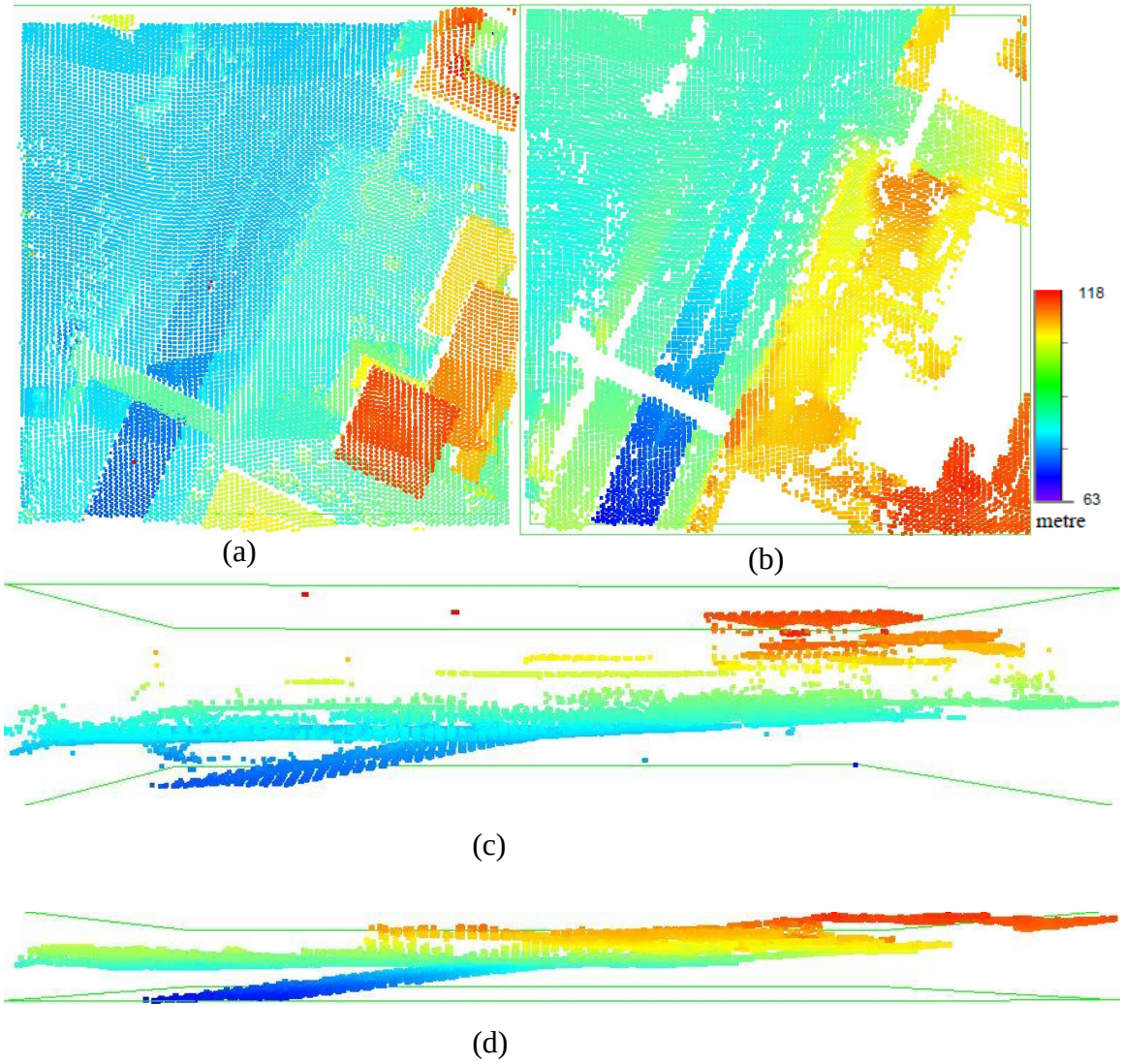
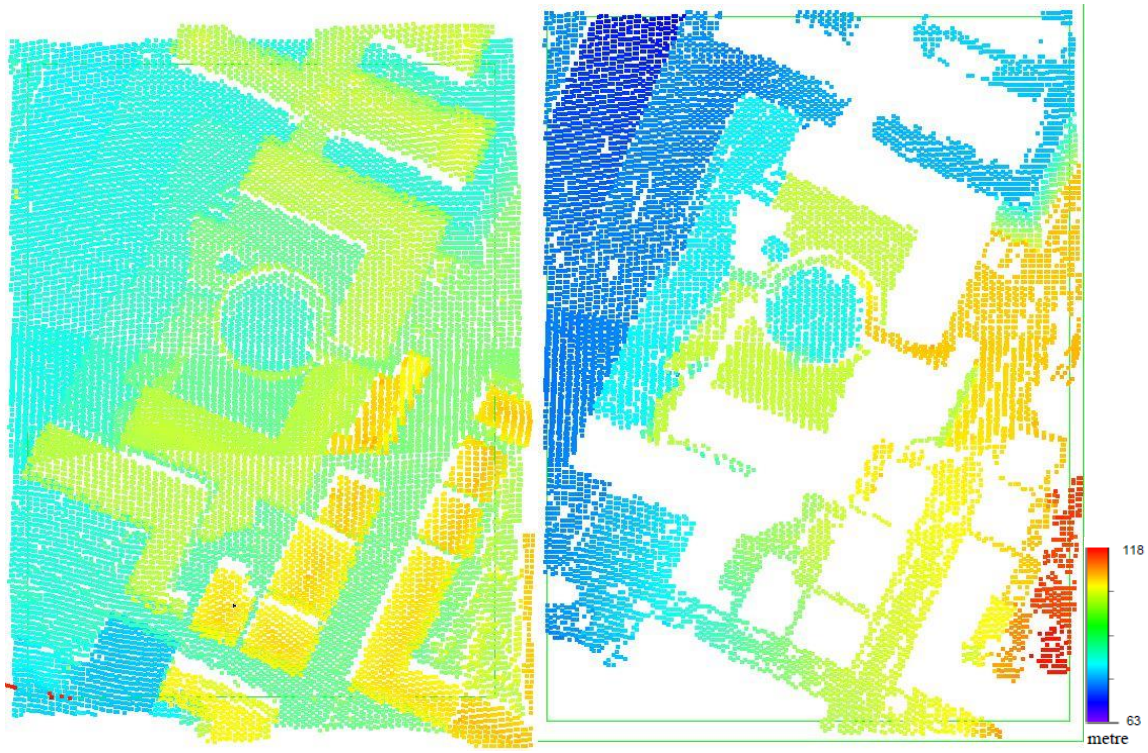
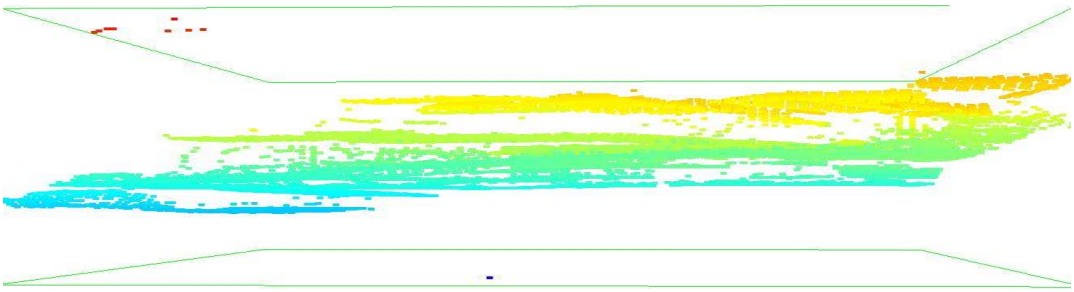


Fig. 43. Sample 22 (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).



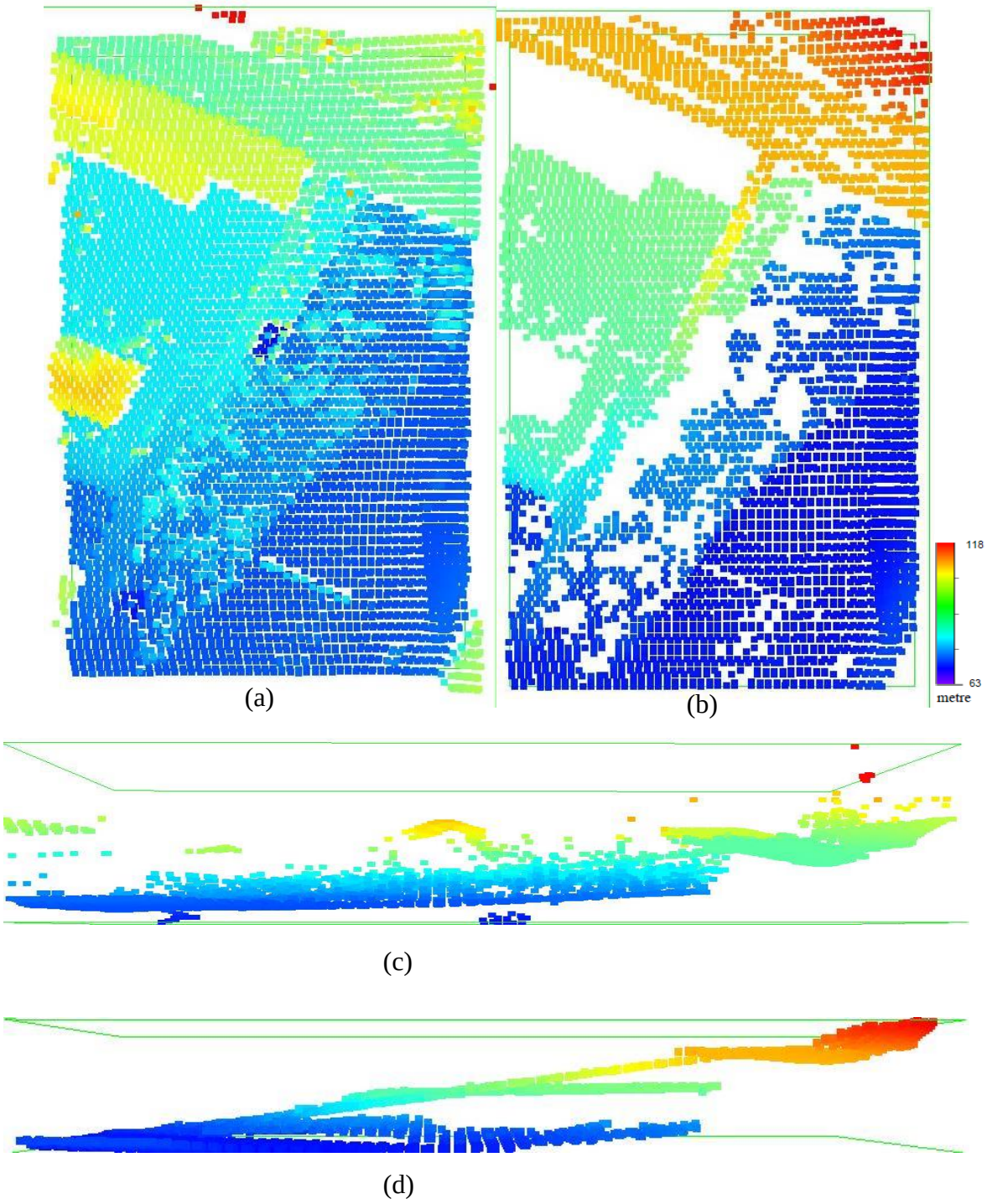
(a) (b)



(c)

(d)

Fig. 44. Sample 23 (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).



(a) (b)

(c)

(d)

Fig. 45. Sample 24 (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

Samp 31

The outliers present here (both high and low) (Fig. 46) are relatively insignificant and therefore their contribution to Type I and Type II errors are minor. However, they can show an important part in filtering the ground and non-ground objects. The proposed method could successfully identify most of the non-ground objects for this area.

Samp 41

In this particular scene 41 (Fig. 47), there are many low outliers (apparently caused by a skylight in one of the roofs). The proposed method performed moderately for this scene. There are several points exist in a group which was treated as ground objects.

Samp 42

In sample 42 (Fig. 48), twelve railway stations can be observed. Since there are few extended and low objects with sparse ground points, some of the points are not removed and considered ground points. This is one of the challenging sample data of this reference group.

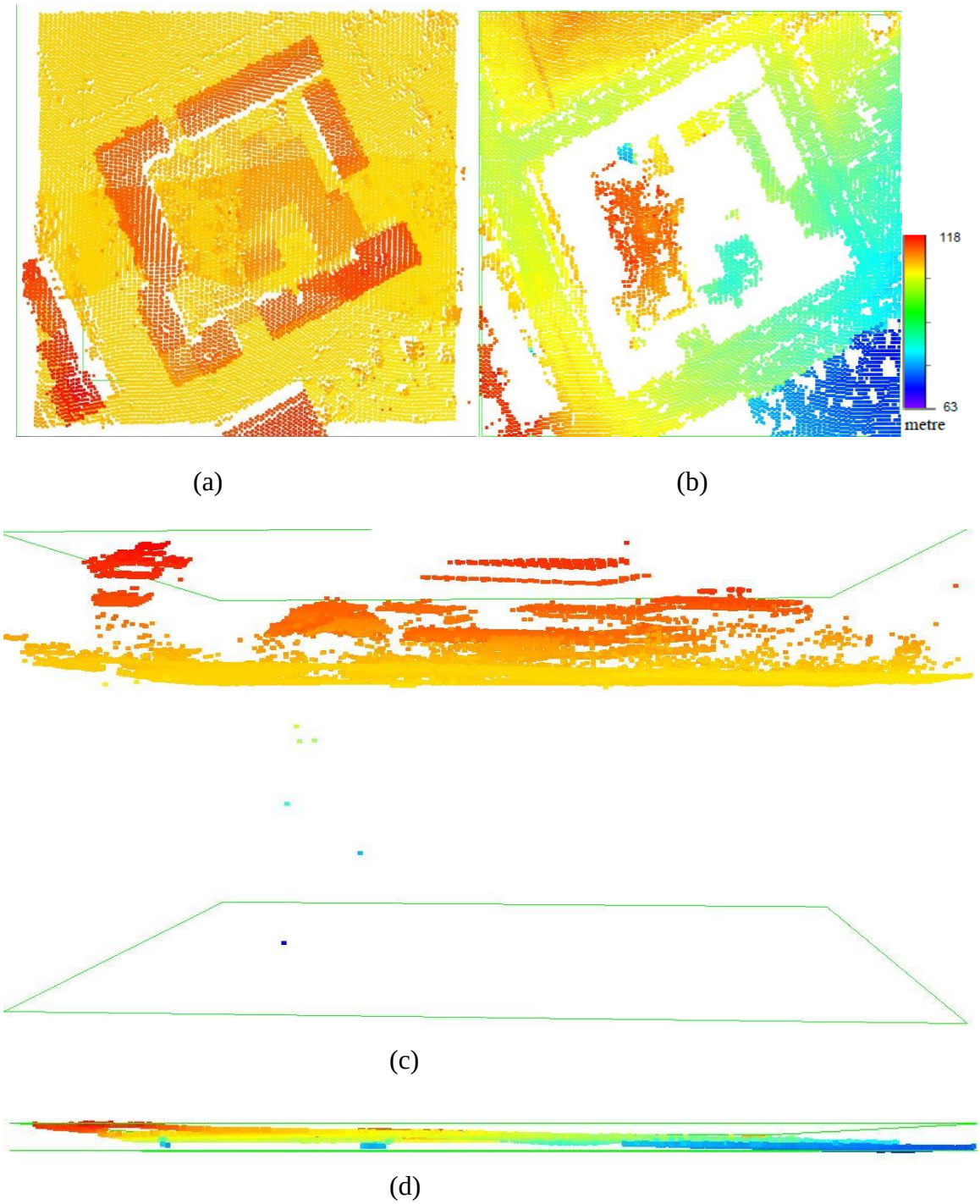
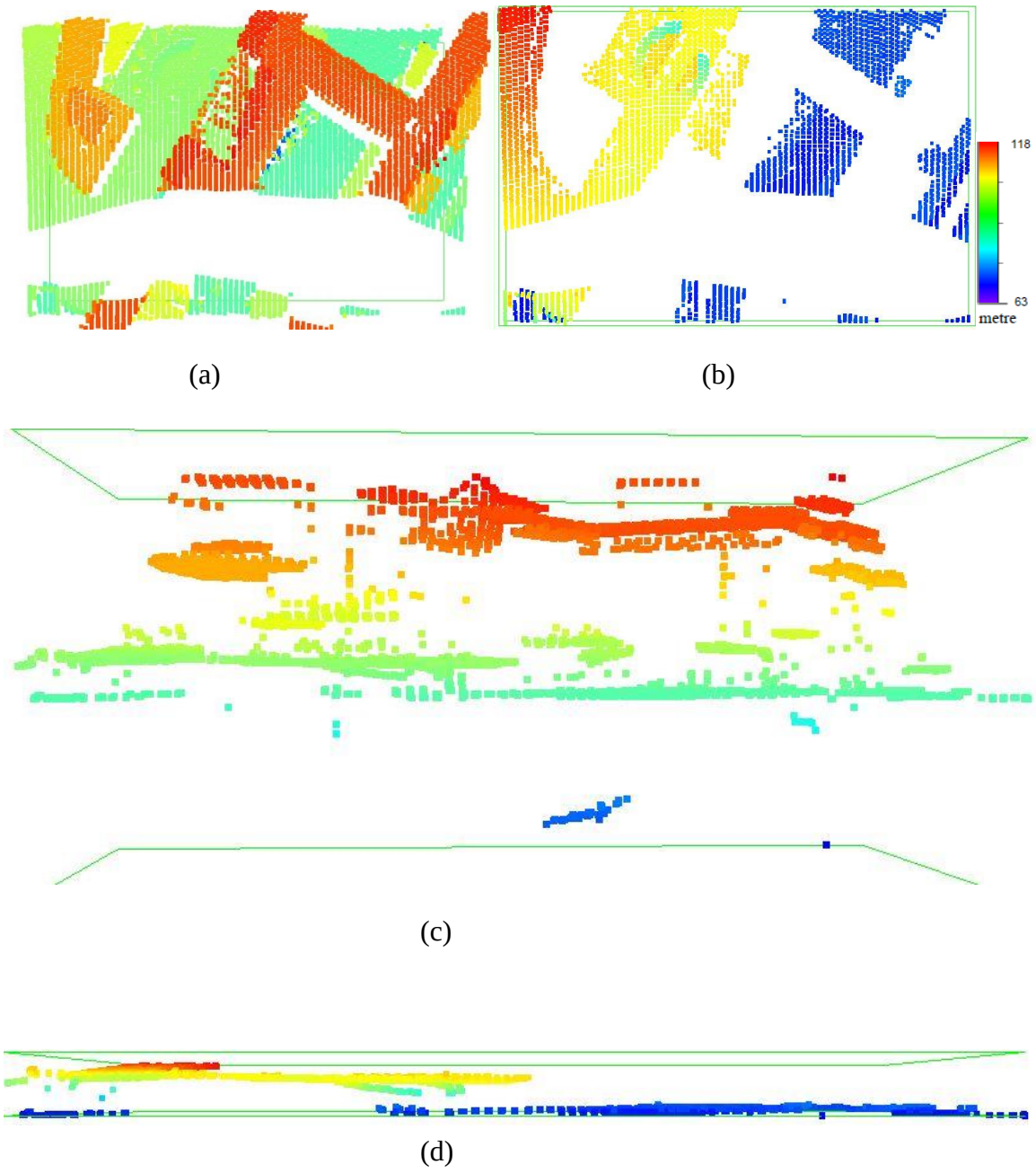


Fig. 46. Sample 31 (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).



(a) (b)

(c)

(d)

Fig. 47. Sample 41. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

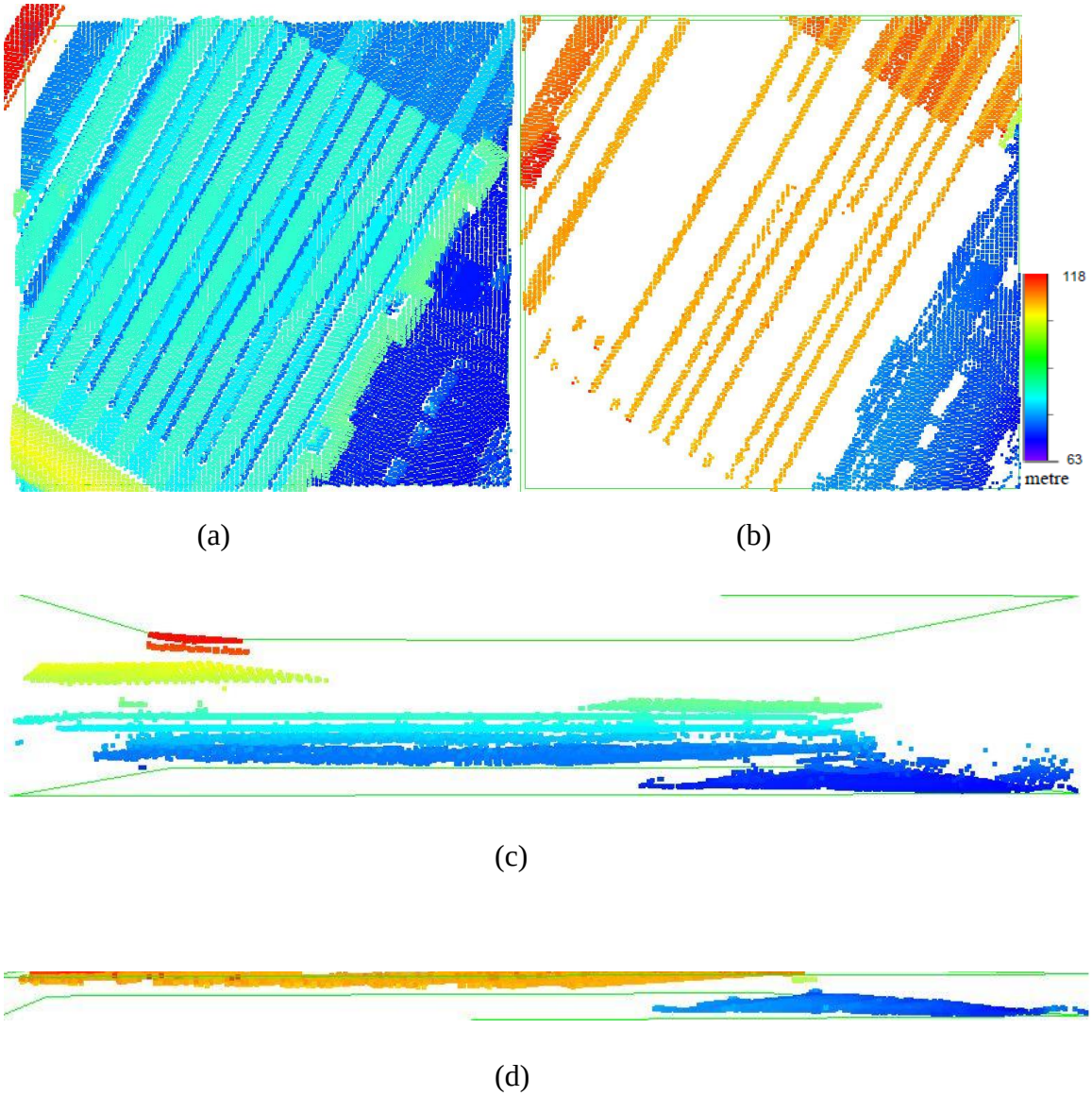


Fig. 48. Sample 42. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

Samp 51

This sample (Fig. 49) has a data gap and low vegetation on a slope. Most of the non-ground objects are identified. Few close to the ground object may be misclassified as a bare earth due to the data gap.

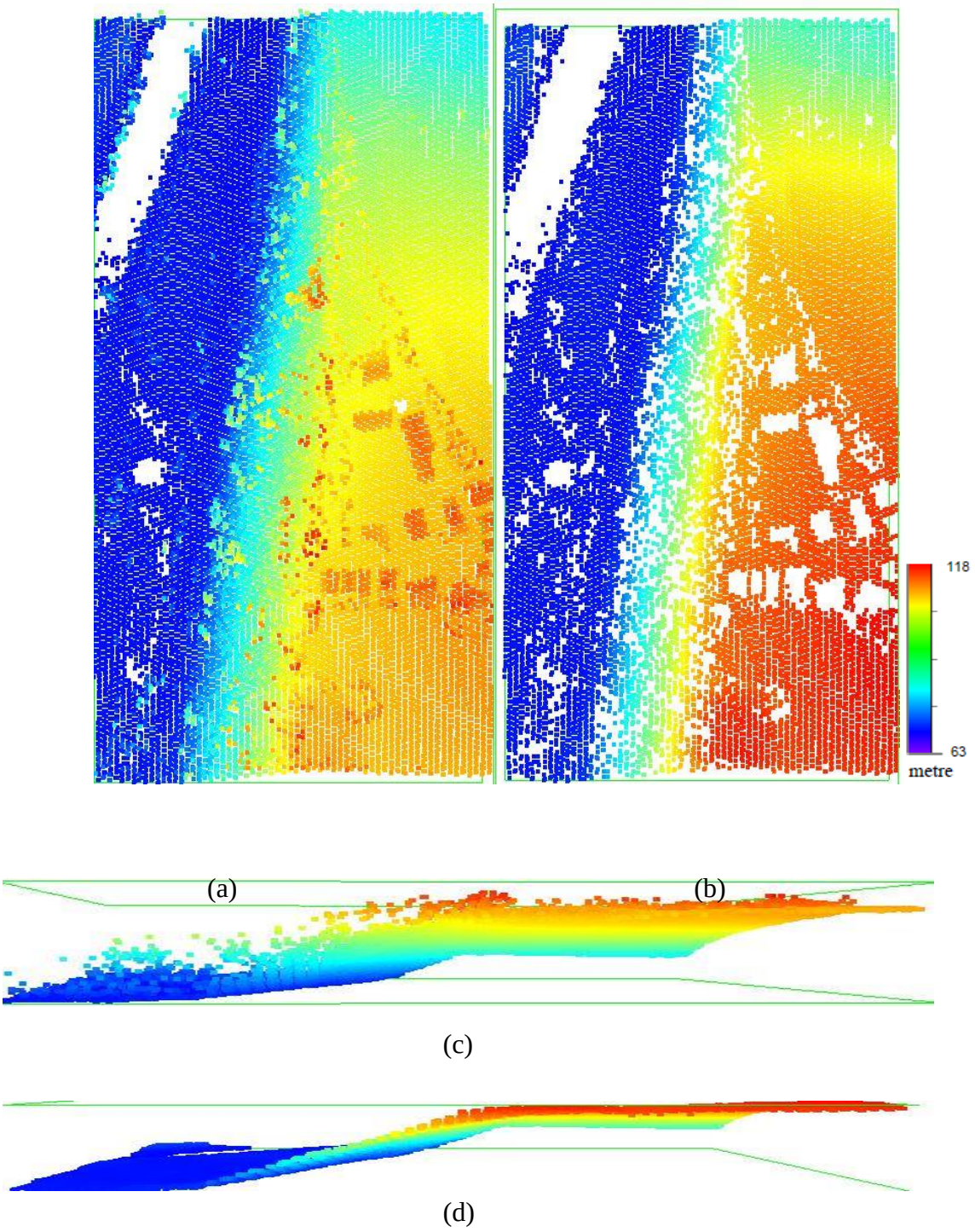


Fig. 49. Sample 51. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

Samp 52

Numerous terrain structures and extreme elevation changes and discontinuity are present in this area (Fig. 50). Most of the ground and non-ground objects are classified accordingly. Some points near the border are clustered which are misclassified as bare earth surface.

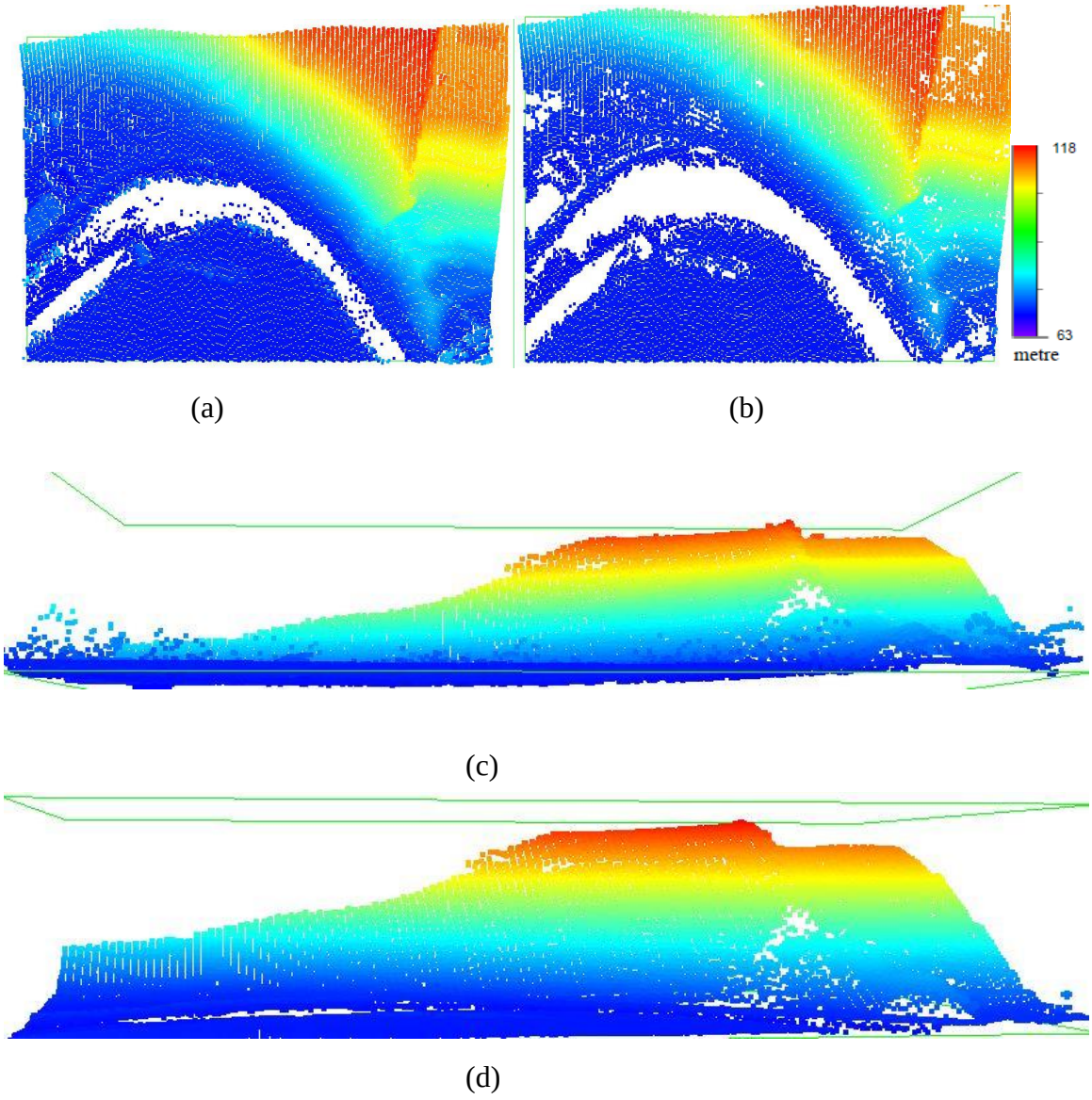


Fig. 50. Sample 52. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

Samp 53

Samp 53 (Fig. 51) consists of numerous slopes in the region. Since this is a scene taken from a mine which features steep and highly stepped slopes, most algorithms tested against this sample performed poorly. The AVM identified most of the ground points, but still, some of the non-ground objects are misclassified as ground in this sample. The large discontinuities in the surface due to the terrace are most likely responsible for these misclassifications.

Samp 54

The overall point cloud density in this region is low (Fig. 52). So, the elevation of the low objects is barely identifiable. The method misclassified some non-ground points as ground in this sample region.

Samp 61

Samp 61 is another challenging scene where most of the points are ground, and few outliers and low earth outliers are present there (Fig. 53). The method based on AVM successfully identifies the ground and the non-ground object except some points that are considered to be ground but essentially they are non-ground points.

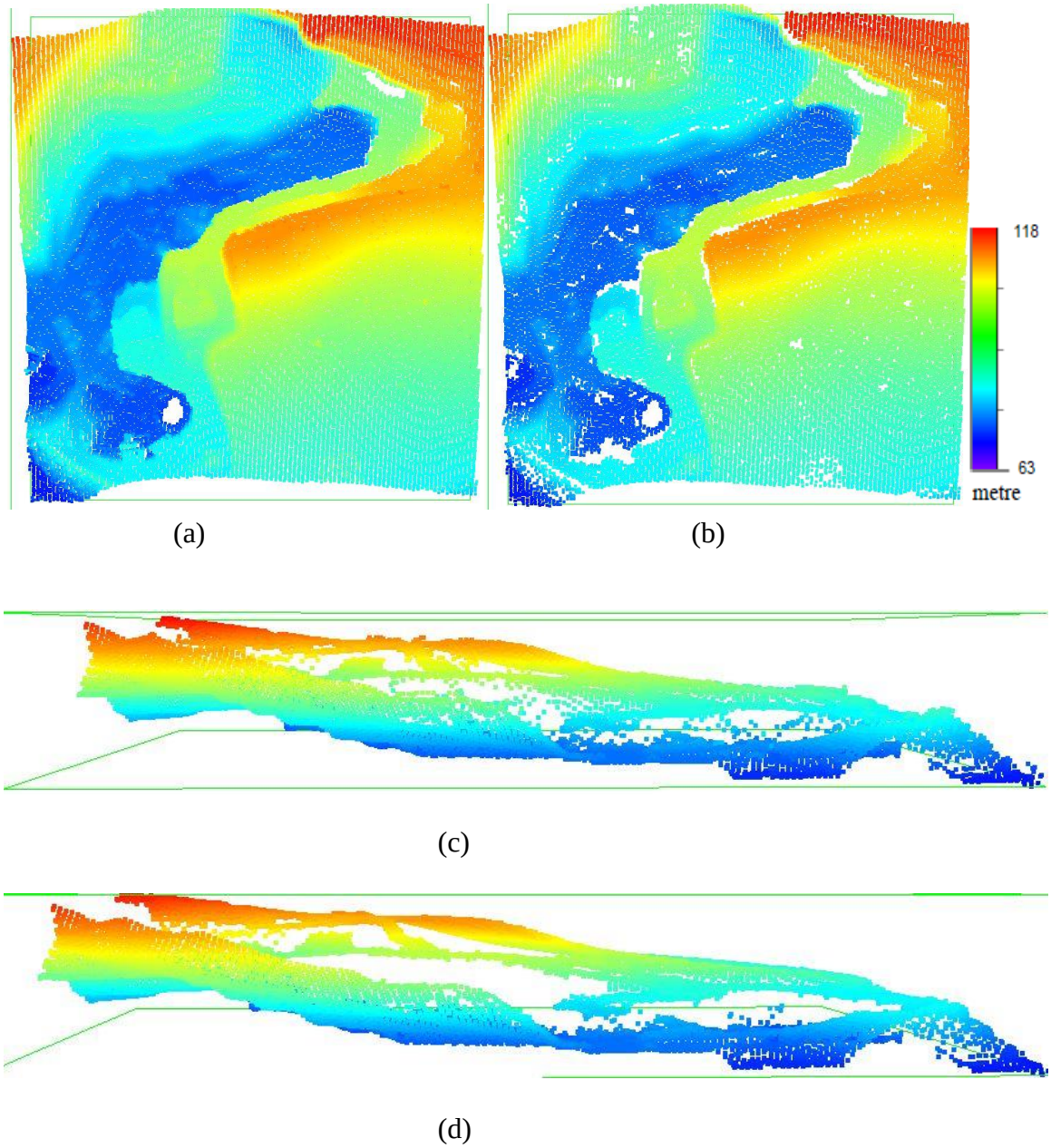


Fig. 51. Sample 53. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

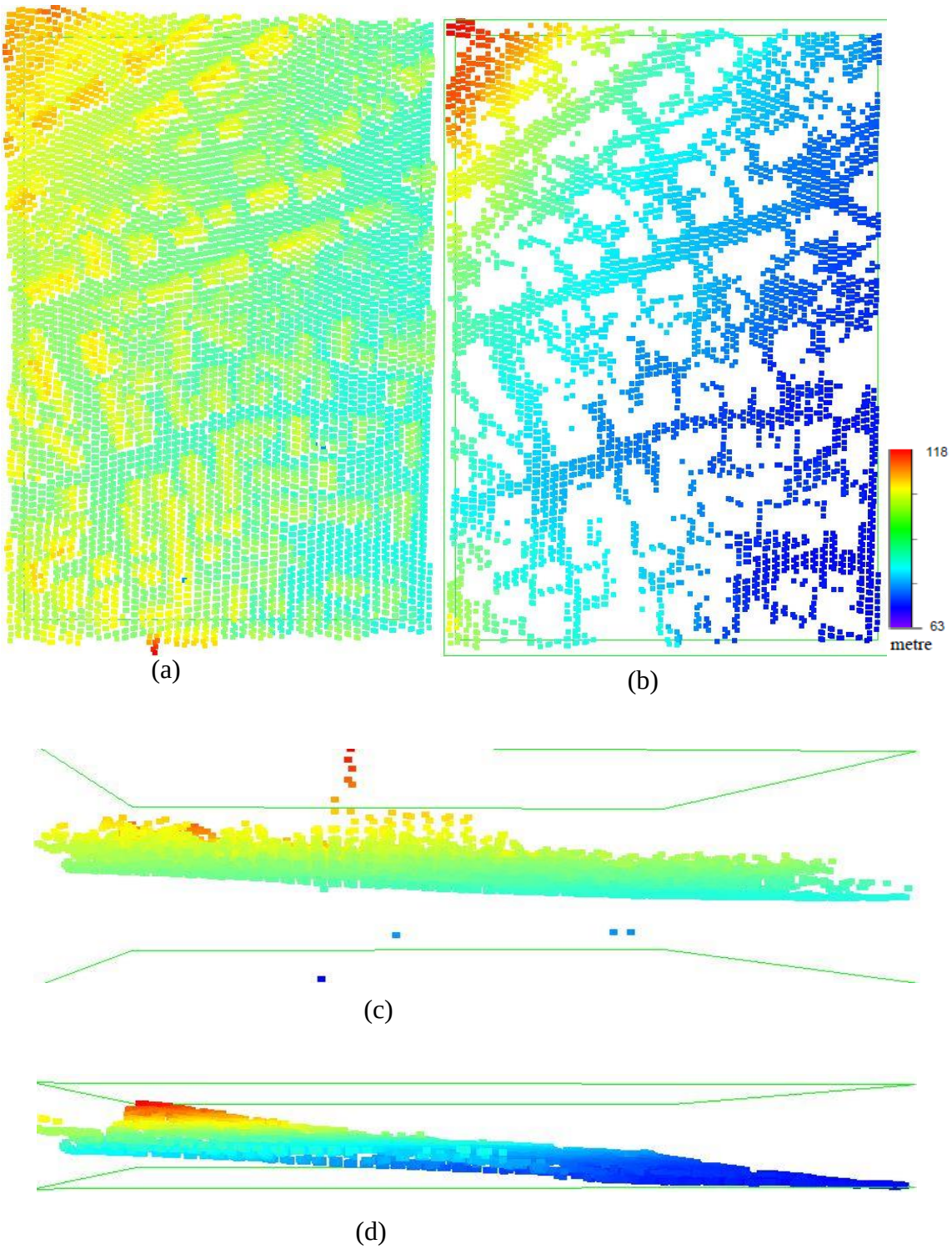


Fig. 52. Sample 54. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

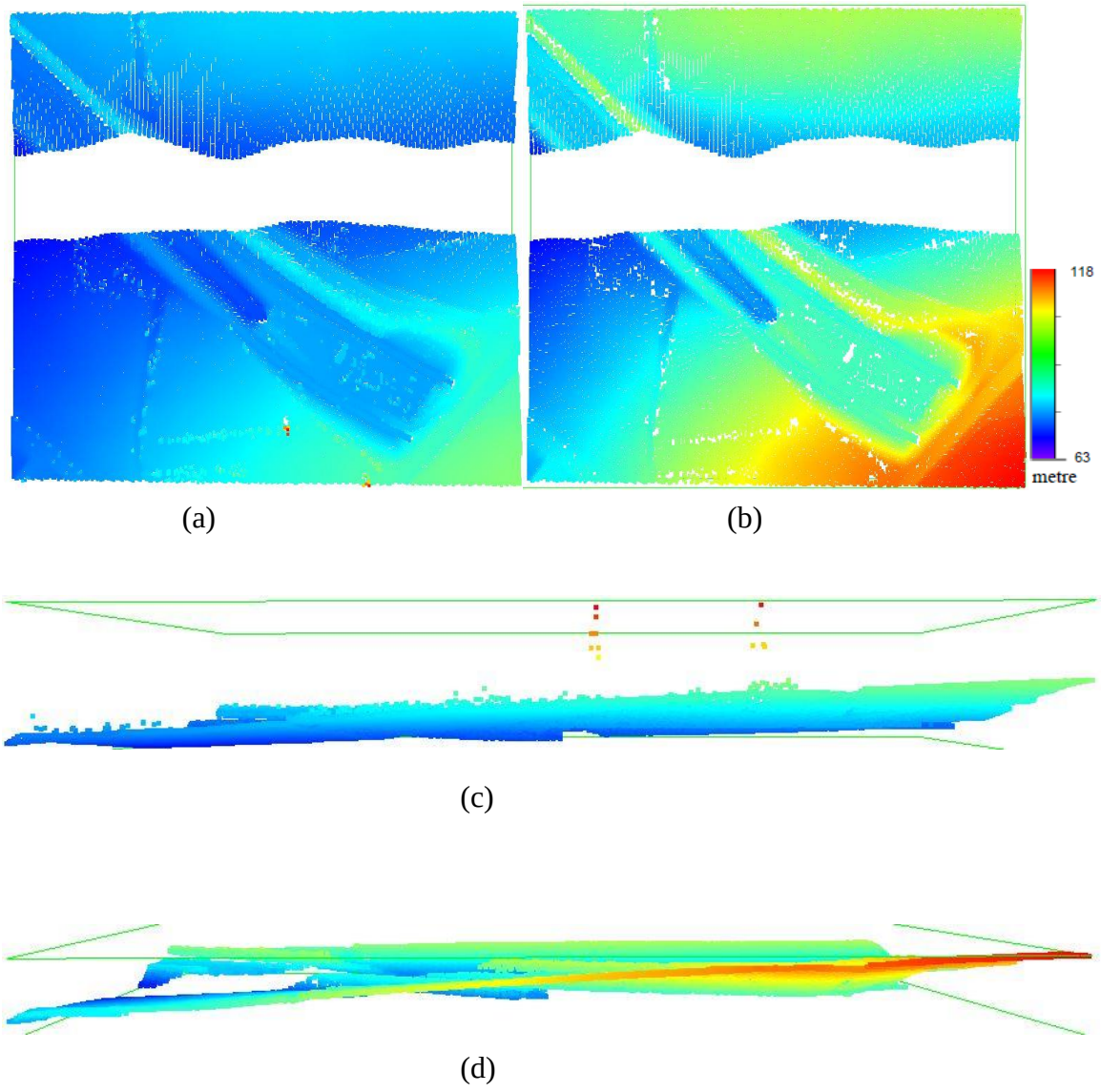


Fig. 53. Sample 61. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

Samp 71

Samp 71 (Fig. 54) has some difficult objects to identify such as a bridge. The bridge is identified as an object in the ISPRS reference datasets, but the adjacent road is treated as ground or bare earth. The algorithm successfully handles the situation and identifies the bridge as an object and the road adjacent to the bridge as ground. The bridge and the river underneath the bridge have significant elevation differences than the surrounding.

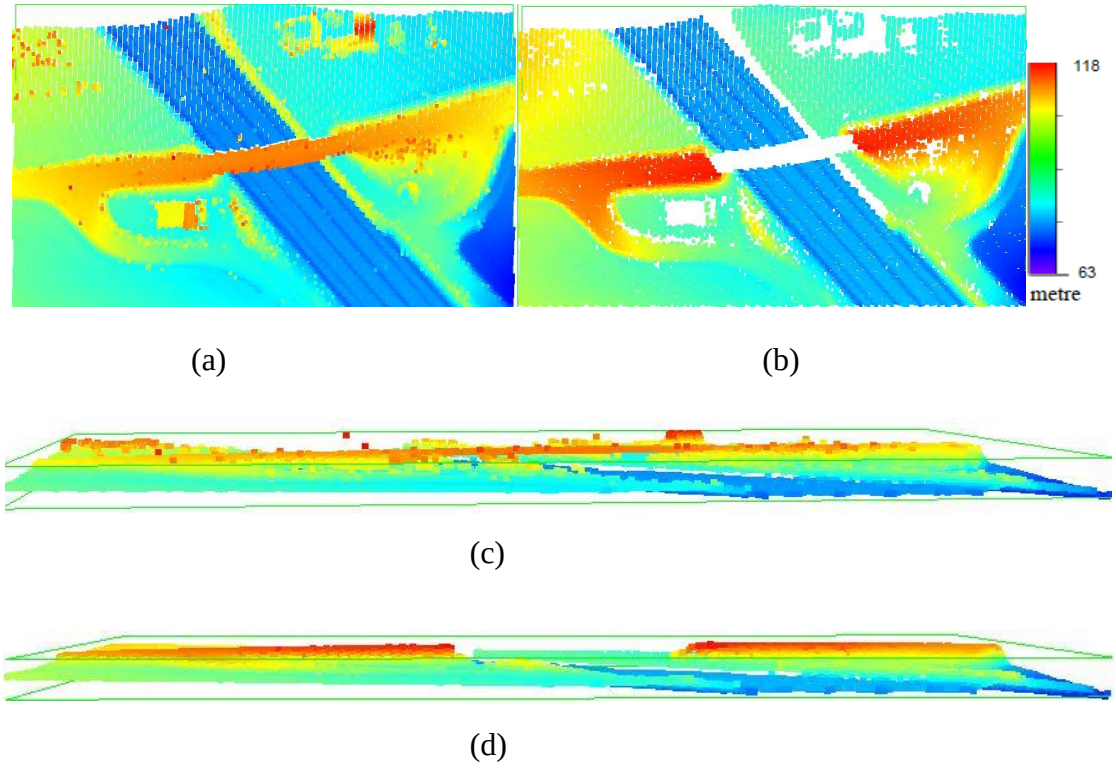


Fig. 54. Sample 71. (a) Original data, (b) Filtered data, (c) Side view (original), (d) Side view (filtered).

TABLE V

PARAMETERS FOR AVM AGAINST ISPRS REFERENCE DATASET

Sample	Maximum Window Size (m)	Height Difference threshold(m)
11	17	0.44
12	14	0.30
21	20	0.58
22	20	0.36
23	14	0.5
24	10	0.21
31	15	0.24
41	15	1.12
42	20	1.04
51	20	0.35
52	15	0.25
53	10	0.11
54	10	0.15
61	15	0.5
71	15	0.75

TABLE V shows the parameter values (maximum window size, height difference threshold) for each of the fifteen samples.

5.5.3 Error Analysis

There are several difficult scenarios in these presented sample data. These situations relate to outliers in the data, object complexity, objects that are attached to the terrain, vegetation, and discontinuities in the bare-earth surface. The points that do not belong to the original surface area and are generated from multi-path errors by laser are called low outliers. Other objects like birds, low-flying aircraft, etc. are called high outliers. In some cases, the size of the objects is not consistent (very large, very small, very low, complex shape, disconnected terrain, etc.). Other difficult situations arise where there is a building on the slope, bridges, ramps, low vegetations, sharp ridges, etc. Fig. 55 illustrates some of these filtering difficulties.

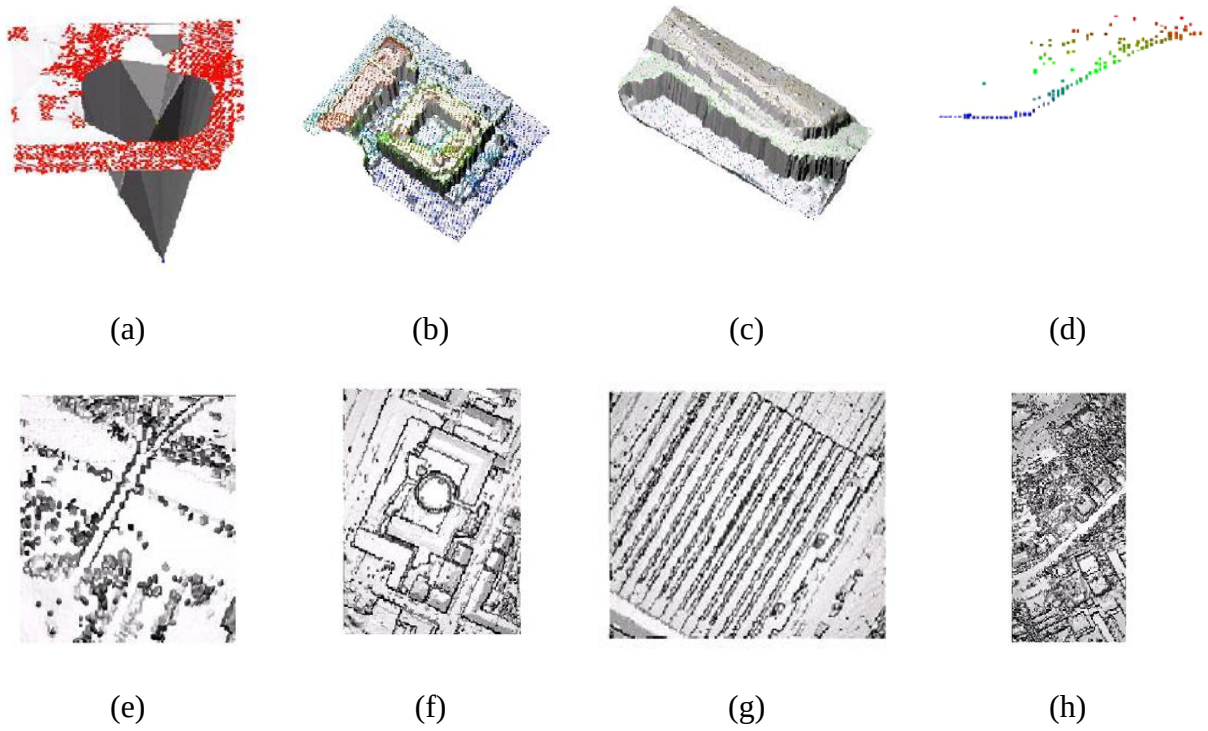


Fig. 55. Filtering difficulties [48]. (a) Erosion caused by low outlier, (b) complex configuration, (c) steep slope, (d) vegetation on slope, (e) bridge, (f) complex urban scene, (g) railway station, (h) steep slope with buildings and dense vegetation.

Quantitative assessment

Cross-matrices and visual representations are two main factors for the quantitative assessment for the 15 subsets of the dataset. The cross matrices were then used to evaluate Type I (or false positive, rejection of bare-earth points) and Type II (or false negative, acceptance of object points as bare-earth) errors, and visual representations were then used to define the association between Type I and Type II errors to features in the site. For each of the samples a cross-matrix is presented graphically given below (Fig. 56):

		Filtered	
		Bare Earth(BE)	Object(Obj)
Reference	Bare Earth(BE)	a	b
	Object(Obj)	c	d

Fig. 56. Cross matrix.

where

- a is the count of bare earth points that have been properly identified as bare earth or ground points.
- b is the count of bare earth points that have been falsely identified as object or non-ground points (contribute to Type I errors).
- c is the count of object points that have been falsely identified as bare earth (contribute to Type II errors).
- d is the count of object points that have been properly identified as object.

Type I, Type II and Total error is computed using equations 17, 18 and 19.

$$\text{Type I error} = \frac{b}{a+b} \times 100 \quad (17)$$

$$\text{Type II error} = \frac{c}{c+d} \times 100 \quad (18)$$

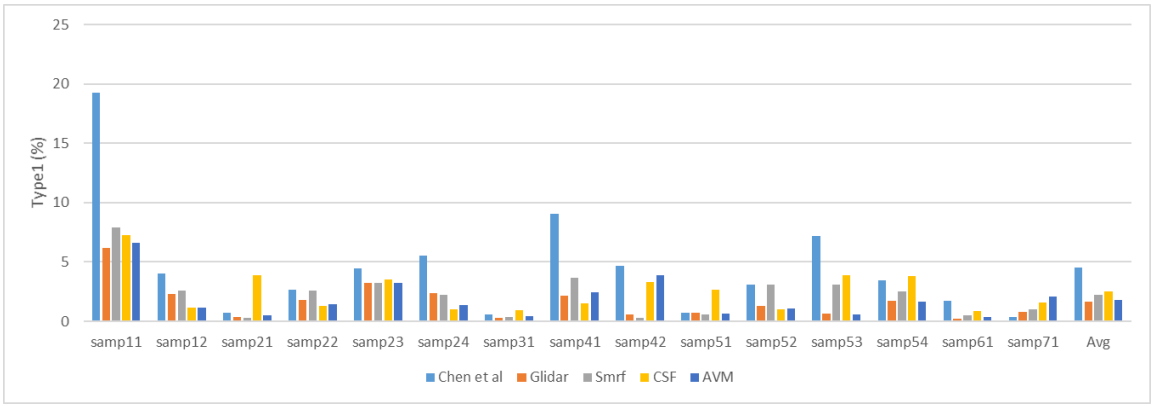
$$\text{Total error} = \frac{b+c}{a+b+c+d} \times 100 \quad (19)$$

The proposed method was compared with several algorithms for ground data filtering (Chen et al. [59], Mongus et al. [49], Pingel et al. [55], Zhang et al. [53]). The comparison shows (Fig. 57) that the proposed method tends to suppress the omission error (Type I error) and achieve a relatively lower average total error. However, the commission error (Type II error) is reasonable compared to the other methods. Thus, our method classifies more non-ground as ground points than the other four methods, while fewer ground points are removed from the dataset as they are classified as non-ground points.

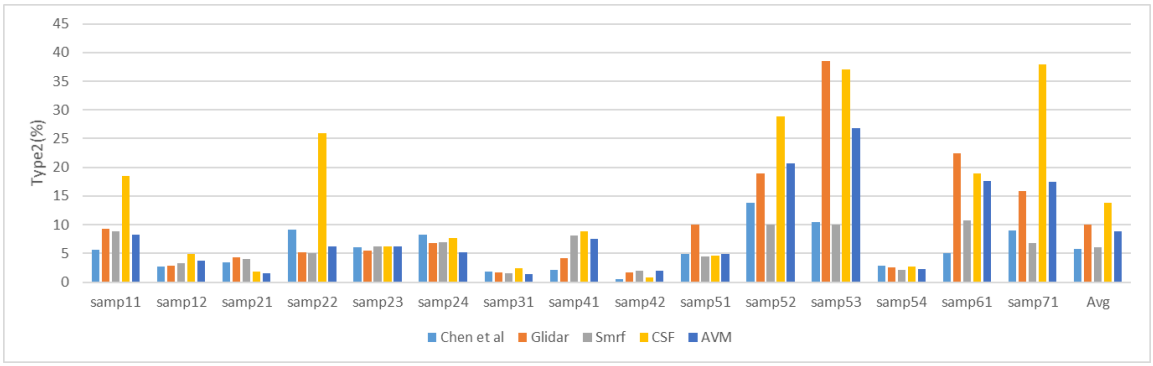
Fig. 58 shows the performance of AVM compared with Chen et al., Mongus et al., Pingel et al. and Zhang et al. in terms of all error types. For Type I error comparison, AVM has five lowest error rates and one highest error rate. AVM generates the lowest error rate for Samp 12, 23, 51, 53 and 54. For Type II error, the position of AVM is third to generate the lowest rate. Samps 21, 24 and 31 got the lowest Type II error with the AVM method. For the total error type, AVM obtains a relatively lower rate. More specifically, AVM generates 2.35%, 2.75%, 4.32%, 0.9%, 1.41%, 2.18% for samp 12, 22, 23, 31, 51 and 53 respectively.

Three error measures (Type I, Type II, Total) have been used to assess the quality of the filter results. To some extent, there should be a choice to be made between minimizing Type I and Type II errors. The problem is which error to minimize depends on the cost of the error for the application that will use the filtered data. However, it will also depend very much on the time and cost of fixing the errors manually, which is often done during quality control. Experience

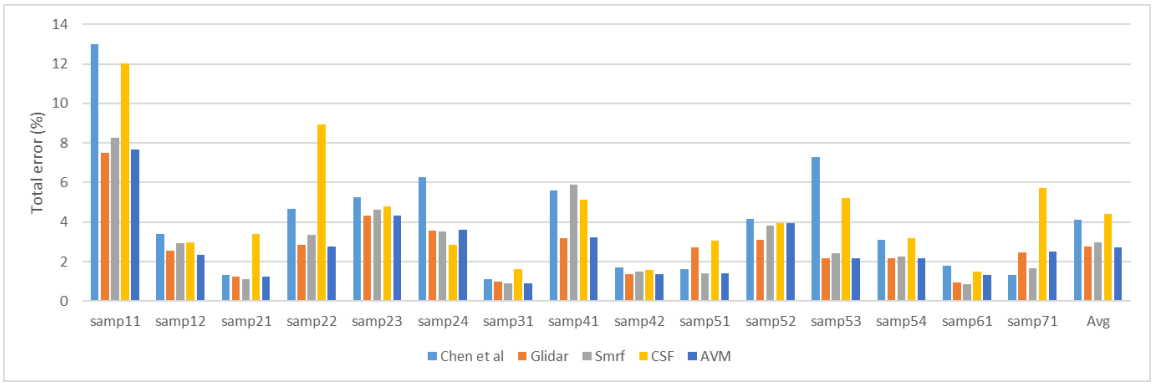
with manual filtering of the data showed that it is far easier to fix Type II errors than Type I errors. Firstly, there will generally be fewer Type II than Type I errors. Secondly, Type II errors are noticeable since they stand out in their neighborhoods. According to the report of Sithole and Vosselman [48], filtering should be biased in favor of minimizing Type I errors, because Type II errors are easier to correct manually during quality control.



(a)



(b)



(c)

Fig. 57. Comparisons of error types (a) Type I, (b) Type II, (c) Total errors (%) compared with Chen et al., Mongus et al., Pingel et al. and Zhang et al. for filtering the International Society for Photogrammetry and Remote Sensing (ISPRS) datasets.

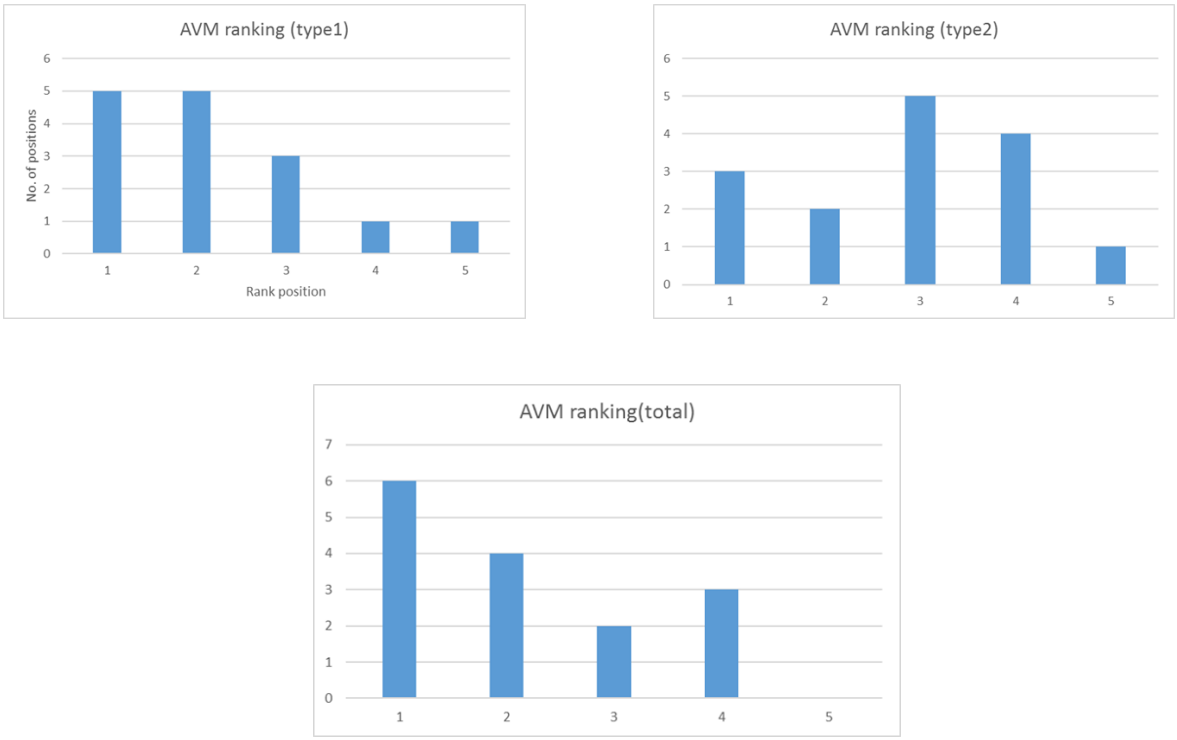


Fig. 58. The ranking order of AVM (type I, type II, total error).

Besides the error rate comparison, we compared the kappa coefficients [138] with some existing top algorithms.

Fig. 59 shows the calculation of the kappa coefficient and TABLE VI shows the interpretation of the kappa coefficient values.

TABLE VII illustrates the performance of several algorithms in terms of the Kappa coefficient along with AVM. The highest among all for all the samples is highlighted.

		Predicted		
		Number of instances classified correctly	Number of instances classified wrong	Subtotal
Actual	Number of instances classified correctly	A	B	A+B
	Number of instances classified wrong	C	D	C+D
	Subtotal	A+C	B+D	

Observed accuracy = (A+D)/Total

Expected accuracy = (((A+B)*(A+C)/Total) + ((C*D)\Total))/Total

Kappa = (Observed accuracy) – (Expected accuracy)) / (1 – (Expected accuracy))

Fig. 59. Kappa Coefficient Calculation.

The interpretation of Kappa can be listed as below:

TABLE VI

INTERPRETATION OF KAPPA

Kappa	Agreement
<0	Poor agreement
0.01-0.20	Slight agreement
0.21-0.40	Fair agreement
0.41-0.60	Moderate agreement
0.61-0.80	Considerable agreement
0.81-0.99	Almost perfect agreement

TABLE VII

COMPARISON OF KAPPA COEFFICIENT

	Chen et al	Smrf (Pingel)	CSF	AVM	Axelsson	Elmqvist	Pfeifer	Hui
Samp11	74.12	**83.12**	75.17	74.68	78.48	56.68	66.09	72.92
Samp12	93.23	94.15	94.04	**94.26**	93.51	83.66	91	93
Samp21	96.1	**96.77**	90.47	95.49	86.34	77.4	92.51	93.35
Samp22	89.03	92.21	77.72	**93.03**	91.33	80.3	84.68	87.58
Samp23	89.49	90.73	90.38	**92.00**	91.97	75.59	83.59	89.74
Samp24	84.53	91.13	**92.68**	83.12	88.5	54.13	78.43	81.93
Samp31	97.76	98.17	96.75	**98.67**	90.43	89.31	96.37	97.33
Samp41	88.83	88.18	**89.73**	88.63	72.21	82.46	78.51	78.78
Samp42	95.81	**96.48**	96.18	95.91	96.15	90.86	93.67	95.38
Samp51	95.17	95.76	91.13	**95.64**	91.68	52.74	89.61	85.06
Samp52	78.91	81.04	77.05	75.23	**83.63**	9.36	41.02	69.51
Samp53	46.69	68.12	46.86	**69.14**	39.13	7.05	30.83	41.84
Samp54	93.9	**95.44**	93.61	92.01	93.52	55.88	88.93	91.63
Samp61	77.36	**87.22**	78.1	73.49	74.52	10.31	47.09	67.82
Samp71	**93.19**	91.81	68.03	71.28	91.44	26.26	75.27	79.86
Avg	86.2746	90.022	83.86	86.238	84.18933	56.7993	75.84	81.71533
Median	89.49	91.81	90.38	92	90.43	56.68	83.59	85.06
Min	46.69	68.12	46.86	69.14	39.13	7.05	30.83	41.84
Max	97.76	98.17	96.75	98.67	96.15	90.86	96.37	97.33
Std	13.1712	7.84638	13.57596	10.5820	14.39149	30.2024	20.56417	14.44962

Overall, the accuracy of the proposed method is close to some top filtering algorithms. The results show that AVM has the relatively good performance for Samp 12, Samp 22, Samp 23, Samp 31, Samp 51, and Samp 53. For the reference dataset, AVM performs well for both rural and urban areas. Specifically, AVM shows better performance where the data consists of building on a hillside, large, irregularly shaped buildings, a mixture of vegetation and buildings, large buildings, bridges. However, with the scene of a steep slope, railway station with trains, AVM could not show significant performance regarding error rate and Kappa coefficient.

The next chapter describes the color mesh sharpening method based on Laplace-Beltrami discretizations. Several discretizations are utilized, and results are illustrated in this chapter.

CHAPTER 6

COLOR MESH SHARPENING

In this chapter, the methodology for mesh color sharpening using discrete Laplace-Beltrami operator and the results are described.

6.1 Introduction

Three-dimensional (3D) meshes are widely used in many fields and applications, such as computer graphics, games, animation films, and virtual reality. 3D meshes are usually generated using one of two methods: 1) artists create the meshes from scratch with 3D modeling software, such as Autodesk Maya and Google SketchUp; or 2) the meshes are created by scanning real 3D objects. The second method is becoming more popular because of the increasing precision and processing power of 3D scanners with the reduced cost at the same time. 3D scanners collect data from the shape and color appearance of a real object or environment. The collected data are later processed to generate a 3D model of the real object. A wide range of commercial 3D scanners has been developed offering varied capabilities in terms of scanning range, precision, and speed. Among them, Microsoft Kinect is a motion-sensing device used by Microsoft Xbox 360 and Xbox One game consoles and Windows PCs and is becoming very popular for scanning objects for various applications. One major advantage of Kinect is its low cost with a price of $150, compared with scanners with typical prices of thousands or tens of thousands of dollars. One of the best available 3D scanning applications that utilize Kinect is ReconstructMe [139]. ReconstructMe creates color meshes with each vertex of the mesh containing position, normal, and color information. To improve the quality of color meshes, two approaches can be utilized:

geometrical processing and color (appearance) processing. Geometrical processing changes each vertex's position while keeping its color information intact; on the other hand, color processing changes each vertex's color while keeping its position (or the object shape) intact. Existing mesh processing methods have been focused on improving the geometrical properties of the meshes.

6.2 Motivation

Image sharpening is an important tool to improve the image quality. Image sharpening emphasizes texture and enhances the contrast of the image. Sharpening filters make the edges of an image appear more defined by darkening the low-intensity pixels and brightening the high-intensity pixels. This creates a crisp edge between bright and dark portions of the image, producing more contrast. With advances in 3D scanning hardware, more and more colored meshes are being generated. Especially with the increasing availability of low-cost 3D scanners such as Microsoft Kinect, colored 3D meshes become more accessible. To the best of our knowledge, no image processing techniques, such as sharpening and Laplace-Beltrami operator, have been combined to improve the visual appearance of 3D colored meshes. This dissertation extends traditional image sharpening techniques for 2D regular images to 3D color meshes with irregular topologies. In particular, this work [140] utilizes several discretizations of the Laplace-Beltrami operator for mesh color sharpening. Several definitions and implementations of the Laplacian-Beltrami operator were investigated for their efficiency and effectiveness for mesh color sharpening. Different ways to discretize the Laplace-Beltrami have been developed by defining the discrete operator polygon-wise on a triangle mesh, with the most prominent one being the cotan-operator, defined by Pinkall et al. [141]. Meyer et al. [142] used Voronoi area, Mayer et al. [143] used the sum of areas of triangles over vertices, and Desbrun et al. [144] used

two different approaches. One approach of the Desbrun et al. paper used the gradient of the normalized area, and the other used the sum of the cotan operator over the edges. These discretizations of the Laplace-Beltrami operator that were previously defined for computational fluid dynamics and mesh geometric processing are extended in this paper for color sharpening, thus providing several new tools for improving the quality of 3D meshes. The color of each vertex in the polygonal mesh is updated using various implementations of the Laplace-Beltrami operator. The performance of various implementations is compared and analyzed for the best approach of mesh color sharpening.

6.3 Image Sharpening

Sharpening is commonly used in image processing to highlight transitions (or edges) in intensity. The main goal of image sharpening is to enhance the image and make the image to appear clearer and brighter. Various image-sharpening filters have been proposed using the first-order and second-order derivatives. The Laplacian, which is a second-order derivative, is commonly used for image sharpening. The Laplacian operator can be defined as a function $f(x,y)$ as follows:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}, \tag{20}$$

where f is image intensity and x, y are pixel positions. Since the Laplacian is a derivative operator, it uses intensity discontinuity in an image and minimizes regions with slowly varying intensity levels. One discrete implementation of the Laplacian operator defined in eq. 20 be written as

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y). \tag{21}$$

Eq. 21 is used to apply to the image as a filter mask. This filter mask can be represented as a grid

which is shown in Fig. 60(a). Fig. 60(b) shows an alternate implementation of the discrete Laplacian operator where each diagonal term contains additional *-2f (x, y)* term for which *-8f (x, y)* would be subtracted from the difference terms. Fig. 60(c) and Fig. 60(d) are the negatives of the previous implementations.

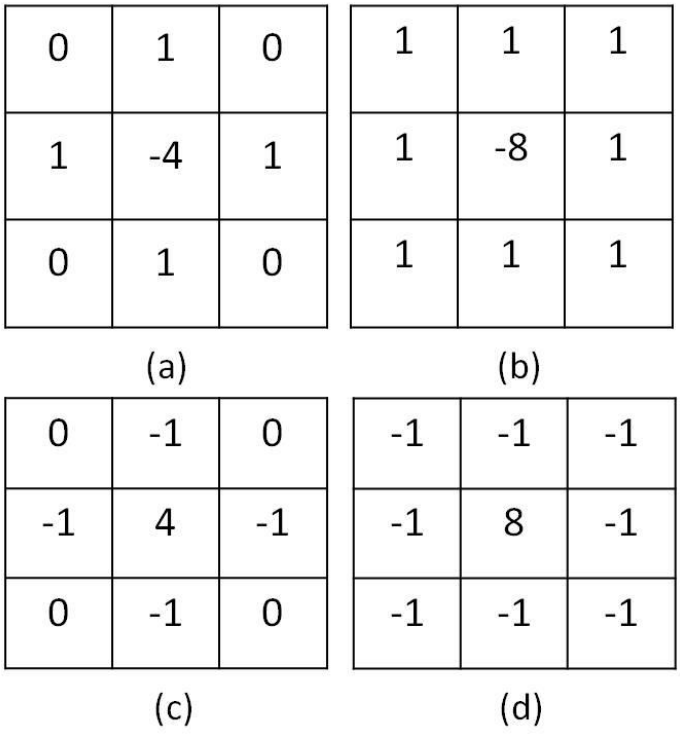


Fig. 60. Filter mask grid. (a) Filter mask to implement Eq. (16), (b) An alternate implementation of Eq. (2), (c) and (d) Two other implementations using negative terms.

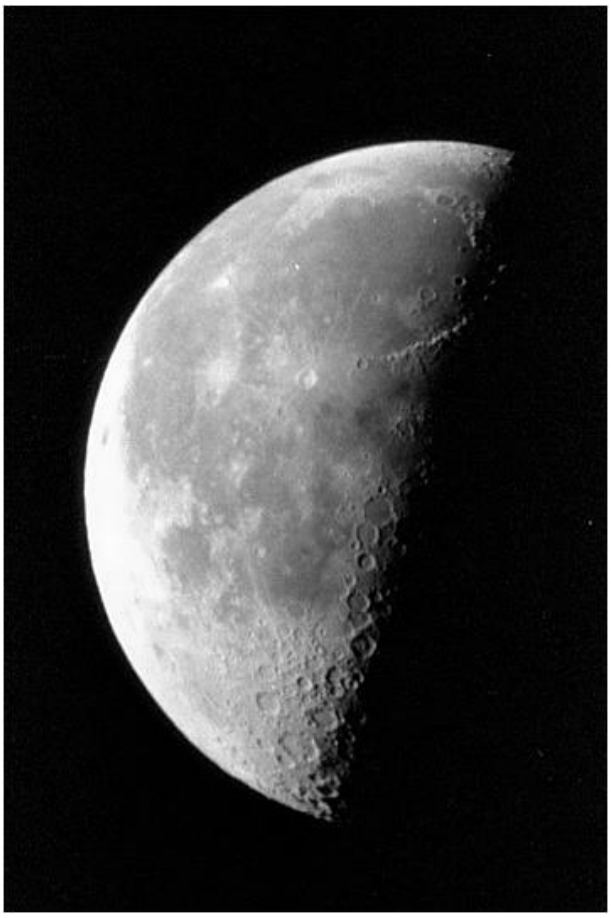
The Laplacian is used for image sharpening using the following formula [123]:

$$g(x,y) = f(x,y) + c[\nabla^2 f(x,y)] \tag{22}$$

where *f (x, y)* and *g (x, y)* are the input and sharpened images, respectively. The constant $c = -1$ if the Laplacian filters in Fig. 60(a) and (b) are used, and $c = 1$ if Fig. 60(c) and (d) are used. Fig. 61 shows the application of this mask for both a grey scale image and a color image. The Moon

image and the color of the pear image appear brighter and sharper. The final results are images with enhanced details and significant improvement in sharpness. The Laplacian is a second order derivative and because the pixels in an image are arranged in a rectangular grid; the discretization of the Laplacian is straightforward [123] and is computed as the second order differences along horizontal and vertical (or diagonal) directions. For most 3D meshes, no such rectangular grids exist, so imaging sharpening methods cannot be directly applied to mesh color sharpening. No such directions (horizontal, vertical, and diagonal) are defined for polygonal or triangular meshes, and Equation 16 cannot be extended directly to 3D meshes. The Laplace-Beltrami operator was proposed as the second order derivative on 3D meshes, which is to be discussed next.

(a)



(b)

(c)

(d)

Fig. 61. Image Sharpening. (a) Original Image- Moon, (b) Sharpened Image- Moon, (c) Original Image- Pear, (d) Sharpened Image- Pear.

6.4 Laplace-Beltrami Operator and Discretizations

The Laplace-Beltrami Operator is mostly utilized in the field of differential geometry to operate on the surfaces in Euclidean spaces. It is a generalization of the second-order derivative operator Laplacian to non-at Riemannian manifolds. Let *f* be a real-valued function defined on a differentiable manifold M with Riemannian metric. The Laplace-Beltrami operator is defined as [145]

$$\Delta f := div(grad\ f) \tag{23}$$

where grad and div are the gradient and divergence on the manifold M [145]. For discrete meshes, the function *f* on a triangular mesh *T* is defined by linearly interpolating the values of $f(v_i)$ at the vertices of *T*. This is done by choosing a base of piecewise linear hat-functions φ_i, with value 1 at vertex v_i and 0 at all the other vertices [146]. Then *f* is given as

$$f = \sum_{i=1}^{n} f(v_i)\varphi_{i.} \tag{24}$$

Discrete Laplace-Beltrami operators are usually represented as [142]

$$\Delta f(v_i) = \frac{1}{d_i}\sum_{j\in N(i)} w_{ij}\left[f(v_i) - f\left(v_j\right)\right]. \tag{25}$$

where *N(i)* denotes the index set of the 1-ring neighborhood of the vertex v_i, i.e., the indices of all neighbors connected to v_i by an edge. The mass d_i is associated to the vertex *i* and the w_{ij} are symmetric edge weights. In the following subsections, several discretizations of Laplace-Beltrami operator will be discussed in detail.

6.4.1 Pinkall Discretization

Pinkall and Polthier used a constant mass in the discretization of the Laplace-Beltrami operator to compute discrete minimal surfaces [141]. The author defined the weight as follows,

$$w_{ij} = \frac{\cot(\alpha_{ij}) + cot(\beta_{ij})}{2}, \tag{26}$$

where α_{ij} and β_{ij} denote the two angles opposite to the edge (i, j) as shown in the Fig. 62.

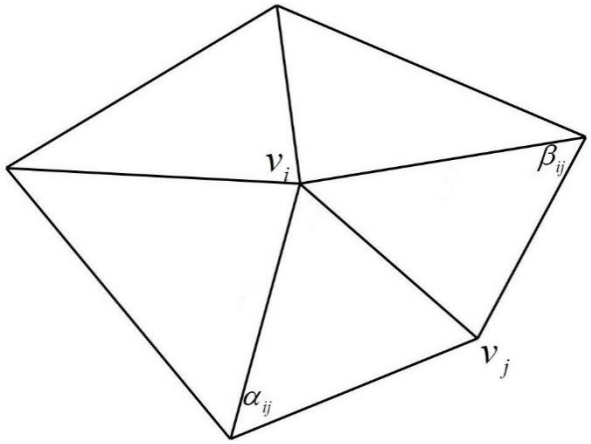


Fig. 62. The angles αij and βij.

6.4.2 Meyer Discretization

A different geometric discretization was suggested by Meyer et al. [142], for triangular meshes. Their approach utilized the voronoi area. If P, Q, and R with circumcenter O is a non-obtuse triangle, as shown in Fig. 63, a+b+c=π/2 can be obtained from the properties of perpendicular bisectors. So, we can write, a= π/2 -∠ Q and c= π/2 -∠ R. The Voronoi area for point P can be computed as below:

$$\frac{1}{8}(|PR|^2 cot\angle Q + |PQ|^2 cot\angle R). \tag{27}$$

Meyer et al. used the areas for the whole 1-ring neighborhood to compute the Voronoi area of the vertex v_i as follows:

$$A_{Voronoi} = \frac{1}{8}\sum_{j\epsilon N(i)}(\cot\alpha_{ij} + \cot\beta_{ij})\left\|v_i - v_j\right\|^2. \tag{28}$$

After computing the area of the 1-ring neighborhood, the weight is updated as follows:

$$w_{ij} = \frac{1}{2A_i}\left(\cot\alpha_{ij} + \cot\beta_{ij}\right), i \neq j, j \in N(i), \tag{29}$$

where *N(i)* is the 1-ring neighborhood of v_i, and A_i is the Voronoi area of the vertex v_i. The author cautioned that this expression for the Voronoi finite volume area does not hold in the presence of obtuse angles [142].

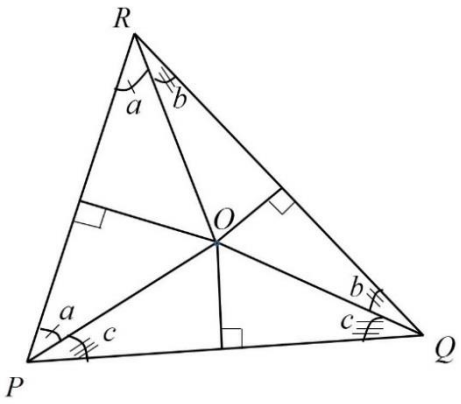


Fig. 63. 1-ring neighborhood and angles opposite to an edge.

Experiments showed that the numerical quality of this operator is equivalent to the finite difference operators for regular sampling [142]. The Voronoi regions of each sample point minimize the bound on the error (created by local averaging of the mean curvature normal) due to spatial averaging since they contain the closest points to each sample [144]. That is why the numerical estimates acquired through this is optimized and require few extra computations. This approach degrades gracefully if irregularity in the mesh is increased. This process is implemented in two steps. First, the Voronoi area of each vertex is calculated. This Voronoi area is summed up for the whole 1-ring neighborhood of the vertex. Second, the weight is updated using this Voronoi area and the cotangent of two opposite angles of an edge.

6.4.3 Mayer Discretization

Mayer et al. developed a method to compute the Laplacian of a function defined on a triangulated surface [143]. The spatial discretization triangulates the surface and approximates any function, which is defined on the surface by its values of the vertex. For a function f on surface S, Green's formula can be written as,

$$\int_{D_\epsilon(z)} \Delta f(x)dx = \int_{\partial D_\epsilon(z)} \partial_n f(s)ds, \tag{30}$$

where $D_\epsilon(z)$ is a small disk at a point *z* on the surface *S* and *n* is the intrinsic outer normal of the boundary of the small disk and it is tangential to the surface. Mayer [143] discretized and replaced the disk of the triangulated surface by the 1-ring neighborhood of the vertex v_i and provided the following approximation:

$$\Delta f(v_i) = \frac{1}{A(v_i)} \sum_{j \in N(i)} \frac{d_{ij}+d_{i,j+1}}{2} \cdot \frac{f(v_j)-f(v_i)}{\|v_i - v_j\|}, \tag{31}$$

where $A(v_i)$ is the sum of areas of triangles around the vertex v_i, and for two consecutive vertices, v_j and $v_{(j+1)}$ on the 1-ring neighborhood of v_i, d_{ij} and $d_{i,j+1}$ are the distances between them respectively. We can write this approximation in the following way:

$$\Delta f(v_i) = \frac{1}{A(v_i)} \sum_{j \in N(i)} \frac{\|v_{a-} v_j\|+\|v_{b-} v_j\|}{2\|v_i - v_j\|} \Big(f\big(v_j\big) - f(v_i)\Big), \tag{32}$$

where $A(v_i)$ is the sum of areas of triangles around vertex v_i, and $a,b \in N_i \cap N_j$.

Equation (27) is derived from equation (26) by approximating $\int_{D_{\epsilon(z)}} \Delta f(x)dx$, $\partial_n f(s)$ and ds with $\Delta f(v_i)A(v_i)$, $\frac{f(v_j)-f(v_i)}{\|v_i - v_j\|}$ and $\frac{\|v_a - v_j\|+\|v_b - v_j\|}{2}$, respectively. This algorithm is used to calculate the area of triangles around each vertex and discretized Green's formula of Riemannian manifold. Here, the distance between a vertex and its neighbor vertex is also taken into account.

6.4.4 Desbrun Discretization

Desbrun et al. [144] found the (area normalized) cotangent Laplacian by computing the area gradient explicitly in the discrete setting. They used two approaches for the Laplace-Beltrami Operator. The first approach uses the cotangent formula of each opposite angles and the sum of the cotangent of every angle and the second approach computes the gradient of 1-ring neighborhood area and use the sum of the areas of each triangle. Eq28 is the normalized version of the computed weight. Desbrun defined the weight as:

$$w_i = \frac{\cot \alpha_i + cot\, \beta_i}{\sum_{j \epsilon N(i)} (cot\, \alpha_j + cot\, \beta_j)}. \tag{33}$$

This algorithm is based on very basic, uniform approximations of the Laplacian. The second approach of Desbrun et al. used the gradient of the 1-ring area with respect to its center vertex. For a non-obtuse triangle, Desbrun also considered 1-ring neighboring vertices of the vertex v_i as shown in Fig. 64. Area A is computed for a small region of a point p. Then the sum of the small areas of the triangles around v_i is computed and denoted as $A(v_i)$. The overall approximation can be computed by as follows:

$$\Delta f(v_i) = \frac{3}{A(v_i)} \sum_{j \in N(i)} \frac{\cot \alpha_j + cot\, \beta_j}{2} \left| f(v_j) - f(v_i) \right|, \tag{34}$$

where $N(i)$ is the index set of the 1-ring neighboring vertices of vertex v_i. α_j and β_j are the angles opposite to an edge as shown in the Fig. 63. This discretization achieves a good sharpening effect with respect to the shape of the geometry, as no drift happens, and only geometric properties are used.

6.5 Mesh Color Sharpening using the Laplace-Beltrami Operator

The previous sections discussed several discretizations of the Laplace-Beltrami operator, including Pinkall, Meyer, Mayer, and two methods proposed by Desbrun. Developed for other

types of applications, none of these discretizations has been utilized for color sharpening on irregular surface meshes. The color of each vertex of a 3D mesh contains three components: red, green, and blue, and each component can be considered as a function defined on the mesh surface. In this work, each color component is treated and processed separately. The Laplace-Beltrami operator is calculated for each color component of a vertex, and then that color component is updated by adding its Laplace-Beltrami operator weighted by a factor to its original value. This operation is repeated for all color components of all vertices.

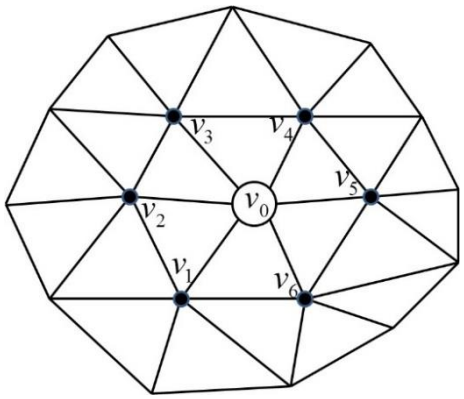


Fig. 64. A vertex and its 1-ring neighborhood in a mesh.

The 1-ring neighborhood of a vertex is used to calculate the Laplace-Beltrami operator in all discretizations. Fig. 64 shows the 1-ring neighborhood of a vertex. If vertex v_0 has a different color than its surrounding vertices, the method searches for the 1-ring neighborhood of each vertex and computes its associated weights and updates each color component of the vertex using the Laplace-Beltrami operator. These weights vary with different discretizations, thus having a different impact on the meshes. As discussed above, Pinkall and Desbrun discretizations compute cotangent of angles, Mayer discretization utilizes the sum of areas of the triangles around vertices, whereas Meyer discretization uses the Voronoi area to calculate weight factors.

The overall system structure of the proposed mesh color sharpening method is illustrated in Fig. 65. The 3D object is first scanned using a 3D scanner such as Microsoft Kinect. The input raw data are usually in the form of a point cloud from which a 3D mesh is then generated that typically consists of triangles. The 1-ring neighborhood is built for all vertices to facilitate fast processing and computing. These two steps establish the topology of the mesh. The mesh color-sharpening phase processes all vertices in the 1-ring neighborhood of the center vertex. First, the Laplace-Beltrami Operator (LBO) of a neighboring vertex is computed. Then the change caused by that neighboring vertex is computed. Finally, the center vertex color is updated. This process is repeated for all three color components (RGB) of all vertices of the 3D mesh.

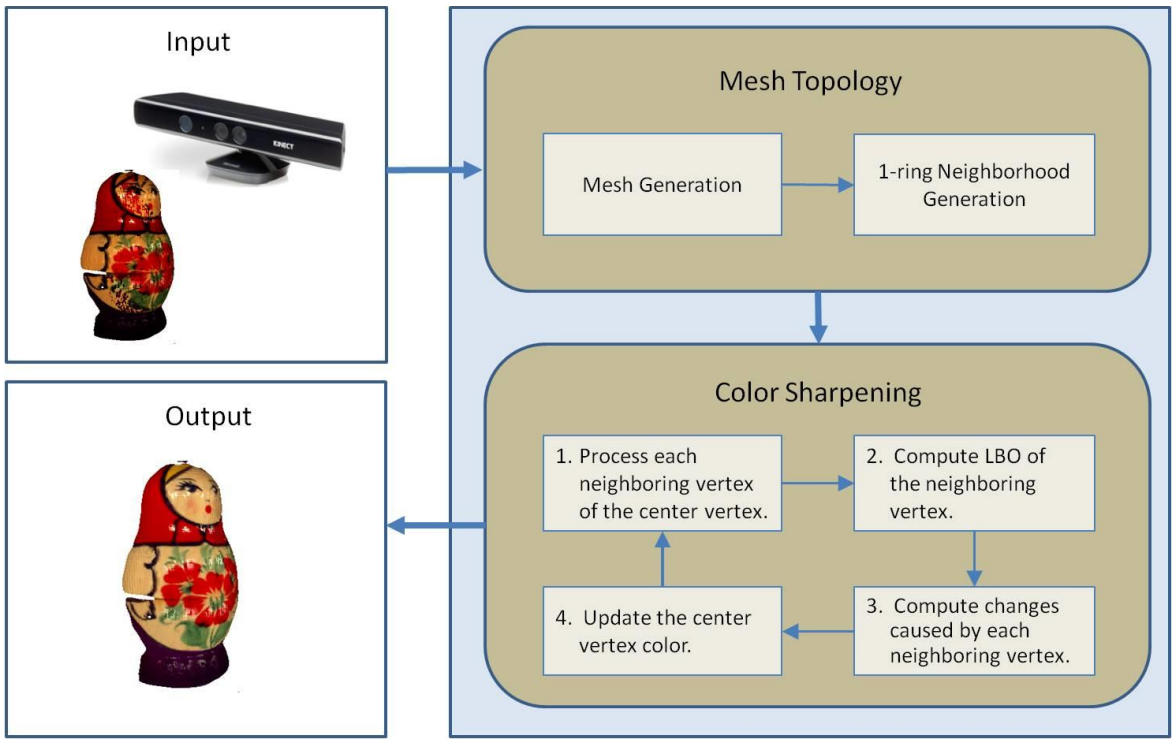


Fig. 65. System architecture of the proposed mesh color sharpening methods.

The various mesh color sharpening methods were implemented in MeshLab [147], an open source 3D mesh processing software package. MeshLab is connected by a central skeleton framework and a large set of independent plugins. This plug-in-based architecture can be used to implement new functionalities. Meshlab also has some components that use the core data structure and basic algorithms provided by the VCG Library. VCG Library is a portable C++ template library to implement algorithms for simplical complexes. All proposed mesh color sharpening methods were implemented as a plug-in C++ based on these data structures and template library. Several experiments were conducted to assess the performance of different implementations of the Laplace-Beltrami operator for mesh color sharpening using a wide range of 3D models. The Coca-Cola can shown in Fig. 66 is a textured model. Fig. 66(b) is the result of conversion from texture to vertex color. After the conversion, the color is distorted. The model in Fig. 66(b) was used as an input to the mesh color sharpening. After applying the Laplace-Beltrami operator for mesh color sharpening, Pinkall, Meyer, Mayer, and Desbrun-1 have similar performances. Meyer discretization performed better than Pinkall, Mayer, and Desbrun-1, as it recovered the Coca-Cola label better than these methods. Desbrun-2 achieved the best performance, and it produced a result that looks even better than the original model (Fig. 66(a)).

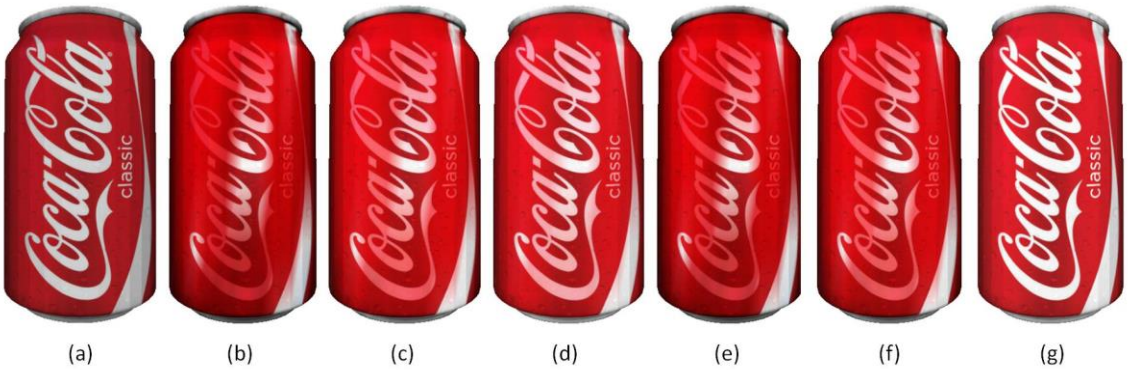


Fig. 66. An artificial textured model (a) Original, (b) Color converted from textured to vertex, (c) Pinkall, (d) Meyer, (e) Mayer, (f) Desbrun-1, (g) Desbrun-2.

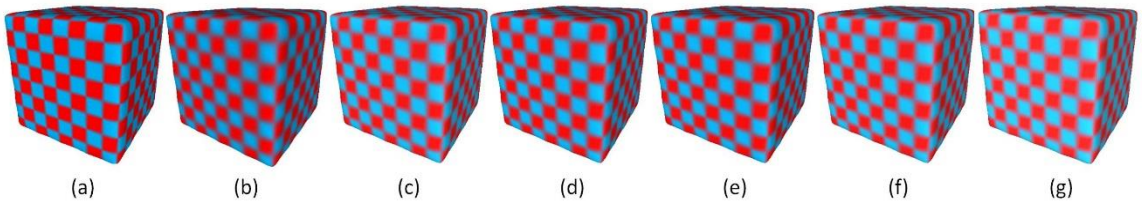


Fig. 67. An artificial 3D model generated using Maya (a) Original, (b) Blurred, (c) Pinkall, (d) Meyer, (e) Mayer, (f) Desbrun-1, (g) Desbrun-2.

To further demonstrate the performance of the proposed mesh color sharpening algorithms, an artificial model cube (Fig. 67(a)) was generated with the 3D modeling software Maya. Each face of the cube has a checkered pattern of red and blue squares. The cube was then blurred in MeshLab to generate the artificial input model shown in Fig. 67(b), which was then processed by different mesh color sharpening algorithms. Fig. 67(b) was generated by replacing the color of each vertex by the average of its neighbors' colors. The Laplace- Beltrami operator using five different discretizations was then applied to the fuzzy mesh, Fig. 67(b), to improve the visual appearance of the mesh color and the results are shown in Fig. 67(c), (d), (e), (f), and (g)

respectively. It is clear that all five discretizations improved the sharpness of the mesh color to different extents. The edges in the cube model become blurry and thicker after the smoothing operation as shown in Fig. 67(b). All of the color-sharpening methods sharpen the cube, but Meyer (Fig. 67(d)) and Desbrun-2 (Fig. 67(g)) performed better than other three discretizations.

Fig. 68. Mesh color sharpening with different implementations of the Laplace-Beltrami operator. The 1st column shows the input models to mesh color sharpening, and remaining columns show the mesh color sharpening results with Pinkall, Meyer, Mayer, Desbrun-1, and Desbrun-2 discretizations (in this order).

Fig. 68 shows more results using different models, including a cube with butterfly texture, a reindeer, and a doll. The first row is a cube with a butterfly texture generated using Google SketchUp. The second row is a scanned model of a reindeer toy in a low light

environment using Microsoft Kinect. The last row is a toy generated using the scanner under natural lighting condition. The original cube with the butterfly texture is dark and lacks details. After applying mesh color sharpening with different discretizations of the Laplace-Beltrami operator, the quality of the cube is improved significantly.

In particular, mesh color sharpening using the Desbrun-1 discretization of the Laplace-Beltrami operator produced stunning results (Fig. 68, 1st row, 5^{th} column). It appears more vivid with visible veins on the leaf and crisp patterns on the butterfly wing. The background appears to have more depth as well. The reindeer model was captured under the low light condition, and it appears very dark, fuzzy, and dull. All mesh color processing methods improved the visual appearance of the reindeer model significantly with more sharpness. The wrinkles of the cloth on the reindeer are clearly visible, as are the eyes of the reindeer and the scarf it wears.

The color variations around the reindeer's nose are more prominent. The face and bottom of the girl doll model are severely contaminated with undesired red and black spots and patches. All mesh color sharpening method using different discretizations of the Laplace-Beltrami operator improve the quality of mesh to varied extents. The result produced by the Meyer discretization (Fig. 68, 3rd row, 3rd column) is almost perfect. It removed completely the undesired red contamination on the face and black contamination on the bottom while enhancing details overall. The Pinkall discretization (Fig. 68, 3rd row, 2nd column) also produced good results with minor red color contamination on the face remaining. Other discretizations were not able to remove the color contaminations completely, but still produced results better than the contaminated input model. It is worth to emphasize that all the mesh color sharpening methods proposed in the paper have been applied to the 3D meshes, not 2D images.

Fig. 69 shows some of the objects and their corresponding mesh structures. Based on the experimental results presented in the paper, mesh color sharpening using different discretizations of the Laplace-Beltrami operator had varied performances on different 3D objects, while overall Desbrun-2 discretization achieved good performance on all models. All the mesh color-sharpening methods have been implemented and incorporated into the open source software Meshlab. It should be kept in mind that the computational cost is also critical for practical applications. TABLE VIII shows the computation time of mesh color sharpening using different discretizations of the Laplace-Beltrami operator on different 3D models. The butterfly, reindeer, and doll have 24, 30338, 29103 vertices and 12, 44850, 56554 faces, respectively.

Fig. 69. 3D objects and their corresponding meshes. (a) and (d): Coca-Cola can. (b) and (e): Girl doll. (c) and (f): Reindeer.

TABLE VIII

COMPUTATION TIME

Methods	Coca-Cola Can	Cube	Butterfly	Reindeer	Doll
Pinkall	125	2463	55	2621	2902
Meyer	171	4341	46	4151	5070
Mayer	156	4136	62	3947	4806
Desbrun-1	156	4141	63	3588	4383
Desbrun-2	171	4056	47	3963	4851

This work proposed a novel method for sharpening mesh colors using different discretizations of the Laplace-Beltrami operator and applied it to color 3D meshes. The Laplace-Beltrami operator is a second-order derivative operator defined for functions on surfaces. This work implemented mesh color sharpening using different discretizations of the Laplace-Beltrami operator, including Pinkall, Meyer, Mayer, Desbrun-1, and Desbrun-2 discretizations and applied to various kinds of 3D models. All mesh color sharpening methods improved the visual appearance of the 3D models. Different discretizations of the Laplace-Beltrami operator had varied performances of mesh color sharpening on different meshes, while the Desbrun-2 discretization achieved good performance on all 3D models in the experiments. Future research is needed to investigate the relationship between the mesh color sharpening performance and mesh structures, such as triangle density and shape. It is also worth noting that mesh color sharpening methods discussed in this part of the dissertation only changed the visual appearance (vertex colors) of the meshes, not the geometrical shapes.

The next chapter gives a conclusive remark of this dissertation.

CHAPTER 7

CONCLUSIONS

The summary and future work of the dissertation are presented in this chapter.

7.1 Summary

This dissertation made several contributions towards point cloud filtering and 3D mesh processing. The first contribution of this dissertation was to develop methods for point cloud processing based on order statistic and adaptive filters, including vector median, fuzzy vector median, adaptive mean, and adaptive median, which were originally developed for image processing. A new filter, namely adaptive vector median, was proposed for point cloud filtering. In the second contribution, a parallel processing method has been implemented using Microsoft Parallel Pattern Library to reduce the computational time of the adaptive vector median filter. This method has been extended in the third contribution for the Aerial LIDAR data filtering, and the fourth contribution proposed a novel method for sharpening mesh colors using different discretizations of the Laplace–Beltrami operator.

Removing noises while improving and sharpening the important features of the data is a challenging task. In most of the cases, the sharp features of the point cloud are occluded or hampered by the outliers and noisy points presented in the point cloud.

The proposed filters not only effectively remove most noise, but also preserve critical features such as edges and corners in a reasonable time. For some cases, the median filter and vector median filter cannot distinguish between fine details and noise and will likely enhance the noise pattern. The noise reduction capabilities of the proposed methods have been compared with several other filtering methods. Experimental results demonstrated a significant amount of

improvement in terms of performance both quantitatively and qualitatively. Although the algorithm could extract most of the outliers in the scene, there were still some that were not detected due to the similarity between the pattern of the data and noise. Another variation was also proposed where normal of the points were taken into account. However, it requires a large amount of time to process the point cloud with moderate performance improvement.

To mitigate this problem, a parallel approach has been adopted using Microsoft Parallel Pattern Library. This technique utilizes the multicore functionality that is now common on desktop and mobile computing devices. The presented results using the Parallel Pattern Library showed a significant gain in computational time. The work demonstrated that the algorithm using PPL scaled very well and achieved significant speed-ups.

A ground-filtering algorithm was proposed for aerial LIDAR data for both rural and urban areas with the differentiated terrain. The method based on AVM was successful in both visual and quantitative ways, achieved comparatively better Kappa and highest in ranking according to the total error rate using the fifteen-reference data from ISPRS compared to five other renowned methods. AVM demonstrated five lowest error rates and one highest error rates for Type I error whereas for Type II error AVM ranked third among five top, well-known methods. AVM generated 2.35%, 2.75%, 4.32%, 0.9%, 1.41%, 2.18% total error type for different areas of the reference data. The average Kappa coefficient is 81.71 which is close to some top filtering algorithm.

A novel method was presented for sharpening mesh colors using different discretizations of the Laplace–Beltrami operator and applied it to color 3D meshes. The Laplace–Beltrami operator is a second-order derivative operator defined for functions on surfaces. This work implemented mesh color sharpening using different discretizations of the Laplace–Beltrami

operator, including Pinkall, Meyer, Mayer, Desbrun-1, and Desbrun-2 discretizations and applied to various kinds of 3D models. All mesh color sharpening methods improved the visual appearance of the 3D models. Different discretizations of the Laplace–Beltrami operator had varied performances of mesh color sharpening on different meshes, while the Desbrun-2 discretization achieved good performance on all 3D models in the experiments. It is also worth noting that mesh color sharpening methods discussed in this dissertation only changed the visual appearance (vertex colors) of the meshes, not the geometrical shapes. A variety of experimental results on synthetic and raw point scans demonstrated that the proposed methods were capable of producing quality results, where sharp features and fine details are recovered well, in the presence of a reasonably high level of noise, outliers, and sparsity.

7.2 Future Work

As future work, the plan is to extend the concept of AVM to other features as well such as using magnitude instead of depth value or normal, different channels of color for point cloud with color information, etc. For the parallel processing, GPU implementation is supposed to speed up the filtering process significantly. Implementation of different deep learning models such as Convolutional Neural Network and Deep Belief Networks for the LIDAR ground filtering would also be of great interest.

REFERENCES

[1] Fuji Technical Research Inc. (August 29). Available: http://www.ftr.co.jp/n/eng/products/galaxy_eye/cad.html. Available: http://www.ftr.co.jp/n/eng/products/galaxy_eye/cad.html.

[2] M. J. Milroy, D. J. Weir, C. Bradley, and G. W. Vickers, "Reverse engineering employing a 3D laser scanner: A case study," *International Journal of Advanced Manufacturing Technology,* vol. 12, no. 2, pp. 111-121, 1996.

[3] M. Levoy, K. Pulli, S. Rusinkiewicz, M. Ginzton, J. Ginsberg, D. Koller, S. Anderson, J. Shade, B. Curless, L. Pereira, J. Davis, and D. Fulk., "The digital Michelangelo project: 3D scanning of large statues.," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques,* 2000, pp. 131-144.

[4] S. F. El-Hakim, J. Beraldin, M. Picard, and G. Godin., "Detailed 3D reconstruction of large-scale heritage sites with integrated techniques.," *IEEE Computer Graphics and Applications,* vol. 24, no. 3, pp. 21-29, 2004.

[5] J. C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R Evans, "Reconstruction and representation of 3D objects with radial basis functions," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001, pp. 67-76.

[6] C. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3d scans," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, 1995, pp. 109-118.

[7] S. Cheng and M. Lau, "Denoising a point cloud for surface reconstruction,," *CoRR,* vol. abs/1704.04038., no. arXiv:1704.04038.

[8] F. Zaman, Y. P. Wong, and B. Y. Ng, "Density-based Denoising of Point Cloud," *CoRR,* vol. abs/1602.05312, 2016.

[9] X. F. Han, J. S. Jin, M. J. Wang, and W. Jiang, "Iterative guidance normal filter for point cloud," *Multimedia Tools and Applications*.

[10] W. Hou, T. Chan, and M. Ding, "Denoising point cloud," *Inverse Problems in Science and Engineering*, vol. 20, no. 3, pp. 287-298, 2012.

[11] L. Wang, B. Yuan, and J. Chen, "Robust Fuzzy C-Means and Bilateral Point Clouds Denoising," presented at the 8th international Conference on Signal Processing 2006, 2006.

[12] S. Sotoodeh, "Outlier detection in laser scanner point clouds," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences,* vol. 36, no. 5, pp. 297-302, 2006.

[13] Y. Song, "Boundary fitting for 2D curve reconstruction," *The Visual Computer,* vol. 26, no. 3, pp. 187-204, 2010.

[14] B. Mederos, L. Velho, and L. H. d. Figueiredo, "Robust smoothing of noisy point clouds," in *Siam Conference on Geometric Design and Computing*, 2003.

[15] C. Lea, "Near Real-time Pointcloud Smoothing for 3D Imaging Systems," 2011.

[16] K. Liu and R. Zayer, "Bundle Adjustment Constrained Smoothing for Multi-view Point Cloud Data," presented at the 8th International Symposium on Visual Computing, 2012.

[17] Y. Wang and H.-Y. Feng, "Outlier detection for scanned point clouds using majority voting," *Computer-Aided Design,* vol. 62, pp. 31-43, 2015.

[18] Z. Yang and D. Xiao, "A systemic point-cloud de-noising and smoothing method for 3D shape reuse," presented at the 2012 12th International Conference on Control Automation Robotics & Vision (ICARCV), 2012.

[19] B. Q. Shi, J. Liang, and Q. Liu, "Adaptive simplification of point cloud using k-means clustering," *Computer-Aided Design,* vol. 43, no. 8, pp. 910-922, 2011.

[20] S. Xu, Z. Yang, and W. Wu, "Algorithm of denoising based on point cloud segmentation," presented at the 5th International Conference on Computer Science and Education (ICCSE), 2010.

[21] X. An, X. Yu, Q. Xu, and J. Wang, "Research on 3D scanning point cloud de-nosing," presented at the 2014 International Conference on Audio, Language and Image Processing (ICALIP), 2014.

[22] Y. P. Lin and K. W. Hsu, "Dealing with Noisy Data on Point Cloud Models," presented at the 2014 IEEE International Symposium on Multimedia (ISM), 2014.

[23] M. Wu, J. Wang, and H. Luo, "Research on 3D Modeling of Point cloud data Based on Terrestrial Laser Scanner," presented at the 3rd International Conference on Mechatronics, Robotics and Automation (ICMRA 2015), 2015.

[24] O. Schall, A. Belyaev, and H.P. Seidel, "Robust Filtering of Noisy Scattered Point Data," in *IEEE VGTC Symposium Point-Based Graphics Proceedings Eurographics*, 2005, pp. 71-144.

[25] H. Han, X. Han, F. Sun, and C. Huang, "Point cloud simplification with preserved edge based on normalvector," *Optik - International Journal for Light and Electron Optics,* vol. 126, no. 19, pp. 2157-2162, 2015.

[26] G. Rosman, A. Dubrovina, and R. Kimmel, "Patch-Collaborative Spectral Point-Cloud Denoising," *Computer Graphics forum,* vol. 32(2013), no. 8, pp. 1-12, 2013.

[27] J. E. Deschaud and F. Goulette, "A Fast and Accurate Plane Detection Algorithm for Large Noisy Point Clouds Using Filtered Normals and Voxel Growing," presented at the 3DPVT, 2014.

[28] J. E. Deschaud and F. Goulette, "Point cloud non local denoising using local surface descriptor similarity," 2010.

[29] J. Digne and C. d. Franchis, "The Bilateral Filter for Point Clouds," *Image Processing On Line,* vol. 7, pp. 278-287, 2017.

[30] Z. Huang, Y. Huang, and J. Huang, "A Method for Noise Removal of LIDAR Point Clouds," presented at the 2013 Third International Conference on Intelligent System Design and Engineering Applications (ISDEA), 2013.

[31] T. Weber, R. Hänsch, and O. Hellwich, "Automatic registration of unordered point clouds acquired by Kinect sensors using an overlap heuristic," vol. 102, pp. 96-109, 2015.

[32] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien, "Spectral Surface Reconstruction from Noisy Point Clouds," presented at the Symposium on Geometry Processing, 2004.

[33] N. Salman, M. Yvinec, and Q. Mérigot, "Feature Preserving Mesh Generation from 3D Point Clouds," presented at the Computer Graphics Forum, 2010.

[34] Y. Zheng, G. Li, S. Wu, Y. Liu, and Y. Gao, "Guided point cloud denoising via sharp feature skeletons," *The Visual Computer,* vol. 33, no. 6, 2017.

[35] A. Javaheri, C. Brites, F. Pereira, and J. Ascenso, "Subjective and objective quality evaluation of 3D point cloud denoising algorithms,," presented at the IEEE International Conference on Multimedia ExpoWorkshops (ICMEW), 2017.

[36] X. F. Han, J. S. Jin, M. J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3D point cloud,," *Signal Processing: Image Communication* vol. 57, pp. 103-112, 2017.

[37] Q. Yang, P. Ji, D. Li, S. Yao, and M. Zhang, "Fast stereo matching using adaptive guided filtering," *Image and Vision Computing,* vol. 32, no. 3, pp. 202-211, 2014/03/01/ 2014.

[38] N. Aitali, B. Cherradi, A. E. Abbasi, and O. Bouattane, "Parallel Implementation of Bias Field Correction Fuzzy C-Means Algorithm for Image Segmentation," *International Journal of Advanced Computer Science and Applications,* vol. 7, no. 3, 2016.

[39] S. Galliani, K. Lasinger, and K. Schindler, "Massively Parallel Multiview Stereopsis by Surface Normal Diffusion," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 873-881.

[40] D. T. Anderson, R. H. Luke, and J. M. Keller, "Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units," *IEEE Transactions on Fuzzy Systems,* vol. 16, no. 4, pp. 1101-1106, 2008.

[41] H. Li, Z. Yang, and H. He, "An Improved Image Segmentation Algorithm Based on GPU Parallel Computing," *Journal of Software,* vol. 9, no. 8, August 2014 2014.

[42] K. Tan, J. Zhang, Q. Du, and X. Wang, "GPU Parallel Implementation of Support Vector Machines for Hyperspectral Image Classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing,* vol. 8, no. 10, pp. 4647-4656, 2015.

[43] Z. Wu *et al.*, "GPU Parallel Implementation of Spatially Adaptive Hyperspectral Image Classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing,* vol. 11, no. 4, pp. 1131-1143, 2018.

[44] A. I. El-Nashar, "To Parallelize or not to parallelize, speedup issue," *International Journal of Distributed and Parallel Systems (IJDPS),* vol. 2, no. 2, 2011.

[45] J. L. Martínez, A. J. Reina, J. Morales, A. Mandow, and A. J. García-Cerezo, "Using multicore processors to parallelize 3D point cloud registration with the Coarse Binary Cubes method," in *2013 IEEE International Conference on Mechatronics (ICM)*, 2013, pp. 335-340.

[46] D. Qiu, S. May, and A. N¨uchter, "GPU-Accelerated Nearest Neighbor Search for 3D Registration*". Computer Vision Systems, ICVS 2009. Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2009.

[47] X. Hu and Y. Yuan, "Deep-Learning-Based Classification for DTM Extraction from ALS Point Cloud," *Remote Sensing,* vol. 8, no. 9, p. 730, 2016.

[48] G. Sithole and G. Vosselman, "Experimental comparison of filter algorithms for bare-Earth extraction from airborne laser scanning point clouds," *The International journal of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* vol. 59, no. 1/2, pp. 85–101, 2004.

[49] D. Mongus, N. Lukač, and B. Žalik, "Ground and building extraction from LiDAR data based on differential morphological profiles and locally fitted surfaces," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 93, pp. 145-156, 2014/07/01/ 2014.

[50] C. Chen, Y. Li, N. Zhao, J. Guo, and G. Liu, "A fast and robust interpolation filter for airborne lidar point clouds," *PLOS ONE,* vol. 12, no. 5, p. e0176954, 2017.

[51] H. Yang, W. Chen, T. Qian, D. Shen, and J. Wang, "The Extraction of Vegetation Points from LiDAR Using 3D Fractal Dimension Analyses," *Remote Sensing,* vol. 7, no. 8, p. 10815, 2015.

[52] Y. Li, B. Yong, P. van Oosterom, M. Lemmens, H. Wu, L. Ren, M. Zheng, and J. Zhou, "Airborne LiDAR Data Filtering Based on Geodesic Transformations of Mathematical Morphology," *Remote Sensing,* vol. 9, no. 11, p. 1104, 2017.

[53] X. Hu, X. Li, and Y. Zhang, "Fast Filtering of LiDAR Point Cloud in Urban Areas Based on Scan Line Segmentation and GPU Acceleration," *IEEE Geoscience and Remote Sensing Letters,* vol. 10, no. 2, pp. 308-312, 2013.

[54] W. Zhang, J. Qi, P. Wan, H. Wang, D. Xie, X. Wang, and G. Yan, "An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation," *Remote Sensing,* vol. 8, no. 6, p. 501, 2016.

[55] T. J. Pingel, K. C. Clarke, and W. A. McBride, "An improved simple morphological filter for the terrain classification of airborne LIDAR data," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 77, pp. 21-30, 2013/03/01/ 2013.

[56] P. E. Axelsson, "DEM generation from laser scanner data using adaptive TIN models. ," *International Archives of the Photogrammetry and Remote Sensing,* vol. 33, pp. 110-117, 2000.

[57] K. Kraus and N. Pfeifer, "Determination of terrain models in wooded areas with airborne laser scanner data. ," *ISPRS Journal of Photogrammetry and Remote Sensing* , vol. 53, no. 4, pp. 193-203, 1998.

[58] C. Chen, Y. Li, W. Li, and H. Dai, "A multiresolution hierarchical classification algorithm for filtering airborne LiDAR data," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 82, pp. 1-9, 2013/08/01/ 2013.

[59] Z. Keqi, C. Shu-Ching, D. Whitman, S. Mei-Ling, Y. Jianhua, and Z. Chengcui, "A progressive morphological filter for removing nonground measurements from airborne LIDAR data," *IEEE Transactions on Geoscience and Remote Sensing,* vol. 41, no. 4, pp. 872-882, 2003.

[60] X. Meng, N. Currit, and K. Zhao, "Ground Filtering Algorithms for Airborne LiDAR Data: A Review of Critical Issues," *Remote Sensing,* vol. 2, no. 3, p. 833, 2010.

[61] X. Meng, L. Wang, J. L. Silván-Cárdenas, and N. Currit, "A multi-directional ground filtering algorithm for airborne LIDAR," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 64, no. 1, pp. 117-124, 2009/01/01/ 2009.

[62] J. Kilian, N. Haala, and M. Englich, *Capture Andevaluation of Airborne Laser Scanner Data* (International Archives of Photogrammetry and Remote Sensing). 1996.

[63] Q. Chen, P. Gong, D. Baldocchi, and G. Xie, "Filtering Airborne Laser Scanning Data with Morphological Methods," *Photogrammetric Engineering & Remote Sensing,* vol. 2, pp. 175-185, 2007.

[64] C. A. Silva, C. Klauberg, Â. M. K. Hentz, A. P. D. Corte, U. Ribeiro, and V. Liesenberg, "Comparing the Performance of Ground Filtering Algorithms for Terrain Modeling in a Forest Environment Using Airborne LiDAR Data," *Floresta e Ambiente,* vol. 25, 2018.

[65] G. Vosselman, "Slope Based Filtering of Laser Altimetry Data," *IAPRS,* vol. 33, 2000.

[66] J. Shan and A. Sampath, "Urban DEM generation from raw lidar data," *Photogrammetric Engineering & Remote Sensing,* vol. 71, pp. 217-226, 2005.

[67] S. Filin, "Surface clustering from airborne laser scanning data," *International Archives of Photogrammetry and Remote Sensing,* vol. XXXII, no. 3A, pp. 119-124, 2002.

[68] M. Roggero, "Airborne Laser Scanning: Clustering in raw data," *IAPRS,* vol. XXXIV, pp. 227-232, 2001.

[69] I. Lee and T. Schenk, "3D Perceptual Organization of Laser Altimetry Data," in *IAPRS,* 2001, vol. XXXIV, pp. 57-65.

[70] S. A. Hosseini, H. Arefi, and Z. Gharib, "Filtering of Lidar Point Cloud Using a Strip Based Algorithm in Residential Mountainous Areas", 2014.

[71] Y. Liu and R. Zhong, "Buildings and Terrain of Urban Area Point Cloud Segmentation based on PCL," in *IOP Conf. Series: Earth and Environmental Science,* 2014, vol. 17.

[72] G. Sithole, *Filtering of Laser Altimetry Data using a Slope Adaptive Filter*. 2011.

[73] Q. Zhan, Y. Liang, Y. Cai, and Y. Xiao, "Pattern detection in airborne LiDAR data using Laplacian of Gaussian filter," *Geo-spatial Information Science,* journal article vol. 14, no. 3, p. 184, July 30 2011.

[74] M. Elmqvist, "Ground Estimation of Lasar Radar Data using Active Shape Models," Stockholm, Sweden, 2001.

[75] M. Elmqvist, E. Jungert, F. Lantz, Å. Persson, and U. Söderman, "Terrain modelling and analysis using laser scanner data," in *International Archives of Photogrammetry and Remote,* 2001, pp. 22-24.

[76] H. Hamraza, N. B. Jacobsa, M. A. Contrerasb, and C. H. Clarkb, "Deep learning for conifer/deciduous classification of airborne LiDAR 3D point clouds representing individual trees," *Arxiv*, 2018.

[77] D. Marmanis, A. Fathalrahman, M. Datcu, T. Esch, and U. Stilla, *Deep Neural Networks For Above-ground Detection in Very High Spatial Resolution Digital Elevation Models*. 2015.

[78] S. I. Oh and H. B. Kang, "Object Detection and Classification by Decision-Level Fusion for Intelligent Vehicle Systems," *Sensors (Basel, Switzerland)*, vol. 17, no. 1, p. 207, 2017.

[79] Z. Ao, Y. Su, W. Li, Q. Guo, and J. Zhang, "One-Class Classification of Airborne LiDAR Data in Urban Areas Using a Presence and Background Learning Algorithm," *Remote Sensing*, vol. 9, no. 10, p. 1001, 2017.

[80] X. Li, X. Cheng, W. Chen, G. Chen, and S. Liu, "Identification of Forested Landslides Using LiDar Data, Object-based Image Analysis, and Machine Learning Algorithms," *Remote Sensing*, vol. 7, no. 8, p. 9705, 2015.

[81] S. K. Lodha, E. J. Kreps, D. P. Helmbold, and D. Fitzpatrick, "Aerial LiDAR Data Classification Using Support Vector Machines (SVM)," in *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, 2006, pp. 567-574.

[82] F. Samadzadegan, B. Bigdeli and Pouria Ramzi, "Classification of Lidar Data Based on Multi-class Svm", 2010.

[83] C. Bellman and M. Shortis, "Building recognition using wavelet analysis and support vector machines," *Spatial Knowledge without Boundaries*, 2003.

[84] L. Guo, N. Chehata, C. Mallet, and S. Boukir, "Relevance of airborne lidar and multispectral image data for urban scene classification using Random Forests," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 1, pp. 56-66, 2011/01/01/ 2011.

[85] H. Hu, Y. Ding, Q. Zhu, B. Wu, H. Lin, Z. Du, Y. Zhang, and Y. Zhang, "An adaptive surface filter for airborne laser scanning point clouds by means of regularization and bending energy," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 92, pp. 98-111, 2014/06/01/ 2014.

[86] A. Victoriano, J. Brasington, M. Guinau, G. Furdada, M. Cabré, and M. Moysset, "Geomorphic impact and assessment of flexible barriers using multi-temporal LiDAR data: The Portainé mountain catchment (Pyrenees)." *Engineering Geology*, 2018.

[87] J. A. Stevenson, X. Sun, and N. C. Mitchell, "Despeckling SRTM and other topographic data with a denoising algorithm," *Geomorphology,* vol. 114, no. 3, pp. 238-252, 2010/01/15/ 2010.

[88] Y. Quan, J. Song, X. Guo, Q. Miao, and Y. Yang, "Filtering LiDAR data based on adjacent triangle of triangulated irregular network," *Multimedia Tools and Applications,* journal article vol. 76, no. 8, pp. 11051-11063, April 01 2017.

[89] A. Nurunnabi, G. West, and D. Belton, "Outlier detection and robust normal-curvature estimation in mobile laser scanning 3D point cloud data," *Pattern Recognition,* vol. 48, no. 4, pp. 1404-1419, 2015/04/01/ 2015.

[90] A. A. Matkan, M. Hajeb, B. Mirbagheri, S. Sadeghian, and M. Ahmadi, "Spatial Analysis for Outlier Removal from LIDAR Data," presented at the The 1st ISPRS International Conference on Geospatial Information Research, 2014.

[91] A. Abdullah and Z. Vojinovic, "Lidar Filtering Algorithms for Urban Flood Application: Review on Current Algorithms and Filters Test," 2009.

[92] W. T. Tinkham *et al.*, "A Comparison of Two Open Source LiDAR Surface Classification Algorithms," *Remote Sensing,* vol. 3, no. 3, p. 638, 2011.

[93] J. Zhongping, L. Ligang, and W. Guojin, "A global Laplacian smoothing approach with feature preservation," in *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, 2005, p. 6.

[94] G. Taubin, "A signal processing approach to fair surface design," presented at the Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995.

[95] J. Vollmer, R. Mencl, and H. Müller, "Improved Laplacian Smoothing of Noisy Surface Meshes," *Computer Graphics Forum,* vol. 18, no. 3, pp. 131-138, 1999.

[96] B. Hamann, S. E. Dillard, M. Hlawitschka, and S. Shafii, "The topological effects of smoothing," *IEEE Transactions on Visualization & Computer Graphics* vol. 18, pp. 160–72, 2011.

[97] C. C. L. Wang, "Bilateral recovering of sharp edges on feature-insensitive sampled meshes," *IEEE Transactions on Visualization and Computer Graphics,* vol. 12, no. 4, pp. 629-639, 2006.

[98] R. Fan and X. Jin, "Controllable Edge Feature Sharpening for Dental Applications," *Computational and Mathematical Methods in Medicine,* vol. 2014, p. 9, 2014, Art. no. 873635.

[99] M. Attene, B. Falcidieno, J. Rossignac, and M. Spagnuolo, "Edge-sharpener: recovering sharp features in triangulations of non-adaptively re-meshed surfaces," presented at the Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing, Aachen, Germany, 2003.

[100] Y. Ohtake, A. G. Belyaev, and I. A. Bogaevski, "Polyhedral Surface Smoothing with Simultaneous Mesh Regularization," presented at the Proceedings of the Geometric Modeling and Processing 2000, 2000.

[101] Y. Shen and K. E. Barner, "Fuzzy vector median-based surface smoothing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 3, pp. 252-265, 2004.

[102] U. Clarenz, U. Diewald, and M. Rumpf, "Anisotropic geometric diffusion in surface processing," in *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*, 2000, pp. 397-405.

[103] K. Hildebrandt and K. Polthier, "Constraint-based fairing of surface meshes," presented at the Proceedings of the fifth Eurographics symposium on Geometry processing, Barcelona, Spain, 2007.

[104] H. Zhao and G. Xu, "Triangular surface mesh fairing via Gaussian curvature flow," *Journal of Computational and Applied Mathematics*, vol. 195, no. 1–2, pp. 300-311, 10/15/ 2006.

[105] J. Shen, S. Zhang, Z. Chen, Y. Zhang, and X. Ye, "Mesh sharpening via normal filtering," *Journal of Zhejiang University-SCIENCE A*, journal article vol. 10, no. 4, pp. 546-553, 2009.

[106] S. Ganesan and L. Tobiska, "Computations of flows with interfaces using arbitrary Lagrangian Eulerian method," in *Proceedings of the European Conference on Computational Fluid Dynamics, ECCOMAS CFD 2006.*, Egmond aan Zee, The Netherlands, 2006.

[107] C. H. Leung and M. Berzins, "A computational model for organism growth based on surface mesh generation," *Journal of Computational Physics*, vol. 188, no. 1, pp. 75-99, 6/10/ 2003.

[108] F. Petronetto, A. Paiva, E. S. Helou, D. E. Stewart, and L. G. Nonato, "Mesh-Free Discrete Laplace–Beltrami Operator," *Computer Graphics Forum*, vol. 32, no. 6, pp. 214-226, 2013.

[109] M. Belkin, J. Sun, and Y. Wang, "Discrete laplace operator on meshed surfaces," presented at the Proceedings of the twenty-fourth annual symposium on Computational geometry, College Park, MD, USA, 2008.

[110] M. Desbrun, M. Meyer, P. Schrŏder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," presented at the Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999.

[111] X. D. Gu, R. Guo, F. Luo, and W. Zeng, "Discrete Laplace–Beltrami operator determines discrete Riemannian metric," *CoRR 2010*.

[112] W. JY, C. MH, and C. SG, "Convergent discrete Laplace–Beltrami operators over surfaces," *CoRR 2010*, 2010.

[113] G. Xu, "Discrete Laplace–Beltrami operator on sphere and optimal spherical triangulations," *International Journal on Computational Geometry and Applications 2006*, vol. 16, pp. 75-93, 2006.

[114] Y. Xiong, G. Li, and G. Han, "Mean Laplace–Beltrami Operator for Quadrilateral Meshes," in *Transactions on Edutainment V*, Z. Pan, A. D. Cheok, W. Müller, and X. Yang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 189-201.

[115] A. Wetzler, Y. Aflalo, A. Dubrovina, and R. Kimmel, "The Laplace-Beltrami Operator: A Ubiquitous Tool for Image and Shape Processing," in *Mathematical Morphology and Its Applications to Signal and Image Processing: 11th International Symposium, ISMM 2013, Uppsala, Sweden, May 27-29, 2013. Proceedings*, C. L. L. Hendriks, G. Borgefors, and R. Strand, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 302-316.

[116] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke, "Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D," *IEEE Robotics & Automation Magazine*, vol. 22, no. 4, pp. 110-124, 2015.

[117] *Leica Geosystems HDS* (08 October 2018). Available: https://leica-geosystems.com/en-US/products/laser-scanners.

[118] *Leica Geosystems Total Stations* (10 October 2018). Available: https://leica-geosystems.com/en-US/products/total-stations.

[119] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," *Künstliche Intelligenz*, vol. 24, pp. 345-348, 2009.

[120] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," Shanghai, China, 2011.

[121] J. Astola, P. Haavisto, and Y. Neuvo, "Vector median filters," *Proceedings of the IEEE*, vol. 78, no. 4, pp. 678-689, 1990.

[122] M. Barni, F. Buti, F. Bartolini, and V. Cappellini, "A quasi-Euclidean norm to speed up vector median filtering," *IEEE Transactions on Image Processing*, vol. 9, no. 10, pp. 1704-1709, 2000.

[123] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1992.

[124] Point Cloud Library (October 2018). Available: http://www.pointclouds.org.

[125] Stanford Computer Graphics Laboratory (October 21 2018). Available: http://www.graphics.stanford.edu/data/3Dscanrep/

[126] J. Berkmann and T. Caelli, "Computation of surface geometry and segmentation using covariance techniques," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 16, no. 11, pp. 1114-1116, 1994.

[127] Parallel Patterns Library (PPL) (October 02, 2018.) Available: https://docs.microsoft.com/en-us/previous-versions/dd492418(v=vs.140).

[128] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," presented at the Proceedings of the April 18-20, 1967, Spring joint computer conference, Atlantic City, New Jersey, 1967.

[129] G. Sohn and I. Dowman, "Data fusion of high-resolution satellite imagery and LiDAR data for automatic building extraction," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 62, no. 1, pp. 43-63, 2007/05/01/ 2007.

[130] M. A. Brovelli, M. Cannata, and U. M. Longoni,, "LIDAR Data Filtering and DTM Interpolation Within GRASS," *Transactions in GIS,* vol. 8, no. 2, pp. 155-174, 2004.

[131] R. A. Haugerud and D. J. Harding, "Some algorithms for virtual deforestation (VDF) of LIDAR topographic survey data," *IAPRS,* vol. XXXIV -3/W4, pp. 211-218., 22-24 October 2001 2001.

[132] X. Meng, N. Currit and K. Zhao, "Ground Filtering Algorithms For Airborne Lidar Data," *Remote Sens*. 2010, vol. 2 no. 3, pp. 833-860.

[133] J. L. Silván-Cárdenas and L. Wang, "A multi-resolution approach for filtering LiDAR altimetry data," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 61, no. 1, pp. 11-22, 2006/10/01/ 2006.

[134] K. Zhang and D. Whitman, "Comparison of Three Algorithms for Filtering Airborne Lidar Data," *Photogrammetric Engineering & Remote Sensing,* vol. 71, no. 3, pp. 313-324, // 2005.

[135] P. Axelsson, "Processing of laser scanner data—algorithms and applications," *ISPRS Journal of Photogrammetry and Remote Sensing,* vol. 54, no. 2, pp. 138-147, 1999/07/01/ 1999.

[136] J. Shan, "Urban DEM Generation from Raw Lidar Data : A Labeling Algorithm and its Performance," 2005.

[137] X. Meng, L. Wang, and N. Currit, "Morphology-based Building Detection from Airborne Lidar Data," *Photogrammetric Engineering & Remote Sensing*, vol. 75, no. 4, pp. 437-442, // 2009.

[138] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37-46, 1960.

[139] ReconstructMe. (12.08.15). *ReconstructMe:RealTime3DScanningSoftware*. Available: http://reconstructme.net/;

[140] Z. Afrose and Y. Shen, "Mesh color sharpening," *Advances in Engineering Software*, vol. 91, pp. 36-43, 2016/01/01/ 2016.

[141] U. Pinkall and K. Polthier, "Computing discrete minimal surfaces and their conjugates," (in en), pp. 15-36, 1993 1993.

[142] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds," in *Visualization and Mathematics III*, H.-C. Hege and K. Polthier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 35-57.

[143] U. F. Mayer, "Numerical solutions for the surface diffusion flow in three space dimensions," *Computational & Applied Mathematics 2001*, vol. 20, pp. 361–79, 2001.

[144] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Discrete Differential-Geometry Operators in nD," vol. 20, pp. 35-57, 2000.

[145] C. I, *Eigenvalues in Riemannian Geometry*. Orlando, FL, USA: Academic Press 1984.

[146] M. Reuter, S. Biasotti, D. Giorgi, G. Patanè, and M. Spagnuolo, "Discrete Laplace–Beltrami operators for shape analysis and segmentation," *Computers & Graphics*, vol. 33, no. 3, pp. 381-390, 6// 2009.

[147] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "Meshlab: an open-source mesh processing tool," in *Proceedings of the Eurographics Italian Chapter Conference*, 2008, pp. 129–36.

VITA

Zinat Afrose

Department of Modeling, Simulation and Visualization Engineering

Old Dominion University

Norfolk, VA 23529

Education:

- Ph.D.: January 2013 – present, Old Dominion University, Norfolk, VA, USA
- M.Sc.: January 2012. Computer Science and Engineering, Jahangirnagar University, Dhaka, Bangladesh.
- B.Sc.: May 2009. Computer Science and Engineering, Jahangirnagar University, Dhaka, Bangladesh.

List of Publications:

1. Z. Afrose, Y. Shen, "Parallelization of adaptive vector median filter for point cloud denoising", *Student Capstone Conference* 2018.
2. Z. Afrose, Y. Shen, "Point cloud denoising using adaptive and order statistic filters", *Student Capstone Conference* 2017.
3. Z. Afrose, Y. Shen, "Mesh color sharpening", *Journal Advances in Engineering Software*, Volume 91 Issue C, January 2016, Pages 36-43, Elsevier Science Ltd. Oxford, UK.
4. Z. Afrose, Y. Shen, "Vector Order Statistic-Based Noise Removal from Point Cloud Data", *Student Capstone Conference* 2016.
5. Z. Afrose, Y. Shen, "Repairing Color Mesh Models", *Student Capstone Conference* 2015.

6. Z. Afrose, Y. Shen, “Applying Discrete Laplace-Beltrami Operator to Mesh Color Sharpening”, *Student Capstone Conference* 2014.
7. Z. Afrose, Y. Shen, “Mesh color sharpening using Laplace-Beltrami operator”, *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*, Atlanta, GA, 2014, pp. 1029-1033. doi: 10.1109/GlobalSIP.2014.7032277.
8. Z. Afrose, “A Comparative Study on Noise Removal of Compound Images using Different Types of Filters”, *IJCA International Journal of Computer Applications (USA),* vol 47,no.14, pp.45-48, June 2012.
9. Z. Afrose, M. A. Bhuiyan, “Road Sign Segmentation and Recognition under Bad Illumination Condition using Modified Fuzzy C-means Clustering”, *IJCA International Journal of Computer Applications(USA),* vol. 50, no.8, July 2012 , pp.1- 6, July 2012.
10. Z. Afrose, “Relaxed Median Filter: A Better Noise Removal Filter for Compound Images”, *IJCSE International Journal on Computer Science and Engineering (India),* vol. 4, no. 07, pp.1376- 1382, July 2012.